Бесплатная электронная книга

УЧУСЬ

R Language

Free unaffiliated eBook created from
Stack Overflow contributors.

#r

Около

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: r-language

It is an unofficial and free R Language ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official R Language.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

глава 1: Начало работы с R Language

замечания

Редактирование R Docs при переполнении стека

При создании документации ознакомьтесь с документацией по общим правилам.

Несколько особенностей R, что иммигранты с другого языка могут найти необычные

- В отличие от других языков переменные в R не требуют объявления типа.
- Одной и той же переменной могут быть назначены разные типы данных в разные моменты времени, если это необходимо.
- Индексирование атомных векторов и списков начинается с 1, а не 0.
- R- `arrays` (и частный случай матриц) имеют атрибут `dim` который отличает их от «атомных векторов» R, которые не имеют атрибутов.
- Список в R позволяет вам собирать различные объекты под одним именем (то есть имя списка) упорядоченным способом. Эти объекты могут быть **матрицами** , **векторами** , **кадрами данных** , **даже другими списками** и т. Д. Даже не требуется, чтобы эти объекты были связаны друг с другом каким-либо образом.
- Переработка отходов
- Отсутствующие значения

Examples

Установка R

Возможно, вы захотите установить RStudio после того, как вы установили R. RStudio - это среда разработки для R, которая упрощает многие задачи программирования.

Только для Windows:

Visual Studio (начиная с версии 2015 Update 3) теперь имеет среду разработки для R,

называемую R Tools , которая включает в себя живой интерпретатор, IntelliSense и модуль отладки. Если вы выберете этот метод, вам не нужно будет устанавливать R, как указано в следующем разделе.

Для Windows

1. Перейдите на сайт CRAN , нажмите на загрузку R для Windows и загрузите последнюю версию R.
2. Щелкните правой кнопкой мыши файл установщика и запустите его как администратор.
3. Выберите язык для установки.
4. Следуйте инструкциям по установке.

Для OSX / macOS

Альтернатива 1

(0. Убедитесь, что XQuartz установлен)

1. Перейдите на сайт CRAN и загрузите последнюю версию R.
2. Откройте образ диска и запустите программу установки.
3. Следуйте инструкциям по установке.

Это установит как R, так и R-MacGUI. Он поместит GUI в / Applications / Folder как R.app, где его можно либо дважды щелкнуть, либо перетащить в Doc. Когда новая версия будет выпущена, процесс (re) -installation перезапишет R.app, но будет сохранен предыдущий вариант основных версий R. Фактический код R будет находиться в каталоге / Library/Frameworks/R.Framework/Versions/. Использование R в RStudio также возможно и будет использовать один и тот же R-код с другим графическим интерфейсом.

Альтернатива 2

1. Установите homebrew (отсутствующий менеджер пакетов для macOS), следуя инструкциям на https://brew.sh/
2. `brew install R`

Те, кто выбирает второй метод, должны знать, что сторонник вилки Mac советует против него и не будет отвечать на вопросы о трудностях в списке рассылки R-SIG-Mac.

Для Debian, Ubuntu и производных

Вы можете получить версию R, соответствующую вашему дистрибутиву, с помощью `apt-get` . Однако эта версия часто будет значительно отставать от самой последней версии, доступной на CRAN. Вы можете добавить CRAN в свой список признанных «источников».

```
sudo apt-get install r-base
```

Вы можете получить более новую версию непосредственно из CRAN, добавив CRAN в список источников. Следуйте указаниям от CRAN для получения более подробной информации. Обратите внимание, в частности, на необходимость также выполнить это, чтобы вы могли использовать `install.packages()` . Пакеты Linux обычно распространяются как исходные файлы и нуждаются в компиляции:

```
sudo apt-get install r-base-dev
```

Для Red Hat и Fedora

```
sudo dnf install R
```

Для Archlinux

R напрямую доступен в `Extra` package.

```
sudo pacman -S r
```

Более подробную информацию об использовании R под Archlinux можно найти на странице ArchWiki R.

Привет, мир!

```
"Hello World!"
```

Кроме того, проверьте подробное обсуждение того, как, когда и зачем печатать строку .

Получать помощь

Вы можете использовать функцию `help()` или `?` для доступа к документам и поиска помощи в R. Для более общих поисков вы можете использовать `help.search()` или `??` ,

```
#For help on the help function of R
help()

#For help on the paste function
```

```
help(paste)     #OR
help("paste")   #OR
?paste          #OR
?"paste"
```

Посетите https://www.r-project.org/help.html для получения дополнительной информации.

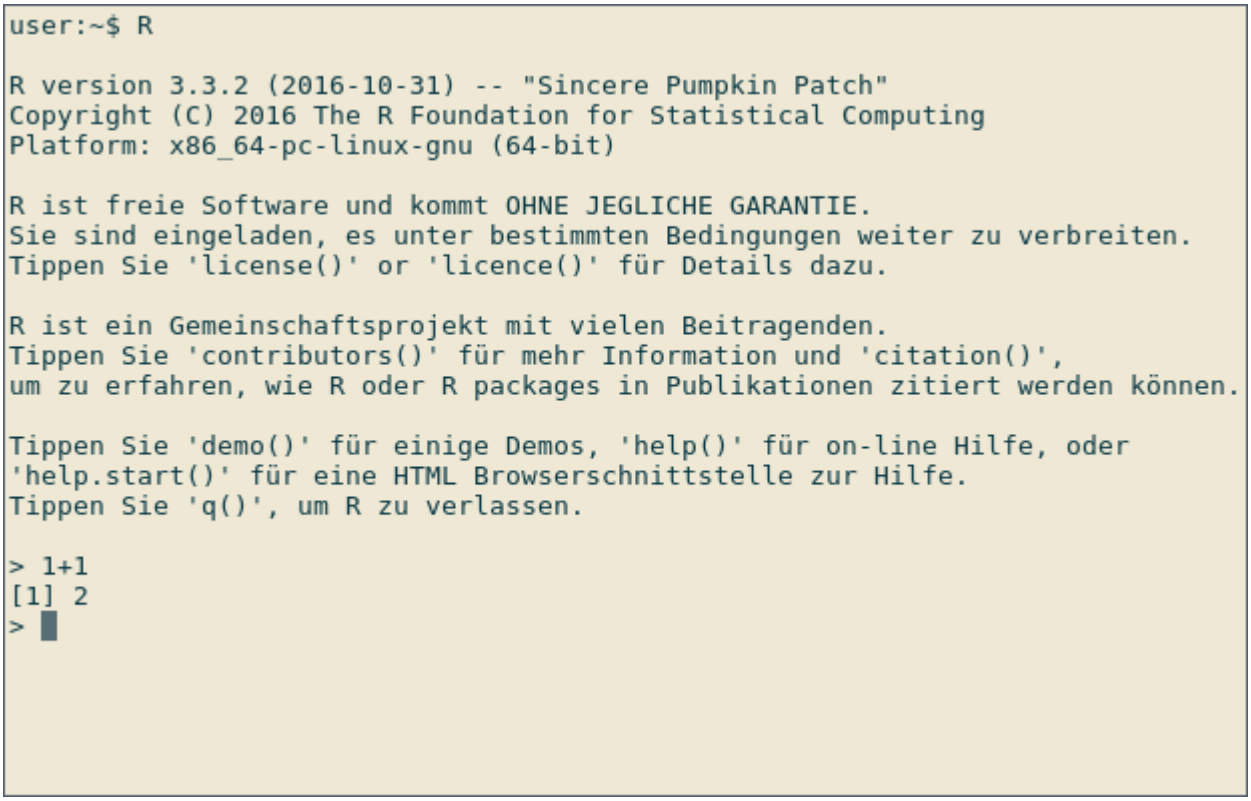
Интерактивный режим и сценарии R

Интерактивный режим

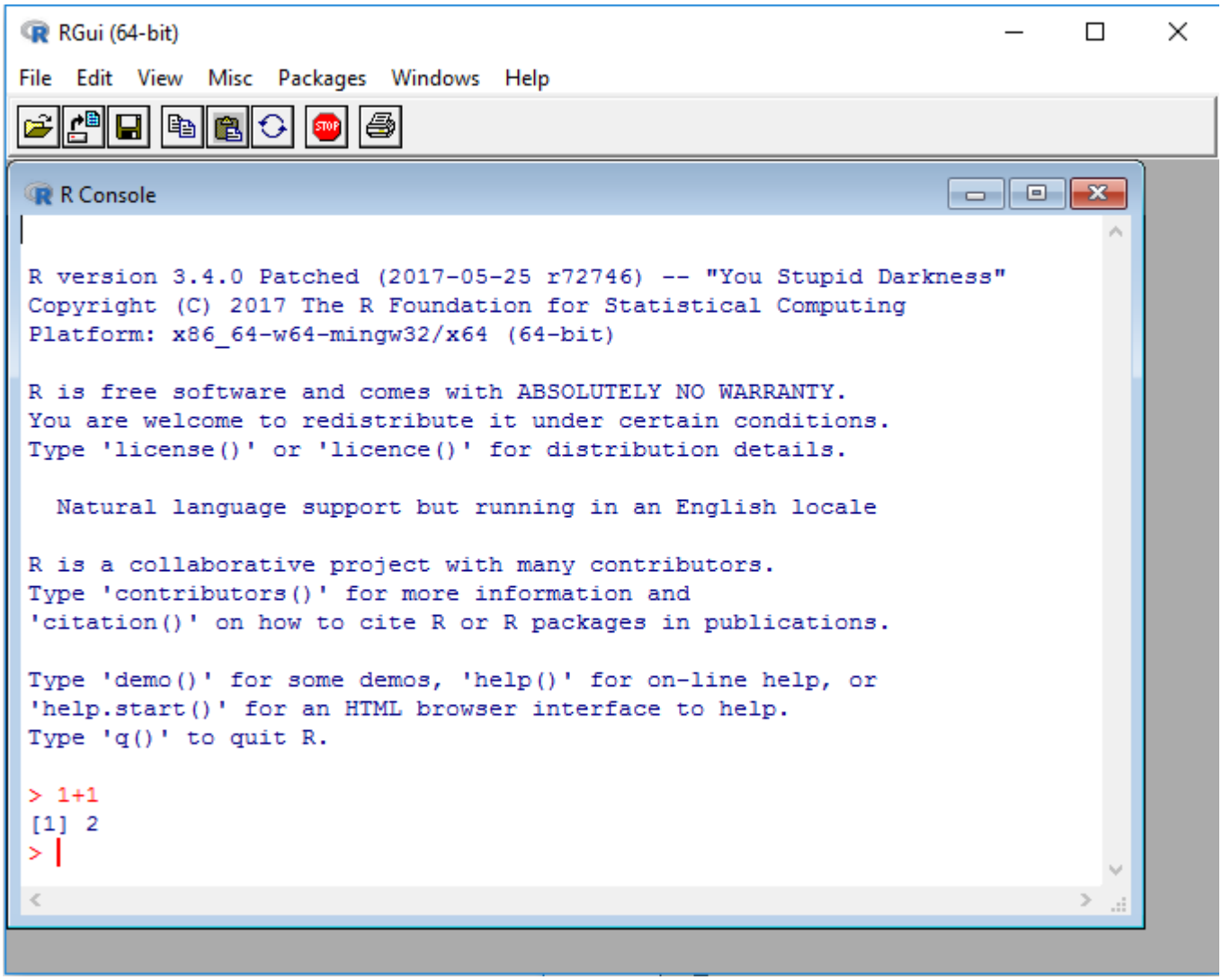
Самый простой способ использования R - *интерактивный* режим. Вы вводите команды и сразу получаете результат от R.

Использование R в качестве калькулятора

Запустите R, набрав `R` в командной строке вашей операционной системы или выполнив `RGui` в Windows. Ниже вы можете увидеть скриншот интерактивной сессии R в Linux:



Это RGui в Windows, самая основная рабочая среда для R под Windows:



После знака `>` можно ввести выражения. После того, как выражение напечатано, результат показан R. На скриншоте выше R используется в качестве калькулятора: Тип

```
1+1
```

чтобы сразу увидеть результат, `2` . Ведущий `[1]` указывает, что R возвращает вектор. В этом случае вектор содержит только одно число (2).

Первый сюжет

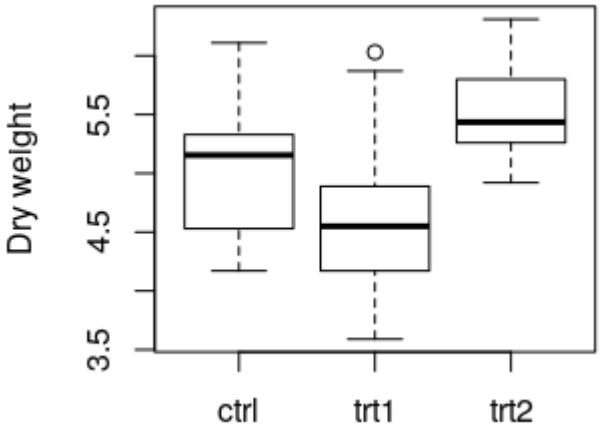
R может использоваться для генерации графиков. В следующем примере используется набор данных `PlantGrowth` , который поставляется в качестве примера набора данных вместе с R

Введите int следующие строки в запрос R, которые не начинаются с `##` . Строки, начинающиеся с `##` , предназначены для документирования результата, который будет возвращен R.

```
data(PlantGrowth)
```

```
str(PlantGrowth)
## 'data.frame':    30 obs. of  2 variables:
## $ weight: num  4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
## $ group : Factor w/ 3 levels "ctrl","trt1",..: 1 1 1 1 1 1 1 1 1 1 ...
anova(lm(weight ~ group, data = PlantGrowth))
## Analysis of Variance Table
##
## Response: weight
##           Df  Sum Sq Mean Sq F value  Pr(>F)
## group      2  3.7663  1.8832  4.8461 0.01591 *
## Residuals 27 10.4921  0.3886
## ---
## Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")
```

Создается следующий график:



`data(PlantGrowth)` загружают примерный набор данных `PlantGrowth` , который представляет собой данные о сухих массах растений, которые подвергались двум различным условиям лечения или вообще не подвергались лечению (контрольная группа). Набор данных доступен под названием `PlantGrowth` . Такое имя также называется переменной .

Для загрузки собственных данных могут оказаться полезными следующие две страницы документации:

- Чтение и запись табличных данных в текстовых файлах (CSV, TSV и т. Д.)
- I / O для внешних таблиц (Excel, SAS, SPSS, Stata)

`str(PlantGrowth)` показывает информацию о загруженном наборе данных. Вывод указывает, что `PlantGrowth` - это `data.frame` , который является именем R для таблицы. `data.frame` содержит два столбца и 30 строк. В этом случае каждая строка соответствует одному растению. Подробности двух столбцов показаны в строках, начинающихся с `$` : первый столбец называется `weight` и содержит числа (`num` , сухой вес соответствующего растения). Вторая колонка, `group` , содержит обработку, на которую растение подвергалось. Это категориальные данные, которые называются `factor` R. Прочитайте больше информации о кадрах данных .

Чтобы сравнить сухие массы трех разных групп, односторонний ANOVA выполняется с

использованием `anova(lm( ... ))` . `weight ~ group` означает «Сравнить значение столбца `weight` , группировку по значениям столбца `group` ». Это называется формулой в R. `data = ...` указывает имя таблицы, в которой данные могут быть найдены.

Результат показывает, среди прочего, что существует значительная разница (Column `Pr(>F)`), `p = 0.01591`) между некоторыми из трех групп. Последующие тесты, такие как Tukey's Test, должны выполняться для определения того, какие средства групп значительно различаются.

`boxplot(...)` создает `boxplot(...)` график данных. где значения, которые должны быть построены, исходят из. `weight ~ group` означает: «Заговор значения веса столба по *сравнению* с значениями столбца `group` . `ylab = ...` указует метку оси у Дополнительной информации:. Базовые заговоры

Введите `q()` или `Ctrl-D`, чтобы выйти из сеанса R.

R-скрипты

Чтобы документировать ваши исследования, удобно сохранять команды, которые вы используете для расчета в файле. Для этого вы можете создавать **R-скрипты** . Сценарий R - это простой текстовый файл, содержащий команды R.

Создайте текстовый файл с именем `plants.R` и заполните его следующим текстом, где некоторые команды знакомы с приведенным выше кодом кода:

```
data(PlantGrowth)

anova(lm(weight ~ group, data = PlantGrowth))

png("plant_boxplot.png", width = 400, height = 300)
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")
dev.off()
```

Выполните скрипт, введя свой терминал (терминал вашей операционной системы, а **не** интерактивный сеанс R, как в предыдущем разделе!)

```
R --no-save <plant.R >plant_result.txt
```

Файл `plant_result.txt` содержит результаты ваших вычислений, как если бы вы ввели их в интерактивную подсказку R. Таким образом, ваши расчеты задокументированы.

Новые команды `png` и `dev.off` используются для сохранения boxplot на диск. Две команды должны заключать команду построения графика, как показано в примере выше. `png("FILENAME", width = ..., height = ...)` открывает новый PNG-файл с указанным именем файла, шириной и высотой в пикселях. `dev.off()` завершит `dev.off()` и сохранит график на диск. Выход не сохраняется до `dev.off()` .

Прочитайте Начало работы с R Language онлайн: https://riptutorial.com/ru/r/topic/360/начало-работы-с-r-language

глава 2: * применять семейство функций (функционалов)

замечания

Функция в семействе `*apply` является абстракцией цикла `for` . По сравнению с петлями `for` `*apply` функции приложения имеют следующие преимущества:

1. Требовать меньше кода для записи.
2. Не имеет счетчика итераций.
3. Не использует временные переменные для хранения промежуточных результатов.

Однако `for` циклов более общие и могут дать нам больший контроль, позволяющий достичь сложных вычислений, которые не всегда тривиальны, используя функции `*apply` .

Отношения между `for` петель и `*apply` функции объясняются в документации `for` петель .

Члены `*apply` Family

Семейство функций `*apply` содержит несколько вариантов того же принципа, которые различаются в основном в зависимости от вида возвращаемого ими результата.

функция	вход	Выход
`apply`	`matrix` , `data.frame` или `array`	вектор или матрица (в зависимости от длины каждого возвращаемого элемента)
`sapply`	вектор или `list`	вектор или матрица (в зависимости от длины каждого возвращаемого элемента)
`lapply`	вектор или `list`	`list`
`vapply`	вектор или `список	вектор или матрица (в зависимости от длины каждого возвращаемого элемента) назначенного пользователем класса
`mapply`	множественные векторы, `lists` или комбинация	`list`

См. «Примеры», чтобы узнать, как используется каждая из этих функций.

Examples

Использовать анонимные функции с применением

`apply` используется для оценки функции (возможно, анонимной) над полями массива или матрицы.

Давайте используем набор `iris` для иллюстрации этой идеи. Набор данных `iris` имеет измерения 150 цветов из 3 видов. Посмотрим, как структурирован этот набор данных:

```
> head(iris)

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Теперь представьте, что вы хотите знать среднее значение *каждой* из этих переменных. Одним из способов решения этой проблемы может быть использование цикла `for` , но программисты R часто предпочитают использовать `apply` (по причинам, см. Примечания):

```
> apply(iris[1:4], 2, mean)

Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
```

- В первом параметре подмножество `iris` включает только первые 4 столбца, потому что `mean` работает только с числовыми данными.
- Второе значение параметра `2` означает, что мы хотим работать только с столбцами (второй индекс массива r × c); `1` даст ряд строк.

Точно так же мы можем рассчитать более значимые значения:

```
# standard deviation
apply(iris[1:4], 2, sd)
# variance
apply(iris[1:4], 2, var)
```

Предостережение : R имеет некоторые встроенные функции, которые лучше вычисляют суммы столбцов и строк и означают: `colMeans` И `rowMeans` .

Теперь давайте сделаем другую и более значимую задачу: вычислим среднее значение *только* для тех значений, которые больше `0.5` . Для этого мы создадим нашу собственную `mean` функцию.

```
> our.mean.function <- function(x) { mean(x[x > 0.5]) }
> apply(iris[1:4], 2, our.mean.function)

Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.665347
```

(Обратите внимание на разницу в значении `Petal.Width`)

Но что, если мы не хотим использовать эту функцию в остальной части нашего кода? Затем мы можем использовать анонимную функцию и написать наш код следующим образом:

```
apply(iris[1:4], 2, function(x) { mean(x[x > 0.5]) })
```

Итак, как мы видели, мы можем использовать `apply` для выполнения той же операции над столбцами или строками набора данных, используя только одну строку.

Предостережение . Поскольку `apply` возвращает очень разные виды вывода в зависимости от длины результатов указанной функции, это может быть не лучший выбор в тех случаях, когда вы не работаете в интерактивном режиме. Некоторые из других `*apply` семейные функции, являются немного более предсказуемыми (см. Примечания).

Загрузка массового файла

для большого количества файлов, которые, возможно, необходимо будет использовать в аналогичном процессе и с хорошо структурированными именами файлов.

во-первых, должен быть создан вектор имен файлов для доступа, для этого есть несколько вариантов:

- Создание вектора вручную с помощью `paste0()`

  ```
  files <- paste0("file_", 1:100, ".rds")
  ```

- Использование `list.files()` с термином поиска в регулярном выражении для типа файла требует знания регулярных выражений (regex), если в каталоге находятся другие файлы того же типа.

  ```
  files <- list.files("./", pattern = "\\.rds$", full.names = TRUE)
  ```

где `X` - вектор части используемого формата имен файлов.

`lapply` выводит каждый ответ как элемент списка.

`readRDS` специфичен для файлов `.rds` и будет меняться в зависимости от применения процесса.

```
my_file_list <- lapply(files, readRDS)
```

Это не обязательно быстрее, чем цикл for от тестирования, но позволяет всем файлам быть элементом списка, не назначая их явно.

Наконец, нам часто приходится загружать сразу несколько пакетов. Этот трюк может сделать это довольно легко, применив `library()` ко всем библиотекам, которые мы хотим импортировать:

```
lapply(c("jsonlite","stringr","igraph"),library,character.only=TRUE)
```

Объединение нескольких `data.frames` (` lapply`, `mapply`)

В этом упражнении мы создадим четыре модели линейной регрессии начальной загрузки и объединим резюме этих моделей в единый кадр данных.

```
library(broom)

#* Create the bootstrap data sets
BootData <- lapply(1:4,
                   function(i) mtcars[sample(1:nrow(mtcars),
                                             size = nrow(mtcars),
                                             replace = TRUE), ])

#* Fit the models
Models <- lapply(BootData,
                 function(BD) lm(mpg ~ qsec + wt + factor(am),
                                 data = BD))

#* Tidy the output into a data.frame
Tidied <- lapply(Models,
                 tidy)

#* Give each element in the Tidied list a name
Tidied <- setNames(Tidied, paste0("Boot", seq_along(Tidied)))
```

На этом этапе мы можем использовать два подхода для вставки имен в data.frame.

```
#* Insert the element name into the summary with `lapply`
#* Requires passing the names attribute to `lapply` and referencing `Tidied` within
#* the applied function.
Described_lapply <-
 lapply(names(Tidied),
        function(nm) cbind(nm, Tidied[[nm]]))

Combined_lapply <- do.call("rbind", Described_lapply)

#* Insert the element name into the summary with `mapply`
#* Allows us to pass the names and the elements as separate arguments.
Described_mapply <-
 mapply(
  function(nm, dframe) cbind(nm, dframe),
  names(Tidied),
  Tidied,
```

```
  SIMPLIFY = FALSE)

Combined_mapply <- do.call("rbind", Described_mapply)
```

Если вы являетесь поклонником `magrittr` стиля `magrittr` , вы можете выполнить всю задачу в одной цепочке (хотя это может быть неразумно делать, если вам нужны какие-либо промежуточные объекты, такие как сами объекты модели):

```
library(magrittr)
library(broom)
Combined <- lapply(1:4,
                   function(i) mtcars[sample(1:nrow(mtcars),
                                             size = nrow(mtcars),
                                             replace = TRUE), ]) %>%
 lapply(function(BD) lm( mpg ~ qsec + wt + factor(am), data = BD)) %>%
 lapply(tidy) %>%
 setNames(paste0("Boot", seq_along(.))) %>%
 mapply(function(nm, dframe) cbind(nm, dframe),
        nm = names(.),
        dframe = .,
        SIMPLIFY = FALSE) %>%
 do.call("rbind", .)
```

Использование встроенных функционалов

Встроенные функционалы: lapply (), sapply () и mapply ()

R поставляется со встроенными функционалами, из которых, возможно, наиболее известными являются семейство функций. Ниже приведено описание некоторых из наиболее распространенных функций приложения:

- `lapply()` = принимает список в качестве аргумента и применяет указанную функцию к списку.
- `sapply()` = то же, что и `lapply()` но пытается упростить вывод к вектору или матрице.
    - `vapply()` = вариант `sapply()` в котором должен указываться тип выходного объекта.
- `mapply()` = like `lapply()` но может передавать несколько векторов в качестве входных данных для указанной функции. Может быть упрощено, например, `sapply()` .
    - `Map()` является псевдонимом `mapply()` с `SIMPLIFY = FALSE` .

---

lapply ()

`lapply()` может использоваться с двумя разными итерациями:

- `lapply(variable, FUN)`

- `lapply(seq_along(variable), FUN)`

```
# Two ways of finding the mean of x
set.seed(1)
df <- data.frame(x = rnorm(25), y = rnorm(25))
lapply(df, mean)
lapply(seq_along(df), function(x) mean(df[[x]]))
```

---

sapply ()

`sapply()` попытается разрешить свой вывод либо вектору, либо матрице.

```
# Two examples to show the different outputs of sapply()
sapply(letters, print)  ## produces a vector
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
sapply(x, quantile)  ## produces a matrix
```

---

mapply ()

`mapply()` работает так же, как и `lapply()` за исключением того, что он может принимать несколько векторов в качестве входных данных (следовательно, m для многомерного).

```
mapply(sum, 1:5, 10:6, 3) # 3 will be "recycled" by mapply
```

Использование пользовательских функций

Пользовательские функции

Пользователи могут создавать свои собственные функционалы в различной степени сложности. Следующие примеры взяты из « Функционалов » Хэдли Уикхэма:

```
randomise <- function(f) f(runif(1e3))

lapply2 <- function(x, f, ...) {
    out <- vector("list", length(x))
    for (i in seq_along(x)) {
        out[[i]] <- f(x[[i]], ...)
    }
    out
}
```

В первом случае `randomise` принимает единственный аргумент `f` и вызывает его на выборке из Uniform случайных величин. Чтобы продемонстрировать эквивалентность, назовем `set.seed` ниже:

```
set.seed(123)
randomise(mean)
```

```
#[1] 0.4972778

set.seed(123)
mean(runif(1e3))
#[1] 0.4972778


set.seed(123)
randomise(max)
#[1] 0.9994045

set.seed(123)
max(runif(1e3))
#[1] 0.9994045
```

Второй пример - это повторная реализация `base::lapply` , которая использует функции для применения операции (`f`) к каждому элементу в списке (`x`). Параметр `...` позволяет пользователю передавать дополнительные аргументы в `f` , например, параметр `na.rm` в `mean` функции:

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA

lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA


lapply(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4

lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

Прочитайте * применять семейство функций (функционалов) онлайн: https://riptutorial.com/ru/r/topic/3567/--применять-семейство-функций--функционалов-

глава 3: .Rprofile

замечания

В Efficient R есть хорошая глава по этому вопросу

Examples

.Rprofile - выполняется первый фрагмент кода

`.Rprofile` - это файл, содержащий R-код, который выполняется при запуске R из каталога, содержащего файл `.Rprofile` . Аналогично `Rprofile.site` , расположенный в домашнем каталоге R, выполняется по умолчанию каждый раз, когда вы загружаете R из любого каталога. `Rprofile.site` и в большей степени. `.Rprofile` может использоваться для инициализации сеанса R с личными настройками и различными функциями утилиты, которые вы определили.

> Важное примечание: если вы используете RStudio, у вас может быть отдельный `.Rprofile` в каждом каталоге проектов RStudio.

Вот несколько примеров кода, которые вы можете включить в файл .Rprofile.

Настройка домашнего каталога R

```
# set R_home
Sys.setenv(R_USER="c:/R_home") # just an example directory
# but don't confuse this with the $R_HOME environment variable.
```

Настройка параметров размера страницы

```
options(papersize="a4")
options(editor="notepad")
options(pager="internal")
```

установить тип справки по умолчанию

```
options(help_type="html")
```

установить библиотеку сайтов

```
.Library.site <- file.path(chartr("\\", "/", R.home()), "site-library")
```

Установите зеркало CRAN

```
local({r <- getOption("repos")
    r["CRAN"] <- "http://my.local.cran"
    options(repos=r)})
```

Настройка местоположения вашей библиотеки

Это позволит вам не устанавливать все пакеты снова с каждым обновлением версии R.

```
# library location
.libPaths("c:/R_home/Rpackages/win")
```

Пользовательские ярлыки или функции

Иногда полезно иметь ярлык для длинного выражения R. Общим примером этого параметра является активное связывание для доступа к последнему результату выражения верхнего уровня без необходимости ввода `.Last.value` :

```
makeActiveBinding(".", function(){.Last.value}, .GlobalEnv)
```

Поскольку .Rprofile - это просто файл R, он может содержать любой произвольный R-код.

Предварительная загрузка наиболее полезных пакетов

Это плохая практика и, как правило, следует избегать, поскольку она отделяет код загрузки пакета от сценариев, где эти пакеты фактически используются.

Смотрите также

См. `help(Startup)` для всех сценариев запуска и другие аспекты. В частности, можно загружать два файла `Profile` всей системы. Первый, `Rprofile` , может содержать глобальные настройки, другой файл `Profile.site` может содержать локальные варианты, которые системный администратор может сделать для всех пользователей. Оба файла находятся в `${RHOME}/etc` установки R. Этот каталог также содержит глобальные файлы `Renviron` и `Renviron.site` которые могут быть дополнены локальным файлом `~/.Renviron` в домашнем каталоге пользователя.

.Rprofile пример

Запускать

```
# Load library setwidth on start - to set the width automatically.
.First <- function() {
  library(setwidth)
  # If 256 color terminal - use library colorout.
  if (Sys.getenv("TERM") %in% c("xterm-256color", "screen-256color")) {
    library("colorout")
  }
}
```

Опции

```
# Select default CRAN mirror for package installation.
options(repos=c(CRAN="https://cran.gis-lab.info/"))

# Print maximum 1000 elements.
options(max.print=1000)

# No scientific notation.
options(scipen=10)

# No graphics in menus.
options(menu.graphics=FALSE)

# Auto-completion for package names.
utils::rc.settings(ipck=TRUE)
```

Пользовательские функции

```
# Invisible environment to mask defined functions
.env = new.env()

# Quit R without asking to save.
.env$q <- function (save="no", ...) {
  quit(save=save, ...)
}

# Attach the environment to enable functions.
attach(.env, warn.conflicts=FALSE)
```

Прочитайте .Rprofile онлайн: https://riptutorial.com/ru/r/topic/4166/-rprofile

глава 4: ANOVA

Examples

Основное использование aov ()

Анализ дисперсии (aov) используется для определения того, отличаются ли средства двух или более групп друг от друга. Ответы считаются независимыми друг от друга, обычно распределяются (внутри каждой группы), а внутригрупповые отклонения принимаются равными.

Для завершения анализа данные должны быть в длинном формате (см. Раздел «Изменение темы данных»). `aov()` является оберткой вокруг функции `lm()` , используя формулу формулы Уилкинсона-Роджерса `y~f` где `y` - отклик (независимая) переменная, а `f` - факторная (категориальная) переменная, представляющая членство в группе. *Если `f` является числовой, а не факторной переменной, `aov()` будет сообщать результаты линейной регрессии в формате ANOVA, что может удивить неопытных пользователей.*

Функция `aov()` использует Type I (последовательный) Sum of Squares. Этот тип суммы квадратов проверяет все (основные и взаимодействующие) эффекты последовательно. В результате первым проверенным эффектом также назначается общая дисперсия между ним и другими эффектами в модели. Чтобы результаты такой модели были надежными, данные должны быть сбалансированы (все группы имеют одинаковый размер).

Если предположения для суммы квадратов типа I не выполняются, могут применяться классы типа II или типа III. Тип II Сумма квадратов проверяет каждый основной эффект после каждого другого основного эффекта и, таким образом, контролирует любую перекрывающуюся дисперсию. Однако Type II Sum of Squares не предполагает взаимодействия между основными эффектами.

Наконец, Type III Sum of Squares проверяет каждый основной эффект после каждого другого основного эффекта *и* каждого взаимодействия. Это делает Type III Sum of Squares необходимостью при наличии взаимодействия.

Совокупность квадратов типа II и типа III реализована в функции `Anova()` .

В качестве `mtcars` набор данных `mtcars` .

```
mtCarsAnovaModel <- aov(wt ~ factor(cyl), data=mtcars)
```

Чтобы просмотреть сводку модели ANOVA:

```
summary(mtCarsAnovaModel)
```

Можно также извлечь коэффициенты базовой модели `lm()` :

```
coefficients(mtCarsAnovaModel)
```

Основное использование Anova ()

При работе с неуравновешенным дизайном и / или неортогональными контрастами необходимо указать сумму квадратов типа II или типа III. Функция `Anova()` из пакета `car` реализует их. Тип II Сумма квадратов не предполагает взаимодействия между основными эффектами. Если предполагается взаимодействие, то подходит тип III квадратов квадратов.

Функция `Anova()` обтекает функцию `lm()` .

Используя `mtcars` данных `mtcars` в качестве примера, демонстрируя разницу между типом II и типом III при тестировании взаимодействия.

```
> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 2)
Anova Table (Type II tests)

Response: wt
                      Sum Sq Df F value    Pr(>F)
factor(cyl)           7.2278  2 11.5266 0.0002606 ***
factor(am)            3.2845  1 10.4758 0.0032895 **
factor(cyl):factor(am) 0.0668  2  0.1065 0.8993714
Residuals             8.1517 26
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 3)
Anova Table (Type III tests)

Response: wt
                       Sum Sq Df F value    Pr(>F)
(Intercept)           25.8427  1 82.4254 1.524e-09 ***
factor(cyl)            4.0124  2  6.3988  0.005498 **
factor(am)             1.7389  1  5.5463  0.026346 *
factor(cyl):factor(am) 0.0668  2  0.1065  0.899371
Residuals              8.1517 26
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Прочитайте ANOVA онлайн: https://riptutorial.com/ru/r/topic/3610/anova

глава 5: boxplot

Синтаксис

- boxplot (x, ...) # общая функция
- boxplot (формула, данные = NULL, ..., подмножество, na.action = NULL) ## Метод S3 для формулы класса '
- boxplot (x, ..., range = 1.5, width = NULL, varwidth = FALSE, notch = FALSE, outline = TRUE, имена, plot = TRUE, border = par ("fg"), col = NULL, log = " ", pars = list (boxwex = 0.8, staplewex = 0.5, outwex = 0.5), horizontal = FALSE, add = FALSE, at = NULL) ## Метод по умолчанию S3

параметры

параметры	Подробности (исходная документация R)
формула	формулу, такую как y ~ grp, где y - числовой вектор значений данных, которые должны быть разделены на группы в соответствии с переменной группировки grp (обычно это фактор).
данные	data.frame (или список), из которого должны быть взяты переменные в формуле.
подмножество	необязательный вектор, определяющий подмножество наблюдений, которое будет использоваться для построения графика.
na.action	функция, которая указывает, что должно произойти, когда данные содержат NA. По умолчанию следует игнорировать отсутствующие значения в ответе или группе.
boxwex	масштабный коэффициент, который должен применяться ко всем ящикам. Когда есть только несколько групп, внешний вид сюжета можно улучшить, сделав ящики более узкими.
сюжет	если TRUE (по умолчанию), тогда создается boxplot. Если нет, возвращаются резюме, на которых основаны ящики.
седло	если col не является нулевым, предполагается, что он содержит цвета, которые будут использоваться для окраски тел полей окна. По умолчанию они находятся в фоновом цвете.

Examples

Создайте графику с квадратным ящиком с boxplot () {graphics}

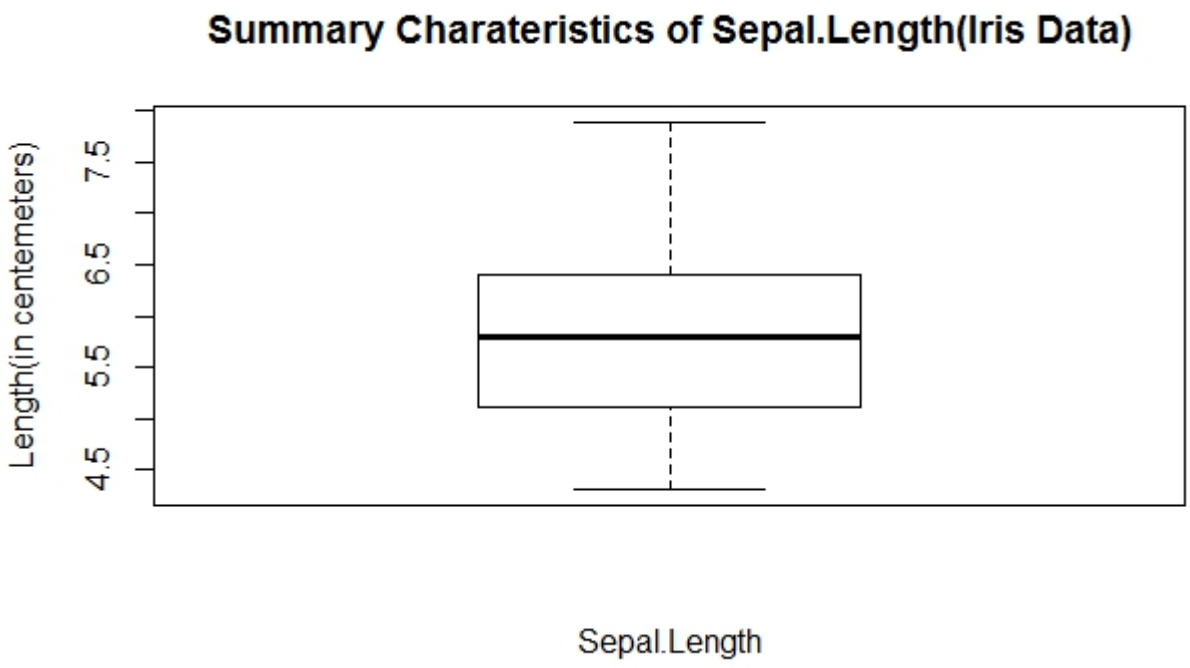
В этом примере используется `boxplot()` по умолчанию и рамка данных `iris` .

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Простой boxplot (Sepal.Length)

Создайте граф с квадратным и усырчатым числом числовой переменной

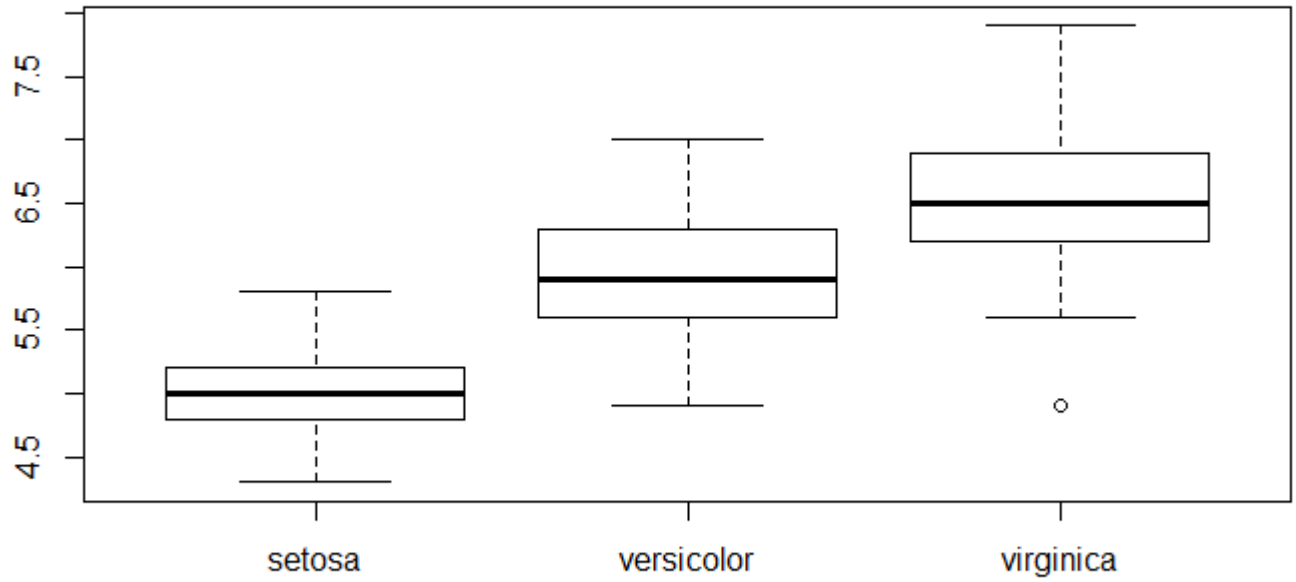
```
boxplot(iris[,1],xlab="Sepal.Length",ylab="Length(in centemeters)",
           main="Summary Charateristics of Sepal.Length(Iris Data)")
```



Коробка длины сепалы, сгруппированная по видам

Создайте прямоугольник с числовой переменной, сгруппированной по категориальной переменной

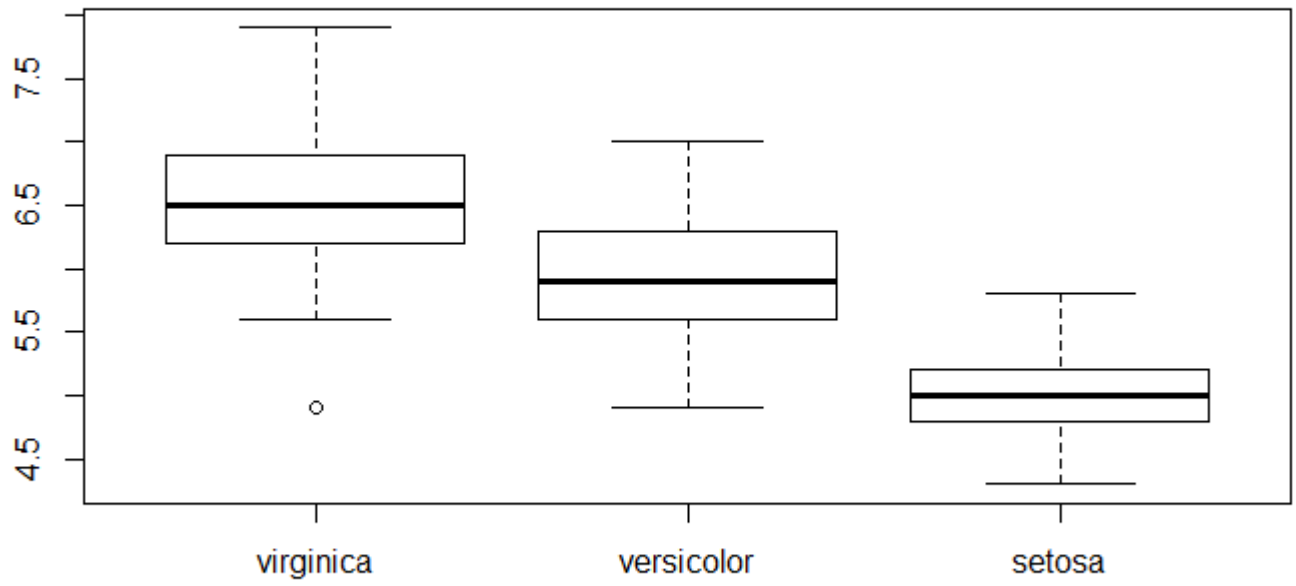
```
boxplot(Sepal.Length~Species,data = iris)
```



Принести заказ

Чтобы изменить порядок поля на графике, вам необходимо изменить порядок уровней категориальной переменной.
Например, если мы хотим иметь порядок `virginica - versicolor - setosa`

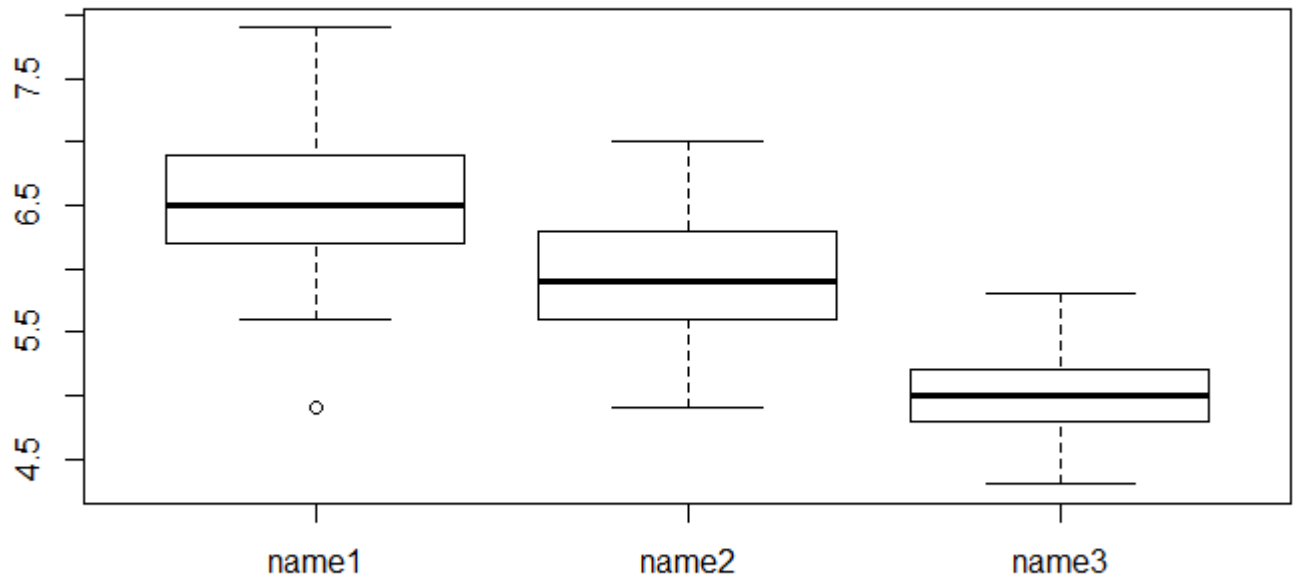
```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica","versicolor","setosa"))
boxplot(Sepal.Length~newSpeciesOrder,data = iris)
```



Изменение имен групп

Если вы хотите указать лучшее имя для своих групп, вы можете использовать параметр `Names` . Он принимает вектор размера уровней категориальной переменной

```
boxplot(Sepal.Length~newSpeciesOrder,data = iris,names= c("name1","name2","name3"))
```

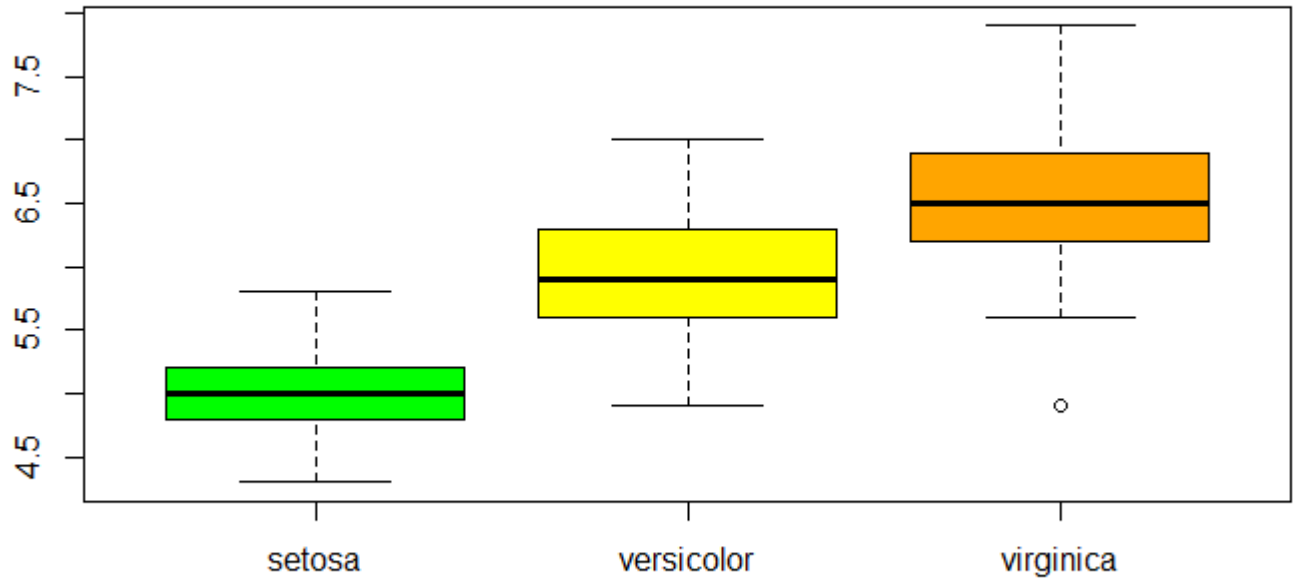


Небольшие улучшения

цвет

`col` : добавить вектор размера уровней категориальной переменной

```
boxplot(Sepal.Length~Species,data = iris,col=c("green","yellow","orange"))
```

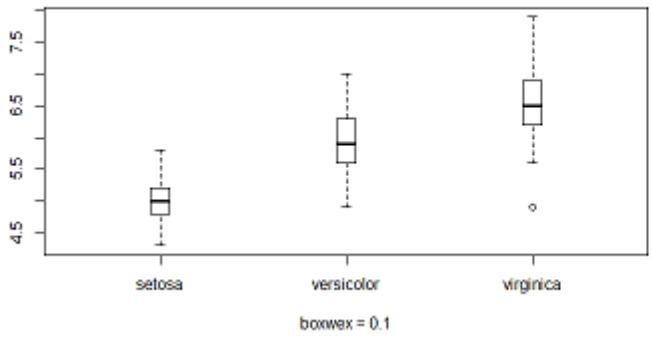


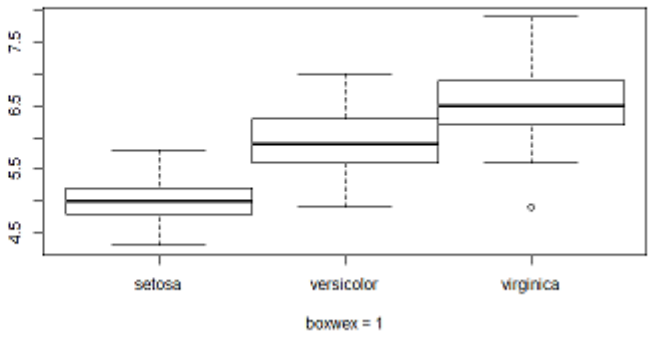
Близость коробки

`boxwex` : установите поле между полями.

Левый `boxplot(Sepal.Length~Species,data = iris,boxwex = 0.1)`

Right `boxplot(Sepal.Length~Species,data = iris,boxwex = 1)`





См. Сводки, которые основаны на ящиках:

`plot=FALSE`

Чтобы увидеть резюме , вы должны поставить В параметре `plot` в `FALSE` .
Даны различные результаты

```
> boxplot(Sepal.Length~newSpeciesOrder,data = iris,plot=FALSE)
```

```
$stats #summary of the numerical variable for the 3 groups
     [,1] [,2] [,3]
[1,]  5.6  4.9  4.3 # extreme value
[2,]  6.2  5.6  4.8 # first quartile limit
[3,]  6.5  5.9  5.0 # median limit
[4,]  6.9  6.3  5.2 # third quartile limit
[5,]  7.9  7.0  5.8 # extreme value

$n #number of observations in each groups
[1] 50 50 50

$conf #extreme value of the notchs
         [,1]     [,2]     [,3]
[1,] 6.343588 5.743588 4.910622
[2,] 6.656412 6.056412 5.089378

$out #extreme value
[1] 4.9

$group #group in which are the extreme value
[1] 1

$names #groups names
[1] "virginica"  "versicolor" "setosa"
```

Дополнительные параметры стиля boxplot.

коробка

- boxlty - тип линии коробки
- boxlwd - ширина линии коробки
- boxcol - цвет линии коробки
- Цвет заливки - заполнение коробки

медиана

- medlty - средний тип линии («пустой» без строки)
- medlwd - средний размер линии
- medcol - средний цвет линии
- medpch - медианная точка (NA без символа)
- medcex - средний размер точки
- medbg - средний цвет фона точки

бакенбарды

- whisklty - тип линии вискеров
- whisklwd - ширина линии вискера
- цвет висколла - цвет вискеров

штапель

- staplelty - тип штапельной линии
- staplelwd - ширина штапельной линии
- Цвет штапеля - цвет штапельной линии

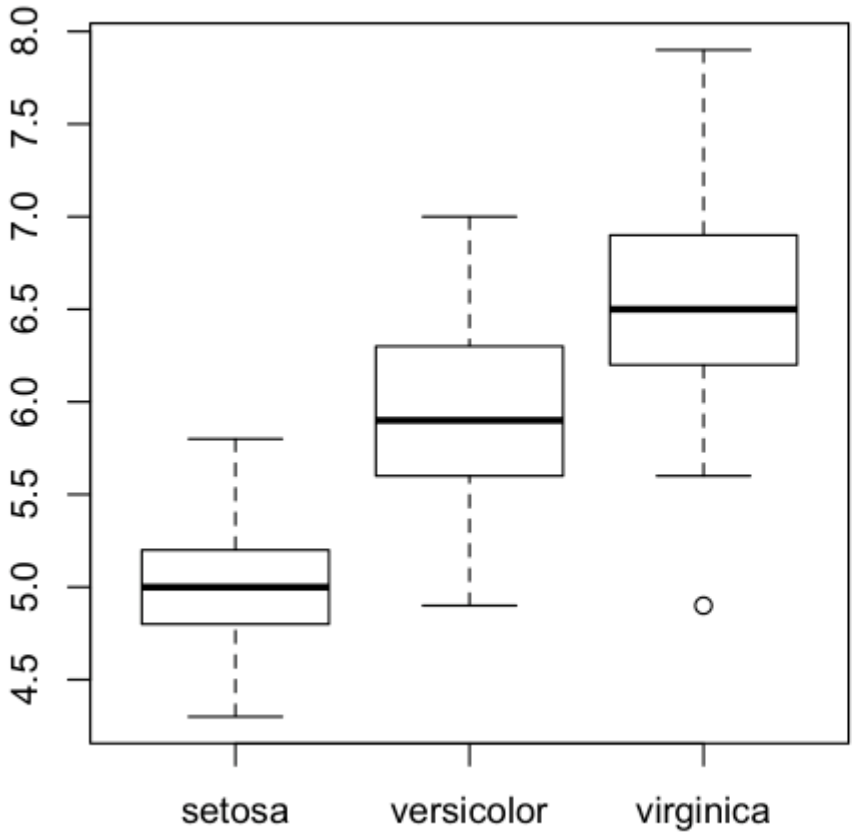
Выпадающие

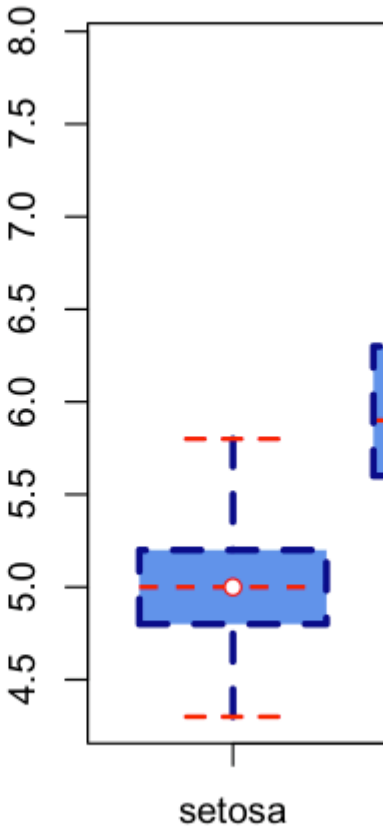
- outlty - тип линии outlier («blank» без строки)
- outlwd - ширина линии outlier
- цвет линии outcol - outlier
- outpch - тип точки outlier (NA без символа)
- outcex - размер точки выброса
- outbg - цвет фона точки outlier

пример

По умолчанию и сильно измененные участки бок о бок

```
par(mfrow=c(1,2))
# Default
boxplot(Sepal.Length ~ Species, data=iris)
# Modified
boxplot(Sepal.Length ~ Species, data=iris,
        boxlty=2, boxlwd=3, boxfill="cornflowerblue", boxcol="darkblue",
        medlty=2, medlwd=2, medcol="red", medpch=21, medcex=1, medbg="white",
        whisklty=2, whisklwd=3, whiskcol="darkblue",
        staplelty=2, staplelwd=2, staplecol="red",
        outlty=3, outlwd=3, outcol="grey", outpch=NA
        )
```





Прочитайте boxplot онлайн: https://riptutorial.com/ru/r/topic/1005/boxplot

глава 6: dplyr

замечания

dplyr - это итерация plyr, которая предоставляет гибкие функции на основе глагола для управления данными в R. Последняя версия dplyr может быть загружена из CRAN с использованием

```
install.package("dplyr")
```

Ключевым объектом в dplyr является tbl, представление структуры табличных данных. В настоящее время dplyr (версия 0.5.0) поддерживает:

- кадры данных
- таблицы данных
- SQLite
- PostgreSQL / Redshift
- MySQL / MariaDB
- BigQuery
- MonetDB
- кубы данных с массивами (частичная реализация)

Examples

Табличные глаголы dplyr

dplyr вводит грамматику манипуляции данными в `R` Он обеспечивает согласованный интерфейс для работы с данными независимо от того, где он хранится: data.frame , data.table или `database` . `dplyr` написаны с использованием Rcpp , что делает его очень быстрым для работы с данными в памяти.

`dplyr` состоит в том, чтобы иметь небольшие функции, которые хорошо справляются. Пять простых функций (`filter` , `arrange` , `select` , `mutate` , и `summarise`) могут быть использованы для выявления новых способов описания данных. В сочетании с `group_by` эти функции могут использоваться для вычисления статистических статистических данных группы.

Синтаксические общности

Все эти функции имеют аналогичный синтаксис:

- Первый аргумент для всех этих функций - это всегда кадр данных
- Столбцы могут передаваться напрямую, используя имена переменных (т. Е. Без

использования `$`)
- Эти функции не изменяют сами исходные данные, т. Е. Не имеют побочных эффектов. Следовательно, результаты всегда должны сохраняться в объекте.

Мы будем использовать встроенный набор данных mtcars для изучения отдельных табличных глаголов `dplyr` . Прежде чем преобразовать тип `mtcars` в `tbl_df` (поскольку он делает очиститель для печати), мы добавляем `rownames` набора данных в виде столбца, используя функцию `rownames_to_column` из пакета тиблей .

```
library(dplyr) # This documentation was written using version 0.5.0

mtcars_tbl <- as_data_frame(tibble::rownames_to_column(mtcars, "cars"))

# examine the structure of data
head(mtcars_tbl)

# A tibble: 6 x 12
#               cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#              <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1          Mazda RX4  21.0     6   160   110  3.90 2.620 16.46     0     1     4     4
#2      Mazda RX4 Wag  21.0     6   160   110  3.90 2.875 17.02     0     1     4     4
#3         Datsun 710  22.8     4   108    93  3.85 2.320 18.61     1     1     4     1
#4     Hornet 4 Drive  21.4     6   258   110  3.08 3.215 19.44     1     0     3     1
#5  Hornet Sportabout  18.7     8   360   175  3.15 3.440 17.02     0     0     3     2
#6            Valiant  18.1     6   225   105  2.76 3.460 20.22     1     0     3     1
```

фильтр

`filter` помогает подмножества строк, которые соответствуют определенным критериям. Первым аргументом является имя `data.frame` а второй (и последующий) аргументы - это критерии, которые фильтруют данные (эти критерии должны оцениваться как `TRUE` и `FALSE`)

Подмножество всех автомобилей, имеющих 4 *цилиндра* - `cyl` :

```
filter(mtcars_tbl, cyl == 4)

# A tibble: 11 x 12
#              cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#             <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1       Datsun 710  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
#2        Merc 240D  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
#3         Merc 230  22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
#4         Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
#5      Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
# ... with 6 more rows
```

Мы можем передать несколько критериев, разделенных запятой. Чтобы подмножить автомобили, которые имеют либо 4, либо 6 *цилиндров* - `cyl` и имеют 5 *передач* - `gear` :

```
filter(mtcars_tbl, cyl == 4 | cyl == 6, gear == 5)
```

```
# A tibble: 3 x 12
#           cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#          <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Porsche 914-2  26.0     4 120.3    91  4.43 2.140  16.7     0     1     5     2
#2  Lotus Europa  30.4     4  95.1   113  3.77 1.513  16.9     1     1     5     2
#3  Ferrari Dino  19.7     6 145.0   175  3.62 2.770  15.5     0     1     5     6
```

`filter` выбирает строки на основе критериев, выбирает строки по положению, использует `slice` . `slice` принимает только 2 аргумента: первый - это `data.frame` а второй - целые значения строк.

Чтобы выбрать строки с 6 по 9:

```
slice(mtcars_tbl, 6:9)

# A tibble: 4 x 12
#        cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#       <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1    Valiant  18.1     6 225.0   105  2.76  3.46 20.22     1     0     3     1
#2 Duster 360  14.3     8 360.0   245  3.21  3.57 15.84     0     0     3     4
#3  Merc 240D  24.4     4 146.7    62  3.69  3.19 20.00     1     0     4     2
#4   Merc 230  22.8     4 140.8    95  3.92  3.15 22.90     1     0     4     2
```

Или же:

```
slice(mtcars_tbl, -c(1:5, 10:n()))
```

Это приводит к тому же выводу, что и `slice(mtcars_tbl, 6:9)`

`n()` представляет количество наблюдений в текущей группе

организовать

`arrange` используется для сортировки данных по указанной переменной (ей). Как и предыдущий глагол (и все остальные функции в `dplyr`), первым аргументом является `data.frame` , а последующие аргументы используются для сортировки данных. Если передано несколько переменных, данные сначала сортируются по первой переменной, а затем по второй переменной и т. Д.

Заказать данные на *лошадиных силах* - `hp`

```
arrange(mtcars_tbl, hp)

# A tibble: 32 x 12
#             cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1     Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
#2       Merc 240D  24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
#3  Toyota Corolla  33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
#4        Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
```

```
#5       Fiat X1-9  27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
#6   Porsche 914-2  26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
# ... with 26 more rows
```

Чтобы `arrange` данные по *мили на галлон* - `mpg` на `mpg` в порядке убывания, а затем *количество цилиндров* - `cyl` :

```
arrange(mtcars_tbl, desc(mpg), cyl)

# A tibble: 32 x 12
#             cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#            <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1  Toyota Corolla  33.9     4  71.1    65  4.22 1.835 19.90     1     1     4     1
#2        Fiat 128  32.4     4  78.7    66  4.08 2.200 19.47     1     1     4     1
#3     Honda Civic  30.4     4  75.7    52  4.93 1.615 18.52     1     1     4     2
#4    Lotus Europa  30.4     4  95.1   113  3.77 1.513 16.90     1     1     5     2
#5       Fiat X1-9  27.3     4  79.0    66  4.08 1.935 18.90     1     1     4     1
#6   Porsche 914-2  26.0     4 120.3    91  4.43 2.140 16.70     0     1     5     2
# ... with 26 more rows
```

Выбрать

`select` используется для выбора только подмножества переменных. Чтобы выбрать только `mpg` , `disp` , `wt` , `qsec` и `vs` из `mtcars_tbl` :

```
select(mtcars_tbl, mpg, disp, wt, qsec, vs)

# A tibble: 32 x 5
#     mpg  disp    wt  qsec    vs
#   <dbl> <dbl> <dbl> <dbl> <dbl>
#1   21.0 160.0 2.620 16.46     0
#2   21.0 160.0 2.875 17.02     0
#3   22.8 108.0 2.320 18.61     1
#4   21.4 258.0 3.215 19.44     1
#5   18.7 360.0 3.440 17.02     0
#6   18.1 225.0 3.460 20.22     1
# ... with 26 more rows
```

`:` нотация может использоваться для выбора последовательных столбцов. Чтобы выбрать столбцы из `cars` через `disp` и `vs` через `carb` :

```
select(mtcars_tbl, cars:disp, vs:carb)

# A tibble: 32 x 8
#                cars   mpg   cyl  disp    vs    am  gear  carb
#               <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1          Mazda RX4  21.0     6 160.0     0     1     4     4
#2      Mazda RX4 Wag  21.0     6 160.0     0     1     4     4
#3         Datsun 710  22.8     4 108.0     1     1     4     1
#4     Hornet 4 Drive  21.4     6 258.0     1     0     3     1
#5  Hornet Sportabout  18.7     8 360.0     0     0     3     2
#6            Valiant  18.1     6 225.0     1     0     3     1
# ... with 26 more rows
```

или `select(mtcars_tbl, -(hp:qsec))`

Для наборов данных, которые содержат несколько столбцов, может оказаться утомительным выбор нескольких столбцов по имени. Для облегчения жизни существует ряд вспомогательных функций (таких как `starts_with()` , `ends_with()` , `contains()` `matches()` , `num_range()` , `one_of()` , `one_of()` и `everything()`), которые могут использоваться в `select` , Чтобы узнать больше о том, как их использовать, см. « `?select_helpers` и « `?select` .

Примечание . При обращении к столбцам непосредственно в `select()` мы используем имена нулевых столбцов, но кавычки должны использоваться при обращении к столбцам в вспомогательных функциях.

Чтобы переименовать столбцы при выборе:

```
select(mtcars_tbl, cylinders = cyl, displacement = disp)

# A tibble: 32 x 2
#   cylinders displacement
#       <dbl>        <dbl>
#1          6        160.0
#2          6        160.0
#3          4        108.0
#4          6        258.0
#5          8        360.0
#6          6        225.0
# ... with 26 more rows
```

Как и ожидалось, это уменьшает все другие переменные.

Чтобы переименовать столбцы, не отбрасывая другие переменные, используйте `rename` :

```
rename(mtcars_tbl, cylinders = cyl, displacement = disp)

# A tibble: 32 x 12
#                cars   mpg cylinders displacement    hp  drat    wt  qsec    vs
#               <chr> <dbl>     <dbl>        <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1          Mazda RX4  21.0         6        160.0   110  3.90 2.620 16.46     0
#2      Mazda RX4 Wag  21.0         6        160.0   110  3.90 2.875 17.02     0
#3         Datsun 710  22.8         4        108.0    93  3.85 2.320 18.61     1
#4     Hornet 4 Drive  21.4         6        258.0   110  3.08 3.215 19.44     1
#5  Hornet Sportabout  18.7         8        360.0   175  3.15 3.440 17.02     0
#6            Valiant  18.1         6        225.0   105  2.76 3.460 20.22     1
# ... with 26 more rows, and 3 more variables: am <dbl>, gear <dbl>, carb <dbl>
```

мутировать

`mutate` можно использовать для добавления новых столбцов в данные. Как и все другие функции в `dplyr` , mutate не добавляет вновь созданные столбцы к исходным данным. Столбцы добавляются в конце `data.frame` .

```
mutate(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)

# A tibble: 32 x 14
#                cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
weight_ton weight_pounds
#               <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
<dbl>         <dbl>
#1          Mazda RX4  21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
1.3100          2620
#2      Mazda RX4 Wag  21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
1.4375          2875
#3         Datsun 710  22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
1.1600          2320
#4     Hornet 4 Drive  21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
1.6075          3215
#5  Hornet Sportabout  18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
1.7200          3440
#6            Valiant  18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
1.7300          3460
# ... with 26 more rows
```

Обратите внимание на использование `weight_ton` при создании `weight_pounds` . В отличие от базы `R` , `mutate` позволяет нам ссылаться на только что созданные столбцы, которые будут использоваться для последующей операции.

Чтобы сохранить только вновь созданные столбцы, используйте `transmute` вместо `mutate` :

```
transmute(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)

# A tibble: 32 x 2
#   weight_ton weight_pounds
#        <dbl>         <dbl>
#1      1.3100          2620
#2      1.4375          2875
#3      1.1600          2320
#4      1.6075          3215
#5      1.7200          3440
#6      1.7300          3460
# ... with 26 more rows
```

суммировать

`summarise` вычисляет сводную статистику переменных, сворачивая несколько значений на одно значение. Он может вычислять множество статистических данных, и мы можем назвать эти сводные столбцы в одном и том же выражении.

Чтобы вычислить *среднее* и *стандартное отклонение* `mpg` и `disp` всех автомобилей в наборе данных:

```
summarise(mtcars_tbl, mean_mpg = mean(mpg), sd_mpg = sd(mpg),
          mean_disp = mean(disp), sd_disp = sd(disp))

# A tibble: 1 x 4
```

```
#  mean_mpg   sd_mpg mean_disp  sd_disp
#     <dbl>    <dbl>     <dbl>    <dbl>
#1 20.09062 6.026948  230.7219 123.9387
```

группа по

`group_by` может использоваться для выполнения групповых действий над данными. Когда глаголы, определенные выше, применяются к этим сгруппированным данным, они автоматически применяются к каждой группе отдельно.

Чтобы найти *`mean`* и *`sd mpg`* by *`cyl`* :

```
by_cyl <- group_by(mtcars_tbl, cyl)
summarise(by_cyl, mean_mpg = mean(mpg), sd_mpg = sd(mpg))


# A tibble: 3 x 3
#    cyl mean_mpg   sd_mpg
#  <dbl>    <dbl>    <dbl>
#1     4 26.66364 4.509828
#2     6 19.74286 1.453567
#3     8 15.10000 2.560048
```

Помещая все это

Мы выбираем столбцы из `cars` через `hp` и `gear` , заказываем ряды по `cyl` и от самого высокого до самого низкого `mpg` на `mpg` , группируем данные по `gear` , и, наконец, подмножество только тех автомобилей имеет `mpg` > 20 и `hp` > 75

```
selected <- select(mtcars_tbl, cars:hp, gear)
ordered <- arrange(selected, cyl, desc(mpg))
by_cyl <- group_by(ordered, gear)
filter(by_cyl, mpg > 20, hp > 75)

Source: local data frame [9 x 6]
Groups: gear [3]

#            cars   mpg   cyl  disp    hp  gear
#           <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#1   Lotus Europa  30.4     4  95.1   113     5
#2  Porsche 914-2  26.0     4 120.3    91     5
#3     Datsun 710  22.8     4 108.0    93     4
#4       Merc 230  22.8     4 140.8    95     4
#5  Toyota Corona  21.5     4 120.1    97     3
# ... with 4 more rows
```

Возможно, нам не нужны промежуточные результаты, мы можем добиться того же результата, что и выше, путем обнуления вызовов функций:

```
filter(
    group_by(
        arrange(
            select(
                mtcars_tbl, cars:hp
            ), cyl, desc(mpg)
        ), cyl
    ),mpg > 20, hp > 75
)
```

Это может быть немного трудно читать. Таким образом, операции `dplyr` могут быть скованы при помощи оператора pipe `%>%` . Вышеприведенный код трансалирует:

```
mtcars_tbl %>%
    select(cars:hp) %>%
    arrange(cyl, desc(mpg)) %>%
    group_by(cyl) %>%
    filter(mpg > 20, hp > 75)
```

суммировать несколько столбцов

`dplyr` предоставляет `summarise_all()` для применения функций ко всем (`summarise_all()`) столбцам.

Чтобы найти количество различных значений для каждого столбца:

```
mtcars_tbl %>%
    summarise_all(n_distinct)

# A tibble: 1 x 12
#   cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#  <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#1    32    25     3    27    22    22    29    30     2     2     3     6
```

Чтобы найти количество различных значений для каждого столбца на `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_all(n_distinct)

# A tibble: 3 x 12
#    cyl  cars   mpg  disp    hp  drat    wt  qsec    vs    am  gear  carb
#  <dbl> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#1     4    11     9    11    10    10    11    11     2     2     3     2
#2     6     7     6     5     4     5     6     7     2     2     3     3
#3     8    14    12    11     9    11    13    14     1     2     2     4
```

Обратите внимание, что нам просто нужно было добавить оператор `group_by` а остальная часть кода такая же. Выход теперь состоит из трех строк - по одному для каждого уникального значения `cyl` .

Чтобы `summarise` определенные несколько столбцов, используйте `summarise_at`

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"), mean)

# A tibble: 3 x 4
#    cyl      mpg     disp        hp
#  <dbl>    <dbl>    <dbl>     <dbl>
#1     4 26.66364 105.1364  82.63636
#2     6 19.74286 183.3143 122.28571
#3     8 15.10000 353.1000 209.21429
```

`helper` функции (`?select_helpers`) могут использоваться вместо имен столбцов для выбора определенных столбцов

Чтобы применить несколько функций, либо передайте имена функций в виде символьного вектора:

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 c("mean", "sd"))
```

или обернуть их внутри `funs` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 funs(mean, sd))

# A tibble: 3 x 7
#    cyl mpg_mean disp_mean   hp_mean   mpg_sd  disp_sd    hp_sd
#  <dbl>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364  105.1364  82.63636 4.509828 26.87159 20.93453
#2     6 19.74286  183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000  353.1000 209.21429 2.560048 67.77132 50.97689
```

Имена столбцов теперь добавляются с именами функций, чтобы они отличались друг от друга. Чтобы изменить это, передайте имя, которое будет добавлено с помощью функции:

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 c(Mean = "mean", SD = "sd"))

mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_at(c("mpg", "disp", "hp"),
                 funs(Mean = mean, SD = sd))


# A tibble: 3 x 7
#    cyl mpg_Mean disp_Mean   hp_Mean   mpg_SD  disp_SD    hp_SD
```

```
#  <dbl>    <dbl>     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364  105.1364  82.63636 4.509828 26.87159 20.93453
#2     6 19.74286  183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000  353.1000 209.21429 2.560048 67.77132 50.97689
```

Чтобы выбрать столбцы условно, используйте `summarise_if` :

Возьмем `mean` всех столбцов, `numeric` сгруппированные по `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_if(is.numeric, mean)

# A tibble: 3 x 11
#    cyl      mpg     disp        hp     drat       wt     qsec
#  <dbl>    <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
# ... with 4 more variables: vs <dbl>, am <dbl>, gear <dbl>,
#   carb <dbl>
```

Однако некоторые переменные являются дискретными, а `mean` этих переменных не имеет смысла.

Чтобы взять `mean` только непрерывных переменных через `cyl` :

```
mtcars_tbl %>%
    group_by(cyl) %>%
    summarise_if(function(x) is.numeric(x) & n_distinct(x) > 6, mean)

# A tibble: 3 x 7
#    cyl      mpg     disp        hp     drat       wt     qsec
#  <dbl>    <dbl>    <dbl>     <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
```

Наблюдение подмножества (строки)

`dplyr::filter()` - выберите подмножество строк в кадре данных, которые соответствуют логическим критериям:

```
dplyr::filter(iris,Sepal.Length>7)
#       Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#    1           7.1         3.0          5.9         2.1 virginica
#    2           7.6         3.0          6.6         2.1 virginica
#    3           7.3         2.9          6.3         1.8 virginica
#    4           7.2         3.6          6.1         2.5 virginica
#    5           7.7         3.8          6.7         2.2 virginica
#    6           7.7         2.6          6.9         2.3 virginica
#    7           7.7         2.8          6.7         2.0 virginica
```

```
#    8           7.2         3.2          6.0         1.8 virginica
#    9           7.2         3.0          5.8         1.6 virginica
#    10          7.4         2.8          6.1         1.9 virginica
#    11          7.9         3.8          6.4         2.0 virginica
#    12          7.7         3.0          6.1         2.3 virginica
```

`dplyr::distinct()` - удалить повторяющиеся строки:

```
distinct(iris, Sepal.Length, .keep_all = TRUE)
#       Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#    1           5.1         3.5          1.4         0.2     setosa
#    2           4.9         3.0          1.4         0.2     setosa
#    3           4.7         3.2          1.3         0.2     setosa
#    4           4.6         3.1          1.5         0.2     setosa
#    5           5.0         3.6          1.4         0.2     setosa
#    6           5.4         3.9          1.7         0.4     setosa
#    7           4.4         2.9          1.4         0.2     setosa
#    8           4.8         3.4          1.6         0.2     setosa
#    9           4.3         3.0          1.1         0.1     setosa
#   10          5.8         4.0          1.2         0.2     setosa
#   11          5.7         4.4          1.5         0.4     setosa
#   12          5.2         3.5          1.5         0.2     setosa
#   13          5.5         4.2          1.4         0.2     setosa
#   14          4.5         2.3          1.3         0.3     setosa
#   15          5.3         3.7          1.5         0.2     setosa
#   16          7.0         3.2          4.7         1.4 versicolor
#   17          6.4         3.2          4.5         1.5 versicolor
#   18          6.9         3.1          4.9         1.5 versicolor
#   19          6.5         2.8          4.6         1.5 versicolor
#   20          6.3         3.3          4.7         1.6 versicolor
#   21          6.6         2.9          4.6         1.3 versicolor
#   22          5.9         3.0          4.2         1.5 versicolor
#   23          6.0         2.2          4.0         1.0 versicolor
#   24          6.1         2.9          4.7         1.4 versicolor
#   25          5.6         2.9          3.6         1.3 versicolor
#   26          6.7         3.1          4.4         1.4 versicolor
#   27          6.2         2.2          4.5         1.5 versicolor
#   28          6.8         2.8          4.8         1.4 versicolor
#   29          7.1         3.0          5.9         2.1  virginica
#   30          7.6         3.0          6.6         2.1  virginica
#   31          7.3         2.9          6.3         1.8  virginica
#   32          7.2         3.6          6.1         2.5  virginica
#   33          7.7         3.8          6.7         2.2  virginica
#   34          7.4         2.8          6.1         1.9  virginica
#   35          7.9         3.8          6.4         2.0  virginica
```

Агрегирование с помощью оператора%>% (pipe)

Трубы (%>%) оператор может быть использован в комбинации с `dplyr` функций. В этом примере мы используем набор данных `mtcars` (см. `help("mtcars")` для получения дополнительной информации), чтобы показать, как суммировать кадр данных, и добавлять переменные к данным с результатом применения функции.

```
library(dplyr)
library(magrittr)
```

```
df <- mtcars
df$cars <- rownames(df) #just add the cars names to the df
df <- df[,c(ncol(df),1:(ncol(df)-1))] # and place the names in the first column
```

1. Сумаризировать данные

Для вычисления статистики мы используем `summarize` и соответствующие функции. В этом случае `n()` используется для подсчета числа случаев.

```
 df %>%
  summarize(count=n(),mean_mpg = mean(mpg, na.rm = TRUE),
            min_weight = min(wt),max_weight = max(wt))

#  count mean_mpg min_weight max_weight
#1    32 20.09062      1.513      5.424
```

2. Вычислить статистику по группам

Можно вычислить статистику по группам данных. В этом случае по *количеству цилиндров* и *количеству передних передач*

```
df %>%
  group_by(cyl, gear) %>%
  summarize(count=n(),mean_mpg = mean(mpg, na.rm = TRUE),
            min_weight = min(wt),max_weight = max(wt))

# Source: local data frame [8 x 6]
# Groups: cyl [?]
#
#    cyl  gear count mean_mpg min_weight max_weight
#  <dbl> <dbl> <int>    <dbl>      <dbl>      <dbl>
#1     4     3     1   21.500      2.465      2.465
#2     4     4     8   26.925      1.615      3.190
#3     4     5     2   28.200      1.513      2.140
#4     6     3     2   19.750      3.215      3.460
#5     6     4     4   19.750      2.620      3.440
#6     6     5     1   19.700      2.770      2.770
#7     8     3    12   15.050      3.435      5.424
#8     8     5     2   15.400      3.170      3.570
```

Примеры NSE и строковых переменных в dplyr

`dplyr` использует `dplyr` оценку (NSE), поэтому мы обычно можем использовать имена переменных без кавычек. Однако иногда во время конвейера данных нам нужно получить наши имена переменных из других источников, таких как окно выбора Shiny. В случае таких функций, как `select` , мы можем просто использовать `select_` для использования строковой переменной для выбора

```
variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
iris %>%
select_(variable1, variable2) %>%
```

```
head(n=5)
#  Sepal.Length Sepal.Width
#  1          5.1         3.5
#  2          4.9         3.0
#  3          4.7         3.2
#  4          4.6         3.1
#  5          5.0         3.6
```

Но если мы хотим использовать другие функции , такие как подвести итог или фильтр необходимо использовать `interp` функцию из `lazyeval` пакета

```
variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
variable3 <- "Species"
iris %>%
select_(variable1, variable2, variable3) %>%
group_by_(variable3) %>%
summarize_(mean1 = lazyeval::interp(~mean(var), var = as.name(variable1)), mean2 =
lazyeval::interp(~mean(var), var = as.name(variable2)))
#      Species mean1 mean2
#       <fctr> <dbl> <dbl>
# 1     setosa 5.006 3.428
# 2 versicolor 5.936 2.770
# 3  virginica 6.588 2.974
```

Прочитайте dplyr онлайн: https://riptutorial.com/ru/r/topic/4250/dplyr

глава 7: ggplot2

замечания

`ggplot2` имеет свой собственный идеальный справочный сайт http://ggplot2.tidyverse.org/ .

В большинстве случаев удобнее адаптировать структуру или содержимое построенных данных (например, `data.frame`), чем затем корректировать объекты внутри графика.

RStudio публикует очень полезную «Визуализацию данных с помощью ggplot2», которую можно найти здесь .

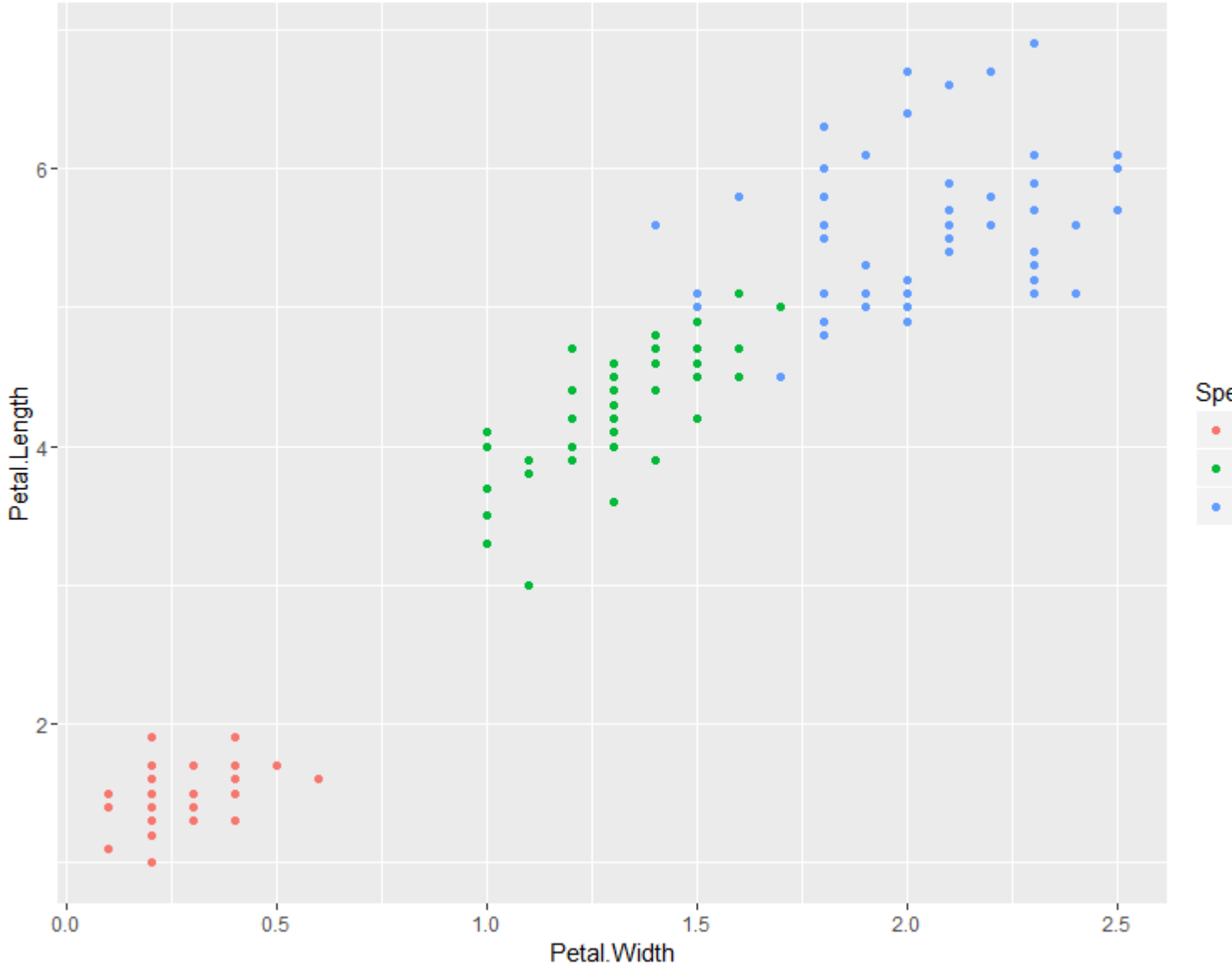
Examples

Границы рассеяния

Мы строим простой график рассеяния, используя набор встроенных диафрагм:

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

Это дает:

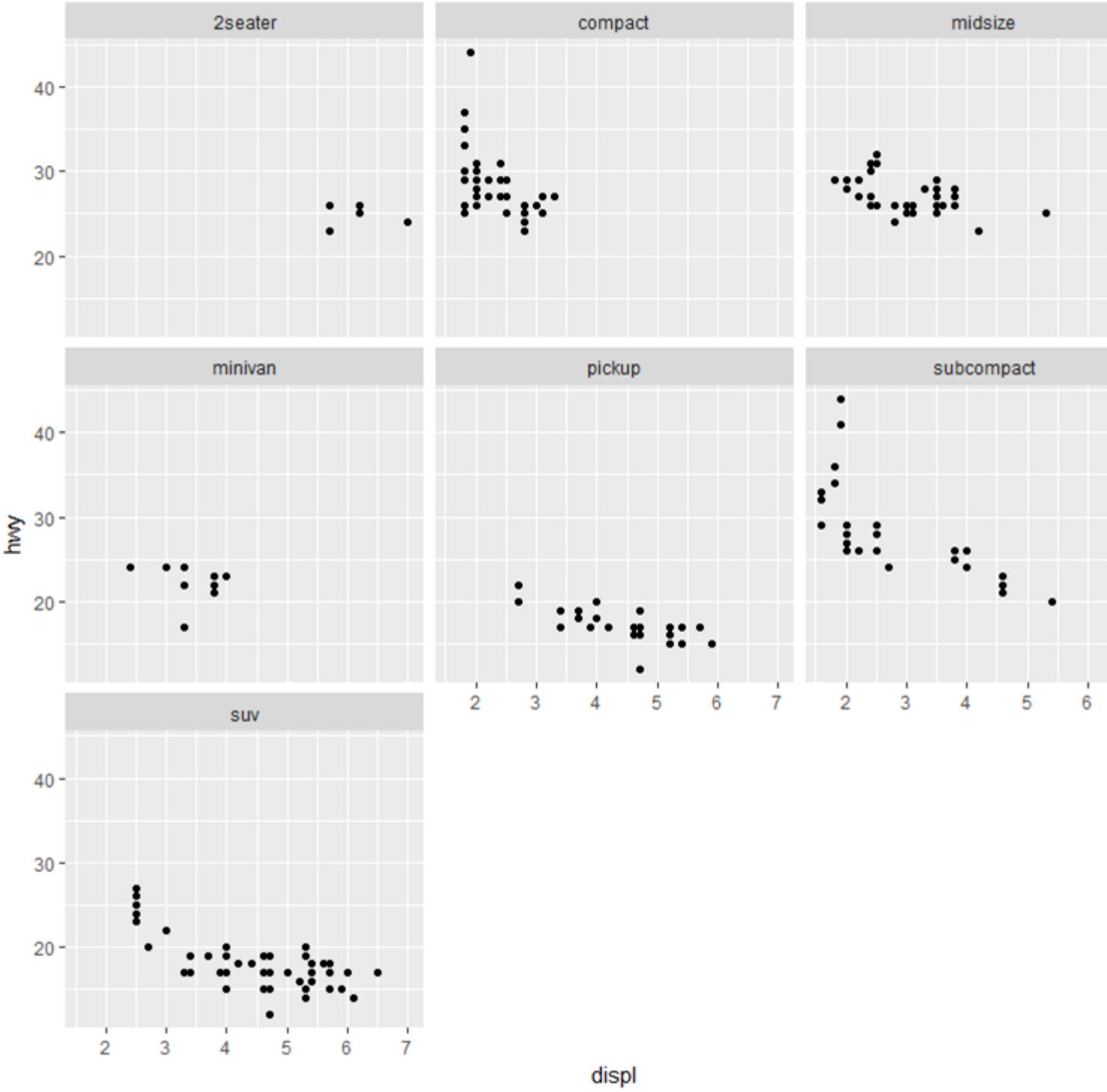


Отображение нескольких графиков

Отображение нескольких графиков в одном изображении с различными функциями `facet` . Преимущество этого метода заключается в том, что все оси имеют один и тот же масштаб в диаграммах, что упрощает их сравнение с первого взгляда. Мы будем использовать набор данных `mpg` включенный в `ggplot2` .

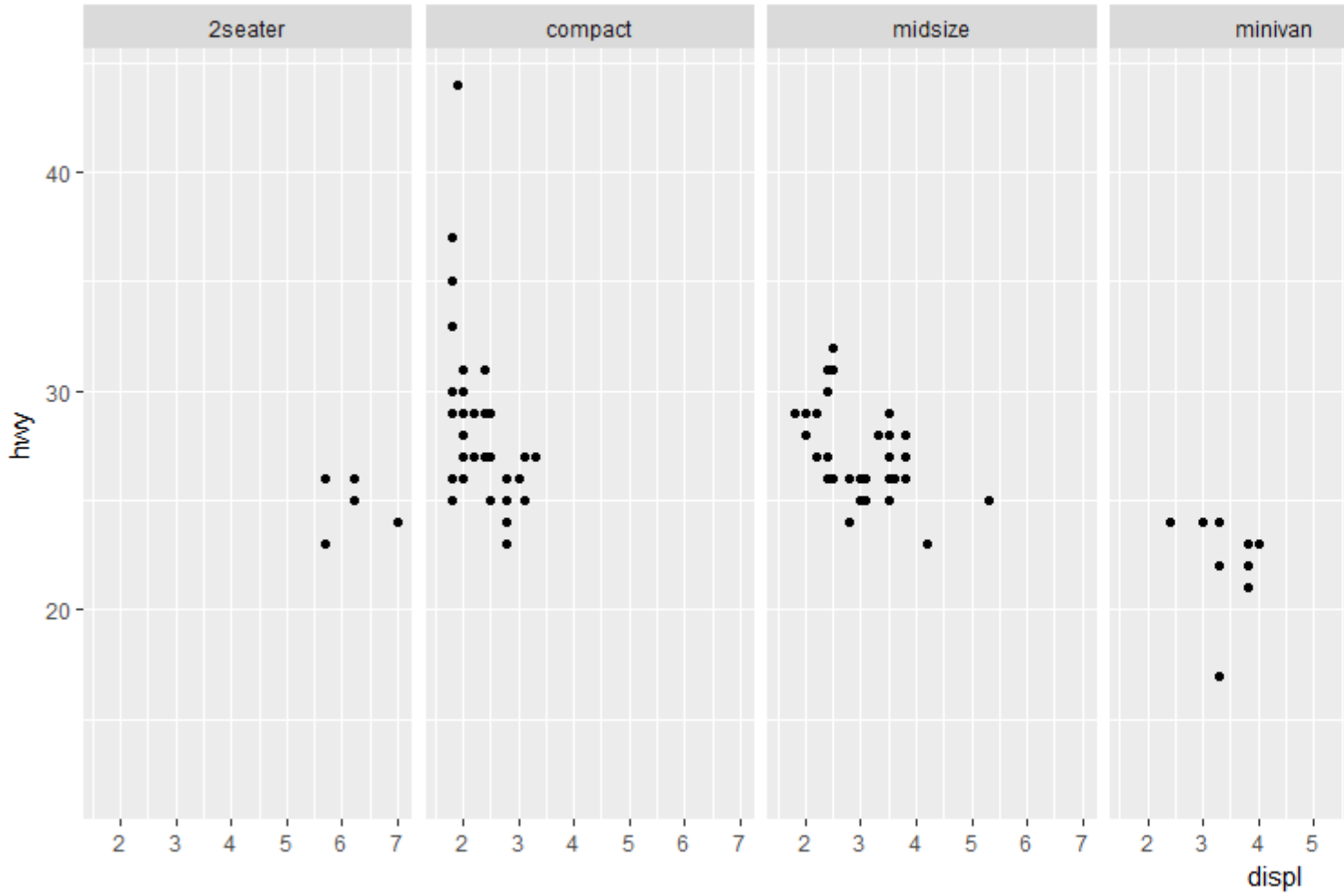
Обтекание диаграмм по строкам (попытки создать квадратную компоновку):

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_wrap(~class)
```



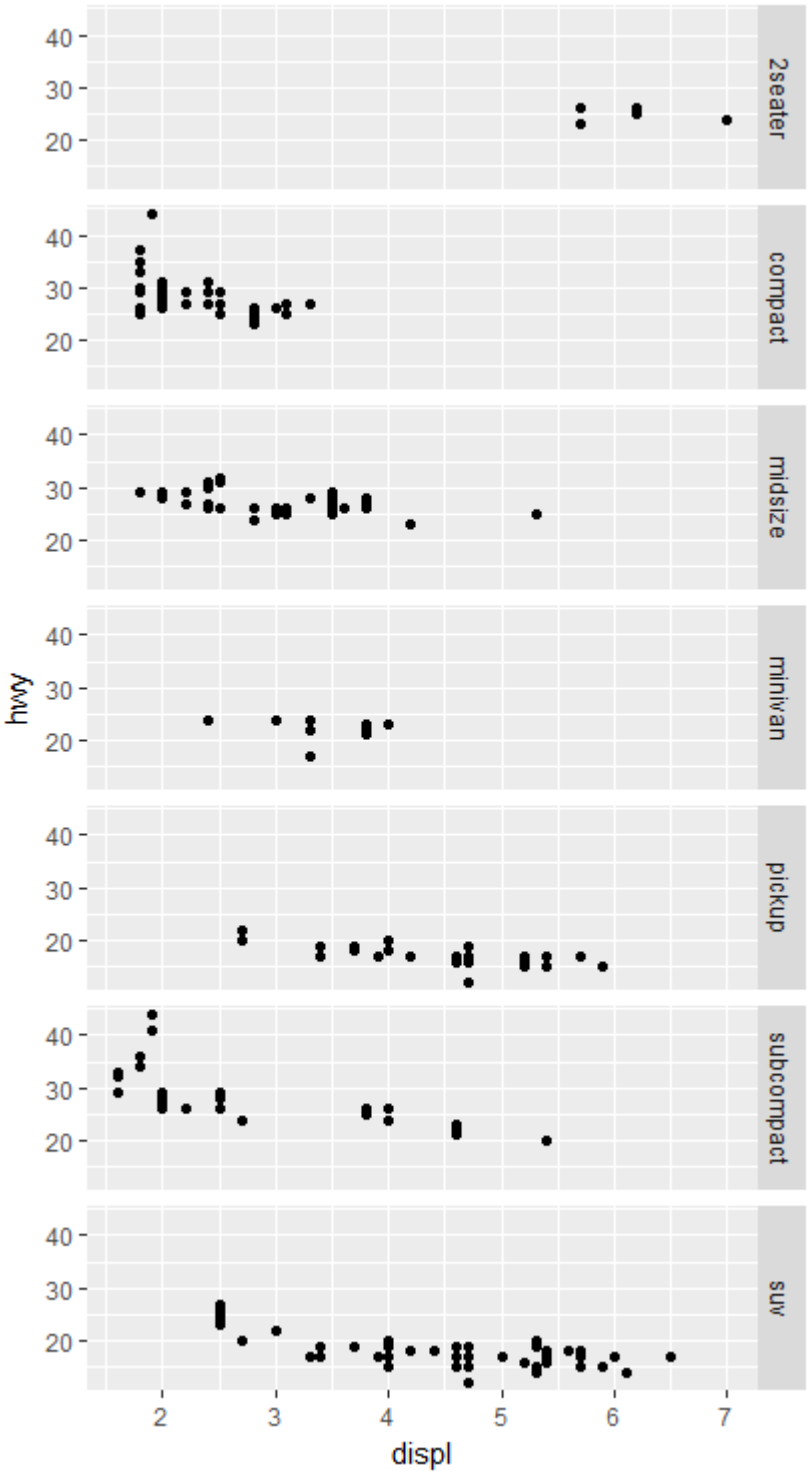
Отображать несколько диаграмм в одной строке, несколько столбцов:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(.~class)
```



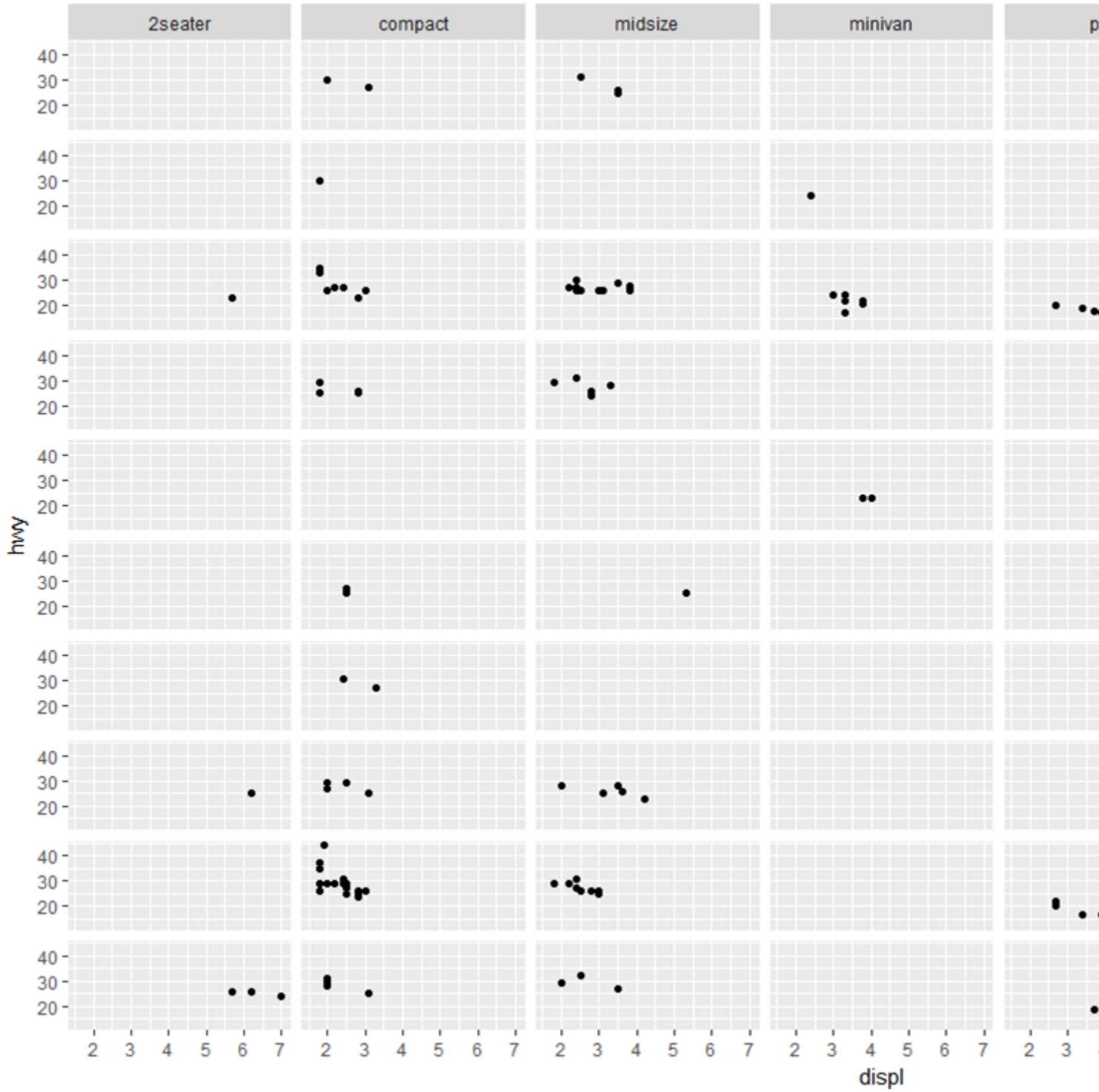
Отображение нескольких диаграмм на один столбец, несколько строк:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(class~.)
```



Отображение нескольких диаграмм в сетке по двум переменным:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #"row" parameter, then "column" parameter
```



Подготовьте свои данные для печати

`ggplot2` лучше всего работает с длинным фреймом данных. Следующие образцы данных, которые представляют цены на сладости в 20 разных дней, в формате, который описывается как широкий, поскольку каждая категория имеет столбец.

```
set.seed(47)
sweetsWide <- data.frame(date      = 1:20,
                         chocolate = runif(20, min = 2, max = 4),
                         iceCream  = runif(20, min = 0.5, max = 1),
                         candy     = runif(20, min = 1, max = 3))
```

```
head(sweetsWide)
##   date chocolate  iceCream    candy
## 1    1  3.953924 0.5890727 1.117311
## 2    2  2.747832 0.7783982 1.740851
## 3    3  3.523004 0.7578975 2.196754
## 4    4  3.644983 0.5667152 2.875028
## 5    5  3.147089 0.8446417 1.733543
## 6    6  3.382825 0.6900125 1.405674
```

Чтобы преобразовать `sweetsWide` в длинный формат для использования с `ggplot2` , можно использовать несколько полезных функций из базы R и пакеты `reshape2` , `data.table` и `tidyr` (в хронологическом порядке):

```
# reshape from base R
sweetsLong <- reshape(sweetsWide, idvar = 'date', direction = 'long',
                      varying = list(2:4), new.row.names = NULL, times = names(sweetsWide)[-
1])

# melt from 'reshape2'
library(reshape2)
sweetsLong <- melt(sweetsWide, id.vars = 'date')

# melt from 'data.table'
# which is an optimized & extended version of 'melt' from 'reshape2'
library(data.table)
sweetsLong <- melt(setDT(sweetsWide), id.vars = 'date')

# gather from 'tidyr'
library(tidyr)
sweetsLong <- gather(sweetsWide, sweet, price, chocolate:candy)
```

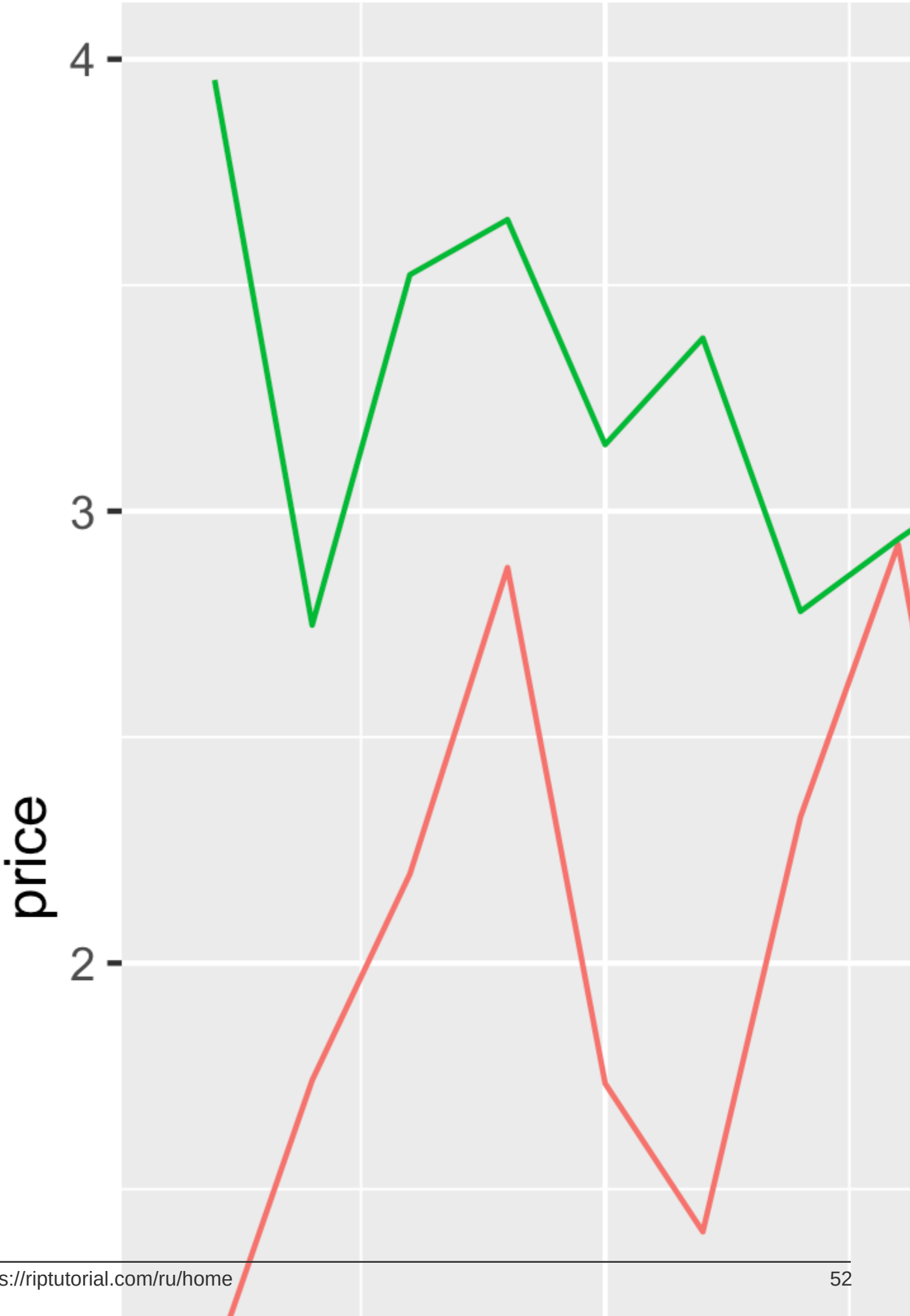
Все дают аналогичный результат:

```
head(sweetsLong)
##   date     sweet    price
## 1    1 chocolate 3.953924
## 2    2 chocolate 2.747832
## 3    3 chocolate 3.523004
## 4    4 chocolate 3.644983
## 5    5 chocolate 3.147089
## 6    6 chocolate 3.382825
```

См. Также Пересмотр данных между длинными и широкими формами для получения подробной информации о преобразовании данных между *длинным* и *широким* форматом.

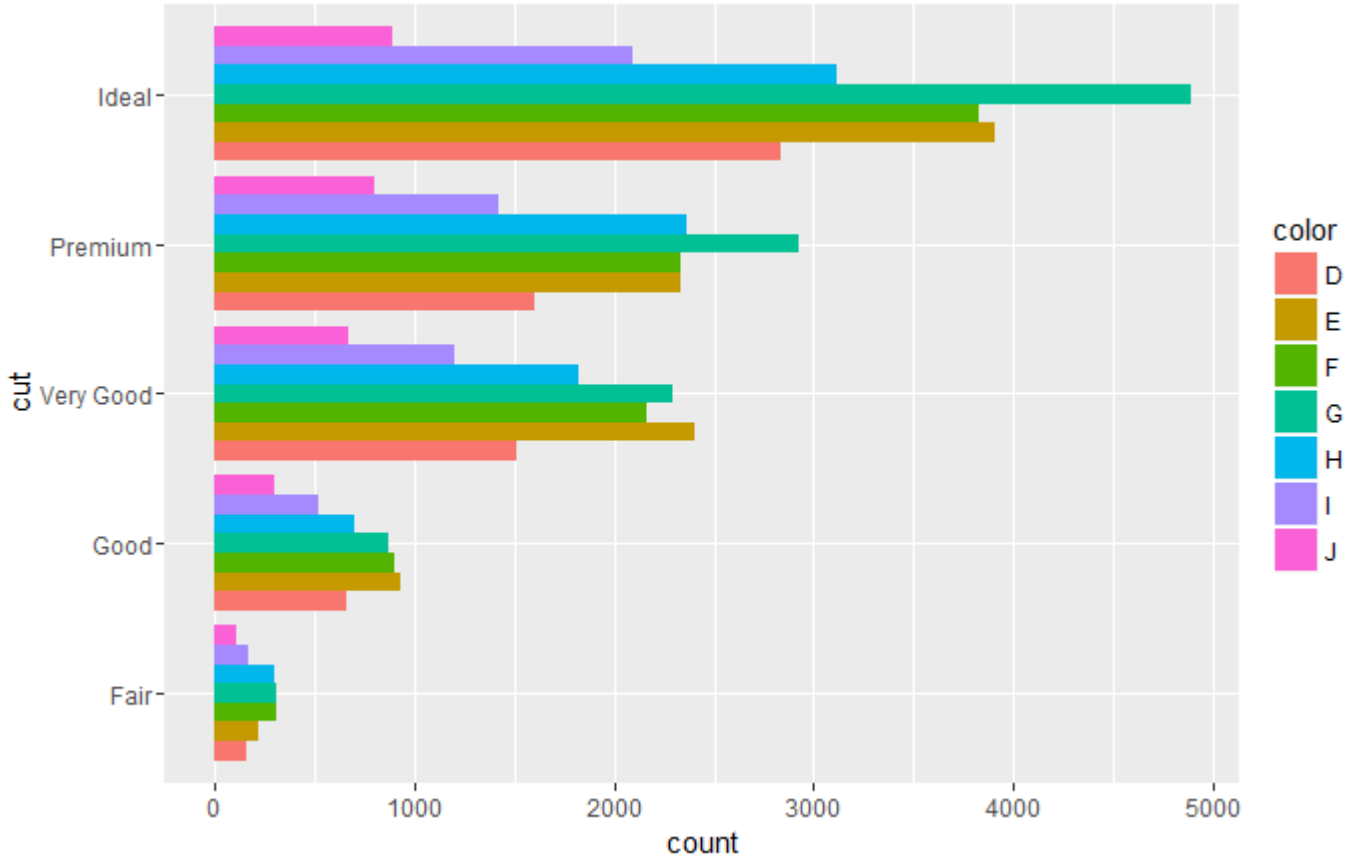
В результате `sweetsLong` имеет один столбец цен и один столбец, описывающий тип сладкого. Теперь заговор намного проще:

```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```


4
3
2
price

ord_flip ():

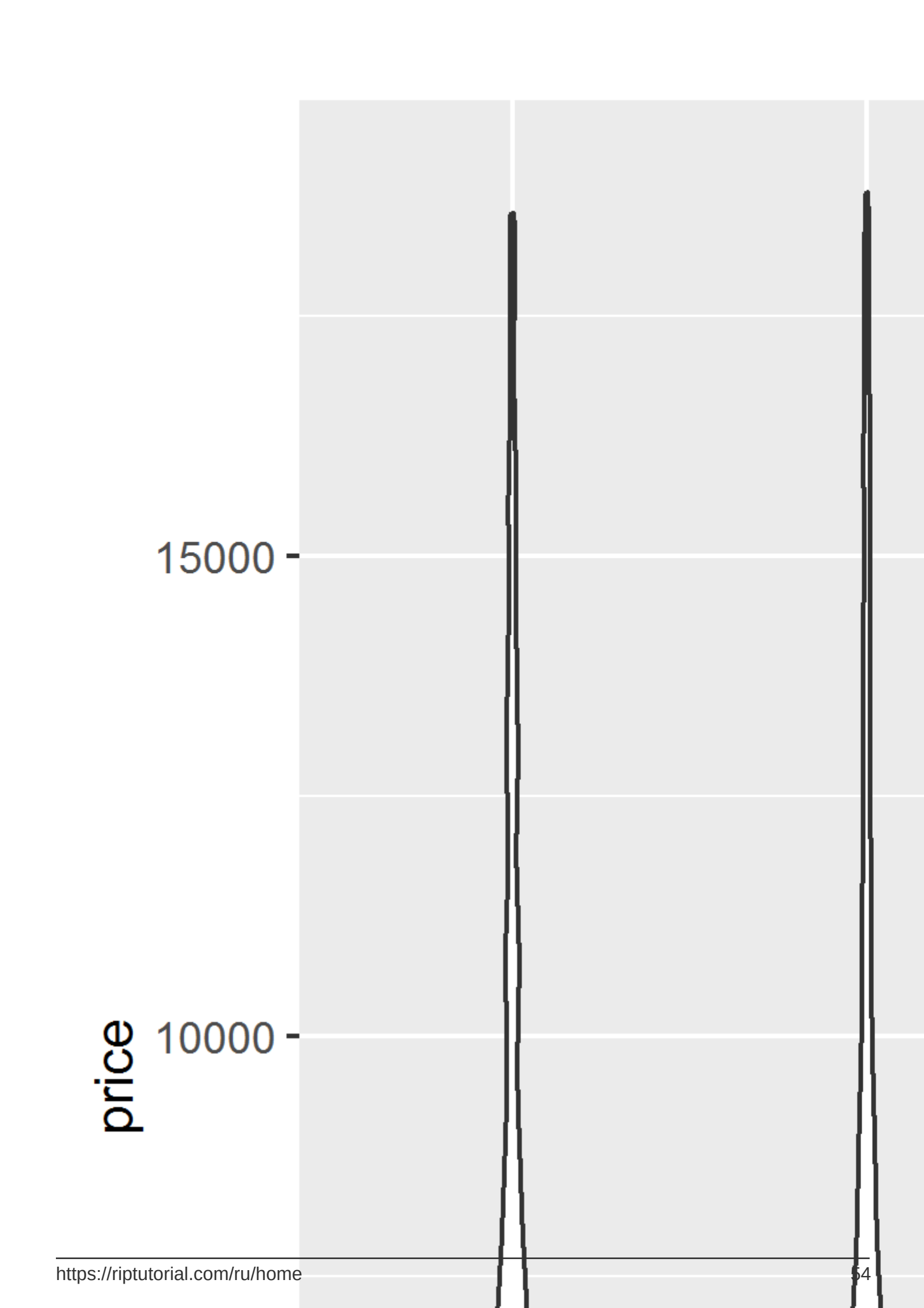
```
ggplot(data = diamonds, aes(x = cut, fill =color)) +
geom_bar(stat = "count", position = "dodge")+
coord_flip()
```



Скрипка

Графики скрипок - это оценки плотности ядра, отраженные в вертикальной плоскости. Они могут использоваться для визуализации нескольких дистрибутивов бок о бок, при этом зеркалирование помогает выявить любые различия.

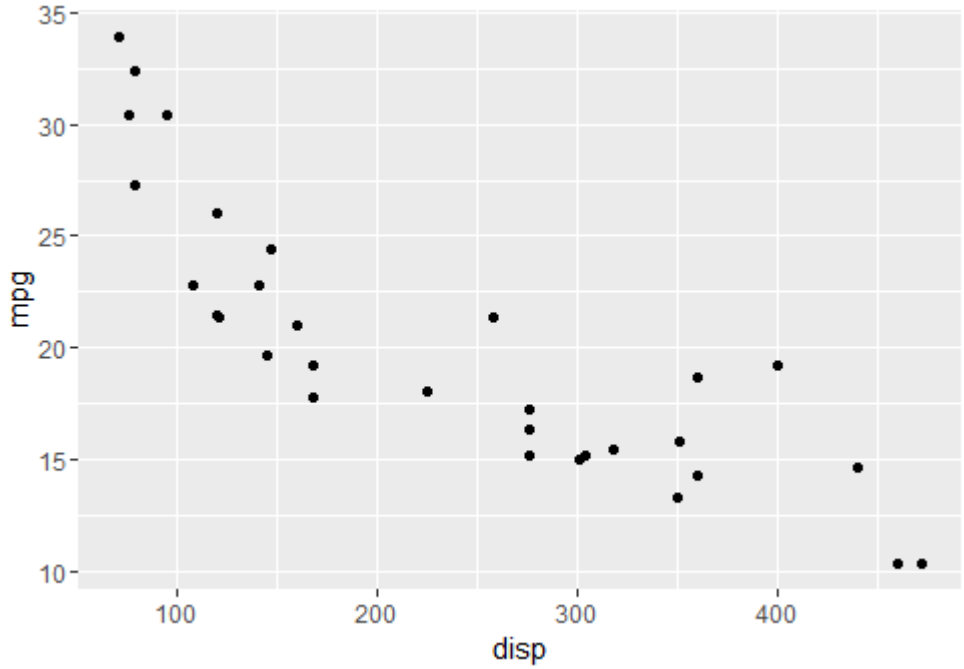
```
ggplot(diamonds, aes(cut, price)) +
  geom_violin()
```


15000
10000
price

пытаясь всегда выводить ваши данные, не требуя слишком больших спецификаций.

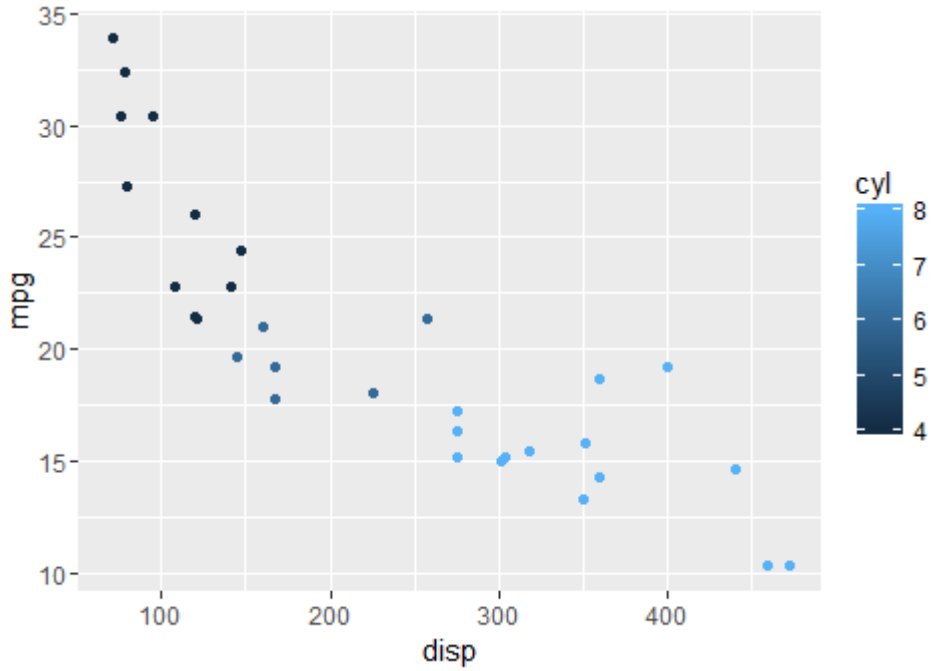
базовый qplot

```
qplot(x = disp, y = mpg, data = mtcars)
```



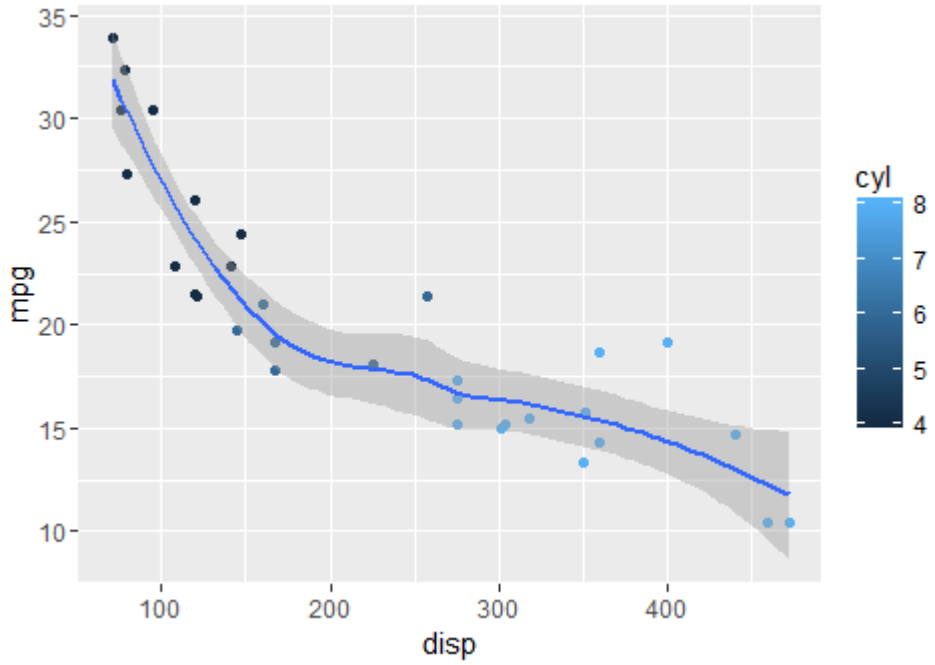
добавление цветов

```
qplot(x = disp, y = mpg, colour = cyl,data = mtcars)
```



добавление более плавного

```
qplot(x = disp, y = mpg, geom = c("point", "smooth"), data = mtcars)
```



Прочитайте ggplot2 онлайн: https://riptutorial.com/ru/r/topic/1334/ggplot2

глава 8: GPU-ускоренные вычисления

замечания

Для GPU-вычислений требуется «платформа», которая может подключаться и использовать оборудование. Двумя основными низкоуровневыми языками, которые достигают этого, являются CUDA и OpenCL. Первый требует установки проприетарного набора NVIDIA CUDA Toolkit и применим только на графических процессорах NVIDIA. Последний является как компанией (например, NVIDIA, AMD, Intel), так и аппаратно независимой (CPU или GPU), но требует установки SDK (набора для разработки программного обеспечения). Чтобы использовать GPU через R, вам необходимо сначала установить одну из этих программ.

Как только установлен CUDA Toolkit или OpenCL SDK, вы можете установить соответствующий пакет R. Почти все пакеты R GPU зависят от CUDA и ограничены графическими процессорами NVIDIA. Они включают:

1. gputools
2. cudaBayesreg
3. HiPLARM
4. gmatrix

В настоящее время существует только два пакета с поддержкой OpenCL

1. OpenCL - интерфейс от R до OpenCL
2. gpuR - библиотека общего назначения

Предупреждение. Установка может быть сложной для разных операционных систем с различными переменными окружающей среды и платформами GPU.

Examples

Объекты gpuR gpuMatrix

```
library(gpuR)

# gpuMatrix objects
X <- gpuMatrix(rnorm(100), 10, 10)
Y <- gpuMatrix(rnorm(100), 10, 10)

# transfer data to GPU when operation called
# automatically copied back to CPU
Z <- X %*% Y
```

Объекты gpuR vclMatrix

```
library(gpuR)

# vclMatrix objects
X <- vclMatrix(rnorm(100), 10, 10)
Y <- vclMatrix(rnorm(100), 10, 10)

# data always on GPU
# no data transfer
Z <- X %*% Y
```

Прочитайте GPU-ускоренные вычисления онлайн: https://riptutorial.com/ru/r/topic/4680/gpu-ускоренные-вычисления

глава 9: HashMaps

Examples

Среды как хэш-карты

*Примечание: в последующих проходах символы **hash-карта** и **хеш-таблица** используются взаимозаменяемо и относятся к одной и той же концепции , а именно к структуре данных, обеспечивающей эффективный поиск ключей с использованием внутренней хэш-функции.*

Вступление

Хотя R не предоставляет собственную структуру хеш-таблицы, подобную функциональность можно достичь, используя тот факт, что объект `environment` возвращенный из `new.env` (по умолчанию), предоставляет хешированные ключевые поисковые запросы. Следующие два утверждения эквивалентны, так как `hash` параметр по умолчанию имеет значение `TRUE` :

```
H <- new.env(hash = TRUE)
H <- new.env()
```

Кроме того, можно указать, что внутренняя хеш-таблица предварительно назначается определенным размером через параметр `size` , который имеет значение по умолчанию 29. Как и все другие объекты R, `environment` управляет собственной памятью и будет увеличиваться по мере необходимости , поэтому, хотя нет необходимости запрашивать значение по умолчанию для `size` , может быть небольшое преимущество в производительности при этом, **если** объект (в конечном счете) будет содержать очень большое количество элементов. Стоит отметить, что выделение дополнительного пространства по `size` само по себе не приводит к созданию объекта с большим объемом памяти:

```
object.size(new.env())
# 56 bytes

object.size(new.env(size = 10e4))
# 56 bytes
```

вставка

Вставка элементов может быть выполнена с использованием любого из методов `[[<-` или `$<-` заданных для класса `environment` , *но **не** с помощью назначения «одиночная скобка» (`[<-`)* :

```
H <- new.env()

H[["key"]] <- rnorm(1)

key2 <- "xyz"
H[[key2]] <- data.frame(x = 1:3, y = letters[1:3])

H$another_key <- matrix(rbinom(9, 1, 0.5) > 0, nrow = 3)

H["error"] <- 42
#Error in H["error"] <- 42 :
#  object of type 'environment' is not subsettable
```

Как и другие грани R, первый метод (`object[[key]] <- value`) обычно предпочитается второму (`object$key <- value`), потому что в первом случае вместо литерального значения может использоваться переменная (например, `key2` в примере выше).

Как правило, в случае реализации хэш-карт объект `environment` **не** будет хранить дубликаты ключей. Попытка вставить пару «ключ-значение» для существующего ключа заменит ранее сохраненное значение:

```
H[["key3"]] <- "original value"

H[["key3"]] <- "new value"

H[["key3"]]
#[1] "new value"
```

Поиск ключевых слов

Аналогичным образом, элементы могут быть доступны с помощью `[[` или `$` , но не с `[` :

```
H[["key"]]
#[1] 1.630631

H[[key2]]   ## assuming key2 <- "xyz"
#   x y
# 1 1 a
# 2 2 b
# 3 3 c

H$another_key
#       [,1]  [,2]  [,3]
# [1,]  TRUE  TRUE  TRUE
# [2,] FALSE FALSE FALSE
# [3,]  TRUE  TRUE  TRUE

H[1]
#Error in H[1] : object of type 'environment' is not subsettable
```

Проверка хэш-карты

Будучи просто обычной `environment` , хэш-карту можно проверить обычными способами:

```
names(H)
#[1] "another_key" "xyz"         "key"         "key3"

ls(H)
#[1] "another_key" "key"         "key3"        "xyz"

str(H)
#<environment: 0x7828228>

ls.str(H)
# another_key :  logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# key :  num 1.63
# key3 :  chr "new value"
# xyz : 'data.frame':    3 obs. of  2 variables:
#  $ x: int  1 2 3
#  $ y: chr  "a" "b" "c"
```

Элементы можно удалить с помощью `rm` :

```
rm(list = c("key", "key3"), envir = H)

ls.str(H)
# another_key :  logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# xyz : 'data.frame':    3 obs. of  2 variables:
#  $ x: int  1 2 3
#  $ y: chr  "a" "b" "c"
```

гибкость

Одним из основных преимуществ использования объектов `environment` как хеш-таблиц является их способность хранить практически любой тип объекта в качестве значения, *даже в других* *`environment`* :

```
H2 <- new.env()

H2[["a"]] <- LETTERS
H2[["b"]] <- as.list(x = 1:5, y = matrix(rnorm(10), 2))
H2[["c"]] <- head(mtcars, 3)
H2[["d"]] <- Sys.Date()
H2[["e"]] <- Sys.time()
H2[["f"]] <- (function() {
    H3 <- new.env()
    for (i in seq_along(names(H2))) {
        H3[[names(H2)[i]]] <- H2[[names(H2)[i]]]
    }
    H3
})()

ls.str(H2)
# a :  chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
#  $ : int 1
```

```
#  $ : int 2
#  $ : int 3
#  $ : int 4
#  $ : int 5
# c : 'data.frame':    3 obs. of  11 variables:
#  $ mpg : num  21 21 22.8
#  $ cyl : num  6 6 4
#  $ disp: num  160 160 108
#  $ hp  : num  110 110 93
#  $ drat: num  3.9 3.9 3.85
#  $ wt  : num  2.62 2.88 2.32
#  $ qsec: num  16.5 17 18.6
#  $ vs  : num  0 0 1
#  $ am  : num  1 1 1
#  $ gear: num  4 4 4
#  $ carb: num  4 4 1
# d :  Date[1:1], format: "2016-08-03"
# e :  POSIXct[1:1], format: "2016-08-03 19:25:14"
# f : <environment: 0x91a7cb8>

ls.str(H2$f)
# a :  chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
#  $ : int 1
#  $ : int 2
#  $ : int 3
#  $ : int 4
#  $ : int 5
# c : 'data.frame':    3 obs. of  11 variables:
#  $ mpg : num  21 21 22.8
#  $ cyl : num  6 6 4
#  $ disp: num  160 160 108
#  $ hp  : num  110 110 93
#  $ drat: num  3.9 3.9 3.85
#  $ wt  : num  2.62 2.88 2.32
#  $ qsec: num  16.5 17 18.6
#  $ vs  : num  0 0 1
#  $ am  : num  1 1 1
#  $ gear: num  4 4 4
#  $ carb: num  4 4 1
# d :  Date[1:1], format: "2016-08-03"
# e :  POSIXct[1:1], format: "2016-08-03 19:25:14"
```

Ограничения

Одним из основных ограничений использования объектов `environment` качестве хэш-карт является то, что в отличие от многих аспектов R векторизация не поддерживается для поиска / вставки элемента:

```
names(H2)
#[1] "a" "b" "c" "d" "e" "f"

H2[[c("a", "b")]]
#Error in H2[[c("a", "b")]] :
#  wrong arguments for subsetting an environment
```

```
Keys <- c("a", "b")
H2[[Keys]]
#Error in H2[[Keys]] : wrong arguments for subsetting an environment
```

В зависимости от характера данных, хранящихся в объекте, может быть возможно использовать `vapply` или `list2env` для назначения сразу нескольких элементов:

```
E1 <- new.env()
invisible({
    vapply(letters, function(x) {
        E1[[x]] <- rnorm(1)
        logical(0)
    }, FUN.VALUE = logical(0))
})

all.equal(sort(names(E1)), letters)
#[1] TRUE

Keys <- letters
E2 <- list2env(
    setNames(
        as.list(rnorm(26)),
        nm = Keys),
    envir = NULL,
    hash = TRUE
)

all.equal(sort(names(E2)), letters)
#[1] TRUE
```

Ни один из вышеперечисленных не является особенно кратким, но может быть предпочтительнее использовать цикл `for` и т. Д., Когда число пар ключ-значение велико.

пакет: хэш

Хэш-пакет предлагает хэш-структуру в R. Однако, сроки определения времени для обеих вставок и чтения сравниваются с неблагоприятными условиями использования в качестве хеша. Эта документация просто подтверждает ее существование и предоставляет примерный временной код ниже по вышеуказанным причинам. Не существует определенного случая, когда хеш является подходящим решением в R-коде сегодня.

Рассматривать:

```
# Generic unique string generator
unique_strings <- function(n){
    string_i <- 1
    string_len <- 1
    ans <- character(n)
    chars <- c(letters,LETTERS)
    new_strings <- function(len,pfx){
    for(i in 1:length(chars)){
        if (len == 1){
        ans[string_i] <<- paste(pfx,chars[i],sep='')
        string_i <<- string_i + 1
```

```
        } else {
        new_strings(len-1,pfx=paste(pfx,chars[i],sep=''))
        }
        if (string_i > n) return ()
    }
    }
    while(string_i <= n){
    new_strings(string_len,'')
    string_len <- string_len + 1
    }
    sample(ans)
}

# Generate timings using an enviornment
timingsEnv <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    ht1 <- new.env(hash=TRUE)
    lapply(strings, function(s){ ht1[[s]] <<- 0L})
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (j in 1:i) ht1[[strings[j]]]==0L)[3]),
    type = c('1_hashedEnv')
    )
})

timingsHash <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    ht <- hash::hash()
    lapply(strings, function(s) ht[[s]] <<- 0L)
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (j in 1:i) ht[[strings[j]]]==0L)[3]),
    type = c('3_stringHash')
    )
})
```

пакет: listenv

Хотя `package:listenv` реализует список-подобный интерфейс для сред, его производительность по сравнению с средами для хэш-подобных целей невелика при хэш-поиске . Однако, если индексы являются числовыми, это может быть довольно быстро при поиске. Однако у них есть другие преимущества, например совместимость с `package:future` . Покрытие этого пакета для этой цели выходит за рамки текущей темы. Однако приведенный здесь код синхронизации может использоваться в сочетании с примером для пакета: хэш для таймингов записи.

```
timingsListEnv <- plyr::adply(2^(10:15),.mar=1,.fun=function(i){
    strings <- unique_strings(i)
    le <- listenv::listenv()
    lapply(strings, function(s) le[[s]] <<- 0L)
    data.frame(
    size=c(i,i),
    seconds=c(
        system.time(for (k in 1:i) le[[k]]==0L)[3]),
```

```
    type = c('2_numericListEnv')
    )
})
```

Прочитайте HashMaps онлайн: https://riptutorial.com/ru/r/topic/5179/hashmaps

глава 10: I / O для внешних таблиц (Excel, SAS, SPSS, Stata)

Examples

Импорт данных с помощью rio

Очень простой способ импортировать данные из многих распространенных форматов файлов - это rio . Этот пакет предоставляет функцию `import()` которая обертывает многие часто используемые функции импорта данных, обеспечивая тем самым стандартный интерфейс. Он работает просто путем передачи имени файла или URL-адреса для `import()` :

```
import("example.csv")        # comma-separated values
import("example.tsv")        # tab-separated values
import("example.dta")        # Stata
import("example.sav")        # SPSS
import("example.sas7bdat")   # SAS
import("example.xlsx")       # Excel
```

`import()` также может считывать из сжатых каталогов, URL-адресов (HTTP или HTTPS) и в буфер обмена. Полный список всех поддерживаемых форматов файлов доступен в репозитории rio-пакета github .

Можно даже указать некоторые дополнительные параметры, относящиеся к определенному файловому формату, который вы пытаетесь прочитать, передавая их непосредственно в функции `import()` :

```
import("example.csv", format = ",") #for csv file where comma is used as separator
import("example.csv", format = ";") #for csv file where semicolon is used as separator
```

Импорт файлов Excel

Существует несколько пакетов R для чтения файлов excel, каждый из которых использует разные языки или ресурсы, как описано в следующей таблице:

R пакет	Пользы
XLSX	Джава
XLconnect	Джава
openxlsx	C ++

R пакет	Польз
readxl	C ++
RODBC	ODBC
GData	Perl

Для пакетов, использующих Java или ODBC, важно знать подробности о вашей системе, потому что у вас могут быть проблемы с совместимостью в зависимости от вашей версии R и ОС. Например, если вы используете R 64 бита, вы также должны иметь Java 64 бит для использования `xlsx` или `XLconnect` .

Ниже приведены некоторые примеры чтения файлов excel с каждым пакетом. Обратите внимание, что многие из пакетов имеют одинаковые или очень похожие имена функций. Поэтому полезно указать пакет явно, например, `package::function` . Пакет `openxlsx` требует предварительной установки RTools.

Чтение файлов excel с пакетом xlsx

```
library(xlsx)
```

Для импорта требуется индекс или имя листа.

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)

xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

Чтение файлов Excel с помощью пакета XLconnect

```
library(XLConnect)
wb <- XLConnect::loadWorkbook("Book1.xlsx")

# Either, if Book1.xlsx has a sheet called "Sheet1":
sheet1 <- XLConnect::readWorksheet(wb, "Sheet1")
# Or, more generally, just get the first sheet in Book1.xlsx:
sheet1 <- XLConnect::readWorksheet(wb, getSheets(wb)[1])
```

`XLConnect` автоматически импортирует предварительно определенные стили ячеек Excel, встроенные в `Book1.xlsx` . Это полезно, если вы хотите отформатировать объект своей книги и экспортировать полностью отформатированный документ Excel. Во-первых, вам нужно будет создать желаемые форматы ячеек в `Book1.xlsx` и сохранить их, например, как `myHeader` , `myBody` и `myPcts` . Затем, после загрузки книги в `R` (см. Выше):

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

Теперь стили ячеек сохраняются в вашей среде `R` Чтобы назначить стили ячеек определенным диапазонам ваших данных, вам необходимо определить диапазон, а затем назначить стиль:

```
Headerrange <- expand.grid(row = 1, col = 1:8)
Bodyrange <- expand.grid(row = 2:6, col = c(1:5, 8))
Pctrange <- expand.grid(row = 2:6, col = c(6, 7))

XLConnect::setCellStyle(wb, sheet = "sheet1", row = Headerrange$row,
             col = Headerrange$col, cellstyle = Headerstyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Bodyrange$row,
             col = Bodyrange$col, cellstyle = Bodystyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Pctrange$row,
             col = Pctrange$col, cellstyle = Pctsstyle)
```

Обратите внимание, что `XLConnect` прост, но в форматировании может стать очень медленным. `openxlsx` предлагает гораздо более быстрый, но более громоздкий вариант форматирования.

Чтение файлов excel с пакетом openxlsx

Файлы Excel можно импортировать с помощью пакета `openxlsx`

```
library(openxlsx)

openxlsx::read.xlsx("spreadsheet1.xlsx", colNames=TRUE, rowNames=TRUE)

#colNames: If TRUE, the first row of data will be used as column names.
#rowNames: If TRUE, first column of data will be used as row names.
```

Лист, который следует читать в R, можно выбрать либо путем указания его позиции в аргументе `sheet` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

или объявив свое имя:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

Кроме того, `openxlsx` может определять столбцы дат в считываемом листе. Чтобы обеспечить автоматическое обнаружение дат, аргумент `detectDates` должен быть установлен в `TRUE` :

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

Чтение файлов excel с помощью пакета readxl

Файлы Excel можно импортировать в виде кадра данных в `R` используя пакет `readxl` .

```
library(readxl)
```

Он может читать файлы `.xls` и `.xlsx` .

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

Лист, который нужно импортировать, может быть указан по номеру или имени.

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

Аргумент `col_names = TRUE` устанавливает первую строку как имена столбцов.

```
 readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

Аргумент `col_types` может использоваться для указания типов столбцов в данных как вектор.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
                   col_types = c("text", "date", "numeric", "numeric"))
```

Чтение файлов excel с пакетом RODBC

Файлы Excel могут быть прочитаны с использованием драйвера ODBC Excel, который взаимодействует с Windows Database Engine Engine (ACE), ранее JET. С пакетом RODBC R может подключаться к этому драйверу и напрямую запрашивать книги. Предполагается, что рабочие листы поддерживают заголовки столбцов в первой строке с данными в организованных столбцах аналогичных типов. **ПРИМЕЧАНИЕ.** Этот подход ограничивается только машинами Windows / ПК, поскольку JET / ACE установлены .dll-файлы и недоступны в других операционных системах.

```
library(RODBC)

xlconn <- odbcDriverConnect('Driver={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)};
                             DBQ=C:\\Path\\To\\Workbook.xlsx')

df <- sqlQuery(xlconn, "SELECT * FROM [SheetName$]")
```

```
close(xlconn)
```

В этом подходе к соединению с движком SQL в Excel можно запросить похожие таблицы базы данных, включая операции `JOIN` и `UNION` . Синтаксис следует диалекту JET / ACE SQL. **ПРИМЕЧАНИЕ.** Только операторы DML для доступа к данным, в частности `SELECT` могут быть запущены на книгах, считаются не обновляемыми запросами.

```
joindf <-  sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
                             INNER JOIN [Sheet2$] t2
                             ON t1.[ID] = t2.[ID]")

uniondf <-  sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
                              UNION
                              SELECT * FROM [Sheet2$]")
```

Даже другие книги могут запрашиваться с одного и того же ODBC-канала, указывающего на текущую книгу:

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
                                 [Excel 12.0 Xml;HDR=Yes;
                                 Database=C:\\Path\\To\\Other\\Workbook.xlsx].[Sheet1$];")
```

Чтение файлов excel с пакетом gdata

пример здесь

Чтение и запись файлов Stata, SPSS и SAS

Пакеты `foreign` and `haven` могут использоваться для импорта и экспорта файлов из множества других статистических пакетов, таких как Stata, SPSS и SAS и соответствующее программное обеспечение. Существует функция `read` для каждого из поддерживаемых типов данных для импорта файлов.

```
# loading the packages
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

Некоторые примеры для наиболее распространенных типов данных:

```
# reading Stata files with `foreign`
read.dta("path\to\your\data")
# reading Stata files with `haven`
read_dta("path\to\your\data")
```

`foreign` пакет может читать файлы stata (.dta) для версий Stata 7-12. Согласно странице

разработки, `read.dta` более или менее заморожен и не будет обновляться для чтения в версиях 13+. Для более поздних версий Stata вы можете использовать пакет `readstata13` или `haven` . Для `readstata13` файлы

```
# reading recent Stata (13+) files with `readstata13`
read.dta13("path\to\your\data")
```

Для чтения в файлах SPSS и SAS

```
# reading SPSS files with `foreign`
read.spss("path\to\your\data.sav", to.data.frame = TRUE)
# reading SPSS files with `haven`
read_spss("path\to\your\data.sav")
read_sav("path\to\your\data.sav")
read_por("path\to\your\data.por")

# reading SAS files with `foreign`
read.ssd("path\to\your\data")
# reading SAS files with `haven`
read_sas("path\to\your\data")
# reading native SAS files with `Hmisc`
sas.get("path\to\your\data")   #requires access to saslib
# Reading SA XPORT format ( *.XPT ) files
sasxport.get("path\to\your\data.xpt")  # does not require access to SAS executable
```

Пакет `SAScii` предоставляет функции, которые будут принимать код импорта SAS SET и создавать текстовый файл, который может обрабатываться с помощью `read.fwf` . Он оказался очень надежным для импорта больших публичных наборов данных. Поддержка осуществляется по адресу https://github.com/ajdamico/SAScii

Чтобы экспортировать кадры данных в другие статистические пакеты, вы можете использовать функции `write.foreign()` . Это будет записывать 2 файла, один из которых содержит данные, а другой - инструкции, которые другой пакет должен читать.

```
# writing to Stata, SPSS or SAS files with `foreign`
write.foreign(dataframe, datafile, codefile,
              package = c("SPSS", "Stata", "SAS"), ...)
write.foreign(dataframe, "path\to\data\file", "path\to\instruction\file", package = "Stata")

# writing to Stata files with `foreign`
write.dta(dataframe, "file", version = 7L,
          convert.dates = TRUE, tz = "GMT",
          convert.factors = c("labels", "string", "numeric", "codes"))

# writing to Stata files with `haven`
write_dta(dataframe, "path\to\your\data")

# writing to Stata files with `readstata13`
save.dta13(dataframe, file, data.label = NULL, time.stamp = TRUE,
  convert.factors = TRUE, convert.dates = TRUE, tz = "GMT",
  add.rownames = FALSE, compress = FALSE, version = 117,
  convert.underscore = FALSE)

# writing to SPSS files with `haven`
```

```
write_sav(dataframe, "path\to\your\data")
```

Файл, хранящийся в SPSS, также может быть прочитан с помощью `read.spss` таким образом:

```
 foreign::read.spss('data.sav', to.data.frame=TRUE, use.value.labels=FALSE,
                    use.missings=TRUE, reencode='UTF-8')
# to.data.frame if TRUE: return a data frame
# use.value.labels if TRUE: convert variables with value labels into R factors with those
levels
# use.missings if TRUE: information on user-defined missing values will used to set the
corresponding values to NA.
# reencode character strings will be re-encoded to the current locale. The default, NA, means
to do so in a UTF-8 locale, only.
```

Импорт или экспорт файла пера

Перо представляет собой реализацию Apache Arrow, предназначенную для хранения кадров данных в агностическом языке, сохраняя при этом метаданные (например, классы дат), увеличивая взаимодействие между Python и R. Чтение перьевого файла приведет к получению, а не стандарту data.frame.

```
library(feather)

path <- "filename.feather"
df <- mtcars

write_feather(df, path)

df2 <- read_feather(path)

head(df2)
##  A tibble: 6 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21.0     6   160   110  3.90 2.620 16.46     0     1     4     4
## 2  21.0     6   160   110  3.90 2.875 17.02     0     1     4     4
## 3  22.8     4   108    93  3.85 2.320 18.61     1     1     4     1
## 4  21.4     6   258   110  3.08 3.215 19.44     1     0     3     1
## 5  18.7     8   360   175  3.15 3.440 17.02     0     0     3     2
## 6  18.1     6   225   105  2.76 3.460 20.22     1     0     3     1

head(df)
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

В текущей документации содержится следующее предупреждение:

> Примечание для пользователей: перо следует рассматривать как альфа-

программное обеспечение. В частности, формат файла, скорее всего, будет развиваться в течение следующего года. Не используйте перо для длительного хранения данных.

Прочитайте I / O для внешних таблиц (Excel, SAS, SPSS, Stata) онлайн:
https://riptutorial.com/ru/r/topic/5536/i---o-для-внешних-таблиц--excel--sas--spss--stata-

глава 11: I / O для географических данных (шейп-файлы и т. Д.)

Вступление

См. Также Введение в географические карты и ввод и вывод

Examples

Импорт и экспорт Shapefiles

С пакетом `rgdal` можно импортировать и экспортировать shapfiles с R. Функция `readOGR` может использоваться для импорта шейп-файлов. Если вы хотите импортировать файл, например, ArcGIS, первым аргументом `dsn` является путь к папке, содержащей шейп-файл. `layer` - это имя шейп-файла без окончания файла (просто `map` а не `map.shp`).

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

Для экспорта шейп- `writeOGR` функция `writeOGR` . Первый аргумент - это пространственный объект, созданный в R. `dsn` и `layer` такие же, как и выше. Обязательным аргументом 4. является драйвер, используемый для создания шейп-файла. Функция `ogrDrivers()` содержит список всех доступных драйверов. Если вы хотите экспортировать шейп-файл в ArcGis или QGis, вы можете использовать `driver = "ESRI Shapefile"` .

```
writeOGR(Rmap, dsn = "path\to\the\folder\containing\the\shapefile", layer = "map",
         driver = "ESRI Shapefile" )
```

Пакет `tmap` имеет очень удобную функцию `read_shape()` , которая является оберткой для `rgdal::reagOGR()` . Функция `read_shape()` упрощает процесс импорта шейп- `read_shape()` . С `tmap` стороны, `tmap` довольно тяжелый.

Прочитайте I / O для географических данных (шейп-файлы и т. Д.) онлайн: https://riptutorial.com/ru/r/topic/5538/i---o-для-географических-данных--шейп-файлы-и-т--д--

глава 12: I / O для двоичного формата R

Examples

Файлы RDS и RData (Rda)

`.rds` и `.Rdata` (также известные как `.rda`) файлы могут быть использованы для хранения объектов R в формате, родном R. Существует несколько преимуществ сохранения этого способа, если они контрастируют с неродными подходами к хранению, например `write.table` :

- Быстрее восстанавливать данные до R
- Он сохраняет R специфическую информацию, закодированную в данных (например, атрибуты, типы переменных и т. Д.).

`saveRDS` / `readRDS` обрабатывает только один объект R. Тем не менее, они более гибкие, чем подход с множеством объектов, поскольку имя объекта восстановленного объекта не обязательно совпадает с именем объекта при сохранении объекта.

Например, используя файл .rds, сохраняя набор данных `iris` мы будем использовать:

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

Чтобы загрузить данные:

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

Чтобы сохранить несколько объектов, мы можем использовать `save()` и выводить как `.Rdata` .

Например, чтобы сохранить 2 фрейма данных: радужная оболочка и автомобили

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

Загрузить:

```
load("myIrisAndCarsData.Rdata")
```

Enviromments

Функции `save` и `load` позволяют нам указать среду, в которой будет размещаться объект:

```
save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo  <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars

save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo  <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars
```

Прочитайте I / O для двоичного формата R онлайн: https://riptutorial.com/ru/r/topic/5540/i---o-для-двоичного-формата-r

глава 13: I / O для таблиц базы данных

замечания

Специализированные пакеты

- RMySQL
- RODBC

Examples

Чтение данных из баз данных MySQL

генеральный

Используя пакет RMySQL, мы можем легко запросить MySQL, а также базы данных MariaDB и сохранить результат в фреймворке R:

```
library(RMySQL)

mydb <- dbConnect(MySQL(), user='user', password='password', dbname='dbname',host='127.0.0.1')

queryString <- "SELECT * FROM table1 t1 JOIN table2 t2 on t1.id=t2.id"
query <- dbSendQuery(mydb, queryString)
data <- fetch(query, n=-1) # n=-1 to return all results
```

Использование ограничений

Также можно определить предел, например, получить только первые 100 000 строк. Для этого просто измените SQL-запрос на требуемый лимит. В упомянутом пакете будут рассмотрены эти варианты. Пример:

```
queryString <- "SELECT * FROM table1 limit 100000"
```

Чтение данных из баз данных MongoDB

Чтобы загрузить данные из базы данных MongoDB в фреймворк R, используйте библиотеку MongoLite :

```
# Use MongoLite library:
```

```
#install.packages("mongolite")
library(jsonlite)
library(mongolite)

# Connect to the database and the desired collection as root:
db <- mongo(collection = "Tweets", db = "TweetCollector", url =
"mongodb://USERNAME:PASSWORD@HOSTNAME")

# Read the desired documents i.e. Tweets inside one dataframe:
documents <- db$find(limit = 100000, skip = 0, fields = '{ "_id" : false, "Text" : true }')
```

Код подключается к серверу `HOSTNAME` как `USERNAME` с помощью `PASSWORD` , пытается открыть базу данных `TweetCollector` и прочитать коллекцию `Tweets` . Запрос пытается прочитать поле, т.е. столбец `Text` .

Результатом является dataframe с столбцами в качестве заданного набора данных. В этом примере в кадре данных содержится столбец `Text` , например `documents$Text` .

Прочитайте I / O для таблиц базы данных онлайн: https://riptutorial.com/ru/r/topic/5537/i---o-для-таблиц-базы-данных

глава 14: JSON

Examples

JSON в / из объектов R

Пакет `jsonlite` - это быстрый анализатор и генератор JSON, оптимизированный для статистических данных и Интернета. Две основные функции, используемые для чтения и записи JSON, от JSON `fromJSON()` и `toJSON()` соответственно, и предназначены для работы с `vectors` , `matrices` и `data.frames` и потоками JSON из Интернета.

Создайте массив JSON из вектора и наоборот

```
library(jsonlite)

## vector to JSON
toJSON(c(1,2,3))
# [1,2,3]

fromJSON('[1,2,3]')
# [1] 1 2 3
```

Создайте именованный массив JSON из списка, и наоборот

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec":[1,2,3]}

fromJSON('{"myVec":[1,2,3]}')
# $myVec
# [1] 1 2 3
```

Более сложные структуры списков

```
## list structures
lst <- list(a = c(1,2,3),
            b = list(letters[1:6]))

toJSON(lst)
# {"a":[1,2,3],"b":[["a","b","c","d","e","f"]]}


fromJSON('{"a":[1,2,3],"b":[["a","b","c","d","e","f"]]} ')
# $a
# [1] 1 2 3
#
# $b
# [,1] [,2] [,3] [,4] [,5] [,6]
# [1,] "a"  "b"  "c"  "d"  "e"  "f"
```

Создайте JSON из `data.frame` и наоборот

```
## converting a data.frame to JSON
df <- data.frame(id = seq_along(1:10),
                 val = letters[1:10])

toJSON(df)
#
[{"id":1,"val":"a"},{"id":2,"val":"b"},{"id":3,"val":"c"},{"id":4,"val":"d"},{"id":5,"val":"e"},{"id":6


## reading a JSON string
fromJSON('[{"id":1,"val":"a"},{"id":2,"val":"b"},{"id":3,"val":"c"},{"id":4,"val":"d"},{"id":5,"val":"e

#     id val
# 1   1   a
# 2   2   b
# 3   3   c
# 4   4   d
# 5   5   e
# 6   6   f
# 7   7   g
# 8   8   h
# 9   9   i
# 10 10   j
```

Прочтите JSON прямо из Интернета

```
## Reading JSON from URL
googleway_issues <- fromJSON("https://api.github.com/repos/SymbolixAU/googleway/issues")

googleway_issues$url
# [1] "https://api.github.com/repos/SymbolixAU/googleway/issues/20"
"https://api.github.com/repos/SymbolixAU/googleway/issues/19"
# [3] "https://api.github.com/repos/SymbolixAU/googleway/issues/14"
"https://api.github.com/repos/SymbolixAU/googleway/issues/11"
# [5] "https://api.github.com/repos/SymbolixAU/googleway/issues/9"
"https://api.github.com/repos/SymbolixAU/googleway/issues/5"
# [7] "https://api.github.com/repos/SymbolixAU/googleway/issues/2"
```

Прочитайте JSON онлайн: https://riptutorial.com/ru/r/topic/2460/json

глава 15: lubridate

Синтаксис

- ymd_hms (..., quiet = FALSE, tz = "UTC", locale = Sys.getlocale ("LC_TIME"))
- теперь (tzone = "")
- интервал (начало, конец, tzone = attr (начало, «tzone»))
- duration (num = NULL, units = "seconds", ...)
- период (num = NULL, units = "second", ...)

замечания

Чтобы установить пакет из CRAN:

```
install.packages("lubridate")
```

Чтобы установить версию разработчика из Github:

```
library(devtools)
# dev mode allows testing of development packages in a sandbox, without interfering
# with the other packages you have installed.
dev_mode(on=T)
install_github("hadley/lubridate")
dev_mode(on=F)
```

Чтобы получить виньетки на пакет lubridate:

```
vignette("lubridate")
```

Чтобы получить справку о некоторой функции `foo` :

```
help(foo)      # help about function foo
?foo           # same thing

# Example
# help("is.period")
# ?is.period
```

Чтобы получить примеры для функции `foo` :

```
example("foo")

# Example
# example("interval")
```

Examples

Синхронизация дат и времени от строк с помощью lubridate

Пакет `lubridate` предоставляет удобные функции для форматирования объектов даты и даты и времени из символьных строк. Функции являются перестановками

Письмо	Элемент для анализа	Базовый эквивалент R
Y	год	`%y` , `%Y`
m (c y и d)	месяц	`%m` , `%b` , `%h` , `%B`
d	день	`%d` , `%e`
час	час	`%H` , `%I%p`
m (c h и s)	минут	`%M`
s	секунд	`%S`

например `ymd()` для разбора даты с годом, за которым следует месяц, за которым следует день, например `"2016-07-22"` или `ymd_hms()` для разбора даты и времени в порядке год, месяц, день, часы, минуты, секунд, например, `"2016-07-22 13:04:47"` .

Функции могут распознавать большинство разделителей (например, `/` , `-` и пробелы) без дополнительных аргументов. Они также работают с непоследовательными разделителями.

Даты

Функции даты возвращают объект класса `Date` .

```
library(lubridate)

mdy(c(' 07/02/2016 ', '7 / 03 / 2016', ' 7 / 4 / 16 '))
## [1] "2016-07-02" "2016-07-03" "2016-07-04"

ymd(c("20160724","2016/07/23","2016-07-25"))    # inconsistent separators
## [1] "2016-07-24" "2016-07-23" "2016-07-25"
```

DateTimes

Полезные функции

DateTimes может быть проанализирован с помощью `ymd_hms` варианты , включая `ymd_hm` и `ymd_h` . Все функции datetime могут принимать аргумент `tz` timezone, аналогичный аргументу `as.POSIXct` или `strptime` , но по умолчанию используется `"UTC"` вместо локального часового пояса.

Функции datetime возвращают объект класса `POSIXct` .

```
x <- c("20160724 130102","2016/07/23 14:02:01","2016-07-25 15:03:00")
ymd_hms(x, tz="EST")
## [1] "2016-07-24 13:01:02 EST" "2016-07-23 14:02:01 EST"
## [3] "2016-07-25 15:03:00 EST"

ymd_hms(x)
## [1] "2016-07-24 13:01:02 UTC" "2016-07-23 14:02:01 UTC"
## [3] "2016-07-25 15:03:00 UTC"
```

Функции анализатора

`lubridate` также включает три функции для синтаксического анализа datetimes с помощью строки форматирования, например `as.POSIXct` или `strptime` :

функция	Класс вывода	Форматирование строк
`parse_date_time`	POSIXct	Гибкое. Будет принимать `strptime` стиль с `%` или `lubridate` datetime имя функции, например, `"ymd hms"` . Примут вектор заказов для гетерогенных данных и угадают, что подходит.
`parse_date_time2`	По умолчанию POSIXct; if `lt = TRUE` , POSIXlt	Строгий. Принимает только токены `strptime` (с или без `%`) из ограниченного набора.
`fast_strptime`	По умолчанию POSIXlt; if `lt = FALSE` , POSIXct	Строгий. Принимает только `%` -delimited `strptime` маркеров с разделителями (`-` , `/` , `:` , и т.д.) из ограниченного набора.

```
x <- c('2016-07-22 13:04:47', '07/22/2016 1:04:47 pm')

parse_date_time(x, orders = c('mdy Imsp', 'ymd hms'))
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 13:04:47 UTC"

x <- c('2016-07-22 13:04:47', '2016-07-22 14:47:58')

parse_date_time2(x, orders = 'Ymd HMS')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"
```

```
fast_strptime(x, format = '%Y-%m-%d %H:%M:%S')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"
```

`parse_date_time2` и `fast_strptime` используют быстрый анализатор C для повышения эффективности.

См. `?parse_date_time` для форматирования токенов.

Синхронизация даты и времени в lubridate

Lubridate предоставляет `ymd()` ряд функций для синтаксического анализа символьных строк в датах. Буквы y, m и d соответствуют элементам года, месяца и дня даты.

```
mdy("07-21-2016")                 # Returns Date

## [1] "2016-07-21"

mdy("07-21-2016", tz = "UTC")     # Returns a vector of class POSIXt

## "2016-07-21 UTC"

dmy("21-07-2016")                 # Returns Date

## [1] "2016-07-21"

dmy(c("21.07.2016", "22.07.2016")) # Returns vector of class Date

## [1] "2016-07-21" "2016-07-22"
```

Манипулирование датой и временем в lubridate

```
date <- now()
date
## "2016-07-22 03:42:35 IST"

year(date)
## 2016

minute(date)
## 42

wday(date, label = T, abbr = T)
# [1] Fri
# Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

day(date) <- 31
## "2016-07-31 03:42:35 IST"

# If an element is set to a larger value than it supports, the difference
#  will roll over into the next higher element
day(date) <- 32
## "2016-08-01 03:42:35 IST"
```

Мгновенные

Момент - это конкретный момент времени. Любой объект времени, который относится к моменту времени, распознается как мгновенный. Чтобы проверить, является ли объект мгновенным, используйте `is.instant` .

```
library(lubridate)

today_start <- dmy_hms("22.07.2016 12:00:00", tz = "IST") # default tz="UTC"
today_start
## [1] "2016-07-22 12:00:00 IST"
is.instant(today_start)
## [1] TRUE

now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
is.instant(now_dt)
## [1] TRUE

is.instant("helloworld")
## [1] FALSE
is.instant(60)
## [1] FALSE
```

Интервалы, длительности и периоды

Интервалы - это самый простой способ записи временных интервалов в lubridate. Интервал представляет собой промежуток времени, который происходит между двумя конкретными **моментами** .

```
# create interval by substracting two instants
today_start <- ymd_hms("2016-07-22 12-00-00", tz="IST")
today_start
## [1] "2016-07-22 12:00:00 IST"
today_end <- ymd_hms("2016-07-22 23-59-59", tz="IST")
today_end
## [1] "2016-07-22 23:59:59 IST"
span <- today_end - today_start
span
## Time difference of 11.99972 hours
as.interval(span, today_start)
## [1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

# create interval using interval() function
span <- interval(today_start, today_end)
[1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST
```

Длительность измеряет точное количество времени, которое происходит между двумя моментами.

```
duration(60, "seconds")
## [1] "60s"

duration(2, "minutes")
## [1] "120s (~2 minutes)"
```

Примечание. Единицы больше, чем недели, не используются из-за их изменчивости.

Длительность может быть создана с использованием `dseconds` , `dminutes` и других вспомогательных функций продолжительности.
Запустить `?quick_durations` для полного списка.

```
dseconds(60)
## [1] "60s"

dhours(2)
## [1] "7200s (~2 hours)"

dyears(1)
## [1] "31536000s (~365 days)"
```

Длительность может быть вычтена и добавлена к моментам, чтобы получить новые мгновения.

```
today_start + dhours(5)
## [1] "2016-07-22 17:00:00 IST"

today_start + dhours(5) + dminutes(30) + dseconds(15)
## [1] "2016-07-22 17:30:15 IST"
```

Длительность может быть создана из интервалов.

```
as.duration(span)
[1] "43199s (~12 hours)"
```

Периоды измеряют изменение часового времени, которое происходит между двумя моментами.

Периоды могут быть созданы с использованием функции `period` а также других вспомогательных функций, таких как `seconds` , `hours` и т. Д. Чтобы получить полный список вспомогательных функций периода, запустите « `?quick_periods` .

```
period(1, "hour")
## [1] "1H 0M 0S"

hours(1)
## [1] "1H 0M 0S"

period(6, "months")
## [1] "6m 0d 0H 0M 0S"

months(6)
## [1] "6m 0d 0H 0M 0S"

years(1)
## [1] "1y 0m 0d 0H 0M 0S"
```

Функция `is.period` может использоваться для проверки того, является ли объект

периодом.

```
is.period(years(1))
## [1] TRUE

is.period(dyears(1))
## [1] FALSE
```

Даты округления

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

`round_date()` принимает объект с датой и округляет его до ближайшего целочисленного значения указанного единицы времени.

```
round_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

round_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"

round_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

`floor_date()` принимает объект с датой и округляет его до ближайшего целочисленного значения указанного единицы времени.

```
floor_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

floor_date(now_dt, "hour")
## [1] "2016-07-22 13:00:00 IST"

floor_date(now_dt, "year")
## [1] "2016-01-01 IST"
```

`ceiling_date()` принимает объект времени и округляет его до ближайшего целочисленного значения указанного единицы времени.

```
ceiling_date(now_dt, "minute")
## [1] "2016-07-22 13:54:00 IST"

ceiling_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"

ceiling_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

Разница между периодом и продолжительностью

В отличие от длительностей, периоды могут использоваться для точного моделирования времени такта, не зная, когда происходят такие события, как прыжки секунд, прыжки дней и изменения DST.

```
start_2012 <- ymd_hms("2012-01-01 12:00:00")
## [1] "2012-01-01 12:00:00 UTC"

# period() considers leap year calculations.
start_2012 + period(1, "years")
## [1] "2013-01-01 12:00:00 UTC"

# Here duration() doesn't consider leap year calculations.
start_2012 + duration(1)
## [1] "2012-12-31 12:00:00 UTC"
```

Часовые пояса

`with_tz` возвращает дату-время, поскольку оно появляется в другом часовом поясе.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

# corresponding Europe/Moscow time
with_tz(nyc_time, tzone = "Europe/Moscow")
## [1] "2016-07-22 12:49:08 MSK"
```

`force_tz` возвращает дату-время с таким же временем часов, что и x в новом часовом поясе.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

force_tz(nyc_time, tzone = "Europe/Moscow") # only timezone changes
## [1] "2016-07-22 05:49:08 MSK"
```

Прочитайте lubridate онлайн: https://riptutorial.com/ru/r/topic/2496/lubridate

глава 16: Meta: Руководство по документации

замечания

Чтобы обсудить редактирование документов тегов R, зайдите в чат R.

Examples

Хорошие примеры

Большинство рекомендаций по созданию хороших примеров для вопросов и ответов переносятся в документацию.

- Сделайте его минимальным и дойдете до сути. Осложнения и отступления являются контрпродуктивными.
- Включите как рабочий код, так и прозу, объясняющие его. Ни один из них не является достаточным.
- Не полагайтесь на внешние источники данных. Создайте данные или используйте библиотеку наборов данных, если это возможно:

```
library(help = "datasets")
```

В контексте Документов есть несколько дополнительных соображений:

- Обратитесь к встроенным документам типа `?data.frame` если это необходимо. SO Docs не являются попыткой заменить встроенные документы. Важно убедиться, что новые пользователи R знают, что существуют встроенные документы, а также как их найти.
- Переместите содержимое, применимое к нескольким примерам, в раздел «Примечания».

Стиль

Запрашивает

Если вы хотите, чтобы ваш код был доступен для копирования, удалите приглашения, такие как `R>` , `>` или `+` в начале каждой новой строки. Некоторые авторы Docs предпочитают не делать скопирования в скором времени, и это нормально.

Консольный выход

Консольный выход должен четко отличаться от кода. Общие подходы включают:

- Включать подсказки на входе (как видно при использовании консоли).
- Прокомментируйте все выходные данные: `#` или `##` начиная каждую строку.
- Печатать как есть, доверяя ведущему `[1]` чтобы выход выделялся из входа.
- Добавьте пустую строку между кодом и выходом консоли.

присваивание

`=` и `<-` отлично подходят для назначения объектов R. Используйте пробел должным образом, чтобы избежать написания кода, который трудно разобрать, например `x<-1` (неоднозначный между `x <- 1` и `x < -1`)

Комментарии кодов

Обязательно объясните цель и функцию самого кода. Не существует жесткого правила о том, должно ли это объяснение быть в прозе или в комментариях к коду. Проза может быть более читаемой и позволяет более подробные объяснения, но комментарии коментариев упрощают копирование. Помните оба варианта.

Разделы

Многие примеры достаточно коротки, чтобы не требовать секций, но если вы их используете, начните с H1 .

Прочитайте Meta: Руководство по документации онлайн:
https://riptutorial.com/ru/r/topic/5410/meta--руководство-по-документации

глава 17: R Markdown Notebooks (от RStudio)

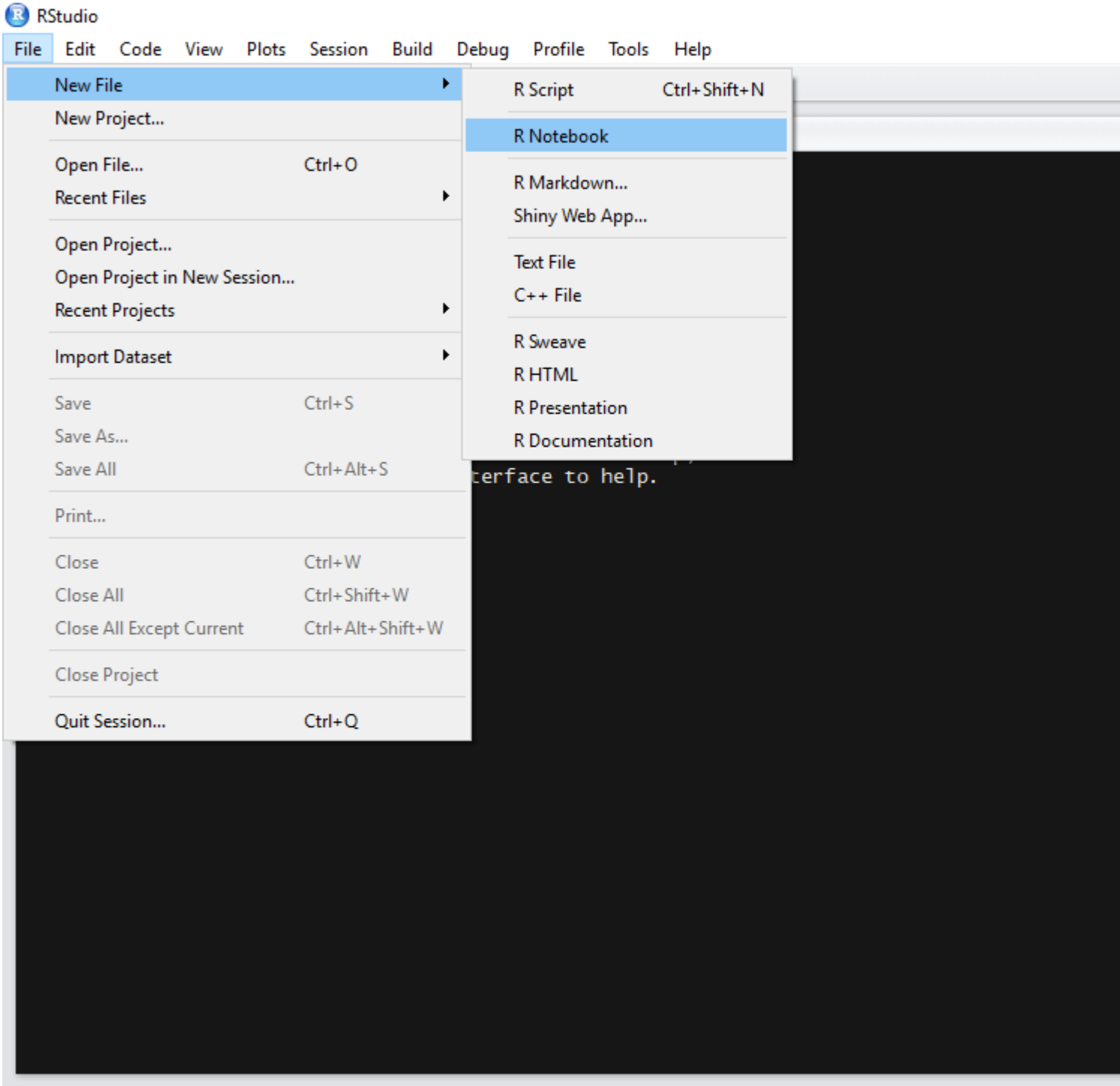
Вступление

Ноутбук R - это документ R Markdown с кусками, которые могут выполняться независимо и интерактивно, причем вывод отображается сразу под входом. Они аналогичны документам R Markdown, за исключением результатов, отображаемых в режиме создания / редактирования ноутбука R, а не в отображаемом выходе. **Примечание.** R Ноутбуки - это новая функция RStudio и доступны только в версии 1.0 или выше от RStudio.

Examples

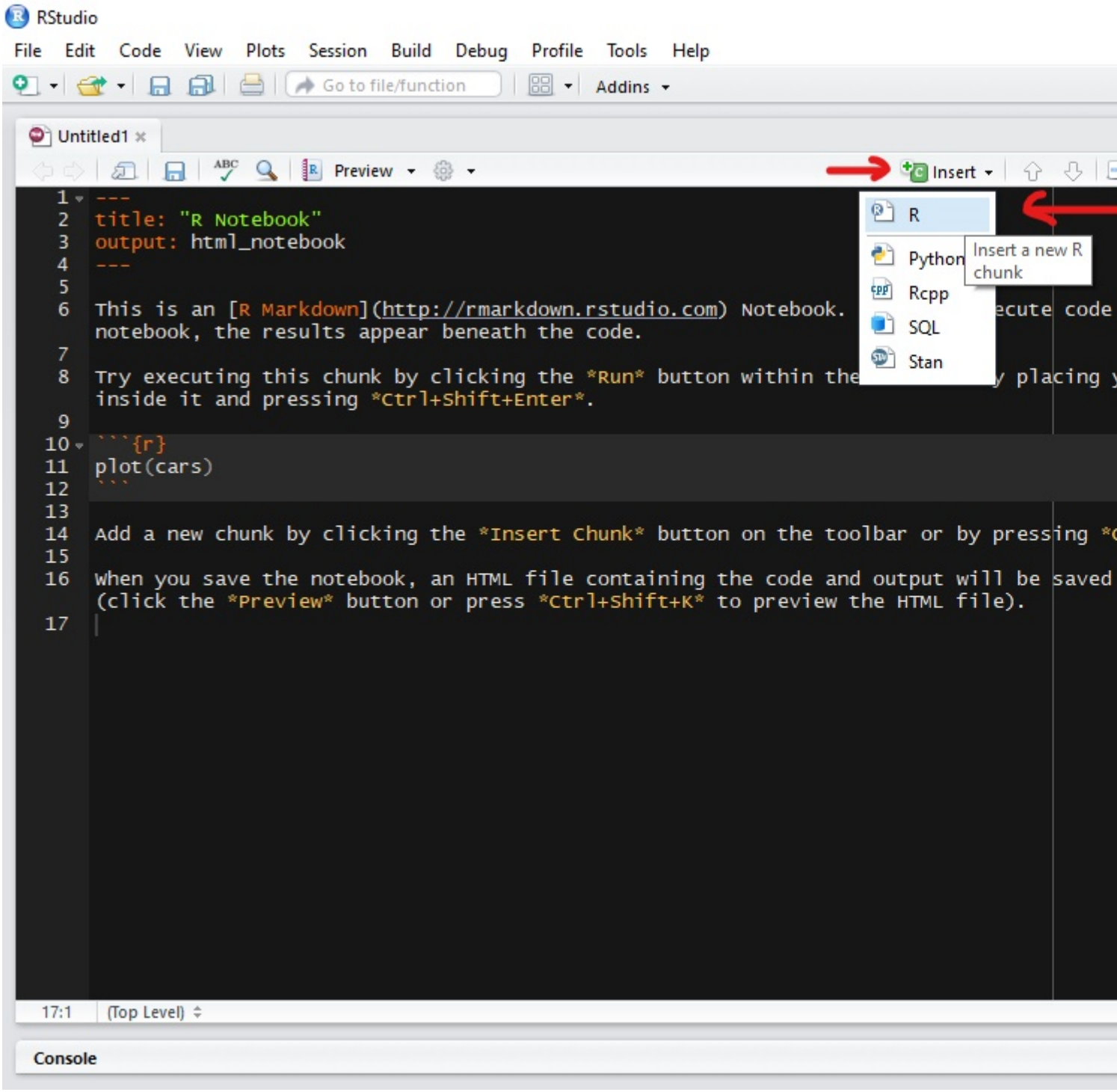
Создание ноутбука

Вы можете создать новый ноутбук в RStudio с помощью команды меню File -> New File -> R Notebook
Если вы не видите вариант для R Notebook, вам необходимо обновить версию RStudio. Для установки RStudio следуйте этому руководству



Вставка кусков

Куски - это фрагменты кода, которые могут быть выполнены интерактивно. Чтобы вставить новый кусок, нажав кнопку **вставки** , находящуюся на панели инструментов ноутбука, и выберите нужную платформу кода (в этом случае R, так как мы хотим записать R-код). В качестве альтернативы мы можем использовать сочетания клавиш для вставки нового фрагмента **Ctrl + Alt + I (OS X: Cmd + Option + I)**



Выполнение кода чанка

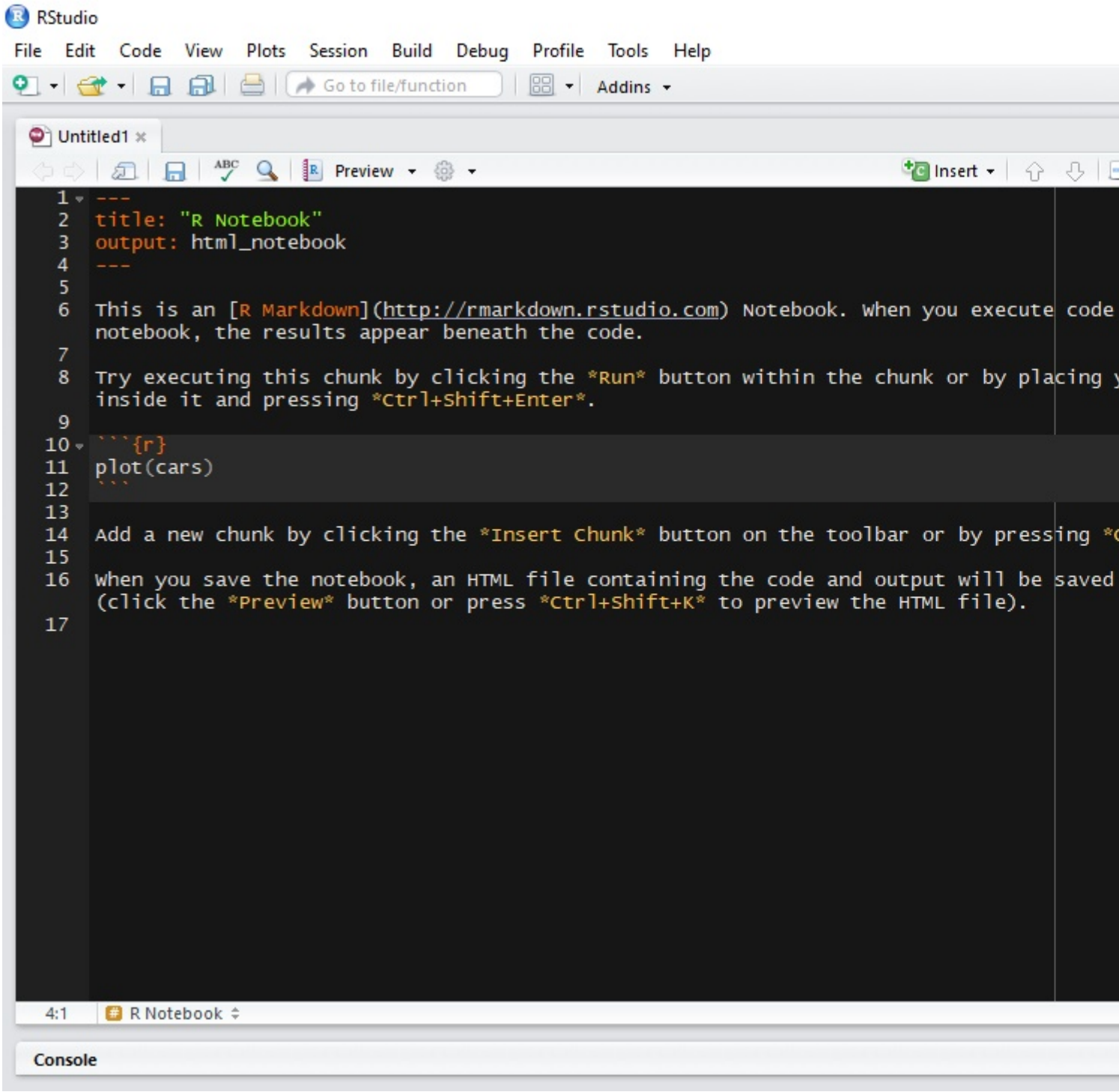
Вы можете запустить текущий кусок, нажав кнопку « **Текущий ток» (зеленая кнопка воспроизведения),** присутствующая на правой стороне фрагмента. В качестве альтернативы мы можем использовать сочетание клавиш **Ctrl + Shift + Enter (OS X: Cmd + Shift + Enter)**

Выход из всех строк в куске будет отображаться под куском.

Разделение кода на куски

Поскольку кусок производит свой вывод под куском, при наличии нескольких строк кода в одном блоке, который производит мультипликативные выходы, часто бывает полезно разбить на несколько кусков, чтобы каждый кусок выдавал один вывод.

Для этого выберите код, который вы хотите разбить на новый фрагмент, и нажмите **Ctrl + Alt + I (OS X: Cmd + Option + I)**



Выполнение

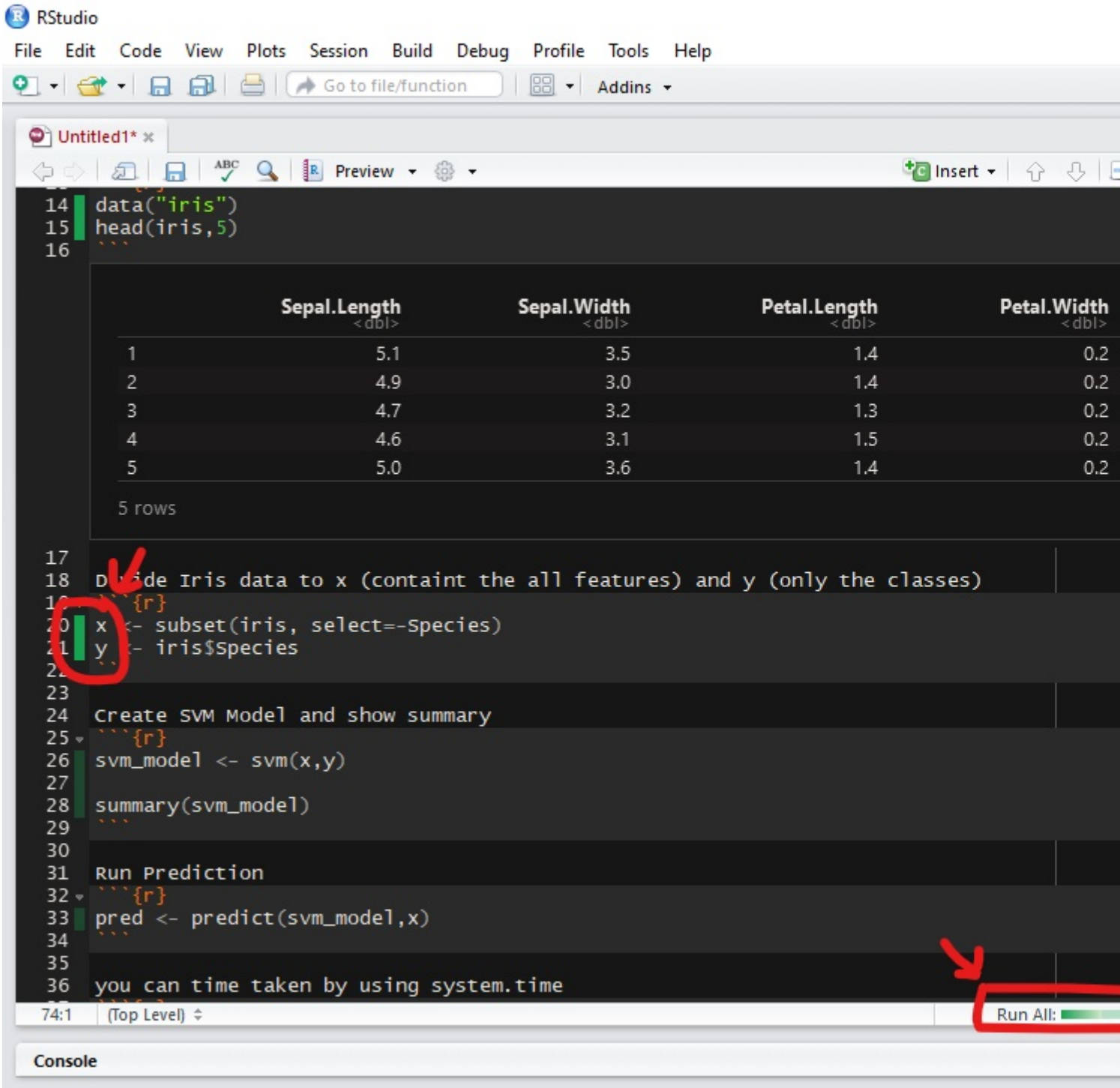
Когда вы выполняете код в записной книжке, в канаве появится индикатор, показывающий ход выполнения. Строки кода, которые были отправлены в R, отмечены темно-зеленым цветом; линии, которые еще не отправлены в R, отмечены светло-зеленым цветом.

Выполнение нескольких фрагментов

Запуск или повторное выполнение отдельных фрагментов путем нажатия кнопки «Выполнить» для всех фрагментов, представленных в документе, может быть болезненным. Мы можем использовать « **Запустить все»** из меню «Вставка» на панели инструментов, чтобы запустить все куски, присутствующие в ноутбуке. Комбинация клавиш - **Ctrl + Alt + R (OS X: Cmd + Option + R)**

Также есть опция **Restart R и Run All Chunks** (доступная в меню «Выполнить» на панели инструментов редактора), которая дает вам новую сессию R перед запуском всех кусков.

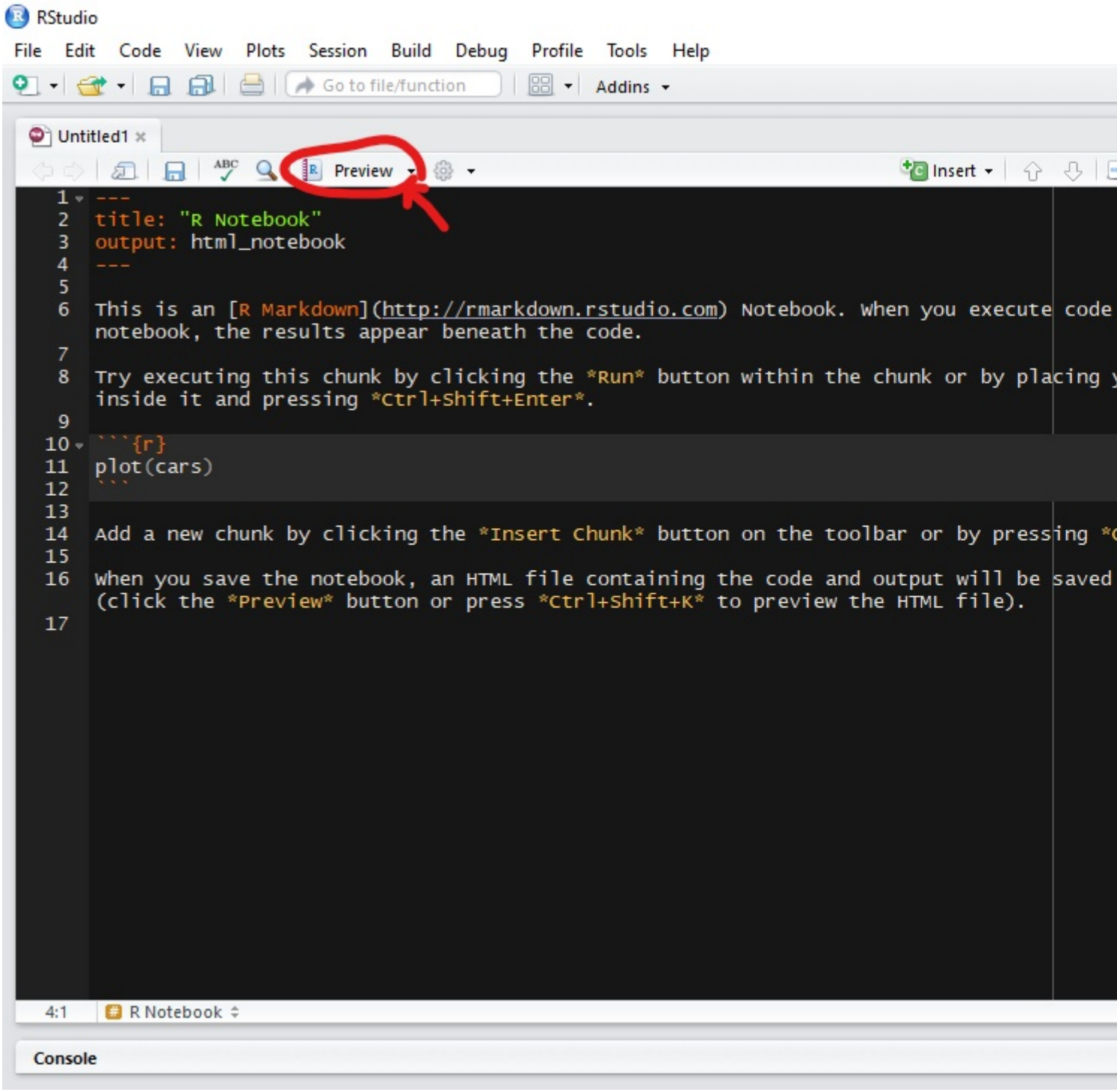
У нас также есть такие опции, как « **Запустить все куски выше»** и « **Запустить все** куски» **ниже,** чтобы запустить куски над или ниже от выбранного фрагмента.



Предварительный просмотр

Перед рендерингом окончательной версии ноутбука мы можем просмотреть результат. Нажмите кнопку « **Просмотр**» на панели инструментов и выберите нужный формат вывода.

Вы можете изменить тип вывода, используя параметры вывода как «pdf_document» или « html_notebook»,



Сохранение и совместное использование

Когда `.Rmd` записная книжка `.Rmd` , рядом с ней `.nb.html` файл `.nb.html` . Этот файл представляет собой автономный HTML-файл, который содержит как визуализированную копию ноутбука со всеми текущими выходами блока (подходит для отображения на веб-сайте), так и копию самого ноутбука.

Больше информации можно найти в документах RStudio

Прочитайте R Markdown Notebooks (от RStudio) онлайн:
https://riptutorial.com/ru/r/topic/10728/r-markdown-notebooks--от-rstudio-

глава 18: R в LaTeX с knitr

Синтаксис

1. << internal-code-chunk-name, options ... >> =
   # R Code Here
   @
2. \ Sexpr {#R Code Here}
3. << read-external-R-file >> =
   read_chunk ('r-file.R')
   @
   << external-code-chunk-name, options ... >> =
   @

параметры

вариант	подробности
эхо	(TRUE / FALSE) - включить ли исходный код R в выходной файл
сообщение	(TRUE / FALSE) - включить ли сообщения из выполнения R-источника в выходной файл
предупреждение	(TRUE / FALSE) - включить ли предупреждения из выполнения R-источника в выходной файл
ошибка	(TRUE / FALSE) - включить ли ошибки из выполнения R-источника в выходной файл
кэш	(TRUE / FALSE) - следует ли кэшировать результаты выполнения R-источника
fig.width	(числовой) - ширина графика, сгенерированного при выполнении R-источника
fig.height	(числовой) - высота графика, сгенерированного при выполнении R-источника

замечания

Knitr - это инструмент, который позволяет нам переплетать естественный язык (в виде LaTeX) и исходный код (в форме R). В общем, концепция перемежающегося естественного

языка и исходного кода называется грамотным программированием . Поскольку файлы knitr содержат смесь LaTeX (традиционно размещенную в .tex-файлах) и R (традиционно размещенную в .R-файлах), требуется новое расширение файла R noweb (.Rnw). Файлы . Rnw содержат смесь LaTeX и R-кода.

Knitr позволяет создавать статистические отчеты в формате PDF и является ключевым инструментом для достижения воспроизводимых исследований .

Компилирование файлов .Rnw в PDF - это двухэтапный процесс. Во-первых, нам нужно знать, как выполнить код R и отобразить вывод в формате, который может понять компилятор LaTeX (процесс называется «kniting»). Мы делаем это, используя пакет knitr. Команда для этого показана ниже, если вы установили пакет knitr :

```
Rscript -e "library(knitr); knit('r-noweb-file.Rnw')
```

Это приведет к созданию нормального файла .tex (например, r-noweb.tex в этом примере), который затем можно преобразовать в файл PDF, используя:

```
pdflatex r-noweb-file.tex
```

Examples

R в латексе с нарушением ввода текста Knitr и кода

Knitr - это R-пакет, который позволяет нам смешивать R-код с кодом LaTeX. Один из способов добиться этого - это внешние куски кода. Внешние фрагменты кода позволяют нам разрабатывать / тестировать R-скрипты в среде разработки R, а затем включать результаты в отчет. Это мощный организационный метод. Этот подход показан ниже.

```
# r-noweb-file.Rnw
\documentclass{article}

 <<echo=FALSE,cache=FALSE>>=
 knitr::opts_chunk$set(echo=FALSE,  cache=TRUE)
 knitr::read_chunk('r-file.R')
 @

\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

One we have called the read\_chunk command above we can reference sections of code in the r-
file.R script.

<<Chunk1>>=
@
\end{document}
```

При использовании этого подхода мы сохраняем наш код в отдельном файле R, как

показано ниже.

```
## r-file.R
## note the specific comment style of a single pound sign followed by four dashes

# ---- Chunk1 ----

print("This is R Code in an external file")

x <- seq(1:10)
y <- rev(seq(1:10))
plot(x,y)
```

R в латексе с кусками Knitr и Inline Code

Knitr - это R-пакет, который позволяет нам смешивать R-код с кодом LaTeX. Один из способов добиться этого - встроенные фрагменты кода. Это утверждение показано ниже.

```
# r-noweb-file.Rnw
\documentclass{article}
\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

<<my-label>>=
print("This is an R Code Chunk")
x <- seq(1:10)
@

Above is an internal code chunk.
We can access data created in any code chunk inline with our LaTeX code like this.
The length of array x is \Sexpr{length(x)}.

\end{document}
```

R в LaTex с Knitr и внутренними кодовыми фрагментами

Knitr - это R-пакет, который позволяет нам смешивать R-код с кодом LaTeX. Один из способов добиться этого - внутренние куски кода. Это утверждение показано ниже.

```
# r-noweb-file.Rnw
\documentclass{article}
\begin{document}
This is an Rnw file (R noweb).  It contains a combination of LateX and R.

<<code-chunk-label>>=
print("This is an R Code Chunk")
x <- seq(1:10)
y <- seq(1:10)
plot(x,y)  # Brownian motion
@

\end{document}
```

Прочитайте R в LaTeX с knitr онлайн: https://riptutorial.com/ru/r/topic/4334/r-в-latex-c-knitr

глава 19: R примера по примерам

Вступление

Эта тема предназначена для напоминания о языке R без какого-либо текста, с поясняющими примерами.

Каждый пример должен быть максимально сукцинт.

Examples

Типы данных

векторы

```
a <- c(1, 2, 3)
b <- c(4, 5, 6)
mean_ab <- (a + b) / 2

d <- c(1, 0, 1)
only_1_3 <- a[d == 1]
```

Матрицы

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
dimnames(mat) <- list(c(), c("a", "b", "c"))
mat[,] == mat
```

Dataframes

```
df <- data.frame(qualifiers = c("Buy", "Sell", "Sell"),
                 symbols = c("AAPL", "MSFT", "GOOGL"),
                 values = c(326.0, 598.3, 201.5))
df$symbols == df[[2]]
df$symbols == df[["symbols"]]
df[[2, 1]] == "AAPL"
```

Списки

```
l <- list(a = 500, "aaa", 98.2)
```

```
length(l)       == 3
class(l[1])     == "list"
class(l[[1]])   == "numeric"
class(l$a)      == "numeric"
```

Среды

```
env <- new.env()
env[["foo"]] = "bar"
env2 <- env
env2[["foo"]] = "BAR"

env[["foo"]] == "BAR"
get("foo", envir = env) == "BAR"
rm("foo", envir = env)
env[["foo"]] == NULL
```

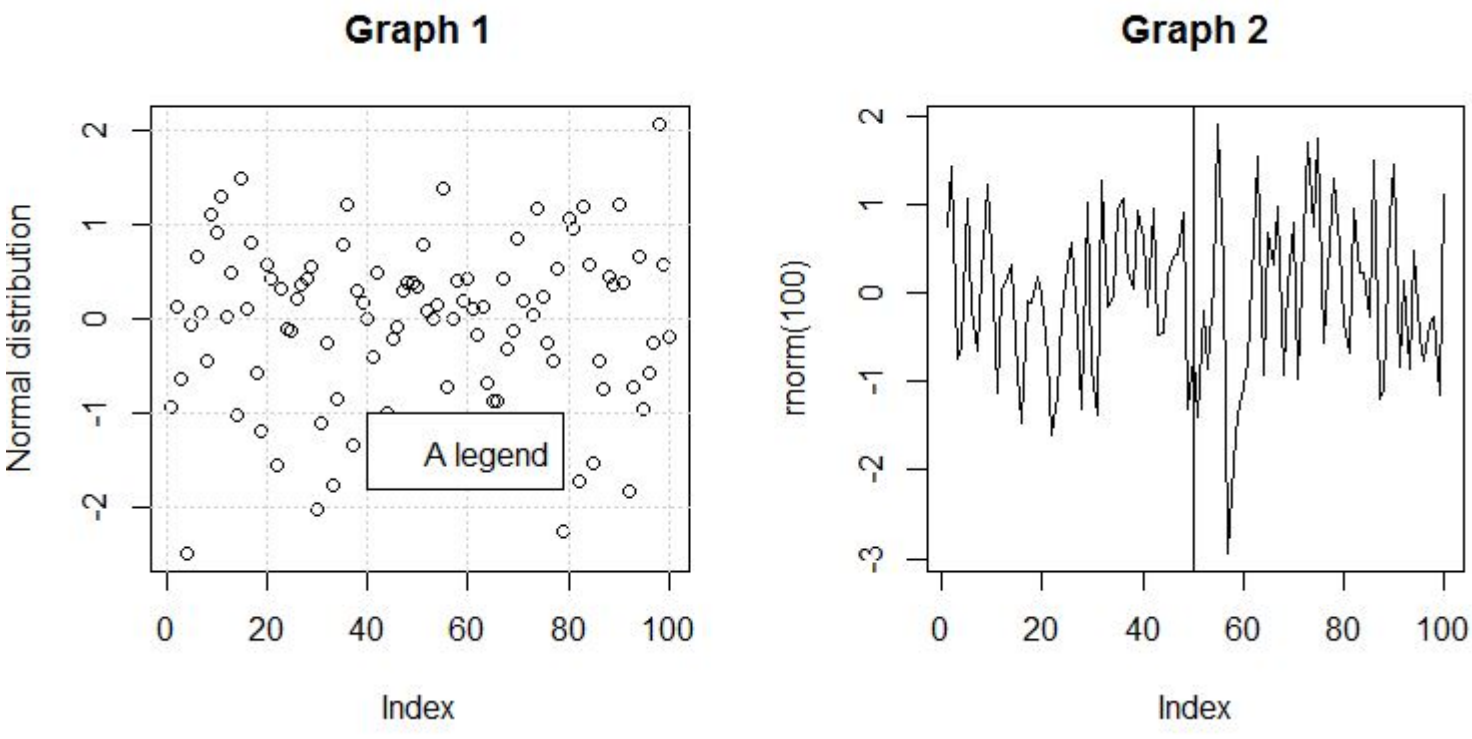
Построение графика (с использованием графика)

```
# Creates a 1 row - 2 columns format
par(mfrow=c(1,2))

plot(rnorm(100), main = "Graph 1", ylab = "Normal distribution")
grid()
legend(x = 40, y = -1, legend = "A legend")

plot(rnorm(100), main = "Graph 2", type = "l")
abline(v = 50)
```

Результат:



Обычно используемые функции

```
# Create 100 standard normals in a vector
x <- rnorm(100, mean = 0, sd = 1)

# Find the lenght of a vector
length(x)

# Compute the mean
mean(x)

# Compute the standard deviation
sd(x)

# Compute the median value
median(x)

# Compute the range (min, max)
range(x)

# Sum an iterable
sum(x)

# Cumulative sum (x[1], x[1]+x[2], ...)
cumsum(x)

# Display the first 3 elements
head(3, x)

# Display min, 1st quartile, median, mean, 3rd quartile, max
summary(x)

# Compute successive difference between elements
diff(x)

# Create a range from 1 to 10 step 1
1:10

# Create a range from 1 to 10 step 0.1
seq(1, 10, 0.1)

# Print a string
print("hello world")
```

Прочитайте R примера по примерам онлайн: https://riptutorial.com/ru/r/topic/10827/r-примера-по-примерам

глава 20: Rcpp

Examples

Компиляция встроенного кода

Rcpp имеет две функции, которые позволяют компиляцию кода встраивать и экспортировать непосредственно в R: `cppFunction()` и `evalCpp()` . Третья функция, называемая `sourceCpp()` существует для чтения в коде C ++ в отдельном файле, но может быть использована сродни `cppFunction()` .

Ниже приведен пример компиляции функции C ++ внутри *R.* Обратите внимание на использование `""` чтобы окружить источник.

```
# Note - This is R code.
# cppFunction in Rcpp allows for rapid testing.
require(Rcpp)

# Creates a function that multiples each element in a vector
# Returns the modified vector.
cppFunction("
NumericVector exfun(NumericVector x, int i){
x = x*i;
return x;
}")

# Calling function in R
exfun(1:5, 3)
```

Чтобы быстро понять выражение C ++, используйте:

```
# Use evalCpp to evaluate C++ expressions
evalCpp("std::numeric_limits<double>::max()")
## [1] 1.797693e+308
```

Атрибуты Rcpp

Атрибуты Rcpp делают процесс работы с R и C ++ простым. Форма атрибутов принимает:

```
// [[Rcpp::attribute]]
```

Использование атрибутов обычно связано с:

```
// [[Rcpp::export]]
```

который помещается непосредственно над объявленным заголовком функции при чтении в файле C ++ через `sourceCpp()` .

Ниже приведен пример внешнего файла C ++, который использует атрибуты.

```
// Add code below into C++ file Rcpp_example.cpp

#include <Rcpp.h>
using namespace Rcpp;

// Place the export tag right above function declaration.
// [[Rcpp::export]]
double muRcpp(NumericVector x){

    int n = x.size(); // Size of vector
    double sum = 0; // Sum value

    // For loop, note cpp index shift to 0
    for(int i = 0; i < n; i++){
        // Shorthand for sum = sum + x[i]
        sum += x[i];
    }

    return sum/n; // Obtain and return the Mean
}

// Place dependent functions above call or
// declare the function definition with:
double muRcpp(NumericVector x);

// [[Rcpp::export]]
double varRcpp(NumericVector x, bool bias = true){

    // Calculate the mean using C++ function
    double mean = muRcpp(x);
    double sum = 0;

    int n = x.size();

    for(int i = 0; i < n; i++){
        sum += pow(x[i] - mean, 2.0); // Square
    }

    return sum/(n-bias); // Return variance
}
```

Чтобы использовать этот внешний файл *C* ++ внутри *R* , мы делаем следующее:

```
require(Rcpp)

# Compile File
sourceCpp("path/to/file/Rcpp_example.cpp")

# Make some sample data
x = 1:5

all.equal(muRcpp(x), mean(x))
## TRUE

all.equal(varRcpp(x), var(x))
## TRUE
```

Расширение Rcpp с помощью плагинов

В C ++ можно установить разные флаги компиляции, используя:

```
// [[Rcpp::plugins(name)]]
```

Список встроенных плагинов:

```
// built-in C++11 plugin
// [[Rcpp::plugins(cpp11)]]

// built-in C++11 plugin for older g++ compiler
// [[Rcpp::plugins(cpp0x)]]

// built-in C++14 plugin for C++14 standard
// [[Rcpp::plugins(cpp14)]]

// built-in C++1y plugin for C++14 and C++17 standard under development
// [[Rcpp::plugins(cpp1y)]]

// built-in OpenMP++11 plugin
// [[Rcpp::plugins(openmp)]]
```

Указание дополнительных зависимостей сборки

Чтобы использовать дополнительные пакеты в экосистеме Rcpp, правильным заголовком может быть не `Rcpp.h` а `Rcpp<PACKAGE>.h` (как, *например,* для RcppArmadillo). Обычно его нужно импортировать, а затем зависимость заявляется внутри

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

Примеры:

```
// Use the RcppArmadillo package
// Requires different header file from Rcpp.h
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]

// Use the RcppEigen package
// Requires different header file from Rcpp.h
#include <RcppEigen.h>
// [[Rcpp::depends(RcppEigen)]]
```

Прочитайте Rcpp онлайн: https://riptutorial.com/ru/r/topic/1404/rcpp

глава 21: RODBC

Examples

Подключение к файлам Excel через RODBC

В то время как `RODBC` ограничен компьютерами Windows с совместимой архитектурой между R и любыми целевыми RDMS, одной из его ключевых гибкостей является работа с файлами Excel, как если бы они были базами данных SQL.

```
require(RODBC)
con = odbcConnectExcel("myfile.xlsx") # open a connection to the Excel file
sqlTables(con)$TABLE_NAME # show all sheets
df = sqlFetch(con, "Sheet1") # read a sheet
df = sqlQuery(con, "select * from [Sheet1 $]") # read a sheet (alternative SQL syntax)
close(con) # close the connection to the file
```

Подключение базы данных управления SQL Server для получения отдельной таблицы

Другое использование RODBC связано с базой данных SQL Server Management. Нам нужно указать здесь «Драйвер», то есть SQL Server, имя базы данных «Atilla», а затем использовать `sqlQuery` для извлечения полной таблицы или ее доли.

```
library(RODBC)
cn  <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

Подключение к реляционным базам данных

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

Это приведет к подключению к экземпляру SQL Server. Для получения дополнительной информации о том, как должна выглядеть ваша строка подключения, посетите connectionstrings.com

Кроме того, поскольку база данных не указана, вы должны убедиться, что вы полностью квалифицируете объект, который хотите запросить, например, это имя_базы.файла.объекта

Прочитайте RODBC онлайн: https://riptutorial.com/ru/r/topic/2471/rodbc

глава 22: roxygen2

параметры

параметр	подробности
автор	Автор пакета
Примеры	Следующие строки будут примерами того, как использовать документированную функцию
экспорт	Чтобы экспортировать функцию - то есть сделать ее вызываемой пользователями пакета
Импортировать	Пакет (ы) пространства имен для импорта
importFrom	Функции для импорта из пакета (имя списка)
пары	Параметр функции для документирования

Examples

Документирование пакета с помощью roxygen2

Запись с помощью roxygen2

roxygen2 - это пакет, созданный Хэдли Уикхэмом для облегчения документирования.

Он позволяет включать документацию внутри R-скрипта в строки, начинающиеся с `#'` . Различные параметры, переданные в документацию, начинаются с `@` , например, создатель пакета будет написан следующим образом:

```
#' @author The Author
```

Например, если мы хотим записать следующую функцию:

```
mean<-function(x) sum(x)/length(x)
```

Мы захотим написать небольшое описание этой функции и объяснить параметры следующим образом (каждая строка будет объяснена и подробно описана после):

```
#' Mean
```

```
#'
#' A function to compute the mean of a vector
#' @param x A numeric vector
#' @keyword mean
#' @importFrom base sum
#' @export
#' @examples
#' mean(1:3)
#' \dontrun{ mean(1:1e99) }
mean<-function(x) sum(x)/length(x)
```

- Первая строка `#' Mean` - это название документации, следующие строки составляют корпус.
- Каждый параметр функции должен быть детализирован с помощью соответствующего `@param` . `@export` указал, что это имя функции должно быть экспортировано и, таким образом, может быть `@export` при загрузке пакета.
- `@keyword` предоставляет релевантные ключевые слова при поиске справки
- `@importFrom` перечисляет все функции для импорта из пакета, который будет использоваться в этой функции или в вашем пакете. Обратите внимание, что импортирование полного пространства имен пакета может быть выполнено с помощью `@import`
- Затем примеры записываются ниже тега `@example` .
    - Первый будет оцениваться при создании пакета;
    - Второй не будет - обычно для предотвращения длинных вычислений - из-за команды `\dontrun` .

Построение документации

Документацию можно создать с помощью `devtools::document()` . Также обратите внимание, что `devtools::check()` автоматически создаст документацию и сообщит о недостающих аргументах в документации о функциях в качестве предупреждений.

Прочитайте roxygen2 онлайн: https://riptutorial.com/ru/r/topic/5171/roxygen2

глава 23: Spark API (SparkR)

замечания

Пакет `SparkR` позволит вам работать с распределенными кадрами данных поверх кластера Spark . Они позволяют выполнять операции, такие как выбор, фильтрация, агрегация на очень больших наборах данных. Обзор SparkR Документация пакета SparkR

Examples

Настройка контекста Spark

Настройка контекста Spark в R

Чтобы начать работу с распределенными файловыми кадрами Sparks, вы должны подключить свою R-программу к существующему Spark Cluster.

```
library(SparkR)
sc <- sparkR.init() # connection to Spark context
sqlContext <- sparkRSQL.init(sc) # connection to SQL context
```

Вот информация о том, как подключить вашу среду IDE к Spark-кластеру.

Получить искровой кластер

В инструкции по установке есть Apache Spark . В принципе, вы можете использовать Spark Cluster локально через java (см. Инструкции) или использовать (несвободные) облачные приложения (например, Microsoft Azure [тема сайта] , IBM).

Данные кэша

Какие:

Кэширование может оптимизировать вычисления в Spark. Кэширование хранит данные в памяти и является особым случаем сохранения. Здесь объясняется, что происходит, когда вы кешируете RDD в Spark.

Зачем:

В основном кэширование сохраняет промежуточный частичный результат - обычно после преобразований - ваших исходных данных. Таким образом, при использовании кэшированного RDD доступны уже преобразованные данные из памяти, не

перераспределяя предыдущие преобразования.

Как:

Вот пример того, как быстро получить доступ к большим данным *(здесь 3 ГБ большой csv)* из хранилища в памяти при обращении к нему более одного раза:

```
library(SparkR)
# next line is needed for direct csv import:
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-
shell"')
sc <- sparkR.init()
sqlContext <- sparkRSQL.init(sc)

# loading 3 GB big csv file:
train <- read.df(sqlContext, "/train.csv", source = "com.databricks.spark.csv", inferSchema =
"true")
cache(train)
system.time(head(train))
# output: time elapsed: 125 s. This action invokes the caching at this point.
system.time(head(train))
# output: time elapsed: 0.2 s (!!)
```

Создание RDD (Resilient Distributed Datasets)

Из фрейма данных:

```
mtrdd <- createDataFrame(sqlContext, mtcars)
```

Из csv:

Для csv вам нужно добавить пакет csv в среду перед началом контекста Spark:

```
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-
shell"') # context for csv import read csv ->
sc <- sparkR.init()
sqlContext <- sparkRSQL.init(sc)
```

Затем вы можете загрузить csv либо путем вывода схемы данных данных в столбцах:

```
train <- read.df(sqlContext, "/train.csv", header= "true", source =
"com.databricks.spark.csv", inferSchema = "true")
```

Или, предварительно указав схему данных:

```
 customSchema <- structType(
    structField("margin", "integer"),
    structField("gross", "integer"),
    structField("name", "string"))
```

```
 train <- read.df(sqlContext, "/train.csv", header= "true", source =
"com.databricks.spark.csv", schema = customSchema)
```

Прочитайте Spark API (SparkR) онлайн: https://riptutorial.com/ru/r/topic/5349/spark-api--sparkr-

глава 24: sqldf

Examples

Основные примеры использования

`sqldf()` из пакета `sqldf` позволяет использовать SQLite-запросы для выбора и управления данными в R. SQL-запросы вводятся как символьные строки.

Чтобы выбрать первые 10 рядов набора данных «бриллианты» из пакета `ggplot2` , например:

```
data("diamonds")
head(diamonds)
```

```
# A tibble: 6 x 10
  carat       cut color clarity depth table price     x     y     z
  <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23     Ideal     E     SI2  61.5    55   326  3.95  3.98  2.43
2  0.21   Premium     E     SI1  59.8    61   326  3.89  3.84  2.31
3  0.23      Good     E     VS1  56.9    65   327  4.05  4.07  2.31
4  0.29   Premium     I     VS2  62.4    58   334  4.20  4.23  2.63
5  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
6  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
```

```
require(sqldf)
sqldf("select * from diamonds limit 10")
```

```
   carat       cut color clarity depth table price    x    y    z
1   0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2   0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3   0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4   0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5   0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6   0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
7   0.24 Very Good     I    VVS1  62.3    57   336 3.95 3.98 2.47
8   0.26 Very Good     H     SI1  61.9    55   337 4.07 4.11 2.53
9   0.22      Fair     E     VS2  65.1    61   337 3.87 3.78 2.49
10  0.23 Very Good     H     VS1  59.4    61   338 4.00 4.05 2.39
```

Чтобы выбрать первые 10 строк, где для цвета «E»:

```
sqldf("select * from diamonds where color = 'E' limit 10")
```

```
   carat       cut color clarity depth table price    x    y    z
1   0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2   0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3   0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4   0.22      Fair     E     VS2  65.1    61   337 3.87 3.78 2.49
```

```
5   0.20   Premium     E     SI2  60.2    62   345 3.79 3.75 2.27
6   0.32   Premium     E      I1  60.9    58   345 4.38 4.42 2.68
7   0.23 Very Good     E     VS2  63.8    55   352 3.85 3.92 2.48
8   0.23 Very Good     E     VS1  60.7    59   402 3.97 4.01 2.42
9   0.23 Very Good     E     VS1  59.5    58   402 4.01 4.06 2.40
10  0.23      Good     E     VS1  64.1    59   402 3.83 3.85 2.46
```

Обратите внимание, что в приведенном выше примере цитируемые строки в SQL-запросе цитируются с использованием ", если общий запрос цитируется с помощью "" (это также работает в обратном порядке).

Предположим, что мы хотим добавить новый столбец для подсчета количества бриллиантов Premium cut более 1 карат:

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
  count(*)
1     1892
```

Результаты создания значений также могут быть возвращены в виде новых столбцов:

```
sqldf("select *, count(*) as cnt_big_E_colored_stones from diamonds where carat > 1 and color
= 'E' group by clarity")
```

```
  carat       cut color clarity depth table price    x    y    z
cnt_big_E_colored_stones
1  1.30      Fair     E      I1  66.5    58  2571 6.79 6.75 4.50
65
2  1.28     Ideal     E      IF  60.7    57 18700 7.09 6.99 4.27
28
3  2.02 Very Good     E     SI1  59.8    59 18731 8.11 8.20 4.88
499
4  2.03   Premium     E     SI2  61.5    59 18477 8.24 8.16 5.04
666
5  1.51     Ideal     E     VS1  61.5    57 18729 7.34 7.40 4.53
158
6  1.72 Very Good     E     VS2  63.4    56 18557 7.65 7.55 4.82
318
7  1.20     Ideal     E    VVS1  61.8    56 16256 6.78 6.87 4.22
52
8  1.55     Ideal     E    VVS2  62.5    55 18188 7.38 7.40 4.62
106
```

Если бы было интересно, какова **максимальная** `price` алмаза **по** `cut` :

```
sqldf("select cut,  max(price) from diamonds group by cut")

        cut max(price)
1      Fair      18574
2      Good      18788
3     Ideal      18806
4   Premium      18823
5 Very Good      18818
```

Прочитайте sqldf онлайн: https://riptutorial.com/ru/r/topic/2100/sqldf

глава 25: tidyverse

Examples

Создание tbl_df's

Tbl_df (произносится как *tibble diff*) - это вариация кадра данных, который часто используется в tidyverse-пакетах. Он реализован в пакете тиблей .

Используйте функцию `as_data_frame` для преобразования фрейма данных в tbl_df:

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

Одним из наиболее заметных различий между data.frames и tbl_dfs является то, как они печатаются:

```
# A tibble: 32 x 11
     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
*  <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
3   22.8     4 108.0    93  3.85 2.320 18.61     1     1     4     1
4   21.4     6 258.0   110  3.08 3.215 19.44     1     0     3     1
5   18.7     8 360.0   175  3.15 3.440 17.02     0     0     3     2
6   18.1     6 225.0   105  2.76 3.460 20.22     1     0     3     1
7   14.3     8 360.0   245  3.21 3.570 15.84     0     0     3     4
8   24.4     4 146.7    62  3.69 3.190 20.00     1     0     4     2
9   22.8     4 140.8    95  3.92 3.150 22.90     1     0     4     2
10  19.2     6 167.6   123  3.92 3.440 18.30     1     0     4     4
# ... with 22 more rows
```

- Печатный результат включает сводку размеров таблицы (`32 x 11`)
- Он включает в себя тип каждого столбца (`dbl`)
- Он печатает ограниченное количество строк. (Чтобы изменить `options(tibble.print_max = [number])` использования `options(tibble.print_max = [number])`).

Многие функции в пакете dplyr работают с tbl_dfs, например `group_by()` .

tidyverse: обзор

Что такое `tidyverse` ?

`tidyverse` - быстрый и элегантный способ превратить базовый `R` в усовершенствованный инструмент, переработанный Hadley / Rstudio. Разработка всех пакетов, включенных в `tidyverse` соответствует основным правилам манифеста « `tidyverse` инструмент» . Но

сначала пусть авторы описывают свой шедевр:

> Tidyverse - это набор пакетов, которые работают в гармонии, потому что они имеют общие представления данных и дизайн API. Пакет tidyverse спроектирован так, чтобы упростить установку и загрузку пакетов ядра с tidyverse в одной команде.
>
> Лучшее место, чтобы узнать обо всех пакетах в tidyverse и о том, как они подходят друг другу, - это R для Data Science. Ожидайте услышать больше о tidyverse в ближайшие месяцы, когда я работаю над улучшенными веб-сайтами пакетов, упрощая цитирование и предоставляя общий дом для обсуждения анализа данных с помощью tidyverse.
>
> (источник))

Как это использовать?

Просто с обычными пакетами `R` вам нужно установить и загрузить пакет.

```
install.package("tidyverse")
library("tidyverse")
```

Разница заключается в том, что по одной команде устанавливаются / загружаются несколько десятков пакетов. В качестве бонуса можно быть уверенным, что все установленные / загруженные пакеты совместимы.

Что это за пакеты?

Общеизвестные и широко используемые пакеты:

- ggplot2 : расширенная визуализация данных SO_doc
- dplyr : быстрый (Rcpp) и когерентный подход к манипулированию данными SO_doc
- tidyr : инструменты для сбора данных SO_doc
- readr : для импорта данных.
- purrr : делает ваши чистые функции мурлыканными, завершая функциональные инструменты программирования R с важными функциями от других языков, в стиле JS-пакетов underscore.js, lodash и lazy.js.
- tibble : современное переосмысление кадров данных.
- magrittr : трубопровод, чтобы сделать код более читаемым. SO_doc

Пакеты для управления конкретными форматами данных:

- hms : легко читается раз

- stringr : обеспечить сплоченный набор функций, предназначенных для работы со строками так же просто, как возможно
- lubridate : расширенные манипуляции с датами / времени SO_doc
- forcats : передовая работа с факторами .

Импорт данных:

- DBI : определяет общий интерфейс между R и системами управления базами данных (СУБД)
- haven : легко импортировать файлы SPSS, SAS и Stata SO_doc
- httr : целью httr является предоставление обертки для пакета curl, адаптированного к требованиям современных веб-API
- jsonlite : быстрый анализатор JSON и генератор, оптимизированный для статистических данных и сети
- readxl : файлы read.xls и .xlsx без пакетов зависимостей SO_doc
- rvest : rvest поможет вам очистить информацию с веб-страниц SO_doc
- xml2 : для XML

И моделирование:

- modelr : предоставляет функции, которые помогут вам создавать элегантные конвейеры при моделировании
- метла : легко извлекайте модели в аккуратные данные

Наконец, `tidyverse` предлагает использовать:

- knitr : удивительный универсальный программный движок, с легким API, разработанным, чтобы дать пользователям полный контроль над выходом без тяжелой работы по кодированию. SO_docs: один , два
- rmarkdown : пакет Rstudio для воспроизводимого программирования. SO_docs: один , два , три , четыре

Прочитайте tidyverse онлайн: https://riptutorial.com/ru/r/topic/1395/tidyverse

глава 26: xgboost

Examples

Кросс-проверка и настройка с помощью xgboost

```
library(caret) # for dummyVars
library(RCurl) # download https data
library(Metrics) # calculate errors
library(xgboost) # model

################################################################################
# Load data from UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.html)
urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
adults <- read.csv(textConnection(x), header=F)

# adults <-read.csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/adult/adult.data', header=F)
names(adults)=c('age','workclass','fnlwgt','education','educationNum',
                'maritalStatus','occupation','relationship','race',
                'sex','capitalGain','capitalLoss','hoursWeek',
                'nativeCountry','income')
# clean up data
adults$income <- ifelse(adults$income==' <=50K',0,1)
# binarize all factors
library(caret)
dmy <- dummyVars(" ~ .", data = adults)
adultsTrsf <- data.frame(predict(dmy, newdata = adults))
################################################################################

# what we're trying to predict adults that make more than 50k
outcomeName <- c('income')
# list of features
predictors <- names(adultsTrsf)[!names(adultsTrsf) %in% outcomeName]

# play around with settings of xgboost - eXtreme Gradient Boosting (Tree) library
# https://github.com/tqchen/xgboost/wiki/Parameters
# max.depth - maximum depth of the tree
# nrounds - the max number of iterations

# take first 10% of the data only!
trainPortion <- floor(nrow(adultsTrsf)*0.1)

trainSet <- adultsTrsf[ 1:floor(trainPortion/2),]
testSet <- adultsTrsf[(floor(trainPortion/2)+1):trainPortion,]

smallestError <- 100
for (depth in seq(1,10,1)) {
        for (rounds in seq(1,20,1)) {

                # train
                bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                               label = trainSet[,outcomeName],
                               max.depth=depth, nround=rounds,
                               objective = "reg:linear", verbose=0)
```

```
                gc()

                # predict
                predictions <- predict(bst, as.matrix(testSet[,predictors]),
outputmargin=TRUE)
                err <- rmse(as.numeric(testSet[,outcomeName]), as.numeric(predictions))

                if (err < smallestError) {
                        smallestError = err
                        print(paste(depth,rounds,err))
                }
        }
}

cv <- 30
trainSet <- adultsTrsf[1:trainPortion,]
cvDivider <- floor(nrow(trainSet) / (cv+1))

smallestError <- 100
for (depth in seq(1,10,1)) {
        for (rounds in seq(1,20,1)) {
                totalError <- c()
                indexCount <- 1
                for (cv in seq(1:cv)) {
                        # assign chunk to data test
                        dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
                        dataTest <- trainSet[dataTestIndex,]
                        # everything else to train
                        dataTrain <- trainSet[-dataTestIndex,]

                        bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                                        label = dataTrain[,outcomeName],
                                        max.depth=depth, nround=rounds,
                                        objective = "reg:linear", verbose=0)
                        gc()
                        predictions <- predict(bst, as.matrix(dataTest[,predictors]),
outputmargin=TRUE)

                        err <- rmse(as.numeric(dataTest[,outcomeName]),
as.numeric(predictions))
                        totalError <- c(totalError, err)
                }
                if (mean(totalError) < smallestError) {
                        smallestError = mean(totalError)
                        print(paste(depth,rounds,smallestError))
                }
        }
}

################################################################################
# Test both models out on full data set

trainSet <- adultsTrsf[ 1:trainPortion,]

# assign everything else to test
testSet <- adultsTrsf[(trainPortion+1):nrow(adultsTrsf),]

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                label = trainSet[,outcomeName],
                max.depth=4, nround=19, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
```

```
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=3, nround=20, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))
```

Прочитайте xgboost онлайн: https://riptutorial.com/ru/r/topic/3239/xgboost

глава 27: Агрегирование кадров данных

Вступление

Агрегация является одним из наиболее распространенных способов использования R. В R есть несколько способов сделать это, что мы проиллюстрируем здесь.

Examples

Агрегация с базой R

Для этого мы будем использовать агрегат функции, который можно использовать следующим образом:

```
aggregate(formula,function,data)
```

Следующий код показывает различные способы использования агрегатной функции.

КОД:

```
df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))

# sum, grouping by one column
aggregate(value~group, FUN=sum, data=df)

# mean, grouping by one column
aggregate(value~group, FUN=mean, data=df)

# sum, grouping by multiple columns
aggregate(value~group+subgroup,FUN=sum,data=df)

# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
```

ВЫХОД:

```
> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
    group subgroup value
1 Group 1        A   2.0
2 Group 1        A   2.5
3 Group 2        A   1.0
4 Group 2        A   2.0
5 Group 2        B   1.5
>
> # sum, grouping by one column
> aggregate(value~group, FUN=sum, data=df)
```

```
    group value
1 Group 1   4.5
2 Group 2   4.5
>
> # mean, grouping by one column
> aggregate(value~group, FUN=mean, data=df)
    group value
1 Group 1  2.25
2 Group 2  1.50
>
> # sum, grouping by multiple columns
> aggregate(value~group+subgroup,FUN=sum,data=df)
    group subgroup value
1 Group 1        A   4.5
2 Group 2        A   3.0
3 Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
    group value
1 Group 1   2.5
2 Group 2   0.0
```

Агрегатирование с помощью dplyr

Агрегация с dplyr проста! Для этого вы можете использовать функции group_by () и summary (). Ниже приведены некоторые примеры.

КОД:

```
# Aggregating with dplyr
library(dplyr)

df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(df)

# sum, grouping by one column
df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()

# mean, grouping by one column
df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()

# sum, grouping by multiple columns
df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()

# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
```

ВЫХОД:

```
> library(dplyr)
>
> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
```

```
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
    group subgroup value
1 Group 1        A   2.0
2 Group 1        A   2.5
3 Group 2        A   1.0
4 Group 2        A   2.0
5 Group 2        B   1.5
>
> # sum, grouping by one column
> df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()
    group value
1 Group 1   4.5
2 Group 2   4.5
>
> # mean, grouping by one column
> df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()
    group value
1 Group 1  2.25
2 Group 2  1.50
>
> # sum, grouping by multiple columns
> df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()
    group subgroup value
1 Group 1        A   4.5
2 Group 2        A   3.0
3 Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
    group value
1 Group 1   2.5
2 Group 2   0.0
```

Агрегация с помощью data.table

Группировка с пакетом data.table выполняется с использованием синтаксиса `dt[i, j, by]` который можно прочитать вслух, как: « *Возьмите dt, подмножество строк с помощью i, затем вычислите j, сгруппированные по.* » Внутри оператора dt , несколько списков или групп должны быть помещены в список. Поскольку псевдоним для `list()` is `.()` , Оба могут использоваться взаимозаменяемо. В приведенных ниже примерах мы используем `.()` .

КОД:

```
# Aggregating with data.table
library(data.table)

dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(dt)

# sum, grouping by one column
dt[,.(value=sum(value)),group]

# mean, grouping by one column
dt[,.(value=mean(value)),group]
```

```
# sum, grouping by multiple columns
dt[,.(value=sum(value)),.(group,subgroup)]

# custom function, grouping by one column
# in this example we want the sum of all values larger than 2 per group.
dt[,.(value=sum(value[value>2])),group]
```

ВЫХОД:

```
> # Aggregating with data.table
> library(data.table)
>
> dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(dt)
     group subgroup value
1: Group 1        A   2.0
2: Group 1        A   2.5
3: Group 2        A   1.0
4: Group 2        A   2.0
5: Group 2        B   1.5
>
> # sum, grouping by one column
> dt[,.(value=sum(value)),group]
     group value
1: Group 1   4.5
2: Group 2   4.5
>
> # mean, grouping by one column
> dt[,.(value=mean(value)),group]
     group value
1: Group 1  2.25
2: Group 2  1.50
>
> # sum, grouping by multiple columns
> dt[,.(value=sum(value)),.(group,subgroup)]
     group subgroup value
1: Group 1        A   4.5
2: Group 2        A   3.0
3: Group 2        B   1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> dt[,.(value=sum(value[value>2])),group]
     group value
1: Group 1   2.5
2: Group 2   0.0
```

Прочитайте Агрегирование кадров данных онлайн: https://riptutorial.com/ru/r/topic/10792/агрегирование-кадров-данных

глава 28: Алгоритм случайного леса

Вступление

RandomForest - это ансамблевый метод классификации или регрессии, который уменьшает вероятность переназначения данных. Подробные сведения о методе можно найти в статье Википедии о случайных лесах . Основная реализация для R находится в пакете randomForest, но есть и другие реализации. См. Представление CRAN по компьютерному обучению .

Examples

Основные примеры - классификация и регрессия

```
######  Used for both Classification and Regression examples
library(randomForest)
library(car)            ## For the Soils data
data(Soils)

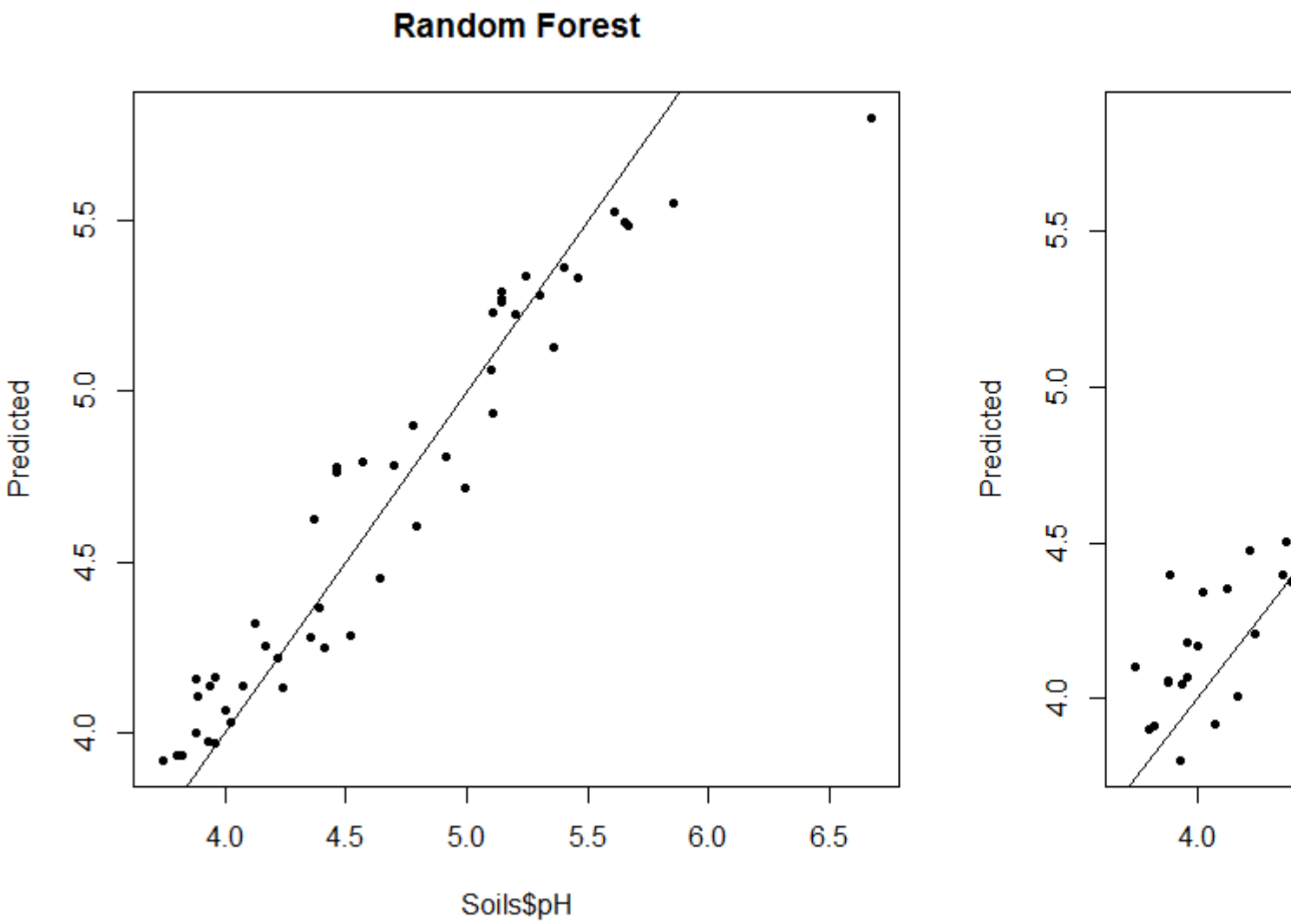
#######################################################
##    RF Classification Example
set.seed(656)            ## for reproducibility
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp))            ## No Errors

## Naive Bayes for comparison
library(e1071)
S_NB  = naiveBayes(Soils[,6:14], Soils[,4])
Gp_NB = predict(S_NB, Soils[,6:14], type="class")
length(which(Gp_NB != Soils$Gp))            ## 6 Errors
```

Этот пример проверен на данных обучения, но показывает, что РФ может создавать очень хорошие модели.

```
#######################################################
##    RF Regression Example
set.seed(656)            ## for reproducibility
S_RF_Reg = randomForest(pH ~ ., data=Soils[,6:14])
pH_RF = predict(S_RF_Reg, Soils[,6:14])

## Compare Predictions with Actual values for RF and Linear Model
S_LM = lm(pH ~ ., data=Soils[,6:14])
pH_LM = predict(S_LM, Soils[,6:14])
par(mfrow=c(1,2))
plot(Soils$pH, pH_RF, pch=20, ylab="Predicted", main="Random Forest")
abline(0,1)
plot(Soils$pH, pH_LM, pch=20, ylab="Predicted", main="Linear Model")
abline(0,1)
```



Прочитайте Алгоритм случайного леса онлайн: https://riptutorial.com/ru/r/topic/8088/алгоритм-случайного-леса

глава 29: Анализ выживаемости

Examples

Случайный анализ выживания леса с помощью randomForestSRC

Подобно тому, как алгоритм случайного леса может применяться к задачам регрессии и классификации, он также может быть расширен для анализа выживаемости.

В приведенном ниже примере модель выживания подходит и используется для прогнозирования, оценки и анализа производительности с использованием пакета `randomForestSRC` из CRAN .

```
require(randomForestSRC)

set.seed(130948) #Other seeds give similar comparative results
x1   <- runif(1000)
y    <- rnorm(1000, mean = x1, sd = .3)
data <- data.frame(x1 = x1, y = y)
head(data)
```

```
          x1          y
1 0.9604353  1.3549648
2 0.3771234  0.2961592
3 0.7844242  0.6942191
4 0.9860443  1.5348900
5 0.1942237  0.4629535
6 0.7442532 -0.0672639
```

```
(modRFSRC <- rfsrc(y ~ x1, data = data, ntree=500, nodesize = 5))
```

```
                            Sample size: 1000
                     Number of trees: 500
          Minimum terminal node size: 5
       Average no. of terminal nodes: 208.258
No. of variables tried at each split: 1
              Total no. of variables: 1
                           Analysis: RF-R
                             Family: regr
                     Splitting rule: mse
                % variance explained: 32.08
                          Error rate: 0.11
```

```
x1new   <- runif(10000)
ynew    <- rnorm(10000, mean = x1new, sd = .3)
newdata <- data.frame(x1 = x1new, y = ynew)

survival.results <- predict(modRFSRC, newdata = newdata)
survival.results
```

```
  Sample size of test (predict) data: 10000
               Number of grow trees: 500
  Average no. of grow terminal nodes: 208.258
         Total no. of grow variables: 1
                             Analysis: RF-R
                               Family: regr
                 % variance explained: 34.97
                  Test set error rate: 0.11
```

Введение - базовая подгонка и построение моделей параметрической выживаемости с пакетом выживания

`survival` является наиболее часто используемым пакетом для анализа выживаемости в R. Используя встроенный набор данных `lung` мы можем начать с анализа выживаемости, установив регрессионную модель с `survreg()` , создав кривую с `survfit()` и `survfit()` прогнозные предсказания кривые выживаемости, вызывая метод `predict` для этого пакета с новыми данными.

В приведенном ниже примере мы изображаем 2 предсказанных кривых и меняем `sex` между двумя наборами новых данных, чтобы визуализировать его эффект:

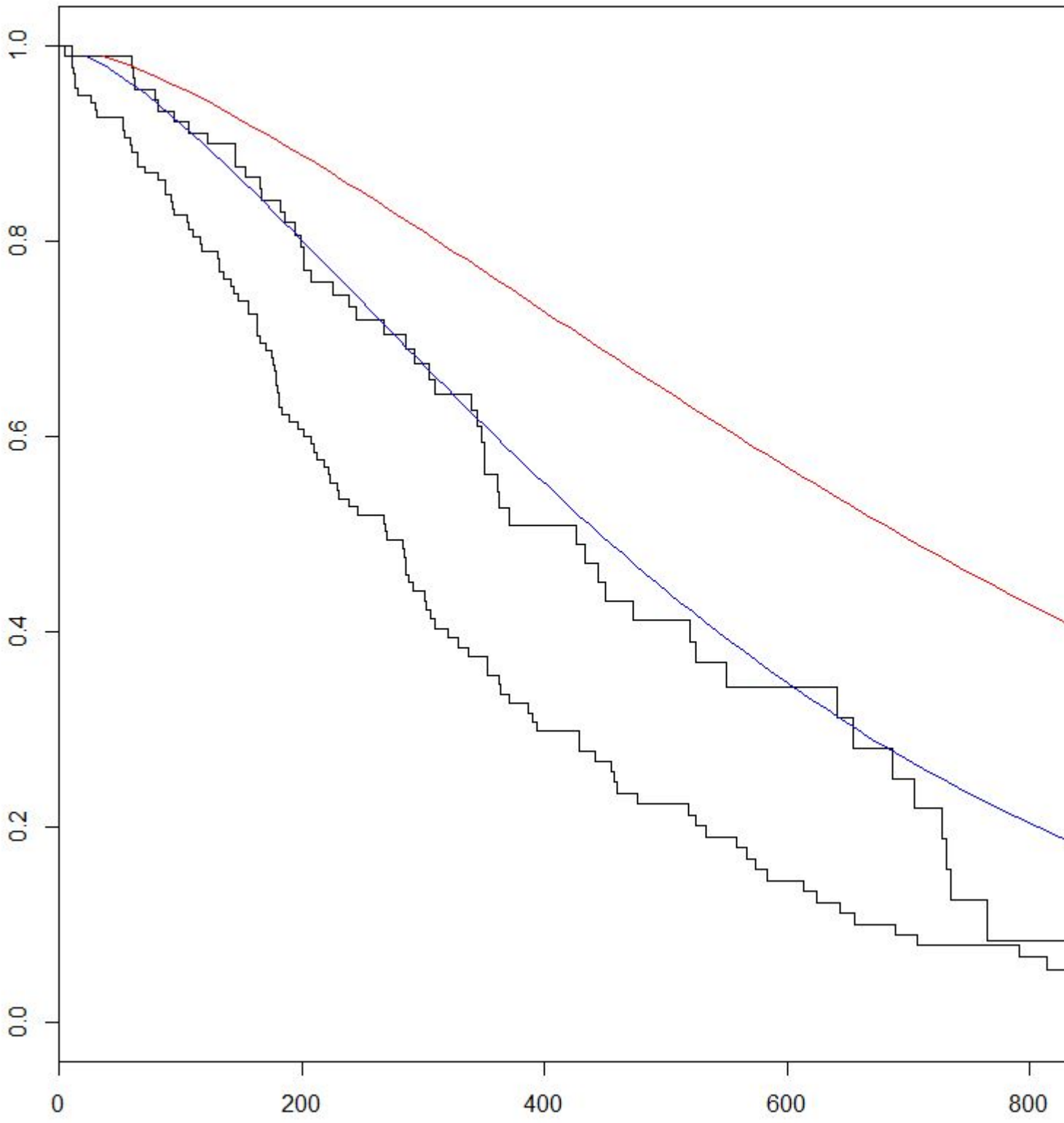
```
require(survival)
s <- with(lung,Surv(time,status))

sWei  <- survreg(s ~ as.factor(sex)+age+ph.ecog+wt.loss+ph.karno,dist='weibull',data=lung)

fitKM <- survfit(s ~ sex,data=lung)
plot(fitKM)

lines(predict(sWei, newdata = list(sex      = 1,
                                   age      = 1,
                                   ph.ecog  = 1,
                                   ph.karno = 90,
                                   wt.loss  = 2),
                                 type = "quantile",
                                 p    = seq(.01, .99, by = .01)),
                                 seq(.99, .01, by        =-.01),
                                 col = "blue")

lines(predict(sWei, newdata = list(sex      = 2,
                                   age      = 1,
                                   ph.ecog  = 1,
                                   ph.karno = 90,
                                   wt.loss  = 2),
                                 type = "quantile",
                                 p    = seq(.01, .99, by = .01)),
                                 seq(.99, .01, by        =-.01),
                                 col = "red")
```

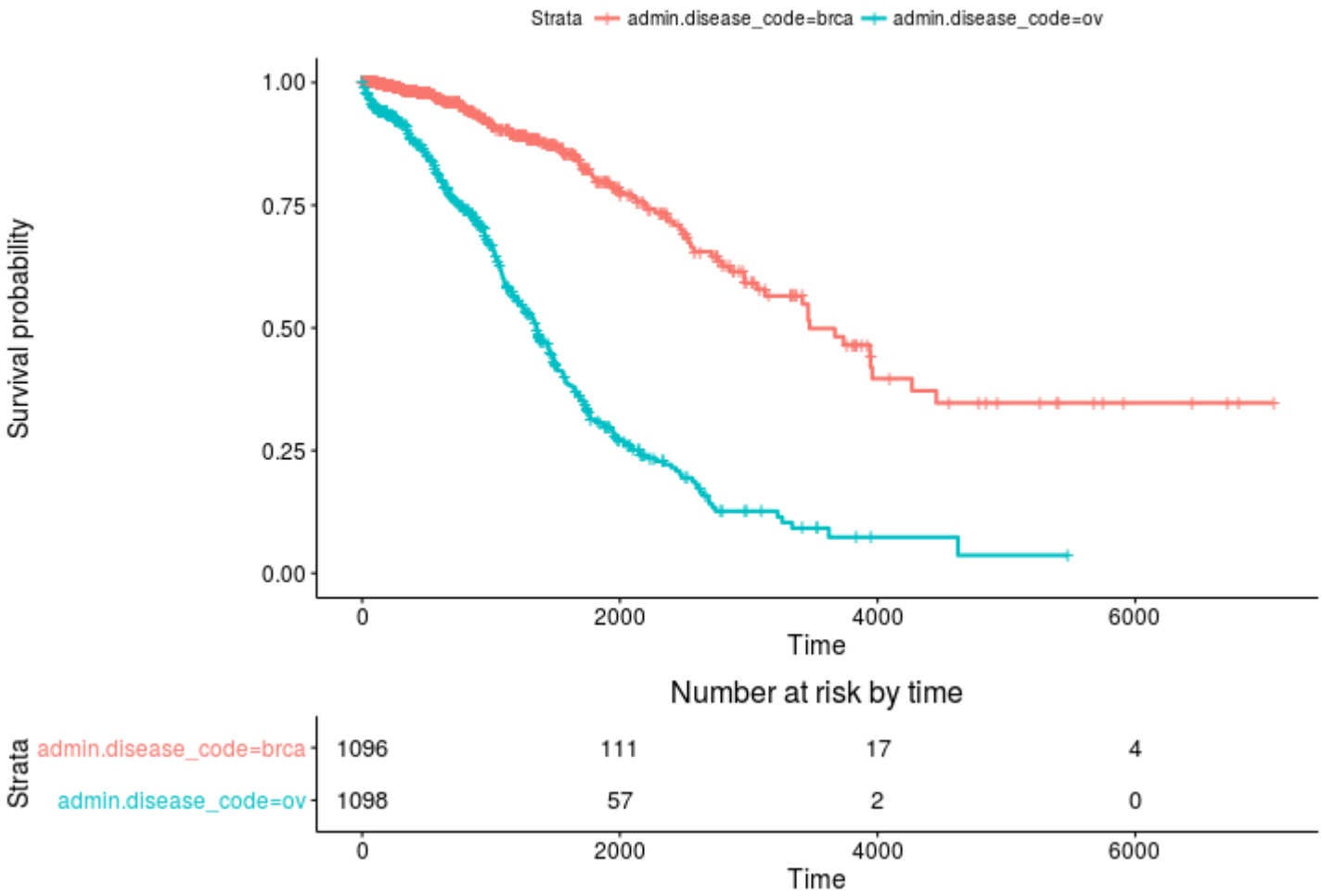


Оценка Каплана Мейера кривых выживаемости и таблиц риска с выжившим

Базовый участок

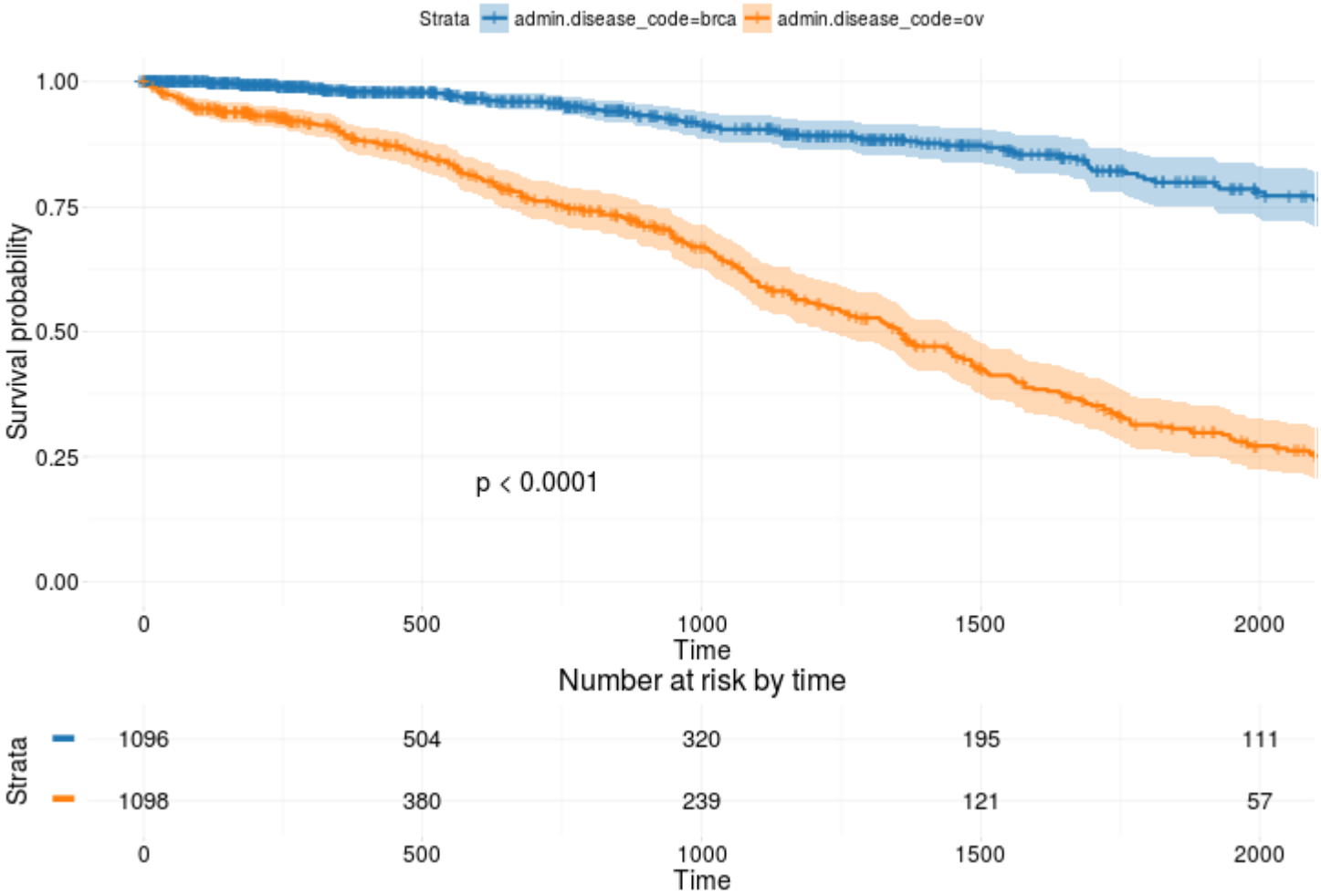
```
install.packages('survminer')
```

```
source("https://bioconductor.org/biocLite.R")
biocLite("RTCGA.clinical") # data for examples
library(RTCGA.clinical)
survivalTCGA(BRCA.clinical, OV.clinical,
             extract.cols = "admin.disease_code") -> BRCAOV.survInfo
library(survival)
fit <- survfit(Surv(times, patient.vital_status) ~ admin.disease_code,
               data = BRCAOV.survInfo)
library(survminer)
ggsurvplot(fit, risk.table = TRUE)
```



Более продвинутый

```
ggsurvplot(
   fit,                     # survfit object with calculated statistics.
   risk.table = TRUE,       # show risk table.
   pval = TRUE,             # show p-value of log-rank test.
   conf.int = TRUE,         # show confidence intervals for
                            # point estimaes of survival curves.
   xlim = c(0,2000),        # present narrower X axis, but not affect
                            # survival estimates.
   break.time.by = 500,     # break X axis in time intervals by 500.
   ggtheme = theme_RTCGA(), # customize plot and risk table with a theme.
 risk.table.y.text.col = T, # colour risk table text annotations.
  risk.table.y.text = FALSE # show bars instead of names in text annotations
                            # in legend of risk table
)
```



Основано на

http://r-addict.com/2016/05/23/Informative-Survival-Plots.html

Прочитайте Анализ выживаемости онлайн: https://riptutorial.com/ru/r/topic/3788/анализ-выживаемости

глава 30: Анализ твитов с R

Вступление

(Необязательно). В каждой теме основное внимание уделяется. Расскажите читателям, что они найдут здесь, и пусть будущие участники узнают, что принадлежит.

Examples

Загрузить твиты

Первое, что вам нужно сделать, это скачать твиты. Вам необходимо настроить свой счет в твитере. В Интернете можно найти много информации о том, как это сделать. Следующие две ссылки были полезны для моей установки (последний проверен в мае 2017 года)

В частности, я нашел следующие полезные ссылки (последний проверен в мае 2017 года):

Ссылка 1

Ссылка 2

Библиотеки R

Вам понадобятся следующие пакеты R

```
library("devtools")
library("twitteR")
library("ROAuth")
```

Предположим, что у вас есть ключи. Вам нужно запустить следующий код

```
api_key <- XXXXXXXXXXXXXXXXXXXXX
api_secret <- XXXXXXXXXXXXXXXXXXXXXX
access_token <- XXXXXXXXXXXXXXXXXXXXX
access_token_secret <- XXXXXXXXXXXXXXXXXXXXX


setup_twitter_oauth(api_key,api_secret)
```

Измените `XXXXXXXXXXXXXXXXXXXXX` на свои ключи (если у вас есть настройка вашего счета в твитере, вы знаете, какие ключи я имею в виду).

Предположим теперь, что мы хотим скачать твиты на кофе. Следующий код будет делать это

```
search.string <- "#coffee"
no.of.tweets <- 1000

c_tweets <- searchTwitter(search.string, n=no.of.tweets, lang="en")
```

Вы получите 1000 твитов на «кофе».

Получить текст твитов

Теперь нам нужно получить доступ к тексту твитов. Таким образом, мы делаем это таким образом (нам также нужно очистить твиты от специальных символов, которые на данный момент нам не нужны, например, смайлики с функцией sapply.)

```
coffee_tweets = sapply(c_tweets, function(t) t$getText())

coffee_tweets <- sapply(coffee_tweets,function(row) iconv(row, "latin1", "ASCII", sub=""))
```

и вы можете проверить свои твиты с помощью функции `head` .

```
head(coffee_tweets)
```

Прочитайте Анализ твитов с R онлайн: https://riptutorial.com/ru/r/topic/10086/анализ-твитов-с-r

глава 31: Арифметические операторы

замечания

Почти все операторы в R действительно являются функциями. Например, `+` - это функция, определенная как `function (e1, e2) .Primitive("+")` где e1 - левая часть оператора, а e2 - правая часть оператора. Это означает, что можно выполнить довольно противоречивые эффекты, маскируя `+` в базе с помощью определенной пользователем функции.

Например:

```
`+` <- function(e1, e2) {e1-e2}

> 3+10
[1] -7
```

Examples

Диапазон и дополнение

Возьмем пример добавления значения в диапазон (как это может быть сделано в цикле, например):

```
3+1:5
```

дает:

```
[1] 4 5 6 7 8
```

Это связано с тем, что оператор диапазона `:` имеет более высокий приоритет, чем оператор сложения `+` .

Что происходит во время оценки:

- `3+1:5`
- `3+c(1, 2, 3, 4, 5)` расширение оператора диапазона для создания вектора целых чисел.
- `c(4, 5, 6, 7, 8)` Добавление 3 к каждому члену вектора.

Чтобы избежать такого поведения, вы должны сообщить интерпретатору R, как вы хотите, чтобы он выполнял операции с `( )` следующим образом:

```
(3+1):5
```

Теперь R вычислит, что находится внутри круглых скобок, прежде чем расширять диапазон и дает:

```
 [1] 4 5
```

Сложение и вычитание

Основные математические операции выполняются в основном по числам или по векторам (списки чисел).

1. Использование одиночных чисел

Мы можем просто ввести числа, объединенные с `+` для *добавления* и `-` для *вычитания* :

```
> 3 + 4.5
# [1] 7.5
> 3 + 4.5 + 2
# [1] 9.5
> 3 + 4.5 + 2 - 3.8
# [1] 5.7
> 3 + NA
#[1] NA
> NA + NA
#[1] NA
> NA - NA
#[1] NA
> NaN - NA
#[1] NaN
> NaN + NA
#[1] NaN
```

Мы можем присвоить числа *переменным* (константы в этом случае) и выполнить те же операции:

```
> a <- 3; B <- 4.5; cc <- 2; Dd <- 3.8 ;na<-NA;nan<-NaN
> a + B
# [1] 7.5
> a + B + cc
# [1] 9.5
> a + B + cc - Dd
# [1] 5.7
> B-nan
#[1] NaN
> a+na-na
#[1] NA
> a + na
#[1] NA
> B-nan
#[1] NaN
> a+na-na
#[1] NA
```

2. Использование векторов

В этом случае мы создаем векторы чисел и выполняем операции с использованием этих векторов или комбинаций с одиночными числами. В этом случае операция выполняется с учетом каждого элемента вектора:

```
> A <- c(3, 4.5, 2, -3.8);
> A
# [1]  3.0  4.5  2.0 -3.8
> A + 2 # Adding a number
# [1]  5.0  6.5  4.0 -1.8
> 8 - A # number less vector
# [1]  5.0  3.5  6.0 11.8
> n <- length(A) #number of elements of vector A
> n
# [1] 4
> A[-n] + A[n] # Add the last element to the same vector without the last element
# [1] -0.8  0.7 -1.8
> A[1:2] + 3 # vector with the first two elements plus a number
# [1] 6.0 7.5
> A[1:2] - A[3:4] # vector with the first two elements less the vector with elements 3 and 4
# [1] 1.0 8.3
```

Мы также можем использовать `sum` функций для добавления всех элементов вектора:

```
> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7
```

Мы должны заботиться об *утилизации* , которая является одной из характеристик `R` , поведение, которое происходит при выполнении математических операций, когда длина векторов различна. *Более короткие векторы в выражении перерабатываются так часто, как нужно (возможно, дробно), пока они не совпадут с длиной самого длинного вектора. В частности, константа просто повторяется* . В этом случае появляется предупреждение.

```
> B <- c(3, 5, -3, 2.7, 1.8)
> B
# [1]  3.0  5.0 -3.0  2.7  1.8
> A
# [1]  3.0  4.5  2.0 -3.8
> A + B # the first element of A is repeated
# [1]  6.0  9.5 -1.0 -1.1  4.8
Warning message:
In A + B : longer object length is not a multiple of shorter object length
> B - A # the first element of A is repeated
# [1]  0.0  0.5 -5.0  6.5 -1.2
Warning message:
In B - A : longer object length is not a multiple of shorter object length
```

В этом случае правильной процедурой будет рассмотрение только элементов более короткого вектора:

```
> B[1:n] + A
# [1]  6.0  9.5 -1.0 -1.1
> B[1:n] - A
# [1]  0.0  0.5 -5.0  6.5
```

При использовании функции `sum` снова добавляются все элементы внутри функции.

```
> sum(A, B)
# [1] 15.2
> sum(A, -B)
# [1] -3.8
> sum(A)+sum(B)
# [1] 15.2
> sum(A)-sum(B)
# [1] -3.8
```

Прочитайте Арифметические операторы онлайн: https://riptutorial.com/ru/r/topic/4389/арифметические-операторы

глава 32: Барная диаграмма

Вступление

Цель штрихового графика - отображать частоты (или пропорции) уровней фактор-переменной. Например, штриховой график используется для визуального отображения частот (или пропорций) отдельных лиц в различных социально-экономических (факторных) группах (уровни - высокий, средний, низкий). Такой график поможет обеспечить визуальное сравнение между различными уровнями факторов.

Examples

Функция barplot ()

В баррете коэффициенты-уровни размещаются на оси x, а частоты (или пропорции) различных степенных уровней рассматриваются на оси y. Для каждого уровня фактора построена одна полоса равномерной ширины с высотой, пропорциональной частоте (или пропорции) уровня фактора.

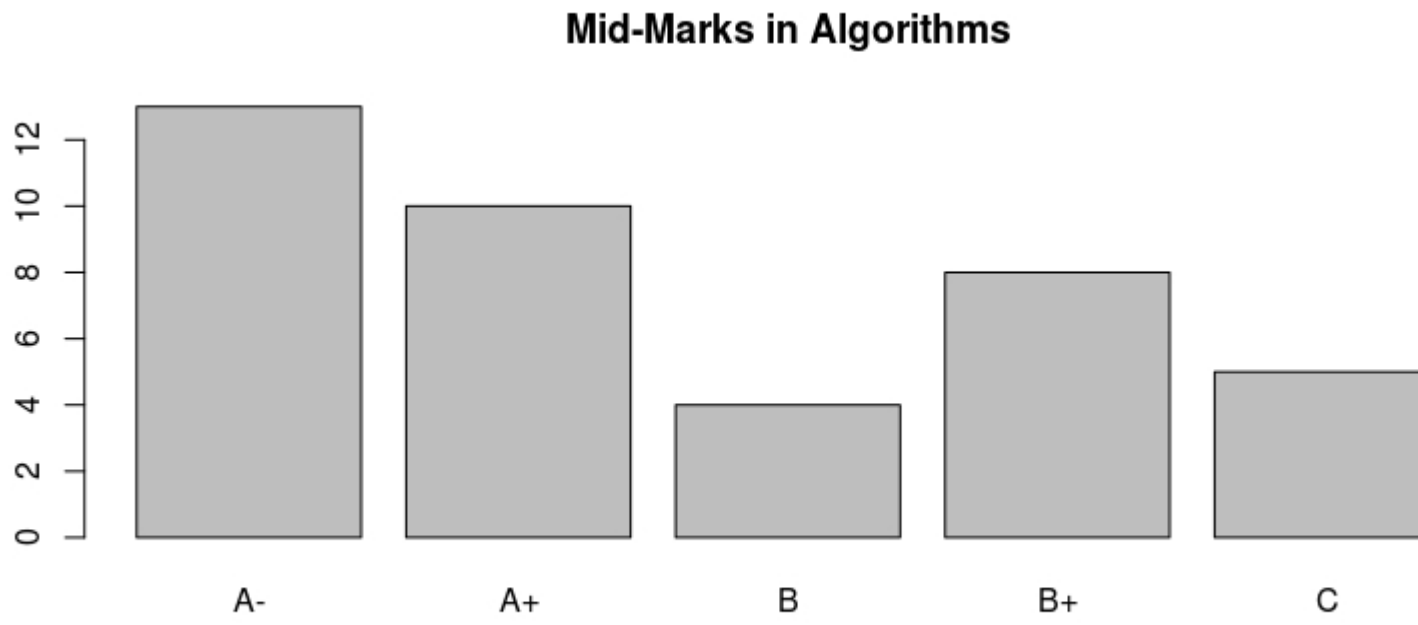
Функция `barplot()` находится в графическом пакете библиотеки системы R. Функция `barplot()` должна содержать хотя бы один аргумент. R помогает называть это как `heights` , которые должны быть либо вектором, либо матрицей. Если это вектор, его членами являются различные уровни факторов.

Чтобы проиллюстрировать `barplot()` , рассмотрим следующую подготовку данных:

```
> grades<-c("A+","A-","B+","B","C")
> Marks<-sample(grades,40,replace=T,prob=c(.2,.3,.25,.15,.1))
> Marks
[1] "A+" "A-" "B+" "A-" "A+" "B"  "A+" "B+" "A-" "B"  "A+" "A-"
[13] "A-" "B+" "A-" "A-" "A-" "A-" "A+" "A-" "A+" "A+" "C"  "C"
[25] "B"  "C"  "B+" "C"  "B+" "B+" "B+" "A+" "B+" "A-" "A+" "A-"
[37] "A-" "B"  "C"  "A+"
>
```

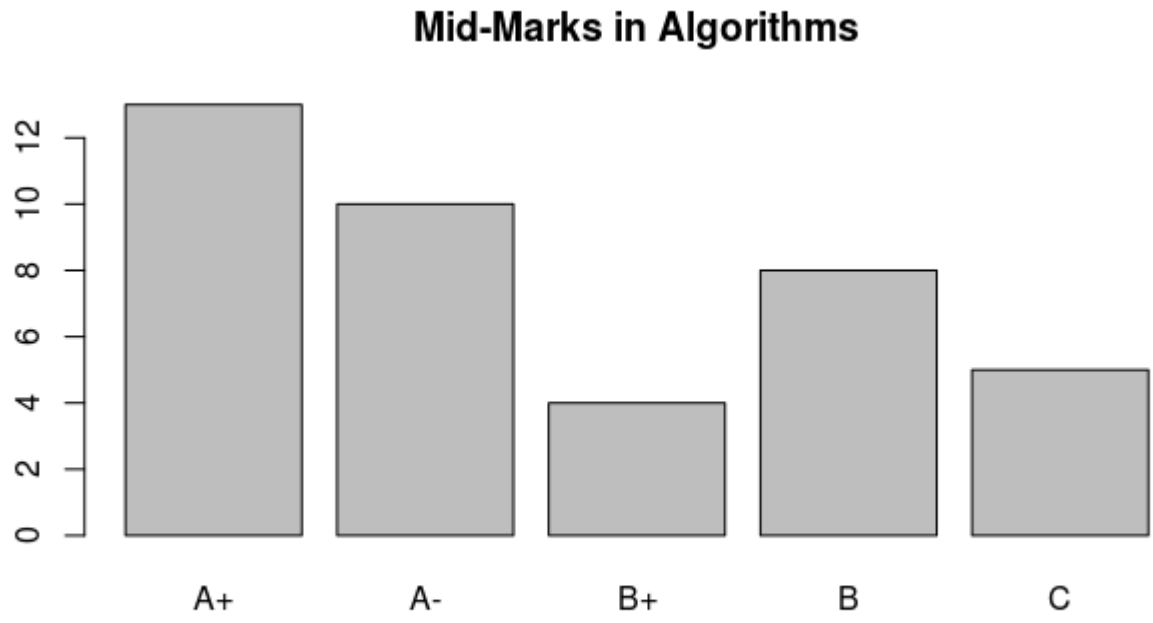
Гистограмма вектора Marks получается из

```
> barplot(table(Marks),main="Mid-Marks in Algorithms")
```



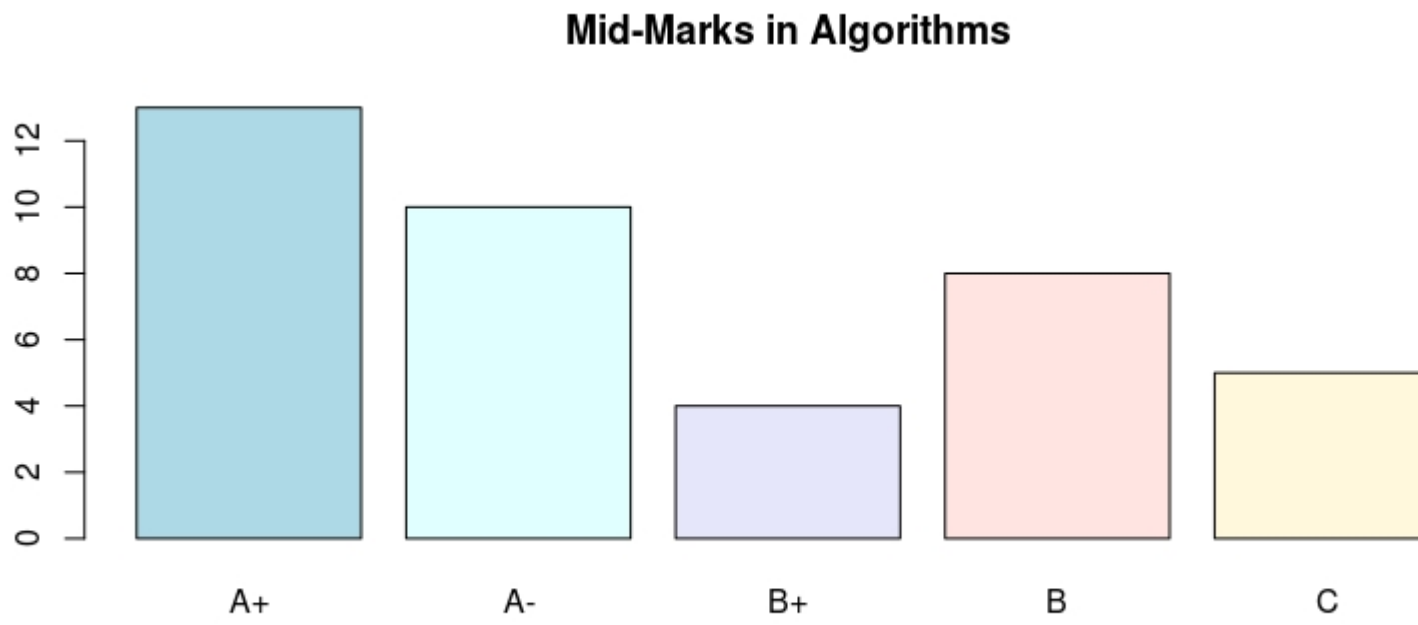
Обратите внимание, что функция barplot () помещает уровни факторов по оси x в `lexicographical order` уровней. Используя параметр `names.arg` , полосы в графике могут быть помещены в порядке, указанном в векторе, *градусах* .

```
# plot to the desired horizontal axis labels
> barplot(table(Marks),names.arg=grades ,main="Mid-Marks in Algorithms")
```



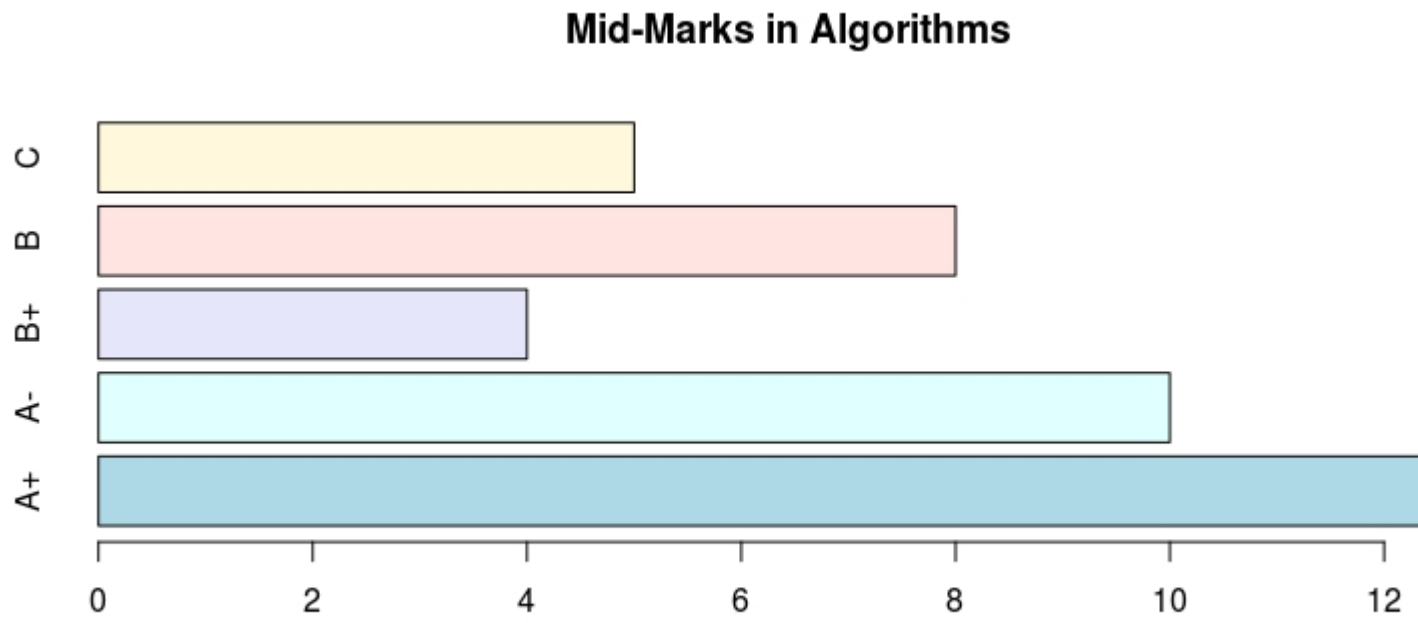
Цветные полосы могут быть нарисованы с использованием параметра `col=` .

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
        "lightcyan", "lavender", "mistyrose",  "cornsilk"),
         main="Mid-Marks in Algorithms")
```



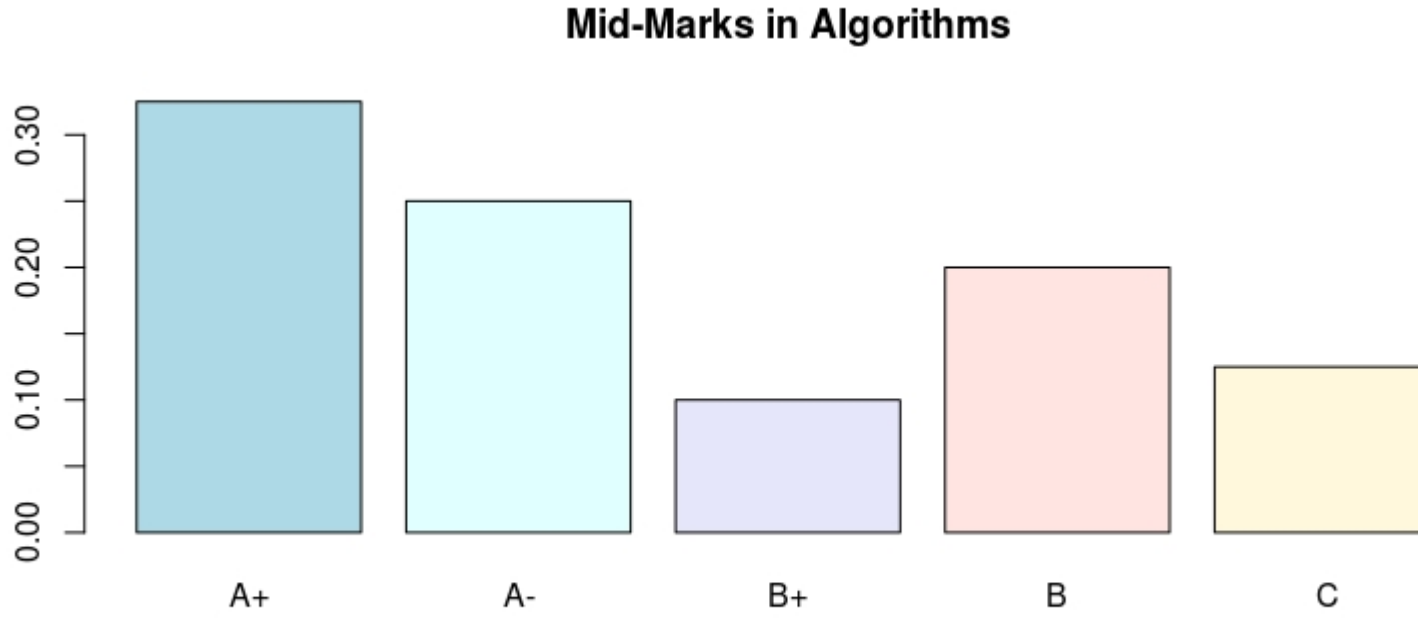
Гистограмма с *горизонтальными полосами* может быть получена следующим образом:

```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms")
```



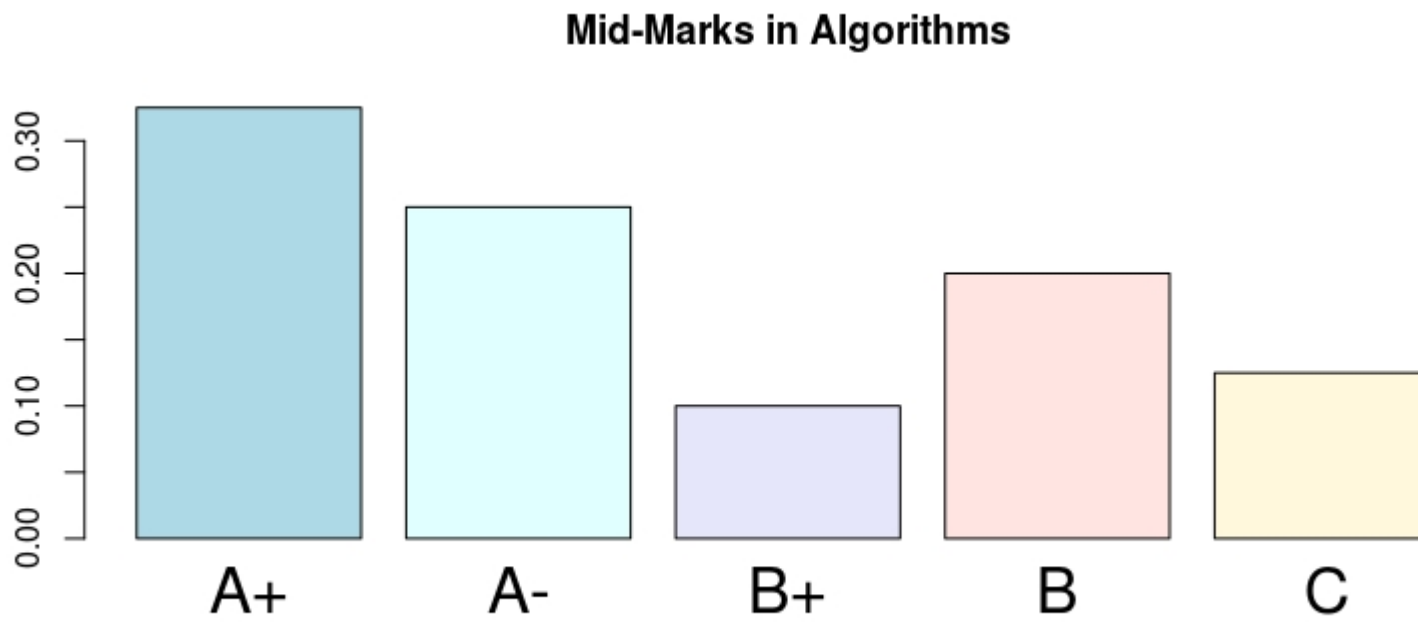
Гистограмма с *пропорциями* по оси y может быть получена следующим образом:

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
           "lightcyan", "lavender", "mistyrose",  "cornsilk"),
            main="Mid-Marks in Algorithms")
```



Размеры имен фактора-фактора по оси x могут быть увеличены с использованием параметра `cex.names` .

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
          "lightcyan", "lavender", "mistyrose",  "cornsilk"),
           main="Mid-Marks in Algorithms",cex.names=2)
```

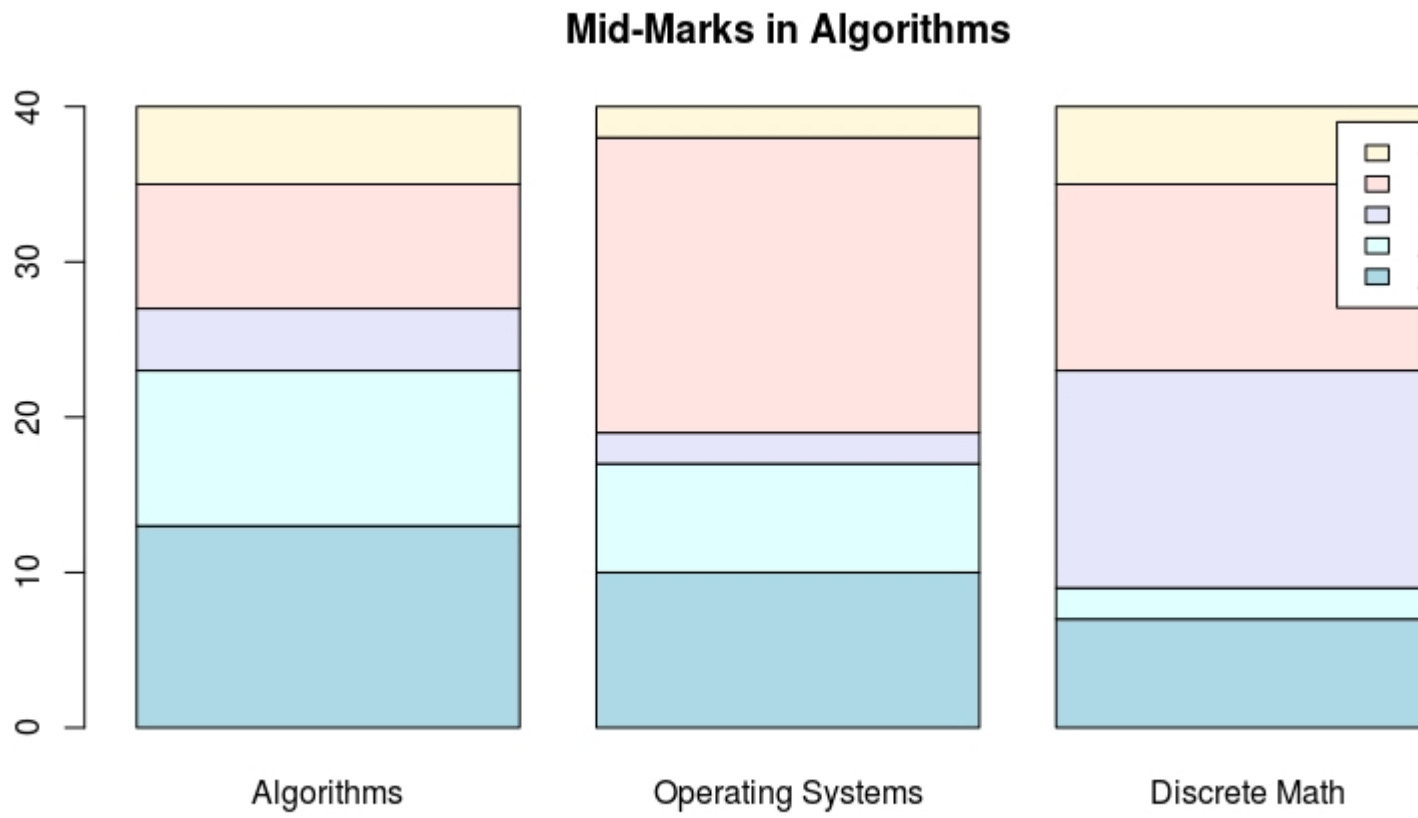


Параметр `heights barplot()` может быть матрицей. Например, это может быть матрица, где столбцы представляют собой различные предметы, взятые в курсе, строки могут быть ярлыками оценок. Рассмотрим следующую матрицу:

```
> gradTab
     Algorithms Operating Systems Discrete Math
  A-         13                10             7
  A+         10                 7             2
  B           4                 2            14
  B+          8                19            12
  C           5                 2             5
```

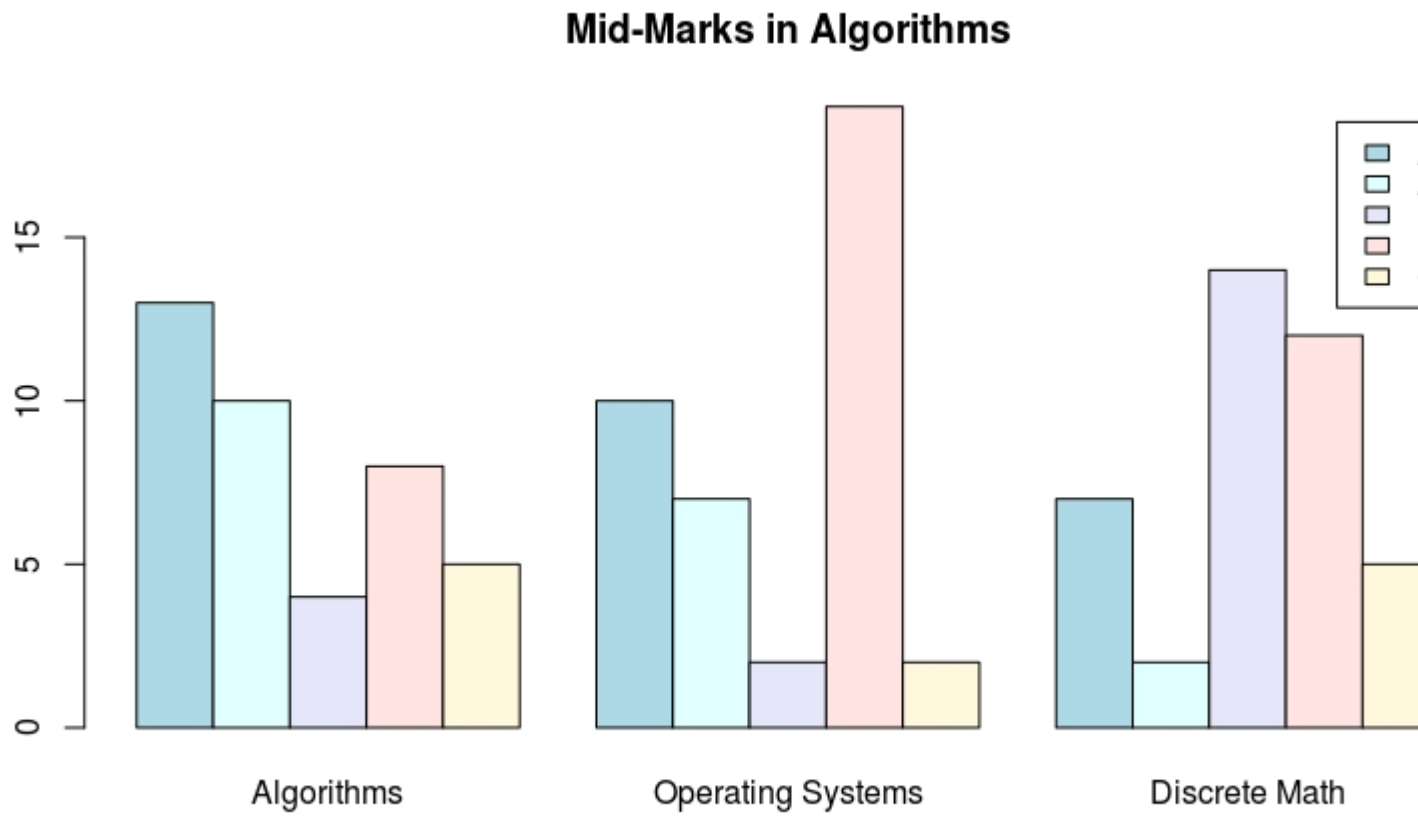
Чтобы нарисовать сложенный столбец, просто используйте команду:

```
> barplot(gradTab,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        main="Mid-Marks in Algorithms")
```



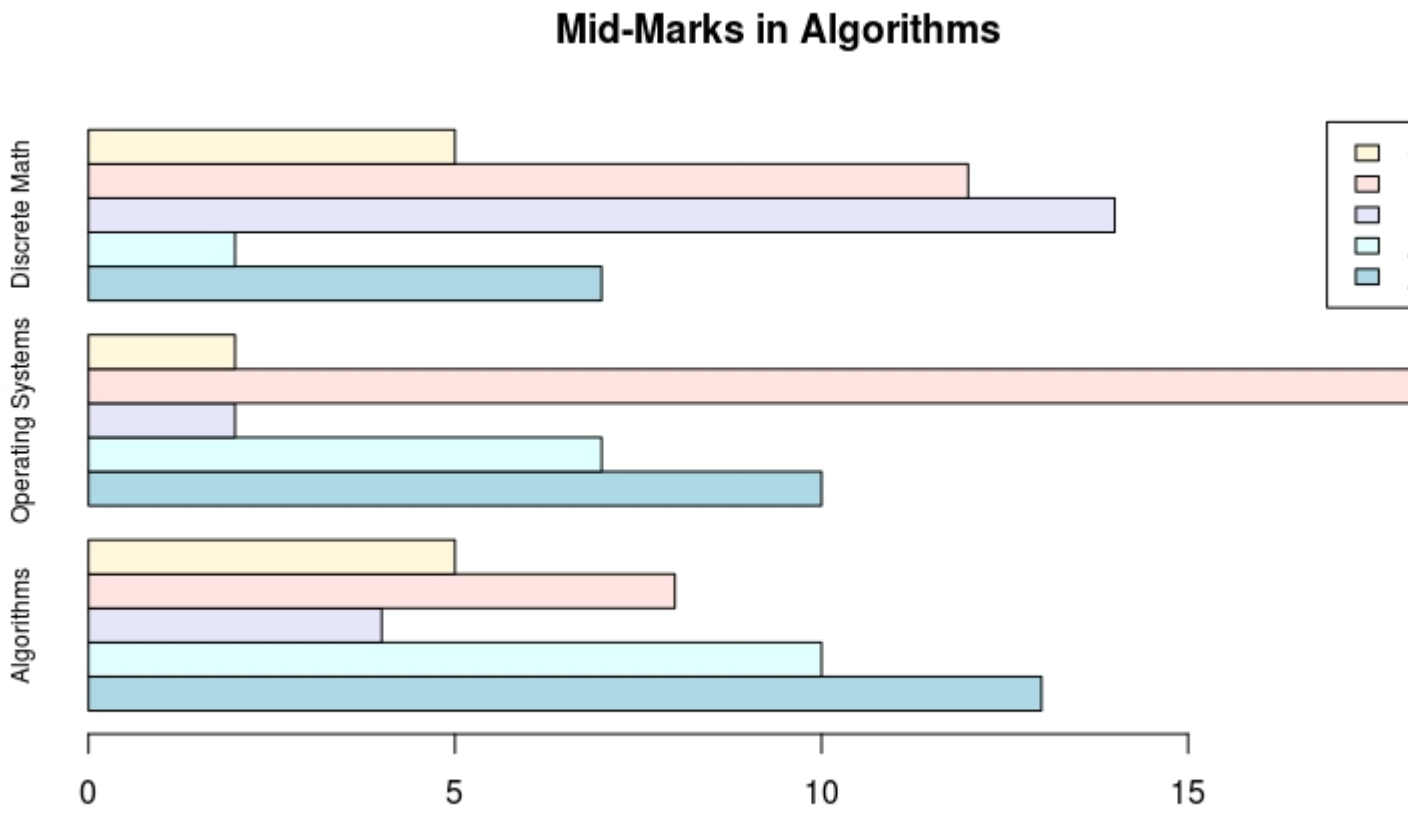
Чтобы нарисовать сопоставленные столбцы, используйте параметр « `besides` , как указано ниже:

```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
      "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
       main="Mid-Marks in Algorithms")
```



Горизонтальная гистограмма может быть получена с использованием параметра `horiz=T` :

```
> barplot(gradTab,beside = T,horiz=T,col = c("lightblue","lightcyan",
       "lavender", "mistyrose",  "cornsilk"),legend.text = grades,
        cex.names=.75,main="Mid-Marks in Algorithms")
```



Прочитайте Барная диаграмма онлайн: https://riptutorial.com/ru/r/topic/8091/барная-диаграмма

глава 33: Библиография в RMD

параметры

Параметр в заголовке YAML	подробность
`toc`	оглавление
`number_sections`	нумерация разделов автоматически
`bibliography`	путь к библиографическому файлу
`csl`	путь к файлу стиля

замечания

- Цель этой документации - объединить академическую библиографию в файле RMD.
- Чтобы использовать приведенную выше документацию, вам необходимо установить `rmarkdown` в R через `install.packages("rmarkdown")` .
- Иногда Rmarkdown удаляет гиперссылки цитат. Решение для этого добавляет следующий код в ваш заголовок YAML: `link-citations: true`
- Библиография может иметь любой из этих форматов:

Формат	Расширение файла
MODS	.mods
BibLaTeX	.bib
BibTeX	.bibtex
RIS	.ris
EndNote	.enl
XML-код EndNote	.xml
ISI	.wos
MEDLINE	.medline
Copac	.copac

Формат	Расширение файла
JSON citeproc	.json

Examples

Задание библиографии и цитирование авторов

Наиболее важной частью вашего файла RMD является заголовок YAML. Для написания академического документа я предлагаю использовать вывод PDF, пронумерованные разделы и таблицу содержимого (toc).

```
---
title: "Writing an academic paper in R"
author: "Author"
date: "Date"
output:
  pdf_document:
    number_sections: yes
toc: yes
bibliography: bibliography.bib
---
```

В этом примере наш файл `bibliography.bib` выглядит так:

```
@ARTICLE{Meyer2000,
  AUTHOR="Bernd Meyer",
  TITLE="A constraint-based framework for diagrammatic reasoning",
  JOURNAL="Applied Artificial Intelligence",
  VOLUME= "14",
  ISSUE = "4",
  PAGES= "327--344",
  YEAR=2000
}
```

Чтобы привести автора, упомянутого в вашем .bib-файле, напишите `@` и bibkey, например `Meyer2000` .

```
# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# References
```

Рендеринг RMD-файла через RStudio (Ctrl + Shift + K) или через консоль `rmarkdown::render("<path-to-your-RMD-file">)` приводит к следующему результату:

Writing an academic paper in

Author

Date

Contents

1 Introduction

`@Meyer2000` results in Meyer (2000).

`@Meyer2000 [p. 328]` results in Meyer (2000, 328)

`[@Meyer2000]` results in (Meyer 2000)

`[-@Meyer2000]` results in (2000)

2 Summary

References

Meyer, Bernd. 2000. "A Constraint-Based Framework for Diagrammatic Rea
Intelligence 14 (4): 327–44.

документа. Это должно включать в себя массив ссылок, закодированных YAML, например:

```
---
title: "Writing an academic paper in R"
author: "Author"
date: "Date"
output:
  pdf_document:
    number_sections: yes
toc: yes
references:
  - id: Meyer2000
    title: A Constraint-Based Framework for Diagrammatic Reasoning
    author:
    - family: Meyer
      given: Bernd
    volume: 14
    issue: 4
    publisher: Applied Artificial Intelligence
    page: 327-344
    type: article-journal
    issued:
      year: 2000
---

# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# References
```

Отображение этого файла приводит к тому же результату, что и в примере «Указание библиографии».

Стили стилей

По умолчанию `pandoc` будет использовать формат даты в формате Chicago для цитат и ссылок. Чтобы использовать другой стиль, вам нужно указать файл стиля CSL 1.0 в поле метаданных csl. Ниже приводится часто используемый стиль цитаты, стиль elsevier (загружается по адресу https://github.com/citation-style-language/styles). Файл стиля должен храниться в том же каталоге, что и файл RMD, или должен быть отправлен абсолютный путь к файлу.

Чтобы использовать другой стиль по умолчанию, используется следующий код:

```
---
title: "Writing an academic paper in R"
```

```
author: "Author"
date: "Date"
output:
  pdf_document:
    number_sections: yes
toc: yes
bibliography: bibliography.bib
csl: elsevier-harvard.csl
---

# Introduction

`@Meyer2000` results in @Meyer2000.

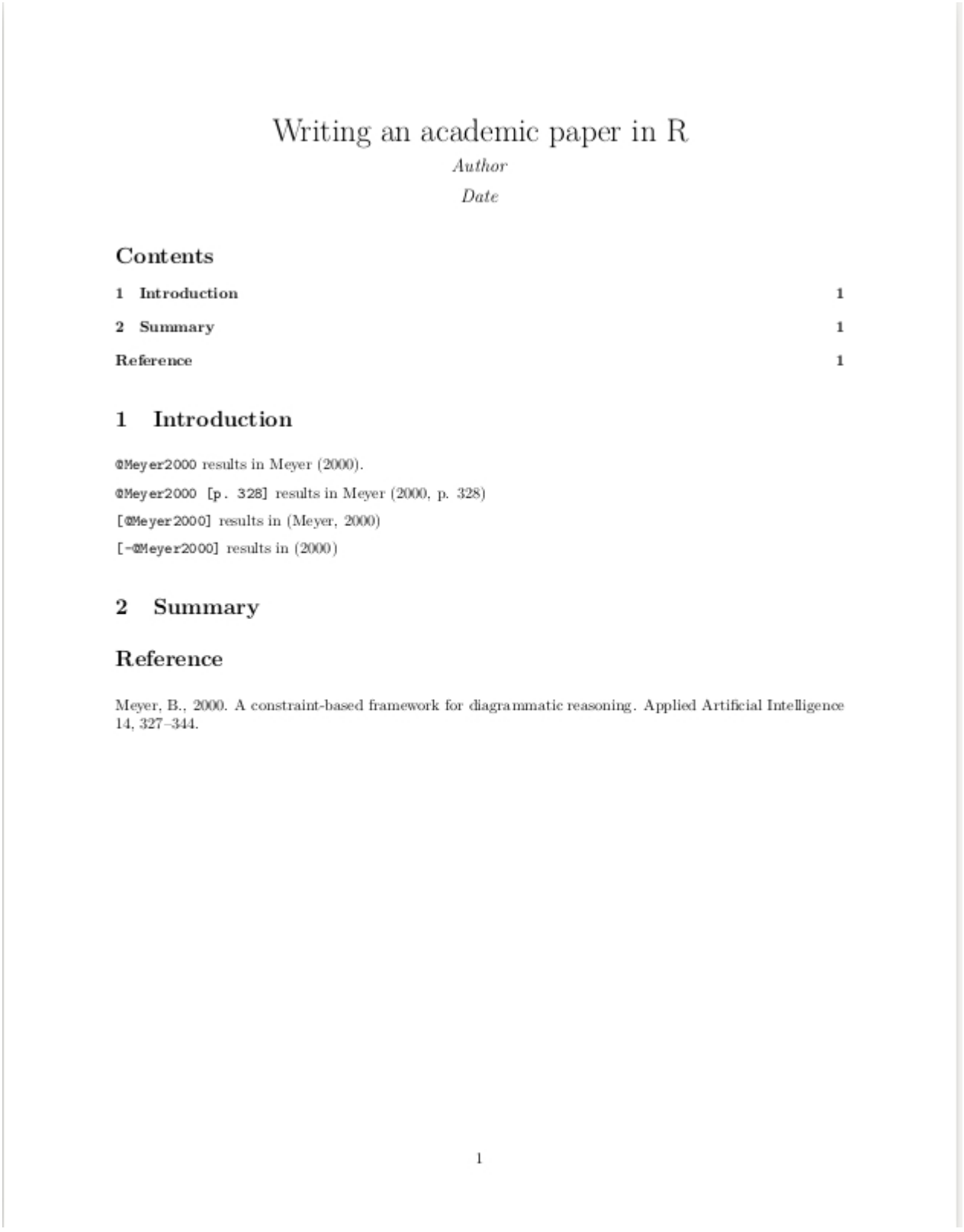
`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# Reference
```



Writing an academic paper in R

Author

Date

Contents

1 Introduction

`@Meyer2000` results in Meyer (2000).

`@Meyer2000 [p. 328]` results in Meyer (2000, p. 328)

`[@Meyer2000]` results in (Meyer, 2000)

`[-@Meyer2000]` results in (2000)

2 Summary

Reference

Meyer, B., 2000. A constraint-based framework for diagrammatic reasoning. Applied Artificial Intelligence 14, 327–344.

1

Обратите внимание на отличия от результата примера «Указание библиографии и цитирующих авторов»

Прочитайте Библиография в RMD онлайн: https://riptutorial.com/ru/r/topic/7606/библиография-в-rmd

глава 34: блестящий

Examples

Создание приложения

Shiny - это R- пакет, разработанный RStudio, который позволяет создавать веб-страницы для интерактивного отображения результатов анализа в R.

Существует два простых способа создания приложения Shiny:

- в одном `.R` файле или
- в двух файлах: `ui.R` и `server.R` .

Блестящее приложение разделено на две части:

- **ui** : скрипт пользовательского интерфейса, контролирующий макет и внешний вид приложения.
- **server** : серверный скрипт, содержащий код, позволяющий приложению реагировать.

Один файл

```
library(shiny)

# Create the UI
ui <- shinyUI(fluidPage(
    # Application title
    titlePanel("Hello World!")
))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)
```

Два файла

Создать файл `ui.R`

```
library(shiny)

# Define UI for application
shinyUI(fluidPage(
  # Application title
```

```
  titlePanel("Hello World!")
))
```

Создать файл `server.R`

```
library(shiny)

# Define server logic
shinyServer(function(input, output){})
```

Переключатель

Вы можете создать набор переключателей, используемых для выбора элемента из списка.

Можно изменить настройки:

- selected: Первоначально выбранное значение (символ (0) без выбора)
- встроенный: горизонтальный или вертикальный
- ширина

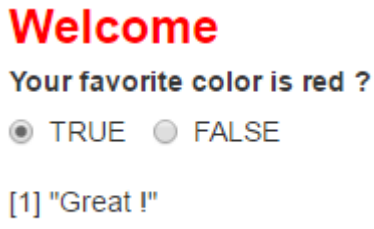
Также можно добавить HTML.

```
library(shiny)

ui <- fluidPage(
  radioButtons("radio",
               label = HTML('<FONT color="red"><FONT size="5pt">Welcome</FONT></FONT><br>
<b>Your favorite color is red ?</b>'),
               choices = list("TRUE" = 1, "FALSE" = 2),
               selected = 1,
               inline = T,
               width = "100%"),
  fluidRow(column(3, textOutput("value"))))


server <- function(input, output){
  output$value <- renderPrint({
    if(input$radio == 1){return('Great !')}
     else{return("Sorry !")}})}

shinyApp(ui = ui, server = server)
```



Группа флажков

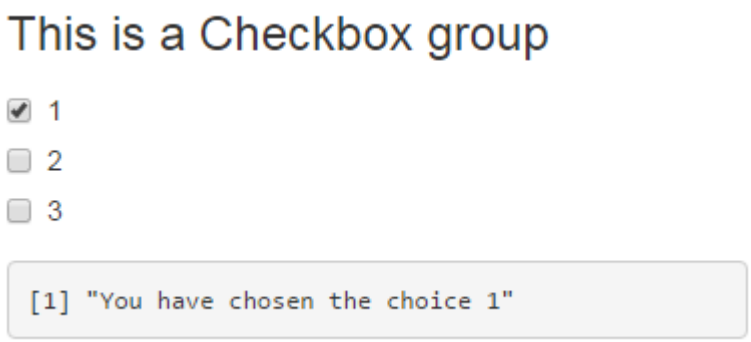
Создайте группу флажков, которые могут использоваться для одновременного переключения нескольких вариантов. Сервер получит вход в качестве символьного вектора выбранных значений.

```
library(shiny)

ui <- fluidPage(
  checkboxGroupInput("checkGroup1", label = h3("This is a Checkbox group"),
                 choices = list("1" = 1, "2" = 2, "3" = 3),
                 selected = 1),
  fluidRow(column(3, verbatimTextOutput("text_choice")))
  )


server <- function(input, output){
  output$text_choice <- renderPrint({
    return(paste0("You have chosen the choice ",input$checkGroup1))})
}

shinyApp(ui = ui, server = server)
```



Можно изменить настройки:

- label: title
- выбор: выбранные значения
- selected: Первоначально выбранное значение (NULL без выбора)
- встроенный: горизонтальный или вертикальный
- ширина

Также можно добавить HTML.

Выберите поле

Создайте список выбора, который можно использовать для выбора одного или нескольких элементов из списка значений.

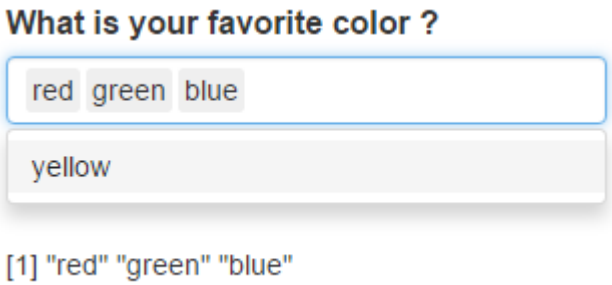
```
library(shiny)

ui <- fluidPage(
  selectInput("id_selectInput",
          label = HTML('<B><FONT size="3">What is your favorite color ?</FONT></B>'),
```

```
          multiple = TRUE,
          choices = list("red" = "red", "green" = "green", "blue" = "blue", "yellow" =
"yellow"),
          selected = NULL),
  br(), br(),
  fluidRow(column(3, textOutput("text_choice"))))


server <- function(input, output){
  output$text_choice <- renderPrint({
    return(input$id_selectInput)})
}

shinyApp(ui = ui, server = server)
```



Можно изменить настройки:

- label: title
- выбор: выбранные значения
- selected: Первоначально выбранное значение (NULL без выбора)
- multiple: TRUE или FALSE
- ширина
- размер
- selectize: TRUE или FALSE (для использования или не selectize.js, измените отображение)

Также можно добавить HTML.

Запустить блестящее приложение

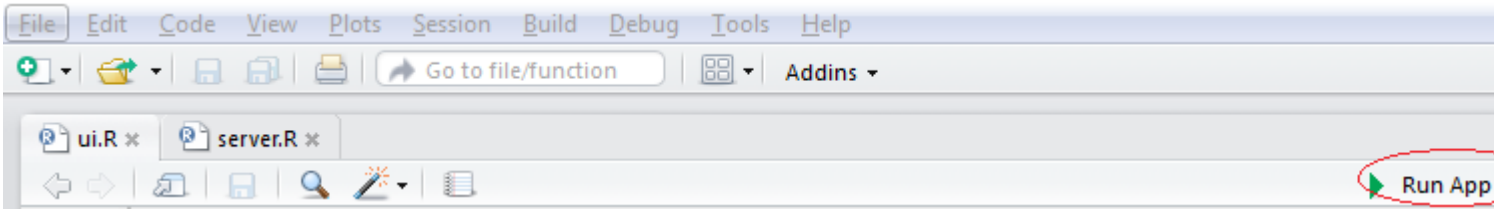
Вы можете запустить приложение несколькими способами, в зависимости от того, как вы создаете приложение. Если ваше приложение разделено на два файла `ui.R` и `server.R` или если все ваше приложение находится в одном файле.

1. Приложение для двух файлов

Два файла `ui.R` и `server.R` должны находиться в одной папке. Затем вы можете запустить свое приложение, запустив в консоли `shinyApp()` и передав путь к каталогу, содержащему приложение Shiny.

```
shinyApp("path_to_the_folder_containing_the_files")
```

Вы также можете запустить приложение непосредственно из Rstudio, нажав кнопку « **Запустить приложение»** , которая появляется в Rstudio при `server.R` файла `ui.R` или `server.R` .



Или вы можете просто написать `runApp()` на консоли, если ваш рабочий каталог - это каталог Shiny App.

2. Одно файловое приложение

Если вы создадите свой один файл `R` вы также можете запустить его с помощью функции `shinyApp()` .

- внутри вашего кода:

```
library(shiny)

ui <- fluidPage() #Create the ui
server <- function(input, output){} #create the server

shinyApp(ui = ui, server = server) #run the App
```

- в консоли, добавив путь к файлу `.R` содержащему приложение Shiny с параметром `appFile` :

```
shinyApp(appFile="path_to_my_R_file_containig_the_app")
```

Элементы управления

функция	Виджет
actionButton	Кнопка действия
checkboxGroupInput	Группа флажков
checkboxInput	Один флажок
dateInput	Календарь для выбора даты
dateRangeInput	Пара календарей для выбора диапазона дат

функция	Виджет
FileInput	Мастер управления загрузкой файлов
HelpText	Текст справки, который можно добавить в форму ввода
numericInput	Поле для ввода чисел
Радио-кнопки	Набор переключателей
selectInput	Ящик с возможностью выбора из
sliderInput	Ползунок
submitButton	Кнопка отправки
ввод текста	Поле для ввода текста

```
library(shiny)

# Create the UI
ui <- shinyUI(fluidPage(
  titlePanel("Basic widgets"),

  fluidRow(

    column(3,
           h3("Buttons"),
           actionButton("action", label = "Action"),
           br(),
           br(),
           submitButton("Submit")),

    column(3,
           h3("Single checkbox"),
           checkboxInput("checkbox", label = "Choice A", value = TRUE)),

    column(3,
           checkboxGroupInput("checkGroup",
                              label = h3("Checkbox group"),
                              choices = list("Choice 1" = 1,
                                             "Choice 2" = 2, "Choice 3" = 3),
                              selected = 1)),

    column(3,
           dateInput("date",
                     label = h3("Date input"),
                     value = "2014-01-01"))
  ),

  fluidRow(

    column(3,
           dateRangeInput("dates", label = h3("Date range"))),

    column(3,
           fileInput("file", label = h3("File input"))),
```

```
    column(3,
           h3("Help text"),
           helpText("Note: help text isn't a true widget,",
                    "but it provides an easy way to add text to",
                    "accompany other widgets.")),

    column(3,
           numericInput("num",
                        label = h3("Numeric input"),
                        value = 1))
  ),

  fluidRow(

    column(3,
           radioButtons("radio", label = h3("Radio buttons"),
                        choices = list("Choice 1" = 1, "Choice 2" = 2,
                                       "Choice 3" = 3),selected = 1)),

    column(3,
           selectInput("select", label = h3("Select box"),
                       choices = list("Choice 1" = 1, "Choice 2" = 2,
                                      "Choice 3" = 3), selected = 1)),

    column(3,
           sliderInput("slider1", label = h3("Sliders"),
                       min = 0, max = 100, value = 50),
           sliderInput("slider2", "",
                       min = 0, max = 100, value = c(25, 75))
    ),

    column(3,
           textInput("text", label = h3("Text input"),
                     value = "Enter text..."))
  )

))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)
```

отладка

`debug()` и `debugonce()` не будут работать хорошо в контексте самой блестящей отладки. Однако инструкции `browser()` вставленные в критические места, могут дать вам много информации о том, как работает ваш блестящий код. См. Также: Отладка с использованием `browser()`

Режим витрины

Режим витрины отображает ваше приложение рядом с кодом, который его генерирует, и

выделяет строки кода на сервере.R, когда он запускает их.

Существует два способа включить режим витрины:

- Запустите приложение Shiny с аргументом display.mode = "showcase", например, `runApp("MyApp", display.mode = "showcase")` .
- Создайте файл под названием `DESCRIPTION` в папке вашего Shiny app и добавьте в него эту строку: `DisplayMode: Showcase` .

Реактивный регистратор документов

Reactive Log Visualizer предоставляет интерактивный браузерный инструмент для визуализации реактивных зависимостей и исполнения в вашем приложении. Чтобы включить `options(shiny.reactlog=TRUE)` Visualizer, выполните `options(shiny.reactlog=TRUE)` в консоли R и добавьте эту строку кода в файл server.R. Чтобы запустить Reactive Log Visualizer, нажмите Ctrl + F3 в Windows или Command + F3 на Mac, когда ваше приложение запущено. Используйте клавиши со стрелками влево и вправо, чтобы перейти в диалоговое окно «Реактивный журнал».

Прочитайте блестящий онлайн: https://riptutorial.com/ru/r/topic/2044/блестящий

глава 35: Введение в географические карты

Вступление

См. Также ввод-вывод для географических данных

Examples

Базовое картографирование с помощью карты () с карт упаковки

Карта функций `map()` из `maps` пакетов обеспечивает простую отправную точку для создания карт с R.

Базовую карту мира можно сделать следующим образом:

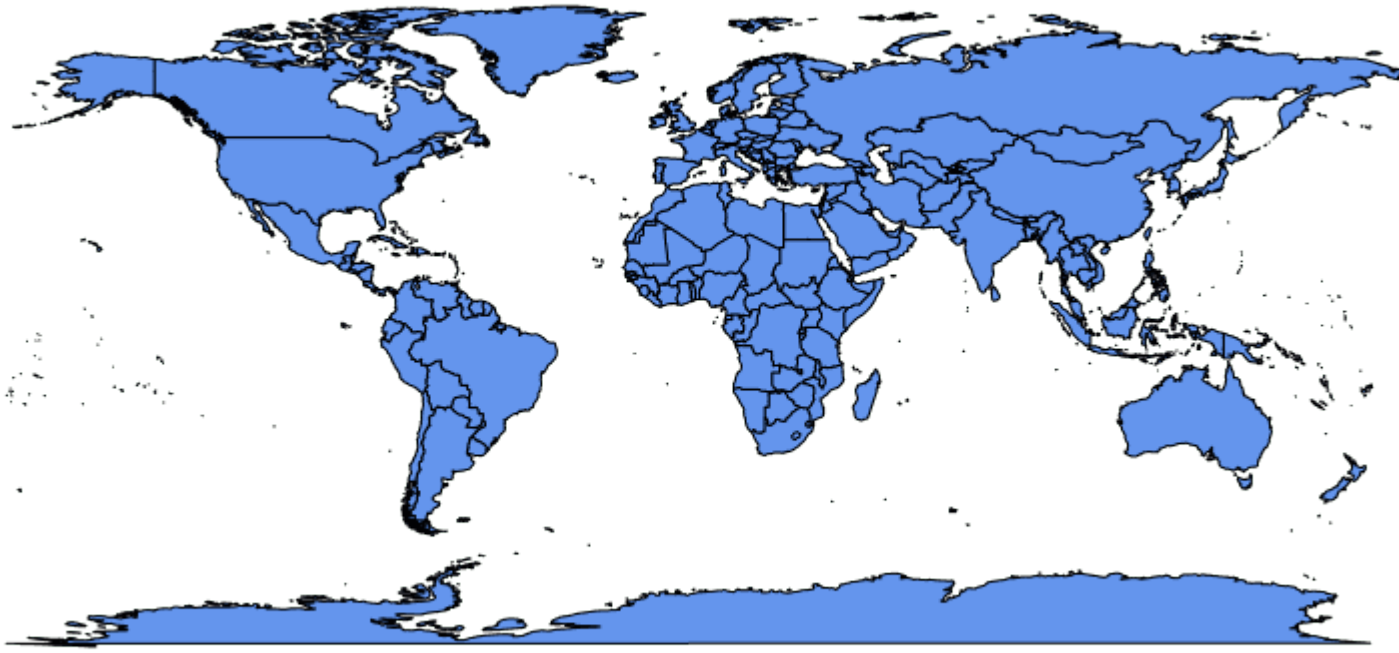
```
require(maps)
map()
```

Цвет контура можно изменить, установив параметр цвета, `col` , либо на имя символа, либо на шестнадцатеричное значение цвета:

```
require(maps)
map(col = "cornflowerblue")
```

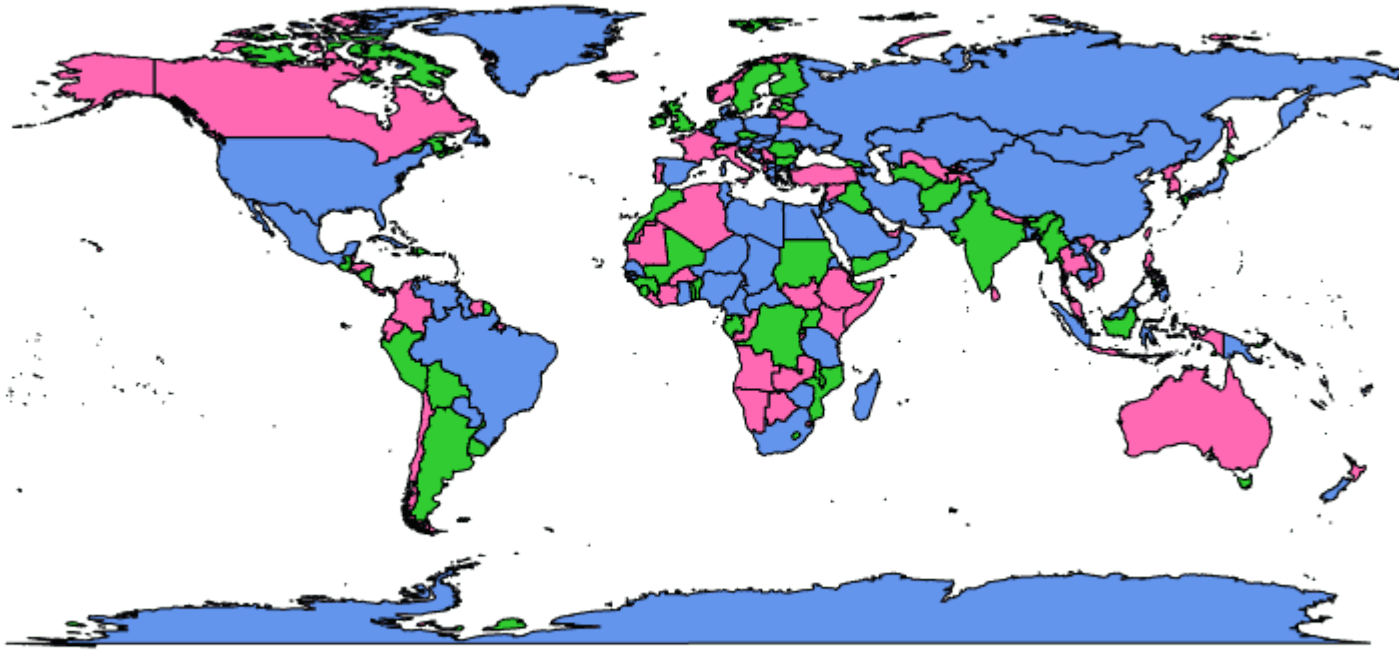
Чтобы заполнить массами земли цветом `col` мы можем установить `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue"))
```



Вектор любой длины может быть задан в `col` если также задано значение `fill = TRUE` :

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



В приведенном выше примере цвета из `col` назначаются произвольно для полигонов на карте, представляющей регионы, и цвета возвращаются, если цветов меньше, чем полигонов.

Мы также можем использовать цветовое кодирование для представления статистической переменной, которая может быть опционально описана в легенде. Карта, созданная как таковая, известна как «choropleth».

Следующий пример choropleth устанавливает первый аргумент `map()` , который представляет собой `database` для `"county"` и `"state"` для безработицы цветового кода, используя данные из встроенных наборов данных `unemp` и `county.fips` то время как `county.fips` строки состояния в белом:

```
require(maps)
if(require(mapproj)) {    # mapproj is used for  projection="polyconic"
  # color US county map by 2009 unemployment rate
  # match counties to map using FIPS county codes
  # Based on J's solution to the "Choropleth Challenge"
  # Code improvements by Hack-R (hack-r.github.io)

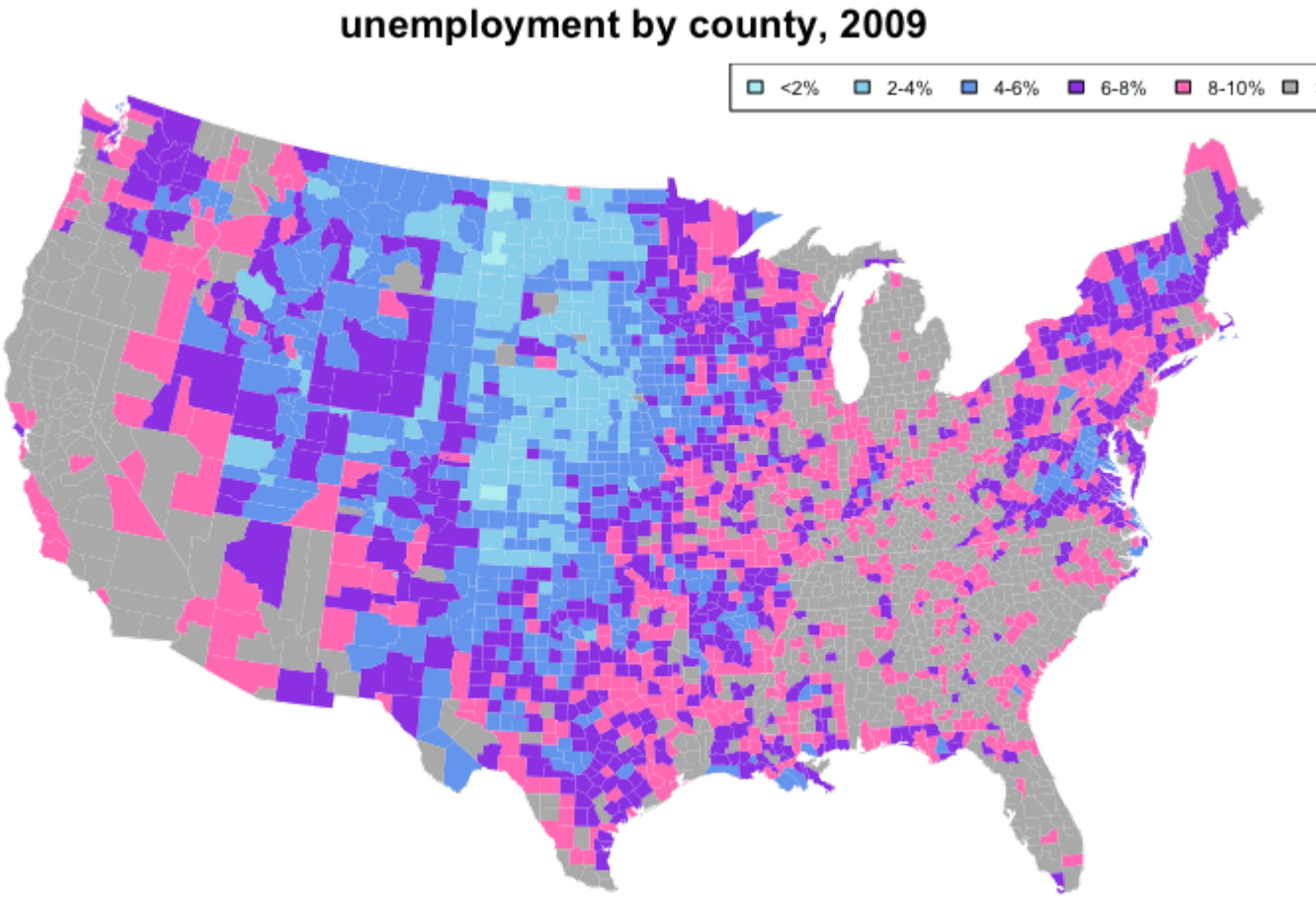
  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  #  cities
```

```
  data(unemp)
  data(county.fips)

  # define color buckets
  colors = c("paleturquoise", "skyblue", "cornflowerblue", "blueviolet", "hotpink",
"darkgrey")
  unemp$colorBuckets <- as.numeric(cut(unemp$unemp, c(0, 2, 4, 6, 8, 10, 100)))
  leg.txt <- c("<2%", "2-4%", "4-6%", "6-8%", "8-10%", ">10%")

  # align data with map definitions by (partial) matching state,county
  # names, which include multiple polygons for some counties
  cnty.fips <- county.fips$fips[match(map("county", plot=FALSE)$names,
                                      county.fips$polyname)]
  colorsmatched <- unemp$colorBuckets[match(cnty.fips, unemp$fips)]

  # draw map
  par(mar=c(1, 1, 2, 1) + 0.1)
  map("county", col = colors[colorsmatched], fill = TRUE, resolution = 0,
      lty = 0, projection = "polyconic")
  map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.1,
      projection="polyconic")
  title("unemployment by county, 2009")
  legend("topright", leg.txt, horiz = TRUE, fill = colors, cex=0.6)
}
```



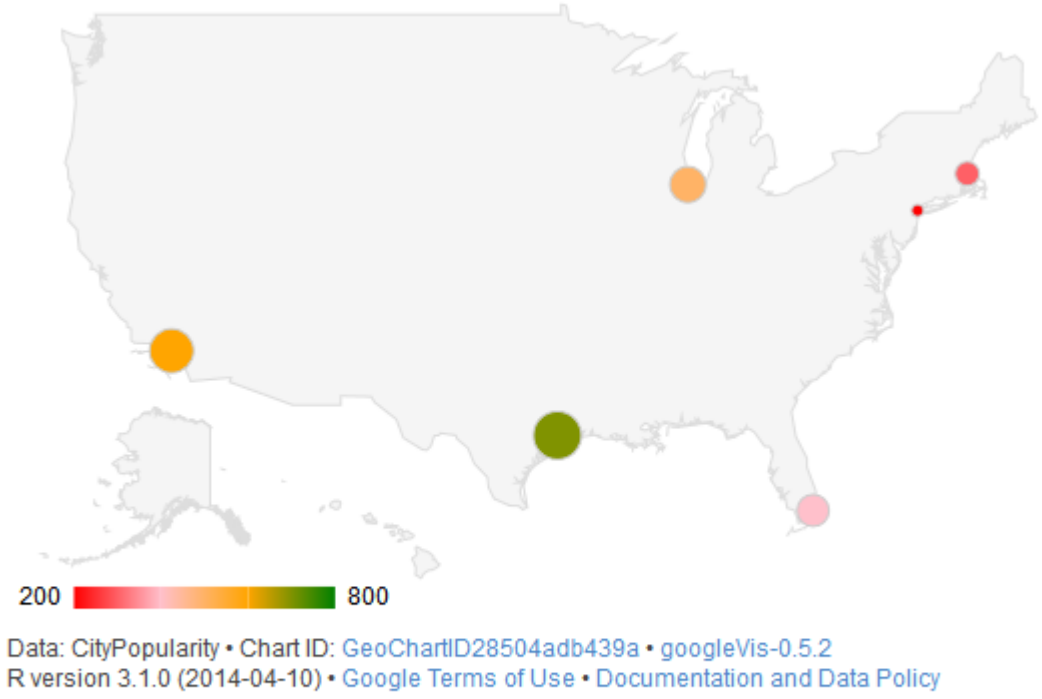
50 государственных карт и продвинутых хороптов с помощью Google Viz

Общий вопрос заключается в том, как сопоставить (объединить) физически отдельные географические регионы на одной карте, например, в случае с choropleth, описывающим все 50 американских государств (материк с Аляской и Гавайями сопоставлен).

Создание привлекательной карты состояния 50 просто при использовании Google Maps. Интерфейсы API Google включают в себя пакеты `googleVis` , `ggmap` и `RgoogleMaps` .

```
require(googleVis)

G4 <- gvisGeoChart(CityPopularity, locationvar='City', colorvar='Popularity',
                   options=list(region='US', height=350,
                                displayMode='markers',
                                colorAxis="{values:[200,400,600,800],
                                 colors:[\'red', \'pink\', \'orange',\'green']}")
)
plot(G4)
```



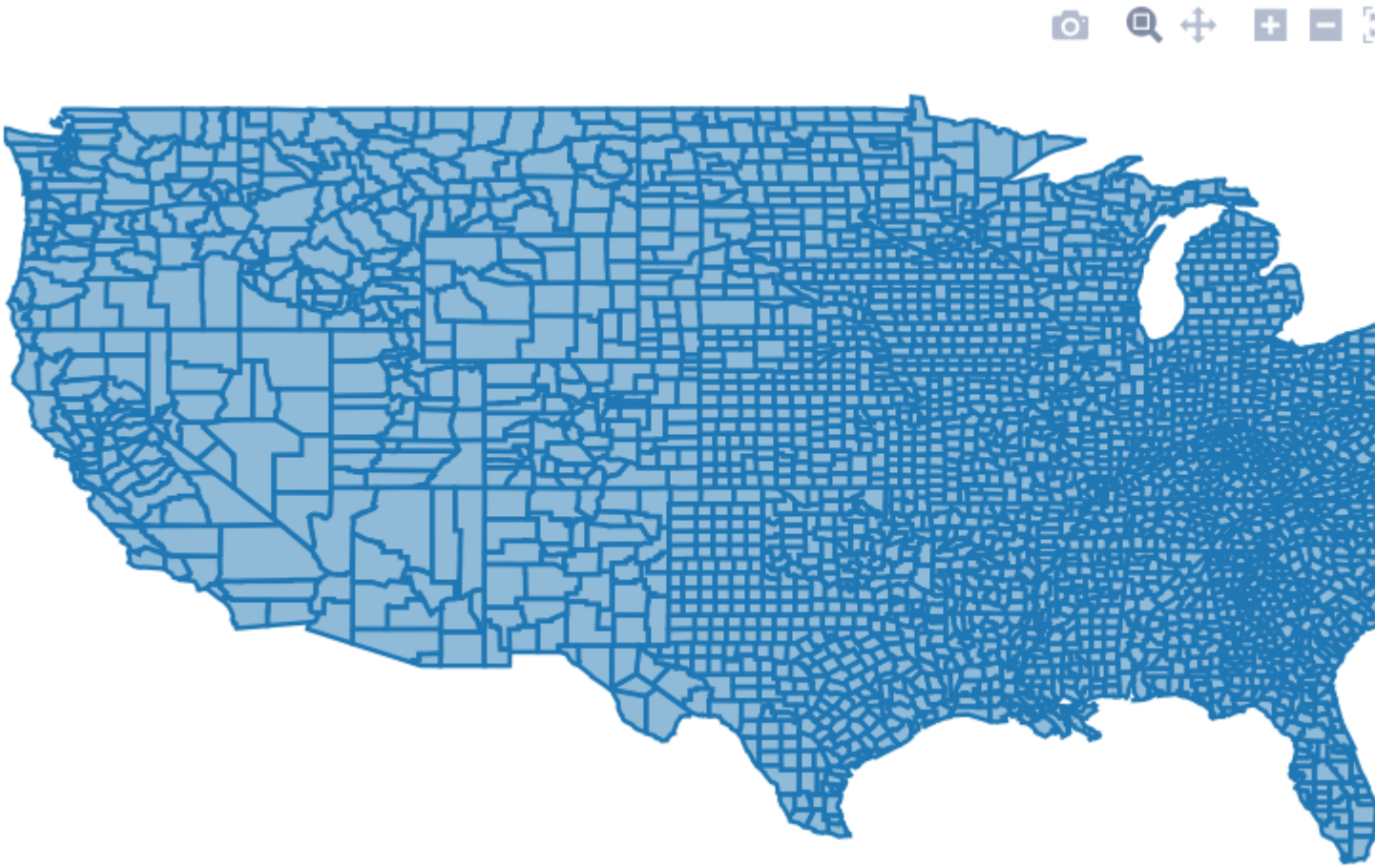
Функция `gvisGeoChart()` требует гораздо меньше кодирования для создания choropleth по сравнению с более старыми методами сопоставления, такими как `map()` из `maps` пакетов. Параметр `colorvar` позволяет легко раскрашивать статистическую переменную на уровне, указанном параметром `locationvar` . Различные параметры , передаваемые `options` в виде списка позволяют настроить детали карте, такие как размер (`height`), форма (`markers`), и цветовое кодирование (`colorAxis` и `colors`).

Интерактивные графические карты

`plotly` пакет позволяет использовать множество интерактивных сюжетов, включая карты.

Существует несколько способов создания карты в `plotly` . Либо `plot_ly()` данные карты самостоятельно (через `plot_ly()` или `ggplotly()`), используйте возможности «нативного» отображения `plot_geo()` через `plot_geo()` или `plot_mapbox()`) или даже комбинацию обоих. Примером предоставления карты может быть:

```
library(plotly)
map_data("county") %>%
    group_by(group) %>%
    plot_ly(x = ~long, y = ~lat) %>%
    add_polygons() %>%
    layout(
        xaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE),
        yaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE)
    )
```



Для комбинации обоих подходов, swap `plot_ly()` для `plot_geo()` или `plot_mapbox()` в приведенном выше примере. См. Сюжетную книгу для большего количества примеров.

Следующий пример - это «строго родной» подход, который использует атрибут layout.geo для установки эстетики и уровня масштабирования карты. Он также использует базу данных `world.cities` из `maps` чтобы фильтровать бразильские города и `world.cities` их поверх «родной» карты.

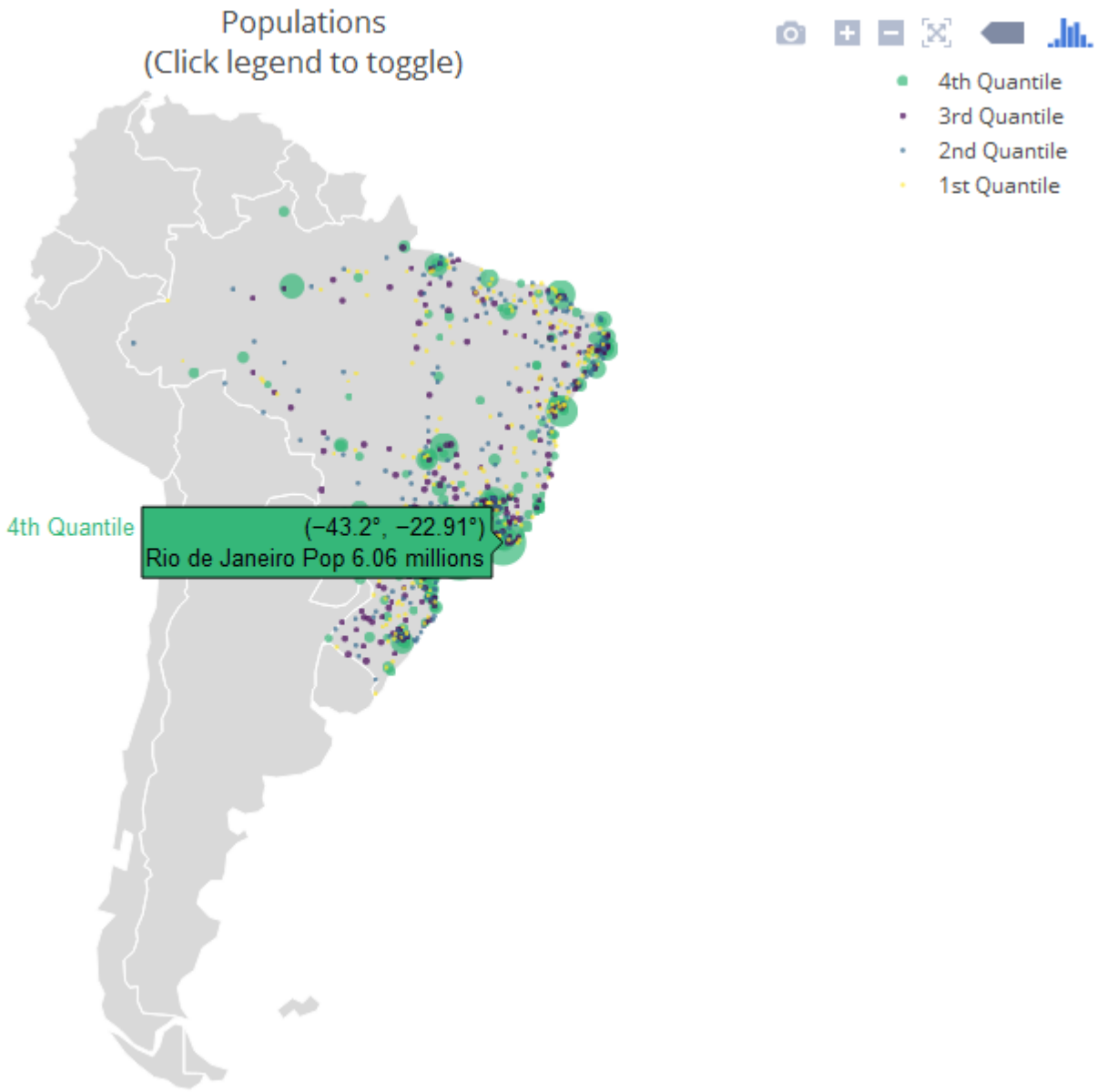
Основные переменные: `poph` - это текст с городом и его населением (который отображается

при наведении мыши); `q` - упорядоченный множитель из квантиля популяции. `ge` имеет информацию для компоновки карт. Дополнительную информацию см. В документации к пакету .

```
library(maps)
dfb <- world.cities[world.cities$country.etc=="Brazil",]
library(plotly)
dfb$poph <- paste(dfb$name, "Pop", round(dfb$pop/1e6,2), " millions")
dfb$q <- with(dfb, cut(pop, quantile(pop), include.lowest = T))
levels(dfb$q) <- paste(c("1st", "2nd", "3rd", "4th"), "Quantile")
dfb$q <- as.ordered(dfb$q)

ge <- list(
  scope = 'south america',
  showland = TRUE,
  landcolor = toRGB("gray85"),
  subunitwidth = 1,
  countrywidth = 1,
  subunitcolor = toRGB("white"),
  countrycolor = toRGB("white")
)

plot_geo(dfb, lon = ~long, lat = ~lat, text = ~poph,
    marker = ~list(size = sqrt(pop/10000) + 1, line = list(width = 0)),
    color = ~q, locationmode = 'country names') %>%
layout(geo = ge, title = 'Populations<br>(Click legend to toggle)')
```



Создание динамических HTML-карт с помощью брошюры

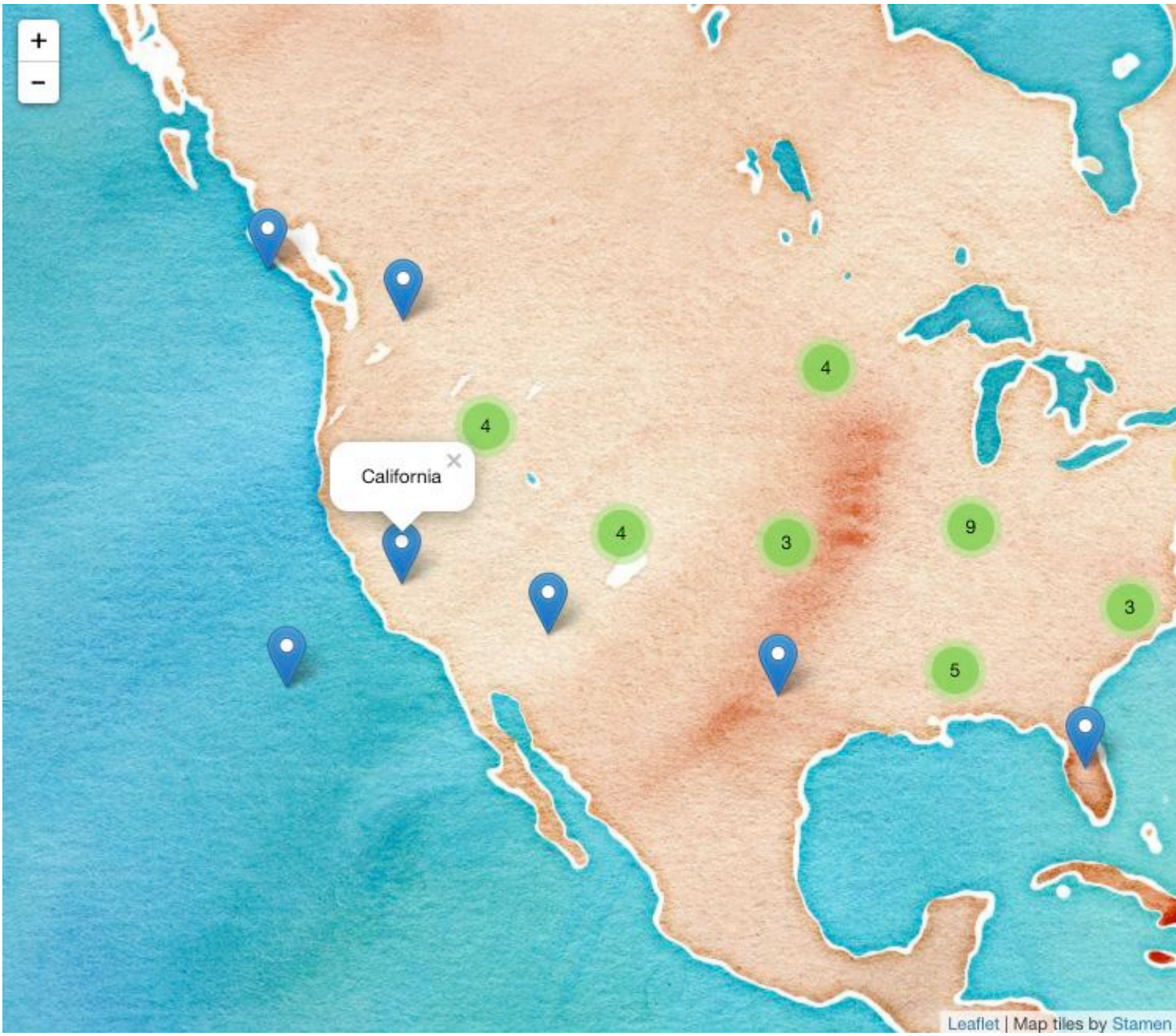
Листовка - это библиотека JavaScript с открытым исходным кодом для создания динамических карт для Интернета. RStudio написал R bindings для Leaflet, доступный через пакет `leaflet` , построенный с помощью `htmlwidgets` . Карты лифтов хорошо интегрируются с экосистемами RMarkdown и Shiny .

Интерфейс передается по каналам , используя функцию `leaflet()` чтобы инициализировать карту и последующие функции, добавляя (или удаляя) слои карты. Доступны многие типы слоев: от маркеров с всплывающими окнами до полигонов для создания карт choropleth. Переменные в data.frame, переданные в `leaflet()` , доступны через котировку function-style `~` .

Чтобы `state.name` `state.center` данных `state.name` и `state.center` :

```
library(leaflet)
```

```
data.frame(state.name, state.center) %>%
    leaflet() %>%
    addProviderTiles('Stamen.Watercolor') %>%
    addMarkers(lng = ~x, lat = ~y,
               popup = ~state.name,
               clusterOptions = markerClusterOptions())
```



(Снимок экрана, нажмите для динамической версии.)

Динамические карты листов в блестящих приложениях

Пакет Leaflet предназначен для интеграции с Shiny

В **ui** вы вызываете `leafletOutput()` а на сервере вы вызываете `renderLeaflet()`

```
library(shiny)
library(leaflet)
```

```
ui <- fluidPage(
    leafletOutput("my_leaf")
)

server <- function(input, output, session){

    output$my_leaf <- renderLeaflet({

        leaflet() %>%
            addProviderTiles('Hydda.Full') %>%
            setView(lat = -37.8, lng = 144.8, zoom = 10)

    })

}

shinyApp(ui, server)
```

Однако реактивные входы, которые влияют на выражение `renderLeaflet` , заставят всю карту перерисовываться каждый раз, когда реактивный элемент обновляется.

Поэтому, чтобы изменить уже запущенную карту, вы должны использовать `leafletProxy()` .

Обычно вы используете `leaflet` для создания статических аспектов карты, а `leafletProxy` для управления динамическими элементами, например:

```
library(shiny)
library(leaflet)

ui <- fluidPage(
    sliderInput(inputId = "slider",
                label = "values",
                min = 0,
                max = 100,
                value = 0,
                step = 1),
    leafletOutput("my_leaf")
)

server <- function(input, output, session){
    set.seed(123456)
    df <- data.frame(latitude = sample(seq(-38.5, -37.5, by = 0.01), 100),
                     longitude = sample(seq(144.0, 145.0, by = 0.01), 100),
                     value = seq(1,100))

    ## create static element
    output$my_leaf <- renderLeaflet({

        leaflet() %>%
            addProviderTiles('Hydda.Full') %>%
            setView(lat = -37.8, lng = 144.8, zoom = 8)

    })

    ## filter data
    df_filtered <- reactive({
        df[df$value >= input$slider, ]
```

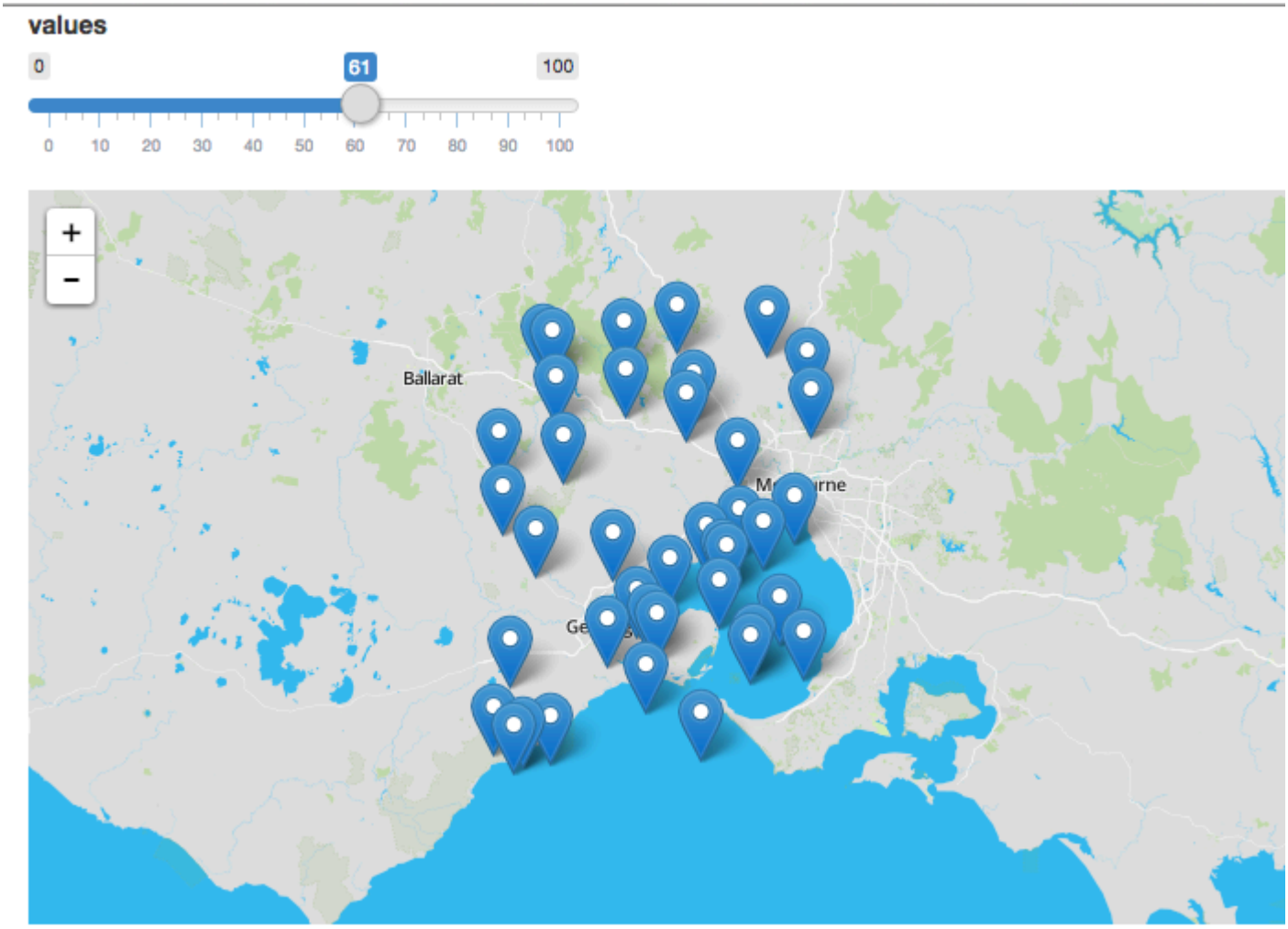
```
    })

    ## respond to the filtered data
    observe({

        leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
            clearMarkers() %>%   ## clear previous markers
            addMarkers()
    })

}

shinyApp(ui, server)
```



Прочитайте Введение в географические карты онлайн: https://riptutorial.com/ru/r/topic/1372/введение-в-географические-карты

глава 36: Веб-сканирование в R

Examples

Стандартный подход скремблирования с использованием пакета RCurl

Мы пытаемся извлечь фильмы и рейтинги лучших фильмов imdb

```
R> library(RCurl)
R> library(XML)
R> url <- "http://www.imdb.com/chart/top"
R> top <- getURL(url)
R> parsed_top <- htmlParse(top, encoding = "UTF-8")
R> top_table <- readHTMLTable(parsed_top)[[1]]
R> head(top_table[1:10, 1:3])


Rank & Title IMDb Rating
1 1. The Shawshank Redemption (1994) 9.2
2 2. The Godfather (1972) 9.2
3 3. The Godfather: Part II (1974) 9.0
4 4. The Dark Knight (2008) 8.9
5 5. Pulp Fiction (1994) 8.9
6 6. The Good, the Bad and the Ugly (1966) 8.9
7 7. Schindler's List (1993) 8.9
8 8. 12 Angry Men (1957) 8.9
9 9. The Lord of the Rings: The Return of the King (2003) 8.9
10 10. Fight Club (1999) 8.8
```

Прочитайте Веб-сканирование в R онлайн: https://riptutorial.com/ru/r/topic/4336/веб-сканирование-в-r

глава 37: Вероятностные распределения с R

Examples

PDF и PMF для разных распределений в R

PMF ДЛЯ БИНОМИЧЕСКОГО РАСПРОСТРАНЕНИЯ

Предположим, что справедливая смерть прокатается 10 раз. Какова вероятность бросать ровно два шестерки?

Вы можете ответить на вопрос, используя функцию dbinom:

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

PMF ДЛЯ РАСПРЕДЕЛЕНИЯ POISSON

Известно, что количество песка, заказанного в ресторане в определенный день, следует за распределением Пуассона со средним значением 20. Какова вероятность того, что ровно восемнадцать песков, которые будут заказаны завтра?

Вы можете ответить на вопрос с помощью функции dpois:

```
> dpois(18, 20)
[1] 0.08439355
```

PDF ДЛЯ НОРМАЛЬНОГО РАСПРОСТРАНЕНИЯ

Чтобы найти значение pdf при x = 2.5 для нормального распределения со средним значением 5 и стандартным отклонением 2, используйте команду:

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

Прочитайте Вероятностные распределения с R онлайн: https://riptutorial.com/ru/r/topic/4333/вероятностные-распределения-с-r

глава 38: Внедрить шаблон государственного устройства с использованием класса S4

Вступление

Конечные состояния. Концепции машин обычно реализуются в рамках языков ООП, например, с использованием языка Java, на основе шаблона State, определенного в GOF (относится к книге «Шаблоны проектирования»).

R предоставляет несколько механизмов для моделирования парадигмы OO, давайте применим S4 Object System для реализации этого шаблона.

Examples

Разбор строк с использованием машины состояния

Применим шаблон State Machine для разбора строк с определенным шаблоном с использованием функции класса S4 от R.

ПРОБЛЕМА

Нам нужно проанализировать файл, в котором каждая строка содержит информацию о человеке, используя разделитель (`";"`), но предоставленная информация является необязательной, и вместо предоставления пустого поля она отсутствует. В каждой строке мы можем получить следующую информацию: `Name;[Address;]Phone` . Если информация адреса не является обязательной, иногда мы ее имеем, а иногда нет, например:

```
GREGORY BROWN; 25 NE 25TH; +1-786-987-6543
DAVID SMITH;786-123-4567
ALAN PEREZ; 25 SE 50TH; +1-786-987-5553
```

Вторая строка не содержит адресной информации. Поэтому число разделителей может быть отклонено, как в этом случае с одним разделителем, а для других линий - двумя разделителями. Поскольку количество разделителей может варьироваться, одним из способов устранить эту проблему является распознавание присутствия или отсутствия заданного поля на основе его шаблона. В этом случае мы можем использовать регулярное выражение для идентификации таких паттернов. Например:

- **Имя** : `"^([AZ]'?\\s+)* *[AZ]+(\\s+[AZ]{1,2}\\.?,? +)*[AZ]+((-|\\s+)[AZ]+)*$"` . Например: `RAFAEL REAL, DAVID R. SMITH, ERNESTO PEREZ GONZALEZ, 0' CONNOR BROWN, LUIS PEREZ-MENA` и т.

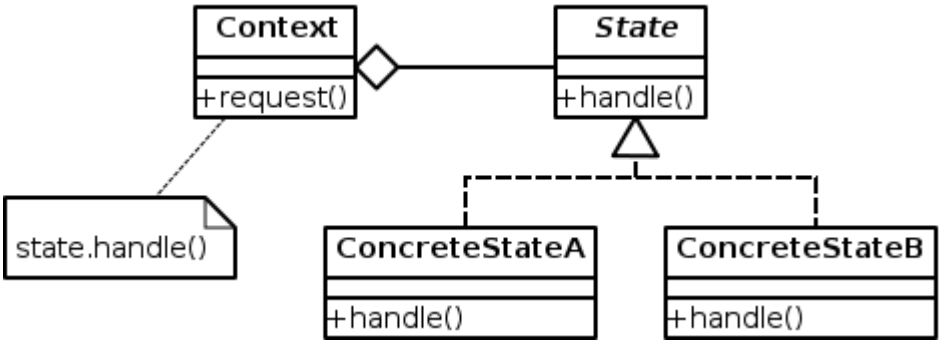
Д.

- **Адрес** : `"^\\s[0-9]{1,4}(\\s+[AZ]{1,2}[0-9]{1,2}[AZ]{1,2}|[AZ\\s0-9]+)$"` . Например: `11020 LE JEUNE ROAD , 87 SW 27TH` . Для простоты здесь мы не включаем zipcode, city, state, но я могу быть включен в это поле или добавлять дополнительные поля.
- **Телефон** : `"^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$"` . Например: `305-123-4567, 305 123 4567, +1-786-123-4567` .

Примечания :

- Я рассматриваю наиболее распространенную схему американских адресов и телефонов, ее можно легко расширить, чтобы рассмотреть более общие ситуации.
- В R знак `"\"` имеет особое значение для символьных переменных, поэтому нам нужно его избежать.
- Чтобы упростить процесс определения регулярных выражений, хорошей рекомендацией является использование следующей веб-страницы: regex101.com , поэтому вы можете играть с ней в данном примере, пока не получите ожидаемый результат для всех возможных комбинаций.

Идея состоит в том, чтобы идентифицировать каждое поле линии на основе ранее определенных шаблонов. Шаблон состояния определяет следующие сущности (классы), которые взаимодействуют для управления конкретным поведением (шаблон состояния - шаблон поведения):



Опишем каждый элемент, рассматривая контекст нашей проблемы:

- `Context` : хранит контекстную информацию процесса синтаксического анализа, то есть текущее состояние и обрабатывает весь процесс машинного процесса. Для каждого состояния выполняется действие (`handle()`), но контекст делегирует его на основе состояния на метод действия, определенный для определенного состояния (`handle()` из класса `State`). Он определяет интерфейс, представляющий интерес для клиентов. Наш класс `Context` можно определить следующим образом:
  - Атрибуты: `state`
  - Методы: `handle()` , ...
- `State` : абстрактный класс, представляющий состояние State Machine. Он определяет интерфейс для инкапсуляции поведения, связанного с конкретным состоянием контекста. Его можно определить следующим образом:

    - Атрибуты: `name, pattern`
    - Методы: `doAction()` , `isState` (с использованием атрибута `pattern` проверить, принадлежит ли входной аргумент этому шаблону состояния), ...
- `Concrete States` (государственные подклассы): Каждый подкласс класса `State` , реализующее поведение , связанное с состоянием `Context` . Наши подклассы: `InitState` , `NameState` , `AddressState` , `PhoneState` . Такие классы просто реализуют общий метод, используя определенную логику для таких состояний. Дополнительные атрибуты не требуются.

Примечание. Это вопрос предпочтения, как назвать метод, который выполняет действия, `handle()` , `doAction()` или `goNext()` . Имя метода `doAction()` может быть одинаковым для обоих классов (`State` или `Context`), которые мы предпочитали называть как `handle()` в классе `Context` чтобы избежать путаницы при определении двух общих методов с одинаковыми входными аргументами, но с другим классом.

ПЕРСОНАЛЬНЫЙ КЛАСС

Используя синтаксис S4, мы можем определить класс Person следующим образом:

```
setClass(Class = "Person",
    slots = c(name = "character", address = "character", phone = "character")
)
```

Это хорошая рекомендация для инициализации атрибутов класса. Документация `setClass` предлагает использовать общий метод, обозначенный как `"initialize"` , вместо использования устаревших атрибутов, таких как: `prototype, representation` .

```
setMethod("initialize", "Person",
  definition = function(.Object, name = NA_character_,
    address = NA_character_, phone = NA_character_) {
        .Object@name <- name
        .Object@address <- address
        .Object@phone <- phone
        .Object
    }
)
```

Поскольку метод initialize уже является стандартным универсальным методом пакетных `methods` , нам необходимо соблюдать определение исходного аргумента. Мы можем проверить его, набрав на подсказке R:

```
> initialize
```

Он возвращает полное определение функции, вы можете видеть вверху, кто определен как функция:

```
function (.Object, ...) {...}
```

Поэтому, когда мы используем `setMethod` мы должны следовать *exaccty* того же синтаксиса (`.Object`).

Другой существующий общий метод - это `show` , он эквивалентен методу `toString()` из Java, и неплохо иметь конкретную реализацию для домена класса:

```
setMethod("show", signature = "Person",
  definition = function(object) {
      info <- sprintf("%s@[name='%s', address='%s', phone='%s']",
        class(object), object@name, object@address, object@phone)
      cat(info)
      invisible(NULL)
  }
)
```

Примечание . Мы используем то же соглашение, что и в реализации Java `toString()` по умолчанию.

Предположим, мы хотим сохранить анализируемую информацию (список объектов `Person`) в наборе данных, тогда мы должны сначала преобразовать список объектов в нечто, что может преобразовать R (например, принудить объект как список). Мы можем определить следующий дополнительный метод (более подробно об этом см. Сообщение)

```
setGeneric(name = "as.list", signature = c('x'),
    def = function(x) standardGeneric("as.list"))

# Suggestion taken from here:
# http://stackoverflow.com/questions/30386009/how-to-extend-as-list-in-a-canonical-way-to-s4-
objects
setMethod("as.list", signature = "Person",
    definition = function(x) {
        mapply(function(y) {
        #apply as.list if the slot is again an user-defined object
        #therefore, as.list gets applied recursively
        if (inherits(slot(x,y),"Person")) {
          as.list(slot(x,y))
        } else {
          #otherwise just return the slot
          slot(x,y)
        }
      },
        slotNames(class(x)),
        SIMPLIFY=FALSE)
    }
)
```

R не обеспечивает синтаксис сахара для OO, потому что язык изначально был задуман, чтобы обеспечить ценные функции для статистиков. Поэтому для каждого пользовательского метода требуются две части: 1) часть определения (через `setGeneric`) и 2) часть реализации (через `setMethod`). Как в приведенном выше примере.

СОСТОЯНИЕ

Следуя синтаксису S4, давайте определим абстрактный класс `State` .

```
setClass(Class = "State", slots = c(name = "character", pattern = "character"))

setMethod("initialize", "State",
  definition = function(.Object, name = NA_character_, pattern = NA_character_) {
      .Object@name <- name
      .Object@pattern <- pattern
      .Object
  }
)

setMethod("show", signature = "State",
  definition = function(object) {
      info <- sprintf("%s@[name='%s', pattern='%s']", class(object),
          object@name, object@pattern)
      cat(info)
      invisible(NULL)
  }
)

setGeneric(name = "isState", signature = c('obj', 'input'),
    def = function(obj, input) standardGeneric("isState"))

setGeneric(name = "doAction", signature = c('obj', 'input', 'context'),
    def = function(obj, input, context) standardGeneric("doAction"))
```

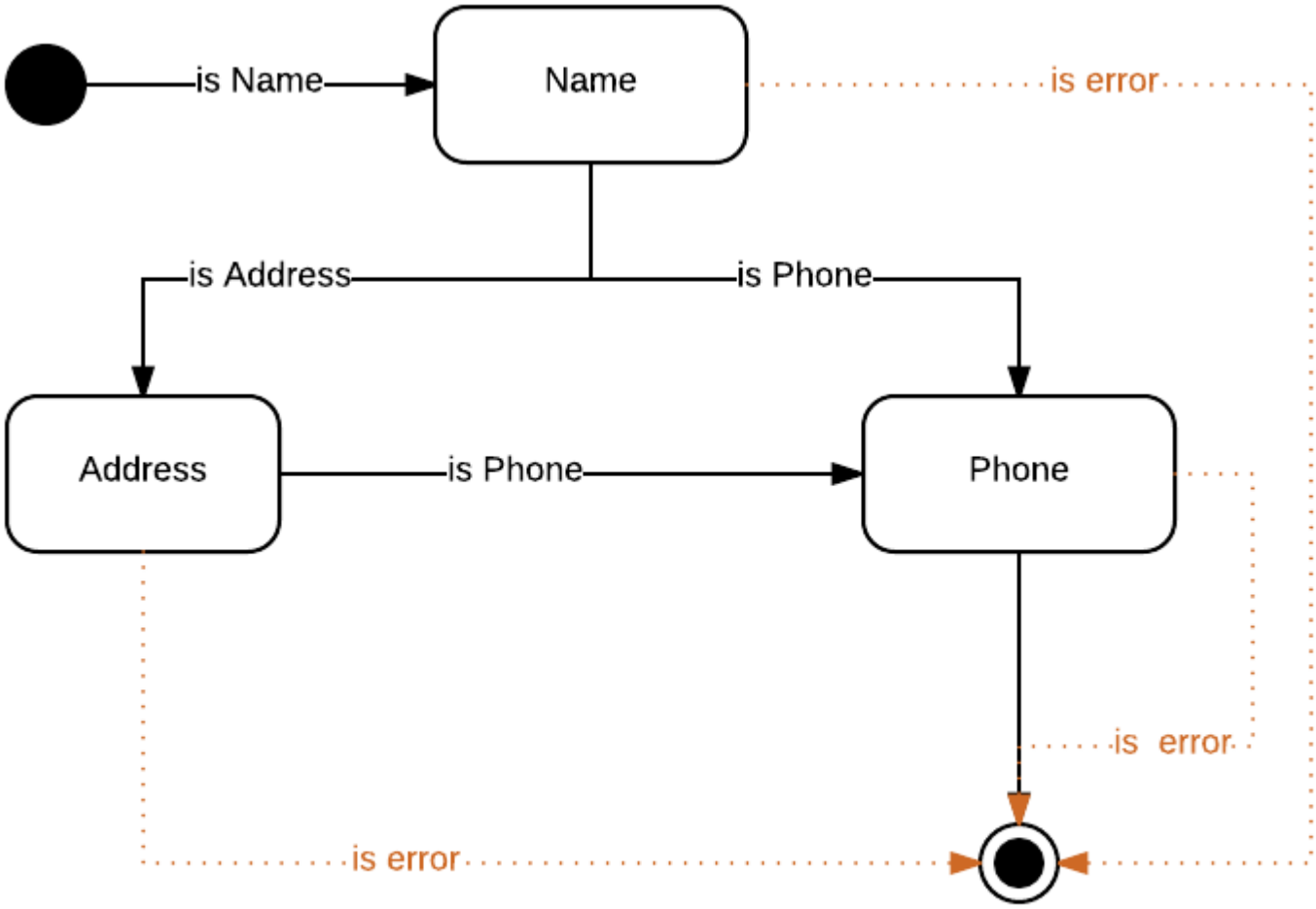
Каждый подкласс из `State` будет ассоциирован с `name` и `pattern` , но также может определить, принадлежит ли данный вход этому состоянию или нет (`isState()`), а также реализовать соответствующие действия для этого состояния (`doAction()` метод).

Чтобы понять процесс, давайте определим матрицу перехода для каждого состояния на основе полученного ввода:

Входное / текущее состояние	В этом	название	Адрес	Телефон
название	название			
Адрес		Адрес		
Телефон		Телефон	Телефон	
Конец				Конец

Примечание: ячейка `[row, col]=[i,j]` представляет состояние назначения для текущего состояния `j` , когда оно принимает вход `i` .

Это означает, что под именем состояния он может принимать два входа: адрес или номер телефона. Другой способ представления таблицы транзакций - использовать следующую диаграмму состояний машины UML :



is error: when the input argument has an invalid pattern

Давайте реализуем каждое конкретное государство как к югу от состояния класса `State`

ГОСУДАРСТВЕННЫЕ СУБ-КЛАССЫ

Init State :

Начальное состояние будет реализовано с помощью следующего класса:

```
setClass("InitState", contains = "State")

setMethod("initialize", "InitState",
  definition = function(.Object, name = "init", pattern = NA_character_) {
      .Object@name <- name
      .Object@pattern <- pattern
      .Object
  }
)

setMethod("show", signature = "InitState",
  definition = function(object) {
      callNextMethod()
  }
```

```
)
```

В R, чтобы указать, что класс является подклассом другого класса, использует атрибут `contains` и указывает имя класса родительского класса.

Поскольку подклассы просто реализуют общие методы, не добавляя дополнительных атрибутов, тогда метод `show` просто вызовет эквивалентный метод из верхнего класса (с помощью метода: `callNextMethod()`)

Начальное состояние не связано с шаблоном, оно просто представляет собой начало процесса, затем мы инициализируем класс значением `NA` .

Теперь позволяет реализовать общие методы из класса `State` :

```
setMethod(f = "isState", signature = "InitState",
  definition = function(obj, input) {
      nameState <- new("NameState")
      result <- isState(nameState, input)
      return(result)
  }
)
```

Для этого конкретного состояния (без `pattern`) идея, что он просто инициализирует процесс синтаксического анализа, ожидая первого поля, будет `name` , иначе это будет ошибка.

```
setMethod(f = "doAction", signature = "InitState",
    definition = function(obj, input, context) {
        nameState <- new("NameState")
        if (isState(nameState, input)) {
            person <- context@person
            person@name <- trimws(input)
            context@person <- person
            context@state <- nameState
        } else {
            msg <- sprintf("The input argument: '%s' cannot be identified", input)
            stop(msg)
        }
        return(context)
    }
)
```

Метод `doAction` обеспечивает переход и обновляет контекст с извлеченной информацией. Здесь мы обращаемся к контекстной информации через `@-operator` . Вместо этого мы можем определить методы `get/set` , чтобы инкапсулировать этот процесс (как это предусмотрено в лучших практиках ОО: инкапсуляция), но это добавит еще четыре метода для каждого `get-set` без добавления значения для этого примера.

Это хорошая рекомендация во всей реализации `doAction` , чтобы добавить защиту, если входной аргумент не определен правильно.

Имя государства

Вот определение этого определения класса:

```
setClass ("NameState", contains = "State")

setMethod("initialize","NameState",
  definition=function(.Object, name="name",
        pattern = "^([A-Z]'?\\s+)* *[A-Z]+(\\s+[A-Z]{1,2}\\.?,? +)*[A-Z]+((-|\\s+)[A-Z]+)*$")
{
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
  }
)

setMethod("show", signature = "NameState",
  definition = function(object) {
      callNextMethod()
  }
)
```

Мы используем функцию `grepl` для проверки того, что вход принадлежит данному шаблону.

```
setMethod(f="isState", signature="NameState",
  definition=function(obj, input) {
      result <- grepl(obj@pattern, input, perl=TRUE)
      return(result)
  }
)
```

Теперь мы определяем действие, выполняемое для данного состояния:

```
setMethod(f = "doAction", signature = "NameState",
  definition=function(obj, input, context) {
      addressState <- new("AddressState")
      phoneState <- new("PhoneState")
      person <- context@person
      if (isState(addressState, input)) {
          person@address <- trimws(input)
          context@person <- person
          context@state <- addressState
      } else if (isState(phoneState, input)) {
          person@phone <- trimws(input)
          context@person <- person
          context@state <- phoneState
      } else {
          msg <- sprintf("The input argument: '%s' cannot be identified", input)
          stop(msg)
      }
      return(context)
  }
)
```

Здесь мы рассмотрим возможные переходы: один для состояния адреса, а другой для состояния телефона. Во всех случаях мы обновляем контекстную информацию:

- Информация о `person` : `address` или `phone` с аргументом ввода.
- `state` процесса

Способ идентификации состояния заключается в вызове метода: `isState()` для определенного состояния. Мы создаем стандартные состояния по умолчанию (`addressState, phoneState`), а затем запрашиваем конкретную проверку.

Логика для реализации других подклассов (по одному на состояние) очень похожа.

Состояние адреса

```
setClass("AddressState", contains = "State")

setMethod("initialize", "AddressState",
  definition = function(.Object, name="address",
    pattern = "^\\s[0-9]{1,4}(\\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2}|[A-Z\\s0-9]+)$") {
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
    }
)

setMethod("show", signature = "AddressState",
  definition = function(object) {
      callNextMethod()
  }
)

setMethod(f="isState", signature="AddressState",
    definition=function(obj, input) {
        result <- grepl(obj@pattern, input, perl=TRUE)
        return(result)
    }
)

setMethod(f = "doAction", "AddressState",
    definition=function(obj, input, context) {
        phoneState <- new("PhoneState")
        if (isState(phoneState, input)) {
            person <- context@person
            person@phone <- trimws(input)
            context@person <- person
            context@state <- phoneState
        } else {
            msg <- sprintf("The input argument: '%s' cannot be identified", input)
            stop(msg)
        }
        return(context)
    }
)
```

Состояние телефона

```
setClass("PhoneState", contains = "State")

setMethod("initialize", "PhoneState",
```

```
  definition = function(.Object, name = "phone",
    pattern = "^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$") {
        .Object@pattern <- pattern
        .Object@name <- name
        .Object
    }
)

setMethod("show", signature = "PhoneState",
  definition = function(object) {
      callNextMethod()
  }
)

setMethod(f = "isState", signature = "PhoneState",
    definition = function(obj, input) {
        result <- grepl(obj@pattern, input, perl = TRUE)
        return(result)
    }
)
```

Здесь мы добавляем информацию человека в список `persons context` .

```
setMethod(f = "doAction", "PhoneState",
    definition = function(obj, input, context) {
        context <- addPerson(context, context@person)
        context@state <- new("InitState")
        return(context)
    }
)
```

КОНТЕКСТНЫЙ КЛАСС

Теперь давайте объясним реализацию класса `Context` . Мы можем определить его, учитывая следующие атрибуты:

```
setClass(Class = "Context",
     slots = c(state = "State", persons = "list", person = "Person")
)
```

куда

- `state` : Текущее состояние процесса
- `person` : Текущее лицо, оно представляет информацию, которую мы уже проанализировали из текущей строки.
- `persons` : список обработанных обработанных лиц.

Примечание . При желании мы можем добавить `name` чтобы идентифицировать контекст по имени, если мы работаем с несколькими типами парсера.

```
setMethod(f="initialize", signature="Context",
  definition = function(.Object) {
        .Object@state <- new("InitState")
```

```
        .Object@persons <- list()
        .Object@person <- new("Person")
        return(.Object)
    }
)

setMethod("show", signature = "Context",
  definition = function(object) {
      cat("An object of class ", class(object), "\n", sep = "")
      info <- sprintf("[state='%s', persons='%s', person='%s']", object@state,
          toString(object@persons), object@person)
      cat(info)
      invisible(NULL)
  }
)

setGeneric(name = "handle", signature = c('obj', 'input', 'context'),
    def = function(obj, input, context) standardGeneric("handle"))

setGeneric(name = "addPerson", signature = c('obj', 'person'),
    def = function(obj, person) standardGeneric("addPerson"))

setGeneric(name = "parseLine", signature = c('obj', 's'),
    def = function(obj, s) standardGeneric("parseLine"))

setGeneric(name = "parseLines", signature = c('obj', 's'),
    def = function(obj, s) standardGeneric("parseLines"))

setGeneric(name = "as.df", signature = c('obj'),
    def = function(obj) standardGeneric("as.df"))
```

Используя такие общие методы, мы контролируем все поведение процесса синтаксического анализа:

- `handle()` : вызывается конкретный `doAction()` текущего `state` .
- `addPerson` : Как только мы достигнем конечного состояния, нам нужно добавить `person` в список `persons` мы проанализировали.
- `parseLine()` : проанализировать одну строку
- `parseLines()` : Разбор нескольких строк (массив строк)
- `as.df()` : Извлечь информацию из списка `persons` в объект фрейма данных.

Перейдем теперь к соответствующим реализациям:

`handle()` , делегаты метода `doAction()` из текущего `state context` :

```
setMethod(f = "handle", signature = "Context",
    definition = function(obj, input) {
        obj <- doAction(obj@state, input, obj)
        return(obj)
    }
)

setMethod(f = "addPerson", signature = "Context",
  definition = function(obj, person) {
      obj@persons <- c(obj@persons, person)
      return(obj)
```

```
  }
)
```

Сначала мы разделим исходную строку в массиве с помощью разделителя, чтобы идентифицировать каждый элемент через R-функцию `strsplit()` , а затем итерации для каждого элемента в качестве входного значения для данного состояния. `handle()` метод снова возвращает `context` с обновленной информацией (`state` , `person` , `persons` , атрибут).

```
setMethod(f = "parseLine", signature = "Context",
  definition = function(obj, s) {
      elements <- strsplit(s, ";")[[1]]
      # Adding an empty field for considering the end state.
      elements <- c(elements, "")
      n <- length(elements)
      input <- NULL
      for (i in (1:n)) {
        input <- elements[i]
        obj <- handle(obj, input)
      }
      return(obj@person)
  }
)
```

Becuase R делает копию входного аргумента, нам нужно вернуть контекст (`obj`):

```
setMethod(f = "parseLines", signature = "Context",
  definition = function(obj, s) {
      n <- length(s)
      listOfPersons <- list()
      for (i in (1:n)) {
          ipersons <- parseLine(obj, s[i])
          listOfPersons[[i]] <- ipersons
      }
      obj@persons <- listOfPersons
      return(obj)
  }
)
```

`persons` атрибута - это список экземпляров класса S4 `Person` . Это не может быть принуждено к любому стандарту, потому что R не знает, как обрабатывать экземпляр класса, определенного пользователем. Решение состоит в том, чтобы преобразовать `Person` в список, используя ранее описанный метод `as.list` . Тогда мы можем применить эту функцию к каждому элементу списка `persons` , через `lapply()` функцию. Затем в следующем вызове функции `lappy()` теперь применяется функция `data.frame` для преобразования каждого элемента `persons.list` в фрейм данных. Наконец, `rbind()` вызывается для добавления каждого элемента, преобразованного в новую строку генерируемого кадра данных (более подробно об этом см. В этом сообщении)

```
# Sugestion taken from this post:
# http://stackoverflow.com/questions/4227223/r-list-to-data-frame
setMethod(f = "as.df", signature = "Context",
  definition = function(obj) {
```

```
    persons <- obj@persons
    persons.list <- lapply(persons, as.list)
    persons.ds <- do.call(rbind, lapply(persons.list, data.frame, stringsAsFactors = FALSE))
    return(persons.ds)
  }
)
```

ВСТУПЛЕНИЕ ВМЕСТЕ

Наконец, позволяет протестировать все решение. Определите строки для разбора, где для второй строки отсутствует адресная информация.

```
s <- c(
    "GREGORY BROWN; 25 NE 25TH; +1-786-987-6543",
    "DAVID SMITH;786-123-4567",
     "ALAN PEREZ; 25 SE 50TH; +1-786-987-5553"
)
```

Теперь мы инициализируем `context` и анализируем строки:

```
context <- new("Context")
context <- parseLines(context, s)
```

Наконец, получите соответствующий набор данных и напечатайте его:

```
df <- as.df(context)
> df
           name    address           phone
1 GREGORY BROWN 25 NE 25TH +1-786-987-6543
2   DAVID SMITH       <NA>    786-123-4567
3    ALAN PEREZ 25 SE 50TH +1-786-987-5553
```

Давайте проверим теперь методы `show` :

```
> show(context@persons[[1]])
Person@[name='GREGORY BROWN', address='25 NE 25TH', phone='+1-786-987-6543']
```

И для некоторого суб-состояния:

```
>show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\s*(\+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$']
```

Наконец, проверьте метод `as.list()` :

```
> as.list(context@persons[[1]])
$name
[1] "GREGORY BROWN"

$address
[1] "25 NE 25TH"

$phone
```

```
[1] "+1-786-987-6543"

>
```

ЗАКЛЮЧЕНИЕ

В этом примере показано, как реализовать шаблон состояния, используя один из доступных механизмов из R для использования парадигмы OO. Тем не менее, решение R OO не является удобным для пользователя и сильно отличается от других языков ООП. Вам нужно переключить свое мышление, потому что синтаксис совершенно другой, он напоминает больше парадигму функционального программирования. Например, вместо: `object.setID("A1")` как в Java / C #, для R вы должны вызвать метод таким образом: `setID(object, "A1")` . Поэтому вам всегда нужно включить объект в качестве входного аргумента, чтобы обеспечить контекст функции. На том же пути, нет никакого специального `this` класс атрибута и либо `"."` нотация для доступа к методам или атрибутам данного класса. Это скорее запрос ошибки, потому что для ссылки на класс или методы выполняется через значение атрибута (`"Person"` , `"isState"` и т. Д.).

Сказанное выше, решение класса S4, требует гораздо больше строк кода, чем традиционные языки Java / C # для выполнения простых задач. Во всяком случае, государственный шаблон является хорошим и универсальным решением для таких проблем. Это упрощает процесс делегирования логики в конкретное состояние. Вместо того, чтобы иметь большой блок `if-else` для управления всеми ситуациями, у нас есть меньшие блоки `if-else` внутри каждой реализации подкласса `State` для реализации действия для выполнения в каждом состоянии.

Приложение : здесь вы можете скачать весь скрипт.

Любое предложение приветствуется.

Прочитайте Внедрить шаблон государственного устройства с использованием класса S4 онлайн: https://riptutorial.com/ru/r/topic/9126/внедрить-шаблон-государственного-устройства-с-использованием-класса-s4

глава 39: Воспроизводимые R

Вступление

С «Воспроизводимостью» мы подразумеваем, что кто-то другой (возможно, вы в будущем) может повторить шаги, которые вы выполнили, и получить тот же результат. См. Результирующее исследование задачи .

замечания

Чтобы создать воспроизводимые результаты, необходимо устранить все источники изменений. Например, если используется генератор случайных чисел (псевдо), семя должно быть исправлено, если вы хотите воссоздать те же результаты. Другой способ уменьшить вариацию - объединить текст и вычисления в том же документе.

Рекомендации

- Пэн, РД (2011). Воспроизводимые исследования в области вычислительной техники. Science, 334 (6060), 1226-1227. http://doi.org/10.1126/science.1213847
- Пэн, Роджер Д. Отчет о написании научных данных в R. Leanpub, 2015 г. https://leanpub.com/reportwriting .

Examples

Воспроизводимость данных

`dput()` и `dget()`

Самый простой способ совместного использования (предпочтительного малого) кадра данных - использовать базовую функцию `dput()` . Он будет экспортировать объект R в текстовой форме.

Примечание. Прежде чем приводить приведенные ниже примеры данных, убедитесь, что вы находитесь в пустой папке, в которую вы можете писать. Запустите `getwd()` и прочитайте `?setwd` если вам нужно сменить папки.

```
dput(mtcars, file = 'df.txt')
```

Затем каждый может загрузить точный объект R в свою GlobalEnvironment с помощью

функции `dget()` .

```
df <- dget('df.txt')
```

Для больших объектов R существует несколько способов сохранения их воспроизводимости. См. Раздел « Ввод и вывод» .

Воспроизводимость пакета

Воспроизводимость пакета - очень распространенная проблема при воспроизведении некоторого R-кода. Когда различные пакеты обновляются, некоторые межсоединения между ними могут сломаться. Идеальное решение проблемы состоит в том, чтобы воспроизвести образ машины для записи кода R на вашем компьютере в день написания кода. И вот идет пакет `checkpoint` .

Начиная с 2014-09-17, авторы пакета делают ежедневные копии всего репозитория пакетов CRAN собственному зеркальному репозиторию - Microsoft R Archived Network. Поэтому, чтобы избежать проблем с воспроизведением пакетов при создании воспроизводимого проекта R, все, что вам нужно, это:

1. Убедитесь, что все ваши пакеты (и версия R) обновлены.
2. Включите в свой код строку `checkpoint::checkpoint('YYYY-MM-DD')` .

`checkpoint` создаст каталог `.checkpoint` в вашем каталоге R_home (`"~/"`). В этот технический каталог будут установлены все пакеты, которые используются в вашем проекте. Это означает, что `checkpoint` просматривает все файлы `.R` в каталоге проекта, чтобы забрать все вызовы `library()` или `require()` и установить все необходимые пакеты в том виде, в котором они существовали в CRAN в указанную дату.

PRO Вы освобождаетесь от проблемы воспроизводимости пакета.
CONTRA Для каждой указанной даты вам необходимо загрузить и установить все пакеты, которые используются в определенном проекте, который вы хотите воспроизвести. Это может занять довольно много времени.

Прочитайте Воспроизводимые R онлайн: https://riptutorial.com/ru/r/topic/4087/воспроизводимые-r

глава 40: Временные ряды и прогнозирование

замечания

Прогнозирование и анализ временных рядов можно обрабатывать с помощью обычных функций из пакета `stats` , например `glm()` или большого количества специализированных пакетов. В представлении задачи CRAN для анализа временных рядов содержится подробный список ключевых пакетов по темам с краткими описаниями.

Examples

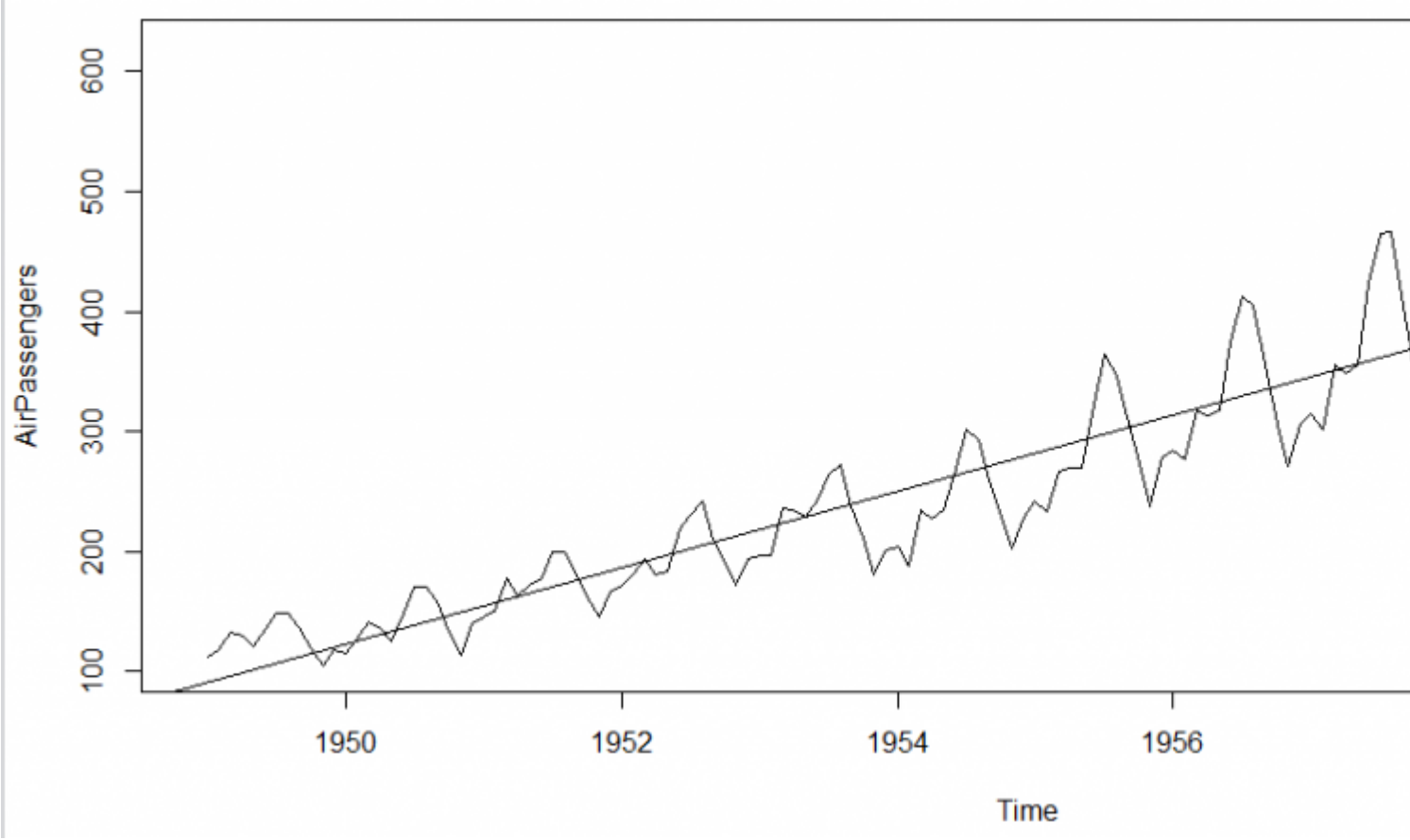
Анализ исследовательских данных с данными временных рядов

```
data(AirPassengers)
class(AirPassengers)
```

1 "ts"

В духе Explorative Data Analysis (EDA) хорошим первым шагом является просмотр графика ваших данных временного ряда:

```
plot(AirPassengers) # plot the raw data
abline(reg=lm(AirPassengers~time(AirPassengers))) # fit a trend line
```

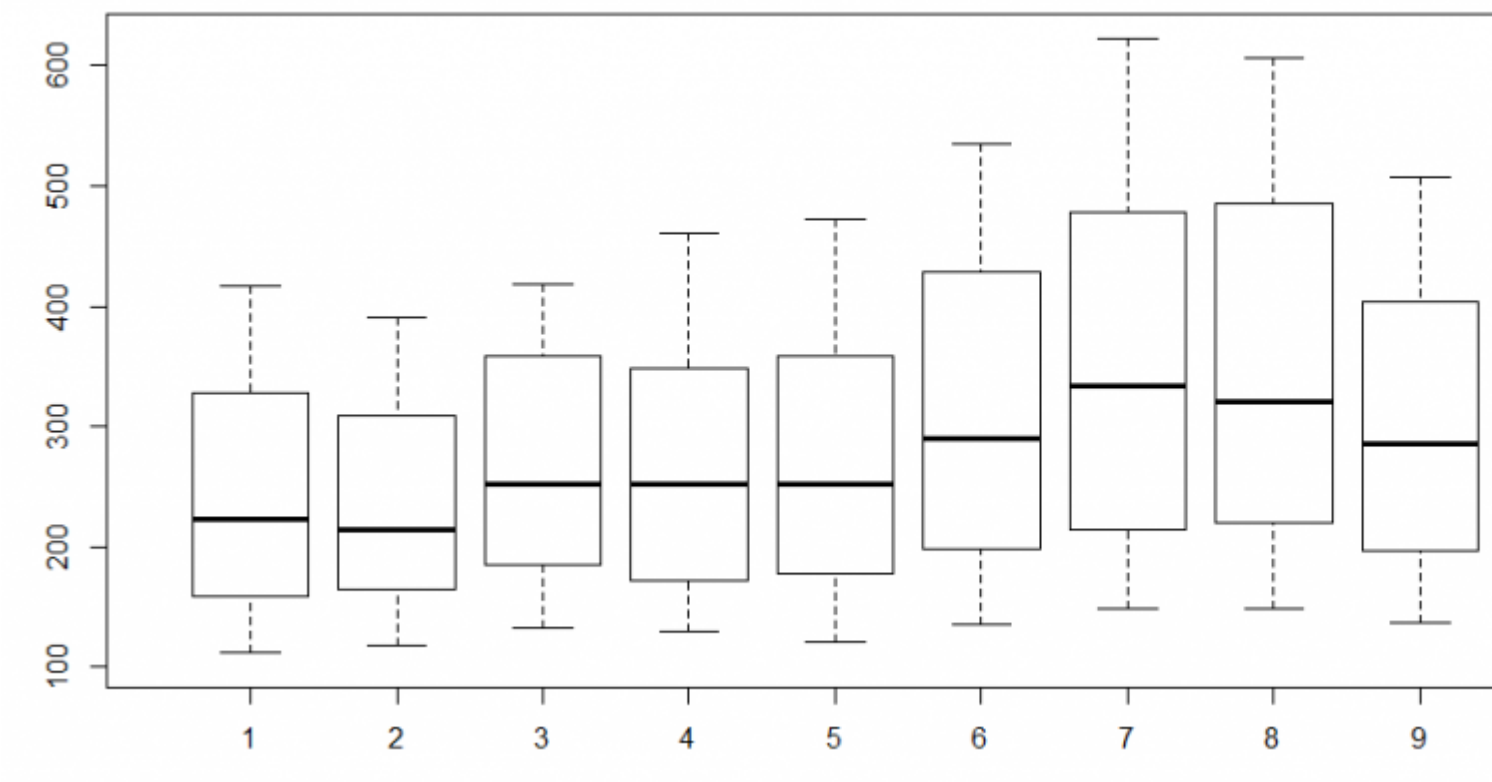


Для дальнейшего EDA мы рассматриваем циклы через годы:

```
cycle(AirPassengers)
```

```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949   1   2   3   4   5   6   7   8   9  10  11  12
1950   1   2   3   4   5   6   7   8   9  10  11  12
1951   1   2   3   4   5   6   7   8   9  10  11  12
1952   1   2   3   4   5   6   7   8   9  10  11  12
1953   1   2   3   4   5   6   7   8   9  10  11  12
1954   1   2   3   4   5   6   7   8   9  10  11  12
1955   1   2   3   4   5   6   7   8   9  10  11  12
1956   1   2   3   4   5   6   7   8   9  10  11  12
1957   1   2   3   4   5   6   7   8   9  10  11  12
1958   1   2   3   4   5   6   7   8   9  10  11  12
1959   1   2   3   4   5   6   7   8   9  10  11  12
1960   1   2   3   4   5   6   7   8   9  10  11  12
```

```
boxplot(AirPassengers~cycle(AirPassengers)) #Box plot across months to explore seasonal
effects
```



Создание объекта ts

Данные временных рядов могут храниться как объект `ts` . `ts` содержат информацию о сезонной частоте, которая используется функциями ARIMA. Он также позволяет вызывать элементы в серии по дате, используя команду `window` .

```
#Create a dummy dataset of 100 observations
x <- rnorm(100)

#Convert this vector to a ts object with 100 annual observations
x <- ts(x, start = c(1900), freq = 1)

#Convert this vector to a ts object with 100 monthly observations starting in July
x <- ts(x, start = c(1900, 7), freq = 12)

    #Alternatively, the starting observation can be a number:
    x <- ts(x, start = 1900.5, freq = 12)

#Convert this vector to a ts object with 100 daily observations and weekly frequency starting
in the first week of 1900
x <- ts(x, start = c(1900, 1), freq = 7)

#The default plot for a ts object is a line plot
plot(x)

#The window function can call elements or sets of elements by date

    #Call the first 4 weeks of 1900
    window(x, start = c(1900, 1), end = (1900, 4))

    #Call only the 10th week in 1900
    window(x, start = c(1900, 10), end = (1900, 10))
```

```
    #Call all weeks including and after the 10th week of 1900
    window(x, start = c(1900, 10))
```

Можно создавать объекты `ts` с несколькими сериями:

```
#Create a dummy matrix of 3 series with 100 observations each
x <- cbind(rnorm(100), rnorm(100), rnorm(100))

#Create a multi-series ts with annual observation starting in 1900
x <- ts(x, start = 1900, freq = 1)

#R will draw a plot for each series in the object
plot(x)
```

Прочитайте Временные ряды и прогнозирование онлайн: https://riptutorial.com/ru/r/topic/2701/временные-ряды-и-прогнозирование

глава 41: Вход и выход

замечания

Чтобы создать пути к файлам, для чтения или записи используйте файл `file.path` .

Используйте `dir` чтобы посмотреть, какие файлы находятся в каталоге.

Examples

Чтение и запись кадров данных

Кадры данных представляют собой структуру табличных данных R. Они могут быть написаны или прочитаны различными способами.

Этот пример иллюстрирует пару общих ситуаций. См. Ссылки в конце для других ресурсов.

Пишу

Прежде чем приводить приведенные ниже примеры данных, убедитесь, что вы находитесь в папке, в которую хотите записать. Запустите `getwd()` чтобы проверить папку, в которой вы находитесь, и читать `?setwd` если вам нужно сменить папки.

```
set.seed(1)
for (i in 1:3)
  write.table(
    data.frame(id = 1:2, v = sample(letters, 2)),
    file = sprintf("file201%s.csv", i)
  )
```

Теперь у нас есть три файла CSV с одинаковым форматированием на диске.

чтение

У нас есть три файла с одинаковым форматированием (из последнего раздела) для чтения. Поскольку эти файлы связаны между собой, мы должны хранить их вместе после чтения в `list` :

```
file_names = c("file2011.csv", "file2012.csv", "file2013.csv")
file_contents = lapply(setNames(file_names, file_names), read.table)

# $file2011.csv
#   id v
```

```
# 1  1 g
# 2  2 j
#
# $file2012.csv
#   id v
# 1  1 o
# 2  2 w
#
# $file2013.csv
#   id v
# 1  1 f
# 2  2 w
```

Чтобы работать с этим списком файлов, сначала проверьте структуру с `str(file_contents)` , затем прочитайте о том, как `?rbind` список с помощью `?rbind` или итерировать по списку с помощью `?lapply` .

Дополнительные ресурсы

Проверьте `?read.table` и `?write.table` чтобы расширить этот пример. Также:

- R двоичных форматов (для таблиц и других объектов)
- Форматы таблиц с обычным текстом
    - CSV с разделителями-запятыми
    - TSV с разделителями табуляции
    - Форматы фиксированной ширины
- Языковые форматы бинарных таблиц
    - Пух Перо
- Формат заграничных таблиц и таблиц
    - ПАВ
    - SPSS
    - Stata
    - превосходить
- Форматы таблиц реляционных баз данных
    - MySQL
    - SQLite
    - PostgreSQL

Прочитайте Вход и выход онлайн: https://riptutorial.com/ru/r/topic/5543/вход-и-выход

глава 42: Вход пользователя

Синтаксис

- variable <- readline (prompt = "Любое сообщение для пользователя")
- name <- readline (prompt = "Как ваше имя")

Examples

Вход пользователя в R

Иногда может быть интересно иметь перекрестный разговор между пользователем и программой, одним из примеров которого является пакет завихрений , который был разработан для обучения R в R.

Можно запросить ввод пользователя с помощью команды `readline` :

```
name <- readline(prompt = "What is your name?")
```

Затем пользователь может дать любой ответ, такой как число, символ, векторы и результаты сканирования, чтобы убедиться, что пользователь дал правильный ответ. Например:

```
result <- readline(prompt = "What is the result of 1+1?")
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Однако следует отметить, что этот код застревает в бесконечном цикле, поскольку пользовательский ввод сохраняется как символ.

Мы должны принудить его к числу, используя `as.numeric` :

```
result <- as.numeric(readline(prompt = "What is the result of 1+1?"))
while(result!=2){
    readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

Прочитайте Вход пользователя онлайн: https://riptutorial.com/ru/r/topic/5098/вход-пользователя

глава 43: Входы / выходы для растровых изображений

Вступление

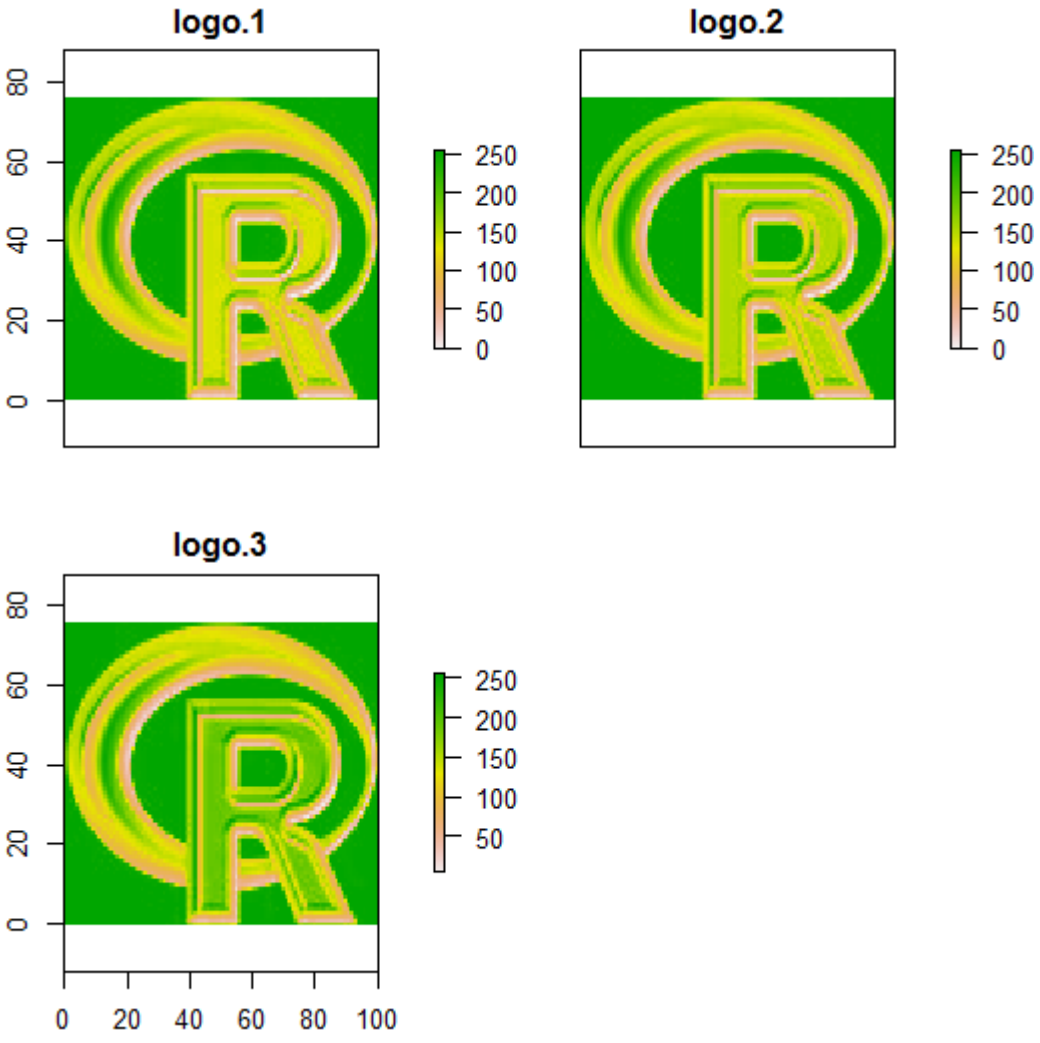
См. Также Анализ растра и изображений, а также ввод и вывод

Examples

Загрузка многослойного растра

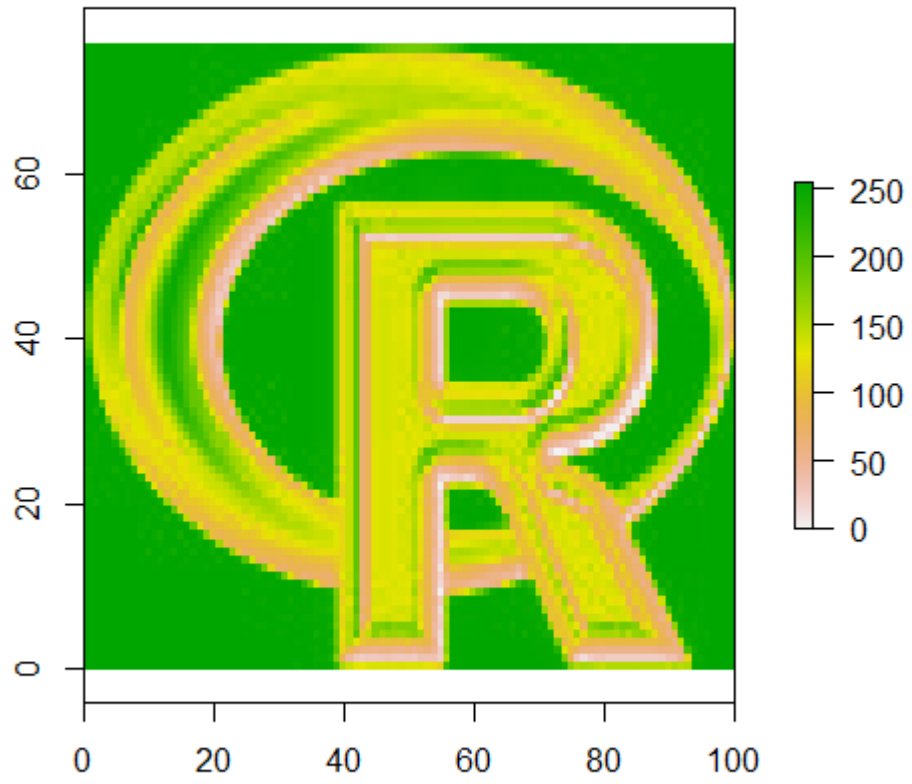
R-Logo - это многослойный растровый файл (красный, зеленый, синий)

```
library(raster)
r <- stack("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Отдельные слои объекта `RasterStack` могут быть адресованы `[[` .

```
plot(r[[1]])
```



Прочитайте Входы / выходы для растровых изображений онлайн:
https://riptutorial.com/ru/r/topic/5539/входы---выходы-для-растровых-изображений

глава 44: Выбор функции в R - Удаление внешних функций

Examples

Удаление функций с нулевой или почти нулевой дисперсией

Хорошим кандидатом для удаления является функция, имеющая почти нулевую дисперсию.

Вы можете вручную определить числовую дисперсию ниже своего собственного порога:

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

Или вы можете использовать пакет каретки, чтобы найти почти нулевую дисперсию. Преимущество здесь заключается в том, что определяется почти нулевая дисперсия не в численном расчете дисперсии, а скорее как функция редкости:

> «nearZeroVar диагностирует предиктора, которые имеют одно уникальное значение (т. е. являются предикторами с нулевой дисперсией) или предикторами, которые имеют обе следующие характеристики: у них очень мало уникальных значений по сравнению с количеством выборок и отношением частоты наиболее распространенного значения к частоте второго наиболее распространенного значения большое ... "

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

Удаление функций с большим количеством NA

Если функция в основном не содержит данных, она является хорошим кандидатом для удаления:

```
library(VIM)
data(sleep)
colMeans(is.na(sleep))

   BodyWgt    BrainWgt       NonD      Dream      Sleep       Span       Gest
0.00000000 0.00000000 0.22580645 0.19354839 0.06451613 0.06451613 0.06451613
      Pred        Exp     Danger
0.00000000 0.00000000 0.00000000
```

В этом случае мы можем захотеть удалить NonD и Dream, каждый из которых имеет около

20% отсутствующих значений (ваше обрезание может отличаться)

Удаление тесно связанных функций

Близко коррелированные функции могут добавить отклонения в вашу модель, и удаление одной из коррелированных пар может помочь уменьшить это. Существует множество способов обнаружения корреляции. Вот один из них:

```
library(purrr) # in order to use keep()

# select correlatable vars
toCorrelate<-mtcars %>% keep(is.numeric)

# calculate correlation matrix
correlationMatrix <- cor(toCorrelate)

# pick only one out of each highly correlated pair's mirror image
correlationMatrix[upper.tri(correlationMatrix)]<-0

# and I don't remove the highly-correlated-with-itself group
diag(correlationMatrix)<-0

# find features that are highly correlated with another feature at the +- 0.85 level
apply(correlationMatrix,2, function(x) any(abs(x)>=0.85))

  mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
 TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Я хочу посмотреть на то, что MPG коррелирует так сильно, и решить, что сохранить и что бросить. То же самое для цил и disp. В качестве альтернативы мне может потребоваться объединить некоторые сильно коррелированные функции.

Прочитайте Выбор функции в R - Удаление внешних функций онлайн:
https://riptutorial.com/ru/r/topic/7561/выбор-функции-в-r---удаление-внешних-функций

глава 45: Выполнение теста перестановки

Examples

Достаточно общая функция

Мы будем использовать встроенный набор данных о росте зуба . Нас интересует, существует ли статистически значимая разница в росте зубов, когда морским свинкам дают витамин C против апельсинового сока.

Вот полный пример:

```
teethVC = ToothGrowth[ToothGrowth$supp == 'VC',]
teethOJ = ToothGrowth[ToothGrowth$supp == 'OJ',]

permutationTest = function(vectorA, vectorB, testStat){
  N = 10^5
  fullSet = c(vectorA, vectorB)
  lengthA = length(vectorA)
  lengthB = length(vectorB)
  trials <- replicate(N,
                      {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)
                      testStat((fullSet[index]), fullSet[-index])  } )
  trials
}
vec1 =teethVC$len;
vec2 =teethOJ$len;
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
result = permutationTest(vec1, vec2, subtractMeans)
observedMeanDifference = subtractMeans(vec1, vec2)
result = c(result, observedMeanDifference)
hist(result)
abline(v=observedMeanDifference, col = "blue")
pValue = 2*mean(result <= (observedMeanDifference))
pValue
```

После того как мы прочитаем в CSV, определим функцию

```
permutationTest = function(vectorA, vectorB, testStat){
  N = 10^5
  fullSet = c(vectorA, vectorB)
  lengthA = length(vectorA)
  lengthB = length(vectorB)
  trials <- replicate(N,
                      {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)
                      testStat((fullSet[index]), fullSet[-index])  } )
  trials
}
```

Эта функция принимает два вектора и перемешивает их содержимое вместе, затем выполняет функцию `testStat` на перетасованных векторах. Результат `teststat` добавляется

к `trials` , что является возвращаемым значением.

Он делает это `N = 10^5` раз. Обратите внимание, что значение `N` вполне могло быть параметром функции.

Это оставляет нам новый набор данных, `trials` , набор средств, которые могут возникнуть, если между этими двумя переменными действительно нет взаимосвязи.

Теперь, чтобы определить нашу тестовую статистику:

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

Выполните тест:

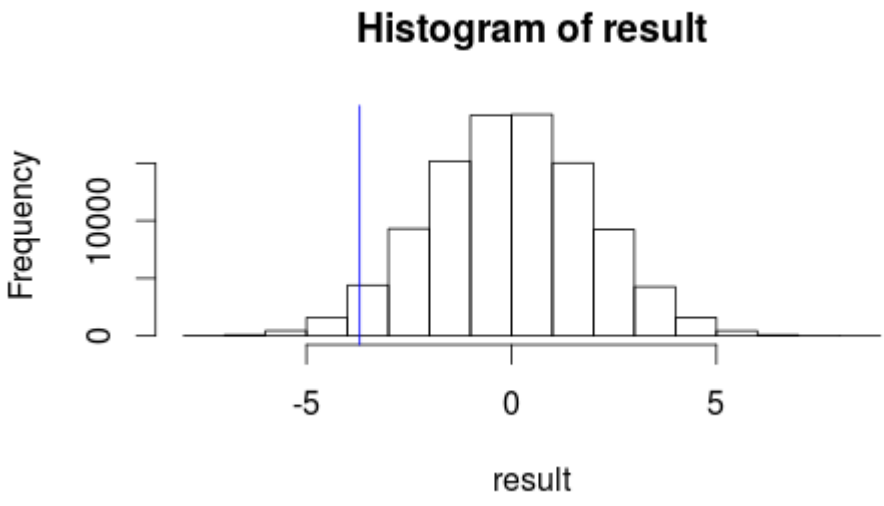
```
result = permutationTest(vec1, vec2, subtractMeans)
```

Рассчитайте нашу фактическую наблюдаемую среднюю разницу:

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

Посмотрим, как выглядит наше наблюдение на гистограмме нашей тестовой статистики.

```
hist(result)
abline(v=observedMeanDifference, col = "blue")
```



Это не *похоже на то,* что наш наблюдаемый результат, скорее всего, произойдет случайно.

Мы хотим рассчитать p-значение, вероятность первоначального наблюдаемого результата, если они не связаны между двумя переменными.

```
pValue = 2*mean(result >= (observedMeanDifference))
```

Давайте сломаем это немного:

```
result >= (observedMeanDifference)
```

Создает логический вектор, например:

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

С `TRUE` каждый раз значение `result` больше или равно `observedMean` .

`mean` функции будет интерпретировать этот вектор как `1` для `TRUE` и `0` для `FALSE` , и дать нам процент `1` в смеси, то есть количество раз, когда наша перетасованная векторная средняя разница превысила или сравнила то, что мы наблюдали.

Наконец, умножим на 2, потому что распределение нашей тестовой статистики очень симметрично, и мы действительно хотим знать, какие результаты «более экстремальные», чем наш наблюдаемый результат.

`0.06093939` только вывести значение p, которое оказалось равным `0.06093939` .
Интерпретация этого значения субъективна, но я бы сказал, что это похоже на то, что витамин С способствует росту зубов намного больше, чем апельсиновый сок.

Прочитайте Выполнение теста перестановки онлайн: https://riptutorial.com/ru/r/topic/3216/выполнение-теста-перестановки

глава 46: Выражение: parse + eval

замечания

Функция `parse` преобразует текст и файлы в выражения.

Функция `eval` оценивает выражения.

Examples

Выполнить код в формате строки

В этом примере мы хотим выполнить код, который хранится в строковом формате.

```
# the string
str <- "1+1"

# A string is not an expression.
is.expression(str)
[1] FALSE

eval(str)
[1] "1+1"

# parse convert string into expressions
parsed.str <- parse(text="1+1")

is.expression(parsed.str)
[1] TRUE

eval(parsed.str)
[1] 2
```

Прочитайте Выражение: parse + eval онлайн: https://riptutorial.com/ru/r/topic/5746/выражение--parse-plus-eval

глава 47: Генератор случайных чисел

Examples

Случайные подстановки

Для генерации произвольной перестановки 5 чисел:

```
sample(5)
# [1] 4 5 3 1 2
```

Для генерации произвольной перестановки любого вектора:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

Можно также использовать пакет `pracma`

```
randperm(a, k)
# Generates one random permutation of k of the elements a, if a is a vector,
# or of 1:a if a is a single integer.
# a: integer or numeric vector of some length n.
# k: integer, smaller as a or length(a).

# Examples
library(pracma)
randperm(1:10, 3)
[1] 3 7 9

randperm(10, 10)
[1]  4  5 10  8  2  7  6  9  3  1

randperm(seq(2, 10, by=2))
[1]  6  4 10  2  8
```

Воспроизводимость генератора случайных чисел

Когда вы ожидаете, что кто-то воспроизведет R-код, который имеет в нем случайные элементы, `set.seed()` станет очень удобной. Например, эти две строки всегда будут производить разные выходные данные (потому что это целая точка генераторов случайных чисел):

```
> sample(1:10,5)
[1]  6  9  2  7 10
> sample(1:10,5)
[1]  7  6  1  2 10
```

Эти два будут также производить разные результаты:

```
> rnorm(5)
[1]  0.4874291  0.7383247  0.5757814 -0.3053884  1.5117812
> rnorm(5)
[1]  0.38984324 -0.62124058 -2.21469989  1.12493092 -0.04493361
```

Однако, если мы установим семя в нечто одинаковое в обоих случаях (большинство людей используют 1 для простоты), мы получаем два одинаковых образца:

```
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
> set.seed(1)
> sample(letters,2)
[1] "g" "j"
```

и то же, что, скажем, `rexp()` рисует:

```
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
```

Генерация случайных чисел с использованием различных функций плотности

Ниже приведены примеры генерации 5 случайных чисел с использованием различных вероятностных распределений.

Равномерное распределение между 0 и 10

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Нормальное распределение с 0 средним и стандартным отклонением 1

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299  1.20032409
```

Биномиальное распределение с 10 испытаниями и вероятностью успеха 0,5

```
rbinom(5, size=10, prob=0.5)
```

```
[1] 4 3 5 2 3
```

Геометрическое распределение с 0,2 вероятностью успеха

```
rgeom(5, prob=0.2)
[1] 14  8 11  1  3
```

Гипергеометрическое распределение с тремя белыми шарами, 10 черными шарами и 5 ничьей

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Отрицательное биномиальное распределение с 10 испытаниями и вероятностью успеха 0,8

```
rnbinom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Распределение Пуассона со средним и дисперсией (лямбда) 2

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Экспоненциальное распределение со скоростью 1,5

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Распределение логистики с 0 местоположением и шкалой 1

```
rlogis(5, location=0, scale=1)
[1]  0.9498992 -1.0287433 -0.4192311  0.7028510 -1.2095458
```

Ши-квадрат распределения с 15 степенями свободы

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Бета- распределение с параметрами формы a = 1 и b = 0,5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Гамма- распределение с параметром формы 3 и шкалой = 0,5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Распределение Коши с 0 местоположением и шкалой 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446  8.71016696 10.60293284 -0.68017185
```

Лог-нормальное распределение с 0 стандартным и стандартным отклонением 1 (по шкале журнала)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Распределение Вейбулла с параметром формы 0,5 и шкалой 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Распределение Вилькоксона с 10 наблюдениями в первом образце и 20 в секунду.

```
rwilcox(5, 10, 20)
[1] 111  88  93 100 124
```

Многочленное распределение с 5 объектами и 3 ящиками

с использованием указанных вероятностей

```
rmultinom(5, size=5, prob=c(0.1,0.1,0.8))
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    2    0    1    1    0
[3,]    3    5    3    3    5
```

Прочитайте Генератор случайных чисел онлайн: https://riptutorial.com/ru/r/topic/1578/генератор-случайных-чисел

глава 48: Дата и время

Вступление

R поставляется с классами для дат, расписаний даты и времени; см. « `?Dates` , « `?DateTimeClasses` , « `?difftime` и следуйте разделу «См. также» этих документов для дальнейшей документации. Связанные документы: даты и классы времени .

замечания

Классы

- POSIXct

  Класс даты, POSIXct хранит время в секундах с эпохи UNIX в `1970-01-01 00:00:00 UTC` . Это формат, возвращаемый при `Sys.Time()` текущего времени с помощью `Sys.Time()` .

- POSIXlt

  Класс даты и времени хранит список дней, месяца, года, часа, минуты, секунды и т. Д. Это формат, возвращаемый `strptime` .

- Дата Единственный класс даты сохраняет дату как число с плавающей запятой.

Выбор формата даты и времени

POSIXct - единственная опция в tidyverse и мире UNIX. Это быстрее и занимает меньше памяти, чем POSIXlt.

```
origin = as.POSIXct("1970-01-01 00:00:00", format ="%Y-%m-%d %H:%M:%S", tz = "UTC")

origin
## [1] "1970-01-01 UTC"

origin + 47
## [1] "1970-01-01 00:00:47 UTC"

as.numeric(origin)      # At epoch
## 0

as.numeric(Sys.time()) # Right now (output as of July 21, 2016 at 11:47:37 EDT)
## 1469116057

posixlt = as.POSIXlt(Sys.time(), format ="%Y-%m-%d %H:%M:%S", tz = "America/Chicago")
```

```
# Conversion to POISXct
posixct = as.POSIXct(posixlt)
posixct

# Accessing components
posixlt$sec   # Seconds 0-61
posixlt$min   # Minutes 0-59
posixlt$hour  # Hour 0-23
posixlt$mday  # Day of the Month 1-31
posixlt$mon   # Months after the first of the year 0-11
posixlt$year  # Years since 1900.

ct = as.POSIXct("2015-05-25")
lt = as.POSIXlt("2015-05-25")

object.size(ct)
# 520 bytes
object.size(lt)
# 1816 bytes
```

Специализированные пакеты

- в любой момент
- data.table IDate и ITime
- fasttime
- lubridate
- nanotime

Examples

Текущая дата и время

R имеет доступ к текущей дате, времени и часовому поясу:

```
Sys.Date()             # Returns date as a Date object

## [1] "2016-07-21"

Sys.time()             # Returns date & time at current locale as a POSIXct object

## [1] "2016-07-21 10:04:39 CDT"

as.numeric(Sys.time()) # Seconds from UNIX Epoch (1970-01-01 00:00:00 UTC)

## [1] 1469113479

Sys.timezone()         # Time zone at current location

## [1] "Australia/Melbourne"
```

Используйте `OlsonNames()` для просмотра имен часовых поясов в базе данных Olson / IANA в текущей системе:

```
str(OlsonNames())
## chr [1:589] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"
"Africa/Asmara" "Africa/Asmera" "Africa/Bamako" ...
```

Перейти к концу месяца

Предположим, мы хотим перейти в последний день месяца, эта функция поможет:

```
eom <- function(x, p=as.POSIXlt(x)) as.Date(modifyList(p, list(mon=p$mon + 1, mday=0)))
```

Тестовое задание:

```
x <- seq(as.POSIXct("2000-12-10"),as.POSIXct("2001-05-10"),by="months")
> data.frame(before=x,after=eom(x))
      before      after
1 2000-12-10 2000-12-31
2 2001-01-10 2001-01-31
3 2001-02-10 2001-02-28
4 2001-03-10 2001-03-31
5 2001-04-10 2001-04-30
6 2001-05-10 2001-05-31
>
```

Использование даты в строчном формате:

```
> eom('2000-01-01')
[1] "2000-01-31"
```

Перейти в первый день месяца

Предположим, мы хотим отправиться в первый день месяца:

```
date <- as.Date("2017-01-20")

> as.POSIXlt(cut(date, "month"))
[1] "2017-01-01 EST"
```

Перемещать дату на несколько месяцев последовательно по месяцам

Допустим, мы хотим , чтобы переместить заданную дату `num` месяцев. Мы можем определить следующую функцию, которая использует пакет `mondate` :

```
moveNumOfMonths <- function(date, num) {
    as.Date(mondate(date) + num)
}
```

Он последовательно перемещает месячную часть даты и корректирует день, если дата относится к последнему дню месяца.

Например:

Назад один месяц:

```
> moveNumOfMonths("2017-10-30",-1)
[1] "2017-09-30"
```

Назад два месяца:

```
> moveNumOfMonths("2017-10-30",-2)
[1] "2017-08-30"
```

Переслать два месяца:

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

Он движется два месяца с последнего дня февраля, поэтому последний день апреля.

Давайте посмотрим, как это работает для операций назад и вперед, когда это последний день месяца:

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

Поскольку ноябрь имеет 30 дней, мы получаем одну и ту же дату в обратной операции, но:

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

Поскольку январь имеет 31 день, то переход на два месяца с последнего дня ноября будет последним днем января.

Прочитайте Дата и время онлайн: https://riptutorial.com/ru/r/topic/1157/дата-и-время

глава 49: знак вставки

Вступление

`caret` - это пакет R, который помогает в обработке данных, необходимой для проблем машинного обучения. Это означает классификацию и регрессионную подготовку. При создании моделей для реального набора данных существуют некоторые задачи, отличные от реального алгоритма обучения, который необходимо выполнить, например, очистка данных, обработка неполных наблюдений, проверка нашей модели на тестовом наборе и сравнение разных моделей.

`caret` помогает в этих сценариях, независимо от используемых алгоритмов обучения.

Examples

предварительная обработка

Предварительная обработка в карете выполняется с помощью функции `preProcess()` . Для объекта `x` типа матрицы или типа данных `preProcess()` применяет преобразования к данным обучения, которые затем могут применяться к данным тестирования.

Сердцем функции `preProcess()` является аргумент `method` . Операции метода применяются в следующем порядке:

1. Фильтр нулевой дисперсии
2. Фильтр дисперсии с нулевым значением
3. Box-Cox / Yeo-Johnson / экспоненциальная трансформация
4. Центрирование
5. пересчет
6. Спектр
7. вменение в вину
8. PCA
9. ICA
10. Пространственный знак

Ниже мы берем набор данных mtcars и выполняем центрирование, масштабирование и пространственное преобразование знака.

```
auto_index <- createDataPartition(mtcars$mpg, p = .8,
                                  list = FALSE,
                                  times = 1)

mt_train <- mtcars[auto_index,]
mt_test <- mtcars[-auto_index,]
```

```
process_mtcars <- preProcess(mt_train, method = c("center","scale","spatialSign"))

mtcars_train_transf <- predict(process_mtcars, mt_train)
mtcars_test_tranf <- predict(process_mtcars,mt_test)
```

Прочитайте знак вставки онлайн: https://riptutorial.com/ru/r/topic/4271/знак-вставки

глава 50: Иерархическая кластеризация с hclust

Вступление

Пакет `stats` предоставляет функцию `hclust` для выполнения иерархической кластеризации.

замечания

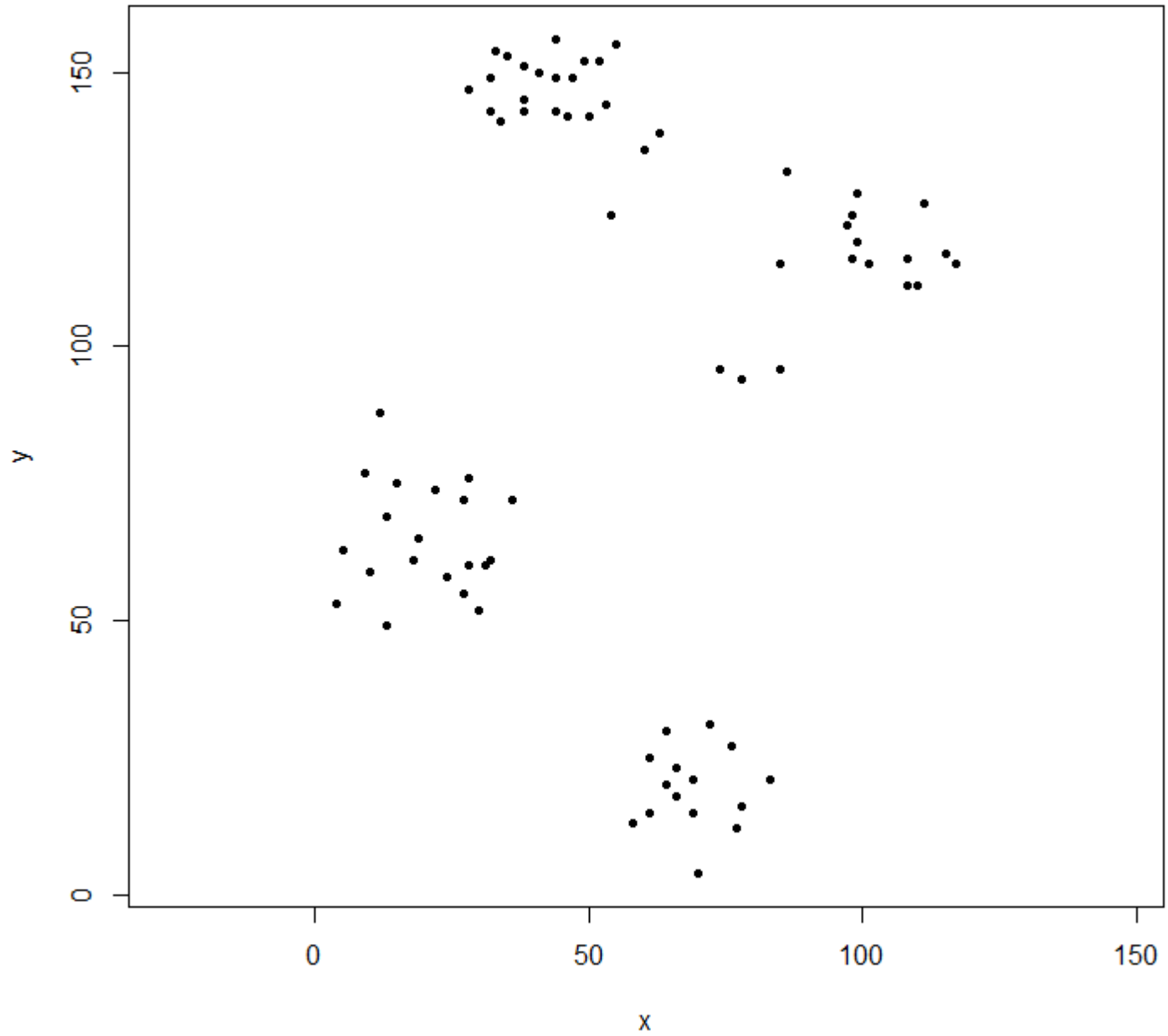
Помимо hclust, доступны другие методы, см. Представление пакета CRAN для кластеризации .

Examples

Пример 1 - Основное использование hclust, отображение дендрограммы, кластеры сюжетов

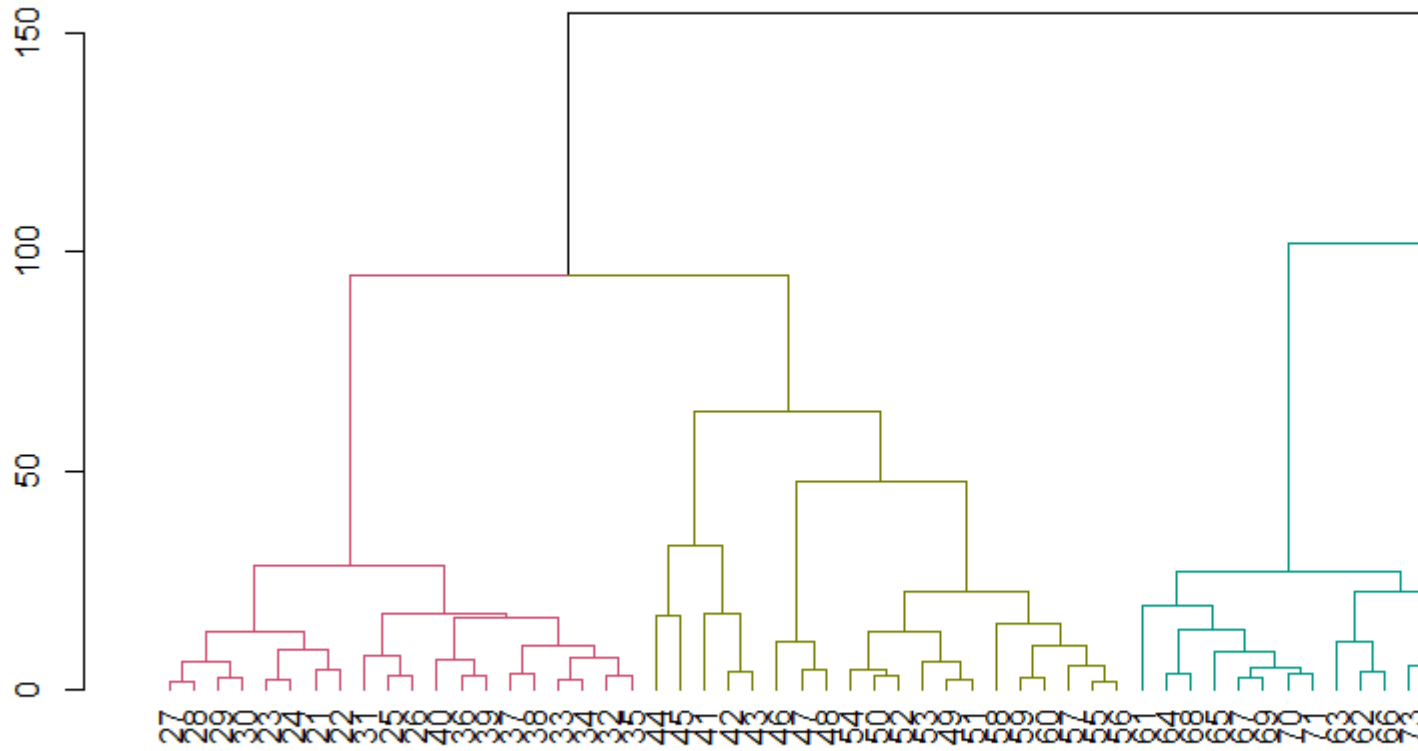
Библиотека кластера содержит данные ruspini - стандартный набор данных для иллюстрации кластерного анализа.

```
library(cluster)                ## to get the ruspini data
plot(ruspini, asp=1, pch=20)    ## take a look at the data
```



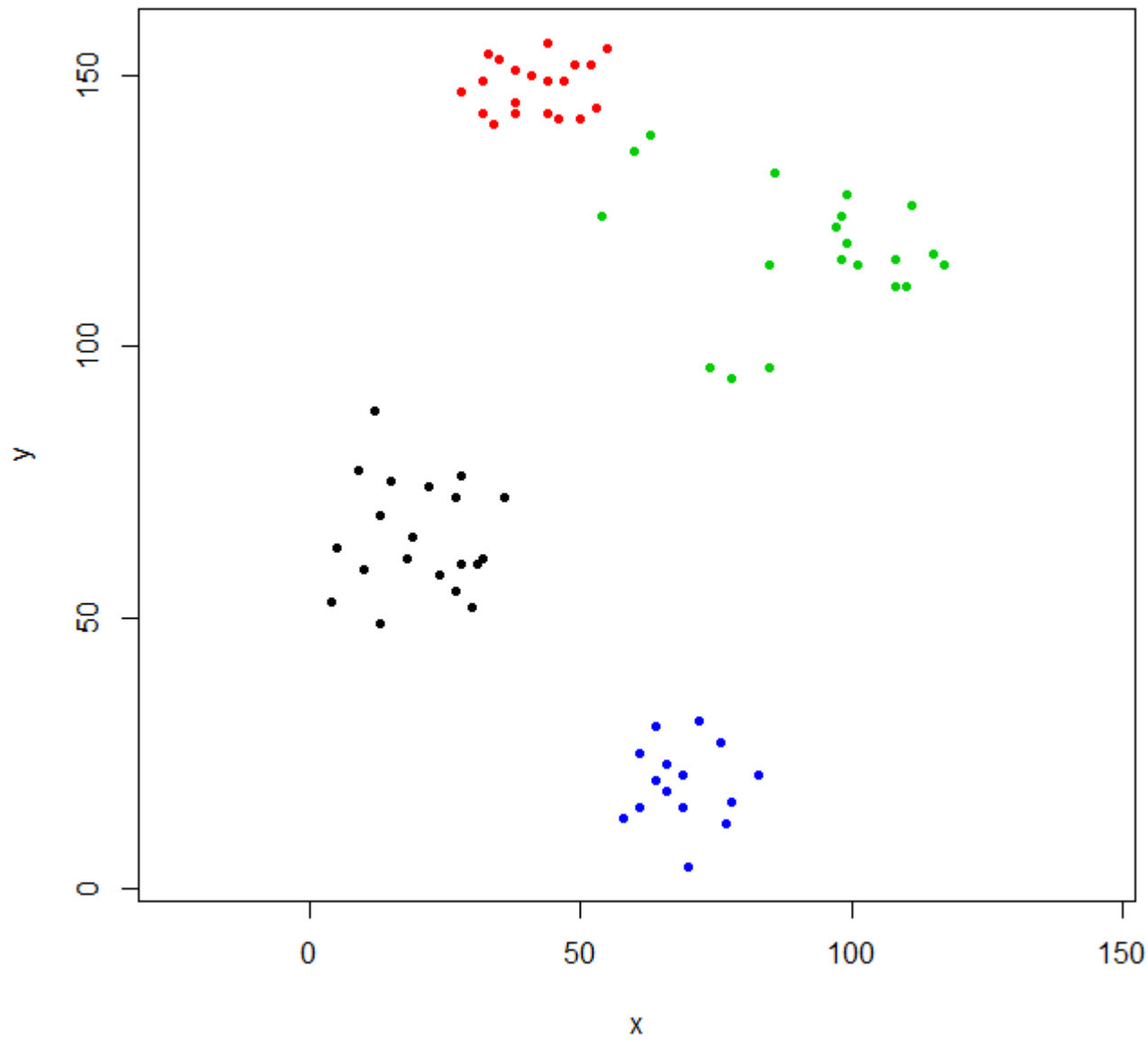
hclust ожидает матрицу расстояния, а не исходные данные. Мы вычисляем дерево, используя параметры по умолчанию и отображаем его. Параметр зависания выравнивает все листья дерева вдоль базовой линии.

```
ruspini_hc_defaults <- hclust(dist(ruspini))
dend <- as.dendrogram(ruspini_hc_defaults)
if(!require(dendextend)) install.packages("dendextend"); library(dendextend)
dend <- color_branches(dend, k = 4)
plot(dend)
```



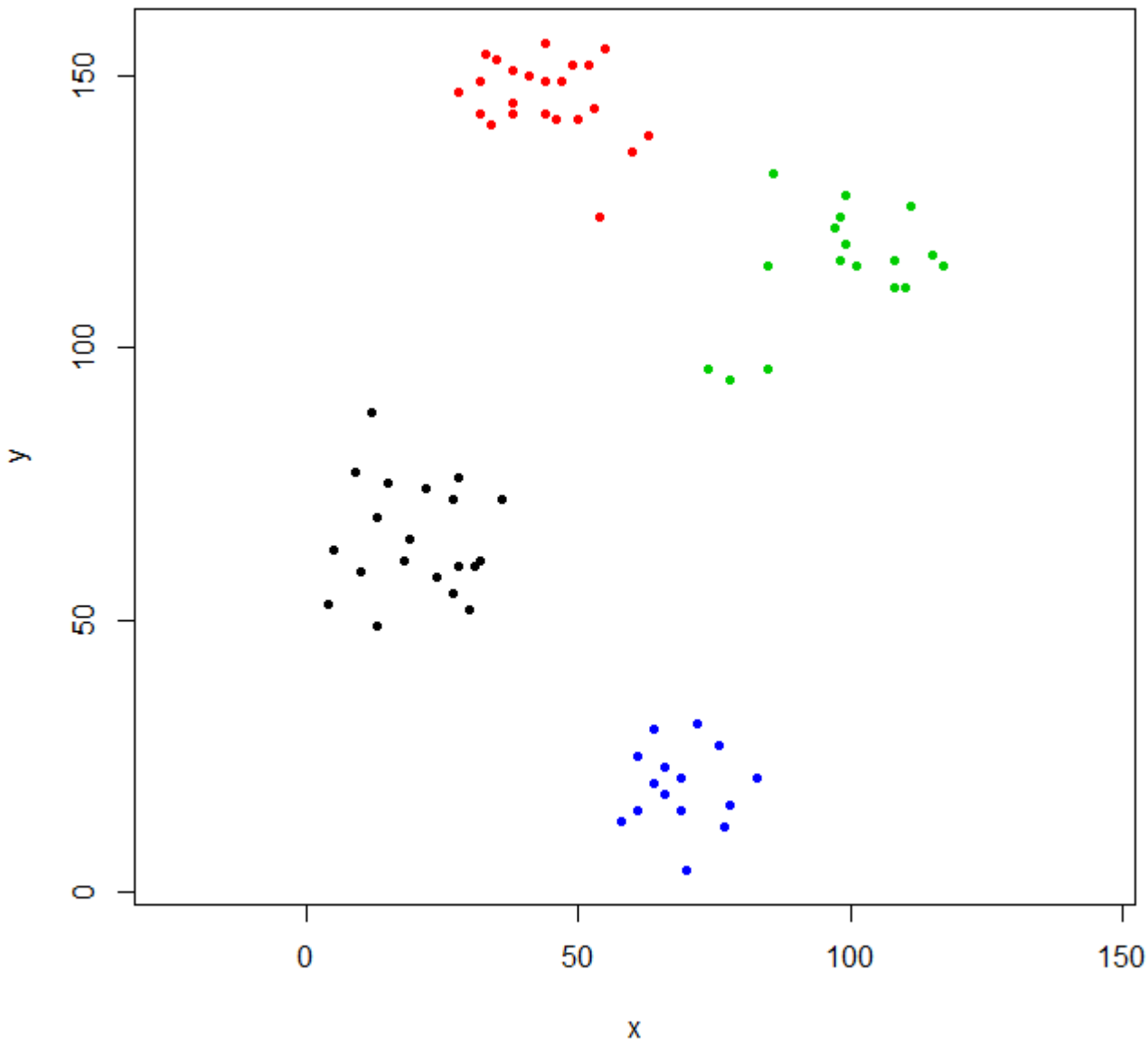
Вырежьте дерево, чтобы дать четыре кластера и поменять данные, окрашивающие точки кластером. k - желаемое количество кластеров.

```
rhc_def_4 = cutree(ruspini_hc_defaults,k=4)
plot(ruspini, pch=20, asp=1, col=rhc_def_4)
```



Эта кластеризация немного странная. Мы можем получить лучшую кластеризацию, сначала масштабируя данные.

```
scaled_ruspini_hc_defaults = hclust(dist(scale(ruspini)))
srhc_def_4 = cutree(scaled_ruspini_hc_defaults,4)
plot(ruspini, pch=20, asp=1, col=srhc_def_4)
```



Показатель разницы по умолчанию для сравнения кластеров «завершен». Вы можете указать другую меру с помощью параметра метода.

```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

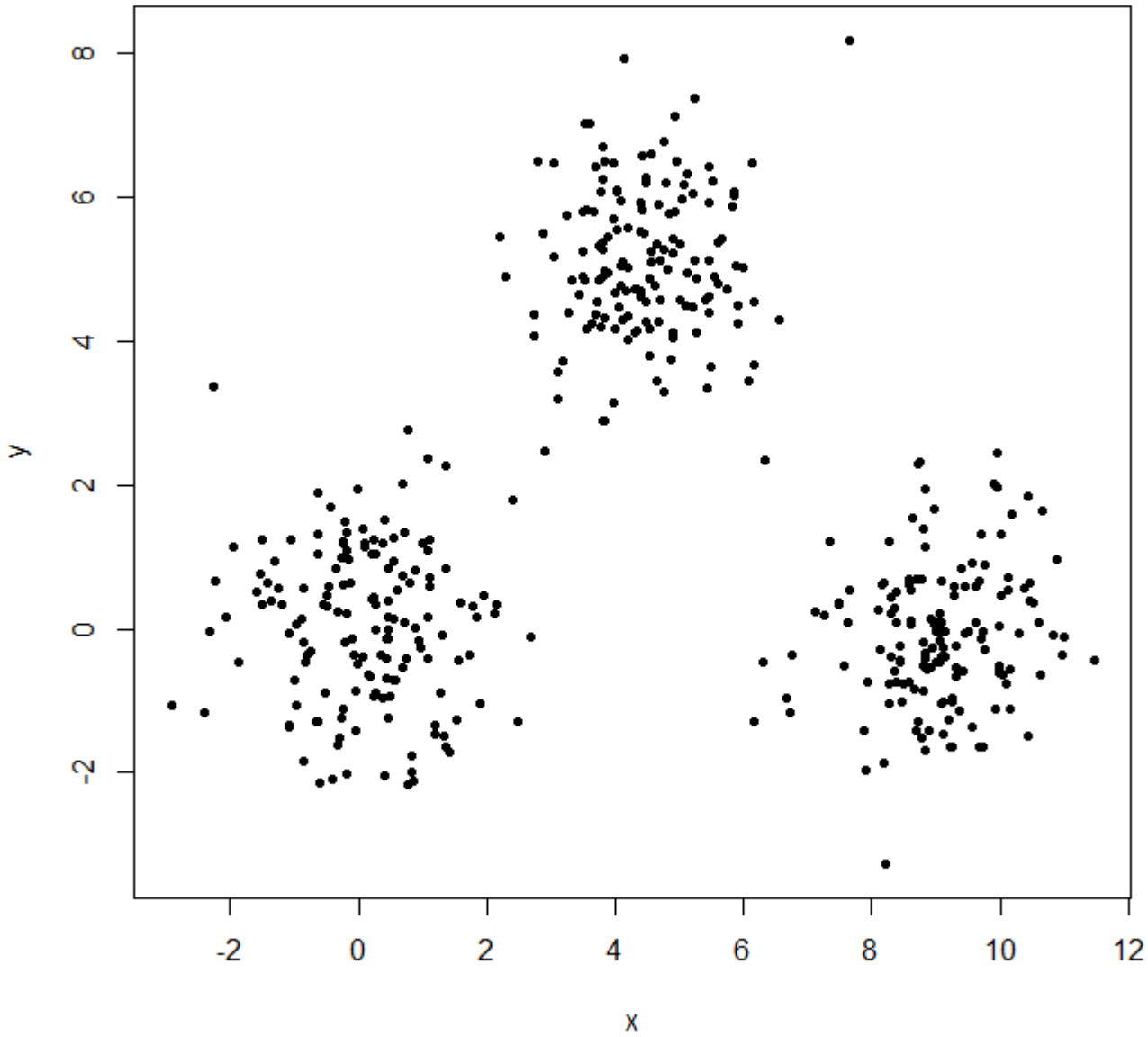
Пример 2 - hclust и outliers

С иерархической кластеризацией выбросы часто отображаются как одноточечные кластеры.

Сгенерируйте три гауссовских распределения, чтобы проиллюстрировать эффект выбросов.

```
set.seed(656)
x = c(rnorm(150, 0, 1), rnorm(150,9,1), rnorm(150,4.5,1))
y = c(rnorm(150, 0, 1), rnorm(150,0,1), rnorm(150,5,1))
XYdf = data.frame(x,y)
```

```
plot(XYdf, pch=20)
```



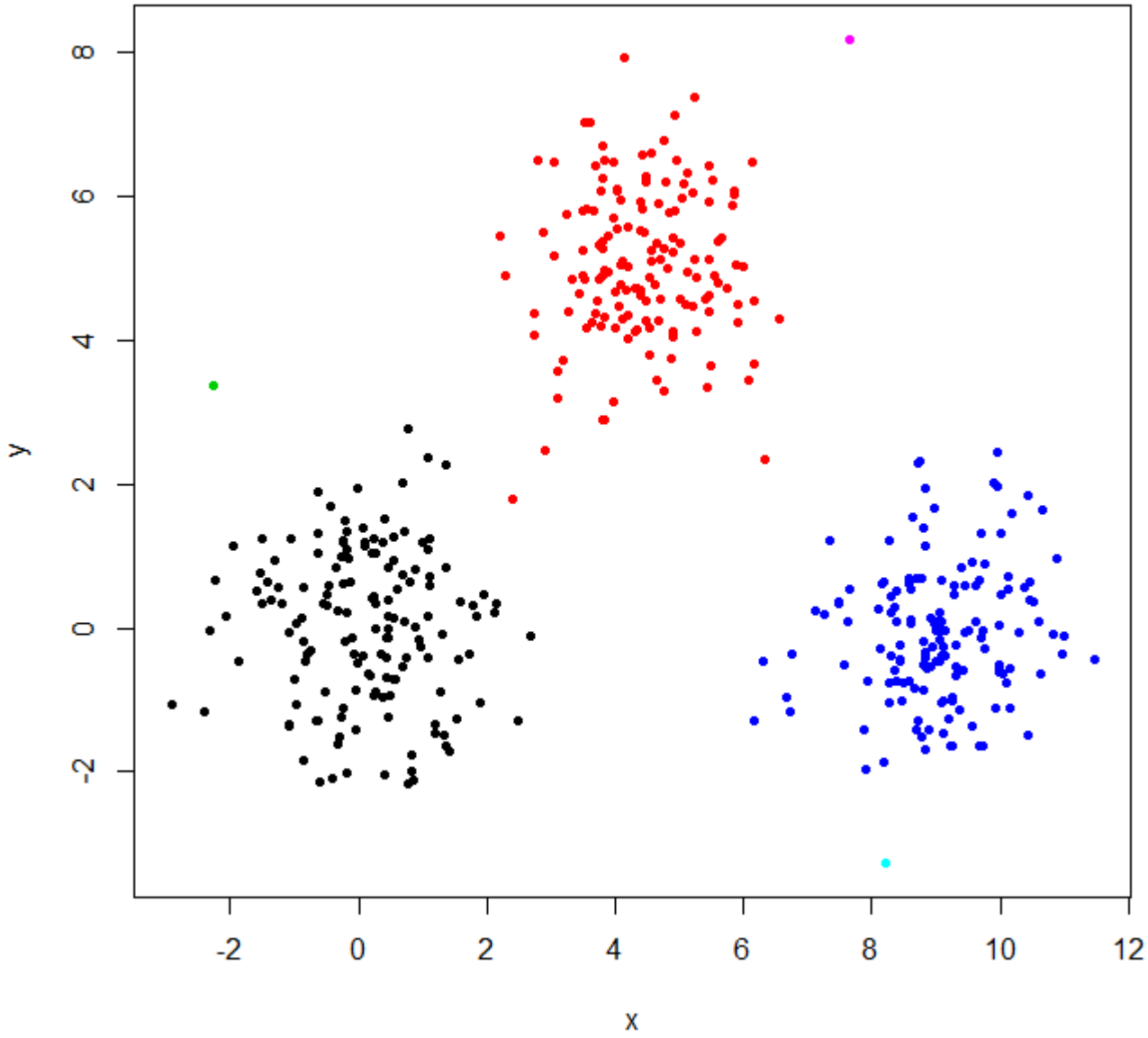
Создайте структуру кластера, разделите его на три кластера.

```
XY_sing = hclust(dist(XYdf), method="single")
XYs3 = cutree(XY_sing,k=3)
table(XYs3)
XYs3
  1   2   3
448   1   1
```

hclust обнаружил два выброса и поместил все остальное в один большой кластер. Чтобы получить «реальные» кластеры, вам может потребоваться установить k выше.

```
XYs6 = cutree(XY_sing,k=6)
```

```
table(XYs6)
XYs6
  1   2   3   4   5   6
148 150   1 149   1   1
plot(XYdf, pch=20, col=XYs6)
```



В этой статье StackOverflow есть некоторые рекомендации о том, как выбрать количество кластеров, но помните об этом поведении в иерархической кластеризации.

Прочитайте Иерархическая кластеризация с hclust онлайн:
https://riptutorial.com/ru/r/topic/8084/иерархическая-кластеризация-с-hclust

глава 51: Иерархическое линейное моделирование

Examples

базовая модель

извинения : *поскольку я не знаю канала для обсуждения / предоставления отзывов о запросах на улучшение, я собираюсь задать свой вопрос здесь. Пожалуйста, не стесняйтесь указать на лучшее место для этого!* @DataTx заявляет, что это «полностью неясное, неполное или имеет серьезные проблемы с форматированием». Поскольку я не вижу больших проблем с форматированием (:-)), немного больше рекомендаций о том, что ожидается здесь для улучшения ясности или полноты, и почему то, что здесь является unsalvageable, было бы полезно.

Первичные пакеты для установки иерархических (альтернативно «смешанных» или «многоуровневых») линейных моделей в R являются `nlme` (старше) и `lme4` (новее). Эти пакеты отличаются многими незначительными способами, но обычно должны приводить к очень похожим моделям.

```
library(nlme)
library(lme4)
m1.nlme <- lme(Reaction~Days,random=~Days|Subject,data=sleepstudy,method="REML")
m1.lme4 <- lmer(Reaction~Days+(Days|Subject),data=sleepstudy,REML=TRUE)
all.equal(fixef(m1.nlme),fixef(m1.lme4))
## [1] TRUE
```

Различия:

- синтаксис формулы несколько отличается
- `nlme` (до сих пор) несколько лучше документирован (например, *модели* `nlme` и Bates 2000 *Mixed-effects в S-PLUS* , однако см. Bates *et al.,* 2015 *Journal of Statistical Software* / `vignette("lmer",package="lme4")` для `lme4`)
- `lme4` быстрее и позволяет упростить установку скрещенных случайных эффектов
- `nlme` предоставляет значения p для линейных смешанных моделей из коробки, `lme4` требует дополнительных пакетов, таких как `lmerTest` или `afex`
- `nlme` позволяет моделировать `nlme` или остаточные корреляции (в пространстве / времени / филогенезе)

Неофициальный FAQ GLMM предоставляет дополнительную информацию, хотя он ориентирован на *обобщенные* линейные смешанные модели (GLMM).

Прочитайте Иерархическое линейное моделирование онлайн:

https://riptutorial.com/ru/r/topic/3460/иерархическое-линейное-моделирование

глава 52: Извлечение и листинг файлов в сжатых архивах

Examples

Извлечение файлов из архива .zip

Распаковка zip-архива выполняется с помощью функции `unzip` из пакета `utils` (который входит в базу R).

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

Это приведет к извлечению всех файлов в `"bar.zip"` в каталог `"foo"` , который будет создан при необходимости. Расширение Tilde выполняется автоматически из вашего рабочего каталога. Кроме того, вы можете передать все имя пути в zip-файл.

Список файлов в архиве .zip

Листинг файлов в zip-архиве выполняется с помощью функции `unzip` из пакета `utils` (который входит в базу R).

```
unzip(zipfile = "bar.zip", list = TRUE)
```

Это `"bar.zip"` все файлы в `"bar.zip"` и не извлечет их. Расширение Tilde выполняется автоматически из вашего рабочего каталога. Кроме того, вы можете передать все имя пути в zip-файл.

Листинг файлов в .tar архиве

Листинг файлов в tar-архиве выполняется с помощью функции `untar` из пакета `utils` (который включен в базу R).

```
untar(zipfile = "bar.tar", list = TRUE)
```

Это `"bar.tar"` все файлы в `"bar.tar"` и не извлечет их. Расширение Tilde выполняется автоматически из вашего рабочего каталога. Кроме того, вы можете передать все имя пути в tarfile.

Извлечение файлов из .tar-архива

Извлечение файлов из tar-архива выполняется с помощью функции `untar` из пакета `utils` (

который включен в базу R).

```
untar(tarfile = "bar.tar", exdir = "./foo")
```

Это приведет к извлечению всех файлов в `"bar.tar"` в каталог `"foo"` , который будет создан при необходимости. Расширение Tilde выполняется автоматически из вашего рабочего каталога. Кроме того, вы можете передать все имя пути в tarfile.

Извлеките все .zip-архивы в каталог

С помощью простого цикла `for` все zip-архивы в каталоге могут быть извлечены.

```
for (i in dir(pattern=".zip$"))
    unzip(i)
```

Функция `dir` создает вектор символов имен файлов в каталоге, соответствующем шаблону регулярного выражения, заданному `pattern` . Этот вектор зацикливается с индексом `i` , используя функцию `unzip` для извлечения каждого zip-архива.

Прочитайте Извлечение и листинг файлов в сжатых архивах онлайн: https://riptutorial.com/ru/r/topic/4323/извлечение-и-листинг-файлов-в-сжатых-архивах

глава 53: Издательский

Вступление

Существует много способов форматирования R-кода, таблиц и графиков для публикации.

замечания

R часто хотят публиковать анализ и результаты воспроизводимым образом. См. « Воспроизводимость R» .

Examples

Таблицы форматирования

Здесь «таблица» означает широко (охватывая `data.frame` , `table` ,

Печать в обычный текст

Печать (как показано на консоли) может быть достаточной для просмотра текстового документа в моноширинном шрифте:

Примечание. Прежде чем приводить приведенные ниже примеры данных, убедитесь, что вы находитесь в пустой папке, в которую вы можете писать. Запустите `getwd()` и прочитайте `?setwd` если вам нужно сменить папки.

```
..w = options()$width
options(width = 500) # reduce text wrapping
sink(file = "mytab.txt")
   summary(mtcars)
sink()
options(width = ..w)
rm(..w)
```

Печать таблиц с разделителями

Запись в CSV (или другой общий формат), а затем открытие в редакторе электронных таблиц для применения заключительных штрихов - это еще один вариант:

Примечание. Прежде чем приводить приведенные ниже примеры данных, убедитесь, что вы находитесь в пустой папке, в которую вы можете писать. Запустите `getwd()` и прочитайте `?setwd`

если вам нужно сменить папки.

```
write.csv(mtcars, file="mytab.csv")
```

Дополнительные ресурсы

- `knitr::kable`
- звездочет
- `tables::tabular`
- texreg
- xtable

Форматирование всех документов

`Sweave` из пакета `utils` позволяет форматировать код, прозу, графики и таблицы вместе в документе LaTeX.

Дополнительные ресурсы

- Книтр и RMarkdown

Прочитайте Издательский онлайн: https://riptutorial.com/ru/r/topic/9039/издательский

глава 54: Изменение строк путем замены

Вступление

`sub` и `gsub` используются для редактирования строк с использованием шаблонов. См. « Согласование образцов и замена» для получения дополнительных сведений о связанных функциях и регулярных выражениях для создания шаблона.

Examples

Переупорядочить строки символов, используя группы захвата

Если вы хотите изменить порядок строк символов, вы можете использовать круглые скобки в `pattern` для группировки частей строки вместе. Эти группы могут в аргументе `replacement` быть добавлены с использованием последовательных чисел.

В следующем примере показано, как вы можете переупорядочить вектор имен формы «фамилия, имя» в вектор формы «имя фамилии».

```
library(randomNames)
set.seed(1)

strings <- randomNames(5)
strings
# [1] "Sigg, Zachary"        "Holt, Jake"           "Ortega, Sandra"       "De La Torre,
Nichole"
# [5] "Perkins, Donovon"

sub("^(.+),\\s(.+)$", "\\2 \\1", strings)
# [1] "Zachary Sigg"        "Jake Holt"           "Sandra Ortega"       "Nichole De La Torre"
# [5] "Donovon Perkins"
```

Если вам нужна только фамилия, вы можете просто обратиться к первым парам круглых скобок.

```
sub("^(.+),\\s(.+)", "\\1", strings)
# [1] "Sigg"        "Holt"        "Ortega"      "De La Torre" "Perkins"
```

Устранение дублированных последовательных элементов

Предположим, мы хотим исключить дублированный элемент подпоследовательности из строки (ее может быть несколько). Например:

```
2,14,14,14,19
```

и преобразовать его в:

```
2,14,19
```

Используя `gsub` , мы можем добиться этого:

```
gsub("(\\d+)(,\\1)+","\\1", "2,14,14,14,19")
[1] "2,14,19"
```

Он работает также для нескольких повторений, например:

```
 > gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,19,20,21")
[1] "2,14,19,20,21"
```

Давайте объясним регулярное выражение:

1. `(\\d+)` : группа 1, ограниченная (), и находит любую цифру (по крайней мере одну). Помните, что нам нужно использовать двойную обратную косую черту (`\\`) здесь, потому что для символьной переменной обратная косая черта представляет собой специальный escape-символ для литеральных разделителей строк (`\"` или `\'`). `\d\` эквивалентен: `[0-9]` .
2. `,` : Знак пунктуации: `,` (мы можем включать пробелы или любой другой разделитель)
3. `\\1` : идентичная строка для группы 1, т. Е. Повторяющееся число. Если этого не произойдет, то шаблон не соответствует.

Попробуем аналогичную ситуацию: устраним последовательные повторяющиеся слова:

```
one,two,two,three,four,four,five,six
```

Затем просто замените `\d` на `\w` , где `\w` соответствует любому символу слова, включая: любую букву, цифру или подчеркивание. Это эквивалентно `[a-zA-Z0-9_]` :

```
> gsub("(\\w+)(,\\1)+", "\\1", "one,two,two,three,four,four,five,six")
[1] "one,two,three,four,five,six"
>
```

Затем вышеуказанный шаблон включает в качестве конкретного случая дублированный разряд.

Прочитайте Изменение строк путем замены онлайн: https://riptutorial.com/ru/r/topic/9219/изменение-строк-путем-замены

глава 55: Изменить форму tidyr

Вступление

tidyr имеет два инструмента для переформатирования данных: `gather` (широкий и длинный) и `spread` (длинный до широкого).

См. Раздел Изменение данных для других параметров.

Examples

Переформатировать из длинного в широкий формат с помощью spread ()

```
library(tidyr)

## example data
set.seed(123)
df <- data.frame(
  name = rep(c("firstName", "secondName"), each=4),
  numbers = rep(1:4, 2),
  value = rnorm(8)
)
df
#          name numbers       value
# 1   firstName       1 -0.56047565
# 2   firstName       2 -0.23017749
# 3   firstName       3  1.55870831
# 4   firstName       4  0.07050839
# 5  secondName       1  0.12928774
# 6  secondName       2  1.71506499
# 7  secondName       3  0.46091621
# 8  secondName       4 -1.26506123
```

Мы можем «разнести» столбец «числа» в отдельные столбцы:

```
spread(data = df,
       key = numbers,
       value = value)
#       name          1          2         3           4
# 1  firstName -0.5604756 -0.2301775 1.5587083  0.07050839
# 2 secondName  0.1292877  1.7150650 0.4609162 -1.26506123
```

Или распространите столбец «имя» на отдельные столбцы:

```
spread(data = df,
       key = name,
       value = value)
#   numbers   firstName secondName
# 1       1 -0.56047565  0.1292877
# 2       2 -0.23017749  1.7150650
```

```
# 3       3  1.55870831  0.4609162
# 4       4  0.07050839 -1.2650612
```

Измените форму с широким и длинным форматом с помощью команды gather ()

```
library(tidyr)

## example data
df <- read.table(text ="  numbers  firstName secondName
1       1  1.5862639  0.4087477
2       2  0.1499581  0.9963923
3       3  0.4117353  0.3740009
4       4 -0.4926862  0.4437916", header = T)
df
#   numbers  firstName secondName
# 1       1  1.5862639  0.4087477
# 2       2  0.1499581  0.9963923
# 3       3  0.4117353  0.3740009
# 4       4 -0.4926862  0.4437916
```

Мы можем собрать столбцы вместе, используя «числа» в качестве столбца ключа:

```
gather(data = df,
       key = numbers,
       value = myValue)
#   numbers    numbers    myValue
# 1       1  firstName  1.5862639
# 2       2  firstName  0.1499581
# 3       3  firstName  0.4117353
# 4       4  firstName -0.4926862
# 5       1 secondName  0.4087477
# 6       2 secondName  0.9963923
# 7       3 secondName  0.3740009
# 8       4 secondName  0.4437916
```

Прочитайте Изменить форму tidyr онлайн: https://riptutorial.com/ru/r/topic/9195/изменить-форму-tidyr

глава 56: интроспекция

Examples

Функции для изучения переменных

Часто в `R` вам понадобятся сведения об объекте или переменной, с которыми вы работаете. Это может быть полезно при чтении чужого кода или даже вашего собственного, особенно при использовании новых для вас пакетов.

Предположим, что мы создаем переменную `a` :

```
a <- matrix(1:9, 3, 3)
```

Какой тип данных это? Вы можете узнать

```
> class(a)
[1] "matrix"
```

Это матрица, поэтому над ней будут работать матричные операции:

```
> a %*% t(a)
      [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

Каковы размеры ? `a`

```
> dim(a)
[1] 3 3
> nrow(a)
[1] 3
> ncol(a)
[2] 3
```

Другими полезными функциями, которые работают для разных типов данных, являются `head` , `tail` и `str` :

```
> head(a, 1)
     [,1] [,2] [,3]
[1,]    1    4    7
> tail(a, 1)
     [,1] [,2] [,3]
[3,]    3    6    9
> str(a)
int [1:3, 1:3] 1 2 3 4 5 6 7 8 9
```

Они гораздо полезнее для больших объектов (например, больших наборов данных). `str` также отлично подходит для изучения вложенности списков. Теперь измените `a` так:

```
a <- c(a)
```

Класс остается прежним?

```
> class(a)
[1] "integer"
```

Нет, `a` не матрица. Я не получу хороший ответ, если я попрошу размеры сейчас:

```
> dim(a)
NULL
```

Вместо этого я могу задать длину:

```
> length(a)
[1] 9
```

Что теперь:

```
> class(a * 1.0)
[1] "numeric"
```

Часто вы можете работать с `data.frames` :

```
a <- as.data.frame(a)
names(a) <- c("var1", "var2", "var3")
```

См. Имена переменных:

```
> names(a)
[1] "var1" "var2" "var3"
```

Эти функции могут помочь во многих отношениях при использовании `R`

Прочитайте интроспекция онлайн: https://riptutorial.com/ru/r/topic/3565/интроспекция

глава 57: Использование texreg для экспорта моделей в бумажном виде

Вступление

Пакет texreg помогает экспортировать модель (или несколько моделей) в аккуратном готовом виде бумаги. Результат может быть экспортирован как HTML или .doc (MS Office Word).

замечания

связи

- Страница CRAN

Examples

Печать результатов линейной регрессии

```
# models
fit1 <- lm(mpg ~ wt, data = mtcars)
fit2 <- lm(mpg ~ wt+hp, data = mtcars)
fit3 <- lm(mpg ~ wt+hp+cyl, data = mtcars)

# export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html')


# export to doc
texreg::htmlreg(list(fit1,fit2,fit3),file='models.doc')
```

Результат выглядит как таблица в документе.

	Model 1	Model 2	Model 3
(Intercept)	37.29***	37.23***	38.75***
	(1.88)	(1.60)	(1.79)
wt	-5.34***	-3.88***	-3.17***
	(0.56)	(0.63)	(0.74)
hp		-0.03**	-0.02
		(0.01)	(0.01)
cyl			-0.94
			(0.55)
R2	0.75	0.83	0.84
Adj. R2	0.74	0.81	0.83
Num. obs.	32	32	32
RMSE	3.05	2.59	2.51

***p < 0.001, **p < 0.01, *p < 0.05

Statistical models

В функции `texreg::htmlreg()` имеется несколько дополнительных параметров. Вот пример использования наиболее полезных параметров.

```
# export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html',
                single.row = T,
                custom.model.names = LETTERS[1:3],
                leading.zero = F,
                digits = 3)
```

Какой результат в такой таблице

	A	B	C
(Intercept)	37.285 (1.878)***	37.227 (1.599)***	38.752 (1.787)***
wt	-5.344 (0.559)***	-3.878 (0.633)***	-3.167 (0.741)***
hp		-0.032 (0.009)**	-0.018 (0.012)
cyl			-0.942 (0.551)
R2	0.753	0.827	0.843
Adj. R2	0.745	0.815	0.826
Num. obs.	32	32	32
RMSE	3.046	2.593	2.512

***p < 0.001, **p < 0.01, *p < 0.05

Statistical models

Прочитайте Использование texreg для экспорта моделей в бумажном виде онлайн: https://riptutorial.com/ru/r/topic/9037/использование-texreg-для-экспорта-моделей-в-бумажном-виде

глава 58: Использование назначения труб в вашем собственном пакете% <>%: Как?

Вступление

Чтобы использовать канал в созданном пользователем пакете, он должен быть указан в NAMESPACE, как и любая другая функция, которую вы хотите импортировать.

Examples

Помещение трубы в файл служебных функций

Один из вариантов для этого - экспортировать канал из самого пакета. Это можно сделать в «традиционных» `zzz.R` или `utils.R` которые многие пакеты используют для полезных небольших функций, которые не экспортируются как часть пакета. Например, ставя:

```
#' Pipe operator
#'
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Прочитайте Использование назначения труб в вашем собственном пакете% <>%: Как? онлайн: https://riptutorial.com/ru/r/topic/10547/использование-назначения-труб-в-вашем-собственном-пакете---lt--gt----как-

глава 59: Кадры данных

Синтаксис

- data.frame (..., row.names = NULL, check.rows = FALSE, check.names = TRUE, strAsAsFactors = default.stringsAsFactors ())
- as.data.frame (x, row.names = NULL, optional = FALSE, ...) # общая функция
- as.data.frame (x, ..., stringsAsFactors = default.stringsAsFactors ()) # Метод S3 для символа класса '
- as.data.frame (x, row.names = NULL, optional = FALSE, ..., stringsAsFactors = default.stringsAsFactors ()) # Метод S3 для класса 'matrix'
- is.data.frame (x)

Examples

Создайте пустой файл data.frame

Data.frame - это особый вид списка: он *прямоугольный* . Каждый элемент (столбец) списка имеет одинаковую длину и где каждая строка имеет «имя строки». Каждый столбец имеет свой собственный класс, но класс одного столбца может отличаться от класса другого столбца (в отличие от матрицы, где все элементы должны иметь один и тот же класс).

В принципе, data.frame не может содержать строк и столбцов:

```
> structure(list(character()), class = "data.frame")
NULL
<0 rows> (or 0-length row.names)
```

Но это необычно. Для data.frame чаще встречается много столбцов и много строк. Вот кадр data.frame с тремя строками и двумя столбцами (`a` - числовой класс, а `b` - класс символов):

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame")
[1] a b
<0 rows> (or 0-length row.names)
```

Для того, чтобы print.frame был напечатан, нам нужно будет указать некоторые имена строк. Здесь мы используем только числа 1: 3:

```
> structure(list(a = 1:3, b = letters[1:3]), class = "data.frame", row.names = 1:3)
  a b
1 1 a
```

```
2 2 b
3 3 c
```

Теперь становится очевидным, что у нас есть data.frame с 3 строками и 2 столбцами. Вы можете проверить это, используя `nrow()` , `ncol()` и `dim()` :

```
> x <- structure(list(a = numeric(3), b = character(3)), class = "data.frame", row.names =
1:3)
> nrow(x)
[1] 3
> ncol(x)
[1] 2
> dim(x)
[1] 3 2
```

R предоставляет две другие функции (помимо `structure()`), которые можно использовать для создания data.frame. Первый называется интуитивно, `data.frame()` . Он проверяет, являются ли допустимые имена столбцов действительными, что элементы списка имеют одинаковую длину и поставляет некоторые автоматически сгенерированные имена строк. Это означает, что вывод `data.frame()` теперь может быть точно таким, какой вы ожидаете:

```
> str(data.frame("a a a" = numeric(3), "b-b-b" = character(3)))
'data.frame':   3 obs. of  2 variables:
 $ a.a.a: num  0 0 0
 $ b.b.b: Factor w/ 1 level "": 1 1 1
```

Другая функция называется `as.data.frame()` . Это можно использовать для принуждения объекта, который не является data.frame, в data.frame, запустив его через `data.frame()` . В качестве примера рассмотрим матрицу:

```
> m <- matrix(letters[1:9], nrow = 3)
> m
     [,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

И результат:

```
> as.data.frame(m)
  V1 V2 V3
1  a  d  g
2  b  e  h
3  c  f  i
> str(as.data.frame(m))
'data.frame':   3 obs. of  3 variables:
 $ V1: Factor w/ 3 levels "a","b","c": 1 2 3
 $ V2: Factor w/ 3 levels "d","e","f": 1 2 3
 $ V3: Factor w/ 3 levels "g","h","i": 1 2 3
```

Подстановка строк и столбцов из кадра данных

Синтаксис для доступа к строкам и столбцам: `[` , `[[` , и `$`

В этом разделе рассматривается наиболее распространенный синтаксис для доступа к определенным строкам и столбцам фрейма данных. Это

- Подобно `matrix` с одиночными скобками `data[rows, columns]`
  - Использование номеров строк и столбцов
  - Использование имен столбцов (и строк)
- Как `list` :
  - С помощью отдельных скобок `data[columns]` для получения кадра данных
  - С двойными скобками `data[[one_column]]` чтобы получить вектор
- С `$` для одного столбца `data$column_name`

Для иллюстрации мы будем использовать встроенный `mtcars` данных `mtcars` .

Как и матрица: `data[rows, columns]`

С числовыми индексами

Используя встроенные `mtcars` фрейма `mtcars` , мы можем извлекать строки и столбцы, используя `[]` скобки с включенной запятой. Индексами перед запятой являются строки:

```
# get the first row
mtcars[1, ]
# get the first five rows
mtcars[1:5, ]
```

Аналогично, после запятой столбцы:

```
# get the first column
mtcars[, 1]
# get the first, third and fifth columns:
mtcars[, c(1, 3, 5)]
```

Как показано выше, если строки или столбцы оставлены пустыми, все будут выбраны. `mtcars[1, ]` указывает первую строку со *всеми* столбцами.

С именами столбцов (и строк)

Пока это идентично тому, как доступны строки и столбцы матриц. При использовании `data.frame` s большую часть времени предпочтительно использовать имя столбца для индекса столбца. Это делается с помощью `character` с именем столбца вместо `numeric` с

номером столбца:

```
# get the mpg column
mtcars[, "mpg"]
# get the mpg, cyl, and disp columns
mtcars[, c("mpg", "cyl", "disp")]
```

Хотя реже, имена строк также могут быть использованы:

```
mtcars["Mazda Rx4", ]
```

Строки и столбцы вместе

Аргументы строки и столбца могут использоваться вместе:

```
# first four rows of the mpg column
mtcars[1:4, "mpg"]

# 2nd and 5th row of the mpg, cyl, and disp columns
mtcars[c(2, 5), c("mpg", "cyl", "disp")]
```

Предупреждение о размерах:

При использовании этих методов, если вы извлекаете несколько столбцов, вы получите обратный кадр данных. Однако, если вы извлекаете *один* столбец, вы получите вектор, а не фрейм данных по умолчанию.

```
## multiple columns returns a data frame
class(mtcars[, c("mpg", "cyl")])
# [1] "data.frame"
## single column returns a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

Существует два пути. Один из них - рассматривать кадр данных как список (см. Ниже), а другой - добавить аргумент `drop = FALSE` . Это говорит R, чтобы не «удалить неиспользуемые измерения»:

```
class(mtcars[, "mpg", drop = FALSE])
# [1] "data.frame"
```

Обратите внимание, что матрицы работают одинаково - по умолчанию один столбец или строка будет вектором, но если вы укажете `drop = FALSE` вы можете сохранить его как одностоечную или однострочную матрицу.

Как список

Кадры данных по существу представляют собой `list` s, т. Е. Они представляют собой список векторов столбцов (все они должны иметь одинаковую длину). Списки могут быть подмножеством с использованием отдельных скобок `[` для вспомогательного списка или двойных скобок `[[` для одного элемента.

С отдельными скобками `data[columns]`

Когда вы используете одиночные скобки и никакие запятые, вы получите столбец назад, потому что кадры данных представляют собой списки столбцов.

```
mtcars["mpg"]
mtcars[c("mpg", "cyl", "disp")]
my_columns <- c("mpg", "cyl", "hp")
mtcars[my_columns]
```

Одиночные скобки, *такие как список* или отдельные скобки, *такие как матрица*

Разница между `data[columns]` и `data[, columns]` заключается в том, что при обработке `data.frame` в виде `list` (без запятой в скобках) возвращенный объект будет *`data.frame`* . Если вы используете запятую для обработки `data.frame` как в `matrix` то выбор одного столбца возвращает вектор, но выбор нескольких столбцов вернет `data.frame` .

```
## When selecting a single column
## like a list will return a data frame
class(mtcars["mpg"])
# [1] "data.frame"
## like a matrix will return a vector
class(mtcars[, "mpg"])
# [1] "numeric"
```

С двойными скобками `data[[one_column]]`

Чтобы извлечь один столбец *в качестве вектора* при обработке вашего `data.frame` в виде `list` , вы можете использовать двойные скобки `[[` . Это будет работать только для одного столбца за раз.

```
# extract a single column by name as a vector
mtcars[["mpg"]]

# extract a single column by name as a data frame (as above)
mtcars["mpg"]
```

Использование `$` для доступа к столбцам

Один столбец можно извлечь с помощью магического ярлыка `$` без использования имени столбца:

```
# get the column "mpg"
```

```
mtcars$mpg
```

Столбцы, к которым обращаются `$` , всегда будут векторами, а не кадрами данных.

Недостатки `$` для доступа к столбцам

`$` Может быть удобным ярлыком, особенно если вы работаете в среде (например, RStudio), которая в этом случае автоматически завершит имя столбца. **Тем не менее, у** `$` есть и недостатки: он использует *нестандартную оценку,* чтобы избежать необходимости в котировках, что означает, что она *не будет работать,* если имя столбца будет храниться в переменной.

```
my_column <- "mpg"
# the below will not work
mtcars$my_column
# but these will work
mtcars[, my_column]  # vector
mtcars[my_column]    # one-column data frame
mtcars[[my_column]]  # vector
```

Из-за этих проблем, `$` лучше всего использовать в *интерактивных* сеансах R, когда имена столбцов являются постоянными. Для использования в *программировании* , например, при написании обобщаемой функции, которая будет использоваться в разных наборах данных с разными именами столбцов, следует избегать `$` .

Также обратите внимание, что поведение по умолчанию заключается в использовании частичного соответствия только при извлечении из рекурсивных объектов (кроме окружений) на `$`

```
# give you the values of "mpg" column
# as "mtcars" has only one column having name starting with "m"
mtcars$m
# will give you "NULL"
# as "mtcars" has more than one columns having name starting with "d"
mtcars$d
```

Расширенное индексирование: отрицательные и логические индексы

Всякий раз, когда у нас есть возможность использовать числа для индекса, мы можем также использовать отрицательные числа, чтобы опустить определенные индексы или логический (логический) вектор, чтобы точно указать, какие элементы сохранить.

Отрицательные индексы опускают элементы

```
mtcars[1, ]   # first row
mtcars[ -1, ] # everything but the first row
mtcars[-(1:10), ] # everything except the first 10 rows
```

Логические векторы указывают на конкретные элементы, которые

Мы можем использовать условие, такое как `<` чтобы генерировать логический вектор и извлекать только строки, которые удовлетворяют условию:

```
# logical vector indicating TRUE when a row has mpg less than 15
# FALSE when a row has mpg >= 15
test <- mtcars$mpg < 15

# extract these rows from the data frame
mtcars[test, ]
```

Мы также можем обойти шаг сохранения промежуточной переменной

```
# extract all columns for rows where the value of cyl is 4.
mtcars[mtcars$cyl == 4, ]
# extract the cyl, mpg, and hp columns where the value of cyl is 4
mtcars[mtcars$cyl == 4, c("cyl", "mpg", "hp")]
```

Удобные функции для манипулирования data.frames

Некоторые удобные функции для манипулирования `data.frames` - это `subset()` , `transform()` , `with()` и `within()` .

подмножество

Функция `subset()` позволяет вам подмножество `data.frame` более удобным способом (подмножество также работает с другими классами):

```
subset(mtcars, subset = cyl == 6, select = c("mpg", "hp"))
                mpg  hp
Mazda RX4      21.0 110
Mazda RX4 Wag  21.0 110
Hornet 4 Drive 21.4 110
Valiant        18.1 105
Merc 280       19.2 123
Merc 280C      17.8 123
Ferrari Dino   19.7 175
```

В приведенном выше коде мы запрашиваем только строки, в которых `cyl == 6` и для столбцов `mpg` и `hp` . Вы можете добиться того же результата, используя `[]` со следующим кодом:

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

преобразование

Функция `transform()` является удобной функцией для изменения столбцов внутри `data.frame` . Например, следующий код добавляет другой столбец с именем `mpg2` с результатом `mpg^2` в `mtcars` `data.frame` :

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

с и внутри

И `with()` и `within()` позволяют оценивать выражения внутри среды `data.frame` , позволяя несколько более чистый синтаксис, сохраняя при этом использование некоторых `$` или `[]` .

Например, если вы хотите создать, изменить и / или удалить несколько столбцов в `airquality` `data.frame` :

```
aq <- within(airquality, {
    lOzone <- log(Ozone) # creates new column
    Month <- factor(month.abb[Month]) # changes Month Column
    cTemp <- round((Temp - 32) * 5/9, 1) # creates new column
    S.cT <- Solar.R / cTemp  # creates new column
    rm(Day, Temp) # removes columns
})
```

Вступление

Кадры данных, скорее всего, представляют собой структуру данных, которую вы больше всего будете использовать в своих анализах. Кадр данных представляет собой особый вид списка, в котором хранятся векторы одинаковой длины разных классов. Вы создаете кадры данных, используя функцию `data.frame` . Пример ниже показывает это, комбинируя числовой и символьный вектор в кадре данных. Он использует оператор `:` который создаст вектор, содержащий все целые числа от 1 до 3.

```
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))
df1
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
class(df1)
## [1] "data.frame"
```

Объекты фрейма данных не печатаются с кавычками, поэтому класс столбцов не всегда очевиден.

```
df2 <- data.frame(x = c("1", "2", "3"), y = c("a", "b", "c"))
df2
##   x y
## 1 1 a
## 2 2 b
## 3 3 c
```

Без дальнейшего изучения столбцы «x» в `df1` и `df2` не могут быть дифференцированы. Функция `str` может использоваться для описания объектов с большей детализацией, чем класс.

```
str(df1)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
str(df2)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: Factor w/ 3 levels "1","2","3": 1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Здесь вы видите, что `df1` является `data.frame` и имеет 3 наблюдения из 2 переменных, «x» и «y». Затем вам сообщают, что «x» имеет целое число типа данных (не важно для этого класса, но для наших целей оно ведет себя как числовое), а «y» - это фактор с тремя уровнями (другой класс данных, который мы не обсуждаем). ***Важно отметить, что по умолчанию кадры данных заставляют персонажей влиять на факторы.*** Поведение по умолчанию может быть изменено с помощью параметра `stringsAsFactors` :

```
df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)
str(df3)
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

Теперь столбец «y» является символом. Как упоминалось выше, каждый «столбец» кадра данных должен иметь одинаковую длину. Попытка создать data.frame из векторов разной длины приведет к ошибке. (Попробуйте запустить `data.frame(x = 1:3, y = 1:4)` чтобы увидеть полученную ошибку.)

В качестве тестовых примеров для кадров данных некоторые данные предоставляются R по умолчанию. Один из них - радужная оболочка, загружается следующим образом:

```
mydataframe <- iris
str(mydataframe)
```

Преобразование данных, хранящихся в списке, в один кадр данных с использованием do.call

Если у вас есть данные, хранящиеся в списке, и вы хотите преобразовать этот список в кадр `do.call` функция `do.call` - это простой способ достичь этого. Однако важно, чтобы все

элементы списка имели одинаковую длину, чтобы предотвратить непреднамеренную рециркуляцию значений.

```
dataList  <- list(1:3,4:6,7:9)
dataList
# [[1]]
# [1] 1 2 3
#
# [[2]]
# [1] 4 5 6
#
# [[3]]
# [1] 7 8 9

dataframe <- data.frame(do.call(rbind, dataList))
dataframe
#   X1 X2 X3
# 1  1  2  3
# 2  4  5  6
# 3  7  8  9
```

Он также работает, если ваш список состоит из самих фреймов данных.

```
dataframeList  <- list(data.frame(a = 1:2, b = 1:2, c = 1:2),
                       data.frame(a = 3:4, b = 3:4, c = 3:4))
dataframeList
# [[1]]
#   a b c
# 1 1 1 1
# 2 2 2 2

# [[2]]
#   a b c
# 1 3 3 3
# 2 4 4 4

dataframe      <- do.call(rbind, dataframeList)
dataframe
#   a b c
# 1 1 1 1
# 2 2 2 2
# 3 3 3 3
# 4 4 4 4
```

Преобразование всех столбцов в data.frame в класс символов

Общая задача состоит в том, чтобы преобразовать все столбцы класса data.frame в класс символов для удобства манипуляции, например, в случае отправки данных. Кадров в СУБД или слияния data.frames, содержащих факторы, в которых уровни могут различаться между входными данными. ,

Лучшее время для этого - когда считываются данные - почти все методы ввода, которые создают кадры данных, имеют `stringsAsFactors` параметровAsFactors, которые могут быть установлены в `FALSE` .

Если данные уже созданы, столбцы факторов могут быть преобразованы в столбцы символов, как показано ниже.

```
bob <- data.frame(jobs = c("scientist", "analyst"),
                  pay  = c(160000, 100000), age = c(30, 25))
str(bob)
```

```
'data.frame':    2 obs. of  3 variables:
 $ jobs: Factor w/ 2 levels "analyst","scientist": 2 1
 $ pay : num  160000 100000
 $ age : num  30 25
```

```
# Convert *all columns* to character
bob[] <- lapply(bob, as.character)
str(bob)
```

```
    'data.frame':    2 obs. of  3 variables:
 $ jobs: chr  "scientist" "analyst"
 $ pay : chr  "160000" "1e+05"
 $ age : chr  "30" "25"
```

```
# Convert only factor columns to character
bob[] <- lapply(bob, function(x) {
    if is.factor(x) x <- as.character(x)
    return(x)
})
```

Подмножество строк по значениям столбцов

Встроенные функции могут подмножать `rows` с `columns` , отвечающими условиям.

```
df <- data.frame(item = c(1:10),
                 price_Elasticity = c(-0.57667, 0.03205, -0.04904, 0.10342, 0.04029,
                                      0.0742, 0.1669, 0.0313, 0.22204, 0.06158),
                 total_Margin = c(-145062, 98671, 20576, -56382, 207623, 43463, 1235,
                                  34521, 146553, -74516))
```

Чтобы найти `rows` с `price_Elasticity > 0` :

```
df[df$price_Elasticity > 0, ]
```

```
   item price_Elasticity total_Margin
2     2          0.03205        98671
4     4          0.10342       -56382
5     5          0.04029       207623
6     6          0.07420        43463
7     7          0.16690         1235
8     8          0.03130        34521
9     9          0.22204       146553
10   10          0.06158       -74516
```

подмножество на основе `price_Elasticity > 0` и `total_Margin > 0` :

```
df[df$price_Elasticity > 0 & df$total_Margin > 0, ]
```

```
  item price_Elasticity total_Margin
2    2          0.03205        98671
5    5          0.04029       207623
6    6          0.07420        43463
7    7          0.16690         1235
8    8          0.03130        34521
9    9          0.22204       146553
```

Прочитайте Кадры данных онлайн: https://riptutorial.com/ru/r/topic/438/кадры-данных

глава 60: Класс Date

замечания

похожие темы

- Дата и время

Смешанные заметки

- `Date` : Сохраняет время как количество дней с момента UNIX в `1970-01-01` . с отрицательными значениями для более ранних дат.
- Он представлен как целое число (однако он не применяется во внутреннем представлении)
- Они всегда печатаются по правилам текущего григорианского календаря, хотя календарь давно не использовался.
- Он не отслеживает временные интервалы, поэтому его не следует использовать для усечения времени из объектов `POSIXct` или `POSIXlt` .
- `sys.Date()` возвращает объект класса `Date`

Другие заметки

- lumridate 's `ymd` , `mdy` и т. д. являются альтернативами `as.Date` которые также анализируют класс Date; см. Синхронизация дат и времени от строк с помощью lubridate .
- data.table экспериментальный класс IDATE «ы получают из и в основном взаимозаменяемыми с датой, но хранится в виде целого числа , а не в два раза.

Examples

Форматирование дат

Для форматирования `Dates` мы используем `format(date, format="%Y-%m-%d")` с помощью `POSIXct` (заданного из `as.POSIXct()`) или `POSIXlt` (данный из `as.POSIXlt()`)

```
d = as.Date("2016-07-21") # Current Date Time Stamp

format(d,"%a")            # Abbreviated Weekday
## [1] "Thu"
```

```
format(d,"%A")            # Full Weekday
## [1] "Thursday"

format(d,"%b")            # Abbreviated Month
## [1] "Jul"

format(d,"%B")            # Full Month
## [1] "July"

format(d,"%m")            # 00-12 Month Format
## [1] "07"

format(d,"%d")            # 00-31 Day Format
## [1] "21"

format(d,"%e")            # 0-31 Day Format
## [1] "21"

format(d,"%y")            # 00-99 Year
## [1] "16"

format(d,"%Y")            # Year with Century
## [1] "2016"
```

Подробнее см. `?strptime` .

Даты

Чтобы принудить переменную к дате использовать `as.Date()` .

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "Date"
```

Функция `as.Date()` позволяет вам предоставить аргумент формата. По умолчанию используется значение `%Y-%m-%d` , которое равно Year-month-day.

```
> as.Date("23-8-2016", format="%d-%m-%Y") # To read in an European-style date
[1] "2016-08-23"
```

Строка формата может быть помещена либо в пару одинарных кавычек, либо в двойные кавычки. Даты обычно выражаются в различных формах, таких как: `"dm-yy"` или `"dm-YYYY"` или `"md-yy"` или `"md-YYYY"` или `"YYYY-md"` или `"YYYY-dm"` . Эти форматы также можно выразить, заменив `"-"` на `"/"` . Более того, даты также выражаются в формах, например: «6 ноября 1986 года» или «6 ноября 1986 года» или «6 ноября 1986 года» или «6 ноября 1986 года» и т. Д. Функция **as.Date ()** принимает все такие символьные строки, и когда мы **указываем** соответствующий формат строки, она всегда выводит дату в форме `"YYYY-md"` .

Предположим, что у нас есть строка даты `"9-6-1962"` в формате `"%d-%m-%Y"` .

```
#
# It tries to interprets the string as YYYY-m-d
#
> as.Date("9-6-1962")
[1] "0009-06-19"       #interprets as "%Y-%m-%d"
>
as.Date("9/6/1962")
[1] "0009-06-19"       #again interprets as "%Y-%m-%d"
>
# It has no problem in understanding, if the date is in form  YYYY-m-d or YYYY/m/d
#
> as.Date("1962-6-9")
[1] "1962-06-09"        # no problem
> as.Date("1962/6/9")
[1] "1962-06-09"        # no problem
>
```

Указав правильный формат входной строки, мы можем получить желаемые результаты. Мы используем следующие коды для указания форматов функции **as.Date ()** .

Формат кода	Имея в виду
`%d`	день
`%m`	месяц
`%y`	год в 2-значных цифрах
`%Y`	год в 4-значных цифрах
`%b`	сокращенный месяц в 3 символах
`%B`	полное название месяца

Рассмотрим следующий пример, определяющий параметр **формата** :

```
> as.Date("9-6-1962",format="%d-%m-%Y")
[1] "1962-06-09"
>
```

Формат имя параметра можно опустить.

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

Несколько раз имена месяцев, сокращенных до первых трех символов, используются при написании дат. В этом случае мы используем спецификатор формата `%b` .

```
> as.Date("6Nov1962","%d%b%Y")
[1] "1962-11-06"
```

```
>
```

Обратите внимание, что между членами в строке даты нет ни `'-'` ни `'/'` или белых пробелов. Строка формата должна точно соответствовать этой входной строке. Рассмотрим следующий пример:

```
> as.Date("6 Nov, 1962","%d %b, %Y")
[1] "1962-11-06"
>
```

Обратите внимание, что в строке даты есть запятая, а значит, и запятая в спецификации формата. Если запятая в строке формата опущена, это приводит к `NA` . Пример использования спецификатора формата `%B` выглядит следующим образом:

```
> as.Date("October 12, 2016", "%B %d, %Y")
[1] "2016-10-12"
>
> as.Date("12 October, 2016", "%d %B, %Y")
[1] "2016-10-12"
>
```

Формат `%y` является системным и, следовательно, следует использовать с осторожностью. Другими параметрами, используемыми с этой функцией, являются **начало** и **tz** (часовой пояс).

Разбор строк на объекты даты

R содержит класс Date, который создается с помощью `as.Date()` , который берет строку или вектор строк, а если дата не соответствует формату `YYYY-MM-DD` ISO 8601 `YYYY-MM-DD` , строка форматирования `strptime` tokens ,

```
as.Date('2016-08-01')    # in ISO format, so does not require formatting string
## [1] "2016-08-01"

as.Date('05/23/16', format = '%m/%d/%y')
## [1] "2016-05-23"

as.Date('March 23rd, 2016', '%B %drd, %Y')    # add separators and literals to format
## [1] "2016-03-23"

as.Date('  2016-08-01  foo')    # leading whitespace and all trailing characters are ignored
## [1] "2016-08-01"

as.Date(c('2016-01-01', '2016-01-02'))
# [1] "2016-01-01" "2016-01-02"
```

Прочитайте Класс Date онлайн: https://riptutorial.com/ru/r/topic/9015/класс-date

глава 61: Класс символов

Вступление

Символы - это то, что другие языки называют «векторами строк».

замечания

похожие темы

Узоры

- Регулярные выражения (регулярное выражение)
- Согласование и замена шаблонов
- Функция strsplit

Вход и выход

- Чтение и запись строк

Examples

принуждение

Чтобы проверить, является ли значение символом, используйте `is.character()` . Чтобы принудить переменную к символу использовать `as.character()` .

```
x <- "The quick brown fox jumps over the lazy dog"
class(x)
[1] "character"
is.character(x)
[1] TRUE
```

Обратите внимание, что числовые символы могут быть принудительно применены к символам, но попытка принудительного принуждения символа к числовому может привести к `NA` .

```
as.numeric("2")
[1] 2
as.numeric("fox")
[1] NA
Warning message:
NAs introduced by coercion
```

Прочитайте Класс символов онлайн: https://riptutorial.com/ru/r/topic/9017/класс-символов

глава 62: Классы

Вступление

Класс объекта данных определяет, какие функции будут обрабатывать его содержимое. `class` атрибут является символьным вектором, а объекты могут иметь нуль, один или несколько классов. Если атрибута класса нет, все равно будет неявный класс, определяемый `mode` объекта. Класс может быть проверен с помощью `class` функций, и его можно установить или изменить с помощью `class<-` function. Система классов S3 была создана в начале истории S. Более сложная система класса S4 была установлена позже

замечания

Существует несколько функций для проверки «типа» объекта. Наиболее полезной такой функцией является `class` , хотя иногда необходимо изучить `mode` объекта. Поскольку мы обсуждаем «типы», можно подумать, что `typeof` будет полезен, но в целом результат из `mode` будет более полезен, потому что объекты, у которых нет явного атрибута «class», будут иметь функцию диспетчеризации, определяемую определяемым «неявным классом» по их режиму.

Examples

векторы

Самая простая структура данных, доступная в R, является вектором. Вы можете создавать векторы числовых значений, логических значений и символьных строк с помощью функции `c()` . Например:

```
c(1, 2, 3)
## [1] 1 2 3
c(TRUE, TRUE, FALSE)
## [1]  TRUE  TRUE FALSE
c("a", "b", "c")
## [1] "a" "b" "c"
```

Вы также можете присоединиться к векторам, используя функцию `c()` .

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
## [1] 1 2 5 3 4 6
```

Более подробное описание того, как создавать векторы, можно найти в теме *«Создание*

векторов»

Осмотреть классы

Каждому объекту в R присваивается класс. Вы можете использовать `class()` чтобы найти класс объекта и `str()` чтобы увидеть его структуру, включая классы, которые она содержит. Например:

```
class(iris)
[1] "data.frame"

str(iris)
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 ...

class(iris$Species)
[1] "factor"
```

Мы видим, что iris имеет класс `data.frame` и использование `str()` позволяет нам исследовать данные внутри. Переменная Species в кадре данных радужки имеет фактор класса, в отличие от других переменных, которые относятся к классу числовых. Функция `str()` также предоставляет длину переменных и показывает первую пару наблюдений, а функция `class()` предоставляет только класс объекта.

Векторы и списки

Данные в R хранятся в векторах. Типичным вектором является последовательность значений, все из которых имеют один и тот же режим хранения (например, векторы символов, числовые векторы). См. « `?atomic` для получения подробных сведений о атомных неявных классах и их соответствующих режимах хранения: `"logical"`, `"integer"`, `"numeric"` `(synonym "double")`, `"complex"`, `"character"` и `"raw"` . Многие классы - это просто атомный вектор с атрибутом `class` сверху:

```
x <- 1826
class(x) <- "Date"
x
# [1] "1975-01-01"
 x <- as.Date("1970-01-01")
 class(x)
#[1] "Date"
 is(x,"Date")
#[1] TRUE
 is(x,"integer")
#[1] FALSE
 is(x,"numeric")
#[1] FALSE
  mode(x)
```

```
#[1] "numeric"
```

Списки представляют собой особый тип вектора, в котором каждый элемент может быть любым, даже другим списком, поэтому термин R для списков: «рекурсивные векторы»:

```
mylist <- list( A = c(5,6,7,8), B = letters[1:10], CC = list( 5, "Z") )
```

Списки имеют два очень важных применения:

- Поскольку функции могут возвращать только одно значение, обычно возвращать сложные результаты в список:

```
f <- function(x) list(xplus = x + 10, xsq = x^2)

f(7)
# $xplus
# [1] 17
#
# $xsq
# [1] 49
```

- Списки также являются базовым классом для фреймов данных . Под капотом кадр данных представляет собой список векторов, имеющих всю длину:

```
L <- list(x = 1:2, y = c("A","B"))
DF <- data.frame(L)
DF
#   x y
# 1 1 A
# 2 2 B
is.list(DF)
# [1] TRUE
```

Другим классом рекурсивных векторов являются R-выражения, которые являются «языковыми» объектами

Прочитайте Классы онлайн: https://riptutorial.com/ru/r/topic/3563/классы

глава 63: Классы времени (POSIXct и POSIXlt)

Вступление

R включает два класса времени - POSIXct и POSIXlt - см. `?DateTimeClasses` .

замечания

Ловушки

С помощью POSIXct в полночь будет отображаться только дата и часовой пояс, хотя все время сохраняется.

похожие темы

- Дата и время

Специализированные пакеты

- lubridate

Examples

Форматирование и печать объектов даты и времени

```
# test date-time object
options(digits.secs = 3)
d = as.POSIXct("2016-08-30 14:18:30.58", tz = "UTC")

format(d,"%S")  # 00-61 Second as integer
## [1] "30"

format(d,"%OS") # 00-60.99… Second as fractional
## [1] "30.579"

format(d,"%M")  # 00-59 Minute
## [1] "18"

format(d,"%H")  # 00-23 Hours
## [1] "14"
```

```
format(d,"%I")  # 01-12 Hours
## [1] "02"

format(d,"%p")  # AM/PM Indicator
## [1] "PM"

format(d,"%z")  # Signed offset
## [1] "+0000"

format(d,"%Z")  # Time Zone Abbreviation
## [1] "UTC"
```

Подробнее о строках формата см. Здесь `?strptime` , а также другие форматы.

Разбор строк в объектах с датой

Функции для синтаксического анализа строки в POSIXct и POSIXlt принимают аналогичные параметры и возвращают аналогичный результат, но существуют различия в том, как хранится дата-время; см. «Примечания».

```
as.POSIXct("11:38",                         # time string
           format = "%H:%M")                # formatting string
## [1] "2016-07-21 11:38:00 CDT"
strptime("11:38",                           # identical, but makes a POSIXlt object
         format = "%H:%M")
## [1] "2016-07-21 11:38:00 CDT"

as.POSIXct("11 AM",
           format = "%I %p")
## [1] "2016-07-21 11:00:00 CDT"
```

Обратите внимание, что дата и часовой пояс вменяются.

```
as.POSIXct("11:38:22",                  # time string without timezone
           format = "%H:%M:%S",
           tz = "America/New_York")     # set time zone
## [1] "2016-07-21 11:38:22 EDT"

as.POSIXct("2016-07-21 00:00:00",
           format = "%F %T")            # shortcut tokens for "%Y-%m-%d" and "%H:%M:%S"
```

Подробнее о строках формата смотрите здесь. `?strptime` .

Заметки

Отсутствующие элементы

- Если элемент даты не указан, то используется текущая дата.

- Если элемент времени не указан, то используется с полуночи, то есть 0 с.
- Если часовой пояс не указан ни в строке, ни в параметре `tz` , используется местный часовой пояс.

Часовые пояса

- Принимаемые значения `tz` зависят от местоположения.
  - `CST` предоставляется `"CST6CDT"` или `"America/Chicago"`
- Для поддерживаемых мест и часовых поясов используйте:
  - В R: `OlsonNames()`
  - Кроме того, попробуйте в R: `system("cat $R_HOME/share/zoneinfo/zone.tab")`
- Эти местоположения предоставляются полномочным органом по присвоению номеров в Интернете (IANA)
  - Список часовых поясов базы данных tz (Википедия)
  - Данные IANA TZ (2016e)

Арифметика по времени

Чтобы добавить / вычесть время, используйте POSIXct, поскольку он хранит время в секундах

```
## adding/subtracting times - 60 seconds
as.POSIXct("2016-01-01") + 60
# [1] "2016-01-01 00:01:00 AEDT"

## adding 3 hours, 14 minutes, 15 seconds
as.POSIXct("2016-01-01") + ( (3 * 60 * 60) + (14 * 60) + 15)
# [1] "2016-01-01 03:14:15 AEDT"
```

Более формально, `as.difftime` можно использовать для указания периодов времени для добавления к дате или объекту datetime. Например:

```
as.POSIXct("2016-01-01")         +
  as.difftime(3,  units="hours") +
  as.difftime(14, units="mins")  +
  as.difftime(15, units="secs")
# [1] "2016-01-01 03:14:15 AEDT"
```

Чтобы найти разницу между датами / временем, используйте `difftime()` для разницы в секундах, минутах, часах, днях или неделях.

```
# using POSIXct objects
difftime(
  as.POSIXct("2016-01-01 12:00:00"),
  as.POSIXct("2016-01-01 11:59:59"),
  unit = "secs")
# Time difference of 1 secs
```

Чтобы генерировать последовательности дат-времени, используйте `seq.POSIXt()` или просто `seq`.

Прочитайте Классы времени (POSIXct и POSIXlt) онлайн: https://riptutorial.com/ru/r/topic/9027/классы-времени--posixct-и-posixlt-

глава 64: Кодировка длины

замечания

Прогон представляет собой последовательную последовательность повторяющихся значений или наблюдений. Для повторных значений R-кодировка «длина строки» кратко описывает вектор в терминах его прогонов. Рассматривать:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

У нас есть пробег длиной 1 с; затем три пробега длиной 2 с; затем пробег длиной 3 с; и так далее. R-образная кодировка фиксирует все длины и значения пробегов вектора.

расширения

Прогон также может ссылаться на последовательные наблюдения в табличных данных. Хотя R не имеет естественного способа их кодирования, их можно обрабатывать с помощью `rleid` из пакета data.table (в настоящее время это тупиковая ссылка) .

Examples

Кодировка длины пробега с `rle`

Кодировка длины выполнения фиксирует длины прогонов последовательных элементов в векторе. Рассмотрим пример вектора:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

Функция `rle` извлекает каждый пробег и его длину:

```
r <- rle(dat)
r
# Run Length Encoding
#   lengths: int [1:6] 1 3 1 1 2 2
#   values : num [1:6] 1 2 3 1 4 1
```

Значения для каждого прогона записываются в `r$values` :

```
r$values
# [1] 1 2 3 1 4 1
```

Это фиксирует, что мы сначала увидели пробег в 1, затем пробег 2, затем пробег в 3, затем пробег в 1 и т. Д.

Длины каждого прогона захватываются по `r$lengths` :

```
r$lengths
# [1] 1 3 1 1 2 2
```

Мы видим, что начальный пробег 1 был длиной 1, пробег 2, следующий за ним, был длиной 3 и т. Д.

Идентификация и группировка пробегами в базе R

Можно захотеть сгруппировать свои данные с помощью пробегов переменной и выполнить какой-то анализ. Рассмотрим следующий простой набор данных:

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1 1 1
# 2 1 2
# 3 2 3
# 4 2 4
# 5 2 5
# 6 1 6
```

Переменная `x` имеет три пробега: пробег длиной 2 со значением 1, пробег 3 со значением 2 и пробег длиной 1 со значением 1. Мы могли бы захотеть вычислить среднее значение переменной `y` в каждом из пробеги переменной `x` (эти средние значения равны 1,5, 4 и 6).

В базовом R, мы сначала вычислить по длине прогона кодирование `x` переменной с помощью `rle` :

```
(r <- rle(dat$x))
# Run Length Encoding
#   lengths: int [1:3] 2 3 1
#   values : num [1:3] 1 2 1
```

Следующим шагом будет вычисление номера прогона каждой строки нашего набора данных. Мы знаем, что общее количество прогонов - `length(r$lengths)` , а длина каждого пробега равна `r$lengths` , поэтому мы можем вычислить номер пробега каждого из наших прогонов с `rep` :

```
(run.id <- rep(seq_along(r$lengths), r$lengths))
# [1] 1 1 2 2 2 3
```

Теперь мы можем использовать `tapply` для вычисления среднего значения `y` для каждого прогона путем группировки на идентификаторе запуска:

```
data.frame(x=r$values, meanY=tapply(dat$y, run.id, mean))
#   x meanY
# 1 1   1.5
# 2 2   4.0
```

```
# 3 1   6.0
```

Идентификация и группировка с помощью прогонов в data.table

Пакет data.table обеспечивает удобный способ группировки по прогонам в данных. Рассмотрим следующие примеры данных:

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#    x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

Переменная `x` имеет три пробега: пробег длиной 2 со значением 1, пробег 3 со значением 2 и пробег длиной 1 со значением 1. Мы могли бы захотеть вычислить среднее значение переменной `y` в каждом из пробеги переменной x (эти средние значения равны 1,5, 4 и 6).

Функция data.table `rleid` предоставляет идентификатор, указывающий идентификатор запуска каждого элемента вектора:

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

Затем можно легко группировать этот идентификатор запуска и суммировать данные `y` :

```
DT[,mean(y),by=.(x, rleid(x))]
#    x rleid  V1
# 1: 1     1 1.5
# 2: 2     2 4.0
# 3: 1     3 6.0
```

Кодировка длины для сжатия и распаковки векторов

Длинные векторы с длинными тиражами одного и того же значения могут быть значительно сжаты, сохраняя их в кодировке их длины (значение каждого прогона и количество повторений этого значения). В качестве примера рассмотрим вектор длиной 10 миллионов с огромным числом 1 и только небольшое число 0:

```
set.seed(144)
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)
table(dat)
#       0       1
#     103 9999897
```

Для хранения 10 миллионов записей потребуется значительное пространство, но вместо

этого мы можем создать кадр данных с кодировкой этого вектора длины:

```
rle.df <- with(rle(dat), data.frame(values, lengths))
dim(rle.df)
# [1] 207   2
head(rle.df)
#   values lengths
# 1      1   52818
# 2      0       1
# 3      1  219329
# 4      0       1
# 5      1  318306
# 6      0       1
```

Из кодирования длины пробега мы видим, что первые 52 818 значений в векторе равны 1, а затем один 0, за которым следуют 219,329 последовательных 1, а затем 0 и т. Д. Кодировка длины прогона содержит только 207 записей, что требует хранения только 414 значений вместо 10 миллионов значений. Поскольку `rle.df` - это кадр данных, его можно сохранить с помощью стандартных функций, таких как `write.csv` .

Декомпрессия вектора в кодировании длины строки может быть выполнена двумя способами. Первый метод просто вызвать `rep` , передавая `values` элемента кодирования длин серий в качестве первого аргумента и `lengths` элемента кодирования длин серий в качестве второго аргумента:

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

Мы можем подтвердить, что наши распакованные данные идентичны нашим исходным данным:

```
identical(decompressed, dat)
# [1] TRUE
```

Второй способ заключается в использовании R встроенного в `inverse.rle` функции на `rle` объект, например:

```
rle.obj <- rle(dat)                              # create a rle object here
class(rle.obj)
# [1] "rle"

dat.inv <- inverse.rle(rle.obj)                 # apply the inverse.rle on the rle object
```

Мы можем еще раз подтвердить , что это производит именно оригинальный `dat` :

```
identical(dat.inv, dat)
# [1] TRUE
```

Прочитайте Кодировка длины онлайн: https://riptutorial.com/ru/r/topic/1133/кодировка-длины

глава 65: Комбинаторика

Examples

Перечисление комбинаций заданной длины

Без замены

С помощью `combn` каждый вектор появляется в столбце:

```
combn(LETTERS, 3)

# Showing only first 10.
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"  "A"
[2,] "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"  "B"
[3,] "C"  "D"  "E"  "F"  "G"  "H"  "I"  "J"  "K"  "L"
```

С заменой

С помощью `expand.grid` каждый вектор появляется в строке:

```
expand.grid(LETTERS, LETTERS, LETTERS)
# or
do.call(expand.grid, rep(list(LETTERS), 3))

# Showing only first 10.
   Var1 Var2 Var3
1     A    A    A
2     B    A    A
3     C    A    A
4     D    A    A
5     E    A    A
6     F    A    A
7     G    A    A
8     H    A    A
9     I    A    A
10    J    A    A
```

Для частного случая пар можно использовать `outer` , поместив каждый вектор в ячейку:

```
# FUN here is used as a function executed on each resulting pair.
# in this case it's string concatenation.
outer(LETTERS, LETTERS, FUN=paste0)

# Showing only first 10 rows and columns
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,] "AA" "AB" "AC" "AD" "AE" "AF" "AG" "AH" "AI" "AJ"
```

```
 [2,] "BA" "BB" "BC" "BD" "BE" "BF" "BG" "BH" "BI" "BJ"
 [3,] "CA" "CB" "CC" "CD" "CE" "CF" "CG" "CH" "CI" "CJ"
 [4,] "DA" "DB" "DC" "DD" "DE" "DF" "DG" "DH" "DI" "DJ"
 [5,] "EA" "EB" "EC" "ED" "EE" "EF" "EG" "EH" "EI" "EJ"
 [6,] "FA" "FB" "FC" "FD" "FE" "FF" "FG" "FH" "FI" "FJ"
 [7,] "GA" "GB" "GC" "GD" "GE" "GF" "GG" "GH" "GI" "GJ"
 [8,] "HA" "HB" "HC" "HD" "HE" "HF" "HG" "HH" "HI" "HJ"
 [9,] "IA" "IB" "IC" "ID" "IE" "IF" "IG" "IH" "II" "IJ"
[10,] "JA" "JB" "JC" "JD" "JE" "JF" "JG" "JH" "JI" "JJ"
```

Подсчет комбинаций заданной длины

Без замены

```
choose(length(LETTERS), 5)
[1] 65780
```

С заменой

```
length(letters)^5
[1] 11881376
```

Прочитайте Комбинаторика онлайн: https://riptutorial.com/ru/r/topic/5836/комбинаторика

глава 66: Линейные модели (регрессия)

Синтаксис

- lm (формула, данные, подмножество, весы, na.action, method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE, контрасты = NULL, смещение, .. .)

параметры

параметр	Имея в виду
формула	формула в обозначениях *Уилкинсона-Роджерса* ; `response ~ ...` where `...` содержит термины, соответствующие переменным в среде или в кадре данных, заданном аргументом `data`
данные	кадр данных, содержащий переменные ответа и предиктора
подмножество	вектор, определяющий подмножество наблюдений, которые будут использоваться: может быть выражен как логический оператор с точки зрения переменных в `data`
веса	аналитические веса (см. раздел « *Веса* выше)
na.action	как обрабатывать отсутствующие значения (`NA`): см. `?na.action`
метод	как выполнить фитинг. Возможны только `"qr"` или `"model.frame"` (последний возвращает модельный кадр без подгонки модели, идентичной заданной `model=TRUE`)
модель	хранить ли модельную рамку в установленном объекте
Икс	хранить ли матрицу модели в приспособленном объекте
Y	следует ли хранить ответ модели в установленном объекте
ор	хранить ли QR-декомпозицию в установленном объекте
singular.ok	разрешить ли *сингулярные подходы* , модели с коллинеарными предикторами (подмножество коэффициентов будет автоматически установлено на `NA` в этом случае
контрасты	список контрастов, которые будут использоваться для конкретных факторов в модели; см `contrasts.arg` аргумент `?model.matrix.default` . Контрасты также можно установить с помощью `options()` (см.

параметр	Имея в виду
	Аргумент `contrasts`) или путем назначения `contrast` атрибутов фактора (см. `?contrasts`)
смещение	используется для указания *априори* известного компонента в модели. Может также указываться как часть формулы. См. `?model.offset`
...	дополнительные аргументы, которые должны быть переданы функциям `lm.fit()` нижнего уровня (`lm.fit()` или `lm.wfit()`)

Examples

Линейная регрессия по набору данных mtcars

Встроенный кадр данных mtcars содержит информацию о 32 машинах, включая их вес, топливную экономичность (в мили на галлон), скорость и т. Д. (Чтобы узнать больше о наборе данных, используйте `help(mtcars)`).

Если нас интересует взаимосвязь между топливной экономичностью (`mpg`) и весом (`wt`), Мы можем начать строить эти переменные с:

```
plot(mpg ~ wt, data = mtcars, col=2)
```

Графики показывают (линейное) соотношение !. Тогда, если мы хотим выполнить линейную регрессию для определения коэффициентов линейной модели, мы будем использовать `lm` функцию:

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Здесь `~` означает «объяснено», поэтому формула `mpg ~ wt` означает, что мы прогнозируем mpg, как объясняется wt. Самый полезный способ просмотра результатов:

```
summary(fit)
```

Что дает результат:

```
Call:
lm(formula = mpg ~ wt, data = mtcars)

Residuals:
    Min      1Q  Median      3Q     Max
-4.5432 -2.3647 -0.1252  1.4096  6.8727

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  37.2851     1.8776  19.858  < 2e-16 ***
```

```
wt           -5.3445     0.5591  -9.559 1.29e-10 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared:  0.7528,    Adjusted R-squared:  0.7446
F-statistic: 91.38 on 1 and 30 DF,  p-value: 1.294e-10
```

Это дает информацию о:

- оцененный наклон каждого коэффициента (`wt` и у-перехват), который предполагает наилучшее предсказание mpg, составляет `37.2851 + (-5.3445) * wt`
- P-значение каждого коэффициента, что предполагает, что перехват и вес, вероятно, не из-за случайности
- Общие оценки подгонки, такие как R ^ 2 и скорректированные R ^ 2, которые показывают, какая часть изменения в `mpg` объясняется моделью

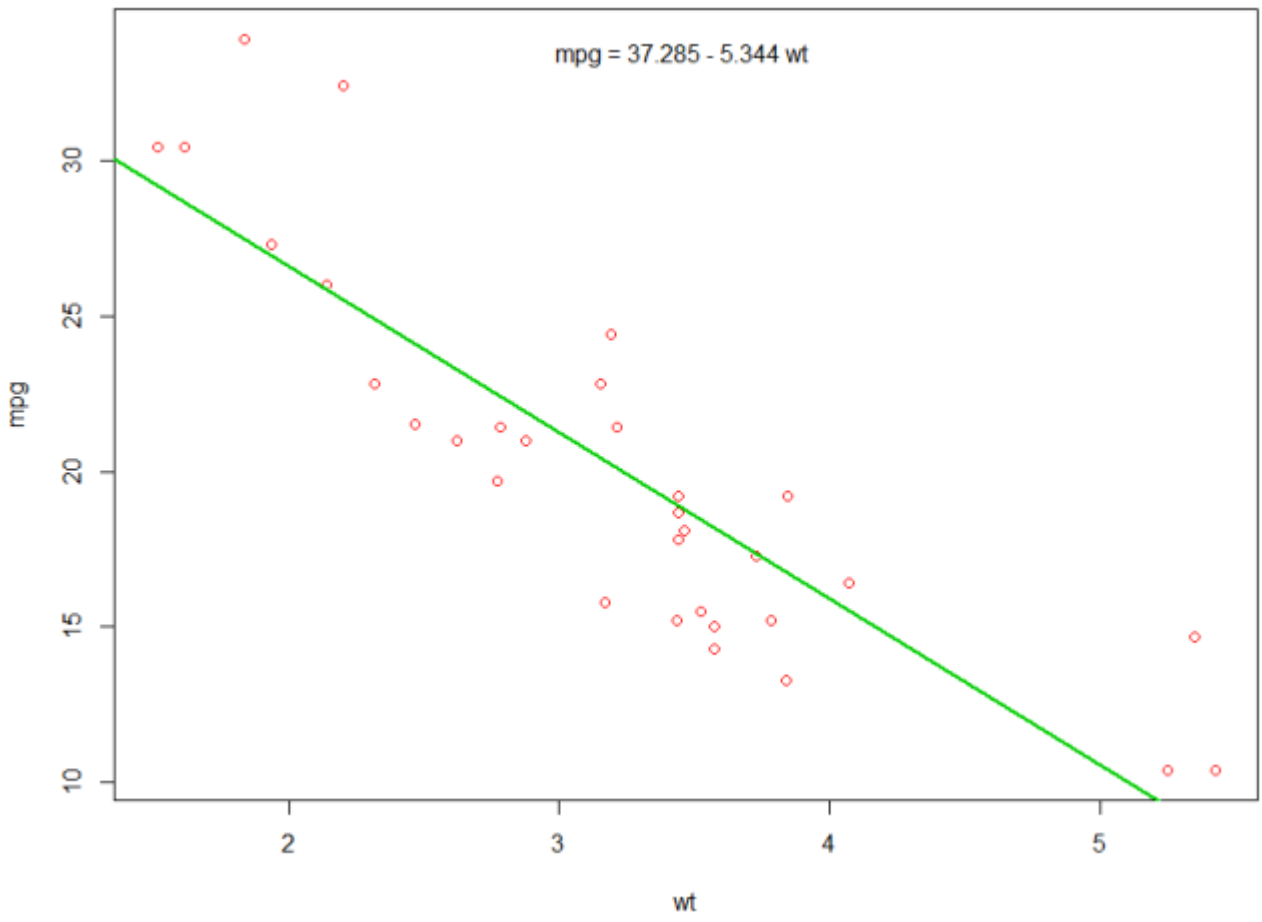
Мы могли бы добавить строку к нашему первому сюжету, чтобы показать предсказанный `mpg` :

```
abline(fit,col=3,lwd=2)
```

Также можно добавить уравнение к этому графику. Сначала получим коэффициенты с коэффициентом `coef` . Затем, используя `paste0` мы `paste0` коэффициенты с соответствующими переменными и `+/-` , чтобы построить уравнение. Наконец, мы добавляем его к сюжету с помощью `mtext` :

```
bs <- round(coef(fit), 3)
lmlab <- paste0("mpg = ", bs[1],
             ifelse(sign(bs[2])==1, " + ", " - "), abs(bs[2]), " wt ")
mtext(lmlab, 3, line=-2)
```

Результат:



Построение регрессии (базы)

Продолжая пример `mtcars` , вот простой способ создать график вашей линейной регрессии, который потенциально подходит для публикации.

Сначала применим линейную модель и

```
fit <- lm(mpg ~ wt, data = mtcars)
```

Затем постройте две интересующие переменные и добавьте линию регрессии в пределах области определения:

```
plot(mtcars$wt,mtcars$mpg,pch=18, xlab = 'wt',ylab = 'mpg')
lines(c(min(mtcars$wt),max(mtcars$wt)),
as.numeric(predict(fit, data.frame(wt=c(min(mtcars$wt),max(mtcars$wt))))))
```

Почти готово! Последний шаг - добавить к графику, уравнение регрессии, rsquare, а также коэффициент корреляции. Это делается с использованием `vector` функции:

```
rp = vector('expression',3)
rp[1] = substitute(expression(italic(y) == MYOTHERVALUE3 + MYOTHERVALUE4 %*% x),
          list(MYOTHERVALUE3 = format(fit$coefficients[1], digits = 2),
                        MYOTHERVALUE4 = format(fit$coefficients[2], digits = 2)))[2]
rp[2] = substitute(expression(italic(R)^2 == MYVALUE),
             list(MYVALUE = format(summary(fit)$adj.r.squared,dig=3)))[2]
```

```
rp[3] = substitute(expression(Pearson-R == MYOTHERVALUE2),
             list(MYOTHERVALUE2 = format(cor(mtcars$wt,mtcars$mpg), digits = 2)))[2]

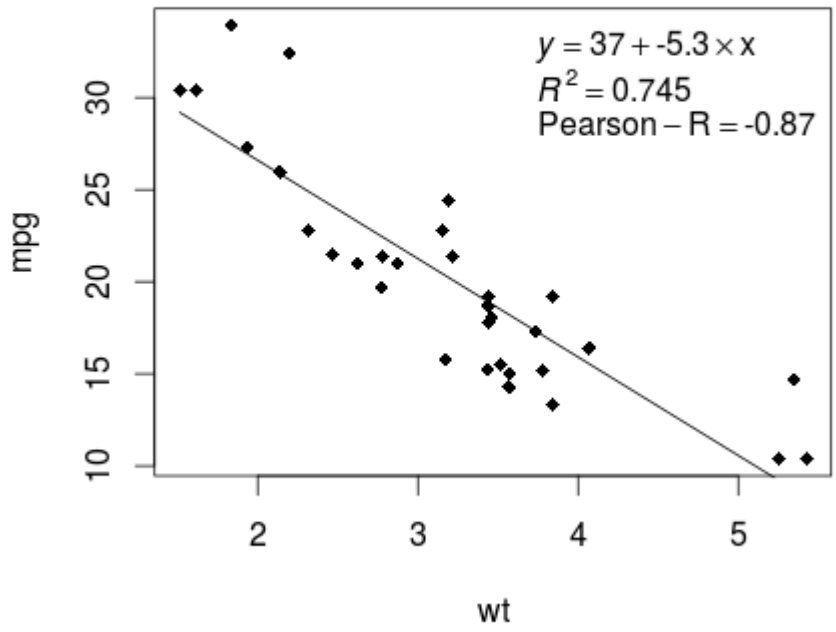
legend("topright", legend = rp, bty = 'n')
```

Обратите внимание, что вы можете добавить любые другие параметры, такие как RMSE, путем адаптации векторной функции. Представьте, что вам нужна легенда с 10 элементами. Векторное определение будет следующим:

```
rp = vector('expression',10)
```

и вам нужно будет определить `r[1]` .... to `r[10]`

Вот результат:



утяжеление

Иногда мы хотим, чтобы модель придавала больший вес некоторым точкам данных или примерам, чем другие. Это возможно, указав вес для входных данных, изучая модель. Обычно существуют два типа сценариев, в которых мы можем использовать неравномерные веса над примерами:

- Аналитические весы: отражают различные уровни точности различных наблюдений. Например, если анализировать данные, где каждое наблюдение является средним результатом географической области, аналитический вес пропорционален обратному оценочной дисперсии. Полезно при работе со средними данными, предоставляя

пропорциональный вес с учетом количества наблюдений. Источник

- Весы выборки (обратные весовые коэффициенты вероятности - IPW): статистический метод для расчета статистики, стандартизованной для населения, отличного от того, в котором собирались данные. В заявке рассматриваются проекты исследований с разрозненной популяцией выборки и населением целевого вывода (целевое население). Полезно при работе с данными, у которых отсутствуют значения. Источник

Функция `lm()` выполняет аналитическое взвешивание. Для выборочных весов пакет `survey` используется для создания объекта дизайна съемки и запуска `svyglm()` . По умолчанию в пакете `survey` используются весы для выборки. (ПРИМЕЧАНИЕ: `lm()` и `svyglm()` с семейством `gaussian()` будут производить одни и те же точечные оценки, потому что они оба решаются для коэффициентов, сводя к минимуму взвешенные наименьшие квадраты. Они отличаются тем, как вычисляются стандартные ошибки.)

Данные испытаний

```
data <- structure(list(lexptot = c(9.1595012302023, 9.86330744180814,
8.92372556833205, 8.58202430280175, 10.1133857229336), progvillm = c(1L,
1L, 1L, 1L, 0L), sexhead = c(1L, 1L, 0L, 1L, 1L), agehead = c(79L,
43L, 52L, 48L, 35L), weight = c(1.04273509979248, 1.01139605045319,
1.01139605045319, 1.01139605045319, 0.76305216550827)), .Names = c("lexptot",
"progvillm", "sexhead", "agehead", "weight"), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -5L))
```

Аналитические весы

```
lm.analytic <- lm(lexptot ~ progvillm + sexhead + agehead,
                  data = data, weight = weight)
summary(lm.analytic)
```

Выход

```
Call:
lm(formula = lexptot ~ progvillm + sexhead + agehead, data = data,
    weights = weight)

Weighted Residuals:
         1          2          3          4          5
 9.249e-02  5.823e-01  0.000e+00 -6.762e-01 -1.527e-16

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.016054   1.744293   5.742    0.110
progvillm   -0.781204   1.344974  -0.581    0.665
sexhead      0.306742   1.040625   0.295    0.818
agehead     -0.005983   0.032024  -0.187    0.882

Residual standard error: 0.8971 on 1 degrees of freedom
```

```
Multiple R-squared:  0.467, Adjusted R-squared:  -1.132
F-statistic: 0.2921 on 3 and 1 DF,  p-value: 0.8386
```

Вес пробы (IPW)

```
library(survey)
data$X <- 1:nrow(data)             # Create unique id

# Build survey design object with unique id, ipw, and data.frame
des1 <- svydesign(id = ~X,  weights = ~weight, data = data)

# Run glm with survey design object
prog.lm <- svyglm(lexptot ~ progvillm + sexhead + agehead, design=des1)
```

Выход

```
Call:
svyglm(formula = lexptot ~ progvillm + sexhead + agehead, design = des1)

Survey design:
svydesign(id = ~X, weights = ~weight, data = data)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.016054   0.183942  54.452   0.0117 *
progvillm   -0.781204   0.640372  -1.220   0.4371
sexhead      0.306742   0.397089   0.772   0.5813
agehead     -0.005983   0.014747  -0.406   0.7546
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 0.2078647)

Number of Fisher Scoring iterations: 2
```

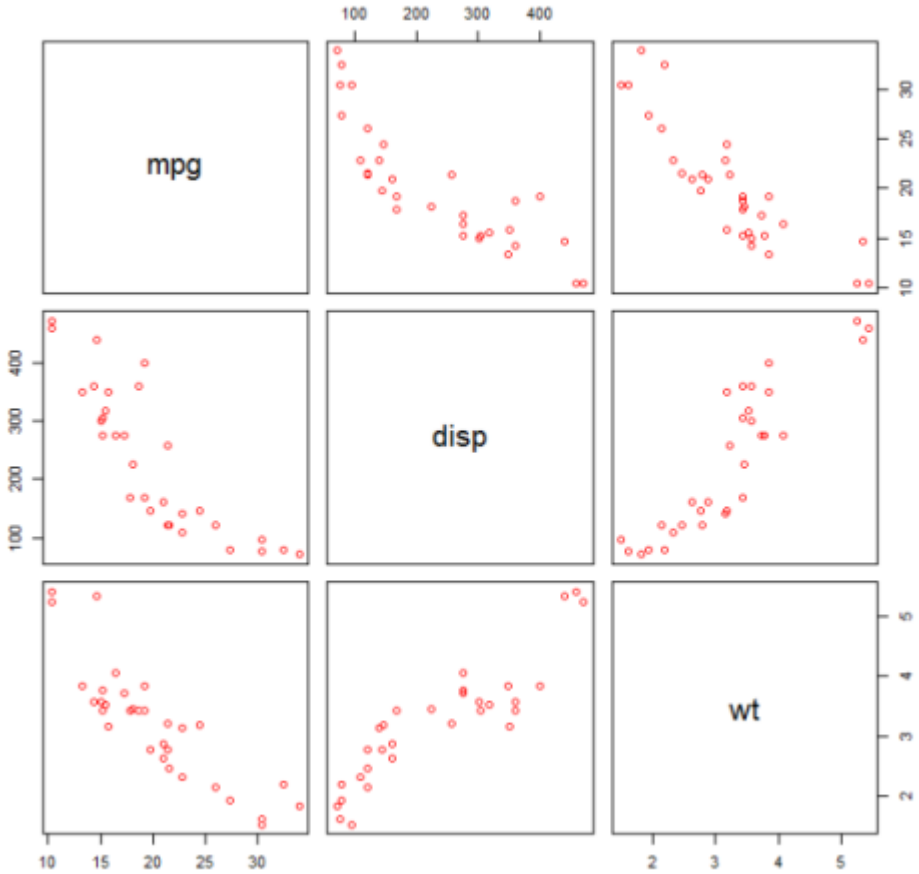
Проверка нелинейности с полиномиальной регрессией

Иногда при работе с линейной регрессией нам нужно проверить нелинейность данных. Один из способов сделать это - установить полиномиальную модель и проверить, соответствует ли она данным лучше, чем линейная модель. Существуют и другие причины, такие как теоретические, которые указывают на то, что они соответствуют квадратичной модели или модели более высокого порядка, поскольку считается, что отношение переменных по своей природе носит полиномиальный характер.

Построим квадратичную модель для набора данных `mtcars` . Для линейной модели см. Линейную регрессию по набору данных mtcars .

Сначала мы создаем график рассеяния переменных `mpg` (Miles / gallon), `disp` (смещение (cu.in.)) и `wt` (вес (1000 фунтов)). Связь между `mpg` и `disp` выглядит нелинейной.

```
plot(mtcars[,c("mpg","disp","wt")])
```



Линейная посадка покажет, что `disp` не имеет значения.

```
fit0 = lm(mpg ~ wt+disp, mtcars)
summary(fit0)

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept) 34.96055    2.16454  16.151 4.91e-16 ***
#wt          -3.35082    1.16413  -2.878  0.00743 **
#disp        -0.01773    0.00919  -1.929  0.06362 .
#---
#Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#Residual standard error: 2.917 on 29 degrees of freedom
#Multiple R-squared:  0.7809,    Adjusted R-squared:  0.7658
```

Затем, чтобы получить результат квадратичной модели, мы добавили `I(disp^2)` . Новая модель выглядит лучше, если смотреть на `R^2` и все переменные значительны.

```
fit1 = lm(mpg ~ wt+disp+I(disp^2), mtcars)
summary(fit1)

# Coefficients:
#               Estimate Std. Error t value Pr(>|t|)
#(Intercept) 41.4019837  2.4266906  17.061  2.5e-16 ***
#wt          -3.4179165  0.9545642  -3.581 0.001278 **
#disp        -0.0823950  0.0182460  -4.516 0.000104 ***
#I(disp^2)    0.0001277  0.0000328   3.892 0.000561 ***
#---
```

```
#Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
#Residual standard error: 2.391 on 28 degrees of freedom
#Multiple R-squared:  0.8578,    Adjusted R-squared:  0.8426
```

Поскольку у нас есть три переменные, подходящая модель представляет собой поверхность, представленную:

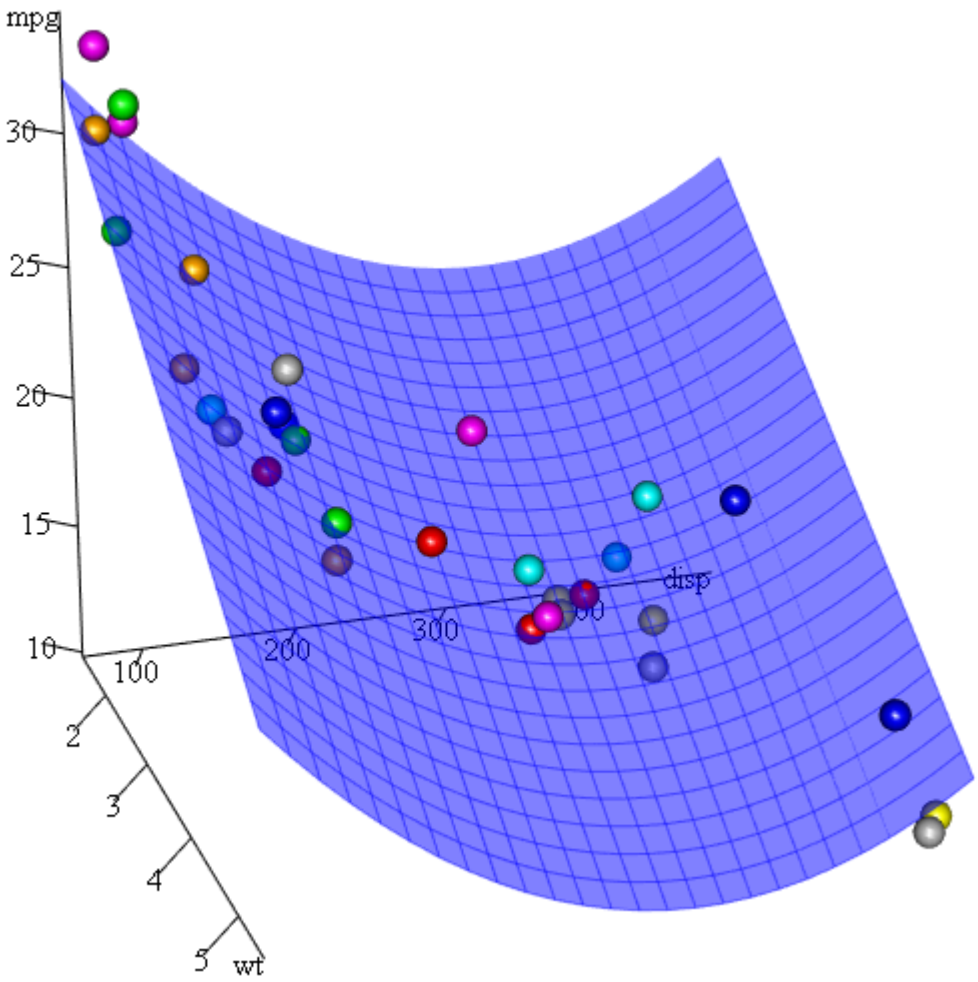
mpg = 41.4020-3.4179*wt-0.0824*disp+0.0001277*disp^2

Другой способ указать полиномиальную регрессию - использовать `poly` с параметром `raw=TRUE` , в противном случае будут рассмотрены *ортогональные многочлены* (дополнительную информацию см. В `help(ploy)`). Мы получаем тот же результат, используя:

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

Наконец, что, если нам нужно показать график предполагаемой поверхности? Ну, есть много вариантов сделать `3D` графики в `R` Здесь мы используем `Fit3d` из пакета `p3d` .

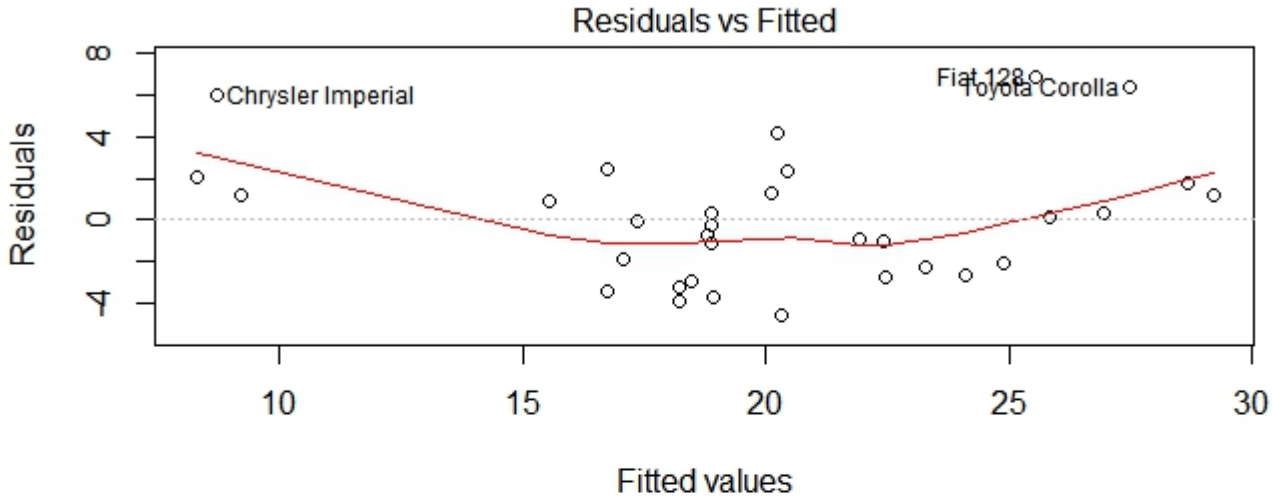
```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```

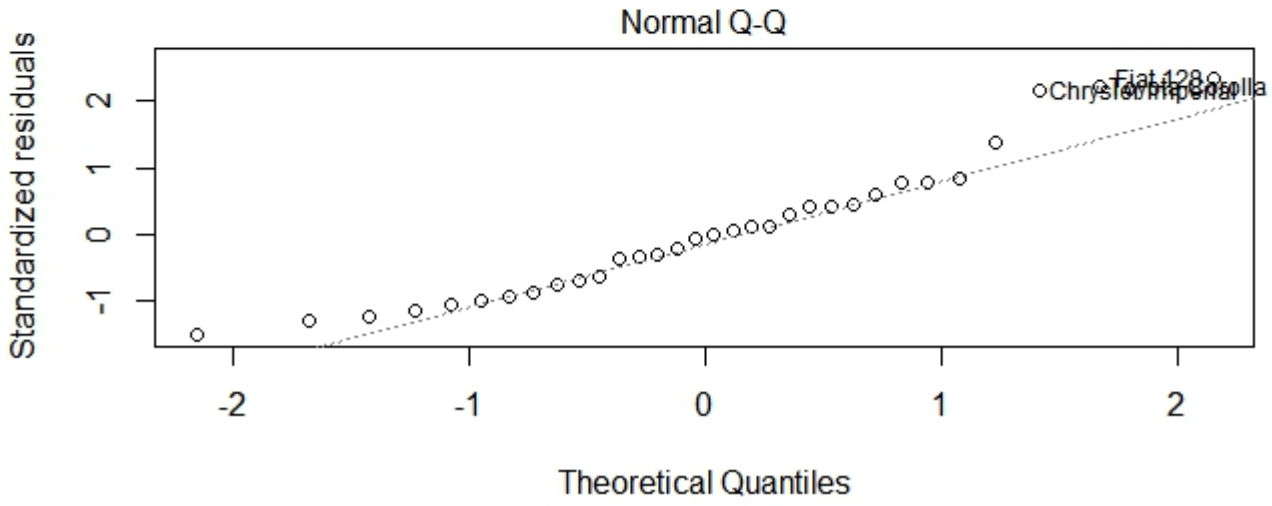


Оценка качества

После построения регрессионной модели важно проверить результат и решить, подходит ли модель и хорошо работает с данными. Это можно сделать, изучив график остатков, а также другие диагностические графики.

```
# fit the model
fit <- lm(mpg ~ wt, data = mtcars)
#
par(mfrow=c(2,1))
# plot model object
plot(fit, which =1:2)
```





Эти графики проверяют два предположения, которые были сделаны при построении модели:

1. То, что ожидаемое значение предсказанной переменной (в данном случае `mpg`) задается линейной комбинацией предсказателей (в данном случае `wt`). Мы ожидаем, что эта оценка будет беспристрастной. Таким образом, остатки должны быть сосредоточены вокруг среднего значения для всех значений предикторов. В этом случае мы видим, что остатки имеют тенденцию быть положительными на концах и отрицательными в середине, что указывает на нелинейную зависимость между переменными.
2. То, что фактическая предсказанная переменная обычно распределяется вокруг ее оценки. Таким образом, остатки должны нормально распределяться. Для нормально распределенных данных точки в нормальном графике QQ должны лежать на диагонали или близко к ней. На концах есть несколько перекосов.

Использование функции «предсказывать»

Как только модель построена, `predict` является основная функция тестирования с новыми данными. В нашем примере будет использоваться встроенный набор данных `mtcars` для регрессии миль на галлон против перемещения:

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl

Call:
lm(formula = mpg ~ disp, data = mtcars)

Coefficients:
(Intercept)         disp
   29.59985     -0.04122
```

Если бы у меня был новый источник данных со смещением, я мог бы видеть приблизительные мили за галлон.

```
set.seed(1234)
newdata <- sample(mtcars$disp, 5)
newdata
[1] 258.0  71.1  75.7 145.0 400.0

newdf <- data.frame(disp=newdata)
predict(my_mdl, newdf)
       1        2        3        4        5
18.96635 26.66946 26.47987 23.62366 13.11381
```

Наиболее важной частью процесса является создание нового фрейма данных с теми же именами столбцов, что и исходные данные. В этом случае исходные данные имели столбец с меткой `disp` , я был уверен, что вы вызовите новые данные с таким же именем.

предосторожность

Давайте рассмотрим несколько распространенных ошибок:

1. не используя data.frame в новом объекте:

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
   numeric 'envir' arg not of length one
```

2. не используя одинаковые имена в новом кадре данных:

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
Error in eval(expr, envir, enclos) : object 'disp' not found
```

точность

Чтобы проверить точность прогноза, вам понадобятся фактические значения у новых данных. В этом примере `newdf` потребуется столбец для «mpg» и «disp».

```
newdf <- data.frame(mpg=mtcars$mpg[1:10], disp=mtcars$disp[1:10])
#     mpg  disp
# 1  21.0 160.0
# 2  21.0 160.0
# 3  22.8 108.0
# 4  21.4 258.0
# 5  18.7 360.0
# 6  18.1 225.0
# 7  14.3 360.0
# 8  24.4 146.7
# 9  22.8 140.8
# 10 19.2 167.6

p <- predict(my_mdl, newdf)

#root mean square error
sqrt(mean((p - newdf$mpg)^2, na.rm=TRUE))
[1] 2.325148
```

Прочитайте Линейные модели (регрессия) онлайн: https://riptutorial.com/ru/r/topic/801/линейные-модели--регрессия-

глава 67: Логический класс

Вступление

Логический - это режим (и неявный класс) для векторов.

замечания

стенография

`TRUE` , `FALSE` и `NA` - единственные значения для логических векторов; и все три зарезервированные слова. `T` и `F` могут быть сокращены для `TRUE` и `FALSE` в чистом сеансе R, но ни `T` ни `F` не зарезервированы, поэтому присвоение значений, отличных от значений по умолчанию, этим именам может привести к затруднениям пользователей.

Examples

Логические операторы

Существует два вида логических операторов: те, которые принимают и возвращают векторы любой длины (элементарные операторы: `!` , `|` , `&` , `xor()`) и те, которые оценивают только первый элемент в каждом аргументе (`&&` , `||`). Второй сорт в основном используется как аргумент `cond` для функции `if` .

Логический оператор	Имея в виду	Синтаксис
!	Не	!Икс
&	элементный (векторизованный) и	x & y
&&	и (только один элемент)	x && y
|	элементный (векторизованный) или	x | Y
||	или (только один элемент)	x || Y
исключающее	элементный (векторный) исключающий ИЛИ	XOR (x, y)

Заметим, что `||` оператор оценивает левое условие, и если левое условие TRUE, правая сторона никогда не оценивается. Это может сэкономить время, если первое является результатом сложной операции. Оператор `&&` также возвращает FALSE без оценки второго

аргумента, когда первый элемент первого аргумента FALSE.

```
> x <- 5
> x > 6 || stop("X is too small")
Error: X is too small
> x > 3 || stop("X is too small")
[1] TRUE
```

Чтобы проверить, является ли значение логическим, вы можете использовать `is.logical()` .

принуждение

Чтобы принудительно использовать переменную для логического использования, используйте `as.logical()` .

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

При применении `as.numeric()` к логическому, возвращается double. `NA` - логическое значение, а логический оператор с `NA` возвращает `NA` если результат неоднозначен.

Интерпретация НС

Подробнее см. Отсутствующие значения .

```
> TRUE & NA
[1] NA
> FALSE & NA
[1] FALSE
> TRUE || NA
[1] TRUE
> FALSE || NA
[1] NA
```

Прочитайте Логический класс онлайн: https://riptutorial.com/ru/r/topic/9016/логический-класс

глава 68: Матрицы

Вступление

Матрицы хранят данные

Examples

Создание матриц

Под капотом матрица представляет собой особый вид вектора с двумя измерениями. Как вектор, матрица может иметь только один класс данных. Вы можете создавать матрицы, используя `matrix` функцию, как показано ниже.

```
matrix(data = 1:6, nrow = 2, ncol = 3)
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Как вы можете видеть, это дает нам матрицу всех чисел от 1 до 6 с двумя строками и тремя столбцами. Параметр `data` принимает вектор значений, `nrow` указывает количество строк в матрице, а `ncol` указывает количество столбцов. По соглашению матрица заполняется столбцом. Поведение по умолчанию может быть изменено с `byrow` параметра `byrow` как показано ниже:

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

Матрицы не должны быть числовыми - любой вектор можно преобразовать в матрицу. Например:

```
matrix(data = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE), nrow = 3, ncol = 2)
##      [,1]  [,2]
## [1,] TRUE FALSE
## [2,] TRUE FALSE
## [3,] TRUE FALSE
matrix(data = c("a", "b", "c", "d", "e", "f"), nrow = 3, ncol = 2)
##      [,1] [,2]
## [1,] "a"  "d"
## [2,] "b"  "e"
## [3,] "c"  "f"
```

Подобно векторам матрицы могут храниться как переменные, а затем вызывать их позже. Строки и столбцы матрицы могут иметь имена. Вы можете посмотреть на них, используя

функции `rownames` и `colnames` . Как показано ниже, строки и столбцы первоначально не имеют имен, которые обозначаются `NULL` . Однако вы можете присвоить им значения.

```
mat1 <- matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
rownames(mat1)
## NULL
colnames(mat1)
## NULL
rownames(mat1) <- c("Row 1", "Row 2")
colnames(mat1) <- c("Col 1", "Col 2", "Col 3")
mat1
##       Col 1 Col 2 Col 3
## Row 1     1     2     3
## Row 2     4     5     6
```

Важно отметить, что аналогично векторам матрицы могут иметь только один тип данных. Если вы попытаетесь указать матрицу с несколькими типами данных, данные будут принуждаться к классу данных более высокого порядка.

Функции `class` , `is` и `as` могут использоваться для проверки и принуждения структур данных таким же образом, как они использовались для векторов в классе 1.

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

Прочитайте Матрицы онлайн: https://riptutorial.com/ru/r/topic/9019/матрицы

глава 69: Машинное обучение

Examples

Создание модели случайного леса

Одним из примеров алгоритмов машинного обучения является алоризм случайного леса (Breiman, L. (2001). Случайные леса. *Machine Learning 45 (5)* , стр. 5-32). Этот алгоритм реализован в R согласно оригинальной реализации Fortran Бреймана в пакете `randomForest` .

Объекты классификатора случайного леса могут быть созданы в R, создавая переменную класса как `factor` , который уже проявляется в наборе данных `iris` . Поэтому мы можем легко создать случайный лес:

```
library(randomForest)

rf <- randomForest(x = iris[, 1:4],
                   y = iris$Species,
                   ntree = 500,
                   do.trace = 100)

rf

# Call:
#   randomForest(x = iris[, 1:4], y = iris$Species, ntree = 500,      do.trace = 100)
# Type of random forest: classification
# Number of trees: 500
# No. of variables tried at each split: 2
#
# OOB estimate of  error rate: 4%
# Confusion matrix:
#   setosa versicolor virginica class.error
# setosa         50          0         0        0.00
# versicolor      0         47         3        0.06
# virginica       0          3        47        0.06
```

параметры	Описание
Икс	кадр данных, содержащий описывающие переменные классов
Y	классы индивидуальных наблюдений. Если этот вектор является `factor` , создается модель классификации, если не создается регрессионная модель.
ntree	Количество построенных отдельных деревьев CART
do.trace	каждый i- *й* шаг, возвращаются ошибки «из коробки» и для каждого класса

Прочитайте Машинное обучение онлайн: https://riptutorial.com/ru/r/topic/8326/машинное-обучение

глава 70: Модели Arima

замечания

Функция `Arima` в пакете прогнозов более ясна в том, как она связана с константами, что может облегчить некоторые пользователи по отношению к функции `arima` в базе R.

ARIMA - это общая структура для моделирования и составления прогнозов из данных временных рядов с использованием (в первую очередь) самой серии. Целью рамок является дифференциация краткосрочной и долгосрочной динамики в серии для повышения точности и достоверности прогнозов. Более поэтически модели ARIMA предоставляют метод описания того, как удары системы передаются во времени.

С эконометрической точки зрения элементы ARIMA необходимы для коррекции последовательной корреляции и обеспечения стационарности.

Examples

Моделирование процесса AR1 с помощью Arima

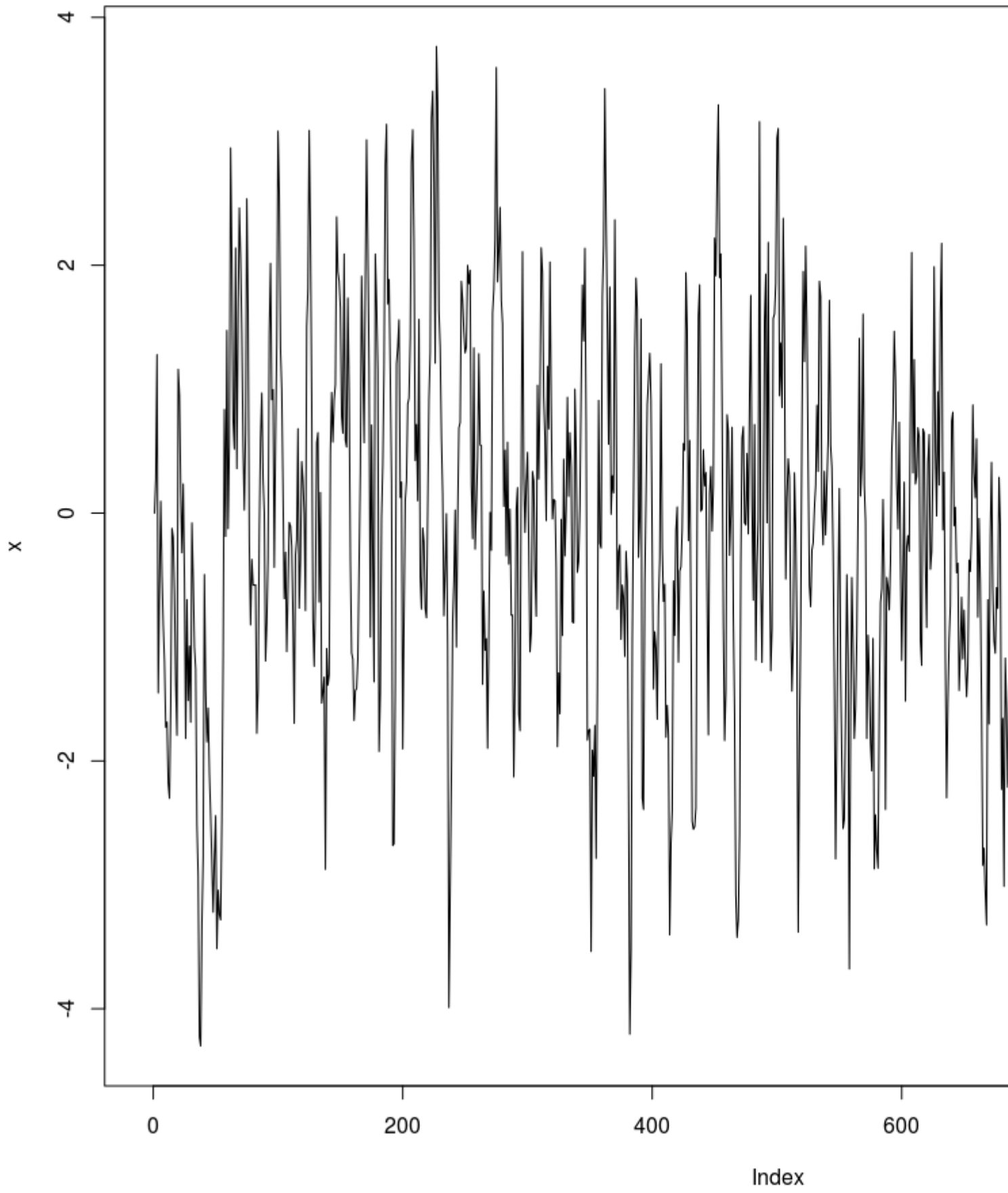
Мы будем моделировать процесс

$$x_t = .7x_{t-1} + \epsilon \qquad \epsilon \sim N(0,1)$$

```
#Load the forecast package
library(forecast)

#Generate an AR1 process of length n (from Cowpertwait & Meltcalfe)
# Set up variables
set.seed(1234)
n <- 1000
x <- matrix(0,1000,1)
w <- rnorm(n)

# loop to create x
for (t in 2:n) x[t] <- 0.7 * x[t-1] + w[t]
plot(x,type='l')
```


x
4
2
0
-2
-4
0
200
400
600
Index

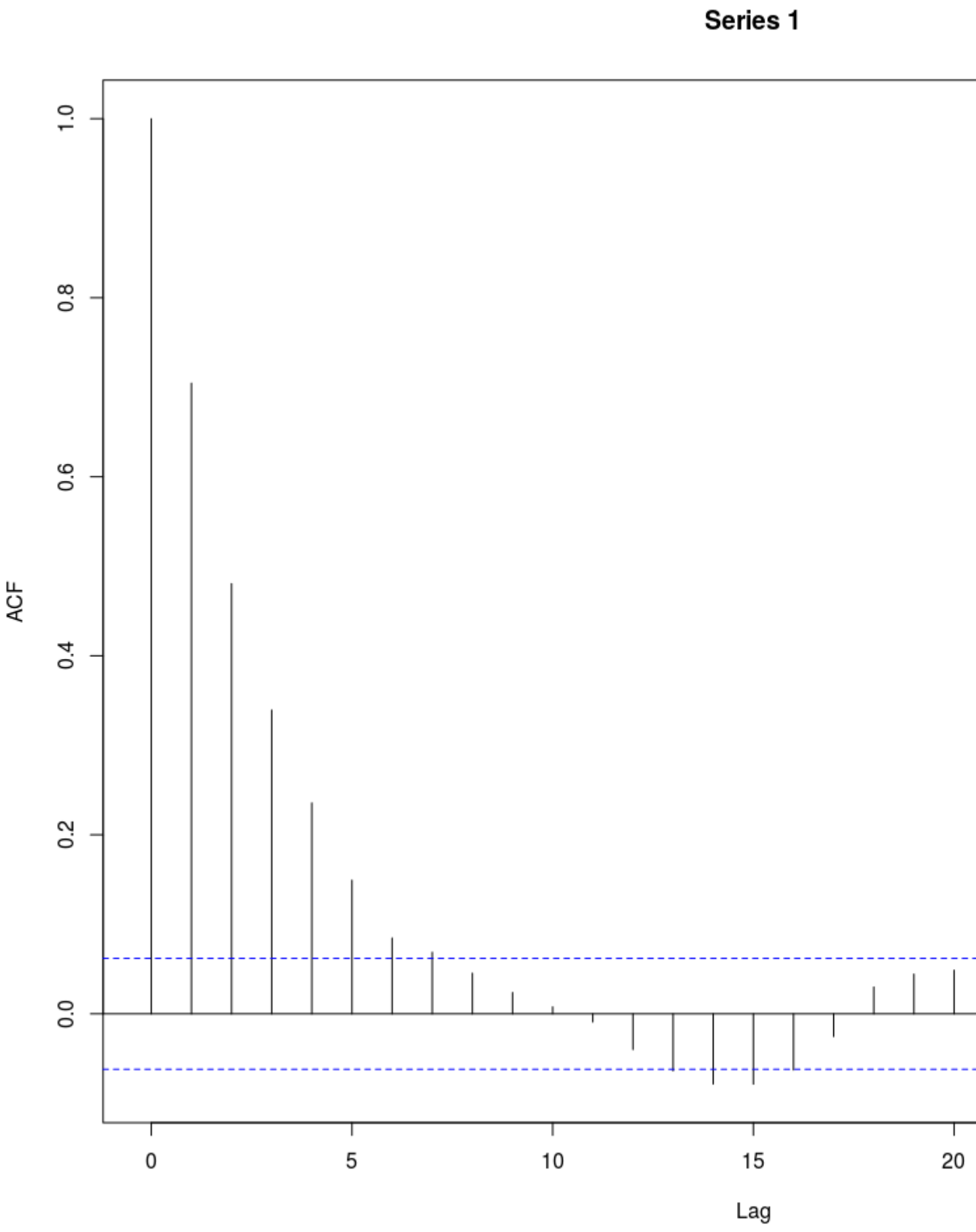
Мы поместим модель Arima с авторегрессивным порядком 1, 0 степеней разности и порядком MA 0.

```
#Fit an AR1 model using Arima
fit <- Arima(x, order = c(1, 0, 0))
summary(fit)
# Series: x
# ARIMA(1,0,0) with non-zero mean
#
# Coefficients:
#          ar1  intercept
#       0.7040    -0.0842
# s.e.  0.0224     0.1062
#
# sigma^2 estimated as 0.9923:  log likelihood=-1415.39
# AIC=2836.79   AICc=2836.81   BIC=2851.51
#
# Training set error measures:
#                         ME      RMSE       MAE MPE MAPE    MASE       ACF1
# Training set -8.369365e-05 0.9961194 0.7835914 Inf  Inf 0.91488 0.02263595
# Verify that the model captured the true AR parameter
```

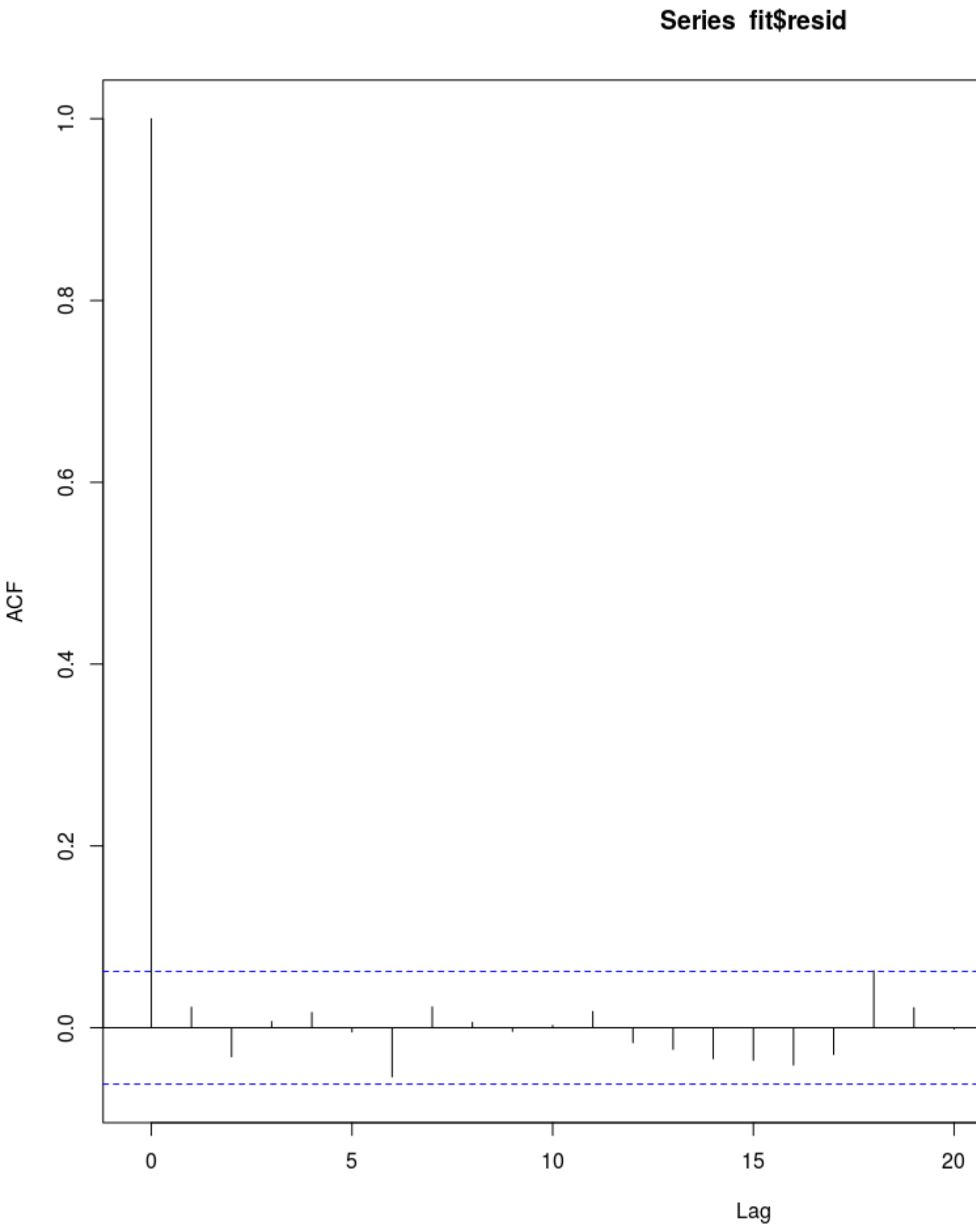
Обратите внимание, что наш коэффициент близок к истинному значению из сгенерированных данных

```
fit$coef[1]
#       ar1
# 0.7040085

#Verify that the model eliminates the autocorrelation
acf(x)
```


Series 1
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
Lag

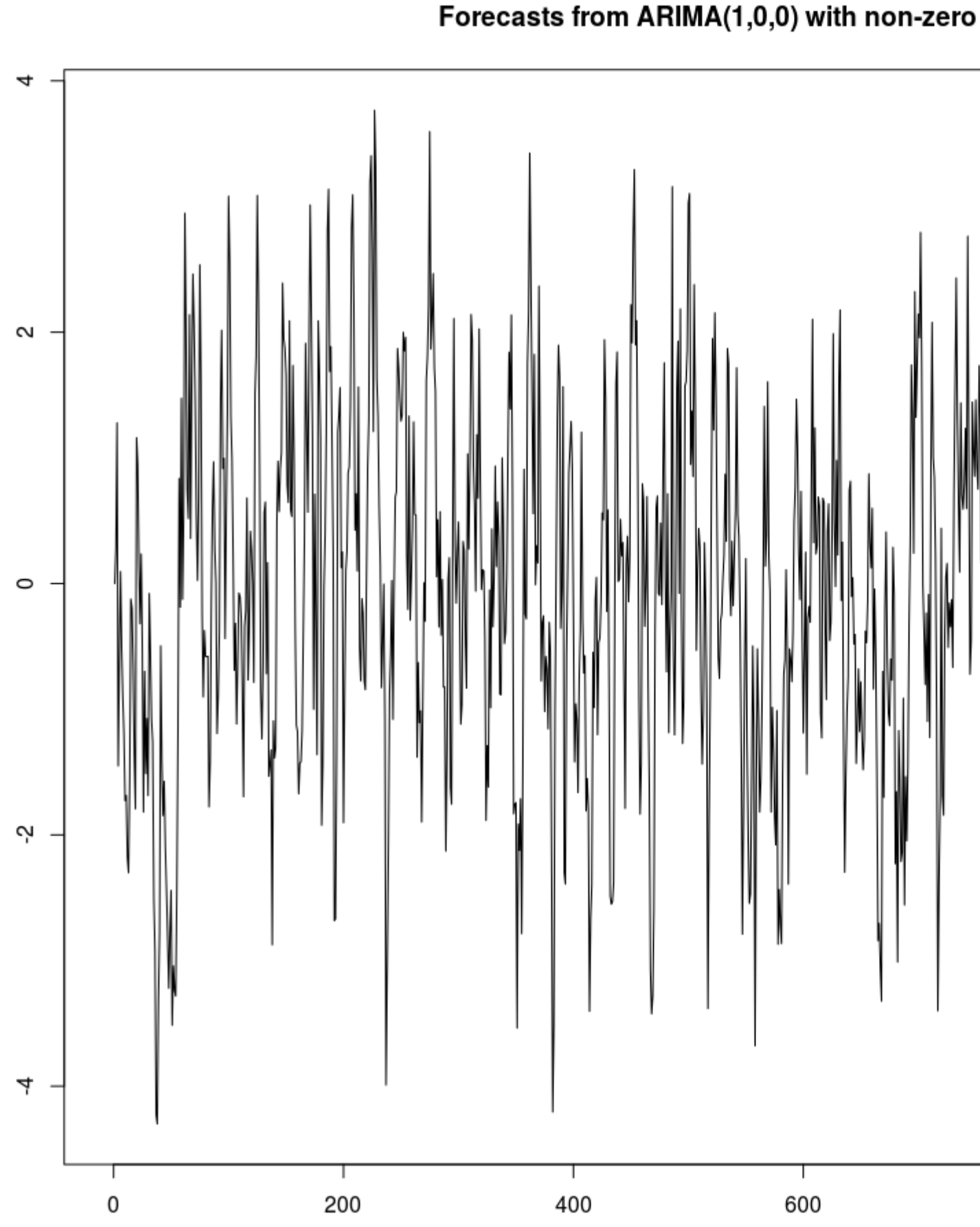
```
acf(fit$resid)
```


Series fit$resid
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
Lag

```
#Forecast 10 periods
fcst <- forecast(fit, h = 100)
fcst
     Point Forecast      Lo 80    Hi 80     Lo 95    Hi 95
1001    0.282529070 -0.9940493 1.559107 -1.669829 2.234887
1002    0.173976408 -1.3872262 1.735179 -2.213677 2.561630
1003    0.097554408 -1.5869850 1.782094 -2.478726 2.673835
1004    0.043752667 -1.6986831 1.786188 -2.621073 2.708578
1005    0.005875783 -1.7645535 1.776305 -2.701762 2.713514
...

#Call the point predictions
fcst$mean
# Time Series:
# Start = 1001
# End = 1100
# Frequency = 1
  [1]  0.282529070  0.173976408  0.097554408  0.043752667  0.005875783 -0.020789866 -
0.039562711 -0.052778954
  [9] -0.062083302
...

#Plot the forecast
plot(fcst)
```


Forecasts from ARIMA(1,0,0) with non-zero
4
2
0
-2
-4
0
200
400
600

Прочитайте Модели Arima онлайн: https://riptutorial.com/ru/r/topic/1725/модели-arima

глава 71: Нестандартная оценка и стандартная оценка

Вступление

Dplyr и многие современные библиотеки в R используют нестандартную оценку (NSE) для интерактивного программирования и стандартной оценки (SE) для программирования 1 .

Например, `summarise()` использование функции нестандартная оценка , но полагается на `summarise_()` , который использует стандартную оценку.

Библиотека lazyeval позволяет легко превратить стандартную функцию оценки в функции NSE.

Examples

Примеры со стандартными глаголами dplyr

Функции NSE должны использоваться в интерактивном программировании. Однако при разработке новых функций в новом пакете лучше использовать версию SE.

Загрузить dplyr и lazyeval:

```
library(dplyr)
library(lazyeval)
```

фильтрация

Версия NSE

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

SE (для использования при программировании функций в новом пакете)

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

Суммировать

Версия NSE

```
summarise(mtcars,  mean(disp))
summarise(mtcars,  mean_disp = mean(disp))
```

Версия SE

```
summarise_(mtcars, .dots = lazyeval::interp(~ mean(x), x = quote(disp)))
summarise_(mtcars, .dots = setNames(list(lazyeval::interp(~ mean(x), x = quote(disp))),
"mean_disp"))
summarise_(mtcars, .dots = list("mean_disp" = lazyeval::interp(~ mean(x), x = quote(disp))))
```

мутировать

Версия NSE

```
mutate(mtcars, displ_l = disp / 61.0237)
```

Версия SE

```
mutate_(
    .data = mtcars,
    .dots = list(
        "displ_l" = lazyeval::interp(
                        ~ x / 61.0237, x = quote(disp)
            )
         )
)
```

Прочитайте Нестандартная оценка и стандартная оценка онлайн:
https://riptutorial.com/ru/r/topic/9365/нестандартная-оценка-и-стандартная-оценка

глава 72: Обновление R и библиотеки пакетов

Examples

В Windows

Установка по умолчанию R на хранящиеся в Windows файлы (и, следовательно, библиотека) по выделенной папке на R-версию в файлах программ.

Это означает, что по умолчанию вы будете работать с несколькими версиями R параллельно и, следовательно, с отдельными библиотеками.

Если это не то, что вы хотите, и вы предпочитаете всегда работать с одним экземпляром R, то вы не будете постепенно обновлять его, рекомендуется изменить папку установки R. В мастере просто укажите эту папку (лично я использую `c:\stats\R`). Затем, для любого обновления, есть одна возможность перезаписать это R. Если вы также хотите обновить (все) пакеты, это деликатный выбор, так как он может сломать часть вашего кода (это появилось для меня с пакетом `tm`). Ты можешь:

- Сначала сделайте копию всей вашей библиотеки перед обновлением пакетов.
- Поддерживайте собственный репозиторий исходных пакетов, например, используя пакет `miniCRAN`

Если вы хотите обновить все пакеты - без каких-либо проверок вы можете вызвать use `packageStatus` как в:

```
pkgs <- packageStatus()  # choose mirror
upgrade(pkgs)
```

Наконец, существует очень удобный пакет для выполнения всех операций, а именно `installr` , даже с выделенным gui. Если вы хотите использовать gui, вы должны использовать Rgui и не загружать пакет в RStudio. Использование пакета с кодом так же просто, как:

```
install.packages("installr") # install
setInternet2(TRUE) # only for R versions older than 3.3.0
installr::updateR() # updating R.
```

Я ссылаюсь на большую документацию https://www.r-statistics.com/tag/installr/ и, в частности, поэтапный процесс со скриншотами в Windows: https://www.r-statistics.com/2015/06/ а-шаг за шагом-картинка-учебник-для-модернизации-р-на окнах /

Обратите внимание, что я все же защищаю использование одного каталога, т. Е. удаление ссылки на версию R в имени папки установки.

Прочитайте Обновление R и библиотеки пакетов онлайн:
https://riptutorial.com/ru/r/topic/4088/обновление-r-и-библиотеки-пакетов

глава 73: Обновление версии R

Вступление

Установка или обновление вашего ПО даст доступ к новым функциям и исправлениям ошибок. Обновление вашей установки R можно выполнить несколькими способами. Один простой способ - перейти на сайт R и загрузить последнюю версию для вашей системы.

Examples

Установка с сайта R

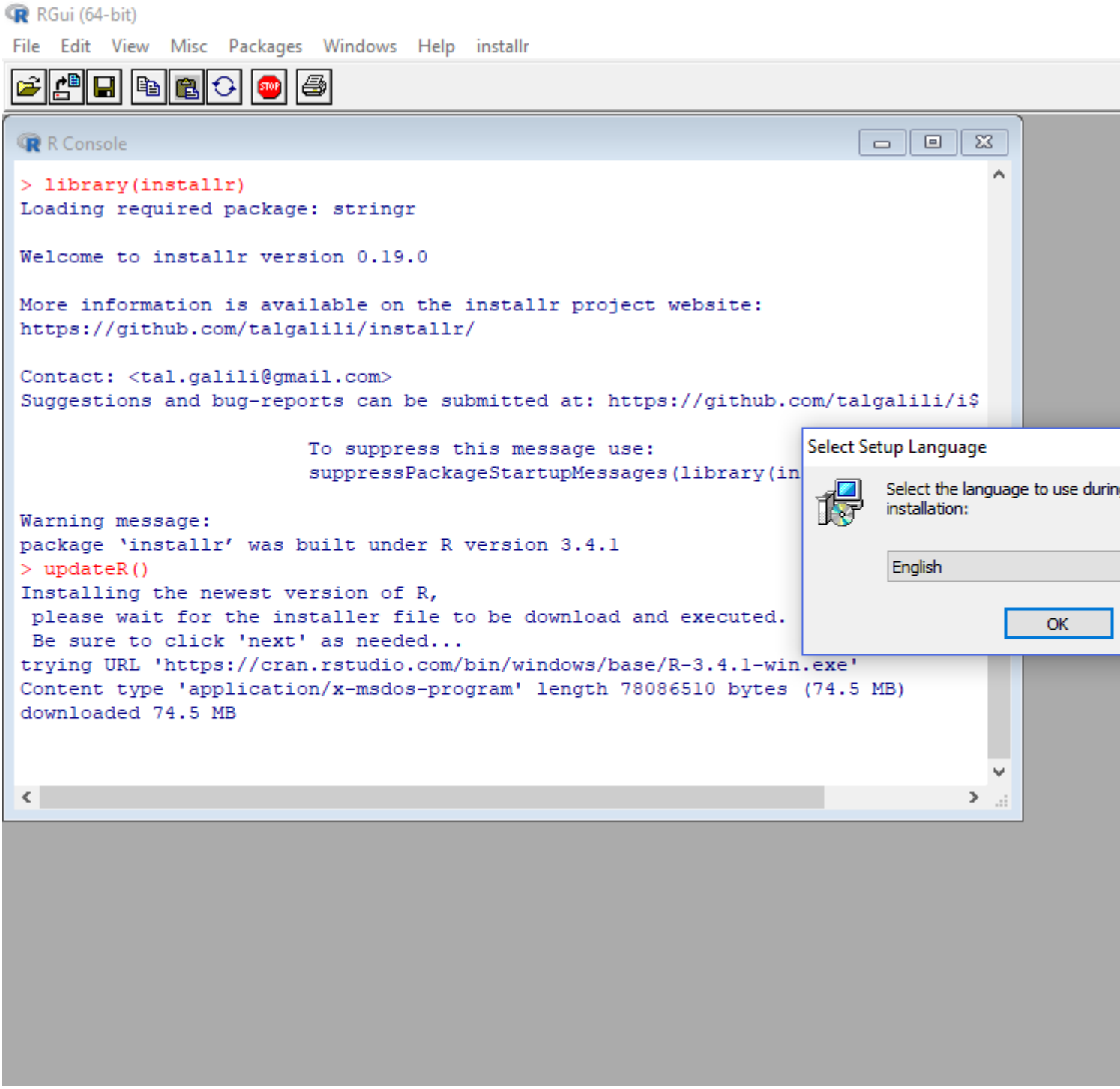
Чтобы получить последнюю версию, перейдите на страницу https://cran.r-project.org/ и загрузите файл для своей операционной системы. Откройте загруженный файл и следуйте инструкциям на экране. Все настройки можно оставить по умолчанию, если вы не хотите изменить определенное поведение.

Обновление из R с помощью пакета installr

Вы также можете обновить R из R, используя удобный пакет под названием **installr** .

Open R Console (НЕ RStudio, это не работает из RStudio) и выполните следующий код, чтобы установить пакет и инициировать обновление.

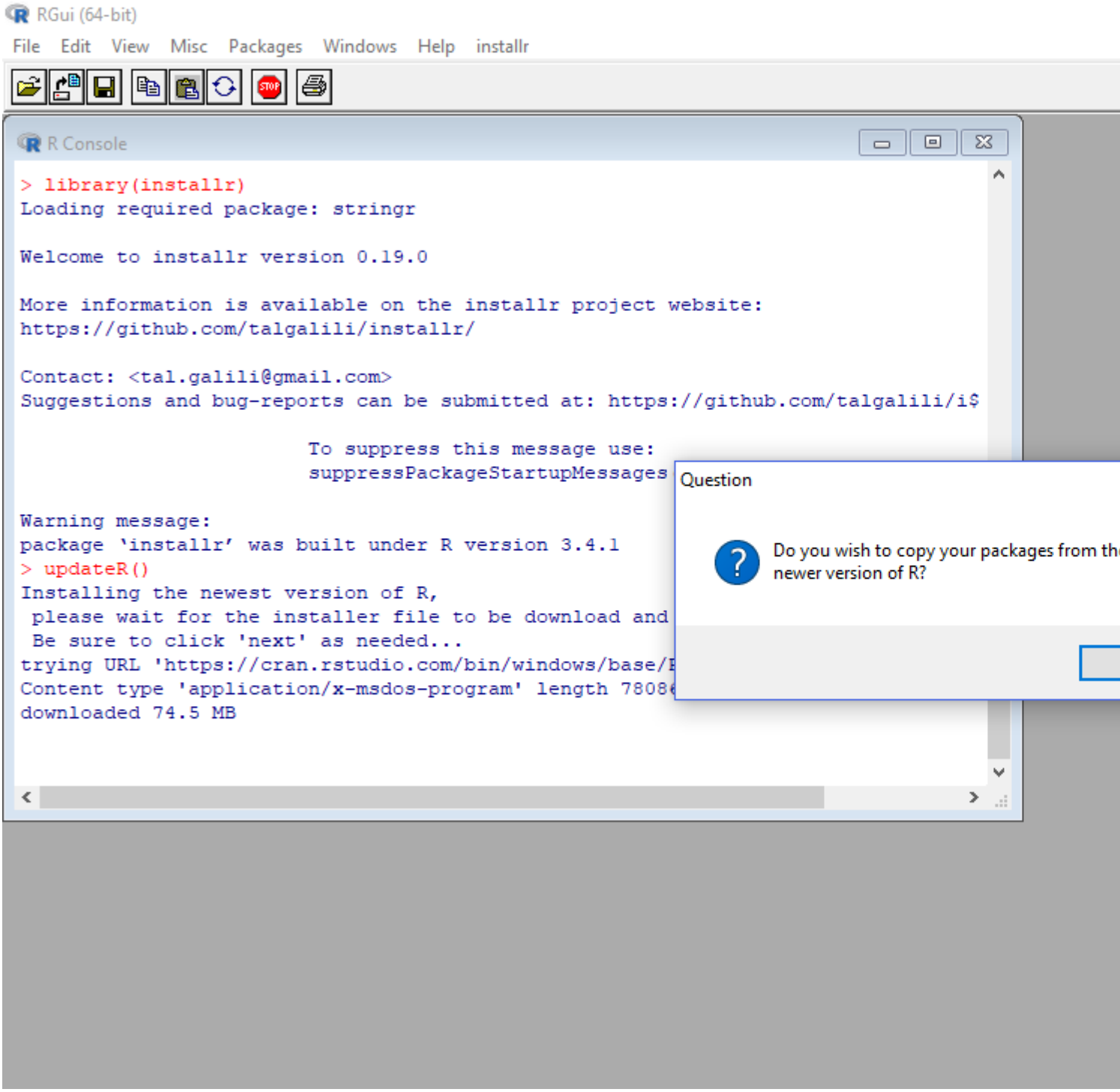
```
install.packages("installr")
library("installr")
updateR()
```



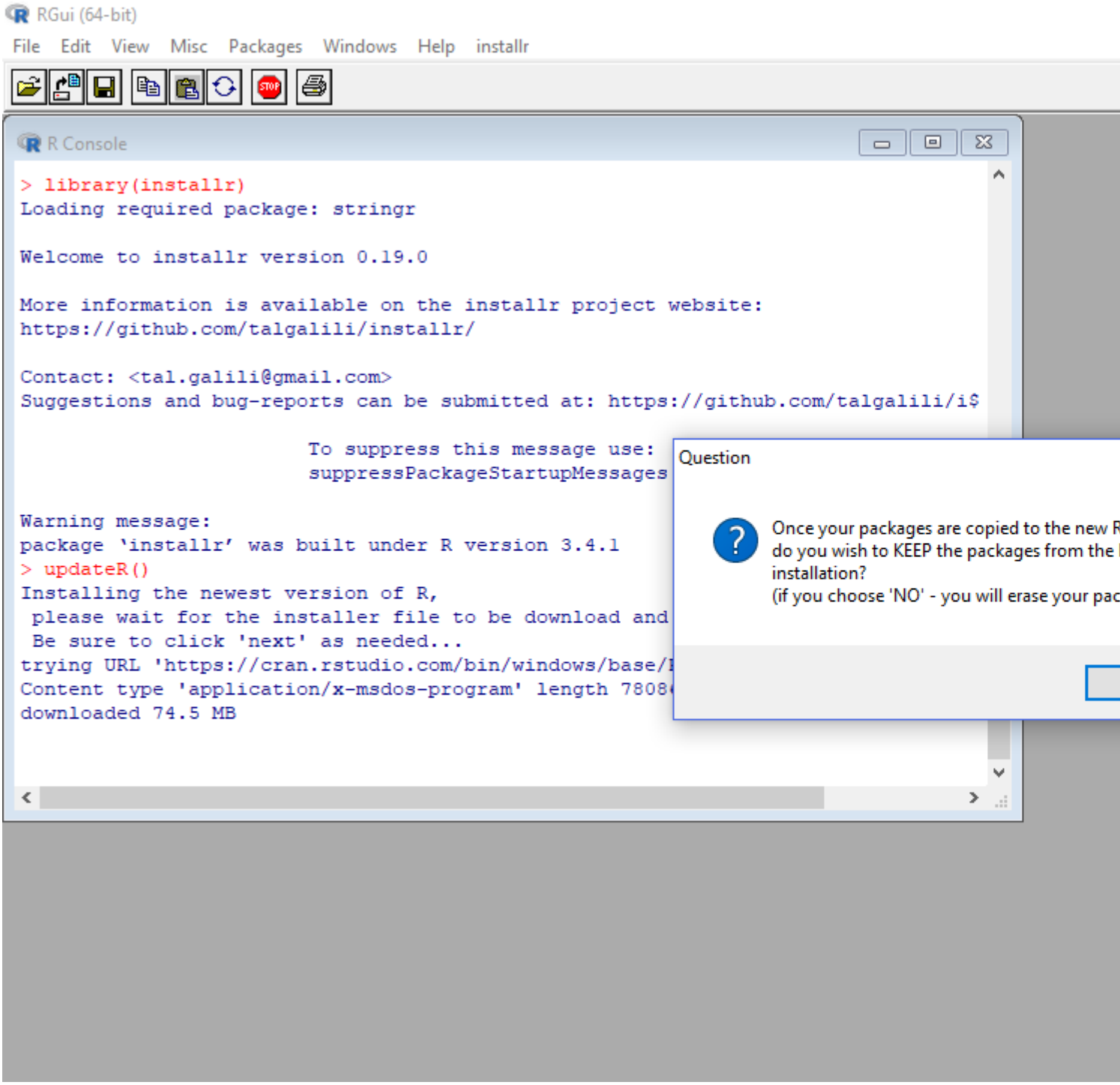
Выбор старых пакетов

По завершении установки нажмите кнопку «Готово».

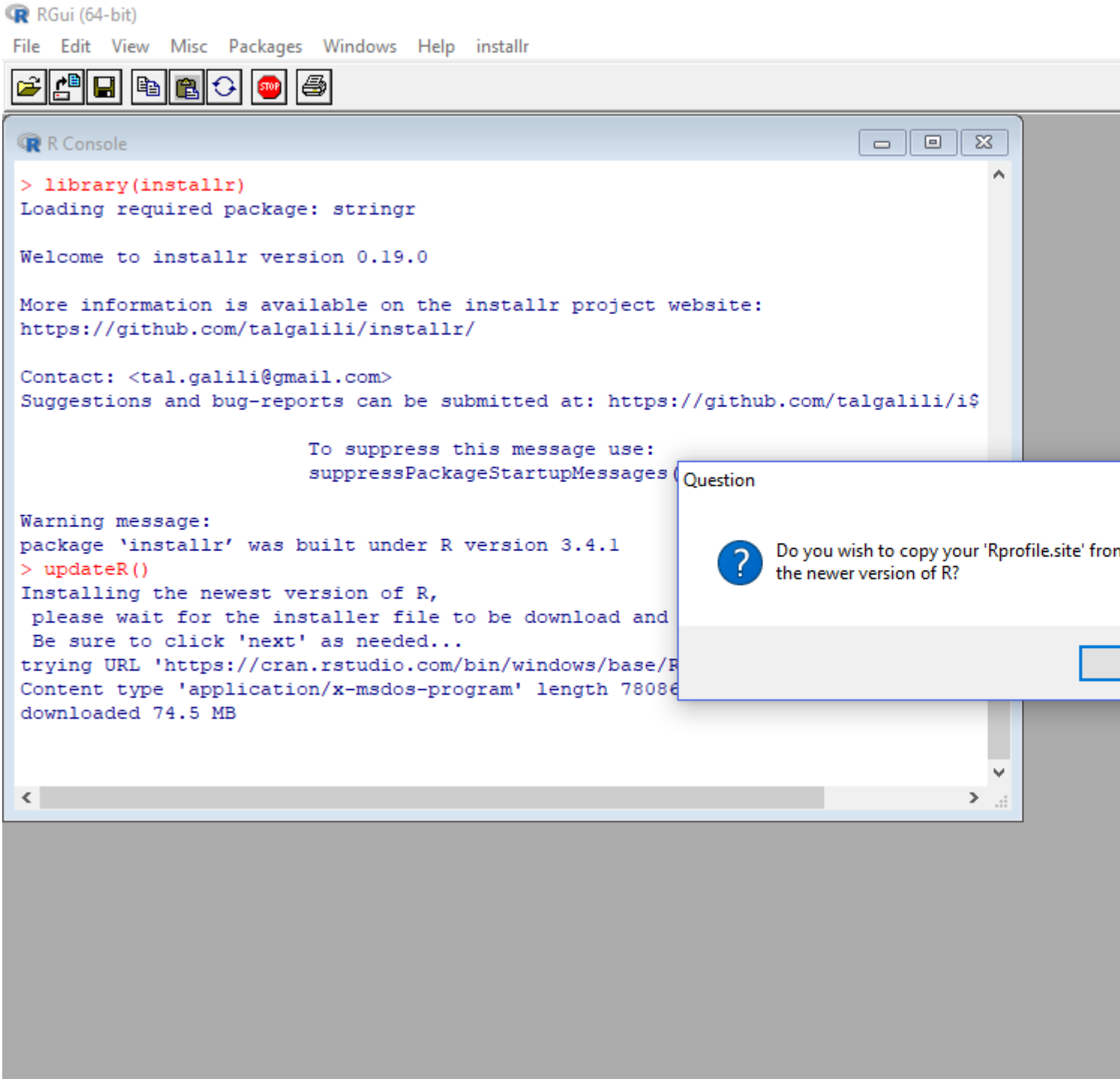
Теперь он спрашивает, хотите ли вы скопировать свои пакеты из старой версии R в более новую версию R. Как только вы выберете да, весь пакет будет скопирован в более новую версию R.



После этого вы можете выбрать, хотите ли вы сохранить старые пакеты или удалить.

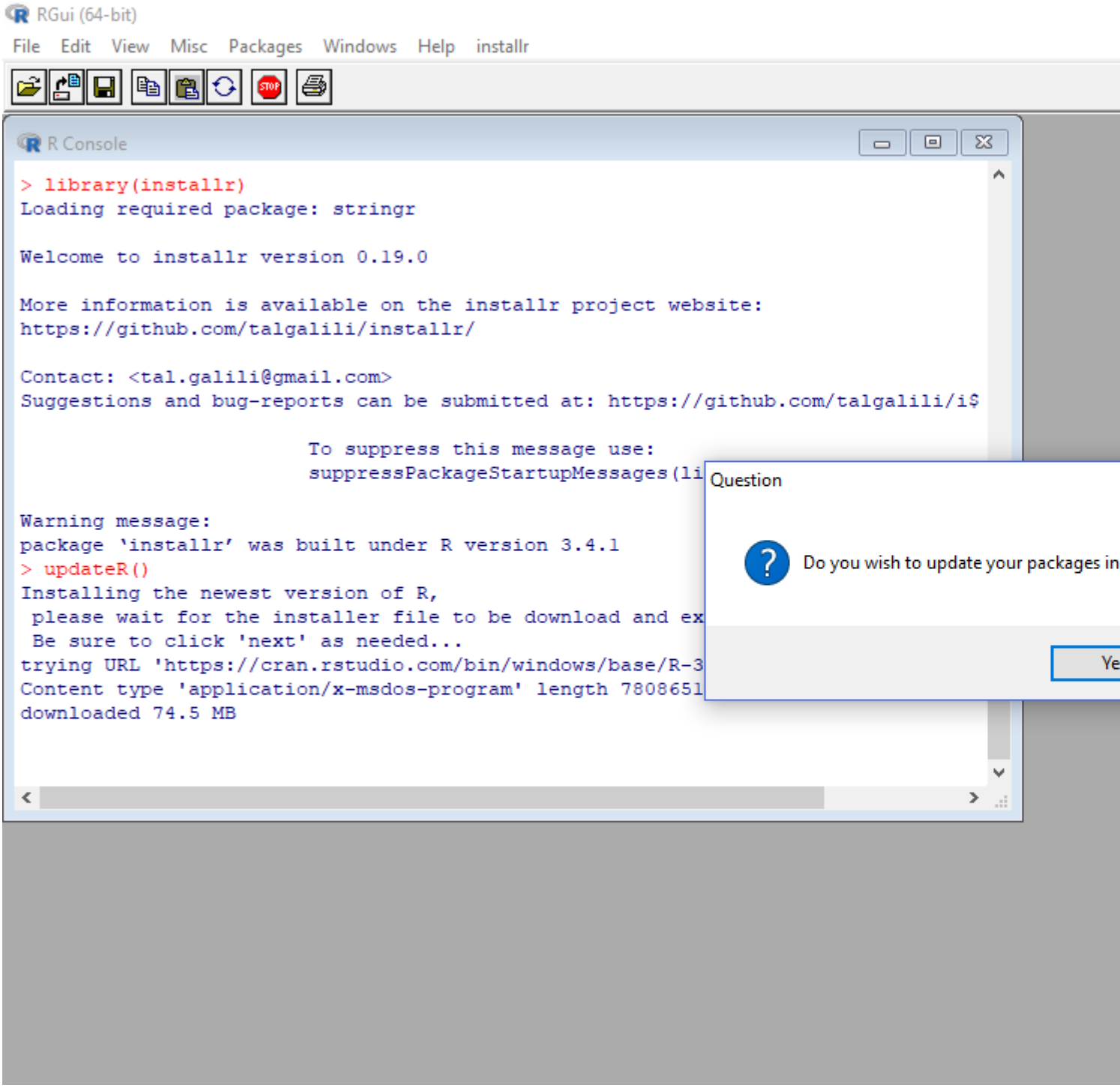


Вы даже можете перенести свой Rprofile.site из старой версии, чтобы сохранить все свои настроенные параметры.



Обновление пакетов

Вы можете обновить установленные пакеты после обновления R.



После его завершения перезапустите R и наслаждайтесь изучением.

Проверить версию R

Вы можете проверить версию R с помощью консоли

```
version
```

Прочитайте Обновление версии R онлайн: https://riptutorial.com/ru/r/topic/10729/обновление-версии-r

глава 74: Обобщенные линейные модели

Examples

Логистическая регрессия по набору данных «Титаник»

Логистическая регрессия является частным случаем *обобщенной линейной модели* , используемой для моделирования дихотомических исходов (*пробит* и *дополнительные лог-лог* - модели тесно связаны).

Имя происходит от используемой *функции ссылки, функции logit* или *log-odds* . Обратная функция *логита* называется *логистической функцией* и задается:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

Эта функция принимает значение между *] -Inf; + Inf [* и возвращает значение от *0* до *1* ; т.е. *логистическая функция* принимает линейный предиктор и возвращает вероятность.

`glm` регрессию можно выполнить с `glm` функции `glm` с опцией `family = binomial` (ярлык для `family = binomial(link="logit")` , *логит* - это функция связи по умолчанию для семейства биномиал).

В этом примере мы пытаемся предсказать судьбу пассажиров на борту RMS Titanic.

Прочтите данные:

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

Очистите отсутствующие значения:

В этом случае мы заменяем недостающие значения на приближение, среднее.

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

Обучить модель:

```
titanic.train <- glm(survived ~ pclass + sex + age,
                         family = binomial, data = titanic)
```

Резюме модели:

```
summary(titanic.train)
```

Выход:

```
Call:
glm(formula = survived ~ pclass + sex + age, family = binomial, data = titanic)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.6452  -0.6641  -0.3679   0.6123   2.5615

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.552261   0.342188  10.381  < 2e-16 ***
pclass2nd   -1.170777   0.211559  -5.534 3.13e-08 ***
pclass3rd   -2.430672   0.195157 -12.455  < 2e-16 ***
sexmale     -2.463377   0.154587 -15.935  < 2e-16 ***
age         -0.042235   0.007415  -5.696 1.23e-08 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1686.8  on 1312  degrees of freedom
Residual deviance: 1165.7  on 1308  degrees of freedom
AIC: 1175.7

Number of Fisher Scoring iterations: 5
```

- Первое, что отображается, - это вызов. Это напоминание о модели и указанных опциях.
- Затем мы видим остатки отклонения, которые являются мерой соответствия модели. Эта часть вывода показывает распределение остатков отклонения для отдельных случаев, используемых в модели.
- В следующей части вывода показаны коэффициенты, их стандартные ошибки, z-статистика (иногда называемая статистикой Wald z) и связанные с ней значения p.
    - Качественные переменные «думмируются». В качестве ссылки рассматривается модальность. Эталонная модальность может быть изменена с `I` в формуле.
    - Все четыре предиктора статистически значимы на уровне 0,1%.
    - Коэффициенты логистической регрессии дают изменение логарифмических коэффициентов результата для однократного увеличения предикторной переменной.
    - Чтобы увидеть *соотношение шансов* (мультипликативное изменение шансов выживания на единицу увеличения предикторной переменной), оценивайте параметр.
    - Чтобы увидеть доверительный интервал (CI) параметра, используйте `confint` .
- Ниже таблицы коэффициентов соответствуют индексы, в том числе нулевые и отклоняющие значения и Akaike Information Criterion (AIC), которые могут использоваться для сравнения производительности модели.

- Сравнивая модели с максимальной вероятностью с теми же данными, чем меньше AIC, тем лучше подходит.
- Одним из показателей соответствия модели является значение общей модели. Этот тест спрашивает, подходит ли модель с предикторами лучше, чем модель с просто перехватом (т. Е. Нулевой моделью).

Пример коэффициентов шансов:

```
exp(coef(titanic.train)[3])

 pclass3rd
0.08797765
```

С этой моделью, по сравнению с первым классом, пассажиры третьего класса имеют примерно одну десятую шансов на выживание.

Пример доверительного интервала для параметров:

```
confint(titanic.train)

Waiting for profiling to be done...
                 2.5 %      97.5 %
(Intercept)  2.89486872  4.23734280
pclass2nd   -1.58986065 -0.75987230
pclass3rd   -2.81987935 -2.05419500
sexmale     -2.77180962 -2.16528316
age         -0.05695894 -0.02786211
```

Пример расчета значения общей модели:

Статистическая статистика распределяется по хи-квадрату со степенями свободы, равными различиям в степенях свободы между током и нулевой моделью (т. Е. Числом предикторных переменных в модели).

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual
, lower.tail = FALSE))
[1] 1.892539e-111
```

Значение p находится вблизи 0, что показывает сильно значимую модель.

Прочитайте Обобщенные линейные модели онлайн: https://riptutorial.com/ru/r/topic/2892/обобщенные-линейные-модели

глава 75: Обработка естественного языка

Вступление

Обработка естественного языка (NLP) - это область компьютерных наук, ориентированная на извлечение информации из текстового ввода, созданного людьми.

Examples

Создать частотную матрицу

Самый простой подход к проблеме (и наиболее часто используемый до сих пор) заключается в разделении предложений на *токены* . Упрощение, *слова* имеют абстрактные и субъективные значения для людей, использующих и получающих их, *токены* имеют объективную интерпретацию: упорядоченную последовательность символов (или байтов). Когда предложения разделены, порядок маркера игнорируется. Такой подход к проблеме в известной модели **мешков слов** .

Терминная частота - это словарь, в котором каждому токену присваивается *вес* . В первом примере мы строим матрицу частот термов из corpus **corpus** (сборник **документов**) с пакетом R `tm` .

```
require(tm)
doc1 <- "drugs hospitals doctors"
doc2 <- "smog pollution environment"
doc3 <- "doctors hospitals healthcare"
doc4 <- "pollution environment water"
corpus <- c(doc1, doc2, doc3, doc4)
tm_corpus <- Corpus(VectorSource(corpus))
```

В этом примере мы создали корпус класса `Corpus` определенный пакетом `tm` с двумя функциями `Corpus` и `VectorSource` , который возвращает объект `VectorSource` из символьного вектора. Объект `tm_corpus` представляет собой список наших документов с дополнительными (и необязательными) метаданными для описания каждого документа.

```
str(tm_corpus)
List of 4
 $ 1:List of 2
  ..$ content: chr "drugs hospitals doctors"
  ..$ meta   :List of 7
  .. ..$ author       : chr(0)
  .. ..$ datetimestamp: POSIXlt[1:1], format: "2017-06-03 00:31:34"
  .. ..$ description  : chr(0)
  .. ..$ heading      : chr(0)
  .. ..$ id           : chr "1"
  .. ..$ language     : chr "en"
  .. ..$ origin       : chr(0)
```

```
  .. ..- attr(*, "class")= chr "TextDocumentMeta"
  ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
[truncated]
```

Когда у нас есть `Corpus` , мы можем перейти к препроцессу токенов, содержащихся в `Corpus` для улучшения качества конечного результата (термин матрица частот). Для этого мы используем функцию `tm` `tm_map` , которая аналогично `apply` семейству функций преобразует документы в корпус, применяя функцию к каждому документу.

```
tm_corpus <- tm_map(tm_corpus, tolower)
tm_corpus <- tm_map(tm_corpus, removeWords, stopwords("english"))
tm_corpus <- tm_map(tm_corpus, removeNumbers)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
tm_corpus <- tm_map(tm_corpus, stemDocument, language="english")
tm_corpus <- tm_map(tm_corpus, stripWhitespace)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
```

После этих преобразований мы, наконец, создаем матрицу частот с

```
tdm <- TermDocumentMatrix(tm_corpus)
```

который дает

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
Non-/sparse entries: 12/20
Sparsity           : 62%
Maximal term length: 9
Weighting          : term frequency (tf)
```

что мы можем рассматривать, преобразуя его в матрицу

```
as.matrix(tdm)

           Docs
Terms       character(0) character(0) character(0) character(0)
  doctor               1            0            1            0
  drug                 1            0            0            0
  environ              0            1            0            1
  healthcar            0            0            1            0
  hospit               1            0            1            0
  pollut               0            1            0            1
  smog                 0            1            0            0
  water                0            0            0            1
```

Каждая строка представляет собой частоту каждого маркера - то , как вы заметили, были обусловлены (например , `environment` для `environ`) - в каждом документе (4 документа, 4 колонки).

В предыдущих строках мы взвешивали каждый токен / документ пары с абсолютной частотой (т. Е. Количество экземпляров токена, которые появляются в документе).

Прочитайте Обработка естественного языка онлайн: https://riptutorial.com/ru/r/topic/10119/обработка-естественного-языка

глава 76: Обработка строк со строковым пакетом

замечания

Для установки пакета просто выполните:

```
install.packages("stringi")
```

загрузить его:

```
require("stringi")
```

Examples

Считать шаблон внутри строки

С фиксированным рисунком

```
stri_count_fixed("babab", "b")
# [1] 3
stri_count_fixed("babab", "ba")
# [1] 2
stri_count_fixed("babab", "bab")
# [1] 1
```

Врожденная:

```
length(gregexpr("b","babab")[[1]])
# [1] 3
length(gregexpr("ba","babab")[[1]])
# [1] 2
length(gregexpr("bab","babab")[[1]])
# [1] 1
```

функция векторизована поверх строки и шаблона:

```
stri_count_fixed("babab", c("b","ba"))
# [1] 3 2
stri_count_fixed(c("babab","bbb","bca","abc"), c("b","ba"))
# [1] 3 0 1 0
```

Базовое решение R :

```
sapply(c("b","ba"),function(x)length(gregexpr(x,"babab")[[1]]))
```

```
# b ba
# 3  2
```

С регулярным выражением

Первый пример - найти `a` и любой символ после

Второй пример - найти `a` и любую цифру после

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

Дублирующие струны

```
stri_dup("abc",3)
# [1] "abcabcabc"
```

Базовое решение R, которое делает то же самое, будет выглядеть так:

```
paste0(rep("abc",3),collapse = "")
# [1] "abcabcabc"
```

Вставить векторы

```
stri_paste(LETTERS,"-", 1:13)
# [1] "A-1"  "B-2"  "C-3"  "D-4"  "E-5"  "F-6"  "G-7"  "H-8"  "I-9"  "J-10" "K-11" "L-12" "M-
13"
# [14] "N-1"  "O-2"  "P-3"  "Q-4"  "R-5"  "S-6"  "T-7"  "U-8"  "V-9"  "W-10" "X-11" "Y-12" "Z-
13"
```

Естественно, мы могли бы сделать это в R через:

```
> paste(LETTERS,1:13,sep="-")
 #[1] "A-1"  "B-2"  "C-3"  "D-4"  "E-5"  "F-6"  "G-7"  "H-8"  "I-9"  "J-10" "K-11" "L-12" "M-
13"
 #[14] "N-1"  "O-2" "P-3"  "Q-4"  "R-5"  "S-6"  "T-7"  "U-8"  "V-9"  "W-10" "X-11" "Y-12" "Z-
13"
```

Разделение текста на определенный фиксированный шаблон

Разделить вектор текстов с использованием одного шаблона:

```
stri_split_fixed(c("To be or not to be.", "This is very short sentence."),", ")
# [[1]]
# [1] "To"  "be"  "or"  "not" "to"  "be."
#
# [[2]]
# [1] "This"      "is"        "very"      "short"     "sentence."
```

Разделите один текст, используя множество шаблонов:

```
stri_split_fixed("Apples, oranges and pineaplles.",c(" ", ",", "s"))
# [[1]]
# [1] "Apples,"     "oranges"     "and"         "pineaplles."
#
# [[2]]
# [1] "Apples"                   " oranges and pineaplles."
#
# [[3]]
# [1] "Apple"          ", orange"       " and pineaplle" "."
```

Прочитайте Обработка строк со строковым пакетом онлайн:
https://riptutorial.com/ru/r/topic/1670/обработка-строк-со-строковым-пакетом

глава 77: Объектно-ориентированное программирование в R

Вступление

На этой странице документации описываются четыре объектные системы в R и их сходства и различия на высоком уровне. Более подробную информацию о каждой отдельной системе можно найти на ее собственной странице темы.

Четыре системы: S3, S4, Reference Classes и S6.

Examples

S3

Объектная система S3 - очень простая система OO в R.

Каждый объект имеет класс S3. Он может быть получен (получен?) С `class` функции.

```
> class(3)
[1] "numeric"
```

Его также можно задать с помощью `class` функций:

```
> bicycle <- 2
> class(bicycle) <- 'vehicle'
> class(bicycle)
[1] "vehicle"
```

Его также можно установить с помощью функции `attr` :

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

Объект может иметь много классов:

```
> class(x = bicycle) <- c('human-powered vehicle', class(x = bicycle))
> class(x = bicycle)
[1] "human-powered vehicle" "vehicle"
```

При использовании общей функции R использует первый элемент класса с имеющимся общим.

Например:

```
> summary.vehicle <- function(object, ...) {
+   message('this is a vehicle')
+ }
> summary(object = my_bike)
this is a vehicle
```

Но если мы теперь определим `summary.bicycle` :

```
> summary.bicycle <- function(object, ...) {
+   message('this is a bicycle')
+ }
> summary(object = my_bike)
this is a bicycle
```

Прочитайте Объектно-ориентированное программирование в R онлайн:
https://riptutorial.com/ru/r/topic/9723/объектно-ориентированное-программирование-в-r

глава 78: Объем переменных

замечания

Наиболее распространенная ловушка с размахом возникает при распараллеливании. Все переменные и функции должны быть переданы в новую среду, которая запускается в каждом потоке.

Examples

Окружающая среда и функции

Переменные, объявленные внутри функции, только существуют (если не переданы) внутри этой функции.

```
x <- 1

foo <- function(x) {
    y <- 3
    z <- x + y
    return(z)
}

y
```

Ошибка: объект 'y' не найден

Переменные, переданные в функцию, а затем переназначенные, перезаписываются, *но только внутри функции* .

```
foo <- function(x) {
    x <- 2
    y <- 3
    z <- x + y
    return(z)
}

foo(1)
x
```

5

1

Переменные, назначенные в более высокой среде, чем функция, существуют внутри этой функции, без передачи.

```
foo <- function() {
```

```
    y <- 3
    z <- x + y
    return(z)
}

foo()
```

4

Субфункции

Функции, вызываемые внутри функции (т. Е. Подфункции), должны быть определены внутри этой функции для доступа к любым переменным, определенным в локальной среде, без передачи.

Это не удается:

```
bar <- function() {
    z <- x + y
    return(z)
}

foo <- function() {
    y <- 3
    z <- bar()
    return(z)
}

foo()
```

Ошибка в bar (): объект 'y' не найден

Это работает:

```
foo <- function() {

    bar <- function() {
        z <- x + y
        return(z)
    }

    y <- 3
    z <- bar()
    return(z)
}

foo()
```

4

Глобальное присвоение

Переменные можно назначить глобально из любой среды, используя `<<-` . `bar()` теперь

может получить доступ к `y` .

```
bar <- function() {
    z <- x + y
    return(z)
}

foo <- function() {
    y <<- 3
    z <- bar()
    return(z)
}

foo()
```

4

Глобальное задание крайне обескуражено. Очень важно использовать функцию-оболочку или явно вызывать переменные из другой локальной среды.

Явное присвоение сред и переменных

Среды в R можно явно вызвать и назвать. Переменные могут быть явно назначены и вызывать в этих средах или из них.

Обычно создаваемая среда - это среда, в которой заключен `package:base` или subenvironment внутри `package:base` .

```
e1 <- new.env(parent = baseenv())
e2 <- new.env(parent = e1)
```

Переменные могут быть явно назначены и вызывать в этих средах или из них.

```
assign("a", 3, envir = e1)
   get("a", envir = e1)
   get("a", envir = e2)
```

3

3

Так как `e2` наследует от `e1` , `a` равно `3` как в `e1` и `e2` . Однако присвоение `a` внутри `e2` не изменяет значение `a` в `e1` .

```
assign("a", 2, envir = e2)
   get("a", envir = e2)
   get("a", envir = e1)
```

3

2

Выход функции

Функция `on.exit()` удобна для очистки переменных, если необходимо назначить глобальные переменные.

Некоторые параметры, особенно для графики, могут быть установлены только глобально. Эта небольшая функция распространена при создании более специализированных графиков.

```
new_plot <- function(...) {

     old_pars <- par(mar = c(5,4,4,2) + .1, mfrow = c(1,1))
     on.exit(par(old_pars))
     plot(...)
 }
```

Пакеты и маскировка

Функции и объекты в разных пакетах могут иметь одно и то же имя. Пакет, загруженный позже, «замаскирует» более ранний пакет, и будет напечатано предупреждающее сообщение. При вызове функции по имени будет запущена функция из последнего загруженного пакета. К более ранней функции можно получить доступ явно.

```
library(plyr)
library(dplyr)
```

> Присоединение пакета: 'dplyr'
>
> Следующие объекты маскируются из пакета: plyr:
>
> упорядочивать, считать, убывать, сбрасывать, id, мутировать, переименовывать, суммировать, суммировать
>
> Следующие объекты маскируются из «package: stats»:
>
> фильтр, лаг
>
> Следующие объекты маскируются из «package: base»:
>
> пересекать, setdiff, setequal, union

При написании кода всегда лучше использовать функции, явно используя `package::function()` чтобы избежать этой проблемы.

Прочитайте Объем переменных онлайн: https://riptutorial.com/ru/r/topic/3138/объем-переменных

глава 79: Операторы труб (%>% и другие)

Вступление

Операторы труб, доступные в `magrittr` , `dplyr` и других R-пакетах, обрабатывают объект данных, используя последовательность операций, передавая результат одного шага в качестве ввода для следующего шага с использованием инфикс-операторов, а не более типичный R-метод вложенных вызовы функций.

Обратите внимание, что целью операторов труб является повышение удобочитаемости письменного кода. См. Раздел «Примечания» для соображений производительности.

Синтаксис

- lhs%>% rhs # pipe синтаксис для `rhs(lhs)`
- lhs%>% rhs (a = 1) # синтаксис канала для `rhs(lhs, a = 1)`
- lhs%>% rhs (a = 1, b =.) # синтаксис канала для `rhs(a = 1, b = lhs)`
- lhs% <>% rhs # pipe синтаксис для `lhs <- rhs(lhs)`
- lhs% $% rhs (a) # синтаксис канала для `with(lhs, rhs(lhs$a))`
- lhs% T>% rhs # синтаксис канала для `{ rhs(lhs); lhs }`

параметры

л.ш.	шк
Значение или заполнитель magrittr.	Вызов функции с использованием семантики magrittr

замечания

Пакеты, которые используют `%>%`

Оператор трубы определен в пакете `magrittr` , но он приобрел огромную видимость и популярность в пакете `dplyr` (который импортирует определение из `magrittr`). Теперь это часть `tidyverse` , которая представляет собой набор пакетов, которые *«работают в гармонии, потому что они имеют общие представления данных и дизайн API»* .

Пакет `magrittr` также предоставляет несколько вариантов оператора труб для тех, кто хочет больше гибкости в трубопроводах, например, составной соединительный трубопровод `%<>%` , трубопровод экспозиции `%$%` и оператор тройника `%T>%` . Он также предоставляет набор функций псевдонимов для замены общих функций, которые имеют специальный синтаксис (`+` , `[` , `[[` и т. Д.]), Чтобы их можно было легко использовать в цепочке труб.

Поиск документации

Как и в любом *инфиксном операторе* (например, `+` , `*` , `^` , `&` , `%in%`), вы можете найти официальную документацию, если вы поместите ее в кавычки: `?'%>%'` Или `help('%>%')` (если вы загрузили пакет, который прикрепляет `pkg:magrittr`).

Клавиатурный

В RStudio для оператора трубы есть специальная горячая клавиша: `Ctrl+Shift+M` (*Windows и Linux*), `Cmd+Shift+M` (*Mac*).

Рекомендации по производительности

В то время как оператор трубы полезен, имейте в виду, что это негативно сказывается на производительности, в основном из-за накладных расходов на его использование. При использовании оператора трубы внимательно рассмотрите следующие две вещи:

- Производительность станка (петли)
- Оценка (`object %>% rm()` не удаляет `object`)

Examples

Основное использование и цепочки

Оператор трубы, `%>%` , используется для вставки аргумента в функцию. Он не является базовым признаком языка и может использоваться только после присоединения пакета, который его предоставляет, например, `magrittr` . Оператор трубы берет левую сторону (LHS) трубы и использует ее в качестве первого аргумента функции справа (RHS) трубы. Например:

```
library(magrittr)

1:10 %>% mean
# [1] 5.5

# is equivalent to
mean(1:10)
# [1] 5.5
```

Труба может использоваться для замены последовательности вызовов функций. Несколько труб позволяют нам читать и записывать последовательность слева направо, а не изнутри наружу. Например, предположим, что мы определили `years` как фактор, но хотим преобразовать его в числовой. Чтобы предотвратить возможную потерю информации, мы сначала конвертируем в символ, а затем в числовой:

```
years <- factor(2008:2012)

# nesting
as.numeric(as.character(years))

# piping
years %>% as.character %>% as.numeric
```

Если мы не хотим, чтобы LHS (Left Hand Side) использовался в качестве *первого* аргумента в RHS (Right Hand Side), есть временные решения, такие как именование аргументов или использование `.` для указания того, куда поступает входной канал.

```
# example with grepl
# its syntax:
# grepl(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)

# note that the `substring` result is the *2nd* argument of grepl
grepl("Wo", substring("Hello World", 7, 11))

# piping while naming other arguments
"Hello World" %>% substring(7, 11) %>% grepl(pattern = "Wo")

# piping with .
"Hello World" %>% substring(7, 11) %>% grepl("Wo", .)

# piping with . and curly braces
"Hello World" %>% substring(7, 11) %>% { c(paste('Hi', .)) }
#[1] "Hi World"

#using LHS multiple times in argument with curly braces and .
"Hello World" %>% substring(7, 11) %>% { c(paste(. ,'Hi', .)) }
#[1] "World Hi World"
```

Функциональные последовательности

Учитывая последовательность шагов, которые мы используем многократно, часто бывает удобно хранить ее в функции. Трубы позволяют сохранить такие функции в читаемом формате, запустив последовательность с точкой, как в:

```
. %>% RHS
```

В качестве примера предположим, что у нас есть даты факторов и хотим извлечь год:

```
library(magrittr) # needed to include the pipe operators
library(lubridate)
read_year <- . %>% as.character %>% as.Date %>% year
```

```
# Creating a dataset
df <- data.frame(now = "2015-11-11", before = "2012-01-01")
#          now     before
# 1 2015-11-11 2012-01-01

# Example 1: applying `read_year` to a single character-vector
df$now %>% read_year
# [1] 2015

# Example 2: applying `read_year` to all columns of `df`
df %>% lapply(read_year) %>% as.data.frame  # implicit `lapply(df, read_year)
#    now before
# 1 2015   2012

# Example 3: same as above using `mutate_all`
library(dplyr)
df %>% mutate_all(funs(read_year))
# if an older version of dplyr use `mutate_each`
#    now before
# 1 2015   2012
```

Мы можем просмотреть состав функции, набрав ее имя или используя `functions` :

```
read_year
# Functional sequence with the following components:
#
#  1. as.character(.)
#  2. as.Date(.)
#  3. year(.)
#
# Use 'functions' to extract the individual functions.
```

Мы также можем получить доступ к каждой функции по ее положению в последовательности:

```
read_year[[2]]
# function (.)
# as.Date(.)
```

Как правило, такой подход может быть полезен, когда ясность важнее скорости.

Присвоение% <>%

Пакет `magrittr` содержит инфикс-оператор составного назначения, `%<>%` , который обновляет значение, сначала передавая его в одно или несколько выражений `rhs` а затем назначая результат. Это устраняет необходимость дважды вводить имя объекта (один раз с каждой стороны оператора присваивания `<-`). `%<>%` должен быть первым инфикс-оператором в цепочке:

```
library(magrittr)
library(dplyr)
```

```
df <- mtcars
```

Вместо написания

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

или же

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

Совокупный оператор присваивания будет как передавать, так и переназначать `df` :

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

Предоставление содержимого с% $%

Оператор трубы изложения, `%$%` , выдает имена столбцов в виде символов R в левом боковом объекте в выражение правой части. Этот оператор удобен при подключении к функциям, у которых нет аргумента `data` (в отличие, скажем, `lm`), и которые не принимают имена data.frame и столбцов в качестве аргументов (большинство основных функций dplyr).

Оператор трубы изложения `%$%` позволяет пользователю избежать разрыва конвейера при необходимости ссылаться на имена столбцов. Например, предположим, что вы хотите отфильтровать файл data.frame, а затем запустить корреляционный тест на двух столбцах с `cor.test` :

```
library(magrittr)
library(dplyr)
mtcars %>%
  filter(wt > 2) %$%
  cor.test(hp, mpg)

#>
#>  Pearson's product-moment correlation
#>
#> data:  hp and mpg
#> t = -5.9546, df = 26, p-value = 2.768e-06
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  -0.8825498 -0.5393217
#> sample estimates:
#>        cor
#> -0.7595673
```

Здесь стандартный `%>%` pipe передает файл data.frame через `filter()` , в то время как `%$%` pipe предоставляет имена столбцов `cor.test()` .

Труба экспозиции работает как труба, пригодная для работы с базовыми функциями R `with()` , и в качестве входных сигналов принимаются одни и те же объекты левой стороны.

Использование трубы с dplyr и ggplot2

Оператор `%>%` также может использоваться для вывода вывода dplyr в ggplot. Это создает единый исследовательский анализ данных (EDA), который легко настраивается. Этот метод быстрее, чем выполнение агрегирования внутри ggplot, и имеет дополнительное преимущество в предотвращении ненужных промежуточных переменных.

```
library(dplyr)
library(ggplot)


diamonds %>%
    filter(depth > 60) %>%
    group_by(cut) %>%
    summarize(mean_price = mean(price)) %>%
    ggplot(aes(x = cut, y = mean_price)) +
        geom_bar(stat = "identity")
```

Создание побочных эффектов при% T>%

Некоторые функции в R создают побочный эффект (например, сохранение, печать, печать и т. Д.) И не всегда возвращают значимое или желаемое значение.

`%T>%` (тройник оператор) позволяет пересылать значение в побочном эффекте продуцирующих функцию, сохраняя при этом оригинальном `lhs` значения нетронутым. Другими словами: оператор tee работает как `%>%` , за исключением того, что возвращаемые значения являются `lhs` , а не результатом функции / выражения `rhs` .

Пример. Создайте, создайте канал, напишите и верните объект. Если `%>%` были использованы вместо `%T>%` в этом примере, то переменная `all_letters` будет содержать `NULL` , а не значение отсортированного объекта.

```
all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    write.csv(file = "all_letters.csv")

read.csv("all_letters.csv") %>% head()
#   x
# 1 a
# 2 A
# 3 b
# 4 B
# 5 c
# 6 C
```

Предупреждение: при вызове неназванного объекта для `save()` будет создан объект с именем `.` при загрузке в рабочую область с `load()` . Однако возможно обходное решение с использованием вспомогательной функции (которая также может быть написана встроенной как анонимная функция).

```
all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    save(file = "all_letters.RData")

load("all_letters.RData", e <- new.env())

get("all_letters", envir = e)
# Error in get("all_letters", envir = e) : object 'all_letters' not found

get(".", envir = e)
#  [1] "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" "f" "F" "g" "G" "h" "H" "i" "I" "j" "J"
# [21] "k" "K" "l" "L" "m" "M" "n" "N" "o" "O" "p" "P" "q" "Q" "r" "R" "s" "S" "t" "T"
# [41] "u" "U" "v" "V" "w" "W" "x" "X" "y" "Y" "z" "Z"

# Work-around
save2 <- function(. = ., name, file = stop("'file' must be specified")) {
  assign(name, .)
  call_save <- call("save", ... = name, file = file)
  eval(call_save)
}

all_letters <- c(letters, LETTERS) %>%
    sort %T>%
    save2("all_letters", "all_letters.RData")
```

Прочитайте Операторы труб (%>% и другие) онлайн: https://riptutorial.com/ru/r/topic/652/операторы-труб----gt---и-другие-

глава 80: Основание

параметры

параметр	подробности
`x`	переменная оси x. Может поставлять либо `data$variablex` либо `data[,x]`
`y`	переменная оси y. Может предоставлять `data$variabley` или `data[,y]`
`main`	Основное название участка
`sub`	Дополнительный субтитр графика
`xlab`	Ярлык для оси x
`ylab`	Обозначение для оси y
`pch`	Целочисленный символ или символ, обозначающий символ построения
`col`	Целое или строковое обозначение цвета
`type`	Тип участка. `"p"` для точек, `"l"` для строк, `"b"` для обоих, `"c"` для одной только линии `"b"` , `"o"` как для «overplotted», `"h"` для «гистограммы» (например, или «высокая плотность») вертикальные линии, `"s"` для ступеней лестницы, `"S"` для других шагов, `"n"` для без заграждения

замечания

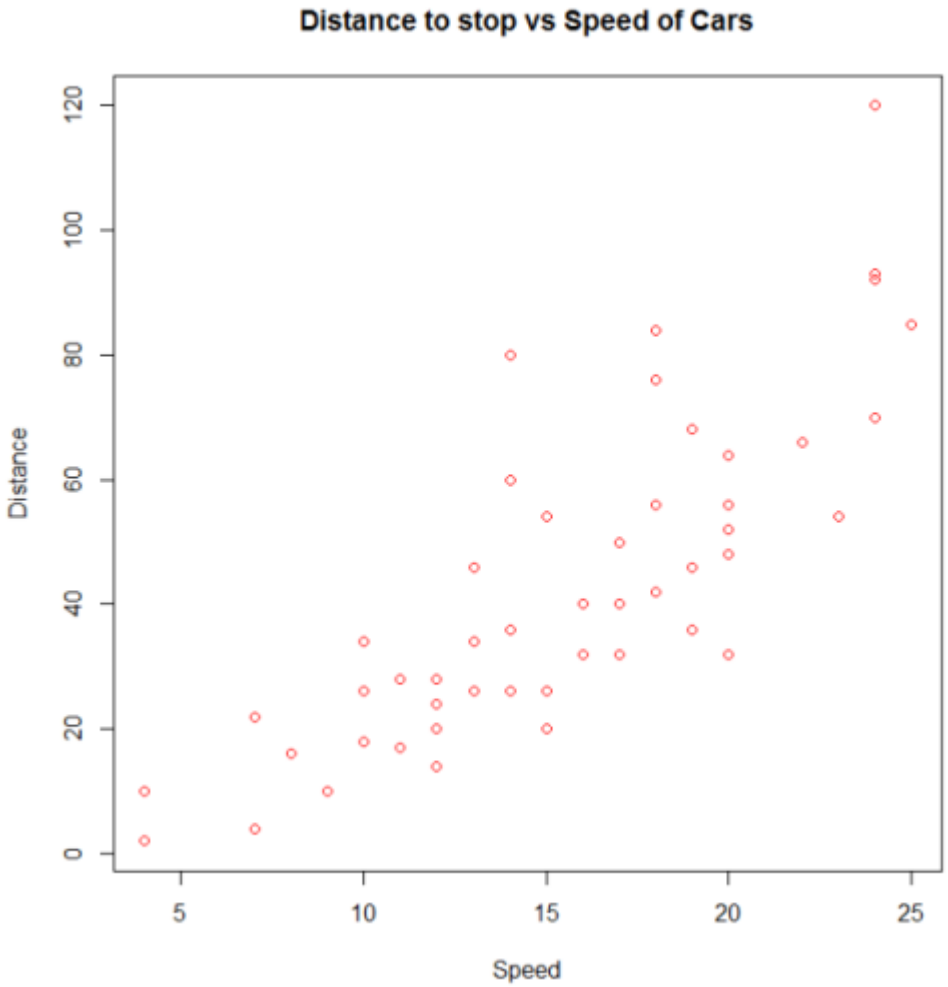
Элементы, перечисленные в разделе «Параметры», представляют собой небольшую часть hte возможных параметров, которые могут быть изменены или заданы с помощью `par` функции. См `par` для более полного списка. Кроме того, все графические устройства, включая системные интерактивные графические устройства, будут иметь набор параметров, которые могут настраивать вывод.

Examples

Основной участок

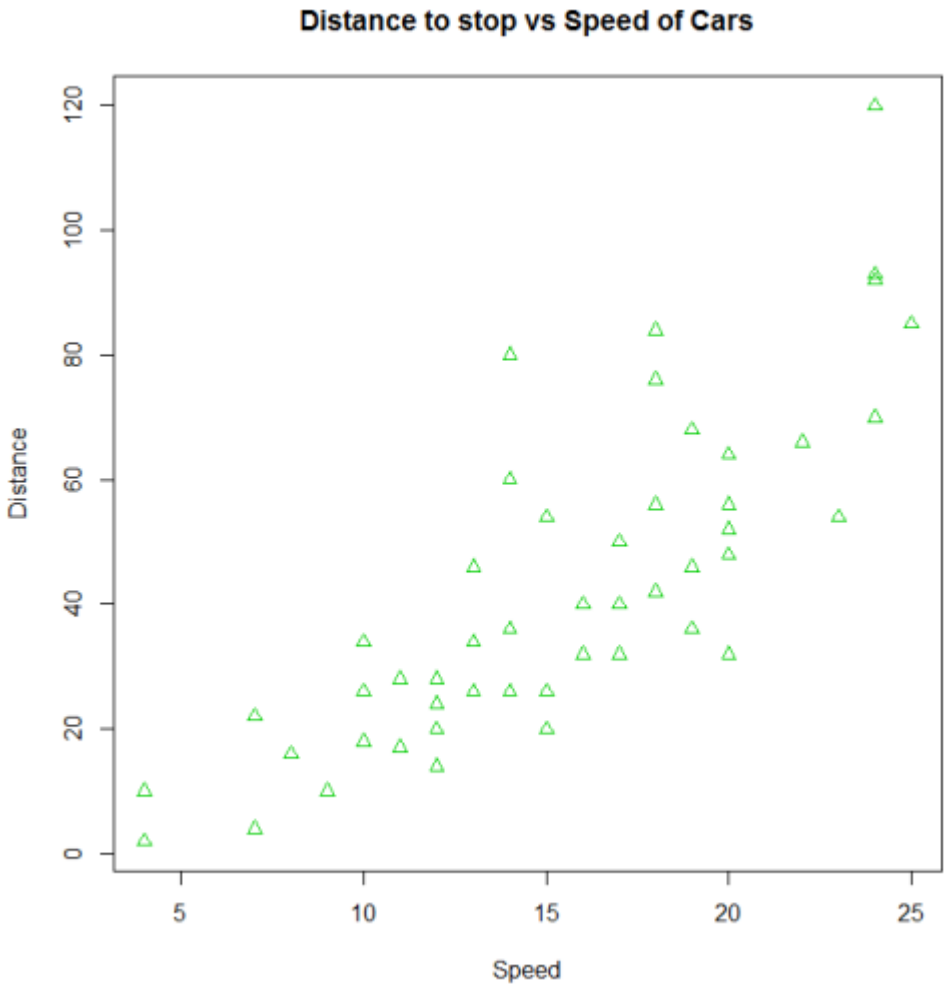
Основной сюжет создается вызовом `plot()` . Здесь мы используем встроенный кадр данных о `cars` который содержит скорость автомобилей и расстояния, снятые для остановки в 1920-х годах. (Чтобы узнать больше о наборе данных, используйте справку (автомобили)).

```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,
     main = "Distance vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
```



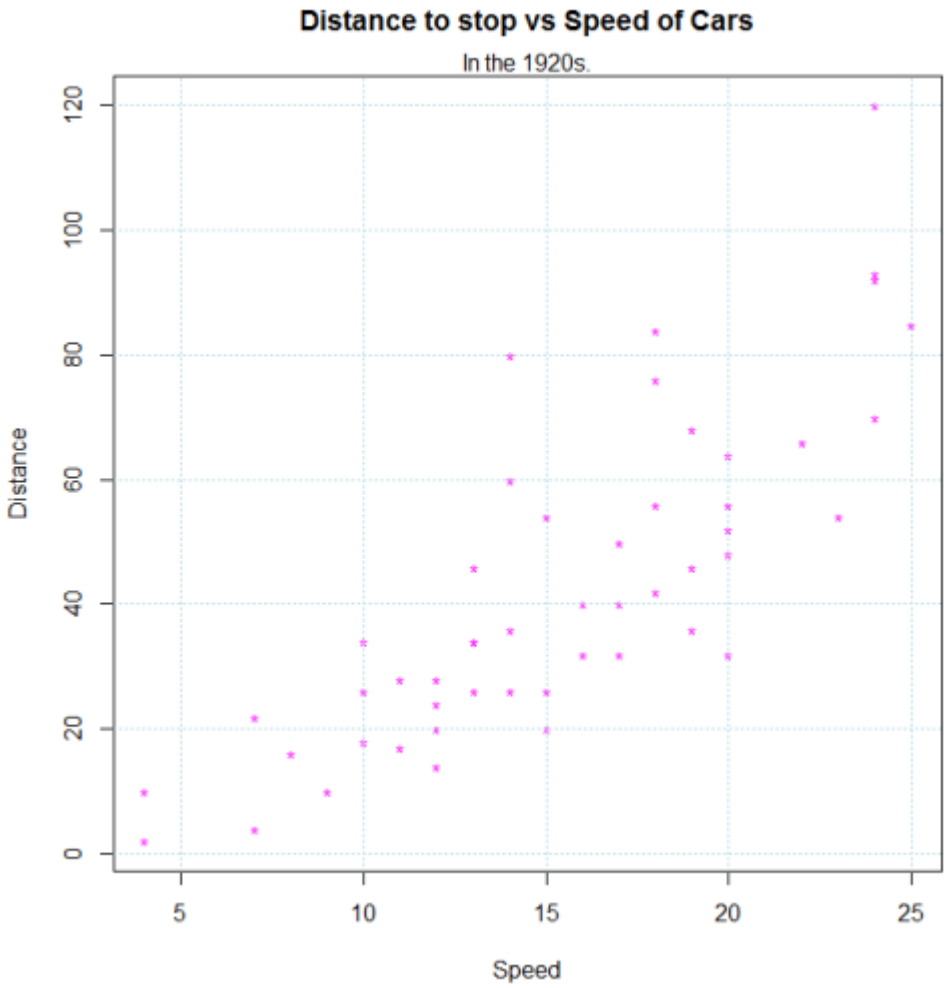
Мы можем использовать многие другие варианты кода, чтобы получить тот же результат. Мы также можем изменить параметры для получения разных результатов.

```
with(cars, plot(dist~speed, pch = 2, col = 3,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance"))
```



Дополнительные возможности могут быть добавлены к этому графику, вызывая `points()` , `text()` , `mtext()` , `lines()` , `grid()` и т. Д.

```
plot(dist~speed, pch = "*", col = "magenta", data=cars,
     main = "Distance to stop vs Speed of Cars",
     xlab = "Speed", ylab = "Distance")
mtext("In the 1920s.")
grid(,col="lightblue")
```



Matplot

`matplot` полезен для быстрой `matplot` нескольких наборов наблюдений от одного и того же объекта, особенно из матрицы, на том же графике.

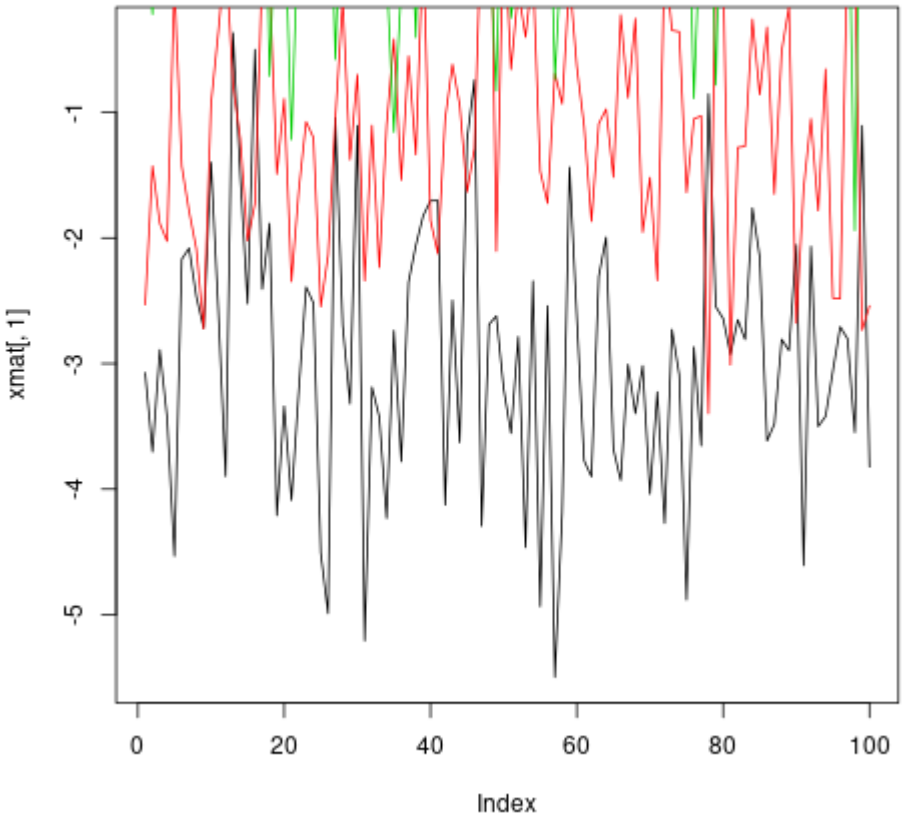
Вот пример матрицы, содержащей четыре набора случайных ничьих, каждая из которых имеет другое среднее значение.

```
xmat <- cbind(rnorm(100, -3), rnorm(100, -1), rnorm(100, 1), rnorm(100, 3))
head(xmat)
#           [,1]        [,2]       [,3]     [,4]
# [1,] -3.072793 -2.53111494  0.6168063 3.780465
# [2,] -3.702545 -1.42789347 -0.2197196 2.478416
# [3,] -2.890698 -1.88476126  1.9586467 5.268474
# [4,] -3.431133 -2.02626870  1.1153643 3.170689
# [5,] -4.532925  0.02164187  0.9783948 3.162121
# [6,] -2.169391 -1.42699116  0.3214854 4.480305
```

Один из способов построения всех этих наблюдений на одном и том же графике состоит в том, чтобы сделать один вызов `plot` за которым следуют еще три `points` или `lines` .

```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
```

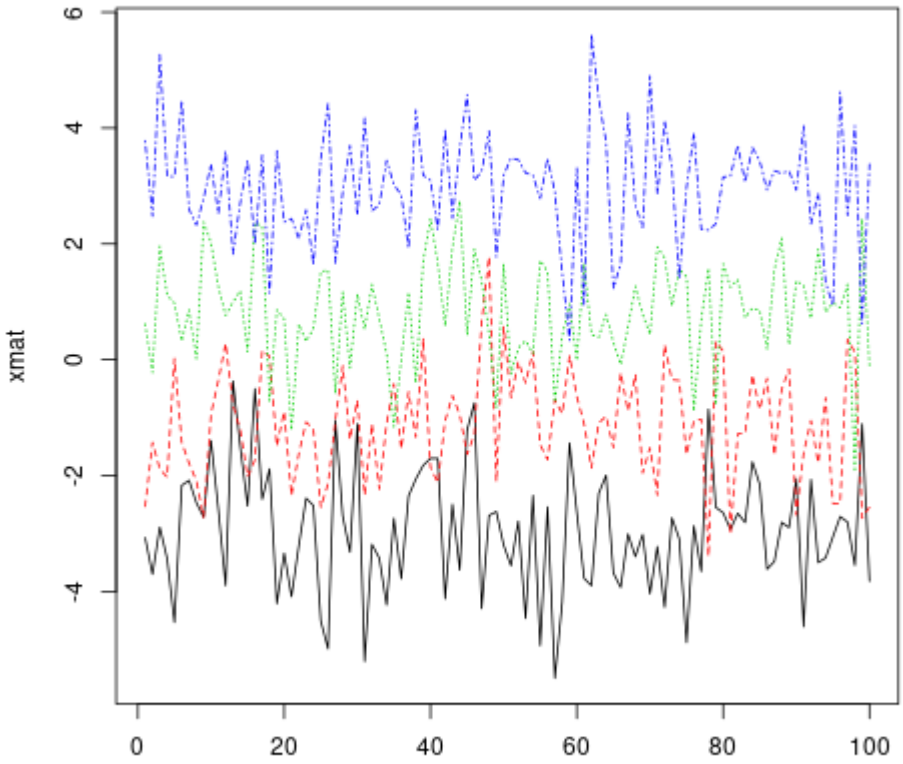
```
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```



Однако это и утомительно, и вызывает проблемы, потому что, между прочим, по умолчанию пределы оси фиксируются `plot` чтобы соответствовать только первому столбцу.

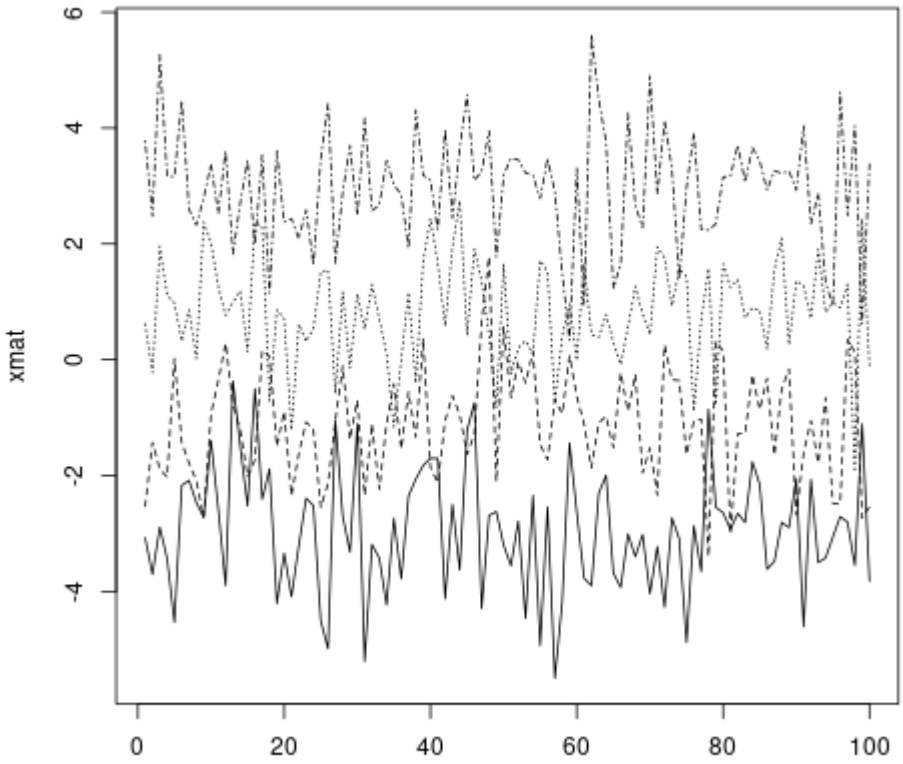
Гораздо удобнее в этой ситуации использовать функцию `matplot` , которая требует только одного вызова и автоматически заботится о границах оси *и* изменяет эстетику для каждого столбца, чтобы сделать их различимыми.

```
matplot(xmat, type = 'l')
```



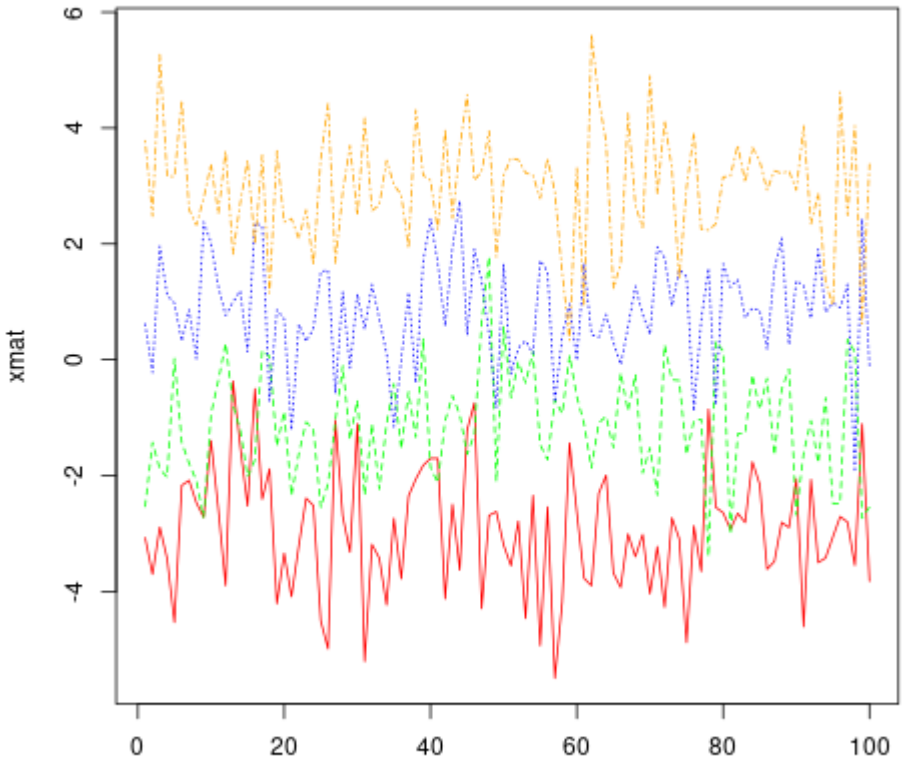
Обратите внимание, что по умолчанию `matplot` варьируется как цвет (`col`), так и `lty` линии (`lty`), потому что это увеличивает количество возможных комбинаций до их повторения. Однако любая (или обе) из этих эстетик могут быть привязаны к одному значению ...

```
matplot(xmat, type = 'l', col = 'black')
```



... или пользовательский вектор (который будет перерабатываться в число столбцов, следуя стандартным правилам утилизации векторных векторов).

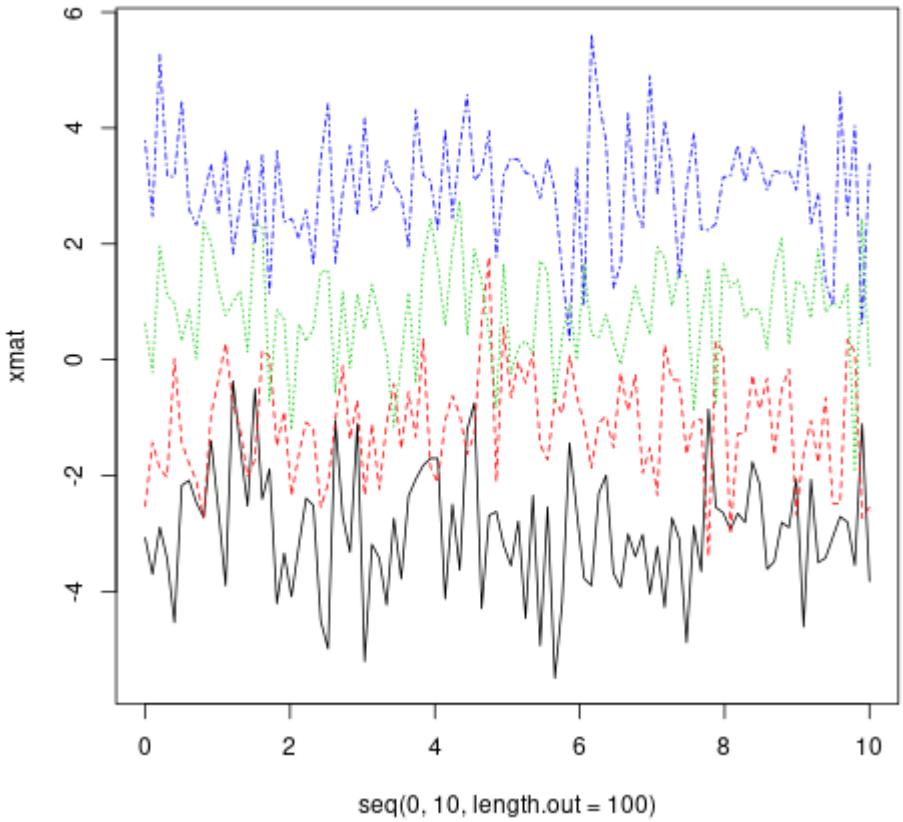
```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



Стандартные графические параметры, включая `main` , `xlab` , `xmin` , работают точно так же, как и для `plot` . Подробнее об этом см. `?par`

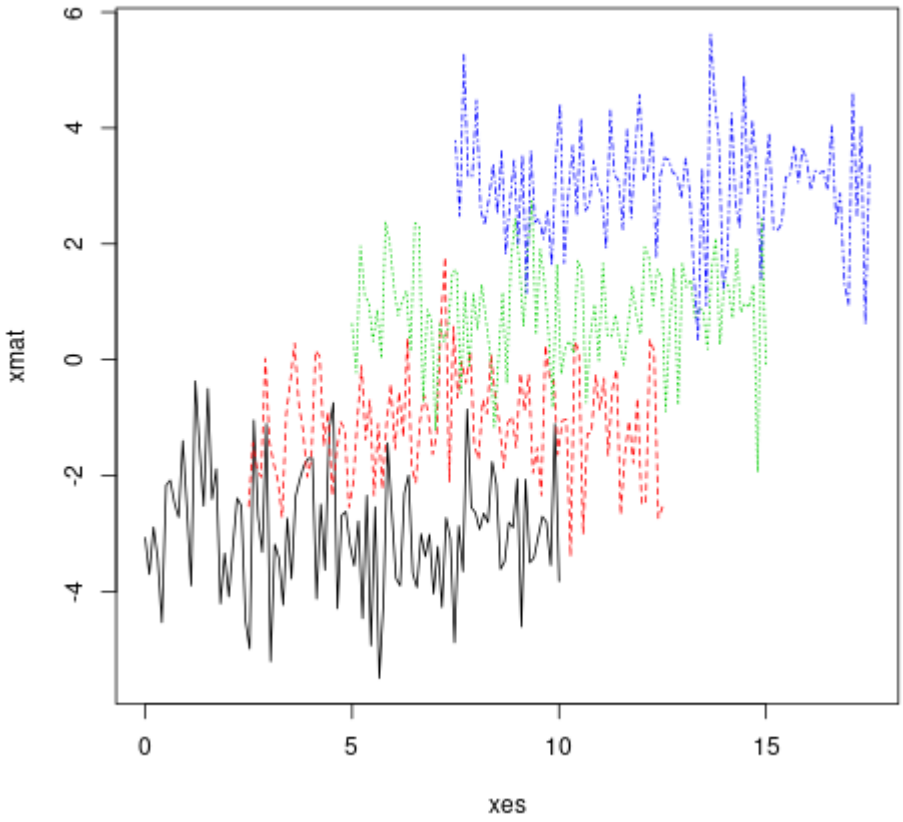
Подобно `plot` , если задан только один объект, `matplot` предполагает, что он является переменной `y` и использует индексы для `x` . Однако `x` и `y` могут быть указаны явно.

```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



Фактически, и `x` и `y` могут быть матрицами.

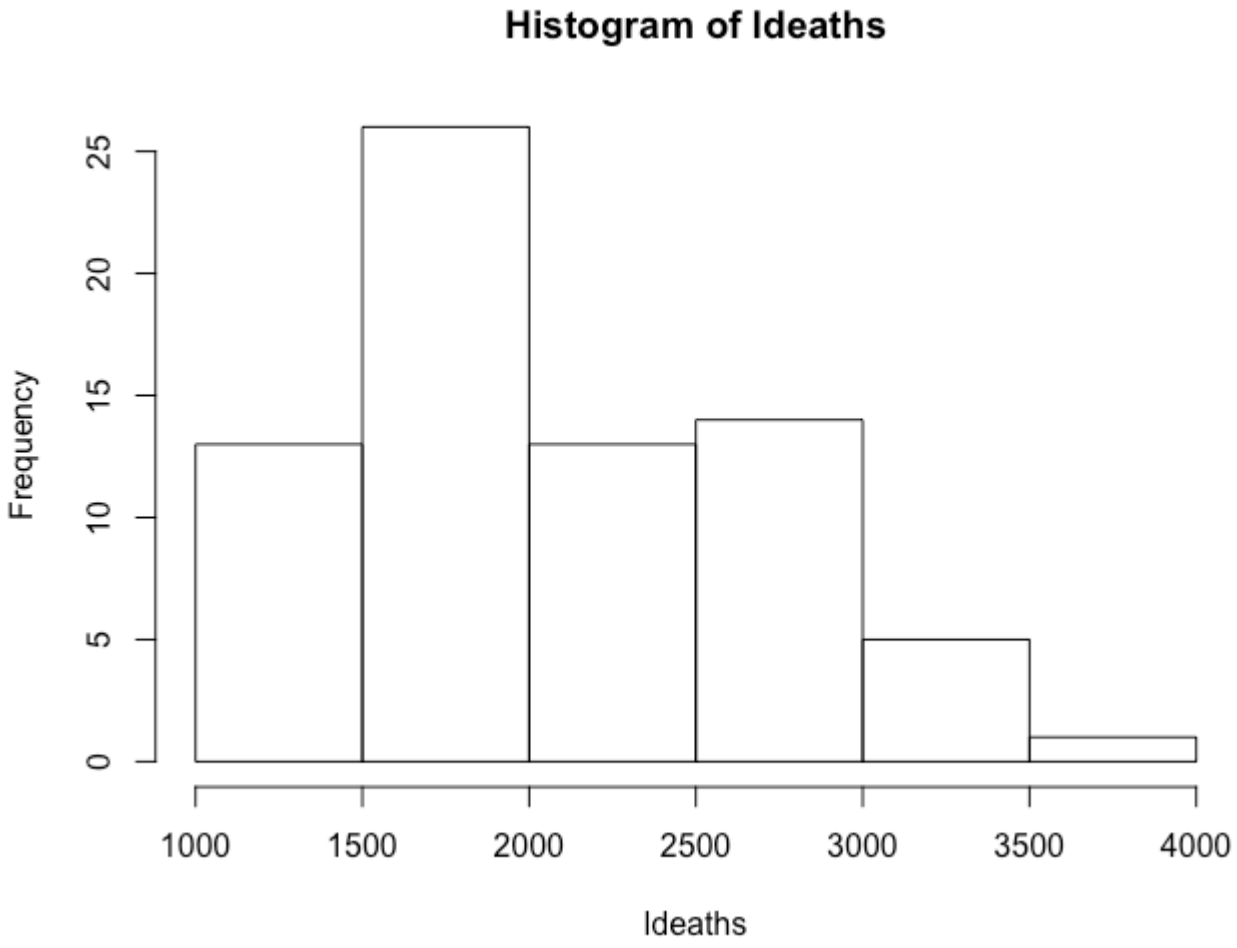
```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```



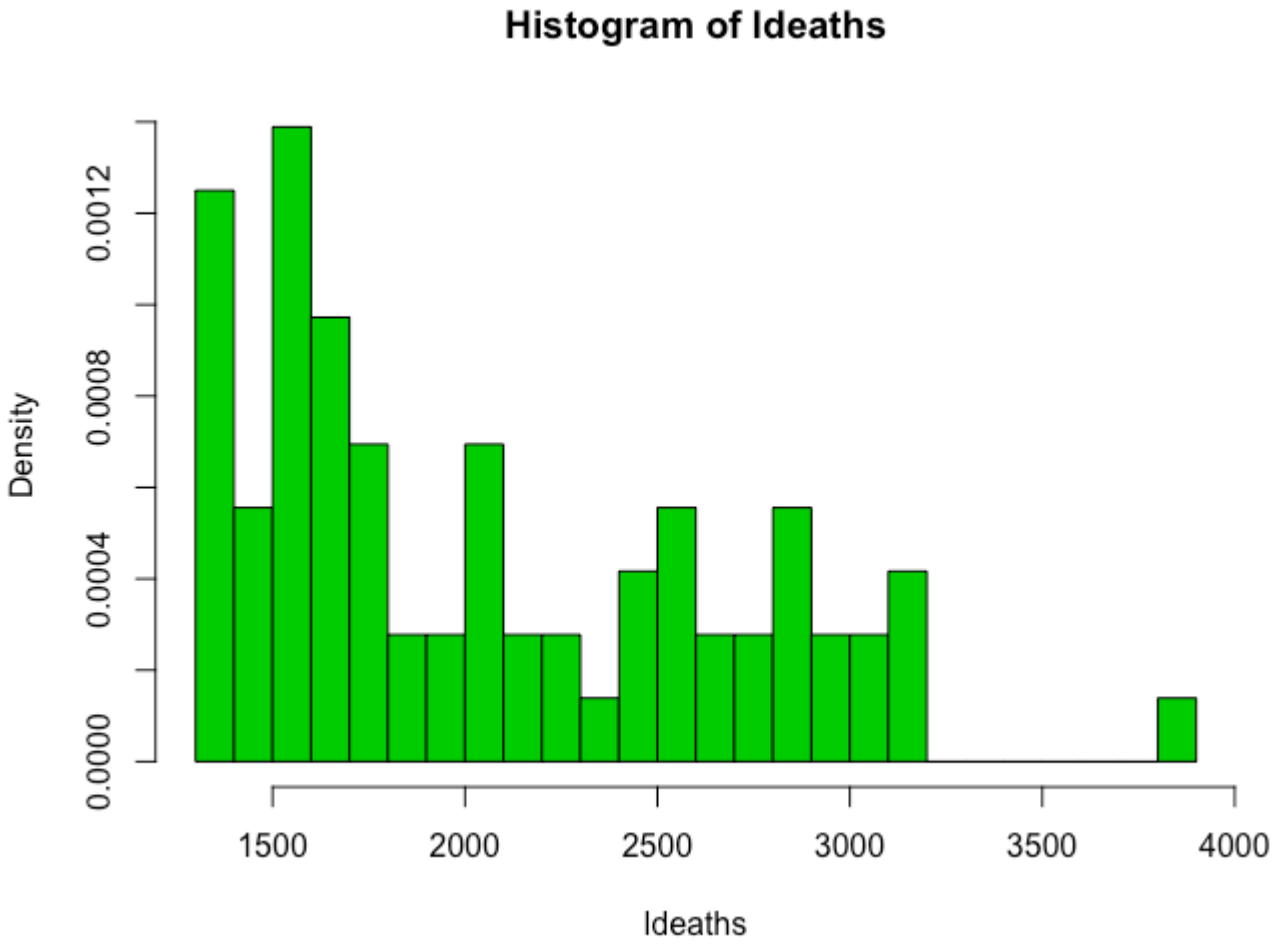
Гистограммы

Гистограммы позволяют использовать псевдо-график лежащего в основе распределения данных.

```
hist(ldeaths)
```




```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```



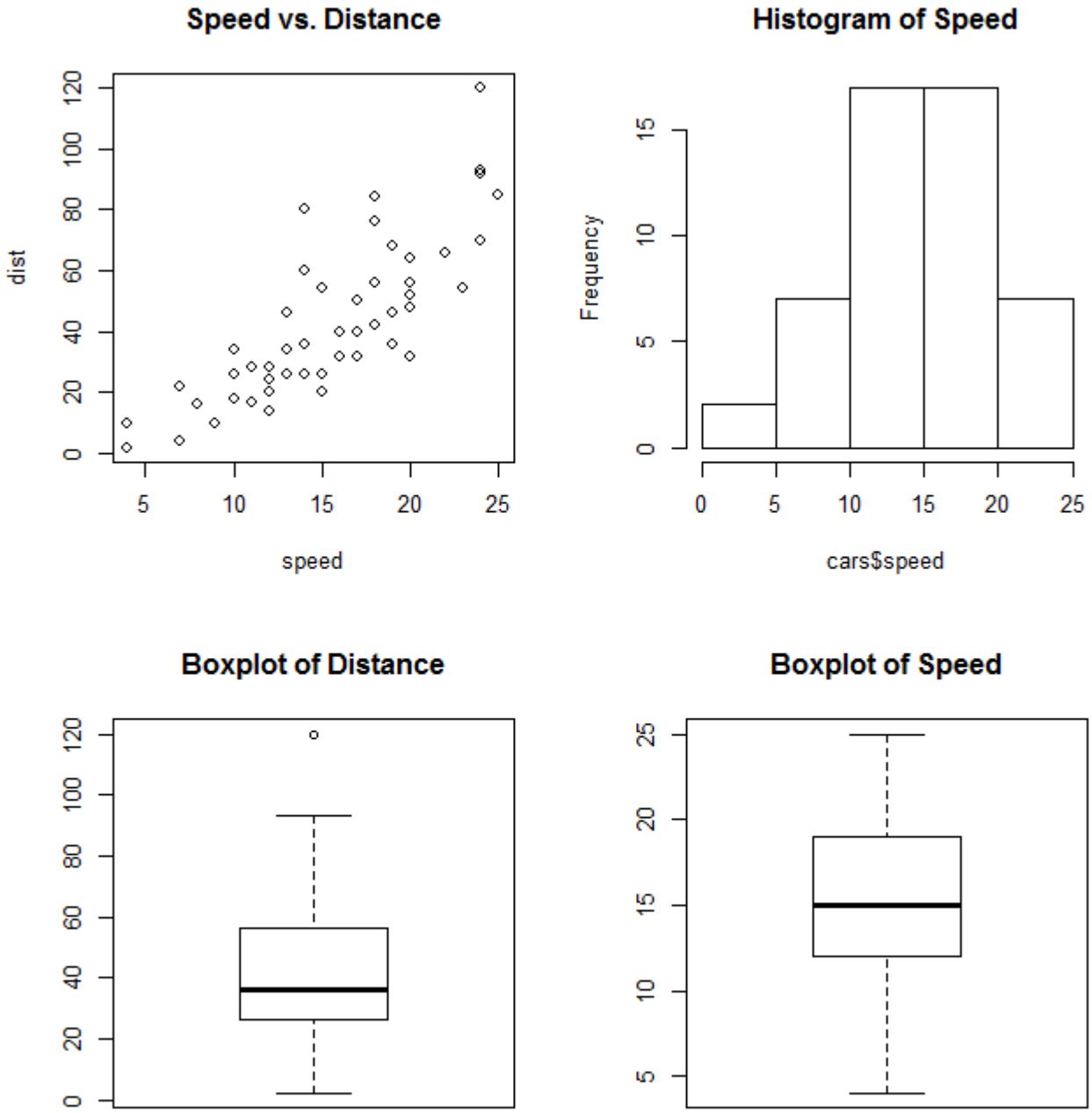
Объединение сюжетов

Часто полезно комбинировать несколько типов графиков в одном графике (например, Barplot рядом с Scatterplot.) R делает это легко с помощью функций `par()` и `layout()` .

`par()`

`par` использует аргументы `mfrow` или `mfcol` для создания матрицы nrows и ncols `c(nrows, ncols)` которая будет служить сеткой для ваших графиков. В следующем примере показано, как объединить четыре графика в одном графике:

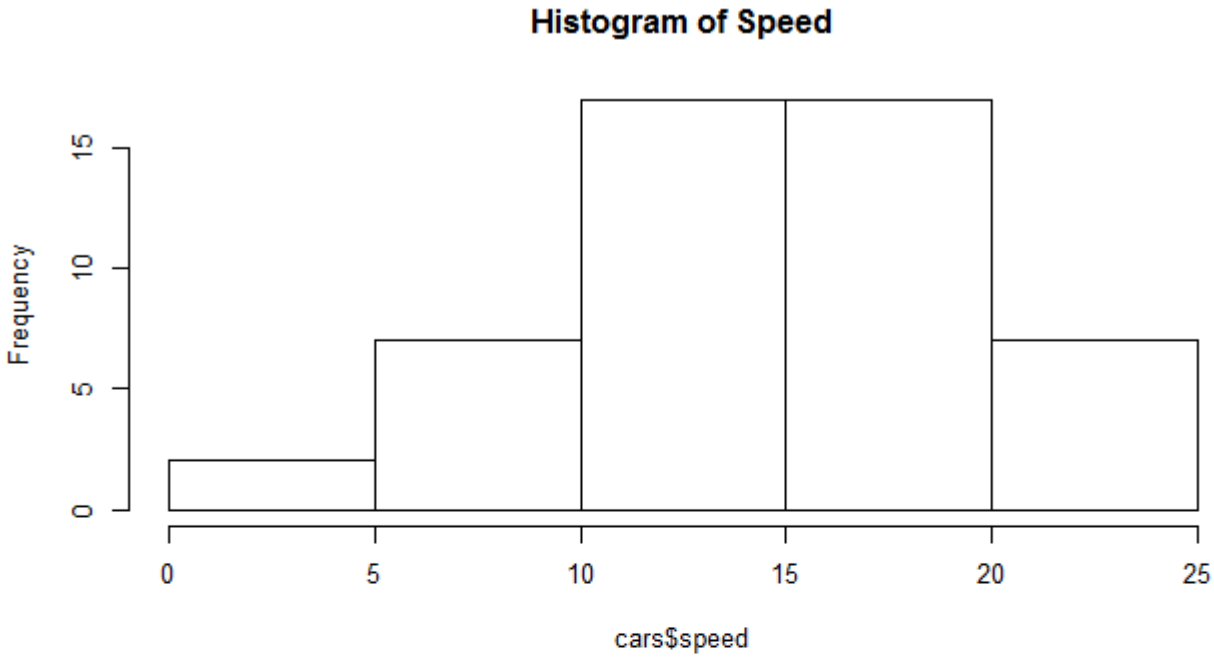
```
par(mfrow=c(2,2))
plot(cars, main="Speed vs. Distance")
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```

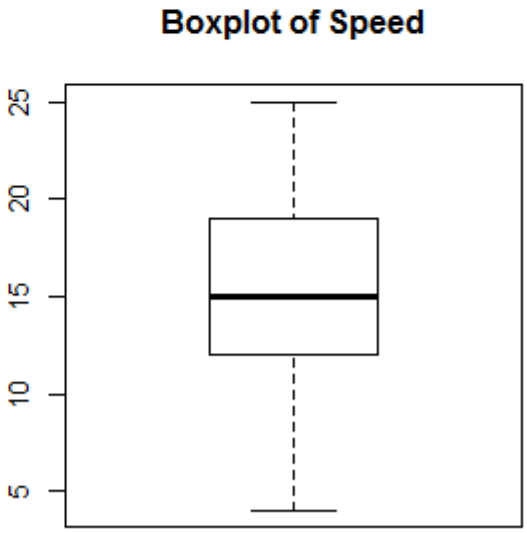


layout()

`layout()` более гибкий и позволяет указать местоположение и размер каждого графика в конечном комбинированном графике. Эта функция ожидает матричный объект как вход:

```
layout(matrix(c(1,1,2,3), 2,2, byrow=T))
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```



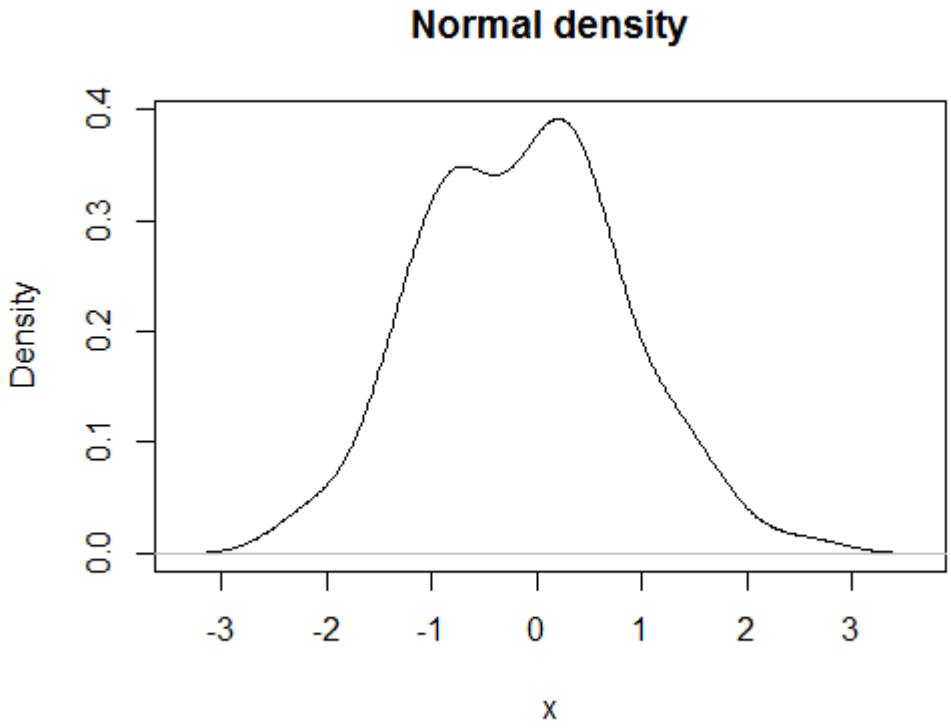


Плотность

Очень полезным и логичным продолжением гистограмм было бы построение сглаженной функции плотности случайной величины. Основной сюжет, созданный командой

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

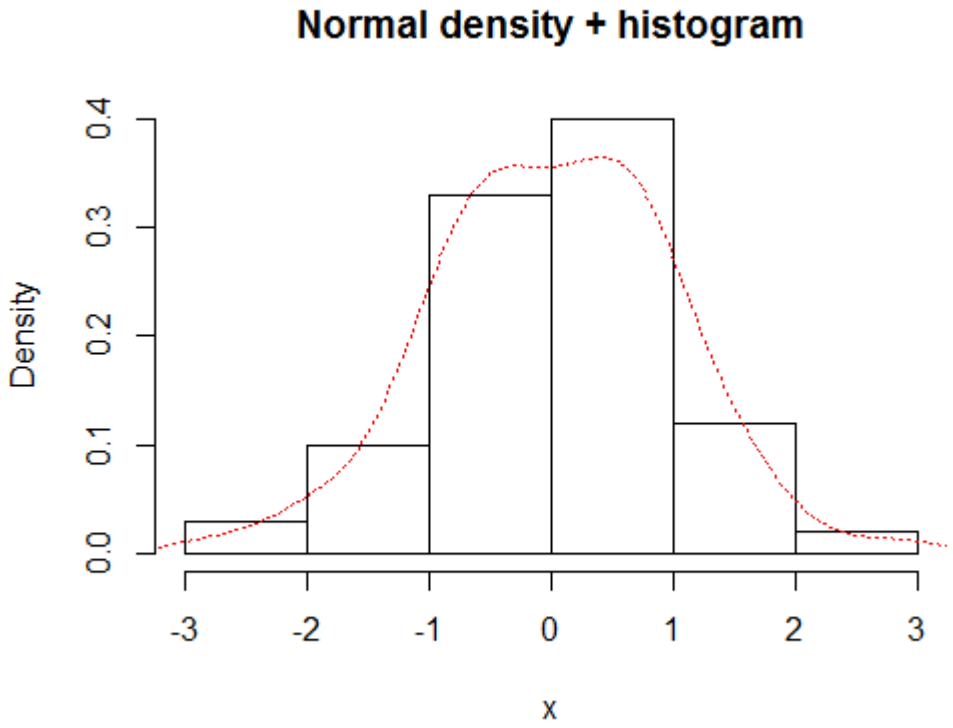
будет выглядеть



Вы можете наложить гистограмму и кривую плотности с помощью

```
x=rnorm(100)
hist(x,prob=TRUE,main="Normal density + histogram")
lines(density(x),lty="dotted",col="red")
```

который дает

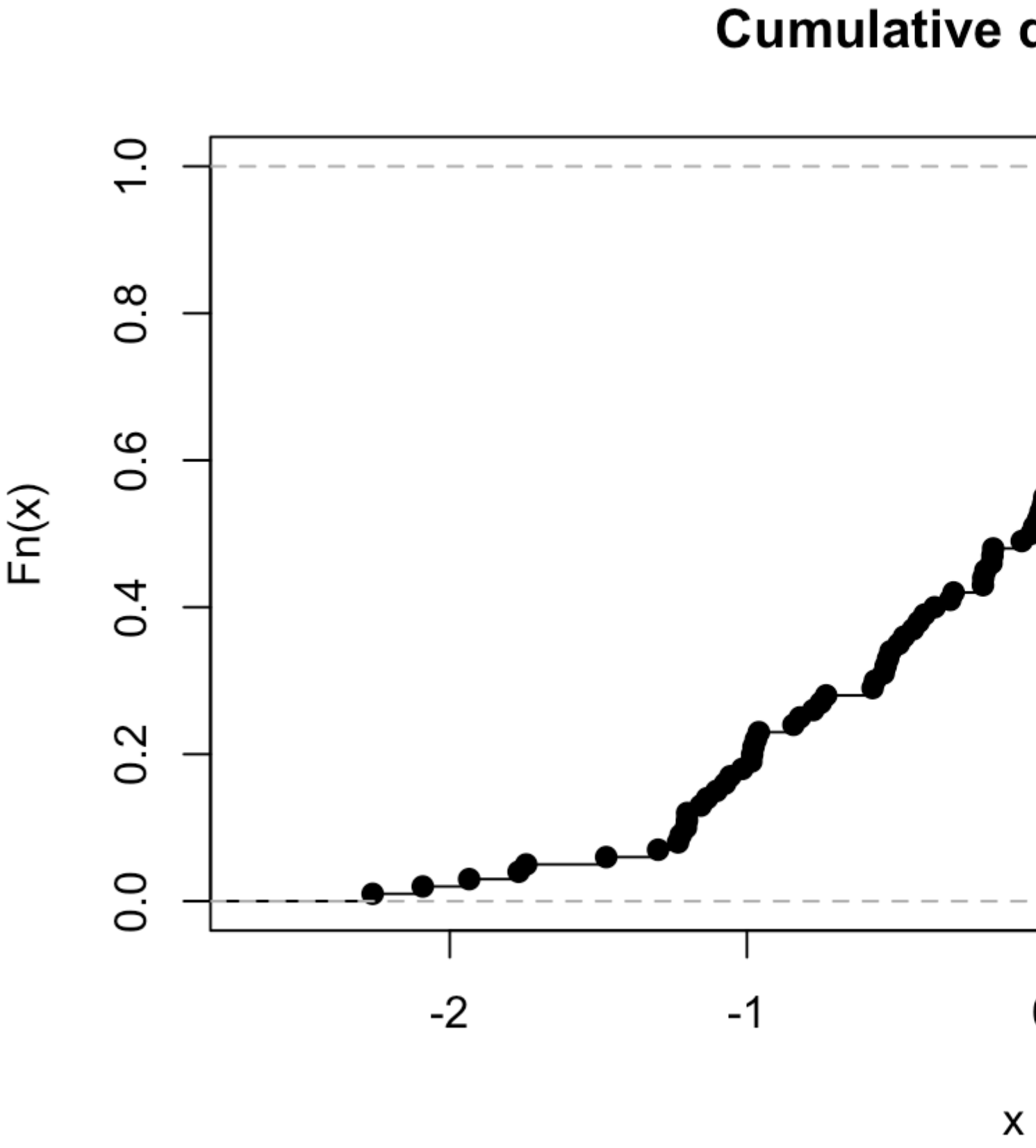


Эмпирическая функция кумулятивного распределения

Очень полезным и логичным продолжением гистограмм и графиков плотности будет Эмпирическая функция кумулятивного распределения. Для этой цели мы можем использовать функцию `ecdf()` . Основной сюжет, созданный командой

```
plot(ecdf(rnorm(100)),main="Cumulative distribution",xlab="x")
```

будет выглядеть



Начало работы с R_Plots

- **разброс** точек

У вас есть два вектора, и вы хотите их построить.

```
x_values <- rnorm(n = 20 , mean = 5 , sd = 8) #20 values generated from Normal(5,8)
y_values <- rbeta(n = 20 , shape1 = 500 , shape2 = 10) #20 values generated from Beta(500,10)
```

Если вы хотите создать график, который имеет `y_values` по вертикальной оси и `x_values` по горизонтальной оси, вы можете использовать следующие команды:

```
plot(x = x_values, y = y_values, type = "p") #standard scatter-plot
plot(x = x_values, y = y_values, type = "l") # plot with lines
plot(x = x_values, y = y_values, type = "n") # empty plot
```

Вы можете набрать « `?plot()` в консоли, чтобы узнать о дополнительных параметрах.

- **Boxplot**

У вас есть некоторые переменные, и вы хотите изучить их распределения

```
#boxplot is an easy way to see if we have some outliers in the data.

z<- rbeta(20 , 500 , 10) #generating values from beta distribution
z[c(19 , 20)] <- c(0.97 , 1.05) # replace the two last values with outliers
boxplot(z) # the two points are the outliers of variable z.
```

- **Гистограммы**

Легкий способ рисования гистограмм

```
hist(x = x_values) # Histogram for x vector
hist(x = x_values, breaks = 3) #use breaks to set the numbers of bars you want
```

- **Круговые диаграммы**

Если вы хотите визуализировать частоты переменной, просто нарисуйте пирог

Сначала мы должны генерировать данные с частотами, например:

```
P <- c(rep('A' , 3) , rep('B' , 10) , rep('C' , 7) )
t <- table(P) # this is a frequency matrix of variable P
pie(t) # And this is a visual version of the matrix above
```

Прочитайте Основание онлайн: https://riptutorial.com/ru/r/topic/1377/основание

глава 81: Отказоустойчивый / устойчивый код

параметры

параметр	подробности
выраж	В случае успешного завершения «try part» `tryCatch` вернет **последнее оцениваемое выражение** . Следовательно, фактическое значение возвращается в случае, если все прошло хорошо, и нет условия (т.е. *предупреждение* или *ошибка*) - это возвращаемое значение `readLines` . Обратите внимание, что вам не нужно объяснять состояние возвращаемого значения через `return` поскольку код в «try part» не завернут в оболочку функции (в отличие от обработчиков условий для предупреждений и ошибок ниже)
предупреждение / ошибка / и т.д.	Предоставить / определить функцию обработчика для всех условий, которые вы хотите обработать явно. AFAIU, вы можете предоставить обработчики для *любых* условий (не только *предупреждения* и *ошибки* , но и *пользовательские* условия, см. `simpleCondition` и друзей для этого), если **имя соответствующей функции обработчика соответствует классу соответствующего условия** (см. *Подробности* части документа для `tryCatch`).
в конце концов	Здесь идет все, что должно быть выполнено в самом конце, **независимо от того, выполнено** ли выражение в «try part» или было ли какое-либо условие. Если вы хотите выполнить более одного выражения, то вам нужно обернуть их фигурными скобками, иначе вы могли бы просто написать `finally = <expression>` (т.е. та же логика, что и для «try part».

замечания

`tryCatch`

`tryCatch` возвращает значение, связанное с выполнением `expr` если есть условие: предупреждение или ошибка. Если это так, конкретные возвращаемые значения (например, `return(NA)` выше) могут быть заданы путем предоставления функции

обработчика для соответствующих условий (см. Аргументы `warning` и `error` в `?tryCatch`). Это могут быть уже существующие функции, но вы также можете определить их в `tryCatch` (как мы это делали выше).

Последствия выбора конкретных возвращаемых значений функций обработчика

Поскольку мы указали, что `NA` должно быть возвращено в случае ошибки в «try part», третьим элементом в `y` является `NA` . Если бы мы выбрали `NULL` как возвращаемое значение, длина `y` бы равна `2` вместо `3` как `lapply` будет просто «игнорировать / отбрасывать» возвращаемые значения, которые являются `NULL` . Также обратите внимание, что если вы не укажете **явное** возвращаемое значение через `return` , функции обработчика вернут `NULL` (т.е. в случае *ошибки* или условия *предупреждения*).

Предупреждение "Нежелательное"

Когда третий элемент нашего вектора `urls` попадает в нашу функцию, мы получаем следующее предупреждение **в дополнение** к тому, что возникает ошибка (`readLines` сначала жалуется, что он не может открыть соединение с помощью *предупреждения* до фактического сбоя с *ошибкой*):

```
Warning message:
    In file(con, "r") : cannot open file 'I'm no URL': No such file or directory
```

Ошибка «выигрывает» над *предупреждением* , поэтому мы действительно не заинтересованы в предупреждении в этом конкретном случае. Таким образом, мы установили `warn = FALSE` в `readLines` , но это, похоже, не имеет никакого эффекта. Альтернативный способ подавления предупреждения - использовать

```
suppressWarnings(readLines(con = url))
```

вместо

```
readLines(con = url, warn = FALSE)
```

Examples

Использование tryCatch ()

Мы определяем надежную версию функции, которая считывает код HTML с заданного URL-адреса. *Надежный* в том смысле, что мы хотим, чтобы он справлялся с ситуациями, когда что-то либо ошибочно (ошибка), либо не совсем так, как мы планировали это (предупреждение). Зонный термин для ошибок и предупреждений является *условием*

Определение функции с помощью `tryCatch`

```
readUrl <- function(url) {
    out <- tryCatch(

        ########################################################
        # Try part: define the expression(s) you want to "try" #
        ########################################################

        {
            # Just to highlight:
            # If you want to use more than one R expression in the "try part"
            # then you'll have to use curly brackets.
            # Otherwise, just write the single expression you want to try and

            message("This is the 'try' part")
            readLines(con = url, warn = FALSE)
        },

        ########################################################################
        # Condition handler part: define how you want conditions to be handled #
        ########################################################################

        # Handler when a warning occurs:
        warning = function(cond) {
            message(paste("Reading the URL caused a warning:", url))
            message("Here's the original warning message:")
            message(cond)

            # Choose a return value when such a type of condition occurs
            return(NULL)
        },

        # Handler when an error occurs:
        error = function(cond) {
            message(paste("This seems to be an invalid URL:", url))
            message("Here's the original error message:")
            message(cond)

            # Choose a return value when such a type of condition occurs
            return(NA)
        },

        ################################################
        # Final part: define what should happen AFTER #
        # everything has been tried and/or handled    #
        ################################################

        finally = {
            message(paste("Processed URL:", url))
            message("Some message at the end\n")
        }
    )
    return(out)
}
```

Тестирование

Давайте определим вектор URL-адресов, где один элемент не является допустимым URL-адресом

```
urls <- c(
    "http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html",
    "http://en.wikipedia.org/wiki/Xz",
    "I'm no URL"
)
```

И передайте это как вход в функцию, определенную выше

```
y <- lapply(urls, readUrl)
# Processed URL: http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html
# Some message at the end
#
# Processed URL: http://en.wikipedia.org/wiki/Xz
# Some message at the end
#
# URL does not seem to exist: I'm no URL
# Here's the original error message:
# cannot open the connection
# Processed URL: I'm no URL
# Some message at the end
#
# Warning message:
# In file(con, "r") : cannot open file 'I'm no URL': No such file or directory
```

Исследование выхода

```
length(y)
# [1] 3

head(y[[1]])
# [1] "<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01 Transitional//EN\">"
# [2] "<html><head><title>R: Functions to Manipulate Connections</title>"
# [3] "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">"
# [4] "<link rel=\"stylesheet\" type=\"text/css\" href=\"R.css\">"
# [5] "</head><body>"
# [6] ""

y[[3]]
# [1] NA
```

Прочитайте Отказоустойчивый / устойчивый код онлайн: https://riptutorial.com/ru/r/topic/4060/отказоустойчивый---устойчивый-код

глава 82: отладка

Examples

Использование браузера

Функция `browser` может использоваться как точка останова: выполнение кода приостанавливается в том месте, которое вызывается. Затем пользователь может затем проверить значения переменных, выполнить произвольный R-код и пройти через код по строкам.

Как только `browser()` попадет в код, начнется интерактивный интерпретатор. Любой R-код можно запускать как обычно, и, кроме того, присутствуют следующие команды,

команда	Имея в виду
c	Выйти из браузера и продолжить программу
e	Завершить текущий цикл или функцию \
N	Step Over (оценить следующий оператор, перейдя через вызовы функций)
s	Step Into (оценивать следующий оператор, вступая в вызовы функций)
где	Трассировка стека печати
p	Вызов «возобновить» перезапуск
Q	Выйти из браузера и выйти

Например, у нас может быть такой скрипт,

```
toDebug <- function() {
    a = 1
    b = 2

    browser()

    for(i in 1:100) {
        a = a * b
    }
}

toDebug()
```

При запуске вышеупомянутого скрипта мы сначала видим что-то вроде:

```
Called from: toDebug
Browser[1]>
```

Затем мы могли бы взаимодействовать с подсказкой,

```
Called from: toDebug
Browser[1]> a
[1] 1
Browser[1]> b
[1] 2
Browse[1]> n
debug at #7: for (i in 1:100) {
    a = a * b
}
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 1
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 2
Browse[2]> Q
```

`browser()` также может использоваться как часть функциональной цепочки, например:

```
mtcars %>% group_by(cyl) %>% {browser()}
```

Использование отладки

Вы можете установить любую функцию для отладки с помощью `debug` .

```
debug(mean)
mean(1:3)
```

Все последующие вызовы функции войдут в режим отладки. Вы можете отключить это поведение с помощью `undebug` .

```
undebug(mean)
mean(1:3)
```

Если вы знаете, что хотите только один раз войти в режим отладки функции, подумайте об использовании `debugonce` .

```
debugonce(mean)
mean(1:3)
mean(1:3)
```

Прочитайте отладка онлайн: https://riptutorial.com/ru/r/topic/1695/отладка

глава 83: Отсутствующие значения

Вступление

Когда мы не знаем значения, которое принимает переменная, мы говорим, что ее значение отсутствует, обозначенное `NA` .

замечания

Недостающие значения представлены символом `NA` (недоступно). Невозможные значения (например, в результате `sqrt(-1)`) представлены символом `NaN` (а не числом).

Examples

Изучение недостающих данных

`anyNA` сообщает, присутствуют ли отсутствующие значения; while `is.na` сообщает пропущенные значения elementwise:

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE  TRUE FALSE
```

`is.na` возвращает логический вектор, который принуждается к целочисленным значениям при арифметических операциях (с FALSE = 0, TRUE = 1). Мы можем использовать это, чтобы узнать, сколько недостающих значений есть:

```
sum(is.na(vec))
# [1] 1
```

Расширяя этот подход, мы можем использовать `colSums` и `is.na` в кадре данных для подсчета NA для каждого столбца:

```
colSums(is.na(airquality))
#   Ozone Solar.R    Wind    Temp   Month     Day
#      37       7       0       0       0       0
```

Пакет naniar (в настоящее время на github, но не CRAN) предлагает дополнительные инструменты для изучения отсутствующих значений.

Чтение и запись данных с значениями NA

При чтении табличных наборов данных с функциями `read.*` R автоматически ищет отсутствующие значения, которые выглядят как `"NA"` . Однако отсутствующие значения не всегда представлены `NA` . Иногда точка (`.`), Дефис (`-`) или значение символа (например: `empty`) указывает, что значение равно `NA` . Параметр `na.strings` функции `read.*` Может использоваться для `na.strings` R, символы / символы которых должны рассматриваться как значения `NA` :

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

Также можно указать, что более одного символа необходимо читать как `NA` :

```
read.csv('missing.csv', na.strings = c('.','-'))
```

Аналогично, `NA` s может быть написано с помощью настраиваемых строк, используя аргумент `na` для `write.csv` . Другие инструменты для чтения и записи таблиц имеют аналогичные варианты.

Использование NA разных классов

Символ `NA` для `logical` отсутствующего значения:

```
class(NA)
#[1] "logical"
```

Это удобно, так как оно легко может быть принудительно применено к другим типам атомных векторов и поэтому обычно является единственным `NA` вам понадобится:

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

Если вам нужно одно значение `NA` другого типа, используйте `NA_character_` , `NA_integer_` , `NA_real_` или `NA_complex_` . Для отсутствующих значений причудливых классов обычно `NA_integer_` подмножество с `NA_integer_` ; например, для получения отсутствующего значения Дата:

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

TRUE / FALSE и / или NA

`NA` является логическим типом, а логический оператор с `NA` возвращает `NA` если результат неоднозначен. Ниже `NA OR TRUE` оценивает значение `TRUE` потому что по крайней мере одна сторона оценивает значение `TRUE` , но `NA OR FALSE` возвращает `NA` потому что мы не знаем, было ли `NA TRUE` или `FALSE`

```
NA | TRUE
# [1] TRUE
# TRUE | TRUE is TRUE and FALSE | TRUE is also TRUE.

NA | FALSE
# [1] NA
# TRUE | FALSE is TRUE but FALSE | FALSE is FALSE.

NA & TRUE
# [1] NA
# TRUE & TRUE is TRUE but FALSE & TRUE is FALSE.

NA & FALSE
# [1] FALSE
# TRUE & FALSE is FALSE and FALSE & FALSE is also FALSE.
```

Эти свойства полезны, если вы хотите подмножить набор данных на основе некоторых столбцов, содержащих `NA` .

```
df <- data.frame(v1=0:9,
                 v2=c(rep(1:2, each=4), NA, NA),
                 v3=c(NA, letters[2:10]))

df[df$v2 == 1 & !is.na(df$v2), ]
#  v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d

df[df$v2 == 1, ]
     v1 v2   v3
#1     0  1 <NA>
#2     1  1    b
#3     2  1    c
#4     3  1    d
#NA   NA NA <NA>
#NA.1 NA NA <NA>
```

Опускание или замена отсутствующих значений

Перекодирование отсутствующих значений

Регулярно отсутствующие данные не кодируются как `NA` в наборах данных. Например, в SPSS отсутствующие значения часто представлены значением `99` .

```
num.vec <- c(1, 2, 3, 99, 5)
num.vec
## [1]  1  2  3 99  5
```

Можно напрямую назначить NA с помощью подмножества

```
num.vec[num.vec == 99] <- NA
```

Однако предпочтительным методом является использование `is.na<-` как `is.na<-` ниже. Файл справки (`?is.na`) гласит:

> `is.na<-` может обеспечить более безопасный способ установки пропусков. Например, для разных факторов это ведет себя иначе.

```
is.na(num.vec) <- num.vec == 99
```

Оба метода возвращаются

```
num.vec
## [1]  1  2  3 NA  5
```

Удаление отсутствующих значений

Отсутствующие значения могут быть удалены несколькими способами из вектора:

```
num.vec[!is.na(num.vec)]
num.vec[complete.cases(num.vec)]
na.omit(num.vec)
## [1] 1 2 3 5
```

Исключение отсутствующих значений из вычислений

При использовании арифметических функций на векторах с отсутствующими значениями будет возвращено отсутствующее значение:

```
mean(num.vec) # returns: [1] NA
```

Параметр `na.rm` сообщает функции исключать значения `NA` из расчета:

```
mean(num.vec, na.rm = TRUE) # returns: [1] 2.75

# an alternative to using 'na.rm = TRUE':
mean(num.vec[!is.na(num.vec)]) # returns: [1] 2.75
```

Некоторые R-функции, такие как `lm` , имеют параметр `na.action` . Значение по умолчанию для этого - `na.omit` , но с `options(na.action = 'na.exclude')` поведение по умолчанию R может

быть изменено.

Если нет необходимости изменять поведение по умолчанию, но для конкретной ситуации `na.action` другое `na.action` , параметр `na.action` должен быть включен в вызов функции, например:

```
lm(y2 ~ y1, data = anscombe, na.action = 'na.exclude')
```

Прочитайте Отсутствующие значения онлайн: https://riptutorial.com/ru/r/topic/3388/отсутствующие-значения

глава 84: Очистка веб-страниц и разбор

замечания

Скребок - это использование компьютера для извлечения кода веб-страницы. Как только код будет получен, он должен быть *проанализирован* в полезную форму для дальнейшего использования в R.

База R не имеет большого количества инструментов, необходимых для этих процессов, поэтому очистка и разбор обычно выполняются с помощью пакетов. Некоторые пакеты наиболее полезны для очистки (`RSelenium` , `httr` , `curl` , `RCurl`), некоторые для синтаксического анализа (`XML` , `xml2`) и некоторые для обоих (`rvest`).

Связанный с этим процесс очищает веб-API, который, в отличие от веб-страницы, возвращает данные, предназначенные для машинного чтения. Многие из тех же пакетов используются для обоих.

легальность

Некоторые веб-сайты возражают против того, чтобы их очищали, из-за увеличения нагрузки на сервер или проблем с владением данными. Если сайт запрещает соскабливать в нем Условия использования, скремблирование является незаконным.

Examples

Основные скребки с rvest

`rvest` - это пакет для веб-соскабливания и разбора Хэдли Уикхема, вдохновленный Прекрасным `rvest` Питона. Он использует привязки `libxml2` пакета `xml2` для HTML для синтаксического анализа HTML.

Как часть tidyverse, `rvest` подается по трубопроводу . Оно использует

- `xml2::read_html` чтобы очистить HTML-страницу веб-страницы,
- который затем может быть подмножеством с его `html_node` и `html_nodes` с использованием селекторов CSS или XPath и
- обрабатываются объектами R с такими функциями, как `html_text` и `html_table` .

Чтобы очистить таблицу основных этапов со страницы Википедии на R , код будет выглядеть так:

```
library(rvest)
```

```
url <- 'https://en.wikipedia.org/wiki/R_(programming_language)'

        # scrape HTML from website
url %>% read_html() %>%
    # select HTML tag with class="wikitable"
    html_node(css = '.wikitable') %>%
    # parse table into data.frame
    html_table() %>%
    # trim for printing
    dplyr::mutate(Description = substr(Description, 1, 70))
```

```
##    Release       Date                                                Description
## 1     0.16            This is the last alpha version developed primarily by Ihaka
## 2     0.49 1997-04-23 This is the oldest source release which is currently availab
## 3     0.60 1997-12-05 R becomes an official part of the GNU Project. The code is h
## 4   0.65.1 1999-10-07 First versions of update.packages and install.packages funct
## 5      1.0 2000-02-29 Considered by its developers stable enough for production us
## 6      1.4 2001-12-19 S4 methods are introduced and the first version for Mac OS X
## 7      2.0 2004-10-04 Introduced lazy loading, which enables fast loading of data
## 8      2.1 2005-04-18 Support for UTF-8 encoding, and the beginnings of internatio
## 9     2.11 2010-04-22                          Support for Windows 64 bit systems.
## 10    2.13 2011-04-14 Adding a new compiler function that allows speeding up funct
## 11    2.14 2011-10-31 Added mandatory namespaces for packages. Added a new paralle
## 12    2.15 2012-03-30 New load balancing functions. Improved serialization speed f
## 13     3.0 2013-04-03 Support for numeric index values 231 and larger on 64 bit sy
```

В то время как это возвращает data.frame, обратите внимание, что, как это типично для очищенных данных, еще предстоит очистка данных: здесь, даты форматирования, вставка `NA` s и т. Д.

Обратите внимание, что данные в менее последовательно прямоугольном формате могут занять цикл или другое дальнейшее переключение для успешного анализа. Если веб-сайт использует jQuery или другие средства для вставки контента, `read_html` может оказаться недостаточным для очистки, и может потребоваться более надежный скребок, такой как `RSelenium` .

Использование rvest при необходимости входа в систему

Я часто сталкиваюсь с проблемой при утилизации веб-страниц, как ввести идентификатор пользователя и пароль для входа на веб-сайт.

В этом примере, который я создал для отслеживания моих ответов, размещенных здесь для переполнения стека. Общий поток - это войти в систему, перейти к информации о сборке веб-страницы, добавить ее в кадр данных и перейти на следующую страницу.

```
library(rvest)

#Address of the login webpage
login<-
"https://stackoverflow.com/users/login?ssrc=head&returnurl=http%3a%2f%2fstackoverflow.com%2f"

#create a web session with the desired login address
pgsession<-html_session(login)
```

```
pgform<-html_form(pgsession)[[2]]  #in this case the submit is the 2nd form
filled_form<-set_values(pgform, email="*****", password="******")
submit_form(pgsession, filled_form)

#pre allocate the final results dataframe.
results<-data.frame()

#loop through all of the pages with the desired info
for (i in 1:5)
{
  #base address of the pages to extract information from
  url<-"http://stackoverflow.com/users/**********?tab=answers&sort=activity&page="
  url<-paste0(url, i)
  page<-jump_to(pgsession, url)

  #collect info on the question votes and question title
  summary<-html_nodes(page, "div .answer-summary")
  question<-matrix(html_text(html_nodes(summary, "div"), trim=TRUE), ncol=2, byrow = TRUE)

  #find date answered, hyperlink and whether it was accepted
  dateans<-html_node(summary, "span") %>% html_attr("title")
  hyperlink<-html_node(summary, "div a") %>% html_attr("href")
  accepted<-html_node(summary, "div") %>% html_attr("class")

  #create temp results then bind to final results
  rtemp<-cbind(question, dateans, accepted, hyperlink)
  results<-rbind(results, rtemp)
}

#Dataframe Clean-up
names(results)<-c("Votes", "Answer", "Date", "Accepted", "HyperLink")
results$Votes<-as.integer(as.character(results$Votes))
results$Accepted<-ifelse(results$Accepted=="answer-votes default", 0, 1)
```

Цикл в этом случае ограничен только 5 страницами, это необходимо изменить в соответствии с вашим приложением. Я заменил пользовательские значения на ******, надеюсь, это предоставит вам некоторые рекомендации.

Прочитайте Очистка веб-страниц и разбор онлайн: https://riptutorial.com/ru/r/topic/2890/очистка-веб-страниц-и-разбор

глава 85: Очистка данных

Вступление

Очистка данных в R имеет первостепенное значение для проведения любого анализа. любые данные, которые у вас есть, будь то измерения, сделанные в поле или очищенные от Интернета, наиболее вероятно, что вам придется изменить его, преобразовать или фильтровать, чтобы он был подходящим для вашего анализа. В этой документации мы рассмотрим следующие темы: - Удаление наблюдений с отсутствующими данными - Факторизация данных - Удаление неполных строк

Examples

Удаление отсутствующих данных из вектора

Сначала создадим вектор Vector1:

```
set.seed(123)
Vector1 <- rnorm(20)
```

И добавьте к нему недостающие данные:

```
set.seed(123)
Vector1[sample(1:length(Vector1), 5)] <- NA
```

Теперь мы можем использовать функцию is.na для подмножества вектора

```
Vector1 <- Vector1[!is.na(Vector1)]
```

Теперь полученный вектор удалит NAs исходного Vector1

Удаление неполных строк

Могут быть моменты, когда у вас есть кадр данных, и вы хотите удалить все строки, которые могут содержать значение NA, поскольку функция *complete.cases* является наилучшим вариантом.

Мы будем использовать первые 6 строк *набора* данных для обеспечения *качества,* чтобы сделать пример, поскольку он уже имеет НС

```
x <- head(airquality)
```

Это имеет две строки с NA в столбце Solar.R, чтобы удалить их, мы делаем следующее

```
x_no_NA <- x[complete.cases(x),]
```

Результирующий информационный *фрейм x_no_NA* будет иметь только полные строки без NA

Прочитайте Очистка данных онлайн: https://riptutorial.com/ru/r/topic/8165/очистка-данных

глава 86: Параллельная обработка

замечания

Для параллелизации на удаленных компьютерах библиотеки необходимо загружать на каждую машину. Предпочитают вызов `package::function()` . Несколько пакетов имеют встроенную распараллеливание, включая `caret` , `pls` и `plyr` .

Microsoft R Open (Revolution R) также использует многопоточные библиотеки BLAS / LAPACK, которые по своей сути распараллеливают многие общие функции.

Examples

Параллельная обработка с помощью пакета foreach

Пакет `foreach` обеспечивает параллельную обработку R. Но прежде чем вы захотите использовать многоядерные процессоры, вам необходимо назначить многоядерный кластер. Пакет `doSNOW` - одна из возможностей.

Простым использованием цикла foreach является вычисление суммы квадратного корня и квадрата всех чисел от 1 до 100000.

```
library(foreach)
library(doSNOW)

cl <- makeCluster(5, type = "SOCK")
registerDoSNOW(cl)

f <- foreach(i = 1:100000, .combine = c, .inorder = F) %dopar% {
    k <- i ** 2 + sqrt(i)
    k
}
```

Структура вывода `foreach` контролируется аргументом `.combine` . Структура вывода по умолчанию - это список. В приведенном выше коде `c` вместо этого используется `c` . Обратите внимание, что функция вычисления (или оператор), такая как `"+"` также может использоваться для выполнения вычисления и возврата еще одного обработанного объекта.

Важно отметить, что результатом каждого цикла foreach является последний вызов. Таким образом, в этом примере `k` будет добавлен `k` .

параметр	подробности
.combine	Комбинация Функция. Определяет, как комбинируются результаты цикла.

параметр	подробности
	Возможные значения: `c` , `cbind` , `rbind` , `"+"` , `"*"` ...
.c целью	если `TRUE` результат упорядочивается в соответствии с порядком итерации vairable (здесь `i`). Если `FALSE` результат не упорядочен. Это может иметь положительные эффекты на время вычислений.
.packages	для функций, которые предоставляются любой упаковкой, кроме `base` , например, `mass` , `randomForest` или же вы должны предоставить эти пакеты с помощью `c("mass", "randomForest")`

Параллельная обработка с параллельным пакетом

`parallel` базовый пакет позволяет осуществлять параллельное вычисление с помощью разветвления, сокетов и генерации случайных чисел.

Определите количество ядер, присутствующих на локальном хосте:

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

Создайте кластер ядер на локальном хосте:

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

Во-первых, необходимо создать функцию, подходящую для распараллеливания. Рассмотрим набор данных `mtcars` . Регрессия на `mpg` может быть улучшена путем создания отдельной модели регрессии для каждого уровня `cyl` .

```
data <- mtcars
yfactor <- 'cyl'
zlevels <- sort(unique(data[[yfactor]]))
datay <- data[,1]
dataz <- data[,2]
datax <- data[,3:11]


fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}
```

Создайте функцию, которая может проходить через все возможные итерации `zlevels` . Это все еще в серийном режиме, но является важным шагом, поскольку он определяет точный процесс, который будет распараллелен.

```
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}
```

```
for (zlevel in zlevels) {
  print("*****")
  print(zlevel)
  print(fitmodel(zlevel, datax, datay, dataz))
}
```

Выполните эту функцию:

```
worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
  }
```

Параллельные вычисления с использованием `parallel` не могут получить доступ к глобальной среде. К счастью, каждая функция создает локальное окружение `parallel` может получить доступ. Создание функции обертки позволяет распараллеливать. Функция, которая должна применяться, также должна быть помещена в окружающую среду.

```
wrapper <- function(datax, datay, dataz) {
  # force evaluation of all paramters not supplied by parallelization apply
  force(datax)
  force(datay)
  force(dataz)
  # these variables are now in an enviroment accessible by parallel function

  # function to be applied also in the environment
  fitmodel <- function(zlevel, datax, datay, dataz) {
    glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
  }

  # calling in this environment iterating over single parameter zlevel
  worker <- function(zlevel) {
    fitmodel(zlevel,datax, datay, dataz)
  }
  return(worker)
}
```

Теперь создайте кластер и запустите функцию обертки.

```
parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster,zlevels,
                              wrapper(datax, datay, dataz))
```

Всегда завершайте кластер, когда закончите.

```
parallel::stopCluster(parallelcluster)
```

`parallel` пакет включает в себя весь семейство `apply()` префиксом `par` .

Генерация случайных чисел

Основной проблемой с распараллеливанием является использование RNG в качестве семян. Случайные числа по числу повторяются числом операций либо с начала сеанса, либо с самого последнего `set.seed()` . Поскольку параллельные процессы возникают из одной и той же функции, он может использовать одно и то же семя, что может привести к одинаковым результатам! Вызовы будут выполняться в последовательном порядке на разных ядрах, не давая преимущества.

Набор семян должен быть сгенерирован и отправлен каждому параллельному процессу. Это делается автоматически в некоторых пакетах (`parallel` , `snow` и т. Д.), Но они должны быть явно адресованы другим.

```
s <- seed
for (i in 1:numofcores) {
    s <- nextRNGStream(s)
    # send s to worker i as .Random.seed
}
```

Семена также могут быть установлены для воспроизводимости.

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

mcparallelDo

Пакет `mcparallelDo` позволяет асинхронно оценивать R-код на Unix-подобных (например, Linux и MacOSX) операционных системах. Основополагающая философия пакета согласовывается с потребностями анализа разведочных данных, а не с кодированием. Для кодирования асинхронности рассмотрим `future` пакет.

пример

Создание данных

```
data(ToothGrowth)
```

Триггер mcparallelДля выполнения анализа на развилке

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)},"interactionPredictorModel")
```

Делайте другие вещи, например

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

Результат mcparallelDo возвращается в вашу целевую среду, например .GlobalEnv, когда она заполнена сообщением (по умолчанию)

```
summary(interactionPredictorModel)
```

Другие примеры

```
# Example of not returning a value until we return to the top level
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
  if (exists("output")) print(i)
}

# Example of getting a value without returning to the top level
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
  mcparallelDoCheck()
  if (exists("output")) print(i)
}
```

Прочитайте Параллельная обработка онлайн: https://riptutorial.com/ru/r/topic/1677/параллельная-обработка

глава 87: переменные

Examples

Переменные, структуры данных и основные операции

В R объекты данных обрабатываются с использованием именованных структур данных. Имена объектов можно назвать «переменными», хотя этот термин не имеет определенного значения в официальной документации R. Названия R *чувствительны к регистру* и могут содержать буквенно-цифровые символы (`az` , `Az` , `0-9`), точку / период (`.`) И подчеркивание (`_`). Чтобы создать имена для структур данных, мы должны следовать следующим правилам:

- Имена, начинающиеся с цифры или символа подчеркивания (например, `1a`) или имен, которые являются действительными численными выражениями (например, `.11`), или имена с тире ('-') или пробелы могут использоваться только при их цитировании: `` `1a` `` и `` `.11` `` . Имена будут напечатаны с обратными окнами:

```
list( '.11' ="a")
#$`.11`
#[1] "a"
```

- Все другие комбинации буквенно-цифровых символов, точек и подчеркивания могут использоваться свободно, где ссылка с обратными тактами или без них указывает на один и тот же объект.

- Имена, которые начинаются с `.` считаются именами систем и не всегда отображаются с помощью функции `ls()` .

Нет ограничений на количество символов в имени переменной.

Некоторые примеры допустимых имен объектов являются: `foobar` , `foo.bar` , `foo_bar` , `.foobar`

В R переменным присваиваются значения с помощью оператора infix-присваивания `<-` . Оператор `=` также может использоваться для назначения значений переменным, однако его правильное использование - это сопоставление значений с именами параметров в вызовах функций. Обратите внимание, что исключение пробелов вокруг операторов может создать путаницу для пользователей. Выражение `a<-1` анализируется как назначение (`a <- 1`), а не как логическое сравнение (`a < -1`).

```
> foo <- 42
> fooEquals = 43
```

Поэтому `foo` присваивается значение `42` . Набрав `foo` в консоли, `fooEquals 42` , а при вводе

`fooEquals` будет выводиться `43` .

```
> foo
[1] 42
> fooEquals
[1] 43
```

Следующая команда присваивает значение переменной `x` и печатает значение одновременно:

```
> (x <- 5)
[1] 5
# actually two function calls: first one to `<-`; second one to the `()`-function
> is.function(`(`)
[1] TRUE  # Often used in R help page examples for its side-effect of printing.
```

Также возможно назначать переменные с помощью `->` .

```
> 5 -> x
> x
[1] 5
>
```

Типы структур данных

Не существует скалярных типов данных в R. Векторы длины-один действуют подобно скалярам.

- Векторы: Атомные векторы должны быть последовательностью объектов одного класса: последовательность чисел или последовательность логических последовательностей или последовательность символов. `v <- c(2, 3, 7, 10)` , `v2 <- c("a", "b", "c")` - оба вектора.
- Матрицы: матрица чисел, логическая или символьная. `a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)` . Подобно векторам, матрица должна быть сделана из элементов одного класса. Для извлечения элементов из матрицы должны быть указаны строки и столбцы: `a[1,2]` возвращает `[1] 5` который является элементом в первой строке, второй столбец.
- Списки: объединение разных элементов `mylist <- list (course = 'stat', date = '04/07/2009', num_isc = 7, num_cons = 6, num_mat = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), results = c(30, 19, 29, NA, 25, 26 ,27) )` . Извлечение элементов из списка может быть выполнено по имени (если список указан) или по индексу. В данном примере `mylist$results` и `mylist[[6]]` получает тот же самый элемент. Предупреждение: если вы попробуете `mylist[6]` , R не даст вам ошибку, но извлечет результат в виде списка. Хотя `mylist[[6]][2]` разрешен (он дает вам 19), `mylist[6][2]` дает вам ошибку.
- data.frame: объект с столбцами, которые являются векторами равной длины, но (возможно) разными типами. Они не являются матрицами. `exam <- data.frame(matr =`

`as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30,22,30,NA,28,28,27))` .
Столбцы могут быть прочитаны по имени `exam$matr` , `exam[, 'matr']` или по индексному `exam[1]` , `exam[,1]` . Строки также можно читать по `exam['rowname', ]` по имени `exam['rowname', ]` или индексному `exam[1,]` . Dataframes на самом деле представляют собой только списки с определенной структурой (атрибуты rownames-attribute и равные длины)

Общие операции и некоторые предостерегающие советы

Операции по умолчанию выполняются по элементам. См. `?Syntax` для правил приоритета оператора. Большинство операторов (и могут выполнять другие функции в базе R) имеют правила утилизации, которые допускают аргументы неравной длины. Учитывая эти объекты:

Примеры объектов

```
> a <- 1
> b <- 2
> c <- c(2,3,4)
> d <- c(10,10,10)
> e <- c(1,2,3,4)
> f <- 1:6
> W <- cbind(1:4,5:8,9:12)
> Z <- rbind(rep(0,3),1:3,rep(10,3),c(4,7,1))
```

Некоторые векторные операции

```
> a+b # scalar + scalar
[1] 3
> c+d # vector + vector
[1] 12 13 14
> a*b # scalar * scalar
[1] 2
> c*d # vector * vector (componentwise!)
[1] 20 30 40
> c+a # vector + scalar
[1] 3 4 5
> c^2 #
[1]  4  9 16
> exp(c)
[1]  7.389056 20.085537 54.598150
```

Некоторые предупреждения о работе в

векторе!

```
> c+e # warning but.. no errors, since recycling is assumed to be desired.
[1] 3 5 7 6
Warning message:
In c + e : longer object length is not a multiple of shorter object length
```

R суммирует то, что он может, а затем повторно использует более короткий вектор, чтобы заполнить пробелы ... Предупреждение было дано только потому, что два вектора имеют длины, которые не являются точно кратными. c + f # никакого предупреждения.

Некоторые операции с матрицами Предупреждение!

```
> Z+W # matrix + matrix #(componentwise)
> Z*W # matrix* matrix#(Standard product is always componentwise)
```

Для умножения матрицы: V% *% W

```
> W + a # matrix+ scalar is still componentwise
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    3    7   11
[3,]    4    8   12
[4,]    5    9   13

> W + c # matrix + vector... : no warnings and R does the operation in a column-wise manner
     [,1] [,2] [,3]
[1,]    3    8   13
[2,]    5   10   12
[3,]    7    9   14
[4,]    6   11   16
```

«Частные» переменные

Ведущая точка в имени переменной или функции в R обычно используется для обозначения того, что переменная или функция должны быть скрыты.

Итак, объявив следующие переменные

```
> foo <- 'foo'
> .foo <- 'bar'
```

И затем, используя функцию `ls` для отображения объектов, будет отображаться только первый объект.

```
> ls()
[1] "foo"
```

Однако при передаче `all.names = TRUE` в функцию будет отображаться «частная» переменная

```
> ls(all.names = TRUE)
[1] ".foo"          "foo"
```

Прочитайте переменные онлайн: https://riptutorial.com/ru/r/topic/9013/переменные

глава 88: Переработка отходов

замечания

Что такое переработка в R

Утилизация - это когда объект автоматически расширяется в определенных операциях, чтобы соответствовать длине другого более длинного объекта.

Например, векторное сложение приводит к следующему:

```
c(1,2,3) + c(1,2,3,4,5,6)
[1] 2 4 6 5 7 9
```

Из-за рециркуляции операция, которая фактически произошла, была:

```
c(1,2,3,1,2,3) + c(1,2,3,4,5,6)
```

В тех случаях, когда более длинный объект не кратен короче, отображается предупреждающее сообщение:

```
c(1,2,3) + c(1,2,3,4,5,6,7)
[1] 2 4 6 5 7 9 8
Warning message:
In c(1, 2, 3) + c(1, 2, 3, 4, 5, 6, 7) :
  longer object length is not a multiple of shorter object length
```

Другой пример утилизации:

```
matrix(nrow =5, ncol = 2, 1:5 )
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    4    4
[5,]    5    5
```

Examples

Использование повторного использования в подмножестве

Утилизация может быть использована умным способом упрощения кода.

Подменю

Если мы хотим сохранить каждый третий элемент вектора, мы можем сделать следующее:

```
my_vec <-   c(1,2,3,4,5,6,7,8,9,10)
my_vec[c(TRUE, FALSE)]

[1] 1 3 5 7 9
```

Здесь логическое выражение было расширено до длины вектора.

Мы также можем проводить сравнения с использованием рециркуляции:

```
my_vec <-   c("foo", "bar", "soap", "mix")
my_vec == "bar"

[1] FALSE  TRUE FALSE FALSE
```

Здесь «бар» перерабатывается.

Прочитайте Переработка отходов онлайн: https://riptutorial.com/ru/r/topic/5649/переработка-отходов

глава 89: Перестановка данных между длинными и широкими формами

Вступление

В R табличные данные хранятся в кадрах данных . В этом разделе рассматриваются различные способы преобразования одной таблицы.

замечания

Полезные пакеты

- Реорганизация, укладка и разбиение на data.table
- Изменить форму tidyr
- splitstackshape

Examples

Функция изменения формы

Наиболее гибкая базовая функция R для переформатирования данных `reshape` . См. « `?reshape` его синтаксис».

```
# create unbalanced longitudinal (panel) data set
set.seed(1234)
df <- data.frame(identifier=rep(1:5, each=3),
                 location=rep(c("up", "down", "left", "up", "center"), each=3),
                 period=rep(1:3, 5), counts=sample(35, 15, replace=TRUE),
                 values=runif(15, 5, 10))[-c(4,8,11),]
df

   identifier location period counts   values
1           1       up      1      4 9.186478
2           1       up      2     22 6.431116
3           1       up      3     22 6.334104
5           2     down      2     31 6.161130
6           2     down      3     23 6.583062
7           3     left      1      1 6.513467
9           3     left      3     24 5.199980
10          4       up      1     18 6.093998
12          4       up      3     20 7.628488
13          5   center      1     10 9.573291
14          5   center      2     33 9.156725
15          5   center      3     11 5.228851
```

Обратите внимание, что data.frame несимметричен, то есть в блоке 2 отсутствует наблюдение в первом периоде, тогда как в блоках 3 и 4 отсутствуют наблюдения во втором периоде. Также обратите внимание, что в течение периодов есть две переменные: количество и значения, а также два значения, которые не различаются: идентификатор и местоположение.

Длинные

Чтобы изменить формат data.frame на широкий формат,

```
# reshape wide on time variable
df.wide <- reshape(df, idvar="identifier", timevar="period",
                   v.names=c("values", "counts"), direction="wide")
df.wide
   identifier location values.1 counts.1 values.2 counts.2 values.3 counts.3
1           1       up 9.186478        4 6.431116       22 6.334104       22
5           2     down       NA       NA 6.161130       31 6.583062       23
7           3     left 6.513467        1       NA       NA 5.199980       24
10          4       up 6.093998       18       NA       NA 7.628488       20
13          5   center 9.573291       10 9.156725       33 5.228851       11
```

Обратите внимание, что отсутствующие периоды времени заполняются NC.

При изменении ширины аргумент «v.names» указывает столбцы, которые меняются со временем. Если переменная местоположения не нужна, ее можно отбросить до изменения с аргументом «drop». При отбрасывании единственного столбца без изменения / non-id из файла data.fram аргумент v.names становится ненужным.

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
        drop="location")
```

Широкий и длинный

Чтобы изменить длину с текущей df.wide, минимальный синтаксис

```
reshape(df.wide, direction="long")
```

Однако это обычно сложнее:

```
# remove "." separator in df.wide names for counts and values
names(df.wide)[grep("\\.", names(df.wide))] <-
              gsub("\\.", "", names(df.wide)[grep("\\.", names(df.wide))])
```

Теперь простой синтаксис приведет к ошибке об неопределенных столбцах.

С именами столбцов, которые сложнее для функции `reshape` автоматически анализировать, иногда необходимо добавить «переменный» аргумент, который говорит `reshape` для

группировки определенных переменных в широком формате для преобразования в длинный формат. Этот аргумент принимает список векторов имен переменных или индексов.

```
reshape(df.wide, idvar="identifier",
        varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

При долгой перестройке аргумент «v.names» может быть предоставлен для переименования полученных переменных.

Иногда спецификацию «меняющегося» можно избежать с помощью аргумента «sep», который говорит о том, чтобы `reshape` какая часть имени переменной указывает аргумент значения и который определяет аргумент времени.

Изменение формы данных

Часто данные поступают в таблицы. Как правило, можно разделить эти табличные данные в широком и длинном форматах. В широком формате каждая переменная имеет свой собственный столбец.

Человек	Высота (см]	Возраст [год]
Alison	178	20
боб	174	45
деревенщина	182	31

Однако иногда удобнее иметь длинный формат, в котором все переменные находятся в одном столбце, а значения находятся во втором столбце.

Человек	переменная	Значение
Alison	Высота (см]	178
боб	Высота (см]	174
деревенщина	Высота (см]	182
Alison	Возраст [год]	20
боб	Возраст [год]	45
деревенщина	Возраст [год]	31

Base R, а также сторонние пакеты могут использоваться для упрощения этого процесса.

Для каждого из вариантов будет использоваться набор данных `mtcars` . По умолчанию этот набор данных имеет длинный формат. Чтобы пакеты работали, мы вставляем имена строк в качестве первого столбца.

```
mtcars # shows the dataset
data <- data.frame(observation=row.names(mtcars),mtcars)
```

База R

В базе R есть две функции, которые можно использовать для преобразования между широким и длинным форматом: `stack()` и `unstack()` .

```
long <- stack(data)
long # this shows the long format
wide <- unstack(long)
wide # this shows the wide format
```

Однако эти функции могут стать очень сложными для более сложных случаев использования. К счастью, есть и другие варианты использования сторонних пакетов.

Пакет tidyr

В этом пакете используется `gather()` для преобразования из широкоугольного в long и `spread()` для преобразования из long в wide.

```
library(tidyr)
long <- gather(data, variable, value, 2:12) # where variable is the name of the
# variable column, value indicates the name of the value column and 2:12 refers to
# the columns to be converted.
long # shows the long result
wide <- spread(long,variable,value)
wide # shows the wide result (~data)
```

Пакет data.table

Пакет data.table расширяет функции `reshape2` и использует функцию `melt()` для перехода от широкого к длинному и `dcast()` для перехода от длинного к широкому.

```
library(data.table)
long <- melt(data,'observation',2:12,'variable', 'value')
long # shows the long result
wide <- dcast(long, observation ~ variable)
wide # shows the wide result (~data)
```

Прочитайте Перестановка данных между длинными и широкими формами онлайн: https://riptutorial.com/ru/r/topic/2904/перестановка-данных-между-длинными-и-широкими-формами

глава 90: Плавный и невольный с data.table

Синтаксис

- `melt(DT, id.vars=c(..), variable.name="CategoryLabel", value.name="Value")` C `melt(DT, id.vars=c(..), variable.name="CategoryLabel", value.name="Value")`
- Cast with `dcast(DT, LHS ~ RHS, value.var="Value", fun.aggregate=sum)`

параметры

параметр	подробности
id.vars	скажите, `melt` какие столбцы сохранить
variable.name	расскажите `melt` , как назвать колонку с ярлыками категорий
value.name	скажите, `melt` что вызвать столбец, который имеет значения, связанные с метками категории
value.var	tell `dcast` где можно найти значения для `dcast` в столбцах
формула	скажите `dcast` какие столбцы сохранить для формирования уникального идентификатора записи (LHS) и который содержит метки категорий (RHS)
fun.aggregate	указать функцию, которую следует использовать, когда операция литья генерирует список значений в каждой ячейке

замечания

Большая часть того, что входит в данные кондиционирования для создания моделей или визуализации, может быть достигнуто с помощью `data.table` . По сравнению с другими вариантами, `data.table` предлагает преимущества скорости и гибкости.

Examples

Сводные и невольные табличные данные с данными. Таблица - I

Преобразование из широкой формы в длинную форму

Загрузка `data USArrests` из `datasets` .

```
data("USArrests")
head(USArrests)

           Murder Assault UrbanPop Rape
Alabama      13.2     236       58 21.2
Alaska       10.0     263       48 44.5
Arizona       8.1     294       80 31.0
Arkansas      8.8     190       50 19.5
California    9.0     276       91 40.6
Colorado      7.9     204       78 38.7
```

Использовать `?USArrests` чтобы узнать больше. Сначала преобразуем в `data.table` . Имена состояний - это имена строк в исходном `data.frame` .

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

Это данные в широкой форме. Он имеет столбец для каждой переменной. Данные также могут храниться в длинной форме без потери информации. У длинной формы есть один столбец, в котором хранятся имена переменных. Затем он имеет другой столбец для значений переменных. Такая длинная форма `USArrests` выглядит так.

```
            State    Crime  Rate
  1:       Alabama   Murder  13.2
  2:        Alaska   Murder  10.0
  3:       Arizona   Murder   8.1
  4:      Arkansas   Murder   8.8
  5:    California   Murder   9.0
 ---
196:      Virginia     Rape  20.7
197:    Washington     Rape  26.2
198: West Virginia     Rape   9.3
199:     Wisconsin     Rape  10.8
200:       Wyoming     Rape  15.6
```

Мы используем функцию `melt` для переключения с широкой формы на длинную.

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

По умолчанию `melt` обрабатывает все столбцы с числовыми данными как переменные со значениями. В `USArrests` переменная `UrbanPop` представляет собой процентное городское население штата. Он отличается от других вариаций: « `Murder` , `Assault` и `Rape` , которые представляют собой насильственные преступления на 100 000 человек. Предположим, мы хотим сохранить столбец `UrbanPop` . Мы достигаем этого, установив `id.vars` следующим образом.

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop" ),
             variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

Обратите внимание, что мы указали имена столбца, содержащего имена категорий (Murder, Assault и т. Д.) С `value.name` `variable.name` и столбцом, содержащим значения с `value.name` . Наши данные выглядят так.

```
            State UrbanPop  Crime Rate
 1:       Alabama       58 Murder 13.2
 2:        Alaska       48 Murder 10.0
 3:       Arizona       80 Murder  8.1
 4:      Arkansas       50 Murder  8.8
 5:    California       91 Murder  9.0
```

Генерация резюме с использованием подхода с разделенным применением и сочетанием стиля - легкий ветерок. Например, чтобы обобщить насильственные преступления со стороны государства?

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

Это дает:

```
        State ViolentCrime
1:    Alabama        270.4
2:     Alaska        317.5
3:    Arizona        333.1
4:   Arkansas        218.3
5: California        325.6
6:   Colorado        250.6
```

Сводные и невольные табличные данные с данными. Таблица - II

Преобразование из длинной формы в широкую форму

Чтобы восстановить данные из предыдущего примера, используйте `dcast` так.

```
DTc <- dcast(DTmu, State + UrbanPop ~ Crime)
```

Это дает данные в оригинальной широкой форме.

```
            State UrbanPop Murder Assault Rape
 1:        Alabama       58   13.2     236 21.2
 2:         Alaska       48   10.0     263 44.5
 3:        Arizona       80    8.1     294 31.0
 4:       Arkansas       50    8.8     190 19.5
 5:     California       91    9.0     276 40.6
```

Здесь обозначение формулы используется для указания столбцов, которые образуют уникальный идентификатор записи (LHS), и столбец, содержащий метки категорий для новых имен столбцов (RHS). Какой столбец используется для числовых значений? По умолчанию `dcast` использует первый столбец с численными значениями, оставшимися после спецификации спецификации. Чтобы сделать явным, используйте параметр `value.var` с

именем столбца.

Когда операция создает список значений в каждой ячейке, `dcast` предоставляет метод `fun.aggregate` для обработки ситуации. Скажем, меня интересуют государства с аналогичным городским населением при расследовании преступлений. Я добавляю столбец `Decile` с вычисленной информацией.

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
levels(DTmu$Decile) <- paste0(1:10, "D")
```

Теперь, `Decile ~ Crime` получаете несколько значений на ячейку. Я могу использовать `fun.aggregate` чтобы определить, как они обрабатываются. Как текстовые, так и числовые значения могут обрабатываться таким образом.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Это дает:

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=mean)
```

Это дает:

```
             State UrbanPop  Crime Rate Decile
  1:       Alabama       58 Murder 13.2     4D
  2:        Alaska       48 Murder 10.0     2D
  3:       Arizona       80 Murder  8.1     8D
  4:      Arkansas       50 Murder  8.8     2D
  5:    California       91 Murder  9.0    10D
```

В каждом дециле городского населения есть несколько государств. Используйте `fun.aggregate` чтобы указать, как они должны обрабатываться.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

Это суммирует данные для подобных состояний, давая следующее.

```
    Decile Murder Assault  Rape
 1:     1D   39.4     808  62.6
 2:     2D   35.3     815  94.3
 3:     3D   22.6     451  67.7
 4:     4D   54.9     898 106.0
 5:     5D   42.4     758 107.6
```

Прочитайте Плавный и невольный с data.table онлайн: https://riptutorial.com/ru/r/topic/6934/плавный-и-невольный-с-data-table

глава 91: Подменю

Вступление

Учитывая объект R, нам может потребоваться отдельный анализ для одной или нескольких частей содержащихся в нем данных. Процесс получения этих частей данных из данного объекта называется `subsetting` .

замечания

Отсутствующие значения:

Отсутствующие значения (`NA` s), используемые в подмножестве с `[` return `NA` с индексом `NA`

> выбирает неизвестный элемент и поэтому возвращает NA в соответствующий элемент.

Тип «по умолчанию» `NA` является «логическим» (`typeof(NA)`), что означает, что как любой «логический» вектор, используемый в подмножестве, будет **перерабатываться** в соответствии с длиной подмножества. Таким образом, `x[NA]` эквивалентно `x[as.logical(NA)]` который эквивалентен `x[rep_len(as.logical(NA), length(x))]` и, следовательно, он возвращает отсутствующее значение (`NA`) для каждого элемента `x` . В качестве примера:

```
x <- 1:3
x[NA]
## [1] NA NA NA
```

При индексировании с «числовым» / «целочисленным» `NA` выбирает один элемент `NA` (для каждого `NA` в индексе):

```
x[as.integer(NA)]
## [1] NA

x[c(NA, 1, NA, NA)]
## [1] NA  1 NA NA
```

Подстановка за пределы:

Оператор `[` с одним аргументом передается, позволяет индексы `> length(x)` и возвращает `NA` для атомных векторов или `NULL` для общих векторов. Напротив, с `[[` и когда `[` передано больше аргументов (т.е. подмножество вне границ объектов с `length(dim(x)) > 2`), возвращается ошибка:

```
(1:3)[10]
## [1] NA
```

```
(1:3)[[10]]
## Error in (1:3)[[10]] : subscript out of bounds
as.matrix(1:3)[10]
## [1] NA
as.matrix(1:3)[, 10]
## Error in as.matrix(1:3)[, 10] : subscript out of bounds
list(1, 2, 3)[10]
## [[1]]
## NULL
list(1, 2, 3)[[10]]
## Error in list(1, 2, 3)[[10]] : subscript out of bounds
```

Поведение такое же, когда подмножество с векторами-символами, которые также не соответствуют атрибуту «names» объекта:

```
c(a = 1, b = 2)["c"]
## <NA>
##   NA
list(a = 1, b = 2)["c"]
## <NA>
## NULL
```

Разделы помощи:

См. « `?Extract` для получения дополнительной информации.

Examples

Атомные векторы

Атомные векторы (которые исключают списки и выражения, которые также являются векторами) являются подмножествами с использованием оператора `[` operator:

```
# create an example vector
v1 <- c("a", "b", "c", "d")

# select the third element
v1[3]
## [1] "c"
```

`[` Оператор также может принимать вектор в качестве аргумента. Например, чтобы выбрать первый и третий элементы:

```
v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"
```

Иногда нам может потребоваться опустить конкретное значение из вектора. Это может быть достигнуто с помощью отрицательного знака (`-`) перед индексом этого значения. Например, чтобы опустить первое значение из v1, используйте `v1[-1]` . Это может быть

расширено до нескольких значений прямым способом. Например, `v1[-c(1,3)]` .

```
> v1[-1]
[1] "b" "c" "d"
> v1[-c(1,3)]
[1] "b" "d"
```

В некоторых случаях мы хотели бы знать, особенно, когда длина вектора велика, индекс определенного значения, если он существует:

```
> v1=="c"
[1] FALSE FALSE  TRUE FALSE
> which(v1=="c")
[1] 3
```

Если атомный вектор имеет имена (атрибут `names`), он может быть подмножеством с использованием символьного вектора имен:

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
##  one   two three
##    1     2     3

v["two"]
## two
##   2
```

Оператор `[[` также может использоваться для индексирования атомных векторов с различиями в том, что он принимает вектор индексирования с длиной одного и разделяет любые имена:

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
##  attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

Векторы также могут быть подмножеством с использованием логического вектора. В отличие от подмножества с числовыми и символьными векторами, логический вектор, используемый для подмножества, должен быть равен длине вектора, элементы которого извлекаются, поэтому, если логический вектор `y` используется для подмножества `x` , то есть `x[y]` , если `length(y) < length(x)` то `y` будет переработано в соответствии с `length(x)` :

```
v[c(TRUE, FALSE, TRUE)]
##  one three
##    1     3

v[c(FALSE, TRUE)]  # recycled to 'c(FALSE, TRUE, FALSE)'
```

```
## two
##   2

v[TRUE]   # recycled to 'c(TRUE, TRUE, TRUE)'
##  one   two three
##    1     2     3

v[FALSE]   # handy to discard elements but save the vector's type and basic structure
## named integer(0)
```

Списки

Список может быть подмножеством с `[` :

```
l1 <- list(c(1, 2, 3), 'two' = c("a", "b", "c"), list(10, 20))
l1
## [[1]]
## [1] 1 2 3
##
## $two
## [1] "a" "b" "c"
##
## [[3]]
## [[3]][[1]]
## [1] 10
##
## [[3]][[2]]
## [1] 20

l1[1]
## [[1]]
## [1] 1 2 3

l1['two']
## $two
## [1] "a" "b" "c"

l1[[2]]
## [1] "a" "b" "c"

l1[['two']]
## [1] "a" "b" "c"
```

Обратите внимание, что результат `l1[2]` по-прежнему является списком, поскольку `[` оператор выбирает элементы списка, возвращая меньший список. `[[` Оператор извлекает элементы списка, возвращая объект типа элемента списка).

Элементы могут индексироваться по номеру или символьной строке имени (если оно существует). Несколько элементов могут быть выбраны с помощью `[` путем передачи вектора чисел или строк имен. Индексирование с вектором `length > 1` в `[` и `[[` возвращает «список» с указанными элементами и рекурсивное подмножество (если доступно) *соответственно* :

```
l1[c(3, 1)]
```

```
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
##
## [[2]]
## [1] 1 2 3
```

По сравнению с:

```
l1[[c(3, 1)]]
## [1] 10
```

что эквивалентно:

```
l1[[3]][[1]]
## [1] 10
```

Оператор `$` позволяет вам выбирать элементы списка исключительно по имени, но в отличие от `[` и `[[` , не требует кавычек. В качестве инфиксного оператора `$` может принимать только одно имя:

```
l1$two
## [1] "a" "b" "c"
```

Кроме того, оператор `$` допускает частичное сопоставление по умолчанию:

```
l1$t
## [1] "a" "b" "c"
```

в отличие от `[[` где необходимо указать, разрешено ли частичное совпадение:

```
l1[["t"]]
## NULL
l1[["t", exact = FALSE]]
## [1] "a" "b" "c"
```

Параметры `options(warnPartialMatchDollar = TRUE)` , при возникновении частичного совпадения с `$` :

```
l1$t
## [1] "a" "b" "c"
## Warning message:
## In l1$t : partial match of 't' to 'two'
```

Матрицы

Для каждого измерения объекта оператор `[` operator принимает один аргумент. Векторы имеют один размер и принимают один аргумент. Матрицы и фреймы данных имеют два измерения и принимают два аргумента, заданные как `[i, j]` где `i` - строка, а `j` - столбец. Индексирование начинается с 1.

```
## a sample matrix
mat <- matrix(1:6, nrow = 2, dimnames = list(c("row1", "row2"), c("col1", "col2", "col3")))

mat
#      col1 col2 col3
# row1    1    3    5
# row2    2    4    6
```

`mat[i,j]` - элемент в `i` строке, `j` столбец матрицы `mat` . Например, значение `i 2` и значение `j 1` задают число во второй строке и первом столбце матрицы. Опускание `i` или `j` возвращает все значения в этом измерении.

```
mat[ , 3]
## row1 row2
##    5    6

mat[1, ]
# col1 col2 col3
#    1    3    5
```

Когда матрица имеет имена строк или столбцов (не обязательно), они могут использоваться для подмножества:

```
mat[ , 'col1']
# row1 row2
#    1    2
```

По умолчанию результат подмножества будет, по возможности, упрощен. Если подмножество имеет только одно измерение, как в приведенных выше примерах, результатом будет одномерный вектор, а не двумерная матрица. Это значение по умолчанию может быть переопределено с аргументом `drop = FALSE` в `[` :

```
## This selects the first row as a vector
class(mat[1, ])
# [1] "integer"

## Whereas this selects the first row as a 1x3 matrix:
class(mat[1, , drop = F])
# [1] "matrix"
```

Конечно, размеры нельзя отбрасывать, если сам выбор имеет два измерения:

```
mat[1:2, 2:3]  ## A 2x2 matrix
#      col2 col3
# row1    3    5
# row2    4    6
```

Выбор отдельных записей матрицы по их позициям

Также возможно использовать матрицу Nx2 для выбора N отдельных элементов из матрицы (например, как работает система координат). Если вы хотите выделить в векторе записи матрицы в `(1st row, 1st column), (1st row, 3rd column), (2nd row, 3rd column), (2nd row, 1st column)` это может легко сделать, создав матрицу индексов с этими координатами и используя ее для подмножества матрицы:

```
mat
#      col1 col2 col3
# row1    1    3    5
# row2    2    4    6

ind = rbind(c(1, 1), c(1, 3), c(2, 3), c(2, 1))
ind
#      [,1] [,2]
# [1,]    1    1
# [2,]    1    3
# [3,]    2    3
# [4,]    2    1

mat[ind]
# [1] 1 5 6 2
```

В приведенном выше примере 1-й столбец матрицы `ind` ссылается на строки в `mat` , второй столбец `ind` ссылается на столбцы в `mat` .

Кадры данных

Подстановка кадра данных в меньший кадр данных может быть выполнена так же, как подмножество списка.

```
> df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)

> df3
##   x y
## 1 1 a
## 2 2 b
## 3 3 c

> df3[1]   # Subset a variable by number
##   x
## 1 1
## 2 2
## 3 3

> df3["x"]   # Subset a variable by name
##   x
## 1 1
## 2 2
## 3 3

> is.data.frame(df3[1])
## TRUE
```

```
> is.list(df3[1])
## TRUE
```

Подсечение кадра данных в вектор столбца может быть выполнено с использованием двойных скобок [[]] или оператора знака доллара $.

```
> df3[[2]]    # Subset a variable by number using [[ ]]
## [1] "a" "b" "c"

> df3[["y"]]  # Subset a variable by name using [[ ]]
## [1] "a" "b" "c"

> df3$x    # Subset a variable by name using $
## [1] 1 2 3

> typeof(df3$x)
## "integer"

> is.vector(df3$x)
## TRUE
```

Подмножество данных в виде двумерной матрицы может быть выполнено с использованием термов `i` и `j` .

```
> df3[1, 2]    # Subset row and column by number
## [1] "a"

> df3[1, "y"]  # Subset row by number and column by name
## [1] "a"

> df3[2, ]     # Subset entire row by number
##   x y
## 2 2 b

> df3[ , 1]    # Subset all first variables
## [1] 1 2 3

> df3[ , 1, drop = FALSE]
##   x
## 1 1
## 2 2
## 3 3
```

Примечание. Только `data.frame` `j` (столбец) упрощает собственный тип переменной, но подмножество только `i` возвращает `data.frame` , так как разные переменные могут иметь разные типы и классы. Установка параметра `drop` в `FALSE` поддерживает фрейм данных.

```
> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
```

```
## TRUE
```

Другие объекты

Операторы `[` и `[[` являются примитивными функциями, которые являются общими. Это означает, что любой *объект* в R (в частности `isTRUE(is.object(x))` --ie имеет явный атрибут класса), может иметь свое определенное поведение при подмножестве; т.е. имеет свои собственные *методы* для `[` и / или `[[` .

Например, это имеет место с объектами «data.frame» (`is.object(iris)`), где `[[.data.frame` методы `[.data.frame` и `[[.data.frame` , и они созданы для отображения как «матричных» и " list" -подобное подмножество. С форсированием ошибки при подмножестве «data.frame» мы видим, что на самом деле функция `[.data.frame` была вызвана, когда мы просто использовали `[` .

```
iris[invalidArgument, ]
## Error in `[.data.frame`(iris, invalidArgument, ) :
##   object 'invalidArgument' not found
```

Без дополнительной информации о текущей теме, пример `[` метод:

```
x = structure(1:5, class = "myClass")
x[c(3, 2, 4)]
## [1] 3 2 4
'[.myClass' = function(x, i) cat(sprintf("We'd expect '%s[%s]' to be returned but this a
custom `[` method and should have a `?[.myClass` help page for its behaviour\n",
deparse(substitute(x)), deparse(substitute(i))))

x[c(3, 2, 4)]
## We'd expect 'x[c(3, 2, 4)]' to be returned but this a custom `[` method and should have a
`?[.myClass` help page for its behaviour
## NULL
```

Мы можем преодолеть метод отправки `[` с использованием эквивалентного не-generic `.subset` (и `.subset2` для `[[`). Это особенно полезно и эффективно при программировании наших собственных «классов» и избегает `unclass(x)` например, `unclass(x)`) при эффективном вычислении на наших «классах» (избегая отправки методов и копирования объектов):

```
.subset(x, c(3, 2, 4))
## [1] 3 2 4
```

Векторное индексирование

В этом примере мы будем использовать вектор:

```
> x <- 11:20
> x
```

```
 [1] 11 12 13 14 15 16 17 18 19 20
```

R векторы 1-индексируются, поэтому, например, `x[1]` вернет `11` . Мы также можем извлечь подвектор `x` , передавая вектор индексов оператору скобки:

```
> x[c(2,4,6)]
[1] 12 14 16
```

Если мы передадим вектор отрицательных индексов, R вернет под-вектор с указанными указанными индексами:

```
> x[c(-1,-3)]
[1] 12 14 15 16 17 18 19 20
```

Мы также можем передать булевой вектор в оператор скобки, и в этом случае он возвращает подвектор, соответствующий координатам, где вектор индексирования имеет значение `TRUE` :

```
> x[c(rep(TRUE,5),rep(FALSE,5))]
[1] 11 12 13 14 15 16
```

Если вектор индексирования короче длины массива, то он будет повторяться, как в:

```
> x[c(TRUE,FALSE)]
[1] 11 13 15 17 19
> x[c(TRUE,FALSE,FALSE)]
[1] 11 14 17 20
```

Операции элементарных матриц

Пусть A и B - две матрицы той же размерности. Операторы `+` , `-` , `/` , `*` , `^` при использовании с матрицами одинаковой размерности выполняют требуемые операции над соответствующими элементами матриц и возвращают новую матрицу той же размерности. Эти операции обычно называются элементарными операциями.

оператор	A op B	Имея в виду
+	A + B	Добавление соответствующих элементов A и B
-	A - B	Вычитает элементы из B из соответствующих элементов A
/	A / B	Разделяет элементы A на соответствующие элементы B
*	A * B	Умножает элементы A на соответствующие элементы B
^	A ^ (-1)	Например, дает матрицу, элементы которой являются обратными от A

Для «истинного» умножения матрицы, как видно из *линейной алгебры* , используйте `%*%` . Например, умножение A на B: `A %*% B` Требования к размеру заключаются в том, что `ncol()` of `A` является таким же, как `nrow()` of `B`

Некоторые функции, используемые с матрицами

функция	пример	Цель
nrow ()	nrow (A)	определяет количество строк A
Ncol ()	Ncol (A)	определяет количество столбцов A
rownames ()	rownames (A)	выводит имена строк матрицы A
COLNAMES ()	COLNAMES (A)	печатает имена столбцов матрицы A
rowMeans ()	rowMeans (A)	вычисляет средства для каждой строки матрицы A
colMeans ()	colMeans (A)	вычисляет средства каждого столбца матрицы A
upper.tri ()	upper.tri (A)	возвращает вектор, элементы которого являются верхними
		треугольная матрица квадратной матрицы A
lower.tri ()	lower.tri (A)	возвращает вектор, элементы которого являются нижними
		треугольная матрица квадратной матрицы A
Det ()	Det (A)	приводит к определителю матрицы A
решать()	решения (A)	приводит к обратному отношению к неособой матрице A
Diag ()	Diag (A)	возвращает диагональную матрицу, чьи недиагностические элементы являются нулями и
		диагонали такие же, как и квадратная матрица A
т ()	т (A)	возвращает транспонирование матрицы A
собственный ()	собственный (A)	ретунирует собственные значения и собственные векторы матрицы A
is.matrix ()	is.matrix (A)	возвращает TRUE или FALSE в зависимости от того,

функция	пример	Цель
		является ли A матрицей или нет.
as.matrix ()	as.matrix (x)	создает матрицу из вектора x

Прочитайте Подменю онлайн: https://riptutorial.com/ru/r/topic/1686/подменю

глава 92: Получение данных

Вступление

Получайте данные непосредственно в сеанс R. Одной из приятных особенностей R является легкость сбора данных. Существует несколько способов распространения данных с использованием пакетов R.

Examples

Встроенные наборы данных

R имеет обширную коллекцию встроенных наборов данных. Обычно они используются для обучения, чтобы создавать быстрые и легко воспроизводимые примеры. Существует хорошая веб-страница, в которой перечислены встроенные наборы данных:

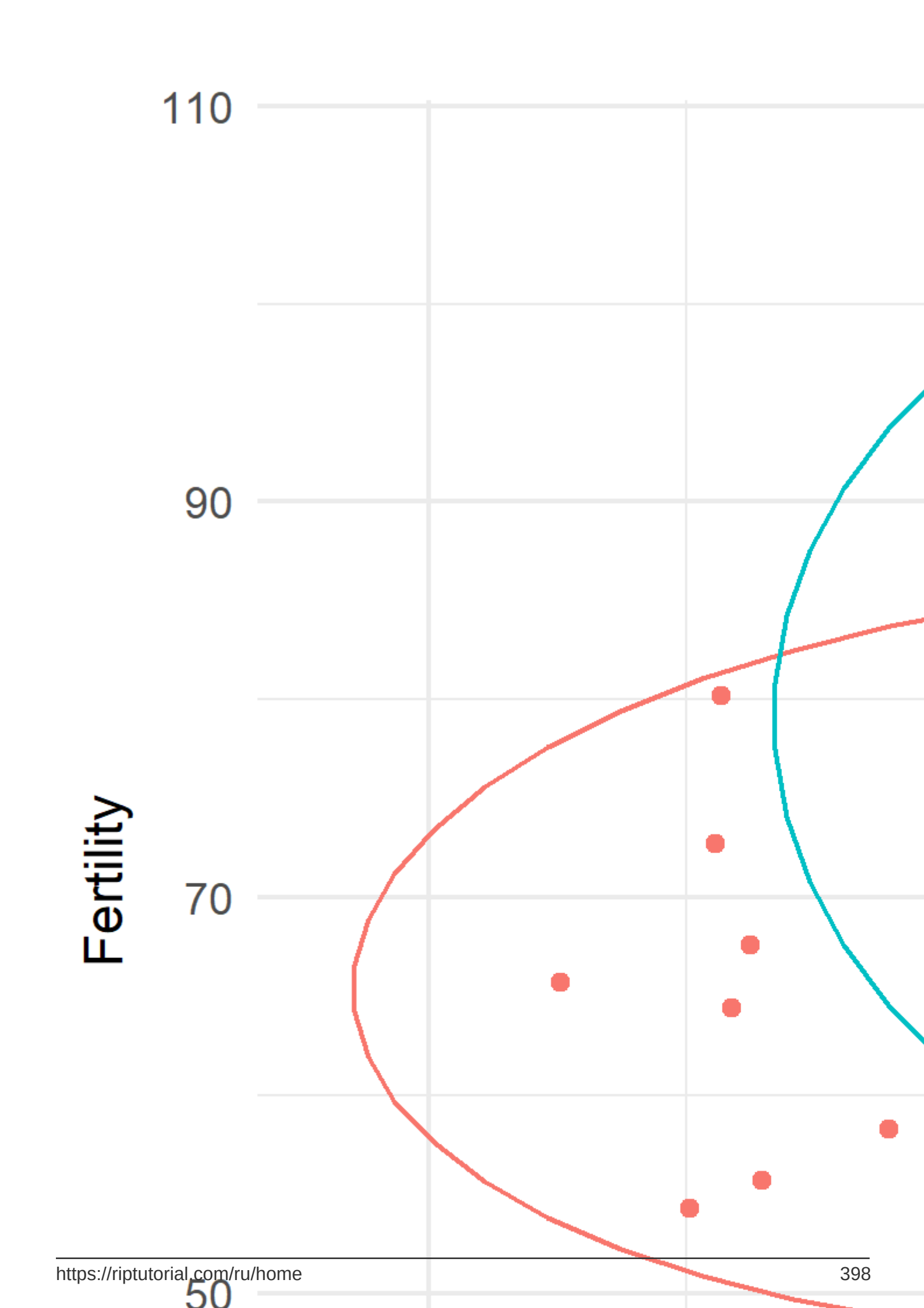
https://vincentarelbundock.github.io/Rdatasets/datasets.html

пример

Швейцарские фертильности и социально-экономические показатели (1888). Давайте проверим разницу в рождаемости, основанную на рождаемости и господстве католического населения.

```
library(tidyverse)

swiss %>%
        ggplot(aes(x = Agriculture, y = Fertility,
                   color = Catholic > 50))+
        geom_point()+
        stat_ellipse()
```

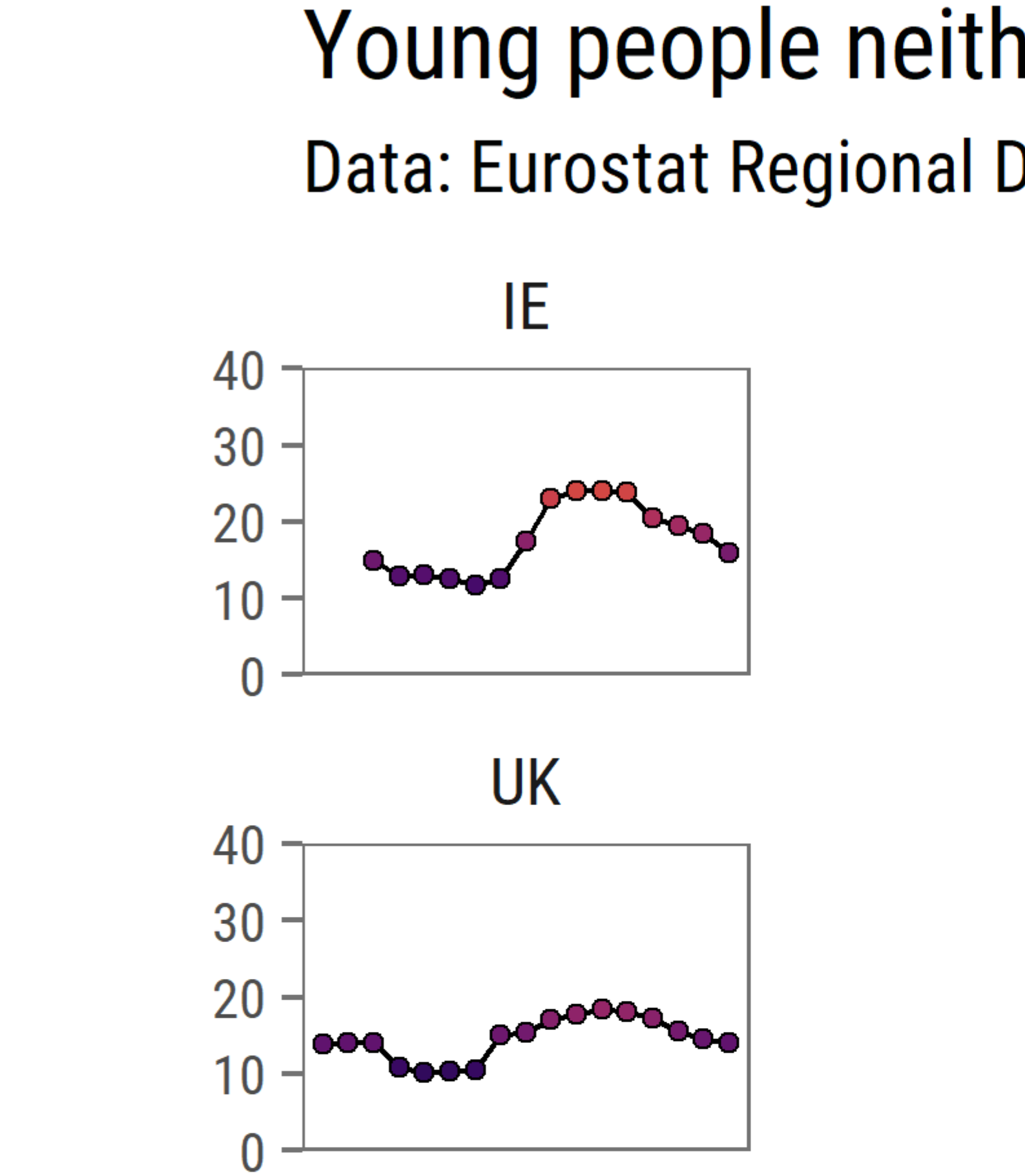

110
90
70
50
Fertility

он не находит всех доступных доступных наборов данных. Это удобнее просматривать код набора данных вручную на веб-сайте Евростата: База данных стран или Региональная база данных . Если автоматическая загрузка не работает, данные можно захватывать вручную через средство массовой загрузки .

```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

# download NEET data for countries
neet <- get_eurostat("edat_lfse_22")

neet %>%
        filter(geo %>% paste %>% nchar == 2,
               sex == "T", age == "Y18-24") %>%
        group_by(geo) %>%
        mutate(avg = values %>% mean()) %>%
        ungroup() %>%
        ggplot(aes(x = time %>% year(),
                   y = values))+
        geom_path(aes(group = 1))+
        geom_point(aes(fill = values), pch = 21)+
        scale_x_continuous(breaks = seq(2000, 2015, 5),
                           labels = c("2000", "'05", "'10", "'15"))+
        scale_y_continuous(expand = c(0, 0), limits = c(0, 40))+
        scale_fill_viridis("NEET, %", option = "B")+
        facet_geo(~ geo, grid = "eu_grid1")+
        labs(x = "Year",
             y = "NEET, %",
             title = "Young people neither in employment nor in education and training in
Europe",
             subtitle = "Data: Eurostat Regional Database, 2000-2016",
             caption = "ikashnitsky.github.io")+
        theme_few(base_family =  "Roboto Condensed", base_size = 15)+
        theme(axis.text = element_text(size = 10),
              panel.spacing.x = unit(1, "lines"),
              legend.position = c(0, 0),
              legend.justification = c(0, 0))
```


Young people neith
Data: Eurostat Regional D
IE
40
30
20
10
0
UK
40
30
20
10
0

Института демографических исследований им. Макса Планка, который собирает и обрабатывает данные о смертности среди людей в тех странах, где имеется более или менее надежная статистика.

```
# load required packages
library(tidyverse)
library(extrafont)
library(HMDHFDplus)

country <- getHMDcountries()

exposures <- list()
for (i in 1: length(country)) {
        cnt <- country[i]
        exposures[[cnt]] <- readHMDweb(cnt, "Exposures_1x1", user_hmd, pass_hmd)
        # let's print the progress
        paste(i,'out of',length(country))
} # this will take quite a lot of time
```

Обратите внимание, что аргументы `user_hmd` и `pass_hmd` являются учетными данными для входа на веб-сайт базы данных смертности человека. Для доступа к данным необходимо создать учетную запись на странице http://www.mortality.org/ и предоставить свои собственные полномочия функции `readHMDweb()` .

```
sr_age <- list()

for (i in 1:length(exposures)) {
        di <- exposures[[i]]
        sr_agei <- di %>% select(Year,Age,Female,Male) %>%
                filter(Year %in% 2012) %>%
                select(-Year) %>%
                transmute(country = names(exposures)[i],
                          age = Age, sr_age = Male / Female * 100)
        sr_age[[i]] <- sr_agei
}
sr_age <- bind_rows(sr_age)

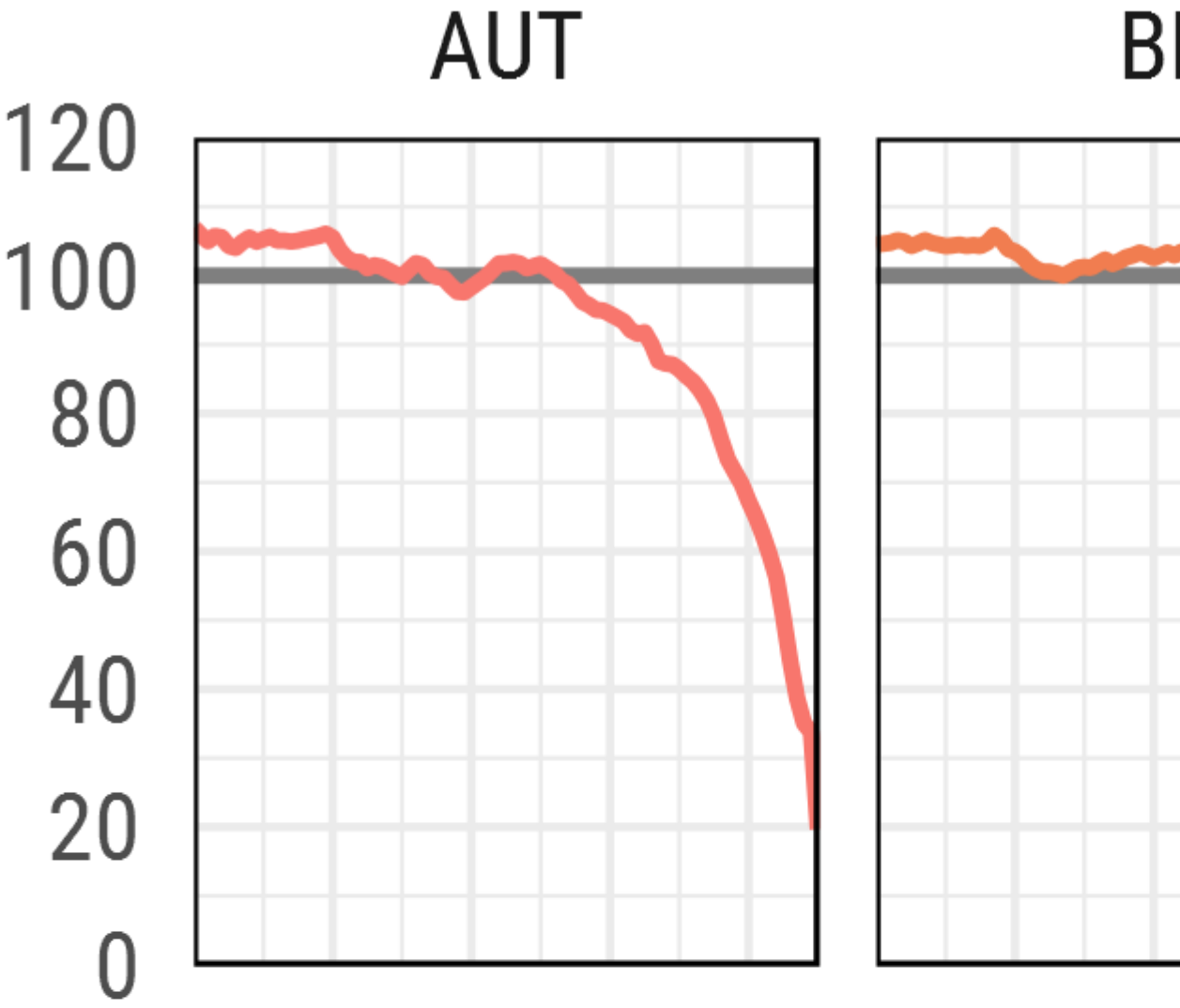
# remove optional populations
sr_age <- sr_age %>% filter(!country %in% c("FRACNP","DEUTE","DEUTW","GBRCENW","GBR_NP"))

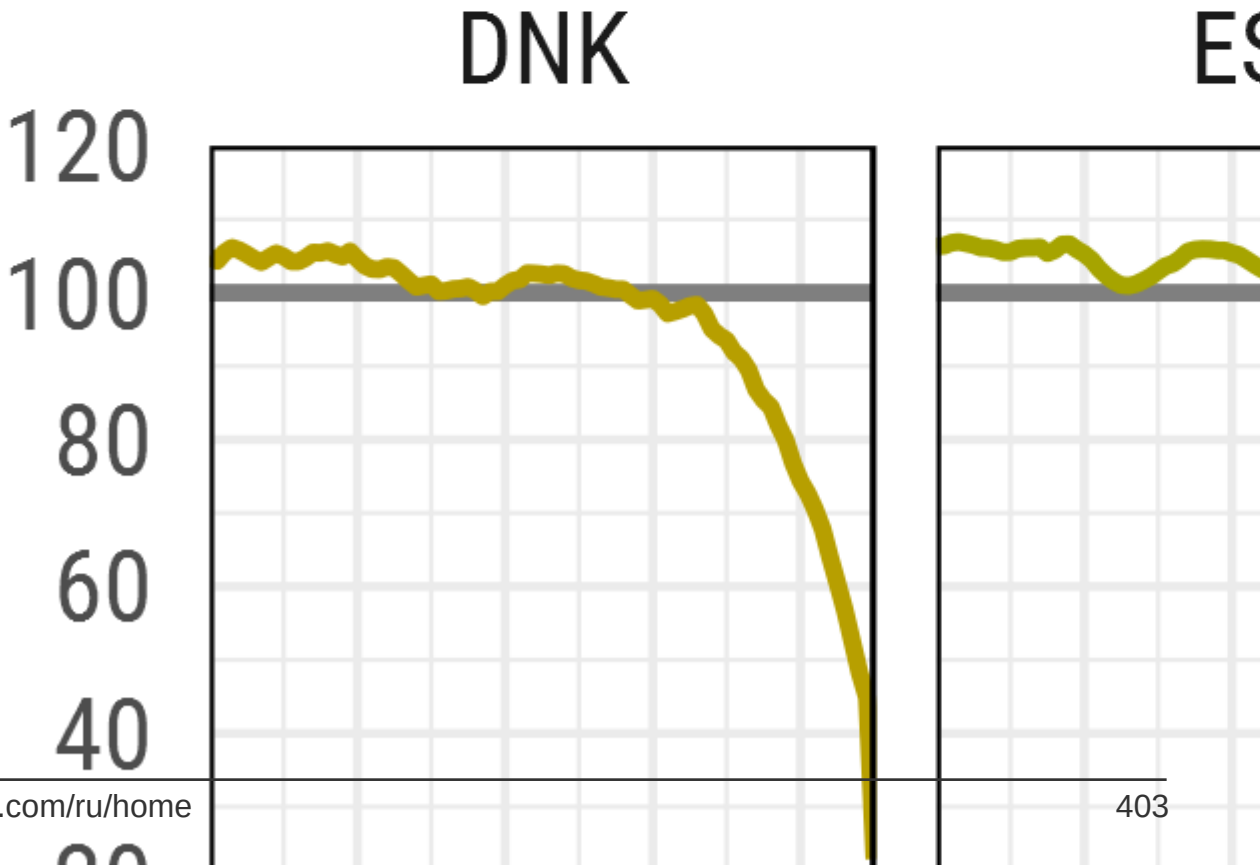
# summarize all ages older than 90 (too jerky)
sr_age_90 <- sr_age %>% filter(age %in% 90:110) %>%
        group_by(country) %>% summarise(sr_age = mean(sr_age, na.rm = T)) %>%
        ungroup() %>% transmute(country, age=90, sr_age)

df_plot <- bind_rows(sr_age %>% filter(!age %in% 90:110), sr_age_90)

# finaly - plot
df_plot %>%
        ggplot(aes(age, sr_age, color = country, group = country))+
        geom_hline(yintercept = 100, color = 'grey50', size = 1)+
        geom_line(size = 1)+
        scale_y_continuous(limits = c(0, 120), expand = c(0, 0), breaks = seq(0, 120, 20))+
        scale_x_continuous(limits = c(0, 90), expand = c(0, 0), breaks = seq(0, 80, 20))+
        xlab('Age')+
        ylab('Sex ratio, males per 100 females')+
        facet_wrap(~country, ncol=6)+
```

```
        theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
        theme(legend.position='none',
              panel.border = element_rect(size = .5, fill = NA))
```


AUT
120
100
80
60
40
20
0


DNK
120
100
80
60
40

получение-данных

глава 93: Презентация RMarkdown и knitr

Синтаксис

- Заголовок:
  - Формат YAML, используемый при компиляции скрипта для определения общего параметра и метаданных

параметры

параметр	определение
заглавие	название документа
автор	Автор документа
Дата	Дата документа: может быть « `r format(Sys.time(), '%d %B, %Y')`
автор	Автор документа
выход	Формат вывода документа: доступно не менее 10 форматов. Для html-документа `html_output` . Для документа PDF, `pdf_document` , ..

замечания

Параметры дополнительных параметров:

Подвариант	описание	HTML	PDF	слово	а.с.	РТФ	Мэриленд	GitHub
citation_package	Пакет LaTeX для обработки цитат, natbib, biblatex или none		Икс				Икс	
code_folding	Пусть читатели переключают отображение R-кода, «none», «hide» или « show»,	Икс						

Подвариант	описание	HTML	PDF	слово	а.с.	РТФ	Мэриленд	GitHub
colortheme	Цветная тема Beamer для использования							
CSS	Файл CSS для стилирования документа	Икс						
DEV	Графическое устройство для вывода рисунка (например, « png»)	Икс	Икс				Икс	Икс
продолжительность	Добавить таймер обратного отсчета (в минутах) до нижнего колонтитула слайдов							
fig_caption	Должны ли цифры отображаться под заголовками?	Икс	Икс	Икс	Икс			
fig_height, fig_width	Высота и ширина рисунка по умолчанию (в дюймах) для документа	Икс	Икс	Икс	Икс	Икс	Икс	Икс
основной момент	Подсветка синтаксиса: «танго», «пигменты», «кате», «дзенбурн», «текст»,	Икс	Икс	Икс				
включает в себя	Файл	Икс	Икс		Икс		Икс	Икс

Подвариант	описание	HTML	PDF	слово	а.с.	РТФ	Мэриленд	GitHub
	содержимого для размещения в документе (in_header, before_body, after_body)							
дополнительный	Должны ли появляться по одному за раз (при нажатие кнопок мыши)?							
keep_md	Сохраните копию файла . md, содержащего вывод knitr	Икс		Икс	Икс	Икс		
keep_tex	Сохраните копию .tex-файла, содержащего вывод knitr		Икс					
latex_engine	Двигатель для визуализации латекса, или " pdflatex", "xelatex", lualatex "		Икс					
lib_dir	Каталог файлов зависимостей для использования (Bootstrap, MathJax и т. Д.)	Икс						
MathJax	Установите локальный или URL-адрес, чтобы использовать	Икс						

Подвариант	описание	HTML	PDF	слово	а.с.	РТФ	Мэриленд	GitHub
	локальную / URL-версию MathJax для рендеринга							
md_extensions	Расширения Markdown для добавления к определению по умолчанию или R Markdown	Икс	Икс	Икс	Икс	Икс	Икс	Икс
number_sections	Добавить нумерацию разделов в заголовки	Икс	Икс					
pandoc_args	Дополнительные аргументы для перехода к Pandoc	Икс	Икс	Икс	Икс	Икс	Икс	Икс
preserve_yaml	Сохранять переднюю часть YAML в финальном документе?						Икс	
reference_docx	docx файл, стили которого должны копироваться при выпуске docx			Икс				
self_contained	Встраивание зависимостей в документ	Икс						
slide_level	Самый низкий уровень заголовка, который							

Подвариант	описание	HTML	PDF	слово	а.с.	РТФ	Мэриленд	GitHub
	определяет отдельные слайды							
меньше	Использовать меньший размер шрифта в презентации?							
умный	Преобразуйте прямые кавычки в фигурные, тире до em-тире, ... до эллипсов и т. Д.	Икс						
шаблон	Шаблон Pandoc для использования при рендеринге файла	Икс	Икс		Икс			
тема	Тема Bootswatch или Beamer для использования на странице	Икс						
TOC	Добавить оглавление в начале документа	Икс	Икс	Икс		Икс	Икс	Икс
toc_depth	Самый низкий уровень заголовков для добавления к оглавлению	Икс	Икс	Икс		Икс	Икс	Икс
toc_float	Поплавьте оглавление слева от основного содержимого	Икс						

Examples

Пример Rstudio

Это скрипт, сохраненный как .Rmd, напротив r скриптов, сохраненных как .R.

Чтобы связать скрипт, используйте функцию `render` или используйте кнопку быстрого доступа в Rstudio.

```
---
title: "Rstudio exemple of a rmd file"
author: 'stack user'
date: "22 July 2016"
output: html_document
---

The header is used to define the general parameters and the metadata.

## R Markdown

This is an R Markdown document.
It is a script written in markdown with the possibility to insert chunk of R code in it.
To insert R code, it needs to be encapsulated into inverted quote.

Like that for a long piece of code:

```{r cars}
summary(cars)
```

And like ``r cat("that")`` for small piece of code.

Including Plots

You can also embed plots, for example:

```{r echo=FALSE}
plot(pressure)
```
```

Добавление нижнего колонтитула в презентацию ioslides

Добавление нижнего колонтитула не является возможным. К счастью, мы можем использовать jQuery и CSS, чтобы добавить нижний колонтитул к слайдам презентации ioslides, представленной с помощью knitr. Прежде всего, мы должны включить плагин jQuery. Это делается по линии

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

Теперь мы можем использовать jQuery для изменения DOM (*объектной модели документа*) нашей презентации. Другими словами: мы изменяем структуру HTML документа. Как только презентация загружается (`$(document).ready(function() { ... })`), мы выбираем все

слайды, которые не имеют атрибутов класса `.title-slide` , `.backdrop` или `.segue` и добавьте тег `<footer></footer>` прямо перед каждым слайдом «закрыто» (так до `</slide>`). `label` атрибута содержит содержимое, которое будет отображаться позже.

Все, что нам нужно сделать, это разбить наш нижний колонтитул на CSS:

После каждого `<footer>` (`footer::after`):

- отобразить содержимое `label` атрибута
- использовать размер шрифта 12
- расположите нижний колонтитул (20 пикселей от нижней части слайда и 60 пикселей слева)

(другие свойства могут быть проигнорированы, но могут быть изменены, если в презентации используется другой шаблон стиля).

```
---
title: "Adding a footer to presentaion slides"
author: "Martin Schmelzer"
date: "26 Juli 2016"
output: ioslides_presentation
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = FALSE)
```

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>

<script>
    $(document).ready(function() {
      $('slide:not(.title-slide, .backdrop, .segue)').append('<footer label=\"My amazing
footer!\"></footer>');
    })
</script>

<style>
  footer:after {
    content: attr(label);
    font-size: 12pt;
    position: absolute;
    bottom: 20px;
    left: 60px;
    line-height: 1.9;
  }
</style>

Slide 1

This is slide 1.

Slide 2

This is slide 2
```

```
# Test

## Slide 3

And slide 3.
```

Результат будет выглядеть так:

Прочитайте Презентация RMarkdown и knitr онлайн: https://riptutorial.com/ru/r/topic/2999/презентация-rmarkdown-и-knitr

глава 94: принуждение

Вступление

Принуждение происходит в R, когда тип объектов изменяется при вычислении либо неявно, либо с помощью функций для явного принуждения (например, as.numeric, as.data.frame и т. Д.).

Examples

Неявное принуждение

Принуждение происходит с типами данных в R, часто неявно, так что данные могут вместить все значения. Например,

```
x = 1:3
x
[1] 1 2 3
typeof(x)
#[1] "integer"

x[2] = "hi"
x
#[1] "1"  "hi" "3"
typeof(x)
#[1] "character"
```

Обратите внимание, что сначала `x` имеет `integer` тип. Но когда мы назначили `x[2] = "hi"` , все элементы `x` были принудительно введены в `character` поскольку векторы в R могут хранить только данные одного типа.

Прочитайте принуждение онлайн: https://riptutorial.com/ru/r/topic/9793/принуждение

глава 95: Проверка пакетов

Вступление

Пакеты основаны на базе R. В этом документе объясняется, как проверять установленные пакеты и их функциональность. Связанные документы: установка пакетов

замечания

Основная архивная сеть (CRAN) - это основной репозиторий пакетов .

Examples

Посмотреть информацию о пакете

Чтобы получить информацию о пакете dplyr и его описаниях функций:

```
help(package = "dplyr")
```

Не нужно сначала загружать пакет.

Просмотр встроенных наборов данных пакета

Чтобы увидеть встроенные наборы данных из пакета dplyr

```
 data(package = "dplyr")
```

Не нужно сначала загружать пакет.

Список экспортируемых функций пакета

Чтобы получить список функций в пакете dplyr, сначала нужно загрузить пакет:

```
library(dplyr)
ls("package:dplyr")
```

Просмотр версии пакета

Условия: пакет должен быть установлен как минимум. Если не загружен в текущий сеанс, это не проблема.

```
  ## Checking package version which was installed at past or
```

```
  ## installed currently but not loaded in the current session

  packageVersion("seqinr")
  # [1] ‘3.3.3’
  packageVersion("RWeka")
  # [1] ‘0.4.29’
```

Просмотр загруженных пакетов в текущем сеансе

Чтобы проверить список загруженных пакетов

```
search()
```

ИЛИ ЖЕ

```
(.packages())
```

Прочитайте Проверка пакетов онлайн: https://riptutorial.com/ru/r/topic/7408/проверка-пакетов

глава 96: пространственный анализ

Examples

Создание пространственных точек из набора данных XY

Когда дело доходит до географических данных, R показывает, что это мощный инструмент для обработки, анализа и визуализации данных.

Часто пространственные данные доступны как координатные данные XY, установленные в табличной форме. В этом примере будет показано, как создать набор пространственных данных из набора данных XY.

Пакеты `rgdal` и `sp` предоставляют мощные функции. Пространственные данные в R могут храниться как `Spatial*DataFrame` (где `*` могут быть `Points` , `Lines` или `Polygons`).

В этом примере используются данные, которые можно загрузить в OpenGeocode .

Сначала рабочий каталог должен быть установлен в папку загруженного набора данных CSV. Кроме того, необходимо `rgdal` пакет `rgdal` .

```
setwd("D:/GeocodeExample/")
library(rgdal)
```

Впоследствии файл CSV, хранящий города и их географические координаты, загружается в R как `data.frame`

```
xy <- read.csv("worldcities.csv", stringsAsFactors = FALSE)
```

Часто полезно получить представление о данных и их структуре (например, имена столбцов, типы данных и т. Д.).

```
head(xy)
str(xy)
```

Это показывает, что столбцы широты и долготы интерпретируются как знаковые значения, поскольку они содержат записи типа «-33.532». Тем не менее, позже используемая функция `SpatialPointsDataFrame()` , которая создает пространственный набор данных требует координатных значений иметь тип данных `numeric` . Таким образом, два столбца должны быть преобразованы.

```
xy$latitude <- as.numeric(xy$latitude)
xy$longitude <- as.numeric(xy$longitude)
```

Немногие из значений не могут быть преобразованы в числовые данные и, таким образом, создаются значения `NA` . Их нужно удалить.

```
xy <- xy[!is.na(xy$longitude),]
```

Наконец, набор данных XY можно преобразовать в набор пространственных данных. Для этого требуются координаты и спецификация системы корреляции координат (CRS), в которой хранятся координаты.

```
xySPoints <- SpatialPointsDataFrame(coords = c(xy[,c("longitude", "latitude")]),
proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"),
data = xy
)
```

Функция основного сюжета может быть легко использована для подкрадывания полученных пространственных точек.

```
plot(xySPoints, pch = ".")
```

Импорт файла формы (.shp)

rgdal

Файлы формы ESRI можно легко импортировать в R, используя функцию `readOGR()` из пакета `rgdal` .

```
library(rgdal)
shp <- readORG(dsn = "/path/to/your/file", layer = "filename")
```

Важно знать, что `dsn` не должен заканчиваться на `/` и `layer` не позволяет завершить файл (например, `.shp`)

растр

Другой возможный способ импорта шейп-файлов - через библиотеку `raster` и функцию `shapefile` :

```
library(raster)
shp <- shapefile("path/to/your/file.shp")
```

Обратите внимание, что определение пути отличается от оператора импорта rgdal.

TMAP

Пакет `tmap` обеспечивает хорошую оболочку для функции `rgdal::readORG` .

```
library(tmap)
sph <- read_shape("path/to/your/file.shp")
```

Прочитайте пространственный анализ онлайн: https://riptutorial.com/ru/r/topic/2093/пространственный-анализ

глава 97: Профилирование кода

Examples

Системное время

Системное время дает вам процессорное время, необходимое для выполнения выражения R, например:

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

Вы можете добавлять большие фрагменты кода с помощью брекетов:

```
system.time({
    library(numbers)
    Primes(1,10^5)
})
```

Или используйте его для тестирования функций:

```
fibb <- function (n) {
    if (n < 3) {
        return(c(0,1)[n])
    } else {
        return(fibb(n - 2) + fibb(n -1))
    }
}

system.time(fibb(30))
```

proc.time ()

В своем простейшем случае `proc.time()` дает общее время процессора в секундах для текущего процесса. Выполнение этого в консоли дает следующий тип вывода:

```
proc.time()

#       user     system    elapsed
#    284.507    120.397 515029.305
```

Это особенно полезно для сопоставления определенных строк кода. Например:

```
t1 <- proc.time()
fibb <- function (n) {
```

```
    if (n < 3) {
        return(c(0,1)[n])
    } else {
        return(fibb(n - 2) + fibb(n -1))
    }
}
print("Time one")
print(proc.time() - t1)

t2 <- proc.time()
fibb(30)

print("Time two")
print(proc.time() - t2)
```

Это дает следующий результат:

```
source('~/.active-rstudio-document')

# [1] "Time one"
#    user  system elapsed
#       0       0       0

# [1] "Time two"
#    user  system elapsed
#   1.534   0.012   1.572
```

`system.time()` является оболочкой для `proc.time()` которая возвращает прошедшее время для конкретной команды / выражения.

```
print(t1 <- system.time(replicate(1000,12^2)))
##  user  system elapsed
## 0.000   0.000   0.002
```

Обратите внимание, что возвращаемый объект класса `proc.time` немного сложнее, чем кажется на поверхности:

```
str(t1)
## Class 'proc_time'  Named num [1:5] 0 0 0.002 0 0
##  ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
```

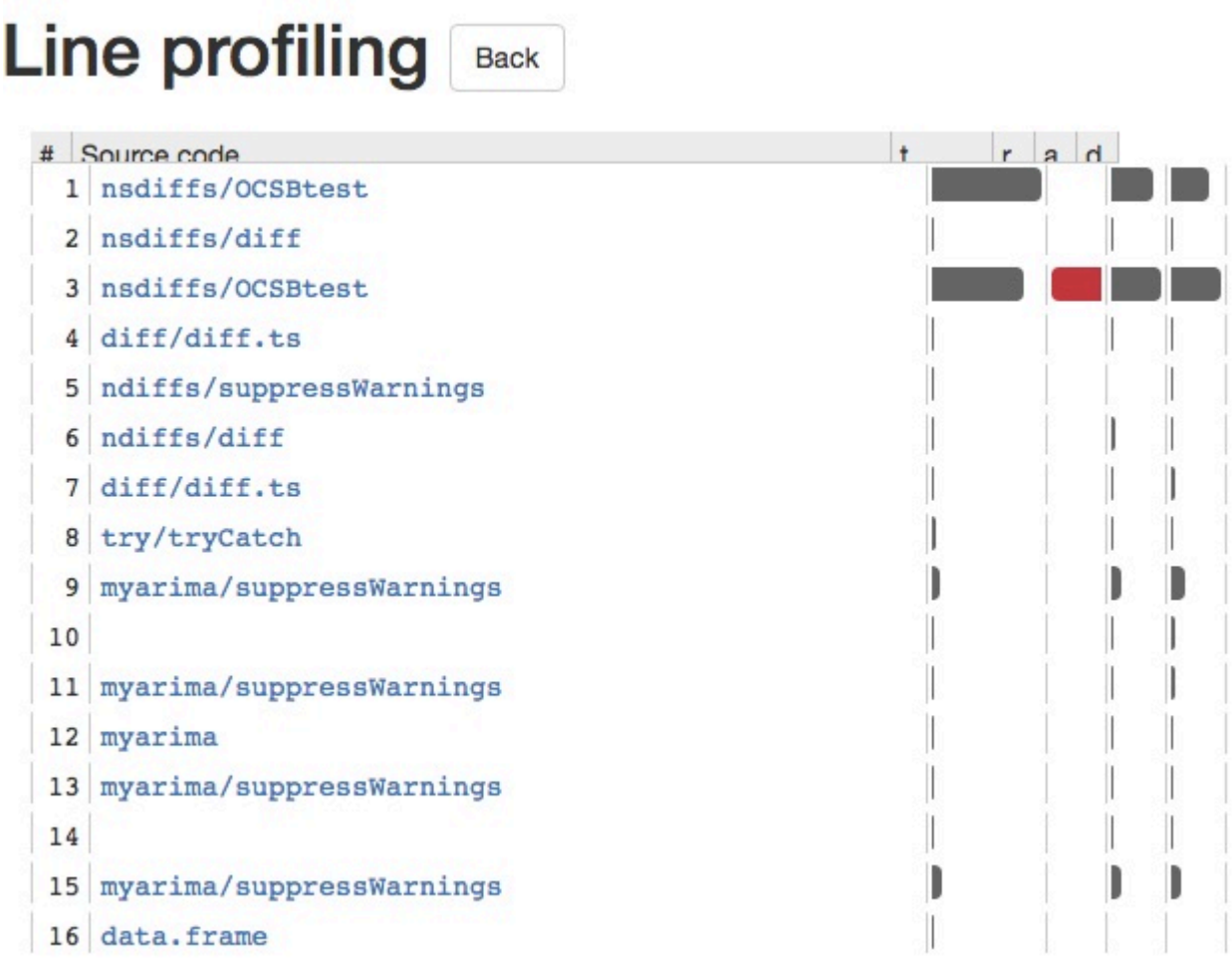
Профилирование линии

Один пакет для профилирования линии - это lineprof, который написан и поддерживается Хэдли Викхэмом. Вот `auto.arima` демонстрация того, как она работает с `auto.arima` в пакете прогноза:

```
library(lineprof)
library(forecast)

l <- lineprof(auto.arima(AirPassengers))
shine(l)
```

Это обеспечит вам блестящее приложение, которое позволит вам глубже углубиться в каждый вызов функции. Это позволяет вам легко видеть, что заставляет ваш R-код замедляться. Скриншот блестящего приложения ниже:



Microbenchmark

Microbenchmark полезен для оценки времени, требуемого для других быстрых процедур. Например, рассмотрите оценку времени, затраченного на печать мира привет.

```
system.time(print("hello world"))

# [1] "hello world"
#    user  system elapsed
#       0       0       0
```

Это связано с тем, что `system.time` по существу является оберточной функцией `proc.time` , которая измеряется в секундах. Поскольку печать «hello world» занимает менее секунды, кажется, что занятое время меньше секунды, однако это неверно. Чтобы увидеть это, мы можем использовать микрообъект пакета:

```
library(microbenchmark)
microbenchmark(print("hello world"))
```

```
# Unit: microseconds
#                 expr    min     lq     mean  median     uq     max neval
# print("hello world") 26.336 29.984 44.11637 44.6835 45.415 158.824   100
```

Здесь мы видим, что после запуска `print("hello world")` 100 раз, среднее время было на самом деле 44 микросекунды. (Обратите внимание, что запуск этого кода будет печатать « hello world» 100 раз на консоль.)

Мы можем сравнить это с эквивалентной процедурой, `cat("hello world\n")` , чтобы убедиться, что она быстрее, чем `print("hello world")` :

```
microbenchmark(cat("hello world\n"))

# Unit: microseconds
#                  expr    min      lq     mean median     uq     max neval
# cat("hello world\\n") 14.093 17.6975 23.73829 19.319 20.996 119.382   100
```

В этом случае `cat()` почти в два раза быстрее, чем `print()` .

В качестве альтернативы можно сравнить две процедуры в рамках одного и того же вызова `microbenchmark` :

```
microbenchmark(print("hello world"), cat("hello world\n"))
# Unit: microseconds
# expr                    min     lq     mean  median     uq     max neval
# print("hello world") 29.122 31.654 39.64255 34.5275 38.852 192.779   100
# cat("hello world\\n")  9.381 12.356 13.83820 12.9930 13.715  52.564   100
```

Бенчмаркинг с использованием microbenchmark

Вы можете использовать пакет `microbenchmark` для проведения «точного времени оценки выражения» в миллисекундах.

В этом примере мы сравниваем скорости шести эквивалентных данных. `data.table` выражения для обновления элементов в группе на основе определенного условия.

Более конкретно:

> `data.table` с тремя столбцами: `id` , `time` и `status` . Для каждого идентификатора я хочу найти запись с максимальным временем - тогда, если для этой записи, если состояние истинно, я хочу установить значение false, если время составляет> 7

```
library(microbenchmark)
library(data.table)

set.seed(20160723)
dt <- data.table(id = c(rep(seq(1:10000), each = 10)),
                 time = c(rep(seq(1:10000), 10)),
                 status = c(sample(c(TRUE, FALSE), 10000*10, replace = TRUE)))
```

```
setkey(dt, id, time)  ## create copies of the data so the 'updates-by-reference' don't affect
other expressions
dt1 <- copy(dt)
dt2 <- copy(dt)
dt3 <- copy(dt)
dt4 <- copy(dt)
dt5 <- copy(dt)
dt6 <- copy(dt)

microbenchmark(

  expression_1 = {
    dt1[ dt1[order(time), .I[.N], by = id]$V1, status := status * time < 7 ]
    },

  expression_2 = {
    dt2[,status := c(.SD[-.N, status], .SD[.N, status * time > 7]), by = id]
    },

  expression_3 = {
    dt3[dt3[,.N, by = id][,cumsum(N)], status := status * time > 7]
    },

  expression_4 = {
    y <- dt4[,.SD[.N],by=id]
    dt4[y, status := status & time > 7]
  },

  expression_5 = {
    y <- dt5[, .SD[.N, .(time, status)], by = id][time > 7 & status]
    dt5[y, status := FALSE]
  },

  expression_6 = {
    dt6[ dt6[, .I == .I[which.max(time)], by = id]$V1 & time > 7, status := FALSE]
    },

  times = 10L ## specify the number of times each expression is evaluated
)

# Unit: milliseconds
#          expr         min          lq        mean      median         uq          max neval
# expression_1   11.646149   13.201670   16.808399   15.643384   18.78640    26.321346    10
# expression_2 8051.898126 8777.016935 9238.323459 8979.553856 9281.93377 12610.869058    10
# expression_3    3.208773    3.385841    4.207903    4.089515    4.70146     5.654702    10
# expression_4   15.758441   16.247833   20.677038   19.028982   21.04170    36.373153    10
# expression_5 7552.970295 8051.080753 8702.064620 8861.608629 9308.62842  9722.234921    10
# expression_6   18.403105   18.812785   22.427984   21.966764   24.66930    28.607064    10
```

Результат показывает, что в этом тесте `expression_3` является самым быстрым.

Рекомендации

data.table - Добавление и изменение столбцов

data.table - специальные символы группировки в data.table

Прочитайте Профилирование кода онлайн: https://riptutorial.com/ru/r/topic/2149/

профилирование-кода

глава 98: Работа с колонкой

Examples

сумма каждого столбца

Предположим, нам нужно сделать `sum` каждого столбца в наборе данных

```
set.seed(20)
df1 <- data.frame(ID = rep(c("A", "B", "C"), each = 3), V1 = rnorm(9), V2 = rnorm(9))
m1 <- as.matrix(df1[-1])
```

Есть много способов сделать это. Используя `base R` , лучшим вариантом будет `colSums`

```
colSums(df1[-1], na.rm = TRUE)
```

Здесь мы удалили первый столбец, поскольку он не является числовым, и сделал `sum` каждого столбца, указав `na.rm = TRUE` (в случае, если в наборе данных есть любые NA)

Это также работает с `matrix`

```
colSums(m1, na.rm = TRUE)
```

Это можно сделать в петле с `lapply/sapply/vapply`

```
lapply(df1[-1], sum, na.rm = TRUE)
```

Следует отметить, что вывод - это `list` . Если нам нужен `vector` вывод

```
sapply(df1[-1], sum, na.rm = TRUE)
```

Или же

```
vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

Для матриц, если мы хотим перебирать столбцы, тогда используйте `apply` с `MARGIN = 1`

```
apply(m1, 2, FUN = sum, na.rm = TRUE)
```

Есть способы сделать это с помощью таких пакетов, как `dplyr` или `data.table`

```
library(dplyr)
df1 %>%
    summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Здесь мы передаем регулярное выражение, чтобы соответствовать именам столбцов, которые нам нужны, чтобы получить `sum` в `summarise_at` . Регулярное выражение будет соответствовать всем столбцам, начинающимся с `v` за которым следует одно или несколько чисел (`\\d+`).

`data.table`

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

Мы преобразуем «data.frame» в «data.table» (`setDT(df1)`), указали столбцы, в которых должна быть применена функция в `.SDcols` и петля через подмножество Data.table (`.SD`) и получить `sum` .

Если нам нужно использовать группу по операции, мы можем сделать это легко, указав группу по столбцу / столбцам

```
 df1 %>%
   group_by(ID) %>%
   summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

В случаях, когда нам нужна `sum` всех столбцов, `summarise_each` может использоваться вместо `summarise_at`

```
df1 %>%
    group_by(ID) %>%
    summarise_each(funs(sum(., na.rm = TRUE)))
```

Параметр `data.table`

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

Прочитайте Работа с колонкой онлайн: https://riptutorial.com/ru/r/topic/2212/работа-с-колонкой

глава 99: Растровый и графический анализ

Вступление

См. Также I / O для растровых изображений

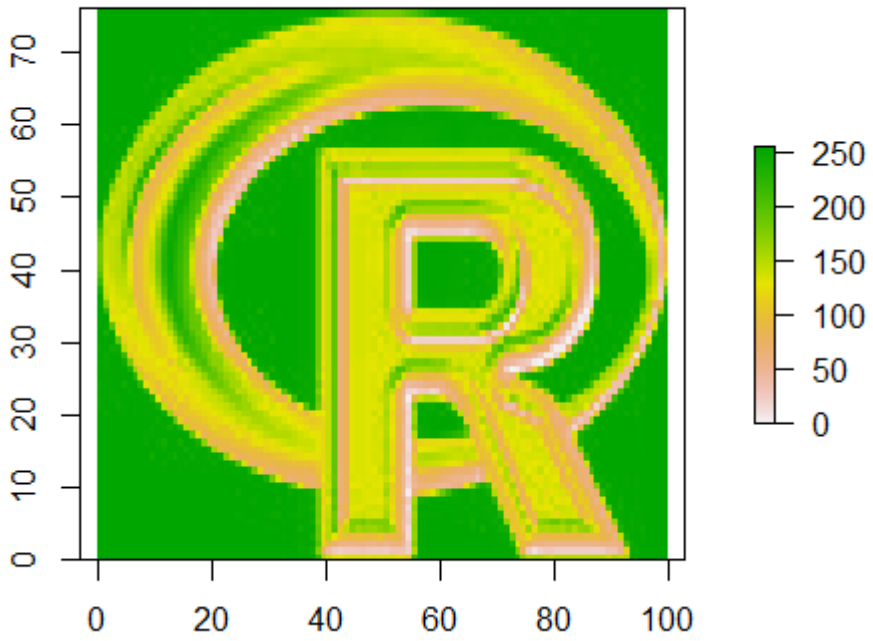
Examples

Вычисление текстуры GLCM

Матрица совпадения уровня серого (Haralick et al., 1973) - мощная функция изображения для анализа изображений. Пакет `glcm` предоставляет простую в использовании функцию для вычисления таких техатровых функций для объектов `RasterLayer` в R.

```
library(glcm)
library(raster)

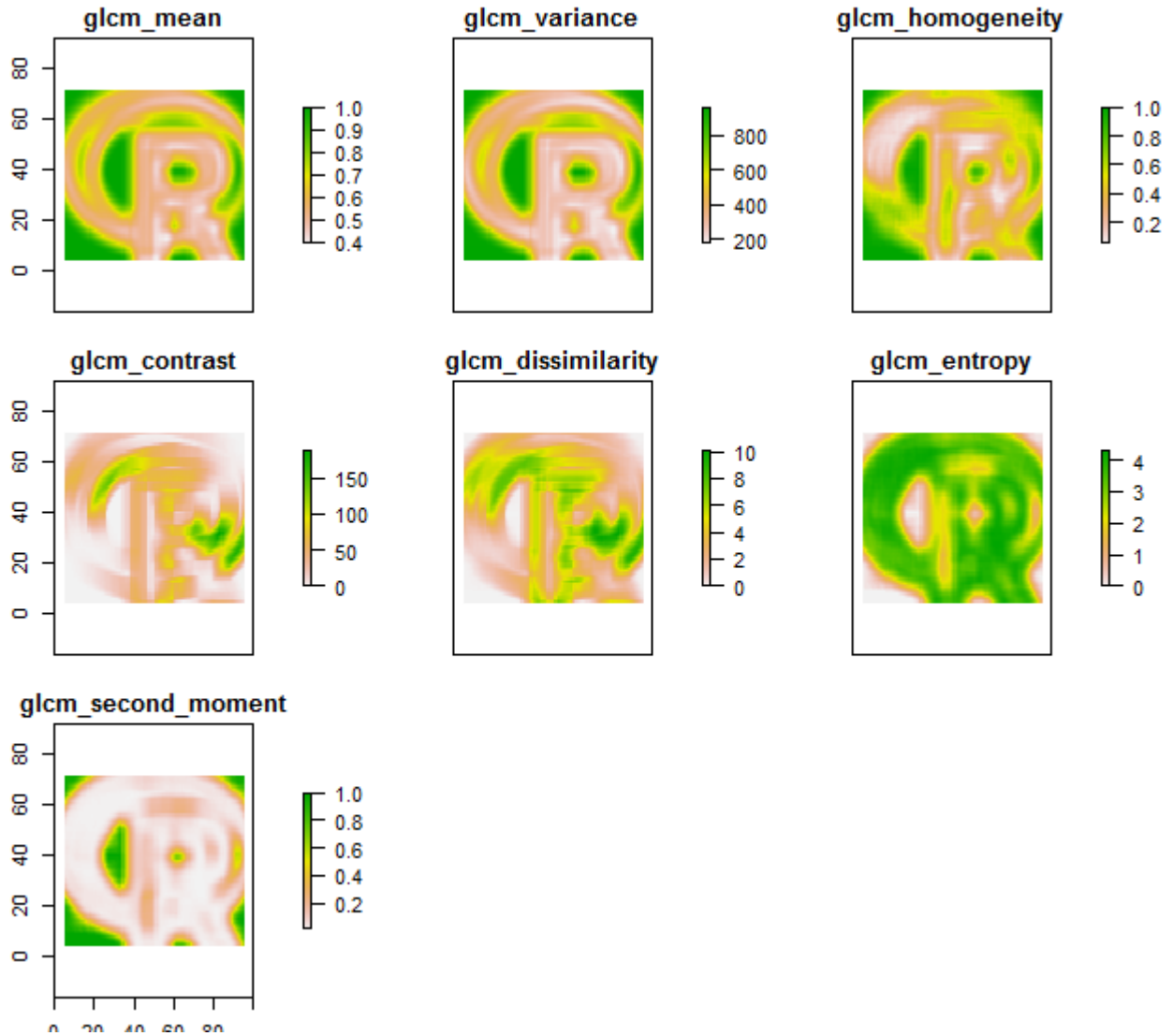
r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Вычисление текстур GLCM в одном направлении

```
rglcm <- glcm(r,
              window = c(9,9),
              shift = c(1,1),
              statistics = c("mean", "variance", "homogeneity", "contrast",
                             "dissimilarity", "entropy", "second_moment")
              )
```

```
plot(rglcm)
```

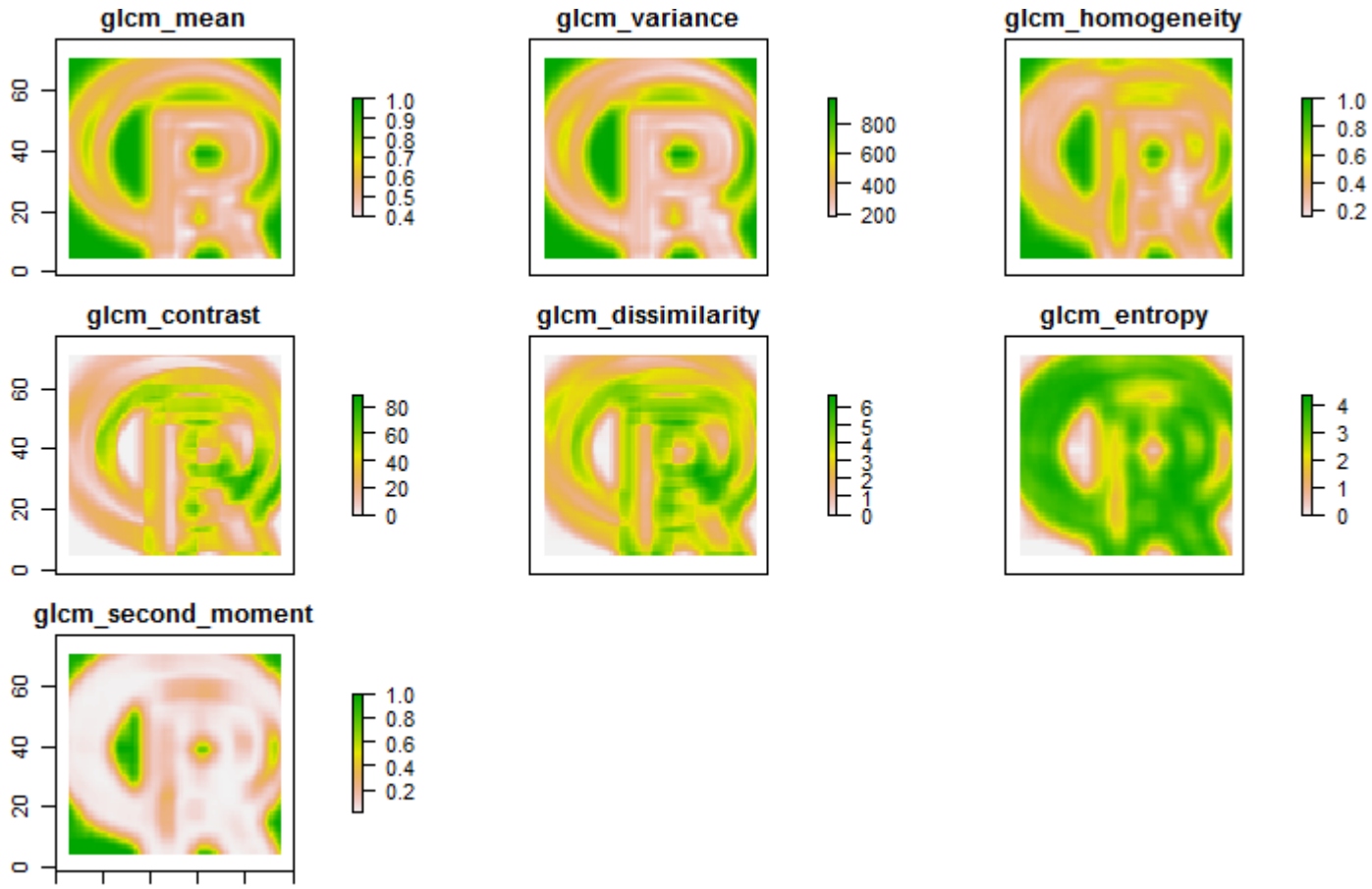


Вычисление вращательно-инвариантных характеристик текстуры

Текстурные особенности также могут быть рассчитаны во всех четырех направлениях (0 °, 45 °, 90 ° и 135 °), а затем объединены с одной вращательно-инвариантной текстурой. Ключом для этого является параметр `shift` :

```
rglcm1 <- glcm(r,
               window = c(9,9),
               shift=list(c(0,1), c(1,1), c(1,0), c(1,-1)),
               statistics = c("mean", "variance", "homogeneity", "contrast",
                              "dissimilarity", "entropy", "second_moment")
               )

plot(rglcm1)
```

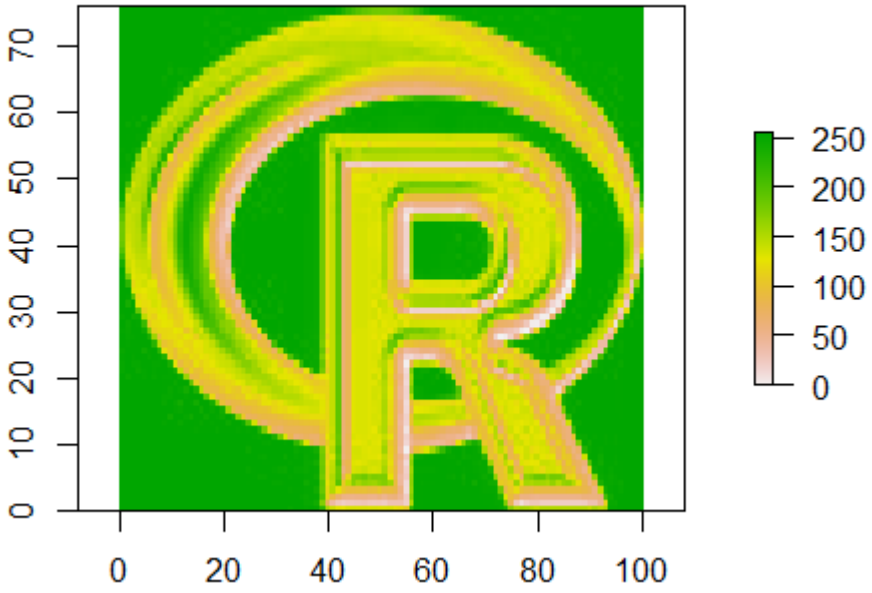


Математические морфологии

Пакет `mmand` предоставляет функции для вычисления математических морфологий для n-мерных массивов. При небольшом обходном пути они также могут быть рассчитаны для растровых изображений.

```
library(raster)
library(mmand)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Сначала ядро (движущееся окно) должно быть задано с размером (например, 9x9) и типом формы (например, `disc` , `box` или `diamond`)

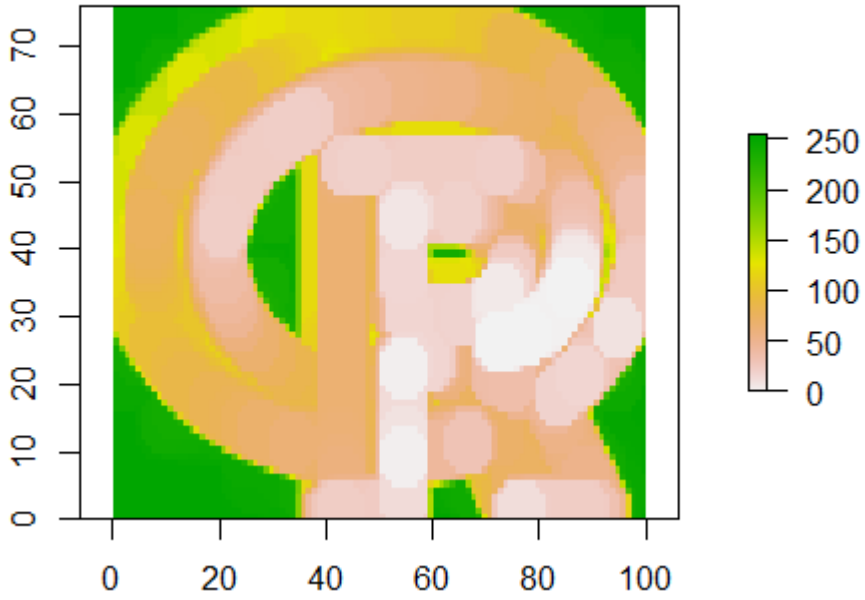
```
sk <- shapeKernel(c(9,9), type="disc")
```

После этого растровый слой должен быть преобразован в массив, который используется как вход для функции `erode()` .

```
rArr <- as.array(r, transpose = TRUE)
rErode <- erode(rArr, sk)
rErode <- setValues(r, as.vector(aperm(rErode)))
```

Помимо `erode()` , также могут применяться морфологические функции `dilate()` , `opening()` и `closing()` .

```
plot(rErode)
```



Прочитайте Растровый и графический анализ онлайн: https://riptutorial.com/ru/r/topic/3726/растровый-и-графический-анализ

глава 100: Регулярные выражения (регулярное выражение)

Вступление

Регулярные выражения (также называемые «регулярное выражение» или «регулярное выражение») определяют шаблоны, которые можно сопоставить с строкой . Введите `?regex` для официальной документации R и просмотрите Документы Regex для более подробной информации. Наиболее важная «gotcha», которая не будет изучена в SO regex / themes, заключается в том, что большинству функций R-regex требуется использование спаренных обратных косых черт для выхода из параметра `pattern` .

замечания

Классы символов

- `"[AB]"` может быть A или B
- `"[[:alpha:]]"` может быть любая буква
- `"[[:lower:]]"` означает любую строчную букву. Обратите внимание, что `"[az]"` близко, но не соответствует, например, `ú` .
- `"[[:upper:]]"` означает любую прописную букву. Обратите внимание, что `"[AZ]"` близко, но не соответствует, например, `Ú` .
- `"[[:digit:]]"` обозначает любую цифру: 0, 1, 2, ... или 9 и эквивалентна `"[0-9]"` .

Кванторы

`+` , `*` и `?` как обычно, в регулярном выражении. - `+` совпадает хотя бы один раз, `*` соответствует 0 или более раз, и `?` соответствует 0 или 1 раз.

Индикаторы начала и конца строки

Вы можете указать положение регулярного выражения в строке:

- `"^..."` заставляет регулярное выражение находиться в начале строки
- `"...$"` заставляет регулярное выражение находиться в конце строки

Различия с другими языками

Обратите внимание, что регулярные выражения в R часто выглядят *так или* иначе отличными от регулярных выражений, используемых на других языках.

- R требует двойных обратных следов (потому что `"\"` уже подразумевает экранирование вообще в R-строках), поэтому, например, для захвата пробелов в большинстве движков регулярных выражений просто нужно ввести `\s` , vs. `\\s` в R ,
- Символы UTF-8 в R должны быть экранированы с капиталом U, например, `[\U{1F600}]` и `[\U1F600]` совпадением , тогда как, например, в Ruby, это будет соответствовать нижнему регистру u.

Дополнительные ресурсы

Следующий сайт reg101 - хорошее место для проверки онлайн-регулярного выражения перед использованием R-скрипта.

В программе программирования Wikibook есть страница, посвященная обработке текста, с множеством примеров с использованием регулярных выражений.

Examples

Устранение пробелов

```
string <- '    some text on line one;
and then some text on line two     '
```

Обрезка пропусков

Пробел «Обрезка» обычно относится к удалению как ведущего, так и конечного пробела из строки. Это можно сделать, используя комбинацию из предыдущих примеров. `gsub` используется для принудительной замены как ведущих, так и конечных совпадений.

До R 3.2.0

```
gsub(pattern = "(^ +| +$)",
     replacement = "",
     x = string)

[1] "some text on line one; \nand then some text on line two"
```

R 3.2.0 и выше

```
trimws(x = string)

[1] "some text on line one; \nand then some text on line two"
```

Удаление ведущих пробелов

До R 3.2.0

```
sub(pattern = "^ +",
    replacement = "",
    x = string)

[1] "some text on line one; \nand then some text on line two     "
```

R 3.2.0 и выше

```
trimws(x = string,
       which = "left")

[1] "some text on line one; \nand then some text on line two     "
```

Удаление пропущенных пробелов

До R 3.2.0

```
sub(pattern = " +$",
    replacement = "",
    x = string)

[1] "    some text on line one; \nand then some text on line two"
```

R 3.2.0 и выше

```
trimws(x = string,
       which = "right")

[1] "    some text on line one; \nand then some text on line two"
```

Удаление всех пробелов

```
gsub(pattern = "\\s",
     replacement = "",
     x = string)

[1] "sometextonlineone;andthensometextonlinetwo"
```

Обратите внимание, что это также удалит символы пробела, такие как вкладки (`\t`), новые строки (`\r` и `\n`) и пробелы.

Проверка даты в формате «YYYYMMDD»

Общепринятой практикой является имя файлов с использованием даты в качестве префикса в следующем формате: `YYYYMMDD` , например: `20170101_results.csv` . Дата в таком строчном формате может быть проверена с использованием следующего регулярного выражения:

```
\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])
```

Вышеприведенное выражение рассматривает даты года: `0000-9999` , месяцев между: `01-12` и днями `01-31` .

Например:

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170101")
[1] TRUE
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20171206")
[1] TRUE
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "29991231")
[1] TRUE
```

Примечание . Он проверяет синтаксис даты, но мы можем иметь неправильную дату с допустимым синтаксисом, например: `20170229` (2017 год - это не високосный год).

```
> grepl("\\d{4}(0[1-9]|1[012])(0[1-9]|[12][0-9]|3[01])", "20170229")
[1] TRUE
```

Если вы хотите проверить дату, это можно сделать с помощью этой функции, определенной пользователем:

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

затем

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE  TRUE  TRUE
```

Подтверждение почтовых абзацев США

Следующее `regex` включает 50 штатов, а также Commonwealth / Territory (см. Www.50states.com):

```
regex <-
"(A[LKSZR])|(C[AOT])|(D[EC])|(F[ML])|(G[AU])|(HI)|(I[DLNA])|(K[SY])|(LA)|(M[EHDAINSOT])|(N[EVHJMYCD])|(M
```

Например:

```
> test <- c("AL", "AZ", "AR", "AJ", "AS", "DC", "FM", "GU","PW", "FL", "AJ", "AP")
```

```
> grepl(us.states.pattern, test)
 [1]  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE
>
```

Примечание .

Если вы хотите проверить только 50 государств, то мы рекомендуем использовать R-`state.abb` : `state.abb` из `state` , например:

```
> data(state)
> test %in% state.abb
[1]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE
```

Мы получаем `TRUE` только для сокращений 50 государств: `AL, AZ, AR, FL` .

Подтвердите номера телефонов США

Следующее регулярное выражение:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

Проверяет номер телефона в виде: `+1-xxx-xxx-xxxx` , включая необязательные ведущие / завершающие пробелы в начале / конце каждой группы чисел, но не посередине, например: `+1-xxx-xxx-xx xx` недействителен. `-` разделитель можно заменить пробелами: `xxx xxx xxx` или без разделителей: `xxxxxxxxxx` . Префикс `+1` является обязательным.

Давайте проверим:

```
us.phones.regex <- "^\\s*(\\+\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"

phones.OK <- c("305-123-4567", "305 123 4567", "+1-786-123-4567",
    "+1 786 123 4567", "7861234567", "786 - 123   4567", "+ 1 786 - 123   4567")

phones.NOK <- c("124-456-78901", "124-456-789", "124-456-78 90",
    "124-45 6-7890", "12 4-456-7890")
```

Действующие случаи:

```
> grepl(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Недействительные случаи:

```
 > grepl(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

Примечание .

- `\\s` Соответствует любому символу пробела, табуляции или символа новой строки

Экранирование символов в шаблонах регулярных выражений R

Поскольку и R, и регулярное выражение разделяют escape-символ, `"\"` , создание правильных шаблонов для `grep` , `sub` , `gsub` или любой другой функции, принимающей аргумент шаблона, часто требует спаривания обратных косых черт. Если вы создаете вектор символа из трех элементов, в котором у одного элемента есть строка, другой символ табуляции и ни один, ни один из них, и hte желание состоит в том, чтобы повернуть либо перевод строки, либо вкладку в 4 пробела, тогда нужна одна обратная косая черта для построения, но смятые обратные косые черты для соответствия:

```
x <- c( "a\nb", "c\td", "e    f")
x  # how it's stored
   #  [1] "a\nb"   "c\td"   "e    f"
cat(x)   # how it will be seen with cat
#a
#b c    d e    f

gsub(patt="\\n|\\t", repl="    ", x)
#[1] "a    b" "c    d" "e    f"
```

Обратите внимание, что аргумент шаблона (который является необязательным, если он появляется первым и требует только частичного написания), является единственным аргументом, требующим этого удвоения или спаривания. Аргумент замены не требует удвоения символов, которые необходимо экранировать. Если вы хотите, чтобы все строки и 4-пространственные вхождения заменялись вкладками, это было бы:

```
gsub("\\n|    ", "\t", x)
#[1] "a\tb" "c\td" "e\tf"
```

Различия между Perl и POSIX regex

В R есть два очень-очень разных двигателя регулярных выражений. Значение по умолчанию называется POSIX-согласованным; все функции регулярных выражений в R также оснащены возможностью включить последний тип: `perl = TRUE` .

Смотри вперед / смотреть-за

`perl = TRUE` позволяет смотреть вперед и смотреть в регулярных выражениях.

- `"(?<=A)B"` соответствует появлению буквы `B` *только в том случае, если* ей предшествует `A` , то есть `"ABACADABRA"` будет соответствовать, но `"abacadabra"` и `"aBacadabra"` не будут.

Прочитайте Регулярные выражения (регулярное выражение) онлайн: https://riptutorial.com/ru/r/topic/5748/регулярные-выражения--регулярное-выражение-

глава 101: Рекомендации по векторизации кода R

Examples

Операциями по строке

Ключ в векторизации кода R состоит в том, чтобы уменьшить или исключить «операции с помощью ряда» или метод отправки функций R.

Это означает, что при приближении к проблеме, которая на первый взгляд требует «операций по строке», например, вычисления средств каждой строки, нужно спросить себя:

- Каковы классы наборов данных, с которыми я имею дело?
- Существует ли существующий скомпилированный код, который может достичь этого без необходимости повторной оценки функций R?
- Если нет, могу ли я выполнить эту операцию столбцами вместо строки?
- Наконец, стоит ли тратить много времени на разработку сложного векторизованную кода вместо того , чтобы просто работает просто `apply` цикл? Другими словами, являются ли данные большими / сложными, что R не может эффективно обрабатывать его с помощью простого цикла?

Отложив проблему выделения памяти и увеличив объект в цикле, в этом примере мы сосредоточимся на том, как избежать `apply` циклов `apply` , отправки методов или переоценки R-функций в циклах.

Стандартный / простой способ вычисления среднего числа строк:

```
apply(mtcars, 1, mean)
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet
Sportabout             Valiant          Duster 360
           29.90727            29.98136            23.59818            38.73955
53.66455            35.04909            59.72000
          Merc 240D            Merc 230            Merc 280           Merc 280C          Merc
450SE          Merc 450SL         Merc 450SLC
           24.63455            27.23364            31.86000            31.78727
46.43091            46.50000            46.35000
 Cadillac Fleetwood Lincoln Continental   Chrysler Imperial            Fiat 128         Honda
Civic      Toyota Corolla       Toyota Corona
           66.23273            66.05855            65.97227            19.44091
17.74227            18.81409            24.88864
   Dodge Challenger         AMC Javelin          Camaro Z28    Pontiac Firebird           Fiat
X1-9       Porsche 914-2        Lotus Europa
           47.24091            46.00773            58.75273            57.37955
18.92864            24.77909            24.88027
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
```

```
          60.97182            34.50818            63.15545            26.26273
```

Но можем ли мы сделать лучше? Давайте посмотрим, что здесь произошло:

1. Сначала мы преобразовали `data.frame` в `matrix` . (Обратите внимание, что он происходит внутри функции `apply` .) Это неэффективно и опасно. `matrix` не может содержать несколько типов столбцов за раз. Следовательно, такое преобразование, вероятно, приведет к потере информации и несколько раз к вводящим в заблуждение результатам (сравнить `apply(iris, 2, class)` с `str(iris)` или с `sapply(iris, class)`).
2. Во-вторых, мы выполняли операцию повторно, один раз для каждой строки. Смысл, нам пришлось оценить некоторые R функции `nrow(mtcars)` раз. В этом конкретном случае `mean` не является дорогостоящей функцией, поэтому R может, вероятно, легко справиться с этим даже для большого набора данных, но что произойдет, если нам нужно вычислить стандартное отклонение по строке (что связано с дорогостоящей работой с квадратным корнем) ? Это подводит нас к следующему пункту:
3. Мы много раз оценивали функцию R, но, возможно, уже есть скомпилированная версия этой операции?

Действительно, мы могли бы просто сделать:

```
rowMeans(mtcars)
          Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive   Hornet
Sportabout             Valiant          Duster 360
           29.90727            29.98136            23.59818            38.73955
53.66455            35.04909            59.72000
          Merc 240D            Merc 230            Merc 280           Merc 280C          Merc
450SE          Merc 450SL         Merc 450SLC
           24.63455            27.23364            31.86000            31.78727
46.43091            46.50000            46.35000
 Cadillac Fleetwood Lincoln Continental   Chrysler Imperial            Fiat 128         Honda
Civic      Toyota Corolla       Toyota Corona
           66.23273            66.05855            65.97227            19.44091
17.74227            18.81409            24.88864
   Dodge Challenger         AMC Javelin          Camaro Z28    Pontiac Firebird           Fiat
X1-9       Porsche 914-2        Lotus Europa
           47.24091            46.00773            58.75273            57.37955
18.92864            24.77909            24.88027
     Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
           60.97182            34.50818            63.15545            26.26273
```

Это не связано с операциями по строке и, следовательно, повторной оценкой R-функций. **Тем не менее** , мы по-прежнему преобразовали `data.frame` в `matrix` . Хотя `rowMeans` имеет механизм обработки ошибок, и он не будет работать в наборе данных, который он не может обрабатывать, он по-прежнему имеет эффективность.

```
rowMeans(iris)
Error in rowMeans(iris) : 'x' must be numeric
```

Но все-таки, можем ли мы сделать лучше? Мы могли бы попробовать вместо преобразования матрицы с обработкой ошибок, другой метод, который позволит нам

использовать `mtcars` в качестве вектора (поскольку `data.frame` по существу является `list` а `list` - `vector`).

```
Reduce(`+`, mtcars)/ncol(mtcars)
 [1] 29.90727 29.98136 23.59818 38.73955 53.66455 35.04909 59.72000 24.63455 27.23364 31.86000
31.78727 46.43091 46.50000 46.35000 66.23273 66.05855
[17] 65.97227 19.44091 17.74227 18.81409 24.88864 47.24091 46.00773 58.75273 57.37955 18.92864
24.77909 24.88027 60.97182 34.50818 63.15545 26.26273
```

Теперь для возможного увеличения скорости мы потеряли имена столбцов и обработку ошибок (включая обработку `NA`).

Другим примером может быть вычисление среднего значения по группе, используя базу R, которую мы могли бы попробовать

```
aggregate(. ~ cyl, mtcars, mean)
cyl      mpg     disp        hp     drat       wt     qsec        vs        am     gear
carb
1   4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
2   6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
3   8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

Тем не менее, мы в основном оцениваем R-функцию в цикле, но цикл теперь скрыт во внутренней C-функции (мало что имеет в виду, является ли это C или R-петлей).

Мы могли бы избежать этого? Ну есть скомпилированная функция в R, называемая `rowsum` , поэтому мы могли бы сделать:

```
rowsum(mtcars[-2], mtcars$cyl)/table(mtcars$cyl)
mpg     disp        hp     drat       wt     qsec        vs        am     gear     carb
4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455
6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571
8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

Хотя нам пришлось сначала преобразовать в матрицу.

В этом вопросе мы можем задать вопрос о том, является ли наша нынешняя структура данных наиболее подходящей. Является ли `data.frame` лучшей практикой? Или нужно просто переключиться на `matrix` структуру данных, чтобы повысить эффективность?

Порядковые операции будут становиться все более и более дорогостоящими (даже в матрицах), когда мы начинаем оценивать дорогостоящие функции каждый раз. Давайте рассмотрим пример вычисления дисперсии по строке.

Допустим, мы имеем матрицу `m` :

```
set.seed(100)
m <- matrix(sample(1e2), 10)
m
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    8   33   39   86   71  100   81   68   89    84
 [2,]   12   16   57   80   32   82   69   11   41    92
 [3,]   62   91   53   13   42   31   60   70   98    79
 [4,]   66   94   29   67   45   59   20   96   64     1
 [5,]   36   63   76    6   10   48   85   75   99     2
 [6,]   18    4   27   19   44   56   37   95   26    40
 [7,]    3   24   21   25   52   51   83   28   49    17
 [8,]   46    5   22   43   47   74   35   97   77    65
 [9,]   55   54   78   34   50   90   30   61   14    58
[10,]   88   73   38   15    9   72    7   93   23    87
```

Можно просто сделать:

```
apply(m, 1, var)
[1]  871.6556  957.5111  699.2111  941.4333 1237.3333  641.8222  539.7889  759.4333  500.4889
1255.6111
```

С другой стороны, можно было бы полностью векторизовать эту операцию, следуя формуле дисперсии

```
RowVar <- function(x) {
  rowSums((x - rowMeans(x))^2)/(dim(x)[2] - 1)
}
RowVar(m)
[1]  871.6556  957.5111  699.2111  941.4333 1237.3333  641.8222  539.7889  759.4333  500.4889
1255.6111
```

Прочитайте Рекомендации по векторизации кода R онлайн:
https://riptutorial.com/ru/r/topic/3327/рекомендации-по-векторизации-кода-r

глава 102: Решение ODE в R

Синтаксис

- ode (y, times, func, parms, method, ...)

параметры

параметр	подробности
Y	(named) числовой вектор: начальные (состояния) значения для системы ODE
раз	временная последовательность, для которой требуется выход; первое значение времени должно быть начальным
FUNC	имя функции, которая вычисляет значения производных в системе ODE
Parms	(named) числовой вектор: параметры, переданные func
метод	интегратор использовать по умолчанию: lsoda

замечания

Обратите внимание, что необходимо вернуть скорость изменения в том же порядке, что и спецификация переменных состояния. В примере «Модель Лоренца» это означает, что в функции «Лоренц»

```
return(list(c(dX, dY, dZ)))
```

имеет тот же порядок, что и определение переменных состояния

```
yini <- c(X = 1, Y = 1, Z = 1)
```

Examples

Модель Лоренца

Модель Лоренца описывает динамику трех переменных состояния: X, Y и Z. Модельными уравнениями являются:

$$\frac{dX}{dt} = a \cdot X + Y \cdot Z$$

$$\frac{dY}{dt} = b \cdot (Y - Z)$$

$$\frac{dZ}{dT} = -X \cdot Y + c \cdot Y - Z$$

Исходными условиями являются:

$$X(0) = Y(0) = Z(0) = 1$$

и a, b и c - три параметра с

$$a = -8/3$$

$$b = -10$$

$$c = 28$$

```
library(deSolve)

## ------------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------------

Lorenz <- function (t, y, parms) {
  with(as.list(c(y, parms)), {
    dX <- a * X + Y * Z
    dY <- b * (Y - Z)
    dZ <- -X * Y + c * Y - Z

    return(list(c(dX, dY, dZ)))
  })
}

## ------------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------------

parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)

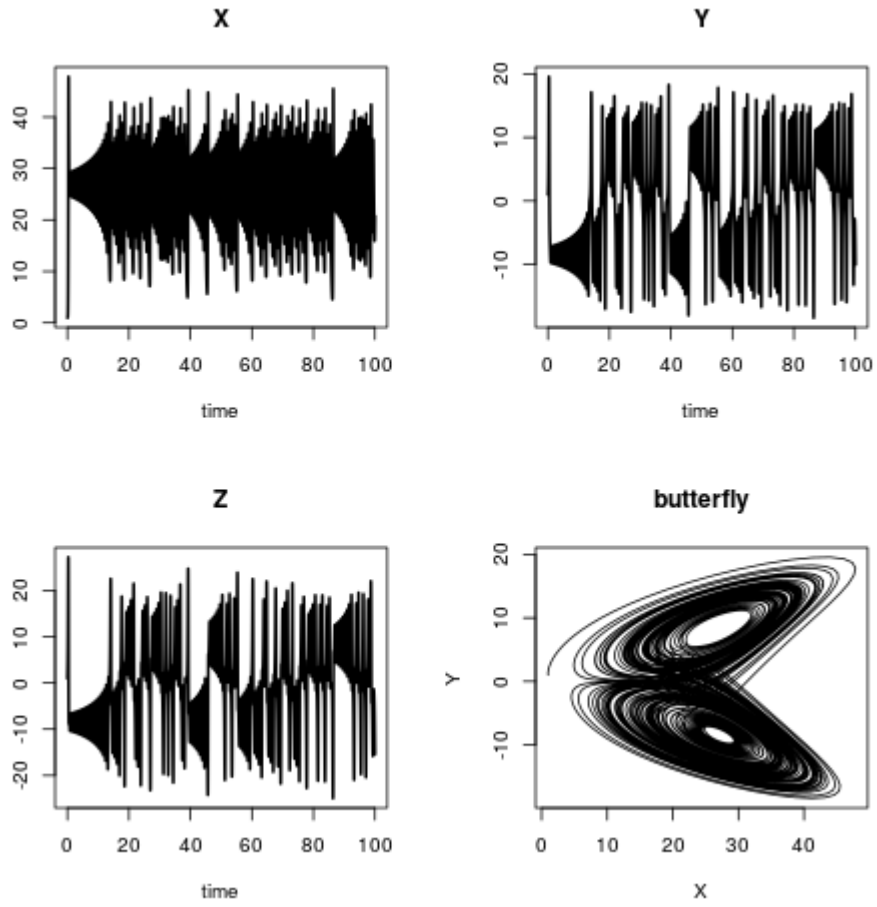

## ------------------------------------------------------------------------------
## Solve the ODEs
## ------------------------------------------------------------------------------

out <- ode(y = yini, times = times, func = Lorenz, parms = parms)

## ------------------------------------------------------------------------------
## Plot the results
## ------------------------------------------------------------------------------

plot(out, lwd = 2)
plot(out[,"X"], out[,"Y"],
     type = "l", xlab = "X",
```

```
     ylab = "Y", main = "butterfly")
```



Лотка-Вольтерра или: жертва против хищника

```
library(deSolve)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

LV <- function(t, y, parms) {
    with(as.list(c(y, parms)), {

        dP <- rG * P * (1 - P/K) - rI * P * C
        dC <- rI * P * C * AE - rM * C

        return(list(c(dP, dC), sum = C+P))
    })
}

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

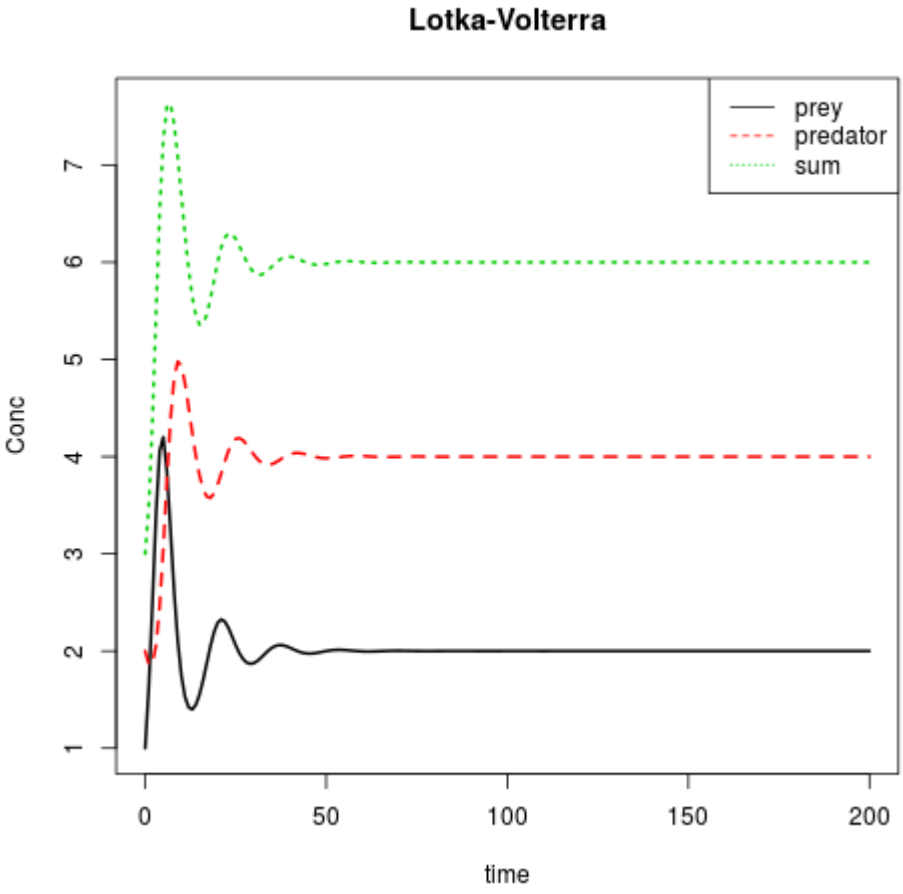
parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)
```

```
## ------------------------------------------------------------------------
## Solve the ODEs
## ------------------------------------------------------------------------

out <- ode(y = yini, times = times, func = LV, parms = parms)

## ------------------------------------------------------------------------
## Plot the results
## ------------------------------------------------------------------------

matplot(out[ ,1], out[ ,2:4], type = "l", xlab = "time", ylab = "Conc",
        main = "Lotka-Volterra", lwd = 2)
legend("topright", c("prey", "predator", "sum"), col = 1:3, lty = 1:3)
```



ODE в скомпилированных языках - определение в R

```
library(deSolve)

## ------------------------------------------------------------------------
## Define parameters and variables
## ------------------------------------------------------------------------

eps <- 0.01;
M <- 10
k <- M * eps^2/2
L <- 1
L0 <- 0.5
r <- 0.1
```

```
w <- 10
g <- 1

parameter <- c(eps = eps, M = M, k = k, L = L, L0 = L0, r = r, w = w, g = g)

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
          ul = -L0/L, vl = 0,
          ur = -L0/L, vr = 0,
          lam1 = 0, lam2 = 0)


times <- seq(from = 0, to = 3, by = 0.01)

## ------------------------------------------------------------------------
## Define R-function
## ------------------------------------------------------------------------

caraxis_R <- function(t, y, parms) {
  with(as.list(c(y, parms)), {

    yb <- r * sin(w * t)
    xb <- sqrt(L * L - yb * yb)
    Ll <- sqrt(xl^2 + yl^2)
    Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

    dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

    dul  <- (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
    dvl  <- (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k * g

    dur  <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
    dvr  <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

    c1   <- xb * xl + yb * yl
    c2   <- (xl - xr)^2 + (yl - yr)^2 - L * L

    return(list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2)))
  })
}
```

ODE в скомпилированных языках - определение в C

```
sink("caraxis_C.c")
cat("
/* suitable names for parameters and state variables */

#include <R.h>
#include <math.h>
static double parms[8];

#define eps parms[0]
#define m   parms[1]
#define k   parms[2]
#define L   parms[3]
#define L0  parms[4]
#define r   parms[5]
#define w   parms[6]
#define g   parms[7]
```

```
/*-----------------------------------------------------------------------
 initialising the parameter common block
-----------------------------------------------------------------------
*/
void init_C(void (* daeparms)(int *, double *)) {
  int N = 8;
  daeparms(&N, parms);
    }
/* Compartments */

#define xl y[0]
#define yl y[1]
#define xr y[2]
#define yr y[3]
#define lam1 y[8]
#define lam2 y[9]

/*-----------------------------------------------------------------------
 the residual function
-----------------------------------------------------------------------
*/
void caraxis_C (int *neq, double *t, double *y, double *ydot,
              double *yout, int* ip)
{
  double yb, xb, Lr, Ll;

  yb  = r * sin(w * *t) ;
  xb  = sqrt(L * L - yb * yb);
  Ll  = sqrt(xl * xl + yl * yl) ;
  Lr  = sqrt((xr-xb)*(xr-xb) + (yr-yb)*(yr-yb));

  ydot[0] = y[4];
  ydot[1] = y[5];
  ydot[2] = y[6];
  ydot[3] = y[7];

  ydot[4] = (L0-Ll) * xl/Ll + lam1*xb + 2*lam2*(xl-xr)    ;
  ydot[5] = (L0-Ll) * yl/Ll + lam1*yb + 2*lam2*(yl-yr) - k*g;
  ydot[6] = (L0-Lr) * (xr-xb)/Lr      - 2*lam2*(xl-xr)       ;
  ydot[7] = (L0-Lr) * (yr-yb)/Lr      - 2*lam2*(yl-yr) - k*g   ;

  ydot[8]  = xb * xl + yb * yl;
  ydot[9]  = (xl-xr) * (xl-xr) + (yl-yr) * (yl-yr) - L*L;

}
", fill = TRUE)
sink()
system("R CMD SHLIB caraxis_C.c")
dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))
dllname_C <- dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))[[1]]
```

ODE в скомпилированных языках - определение fortran

```
sink("caraxis_fortran.f")
cat("
c-----------------------------------------------------------------
c Initialiser for parameter common block
c-----------------------------------------------------------------
       subroutine init_fortran(daeparms)
```

```
        external daeparms
        integer, parameter :: N = 8
        double precision parms(N)
        common /myparms/parms

        call daeparms(N, parms)
        return
        end

c-----------------------------------------------------------------------
c rate of change
c-----------------------------------------------------------------------
        subroutine caraxis_fortran(neq, t, y, ydot, out, ip)
        implicit none
        integer          neq, IP(*)
        double precision t, y(neq), ydot(neq), out(*)
        double precision eps, M, k, L, L0, r, w, g
        common /myparms/ eps, M, k, L, L0, r, w, g

        double precision xl, yl, xr, yr, ul, vl, ur, vr, lam1, lam2
        double precision yb, xb, Ll, Lr, dxl, dyl, dxr, dyr
        double precision dul, dvl, dur, dvr, c1, c2

c expand state variables
         xl = y(1)
         yl = y(2)
         xr = y(3)
         yr = y(4)
         ul = y(5)
         vl = y(6)
         ur = y(7)
         vr = y(8)
         lam1 = y(9)
         lam2 = y(10)

         yb = r * sin(w * t)
         xb = sqrt(L * L - yb * yb)
         Ll = sqrt(xl**2 + yl**2)
         Lr = sqrt((xr - xb)**2 + (yr - yb)**2)

         dxl = ul
         dyl = vl
         dxr = ur
         dyr = vr

         dul = (L0-Ll) * xl/Ll      + 2 * lam2 * (xl-xr) + lam1*xb
         dvl = (L0-Ll) * yl/Ll      + 2 * lam2 * (yl-yr) + lam1*yb - k*g
         dur = (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
         dvr = (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k*g

         c1  = xb * xl + yb * yl
         c2  = (xl - xr)**2 + (yl - yr)**2 - L * L

c function values in ydot
         ydot(1)  = dxl
         ydot(2)  = dyl
         ydot(3)  = dxr
         ydot(4)  = dyr
         ydot(5)  = dul
         ydot(6)  = dvl
```

```
         ydot(7)  = dur
         ydot(8)  = dvr
         ydot(9)  = c1
         ydot(10) = c2
        return
        end
", fill = TRUE)

sink()
system("R CMD SHLIB caraxis_fortran.f")
dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))
dllname_fortran <- dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))[[1]]
```

ODE на скомпилированных языках - тестовый тест

Когда вы скомпилировали и загрузили код в трех примерах раньше (ODE в скомпилированных языках - определение в R, ODEs в скомпилированных языках - определение в C и ODE в скомпилированных языках - определение в fortran), вы можете запустить тестовый тест.

```
library(microbenchmark)

R <- function(){
  out <- ode(y = yini, times = times, func = caraxis_R,
             parms = parameter)
}


C <- function(){
  out <- ode(y = yini, times = times, func = "caraxis_C",
             initfunc = "init_C", parms = parameter,
             dllname = dllname_C)
}

fortran <- function(){
  out <- ode(y = yini, times = times, func = "caraxis_fortran",
             initfunc = "init_fortran", parms = parameter,
             dllname = dllname_fortran)
}
```

Проверьте, равны ли результаты:

```
all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])
```

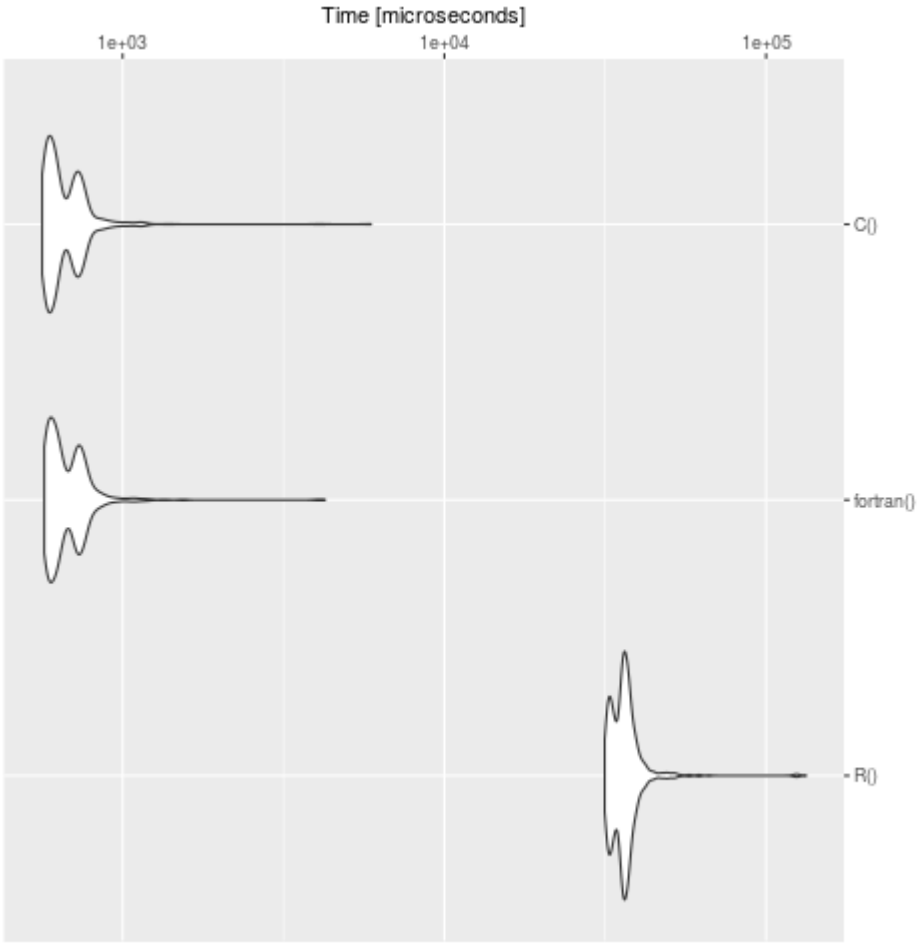
Сделайте контрольную отметку (Примечание: на вашей машине времена, разумеется, разные):

```
bench <- microbenchmark::microbenchmark(
  R(),
  fortran(),
  C(),
  times = 1000
```

```
)

summary(bench)

     expr         min        lq       mean     median         uq        max neval cld
      R()   31508.928 33651.541 36747.8733 36062.2475 37546.8025 132996.564  1000   b
fortran()     570.674   596.700   686.1084   637.4605   730.1775   4256.555  1000  a
      C()     562.163   590.377   673.6124   625.0700   723.8460   5914.347  1000  a
```



Мы ясно видим, что R медленнее, чем определение в C и fortran. Для больших моделей стоит перевести проблему на скомпилированный язык. Пакет `cOde` является одной из возможностей перевода ODE с R на C.

Прочитайте Решение ODE в R онлайн: https://riptutorial.com/ru/r/topic/7448/решение-ode-в-r

глава 103: Серия Фурье и преобразования

замечания

Преобразование Фурье разлагает функцию времени (сигнала) на частоты, которые составляют ее, подобно тому, как музыкальный аккорд может быть выражен как амплитуда (или громкость) его составляющих заметок. Преобразование Фурье функции самого времени представляет собой комплекснозначную функцию частоты, абсолютная величина которой представляет собой величину этой частоты, присутствующую в исходной функции, и сложным аргументом которой является смещение фазы основной синусоиды на этой частоте.

Преобразование Фурье называется представлением частотной области исходного сигнала. Термин преобразование Фурье относится как к представлению частотной области, так и к математической операции, которая связывает представление частотной области с функцией времени. Преобразование Фурье не ограничено функциями времени, но для того, чтобы иметь унифицированный язык, область исходной функции обычно называется временной областью. Для многих практических задач можно определить операцию, которая меняет это: обратное преобразование Фурье, также называемое синтезом Фурье, представления частотной области объединяет вклады всех разных частот для восстановления исходной функции времени.

Линейные операции, выполняемые в одном домене (время или частота), имеют соответствующие операции в другом домене, которые иногда легче выполнять. Операция дифференцирования во временной области соответствует умножению на частоту, поэтому некоторые дифференциальные уравнения легче анализировать в частотной области. Кроме того, свертка во временной области соответствует обычным умножениям в частотной области. Конкретно это означает, что любая линейная система, зависящая от времени, такая как электронный фильтр, применяемый к сигналу, может быть выражена относительно просто как операция на частотах. Поэтому значительное упрощение часто достигается путем преобразования временных функций в частотную область, выполнения желаемых операций и преобразования результата во времени.

Гармонический анализ - это систематическое исследование взаимосвязи между частотной и временной областями, в том числе виды функций или операций, которые «проще» в одном или другом, и имеют глубокие связи практически со всеми областями современной математики.

Функции, локализованные во временной области, имеют преобразования Фурье, которые распространяются по частотной области и наоборот. Критическим случаем является гауссова функция, имеющая существенное значение в теории вероятностей и статистике, а также при изучении физических явлений, имеющих нормальное распределение (например,

диффузию), которое при соответствующих нормализациях переходит в себя при преобразовании Фурье. Джозеф Фурье представил преобразование в своем исследовании теплообмена, где гауссовы функции появляются как решения уравнения теплопроводности.

Преобразование Фурье может быть формально определено как несобственный интеграл Римана, что делает его интегральным преобразованием, хотя это определение не подходит для многих приложений, требующих более сложной теории интегрирования.

Например, многие относительно простые приложения используют дельта-функцию Дирака, которую можно рассматривать формально, как если бы это была функция, но для обоснования требуется математически более сложная точка зрения. Преобразование Фурье также может быть обобщено на функции нескольких переменных в евклидовом пространстве, посылая функцию 3-мерного пространства на функцию трехмерного импульса (или функцию пространства и времени на функцию 4-импульса).

Эта идея делает пространственное преобразование Фурье очень естественным в изучении волн, а также в квантовой механике, где важно иметь возможность представлять волновые решения либо как функции пространства, так и импульса, а иногда и то и другое. В общем случае функции, к которым применимы методы Фурье, являются комплекснозначными и, возможно, векторными. Еще одно обобщение возможно для функций на группах, которые, помимо исходного преобразования Фурье на $\mathbb{R}$ или $\mathbb{R}n$ (рассматриваются как группы при добавлении), в частности включают в себя преобразование Фурье с дискретным временем (DTFT, group = $\mathbb{Z}$), дискретное преобразование Фурье (DFT, group = $\mathbb{Z} \bmod N$) и ряд Фурье или круговое преобразование Фурье (группа = S1, единичный круг ≈ замкнутый конечный интервал с идентифицированными оконечными точками). Последнее обычно используется для обработки периодических функций. Быстрое преобразование Фурье (БПФ) является алгоритмом для вычисления ДПФ.

Examples

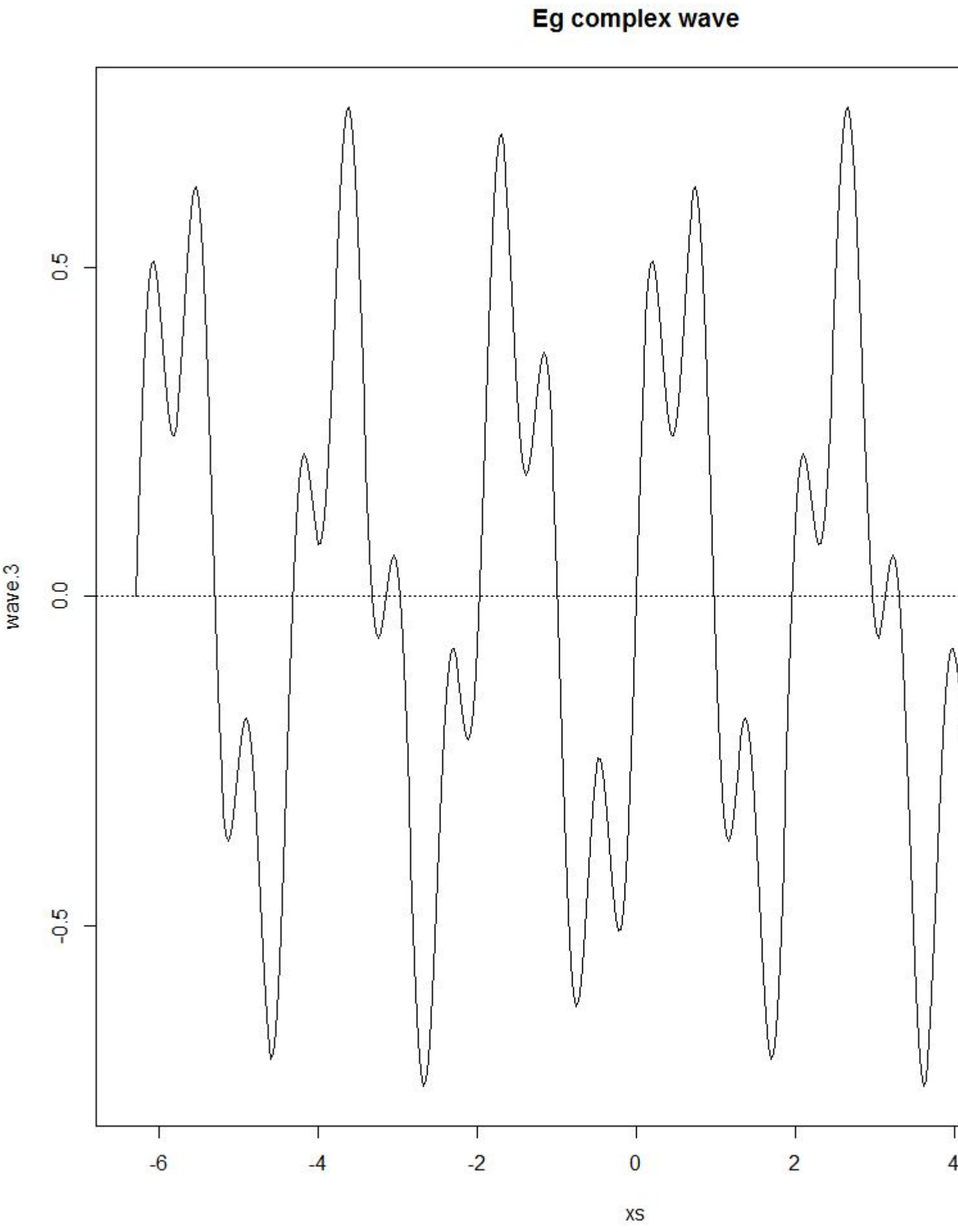
Серия Фурье

Джозеф Фурье показал, что любая периодическая волна может быть представлена суммой простых синусоидальных волн. Эта сумма называется рядами Фурье. Ряд Фурье выполняется только тогда, когда система линейна. Если есть, например, некоторый эффект переполнения (порог, где выход остается неизменным независимо от того, сколько вводится данных), нелинейный эффект входит в изображение, нарушая синусоидальную волну и принцип суперпозиции.

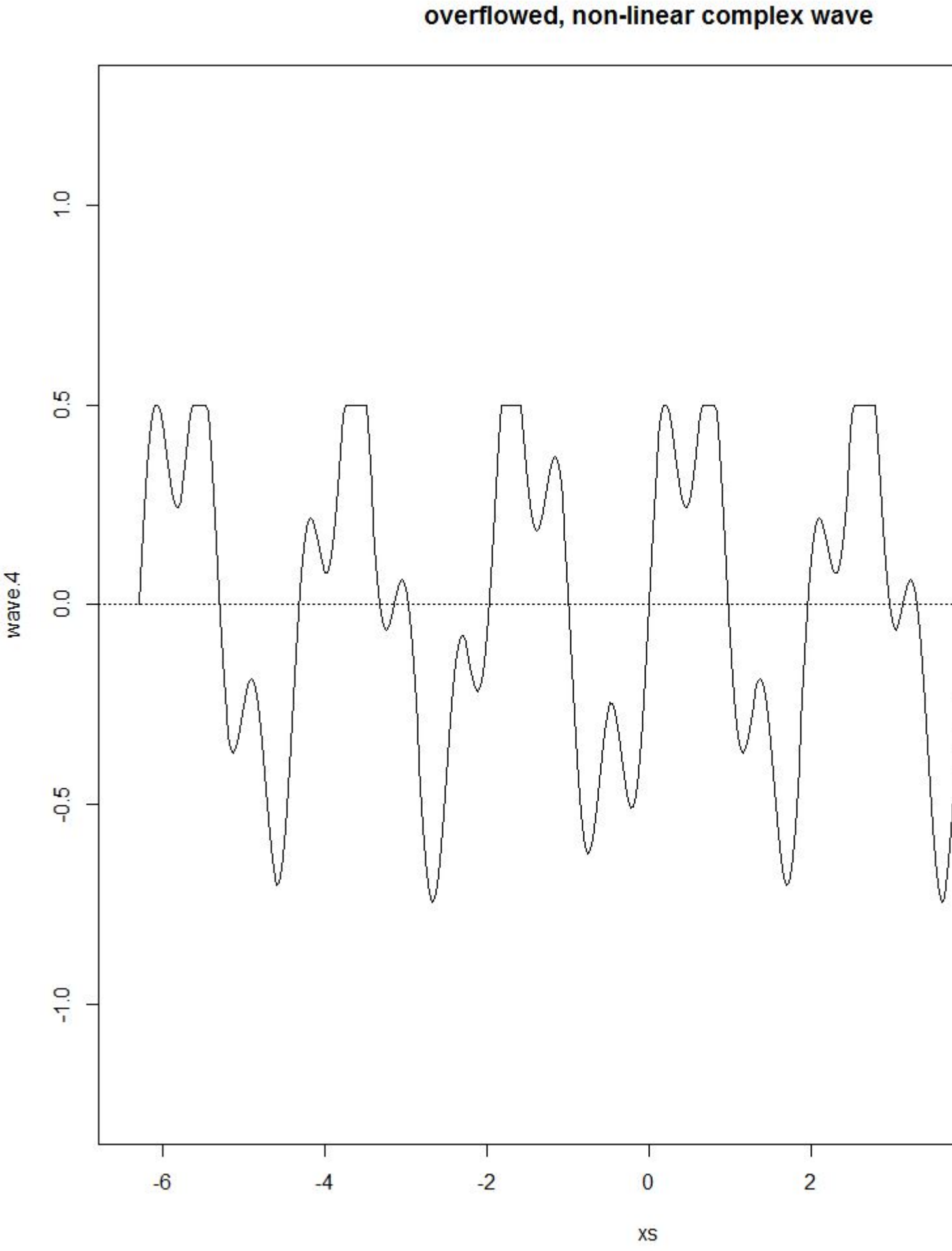
```
# Sine waves
xs <- seq(-2*pi,2*pi,pi/100)
wave.1 <- sin(3*xs)
wave.2 <- sin(10*xs)
```

```
par(mfrow = c(1, 2))
plot(xs,wave.1,type="l",ylim=c(-1,1)); abline(h=0,lty=3)
plot(xs,wave.2,type="l",ylim=c(-1,1)); abline(h=0,lty=3)

# Complex Wave
wave.3 <- 0.5 * wave.1 + 0.25 * wave.2
plot(xs,wave.3,type="l"); title("Eg complex wave"); abline(h=0,lty=3)
```


Eg complex wave
wave.3
0.5
0.0
-0.5
-6
-4
-2
0
2
4
xs

```
wave.4 <- wave.3
wave.4[wave.3>0.5] <- 0.5
plot(xs,wave.4,type="l",ylim=c(-1.25,1.25))
title("overflowed, non-linear complex wave")
abline(h=0,lty=3)
```


overflowed, non-linear complex wave
wave.4
1.0
0.5
0.0
-0.5
-1.0
-6
-4
-2
0
2
xs

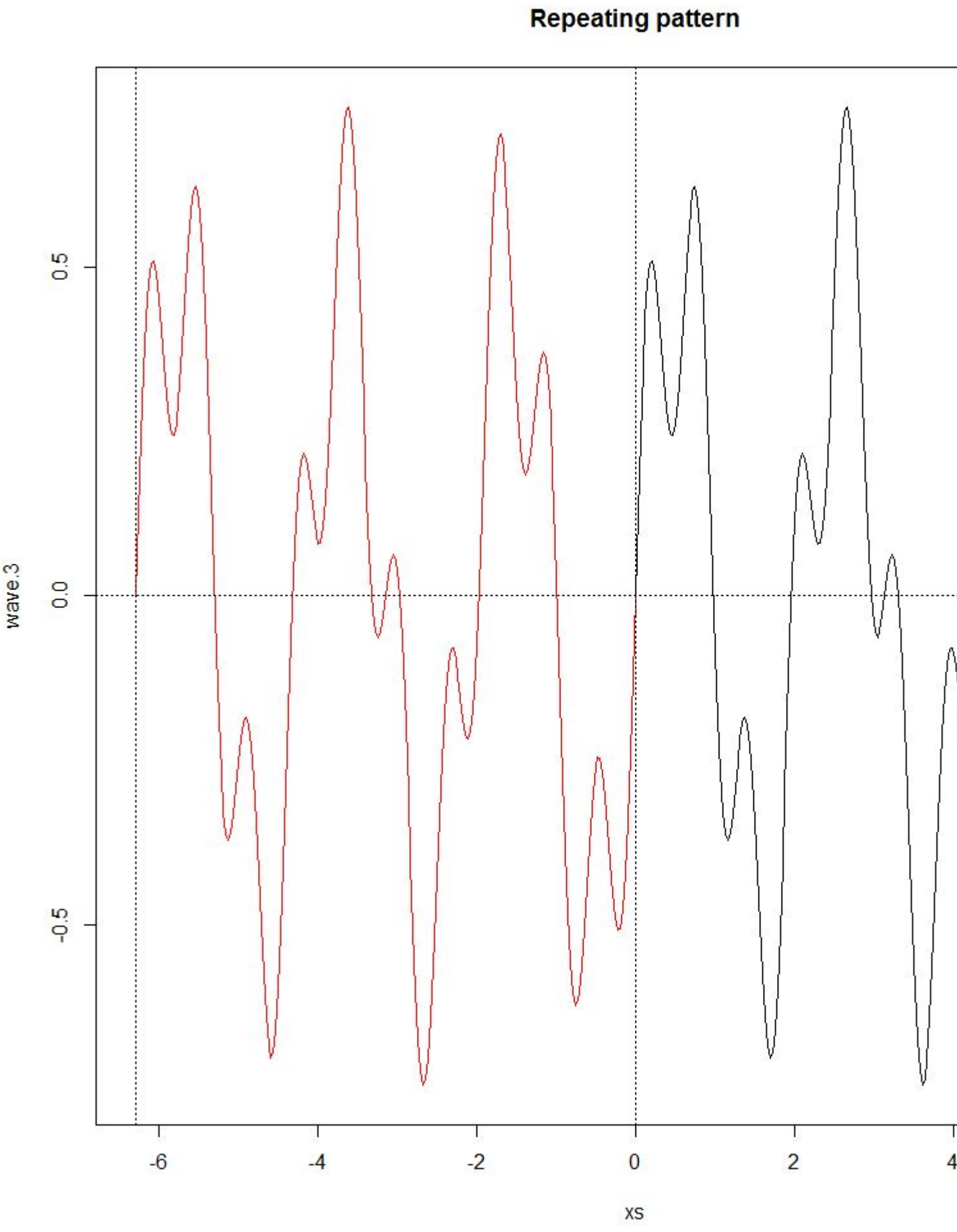
Кроме того, ряд Фурье выполняется только в том случае, если волны являются периодическими, т. Е. Имеют повторяющийся паттерн (непериодические волны обрабатываются преобразованием Фурье, см. Ниже). Периодическая волна имеет частоту f и длину волны λ (длина волны - это расстояние в среде между началом и концом цикла, λ = v / f0, где v - скорость волны), которые определяются повторяющимся рисунком. Непериодическая волна не имеет частоты или длины волны.

Некоторые концепции:

- Фундаментальный период T - период всех взятых проб, время между первым и последним
- Частота дискретизации, sr, - это количество выборок, взятых за период времени (ака частота сбора). Для простоты мы сделаем временной интервал между образцами равными. Этот интервал времени называется интервалом выборки, si, который является фундаментальным периодом времени, деленным на количество выборок N. Итак, si = TN
- Основная частота f0, равная 1T. Основная частота - это частота повторяющегося шаблона или длина волны. В предыдущих волнах основная частота составляла 12π. Частоты компонентов волны должны быть целыми кратными основной частоте. f0 называется первой гармоникой, вторая гармоника 2 * f0, третья - 3 * f0 и т. д.

```
repeat.xs     <- seq(-2*pi,0,pi/100)
wave.3.repeat <- 0.5*sin(3*repeat.xs) + 0.25*sin(10*repeat.xs)
plot(xs,wave.3,type="l")

title("Repeating pattern")
points(repeat.xs,wave.3.repeat,type="l",col="red");
abline(h=0,v=c(-2*pi,0),lty=3)
```


Repeating pattern
wave.3
0.5
0.0
-0.5
-6
-4
-2
0
2
4
xs

Вот функция R для построения траекторий с заданной последовательностью Фурье:

```
plot.fourier <- function(fourier.series, f.0, ts) {
                        w <- 2*pi*f.0 trajectory <- sapply(ts, function(t)
fourier.series(t,w))
                        plot(ts, trajectory, type="l", xlab="time", ylab="f(t)");
                        abline(h=0,lty=3)}
```

Прочитайте Серия Фурье и преобразования онлайн: https://riptutorial.com/ru/r/topic/4139/серия-фурье-и-преобразования

глава 104: Сетевой анализ с пакетом igraph

Examples

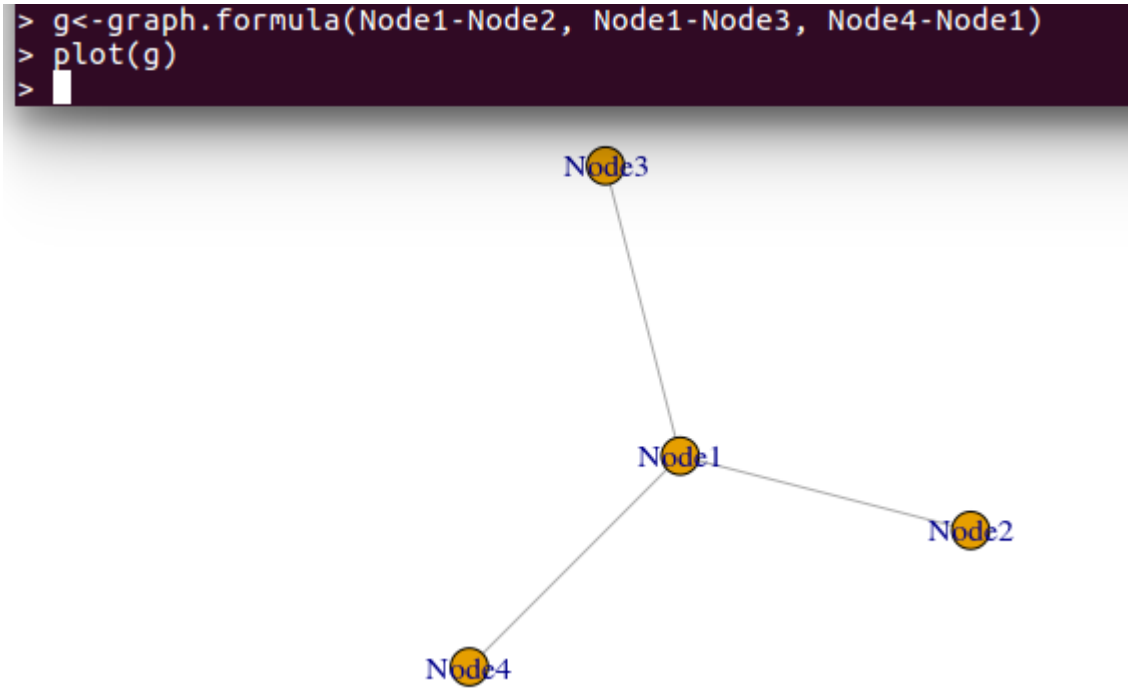
Простая директивная и ненаправленная сетевая графика

Пакет igraph для R - прекрасный инструмент, который можно использовать для моделирования сетей, как реальных, так и виртуальных, с простотой. Этот пример предназначен для демонстрации того, как создать два простых сетевых графика, используя пакет igraph в R v.3.2.3.

Неправленная сеть

Сеть создается с помощью этого фрагмента кода:

```
g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
plot(g)
```

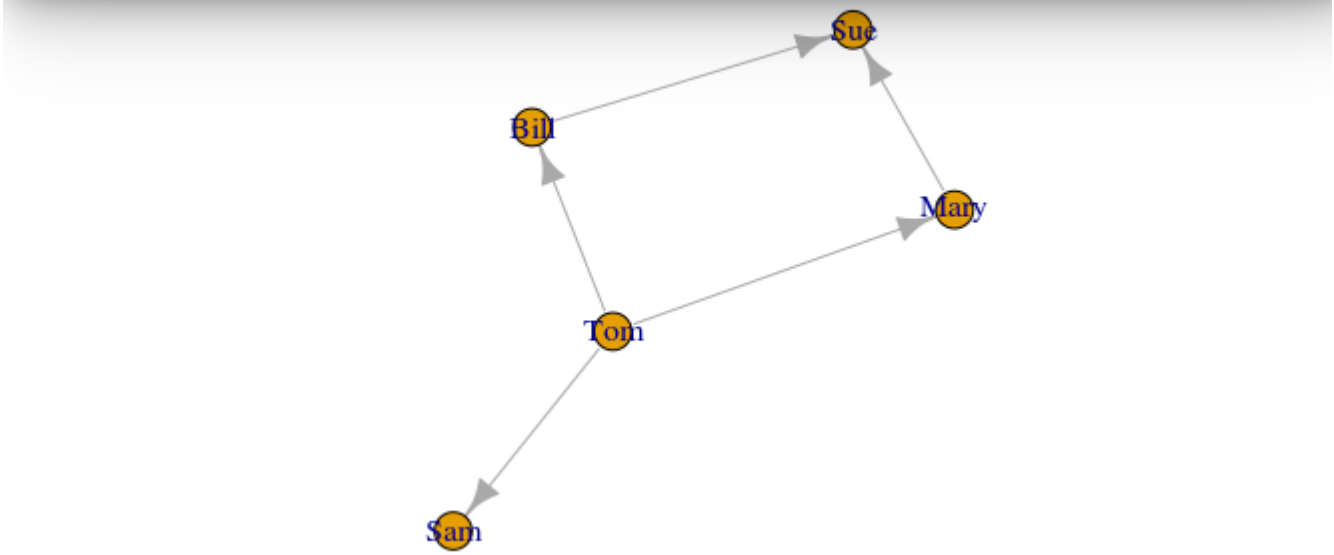


Направленная сеть

```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
plot(dg)
```

Затем этот код генерирует сеть со стрелками:

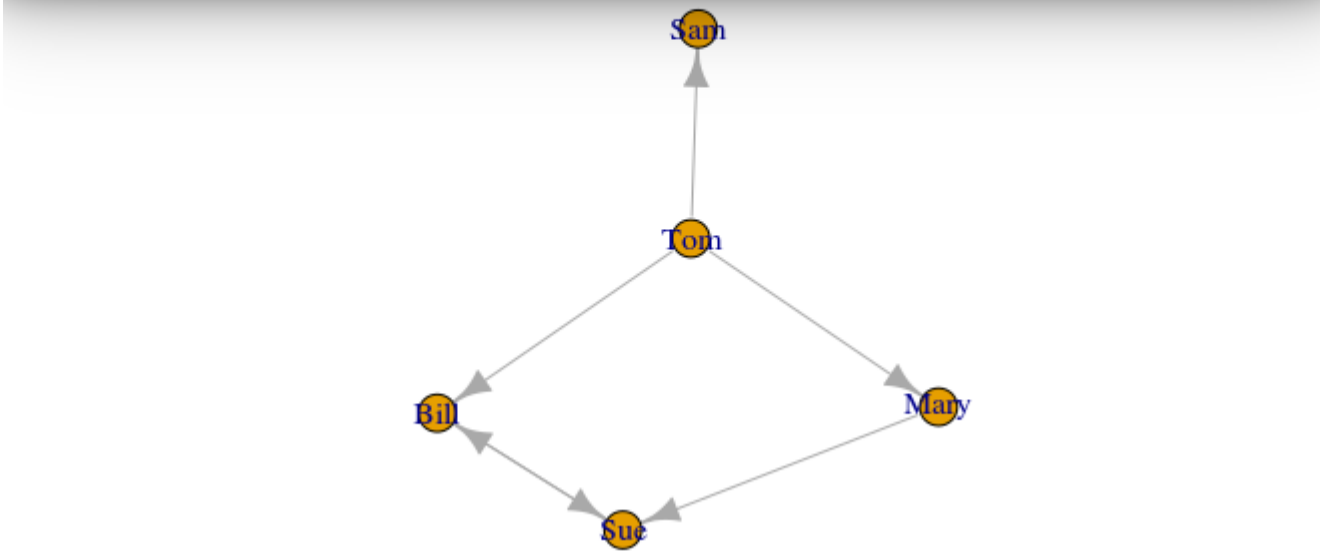
```
> dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill-+Sue)
> plot(dg)
>
```



Пример кода, как сделать двустороннюю стрелку:

```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill++Sue)
plot(dg)
```

```
> dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue+-Mary, Bill++Sue)
> plot(dg)
>
```



Прочитайте Сетевой анализ с пакетом igraph онлайн: https://riptutorial.com/ru/r/topic/4851/сетевой-анализ-с-пакетом-igraph

глава 105: Синтаксис регулярного выражения в R

Вступление

В этом документе представлены основы регулярных выражений, используемых в R. Для получения дополнительной информации о синтаксисе регулярных выражений R см. « `?regex` . Полный список операторов регулярного выражения см. В этом руководстве ICU для регулярных выражений .

Examples

Используйте `grep`, чтобы найти строку в символьном векторе

```
# General syntax:
# grep(<pattern>, <character vector>)

mystring <- c('The number 5',
              'The number 8',
              '1 is the loneliest number',
              'Company, 3 is',
              'Git SSH tag is git@github.com',
              'My personal site is www.personal.org',
              'path/to/my/file')

grep('5', mystring)
# [1] 1
grep('@', mystring)
# [1] 5
grep('number', mystring)
# [1] 1 2 3
```

`x|y` означает поиск «x» или «y»,

```
grep('5|8', mystring)
# [1] 1 2
grep('com|org', mystring)
# [1] 5 6
```

. является специальным символом в Regex. Это означает «соответствовать любому персонажу»,

```
grep('The number .', mystring)
# [1] 1 2
```

Будьте осторожны при попытке сопоставить точки!

```
tricky <- c('www.personal.org', 'My friend is a cyborg')
grep('.org', tricky)
# [1] 1 2
```

Чтобы соответствовать буквальному символу, вам нужно избежать строки с обратным слэшем (`\`). Тем не менее, R пытается найти escape-символы при создании строк, поэтому вам действительно нужно избежать самой обратной косой черты (т. Е. Вам нужно *удвоить* символы регулярного выражения *escape*- символов).

```
grep('\.org', tricky)
# Error: '\.' is an unrecognized escape in character string starting "'\."
grep('\\.org', tricky)
# [1] 1
```

Если вы хотите совместить один из нескольких символов, вы можете обернуть эти символы в скобки (`[]`)

```
grep('[13]', mystring)
# [1] 3 4
grep('[@/]', mystring)
# [1] 5 7
```

Может быть полезно указать последовательности символов. Например, `[0-4]` будет соответствовать 0, 1, 2, 3 или 4, `[AZ]` будет соответствовать любой заглавной букве, `[Az]` будет соответствовать любой прописной или строчной букве, а `[A-z0-9]` будет соответствовать любому буква или номер (т.е. все буквенно-цифровые символы)

```
grep('[0-4]', mystring)
# [1] 3 4
grep('[A-Z]', mystring)
# [1] 1 2 4 5 6
```

R также имеет несколько классов ярлыков, которые могут использоваться в скобках. Например, `[:lower:]` является коротким для `az` , `[:upper:]` является сокращением для `AZ` , `[:alpha:]` - `Az` , `[:digit:]` - `0-9` , а `[:alnum:]` - `A-z0-9` . Обратите внимание, что все эти *выражения* должны использоваться внутри скобок; например, чтобы соответствовать одной цифре, вы можете использовать `[[:digit:]]` (обратите внимание на двойные скобки). В качестве другого примера, `[@[:digit:]/]` будет соответствовать символам `@` , `/` или `0-9` .

```
grep('[[:digit:]]', mystring)
# [1] 1 2 3 4
grep('[@[:digit:]/]', mystring)
# [1] 1 2 3 4 5 7
```

Кронштейны также могут использоваться, чтобы отрицать совпадение с каратом (`^`). Например, `[^5]` будет соответствовать любому символу, отличному от «5».

```
grep('The number [^5]', mystring)
```

```
# [1] 2
```

Прочитайте Синтаксис регулярного выражения в R онлайн:
https://riptutorial.com/ru/r/topic/9743/синтаксис-регулярного-выражения-в-r

глава 106: Случайность

Вступление

Язык R обычно используется для статистического анализа. Таким образом, он содержит надежный набор вариантов рандомизации. Для получения конкретной информации о выборке из распределений вероятности см. Документацию для функций распределения .

замечания

Пользователи, которые приходят с других языков программирования, могут быть смущены отсутствием функции `rand` эквивалентной тому, что они могли испытывать раньше. Генерация основного случайного числа выполняется с использованием семейства функций `r*` для каждого распределения (см. Ссылку выше). Случайные числа, нарисованные равномерно из диапазона, могут быть сгенерированы с использованием `runif` для «случайной однородности». Поскольку это также выглядит подозрительно, как «запустить if», часто бывает трудно найти новых пользователей R.

Examples

Случайные ничьи и перестановки

Команда `sample` может использоваться для моделирования классических проблем с вероятностью, таких как рисование из урны с заменой или без нее, или создание случайных перестановок.

Обратите внимание, что в этом примере `set.seed` используется для обеспечения воспроизводимости кода примера. Однако `sample` будет работать без явного вызова `set.seed` .

Случайная перестановка

В простейшей форме `sample` создает случайную перестановку вектора целых чисел. Это может быть выполнено с помощью:

```
set.seed(1251)
sample(x = 10)

[1]  7  1  4  8  6  3 10  5  2  9
```

При отсутствии других аргументов `sample` возвращает случайную перестановку вектора от 1

до `x` . Это может быть полезно при попытке рандомизировать порядок строк в кадре данных. Это общая задача при создании таблиц рандомизации для испытаний или при выборе случайного подмножества строк для анализа.

```
library(datasets)
set.seed(1171)
iris_rand <- iris[sample(x = 1:nrow(iris)),]

> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa

> head(iris_rand)
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
145          6.7         3.3          5.7         2.5  virginica
5            5.0         3.6          1.4         0.2     setosa
85           5.4         3.0          4.5         1.5 versicolor
137          6.3         3.4          5.6         2.4  virginica
128          6.1         3.0          4.9         1.8  virginica
105          6.5         3.0          5.8         2.2  virginica
```

Ничья без замены

Используя `sample` , мы также можем имитировать чертеж из набора с заменой и без нее. Чтобы пробовать без замены (по умолчанию), вы должны предоставить образец с набором, который нужно нарисовать, и количеством ничьих. Набор, который нужно нарисовать, задается как вектор.

```
set.seed(7043)
sample(x = LETTERS,size = 7)

[1] "S" "P" "J" "F" "Z" "G" "R"
```

Обратите внимание: если аргумент `size` совпадает с длиной аргумента `x` , вы создаете случайную перестановку. Также обратите внимание, что вы не можете указать размер, превышающий длину `x` при выполнении выборки без замены.

```
set.seed(7305)
sample(x = letters,size = 26)

[1] "x" "z" "y" "i" "k" "f" "d" "s" "g" "v" "j" "o" "e" "c" "m" "n" "h" "u" "a" "b" "l" "r"
"w" "t" "q" "p"

sample(x = letters,size = 30)
Error in sample.int(length(x), size, replace, prob) :
  cannot take a sample larger than the population when 'replace = FALSE'
```

Это подводит нас к рисованию с заменой.

Рисование с заменой

Чтобы сделать случайные ничьи из набора с заменой, вы используете аргумент `replace` для `sample` . По умолчанию `replace` `FALSE` . Установка его в значение `TRUE` означает, что каждый элемент набора, который вытягивает, может появляться более одного раза в конечном результате.

```
set.seed(5062)
sample(x = c("A","B","C","D"),size = 8,replace = TRUE)

[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

Изменение вероятности рисования

По умолчанию, когда вы используете `sample` , он предполагает, что вероятность выбора каждого элемента одинакова. Рассмотрите это как основную проблему «урны». Приведенный ниже код эквивалентен вычерчиванию цветного мрамора из урны 20 раз, записывая цвет и затем помещая мрамор обратно в урну. Урна содержит один красный, один синий и один зеленый мрамор, что означает, что вероятность рисования каждого цвета составляет 1/3.

```
set.seed(6472)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE)
```

Предположим, что вместо этого мы хотели выполнить ту же задачу, но наша урна содержит 2 красных мрамора, 1 синий мрамор и 1 зеленый мрамор. Один из вариантов - изменить аргумент, который мы отправляем на `x` чтобы добавить дополнительный `Red` . Однако лучший выбор - использовать аргумент `prob` для `sample` .

Аргумент `prob` принимает вектор с вероятностью рисования каждого элемента. В нашем примере выше вероятность рисования красного мрамора составит 1/2, а вероятность рисования синего или зеленого мрамора составит 1/4.

```
set.seed(28432)
sample(x = c("Red","Blue","Green"),
       size = 20,
       replace = TRUE,
       prob = c(0.50,0.25,0.25))
```

Контр-интуитивно аргумент, заданный для `prob` , не должен суммироваться до 1. R всегда будет преобразовывать данные аргументы в вероятности, равные 1. Например, рассмотрим

наш приведенный выше пример из 2 красных, 1 синих и 1 зеленых. Вы можете добиться тех же результатов, что и наш предыдущий код, используя эти цифры:

```
set.seed(28432)
frac_prob_example <- sample(x = c("Red","Blue","Green"),
                            size = 200,
                            replace = TRUE,
                            prob = c(0.50,0.25,0.25))

set.seed(28432)
numeric_prob_example <- sample(x = c("Red","Blue","Green"),
                               size = 200,
                               replace = TRUE,
                               prob = c(2,1,1))

> identical(frac_prob_example,numeric_prob_example)
[1] TRUE
```

Основное ограничение заключается в том, что вы не можете установить все вероятности равными нулю, и ни одно из них не может быть меньше нуля.

Вы также можете использовать `prob` если для параметра `replace` установлено значение `FALSE` . В этой ситуации после того, как каждый элемент нарисован, пропорции `prob` значений для остальных элементов дают вероятность для следующего розыгрыша. В этой ситуации у вас должно быть достаточно ненулевых вероятностей для достижения `size` образца, который вы рисуете. Например:

```
set.seed(21741)
sample(x = c("Red","Blue","Green"),
       size = 2,
       replace = FALSE,
       prob = c(0.8,0.19,0.01))
```

В этом примере красный рисуется в первом розыгрыше (в качестве первого элемента). 80% шансов на то, что Red будет нарисован, 19% -ный шанс нанесения Синего и 1% шанс нарисовать Грина.

Для следующего розыгрыша, Красный больше не находится в урне. Общее количество вероятностей среди остальных предметов составляет 20% (19% для синего и 1% для зеленого). Для этой ничьей вероятность 95% будет синей (19/20) и 5% -ной вероятностью будет зеленый (1/20).

Установка семян

Функция `set.seed` используется для установки случайного семени для всех функций рандомизации. Если вы используете R для создания рандомизации, которую хотите воспроизвести, сначала следует использовать `set.seed` .

```
set.seed(1643)
samp1 <- sample(x = 1:5,size = 200,replace = TRUE)
```

```
set.seed(1643)
samp2 <- sample(x = 1:5,size = 200,replace = TRUE)

> identical(x = samp1,y = samp2)
[1] TRUE
```

Обратите внимание, что для параллельной обработки требуется специальная обработка случайного семени, описанная в другом месте.

Прочитайте Случайность онлайн: https://riptutorial.com/ru/r/topic/9574/случайность

глава 107: Согласование и замена шаблонов

Вступление

В этом разделе описываются соответствующие шаблоны строк, а также их извлечение или замена. Подробные сведения об определении сложных шаблонов см. В разделе Регулярные выражения .

Синтаксис

- grep ("query", "subject", optional_args)
- grepl ("query", "subject", optional_args)
- gsub ("(group1) (group2)", "\\ group #", "subject")

замечания

Различия с другими языками

Экранированные символы регулярных выражений (например, `\1`) должны быть экранированы во второй раз (например, `\\1`) не только в аргументе `pattern` , но и в `replacement` `sub` и `gsub` .

По умолчанию шаблон для всех команд (grep, sub, regexpr) не является Perl Compatible Regular Expression (PCRE), поэтому некоторые вещи, такие как lookarounds, не поддерживаются. Однако каждая функция принимает аргумент `perl=TRUE` для их включения. Дополнительную информацию см. В разделе « Регулярные выражения R» .

Специализированные пакеты

- стринги
- stringr

Examples

Выполнение замен

```
# example data
test_sentences <- c("The quick brown fox quickly", "jumps over the lazy dog")
```

Давайте сделаем коричневую лисицу красной:

```
sub("brown","red", test_sentences)
#[1] "The quick red fox quickly"       "jumps over the lazy dog"
```

Теперь давайте сделаем `"fast"` лисицу `"fastly"` . Это не будет сделано:

```
sub("quick", "fast", test_sentences)
#[1] "The fast red fox quickly"       "jumps over the lazy dog"
```

`sub` делает только первую доступную замену, нам нужен `gsub` для глобальной замены :

```
gsub("quick", "fast", test_sentences)
#[1] "The fast red fox fastly"       "jumps over the lazy dog"
```

См. « Изменение строк» путем замены для большего количества примеров.

Поиск матчей

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")
```

Есть ли матч?

`grepl()` используется для проверки наличия слова или регулярного выражения в строке или символьном векторе. Функция возвращает вектор TRUE / FALSE (или «Boolean»).

Обратите внимание, что мы можем проверить каждую строку для слова «лиса» и получить взамен булевой вектор.

```
grepl("fox", test_sentences)
#[1]  TRUE FALSE
```

Место встречи

`grep` принимает строку символов и регулярное выражение. Он возвращает числовой вектор индексов. Это вернет, какое предложение содержит в нем слово «лиса».

```
grep("fox", test_sentences)
#[1] 1
```

Соответствующие значения

Чтобы выбрать предложения, соответствующие шаблону:

```
# each of the following lines does the job:
test_sentences[grep("fox", test_sentences)]
test_sentences[grepl("fox", test_sentences)]
grep("fox", test_sentences, value = TRUE)
# [1] "The quick brown fox"
```

подробности

Поскольку шаблон `"fox"` - это просто слово, а не регулярное выражение, мы могли бы повысить производительность (с помощью `grep` или `grepl`), указав `fixed = TRUE` .

```
grep("fox", test_sentences, fixed = TRUE)
#[1] 1
```

Чтобы выбрать предложения, *не* соответствующие шаблону, можно использовать `grep` с `invert = TRUE` ; или следуйте правилам подмножества с помощью `-grep(...)` или `!grepl(...)` .

В обоих `grepl(pattern, x)` и `grep(pattern, x)` параметр `x` векторизован , параметр `pattern` - нет. В результате вы не можете использовать их напрямую, чтобы сопоставить `pattern[1]` с `x[1]` , `pattern[2]` против `x[2]` и т. Д.

Резюме матчей

После выполнения команды `grepl` , возможно, вы хотите получить обзор о том, сколько совпадений имеют `TRUE` или `FALSE` . Это полезно, например, в случае больших наборов данных. Для этого выполните команду `summary` :

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

# find matches
matches <- grepl("fox", test_sentences)

# overview
summary(matches)
```

Единый и глобальный матч.

При работе с регулярными выражениями один модификатор для PCRE равен `g` для глобального соответствия.

В функциях соответствия и замены R есть две версии: первое совпадение и глобальное соответствие:

- `sub(pattern,replacement,text)` заменит первое вхождение шаблона заменой в тексте
- `gsub(pattern,replacement,text)` будет делать то же самое, что и sub, но для каждого появления шаблона
- `regexpr(pattern,text)` вернет позицию соответствия для первого экземпляра шаблона
- `gregexpr(pattern,text)` вернет все совпадения.

Некоторые случайные данные:

```
set.seed(123)
teststring <- paste0(sample(letters,20),collapse="")

# teststring
#[1] "htjuwakqxzpgrsbncvyo"
```

Давайте посмотрим, как это работает, если мы хотим заменить гласные чем-то другим:

```
sub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** wakqxzpgrsbncvyo"

gsub("[aeiouy]"," ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** w ** HERE WAS A VOWEL** kqxzpgrsbncv ** HERE WAS A VOWEL**  **
HERE WAS A VOWEL** "
```

Теперь давайте посмотрим, как мы можем найти согласный, за которым следует один или несколько гласных:

```
regexpr("[^aeiou][aeiou]+",teststring)
#[1] 3
#attr(,"match.length")
#[1] 2
#attr(,"useBytes")
#[1] TRUE
```

Мы имеем совпадение на позиции 3 строки длины 2, т. `ju`

Теперь, если мы хотим получить все совпадения:

```
gregexpr("[^aeiou][aeiou]+",teststring)
#[[1]]
#[1]  3  5 19
#attr(,"match.length")
#[1] 2 2 2
#attr(,"useBytes")
#[1] TRUE
```

Все это действительно здорово, но это только дает использование позиции соответствия,

и это не так просто получить то, что соответствует, и здесь идут `regmatches` , единственная цель - извлечь строку, сопоставленную с regexpr, но она имеет другой синтаксис.

Давайте сохраним наши совпадения в переменной, а затем извлечем их из исходной строки:

```
matches <- gregexpr("[^aeiou][aeiou]+",teststring)
regmatches(teststring,matches)
#[[1]]
#[1] "ju" "wa" "yo"
```

Это может показаться странным, если у вас нет ярлыка, но это позволяет извлечь из другой строки совпадения нашей первой (подумайте о сравнении двух длинных векторов, где вы знаете, что существует общий шаблон для первого, но не для второго, это позволяет простое сравнение):

```
teststring2 <- "this is another string to match against"
regmatches(teststring2,matches)
#[[1]]
#[1] "is" " i" "ri"
```

Внимание: по умолчанию шаблон не является Perl Compatible Regular Expression, некоторые вещи, такие как lookarounds, не поддерживаются, но каждая представленная здесь функция допускает аргумент `perl=TRUE` для их включения.

Найти совпадения в больших наборах данных

В случае больших наборов данных вызов `grepl("fox", test_sentences)` работает не очень хорошо. Большие наборы данных, например, обходные сайты или миллионы твитов и т. Д.

Первым ускорением является использование параметра `perl = TRUE` . Еще быстрее опция `fixed = TRUE` . Полный пример:

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

grepl("fox", test_sentences, perl = TRUE)
#[1]  TRUE FALSE
```

В случае текстовой обработки часто используется корпус. Корпус нельзя использовать напрямую с `grepl` . Поэтому рассмотрим эту функцию:

```
searchCorpus <- function(corpus, pattern) {
  return(tm_index(corpus, FUN = function(x) {
    grepl(pattern, x, ignore.case = TRUE, perl = TRUE)
  }))
}
```

Прочитайте Согласование и замена шаблонов онлайн: https://riptutorial.com/ru/r/topic/1123/

согласование-и-замена-шаблонов

глава 108: Создание векторов

Examples

Последовательность чисел

Используйте оператор `:` для создания последовательностей чисел, например, для использования в векторизации больших фрагментов кода:

```
x <- 1:5
x
## [1] 1 2 3 4 5
```

Это работает в обоих направлениях

```
10:4
# [1] 10  9  8  7  6  5  4
```

и даже с числами с плавающей запятой

```
1.25:5
# [1] 1.25 2.25 3.25 4.25
```

или негативы

```
-4:4
#[1] -4 -3 -2 -1  0  1  2  3  4
```

сл ()

`seq` является более гибкой функцией, чем оператор `:` позволяющий указать шаги, отличные от 1.

Функция создает последовательность с `start` (по умолчанию 1) до конца, включая это число.

Вы можете указать только параметр end (`to`)

```
seq(5)
# [1] 1 2 3 4 5
```

Как и начало

```
seq(2, 5) # or seq(from=2, to=5)
# [1] 2 3 4 5
```

И, наконец, шаг (`by`)

```
seq(2, 5, 0.5) # or seq(from=2, to=5, by=0.5)
# [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

`seq` может дополнительно вывести (равномерно разнесенные) шаги, когда в качестве альтернативы поставляется желаемая длина выхода (`length.out`)

```
seq(2,5, length.out = 10)
# [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
```

Если последовательность должна иметь ту же длину, что и другой вектор, мы можем использовать `along.with` сокращением для `length.out = length(x)`

```
x = 1:8
seq(2,5,along.with = x)
# [1] 2.000000 2.428571 2.857143 3.285714 3.714286 4.142857 4.571429 5.000000
```

В семействе `seq` есть две полезные упрощенные функции: `seq_along` , `seq_len` и `seq.int` . `seq_along` и `seq_len` строят естественные (`seq_len`) числа от 1 до N, где N определяется аргументом функции, длиной вектора или списка с `seq_along` и целочисленным аргументом с `seq_len` .

```
seq_along(x)
# [1] 1 2 3 4 5 6 7 8
```

Обратите внимание: `seq_along` возвращает индексы существующего объекта.

```
# counting numbers 1 through 10
seq_len(10)
[1]  1  2  3  4  5  6  7  8  9 10
# indices of existing vector (or list) with seq_along
letters[1:10]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
seq_along(letters[1:10])
[1]  1  2  3  4  5  6  7  8  9 10
```

`seq.int` является тем же самым, что и `seq` поддерживаемым для древней совместимости.

Существует также старая функциональная `sequence` которая создает вектор последовательностей из не отрицательного аргумента.

```
sequence(4)
# [1] 1 2 3 4
sequence(c(3, 2))
# [1] 1 2 3 1 2
sequence(c(3, 2, 5))
# [1] 1 2 3 1 2 1 2 3 4 5
```

векторы

Векторы в R могут иметь разные типы (например, целочисленные, логические, символьные). Наиболее общий способ определения вектора - использовать функцию `vector()` .

```
vector('integer',2) # creates a vector of integers of size 2.
vector('character',2) # creates a vector of characters of size 2.
vector('logical',2) # creates a vector of logicals of size 2.
```

Однако в R, сокращенные функции, как правило, более популярны.

```
integer(2) # is the same as vector('integer',2) and creates an integer vector with two
elements
character(2) # is the same as vector('integer',2) and creates an character vector with two
elements
logical(2) # is the same as vector('logical',2) and creates an logical vector with two
elements
```

Также возможно создание векторов со значениями, отличными от значений по умолчанию. Часто для этого используется функция `c()` . C не подходит для объединения или конкатенации.

```
c(1, 2) # creates a integer vector of two elements: 1 and 2.
c('a', 'b') # creates a character vector of two elements: a and b.
c(T,F) # creates a logical vector of two elements: TRUE and FALSE.
```

Важно отметить, что R интерпретирует любое целое число (например, 1) как целочисленный вектор размера один. То же самое справедливо для чисел (например, 1.1), логических элементов (например, T или F) или символов (например, «a»). Таким образом, вы по существу объединяете векторы, которые, в свою очередь, являются векторами.

Обратите внимание, что вам всегда нужно комбинировать похожие векторы. В противном случае R попытается преобразовать векторы в векторы того же типа.

```
c(1,1.1,'a',T) # all types (integer, numeric, character and logical) are converted to the
'lowest' type which is character.
```

Поиск элементов в векторах может выполняться с помощью оператора `[` .

```
vec_int <- c(1,2,3)
vec_char <- c('a','b','c')
vec_int[2] # accessing the second element will return 2
vec_char[2] # accessing the second element will return 'b'
```

Это также можно использовать для изменения значений

```
vec_int[2] <- 5 # change the second value from 2 to 5
```

```
vec_int # returns [1] 1 5 3
```

Наконец, оператор `:` (short для функции `seq()`) можно использовать для быстрого создания вектора чисел.

```
vec_int <- 1:10
vec_int # returns [1] 1 2 3 4 5 6 7 8 9 10
```

Это также можно использовать для подмножества векторов (от простых до более сложных подмножеств)

```
vec_char <- c('a','b','c','d','e')
vec_char[2:4] # returns [1] "b" "c" "d"
vec_char[c(1,3,5)] # returns [1] "a" "c" "e"
```

Создание названных векторов

Именованный вектор может быть создан несколькими способами. С `c` :

```
xc <- c('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

что приводит к:

```
> xc
a b c d
5 6 7 8
```

со `list` :

```
xl <- list('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

что приводит к:

```
> xl
$a
[1] 5

$b
[1] 6

$c
[1] 7

$d
[1] 8
```

С `setNames` функции `setNames` два вектора одинаковой длины могут использоваться для создания именованного вектора:

```
x <- 5:8
y <- letters[1:4]

xy <- setNames(x, y)
```

который приводит к названному целочисленному вектору:

```
> xy
a b c d
5 6 7 8
```

Как видно, это дает тот же результат, что и метод `c` .

Вы также можете использовать функцию `names` чтобы получить тот же результат:

```
xy <- 5:8
names(xy) <- letters[1:4]
```

С таким вектором также возможно выбрать элементы по имени:

```
> xy["c"]
c
7
```

Эта функция позволяет использовать такой именованный вектор как вектор поиска / таблицу для соответствия значениям значениям другого вектора или столбца в фрейме данных. Учитывая следующий фрейм данных:

```
mydf <- data.frame(let = c('c','a','b','d'))

> mydf
  let
1   c
2   a
3   b
4   d
```

Предположим, вы хотите создать новую переменную в `mydf` называемом `num` с правильными значениями из `xy` в строках. Использование `match` функции соответствующие значения из `xy` могут быть выбраны:

```
mydf$num <- xy[match(mydf$let, names(xy))]
```

что приводит к:

```
> mydf
  let num
1   c   7
2   a   5
3   b   6
```

```
4   d   8
```

Развертывание вектора с функцией rep ()

Функция `rep` может использоваться для повторения вектора достаточно гибким образом.

```
# repeat counting numbers, 1 through 5 twice
rep(1:5, 2)
[1] 1 2 3 4 5 1 2 3 4 5

# repeat vector with incomplete recycling
rep(1:5, 2, length.out=7)
[1] 1 2 3 4 5 1 2
```

Каждый аргумент особенно полезен для расширения вектора статистики наблюдательных / экспериментальных единиц в вектор данных с повторными наблюдениями этих единиц.

```
# same except repeat each integer next to each other
rep(1:5, each=2)
[1] 1 1 2 2 3 3 4 4 5 5
```

Хорошей особенностью `rep` относительно расширения этой структуры данных является то, что расширение вектора на несбалансированную панель может быть выполнено путем замены аргумента length вектором, который определяет количество повторений каждого элемента в векторе:

```
# automated length repetition
rep(1:5, 1:5)
 [1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5
# hand-fed repetition length vector
rep(1:5, c(1,1,1,2,2))
[1] 1 2 3 4 4 5 5
```

Это должно обеспечить возможность предоставления внешней функции для подачи второго аргумента `rep` , чтобы динамически построить вектор, который расширяется в соответствии с данными.

Как и в `seq` , более быстрыми, упрощенными версиями `rep` являются `rep_len` и `rep.int` . Они отбрасывают некоторые атрибуты, которые поддерживает `rep` и поэтому могут быть наиболее полезными в ситуациях, когда скорость является проблемой, а дополнительные аспекты повторяющегося вектора не нужны.

```
# repeat counting numbers, 1 through 5 twice
rep.int(1:5, 2)
[1] 1 2 3 4 5 1 2 3 4 5

# repeat vector with incomplete recycling
rep_len(1:5, length.out=7)
```

```
[1] 1 2 3 4 5 1 2
```

Векторы от построения констант: Последовательности букв и названий месяцев

`R` имеет ряд построений в константах. Доступны следующие константы:

- `LETTERS` : 26 строчных букв латинского алфавита
- `letters` : 26 строчных букв латинского алфавита
- `month.abb` : трехбуквенные аббревиатуры для английских месяцев
- `month.name` : английские имена за месяцы года
- `pi` : отношение окружности круга к его диаметру

Из букв и констант месяца могут быть созданы векторы.

1) Последовательности букв:

```
> letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v"
"w" "x" "y" "z"

> LETTERS[7:9]
[1] "G" "H" "I"

> letters[c(1,5,3,2,4)]
[1] "a" "e" "c" "b" "d"
```

2) Последовательности аббревиатур месяца или месяцев:

```
> month.abb
 [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

> month.name[1:4]
[1] "January"  "February" "March"    "April"

> month.abb[c(3,6,9,12)]
[1] "Mar" "Jun" "Sep" "Dec"
```

Прочитайте Создание векторов онлайн: https://riptutorial.com/ru/r/topic/1088/создание-векторов

глава 109: Создание отчетов с помощью RMarkdown

Examples

Печать таблиц

Существует несколько пакетов, которые позволяют выводить данные в виде таблиц HTML или LaTeX. Они в основном отличаются гибкостью.

Здесь я использую пакеты:

- knitr
- xtable
- потворствовать

Для документов HTML

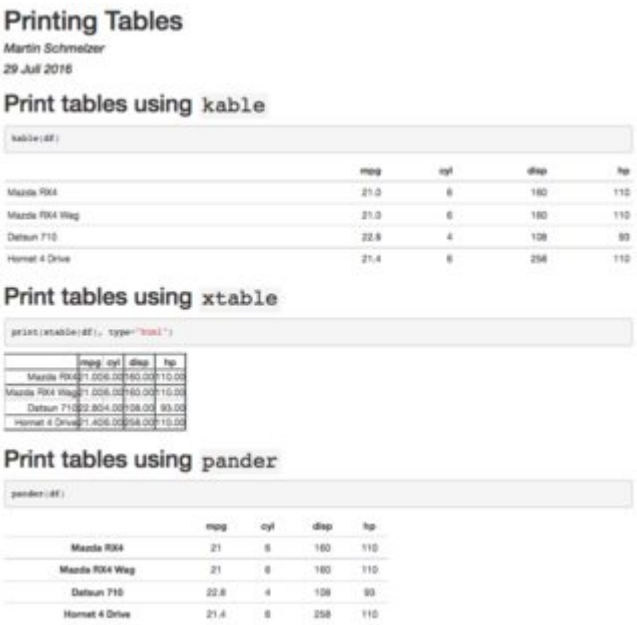
````
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```

# Print tables using `kable`
```{r, 'kable'}
kable(df)
```

# Print tables using `xtable`
```{r, 'xtable', results='asis'}
print(xtable(df), type="html")
```

# Print tables using `pander`
```{r, 'pander'}
pander(df)
```
````



Для документов в формате PDF

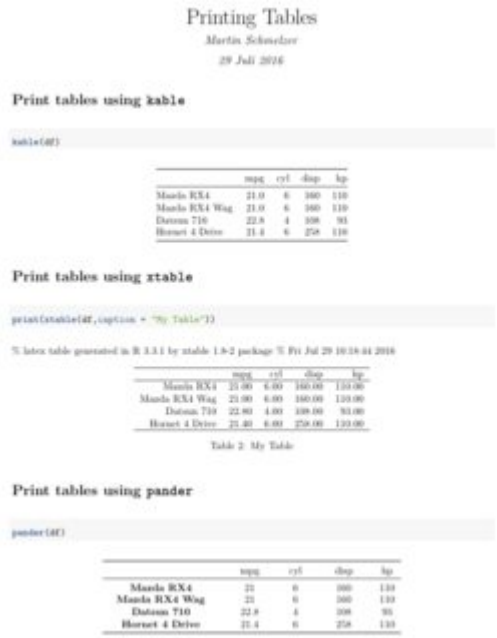
```
---
title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```

Print tables using `kable`
```{r, 'kable'}
kable(df)
```

Print tables using `xtable`
```{r, 'xtable', results='asis'}
print(xtable(df, caption="My Table"))
```

Print tables using `pander`
```{r, 'pander'}
pander(df)
```
```



Как я могу остановить xxtable печать комментария перед каждой таблицей?

```
options(xtable.comment = FALSE)
```

Включая команды Preample LaTeX

Существует два возможных способа включения команд преамбулы LaTeX (например, `\usepackage`) в документе RMarkdown.

1. Использование `header-includes` опции YAML `header-includes` :

```
---
title: "Including LaTeX Preample Commands in RMarkdown"
header-includes:
   - \renewcommand{\familydefault}{cmss}
   - \usepackage[cm, slantedGreek]{sfmath}
   - \usepackage[T1]{fontenc}
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

Section 1

As you can see, this text uses the Computer Moden Font!
```

2. Включение внешних команд с `includes` B `in_header` `includes` , `in_header`

```
---
title: "Including LaTeX Preample Commands in RMarkdown"
output:
  pdf_document:
    includes:
      in_header: includes.tex
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

Section 1

As you can see, this text uses the Computer Modern Font!
```

Здесь содержимое include.tex - это те же три команды, которые мы включили в `header-includes` .

Создание нового шаблона

Возможный третий вариант - написать собственный шаблон LaTex и включить его в `template` . Но это охватывает гораздо больше структуры, чем только преамбулу.

```
---
title: "My Template"
author: "Martin Schmelzer"
output:
  pdf_document:
    template: myTemplate.tex
---
```

В том числе библиографии

Каталог bibtex cna легко включается в `bibliography:` опций YAML `bibliography:` Определенный стиль для библиографии можно добавить с `biblio-style:` Ссылки добавляются в конце документа.

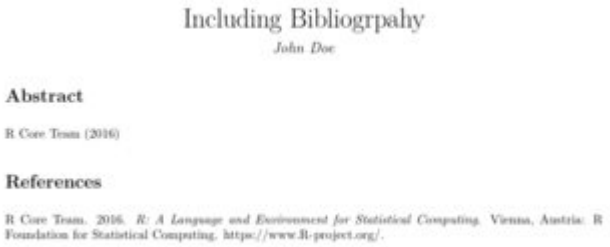
```
---
title: "Including Bibliography"
author: "John Doe"
output: pdf_document
```

```
bibliography: references.bib
---

# Abstract

@R_Core_Team_2016

# References
```



Including Bibliogrpahy

John Doe

Abstract

R Core Team (2016)

References

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

Основная структура документа R-markdown

Кодовые фрагменты кода R-markdown

R-markdown - это файл с разметкой со встроенными блоками R-кода, называемыми *фрагментами* . Существует два типа фрагментов кода R: **встроенный** и **блок** .

Встроенные фрагменты добавляются с использованием следующего синтаксиса:

```
`r 2*2`
```

Они оцениваются и вставляют свой ответ на результат.

Блок- фрагменты имеют другой синтаксис:

````
```{r name, echo=TRUE, include=TRUE, ...}

2*2

````
````

И у них есть несколько возможных вариантов. Вот основные (но есть и многие другие):

- **echo** (boolean) управляет тем, что код внутри блока будет включен в документ
- **включают** (логические) элементы управления, которые должны быть включены в документ
- **fig.width** (numeric) устанавливает ширину выходных данных
- **fig.height** (числовое значение) задает высоту выходных данных
- **fig.cap** (символ) устанавливает подписи к **рисункам**

Они написаны в простом формате `tag=value` , как в приведенном выше примере.

Пример документа R-markdown

Ниже приведен базовый пример файла R-markdown, иллюстрирующий, как фрагменты кода R встроены в r-уценку.

````
# Title #

This is **plain markdown** text.

```{r code, include=FALSE, echo=FALSE}

# Just declare variables

income <- 1000
taxes  <- 125

```

My income is: `r income ` dollars and I payed `r taxes ` dollars in taxes.

Below is the sum of money I will have left:

```{r gain, include=TRUE, echo=FALSE}

gain <- income-taxes

gain

```

```{r plotOutput, include=TRUE, echo=FALSE, fig.width=6, fig.height=6}

pie(c(income,taxes), label=c("income", "taxes"))

```
````

Преобразование R-уценки в другие форматы

Пакет R `knitr` можно использовать для оценки фрагментов R внутри файла R-markdown и превращения его в обычный файл разметки.

Для превращения файла R-markdown в файл pdf / html необходимы следующие шаги:

1. Преобразуйте файл R-markdown в файл уценки с помощью `knitr` .
2. Преобразуйте полученный файл разметки в pdf / html с помощью специализированных инструментов, таких как *pandoc* .

В дополнение к вышеуказанному пакету `knitr` есть функции-обертки `knit2html()` и `knit2pdf()` которые могут использоваться для создания окончательного документа без промежуточного шага вручную, преобразующего его в формат уценки:

Если приведенный выше примерный файл был сохранен как `income.Rmd` его можно

преобразовать в `pdf` файл, используя следующие команды R:

```
library(knitr)
knit2pdf("income.Rmd", "income.pdf")
```

Окончательный документ будет похож на тот, который приведен ниже.

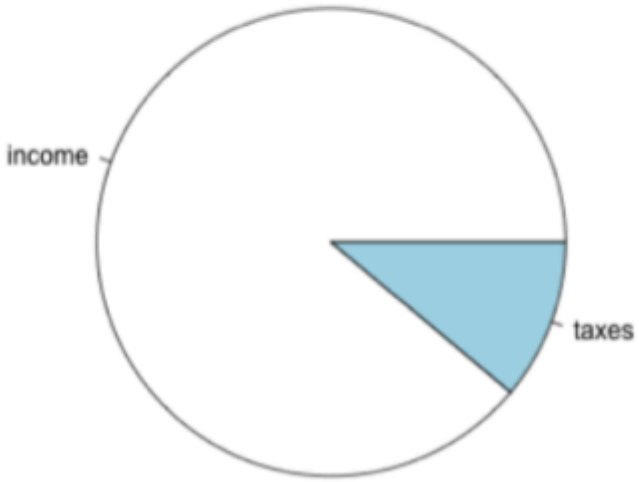
Title

This is **plain markdown** text.

My income is: 1000 dollars and I payed 125 dollars in taxes.

Below is the sum of money I will have left:

```
## [1] 875
```



Прочитайте Создание отчетов с помощью RMarkdown онлайн:
https://riptutorial.com/ru/r/topic/4572/создание-отчетов-с-помощью-rmarkdown

глава 110: Создание пакетов с помощью devtools

Вступление

Эта тема будет посвящена созданию пакетов R с нуля с пакетом devtools.

замечания

1. Официальное руководство R для создания пакетов
2. справочное руководство `roxygen2`
3. руководство пользователя `devtools`

Examples

Создание и распространение пакетов

Это *компактное руководство* о том, как быстро создать пакет R из вашего кода. Исчерпывающие документы будут связаны, когда они доступны, и их следует читать, если вы хотите более глубокое знание ситуации. См. *Примечания* для получения дополнительных ресурсов.

Каталог, в котором находится ваш код, будет обозначаться как `./` , и все команды предназначены для выполнения из приглашения R в этой папке.

Создание документации

Документация для вашего кода должна быть в формате, который очень похож на LaTeX.

Однако для упрощения процесса мы будем использовать инструмент с именем `roxygen` :

```
install.packages("devtools")
library("devtools")
install.packages("roxygen2")
library("roxygen2")
```

Полную страницу man для roxygen можно найти здесь . Он очень похож на *doxygen* .

Ниже приведен практический пример о том, как документировать функцию с помощью *roxygen* :

```
#' Increment a variable.
#'
#' Note that the behavior of this function
#' is undefined if `x` is not of class `numeric`.
#'
#' @export
#' @author  another guy
#' @name    Increment Function
#' @title   increment
#'
#' @param x   Variable to increment
#' @return    `x` incremented of 1
#'
#' @seealso   `other_function`
#'
#' @examples
#' increment(3)
#' > 4
increment <- function(x) {
  return (x+1)
}
```

И вот будет результат .

Также рекомендуется создать виньетку (см. Раздел « *Создание виньет*»), который является полным руководством по вашему пакету.

Построение скелета упаковки

Предполагая, что ваш код написан, например, в файлах `./script1.R` и `./script2.R` , запустите следующую команду, чтобы создать дерево файлов вашего пакета:

```
package.skeleton(name="MyPackage", code_files=c("script1.R","script2.R"))
```

Затем удалите все файлы в `./MyPackage/man/` . Теперь вы должны составить документацию:

```
roxygenize("MyPackage")
```

Вы также должны создать справочное руководство из своей документации, используя `R CMD Rd2pdf MyPackage` из *командной строки,* запущенной в `./` .

Выдача свойств пакета

1. Описание пакета

Измените `./MyPackage/DESCRIPTION` соответствии с вашими потребностями. Поля `Package` ,

`Version` , `License` , `Description` , `Title` , `Author` и `Maintainer` являются обязательными, другие необязательны.

Если ваш пакет зависит от других пакетов, укажите их в поле с именем `Depends` (версия R <3.2.0) или `Imports` (версия R> 3.2.0).

2. Дополнительные папки

Когда вы запустили сборку скелета, `./MyPackage/` только `R/` и `man/` subfolders. Однако у него могут быть и другие:

- `data/` : здесь вы можете поместить данные, которые нужны вашей библиотеке, и которые не являются кодом. Он должен быть сохранен как набор данных с расширением `.RData` , и вы можете загрузить его во время выполнения с `data()` и `load()`
- `tests/` : все файлы кода в этой папке будут запускаться во время установки. Если произошла ошибка, установка завершится с ошибкой.
- `src/` : для исходных файлов C / C ++ / Fortran (с помощью `Rcpp` ...).
- `exec/` : для других исполняемых файлов.
- `misc/` : для всего остального.

Завершение и сборка

Вы можете удалить `./MyPackage/Read-and-delete-me` .

Как и сейчас, ваш пакет готов к установке.

Вы можете установить его с помощью `devtools::install("MyPackage")` .

Чтобы создать свой пакет в качестве исходного tarball, вам нужно выполнить следующую команду из *командной строки* в `./` : `R CMD build MyPackage`

Распространение вашего пакета

Через Гитуб

Просто создайте новый репозиторий *MyPackage* и загрузите все в `MyPackage/` в главную ветку. Вот пример .

Тогда любой может установить ваш пакет из github с помощью devtools:

```
install_package("MyPackage", "your_github_usename")
```

Через CRAN

Ваш пакет должен соответствовать политике репозитория CRAN . Включая, но не ограничиваясь: ваш пакет должен быть межплатформенным (за исключением некоторых особых случаев), он должен пройти `R CMD check` проверки `R CMD check` .

Вот форма подачи . Вы должны загрузить исходный архив.

Создание виньет

> Виньетка - это длинный путеводитель по вашему пакету. Документация по функциям замечательна, если вы знаете имя нужной вам функции, но это бесполезно иначе. Виньетка похожа на книгу или научную статью: она может описать проблему, которую ваш пакет предназначен для решения, а затем показать читателю, как его решить.

Виньетки будут созданы полностью в уценке.

Требования

- Rmarkdown: `install.packages("rmarkdown")`
- Pandoc

Создание виньетки

```
devtools::use_vignette("MyVignette", "MyPackage")
```

Теперь вы можете редактировать свою виньетку `./vignettes/MyVignette.Rmd` .

Текст в вашей виньетке отформатирован как Markdown .

Единственное дополнение к оригинальному Markdown - это тег, который использует R-код, запускает его, захватывает вывод и переводит его в форматированный Markdown:

````
```{r}
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
```
````

Отобразится как:

```
# Add two numbers together
add <- function(a, b) a + b
```

```
add(10, 20)
## [1] 30
```

Таким образом, все пакеты, которые вы будете использовать в ваших виньетках, должны быть указаны как зависимости в `./DESCRIPTION`.

Прочитайте Создание пакетов с помощью devtools онлайн:
https://riptutorial.com/ru/r/topic/10884/создание-пакетов-с-помощью-devtools

глава 111: Списки

Examples

Краткое введение в списки

В общем, большинство объектов, с которыми вы могли бы взаимодействовать как пользователь, были бы вектором; например, числовой вектор, логический вектор. Эти объекты могут принимать только один тип переменной (числовой вектор может содержать только цифры внутри него).

Список мог бы хранить в нем любую переменную типа, превращая ее в общий объект, который может хранить любые типы переменных, которые нам понадобятся.

Пример инициализации списка

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

Чтобы понять данные, которые были определены в списке, мы можем использовать функцию str.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

Подмножество списков различает извлечение фрагмента списка, т. Е. Получение списка, содержащего подмножество элементов в исходном списке, и извлечение одного элемента. Использование `[` оператора, обычно используемого для векторов, создает новый список.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

Для получения одного элемента используйте `[[` вместо этого.

```
# Returns Character
exampleList3[[1]]
```

Список записей может быть назван:

```
exampleList4 <- list(
    num = 1:3,
    numeric = 0.5,
    char = c('a', 'b')
```

```
)
```

Записи в именованных списках могут быть доступны по имени, а не по индексу.

```
exampleList4[['char']]
```

Альтернативно, оператор `$` можно использовать для доступа к именованным элементам.

```
exampleList4$num
```

Это имеет то преимущество, что оно быстрее печатается и может быть легче читать, но важно знать о потенциальной ловушке. Оператор `$` использует частичное сопоставление для идентификации элементов списка соответствия и может привести к неожиданным результатам.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Списки могут быть особенно полезными, поскольку они могут хранить объекты различной длины и разных классов.

```
## Numeric vector
exampleVector1 <- c(12, 13, 14)
## Character vector
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## Matrix
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## List
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
    num = exampleVector1,
    char = exampleVector2,
    mat = exampleMatrix1,
    list = exampleList3
)
exampleList6
#$num
#[1] 12 13 14
#
#$char
#[1] "a" "b" "c" "d" "e" "f"
#
#$mat
#           [,1]        [,2]
#[1,] 0.5013050 -1.88801542
```

```
#[2,] 0.4295266  0.09751379
#
#$list
#$list[[1]]
#[1] "a"
#
#$list[[2]]
#[1] 1
#
#$list[[3]]
#[1] 2
```

Введение в списки

Списки позволяют пользователям хранить несколько элементов (например, векторов и матриц) под одним объектом. Вы можете использовать функцию `list` для создания списка:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Обратите внимание, что векторы, составляющие вышеуказанный список, представляют собой разные классы. Списки позволяют пользователям группировать элементы разных классов. Каждый элемент в списке также может иметь имя. Имена списков доступны с помощью функции `names` и назначаются тем же образом, имена строк и столбцов назначаются в матрице.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

При создании объекта списка часто проще и безопаснее объявлять имена списков.

```
l2 <- list(vec = c(1, 3, 5, 7, 9),
       mat = matrix(data = c(1, 2, 3), nrow = 3))
l2
## $vec
## [1] 1 3 5 7 9
##
## $mat
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

```
names(l2)
## [1] "vec" "mat"
```

Выше списка есть два элемента, называемые «vec» и «mat», вектор и матрица, нереально.

Причины использования списков

Для среднего пользователя R структура списка может оказаться одной из более сложных структур данных для управления. Нет никаких гарантий того, что все элементы внутри него одного типа; Не существует гарантированной структуры того, насколько сложным / не сложным будет список (элементом в списке может быть список)

Однако, одна из основных причин, когда использовать списки, чтобы использовать его для передачи параметров между функциями.

```
# Function example which returns a single element numeric vector
exampleFunction1 <- function(num1, num2){
    result <- num1 + num2
    return(result)
}

# Using example function 1
exampleFunction1(1, 2)

# Function example which returns a simple numeric vector
exampleFunction2 <- function(num1, num2, multiplier){
    tempResult1 <- num1 + num2
    tempResult2 <- tempResult1 * multiplier
    result <- c(tempResult1, tempResult2)
    return(result)
}

# Using example function 2
exampleFunction2(1, 2, 4)
```

В приведенном выше примере возвращаемые результаты - это просто простые числовые векторы. Нет никаких проблем для прохождения таких простых векторов.

На этом этапе важно отметить, что в общем случае функции R возвращают только один результат за один раз (вы можете использовать, если условия возвращают разные результаты). Однако, если вы намереваетесь создать функцию, которая принимает набор параметров и возвращает несколько типов результатов, таких как числовой вектор (значение настроек) и кадр данных (из расчета), вам нужно будет сбросить все эти результаты в список прежде чем возвращать его.

```
# We will be using mtcars dataset here
# Function which returns a result that is supposed to contain multiple type of results
# This can be solved by putting the results into a list
exampleFunction3 <- function(dataframe, removeColumn, sumColumn){
    resultDataFrame <- dataframe[, -removeColumn]
    resultSum <- sum(dataframe[, sumColumn])
    resultList <- list(resultDataFrame, resultSum)
```

```
    return(resultList)
}

# Using example function 3
exampleResult <- exampleFunction3(mtcars, 2, 4)
exampleResult[[1]]
exampleResult[[2]]
```

Преобразование списка в вектор, сохраняя пустые элементы списка

Когда вы хотите преобразовать список в вектор или объект data.frame, пустые элементы обычно удаляются.

Это может быть проблематично, что список создается из требуемой длины, создаются с пустым значением (например, создается список с n элементами, которые должны быть добавлены в матрицу mxn, data.frame или data.table). Однако можно без потерь преобразовать список в вектор, сохраняя пустые элементы:

```
res <- list(character(0), c("Luzhuang", "Laisu", "Peihui"), character(0),
    c("Anjiangping", "Xinzhai", "Yongfeng"), character(0), character(0),
    c("Puji", "Gaotun", "Banjingcun"), character(0), character(0),
    character(0))
res
```

```
[[1]]
character(0)

[[2]]
[1] "Luzhuang" "Laisu"    "Peihui"

[[3]]
character(0)

[[4]]
[1] "Anjiangping" "Xinzhai"     "Yongfeng"

[[5]]
character(0)

[[6]]
character(0)

[[7]]
[1] "Puji"       "Gaotun"     "Banjingcun"

[[8]]
character(0)

[[9]]
character(0)

[[10]]
character(0)
```

```
res <- sapply(res, function(s) if (length(s) == 0) NA_character_ else paste(s, collapse = "
```

```
"))
res
```

```
 [1] NA                              "Luzhuang Laisu Peihui"        NA
"Anjiangping Xinzhai Yongfeng" NA

 [6] NA                              "Puji Gaotun Banjingcun"       NA
NA                             NA
```

Сериализация: использование списков для передачи информации

Существуют случаи, когда необходимо объединить данные разных типов. Например, в Azure ML необходимо передавать информацию из модуля сценариев R в другой исключительно через данные. Предположим, что у нас есть dataframe и число:

```
> df
       name height        team fun_index title age         desc Y
1    Andrea    195       Lazio        97     6  33   eccellente 1
2      Paja    165  Fiorentina        87     6  31       deciso 1
3      Roro    190       Lazio        65     6  28       strano 0
4    Gioele     70       Lazio       100     0   2    simpatico 1
5     Cacio    170    Juventus        81     3  33         duro 0
6     Edola    171       Lazio        72     5  32     svampito 1
7    Salami    175       Inter        75     3  30  doppiopasso 1
8    Braugo    180       Inter        79     5  32          gjn 0
9     Benna    158    Juventus        80     6  28     esaurito 0
10   Riggio    182       Lazio        92     5  31     certezza 1
11 Giordano    185        Roma        79     5  29        buono 1

> number <- "42"
```

Мы можем получить доступ к этой информации:

```
> paste(df$name[4],"is a",df3$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number )
[1] "The answer to THE question is 42"
```

Чтобы поместить различные типы данных в фрейм данных, мы должны использовать объект списка и сериализацию. В частности, мы должны помещать данные в общий список, а затем помещать список в определенный фреймворк данных:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialize(l, connection=NULL)))
```

После того, как мы сохранили информацию в фрейме данных, нам нужно десериализовать его, чтобы использовать его:

```
#----- unserialize ----------------------------------------+
unser_obj <- unserialize(as.raw(dataframe_container$out2))
#----- taking back the elements---------------------------+
```

```
df_mod        <- unser_obj[1][[1]]
number_mod    <- unser_obj[2][[1]]
```

Затем мы можем проверить правильность передачи данных:

```
> paste(df_mod$name[4],"is a",df_mod$team[4], "supporter." )
[1] "Gioele is a  Lazio supporter."
> paste("The answer to THE question is", number_mod )
[1] "The answer to THE question is 42"
```

Прочитайте Списки онлайн: https://riptutorial.com/ru/r/topic/1365/списки

глава 112: Стандартизировать анализ путем написания автономных R-скриптов

Вступление

Если вы хотите регулярно применять анализ R к большому количеству отдельных файлов данных или предоставлять повторяемый метод анализа другим людям, исполняемый R-скрипт является удобным для пользователя способом. Вместо того, чтобы вы или ваш пользователь должны были вызвать `R` и выполнить ваш скрипт внутри R через `source(.)` Или вызов функции, ваш пользователь может просто вызвать сам сценарий, как если бы это была программа.

замечания

Для представления стандартных каналов ввода / вывода используйте `file("stdin")` функций `file("stdin")` (ввод с терминала или другой программы через канал), `stdout()` (стандартный вывод) и `stderr()` (стандартная ошибка). Обратите внимание, что, хотя есть функция `stdin()` , ее нельзя использовать при поставке готового скрипта в R, потому что он будет читать следующие строки этого скрипта вместо ввода пользователем.

Examples

Базовая структура автономной программы R и как ее называть

Первый автономный сценарий R

Отдельные сценарии R не выполняются программой `R` (`R.exe` под Windows), а программой `Rscript` (`Rscript.exe`), которая по умолчанию включена в вашу установку `R`

Чтобы намекнуть на этот факт, автономные R-скрипты начинаются со специальной строки под названием **Shebang line** , которая содержит следующий контент: `#!/usr/bin/env Rscript` . В Windows требуется дополнительная мера, которая позже будет описана.

Следующий простой автономный скрипт R сохраняет гистограмму под именем файла « hist.png» из чисел, которые он получает в качестве входных данных:

```
#!/usr/bin/env Rscript

# User message (\n = end the line)
cat("Input numbers, separated by space:\n")
# Read user input as one string (n=1 -> Read only one line)
```

```
input <- readLines(file('stdin'), n=1)
# Split the string at each space (\\s == any space)
input <- strsplit(input, "\\s")[[1]]
# convert the obtained vector of strings to numbers
input <- as.numeric(input)

# Open the output picture file
png("hist.png",width=400, height=300)
# Draw the histogram
hist(input)
# Close the output file
dev.off()
```

Вы можете увидеть несколько ключевых элементов автономного сценария R. В первой строке вы видите линию Шебанга. Вслед за этим `cat("....\n")` используется для печати сообщения пользователю. Используйте `file("stdin")` когда вы хотите указать «Пользовательский ввод на консоли» в качестве источника данных. Это можно использовать вместо имени файла в нескольких функциях чтения данных (`scan` , `read.table` , `read.csv` , ...). После того, как пользовательский ввод преобразуется из строк в числа, начинается печать. Там видно, что команды построения, которые должны быть записаны в файл, должны быть заключены в две команды. В этом случае `png(.)` И `dev.off()` . Первая функция зависит от желаемого формата выходного файла (другие общие варианты: `jpeg(.)` И `pdf(.)`). Вторая функция, `dev.off()` всегда требуется. Он записывает график в файл и завершает процесс построения графика.

Подготовка автономного сценария R

Linux / Mac

Файл автономного скрипта должен быть сначала выполнен исполняемым. Это можно сделать, щелкнув правой кнопкой мыши файл, открыв «Свойства» в открывшемся меню и установив флажок «Исполняемый файл» на вкладке «Разрешения». Кроме того, команда

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

можно вызвать в терминале.

Windows

Для каждого отдельного сценария командный файл должен быть написан со следующим содержимым:

```
"C:\Program Files\R-XXXXXXX\bin\Rscript.exe" "%~dp0\XXXXXXX.R" %*
```

Пакетный файл является обычным текстовым файлом, но с расширением `*.bat` кроме

расширения `*.txt` . Создайте его, используя текстовый редактор, например `notepad` (не `Word`) или аналогичный, и поместите имя файла в кавычки `"FILENAME.bat"`) в диалоговом окне сохранения. Чтобы отредактировать существующий пакетный файл, щелкните его правой кнопкой мыши и выберите «Изменить».

Вы должны адаптировать код, показанный выше, везде `XXX...` написано:

- Вставьте правильную папку, в которой находится ваша установка R
- Вставьте правильное имя своего сценария и поместите его в тот же каталог, что и этот командный файл.

Объяснение элементов в коде: первая часть `"C:\...\Rscript.exe"` сообщает Windows, где найти программу `Rscript.exe` . Вторая часть `"%~dp0\XXX.R"` сообщает `Rscript` выполнить написанный вами R-скрипт, который находится в той же папке, что и командный файл (`%~dp0` обозначает папку пакетного файла). Наконец, `%*` пересылает любые аргументы командной строки, которые вы передаете пакетному файлу в R-скрипт.

Если вы дважды щелкните командный файл, будет выполнен сценарий R. Если вы перетаскиваете файлы в пакетном файле, соответствующие имена файлов присваиваются сценарию R в качестве аргументов командной строки.

Использование littler для выполнения R-скриптов

littler (произносится *little r*) (cran) предоставляет помимо других функций две возможности запуска R-скриптов из командной строки с командой littler's `r` (когда вы работаете с Linux или MacOS).

Установка Littler

От R:

```
install.packages("littler")
```

Путь `r` печатается в терминале, например

```
You could link to the 'r' binary installed in
'/home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r'
from '/usr/local/bin' in order to use 'r' for scripting.
```

Чтобы иметь возможность вызывать `r` из командной строки системы, необходима символическая ссылка:

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

Использование apt-get (Debian, Ubuntu):

```
sudo apt-get install littler
```

Использование littler со стандартными скриптами .r

С помощью `r` from `littler` можно выполнять автономные R-скрипты без каких-либо изменений в скрипте. Пример скрипта:

```
# User message (\n = end the line)
cat("Input numbers, separated by space:\n")
# Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
# Split the string at each space (\\s == any space)
input <- strsplit(input, "\\s")[[1]]
# convert the obtained vector of strings to numbers
input <- as.numeric(input)

# Open the output picture file
png("hist.png",width=400, height=300)
# Draw the histogram
hist(input)
# Close the output file
dev.off()
```

Обратите внимание, что ни одна shebang не находится в верхней части скриптов. При сохранении, например, `hist.r` , он может быть `hist.r` непосредственно из системной команды:

```
r hist.r
```

Использование littler в сценариях *shebanged*

Также возможно создать исполняемые R-скрипты с littler, с использованием shebang

```
#!/usr/bin/env r
```

в верхней части скрипта. Соответствующий R-скрипт должен быть выполнен исполняемым с помощью `chmod +X /path/to/script.r` и может быть непосредственно вызван из системного терминала.

Прочитайте Стандартизировать анализ путем написания автономных R-скриптов онлайн: https://riptutorial.com/ru/r/topic/9937/стандартизировать-анализ-путем-написания-автономных-r-скриптов

глава 113: Структуры управляющих потоков

замечания

Для циклов - метод управления потоком для повторения задачи или набора задач над доменом. Основная структура цикла for

```
for ( [index] in [domain]){
  [body]
}
```

куда

1. `[index]` - это имя принимает ровно одно значение `[domain]` за каждую итерацию цикла.
2. `[domain]` - это вектор значений, для которого выполняется итерация.
3. `[body]` - это набор инструкций для применения на каждой итерации.

В качестве тривиального примера рассмотрим использование цикла for для получения суммарной суммы вектора значений.

```
x <- 1:4
cumulative_sum <- 0
for (i in x){
  cumulative_sum <- cumulative_sum + x[i]
}
cumulative_sum
```

Оптимизация структуры для циклов

Для циклов может быть полезно для концептуализации и выполнения задач для повторения. Однако, если они не сконструированы тщательно, они могут быть очень медленными для выполнения по сравнению с предпочтительным `apply` семейства функций. Тем не менее, есть несколько элементов, которые вы можете включить в свою конструкцию цикла для оптимизации цикла. Во многих случаях хорошая конструкция цикла for обеспечит вычислительную эффективность, очень близкую к эффективности функции приложения.

Строка «правильно построенная» для цикла строится на основной структуре и включает оператор, объявляющий объект, который будет захватывать каждую итерацию цикла. Этот объект должен иметь как объявленный класс, так и длину.

```
[output] <- [vector_of_length]
for ([index] in [length_safe_domain]){
  [output][index] <- [body]
}
```

Чтобы проиллюстрировать это, напишем цикл для квадрата каждого значения в числовом векторе (это только тривиальный пример для иллюстрации. «Правильный» способ выполнения этой задачи - `x_squared <- x^2`).

```
x <- 1:100
x_squared <- vector("numeric", length = length(x))
for (i in seq_along(x)){
  x_squared[i] <- x[i]^2
}
```

Опять же, обратите внимание, что мы сначала объявили приемник для вывода `x_squared` и дали ему класс «числовой» с той же длиной, что и `x` . Кроме того, мы объявили «безопасный домен», используя функцию `seq_along` . `seq_along` генерирует вектор индексов для объекта, который подходит для использования в циклах for. Хотя кажется интуитивным для использования `for (i in 1:length(x))` , если `x` имеет длину 0, цикл попытается выполнить итерацию по домену `1:0` , что приведет к ошибке (0-й индекс не определен в R).

Объекты сосуда и длина безопасные домены обрабатываются внутри `apply` семейства функций и пользователи рекомендуются принять `apply` подход вместо петель для как можно больше. Однако, если правильно сконструировано, цикл for может иногда обеспечивать большую четкость кода с минимальной потерей эффективности.

Векторизация для циклов

Для циклов часто может быть полезным инструментом в концептуализации задач, которые необходимо выполнить на каждой итерации. Когда цикл полностью разработан и концептуализирован, могут быть преимущества превращения петли в функцию.

В этом примере мы разработаем цикл for для вычисления среднего значения каждого столбца в `mtcars` данных `mtcars` (опять же, тривиальный пример, который может быть выполнен с помощью функции `colMeans`).

```
column_mean_loop <- vector("numeric", length(mtcars))
for (k in seq_along(mtcars)){
  column_mean_loop[k] <- mean(mtcars[[k]])
}
```

Цикл for может быть преобразован в функцию приложения, переписав тело цикла как функцию.

```
col_mean_fn <- function(x) mean(x)
column_mean_apply <- vapply(mtcars, col_mean_fn, numeric(1))
```

И для сравнения результатов:

```
identical(column_mean_loop,
          unname(column_mean_apply)) #* vapply added names to the elements
                                     #* remove them for comparison
```

Преимущества векторизованной формы заключаются в том, что мы смогли устранить несколько строк кода. Механика определения длины и типа выходного объекта и итерации по длине безопасного домена обрабатывается нами с помощью функции apply. Кроме того, функция apply немного быстрее, чем цикл. Разница скорости часто незначительна в человеческих терминах в зависимости от количества итераций и сложности тела.

Examples

Основные для строительства петли

В этом примере мы вычислим квадрат отклонения для каждого столбца в кадре данных, в этом случае `mtcars` .

Вариант A: целочисленный индекс

```
squared_deviance <- vector("list", length(mtcars))
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
```

`squared_deviance` - это список из 11 элементов, как и ожидалось.

```
class(squared_deviance)
length(squared_deviance)
```

Вариант B: индекс символа

```
squared_deviance <- vector("list", length(mtcars))
Squared_deviance <- setNames(squared_deviance, names(mtcars))
for (k in names(mtcars)){
  squared_deviance[[k]] <- (mtcars[[k]] - mean(mtcars[[k]]))^2
}
```

Что делать, если мы хотим получить `data.frame` ? Ну, есть много вариантов преобразования списка в другие объекты. Тем не менее, и , возможно , самый простой в данном случае, будет хранить `for` результатов в `data.frame` .

```
squared_deviance <- mtcars #copy the original
squared_deviance[TRUE]<-NA  #replace with NA or do squared_deviance[,]<-NA
```

```
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
dim(squared_deviance)
[1] 32 11
```

Результат будет тем же самым событием, хотя мы используем опцию character (B).

Оптимальное построение цикла

Чтобы проиллюстрировать эффект хорошей конструкции цикла, мы вычислим среднее значение для каждого столбца четырьмя различными способами:

1. Использование плохо оптимизированного для цикла
2. Использование хорошо оптимизированного для цикла
3. Использование `*apply` семейство функций
4. Использование функции `colMeans`

Каждый из этих параметров будет отображаться в коде; будет показано сравнение времени вычисления для выполнения каждой опции; и, наконец, будет дано обсуждение различий.

Плохо оптимизирован для цикла

```
column_mean_poor <- NULL
for (i in 1:length(mtcars)){
  column_mean_poor[i] <- mean(mtcars[[i]])
}
```

Хорошо оптимизирован для цикла

```
column_mean_optimal <- vector("numeric", length(mtcars))
for (i in seq_along(mtcars)){
  column_mean_optimal <- mean(mtcars[[i]])
}
```

Функция `vapply`

```
column_mean_vapply <- vapply(mtcars, mean, numeric(1))
```

Функция `colMeans`

```
column_mean_colMeans <- colMeans(mtcars)
```

Сравнение эффективности

Результаты сравнительного анализа этих четырех подходов показаны ниже (код не отображается)

```
Unit: microseconds
     expr     min       lq     mean   median       uq     max neval  cld
     poor 240.986 262.0820 287.1125 275.8160 307.2485 442.609   100    d
  optimal 220.313 237.4455 258.8426 247.0735 280.9130 362.469   100   c
   vapply 107.042 109.7320 124.4715 113.4130 132.6695 202.473   100 a
 colMeans 155.183 161.6955 180.2067 175.0045 194.2605 259.958   100  b
```

Обратите внимание, что оптимизированный `for` цикла выход из плохо сконструированного цикла. Плохо построенный для цикла постоянно увеличивает длину выходного объекта, и при каждом изменении длины R переоценивает класс объекта.

Часть этой накладной нагрузки удаляется оптимизированной для цикла, объявляя тип выходного объекта и его длину перед запуском цикла.

Однако в этом примере использование функции `vapply` удваивает вычислительную эффективность, в основном потому, что мы сказали R, что результат должен быть числовым (если какой-либо результат не был числовым, ошибка была бы возвращена).

Использование функции `colMeans` на ощупь медленнее, чем функция `vapply` . Эта разница объясняется некоторыми проверками ошибок, выполненными в `colMeans` и главным образом преобразованием `as.matrix` (поскольку `mtcars` - это `data.frame`), которые не выполнялись в функции `vapply` .

Другие контуры Looping: while и repeat

R обеспечивает две дополнительных конструкции зацикливания, `while` и `repeat` , которые обычно используются в ситуациях , когда число итераций требуется неопределенное.

В `while` цикл

Общий вид в `while` цикла выглядит следующим образом ,

```
while (condition) {
    ## do something
    ## in loop body
}
```

где `condition` оценивается до ввода тела цикла. Если `condition` принимает значение `TRUE` , выполняется код внутри тела цикла, и этот процесс повторяется до тех пор, пока `condition` примет значение `FALSE` (или будет достигнута инструкция `break` , см. Ниже). В отличие от `for`

цикла, если во `while` цикла используется переменная для выполнения дополнительных итераций, переменная должна быть объявлена и инициализирована раньше времени, и должен быть обновлен в теле цикла. Например, следующие циклы выполняют одну и ту же задачу:

```
for (i in 0:4) {
    cat(i, "\n")
}
# 0
# 1
# 2
# 3
# 4

i <- 0
while (i < 5) {
    cat(i, "\n")
    i <- i + 1
}
# 0
# 1
# 2
# 3
# 4
```

В `while` петли выше, линия `i <- i + 1` необходимо , чтобы предотвратить бесконечный цикл.

Кроме того, можно завершить `while` цикл с вызовом `break` внутри тела цикла:

```
iter <- 0
while (TRUE) {
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
    }
}
iter
#[1] 4
```

В этом примере `condition` всегда имеет значение `TRUE` , поэтому единственный способ завершить цикл - это вызов `break` внутри тела. Обратите внимание, что конечное значение `iter` будет зависеть от состояния вашего PRNG при запуске этого примера и должно производить разные результаты (по существу) каждый раз, когда выполняется код.

Цикл `repeat`

Конструкция `repeat` по существу такая же, как `while (TRUE) { ## something }` , и имеет следующий вид:

```
repeat ({
    ## do something
    ## in loop body
})
```

Дополнительные `{}` не требуются, но `()` . Переписывая предыдущий пример, используя `repeat` ,

```
iter <- 0
repeat ({
    if (runif(1) < 0.25) {
        break
    } else {
        iter <- iter + 1
    }
})
iter
#[1] 2
```

Подробнее о `break`

Важно отметить, что `break` *прерывает только замкнутый цикл* . То есть, это бесконечный цикл:

```
while (TRUE) {
    while (TRUE) {
        cat("inner loop\n")
        break
    }
    cat("outer loop\n")
}
```

Однако с небольшим творческим потенциалом можно полностью сломать изнутри вложенного цикла. В качестве примера рассмотрим следующее выражение, которое в своем текущем состоянии будет бесконечно зацикливаться:

```
while (TRUE) {
    cat("outer loop body\n")
    while (TRUE) {
        cat("inner loop body\n")
        x <- runif(1)
        if (x < .3) {
            break
        } else {
            cat(sprintf("x is %.5f\n", x))
        }
    }
}
```

Одна возможность состоит в том, чтобы признать , что, в отличие от `break` , то `return` выражение **не** имеет возможности вернуть контроль на нескольких уровнях вмещающих

петель. Однако, поскольку `return` действителен только при использовании внутри функции, мы не можем просто заменить `break` с `return()` выше, но также необходимо обернуть все выражение как анонимную функцию:

```
(function() {
    while (TRUE) {
        cat("outer loop body\n")
        while (TRUE) {
            cat("inner loop body\n")
            x <- runif(1)
            if (x < .3) {
                return()
            } else {
                cat(sprintf("x is %.5f\n", x))
            }
        }
    }
})()
```

В качестве альтернативы мы можем создать фиктивную переменную (`exit`) перед выражением и активировать ее через `<<-` из внутреннего цикла, когда мы готовы к завершению:

```
exit <- FALSE
while (TRUE) {
    cat("outer loop body\n")
    while (TRUE) {
        cat("inner loop body\n")
        x <- runif(1)
        if (x < .3) {
            exit <<- TRUE
            break
        } else {
            cat(sprintf("x is %.5f\n", x))
        }
    }
    if (exit) break
}
```

Прочитайте Структуры управляющих потоков онлайн: https://riptutorial.com/ru/r/topic/2201/структуры-управляющих-потоков

глава 114: Таблица данных

Вступление

Data.table - это пакет, который расширяет функциональные возможности фреймов данных из базы R, особенно улучшая их производительность и синтаксис. Подробную информацию см. В разделе «Документы» пакета в разделе « Начало работы с data.table» .

Синтаксис

- `DT[i, j, by]`
  # DT [где, выберите | update | do, by]
- `DT[...][...]`
  # цепочка
- `################# Shortcuts, special functions and special symbols inside DT[...]`
- . ()
  # в нескольких аргументах, заменяет список ()
- J ()
  # in i, заменяет список ()
- знак равно
  # in j, функция, используемая для добавления или изменения столбцов
- .N
  # в i, общее количество строк
  # в j, количество строк в группе
- .Я
  # in j, вектор номеров строк в таблице (отфильтрованный i)
- .sd
  # в j, текущее подмножество данных
  #, выбранный аргументом .SDcols
- .grp
  # в j, текущий индекс подмножества данных
- .OT
  # в j, список по значениям для текущего подмножества данных
- V1, V2, ...
  # имена по умолчанию для неназванных столбцов, созданных в j
- `################# Joins inside DT[...]`
- DT1 [DT2, on, j]
  # присоединиться к двум таблицам
- я.*
  # специальный префикс на столбцах DT2 после объединения
- от = .EACHI
  # специальный вариант доступен только с присоединением
- DT1 [! DT2, on, j]

анти-объединение двух таблиц
- DT1 [DT2, on, roll, j]
  # присоединиться к двум таблицам, перекатывая последний столбец в on =
- `################# Reshaping, stacking and splitting`
- расплав (DT, id.vars, measure.vars)
  # преобразовать в длинный формат
  # для нескольких столбцов, используйте measure.vars = patterns (...)
- dcast (DT, формула)
  # преобразовать в широкий формат
- rbind (DT1, DT2, ...)
  # stack перечислены data.tables
- rbindlist (DT_list, idcol)
  # стек список data.tables
- split (DT, by)
  # разбить таблицу data.table в список
- `################# Some other functions specialized for data.tables`
- foverlaps
  # перекрытие объединяется
- сливаться
  # другой способ соединения двух таблиц
- задавать
  # другой способ добавления или изменения столбцов
- fintersect, fsetdiff, funion, fsetequal, уникальный, дублированный, anyDuplicated
  # операции теории множеств со строками как элементами
- uniqueN
  # количество отдельных строк
- rowidv (DT, cols)
  # row ID (от 1 до .N) в каждой группе, определяемой cols
- rleidv (DT, cols)
  # идентификатор группы (от 1 до .GRP) в каждой группе, определяемый циклами cols
- shift (DT, n, type = c ("lag", "lead"))
  # применить оператор сдвига к каждому столбцу
- setorder, setcolorder, setnames, setkey, setindex, setattr
  # изменять атрибуты и порядок по ссылке

замечания

Установка и поддержка

Чтобы установить пакет data.table:

```
# install from CRAN
install.packages("data.table")
```

```
# or install development version
install.packages("data.table", type = "source", repos =
"http://Rdatatable.github.io/data.table")

# and to revert from devel to CRAN, the current version must first be removed
remove.packages("data.table")
install.packages("data.table")
```

На официальном сайте пакета есть страницы вики, которые помогают начать работу, а также списки презентаций и статей со всего Интернета. Прежде чем задавать вопрос - здесь, в StackOverflow или где-либо еще - прочитайте страницу поддержки .

Загрузка пакета

Многие функции в приведенных выше примерах существуют в пространстве имен data.table. Чтобы использовать их, вам нужно сначала добавить строку, такую как `library(data.table)` или использовать полный путь, например `data.table::fread` а не просто `fread` . Для помощи по отдельным функциям синтаксис - это `help("fread")` или « `?fread` . Опять же, если пакет не загружен, используйте полное имя типа `?data.table::fread` .

Examples

Создание таблицы данных.

Data.table - это расширенная версия класса data.frame из базы R. Таким образом, атрибут `class()` представляет собой вектор `"data.table" "data.frame"` а функции, которые работают с data.frame, также будут работа с таблицей данных. Существует множество способов создания, загрузки или принуждения к таблице данных.

строить

Не забудьте установить и активировать пакет `data.table`

```
library(data.table)
```

Существует одноименный конструктор:

```
DT <- data.table(
  x = letters[1:5],
  y = 1:5,
  z = (1:5) > 3
)
#    x y     z
# 1: a 1 FALSE
```

```
# 2: b 2 FALSE
# 3: c 3 FALSE
# 4: d 4  TRUE
# 5: e 5  TRUE
```

В отличие от `data.frame` , `data.table` не будет принуждать строки к факторам:

```
sapply(DT, class)
#               x           y           z
#     "character"   "integer"   "logical"
```

Читайте в

Мы можем читать из текстового файла:

```
dt <- fread("my_file.csv")
```

В отличие от `read.csv` , `fread` будет читать строки как строки, а не как факторы.

Измените файл data.frame

Для эффективности data.table предлагает способ изменения data.frame или списка, чтобы сделать data.table на месте (без копирования или изменения его памяти):

```
# example data.frame
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)
# modification
setDT(DF)
```

Обратите внимание, что мы не `<-` присваиваем результат, так как объект `DF` был изменен на месте. Атрибуты класса data.frame будут сохранены:

```
sapply(DF, class)
#         x         y         z
#  "factor" "integer" "logical"
```

Объект принуждения к data.table

Если у вас есть `list` , `data.frame` или `data.table` , вы должны использовать функцию `setDT` для преобразования в `data.table` потому что она делает преобразование по ссылке вместо создания копии (что делает `as.data.table`). Это важно, если вы работаете с большими наборами данных.

Если у вас есть другой объект R (такой как матрица), вы должны использовать

`as.data.table` чтобы принудить его к `data.table` .

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# or
DT <- data.table(mat)
```

Добавление и изменение столбцов

`DT[where, select|update|do, by]` используется для работы с столбцами таблицы данных.

- Часть «где» является аргументом `i`
- Часть «select | update | do» является аргументом `j`

Эти два аргумента обычно передаются положением вместо имени.

Ниже приведены приведенные ниже примеры

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

Редактирование целых столбцов

Используйте оператор `:=` внутри `j` для назначения новых столбцов:

```
mtcars[, mpg_sq := mpg^2]
```

Удалите столбцы, установив `NULL` :

```
mtcars[, mpg_sq := NULL]
```

Добавьте несколько столбцов, используя многомерный формат оператора `:=` :

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]
# or
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

Если столбцы зависят и должны быть определены последовательно, один из способов:

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

Синтаксис `.()` Используется, когда правая часть `LHS := RHS` - это список столбцов.

Для динамически определяемых имен столбцов используйте круглые скобки:

```
vn = "mpg_sq"
```

```
mtcars[, (vn) := mpg^2]
```

Столбцы также могут быть изменены с помощью `set` , хотя это редко необходимо:

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

Редактирование подмножеств столбцов

Используйте аргумент `i` для подмножества в строки «где» должны быть сделаны изменения:

```
mtcars[1:3, newvar := "Hello"]
# or
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

Как и в data.frame, мы можем подмножество, используя номера строк или логические тесты. Также возможно использовать «соединение» в `i` , но эта более сложная задача рассматривается в другом примере.

Редактирование атрибутов столбцов

Функции, редактирующие атрибуты, такие как `levels<-` или `names<-` , фактически заменяют объект на измененную копию. Даже если он используется только в одном столбце в таблице данных. Весь объект копируется и заменяется.

Чтобы изменить объект без копий, используйте `setnames` для изменения имен столбцов data.table или data.frame и `setattr` для изменения атрибута для любого объекта.

```
# Print a message to the console whenever the data.table is copied
tracemem(mtcars)
mtcars[, cyl2 := factor(cyl)]

# Neither of these statements copy the data.table
setnames(mtcars, old = "cyl2", new = "cyl_fac")
setattr(mtcars$cyl_fac, "levels", c("four", "six", "eight"))

# Each of these statements copies the data.table
names(mtcars)[names(mtcars) == "cyl_fac"] <- "cf"
levels(mtcars$cf) <- c("IV", "VI", "VIII")
```

Имейте в виду, что эти изменения сделаны по ссылке, поэтому они являются *глобальными* . Изменение их в одной среде влияет на объект во всех средах.

```
# This function also changes the levels in the global environment
edit_levels <- function(x) setattr(x, "levels", c("low", "med", "high"))
edit_levels(mtcars$cyl_factor)
```

.sd

`.SD` относится к подмножеству `data.table` для каждой группы, за исключением всех столбцов , используемых в `by` .

`.SD` вместе с `lapply` может использоваться для применения любой функции к нескольким столбцам по группам в `data.table`

Мы продолжим использовать тот же встроенный набор данных, `mtcars` :

```
mtcars = data.table(mtcars) # Let's not include rownames to keep things simpler
```

Среднее значение всех столбцов в наборе данных по *количеству цилиндров* , `cyl` :

```
mtcars[ , lapply(.SD, mean), by = cyl]

#   cyl      mpg     disp        hp     drat       wt     qsec        vs        am     gear
carb
#1:   6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
#2:   4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
#3:   8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

Помимо `cyl` , в наборе данных есть другие категориальные столбцы, такие как `vs` , `am` , `gear` и `carb` . На самом деле нет смысла воспринимать `mean` этих столбцов. Поэтому давайте исключаем эти столбцы. Это где `.SDcols` входит в картину.

.SDcols

`.SDcols` указывает столбцы `data.table` , которые включены в `.SD` .

Среднее значение всех столбцов (сплошные столбцы) в наборе данных по *количеству передач* `gear` , и *количество цилиндров,* `cyl` , устроенных `gear` и `cyl` :

```
# All the continuous variables in the dataset
cols_chosen <- c("mpg", "disp", "hp", "drat", "wt", "qsec")

mtcars[order(gear, cyl), lapply(.SD, mean), by = .(gear, cyl), .SDcols = cols_chosen]

#   gear cyl    mpg     disp       hp     drat       wt    qsec
#1:    3   4 21.500 120.1000  97.0000 3.700000 2.465000 20.0100
#2:    3   6 19.750 241.5000 107.5000 2.920000 3.337500 19.8300
#3:    3   8 15.050 357.6167 194.1667 3.120833 4.104083 17.1425
#4:    4   4 26.925 102.6250  76.0000 4.110000 2.378125 19.6125
```

```
#5:    4   6 19.750 163.8000 116.5000 3.910000 3.093750 17.6700
#6:    5   4 28.200 107.7000 102.0000 4.100000 1.826500 16.8000
#7:    5   6 19.700 145.0000 175.0000 3.620000 2.770000 15.5000
#8:    5   8 15.400 326.0000 299.5000 3.880000 3.370000 14.5500
```

Может быть, мы не хотим вычислять `mean` по группам. Чтобы вычислить среднее значение для всех автомобилей в наборе данных, мы не укажем переменную `by` .

```
mtcars[ , lapply(.SD, mean), .SDcols = cols_chosen]

#        mpg     disp       hp     drat      wt     qsec
#1: 20.09062 230.7219 146.6875 3.596563 3.21725 17.84875
```

Замечания:

- Нет необходимости заранее определять `cols_chosen` . `.SDcols` могут напрямую принимать имена столбцов
- `.SDcols` также могут непосредственно брать вектор столбчатых чисел. В приведенном выше примере это будут `mtcars[ , lapply(.SD, mean), .SDcols = c(1,3:7)]`

.N

`.N` - сокращенное число строк в группе.

```
iris[, .(count=.N), by=Species]

#      Species count
#1:     setosa    50
#2: versicolor    50
#3:  virginica    50
```

Написание кода, совместимого как с data.frame, так и с data.table

Различия в синтаксисе подмножества

`data.table` является одним из нескольких двумерных структур данных , доступных в R, помимо `data.frame` , `matrix` и (2D) `array` . Все эти классы используют очень похожий, но не идентичный синтаксис для подмножества, схему `A[rows, cols]` .

Рассмотрим следующие данные, хранящиеся в `matrix` : `data.frame` и `data.table` :

```
ma <- matrix(rnorm(12), nrow=4, dimnames=list(letters[1:4], c('X', 'Y', 'Z')))
df <- as.data.frame(ma)
dt <- as.data.table(ma)

ma[2:3]  #---> returns the 2nd and 3rd items, as if 'ma' were a vector (because it is!)
df[2:3]  #---> returns the 2nd and 3rd columns
```

```
dt[2:3]  #---> returns the 2nd and 3rd rows!
```

Если вы хотите быть уверенными в том, что будет возвращено, лучше быть *явным* .

Чтобы получить определенные **строки** , просто добавьте запятую после диапазона:

```
ma[2:3, ]  # \
df[2:3, ]  #  }---> returns the 2nd and 3rd rows
dt[2:3, ]  # /
```

Но, если вы хотите подмножество **столбцов** , некоторые случаи интерпретируются по-разному. Все три могут быть подмножеством одинаковым образом с целыми или символьными индексами, *не* сохраненными в переменной.

```
ma[, 2:3]          #  \
df[, 2:3]          #   \
dt[, 2:3]          #    }---> returns the 2nd and 3rd columns
ma[, c("Y", "Z")]  #   /
df[, c("Y", "Z")]  #  /
dt[, c("Y", "Z")]  # /
```

Однако они отличаются для имен без кавычек

```
mycols <- 2:3
ma[, mycols]                 # \
df[, mycols]                 #  }---> returns the 2nd and 3rd columns
dt[, mycols, with = FALSE]   # /

dt[, mycols]                 # ---> Raises an error
```

В последнем случае `mycols` оцениваются как имя столбца. Поскольку `dt` не может найти столбец с именем `mycols` , возникает ошибка.

Примечание: Для версии `data.table` пакета priorto 1.9.8, такое поведение было несколько иначе. Все, что было в индексе столбца, было бы оценено с использованием `dt` в качестве среды. Таким образом, оба `dt[, 2:3]` и `dt[, mycols]` вернут вектор `2:3` . `mycols` втором случае `mycols` не `mycols` , потому что переменная `mycols` существует в родительской среде.

Стратегии обеспечения совместимости с data.frame и data.table

Есть много причин писать код, который гарантированно работает с `data.frame` и `data.table` . Возможно, вы вынуждены использовать `data.frame` , или вам может понадобиться поделиться некоторым кодом, который вы не знаете, как будет использоваться. Итак, для удобства есть некоторые основные стратегии для достижения этого:

1. Используйте синтаксис, который ведет себя одинаково для обоих классов.
2. Используйте общую функцию, которая делает то же самое, что и самый короткий синтаксис.
3. Принудите `data.table` вести себя как `data.frame` (например: вызов определенного метода `print.data.frame`).
4. Рассматривайте их как `list` , который они в конечном счете.
5. Преобразуйте таблицу в `data.frame` прежде чем что-либо делать (плохая идея, если это огромная таблица).
6. Преобразуйте таблицу в таблицу `data.table` , если зависимости не являются проблемой.

Строки подмножества. Просто, просто используйте селектор `[, ]` *с* запятой:

```
A[1:10, ]
A[A$var > 17, ]  # A[var > 17, ] just works for data.table
```

Подстрочные столбцы. Если вам нужен один столбец, используйте селектор `$` или `[[ ]]` :

```
A$var
colname <- 'var'
A[[colname]]
A[[1]]
```

Если вам нужен единый способ захватить более одного столбца, необходимо немного обжаловать:

```
B <- `[.data.frame`(A, 2:4)

# We can give it a better name
select <- `[.data.frame`
B <- select(A, 2:4)
C <- select(A, c('foo', 'bar'))
```

Подстрочные «индексированные» строки. Хотя `data.frame` имеет `row.names` , `data.table` имеет свою уникальную `key` функцию. Лучше всего избегать `row.names` и использовать существующие оптимизации в случае `data.table` когда это возможно.

```
B <- A[A$var != 0, ]
# or...
B <- with(A, A[var != 0, ])  # data.table will silently index A by var before subsetting

stuff <- c('a', 'c', 'f')
C <- A[match(stuff, A$name), ]  # really worse than: setkey(A); A[stuff, ]
```

Получите таблицу с 1 столбцом, получите строку как вектор. Это легко понять, что мы видели до сих пор:

```
B <- select(A, 2)    #---> a table with just the second column
C <- unlist(A[1, ])  #---> the first row as a vector (coerced if necessary)
```

Установка ключей в data.table

Да, вам нужно SETKEY до 1.9.6

В прошлом (pre 1.9.6) ваша `data.table` установкой столбцов в качестве ключей к таблице, особенно для больших таблиц. [См. Интро-виньетку на стр. 5 от версии 2015 года, где скорость поиска была в 544 раза лучше.] Вы можете найти более старый код, используя эти установочные ключи с помощью «setkey» или установки столбца «key =» при настройке таблицы.

```
library(data.table)
DT <- data.table(
  x = letters[1:5],
  y = 5:1,
  z = (1:5) > 3
)

#> DT
#   x y     z
#1: a 5 FALSE
#2: b 4 FALSE
#3: c 3 FALSE
#4: d 2  TRUE
#5: e 1  TRUE
```

Установите ключ с `setkey` команды `setkey` . У вас может быть ключ с несколькими столбцами.

```
setkey(DT, y)
```

Проверьте ключ таблицы в таблицах ()

```
tables()

> tables()
     NAME NROW NCOL MB COLS  KEY
[1,] DT      5    3  1 x,y,z y
Total: 1MB
```

Обратите внимание, что это приведет к повторной сортировке ваших данных.

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

Теперь это не нужно

До v1.9.6 вам нужно было установить ключ для определенных операций, особенно соединяя таблицы. Разработчики data.table ускорились и внедрили функцию `"on="` которая может заменить зависимость от ключей. См. « Ответ» здесь для подробного обсуждения.

В январе 2017 года разработчики написали виньетку вокруг вторичных индексов, которая объясняет синтаксис «on» и позволяет идентифицировать другие столбцы для быстрой индексации.

Создание вторичных индексов?

Аналогично ключу можно `setindex(DT, key.col)` или `setindexv(DT, "key.col.string")` , где DT - ваша таблица данных. Удалите все индексы с помощью `setindex(DT, NULL)` .

См. Ваши вторичные индексы с `indices(DT)` .

Почему вторичные индексы?

Это **не сортирует** таблицу (в отличие от ключа), но позволяет быстро индексировать с помощью синтаксиса «on». Обратите внимание, что может быть только один ключ, но вы можете использовать несколько вторичных индексов, что экономит необходимость повторного подключения и использования таблицы. Это ускорит подмножество при изменении столбцов, на которые вы хотите настроить.

Напомним, в примере выше у был ключом для таблицы DT:

```
DT
# x y     z
# 1: e 1  TRUE
# 2: d 2  TRUE
# 3: c 3 FALSE
# 4: b 4 FALSE
# 5: a 5 FALSE

# Let us set x as index
setindex(DT, x)

# Use indices to see what has been set
indices(DT)
# [1] "x"

# fast subset using index and not keyed column
DT["c", on ="x"]
#x y     z
#1: c 3 FALSE

# old way would have been rekeying DT from y to x, doing subset and
# perhaps keying back to y (now we save two sorts)
# This is a toy example above but would have been more valuable with big data sets
```

Прочитайте Таблица данных онлайн: https://riptutorial.com/ru/r/topic/849/таблица-данных

глава 115: Текстовая обработка

Examples

Скремблирование данных для создания N-грамм Word Clouds

В следующем примере используется текстовый интеллектуальный пакет `tm` для очистки и передачи текстовых данных из Интернета для создания облаков слов с символическим затенением и упорядочением.

```
require(RWeka)
require(tau)
require(tm)
require(tm.plugin.webmining)
require(wordcloud)

# Scrape Google Finance ------------------------------------------------------
googlefinance <- WebCorpus(GoogleFinanceSource("NASDAQ:LFVN"))

# Scrape Google News ---------------------------------------------------------
lv.googlenews <- WebCorpus(GoogleNewsSource("LifeVantage"))
p.googlenews  <- WebCorpus(GoogleNewsSource("Protandim"))
ts.googlenews <- WebCorpus(GoogleNewsSource("TrueScience"))

# Scrape NYTimes -------------------------------------------------------------
lv.nytimes <- WebCorpus(NYTimesSource(query = "LifeVantage", appid = nytimes_appid))
p.nytimes  <- WebCorpus(NYTimesSource("Protandim", appid = nytimes_appid))
ts.nytimes <- WebCorpus(NYTimesSource("TrueScience", appid = nytimes_appid))

# Scrape Reuters -------------------------------------------------------------
lv.reutersnews <- WebCorpus(ReutersNewsSource("LifeVantage"))
p.reutersnews  <- WebCorpus(ReutersNewsSource("Protandim"))
ts.reutersnews <- WebCorpus(ReutersNewsSource("TrueScience"))

# Scrape Yahoo! Finance ------------------------------------------------------
lv.yahoofinance <- WebCorpus(YahooFinanceSource("LFVN"))

# Scrape Yahoo! News ---------------------------------------------------------
lv.yahoonews <- WebCorpus(YahooNewsSource("LifeVantage"))
p.yahoonews  <- WebCorpus(YahooNewsSource("Protandim"))
ts.yahoonews <- WebCorpus(YahooNewsSource("TrueScience"))

# Scrape Yahoo! Inplay -------------------------------------------------------
lv.yahooinplay <- WebCorpus(YahooInplaySource("LifeVantage"))

# Text Mining the Results ----------------------------------------------------
corpus <- c(googlefinance, lv.googlenews, p.googlenews, ts.googlenews, lv.yahoofinance,
lv.yahoonews, p.yahoonews,
ts.yahoonews, lv.yahooinplay) #lv.nytimes, p.nytimes, ts.nytimes,lv.reutersnews,
p.reutersnews, ts.reutersnews,

inspect(corpus)
wordlist <- c("lfvn", "lifevantage", "protandim", "truescience", "company", "fiscal",
"nasdaq")
```

```
ds0.1g <- tm_map(corpus, content_transformer(tolower))
ds1.1g <- tm_map(ds0.1g, content_transformer(removeWords), wordlist)
ds1.1g <- tm_map(ds1.1g, content_transformer(removeWords), stopwords("english"))
ds2.1g <- tm_map(ds1.1g, stripWhitespace)
ds3.1g <- tm_map(ds2.1g, removePunctuation)
ds4.1g <- tm_map(ds3.1g, stemDocument)

tdm.1g <- TermDocumentMatrix(ds4.1g)
dtm.1g <- DocumentTermMatrix(ds4.1g)

findFreqTerms(tdm.1g, 40)
findFreqTerms(tdm.1g, 60)
findFreqTerms(tdm.1g, 80)
findFreqTerms(tdm.1g, 100)

findAssocs(dtm.1g, "skin", .75)
findAssocs(dtm.1g, "scienc", .5)
findAssocs(dtm.1g, "product", .75)

tdm89.1g <- removeSparseTerms(tdm.1g, 0.89)
tdm9.1g  <- removeSparseTerms(tdm.1g, 0.9)
tdm91.1g <- removeSparseTerms(tdm.1g, 0.91)
tdm92.1g <- removeSparseTerms(tdm.1g, 0.92)

tdm2.1g <- tdm92.1g

# Creates a Boolean matrix (counts # docs w/terms, not raw # terms)
tdm3.1g <- inspect(tdm2.1g)
tdm3.1g[tdm3.1g>=1] <- 1

# Transform into a term-term adjacency matrix
termMatrix.1gram <- tdm3.1g %*% t(tdm3.1g)

# inspect terms numbered 5 to 10
termMatrix.1gram[5:10,5:10]
termMatrix.1gram[1:10,1:10]

# Create a WordCloud to Visualize the Text Data ----------------------------
notsparse <- tdm2.1g
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)
```

Обратите внимание на использование `random.order` и последовательного поддона RColorBrewer, который позволяет программисту захватывать больше информации в облаке, присваивая смысл порядку и раскраске терминов.

Выше 1-граммовый случай.

Мы можем сделать большой скачок к облакам слов n-грамм, и при этом мы увидим, как сделать практически любой анализ текстового анализа достаточно гибким, чтобы обрабатывать n-граммы, преобразовывая наш TDM.

Первоначальная трудность, с которой вы столкнулись с n-граммами в R, состоит в том, что `tm` , самый популярный пакет для интеллектуального анализа текста, не предусматривает поддержку токенизации биграмм или n-граммов. Токенизация - это процесс представления слова, части слова или группы слов (или символов) в виде единого элемента данных, называемого токеном.

К счастью, у нас есть некоторые хаки, которые позволяют нам продолжать использовать `tm` с обновленным токенизатором. Существует не один способ добиться этого. Мы можем написать собственный простой токенизатор, используя `textcnt()` из tau:

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

или мы можем использовать `RWeka` RWeka в `tm` :

```
# BigramTokenize
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

С этого момента вы можете действовать так же, как в случае с 1 графом:

```
# Create an n-gram Word Cloud ------------------------------------------------
tdm.ng <- TermDocumentMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))
dtm.ng <- DocumentTermMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))

# Try removing sparse terms at a few different levels
tdm89.ng <- removeSparseTerms(tdm.ng, 0.89)
tdm9.ng  <- removeSparseTerms(tdm.ng, 0.9)
tdm91.ng <- removeSparseTerms(tdm.ng, 0.91)
tdm92.ng <- removeSparseTerms(tdm.ng, 0.92)

notsparse <- tdm91.ng
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)
```

Пример, приведенный выше, воспроизводится с разрешения блога Science Science от Hack-R. Дополнительный комментарий можно найти в оригинальной статье.

Прочитайте Текстовая обработка онлайн: https://riptutorial.com/ru/r/topic/3579/текстовая-

обработка

глава 116: тепловая карта и тепловая карта.2

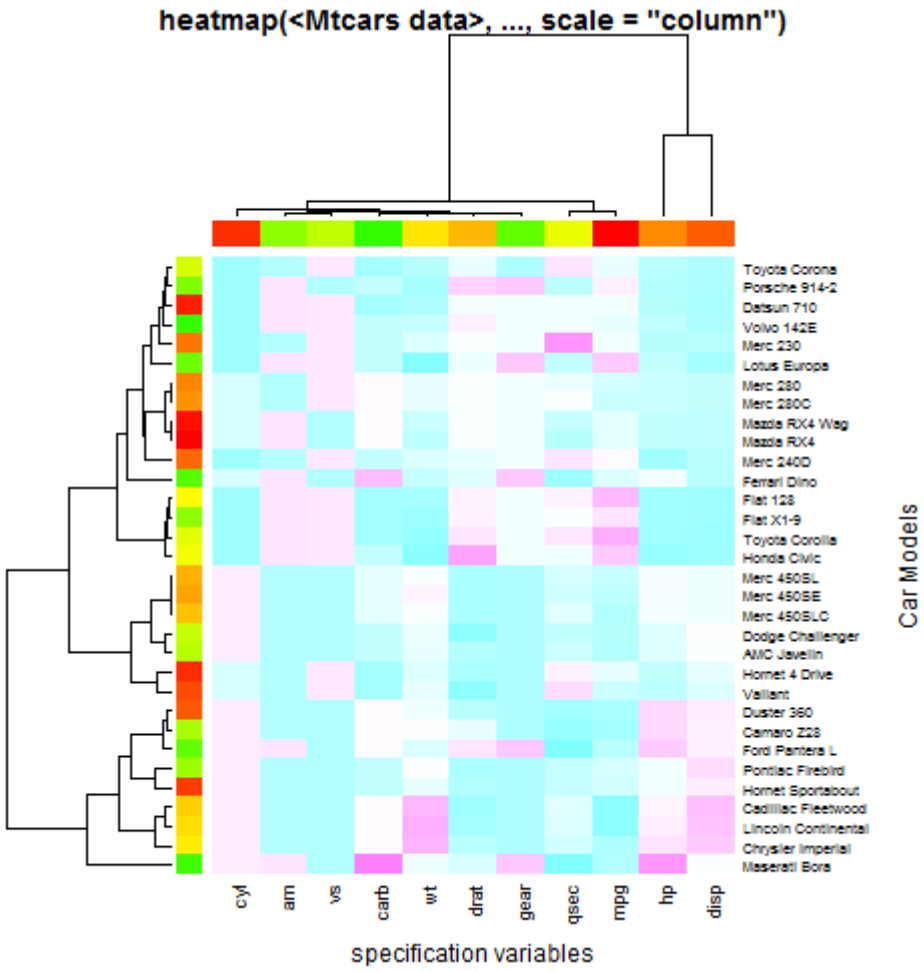
Examples

Примеры из официальной документации

Статистика :: Heatmap

Пример 1 (Основное использование)

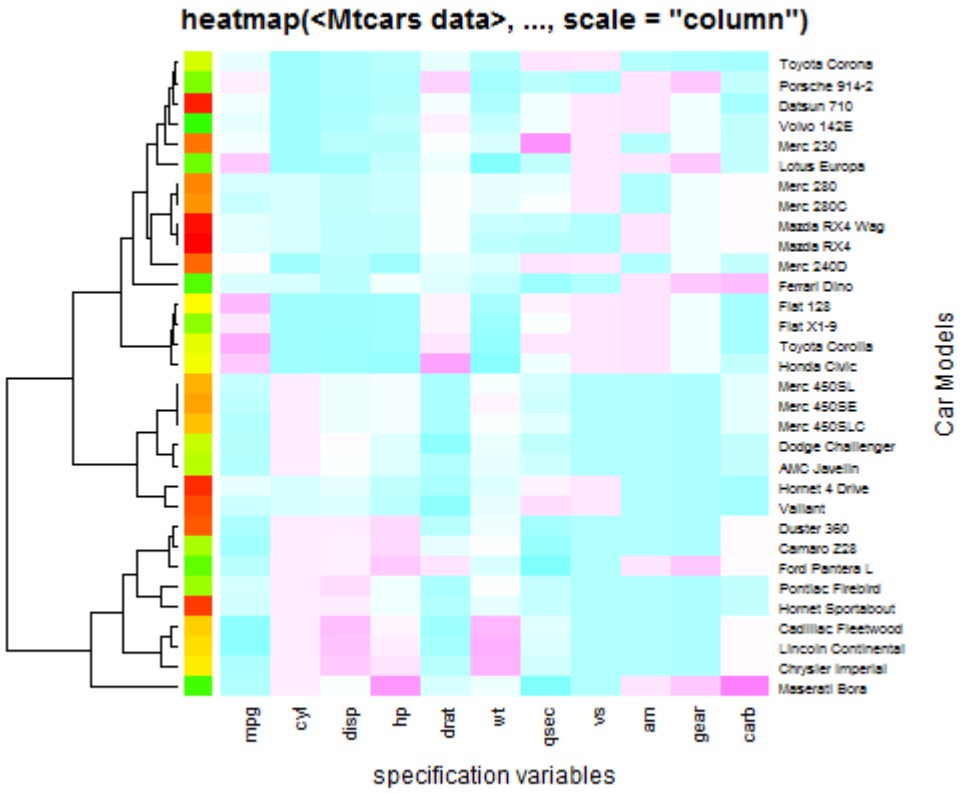
```
require(graphics); require(grDevices)
x  <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
hv <- heatmap(x, col = cm.colors(256), scale = "column",
              RowSideColors = rc, ColSideColors = cc, margins = c(5,10),
              xlab = "specification variables", ylab =  "Car Models",
              main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```



```
utils::str(hv) # the two re-ordering index vectors
# List of 4
#  $ rowInd: int [1:32] 31 17 16 15 5 25 29 24 7 6 ...
#  $ colInd: int [1:11] 2 9 8 11 6 5 10 7 1 4 ...
#  $ Rowv  : NULL
#  $ Colv  : NULL
```

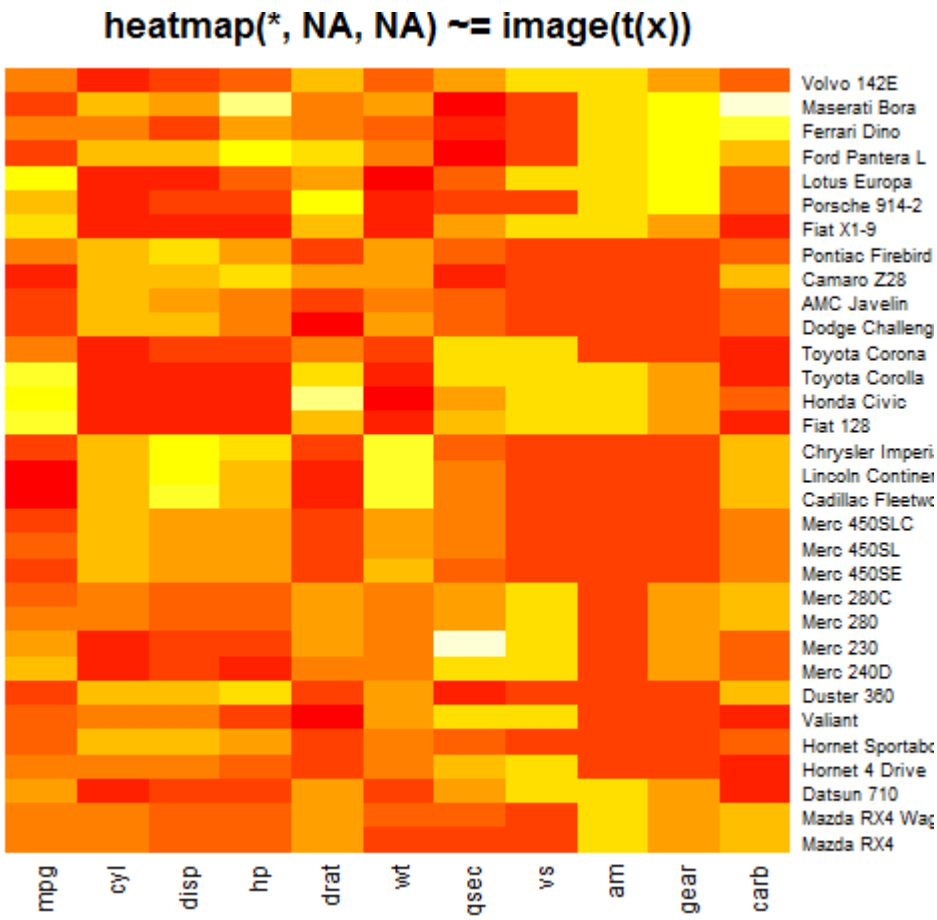
Пример 2 (вообще отсутствует дендрограмма столбцов (или переупорядочение))

```
heatmap(x, Colv = NA, col = cm.colors(256), scale = "column",
        RowSideColors = rc, margins = c(5,10),
        xlab = "specification variables", ylab =  "Car Models",
        main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```



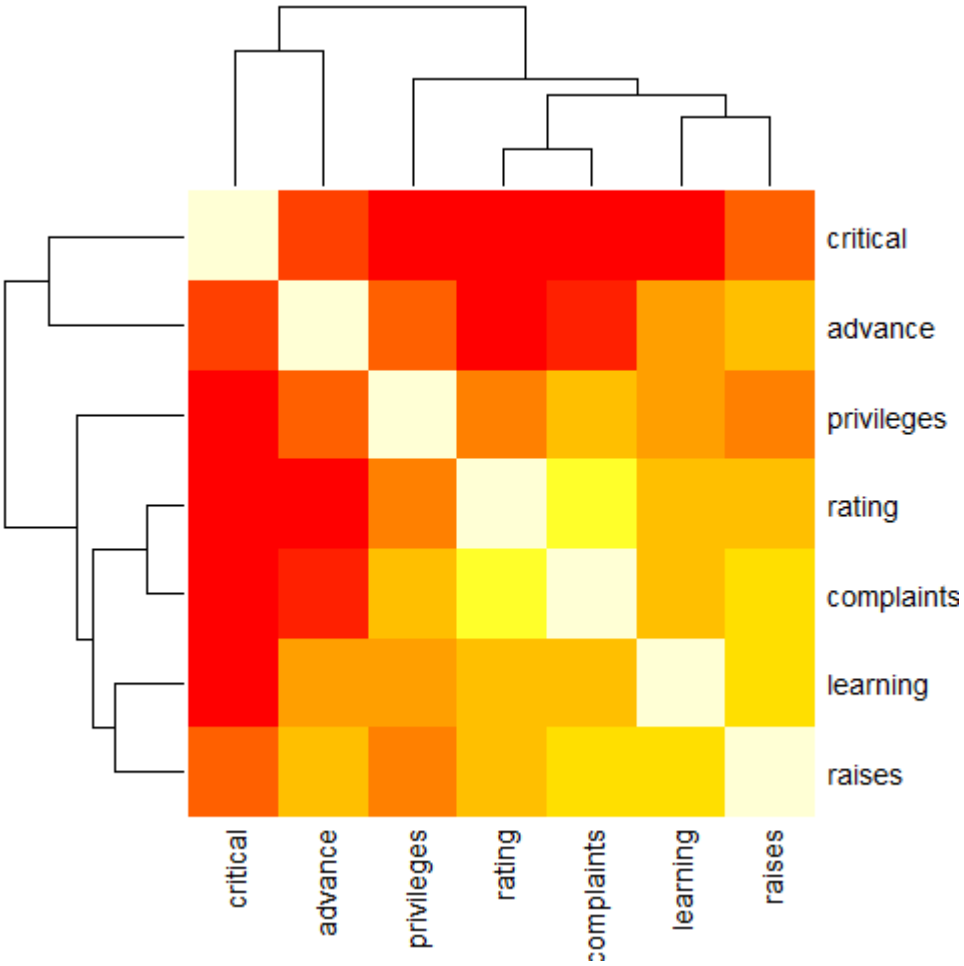
Пример 3 («нет ничего»)

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
        main = "heatmap(*, NA, NA) ~= image(t(x))")
```



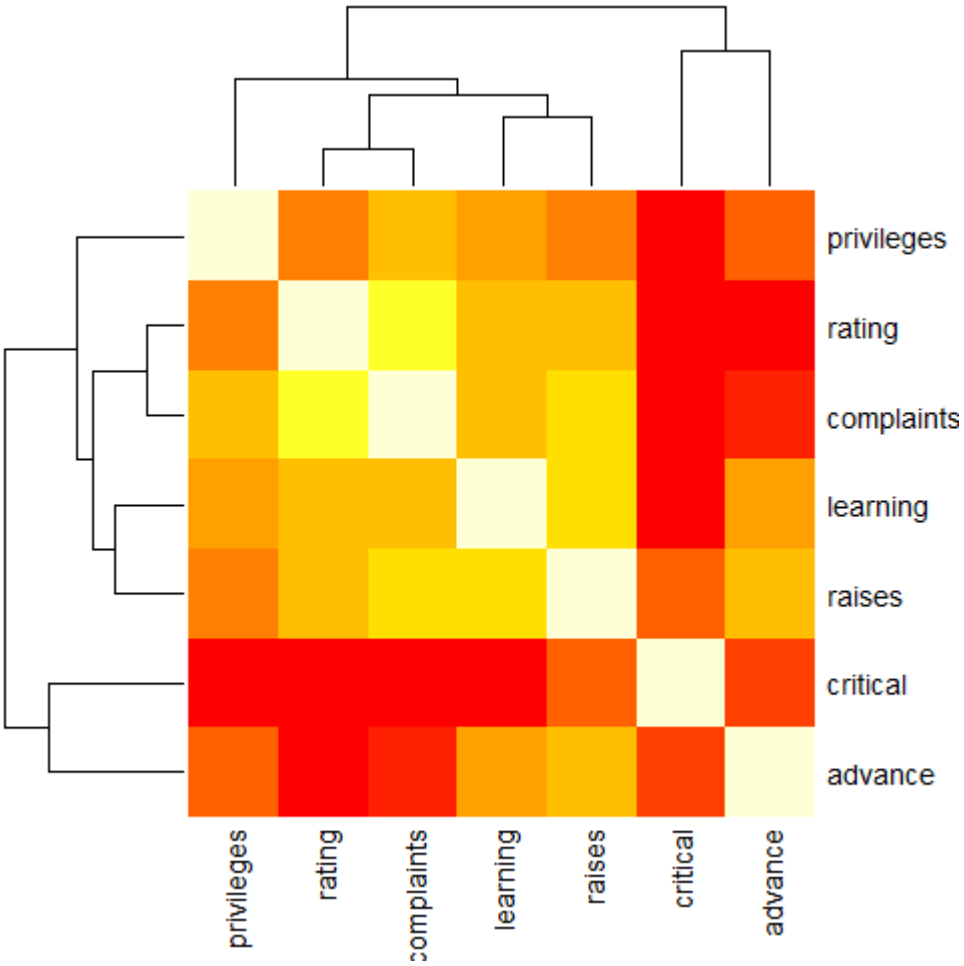
Пример 4 (с переупорядочением ())

```
round(Ca <- cor(attitude), 2)
#            rating complaints privileges learning raises critical advance
# rating       1.00       0.83       0.43     0.62   0.59     0.16    0.16
# complaints   0.83       1.00       0.56     0.60   0.67     0.19    0.22
# privileges   0.43       0.56       1.00     0.49   0.45     0.15    0.34
# learning     0.62       0.60       0.49     1.00   0.64     0.12    0.53
# raises       0.59       0.67       0.45     0.64   1.00     0.38    0.57
# critical     0.16       0.19       0.15     0.12   0.38     1.00    0.28
# advance      0.16       0.22       0.34     0.53   0.57     0.28    1.00
symnum(Ca) # simple graphic
#            rt cm p l rs cr a
# rating     1
# complaints +  1
# privileges .  .  1
# learning   ,  .  . 1
# raises     .  ,  . , 1
# critical             .  1
# advance          . . .     1
# attr(,"legend")
# [1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
heatmap(Ca,               symm = TRUE, margins = c(6,6))
```



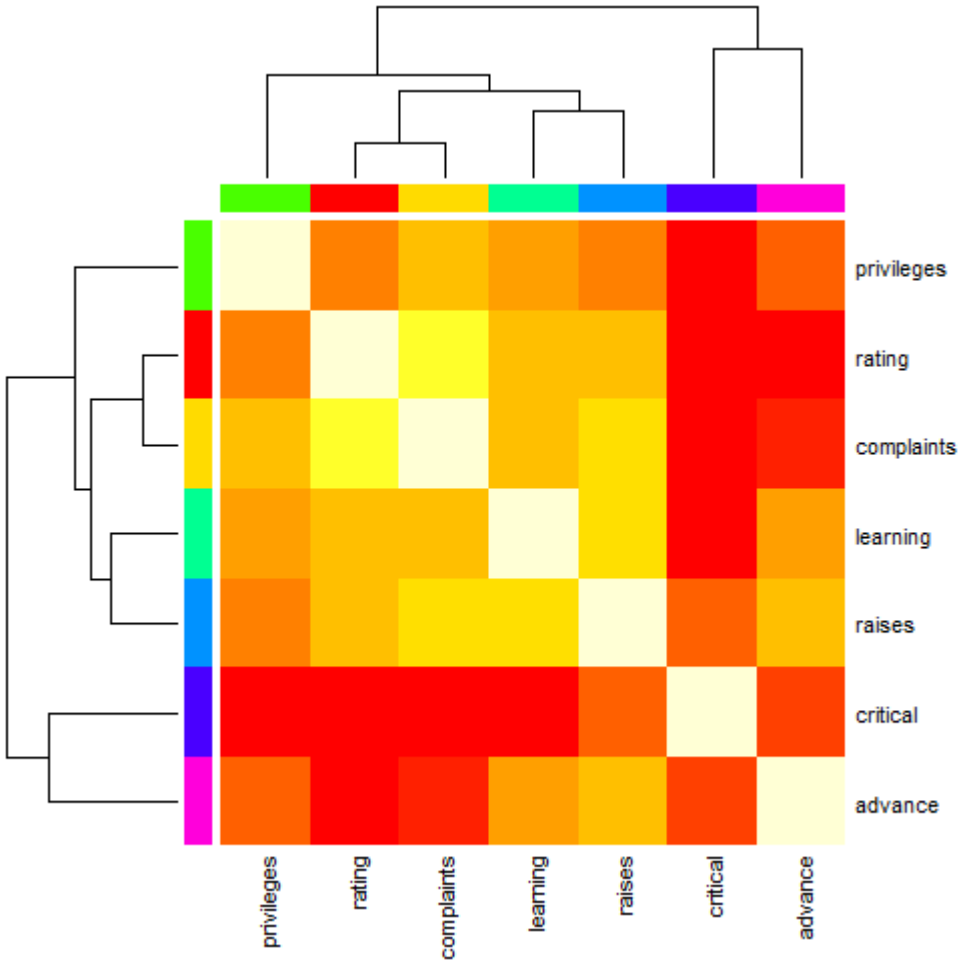
Пример 5 (*NO* reorder ())

```
heatmap(Ca, Rowv = FALSE, symm = TRUE, margins = c(6,6))
```



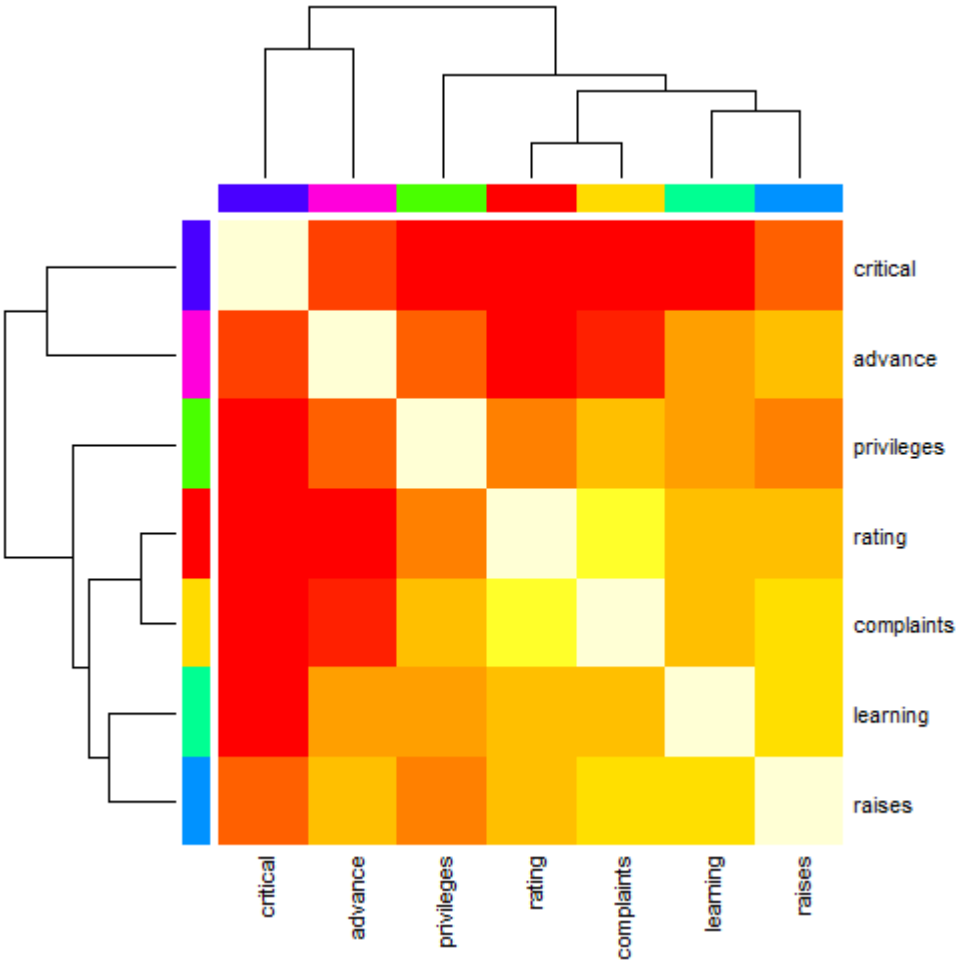
Пример 6 (слегка искусственный с цветовой полосой без заказа)

```
cc <- rainbow(nrow(Ca))
heatmap(Ca, Rowv = FALSE, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Пример 7 (слегка искусственный с цветовой полосой, с упорядочением)

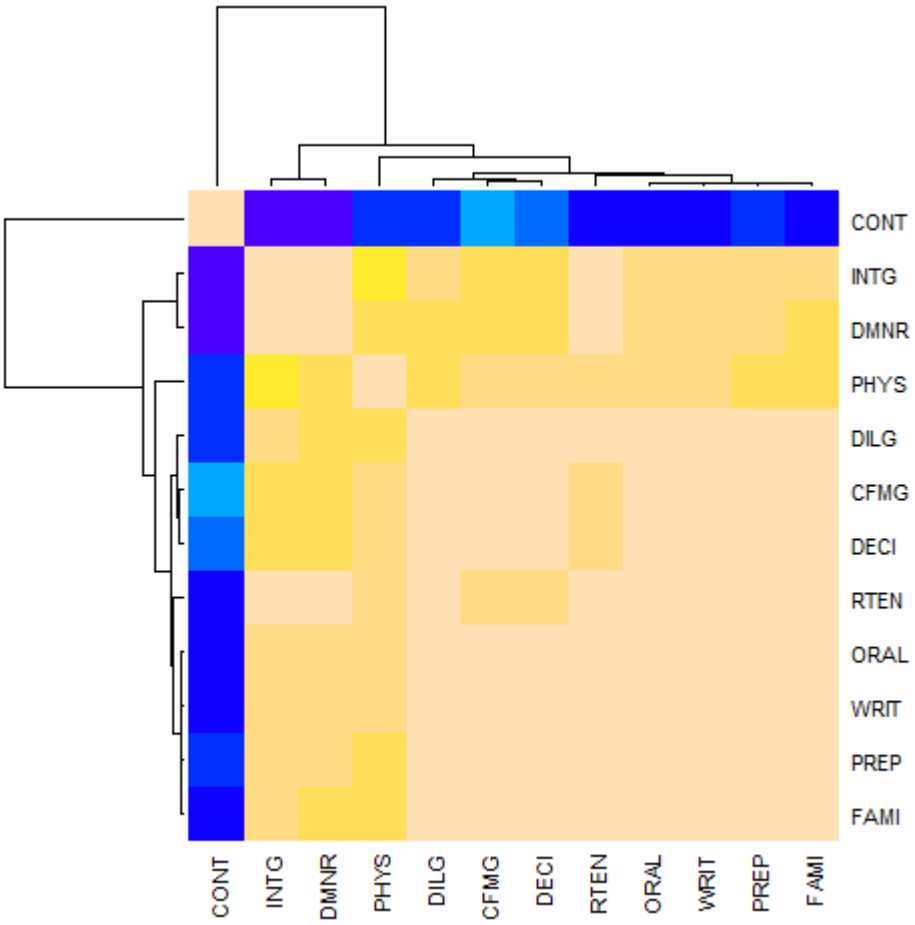
```
heatmap(Ca,         symm = TRUE, RowSideColors = cc, ColSideColors = cc,
    margins = c(6,6))
```



Пример 8 (для переменной кластеризации, скорее используйте расстояние, основанное на cor ())

```
symnum( cU <- cor(USJudgeRatings) )
#      CO I DM DI CF DE PR F O W PH R
# CONT 1
# INTG    1
# DMNR    B 1
# DILG    + +  1
# CFMG    + +  B  1
# DECI    + +  B  B  1
# PREP    + +  B  B  B  1
# FAMI    + +  B  *  *  B  1
# ORAL    * *  B  B  *  B  B 1
# WRIT    * +  B  *  *  B  B B 1
# PHYS    , ,  +  +  +  +  + + + 1
# RTEN    * *  *  *  *  B  * B B *  1
# attr(,"legend")
# [1] 0 ' ' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1

hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
              distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)
```




```
## The Correlation matrix with same reordering:
round(100 * cU[hU[[1]], hU[[2]]])
#      CONT INTG DMNR PHYS DILG CFMG DECI RTEN ORAL WRIT PREP FAMI
# CONT  100  -13  -15    5    1   14    9   -3   -1   -4    1   -3
# INTG  -13  100   96   74   87   81   80   94   91   91   88   87
# DMNR  -15   96  100   79   84   81   80   94   91   89   86   84
# PHYS    5   74   79  100   81   88   87   91   89   86   85   84
# DILG    1   87   84   81  100   96   96   93   95   96   98   96
# CFMG   14   81   81   88   96  100   98   93   95   94   96   94
# DECI    9   80   80   87   96   98  100   92   95   95   96   94
# RTEN   -3   94   94   91   93   93   92  100   98   97   95   94
# ORAL   -1   91   91   89   95   95   95   98  100   99   98   98
# WRIT   -4   91   89   86   96   94   95   97   99  100   99   99
# PREP    1   88   86   85   98   96   96   95   98   99  100   99
# FAMI   -3   87   84   84   96   94   94   94   98   99   99  100
```

```
## The column dendrogram:
utils::str(hU$Colv)
# --[dendrogram w/ 2 branches and 12 members at h = 1.15]
#   |--leaf "CONT"
#   `--[dendrogram w/ 2 branches and 11 members at h = 0.258]
#      |--[dendrogram w/ 2 branches and 2 members at h = 0.0354]
#      |  |--leaf "INTG"
#      |  `--leaf "DMNR"
#      `--[dendrogram w/ 2 branches and 9 members at h = 0.187]
#         |--leaf "PHYS"
#         `--[dendrogram w/ 2 branches and 8 members at h = 0.075]
#            |--[dendrogram w/ 2 branches and 3 members at h = 0.0438]
#            |  |--leaf "DILG"
```

```
#             |  `--[dendrogram w/ 2 branches and 2 members at h = 0.0189]
#             |     |--leaf "CFMG"
#             |     `--leaf "DECI"
#             `--[dendrogram w/ 2 branches and 5 members at h = 0.0584]
#                |--leaf "RTEN"
#                `--[dendrogram w/ 2 branches and 4 members at h = 0.0187]
#                   |--[dendrogram w/ 2 branches and 2 members at h = 0.00657]
#                   |  |--leaf "ORAL"
#                   |  `--leaf "WRIT"
#                   `--[dendrogram w/ 2 branches and 2 members at h = 0.0101]
#                      |--leaf "PREP"
#                      `--leaf "FAMI"
```

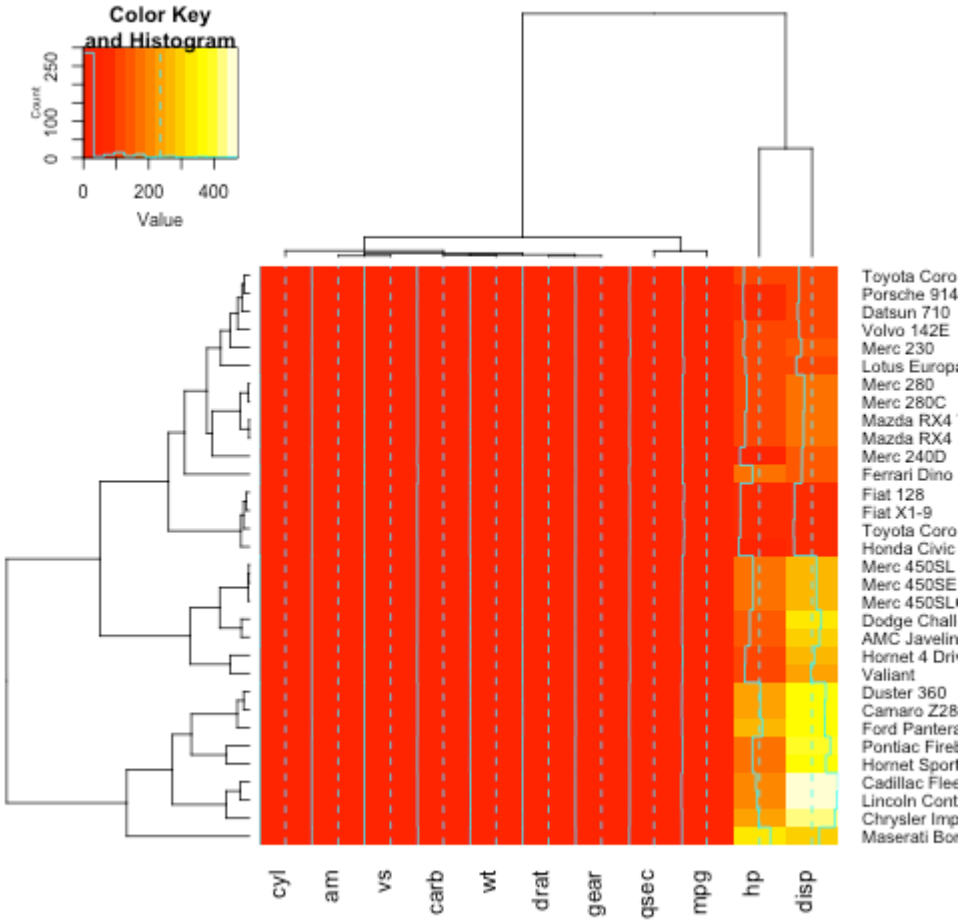
Параметры настройки в тепловой карте.2

Дано:

```
x  <- as.matrix(mtcars)
```

Можно использовать `heatmap.2` - более свежую оптимизированную версию `heatmap` , загрузив следующую библиотеку:

```
require(gplots)
heatmap.2(x)
```

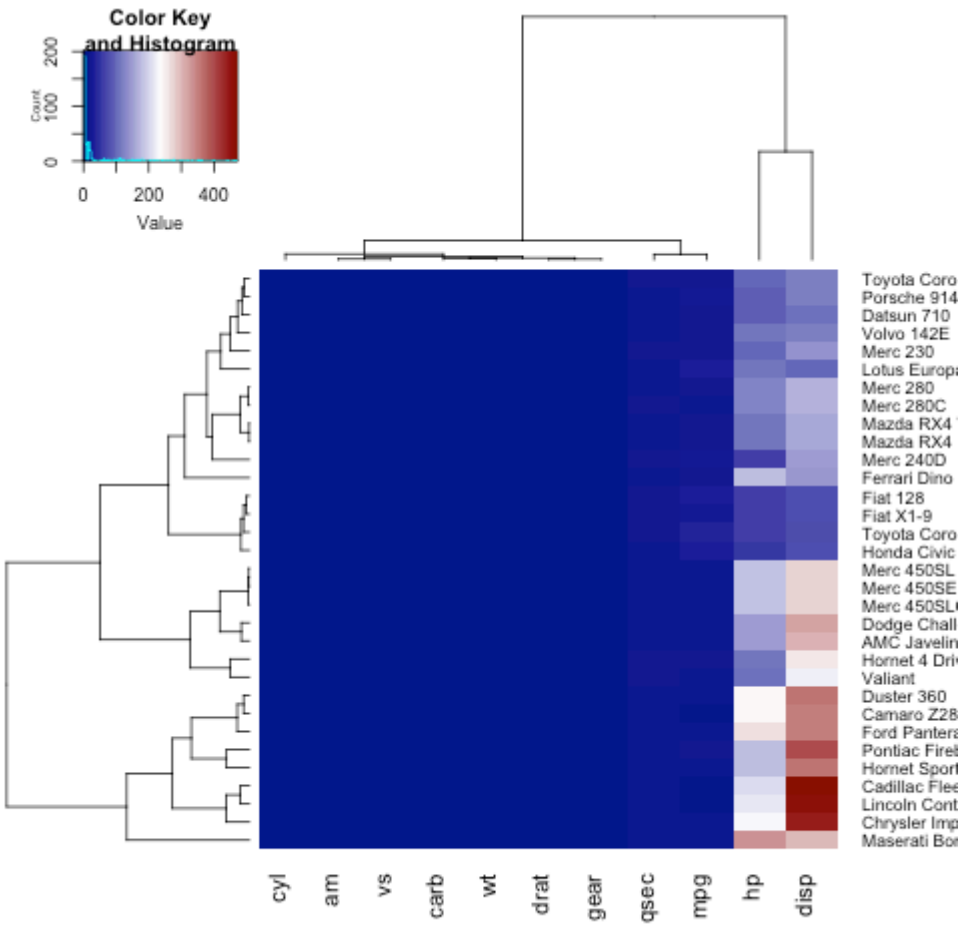


Чтобы добавить заголовок, x- или y-метку в вашу карту памяти, вам нужно установить `main` , `xlab` и `ylab` :

```
heatmap.2(x, main = "My main title: Overview of car features", xlab="Car features", ylab =
"Car brands")
```

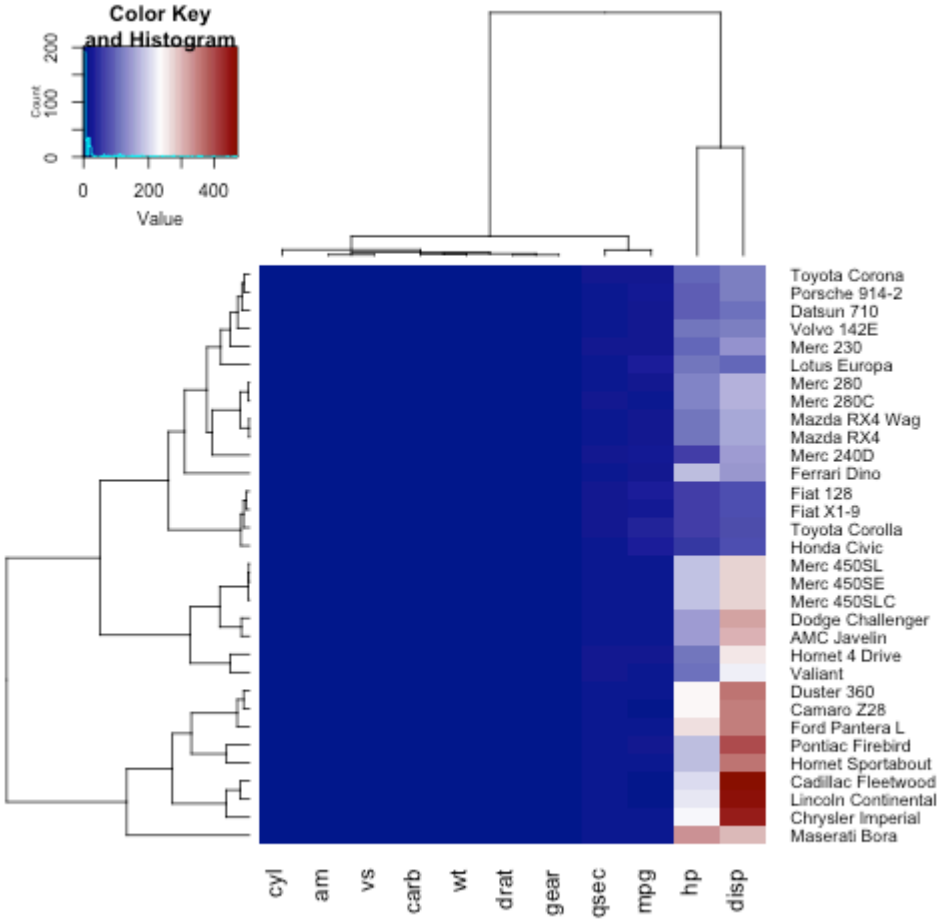
Если вы хотите определить свою собственную цветовую палитру для своей `colorRampPalette` , вы можете установить параметр `col` с помощью функции `colorRampPalette` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE,dendrogram = "row",col =
colorRampPalette(c("darkblue","white","darkred"))(100))
```

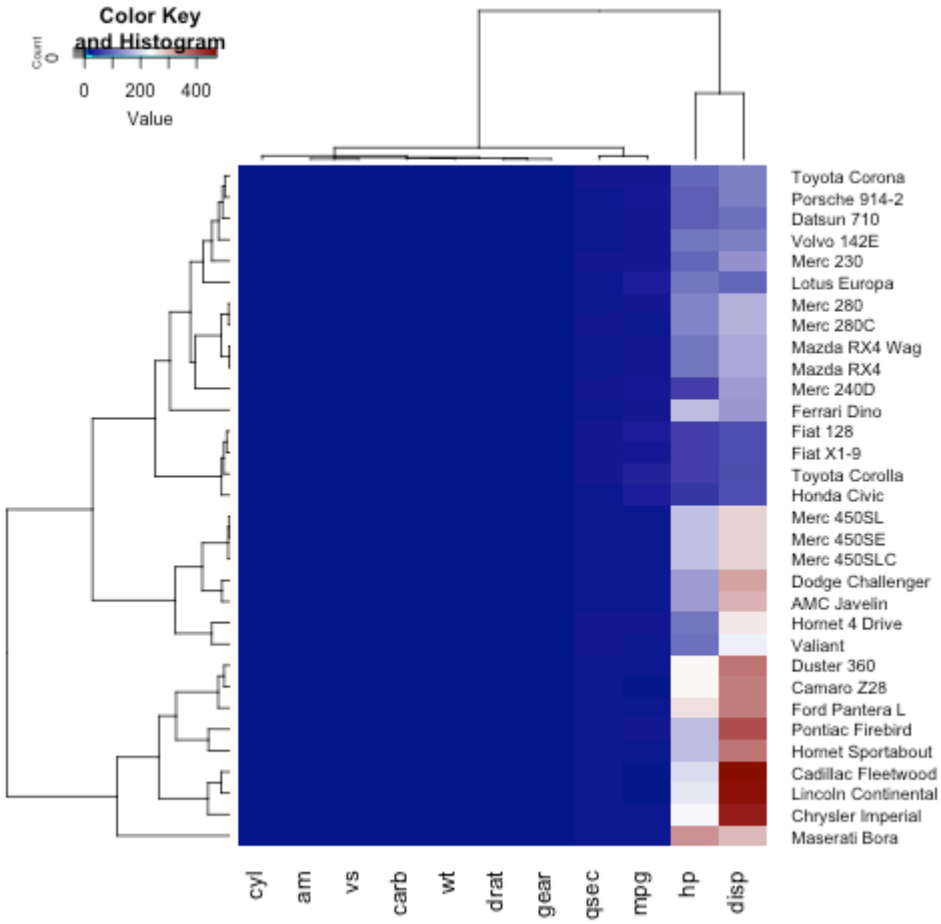


Как вы можете заметить, метки на оси y (названия автомобилей) не соответствуют рисунку. Чтобы исправить это, пользователь может настроить параметр `margins` :

```
heatmap.2(x, trace="none", key=TRUE,col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8))
```

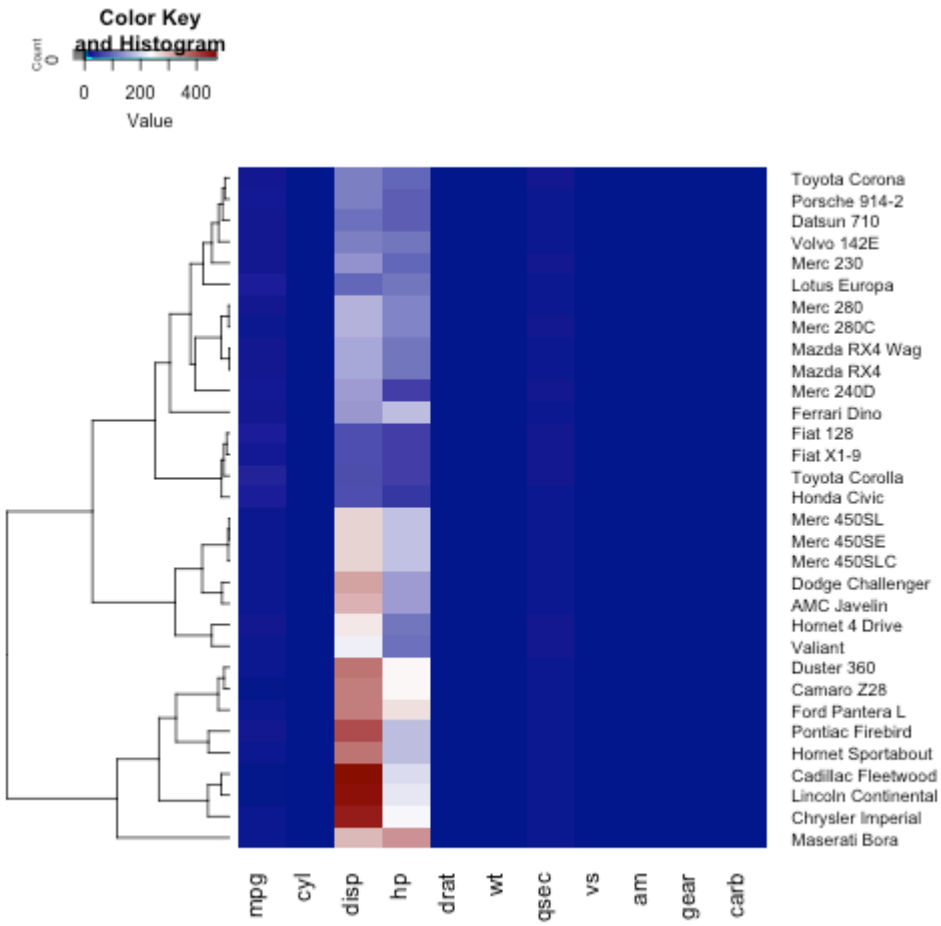


Кроме того, мы можем изменять размеры каждого раздела нашей тепловой карты (ключевая гистограмма, дендограммы и сама тепловая карта), настраивая `lhei` и `lwid` :



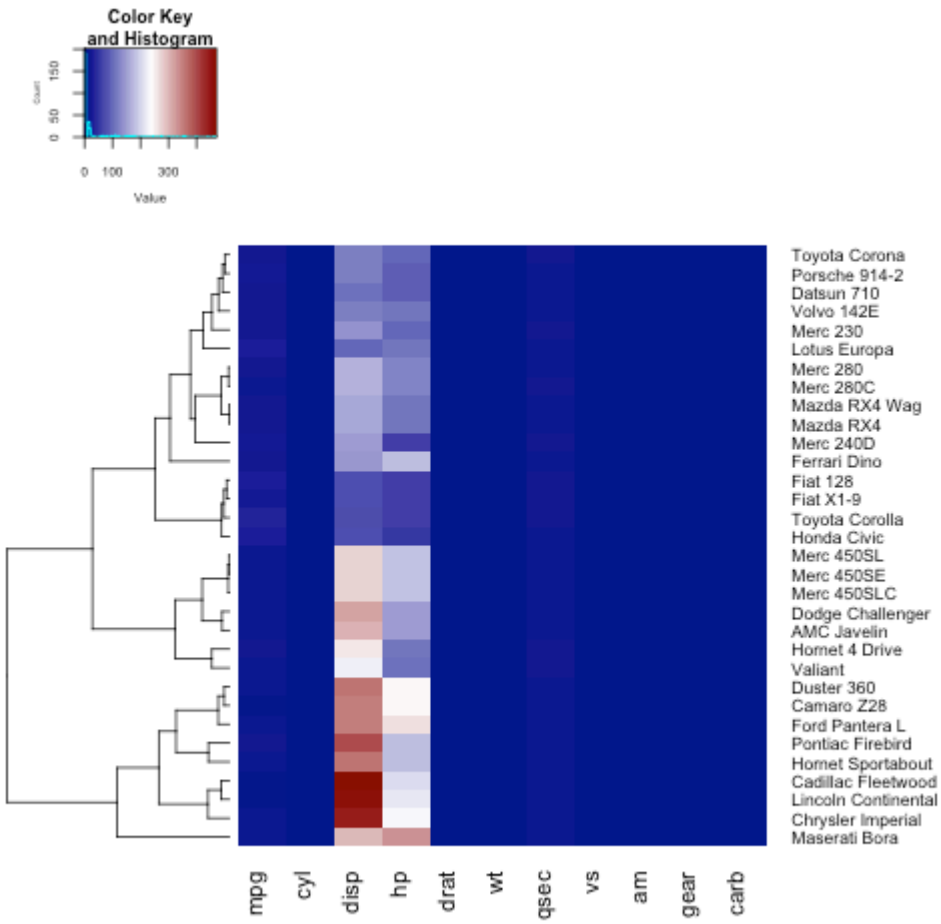
Если мы хотим показать только дендограмму строки (или столбца), нам нужно установить `Colv=FALSE` (или `Rowv=FALSE`) и отрегулировать параметр `dendogram` :

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(3,15))
```



Для изменения размера шрифта заголовка легенды, меток и оси пользователь должен установить `cex.main, cex.lab, cex.axis` в `par` списке:

```
par(cex.main=1, cex.lab=0.7, cex.axis=0.7)
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(5,15))
```



Прочитайте тепловая карта и тепловая карта.2 онлайн: https://riptutorial.com/ru/r/topic/4814/тепловая-карта-и-тепловая-карта-2

глава 117: Ускорение жестко-векторизованного кода

Examples

Ускорение жесткой векторизации для циклов с Rcpp

Возьмем следующий цикл for-loopize, который создает вектор длины `len` где указан первый элемент (`first`), и каждый элемент `x_i` равен `cos(x_{i-1} + 1)` :

```
repeatedCosPlusOne <- function(first, len) {
  x <- numeric(len)
  x[1] <- first
  for (i in 2:len) {
    x[i] <- cos(x[i-1] + 1)
  }
  return(x)
}
```

Этот код включает цикл for с быстрой операцией (`cos(x[i-1]+1)`), которые часто выигрывают от векторизации. Однако нетривиально векторизовать эту операцию с базой R, так как R не имеет «кумулятивного косинуса x + 1».

Одним из возможных подходов к ускорению этой функции было бы реализовать ее на C ++, используя пакет Rcpp:

```
library(Rcpp)
cppFunction("NumericVector repeatedCosPlusOneRcpp(double first, int len) {
  NumericVector x(len);
  x[0] = first;
  for (int i=1; i < len; ++i) {
    x[i] = cos(x[i-1]+1);
  }
  return x;
}")
```

Это часто обеспечивает значительное ускорение для больших вычислений, при этом получая точные результаты:

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneRcpp(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#    user  system elapsed
#   1.274   0.015   1.310
system.time(repeatedCosPlusOneRcpp(1, 1e6))
#    user  system elapsed
#   0.028   0.001   0.030
```

В этом случае код Rcpp генерирует вектор длиной 1 миллион за 0,03 секунды вместо 1,31 секунды с базовым R-подходом.

Ускорение жесткой для векторизации для циклов путем составления байтов

Следуя примеру Rcpp в этой записи документации, рассмотрим следующую жесткую функцию для векторизации, которая создает вектор длины `len` где указан первый элемент (`first`), и каждый элемент `x_i` равен `cos(x_{i-1} + 1)` :

```
repeatedCosPlusOne <- function(first, len) {
  x <- numeric(len)
  x[1] <- first
  for (i in 2:len) {
    x[i] <- cos(x[i-1] + 1)
  }
  return(x)
}
```

Один простой подход к ускорению такой функции без перезаписи одной строки кода - это байт, компилирующий код с использованием компилируемого пакета R:

```
library(compiler)
repeatedCosPlusOneCompiled <- cmpfun(repeatedCosPlusOne)
```

Результирующая функция часто будет значительно быстрее, при этом возвращая те же результаты:

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneCompiled(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#    user  system elapsed
#   1.175   0.014   1.201
system.time(repeatedCosPlusOneCompiled(1, 1e6))
#    user  system elapsed
#   0.339   0.002   0.341
```

В этом случае байт-компиляция ускоряла жесткую для векторизации операцию с вектором длиной 1 миллион с 1,20 секунды до 0,34 секунды.

замечание

Сущность `repeatedCosPlusOne` , как совокупное применение одной функции, может быть более прозрачно выражена с помощью `Reduce` :

```
iterFunc <- function(init, n, func) {
  funcs <- replicate(n, func)
  Reduce(function(., f) f(.), funcs, init = init, accumulate = TRUE)
}
repeatedCosPlusOne_vec <- function(first, len) {
```

```
  iterFunc(first, len - 1, function(.) cos(. + 1))
}
```

`repeatedCosPlusOne_vec` может рассматриваться как «векторизация» `repeatedCosPlusOne` . Однако можно ожидать, что он будет *медленнее* в 2 раза:

```
library(microbenchmark)
microbenchmark(
  repeatedCosPlusOne(1, 1e4),
  repeatedCosPlusOne_vec(1, 1e4)
)
#> Unit: milliseconds
#>                              expr       min        lq     mean   median       uq      max
neval cld
#>      repeatedCosPlusOne(1, 10000)  8.349261  9.216724 10.22715 10.23095 11.10817 14.33763
100  a
#>  repeatedCosPlusOne_vec(1, 10000) 14.406291 16.236153 17.55571 17.22295 18.59085 24.37059
100   b
```

Прочитайте Ускорение жестко-векторизованного кода онлайн:
https://riptutorial.com/ru/r/topic/1203/ускорение-жестко-векторизованного-кода

глава 118: Услуги RESTful R

Вступление

OpenCPU использует стандартную упаковку R для разработки, отправки и развертывания веб-приложений.

Examples

Приложения opencpu

Официальный сайт содержит хороший пример приложений:
https://www.opencpu.org/apps.html

Следующий код используется для обслуживания сеанса R:

```
library(opencpu)
opencpu$start(port = 5936)
```

После выполнения этого кода вы можете использовать URL-адреса для доступа к функциям сеанса R. Результатом могут быть XML, html, JSON или некоторые другие определенные форматы.

Например, к предыдущему сеансу R можно получить вызов cURL:

```
#curl uses http post method for -X POST or -d "arg=value"
curl http://localhost:5936/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://localhost:5936/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
```

Вызов является асинхронным, что означает, что сеанс R не блокируется в ожидании завершения вызова (в отличие от блестящего).

Результат вызова сохраняется во временном сеансе, хранящемся в `/ocpu/tmp/`

Пример получения временного сеанса:

```
curl https://public.opencpu.org/ocpu/library/stats/R/rnorm -d n=5
/ocpu/tmp/x009f9e7630/R/.val
/ocpu/tmp/x009f9e7630/stdout
/ocpu/tmp/x009f9e7630/source
/ocpu/tmp/x009f9e7630/console
/ocpu/tmp/x009f9e7630/info
```

`x009f9e7630` - это имя сеанса.

Указание на `/ocpu/tmp/x009f9e7630/R/.val` вернет значение, полученное в результате `rnorm(5)`

, `/ocpu/tmp/x009f9e7630/R/console` вернет содержимое консоли `rnorm(5)` и т. Д.

Прочитайте Услуги RESTful R онлайн: https://riptutorial.com/ru/r/topic/8323/услуги-restful-r

глава 119: Установить операции

замечания

Набор содержит только одну копию каждого отдельного элемента. В отличие от некоторых других языков программирования, база R не имеет специального типа данных для наборов. Вместо этого R обрабатывает вектор, подобный множеству, беря только его отдельные элементы. Это относится к операторам множества, `setdiff` , `intersect` , `union` , `setequal` и `%in%` . При `v %in% S` только `S` рассматривается как множество, а не вектор `v` .

Для истинно заданного типа данных в R пакет Rcpp предоставляет некоторые параметры .

Examples

Установить операторы для пар векторов

Сравнение наборов

В R вектор может содержать дублированные элементы:

```
v = "A"
w = c("A", "A")
```

Однако набор содержит только одну копию каждого элемента. R обрабатывает вектор, подобный множеству, беря только его отдельные элементы, поэтому два вектора выше считаются одинаковыми:

```
setequal(v, w)
# TRUE
```

Комбинированные наборы

Основные функции имеют естественные имена:

```
x = c(1, 2, 3)
y = c(2, 4)

union(x, y)
# 1 2 3 4

intersect(x, y)
# 2
```

```
setdiff(x, y)
# 1 3
```

Они все документировано на той же странице, `?union` .

Установить членство для векторов

Оператор `%in%` сравнивает вектор с множеством.

```
v = "A"
w = c("A", "A")

w %in% v
# TRUE TRUE

v %in% w
# TRUE
```

Каждый элемент слева обрабатывается индивидуально и проверяется на принадлежность к набору, связанному с вектором справа (состоящим из всех его отдельных элементов).

В отличие от тестов равенства, `%in%` всегда возвращает `TRUE` или `FALSE` :

```
c(1, NA) %in% c(1, 2, 3, 4)
# TRUE FALSE
```

Документация находится в `?`%in%` .

Декартовы или «кросс» произведения векторов

Чтобы найти каждый вектор вида (x, y), где x выведен из вектора X и y из Y, мы используем `expand.grid` :

```
X = c(1, 1, 2)
Y = c(4, 5)

expand.grid(X, Y)

#   Var1 Var2
# 1    1    4
# 2    1    4
# 3    2    4
# 4    1    5
# 5    1    5
# 6    2    5
```

Результатом является data.frame с одним столбцом для каждого переданного ему вектора. Часто мы хотим взять декартово произведение множеств, а не расширять «сетку» векторов. Мы можем использовать `unique` , `lapply` и `do.call` :

```
m = do.call(expand.grid, lapply(list(X, Y), unique))

#   Var1 Var2
# 1    1    4
# 2    2    4
# 3    1    5
# 4    2    5
```

Применение функций к комбинациям

Если вы хотите применить функцию к каждой результирующей комбинации `f(x,y)` , ее можно добавить в качестве другого столбца:

```
m$p = with(m, Var1*Var2)
#   Var1 Var2  p
# 1    1    4  4
# 2    2    4  8
# 3    1    5  5
# 4    2    5 10
```

Этот подход работает на столько векторов, сколько нам нужно, но в частном случае из двух иногда лучше иметь результат в матрице, которая может быть достигнута с помощью `outer` :

```
uX = unique(X)
uY = unique(Y)

outer(setNames(uX, uX), setNames(uY, uY), `*`)

#   4  5
# 1 4  5
# 2 8 10
```

Соответствующие понятия и инструменты см. В теме комбинаторики.

Сделайте уникальные / капли дубликатов / выберите отдельные элементы из вектора

`unique` капли дублируют, так что каждый элемент в результате уникален (появляется только один раз):

```
x = c(2, 1, 1, 2, 1)

unique(x)
# 2 1
```

Значения возвращаются в том порядке, в котором они впервые появились.

`duplicated` теги каждый дублированный элемент:

```
duplicated(x)
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` - это быстрый способ проверить, содержит ли вектор любые дубликаты.

Интервалы измерительных наборов / диаграммы Венна для векторов

Чтобы подсчитать, сколько элементов двух наборов перекрывается, можно написать настраиваемую функцию:

```
xtab_set <- function(A, B){
    both    <-  union(A, B)
    inA     <-  both %in% A
    inB     <-  both %in% B
    return(table(inA, inB))
}

A = 1:20
B = 10:30

xtab_set(A, B)

#        inB
# inA     FALSE TRUE
#   FALSE     0   10
#   TRUE      9   11
```

Диаграмма Венна, предлагаемая различными пакетами, может быть использована для визуализации совпадений между несколькими наборами.

Прочитайте Установить операции онлайн: https://riptutorial.com/ru/r/topic/1383/установить-операции

глава 120: Установка пакетов

Синтаксис

- install.packages (pkgs, lib, repos, method, destdir, зависимости, ...)

параметры

параметр	подробности
pkgs	символьный вектор имен пакетов. Если `repos = NULL` , символьный вектор путей файла.
Lib	символьный вектор, дающий каталоги библиотек, где устанавливать пакеты.
РЕПО	символьный вектор, базовый URL (-ы) используемых репозиториев, может быть `NULL` для установки из локальных файлов
метод	способ загрузки
DESTDIR	каталог, в котором хранятся загруженные пакеты
зависимости	логическое указание, следует ли устанавливать удаленные пакеты, которые эти пакеты зависят от / link to / import / suggest (и так далее рекурсивно). Не используется, если `repos = NULL` .
...	Аргументы, которые должны быть переданы в 'download.file' или в функции для двоичной установки в OS X и Windows.

замечания

Связанные документы

- Проверка пакетов

Examples

Загрузка и установка пакетов из репозиториев

Пакеты представляют собой коллекции R-функций, данных и скомпилированного кода в

четко определенном формате . Публичные (и частные) репозитории используются для размещения коллекций пакетов R. Самая большая коллекция R-пакетов доступна в CRAN.

Использование CRAN

Пакет можно установить из CRAN, используя следующий код:

```
install.packages("dplyr")
```

Где `"dplyr"` называется символьным вектором.

Несколько пакетов можно установить за один раз с помощью функции `c()` и передачи серии символов символов имен пакетов:

```
install.packages(c("dplyr", "tidyr", "ggplot2"))
```

В некоторых случаях `install.packages` может запрашивать зеркало CRAN или сбой, в зависимости от значения `getOption("repos")` . Чтобы предотвратить это, укажите зеркало CRAN в качестве аргумента `repos` :

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

Используя аргумент `repos` вы также можете установить его из других репозиториев. Для получения полной информации обо всех доступных параметрах запустите `?install.packages` .

В большинстве пакетов требуются функции, которые были реализованы в других пакетах (например, в `data.table`). Чтобы установить пакет (или несколько пакетов) со всеми пакетами, которые используются этим пакетом, для `dependencies` аргументов следует установить значение `TRUE`):

```
install.packages("data.table", dependencies = TRUE)
```

Использование биокондуктора

Bioconductor размещает значительную коллекцию пакетов, связанных с биоинформатикой. Они обеспечивают собственное управление пакетами, основанное на функции `biocLite` :

```
    ## Try http:// if https:// URLs are not supported
    source("https://bioconductor.org/biocLite.R")
    biocLite()
```

По умолчанию это устанавливает подмножество пакетов, которые предоставляют

наиболее часто используемые функции. Конкретные пакеты могут быть установлены путем передачи вектора имен пакетов. Например, для установки `RImmPort` из Bioconductor:

```
    source("https://bioconductor.org/biocLite.R")
    biocLite("RImmPort")
```

Установить пакет из локального источника

Чтобы установить пакет из локального исходного файла:

```
install.packages(path_to_source, repos = NULL, type="source")

install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

Здесь `path_to_source` является абсолютным путем для локального исходного файла.

Другая команда, открывающая окно для выбора загруженных исходных файлов zip или tar.gz:

```
install.packages(file.choose(), repos=NULL)
```

Другим возможным способом является использование *RStudio на основе графического интерфейса* :

Шаг 1. Перейдите в **раздел «Инструменты»** .

Шаг 2. Перейдите в **папку Install Packages** .

Шаг 3: В *Install From* установите его как **файл архива пакетов (.zip; .tar.gz)**

Шаг 4: Затем *Browse* найдите файл пакета (скажем crayon_1.3.1.zip) и *через некоторое время (после того, как он показывает **путь пакета и имя файла** на вкладке « Архив архива »)*

Другой способ установки пакета R из локального источника - использовать функцию `install_local()` из пакета devtools.

```
library(devtools)
install_local("~/Downloads/dplyr-master.zip")
```

Установка пакетов из GitHub

Для установки пакетов непосредственно из GitHub используйте пакет `devtools` :

```
library(devtools)
install_github("authorName/repositoryName")
```

Чтобы установить `ggplot2` из github:

```
devtools::install_github("tidyverse/ggplot2")
```

Вышеупомянутая команда установит версию `ggplot2` которая соответствует *основной* ветке. Для установки из другой ветви репозитория используйте аргумент `ref` чтобы указать имя ветки. Например, следующая команда установит `dev_general` ветвь `googleway` пакета.

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

Другой вариант - использовать пакет `ghit` . Он обеспечивает легкую альтернативу для установки пакетов из github:

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

Чтобы установить пакет, который находится в **частном** репозитории в Github, создайте **токен доступа для доступа по** адресу http://www.github.com/settings/tokens/ (см.? Install_github для документации по тому же). Следуй этим шагам:

1. ```
   install.packages(c("curl", "httr"))
   ```

2. ```
   config = httr::config(ssl_verifypeer = FALSE)
   ```

3. ```
    install.packages("RCurl")
    options(RCurlOptions = c(getOption("RCurlOptions"),ssl.verifypeer = FALSE,
   ssl.verifyhost = FALSE ) )
   ```

4. ```
   getOption("RCurlOptions")
   ```

   Вы должны увидеть следующее:

   ```
   ssl.verifypeer ssl.verifyhost

   FALSE          FALSE
   ```

5. ```
   library(httr)
   set_config(config(ssl_verifypeer = 0L))
   ```

   Это предотвращает общую ошибку: «Сертификат peer не может быть аутентифицирован с данными сертификатами CA»

6. Наконец, используйте следующую команду для простой установки пакета

   ```
   install_github("username/package_name",auth_token="abc")
   ```

Альтернативно, установите переменную среды `GITHUB_PAT` , используя
   ```

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

PAT, сгенерированный в Github, отображается только один раз, т. `.Rprofile` Когда он создан изначально, поэтому его разумно сохранить этот токен в `.Rprofile` . Это также полезно, если у организации много частных репозиториев.

Использование диспетчера пакетов CLI - базовое использование pacman

`pacman` - простой менеджер пакетов для R.

`pacman` позволяет пользователю компактно загружать все нужные пакеты, устанавливая все, что отсутствует (и их зависимости), с помощью одной команды `p_load` . `pacman` не требует, чтобы пользователь вводил кавычки вокруг имени пакета. Основное использование:

```
p_load(data.table, dplyr, ggplot2)
```

Единственный пакет требует `library` , `require` , или `install.packages` заявление с этим подходом является `pacman` сам:

```
library(pacman)
p_load(data.table, dplyr, ggplot2)
```

или, что в равной степени справедливо:

```
pacman::p_load(data.table, dplyr, ggplot2)
```

Помимо экономии времени, требуя меньше кода для управления пакетами, `pacman` также облегчает создание воспроизводимого кода, устанавливая любые необходимые пакеты тогда и только тогда, когда они еще не установлены.

Поскольку вы не уверены в том, что `pacman` установлен в библиотеке пользователя, который будет использовать ваш код (или самостоятельно в будущем использовании вашего собственного кода), лучше всего включить условный оператор для установки `pacman` если он еще не установлен нагрузить:

```
if(!(require(pacman)) install.packages("pacman")
pacman::p_load(data.table, dplyr, ggplot2)
```

Установите локальную версию пакета

При работе над разработкой пакета R часто необходимо установить последнюю версию пакета. Это может быть достигнуто путем создания исходного дистрибутива пакета (в командной строке)

```
R CMD build my_package
```

а затем установите его в R. Любые запущенные сеансы R с предыдущей версией загруженного пакета необходимо перезагрузить.

```
unloadNamespace("my_package")
library(my_package)
```

Более удобный подход использует пакет `devtools` для упрощения процесса. В сеансе R с рабочим каталогом, установленным в каталог пакета

```
devtools::install()
```

будет создавать, устанавливать и перезагружать пакет.

Прочитайте Установка пакетов онлайн: https://riptutorial.com/ru/r/topic/1719/установка-пакетов

глава 121: факторы

Синтаксис

1. фактор (x = символ (), уровни, метки = уровни, исключить = NA, упорядочено = is.ordered (x), nmax = NA)
2. Запустить `?factor` Или просмотреть документацию в Интернете.

замечания

Объектом с `factor` класса является вектор с определенным набором характеристик.

1. Он хранится внутри как `integer` вектор.
2. Он поддерживает атрибут `levels` показывает отображение символа значений.
3. Его класс хранится как `factor`

Чтобы проиллюстрировать, создадим вектор 1000 наблюдений из набора цветов.

```
set.seed(1)
Color <- sample(x = c("Red", "Blue", "Green", "Yellow"),
                size = 1000,
                replace = TRUE)
Color <- factor(Color)
```

Мы можем наблюдать каждую из характеристик `Color` перечисленных выше:

```
#* 1. It is stored internally as an `integer` vector
typeof(Color)
```

```
[1] "integer"
```

```
#* 2. It maintains a `levels` attribute the shows the character representation of the values.
#* 3. Its class is stored as `factor`
attributes(Color)
```

```
$levels
[1] "Blue"   "Green"  "Red"    "Yellow"

$class
[1] "factor"
```

Основным преимуществом фактор-объекта является эффективность хранения данных. Целое число требует меньше памяти для хранения, чем символ. Такая эффективность была очень желательна, когда многие компьютеры имели гораздо более ограниченные ресурсы, чем текущие машины (для более подробной истории мотиваций, связанных с

использованием факторов, см. `stringsAsFactors` : Unauthorized Biography). Разницу в использовании памяти можно увидеть даже в нашем объекте `Color` . Как вы можете видеть, для сохранения `Color` в качестве символа требуется примерно в 1,7 раза больше памяти, чем фактор-объект.

```
#* Amount of memory required to store Color as a factor.
object.size(Color)
```

```
4624 bytes
```

```
#* Amount of memory required to store Color as a character
object.size(as.character(Color))
```

```
8232 bytes
```

Отображение целого числа до уровня

Хотя внутреннее вычисление факторов рассматривает объект как целое, желательным представлением для потребления человеком является уровень персонажа. Например,

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
Levels: Blue Green Red Yellow
```

легче для понимания человеком, чем

```
head(as.numeric(Color))
```

```
[1] 1 1 2 4 3 4
```

Примерная иллюстрация того, как R идет о совпадении представления символа с внутренним целочисленным значением:

```
head(levels(Color)[as.numeric(Color)])
```

```
[1] "Blue"   "Blue"   "Green"  "Yellow" "Red"    "Yellow"
```

Сравните эти результаты с

```
head(Color)
```

```
[1] Blue   Blue   Green  Yellow Red    Yellow
```

```
Levels: Blue Green Red Yellow
```

Современное использование факторов

В 2007 году R представил метод хэширования для символов, который уменьшил нагрузку на память векторных векторов (ref: `stringsAsFactors` : Unauthorized Biography). Обратите внимание, что, когда мы определили, что персонажам требуется в 1,7 раза больше места для хранения, чем факторы, которые были рассчитаны в недавней версии R, это означает, что использование символьных векторов памяти было еще более подвержено налогообложению до 2007 года.

Благодаря методу хеширования в современном R и значительно большему ресурсу памяти в современных компьютерах проблема эффективности памяти при хранении символьных значений была сведена к очень малой проблеме. Преобладающая позиция в сообществе R - это предпочтение векторам символов над факторами в большинстве ситуаций. Основными причинами перехода от факторов являются

1. Увеличение неструктурированных и / или слабо управляемых символьных данных
2. Тенденция факторов не вести себя по желанию, когда пользователь забывает, что она имеет дело с фактором, а не с характером

В первом случае нет смысла хранить свободные текстовые или открытые поля ответа в качестве факторов, так как вряд ли будет какой-либо шаблон, который допускает более одного наблюдения за уровень. В качестве альтернативы, если структура данных не контролируется тщательно, можно получить несколько уровней, соответствующих одной и той же категории (например, «синий», «синий» и «ГОЛУБОЙ»). В таких случаях многие предпочитают управлять этими расхождениями как символы перед преобразованием в фактор (если конверсия вообще происходит).

Во втором случае, если пользователь считает, что она работает с символьным вектором, некоторые методы могут не реагировать так, как ожидалось. Это базовое понимание может привести к путанице и разочарованию при попытке отладки сценариев и кодов. Хотя, строго говоря, это может считаться ошибкой пользователя, большинство пользователей с радостью избегают использования факторов и вообще избегают этих ситуаций.

Examples

Базовое создание факторов

Факторы - один из способов представления категориальных переменных в R. Фактор хранится внутри как **вектор целых чисел** . Уникальные элементы предоставленного символа вектора называются *уровнями* фактора. По умолчанию, если уровни не

предоставляются пользователем, тогда R генерирует набор уникальных значений в векторе, сортирует эти значения буквенно-цифровым способом и использует их в качестве уровней.

```
 charvar <- rep(c("n", "c"), each = 3)
 f <- factor(charvar)
 f
 levels(f)

> f
[1] n n n c c c
Levels: c n
> levels(f)
[1] "c" "n"
```

Если вы хотите изменить порядок уровней, то один вариант указать уровни вручную:

```
levels(factor(charvar, levels = c("n","c")))

> levels(factor(charvar, levels = c("n","c")))
[1] "n" "c"
```

Факторы имеют ряд свойств. Например, уровням могут быть присвоены метки:

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt     Newt     Newt     Capybara Capybara Capybara
Levels: Newt Capybara
```

Другое свойство, которое можно назначить, заключается в том, упорядочен ли фактор:

```
> Weekdays <- factor(c("Monday", "Wednesday", "Thursday", "Tuesday", "Friday", "Sunday",
"Saturday"))
> Weekdays
[1] Monday    Wednesday Thursday  Tuesday   Friday    Sunday    Saturday
Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
> Weekdays <- factor(Weekdays, levels=c("Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"), ordered=TRUE)
> Weekdays
[1] Monday    Wednesday Thursday  Tuesday   Friday    Sunday    Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
```

Когда уровень фактора больше не используется, вы можете отказаться от него с помощью функции `droplevels()` :

```
> Weekend <- subset(Weekdays, Weekdays == "Saturday" |  Weekdays == "Sunday")
> Weekend
[1] Sunday   Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
> Weekend <- droplevels(Weekend)
> Weekend
[1] Sunday   Saturday
Levels: Saturday < Sunday
```

Консолидация уровней факторов со списком

Бывают моменты, когда желательно объединить уровни факторов в меньшее количество групп, возможно, из-за редких данных в одной из категорий. Это может также произойти, когда у вас разные варианты написания или капитализация названий категорий. Рассмотрим в качестве примера фактор

```
set.seed(1)
colorful <- sample(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                   size = 20,
                   replace = TRUE)
colorful <- factor(colorful)
```

Так как R чувствительна к регистру, таблица частот этого вектора будет выглядеть так, как показано ниже.

```
table(colorful)
```

```
colorful
blue  Blue  BLUE green  gren   red   Red   RED
   3     1     4     2     4     1     3     2
```

Однако эта таблица не отражает истинное распределение данных, и категории могут быть эффективно уменьшены до трех типов: синий, зеленый и красный. Приведены три примера. Первый иллюстрирует то, что кажется очевидным решением, но на самом деле не даст решения. Второе дает рабочее решение, но оно многословно и вычислительно дорого. Третий не является очевидным решением, но относительно компактен и эффективно вычисляется.

Консолидация уровней с использованием `factor` (`factor_approach`)

```
factor(as.character(colorful),
       levels = c("blue", "Blue", "BLUE", "green", "gren", "red", "Red", "RED"),
       labels = c("Blue", "Blue", "Blue", "Green", "Green", "Red", "Red", "Red"))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
Levels: Blue Blue Blue Green Green Red Red Red
Warning message:
In `levels<-`(`*tmp*`, value = if (nl == nL) as.character(labels) else
paste0(labels,  :
  duplicated levels in factors are deprecated
```

Обратите внимание, что существуют дублированные уровни. У нас все еще есть три категории для «Синего», что не завершает нашу задачу консолидации уровней. Кроме

того, существует предупреждение о том, что дублированные уровни устарели, что означает, что этот код может вызвать ошибку в будущем.

Консолидация уровней с использованием `ifelse` (`ifelse_approach`)

```
factor(ifelse(colorful %in% c("blue", "Blue", "BLUE"),
       "Blue",
       ifelse(colorful %in% c("green", "gren"),
              "Green",
              "Red")))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
Levels: Blue Green Red
```

Этот код генерирует желаемый результат, но требует использования вложенных операторов `ifelse` . Хотя в этом подходе нет ничего плохого, управление вложенными `ifelse` может быть утомительной задачей и должно быть сделано тщательно.

Консолидация уровней факторов со списком (`list_approach`)

Менее очевидным способом консолидации уровней является использование списка, в котором имя каждого элемента является желаемым именем категории, а элемент является символьным вектором уровней в коэффициенте, который должен отображаться в нужной категории. Это имеет дополнительное преимущество, непосредственно работая над атрибутом `levels` фактора без необходимости назначать новые объекты.

```
levels(colorful) <-
     list("Blue" = c("blue", "Blue", "BLUE"),
          "Green" = c("green", "gren"),
          "Red" = c("red", "Red", "RED"))
```

```
 [1] Green Blue  Red   Red   Blue  Red   Red   Red   Blue  Red   Green Green Green
Blue  Red   Green
[17] Red   Green Green Red
Levels: Blue Green Red
```

Бенчмаркинг каждого подхода

Время, необходимое для выполнения каждого из этих подходов, приводится ниже. (Для космоса код для создания этого резюме не показан)

```
Unit: microseconds
          expr     min      lq      mean   median      uq     max neval cld
```

```
        factor  78.725  83.256  93.26023  87.5030  97.131 218.899   100  b
        ifelse 104.494 107.609 123.53793 113.4145 128.281 254.580   100   c
 list_approach  49.557  52.955  60.50756  54.9370  65.132 138.193   100 a
```

Подход к списку работает примерно в два раза быстрее, чем подход `ifelse` . Однако, за исключением очень больших объемов данных, различия во времени выполнения, вероятно, будут измеряться либо в микросекундах, либо в миллисекундах. При таких небольших временных различиях эффективность не должна определять решение о том, какой подход использовать. Вместо этого используйте знакомый и удобный подход, который вы и ваши сотрудники поймете в будущем обзоре.

факторы

Факторы - это один из способов представления категориальных переменных в R. Для вектора `x` , значения которого могут быть преобразованы в символы с использованием `as.character()` , аргументы по умолчанию для `factor()` и `as.factor()` присваивают целое число каждому отдельному элементу из вектор, а также атрибут уровня и атрибут метки. Уровни - это значения, которые могут принимать значения `x` и метки могут быть либо заданным элементом, либо определенным пользователем.

Например, как работают факторы, мы создадим фактор с атрибутами по умолчанию, а затем с пользовательскими уровнями, а затем с пользовательскими уровнями и метками.

```
# standard
factor(c(1,1,2,2,3,3))
[1] 1 1 2 2 3 3
Levels: 1 2 3
```

Экземпляры могут возникать, когда пользователь знает количество возможных значений, которые может принимать фактор, больше текущих значений в векторе. Для этого мы сами присваиваем уровни в `factor()` .

```
factor(c(1,1,2,2,3,3),
         levels = c(1,2,3,4,5))
[1] 1 1 2 2 3 3
Levels: 1 2 3 4 5
```

В целях стиля пользователь может пожелать присвоить метки каждому уровню. По умолчанию метки являются символьным представлением уровней. Здесь мы назначаем метки для каждого из возможных уровней фактора.

```
factor(c(1,1,2,2,3,3),
       levels = c(1,2,3,4,5),
       labels = c("Fox","Dog","Cow","Brick","Dolphin"))
[1] Fox Fox Dog Dog Cow Cow
Levels: Fox Dog Cow Brick Dolphin
```

Обычно коэффициенты можно сравнивать только с помощью `==` и `!=` И если коэффициенты имеют одинаковые уровни. Следующее сравнение факторов не удается, хотя они выглядят одинаковыми, поскольку факторы имеют разные уровни факторов.

```
factor(c(1,1,2,2,3,3),levels = c(1,2,3)) == factor(c(1,1,2,2,3,3),levels = c(1,2,3,4,5))
Error in Ops.factor(factor(c(1, 1, 2, 2, 3, 3), levels = c(1, 2, 3)),  :
  level sets of factors are different
```

Это имеет смысл, поскольку дополнительные уровни в RHS означают, что R не имеет достаточной информации о каждом факторе, чтобы сравнить их значимым образом.

Операторы `<` , `<=` , `>` и `>=` применимы только для упорядоченных множителей. Они могут представлять собой категориальные значения, которые все еще имеют линейный порядок. Упорядоченный множитель может быть создан путем предоставления `ordered = TRUE` аргумента `ordered = TRUE factor` функции или просто использования `ordered` функции.

```
x <- factor(1:3, labels = c('low', 'medium', 'high'), ordered = TRUE)
print(x)
[1] low    medium high
Levels: low < medium < high

y <- ordered(3:1, labels = c('low', 'medium', 'high'))
print(y)
[1] high   medium low
Levels: low < medium < high

x < y
[1]  TRUE FALSE FALSE
```

Для получения дополнительной информации см. Документацию Factor .

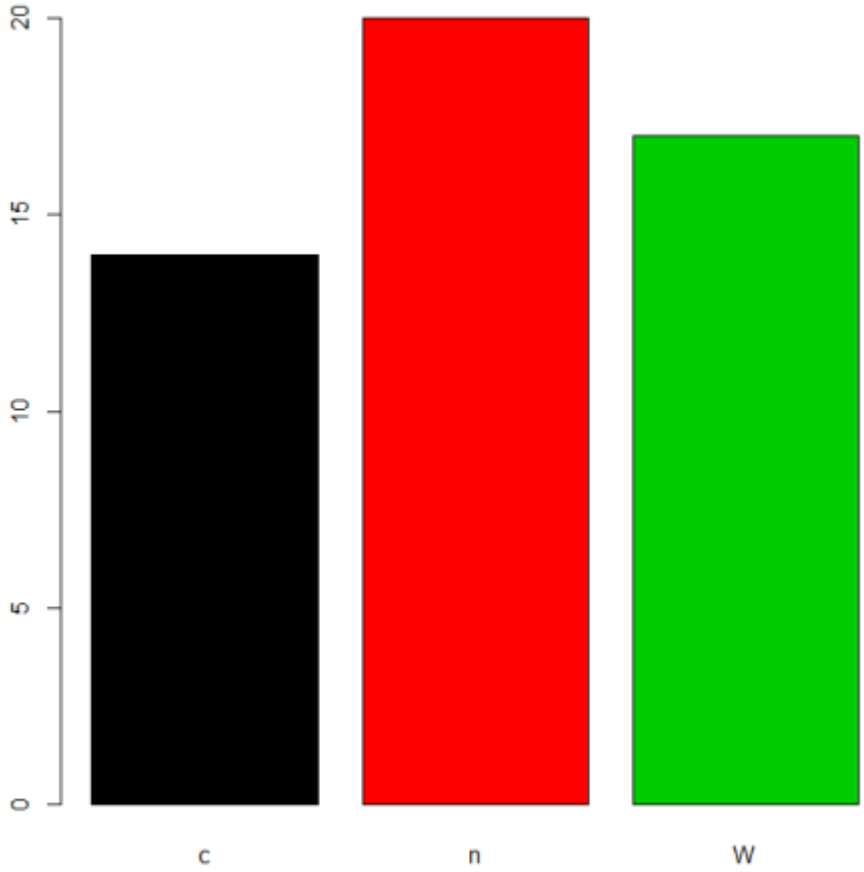
Изменение и изменение порядка

Когда факторы создаются со значениями по умолчанию, `levels` формируются с помощью `as.character` применяемого к входам, и упорядочиваются по алфавиту.

```
charvar <- rep(c("W", "n", "c"), times=c(17,20,14))
f <- factor(charvar)
levels(f)
# [1] "c" "n" "W"
```

В некоторых ситуациях обработка по умолчанию порядка `levels` (буквенно-лексический порядок) будет приемлемой. Например, если кто-то хочет `plot` частоту, это будет результат:

```
plot(f,col=1:length(levels(f)))
```



Но если нам нужен другой порядок `levels` , мы должны указать это в параметрах `levels` или `labels` (учитывая, что значение «порядок» здесь отличается от *упорядоченных* факторов, см. Ниже). Существует много альтернатив для выполнения этой задачи в зависимости от ситуации.

1. Пересмотреть фактор

Когда это возможно, мы можем воссоздать коэффициент, используя параметр `levels` с нужным нам порядком.

```
ff <- factor(charvar, levels = c("n", "W", "c"))
levels(ff)
# [1] "n" "W" "c"

gg <- factor(charvar, levels = c("W", "c", "n"))
levels(gg)
# [1] "W" "c" "n"
```

Когда уровни ввода отличаются от желаемых уровней выходного сигнала, мы используем параметр `labels` который заставляет параметр `levels` становиться «фильтром» для допустимых входных значений, но оставляет конечные значения «уровней» для вектора факторов как аргумент `labels` :

```
fm <- factor(as.numeric(f),levels = c(2,3,1),
             labels = c("nn", "WW", "cc"))
levels(fm)
# [1] "nn" "WW" "cc"

fm <- factor(LETTERS[1:6], levels = LETTERS[1:4],  # only 'A'-'D' as input
                 labels = letters[1:4])            # but assigned to 'a'-'d'
fm
#[1] a    b    c    d    <NA> <NA>
#Levels: a b c d
```

2. Используйте функцию `relevel`

Когда есть один конкретный `level` который должен быть первым, мы можем использовать `relevel` . Это происходит, например, в контексте статистического анализа, когда `base` категория необходима для проверки гипотезы.

```
g<-relevel(f, "n") # moves n to be the first level
levels(g)
# [1] "n" "c" "W"
```

Как можно проверить, `f` и `g` одинаковы

```
all.equal(f, g)
# [1] "Attributes: < Component “levels”: 2 string mismatches >"
all.equal(f, g, check.attributes = F)
# [1] TRUE
```

3. Переупорядочивающие факторы

Бывают случаи, когда нам необходимо `reorder levels` на основе числа, частичного результата, вычисленной статистики или предыдущих вычислений. Давайте переупорядочиваем, основываясь на **частотах** `levels`

```
table(g)
# g
#  n  c  W
# 20 14 17
```

Функция `reorder` является общей (см. `help(reorder)`), но в этом контексте необходимо: `x` , в данном случае фактор; `X` , числовое значение той же длины, что и `x` ; и `FUN` - функция, которая должна применяться к `X` и вычисляется по уровню `x` , который определяет порядок `levels` , по умолчанию увеличивается. Результат - тот же самый фактор, что и его уровни переупорядочены.

```
g.ord <- reorder(g,rep(1,length(g)), FUN=sum) #increasing
levels(g.ord)
# [1] "c" "W" "n"
```

Чтобы получить убывающий порядок, рассмотрим отрицательные значения (`-1`)

```
g.ord.d <- reorder(g,rep(-1,length(g)), FUN=sum)
levels(g.ord.d)
# [1] "n" "W" "c"
```

Опять фактор тот же, что и другие.

```
data.frame(f,g,g.ord,g.ord.d)[seq(1,length(g),by=5),] #just same lines
#    f g g.ord g.ord.d
# 1  W W     W       W
# 6  W W     W       W
# 11 W W     W       W
# 16 W W     W       W
# 21 n n     n       n
# 26 n n     n       n
# 31 n n     n       n
# 36 n n     n       n
# 41 c c     c       c
# 46 c c     c       c
# 51 c c     c       c
```

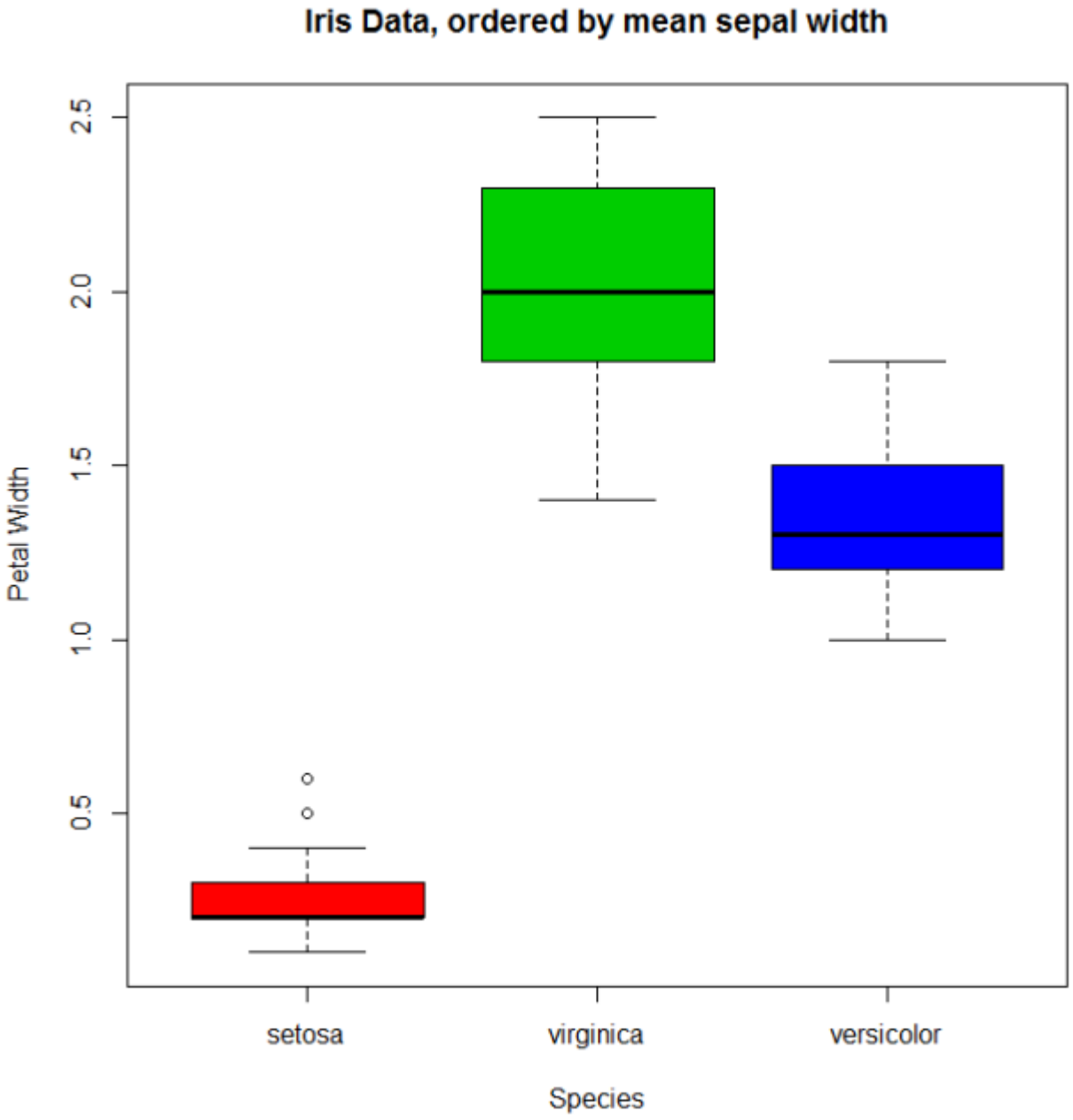
Когда есть **количественная переменная,** связанная с факторной переменной, мы могли бы использовать другие функции для изменения порядка `levels` . Давайте возьмем данные `iris` (`help("iris")` для получения дополнительной информации), для переупорядочения коэффициента `Species` , используя его средний `Sepal.Width` .

```
miris <- iris  #help("iris") # copy the data
with(miris, tapply(Sepal.Width,Species,mean))
#    setosa versicolor  virginica
#     3.428      2.770      2.974

miris$Species.o<-with(miris,reorder(Species,-Sepal.Width))
levels(miris$Species.o)
# [1] "setosa"     "virginica"  "versicolor"
```

Обычная `boxplot` (скажем: `with(miris, boxplot(Petal.Width~Species)`) покажет especies в этом порядке: *setosa* , *versicolor* и *virginica* . Но используя упорядоченный множитель, мы получаем вид, упорядоченный по его среднему `Sepal.Width` :

```
boxplot(Petal.Width~Species.o, data = miris,
        xlab = "Species", ylab = "Petal Width",
        main = "Iris Data, ordered by mean sepal width", varwidth = TRUE,
        col = 2:4)
```



Кроме того, можно также изменить названия `levels` , объединить их в группы или добавить новые `levels` . Для этого мы используем функцию одноименных `levels` .

```
f1<-f
levels(f1)
# [1] "c" "n" "W"
levels(f1) <- c("upper","upper","CAP") #rename and grouping
levels(f1)
# [1] "upper" "CAP"

f2<-f1
levels(f2) <- c("upper","CAP", "Number") #add Number level, which is empty
levels(f2)
# [1] "upper"  "CAP"    "Number"
f2[length(f2):(length(f2)+5)]<-"Number" # add cases for the new level
table(f2)
# f2
```

```
#  upper    CAP Number
#     33     17      6

f3<-f1
levels(f3) <- list(G1 = "upper", G2 = "CAP", G3 = "Number") # The same using list
levels(f3)
# [1] "G1" "G2" "G3"
f3[length(f3):(length(f3)+6)]<-"G3" ## add cases for the new level
table(f3)
# f3
# G1 G2 G3
# 33 17  7
```

- Упорядоченные факторы

Наконец, мы знаем, что `ordered` факторы отличаются от `factors` , первый из которых используется для представления *порядковых данных* , а второй - для работы с *номинальными данными* . Во-первых, не имеет смысла изменять порядок `levels` для упорядоченных факторов, но мы можем изменить его `labels` .

```
ordvar<-rep(c("Low", "Medium", "High"), times=c(7,2,4))

of<-ordered(ordvar,levels=c("Low", "Medium", "High"))
levels(of)
# [1] "Low"    "Medium" "High"

of1<-of
levels(of1)<- c("LOW", "MEDIUM", "HIGH")
levels(of1)
# [1] "LOW"    "MEDIUM" "HIGH"
is.ordered(of1)
# [1] TRUE
of1
# [1] LOW    LOW    LOW    LOW    LOW    LOW    LOW    MEDIUM MEDIUM HIGH   HIGH   HIGH   HIGH

# Levels: LOW < MEDIUM < HIGH
```

Восстановление факторов от нуля

проблема

Факторы используются для представления переменных, которые принимают значения из набора категорий, известных как Уровни в R. Например, некоторый эксперимент может характеризоваться уровнем энергии батареи с четырьмя уровнями: пустым, низким, нормальным и полным. Затем для 5 различных участков отбора проб эти уровни можно было бы определить в этих терминах следующим образом:

> **полный** , **полный** , **нормальный** , **пустой** , **низкий**

Как правило, в базах данных или других источниках информации обработка этих данных осуществляется с помощью произвольных целочисленных индексов, связанных с категориями или уровнями. Если мы предположим, что для данного примера мы будем

присваивать индексы следующим образом: 1 = пустое, 2 = низкое, 3 = нормальное, 4 = полное, тогда 5 выборок могут быть закодированы как:

4 , **4** , **3** , **1** , **2**

Может случиться так, что из вашего источника информации, например базы данных, у вас есть только кодированный список целых чисел и каталог, связывающий каждое целое число с каждым ключевым словом. Как можно восстановить фактор R из этой информации?

Решение

Мы будем моделировать вектор из 20 целых чисел, который представляет образцы, каждый из которых может иметь одно из четырех значений:

```
set.seed(18)
ii <- sample(1:4, 20, replace=T)
ii
```

[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1

Первый шаг - сделать из предыдущей последовательности коэффициент, в котором уровни или категории - это точно числа от 1 до 4.

```
fii <- factor(ii, levels=1:4) # it is necessary to indicate the numeric levels
fii
```

[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
Уровни: 1 2 3 4

Теперь просто, вы должны *одеть* фактор, уже созданный с помощью индексных тегов:

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

[1] полный нормальный полный пустой пустой нормальный низкий нормальный низкий пустой
[11] нормальный полный пустой низкий полный пустой нормальный пустой полный пустой
Уровни: пустой низкий нормальный полный

Прочитайте факторы онлайн: https://riptutorial.com/ru/r/topic/1104/факторы

глава 122: формула

Examples

Основы формулы

Статистические функции в R сильно используют так называемую формулу формулы Уилкинсона-Роджерса [1] .

При выполнении функций модели, таких как `lm` для линейных регрессий , им нужна `formula` . Эта `formula` определяет, какие коэффициенты регрессии должны быть оценены.

```
my_formula1 <- formula(mpg ~ wt)
class(my_formula1)
# gives "formula"

mod1 <- lm(my_formula1, data = mtcars)
coef(mod1)
# gives (Intercept)          wt
#         37.285126   -5.344472
```

В левой части `~` (LHS) задается зависимая переменная, а правая часть (RHS) содержит независимые переменные. Технически вызов `formula` выше избыточен, потому что тильд-оператор является функцией infix, которая возвращает объект с классом формул:

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

Преимущество функции `formula` над `~` заключается в том, что она также позволяет указать среду для оценки:

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

В этом случае выход показывает, что коэффициент регрессии для `wt` оценивается, а также (по умолчанию) параметр перехвата. Перехват может быть исключен / вынужден быть равным 0, включая `0` или `-1` в `formula` :

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

Взаимодействие между переменными `a` и `b` может быть добавлено путем включения `a:b` в `formula` :

```
 coef(lm(mpg ~ wt:vs, data = mtcars))
```

Поскольку (с статистической точки зрения, вообще-то целесообразно) не иметь взаимодействия в модели без основных эффектов, наивный подход заключался бы в том, чтобы расширить `formula` до `a + b + a:b` . Это работает, но может быть упрощено путем записи `a*b` , где оператор `*` указывает пересечение факторов (когда между двумя столбцами факторов) или умножение, когда один или оба столбца являются «числовыми»:

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

Использование нотации `*` расширяет термин для включения всех эффектов нижнего порядка, таких, что:

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

даст в дополнение к перехвату 7 коэффициентов регрессии. Один для трехстороннего взаимодействия, три для двухсторонних взаимодействий и три для основных эффектов.

Если вы хотите, например, исключить трехстороннее взаимодействие, но сохранить все двусторонние взаимодействия, то есть два сокращения. Во-первых, используя `-` мы можем вычесть любой конкретный термин:

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

Или мы можем использовать обозначение `^` чтобы указать, какой уровень взаимодействия нам требуется:

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

Эти две спецификации формул должны создавать одну и ту же модельную матрицу.

И, наконец, `.` является сокращением использования всех доступных переменных в качестве основных эффектов. В этом случае аргумент `data` используется для получения доступных переменных (которые не относятся к LHS). Следовательно:

```
coef(lm(mpg ~ ., data = mtcars))
```

дает коэффициенты для перехвата и 10 независимых переменных. Эта нотация часто используется в пакетах машинного обучения, где вы хотите использовать все переменные для прогнозирования или классификации. Обратите внимание, что значение `.` зависит от контекста (см., например, « `?update.formula` для другого значения).

1. Г. Н. Уилкинсон и К. Роджерс. *Журнал Королевского статистического общества. Серия С (прикладная статистика)* Vol. 22, № 3 (1973), стр. 392-399

Создание условий линейного, квадратичного и второго порядка

взаимодействия

`y ~ .` : Здесь `.` интерпретируется как все переменные, кроме `y` в фрейме данных, используемом при подгонке модели. Это эквивалентно линейным комбинациям предикторных переменных. Например, `y ~ var1 + var2 + var3+...+var15`

`y ~ . ^ 2` даст все линейные (основные эффекты) и условия взаимодействия второго порядка переменных в кадре данных. Это эквивалентно `y ~ var1 + var2 + ...+var15 + var1:var2 + var1:var3 + var1:var4...and so on`

`y ~ var1 + var2 + ...+var15 + I(var1^2) + I(var2^2) + I(var3^2)...+I(var15^2)` : Здесь `I(var^2)` указывает квадратичный многочлен одной переменной в кадре данных.

`y ~ poly(var1, degree = 2) + poly(var2, degree = 2)+...poly(var15, degree = 2)`

или же

`y ~ poly(var1, var2, var3, ....var15, degree = 2)` будет эквивалентно приведенному выше выражению.

`poly(var1, degree = 2)` эквивалентно `var1 + I(var1^2)` .

Чтобы получить кубические полиномы, используйте `degree = 3` в `poly()` .

Существует предостережение в использовании `poly` против `I(var, 2)` , которое после установки модели, каждый из которых будет производить разные коэффициенты, но установленные значения эквивалентны, поскольку они представляют собой различные параметризации одной и той же модели. Рекомендуется использовать `I(var, 2)` над `poly()` чтобы избежать суммарного эффекта, наблюдаемого в `poly()` .

Таким образом, чтобы получить термины взаимодействия по линейному, квадратичному и второму порядкам, вы получите выражение типа

`y ~ .^2 + I(var1^2) + I(var2^2)+...I(var15^2)`

Демо для четырех переменных:

```
old <- reformulate( 'y ~ x1+x2+x3+x4' )
new <- reformulate( " y ~ .^2 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) ")
tmp <- .Call(stats:::C_updateform, old, new)
terms.formula(tmp, simplify = TRUE )

# ~y ~ x1 + x2 + x3 + x4 + I(x1^2) + I(x2^2) + I(x3^2) + I(x4^2) +
#   x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 + x3:x4
# attr(,"variables")
# list(~y, x1, x2, x3, x4, I(x1^2), I(x2^2), I(x3^2), I(x4^2))
# attr(,"factors")
#         x1 x2 x3 x4 I(x1^2) I(x2^2) I(x3^2) I(x4^2) x1:x2 x1:x3 x1:x4 x2:x3 x2:x4 x3:x4
# ~y       0  0  0  0       0       0       0       0     0     0     0     0     0     0
# x1       1  0  0  0       0       0       0       0     1     1     1     0     0     0
# x2       0  1  0  0       0       0       0       0     1     0     0     1     1     0
```

```
# x3        0  0  1  0       0       0       0       0     0     1     0     1     0     1
# x4        0  0  0  1       0       0       0       0     0     0     1     0     1     1
# I(x1^2)   0  0  0  0       1       0       0       0     0     0     0     0     0     0
# I(x2^2)   0  0  0  0       0       1       0       0     0     0     0     0     0     0
# I(x3^2)   0  0  0  0       0       0       1       0     0     0     0     0     0     0
# I(x4^2)   0  0  0  0       0       0       0       1     0     0     0     0     0     0
# attr(,"term.labels")
# [1] "x1"       "x2"       "x3"       "x4"       "I(x1^2)" "I(x2^2)" "I(x3^2)" "I(x4^2)"
# [9] "x1:x2"    "x1:x3"    "x1:x4"    "x2:x3"    "x2:x4"    "x3:x4"
# attr(,"order")
# [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2
# attr(,"intercept")
# [1] 1
# attr(,"response")
# [1] 1
# attr(,".Environment")
# <environment: R_GlobalEnv>
```

Прочитайте формула онлайн: https://riptutorial.com/ru/r/topic/1061/формула

глава 123: Функции записи в R

Examples

Именованные функции

R полна функций, это, в конце концов, функциональный язык программирования , но иногда точная функция, которая вам нужна, не предоставляется в базовых ресурсах. Вы могли бы, возможно, установить пакет, содержащий эту функцию, но, может быть, ваши требования настолько специфичны, что никакая заранее подготовленная функция не соответствует законопроекту? Тогда у вас останется возможность сделать свой собственный.

Функция может быть очень простой, до такой степени, что она почти бессмысленна. Это даже не нужно принимать аргумент:

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

Что между фигурными фигурными скобками `{ }` является собственно функцией. Пока вы можете поместить все на одной линии, они не нужны строго, но могут быть полезны для того, чтобы все было организовано.

Функция может быть очень простой, но очень специфичной. Эта функция принимает на входе вектор (`vec` в этом примере) и выводит один и тот же вектор с длиной вектора (6 в этом случае), вычитаемым из каждого из элементов вектора.

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1  0  1  2  3
```

Обратите внимание, что `length()` сама по себе является предустановленной (т.е. *базовой*) функцией. Вы можете, конечно, использовать ранее выполненную самодельную функцию в рамках другой самодельной функции, а также назначать переменные и выполнять другие операции, охватывая несколько строк:

```
vec2 <- (4:7)/2

msdf <- function(x, multiplier=4) {
    mult <- x * multiplier
```

```
    subl <- subtract.length(x)
    data.frame(mult, subl)
}

msdf(vec2, 5)
  mult subl
1 10.0 -2.0
2 12.5 -1.5
3 15.0 -1.0
4 17.5 -0.5
```

`multiplier=4` гарантирует, что `4` является значением по умолчанию для `multiplier` аргументов, если значение не задано при вызове функции `4` это то, что будет использоваться.

Вышеприведенные примеры всех *названных* функций называются просто потому, что им даны имена (`one` , `two` , `subtract.length` и т. Д.),

Анонимные функции

Анонимная функция, как следует из названия, не назначает имя. Это может быть полезно, когда функция является частью более крупной операции, но сама по себе не занимает много места. Одна из наиболее частых вариантов использования анонимных функций в `*apply` семейство функций базы.

Вычислить средний квадрат корня для каждого столбца в `data.frame` :

```
df <- data.frame(first=5:9, second=(0:4)^2, third=-1:3)

apply(df, 2, function(x) { sqrt(sum(x^2)) })
    first    second     third
15.968719 18.814888  3.872983
```

Создайте последовательность шагов по длине от наименьшего до наибольшего значения для каждой строки в матрице.

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

Анонимная функция также может стоять сама по себе:

```
(function() { 1 })()
[1] 1
```

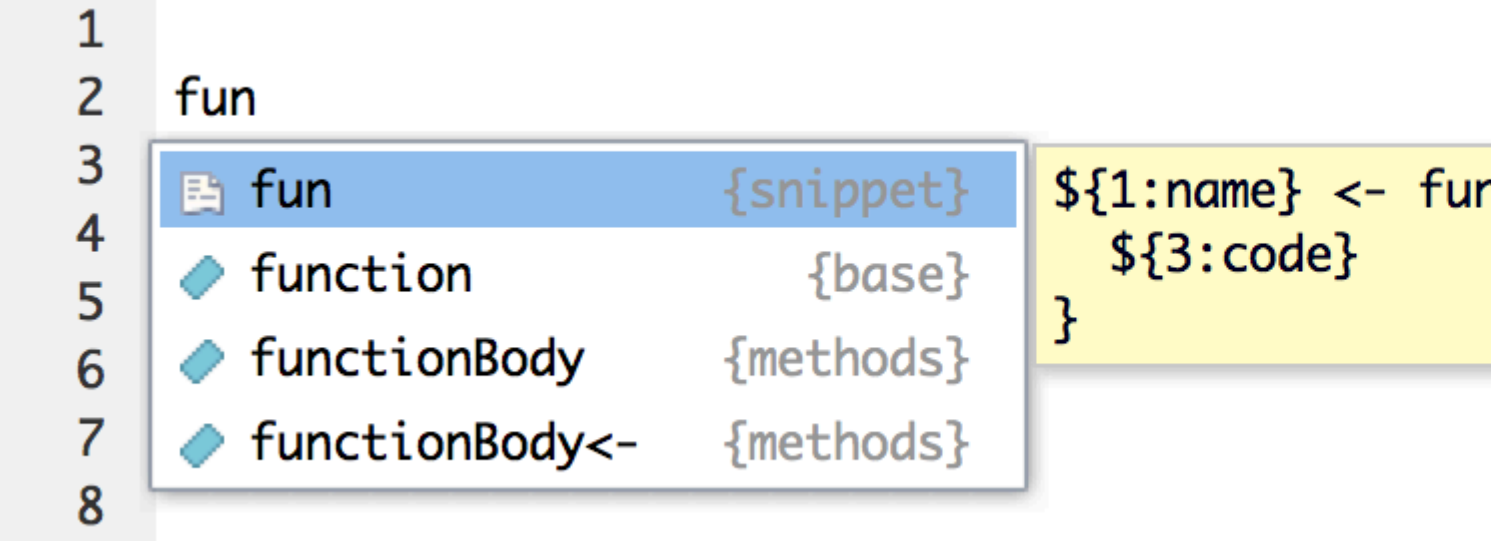
эквивалентно

```
f <- function() { 1 })
f()
```

```
[1] 1
```

Фрагменты кода RStudio

Это всего лишь небольшой взлом для тех, кто часто использует самоопределяемые функции.
Введите «fun» RStudio IDE и нажмите «TAB».



Результатом будет скелет новой функции.

```
name <- function(variables) {

}
```

Можно легко определить собственный шаблон фрагмента, т. Е. Тот, который приведен ниже

```
name <- function(df, x, y) {
        require(tidyverse)
        out <-
        return(out)
}
```

Опция « `Edit Snippets` в меню « `Global Options -> Code` .

Передача имен столбцов в качестве аргумента функции

Иногда хотелось бы передать имена столбцов из фрейма данных в функцию. Они могут быть представлены в виде строк и использоваться в функции с помощью `[[` . Давайте посмотрим на следующий пример, который выводит на базовую статистику R-групп выбранных переменных:

```
basic.stats <- function(dset, vars){
    for(i in 1:length(vars)){
        print(vars[i])
        print(summary(dset[[vars[i]]]))
    }
}

basic.stats(iris, c("Sepal.Length", "Petal.Width"))
```

В результате выполнения вышеописанного кода в R-консоли печатаются имена выбранных переменных и их основные сводные статистические данные (минимумы, первые квантили, медианы, средства, третьи кванты и максимумы). Код `dset[[vars[i]]]` выбирает i-й элемент из аргументов `vars` и выбирает соответствующий столбец в объявленном наборе данных `dset` . Например, объявление только `iris[["Sepal.Length"]]` будет печатать столбец `Sepal.Length` из набора данных `iris` как вектор.

Прочитайте Функции записи в R онлайн: https://riptutorial.com/ru/r/topic/7937/функции-записи-в-r

глава 124: Функции распределения

Вступление

R имеет множество встроенных функций для работы с распределением вероятностей, причем официальные документы начинаются с « `?Distributions` .

замечания

Как правило, четыре префикса:

- **d** - Функция **плотности** для данного распределения
- **p** - Функция кумулятивного распределения
- **q** -Вставить **квантиль,** связанный с данной вероятностью
- **r -** Получить **случайную** выборку

Распределения, встроенные в базовую установку R, см. В разделе « `?Distributions` .

Examples

Нормальное распределение

Давайте используем `*norm` в качестве примера. Из документации:

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

Поэтому, если бы я хотел знать значение стандартного нормального распределения в 0, я бы сделал

```
dnorm(0)
```

Это дает нам `0.3989423` , разумный ответ.

Точно так же `pnorm(0)` дает `.5` . Опять же, это имеет смысл, потому что половина распределения находится слева от 0.

`qnorm` будет по существу делать противоположность `pnorm` . `qnorm(.5)` дает `0` .

Наконец, существует функция `rnorm` :

```
rnorm(10)
```

Будет генерировать 10 образцов из стандартного нормального.

Если вы хотите изменить параметры данного дистрибутива, просто измените их так

```
rnorm(10, mean=4, sd= 3)
```

Биномиальное распределение

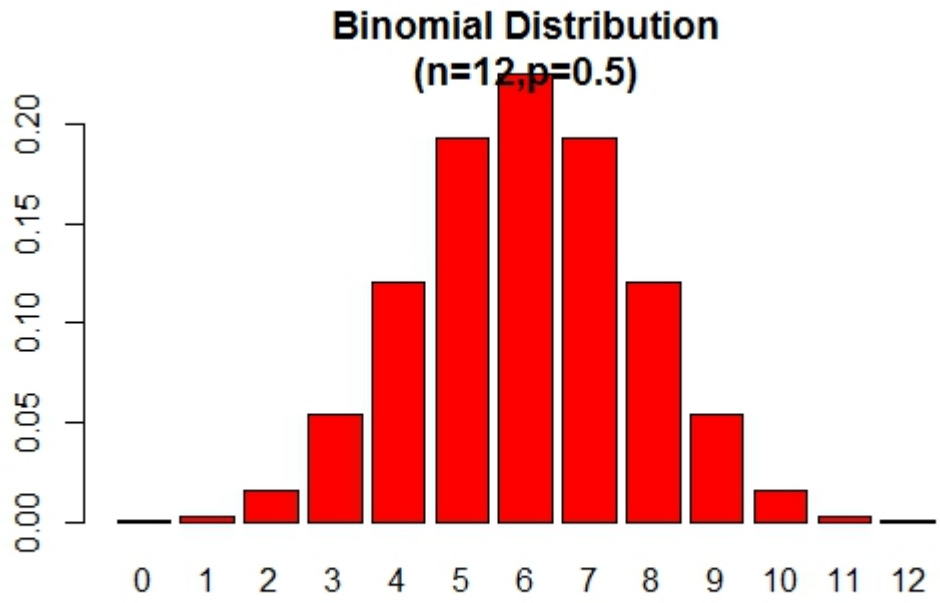
Теперь проиллюстрируем функции `dbinom` , `pbinom` , `qbinom` и `rbinom` определенные для *биномиального распределения* .

Функция `dbinom()` дает вероятности для различных значений биномиальной переменной. Минимально это требует трех аргументов. Первым аргументом для этой функции должен быть вектор квантилей (возможные значения случайной величины `X`). Второй и третий аргументы являются `defining parameters` распределения, а именно `n` (количество независимых испытаний) и `p` (вероятность успеха в каждом испытании). Например, для биномиального распределения с `n = 5` , `p = 0.5` возможные значения для X составляют `0,1,2,3,4,5` . То есть `dbinom(x,n,p)` дает значения вероятности `P( X = x )` для `x = 0, 1, 2, 3, 4, 5` .

```
#Binom(n = 5, p = 0.5) probabilities
> n <- 5; p<- 0.5; x <- 0:n
> dbinom(x,n,p)
[1] 0.03125 0.15625 0.31250 0.31250 0.15625 0.03125
#To verify the total probability is 1
> sum(dbinom(x,n,p))
[1] 1
>
```

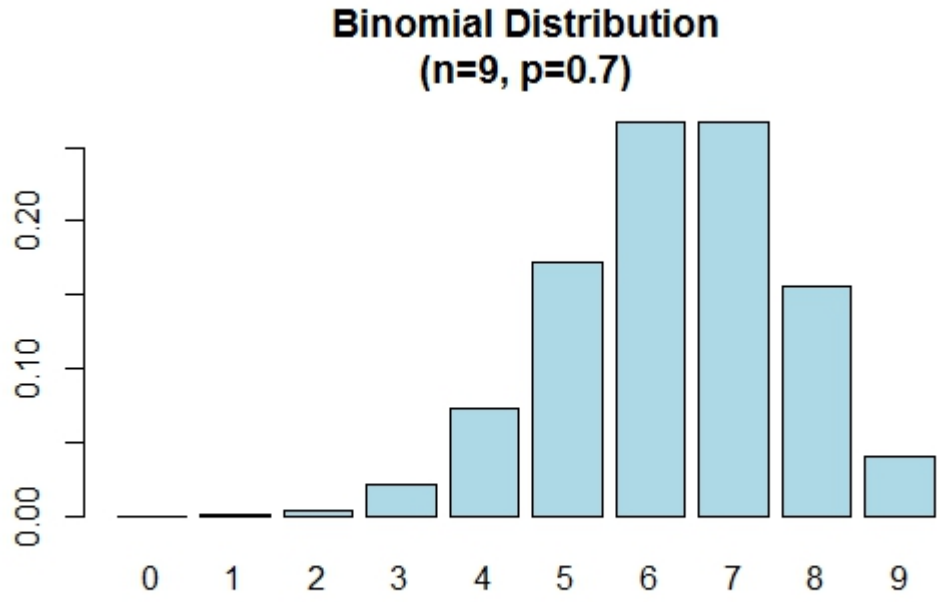
График распределения биномиальной вероятности можно отобразить, как показано на следующем рисунке:

```
> x <- 0:12
> prob <- dbinom(x,12,.5)
> barplot(prob,col = "red",ylim = c(0,.2),names.arg=x,
                           main="Binomial Distribution\n(n=12,p=0.5)")
```



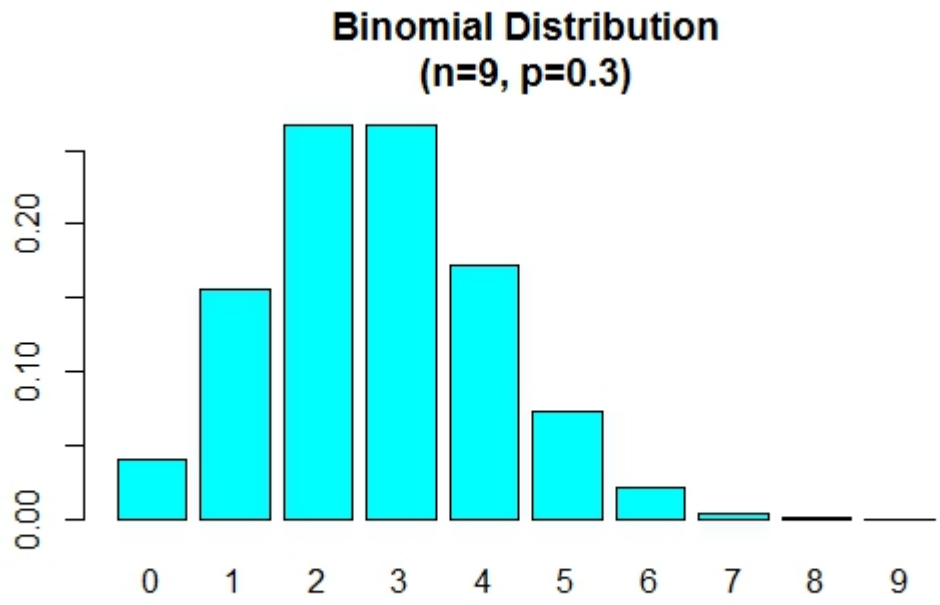
Заметим, что биномиальное распределение симметрично при `p = 0.5` . Чтобы продемонстрировать, что биномиальное распределение отрицательно искажено, когда `p` больше `0.5` , рассмотрим следующий пример:

```
> n=9; p=.7; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="Binomial Distribution\n(n=9, p=0.7)",col="lightblue")
```



Когда `p` меньше `0.5` биномиальное распределение положительно перекошено, как показано ниже.

```
> n=9; p=.3; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="Binomial Distribution\n(n=9, p=0.3)",col="cyan")
```



Теперь мы проиллюстрируем использование кумулятивной функции распределения `pbinom()` . Эта функция может быть использована для вычисления вероятностей, таких как `P( X <= x )` . Первым аргументом этой функции является вектор квантилей (значения x).

```
# Calculating Probabilities
# P(X <= 2) in a Bin(n=5,p=0.5) distribution
> pbinom(2,5,0.5)
[1] 0.5
```

Вышеуказанная вероятность также может быть получена следующим образом:

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

Для вычисления вероятностей типа: `P( a <= X <= b )`

```
# P(3<= X <= 5) = P(X=3) + P(X=4) + P(X=5) in a Bin(n=9,p=0.6) dist
> sum(dbinom(c(3,4,5),9,0.6))
[1] 0.4923556
>
```

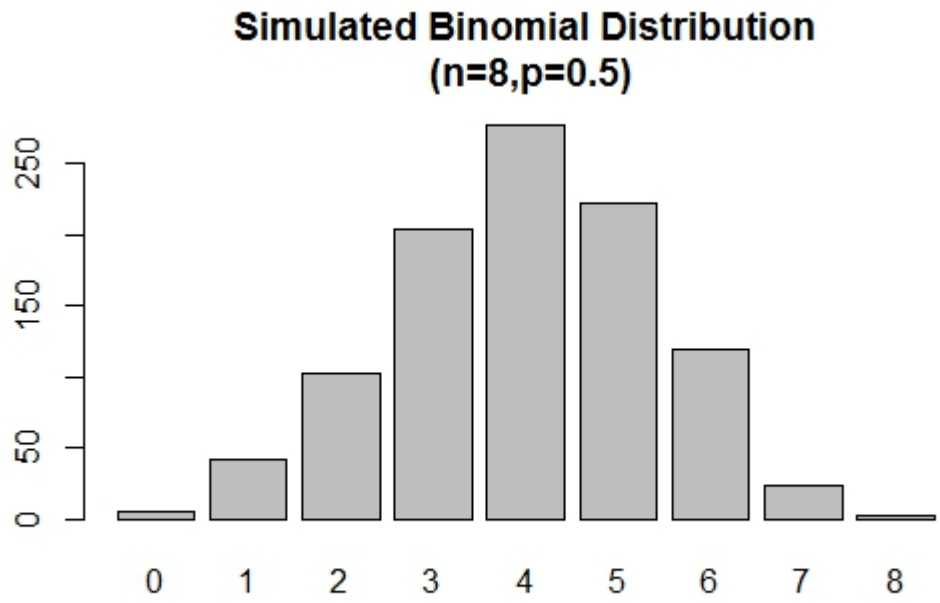
Представление биномиального распределения в виде таблицы:

```
> n = 10; p = 0.4; x = 0:n;
> prob = dbinom(x,n,p)
> cdf = pbinom(x,n,p)
```

```
> distTable = cbind(x,prob,cdf)
> distTable
       x          prob         cdf
 [1,]  0 0.0060466176 0.006046618
 [2,]  1 0.0403107840 0.046357402
 [3,]  2 0.1209323520 0.167289754
 [4,]  3 0.2149908480 0.382280602
 [5,]  4 0.2508226560 0.633103258
 [6,]  5 0.2006581248 0.833761382
 [7,]  6 0.1114767360 0.945238118
 [8,]  7 0.0424673280 0.987705446
 [9,]  8 0.0106168320 0.998322278
[10,]  9 0.0015728640 0.999895142
[11,] 10 0.0001048576 1.000000000
>
```

`rbinom()` используется для генерации случайных выборок заданных размеров с заданными значениями параметров.

```
# Simulation
> xVal<-names(table(rbinom(1000,8,.5)))
> barplot(as.vector(table(rbinom(1000,8,.5))),names.arg =xVal,
                   main="Simulated Binomial Distribution\n (n=8,p=0.5)")
```



Прочитайте Функции распределения онлайн: https://riptutorial.com/ru/r/topic/1885/функции-распределения

глава 125: Функциональное программирование

Examples

Встроенные функции более высокого порядка

R имеет набор встроенных функций более высокого порядка: `Map` , `Reduce` , `Filter` , `Find` , `Position` , `Negate` .

`Map` применяет данную функцию к списку значений:

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

`Reduce` последовательно применяет двоичную функцию к списку значений рекурсивным образом.

```
Reduce(`*`, 1:10)
```

`Filter` задает функцию предиката, а список значений возвращает отфильтрованный список, содержащий только значения, для которых предикатная функция имеет значение ИСТИНА.

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

`Find` заданную функцию предиката, а список значений возвращает первое значение, для которого предикатная функция TRUE.

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

`Position` заданная предикатной функцией, и список значений возвращает позицию первого значения в списке, для которого предикатная функция TRUE.

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

`Negate` инвертирует функцию предиката, заставляя ее возвращать FALSE для значений, где она возвращала TRUE и наоборот.

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

Прочитайте Функциональное программирование онлайн: https://riptutorial.com/ru/r/topic/5050/функциональное-программирование

глава 126: Функция strsplit

Синтаксис

- strsplit (
- Икс
- Трещина
- fixed = FALSE
- perl = FALSE
- useBytes = FALSE)

Examples

Вступление

`strsplit` - полезная функция для разбиения вектора на список на некотором символьном шаблоне. С типичными инструментами R весь список может быть реинкорпорирован в data.frame, или часть списка может использоваться в графическом упражнении.

Вот общее использование `strsplit` : разбить вектор символа вдоль разделителя запятой:

```
temp <- c("this,that,other", "hat,scarf,food", "woman,man,child")
# get a list split by commas
myList <- strsplit(temp, split=",")
# print myList
myList
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

Как было намечено выше, аргумент split не ограничивается символами, но может следовать шаблону, продиктованному регулярным выражением. Например, temp2 идентичен темпам выше, за исключением того, что разделители были изменены для каждого элемента. Мы можем воспользоваться тем фактом, что аргумент split принимает регулярные выражения для облегчения неравномерности вектора.

```
temp2 <- c("this, that, other", "hat,scarf ,food", "woman; man ; child")
myList2 <- strsplit(temp2, split=" ?[,;] ?")
myList2
[[1]]
[1] "this"  "that"  "other"
```

```
[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

Примечания :

1. разбиение синтаксиса регулярных выражений выходит за рамки этого примера.
2. Иногда совпадение регулярных выражений может замедлить процесс. Как и во многих функциях R, которые позволяют использовать регулярные выражения, фиксированный аргумент доступен, чтобы сказать, что R соответствует буквам разделения.

Прочитайте Функция strsplit онлайн: https://riptutorial.com/ru/r/topic/2762/функция-strsplit

глава 127: Функция разделения

Examples

Основное использование раскола

`split` позволяет разделить вектор или data.frame на ведра в отношении переменных фактора / группы. Эта вентиляция в ведрах принимает форму списка, который затем может использоваться для применения группового вычисления (`for` циклов или `lapply` / `sapply`).

В первом примере показано использование `split` на вектор:

Рассмотрим следующий вектор букв:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

Цель состоит в том, чтобы отделить эти буквы от `voyels` и `consonants` , т. `voyels` Разделить их на буквенный тип.

Сначала создадим вектор группировки:

```
 vowels <- c('a','e','i','o','u','y')
 letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Обратите внимание, что `letter_type` имеет ту же длину, что и наши векторные `testdata` . Теперь мы можем `split` эти тестовые данные на две группы, `vowels` и `consonants` :

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"

#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Следовательно, результатом является список, имена которого исходят из нашего вектора `letter_type` / factor `letter_type` .

`split` также имеет метод работы с data.frames.

Рассмотрим, например, данные `iris` :

```
data(iris)
```

Используя `split` , можно создать список, содержащий один файл data.frame per iris (variable: Species):

```
> liris <- split(iris, iris$Species)
> names(liris)
[1] "setosa"     "versicolor" "virginica"
> head(liris$setosa)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

(содержит только данные для группы setosa).

Одной из примерных операций было бы вычисление матрицы корреляции для каждой диафрагмы; тогда можно было бы использовать `lapply` :

```
> (lcor <- lapply(liris, FUN=function(df) cor(df[,1:4])))

    $setosa
              Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.7425467    0.2671758   0.2780984
Sepal.Width     0.7425467   1.0000000    0.1777000   0.2327520
Petal.Length    0.2671758   0.1777000    1.0000000   0.3316300
Petal.Width     0.2780984   0.2327520    0.3316300   1.0000000

$versicolor
              Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.5259107    0.7540490   0.5464611
Sepal.Width     0.5259107   1.0000000    0.5605221   0.6639987
Petal.Length    0.7540490   0.5605221    1.0000000   0.7866681
Petal.Width     0.5464611   0.6639987    0.7866681   1.0000000

$virginica
              Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000   0.4572278    0.8642247   0.2811077
Sepal.Width     0.4572278   1.0000000    0.4010446   0.5377280
Petal.Length    0.8642247   0.4010446    1.0000000   0.3221082
Petal.Width     0.2811077   0.5377280    0.3221082   1.0000000
```

Затем мы можем получить для каждой группы лучшую пару коррелированных переменных: (корреляционная матрица изменена / расплавлена, диагональ отфильтрована и выполняется выбор наилучшей записи)

```
> library(reshape)
> (topcor <- lapply(lcor, FUN=function(cormat){
   correlations <- melt(cormat,variable_name="correlatio);
   filtered <- correlations[correlations$X1 != correlations$X2,];
   filtered[which.max(filtered$correlation),]
}))

$setosa
           X1           X2     correlation
2 Sepal.Width Sepal.Length       0.7425467

$versicolor
            X1           X2     correlation
```

```
12 Petal.Width Petal.Length       0.7866681

$virginica
            X1           X2     correlation
3 Petal.Length Sepal.Length       0.8642247
```

Обратите внимание, что одни вычисления выполняются на таком групповом уровне, может быть интересен укладку результатов, что можно сделать с помощью:

```
> (result <- do.call("rbind", topcor))

                     X1           X2     correlation
setosa      Sepal.Width Sepal.Length       0.7425467
versicolor  Petal.Width Petal.Length       0.7866681
virginica  Petal.Length Sepal.Length       0.8642247
```

Использование split в парадигме split-apply-comb

Популярной формой анализа данных является split-apply-comb , в котором вы разбиваете свои данные на группы, применяете какую-то обработку в каждой группе, а затем объединяете результаты.

Рассмотрим анализ данных, где мы хотим получить два автомобиля с лучшими милями на галлон (миль на галлон) для каждого количества цилиндров (цил) во встроенном наборе данных mtcars. Во-первых, мы разделили `mtcars` данных `mtcars` на количество цилиндров:

```
(spl <- split(mtcars, mtcars$cyl))
# $`4`
#                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Datsun 710     22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
# Merc 240D      24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
# Merc 230       22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
# Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
# ...
#
# $`6`
#                 mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Mazda RX4      21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
# Mazda RX4 Wag  21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
# Valiant        18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
# ...
#
# $`8`
#                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
# Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
# Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
# Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
# ...
```

Это вернуло список кадров данных, по одному для каждого подсчета баллонов. Как видно из вывода, мы могли бы получить соответствующие фреймы данных с помощью `spl$`4`` , `spl$`6``

и `spl$`8`` (некоторые могут показаться более визуально привлекательными для использования `spl$"4"` или `spl[["4"]]`).

Теперь мы можем использовать `lapply` чтобы перебирать этот список, применяя нашу функцию, которая извлекает автомобили с лучшими значениями 2 mpg от каждого элемента списка:

```
(best2 <- lapply(spl, function(x) tail(x[order(x$mpg),], 2)))
# $`4`
#                 mpg cyl disp hp drat    wt  qsec vs am gear carb
# Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
# Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
#
# $`6`
#                 mpg cyl disp  hp drat    wt  qsec vs am gear carb
# Mazda RX4 Wag  21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
# Hornet 4 Drive 21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
#
# $`8`
#                     mpg cyl disp  hp drat    wt  qsec vs am gear carb
# Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
# Pontiac Firebird  19.2   8  400 175 3.08 3.845 17.05  0  0    3    2
```

Наконец, мы можем объединить все вместе, используя `rbind` . Мы хотим называть `rbind(best2[["4"]], best2[["6"]], best2[["8"]])` , но это было бы утомительно, если бы у нас был огромный список. В результате мы используем:

```
do.call(rbind, best2)
#                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
# 4.Fiat 128          32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
# 4.Toyota Corolla    33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
# 6.Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
# 6.Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
# 8.Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
# 8.Pontiac Firebird  19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
```

Это возвращает результат `rbind` (аргумент 1, функция) со всеми элементами `best2` (аргумент 2, список), переданный в качестве аргументов.

С помощью простых анализов, подобных этому, он может быть более компактным (и, возможно, гораздо менее читаемым!), Чтобы сделать весь split-apply-comb в одной строке кода:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg),], 2)))
```

Стоит также отметить, что `lapply(split(x,f), FUN)` может быть альтернативно обрамлена с `?by` функции:

```
by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg),], 2))
do.call(rbind, by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg),], 2)))
```

Прочитайте Функция разделения онлайн: https://riptutorial.com/ru/r/topic/1073/функция-разделения

глава 128: Цветовые схемы для графики

Examples

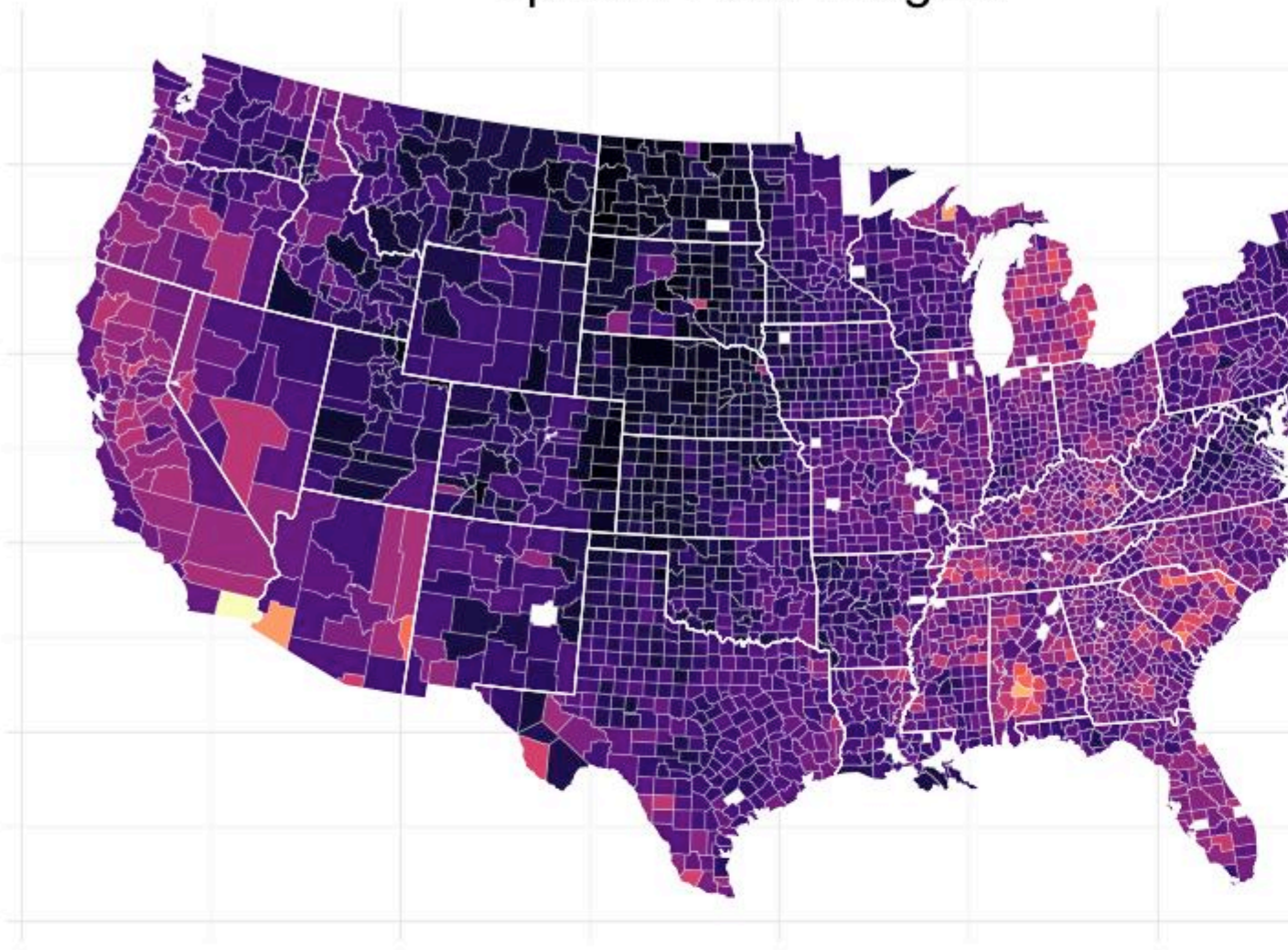
viridis - печатные и цветные слепые палитры

Viridis (названный в честь рыбы chromis viridis) - это недавно разработанная цветовая схема для библиотеки `matplotlib` Python (видео-презентация по ссылке объясняет, как была разработана цветовая схема и каковы ее основные преимущества). Он легко переносится на `R`

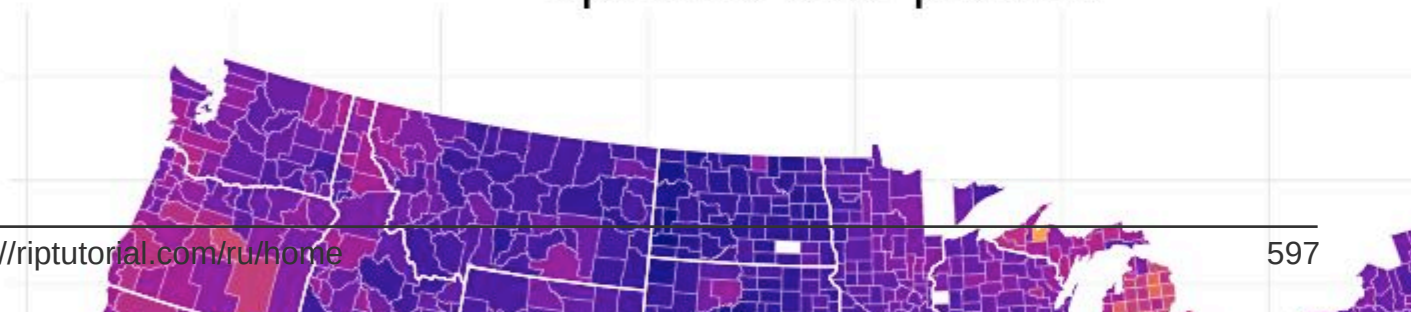
Существует 4 варианта цветовых схем: `magma` , `plasma` , `inferno` и `viridis` (по умолчанию). Они выбираются с параметром `option` и кодируются как `A` , `B` , `C` и `D` соответственно. Чтобы иметь представление о 4 цветовых схемах, посмотрите на карты:

U

option A aka 'magma'



option C aka 'plasma'



(изображение souce)

Пакет можно установить из CRAN или github .

Виньетка для пакета `viridis` просто великолепна.

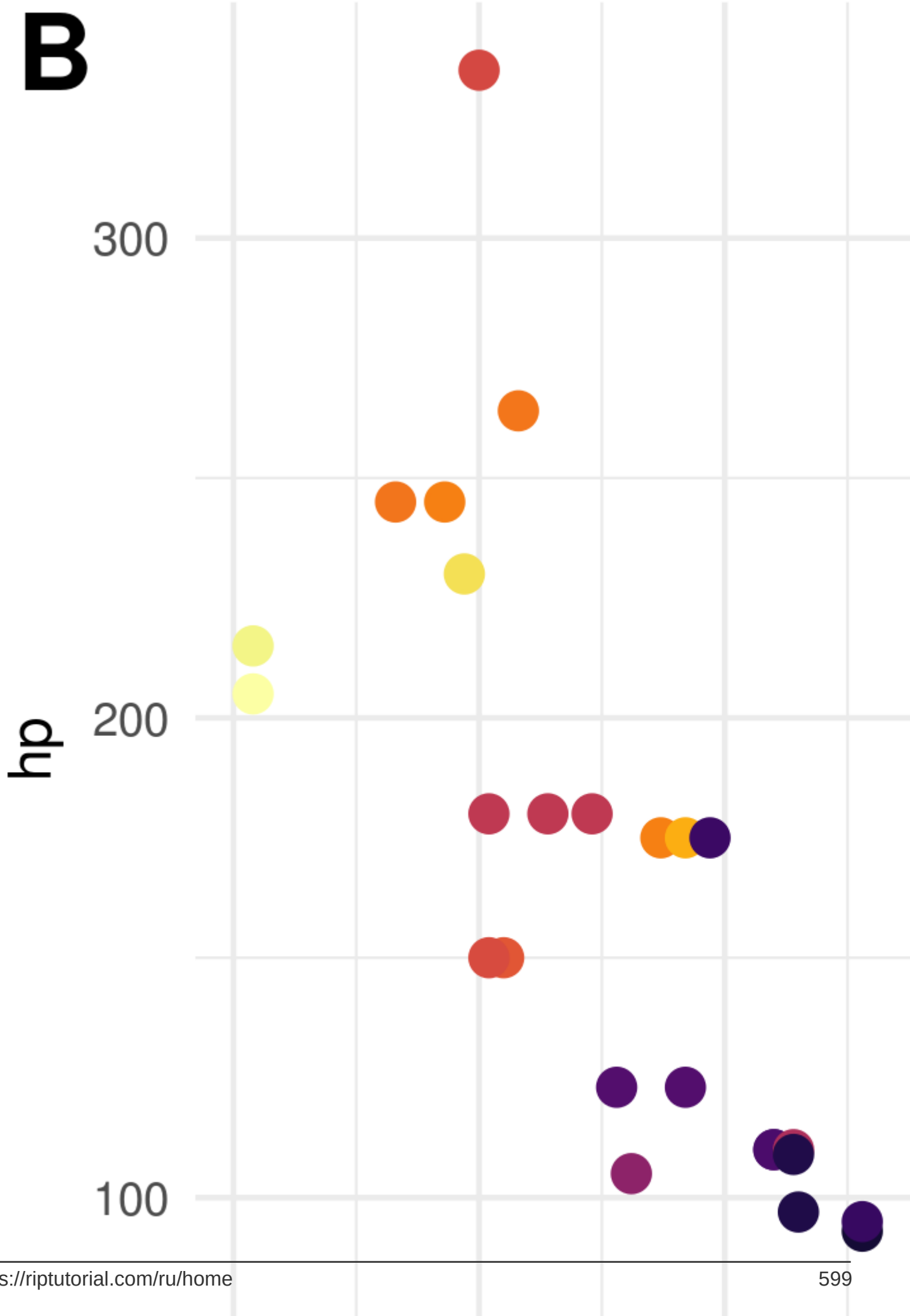
`ggplot2` особенностью цветовой схемы `viridis` является интеграция с `ggplot2` . В рамках пакета два `ggplot2` -специфические функции определены: `scale_color_viridis()` и `scale_fill_viridis()` . См. Пример ниже:

```
library(viridis)
library(ggplot2)

gg1 <- ggplot(mtcars)+
    geom_point(aes(x = mpg, y = hp, color = disp), size = 3)+
    scale_color_viridis(option = "B")+
    theme_minimal()+
    theme(legend.position = c(.8,.8))

gg2 <- ggplot(mtcars)+
        geom_violin(aes(x = factor(cyl), y = hp, fill = factor(cyl)))+
        scale_fill_viridis(discrete = T)+
        theme_minimal()+
        theme(legend.position = 'none')

library(cowplot)
output <- plot_grid(gg1,gg2, labels = c('B','D'),label_size = 20)
print(output)
```


B
300
200
100
hp

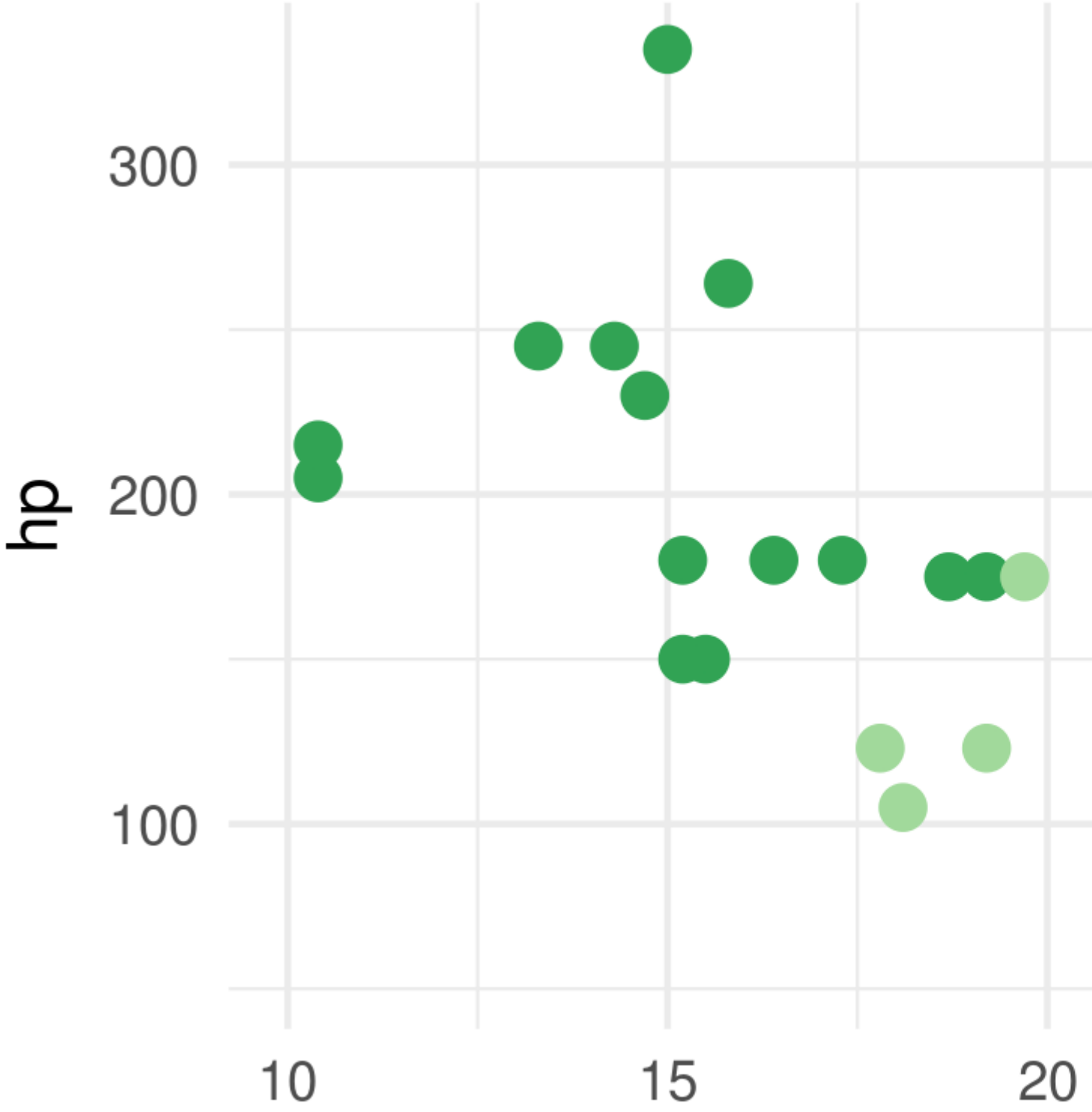
очень популярный инструмент для выбора гармоничного соответствия цветовых палитр. `RColorBrewer` - это порт проекта для `R` и обеспечивает также палитры, `RColorBrewer` .

Пример использования

```
colors_vec <- brewer.pal(5, name = 'BrBG')
print(colors_vec)
[1] "#A6611A" "#DFC27D" "#F5F5F5" "#80CDC1" "#018571"
```

`RColorBrewer` создает параметры раскраски для `ggplot2` : `scale_color_brewer` и `scale_fill_brewer` .

```
library(ggplot2)
ggplot(mtcars)+
        geom_point(aes(x = mpg, y = hp, color = factor(cyl)), size = 3)+
        scale_color_brewer(palette = 'Greens')+
        theme_minimal()+
        theme(legend.position = c(.8,.8))
```



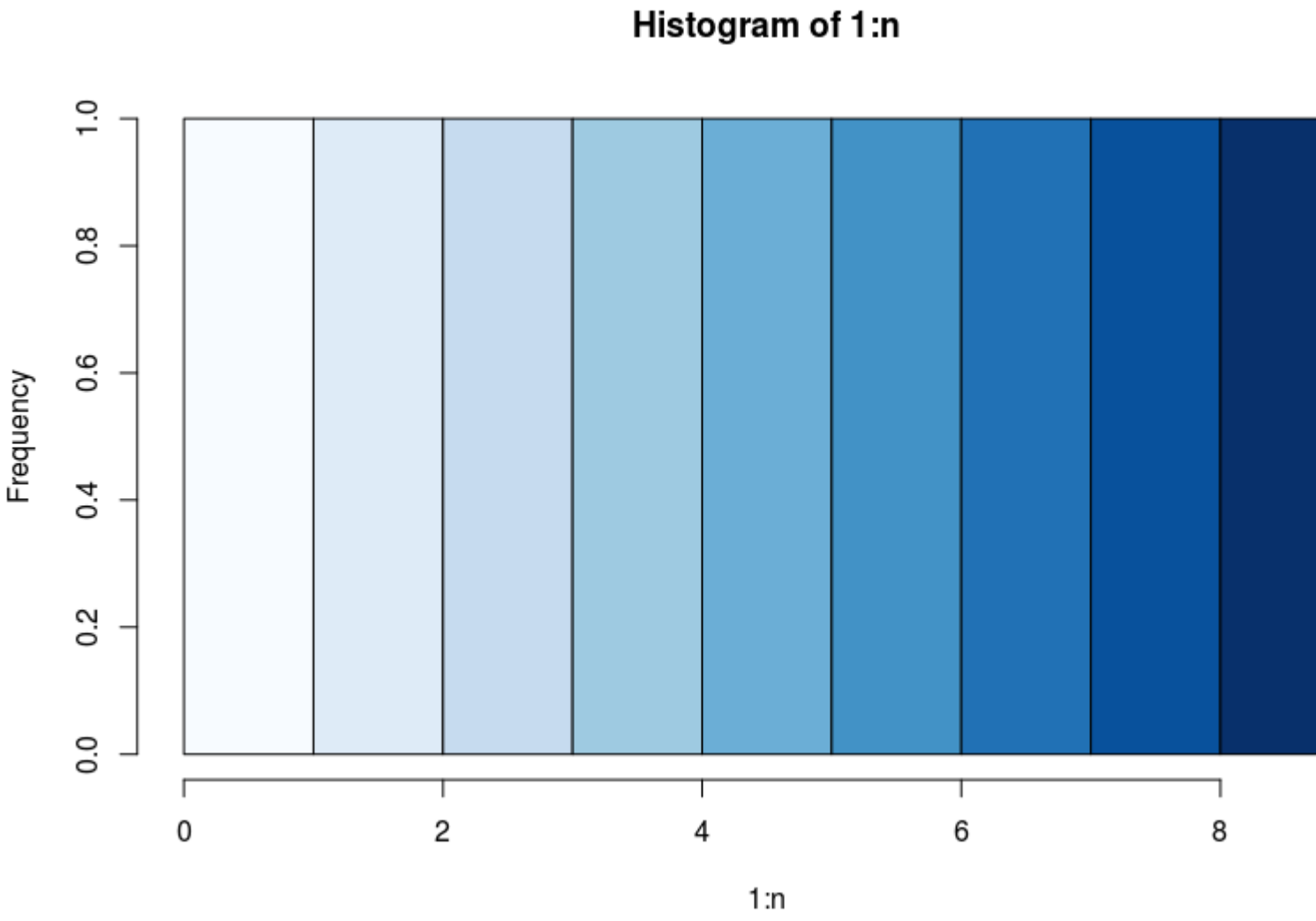
Удобная функция для прорисовки вектора цветов

Довольно часто возникает необходимость взглянуть на выбранную цветовую палитру. Одним из элегантных решений является следующая самоопределяемая функция:

```
color_glimpse <- function(colors_string){
        n <- length(colors_string)
        hist(1:n,breaks=0:n,col=colors_string)
}
```

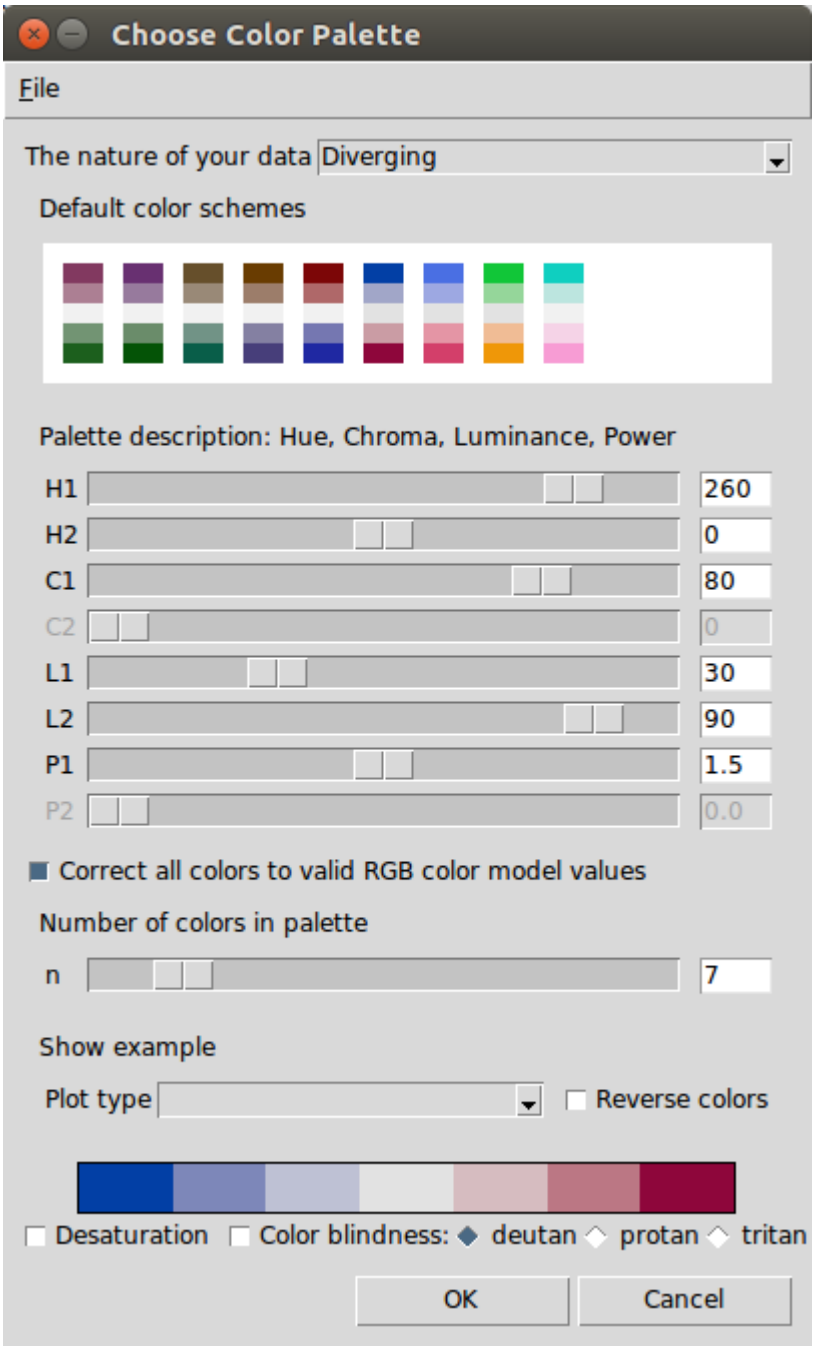
Пример использования

```
color_glimpse(blues9)
```



Цветовое пространство - нажмите и перетащите интерфейс для цветов

`colorspace` пакета предоставляет графический интерфейс для выбора палитры. При вызове функции `choose_palette()` появляется следующее окно:



Когда выбрана палитра, просто нажмите « `OK` и не забудьте сохранить результат в переменной, например, `pal` .

```
pal <- choose_palette()
```

Выход представляет собой функцию, которая принимает `n` (число) в качестве входных данных и создает вектор цвета длины `n` соответствии с выбранной палитрой.

```
pal(10)
[1] "#023FA5" "#6371AF" "#959CC3" "#BEC1D4" "#DBDCE0" "#E0DBDC" "#D6BCC0" "#C6909A" "#AE5A6D"
"#8E063B"
```

основные функции цвета R

`colors()` функции `colors()` перечисляют все имена цветов, которые распознаются R. Существует хороший PDF-документ, где можно увидеть эти цвета.

`colorRampPalette` создает функцию, которая интерполирует набор заданных цветов для создания новых цветовых палитр. Эта функция вывода принимает `n` (число) в качестве входных данных и создает вектор цвета длины `n` интерполирующий исходные цвета.

```
pal <- colorRampPalette(c('white','red'))
pal(5)
[1] "#FFFFFF" "#FFBFBF" "#FF7F7F" "#FF3F3F" "#FF0000"
```

Любой конкретный цвет может быть создан с помощью функции `rgb()` :

```
rgb(0,1,0)
```

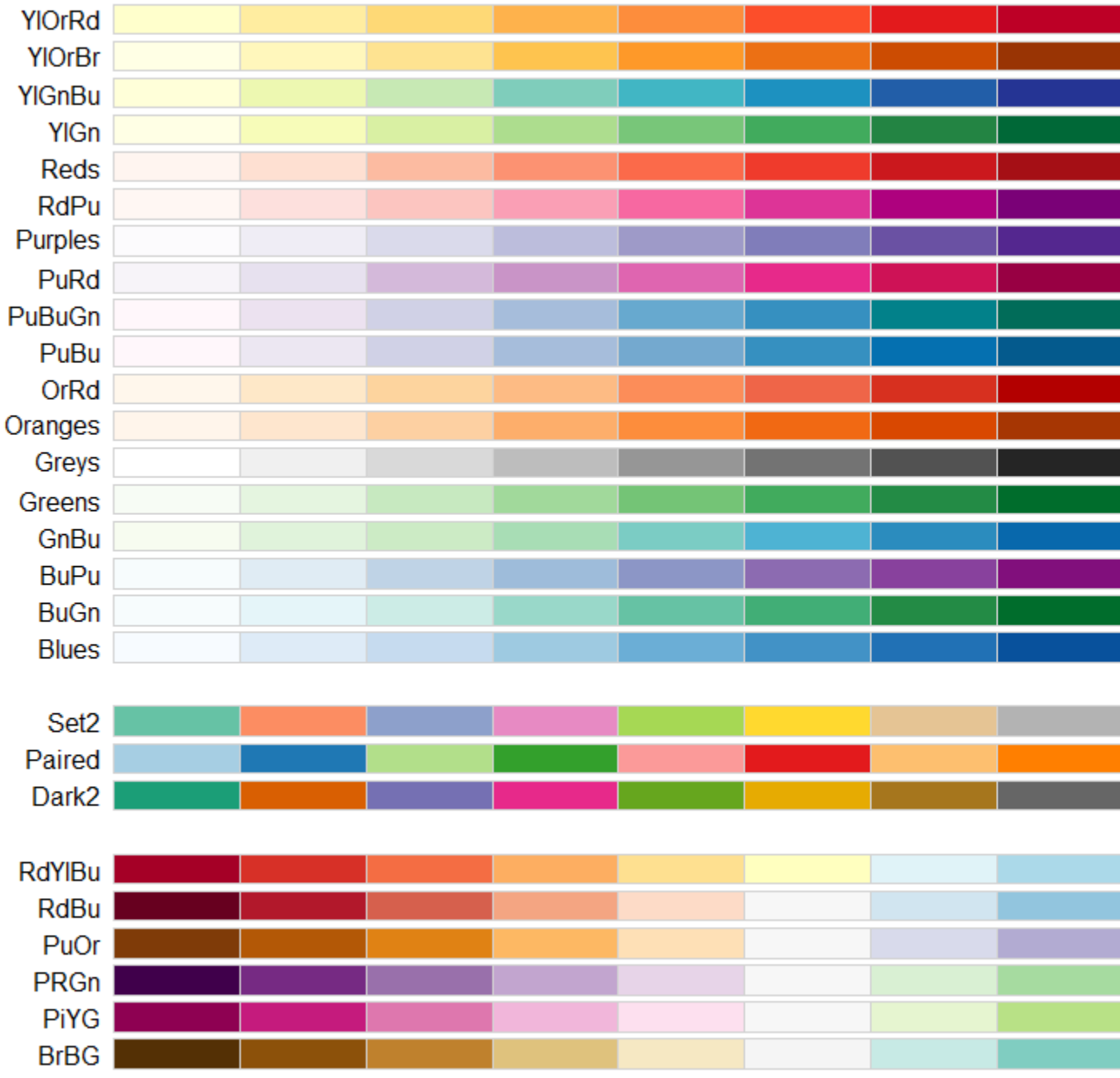
производит `green` цвет.

Панели, удобные для цветной печати

Несмотря на то, что люди с цветным слепым могут распознавать широкий спектр цветов, может быть трудно различать определенные цвета.

`RColorBrewer` обеспечивает `RColorBrewer` палитры:

```
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```



Color Universal Design из Токийского университета предлагает следующие палитры:

```
#palette using grey
```

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")

#palette using black
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
"#CC79A7")
```

Прочитайте Цветовые схемы для графики онлайн: https://riptutorial.com/ru/r/topic/8005/цветовые-схемы-для-графики

глава 129: Числовые классы и режимы хранения

Examples

числовой

Числовой представляет целые числа и удваивает и является режимом по умолчанию, назначенным векторам чисел. Функция `is.numeric()` будет определять, является ли вектор числовым. Важно отметить, что хотя целые числа и удвоения пройдут `is.numeric()` , функция `as.numeric()` всегда будет пытаться преобразовать в double.

```
x <- 12.3
y <- 12L

#confirm types
typeof(x)
[1] "double"
typeof(y)
[1] "integer"

# confirm both numeric
is.numeric(x)
[1] TRUE
is.numeric(y)
[1] TRUE

# logical to numeric
as.numeric(TRUE)
[1] 1

# While TRUE == 1, it is a double and not an integer
is.integer(as.numeric(TRUE))
[1] FALSE
```

Двумя значениями являются числовые значения по умолчанию R. Это векторы с двойной точностью, что означает, что они занимают 8 байтов памяти для каждого значения в векторе. R не имеет единого типа данных точности, и поэтому все реальные числа хранятся в формате двойной точности.

```
is.double(1)
TRUE
is.double(1.0)
TRUE
is.double(1L)
FALSE
```

Целые числа - это целые числа, которые могут быть записаны без дробной составляющей. Целые числа представлены числом с L после него. Любое число без L после него будет считаться двойным.

```
typeof(1)
[1] "double"
class(1)
[1] "numeric"
typeof(1L)
[1] "integer"
class(1L)
[1] "integer"
```

Хотя в большинстве случаев использование целого или двойного значения не имеет значения, иногда замена двойников целыми числами будет меньше памяти и времени работы. Двойной вектор использует 8 байтов на элемент, а целочисленный вектор использует только 4 байта на элемент. По мере увеличения размеров векторов использование подходящих типов может значительно ускорить процессы.

```
#  test speed on lots of arithmetic
microbenchmark(
  for( i in 1:100000){
  2L * i
  10L + i
},

for( i in 1:100000){
  2.0 * i
  10.0 + i
}
)
Unit: milliseconds
                                          expr      min       lq     mean   median       uq
max neval
 for (i in 1:1e+05) {     2L * i     10L + i } 40.74775 42.34747 50.70543 42.99120 65.46864
94.11804   100
   for (i in 1:1e+05) {     2 * i     10 + i } 41.07807 42.38358 53.52588 44.26364 65.84971
83.00456   100
```

Прочитайте Числовые классы и режимы хранения онлайн:
https://riptutorial.com/ru/r/topic/9018/числовые-классы-и-режимы-хранения

глава 130: Чтение и запись строк

замечания

Связанные документы:

- Вход пользователя

Examples

Печать и отображение строк

R имеет несколько встроенных функций, которые могут использоваться для печати или отображения информации, но `print` и `cat` являются самыми основными. Поскольку R - интерпретируемый язык , вы можете попробовать их прямо в консоли R:

```
print("Hello World")
#[1] "Hello World"
cat("Hello World\n")
#Hello World
```

Обратите внимание на разницу в обоих входах и выходах для двух функций. (Примечание: в значении `x` созданных с помощью `x <- "Hello World"` , нет никаких котировочных символов. Они добавляются `print` на выходном этапе.)

`cat` принимает один или несколько векторов символов в качестве аргументов и выводит их на консоль. Если вектор символов имеет длину больше 1, аргументы разделяются пробелом (по умолчанию):

```
cat(c("hello", "world", "\n"))
#hello world
```

Без символа новой строки (`\n`) вывод будет следующим:

```
cat("Hello World")
#Hello World>
```

Приглашение для следующей команды появляется сразу после выхода. (Некоторые консоли, такие как RStudio, могут автоматически добавлять новую строку к строкам, которые не заканчиваются новой линией.)

`print` является примером «общей» функции, что означает, что класс первого переданного аргумента обнаружен, и для вывода используется *метод* класса. Для символьного вектора, такого как `"Hello World"` , результат аналогичен выходу `cat` . Однако символьная строка

цитируется, а число `[1]` выводится, чтобы указать первый элемент символьного вектора (в этом случае - первый и единственный элемент):

```
print("Hello World")
#[1] "Hello World"
```

Этот метод печати по умолчанию также является тем, что мы видим, когда просто запрашиваем R для печати переменной. Обратите внимание, как вывод ввода `s` совпадает с выводом `print(s)` или `print("Hello World")` :

```
s <- "Hello World"
s
#[1] "Hello World"
```

Или даже не присваивая его чему-либо:

```
"Hello World"
#[1] "Hello World"
```

Если мы добавим еще одну символьную строку в качестве второго элемента вектора (используя функцию `c()` чтобы **с** онкатировал элементы вместе), то поведение `print()` выглядит немного иначе, чем поведение `cat` :

```
print(c("Hello World", "Here I am."))
#[1] "Hello World" "Here I am."
```

Обратите внимание, что функция `c()` *не* выполняет конкатенацию строк. (Для этой цели нужно использовать `paste` .) R показывает, что вектор символов имеет два элемента, цитируя их отдельно. Если у нас есть вектор, достаточно длинный, чтобы охватить несколько строк, R будет печатать индекс элемента, начинающегося с каждой строки, точно так же, как он печатает `[1]` в начале первой строки.

```
c("Hello World", "Here I am!", "This next string is really long.")
#[1] "Hello World"                    "Here I am!"
#[3] "This next string is really long."
```

Конкретное поведение `print` зависит от *класса* объекта, переданного функции.

Если мы вызываем `print` объекта с другим классом, например «числовой» или «логический», кавычки опускаются с вывода, чтобы указать, что мы имеем дело с объектом, который не является символьным классом:

```
print(1)
#[1] 1
print(TRUE)
#[1] TRUE
```

Факторные объекты печатаются так же, как символьные переменные, которые часто создают неоднозначность, когда вывод консоли используется для отображения объектов в телах вопросов SO. Редко использовать `cat` или `print` за исключением интерактивного контекста. Явный вызов `print()` особенно редок (если вы не хотите подавить внешний вид кавычек или просмотреть объект, который возвращается как `invisible` функцией), поскольку ввод `foo` на консоли является ярлыком для `print(foo)` . Интерактивная консоль R известна как REPL, «read-eval-print-loop». Функция `cat` лучше всего сохраняется для специальных целей (например, запись вывода в открытое подключение к файлу). Иногда он используется внутри функций (когда вызовы `print()` подавляются), однако **использование функции `cat()` внутри функции для генерации вывода на консоль является плохой практикой** . Предпочтительным методом является `message()` или `warning()` для промежуточных сообщений; они ведут себя аналогично `cat` но могут быть необязательно подавлены конечным пользователем. Конечный результат должен быть просто возвращен, чтобы пользователь мог назначить его, если необходимо, сохранить его.

```
message("hello world")
#hello world
suppressMessages(message("hello world"))
```

Чтение или запись в файл

Не всегда у нас есть свобода чтения или записи на локальный системный путь. Например, если R-stream streaming map-reduce необходимо читать и записывать в соединение с файлом. Могут быть и другие сценарии, где вы выходите за пределы локальной системы, и с появлением облака и больших данных это становится все более распространенным явлением. Один из способов сделать это - в логической последовательности.

Установите соединение с `file()` для чтения с помощью команды `file()` («r» для режима чтения):

```
conn <- file("/path/example.data", "r") #when file is in local system
conn1 <- file("stdin", "r") #when just standard input/output for files are available
```

Поскольку это установит только подключение к файлу, можно прочитать данные из этих подключений файлов следующим образом:

```
line <- readLines(conn, n=1, warn=FALSE)
```

Здесь мы читаем данные из файла подключение `conn` построчно , как `n=1` . можно изменить значение `n` (например, 10, 20 и т. д.) для чтения блоков данных для более быстрого считывания (10 или 20 строк, считанных за один раз). Чтобы прочитать полный файл за один раз, установите `n=-1` .

После обработки данных или выполнения модели; можно вернуть результаты обратно к

подключению к файлу, используя множество разных команд, таких как `writeLines(),cat()` и т. д., которые способны записывать в соединение с файлом. Однако все эти команды будут использовать соединение с файлом, установленное для записи. Это можно сделать с помощью команды `file()` :

```
conn2 <- file("/path/result.data", "w") #when file is in local system
conn3 <- file("stdout", "w") #when just standard input/output for files are available
```

Затем напишите данные следующим образом:

```
writeLines("text",conn2, sep = "\n")
```

Захват вывода команды операционной системы

Функции, возвращающие вектор символов

База `R` имеет две функции для вызова системной команды. Оба требуют дополнительный параметр для захвата вывода системной команды.

```
system("top -a -b -n 1", intern = TRUE)
system2("top", "-a -b -n 1", stdout = TRUE)
```

Оба возвращают вектор символов.

```
  [1] "top - 08:52:03 up 70 days, 15:09,  0 users,  load average: 0.00, 0.00, 0.00"
  [2] "Tasks: 125 total,   1 running, 124 sleeping,   0 stopped,   0 zombie"
  [3] "Cpu(s):  0.9%us,  0.3%sy,  0.0%ni, 98.7%id,  0.1%wa,  0.0%hi,  0.0%si,  0.0%st"
  [4] "Mem:  12194312k total,  3613292k used,  8581020k free,   216940k buffers"
  [5] "Swap: 12582908k total,  2334156k used, 10248752k free,  1682340k cached"
  [6] ""
  [7] "  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND           "
  [8] "11300 root      20   0 1278m 375m 3696 S  0.0  3.2 124:40.92 trala             "
  [9] " 6093 user1     20   0 1817m 269m 1888 S  0.0  2.3  12:17.96 R                 "
 [10] " 4949 user2     20   0 1917m 214m 1888 S  0.0  1.8  11:16.73 R                 "
```

> Для иллюстрации используется команда `top -a -b -n 1` команды UNIX. Это спецификация ОС и, возможно, ее необходимо изменить, чтобы запустить примеры на вашем компьютере.

Пакет `devtools` имеет функцию для запуска системной команды и записи вывода без дополнительного параметра. Он также возвращает вектор символов.

```
devtools::system_output("top", "-a -b -n 1")
```

Функции, которые возвращают кадр данных

Функция `fread` в package `data.table` позволяет выполнить команду оболочки и прочитать вывод, например `read.table` . Он возвращает `data.table` или `data.frame` .

```
fread("top -a -b -n 1", check.names = TRUE)
       PID     USER PR NI  VIRT  RES  SHR S X.CPU X.MEM     TIME.         COMMAND
  1: 11300     root 20  0 1278m 375m 3696 S     0   3.2 124:40.92           trala
  2:  6093    user1 20  0 1817m 269m 1888 S     0   2.3  12:18.56               R
  3:  4949    user2 20  0 1917m 214m 1888 S     0   1.8  11:17.33               R
  4:  7922    user3 20  0 3094m 131m 1892 S     0   1.1  21:04.95               R
```

Обратите внимание, что `fread` автоматически пропускает верхние 6 строк заголовка.

> Здесь был добавлен параметр `check.names = TRUE` для преобразования `%CPU` , `%MEN` и `TIME+` в синтаксически допустимые имена столбцов.

Прочитайте Чтение и запись строк онлайн: https://riptutorial.com/ru/r/topic/5541/чтение-и-запись-строк

глава 131: Чтение и запись табличных данных в текстовых файлах (CSV, TSV и т. Д.)

Синтаксис

- read.csv (file, header = TRUE, sep = ",", quote = "" ", dec =". ", fill = TRUE, comment.char =" ", ...)
- read.csv2 (file, header = TRUE, sep = ";", quote = "" ", dec =", ", fill = TRUE, comment.char =" ", ...)
- readr :: read_csv (файл, col_names = TRUE, col_types = NULL, locale = default_locale (), na = c ("", "NA"), comment = "", trim_ws = TRUE, skip = 0, n_max = -1 , progress = interactive ())
- data.table :: fread (input, sep = "auto", sep2 = "auto", nrows = -1L, header = "auto", na.strings = "NA", strAsAsFactors = FALSE, verbose = getOption ("datatable .wowbose "), autostart = 1L, skip = . ")". else ",", col.names, check.names = FALSE, encoding = "unknown", strip.white = TRUE, showProgress = getOption ("datatable.showProgress"), # default: ИСТИНА. table = getOption ("datatable.fread.datatable") # default: TRUE)

параметры

параметр	подробности
файл	имя файла CSV для чтения
заголовок	логически: содержит ли файл .csv строку заголовка с именами столбцов?
сентябрь	символ: символ, который разделяет ячейки на каждой строке
котировка	символ: символ, используемый для указания символьных строк
декабрь	символ: символ, используемый как десятичный разделитель
заполнить	логическое: если TRUE, строки с неравной длиной заполняются пустым полем.
comment.char	character: character используется как комментарий в файле csv. Строки, которым предшествует этот символ, игнорируются.

параметр	подробности
...	дополнительные аргументы, которые необходимо передать `read.table`

замечания

Обратите внимание, что экспорт в текстовый формат жертвует большей частью информации, закодированной в таких данных, как переменные классы, для широкой переносимости. Для случаев, которые не требуют такой переносимости, формат, например .RData или Feather, может быть более полезным.

Ввод / вывод для других типов файлов описывается несколькими другими темами, связанными с входом и выходом .

Examples

Импорт файлов .csv

Импорт с использованием базы R

Файлы значений с разделителями-запятыми (CSV) можно импортировать с помощью `read.csv` , который обертывает `read.table` , но использует `sep = ","` чтобы установить разделитель в запятую.

```
# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

df <- read.csv(csv_path)

df
##    Var1 Var2
## 1  2.70    A
## 2  3.14    B
## 3 10.00    A
## 4 -7.00    A
```

Удобный для пользователя вариант, `file.choose` , позволяет просматривать каталоги:

```
df <- read.csv(file.choose())
```

Заметки

- В отличие от `read.table` , `read.csv` умолчанию имеет значение `header = TRUE` и использует первую строку как имена столбцов.
- Все эти функции будут преобразовывать строки в класс `factor` по умолчанию, если только `as.is = TRUE` или `stringsAsFactors = FALSE` .
- Вариант `read.csv2` умолчанию имеет значение `sep = ";"` и `dec = ","` для использования в данных из стран, где запятая используется как десятичная точка и точка с запятой как разделитель полей.

Импорт с использованием пакетов

В `readr` ПАКЕТ в `read_csv` функция предлагает гораздо более высокую производительность, индикатор выполнения для больших файлов, а также более популярные опции по умолчанию , чем стандартный `read.csv` , включая `stringsAsFactors = FALSE` .

```
library(readr)

df <- read_csv(csv_path)

df
## # A tibble: 4 x 2
##    Var1  Var2
##   <dbl> <chr>
## 1  2.70     A
## 2  3.14     B
## 3 10.00     A
## 4 -7.00     A
```

Импорт с помощью data.table

Пакет `data.table` вводит функцию `fread` . Хотя он похож на `read.table` , `fread` обычно быстрее и гибче, угадывая разделитель файла автоматически.

```
# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on R installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

dt <- fread(csv_path)

dt
##     Var1 Var2
## 1:  2.70    A
## 2:  3.14    B
## 3: 10.00    A
## 4: -7.00    A
```

Где `input` аргумента представляет собой строку, представляющую:

- имя файла (*например,* `"filename.csv"`),
- команда оболочки, которая действует на файл (*например,* `"grep 'word' filename"`) или
- сам вход (*например,* `"input1, input2 \n A, B \n C, D"`).

`fread` возвращает объект класса `data.table` который наследует класс `data.frame` , подходящий для использования с использованием data.table `[]` . Чтобы вернуть обычный data.frame, установите для параметра `data.table` значение `FALSE` :

```
df <- fread(csv_path, data.table = FALSE)

class(df)
## [1] "data.frame"

df
##    Var1 Var2
## 1  2.70    A
## 2  3.14    B
## 3 10.00    A
## 4 -7.00    A
```

Заметки

- `fread` не имеет всех параметров, таких как `read.table` . Один отсутствующий аргумент - `na.comment` , что может привести к нежелательному поведению, если исходный файл содержит `#` .
- `fread` использует только `"` для параметра `quote` .
- `fread` использует несколько (5) строк для определения типов переменных.

Импорт файлов .tsv в виде матриц (базовый R)

Многие люди не используют `file.path` при создании пути к файлу. Но если вы работаете на машинах Windows, Mac и Linux, обычно рекомендуется использовать их для создания путей вместо `paste` .

```
FilePath <- file.path(AVariableWithFullProjectPath,"SomeSubfolder","SomeFileName.txt.gz")

Data <- as.matrix(read.table(FilePath, header=FALSE, sep ="\t"))
```

Обычно этого достаточно для большинства людей.

Иногда бывает, что размеры матрицы настолько велики, что при расчете в матрице необходимо учитывать процедуру распределения памяти, что означает чтение в матрице по строкам.

Возьмем предыдущий пример. В этом случае `FilePath` содержит файл размером `8970 8970` с 79% ячеек, содержащих ненулевые значения.

```
system.time(expr=Data<-as.matrix(read.table(file=FilePath,header=FALSE,sep=" ") ))
```

`system.time` говорит, что 267 секунд были взяты для чтения файла.

```
   user  system elapsed
265.563   1.949 267.563
```

Аналогично, этот файл можно читать по строкам,

```
FilePath <- "SomeFile"
connection<- gzfile(FilePath,open="r")
TableList <- list()
Counter <- 1
system.time(expr= while ( length( Vector<-as.matrix(scan(file=connection, sep=" ", nlines=1,
quiet=TRUE)) ) > 0 ) {
    TableList[[Counter]]<-Vector
    Counter<-Counter+1
})
   user  system elapsed
165.976   0.060 165.941
close(connection)
system.time(expr=(Data <- do.call(rbind,TableList)))
   user  system elapsed
  0.477   0.088   0.565
```

Существует также пакет `futile.matrix` , который реализует метод `read.matrix` , сам код будет показывать то же самое, что описано в примере 1.

Экспорт файлов .csv

Экспорт с использованием базы R

Данные могут быть записаны в CSV-файл с помощью `write.csv()` :

```
write.csv(mtcars, "mtcars.csv")
```

Обычно заданные параметры включают `row.names = FALSE` и `na = ""` .

Экспорт с использованием пакетов

`readr::write_csv` значительно быстрее, чем `write.csv` и не записывает имена строк.

```
library(readr)

write_csv(mtcars, "mtcars.csv")
```

Импортировать несколько файлов csv

```
files = list.files(pattern="*.csv")
data_list = lapply(files, read.table, header = TRUE)
```

Это читает каждый файл и добавляет его в список. Впоследствии, если все data.frame имеют одинаковую структуру, их можно объединить в один большой файл data.frame:

```
df <- do.call(rbind, data_list)
```

Импорт файлов фиксированной ширины

Файлы с фиксированной шириной - это текстовые файлы, в которых столбцы не разделены каким-либо разделителем символов, например , или ; , но имеют фиксированную длину символа (*ширину*). Обычно данные заполняются пробелами.

Пример:

```
Column1 Column2   Column3           Column4Column5
1647    pi        'important'       3.141596.28318
1731    euler     'quite important' 2.718285.43656
1979    answer    'The Answer.'     42     42
```

Предположим, что эта таблица данных существует в локальном файле `constants.txt` в рабочем каталоге.

Импорт с базой R

```
df <- read.fwf('constants.txt', widths = c(8,10,18,7,8), header = FALSE, skip = 1)

df
#>     V1     V2                 V3         V4        V5
#> 1 1647     pi          'important'   3.14159   6.28318
#> 2 1731  euler   'quite important'   2.71828   5.43656
#> 3 1979 answer       'The Answer.'   42        42.0000
```

Замечания:

- Названия столбцов не должны разделяться символом (`Column4Column5`)
- Параметр `widths` определяет ширину каждого столбца
- Не разделенные заголовки не читаются с помощью `read.fwf()`

Импорт с помощью readr

```
library(readr)

df <- read_fwf('constants.txt',
               fwf_cols(Year = 8, Name = 10, Importance = 18, Value = 7, Doubled = 8),
               skip = 1)
df
#> # A tibble: 3 x 5
#>    Year    Name        Importance    Value  Doubled
#>    <int>   <chr>           <chr>     <dbl>    <dbl>
#> 1  1647      pi       'important'  3.14159  6.28318
#> 2  1731   euler 'quite important'  2.71828  5.43656
#> 3  1979  answer     'The Answer.' 42.00000 42.00000
```

Замечания:

- Функции readw's `fwf_*` helper предоставляют альтернативные способы указания длины столбцов, включая автоматическое угадывание (`fwf_empty`)
- readr быстрее, чем база R
- Заголовки столбцов не могут быть автоматически импортированы из файла данных

Прочитайте Чтение и запись табличных данных в текстовых файлах (CSV, TSV и т. Д.) онлайн: https://riptutorial.com/ru/r/topic/481/чтение-и-запись-табличных-данных-в-текстовых-файлах--csv--tsv-и-т--д--

кредиты

S. No	Главы	Contributors
1	Начало работы с R Language	42-, akraf, Ale, Andrea Cirillo, Andrew Brēza, Axeman, Community, Craig Vermeer, d.b, dotancohen, Francesco Dondi, Frank, G5W, George Bonebright, GForce, Giorgos K, Gregor, H. Pauwelyn, kartoffelsalat, kdopen, Konrad Rudolph, L.V.Rao, lmckeogh, Lovy, Matt, mnoronha, pitosalas, polka, Rahul Saini, RetractedAndRetired, russellpierce, Steve_Corrin, theArun, Thomas, torina, user2100721, while
2	* применять семейство функций (функционалов)	Benjamin, FisherDisinformation, Gavin Simpson, jcb, Karolis Koncevičius, kneijenhuijs, Maximilian Kohl, nrussell, omar, Robert, seasmith, zacdav
3	.Rprofile	42-, Dirk Eddelbuettel, ikashnitsky, Karolis Koncevičius, Nikos Alexandris, Stedy, Thomas
4	ANOVA	Ben Bolker, DataTx, kneijenhuijs
5	boxplot	Carlos Cinelli, Christophe D., Karolis Koncevičius, L.V.Rao
6	dplyr	4444, Alihan Zıhna, ikashnitsky, Robert, skoh, Sumedh, theArun
7	ggplot2	akraf, Alex, alistaire, Andrea Cirillo, Artem Klevtsov, Axeman, baptiste, blmoore, Boern, gitblame, ikashnitsky, Jaap, jmax, loki , Matt, Mine Cetinkaya-Rundel, Paolo, smci, Steve_Corrin, Sumedh, Taylor Ostberg, theArun, void, YCR, Yun Ching
8	GPU-ускоренные вычисления	cdeterman
9	HashMaps	nrussell, russellpierce
10	I / O для внешних таблиц (Excel, SAS, SPSS, Stata)	42-, Alex, alistaire, Andrea Cirillo, Carlos Cinelli, Charmgoggles , Crops, Frank, Jaap, Jeromy Anglim, kaksat, Ken S., kitman0804, lmo, Miha, Parfait, polka, Thomas
11	I / O для географических данных (шейп-файлы и т. Д.)	Alex, Frank, ikashnitsky

12	I / O для двоичного формата R	Frank, ikashnitsky, Mario, russellpierce, zacdav, zx8754
13	I / O для таблиц базы данных	Frank, JHowIX, SommerEngineering
14	JSON	SymbolixAU
15	lubridate	alistaire, Angelo, Frank, gitblame, Hendrik, scoa
16	Meta: Руководство по документации	Frank, Gregor, Stephen Leppik, Steve_Corrin
17	R Markdown Notebooks (от RStudio)	dmail
18	R в LaTeX с knitr	JHowIX
19	R примера по примерам	Lovy
20	Rcpp	Artem Klevtsov, coatless, Dirk Eddelbuettel
21	RODBC	akrun, Hack-R, Parfait, Tim Coker
22	roxygen2	DeveauP, PAC
23	Spark API (SparkR)	Maximilian Kohl
24	sqldf	Hack-R, Miha
25	tidyverse	David Robinson, egnha, Frank, ikashnitsky, RamenChef, Sumedh
26	xgboost	Hack-R
27	Агрегирование кадров данных	Florian, Frank
28	Алгоритм случайного леса	G5W
29	Анализ выживаемости	42-, Axeman, Hack-R, Marcin Kosiński
30	Анализ твитов с R	Umberto
31	Арифметические	Batanichek, FisherDisinformation, Matt Sandgren, Robert,

	операторы	russellpierce, Tensibai
32	Барная диаграмма	L.V.Rao
33	Библиография в RMD	J_F, RamenChef
34	блестящий	alistaire, CClaire, Christophe D., JvH, russellpierce, SymbolixAU, tuomastik, zx8754
35	Введение в географические карты	4444, AkselA, alistaire, beetroot, Carson, Frank, Hack-R, HypnoGenX, Robert, russellpierce, SymbolixAU, symbolrush
36	Веб-сканирование в R	Pankaj Sharma
37	Вероятностные распределения с R	Pankaj Sharma
38	Внедрить шаблон государственного устройства с использованием класса S4	David Leal
39	Воспроизводимые R	Charmgoggles, Frank, ikashnitsky
40	Временные ряды и прогнозирование	Andras Deak, Andrew Bryk, coatless, Hack-R, JGreenwell, Pankaj Sharma, Steve_Corrin, µ Muthupandian
41	Вход и выход	Frank
42	Вход пользователя	Ashish, DeveauP
43	Входы / выходы для растровых изображений	Frank, loki
44	Выбор функции в R - Удаление внешних функций	Joy
45	Выполнение теста перестановки	Stephen Leppik, tenCupMaximum
46	Выражение: parse	YCR

	+ eval	
47	Генератор случайных чисел	bartektartanus, FisherDisinformation, Karolis Koncevičius, Miha , mnoronha
48	Дата и время	AkselA, alistaire, Angelo, coatless, David Leal, Dean MacGregor, Frank, kneijenhuijs, MichaelChirico, scoa, SymbolixAU, takje, theArun, thelatemail
49	знак вставки	highBandWidth, Steve_Corrin
50	Иерархическая кластеризация с hclust	Frank, G5W, Tal Galili
51	Иерархическое линейное моделирование	Ben Bolker
52	Извлечение и листинг файлов в сжатых архивах	catastrophic-failure, Jeff
53	Издательский	Frank
54	Изменение строк путем замены	Alex, David Leal, Frank
55	Изменить форму tidyr	Charmgoggles, Frank, Jeromy Anglim, SymbolixAU, user2100721
56	интроспекция	Jason
57	Использование texreg для экспорта моделей в бумажном виде	Frank, ikashnitsky
58	Использование назначения труб в вашем собственном пакете% <>%: Как?	RobertMc
59	Кадры данных	Alex, Andrea Ianni , Batanichek, Carlos Cinelli, Christophe D., DataTx, David Arenburg, David Robinson, dayne, Frank, Gregor, Hack-R, kaksat, R. Schifini, scoa, Sumedh, Thomas, Tomás Barcellos, user2100721

60	Класс Date	alistaire, coatless, Frank, L.V.Rao, MichaelChirico, Steve_Corrin
61	Класс символов	Frank, Steve_Corrin
62	Классы	42-, AkselA, David Heckmann, dayne, Frank, Gregor, Jaap, kneijenhuijs, L.V.Rao, Nathan Werth, Steve_Corrin
63	Классы времени (POSIXct и POSIXlt)	AkselA, alistaire, coatless, Frank, MichaelChirico, SymbolixAU, thelatemail
64	Кодировка длины	Frank, josliber, Psidom
65	Комбинаторика	Frank, Karolis Koncevičius
66	Линейные модели (регрессия)	Amstell, Ben Bolker, Carl, Carlos Cinelli, David Robinson, fortune_p, Frank, highBandWidth, ikashnitsky, jaySf, Robert, russellpierce, thelatemail, USER_1, WAF
67	Логический класс	42-, Frank, Gregor, L.V.Rao, Steve_Corrin
68	Матрицы	dayne, Frank
69	Машинное обучение	loki
70	Модели Arima	Andrew Bryk, Steve_Corrin
71	Нестандартная оценка и стандартная оценка	PAC
72	Обновление R и библиотеки пакетов	Eric Lecoutre
73	Обновление версии R	dmail
74	Обобщенные линейные модели	Ben Bolker, YCR
75	Обработка естественного языка	CptNemo
76	Обработка строк со	bartektartanus, FisherDisinformation

	строковым пакетом	
77	Объектно-ориентированное программирование в R	Jon Ericson, rcorty
78	Объем переменных	Artem Klevtsov, K.Daisey, RamenChef
79	Операторы труб (%>% и другие)	42-, Alexandru Papiu, Alihan Zıhna, alistaire, AndreyAkinshin, Artem Klevtsov, Atish, Axeman, Benjamin, Carlos Cinelli, CMichael, DrPositron, Franck Dernoncourt, Frank, Gal Dreiman , Gavin Simpson, Gregor, ikashnitsky, James McCalden, Kay Brodersen, Matt, polka, RamenChef, Ryan Hilbert, Sam Firke, seasmith, Shawn Mehan, Simplans, Spacedman, SymbolixAU, thelatemail, tomw, TriskalJM, user2100721
80	Основание	42-, Alexey Shiklomanov, catastrophic-failure, FisherDisinformation, Frank, Giorgos K, K.Daisey, maRtin, MichaelChirico, RamenChef, Robert, symbolrush
81	Отказоустойчивый / устойчивый код	Rappster
82	отладка	James Elderfield, russellpierce
83	Отсутствующие значения	Amit Kohli, Artem Klevtsov, Axeman, Eric Lecoutre, Frank, Gregor, Jaap, kitman0804, lmo, seasmith, Steve_Corrin, theArun, user2100721
84	Очистка веб-страниц и разбор	alistaire, Dave2e
85	Очистка данных	Derek Corcoran
86	Параллельная обработка	Artem Klevtsov, jameselmore, K.Daisey, lmo, loki, russellpierce
87	переменные	42-, Ale, Axeman, Craig Vermeer, Frank, L.V.Rao, lmckeogh
88	Переработка отходов	Frank, USER_1
89	Перестановка данных между длинными и широкими формами	Charmgoggles, David Arenburg, demonplus, Frank, Jeromy Anglim, kneijenhuijs, lmo, Steve_Corrin, SymbolixAU, takje, user2100721, zx8754

90	Плавный и невольный с data.table	Sun Bee
91	Подменю	42-, Agriculturist, alexis_laz, alistaire, dayne, Frank, Gavin Simpson, Gregor, L.V.Rao, Mario, mrip, RamenChef, smci, user2100721, zx8754
92	Получение данных	ikashnitsky
93	Презентация RMarkdown и knitr	Martin Schmelzer, YCR
94	принуждение	d.b
95	Проверка пакетов	Frank, Sowmya S. Manian
96	пространственный анализ	beetroot, ikashnitsky, loki, maRtin
97	Профилирование кода	Ben Bolker, Glen Moutrie, Jav, SymbolixAU, USER_1
98	Работа с колонкой	akrun
99	Растровый и графический анализ	Frank, loki
100	Регулярные выражения (регулярное выражение)	42-, Benjamin, David Leal, etienne, Frank, MichaelChirico, PAC
101	Рекомендации по векторизации кода R	Axeman, David Arenburg, snaut
102	Решение ODE в R	J_F
103	Серия Фурье и преобразования	Hack-R
104	Сетевой анализ с пакетом igraph	Boysenb3rry
105	Синтаксис регулярного	Alexey Shiklomanov

	выражения в R	
106	Случайность	TARehman
107	Согласование и замена шаблонов	Abdou, Alex, Artem Klevtsov, David Arenburg, David Leal, Frank, Gavin Simpson, Jaap, NWaters, R. Schifini, SommerEngineering, Steve_Corrin, Tensibai, thelatemail, user2100721
108	Создание векторов	alistaire, bartektartanus, Jaap, Karsten W., lmo, Rich Scriven, Robert, Robin Gertenbach, smci, takje
109	Создание отчетов с помощью RMarkdown	ikashnitsky, Karolis Koncevičius, Martin Schmelzer
110	Создание пакетов с помощью devtools	Frank, Lovy
111	Списки	Andrea Ianni , BarkleyBG, dayne, Frank, Hack-R, Hairizuan Noorazman, Peter Humburg, RamenChef
112	Стандартизировать анализ путем написания автономных R-скриптов	akraf, herbaman
113	Структуры управляющих потоков	Benjamin, David Arenburg, nrussell, Robert, Steve_Corrin
114	Таблица данных	akrun, Allen Wang, bartektartanus, cderv, David, David Arenburg, Dean MacGregor, Eric Lecoutre, Frank, Jaap, jogo, L Co, leogama, Mallick Hossain, micstr, Nathan Werth, oshun, Peter Humburg, Sowmya S. Manian, stanekam, Steve_Corrin, Sumedh, Tensibai, user2100721, Uwe
115	Текстовая обработка	Hack-R
116	тепловая карта и тепловая карта.2	AndreyAkinshin, Nanami
117	Ускорение жестко-векторизованного кода	egnha, josliber

118	Услуги RESTful R	YCR
119	Установить операции	DeveauP, FisherDisinformation, Frank
120	Установка пакетов	Aaghaz Hussain, akraf, alko989, Andrew Brēza, Artem Klevtsov, Arun Balakrishnan, Christophe D., CL., Frank, gitblame, Hack-R, hongsy, Jaap, kaksat, kneijenhuijs, lmckeogh, loki, Marc Brinkmann, Miha, Peter Humburg, Pragyaditya Das, Raj Padmanabhan, seasmith, SymbolixAU, theArun, user890739, xamgore, zx8754
121	факторы	42-, Benjamin, dash2, Frank, Gavin Simpson, JulioSergio, kneijenhuijs, Nathan Werth, omar, Rich Scriven, Robert, Steve_Corrin
122	формула	42-, Axeman, Qaswed, Sathish
123	Функции записи в R	AkselA, ikashnitsky, kaksat
124	Функции распределения	FisherDisinformation, Frank, L.V.Rao, tenCupMaximum
125	Функциональное программирование	Karolis Koncevičius
126	Функция strsplit	lmo
127	Функция разделения	Eric Lecoutre, etienne, josliber, Sathish, Tensibai, thelatemail, user2100721
128	Цветовые схемы для графики	ikashnitsky, munirbe
129	Числовые классы и режимы хранения	Frank, Steve_Corrin
130	Чтение и запись строк	42-, 4444, abhiieor, cdrini, dotancohen, Frank, Gregor, kdopen, Rich Scriven, Thomas, Uwe
131	Чтение и запись табличных данных в текстовых файлах (CSV, TSV и т. Д.)	a.powell, Aaghaz Hussain, abhiieor, Alex, alistaire, Andrea Cirillo, bartektartanus, Carl Witthoft, Carlos Cinelli, catastrophic-failure, cdrini, Charmgoggles, Crops, DaveRGP, David Arenburg, Dawny33, Derwin McGeary, EDi, Eric Lecoutre, FoldedChromatin, Frank, Gavin Simpson, gitblame, Hairizuan Noorazman, herbaman, ikashnitsky, Jaap, Jeromy Anglim, JHowIX, joeyreid, Jordan Kassof, K.Daisey, kitman0804,

		kneijenhuijs, lmo, loki, Miha, PAC, polka, russellpierce, Sam Firke, stats-hb, Thomas, Uwe, zacdav, zelite, zx8754