Бесплатная электронная книга

учусь pandas

Free unaffiliated eBook created from
Stack Overflow contributors.

#pandas

Около

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

It is an unofficial and free pandas 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 pandas.

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 пандами

замечания

Pandas - это пакет Python, обеспечивающий быструю, гибкую и выразительную структуру данных, предназначенную для работы с «реляционными» или «помеченными» данными как простой, так и интуитивно понятной. Он призван стать фундаментальным строительным блоком высокого уровня для практического анализа данных реального мира в Python.

Официальную документацию Pandas можно найти здесь .

Версии

Панды

Версия	Дата выхода
0.19.1	2016-11-03
0.19.0	2016-10-02
0.18.1	2016-05-03
0.18.0	2016-03-13
0.17.1	2015-11-21
0.17.0	2015-10-09
0.16.2	2015-06-12
0.16.1	2015-05-11
0.16.0	2015-03-22
0.15.2	2014-12-12
0.15.1	2014-11-09
0.15.0	2014-10-18
0.14.1	2014-07-11
0.14.0	2014-05-31
0.13.1	2014-02-03

Версия	Дата выхода
0.13.0	2014-01-03
0.12.0	2013-07-23

Examples

Установка или настройка

Подробные инструкции по установке или установке панд можно найти здесь, в официальной документации .

Установка панд с помощью Anaconda

Установка pandas и остальной части стека NumPy и SciPy может быть немного сложной для неопытных пользователей.

Самый простой способ установить не только pandas, но и Python и самые популярные пакеты, составляющие стек SciPy (IPython, NumPy, Matplotlib, ...), - это Anaconda , кросс-платформенная (Linux, Mac OS X, Windows) Распределение Python для анализа данных и научных вычислений.

После запуска простого установщика пользователь получит доступ к pandas и остальной части стека SciPy без необходимости устанавливать что-либо еще и без необходимости компилировать какое-либо программное обеспечение.

Инструкции по установке для Anaconda можно найти здесь .

Полный список пакетов, доступных в составе дистрибутива Anaconda, можно найти здесь .

Дополнительным преимуществом установки с Anaconda является то, что вам не требуются права администратора для его установки, она будет установлена в домашнем каталоге пользователя, и это также упростит удаление Anaconda на более позднюю дату (просто удалите эту папку).

Установка панд с помощью Miniconda

В предыдущем разделе описано, как установить pandas как часть дистрибутива Anaconda. Однако этот подход означает, что вы установите более ста пакетов и загрузите установщик размером в несколько сотен мегабайт.

Если вы хотите иметь больше контроля над пакетами или иметь ограниченную пропускную способность Интернета, то установка pandas с помощью Miniconda может быть лучшим решением.

Конда - это менеджер пакетов, на котором основан дистрибутив Anaconda. Это менеджер пакетов, который является как межплатформенным, так и языковым агностиком (он может играть аналогичную роль в сочетании с pip и virtualenv).

Miniconda позволяет создавать минимальную автономную установку Python, а затем использовать команду Conda для установки дополнительных пакетов.

Сначала вам понадобится Conda для установки, и загрузка и запуск Miniconda сделает это за вас. Установщик можно найти здесь .

Следующий шаг - создать новую среду conda (они аналогичны виртуальным, но они также позволяют точно указать, какую версию Python также установить). Выполните следующие команды из окна терминала:

```
conda create -n name_of_my_env python
```

Это создаст минимальную среду, в которой будет установлен только Python. Чтобы запустить себя в эту среду, выполните следующие действия:

```
source activate name_of_my_env
```

В Windows команда:

```
activate name_of_my_env
```

Последний шаг - установка панд. Это можно сделать с помощью следующей команды:

```
conda install pandas
```

Чтобы установить конкретную версию pandas:

```
conda install pandas=0.13.1
```

Чтобы установить другие пакеты, IPython, например:

```
conda install ipython
```

Чтобы установить полный дистрибутив Anaconda:

```
conda install anaconda
```

Если вам нужны пакеты, доступные для pip, но не conda, просто установите pip и используйте pip для установки этих пакетов:

```
conda install pip
pip install django
```

Обычно вы устанавливаете панды с одним из менеджеров пакетов.

Пример примера:

```
pip install pandas
```

Это, скорее всего, потребует установки ряда зависимостей, в том числе NumPy, потребует от компилятора компиляции необходимых битов кода и может занять несколько минут.

Установить через anaconda

Сначала загрузите anaconda с сайта Continuum. Либо через графический установщик (Windows / OSX), либо запустите сценарий оболочки (OSX / Linux). Сюда входят панды!

Если вы не хотите, чтобы 150 пакетов были в комплекте в анаконде, вы можете установить миниконду . Либо через графический установщик (Windows), либо скрипт оболочки (OSX / Linux).

Установите pandas на miniconda, используя:

```
conda install pandas
```

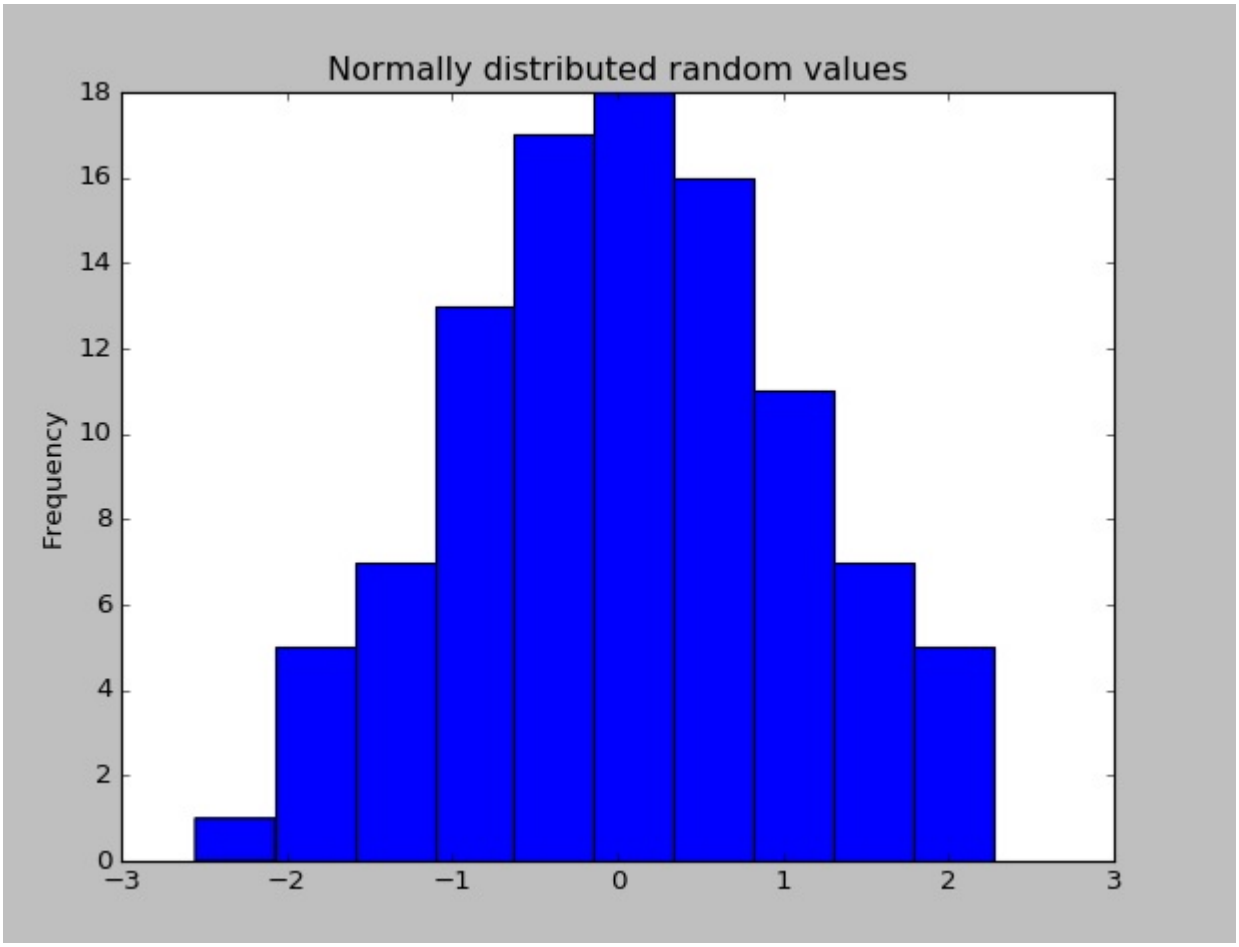
Чтобы обновить pandas до последней версии в anaconda или miniconda, используйте:

```
conda update pandas
```

Привет, мир

Как только Pandas будет установлен, вы можете проверить, работает ли он правильно, создав набор данных случайным образом распределенных значений и построив его гистограмму.

```
import pandas as pd  # This is always assumed but is included here as an introduction.
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(0)

values = np.random.randn(100) # array of normally distributed random numbers
s = pd.Series(values) # generate a pandas series
s.plot(kind='hist', title='Normally distributed random values') # hist computes distribution
plt.show()
```



Проверьте некоторые данные статистики (среднее значение, стандартное отклонение и т. Д.).

```
s.describe()
# Output: count    100.000000
# mean       0.059808
# std        1.012960
# min       -2.552990
# 25%       -0.643857
# 50%        0.094096
# 75%        0.737077
# max        2.269755
# dtype: float64
```

Описательная статистика

Описательная статистика (среднее, стандартное отклонение, количество наблюдений, минимальное, максимальное и квартили) числовых столбцов может быть рассчитана с использованием `.describe()` , который возвращает `.describe()` pandas описательной статистики.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 1, 4, 3, 5, 2, 3, 4, 1],
                           'B': [12, 14, 11, 16, 18, 18, 22, 13, 21, 17],
                           'C': ['a', 'a', 'b', 'a', 'b', 'c', 'b', 'a', 'b', 'a']})

In [2]: df
```

```
Out[2]:
   A   B  C
0  1  12  a
1  2  14  a
2  1  11  b
3  4  16  a
4  3  18  b
5  5  18  c
6  2  22  b
7  3  13  a
8  4  21  b
9  1  17  a

In [3]: df.describe()
Out[3]:
                A          B
count  10.000000  10.000000
mean    2.600000  16.200000
std     1.429841   3.705851
min     1.000000  11.000000
25%     1.250000  13.250000
50%     2.500000  16.500000
75%     3.750000  18.000000
max     5.000000  22.000000
```

Заметим, что поскольку `C` не является числовым столбцом, он исключается из вывода.

```
In [4]: df['C'].describe()
Out[4]:
count     10
unique     3
freq       5
Name: C, dtype: object
```

В этом случае метод суммирует категориальные данные по количеству наблюдений, количеству уникальных элементов, режиму и частоте режима.

Прочитайте Начало работы с пандами онлайн: https://riptutorial.com/ru/pandas/topic/796/начало-работы-с-пандами

глава 2: Gotchas of pandas

замечания

Gotcha вообще является конструкцией, которая хотя и документирована, но не интуитивно понятна. Gotchas производят некоторый результат, который обычно не ожидается из-за его встречно-интуитивного характера.

В пакете Pandas есть несколько gotchas, которые могут смутить кого-то, кто не знает о них, и некоторые из них представлены на этой странице документации.

Examples

Обнаружение отсутствующих значений с помощью np.nan

Если вы хотите обнаружить пропуски с

```
df=pd.DataFrame({'col':[1,np.nan]})
df==np.nan
```

вы получите следующий результат:

```
col
0    False
1    False
```

Это связано с тем, что сравнение отсутствующего значения с чем-либо приводит к False - вместо этого вы должны использовать

```
df=pd.DataFrame({'col':[1,np.nan]})
df.isnull()
```

что приводит к:

```
col
0    False
1    True
```

Integer и NA

Панды не поддерживают отсутствие атрибутов типа integer. Например, если у вас есть пропуски в столбце класса:

```
df= pd.read_csv("data.csv", dtype={'grade': int})
```

```
error: Integer column has NA values
```

В этом случае вам просто нужно использовать float вместо целых чисел или установить объект dtype.

Автоматическое выравнивание данных (поведение с учетом индекса)

Если вы хотите добавить серию значений [1,2] в столбец dataframe df, вы получите NaNs:

```
import pandas as pd

series=pd.Series([1,2])
df=pd.DataFrame(index=[3,4])
df['col']=series
df

   col
3    NaN
4    NaN
```

потому что установка нового столбца автоматически выравнивает данные индексом, а ваши значения 1 и 2 будут получать индексы 0 и 1, а не 3 и 4, как в вашем кадре данных:

```
df=pd.DataFrame(index=[1,2])
df['col']=series
df

   col
1      2.0
2      NaN
```

Если вы хотите игнорировать индекс, вы должны установить значения в конце:

```
df['col']=series.values

   col
3    1
4    2
```

Прочитайте Gotchas of pandas онлайн: https://riptutorial.com/ru/pandas/topic/6425/gotchas-of-pandas

глава 3: IO для Google BigQuery

Examples

Чтение данных из BigQuery с учетными данными учетной записи пользователя

```
In [1]: import pandas as pd
```

Чтобы выполнить запрос в BigQuery, вам нужно иметь собственный проект BigQuery. Мы можем запросить некоторые общедоступные данные выборки:

```
In [2]: data = pd.read_gbq('''SELECT title, id, num_characters
   ...:                       FROM [publicdata:samples.wikipedia]
   ...:                       LIMIT 5'''
   ...:                     , project_id='<your-project-id>')
```

Это напечатает:

```
Your browser has been opened to visit:

    https://accounts.google.com/o/oauth2/v2/auth...[looong url cutted]

If your browser is on a different machine then exit and re-run this
application with the command-line parameter

  --noauth_local_webserver
```

Если вы работаете с локальной машины, то вы можете всплывать в браузере. После предоставления привилегий панды будут продолжать выпуск:

```
Authentication successful.
Requesting query... ok.
Query running...
Query done.
Processed: 13.8 Gb

Retrieving results...
Got 5 rows.

Total time taken 1.5 s.
Finished at 2016-08-23 11:26:03.
```

Результат:

```
In [3]: data
Out[3]:
               title       id  num_characters
0       Fusidic acid   935328            1112
```

```
1     Clark Air Base   426241            8257
2  Watergate scandal    52382           25790
3               2005    35984           75813
4               .BLP  2664340            1659
```

В качестве побочного эффекта pandas создаст json-файл `bigquery_credentials.dat` который позволит вам запускать дополнительные запросы без необходимости предоставления привилегий:

```
In [9]: pd.read_gbq('SELECT count(1) cnt FROM [publicdata:samples.wikipedia]'
                    , project_id='<your-project-id>')
Requesting query... ok.
[rest of output cutted]

Out[9]:
         cnt
0  313797035
```

Чтение данных из BigQuery с учетными данными учетной записи службы

Если вы создали учетную запись службы и для нее есть файл json для частного ключа, вы можете использовать этот файл для аутентификации с помощью pandas

```
In [5]: pd.read_gbq('''SELECT corpus, sum(word_count) words
                       FROM [bigquery-public-data:samples.shakespeare]
                       GROUP BY corpus
                       ORDER BY words desc
                       LIMIT 5'''
                    , project_id='<your-project-id>'
                    , private_key='<private key json contents or file path>')
Requesting query... ok.
[rest of output cutted]

Out[5]:
           corpus  words
0          hamlet  32446
1  kingrichardiii  31868
2      coriolanus  29535
3       cymbeline  29231
4    2kinghenryiv  28241
```

Прочитайте IO для Google BigQuery онлайн: https://riptutorial.com/ru/pandas/topic/5610/io-для-google-bigquery

глава 4: JSON

Examples

Читать JSON

может либо передать строку json, либо путь к файлу с действительным json

```
In [99]: pd.read_json('[{"A": 1, "B": 2}, {"A": 3, "B": 4}]')
Out[99]:
   A  B
0  1  2
1  3  4
```

В качестве альтернативы для сохранения памяти:

```
with open('test.json') as f:
    data = pd.DataFrame(json.loads(line) for line in f)
```

Dataframe во вложенный JSON, как в файлах flare.js, используемых в D3.js

```
def to_flare_json(df, filename):
    """Convert dataframe into nested JSON as in flare files used for D3.js"""
    flare = dict()
    d = {"name":"flare", "children": []}

    for index, row in df.iterrows():
        parent = row[0]
        child = row[1]
        child_size = row[2]

        # Make a list of keys
        key_list = []
        for item in d['children']:
            key_list.append(item['name'])

        #if 'parent' is NOT a key in flare.JSON, append it
        if not parent in key_list:
            d['children'].append({"name": parent, "children":[{"value": child_size, "name":
child}]})
        # if parent IS a key in flare.json, add a new child to it
        else:
            d['children'][key_list.index(parent)]['children'].append({"value": child_size,
"name": child})
    flare = d
    # export the final result to a json file
```

```
    with open(filename +'.json', 'w') as outfile:
        json.dump(flare, outfile, indent=4)
    return ("Done")
```

Чтение JSON из файла

Содержимое файла file.json (один объект JSON в строке):

```
{"A": 1, "B": 2}
{"A": 3, "B": 4}
```

Как читать непосредственно из локального файла:

```
pd.read_json('file.json', lines=True)
# Output:
#    A  B
# 0  1  2
# 1  3  4
```

Прочитайте JSON онлайн: https://riptutorial.com/ru/pandas/topic/4752/json

глава 5: Meta: Руководство по документации

замечания

Эта мета-запись похожа на версию python http://stackoverflow.com/documentation/python/394/meta-documentation-guidelines#t=20160724005840635952 1 .

Просьба внести предложения по редактированию и прокомментировать их (вместо правильных комментариев), чтобы мы могли выполнить / повторить эти предложения :)

Examples

Отображение фрагментов кода и вывода

Два популярных варианта использования:

Обозначение ipython:

```
In [11]: df = pd.DataFrame([[1, 2], [3, 4]])

In [12]: df
Out[12]:
   0  1
0  1  2
1  3  4
```

Альтернативно (это популярно в документации python) и более кратко:

```
df.columns  # Out: RangeIndex(start=0, stop=2, step=1)

df[0]
# Out:
# 0    1
# 1    3
# Name: 0, dtype: int64

for col in df:
    print(col)
# prints:
# 0
# 1
```

Как правило, это лучше для небольших примеров.

Примечание. Различие между выводом и печатью. ipython делает это ясным (отпечатки

происходят до вывода вывода):

```
In [21]: [print(col) for col in df]
0
1
Out[21]: [None, None]
```

стиль

Используйте библиотеку pandas как `pd` , это можно предположить (импорт не обязательно должен быть в каждом примере)

```
import pandas as pd
```

PEP8!

- 4 отпечатка в пространстве
- kwargs не должны использовать пробелы `f(a=1)`
- 80 символов (вся строка, связанная с предоставленным фрагментом кода, должна быть строго предпочтительной)

Поддержка версии Pandas

Большинство примеров будут работать в нескольких версиях, если вы используете «новую» функцию, которую вы должны указать, когда это было введено.

Пример: `sort_values` .

печать заявлений

Большую часть времени следует избегать, так как это может быть отвлечение (следует предпочесть Out Out).
То есть:

```
a
# Out: 1
```

всегда лучше, чем

```
print(a)
# prints: 1
```

Предпочитают поддерживать python 2 и 3:

```
print(x)    # yes! (works same in python 2 and 3)
print x     # no! (python 2 only)
```

```
print(x, y) # no! (works differently in python 2 and 3)
```

Прочитайте Meta: Руководство по документации онлайн:
https://riptutorial.com/ru/pandas/topic/3253/meta--руководство-по-документации

глава 6: Pandas Datareader

замечания

Pandas datareader - это дополнительный пакет, который позволяет создавать данные из различных интернет-источников данных, в том числе:

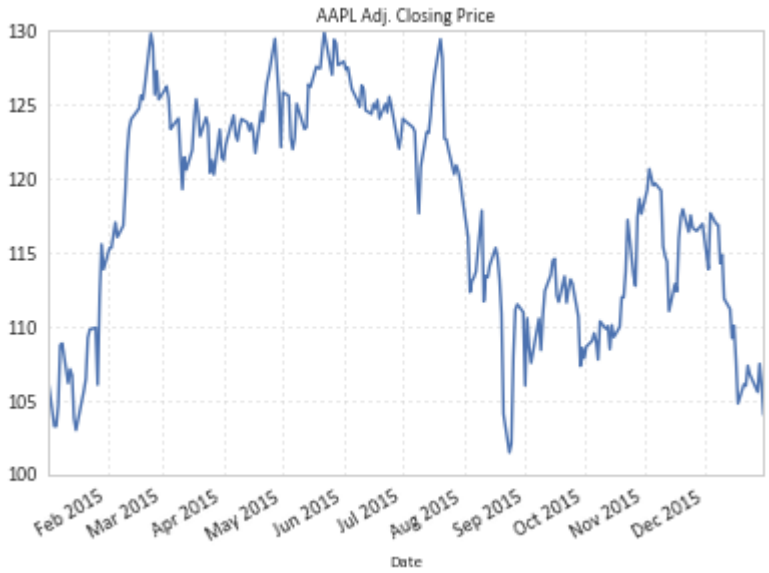
- Yahoo! финансов
- Google Finance
- St.Louis FED (FRED)
- Библиотека данных Кеннета Франца
- Всемирный банк
- Гугл Аналитика

Для получения дополнительной информации см. Здесь .

Examples

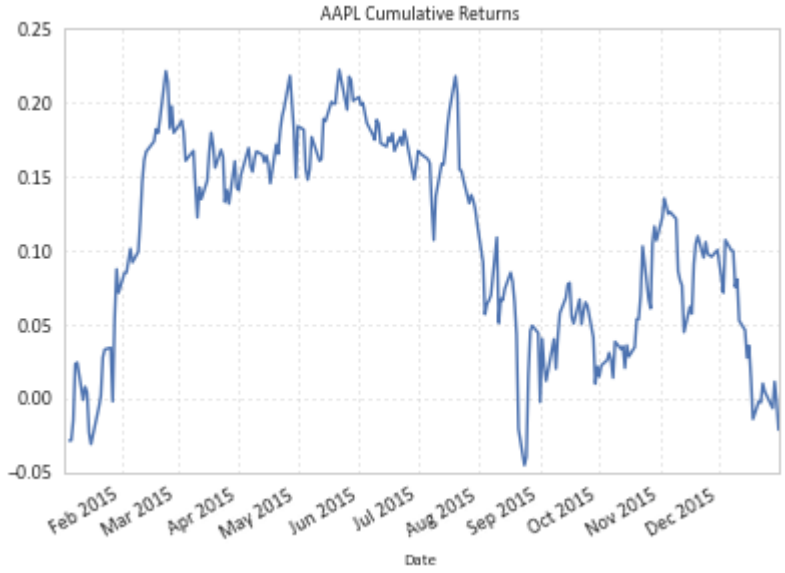
Основной пример Datareader (Yahoo Finance)

```
from pandas_datareader import data

# Only get the adjusted close.
aapl = data.DataReader("AAPL",
                       start='2015-1-1',
                       end='2015-12-31',
                       data_source='yahoo')['Adj Close']

>>> aapl.plot(title='AAPL Adj. Closing Price')
```




```
# Convert the adjusted closing prices to cumulative returns.
```

```
returns = aapl.pct_change()
>>> ((1 + returns).cumprod() - 1).plot(title='AAPL Cumulative Returns')
```



Чтение финансовых данных (для нескольких тикеров) в панель pandas - демонстрация

```
from datetime import datetime
import pandas_datareader.data as wb

stocklist = ['AAPL','GOOG','FB','AMZN','COP']

start = datetime(2016,6,8)
end = datetime(2016,6,11)

p = wb.DataReader(stocklist, 'yahoo',start,end)
```

`p` - панель панд, с которой мы можем делать смешные вещи:

посмотрим, что у нас на нашей панели

```
In [388]: p.axes
Out[388]:
[Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], dtype='object'),
 DatetimeIndex(['2016-06-08', '2016-06-09', '2016-06-10'], dtype='datetime64[ns]',
name='Date', freq='D'),
 Index(['AAPL', 'AMZN', 'COP', 'FB', 'GOOG'], dtype='object')]

In [389]: p.keys()
Out[389]: Index(['Open', 'High', 'Low', 'Close', 'Volume', 'Adj Close'], dtype='object')
```

выбор и резка данных

```
In [390]: p['Adj Close']
Out[390]:
                AAPL        AMZN        COP          FB        GOOG
Date
```

```
2016-06-08  98.940002  726.640015  47.490002  118.389999  728.280029
2016-06-09  99.650002  727.650024  46.570000  118.559998  728.580017
2016-06-10  98.830002  717.909973  44.509998  116.620003  719.409973

In [391]: p['Volume']
Out[391]:
                AAPL       AMZN        COP          FB       GOOG
Date
2016-06-08  20812700.0  2200100.0  9596700.0  14368700.0  1582100.0
2016-06-09  26419600.0  2163100.0  5389300.0  13823400.0   985900.0
2016-06-10  31462100.0  3409500.0  8941200.0  18412700.0  1206000.0

In [394]: p[:,:,'AAPL']
Out[394]:
                Open       High        Low      Close      Volume  Adj Close
Date
2016-06-08  99.019997  99.559998  98.680000  98.940002  20812700.0  98.940002
2016-06-09  98.500000  99.989998  98.459999  99.650002  26419600.0  99.650002
2016-06-10  98.529999  99.349998  98.480003  98.830002  31462100.0  98.830002

In [395]: p[:,'2016-06-10']
Out[395]:
            Open        High         Low       Close      Volume   Adj Close
AAPL   98.529999   99.349998   98.480003   98.830002  31462100.0   98.830002
AMZN  722.349976  724.979980  714.210022  717.909973   3409500.0  717.909973
COP    45.900002   46.119999   44.259998   44.509998   8941200.0   44.509998
FB    117.540001  118.110001  116.260002  116.620003  18412700.0  116.620003
GOOG  719.469971  725.890015  716.429993  719.409973   1206000.0  719.409973
```

Прочитайте Pandas Datareader онлайн: https://riptutorial.com/ru/pandas/topic/1912/pandas-datareader

глава 7: pd.DataFrame.apply

Examples

pandas.DataFrame.apply Основное использование

Метод pandas.DataFrame.apply () используется для применения данной функции ко всему `DataFrame` --- например, вычисляя квадратный корень из каждой записи данного `DataFrame` или суммируя по каждой строке `DataFrame` чтобы возвращать `Series` .

Ниже приведен базовый пример использования этой функции:

```
# create a random DataFrame with 7 rows and 2 columns
df = pd.DataFrame(np.random.randint(0,100,size = (7,2)),
                  columns = ['fst','snd'])

>>> df
   fst  snd
0   40   94
1   58   93
2   95   95
3   88   40
4   25   27
5   62   64
6   18   92

# apply the square root function to each column:
# (this returns a DataFrame where each entry is the sqrt of the entry in df;
# setting axis=0 or axis=1 doesn't make a difference)
>>> df.apply(np.sqrt)
        fst       snd
0  6.324555  9.695360
1  7.615773  9.643651
2  9.746794  9.746794
3  9.380832  6.324555
4  5.000000  5.196152
5  7.874008  8.000000
6  4.242641  9.591663

# sum across the row (axis parameter now makes a difference):
>>> df.apply(np.sum, axis=1)
0    134
1    151
2    190
3    128
4     52
5    126
6    110
dtype: int64

>>> df.apply(np.sum)
fst    386
snd    505
dtype: int64
```

Прочитайте pd.DataFrame.apply онлайн: https://riptutorial.com/ru/pandas/topic/7024/pd-dataframe-apply

глава 8: Resampling

Examples

Даунсэмплинг и повышение частоты дискретизации

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=10, freq='T')
df = pd.DataFrame({'Val' : np.random.randn(len(rng))}, index=rng)
print (df)
                           Val
2015-02-24 00:00:00  1.764052
2015-02-24 00:01:00  0.400157
2015-02-24 00:02:00  0.978738
2015-02-24 00:03:00  2.240893
2015-02-24 00:04:00  1.867558
2015-02-24 00:05:00 -0.977278
2015-02-24 00:06:00  0.950088
2015-02-24 00:07:00 -0.151357
2015-02-24 00:08:00 -0.103219
2015-02-24 00:09:00  0.410599
```

```
#downsampling with aggregating sum
print (df.resample('5Min').sum())
                           Val
2015-02-24 00:00:00  7.251399
2015-02-24 00:05:00  0.128833

#5Min is same as 5T
print (df.resample('5T').sum())
                           Val
2015-02-24 00:00:00  7.251399
2015-02-24 00:05:00  0.128833

#upsampling and fill NaN values method forward filling
print (df.resample('30S').ffill())
                           Val
2015-02-24 00:00:00  1.764052
2015-02-24 00:00:30  1.764052
2015-02-24 00:01:00  0.400157
2015-02-24 00:01:30  0.400157
2015-02-24 00:02:00  0.978738
2015-02-24 00:02:30  0.978738
2015-02-24 00:03:00  2.240893
2015-02-24 00:03:30  2.240893
2015-02-24 00:04:00  1.867558
2015-02-24 00:04:30  1.867558
2015-02-24 00:05:00 -0.977278
2015-02-24 00:05:30 -0.977278
2015-02-24 00:06:00  0.950088
2015-02-24 00:06:30  0.950088
2015-02-24 00:07:00 -0.151357
2015-02-24 00:07:30 -0.151357
```

```
2015-02-24 00:08:00 -0.103219
2015-02-24 00:08:30 -0.103219
2015-02-24 00:09:00  0.410599
```

Прочитайте Resampling онлайн: https://riptutorial.com/ru/pandas/topic/2164/resampling

глава 9: Анализ: объединение всех решений и принятие решений

Examples

Анализ Quintile: со случайными данными

Анализ Quintile является общей основой для оценки эффективности факторов безопасности.

Что такое фактор

Фактор - это метод оценки / оценки наборов ценных бумаг. В определенный момент времени и для определенного набора ценных бумаг фактор может быть представлен как серия панд, где индекс представляет собой массив идентификаторов безопасности, а значения - это оценки или ранги.

Если мы будем оценивать множители с течением времени, мы можем в каждый момент времени разбить набор ценных бумаг на 5 равных ковшей или квинтилей на основе порядка коэффициентов. Нет ничего особенного в отношении числа 5. Мы могли бы использовать 3 или 10. Но мы часто используем 5. Наконец, мы отслеживаем производительность каждого из пяти ведер, чтобы определить, есть ли значительная разница в доходах. Мы склонны более пристально фокусироваться на различиях в доходности ведра с самым высоким рангом относительно наименьшего ранга.

Начнем с установки некоторых параметров и создания случайных данных.

Чтобы облегчить эксперименты с механикой, мы предоставляем простой код для создания случайных данных, чтобы дать нам представление о том, как это работает.

Случайные данные включают

- **Возвращает** : генерирует случайные доходности для указанного количества ценных бумаг и периодов.
- **Сигналы** : генерируют случайные сигналы для определенного количества ценных бумаг и периодов и с заданным уровнем корреляции с **Returns** . Для того чтобы фактор был полезным, должна быть какая-то информация или корреляция между баллами / рангами и последующими доходами. Если бы не было корреляции, мы бы это увидели. Это было бы хорошим упражнением для читателя, дублируйте этот анализ со случайными данными, созданными с корреляцией `0` .

инициализация

```
import pandas as pd
import numpy as np

num_securities = 1000
num_periods = 1000
period_frequency = 'W'
start_date = '2000-12-31'

np.random.seed([3,1415])

means = [0, 0]
covariance = [[  1., 5e-3],
              [5e-3,   1.]]

# generates to sets of data m[0] and m[1] with ~0.005 correlation
m = np.random.multivariate_normal(means, covariance,
                                  (num_periods, num_securities)).T
```

Теперь создадим индекс временных рядов и индекс, представляющий идентификаторы безопасности. Затем используйте их для создания dataframes для возвратов и сигналов

```
ids = pd.Index(['s{:05d}'.format(s) for s in range(num_securities)], 'ID')
tidx = pd.date_range(start=start_date, periods=num_periods, freq=period_frequency)
```

Я делю `m[0]` на `25` чтобы уменьшить масштаб до того, что выглядит как возврат акций. Я также добавляю `1e-7` чтобы дать умеренный положительный средний доход.

```
security_returns = pd.DataFrame(m[0] / 25 + 1e-7, tidx, ids)
security_signals = pd.DataFrame(m[1], tidx, ids)
```

`pd.qcut` - Создайте `pd.qcut` Quintile

Давайте используем `pd.qcut` чтобы разделить мои сигналы на квинтильные ведра за каждый период.

```
def qcut(s, q=5):
    labels = ['q{}'.format(i) for i in range(1, 6)]
    return pd.qcut(s, q, labels=labels)

cut = security_signals.stack().groupby(level=0).apply(qcut)
```

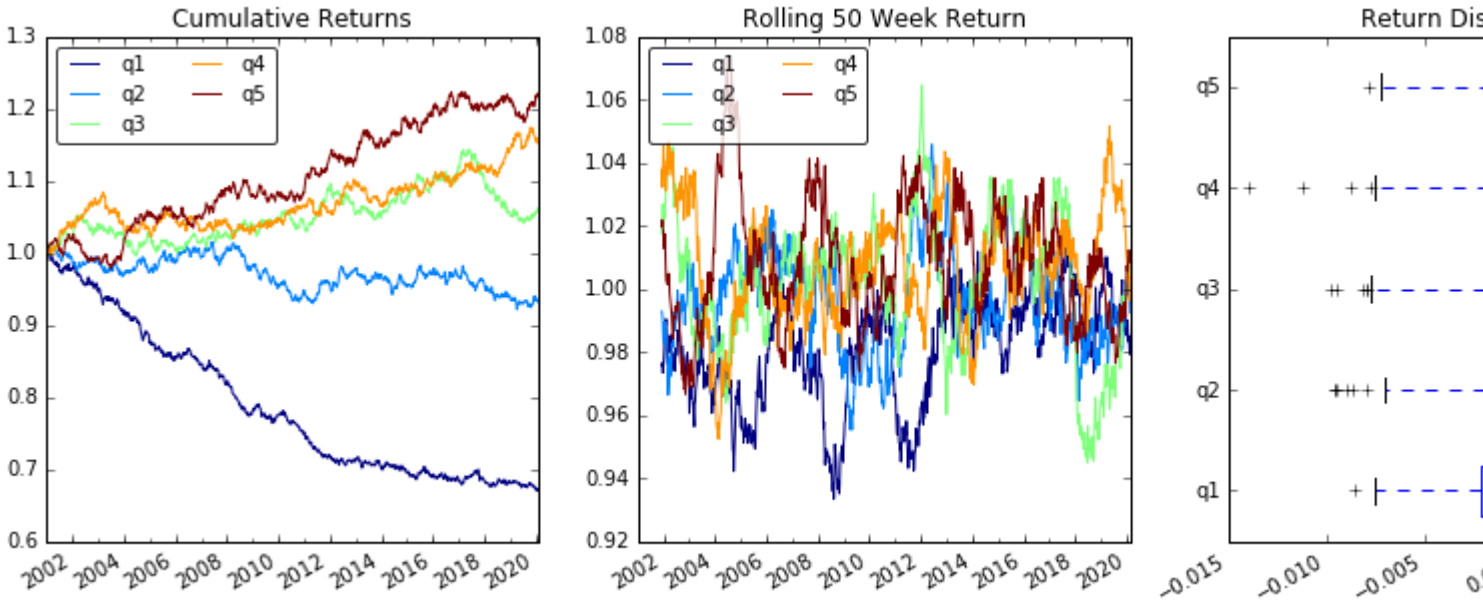
Используйте эти сокращения как индекс для наших возвратов

```
returns_cut = security_returns.stack().rename('returns') \
    .to_frame().set_index(cut, append=True) \
    .swaplevel(2, 1).sort_index().squeeze() \
    .groupby(level=[0, 1]).mean().unstack()
```

Анализ

Возврат к списку

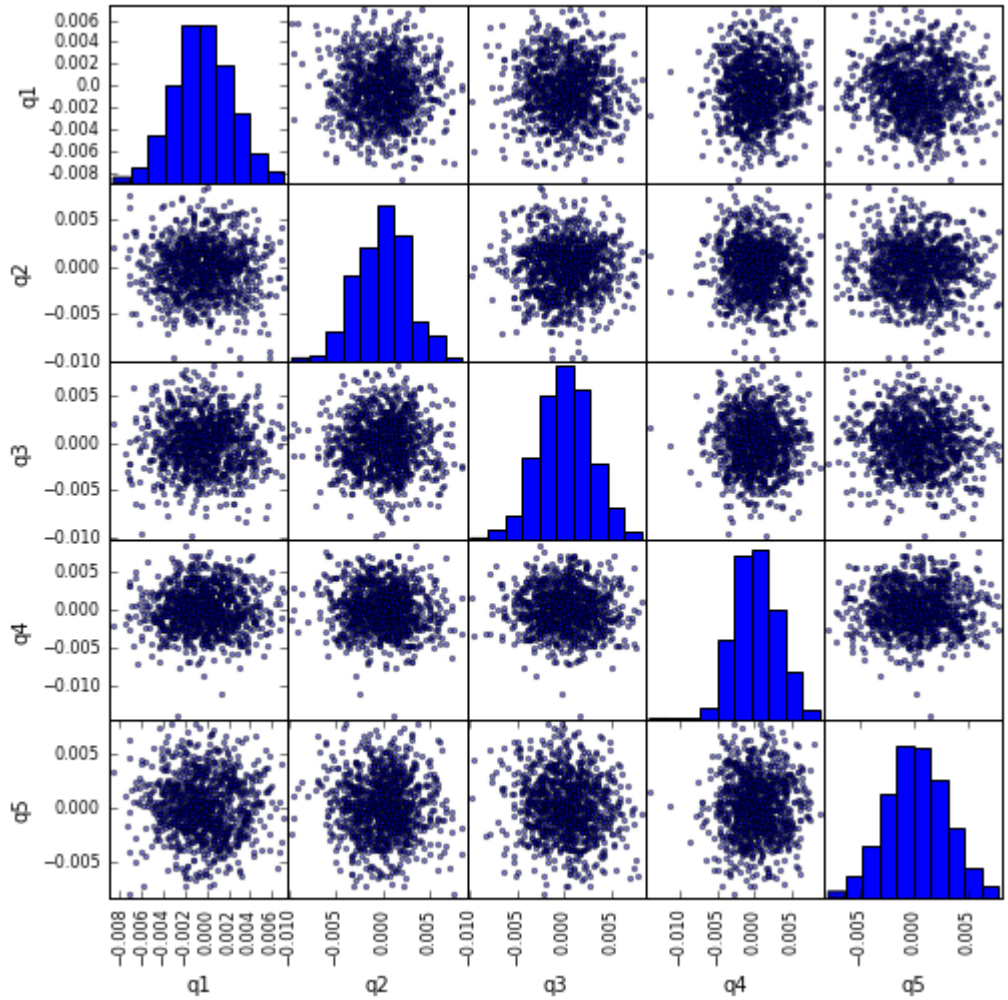
```
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(15, 5))
ax1 = plt.subplot2grid((1,3), (0,0))
ax2 = plt.subplot2grid((1,3), (0,1))
ax3 = plt.subplot2grid((1,3), (0,2))

# Cumulative Returns
returns_cut.add(1).cumprod() \
    .plot(colormap='jet', ax=ax1, title="Cumulative Returns")
leg1 = ax1.legend(loc='upper left', ncol=2, prop={'size': 10}, fancybox=True)
leg1.get_frame().set_alpha(.8)

# Rolling 50 Week Return
returns_cut.add(1).rolling(50).apply(lambda x: x.prod()) \
    .plot(colormap='jet', ax=ax2, title="Rolling 50 Week Return")
leg2 = ax2.legend(loc='upper left', ncol=2, prop={'size': 10}, fancybox=True)
leg2.get_frame().set_alpha(.8)

# Return Distribution
returns_cut.plot.box(vert=False, ax=ax3, title="Return Distribution")

fig.autofmt_xdate()

plt.show()
```



Визуализируйте корреляцию Quintile с помощью `scatter_matrix`

```
from pandas.tools.plotting import scatter_matrix
```

```
scatter_matrix(returns_cut, alpha=0.5, figsize=(8, 8), diagonal='hist')
plt.show()
```



Вычислить и визуализировать максимальный Draw Down

```
def max_dd(returns):
    """returns is a series"""
    r = returns.add(1).cumprod()
    dd = r.div(r.cummax()).sub(1)
    mdd = dd.min()
    end = dd.argmin()
    start = r.loc[:end].argmax()
    return mdd, start, end

def max_dd_df(returns):
    """returns is a dataframe"""
    series = lambda x: pd.Series(x, ['Draw Down', 'Start', 'End'])
    return returns.apply(max_dd).apply(series)
```

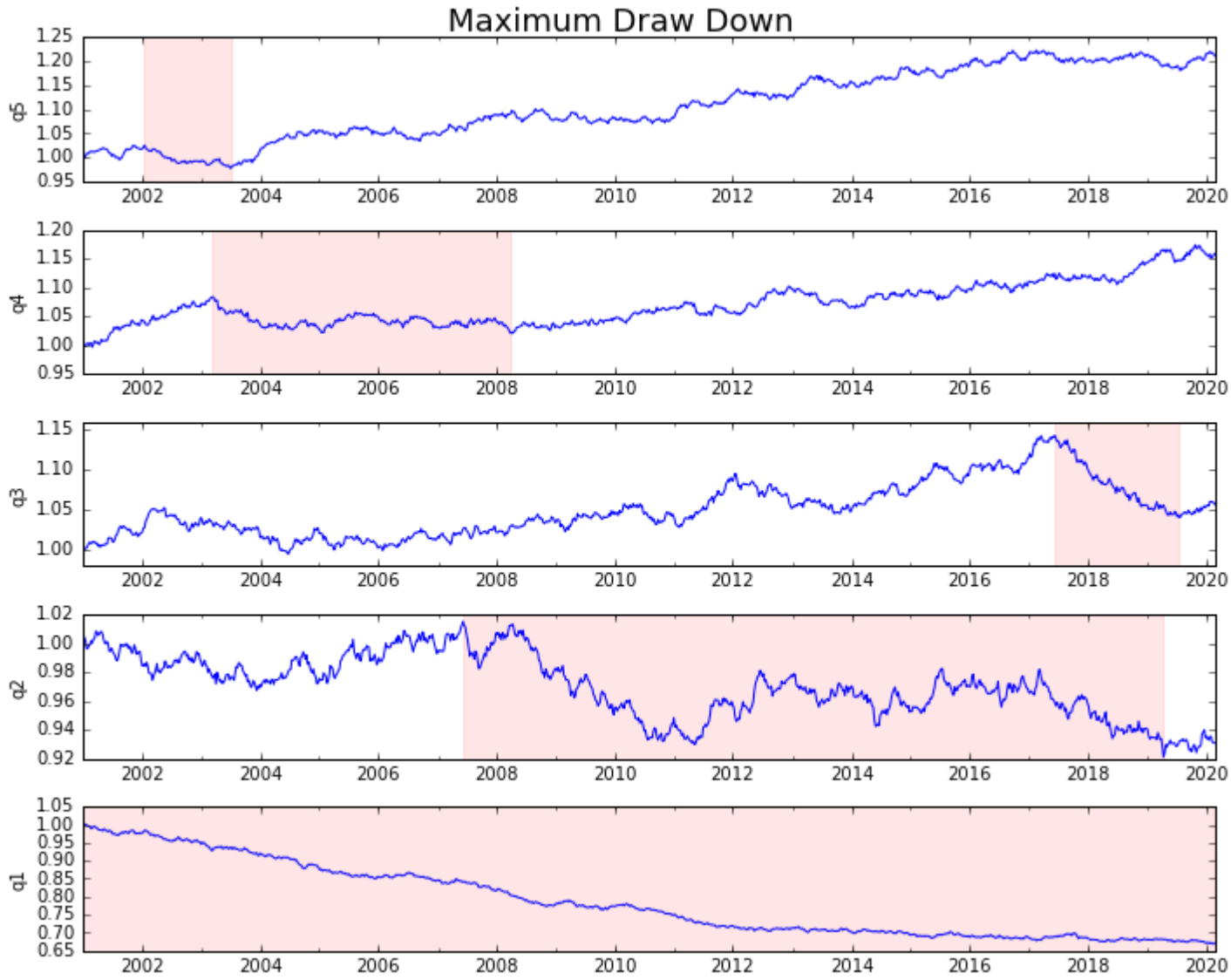
Как это выглядит

```
max_dd_df(returns_cut)
```

	Draw Down	Start	End
q1	-0.333527	2001-01-07	2020-02-16
q2	-0.092659	2007-06-10	2019-04-14
q3	-0.089682	2017-06-11	2019-07-21
q4	-0.058225	2003-03-16	2008-03-30
q5	-0.046822	2002-01-20	2003-07-06

Давайте заговорим

```
draw_downs = max_dd_df(returns_cut)

fig, axes = plt.subplots(5, 1, figsize=(10, 8))
for i, ax in enumerate(axes[::-1]):
    returns_cut.iloc[:, i].add(1).cumprod().plot(ax=ax)
    sd, ed = draw_downs[['Start', 'End']].iloc[i]
    ax.axvspan(sd, ed, alpha=0.1, color='r')
    ax.set_ylabel(returns_cut.columns[i])

fig.suptitle('Maximum Draw Down', fontsize=18)
fig.tight_layout()
plt.subplots_adjust(top=.95)
```



Рассчитать статистику

Есть много потенциальных статистических данных, которые мы можем включить. Ниже всего несколько, но продемонстрируйте, как просто мы можем включить новую статистику в наше резюме.

```
def frequency_of_time_series(df):
    start, end = df.index.min(), df.index.max()
    delta = end - start
    return round((len(df) - 1.) * 365.25 / delta.days, 2)

def annualized_return(df):
    freq = frequency_of_time_series(df)
    return df.add(1).prod() ** (1 / freq) - 1

def annualized_volatility(df):
    freq = frequency_of_time_series(df)
    return df.std().mul(freq ** .5)

def sharpe_ratio(df):
    return annualized_return(df) / annualized_volatility(df)
```

```
def describe(df):
    r = annualized_return(df).rename('Return')
    v = annualized_volatility(df).rename('Volatility')
    s = sharpe_ratio(df).rename('Sharpe')
    skew = df.skew().rename('Skew')
    kurt = df.kurt().rename('Kurtosis')
    desc = df.describe().T

    return pd.concat([r, v, s, skew, kurt, desc], axis=1).T.drop('count')
```

Мы закончим тем, что будем использовать только функцию `describe` как она объединяет всех остальных.

```
describe(returns_cut)
```

	q1	q2	q3	q4	q5
Return	-0.007609	-0.001375	0.001067	0.002821	0.003687
Volatility	0.019584	0.020445	0.020629	0.021185	0.020172
Sharpe	-0.388525	-0.067278	0.051709	0.133176	0.182792
Skew	0.040430	-0.085828	-0.078071	-0.067522	0.005652
Kurtosis	-0.174206	0.203038	0.026385	0.370249	-0.160678
mean	-0.000395	-0.000068	0.000060	0.000151	0.000196
std	0.002711	0.002830	0.002856	0.002933	0.002792
min	-0.008608	-0.009614	-0.009845	-0.014037	-0.007913
25%	-0.002196	-0.002018	-0.001956	-0.001833	-0.001694
50%	-0.000434	0.000065	0.000210	0.000029	0.000146
75%	0.001444	0.001768	0.001989	0.002107	0.002081
max	0.007070	0.008432	0.008100	0.008687	0.007791

Это не должно быть всеобъемлющим. Он призван объединить многие функции панд и продемонстрировать, как вы можете использовать его, чтобы помочь ответить на важные для вас вопросы. Это подмножество типов показателей, которые я использую для оценки эффективности количественных факторов.

Прочитайте Анализ: объединение всех решений и принятие решений онлайн: https://riptutorial.com/ru/pandas/topic/5238/анализ--объединение-всех-решений-и-принятие-решений

глава 10: Булевое индексирование данных

Вступление

Доступ к строкам в фрейме данных с использованием объектов индексатора `.ix` , `.loc` , `.iloc` и того, как он отличается от использования булевой маски.

Examples

Доступ к DataFrame с булевым индексом

Это будет наш примерный кадр данных:

```
df = pd.DataFrame({"color": ['red', 'blue', 'red', 'blue']},
                  index=[True, False, True, False])
      color
True    red
False  blue
True    red
False  blue
```

Доступ с помощью `.loc`

```
df.loc[True]
     color
True   red
True   red
```

Доступ с помощью `.iloc`

```
df.iloc[True]
>> TypeError

df.iloc[1]
color    blue
dtype: object
```

> Важно отметить, что старые версии pandas не различали логический и целочисленный вход, поэтому `.iloc[True]` вернет то же, что и `.iloc[1]`

Доступ с помощью `.ix`

```
df.ix[True]
     color
True   red
True   red

df.ix[1]
```

```
color    blue
dtype: object
```

Как вы можете видеть, `.ix` имеет два поведения. Это очень плохая практика в коде, и поэтому ее следует избегать. Пожалуйста, используйте `.iloc` или `.loc` чтобы быть более явным.

Применение булевой маски к кадру данных

Это будет наш примерный кадр данных:

```
  color      name   size
0   red      rose    big
1  blue    violet    big
2   red     tulip  small
3  blue  harebell  small
```

Использование магии `__getitem__` или `[]` accessor. Предоставляя ему список True и False той же длины, что и dataframe, вы получите:

```
df[[True, False, True, False]]
  color   name   size
0   red   rose    big
2   red  tulip  small
```

Маскирование данных на основе значения столбца

Это будет наш примерный кадр данных:

```
  color      name   size
0   red      rose    big
1  blue    violet  small
2   red     tulip  small
3  blue  harebell  small
```

`pd.Series` к одному столбцу из фрейма данных, мы можем использовать простое сравнение `==` для сравнения каждого элемента в столбце с заданной переменной, создавая `pd.Series` из True и False

```
df['size'] == 'small'
0    False
1     True
2     True
3     True
Name: size, dtype: bool
```

Этот `pd.Series` является расширением `np.array` который является расширением простого `list` Таким образом, мы можем передать это `__getitem__` или `[]` как в приведенном выше примере.

```
size_small_mask = df['size'] == 'small'
df[size_small_mask]
  color      name   size
1  blue    violet  small
2   red     tulip  small
3  blue  harebell  small
```

Маскирование данных на основе значения индекса

Это будет наш примерный кадр данных:

```
         color   size
name
rose       red    big
violet    blue  small
tulip      red  small
harebell  blue  small
```

Мы можем создать маску на основе значений индекса, так же как и на значении столбца.

```
rose_mask = df.index == 'rose'
df[rose_mask]
     color size
name
rose   red  big
```

Но делать это *почти так* же, как

```
df.loc['rose']
color    red
size     big
Name: rose, dtype: object
```

Важным отличием является то, что когда `.loc` встречает только одну строку в соответствующем индексе, он возвращает `pd.Series` , если он встречает больше строк, которые соответствуют, он вернет `pd.DataFrame` . Это делает этот метод довольно неустойчивым.

Это поведение можно контролировать, предоставляя `.loc` список одной записи. Это заставит его вернуть кадр данных.

```
df.loc[['rose']]
         color   size
name
rose       red    big
```

Прочитайте Булевое индексирование данных онлайн:
https://riptutorial.com/ru/pandas/topic/9589/булевое-индексирование-данных

глава 11: Вычислительные инструменты

Examples

Найти корреляцию между столбцами

Предположим, что у вас есть DataFrame числовых значений, например:

```
df = pd.DataFrame(np.random.randn(1000, 3), columns=['a', 'b', 'c'])
```

затем

```
>>> df.corr()
    a    b    c
a    1.000000    0.018602    0.038098
b    0.018602    1.000000    -0.014245
c    0.038098    -0.014245    1.000000
```

найдет корреляцию Пирсона между столбцами. Обратите внимание, как диагональ равна 1, так как каждый столбец (очевидно) полностью коррелирован с самим собой.

`pd.DataFrame.correlation` принимает необязательный параметр `method` , указав, какой алгоритм использовать. По умолчанию используется `pearson` . Например, для использования корреляции Спирмена используйте

```
>>> df.corr(method='spearman')
    a    b    c
a    1.000000    0.007744    0.037209
b    0.007744    1.000000    -0.011823
c    0.037209    -0.011823    1.000000
```

Прочитайте Вычислительные инструменты онлайн:
https://riptutorial.com/ru/pandas/topic/5620/вычислительные-инструменты

глава 12: Графики и визуализации

Examples

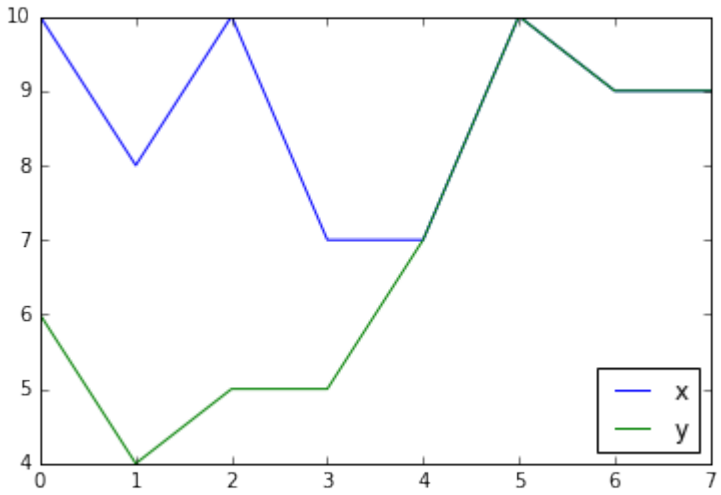
Основные диаграммы данных

Pandas использует несколько способов сделать графики данных внутри фрейма данных. Для этой цели используется matplotlib .

Основные графики имеют свои обертки для объектов DataFrame и Series:

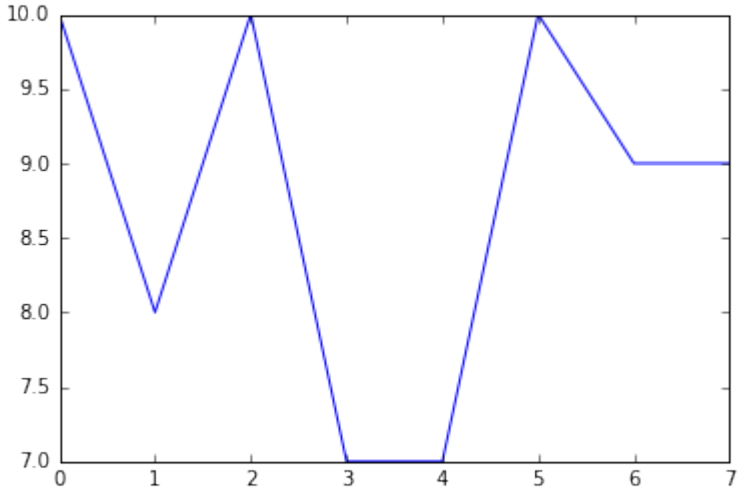
Линейный график

```
df = pd.DataFrame({'x': [10, 8, 10, 7, 7, 10, 9, 9],
                   'y': [6, 4, 5, 5, 7, 10, 9, 9]})
df.plot()
```



Вы можете вызвать тот же метод для объекта Series для построения подмножества Data Frame:

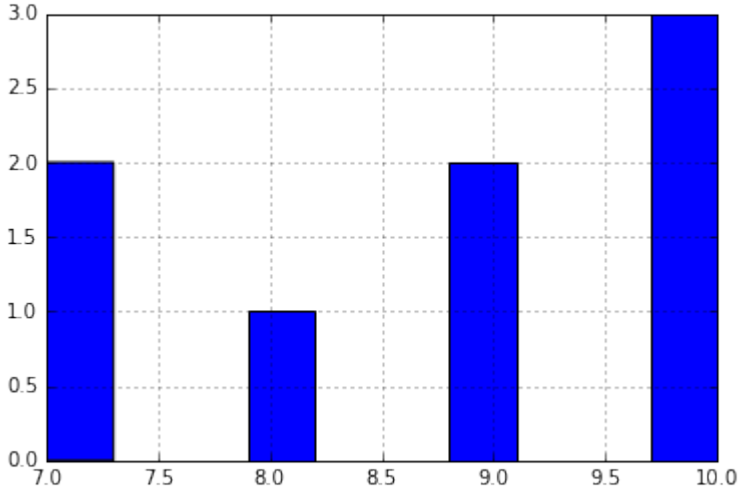
```
df['x'].plot()
```



Барная диаграмма

Если вы хотите изучить распределение ваших данных, вы можете использовать метод `hist()` .

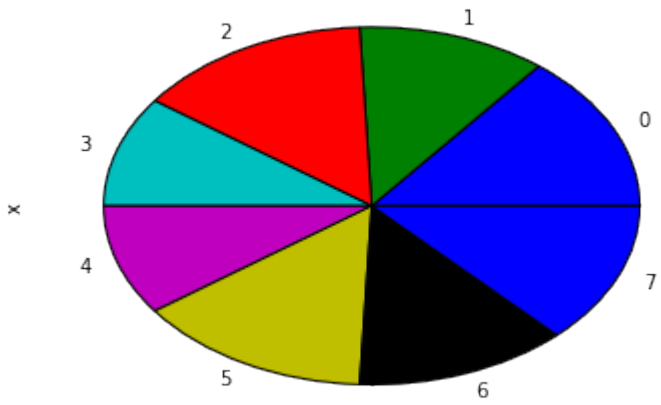
```
df['x'].hist()
```



Общий метод построения графика ()

Все возможные графики доступны через метод построения графика. Тип диаграммы выбирается аргументом **вида** .

```
df['x'].plot(kind='pie')
```



Примечание. Во многих средах круговая диаграмма выйдет овальной. Чтобы сделать его круг, используйте следующее:

```
from matplotlib import pyplot

pyplot.axis('equal')
df['x'].plot(kind='pie')
```

Стилирование сюжета

`plot()` может принимать аргументы, которые передаются matplotlib для стилизации графика по-разному.

```
df.plot(style='o')  # plot as dots, not lines
df.plot(style='g--')  # plot as green dashed line
df.plot(style='o', markeredgecolor='white')  # plot as dots with white edge
```

Участок на существующей оси matplotlib

По умолчанию `plot()` создает новый рисунок каждый раз, когда он вызывается. Можно построить на существующей оси, передав параметр `ax` .

```
plt.figure()  # create a new figure
ax = plt.subplot(121)  # create the left-side subplot
df1.plot(ax=ax)  # plot df1 on that subplot
ax = plt.subplot(122)  # create the right-side subplot
df2.plot(ax=ax)  # and plot df2 there
plt.show()  # show the plot
```

Прочитайте Графики и визуализации онлайн: https://riptutorial.com/ru/pandas/topic/3839/графики-и-визуализации

глава 13: Группирование данных временных рядов

Examples

Генерировать временные ряды случайных чисел, затем вниз образец

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# I want 7 days of 24 hours with 60 minutes each
periods = 7 * 24 * 60
tidx = pd.date_range('2016-07-01', periods=periods, freq='T')
#                     ^                                   ^
#                     |                                   |
#                 Start Date        Frequency Code for Minute
# This should get me 7 Days worth of minutes in a datetimeindex

# Generate random data with numpy.  We'll seed the random
# number generator so that others can see the same results.
# Otherwise, you don't have to seed it.
np.random.seed([3,1415])

# This will pick a number of normally distributed random numbers
# where the number is specified by periods
data = np.random.randn(periods)

ts = pd.Series(data=data, index=tidx, name='HelloTimeSeries')

ts.describe()

count    10080.000000
mean        -0.008853
std          0.995411
min         -3.936794
25%         -0.683442
50%          0.002640
75%          0.654986
max          3.906053
Name: HelloTimeSeries, dtype: float64
```

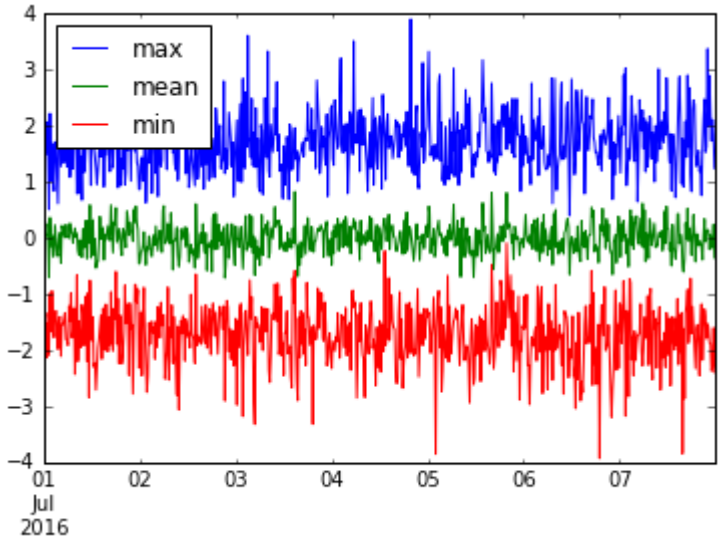
Давайте возьмем эти 7 дней в минуту данных и вниз образец каждые 15 минут. Все частотные коды можно найти здесь .

```
# resample says to group by every 15 minutes.  But now we need
# to specify what to do within those 15 minute chunks.

# We could take the last value.
ts.resample('15T').last()
```

Или любую другую вещь, которую мы можем сделать для объекта `groupby` , документации .

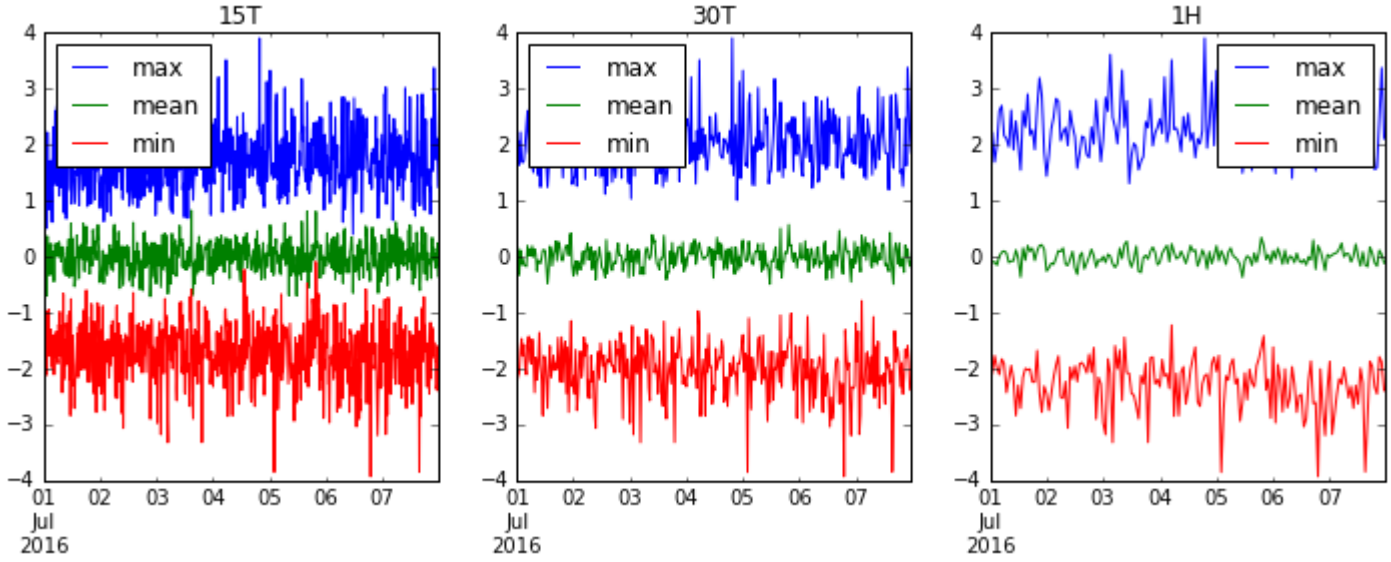
Мы можем даже объединить несколько полезных вещей. Давайте нарисуем `min` , `mean` и `max` этих данных `resample('15M')` .

```
ts.resample('15T').agg(['min', 'mean', 'max']).plot()
```



Давайте переконсервируем `'15T'` (15 минут), `'30T'` (полчаса) и `'1H'` (1 час) и посмотрим, как наши данные становятся более плавными.

```
fig, axes = plt.subplots(1, 3, figsize=(12, 4))
for i, freq in enumerate(['15T', '30T', '1H']):
    ts.resample(freq).agg(['max', 'mean', 'min']).plot(ax=axes[i], title=freq)
```



Прочитайте Группирование данных временных рядов онлайн:
https://riptutorial.com/ru/pandas/topic/4747/группирование-данных-временных-рядов

глава 14: Группировка данных

Examples

Основная группировка

Группировать по одному столбцу

Используя следующий DataFrame

```
df = pd.DataFrame({'A': ['a', 'b', 'c', 'a', 'b', 'b'],
                   'B': [2, 8, 1, 4, 3, 8],
                   'C': [102, 98, 107, 104, 115, 87]})

df
# Output:
#    A  B    C
# 0  a  2  102
# 1  b  8   98
# 2  c  1  107
# 3  a  4  104
# 4  b  3  115
# 5  b  8   87
```

Группируйте по столбцу A и получите среднее значение других столбцов:

```
df.groupby('A').mean()
# Output:
#           B    C
# A
# a  3.000000  103
# b  6.333333  100
# c  1.000000  107
```

Группировать по нескольким столбцам

```
df.groupby(['A','B']).mean()
# Output:
#          C
# A B
# a 2  102.0
#   4  104.0
# b 3  115.0
#   8   92.5
# c 1  107.0
```

Обратите внимание, как после группировки каждая строка в результирующем DataFrame индексируется кортежем или MultiIndex (в этом случае пара элементов из столбцов A и B).

Чтобы применить сразу несколько методов агрегирования, например, чтобы подсчитать

количество элементов в каждой группе и вычислить их среднее значение, используйте функцию `agg` :

```
df.groupby(['A','B']).agg(['count', 'mean'])
# Output:
#         C
#     count   mean
# A B
# a 2     1  102.0
#   4     1  104.0
# b 3     1  115.0
#   8     2   92.5
# c 1     1  107.0
```

Группировка номеров

Для следующего DataFrame:

```
import numpy as np
import pandas as pd
np.random.seed(0)
df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),
                   'Sex': np.random.choice(['Male', 'Female'], 100),
                   'number_of_foo': np.random.randint(1, 20, 100)})
df.head()
# Output:

#    Age     Sex  number_of_foo
# 0   64  Female             14
# 1   67  Female             14
# 2   20  Female             12
# 3   23    Male             17
# 4   23  Female             15
```

Групповой `Age` на три категории (или корзины). Бункеры могут быть указаны как

- целое число `n` указывающее количество ящиков, - в этом случае данные dataframe делятся на `n` интервалов равного размера
- последовательность целых чисел, обозначающая конечную точку левых открытых интервалов, в которой данные делятся на, например, `bins=[19, 40, 65, np.inf]` создает три возрастные группы `(19, 40]` , `(40, 65]` и `(65, np.inf]` .

Pandas автоматически назначает строковые версии интервалов в качестве метки. Также можно определить собственные метки, указав параметр `labels` в виде списка строк.

```
pd.cut(df['Age'], bins=4)
# this creates four age groups: (19.951, 32.25] < (32.25, 44.5] < (44.5, 56.75] < (56.75, 69]
Name: Age, dtype: category
Categories (4, object): [(19.951, 32.25] < (32.25, 44.5] < (44.5, 56.75] < (56.75, 69]]

pd.cut(df['Age'], bins=[19, 40, 65, np.inf])
# this creates three age groups: (19, 40], (40, 65] and (65, infinity)
Name: Age, dtype: category
```

```
Categories (3, object): [(19, 40] < (40, 65] < (65, inf]]
```

Используйте его в `groupby` чтобы получить среднее число foo:

```
age_groups = pd.cut(df['Age'], bins=[19, 40, 65, np.inf])
df.groupby(age_groups)['number_of_foo'].mean()
# Output:
# Age
# (19, 40]     9.880000
# (40, 65]     9.452381
# (65, inf]    9.250000
# Name: number_of_foo, dtype: float64
```

Кросс-таблицы Возрастные группы и пол:

```
pd.crosstab(age_groups, df['Sex'])
# Output:
# Sex         Female  Male
# Age
# (19, 40]        22    28
# (40, 65]        18    24
# (65, inf]        3     5
```

Выбор столбца группы

Когда вы делаете группу, вы можете выбрать один столбец или список столбцов:

```
In [11]: df = pd.DataFrame([[1, 1, 2], [1, 2, 3], [2, 3, 4]], columns=["A", "B", "C"])

In [12]: df
Out[12]:
   A  B  C
0  1  1  2
1  1  2  3
2  2  3  4

In [13]: g = df.groupby("A")

In [14]: g["B"].mean()           # just column B
Out[14]:
A
1    1.5
2    3.0
Name: B, dtype: float64

In [15]: g[["B", "C"]].mean()    # columns B and C
Out[15]:
     B    C
A
1  1.5  2.5
2  3.0  4.0
```

Вы также можете использовать `agg` для указания столбцов и агрегации для выполнения:

```
In [16]: g.agg({'B': 'mean', 'C': 'count'})
Out[16]:
   C    B
A
1  2  1.5
2  1  3.0
```

Агрегация по размеру по сравнению с подсчетом

Разница между `size` и `count` :

`size` имеет значение `NaN` значения, `count` не делает.

```
df = pd.DataFrame(
        {"Name":["Alice", "Bob", "Mallory", "Mallory", "Bob" , "Mallory"],
         "City":["Seattle", "Seattle", "Portland", "Seattle", "Seattle", "Portland"],
         "Val": [4, 3, 3, np.nan, np.nan, 4]})

df
# Output:
#        City     Name  Val
# 0   Seattle    Alice  4.0
# 1   Seattle      Bob  3.0
# 2  Portland  Mallory  3.0
# 3   Seattle  Mallory  NaN
# 4   Seattle      Bob  NaN
# 5  Portland  Mallory  4.0


df.groupby(["Name", "City"])['Val'].size().reset_index(name='Size')
# Output:
#       Name      City  Size
# 0    Alice   Seattle     1
# 1      Bob   Seattle     2
# 2  Mallory  Portland     2
# 3  Mallory   Seattle     1

df.groupby(["Name", "City"])['Val'].count().reset_index(name='Count')
# Output:
#       Name      City  Count
# 0    Alice   Seattle      1
# 1      Bob   Seattle      1
# 2  Mallory  Portland      2
# 3  Mallory   Seattle      0
```

Агрегирующие группы

```
In [1]: import numpy as np
In [2]: import pandas as pd

In [3]: df = pd.DataFrame({'A': list('XYZXYZXYZX'), 'B': [1, 2, 1, 3, 1, 2, 3, 3, 1, 2],
                           'C': [12, 14, 11, 12, 13, 14, 16, 12, 10, 19]})

In [4]: df.groupby('A')['B'].agg({'mean': np.mean, 'standard deviation': np.std})
Out[4]:
   standard deviation      mean
```

```
A
X            0.957427  2.250000
Y            1.000000  2.000000
Z            0.577350  1.333333
```

Для нескольких столбцов:

```
In [5]: df.groupby('A').agg({'B': [np.mean, np.std], 'C': [np.sum, 'count']})
Out[5]:
    C              B
  sum count      mean       std
A
X  59     4  2.250000  0.957427
Y  39     3  2.000000  1.000000
Z  35     3  1.333333  0.577350
```

Экспорт групп в разные файлы

Вы можете перебирать объект, возвращенный `groupby()` . Итератор содержит `(Category, DataFrame)` кортежи.

```
# Same example data as in the previous example.
import numpy as np
import pandas as pd
np.random.seed(0)
df = pd.DataFrame({'Age': np.random.randint(20, 70, 100),
                   'Sex': np.random.choice(['Male', factor'Female'], 100),
                   'number_of_foo': np.random.randint(1, 20, 100)})

# Export to Male.csv and Female.csv files.
for sex, data in df.groupby('Sex'):
    data.to_csv("{}.csv".format(sex))
```

используя преобразование для получения статистики на уровне группы при сохранении исходного кадра данных

пример:

```
df = pd.DataFrame({'group1' :  ['A', 'A', 'A', 'A',
                                'B', 'B', 'B', 'B'],
                   'group2' :  ['C', 'C', 'C', 'D',
                                'E', 'E', 'F', 'F'],
                   'B'      :  ['one', np.NaN, np.NaN, np.NaN,
                                 np.NaN, 'two', np.NaN, np.NaN],
                   'C'      :  [np.NaN, 1, np.NaN, np.NaN,
                                np.NaN, np.NaN, np.NaN, 4]})

 df
Out[34]:
     B    C group1 group2
0  one  NaN      A      C
1  NaN  1.0      A      C
2  NaN  NaN      A      C
3  NaN  NaN      A      D
```

```
4  NaN  NaN      B      E
5  two  NaN      B      E
6  NaN  NaN      B      F
7  NaN  4.0      B      F
```

Я хочу , чтобы получить количество не-пропущенных наблюдений B для каждой комбинации `group1` и `group2` . `groupby.transform` - очень мощная функция, которая делает именно это.

```
df['count_B']=df.groupby(['group1','group2']).B.transform('count')

df
Out[36]:
     B    C group1 group2  count_B
0  one  NaN      A      C        1
1  NaN  1.0      A      C        1
2  NaN  NaN      A      C        1
3  NaN  NaN      A      D        0
4  NaN  NaN      B      E        1
5  two  NaN      B      E        1
6  NaN  NaN      B      F        0
7  NaN  4.0      B      F        0
```

Прочитайте Группировка данных онлайн: https://riptutorial.com/ru/pandas/topic/1822/группировка-данных

глава 15: Данные о сдвиге и запаздывании

Examples

Смещение или отставание значений в кадре данных

```
import pandas as pd

df = pd.DataFrame({'eggs': [1,2,4,8,], 'chickens': [0,1,2,4,]})

df

#    chickens  eggs
# 0         0     1
# 1         1     2
# 2         2     4
# 3         4     8

df.shift()

#    chickens  eggs
# 0       NaN   NaN
# 1       0.0   1.0
# 2       1.0   2.0
# 3       2.0   4.0

df.shift(-2)

#    chickens  eggs
# 0       2.0   4.0
# 1       4.0   8.0
# 2       NaN   NaN
# 3       NaN   NaN

df['eggs'].shift(1) - df['chickens']

# 0    NaN
# 1    0.0
# 2    0.0
# 3    0.0
```

Первый аргумент `.shift()` - это `periods` , количество пробелов для перемещения данных. Если не указано, значение по умолчанию равно `1` .

Прочитайте Данные о сдвиге и запаздывании онлайн:
https://riptutorial.com/ru/pandas/topic/7554/данные-о-сдвиге-и-запаздывании

глава 16: Добавление к DataFrame

Examples

Добавление новой строки в DataFrame

```
In [1]: import pandas as pd

In [2]: df = pd.DataFrame(columns = ['A', 'B', 'C'])

In [3]: df
Out[3]:
Empty DataFrame
Columns: [A, B, C]
Index: []
```

Добавление строки по одному столбцу:

```
In [4]: df.loc[0, 'A'] = 1

In [5]: df
Out[5]:
   A    B    C
0  1  NaN  NaN
```

Добавляя строку, заданный список значений:

```
In [6]: df.loc[1] = [2, 3, 4]

In [7]: df
Out[7]:
   A    B    C
0  1  NaN  NaN
1  2    3    4
```

Добавление строки с использованием словаря:

```
In [8]: df.loc[2] = {'A': 3, 'C': 9, 'B': 9}

In [9]: df
Out[9]:
   A    B    C
0  1  NaN  NaN
1  2    3    4
2  3    9    9
```

Первый вход в .loc [] - это индекс. Если вы используете существующий индекс, вы будете перезаписывать значения в этой строке:

```
In [17]: df.loc[1] = [5, 6, 7]
```

```
In [18]: df
Out[18]:
   A    B    C
0  1  NaN  NaN
1  5    6    7
2  3    9    9


In [19]: df.loc[0, 'B'] = 8

In [20]: df
Out[20]:
   A  B    C
0  1  8  NaN
1  5  6    7
2  3  9    9
```

Добавить DataFrame в другой DataFrame

Предположим, что мы имеем следующие два DataFrames:

```
In [7]: df1
Out[7]:
    A   B
0  a1  b1
1  a2  b2

In [8]: df2
Out[8]:
    B   C
0  b1  c1
```

Оба DataFrames не должны иметь одинаковый набор столбцов. Метод append не изменяет ни один из исходных DataFrames. Вместо этого он возвращает новый DataFrame, добавляя исходные два. Добавление DataFrame в другое довольно просто:

```
In [9]: df1.append(df2)
Out[9]:
     A   B    C
0   a1  b1  NaN
1   a2  b2  NaN
0  NaN  b1   c1
```

Как вы можете видеть, возможно иметь повторяющиеся индексы (0 в этом примере). Чтобы избежать этой проблемы, вы можете попросить Pandas повторно проиндексировать новый DataFrame для вас:

```
In [10]: df1.append(df2, ignore_index = True)
Out[10]:
     A   B    C
0   a1  b1  NaN
1   a2  b2  NaN
2  NaN  b1   c1
```

Прочитайте Добавление к DataFrame онлайн: https://riptutorial.com/ru/pandas/topic/6456/добавление-к-dataframe

глава 17: Дублированные данные

Examples

Выберите дублированный

Если нужно установить значение `0` в столбец `B` , где в столбце `A` дублируются данные, сначала создайте маску с помощью `Series.duplicated` а затем используйте `DataFrame.ix` или `Series.mask` :

```
In [224]: df = pd.DataFrame({'A':[1,2,3,3,2],
     ...:                    'B':[1,7,3,0,8]})

In [225]: mask = df.A.duplicated(keep=False)

In [226]: mask
Out[226]:
0    False
1     True
2     True
3     True
4     True
Name: A, dtype: bool

In [227]: df.ix[mask, 'B'] = 0

In [228]: df['C'] = df.A.mask(mask, 0)

In [229]: df
Out[229]:
   A  B  C
0  1  1  1
1  2  0  0
2  3  0  0
3  3  0  0
4  2  0  0
```

Если нужно инвертировать маску, используйте `~` :

```
In [230]: df['C'] = df.A.mask(~mask, 0)

In [231]: df
Out[231]:
   A  B  C
0  1  1  0
1  2  0  2
2  3  0  3
3  3  0  3
4  2  0  2
```

Двойное дублирование

Использовать `drop_duplicates` :

```
In [216]: df = pd.DataFrame({'A':[1,2,3,3,2],
     ...:                    'B':[1,7,3,0,8]})

In [217]: df
Out[217]:
   A  B
0  1  1
1  2  7
2  3  3
3  3  0
4  2  8

# keep only the last value
In [218]: df.drop_duplicates(subset=['A'], keep='last')
Out[218]:
   A  B
0  1  1
3  3  0
4  2  8

# keep only the first value, default value
In [219]: df.drop_duplicates(subset=['A'], keep='first')
Out[219]:
   A  B
0  1  1
1  2  7
2  3  3

# drop all duplicated values
In [220]: df.drop_duplicates(subset=['A'], keep=False)
Out[220]:
   A  B
0  1  1
```

Если вы не хотите получать копию фрейма данных, но для изменения существующего:

```
In [221]: df = pd.DataFrame({'A':[1,2,3,3,2],
     ...:                    'B':[1,7,3,0,8]})

In [222]: df.drop_duplicates(subset=['A'], inplace=True)

In [223]: df
Out[223]:
   A  B
0  1  1
1  2  7
2  3  3
```

Подсчет и получение уникальных элементов

Количество уникальных элементов в серии:

```
In [1]: id_numbers = pd.Series([111, 112, 112, 114, 115, 118, 114, 118, 112])
In [2]: id_numbers.nunique()
Out[2]: 5
```

Получите уникальные элементы в серии:

```
In [3]: id_numbers.unique()
Out[3]: array([111, 112, 114, 115, 118], dtype=int64)

In [4]: df = pd.DataFrame({'Group': list('ABAABABAAB'),
                           'ID': [1, 1, 2, 3, 3, 2, 1, 2, 1, 3]})

In [5]: df
Out[5]:
  Group  ID
0     A   1
1     B   1
2     A   2
3     A   3
4     B   3
5     A   2
6     B   1
7     A   2
8     A   1
9     B   3
```

Количество уникальных элементов в каждой группе:

```
In [6]: df.groupby('Group')['ID'].nunique()
Out[6]:
Group
A    3
B    2
Name: ID, dtype: int64
```

Получите уникальные элементы в каждой группе:

```
In [7]: df.groupby('Group')['ID'].unique()
Out[7]:
Group
A    [1, 2, 3]
B       [1, 3]
Name: ID, dtype: object
```

Получите уникальные значения из столбца.

```
In [15]: df = pd.DataFrame({"A":[1,1,2,3,1,1],"B":[5,4,3,4,6,7]})

In [21]: df
Out[21]:
   A  B
0  1  5
1  1  4
2  2  3
3  3  4
4  1  6
5  1  7
```

Чтобы получить уникальные значения в столбцах A и B.

```
In [22]: df["A"].unique()
Out[22]: array([1, 2, 3])

In [23]: df["B"].unique()
Out[23]: array([5, 4, 3, 6, 7])
```

Чтобы получить уникальные значения в столбце A в виде списка (обратите внимание, что `unique()` может использоваться двумя разными способами)

```
In [24]: pd.unique(df['A']).tolist()
Out[24]: [1, 2, 3]
```

Вот более сложный пример. Скажем, мы хотим найти уникальные значения из столбца «B», где «A» равно 1.

Во-первых, давайте представим дубликат, чтобы вы могли видеть, как он работает. Давайте заменим 6 в строке «4», столбец «B» на 4:

```
In [24]: df.loc['4', 'B'] = 4
Out[24]:
   A  B
0  1  5
1  1  4
2  2  3
3  3  4
4  1  4
5  1  7
```

Теперь выберите данные:

```
In [25]: pd.unique(df[df['A'] == 1 ]['B']).tolist()
Out[25]: [5, 4, 7]
```

Это можно разбить, если сначала подумать о внутреннем DataFrame:

```
df['A'] == 1
```

Это находит значения в столбце A, равные 1, и применяет к ним True или False. Затем мы можем использовать это для выбора значений из столбца «B» DataFrame (внешний выбор DataFrame)

Для сравнения, вот список, если мы не используем уникальный. Он извлекает каждое значение в столбце «B», где столбец «A» равен 1

```
In [26]: df[df['A'] == 1]['B'].tolist()
Out[26]: [5, 4, 4, 7]
```

Прочитайте Дублированные данные онлайн: https://riptutorial.com/ru/pandas/topic/2082/дублированные-данные

глава 18: Значения карты

замечания

следует отметить, что если ключевое значение не существует, это приведет к повышению `KeyError` , в таких ситуациях, возможно, лучше использовать `merge` или `get` что позволяет указать значение по умолчанию, если ключ не существует

Examples

Карта из словаря

Начиная с кадра данных `df` :

```
  U   L
111  en
112  en
112  es
113  es
113  ja
113  zh
114  es
```

Представьте, что вы хотите добавить новый столбец `S` принимающий значения из следующего словаря:

```
d = {112: 'en', 113: 'es', 114: 'es', 111: 'en'}
```

Вы можете использовать `map` для выполнения поиска по клавишам, возвращающим соответствующие значения в качестве нового столбца:

```
df['S'] = df['U'].map(d)
```

который возвращает:

```
  U   L   S
111  en  en
112  en  en
112  es  en
113  es  es
113  ja  es
113  zh  es
114  es  es
```

Прочитайте Значения карты онлайн: https://riptutorial.com/ru/pandas/topic/3928/значения-карты

глава 19: Изменение формы и поворот

Examples

Простой поворот

Сначала попробуйте использовать `pivot` :

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Name':['Mary', 'Josh','Jon','Lucy', 'Jane', 'Sue'],
                   'Age':[34, 37, 29, 40, 29, 31],
                   'City':['Boston','New York', 'Chicago', 'Los Angeles', 'Chicago',
'Boston'],
                   'Position':['Manager','Programmer','Manager','Manager','Programmer',
'Programmer']},
                    columns=['Name','Position','City','Age'])

print (df)
   Name    Position         City  Age
0  Mary     Manager       Boston   34
1  Josh  Programmer     New York   37
2   Jon     Manager      Chicago   29
3  Lucy     Manager  Los Angeles   40
4  Jane  Programmer      Chicago   29
5   Sue  Programmer       Boston   31

print (df.pivot(index='Position', columns='City', values='Age'))
City        Boston  Chicago  Los Angeles  New York
Position
Manager       34.0     29.0         40.0       NaN
Programmer    31.0     29.0          NaN      37.0
```

Если необходимо сбросить индекс, удалите имена столбцов и заполните значения NaN:

```
#pivoting by numbers - column Age
print (df.pivot(index='Position', columns='City', values='Age')
         .reset_index()
         .rename_axis(None, axis=1)
         .fillna(0))

     Position  Boston  Chicago  Los Angeles  New York
0     Manager    34.0     29.0         40.0       0.0
1  Programmer    31.0     29.0          0.0      37.0


#pivoting by strings - column Name
print (df.pivot(index='Position', columns='City', values='Name'))

City       Boston Chicago Los Angeles New York
Position
Manager      Mary     Jon        Lucy     None
Programmer    Sue    Jane        None     Josh
```

Поворот с агрегированием

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Name':['Mary', 'Jon','Lucy', 'Jane', 'Sue', 'Mary', 'Lucy'],
                   'Age':[35, 37, 40, 29, 31, 26, 28],
                   'City':['Boston', 'Chicago', 'Los Angeles', 'Chicago', 'Boston', 'Boston',
'Chicago'],
                   'Position':['Manager','Manager','Manager','Programmer',
'Programmer','Manager','Manager'],
                    'Sex':['Female','Male','Female','Female', 'Female','Female','Female']},
                    columns=['Name','Position','City','Age','Sex'])

print (df)
   Name    Position         City  Age  Sex
0  Mary     Manager       Boston   35  Female
1   Jon     Manager      Chicago   37  Male
2  Lucy     Manager  Los Angeles   40  Female
3  Jane  Programmer      Chicago   29  Female
4   Sue  Programmer       Boston   31  Female
5  Mary     Manager       Boston   26  Female
6  Lucy     Manager      Chicago   28  Female
```

Если используется `pivot` , получите ошибку:

```
print (df.pivot(index='Position', columns='City', values='Age'))
```

ValueError: индекс содержит повторяющиеся записи, не может изменять форму

Используйте `pivot_table` с агрегирующей функцией:

```
#default aggfunc is np.mean
print (df.pivot_table(index='Position', columns='City', values='Age'))
City        Boston  Chicago  Los Angeles
Position
Manager       30.5     32.5         40.0
Programmer    31.0     29.0          NaN

print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc=np.mean))
City        Boston  Chicago  Los Angeles
Position
Manager       30.5     32.5         40.0
Programmer    31.0     29.0          NaN
```

Другие функции agg:

```
print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc=sum))
City        Boston  Chicago  Los Angeles
Position
Manager       61.0     65.0         40.0
Programmer    31.0     29.0          NaN

#lost data !!!
print (df.pivot_table(index='Position', columns='City', values='Age', aggfunc='first'))
```

```
City        Boston  Chicago  Los Angeles
Position
Manager       35.0     37.0         40.0
Programmer    31.0     29.0          NaN
```

Если необходимо заполнить по столбцам со `string` значениями:

```
print (df.pivot_table(index='Position', columns='City', values='Name'))
```

DataError: нет числовых типов для агрегирования

Вы можете использовать эти аггрегирующие функции:

```
print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='first'))
City       Boston Chicago Los Angeles
Position
Manager      Mary     Jon        Lucy
Programmer    Sue    Jane        None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='last'))
City       Boston Chicago Los Angeles
Position
Manager      Mary    Lucy        Lucy
Programmer    Sue    Jane        None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc='sum'))
City          Boston  Chicago Los Angeles
Position
Manager     MaryMary  JonLucy        Lucy
Programmer       Sue     Jane        None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc=', '.join))
City            Boston    Chicago Los Angeles
Position
Manager     Mary, Mary  Jon, Lucy        Lucy
Programmer         Sue       Jane        None

print (df.pivot_table(index='Position', columns='City', values='Name', aggfunc=', '.join,
fill_value='-')
         .reset_index()
         .rename_axis(None, axis=1))
     Position      Boston    Chicago Los Angeles
0     Manager  Mary, Mary  Jon, Lucy        Lucy
1  Programmer         Sue       Jane           -
```

Информация о *Сексе* пока не используется. Он может быть переключен одним из столбцов, или он может быть добавлен как другой уровень:

```
print (df.pivot_table(index='Position', columns=['City','Sex'], values='Age',
aggfunc='first'))

City       Boston Chicago       Los Angeles
Sex        Female  Female  Male      Female
Position
Manager      35.0    28.0  37.0        40.0
Programmer   31.0    29.0   NaN         NaN
```

Несколько столбцов могут быть указаны в любом из атрибутов index, columns и values.

```
print (df.pivot_table(index=['Position','Sex'], columns='City', values='Age',
aggfunc='first'))

City               Boston  Chicago  Los Angeles
Position   Sex
Manager    Female    35.0     28.0         40.0
           Male       NaN     37.0          NaN
Programmer Female    31.0     29.0          NaN
```

Применение нескольких агрегирующих функций

Вы можете легко применять несколько функций во время одного поворота:

```
In [23]: import numpy as np

In [24]: df.pivot_table(index='Position', values='Age', aggfunc=[np.mean, np.std])
Out[24]:
                mean       std
Position
Manager     34.333333  5.507571
Programmer  32.333333  4.163332
```

Иногда вы можете использовать определенные функции для определенных столбцов:

```
In [35]: df['Random'] = np.random.random(6)
In [36]: df
Out[36]:
   Name    Position         City  Age    Random
0  Mary     Manager       Boston   34  0.678577
1  Josh  Programmer     New York   37  0.973168
2   Jon     Manager      Chicago   29  0.146668
3  Lucy     Manager  Los Angeles   40  0.150120
4  Jane  Programmer      Chicago   29  0.112769
5   Sue  Programmer       Boston   31  0.185198

For example, find the mean age, and standard deviation of random by Position:

In [37]: df.pivot_table(index='Position', aggfunc={'Age': np.mean, 'Random': np.std})
Out[37]:
                 Age    Random
Position
Manager     34.333333  0.306106
Programmer  32.333333  0.477219
```

Можно также передать список функций для применения к отдельным столбцам:

```
In [38]: df.pivot_table(index='Position', aggfunc={'Age': np.mean, 'Random': [np.mean,
np.std]})]
Out[38]:
                 Age    Random
                mean      mean       std
Position
Manager     34.333333  0.325122  0.306106
Programmer  32.333333  0.423712  0.477219
```

Складывание и развёртывание

```
import pandas as pd
import numpy as np

np.random.seed(0)
tuples = list(zip(*[['bar', 'bar', 'foo', 'foo', 'qux', 'qux'],
                    ['one', 'two', 'one', 'two','one', 'two']]))

idx = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
df = pd.DataFrame(np.random.randn(6, 2), index=idx, columns=['A', 'B'])
print (df)
                     A         B
first second
bar   one     1.764052  0.400157
      two     0.978738  2.240893
foo   one     1.867558 -0.977278
      two     0.950088 -0.151357
qux   one    -0.103219  0.410599
      two     0.144044  1.454274
```

```
print (df.stack())
first  second
bar    one     A    1.764052
               B    0.400157
       two     A    0.978738
               B    2.240893
foo    one     A    1.867558
               B   -0.977278
       two     A    0.950088
               B   -0.151357
qux    one     A   -0.103219
               B    0.410599
       two     A    0.144044
               B    1.454274
dtype: float64

#reset index, rename column name
print (df.stack().reset_index(name='val2').rename(columns={'level_2': 'val1'}))
   first second val1      val2
0    bar    one    A  1.764052
1    bar    one    B  0.400157
2    bar    two    A  0.978738
3    bar    two    B  2.240893
4    foo    one    A  1.867558
5    foo    one    B -0.977278
6    foo    two    A  0.950088
7    foo    two    B -0.151357
8    qux    one    A -0.103219
9    qux    one    B  0.410599
10   qux    two    A  0.144044
11   qux    two    B  1.454274
```

```
print (df.unstack())
               A                   B
second       one       two       one       two
first
bar     1.764052  0.978738  0.400157  2.240893
```

```
foo     1.867558  0.950088 -0.977278 -0.151357
qux    -0.103219  0.144044  0.410599  1.454274
```

`rename_axis` (новый в `pandas 0.18.0`):

```
#reset index, remove columns names
df1 = df.unstack().reset_index().rename_axis((None,None), axis=1)
#reset MultiIndex in columns with list comprehension
df1.columns = ['_'.join(col).strip('_') for col in df1.columns]
print (df1)
  first     A_one     A_two     B_one     B_two
0   bar  1.764052  0.978738  0.400157  2.240893
1   foo  1.867558  0.950088 -0.977278 -0.151357
2   qux -0.103219  0.144044  0.410599  1.454274
```

pandas ниже 0.18.0

```
#reset index
df1 = df.unstack().reset_index()
#remove columns names
df1.columns.names = (None, None)
#reset MultiIndex in columns with list comprehension
df1.columns = ['_'.join(col).strip('_') for col in df1.columns]
print (df1)
  first     A_one     A_two     B_one     B_two
0   bar  1.764052  0.978738  0.400157  2.240893
1   foo  1.867558  0.950088 -0.977278 -0.151357
2   qux -0.103219  0.144044  0.410599  1.454274
```

Кросстабуляция

```
import pandas as pd
df = pd.DataFrame({'Sex': ['M', 'M', 'F', 'M', 'F', 'F', 'M', 'M', 'F', 'F'],
                   'Age': [20, 19, 17, 35, 22, 22, 12, 15, 17, 22],
                   'Heart Disease': ['Y', 'N', 'Y', 'N', 'N', 'Y', 'N', 'Y', 'N', 'Y']})

df

  Age Heart Disease Sex
0   20             Y   M
1   19             N   M
2   17             Y   F
3   35             N   M
4   22             N   F
5   22             Y   F
6   12             N   M
7   15             Y   M
8   17             N   F
9   22             Y   F

pd.crosstab(df['Sex'], df['Heart Disease'])

Hearth Disease  N  Y
Sex
F               2  3
M               3  2
```

Использование точечной нотации:

```
pd.crosstab(df.Sex, df.Age)

Age  12  15  17  19  20  22  35
Sex
F     0   0   2   0   0   3   0
M     1   1   0   1   1   0   1
```

Получение транспонирования DF:

```
pd.crosstab(df.Sex, df.Age).T

Sex  F  M
Age
12   0  1
15   0  1
17   2  0
19   0  1
20   0  1
22   3  0
35   0  1
```

Получение полей или кумулятивов:

```
pd.crosstab(df['Sex'], df['Heart Disease'], margins=True)

Heart Disease  N  Y  All
Sex
F              2  3    5
M              3  2    5
All            5  5   10
```

Получение транспонирования кумулятивных:

```
pd.crosstab(df['Sex'], df['Age'], margins=True).T


Sex  F  M  All
Age
12   0  1    1
15   0  1    1
17   2  0    2
19   0  1    1
20   0  1    1
22   3  0    3
35   0  1    1
All  5  5   10
```

Получение процентов:

```
pd.crosstab(df["Sex"],df['Heart Disease']).apply(lambda r: r/len(df), axis=1)

Heart Disease    N    Y
Sex
```

```
F              0.2  0.3
M              0.3  0.2
```

Накопление и умножение на 100:

```
df2 = pd.crosstab(df["Age"],df['Sex'], margins=True ).apply(lambda r: r/len(df)*100, axis=1)

df2

Sex     F     M    All
Age
12    0.0  10.0   10.0
15    0.0  10.0   10.0
17   20.0   0.0   20.0
19    0.0  10.0   10.0
20    0.0  10.0   10.0
22   30.0   0.0   30.0
35    0.0  10.0   10.0
All  50.0  50.0  100.0
```

Удаление столбца из DF (в одну сторону):

```
df2[["F","M"]]

Sex     F     M
Age
12    0.0  10.0
15    0.0  10.0
17   20.0   0.0
19    0.0  10.0
20    0.0  10.0
22   30.0   0.0
35    0.0  10.0
All  50.0  50.0
```

Панды тают, чтобы идти от широкого до длинного

```
>>> df
   ID  Year  Jan_salary  Feb_salary  Mar_salary
0   1  2016        4500        4200        4700
1   2  2016        3800        3600        4400
2   3  2016        5500        5200        5300

>>> melted_df = pd.melt(df,id_vars=['ID','Year'],
                        value_vars=['Jan_salary','Feb_salary','Mar_salary'],
                        var_name='month',value_name='salary')

>>> melted_df
   ID  Year       month  salary
0   1  2016  Jan_salary    4500
1   2  2016  Jan_salary    3800
2   3  2016  Jan_salary    5500
3   1  2016  Feb_salary    4200
4   2  2016  Feb_salary    3600
5   3  2016  Feb_salary    5200
6   1  2016  Mar_salary    4700
```

```
7   2  2016  Mar_salary    4400
8   3  2016  Mar_salary    5300

>>> melted_['month'] = melted_['month'].str.replace('_salary','')

>>> import calendar
>>> def mapper(month_abbr):
...     # from http://stackoverflow.com/a/3418092/42346
...     d = {v: str(k).zfill(2) for k,v in enumerate(calendar.month_abbr)}
...     return d[month_abbr]

>>> melted_df['month'] = melted_df['month'].apply(mapper)
>>> melted_df
   ID  Year month  salary
0   1  2016    01    4500
1   2  2016    01    3800
2   3  2016    01    5500
3   1  2016    02    4200
4   2  2016    02    3600
5   3  2016    02    5200
6   1  2016    03    4700
7   2  2016    03    4400
8   3  2016    03    5300
```

Разбить (изменить) CSV-строки в столбцах на несколько строк, имеющих один элемент в строке

```
import pandas as pd

df = pd.DataFrame([{'var1': 'a,b,c', 'var2': 1, 'var3': 'XX'},
                   {'var1': 'd,e,f,x,y', 'var2': 2, 'var3': 'ZZ'}])

print(df)

reshaped = \
(df.set_index(df.columns.drop('var1',1).tolist())
   .var1.str.split(',', expand=True)
   .stack()
   .reset_index()
   .rename(columns={0:'var1'})
   .loc[:, df.columns]
)

print(reshaped)
```

Выход:

```
        var1  var2 var3
0      a,b,c     1   XX
1  d,e,f,x,y     2   ZZ

  var1  var2 var3
0    a     1   XX
1    b     1   XX
2    c     1   XX
3    d     2   ZZ
4    e     2   ZZ
```

```
5    f     2   zz
6    x     2   zz
7    y     2   zz
```

Прочитайте Изменение формы и поворот онлайн: https://riptutorial.com/ru/pandas/topic/1463/изменение-формы-и-поворот

глава 20: Индексирование и выбор данных

Examples

Выбрать столбец по метке

```
# Create a sample DF
df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

# Show DF
df
          A         B         C
0 -0.467542  0.469146 -0.861848
1 -0.823205 -0.167087 -0.759942
2 -1.508202  1.361894 -0.166701
3  0.394143 -0.287349 -0.978102
4 -0.160431  1.054736 -0.785250

# Select column using a single label, 'A'
df['A']
0   -0.467542
1   -0.823205
2   -1.508202
3    0.394143
4   -0.160431

# Select multiple columns using an array of labels, ['A', 'C']
df[['A', 'C']]
          A         C
0 -0.467542 -0.861848
1 -0.823205 -0.759942
2 -1.508202 -0.166701
3  0.394143 -0.978102
4 -0.160431 -0.785250
```

Дополнительная информация: http://pandas.pydata.org/pandas-docs/version/0.18.0/indexing.html#selection-by-label

Выбрать по местоположению

Метод `iloc` (short for *integer location*) позволяет выбирать строки фрейма данных на основе их индекса местоположения. Таким образом можно срезать тактовые кадры так же, как с помощью списка разрезов на языке Python.

```
df = pd.DataFrame([[11, 22], [33, 44], [55, 66]], index=list("abc"))

df
# Out:
#     0   1
# a  11  22
# b  33  44
# c  55  66
```

```
df.iloc[0]  # the 0th index (row)
# Out:
# 0    11
# 1    22
# Name: a, dtype: int64

df.iloc[1]  # the 1st index (row)
# Out:
# 0    33
# 1    44
# Name: b, dtype: int64

df.iloc[:2] # the first 2 rows
#     0   1
# a  11  22
# b  33  44

df[::-1]    # reverse order of rows
#     0   1
# c  55  66
# b  33  44
# a  11  22
```

Расположение строк может быть объединено с расположением столбца

```
df.iloc[:, 1]  # the 1st column
# Out[15]:
# a    22
# b    44
# c    66
# Name: 1, dtype: int64
```

См. Также: Выбор по позиции

Нарезка этикетками

При использовании меток в результаты включены как начало, так и стоп.

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])

# Out:
#      A   B   C   D   E
# R0  99  78  61  16  73
# R1   8  62  27  30  80
# R2   7  76  15  53  80
# R3  27  44  77  75  65
# R4  47  30  84  86  18
```

Строки от `R0` до `R2` :

```
df.loc['R0':'R2']
# Out:
#      A   B   C   D   E
# R0   9  41  62   1  82
# R1  16  78   5  58   0
# R2  80   4  36  51  27
```

Обратите внимание, что `loc` отличается от `iloc` потому что `iloc` исключает конечный индекс

```
df.loc['R0':'R2'] # rows labelled R0, R1, R2
# Out:
#      A   B   C   D   E
# R0   9  41  62   1  82
# R1  16  78   5  58   0
# R2  80   4  36  51  27


# df.iloc[0:2] # rows indexed by 0, 1
#      A   B   C   D   E
# R0  99  78  61  16  73
# R1   8  62  27  30  80
```

Столбцы от `C` до `E` :

```
df.loc[:, 'C':'E']
# Out:
#      C   D   E
# R0  62   1  82
# R1   5  58   0
# R2  36  51  27
# R3  68  38  83
# R4   7  30  62
```

Выбор смешанной позиции и метки

DataFrame:

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])


df
Out[12]:
     A   B   C   D   E
R0  99  78  61  16  73
R1   8  62  27  30  80
R2   7  76  15  53  80
R3  27  44  77  75  65
R4  47  30  84  86  18
```

Выберите строки по положению и столбцы по метке:

```
df.ix[1:3, 'C':'E']
Out[19]:
     C   D   E
R1   5  58   0
R2  36  51  27
```

Если индекс является целым числом, `.ix` будет использовать метки, а не позиции:

```
df.index = np.arange(5, 10)

df
Out[22]:
    A   B   C   D   E
5   9  41  62   1  82
6  16  78   5  58   0
7  80   4  36  51  27
8  31   2  68  38  83
9  19  18   7  30  62

#same call returns an empty DataFrame because now the index is integer
df.ix[1:3, 'C':'E']
Out[24]:
Empty DataFrame
Columns: [C, D, E]
Index: []
```

Булевское индексирование

Можно выбрать строки и столбцы блока данных с помощью булевых массивов.

```
import pandas as pd
import numpy as np
np.random.seed(5)
df = pd.DataFrame(np.random.randint(100, size=(5, 5)), columns = list("ABCDE"),
                  index = ["R" + str(i) for i in range(5)])
print (df)
#      A   B   C   D   E
# R0  99  78  61  16  73
# R1   8  62  27  30  80
# R2   7  76  15  53  80
# R3  27  44  77  75  65
# R4  47  30  84  86  18
```

```
mask = df['A'] > 10
print (mask)
# R0     True
# R1    False
# R2    False
# R3     True
# R4     True
# Name: A, dtype: bool

print (df[mask])
#      A   B   C   D   E
# R0  99  78  61  16  73
# R3  27  44  77  75  65
# R4  47  30  84  86  18
```

```
print (df.ix[mask, 'C'])
# R0    61
# R3    77
# R4    84
# Name: C, dtype: int32

print(df.ix[mask, ['C', 'D']])
#       C   D
# R0  61  16
# R3  77  75
# R4  84  86
```

Больше в документации pandas .

Фильтрация столбцов (выбор «интересный», удаление ненужных, использование RegEx и т. Д.)

сгенерировать образец DF

```
In [39]: df = pd.DataFrame(np.random.randint(0, 10, size=(5, 6)),
columns=['a10','a20','a25','b','c','d'])

In [40]: df
Out[40]:
   a10  a20  a25  b  c  d
0    2    3    7  5  4  7
1    3    1    5  7  2  6
2    7    4    9  0  8  7
3    5    8    8  9  6  8
4    8    1    0  4  4  9
```

показать столбцы, содержащие букву 'a'

```
In [41]: df.filter(like='a')
Out[41]:
   a10  a20  a25
0    2    3    7
1    3    1    5
2    7    4    9
3    5    8    8
4    8    1    0
```

показать столбцы с использованием фильтра RegEx `(b|c|d)` - `b` или `c` или `d` :

```
In [42]: df.filter(regex='(b|c|d)')
Out[42]:
   b  c  d
0  5  4  7
1  7  2  6
2  0  8  7
3  9  6  8
4  4  4  9
```

показать все столбцы , кроме тех , начиная с (другими словами удалять / удалить все столбцы , удовлетворяющие заданной RegEx) a

```
In [43]: df.ix[:, ~df.columns.str.contains('^a')]
Out[43]:
   b  c  d
0  5  4  7
1  7  2  6
2  0  8  7
3  9  6  8
4  4  4  9
```

Фильтрация / выбор строк с использованием метода `.query ()`

```
import pandas as pd
```

генерировать случайные DF

```
df = pd.DataFrame(np.random.randint(0,10,size=(10, 3)), columns=list('ABC'))

In [16]: print(df)
   A  B  C
0  4  1  4
1  0  2  0
2  7  8  8
3  2  1  9
4  7  3  8
5  4  0  7
6  1  5  5
7  6  7  8
8  6  7  3
9  6  4  5
```

выберите строки, где значения в столбце `A > 2` и значения

в столбце `B < 5`

```
In [18]: df.query('A > 2 and B < 5')
Out[18]:
   A  B  C
0  4  1  4
4  7  3  8
5  4  0  7
9  6  4  5
```

с использованием `.query()` с переменными для фильтрации

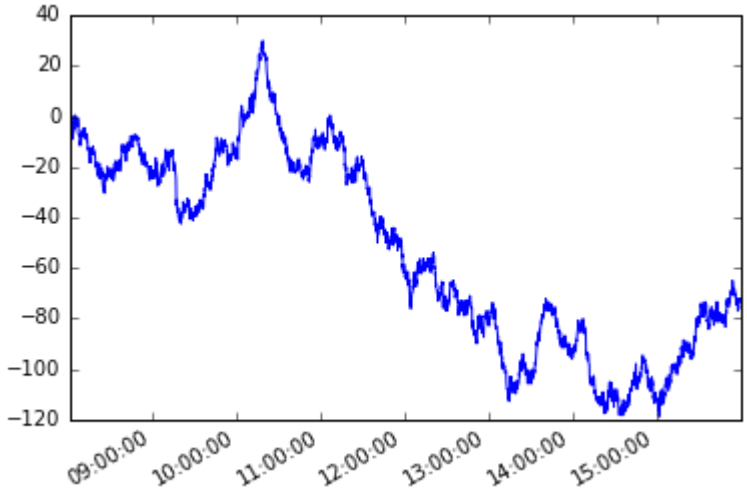
```
In [23]: B_filter = [1,7]

In [24]: df.query('B == @B_filter')
Out[24]:
   A  B  C
0  4  1  4
3  2  1  9
7  6  7  8
8  6  7  3

In [25]: df.query('@B_filter in B')
Out[25]:
   A  B  C
0  4  1  4
```

Наклонная нарезка

Может возникнуть необходимость пересекать элементы серии или строки кадра данных таким образом, что следующий элемент или следующая строка зависит от ранее выбранного элемента или строки. Это называется зависимостью пути.

Рассмотрим следующие временные ряды `s` с нерегулярной частотой.

```
#starting python community conventions
import numpy    as np
import pandas   as pd

# n is number of observations
n = 5000

day = pd.to_datetime(['2013-02-06'])
# irregular seconds spanning 28800 seconds (8 hours)
seconds = np.random.rand(n) * 28800 * pd.Timedelta(1, 's')
# start at 8 am
start = pd.offsets.Hour(8)
# irregular timeseries
tidx = day + start + seconds
tidx = tidx.sort_values()

s = pd.Series(np.random.randn(n), tidx, name='A').cumsum()
s.plot();
```



Предположим, что условие зависит от пути. Начиная с первого члена серии, я хочу захватить каждый последующий элемент таким образом, чтобы абсолютная разница между этим элементом и текущим элементом была больше или равна `x` .

Мы решим эту проблему, используя генераторы python.

Функция генератора

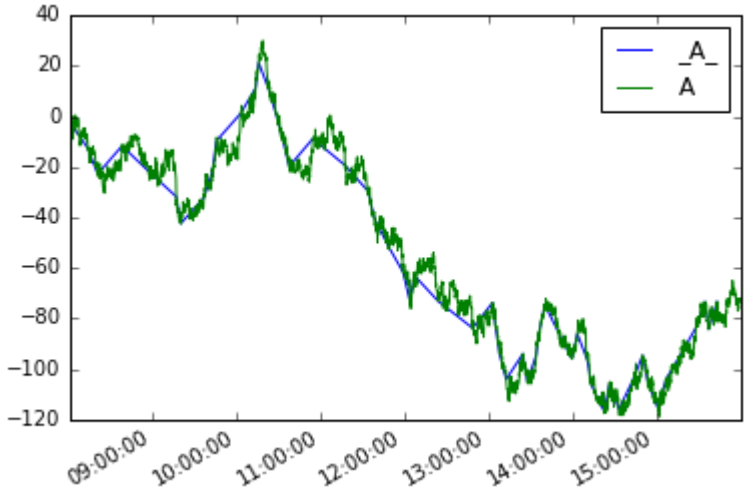
```
def mover(s, move_size=10):
    """Given a reference, find next value with
    an absolute difference >= move_size"""
    ref = None
    for i, v in s.iteritems():
        if ref is None or (abs(ref - v) >= move_size):
            yield i, v
            ref = v
```

Тогда мы можем определить, что новая серия `moves` так

```
moves = pd.Series({i:v for i, v in mover(s, move_size=10)},
                  name='_{}_'.format(s.name))
```

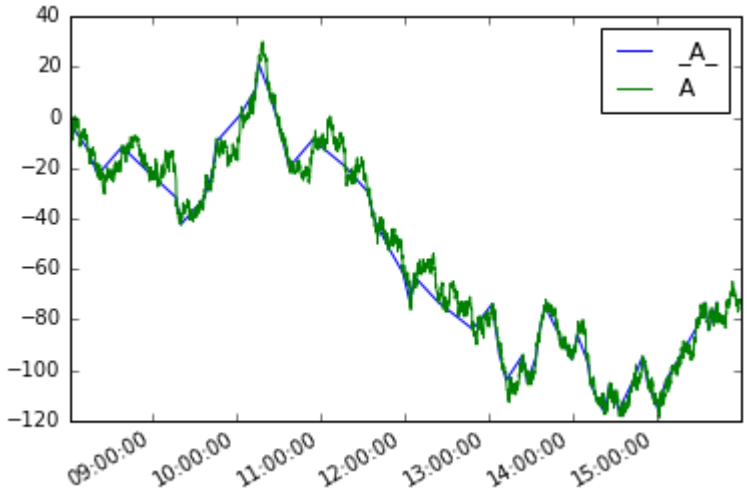
Построение их обоих

```
moves.plot(legend=True)
s.plot(legend=True)
```



Аналогом для data-кадров будет:

```
def mover_df(df, col, move_size=2):
    ref = None
    for i, row in df.iterrows():
        if ref is None or (abs(ref - row.loc[col]) >= move_size):
            yield row
            ref = row.loc[col]

df = s.to_frame()
moves_df = pd.concat(mover_df(df, 'A', 10), axis=1).T

moves_df.A.plot(label='_A_', legend=True)
df.A.plot(legend=True)
```



Получить первые / последние n строк кадра данных

Чтобы просмотреть первые или последние несколько записей фрейма данных, вы можете использовать методы `head` и `tail`

Чтобы вернуть первые n строк, используйте `DataFrame.head([n])`

```
df.head(n)
```

Чтобы вернуть последние n строк, используйте `DataFrame.tail([n])`

```
df.tail(n)
```

Без аргумента n эти функции возвращают 5 строк.

Обратите внимание, что обозначение среза для `head` / `tail` будет:

```
df[:10]  # same as df.head(10)
df[-10:] # same as df.tail(10)
```

Выбор отдельных строк в кадре данных

Позволять

```
df = pd.DataFrame({'col_1':['A','B','A','B','C'], 'col_2':[3,4,3,5,6]})
df
# Output:
#   col_1  col_2
# 0     A      3
# 1     B      4
# 2     A      3
# 3     B      5
# 4     C      6
```

Чтобы получить отдельные значения в `col_1` вы можете использовать `Series.unique()`

```
df['col_1'].unique()
# Output:
# array(['A', 'B', 'C'], dtype=object)
```

Но Series.unique () работает только для одного столбца.

Для имитации *выбора уникального col_1, col_2* SQL вы можете использовать `DataFrame.drop_duplicates()` :

```
df.drop_duplicates()
#   col_1  col_2
# 0     A      3
# 1     B      4
# 3     B      5
# 4     C      6
```

Это даст вам все уникальные строки в области данных. Так что если

```
df = pd.DataFrame({'col_1':['A','B','A','B','C'], 'col_2':[3,4,3,5,6],
'col_3':[0,0.1,0.2,0.3,0.4]})
df
```

```
# Output:
#   col_1  col_2  col_3
# 0     A      3    0.0
# 1     B      4    0.1
# 2     A      3    0.2
# 3     B      5    0.3
# 4     C      6    0.4

df.drop_duplicates()
#   col_1  col_2  col_3
# 0     A      3    0.0
# 1     B      4    0.1
# 2     A      3    0.2
# 3     B      5    0.3
# 4     C      6    0.4
```

Чтобы указать столбцы, которые следует учитывать при выборе уникальных записей, передайте их в качестве аргументов

```
df = pd.DataFrame({'col_1':['A','B','A','B','C'], 'col_2':[3,4,3,5,6],
'col_3':[0,0.1,0.2,0.3,0.4]})
df.drop_duplicates(['col_1','col_2'])
# Output:
#   col_1  col_2  col_3
# 0     A      3    0.0
# 1     B      4    0.1
# 3     B      5    0.3
# 4     C      6    0.4

# skip last column
# df.drop_duplicates(['col_1','col_2'])[['col_1','col_2']]
#   col_1  col_2
# 0     A      3
# 1     B      4
# 3     B      5
# 4     C      6
```

Источник: как «выбрать отдельный» для нескольких столбцов фрейма данных в пандах? ,

Отфильтруйте строки с отсутствующими данными (NaN, None, NaT)

Если у вас есть dataframe с отсутствующими данными (`NaN` , `pd.NaT` , `None`), вы можете отфильтровать неполные строки

```
df = pd.DataFrame([[0,1,2,3],
                   [None,5,None,pd.NaT],
                   [8,None,10,None],
                   [11,12,13,pd.NaT]],columns=list('ABCD'))
df
# Output:
#      A   B   C     D
# 0    0   1   2     3
# 1  NaN   5 NaN   NaT
# 2    8 NaN  10  None
# 3   11  12  13   NaT
```

`DataFrame.dropna` все строки, содержащие хотя бы одно поле с отсутствующими данными

```
df.dropna()
# Output:
#    A  B  C  D
# 0  0  1  2  3
```

Чтобы просто удалить строки, в которых отсутствуют данные в указанных столбцах, используйте `subset`

```
df.dropna(subset=['C'])
# Output:
#      A   B   C     D
# 0    0   1   2     3
# 2    8 NaN  10  None
# 3   11  12  13   NaT
```

Используйте параметр `inplace = True` для замены на месте фильтрованным фреймом.

Прочитайте Индексирование и выбор данных онлайн:
https://riptutorial.com/ru/pandas/topic/1751/индексирование-и-выбор-данных

глава 21: Инструменты ввода-вывода Pandas (считывание и сохранение наборов данных)

замечания

Официальная документация pandas включает страницу в IO Tools со списком соответствующих функций для чтения и записи в файлы, а также некоторые примеры и общие параметры.

Examples

Чтение csv-файла в DataFrame

Пример для чтения файла `data_file.csv` например:

Файл:

```
index,header1,header2,header3
1,str_data,12,1.4
3,str_data,22,42.33
4,str_data,2,3.44
2,str_data,43,43.34

7, str_data, 25, 23.32
```

Код:

```
pd.read_csv('data_file.csv')
```

Выход:

```
   index    header1  header2  header3
0      1   str_data       12     1.40
1      3   str_data       22    42.33
2      4   str_data        2     3.44
3      2   str_data       43    43.34
4      7   str_data       25    23.32
```

Некоторые полезные аргументы:

- **`sep`** Поле по умолчанию разделителем является запятая `,` . Используйте эту опцию, если вам нужен другой разделитель, например `pd.read_csv('data_file.csv', sep=';')`

- **`index_col`** С `index_col = n` (`n` целым числом) вы указываете pandas использовать столбец `n` для индексации DataFrame. В приведенном выше примере:

  ```
  pd.read_csv('data_file.csv',  index_col=0)
  ```

  Выход:

  ```
            header1  header2  header3
  index
   1       str_data       12     1.40
   3       str_data       22    42.33
   4       str_data        2     3.44
   2       str_data       43    43.34
   7       str_data       25    23.32
  ```

- **`skip_blank_lines`** По умолчанию пустые строки пропускаются. Используйте `skip_blank_lines=False` чтобы включить пустые строки (они будут заполнены значениями `NaN`)

  ```
  pd.read_csv('data_file.csv',  index_col=0,skip_blank_lines=False)
  ```

  Выход:

  ```
           header1  header2  header3
  index
   1      str_data       12     1.40
   3      str_data       22    42.33
   4      str_data        2     3.44
   2      str_data       43    43.34
  NaN          NaN      NaN      NaN
   7      str_data       25    23.32
  ```

- **`parse_dates`** Используйте этот параметр для анализа данных даты.

  Файл:

  ```
  date_begin;date_end;header3;header4;header5
  1/1/2017;1/10/2017;str_data;1001;123,45
  2/1/2017;2/10/2017;str_data;1001;67,89
  3/1/2017;3/10/2017;str_data;1001;0
  ```

  Код для разбора столбцов `0` и `1` качестве дат:

  ```
  pd.read_csv('f.csv', sep=';', parse_dates=[0,1])
  ```

  Выход:

```
  date_begin   date_end   header3  header4 header5
0 2017-01-01 2017-01-10  str_data     1001  123,45
1 2017-02-01 2017-02-10  str_data     1001   67,89
2 2017-03-01 2017-03-10  str_data     1001       0
```

По умолчанию выводится формат даты. Если вы хотите указать формат даты, который вы можете использовать, например

```
dateparse = lambda x: pd.datetime.strptime(x, '%d/%m/%Y')
pd.read_csv('f.csv', sep=';',parse_dates=[0,1],date_parser=dateparse)
```

Выход:

```
  date_begin   date_end   header3  header4 header5
0 2017-01-01 2017-10-01  str_data     1001  123,45
1 2017-01-02 2017-10-02  str_data     1001   67,89
2 2017-01-03 2017-10-03  str_data     1001       0
```

Более подробную информацию о параметрах функции можно найти в официальной документации .

Базовая загрузка в файл csv

```
raw_data = {'first_name': ['John', 'Jane', 'Jim'],
            'last_name': ['Doe', 'Smith', 'Jones'],
            'department': ['Accounting', 'Sales', 'Engineering'],}
df = pd.DataFrame(raw_data,columns=raw_data.keys())
df.to_csv('data_file.csv')
```

Синхронизация дат при чтении из csv

Вы можете указать столбец, содержащий даты, поэтому панды будут автоматически анализировать их при чтении из csv

```
pandas.read_csv('data_file.csv', parse_dates=['date_column'])
```

Электронная таблица для данных DataFrames

```
with pd.ExcelFile('path_to_file.xls) as xl:
    d = {sheet_name: xl.parse(sheet_name) for sheet_name in xl.sheet_names}
```

Прочтите конкретный лист

```
pd.read_excel('path_to_file.xls', sheetname='Sheet1')
```

Существует много вариантов синтаксического анализа для `read_excel` *(аналогично параметрам* `read_csv` *.*

```
pd.read_excel('path_to_file.xls',
              sheetname='Sheet1', header=[0, 1, 2],
              skiprows=3, index_col=0)  # etc.
```

Тестирование read_csv

```
import pandas as pd
import io

temp=u"""index; header1; header2; header3
1; str_data; 12; 1.4
3; str_data; 22; 42.33
4; str_data; 2; 3.44
2; str_data; 43; 43.34
7; str_data; 25; 23.32"""
#after testing replace io.StringIO(temp) to filename
df = pd.read_csv(io.StringIO(temp),
                 sep = ';',
                 index_col = 0,
                 skip_blank_lines = True)
print (df)
         header1   header2   header3
index
1       str_data        12      1.40
3       str_data        22     42.33
4       str_data         2      3.44
2       str_data        43     43.34
7       str_data        25     23.32
```

Учет списка

Все файлы находятся в папке `files` . Сначала создайте список DataFrames, а затем `concat` их:

```
import pandas as pd
import glob

#a.csv
#a,b
#1,2
#5,8

#b.csv
#a,b
#9,6
#6,4

#c.csv
#a,b
#4,3
#7,0

files = glob.glob('files/*.csv')
dfs = [pd.read_csv(fp) for fp in files]
```

```
#duplicated index inherited from each Dataframe
```

```
df = pd.concat(dfs)
print (df)
   a  b
0  1  2
1  5  8
0  9  6
1  6  4
0  4  3
1  7  0
#'reseting' index
df = pd.concat(dfs, ignore_index=True)
print (df)
   a  b
0  1  2
1  5  8
2  9  6
3  6  4
4  4  3
5  7  0
#concat by columns
df1 = pd.concat(dfs, axis=1)
print (df1)
   a  b  a  b  a  b
0  1  2  9  6  4  3
1  5  8  6  4  7  0
#reset column names
df1 = pd.concat(dfs, axis=1, ignore_index=True)
print (df1)
   0  1  2  3  4  5
0  1  2  9  6  4  3
1  5  8  6  4  7  0
```

Читайте в кусках

```
import pandas as pd

chunksize = [n]
for chunk in pd.read_csv(filename, chunksize=chunksize):
    process(chunk)
    delete(chunk)
```

Сохранить в CSV-файле

Сохранить с параметрами по умолчанию:

```
df.to_csv(file_name)
```

Напишите конкретные столбцы:

```
df.to_csv(file_name, columns =['col'])
```

Разделитель Difault - это «,» - изменить его:

```
df.to_csv(file_name,sep="|")
```

Запись без заголовка:

```
df.to_csv(file_name, header=False)
```

Напишите с заданным заголовком:

```
df.to_csv(file_name, header = ['A','B','C',...]
```

Чтобы использовать конкретную кодировку (например, «utf-8»), используйте аргумент кодировки:

df.to_csv (имя_файла, encoding = 'utf-8')

Синхронизация столбцов даты с read_csv

Дата всегда имеет другой формат, они могут быть проанализированы с использованием определенной функции parse_dates.

Этот *input.csv* :

```
2016 06 10 20:30:00    foo
2016 07 11 19:45:30    bar
2013 10 12 4:30:00     foo
```

Может быть проанализирован следующим образом:

```
mydateparser = lambda x: pd.datetime.strptime(x, "%Y %m %d %H:%M:%S")
df = pd.read_csv("file.csv", sep='\t', names=['date_column', 'other_column'],
parse_dates=['date_column'], date_parser=mydateparser)
```

Параметр parse_dates - это столбец для анализа
date_parser - это функция парсера

Чтение и объединение нескольких файлов CSV (с той же структурой) в один DF

```
import os
import glob
import pandas as pd

def get_merged_csv(flist, **kwargs):
    return pd.concat([pd.read_csv(f, **kwargs) for f in flist], ignore_index=True)

path = 'C:/Users/csvfiles'
```

```
fmask = os.path.join(path, '*mask*.csv')

df = get_merged_csv(glob.glob(fmask), index_col=None, usecols=['col1', 'col3'])

print(df.head())
```

Если вы хотите объединить CSV-файлы по горизонтали (добавление столбцов), используйте функцию `axis=1` при вызове функции `pd.concat()` :

```
def merged_csv_horizontally(flist, **kwargs):
    return pd.concat([pd.read_csv(f, **kwargs) for f in flist], axis=1)
```

Чтение cvs-файла в кадре данных pandas, когда нет строки заголовка

Если файл не содержит строку заголовка,

Файл:

```
1;str_data;12;1.4
3;str_data;22;42.33
4;str_data;2;3.44
2;str_data;43;43.34

7; str_data; 25; 23.32
```

вы можете использовать `names` ключевых слов для предоставления имен столбцов:

```
df = pandas.read_csv('data_file.csv', sep=';', index_col=0,
                     skip_blank_lines=True, names=['a', 'b', 'c'])

df
Out:
           a   b      c
1   str_data  12   1.40
3   str_data  22  42.33
4   str_data   2   3.44
2   str_data  43  43.34
7   str_data  25  23.32
```

Использование HDFStore

```
import string
import numpy as np
import pandas as pd
```

генерировать образец DF с различными типами

```
df = pd.DataFrame({
     'int32':    np.random.randint(0, 10**6, 10),
     'int64':    np.random.randint(10**7, 10**9, 10).astype(np.int64)*10,
     'float':    np.random.rand(10),
     'string':   np.random.choice([c*10 for c in string.ascii_uppercase], 10),
     })

In [71]: df
Out[71]:
      float   int32       int64      string
0  0.649978  848354  5269162190  DDDDDDDDDD
1  0.346963  490266  6897476700  OOOOOOOOOO
2  0.035069  756373  6711566750  ZZZZZZZZZZ
3  0.066692  957474  9085243570  FFFFFFFFFF
4  0.679182  665894  3750794810  MMMMMMMMMM
5  0.861914  630527  6567684430  TTTTTTTTTT
6  0.697691  825704  8005182860  FFFFFFFFFF
7  0.474501  942131  4099797720  QQQQQQQQQQ
8  0.645817  951055  8065980030  VVVVVVVVVV
9  0.083500  349709  7417288920  EEEEEEEEEE
```

сделать больше DF (10 * 100.000 = 1.000.000 строк)

```
df = pd.concat([df] * 10**5, ignore_index=True)
```

создать (или открыть существующий) файл HDFStore

```
store = pd.HDFStore('d:/temp/example.h5')
```

сохраните наш фрейм данных в файл h5 (HDFStore), индексируя столбцы [int32, int64, string]:

```
store.append('store_key', df, data_columns=['int32','int64','string'])
```

показать подробности HDFStore

```
In [78]: store.get_storer('store_key').table
Out[78]:
/store_key/table (Table(10,)) ''
  description := {
  "index": Int64Col(shape=(), dflt=0, pos=0),
  "values_block_0": Float64Col(shape=(1,), dflt=0.0, pos=1),
  "int32": Int32Col(shape=(), dflt=0, pos=2),
  "int64": Int64Col(shape=(), dflt=0, pos=3),
  "string": StringCol(itemsize=10, shape=(), dflt=b'', pos=4)}
  byteorder := 'little'
  chunkshape := (1724,)
  autoindex := True
  colindexes := {
    "index": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "int32": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "string": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "int64": Index(6, medium, shuffle, zlib(1)).is_csi=False}
```

показывать индексированные столбцы

```
In [80]: store.get_storer('store_key').table.colindexes
Out[80]:
{
    "int32": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "index": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "string": Index(6, medium, shuffle, zlib(1)).is_csi=False,
    "int64": Index(6, medium, shuffle, zlib(1)).is_csi=False}
```

закрыть (flush to disk) наш файл магазина

```
store.close()
```

Чтение журнала доступа Nginx (несколько кавычек)

Для нескольких кавычек используйте regex вместо sep:

```
df = pd.read_csv(log_file,
              sep=r'\s(?=(?:[^"]*"[^"]*")*[^"]*$)(?![^\[]*\])',
              engine='python',
              usecols=[0, 3, 4, 5, 6, 7, 8],
              names=['ip', 'time', 'request', 'status', 'size', 'referer', 'user_agent'],
              na_values='-',
              header=None
                )
```

Прочитайте Инструменты ввода-вывода Pandas (считывание и сохранение наборов данных) онлайн: https://riptutorial.com/ru/pandas/topic/2896/инструменты-ввода-вывода-pandas--считывание-и-сохранение-наборов-данных-

глава 22: Использование .ix, .iloc, .loc, .at и .iat для доступа к DataFrame

Examples

Использование .iloc

.iloc использует целые числа для чтения и записи данных в DataFrame.

Во-первых, давайте создадим DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],
                   'two': [6, 7, 8, 9, 10],
                }, index=['a', 'b', 'c', 'd', 'e'])
```

Этот DataFrame выглядит так:

```
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
```

Теперь мы можем использовать .iloc для чтения и записи значений. Давайте прочитаем первую строку, первый столбец:

```
print df.iloc[0, 0]
```

Это напечатает:

```
1
```

Мы также можем установить значения. Позволяет установить второй столбец, второй ряд, на что-то новое:

```
df.iloc[1, 1] = '21'
```

А потом посмотрите, что произошло:

```
print df

   one two
a    1   6
b    2  21
```

```
c    3   8
d    4   9
e    5  10
```

Использование .loc

.loc использует **метки** для чтения и записи данных.

Давайте настроим DataFrame:

```
df = pd.DataFrame({'one': [1, 2, 3, 4, 5],
                   'two': [6, 7, 8, 9, 10],
                  }, index=['a', 'b', 'c', 'd', 'e'])
```

Затем мы можем напечатать DataFrame, чтобы посмотреть на форму:

```
print df
```

Это приведет к выводу

```
   one  two
a    1    6
b    2    7
c    3    8
d    4    9
e    5   10
```

Мы используем **метки** столбцов и строк для доступа к данным с помощью .loc. Давайте установим строку 'c', столбец 'two' в значение 33:

```
df.loc['c', 'two'] = 33
```

Вот как выглядит DataFrame:

```
   one  two
a    1    6
b    2    7
c    3   33
d    4    9
e    5   10
```

Следует отметить, что использование `df['two'].loc['c'] = 33` может не сообщать о предупреждении и даже работать, однако использование `df.loc['c', 'two']` гарантированно работает правильно , а первое - нет.

Мы можем считывать фрагменты данных, например

```
print df.loc['a':'c']
```

будет печатать строки от a до c. Это включено.

```
   one  two
a    1    6
b    2    7
c    3    8
```

И, наконец, мы можем сделать оба вместе:

```
print df.loc['b':'d', 'two']
```

Выведет строки b в c столбца «два». Обратите внимание, что метка столбца не печатается.

```
b    7
c    8
d    9
```

Если .loc поставляется с целым аргументом, который не является меткой, он возвращается к целочисленной индексации осей (поведение .iloc). Это позволяет использовать смешанную метку и целую индексацию:

```
df.loc['b', 1]
```

вернет значение во втором столбце (индекс, начинающийся с 0) в строке «b»:

```
7
```

Прочитайте Использование .ix, .iloc, .loc, .at и .iat для доступа к DataFrame онлайн: https://riptutorial.com/ru/pandas/topic/7074/использование--ix---iloc---loc---at-и--iat-для-доступа-к-dataframe

глава 23: Категориальные данные

Вступление

Категориями являются тип данных панд, которые соответствуют статистическим переменным в статистике: переменная, которая может принимать только ограниченное и обычно фиксированное количество возможных значений (категорий; уровней в R). Примерами являются пол, социальный класс, типы крови, принадлежность страны, время наблюдения или рейтинги через шкалы Ликерта. Источник: Pandas Docs

Examples

Создание объекта

```
In [188]: s = pd.Series(["a","b","c","a","c"], dtype="category")

In [189]: s
Out[189]:
0    a
1    b
2    c
3    a
4    c
dtype: category
Categories (3, object): [a, b, c]

In [190]: df = pd.DataFrame({"A":["a","b","c","a", "c"]})

In [191]: df["B"] = df["A"].astype('category')

In [192]: df["C"] = pd.Categorical(df["A"])

In [193]: df
Out[193]:
   A  B  C
0  a  a  a
1  b  b  b
2  c  c  c
3  a  a  a
4  c  c  c

In [194]: df.dtypes
Out[194]:
A      object
B    category
C    category
dtype: object
```

Создание больших случайных наборов данных

```
In [1]: import pandas as pd
        import numpy as np

In [2]: df = pd.DataFrame(np.random.choice(['foo','bar','baz'], size=(100000,3)))
        df = df.apply(lambda col: col.astype('category'))

In [3]: df.head()
Out[3]:
     0    1    2
0  bar  foo  baz
1  baz  bar  baz
2  foo  foo  bar
3  bar  baz  baz
4  foo  bar  baz

In [4]: df.dtypes
Out[4]:
0    category
1    category
2    category
dtype: object

In [5]: df.shape
Out[5]: (100000, 3)
```

Прочитайте Категориальные данные онлайн: https://riptutorial.com/ru/pandas/topic/3887/категориальные-данные

глава 24: мультииндексных

Examples

Выбрать из MultiIndex by Level

Учитывая следующий DataFrame:

```
In [11]: df = pd.DataFrame(np.random.randn(6, 3), columns=['A', 'B', 'C'])

In [12]: df.set_index(['A', 'B'], inplace=True)

In [13]: df
Out[13]:
                           C
A         B
 0.902764 -0.259656 -1.864541
-0.695893  0.308893  0.125199
 1.696989 -1.221131 -2.975839
-1.132069 -1.086189 -1.945467
 2.294835 -1.765507  1.567853
-1.788299  2.579029  0.792919
```

Получите значения `A` по имени:

```
In [14]: df.index.get_level_values('A')
Out[14]:
Float64Index([0.902764041011, -0.69589264969,  1.69698924476, -1.13206872067,
               2.29483481146,   -1.788298829],
             dtype='float64', name='A')
```

Или по количеству уровней:

```
In [15]: df.index.get_level_values(level=0)
Out[15]:
Float64Index([0.902764041011, -0.69589264969,  1.69698924476, -1.13206872067,
               2.29483481146,   -1.788298829],
             dtype='float64', name='A')
```

И для определенного диапазона:

```
In [16]: df.loc[(df.index.get_level_values('A') > 0.5) & (df.index.get_level_values('A') <
2.1)]
Out[16]:
                           C
A        B
0.902764 -0.259656 -1.864541
1.696989 -1.221131 -2.975839
```

Диапазон также может включать несколько столбцов:

```
In [17]: df.loc[(df.index.get_level_values('A') > 0.5) & (df.index.get_level_values('B') < 0)]
Out[17]:
                            C
A        B
0.902764 -0.259656 -1.864541
1.696989 -1.221131 -2.975839
2.294835 -1.765507  1.567853
```

Чтобы извлечь конкретное значение, вы можете использовать xs (поперечное сечение):

```
In [18]: df.xs(key=0.9027639999999999)
Out[18]:
                  C
B
-0.259656 -1.864541

In [19]: df.xs(key=0.9027639999999999, drop_level=False)
Out[19]:
                            C
A        B
0.902764 -0.259656 -1.864541
```

Итерация над DataFrame с помощью MultiIndex

Учитывая следующий DataFrame:

```
In [11]: df = pd.DataFrame({'a':[1,1,1,2,2,3],'b':[4,4,5,5,6,7,],'c':[10,11,12,13,14,15]})

In [12]: df.set_index(['a','b'], inplace=True)

In [13]: df
Out[13]:
      c
a b
1 4  10
  4  11
  5  12
2 5  13
  6  14
3 7  15
```

Вы можете выполнять итерацию на любом уровне MultiIndex. Например, `level=0` (вы также можете выбрать уровень по имени eg `level='a'`):

```
In[21]: for idx, data in df.groupby(level=0):
            print('---')
            print(data)
---
      c
a b
1 4  10
  4  11
  5  12
---
      c
```

```
a b
2 5  13
  6  14
---
      c
a b
3 7  15
```

Вы также можете выбрать уровни по имени, например `level = 'b':

```
In[22]: for idx, data in df.groupby(level='b'):
            print('---')
            print(data)
---
      c
a b
1 4  10
  4  11
---
      c
a b
1 5  12
2 5  13
---
      c
a b
2 6  14
---
      c
a b
3 7  15
```

Настройка и сортировка MultiIndex

В этом примере показано, как использовать данные столбцов для установки `MultiIndex` в `pandas.DataFrame` .

```
In [1]: df = pd.DataFrame([['one', 'A', 100], ['two', 'A', 101], ['three', 'A', 102],
   ...:                    ['one', 'B', 103], ['two', 'B', 104], ['three', 'B', 105]],
   ...:                   columns=['c1', 'c2', 'c3'])


In [2]: df
Out[2]:
      c1 c2   c3
0    one  A  100
1    two  A  101
2  three  A  102
3    one  B  103
4    two  B  104
5  three  B  105



In [3]: df.set_index(['c1', 'c2'])
Out[3]:
           c3
```

```
c1    c2
one   A   100
two   A   101
three A   102
one   B   103
two   B   104
three B   105
```

Вы можете отсортировать индекс сразу после его установки:

```
In [4]: df.set_index(['c1', 'c2']).sort_index()
Out[4]:
           c3
c1    c2
one   A   100
      B   103
three A   102
      B   105
two   A   101
      B   104
```

Имея отсортированный индекс, вы получите несколько более эффективный поиск на первом уровне:

```
In [5]: df_01 = df.set_index(['c1', 'c2'])

In [6]: %timeit df_01.loc['one']
1000 loops, best of 3: 607 µs per loop


In [7]: df_02 = df.set_index(['c1', 'c2']).sort_index()

In [8]: %timeit df_02.loc['one']
1000 loops, best of 3: 413 µs per loop
```

После того, как индекс установлен, вы можете выполнять поиск для определенных записей или групп записей:

```
In [9]: df_indexed = df.set_index(['c1', 'c2']).sort_index()

In [10]: df_indexed.loc['one']
Out[10]:
     c3
c2
A   100
B   103


In [11]: df_indexed.loc['one', 'A']
Out[11]:
c3    100
Name: (one, A), dtype: int64


In [12]: df_indexed.xs((slice(None), 'A'))
Out[12]:
```

```
        c3
c1
one    100
three  102
two    101
```

Как изменить столбцы MultiIndex на стандартные столбцы

Учитывая DataFrame с столбцами MultiIndex

```
# build an example DataFrame
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x','y']], labels=[[1,1,0,],[1,0,1,]])
df = pd.DataFrame(np.random.randn(2,3), columns=midx)

In [2]: df
Out[2]:
        one                zero
          y         x         y
0  0.785806 -0.679039  0.513451
1 -0.337862 -0.350690 -1.423253
```

Если вы хотите изменить столбцы на стандартные столбцы (не MultiIndex), просто переименуйте столбцы.

```
df.columns = ['A','B','C']
In [3]: df
Out[3]:
          A         B         C
0  0.785806 -0.679039  0.513451
1 -0.337862 -0.350690 -1.423253
```

Как изменить стандартные столбцы на MultiIndex

Начните со стандартного DataFrame

```
df = pd.DataFrame(np.random.randn(2,3), columns=['a','b','c'])

In [91]: df
Out[91]:
          a         b         c
0 -0.911752 -1.405419 -0.978419
1  0.603888 -1.187064 -0.035883
```

Теперь, чтобы перейти на MultiIndex, создайте объект `MultiIndex` и назначьте его `df.columns` .

```
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x','y']], labels=[[1,1,0,],[1,0,1,]])
df.columns = midx

In [94]: df
Out[94]:
            one                zero
              y         x         y
```

```
    0 -0.911752 -1.405419 -0.978419
    1  0.603888 -1.187064 -0.035883
```

Столбцы MultiIndex

MultiIndex также может использоваться для создания DataFrames с многоуровневыми столбцами. Просто используйте `columns` ключевое слово в команде DataFrame.

```
midx = pd.MultiIndex(levels=[['zero', 'one'], ['x','y']], labels=[[1,1,0,],[1,0,1,]])
df = pd.DataFrame(np.random.randn(6,4), columns=midx)

In [86]: df
Out[86]:
        one                zero
          y         x         y
0  0.625695  2.149377  0.006123
1 -1.392909  0.849853  0.005477
```

Отображение всех элементов в индексе

Чтобы просмотреть все элементы в индексе, измените параметры печати, которые «разрешают» отображение MultiIndex.

```
pd.set_option('display.multi_sparse', False)
df.groupby(['A','B']).mean()
# Output:
#        C
# A B
# a 1  107
# a 2  102
# a 3  115
# b 5   92
# b 8   98
# c 2   87
# c 4  104
# c 9  123
```

Прочитайте мультииндексных онлайн: https://riptutorial.com/ru/pandas/topic/3840/мультииндексных

глава 25: Объединить, объединить и объединить

Синтаксис

- DataFrame. **merge** (*right, how = 'inner', on = None, left_on = None, right_on = None, left_index = False, right_index = False, sort = False, suffixes = ('_ x', '_y'), copy = True, индикатор = False*)
- Объедините объекты DataFrame, выполнив операцию объединения в стиле базы данных по столбцам или индексам.
- При объединении столбцов в столбцах индексы DataFrame будут игнорироваться. В противном случае при объединении индексов по индексам или индексам в столбце или столбцах индекс будет передан.

параметры

параметры	объяснение
право	DataFrame
как	{'left', 'right', 'outer', 'inner'}, default 'inner'
вышел на	ярлык или список или массив. Имена полей для объединения в левом DataFrame. Может быть вектором или списком векторов длины DataFrame для использования конкретного вектора в качестве ключа соединения вместо столбцов
Право на	ярлык или список или массив. Имена полей для присоединения в правом формате DataFrame или вектора / списка векторов на left_on docs
left_index	boolean, по умолчанию False. Используйте индекс из левого DataFrame в качестве ключа (ов) соединения. Если это MultiIndex, количество ключей в другом DataFrame (либо в индексе, либо в количестве столбцов) должно соответствовать количеству уровней
right_index	boolean, по умолчанию False. Используйте индекс из нужного DataFrame в качестве ключа соединения. Те же оговорки, что и left_index

параметры	объяснение
Сортировать	boolean, по умолчанию Fals. Сортируйте ключи соединения лексикографически в результате DataFrame
суффиксы	2-строчная последовательность (кортеж, список, ...). Суффикс применяется для перекрытия имен столбцов в левой и правой частях, соответственно
копия	boolean, по умолчанию True. Если False, не копируйте данные без необходимости
индикатор	boolean или string, по умолчанию False. Если True, добавляет столбец для вывода DataFrame с именем «_merge» с информацией об источнике каждой строки. Если строка, колонка с информацией об источнике каждой строки будет добавлена к выходу DataFrame, а столбцу будет присвоено значение string. Информационный столбец является категориальным и принимает значение «left_only» для наблюдений, чей ключ слияния появляется только в «левом» DataFrame, «right_only» для наблюдений, чей ключ слияния отображается только в «правильном» DataFrame и «both», если ключ слияния наблюдения находится в обоих.

Examples

сливаться

Например, даны две таблицы,

T1

```
id    x        y
8    42        1.9
9    30        1.9
```

T2

```
id    signal
8    55
8    56
8    59
9    57
9    58
9    60
```

Цель состоит в том, чтобы получить новую таблицу T3:

```
id    x        y        s1        s2        s3
```

```
8    42        1.9        55        56        58
9    30        1.9        57        58        60
```

Который должен создавать столбцы `s1` , `s2` и `s3` , каждый из которых соответствует строке (количество строк на `id` всегда фиксировано и равно 3)

Применяя `join` (который принимает необязательный аргумент, который может быть столбцом или несколькими именами столбцов, который указывает, что переданный DataFrame должен быть выровнен по этому столбцу в DataFrame). Таким образом, решение может быть таким, как показано ниже:

df = df1.merge (df2.groupby ('id') ['signal']. apply (lambda x: x.reset_index (drop = True)). unstack (). reset_index ())

```
df
Out[63]:
   id   x    y   0   1   2
0   8  42  1.9  55  56  59
1   9  30  1.9  57  58  60
```

Если я их разделю:

```
df2t = df2.groupby('id')['signal'].apply(lambda x:
x.reset_index(drop=True)).unstack().reset_index()

df2t
Out[59]:
   id   0   1   2
0   8  55  56  59
1   9  57  58  60

df = df1.merge(df2t)

df
Out[61]:
   id   x    y   0   1   2
0   8  42  1.9  55  56  59
1   9  30  1.9  57  58  60
```

Объединение двух DataFrames

```
In [1]: df1 = pd.DataFrame({'x': [1, 2, 3], 'y': ['a', 'b', 'c']})

In [2]: df2 = pd.DataFrame({'y': ['b', 'c', 'd'], 'z': [4, 5, 6]})

In [3]: df1
Out[3]:
   x  y
0  1  a
1  2  b
2  3  c

In [4]: df2
```

```
Out[4]:
   y  z
0  b  4
1  c  5
2  d  6
```

Внутреннее соединение:

Использует пересечение ключей из двух DataFrames.

```
In [5]: df1.merge(df2) # by default, it does an inner join on the common column(s)
Out[5]:
   x  y  z
0  2  b  4
1  3  c  5
```

В качестве альтернативы укажите пересечение ключей из двух Dataframes.

```
In [5]: merged_inner = pd.merge(left=df1, right=df2, left_on='y', right_on='y')
Out[5]:
   x  y  z
0  2  b  4
1  3  c  5
```

Внешнее соединение:

Использует объединение ключей из двух DataFrames.

```
In [6]: df1.merge(df2, how='outer')
Out[6]:
     x  y    z
0  1.0  a  NaN
1  2.0  b  4.0
2  3.0  c  5.0
3  NaN  d  6.0
```

Левое соединение:

Использует только ключи из левого DataFrame.

```
In [7]: df1.merge(df2, how='left')
Out[7]:
   x  y    z
0  1  a  NaN
1  2  b  4.0
2  3  c  5.0
```

Право Присоединиться

Использует только ключи от правого DataFrame.

```
In [8]: df1.merge(df2, how='right')
Out[8]:
     x  y  z
0  2.0  b  4
1  3.0  c  5
2  NaN  d  6
```

Объединение / объединение / объединение нескольких кадров данных (по горизонтали и по вертикали)

генерировать образцы данных:

```
In [57]: df3 = pd.DataFrame({'col1':[211,212,213], 'col2': [221,222,223]})

In [58]: df1 = pd.DataFrame({'col1':[11,12,13], 'col2': [21,22,23]})

In [59]: df2 = pd.DataFrame({'col1':[111,112,113], 'col2': [121,122,123]})

In [60]: df3 = pd.DataFrame({'col1':[211,212,213], 'col2': [221,222,223]})

In [61]: df1
Out[61]:
   col1  col2
0    11    21
1    12    22
2    13    23

In [62]: df2
Out[62]:
   col1  col2
0   111   121
1   112   122
2   113   123

In [63]: df3
Out[63]:
   col1  col2
0   211   221
1   212   222
2   213   223
```

объединить / объединить / объединить кадры данных [df1, df2, df3] по вертикали - добавить строки

```
In [64]: pd.concat([df1,df2,df3], ignore_index=True)
Out[64]:
   col1  col2
0    11    21
1    12    22
```

```
2    13    23
3   111   121
4   112   122
5   113   123
6   211   221
7   212   222
8   213   223
```

объединить / объединить / объединить кадры данных по горизонтали (выравнивание по индексу):

```
In [65]: pd.concat([df1,df2,df3], axis=1)
Out[65]:
   col1  col2  col1  col2  col1  col2
0    11    21   111   121   211   221
1    12    22   112   122   212   222
2    13    23   113   123   213   223
```

Объединить, присоединиться и присоединиться

Слияние имен ключей одинаково

```
pd.merge(df1, df2, on='key')
```

Слияние имен ключей различно

```
pd.merge(df1, df2, left_on='l_key', right_on='r_key')
```

Различные типы присоединения

```
pd.merge(df1, df2, on='key', how='left')
```

Объединение нескольких клавиш

```
pd.merge(df1, df2, on=['key1', 'key2'])
```

Обработка перекрывающихся столбцов

```
pd.merge(df1, df2, on='key', suffixes=('_left', '_right'))
```

Использование индекса строки вместо слияния ключей

```
pd.merge(df1, df2, right_index=True, left_index=True)
```

Избегайте использования синтаксиса `.join` поскольку он дает исключение для перекрывающихся столбцов

Объединение по левому индексу данных и правильному столбцу данных

```
pd.merge(df1, df2, right_index=True, left_on='l_key')
```

Конкретные информационные кадры

Клеены вертикально

```
pd.concat([df1, df2, df3], axis=0)
```

Клеены горизонтально

```
pd.concat([df1, df2, df3], axis=1)
```

В чем разница между объединением и объединением

Рассмотрим рамки данных `left` и `right`

```
left = pd.DataFrame([['a', 1], ['b', 2]], list('XY'), list('AB'))
left

   A  B
X  a  1
Y  b  2
```

```
right = pd.DataFrame([['a', 3], ['b', 4]], list('XY'), list('AC'))
right

   A  C
X  a  3
Y  b  4
```

join

Подумайте о том, чтобы `join` как хотите объединиться с dataframes на основе их соответствующих индексов. Если есть перекрывающиеся столбцы, `join` захочет, чтобы вы добавили суффикс к совпадающему имени столбца из левого фрейма. Наши два блока данных имеют перекрывающееся имя столбца `A`

```
left.join(right, lsuffix='_')

  A_  B  A  C
X  a  1  a  3
Y  b  2  b  4
```

Обратите внимание, что индекс сохраняется, и у нас есть 4 столбца. 2 столбца `left` и 2 `right` .

Если индексы не выровнены

```
left.join(right.reset_index(), lsuffix='_', how='outer')
```

```
    A_    B index    A    C
0  NaN  NaN     X    a  3.0
1  NaN  NaN     Y    b  4.0
X    a  1.0   NaN  NaN  NaN
Y    b  2.0   NaN  NaN  NaN
```

Я использовал внешнее соединение, чтобы лучше проиллюстрировать суть. Если индексы не выравниваются, результатом будет объединение индексов.

Мы можем сказать, что `join` к использованию конкретного столбца в левом фрейме данных для использования в качестве ключа соединения, но он по-прежнему будет использовать индекс справа.

```
left.reset_index().join(right, on='index', lsuffix='_')

  index A_  B  A  C
0     X  a  1  a  3
1     Y  b  2  b  4
```

merge

Подумайте о `merge` с выравниванием по столбцам. По умолчанию `merge` будет искать перекрывающиеся столбцы для объединения. `merge` дает лучший контроль над ключами слияния, позволяя пользователю указывать подмножество перекрывающихся столбцов для использования с параметром `on` или отдельно разрешать спецификацию столбцов слева и столбцов справа для слияния.

`merge` вернет комбинированный блок данных, в котором индекс будет уничтожен.

Этот простой пример находит перекрывающий столбец `'A'` и объединяет его на основе.

```
left.merge(right)

   A  B  C
0  a  1  3
1  b  2  4
```

Обратите внимание, что индекс `[0, 1]` и больше `['X', 'Y']`

Вы можете явно указать, что вы объединяетесь с индексом с параметром `left_index` или `right_index`

```
left.merge(right, left_index=True, right_index=True, suffixes=['_', ''])

  A_  B  A  C
X  a  1  a  3
Y  b  2  b  4
```

И это выглядит точно так же, как пример `join` выше.

Прочитайте Объединить, объединить и объединить онлайн:

https://riptutorial.com/ru/pandas/topic/1966/объединить--объединить-и-объединить

глава 26: Отсутствующие данные

замечания

Должны ли мы включать незарегистрированные `ffill` и `bfill` ?

Examples

Заполнение пропущенных значений

```
In [11]: df = pd.DataFrame([[1, 2, None, 3], [4, None, 5, 6],
                            [7, 8, 9, 10], [None, None, None, None]])

Out[11]:
     0    1    2     3
0  1.0  2.0  NaN   3.0
1  4.0  NaN  5.0   6.0
2  7.0  8.0  9.0  10.0
3  NaN  NaN  NaN   NaN
```

Заполните пропущенные значения одним значением:

```
In [12]: df.fillna(0)
Out[12]:
     0    1    2     3
0  1.0  2.0  0.0   3.0
1  4.0  0.0  5.0   6.0
2  7.0  8.0  9.0  10.0
3  0.0  0.0  0.0   0.0
```

Это возвращает новый DataFrame. Если вы хотите изменить оригинальный DataFrame, либо использовать `inplace` параметр (`df.fillna(0, inplace=True)`) или назначить его обратно в исходное DataFrame (`df = df.fillna(0)`).

Заполните пропущенные значения предыдущими:

```
In [13]: df.fillna(method='pad')  # this is equivalent to both method='ffill' and .ffill()
Out[13]:
     0    1    2     3
0  1.0  2.0  NaN   3.0
1  4.0  2.0  5.0   6.0
2  7.0  8.0  9.0  10.0
3  7.0  8.0  9.0  10.0
```

Заполните следующие поля:

```
In [14]: df.fillna(method='bfill')  # this is equivalent to .bfill()
Out[14]:
     0    1    2     3
0  1.0  2.0  5.0   3.0
1  4.0  8.0  5.0   6.0
2  7.0  8.0  9.0  10.0
3  NaN  NaN  NaN   NaN
```

Заполните с помощью другого DataFrame:

```
In [15]: df2 = pd.DataFrame(np.arange(100, 116).reshape(4, 4))
         df2
Out[15]:
     0    1    2    3
0  100  101  102  103
1  104  105  106  107
2  108  109  110  111
3  112  113  114  115

In [16]: df.fillna(df2) #  takes the corresponding cells in df2 to fill df
Out[16]:
       0      1      2      3
0    1.0    2.0  102.0    3.0
1    4.0  105.0    5.0    6.0
2    7.0    8.0    9.0   10.0
3  112.0  113.0  114.0  115.0
```

Удаление отсутствующих значений

При создании DataFrame `None` (отсутствующее значение python) преобразуется в `NaN` (отсутствует значение pandas):

```
In [11]: df = pd.DataFrame([[1, 2, None, 3], [4, None, 5, 6],
                            [7, 8, 9, 10], [None, None, None, None]])

Out[11]:
     0    1    2     3
0  1.0  2.0  NaN   3.0
1  4.0  NaN  5.0   6.0
2  7.0  8.0  9.0  10.0
3  NaN  NaN  NaN   NaN
```

Отбрасывайте строки, если хотя бы один столбец имеет отсутствующее значение

```
In [12]: df.dropna()
Out[12]:
     0    1    2     3
2  7.0  8.0  9.0  10.0
```

Это возвращает новый DataFrame. Если вы хотите изменить оригинальный DataFrame, либо

использовать `inplace` параметр (`df.dropna(inplace=True)`) или назначить его обратно в исходное DataFrame (`df = df.dropna()`).

Отбрасывайте строки, если все значения в этой строке отсутствуют.

```
In [13]: df.dropna(how='all')
Out[13]:
     0    1    2     3
0  1.0  2.0  NaN   3.0
1  4.0  NaN  5.0   6.0
2  7.0  8.0  9.0  10.0
```

Отбрасывать *столбцы,* которые не имеют как минимум 3 не пропущенных значений

```
In [14]: df.dropna(axis=1, thresh=3)
Out[14]:
     0     3
0  1.0   3.0
1  4.0   6.0
2  7.0  10.0
3  NaN   NaN
```

интерполирование

```
import pandas as pd
import numpy as np

df = pd.DataFrame({'A':[1,2,np.nan,3,np.nan],
                   'B':[1.2,7,3,0,8]})

df['C'] = df.A.interpolate()
df['D'] = df.A.interpolate(method='spline', order=1)

print (df)
     A    B    C         D
0  1.0  1.2  1.0  1.000000
1  2.0  7.0  2.0  2.000000
2  NaN  3.0  2.5  2.428571
3  3.0  0.0  3.0  3.000000
4  NaN  8.0  3.0  3.714286
```

Проверка отсутствующих значений

Чтобы проверить, является ли значение NaN, могут использоваться функции `isnull()` или `notnull()` .

```
In [1]: import numpy as np
```

```
In [2]: import pandas as pd
In [3]: ser = pd.Series([1, 2, np.nan, 4])
In [4]: pd.isnull(ser)
Out[4]:
0    False
1    False
2     True
3    False
dtype: bool
```

Обратите внимание, что `np.nan == np.nan` возвращает False, поэтому вам следует избегать сравнения с np.nan:

```
In [5]: ser == np.nan
Out[5]:
0    False
1    False
2    False
3    False
dtype: bool
```

Обе функции также определяются как методы для рядов и DataFrames.

```
In [6]: ser.isnull()
Out[6]:
0    False
1    False
2     True
3    False
dtype: bool
```

Тестирование на DataFrames:

```
In [7]: df = pd.DataFrame({'A': [1, np.nan, 3], 'B': [np.nan, 5, 6]})
In [8]: print(df)
Out[8]:
     A    B
0  1.0  NaN
1  NaN  5.0
2  3.0  6.0

In [9]: df.isnull()  # If the value is NaN, returns True.
Out[9]:
       A      B
0  False   True
1   True  False
2  False  False

In [10]: df.notnull()  # Opposite of .isnull(). If the value is not NaN, returns True.
Out[10]:
       A      B
0   True  False
1  False   True
2   True   True
```

Прочитайте Отсутствующие данные онлайн: https://riptutorial.com/ru/pandas/topic/1896/

отсутствующие-данные

глава 27: Получение информации о DataFrames

Examples

Получение данных и использования памяти DataFrame

Чтобы получить основную информацию о DataFrame, включая имена столбцов и типы данных:

```
import pandas as pd

df = pd.DataFrame({'integers': [1, 2, 3],
                   'floats': [1.5, 2.5, 3],
                   'text': ['a', 'b', 'c'],
                   'ints with None': [1, None, 3]})

df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 4 columns):
floats            3 non-null float64
integers          3 non-null int64
ints with None    2 non-null float64
text              3 non-null object
dtypes: float64(2), int64(1), object(1)
memory usage: 120.0+ bytes
```

Чтобы получить использование памяти DataFrame:

```
>>> df.info(memory_usage='deep')
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3 entries, 0 to 2
Data columns (total 4 columns):
floats            3 non-null float64
integers          3 non-null int64
ints with None    2 non-null float64
text              3 non-null object
dtypes: float64(2), int64(1), object(1)
memory usage: 234.0 bytes
```

Список имен столбцов DataFrame

```
df = pd.DataFrame({'a': [1, 2, 3], 'b': [4, 5, 6], 'c': [7, 8, 9]})
```

Чтобы перечислить имена столбцов в DataFrame:

```
>>> list(df)
['a', 'b', 'c']
```

Этот метод понимания списка особенно полезен при использовании отладчика:

```
>>> [c for c in df]
['a', 'b', 'c']
```

Это долгий путь:

```
sampledf.columns.tolist()
```

Вы также можете распечатать их как индекс, а не список (это будет не очень заметно для фреймов данных со многими столбцами):

```
df.columns
```

Различные сводные статистические данные Dataframe.

```
import pandas as pd
df = pd.DataFrame(np.random.randn(5, 5), columns=list('ABCDE'))
```

Чтобы создать различные сводные статистические данные. Для числовых значений - число не-NA / нулевых значений (`count`), среднее (`mean`), стандартное отклонение `std` и значения, известные как сводка с пятью цифрами :

- `min` : минимум (наименьшее наблюдение)
- `25%` : нижняя квартиль или первый квартиль (Q1)
- `50%` : медиана (среднее значение, Q2)
- `75%` : верхний квартиль или третий квартиль (Q3)
- `max` : максимум (наибольшее наблюдение)

```
>>> df.describe()
              A         B         C         D         E
count  5.000000  5.000000  5.000000  5.000000  5.000000
mean  -0.456917 -0.278666  0.334173  0.863089  0.211153
std    0.925617  1.091155  1.024567  1.238668  1.495219
min   -1.494346 -2.031457 -0.336471 -0.821447 -2.106488
25%   -1.143098 -0.407362 -0.246228 -0.087088 -0.082451
50%   -0.536503 -0.163950 -0.004099  1.509749  0.313918
75%    0.092630  0.381407  0.120137  1.822794  1.060268
max    0.796729  0.828034  2.137527  1.891436  1.870520
```

Прочитайте Получение информации о DataFrames онлайн:
https://riptutorial.com/ru/pandas/topic/6697/получение-информации-о-dataframes

глава 28: Праздничные календари

Examples

Создание пользовательского календаря

Вот как создать пользовательский календарь. Приведенный пример - французский календарь, поэтому он содержит много примеров.

```
from pandas.tseries.holiday import AbstractHolidayCalendar, Holiday, EasterMonday, Easter
from pandas.tseries.offsets import Day, CustomBusinessDay

class FrBusinessCalendar(AbstractHolidayCalendar):
    """ Custom Holiday calendar for France based on
        https://en.wikipedia.org/wiki/Public_holidays_in_France
      - 1 January: New Year's Day
      - Moveable: Easter Monday (Monday after Easter Sunday)
      - 1 May: Labour Day
      - 8 May: Victory in Europe Day
      - Moveable Ascension Day (Thursday, 39 days after Easter Sunday)
      - 14 July: Bastille Day
      - 15 August: Assumption of Mary to Heaven
      - 1 November: All Saints' Day
      - 11 November: Armistice Day
      - 25 December: Christmas Day
    """
    rules = [
        Holiday('New Years Day', month=1, day=1),
        EasterMonday,
        Holiday('Labour Day', month=5, day=1),
        Holiday('Victory in Europe Day', month=5, day=8),
        Holiday('Ascension Day', month=1, day=1, offset=[Easter(), Day(39)]),
        Holiday('Bastille Day', month=7, day=14),
        Holiday('Assumption of Mary to Heaven', month=8, day=15),
        Holiday('All Saints Day', month=11, day=1),
        Holiday('Armistice Day', month=11, day=11),
        Holiday('Christmas Day', month=12, day=25)
    ]
```

Использовать собственный календарь

Вот как использовать пользовательский календарь.

Получите отпуск между двумя датами

```
import pandas as pd
from datetime import date

# Creating some boundaries
year = 2016
```

```
start = date(year, 1, 1)
end = start + pd.offsets.MonthEnd(12)

# Creating a custom calendar
cal = FrBusinessCalendar()
# Getting the holidays (off-days) between two dates
cal.holidays(start=start, end=end)

# DatetimeIndex(['2016-01-01', '2016-03-28', '2016-05-01', '2016-05-05',
#                '2016-05-08', '2016-07-14', '2016-08-15', '2016-11-01',
#                '2016-11-11', '2016-12-25'],
#               dtype='datetime64[ns]', freq=None)
```

Подсчитайте количество рабочих дней между двумя датами

Иногда полезно получать количество рабочих дней в месяц независимо от года в будущем или в прошлом. Вот как это сделать с помощью настраиваемого календаря.

```
from pandas.tseries.offsets import CDay

# Creating a series of dates between the boundaries
# by using the custom calendar
se = pd.bdate_range(start=start,
                    end=end,
                    freq=CDay(calendar=cal)).to_series()
# Counting the number of working days by month
se.groupby(se.dt.month).count().head()

# 1    20
# 2    21
# 3    22
# 4    21
# 5    21
```

Прочитайте Праздничные календари онлайн: https://riptutorial.com/ru/pandas/topic/7976/праздничные-календари

глава 29: Простое управление DataFrames

Examples

Удаление столбца в DataFrame

Есть несколько способов удалить столбец в DataFrame.

```
import numpy as np
import pandas as pd

np.random.seed(0)

pd.DataFrame(np.random.randn(5, 6), columns=list('ABCDEF'))

print(df)
# Output:
#           A         B         C         D         E         F
# 0 -0.895467  0.386902 -0.510805 -1.180632 -0.028182  0.428332
# 1  0.066517  0.302472 -0.634322 -0.362741 -0.672460 -0.359553
# 2 -0.813146 -1.726283  0.177426 -0.401781 -1.630198  0.462782
# 3 -0.907298  0.051945  0.729091  0.128983  1.139401 -1.234826
# 4  0.402342 -0.684810 -0.870797 -0.578850 -0.311553  0.056165
```

1) Использование `del`

```
del df['C']

print(df)
# Output:
#           A         B         D         E         F
# 0 -0.895467  0.386902 -1.180632 -0.028182  0.428332
# 1  0.066517  0.302472 -0.362741 -0.672460 -0.359553
# 2 -0.813146 -1.726283 -0.401781 -1.630198  0.462782
# 3 -0.907298  0.051945  0.128983  1.139401 -1.234826
# 4  0.402342 -0.684810 -0.578850 -0.311553  0.056165
```

2) Использование `drop`

```
df.drop(['B', 'E'], axis='columns', inplace=True)
# or df = df.drop(['B', 'E'], axis=1) without the option inplace=True

print(df)
# Output:
#           A         D         F
# 0 -0.895467 -1.180632  0.428332
# 1  0.066517 -0.362741 -0.359553
# 2 -0.813146 -0.401781  0.462782
# 3 -0.907298  0.128983 -1.234826
# 4  0.402342 -0.578850  0.056165
```

3) Использование `drop` с номерами столбцов

Использовать столбцы целых чисел вместо имен (помните, что индексы столбцов начинаются с нуля):

```
df.drop(df.columns[[0, 2]], axis='columns')

print(df)
# Output:
#           D
# 0 -1.180632
# 1 -0.362741
# 2 -0.401781
# 3  0.128983
# 4 -0.578850
```

Переименовать столбец

```
df = pd.DataFrame({'old_name_1': [1, 2, 3], 'old_name_2': [5, 6, 7]})

print(df)
# Output:
#    old_name_1  old_name_2
# 0           1           5
# 1           2           6
# 2           3           7
```

Чтобы переименовать один или несколько столбцов, передайте старые имена и новые имена в качестве словаря:

```
df.rename(columns={'old_name_1': 'new_name_1', 'old_name_2': 'new_name_2'}, inplace=True)
print(df)
# Output:
#   new_name_1  new_name_2
# 0           1           5
# 1           2           6
# 2           3           7
```

Или функция:

```
df.rename(columns=lambda x: x.replace('old_', '_new'), inplace=True)
print(df)
# Output:
#   new_name_1  new_name_2
# 0           1           5
# 1           2           6
# 2           3           7
```

Вы также можете установить `df.columns` как список новых имен:

```
df.columns = ['new_name_1','new_name_2']
print(df)
# Output:
#   new_name_1  new_name_2
# 0           1           5
```

```
# 1           2           6
# 2           3           7
```

Более подробную информацию можно найти здесь .

Добавление нового столбца

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})

print(df)
# Output:
#    A  B
# 0  1  4
# 1  2  5
# 2  3  6
```

Непосредственно назначить

```
df['C'] = [7, 8, 9]

print(df)
# Output:
#    A  B  C
# 0  1  4  7
# 1  2  5  8
# 2  3  6  9
```

Добавить постоянный столбец

```
df['C'] = 1

print(df)

# Output:
#    A  B  C
# 0  1  4  1
# 1  2  5  1
# 2  3  6  1
```

Столбец как выражение в других столбцах

```
df['C'] = df['A'] + df['B']

# print(df)
# Output:
#    A  B  C
# 0  1  4  5
# 1  2  5  7
# 2  3  6  9

df['C'] = df['A']**df['B']
```

```
print(df)
# Output:
#    A  B    C
# 0  1  4    1
# 1  2  5   32
# 2  3  6  729
```

Операции вычисляются по компонентам, поэтому, если бы у нас были столбцы в виде списков

```
a = [1, 2, 3]
b = [4, 5, 6]
```

столбец в последнем выражении будет получен как

```
c = [x**y for (x,y) in zip(a,b)]

print(c)
# Output:
# [1, 32, 729]
```

Создайте его на лету

```
df_means = df.assign(D=[10, 20, 30]).mean()

print(df_means)
# Output:
# A     2.0
# B     5.0
# C     7.0
# D    20.0  # adds a new column D before taking the mean
# dtype: float64
```

добавить несколько столбцов

```
df = pd.DataFrame({'A': [1, 2, 3], 'B': [4, 5, 6]})
df[['A2','B2']] = np.square(df)

print(df)
# Output:
#    A  B  A2  B2
# 0  1  4   1  16
# 1  2  5   4  25
# 2  3  6   9  36
```

добавить несколько столбцов на лету

```
new_df = df.assign(A3=df.A*df.A2, B3=5*df.B)
```

```
print(new_df)
# Output:
#    A  B  A2  B2  A3  B3
# 0  1  4   1  16   1  20
# 1  2  5   4  25   8  25
# 2  3  6   9  36  27  30
```

Локализовать и заменить данные в столбце

```
import pandas as pd

df = pd.DataFrame({'gender': ["male", "female","female"],
                   'id': [1, 2, 3] })
>>> df
   gender  id
0    male   1
1  female   2
2  female   3
```

Чтобы закодировать самец до 0 и женский до 1:

```
df.loc[df["gender"] == "male","gender"] = 0
df.loc[df["gender"] == "female","gender"] = 1

>>> df
       gender  id
    0       0   1
    1       1   2
    2       1   3
```

Добавление новой строки в DataFrame

Учитывая DataFrame:

```
s1 = pd.Series([1,2,3])
s2 = pd.Series(['a','b','c'])

df = pd.DataFrame([list(s1), list(s2)],  columns =  ["C1", "C2", "C3"])
print df
```

Выход:

```
  C1 C2 C3
0  1  2  3
1  a  b  c
```

Давайте добавим новую строку, `[10,11,12]` :

```
df = pd.DataFrame(np.array([[10,11,12]]), \
        columns=["C1", "C2", "C3"]).append(df, ignore_index=True)
print df
```

Выход:

```
   C1  C2  C3
0  10  11  12
1   1   2   3
2   a   b   c
```

Удалить / удалить строки из DataFrame

давайте сначала сгенерируем DataFrame:

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

print(df)
# Output:
#    a  b
# 0  0  1
# 1  2  3
# 2  4  5
# 3  6  7
# 4  8  9
```

падение строк с индексами: `0` и `4` с использованием метода `drop([...], inplace=True)` :

```
df.drop([0,4], inplace=True)

print(df)
# Output
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

drop rows с индексами: `0` и `4` с использованием метода `df = drop([...])` :

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

df = df.drop([0,4])

print(df)
# Output:
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

используя метод отрицательного выбора:

```
df = pd.DataFrame(np.arange(10).reshape(5,2), columns=list('ab'))

df = df[~df.index.isin([0,4])]

print(df)
# Output:
```

```
#    a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

Изменить порядок столбцов

```
# get a list of columns
cols = list(df)

# move the column to head of list using index, pop and insert
cols.insert(0, cols.pop(cols.index('listing')))

# use ix to reorder
df2 = df.ix[:, cols]
```

Прочитайте Простое управление DataFrames онлайн:
https://riptutorial.com/ru/pandas/topic/6694/простое-управление-dataframes

глава 30: Работа с временными рядами

Examples

Создание временных рядов

Вот как создать простой Time Series.

```
import pandas as pd
import numpy as np

# The number of sample to generate
nb_sample = 100

# Seeding to obtain a reproductible dataset
np.random.seed(0)

se = pd.Series(np.random.randint(0, 100, nb_sample),
                  index = pd.date_range(start = pd.to_datetime('2016-09-24'),
                                        periods = nb_sample, freq='D'))
se.head(2)

# 2016-09-24    44
# 2016-09-25    47

se.tail(2)

# 2016-12-31    85
# 2017-01-01    48
```

Индексирование частичной строки

Очень удобный способ подмножества Time Series - использовать **частичную индексацию строк** . Он позволяет выбирать диапазон дат с четким синтаксисом.

Получение данных

Мы используем набор данных в примере создания временного ряда

Отображение головы и хвоста для просмотра границ

```
se.head(2).append(se.tail(2))

# 2016-09-24    44
# 2016-09-25    47
# 2016-12-31    85
# 2017-01-01    48
```

Подменю

Теперь мы можем подмножество по годам, месяцам, дням очень интуитивно.

По годам

```
se['2017']

# 2017-01-01    48
```

По месяцам

```
se['2017-01']

# 2017-01-01    48
```

Днем

```
se['2017-01-01']

# 48
```

С диапазоном года, месяца, дня в соответствии с вашими потребностями.

```
se['2016-12-31':'2017-01-01']

# 2016-12-31    85
# 2017-01-01    48
```

pandas также предоставляет выделенную функцию `truncate` для этого использования через параметры `after` и `before` но я думаю, что это менее понятно.

```
se.truncate(before='2017')

# 2017-01-01    48

se.truncate(before='2016-12-30', after='2016-12-31')

# 2016-12-30    13
# 2016-12-31    85
```

Прочитайте Работа с временными рядами онлайн: https://riptutorial.com/ru/pandas/topic/7029/работа-с-временными-рядами

глава 31: Работа с категориальными переменными

Examples

Однострочное кодирование с помощью `get_dummies ()`

```
>>> df = pd.DataFrame({'Name':['John Smith', 'Mary Brown'],
                       'Gender':['M', 'F'], 'Smoker':['Y', 'N']})
>>> print(df)
```

```
  Gender        Name Smoker
0      M  John Smith      Y
1      F  Mary Brown      N
```

```
>>> df_with_dummies = pd.get_dummies(df, columns=['Gender', 'Smoker'])
>>> print(df_with_dummies)
```

```
         Name  Gender_F  Gender_M  Smoker_N  Smoker_Y
0  John Smith       0.0       1.0       0.0       1.0
1  Mary Brown       1.0       0.0       1.0       0.0
```

Прочитайте Работа с категориальными переменными онлайн:
https://riptutorial.com/ru/pandas/topic/5999/работа-с-категориальными-переменными

глава 32: Серии

Examples

Примеры создания простых серий

Серия представляет собой одномерную структуру данных. Это немного похоже на суперзарядный массив или словарь.

```
import pandas as pd

s = pd.Series([10, 20, 30])

>>> s
0    10
1    20
2    30
dtype: int64
```

Каждое значение в ряду имеет индекс. По умолчанию индексы являются целыми числами, начиная от 0 до длины серии минус 1. В приведенном выше примере вы можете увидеть индексы, напечатанные слева от значений.

Вы можете указать свои собственные индексы:

```
s2 = pd.Series([1.5, 2.5, 3.5], index=['a', 'b', 'c'], name='my_series')

>>> s2
a    1.5
b    2.5
c    3.5
Name: my_series, dtype: float64

s3 = pd.Series(['a', 'b', 'c'], index=list('ABC'))

>>> s3
A    a
B    b
C    c
dtype: object
```

Серия с датой

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=5, freq='T')
s = pd.Series(np.random.randn(len(rng)), index=rng)
print (s)
```

```
2015-02-24 00:00:00    1.764052
2015-02-24 00:01:00    0.400157
2015-02-24 00:02:00    0.978738
2015-02-24 00:03:00    2.240893
2015-02-24 00:04:00    1.867558
Freq: T, dtype: float64

rng = pd.date_range('2015-02-24', periods=5, freq='T')
s1 = pd.Series(rng)
print (s1)

0   2015-02-24 00:00:00
1   2015-02-24 00:01:00
2   2015-02-24 00:02:00
3   2015-02-24 00:03:00
4   2015-02-24 00:04:00
dtype: datetime64[ns]
```

Несколько советов о серии в Пандах

Предположим, что мы имеем следующую серию:

```
>>> import pandas as pd
>>> s = pd.Series([1, 4, 6, 3, 8, 7, 4, 5])
>>> s
0    1
1    4
2    6
3    3
4    8
5    7
6    4
7    5
dtype: int64
```

Подходы - это несколько простых вещей, которые пригодится, когда вы работаете с серией:

Чтобы получить длину s:

```
>>> len(s)
8
```

Чтобы получить доступ к элементу в s:

```
>>> s[4]
8
```

Чтобы получить доступ к элементу с помощью индекса:

```
>>> s.loc[2]
6
```

Для доступа к подсерии внутри s:

```
>>> s[1:3]
1    4
2    6
dtype: int64
```

Чтобы получить подсерию s с значениями больше 5:

```
>>> s[s > 5]
2    6
4    8
5    7
dtype: int64
```

Чтобы получить минимальное, максимальное, среднее и стандартное отклонение:

```
>>> s.min()
1
>>> s.max()
8
>>> s.mean()
4.75
>>> s.std()
2.2519832529192065
```

Чтобы преобразовать тип Series в float:

```
>>> s.astype(float)
0    1.0
1    4.0
2    6.0
3    3.0
4    8.0
5    7.0
6    4.0
7    5.0
dtype: float64
```

Чтобы получить значения в s как массив numpy:

```
>>> s.values
array([1, 4, 6, 3, 8, 7, 4, 5])
```

Чтобы сделать копию s:

```
>>> d = s.copy()
>>> d
0    1
1    4
2    6
3    3
4    8
```

```
5    7
6    4
7    5
dtype: int64
```

Применение функции к серии

Pandas обеспечивает эффективный способ применения функции к каждому элементу серии и получения новой серии. Предположим, что мы имеем следующую серию:

```
>>> import pandas as pd
>>> s = pd.Series([3, 7, 5, 8, 9, 1, 0, 4])
>>> s
0    3
1    7
2    5
3    8
4    9
5    1
6    0
7    4
dtype: int64
```

и квадратная функция:

```
>>> def square(x):
...     return x*x
```

Мы можем просто применить квадрат к каждому элементу s и получить новую серию:

```
>>> t = s.apply(square)
>>> t
0     9
1    49
2    25
3    64
4    81
5     1
6     0
7    16
dtype: int64
```

В некоторых случаях легче использовать лямбда-выражение:

```
>>> s.apply(lambda x: x ** 2)
0     9
1    49
2    25
3    64
4    81
5     1
6     0
7    16
dtype: int64
```

или мы можем использовать любую встроенную функцию:

```
>>> q = pd.Series(['Bob', 'Jack', 'Rose'])
>>> q.apply(str.lower)
0     bob
1    jack
2    rose
dtype: object
```

Если все элементы серии являются строками, существует более простой способ применения строковых методов:

```
>>> q.str.lower()
0     bob
1    jack
2    rose
dtype: object
>>> q.str.len()
0    3
1    4
2    4
```

Прочитайте Серии онлайн: https://riptutorial.com/ru/pandas/topic/1898/серии

глава 33: Сечения различных осей с помощью MultiIndex

Examples

Выбор поперечных сечений с использованием .xs

```
In [1]:
import pandas as pd
import numpy as np
arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
idx_row = pd.MultiIndex.from_arrays(arrays, names=['Row_First', 'Row_Second'])
idx_col = pd.MultiIndex.from_product([['A','B'], ['i', 'ii']],
names=['Col_First','Col_Second'])
df = pd.DataFrame(np.random.randn(8,4), index=idx_row, columns=idx_col)

Out[1]:
Col_First                   A                   B
Col_Second                  i        ii         i        ii
Row_First Row_Second
bar       one        -0.452982 -1.872641  0.248450 -0.319433
          two        -0.460388 -0.136089 -0.408048  0.998774
baz       one         0.358206 -0.319344 -2.052081 -0.424957
          two        -0.823811 -0.302336  1.158968  0.272881
foo       one        -0.098048 -0.799666  0.969043 -0.595635
          two        -0.358485  0.412011 -0.667167  1.010457
qux       one         1.176911  1.578676  0.350719  0.093351
          two         0.241956  1.082138 -0.516898 -0.196605
```

`.xs` принимает `level` (либо имя указанного уровня, либо целое число), и `axis` : 0 для строк, 1 для столбцов.

`.xs` доступен как для `pandas.Series` и для `pandas.DataFrame` .

Выбор по строкам:

```
In [2]: df.xs('two', level='Row_Second', axis=0)
Out[2]:
Col_First          A                   B
Col_Second         i        ii         i        ii
Row_First
bar        -0.460388 -0.136089 -0.408048  0.998774
baz        -0.823811 -0.302336  1.158968  0.272881
foo        -0.358485  0.412011 -0.667167  1.010457
qux         0.241956  1.082138 -0.516898 -0.196605
```

Выбор по столбцам:

```
In [3]: df.xs('ii', level=1, axis=1)
```

```
Out[3]:
Col_First                   A         B
Row_First Row_Second
bar       one        -1.872641 -0.319433
          two        -0.136089  0.998774
baz       one        -0.319344 -0.424957
          two        -0.302336  0.272881
foo       one        -0.799666 -0.595635
          two         0.412011  1.010457
qux       one         1.578676  0.093351
          two         1.082138 -0.196605
```

`.xs` работает только для выбора, назначение НЕ возможно (получение, а не настройка): ¨

```
In [4]: df.xs('ii', level='Col_Second', axis=1) = 0
  File "<ipython-input-10-92e0785187ba>", line 1
    df.xs('ii', level='Col_Second', axis=1) = 0
                                            ^
SyntaxError: can't assign to function call
```

Использование .loc и slicers

В отличие от метода `.xs` , это позволяет вам присваивать значения. Индексирование с помощью slicers доступно с версии `0.14.0` .

```
In [1]:
import pandas as pd
import numpy as np
arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],
          ['one', 'two', 'one', 'two', 'one', 'two', 'one', 'two']]
idx_row = pd.MultiIndex.from_arrays(arrays, names=['Row_First', 'Row_Second'])
idx_col = pd.MultiIndex.from_product([['A','B'], ['i', 'ii']],
names=['Col_First','Col_Second'])
df = pd.DataFrame(np.random.randn(8,4), index=idx_row, columns=idx_col)

Out[1]:
Col_First                   A                   B
Col_Second                  i        ii         i        ii
Row_First Row_Second
bar       one        -0.452982 -1.872641  0.248450 -0.319433
          two        -0.460388 -0.136089 -0.408048  0.998774
baz       one         0.358206 -0.319344 -2.052081 -0.424957
          two        -0.823811 -0.302336  1.158968  0.272881
foo       one        -0.098048 -0.799666  0.969043 -0.595635
          two        -0.358485  0.412011 -0.667167  1.010457
qux       one         1.176911  1.578676  0.350719  0.093351
          two         0.241956  1.082138 -0.516898 -0.196605
```

Выбор по строкам :

```
In [2]: df.loc[(slice(None),'two'),:]
Out[2]:
Col_First                   A                   B
Col_Second                  i        ii         i        ii
```

```
Row_First Row_Second
bar       two        -0.460388 -0.136089 -0.408048  0.998774
baz       two        -0.823811 -0.302336  1.158968  0.272881
foo       two        -0.358485  0.412011 -0.667167  1.010457
qux       two         0.241956  1.082138 -0.516898 -0.196605
```

Выбор по столбцам:

```
In [3]: df.loc[:,(slice(None),'ii')]
Out[3]:
Col_First                   A         B
Col_Second                 ii        ii
Row_First Row_Second
bar       one        -1.872641 -0.319433
          two        -0.136089  0.998774
baz       one        -0.319344 -0.424957
          two        -0.302336  0.272881
foo       one        -0.799666 -0.595635
          two         0.412011  1.010457
qux       one         1.578676  0.093351
          two         1.082138 -0.196605
```

Выбор по обоим осям ::

```
In [4]: df.loc[(slice(None),'two'),(slice(None),'ii')]
Out[4]:
Col_First                   A         B
Col_Second                 ii        ii
Row_First Row_Second
bar       two        -0.136089  0.998774
baz       two        -0.302336  0.272881
foo       two         0.412011  1.010457
qux       two         1.082138 -0.196605
```

Задание `.xs` (в отличие от `.xs`):

```
In [5]: df.loc[(slice(None),'two'),(slice(None),'ii')]=0
        df
Out[5]:
Col_First                   A                   B
Col_Second                  i        ii         i        ii
Row_First Row_Second
bar       one        -0.452982 -1.872641  0.248450 -0.319433
          two        -0.460388  0.000000 -0.408048  0.000000
baz       one         0.358206 -0.319344 -2.052081 -0.424957
          two        -0.823811  0.000000  1.158968  0.000000
foo       one        -0.098048 -0.799666  0.969043 -0.595635
          two        -0.358485  0.000000 -0.667167  0.000000
qux       one         1.176911  1.578676  0.350719  0.093351
          two         0.241956  0.000000 -0.516898  0.000000
```

Прочитайте Сечения различных осей с помощью MultiIndex онлайн:
https://riptutorial.com/ru/pandas/topic/8099/сечения-различных-осей-с-помощью-multiindex

глава 34: Создание DataFrames

Вступление

DataFrame - это структура данных, предоставляемая библиотекой pandas, кроме *Series* & *Panel* . Это двумерная структура и может быть сравнена с таблицей строк и столбцов.

Каждая строка может быть идентифицирована целым индексом (0..N) или меткой, явно заданной при создании объекта DataFrame. Каждый столбец может иметь различный тип и идентифицируется меткой.

В этом разделе рассматриваются различные способы создания / создания объекта DataFrame. Ex. из массивов Numpy, из списка кортежей, из словаря.

Examples

Создать образец DataFrame

```
import pandas as pd
```

Создайте DataFrame из словаря, содержащего два столбца: `numbers` и `colors` . Каждый ключ представляет собой имя столбца, а значение представляет собой серию данных, содержимое столбца:

```
df = pd.DataFrame({'numbers': [1, 2, 3], 'colors': ['red', 'white', 'blue']})
```

Показывать содержимое dataframe:

```
print(df)
# Output:
#   colors  numbers
# 0    red        1
# 1  white        2
# 2   blue        3
```

Pandas упорядочивает столбцы в алфавитном порядке, поскольку `dict` не упорядочен. Чтобы указать порядок, используйте параметр `columns` .

```
df = pd.DataFrame({'numbers': [1, 2, 3], 'colors': ['red', 'white', 'blue']},
                  columns=['numbers', 'colors'])

print(df)
# Output:
#    numbers colors
# 0        1    red
# 1        2  white
```

```
# 2        3   blue
```

Создайте образец DataFrame с помощью Numpy

Создайте `DataFrame` случайных чисел:

```
import numpy as np
import pandas as pd

# Set the seed for a reproducible sample
np.random.seed(0)

df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

print(df)
# Output:
#           A         B         C
# 0  1.764052  0.400157  0.978738
# 1  2.240893  1.867558 -0.977278
# 2  0.950088 -0.151357 -0.103219
# 3  0.410599  0.144044  1.454274
# 4  0.761038  0.121675  0.443863
```

Создайте `DataFrame` с целыми числами:

```
df = pd.DataFrame(np.arange(15).reshape(5,3),columns=list('ABC'))

print(df)
# Output:
#     A   B   C
# 0   0   1   2
# 1   3   4   5
# 2   6   7   8
# 3   9  10  11
# 4  12  13  14
```

Создайте `DataFrame` и включите nans (`NaT, NaN, 'nan', None`) по столбцам и строкам:

```
df = pd.DataFrame(np.arange(48).reshape(8,6),columns=list('ABCDEF'))

print(df)
# Output:
#     A   B   C   D   E   F
# 0   0   1   2   3   4   5
# 1   6   7   8   9  10  11
# 2  12  13  14  15  16  17
# 3  18  19  20  21  22  23
# 4  24  25  26  27  28  29
# 5  30  31  32  33  34  35
# 6  36  37  38  39  40  41
# 7  42  43  44  45  46  47

df.ix[::2,0] = np.nan # in column 0, set elements with indices 0,2,4, ... to NaN
df.ix[::4,1] = pd.NaT # in column 1, set elements with indices 0,4, ... to np.NaT
df.ix[:3,2] = 'nan'   # in column 2, set elements with index from 0 to 3 to 'nan'
df.ix[:,5] = None     # in column 5, set all elements to None
```

```
df.ix[5,:] = None     # in row 5, set all elements to None
df.ix[7,:] = np.nan   # in row 7, set all elements to NaN

print(df)
# Output:
#     A     B     C   D   E     F
# 0 NaN   NaT   nan   3   4  None
# 1   6     7   nan   9  10  None
# 2 NaN    13   nan  15  16  None
# 3  18    19   nan  21  22  None
# 4 NaN   NaT    26  27  28  None
# 5 NaN  None  None NaN NaN  None
# 6 NaN    37    38  39  40  None
# 7 NaN   NaN   NaN NaN NaN   NaN
```

Создайте образец DataFrame из нескольких коллекций, используя словарь

```
import pandas as pd
import numpy as np

np.random.seed(123)
x = np.random.standard_normal(4)
y = range(4)
df = pd.DataFrame({'X':x, 'Y':y})
>>> df
          X  Y
0 -1.085631  0
1  0.997345  1
2  0.282978  2
3 -1.506295  3
```

Создайте DataFrame из списка кортежей

Вы можете создать DataFrame из списка простых кортежей и даже выбрать определенные элементы кортежей, которые вы хотите использовать. Здесь мы создадим DataFrame, используя все данные в каждом кортеже, за исключением последнего элемента.

```
import pandas as pd

data = [
('p1', 't1', 1, 2),
('p1', 't2', 3, 4),
('p2', 't1', 5, 6),
('p2', 't2', 7, 8),
('p2', 't3', 2, 8)
]

df = pd.DataFrame(data)

print(df)
#     0   1  2  3
# 0  p1  t1  1  2
# 1  p1  t2  3  4
# 2  p2  t1  5  6
```

```
# 3  p2  t2  7  8
# 4  p2  t3  2  8
```

Создайте DataFrame из словаря списков

Создайте DataFrame из нескольких списков, передав dict, чьи значения перечислены. Клавиши словаря используются как метки столбцов. Списки также могут быть ndarrays. Списки / ndarrays должны иметь одинаковую длину.

```
import pandas as pd

# Create DF from dict of lists/ndarrays
df = pd.DataFrame({'A' : [1, 2, 3, 4],
                   'B' : [4, 3, 2, 1]})
df
# Output:
#       A  B
#    0  1  4
#    1  2  3
#    2  3  2
#    3  4  1
```

Если массивы не имеют одинаковой длины, возникает ошибка

```
df = pd.DataFrame({'A' : [1, 2, 3, 4], 'B' : [5, 5, 5]}) # a ValueError is raised
```

Использование ndarrays

```
import pandas as pd
import numpy as np

np.random.seed(123)
x = np.random.standard_normal(4)
y = range(4)
df = pd.DataFrame({'X':x, 'Y':y})
df
# Output:           X  Y
#         0 -1.085631  0
#         1  0.997345  1
#         2  0.282978  2
#         3 -1.506295  3
```

См. Дополнительную информацию по адресу: http://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists

Создайте образец DataFrame с датой

```
import pandas as pd
import numpy as np

np.random.seed(0)
# create an array of 5 dates starting at '2015-02-24', one per minute
```

```
rng = pd.date_range('2015-02-24', periods=5, freq='T')
df = pd.DataFrame({ 'Date': rng, 'Val': np.random.randn(len(rng)) })

print (df)
# Output:
#                  Date       Val
# 0 2015-02-24 00:00:00  1.764052
# 1 2015-02-24 00:01:00  0.400157
# 2 2015-02-24 00:02:00  0.978738
# 3 2015-02-24 00:03:00  2.240893
# 4 2015-02-24 00:04:00  1.867558

# create an array of 5 dates starting at '2015-02-24', one per day
rng = pd.date_range('2015-02-24', periods=5, freq='D')
df = pd.DataFrame({ 'Date': rng, 'Val' : np.random.randn(len(rng))})

print (df)
# Output:
#         Date       Val
# 0 2015-02-24 -0.977278
# 1 2015-02-25  0.950088
# 2 2015-02-26 -0.151357
# 3 2015-02-27 -0.103219
# 4 2015-02-28  0.410599

# create an array of 5 dates starting at '2015-02-24', one every 3 years
rng = pd.date_range('2015-02-24', periods=5, freq='3A')
df = pd.DataFrame({ 'Date': rng, 'Val' : np.random.randn(len(rng))})

print (df)
# Output:
#         Date       Val
# 0 2015-12-31  0.144044
# 1 2018-12-31  1.454274
# 2 2021-12-31  0.761038
# 3 2024-12-31  0.121675
# 4 2027-12-31  0.443863
```

DataFrame с `DatetimeIndex` :

```
import pandas as pd
import numpy as np

np.random.seed(0)
rng = pd.date_range('2015-02-24', periods=5, freq='T')
df = pd.DataFrame({ 'Val' : np.random.randn(len(rng)) }, index=rng)

print (df)
# Output:
#                            Val
# 2015-02-24 00:00:00  1.764052
# 2015-02-24 00:01:00  0.400157
# 2015-02-24 00:02:00  0.978738
# 2015-02-24 00:03:00  2.240893
# 2015-02-24 00:04:00  1.867558
```

`Offset-aliases` для параметра `freq` в `date_range` :

```
Alias     Description
B         business day frequency
C         custom business day frequency (experimental)
D         calendar day frequency
W         weekly frequency
M         month end frequency
BM        business month end frequency
CBM       custom business month end frequency
MS        month start frequency
BMS       business month start frequency
CBMS      custom business month start frequency
Q         quarter end frequency
BQ        business quarter endfrequency
QS        quarter start frequency
BQS       business quarter start frequency
A         year end frequency
BA        business year end frequency
AS        year start frequency
BAS       business year start frequency
BH        business hour frequency
H         hourly frequency
T, min    minutely frequency
S         secondly frequency
L, ms     milliseconds
U, us     microseconds
N         nanoseconds
```

Создайте образец DataFrame с помощью MultiIndex

```
import pandas as pd
import numpy as np
```

Использование `from_tuples` :

```
np.random.seed(0)
tuples = list(zip(*[['bar', 'bar', 'baz', 'baz',
                     'foo', 'foo', 'qux', 'qux'],
                      ['one', 'two', 'one', 'two',
                       'one', 'two', 'one', 'two']]))

idx = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
```

Использование `from_product` :

```
idx = pd.MultiIndex.from_product([['bar', 'baz', 'foo', 'qux'],['one','two']])
```

Затем используйте этот MultiIndex:

```
df = pd.DataFrame(np.random.randn(8, 2), index=idx, columns=['A', 'B'])
print (df)
                     A         B
first second
bar   one     1.764052  0.400157
      two     0.978738  2.240893
baz   one     1.867558 -0.977278
```

```
      two     0.950088 -0.151357
foo   one    -0.103219  0.410599
      two     0.144044  1.454274
qux   one     0.761038  0.121675
      two     0.443863  0.333674
```

Сохранение и загрузка DataFrame в формате pickle (.plk)

```
import pandas as pd

# Save dataframe to pickled pandas object
df.to_pickle(file_name) # where to save it usually as a .plk

# Load dataframe from pickled pandas object
df= pd.read_pickle(file_name)
```

Создайте DataFrame из списка словарей

DataFrame может быть создан из списка словарей. Ключи используются как имена столбцов.

```
import pandas as pd
L = [{'Name': 'John', 'Last Name': 'Smith'},
         {'Name': 'Mary', 'Last Name': 'Wood'}]
pd.DataFrame(L)
# Output:  Last Name  Name
# 0     Smith  John
# 1      Wood  Mary
```

Недостающие значения заполняются `NaN` s

```
L = [{'Name': 'John', 'Last Name': 'Smith', 'Age': 37},
     {'Name': 'Mary', 'Last Name': 'Wood'}]
pd.DataFrame(L)
# Output:     Age Last Name  Name
#          0   37     Smith  John
#          1  NaN      Wood  Mary
```

Прочитайте Создание DataFrames онлайн: https://riptutorial.com/ru/pandas/topic/1595/создание-dataframes

глава 35: Создание Pandas Play Nice с родными типами Python

Examples

Перенос данных из панд в составные структуры данных Python и Numpy

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],
                           'D': [True, False, True]})

In [2]: df
Out[2]:
   A    B  C      D
0  1  1.0  a   True
1  2  2.0  b  False
2  3  3.0  c   True
```

Получение списка python из серии:

```
In [3]: df['A'].tolist()
Out[3]: [1, 2, 3]
```

В DataFrames нет метода `tolist()` . Попытка его приводит к атрибуту AttributeError:

```
In [4]: df.tolist()
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-4-fc6763af1ff7> in <module>()
----> 1 df.tolist()

//anaconda/lib/python2.7/site-packages/pandas/core/generic.pyc in __getattr__(self, name)
   2742             if name in self._info_axis:
   2743                 return self[name]
-> 2744             return object.__getattribute__(self, name)
   2745
   2746     def __setattr__(self, name, value):

AttributeError: 'DataFrame' object has no attribute 'tolist'
```

Получение массива numpy из серии:

```
In [5]: df['B'].values
Out[5]: array([ 1.,  2.,  3.])
```

Вы также можете получить массив столбцов в виде отдельных массивов numpy из всего фрейма данных:

```
In [6]: df.values
```

```
Out[6]:
array([[1, 1.0, 'a', True],
       [2, 2.0, 'b', False],
       [3, 3.0, 'c', True]], dtype=object)
```

Получение словаря из серии (использует индекс как ключи):

```
In [7]: df['C'].to_dict()
Out[7]: {0: 'a', 1: 'b', 2: 'c'}
```

Вы также можете вернуть весь DataFrame в качестве словаря:

```
In [8]: df.to_dict()
Out[8]:
{'A': {0: 1, 1: 2, 2: 3},
 'B': {0: 1.0, 1: 2.0, 2: 3.0},
 'C': {0: 'a', 1: 'b', 2: 'c'},
 'D': {0: True, 1: False, 2: True}}
```

Метод `to_dict` имеет несколько разных параметров для настройки формата форматирования словарей. Чтобы получить список dicts для каждой строки:

```
In [9]: df.to_dict('records')
Out[9]:
[{'A': 1, 'B': 1.0, 'C': 'a', 'D': True},
 {'A': 2, 'B': 2.0, 'C': 'b', 'D': False},
 {'A': 3, 'B': 3.0, 'C': 'c', 'D': True}]
```

См. Документацию для полного списка опций, доступных для создания словарей.

Прочитайте Создание Pandas Play Nice с родными типами Python онлайн: https://riptutorial.com/ru/pandas/topic/8008/создание-pandas-play-nice-с-родными-типами-python

глава 36: Сохраните файл данных pandas в файл csv

параметры

параметр	Описание
path_or_buf	string или file handle, по умолчанию None Путь к файлу или объект, если None предоставлен, результат возвращается как строка.
сентябрь	character, default ',' Разделитель полей для выходного файла.
na_rep	string, default '' Отсутствие представления данных
float_format	string, default None Строка формата для чисел с плавающей запятой
столбцы	последовательность, необязательные столбцы для записи
заголовок	boolean или список строк, по умолчанию True Вывести имена столбцов. Если указан список строк, предполагается, что это псевдонимы для имен столбцов
индекс	boolean, по умолчанию True Имена строк строк (индекс)
index_label	строка или последовательность или False, по умолчанию Нет. Значок столбца для столбца индекса, если требуется. Если задано None, а заголовок и индекс - True, то используются имена индексов. Последовательность должна быть задана, если DataFrame использует MultiIndex. Если False не печатает поля для имен индексов. Используйте index_label = False для упрощения импорта в R
nanRep	Ничего не рекомендуется, используйте na_rep
Режим	str Python, по умолчанию 'w'
кодирование	string, optional Строка, представляющая кодировку, которая будет использоваться в выходном файле, по умолчанию равна «ascii» на Python 2 и «utf-8» на Python 3.
компрессия	string, необязательно строка, представляющая сжатие для использования в выходном файле, допустимыми значениями являются «gzip», «bz2», «xz», используется только тогда, когда первый аргумент является именем файла

параметр	Описание
line_terminator	string, default 'n' Символ новой строки или символьная последовательность для использования в выходном файле
квотирование	необязательная константа из модуля csv по умолчанию - csv.QUOTE_MINIMAL
QuoteChar	string (длина 1), символ по умолчанию " ', используемый для цитирования полей
двойная кавычка	boolean, по умолчанию True Control цитирование катчара внутри поля
escapechar	string (длина 1), по умолчанию None используется символ, чтобы избежать sep и quotechar, когда это необходимо
chunksize	int или Нет строк для записи одновременно
tupleize_cols	boolean, по умолчанию False записывать столбцы multi_index в виде списка кортежей (если True) или нового (расширенный формат), если False)
Формат даты	string, default None Строка формата для объектов datetime
десятичный	string, default '.' Символ распознается как десятичный разделитель. Например, используйте «,» для европейских данных

Examples

Создайте случайный DataFrame и напишите в .csv

Создайте простой DataFrame.

```
import numpy as np
import pandas as pd

# Set the seed so that the numbers can be reproduced.
np.random.seed(0)

df = pd.DataFrame(np.random.randn(5, 3), columns=list('ABC'))

# Another way to set column names is
"columns=['column_1_name','column_2_name','column_3_name']"

df

      A         B         C
0  1.764052  0.400157  0.978738
1  2.240893  1.867558 -0.977278
```

```
2  0.950088 -0.151357 -0.103219
3  0.410599  0.144044  1.454274
4  0.761038  0.121675  0.443863
```

Теперь напишите CSV-файл:

```
df.to_csv('example.csv', index=False)
```

Содержание example.csv:

```
A,B,C
1.76405234597,0.400157208367,0.978737984106
2.2408931992,1.86755799015,-0.977277879876
0.950088417526,-0.151357208298,-0.103218851794
0.410598501938,0.144043571161,1.45427350696
0.761037725147,0.121675016493,0.443863232745
```

Обратите внимание, что мы указываем `index=False` чтобы автоматически сгенерированные индексы (строка #s 0,1,2,3,4) не были включены в файл CSV. Включите его, если вам нужен индексный столбец, например:

```
df.to_csv('example.csv', index=True)  # Or just leave off the index param; default is True
```

Содержание example.csv:

```
,A,B,C
0,1.76405234597,0.400157208367,0.978737984106
1,2.2408931992,1.86755799015,-0.977277879876
2,0.950088417526,-0.151357208298,-0.103218851794
3,0.410598501938,0.144043571161,1.45427350696
4,0.761037725147,0.121675016493,0.443863232745
```

Также обратите внимание, что вы можете удалить заголовок, если он не нужен с `header=False` . Это самый простой вывод:

```
df.to_csv('example.csv', index=False, header=False)
```

Содержание example.csv:

```
1.76405234597,0.400157208367,0.978737984106
2.2408931992,1.86755799015,-0.977277879876
0.950088417526,-0.151357208298,-0.103218851794
0.410598501938,0.144043571161,1.45427350696
0.761037725147,0.121675016493,0.443863232745
```

Разделитель может быть установлен параметром `sep=` , хотя стандартный разделитель для файлов csv равен `','` .

```
df.to_csv('example.csv', index=False, header=False, sep='\t')
```

```
1.76405234597    0.400157208367    0.978737984106
2.2408931992    1.86755799015    -0.977277879876
0.950088417526    -0.151357208298    -0.103218851794
0.410598501938    0.144043571161    1.45427350696
0.761037725147    0.121675016493    0.443863232745
```

Сохранить Pandas DataFrame из списка в dicts в csv без индекса и с кодировкой данных

```
import pandas as pd
data = [
    {'name': 'Daniel', 'country': 'Uganda'},
    {'name': 'Yao', 'country': 'China'},
    {'name': 'James', 'country': 'Colombia'},
]
df = pd.DataFrame(data)
filename = 'people.csv'
df.to_csv(filename, index=False, encoding='utf-8')
```

Прочитайте Сохраните файл данных pandas в файл csv онлайн:
https://riptutorial.com/ru/pandas/topic/1558/сохраните-файл-данных-pandas-в-файл-csv

глава 37: Строчная манипуляция

Examples

Регулярные выражения

```
# Extract strings with a specific regex
df= df['col_name'].str.extract[r'[Aa-Zz]']

# Replace strings within a regex
df['col_name'].str.replace('Replace this', 'With this')
```

Сведения о том, как сопоставлять строки с использованием регулярных выражений, см. В разделе Начало работы с регулярными выражениями .

Строки для нарезки

Строки в серии можно нарезать с использованием `.str.slice()` или, более удобно, с помощью скобок (`.str[]`).

```
In [1]: ser = pd.Series(['Lorem ipsum', 'dolor sit amet', 'consectetur adipiscing elit'])
In [2]: ser
Out[2]:
0                    Lorem ipsum
1                 dolor sit amet
2    consectetur adipiscing elit
dtype: object
```

Получить первый символ каждой строки:

```
In [3]: ser.str[0]
Out[3]:
0    L
1    d
2    c
dtype: object
```

Получите первые три символа каждой строки:

```
In [4]: ser.str[:3]
Out[4]:
0    Lor
1    dol
2    con
dtype: object
```

Получить последний символ каждой строки:

```
In [5]: ser.str[-1]
Out[5]:
0    m
1    t
2    t
dtype: object
```

Получите последние три символа каждой строки:

```
In [6]: ser.str[-3:]
Out[6]:
0    sum
1    met
2    lit
dtype: object
```

Получить каждый другой символ из первых 10 символов:

```
In [7]: ser.str[:10:2]
Out[7]:
0    Lrmis
1    dlrst
2    cnett
dtype: object
```

Pandas ведет себя аналогично Python при обработке срезов и индексов. Например, если индекс находится за пределами диапазона, Python вызывает ошибку:

```
In [8]:'Lorem ipsum'[12]
# IndexError: string index out of range
```

Однако, если срез находится за пределами диапазона, возвращается пустая строка:

```
In [9]: 'Lorem ipsum'[12:15]
Out[9]: ''
```

Pandas возвращает NaN, когда индекс выходит за пределы диапазона:

```
In [10]: ser.str[12]
Out[10]:
0    NaN
1      e
2      a
dtype: object
```

И возвращает пустую строку, если срез выходит за пределы диапазона:

```
In [11]: ser.str[12:15]
Out[11]:
0
1     et
2    adi
```

```
dtype: object
```

Проверка содержимого строки

`str.contains()` может использоваться, чтобы проверить, существует ли шаблон в каждой строке серии. `str.startswith()` и `str.endswith()` также могут использоваться как более специализированные версии.

```
In [1]: animals = pd.Series(['cat', 'dog', 'bear', 'cow', 'bird', 'owl', 'rabbit', 'snake'])
```

Проверьте, содержат ли строки букву «a»:

```
In [2]: animals.str.contains('a')
Out[2]:
0      True
1     False
2      True
3     False
4     False
5     False
6      True
7      True
8      True
dtype: bool
```

Это можно использовать в качестве логического индекса для возвращения только животных, содержащих букву «a»:

```
In [3]: animals[animals.str.contains('a')]
Out[3]:
0       cat
2      bear
6    rabbit
7     snake
dtype: object
```

`str.startswith` и `str.endswith` работают аналогично, но они также принимают кортежи как входы.

```
In [4]: animals[animals.str.startswith(('b', 'c'))]
# Returns animals starting with 'b' or 'c'
Out[4]:
0     cat
2    bear
3     cow
4    bird
dtype: object
```

Капитализация строк

```
In [1]: ser = pd.Series(['lORem ipSuM', 'Dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Преобразовать все в верхний регистр:

```
In [2]: ser.str.upper()
Out[2]:
0                    LOREM IPSUM
1                 DOLOR SIT AMET
2    CONSECTETUR ADIPISCING ELIT
dtype: object
```

Все строчные:

```
In [3]: ser.str.lower()
Out[3]:
0                    lorem ipsum
1                 dolor sit amet
2    consectetur adipiscing elit
dtype: object
```

Заглавные буквы первого и нижнего регистра остальных:

```
In [4]: ser.str.capitalize()
Out[4]:
0                    Lorem ipsum
1                 Dolor sit amet
2    Consectetur adipiscing elit
dtype: object
```

Преобразуйте каждую строку в заголовок (запишите первый символ каждого слова в каждой строке, введите нижний регистр остальных):

```
In [5]: ser.str.title()
Out[5]:
0                    Lorem Ipsum
1                 Dolor Sit Amet
2    Consectetur Adipiscing Elit
dtype: object
```

Сменные чехлы (преобразование в нижнем регистре в верхний регистр и наоборот):

```
In [6]: ser.str.swapcase()
Out[6]:
0                    LorEM IPsUm
1                 dOLOR SIT AMET
2    cONSECTETUR aDIPISCING eLIT
dtype: object
```

Помимо этих методов, которые изменяют капитализацию, для проверки капитализации строк можно использовать несколько методов.

```
In [7]: ser = pd.Series(['LOREM IPSUM', 'dolor sit amet', 'Consectetur Adipiscing Elit'])
```

Проверьте, все ли в нижнем регистре:

```
In [8]: ser.str.islower()
Out[8]:
0    False
1     True
2    False
dtype: bool
```

Все ли в верхнем регистре:

```
In [9]: ser.str.isupper()
Out[9]:
0     True
1    False
2    False
dtype: bool
```

Это строка с заголовком:

```
In [10]: ser.str.istitle()
Out[10]:
0    False
1    False
2     True
dtype: bool
```

Прочитайте Строчная манипуляция онлайн: https://riptutorial.com/ru/pandas/topic/2372/строчная-манипуляция

глава 38: Типы данных

замечания

dtypes не являются родными для панд. Они являются результатом пандса, близкого архитектурному соединению, к numpy.

dtype столбца никоим образом не должен коррелировать с типом python объекта, содержащегося в столбце.

Здесь у нас есть `pd.Series` с поплавками. Dtype будет `float` .

Затем мы используем `astype` для «отливки» объекта.

```
pd.Series([1.,2.,3.,4.,5.]).astype(object)
0    1
1    2
2    3
3    4
4    5
dtype: object
```

Dtype теперь объект, но объекты в списке все еще плавают. Логически, если вы знаете, что в python все является объектом и может быть повышенным до объекта.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(object)[0])
float
```

Здесь мы пытаемся «отличить» поплавки к строкам.

```
pd.Series([1.,2.,3.,4.,5.]).astype(str)
0    1.0
1    2.0
2    3.0
3    4.0
4    5.0
dtype: object
```

Dtype теперь объект, но тип записей в списке - строка. Это потому, что `numpy` не имеет отношения к строкам и, таким образом, действует так, как если бы они были просто объектами и не вызывали беспокойства.

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(str)[0])
str
```

Не доверяйте dtypes, они являются артефактом архитектурного недостатка в пандах. Укажите их так, как вы должны, но не полагайтесь на то, что dtype задано в столбце.

Examples

Проверка типов столбцов

Типы столбцов можно проверить с помощью `.dtypes` atrribute из DataFrames.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': [True, False, True]})

In [2]: df
Out[2]:
   A    B      C
0  1  1.0   True
1  2  2.0  False
2  3  3.0   True

In [3]: df.dtypes
Out[3]:
A      int64
B    float64
C       bool
dtype: object
```

Для одной серии вы можете использовать атрибут `.dtype` .

```
In [4]: df['A'].dtype
Out[4]: dtype('int64')
```

Изменение типов

`astype()` изменяет dtype серии и возвращает новую серию.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0],
                           'C': ['1.1.2010', '2.1.2011', '3.1.2011'],
                           'D': ['1 days', '2 days', '3 days'],
                           'E': ['1', '2', '3']})
In [2]: df
Out[2]:
   A    B          C       D  E
0  1  1.0   1.1.2010  1 days  1
1  2  2.0   2.1.2011  2 days  2
2  3  3.0   3.1.2011  3 days  3

In [3]: df.dtypes
Out[3]:
A      int64
B    float64
C     object
D     object
E     object
dtype: object
```

Измените тип столбца A на float и введите столбец B в целое число:

```
In [4]: df['A'].astype('float')
Out[4]:
0    1.0
1    2.0
2    3.0
Name: A, dtype: float64

In [5]: df['B'].astype('int')
Out[5]:
0    1
1    2
2    3
Name: B, dtype: int32
```

`astype()` предназначен для определенного преобразования типов (т. е. вы можете указать `.astype(float64')` , `.astype(float32)` или `.astype(float16)`). Для общего преобразования вы можете использовать `pd.to_numeric` , `pd.to_datetime` и `pd.to_timedelta` .

Изменение типа на числовое

`pd.to_numeric` изменяет значения на числовой тип.

```
In [6]: pd.to_numeric(df['E'])
Out[6]:
0    1
1    2
2    3
Name: E, dtype: int64
```

По умолчанию `pd.to_numeric` вызывает ошибку, если вход не может быть преобразован в число. Вы можете изменить это поведение, используя параметр `errors` .

```
# Ignore the error, return the original input if it cannot be converted
In [7]: pd.to_numeric(pd.Series(['1', '2', 'a']), errors='ignore')
Out[7]:
0    1
1    2
2    a
dtype: object

# Return NaN when the input cannot be converted to a number
In [8]: pd.to_numeric(pd.Series(['1', '2', 'a']), errors='coerce')
Out[8]:
0    1.0
1    2.0
2    NaN
dtype: float64
```

Если нужно проверить, что все строки с вводом не могут быть преобразованы в числовые, используйте `boolean indexing` с помощью `isnull` :

```
In [9]: df = pd.DataFrame({'A': [1, 'x', 'z'],
                           'B': [1.0, 2.0, 3.0],
```

```
                           'C': [True, False, True]})

In [10]: pd.to_numeric(df.A, errors='coerce').isnull()
Out[10]:
0    False
1     True
2     True
Name: A, dtype: bool

In [11]: df[pd.to_numeric(df.A, errors='coerce').isnull()]
Out[11]:
   A    B      C
1  x  2.0  False
2  z  3.0   True
```

Изменение типа datetime

```
In [12]: pd.to_datetime(df['C'])
Out[12]:
0   2010-01-01
1   2011-02-01
2   2011-03-01
Name: C, dtype: datetime64[ns]
```

Обратите внимание, что 2.1.2011 конвертируется в 1 февраля 2011 года. Если вы хотите 2 января 2011 года вместо этого, вам нужно использовать параметр `dayfirst` .

```
In [13]: pd.to_datetime('2.1.2011', dayfirst=True)
Out[13]: Timestamp('2011-01-02 00:00:00')
```

Изменение типа timedelta

```
In [14]: pd.to_timedelta(df['D'])
Out[14]:
0   1 days
1   2 days
2   3 days
Name: D, dtype: timedelta64[ns]
```

Выбор столбцов на основе dtype

`select_dtypes` метод может использоваться для выбора столбцов на основе dtype.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],
                           'D': [True, False, True]})

In [2]: df
Out[2]:
   A    B  C      D
0  1  1.0  a   True
1  2  2.0  b  False
2  3  3.0  c   True
```

С параметрами `include` и `exclude` вы можете указать, какие типы вы хотите:

```
# Select numbers
In [3]: df.select_dtypes(include=['number'])  # You need to use a list
Out[3]:
   A    B
0  1  1.0
1  2  2.0
2  3  3.0

# Select numbers and booleans
In [4]: df.select_dtypes(include=['number', 'bool'])
Out[4]:
   A    B      D
0  1  1.0   True
1  2  2.0  False
2  3  3.0   True

# Select numbers and booleans but exclude int64
In [5]: df.select_dtypes(include=['number', 'bool'], exclude=['int64'])
Out[5]:
     B      D
0  1.0   True
1  2.0  False
2  3.0   True
```

Подведение итогов

Метод `get_dtype_counts` может использоваться для просмотра разбивки dtypes.

```
In [1]: df = pd.DataFrame({'A': [1, 2, 3], 'B': [1.0, 2.0, 3.0], 'C': ['a', 'b', 'c'],
                           'D': [True, False, True]})

In [2]: df.get_dtype_counts()
Out[2]:
bool       1
float64    1
int64      1
object     1
dtype: int64
```

Прочитайте Типы данных онлайн: https://riptutorial.com/ru/pandas/topic/2959/типы-данных

глава 39: Чтение MySQL в DataFrame

Examples

Использование sqlalchemy и PyMySQL

```
from sqlalchemy import create_engine

cnx = create_engine('mysql+pymysql://username:password@server:3306/database').connect()
sql = 'select * from mytable'
df = pd.read_sql(sql, cnx)
```

Чтобы прочитать mysql для dataframe, в случае большого объема данных

Для получения больших данных мы можем использовать генераторы в пандах и загружать данные в куски.

```
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.engine.url import URL


# sqlalchemy engine
engine = create_engine(URL(
    drivername="mysql"
    username="user",
    password="password"
    host="host"
    database="database"
))

conn = engine.connect()

generator_df = pd.read_sql(sql=query,  # mysql query
                           con=conn,
                           chunksize=chunksize)  # size you want to fetch each time

for dataframe in generator_df:
    for row in dataframe:
        pass  # whatever you want to do
```

Прочитайте Чтение MySQL в DataFrame онлайн: https://riptutorial.com/ru/pandas/topic/8809/чтение-mysql-в-dataframe

глава 40: Чтение SQL Server в DataFrame

Examples

Использование pyodbc

```
import pandas.io.sql
import pyodbc
import pandas as pd
```

Укажите параметры

```
# Parameters
server = 'server_name'
db = 'database_name'
UID = 'user_id'
```

Создать соединение

```
# Create the connection
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=' + server + ';DATABASE=' + db + '; UID = '
+ UID + '; PWD = ' + UID + 'Trusted_Connection=yes')
```

Запрос в базу данных pandas

```
# Query into dataframe
df= pandas.io.sql.read_sql('sql_query_string', conn)
```

Использование pyodbc с контуром соединения

```
import os, time
import pyodbc
import pandas.io.sql as pdsql

def todf(dsn='yourdsn', uid=None, pwd=None, query=None, params=None):
    ''' if `query` is not an actual query but rather a path to a text file
        containing a query, read it in instead '''
    if query.endswith('.sql') and os.path.exists(query):
        with open(query,'r') as fin:
            query = fin.read()

    connstr = "DSN={};UID={};PWD={}".format(dsn,uid,pwd)
    connected = False
    while not connected:
        try:
            with pyodbc.connect(connstr,autocommit=True) as con:
                cur = con.cursor()
                if params is not None: df = pdsql.read_sql(query, con,
                                                           params=params)
                else: df = pdsql.read_sql(query, con)
```

```
            cur.close()
            break
        except pyodbc.OperationalError:
            time.sleep(60) # one minute could be changed
    return df
```

Прочитайте Чтение SQL Server в DataFrame онлайн:
https://riptutorial.com/ru/pandas/topic/2176/чтение-sql-server-в-dataframe

глава 41: Чтение файлов в pandas DataFrame

Examples

Чтение таблицы в DataFrame

Файл таблицы с заголовком, нижним колонтитулом, именами строк и столбцом индекса:

file: table.txt

```
This is a header that discusses the table file
to show space in a generic table file

index  name     occupation
1      Alice    Salesman
2      Bob      Engineer
3      Charlie  Janitor

This is a footer because your boss does not understand data files
```

код:

```
import pandas as pd
# index_col=0 tells pandas that column 0 is the index and not data
pd.read_table('table.txt', delim_whitespace=True, skiprows=3, skipfooter=2, index_col=0)
```

выход:

```
          name occupation
index
1        Alice   Salesman
2          Bob   Engineer
3      Charlie    Janitor
```

Файл таблицы без имен строк или индекса:

file: table.txt

```
Alice    Salesman
Bob      Engineer
Charlie  Janitor
```

код:

```
import pandas as pd
pd.read_table('table.txt', delim_whitespace=True, names=['name','occupation'])
```

выход:

```
      name occupation
0    Alice   Salesman
1      Bob   Engineer
2  Charlie    Janitor
```

Все варианты можно найти в документации pandas здесь

Чтение файла CSV

Данные с заголовком, разделенные точками с запятой, а не запятыми

файл: table.csv

```
index;name;occupation
1;Alice;Saleswoman
2;Bob;Engineer
3;Charlie;Janitor
```

код:

```
import pandas as pd
pd.read_csv('table.csv', sep=';', index_col=0)
```

выход :

```
          name occupation
index
1        Alice   Salesman
2          Bob   Engineer
3      Charlie    Janitor
```

Таблица без имен строк или индексов и запятых в качестве разделителей

файл: table.csv

```
Alice,Saleswoman
Bob,Engineer
Charlie,Janitor
```

код:

```
import pandas as pd
pd.read_csv('table.csv', names=['name','occupation'])
```

выход:

```
      name occupation
0    Alice   Salesman
1      Bob   Engineer
2  Charlie    Janitor
```

дальнейшее разъяснение можно найти на `read_csv` документации `read_csv`

Собирать данные электронной таблицы Google в базу данных pandas

Иногда нам нужно собирать данные из электронных таблиц Google. Мы можем использовать библиотеки **gspread** и **oauth2client** для сбора данных из электронных таблиц Google. Вот пример для сбора данных:

Код:

```
from __future__ import print_function
import gspread
from oauth2client.client import SignedJwtAssertionCredentials
import pandas as pd
import json

scope = ['https://spreadsheets.google.com/feeds']

credentials = ServiceAccountCredentials.from_json_keyfile_name('your-authorization-file.json',
scope)

gc = gspread.authorize(credentials)

work_sheet = gc.open_by_key("spreadsheet-key-here")
sheet = work_sheet.sheet1
data = pd.DataFrame(sheet.get_all_records())

print(data.head())
```

Прочитайте Чтение файлов в pandas DataFrame онлайн:
https://riptutorial.com/ru/pandas/topic/1988/чтение-файлов-в-pandas-dataframe

кредиты

S. No	Главы	Contributors
1	Начало работы с пандами	Alexander, Andy Hayden, ayhan, Bryce Frank, Community, hashcode55, Nikita Pestrov, user2314737
2	Gotchas of pandas	vlad.rad
3	IO для Google BigQuery	ayhan, tworec
4	JSON	PinoSan, SerialDev, user2314737
5	Meta: Руководство по документации	Andy Hayden, ayhan, Stephen Leppik
6	Pandas Datareader	Alexander, MaxU
7	pd.DataFrame.apply	ptsw, Romain
8	Resampling	jezrael
9	Анализ: объединение всех решений и принятие решений	piRSquared
10	Булевое индексирование данных	firelynx
11	Вычислительные инструменты	Ami Tavory
12	Графики и визуализации	Ami Tavory, Nikita Pestrov, Scimonster
13	Группирование данных временных рядов	ayhan, piRSquared
14	Группировка данных	Andy Hayden, ayhan, danio, Geeklhem, jezrael, 𝕹ѳѳ𝔹𝕀𝔼, QM.py, Romain, user2314737

15	Данные о сдвиге и запаздывании	ASGM
16	Добавление к DataFrame	shahins
17	Дублированные данные	ayhan, Ayush Kumar Singh, bee-sting, jezrael
18	Значения карты	EdChum, Fabio Lamanna
19	Изменение формы и поворот	Albert Camps, ayhan, bernie, DataSwede, jezrael, MaxU, Merlin
20	Индексирование и выбор данных	amin, Andy Hayden, ayhan, double0darbo, jasimpson, jezrael, Joseph Dasenbrock, MaxU, Merlin, piRSquared, SerialDev, user2314737
21	Инструменты ввода-вывода Pandas (считывание и сохранение наборов данных)	amin, Andy Hayden, bernie, Fabich, Gal Dreiman, jezrael, João Almeida, Julien Spronck, MaxU, Nikita Pestrov, SerialDev, user2314737
22	Использование .ix, .iloc, .loc, .at и .iat для доступа к DataFrame	bee-sting, DataSwede, farleytpm
23	Категориальные данные	jezrael, Julien Marrec
24	мультииндексных	Andy Hayden, benten, danielhadar, danio, Pedro M Duarte
25	Объединить, объединить и объединить	ayhan, Josh Garlitos, MaThMaX, MaxU, piRSquared, SerialDev, varunsinghal
26	Отсутствующие данные	Andy Hayden, ayhan, EdChum, jezrael, Zdenek
27	Получение информации о DataFrames	Alexander, ayhan, Ayush Kumar Singh, bernie, Romain, ysearka
28	Праздничные	Romain

	календари	
29	Простое управление DataFrames	Alexander, ayhan, Ayush Kumar Singh, Gal Dreiman, Geeklhem, MaxU, paulo.filip3, R.M., SerialDev, user2314737, ysearka
30	Работа с временными рядами	Romain
31	Работа с категориальными переменными	Gorkem Ozkaya
32	Серии	Alexander, daphshez, EdChum, jezrael, shahins
33	Сечения различных осей с помощью MultiIndex	Julien Marrec
34	Создание DataFrames	Ahamed Mustafa M, Alexander, ayhan, Ayush Kumar Singh, bernie, Gal Dreiman, Geeklhem, Gorkem Ozkaya, jasimpson, jezrael, JJD, Julien Marrec, MaxU, Merlin, pylang, Romain, SerialDev, user2314737, vaerek, ysearka
35	Создание Pandas Play Nice с родными типами Python	DataSwede
36	Сохраните файл данных pandas в файл csv	amin, bernie, eraoul, Gal Dreiman, maxliving, Musafir Safwan, Nikita Pestrov, Olel Daniel, Stephan
37	Строчная манипуляция	ayhan, mnoronha, SerialDev
38	Типы данных	Andy Hayden, ayhan, firelynx, jezrael
39	Чтение MySQL в DataFrame	andyabel, rrawat
40	Чтение SQL Server в DataFrame	bernie, SerialDev
41	Чтение файлов в	Arthur Camara, bee-sting, Corey Petty, Sirajus Salayhin

	pandas DataFrame	