

 免费电子书

学习

pandas

Free unaffiliated eBook created from  
**Stack Overflow contributors.**

#pandas

.....	1
<b>1:</b> .....	<b>2</b>
.....	2
.....	2
Examples.....	2
.....	2
anaconda.....	4
.....	4
.....	5
<b>2: JSON</b> .....	<b>7</b>
Examples.....	7
JSON.....	7
<b>jsonjson</b> .....	<b>7</b>
JSOND3.jsflare.js.....	7
JSON.....	7
<b>3: Meta</b> .....	<b>9</b>
.....	9
Examples.....	9
.....	9
.....	9
.....	10
.....	10
python 23.....	10
<b>4: Pandas IO</b> .....	<b>11</b>
.....	11
Examples.....	11
csvDataFrame.....	11
.....	11
.....	11
.....	11
.....	11

csv.....	12
csv.....	12
DataFrames.....	12
.....	13
read_csv.....	13
.....	13
.....	14
CSV.....	14
read_csv.....	15
CSVDF.....	15
cvspandas.....	15
HDFStore.....	16
<b>dtypesDF.....</b>	<b>16</b>
<b>DF10 * 100.000 = 1.000.000.....</b>	<b>16</b>
<b>HDFStore.....</b>	<b>17</b>
<b>h5 HDFStore[int32int64string].....</b>	<b>17</b>
<b>HDFStore.....</b>	<b>17</b>
.....	17
.....	17
Nginx.....	17
<b>5: pd.DataFrame.apply.....</b>	<b>19</b>
Examples.....	19
pandas.DataFrame.apply.....	19
<b>6: .ix.iloc.loc.at.iatDataFrame.....</b>	<b>20</b>
Examples.....	20
.iloc.....	20
.loc.....	20
<b>7: MultiIndex.....</b>	<b>23</b>
Examples.....	23
.xs.....	23
.loc.....	24

<b>8:</b>	<b>26</b>
Examples	26
.....	26
.....	26
.....	26
.....	26
<b>9: PythonPandas</b>	<b>28</b>
Examples	28
PandasPythonNumpy	28
<b>10:</b>	<b>30</b>
Examples	30
.....	30
.....	30
.....	30
.....	30
.....	31
<b>11:</b>	<b>32</b>
Examples	32
.....	32
.....	32
.....	32
.....	32
<b>pd.qcut - Quintile Buckets</b>	<b>32</b>
.....	33
.....	33
scatter_matrixscatter_matrix	34
.....	35
.....	36
<b>12:</b>	<b>38</b>
.....	38
Examples	38
.....	38
.....	38

<b>13:</b>	.....	<b>40</b>
Examples	.....	40
.....	.....	40
.....	.....	40
.....	.....	40
.....	.....	41
.....	.....	42
.....	.....	42
.....	.....	43
.....	.....	43
transform	.....	43
<b>14:</b>	.....	<b>45</b>
Examples	.....	45
.....	.....	45
<b>15: DataFrame</b>	.....	<b>47</b>
.....	.....	47
Examples	.....	47
DataFrame	.....	47
NumpyDataFrame	.....	47
DictionaryDataFrame	.....	49
DataFrame	.....	49
DataFrame	.....	49
datetimeDataFrame	.....	50
MultiIndexDataFrame	.....	52
pickle.pklDataFrame	.....	52
DataFrame	.....	52
<b>16:</b>	.....	<b>54</b>
.....	.....	54
.....	.....	54
Examples	.....	54
.....	.....	54
DataFrame	.....	55
.....	.....	.....

.....	56
.....	56
.....	56
//.....	56
Concat.....	57
joinmerge.....	58
<b>17:</b> .....	<b>60</b>
Examples.....	60
.....	60
.....	61
matplotlib.....	61
<b>18:</b> .....	<b>63</b>
.....	63
Examples.....	63
.....	63
<b>19:</b> .....	<b>64</b>
Examples.....	64
`get_dummies`.....	64
<b>20:</b> .....	<b>65</b>
Examples.....	65
MultiIndex.....	65
MultiIndexDataFrame.....	66
MultiIndex.....	67
MultiIndex.....	68
MultiIndex.....	69
MultiIndex.....	69
.....	69
<b>21:</b> .....	<b>71</b>
.....	71
Examples.....	71

np.nan.....	71
NA.....	71
.....	71
<b>22:</b> .....	<b>73</b>
Examples.....	73
.....	73
.....	73
.....	74
.....	75
<b>23: MySQLDataFrame</b> .....	<b>77</b>
Examples.....	77
sqlalchemyPyMySQL.....	77
mysqldataframe.....	77
<b>24: pandascsv</b> .....	<b>78</b>
.....	78
Examples.....	78
DataFrame.csv.....	79
Pandas DataFramecsv.....	80
<b>25: SQL ServerDataframe</b> .....	<b>81</b>
Examples.....	81
pyodbc.....	81
pyodbc.....	81
<b>26: pandas DataFrame</b> .....	<b>83</b>
Examples.....	83
DataFrame.....	83
.....	83
.....	83
CSV.....	83
.....	83
.....	84
Googlepandas.....	84
<b>27:</b> .....	<b>86</b>

.....	86
Examples.....	86
DataFrame.....	86
.....	86
.....	87
.....	87
<b>28:</b> .....	<b>89</b>
.....	89
Examples.....	89
.....	89
dtypes.....	90
.....	91
datetime.....	91
timedelta.....	92
dtype.....	92
dtypes.....	93
<b>29: Datareader</b> .....	<b>94</b>
.....	94
Examples.....	94
Datareader.....	94
pandas - .....	95
<b>30: DataFrames</b> .....	<b>97</b>
Examples.....	97
DataFrame.....	97
.....	98
.....	98
.....	99
.....	99
.....	99
.....	99
.....	100
.....	100



.....	100
DataFrame.....	100
DataFrame/.....	101
.....	102
<b>31:</b> .....	<b>103</b>
Examples.....	103
.....	103
.....	103
Pandas.....	104
.....	105
<b>32:</b> .....	<b>107</b>
Examples.....	107
.....	107
.....	107
.....	108
.....	109
.....	110
""RegEx.....	110
<b>DF</b> .....	<b>110</b>
<b>'a'</b> .....	<b>111</b>
<b>RegEx(b c d) - bcd</b> .....	<b>111</b>
<b>a/RegEx</b> .....	<b>111</b>
`.query`/.....	111
DF.....	111
A > 2B < 5.....	112
.query().....	112
.....	112
nn.....	114
.....	115
NaNNoneNaT.....	116
<b>33:</b> .....	<b>118</b>
.....	118

Examples.....	118
.....	118
.....	118
.....	118
.....	118
DataFrame.....	118
.....	119
.....	119
.....	119
3.....	119
.....	119
.....	120
<b>34: DataFrame.....</b>	<b>122</b>
Examples.....	122
DataFrame.....	122
DataFrame.....	122
Dataframe.....	123
<b>35: .....</b>	<b>124</b>
Examples.....	124
.....	124
<b>36: .....</b>	<b>125</b>
Examples.....	125
.....	125
<b>37: Google BigQueryIO.....</b>	<b>126</b>
Examples.....	126
BigQuery.....	126
BigQuery.....	127
<b>38: .....</b>	<b>128</b>
Examples.....	128
.....	128
.....	128
.....	131

.....	133
.....	135
CSV.....	135
<b>39:</b> .....	<b>137</b>
Examples.....	137
.....	137
.....	137
.....	138
.....	139
<b>40:</b> .....	<b>141</b>
Examples.....	141
.....	141
<b>41: DataFrame</b> .....	<b>143</b>
Examples.....	143
DataFrame.....	143
DataFrameDataFrame.....	144
.....	145

---

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.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

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](mailto:info@zzzprojects.com)

# 1:

PandasPython“”“”。Python。

Pandas 。

0.19.1	2016113
0.19.0	2016102
0.18.1	201653
0.18.0	2016313
0.17.1	20151121
0.17.0	2015109
0.16.2	2015612
0.16.1	2015511
0.16.0	2015322
0.15.2	2014-12-12
0.15.1	2014-11-09
0.15.0	20141018
0.14.1	2014711
0.14.0	2014531
0.13.1	201423
0.13.0	201413
0.12.0	2013723

## Examples

pandas 。

**Anaconda**

pandasNumPySciPy。

pandas Python SciPy IPython NumPy Matplotlib.....[Anaconda](#) Linux Mac OS X Windows Python

pandas SciPy

Anaconda

Anaconda

Anaconda Anaconda

## Miniconda

Anaconda pandas

[Miniconda](#) pandas

[Conda](#) Anaconda pip virtualenv

[Miniconda](#) Python [Conda](#)

Conda Miniconda

conda virtualenv Python

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

Python

```
source activate name_of_my_env
```

Windows

```
activate name_of_my_env
```

pandas

```
conda install pandas
```

pandas

```
conda install pandas=0.13.1
```

IPython

```
conda install ipython
```

Anaconda

```
conda install anaconda
```

## pipconda pip

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

## pandas

```
pip install pandas
```

## NumPy

## anaconda

Continuum [anaconda](#) ◦ Windows / OSX shell OSX / Linux ◦

---

anaconda150 [miniconda](#) ◦ Windows shell OSX / Linux ◦

## miniconda pandas

```
conda install pandas
```

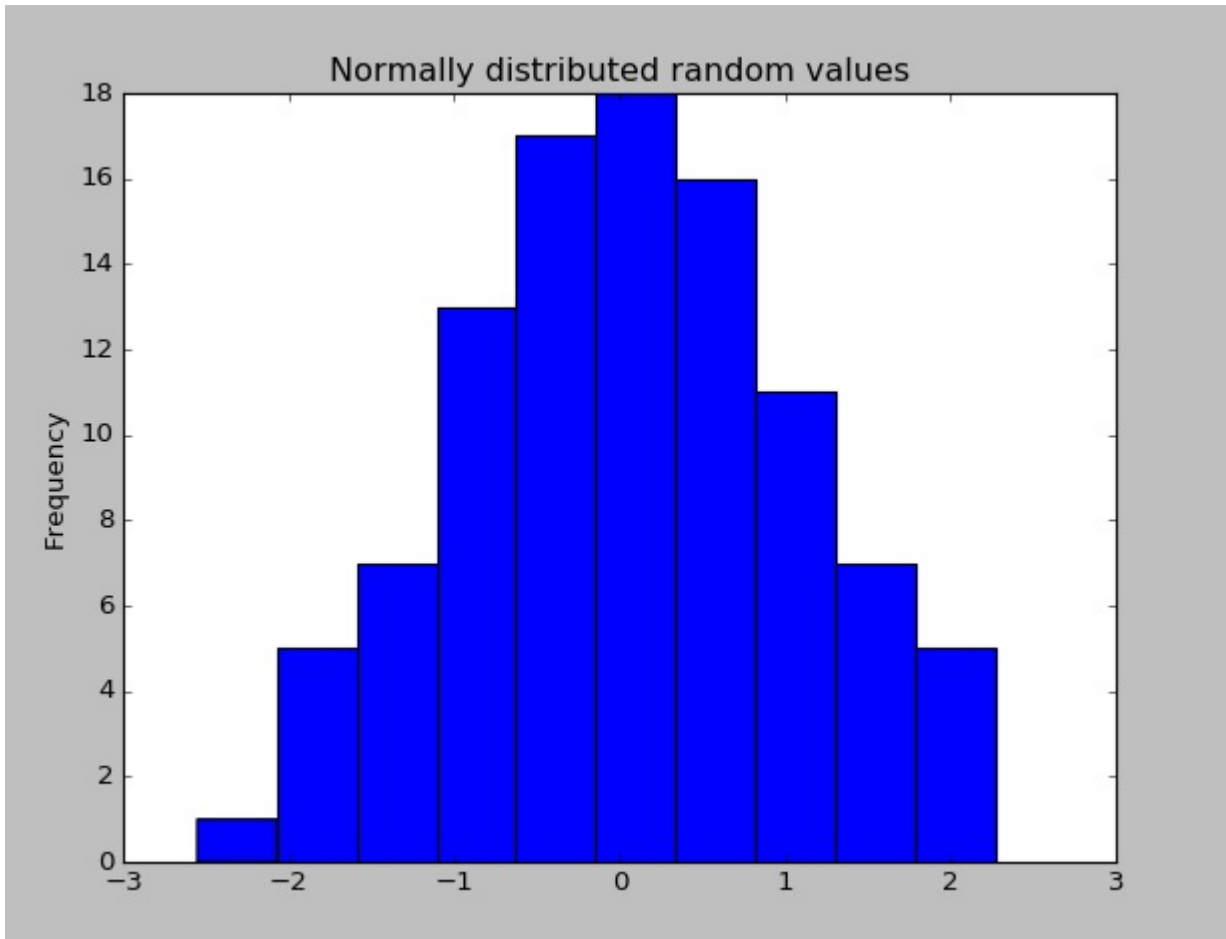
---

## anaconda miniconda

```
conda update pandas
```

## Pandas

```
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()
```



```
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
```

`.describe()` pandas

```
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

C°

```
In [4]: df['C'].describe()
```

```
Out[4]:
```

```
count      10
unique       3
freq        5
Name: C, dtype: object
```

°

<https://riptutorial.com/zh-CN/pandas/topic/796/>

## 2: JSON

### Examples

#### JSON

#### jsonjson

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

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

#### JSOND3.jsflare.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")
```

#### JSON

## file.jsonJSON

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

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

JSON <https://riptutorial.com/zh-CN/pandas/topic/4752/json>

## 3: Meta

python<http://stackoverflow.com/documentation/python/394/meta-documentation-guidelines#t=201607240058406359521> ◦

/:)

## Examples

ipython

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

```
In [12]: df
```

```
Out[12]:
```

```
   0  1
0  1  2
1  3  4
```

python

```
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
```

◦

◦ ipython

```
In [21]: [print(col) for col in df]
```

```
0
```

```
1
```

```
Out[21]: [None, None]
```

pandas<sub>pd</sub>

```
import pandas as pd
```

PEP8

- 4
- `kwargs` (a=1)
- 80

“”。

`sort_values`。

Out。

```
a
# Out: 1
```

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

## python 23

```
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)
```

Meta <https://riptutorial.com/zh-CN/pandas/topic/3253/meta->

# 4: Pandas IO

pandasIO Tools

## Examples

### csvDataFrame

data\_file.csv

```
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
```

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

	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

- **sep**, ◦ `pd.read_csv('data_file.csv', sep=';')`
- **index\_col**`index_col = n pandas.DataFrame`◦

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

	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**◦ `skip_blank_lines=FalseNaN`

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

	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**◦

```

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

```

01 0

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

```

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

```

◦

```

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

```

```

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

```

◦

## 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')

```

## CSV

### csvpandas

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

## DataFrames

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

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

[read\\_excel](#) [read\\_csv](#) ◦

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

## 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

files ◦ [DataFrames](#) [concat](#)

```
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
```

```
import pandas as pd

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

## CSV

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

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

### Default" -

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

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

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

## 'utf-8'encoding

```
df.to_csvfile_nameencoding='utf-8'
```

## read\_csv

parse\_dates◦

*input.csv*

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

```
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)
```

*parse\_dates*

*date\_parser*

## CSVDF

```
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())
```

CSVpd.concat()axis=1

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

## cvspandas

```
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
```

names

```
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
```

## HDFStore

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

---

## dtypesDF

```
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
```

---

## DF10 \* 100.000 = 1.000.000

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

---

## HDFStore

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

---

## h5 HDFStore[int32int64string]

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

---

## 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}
```

```
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}
```

```
store.close()
```

---

## Nginx

sep

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

**Pandas IO** <https://riptutorial.com/zh-CN/pandas/topic/2896/pandas-io-->

# 5: pd.DataFrame.apply

## Examples

### pandas.DataFrame.apply

pandas.DataFrame.apply DataFrame - DataFrame DataFrame DataFrameSeries ◦

```
# 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
```

pd.DataFrame.apply <https://riptutorial.com/zh-CN/pandas/topic/7024/pd-dataframe-apply>

# 6: .ix.iloc.loc.at.iatDataFrame

## Examples

### .iloc

.ilocDataFrame

DataFrame

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

DataFrame

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

.iloc

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

```
1
```

.

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

```
print df
```

	one	two
a	1	6
b	2	21
c	3	8
d	4	9
e	5	10

.loc

.loc

DataFrame

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

## DataFrame

```
print df
```

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

`.loc` 'c' 'two' 33

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

## DataFrame

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

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

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

`ac` ° °

	one	two
a	1	6
b	2	7
c	3	8

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

`b""b` ° °

b	7
c	8
d	9

`.loc.iloc` ° °



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

'b'

```
7
```

[.ix.iloc.loc.at.iatDataFrame](https://riptutorial.com/zh-CN/pandas/topic/7074/-ix--iloc--loc--at-iat-dataframe) <https://riptutorial.com/zh-CN/pandas/topic/7074/-ix--iloc--loc--at-iat-dataframe>

# 7: MultiIndex

## Examples

`.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` level axis 0<sup>1</sup>

`.xs` pandas.Series pandas.DataFrame ◦

```
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
```

```
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**

```
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
```

**.loc**

**.xs** 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
```

```
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
```

```
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

```

```
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

```

**.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

```

**MultiIndex** <https://riptutorial.com/zh-CN/pandas/topic/8099/multiindex>

---

# 8:

## Examples

◦

```
import pandas as pd
import numpy as np

# The number of sample to generate
nb_sample = 100

# Seeding to obtain a reproducible 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
```

◦ ◦

---

```
se.head(2).append(se.tail(2))

# 2016-09-24    44
# 2016-09-25    47
# 2016-12-31    85
# 2017-01-01    48
```

---

◦

```
se['2017']

# 2017-01-01    48
```

```
se['2017-01']

# 2017-01-01    48
```

```
se['2017-01-01']
```

```
# 48
```

◦

```
se['2016-12-31':'2017-01-01']
```

```
# 2016-12-31    85  
# 2017-01-01    48
```

**pandas** `afterbeforetruncate` ◦

```
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
```

<https://riptutorial.com/zh-CN/pandas/topic/7029/>

# 9: PythonPandas

## Examples

### PandasPythonNumpy

```
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
```

### python

```
In [3]: df['A'].tolist()
```

```
Out[3]: [1, 2, 3]
```

### DataFrames.tolist() ◦ 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.__getattr__(self, name)  
    2745  
    2746     def __setattr__(self, name, value):
```

```
AttributeError: 'DataFrame' object has no attribute 'tolist'
```

### numpy

```
In [5]: df['B'].values
```

```
Out[5]: array([ 1.,  2.,  3.])
```

### numpy

```
In [6]: df.values
```

```
Out[6]:
```

```
array([[1, 1.0, 'a', True],  
       [2, 2.0, 'b', False],  
       [3, 3.0, 'c', True]], dtype=object)
```

```
In [7]: df['C'].to_dict()
Out[7]: {0: 'a', 1: 'b', 2: 'c'}
```

## DataFrame

```
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}}
```

to\_dict◦ **dicts**

```
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}]
```

◦

**PythonPandas** <https://riptutorial.com/zh-CN/pandas/topic/8008/pythonpandas>



# 10:

## Examples

◦ - ◦

```
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)
    ]
```

◦

```
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='BMS')
```

```
# dtype='datetime64[ns]', freq=None)
```

o o

```
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
```

<https://riptutorial.com/zh-CN/pandas/topic/7976/>

# 11:

## Examples

◦  
/◦ ◦  
5◦ 5.310.5◦ ◦ ◦

---

- 
- - ◦
  - ◦ /◦ ◦ 0◦

```
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
```

ID◦

```
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)
```

m[0]25 ◦ 1e-7◦

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

---

## pd.qcut - Quintile Buckets

pd.qcut◦

```
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)
```

```
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()
```

---

```
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))

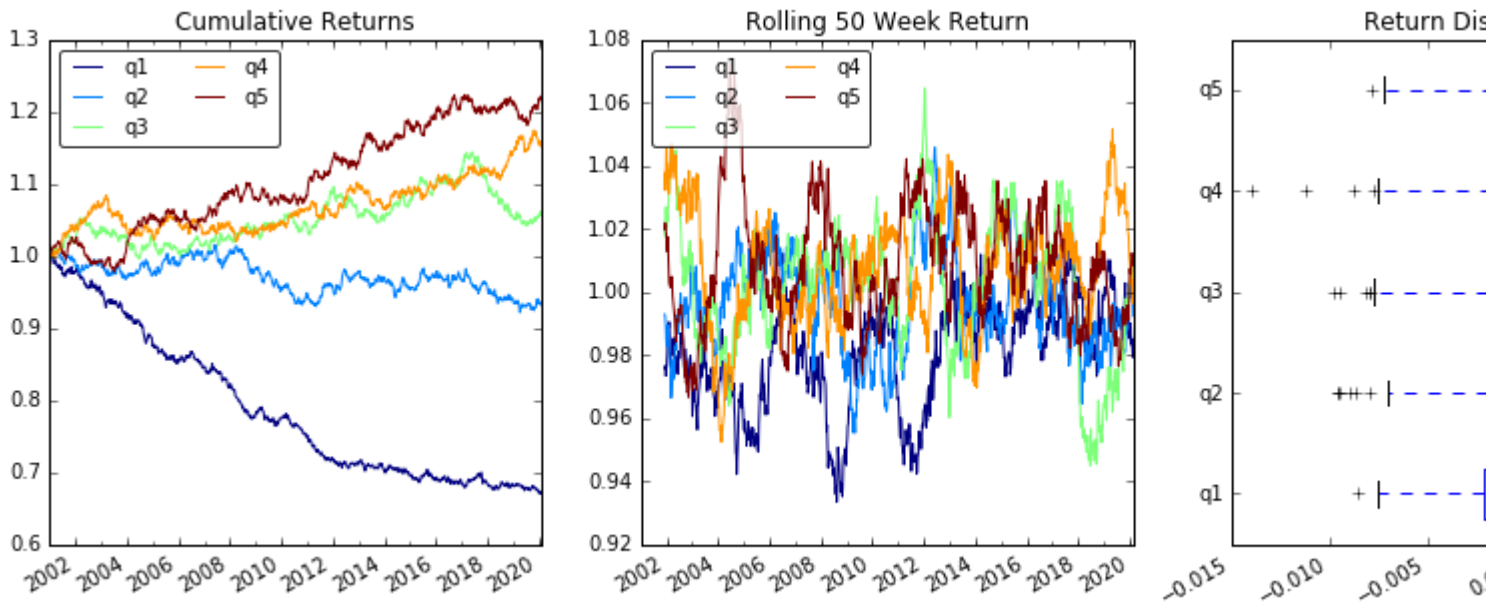
# Cumulative Returns
returns_cut.add(1).cumprod() \
    .plot(colormap='jet', ax=ax1, title="Cumulative Returns")
leg1 = ax1.legend(loc='upper left', ncol=2, prop={'size': 10}, fancybox=True)
leg1.get_frame().set_alpha(.8)

# 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()

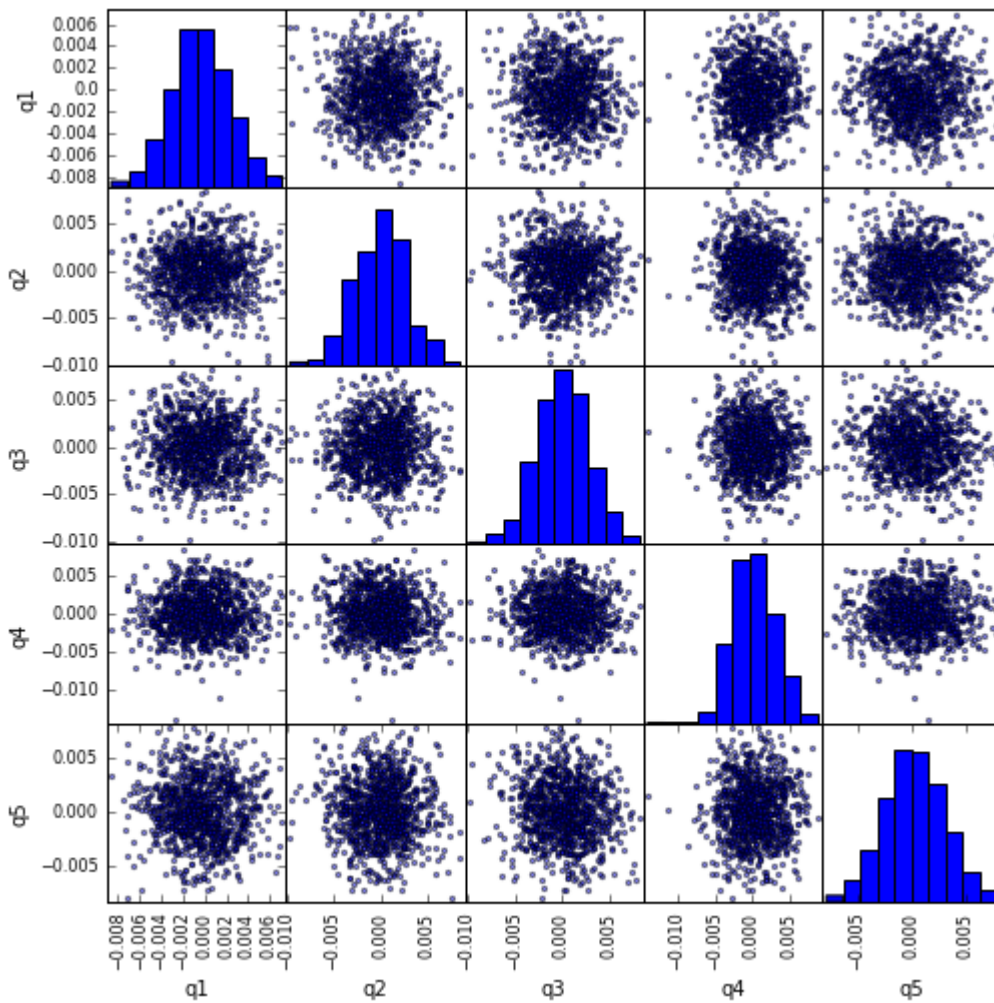
plt.show()
```



`scatter_matrix`  
`scatter_matrix`

```
from pandas.tools.plotting import scatter_matrix

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



```

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.argmax()
    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)

```

```
max_dd_df(returns_cut)
```

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

```

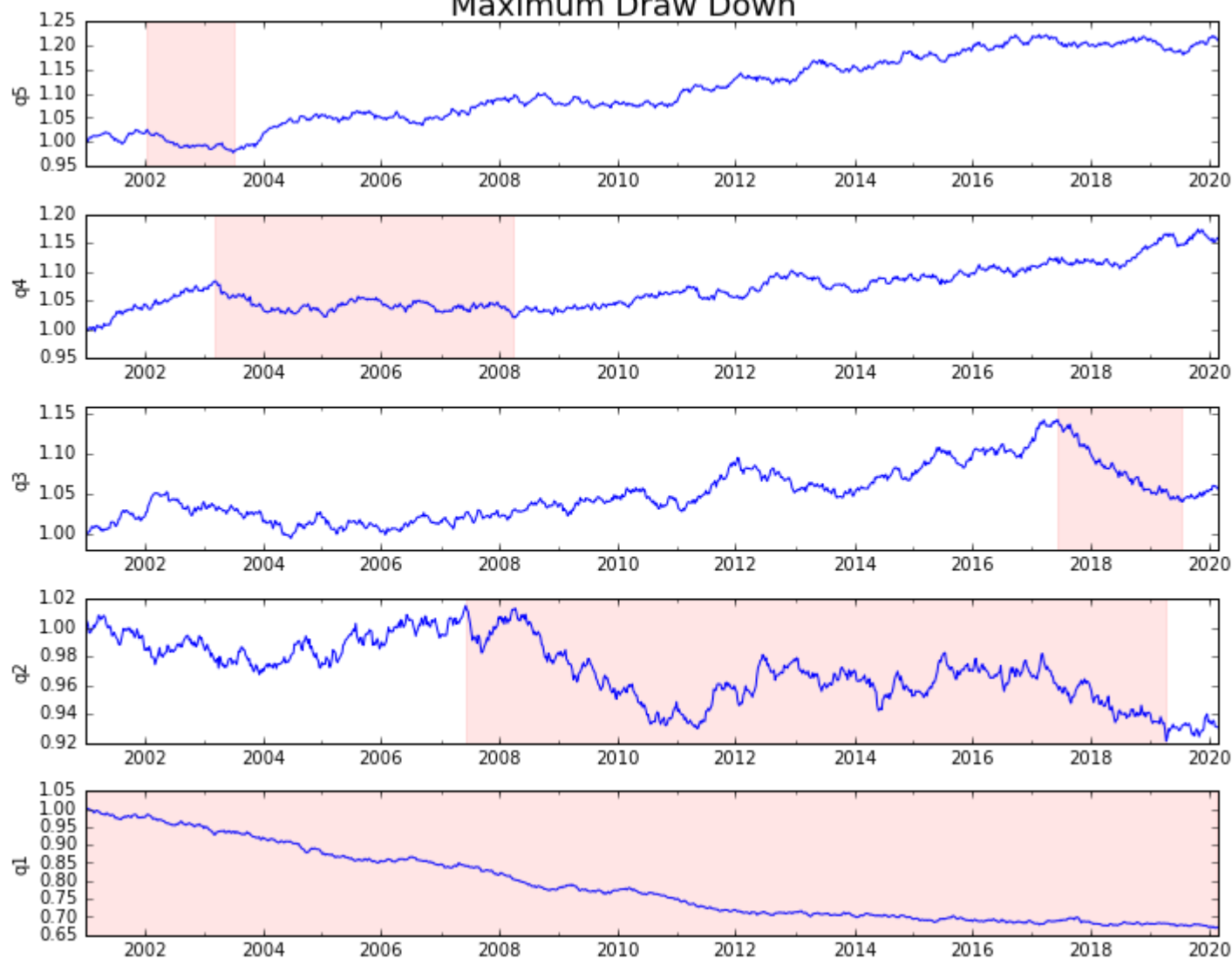
draw_downs = max_dd_df(returns_cut)

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)

```

## Maximum Draw Down



o o

```
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')
```

describe°

```
describe(returns_cut)
```

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

° ° °

[https://riptutorial.com/zh-CN/pandas/topic/5238/-](https://riptutorial.com/zh-CN/pandas/topic/5238/)



# 12:

;R. .

## Examples

```
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
```

```
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)
```

<https://riptutorial.com/zh-CN/pandas/topic/3887/>

# 13:

## Examples

### 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
```

### A

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

```
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
```

### DataFrame **MultIndex** AB.

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
```

## 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
```

Age◦

- `n - n`
- `-bins=[19, 40, 65, np.inf]` (19, 40] (40, 65] (65, np.inf] ◦

Pandas◦ labels◦

```
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]]
```

groupby**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
```

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

## groupby

```
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
```

## agg

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

## sizecount

[sizeNaN](#) [countcount](#) ◦

```
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
```

```
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

```
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

groupby() ◦ (Category, DataFrame) ◦

```
# 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', '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))
```

## transform

```
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

```

group1group2B。 groupby.transform

```
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

```

<https://riptutorial.com/zh-CN/pandas/topic/1822/>

# 14:

## Examples

```
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
```

### 715. ◦

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

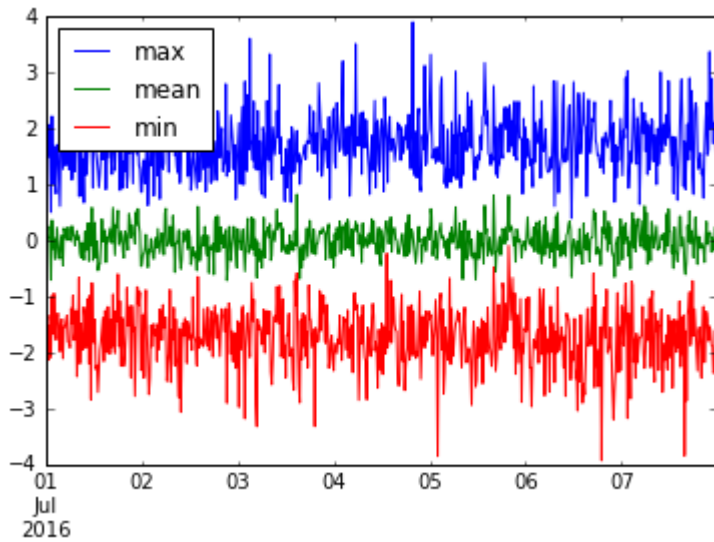
# We could take the last value.
ts.resample('15T').last()
```

### groupby ◦

#### ◦ resample('15M')min meanmax ◦

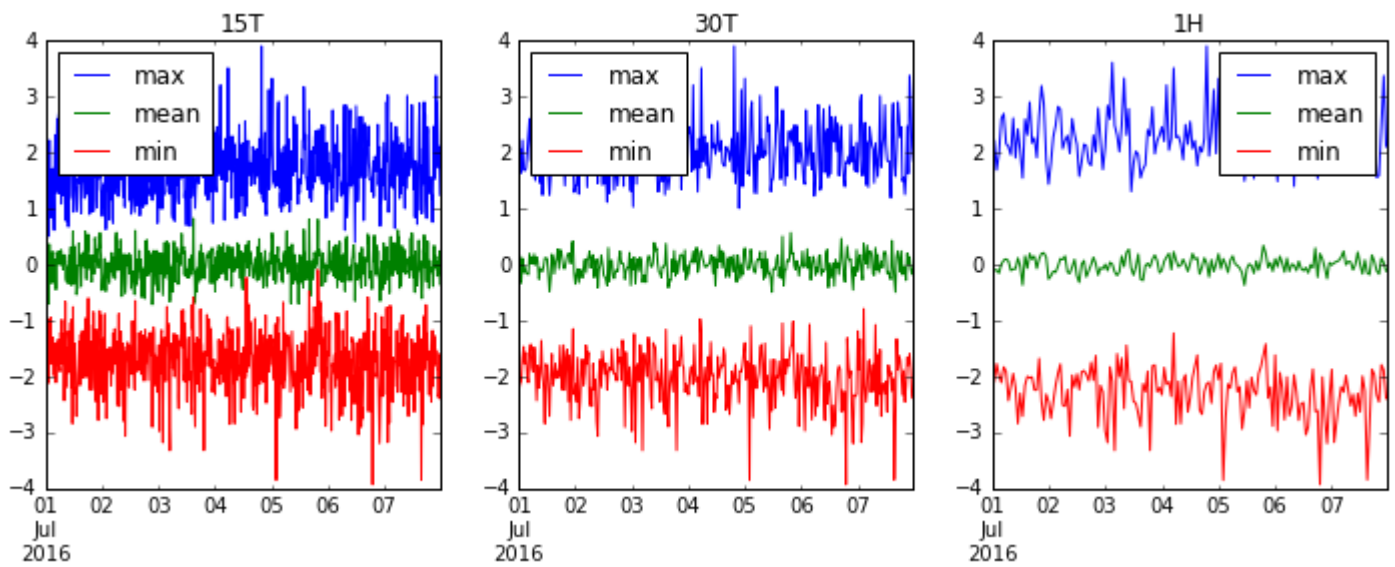
```
ts.resample('15T').agg(['min', 'mean', 'max']).plot()
```





'15T' 15 '30T' '1H' 1。

```
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)
```



<https://riptutorial.com/zh-CN/pandas/topic/4747/>

# 15: DataFrame

DataFrame pandas Series Panel

0..N DataFrame

/DataFrame Numpy

## Examples

### DataFrame

```
import pandas as pd
```

DataFrame numbers colors

```
df = pd.DataFrame({'numbers': [1, 2, 3], 'colors': ['red', 'white', 'blue']})
```

dataframe

```
print(df)
# Output:
#   colors  numbers
# 0    red         1
# 1  white         2
# 2   blue         3
```

dict Pandas 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
```

### Numpy DataFrame

DataFrame

```
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
```

DataFrame

```
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
```

DataFrame DataFrame NaT, NaN, 'nan', None

```
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[:,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
```

## DictionaryDataFrame

```
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

## DataFrame

DataFrame。 DataFrame。

```
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

## DataFrame

dictDataFrame。 ndarray。 / ndarray。

```
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
```

```
df = pd.DataFrame({'A' : [1, 2, 3, 4], 'B' : [5, 5, 5]}) # a ValueError is raised
```

## 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
```

[http //pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists](http://pandas.pydata.org/pandas-docs/stable/dsintro.html#from-dict-of-ndarrays-lists)

## datetimeDataFrame

```
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 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
```

### date\_range freq Offset-aliases

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 end frequency
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

## MultIndexDataFrame

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

### 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'])
```

### from\_product

```
idx = pd.MultiIndex.from_product(['bar', 'baz', 'foo', 'qux'], ['one', 'two'])
```

## MultIndex

```
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

## pickle.plkDataFrame

```
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)
```

## DataFrame

### DataFrame

```
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
```

NaN

```
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
```

DataFrame <https://riptutorial.com/zh-CN/pandas/topic/1595/dataframe>



# 16:

- **merge** `how='inner'on=Noneleft_on=Noneright_on=Noneleft_index=Falseright_index=False`  
`sort=False``suffixes=('_x'_y')``copy=True`
- DataFrame
- DataFrame

	{'left','right','outer','inner'}'inner'
	◦ DataFrame DataFrame
right_on	◦ DataFrameleft_on/
left_index	False◦ DataFrame◦ MultiIndexDataFrame
right_index	False◦ DataFrame◦ left_index
	False◦ DataFrame
	2.....◦
	True◦ False
	False◦ True“_merge”DataFrame◦ stringDataFramesstring◦ “”DataFrame “left_only”“”DataFrame“right_only”◦

## Examples

### 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

## T3

```
id    x     y     s1     s2     s3
8     42    1.9    55     56     58
9     30    1.9    57     58     60
```

```
s1 s2s3 id3
```

```
join onDataFrameDataFrame
```

```
df = df1.merge(df2.groupby('id')['signal'].apply(lambda xx: xx.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
```

```
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
```

## DataFrame

```
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
```

---

## 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
```

## Dataframe.

```
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
```

---

## DataFrame.

```
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
```

---

## 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
```

---

## 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
```

//

```

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

```

//[df1df2df3] -

```

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

```

//

```

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

```

## Concat

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

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

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

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

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

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

`.join`

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

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

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

## joinmerge

### dataframes leftright

```
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

`join` ◦ `join` ◦ A

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

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

4 ◦  $2_{\text{left}} 2_{\text{right}}$  ◦

```
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

◦ ◦

join◦

```
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**

merge◦ mergemerge◦ mergeon◦

merge◦

'A'◦

```
left.merge(right)
```

	A	B	C
0	a	1	3
1	b	2	4

[0, 1]['X', 'Y']

left\_indexright\_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

join◦

[https://riptutorial.com/zh-CN/pandas/topic/1966/-](https://riptutorial.com/zh-CN/pandas/topic/1966/)

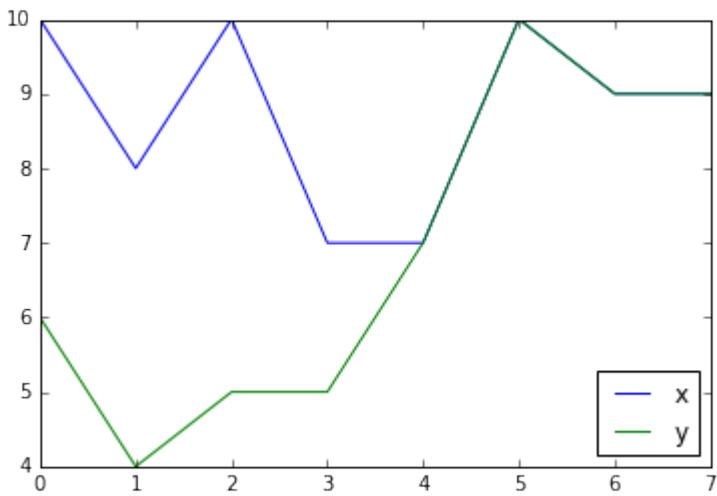
# 17:

## Examples

Pandas: [matplotlib](#).

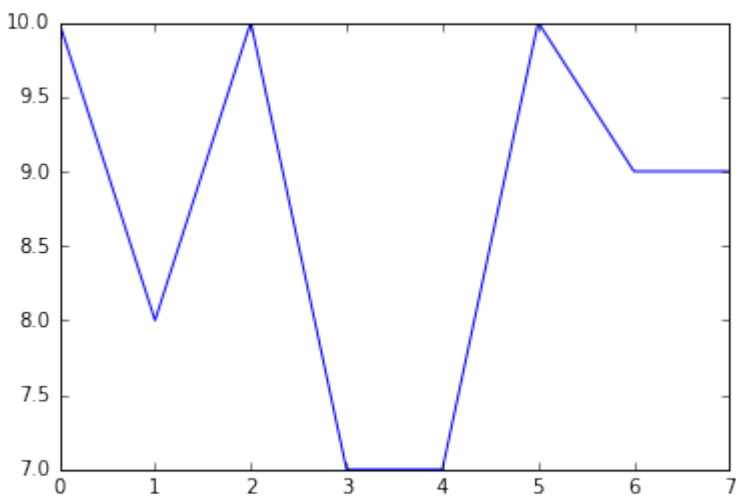
### DataFrameSeries

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



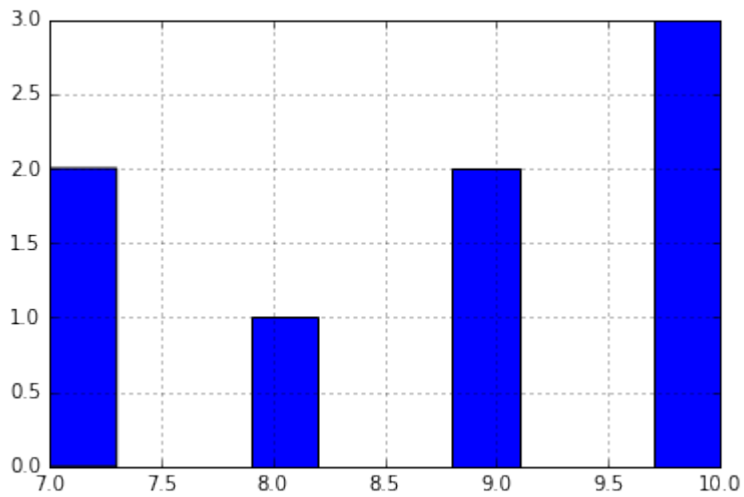
### Series

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



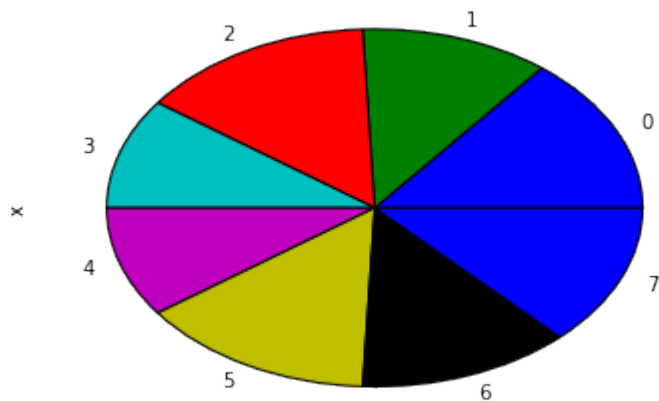
hist() ◦

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



◦ **kind**◦

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



◦

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

plot() matplotlib◦

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

## matplotlib

plot()◦ 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
```

<https://riptutorial.com/zh-CN/pandas/topic/3839/>

# 18:

KeyError mergeget

## Examples

df

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

S

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

map

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

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

<https://riptutorial.com/zh-CN/pandas/topic/3928/>

# 19:

## Examples

### `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
```

<https://riptutorial.com/zh-CN/pandas/topic/5999/>

# 20:

## Examples

### Multindex

#### 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

A

```
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')
```

```
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')
```

```
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

```
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

XS

```
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
```

## MultilIndexDataFrame

### 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
```

MultilIndex◦ level=0 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
```

`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

```

## Multindex

MultiIndexpandas.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

```

```

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

```

```
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
```

```
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

```

## MultIndex

### MultIndexDataFrame

```

# 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

```

## MultIndex◦

```
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

## MultIndex

### 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

### MultIndexMultiIndexdf.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

## MultIndex

### MultIndexDataFrame◦ DataFramecolumns◦

```
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

### “”MultIndex◦

```
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
```

<https://riptutorial.com/zh-CN/pandas/topic/3840/>

# 21:

◦ ◦

Pandas◦

## Examples

### np.nan

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

```
col
0    False
1    False
```

-

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

```
col
0    False
1     True
```

### NA

Pandasinteger◦

```
df= pd.read_csv("data.csv", dtype={'grade': int})
error: Integer column has NA values
```

floatdtype◦

### [1,2]dataframe dfNaN

```
import pandas as pd

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

   col
3   NaN
4   NaN
```

120134

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

	col
1	2.0
2	NaN

**.values**

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

	col
3	1
4	2

<https://riptutorial.com/zh-CN/pandas/topic/6425/>

# 22:

## Examples

```
# 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')
```

◦

.str.slice() .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
```

```
In [3]: ser.str[0]
Out[3]:
0    L
1    d
2    c
dtype: object
```

```
In [4]: ser.str[:3]
Out[4]:
0    Lor
1    dol
2    con
dtype: object
```

```
In [5]: ser.str[-1]
Out[5]:
0    m
1    t
2    t
dtype: object
```

```
In [6]: ser.str[-3:]
Out[6]:
0    sum
1    met
2    lit
dtype: object
```

```
In [7]: ser.str[:10:2]
Out[7]:
0    Lrmis
1    dlrst
2    cnett
dtype: object
```

## PandasPython◦ Python

```
In [8]: 'Lorem ipsum'[12]
# IndexError: string index out of range
```

```
In [9]: 'Lorem ipsum'[12:15]
Out[9]: ''
```

## PandasNaN

```
In [10]: ser.str[12]
Out[10]:
0    NaN
1     e
2     a
dtype: object
```

```
In [11]: ser.str[12:15]
Out[11]:
0
1    et
2    adi
dtype: object
```

`str.contains()` **Series**◦ `str.startswith()` `str.endswith()`◦

```
In [1]: animals = pd.Series(['cat', 'dog', 'bear', 'cow', 'bird', 'owl', 'rabbit', 'snake'])
```

'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
```

'a'

```
In [3]: animals[animals.str.contains('a')]
Out[3]:
0      cat
2      bear
6      rabbit
7      snake
dtype: object
```

str.startswithstr.endswith

```
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
```

```
In [1]: ser = pd.Series(['lORem ipSuM', 'Dolor sit amet', 'Consectetur Adipiscing Elit'])
```

```
In [2]: ser.str.upper()
Out[2]:
0          LOREM IPSUM
1          DOLOR SIT AMET
2  CONSECTETUR ADIPISCING ELIT
dtype: object
```

```
In [3]: ser.str.lower()
Out[3]:
0          lorem ipsum
1          dolor sit amet
2  consectetur adipiscing elit
dtype: object
```

```
In [4]: ser.str.capitalize()
Out[4]:
0          Lorem ipsum
1          Dolor sit amet
2  Consectetur adipiscing elit
dtype: object
```

```
In [5]: ser.str.title()
Out[5]:
0          Lorem Ipsum
1          Dolor Sit Amet
2  Consectetur Adipiscing Elit
dtype: object
```

```
In [6]: ser.str.swapcase()
Out[6]:
0          LorEM IPsUm
1          dOLOR SIT AMET
2  cONSECTETUR aDIPISCING eLIT
```

```
dtype: object
```

◦

```
In [7]: ser = pd.Series(['LOREM IPSUM', 'dolor sit amet', 'Consectetur Adipiscing Elit'])
```

```
In [8]: ser.str.islower()
```

```
Out[8]:
```

```
0    False
```

```
1     True
```

```
2    False
```

```
dtype: bool
```

```
In [9]: ser.str.isupper()
```

```
Out[9]:
```

```
0     True
```

```
1    False
```

```
2    False
```

```
dtype: bool
```

```
In [10]: ser.str.istitle()
```

```
Out[10]:
```

```
0    False
```

```
1    False
```

```
2     True
```

```
dtype: bool
```

<https://riptutorial.com/zh-CN/pandas/topic/2372/>

# 23: MySQLDataFrame

## Examples

### sqlalchemyPyMySQL

```
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)
```

### mysqldataframe

#### pandas

```
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
```

MySQLDataFrame <https://riptutorial.com/zh-CN/pandas/topic/8809/mysqldataframe>



## 24: pandascsv

path_or_buf	“”。
	”。
na_rep	stringdefault"
float_format	stringdefault
	sequence
	True。
	booleandefault True
index_label	False。 NoneheaderindexTrue。 DataFrameMultiIndex。 False。 index_label = FalseR
nanRep	na_rep
	str Python'w'
	stringoptionalPython 2“ascii”Python 3“utf-8”。
	string'gzip"bz2"xz'
line_terminator	stringdefault'n'
	csvcsv.QUOTE_MINIMAL
quotechar	string1'''
	booleanTruequotechar
escapechar	stringlength 1defaultsepquotechar
CHUNKSIZE	intNone
tupleize_cols	booleandefault Falsesmulti_indexTruenewFalse
	stringdefault
	'。 '。 "

### Examples

## DataFrame.csv

### 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

### CSV

```
df.to_csv('example.csv', index=False)
```

### 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
```

index=False #s 0,1,2,3,4CSV

```
df.to_csv('example.csv', index=True) # Or just leave off the index param; default is True
```

### 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
```

header=False

```
df.to_csv('example.csv', index=False, header=False)
```

## 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
```

sep= **argumentcsv**, ' ' ◦

```
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
```

## Pandas DataFramecsv

```
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')
```

**pandascsv** <https://riptutorial.com/zh-CN/pandas/topic/1558/pandascsv>

# 25: SQL ServerDataframe

## Examples

### pyodbc

```
import pandas.io.sql
import pyodbc
import pandas as pd
```

```
# Parameters
server = 'server_name'
db = 'database_name'
UID = 'user_id'
```

```
# Create the connection
conn = pyodbc.connect('DRIVER={SQL Server};SERVER=' + server + ';DATABASE=' + db + '; UID = '
+ UID + '; PWD = ' + UID + 'Trusted_Connection=yes')
```

### pandas

```
# Query into dataframe
df= pandas.io.sql.read_sql('sql_query_string', conn)
```

### pyodbc

```
import os, time
import pyodbc
import pandas.io.sql as pdsq

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 = pdsq.read_sql(query, con,
                                                         params=params)
                else: df = pdsq.read_sql(query, con)
                cur.close()
            break
        except pyodbc.OperationalError:
            time.sleep(60) # one minute could be changed
    return df
```



# 26: pandas DataFrame

## Examples

### DataFrame

#### filetable.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
```

```
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)
```

```
      name occupation
index
1      Alice    Salesman
2       Bob     Engineer
3   Charlie    Janitor
```

#### filetable.txt

```
Alice    Salesman
Bob       Engineer
Charlie   Janitor
```

```
import pandas as pd
pd.read_table('table.txt', delim_whitespace=True, names=['name', 'occupation'])
```

```
      name occupation
0   Alice    Salesman
1     Bob     Engineer
2  Charlie    Janitor
```

### pandas

### CSV

#### filetable.csv

```
index;name;occupation
1;Alice;Saleswoman
2;Bob;Engineer
3;Charlie;Janitor
```

```
import pandas as pd
pd.read_csv('table.csv', sep=';', index_col=0)
```

```
      name occupation
index
1      Alice  Salesman
2        Bob   Engineer
3    Charlie   Janitor
```

## filetable.csv

```
Alice,Saleswoman
Bob,Engineer
Charlie,Janitor
```

```
import pandas as pd
pd.read_csv('table.csv', names=['name','occupation'])
```

```
      name occupation
0    Alice  Salesman
1       Bob   Engineer
2  Charlie   Janitor
```

[read\\_csv](#)

## Googlepandas

- **gspreadoauth2client**◦

```
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())
```

pandas DataFrame <https://riptutorial.com/zh-CN/pandas/topic/1988/pandas-dataframe>



# 27:

DataFrame.ix .loc .iloc

## Examples

### DataFrame

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

```
   color
True  red
False blue
True  red
False blue
```

.loc

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

.iloc

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

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

```
pandas.iloc[True].iloc[1].iloc[1]
```

.ix

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

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

.ix ° ° .iloc.loc

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

\_\_getitem\_\_[ ]◦ TrueFalse

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

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

==pd.Series

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

pd.Seriesnp.arraylist \_\_getitem\_\_[ ]◦

```
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
```

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

◦

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

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

.locpd.Series pd.DataFrame ◦ ◦

.loc °

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

<https://riptutorial.com/zh-CN/pandas/topic/9589/>

## 28:

`dtypes`。 `numpy`。

`dtypepython`。

`pd.Series`。 `dtypefloat`。

`astype`“”。

```
pd.Series([1.,2.,3.,4.,5.]).astype(object)
0    1
1    2
2    3
3    4
4    5
dtype: object
```

`dtype`。 `python`。

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(object)[0])
float
```

“”。

```
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
```

`dtypeobjectstring`。 `numpy`。

```
type(pd.Series([1.,2.,3.,4.,5.]).astype(str)[0])
str
```

`dtypes`。 `dtype`。

## Examples

`.dtypes`。 `.dtypes` attribute。

```
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
```

.dtype°

```
In [4]: df['A'].dtype
```

```
Out[4]: dtype('int64')
```

## dtypes

astype() Series astype() Series°

```
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
```

## AfloatB

```
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().astype(float64').astype(float32).astype(float16) ◦ pd.to_numeric pd.to_datetime  
pd.to_timedelta ◦
```

```
pd.to_numeric◦
```

```
In [6]: pd.to_numeric(df['E'])  
Out[6]:  
0    1  
1    2  
2    3  
Name: E, dtype: int64
```

```
pd.to_numeric◦ 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
```

[isnull boolean indexing](#)

```
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
```

## 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]
```

## 2.1.2011201121。201112dayfirst。

```
In [13]: pd.to_datetime('2.1.2011', dayfirst=True)
Out[13]: Timestamp('2011-01-02 00:00:00')
```

## timedelta

```
In [14]: pd.to_timedelta(df['D'])
Out[14]:
0    1 days
1    2 days
2    3 days
Name: D, dtype: timedelta64[ns]
```

## dtype

### select\_dtypesdtype。

```
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
```

### includeexclude

```
# 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
```

## dtypes

`get_dtype_counts` [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
```

<https://riptutorial.com/zh-CN/pandas/topic/2959/>



# 29: Datareader

## Pandas datareader

- 
- Google
- St.Louis FEDFRED
- Kenneth French
- 
- 

◦

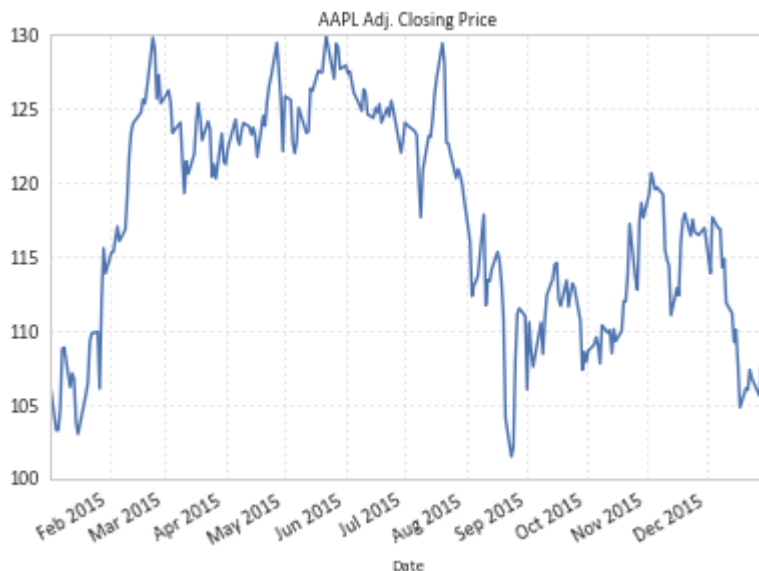
## Examples

### Datareader

```
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')
```



## pandas -

```
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 -

```
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')
```

```
In [390]: p['Adj Close']
Out[390]:
```

Date	AAPL	AMZN	COP	FB	GOOG
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]:
```

Date	AAPL	AMZN	COP	FB	GOOG
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]:
```

Date	Open	High	Low	Close	Volume	Adj Close
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

**Datareader** <https://riptutorial.com/zh-CN/pandas/topic/1912/datareader>

# 30: DataFrames

## Examples

### DataFrame

#### 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
```

#### 1del

```
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
```

#### 2drop

```
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
```

#### 3drop with column number

```
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
```

```
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
```

```
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
```

```
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
```

```
df.columns
```

```
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
```

o

```
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
```

```
df['C'] = [7, 8, 9]
```

```
print(df)
```

```
# Output:
```

```
#   A  B  C
# 0  1  4  7
# 1  2  5  8
# 2  3  6  9
```

```
df['C'] = 1
```

```
print(df)
```

```
# Output:
```

```
#   A  B  C
# 0  1  4  1
# 1  2  5  1
# 2  3  6  1
```

```
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
```

```
a = [1, 2, 3]
```

```
b = [4, 5, 6]
```

```
c = [x*y for (x,y) in zip(a,b)]
```

```
print(c)
```

```
# Output:
```

```
# [1, 32, 729]
```

```
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
```

```
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
```

```
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
```

```
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
```

01

```
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
```

## DataFrame

### 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
```

```
   C1  C2  C3
0   1   2   3
1   a   b   c
```

```
[10,11,12]
```

```
df = pd.DataFrame(np.array([[10,11,12]]), \
                  columns=["C1", "C2", "C3"]).append(df, ignore_index=True)
print df
```

```
   C1  C2  C3
0  10  11  12
1   1   2   3
2   a   b   c
```

## DataFrame/

### 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
```

```
df.drop([0,4], inplace=True)
```

```
df.drop([0,4], inplace=True)

print(df)
# Output
#   a  b
# 1  2  3
# 2  4  5
# 3  6  7
```

```
df = df.drop([0,4])
```

```
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
```

```
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
```

```
# 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]
```

DataFrames <https://riptutorial.com/zh-CN/pandas/topic/6694/dataframes>

# 31:

## Examples

◦ ◦

```
import pandas as pd

s = pd.Series([10, 20, 30])

>>> s
0    10
1    20
2    30
dtype: int64
```

◦ 01.◦

```
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
```

```
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]
```

## Pandas

```
>>> 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
```

## Series

**s**

```
>>> len(s)
8
```

**s**

```
>>> s[4]
8
```

**s**

```
>>> s.loc[2]
6
```

**s**

```
>>> s[1:3]
1    4
2    6
dtype: int64
```

**5**

```
>>> s[s > 5]
2    6
4    8
5    7
dtype: int64
```

```
>>> s.min()
1
>>> s.max()
8
>>> s.mean()
4.75
>>> s.std()
2.2519832529192065
```

## Seriesfloat

```
>>> 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
```

## snumpy

```
>>> s.values
array([1, 4, 6, 3, 8, 7, 4, 5])
```

## s

```
>>> d = s.copy()
>>> d
0    1
1    4
2    6
3    3
4    8
5    7
6    4
7    5
dtype: int64
```

## PandasSeries。

```
>>> 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
```

```
>>> def square(x):
...     return x*x
```

## squares

```
>>> t = s.apply(square)
>>> t
0      9
1     49
2     25
3     64
4     81
5      1
6      0
7     16
dtype: int64
```

## lambda

```
>>> s.apply(lambda x: x ** 2)
0      9
1     49
2     25
3     64
4     81
5      1
6      0
7     16
dtype: int64
```

```
>>> q = pd.Series(['Bob', 'Jack', 'Rose'])
>>> q.apply(str.lower)
0      bob
1     jack
2     rose
dtype: object
```

## Series

```
>>> q.str.lower()
0      bob
1     jack
2     rose
dtype: object
>>> q.str.len()
0      3
1      4
2      4
```

<https://riptutorial.com/zh-CN/pandas/topic/1898/>

## 32:

### Examples

```
# 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']

```

	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

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

`iloc` ◦ 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
```

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

o

```
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
```

R0R2

```
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
```

locilociloc

```
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
```

CE

```
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
```

```
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
```

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

.ix

```
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: []
```



o

```
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
```

[pandas](#) o

“”RegEx

# 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
```

## '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
```

## RegEx (b|c|d) - bcd

```
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
```

## ^/RegEx

```
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
```

## `.query`/

```
import pandas as pd
```

## 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

**A > 2B < 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
```

**.query()**

```
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
```

◦ ◦

s◦

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

# n is number of observations
n = 5000

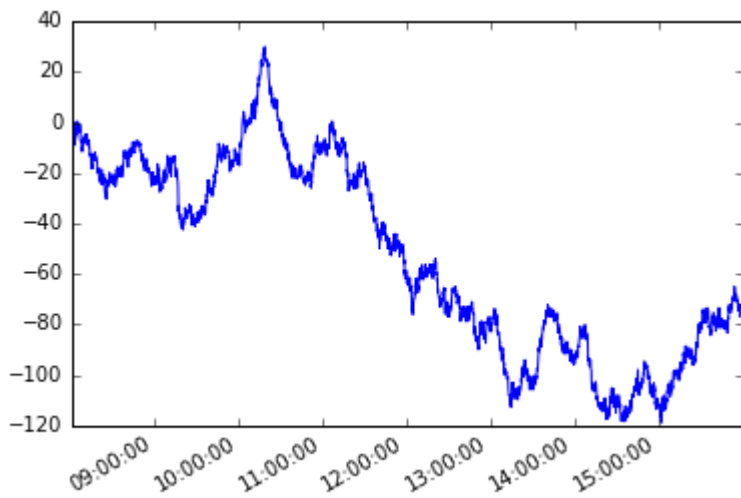
day = pd.to_datetime(['2013-02-06'])
# irregular seconds spanning 28800 seconds (8 hours)
seconds = np.random.rand(n) * 28800 * pd.Timedelta(1, 's')
# start at 8 am
start = pd.offsets.Hour(8)
# irregular timeseries
```

```

tidx = day + start + seconds
tidx = tidx.sort_values()

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

```



◦ x ◦

python◦

```

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

```

moves

```

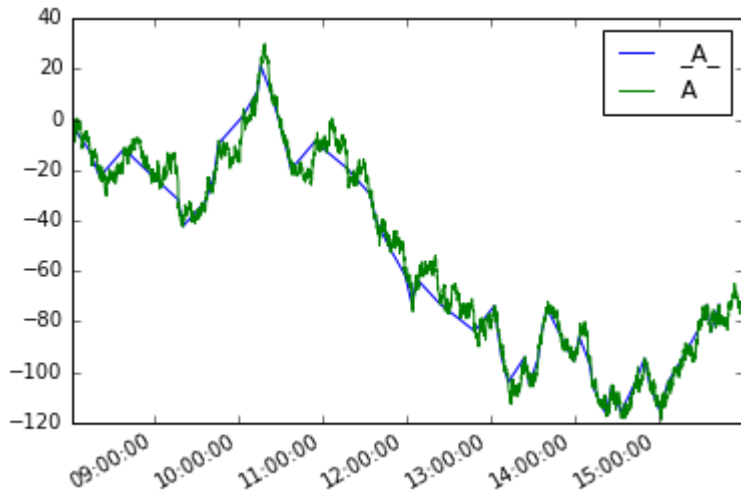
moves = pd.Series({i:v for i, v in mover(s, move_size=10)},
                  name='_{}_{}'.format(s.name))

```

```

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

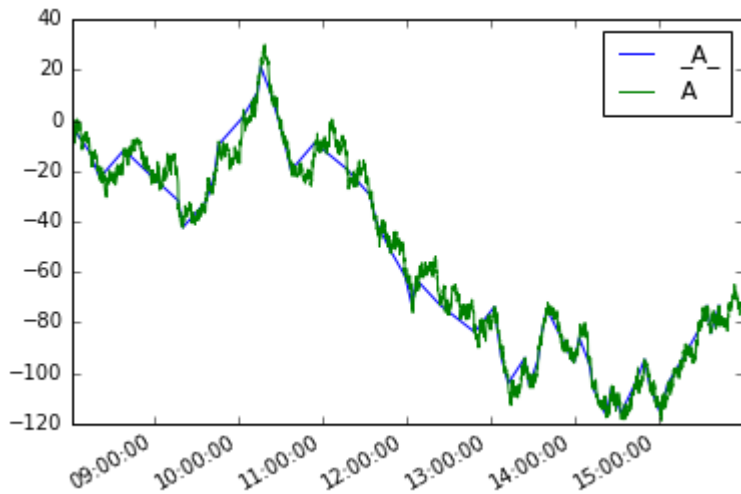
```



```
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)
```



## nn

headtail

`nDataFrame.head([n])`

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

`nDataFrame.tail([n])`

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

n5。

head / tail

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

```
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
```

col\_1Series.unique()

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

Series.unique。

col\_1 SQLcol\_2 DataFrame.drop\_duplicates()

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

。

```
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
```

```
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
```

[pandas](#)。

## NaNNoneNaT

pd.NaT NaN pd.NaT None

```
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](#)

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

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
```

inplace = True inplace = True

◦

<https://riptutorial.com/zh-CN/pandas/topic/1751/>



# 33:

ffillbfill

## Examples

```
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
```

```
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
```

**DataFrame**◦ `inplace df.fillna(0, inplace=True) df = df.fillna(0)`

```
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
```

```
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
```

## 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
```

None pythonNaN

```
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
```

```
In [12]: df.dropna()
```

```
Out[12]:
   0      1      2      3
2  7.0  8.0  9.0   10.0
```

**DataFrame** ◦ inplace `df.dropna(inplace=True)` `df = df.dropna()`

```
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
```

### 3

```
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
```

```
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

**NaN** `isnull()` `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
```

`np.nan == np.nan` **False** `np.nan`

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

**Series** `DataFrames`

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

**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
```

<https://riptutorial.com/zh-CN/pandas/topic/1896/>

# 34: DataFrame

## Examples

### DataFrame

#### DataFrame

```
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
```

#### 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
```

#### DataFrame

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

#### DataFrame

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

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

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

```
df.columns
```

## Dataframe

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

◦ **NA** / count mean std

- min
- 25% **Q1**
- 50% **Q2**
- 75% **Q3**
- max

```
>>> df.describe()  
count      A      B      C      D      E  
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
```

DataFrame <https://riptutorial.com/zh-CN/pandas/topic/6697/dataframe>

# 35:

## Examples

### DataFrame

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

```
>>> 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
```

### Pearson ◦ 1◦

`pd.DataFrame.correlation`method◦ pearson ◦ Spearman

```
>>> 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
```

<https://riptutorial.com/zh-CN/pandas/topic/5620/>

# 36:

## Examples

```
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
```

`.shift()` periods ◦ 1 ◦

<https://riptutorial.com/zh-CN/pandas/topic/7554/>



# 37: Google BigQueryIO

## Examples

### BigQuery

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

### BigQueryBigQuery◦

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

Your browser has been opened to visit:

```
https://accounts.google.com/o/oauth2/v2/auth...[loong 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
```

### ◦ pandas

```
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.
```

```
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

### pandasjsonbigquery\_credentials.dat

```
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
```

## BigQuery

### jsonpandas

```
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
```

Google BigQueryIO <https://riptutorial.com/zh-CN/pandas/topic/5610/google-bigqueryio>

# 38:

## Examples

### 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
```

### 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
```

```
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

## pivot

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

## ValueError

## pivot\_table

```
#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
```

## 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
```

string

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

## DataError

### aggragating

```
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     None
```

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

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

```
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

Male	NaN	37.0	NaN
Programmer Female	31.0	29.0	NaN

```
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

```
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

```
In [38]: df.pivot_table(index='Position', aggfunc={'Age': np.mean, 'Random': [np.mean, np.std]})
Out[38]:
```

```
Out[38]:
```

	Age	Random	std
	mean	mean	
Position			
Manager	34.333333	0.325122	0.306106
Programmer	32.333333	0.423712	0.477219

```
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())
second      A      B
           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](#) 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
```

**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

```
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'])
```

	Heart Disease	N	Y
Sex			
F		2	3
M		3	2

```
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

## 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



```
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

```
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

```
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

## 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

## DF

```
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
```

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

>>> 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
```

## CSV

```
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)
```

```
      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
```

<https://riptutorial.com/zh-CN/pandas/topic/1463/>

# 39:

## Examples

`0BASeries.duplicatedDataFrame.ixSeries.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
```

~

```
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
```

`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

```

```

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

```

```

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

```

```

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
```

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

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

◦

```
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
```

## AB

```
In [22]: df["A"].unique()
```

```
Out[22]: array([1, 2, 3])
```

```
In [23]: df["B"].unique()
```

```
Out[23]: array([5, 4, 3, 6, 7])
```

## A unique()

```
In [24]: pd.unique(df['A']).tolist()
```

```
Out[24]: [1, 2, 3]
```

◦ 'B'A'1。

◦ 4'4'6'B'

```
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
```

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

## DataFrame

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

A1TrueFalse DataFrame'B'DataFrame

unique. "B""A"1

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

<https://riptutorial.com/zh-CN/pandas/topic/2082/>

# 40:

## Examples

```
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





# 41: DataFrame

## Examples

### 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: []
```

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

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

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

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

```
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
```

### .loc []

```
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

## DataFrameDataFrame

### DataFrame

```
In [7]: df1
```

```
Out[7]:
```

	A	B
0	a1	b1
1	a2	b2

```
In [8]: df2
```

```
Out[8]:
```

	B	C
0	b1	c1

### DataFrame。 appendDataFrame。 DataFrame。 DataFrame

```
In [9]: df1.append(df2)
```

```
Out[9]:
```

	A	B	C
0	a1	b1	NaN
1	a2	b2	NaN
0	NaN	b1	c1

### 0。 PandasDataFrame

```
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

DataFrame <https://riptutorial.com/zh-CN/pandas/topic/6456/dataframe>

S. No		Contributors
1		<a href="#">Alexander</a> , <a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">Bryce Frank</a> , <a href="#">Community</a> , <a href="#">hashcode55</a> , <a href="#">Nikita Pestrov</a> , <a href="#">user2314737</a>
2	JSON	<a href="#">PinoSan</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
3	Meta	<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">Stephen Leppik</a>
4	Pandas IO	<a href="#">amin</a> , <a href="#">Andy Hayden</a> , <a href="#">bernie</a> , <a href="#">Fabich</a> , <a href="#">Gal Dreiman</a> , <a href="#">jezrael</a> , <a href="#">João Almeida</a> , <a href="#">Julien Spronck</a> , <a href="#">MaxU</a> , <a href="#">Nikita Pestrov</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
5	pd.DataFrame.apply	<a href="#">ptsw</a> , <a href="#">Romain</a>
6	.ix.iloc.loc.at.iat DataFrame	<a href="#">bee-sting</a> , <a href="#">DataSwede</a> , <a href="#">farleytpm</a>
7	MultIndex	<a href="#">Julien Marrec</a>
8		<a href="#">Romain</a>
9	PythonPandas	<a href="#">DataSwede</a>
10		<a href="#">Romain</a>
11		<a href="#">piRSquared</a>
12		<a href="#">jezrael</a> , <a href="#">Julien Marrec</a>
13		<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">danio</a> , <a href="#">Geeklhern</a> , <a href="#">jezrael</a> , <a href="#">NooBIE</a> , <a href="#">QM.py</a> , <a href="#">Romain</a> , <a href="#">user2314737</a>
14		<a href="#">ayhan</a> , <a href="#">piRSquared</a>
15	DataFrame	<a href="#">Ahamed Mustafa M</a> , <a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bernie</a> , <a href="#">Gal Dreiman</a> , <a href="#">Geeklhern</a> , <a href="#">Gorkem Ozkaya</a> , <a href="#">jasimpson</a> , <a href="#">jezrael</a> , <a href="#">JJD</a> , <a href="#">Julien Marrec</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a> , <a href="#">pylang</a> , <a href="#">Romain</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a> , <a href="#">vaerek</a> , <a href="#">ysearka</a>
16		<a href="#">ayhan</a> , <a href="#">Josh Garlitos</a> , <a href="#">MaThMaX</a> , <a href="#">MaxU</a> , <a href="#">piRSquared</a> , <a href="#">SerialDev</a> , <a href="#">varunsinghal</a>
17		<a href="#">Ami Tavory</a> , <a href="#">Nikita Pestrov</a> , <a href="#">Scimonster</a>
18		<a href="#">EdChum</a> , <a href="#">Fabio Lamanna</a>
19		<a href="#">Gorkem Ozkaya</a>

20		<a href="#">Andy Hayden</a> , <a href="#">benten</a> , <a href="#">danielhadar</a> , <a href="#">danio</a> , <a href="#">Pedro M Duarte</a>
21		<a href="#">vlad.rad</a>
22		<a href="#">ayhan</a> , <a href="#">mnoronha</a> , <a href="#">SerialDev</a>
23	MySQLDataFrame	<a href="#">andyabel</a> , <a href="#">rrawat</a>
24	pandascsv	<a href="#">amin</a> , <a href="#">bernie</a> , <a href="#">eraoul</a> , <a href="#">Gal Dreiman</a> , <a href="#">maxliving</a> , <a href="#">Musafir Safwan</a> , <a href="#">Nikita Pestrov</a> , <a href="#">Olel Daniel</a> , <a href="#">Stephan</a>
25	SQL Server Dataframe	<a href="#">bernie</a> , <a href="#">SerialDev</a>
26	pandas DataFrame	<a href="#">Arthur Camara</a> , <a href="#">bee-sting</a> , <a href="#">Corey Petty</a> , <a href="#">Sirajus Salayhin</a>
27		<a href="#">firelynx</a>
28		<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">firelynx</a> , <a href="#">jezrael</a>
29	Datareader	<a href="#">Alexander</a> , <a href="#">MaxU</a>
30	DataFrames	<a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">Gal Dreiman</a> , <a href="#">Geeklhem</a> , <a href="#">MaxU</a> , <a href="#">paulo.filip3</a> , <a href="#">R.M.</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a> , <a href="#">ysearka</a>
31		<a href="#">Alexander</a> , <a href="#">daphshez</a> , <a href="#">EdChum</a> , <a href="#">jezrael</a> , <a href="#">shahins</a>
32		<a href="#">amin</a> , <a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">double0darbo</a> , <a href="#">jasimpson</a> , <a href="#">jezrael</a> , <a href="#">Joseph Dasenbrock</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a> , <a href="#">piRSquared</a> , <a href="#">SerialDev</a> , <a href="#">user2314737</a>
33		<a href="#">Andy Hayden</a> , <a href="#">ayhan</a> , <a href="#">EdChum</a> , <a href="#">jezrael</a> , <a href="#">Zdenek</a>
34	DataFrame	<a href="#">Alexander</a> , <a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bernie</a> , <a href="#">Romain</a> , <a href="#">ysearka</a>
35		<a href="#">Ami Tavory</a>
36		<a href="#">ASGM</a>
37	Google BigQueryIO	<a href="#">ayhan</a> , <a href="#">tworec</a>
38		<a href="#">Albert Camps</a> , <a href="#">ayhan</a> , <a href="#">bernie</a> , <a href="#">DataSwede</a> , <a href="#">jezrael</a> , <a href="#">MaxU</a> , <a href="#">Merlin</a>
39		<a href="#">ayhan</a> , <a href="#">Ayush Kumar Singh</a> , <a href="#">bee-sting</a> , <a href="#">jezrael</a>
40		<a href="#">jezrael</a>
41	DataFrame	<a href="#">shahins</a>