数据科学 2：Pandas 库

QuantEcon, 张帆

2019 年 12 月 16 日

1 数据处理

该部分将介绍 Pandas 库，它是一个强大的分析结构化数据的工具集；它的使用基础是 Numpy（提供高性能的矩阵运算）；用于数据挖掘和数据分析，同时也提供数据清洗功能。

相关资料：

QuantEcon-Pandas

Pandas 中文

Pandas 官网

1.1 1. 基础概念

本部分将围绕 Series 和 DataFrame 两个概念做简要介绍

在开始之前，我们先导入 pandas 并起个别名"pd"

```
In [3]: import pandas as pd

        %matplotlib inline
```

1.1.1 （1）Series

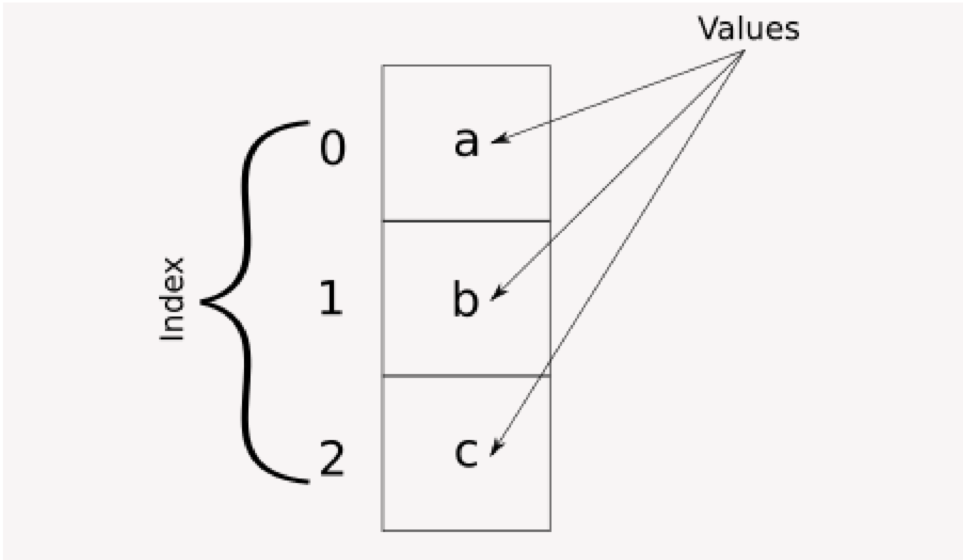
它是一种类似于一维数组的对象，是由一组数据 (各种 NumPy 数据类型) 以及一组与之相关的数据标签 (即索引) 组成。仅由一组数据也可产生简单的 Series 对象。

通过调用 pd.Series 函数即可创建 Series，这里以美国失业率数据为例。

```
In [3]: values = [5.6, 5.3, 4.3, 4.2, 5.8, 5.3, 4.6, 7.8, 9.1, 8., 5.7]
        years = list(range(1995, 2017, 2))

        unemp = pd.Series(data=values, index=years, name="Unemployment")

In [4]: unemp
```



Out[4]:
```
1995    5.6
1997    5.3
1999    4.3
2001    4.2
2003    5.8
2005    5.3
2007    4.6
2009    7.8
2011    9.1
2013    8.0
2015    5.7
Name: Unemployment, dtype: float64
```

In [6]:
```
# 索引
unemp.index
```

Out[6]:
```
Int64Index([1995, 1997, 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015], dtype='i
```

In [7]:
```
# 值
unemp.values
```

Out[7]:
```
array([5.6, 5.3, 4.3, 4.2, 5.8, 5.3, 4.6, 7.8, 9.1, 8. , 5.7])
```

In [8]:
```
# 头部数据
unemp.head()
```

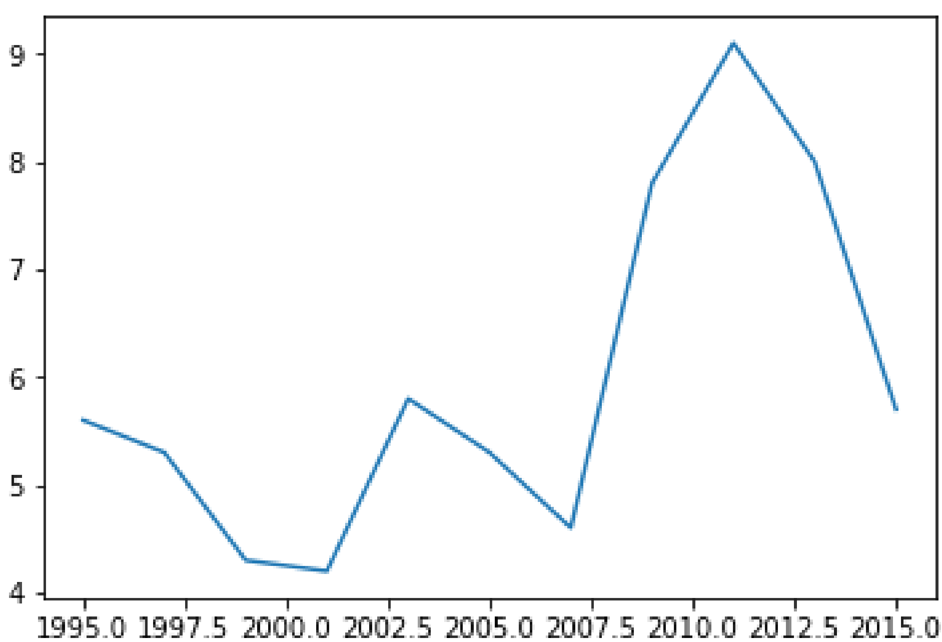
Out[8]: 1995   5.6
        1997   5.3
        1999   4.3
        2001   4.2
        2003   5.8
        Name: Unemployment, dtype: float64

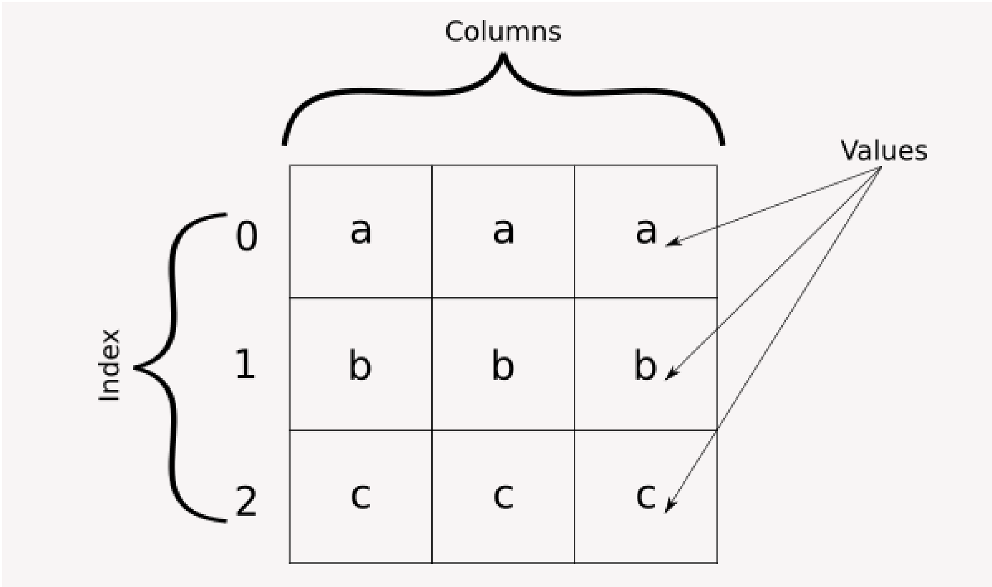
In [9]:
```
# 尾部数据
unemp.tail()
```

Out[9]: 2007   4.6
        2009   7.8
        2011   9.1
        2013   8.0
        2015   5.7
        Name: Unemployment, dtype: float64

In [13]:
```
# 趋势
unemp.plot()
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1ea1aabd2e8>






```
In [14]: # 唯一值
         unemp.unique()
```

```
Out[14]: array([5.6, 5.3, 4.3, 4.2, 5.8, 4.6, 7.8, 9.1, 8. , 5.7])
```

```
In [15]: # 特定值
         unemp.loc[1995]
```

```
Out[15]: 5.6
```

```
In [16]: unemp.loc[[1995, 2005, 2015]]
```

```
Out[16]: 1995    5.6
         2005    5.3
         2015    5.7
         Name: Unemployment, dtype: float64
```

1.1.2 (2) DataFrame

DataFrame 是 Pandas 中的一个表格型的数据结构，包含有一组有序的列，每列可以是不同的值类型 (数值、字符串、布尔型等)，DataFrame 即有行索引也有列索引，可以被看做是由 Series 组成的字典。

通过调用 pd.DataFrame 函数即可创建 DataFrame，同样地，这里以美国失业率数据为例。

```
In [17]: data = {
            "NorthEast": [5.9,  5.6,  4.4,  3.8,  5.8,  4.9,  4.3,  7.1,  8.3,  7.9,  5.7],
```

```
    "MidWest": [4.5,  4.3,  3.6,  4. ,  5.7,  5.7,  4.9,  8.1,  8.7,  7.4,  5.1],
    "South": [5.3,  5.2,  4.2,  4. ,  5.7,  5.2,  4.3,  7.6,  9.1,  7.4,  5.5],
    "West": [6.6, 6., 5.2, 4.6, 6.5, 5.5, 4.5, 8.6, 10.7, 8.5, 6.1],
    "National": [5.6, 5.3, 4.3, 4.2, 5.8, 5.3, 4.6, 7.8, 9.1, 8., 5.7]
}

unemp_region = pd.DataFrame(data, index=years)
unemp_region
```

Out[17]:

	NorthEast	MidWest	South	West	National
1995	5.9	4.5	5.3	6.6	5.6
1997	5.6	4.3	5.2	6.0	5.3
1999	4.4	3.6	4.2	5.2	4.3
2001	3.8	4.0	4.0	4.6	4.2
2003	5.8	5.7	5.7	6.5	5.8
2005	4.9	5.7	5.2	5.5	5.3
2007	4.3	4.9	4.3	4.5	4.6
2009	7.1	8.1	7.6	8.6	7.8
2011	8.3	8.7	9.1	10.7	9.1
2013	7.9	7.4	7.4	8.5	8.0
2015	5.7	5.1	5.5	6.1	5.7

In [18]:

```
# 索引
unemp_region.index
```

Out[18]:

```
Int64Index([1995, 1997, 1999, 2001, 2003, 2005, 2007, 2009, 2011, 2013, 2015], dtype=
```

In [19]:

```
# 值
unemp_region.values
```

Out[19]:

```
array([[ 5.9,  4.5,  5.3,  6.6,  5.6],
       [ 5.6,  4.3,  5.2,  6. ,  5.3],
       [ 4.4,  3.6,  4.2,  5.2,  4.3],
       [ 3.8,  4. ,  4. ,  4.6,  4.2],
       [ 5.8,  5.7,  5.7,  6.5,  5.8],
       [ 4.9,  5.7,  5.2,  5.5,  5.3],
       [ 4.3,  4.9,  4.3,  4.5,  4.6],
       [ 7.1,  8.1,  7.6,  8.6,  7.8],
       [ 8.3,  8.7,  9.1, 10.7,  9.1],
```

```
       [ 7.9,  7.4,  7.4,  8.5,  8. ],
       [ 5.7,  5.1,  5.5,  6.1,  5.7]])
```

In [20]:
```
# 头部
unemp_region.head()
```

Out[20]:

	NorthEast	MidWest	South	West	National
1995	5.9	4.5	5.3	6.6	5.6
1997	5.6	4.3	5.2	6.0	5.3
1999	4.4	3.6	4.2	5.2	4.3
2001	3.8	4.0	4.0	4.6	4.2
2003	5.8	5.7	5.7	6.5	5.8

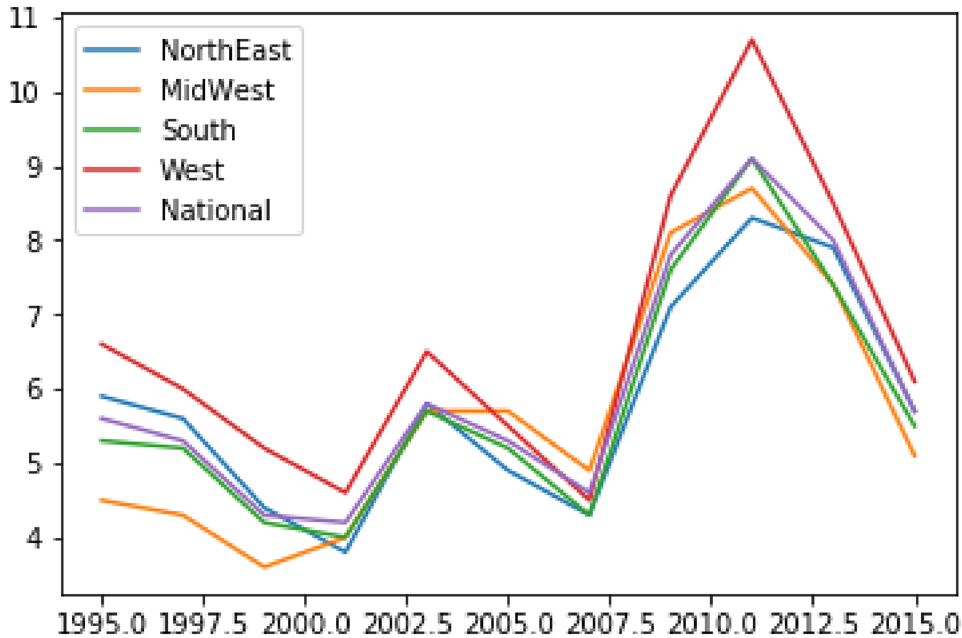
In [21]:
```
# 尾部
unemp_region.tail()
```

Out[21]:

	NorthEast	MidWest	South	West	National
2007	4.3	4.9	4.3	4.5	4.6
2009	7.1	8.1	7.6	8.6	7.8
2011	8.3	8.7	9.1	10.7	9.1
2013	7.9	7.4	7.4	8.5	8.0
2015	5.7	5.1	5.5	6.1	5.7

In [22]:
```
# 趋势
unemp_region.plot()
```

Out[22]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1ea1ab2cb38>
```



In [23]:
```
# 特定值
unemp_region.loc[1995, "NorthEast"]
```

Out[23]: 5.9

In [24]:
```
unemp_region.loc[[1995, 2005], "South"]
```

Out[24]:
```
1995    5.3
2005    5.2
Name: South, dtype: float64
```

In [25]:
```
unemp_region.loc[1995, ["NorthEast", "National"]]
```

Out[25]:
```
NorthEast    5.9
National     5.6
Name: 1995, dtype: float64
```

In [26]:
```
unemp_region.loc[:, "NorthEast"]
```

Out[26]:
```
1995    5.9
1997    5.6
1999    4.4
```

```
2001    3.8
2003    5.8
2005    4.9
2007    4.3
2009    7.1
2011    8.3
2013    7.9
2015    5.7
Name: NorthEast, dtype: float64
```

In [27]:
```
unemp_region["MidWest"]
```

Out[27]:
```
1995    4.5
1997    4.3
1999    3.6
2001    4.0
2003    5.7
2005    5.7
2007    4.9
2009    8.1
2011    8.7
2013    7.4
2015    5.1
Name: MidWest, dtype: float64
```

In [28]:
```
# 计算
unemp_region["West"]/100
```

Out[28]:
```
1995    0.066
1997    0.060
1999    0.052
2001    0.046
2003    0.065
2005    0.055
2007    0.045
2009    0.086
2011    0.107
2013    0.085
```

```
2015    0.061
Name: West, dtype: float64
```

```
In [29]: # 最大值
         unemp_region["West"].max()
```

```
Out[29]: 10.7
```

```
In [30]: # 差值
         unemp_region["West"] - unemp_region["MidWest"]
```

```
Out[30]: 1995    2.1
         1997    1.7
         1999    1.6
         2001    0.6
         2003    0.8
         2005   -0.2
         2007   -0.4
         2009    0.5
         2011    2.0
         2013    1.1
         2015    1.0
         dtype: float64
```

```
In [31]: # 相关性
         unemp_region.West.corr(unemp_region["MidWest"])
```

```
Out[31]: 0.9006381255384481
```

```
In [32]: unemp_region.corr()
```

Out[32]:

	NorthEast	MidWest	South	West	National
NorthEast	1.000000	0.875654	0.964415	0.967875	0.976016
MidWest	0.875654	1.000000	0.951379	0.900638	0.952389
South	0.964415	0.951379	1.000000	0.987259	0.995030
West	0.967875	0.900638	0.987259	1.000000	0.981308
National	0.976016	0.952389	0.995030	0.981308	1.000000

在计算时，请注意数据类型的问题。

In [33]:
```python
str_unemp = unemp_region.copy()
str_unemp["South"] = str_unemp["South"].astype(str)
str_unemp.dtypes
```

Out[33]:
```
NorthEast    float64
MidWest      float64
South         object
West         float64
National     float64
dtype: object
```

In [34]:
```python
str_unemp.sum()
```

Out[34]:
```
NorthEast                               63.7
MidWest                                   62
South        5.35.24.24.05.75.24.37.69.17.45.5
West                                    72.8
National                                65.7
dtype: object
```

会发现 South 列出了问题，这是由于 South 列数据为字符的缘故，因此 sum 计算体现为字符的拼接

In [35]:
```python
# 添加新列
unemp_region["UnweightedMean"] = (unemp_region["NorthEast"] +
                                  unemp_region["MidWest"] +
                                  unemp_region["South"] +
                                  unemp_region["West"])/4
```

In [36]:
```python
unemp_region.head()
```

Out[36]:

	NorthEast	MidWest	South	West	National	UnweightedMean
1995	5.9	4.5	5.3	6.6	5.6	5.575
1997	5.6	4.3	5.2	6.0	5.3	5.275
1999	4.4	3.6	4.2	5.2	4.3	4.350
2001	3.8	4.0	4.0	4.6	4.2	4.100
2003	5.8	5.7	5.7	6.5	5.8	5.925

In [37]:
```python
# 改变特定值
unemp_region.loc[1995, "UnweightedMean"] = 0.0
```

In [38]: unemp_region.head()

Out[38]:

	NorthEast	MidWest	South	West	National	UnweightedMean
1995	5.9	4.5	5.3	6.6	5.6	0.000
1997	5.6	4.3	5.2	6.0	5.3	5.275
1999	4.4	3.6	4.2	5.2	4.3	4.350
2001	3.8	4.0	4.0	4.6	4.2	4.100
2003	5.8	5.7	5.7	6.5	5.8	5.925

In [39]:
```
# 重命名列名
names = {"NorthEast": "NE",
         "MidWest": "MW",
         "South": "S",
         "West": "W"}
unemp_region.rename(columns=names)
```

Out[39]:

	NE	MW	S	W	National	UnweightedMean
1995	5.9	4.5	5.3	6.6	5.6	0.000
1997	5.6	4.3	5.2	6.0	5.3	5.275
1999	4.4	3.6	4.2	5.2	4.3	4.350
2001	3.8	4.0	4.0	4.6	4.2	4.100
2003	5.8	5.7	5.7	6.5	5.8	5.925
2005	4.9	5.7	5.2	5.5	5.3	5.325
2007	4.3	4.9	4.3	4.5	4.6	4.500
2009	7.1	8.1	7.6	8.6	7.8	7.850
2011	8.3	8.7	9.1	10.7	9.1	9.200
2013	7.9	7.4	7.4	8.5	8.0	7.800
2015	5.7	5.1	5.5	6.1	5.7	5.600

In [40]: unemp_region.head()

Out[40]:

	NorthEast	MidWest	South	West	National	UnweightedMean
1995	5.9	4.5	5.3	6.6	5.6	0.000
1997	5.6	4.3	5.2	6.0	5.3	5.275
1999	4.4	3.6	4.2	5.2	4.3	4.350
2001	3.8	4.0	4.0	4.6	4.2	4.100
2003	5.8	5.7	5.7	6.5	5.8	5.925

Pandas 在默认情况下会创建一个数据副本，以保护数据，并防止操作覆盖本应保留的信息。如果想改变原始数据，可按照如下操作 df.rename(columns=rename_dict, inplace=True)，但是并不建议这样做。

```
In [41]: names = {"NorthEast": "NE",
                  "MidWest": "MW",
                  "South": "S",
                  "West": "W"}

         unemp_shortname = unemp_region.rename(columns=names)
         unemp_shortname.head()
```

```
Out[41]:        NE   MW    S    W  National  UnweightedMean
         1995  5.9  4.5  5.3  6.6       5.6           0.000
         1997  5.6  4.3  5.2  6.0       5.3           5.275
         1999  4.4  3.6  4.2  5.2       4.3           4.350
         2001  3.8  4.0  4.0  4.6       4.2           4.100
         2003  5.8  5.7  5.7  6.5       5.8           5.925
```

1.2 2. 基本功能

首先下载样例数据

```
In [43]: url = "https://storage.googleapis.com/qeds/data/state_unemployment.csv"
         unemp_raw = pd.read_csv(url, parse_dates=["Date"])
```

这里添加 parse_dates=["Date"] 的原因是，read_csv 只规定了基础的数据类型，但我们需要把 Date 列的数据读取为时间类型。

```
In [44]: unemp_raw.head()
```

```
Out[44]:         Date       state  LaborForce  UnemploymentRate
         0 2000-01-01     Alabama   2142945.0               4.7
         1 2000-01-01      Alaska    319059.0               6.3
         2 2000-01-01     Arizona   2499980.0               4.1
         3 2000-01-01    Arkansas   1264619.0               4.4
         4 2000-01-01  California  16680246.0               5.0
```

1.2.1 （1）数据透视表

为了方便观察随着时间推移，不同州的失业率变化，我们需要一个 “透视表” 视图

```
In [46]: unemp_all = (
             unemp_raw
```

```python
    .reset_index()
    .pivot_table(index="Date", columns="state", values="UnemploymentRate")
)
unemp_all.head()
```

Out[46]:

```
state       Alabama  Alaska  Arizona  Arkansas  California  Colorado  \
Date
2000-01-01      4.7     6.3      4.1       4.4         5.0       2.8
2000-02-01      4.7     6.3      4.1       4.3         5.0       2.8
2000-03-01      4.6     6.3      4.0       4.3         5.0       2.7
2000-04-01      4.6     6.3      4.0       4.3         5.1       2.7
2000-05-01      4.5     6.3      4.0       4.2         5.1       2.7

state       Connecticut  Delaware  Florida  Georgia  ...    South Dakota  \
Date                                                 ...
2000-01-01          2.8       3.5      3.7      3.7  ...             2.4
2000-02-01          2.7       3.6      3.7      3.6  ...             2.4
2000-03-01          2.6       3.6      3.7      3.6  ...             2.4
2000-04-01          2.5       3.7      3.7      3.7  ...             2.4
2000-05-01          2.4       3.7      3.7      3.7  ...             2.4

state       Tennessee  Texas  Utah  Vermont  Virginia  Washington  \
Date
2000-01-01        3.7    4.6   3.1      2.7       2.6         4.9
2000-02-01        3.7    4.6   3.1      2.6       2.5         4.9
2000-03-01        3.8    4.5   3.1      2.6       2.4         5.0
2000-04-01        3.8    4.4   3.1      2.7       2.4         5.0
2000-05-01        3.9    4.3   3.2      2.7       2.3         5.1

state       West Virginia  Wisconsin  Wyoming
Date
2000-01-01            5.8        3.2      4.1
2000-02-01            5.6        3.2      3.9
2000-03-01            5.5        3.3      3.9
2000-04-01            5.4        3.4      3.8
2000-05-01            5.4        3.5      3.8
```

[5 rows x 50 columns]

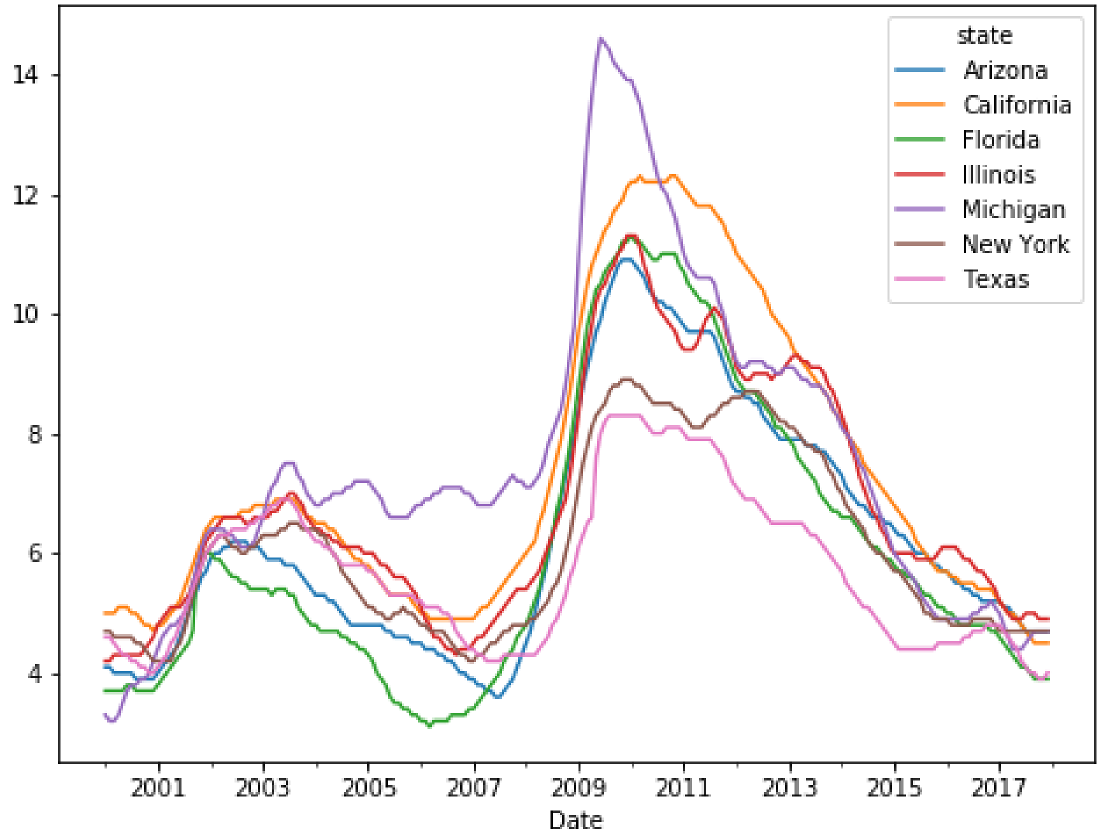
In [47]:
```
# 查看部分数据
states = [
    "Arizona", "California", "Florida", "Illinois",
    "Michigan", "New York", "Texas"
]
unemp = unemp_all[states]
unemp.head()
```

Out[47]:

state	Arizona	California	Florida	Illinois	Michigan	New York	Texas
Date							
2000-01-01	4.1	5.0	3.7	4.2	3.3	4.7	4.6
2000-02-01	4.1	5.0	3.7	4.2	3.2	4.7	4.6
2000-03-01	4.0	5.0	3.7	4.3	3.2	4.6	4.5
2000-04-01	4.0	5.1	3.7	4.3	3.3	4.6	4.4
2000-05-01	4.0	5.1	3.7	4.3	3.5	4.6	4.3

In [49]:
```
# 趋势
unemp.plot(figsize=(8, 6))
```

Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x1ea1ae896a0>



1.2.2 （2）日期数据

```
In [51]: unemp.index
```

```
Out[51]: DatetimeIndex(['2000-01-01', '2000-02-01', '2000-03-01', '2000-04-01',
                        '2000-05-01', '2000-06-01', '2000-07-01', '2000-08-01',
                        '2000-09-01', '2000-10-01',
                        ...
                        '2017-03-01', '2017-04-01', '2017-05-01', '2017-06-01',
                        '2017-07-01', '2017-08-01', '2017-09-01', '2017-10-01',
                        '2017-11-01', '2017-12-01'],
                       dtype='datetime64[ns]', name='Date', length=216, freq=None)
```

```
In [52]: # 特定值
         unemp.loc["01/01/2000", :]
```

```
Out[52]: state
         Arizona       4.1
         California    5.0
         Florida       3.7
         Illinois      4.2
         Michigan      3.3
         New York      4.7
         Texas         4.6
         Name: 2000-01-01 00:00:00, dtype: float64
```

```
In [53]: unemp.loc["01/01/2000":"06/01/2000", :]
```

```
Out[53]: state        Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01       4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-02-01       4.1         5.0      3.7       4.2       3.2       4.7    4.6
         2000-03-01       4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-04-01       4.0         5.1      3.7       4.3       3.3       4.6    4.4
         2000-05-01       4.0         5.1      3.7       4.3       3.5       4.6    4.3
         2000-06-01       4.0         5.1      3.8       4.3       3.7       4.6    4.3
```

1.2.3 （3）聚合操作

简单地说，聚合就是将多个值组合成单个值的操作，比如均值，方差，标准差，最值等。

```
In [55]: # 均值
         unemp.mean()
```

```
Out[55]: state
         Arizona       6.301389
         California    7.299074
         Florida       6.048611
         Illinois      6.822685
         Michigan      7.492593
         New York      6.102315
         Texas         5.695370
         dtype: float64
```

可以看到，聚合操作默认是按照列进行的，但是通过 axis 也可实现按行操作。

In [56]:
```
unemp.var(axis=1).head()
```

Out[56]:
```
Date
2000-01-01    0.352381
2000-02-01    0.384762
2000-03-01    0.364762
2000-04-01    0.353333
2000-05-01    0.294762
dtype: float64
```

In [57]:
```
# 编写聚合
# 我们将根据各州的平均失业水平是高于还是低于 6.5，将各州分为“低失业率”或“高失业率”。

def high_or_low(s):
    if s.mean() < 6.5:
        out = "Low"
    else:
        out = "High"

    return out
```

In [58]:
```
unemp.agg(high_or_low)
```

Out[58]:
```
state
Arizona        Low
California    High
Florida        Low
Illinois      High
Michigan      High
New York       Low
Texas          Low
dtype: object
```

In [59]:
```
unemp.agg(high_or_low, axis=1).head()
```

Out[59]:
```
Date
2000-01-01    Low
2000-02-01    Low
2000-03-01    Low
```

```
2000-04-01     Low
2000-05-01     Low
dtype: object
```

In [60]:
```
# 多个函数
unemp.agg([min, max, high_or_low])
```

Out[60]:
```
             Arizona California Florida Illinois Michigan New York Texas
min              3.6        4.5     3.1      4.2      3.2      4.2   3.9
max             10.9       12.3    11.3     11.3     14.6      8.9   8.3
high_or_low      Low       High     Low     High     High      Low   Low
```

1.2.4 （4）内置变换

In [63]:
```
unemp.head()
```

Out[63]:
```
state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
Date
2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
2000-02-01      4.1         5.0      3.7       4.2       3.2       4.7    4.6
2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

In [64]:
```
# 变化率
unemp.pct_change().head()
```

Out[64]:
```
state       Arizona  California  Florida  Illinois  Michigan  New York  \
Date
2000-01-01      NaN         NaN      NaN       NaN       NaN       NaN
2000-02-01  0.00000        0.00      0.0   0.00000 -0.030303  0.000000
2000-03-01 -0.02439        0.00      0.0   0.02381  0.000000 -0.021277
2000-04-01  0.00000        0.02      0.0   0.00000  0.031250  0.000000
2000-05-01  0.00000        0.00      0.0   0.00000  0.060606  0.000000

state          Texas
Date
2000-01-01       NaN
2000-02-01  0.000000
```

```
2000-03-01 -0.021739
2000-04-01 -0.022222
2000-05-01 -0.022727
```

In [65]:
```
# 差值
unemp.diff().head()
```

Out[65]:
```
state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
Date
2000-01-01      NaN         NaN      NaN       NaN       NaN       NaN    NaN
2000-02-01      0.0         0.0      0.0       0.0      -0.1       0.0    0.0
2000-03-01     -0.1         0.0      0.0       0.1       0.0      -0.1   -0.1
2000-04-01      0.0         0.1      0.0       0.0       0.1       0.0   -0.1
2000-05-01      0.0         0.0      0.0       0.0       0.2       0.0   -0.1
```

In [66]:
```
# 自编变换

def standardize_data(x):
    mu = x.mean()
    std = x.std()

    return (x - mu)/std
```

In [68]:
```
std_unemp = unemp.apply(standardize_data)
std_unemp.head()
```

Out[68]:
```
state        Arizona  California   Florida  Illinois  Michigan  New York  \
Date
2000-01-01 -1.076861   -0.935545 -0.976846 -1.337203 -1.605740 -0.925962
2000-02-01 -1.076861   -0.935545 -0.976846 -1.337203 -1.644039 -0.925962
2000-03-01 -1.125778   -0.935545 -0.976846 -1.286217 -1.644039 -0.991993
2000-04-01 -1.125778   -0.894853 -0.976846 -1.286217 -1.605740 -0.991993
2000-05-01 -1.125778   -0.894853 -0.976846 -1.286217 -1.529141 -0.991993

state          Texas
Date
2000-01-01 -0.849345
2000-02-01 -0.849345
2000-03-01 -0.926885
```

```
2000-04-01 -1.004424
2000-05-01 -1.081964
```

此处使用 apply 和 agg 均可，但是若操作涉及排序等操作时则需要用到 apply。相比较 agg 来说，apply 更一般。

In [69]:
```python
# 绝对值
abs_std_unemp = std_unemp.abs()

abs_std_unemp.head()
```

Out[69]:
```
state         Arizona  California   Florida  Illinois  Michigan  New York  \
Date
2000-01-01  1.076861    0.935545  0.976846  1.337203  1.605740  0.925962
2000-02-01  1.076861    0.935545  0.976846  1.337203  1.644039  0.925962
2000-03-01  1.125778    0.935545  0.976846  1.286217  1.644039  0.991993
2000-04-01  1.125778    0.894853  0.976846  1.286217  1.605740  0.991993
2000-05-01  1.125778    0.894853  0.976846  1.286217  1.529141  0.991993

state          Texas
Date
2000-01-01  0.849345
2000-02-01  0.849345
2000-03-01  0.926885
2000-04-01  1.004424
2000-05-01  1.081964
```

In [70]:
```python
def idxmax(x):
    return x.idxmax()
abs_std_unemp.agg(idxmax)
```

Out[70]:
```
state
Arizona       2009-11-01
California    2010-03-01
Florida       2010-01-01
Illinois      2009-12-01
Michigan      2009-06-01
New York      2009-11-01
Texas         2009-08-01
dtype: datetime64[ns]
```

1.2.5 （5）布尔选择

前面我们介绍了通过索引，列名等方式从数据集中找到特定的数据。但是有时，我们需要根据数据自身的特征来选择数据，比如：获取特定产品或客户 ID 的数据、分析与经济衰退相对应的数据等。此时可以使用布尔值来实现目的。

```
In [71]: unemp_small = unemp.head()
         unemp_small
```

```
Out[71]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-02-01      4.1         5.0      3.7       4.2       3.2       4.7    4.6
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
         2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

```
In [72]: unemp_small.loc[[True, True, True, False, False]]
```

```
Out[72]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-02-01      4.1         5.0      3.7       4.2       3.2       4.7    4.6
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
```

```
In [73]: unemp_small.loc[[True, False, True, False, True], :]
```

```
Out[73]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

```
In [74]: unemp_small.loc[[True, True, True, False, False], [True, False, False, False, False,
```

```
Out[74]: state       Arizona  New York  Texas
         Date
         2000-01-01      4.1       4.7    4.6
         2000-02-01      4.1       4.7    4.6
         2000-03-01      4.0       4.6    4.5
```

```
In [75]: # 自创布尔值
         unemp_small["Texas"] < 4.5
```

```
Out[75]: Date
         2000-01-01    False
         2000-02-01    False
         2000-03-01    False
         2000-04-01     True
         2000-05-01     True
         Freq: MS, Name: Texas, dtype: bool
```

```
In [76]: unemp_small.loc[unemp_small["Texas"] < 4.5]
```

```
Out[76]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
         2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

```
In [77]: # 自创布尔值
         unemp_small["New York"] > unemp_small["Texas"]
```

```
Out[77]: Date
         2000-01-01    True
         2000-02-01    True
         2000-03-01    True
         2000-04-01    True
         2000-05-01    True
         Freq: MS, dtype: bool
```

```
In [78]: big_NY = unemp_small["New York"] > unemp_small["Texas"]
         unemp_small.loc[big_NY]
```

```
Out[78]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-02-01      4.1         5.0      3.7       4.2       3.2       4.7    4.6
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
         2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

```
In [79]: # 多重条件
         small_NYTX = (unemp_small["Texas"] < 4.7) & (unemp_small["New York"] < 4.7)
         small_NYTX
```

```
Out[79]: Date
         2000-01-01    False
         2000-02-01    False
         2000-03-01     True
         2000-04-01     True
         2000-05-01     True
         Freq: MS, dtype: bool
```

```
In [80]: unemp_small[small_NYTX]
```

```
Out[80]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
         2000-05-01      4.0         5.1      3.7       4.3       3.5       4.6    4.3
```

```
In [81]: unemp_small["Michigan"].isin([3.3, 3.2])
```

```
Out[81]: Date
         2000-01-01     True
         2000-02-01     True
         2000-03-01     True
         2000-04-01     True
         2000-05-01    False
         Freq: MS, Name: Michigan, dtype: bool
```

```
In [82]: unemp_small.loc[unemp_small["Michigan"].isin([3.3, 3.2])]
```

```
Out[82]: state       Arizona  California  Florida  Illinois  Michigan  New York  Texas
         Date
         2000-01-01      4.1         5.0      3.7       4.2       3.3       4.7    4.6
         2000-02-01      4.1         5.0      3.7       4.2       3.2       4.7    4.6
         2000-03-01      4.0         5.0      3.7       4.3       3.2       4.6    4.5
         2000-04-01      4.0         5.1      3.7       4.3       3.3       4.6    4.4
```

1.3 3. 索引

1.3.1 （1）自动对准

In [2]:
```
url = "https://storage.googleapis.com/qeds/data/wdi_data.csv"
df = pd.read_csv(url)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 72 entries, 0 to 71
Data columns (total 7 columns):
country        72 non-null object
year           72 non-null int64
GovExpend      72 non-null float64
Consumption    72 non-null float64
Exports        72 non-null float64
Imports        72 non-null float64
GDP            72 non-null float64
dtypes: float64(5), int64(1), object(1)
memory usage: 4.0+ KB
```

In [3]:
```
df.head()
```

Out[3]:

	country	year	GovExpend	Consumption	Exports	Imports	GDP
0	Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
1	Canada	2016	0.364899	1.058426	0.576394	0.575775	1.814016
2	Canada	2015	0.358303	1.035208	0.568859	0.575793	1.794270
3	Canada	2014	0.353485	1.011988	0.550323	0.572344	1.782252
4	Canada	2013	0.351541	0.986400	0.518040	0.558636	1.732714

In [5]:
```
df_small = df.head(5)
df_small
```

Out[5]:

	country	year	GovExpend	Consumption	Exports	Imports	GDP
0	Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
1	Canada	2016	0.364899	1.058426	0.576394	0.575775	1.814016
2	Canada	2015	0.358303	1.035208	0.568859	0.575793	1.794270
3	Canada	2014	0.353485	1.011988	0.550323	0.572344	1.782252
4	Canada	2013	0.351541	0.986400	0.518040	0.558636	1.732714

```
In [6]: df_tiny = df.iloc[[0, 3, 2, 4], :]
        df_tiny
```

Out[6]:

	country	year	GovExpend	Consumption	Exports	Imports	GDP
0	Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
3	Canada	2014	0.353485	1.011988	0.550323	0.572344	1.782252
2	Canada	2015	0.358303	1.035208	0.568859	0.575793	1.794270
4	Canada	2013	0.351541	0.986400	0.518040	0.558636	1.732714

```
In [7]: im_ex = df_small[["Imports", "Exports"]]
        im_ex_copy = im_ex.copy()
        im_ex_copy
```

Out[7]:

	Imports	Exports
0	0.600031	0.582831
1	0.575775	0.576394
2	0.575793	0.568859
3	0.572344	0.550323
4	0.558636	0.518040

```
In [8]: im_ex + im_ex_copy
```

Out[8]:

	Imports	Exports
0	1.200063	1.165661
1	1.151550	1.152787
2	1.151585	1.137718
3	1.144688	1.100646
4	1.117272	1.036081

```
In [9]: df_tiny
```

Out[9]:

	country	year	GovExpend	Consumption	Exports	Imports	GDP
0	Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
3	Canada	2014	0.353485	1.011988	0.550323	0.572344	1.782252
2	Canada	2015	0.358303	1.035208	0.568859	0.575793	1.794270
4	Canada	2013	0.351541	0.986400	0.518040	0.558636	1.732714

```
In [10]: im_ex_tiny = df_tiny + im_ex
         im_ex_tiny
```

Out[10]:

	Consumption	Exports	GDP	GovExpend	Imports	country	year
0	NaN	1.165661	NaN	NaN	1.200063	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	1.137718	NaN	NaN	1.151585	NaN	NaN
3	NaN	1.100646	NaN	NaN	1.144688	NaN	NaN
4	NaN	1.036081	NaN	NaN	1.117272	NaN	NaN

以上可以看出即使两个数据集的行与列不同，pandas 可以帮助我们自动对准并完成运算。

1.3.2 （2）设置索引

In [13]:

```
# 设定年份为索引
df_year = df.set_index(["year"])
df_year.head()
```

Out[13]:

	country	GovExpend	Consumption	Exports	Imports	GDP
year						
2017	Canada	0.372665	1.095475	0.582831	0.600031	1.868164
2016	Canada	0.364899	1.058426	0.576394	0.575775	1.814016
2015	Canada	0.358303	1.035208	0.568859	0.575793	1.794270
2014	Canada	0.353485	1.011988	0.550323	0.572344	1.782252
2013	Canada	0.351541	0.986400	0.518040	0.558636	1.732714

In [14]:

```
# 特定年份
df_year.loc[2010]
```

Out[14]:

	country	GovExpend	Consumption	Exports	Imports	GDP
year						
2010	Canada	0.347332	0.921952	0.469949	0.500341	1.613543
2010	Germany	0.653386	1.915481	1.443735	1.266126	3.417095
2010	United Kingdom	0.521146	1.598563	0.690824	0.745065	2.452900
2010	United States	2.510143	10.185836	1.846280	2.360183	14.992053

In [15]:

```
#2008 年和 2009 年数据差值
df_year.loc[2009].mean() - df_year.loc[2008].mean()
```

Out[15]:

```
GovExpend       0.033317
Consumption    -0.042998
Exports        -0.121425
Imports        -0.140042
```

```
GDP           -0.182610
dtype: float64
```

In [16]:
```
#2010 年美国的 GDP
df_year.loc[df_year["country"] == "United States", "GDP"].loc[2010]
```

Out[16]:
```
14.992052727
```

In [18]:
```
#2010 年德国和英国的 GDP
df_year.loc[df_year["country"].isin(["United Kingdom", "Germany"]), "GDP"].loc[2010]
```

Out[18]:
```
year
2010    3.417095
2010    2.452900
Name: GDP, dtype: float64
```

这里出现了一个问题，我们如何知道两个 "2010 年" 数据分别对应的是哪个国家？因为 isin 里 UK 是首位，所以 GDP 首位就是指 UK 的？

In [19]:
```
df_year.loc[2010]
```

Out[19]:

	country	GovExpend	Consumption	Exports	Imports	GDP
year						
2010	Canada	0.347332	0.921952	0.469949	0.500341	1.613543
2010	Germany	0.653386	1.915481	1.443735	1.266126	3.417095
2010	United Kingdom	0.521146	1.598563	0.690824	0.745065	2.452900
2010	United States	2.510143	10.185836	1.846280	2.360183	14.992053

显然，我们上面的猜想是错误的。这反映了一个问题，仅仅将数据按年份对准是不够的。

1.3.3 （3）分层索引

In [21]:
```
wdi = df.set_index(["country", "year"])
wdi.head(20)
```

Out[21]:

		GovExpend	Consumption	Exports	Imports	GDP
country	year					
Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
	2016	0.364899	1.058426	0.576394	0.575775	1.814016
	2015	0.358303	1.035208	0.568859	0.575793	1.794270
	2014	0.353485	1.011988	0.550323	0.572344	1.782252

	2013	0.351541	0.986400	0.518040	0.558636	1.732714
	2012	0.354342	0.961226	0.505969	0.547756	1.693428
	2011	0.351887	0.943145	0.492349	0.528227	1.664240
	2010	0.347332	0.921952	0.469949	0.500341	1.613543
	2009	0.339686	0.890078	0.440692	0.439796	1.565291
	2008	0.330766	0.889602	0.506350	0.502281	1.612862
	2007	0.318777	0.864012	0.530453	0.498002	1.596876
	2006	0.311382	0.827643	0.524461	0.470931	1.564608
	2005	0.303043	0.794390	0.519950	0.447222	1.524608
	2004	0.299854	0.764357	0.508657	0.416754	1.477317
	2003	0.294335	0.741796	0.481993	0.384199	1.433089
	2002	0.286094	0.721974	0.490465	0.368615	1.407725
	2001	0.279767	0.694230	0.484696	0.362023	1.366590
	2000	0.270553	0.677713	0.499526	0.380823	1.342805
Germany	2017	0.745579	2.112009	1.930563	1.666348	3.883870
	2016	0.734014	2.075615	1.844949	1.589495	3.801859

In [22]:
```
#2010 年美国 GDP
wdi.loc[("United States", 2010), "GDP"]

#df_year.loc[df_year["country"] == "United States", "GDP"].loc[2010]
```

Out[22]: 14.992052727

In [23]:
```
#2010 年美国和德国 GDP
wdi.loc[(["United Kingdom", "Germany"], 2010), "GDP"]
```

Out[23]:
```
country         year
Germany         2010    3.417095
United Kingdom  2010    2.452900
Name: GDP, dtype: float64
```

In [24]:
```
# 美国和加拿大数据
wdi.loc[["United States", "Canada"]]
```

Out[24]:

		GovExpend	Consumption	Exports	Imports	GDP
country	year					
Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
	2016	0.364899	1.058426	0.576394	0.575775	1.814016

	2015	0.358303	1.035208	0.568859	0.575793	1.794270
	2014	0.353485	1.011988	0.550323	0.572344	1.782252
	2013	0.351541	0.986400	0.518040	0.558636	1.732714
	2012	0.354342	0.961226	0.505969	0.547756	1.693428
	2011	0.351887	0.943145	0.492349	0.528227	1.664240
	2010	0.347332	0.921952	0.469949	0.500341	1.613543
	2009	0.339686	0.890078	0.440692	0.439796	1.565291
	2008	0.330766	0.889602	0.506350	0.502281	1.612862
	2007	0.318777	0.864012	0.530453	0.498002	1.596876
	2006	0.311382	0.827643	0.524461	0.470931	1.564608
	2005	0.303043	0.794390	0.519950	0.447222	1.524608
	2004	0.299854	0.764357	0.508657	0.416754	1.477317
	2003	0.294335	0.741796	0.481993	0.384199	1.433089
	2002	0.286094	0.721974	0.490465	0.368615	1.407725
	2001	0.279767	0.694230	0.484696	0.362023	1.366590
	2000	0.270553	0.677713	0.499526	0.380823	1.342805
United States	2017	2.405743	12.019266	2.287071	3.069954	17.348627
	2016	2.407981	11.722133	2.219937	2.936004	16.972348
	2015	2.373130	11.409800	2.222228	2.881337	16.710459
	2014	2.334071	11.000619	2.209555	2.732228	16.242526
	2013	2.353381	10.687214	2.118639	2.600198	15.853796
	2012	2.398873	10.534042	2.045509	2.560677	15.567038
	2011	2.434378	10.378060	1.978083	2.493194	15.224555
	2010	2.510143	10.185836	1.846280	2.360183	14.992053
	2009	2.507390	10.010687	1.646432	2.086299	14.617299
	2008	2.407771	10.137847	1.797347	2.400349	14.997756
	2007	2.351987	10.159387	1.701096	2.455016	15.018268
	2006	2.314957	9.938503	1.564920	2.395189	14.741688
	2005	2.287022	9.643098	1.431205	2.246246	14.332500
	2004	2.267999	9.311431	1.335978	2.108585	13.846058
	2003	2.233519	8.974708	1.218199	1.892825	13.339312
	2002	2.193188	8.698306	1.192180	1.804105	12.968263
	2001	2.112038	8.480461	1.213253	1.740797	12.746262
	2000	2.040500	8.272097	1.287739	1.790995	12.620268

In [27]:
```
#2005、2007、2009 年所有国家数据
wdi.loc[pd.IndexSlice[:, [2005, 2007, 2009]], :]
```

```
#wdi.loc[(["United Kingdom", "Germany"], 2010), "GDP"]
```

Out[27]:

```
                        GovExpend  Consumption   Exports   Imports        GDP
country        year
Canada         2009      0.339686     0.890078  0.440692  0.439796   1.565291
               2007      0.318777     0.864012  0.530453  0.498002   1.596876
               2005      0.303043     0.794390  0.519950  0.447222   1.524608
Germany        2009      0.645023     1.908393  1.260525  1.121914   3.283144
               2007      0.605624     1.894219  1.442436  1.213835   3.441356
               2005      0.591184     1.866253  1.175200  1.028094   3.213777
United Kingdom 2009      0.519716     1.587152  0.653830  0.689011   2.411632
               2007      0.504549     1.644789  0.710200  0.767699   2.527327
               2005      0.490806     1.578914  0.640088  0.715951   2.403352
United States  2009      2.507390    10.010687  1.646432  2.086299  14.617299
               2007      2.351987    10.159387  1.701096  2.455016  15.018268
               2005      2.287022     9.643098  1.431205  2.246246  14.332500
```

In [26]:

```
wdi.loc[(:,[2005, 2007, 2009]), :]
```

```
  File "<ipython-input-26-bde458006c3a>", line 1
    wdi.loc[(:,[2005, 2007, 2009]), :]
             ^
SyntaxError: invalid syntax
```

此时，loc 不起作用，需要借助 pd.IndexSlice

In [28]:

```
# 多索引列
wdiT = wdi.T
wdiT
```

Out[28]:

```
country        Canada                                                         \
year             2017      2016      2015      2014      2013      2012
GovExpend    0.372665  0.364899  0.358303  0.353485  0.351541  0.354342
Consumption  1.095475  1.058426  1.035208  1.011988  0.986400  0.961226
Exports      0.582831  0.576394  0.568859  0.550323  0.518040  0.505969
```

```
Imports      0.600031  0.575775  0.575793  0.572344  0.558636  0.547756
GDP          1.868164  1.814016  1.794270  1.782252  1.732714  1.693428

country                                                ...  United States  \
year             2011      2010      2009      2008    ...           2009
GovExpend    0.351887  0.347332  0.339686  0.330766    ...       2.507390
Consumption  0.943145  0.921952  0.890078  0.889602    ...      10.010687
Exports      0.492349  0.469949  0.440692  0.506350    ...       1.646432
Imports      0.528227  0.500341  0.439796  0.502281    ...       2.086299
GDP          1.664240  1.613543  1.565291  1.612862    ...      14.617299

country                                                                   \
year              2008       2007       2006       2005       2004       2003
GovExpend     2.407771   2.351987   2.314957   2.287022   2.267999   2.233519
Consumption  10.137847  10.159387   9.938503   9.643098   9.311431   8.974708
Exports       1.797347   1.701096   1.564920   1.431205   1.335978   1.218199
Imports       2.400349   2.455016   2.395189   2.246246   2.108585   1.892825
GDP          14.997756  15.018268  14.741688  14.332500  13.846058  13.339312

country
year              2002       2001       2000
GovExpend     2.193188   2.112038   2.040500
Consumption   8.698306   8.480461   8.272097
Exports       1.192180   1.213253   1.287739
Imports       1.804105   1.740797   1.790995
GDP          12.968263  12.746262  12.620268

[5 rows x 72 columns]
```

In [30]:
```python
# 美国数据
wdiT.loc[:, "United States"]
```

Out[30]:
```
year              2017       2016       2015       2014       2013       2012  \
GovExpend     2.405743   2.407981   2.373130   2.334071   2.353381   2.398873
Consumption  12.019266  11.722133  11.409800  11.000619  10.687214  10.534042
Exports       2.287071   2.219937   2.222228   2.209555   2.118639   2.045509
Imports       3.069954   2.936004   2.881337   2.732228   2.600198   2.560677
GDP          17.348627  16.972348  16.710459  16.242526  15.853796  15.567038
```

```
year                2011       2010       2009       2008       2007       2006  \
GovExpend       2.434378   2.510143   2.507390   2.407771   2.351987   2.314957
Consumption    10.378060  10.185836  10.010687  10.137847  10.159387   9.938503
Exports         1.978083   1.846280   1.646432   1.797347   1.701096   1.564920
Imports         2.493194   2.360183   2.086299   2.400349   2.455016   2.395189
GDP            15.224555  14.992053  14.617299  14.997756  15.018268  14.741688

year                2005       2004       2003       2002       2001       2000
GovExpend       2.287022   2.267999   2.233519   2.193188   2.112038   2.040500
Consumption     9.643098   9.311431   8.974708   8.698306   8.480461   8.272097
Exports         1.431205   1.335978   1.218199   1.192180   1.213253   1.287739
Imports         2.246246   2.108585   1.892825   1.804105   1.740797   1.790995
GDP            14.332500  13.846058  13.339312  12.968263  12.746262  12.620268
```

In [32]:

```python
# 美国和加拿大数据
wdiT.loc[:, ["United States", "Canada"]]
```

Out[32]:

```
country          Canada                                                      \
year               2017      2016      2015      2014      2013      2012
GovExpend      0.372665  0.364899  0.358303  0.353485  0.351541  0.354342
Consumption    1.095475  1.058426  1.035208  1.011988  0.986400  0.961226
Exports        0.582831  0.576394  0.568859  0.550323  0.518040  0.505969
Imports        0.600031  0.575775  0.575793  0.572344  0.558636  0.547756
GDP            1.868164  1.814016  1.794270  1.782252  1.732714  1.693428

country                                                ...  United States  \
year               2011      2010      2009      2008  ...           2009
GovExpend      0.351887  0.347332  0.339686  0.330766  ...       2.507390
Consumption    0.943145  0.921952  0.890078  0.889602  ...      10.010687
Exports        0.492349  0.469949  0.440692  0.506350  ...       1.646432
Imports        0.528227  0.500341  0.439796  0.502281  ...       2.086299
GDP            1.664240  1.613543  1.565291  1.612862  ...      14.617299

country                                                                      \
year                2008       2007       2006       2005       2004       2003
GovExpend       2.407771   2.351987   2.314957   2.287022   2.267999   2.233519
Consumption    10.137847  10.159387   9.938503   9.643098   9.311431   8.974708
```

```
Exports        1.797347   1.701096   1.564920   1.431205   1.335978   1.218199
Imports        2.400349   2.455016   2.395189   2.246246   2.108585   1.892825
GDP           14.997756  15.018268  14.741688  14.332500  13.846058  13.339312

country
year                2002       2001       2000
GovExpend       2.193188   2.112038   2.040500
Consumption     8.698306   8.480461   8.272097
Exports         1.192180   1.213253   1.287739
Imports         1.804105   1.740797   1.790995
GDP            12.968263  12.746262  12.620268

[5 rows x 36 columns]
```

In [33]:
```python
#2010 年美国和加拿大数据
wdiT.loc[:, (["United States", "Canada"], 2010)]
```

Out[33]:
```
country          Canada United States
year               2010          2010
GovExpend      0.347332      2.510143
Consumption    0.921952     10.185836
Exports        0.469949      1.846280
Imports        0.500341      2.360183
GDP            1.613543     14.992053
```

1.3.4 （4）重设索引

df.reset_index 方法将索引的一个或多个级别作为普通列移回到 DataFrame

In [34]:
```python
wdi.reset_index()
```

Out[34]:
```
            country  year  GovExpend  Consumption   Exports   Imports  \
0            Canada  2017   0.372665     1.095475  0.582831  0.600031
1            Canada  2016   0.364899     1.058426  0.576394  0.575775
2            Canada  2015   0.358303     1.035208  0.568859  0.575793
3            Canada  2014   0.353485     1.011988  0.550323  0.572344
4            Canada  2013   0.351541     0.986400  0.518040  0.558636
5            Canada  2012   0.354342     0.961226  0.505969  0.547756
6            Canada  2011   0.351887     0.943145  0.492349  0.528227
```

7	Canada	2010	0.347332	0.921952	0.469949	0.500341
8	Canada	2009	0.339686	0.890078	0.440692	0.439796
9	Canada	2008	0.330766	0.889602	0.506350	0.502281
10	Canada	2007	0.318777	0.864012	0.530453	0.498002
11	Canada	2006	0.311382	0.827643	0.524461	0.470931
12	Canada	2005	0.303043	0.794390	0.519950	0.447222
13	Canada	2004	0.299854	0.764357	0.508657	0.416754
14	Canada	2003	0.294335	0.741796	0.481993	0.384199
15	Canada	2002	0.286094	0.721974	0.490465	0.368615
16	Canada	2001	0.279767	0.694230	0.484696	0.362023
17	Canada	2000	0.270553	0.677713	0.499526	0.380823
18	Germany	2017	0.745579	2.112009	1.930563	1.666348
19	Germany	2016	0.734014	2.075615	1.844949	1.589495
20	Germany	2015	0.706115	2.033666	1.803081	1.527074
21	Germany	2014	0.685990	1.999953	1.712270	1.445409
22	Germany	2013	0.675471	1.979458	1.635030	1.394385
23	Germany	2012	0.666454	1.967390	1.607455	1.354122
24	Germany	2011	0.659528	1.941340	1.563277	1.355008
25	Germany	2010	0.653386	1.915481	1.443735	1.266126
26	Germany	2009	0.645023	1.908393	1.260525	1.121914
27	Germany	2008	0.626140	1.905520	1.470300	1.241057
28	Germany	2007	0.605624	1.894219	1.442436	1.213835
29	Germany	2006	0.596868	1.894219	1.319574	1.142552
..	...	...	...	...	...	...
42	United Kingdom	2011	0.521716	1.588172	0.735331	0.750540
43	United Kingdom	2010	0.521146	1.598563	0.690824	0.745065
44	United Kingdom	2009	0.519716	1.587152	0.653830	0.689011
45	United Kingdom	2008	0.513870	1.635333	0.713184	0.753502
46	United Kingdom	2007	0.504549	1.644789	0.710200	0.767699
47	United Kingdom	2006	0.499312	1.604404	0.717506	0.786134
48	United Kingdom	2005	0.490806	1.578914	0.640088	0.715951
49	United Kingdom	2004	0.471828	1.531808	0.593046	0.667995
50	United Kingdom	2003	0.452743	1.484092	0.562653	0.625696
51	United Kingdom	2002	0.434954	1.433861	0.546092	0.606795
52	United Kingdom	2001	0.418387	1.380779	0.533999	0.574277
53	United Kingdom	2000	0.401274	1.332093	0.523797	0.548044
54	United States	2017	2.405743	12.019266	2.287071	3.069954

55	United States	2016	2.407981	11.722133	2.219937	2.936004
56	United States	2015	2.373130	11.409800	2.222228	2.881337
57	United States	2014	2.334071	11.000619	2.209555	2.732228
58	United States	2013	2.353381	10.687214	2.118639	2.600198
59	United States	2012	2.398873	10.534042	2.045509	2.560677
60	United States	2011	2.434378	10.378060	1.978083	2.493194
61	United States	2010	2.510143	10.185836	1.846280	2.360183
62	United States	2009	2.507390	10.010687	1.646432	2.086299
63	United States	2008	2.407771	10.137847	1.797347	2.400349
64	United States	2007	2.351987	10.159387	1.701096	2.455016
65	United States	2006	2.314957	9.938503	1.564920	2.395189
66	United States	2005	2.287022	9.643098	1.431205	2.246246
67	United States	2004	2.267999	9.311431	1.335978	2.108585
68	United States	2003	2.233519	8.974708	1.218199	1.892825
69	United States	2002	2.193188	8.698306	1.192180	1.804105
70	United States	2001	2.112038	8.480461	1.213253	1.740797
71	United States	2000	2.040500	8.272097	1.287739	1.790995

	GDP
0	1.868164
1	1.814016
2	1.794270
3	1.782252
4	1.732714
5	1.693428
6	1.664240
7	1.613543
8	1.565291
9	1.612862
10	1.596876
11	1.564608
12	1.524608
13	1.477317
14	1.433089
15	1.407725
16	1.366590
17	1.342805

18	3.883870
19	3.801859
20	3.718482
21	3.654924
22	3.577015
23	3.559587
24	3.542160
25	3.417095
26	3.283144
27	3.478602
28	3.441356
29	3.332692
..	...
42	2.493244
43	2.452900
44	2.411632
45	2.518585
46	2.527327
47	2.464591
48	2.403352
49	2.329987
50	2.276538
51	2.202971
52	2.149246
53	2.089877
54	17.348627
55	16.972348
56	16.710459
57	16.242526
58	15.853796
59	15.567038
60	15.224555
61	14.992053
62	14.617299
63	14.997756
64	15.018268
65	14.741688

```
66   14.332500
67   13.846058
68   13.339312
69   12.968263
70   12.746262
71   12.620268

[72 rows x 7 columns]
```

1.4 4. 数据存储

In [3]:
```python
import pandas as pd
import numpy as np
```

In [4]:
```python
# 生成一份数据集
np.random.seed(42)
df1 = pd.DataFrame(
    np.random.randint(0, 100, size=(10, 4)),
    columns=["a", "b", "c", "d"]
)

wanted_mb = 10
nrow = 100000
ncol = int(((wanted_mb * 1024**2) / 8) / nrow)


df2 = pd.DataFrame(
    np.random.rand(nrow, ncol),
    columns=["x{}".format(i) for i in range(ncol)]
)

print("df2.shape = ", df2.shape)
print("df2 is approximately {} MB".format(df2.memory_usage().sum() / (1024**2)))
```

```
df2.shape =  (100000, 13)
df2 is approximately 9.918289184570312 MB
```

1.4.1 （1）csv

In [37]:
```python
print(df1.to_csv())
```

```
,a,b,c,d
0,51,92,14,71
1,60,20,82,86
2,74,74,87,99
3,23,2,21,52
4,1,87,29,37
5,1,63,59,20
6,32,75,57,21
7,88,48,90,58
8,41,91,59,79
9,14,61,61,46
```

In [40]:
```python
# 数据集命名为 df1
df1.to_csv("df1.csv")
```

In [41]:
```python
# 查看数据集 df1 是否创建
import os
os.path.isfile("df1.csv")
```

Out[41]: True

In [45]:
```python
# 保存 df2 所需时间
%time df2.to_csv("df2.csv")
```

```
Wall time: 3.94 s
```

In [50]:
```python
# 读取数据
df1_csv = pd.read_csv("df1.csv", index_col=0)
df1_csv.head()
```

Out[50]:

	a	b	c	d
0	51	92	14	71
1	60	20	82	86
2	74	74	87	99
3	23	2	21	52
4	1	87	29	37

1.4.2 （2）Excel

In [46]:
```python
# 第一个参数是工作簿名称，第二个是工作表名称
df1.to_excel("df1.xlsx", "df1")
```

In [48]:
```python
# 工作簿上写入多份数据集（多张数据表）
with pd.ExcelWriter("df1.xlsx") as writer:
    df1.to_excel(writer, "df1")
    (df1 + 10).to_excel(writer, "df1 plus 10")
```

In [49]:
```python
%%time
df2.to_excel("df2.xlsx")
```

```
Wall time: 38 s
```

In [51]:
```python
# 读取数据
df1_xlsx = pd.read_excel("df1.xlsx", "df1", index_col=0)
df1_xlsx.head()
```

Out[51]:

	a	b	c	d
0	51	92	14	71
1	60	20	82	86
2	74	74	87	99
3	23	2	21	52
4	1	87	29	37

1.4.3 （3）feather

The feather file format was developed for very efficient reading and writing between Python and your computer.

In [1]:
```
!pip install pyarrow
```

```
Requirement already satisfied: pyarrow in c:\users\van\anaconda3\lib\site-packages (0.15.1)
Requirement already satisfied: six>=1.0.0 in c:\users\van\anaconda3\lib\site-packages (from pya
Requirement already satisfied: numpy>=1.14 in c:\users\van\anaconda3\lib\site-packages (from py


distributed 1.21.8 requires msgpack, which is not installed.
You are using pip version 10.0.1, however version 19.3.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.
```

In [5]:
```
import pyarrow.feather
pyarrow.feather.write_feather(df1, "df1.feather")
```

In [6]:
```
%%time
pyarrow.feather.write_feather(df2, "df2.feather")
```

```
Wall time: 26.9 ms
```

In [7]:
```
df1_feather = pyarrow.feather.read_feather("df1.feather")
df1_feather.head()
```

Out[7]:

	a	b	c	d
0	51	92	14	71
1	60	20	82	86
2	74	74	87	99
3	23	2	21	52
4	1	87	29	37

对于同一份数据集 df2(10mb)，保存为 csv 格式需要 3.94s，excel 格式需要 38s，feather 格式仅需要 26.9ms

1.5 5. 清洗数据

In [8]:
```
import pandas as pd
import numpy as np
```

In [9]:
```
# 生成数据集
df = pd.DataFrame({"numbers": ["#23", "#24", "#18", "#14", "#12", "#10", "#35"],
                   "nums": ["23", "24", "18", "14", np.nan, "XYZ", "35"],
                   "colors": ["green", "red", "yellow", "orange", "purple", "blue", "pi
                   "other_column": [0, 1, 0, 2, 1, 0, 2]})
df
```

Out[9]:

	numbers	nums	colors	other_column
0	#23	23	green	0
1	#24	24	red	1
2	#18	18	yellow	0
3	#14	14	orange	2
4	#12	NaN	purple	1
5	#10	XYZ	blue	0
6	#35	35	pink	2

1.5.1 （1）字符串

```
In [10]: df["numbers"].mean()
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    821         try:
--> 822             x = float(x)
    823         except Exception:


ValueError: could not convert string to float: '#23#24#18#14#12#10#35'


During handling of the above exception, another exception occurred:


ValueError                                Traceback (most recent call last)

~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    824             try:
--> 825                 x = complex(x)
    826             except Exception:


ValueError: complex() arg is a malformed string


During handling of the above exception, another exception occurred:


TypeError                                 Traceback (most recent call last)

~\Anaconda3\lib\site-packages\pandas\core\nanops.py in f(values, axis, skipna, **kwds)
```

```
    127                 else:
--> 128                     result = alt(values, axis=axis, skipna=skipna, **kwds)
    129             except Exception:


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in nanmean(values, axis, skipna)
    354     count = _get_counts(mask, axis, dtype=dtype_count)
--> 355     the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
    356


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    827                 raise TypeError('Could not convert {value!s} to numeric'
--> 828                                 .format(value=x))
    829     return x


    TypeError: Could not convert #23#24#18#14#12#10#35 to numeric


During handling of the above exception, another exception occurred:


    ValueError                                Traceback (most recent call last)

    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    821         try:
--> 822             x = float(x)
    823         except Exception:


    ValueError: could not convert string to float: '#23#24#18#14#12#10#35'


During handling of the above exception, another exception occurred:
```

```
    ValueError                                Traceback (most recent call last)

    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    824             try:
--> 825                 x = complex(x)
    826             except Exception:


    ValueError: complex() arg is a malformed string


During handling of the above exception, another exception occurred:


    TypeError                                 Traceback (most recent call last)

    <ipython-input-10-c62df3911a80> in <module>()
----> 1 df["numbers"].mean()


    ~\Anaconda3\lib\site-packages\pandas\core\generic.py in stat_func(self, axis, skipna, l
   9587                                       skipna=skipna)
   9588         return self._reduce(f, name, axis=axis, skipna=skipna,
-> 9589                             numeric_only=numeric_only)
   9590
   9591     return set_function_name(stat_func, name, cls)


    ~\Anaconda3\lib\site-packages\pandas\core\series.py in _reduce(self, op, name, axis, sk
   3216                                           'numeric_only.'.format(name))
   3217             with np.errstate(all='ignore'):
-> 3218                 return op(delegate, skipna=skipna, **kwds)
   3219
   3220         return delegate._reduce(op=op, name=name, axis=axis, skipna=skipna,


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _f(*args, **kwargs)
```

```
     75             try:
     76                 with np.errstate(invalid='ignore'):
---> 77                     return f(*args, **kwargs)
     78             except ValueError as e:
     79                 # we want to transform an object array


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in f(values, axis, skipna, **kwds)
    129             except Exception:
    130                 try:
--> 131                     result = alt(values, axis=axis, skipna=skipna, **kwds)
    132                 except ValueError as e:
    133                     # we want to transform an object array


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in nanmean(values, axis, skipna)
    353         dtype_count = dtype
    354     count = _get_counts(mask, axis, dtype=dtype_count)
--> 355     the_sum = _ensure_numeric(values.sum(axis, dtype=dtype_sum))
    356
    357     if axis is not None and getattr(the_sum, 'ndim', False):


    ~\Anaconda3\lib\site-packages\pandas\core\nanops.py in _ensure_numeric(x)
    826             except Exception:
    827                 raise TypeError('Could not convert {value!s} to numeric'
--> 828                                 .format(value=x))
    829     return x
    830


    TypeError: Could not convert #23#24#18#14#12#10#35 to numeric
```

```
In [11]: df["numbers_str"] = df["numbers"].str.replace("#", "")

In [13]: df["numbers_str"].mean()

Out[13]: 3320259160147.857
```

```
In [14]: df["colors"].str.contains("p")
```

```
Out[14]: 0    False
         1    False
         2    False
         3    False
         4     True
         5    False
         6     True
         Name: colors, dtype: bool
```

```
In [15]: df["colors"].str.capitalize()
```

```
Out[15]: 0     Green
         1       Red
         2    Yellow
         3    Orange
         4    Purple
         5      Blue
         6      Pink
         Name: colors, dtype: object
```

1.5.2 （2）类型转换

```
In [16]: df["numbers_numeric"] = pd.to_numeric(df["numbers_str"])
```

```
In [17]: df.dtypes
```

```
Out[17]: numbers            object
         nums               object
         colors             object
         other_column        int64
         numbers_str        object
         numbers_numeric     int64
         dtype: object
```

```
In [18]: df.head()
```

```
Out[18]:   numbers nums  colors  other_column numbers_str  numbers_numeric
         0     #23   23   green             0          23               23
```

```
1      #24   24     red              1          24                24
2      #18   18  yellow              0          18                18
3      #14   14  orange              2          14                14
4      #12  NaN  purple              1          12                12
```

```
In [19]: df["numbers_numeric"].astype(str)
```

```
Out[19]: 0    23
         1    24
         2    18
         3    14
         4    12
         5    10
         6    35
         Name: numbers_numeric, dtype: object
```

```
In [20]: df["numbers_numeric"].astype(float)
```

```
Out[20]: 0    23.0
         1    24.0
         2    18.0
         3    14.0
         4    12.0
         5    10.0
         6    35.0
         Name: numbers_numeric, dtype: float64
```

1.5.3 （3）缺失数据

```
In [21]: df
```

```
Out[21]:   numbers nums  colors  other_column numbers_str  numbers_numeric
         0      #23   23   green             0          23               23
         1      #24   24     red             1          24               24
         2      #18   18  yellow             0          18               18
         3      #14   14  orange             2          14               14
         4      #12  NaN  purple             1          12               12
         5      #10  XYZ    blue             0          10               10
         6      #35   35    pink             2          35               35
```

In [22]:
```
# 找到缺失数据
df.isnull()
```

Out[22]:

	numbers	nums	colors	other_column	numbers_str	numbers_numeric
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	False	False	False
3	False	False	False	False	False	False
4	False	True	False	False	False	False
5	False	False	False	False	False	False
6	False	False	False	False	False	False

In [23]:
```
# 列是否有缺失数据
df.isnull().any(axis=0)
```

Out[23]:
```
numbers            False
nums                True
colors             False
other_column       False
numbers_str        False
numbers_numeric    False
dtype: bool
```

In [24]:
```
# 行是否有缺失数据
df.isnull().any(axis=1)
```

Out[24]:
```
0    False
1    False
2    False
3    False
4     True
5    False
6    False
dtype: bool
```

In [25]:
```
# 剔除缺失数据
df.dropna()
```

Out[25]:

	numbers	nums	colors	other_column	numbers_str	numbers_numeric
0	#23	23	green	0	23	23

1	#24	24	red	1	24	24
2	#18	18	yellow	0	18	18
3	#14	14	orange	2	14	14
5	#10	XYZ	blue	0	10	10
6	#35	35	pink	2	35	35

In [26]:
```
# 补充缺失数据
df.fillna(value=100)
```

Out[26]:

	numbers	nums	colors	other_column	numbers_str	numbers_numeric
0	#23	23	green	0	23	23
1	#24	24	red	1	24	24
2	#18	18	yellow	0	18	18
3	#14	14	orange	2	14	14
4	#12	100	purple	1	12	12
5	#10	XYZ	blue	0	10	10
6	#35	35	pink	2	35	35

In [28]:
```
# 使用后面的值补充缺失值
df.fillna(method="bfill")
```

Out[28]:

	numbers	nums	colors	other_column	numbers_str	numbers_numeric
0	#23	23	green	0	23	23
1	#24	24	red	1	24	24
2	#18	18	yellow	0	18	18
3	#14	14	orange	2	14	14
4	#12	XYZ	purple	1	12	12
5	#10	XYZ	blue	0	10	10
6	#35	35	pink	2	35	35

In [29]:
```
# 使用前面的值补充缺失值
df.fillna(method="ffill")
```

Out[29]:

	numbers	nums	colors	other_column	numbers_str	numbers_numeric
0	#23	23	green	0	23	23
1	#24	24	red	1	24	24
2	#18	18	yellow	0	18	18
3	#14	14	orange	2	14	14
4	#12	14	purple	1	12	12
5	#10	XYZ	blue	0	10	10
6	#35	35	pink	2	35	35

1.6 6. 数据变形

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

%matplotlib inline
```

In [2]:
```
# 示例数据
url = "https://storage.googleapis.com/qeds/data/bball.csv"
bball = pd.read_csv(url)
bball.info()

bball
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 8 columns):
Year        9 non-null int64
Player      9 non-null object
Team        9 non-null object
TeamName    9 non-null object
Games       9 non-null int64
Pts         9 non-null float64
Assist      9 non-null float64
Rebound     9 non-null float64
dtypes: float64(3), int64(2), object(3)
memory usage: 656.0+ bytes
```

Out[2]:

	Year	Player	Team	TeamName	Games	Pts	Assist	Rebound
0	2015	Curry	GSW	Warriors	79	30.1	6.7	5.4
1	2016	Curry	GSW	Warriors	79	25.3	6.6	4.5
2	2017	Curry	GSW	Warriors	51	26.4	6.1	5.1
3	2015	Durant	OKC	Thunder	72	28.2	5.0	8.2
4	2016	Durant	GSW	Warriors	62	25.1	4.8	8.3
5	2017	Durant	GSW	Warriors	68	26.4	5.4	6.8
6	2015	Ibaka	OKC	Thunder	78	12.6	0.8	6.8
7	2016	Ibaka	ORL	Magic	56	15.1	1.1	6.8
8	2016	Ibaka	TOR	Raptors	23	14.2	0.7	6.8

1.6.1 （1）长宽变换

In [3]:
```
# 宽数据 长数据
#pandas.melt(frame, id_vars=None, value_vars=None,
# var_name=None, value_name='value', col_level=None)
#id_vars: 不需要被转换的列名
#value_vars: 需要转换的列名，如果剩下的列全部都要转换，就不用写了

bball_long = bball.melt(id_vars=["Year", "Player", "Team", "TeamName"])

bball_long
```

Out[3]:

	Year	Player	Team	TeamName	variable	value
0	2015	Curry	GSW	Warriors	Games	79.0
1	2016	Curry	GSW	Warriors	Games	79.0
2	2017	Curry	GSW	Warriors	Games	51.0
3	2015	Durant	OKC	Thunder	Games	72.0
4	2016	Durant	GSW	Warriors	Games	62.0
5	2017	Durant	GSW	Warriors	Games	68.0
6	2015	Ibaka	OKC	Thunder	Games	78.0
7	2016	Ibaka	ORL	Magic	Games	56.0
8	2016	Ibaka	TOR	Raptors	Games	23.0
9	2015	Curry	GSW	Warriors	Pts	30.1
10	2016	Curry	GSW	Warriors	Pts	25.3
11	2017	Curry	GSW	Warriors	Pts	26.4
12	2015	Durant	OKC	Thunder	Pts	28.2
13	2016	Durant	GSW	Warriors	Pts	25.1
14	2017	Durant	GSW	Warriors	Pts	26.4
15	2015	Ibaka	OKC	Thunder	Pts	12.6
16	2016	Ibaka	ORL	Magic	Pts	15.1
17	2016	Ibaka	TOR	Raptors	Pts	14.2
18	2015	Curry	GSW	Warriors	Assist	6.7
19	2016	Curry	GSW	Warriors	Assist	6.6
20	2017	Curry	GSW	Warriors	Assist	6.1
21	2015	Durant	OKC	Thunder	Assist	5.0
22	2016	Durant	GSW	Warriors	Assist	4.8
23	2017	Durant	GSW	Warriors	Assist	5.4
24	2015	Ibaka	OKC	Thunder	Assist	0.8

```
25  2016   Ibaka  ORL      Magic   Assist    1.1
26  2016   Ibaka  TOR    Raptors   Assist    0.7
27  2015   Curry  GSW   Warriors  Rebound    5.4
28  2016   Curry  GSW   Warriors  Rebound    4.5
29  2017   Curry  GSW   Warriors  Rebound    5.1
30  2015  Durant  OKC    Thunder  Rebound    8.2
31  2016  Durant  GSW   Warriors  Rebound    8.3
32  2017  Durant  GSW   Warriors  Rebound    6.8
33  2015   Ibaka  OKC    Thunder  Rebound    6.8
34  2016   Ibaka  ORL      Magic  Rebound    6.8
35  2016   Ibaka  TOR    Raptors  Rebound    6.8
```

In [4]:
```
# 长数据 宽数据
#pivot_table(data, values=None, index=None, columns=None,aggfunc='mean',
# fill_value=None, margins=False, dropna=True, margins_name='All')

bball_wide = bball_long.pivot_table(
    index="Year",
    columns=["Player", "variable", "Team"],
    values="value"
)
bball_wide
```

Out[4]:
```
Player    Curry                         Durant                              ...   \
variable Assist Games   Pts Rebound Assist       Games         Pts          ...
Team        GSW   GSW   GSW     GSW    GSW  OKC    GSW   OKC   GSW   OKC ...
Year                                                                      ...
2015        6.7  79.0  30.1     5.4    NaN  5.0    NaN  72.0   NaN  28.2 ...
2016        6.6  79.0  25.3     4.5    4.8  NaN   62.0   NaN  25.1   NaN ...
2017        6.1  51.0  26.4     5.1    5.4  NaN   68.0   NaN  26.4   NaN ...

Player    Ibaka
variable Assist Games                   Pts               Rebound
Team        TOR   OKC   ORL   TOR   OKC   ORL   TOR        OKC  ORL  TOR
Year
2015        NaN  78.0   NaN   NaN  12.6   NaN   NaN        6.8  NaN  NaN
2016        0.7   NaN  56.0  23.0   NaN  15.1  14.2        NaN  6.8  6.8
2017        NaN   NaN   NaN   NaN   NaN   NaN   NaN        NaN  NaN  NaN
```

```
[3 rows x 24 columns]
```

1.6.2 （2）索引操作

In [5]:
```
# 设置索引
bball2 = bball.set_index(["Player", "Year"])
bball2.head()
```

Out[5]:
```
             Team  TeamName  Games   Pts  Assist  Rebound
Player Year
Curry  2015   GSW  Warriors     79  30.1     6.7      5.4
       2016   GSW  Warriors     79  25.3     6.6      4.5
       2017   GSW  Warriors     51  26.4     6.1      5.1
Durant 2015   OKC   Thunder     72  28.2     5.0      8.2
       2016   GSW  Warriors     62  25.1     4.8      8.3
```

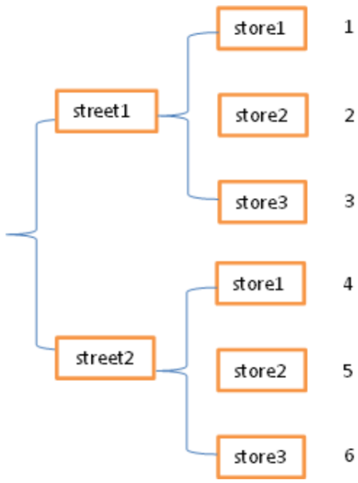
In [7]:
```
# 转置
bball3 = bball2.T
bball3.head()
```

Out[7]:
```
Player       Curry                        Durant                         Ibaka  \
Year          2015      2016      2017      2015      2016      2017      2015
Team           GSW       GSW       GSW       OKC       GSW       GSW       OKC
TeamName  Warriors  Warriors  Warriors   Thunder  Warriors  Warriors   Thunder
Games           79        79        51        72        62        68        78
Pts           30.1      25.3      26.4      28.2      25.1      26.4      12.6
Assist         6.7       6.6       6.1         5       4.8       5.4       0.8

Player
Year        2016     2016
Team         ORL      TOR
TeamName   Magic  Raptors
Games         56       23
Pts         15.1     14.2
Assist       1.1      0.7
```

stack 和 unstack

stack 和 unstack 是 python 进行层次化索引的重要操作。层次化索引就是对索引进行层次化分类，便于使用，这里的索引可以是行索引，也可以是列索引。



	store1	store2	store3
street1	1	2	3
street2	4	5	6

常见的数据的层次化结构有两种，一种是表格，一种是“花括号”，即下面这样的两种形式：表格在行列方向上均有索引，花括号结构只有“列方向”上的索引。

- stack: 将数据从”表格结构“变成”花括号结构“，即将其列索引变成行索引。
- unstack: 数据从”花括号结构“变成”表格结构“，即要将其中一层的行索引变成列索引。

参考资料：
Python--pandas--unstack() 与 stack()

In [8]:
```
bball_wide
```

Out[8]:
```
Player     Curry                         Durant                          ...   \
variable Assist Games   Pts Rebound Assist      Games        Pts         ...
Team        GSW   GSW   GSW     GSW    GSW  OKC   GSW   OKC  GSW   OKC ...
Year                                                                     ...
2015        6.7  79.0  30.1     5.4    NaN  5.0   NaN  72.0  NaN  28.2 ...
2016        6.6  79.0  25.3     4.5    4.8  NaN  62.0   NaN 25.1   NaN ...
2017        6.1  51.0  26.4     5.1    5.4  NaN  68.0   NaN 26.4   NaN ...

Player     Ibaka
variable Assist Games                 Pts            Rebound
Team        TOR   OKC   ORL   TOR   OKC   ORL   TOR     OKC  ORL  TOR
Year
2015        NaN  78.0   NaN   NaN  12.6   NaN   NaN     6.8  NaN  NaN
2016        0.7   NaN  56.0  23.0   NaN  15.1  14.2     NaN  6.8  6.8
2017        NaN   NaN   NaN   NaN   NaN   NaN   NaN     NaN  NaN  NaN
```

```
[3 rows x 24 columns]
```

In [10]:
```
# 把列索引变为行索引
bball_wide.stack()
```

Out[10]:
```
Player     Curry                          Durant                          Ibaka          \
variable  Assist Games   Pts Rebound Assist Games   Pts Rebound Assist Games
Year Team
2015 GSW     6.7  79.0  30.1     5.4    NaN   NaN   NaN     NaN    NaN   NaN
     OKC     NaN   NaN   NaN     NaN    5.0  72.0  28.2     8.2    0.8  78.0
2016 GSW     6.6  79.0  25.3     4.5    4.8  62.0  25.1     8.3    NaN   NaN
     ORL     NaN   NaN   NaN     NaN    NaN   NaN   NaN     NaN    1.1  56.0
     TOR     NaN   NaN   NaN     NaN    NaN   NaN   NaN     NaN    0.7  23.0
2017 GSW     6.1  51.0  26.4     5.1    5.4  68.0  26.4     6.8    NaN   NaN

Player
variable   Pts Rebound
Year Team
2015 GSW   NaN     NaN
     OKC  12.6     6.8
2016 GSW   NaN     NaN
     ORL  15.1     6.8
     TOR  14.2     6.8
2017 GSW   NaN     NaN
```

In [12]:
```
# 对运动员的数据，按照 Year 和 Team 求均值
player_stats = bball_wide.stack().mean()
player_stats
```

Out[12]:
```
Player  variable
Curry   Assist       6.466667
        Games       69.666667
        Pts         27.266667
        Rebound      5.000000
Durant  Assist       5.066667
        Games       67.333333
        Pts         26.566667
        Rebound      7.766667
```

```
Ibaka   Assist       0.866667
        Games       52.333333
        Pts         13.966667
        Rebound      6.800000
dtype: float64
```

In [13]:
```
# 对数据和球队，按照 Year 和 Player 求均值
bball_wide.stack(level="Player")
```

Out[13]:
```
variable     Assist                Games                      Pts             \
Team            GSW  OKC  ORL  TOR   GSW   OKC   ORL   TOR   GSW   OKC   ORL
Year Player
2015 Curry      6.7  NaN  NaN  NaN  79.0   NaN   NaN   NaN  30.1   NaN   NaN
     Durant     NaN  5.0  NaN  NaN   NaN  72.0   NaN   NaN   NaN  28.2   NaN
     Ibaka      NaN  0.8  NaN  NaN   NaN  78.0   NaN   NaN   NaN  12.6   NaN
2016 Curry      6.6  NaN  NaN  NaN  79.0   NaN   NaN   NaN  25.3   NaN   NaN
     Durant     4.8  NaN  NaN  NaN  62.0   NaN   NaN   NaN  25.1   NaN   NaN
     Ibaka      NaN  NaN  1.1  0.7   NaN   NaN  56.0  23.0   NaN   NaN  15.1
2017 Curry      6.1  NaN  NaN  NaN  51.0   NaN   NaN   NaN  26.4   NaN   NaN
     Durant     5.4  NaN  NaN  NaN  68.0   NaN   NaN   NaN  26.4   NaN   NaN

variable          Rebound
Team           TOR     GSW  OKC  ORL  TOR
Year Player
2015 Curry     NaN     5.4  NaN  NaN  NaN
     Durant    NaN     NaN  8.2  NaN  NaN
     Ibaka     NaN     NaN  6.8  NaN  NaN
2016 Curry     NaN     4.5  NaN  NaN  NaN
     Durant    NaN     8.3  NaN  NaN  NaN
     Ibaka    14.2     NaN  NaN  6.8  6.8
2017 Curry     NaN     5.1  NaN  NaN  NaN
     Durant    NaN     6.8  NaN  NaN  NaN
```

In [14]:
```
bball_wide.stack(level="Player").mean()
```

Out[14]:
```
variable  Team
Assist    GSW     5.92
          OKC     2.90
          ORL     1.10
```

```
         TOR      0.70
Games    GSW     67.80
         OKC     75.00
         ORL     56.00
         TOR     23.00
Pts      GSW     26.66
         OKC     20.40
         ORL     15.10
         TOR     14.20
Rebound  GSW      6.02
         OKC      7.50
         ORL      6.80
         TOR      6.80
dtype: float64
```

In [15]:
```
bball_wide.stack(level=["Player", "Team"])
```

Out[15]:
```
variable          Assist  Games   Pts  Rebound
Year Player Team
2015 Curry  GSW      6.7   79.0  30.1      5.4
     Durant OKC      5.0   72.0  28.2      8.2
     Ibaka  OKC      0.8   78.0  12.6      6.8
2016 Curry  GSW      6.6   79.0  25.3      4.5
     Durant GSW      4.8   62.0  25.1      8.3
     Ibaka  ORL      1.1   56.0  15.1      6.8
            TOR      0.7   23.0  14.2      6.8
2017 Curry  GSW      6.1   51.0  26.4      5.1
     Durant GSW      5.4   68.0  26.4      6.8
```

In [19]:
```
player_stats
```

Out[19]:
```
Player  variable
Curry   Assist       6.466667
        Games       69.666667
        Pts         27.266667
        Rebound      5.000000
Durant  Assist       5.066667
        Games       67.333333
        Pts         26.566667
```

```
        Rebound      7.766667
Ibaka   Assist       0.866667
        Games       52.333333
        Pts         13.966667
        Rebound      6.800000
dtype: float64
```

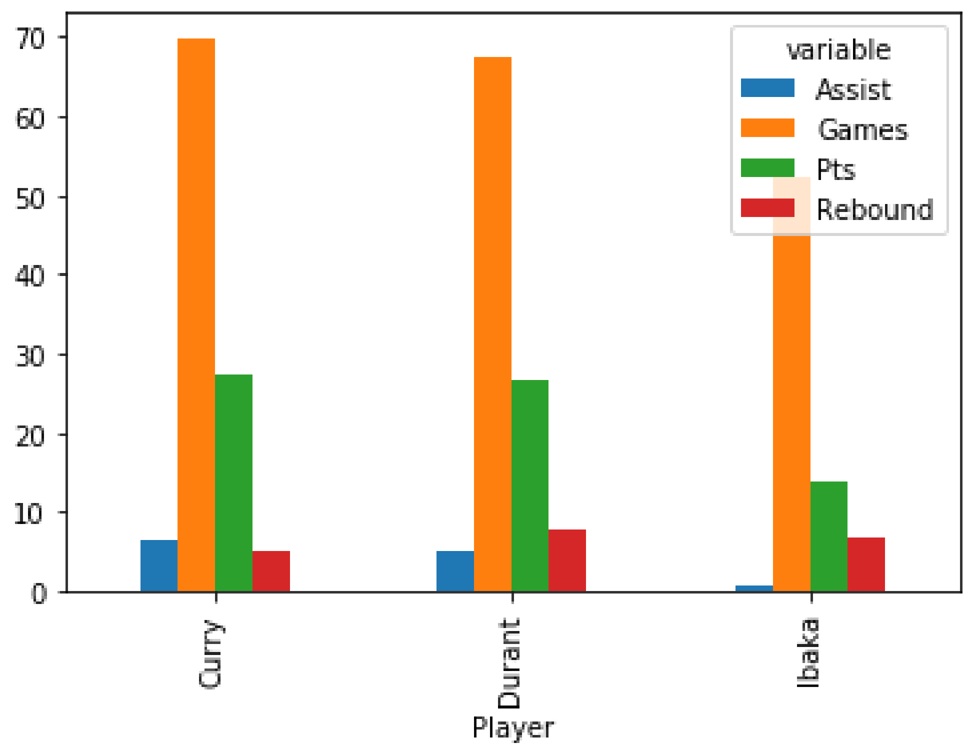
In [17]:
```
# 把行索引变为列索引
player_stats.unstack()
```

Out[17]:

variable	Assist	Games	Pts	Rebound
Player				
Curry	6.466667	69.666667	27.266667	5.000000
Durant	5.066667	67.333333	26.566667	7.766667
Ibaka	0.866667	52.333333	13.966667	6.800000

In [20]:
```
player_stats.unstack().plot.bar()
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1e662e22fd0>

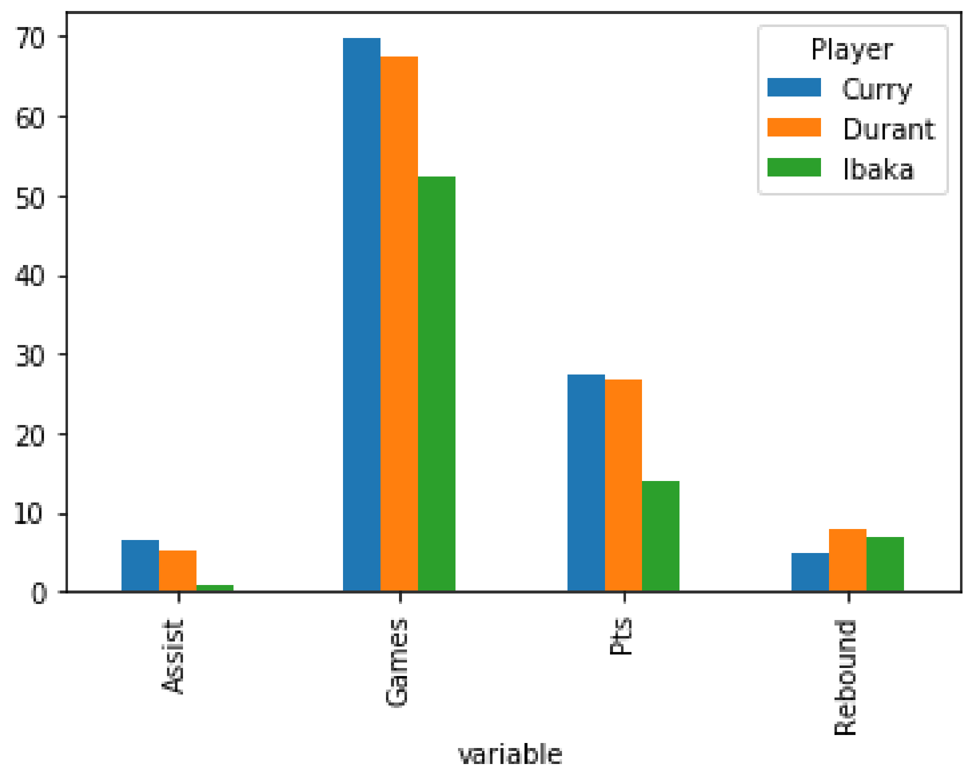


```
In [21]: player_stats.unstack(level="Player")
```

```
Out[21]: Player        Curry     Durant      Ibaka
         variable
         Assist     6.466667   5.066667   0.866667
         Games     69.666667  67.333333  52.333333
         Pts       27.266667  26.566667  13.966667
         Rebound    5.000000   7.766667   6.800000
```

```
In [22]: player_stats.unstack(level="Player").plot.bar()
```

```
Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1e6631a0c50>
```



1.6.3 （3）透视表

```
In [23]: bball
```

```
Out[23]:    Year  Player Team  TeamName  Games   Pts  Assist  Rebound
         0  2015   Curry  GSW  Warriors     79  30.1     6.7      5.4
```

```
1  2016   Curry  GSW  Warriors     79  25.3     6.6       4.5
2  2017   Curry  GSW  Warriors     51  26.4     6.1       5.1
3  2015  Durant  OKC   Thunder     72  28.2     5.0       8.2
4  2016  Durant  GSW  Warriors     62  25.1     4.8       8.3
5  2017  Durant  GSW  Warriors     68  26.4     5.4       6.8
6  2015   Ibaka  OKC   Thunder     78  12.6     0.8       6.8
7  2016   Ibaka  ORL     Magic     56  15.1     1.1       6.8
8  2016   Ibaka  TOR   Raptors     23  14.2     0.7       6.8
```

```
In [24]: # 多行索引
         bball.pivot_table(index = ["Year", "Team"], columns = "Player", values = "Pts")
```

```
Out[24]: Player     Curry  Durant  Ibaka
         Year Team
         2015 GSW    30.1     NaN    NaN
              OKC     NaN    28.2   12.6
         2016 GSW    25.3    25.1    NaN
              ORL     NaN     NaN   15.1
              TOR     NaN     NaN   14.2
         2017 GSW    26.4    26.4    NaN
```

```
In [25]: # 多列索引
         bball.pivot_table(index = "Year", columns = ["Player", "Team"], values = "Pts")
```

```
Out[25]: Player Curry Durant       Ibaka
         Team     GSW    GSW   OKC   OKC   ORL   TOR
         Year
         2015    30.1    NaN  28.2  12.6   NaN   NaN
         2016    25.3   25.1   NaN   NaN  15.1  14.2
         2017    26.4   26.4   NaN   NaN   NaN   NaN
```

```
In [26]: # 求最值
         bball.pivot_table(index="Year", columns="Player", values="Pts", aggfunc=max)
```

```
Out[26]: Player  Curry  Durant  Ibaka
         Year
         2015     30.1    28.2   12.6
         2016     25.3    25.1   15.1
         2017     26.4    26.4    NaN
```

In [27]: # 有多少数值
```
bball.pivot_table(index="Year", columns="Player", values="Pts", aggfunc=len)
```

Out[27]:

Player	Curry	Durant	Ibaka
Year			
2015	1.0	1.0	1.0
2016	1.0	1.0	2.0
2017	1.0	1.0	NaN

In [29]: # 多函操作
```
bball.pivot_table(index="Year", columns="Player", values="Pts", aggfunc=[max, len])
```

Out[29]:

	max			len		
Player	Curry	Durant	Ibaka	Curry	Durant	Ibaka
Year						
2015	30.1	28.2	12.6	1.0	1.0	1.0
2016	25.3	25.1	15.1	1.0	1.0	2.0
2017	26.4	26.4	NaN	1.0	1.0	NaN

1.7 7. 数据合并

In [ ]:
```
import pandas as pd

%matplotlib inline
```

In [36]:
```
# 示例数据

url = "https://storage.googleapis.com/qeds/data/wdi_data.csv"
wdi = pd.read_csv(url).set_index(["country", "year"])
wdi.info()

wdi2017 = wdi.xs(2017, level="year")
wdi2017
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 72 entries, (Canada, 2017) to (United States, 2000)
Data columns (total 5 columns):
GovExpend      72 non-null float64
Consumption    72 non-null float64
```

```
Exports        72 non-null float64
Imports        72 non-null float64
GDP            72 non-null float64
dtypes: float64(5)
memory usage: 3.2+ KB
```

Out[36]:

country	GovExpend	Consumption	Exports	Imports	GDP
Canada	0.372665	1.095475	0.582831	0.600031	1.868164
Germany	0.745579	2.112009	1.930563	1.666348	3.883870
United Kingdom	0.549538	1.809154	0.862629	0.933145	2.818704
United States	2.405743	12.019266	2.287071	3.069954	17.348627

In [33]:

```
wdi2016_17 = wdi.loc[pd.IndexSlice[:, [2016, 2017]],: ]
wdi2016_17
```

Out[33]:

country	year	GovExpend	Consumption	Exports	Imports	GDP
Canada	2017	0.372665	1.095475	0.582831	0.600031	1.868164
	2016	0.364899	1.058426	0.576394	0.575775	1.814016
Germany	2017	0.745579	2.112009	1.930563	1.666348	3.883870
	2016	0.734014	2.075615	1.844949	1.589495	3.801859
United Kingdom	2017	0.549538	1.809154	0.862629	0.933145	2.818704
	2016	0.550596	1.772348	0.816792	0.901494	2.768241
United States	2017	2.405743	12.019266	2.287071	3.069954	17.348627
	2016	2.407981	11.722133	2.219937	2.936004	16.972348

In [34]:

```
sq_miles = pd.Series({
   "United States": 3.8,
   "Canada": 3.8,
   "Germany": 0.137,
   "United Kingdom": 0.0936,
   "Russia": 6.6,
}, name="sq_miles").to_frame()
sq_miles.index.name = "country"
sq_miles
```

Out[34]:

```
                sq_miles
country
```

```
United States     3.8000
Canada            3.8000
Germany           0.1370
United Kingdom    0.0936
Russia            6.6000
```

In [35]:
```python
pop_url = "https://storage.googleapis.com/qeds/data/wdi_population.csv"
pop = pd.read_csv(pop_url).set_index(["country", "year"])
pop.info()
pop.head(10)
```

```
<class 'pandas.core.frame.DataFrame'>
MultiIndex: 72 entries, (Canada, 2017) to (United States, 2000)
Data columns (total 1 columns):
Population    72 non-null float64
dtypes: float64(1)
memory usage: 1005.0+ bytes
```

Out[35]:

country	year	Population
Canada	2017	36.540268
	2016	36.109487
	2015	35.702908
	2014	35.437435
	2013	35.082954
	2012	34.714222
	2011	34.339328
	2010	34.004889
	2009	33.628895
	2008	33.247118

1.7.1 （1）contact

- As a measure of land usage or productivity, what is Consumption per square mile?
- What is GDP per capita (per person) for each country in each year? How about Consumption per person?
- What is the population density of each country? How much does it change over time?

To answer any of the questions from above, we will have to use data from more than one of our DataFrames.

```
In [37]: # 纵向堆叠
         pd.concat([wdi2017, sq_miles], axis=0)
```

```
C:\Users\Van\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Sorting becau
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False
```

```
Out[37]:                   Consumption   Exports        GDP  GovExpend   Imports  \
         country
         Canada               1.095475  0.582831   1.868164   0.372665  0.600031
         Germany              2.112009  1.930563   3.883870   0.745579  1.666348
         United Kingdom       1.809154  0.862629   2.818704   0.549538  0.933145
         United States       12.019266  2.287071  17.348627   2.405743  3.069954
         United States             NaN       NaN        NaN        NaN       NaN
         Canada                    NaN       NaN        NaN        NaN       NaN
         Germany                   NaN       NaN        NaN        NaN       NaN
         United Kingdom            NaN       NaN        NaN        NaN       NaN
         Russia                    NaN       NaN        NaN        NaN       NaN

                           sq_miles
         country
         Canada                 NaN
         Germany                NaN
         United Kingdom         NaN
         United States          NaN
         United States       3.8000
         Canada              3.8000
         Germany             0.1370
         United Kingdom      0.0936
         Russia              6.6000
```

In [38]:
```python
# 横向堆叠
pd.concat([wdi2017, sq_miles], axis=1)
```

```
C:\Users\Van\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Sorting becau
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False

  """Entry point for launching an IPython kernel.
```

Out[38]:

```
                GovExpend  Consumption   Exports   Imports        GDP  \
Canada           0.372665     1.095475  0.582831  0.600031   1.868164
Germany          0.745579     2.112009  1.930563  1.666348   3.883870
Russia                NaN          NaN       NaN       NaN        NaN
United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
United States    2.405743    12.019266  2.287071  3.069954  17.348627

                sq_miles
Canada            3.8000
Germany           0.1370
Russia            6.6000
United Kingdom    0.0936
United States     3.8000
```

In [39]:
```python
#What is Consumption per square mile?
temp = pd.concat([wdi2017, sq_miles], axis=1)
temp["Consumption"] / temp["sq_miles"]
```

```
C:\Users\Van\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: Sorting becau
of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=True'.

To retain the current behavior and silence the warning, pass sort=False
```

```
Out[39]: Canada             0.288283
         Germany           15.416124
         Russia                  NaN
         United Kingdom    19.328569
         United States      3.162965
         dtype: float64
```

1.7.2 （2）merge

```
In [40]: # 和使用 contact 函数（axis=1）相比，Russia 数据不见了
         pd.merge(wdi2017, sq_miles, on="country")
```

```
Out[40]:                 GovExpend  Consumption   Exports   Imports        GDP  \
         country
         Canada           0.372665     1.095475  0.582831  0.600031   1.868164
         Germany          0.745579     2.112009  1.930563  1.666348   3.883870
         United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
         United States    2.405743    12.019266  2.287071  3.069954  17.348627

                         sq_miles
         country
         Canada            3.8000
         Germany           0.1370
         United Kingdom    0.0936
         United States     3.8000
```

```
In [41]: # 年份信息丢失了
         pd.merge(wdi2016_17, sq_miles, on="country")
```

```
Out[41]:                 GovExpend  Consumption   Exports   Imports        GDP  \
         country
         Canada           0.372665     1.095475  0.582831  0.600031   1.868164
         Canada           0.364899     1.058426  0.576394  0.575775   1.814016
         Germany          0.745579     2.112009  1.930563  1.666348   3.883870
         Germany          0.734014     2.075615  1.844949  1.589495   3.801859
         United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
```

```
United Kingdom   0.550596     1.772348  0.816792  0.901494   2.768241
United States    2.405743    12.019266  2.287071  3.069954  17.348627
United States    2.407981    11.722133  2.219937  2.936004  16.972348

                sq_miles
country
Canada            3.8000
Canada            3.8000
Germany           0.1370
Germany           0.1370
United Kingdom    0.0936
United Kingdom    0.0936
United States     3.8000
United States     3.8000
```

In [42]:
```
pd.merge(wdi2016_17.reset_index(), sq_miles, on="country")
```

Out[42]:
```
          country  year  GovExpend  Consumption   Exports   Imports  \
0          Canada  2017   0.372665     1.095475  0.582831  0.600031
1          Canada  2016   0.364899     1.058426  0.576394  0.575775
2         Germany  2017   0.745579     2.112009  1.930563  1.666348
3         Germany  2016   0.734014     2.075615  1.844949  1.589495
4  United Kingdom  2017   0.549538     1.809154  0.862629  0.933145
5  United Kingdom  2016   0.550596     1.772348  0.816792  0.901494
6   United States  2017   2.405743    12.019266  2.287071  3.069954
7   United States  2016   2.407981    11.722133  2.219937  2.936004

         GDP  sq_miles
0   1.868164    3.8000
1   1.814016    3.8000
2   3.883870    0.1370
3   3.801859    0.1370
4   2.818704    0.0936
5   2.768241    0.0936
6  17.348627    3.8000
7  16.972348    3.8000
```

In [43]:
```
# 合并多列
pd.merge(wdi2016_17, pop, on=["country", "year"])
```

```
Out[43]:                         GovExpend  Consumption   Exports   Imports        GDP  \
         country        year
         Canada         2017   0.372665     1.095475  0.582831  0.600031   1.868164
                        2016   0.364899     1.058426  0.576394  0.575775   1.814016
         Germany        2017   0.745579     2.112009  1.930563  1.666348   3.883870
                        2016   0.734014     2.075615  1.844949  1.589495   3.801859
         United Kingdom 2017   0.549538     1.809154  0.862629  0.933145   2.818704
                        2016   0.550596     1.772348  0.816792  0.901494   2.768241
         United States  2017   2.405743    12.019266  2.287071  3.069954  17.348627
                        2016   2.407981    11.722133  2.219937  2.936004  16.972348

                         Population
         country        year
         Canada         2017   36.540268
                        2016   36.109487
         Germany        2017   82.657002
                        2016   82.348669
         United Kingdom 2017   66.058859
                        2016   65.595565
         United States  2017  325.147121
                        2016  323.071342
```

```
In [44]: #What is GDP per capita (per person) for each country in each year?
         wdi_pop = pd.merge(wdi2016_17, pop, on=["country", "year"])
         wdi_pop["GDP"] / wdi_pop["Population"]
```

```
Out[44]: country         year
         Canada          2017    0.051126
                         2016    0.050237
         Germany         2017    0.046988
                         2016    0.046168
         United Kingdom  2017    0.042670
                         2016    0.042202
         United States   2017    0.053356
                         2016    0.052534
         dtype: float64
```

```
In [46]: #How about Consumption per person?
         wdi_pop["Consumption"] / wdi_pop["Population"]
```

Out[46]:
```
country         year
Canada          2017    0.029980
                2016    0.029312
Germany         2017    0.025551
                2016    0.025205
United Kingdom  2017    0.027387
                2016    0.027019
United States   2017    0.036966
                2016    0.036283
dtype: float64
```

In [51]:
```
wdi2017_no_US = wdi2017.drop("United States")
wdi2017_no_US
```

Out[51]:

	GovExpend	Consumption	Exports	Imports	GDP
country					
Canada	0.372665	1.095475	0.582831	0.600031	1.868164
Germany	0.745579	2.112009	1.930563	1.666348	3.883870
United Kingdom	0.549538	1.809154	0.862629	0.933145	2.818704

In [52]:
```
sq_miles_no_germany = sq_miles.drop("Germany")
sq_miles_no_germany
```

Out[52]:

	sq_miles
country	
United States	3.8000
Canada	3.8000
United Kingdom	0.0936
Russia	6.6000

In [47]:
```
# 以左表为基准
pd.merge(wdi2017, sq_miles, on="country", how="left")
```

Out[47]:

	GovExpend	Consumption	Exports	Imports	GDP \
country					
Canada	0.372665	1.095475	0.582831	0.600031	1.868164
Germany	0.745579	2.112009	1.930563	1.666348	3.883870
United Kingdom	0.549538	1.809154	0.862629	0.933145	2.818704
United States	2.405743	12.019266	2.287071	3.069954	17.348627

```
                 sq_miles
country
Canada             3.8000
Germany            0.1370
United Kingdom     0.0936
United States      3.8000
```

In [48]:
```python
# 以右表为基准
pd.merge(wdi2017, sq_miles, on="country", how="right")
```

Out[48]:
```
                GovExpend  Consumption   Exports   Imports        GDP  \
country
Canada           0.372665     1.095475  0.582831  0.600031   1.868164
Germany          0.745579     2.112009  1.930563  1.666348   3.883870
United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
United States    2.405743    12.019266  2.287071  3.069954  17.348627
Russia                NaN          NaN       NaN       NaN        NaN

                 sq_miles
country
Canada             3.8000
Germany            0.1370
United Kingdom     0.0936
United States      3.8000
Russia             6.6000
```

In [53]:
```python
# 内连接，相当于交集操作
pd.merge(wdi2017_no_US, sq_miles, on="country", how="inner")
```

Out[53]:
```
                GovExpend  Consumption   Exports   Imports       GDP  sq_miles
country
Canada           0.372665     1.095475  0.582831  0.600031  1.868164    3.8000
Germany          0.745579     2.112009  1.930563  1.666348  3.883870    0.1370
United Kingdom   0.549538     1.809154  0.862629  0.933145  2.818704    0.0936
```

In [54]:
```python
# 外连接，相当于并集操作
pd.merge(wdi2017_no_US, sq_miles_no_germany, on="country", how="outer")
```

```
Out[54]:                 GovExpend  Consumption   Exports   Imports       GDP  sq_miles
         country
         Canada           0.372665     1.095475  0.582831  0.600031  1.868164    3.8000
         Germany          0.745579     2.112009  1.930563  1.666348  3.883870       NaN
         United Kingdom   0.549538     1.809154  0.862629  0.933145  2.818704    0.0936
         United States         NaN          NaN       NaN       NaN       NaN    3.8000
         Russia                NaN          NaN       NaN       NaN       NaN    6.6000
```

1.7.3 （3）join

```
In [55]: wdi2017.join(sq_miles, on="country")
```

```
Out[55]:                 GovExpend  Consumption   Exports   Imports        GDP  \
         country
         Canada           0.372665     1.095475  0.582831  0.600031   1.868164
         Germany          0.745579     2.112009  1.930563  1.666348   3.883870
         United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
         United States    2.405743    12.019266  2.287071  3.069954  17.348627

                         sq_miles
         country
         Canada            3.8000
         Germany           0.1370
         United Kingdom    0.0936
         United States     3.8000
```

```
In [56]: pd.merge(wdi2017, sq_miles, left_on="country", right_index=True)
```

```
Out[56]:                 GovExpend  Consumption   Exports   Imports        GDP  \
         country
         Canada           0.372665     1.095475  0.582831  0.600031   1.868164
         Germany          0.745579     2.112009  1.930563  1.666348   3.883870
         United Kingdom   0.549538     1.809154  0.862629  0.933145   2.818704
         United States    2.405743    12.019266  2.287071  3.069954  17.348627

                         sq_miles
         country
         Canada            3.8000
         Germany           0.1370
```

```
United Kingdom    0.0936
United States     3.8000
```

1.8 8. 分组操作

```
In [65]: import random
         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
         %matplotlib inline
```

```
In [66]: C = np.arange(1, 7, dtype=float)
         C[[3, 5]] = np.nan
         df = pd.DataFrame({
             "A" : [1, 1, 1, 2, 2, 2],
             "B" : [1, 1, 2, 2, 1, 1],
             "C": C,
         })
         df
```

Out[66]:

	A	B	C
0	1	1	1.0
1	1	1	2.0
2	1	2	3.0
3	2	2	NaN
4	2	1	5.0
5	2	1	NaN

```
In [67]: # 按照 A 列信息分组
         gbA = df.groupby("A")
```

```
In [69]: type(gbA)
```

```
Out[69]: pandas.core.groupby.groupby.DataFrameGroupBy
```

```
In [70]: gbA.get_group(1)
```

Out[70]:

	A	B	C
0	1	1	1.0
1	1	1	2.0
2	1	2	3.0

```
In [71]: gbA.get_group(2)
```

```
Out[71]:    A  B    C
         3  2  2  NaN
         4  2  1  5.0
         5  2  1  NaN
```

```
In [72]: # 按 A、B 列信息分组
         gbAB = df.groupby(["A", "B"])
         type(gbAB)
```

```
Out[72]: pandas.core.groupby.groupby.DataFrameGroupBy
```

```
In [73]: gbAB.get_group((1, 1))
```

```
Out[73]:    A  B    C
         0  1  1  1.0
         1  1  1  2.0
```

```
In [74]: # 使用函数
         gbAB.count()
```

```
Out[74]:      C
         A B
         1 1  2
           2  1
         2 1  1
           2  0
```

```
In [75]: def num_missing(df):
             return df.isnull().sum()
```

```
In [76]: num_missing(df)
```

```
Out[76]: A    0
         B    0
         C    2
         dtype: int64
```

```
In [77]: gbA.agg(num_missing)
```

```
Out[77]:    B    C
         A
         1  0  0.0
         2  0  2.0
```

```
In [80]: df
```

```
Out[80]:    A  B    C
         0  1  1  1.0
         1  1  1  2.0
         2  1  2  3.0
         3  2  2  NaN
         4  2  1  5.0
         5  2  1  NaN
```

```
In [78]: #B 列最小的 2 个数
         def smallest_by_b(df):
             return df.nsmallest(2, "B")
```

```
In [79]: gbA.apply(smallest_by_b)
```

```
Out[79]:      A  B    C
         A
         1 0  1  1  1.0
           1  1  1  2.0
         2 4  2  1  5.0
           5  2  1  NaN
```

Sometimes, in order to construct the groups you want, you need to give pandas more information than just a column name.

Some examples are:

- Grouping by a column and a level of the index.
- Grouping time series data at a particular frequency.

pandas lets you do this through the pd.Grouper type.

```
In [81]: df2 = df.copy()
         df2["Date"] = pd.date_range(
             start=pd.datetime.today().strftime("%m/%d/%Y"),
```

```
    freq="BQ",
    periods=df.shape[0]
)
df2 = df2.set_index("A")
df2
```

Out[81]:

```
   B    C        Date
A
1  1  1.0 2019-12-31
1  1  2.0 2020-03-31
1  2  3.0 2020-06-30
2  2  NaN 2020-09-30
2  1  5.0 2020-12-31
2  1  NaN 2021-03-31
```

In [86]:

```
#freq = “A” 年频率
#https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-alias

df2.groupby(pd.Grouper(key="Date", freq="A")).count()
```

Out[86]:

```
            B  C
Date
2019-12-31  1  1
2020-12-31  4  3
2021-12-31  1  0
```

In [87]:

```
df2.groupby(pd.Grouper(level="A")).count()
```

Out[87]:

```
   B  C  Date
A
1  3  3     3
2  3  1     3
```

In [88]:

```
df2.groupby([pd.Grouper(key="Date", freq="A"), pd.Grouper(level="A")]).count()
```

Out[88]:

```
              B  C
Date       A
2019-12-31 1  1  1
2020-12-31 1  2  2
           2  2  1
2021-12-31 2  1  0
```

1.9 9. 时间序列

In [92]:
```
#!pip install quandl

import os
import pandas as pd
import matplotlib.pyplot as plt
import quandl

quandl.ApiConfig.api_key = os.environ.get("QUANDL_AUTH", "Dn6BtVoBhzuKTuyo6hbp")
start_date = "2014-05-01"

%matplotlib inline
```

In [94]:
```
christmas_str = "2019-12-25"
christmas = pd.to_datetime(christmas_str)
print("The type of christmas is", type(christmas))
christmas
```

```
The type of christmas is <class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

Out[94]:
```
Timestamp('2019-12-25 00:00:00')
```

In [95]:
```
for date in ["December 25, 2019", "Dec. 25, 2019",
             "Monday, Dec. 25, 2019", "25 Dec. 2019", "25th Dec. 2019"]:
    print("pandas interprets {} as {}".format(date, pd.to_datetime(date)))
```

```
pandas interprets December 25, 2019 as 2019-12-25 00:00:00
pandas interprets Dec. 25, 2019 as 2019-12-25 00:00:00
pandas interprets Monday, Dec. 25, 2019 as 2019-12-25 00:00:00
pandas interprets 25 Dec. 2019 as 2019-12-25 00:00:00
pandas interprets 25th Dec. 2019 as 2019-12-25 00:00:00
```

In [97]:
```
christmas_amzn = "2019-12-25T00:00:00+ 00 :00"
amzn_strftime = "%Y-%m-%dT%H:%M:%S+ 00 :00"
pd.to_datetime(christmas_amzn, format=amzn_strftime)
```

Out[97]:
```
Timestamp('2019-12-25 00:00:00')
```

```
In [98]: pd.to_datetime(["2017-12-25", "2017-12-31"])
```

```
Out[98]: DatetimeIndex(['2017-12-25', '2017-12-31'], dtype='datetime64[ns]', freq=None)
```

1.9.1 （1）提取数据

Here, we have the Bitcoin (BTC) to US dollar (USD) exchange rate from March 2014 until today.

```
In [100]: btc_usd = quandl.get("BCHARTS/BITSTAMPUSD", start_date=start_date)
          btc_usd.info()
          btc_usd.head()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2056 entries, 2014-05-01 to 2019-12-16
Data columns (total 7 columns):
Open                 2056 non-null float64
High                 2056 non-null float64
Low                  2056 non-null float64
Close                2056 non-null float64
Volume (BTC)         2056 non-null float64
Volume (Currency)    2056 non-null float64
Weighted Price       2056 non-null float64
dtypes: float64(7)
memory usage: 128.5 KB
```

```
Out[100]:               Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
          Date
          2014-05-01  449.00  465.88  447.97  460.97   9556.037886       4.381969e+06
          2014-05-02  460.97  462.99  444.51  454.50   8269.891417       3.731061e+06
          2014-05-03  452.00  454.50  431.00  439.00   7431.626480       3.271086e+06
          2014-05-04  439.00  442.83  429.55  438.04   5154.407794       2.245293e+06
          2014-05-05  435.88  445.00  425.00  433.00   8188.082795       3.547855e+06

                      Weighted Price
          Date
          2014-05-01      458.554960
          2014-05-02      451.162018
```

```
2014-05-03      440.157544
2014-05-04      435.606483
2014-05-05      433.294968
```

In [108]:
```
# 年
btc_usd.loc["2015"].head()
```

Out[108]:
```
              Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
Date
2015-01-01  321.00  321.00  312.60  313.81   3087.436554       9.745096e+05
2015-01-02  313.82  317.01  311.96  315.42   3468.281375       1.092446e+06
2015-01-03  315.42  316.58  280.00  282.00  21752.719146       6.475952e+06
2015-01-04  280.00  289.39  255.00  264.00  41441.278553       1.126676e+07
2015-01-05  264.55  280.00  264.07  276.80   9528.271002       2.596898e+06

            Weighted Price
Date
2015-01-01      315.637119
2015-01-02      314.981849
2015-01-03      297.707695
2015-01-04      271.872950
2015-01-05      272.546601
```

In [103]:
```
# 月
btc_usd.loc["August 2017"].head()
```

Out[103]:
```
               Open     High      Low    Close  Volume (BTC)  \
Date
2017-08-01  2855.81  2929.17  2615.00  2731.00  12525.076691
2017-08-02  2732.00  2760.00  2650.00  2703.51   9486.625526
2017-08-03  2703.51  2807.44  2698.83  2793.37   7963.697999
2017-08-04  2793.34  2877.52  2765.91  2855.00   7635.821672
2017-08-05  2851.01  3339.66  2848.32  3263.62  16996.273101

            Volume (Currency)  Weighted Price
Date
2017-08-01       3.432280e+07     2740.326259
2017-08-02       2.570111e+07     2709.193699
2017-08-03       2.193830e+07     2754.788542
```

```
2017-08-04       2.165009e+07     2835.331752
2017-08-05       5.386193e+07     3169.043337
```

In [104]:
```python
# 月
btc_usd.loc["08/2017"].head()
```

Out[104]:
```
               Open     High      Low    Close  Volume (BTC)  \
Date
2017-08-01  2855.81  2929.17  2615.00  2731.00  12525.076691
2017-08-02  2732.00  2760.00  2650.00  2703.51   9486.625526
2017-08-03  2703.51  2807.44  2698.83  2793.37   7963.697999
2017-08-04  2793.34  2877.52  2765.91  2855.00   7635.821672
2017-08-05  2851.01  3339.66  2848.32  3263.62  16996.273101

            Volume (Currency)  Weighted Price
Date
2017-08-01       3.432280e+07     2740.326259
2017-08-02       2.570111e+07     2709.193699
2017-08-03       2.193830e+07     2754.788542
2017-08-04       2.165009e+07     2835.331752
2017-08-05       5.386193e+07     3169.043337
```

In [105]:
```python
# 日
btc_usd.loc["August 1, 2017"]
```

Out[105]:
```
Open                 2.855810e+03
High                 2.929170e+03
Low                  2.615000e+03
Close                2.731000e+03
Volume (BTC)         1.252508e+04
Volume (Currency)    3.432280e+07
Weighted Price       2.740326e+03
Name: 2017-08-01 00:00:00, dtype: float64
```

In [106]:
```python
# 日
btc_usd.loc["08-01-2017"]
```

Out[106]:
```
Open                 2.855810e+03
High                 2.929170e+03
```

```
Low                  2.615000e+03
Close                2.731000e+03
Volume (BTC)         1.252508e+04
Volume (Currency)    3.432280e+07
Weighted Price       2.740326e+03
Name: 2017-08-01 00:00:00, dtype: float64
```

In [107]:
```
# 区段
btc_usd.loc["April 1, 2015":"April 10, 2015"]
```

Out[107]:
```
              Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
Date
2015-04-01  243.93  246.83  239.32  246.69   6226.016464       1.513601e+06
2015-04-02  246.68  256.96  244.52  253.28   9806.822203       2.453664e+06
2015-04-03  253.22  256.67  251.23  254.19   5048.577376       1.283171e+06
2015-04-04  254.19  255.85  250.76  253.70   2769.281658       7.002845e+05
2015-04-05  253.60  261.00  251.65  260.54   5759.360160       1.479483e+06
2015-04-06  260.57  262.98  254.00  255.58   5960.677633       1.535495e+06
2015-04-07  255.54  256.62  251.50  253.72   6010.267582       1.528034e+06
2015-04-08  253.71  254.96  243.06  244.58  11663.656155       2.878712e+06
2015-04-09  244.84  246.30  238.47  243.43   7943.710541       1.932558e+06
2015-04-10  243.75  243.94  231.00  235.99  11549.630656       2.728444e+06

            Weighted Price
Date
2015-04-01      243.109050
2015-04-02      250.199655
2015-04-03      254.164902
2015-04-04      252.875870
2015-04-05      256.883285
2015-04-06      257.604180
2015-04-07      254.237260
2015-04-08      246.810407
2015-04-09      243.281478
2015-04-10      236.236497
```

In [109]:
```
# 属性
btc_usd.index.year
```

```
Out[109]: Int64Index([2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014,
            ...
            2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019],
           dtype='int64', name='Date', length=2056)
```

```
In [110]: btc_usd.index.day
```

```
Out[110]: Int64Index([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
            ...
             7,  8,  9, 10, 11, 12, 13, 14, 15, 16],
           dtype='int64', name='Date', length=2056)
```

```
In [111]: # 重置索引
          btc_date_column = btc_usd.reset_index()
          btc_date_column.head()
```

```
Out[111]:         Date    Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
          0 2014-05-01  449.00  465.88  447.97  460.97   9556.037886       4.381969e+06
          1 2014-05-02  460.97  462.99  444.51  454.50   8269.891417       3.731061e+06
          2 2014-05-03  452.00  454.50  431.00  439.00   7431.626480       3.271086e+06
          3 2014-05-04  439.00  442.83  429.55  438.04   5154.407794       2.245293e+06
          4 2014-05-05  435.88  445.00  425.00  433.00   8188.082795       3.547855e+06

             Weighted Price
          0      458.554960
          1      451.162018
          2      440.157544
          3      435.606483
          4      433.294968
```

```
In [112]: btc_date_column["Date"].dt.year.head()
```

```
Out[112]: 0    2014
          1    2014
          2    2014
          3    2014
          4    2014
          Name: Date, dtype: int64
```

```
In [113]: btc_date_column["Date"].dt.month.head()
```

```
Out[113]: 0    5
          1    5
          2    5
          3    5
          4    5
          Name: Date, dtype: int64
```

1.9.2 （2）常用处理

滞后函数 shift

When doing time series analysis, we often want to compare data at one date against data at another date.

pandas can help us with this if we leverage the shift method.

Without any additional arguments, shift() will move all data forward one period, filling the first row with missing data.

```
In [114]: btc_usd.head()
```

```
Out[114]:               Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
          Date
          2014-05-01  449.00  465.88  447.97  460.97   9556.037886       4.381969e+06
          2014-05-02  460.97  462.99  444.51  454.50   8269.891417       3.731061e+06
          2014-05-03  452.00  454.50  431.00  439.00   7431.626480       3.271086e+06
          2014-05-04  439.00  442.83  429.55  438.04   5154.407794       2.245293e+06
          2014-05-05  435.88  445.00  425.00  433.00   8188.082795       3.547855e+06

                      Weighted Price
          Date
          2014-05-01      458.554960
          2014-05-02      451.162018
          2014-05-03      440.157544
          2014-05-04      435.606483
          2014-05-05      433.294968
```

```
In [115]: btc_usd.shift().head()
```

```
Out[115]:               Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
          Date
          2014-05-01     NaN     NaN     NaN     NaN           NaN                NaN
```

```
2014-05-02  449.00  465.88  447.97  460.97   9556.037886        4.381969e+06
2014-05-03  460.97  462.99  444.51  454.50   8269.891417        3.731061e+06
2014-05-04  452.00  454.50  431.00  439.00   7431.626480        3.271086e+06
2014-05-05  439.00  442.83  429.55  438.04   5154.407794        2.245293e+06

            Weighted Price
Date
2014-05-01             NaN
2014-05-02      458.554960
2014-05-03      451.162018
2014-05-04      440.157544
2014-05-05      435.606483
```

In [116]:
```python
((btc_usd - btc_usd.shift()) / btc_usd.shift()).head()
```

Out[116]:
```
                Open      High       Low     Close  Volume (BTC)  \
Date
2014-05-01       NaN       NaN       NaN       NaN           NaN
2014-05-02  0.026659 -0.006203 -0.007724 -0.014036     -0.134590
2014-05-03 -0.019459 -0.018337 -0.030393 -0.034103     -0.101363
2014-05-04 -0.028761 -0.025677 -0.003364 -0.002187     -0.306423
2014-05-05 -0.007107  0.004900 -0.010592 -0.011506      0.588559

            Volume (Currency)  Weighted Price
Date
2014-05-01                NaN             NaN
2014-05-02          -0.148542       -0.016122
2014-05-03          -0.123282       -0.024391
2014-05-04          -0.313594       -0.010340
2014-05-05           0.580130       -0.005306
```

In [117]:
```python
# 滞后三阶
btc_usd.shift(3).head()
```

Out[117]:
```
              Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
Date
2014-05-01     NaN     NaN     NaN     NaN           NaN                NaN
2014-05-02     NaN     NaN     NaN     NaN           NaN                NaN
2014-05-03     NaN     NaN     NaN     NaN           NaN                NaN
```

```
2014-05-04  449.00  465.88  447.97  460.97   9556.037886        4.381969e+06
2014-05-05  460.97  462.99  444.51  454.50   8269.891417        3.731061e+06

            Weighted Price
Date
2014-05-01             NaN
2014-05-02             NaN
2014-05-03             NaN
2014-05-04      458.554960
2014-05-05      451.162018
```

In [118]: `btc_usd.shift(-2).head()`

Out[118]:

```
              Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
Date
2014-05-01  452.00  454.50  431.00  439.00   7431.626480        3.271086e+06
2014-05-02  439.00  442.83  429.55  438.04   5154.407794        2.245293e+06
2014-05-03  435.88  445.00  425.00  433.00   8188.082795        3.547855e+06
2014-05-04  431.64  434.00  420.27  428.01   8041.198415        3.439331e+06
2014-05-05  429.00  452.00  425.67  440.00  13248.349023        5.842712e+06

            Weighted Price
Date
2014-05-01      440.157544
2014-05-02      435.606483
2014-05-03      433.294968
2014-05-04      427.713734
2014-05-05      441.014383
```

In [119]: `btc_usd.shift(-2).tail()`

Out[119]:

```
               Open     High      Low    Close  Volume (BTC)  \
Date
2019-12-12  7255.94  7269.00  7007.48  7059.03   2012.348906
2019-12-13  7066.35  7225.23  7007.00  7115.08   1848.345728
2019-12-14  7115.08  7115.08  7108.00  7108.00      0.502977
2019-12-15      NaN      NaN      NaN      NaN           NaN
2019-12-16      NaN      NaN      NaN      NaN           NaN
```

```
             Volume (Currency)  Weighted Price
Date
2019-12-12        1.433426e+07     7123.150732
2019-12-13        1.312936e+07     7103.305577
2019-12-14        3.578212e+03     7114.063353
2019-12-15                 NaN             NaN
2019-12-16                 NaN             NaN
```

滚动计算 rolling

In [120]:
```
btc_small = btc_usd.head(6)
btc_small
```

Out[120]:
```
              Open    High     Low   Close  Volume (BTC)  Volume (Currency)  \
Date
2014-05-01  449.00  465.88  447.97  460.97   9556.037886       4.381969e+06
2014-05-02  460.97  462.99  444.51  454.50   8269.891417       3.731061e+06
2014-05-03  452.00  454.50  431.00  439.00   7431.626480       3.271086e+06
2014-05-04  439.00  442.83  429.55  438.04   5154.407794       2.245293e+06
2014-05-05  435.88  445.00  425.00  433.00   8188.082795       3.547855e+06
2014-05-06  431.64  434.00  420.27  428.01   8041.198415       3.439331e+06

            Weighted Price
Date
2014-05-01      458.554960
2014-05-02      451.162018
2014-05-03      440.157544
2014-05-04      435.606483
2014-05-05      433.294968
2014-05-06      427.713734
```

In [121]:
```
#To compute the 2 day moving average (for all columns).
btc_small.rolling("2d").mean()
```

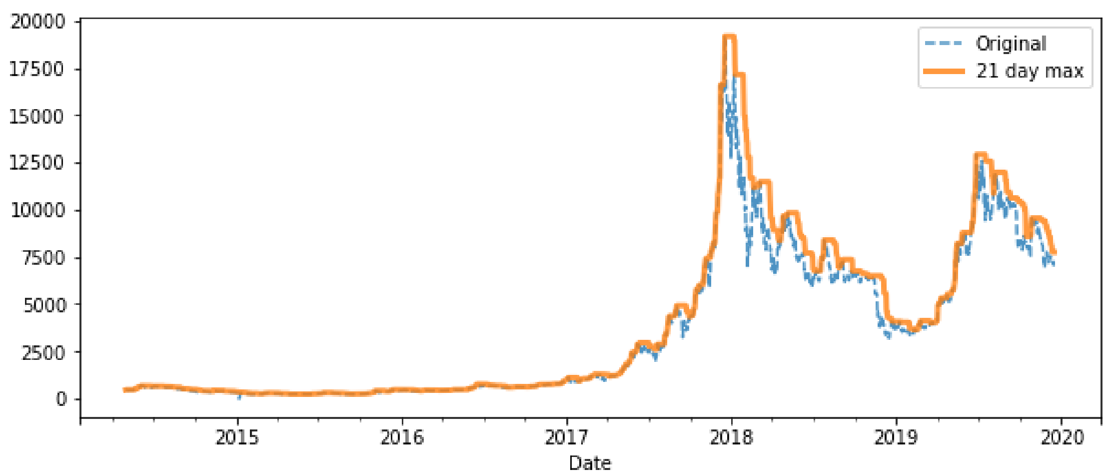
Out[121]:
```
               Open     High      Low    Close  Volume (BTC)  \
Date
2014-05-01  449.000  465.880  447.970  460.970   9556.037886
2014-05-02  454.985  464.435  446.240  457.735   8912.964652
2014-05-03  456.485  458.745  437.755  446.750   7850.758948
```

```
2014-05-04  445.500  448.665  430.275  438.520   6293.017137
2014-05-05  437.440  443.915  427.275  435.520   6671.245295
2014-05-06  433.760  439.500  422.635  430.505   8114.640605

            Volume (Currency)  Weighted Price
Date
2014-05-01       4.381969e+06      458.554960
2014-05-02       4.056515e+06      454.858489
2014-05-03       3.501074e+06      445.659781
2014-05-04       2.758190e+06      437.882013
2014-05-05       2.896574e+06      434.450725
2014-05-06       3.493593e+06      430.504351
```

In [122]:
```python
fig, ax = plt.subplots(figsize=(10, 4))
btc_usd["Open"].plot(ax=ax, linestyle="--", alpha=0.8)
btc_usd.rolling("21d").max()["Open"].plot(ax=ax, alpha=0.8, linewidth=3)
ax.legend(["Original", "21 day max"])
```

Out[122]: <matplotlib.legend.Legend at 0x1e664c82c18>



自定义函数

In [123]:
```python
def is_volatile(x):
    if x.var() > 1.0:
        return 1.0
```

```
        else:
            return 0.0
```

In [124]:
```
btc_small.rolling("2d").apply(is_volatile)
```

```
C:\Users\Van\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: Currently, 'ap
  """Entry point for launching an IPython kernel.
```

Out[124]:
```
            Open  High  Low  Close  Volume (BTC)  Volume (Currency)  \
Date
2014-05-01   0.0   0.0  0.0    0.0           0.0                0.0
2014-05-02   1.0   1.0  1.0    1.0           1.0                1.0
2014-05-03   1.0   1.0  1.0    1.0           1.0                1.0
2014-05-04   1.0   1.0  0.0    0.0           1.0                1.0
2014-05-05   1.0   1.0  1.0    1.0           1.0                1.0
2014-05-06   1.0   1.0  1.0    1.0           1.0                1.0

            Weighted Price
Date
2014-05-01             0.0
2014-05-02             1.0
2014-05-03             1.0
2014-05-04             1.0
2014-05-05             1.0
2014-05-06             1.0
```

改变频率

In [125]:
```
btc_usd.resample("BQ").mean()
```

Out[125]:
```
                  Open        High         Low       Close  \
Date
2014-06-30  546.296557  560.478689  534.375574  549.274426
2014-09-30  536.939130  544.464239  523.571087  533.646087
2014-12-31  358.113261  367.361413  347.553261  357.277935
2015-03-31  242.648111  250.373889  233.306667  241.823889
2015-06-30  236.023956  239.404396  232.456593  236.209451
2015-09-30  255.205435  259.369348  250.738478  254.964565
```

2015-12-31	343.968043	354.843587	335.019783	346.029674
2016-03-31	409.723956	415.345165	402.351319	409.552747
2016-06-30	508.830549	519.757143	491.237582	511.820440
2016-09-30	614.719891	621.399783	604.309022	614.092500
2016-12-30	722.921538	732.714176	714.873846	726.758462
2017-03-31	1031.095275	1058.529451	998.373077	1032.510000
2017-06-30	1886.373956	1961.381868	1823.321319	1902.486484
2017-09-29	3452.060000	3578.992637	3312.851758	3471.192088
2017-12-29	9132.594945	9602.524725	8655.917473	9245.341978
2018-03-30	10654.949560	11107.304725	9981.257363	10571.343077
2018-06-29	7760.852637	7950.978132	7543.325275	7754.252637
2018-09-28	6792.599011	6950.023956	6639.782857	6797.711209
2018-12-31	5199.605319	5297.541809	5063.396915	5169.268191
2019-03-29	3737.812500	3799.089886	3676.291136	3742.059432
2019-06-28	7026.032308	7308.565604	6829.843407	7117.250110
2019-09-30	10429.423617	10693.213511	10038.982872	10384.737660
2019-12-31	8153.517403	8338.461818	7953.086753	8136.907922

	Volume (BTC)	Volume (Currency)	Weighted Price
Date			
2014-06-30	9692.850513	5.418439e+06	547.779109
2014-09-30	8921.203260	4.473120e+06	533.775317
2014-12-31	14487.630863	5.182557e+06	357.025437
2015-03-31	15871.493002	3.827028e+06	242.062305
2015-06-30	7568.650463	1.787362e+06	236.050265
2015-09-30	15057.220448	3.752239e+06	255.216907
2015-12-31	21987.668969	7.534306e+06	345.209689
2016-03-31	7276.401365	2.957477e+06	409.345434
2016-06-30	5981.137422	3.332013e+06	509.388134
2016-09-30	3888.241125	2.364365e+06	613.211596
2016-12-30	4658.746976	3.501319e+06	724.799640
2017-03-31	9574.254814	9.767086e+06	1029.350854
2017-06-30	11808.708093	2.482040e+07	1897.089126
2017-09-29	14591.656328	5.001818e+07	3446.717092
2017-12-29	15339.474034	1.498917e+08	9150.932048
2018-03-30	16971.607663	1.711173e+08	10536.375918
2018-06-29	10412.011976	8.196025e+07	7744.048061

```
2018-09-28   7098.911451       4.863160e+07     6790.558832
2018-12-31   8805.848653       3.969870e+07     5178.241610
2019-03-29   6446.776373       2.414045e+07     3737.221237
2019-06-28  10796.082022       8.051510e+07     7077.940876
2019-09-30   9112.727636       9.513540e+07    10360.841564
2019-12-31   6722.605362       5.535318e+07     8136.590657
```

In [126]:
```python
btc_usd.resample("2BQS").agg(["min", "max"])
```

Out[126]:
```
                 Open               High                Low               Close  \
                  min       max      min       max      min       max      min
Date
2014-04-01     374.17    668.90   386.03    683.26   365.20    651.70   374.20
2014-10-01       0.00    426.64     0.00    453.92     0.00    390.48     0.00
2015-04-01     209.76    310.55   222.88    317.99   198.12    292.19   209.72
2015-10-01     235.87    464.53   239.06    502.00   235.00    453.50   237.15
2016-04-01     414.66    767.37   416.99    778.85     1.50    740.11   416.31
2016-10-03     607.19   1287.38   610.50   1350.00   604.99   1255.00   607.19
2017-04-03    1076.59   4921.71  1145.00   4979.90  1076.19   4671.09  1134.58
2017-10-02    4219.74  19187.78  4343.00  19666.00  4137.96  18465.00  4219.53
2018-04-02    5845.20   9827.04  6165.49   9948.98  5774.72   9670.68  5848.33
2018-10-01    3180.84   6604.76  3230.00   6756.00  3122.28   6553.13  3179.54
2019-04-01    4092.02  12927.44  4150.00  13880.00  4052.56  12030.43  4136.32
2019-10-01    6900.90   9547.32  7115.08  10350.00  6515.00   9254.68  6908.36

                      Volume (BTC)                Volume (Currency)  \
                  max          min            max               min
Date
2014-04-01     670.14  1467.591402   29732.720362      9.159133e+05
2014-10-01     426.63     0.000000  124188.885083      0.000000e+00
2015-04-01     310.55  1946.293030   42308.005630      4.732609e+05
2015-10-01     464.53  1253.006376  105959.259141      5.210775e+05
2016-04-01     766.62   719.159825   33056.289644      4.709121e+05
2016-10-03    1285.33   888.660021   36018.861120      5.460154e+05
2017-04-03    4921.70  1804.450797   60278.946542      2.128068e+06
2017-10-02   19187.78  4646.405621   70961.369658      2.032007e+07
2018-04-02    9823.28  1098.628060   33035.904045      7.093171e+06
2018-10-01    6604.75   839.297665   39775.389439      5.373482e+06
```

2019-04-01	12920.54	1572.155427	37487.802426	1.506857e+07
2019-10-01	9557.08	0.502977	38751.800255	3.578212e+03

		Weighted Price	
	max	min	max
Date			
2014-04-01	1.561239e+07	376.976877	667.690345
2014-10-01	2.357627e+07	0.000000	420.127183
2015-04-01	9.091325e+06	214.884260	306.748292
2015-10-01	4.719959e+07	237.116083	461.494358
2016-04-01	2.225764e+07	415.569853	754.723539
2016-10-03	3.883046e+07	607.560859	1275.581651
2017-04-03	2.031684e+08	1127.151197	4808.168193
2017-10-02	7.721430e+08	4233.863791	19110.244062
2018-04-02	3.032200e+08	5936.398196	9827.536792
2018-10-01	1.773528e+08	3171.722851	6593.879882
2019-04-01	4.769830e+08	4121.008519	12723.686028
2019-10-01	3.644311e+08	6911.863388	9504.401543