

Theodore Petrou

Pandas Cookbook

Recipes for Scientific Computing, Time Series Analysis
and Data Visualization using Python

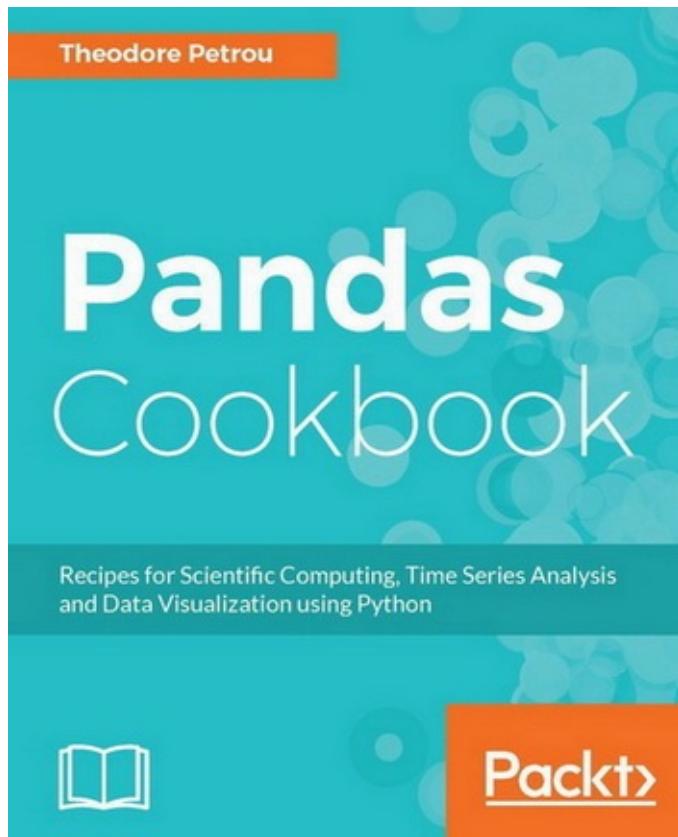


Packt

目錄

Pandas Cookbook 带注释源码	1.1
第01章 Pandas基础	1.2
第02章 DataFrame基本操作	1.3
第03章 数据分析入门	1.4
第04章 选取数据子集	1.5
第05章 布尔索引	1.6
第06章 索引对齐	1.7
第07章 分组聚合、过滤、转换	1.8
第08章 数据清理	1.9
第09章 合并Pandas对象	1.10
第10章 时间序列分析	1.11
第11章 用Matplotlib、Pandas、Seaborn进行可视化	1.12

Pandas Cookbook 带注释源码



贡献者 : SeanCheney

欢迎任何人参与和完善：一个人可以走的很快，但是一群人却可以走的更远。

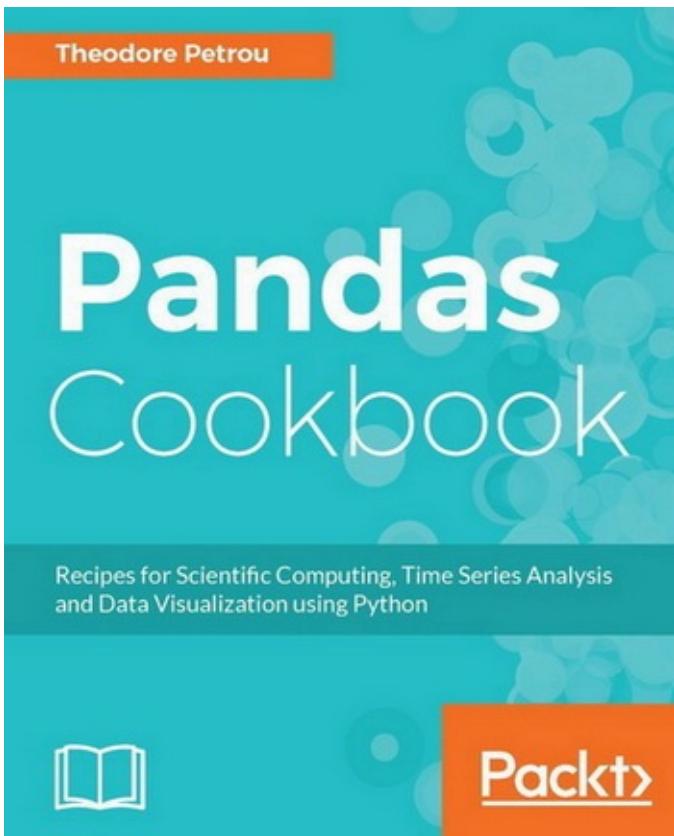
- ApacheCN 机器学习交流群 629470233
- ApacheCN 学习资源

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- 代码仓库

第01章 Pandas基础

相较于《利用Python进行数据分析》，本书最大的特点是所有操作都变成了分解动作，而且每步都有详细讲解。但是，书写的有点啰嗦，而Jupyter Notebook又有些错。我对两者做了整合和总结。

第一遍整理完，还有许多地方不足，还要再弄。





作者Theodore Petrou，Dunder Data

创始人

公司网址：<http://www.dunderdata.com> (dunder是蒸馏朗姆酒的残留液体，取这个名字是类比数据分析过程) GitHub地址：<https://github.com/tdpetrou> 领英个人页面：<https://www.linkedin.com/in/tedpetrou> 推特：<https://twitter.com/tedpetrou?lang=en> Medium博客：<https://medium.com/@petrou.theodore>

```
# 引入pandas和numpy的约定
in[1]: import pandas as pd
        import numpy as np
```

1. DataFrame的结构

```
# 设定最大列数和最大行数
in[2]: pd.set_option('max_columns', 8, 'max_rows', 10)
```

```
# 用read_csv()方法读取csv文件
# head()方法可以查看前五行，head(n)可以查看前n行
in[3]: movie = pd.read_csv('data/movie.csv')
        movie.head()
out[3]:
```

	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

5 rows × 28 columns

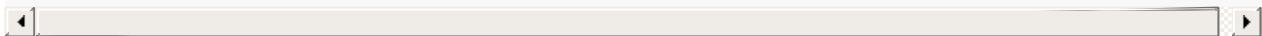
2. 访问DataFrame的组件

```
in[4]: # 提取列索引
        columns = movie.columns
        # 提取行索引
        index = movie.index
        # 提取数据
        data = movie.values
```

```
in[5]: columns
out[5]: Index(['color', 'director_name', 'num_critic_for_reviews',
       'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'act
or_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_nam
e',
       'movie_title', 'num_voted_users', 'cast_total_facebook_li
kes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'c
ountry',
       'content_rating', 'budget', 'title_year', 'actor_2_facebo
ok_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

```
in[6]: index
out[6]: RangeIndex(start=0, stop=4916, step=1)
```

```
in[7]: data
out[7]: array([['Color', 'James Cameron', 723.0, ..., 7.9, 1.78,
33000],
      ['Color', 'Gore Verbinski', 302.0, ..., 7.1, 2.35, 0],
      ['Color', 'Sam Mendes', 602.0, ..., 6.8, 2.35, 85000],
      ...,
      ['Color', 'Benjamin Roberds', 13.0, ..., 6.3, nan, 16],
      ['Color', 'Daniel Hsia', 14.0, ..., 6.3, 2.35, 660],
      ['Color', 'Jon Gunn', 43.0, ..., 6.6, 1.85, 456]], dtype=
object)
```



```
in[8]: # index的类型
        type(index) # pandas.core.indexes.range.RangeIndex
out[8]: pandas.core.indexes.range.RangeIndex
```

```
in[9]: # columns的类型
        type(columns) # pandas.core.indexes.base.Index
out[9]: pandas.core.indexes.base.Index
```

```
in[10]: # data的类型
        type(data) # numpy.ndarray
out[10]: numpy.ndarray
```

```
in[11]: # 判断是不是子类型
        issubclass(pd.RangeIndex, pd.Index) # True
out[11]: True
```

更多

```
in[12]: # 访问index的值
        index.values
        # index的值是个列表，所以可以索引或切片
        index.values[0]
out[12]: array([ 0,  1,  2, ..., 4913, 4914, 4915])
```

```

in[13]: # 访问columns的值
         columns.values
out[13]: array(['color', 'director_name', 'num_critic_for_reviews',
   'duration',
   'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
   'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
   'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
   'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
   'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
   'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
   'imdb_score', 'aspect_ratio', 'movie_facebook_likes'], dtype=object)

```

3. 理解数据类型

```

in[14]: movie = pd.read_csv('data/movie.csv')
         # 各列的类型
in[15]: movie.dtypes
out[15]: color                  object
         director_name          object
         num_critic_for_reviews float64
         duration                float64
         director_facebook_likes float64
                           ...
         title_year              float64
         actor_2_facebook_likes float64
         imdb_score              float64
         aspect_ratio             float64
         movie_facebook_likes     int64
Length: 28, dtype: object

```

```

in[16]: # 显示各类型的数量
         movie.get_dtype_counts()
out[16]: float64    13
         int64      3
         object     12
         dtype: int64

```

4. 选择一列数据，作为**Series**

```

in[17]: movie = pd.read_csv('data/movie.csv')
in[18]: # 选择director_name这列
         movie['director_name']
out[18]: 0      James Cameron
         1      Gore Verbinski
         2      Sam Mendes
         3      Christopher Nolan
         4      Doug Walker
         ...
         4911     Scott Smith
         4912     NaN
         4913     Benjamin Roberds
         4914     Daniel Hsia
         4915     Jon Gunn
Name: director_name, Length: 4916, dtype: object
in[19]: # 也可以通过属性的方式选取
         movie.director_name
out[19]: 0      James Cameron
         1      Gore Verbinski
         2      Sam Mendes
         3      Christopher Nolan
         4      Doug Walker
         ...
         4911     Scott Smith
         4912     NaN
         4913     Benjamin Roberds
         4914     Daniel Hsia
         4915     Jon Gunn
Name: director_name, Length: 4916, dtype: object
# 查看类型
in[20]: type(movie['director_name'])
out[20]: pandas.core.series.Series

```

更多

```

in[21]: director = movie['director_name']
         # 查看选取的列的名字
         director.name
out[21]: 'director_name'

```

```
in[22]: # 单列Series转换为DataFrame
    director.to_frame().head()
out[22]:
    director_name
0    James Cameron
1    Gore Verbinski
2    Sam Mendes
3    Christopher Nolan
4    Doug Walker
```

5. 调用 Series 方法

准备

```
in[23]: # 查看Series所有不重复的指令
    s_attr_methods = set(dir(pd.Series))
    # 该集合的大小
    len(s_attr_methods)
out[23]: 442
```

```
in[24]: # 查看DataFrame所有不重复的指令
    df_attr_methods = set(dir(pd.DataFrame))
    len(df_attr_methods)
out[24]: 445
```

```
in[25]: # 这两个集合中有多少共有的指令
    len(s_attr_methods & df_attr_methods)
out[25]: 376
```

原理

```
in[26]: # 选取director和actor_1_fb_likes两列
    movie = pd.read_csv('data/movie.csv')
    director = movie['director_name']
    actor_1_fb_likes = movie['actor_1_facebook_likes']
```

```
# 查看头部
in[27]: director.head()
out[27]: 0      James Cameron
          1      Gore Verbinski
          2      Sam Mendes
          3      Christopher Nolan
          4      Doug Walker
          Name: director_name, dtype: object
in[28]: actor_1_fb_likes.head()
out[28]: 0    1000.0
          1   40000.0
          2   11000.0
          3   27000.0
          4     131.0
          Name: actor_1_facebook_likes, dtype: float64
```

```
in[29]: # 分别计数
pd.set_option('max_rows', 8)
director.value_counts()
out[29]: Steven Spielberg    26
          Woody Allen        22
          Clint Eastwood      20
          Martin Scorsese     20
          .
          .
          James Nunn           1
          Gerard Johnstone    1
          Ethan Manquis        1
          Antony Hoffman       1
          Name: director_name, Length: 2397, dtype: int64
in[30]: actor_1_fb_likes.value_counts()
out[30]: 1000.0      436
          11000.0     206
          2000.0      189
          3000.0      150
          .
          .
          216.0        1
          859.0        1
          225.0        1
          334.0        1
          Name: actor_1_facebook_likes, Length: 877, dtype: int64
```

```
in[31]: director.size
out[31]: 4916
in[32]: director.shape
out[33]: (4916,)
in[33]: len(director)
out[33]: 4916
```

```
in[34]: # director有多少非空值
         director.count()
out[34]: 4814 # 说明有102个缺失值
```

```
in[35]: # actor_1_fb_likes有多少非空值
         actor_1_fb_likes.count()
out[35]: 4909
```

```
in[36]: # actor_1_fb_likes的中位分位数
         actor_1_fb_likes.quantile()
out[36]: 982.0
```

```
in[37]: # 求最小值、最大值、平均值、中位数、标准差、总和
         actor_1_fb_likes.min(), actor_1_fb_likes.max(), \
         actor_1_fb_likes.mean(), actor_1_fb_likes.median(), \
         actor_1_fb_likes.std(), actor_1_fb_likes.sum()
out[37]: (0.0, 640000.0, 6494.488490527602, 982.0, 15106.9868838
48309, 31881444.0)
```

```
in[38]: # 打印描述信息
         actor_1_fb_likes.describe()
out[38]: count      4909.000000
          mean      6494.488491
          std       15106.986884
          min       0.000000
          25%      607.000000
          50%      982.000000
          75%     11000.000000
          max     640000.000000
          Name: actor_1_facebook_likes, dtype: float64
in[39]: director.describe()
out[39]: count      4814
          unique    2397
          top       Steven Spielberg
          freq      26
          Name: director_name, dtype: object
```

```
in[40]: actor_1_fb_likes.quantile(.2)
out[41]: 510.0
```

```

In[41]: # 各个十分之一分位数
    actor_1_fb_likes.quantile([.1, .2, .3, .4, .5, .6, .7,
    .8, .9])
Out[41]: 0.1      240.0
          0.2      510.0
          0.3     694.0
          0.4     854.0
          ...
          0.6    1000.0
          0.7   8000.0
          0.8  13000.0
          0.9  18000.0
Name: actor_1_facebook_likes, Length: 9, dtype: float64

```

```

# 非空值
In[42]: director.isnull()
Out[42]: 0      False
          1      False
          2      False
          3      False
          ...
          4912     True
          4913    False
          4914    False
          4915    False
Name: director_name, Length: 4916, dtype: bool

```

```

# 填充缺失值
In[43]: actor_1_fb_likes_filled = actor_1_fb_likes.fillna(0)
          actor_1_fb_likes_filled.count()
Out[43]: 4916

```

```

# 删除缺失值
In[44]: actor_1_fb_likes_dropped = actor_1_fb_likes.dropna()
          actor_1_fb_likes_dropped.size
Out[44]: 4909

```

更多

```
# value_counts(normalize=True) 可以返回频率
In[45]: director.value_counts(normalize=True)
Out[45]: Steven Spielberg      0.005401
          Woody Allen         0.004570
          Clint Eastwood       0.004155
          Martin Scorsese      0.004155
          ...
          James Nunn            0.000208
          Gerard Johnstone     0.000208
          Ethan Maniquis        0.000208
          Antony Hoffman         0.000208
Name: director_name, Length: 2397, dtype: float64
```

```
# 判断是否有缺失值
In[46]: director.hasnans
Out[46]: True
```

```
# 判断是否是非缺失值
In[47]: director.notnull()
Out[47]: 0      True
          1      True
          2      True
          3      True
          ...
          4912    False
          4913    True
          4914    True
          4915    True
Name: director_name, Length: 4916, dtype: bool
```

6. 在Series上使用运算符

```
In[48]: pd.options.display.max_rows = 6
In[49]: 5 + 9      # 加法
Out[49]: 14
```

```
In[50]: 4 ** 2    # 幂运算
Out[50]: 16
```

```
In[51]: a = 10    # 赋值
```

```
In[52]: 5 <= 9 # 小于等于
Out[52]: True
```

```
In[53]: 'abcde' + 'fg' # 字符串拼接
Out[53]: 'abcdefg'
```

```
In[54]: not (5 <= 9) # 非运算符
Out[54]: False
```

```
In[55]: 7 in [1, 2, 6] # in运算符
Out[55]: False
```

```
In[56]: set([1, 2, 3]) & set([2, 3, 4]) # 求交集
Out[56]: {2, 3}
```

```
# 不支持列表和整数间的运算
In[57]: [1, 2, 3] - 3
-----
-----
TypeError                                     Traceback (most recent
    call last)
<ipython-input-57-7ca967348b32> in <module>()
----> 1 [1, 2, 3] - 3

TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

```
In[58]: a = set([1, 2, 3])
a[0]                                         # 集合不支持索引
```

准备

```
# 选取imdb_score这列
In[59]: movie = pd.read_csv('data/movie.csv')
          imdb_score = movie['imdb_score']
          imdb_score
Out[59]: 0      7.9
         1      7.1
         2      6.8
         ...
        4913    6.3
        4914    6.3
        4915    6.6
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 每列值加1
In[60]: imdb_score + 1
Out[60]: 0      8.9
         1      8.1
         2      7.8
         ...
        4913    7.3
        4914    7.3
        4915    7.6
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 每列值乘以2.5
In[61]: imdb_score * 2.5
Out[61]: 0      19.75
         1      17.75
         2      17.00
         ...
        4913    15.75
        4914    15.75
        4915    16.50
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 每列值除以7的余数
In[62]: imdb_score // 7
Out[62]: 0      1.0
         1      1.0
         2      0.0
         ...
        4913    0.0
        4914    0.0
        4915    0.0
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 判断是否大于7
In[63]: imdb_score > 7
Out[63]: 0      True
          1      True
          2     False
          ...
          4913    False
          4914    False
          4915    False
Name: imdb_score, Length: 4916, dtype: bool
```

```
# 判断是否等于字符串
In[64]: director = movie['director_name']
In[65]: director == 'James Cameron'
Out[65]: 0      True
          1     False
          2     False
          ...
          4913    False
          4914    False
          4915    False
Name: director_name, Length: 4916, dtype: bool
```

更多

```
# 利用通用函数实现加法
In[66]: imdb_score.add(1)                      # imdb_score + 1
Out[66]: 0      8.9
          1      8.1
          2      7.8
          ...
          4913    7.3
          4914    7.3
          4915    7.6
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 利用通用函数实现乘法
In[67]: imdb_score.mul(2.5)                                # imdb_score * 2.5
Out[67]: 0      19.75
          1      17.75
          2      17.00
          ...
          4913   15.75
          4914   15.75
          4915   16.50
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 利用通用函数实现底除
In[68]: imdb_score.floordiv(7)                            # imdb_score // 7
Out[68]: 0      1.0
          1      1.0
          2      0.0
          ...
          4913   0.0
          4914   0.0
          4915   0.0
Name: imdb_score, Length: 4916, dtype: float64
```

```
# 利用通用函数实现大于
In[69]: imdb_score.gt(7)                                 # imdb_score > 7
Out[69]: 0      True
          1      True
          2      False
          ...
          4913   False
          4914   False
          4915   False
Name: imdb_score, Length: 4916, dtype: bool
```

```
# 利用通用函数实现等于
In[70]: director.eq('James Cameron')    # director == 'James Cameron'
Out[70]: 0      True
          1      False
          2      False
          ...
          4913   False
          4914   False
          4915   False
Name: director_name, Length: 4916, dtype: bool
```

```
# 利用通用函数实现取模
In[71]: imdb_score.astype(int).mod(5)
Out[71]: 0      2
          1      2
          2      1
          ..
          4913    1
          4914    1
          4915    1
Name: imdb_score, Length: 4916, dtype: int64
```

```
# a是int对象
In[72]: a = type(1)
In[73]: type(a)
Out[73]: type
```

```
# a是pandas.core.series.Series对象
In[74]: a = type(imdb_score)
In[75]: a([1,2,3])
Out[75]: 0    1
          1    2
          2    3
          dtype: int64
```

7. 串联 Series 方法

```
# value_counts().head(3)，计数，查看前三
In[76]: movie = pd.read_csv('data/movie.csv')
         actor_1_fb_likes = movie['actor_1_facebook_likes']
         director = movie['director_name']
In[77]: director.value_counts().head(3)
Out[77]: Steven Spielberg    26
          Woody Allen        22
          Clint Eastwood      20
          Name: director_name, dtype: int64
```

```
# 统计缺失值的数量
In[78]: actor_1_fb_likes.isnull().sum()
Out[78]: 7
```

```
# actor_1_fb_likes的数据类型
In[79]: actor_1_fb_likes.dtype
Out[79]: dtype('float64')
```

```
# 缺失值填充为0、转换为整型、查看前五
In[80]: actor_1_fb_likes.fillna(0)\n        .astype(int)\n        .head()
Out[80]: 0      1000\n         1     40000\n         2    11000\n         3   27000\n         4     131\nName: actor_1_facebook_likes, dtype: int64
```

更多

```
# 缺失值的比例
In[81]: actor_1_fb_likes.isnull().mean()
Out[81]: 0.0014239218877135883
```

```
# 使用括号串联
In[82]: (actor_1_fb_likes.fillna(0)\n        .astype(int)\n        .head())
Out[82]: 0      1000\n         1     40000\n         2    11000\n         3   27000\n         4     131\nName: actor_1_facebook_likes, dtype: int64
```

8. 使索引有意义

```
# set_index()给行索引命名
In[83]: movie = pd.read_csv('data/movie.csv')
In[84]: movie.shape
Out[84]: (4916, 28)
In[85]: movie2 = movie.set_index('movie_title')
          movie2
Out[85]:
```

	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
movie_title									
Avatar	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
Spectre	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
...
A Plague So Pleasant	Color	Benjamin Roberds	13.0	76.0	...	0.0	6.3	NaN	16
Shanghai Calling	Color	Daniel Hsia	14.0	100.0	...	719.0	6.3	2.35	660
My Date with Drew	Color	Jon Gunn	43.0	90.0	...	23.0	6.6	1.85	456

4916 rows × 27 columns

通过index_col参数命名

In[86]: pd.read_csv('data/movie.csv', index_col='movie_title')
Out[86]:

	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
movie_title									
Avatar	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
Spectre	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
...
A Plague So Pleasant	Color	Benjamin Roberds	13.0	76.0	...	0.0	6.3	NaN	16
Shanghai Calling	Color	Daniel Hsia	14.0	100.0	...	719.0	6.3	2.35	660
My Date with Drew	Color	Jon Gunn	43.0	90.0	...	23.0	6.6	1.85	456

4916 rows × 27 columns

更多

复原索引

In[87]: movie2.reset_index()

9. 重命名行名和列名

通过rename()重命名

In[88]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')

In[89]: idx_rename = {'Avatar': 'Ratava', 'Spectre': 'Ertceps'}
col_rename = {'director_name': 'Director Name',
'num_critic_for_reviews': 'Critical Revie
ws'}In[90]: movie.rename(index=idx_rename,
columns=col_rename).head()

Out[90]:

	color	Director Name	Critical Reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
movie_title									
Ratava	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
Ertceps	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

5 rows × 27 columns

更多

```
In[91]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
index = movie.index
columns = movie.columns

index_list = index.tolist()
column_list = columns.tolist()

index_list[0] = 'Ratava'
index_list[2] = 'Ertceps'
column_list[1] = 'Director Name'
column_list[2] = 'Critical Reviews'

In[92]: print(index_list[:5])
['Ratava', "Pirates of the Caribbean: At World's End",
'Ertceps', 'The Dark Knight Rises', 'Star Wars: Episode VII - The Force Awakens']

In[93]: print(column_list)
['color', 'Director Name', 'Critical Reviews', 'duration',
'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
'num_voted_users', 'cast_total_facebook_likes', 'actor_3_name',
'facenumber_in_poster', 'plot_keywords', 'movie_imdb_link',
'num_user_for_reviews', 'language', 'country', 'content_rating',
'budget', 'title_year', 'actor_2_facebook_likes', 'imdb_score',
'aspect_ratio', 'movie_facebook_likes']

In[94]: movie.index = index_list
movie.columns = column_list

In[95]: movie.head()
Out[95]:
```

	color	Director Name	Critical Reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
Ratava	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
Ertceps	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

5 rows × 27 columns

10. 创建、删除列

```
# 通过[列名]添加新列
In[96]: movie = pd.read_csv('data/movie.csv')
In[97]: movie['has_seen'] = 0
In[98]: movie.columns
Out[98]: Index(['color', 'director_name', 'num_critic_for_reviews',
   'duration',
              'director_facebook_likes', 'actor_3_facebook_likes',
   'actor_2_name',
              'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
              'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
   'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
              'movie_imdb_link', 'num_user_for_reviews', 'language',
   'country',
              'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
   'imdb_score', 'aspect_ratio', 'movie_facebook_likes', 'has_seen'],
   dtype='object')
```

```
# 给新列赋值
In[99]: movie['actor_director_facebook_likes'] = (movie['actor_1_facebook_likes'] +
                                                       movie['actor_2_facebook_likes'] +
                                                       movie['actor_3_facebook_likes'] +
                                                       movie['director_facebook_likes'])
In[100]: movie['actor_director_facebook_likes'].isnull().sum()
Out[100]: 122
```

```
# 用all()检查是否所有的布尔值都为True
In[101]: movie['actor_director_facebook_likes'] = movie['actor_
director_facebook_likes'].fillna(0)
In[102]: movie['is_cast_likes_more'] = (movie['cast_total_faceb
ook_likes'] >=
                                         movie['actor_director_facebook
_likes'])
In[103]: movie['is_cast_likes_more'].all()
Out[103]: False

In[104]: movie = movie.drop('actor_director_facebook_likes', ax
is='columns')
In[105]: movie['actor_total_facebook_likes'] = (movie['actor_1_
facebook_likes'] +
                                         movie['actor_2_facebook_li
kes'] +
                                         movie['actor_3_facebook_li
kes'])

movie['actor_total_facebook_likes'] = movie['actor_tot
al_facebook_likes'].fillna(0)
In[106]: movie['is_cast_likes_more'] = movie['cast_total_facebo
ok_likes'] >=
                                         movie['actor_total_facebook_li
kes']

movie['is_cast_likes_more'].all()
Out[106]: True
```

```
In[107]: movie['pct_actor_cast_like'] = (movie['actor_total_fac
ebook_likes'] /
                                         movie['cast_total_facebook_likes
'])
In[108]: movie['pct_actor_cast_like'].min(), movie['pct_actor_c
ast_like'].max()
Out[108]: (0.0, 1.0)

In[109]: movie.set_index('movie_title')['pct_actor_cast_like'].head()
Out[109]: movie_title
          Avatar                  0.577369
          Pirates of the Caribbean: At World's End      0.951396
          Spectre                  0.987521
          The Dark Knight Rises                  0.683783
          Star Wars: Episode VII - The Force Awakens      0.000000
          Name: pct_actor_cast_like, dtype: float64
```

更多

```
# 用insert()方法原地插入列
In[110]: profit_index = movie.columns.get_loc('gross') + 1
          profit_index
In[111]: movie.insert(loc=profit_index,
                      column='profit',
                      value=movie['gross'] - movie['budget'])
In[112]: movie.head()
Out[112]:
```

	color	director_name	num_critic_for_reviews	duration	...	has_seen	is_cast_likes_more	actor_total_facebook_likes	pct_actor_cast_like
0	Color	James Cameron	723.0	178.0	...	0	True	2791.0	0.577369
1	Color	Gore Verbinski	302.0	169.0	...	0	True	46000.0	0.951396
2	Color	Sam Mendes	602.0	148.0	...	0	True	11554.0	0.987521
3	Color	Christopher Nolan	813.0	164.0	...	0	True	73000.0	0.683783
4	NaN	Doug Walker	NaN	NaN	...	0	True	0.0	0.000000

5 rows × 33 columns

第02章 DataFrame基本操作

```
In[1]: import pandas as pd
import numpy as np
pd.options.display.max_columns = 40
```

1. 选取多个DataFrame列

```
# 用列表选取多个列
In[2]: movie = pd.read_csv('data/movie.csv')
        movie_actor_director = movie[['actor_1_name', 'actor_2_name',
                                         'actor_3_name', 'director_name']]
        movie_actor_director.head()
Out[2]:
```

	actor_1_name	actor_2_name	actor_3_name	director_name
0	CCH Pounder	Joel David Moore	Wes Studi	James Cameron
1	Johnny Depp	Orlando Bloom	Jack Davenport	Gore Verbinski
2	Christoph Waltz	Rory Kinnear	Stephanie Sigman	Sam Mendes
3	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	Christopher Nolan
4	Doug Walker	Rob Walker	NaN	Doug Walker

```
# 选取单列
In[3]: movie[['director_name']].head()
Out[3]:
```

director_name

0 James Cameron

1 Gore Verbinski

2 Sam Mendes

3 Christopher Nolan

4 Doug Walker

```
# 错误的选取多列的方式
In[4]: movie['actor_1_name', 'actor_2_name', 'actor_3_name', 'director_name']
-----
-----
KeyError                                     Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
    2441             try:
-> 2442                 return self._engine.get_loc(key)
    2443             except KeyError:
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5280)()
```

```

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20523)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20477)()

KeyError: ('actor_1_name', 'actor_2_name', 'actor_3_name', 'director_name')

During handling of the above exception, another exception occurred:

KeyError Traceback (most recent
call last)
<ipython-input-4-954222273e42> in <module>()
----> 1 movie['actor_1_name', 'actor_2_name', 'actor_3_name', 'director_name']

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/frame.py in __getitem__(self, key)
    1962         return self._getitem_multilevel(key)
    1963     else:
-> 1964         return self._getitem_column(key)
    1965
    1966     def _getitem_column(self, key):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/frame.py in _getitem_column(self, key)
    1969             # get column
    1970             if self.columns.is_unique:
-> 1971                 return self._get_item_cache(key)
    1972
    1973             # duplicate columns & possible reduce dimensionality

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/generic.py in _get_item_cache(self, item)
    1643             res = cache.get(item)
    1644             if res is None:
-> 1645                 values = self._data.get(item)
    1646                 res = self._box_item_values(item, values)
    1647                 cache[item] = res

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/internals.py in get(self, item, fastpath)
    3588
    3589             if not isnull(item):
-> 3590                 loc = self.items.get_loc(item)
    3591             else:
    3592                 indexer = np.arange(len(self.items))[isn

```

```

ull(self.items)]

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
    2442                 return self._engine.get_loc(key)
    2443             except KeyError:
-> 2444                 return self._engine.get_loc(self._maybe_
cast_indexer(key))
    2445
    2446         indexer = self.get_indexer([key], method=method,
tolerance=tolerance)

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5280)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20523)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20477)()

KeyError: ('actor_1_name', 'actor_2_name', 'actor_3_name', 'director_name')

```

更多

```

# 将列表赋值给一个变量，便于多选
In[6]: cols =['actor_1_name', 'actor_2_name', 'actor_3_name', 'director_name']
        movie_actor_director = movie[cols]
Out[6]: float64    13
          int64      3
          object     11
          dtype: int64

```

```

# 使用select_dtypes()，选取整数列
In[7]: movie.select_dtypes(include=[ 'int']).head()
Out[7]:

```

		num_voted_users	cast_total_facebook_likes	movie_facebook_likes
	movie_title			
	Avatar	886204	4834	33000
	Pirates of the Caribbean: At World's End	471220	48350	0
	Spectre	275868	11700	85000
	The Dark Knight Rises	1144337	106759	164000
	Star Wars: Episode VII - The Force Awakens	8	143	0

选取所有的数值列

In[8]: movie.select_dtypes(include=['number']).head()

Out[8]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_total_fb
movie_title								
Avatar	723.0	178.0	0.0	855.0	1000.0	760505847.0	886204	
Pirates of the Caribbean: At World's End	302.0	169.0	563.0	1000.0	40000.0	309404152.0	471220	
Spectre	602.0	148.0	0.0	161.0	11000.0	200074175.0	275868	
The Dark Knight Rises	813.0	164.0	22000.0	23000.0	27000.0	448130642.0	1144337	
Star Wars: Episode VII - The Force Awakens	Nan	Nan	131.0	Nan	131.0	Nan	8	

通过filter()函数过滤选取多列

In[9]: movie.filter(like='facebook').head()

Out[9]:

	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	cast_total_facebook_likes	actor_2_facebook_likes	movie_facebook_likes
movie_title						
Avatar	0.0	855.0	1000.0	4834	936.0	33000
Pirates of the Caribbean: At World's End	563.0	1000.0	40000.0	48350	5000.0	0
Spectre	0.0	161.0	11000.0	11700	393.0	85000
The Dark Knight Rises	22000.0	23000.0	27000.0	106759	23000.0	164000
Star Wars: Episode VII - The Force Awakens	131.0	Nan	131.0	143	12.0	0

通过正则表达式选取多列

In[10]: movie.filter(regex='\d').head()

Out[10]:

movie_title	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	actor_1_name	actor_3_name	actor_2_facebook_likes
Avatar	855.0	Joel David Moore	1000.0	CCH Pounder	Wes Studi	936.0
Pirates of the Caribbean: At World's End	1000.0	Orlando Bloom	40000.0	Johnny Depp	Jack Davenport	5000.0
Spectre	161.0	Rory Kinnear	11000.0	Christoph Waltz	Stephanie Sigman	393.0
The Dark Knight Rises	23000.0	Christian Bale	27000.0	Tom Hardy	Joseph Gordon-Levitt	23000.0
Star Wars: Episode VII - The Force Awakens	NaN	Rob Walker	131.0	Doug Walker	NaN	12.0

```
# filter()函数，传递列表到参数items，选取多列
```

```
In[11]: movie.filter(items=['actor_1_name', 'asdf']).head()
Out[11]:
```

actor_1_name
movie_title
Avatar
CCH Pounder
Pirates of the Caribbean: At World's End
Johnny Depp
Spectre
Christoph Waltz
The Dark Knight Rises
Tom Hardy
Star Wars: Episode VII - The Force Awakens
Doug Walker

2. 对列名进行排序

```
# 读取movie数据集
In[12]: movie = pd.read_csv('data/movie.csv')
In[13]: movie.head()
Out[13]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross	Ac
0	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	1000.0	760505847.0	A
1	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	40000.0	309404152.0	
2	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	11000.0	200074175.0	
3	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	27000.0	448130642.0	
4	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	131.0	NaN	

打印列索引

```
In[14]: movie.columns
Out[14]: Index(['color', 'director_name', 'num_critic_for_reviews',
       'duration',
       'director_facebook_likes', 'actor_3_facebook_likes', 'actor_2_name',
       'actor_1_facebook_likes', 'gross', 'genres', 'actor_1_name',
       'movie_title', 'num_voted_users', 'cast_total_facebook_likes',
       'actor_3_name', 'facenumber_in_poster', 'plot_keywords',
       'movie_imdb_link', 'num_user_for_reviews', 'language', 'country',
       'content_rating', 'budget', 'title_year', 'actor_2_facebook_likes',
       'imdb_score', 'aspect_ratio', 'movie_facebook_likes'],
      dtype='object')
```

```
# 将列索引按照指定的顺序排列
In[15]: disc_core = ['movie_title', 'title_year', 'content_rating', 'genres']
         disc_people = ['director_name', 'actor_1_name', 'actor_2_name', 'actor_3_name']
         disc_other = ['color', 'country', 'language', 'plot_keywords', 'movie_imdb_link']
         cont_fb = ['director_facebook_likes', 'actor_1_facebook_likes', 'actor_2_facebook_likes', 'actor_3_facebook_likes', 'cast_total_facebook_likes', 'movie_facebook_likes']
         cont_finance = ['budget', 'gross']
         cont_num_reviews = ['num_voted_users', 'num_user_for_reviews', 'num_critic_for_reviews']
         cont_other = ['imdb_score', 'duration', 'aspect_ratio', 'facenumber_in_poster']

In[16]: new_col_order = disc_core + disc_people + disc_other +
         cont_fb + cont_finance + cont_num_reviews +
         cont_other
         set(movie.columns) == set(new_col_order)
Out[16]: True

In[17]: movie2 = movie[new_col_order]
         movie2.head()
Out[17]:
```

	movie_title	title_year	content_rating	genres	director_name	actor_1_name	actor_2_name	actor_3_name	color	country	language
0	Avatar	2009.0	PG-13	Action Adventure Fantasy Sci-Fi	James Cameron	CCH Pounder	Joel David Moore	Wes Studi	Color	USA	English
1	Pirates of the Caribbean: At World's End	2007.0	PG-13	Action Adventure Fantasy	Gore Verbinski	Johnny Depp	Orlando Bloom	Jack Davenport	Color	USA	English
2	Spectre	2015.0	PG-13	Action Adventure Thriller	Sam Mendes	Christoph Waltz	Rory Kinnear	Stephanie Sigman	Color	UK	English
3	The Dark Knight Rises	2012.0	PG-13	Action Thriller	Christopher Nolan	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	Color	USA	English
4	Star Wars: Episode VII - The Force Awakens	NaN	NaN	Documentary	Doug Walker	Doug Walker	Rob Walker	NaN	NaN	NaN	dece

3. 在整个DataFrame上操作

```
In[18]: pd.options.display.max_rows = 8
         movie = pd.read_csv('data/movie.csv')
         # 打印行数和列数
         movie.shape
Out[18]: (4916, 28)
```

```
# 打印数据的个数
In[19]: movie.size
Out[19]: 137648
```

```
# 该数据集的维度
In[20]: movie.ndim
Out[20]: 2
```

```
# 该数据集的长度
In[21]: len(movie)
Out[21]: 4916
```

```
# 各个列的值的个数
In[22]: movie.count()
Out[22]:
color           4897
director_name   4814
num_critic_for_reviews 4867
duration        4901
...
actor_2_facebook_likes 4903
imdb_score      4916
aspect_ratio    4590
movie_facebook_likes 4916
Length: 28, dtype: int64
```

```
# 各列的最小值
In[23]: movie.min()
Out[23]:
num_critic_for_reviews  1.00
duration                7.00
director_facebook_likes 0.00
actor_3_facebook_likes  0.00
...
actor_2_facebook_likes  0.00
imdb_score               1.60
aspect_ratio              1.18
movie_facebook_likes     0.00
Length: 16, dtype: float64
```

```
# 打印描述信息
In[24]: movie.describe()
Out[24]:
```

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_total_fa
count	4867.000000	4901.000000	4814.000000	4893.000000	4909.000000	4.054000e+03	4.916000e+03	
mean	137.988905	107.090798	691.014541	631.276313	6494.488491	4.764451e+07	8.264492e+04	
std	120.239379	25.286015	2832.954125	1625.874802	15106.986884	6.737255e+07	1.383222e+05	
min	1.000000	7.000000	0.000000	0.000000	0.000000	1.620000e+02	5.000000e+00	
25%	49.000000	93.000000	7.000000	132.000000	607.000000	5.019656e+06	8.361750e+03	
50%	108.000000	103.000000	48.000000	366.000000	982.000000	2.504396e+07	3.313250e+04	
75%	191.000000	118.000000	189.750000	633.000000	11000.000000	6.110841e+07	9.377275e+04	
max	813.000000	511.000000	23000.000000	23000.000000	640000.000000	7.605058e+08	1.689764e+06	6

使用percentiles参数指定分位数

In[25]: pd.options.display.max_rows = 10

In[26]: movie.describe(percentiles=[.01, .3, .99])

Out[26]:

	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_1_facebook_likes	gross	num_voted_users	cast_total_fa
count	4867.000000	4901.000000	4814.000000	4893.000000	4909.000000	4.054000e+03	4.916000e+03	
mean	137.988905	107.090798	691.014541	631.276313	6494.488491	4.764451e+07	8.264492e+04	
std	120.239379	25.286015	2832.954125	1625.874802	15106.986884	6.737255e+07	1.383222e+05	
min	1.000000	7.000000	0.000000	0.000000	0.000000	1.620000e+02	5.000000e+00	
1%	2.000000	43.000000	0.000000	0.000000	6.080000	8.474800e+03	5.300000e+01	
30%	60.000000	95.000000	11.000000	176.000000	694.000000	7.914069e+06	1.186450e+04	
50%	108.000000	103.000000	48.000000	366.000000	982.000000	2.504396e+07	3.313250e+04	
99%	546.680000	189.000000	16000.000000	11000.000000	44920.000000	3.264128e+08	6.815846e+05	
max	813.000000	511.000000	23000.000000	23000.000000	640000.000000	7.605058e+08	1.689764e+06	6

打印各列空值的个数

In[27]: pd.options.display.max_rows = 8

In[28]: movie.isnull().sum()

Out[28]: color 19
 director_name 102
 num_critic_for_reviews 49
 duration 15
 ...
 actor_2_facebook_likes 13
 imdb_score 0
 aspect_ratio 326
 movie_facebook_likes 0
 Length: 28, dtype: int64

更多

```
# 设定skipna=False，没有缺失值的数值列才会计算结果
In[29]: movie.min(skipna=False)
Out[29]: num_critic_for_reviews      NaN
          duration                  NaN
          director_facebook_likes   NaN
          actor_3_facebook_likes   NaN
          ...
          actor_2_facebook_likes   NaN
          imdb_score                1.6
          aspect_ratio               NaN
          movie_facebook_likes      0.0
Length: 16, dtype: float64
```

4. 串联DataFrame方法

```
# 使用isnull方法将每个值转变为布尔值
In[30]: movie = pd.read_csv('data/movie.csv')
          movie.isnull().head()
Out[30]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes	gross	genres
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	True	False	True	True	False	True	False	False	True	False

```
# 使用sum统计布尔值，返回的是Series
In[31]: movie.isnull().sum().head()
Out[31]: color                      19
          director_name              102
          num_critic_for_reviews     49
          duration                   15
          director_facebook_likes  102
dtype: int64
```

```
# 对这个Series再使用sum，返回整个DataFrame的缺失值的个数，返回值是个标量
In[32]: movie.isnull().sum().sum()
Out[32]: 2654
```

```
# 判断整个DataFrame有没有缺失值，方法是连着使用两个any
In[33]: movie.isnull().any().any()
Out[33]: True
```

原理

```
# isnull返回同样大小的DataFrame，但所有的值变为布尔值
In[34]: movie.isnull().get_dtype_counts()
Out[34]: bool    28
          dtype: int64
```

更多

```
# movie数据集的对象数据包含缺失值。默认条件下，聚合方法min、max、sum，不会
返回任何值。
In[35]: movie[['color', 'movie_title', 'color']].max()
Out[35]: Series([], dtype: float64)
```

```
# 要让pandas强行返回每列的值，必须填入缺失值。下面填入的是空字符串：
In[36]: movie.select_dtypes(['object']).fillna('').max()
Out[36]:
color
      Color
      director_name
Étienne Faure
      actor_2_name
Zubaida Sahar
      genres
      Western

      ...
      movie_imdb_link [http://www.imdb.com/title/tt5574490
/?ref_=fn_t...](http://www.imdb.com/title/tt5574490/?ref_=fn_t..
.)
      language
      Zulu
      country
West Germany
      content_rating
      X
Length: 12, dtype: object</pre>
```

5. 在DataFrame上使用运算符

```
# college数据集的值既有数值也有对象，整数5不能与字符串相加
In[37]: college = pd.read_csv('data/college.csv')
college + 5
-----
```

```

TypeError                                         Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in na_op(x, y)
    1175         result = expressions.evaluate(op, str_rep, x
, y,
-> 1176                                         raise_on_error
=True, **eval_kwargs)
    1177     except TypeError:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/com-
putation/expressions.py in evaluate(op, op_str, a, b, raise_on_erro-
r, use_numexpr, **eval_kwargs)
    210         return _evaluate(op, op_str, a, b, raise_on_erro-
r=raise_on_error,
--> 211                                         **eval_kwargs)
    212     return _evaluate_standard(op, op_str, a, b, raise_on_
error=raise_on_error)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/com-
putation/expressions.py in _evaluate_numexpr(op, op_str, a, b, ra-
ise_on_error, truediv, reversed, **eval_kwargs)
    121     if result is None:
--> 122         result = _evaluate_standard(op, op_str, a, b, ra-
ise_on_error)
    123

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/com-
putation/expressions.py in _evaluate_standard(op, op_str, a, b, r-
aise_on_error, **eval_kwargs)
    63     with np.errstate(all='ignore'):
--> 64         return op(a, b)
    65

TypeError: must be str, not int

During handling of the above exception, another exception occurred:

TypeError                                         Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte-
rnals.py in eval(self, func, other, raise_on_error, try_cast, mg-
r)
    1183         with np.errstate(all='ignore'):
-> 1184             result = get_result(other)
    1185

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte-
rnals.py in get_result(other)
    1152             else:
-> 1153                 result = func(values, other)
    1154

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.py in na_op(x, y)
    1201                     with np.errstate(all='ignore'):
-> 1202                         result[mask] = op(xrav, y)
    1203                     else:
TypeError: must be str, not int

During handling of the above exception, another exception occurred:

TypeError                                     Traceback (most recent
call last)
<ipython-input-37-4749f68a2501> in <module>()
      1 college = pd.read_csv('data/college.csv')
----> 2 college + 5

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.py in f(self, other, axis, level, fill_value)
    1239             self = self.fillna(fill_value)
    1240
-> 1241             return self._combine_const(other, na_op)
    1242
    1243         f.__name__ = name

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/frame.py in _combine_const(self, other, func, raise_on_error)
    3541     def _combine_const(self, other, func, raise_on_error
=True):
    3542         new_data = self._data.eval(func=func, other=other,
r,
-> 3543                                         raise_on_error=raise_
on_error)
    3544         return self._constructor(new_data)
    3545

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/internals.py in eval(self, **kwargs)
    3195
    3196     def eval(self, **kwargs):
-> 3197         return self.apply('eval', **kwargs)
    3198
    3199     def quantile(self, **kwargs):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/internals.py in apply(self, f, axes, filter, do_integrity_check, consolidate, **kwargs)
    3089
    3090         kwargs['mgr'] = self
-> 3091         applied = getattr(b, f)(**kwargs)
    3092         result_blocks = _extend_blocks(applied, resu
lt_blocks)

```

3093

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in eval(self, func, other, raise_on_error, try_cast, mg
r)
    1189         raise
    1190     except Exception as detail:
-> 1191         result = handle_error()
    1192
    1193     # technically a broadcast error in numpy can 'wo
rk' by returning a

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in handle_error()
    1172             # The 'detail' variable is defined in ou
ter scope.
    1173             raise TypeError('Could not operate %s wi
th block values %s' %
-> 1174                                     (repr(other), str(detail
))) # noqa
    1175         else:
    1176             # return the values

TypeError: Could not operate 5 with block values must be str, no
t int

```

```

# 行索引名设为INSTNM，用UGDS_过滤出本科生的种族比例
In[38]: college = pd.read_csv('data/college.csv', index_col='INS
TNM')
        college_ugds_ = college.filter(like='UGDS_')
In[39]: college == 'asdf' # 这是jn上的，想要比较college和'asdf'，没
有意义，忽略
-----
-----
TypeError                                         Traceback (most recent
call last)
<ipython-input-39-697c8af60bcf> in <module>()
----> 1 college == 'asdf'

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in f(self, other)
    1302             # straight boolean comparisions we want to a
llow all columns
    1303             # (regardless of dtype to pass thru) See #45
37 for discussion.
-> 1304             res = self._combine_const(other, func, raise
_on_error=False)
    1305             return res.fillna(True).astype(bool)
    1306

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram

```

```

e.py in _combine_const(self, other, func, raise_on_error)
  3541      def _combine_const(self, other, func, raise_on_error
=True):
  3542          new_data = self._data.eval(func=func, other=other,
r,
-> 3543                                      raise_on_error=raise_
on_error)
  3544      return self._constructor(new_data)
  3545

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in eval(self, **kwargs)
  3195
  3196      def eval(self, **kwargs):
-> 3197          return self.apply('eval', **kwargs)
  3198
  3199      def quantile(self, **kwargs):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in apply(self, f, axes, filter, do_integrity_check, con
solidate, **kwargs)
  3089
  3090          kwargs['mgr'] = self
-> 3091          applied = getattr(b, f)(**kwargs)
  3092          result_blocks = _extend_blocks(applied, resu
lt_blocks)
  3093

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in eval(self, func, other, raise_on_error, try_cast, mg
r)
  1203
  1204          raise TypeError('Could not compare [%s]
with block values' %
-> 1205                                      repr(other))
  1206
  1207          # transpose if needed

TypeError: Could not compare ['asdf'] with block values

```

```

# 查看前5行
In[40]: college_ugds_.head()
Out[40]:

```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000
University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172
Alabama State	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098

现在都是均质数据了，可以进行数值运算

In[41]: college_ugds_.head() + .00501

Out[41]:

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR
Alabama A & M University	0.03831	0.94031	0.01051	0.00691	0.00741	0.00691	0.0000
University of Alabama at Birmingham	0.59721	0.26501	0.03331	0.05681	0.00721	0.00571	0.0368
Amridge University	0.30401	0.42421	0.01191	0.00841	0.00501	0.00501	0.0000
University of Alabama in Huntsville	0.70381	0.13051	0.04321	0.04261	0.01931	0.00521	0.0172
Alabama State University	0.02081	0.92581	0.01711	0.00691	0.00601	0.00561	0.0098

用底除计算百分比分数

In[42]: (college_ugds_.head() + .00501) // .01

Out[42]:

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	3.0	94.0	1.0	0.0	0.0	0.0	0.0
University of Alabama at Birmingham	59.0	26.0	3.0	5.0	0.0	0.0	0.0
Amridge University	30.0	42.0	1.0	0.0	0.0	0.0	0.0
University of Alabama in Huntsville	70.0	13.0	4.0	4.0	1.0	0.0	0.0
Alabama State University	2.0	92.0	1.0	0.0	0.0	0.0	0.0

```
# 再除以100
In[43]: college_ugds_op_round = (college_ugds_ + .00501) // .01
          / 100
          college_ugds_op_round.head()
Out[43]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	0.03	0.94	0.01	0.00	0.00	0.0	0.0
University of Alabama at Birmingham	0.59	0.26	0.03	0.05	0.00	0.0	0.0
Amridge University	0.30	0.42	0.01	0.00	0.00	0.0	0.0
University of Alabama in Huntsville	0.70	0.13	0.04	0.04	0.01	0.0	0.0
Alabama State University	0.02	0.92	0.01	0.00	0.00	0.0	0.0

```
# 保留两位小数
```

```
In[44]: college_ugds_round = (college_ugds_ + .00001).round(2)
college_ugds_round.head()
```

```
Out[44]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	0.03	0.94	0.01	0.00	0.00	0.0	
University of Alabama at Birmingham	0.59	0.26	0.03	0.05	0.00	0.0	
Amridge University	0.30	0.42	0.01	0.00	0.00	0.0	
University of Alabama in Huntsville	0.70	0.13	0.04	0.04	0.01	0.0	
Alabama State University	0.02	0.92	0.01	0.00	0.00	0.0	

< >

```
In[45]: .045 + .005
```

```
Out[45]: 0.04999999999999996
```

```
In[46]: college_ugds_op_round.equals(college_ugds_round)
Out[46]: True
```

更多

```
# DataFrame的通用函数也可以实现上述方法
```

```
In[47]: college_ugds_op_round_methods = college_ugds_.add(.00501
).floordiv(.01).div(100)
```

< >

6. 比较缺失值

```
# Pandas使用NumPy NaN (np.nan) 对象表示缺失值。这是一个不等于自身的特殊对象：  
In[48]: np.nan == np.nan  
Out[48]: False
```

```
# Python的None对象是等于自身的  
In[49]: None == None  
Out[49]: True
```

```
# 所有和np.nan的比较都返回False，除了不等于：  
In[50]: 5 > np.nan  
Out[50]: False  
  
In[51]: np.nan > 5  
Out[51]: False  
  
In[52]: 5 != np.nan  
Out[52]: True
```

```
# college_ugds_所有值和.0019比较，返回布尔值DataFrame  
In[53]: college = pd.read_csv('data/college.csv', index_col='INSTNM')  
        college_ugds_ = college.filter(like='UGDS_')  
In[54]: college_ugds_.head() == .0019  
Out[54]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	False	False	False	True	False	True	
University of Alabama at Birmingham	False	False	False	False	False	False	
Amridge University	False	False	False	False	False	False	
University of Alabama in Huntsville	False	False	False	False	False	False	
Alabama State University	False	False	False	True	False	False	

```
# 用DataFrame和DataFrame进行比较
In[55]: college_self_compare = college_ugds_ == college_ugds_
college_self_compare.head()
Out[55]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA	UGDS_UNKN
Alabama A & M University	True	True	True	True	True	True	True	True	True
University of Alabama at Birmingham	True	True	True	True	True	True	True	True	True
Amridge University	True	True	True	True	True	True	True	True	True
University of Alabama in Huntsville	True	True	True	True	True	True	True	True	True
Alabama State	True	True	True	True	True	True	True	True	True

用all()检查是否所有的值都是True；这是因为缺失值不互相等于。

```
In[56]: college_self_compare.all()
Out[56]: UGDS_WHITE      False
          UGDS_BLACK     False
          UGDS_HISP      False
          UGDS_ASIAN     False
          ...
          UGDS_NHPI      False
          UGDS_2MOR       False
          UGDS_NRA        False
          UGDS_UNKN      False
Length: 9, dtype: bool
```

```
# 可以用==号判断，然后求和
In[57]: (college_ugds_ == np.nan).sum()
Out[57]: UGDS_WHITE      0
          UGDS_BLACK      0
          UGDS_HISP        0
          UGDS_ASIAN       0
          ...
          UGDS_NHPI        0
          UGDS_2MOR         0
          UGDS_NRA         0
          UGDS_UNKN        0
Length: 9, dtype: int64
```

```
# 统计缺失值最主要方法是使用isnull方法：
In[58]: college_ugds_.isnull().sum()
Out[58]: UGDS_WHITE      661
          UGDS_BLACK      661
          UGDS_HISP        661
          UGDS_ASIAN       661
          ...
          UGDS_NHPI        661
          UGDS_2MOR         661
          UGDS_NRA         661
          UGDS_UNKN        661
Length: 9, dtype: int64
```

```
# 比较两个DataFrame最直接的方法是使用equals()方法
In[59]: from pandas.testing import assert_frame_equal
In[60]: assert_frame_equal(college_ugds_, college_ugds_)
Out[60]: True
```

更多

```
# eq()方法类似于==，和前面的equals有所不同
In[61]: college_ugds_.eq(.0019).head()
Out[61]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	False	False	False	True	False	True	
University of Alabama at Birmingham	False	False	False	False	False	False	
Amridge University	False	False	False	False	False	False	
University of Alabama in Huntsville	False	False	False	False	False	False	
Alabama State University	False	False	False	True	False	False	

7. 矩阵转置

```
In[62]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
college_ugds_ = college.filter(like='UGDS_')
college_ugds_.head()
Out[62]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	
University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	

```
# count()返回非缺失值的个数
In[63]: college_ugds_.count()
Out[63]: UGDS_WHITE      6874
          UGDS_BLACK     6874
          UGDS_HISP       6874
          UGDS_ASIAN      6874
          ...
          UGDS_NHPI        6874
          UGDS_2MOR        6874
          UGDS_NRA         6874
          UGDS_UNKN        6874
Length: 9, dtype: int64
```

```
# axis默认设为0
In[64]: college_ugds_.count(axis=0)
Out[64]: UGDS_WHITE      6874
          UGDS_BLACK     6874
          UGDS_HISP       6874
          UGDS_ASIAN      6874
          ...
          UGDS_NHPI        6874
          UGDS_2MOR        6874
          UGDS_NRA         6874
          UGDS_UNKN        6874
Length: 9, dtype: int64
```

```
# 等价于axis='index'
In[65]: college_ugds_.count(axis='index')
Out[65]: UGDS_WHITE      6874
          UGDS_BLACK     6874
          UGDS_HISP       6874
          UGDS_ASIAN      6874
          ...
          UGDS_NHPI        6874
          UGDS_2MOR        6874
          UGDS_NRA         6874
          UGDS_UNKN        6874
Length: 9, dtype: int64
```

```
# 统计每行的非缺失值个数
In[66]: college_ugds_.count(axis='columns').head()
Out[66]: INSTNM
          Alabama A & M University      9
          University of Alabama at Birmingham  9
          Amridge University      9
          University of Alabama in Huntsville  9
          Alabama State University      9
dtype: int64
```

```
# 除了统计每行的非缺失值个数，也可以求和加以确认
In[67]: college_ugds_.sum(axis='columns').head()
Out[67]: INSTNM
          Alabama A & M University      1.0000
          University of Alabama at Birmingham  0.9999
          Amridge University      1.0000
          University of Alabama in Huntsville  1.0000
          Alabama State University      1.0000
dtype: float64
```

```
# 用中位数了解每列的分布
In[68]: college_ugds_.median(axis='index')
Out[68]: UGDS_WHITE      0.55570
          UGDS_BLACK      0.10005
          UGDS_HISP       0.07140
          UGDS_ASIAN      0.01290
          ...
          UGDS_NHPI       0.00000
          UGDS_2MOR       0.01750
          UGDS_NRA        0.00000
          UGDS_UNKN       0.01430
Length: 9, dtype: float64
```

更多

```
# 使用累积求和cumsum()可以很容易看到白人、黑人、西班牙裔的比例
In[69]: college_ugds_cumsum = college_ugds_.cumsum(axis=1)
college_ugds_cumsum.head()
Out[69]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	0.0333	0.9686	0.9741	0.9760	0.9784	0.9803	
University of Alabama at Birmingham	0.5922	0.8522	0.8805	0.9323	0.9345	0.9352	
Amridge University	0.2990	0.7182	0.7251	0.7285	0.7285	0.7285	
University of Alabama in Huntsville	0.6988	0.8243	0.8625	0.9001	0.9144	0.9146	
Alabama State University	0.0158	0.9366	0.9487	0.9506	0.9516	0.9522	

< [] >

```
# UGDS_HISP一列降序排列
In[70]: college_ugds_cumsum.sort_values('UGDS_HISP', ascending=False)
Out[70]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_AMER	UGDS_AFRICAN
New Beginning College of Cosmetology	0.8957	0.9305	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001
Virginia University of Lynchburg	0.0120	0.9921	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001
Turning Point Beauty College	0.1915	0.2341	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001
First Coast Barber Academy	0.1667	0.9445	1.0001	1.0001	1.0001	1.0001	1.0001	1.0001
...
Rasmussen College - Overland Park	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
National Personal Training Institute of Cleveland	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Bay Area Medical Academy - San Jose Satellite Location	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Excel Learning Center-San Antonio South	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

7535 rows × 9 columns

8. 确定大学校园多样性

```
# US News给出的美国10所最具多样性的大学
In[71]: pd.read_csv('data/college_diversity.csv', index_col='School')
Out[71]:
```

School	Diversity Index
Rutgers University--Newark Newark, NJ	0.76
Andrews University Berrien Springs, MI	0.74
Stanford University Stanford, CA	0.74
University of Houston Houston, TX	0.74
...	...
San Francisco State University San Francisco, CA	0.73
University of Illinois--Chicago Chicago, IL	0.73
New Jersey Institute of Technology Newark, NJ	0.72
Texas Woman's University Denton, TX	0.72

10 rows × 1 columns

```
In[72]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
college_ugds_ = college.filter(like='UGDS_')
college_ugds_.head()
Out[72]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	
University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	

< >

```
In[73]: college_ugds_.isnull().sum(axis=1).sort_values(ascending=False).head()
Out[73]: INSTNM
Excel Learning Center-San Antonio South          9
Philadelphia College of Osteopathic Medicine      9
Assemblies of God Theological Seminary          9
Episcopal Divinity School                      9
Phillips Graduate Institute                    9
dtype: int64
```

如果所有列都是缺失值，则将其去除

```
In[74]: college_ugds_ = college_ugds_.dropna(how='all')
In[75]: college_ugds_.isnull().sum()
Out[75]: UGDS_WHITE      0
          UGDS_BLACK     0
          UGDS_HISP       0
          UGDS_ASIAN      0
          ...
          UGDS_NHPI       0
          UGDS_2MOR        0
          UGDS_NRA         0
          UGDS_UNKN        0
Length: 9, dtype: int64
```

用大于或等于方法ge()，将DataFrame变为布尔值矩阵

```
In[76]: college_ugds_.ge(.15).head()
Out[76]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	U
Alabama A & M University	False	True	False	False	False	False	False
University of Alabama at Birmingham	True	True	False	False	False	False	False
Amridge University	True	True	False	False	False	False	False
University of Alabama in Huntsville	True	False	False	False	False	False	False
Alabama State University	False	True	False	False	False	False	False

< >

```
# 对所有True值求和
In[77]: diversity_metric = college_ugds_.ge(.15).sum(axis='columns')
          diversity_metric.head()
Out[77]: INSTNM
          Alabama A & M University    1
          University of Alabama at Birmingham  2
          Amridge University      3
          University of Alabama in Huntsville  1
          Alabama State University    1
          dtype: int64
```

```
# 使用value_counts()，查看分布情况
In[78]: diversity_metric.value_counts()
Out[78]: 1    3042
         2    2884
         3     876
         4      63
         0       7
         5       2
         dtype: int64
```

```
# 查看哪些学校种族比例超过15%的数量多
In[79]: diversity_metric.sort_values(ascending=False).head()
Out[79]: INSTNM
Regency Beauty Institute-Austin      5
Central Texas Beauty College-Temple   5
Sullivan and Coglian Training Center 4
Ambria College of Nursing             4
Berkeley College-New York            4
dtype: int64
```

```
# 用loc()方法查看对应行索引的行
In[80]: college_ugds_.loc[['Regency Beauty Institute-Austin',
                           'Central Texas Beauty College-Temple']]
Out[80]:
```

	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGD
INSTNM							
Regency Beauty Institute- Austin	0.1867	0.2133	0.1600	0.0000	0.0	0.0	
Central Texas Beauty College- Temple	0.1616	0.2323	0.2626	0.0202	0.0	0.0	

< >

```
# 查看US News前五所最具多样性的大学在diversity_metric中的情况
In[81]: us_news_top = ['Rutgers University-Newark',
                      'Andrews University',
                      'Stanford University',
                      'University of Houston',
                      'University of Nevada-Las Vegas']
In[82]: diversity_metric.loc[us_news_top]
Out[82]: INSTNM
Rutgers University-Newark      4
Andrews University              3
Stanford University             3
University of Houston           3
University of Nevada-Las Vegas 3
dtype: int64
```

更多

```
# 可以用最大种群比例查看哪些学校最不具有多样性
In[83]: college_ugds_.max(axis=1).sort_values(ascending=False).
head(10)
Out[83]: INSTNM
Dewey University-Manati                               1
.0
Yeshiva and Kollel Harbotzas Torah                  1
.0
Mr Leon's School of Hair Design-Lewiston             1
.0
Dewey University-Bayamon                             1
.0
.
.
Monteclaro Escuela de Hoteleria y Artes Culinarias 1
.0
Yeshiva Shaar Hatorah                            1
.0
Bais Medrash Elyon                                1
.0
Yeshiva of Nitra Rabbinical College                1
.0
Length: 10, dtype: float64
```

```
# 查看Talmudical Seminary Oholei Torah哲学学校
In[84]: college_ugds_.loc['Talmudical Seminary Oholei Torah']
Out[84]: UGDS_WHITE      1.0
UGDS_BLACK      0.0
UGDS_HISP       0.0
UGDS_ASIAN      0.0
.
.
UGDS_NHPI       0.0
UGDS_2MOR       0.0
UGDS_NRA        0.0
UGDS_UNKN      0.0
Name: Talmudical Seminary Oholei Torah, Length: 9, dtype
e: float64
```

```
# 查看是否有学校九个种族的比例都超过了1%
In[85]: (college_ugds_ > .01).all(axis=1).any()
Out[85]: True
```

第03章 数据分析入门

```
In[1]: import pandas as pd
import numpy as np
from IPython.display import display
pd.options.display.max_columns = 50
```

1. 规划数据分析路线

```
# 读取查看数据
In[2]: college = pd.read_csv('data/college.csv')
In[3]: college.head()
Out[3]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATI
0	Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	
2	Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	
3	University of Alabama in Huntsville	Huntsville	AL	0.0	0.0	0.0	0	595.0	
4	Alabama State University	Montgomery	AL	1.0	0.0	0.0	0	425.0	

```
# 数据的行数与列数
In[4]: college.shape
Out[4]: (7535, 27)
```

```
# 统计数值列，并进行转置
In[5]: with pd.option_context('display.max_rows', 8):
         display(college.describe(include=[np.number]).T)
Out[5]:
```

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.0000	0.000000	0.000000	1.0
MENONLY	7164.0	0.009213	0.095546	0.0	0.0000	0.000000	0.000000	1.0
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.0000	0.000000	0.000000	1.0
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.0000	0.000000	0.000000	1.0
...
CURROPER	7535.0	0.923291	0.266146	0.0	1.0000	1.000000	1.000000	1.0
PCTPELL	6849.0	0.530643	0.225544	0.0	0.3578	0.52150	0.712900	1.0
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.3329	0.58330	0.745000	1.0
UG25ABV	6718.0	0.410021	0.228939	0.0	0.2415	0.40075	0.572275	1.0

22 rows × 8 columns

统计对象和类型列

```
In[6]: college.describe(include=[np.object, pd.Categorical]).T
Out[6]:
```

	count	unique		top	freq
INSTNM	7535	7535	San Francisco State University	1	
CITY	7535	2514	New York	87	
STABBR	7535	59	CA	773	
MD_EARN_WNE_P10	6413	598	PrivacySuppressed	822	
GRAD_DEBT_MDN_SUPP	7503	2038	PrivacySuppressed	1510	

```
# 列出每列的数据类型，非缺失值的数量，以及内存的使用
In[7]: college.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7535 entries, 0 to 7534
Data columns (total 27 columns):
INSTNM                7535 non-null object
CITY                  7535 non-null object
STABBR                7535 non-null object
HBCU                  7164 non-null float64
MENONLY               7164 non-null float64
WOMENONLY              7164 non-null float64
RELAFFIL               7535 non-null int64
SATVRMID              1185 non-null float64
SATMTMID              1196 non-null float64
DISTANCEONLY           7164 non-null float64
UGDS                  6874 non-null float64
UGDS_WHITE              6874 non-null float64
UGDS_BLACK              6874 non-null float64
UGDS_HISP                6874 non-null float64
UGDS_ASIAN                6874 non-null float64
UGDS_AIAN                6874 non-null float64
UGDS_NHPI                6874 non-null float64
UGDS_2MOR                 6874 non-null float64
UGDS_NRA                  6874 non-null float64
UGDS_UNKN                 6874 non-null float64
PPTUG_EF                  6853 non-null float64
CURROPER                7535 non-null int64
PCTPELL                  6849 non-null float64
PCTFLOAN                  6849 non-null float64
UG25ABV                  6718 non-null float64
MD_EARN_WNE_P10            6413 non-null object
GRAD_DEBT_MDN_SUPP          7503 non-null object
dtypes: float64(20), int64(2), object(5)
memory usage: 1.6+ MB
```

```
# 重复了，但没设置最大行数
In[8]: college.describe(include=[np.number]).T
Out[8]:
```

		count	mean	std	min	25%	50%	75%
HBCU	7164.0	0.014238	0.118478	0.0	0.000000	0.000000	0.000000	
MENONLY	7164.0	0.009213	0.095546	0.0	0.000000	0.000000	0.000000	
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.000000	0.000000	0.000000	
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.000000	0.000000	0.000000	
SATVRMID	1185.0	522.819409	68.578862	290.0	475.000000	510.000000	555.000000	71
SATMTMID	1196.0	530.765050	73.469767	310.0	482.000000	520.000000	565.000000	71
DISTANCEONLY	7164.0	0.005583	0.074519	0.0	0.000000	0.000000	0.000000	
UGDS	6874.0	2356.837940	5474.275871	0.0	117.000000	412.500000	1929.500000	1515
UGDS_WHITE	6874.0	0.510207	0.286958	0.0	0.267500	0.55570	0.747875	
UGDS_BLACK	6874.0	0.189997	0.224587	0.0	0.036125	0.10005	0.257700	
UGDS_HISP	6874.0	0.161635	0.221854	0.0	0.027600	0.07140	0.198875	
UGDS_ASIAN	6874.0	0.033544	0.073777	0.0	0.002500	0.01290	0.032700	
UGDS_AIAN	6874.0	0.013813	0.070196	0.0	0.000000	0.00260	0.007300	
UGDS_NHPI	6874.0	0.004569	0.033125	0.0	0.000000	0.00000	0.002500	
UGDS_2MOR	6874.0	0.023950	0.031288	0.0	0.000000	0.01750	0.033900	
UGDS_NRA	6874.0	0.016086	0.050172	0.0	0.000000	0.00000	0.011700	
UGDS_UNKN	6874.0	0.045181	0.093440	0.0	0.000000	0.01430	0.045400	
PPTUG_EF	6853.0	0.226639	0.246470	0.0	0.000000	0.15040	0.376900	
CURROPER	7535.0	0.923291	0.266146	0.0	1.000000	1.00000	1.000000	
PCTPELL	6849.0	0.530643	0.225544	0.0	0.357800	0.52150	0.712900	
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.332900	0.58330	0.745000	
UG25ABV	6718.0	0.410021	0.228939	0.0	0.241500	0.40075	0.572275	

和前面重复了

```
In[9]: college.describe(include=[np.object, pd.Categorical]).T
Out[9]:
```

	count	unique		top	freq
INSTNM	7535	7535	San Francisco State University	1	
CITY	7535	2514		New York	87
STABBR	7535	59		CA	773
MD_EARN_WNE_P10	6413	598	PrivacySuppressed	822	
GRAD_DEBT_MDN_SUPP	7503	2038	PrivacySuppressed	1510	

更多

```
# 在describe方法中，打印分位数
In[10]: with pd.option_context('display.max_rows', 5):
    display(college.describe(include=[np.number],
                           percentiles=[.01, .05, .10, .25, .5, .75, .9,
                           .95, .99]).T)
```

	count	mean	std	min	1%	5%	10%	25%	50%	75%	90%	95%	99%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.0000	0.0000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000	1.000000	1.
MENONLY	7164.0	0.009213	0.095546	0.0	0.0000	0.0000	0.0000	0.0000	0.000000	0.000000	0.000000	0.000000	0.000000	1.
...
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.0000	0.0000	0.0000	0.3329	0.58330	0.745000	0.84752	0.89792	0.986368	1.
UG25ABV	6718.0	0.410021	0.228939	0.0	0.0025	0.0374	0.0899	0.2415	0.40075	0.572275	0.72666	0.80000	0.917383	1.

22 rows × 14 columns

< >

```
# 展示一个数据字典：数据字典的主要作用是解释列名的意义
In[11]: college_dd = pd.read_csv('data/college_data_dictionary.csv')
In[12]: with pd.option_context('display.max_rows', 8):
    display(college_dd)
```

	column_name	description
0	INSTNM	Institution Name
1	CITY	City Location
2	STABBR	State Abbreviation
3	HBCU	Historically Black College or University
...
23	PCTFLOAN	Percent Students with federal loan
24	UG25ABV	Percent Students Older than 25
25	MD_EARN_WNE_P10	Median Earnings 10 years after enrollment
26	GRAD_DEBT_MDN_SUPP	Median debt of completers

27 rows × 2 columns

2. 改变数据类型，降低内存消耗

```
# 选取五列
In[13]: college = pd.read_csv('data/college.csv')
different_cols = ['RELAFFIL', 'SATMTMID', 'CURROPER',
INSTNM', 'STABBR']
col2 = college.loc[:, different_cols]
col2.head()
Out[13]:
```

	RELAFFIL	SATMTMID	CURROPER	INSTNM	STABBR
0	0	420.0	1	Alabama A & M University	AL
1	0	565.0	1	University of Alabama at Birmingham	AL
2	1	NaN	1	Amridge University	AL
3	0	590.0	1	University of Alabama in Huntsville	AL
4	0	430.0	1	Alabama State University	AL

```
# 查看数据类型
In[14]: col2.dtypes
Out[14]: RELAFFIL      int64
          SATMTMID     float64
          CURROPER      int64
          INSTNM        object
          STABBR        object
          dtype: object
```

```
# 用memory_usage方法查看每列的内存消耗
In[15]: original_mem = col2.memory_usage(deep=True)
original_mem
Out[15]: Index      80
          RELAFFIL  60280
          SATMTMID  60280
          CURROPER  60280
          INSTNM   660240
          STABBR    444565
          dtype: int64
```

```
# RELAFFIL这列只包含0或1，因此没必要用64位，使用astype方法将其变为8位（1字节）整数
```

```
In[16]: col2['RELAFFIL'] = col2['RELAFFIL'].astype(np.int8)
# 再次查看数据类型
```

```
In[17]: col2.dtypes
Out[17]: RELAFFIL      int8
          SATMTMID    float64
          CURROPER     int64
          INSTNM       object
          STABBR        object
          dtype: object
```

```
# 检查两个对象列的独立值的个数
```

```
In[18]: col2.select_dtypes(include=['object']).nunique()
Out[18]: INSTNM    7535
          STABBR     59
          dtype: int64
```

```
# STABBR列可以转变为“类型”（Categorical），独立值的个数小于总数的1%
```

```
In[19]: col2['STABBR'] = col2['STABBR'].astype('category')
col2.dtypes
Out[19]: RELAFFIL      int8
          SATMTMID    float64
          CURROPER     int64
          INSTNM       object
          STABBR       category
          dtype: object
```

```
# 再次检查内存的使用
```

```
In[20]: new_mem = col2.memory_usage(deep=True)
new_mem
Out[20]: Index      80
          RELAFFIL  7535
          SATMTMID  60280
          CURROPER  60280
          INSTNM   660699
          STABBR    13576
          dtype: int64
```

```
# 通过和原始数据比较，RELAFFIL列变为了原来的八分之一，STABBR列只有原始大小的3%
In[21]: new_mem / original_mem
Out[21]: Index      1.000000
          RELAFFIL  0.125000
          SATMTMID  1.000000
          CURROPER  1.000000
          INSTNM    1.000695
          STABBR    0.030538
          dtype: float64
```

更多

```
# CURROPER和INSTNM分别是int64和对象类型
In[22]: college = pd.read_csv('data/college.csv')
In[23]: college[['CURROPER', 'INSTNM']].memory_usage(deep=True)
Out[23]: Index      80
          CURROPER  60280
          INSTNM    660240
          dtype: int64
```

```
# CURROPER列加上了100000000，但是内存使用没有变化；但是INSTNM列加上了一个a，内存消耗增加了105字节
In[24]: college.loc[0, 'CURROPER'] = 100000000
         college.loc[0, 'INSTNM'] = college.loc[0, 'INSTNM'] + 'a'
         # college.loc[1, 'INSTNM'] = college.loc[1, 'INSTNM'] + 'a'
         college[['CURROPER', 'INSTNM']].memory_usage(deep=True)
Out[24]: Index      80
          CURROPER  60280
          INSTNM    660345
          dtype: int64
```

```
# 数据字典中的信息显示MENONLY这列只包含0和1，但是由于含有缺失值，它的类型是浮点型
In[25]: college['MENONLY'].dtype
Out[25]: dtype('float64')
```

```
# 任何数值类型的列，只要有一个缺失值，就会成为浮点型；这列中的任何整数都会强制成为浮点型
In[26]: college['MENONLY'].astype('int8') # ValueError: Cannot
convert non-finite values (NA or inf) to integer
-----
```

```

-----
ValueError                                Traceback (most recent
call last)
<ipython-input-26-98afc27c1701> in <module>()
----> 1 college['MENONLY'].astype('int8') # ValueError: Cannot c
onvert non-finite values (NA or inf) to integer

~/anaconda3/lib/python3.6/site-packages/pandas/util/_decorators.
py in wrapper(*args, **kwargs)
    116             else:
    117                 kwargs[new_arg_name] = new_arg_value
--> 118         return func(*args, **kwargs)
    119     return wrapper
    120 return _deprecate_kwarg

~/anaconda3/lib/python3.6/site-packages/pandas/core/generic.py i
n astype(self, dtype, copy, errors, **kwargs)
    4002         # else, only a single dtype is given
    4003         new_data = self._data.astype(dtype=dtype, copy=c
opy, errors=errors,
-> 4004                                         **kwargs)
    4005         return self._constructor(new_data).__finalize__(
self)
    4006

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py
in astype(self, dtype, **kwargs)
    3455
    3456     def astype(self, dtype, **kwargs):
-> 3457         return self.apply('astype', dtype=dtype, **kwarg
s)
    3458
    3459     def convert(self, **kwargs):

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py
in apply(self, f, axes, filter, do_integrity_check, consolidate
, **kwargs)
    3322
    3323         kwargs['mgr'] = self
-> 3324         applied = getattr(b, f)(**kwargs)
    3325         result_blocks = _extend_blocks(applied, resu
lt_blocks)
    3326

~/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py
in astype(self, dtype, copy, errors, values, **kwargs)
    542     def astype(self, dtype, copy=False, errors='raise',
values=None, **kwargs):
    543         return self._astype(dtype, copy=copy, errors=err
ors, values=values,
--> 544                                         **kwargs)
    545
    546     def _astype(self, dtype, copy=False, errors='raise',

```

```
values=None,  
~/anaconda3/lib/python3.6/site-packages/pandas/core/internals.py  
in _astype(self, dtype, copy, errors, values, klass, mgr, **kwargs)  
    623  
    624                # _astype_nansafe works fine with 1-d only  
ly  
--> 625                values = astype_nansafe(values.ravel(),  
dtype, copy=True)  
    626                values = values.reshape(self.shape)  
    627  
  
~/anaconda3/lib/python3.6/site-packages/pandas/core/dtypes/cast.py  
in astype_nansafe(arr, dtype, copy)  
    685  
    686            if not np.isfinite(arr).all():  
--> 687                raise ValueError('Cannot convert non-finite  
values (NA or inf) to '  
    688                'integer')  
    689  
  
ValueError: Cannot convert non-finite values (NA or inf) to integer
```

```
# 对于数据类型，可以替换字符串名：27、28、30、31是等价的  
In[27]: college.describe(include=['int64', 'float64']).T  
Out[27]:
```

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.000000	0.000000	0.000000	1.000000e+00
MENONLY	7164.0	0.009213	0.095546	0.0	0.000000	0.000000	0.000000	1.000000e+00
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.000000	0.000000	0.000000	1.000000e+00
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.000000	0.000000	0.000000	1.000000e+00
SATVRMID	1185.0	522.819409	68.578862	290.0	475.000000	510.000000	555.000000	7.650000e+02
SATMTMID	1196.0	530.765050	73.469767	310.0	482.000000	520.000000	565.000000	7.850000e+02
DISTANCEONLY	7164.0	0.005583	0.074519	0.0	0.000000	0.000000	0.000000	1.000000e+00
UGDS	6874.0	2356.837940	5474.275871	0.0	117.000000	412.500000	1929.500000	1.515580e+05
UGDS_WHITE	6874.0	0.510207	0.286958	0.0	0.267500	0.55570	0.747875	1.000000e+00
UGDS_BLACK	6874.0	0.189997	0.224587	0.0	0.036125	0.10005	0.257700	1.000000e+00
UGDS_HISP	6874.0	0.161635	0.221854	0.0	0.027600	0.07140	0.198875	1.000000e+00
UGDS_ASIAN	6874.0	0.033544	0.073777	0.0	0.002500	0.01290	0.032700	9.727000e-01
UGDS_AIAN	6874.0	0.013813	0.070196	0.0	0.000000	0.00260	0.007300	1.000000e+00
UGDS_NHPI	6874.0	0.004569	0.033125	0.0	0.000000	0.00000	0.002500	9.983000e-01
UGDS_2MOR	6874.0	0.023950	0.031288	0.0	0.000000	0.01750	0.033900	5.333000e-01
UGDS_NRA	6874.0	0.016086	0.050172	0.0	0.000000	0.00000	0.011700	9.286000e-01
UGDS_UNKN	6874.0	0.045181	0.093440	0.0	0.000000	0.01430	0.045400	9.027000e-01
PPTUG_EF	6853.0	0.226639	0.246470	0.0	0.000000	0.15040	0.376900	1.000000e+00
CURROPER	7535.0	1328.063172	115201.552429	0.0	1.000000	1.000000	1.000000	1.000000e+07
PCTPELL	6849.0	0.530643	0.225544	0.0	0.357800	0.52150	0.712900	1.000000e+00
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.332900	0.58330	0.745000	1.000000e+00
UG25ABV	6718.0	0.410021	0.228939	0.0	0.241500	0.40075	0.572275	1.000000e+00

In[28]: college.describe(include=[np.int64, np.float64]).T
Out[28]:

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.000000	0.000000	0.000000	1.000000e+00
MENONLY	7164.0	0.009213	0.095546	0.0	0.000000	0.000000	0.000000	1.000000e+00
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.000000	0.000000	0.000000	1.000000e+00
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.000000	0.000000	0.000000	1.000000e+00
SATVRMID	1185.0	522.819409	68.578862	290.0	475.000000	510.000000	555.000000	7.650000e+02
SATMTMID	1196.0	530.765050	73.469767	310.0	482.000000	520.000000	565.000000	7.850000e+02
DISTANCEONLY	7164.0	0.005583	0.074519	0.0	0.000000	0.000000	0.000000	1.000000e+00
UGDS	6874.0	2356.837940	5474.275871	0.0	117.000000	412.500000	1929.500000	1.515580e+05
UGDS_WHITE	6874.0	0.510207	0.286958	0.0	0.267500	0.55570	0.747875	1.000000e+00
UGDS_BLACK	6874.0	0.189997	0.224587	0.0	0.036125	0.10005	0.257700	1.000000e+00
UGDS_HISP	6874.0	0.161635	0.221854	0.0	0.027600	0.07140	0.198875	1.000000e+00
UGDS_ASIAN	6874.0	0.033544	0.073777	0.0	0.002500	0.01290	0.032700	9.727000e-01
UGDS_AIAN	6874.0	0.013813	0.070196	0.0	0.000000	0.00260	0.007300	1.000000e+00
UGDS_NHPI	6874.0	0.004569	0.033125	0.0	0.000000	0.00000	0.002500	9.983000e-01
UGDS_2MOR	6874.0	0.023950	0.031288	0.0	0.000000	0.01750	0.033900	5.333000e-01
UGDS_NRA	6874.0	0.016086	0.050172	0.0	0.000000	0.00000	0.011700	9.286000e-01
UGDS_UNKN	6874.0	0.045181	0.093440	0.0	0.000000	0.01430	0.045400	9.027000e-01
PPTUG_EF	6853.0	0.226639	0.246470	0.0	0.000000	0.15040	0.376900	1.000000e+00
CURROPER	7535.0	1328.063172	115201.552429	0.0	1.000000	1.000000	1.000000	1.000000e+07
PCTPELL	6849.0	0.530643	0.225544	0.0	0.357800	0.52150	0.712900	1.000000e+00
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.332900	0.58330	0.745000	1.000000e+00
UG25ABV	6718.0	0.410021	0.228939	0.0	0.241500	0.40075	0.572275	1.000000e+00

```
In[29]: college['RELAFFIL'] = college['RELAFFIL'].astype(np.int8)
In[30]: college.describe(include=['int', 'float']).T # defaults to 64 bit int/floors
Out[30]:
```

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.000000	0.000000	0.000000	1.000000e+00
MENONLY	7164.0	0.009213	0.095546	0.0	0.000000	0.000000	0.000000	1.000000e+00
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.000000	0.000000	0.000000	1.000000e+00
SATVRMID	1185.0	522.819409	68.578862	290.0	475.000000	510.000000	555.000000	7.650000e+02
SATMTMID	1196.0	530.765050	73.469767	310.0	482.000000	520.000000	565.000000	7.850000e+02
DISTANCEONLY	7164.0	0.005583	0.074519	0.0	0.000000	0.000000	0.000000	1.000000e+00
UGDS	6874.0	2356.837940	5474.275871	0.0	117.000000	412.50000	1929.500000	1.515580e+05
UGDS_WHITE	6874.0	0.510207	0.286958	0.0	0.267500	0.55570	0.747875	1.000000e+00
UGDS_BLACK	6874.0	0.189997	0.224587	0.0	0.036125	0.10005	0.257700	1.000000e+00
UGDS_HISP	6874.0	0.161635	0.221854	0.0	0.027600	0.07140	0.198875	1.000000e+00
UGDS_ASIAN	6874.0	0.033544	0.073777	0.0	0.002500	0.01290	0.032700	9.727000e-01
UGDS_AIAN	6874.0	0.013813	0.070196	0.0	0.000000	0.00260	0.007300	1.000000e+00
UGDS_NHPI	6874.0	0.004569	0.033125	0.0	0.000000	0.00000	0.002500	9.983000e-01
UGDS_2MOR	6874.0	0.023950	0.031288	0.0	0.000000	0.01750	0.033900	5.333000e-01
UGDS_NRA	6874.0	0.016086	0.050172	0.0	0.000000	0.00000	0.011700	9.286000e-01
UGDS_UNKN	6874.0	0.045181	0.093440	0.0	0.000000	0.01430	0.045400	9.027000e-01
PPTUG_EF	6853.0	0.226639	0.246470	0.0	0.000000	0.15040	0.376900	1.000000e+00
CURROPER	7535.0	1328.063172	115201.552429	0.0	1.000000	1.00000	1.000000	1.000000e+07
PCTPELL	6849.0	0.530643	0.225544	0.0	0.357800	0.52150	0.712900	1.000000e+00
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.332900	0.58330	0.745000	1.000000e+00
UG25ABV	6718.0	0.410021	0.228939	0.0	0.241500	0.40075	0.572275	1.000000e+00

```
In[31]: college.describe(include=[ 'number' ]).T # also works as  
the default int/float are 64 bits
```

```
Out[31]:
```

	count	mean	std	min	25%	50%	75%	max
HBCU	7164.0	0.014238	0.118478	0.0	0.000000	0.000000	0.000000	1.000000e+00
MENONLY	7164.0	0.009213	0.095546	0.0	0.000000	0.000000	0.000000	1.000000e+00
WOMENONLY	7164.0	0.005304	0.072642	0.0	0.000000	0.000000	0.000000	1.000000e+00
RELAFFIL	7535.0	0.190975	0.393096	0.0	0.000000	0.000000	0.000000	1.000000e+00
SATVRMID	1185.0	522.819409	68.578862	290.0	475.000000	510.000000	555.000000	7.650000e+02
SATMTMID	1196.0	530.765050	73.469767	310.0	482.000000	520.000000	565.000000	7.850000e+02
DISTANCEONLY	7164.0	0.005583	0.074519	0.0	0.000000	0.000000	0.000000	1.000000e+00
UGDS	6874.0	2356.837940	5474.275871	0.0	117.000000	412.500000	1929.500000	1.515580e+05
UGDS_WHITE	6874.0	0.510207	0.286958	0.0	0.267500	0.55570	0.747875	1.000000e+00
UGDS_BLACK	6874.0	0.189997	0.224587	0.0	0.036125	0.10005	0.257700	1.000000e+00
UGDS_HISP	6874.0	0.161635	0.221854	0.0	0.027600	0.07140	0.198875	1.000000e+00
UGDS_ASIAN	6874.0	0.033544	0.073777	0.0	0.002500	0.01290	0.032700	9.727000e-01
UGDS_AIAN	6874.0	0.013813	0.070196	0.0	0.000000	0.00260	0.007300	1.000000e+00
UGDS_NHPI	6874.0	0.004569	0.033125	0.0	0.000000	0.00000	0.002500	9.983000e-01
UGDS_2MOR	6874.0	0.023950	0.031288	0.0	0.000000	0.01750	0.033900	5.333000e-01
UGDS_NRA	6874.0	0.016086	0.050172	0.0	0.000000	0.00000	0.011700	9.286000e-01
UGDS_UNKN	6874.0	0.045181	0.093440	0.0	0.000000	0.01430	0.045400	9.027000e-01
PPTUG_EF	6853.0	0.226639	0.246470	0.0	0.000000	0.15040	0.376900	1.000000e+00
CURROPER	7535.0	1328.063172	115201.552429	0.0	1.000000	1.000000	1.000000	1.000000e+07
PCTPELL	6849.0	0.530643	0.225544	0.0	0.357800	0.52150	0.712900	1.000000e+00
PCTFLOAN	6849.0	0.522211	0.283616	0.0	0.332900	0.58330	0.745000	1.000000e+00
UG25ABV	6718.0	0.410021	0.228939	0.0	0.241500	0.40075	0.572275	1.000000e+00

```
# 转变数据类型时也可以如法炮制
In[32]: college['MENONLY'] = college['MENONLY'].astype('float16')
          college['RELAFFIL'] = college['RELAFFIL'].astype('int8')
)
In[33]: college.index = pd.Int64Index(college.index)
          college.index.memory_usage()
Out[33]: 60280
```

3. 从最大中选择最小

```
# 读取movie.csv，选取'movie_title', 'imdb_score', 'budget'三列
In[34]: movie = pd.read_csv('data/movie.csv')
          movie2 = movie[['movie_title', 'imdb_score', 'budget']]
          movie2.head()
Out[34]:
```

	movie_title	imdb_score	budget
0	Avatar	7.9	237000000.0
1	Pirates of the Caribbean: At World's End	7.1	300000000.0
2	Spectre	6.8	245000000.0
3	The Dark Knight Rises	8.5	250000000.0
4	Star Wars: Episode VII - The Force Awakens	7.1	NaN

```
# 用nlargest方法，选出imdb_score分数最高的100个
```

```
In[35]: movie2.nlargest(100, 'imdb_score').head()
Out[35]:
```

	movie_title	imdb_score	budget
2725	Towering Inferno	9.5	NaN
1920	The Shawshank Redemption	9.3	25000000.0
3402	The Godfather	9.2	6000000.0
2779	Dekalog	9.1	NaN
4312	Kickboxer: Vengeance	9.1	17000000.0

```
# 用链式操作，nsmallest方法再从中挑出预算最小的五部
```

```
In[36]: movie2.nlargest(100, 'imdb_score').nsmallest(5, 'budget')
)
Out[36]:
```

	movie_title	imdb_score	budget
4804	Butterfly Girl	8.7	180000.0
4801	Children of Heaven	8.5	180000.0
4706	12 Angry Men	8.9	350000.0
4550	A Separation	8.4	500000.0
4636	The Other Dream Team	8.4	500000.0

4. 通过排序选取每组的最大值

```
# 同上，选取出三列。按照title_year降序排列
In[37]: movie = pd.read_csv('data/movie.csv')
          movie2 = movie[['movie_title', 'title_year', 'imdb_score']]
In[38]: movie2.sort_values('title_year', ascending=False).head()
Out[38]:
```

	movie_title	title_year	imdb_score
3884	The Veil	2016.0	4.7
2375	My Big Fat Greek Wedding 2	2016.0	6.1
2794	Miracles from Heaven	2016.0	6.8
92	Independence Day: Resurgence	2016.0	5.5
153	Kung Fu Panda 3	2016.0	7.2

```
# 用列表同时对两列进行排序
In[39]: movie3 = movie2.sort_values(['title_year', 'imdb_score'],
          ascending=False)
          movie3.head()
Out[39]:
```

	movie_title	title_year	imdb_score
4312	Kickboxer: Vengeance	2016.0	9.1
4277	A Beginner's Guide to Snuff	2016.0	8.7
3798	Airlift	2016.0	8.5
27	Captain America: Civil War	2016.0	8.2
98	Godzilla Resurgence	2016.0	8.2

```
# 用drop_duplicates去重，只保留每年的第一条数据
In[40]: movie_top_year = movie3.drop_duplicates(subset='title_year')
          movie_top_year.head()
Out[40]:
```

	movie_title	title_year	imdb_score
4312	Kickboxer: Vengeance	2016.0	9.1
3745	Running Forever	2015.0	8.6
4369	Queen of the Mountains	2014.0	8.7
3935	Batman: The Dark Knight Returns, Part 2	2013.0	8.4
3	The Dark Knight Rises	2012.0	8.5

```
# 通过给ascending设置列表，可以同时对一列降序排列，一列升序排列
In[41]: movie4 = movie[['movie_title', 'title_year', 'content_rating', 'budget']]
          movie4_sorted = movie4.sort_values(['title_year', 'content_rating', 'budget'],
                                              ascending=[False, False, True])
          movie4_sorted.drop_duplicates(subset=['title_year', 'content_rating']).head(10)
Out[41]:
```

	movie_title	title_year	content_rating	budget
4026	Compadres	2016.0	R	3000000.0
4658	Fight to the Finish	2016.0	PG-13	150000.0
4661	Rodeo Girl	2016.0	PG	500000.0
3252	The Wailing	2016.0	Not Rated	NaN
4659	Alleluia! The Devil's Carnival	2016.0	NaN	500000.0
4731	Bizarre	2015.0	Unrated	500000.0
812	The Ridiculous 6	2015.0	TV-14	NaN
4831	The Gallows	2015.0	R	100000.0
4825	Romantic Schemer	2015.0	PG-13	125000.0
3796	R.L. Stine's Monsterville: The Cabinet of Souls	2015.0	PG	4400000.0

5. 用sort_values复现nlargest方法

```
# 和前面一样nlargest和nsmallest链式操作进行选取
In[42]: movie = pd.read_csv('data/movie.csv')
        movie2 = movie[['movie_title', 'imdb_score', 'budget']]
        movie_smallest_largest = movie2.nlargest(100, 'imdb_score').nsmallest(5, 'budget')
        movie_smallest_largest
Out[42]:
```

	movie_title	imdb_score	budget
4804	Butterfly Girl	8.7	180000.0
4801	Children of Heaven	8.5	180000.0
4706	12 Angry Men	8.9	350000.0
4550	A Separation	8.4	500000.0
4636	The Other Dream Team	8.4	500000.0

```
# 用sort_values方法，选取imdb_score最高的100个
In[43]: movie2.sort_values('imdb_score', ascending=False).head(100).head()
Out[43]:
# 然后可以再.sort_values('budget').head()，选出预算最低的5个，结果如下
```

	movie_title	imdb_score	budget
4815	A Charlie Brown Christmas	8.4	150000.0
4801	Children of Heaven	8.5	180000.0
4804	Butterfly Girl	8.7	180000.0
4706	12 Angry Men	8.9	350000.0
4636	The Other Dream Team	8.4	500000.0

这两种方法得到的最小的5部电影不同，用tail进行调查：

```
# tail可以查看尾部
In[45]: movie2.nlargest(100, 'imdb_score').tail()
Out[45]:
```

	movie_title	imdb_score	budget
4815	A Charlie Brown Christmas	8.4	150000.0
4801	Children of Heaven	8.5	180000.0
4804	Butterfly Girl	8.7	180000.0
4706	12 Angry Men	8.9	350000.0
4636	The Other Dream Team	8.4	500000.0

```
In[46]: movie2.sort_values('imdb_score', ascending=False).head(100).tail()
Out[46]:
```

	movie_title	imdb_score	budget
3799	Anne of Green Gables	8.4	NaN
3777	Requiem for a Dream	8.4	4500000.0
3935	Batman: The Dark Knight Returns, Part 2	8.4	3500000.0
4636	The Other Dream Team	8.4	500000.0
2455	Aliens	8.4	18500000.0

这是因为评分在8.4以上的电影超过了100部。

6. 计算跟踪止损单价格

```
# pip install pandas_datareader 或 conda install pandas_datareader  
来安装pandas_datareader  
In[47]: import pandas_datareader as pd
```

笔记：pandas_datareader的问题 pandas_datareader在读取“google”源时会有问题。如果碰到问题，切换到“Yahoo”。

```
# 查询特斯拉在2017年第一天的股价  
In[49]: tsla = pd.read_data('tsla', data_source='yahoo', start='2017-1-1')  
        tsla.head(8)  
Out[49]:
```

	Open	High	Low	Close	Adj Close	Volume
Date						
2017-01-03	214.860001	220.330002	210.960007	216.990005	216.990005	5923300
2017-01-04	214.750000	228.000000	214.309998	226.990005	226.990005	11213500
2017-01-05	226.419998	227.479996	221.949997	226.750000	226.750000	5911700
2017-01-06	226.929993	230.309998	225.449997	229.009995	229.009995	5527900
2017-01-09	228.970001	231.919998	228.000000	231.279999	231.279999	3957000
2017-01-10	232.000000	232.000000	226.889999	229.869995	229.869995	3660000
2017-01-11	229.070007	229.979996	226.679993	229.729996	229.729996	3650800
2017-01-12	229.059998	230.699997	225.580002	229.589996	229.589996	3790200

```
# 只关注每天的收盘价，使用cummax得到迄今为止的收盘价最大值
```

```
In[50]: tsla_close = tsla['Close']
In[51]: tsla_cummax = tsla_close.cummax()
tsla_cummax.head(8)
```

```
Out[51]:
```

```
Date
2017-01-03    216.990005
2017-01-04    226.990005
2017-01-05    226.990005
2017-01-06    229.009995
2017-01-09    231.279999
2017-01-10    231.279999
2017-01-11    231.279999
2017-01-12    231.279999
Name: Close, dtype: float64
```

```
# 将下行区间限制到10%，将tsla_cummax乘以0.9
>>> tsla_trailing_stop = tsla_cummax * .9
>>> tsla_trailing_stop.head(8)
Date
2017-01-03    195.291
2017-01-04    204.291
2017-01-05    204.291
2017-01-06    206.109
2017-01-09    208.152
2017-01-10    208.152
2017-01-11    208.152
2017-01-12    208.152
Name: Close, dtype: float64
```

更多

```
# 将上述功能包装成一个函数
In[52]: def set_trailing_loss(symbol, purchase_date, perc):
          close = pdr.DataReader(symbol, 'yahoo', start=purchase_date)[['Close']]
          return close.cummax() * perc
In[53]: tsla_cummax = tsla_close.cummax()
          tsla_cummax.head(8)
Out[53]:
```

```
Date
2017-06-01    59.584998
2017-06-02    60.996002
2017-06-05    61.437999
2017-06-06    61.641997
2017-06-07    61.641997
Name: Close, dtype: float64
```

第04章 选取数据子集

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

1. 选取**Series**数据

```
# 读取college数据集，查看CITY的前5行
In[2]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
        city = college['CITY']
        city.head()
Out[2]: INSTNM
Alabama A & M University           Normal
University of Alabama at Birmingham  Birmingham
Amridge University                   Montgomery
University of Alabama in Huntsville Huntsville
Alabama State University            Montgomery
Name: CITY, dtype: object
```

```
# iloc可以通过整数选取
In[3]: city.iloc[3]
Out[3]: 'Huntsville'
```

```
# iloc通过整数列表选取多行，返回结果是Series
In[4]: city.iloc[[10, 20, 30]]
Out[4]: INSTNM
Birmingham Southern College          B
irmingham
George C Wallace State Community College-Hanceville   H
anceville
Judson College
Marion
Name: CITY, dtype: object
```

```
# 选择等分的数据，可以使用切片语法
In[5]: city.iloc[4:50:10]
Out[5]: INSTNM
        Alabama State University      Montgomery
        Enterprise State Community College Enterprise
        Heritage Christian University    Florence
        Marion Military Institute       Marion
        Reid State Technical College    Evergreen
Name: CITY, dtype: object
```

```
# loc只接收行索引标签
In[6]: city.loc['Heritage Christian University']
Out[6]: 'Florence'
```

```
# 随机选择4个标签
In[7]: np.random.seed(1)
        labels = list(np.random.choice(city.index, 4))
        labels
Out[7]: ['Northwest HVAC/R Training Center',
        'California State University-Dominguez Hills',
        'Lower Columbia College',
        'Southwest Acupuncture College-Boulder']
```

```
# 通过标签列表选择多行
In[8]: city.loc[labels]
Out[8]: INSTNM
        Northwest HVAC/R Training Center      Spokane
        California State University-Dominguez Hills   Carson
        Lower Columbia College           Longview
        Southwest Acupuncture College-Boulder    Boulder
Name: CITY, dtype: object
```

```
# 也可以通过切片语法均匀选择多个
In[9]: city.loc['Alabama State University':'Reid State Technica
1 College':10]
Out[9]: INSTNM
        Alabama State University      Montgomery
        Enterprise State Community College Enterprise
        Heritage Christian University    Florence
        Marion Military Institute       Marion
        Reid State Technical College    Evergreen
Name: CITY, dtype: object
```

```
# 也可以不使用loc，直接使用类似Python的语法
In[10]: city['Alabama State University':'Reid State Technical College':10]
Out[10]: INSTNM
         Alabama State University      Montgomery
         Enterprise State Community College Enterprise
         Heritage Christian University      Florence
         Marion Military Institute      Marion
         Reid State Technical College      Evergreen
Name: CITY, dtype: object
```

更多

```
# 要想只选取一项，并保留其Series类型，则传入一个只包含一项的列表
In[11]: city.iloc[[3]]
Out[11]: INSTNM
         University of Alabama in Huntsville      Huntsville
Name: CITY, dtype: object
```

使用loc切片时要注意，如果start索引再stop索引之后，则会返回空，并且不会报警

```
In[12]: city.loc['Reid State Technical College':'Alabama State University':10]
Out[12]: Series([], Name: CITY, dtype: object)
```

```
# 也可以切片逆序选取
In[13]: city.loc['Reid State Technical College':'Alabama State University':-10]
Out[13]: INSTNM
         Reid State Technical College      Evergreen
         Marion Military Institute      Marion
         Heritage Christian University      Florence
         Enterprise State Community College Enterprise
         Alabama State University      Montgomery
Name: CITY, dtype: object
```

2. 选取DataFrame的行

```
# 还是读取college数据集
In[14]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
         college.head()
Out[14]:
```

INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UG
Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	420.0	0.0	420
University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	565.0	0.0	1138
Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	1.0	29
University of Alabama in Huntsville	Huntsville	AL	0.0	0.0	0.0	0	595.0	590.0	0.0	545
Alabama State	Montgomery	AL	1.0	0.0	0.0	0	425.0	430.0	0.0	481

```
# 选取第61行
In[15]: pd.options.display.max_rows = 6
In[16]: college.iloc[60]
Out[16]:
```

CITY	Anchorage
STABBR	AK
HBCU	0
UG25ABV	0.4386
MD_EARN_WNE_P10	42500
GRAD_DEBT_MDN_SUPP	19449.5
Name:	University of Alaska Anchorage, Length: 26, dtype: object

也可以通过行标签选取

```
In[17]: college.loc['University of Alaska Anchorage']
Out[17]: CITY           Anchorage
          STABBR          AK
          HBCU            0
          ...
          UG25ABV        0.4386
          MD_EARN_WNE_P10 42500
          GRAD_DEBT_MDN_SUPP 19449.5
Name: University of Alaska Anchorage, Length: 26, dtype
: object
```

选取多个不连续的行

```
In[18]: college.iloc[[60, 99, 3]]
Out[18]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGD:
INSTNM										
University of Alaska Anchorage	Anchorage	AK	0.0	0.0	0.0	0	NaN	NaN	0.0	12865.1
International Academy of Hair Design	Tempe	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	188.1
University of Alabama in Huntsville	Huntsville	AL	0.0	0.0	0.0	0	595.0	590.0	0.0	5451.1

3 rows × 26 columns

< [REDACTED] >

也可以用loc加列表来选取

```
In[19]: labels = ['University of Alaska Anchorage',
                 'International Academy of Hair Design',
                 'University of Alabama in Huntsville']
college.loc[labels]
Out[19]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGD:
INSTNM										
University of Alaska Anchorage	Anchorage	AK	0.0	0.0	0.0	0	NaN	NaN	0.0	12865.1
International Academy of Hair Design	Tempe	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	188.1
University of Alabama in Huntsville	Huntsville	AL	0.0	0.0	0.0	0	595.0	590.0	0.0	5451.1

3 rows × 26 columns

< [REDACTED] >

```
# iloc可以用切片连续选取
```

```
In[20]: college.iloc[99:102]
Out[20]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
INSTNM										
International Academy of Hair Design	Tempe	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	188.0
GateWay Community College	Phoenix	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	5211.0
Mesa Community College	Mesa	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	19055.0

3 rows × 26 columns

```
# loc可以用标签连续选取
```

```
In[21]: start = 'International Academy of Hair Design'
stop = 'Mesa Community College'
college.loc[start:stop]
Out[21]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
INSTNM										
International Academy of Hair Design	Tempe	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	188.0
GateWay Community College	Phoenix	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	5211.0
Mesa Community College	Mesa	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	19055.0

3 rows × 26 columns

更多

```
# .index.tolist()可以直接提取索引标签，生成一个列表
In[22]: college.iloc[[60, 99, 3]].index.tolist()
Out[22]: ['University of Alaska Anchorage',
          'International Academy of Hair Design',
          'University of Alabama in Huntsville']
```

3. 同时选取DataFrame的行和列

```
# 读取college数据集，给行索引命名为INSTNM；选取前3行和前4列
In[23]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
         college.iloc[:3, :4]
Out[23]:
```

		CITY	STABBR	HBCU	MENONLY
	INSTNM				
	Alabama A & M University	Normal	AL	1.0	0.0
	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0
	Amridge University	Montgomery	AL	0.0	0.0

```
# 用loc实现同上功能
In[24]: college.loc[:'Amridge University', :'MENONLY']
Out[24]:
```

		CITY	STABBR	HBCU	MENONLY
	INSTNM				
	Alabama A & M University	Normal	AL	1.0	0.0
	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0
	Amridge University	Montgomery	AL	0.0	0.0

```
# 选取两列的所有行
In[25]: college.iloc[:, [4, 6]].head()
Out[25]:
```

	INSTNM	WOMENONLY	SATVRMID
Alabama A & M University	0.0	424.0	
University of Alabama at Birmingham	0.0	570.0	
Amridge University	0.0	NaN	
University of Alabama in Huntsville	0.0	595.0	
Alabama State University	0.0	425.0	

```
# loc实现同上功能
In[26]: college.loc[:, ['WOMENONLY', 'SATVRMID']]
Out[26]:
```

	INSTNM	WOMENONLY	SATVRMID
Alabama A & M University	0.0	424.0	
University of Alabama at Birmingham	0.0	570.0	
Amridge University	0.0	NaN	
...	
National Personal Training Institute of Cleveland	NaN	NaN	
Bay Area Medical Academy - San Jose Satellite Location	NaN	NaN	
Excel Learning Center-San Antonio South	NaN	NaN	

7535 rows × 2 columns

```
# 选取不连续的行和列
In[27]: college.iloc[[100, 200], [7, 15]]
Out[27]:
```

SATMTMID	UGDS_NHPI
----------	-----------

INSTNM

GateWay Community College	NaN	0.0029
American Baptist Seminary of the West	NaN	NaN

```
# 用loc和列表，选取不连续的行和列
```

```
In[28]: rows = ['GateWay Community College', 'American Baptist Seminary of the West']
columns = ['SATMTMID', 'UGDS_NHPI']
college.loc[rows, columns]
```

```
Out[28]:
```

SATMTMID	UGDS_NHPI
----------	-----------

INSTNM

GateWay Community College	NaN	0.0029
American Baptist Seminary of the West	NaN	NaN

```
# iloc选取一个标量值
```

```
In[29]: college.iloc[5, -4]
Out[29]: 0.4010000000000002
```

```
# loc选取一个标量值
```

```
In[30]: college.loc['The University of Alabama', 'PCTFLOAN']
Out[30]: 0.4010000000000002
```

```
# iloc对行切片，并只选取一列
```

```
In[31]: college.iloc[90:80:-2, 5]
Out[31]: INSTNM
      Empire Beauty School-Flagstaff      0
      Charles of Italy Beauty College      0
      Central Arizona College              0
      University of Arizona                0
      Arizona State University-Tempe       0
      Name: RELAFFIL, dtype: int64
```

```
# loc对行切片，并只选取一列
In[32]: start = 'Empire Beauty School-Flagstaff'
          stop = 'Arizona State University-Tempe'
          college.loc[start:stop:-2, 'RELAFFIL']
Out[32]: INSTNM
          Empire Beauty School-Flagstaff      0
          Charles of Italy Beauty College     0
          Central Arizona College             0
          University of Arizona              0
          Arizona State University-Tempe       0
          Name: RELAFFIL, dtype: int64
```

4. 用整数和标签选取数据

```
# 读取college数据集，行索引命名为INSTNM
In[33]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
# 用索引方法get_loc，找到指定列的整数位置
In[34]: col_start = college.columns.get_loc('UGDS_WHITE')
          col_end = college.columns.get_loc('UGDS_UNKN') + 1
          col_start, col_end
Out[34]: (10, 19)
# 用切片选取连续的列
In[35]: college.iloc[:5, col_start:col_end]
Out[35]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA	UGDS_UNKN
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059	0.0000
University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179	0.0000
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000	0.0000
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332	0.0000
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098	0.0243	0.0000

更多

```
# index()方法可以获得整数行对应的标签名
In[36]: row_start = college.index[10]
         row_end = college.index[15]
         college.loc[row_start:row_end, 'UGDS_WHITE':'UGDS_UNKN']
]
Out[36]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA
Birmingham Southern College	0.7983	0.1102	0.0195	0.0517	0.0102	0.0000	0.0051	0.0000
Chattahoochee Valley Community College	0.4661	0.4372	0.0492	0.0127	0.0023	0.0035	0.0151	0.0000
Concordia College Alabama	0.0280	0.8758	0.0373	0.0093	0.0000	0.0000	0.0031	0.0466
South University-Montgomery	0.3046	0.6054	0.0153	0.0153	0.0153	0.0096	0.0000	0.0019
Enterprise State Community College	0.6408	0.2435	0.0509	0.0202	0.0081	0.0029	0.0254	0.0012
James H Faulkner State Community College	0.6979	0.2259	0.0320	0.0084	0.0177	0.0014	0.0152	0.0007

5. 快速选取标量

```
# 通过将行标签赋值给一个变量，用loc选取
In[37]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
          cn = 'Texas A & M University-College Station'
          college.loc[cn, 'UGDS_WHITE']
Out[37]: 0.6609999999999992
```

```
# at可以实现同样的功能
In[38]: college.at[cn, 'UGDS_WHITE']
Out[38]: 0.6609999999999992
```

```
# 用魔术方法%timeit，对速度进行比较
In[39]: %timeit college.loc[cn, 'UGDS_WHITE']
Out[39]: 9.93 µs ± 274 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)
In[40]: %timeit college.at[cn, 'UGDS_WHITE']
Out[40]: 6.69 µs ± 223 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)
```

.iat 和 .at 只接收标量值，是专门用来取代 .iloc 和 .loc 选取标量的，可以节省大概2.5微妙。

```
# 用get_loc找到整数位置，再进行速度比较
In[41]: row_num = college.index.get_loc(cn)
         col_num = college.columns.get_loc('UGDS_WHITE')
In[42]: row_num, col_num
Out[42]: (3765, 10)

In[43]: %timeit college.iloc[row_num, col_num]
Out[43]: 11.1 µs ± 426 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)

In[44]: %timeit college.iat[row_num, col_num]
Out[44]: 7.47 µs ± 109 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)

In[45]: %timeit college.iloc[5, col_num]
Out[45]: 10.8 µs ± 467 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)

In[46]: %timeit college.iat[5, col_num]
Out[46]: 7.12 µs ± 297 ns per loop (mean ± std. dev. of 7 runs,
100000 loops each)
```

更多

```
# Series对象也可以使用.iat和.at选取标量
In[47]: state = college['STABBR']
In[48]: state.iat[1000]
Out[48]: 'IL'

In[49]: state.at['Stanford University']
Out[49]: 'CA'
```

6. 惰性行切片

```
# 读取college数据集；从行索引10到20，每隔一个取一行
In[50]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
         college[10:20:2]
Out[50]:
```

INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGE
Birmingham Southern College	Birmingham	AL	0.0	0.0	0.0	1	560.0	560.0	0.0	1180
Concordia College Alabama	Selma	AL	1.0	0.0	0.0	1	420.0	400.0	0.0	322
Enterprise State Community College	Enterprise	AL	0.0	0.0	0.0	0	NaN	NaN	0.0	1729
Faulkner University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	0.0	2367
New Beginning College of Cosmetology	Albertville	AL	0.0	0.0	0.0	0	NaN	NaN	0.0	115

5 rows × 26 columns

```
# Series也可以进行同样的切片
In[51]: city = college['CITY']
         city[10:20:2]
Out[51]: INSTNM
Birmingham Southern College
Concordia College Alabama
Enterprise State Community College
Faulkner University
New Beginning College of Cosmetology
Name: CITY, dtype: object
                                         Birmingham
                                         Selma
                                         Enterprise
                                         Montgomery
                                         Albertville
```

```
# 查看第4002个行索引标签
In[52]: college.index[4001]
Out[52]: 'Spokane Community College'
```

```
# Series和DataFrame都可以用标签进行切片。下面是对DataFrame用标签切片
In[53]: start = 'Mesa Community College'
         stop = 'Spokane Community College'
         college[start:stop:1500]
Out[53]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
INSTNM										
Mesa Community College	Mesa	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	19055.0
Hair Academy Inc-New Carrollton	New Carrollton	MD	0.0	0.0	0.0	0	NaN	NaN	0.0	504.0
National College of Natural Medicine	Portland	OR	0.0	0.0	0.0	0	NaN	NaN	0.0	NaN

3 rows × 26 columns

< |>

下面是对Series用标签切片

In[54]: city[start:stop:1500]

Out[54]: INSTNM

Mesa Community College	Mesa
Hair Academy Inc-New Carrollton	New Carrollton
National College of Natural Medicine	Portland
Name: CITY, dtype: object	

更多

惰性切片不能用于列，只能用于DataFrame的行和Series，也不能同时选取行和列。

```
# 下面尝试选取两列，导致错误
In[55]: college[:10, ['CITY', 'STABBR']]
-----
-----
TypeError                                 Traceback (most recent
call last)
<ipython-input-55-92538c61bd़fa> in <module>()
----> 1 college[:10, ['CITY', 'STABBR']]

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in __getitem__(self, key)
    1962         return self._getitem_multilevel(key)
    1963     else:
-> 1964         return self._getitem_column(key)
    1965
    1966     def _getitem_column(self, key):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in _getitem_column(self, key)
    1969         # get column
    1970         if self.columns.is_unique:
-> 1971             return self._get_item_cache(key)
    1972
    1973         # duplicate columns & possible reduce dimensiona
lity

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/gene
ric.py in _get_item_cache(self, item)
    1641         """Return the cached item, item represents a lab
el indexer."""
    1642         cache = self._item_cache
-> 1643         res = cache.get(item)
    1644         if res is None:
    1645             values = self._data.get(item)

TypeError: unhashable type: 'slice'
```

```
# 只能用.loc和.iloc选取
In[56]: first_ten_instnm = college.index[:10]
college.loc[first_ten_instnm, ['CITY', 'STABBR']]
Out[56]:
```

	CITY	STABBR
INSTNM		
Alabama A & M University	Normal	AL
University of Alabama at Birmingham	Birmingham	AL
Amridge University	Montgomery	AL
...
Athens State University	Athens	AL
Auburn University at Montgomery	Montgomery	AL
Auburn University	Auburn	AL

10 rows × 2 columns

7. 按照字母切片

```
# 读取college数据集；尝试选取字母顺序在'Sp'和'Su'之间的学校
In[57]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
         college.loc['Sp':'Su']

-----
ValueError                                     Traceback (most recent call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_slice_bound(self, label, side, kind)
    3483             try:
-> 3484                 return self._searchsorted_monotonic(label,
    3485             except ValueError:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in _searchsorted_monotonic(self, label, side)
    3442
-> 3443         raise ValueError('index must be monotonic increasing or decreasing')
    3444

ValueError: index must be monotonic increasing or decreasing

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

```

KeyError                                     Traceback (most recent
call last)
<ipython-input-57-c9f1c69a918b> in <module>()
    1 college = pd.read_csv('data/college.csv', index_col='INS
TNM')
--> 2 college.loc['Sp':'Su']

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in __getitem__(self, key)
    1326         else:
    1327             key = com._apply_if_callable(key, self.obj)
-> 1328             return self._getitem_axis(key, axis=0)
    1329
    1330     def _is_scalar_access(self, key):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in _getitem_axis(self, key, axis)
    1504         if isinstance(key, slice):
    1505             self._has_valid_type(key, axis)
-> 1506             return self._get_slice_axis(key, axis=axis)
    1507         elif is_bool_indexer(key):
    1508             return self._getbool_axis(key, axis=axis)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in _get_slice_axis(self, slice_obj, axis)
    1354         labels = obj._get_axis(axis)
    1355         indexer = labels.slice_indexer(slice_obj.start,
slice_obj.stop,
-> 1356                                         slice_obj.step, k
ind=self.name)
    1357
    1358         if isinstance(indexer, slice):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in slice_indexer(self, start, end, step, kind)
    3348         """
    3349         start_slice, end_slice = self.slice_locs(start,
end, step=step,
-> 3350                                         kind=ki
nd)
    3351
    3352         # return a slice

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in slice_locs(self, start, end, step, kind)
    3536         start_slice = None
    3537         if start is not None:
-> 3538             start_slice = self.get_slice_bound(start, 'l
eft', kind)
    3539         if start_slice is None:
    3540             start_slice = 0

```

```
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_slice_bound(self, label, side, kind)
    3485             except ValueError:
    3486                 # raise the original KeyError
-> 3487                 raise err
    3488
    3489         if isinstance(slc, np.ndarray):
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_slice_bound(self, label, side, kind)
    3479             # we need to look up the label
    3480             try:
-> 3481                 slc = self._get_loc_only_exact_matches(label)
    3482             except KeyError as err:
    3483                 try:
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in _get_loc_only_exact_matches(self, key)
    3448         get_slice_bound.
    3449         """
-> 3450         return self.get_loc(key)
    3451
    3452     def get_slice_bound(self, label, side, kind):
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
    2442             return self._engine.get_loc(key)
    2443         except KeyError:
-> 2444             return self._engine.get_loc(self._maybe_
cast_indexer(key))
    2445
    2446         indexer = self.get_indexer([key], method=method,
tolerance=tolerance)
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5280)()
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20523)()
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item (pandas/_libs/hashtable.c:20477)()
KeyError: 'Sp'
```

```
# 对college进行排序
In[58]: college = college.sort_index()
In[59]: college = college.head()
Out[59]:
```

INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
A & W Healthcare Educators	New Orleans	LA	0.0	0.0	0.0	0	NaN	NaN	0.0	40.0
A T Still University of Health Sciences	Kirksville	MO	0.0	0.0	0.0	0	NaN	NaN	0.0	NaN
ABC Beauty Academy	Garland	TX	0.0	0.0	0.0	0	NaN	NaN	0.0	30.0
ABC Beauty College Inc	Arkadelphia	AR	0.0	0.0	0.0	0	NaN	NaN	0.0	38.0
AI Miami International University of Art and Design	Miami	FL	0.0	0.0	0.0	0	NaN	NaN	0.0	2778.0

5 rows × 26 columns

```
# 再尝试选取字母顺序在'Sp'和'Su'之间的学校
In[60]: pd.options.display.max_rows = 6
In[61]: college.loc['Sp':'Su']
Out[61]:
```

INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
Spa Tech Institute-Ipswich	Ipswich	MA	0.0	0.0	0.0	0	NaN	NaN	0.0	37.0
Spa Tech Institute-Plymouth	Plymouth	MA	0.0	0.0	0.0	0	NaN	NaN	0.0	153.0
Spa Tech Institute-Westboro	Westboro	MA	0.0	0.0	0.0	0	NaN	NaN	0.0	90.0
...
Stylemaster College of Hair Design	Longview	WA	0.0	0.0	0.0	0	NaN	NaN	0.0	77.0
Styles and Profiles Beauty College	Selmer	TN	0.0	0.0	0.0	0	NaN	NaN	0.0	31.0
Styletrends Barber and Hairstyling Academy	Rock Hill	SC	0.0	0.0	0.0	0	NaN	NaN	0.0	45.0

201 rows × 26 columns

```
# 可以用is_monotonic_increasing或is_monotonic_decreasing检测字母排序的顺序
```

```
In[62]: college = college.sort_index(ascending=False)
college.index.is_monotonic_decreasing
```

```
Out[62]: True
```

```
# 字母逆序选取
```

```
In[63]: college.loc['E':'B']
```

```
Out[63]:
```

INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGD
Dyersburg State Community College	Dyersburg	TN	0.0	0.0	0.0	0	NaN	NaN	0.0	2001
Dutchess Community College	Poughkeepsie	NY	0.0	0.0	0.0	0	NaN	NaN	0.0	6885
Dutchess BOCES-Practical Nursing Program	Poughkeepsie	NY	0.0	0.0	0.0	0	NaN	NaN	0.0	155
...
BJ's Beauty & Barber College	Auburn	WA	0.0	0.0	0.0	0	NaN	NaN	0.0	28
BIR Training Center	Chicago	IL	0.0	0.0	0.0	0	NaN	NaN	0.0	2132
B M Spurr School of Practical Nursing	Glen Dale	WV	0.0	0.0	0.0	0	NaN	NaN	0.0	31

1411 rows × 26 columns

< ⟲ ⟳ >

第05章 布尔索引

```
In[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

1. 计算布尔值统计信息

```
# 读取movie，设定行索引是movie_title
In[2]: pd.options.display.max_columns = 50
In[3]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
        movie.head()
Out[3]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_name
movie_title								
Avatar	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore	Eden Richey
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom	Keira Knightley
Spectre	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear	Lea Seydoux
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale	Margot Robbie
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker	John Boyega

```
# 判断电影时长是否超过两小时
In[4]: movie_2_hours = movie['duration'] > 120
movie_2_hours.head(10)
Out[4]: movie_title
Avatar                                True
Pirates of the Caribbean: At World's End  True
Spectre                                 True
The Dark Knight Rises                  True
Star Wars: Episode VII - The Force Awakens False
John Carter                            True
Spider-Man 3                           True
Tangled                                 False
Avengers: Age of Ultron                True
Harry Potter and the Half-Blood Prince  True
Name: duration, dtype: bool
```

```
# 有多少时长超过两小时的电影
In[5]: movie_2_hours.sum()
Out[5]: 1039
```

```
# 超过两小时的电影的比例
In[6]: movie_2_hours.mean()
Out[6]: 0.21135069161920261
```

```
# 用describe()输出一些该布尔Series信息
In[7]: movie_2_hours.describe()
Out[7]: count      4916
unique       2
top        False
freq      3877
Name: duration, dtype: object
```

```
# 实际上，dureation这列是有缺失值的，要想获得真正的超过两小时的电影的比例，需要先删掉缺失值
In[8]: movie['duration'].dropna().gt(120).mean()
Out[8]: 0.21199755152009794
```

原理

```
# 统计False和True值的比例
In[9]: movie_2_hours.value_counts(normalize=True)
Out[9]: False    0.788649
         True    0.211351
Name: duration, dtype: float64
```

更多

```
# 比较同一个DataFrame中的两列
In[10]: actors = movie[['actor_1_facebook_likes', 'actor_2_facebook_likes']].dropna()
         (actors['actor_1_facebook_likes'] > actors['actor_2_facebook_likes']).mean()
Out[10]: 0.97776871303283708
```

2. 构建多个布尔条件

```
In[11]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
         movie.head()
Out[11]:
```

movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name
Avatar	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom
Spectre	Color	Sam Mendes	602.0	148.0	0.0	161.0	Rory Kinnear
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale
Star Wars: Episode VII - The Force Awakens	NaN	Doug Walker	NaN	NaN	131.0	NaN	Rob Walker

```
# 创建多个布尔条件
In[12]: criteria1 = movie.imdb_score > 8
         criteria2 = movie.content_rating == 'PG-13'
         criteria3 = (movie.title_year < 2000) | (movie.title_year >= 2010)
         criteria2.head()
Out[12]: movie_title
          Avatar                         True
          Pirates of the Caribbean: At World's End  True
          Spectre                         True
          The Dark Knight Rises                True
          Star Wars: Episode VII - The Force Awakens False
          Name: content_rating, dtype: bool
```

```
# 将这些布尔条件合并成一个
In[13]: criteria_final = criteria1 & criteria2 & criteria3
         criteria_final.head()
Out[13]: movie_title
          Avatar                         False
          Pirates of the Caribbean: At World's End  False
          Spectre                         False
          The Dark Knight Rises                True
          Star Wars: Episode VII - The Force Awakens False
          Name: content_rating, dtype: bool
```

更多

```
# 在Pandas中，位运算符(&, |, ~)的优先级高于比较运算符，因此如过前面的条件3不加括号，就会报错
In[14]: movie.title_year < 2000 | movie.title_year > 2009
-----
-----
TypeError                                         Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in na_op(x, y)
    882         try:
--> 883             result = op(x, y)
    884         except TypeError:
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in <lambda>(x, y)
    130                                         names('rand_'), op('&
')), 
--> 131                                         ror_=bool_method(lambda x, y: operator.
or_(y, x),
    132                                         names('ror_'), op('|'))
),
```

```
TypeError: ufunc 'bitwise_or' not supported for the input types,
and the inputs could not be safely coerced to any supported typ
es according to the casting rule ''safe''
```

During handling of the above exception, another exception occurred:

```
ValueError                                Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in na_op(x, y)
    900             y = bool(y)
--> 901         result = lib.scalar_binop(x, y, op)
    902     except:
pandas/_libs/lib.pyx in pandas._libs.lib.scalar_binop (pandas/_l
ibs/lib.c:15035)()
```

`ValueError: Buffer dtype mismatch, expected 'Python object' but got 'double'`

During handling of the above exception, another exception occurred:

```
TypeError                                Traceback (most recent
call last)
<ipython-input-14-1e7ee3f1401c> in <module>()
----> 1 movie.title_year < 2000 | movie.title_year > 2009

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in wrapper(self, other)
    933             is_integer_dtype(np.asarray(other))
) else fill_bool)
    934         return filler(self._constructor(
--> 935             na_op(self.values, other),
    936             index=self.index)).__finalize__(self)
    937

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/ops.
py in na_op(x, y)
    903             raise TypeError("cannot compare a dt
yped [{0}] array with "
    904                                         "a scalar of type [{
1}].format(
--> 905                                         x.dtype, type(y)
    906                                         .__name__)
    907             return result

TypeError: cannot compare a dtypes [float64] array with a scalar
of type [bool]
```

3. 用布尔索引过滤

```
# 读取movie数据集，创建布尔条件
In[15]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')

crit_a1 = movie.imdb_score > 8
crit_a2 = movie.content_rating == 'PG-13'
crit_a3 = (movie.title_year < 2000) | (movie.title_year
> 2009)
final_crit_a = crit_a1 & crit_a2 & crit_a3

# 创建第二个布尔条件
In[16]: crit_b1 = movie.imdb_score < 5
crit_b2 = movie.content_rating == 'R'
crit_b3 = (movie.title_year >= 2000) & (movie.title_yea
r <= 2010)
final_crit_b = crit_b1 & crit_b2 & crit_b3

# 将这两个条件用或运算合并起来
In[17]: final_crit_all = final_crit_a | final_crit_b
final_crit_all.head()

Out[17]: movie_title
          Avatar                         False
          Pirates of the Caribbean: At World's End  False
          Spectre                        False
          The Dark Knight Rises            True
          Star Wars: Episode VII - The Force Awakens  False
          dtype: bool
```

```
# 用最终的布尔条件过滤数据
In[18]: movie[final_crit_all].head()
Out[18]:
```

movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name
The Dark Knight Rises	Color	Christopher Nolan	813.0	164.0	22000.0	23000.0	Christian Bale
The Avengers	Color	Joss Whedon	703.0	173.0	0.0	19000.0	Robert Downey Jr.
Captain America: Civil War	Color	Anthony Russo	516.0	147.0	94.0	11000.0	Scarlett Johansson
Guardians of the Galaxy	Color	James Gunn	653.0	121.0	571.0	3000.0	Vin Diesel
Interstellar	Color	Christopher Nolan	712.0	169.0	22000.0	6000.0	Anne Hathaway

```
# 使用loc，对指定的列做过滤操作，可以清楚地看到过滤是否起作用
In[19]: cols = ['imdb_score', 'content_rating', 'title_year']
         movie_filtered = movie.loc[final_crit_all, cols]
         movie_filtered.head(10)
Out[19]:
```

movie_title	imdb_score	content_rating	title_year
The Dark Knight Rises	8.5	PG-13	2012.0
The Avengers	8.1	PG-13	2012.0
Captain America: Civil War	8.2	PG-13	2016.0
Guardians of the Galaxy	8.1	PG-13	2014.0
Interstellar	8.6	PG-13	2014.0
Inception	8.8	PG-13	2010.0
The Martian	8.1	PG-13	2015.0
Town & Country	4.4	R	2001.0
Sex and the City 2	4.3	R	2010.0
Rollerball	3.0	R	2002.0

更多

```
# 用一个长布尔表达式代替前面由短表达式生成的布尔条件
In[21]: final_crit_a2 = (movie.imdb_score > 8) & \
           (movie.content_rating == 'PG-13') & \
           ((movie.title_year < 2000) | (movie.title_year > 2009))
         final_crit_a2.equals(final_crit_a)
Out[21]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGE
3610	Abilene Christian University	Abilene	TX	0.0	0.0	0.0	1	530.0	545.0	0.0	3572
3611	Alvin Community College	Alvin	TX	0.0	0.0	0.0	0	NaN	NaN	0.0	4682
3612	Amarillo College	Amarillo	TX	0.0	0.0	0.0	0	NaN	NaN	0.0	9346
3613	Angelina College	Lufkin	TX	0.0	0.0	0.0	0	NaN	NaN	0.0	3825
3614	Angelo State University	San Angelo	TX	0.0	0.0	0.0	0	475.0	490.0	0.0	5290

4. 用标签索引代替布尔索引

```
# 用布尔索引选取所有得克萨斯州的学校
>>> college = pd.read_csv('data/college.csv')
>>> college[college['STABBR'] == 'TX'].head()
```

```
# 用STABBR作为行索引，然后用loc选取
In[22]: college2 = college.set_index('STABBR')
college2.loc['TX'].head()
Out[22]:
```

STABBR	INSTNM	CITY	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS	UG
TX	Abilene Christian University	Abilene	0.0	0.0	0.0	1	530.0	545.0	0.0	3572.0	
TX	Alvin Community College	Alvin	0.0	0.0	0.0	0	NaN	NaN	0.0	4682.0	
TX	Amarillo College	Amarillo	0.0	0.0	0.0	0	NaN	NaN	0.0	9346.0	
TX	Angelina College	Lufkin	0.0	0.0	0.0	0	NaN	NaN	0.0	3825.0	
TX	Angelo State University	San Angelo	0.0	0.0	0.0	0	475.0	490.0	0.0	5290.0	

```
# 比较二者的速度
```

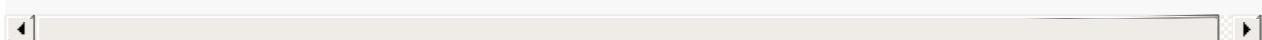
```
In[23]: %timeit college[college['STABBR'] == 'TX']
1.51 ms ± 51.4 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)
```

```
In[24]: %timeit college2.loc['TX']
604 µs ± 23.9 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)
```



```
# 使用STABBR作为行索引所用的时间
```

```
In[25]: %timeit college2 = college.set_index('STABBR')
1.28 ms ± 47.5 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)
```



更多

```
# 使用布尔索引和标签选取多列
```

```
In[26]: states =['TX', 'CA', 'NY']
college[college['STABBR'].isin(states)]
college2.loc[states].head()
```

```
Out[26]:
```

	INSTNM	CITY	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS	UG
STABBR											
TX	Abilene Christian University	Abilene	0.0	0.0	0.0	1	530.0	545.0	0.0	3572.0	
TX	Alvin Community College	Alvin	0.0	0.0	0.0	0	NaN	NaN	0.0	4682.0	
TX	Amarillo College	Amarillo	0.0	0.0	0.0	0	NaN	NaN	0.0	9346.0	
TX	Angelina College	Lufkin	0.0	0.0	0.0	0	NaN	NaN	0.0	3825.0	
TX	Angelo State University	San Angelo	0.0	0.0	0.0	0	475.0	490.0	0.0	5290.0	

5. 用唯一和有序索引选取

```
# 读取college数据集，使用STABBR作为行索引，检查行索引是否有序
In[27]: college = pd.read_csv('data/college.csv')
         college2 = college.set_index('STABBR')
In[28]: college2.index.is_monotonic
Out[28]: False
```

```
# 将college2排序，存储成另一个对象，查看其是否有序
In[29]: college3 = college2.sort_index()
         college3.index.is_monotonic
Out[29]: True
```

```
# 从这三个DataFrame选取得克萨斯州，比较速度
In[30]: %timeit college[college['STABBR'] == 'TX']
        1.58 ms ± 63.8 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)

In[31]: %timeit college2.loc['TX']
        622 µs ± 18.1 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)

In[32]: %timeit college3.loc['TX']
        198 µs ± 5.8 µs per loop (mean ± std. dev. of 7 runs, 1
000 loops each)
```

```
# 使用INSTNM作为行索引，检测行索引是否唯一
In[33]: college_unique = college.set_index('INSTNM')
         college_unique.index.is_unique
Out[33]: True
```

```
# 用布尔索引选取斯坦福大学
In[34]: college[college['INSTNM'] == 'Stanford University']
Out[34]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
4217	Stanford University	Stanford	CA	0.0	0.0	0.0	0	730.0	745.0	0.0	7018.0

```
# 用行索引标签选取斯坦福大学
In[35]: college_unique.loc['Stanford University']
Out[35]:
CITY                      Stanford
STABBR                     CA
HBCU                         0
MENONLY                      0
WOMENONLY                     0
RELAFFIL                      0
SATVRMID                   730
SATMTMID                   745
DISTANCEONLY                  0
UGDS                       7018
UGDS_WHITE                 0.3752
UGDS_BLACK                 0.0591
UGDS_HISP                  0.1607
UGDS_ASIAN                 0.1979
UGDS_AIAN                  0.0114
UGDS_NHPI                  0.0038
UGDS_2MOR                  0.1067
UGDS_NRA                  0.0819
UGDS_UNKN                  0.0031
PPTUG_EF                      0
CURROPER                      1
PCTPELL                   0.1556
PCTFLOAN                   0.1256
UG25ABV                     0.0401
MD_EARN_WNE_P10            86000
GRAD_DEBT_MDN_SUPP        12782
Name: Stanford University, dtype: object
```

```
# 比较两种方法的速度
In[36]: %timeit college[college['INSTNM'] == 'Stanford University']
          1.44 ms ± 66 µs per loop (mean ± std. dev. of 7 runs, 1
          000 loops each)

In[37]: %timeit college_unique.loc['Stanford University']
          191 µs ± 5.31 µs per loop (mean ± std. dev. of 7 runs,
          10000 loops each)
```

更多

```
# 使用CITY和STABBR两列作为行索引，并进行排序
```

```
In[38]: college.index = college['CITY'] + ', ' + college['STABBR']
college = college.sort_index()
college.head()
```

```
Out[38]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONL
ARTESIA, CA	Angeles Institute	ARTESIA	CA	0.0	0.0	0.0	0	NaN	NaN	0
Aberdeen, SD	Presentation College	Aberdeen	SD	0.0	0.0	0.0	1	440.0	480.0	0
Aberdeen, SD	Northern State University	Aberdeen	SD	0.0	0.0	0.0	0	480.0	475.0	0
Aberdeen, WA	Grays Harbor College	Aberdeen	WA	0.0	0.0	0.0	0	NaN	NaN	0
Abilene, TX	Hardin- Simmons University	Abilene	TX	0.0	0.0	0.0	1	508.0	515.0	0

```
# 选取所有Miami, FL的大学
```

```
In[39]: college.loc['Miami, FL'].head()
```

```
Out[39]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UG
Miami, FL	New Professions Technical Institute	Miami	FL	0.0	0.0	0.0	0	NaN	NaN	0.0	5
Miami, FL	Management Resources College	Miami	FL	0.0	0.0	0.0	0	NaN	NaN	0.0	70
Miami, FL	Strayer University- Doral	Miami	FL	NaN	NaN	NaN	1	NaN	NaN	NaN	N
Miami, FL	Keiser University- Miami	Miami	FL	NaN	NaN	NaN	1	NaN	NaN	NaN	N
Miami, FL	George T Baker Aviation Technical College	Miami	FL	0.0	0.0	0.0	0	NaN	NaN	0.0	64

```
# 速度比较
In[40]: %%timeit
crit1 = college['CITY'] == 'Miami'
crit2 = college['STABBR'] == 'FL'
college[crit1 & crit2]
2.83 ms ± 82.4 µs per loop (mean ± std. dev. of 7 runs,
100 loops each)

In[41]: %timeit college.loc['Miami, FL']
226 µs ± 17.3 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)
```

```
# 判断这两个条件是否相同
In[42]: college[(college['CITY'] == 'Miami') & (college['STABBR'] == 'FL')].equals(college.loc['Miami, FL'])
Out[42]: True
```

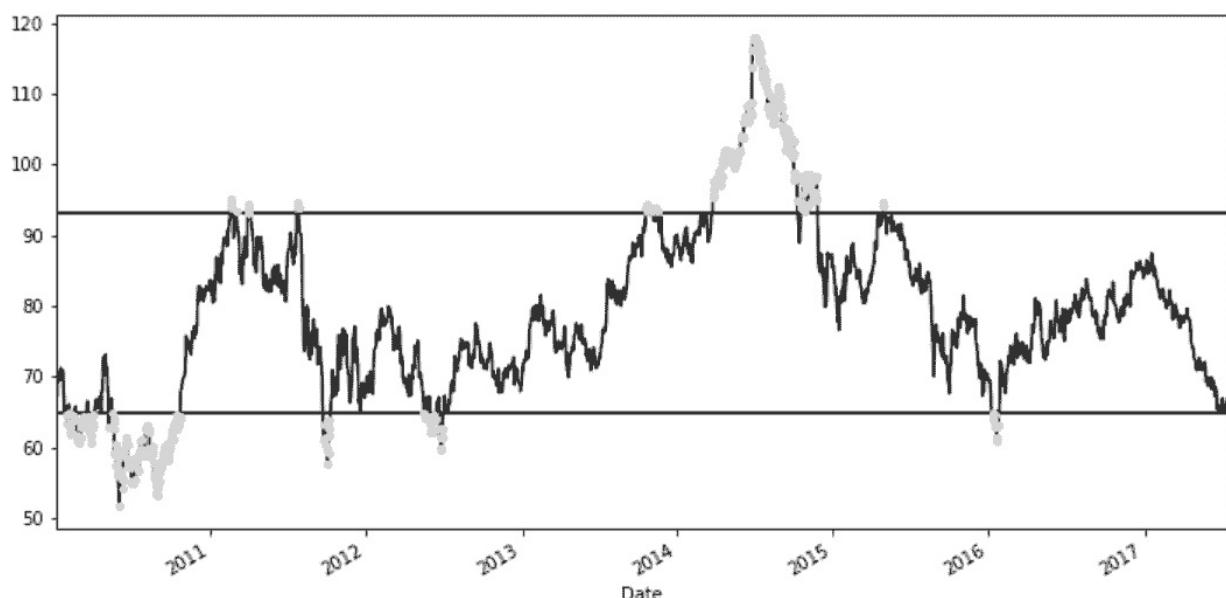
6. 观察股价

```
# 读取Schlumberger stock数据集，行索引设为Date列，并将其转变为Datetime
Index
In[43]: slb = pd.read_csv('data/slb_stock.csv', index_col='Date',
, parse_dates=['Date'])
      slb.head()
Out[43]:
```

	Open	High	Low	Close	Volume
Date					
2010-01-04	66.39	67.20	66.12	67.11	5771234
2010-01-05	66.99	67.62	66.73	67.30	7366270
2010-01-06	67.17	68.94	67.03	68.80	9949946
2010-01-07	68.49	69.81	68.21	69.51	7700297
2010-01-08	69.19	72.00	69.09	70.65	13487621

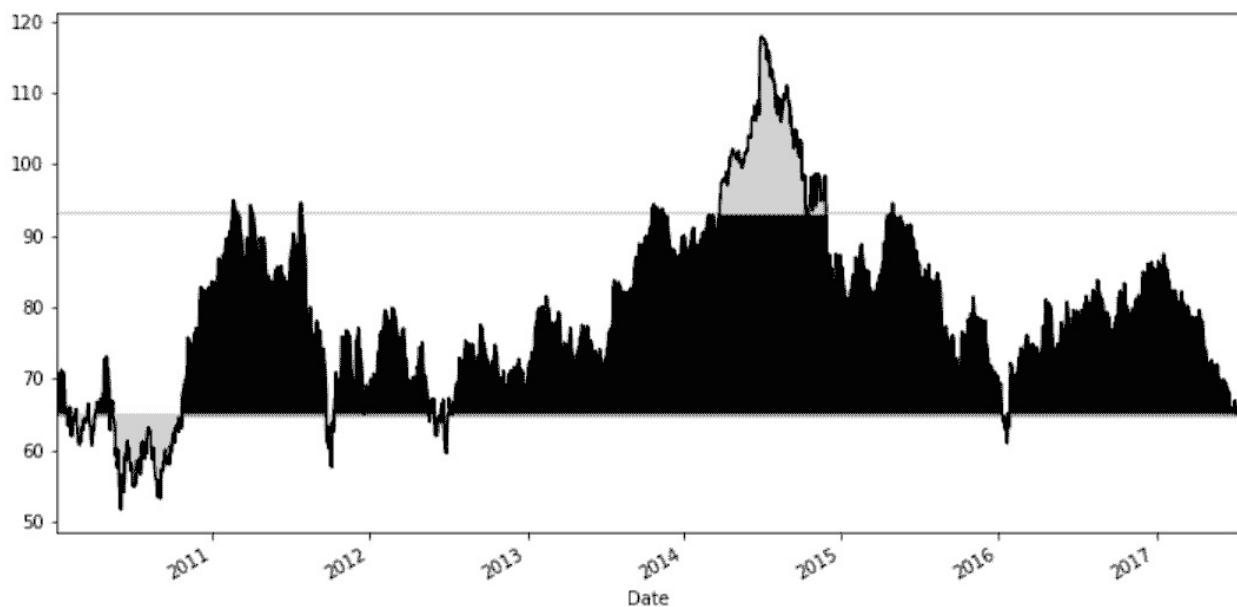
```
# 选取Close这列，用describe返回统计信息
In[44]: slb_close = slb['Close']
          slb_summary = slb_close.describe(percentiles=[.1, .9])
          slb_summary
Out[44]: count    1895.000000
          mean     79.121905
          std      11.767802
          min      51.750000
          10%     64.892000
          50%     78.000000
          90%     93.248000
          max     117.950000
          Name: Close, dtype: float64
```

```
# 用布尔索引选取最高和最低10%的收盘价
In[45]: upper_10 = slb_summary.loc['90%']
          lower_10 = slb_summary.loc['10%']
          criteria = (slb_close < lower_10) | (slb_close > upper_
10)
          slb_top_bottom_10 = slb_close[criteria]
# 过滤出的数据使用灰色，所有的收盘价使用黑色，用matplotlib在十分之一和十分
之九分位数位置画横线
In[46]: slb_close.plot(color='black', figsize=(12, 6))
          slb_top_bottom_10.plot(marker='o', style=' ', ms=4, col
or='lightgray')
          xmin = criteria.index[0]
          xmax = criteria.index[-1]
          plt.hlines(y=[lower_10, upper_10], xmin=xmin, xmax=xmax
, color='black')
Out[46]: <matplotlib.collections.LineCollection at 0x1174b3278>
```



更多

```
# 使用fill_between可以在两条线之间填充颜色
In[47]: slb_close.plot(color='black', figsize=(12,6))
    plt.hlines(y=[lower_10, upper_10],
                xmin=xmin, xmax=xmax, color='lightgray')
    plt.fill_between(x=criteria.index, y1=lower_10,
                      y2=slb_close.values, color='black')
    plt.fill_between(x=criteria.index, y1=lower_10,
                      y2=slb_close.values, where=slb_close <
lower_10,
                      color='lightgray')
    plt.fill_between(x=criteria.index, y1=upper_10,
                      y2=slb_close.values, where=slb_close >
upper_10,
                      color='lightgray')
Out[47]:
```



7. 翻译SQL的WHERE语句

```
# 读取employee数据集
In[48]: employee = pd.read_csv('data/employee.csv')
# 对各项做下了解
In[49]: employee.DEPARTMENT.value_counts().head()
Out[49]: Houston Police Department-HPD      638
          Houston Fire Department (HFD)      384
          Public Works & Engineering-PWE    343
          Health & Human Services        110
          Houston Airport System (HAS)      106
          Name: DEPARTMENT, dtype: int64

In[50]: employee.GENDER.value_counts()
Out[50]: Male      1397
          Female     603
          Name: GENDER, dtype: int64

In[51]: employee.BASE_SALARY.describe().astype(int)
Out[51]: count      1886
          mean      55767
          std       21693
          min      24960
          25%      40170
          50%      54461
          75%      66614
          max      275000
          Name: BASE_SALARY, dtype: int64
```

```
# 创建布尔条件，并从'UNIQUE_ID', 'DEPARTMENT', 'GENDER', 'BASE_SALARY'四列选取
In[52]: depts = ['Houston Police Department-HPD',
                 'Houston Fire Department (HFD)']
criteria_dept = employee.DEPARTMENT.isin(depts)
criteria_gender = employee.GENDER == 'Female'
criteria_sal = (employee.BASE_SALARY >= 80000) & \
                (employee.BASE_SALARY <= 120000)
In[53]: criteria_final = criteria_dept & criteria_gender & criteria_sal
In[54]: select_columns = ['UNIQUE_ID', 'DEPARTMENT', 'GENDER',
                         'BASE_SALARY']
                     employee.loc[criteria_final, select_columns].head()
Out[54]:
```

UNIQUE_ID		DEPARTMENT	GENDER	BASE_SALARY
61	61	Houston Fire Department (HFD)	Female	96668.0
136	136	Houston Police Department-HPD	Female	81239.0
367	367	Houston Police Department-HPD	Female	86534.0
474	474	Houston Police Department-HPD	Female	91181.0
513	513	Houston Police Department-HPD	Female	81239.0

更多

```
# 使用between选取80000到120000之间的薪水
In[55]: criteria_sal = employee.BASE_SALARY.between(80000, 120000)
# 排除最常出现的5家单位
In[56]: top_5_depts = employee.DEPARTMENT.value_counts().index[:5]
         criteria = ~employee.DEPARTMENT.isin(top_5_depts)
         employee[criteria].head()
Out[56]:
```

UNIQUE_ID	POSITION_TITLE	DEPARTMENT	BASE_SALARY	RACE	EMPLOYMENT_TYPE	GENDER	EMPLOYMENT
0	ASSISTANT DIRECTOR (EX LVL)	Municipal Courts Department	121862.0	Hispanic/Latino	Full Time	Female	
1	LIBRARY ASSISTANT	Library	26125.0	Hispanic/Latino	Full Time	Female	
4	ELECTRICIAN	General Services Department	56347.0	White	Full Time	Male	
18	MAINTENANCE MECHANIC III	General Services Department	40581.0	Hispanic/Latino	Full Time	Male	
32	SENIOR ACCOUNTANT	Finance	46963.0	Black or African American	Full Time	Male	

功能一样的SQL语句是：

```

SELECT
*
FROM
    EMPLOYEE
WHERE
    DEPARTMENT not in
(
    SELECT
        DEPARTMENT
    FROM (
        SELECT
            DEPARTMENT,
            COUNT(1) as CT
        FROM
            EMPLOYEE
        GROUP BY
            DEPARTMENT
        ORDER BY
            CT DESC
        LIMIT 5
    )
);

```

8. 确定股票收益的正态值

```

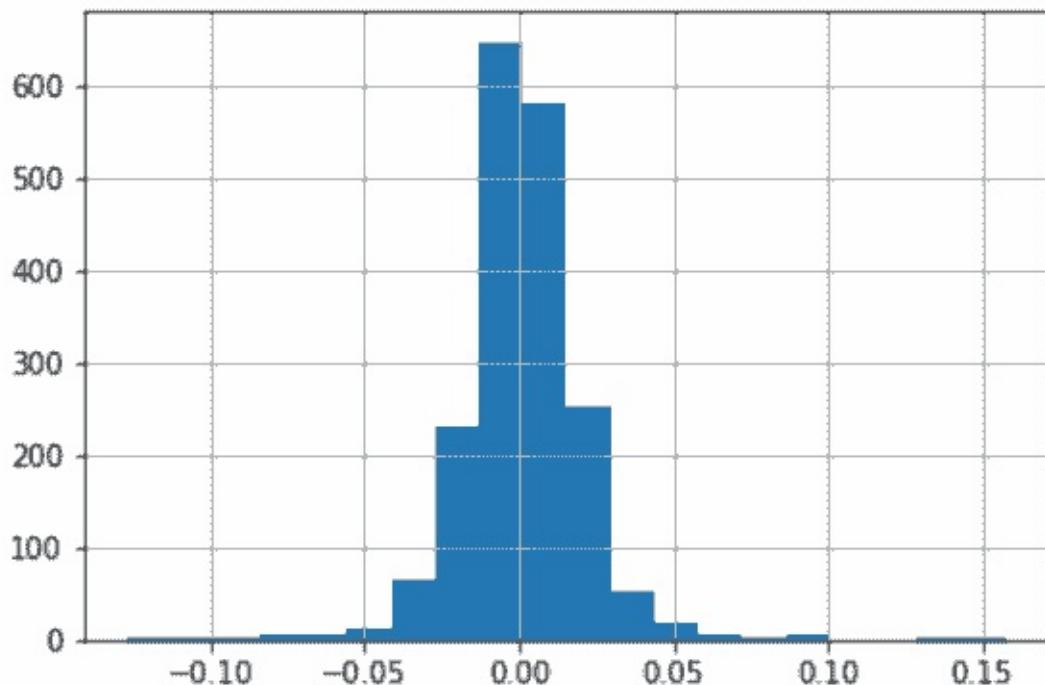
# 加载亚马逊的股票数据，使用Date作为行索引
In[57]: amzn = pd.read_csv('data/amzn_stock.csv', index_col='Date',
                           parse_dates=['Date'])
          amzn.head()
Out[57]:

```

	Open	High	Low	Close	Volume
Date					
2010-01-04	136.25	136.61	133.14	133.90	7600543
2010-01-05	133.43	135.48	131.81	134.69	8856456
2010-01-06	134.60	134.73	131.65	132.25	7180977
2010-01-07	132.01	132.32	128.80	130.00	11030124
2010-01-08	130.56	133.68	129.03	133.52	9833829

```
# 选取Close收盘价，用pct_change()计算每日回报率
In[58]: amzn_daily_return = amzn.Close.pct_change()
amzn_daily_return.head()
Out[58]: Date
2010-01-04      NaN
2010-01-05    0.005900
2010-01-06   -0.018116
2010-01-07   -0.017013
2010-01-08    0.027077
Name: Close, dtype: float64
```

```
# 去掉缺失值，画一张柱状图，查看分布情况
In[59]: amzn_daily_return = amzn_daily_return.dropna()
amzn_daily_return.hist(bins=20)
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x1174b3128>
```



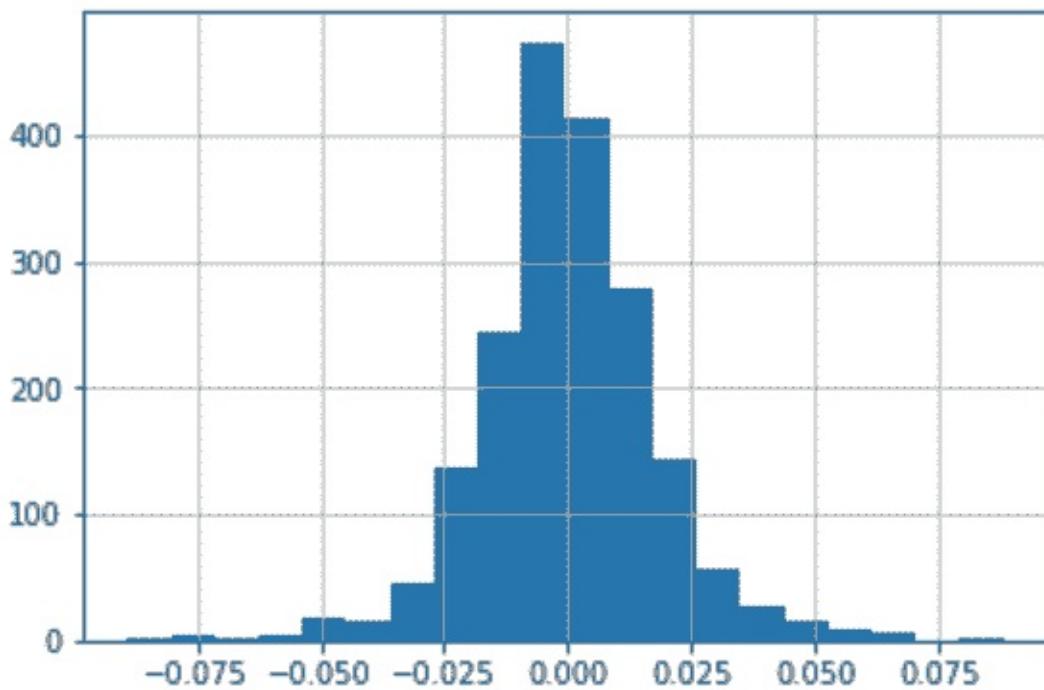
```
# 计算平均值和标准差
In[60]: mean = amzn_daily_return.mean()
          std = amzn_daily_return.std()
# 计算每个数据的z-score的绝对值：z-score是远离平均值的标准差值得个数
In[61]: abs_z_score = amzn_daily_return.sub(mean).abs().div(std)
# 计算位于1, 2, 3个标准差之内的收益率的比例
In[62]: pcts = [abs_z_score.lt(i).mean() for i in range(1,4)]
          print('{:.3f} fall within 1 standard deviation.\n' +
                 '{:.3f} within 2 and {:.3f} within 3'.format(*pcts))
          0.787 fall within 1 standard deviation. 0.956 within 2
and 0.985 within 3
```

更多

```
# 将上面的方法整合成一个函数
In[63]: def test_return_normality(stock_data):
            close = stock_data['Close']
            daily_return = close.pct_change().dropna()
            daily_return.hist(bins=20)
            mean = daily_return.mean()
            std = daily_return.std()

            abs_z_score = abs(daily_return - mean) / std
            pcts = [abs_z_score.lt(i).mean() for i in range(1,4)
]

            print('{:.3f} fall within 1 standard deviation.\n' +
                  '{:.3f} within 2 and {:.3f} within 3'.format(
* pcts))
In[64]: slb = pd.read_csv('data/slb_stock.csv',
                           index_col='Date', parse_dates=['Date'])
            test_return_normality(slb)
            0.742 fall within 1 standard deviation. 0.946 within 2
and 0.986 within 3
```



9. 使用查询方法提高布尔索引的可读性

```
# 读取employee数据，确定选取的部门和列
In[65]: employee = pd.read_csv('data/employee.csv')
        depts = ['Houston Police Department-HPD', 'Houston Fire
        Department (HFD)']
        select_columns = ['UNIQUE_ID', 'DEPARTMENT', 'GENDER',
        'BASE_SALARY']

# 创建查询字符串，并执行query方法
In[66]: qs = "DEPARTMENT in @depts " \
          "and GENDER == 'Female' " \
          "and 80000 <= BASE_SALARY <= 120000"

        emp_filtered = employee.query(qs)
        emp_filtered[select_columns].head()

Out[66]:
```

	UNIQUE_ID	DEPARTMENT	GENDER	BASE_SALARY
61	61	Houston Fire Department (HFD)	Female	96668.0
136	136	Houston Police Department-HPD	Female	81239.0
367	367	Houston Police Department-HPD	Female	86534.0
474	474	Houston Police Department-HPD	Female	91181.0
513	513	Houston Police Department-HPD	Female	81239.0

更多

```
# 若要不使用部门列表，也可以使用下面的方法
In[67]: top10_depts = employee.DEPARTMENT.value_counts().index[:10].tolist()
          qs = "DEPARTMENT not in @top10_depts and GENDER == 'Female'"
          employee_filtered2 = employee.query(qs)
          employee_filtered2[['DEPARTMENT', 'GENDER']].head()
Out[67]:
```

	DEPARTMENT	GENDER
0	Municipal Courts Department	Female
73	Human Resources Dept.	Female
96	City Controller's Office	Female
117	Legal Department	Female
146	Houston Information Tech Svcs	Female

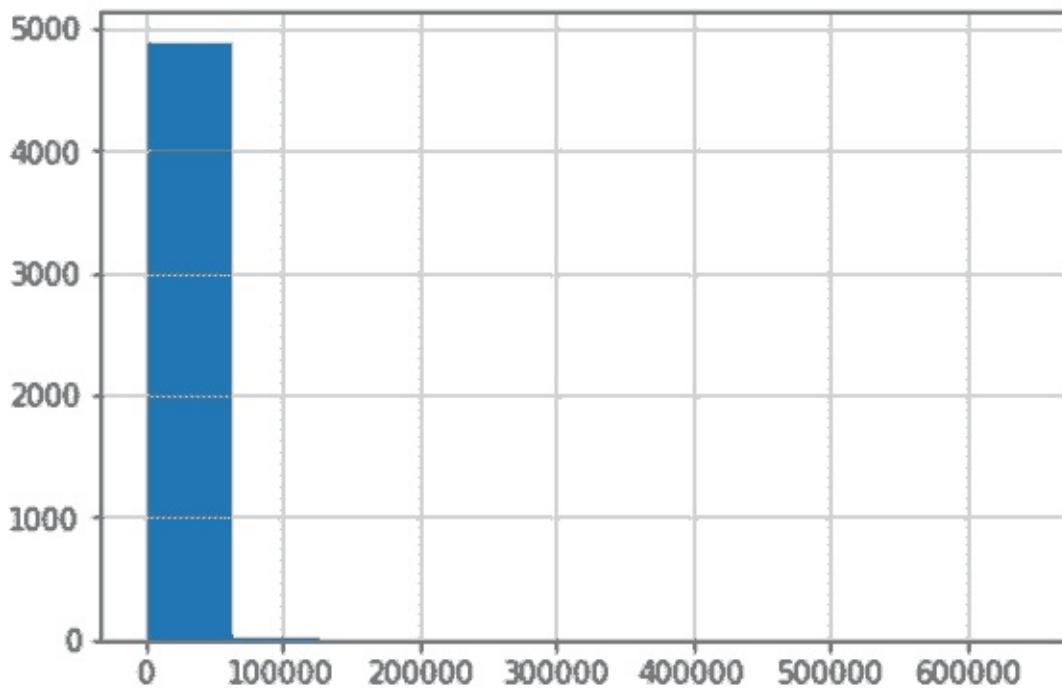
10. 用where方法保留Series

```
# 读取movie数据集，movie_title作为行索引，actor_1_facebook_likes列删除缺失值
In[68]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
          fb_likes = movie['actor_1_facebook_likes'].dropna()
          fb_likes.head()
Out[68]: movie_title
          Avatar                  1000.0
          Pirates of the Caribbean: At World's End    40000.0
          Spectre                  11000.0
          The Dark Knight Rises                27000.0
          Star Wars: Episode VII - The Force Awakens      131.0
          Name: actor_1_facebook_likes, dtype: float64
```

```
# 使用describe获得对数据的认知
In[69]: fb_likes.describe(percentiles=[.1, .25, .5, .75, .9]).a
stype(int)
Out[69]: count      4909
          mean       6494
          std        15106
          min         0
          10%        240
          25%        607
          50%        982
          75%       11000
          90%       18000
          max       640000
Name: actor_1_facebook_likes, dtype: int64
```

```
# 作用和前面相同（这里是作者代码弄乱了）
In[70]: fb_likes.describe(percentiles=[.1, .25, .5, .75, .9])
Out[70]: count      4909.000000
          mean       6494.488491
          std        15106.986884
          min        0.000000
          10%       240.000000
          25%       607.000000
          50%       982.000000
          75%      11000.000000
          90%      18000.000000
          max      640000.000000
Name: actor_1_facebook_likes, dtype: float64
```

```
# 画一张柱状图
In[71]: fb_likes.hist()
Out[71]: <matplotlib.axes._subplots.AxesSubplot at 0x10f9fbe80>
```



```
# 检测小于20000个喜欢的的比例
In[72]: criteria_high = fb_likes < 20000
         criteria_high.mean().round(2)
Out[71]: 0.9100000000000003
```

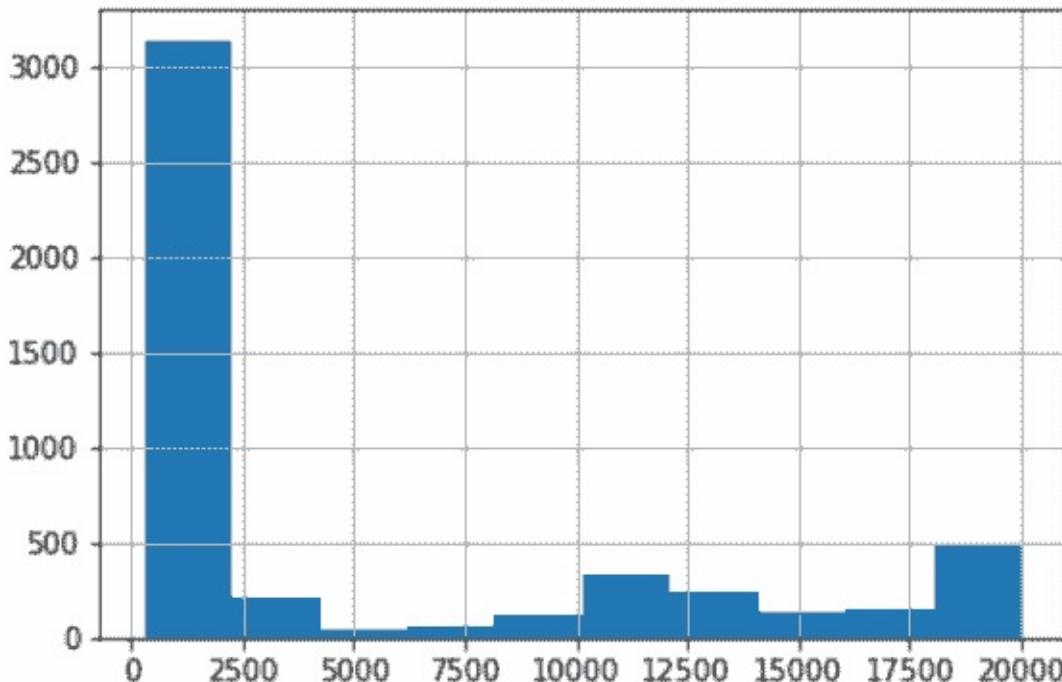
```
# where条件可以返回一个同样大小的Series，但是所有False会被替换成缺失值
In[73]: fb_likes.where(criteria_high).head()
Out[73]: movie_title
          Avatar           1000.0
          Pirates of the Caribbean: At World's End      NaN
          Spectre          11000.0
          The Dark Knight Rises      NaN
          Star Wars: Episode VII - The Force Awakens     131.0
          Name: actor_1_facebook_likes, dtype: float64
```

```
# 第二个参数other，可以让你控制替换值
In[74]: fb_likes.where(criteria_high, other=20000).head()
Out[74]: movie_title
          Avatar           1000.0
          Pirates of the Caribbean: At World's End    20000.0
          Spectre          11000.0
          The Dark Knight Rises          20000.0
          Star Wars: Episode VII - The Force Awakens     131.0
          Name: actor_1_facebook_likes, dtype: float64
```

```
# 通过where条件，设定上下限的值
In[75]: criteria_low = fb_likes > 300
         fb_likes_cap = fb_likes.where(criteria_high, other=2000
0)\                                         .where(criteria_low, 300)
         fb_likes_cap.head()
Out[75]: movie_title
          Avatar                               1000.0
          Pirates of the Caribbean: At World's End 20000.0
          Spectre                                11000.0
          The Dark Knight Rises                  20000.0
          Star Wars: Episode VII - The Force Awakens   300.0
          Name: actor_1_facebook_likes, dtype: float64
```

```
# 原始Series和修改过的Series的长度是一样的
In[76]: len(fb_likes), len(fb_likes_cap)
Out[76]: (4909, 4909)
```

```
# 再做一张柱状图，效果好多了
In[77]: fb_likes_cap.hist()
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x10eeeea8d0>
```



```
In[78]: fb_likes_cap2 = fb_likes.clip(lower=300, upper=20000)
fb_likes_cap2.equals(fb_likes_cap)
Out[78]: True
```

11. 对DataFrame的行做mask

```
# 读取movie，根据条件进行筛选
In[79]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
        c1 = movie['title_year'] >= 2010
        c2 = movie['title_year'].isnull()
        criteria = c1 | c2
# 使用mask方法，使所有满足条件的数据消失
In[80]: movie.mask(criteria).head()
Out[80]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes
movie_title								
Avatar	Color	James Cameron		723.0	178.0	0.0	855.0	Joel David Moore
Pirates of the Caribbean: At World's End	Color	Gore Verbinski		302.0	169.0	563.0	1000.0	Orlando Bloom
Spectre	NaN	NaN		NaN	NaN	NaN	NaN	NaN
The Dark Knight Rises	NaN	NaN		NaN	NaN	NaN	NaN	NaN
Star Wars: Episode VII - The Force Awakens	NaN	NaN		NaN	NaN	NaN	NaN	NaN

```
# 去除缺失值
In[81]: movie_mask = movie.mask(criteria).dropna(how='all')
          movie_mask.head()
Out[81]:
```

	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name	actor_1_facebook_likes
movie_title								
Avatar	Color	James Cameron		723.0	178.0	0.0	855.0	Joel Dav Moor
Pirates of the Caribbean: At World's End	Color	Gore Verbinski		302.0	169.0	563.0	1000.0	Orlando Bloo
Spider-Man 3	Color	Sam Raimi		392.0	156.0	0.0	4000.0	James Franc
Harry Potter and the Half-Blood Prince	Color	David Yates		375.0	153.0	282.0	10000.0	Dani Radclif
Superman Returns	Color	Bryan Singer		434.0	169.0	0.0	903.0	Marlon Branc

```
# 用布尔索引选取title_year小于2010的电影
```

```
In[82]: movie_boolean = movie[movie['title_year'] < 2010]
          movie_boolean.head()
```

```
Out[82]:
```

movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name
Avatar	Color	James Cameron	723.0	178.0	0.0	855.0	Joel David Moore
Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0	563.0	1000.0	Orlando Bloom
Spider-Man 3	Color	Sam Raimi	392.0	156.0	0.0	4000.0	James Franco
Harry Potter and the Half-Blood Prince	Color	David Yates	375.0	153.0	282.0	10000.0	Daniel Radcliffe
Superman Returns	Color	Bryan Singer	434.0	169.0	0.0	903.0	Marion Brando

< >

```
# 判断这两种方法是否相同
```

```
In[83]: movie_mask.equals(movie_boolean)
```

```
Out[83]: False
```

```
# 判断二者的形状是否相同
```

```
In[84]: movie_mask.shape == movie_boolean.shape
```

```
Out[84]: True
```

mask方法产生了许多缺失值，缺失值是float类型，所以之前是整数类型的列都变成了浮点型

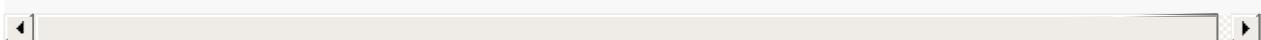
```
In[85]: movie_mask.dtypes == movie_boolean.dtypes
Out[85]:
color                      True
director_name               True
num_critic_for_reviews     True
duration                    True
director_facebook_likes    True
actor_3_facebook_likes     True
actor_2_name                True
actor_1_facebook_likes    True
gross                       True
genres                      True
actor_1_name                True
num_voted_users              False
cast_total_facebook_likes   False
actor_3_name                True
facenumber_in_poster        True
plot_keywords                 True
movie_imdb_link               True
num_user_for_reviews       True
language                     True
country                      True
content_rating                 True
budget                       True
title_year                   True
actor_2_facebook_likes    True
imdb_score                   True
aspect_ratio                  True
movie_facebook_likes        False
dtype: bool
```

Pandas有一个assert_frame_equal方法，可以判断两个Pandas对象是否一样，而不检测其数据类型

```
In[86]: from pandas.testing import assert_frame_equal
        assert_frame_equal(movie_boolean, movie_mask, check_dtypes=False)
```

更多

```
# 比较mask和布尔索引的速度，两者相差了一个数量级
In[87]: %timeit movie.mask(criteria).dropna(how='all')
        11.1 ms ± 48.3 µs per loop (mean ± std. dev. of 7 runs,
100 loops each)
In[88]: %timeit movie[movie['title_year'] < 2010]
        1.12 ms ± 36.7 µs per loop (mean ± std. dev. of 7 runs,
1000 loops each)
```



12. 使用布尔值、整数、标签进行选取

```
# 读取movie，根据布尔条件选取
In[89]: movie = pd.read_csv('data/movie.csv', index_col='movie_title')
         c1 = movie['content_rating'] == 'G'
         c2 = movie['imdb_score'] < 4
         criteria = c1 & c2
In[90]: movie_loc = movie.loc[criteria]
         movie_loc.head()
Out[90]:
```

movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name
The True Story of Puss'N Boots	Color	Jérôme Deschamps	4.0	80.0	0.0	0.0	André Wilms
Doogal	Color	Dave Borthwick	31.0	77.0	3.0	593.0	Kylie Minogue
Thomas and the Magic Railroad	Color	Britt Allcroft	47.0	85.0	2.0	402.0	Colm Feore
Barney's Great Adventure	Color	Steve Gomer	24.0	76.0	9.0	47.0	Kyla Pratt
Justin Bieber: Never Say Never	Color	Jon M. Chu	84.0	115.0	209.0	41.0	Sean Kingston



```
# 检查loc条件和布尔条件创建出来的两个DataFrame是否一样
In[91]: movie_loc.equals(movie[criteria])
Out[91]: True
```

```
# 尝试用.iloc使用布尔索引
In[92]: movie.iloc[movie[criteria]]
-----
-----
ValueError                                Traceback (most recent
call last)
<ipython-input-92-24a12062c6c3> in <module>()
----> 1 movie.iloc[movie[criteria]]

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexing.py in __getitem__(self, key)
    1326         else:
    1327             key = com._apply_if_callable(key, self.obj)
-> 1328             return self._getitem_axis(key, axis=0)
    1329
    1330     def _is_scalar_access(self, key):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexing.py in _getitem_axis(self, key, axis)
    1731
    1732     if is_bool_indexer(key):
-> 1733         self._has_valid_type(key, axis)
    1734         return self._getbool_axis(key, axis=axis)
    1735

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexing.py in _has_valid_type(self, key, axis)
    1588                                         "indexing
on an integer type "
    1589                                         "is not av
ailable")
-> 1590         raise ValueError("iLocation based boolean
n indexing cannot use "
    1591                                         "an indexable as a mask
")
    1592         return True

ValueError: iLocation based boolean indexing cannot use an index
able as a mask
```

```
# 但是，却可以使用布尔值得ndarray，用values可以取出array
In[93]: movie.iloc[movie[criteria].values]
In[94]: movie.iloc.equals(movie.loc)
Out[94]: True
In[95]: movie.loc[criteria.values]
Out[95]:
```

movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes	actor_3_facebook_likes	actor_2_name
The True Story of Puss'N Boots	Color	Jérôme Deschamps	4.0	80.0	0.0	0.0	André Wilms
Doogal	Color	Dave Birthwick	31.0	77.0	3.0	593.0	Kylie Minogue
Thomas and the Magic Railroad	Color	Britt Allcroft	47.0	85.0	2.0	402.0	Colm Feore
Barney's Great Adventure	Color	Steve Gomer	24.0	76.0	9.0	47.0	Kyla Pratt
Justin Bieber: Never Say Never	Color	Jon M. Chu	84.0	115.0	209.0	41.0	Sean Kingston
Sunday School Musical	Color	Rachel Goldenberg	5.0	93.0	7.0	73.0	Mark Heng

布尔索引也可以用来选取列

```
In[96]: criteria_col = movie.dtypes == np.int64
criteria_col.head()
```

```
Out[96]: color           False
         director_name  False
         num_critic_for_reviews  False
         duration        False
         director_facebook_likes  False
         dtype: bool
```

```
In[97]: movie.loc[:, criteria_col].head()
```

```
Out[97]:
```

movie_title	num_voted_users	cast_total_facebook_likes	movie_facebook_likes
Avatar	886204	4834	33000
Pirates of the Caribbean: At World's End	471220	48350	0
Spectre	275868	11700	85000
The Dark Knight Rises	1144337	106759	164000
Star Wars: Episode VII - The Force Awakens	8	143	0

因为criteria_col是包含行索引的一个Series，必须要使用底层的ndarray，才能使用iloc

```
In[98]: movie.iloc[:, criteria_col.values].head()
```

```
Out[98]:
```

movie_title	num_voted_users	cast_total_facebook_likes	movie_facebook_likes
Avatar	886204	4834	33000
Pirates of the Caribbean: At World's End	471220	48350	0
Spectre	275868	11700	85000
The Dark Knight Rises	1144337	106759	164000
Star Wars: Episode VII - The Force Awakens	8	143	0

```
# 选取'content_rating', 'imdb_score', 'title_year', 'gross'四列，  
按照imdb_score升序排列  
In[99]: cols = ['content_rating', 'imdb_score', 'title_year', '  
gross']  
        movie.loc[criteria, cols].sort_values('imdb_score')  
Out[99]:
```

movie_title	content_rating	imdb_score	title_year	gross
Justin Bieber: Never Say Never	G	1.6	2011.0	73000942.0
Sunday School Musical	G	2.5	2008.0	NaN
Doogal	G	2.8	2006.0	7382993.0
Barney's Great Adventure	G	2.8	1998.0	11144518.0
The True Story of Puss'N Boots	G	2.9	2009.0	NaN
Thomas and the Magic Railroad	G	3.6	2000.0	15911333.0

```
# 用get_loc获取这四列的整数位置  
In[100]: col_index = [movie.columns.get_loc(col) for col in col  
s]  
        col_index  
Out[100]: [20, 24, 22, 8]
```

```
# 这时候就可以使用iloc了  
In[101]: movie.iloc[criteria.values, col_index].sort_values('im  
db_score')  
Out[101]:
```

movie_title	content_rating	imdb_score	title_year	gross
Justin Bieber: Never Say Never	G	1.6	2011.0	73000942.0
Sunday School Musical	G	2.5	2008.0	NaN
Doogal	G	2.8	2006.0	7382993.0
Barney's Great Adventure	G	2.8	1998.0	11144518.0
The True Story of Puss'N Boots	G	2.9	2009.0	NaN
Thomas and the Magic Railroad	G	3.6	2000.0	15911333.0

原理

```
# 查看Series的底层结构
In[102]: a = criteria.values
          a[:5]
Out[102]: array([False, False, False, False, False], dtype=bool)

In[103]: len(a), len(criteria)
Out[103]: (4916, 4916)
```

更多

```
# 传入的布尔索引可以跟要操作的DataFrame长度不同
In[104]: movie.loc[[True, False, True], [True, False, False, True]]
Out[104]:
```

movie_title	color	duration
Avatar	Color	178.0
Spectre	Color	148.0

第06章 索引对齐

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

1. 检查索引

```
# 读取college数据集，提取所有的列
In[2]: college = pd.read_csv('data/college.csv')
columns = college.columns
columns
Out[2]: Index(['INSTNM', 'CITY', 'STABBR', 'HBCU', 'MENONLY', 'WOMENONLY', 'RELAFFIL',
               'SATVRMID', 'SATMTMID', 'DISTANCEONLY', 'UGDS',
               'UGDS_WHITE',
               'UGDS_BLACK', 'UGDS_HISP', 'UGDS_ASIAN', 'UGDS_AI
IAN', 'UGDS_NHPI',
               'UGDS_2MOR', 'UGDS_NRA', 'UGDS_UNKN', 'PPTUG_EF',
               'CURROPER', 'PCTPELL',
               'PCTFLOAN', 'UG25ABV', 'MD_EARN_WNE_P10', 'GRAD_
DEBT_MDN_SUPP'], dtype='object')
```

```
# 用values属性，访问底层的NumPy数组
In[3]: columns.values
Out[3]: array(['INSTNM', 'CITY', 'STABBR', 'HBCU', 'MENONLY', 'WOMENONLY',
               'RELAFFIL', 'SATVRMID', 'SATMTMID', 'DISTANCEONLY',
               'UGDS',
               'UGDS_WHITE', 'UGDS_BLACK', 'UGDS_HISP', 'UGDS_ASIAN',
               'UGDS_AI
IAN', 'UGDS_NHPI',
               'UGDS_2MOR', 'UGDS_NRA', 'UGDS_UNKN', 'PPTUG_EF',
               'CURROPER', 'PCTPELL', 'PCTFLOAN', 'UG25ABV', 'MD_EARN_WNE_P10',
               'GRAD_DEBT_MDN_SUPP'], dtype=object)
```

```
# 取出该数组的第6个值
In[4]: columns[5]
Out[4]: 'WOMENONLY'
```

```
# 取出该数组的第2\9\11
In[5]: columns[[1, 8, 10]]
Out[5]: Index(['CITY', 'SATMTMID', 'UGDS'], dtype='object')
```

```
# 逆序切片选取
In[6]: columns[-7:-4]
Out[6]: Index(['PPTUG_EF', 'CURROPER', 'PCTPELL'], dtype='object')
```

```
# 索引有许多和Series和DataFrame相同的方法
In[7]: columns.min(), columns.max(), columns.isnull().sum()
Out[7]: ('CITY', 'WOMENONLY', 0)
```

```
# 索引对象可以直接通过字符串方法修改
In[8]: columns + '_A'
Out[8]: Index(['INSTNM_A', 'CITY_A', 'STABBR_A', 'HBCU_A', 'MENO
NLY_A', 'WOMENONLY_A',
              'RELAFFIL_A', 'SATVRMID_A', 'SATMTMID_A', 'DISTAN
CEONLY_A', 'UGDS_A',
              'UGDS_WHITE_A', 'UGDS_BLACK_A', 'UGDS_HISP_A', 'U
GDS_ASIAN_A',
              'UGDS_AIAN_A', 'UGDS_NHPI_A', 'UGDS_2MOR_A', 'UGD
S_NRA_A',
              'UGDS_UNKN_A', 'PPTUG_EF_A', 'CURROPER_A', 'PCTPE
LL_A', 'PCTFLOAN_A',
              'UG25ABV_A', 'MD_EARN_WNE_P10_A', 'GRAD_DEBT_MDN_
SUPP_A'],
              dtype='object')
```

```
# 索引对象也可以通过比较运算符，得到布尔索引
In[9]: columns > 'G'
Out[9]: array([ True, False,  True,  True,  True,  True,  True,
   True,  True,
              False,  True,  True,  True,  True,  True,  True,
   True,  True,
              True,  True,  True, False,  True,  True,  True,
   True,  True], dtype=bool)
```

```
# 尝试用赋值的方法，修改索引对象的一个值，会导致类型错误，因为索引对象是不可
# 变类型
In[10]: columns[1] = 'city'
-----
-----
TypeError                                 Traceback (most recent
call last)
<ipython-input-10-1e9e8e3125de> in <module>()
----> 1 columns[1] = 'city'

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in __setitem__(self, key, value)
    1668
    1669     def __setitem__(self, key, value):
-> 1670         raise TypeError("Index does not support mutable
operations")
    1671
    1672     def __getitem__(self, key):

TypeError: Index does not support mutable operations
```

更多

索引对象支持集合运算：联合、交叉、求差、对称差

```
# 切片
In[11]: c1 = columns[:4]
          c1
Out[11]: Index(['INSTNM', 'CITY', 'STABBR', 'HBCU'], dtype='obje
ct')

In[12]: c2 = columns[2:5]
          c2
Out[12]: Index(['STABBR', 'HBCU', 'MENONLY'], dtype='object')
```

```
# 联合
In[13]: c1.union(c2)
Out[13]: Index(['CITY', 'HBCU', 'INSTNM', 'MENONLY', 'STABBR'],
dtype='object')

In[14]: c1 | c2
Out[14]: Index(['CITY', 'HBCU', 'INSTNM', 'MENONLY', 'STABBR'],
dtype='object')
```

```
# 对称差
In[15]: c1.symmetric_difference(c2)
Out[15]: Index(['CITY', 'INSTNM', 'MENONLY'], dtype='object')

In[16]: c1 ^ c2
Out[16]: Index(['CITY', 'INSTNM', 'MENONLY'], dtype='object')
```

2. 求笛卡尔积

```
# 创建两个有不同索引、但包含一些相同值的Series
In[17]: s1 = pd.Series(index=list('aaab'), data=np.arange(4))
s1
Out[17]: a    0
          a    1
          a    2
          b    3
dtype: int64

In[18]: s2 = pd.Series(index=list('cababb'), data=np.arange(6))
s2
Out[18]: c    0
          a    1
          b    2
          a    3
          b    4
          b    5
dtype: int64
```

```
# 二者相加，以产生一个笛卡尔积
In[19]: s1 + s2
Out[19]: a    1.0
          a    3.0
          a    2.0
          a    4.0
          a    3.0
          a    5.0
          b    5.0
          b    7.0
          b    8.0
          c    NaN
dtype: float64
```

更多

```
# 当两组索引元素完全相同、顺序也相同时，不会生成笛卡尔积；索引会按照它们的位置对齐。下面的例子，两个Series完全相同，结果也是整数
```

```
In[20]: s1 = pd.Series(index=list('aaabb'), data=np.arange(5))
          s2 = pd.Series(index=list('aaabb'), data=np.arange(5))
          s1 + s2
Out[20]: a    0
          a    2
          a    4
          b    6
          b    8
          dtype: int64
```

```
# 如果索引元素相同，但顺序不同，是能产生笛卡尔积的
```

```
In[21]: s1 = pd.Series(index=list('aaabb'), data=np.arange(5))
          s2 = pd.Series(index=list('bbaaa'), data=np.arange(5))
          s1 + s2
Out[21]: a    2
          a    3
          a    4
          a    3
          a    4
          a    5
          a    4
          a    5
          a    6
          b    3
          b    4
          b    4
          b    5
          dtype: int64
```

3. 索引爆炸

```
# 读取employee数据集，设定行索引是RACE
```

```
In[22]: employee = pd.read_csv('data/employee.csv', index_col='RACE')
          employee.head()
Out[22]:
```

RACE	UNIQUE_ID	POSITION_TITLE	DEPARTMENT	BASE_SALARY	EMPLOYMENT_TYPE	GENDER	EMPLOYME
Hispanic/Latino	0	ASSISTANT DIRECTOR (EX LVL)	Municipal Courts Department	121862.0	Full Time	Female	
Hispanic/Latino	1	LIBRARY ASSISTANT	Library	26125.0	Full Time	Female	
White	2	POLICE OFFICER	Houston Police Department-HPD	45279.0	Full Time	Male	
White	3	ENGINEER/OPERATOR	Houston Fire Department (HFD)	63166.0	Full Time	Male	
White	4	ELECTRICIAN	General Services Department	56347.0	Full Time	Male	

< >

选取BASE_SALARY做成两个Series，判断二者是否相同

```
In[23]: salary1 = employee['BASE_SALARY']
salary2 = employee['BASE_SALARY']
salary1 is salary2
```

```
Out[23]: True
```

结果是True，表明二者指向的同一个对象。这意味着，如果修改一个，另一个也会去改变。为了收到一个全新的数据，使用copy方法：

```
In[24]: salary1 = employee['BASE_SALARY'].copy()
salary2 = employee['BASE_SALARY'].copy()
salary1 is salary2
```

```
Out[24]: False
```

对其中一个做索引排序，比较二者是否不同

```
In[25]: salary1 = salary1.sort_index()
salary1.head()
```

```
Out[25]: RACE
```

American Indian or Alaskan Native	78355.0
American Indian or Alaskan Native	26125.0
American Indian or Alaskan Native	98536.0
American Indian or Alaskan Native	NaN
American Indian or Alaskan Native	55461.0

Name: BASE_SALARY, dtype: float64

```
In[26]: salary2.head()
```

```
Out[26]: RACE
```

Hispanic/Latino	121862.0
Hispanic/Latino	26125.0
White	45279.0
White	63166.0
White	56347.0

Name: BASE_SALARY, dtype: float64

```
# 将两个Series相加
In[27]: salary_add = salary1 + salary2
In[28]: salary_add.head()
Out[28]: RACE
          American Indian or Alaskan Native    138702.0
          American Indian or Alaskan Native    156710.0
          American Indian or Alaskan Native    176891.0
          American Indian or Alaskan Native    159594.0
          American Indian or Alaskan Native    127734.0
          Name: BASE_SALARY, dtype: float64
```

```
# 再将salary1与其自身相加；查看几个所得结果的长度，可以看到长度从2000到达了117万
In[29]: salary_add1 = salary1 + salary1
         len(salary1), len(salary2), len(salary_add), len(salary_add1)
Out[29]: (2000, 2000, 1175424, 2000)
```

更多

```
# 验证salary_add值的个数。因为笛卡尔积是作用在相同索引元素上的，可以对其平方值求和
In[30]: index_vc = salary1.index.value_counts(dropna=False)
         index_vc
Out[30]: Black or African American      700
          White                           665
          Hispanic/Latino                  480
          Asian/Pacific Islander           107
          NaN                             35
          American Indian or Alaskan Native 11
          Others                          2
          Name: RACE, dtype: int64

In[31]: index_vc.pow(2).sum()
Out[31]: 1175424
```

4. 用不等索引填充数值

```
# 读取三个baseball数据集，行索引设为playerID
In[32]: baseball_14 = pd.read_csv('data/baseball14.csv', index_
col='playerID')
        baseball_15 = pd.read_csv('data/baseball15.csv', index_
col='playerID')
        baseball_16 = pd.read_csv('data/baseball16.csv', index_
col='playerID')
        baseball_14.head()
Out[32]:
```

playerID	yearID	stint	teamID	IgID	G	AB	R	H	2B	3B	...	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GDP
altuvjo01	2014	1	HOU	AL	158	660	85	225	47	3	...	59.0	56.0	9.0	36	53.0	7.0	5.0	1.0	5.0	20.0
caritech02	2014	1	HOU	AL	145	507	68	115	21	1	...	88.0	5.0	2.0	56	182.0	6.0	5.0	0.0	4.0	12.0
castrja01	2014	1	HOU	AL	126	465	43	103	21	2	...	56.0	1.0	0.0	34	151.0	1.0	9.0	1.0	3.0	11.0
corpoca01	2014	1	HOU	AL	55	170	22	40	6	0	...	19.0	0.0	0.0	14	37.0	0.0	3.0	1.0	2.0	3.0
dominma01	2014	1	HOU	AL	157	564	51	121	17	0	...	57.0	0.0	1.0	29	125.0	2.0	5.0	2.0	7.0	23.0

5 rows × 21 columns

```
# 用索引方法difference，找到哪些索引标签在baseball_14中，却不在baseball
_15、baseball_16中
In[33]: baseball_14.index.difference(baseball_15.index)
Out[33]: Index(['corpoca01', 'dominma01', 'fowlede01', 'grossroo
1', 'guzmaje01',
               'hoeslj01', 'krausma01', 'preslal01', 'singljo02'
],,
              dtype='object', name='playerID')

In[34]: baseball_14.index.difference(baseball_16.index)
Out[34]: Index(['congeha01', 'correca01', 'gattiev01', 'gomezca0
1',
               'lowrije01', 'rasmuco01', 'tuckepr01', 'valbulu0
1'],
              dtype='object', name='playerID')
```

```
# 找到每名球员在过去三个赛季的击球数，H列包含了这个数据
In[35]: hits_14 = baseball_14['H']
        hits_15 = baseball_15['H']
        hits_16 = baseball_16['H']
        hits_14.head()
Out[35]: Index(['corpoca01', 'dominma01', 'fowlede01', 'grossroo
1', 'guzmaje01',
               'hoeslj01', 'krausma01', 'preslal01', 'singljo02'
],,
              dtype='object', name='playerID')
```

```
# 将hits_14和hits_15两列相加
In[36]: (hits_14 + hits_15).head()
Out[36]: playerID
          altuvjo01    425.0
          cartech02   193.0
          castrja01   174.0
          congeha01      NaN
          corpoca01      NaN
          Name: H, dtype: float64
```

```
# congeha01 和 corpoca01 在2015年是有记录的，但是结果缺失了。使用add方法和参数fill_value，避免产生缺失值
In[37]: hits_14.add(hits_15, fill_value=0).head()
Out[37]: playerID
          altuvjo01    425.0
          cartech02   193.0
          castrja01   174.0
          congeha01     46.0
          corpoca01    40.0
          Name: H, dtype: float64
```

```
# 再将2016的数据也加上
In[38]: hits_total = hits_14.add(hits_15, fill_value=0).add(hits_16, fill_value=0)
          hits_total.head()
Out[38]: playerID
          altuvjo01    641.0
          bregmal01    53.0
          cartech02   193.0
          castrja01   243.0
          congeha01     46.0
          Name: H, dtype: float64
```

```
# 检查结果中是否有缺失值
In[39]: hits_total.hasnans
Out[39]: False
```

原理

```
# 如果一个元素在两个Series都是缺失值，即便使用了fill_value，相加的结果也
仍是缺失值
In[40]: s = pd.Series(index=['a', 'b', 'c', 'd'], data=[np.nan,
3, np.nan, 1])
s
Out[40]: a      NaN
         b    3.0
         c      NaN
         d    1.0
dtype: float64

In[41]: s1 = pd.Series(index=['a', 'b', 'c'], data=[np.nan, 6,
10])
s1
Out[41]: a      NaN
         b    6.0
         c   10.0
dtype: float64

In[42]: s.add(s1, fill_value=5)
Out[42]: a      NaN
         b    9.0
         c   15.0
         d    6.0
dtype: float64

In[43]: s1.add(s, fill_value=5)
Out[43]: a      NaN
         b    9.0
         c   15.0
         d    6.0
dtype: float64
```

更多

```
# 从baseball_14中选取一些列
In[44]: df_14 = baseball_14[['G', 'AB', 'R', 'H']]
df_14.head()
Out[44]:
```

	G	AB	R	H
playerID				
altuvjo01	158	660	85	225
cartech02	145	507	68	115
castrja01	126	465	43	103
corpoca01	55	170	22	40
dominma01	157	564	51	121

```
# 再从baseball_15中选取一些列，有相同的、也有不同的
In[45]: df_15 = baseball_15[['AB', 'R', 'H', 'HR']]
df_15.head()
Out[45]:
```

	AB	R	H	HR
playerID				
altuvjo01	638	86	200	15
cartech02	391	50	78	24
castrja01	337	38	71	11
congeha01	201	25	46	11
correca01	387	52	108	22

```
# 将二者相加的话，只要行或列不能对齐，就会产生缺失值。style属性的highlight_null方法可以高亮缺失值
In[46]: (df_14 + df_15).head(10).style.highlight_null('yellow')
Out[46]:
```

	AB	G	H	HR	R
playerID					
altuvjo01	1298	nan	425	nan	171
cartech02	898	nan	193	nan	118
castrja01	802	nan	174	nan	81
congeha01	nan	nan	nan	nan	nan
corpoca01	nan	nan	nan	nan	nan
correca01	nan	nan	nan	nan	nan
dominma01	nan	nan	nan	nan	nan
fowlede01	nan	nan	nan	nan	nan
gattiev01	nan	nan	nan	nan	nan
gomezca01	nan	nan	nan	nan	nan

```
# 即便使用了fill_value=0，有些值也会是缺失值，这是因为一些行和列的组合根本不存在输入的数据中
```

```
In[47]: df_14.add(df_15, fill_value=0).head(10).style.highlight_null('yellow')
Out[47]:
```

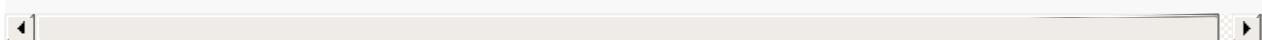
		AB	G	H	HR	R
playerID						
altuvjo01	1298	158	425	15	171	
cartech02	898	145	193	24	118	
castrja01	802	126	174	11	81	
congeha01	201	nan	46	11	25	
corpoca01	170	55	40	nan	22	
correca01	387	nan	108	22	52	
dominma01	564	157	121	nan	51	
fowlede01	434	116	120	nan	61	
gattiev01	566	nan	139	27	66	
gomezca01	149	nan	36	4	19	

5. 从不同的 DataFrame 追加列

```
# 读取employee数据，选取'DEPARTMENT', 'BASE_SALARY'这两列
In[48]: employee = pd.read_csv('data/employee.csv')
         dept_sal = employee[['DEPARTMENT', 'BASE_SALARY']]
# 在每个部门内，对BASE_SALARY进行排序
In[49]: dept_sal = dept_sal.sort_values(['DEPARTMENT', 'BASE_SALARY'],
                                         ascending=[True, False])
)
# 用drop_duplicates方法保留每个部门的第一行
In[50]: max_dept_sal = dept_sal.drop_duplicates(subset='DEPARTMENT')
         max_dept_sal.head()
Out[50]:
```

	DEPARTMENT	BASE_SALARY
1494	Admn. & Regulatory Affairs	140416.0
149	City Controller's Office	64251.0
236	City Council	100000.0
647	Convention and Entertainment	38397.0
1500	Dept of Neighborhoods (DON)	89221.0

```
# 使用DEPARTMENT作为行索引
In[51]: max_dept_sal = max_dept_sal.set_index('DEPARTMENT')
         employee = employee.set_index('DEPARTMENT')
# 现在行索引包含匹配值了，可以向employee的DataFrame新增一列
In[52]: employee['MAX_DEPT_SALARY'] = max_dept_sal['BASE_SALARY']
]
In[53]: pd.options.display.max_columns = 6
Out[54]:
```



DEPARTMENT	UNIQUE_ID	POSITION_TITLE	BASE_SALARY	...	HIRE_DATE	JOB_DATE	MAX_DEPT_SALARY
Municipal Courts Department	0	ASSISTANT DIRECTOR (EX LVL)	121862.0	...	2006-06-12	2012-10-13	121862.0
Library	1	LIBRARY ASSISTANT	26125.0	...	2000-07-19	2010-09-18	107763.0
Houston Police Department-HPD	2	POLICE OFFICER	45279.0	...	2015-02-03	2015-02-03	199596.0
Houston Fire Department (HFD)	3	ENGINEER/OPERATOR	63166.0	...	1982-02-08	1991-05-25	210588.0
General Services Department	4	ELECTRICIAN	56347.0	...	1989-06-19	1994-10-22	89194.0

5 rows × 10 columns

```
# 现在可以用query查看是否有BASE_SALARY大于MAX_DEPT_SALARY的
In[55]: employee.query('BASE_SALARY > MAX_DEPT_SALARY')
Out[55]:
```

```
UNIQUE_ID POSITION_TITLE BASE_SALARY ... HIRE_DATE JOB_DATE MAX_DEPT_SALARY
DEPARTMENT
0 rows × 10 columns
```

原理

```
# 用random从dept_sal随机取10行，不做替换
In[56]: np.random.seed(1234)
         random_salary = dept_sal.sample(n=10).set_index('DEPARTMENT')
         random_salary
Out[56]:
```

BASE_SALARY**DEPARTMENT**

Public Works & Engineering-PWE	50586.0
Houston Police Department-HPD	66614.0
Houston Police Department-HPD	66614.0
Housing and Community Devp.	78853.0
Houston Police Department-HPD	66614.0
Parks & Recreation	NaN
Public Works & Engineering-PWE	37211.0
Public Works & Engineering-PWE	54683.0
Human Resources Dept.	58474.0
Health & Human Services	47050.0

```
# random_salary中是有重复索引的，employee DataFrame的标签要对应random
_salary中的多个标签
In[57]: employee['RANDOM_SALARY'] = random_salary['BASE_SALARY']
-----
ValueError                                     Traceback (most recent
call last)
<ipython-input-57-1cbebe15fa39> in <module>()
----> 1 employee['RANDOM_SALARY'] = random_salary['BASE_SALARY']

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in __setitem__(self, key, value)
    2329         else:
    2330             # set column
-> 2331             self._set_item(key, value)
    2332
```

```

2333     def _setitem_slice(self, key, value):
2344         /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in _set_item(self, key, value)
2345             2395                 self._ensure_valid_index(value)
2346             2396                     value = self._sanitize_column(key, value)
2347             -> 2397                         NDFrame._set_item(self, key, value)
2348             2398
2349             2399
2350
2351     /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in _sanitize_column(self, key, value, broadcast)
2352         2545                     if isinstance(value, Series):
2353             2546                         value = reindexer(value)
2354             2547                     elif isinstance(value, DataFrame):
2355
2356     /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in reindexer(value)
2357         2537                         # duplicate axis
2358             2538                             if not value.index.is_unique:
2359             -> 2539                                 raise e
2360             2540
2361             2541                         # other
2362
2363     /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/fram
e.py in reindexer(value)
2364         2532                         # GH 4107
2365             2533                             try:
2366             -> 2534                                 value = value.reindex(self.index)._v
2367 alues
2368             2535                             except Exception as e:
2369             2536
2370
2371     /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/seri
es.py in reindex(self, index, **kwargs)
2372         2424             @Appender(generic._shared_docs['reindex']) % _shared_
2373 doc_kw_args
2374             2425                 def reindex(self, index=None, **kwargs):
2375             -> 2426                     return super(Series, self).reindex(index=index,
2376 **kwargs)
2377             2427
2378             2428                 @Appender(generic._shared_docs['fillna']) % _shared_d
2379 oc_kw_args
2380
2381     /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/gene
ric.py in reindex(self, *args, **kwargs)
2382         2513                     # perform the reindex on the axes
2383             2514                         return self._reindex_axes(axes, level, limit, to
2384 lerance, method,
2385             -> 2515                                 fill_value, copy).__fi
2386 nalize__(self)

```

```

2516
2517     def _reindex_axes(self, axes, level, limit, tolerance,
e, method, fill_value,
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/gene
ric.py in _reindex_axes(self, axes, level, limit, tolerance, met
hod, fill_value, copy)
    2531             obj = obj._reindex_with_indexers({axis: [new
_index, indexer]}, fill_value=
2532                                         fill_value=
fill_value,
-> 2533                                         copy=copy,
allow_dups=False)
    2534
    2535         return obj

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/gene
ric.py in _reindex_with_indexers(self, reindexers, fill_value, c
opy, allow_dups)
    2625                                         fill_val
ue=fill_value,
    2626                                         allow_du
ps=allow_dups,
-> 2627                                         copy=cop
y)
    2628
    2629         if copy and new_data is self._data:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inte
rnals.py in reindex_indexer(self, new_axis, indexer, axis, fill_
value, allow_dups, copy)
    3884             # some axes don't allow reindexing with dups
    3885             if not allow_dups:
-> 3886                 self.axes[axis]._can_reindex(indexer)
    3887
    3888         if axis >= self.ndim:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in _can_reindex(self, indexer)
    2834             # trying to reindex on an axis with duplicates
    2835             if not self.is_unique and len(indexer):
-> 2836                 raise ValueError("cannot reindex from a dupl
icate axis")
    2837
    2838     def reindex(self, target, method=None, level=None, l
imit=None,
ValueError: cannot reindex from a duplicate axis

```

更多

```
# 选取max_dept_sal['BASE_SALARY'] 的前三行，赋值给employee['MAX_SALARY2']
In[58]: employee['MAX_SALARY2'] = max_dept_sal['BASE_SALARY'].head(3)
# 对MAX_SALARY2统计
In[59]: employee.MAX_SALARY2.value_counts()
Out[59]: 140416.0    29
          100000.0    11
          64251.0     5
          Name: MAX_SALARY2, dtype: int64
# 因为只填充了三个部门的值，所有其它部门在结果中都是缺失值
In[60]: employee.MAX_SALARY2.isnull().mean()
Out[60]: 0.9775000000000004
```

6. 高亮每列的最大值

```
In[61]: pd.options.display.max_rows = 8
# 读取college数据集，INSTNM作为列
In[62]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
college.dtypes
Out[62]: CITY                  object
          STABBR                object
          HBCU                  float64
          MENONLY                float64
          ...
          PCTFLOAN                float64
          UG25ABV                float64
          MD_EARN_WNE_P10         object
          GRAD_DEBT_MDN_SUPP      object
          Length: 26, dtype: object
```

```
# MD_EARN_WNE_P10 和 GRAD_DEBT_MDN_SUPP 两列是对象类型，对其进行检查
# ,发现含有字符串
In[63]: college.MD_EARN_WNE_P10.iloc[0]
Out[63]: '30300'

In[64]: college.GRAD_DEBT_MDN_SUPP.iloc[0]
Out[64]: '30300'
```

```
# 降序检查
In[65]: college.MD_EARN_WNE_P10.sort_values(ascending=False).head()
Out[65]: INSTNM
          Sharon Regional Health System School of Nursing      Priv
          acySuppressed
          Northcoast Medical Training Academy                  Priv
          acySuppressed
          Success Schools                               Priv
          acySuppressed
          Louisiana Culinary Institute                Priv
          acySuppressed
          Bais Medrash Toras Chesed                  Priv
          acySuppressed
          Name: MD_EARN_WNE_P10, dtype: object
```

```
# 可以用to_numeric，将某列的值做强制转换
In[66]: cols = ['MD_EARN_WNE_P10', 'GRAD_DEBT_MDN_SUPP']
         for col in cols:
             college[col] = pd.to_numeric(college[col], errors='coerce')

         college.dtypes.loc[cols]
Out[66]: MD_EARN_WNE_P10      float64
         GRAD_DEBT_MDN_SUPP    float64
         dtype: object
```

```
# 用select_dtypes方法过滤出数值列
In[67]: college_n = college.select_dtypes(include=[np.number])
         college_n.head()
Out[67]:
```

	HBCU	MENONLY	WOMENONLY	...	UG25ABV	MD_EARN_WNE_P10	GRAD_DEBT_MDN_SUPP
INSTNM							
Alabama A & M University	1.0	0.0	0.0	...	0.1049	30300.0	33888.0
University of Alabama at Birmingham	0.0	0.0	0.0	...	0.2422	39700.0	21941.5
Amridge University	0.0	0.0	0.0	...	0.8540	40100.0	23370.0
University of Alabama in Huntsville	0.0	0.0	0.0	...	0.2640	45500.0	24097.0
Alabama State University	1.0	0.0	0.0	...	0.1270	26600.0	33118.5

5 rows × 24 columns

```
# 有的列只含有两个值，用nunique()方法挑出这些列
In[68]: criteria = college_n.nunique() == 2
criteria.head()
Out[68]: HBCU           True
          MENONLY        True
          WOMENONLY       True
          RELAFFIL        True
          SATVRMID       False
          dtype: bool
```

```
# 将布尔Series传给索引运算符，生成二元列的列表
In[69]: binary_cols = college_n.columns[criteria].tolist()
binary_cols
Out[69]: ['HBCU', 'MENONLY', 'WOMENONLY', 'RELAFFIL', 'DISTANCEONLY', 'CURROPER']
```

```
# 用drop方法删除这些列
In[70]: college_n2 = college_n.drop(labels=binary_cols, axis='columns')
college_n2.head()
Out[70]:
```

	SATVRMID	SATMTMID	UGDS	...	UG25ABV	MD_EARN_WNE_P10	GRAD_DEBT_MDN_SUPP
INSTNM							
Alabama A & M University	424.0	420.0	4206.0	...	0.1049	30300.0	33888.0
University of Alabama at Birmingham	570.0	565.0	11383.0	...	0.2422	39700.0	21941.5
Amridge University	NaN	NaN	291.0	...	0.8540	40100.0	23370.0
University of Alabama in Huntsville	595.0	590.0	5451.0	...	0.2640	45500.0	24097.0
Alabama State University	425.0	430.0	4811.0	...	0.1270	26600.0	33118.5

5 rows × 18 columns

```
# 用idxmax方法选出每列最大值的行索引标签
In[71]: max_cols = college_n2.idxmax()
          max_cols
Out[71]: SATVRMID                               California Institute of T
          technology
          SATMTMID                               California Institute of T
          echnology
          UGDS                                 University of Phoenix-Arizona
          x-Arizona
          UGDS_WHITE                            Mr Leon's School of Hair Design-Moscow
          gn-Moscow
          ...
          PCTFLOAN                             ABC Beauty Co
          llege Inc
          UG25ABV                               Dongguk University-Los Angeles
          s Angeles
          MD_EARN_WNE_P10                         Medical College of Wisconsin
          GRAD_DEBT_MDN_SUPP                     Southwest University of Visual Arts-Tucson
          Length: 18, dtype: object
```

```
# 用unique()方法选出所有不重复的列名
In[72]: unique_max_cols = max_cols.unique()
          unique_max_cols[:5]
Out[72]: array(['California Institute of Technology',
               'University of Phoenix-Arizona',
               "Mr Leon's School of Hair Design-Moscow",
               'Velvatec College of Beauty Culture',
               'Thunderbird School of Global Management'], dtype=object)
```

```
# 用max_cols选出只包含最大值的行，用style的highlight_max()高亮
In[73]: college_n2.loc[unique_max_cols].style.highlight_max()
Out[73]:
```

INSTNM	SATVRMID	SATMTMID	UGDS	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_UNKN	UGDS_2MOR	UGDS_NRA	UGDS_AIAN	UGDS_NHPI	UGDS_MHPI
California Institute of Technology	765	785	983	0.2787	0.0153	0.1221	0.4385	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
University of Phoenix-Arizona	nan	nan	151558	0.3098	0.1555	0.076	0.0082	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Mr Leon's School of Hair Design-Moscow	nan	nan	16	1	0	0	0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Velvatex College of Beauty Culture	nan	nan	25	0	1	0	0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Thunderbird School of Global Management	nan	nan	1	0	0	1	0	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
Cosmopolitan Beauty and Tech School	nan	nan	110	0.0091	0	0.0182	0.9727	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000

更多

```
# 用axis参数可以高亮每行的最大值
In[74]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
college_ugds = college.filter(like='UGDS_').head()
college_ugds.style.highlight_max(axis='columns')

Out[74]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA	UGDS_UNKN
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0	0.0059	0.0138
University of Alabama at Birmingham	0.5922	0.26	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179	0.01
Amridge University	0.299	0.4192	0.0069	0.0034	0	0	0	0	0.2715
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332	0.035
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.001	0.0006	0.0098	0.0243	0.0137

```
In[75]: pd.Timedelta(1, unit='Y')
Out[75]: Timedelta('365 days 05:49:12')
```

7. 用链式方法重现idxmax

```
# 和前面一样，只选出数值列
In[76]: college = pd.read_csv('data/college.csv', index_col='INSTNM')

cols = ['MD_EARN_WNE_P10', 'GRAD_DEBT_MDN_SUPP']
for col in cols:
    college[col] = pd.to_numeric(college[col], errors='coerce')

college_n = college.select_dtypes(include=[np.number])
criteria = college_n.nunique() == 2
binary_cols = college_n.columns[criteria].tolist()
college_n = college_n.drop(labels=binary_cols, axis='columns')

In[77]: college_n.max().head()
Out[77]: SATVRMID      765.0
SATMTMID      785.0
UGDS          151558.0
UGDS_WHITE     1.0
UGDS_BLACK     1.0
dtype: float64
```

college_n.max()可以选出每列的最大值，用eq方法比较DataFrame的每个值和该列的最大值

```
In[78]: college_n.eq(college_n.max()).head()
Out[78]:
```

INSTNM	SATVRMID	SATMTMID	UGDS	...	UG25ABV	MD_EARN_WNE_P10	GRAD_DEBT_MDN_SUPP
Alabama A & M University	False	False	False	...	False	False	False
University of Alabama at Birmingham	False	False	False	...	False	False	False
Amridge University	False	False	False	...	False	False	False
University of Alabama in Huntsville	False	False	False	...	False	False	False
Alabama State University	False	False	False	...	False	False	False

5 rows × 18 columns

```
# 用any方法，选出至少包含一个True值的行
In[79]: has_row_max = college_n.eq(college_n.max()).any(axis='columns')
          has_row_max.head()
Out[79]: INSTNM
          Alabama A & M University      False
          University of Alabama at Birmingham  False
          Amridge University      False
          University of Alabama in Huntsville  False
          Alabama State University      False
          dtype: bool
```

```
# 因为只有18列，has_row_max最多只能有18个True，来看下实际共有多少个
In[80]: college_n.shape
Out[80]: (7535, 18)
```

```
In[81]: has_row_max.sum()
Out[81]: 401
```

```
# 结果很奇怪，这是因为许多百分比的列的最大值是1。转而使用cumsum()累积求和
In[82]: has_row_max.sum()
In[83]: college_n.eq(college_n.max()).cumsum()
Out[83]:
```

	SATVRMID	SATMTMID	UGDS	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR
INSTNM										
Alabama A & M University	0	0	0	0	0	0	0	0	0	0
University of Alabama at Birmingham	0	0	0	0	0	0	0	0	0	0
Amridge University	0	0	0	0	0	0	0	0	0	0
...
National Personal Training Institute of Cleveland	1	1	1	109	28	136	1	2	1	1
Bay Area Medical Academy - San Jose Satellite Location	1	1	1	109	28	136	1	2	1	1
Excel Learning Center-San Antonio South	1	1	1	109	28	136	1	2	1	1

7535 rows × 18 columns

一些列只有一个最大值，比如SATVRMID和SATMTMID，UGDS_WHITE列却有许多最大值。有109所学校的学生100%是白人。如果再使用一次cumsum，1在每列中就只出现一次，而且会是最大值首次出现的位置：

```
>>> college_n.eq(college_n.max()).cumsum().cumsum()
```

INSTNM	SATVRMID	SATMTMID	UGDS	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA
Alabama A & M University	0	0	0	0	0	0	0	0	0	0	0
University of Alabama at Birmingham	0	0	0	0	0	0	0	0	0	0	0
Amridge University	0	0	0	0	0	0	0	0	0	0	0
...
National Personal Training Institute of Cleveland	7307	7307	417	379968	73163	341375	985	11386	3318	5058	1078
Bay Area Medical Academy - San Jose Satellite Location	7308	7308	418	380077	73191	341511	986	11388	3319	5059	1079
Excel Learning Center-San Antonio South	7309	7309	419	380186	73219	341647	987	11390	3320	5060	1080

现在就可以用eq方法去和1进行比较，然后用any方法，选出所有至少包含一个True值的行

```
In[84]: has_row_max2 = college_n.eq(college_n.max())\
    .cumsum()\
    .cumsum()\
    .eq(1)\
    .any(axis='columns')
```

```
has_row_max2.head()
```

```
Out[84]: INSTNM
Alabama A & M University      False
University of Alabama at Birmingham  False
Amridge University      False
University of Alabama in Huntsville  False
Alabama State University      False
dtype: bool
```

```
# 查看有多少True值
```

```
In[85]: has_row_max2.sum()
```

```
Out[85]: 16
```

```
# 直接通过布尔索引选出这些学校
In[86]: idxmax_cols = has_row_max2[has_row_max2].index
          idxmax_cols
Out[86]: Index(['Thunderbird School of Global Management',
                 'Southwest University of Visual Arts-Tucson', 'A
BC Beauty College Inc',
                 'Velvatec College of Beauty Culture',
                 'California Institute of Technology',
                 'Le Cordon Bleu College of Culinary Arts-San Fra
ncisco',
                 'MTI Business College Inc', 'Dongguk University-
Los Angeles',
                 'Mr Leon's School of Hair Design-Moscow',
                 'Haskell Indian Nations University', 'LIU Brentw
ood',
                 'Medical College of Wisconsin', 'Palau Community
College',
                 'California University of Management and Science
s',
                 'Cosmopolitan Beauty and Tech School', 'Universi
ty of Phoenix-Arizona'],
                dtype='object', name='INSTNM')
```

```
# 和idxmax方法的结果比较
In[87]: set(college_n.idxmax().unique()) == set(idxmax_cols)
Out[87]: True
```

更多

```
# 耗时比较
In[88]: %timeit college_n.idxmax().values
          1.11 ms ± 50.9 µs per loop (mean ± std. dev. of 7 runs,
          1000 loops each)

Out[89]: %timeit college_n.eq(college_n.max())\
              .cumsum()\
              .cumsum()\
              .eq(1)\
              .any(axis='columns')\
              [lambda x: x].index
          5.26 ms ± 35.6 µs per loop (mean ± std. dev. of 7 runs,
          100 loops each)
```

8. 找到最常见的最大值

```
# 读取college，过滤出只包含本科生种族比例信息的列
In[90]: pd.options.display.max_rows= 40
In[91]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
         college_ugds = college.filter(like='UGDS_')
         college_ugds.head()
Out[91]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_2MOR	UGDS_NRA	UGDS_UNKN
Alabama A & M University	0.0333	0.9353	0.0055	...	0.0000	0.0059
University of Alabama at Birmingham	0.5922	0.2600	0.0283	...	0.0368	0.0179
Amridge University	0.2990	0.4192	0.0069	...	0.0000	0.0000
University of Alabama in Huntsville	0.6988	0.1255	0.0382	...	0.0172	0.0332
Alabama State University	0.0158	0.9208	0.0121	...	0.0098	0.0243

5 rows × 9 columns

```
# 用idxmax方法选出每行种族比例最高的列名
In[92]: highest_percentage_race = college_ugds.idxmax(axis='columns')
         highest_percentage_race.head()
Out[92]: INSTNM
          Alabama A & M University           UGDS_BLACK
          University of Alabama at Birmingham   UGDS_WHITE
          Amridge University                   UGDS_BLACK
          University of Alabama in Huntsville   UGDS_WHITE
          Alabama State University              UGDS_BLACK
          dtype: object
```

```
# 用value_counts，查看最大值的分布
In[93]: highest_percentage_race.value_counts(normalize=True)
Out[93]: UGDS_WHITE    0.670352
          UGDS_BLACK     0.151586
          UGDS_HISP      0.129473
          UGDS_UNKN      0.023422
          UGDS_ASIAN     0.012074
          UGDS_AIAN      0.006110
          UGDS_NRA       0.004073
          UGDS_NHPI      0.001746
          UGDS_2MOR      0.001164
          dtype: float64
```

更多

```
# 对于黑人比例最高的学校，排名第二的种族的分布情况
In[94]: college_black = college_ugds[highest_percentage_race ==
'UGDS_BLACK']
college_black = college_black.drop('UGDS_BLACK', axis='columns')
college_black.idxmax(axis='columns').value_counts(normalize=True)
Out[94]: UGDS_WHITE      0.670352
          UGDS_BLACK     0.151586
          UGDS_HISP       0.129473
          UGDS_UNKN       0.023422
          UGDS_ASIAN      0.012074
          UGDS_AIAN       0.006110
          UGDS_NRA        0.004073
          UGDS_NHPI       0.001746
          UGDS_2MOR       0.001164
dtype: float64
```

第07章 分组聚合、过滤、转换

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

1. 定义聚合

```
# 读取flights数据集，查询头部
In[2]: flights = pd.read_csv('data/flights.csv')
flights.head()
Out[2]:
```

	MONTH	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_DELAY	DIVERTED	CANCELLED
0	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905	65.0	0	0
1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333	-13.0	0	0
2	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453	35.0	0	0
3	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935	-7.0	0	0
4	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225	39.0	0	0

```
# 按照AIRLINE分组，使用agg方法，传入要聚合的列和聚合函数
In[3]: flights.groupby('AIRLINE').agg({'ARR_DELAY':'mean'}).head()
Out[3]:
```

```
# 或者要选取的列使用索引，聚合函数作为字符串传入agg
In[4]: flights.groupby('AIRLINE')[['ARR_DELAY']].agg('mean').head()
Out[4]:
AIRLINE
AA      5.542661
AS     -0.833333
B6      8.692593
DL      0.339691
EV      7.034580
Name: ARR_DELAY, dtype: float64
```

ARR_DELAY

AIRLINE

AA	5.542661
AS	-0.833333
B6	8.692593
DL	0.339691
EV	7.034580

```
# 也可以向agg中传入NumPy的mean函数
```

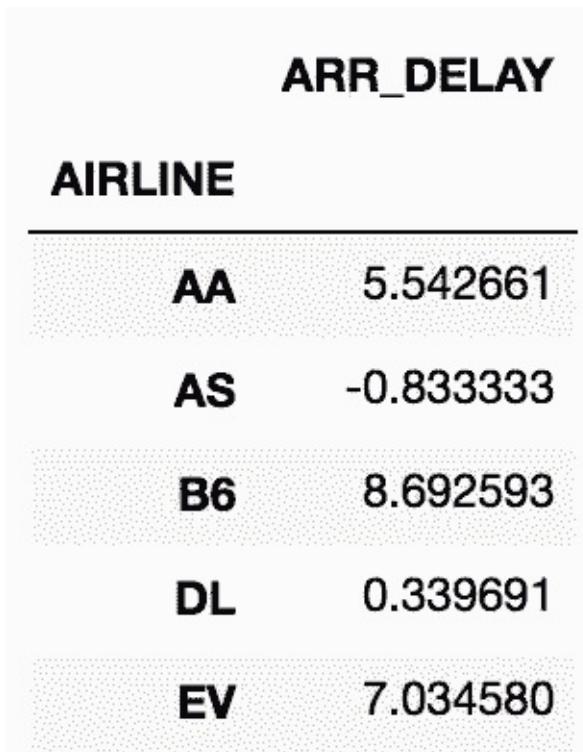
```
In[5]: flights.groupby('AIRLINE')[['ARR_DELAY']].agg(np.mean).head()  
Out[5]:
```

ARR_DELAY

AIRLINE

AA	5.542661
AS	-0.833333
B6	8.692593
DL	0.339691
EV	7.034580

```
# 也可以直接使用mean()函数
In[6]: flights.groupby('AIRLINE')['ARR_DELAY'].mean().head()
Out[6]:
```



原理

```
# groupby方法产生的是一个DataFrameGroupBy对象
In[7]: grouped = flights.groupby('AIRLINE')
type(grouped)
Out[7]: pandas.core.groupby.DataFrameGroupBy
```

更多

```
# 如果agg接收的不是聚合函数，则会导致异常
In[8]: flights.groupby('AIRLINE')['ARR_DELAY'].agg(np.sqrt)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py:842: RuntimeWarning: invalid value encountered in sqrt
    f = lambda x: func(x, *args, **kwargs)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py:3015: RuntimeWarning: invalid value encountered in sqrt
    output = func(group, *args, **kwargs)

-----
ValueError
call last)                                            Traceback (most recent
```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in agg_series(self, obj, func)
    2177         try:
-> 2178             return self._aggregate_series_fast(obj, func
)
    2179         except Exception:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_series_fast(self, obj, func)
    2197                     dummy)
-> 2198             result, counts = grouper.get_result()
    2199             return result, counts

pandas/_libs/src/reduce.pyx in pandas._libs.lib.SeriesGrouper.ge
t_result (pandas/_libs/lib.c:39105)()

pandas/_libs/src/reduce.pyx in pandas._libs.lib.SeriesGrouper.ge
t_result (pandas/_libs/lib.c:38973)()

pandas/_libs/src/reduce.pyx in pandas._libs.lib._get_result_arra
y (pandas/_libs/lib.c:32039)()

ValueError: function does not reduce

During handling of the above exception, another exception occur
ed:

ValueError                                Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
    2882         try:
-> 2883             return self._python_agg_general(func_or_
funcs, *args, **kwargs)
    2884         except Exception:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _python_agg_general(self, func, *args, **kwargs)
    847             try:
--> 848                 result, counts = self.grouper.agg_series
(obj, f)
    849                 output[name] = self._try_cast(result, ob
j, numeric_only=True)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in agg_series(self, obj, func)
    2179         except Exception:
-> 2180             return self._aggregate_series_pure_python(ob
j, func)
    2181

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_series_pure_python(self, obj, func)

```

```

2214                     isinstance(res, list)):
-> 2215                         raise ValueError('Function does not
reduce')
2216                         result = np.empty(nGroups, dtype='O')

ValueError: Function does not reduce

During handling of the above exception, another exception occurred:

Exception                                         Traceback (most recent
call last)
<ipython-input-8-2bcc9ccfec77> in <module>()
----> 1 flights.groupby('AIRLINE')['ARR_DELAY'].agg(np.sqrt)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
    2883             return self._python_agg_general(func_or_
funcs, *args, **kwargs)
    2884         except Exception:
-> 2885             result = self._aggregate_named(func_or_f
uncs, *args, **kwargs)
    2886
    2887             index = Index(sorted(result), name=self.grou
per.names[0])

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_named(self, func, *args, **kwargs)
    3015             output = func(group, *args, **kwargs)
    3016             if isinstance(output, (Series, Index, np.nda
rray)):
-> 3017                 raise Exception('Must produce aggregated
value')
    3018             result[name] = self._try_cast(output, group)
    3019

```

Exception: Must produce aggregated value

2. 用多个列和函数进行分组和聚合

```

# 导入数据
In[9]: flights = pd.read_csv('data/flights.csv')
flights.head()
Out[9]:

```

MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225

每家航空公司每周平均每天取消的航班数

```
In[10]: flights.groupby(['AIRLINE', 'WEEKDAY'])['CANCELLED'].agg('sum').head(7)
```

```
Out[10]: AIRLINE    WEEKDAY
```

AA	1	41
	2	9
	3	16
	4	20
	5	18
	6	21
	7	29

```
Name: CANCELLED, dtype: int64
```

分组可以是多组，选取可以是多组，聚合函数也可以是多个

每周每家航空公司取消或改变航线的航班总数和比例

```
In[11]: flights.groupby(['AIRLINE', 'WEEKDAY'])['CANCELLED', 'DIVERTED'].agg(['sum', 'mean']).head(7)
```

```
Out[11]:
```

AIRLINE	WEEKDAY	CANCELLED		DIVERTED	
		sum	mean	sum	mean
AA	1	41	0.032106	6	0.004699
	2	9	0.007341	2	0.001631
	3	16	0.011949	2	0.001494
	4	20	0.015004	5	0.003751
	5	18	0.014151	1	0.000786
	6	21	0.018667	9	0.008000
	7	29	0.021837	1	0.000753

```
# 用列表和嵌套字典对多列分组和聚合
# 对于每条航线，找到总航班数，取消的数量和比例，飞行时间的平均时间和方差
In[12]: group_cols = ['ORG_AIR', 'DEST_AIR']
agg_dict = {'CANCELLED': ['sum', 'mean', 'size'],
            'AIR_TIME': ['mean', 'var']}
flights.groupby(group_cols).agg(agg_dict).head()
# flights.groupby(['ORG_AIR', 'DEST_AIR']).agg({'CANCELLED': ['sum', 'mean', 'size'],
#                                                 'AIR_TIME': ['mean', 'var']}).head()
Out[12]:
```

	ORG_AIR	DEST_AIR	CANCELLED			AIR_TIME	
			sum	mean	size	mean	var
ATL	ABE	0	0.0	31	96.387097	45.778495	
	ABQ	0	0.0	16	170.500000	87.866667	
	ABY	0	0.0	19	28.578947	6.590643	
	ACY	0	0.0	6	91.333333	11.466667	
	AEX	0	0.0	40	78.725000	47.332692	

3. 分组后去除多级索引

```
# 读取数据
In[13]: flights = pd.read_csv('data/flights.csv')
flights.head()
Out[13]:
```

MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225

```
# 按'AIRLINE', 'WEEKDAY'分组，分别对DIST和ARR_DELAY聚合
In[14]: airline_info = flights.groupby(['AIRLINE', 'WEEKDAY'])\n        .agg({'DIST': ['sum', 'mean'],\n              'ARR_DELAY': ['min', 'max']})\n        .astype(int)\n        airline_info.head()\nOut[14]:
```

		DIST		ARR_DELAY	
		sum	mean	min	max
AIRLINE	WEEKDAY				
	1	1455386	1139	-60	551
	2	1358256	1107	-52	725
AA	3	1496665	1117	-45	473
	4	1452394	1089	-46	349
	5	1427749	1122	-41	732

```
# 行和列都有两级索引，get_level_values(0)取出第一级索引
In[15]: level0 = airline_info.columns.get_level_values(0)
level0
Out[15]: Index(['DIST', 'DIST', 'ARR_DELAY', 'ARR_DELAY'], dtype='object')
```

```
# get_level_values(1)取出第二级索引
In[16]: level1 = airline_info.columns.get_level_values(1)
level1
Out[16]: Index(['sum', 'mean', 'min', 'max'], dtype='object')
```

```
# 一级和二级索引拼接成新的列索引
In[17]: airline_info.columns = level0 + '_' + level1
In[18]: airline_info.head(7)
Out[18]:
```

		DIST_sum	DIST_mean	ARR_DELAY_min	ARR_DELAY_max
AIRLINE	WEEKDAY				
AA	1	1455386	1139	-60	551
	2	1358256	1107	-52	725
	3	1496665	1117	-45	473
	4	1452394	1089	-46	349
	5	1427749	1122	-41	732
	6	1265340	1124	-50	858
	7	1461906	1100	-49	626

```
# reset_index()可以将行索引变成单级
```

```
In[19]: airline_info.reset_index().head(7)
```

```
Out[19]:
```

	AIRLINE	WEEKDAY	DIST_sum	DIST_mean	ARR_DELAY_min	ARR_DELAY_max
0	AA	1	1455386	1139	-60	551
1	AA	2	1358256	1107	-52	725
2	AA	3	1496665	1117	-45	473
3	AA	4	1452394	1089	-46	349
4	AA	5	1427749	1122	-41	732
5	AA	6	1265340	1124	-50	858
6	AA	7	1461906	1100	-49	626

更多

```
# Pandas默认会在分组运算后，将所有分组的列放在索引中，as_index设为False可以避免这么做。分组后使用reset_index，也可以达到同样的效果
```

```
In[20]: flights.groupby(['AIRLINE'], as_index=False)[['DIST']].agg('mean').round(0)
Out[20]:
```

	AIRLINE	DIST
0	AA	1114.0
1	AS	1066.0
2	B6	1772.0
3	DL	866.0
4	EV	460.0
5	F9	970.0
6	HA	2615.0
7	MQ	404.0
8	NK	1047.0
9	OO	511.0
10	UA	1231.0
11	US	1181.0
12	VX	1240.0
13	WN	810.0

```
# 上面这么做，会默认对AIRLINE排序，sort设为False可以避免排序
In[21]: flights.groupby(['AIRLINE'], as_index=False, sort=False)
          [ 'DIST'].agg('mean')
Out[21]:
```

	AIRLINE	DIST
0	WN	809.985626
1	UA	1230.918891
2	MQ	404.229041
3	AA	1114.347865
4	F9	969.593014
5	EV	460.237453
6	OO	511.239375
7	NK	1047.428100
8	US	1181.226625
9	AS	1065.884115
10	DL	866.448448
11	VX	1240.296073
12	B6	1771.882136
13	HA	2615.178571

4. 自定义聚合函数

```
In[22]: college = pd.read_csv('data/college.csv')
college.head()
Out[22]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...
0	Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	420.0	0.0	...
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	565.0	0.0	...
2	Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	1.0	...
3	University of Alabama in Huntsville	Huntsville	AL	0.0	0.0	0.0	0	595.0	590.0	0.0	...
4	Alabama State University	Montgomery	AL	1.0	0.0	0.0	0	425.0	430.0	0.0	...

5 rows × 27 columns

< | ... | >

求出每个州的本科生的平均值和标准差

```
In[23]: college.groupby('STABBR')['UGDS'].agg(['mean', 'std']).round(0).head()
Out[23]:
```

	mean	std
STABBR		
AK	2493.0	4052.0
AL	2790.0	4658.0
AR	1644.0	3143.0
AS	1276.0	NaN
AZ	4130.0	14894.0

```
# 远离平均值的标准差的最大个数，写一个自定义函数
In[24]: def max_deviation(s):
    std_score = (s - s.mean()) / s.std()
    return std_score.abs().max()
# agg聚合函数在调用方法时，直接引入自定义的函数名
In[25]: college.groupby('STABBR')['UGDS'].agg(max_deviation).round(1).head()
Out[25]: STABBR
          AK    2.6
          AL    5.8
          AR    6.3
          AS    NaN
          AZ    9.9
Name: UGDS, dtype: float64
```

更多

```
# 自定义的聚合函数也适用于多个数值列
In[26]: college.groupby('STABBR')[['UGDS', 'SATVRMID', 'SATMTMID']].agg(max_deviation).round(1).head()
Out[26]:
```

	UGDS	SATVRMID	SATMTMID
STABBR			
AK	2.6	NaN	NaN
AL	5.8	1.6	1.8
AR	6.3	2.2	2.3
AS	NaN	NaN	NaN
AZ	9.9	1.9	1.4

```
# 自定义聚合函数也可以和预先定义的函数一起使用
In[27]: college.groupby(['STABBR', 'RELAFFIL'])[['UGDS', 'SATVRMID', 'SATMTMID']].agg([max_deviation, 'mean', 'std']).round(1).head()
Out[27]:
```

		UGDS			SATVRMID			SATMTMID		
		max_deviation	mean	std	max_deviation	mean	std	max_deviation	mean	std
STABBR	RELAFFIL									
AK	0	2.1	3508.9	4539.5	NaN	NaN	NaN	NaN	NaN	NaN
	1	1.1	123.3	132.9	NaN	555.0	NaN	NaN	503.0	NaN
AL	0	5.2	3248.8	5102.4	1.6	514.9	56.5	1.7	515.8	56.7
	1	2.4	979.7	870.8	1.5	498.0	53.0	1.4	485.6	61.4
AR	0	5.8	1793.7	3401.6	1.9	481.1	37.9	2.0	503.6	39.0

```
# Pandas使用函数名作为返回列的名字；你可以直接使用rename方法修改，或通过__name__属性修改
```

```
In[28]: max_deviation.__name__
Out[28]: 'max_deviation'
```

```
In[29]: max_deviation.__name__ = 'Max Deviation'
In[30]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS', 'SATVRMID', 'SATMTMID']\
         .agg([max_deviation, 'mean', 'std']).round(1).head()
Out[30]:
```

		UGDS			SATVRMID			SATMTMID		
		Max Deviation	mean	std	Max Deviation	mean	std	Max Deviation	mean	std
STABBR	RELAFFIL									
AK	0	2.1	3508.9	4539.5	NaN	NaN	NaN	NaN	NaN	NaN
	1	1.1	123.3	132.9	NaN	555.0	NaN	NaN	503.0	NaN
AL	0	5.2	3248.8	5102.4	1.6	514.9	56.5	1.7	515.8	56.7
	1	2.4	979.7	870.8	1.5	498.0	53.0	1.4	485.6	61.4
AR	0	5.8	1793.7	3401.6	1.9	481.1	37.9	2.0	503.6	39.0

5. 用 args 和 *kwargs 自定义聚合函数

```
# 用inspect模块查看groupby对象的agg方法的签名
In[31]: college = pd.read_csv('data/college.csv')
          grouped = college.groupby(['STABBR', 'RELAFFIL'])
In[32]: import inspect
          inspect.signature(grouped.agg)
Out[32]: <Signature (arg, *args, **kwargs)>
```

如何做

```
# 自定义一个返回去本科生人数在1000和3000之间的比例的函数
In[33]: def pct_between_1_3k(s):
    return s.between(1000, 3000).mean()

# 用州和宗教分组，再聚合
In[34]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(pct
_between_1_3k).head(9)
Out[34]:
STABBR  RELAFFIL
AK        0      0.142857
          1      0.000000
AL        0      0.236111
          1      0.333333
AR        0      0.279412
          1      0.111111
AS        0      1.000000
AZ        0      0.096774
          1      0.000000
Name: UGDS, dtype: float64
```

```
# 但是这个函数不能让用户自定义上下限，再新写一个函数
In[35]: def pct_between(s, low, high):
    return s.between(low, high).mean()

# 使用这个自定义聚合函数，并传入最大和最小值
In[36]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(pct
_between, 1000, 10000).head(9)
Out[36]:
STABBR  RELAFFIL
AK        0      0.428571
          1      0.000000
AL        0      0.458333
          1      0.375000
AR        0      0.397059
          1      0.166667
AS        0      1.000000
AZ        0      0.233871
          1      0.111111
Name: UGDS, dtype: float64
```

原理

```
# 显示指定最大和最小值
In[37]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(pct
_between, high=10000, low=1000).head(9)
Out[37]:
STABBR  RELAFFIL
AK      0      0.428571
        1      0.000000
AL      0      0.458333
        1      0.375000
AR      0      0.397059
        1      0.166667
AS      0      1.000000
AZ      0      0.233871
        1      0.111111
Name: UGDS, dtype: float64
```

```
# 也可以关键字参数和非关键字参数混合使用，只要非关键字参数在后面
In[38]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(pct
_between, 1000, high=10000).head(9)
Out[38]:
STABBR  RELAFFIL
AK      0      0.428571
        1      0.000000
AL      0      0.458333
        1      0.375000
AR      0      0.397059
        1      0.166667
AS      0      1.000000
AZ      0      0.233871
        1      0.111111
Name: UGDS, dtype: float64
```

更多

```
# Pandas不支持多重聚合时，使用参数
In[39]: college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(['m
ean', pct_between], low=100, high=1000)
-----
-----
TypeError                                     Traceback (most recent
call last)
<ipython-input-39-3e3e18919cf9> in <module>()
----> 1 college.groupby(['STABBR', 'RELAFFIL'])['UGDS'].agg(['me
an', pct_between], low=100, high=1000)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
    2871         if hasattr(func_or_funcs, '__iter__'):
```

```

2872             ret = self._aggregate_multiple_funcs(func_or
_funcs,
-> 2873                                         (_level
or 0) + 1)
2874         else:
2875             cyfunc = self._is_cython_func(func_or_funcs)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_multiple_funcs(self, arg, _level)
2944             obj._reset_cache()
2945             obj._selection = name
-> 2946             results[name] = obj.aggregate(func)
2947
2948             if isinstance(list(compat.itervalues(results))[0
],
[

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
2878
2879             if self.grouper.nkeys > 1:
-> 2880                 return self._python_agg_general(func_or_
funcs, *args, **kwargs)
2881
2882             try:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _python_agg_general(self, func, *args, **kwargs)
852
853             if len(output) == 0:
--> 854                 return self._python_apply_general(f)
855
856             if self.grouper._filter_empty_groups:

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _python_apply_general(self, f)
718     def _python_apply_general(self, f):
719         keys, values, mutated = self.grouper.apply(f, se
lf._selected_obj,
--> 720                                         self.
axis)
721
722         return self._wrap_applied_output(
[

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in apply(self, f, data, axis)
1800             # group might be modified
1801             group_axes = _get_axes(group)
-> 1802             res = f(group)
1803             if not _is_indexed_like(res, group_axes):
1804                 mutated = True

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in <lambda>(x)

```

```

840     def _python_agg_general(self, func, *args, **kwargs):
841         :
842             func = self._is_builtin_func(func)
--> 842             f = lambda x: func(x, *args, **kwargs)
843
844             # iterate through "columns" ex exclusions to populate output dict

```

TypeError: pct_between() missing 2 required positional arguments : 'low' and 'high'

用闭包自定义聚合函数

```

In[40]: def make_agg_func(func, name, *args, **kwargs):
          def wrapper(x):
              return func(x, *args, **kwargs)
          wrapper.__name__ = name
          return wrapper

my_agg1 = make_agg_func(pct_between, 'pct_1_3k', low=10
00, high=3000)
my_agg2 = make_agg_func(pct_between, 'pct_10_30k', 10000
, 30000)[ 'UGDS' ].agg(pct_between, 1000, high=10000).head(9)
Out[41]:

```

		mean	pct_1_3k	pct_10_30k
STABBR	RELAFFIL			
AK	0	3508.857143	0.142857	0.142857
	1	123.333333	0.000000	0.000000
AL	0	3248.774648	0.236111	0.083333
	1	979.722222	0.333333	0.000000
AR	0	1793.691176	0.279412	0.014706

6. 检查分组对象

```
# 查看分组对象的类型
In[42]: college = pd.read_csv('data/college.csv')
         grouped = college.groupby(['STABBR', 'RELAFFIL'])
         type(grouped)
Out[42]: pandas.core.groupby.DataFrameGroupBy
```

```
# 用dir函数找到该对象所有的可用函数
In[43]: print([attr for attr in dir(grouped) if not attr.startswith('_')])
['CITY', 'CURROPER', 'DISTANCEONLY', 'GRAD_DEBT_MDN_SUPP', 'HBCU',
 'INSTNM', 'MD_EARN_WNE_P10', 'MENONLY', 'PCTFLOAN', 'PCTPELL',
 'PPTUG_EF', 'RELAFFIL', 'SATMTMID', 'SATVRMID', 'STABBR', 'UG25A
 BV', 'UGDS', 'UGDS_2MOR', 'UGDS_AIAN', 'UGDS_ASIAN', 'UGDS_BLACK',
 'UGDS_HISP', 'UGDS_NHPI', 'UGDS_NRA', 'UGDS_UNKN', 'UGDS_WHITE',
 'WOMENONLY', 'agg', 'aggregate', 'all', 'any', 'apply', 'backf
 ill', 'bfill', 'boxplot', 'corr', 'corrwith', 'count', 'cov', 'c
 umcount', 'cummax', 'cummin', 'cumprod', 'cumsum', 'describe',
 'diff', 'dtypes', 'expanding', 'ffill', 'fillna', 'filter', 'firs
 t', 'get_group', 'groups', 'head', 'hist', 'idxmax', 'idxmin',
 'indices', 'last', 'mad', 'max', 'mean', 'median', 'min', 'ndim',
 'ngroup', 'ngroups', 'nth', 'nunique', 'ohlc', 'pad', 'pct_chang
 e', 'plot', 'prod', 'quantile', 'rank', 'resample', 'rolling',
 'sem', 'shift', 'size', 'skew', 'std', 'sum', 'tail', 'take', 'tr
 ansform', 'tshift', 'var']
```

```
# 用ngroups属性查看分组的数量
In[44]: grouped.ngroups
Out[44]: 112
```

```
# 查看每个分组的唯一识别标签，groups属性是一个字典，包含每个独立分组与行索
引标签的对应
In[45]: groups = list(grouped.groups.keys())
         groups[:6]
Out[45]: [('AK', 0), ('AK', 1), ('AL', 0), ('AL', 1), ('AR', 0),
          ('AR', 1)]
```

```
# 用get_group，传入分组标签的元组。例如，获取佛罗里达州所有与宗教相关的学校
In[46]: grouped.get_group(('FL', 1)).head()
Out[46]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...
712	The Baptist College of Florida	Graceville	FL	0.0	0.0	0.0	1	545.0	465.0	0.0	...
713	Barry University	Miami	FL	0.0	0.0	0.0	1	470.0	462.0	0.0	...
714	Gooding Institute of Nurse Anesthesia	Panama City	FL	0.0	0.0	0.0	1	NaN	NaN	0.0	...
715	Bethune-Cookman University	Daytona Beach	FL	1.0	0.0	0.0	1	405.0	395.0	0.0	...
724	Johnson University Florida	Kissimmee	FL	0.0	0.0	0.0	1	480.0	470.0	0.0	...

5 rows × 27 columns

< >

groupby对象是一个可迭代对象，可以挨个查看每个独立分组

In[47]: from IPython.display import display

In[48]: i = 0

```

for name, group in grouped:
    print(name)
    display(group.head(2))
    i += 1
    if i == 5:
        break

```

('AK', 0)

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...	UGDS
60	University of Alaska Anchorage	Anchorage	AK	0.0	0.0	0.0	0	NaN	NaN	0.0	...	
62	University of Alaska Fairbanks	Fairbanks	AK	0.0	0.0	0.0	0	NaN	NaN	0.0	...	

2 rows × 27 columns

< >

('AK', 1)

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...	UGDS
61	Alaska Bible College	Palmer	AK	0.0	0.0	0.0	1	NaN	NaN	0.0	...	
64	Alaska Pacific University	Anchorage	AK	0.0	0.0	0.0	1	555.0	503.0	0.0	...	

2 rows × 27 columns

< >

('AL', 0)

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...	UGDS
0	Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	420.0	0.0	...	
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	565.0	0.0	...	

('AL', 1)

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...	UGDS
2	Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	1.0	...	
10	Birmingham Southern College	Birmingham	AL	0.0	0.0	0.0	1	560.0	560.0	0.0	...	

2 rows × 27 columns

< >

('AR', 0)

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...	UGDS
128	University of Arkansas at Little Rock	Little Rock	AR	0.0	0.0	0.0	0	470.0	510.0	0.0	...	
129	University of Arkansas for Medical Sciences	Little Rock	AR	0.0	0.0	0.0	0	NaN	NaN	0.0	...	

2 rows × 27 columns

< >

```
# groupby对象使用head方法，可以在一个DataFrame中显示每个分组的头几行
In[49]: grouped.head(2).head(6)
Out[49]:
```

	INSTNM	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	...
0	Alabama A & M University	Normal	AL	1.0	0.0	0.0	0	424.0	420.0	0.0	...
1	University of Alabama at Birmingham	Birmingham	AL	0.0	0.0	0.0	0	570.0	565.0	0.0	...
2	Amridge University	Montgomery	AL	0.0	0.0	0.0	1	NaN	NaN	1.0	...
10	Birmingham Southern College	Birmingham	AL	0.0	0.0	0.0	1	560.0	560.0	0.0	...
43	Prince Institute-Southeast	Elmhurst	IL	0.0	0.0	0.0	0	NaN	NaN	0.0	...
60	University of Alaska Anchorage	Anchorage	AK	0.0	0.0	0.0	0	NaN	NaN	0.0	...

6 rows × 27 columns

更多

```
# nth方法可以选出每个分组指定行的数据，下面选出的是第1行和最后1行
In[50]: grouped.nth([1, -1]).head(8)
Out[50]:
```

STABBR	RELAFFIL	CITY	CURROPER	DISTANCEONLY	GRAD_DEBT_MDN_SUPP	HBCU	INSTNM	MD_EARN_WNE_P
AK	0	Fairbanks	1	0.0	19355	0.0	University of Alaska Fairbanks	3621
	0	Barrow	1	0.0	PrivacySuppressed	0.0	Ilisagvik College	2491
	1	Anchorage	1	0.0	23250	0.0	Alaska Pacific University	4701
	1	Soldotna	1	0.0	PrivacySuppressed	0.0	Alaska Christian College	Nan
AL	0	Birmingham	1	0.0	21941.5	0.0	University of Alabama at Birmingham	3971
	0	Dothan	1	0.0	PrivacySuppressed	0.0	Alabama College of Osteopathic Medicine	Nan
	1	Birmingham	1	0.0	27000	0.0	Birmingham Southern College	4421
	1	Huntsville	1	NaN	36173.5	NaN	Strayer University-Huntsville Campus	4921

8 rows × 25 columns

< ⏪ ⏩ >

7. 过滤状态

```
In[51]: college = pd.read_csv('data/college.csv', index_col='INSTNM')
grouped = college.groupby('STABBR')
grouped.ngroups
Out[51]: 59
```

```
# 这等于求出不同州的个数，nunique()可以得到同样的结果
In[52]: college['STABBR'].nunique()
Out[52]: 59
```

```
# 自定义一个计算少数民族学生总比例的函数，如果比例大于阈值，还返回True
In[53]: def check_minority(df, threshold):
    minority_pct = 1 - df['UGDS_WHITE']
    total_minority = (df['UGDS'] * minority_pct).sum()
    total_ugds = df['UGDS'].sum()
    total_minority_pct = total_minority / total_ugds
    return total_minority_pct > threshold

# grouped变量有一个filter方法，可以接收一个自定义函数，决定是否保留一个分组

In[54]: college_filtered = grouped.filter(check_minority, threshold=.5)
college_filtered.head()

Out[54]:
```

	CITY	STABBR	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS
INSTNM										
Everest College-Phoenix	Phoenix	AZ	0.0	0.0	0.0	1	NaN	NaN	0.0	4102.0
Collins College	Phoenix	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	83.0
Empire Beauty School-Paradise Valley	Phoenix	AZ	0.0	0.0	0.0	1	NaN	NaN	0.0	25.0
Empire Beauty School-Tucson	Tucson	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	126.0
Thunderbird School of Global Management	Glendale	AZ	0.0	0.0	0.0	0	NaN	NaN	0.0	1.0

5 rows × 26 columns

```
# 通过查看形状，可以看到过滤了60%，只有20个州的少数民族学生占据多数
In[55]: college.shape
Out[55]: (7535, 26)

In[56]: college_filtered.shape
Out[56]: (3028, 26)

In[57]: college_filtered['STABBR'].nunique()
Out[57]: 20
```

更多

```
# 用一些不同的阈值，检查形状和不同州的个数
In[58]: college_filtered_20 = grouped.filter(check_minority, threshold=.2)
college_filtered_20.shape
Out[58]: (7461, 26)

In[59]: college_filtered_20['STABBR'].nunique()
Out[59]: 57

In[60]: college_filtered_70 = grouped.filter(check_minority, threshold=.7)
college_filtered_70.shape
Out[60]: (957, 26)

In[61]: college_filtered_70['STABBR'].nunique()
Out[61]: 10

In[62]: college_filtered_95 = grouped.filter(check_minority, threshold=.95)
college_filtered_95.shape
Out[62]: (156, 26)
```

8. 减肥对赌

```
# 读取减肥数据集，查看一月的数据
In[63]: weight_loss = pd.read_csv('data/weight_loss.csv')
weight_loss.query('Month == "Jan"')
Out[63]:
```

	Name	Month	Week	Weight
0	Bob	Jan	Week 1	291
1	Amy	Jan	Week 1	197
2	Bob	Jan	Week 2	288
3	Amy	Jan	Week 2	189
4	Bob	Jan	Week 3	283
5	Amy	Jan	Week 3	189
6	Bob	Jan	Week 4	283
7	Amy	Jan	Week 4	190

```
# 定义一个求减肥比例的函数
In[64]: def find_perc_loss(s):
    return (s - s.iloc[0]) / s.iloc[0]
# 查看Bob在一月的减肥成果
In[65]: bob_jan = weight_loss.query('Name=="Bob" and Month=="Jan"')
         find_perc_loss(bob_jan['Weight'])
Out[65]: 0      0.000000
          2     -0.010309
          4     -0.027491
          6     -0.027491
Name: Weight, dtype: float64
```

```
# 对Name和Month进行分组，然后使用transform方法，传入函数，对数值进行转换
In[66]: pcnt_loss = weight_loss.groupby(['Name', 'Month'])['Weight'].transform(find_perc_loss)
         pcnt_loss.head(8)
Out[66]: 0      0.000000
          1      0.000000
          2     -0.010309
          3     -0.040609
          4     -0.027491
          5     -0.040609
          6     -0.027491
          7     -0.035533
Name: Weight, dtype: float64
```

```
# transform之后的结果，行数不变，可以赋值给原始DataFrame作为一个新列；
# 为了缩短输出，只选择Bob的前两个月数据
In[67]: weight_loss['Perc Weight Loss'] = pcnt_loss.round(3)
         weight_loss.query('Name=="Bob" and Month in ["Jan", "Feb"]')
Out[67]:
```

	Name	Month	Week	Weight	Perc Weight Loss
0	Bob	Jan	Week 1	291	0.000
2	Bob	Jan	Week 2	288	-0.010
4	Bob	Jan	Week 3	283	-0.027
6	Bob	Jan	Week 4	283	-0.027
8	Bob	Feb	Week 1	283	0.000
10	Bob	Feb	Week 2	275	-0.028
12	Bob	Feb	Week 3	268	-0.053
14	Bob	Feb	Week 4	268	-0.053

```
# 因为最重要的是每个月的第4周，只选择第4周的数据
```

```
In[68]: week4 = weight_loss.query('Week == "Week 4"')
week4
```

```
Out[68]:
```

	Name	Month	Week	Weight	Perc Weight Loss
6	Bob	Jan	Week 4	283	-0.027
7	Amy	Jan	Week 4	190	-0.036
14	Bob	Feb	Week 4	268	-0.053
15	Amy	Feb	Week 4	173	-0.089
22	Bob	Mar	Week 4	261	-0.026
23	Amy	Mar	Week 4	170	-0.017
30	Bob	Apr	Week 4	250	-0.042
31	Amy	Apr	Week 4	161	-0.053

```
# 用pivot重构DataFrame，让Amy和Bob的数据并排放置
```

```
In[69]: winner = week4.pivot(index='Month', columns='Name', values='Perc Weight Loss')
winner
```

```
Out[69]:
```

Name	Amy	Bob
Month		
Apr	-0.053	-0.042
Feb	-0.089	-0.053
Jan	-0.036	-0.027
Mar	-0.017	-0.026

```
# 用where方法选出每月的赢家
In[70]: winner['Winner'] = np.where(winner['Amy'] < winner['Bob'],
    ['Amy', 'Bob'])
winner.style.highlight_min(axis=1)
Out[70]:
```

Name	Amy	Bob	Winner
Month			
Apr	-0.053	-0.042	Amy
Feb	-0.089	-0.053	Amy
Jan	-0.036	-0.027	Amy
Mar	-0.017	-0.026	Bob

```
# 用value_counts()返回最后的比分
In[71]: winner.Winner.value_counts()
Out[71]: Amy      3
          Bob     1
Name: Winner, dtype: int64
```

更多

```
# Pandas默认是按字母排序的
In[72]: week4a = week4.copy()
month_chron = week4a['Month'].unique()
month_chron
Out[72]: array(['Jan', 'Feb', 'Mar', 'Apr'], dtype=object)
```

```
# 转换为Categorical变量，可以做成按时间排序
In[73]: week4a['Month'] = pd.Categorical(week4a['Month'],
                                         categories=month_chron,
                                         ordered=True)
week4a.pivot(index='Month', columns='Name', values='Per
c Weight Loss')
Out[73]:
```

Name	Amy	Bob
Month		
Jan	-0.036	-0.027
Feb	-0.089	-0.053
Mar	-0.017	-0.026
Apr	-0.053	-0.042

9. 用apply计算每州的加权平均SAT分数

```
# 读取college，'UGDS', 'SATMTMID', 'SATVRMID'三列如果有缺失值则删除行
In[74]: college = pd.read_csv('data/college.csv')
subset = ['UGDS', 'SATMTMID', 'SATVRMID']
college2 = college.dropna(subset=subset)
college.shape
Out[74]: (7535, 27)

In[75]: college2.shape
Out[75]: (1184, 27)
```

```
# 自定义一个求SAT数学成绩的加权平均值的函数
In[76]: def weighted_math_average(df):
    weighted_math = df['UGDS'] * df['SATMTMID']
    return int(weighted_math.sum() / df['UGDS'].sum())
# 按州分组，并调用apply方法，传入自定义函数
In[77]: college2.groupby('STABBR').apply(weighted_math_average)
.head()
Out[77]: STABBR
          AK    503
          AL    536
          AR    529
          AZ    569
          CA    564
dtype: int64
```

效果同上

```
In[78]: college2.groupby('STABBR').agg(weighted_math_average).head()
Out[78]:
```

	INSTNM	CITY	HBCU	MENONLY	WOMENONLY	RELAFFIL	SATVRMID	SATMTMID	DISTANCEONLY	UGDS	...	UGD
STABBR	AK	503	503	503	503	503	503	503	503	503	503	...
AK	503	503	503	503	503	503	503	503	503	503	503	...
AL	536	536	536	536	536	536	536	536	536	536	536	...
AR	529	529	529	529	529	529	529	529	529	529	529	...
AZ	569	569	569	569	569	569	569	569	569	569	569	...
CA	564	564	564	564	564	564	564	564	564	564	564	...

5 rows × 26 columns

```
# 如果将列限制到SATMTMID，会报错。这是因为不能访问UGDS。
In[79]: college2.groupby('STABBR')['SATMTMID'].agg(weighted_math_average)
-----
-----
TypeError                                 Traceback (most recent call last)
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.Int64HashTable.get_item (pandas/_libs/hashtable.c:14010)()

TypeError: an integer is required
```

During handling of the above exception, another exception occurred:

```

KeyError                                     Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in agg_series(self, obj, func)
    2177         try:
-> 2178             return self._aggregate_series_fast(obj, func
)
    2179         except Exception:

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_series_fast(self, obj, func)
    2197                     dummy)
-> 2198             result, counts = grouper.get_result()
    2199             return result, counts

```

```

pandas/_libs/src/reduce.pyx in pandas._libs.lib.SeriesGrouper.ge
t_result (pandas/_libs/lib.c:39105)()

```

```

pandas/_libs/src/reduce.pyx in pandas._libs.lib.SeriesGrouper.ge
t_result (pandas/_libs/lib.c:38888)()

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in <lambda>(x)
    841         func = self._is_builtin_func(func)
--> 842         f = lambda x: func(x, *args, **kwargs)
    843

```

```

<ipython-input-76-01eb90aa258d> in weighted_math_average(df)
      1 def weighted_math_average(df):
----> 2     weighted_math = df['UGDS'] * df['SATMTMID']
      3     return int(weighted_math.sum() / df['UGDS'].sum())

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/seri
es.py in __getitem__(self, key)
    600         try:
--> 601             result = self.index.get_value(self, key)
    602

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in get_value(self, series, key)
    2476             return self._engine.get_value(s, k,
-> 2477                                         tz=getattr(ser
ies.dtype, 'tz', None))
    2478         except KeyError as e1:

```

```

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_val
ue (pandas/_libs/index.c:4404)()

```

```

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_val
ue (pandas/_libs/index.c:4087)()

```

```

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc

```

```
(pandas/_libs/index.c:5210)()

KeyError: 'UGDS'

During handling of the above exception, another exception occurred:

TypeError                                 Traceback (most recent
call last)
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable
.Int64HashTable.get_item (pandas/_libs/hashtable.c:14010)()

TypeError: an integer is required

During handling of the above exception, another exception occurred:

KeyError                                 Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
    2882         try:
-> 2883             return self._python_agg_general(func_or_
funcs, *args, **kwargs)
    2884         except Exception:
        /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _python_agg_general(self, func, *args, **kwargs)
    847         try:
--> 848             result, counts = self.grouper.agg_series
(obj, f)
    849             output[name] = self._try_cast(result, obj
, numeric_only=True)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in agg_series(self, obj, func)
    2179         except Exception:
-> 2180             return self._aggregate_series_pure_python(ob
j, func)
    2181

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_series_pure_python(self, obj, func)
    2210         for label, group in splitter:
-> 2211             res = func(group)
    2212             if result is None:
        /Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in <lambda>(x)
    841             func = self._is_builtin_func(func)
```

```

--> 842           f = lambda x: func(x, *args, **kwargs)
843

<ipython-input-76-01eb90aa258d> in weighted_math_average(df)
    1 def weighted_math_average(df):
--> 2     weighted_math = df['UGDS'] * df['SATMTMID']
    3     return int(weighted_math.sum() / df['UGDS'].sum())

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/series.py in __getitem__(self, key)
600         try:
--> 601             result = self.index.get_value(self, key)
602

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_value(self, series, key)
2476             return self._engine.get_value(s, k,
-> 2477                                         tz=getattr(series.dtype, 'tz', None))
2478         except KeyError as e1:
2479             if isna(key) or key == ...:
2480                 return missing
2481             else:
2482                 raise

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value(pandas/_libs/index.c:4404)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value(pandas/_libs/index.c:4087)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc(pandas/_libs/index.c:5210)()

KeyError: 'UGDS'

```

During handling of the above exception, another exception occurred:

```

TypeError                                 Traceback (most recent
call last)
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5126)()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.
Int64HashTable.get_item (pandas/_libs/hashtable.c:14010)()

TypeError: an integer is required

```

During handling of the above exception, another exception occurred:

```

KeyError                                 Traceback (most recent
call last)
<ipython-input-79-1351e4f306c7> in <module>()
--> 1 college2.groupby('STABBR')['SATMTMID'].agg(weighted_math
_average)

```

```

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in aggregate(self, func_or_funcs, *args, **kwargs)
    2883             return self._python_agg_general(func_or_f
uncs, *args, **kwargs)
    2884         except Exception:
-> 2885             result = self._aggregate_named(func_or_f
uncs, *args, **kwargs)
    2886
    2887         index = Index(sorted(result), name=self.grou
per.names[0])

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/grou
pby.py in _aggregate_named(self, func, *args, **kwargs)
    3013         for name, group in self:
    3014             group.name = name
-> 3015             output = func(group, *args, **kwargs)
    3016             if isinstance(output, (Series, Index, np.ndar
ray)):
    3017                 raise Exception('Must produce aggregated
value')

<ipython-input-76-01eb90aa258d> in weighted_math_average(df)
    1 def weighted_math_average(df):
----> 2     weighted_math = df['UGDS'] * df['SATMTMID']
    3     return int(weighted_math.sum() / df['UGDS'].sum())

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/seri
es.py in __getitem__(self, key)
    599         key = com._apply_if_callable(key, self)
    600         try:
--> 601             result = self.index.get_value(self, key)
    602
    603             if not is_scalar(result):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/base.py in get_value(self, series, key)
    2475         try:
    2476             return self._engine.get_value(s, k,
-> 2477                                         tz=getattr(ser
ies.dtype, 'tz', None))
    2478         except KeyError as e1:
    2479             if len(self) > 0 and self.inferred_type in ['
integer', 'boolean']:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_val
ue (pandas/_libs/index.c:4404)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_val
ue (pandas/_libs/index.c:4087)()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc
(pandas/_libs/index.c:5210)()

```

```
KeyError: 'UGDS'
```

```
# apply的一个不错的功能是通过返回Series，创建多个新的列
In[80]: from collections import OrderedDict
def weighted_average(df):
    data = OrderedDict()
    weight_m = df['UGDS'] * df['SATMTMID']
    weight_v = df['UGDS'] * df['SATVRMID']

    data['weighted_math_avg'] = weight_m.sum() / df['UGDS'].sum()
    data['weighted_verbal_avg'] = weight_v.sum() / df['UGDS'].sum()
    data['math_avg'] = df['SATMTMID'].mean()
    data['verbal_avg'] = df['SATVRMID'].mean()
    data['count'] = len(df)
    return pd.Series(data, dtype='int')

college2.groupby('STABBR').apply(weighted_average).head(10)
```

Out[80]:

STABBR	weighted_math_avg	weighted_verbal_avg	math_avg	verbal_avg	count
AK	503	555	503	555	1
AL	536	533	504	508	21
AR	529	504	515	491	16
AZ	569	557	536	538	6
CA	564	539	562	549	72
CO	553	547	540	537	14
CT	545	533	522	517	14
DC	621	623	588	589	6
DE	569	553	495	486	3
FL	565	565	521	529	38

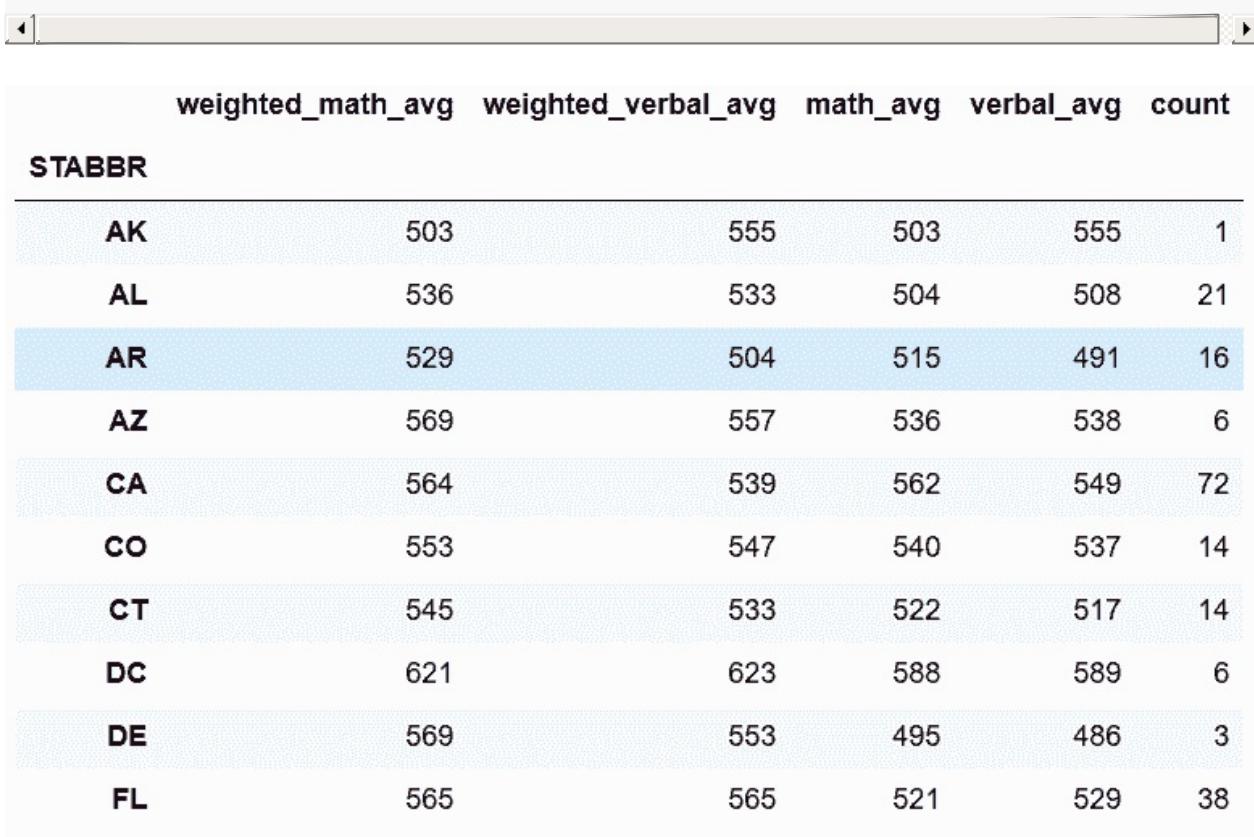
```
# 多创建两个新的列
In[81]: from collections import OrderedDict
def weighted_average(df):
    data = OrderedDict()
    weight_m = df['UGDS'] * df['SATMTMID']
    weight_v = df['UGDS'] * df['SATVRMID']

    wm_avg = weight_m.sum() / df['UGDS'].sum()
    wv_avg = weight_v.sum() / df['UGDS'].sum()

    data['weighted_math_avg'] = wm_avg
    data['weighted_verbal_avg'] = wv_avg
    data['math_avg'] = df['SATMTMID'].mean()
    data['verbal_avg'] = df['SATVRMID'].mean()
    data['count'] = len(df)
    return pd.Series(data, dtype='int')

college2.groupby('STABBR').apply(weighted_average).head(10)
```

Out[81]:



STABBR	weighted_math_avg	weighted_verbal_avg	math_avg	verbal_avg	count
AK	503	555	503	555	1
AL	536	533	504	508	21
AR	529	504	515	491	16
AZ	569	557	536	538	6
CA	564	539	562	549	72
CO	553	547	540	537	14
CT	545	533	522	517	14
DC	621	623	588	589	6
DE	569	553	495	486	3
FL	565	565	521	529	38

更多

```
# 自定义一个返回DataFrame的函数，使用NumPy的函数average计算加权平均值，  
使用SciPy的gmean和hmean计算几何和调和平均值  
In[82]: from scipy.stats import gmean, hmean  
def calculate_means(df):  
    df_means = pd.DataFrame(index=['Arithmetic', 'Weighted', 'Geometric', 'Harmonic'])  
    cols = ['SATMTMID', 'SATVRMID']  
    for col in cols:  
        arithmetic = df[col].mean()  
        weighted = np.average(df[col], weights=df['UGDS'])  
    geometric = gmean(df[col])  
    harmonic = hmean(df[col])  
    df_means[col] = [arithmetic, weighted, geometric, harmonic]  
  
    df_means['count'] = len(df)  
    return df_means.astype(int)  
  
college2.groupby('STABBR').filter(lambda x: len(x) != 1).groupby('STABBR').apply(calculate_means).head(10)
```

Out[82]:

		SATMTMID	SATVRMID	count
		STABBR		
AL	Arithmetic	504	508	21
	Weighted	536	533	21
	Geometric	500	505	21
	Harmonic	497	502	21
AR	Arithmetic	515	491	16
	Weighted	529	504	16
	Geometric	514	489	16
AZ	Harmonic	513	487	16
	Arithmetic	536	538	6
	Weighted	569	557	6

10. 用连续变量分组

```
In[83]: flights = pd.read_csv('data/flights.csv')
flights.head()
Out[83]:
```

MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225

```
# 判断DIST列有无缺失值
In[84]: flights.DIST.hasnans
Out[84]: False
```

```
# 再次删除DIST列的缺失值（原书是没有这两段的）
In[85]: flights.dropna(subset=['DIST']).shape
Out[85]: (58492, 14)
```

```
# 使用Pandas的cut函数，将数据分成5个面元
In[86]: bins = [-np.inf, 200, 500, 1000, 2000, np.inf]
          cuts = pd.cut(flights['DIST'], bins=bins)
          cuts.head()
Out[86]: 0      (500.0, 1000.0]
        1      (1000.0, 2000.0]
        2      (500.0, 1000.0]
        3      (1000.0, 2000.0]
        4      (1000.0, 2000.0]
        Name: DIST, dtype: category
        Categories (5, interval[float64]): [(-inf, 200.0] < (200.0, 500.0] < (500.0, 1000.0] < (1000.0, 2000.0] < (2000.0, inf]
```

```
# 对每个面元进行统计
In[87]: cuts.value_counts()
Out[87]: (500.0, 1000.0]    20659
          (200.0, 500.0]     15874
          (1000.0, 2000.0]   14186
          (2000.0, inf]      4054
          (-inf, 200.0]      3719
Name: DIST, dtype: int64
```

```
# 面元Series可以用来进行分组
In[88]: flights.groupby(cuts)[ 'AIRLINE' ].value_counts(normalize=True).round(3).head(15)
Out[88]: DIST           AIRLINE
          (-inf, 200.0]  OO      0.326
                           EV      0.289
                           MQ      0.211
                           DL      0.086
                           AA      0.052
                           UA      0.027
                           WN      0.009
          (200.0, 500.0]  WN      0.194
                           DL      0.189
                           OO      0.159
                           EV      0.156
                           MQ      0.100
                           AA      0.071
                           UA      0.062
                           VX      0.028
Name: AIRLINE, dtype: float64
```

原理

```
In[89]: flights.groupby(cuts)[ 'AIRLINE' ].value_counts(normalize=True)[ 'AIRLINE' ].value_counts(normalize=True).round(3).head(15)
Out[89]:
DIST           AIRLINE
          (-inf, 200.0]  OO      0.325625
                           EV      0.289325
                           MQ      0.210809
                           DL      0.086045
                           AA      0.052165
                           UA      0.027427
                           WN      0.008604
          (200.0, 500.0]  WN      0.193902
                           DL      0.188736
                           OO      0.158687
                           EV      0.156293
```

	MQ	0.100164
	AA	0.071375
	UA	0.062051
	VX	0.028222
	US	0.016001
	NK	0.011843
	B6	0.006867
	F9	0.004914
	AS	0.000945
(500.0, 1000.0]	DL	0.205625
	AA	0.143908
	WN	0.138196
	UA	0.131129
	OO	0.106443
	EV	0.100683
	MQ	0.051213
	F9	0.038192
	NK	0.029527
	US	0.025316
	AS	0.023234
	VX	0.003582
	B6	0.002953
(1000.0, 2000.0]	AA	0.263781
	UA	0.199070
	DL	0.165092
	WN	0.159664
	OO	0.046454
	NK	0.045115
	US	0.040462
	F9	0.030664
	AS	0.015931
	EV	0.015579
	VX	0.012125
	B6	0.003313
	MQ	0.002749
(2000.0, inf]	UA	0.289097
	AA	0.211643
	DL	0.171436
	B6	0.080414
	VX	0.073754
	US	0.065121
	WN	0.046374
	HA	0.027627
	NK	0.019240
	AS	0.011593
	F9	0.003700

Name: AIRLINE, dtype: float64



更多

```
# 求飞行时间的0.25, 0.5, 0.75分位数
In[90]: flights.groupby(cuts)[ 'AIR_TIME' ].quantile(q=[ .25, .5,
.75]).div(60).round(2)
Out[90]: DIST
(-inf, 200.0]      0.25    0.43
                  0.50    0.50
                  0.75    0.57
(200.0, 500.0]     0.25    0.77
                  0.50    0.92
                  0.75    1.05
(500.0, 1000.0]    0.25   1.43
                  0.50   1.65
                  0.75   1.92
(1000.0, 2000.0]   0.25   2.50
                  0.50   2.93
                  0.75   3.40
(2000.0, inf]      0.25   4.30
                  0.50   4.70
                  0.75   5.03
Name: AIR_TIME, dtype: float64
```

```
# unstack方法可以将内层的索引变为列名
In[91]: labels=['Under an Hour', '1 Hour', '1-2 Hours', '2-4 Hours',
'4+ Hours']
         cuts2 = pd.cut(flights['DIST'], bins=bins, labels=labels)
         flights.groupby(cuts2)[ 'AIRLINE' ].value_counts(normalize=True).round(3).unstack().style.highlight_max(axis=1)
Out[91]:
```

AIRLINE	AA	AS	B6	DL	EV	F9	HA	MQ	NK	OO	UA	US	VX	WN
DIST														
Under an Hour	0.052	nan	nan	0.086	0.289	nan	nan	0.211	nan	0.326	0.027	nan	nan	0.009
1 Hour	0.071	0.001	0.007	0.189	0.156	0.005	nan	0.1	0.012	0.159	0.062	0.016	0.028	0.194
1-2 Hours	0.144	0.023	0.003	0.206	0.101	0.038	nan	0.051	0.03	0.106	0.131	0.025	0.004	0.138
2-4 Hours	0.264	0.016	0.003	0.165	0.016	0.031	nan	0.003	0.045	0.046	0.199	0.04	0.012	0.16
4+ Hours	0.212	0.012	0.08	0.171	nan	0.004	0.028	nan	0.019	nan	0.289	0.065	0.074	0.046

11. 计算城市之间的航班总数

```
In[92]: flights = pd.read_csv('data/flights.csv')
flights.head()
Out[92]:
```

MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225

< >

求每两个城市间的航班总数

In[93]: flights_ct = flights.groupby(['ORG_AIR', 'DEST_AIR']).size()

flights_ct.head()
Out[93]: ORG_AIR DEST_AIR
ATL ABE 31
ABQ 16
ABY 19
ACY 6
AEX 40
dtype: int64

选出休斯顿 (IAH) 和亚特兰大 (ATL) 之间双方向的航班总数

In[94]: flights_ct.loc[[('ATL', 'IAH'), ('IAH', 'ATL')]]

Out[94]: ORG_AIR DEST_AIR
ATL IAH 121
IAH ATL 148
dtype: int64

分别对每行按照出发地和目的地，按字母排序

In[95]: flights_sort = flights[['ORG_AIR', 'DEST_AIR']].apply(sorted, axis=1)

flights_sort.head()

Out[95]:

	ORG_AIR	DEST_AIR
0	LAX	SLC
1	DEN	IAD
2	DFW	VPS
3	DCA	DFW
4	LAX	MCI

```
# 因为现在每行都是独立排序的，列名存在问题。对列重命名，然后再计算所有城市间的航班数
```

```
In[96]: rename_dict = {'ORG_AIR':'AIR1', 'DEST_AIR':'AIR2'}
flights_sort = flights_sort.rename(columns=rename_dict)
flights_ct2 = flights_sort.groupby(['AIR1', 'AIR2']).size()
flights_ct2.head()
Out[96]: AIR1  AIR2
          ABE    ATL    31
                  ORD    24
          ABI    DFW    74
          ABQ    ATL    16
                  DEN    46
dtype: int64
```

```
# 找到亚特兰大和休斯顿之间的航班数
```

```
In[97]: flights_ct2.loc[('ATL', 'IAH')]
Out[97]: 269
```

```
# 如果调换顺序，则会出错
In[98]: flights_ct2.loc[('IAH', 'ATL')]
-----
-----
IndexingError                                 Traceback (most recent
    call last)
<ipython-input-98-56147a7d0bb5> in <module>()
----> 1 flights_ct2.loc[('IAH', 'ATL')]

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in __getitem__(self, key)
    1323         except (KeyError, IndexError):
    1324             pass
-> 1325         return self._getitem_tuple(key)
    1326     else:
    1327         key = com._apply_if_callable(key, self.obj)

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in _getitem_tuple(self, tup)
    839
    840     # no multi-index, so validate all of the indexers
--> 841     self._has_valid_tuple(tup)
    842
    843     # ugly hack for GH #836

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xing.py in _has_valid_tuple(self, key)
    186         for i, k in enumerate(key):
    187             if i >= self.obj.ndim:
--> 188                 raise IndexingError('Too many indexers')
    189             if not self._has_valid_type(k, i):
    190                 raise ValueError("Location based indexing
can only have [%s] "


IndexingError: Too many indexers
```

更多

```
# 用NumPy的sort函数可以大大提高速度
In[99]: data_sorted = np.sort(flights[['ORG_AIR', 'DEST_AIR']])
          data_sorted[:10]
Out[99]: array([['LAX', 'SLC'],
               ['DEN', 'IAD'],
               ['DFW', 'VPS'],
               ['DCA', 'DFW'],
               ['LAX', 'MCI'],
               ['IAH', 'SAN'],
               ['DFW', 'MSY'],
               ['PHX', 'SFO'],
               ['ORD', 'STL'],
               ['IAH', 'SJC']], dtype=object)
```

```
# 重新用DataFrame构造器创建一个DataFrame，检测其是否与flights_sorted相等
In[100]: flights_sort2 = pd.DataFrame(data_sorted, columns=['AIR1', 'AIR2'])
           fs_orig = flights_sort.rename(columns={'ORG_AIR':'AIR1',
               'DEST_AIR':'AIR2'})
           flights_sort2.equals(fs_orig)
Out[100]: True
```

```
# 比较速度
In[101]: %timeit flights_sort = flights[['ORG_AIR', 'DEST_AIR']]
           .apply(sorted, axis=1)
           7.82 s ± 189 ms per loop (mean ± std. dev. of 7 runs, 1
           loop each)

In[102]: %%timeit
           data_sorted = np.sort(flights[['ORG_AIR', 'DEST_AIR']])
           flights_sort2 = pd.DataFrame(data_sorted, columns=['AIR1', 'AIR2'])
           10.9 ms ± 325 µs per loop (mean ± std. dev. of 7 runs,
           100 loops each)
```

12. 找到持续最长的准时航班

```
# 创建一个Series
In[103]: s = pd.Series([1, 1, 1, 0, 1, 1, 1, 0])
           s
Out[103]: 0    1
          1    1
          2    1
          3    0
          4    1
          5    1
          6    1
          7    0
           dtype: int64
```

```
# 累积求和
In[104]: s1 = s.cumsum()
           s1
Out[104]: 0    1
          1    2
          2    3
          3    3
          4    4
          5    5
          6    6
          7    6
           dtype: int64
```

```
In[105]: s.mul(s1).diff()
Out[105]: 0      NaN
          1    1.0
          2    1.0
          3   -3.0
          4    4.0
          5    1.0
          6    1.0
          7   -6.0
           dtype: float64
```

```
# 将所有非负值变为缺失值
In[106]: s.mul(s1).diff().where(lambda x: x < 0)
Out[106]: 0      NaN
           1      NaN
           2      NaN
           3    -3.0
           4      NaN
           5      NaN
           6      NaN
           7   -6.0
dtype: float64
```

```
In[107]: s.mul(s1).diff().where(lambda x: x < 0).ffill().add(s1
, fill_value=0)
Out[107]: 0    1.0
           1    2.0
           2    3.0
           3    0.0
           4    1.0
           5    2.0
           6    3.0
           7    0.0
dtype: float64
```

```
# 创建一个准时的列 ON_TIME
In[108]: flights = pd.read_csv('data/flights.csv')
          flights['ON_TIME'] = flights['ARR_DELAY'].lt(15).astype
(int)
          flights[['AIRLINE', 'ORG_AIR', 'ON_TIME']].head(10)
Out[108]:
```

	AIRLINE	ORG_AIR	ON_TIME
0	WN	LAX	0
1	UA	DEN	1
2	MQ	DFW	0
3	AA	DFW	1
4	WN	LAX	0
5	UA	IAH	1
6	AA	DFW	0
7	F9	SFO	1
8	AA	ORD	1
9	UA	IAH	1

```
# 将之前的逻辑做成一个函数
```

```
In[109]: def max_streak(s):
    s1 = s.cumsum()
    return s.mul(s1).diff().where(lambda x: x < 0) \
        .ffill().add(s1, fill_value=0).max()
In[110]: flights.sort_values(['MONTH', 'DAY', 'SCHED_DEP']) \
    .groupby(['AIRLINE', 'ORG_AIR'])['ON_TIME'] \
    .agg(['mean', 'size', max_streak]).round(2).hea
d()
Out[110]:
```

		mean	size	max_streak
AIRLINE	ORG_AIR			
	ATL	0.82	233	15
	DEN	0.74	219	17
AA	DFW	0.78	4006	64
	IAH	0.80	196	24
	LAS	0.79	374	29

更多

```
# 求最长的延误航班
In[111]: def max_delay_streak(df):
    df = df.reset_index(drop=True)
    s = 1 - df['ON_TIME']
    s1 = s.cumsum()
    streak = s.mul(s1).diff().where(lambda x: x < 0) \
        .ffill().add(s1, fill_value=0)
    last_idx = streak.idxmax()
    first_idx = last_idx - streak.max() + 1
    df_return = df.loc[[first_idx, last_idx], ['MONTH',
    'DAY']]
    df_return['streak'] = streak.max()
    df_return.index = ['first', 'last']
    df_return.index.name='streak_row'
    return df_return
In[112]: flights.sort_values(['MONTH', 'DAY', 'SCHED_DEP']) \
    .groupby(['AIRLINE', 'ORG_AIR']) \
    .apply(max_delay_streak) \
    .sort_values(['streak', 'MONTH', 'DAY'], ascending=False,
g=[False, True, True]).head(10)
Out[112]:
```

			MONTH	DAY	streak
AIRLINE	ORG_AIR	streak_row			
AA	DFW	first	2.0	26.0	38.0
		last	3.0	1.0	38.0
	ORD	first	1.0	6.0	28.0
		last	1.0	12.0	28.0
MQ	DFW	first	2.0	21.0	25.0
		last	2.0	26.0	25.0
	ORD	first	6.0	7.0	15.0
		last	6.0	18.0	15.0
NK	ATL	first	12.0	23.0	14.0
		last	12.0	24.0	14.0

第08章 数据清理

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

1. 用stack清理变量值作为列名

```
# 加载state_fruit数据集
In[2]: state_fruit = pd.read_csv('data/state_fruit.csv', index_
col=0)
       state_fruit
out[2]:
```

	Apple	Orange	Banana
Texas	12	10	40
Arizona	9	7	12
Florida	0	14	190

```
# stack方法可以将所有列名，转变为垂直的一级行索引
In[3]: state_fruit.stack()
out[3]: Texas      Apple    12
           Orange   10
           Banana   40
      Arizona     Apple    9
           Orange   7
           Banana  12
      Florida     Apple    0
           Orange  14
           Banana 190
      dtype: int64
```

```
# 使用reset_index()，将结果变为DataFrame
In[4]: state_fruit_tidy = state_fruit.stack().reset_index()
       state_fruit_tidy
out[4]:
```

	level_0	level_1	0
0	Texas	Apple	12
1	Texas	Orange	10
2	Texas	Banana	40
3	Arizona	Apple	9
4	Arizona	Orange	7
5	Arizona	Banana	12
6	Florida	Apple	0
7	Florida	Orange	14
8	Florida	Banana	190

```
# 重命名列名
In[5]: state_fruit_tidy.columns = ['state', 'fruit', 'weight']
state_fruit_tidy
out[5]:
```

	state	fruit	weight
0	Texas	Apple	12
1	Texas	Orange	10
2	Texas	Banana	40
3	Arizona	Apple	9
4	Arizona	Orange	7
5	Arizona	Banana	12
6	Florida	Apple	0
7	Florida	Orange	14
8	Florida	Banana	190

```
# 也可以使用rename_axis给不同的行索引层级命名
In[6]: state_fruit.stack()\
         .rename_axis(['state', 'fruit'])\
out[6]: state      fruit
         Texas     Apple    12
                  Orange   10
                  Banana   40
         Arizona   Apple    9
                  Orange   7
                  Banana  12
         Florida   Apple    0
                  Orange  14
                  Banana  190
dtype: int64
```

```
# 再次使用reset_index方法
In[7]: state_fruit.stack()\
         .rename_axis(['state', 'fruit'])\
         .reset_index(name='weight')
out[7]:
```

	state	fruit	weight
0	Texas	Apple	12
1	Texas	Orange	10
2	Texas	Banana	40
3	Arizona	Apple	9
4	Arizona	Orange	7
5	Arizona	Banana	12
6	Florida	Apple	0
7	Florida	Orange	14
8	Florida	Banana	190

更多

```
# 读取state_fruit2数据集
In[8]: state_fruit2 = pd.read_csv('data/state_fruit2.csv')
state_fruit2
out[8]:
```

	State	Apple	Orange	Banana
0	Texas	12	10	40
1	Arizona	9	7	12
2	Florida	0	14	190

```
# 州名不在行索引的位置上，使用stack将所有列名变为一个长Series
In[9]: state_fruit2.stack()
out[9]: 0   State      Texas
          Apple     12
          Orange    10
          Banana    40
        1   State      Arizona
          Apple      9
          Orange     7
          Banana    12
        2   State      Florida
          Apple      0
          Orange    14
          Banana   190
dtype: object
```

```
# 先设定state作为行索引名，再stack，可以得到和前面相似的结果
In[10]: state_fruit2.set_index('State').stack()
out[10]:
0   State      Texas
    Apple      12
    Orange     10
    Banana     40
1   State      Arizona
    Apple       9
    Orange      7
    Banana     12
2   State      Florida
    Apple       0
    Orange     14
    Banana    190
dtype: object
```

2. 用melt清理变量值作为列名

```
# 读取state_fruit2数据集
In[11]: state_fruit2 = pd.read_csv('data/state_fruit2.csv')
state_fruit2
out[11]:
```

	State	Apple	Orange	Banana
0	Texas	12	10	40
1	Arizona	9	7	12
2	Florida	0	14	190

```
# 使用melt方法，将列传给id_vars和value_vars。melt可以将原先的列名作为变量，原先的值作为值。
In[12]: state_fruit2.melt(id_vars=['State'],
                           value_vars=['Apple', 'Orange', 'Banana'])
out[12]:
```

	State	variable	value
0	Texas	Apple	12
1	Arizona	Apple	9
2	Florida	Apple	0
3	Texas	Orange	10
4	Arizona	Orange	7
5	Florida	Orange	14
6	Texas	Banana	40
7	Arizona	Banana	12
8	Florida	Banana	190

```
# 随意设定一个行索引
In[13]: state_fruit2.index=list('abc')
state_fruit2.index.name = 'letter'
In[14]: state_fruit2
out[14]:
```

	State	Apple	Orange	Banana
letter				
a	Texas	12	10	40
b	Arizona	9	7	12
c	Florida	0	14	190

```
# var_name和value_name可以用来重命名新生成的变量列和值的列
In[15]: state_fruit2.melt(id_vars=['State'],
                           value_vars=['Apple', 'Orange', 'Banana'],
                           var_name='Fruit',
                           value_name='Weight')
out[15]:
```

	State	Fruit	Weight
0	Texas	Apple	12
1	Arizona	Apple	9
2	Florida	Apple	0
3	Texas	Orange	10
4	Arizona	Orange	7
5	Florida	Orange	14
6	Texas	Banana	40
7	Arizona	Banana	12
8	Florida	Banana	190

```
# 如果你想让所有值都位于一列，旧的列标签位于另一列，可以直接使用melt
```

```
In[16]: state_fruit2.melt()
```

```
out[16]:
```

	variable	value
0	State	Texas
1	State	Arizona
2	State	Florida
3	Apple	12
4	Apple	9
5	Apple	0
6	Orange	10
7	Orange	7
8	Orange	14
9	Banana	40
10	Banana	12
11	Banana	190

```
# 要指明id变量，只需使用id_vars参数  
In[17]: state_fruit2.melt(id_vars='State')  
out[17]:
```

	State	variable	value
0	Texas	Apple	12
1	Arizona	Apple	9
2	Florida	Apple	0
3	Texas	Orange	10
4	Arizona	Orange	7
5	Florida	Orange	14
6	Texas	Banana	40
7	Arizona	Banana	12
8	Florida	Banana	190

3. 同时stack多组变量

```
# 读取movie数据集，选取所有演员名和其Facebook likes
In[18]: movie = pd.read_csv('data/movie.csv')
         actor = movie[['movie_title', 'actor_1_name', 'actor_2_
name', 'actor_3_name',
                  'actor_1_facebook_likes', 'actor_2_facebook_likes'
, 'actor_3_facebook_likes']]
         actor.head()
out[18]:
```

	movie_title	actor_1_name	actor_2_name	actor_3_name	actor_1_facebook_likes	actor_2_facebook_likes	actor_3_facebook_likes
0	Avatar	CCH Pounder	Joel David Moore	Wes Studi	1000.0	936.0	855.0
1	Pirates of the Caribbean: At World's End	Johnny Depp	Orlando Bloom	Jack Davenport	40000.0	5000.0	1000.0
2	Spectre	Christoph Waltz	Rory Kinnear	Stephanie Sigman	11000.0	393.0	161.0
3	The Dark Knight Rises	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000.0	23000.0	23000.0
4	Star Wars: Episode VII - The Force Awakens	Doug Walker	Rob Walker	Nan	131.0	12.0	Nan

创建一个自定义函数，用来改变列名。wide_to_long要求分组的变量要有相同的数字结尾：

```
In[19]: def change_col_name(col_name):
    col_name = col_name.replace('_name', '')
    if 'facebook' in col_name:
        fb_idx = col_name.find('facebook')
        col_name = col_name[:5] + col_name[fb_idx - 1:]
    + col_name[5:fb_idx-1]
    return col_name
In[20]: actor2 = actor.rename(columns=change_col_name)
         actor2.head()
out[20]:
```

```
# 使用wide_to_long函数，同时stack两列actor和Facebook
In[21]: stubs = ['actor', 'actor_facebook_likes']
         actor2_tidy = pd.wide_to_long(actor2,
                                         stubnames=stubs,
                                         i=['movie_title'],
                                         j='actor_num',
                                         sep='_').reset_index()
         actor2_tidy.head()
out[21]:
```

	movie_title	actor_num	actor	actor_facebook_likes
0	Avatar	1	CCH Pounder	1000.0
1	Pirates of the Caribbean: At World's End	1	Johnny Depp	40000.0
2	Spectre	1	Christoph Waltz	11000.0
3	The Dark Knight Rises	1	Tom Hardy	27000.0
4	Star Wars: Episode VII - The Force Awakens	1	Doug Walker	131.0

更多

```
# 加载数据
In[22]: df = pd.read_csv('data/stackme.csv')
df
out[22]:
```

	State	Country	a1	b2	Test	d	e
0	TX	US	0.45	0.3	Test1	2	6
1	MA	US	0.03	1.2	Test2	9	7
2	ON	CAN	0.70	4.2	Test3	4	2

```
# 对列重命名
In[23]: df2 = df.rename(columns = {'a1':'group1_a1', 'b2':'group1_b2',
                                    'd':'group2_a1', 'e':'group2_b2'})
df2
out[23]:
```

	State	Country	group1_a1	group1_b2	Test	group2_a1	group2_b2
0	TX	US	0.45	0.3	Test1	2	6
1	MA	US	0.03	1.2	Test2	9	7
2	ON	CAN	0.70	4.2	Test3	4	2

```
# 设定stubnames=['group1', 'group2']，对任何数字都起作用
```

```
In[24]: pd.wide_to_long(df2,
                      stubnames=['group1', 'group2'],
                      i=['State', 'Country', 'Test'],
                      j='Label',
                      suffix='.+',
                      sep='_')
```

```
out[24]:
```

State	Country	Test	group1 group2	
			Label	
TX	US	Test1	a1	0.45
			b2	0.30
MA	US	Test2	a1	0.03
			b2	1.20
ON	CAN	Test3	a1	0.70
			b2	4.20

4. 反转stacked数据

```
# 读取college数据集，学校名作为行索引，只选取本科生的列
In[25]: usecol_func = lambda x: 'UGDS_' in x or x == 'INSTNM'
college = pd.read_csv('data/college.csv',
                      index_col='INSTNM',
                      usecols=usecol_func)

college.head()
out[25]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA	UGDS_UNKN
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059	
University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179	
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000	
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332	
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098	0.0243	

用stack方法，将所有水平列名，转化为垂直的行索引

```
In[26]: college_stacked = college.stack()
college_stacked.head(18)
```

out[26]: INSTNM

Alabama A & M University	UGDS_WHITE	0.0333
	UGDS_BLACK	0.9353
	UGDS_HISP	0.0055
	UGDS_ASIAN	0.0019
	UGDS_AIAN	0.0024
	UGDS_NHPI	0.0019
	UGDS_2MOR	0.0000
	UGDS_NRA	0.0059
	UGDS_UNKN	0.0138
University of Alabama at Birmingham	UGDS_WHITE	0.5922
	UGDS_BLACK	0.2600
	UGDS_HISP	0.0283
	UGDS_ASIAN	0.0518
	UGDS_AIAN	0.0022
	UGDS_NHPI	0.0007
	UGDS_2MOR	0.0368
	UGDS_NRA	0.0179
	UGDS_UNKN	0.0100

dtype: float64

unstack方法可以将其还原

```
In[27]: college_stacked.unstack().head()
out[27]:
```

INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA	U
Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059	
University of Alabama at Birmingham	roll output; double click to hide		0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179
Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000	
University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332	
Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098	0.0243	

```
# 另一种方式是先用melt，再用pivot。先加载数据，不指定行索引名
```

```
In[28]: college2 = pd.read_csv('data/college.csv',
                               usecols=usecol_func)
college2.head()
```

```
out[28]:
```

	INSTNM	UGDS_WHITE	UGDS_BLACK	UGDS_HISP	UGDS_ASIAN	UGDS_AIAN	UGDS_NHPI	UGDS_2MOR	UGDS_NRA
0	Alabama A & M University	0.0333	0.9353	0.0055	0.0019	0.0024	0.0019	0.0000	0.0059
1	University of Alabama at Birmingham	0.5922	0.2600	0.0283	0.0518	0.0022	0.0007	0.0368	0.0179
2	Amridge University	0.2990	0.4192	0.0069	0.0034	0.0000	0.0000	0.0000	0.0000
3	University of Alabama in Huntsville	0.6988	0.1255	0.0382	0.0376	0.0143	0.0002	0.0172	0.0332
4	Alabama State University	0.0158	0.9208	0.0121	0.0019	0.0010	0.0006	0.0098	0.0243

```
# 使用melt，将所有race列变为一列
```

```
In[29]: college_melted = college2.melt(id_vars='INSTNM',
                                         var_name='Race',
                                         value_name='Percentage')
```

```
college_melted.head()
```

```
out[29]:
```

	INSTNM	Race	Percentage
0	Alabama A & M University	UGDS_WHITE	0.0333
1	University of Alabama at Birmingham	UGDS_WHITE	0.5922
2	Amridge University	UGDS_WHITE	0.2990
3	University of Alabama in Huntsville	UGDS_WHITE	0.6988
4	Alabama State University	UGDS_WHITE	0.0158

用pivot还原

```
In[30]: melted_inv = college_melted.pivot(index='INSTNM',
                                         columns='Race',
                                         values='Percentage')
melted_inv.head()
Out[30]:
```

Race	UGDS_2MOR	UGDS_AIAN	UGDS_ASIAN	UGDS_BLACK	UGDS_HISP	UGDS_NHPI	UGDS_NRA	UGDS_UNKN	UG
INSTNM									
A & W Healthcare Educators	0.0000	0.0	0.0000	0.9750	0.0250	0.0	0.0000	0.0000	
A T Still University of Health Sciences	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ABC Beauty Academy	0.0000	0.0	0.9333	0.0333	0.0333	0.0	0.0000	0.0000	
ABC Beauty College Inc	0.0000	0.0	0.0000	0.6579	0.0526	0.0	0.0000	0.0000	
AI Miami International University of Art and Design	0.0018	0.0	0.0018	0.0198	0.4773	0.0	0.0025	0.4644	

```
# 用loc同时选取行和列，然后重置索引，可以获得和原先索引顺序一样的DataFrame
In[31]: college2_replication = melted_inv.loc[college2['INSTNM'],
], college2.columns[:]\n:]\n\ncollege2.equals(college2_replication)
out[31]: True
```

更多

```
# 使用最外层的行索引做unstack
```

```
In[32]: college.stack().unstack(0)
```

out[32]:

INSTNM	Alabama A & M University	University of Alabama at Birmingham	Amridge University	University of Alabama in Huntsville	Alabama State University	The University of Alabama	Central Alabama Community College	Athens State University	Auburn University at Montgomery	Al Univ
UGDS_WHITE	0.0333	0.5922	0.2990	0.6988	0.0158	0.7825	0.7255	0.7823	0.5328	C
UGDS_BLACK	0.9353	0.2600	0.4192	0.1255	0.9208	0.1119	0.2613	0.1200	0.3376	C
UGDS_HISP	0.0055	0.0283	0.0069	0.0382	0.0121	0.0348	0.0044	0.0191	0.0074	C
UGDS_ASIAN	0.0019	0.0518	0.0034	0.0376	0.0019	0.0106	0.0025	0.0053	0.0221	C
UGDS_AIAN	0.0024	0.0022	0.0000	0.0143	0.0010	0.0038	0.0044	0.0157	0.0044	C
UGDS_NHPI	0.0019	0.0007	0.0000	0.0002	0.0006	0.0009	0.0000	0.0010	0.0016	C
UGDS_2MOR	0.0000	0.0368	0.0000	0.0172	0.0098	0.0261	0.0000	0.0174	0.0297	C
UGDS_NRA	0.0059	0.0179	0.0000	0.0332	0.0243	0.0268	0.0000	0.0057	0.0397	C
UGDS_UNKN	0.0138	0.0100	0.2715	0.0350	0.0137	0.0026	0.0019	0.0334	0.0246	C

9 rows × 6874 columns

```
# 转置DataFrame更简单的方法是transpose()或T
```

```
In[33]: college.T
```

```
[33]:
```

INSTNM	Alabama A & M University	University of Alabama at Birmingham	Amridge University	University of Alabama in Huntsville	Alabama State University	The University of Alabama	Central Alabama Community College	Athens State University	Auburn University at Montgomery	A Unit
UGDS_WHITE	0.0333	0.5922	0.2990	0.6988	0.0158	0.7825	0.7255	0.7823	0.5328	
UGDS_BLACK	0.9353	0.2600	0.4192	0.1255	0.9208	0.1119	0.2613	0.1200	0.3376	
UGDS_HISP	0.0055	0.0283	0.0069	0.0382	0.0121	0.0348	0.0044	0.0191	0.0074	
UGDS_ASIAN	0.0019	0.0518	0.0034	0.0376	0.0019	0.0106	0.0025	0.0053	0.0221	
UGDS_AIAN	0.0024	0.0022	0.0000	0.0143	0.0010	0.0038	0.0044	0.0157	0.0044	
UGDS_NHPI	0.0019	0.0007	0.0000	0.0002	0.0006	0.0009	0.0000	0.0010	0.0016	
UGDS_2MOR	0.0000	0.0368	0.0000	0.0172	0.0098	0.0261	0.0000	0.0174	0.0297	
UGDS_NRA	0.0059	0.0179	0.0000	0.0332	0.0243	0.0268	0.0000	0.0057	0.0397	

5. 分组聚合后unstacking

```
# 读取employee数据集，求出每个种族的平均工资
In[34]: employee = pd.read_csv('data/employee.csv')
In[35]: employee.groupby('RACE')['BASE_SALARY'].mean().astype(int)
out[35]: RACE
          American Indian or Alaskan Native    60272
          Asian/Pacific Islander                61660
          Black or African American             50137
          Hispanic/Latino                      52345
          Others                               51278
          White                                64419
Name: BASE_SALARY, dtype: int64
```

```
# 对种族和性别分组，求平均工资
In[36]: agg = employee.groupby(['RACE', 'GENDER'])['BASE_SALARY'].mean().astype(int)
agg
out[36]: RACE           GENDER
          American Indian or Alaskan Native
          Female            60238
          Male              60305
          Asian/Pacific Islander
          Female            63226
          Male              61033
          Black or African American
          Female            48915
          Male              51082
          Hispanic/Latino
          Female            46503
          Male              54782
          Others
          Female            63785
          Male              38771
          White
          Female            66793
          Male              63940
Name: BASE_SALARY, dtype: int64
```

```
# 对索引层GENDER做unstack
In[37]: agg.unstack('GENDER')
out[37]:
```

	GENDER	Female	Male
	RACE		
American Indian or Alaskan Native		60238	60305
Asian/Pacific Islander		63226	61033
Black or African American		48915	51082
Hispanic/Latino		46503	54782
Others		63785	38771
White		66793	63940

```
# 对索引层RACE做unstack
In[38]: agg.unstack('RACE')
out[38]:
```

	RACE	American Indian or Alaskan Native	Asian/Pacific Islander	Black or African American	Hispanic/Latino	Others	White
GENDER							
Female		60238	63226	48915	46503	63785	66793
Male		60305	61033	51082	54782	38771	63940

更多

```
# 按RACE和GENDER分组，求工资的平均值、最大值和最小值
In[39]: agg2 = employee.groupby(['RACE', 'GENDER'])['BASE_SALAR
Y'].agg(['mean', 'max', 'min']).astype(int)
agg2
out[39]:
```

			mean	max	min
	RACE	GENDER			
American Indian or Alaskan Native		Female	60238	98536	26125
		Male	60305	81239	26125
Asian/Pacific Islander		Female	63226	130416	26125
		Male	61033	163228	27914
Black or African American		Female	48915	150416	24960
		Male	51082	275000	26125
Hispanic/Latino		Female	46503	126115	26125
		Male	54782	165216	26104
Others		Female	63785	63785	63785
		Male	38771	38771	38771
White		Female	66793	178331	27955
		Male	63940	210588	26125

此时unstack('GENDER')会生成多级列索引，可以用stack和unstack调整结构
agg2.unstack('GENDER')

GENDER		mean		max		min	
		Female	Male	Female	Male	Female	Male
RACE							
American Indian or Alaskan Native		60238	60305	98536	81239	26125	26125
Asian/Pacific Islander		63226	61033	130416	163228	26125	27914
Black or African American		48915	51082	150416	275000	24960	26125
Hispanic/Latino		46503	54782	126115	165216	26125	26104
Others		63785	38771	63785	38771	63785	38771
White		66793	63940	178331	210588	27955	26125

6. 用分组聚合实现透视表

```
# 读取flights数据集
In[40]: flights = pd.read_csv('data/flights.csv')
flights.head()
out[40]:
```

	MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905	
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333	
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453	
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935	
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225	

```
# 用pivot_table方法求出每条航线每个始发地的被取消的航班总数
In[41]: fp = flights.pivot_table(index='AIRLINE',
                                 columns='ORG_AIR',
                                 values='CANCELLED',
                                 aggfunc='sum',
                                 fill_value=0).round(2)
fp.head()
out[41]:
```

ORG_AIR	ATL	DEN	DFW	IAH	LAS	LAX	MSP	ORD	PHX	SFO
AIRLINE										
AA	3	4	86	3	3	11	3	35	4	2
AS	0	0	0	0	0	0	0	0	0	0
B6	0	0	0	0	0	0	0	0	0	1
DL	28	1	0	0	1	1	4	0	1	2
EV	18	6	27	36	0	0	6	53	0	0

```
# groupby聚合不能直接复现这张表。需要先按所有index和columns的列聚合
In[42]: fg = flights.groupby(['AIRLINE', 'ORG_AIR'])['CANCELLED'].sum()
fg.head()
out[42]: AIRLINE  ORG_AIR
          AA      ATL      3
                  DEN      4
                  DFW     86
                  IAH      3
                  LAS      3
Name: CANCELLED, dtype: int64
```

```
# 再使用unstack，将ORG_AIR这层索引作为列名
In[43]: fg_unstack = fg.unstack('ORG_AIR', fill_value=0)
fg_unstack.head()
out[43]:
```

AIRLINE	ORG_AIR	ATL	DEN	DFW	IAH	LAS	LAX	MSP	ORD	PHX	SFO
AA	3	4	86	3	3	11	3	35	4	2	
AS	0	0	0	0	0	0	0	0	0	0	
B6	0	0	0	0	0	0	0	0	0	1	
DL	28	1	0	0	1	1	4	0	1	2	
EV	18	6	27	36	0	0	6	53	0	0	

```
# 判断两个方式是否等价
In[44]: fg_unstack = fg.unstack('ORG_AIR', fill_value=0)
fp.equals(fg_unstack)
out[44]: True
```

更多

先实现一个稍微复杂的透视表

```
In[45]: fp2 = flights.pivot_table(index=['AIRLINE', 'MONTH'],
                                columns=['ORG_AIR', 'CANCELLED'],
                                values=['DEP_DELAY', 'DIST'],
                                aggfunc=[np.mean, np.sum],
                                fill_value=0)

fp2.head()

out[45]:
```

AIRLINE	MONTH	ORG_AIR	ATL		DEN		DFW		IAH		LAS		LAX		MSP	
			mean ...		DEP_DELAY ...		mean ...		DEP_DELAY ...		mean ...		DEP_DELAY ...		mean ...	
			CANCELLED	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
AA		1	-3.250000	0	7.062500	0	11.977591	-3.0	9.750000	0	32.375000	0	... 135921	2475	7281	0 12
		2	-3.000000	0	5.461538	0	8.756579	0.0	1.000000	0	-3.055556	0	... 113483	5454	5040	0 12
		3	-0.166667	0	7.666667	0	15.383784	0.0	10.900000	0	12.074074	0	... 131836	1744	14471	0 12
		4	0.071429	0	20.266667	0	10.501493	0.0	6.933333	0	27.241379	0	... 170285	0	4541	0 15
		5	5.777778	0	23.466667	0	16.798780	0.0	3.055556	0	2.818182	0	... 167484	0	6298	0 11

5 rows × 80 columns

用groupby和unstack复现上面的方法

```
In[46]: flights.groupby(['AIRLINE', 'MONTH', 'ORG_AIR', 'CANCELLED'])['DEP_DELAY', 'DIST'] \
    .agg(['mean', 'sum']) \
    .unstack(['ORG_AIR', 'CANCELLED'], fill_value=0) \
    .swaplevel(0, 1, axis='columns') \
    .head()

out[46]:
```

AIRLINE	MONTH	ORG_AIR	ATL		DEN		DFW		IAH		LAS		LAX	
			mean ...		DEP_DELAY ...		mean ...		DEP_DELAY ...		mean ...		DEP_DELAY ...	
			CANCELLED	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1	0 1
AA		1	-3.250000	NaN	7.062500	NaN	11.977591	-3.0	9.750000	NaN	32.375000	NaN	... 135921.0	2475.0
		2	-3.000000	NaN	5.461538	NaN	8.756579	NaN	1.000000	NaN	-3.055556	NaN	... 113483.0	5454.0
		3	-0.166667	NaN	7.666667	NaN	15.383784	NaN	10.900000	NaN	12.074074	NaN	... 131836.0	1744.0
		4	0.071429	NaN	20.266667	NaN	10.501493	NaN	6.933333	NaN	27.241379	NaN	... 170285.0	NaN
		5	5.777778	NaN	23.466667	NaN	16.798780	NaN	3.055556	NaN	2.818182	NaN	... 167484.0	NaN

5 rows × 80 columns

7. 为了更容易reshaping，重新命名索引层

```
# 读取college数据集，分组后，统计本科生的SAT数学成绩信息
In[47]: college = pd.read_csv('data/college.csv')
In[48]: cg = college.groupby(['STABBR', 'RELAFFIL'])['UGDS', 'SATMTMID'] \
         .agg(['count', 'min', 'max']).head(6)
In[49]: cg
out[49]:
```

		UGDS			SATMTMID		
		count	min	max	count	min	max
STABBR	RELAFFIL						
AK	0	7	109.0	12865.0	0	NaN	NaN
	1	3	27.0	275.0	1	503.0	503.0
AL	0	71	12.0	29851.0	13	420.0	590.0
	1	18	13.0	3033.0	8	400.0	560.0
AR	0	68	18.0	21405.0	9	427.0	565.0
	1	14	20.0	4485.0	7	495.0	600.0

```
# 行索引的两级都有名字，而列索引没有名字。用rename_axis给列索引的两级命名
In[50]: cg = cg.rename_axis(['AGG_COLS', 'AGG_FUNCS'], axis='columns')
          cg
out[50]:
```

	STABBR	RELAFFIL	AGG_COLS		UGDS		SATMTMID	
			AGG_FUNCS	count	min	max	count	min
AK			0	7	109.0	12865.0	0	NaN
			1	3	27.0	275.0	1	503.0
AL			0	71	12.0	29851.0	13	420.0
			1	18	13.0	3033.0	8	400.0
AR			0	68	18.0	21405.0	9	427.0
			1	14	20.0	4485.0	7	495.0

```
# 将AGG_FUNCS列移到行索引
In[51]:cg.stack('AGG_FUNCS').head()
out[51]:
```

STABBR	RELAFFIL	AGG_COLS		UGDS	SATMTMID	
		AGG_FUNCS		count	7.0	
AK			0	min	109.0	NaN
			1	max	12865.0	NaN
			0	count	3.0	1.0
			1	min	27.0	503.0

```
# stack默认是将列放到行索引的最内层，可以使用swaplevel改变层级
In[52]:cg.stack('AGG_FUNCS').swaplevel('AGG_FUNCS', 'STABBR', axis='index').head()
out[52]:
```

		AGG_COLS	UGDS	SATMTMID
AGG_FUNCS	RELAFFIL	STABBR		
count	0	AK	7.0	0.0
min	0	AK	109.0	NaN
max	0	AK	12865.0	NaN
count	1	AK	3.0	1.0
min	1	AK	27.0	503.0

```
# 在此前的基础上再做sort_index
In[53]:cg.stack('AGG_FUNCS') \
    .swaplevel('AGG_FUNCS', 'STABBR', axis='index') \
    .sort_index(level='RELAFFIL', axis='index') \
    .sort_index(level='AGG_COLS', axis='columns').head(6)
out[53]:
```

		AGG_COLS	SATMTMID	UGDS
AGG_FUNCS	RELAFFIL	STABBR		
		AK	0.0	7.0
count	0	AL	13.0	71.0
		AR	9.0	68.0
		AK	NaN	109.0
min	0	AL	420.0	12.0
		AR	427.0	18.0

```
# 对一些列做stack，对其它列做unstack
In[54]:cg.stack('AGG_FUNCS').unstack(['RELAFFIL', 'STABBR'])
out[54]:
```

AGG_COLS				UGDS				SATMTMID			
RELAFFIL	0	1	0	1	0	1	0	1	0	1	0
STABBR	AK	AK	AL	AL	AR	AR	AK	AK	AL	AL	AR
AGG_FUNCS											
count	7.0	3.0	71.0	18.0	68.0	14.0	0.0	1.0	13.0	8.0	9.0
min	109.0	27.0	12.0	13.0	18.0	20.0	NaN	503.0	420.0	400.0	427.0
max	12865.0	275.0	29851.0	3033.0	21405.0	4485.0	NaN	503.0	590.0	560.0	565.0
											600.0

```
# 对所有列做stack，会返回一个Series
```

```
In[55]:cg.stack(['AGG_FUNCS', 'AGG_COLS']).head(12)
out[55]:
```

STABBR	RELAFFIL	AGG_FUNCS	AGG_COLS
AK	0	count	UGDS
		min	SATMTMID
		max	UGDS
	1	count	UGDS
		min	SATMTMID
		max	UGDS
AL	0	count	UGDS
		min	SATMTMID
		max	UGDS
	1	count	SATMTMID
		min	UGDS
		max	SATMTMID

dtype: float64

更多

```
# 删除行和列索引所有层级的名称
```

```
In[56]:cg.rename_axis([None, None], axis='index').rename_axis([
None, None], axis='columns')
out[56]:
```

		UGDS			SATMTMID		
		count	min	max	count	min	max
AK	0	7	109.0	12865.0	0	NaN	NaN
	1	3	27.0	275.0	1	503.0	503.0
AL	0	71	12.0	29851.0	13	420.0	590.0
	1	18	13.0	3033.0	8	400.0	560.0
AR	0	68	18.0	21405.0	9	427.0	565.0
	1	14	20.0	4485.0	7	495.0	600.0

8. 当多个变量被存储为列名时进行清理

```
# 读取weightlifting数据集
In[57]:weightlifting = pd.read_csv('data/weightlifting_men.csv')
)
weightlifting
out[57]:
```

	Weight Category	M35 35-39	M40 40-44	M45 45-49	M50 50-54	M55 55-59	M60 60-64	M65 65-69	M70 70-74	M75 75-79	M80 80+
0	56	137	130	125	115	102	92	80	67	62	55
1	62	152	145	137	127	112	102	90	75	67	57
2	69	167	160	150	140	125	112	97	82	75	60
3	77	182	172	165	150	135	122	107	90	82	65
4	85	192	182	175	160	142	130	112	95	87	70
5	94	202	192	182	167	150	137	120	100	90	75
6	105	210	200	190	175	157	142	122	102	95	80
7	105+	217	207	197	182	165	150	127	107	100	85

```
# 用melt方法，将sex_age放入一个单独的列
In[58]:wl_melt = weightlifting.melt(id_vars='Weight Category',
                                     var_name='sex_age',
                                     value_name='Qual Total')
wl_melt.head()
out[58]:
```

	Weight Category	sex_age	Qual Total
0	56	M35 35-39	137
1	62	M35 35-39	152
2	69	M35 35-39	167
3	77	M35 35-39	182
4	85	M35 35-39	192

```
# 用split方法将sex_age列分为两列
In[59]:sex_age = wl_melt['sex_age'].str.split(expand=True)
         sex_age.head()
out[59]:      0          1
      0    M35    35-39
      1    M35    35-39
      2    M35    35-39
      3    M35    35-39
      4    M35    35-39
```

```
# 给列起名
In[60]:sex_age.columns = ['Sex', 'Age Group']
         sex_age.head()
out[60]:
```

	Sex	Age Group
0	M35	35-39
1	M35	35-39
2	M35	35-39
3	M35	35-39
4	M35	35-39

```
# 只取出字符串中的M
In[61]:sex_age['Sex'] = sex_age['Sex'].str[0]
         sex_age.head()
out[61]:
```

Sex	Age Group
0	M 35-39
1	M 35-39
2	M 35-39
3	M 35-39
4	M 35-39

```
# 用concat方法，将sex_age, 与wl_cat_total连接起来
In[62]:wl_cat_total = wl_melt[['Weight Category', 'Qual Total']]
wl_tidy = pd.concat([sex_age, wl_cat_total], axis='columns')
wl_tidy.head()
out[62]:
```

Sex	Age Group	Weight Category	Qual Total
0	M 35-39	56	137
1	M 35-39	62	152
2	M 35-39	69	167
3	M 35-39	77	182
4	M 35-39	85	192

```
# 上面的结果也可以如下实现
In[63]:cols = ['Weight Category', 'Qual Total']
sex_age[cols] = wl_melt[cols]
```

更多

```
# 也可以通过assign的方法，动态加载新的列
In[64]: age_group = wl_melt.sex_age.str.extract('(\d{2})[-+](?:\d{2})?', expand=False)
          sex = wl_melt.sex_age.str[0]
          new_cols = {'Sex':sex,
                      'Age Group': age_group}
In[65]: wl_tidy2 = wl_melt.assign(**new_cols).drop('sex_age', axis='columns')
          wl_tidy2.head()
out[65]:
```

	Weight Category	Qual Total	Age Group	Sex
0	56	137	35-39	M
1	62	152	35-39	M
2	69	167	35-39	M
3	77	182	35-39	M
4	85	192	35-39	M

```
# 判断两种方法是否等效
In[66]: wl_tidy2.sort_index(axis=1).equals(wl_tidy.sort_index(axis=1))
out[66]: True
```

9. 当多个变量被存储为列的值时进行清理

```
# 读取restaurant_inspections数据集，将Date列的数据类型变为datetime64
In[67]: inspections = pd.read_csv('data/restaurant_inspections.csv',
           parse_dates=['Date'])
          inspections.head(10)
out[67]:
```

	Name	Date	Info	Value
0	E & E Grill House	2017-08-08	Borough	MANHATTAN
1	E & E Grill House	2017-08-08	Cuisine	American
2	E & E Grill House	2017-08-08	Description	Non-food contact surface improperly constructe...
3	E & E Grill House	2017-08-08	Grade	A
4	E & E Grill House	2017-08-08	Score	9.0
5	PIZZA WAGON	2017-04-12	Borough	BROOKLYN
6	PIZZA WAGON	2017-04-12	Cuisine	Pizza
7	PIZZA WAGON	2017-04-12	Description	Food contact surface not properly washed, rins...
8	PIZZA WAGON	2017-04-12	Grade	A
9	PIZZA WAGON	2017-04-12	Score	10.0

```
# 用info列的所有值造一个新列。但是，Pandas不支持这种功能
In[68]: inspections.pivot(index=['Name', 'Date'], columns='Info',
                           values='Value')
-----
ValueError                                     Traceback (most recent
call last)
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/cate-
gorical.py in __init__(self, values, categories, ordered, fastpa-
th)
    297         try:
--> 298             codes, categories = factorize(values, so-
rt=True)
    299         except TypeError:
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/algo-
rithms.py in factorize(values, sort, order, na_sentinel, size_hi-
nt)
    559     check_nulls = not is_integer_dtype(original)
--> 560     labels = table.get_labels(values, uniques, 0, na_se-
ntinel, check_nulls)
    561
pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtabl-
e.PyObjectHashTable.get_labels (pandas/_libs/hashtable.c:21922)()
ValueError: Buffer has wrong number of dimensions (expected 1, g-
ot 2)

During handling of the above exception, another exception occur-
ed:
```

```

NotImplementedError                                Traceback (most recent
call last)
<ipython-input-68-754f69d68d6c> in <module>()
----> 1 inspections.pivot(index=['Name', 'Date'], columns='Info'
, values='Value')

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/corefram
e.py in pivot(self, index, columns, values)
    3851     """
    3852         from pandas.core.reshape.reshape import pivot
-> 3853         return pivot(self, index=index, columns=columns,
values=values)
    3854
    3855     def stack(self, level=-1, dropna=True):

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/resh
ape/reshape.py in pivot(self, index, columns, values)
    375         index = self[index]
    376         indexed = Series(self[values].values,
--> 377                         index=MultiIndex.from_arrays([i
ndex, self[columns]]))
    378         return indexed.unstack(columns)
    379

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/inde
xes/multi.py in from_arrays(cls, arrays, sortorder, names)
    1098         from pandas.core.categorical import _factorize_f
rom_iterables
    1099
-> 1100         labels, levels = _factorize_from_iterables(array
s)
    1101         if names is None:
    1102             names = [getattr(arr, "name", None) for arr
in arrays]

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/cate
gorical.py in _factorize_from_iterables(iterables)
    2191         # For consistency, it should return a list of 2
lists.
    2192         return [], []
-> 2193     return map(list, lzip(*[_factorize_from_iterable(it)
for it in iterables]))

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/cate
gorical.py in <listcomp>(.0)
    2191         # For consistency, it should return a list of 2
lists.
    2192         return [], []
-> 2193     return map(list, lzip(*[_factorize_from_iterable(it)
for it in iterables]))


/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/cate
gorical.py in _factorize_from_iterable(values)

```

```

2163         codes = values.codes
2164     else:
-> 2165         cat = Categorical(values, ordered=True)
2166         categories = cat.categories
2167         codes = cat.codes

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/cate-
gorical.py in __init__(self, values, categories, ordered, fastpa-
th)
    308
    309             # FIXME
--> 310             raise NotImplementedError("> 1 ndim Cate-
gorical are not "
    311                               "supported at "
this time")
    312

NotImplementedError: > 1 ndim Categorical are not supported at t-
his time

```

```

# 将'Name', 'Date', 'Info'作为所索引
In[69]: inspections.set_index(['Name', 'Date', 'Info']).head(10)
out[69]:

```

			Value
Name	Date	Info	
		Borough	MANHATTAN
		Cuisine	American
E & E Grill House	2017-08-08	Description	Non-food contact surface improperly construct...
		Grade	A
		Score	9.0
		Borough	BROOKLYN
		Cuisine	Pizza
PIZZA WAGON	2017-04-12	Description	Food contact surface not properly washed, rins...
		Grade	A
		Score	10.0

```

# 用pivot，将info列中的值变为新的列
In[70]: inspections.set_index(['Name', 'Date', 'Info']).unstack(
'Info').head()
out[70]:

```

							Value
	Name	Date	Borough	Cuisine	Description	Grade	Score
3 STAR JUICE CENTER	2017-05-10	BROOKLYN	Juice, Smoothies, Fruit Salads	Facility not vermin proof. Harborage or condit...	A	12.0	
A & L PIZZA RESTAURANT	2017-08-22	BROOKLYN	Pizza	Facility not vermin proof. Harborage or condit...	A	9.0	
AKSARAY TURKISH CAFE AND RESTAURANT	2017-07-25	BROOKLYN	Turkish	Plumbing not properly installed or maintained;...	A	13.0	
ANTOJITOS DELI FOOD	2017-06-01	BROOKLYN	Latin (Cuban, Dominican, Puerto Rican, South &...)	Live roaches present in facility's food and/or...	A	10.0	
BANGIA	2017-06-16	MANHATTAN	Korean	Covered garbage receptacle not provided or ina...	A	9.0	

用reset_index方法，使行索引层级与列索引相同

```
In[71]: insp_tidy = inspections.set_index(['Name', 'Date', 'Info'])
        .unstack('Info')
        .reset_index(col_level=-1)
)
insp_tidy.head()
out[71]:
```

							Value
Info	Name	Date	Borough	Cuisine	Description	Grade	Score
0	3 STAR JUICE CENTER	2017-05-10	BROOKLYN	Juice, Smoothies, Fruit Salads	Facility not vermin proof. Harborage or condit...	A	12.0
1	A & L PIZZA RESTAURANT	2017-08-22	BROOKLYN	Pizza	Facility not vermin proof. Harborage or condit...	A	9.0
2	AKSARAY TURKISH CAFE AND RESTAURANT	2017-07-25	BROOKLYN	Turkish	Plumbing not properly installed or maintained;...	A	13.0
3	ANTOJITOS DELI FOOD	2017-06-01	BROOKLYN	Latin (Cuban, Dominican, Puerto Rican, South &...)	Live roaches present in facility's food and/or...	A	10.0
4	BANGIA	2017-06-16	MANHATTAN	Korean	Covered garbage receptacle not provided or ina...	A	9.0

除掉列索引的最外层，重命名行索引的层为None

```
In[72]: insp_tidy.columns = insp_tidy.columns.droplevel(0).rename(None)
insp_tidy.head()
out[72]:
```

	Name	Date	Borough	Cuisine	Description	Grade	Score
0	3 STAR JUICE CENTER	2017-05-10	BROOKLYN	Juice, Smoothies, Fruit Salads	Facility not vermin proof. Harborage or condit...	A	12.0
1	A & L PIZZA RESTAURANT	2017-08-22	BROOKLYN	Pizza	Facility not vermin proof. Harborage or condit...	A	9.0
2	AKSARAY TURKISH CAFE AND RESTAURANT	2017-07-25	BROOKLYN	Turkish	Plumbing not properly installed or maintained;...	A	13.0
3	ANTOJITOS DELI FOOD	2017-06-01	BROOKLYN	Latin (Cuban, Dominican, Puerto Rican, South &...)	Live roaches present in facility's food and/or...	A	10.0
4	BANGIA	2017-06-16	MANHATTAN	Korean	Covered garbage receptacle not provided or ina...	A	9.0

使用squeeze方法，可以避免前面的多级索引

```
In[73]: inspections.set_index(['Name', 'Date', 'Info']) \
    .squeeze() \
    .unstack('Info') \
    .reset_index() \
    .rename_axis(None, axis='columns')
```

out[73]:

	Name	Date	Borough	Cuisine	Description	Grade	Score
0	3 STAR JUICE CENTER	2017-05-10	BROOKLYN	Juice, Smoothies, Fruit Salads	Facility not vermin proof. Harborage or condit...	A	12.0
1	A & L PIZZA RESTAURANT	2017-08-22	BROOKLYN	Pizza	Facility not vermin proof. Harborage or condit...	A	9.0
2	AKSARAY TURKISH CAFE AND RESTAURANT	2017-07-25	BROOKLYN	Turkish	Plumbing not properly installed or maintained;...	A	13.0
3	ANTOJITOS DELI FOOD	2017-06-01	BROOKLYN	Latin (Cuban, Dominican, Puerto Rican, South &...)	Live roaches present in facility's food and/or...	A	10.0
4	BANGIA	2017-06-16	MANHATTAN	Korean	Covered garbage receptacle not provided or ina...	A	9.0
5	BANGKOK CUISINE	2017-07-19	MANHATTAN	Thai	Non-food contact surface improperly construct...	A	13.0
6	BASIL	2017-05-03	BROOKLYN	Jewish/Kosher	Cold food item held above 41°F (smoked fish ...)	A	13.0

更多

pivot_table需要传入聚合函数，才能产生一个单一值

```
In[74]: inspections.pivot_table(index=['Name', 'Date'],
                                 columns='Info',
                                 values='Value',
                                 aggfunc='first') \
    .reset_index() \
    .rename_axis(None, axis='columns')
```

out[74]:

	Name	Date	Borough	Cuisine	Description	Grade	Score
0	3 STAR JUICE CENTER	2017-05-10	BROOKLYN	Juice, Smoothies, Fruit Salads	Facility not vermin proof. Harborage or condit...	A	12.0
1	A & L PIZZA RESTAURANT	2017-08-22	BROOKLYN	Pizza	Facility not vermin proof. Harborage or condit...	A	9.0
2	AKSARAY TURKISH CAFE AND RESTAURANT	2017-07-25	BROOKLYN	Turkish	Plumbing not properly installed or maintained;...	A	13.0
3	ANTOJITOS DELI FOOD	2017-06-01	BROOKLYN	Latin (Cuban, Dominican, Puerto Rican, South &...)	Live roaches present in facility's food and/or...	A	10.0
4	BANGIA	2017-06-16	MANHATTAN	Korean	Covered garbage receptacle not provided or ina...	A	9.0
5	BANGKOK CUISINE	2017-07-19	MANHATTAN	Thai	Non-food contact surface improperly constructe...	A	13.0
6	BASIL	2017-05-03	BROOKLYN	Jewish/Kosher	Cold food item held above 41° F (smoked fish ...	A	13.0

10. 当两个或多个值存储于一个单元格时进行清理

```
# 读取texas_cities数据集
In[75]: cities = pd.read_csv('data/texas_cities.csv')
         cities
out[75]:
```

	City	Geolocation
0	Houston	29.7604° N, 95.3698° W
1	Dallas	32.7767° N, 96.7970° W
2	Austin	30.2672° N, 97.7431° W

```
# 将Geolocation分解为四个单独的列
In[76]: geolocations = cities.Geolocation.str.split(pat='.', expand=True)
         geolocations.columns = ['latitude', 'latitude direction', 'longitude', 'longitude direction']
         geolocations
out[76]:
```

	latitude	latitude direction	longitude	longitude direction
0	29.7604	N	95.3698	W
1	32.7767	N	96.7970	W
2	30.2672	N	97.7431	W

```
# 转变数据类型
In[77]: geolocations = geolocations.astype({'latitude':'float',
'longitude':'float'})
geolocations.dtypes
out[77]: latitude           float64
latitude direction        object
longitude              float64
longitude direction        object
dtype: object
```

```
# 将新列与原先的city列连起来
In[78]: cities_tidy = pd.concat([cities['City'], geolocations],
axis='columns')
cities_tidy
out[78]:
```

	City	latitude	latitude direction	longitude	longitude direction
0	Houston	29.7604	N	95.3698	W
1	Dallas	32.7767	N	96.7970	W
2	Austin	30.2672	N	97.7431	W

```
# 忽略，作者这里是写重复了
In[79]: pd.concat([cities['City'], geolocations], axis='columns')
out[79]:
```

	City	latitude	latitude direction	longitude	longitude direction
0	Houston	29.7604		N	95.3698
1	Dallas	32.7767		N	96.7970
2	Austin	30.2672		N	97.7431

原理

```
# 函数to_numeric可以将每列自动变为整数或浮点数
In[80]: temp = geolocations.apply(pd.to_numeric, errors='ignore')
          temp
out[80]:
```

	latitude	latitude direction	longitude	longitude direction
0	29.7604		N	95.3698
1	32.7767		N	96.7970
2	30.2672		N	97.7431

```
# 再查看数据类型
In[81]: temp.dtypes
out[81]: latitude           float64
          latitude direction    object
          longitude            float64
          longitude direction    object
          dtype: object
```

更多

```
# | 符，可以对多个标记进行分割
In[82]: cities.GeoLocation.str.split(pat='° |, ', expand=True)
out[82]:
```

	0	1	2	3
0	29.7604	N	95.3698	W
1	32.7767	N	96.7970	W
2	30.2672	N	97.7431	W

更复杂的提取方式

```
In[83]: cities.GeoLocation.str.extract('([0-9.]+). ([N|S]), ([0-9.]+). ([E|W])', expand=True)
out[83]:
```

	0	1	2	3
0	29.7604	N	95.3698	W
1	32.7767	N	96.7970	W
2	30.2672	N	97.7431	W

11. 当多个变量被存储为列名和列值时进行清理

```
# 读取sensors数据集
In[84]: sensors = pd.read_csv('data/sensors.csv')
          sensors
out[84]:
```

	Group	Property	2012	2013	2014	2015	2016
0	A	Pressure	928	873	814	973	870
1	A	Temperature	1026	1038	1009	1036	1042
2	A	Flow	819	806	861	882	856
3	B	Pressure	817	877	914	806	942
4	B	Temperature	1008	1041	1009	1002	1013
5	B	Flow	887	899	837	824	873

```
# 用melt清理数据
In[85]: sensors.melt(id_vars=['Group', 'Property'], var_name='Year')
out[85]:
```

	Group	Property	Year	value
0	A	Pressure	2012	928
1	A	Temperature	2012	1026
2	A	Flow	2012	819
3	B	Pressure	2012	817
4	B	Temperature	2012	1008
5	B	Flow	2012	887

```
# 用pivot_table，将Property列转化为新的列名
In[86]: sensors.melt(id_vars=['Group', 'Property'], var_name='Year') \
          .pivot_table(index=['Group', 'Year'], columns='Property', values='value') \
          .reset_index() \
          .rename_axis(None, axis='columns')
out[86]:
```

	Group	Year	Flow	Pressure	Temperature
0	A	2012	819	928	1026
1	A	2013	806	873	1038
2	A	2014	861	814	1009
3	A	2015	882	973	1036
4	A	2016	856	870	1042
5	B	2012	887	817	1008
6	B	2013	899	877	1041
7	B	2014	837	914	1009
8	B	2015	824	806	1002
9	B	2016	873	942	1013

更多

```
# 用stack和unstack实现上述方法
In[87]: sensors.set_index(['Group', 'Property']) \
    .stack() \
    .unstack('Property') \
    .rename_axis(['Group', 'Year'], axis='index') \
    .rename_axis(None, axis='columns') \
    .reset_index()
out[87]:
```

	Group	Year	Flow	Pressure	Temperature
0	A	2012	819	928	1026
1	A	2013	806	873	1038
2	A	2014	861	814	1009
3	A	2015	882	973	1036
4	A	2016	856	870	1042
5	B	2012	887	817	1008
6	B	2013	899	877	1041
7	B	2014	837	914	1009
8	B	2015	824	806	1002
9	B	2016	873	942	1013

12. 当多个观察单位被存储于同一张表时进行清理

```
# 读取movie_altered数据集
In[88]: movie = pd.read_csv('data/movie_altered.csv')
          movie.head()
out[88]:
```

	title	rating	year	duration	director_1	director_fb_likes_1	actor_1	actor_2	actor_3	actor_fb_likes_1	actor_fb_
0	Avatar	PG-13	2009.0	178.0	James Cameron	0.0	CCH Pounder	Joel David Moore	Wes Studi		1000.0
1	Pirates of the Caribbean: At World's End	PG-13	2007.0	169.0	Gore Verbinski	563.0	Johnny Depp	Orlando Bloom	Jack Davenport		40000.0
2	Spectre	PG-13	2015.0	148.0	Sam Mendes	0.0	Christoph Waltz	Rory Kinnear	Stephanie Sigman		11000.0
3	The Dark Knight Rises	PG-13	2012.0	164.0	Christopher Nolan	22000.0	Tom Hardy	Christian Bale	Joseph Gordon-Levitt		27000.0
4	Star Wars: Episode VII - The Force Awakens	Nan	Nan	Nan	Doug Walker	131.0	Doug Walker	Rob Walker	Nan		131.0

```
# 插入新的列，用来标识每一部电影
```

```
In[89]: movie.insert(0, 'id', np.arange(len(movie)))
movie.head()
```

```
out[89]:
```

	id	title	rating	year	duration	director_1	director_fb_likes_1	actor_1	actor_2	actor_3	actor_fb_likes_1	actor_fb_likes_2
0	0	Avatar	PG-13	2009.0	178.0	James Cameron	0.0	CCH Pounder	Joel David Moore	Wes Studi	1000.0	
1	1	Pirates of the Caribbean: At World's End	PG-13	2007.0	169.0	Gore Verbinski	563.0	Johnny Depp	Orlando Bloom	Jack Davenport	40000.0	
2	2	Spectre	PG-13	2015.0	148.0	Sam Mendes	0.0	Christoph Waltz	Rory Kinnear	Stephanie Sigman	11000.0	
3	3	The Dark Knight Rises	PG-13	2012.0	164.0	Christopher Nolan	22000.0	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000.0	
4	4	Star Wars: Episode VII - The Force Awakens	Nan	Nan	Nan	Doug Walker	131.0	Doug Walker	Rob Walker	Nan	131.0	

```
# 用wide_to_long，将所有演员放到一列，将所有Facebook likes放到一列
```

```
In[90]: stubnames = ['director', 'director_fb_likes', 'actor',
'actor_fb_likes']
```

```
movie_long = pd.wide_to_long(movie,
                             stubnames=stubnames,
                             i='id',
                             j='num',
                             sep='_').reset_index()
movie_long['num'] = movie_long['num'].astype(int)
movie_long.head(9)
```

```
out[90]:
```

	id	num	year	rating	title	duration	director	director_fb_likes	actor	actor_fb_likes
0	0	1	2009.0	PG-13	Avatar	178.0	James Cameron	0.0	CCH Pounder	1000.0
1	0	2	2009.0	PG-13	Avatar	178.0	Nan	Nan	Joel David Moore	936.0
2	0	3	2009.0	PG-13	Avatar	178.0	Nan	Nan	Wes Studi	855.0
3	1	1	2007.0	PG-13	Pirates of the Caribbean: At World's End	169.0	Gore Verbinski	563.0	Johnny Depp	40000.0
4	1	2	2007.0	PG-13	Pirates of the Caribbean: At World's End	169.0	Nan	Nan	Orlando Bloom	5000.0
5	1	3	2007.0	PG-13	Pirates of the Caribbean: At World's End	169.0	Nan	Nan	Jack Davenport	1000.0
6	2	1	2015.0	PG-13	Spectre	148.0	Sam Mendes	0.0	Christoph Waltz	11000.0
7	2	2	2015.0	PG-13	Spectre	148.0	Nan	Nan	Rory Kinnear	393.0
8	2	3	2015.0	PG-13	Spectre	148.0	Nan	Nan	Stephanie Sigman	161.0

```
# 将这个数据分解成多个小表
In[91]: movie_table = movie_long[['id', 'title', 'year', 'duration', 'rating']]
         director_table = movie_long[['id', 'director', 'num', 'director_fb_likes']]
         actor_table = movie_long[['id', 'actor', 'num', 'actor_fb_likes']]
In[92]: movie_table.head(9)
out[90]:
```

	id		title	year	duration	rating
0	0		Avatar	2009.0	178.0	PG-13
1	0		Avatar	2009.0	178.0	PG-13
2	0		Avatar	2009.0	178.0	PG-13
3	1	Pirates of the Caribbean: At World's End		2007.0	169.0	PG-13
4	1	Pirates of the Caribbean: At World's End		2007.0	169.0	PG-13
5	1	Pirates of the Caribbean: At World's End		2007.0	169.0	PG-13
6	2		Spectre	2015.0	148.0	PG-13
7	2		Spectre	2015.0	148.0	PG-13
8	2		Spectre	2015.0	148.0	PG-13

```
In[93]: director_table.head(9)
out[93]:
```

	id	director	num	director_fb_likes
0	0	James Cameron	1	0.0
1	0	NaN	2	NaN
2	0	NaN	3	NaN
3	1	Gore Verbinski	1	563.0
4	1	NaN	2	NaN
5	1	NaN	3	NaN
6	2	Sam Mendes	1	0.0
7	2	NaN	2	NaN
8	2	NaN	3	NaN

```
In[94]: actor_table.head(9)
out[94]:
```

	id	actor	num	actor_fb_likes
0	0	CCH Pounder	1	1000.0
1	0	Joel David Moore	2	936.0
2	0	Wes Studi	3	855.0
3	1	Johnny Depp	1	40000.0
4	1	Orlando Bloom	2	5000.0
5	1	Jack Davenport	3	1000.0
6	2	Christoph Waltz	1	11000.0
7	2	Rory Kinnear	2	393.0
8	2	Stephanie Sigman	3	161.0

```
# 做一些去重和去除缺失值的工作
In[95]: movie_table = movie_table.drop_duplicates().reset_index(
    drop=True)
        director_table = director_table.dropna().reset_index(
            drop=True)
        actor_table = actor_table.dropna().reset_index(drop=True)
)
In[96]: movie_table.head()
out[96]:
```

	id		title	year	duration	rating
0	0		Avatar	2009.0	178.0	PG-13
1	1	Pirates of the Caribbean: At World's End		2007.0	169.0	PG-13
2	2		Spectre	2015.0	148.0	PG-13
3	3		The Dark Knight Rises	2012.0	164.0	PG-13
4	4	Star Wars: Episode VII - The Force Awakens		NaN	NaN	NaN

```
In[97]: director_table.head()
out[97]:
```

	id	director	num	director_fb_likes
0	0	James Cameron	1	0.0
1	1	Gore Verbinski	1	563.0
2	2	Sam Mendes	1	0.0
3	3	Christopher Nolan	1	22000.0
4	4	Doug Walker	1	131.0

```
# 比较内存的使用量
In[98]: movie.memory_usage(deep=True).sum()
out[98]: 2318234

In[99]: movie_table.memory_usage(deep=True).sum() + \
          director_table.memory_usage(deep=True).sum() + \
          actor_table.memory_usage(deep=True).sum()
out[99]: 2624898
```

```
# 创建演员和导演的id列
In[100]: director_cat = pd.Categorical(director_table['director'])
          director_table.insert(1, 'director_id', director_cat.codes)

          actor_cat = pd.Categorical(actor_table['actor'])
          actor_table.insert(1, 'actor_id', actor_cat.codes)

          director_table.head()
out[100]:
```

◀ ▶

	id	director_id	director	num	director_fb_likes
0	0	922	James Cameron	1	0.0
1	1	794	Gore Verbinski	1	563.0
2	2	2020	Sam Mendes	1	0.0
3	3	373	Christopher Nolan	1	22000.0
4	4	600	Doug Walker	1	131.0

```
In[101]: actor_table.head()
out[101]:
```

	<u>id</u>	<u>actor_id</u>	actor	num	<u>actor_fb_likes</u>
0	0	824	CCH Pounder	1	1000.0
1	0	2867	Joel David Moore	2	936.0
2	0	6099	Wes Studi	3	855.0
3	1	2971	Johnny Depp	1	40000.0
4	1	4536	Orlando Bloom	2	5000.0

可以用这两张表生成要用的中间表。先来做director表

```
In[102]: director_associative = director_table[['id', 'director_id', 'num']]
         dcols = ['director_id', 'director', 'director_fb_likes']
         director_unique = director_table[dcols].drop_duplicate()
         .reset_index(drop=True)
         director_associative.head()
out[102]:
```

	<u>id</u>	<u>director_id</u>	num
0	0	922	1
1	1	794	1
2	2	2020	1
3	3	373	1
4	4	600	1

```
In[103]: director_unique.head()
out[103]:
```

	director_id	director	director_fb_likes
0	922	James Cameron	0.0
1	794	Gore Verbinski	563.0
2	2020	Sam Mendes	0.0
3	373	Christopher Nolan	22000.0
4	600	Doug Walker	131.0

```
# 再来做actor表
In[104]: actor_associative = actor_table[['id', 'actor_id', 'num']]
          acols = ['actor_id', 'actor', 'actor_fb_likes']
          actor_unique = actor_table[acols].drop_duplicates().reset_index(drop=True)
          actor_associative.head()
out[104]:
```

	id	actor_id	num
0	0	824	1
1	0	2867	2
2	0	6099	3
3	1	2971	1
4	1	4536	2

```
In[105]: actor_unique.head()
out[105]:
```

	actor_id	actor	actor_fb_likes
0	824	CCH Pounder	1000.0
1	2867	Joel David Moore	936.0
2	6099	Wes Studi	855.0
3	2971	Johnny Depp	40000.0
4	4536	Orlando Bloom	5000.0

```
# 查看新的表所使用的内存量
```

```
In[106]: movie_table.memory_usage(deep=True).sum() + \
           director_associative.memory_usage(deep=True).sum() + \
           director_unique.memory_usage(deep=True).sum() + \
           actor_associative.memory_usage(deep=True).sum() + \
           actor_unique.memory_usage(deep=True).sum()
out[106]: 1833402
```

```
In[107]: movie_table.head()
```

```
out[107]:
```

	id	title	year	duration	rating
0	0	Avatar	2009.0	178.0	PG-13
1	1	Pirates of the Caribbean: At World's End	2007.0	169.0	PG-13
2	2	Spectre	2015.0	148.0	PG-13
3	3	The Dark Knight Rises	2012.0	164.0	PG-13
4	4	Star Wars: Episode VII - The Force Awakens	NaN	NaN	NaN

```
# 可以通过将左右表组合起来形成movie表。首先将附表与actor/director表结合，  
然后将num列pivot，再加上列的前缀  
In[108]: actors = actor_associative.merge(actor_unique, on='actor_id') \  
          .drop('actor_id', 1) \  
          .pivot_table(index='id', col  
umns='num', aggfunc='first')  
  
          actors.columns = actors.columns.get_level_values(0) +  
'_' + \  
          actors.columns.get_level_values(1).as  
type(str)  
  
          directors = director_associative.merge(director_unique  
, on='director_id') \  
          .drop('director_id', 1) \  
          .pivot_table(index='id'  
, columns='num', aggfunc='first')  
  
          directors.columns = directors.columns.get_level_values(0) + '_' + \  
          directors.columns.get_level_values(1).astype(str)  
In[109]: actors.head()  
out[109]:
```

	actor_1	actor_2	actor_3	actor_fb_likes_1	actor_fb_likes_2	actor_fb_likes_3
id						
0	CCH Pounder	Joel David Moore	Wes Studi	1000.0	936.0	855.0
1	Johnny Depp	Orlando Bloom	Jack Davenport	40000.0	5000.0	1000.0
2	Christoph Waltz	Rory Kinnear	Stephanie Sigman	11000.0	393.0	161.0
3	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000.0	23000.0	23000.0
4	Doug Walker	Rob Walker	None	131.0	12.0	NaN

```
In[110]: directors.head()  
out[110]:
```

director_1 director_fb_likes_1

id

0	James Cameron	0.0
1	Gore Verbinski	563.0
2	Sam Mendes	0.0
3	Christopher Nolan	22000.0
4	Doug Walker	131.0

```
In[111]: movie2 = movie_table.merge(directors.reset_index(), on='id', how='left') \
          .merge(actors.reset_index(), on='id', how='left')
```

```
In[112]: movie2.head()
out[112]:
```

	id	title	year	duration	rating	director_1	director_fb_likes_1	actor_1	actor_2	actor_3	actor_fb_likes_1	actor
0	0	Avatar	2009.0	178.0	PG-13	James Cameron	0.0	CCH Pounder	Joel David Moore	Wes Studi	1000.0	
1	1	Pirates of the Caribbean: At World's End	2007.0	169.0	PG-13	Gore Verbinski	563.0	Johnny Depp	Orlando Bloom	Jack Davenport	40000.0	
2	2	Spectre	2015.0	148.0	PG-13	Sam Mendes	0.0	Christoph Waltz	Rory Kinnear	Stephanie Sigman	11000.0	
3	3	The Dark Knight Rises	2012.0	164.0	PG-13	Christopher Nolan	22000.0	Tom Hardy	Christian Bale	Joseph Gordon-Levitt	27000.0	
4	4	Star Wars: Episode VII - The Force Awakens	Nan	Nan	Nan	Doug Walker	131.0	Doug Walker	Rob Walker	None	131.0	

```
In[113]: movie.equals(movie2[movie.columns])
out[113]: True
```

第09章 合并Pandas对象

```
In[1]: import pandas as pd  
import numpy as np  
  
import matplotlib.pyplot as plt  
%matplotlib inline
```

1. DataFrame添加新的行

```
# 读取names数据集  
In[2]: names = pd.read_csv('data/names.csv')  
names  
Out[2]:
```

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2

```
# 用loc直接赋值新的行  
In[3]: new_data_list = ['Aria', 1]  
names.loc[4] = new_data_list  
names  
Out[3]:
```

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1

```
# 用loc的标签直接赋值新的行
```

```
In[4]: names.loc['five'] = ['Zach', 3]
```

```
names
```

```
Out[4]:
```

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1
five	Zach	3

```
# 也可以用字典赋值新行
```

```
In[5]: names.loc[len(names)] = {'Name': 'Zayd', 'Age': 2}
```

```
names
```

```
Out[5]:
```

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1
five	Zach	3
6	Zayd	2

In[6]: names

Out[6]:

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1
five	Zach	3
6	Zayd	2

字典可以打乱列名的顺序

In[7]: names.loc[len(names)] = pd.Series({'Age':32, 'Name':'Deahn'})

names

Out[7]:

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1
five	Zach	3
6	Zayd	2
7	Dean	32

```
# 直接append一个字典
In[8]: names = pd.read_csv('data/names.csv')
names.append({'Name':'Aria', 'Age':1})
-----
-----
TypeError                                     Traceback (most recent
call last)
<ipython-input-8-562aecc73587> in <module>()
      1 # Use append with fresh copy of names
      2 names = pd.read_csv('data/names.csv')
-> 3 names.append({'Name':'Aria', 'Age':1})

/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/corefram
e.py in append(self, other, ignore_index, verify_integrity)
    4515             other = Series(other)
    4516             if other.name is None and not ignore_index:
-> 4517                 raise TypeError('Can only append a Serie
s if ignore_index=True'
    4518                                         ' or if the Series has a
name')
    4519

TypeError: Can only append a Series if ignore_index=True or if t
he Series has a name
```

```
# 按照错误提示，加上ignore_index=True
In[9]: names.append({'Name':'Aria', 'Age':1}, ignore_index=True)
Out[9]:
```

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1

设定行索引

In[10]: names.index = ['Canada', 'Canada', 'USA', 'USA']
names

Out[10]:

	Name	Age
Canada	Cornelia	70
Canada	Abbas	69
USA	Penelope	4
USA	Niko	2

添加一行

In[11]: names.append({'Name': 'Aria', 'Age': 1}, ignore_index=True)
Out[11]:

	Name	Age
0	Cornelia	70
1	Abbas	69
2	Penelope	4
3	Niko	2
4	Aria	1

```
# 创建一个Series对象
In[12]: s = pd.Series({'Name': 'Zach', 'Age': 3}, name=len(name))
          s
Out[12]: Age      3
          Name    Zach
          Name: 4, dtype: object
```

```
# append方法可以将DataFrame和Series相连
In[13]: names.append(s)
Out[13]:
```

	Name	Age
Canada	Cornelia	70
Canada	Abbas	69
USA	Penelope	4
USA	Niko	2
4	Zach	3

```
# append方法可以同时连接多行，只要将对象放到列表中
In[14]: s1 = pd.Series({'Name': 'Zach', 'Age': 3}, name=len(names))
          s2 = pd.Series({'Name': 'Zayd', 'Age': 2}, name='USA')
          names.append([s1, s2])
Out[14]:
```

	Name	Age
Canada	Cornelia	70
Canada	Abbas	69
USA	Penelope	4
USA	Niko	2
4	Zach	3
USA	Zayd	2

```
# 读取baseball16数据集
In[15]: bball_16 = pd.read_csv('data/baseball16.csv')
         bball_16.head()
Out[15]:
```

	playerID	yearID	stint	teamID	lgID	G	AB	R	H	2B	...	RBI	SB	CS	BB	SO	IBB	HBP	SH	SF	GIDP
0	altuvjo01	2016	1	HOU	AL	161	640	108	216	42	...	96.0	30.0	10.0	60	70.0	11.0	7.0	3.0	7.0	15.0
1	bregmal01	2016	1	HOU	AL	49	201	31	53	13	...	34.0	2.0	0.0	15	52.0	0.0	0.0	0.0	1.0	1.0
2	castrja01	2016	1	HOU	AL	113	329	41	69	16	...	32.0	2.0	1.0	45	123.0	0.0	1.0	1.0	0.0	9.0
3	correca01	2016	1	HOU	AL	153	577	76	158	36	...	96.0	13.0	3.0	75	139.0	5.0	5.0	0.0	3.0	12.0
4	gattiev01	2016	1	HOU	AL	128	447	58	112	19	...	72.0	2.0	1.0	43	127.0	6.0	4.0	0.0	5.0	12.0

5 rows × 22 columns

```
# 选取一行，并将其转换为字典
In[16]: data_dict = bball_16.iloc[0].to_dict()
          print(data_dict)
{'playerID': 'altuvjo01', 'yearID': 2016, 'stint': 1, 'teamID': 'HOU', 'lgID': 'AL', 'G': 161, 'AB': 640, 'R': 108, 'H': 216, '2B': 42, '3B': 5, 'HR': 24, 'RBI': 96.0, 'SB': 30.0, 'CS': 10.0, 'BB': 60, 'SO': 70.0, 'IBB': 11.0, 'HBP': 7.0, 'SH': 3.0, 'SF': 7.0, 'GIDP': 15.0}
```

```
# 对这个字典做格式处理，如果是字符串则为空，否则为缺失值
In[17]: new_data_dict = {k: '' if isinstance(v, str) else np.nan for k, v in data_dict.items()}
          print(new_data_dict)
{'playerID': '', 'yearID': nan, 'stint': nan, 'teamID': '', 'lgID': '', 'G': nan, 'AB': nan, 'R': nan, 'H': nan, '2B': nan, '3B': nan, 'HR': nan, 'RBI': nan, 'SB': nan, 'CS': nan, 'BB': nan, 'SO': nan, 'IBB': nan, 'HBP': nan, 'SH': nan, 'SF': nan, 'GIDP': nan}
```

更多

将一行数据添加到DataFrame是非常消耗资源的，不能通过循环的方法来做。下面是创建一千行的新数据，用作Series的列表：

```
In[18]: random_data = []
for i in range(1000):
    d = dict()
    for k, v in data_dict.items():
        if isinstance(v, str):
            d[k] = np.random.choice(list('abcde'))
        else:
            d[k] = np.random.randint(10)
    random_data.append(pd.Series(d, name=i + len(bball_16)))
random_data[0].head()

Out[18]: 2B    2
          3B    6
          AB    8
          BB    2
          CS    0
          Name: 16, dtype: object
```

给上面的append操作计时，1000行的数据用了5秒钟

```
In[19]: %%timeit
        bball_16_copy = bball_16.copy()
        for row in random_data:
            bball_16_copy = bball_16_copy.append(row)
5.36 s ± 298 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

如果是通过列表的方式append，可以大大节省时间

```
In[20]: %%timeit
        bball_16_copy = bball_16.copy()
        bball_16_copy = bball_16_copy.append(random_data)
86.2 ms ± 3.71 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

2. 连接多个DataFrame

```
# 读取stocks_2016和stocks_2017两个数据集，用Symbol作为行索引名
In[21]: stocks_2016 = pd.read_csv('data/stocks_2016.csv', index_col='Symbol')
        stocks_2017 = pd.read_csv('data/stocks_2017.csv', index_col='Symbol')
In[22]: stocks_2016
Out[22]:
```

	Shares	Low	High
Symbol			
AAPL	80	95	110
TSLA	50	80	130
WMT	40	55	70

```
In[23]: stocks_2017  
Out[23]:
```

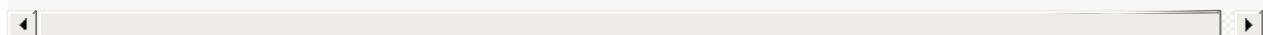
	Shares	Low	High
Symbol			
AAPL	50	120	140
GE	100	30	40
IBM	87	75	95
SLB	20	55	85
TXN	500	15	23
TSLA	100	100	300

```
# 将两个DataFrame放到一个列表中，用pandas的concat方法将它们连接起来  
In[24]: s_list = [stocks_2016, stocks_2017]  
pd.concat(s_list)  
Out[24]:
```

	Shares	Low	High
Symbol			
AAPL	80	95	110
TSLA	50	80	130
WMT	40	55	70
AAPL	50	120	140
GE	100	30	40
IBM	87	75	95
SLB	20	55	85
TXN	500	15	23
TSLA	100	100	300

keys参数可以给两个DataFrame命名，该标签会出现在行索引的最外层，会生成多层索引，names参数可以重命名每个索引层

```
In[25]: pd.concat(s_list, keys=['2016', '2017'], names=['Year', 'Symbol'])  
Out[25]:
```



			Shares	Low	High
Year	Symbol				
	AAPL		80	95	110
2016	TSLA		50	80	130
	WMT		40	55	70
	AAPL		50	120	140
2017	GE		100	30	40
	IBM		87	75	95
	SLB		20	55	85
	TXN		500	15	23
	TSLA		100	100	300

也可以横向连接。只要将axis参数设为columns或1

```
In[26]: pd.concat(s_list, keys=['2016', '2017'], axis='columns'
, names=['Year', None])
Out[26]:
```

Year	2016			2017		
	Shares	Low	High	Shares	Low	High
AAPL	80.0	95.0	110.0	50.0	120.0	140.0
GE	NaN	NaN	NaN	100.0	30.0	40.0
IBM	NaN	NaN	NaN	87.0	75.0	95.0
SLB	NaN	NaN	NaN	20.0	55.0	85.0
TSLA	50.0	80.0	130.0	100.0	100.0	300.0
TXN	NaN	NaN	NaN	500.0	15.0	23.0
WMT	40.0	55.0	70.0	NaN	NaN	NaN

```
# concat函数默认使用的是外连接，会保留每个DataFrame中的所有行。也可以通过
设定join参数，使用内连接：
```

```
In[27]: pd.concat(s_list, join='inner', keys=['2016', '2017'],
axis='columns', names=['Year', None])
Out[27]:
```

Symbol	2016			2017		
	Shares	Low	High	Shares	Low	High
AAPL	80	95	110	50	120	140
TSLA	50	80	130	100	100	300

更多

```
# append是concat方法的超简化版本，append内部其实就是调用concat。前本节的
第二个例子，pd.concat也可以如下实现：
```

```
In[28]: stocks_2016.append(stocks_2017)
Out[28]:
```

Symbol	Shares	Low	High
	Shares	Low	High
AAPL	80	95	110
TSLA	50	80	130
WMT	40	55	70
AAPL	50	120	140
GE	100	30	40
IBM	87	75	95
SLB	20	55	85
TXN	500	15	23
TSLA	100	100	300

```
# 原书没有下面三行代码
In[29]: stocks_2015 = stocks_2016.copy()
In[30]: stocks_2017
Out[30]:
```

Symbol	Shares	Low	High
AAPL	50	120	140
GE	100	30	40
IBM	87	75	95
SLB	20	55	85
TXN	500	15	23
TSLA	100	100	300

3. 比较特朗普和奥巴马的支持率

```
# pandas的read_html函数可以从网页抓取表格数据
In[31]: base_url = 'http://www.presidency.ucsb.edu/data/popular
ity.php?pres={}'
         trump_url = base_url.format(45)

         df_list = pd.read_html(trump_url)
         len(df_list)
Out[31]: 14
```

```
# 一共返回了14个表的DataFrame，取第一个
In[32]: df0 = df_list[0]
          df0.shape
Out[32]: (324, 1906)

In[33]: df0.head(7)
Out[33]:
```

	0	1	2	3	4	5	6	7	8	9	...	1896	1897
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN
6	Document Archive • Public Papers of the Presi... Presi...	Document Archive • Public Papers of the Presi... Presi...	Document Archive • Public Papers of the Presi... Presi...	NaN	NaN	Document Archive • Public Papers of the Presidents	• Public Papers of the Presidents	• State of the Union Addresses & Messages	• Inaugural Addresses	• Farewell Addresses	...	NaN	01/20/2017 01/2

7 rows × 1906 columns

```
# 用match参数匹配table中的字符串
In[34]: df_list = pd.read_html(trump_url, match='Start Date')
len(df_list)
Out[34]: 3
```

```
# 通过检查页面元素的属性，用attrs参数进行匹配
In[35]: df_list = pd.read_html(trump_url, match='Start Date', attrs={'align':'center'})
len(df_list)
Out[35]: 1
```

```
# 查看DataFrame的形状
In[36]: trump = df_list[0]
trump.shape
Out[36]: (265, 19)
```

```
In[37]: trump.head(8)
Out[37]:
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	>84 84-67 66-55 54-50 49-45 44-40 39-35 ...	>84	84-67	66-55	54-50	49-45	44-40	39-35	34-25	<25	NaN	NaN	NaN	NaN	NaN
1	>84	84-67	66-55	54-50	49-45	44-40	39-35	34-25	<25	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	%	%	%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	President	Start Date	End Date	NaN	Approving	Disapproving	unsure/no data	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
6	Donald J. Trump	10/09/2017	10/11/2017	NaN	37	57	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
7	NaN	10/08/2017	10/10/2017	NaN	37	56	7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

skiprows可以指定跳过一些行，header参数可以指定列名，用parse_dates指定开始和结束日期

```
In[38]: df_list = pd.read_html(trump_url, match='Start Date', attrs={'align':'center'}, header=0, skiprows=[0,1,2,3,5], parse_dates=['Start Date', 'End Date'])
trump = df_list[0]
trump.head()
```

Out[38]:

	President	Start Date	End Date	Unnamed: 3	Approving	Disapproving	unsure/no data	Unnamed: 7	Unnamed: 8	Unnamed: 9	Unnamed: 10	Unnamed: 11	Unnamed: 12	Unnamed: 13	Unnamed: 14
0	Donald J. Trump	2017-10-09	2017-10-11	NaN	37	57	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	NaN	2017-10-08	2017-10-10	NaN	37	56	7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	2017-10-07	2017-10-09	NaN	36	58	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	2017-10-06	2017-10-08	NaN	37	56	7	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	2017-10-05	2017-10-07	NaN	38	57	5	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

删除所有值都是缺失值的列

```
In[39]: trump = trump.dropna(axis=1, how='all')
trump.head()
```

Out[39]:

	President	Start Date	End Date	Approving	Disapproving	unsure/no data
0	Donald J. Trump	2017-10-09	2017-10-11	37	57	6
1		Nan	2017-10-08	2017-10-10	37	56
2		Nan	2017-10-07	2017-10-09	36	58
3		Nan	2017-10-06	2017-10-08	37	56
4		Nan	2017-10-05	2017-10-07	38	57

统计各列的缺失值个数

```
In[40]: trump.isnull().sum()
Out[40]: President      258
          Start Date     0
          End Date       0
          Approving      0
          Disapproving    0
          unsure/no data  0
          dtype: int64
```

缺失值向前填充

```
In[41]: trump = trump.fillna()
trump.head()
Out[41]:
```

	President	Start Date	End Date	Approving	Disapproving	unsure/no data
0	Donald J. Trump	2017-10-09	2017-10-11	37	57	6
1	Donald J. Trump	2017-10-08	2017-10-10	37	56	7
2	Donald J. Trump	2017-10-07	2017-10-09	36	58	6
3	Donald J. Trump	2017-10-06	2017-10-08	37	56	7
4	Donald J. Trump	2017-10-05	2017-10-07	38	57	5

确认数据类型

```
In[42]: trump.dtypes
Out[42]: President           object
          Start Date        datetime64[ns]
          End Date          datetime64[ns]
          Approving         int64
          Disapproving       int64
          unsure/no data    int64
          dtype: object
```

```
# 将前面的步骤做成一个函数，用于获取任意总统的信息
In[43]: def get_pres_appr(pres_num):
    base_url = 'http://www.presidency.ucsb.edu/data/popularity.php?pres={}'
    pres_url = base_url.format(pres_num)
    df_list = pd.read_html(pres_url, match='Start Date',
    attrs={'align':'center'},
                           header=0, skiprows=[0,1,2,3,5],
    parse_dates=['Start Date', 'End Date'])
    pres = df_list[0].copy()
    pres = pres.dropna(axis=1, how='all')
    pres['President'] = pres['President'].ffill()
    return pres.sort_values('End Date').reset_index(drop=True)

# 括号中的数字是总统的编号，奥巴马是44
In[44]: obama = get_pres_appr(44)
obama.head()
Out[44]:
```

	President	Start Date	End Date	Approving	Disapproving	unsure/no data
0	Barack Obama	2009-01-21	2009-01-23	68	12	21
1	Barack Obama	2009-01-22	2009-01-24	69	13	18
2	Barack Obama	2009-01-23	2009-01-25	67	14	19
3	Barack Obama	2009-01-24	2009-01-26	65	15	20
4	Barack Obama	2009-01-25	2009-01-27	64	16	20

```
# 获取最近五位总统的数据，输出每位的前三行数据
In[45]: pres_41_45 = pd.concat([get_pres_appr(x) for x in range(41,46)],
                           ignore_index=True)
pres_41_45.groupby('President').head(3)
Out[45]:
```

	President	Start Date	End Date	Approving	Disapproving	unsure/no data
0	George Bush	1989-01-24	1989-01-26	51	6	43
1	George Bush	1989-02-24	1989-02-27	60	11	27
2	George Bush	1989-02-28	1989-03-02	62	13	24
158	William J. Clinton	1993-01-24	1993-01-26	58	20	22
159	William J. Clinton	1993-01-29	1993-01-31	53	30	16
160	William J. Clinton	1993-02-12	1993-02-14	51	33	15
386	George W. Bush	2001-02-01	2001-02-04	57	25	18
387	George W. Bush	2001-02-09	2001-02-11	57	24	17
388	George W. Bush	2001-02-19	2001-02-21	61	21	16
656	Barack Obama	2009-01-21	2009-01-23	68	12	21
657	Barack Obama	2009-01-22	2009-01-24	69	13	18
658	Barack Obama	2009-01-23	2009-01-25	67	14	19

```
# 确认一下是否有一个日期对应多个支持率
```

```
In[46]: pres_41_45['End Date'].value_counts().head(8)
```

```
Out[46]: 1990-03-11    2
         1990-08-12    2
         1990-08-26    2
         2013-10-10    2
         1999-02-09    2
         1992-11-22    2
         1990-05-22    2
         2005-01-05    1
```

```
Name: End Date, dtype: int64
```

```
# 去除重复值
```

```
In[47]: pres_41_45 = pres_41_45.drop_duplicates(subset='End Date')
```

```
In[48]: pres_41_45.shape
```

```
Out[48]: (3695, 6)
```

```
# 对数据做简单的统计
In[49]: pres_41_45['President'].value_counts()
Out[49]:
Barack Obama      2786
George W. Bush    270
Donald J. Trump   259
William J. Clinton 227
George Bush        153
Name: President, dtype: int64

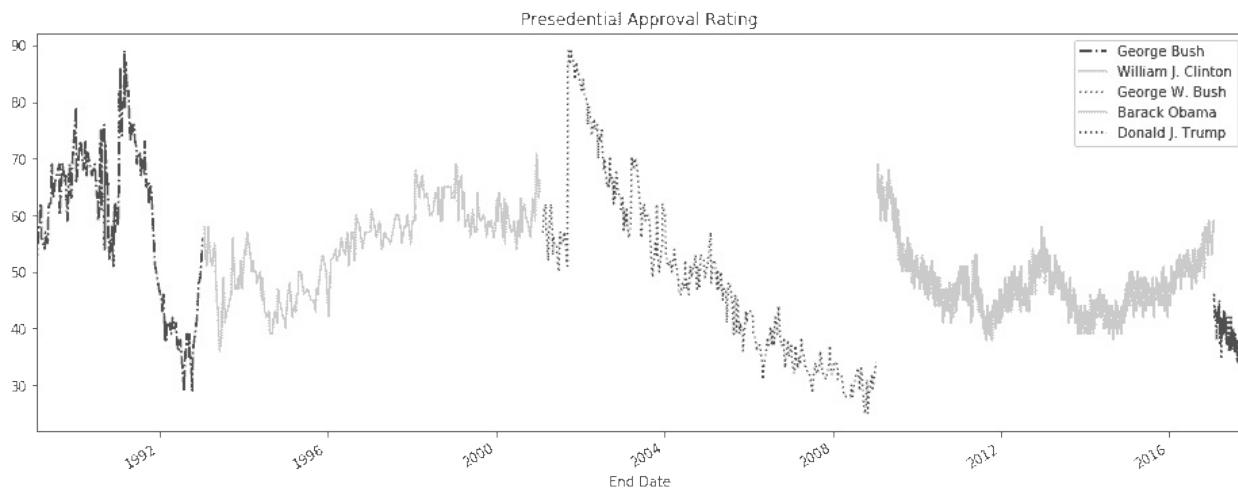
In[50]: pres_41_45.groupby('President', sort=False).median().round(1)
Out[50]:
```

President	Approving	Disapproving	unsure/no data
George Bush	62.0	22.0	9.0
William J. Clinton	57.0	36.0	6.0
George W. Bush	50.5	45.5	4.0
Barack Obama	47.0	47.0	7.0
Donald J. Trump	39.0	56.0	6.0

```
# 画出每任总统的支持率变化
In[51]: from matplotlib import cm
fig, ax = plt.subplots(figsize=(16, 6))

styles = [':', '--', ':', '--', ':']
colors = [.9, .3, .7, .3, .9]
groups = pres_41_45.groupby('President', sort=False)

for style, color, (pres, df) in zip(styles, colors, groups):
    df.plot('End Date', 'Approving', ax=ax, label=pres,
            style=style, color=cm.Greys(color),
            title='Presidential Approval Rating')
```



上面的图是将数据前后串起来，也可以用支持率对在职天数作图

```
In[52]: days_func = lambda x: x - x.iloc[0]
pres_41_45['Days in Office'] = pres_41_45.groupby('President') \
                                ['End Date'] \
                                .transform(day
s_func)
In[82]: pres_41_45['Days in Office'] = pres_41_45.groupby('President')['End Date'].transform(lambda x: x - x.iloc[0])
pres_41_45.groupby('President').head(3)
Out[82]:
```

	President	Start Date	End Date	Approving	Disapproving	unsure/no data	Days in Office
0	George Bush	1989-01-24	1989-01-26	51	6	43	0 days
1	George Bush	1989-02-24	1989-02-27	60	11	27	32 days
2	George Bush	1989-02-28	1989-03-02	62	13	24	35 days
158	William J. Clinton	1993-01-24	1993-01-26	58	20	22	0 days
159	William J. Clinton	1993-01-29	1993-01-31	53	30	16	5 days
160	William J. Clinton	1993-02-12	1993-02-14	51	33	15	19 days
386	George W. Bush	2001-02-01	2001-02-04	57	25	18	0 days
387	George W. Bush	2001-02-09	2001-02-11	57	24	17	7 days
388	George W. Bush	2001-02-19	2001-02-21	61	21	16	17 days
656	Barack Obama	2009-01-21	2009-01-23	68	12	21	0 days
657	Barack Obama	2009-01-22	2009-01-24	69	13	18	1 days
658	Barack Obama	2009-01-23	2009-01-25	67	14	19	2 days
3443	Donald J. Trump	2017-01-20	2017-01-22	45	45	10	0 days
3444	Donald J. Trump	2017-01-21	2017-01-23	45	46	9	1 days
3445	Donald J. Trump	2017-01-22	2017-01-24	46	45	9	2 days

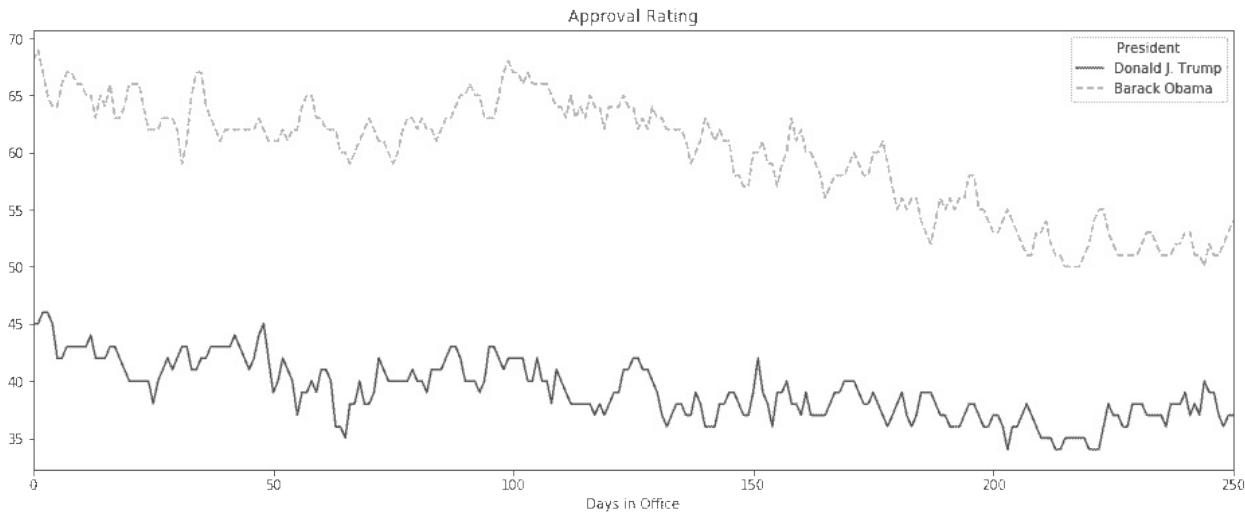
```
# 查看数据类型
In[83]: pres_41_45.dtypes
Out[83]: President          object
          Start Date      datetime64[ns]
          End Date       datetime64[ns]
          Approving        int64
          Disapproving      int64
          unsure/no data    int64
          Days in Office   timedelta64[ns]
          dtype: object
```

```
# Days in Office的数据类型是timedelta64[ns]，单位是纳秒，将其转换为整数
In[86]: pres_41_45['Days in Office'] = pres_41_45['Days in Office'].dt.days
          pres_41_45['Days in Office'].head()
Out[86]: 0    0
          1    32
          2    35
          3    43
          4    46
          Name: Days in Office, dtype: int64
```

```
# 转换数据，使每位总统的支持率各成一列
In[87]: pres_pivot = pres_41_45.pivot(index='Days in Office', columns='President', values='Approving')
          pres_pivot.head()
Out[87]:
```

	President	Barack Obama	Donald J. Trump	George Bush	George W. Bush	William J. Clinton
Days in Office						
0		68.0	45.0	51.0	57.0	58.0
1		69.0	45.0	NaN	NaN	NaN
2		67.0	46.0	NaN	NaN	NaN
3		65.0	46.0	NaN	NaN	NaN
4		64.0	45.0	NaN	NaN	NaN

```
# 只画出特朗普和奥巴马的支持率
In[88]: plot_kwargs = dict(figsize=(16,6), color=cm.gray([.3, .7]), style=['-', '--'], title='Approval Rating')
          pres_pivot.loc[:250, ['Donald J. Trump', 'Barack Obama']].ffill().plot(**plot_kwargs)
Out[88]: <matplotlib.axes._subplots.AxesSubplot at 0x1152254a8>
```



更多

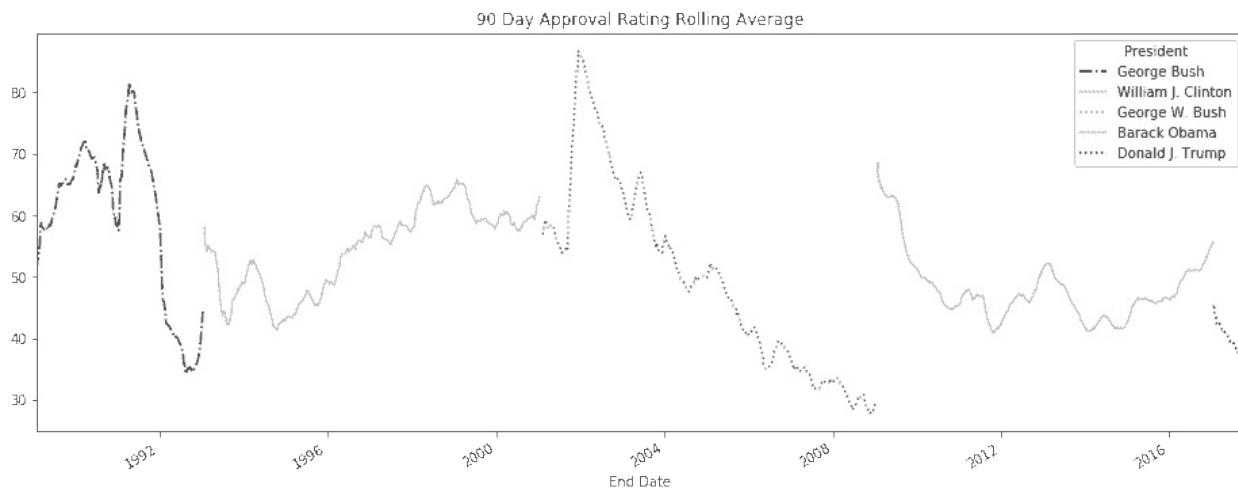
```
# rolling average方法可以平滑曲线，在这个例子中，使用的是90天求平均，参数on指明了滚动窗口是从哪列计算的
```

```
In[89]: pres_rm = pres_41_45.groupby('President', sort=False) \
          .rolling('90D', on='End Date')[['App
roving']] \
          .mean()
```

```
pres_rm.head()
Out[89]: President      End Date
          George Bush 1989-01-26    51.000000
                    1989-02-27    55.500000
                    1989-03-02    57.666667
                    1989-03-10    58.750000
                    1989-03-13    58.200000
Name: Approving, dtype: float64
```

```
# 对数据的行和列做调整，然后作图
```

```
In[90]: styles = [':', '-.', ':', '--', ':']
colors = [.9, .3, .7, .3, .9]
color = cm.Greys(colors)
title='90 Day Approval Rating Rolling Average'
plot_kw_args = dict(figsize=(16,6), style=styles, color
= color, title=title)
correct_col_order = pres_41_45.President.unique()
pres_rm.unstack('President')[correct_col_order].plot(**
plot_kw_args)
Out[90]: <matplotlib.axes._subplots.AxesSubplot at 0x1162d0780>
```



4. concat, join, 和merge的区别

`concat` :

- Pandas函数
- 可以垂直和水平地连接两个或多个pandas对象
- 只用索引对齐
- 索引出现重复值时会报错
- 默认是外连接（也可以设为内连接）

`join` :

- DataFrame方法
- 只能水平连接两个或多个pandas对象
- 对齐是靠被调用的DataFrame的列索引或行索引和另一个对象的行索引（不能是列索引）
- 通过笛卡尔积处理重复的索引值
- 默认是左连接（也可以设为内连接、外连接和右连接）

`merge` :

- DataFrame方法
- 只能水平连接两个DataFrame对象
- 对齐是靠被调用的DataFrame的列或行索引和另一个DataFrame的列或行索引
- 通过笛卡尔积处理重复的索引值
- 默认是内连接（也可以设为左连接、外连接、右连接）

```
# 用户自定义的display_frames函数，可以接收一列DataFrame，然后在一行中显示：
In[91]: from IPython.display import display_html

years = 2016, 2017, 2018
stock_tables = [pd.read_csv('data/stocks_{}.csv'.format(year), index_col='Symbol')
                 for year in years]

def display_frames(frames, num_spaces=0):
    t_style = '<table style="display: inline;"'
    tables_html = [df.to_html().replace('<table', t_style)
                  for df in frames]

    space = ' ' * num_spaces
    display_html(space.join(tables_html), raw=True)

display_frames(stock_tables, 30)
stocks_2016, stocks_2017, stocks_2018 = stock_tables
```

	Shares	Low	High		Shares	Low	High
Symbol				Symbol			
AAPL	80	95	110	AAPL	50	120	140
TSLA	50	80	130	GE	100	30	40
WMT	40	55	70	IBM	87	75	95
				SLB	20	55	85
				TXN	500	15	23
				TSLA	100	100	300
Shares Low High							
Symbol				Symbol			
AAPL	40	135	170	AAPL	50	120	140
AMZN	8	900	1125	GE	100	30	40
TSLA	50	220	400	IBM	87	75	95

```
# concat是唯一一个可以将DataFrames垂直连接起来的函数
In[92]: pd.concat(stock_tables, keys=[2016, 2017, 2018])
Out[92]:
```

			Shares	Low	High
		Symbol			
		AAPL	80	95	110
2016	TSLA		50	80	130
		WMT	40	55	70
		AAPL	50	120	140
		GE	100	30	40
2017	IBM		87	75	95
		SLB	20	55	85
		TXN	500	15	23
		TSLA	100	100	300
		AAPL	40	135	170
2018	AMZN		8	900	1125

```
# concat也可以将DataFrame水平连起来
In[93]: pd.concat(dict(zip(years, stock_tables)), axis='columns')
)
Out[93]:
```

	2016			2017			2018		
	Shares	Low	High	Shares	Low	High	Shares	Low	High
AAPL	80.0	95.0	110.0	50.0	120.0	140.0	40.0	135.0	170.0
AMZN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	900.0	1125.0
GE	NaN	NaN	NaN	100.0	30.0	40.0	NaN	NaN	NaN
IBM	NaN	NaN	NaN	87.0	75.0	95.0	NaN	NaN	NaN
SLB	NaN	NaN	NaN	20.0	55.0	85.0	NaN	NaN	NaN
TSLA	50.0	80.0	130.0	100.0	100.0	300.0	50.0	220.0	400.0
TXN	NaN	NaN	NaN	500.0	15.0	23.0	NaN	NaN	NaN
WMT	40.0	55.0	70.0	NaN	NaN	NaN	NaN	NaN	NaN

```
# 用join将DataFrame连起来；如果列名有相同的，需要设置lsuffix或rsuffix以进行区分
```

```
In[94]: stocks_2016.join(stocks_2017, lsuffix='_2016', rsuffix='_2017', how='outer')
```

```
Out[94]:
```

Symbol	Shares_2016	Low_2016	High_2016	Shares_2017	Low_2017	High_2017
AAPL	80.0	95.0	110.0	50.0	120.0	140.0
GE	NaN	NaN	NaN	100.0	30.0	40.0
IBM	NaN	NaN	NaN	87.0	75.0	95.0
SLB	NaN	NaN	NaN	20.0	55.0	85.0
TSLA	50.0	80.0	130.0	100.0	100.0	300.0
TXN	NaN	NaN	NaN	500.0	15.0	23.0
WMT	40.0	55.0	70.0	NaN	NaN	NaN

```
In[95]: stocks_2016
```

```
Out[95]:
```

	Shares	Low	High
Symbol			
AAPL	80	95	110
TSLA	50	80	130
WMT	40	55	70

```
# 要重现前面的concat方法，可以将一个DataFrame列表传入join
In[96]: other = [stocks_2017.add_suffix('_2017'), stocks_2018.add_suffix('_2018')]
          stocks_2016.add_suffix('_2016')).join(other, how='outer')
)
Out[96]:
```

	Shares_2016	Low_2016	High_2016	Shares_2017	Low_2017	High_2017	Shares_2018	Low_2018	High_2018
AAPL	80.0	95.0	110.0	50.0	120.0	140.0	40.0	135.0	170.0
AMZN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	900.0	1125.0
GE	NaN	NaN	NaN	100.0	30.0	40.0	NaN	NaN	NaN
IBM	NaN	NaN	NaN	87.0	75.0	95.0	NaN	NaN	NaN
SLB	NaN	NaN	NaN	20.0	55.0	85.0	NaN	NaN	NaN
TSLA	50.0	80.0	130.0	100.0	100.0	300.0	50.0	220.0	400.0
TXN	NaN	NaN	NaN	500.0	15.0	23.0	NaN	NaN	NaN
WMT	40.0	55.0	70.0	NaN	NaN	NaN	NaN	NaN	NaN

```
# 检验这两个方法是否相同
In[97]: stock_join = stocks_2016.add_suffix('_2016')).join(other
, how='outer')
          stock_concat = pd.concat(dict(zip(years, stock_tables)),
axis='columns')
In[98]: stock_concat.columns = stock_concat.columns.get_level_v
values(1) + '_' + \
stock_concat.columns.get_le
vel_values(0).astype(str)
In[99]: stock_concat
Out[99]:
```

	Shares_2016	Low_2016	High_2016	Shares_2017	Low_2017	High_2017	Shares_2018	Low_2018	High_2018
AAPL	80.0	95.0	110.0	50.0	120.0	140.0	40.0	135.0	170.0
AMZN	NaN	NaN	NaN	NaN	NaN	NaN	8.0	900.0	1125.0
GE	NaN	NaN	NaN	100.0	30.0	40.0	NaN	NaN	NaN
IBM	NaN	NaN	NaN	87.0	75.0	95.0	NaN	NaN	NaN
SLB	NaN	NaN	NaN	20.0	55.0	85.0	NaN	NaN	NaN
TSLA	50.0	80.0	130.0	100.0	100.0	300.0	50.0	220.0	400.0
TXN	NaN	NaN	NaN	500.0	15.0	23.0	NaN	NaN	NaN
WMT	40.0	55.0	70.0	NaN	NaN	NaN	NaN	NaN	NaN

```
In[100]: step1 = stocks_2016.merge(stocks_2017, left_index=True,
, right_index=True,
, how='outer', suffixes=('_201
6', '_2017'))
stock_merge = step1.merge(stocks_2018.add_suffix('_201
8'),
left_index=True, right_index=
True, how='outer')

stock_concat.equals(stock_merge)
Out[100]: True
```

◀ ▶

```
# 查看food_prices和food_transactions两个小数据集
In[101]: names = ['prices', 'transactions']
food_tables = [pd.read_csv('data/food_{}.csv'.format(name)) for name in names]
food_prices, food_transactions = food_tables
display_frames(food_tables, 30)
```

	item	store	price	Date	customer				
					custid	item	store	quantity	order
0	pear	A	0.99	2017	0	1	pear	A	5
1	pear	B	1.99	2017	1	1	banana	A	10
2	peach	A	2.99	2017	2	2	steak	B	3
3	peach	B	3.49	2017	3	2	pear	B	1
4	banana	A	0.39	2017	4	2	peach	B	2
5	banana	B	0.49	2017	5	2	steak	B	1
6	steak	A	5.99	2017	6	2	coconut	B	4
7	steak	B	6.99	2017					
8	steak	B	4.99	2015					

```
# 通过键item和store，将food_transactions和food_prices两个数据集融合
In[102]: food_transactions.merge(food_prices, on=['item', 'store'])
Out[102]:
```

	custid	item	store	quantity	price	Date
0	1	pear	A	5	0.99	2017
1	1	banana	A	10	0.39	2017
2	2	steak	B	3	6.99	2017
3	2	steak	B	3	4.99	2015
4	2	steak	B	1	6.99	2017
5	2	steak	B	1	4.99	2015
6	2	pear	B	1	1.99	2017
7	2	peach	B	2	3.49	2017

```
# 因为steak在两张表中分别出现了两次，融合时产生了笛卡尔积，造成结果中出现了四行steak；因为coconut没有对应的价格，造成结果中没有coconut
# 下面只融合2017年的数据
```

```
In[103]: food_transactions.merge(food_prices.query('Date == 2017'), how='left')
Out[103]:
```

	custid	item	store	quantity	price	Date
0	1	pear	A	5	0.99	2017.0
1	1	banana	A	10	0.39	2017.0
2	2	steak	B	3	6.99	2017.0
3	2	pear	B	1	1.99	2017.0
4	2	peach	B	2	3.49	2017.0
5	2	steak	B	1	6.99	2017.0
6	2	coconut	B	4	NaN	NaN

```
# 使用join复现上面的方法，需要将要连接的food_prices列转换为行索引
In[104]: food_prices_join = food_prices.query('Date == 2017').set_index(['item', 'store'])
          food_prices_join
Out[104]:
```

		price	Date
	item	store	
pear	A	0.99	2017
	B	1.99	2017
peach	A	2.99	2017
	B	3.49	2017
banana	A	0.39	2017
	B	0.49	2017
steak	A	5.99	2017
	B	6.99	2017

```
# join方法只对齐传入DataFrame的行索引，但可以对齐调用DataFrame的行索引和列索引；
# 要使用列做对齐，需要将其传给参数on
In[105]: food_transactions.join(food_prices_join, on=['item', 'store'])
Out[105]:
```

	custid	item	store	quantity	price	Date
0	1	pear	A	5	0.99	2017.0
1	1	banana	A	10	0.39	2017.0
2	2	steak	B	3	6.99	2017.0
3	2	pear	B	1	1.99	2017.0
4	2	peach	B	2	3.49	2017.0
5	2	steak	B	1	6.99	2017.0
6	2	coconut	B	4	NaN	NaN


```
# glob模块的glob函数可以将文件夹中的文件迭代取出，取出的是文件名字符串列表
，可以直接传给read_csv函数
```

```
In[107]: import glob
```

```
df_list = []
for filename in glob.glob('data/gas_prices/*.csv'):
    df_list.append(pd.read_csv(filename, index_col='Week',
                           parse_dates=['Week']))

gas = pd.concat(df_list, axis='columns')
gas.head()
```

```
Out[107]:
```

	All Grades	Diesel	Midgrade	Premium	Regular
Week					
2017-09-25	2.701	2.788	2.859	3.105	2.583
2017-09-18	2.750	2.791	2.906	3.151	2.634
2017-09-11	2.800	2.802	2.953	3.197	2.685
2017-09-04	2.794	2.758	2.946	3.191	2.679
2017-08-28	2.513	2.605	2.668	2.901	2.399

5. 连接SQL数据库

```
# 在读取chinook数据库之前，需要创建SQLAlchemy引擎
```

```
In[108]: from sqlalchemy import create_engine
```

```
engine = create_engine('sqlite:///data/chinook.db')
```

```
In[109]: tracks = pd.read_sql_table('tracks', engine)
```

```
tracks.head()
```

```
Out[109]:
```

TrackId	Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirksneider & W. Ho...	230619	3990994	0.99
3	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051	4331779	0.99
4	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99

```
# read_sql_table函数可以读取一张表，第一个参数是表名，第二个参数是引擎
In[110]: genres = pd.read_sql_table('genres', engine)
          genres.head()
Out[110]:
```

GenreId	Name
0	Rock
1	Jazz
2	Metal
3	Alternative & Punk
4	Rock And Roll

```
# 找到每种类型歌曲的平均时长
In[111]: genre_track = genres.merge(tracks[['GenreId', 'Milliseconds']],
                                     on='GenreId', how='left') \
.drop('GenreId', axis='columns')
genre_track.head()
Out[111]:
```

	Name	Milliseconds
0	Rock	343719
1	Rock	342562
2	Rock	230619
3	Rock	252051
4	Rock	375418

```
# 将Milliseconds列转变为timedelta数据类型
In[112]: genre_time = genre_track.groupby('Name')['Milliseconds'].mean()
          pd.to_timedelta(genre_time, unit='ms').dt.floor('s').sort_values()
Out[112]:
Name
Rock And Roll      00:02:14
Opera               00:02:54
Hip Hop/Rap        00:02:58
Easy Listening     00:03:09
Bossa Nova         00:03:39
R&B/Soul           00:03:40
World               00:03:44
Pop                 00:03:49
Latin               00:03:52
Alternative & Punk 00:03:54
Soundtrack          00:04:04
Reggae              00:04:07
Alternative         00:04:24
Blues               00:04:30
Rock                00:04:43
Jazz                00:04:51
Classical           00:04:53
Heavy Metal         00:04:57
Electronica/Dance   00:05:02
Metal               00:05:09
Comedy              00:26:25
TV Shows            00:35:45
Drama               00:42:55
Science Fiction     00:43:45
Sci Fi & Fantasy    00:48:31
Name: Milliseconds, dtype: timedelta64[ns]
```



```
# 找到每名顾客花费的总时长
In[113]: cust = pd.read_sql_table('customers', engine,
                                    columns=['CustomerId', 'FirstName',
                                              'LastName'])
          invoice = pd.read_sql_table('invoices', engine,
                                         columns=['InvoiceId', 'CustomerID'])
          ii = pd.read_sql_table('invoice_items', engine,
                                  columns=['InvoiceId', 'UnitPrice',
                                            'Quantity'])
In[114]: cust_inv = cust.merge(invoice, on='CustomerId') \
          .merge(ii, on='InvoiceId')
          cust_inv.head()
Out[114]:
```

CustomerId	FirstName	LastName	Invoiceld	UnitPrice	Quantity
0	1	Luís Gonçalves	98	1.99	1
1	1	Luís Gonçalves	98	1.99	1
2	1	Luís Gonçalves	121	0.99	1
3	1	Luís Gonçalves	121	0.99	1
4	1	Luís Gonçalves	121	0.99	1

```
# 现在可以用总量乘以单位价格，找到每名顾客的总消费
```

```
In[115]: total = cust_inv['Quantity'] * cust_inv['UnitPrice']
          cols = ['CustomerId', 'FirstName', 'LastName']
          cust_inv.assign(Total = total).groupby(cols)['Total']
          \
          .sum() \
          .sort_values(ascending=False)
```

```
else).head()
Out[115]:
```

```
CustomerId FirstName LastName      Total
6           Helena   Holý        49.62
26          Richard Cunningham 47.62
57          Luis     Rojas       46.62
46          Hugh     O'Reilly 45.62
45          Ladislav Kovács    45.62
Name: Total, dtype: float64
```

更多

```
# sql语句查询方法read_sql_query
```

```
In[116]: pd.read_sql_query('select * from tracks limit 5', engine)
Out[116]:
```

TrackId		Name	AlbumId	MediaTypeId	GenreId	Composer	Milliseconds	Bytes	UnitPrice
0	1	For Those About To Rock (We Salute You)	1	1	1	Angus Young, Malcolm Young, Brian Johnson	343719	11170334	0.99
1	2	Balls to the Wall	2	2	1	None	342562	5510424	0.99
2	3	Fast As a Shark	3	2	1	F. Baltes, S. Kaufman, U. Dirksneider & W. Ho...	230619	3990994	0.99
3	4	Restless and Wild	3	2	1	F. Baltes, R.A. Smith-Diesel, S. Kaufman, U. D...	252051	4331779	0.99
4	5	Princess of the Dawn	3	2	1	Deaffy & R.A. Smith-Diesel	375418	6290521	0.99

```
# 可以将长字符串传给read_sql_query
In[117]: sql_string1 = """
    select
        Name,
        time(avg(Milliseconds) / 1000, 'unixepoch') as avg
    _time
    from (
        select
            g.Name,
            t.Milliseconds
        from
            genres as g
        join
            tracks as t
        on
            g.genreid == t.genreid
    )
    group by
        Name
    order by
        avg_time
    ...
    pd.read_sql_query(sql_string1, engine)
Out[117]:
```

	Name	avg_time
0	Rock And Roll	00:02:14
1	Opera	00:02:54
2	Hip Hop/Rap	00:02:58
3	Easy Listening	00:03:09
4	Bossa Nova	00:03:39
5	R&B/Soul	00:03:40
6	World	00:03:44
7	Pop	00:03:49
8	Latin	00:03:52
9	Alternative & Punk	00:03:54
10	Soundtrack	00:04:04
11	Reggae	00:04:07

```
In[118]: sql_string2 = '''
    select
        c.customerid,
        c.FirstName,
        c.LastName,
        sum(ii.quantity * ii.unitprice) as Total
    from
        customers as c
    join
        invoices as i
            on c.customerid = i.customerid
    join
        invoice_items as ii
            on i.invoiceid = ii.invoiceid
    group by
        c.customerid, c.FirstName, c.LastName
    order by
        Total desc
    ...
    pd.read_sql_query(sql_string2, engine)
Out[118]:
```

	CustomerId	FirstName	LastName	Total
0	6	Helena	Holý	49.62
1	26	Richard	Cunningham	47.62
2	57	Luis	Rojas	46.62
3	45	Ladislav	Kovács	45.62
4	46	Hugh	O'Reilly	45.62
5	37	Fynn	Zimmermann	43.62
6	24	Frank	Ralston	43.62
7	28	Julia	Barnett	43.62
8	25	Victor	Stevens	42.62
9	7	Astrid	Gruber	42.62
10	44	Terhi	Hämäläinen	41.62
11	5	František	Wichterlová	40.62
12	43	Isabelle	Mercier	40.62
13	48	Johannes	Van der Berg	40.62
14	20	Dan	Miller	39.62
15	34	João	Fernandes	39.62
16	1	Luís	Gonçalves	39.62
17	3	François	Tremblay	39.62
18	4	Bjørn	Hansen	39.62
19	17	Jack	Smith	39.62
20	22	Heather	Leacock	39.62
21	42	Wyatt	Girard	39.62
22	15	Jennifer	Peterson	38.62

第10章 时间序列分析

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

%matplotlib inline
```

1. Python和Pandas日期工具的区别

```
# 引入datetime模块，创建date、time和datetime对象
In[2]: import datetime

    date = datetime.date(year=2013, month=6, day=7)
    time = datetime.time(hour=12, minute=30, second=19, microsecond=463198)
    dt = datetime.datetime(year=2013, month=6, day=7,
                           hour=12, minute=30, second=19, microsecond=463198)

    print("date is ", date)
    print("time is", time)
    print("datetime is", dt)

date is 2013-06-07
time is 12:30:19.463198
datetime is 2013-06-07 12:30:19.463198
```

```
# 创建并打印一个timedelta对象
In[3]: td = datetime.timedelta(weeks=2, days=5, hours=10, minutes=20,
                           seconds=6.73, milliseconds=99, microseconds=8)
        print(td)

19 days, 10:20:06.829008
```

```
# 将date和datetime，与timedelta做加减
In[4]: print('new date is', date + td)
        print('new datetime is', dt + td)

new date is 2013-06-26
new datetime is 2013-06-26 22:50:26.292206
```

```
# time和timedelta不能做加法
In[5]: time + td
-----
-----
TypeError                                 Traceback (most recent
call last)
<ipython-input-5-bd4e11db43bd> in <module>()
----> 1 time + td

TypeError: unsupported operand type(s) for +: 'datetime.time' an
d 'datetime.timedelta'
```

再来看一下pandas的Timestamp对象。Timestamp构造器比较灵活，可以处理多种输入

```
In[6]: pd.Timestamp(year=2012, month=12, day=21, hour=5, minute=
10, second=8, microsecond=99)
Out[6]: Timestamp('2012-12-21 05:10:08.000099')

In[7]: pd.Timestamp('2016/1/10')
Out[7]: Timestamp('2016-01-10 00:00:00')

In[8]: pd.Timestamp('2014-5/10')
Out[8]: Timestamp('2014-05-10 00:00:00')

In[9]: pd.Timestamp('Jan 3, 2019 20:45.56')
Out[9]: Timestamp('2019-01-03 20:45:33')

In[10]: pd.Timestamp('2016-01-05T05:34:43.123456789')
Out[10]: Timestamp('2016-01-05 05:34:43.123456789')
```

也可以传入一个整数或浮点数，表示距离1970年1月1日的时间

```
In[11]: pd.Timestamp(500)
Out[11]: Timestamp('1970-01-01 00:00:00.000000500')

In[12]: pd.Timestamp(5000, unit='D')
Out[12]: Timestamp('1983-09-10 00:00:00')
```

```
# pandas的to_datetime函数与Timestamp类似，但有些参数不同
In[13]: pd.to_datetime('2015-5-13')
Out[13]: Timestamp('2015-05-13 00:00:00')

In[14]: pd.to_datetime('2015-13-5', dayfirst=True)
Out[14]: Timestamp('2015-05-13 00:00:00')

In[15]: pd.Timestamp('Saturday September 30th, 2017')
Out[15]: Timestamp('2017-09-30 00:00:00')

In[16]: pd.to_datetime('Start Date: Sep 30, 2017 Start Time: 1:30 pm', format='Start Date: %b %d, %Y Start Time: %I:%M %p')
Out[16]: Timestamp('2017-09-30 13:30:00')

In[17]: pd.to_datetime(100, unit='D', origin='2013-1-1')
Out[17]: Timestamp('2013-04-11 00:00:00')
```

```
# to_datetime可以将一个字符串或整数列表或Series转换为时间戳
In[18]: s = pd.Series([10, 100, 1000, 10000])
         pd.to_datetime(s, unit='D')
Out[18]: 0    1970-01-11
         1    1970-04-11
         2    1972-09-27
         3    1997-05-19
       dtype: datetime64[ns]

In[19]: s = pd.Series(['12-5-2015', '14-1-2013', '20/12/2017',
        '40/23/2017'])
         pd.to_datetime(s, dayfirst=True, errors='coerce')
Out[19]: 0    2015-05-12
         1    2013-01-14
         2    2017-12-20
         3        NaT
       dtype: datetime64[ns]

In[20]: pd.to_datetime(['Aug 3 1999 3:45:56', '10/31/2017'])
Out[20]: DatetimeIndex(['1999-08-03 03:45:56', '2017-10-31 00:00:00'],
       dtype='datetime64[ns]', freq=None)
```

```

# Pandas的Timedelta和to_timedelta也可以用来表示一定的时间量。
# to_timedelta函数可以产生一个Timedelta对象。
# 与to_datetime类似，to_timedelta也可以转换列表或Series变成Timedelta
# 对象。
In[21]: pd.Timedelta('12 days 5 hours 3 minutes 123456789 nanos
                     seconds')
Out[21]: Timedelta('12 days 05:03:00.123456')

In[22]: pd.Timedelta(days=5, minutes=7.34)
Out[22]: Timedelta('5 days 00:07:20.400000')

In[23]: pd.Timedelta(100, unit='W')
Out[23]: Timedelta('700 days 00:00:00')

In[24]: pd.to_timedelta('5 dayz', errors='ignore')
Out[24]: '5 dayz'

In[25]: pd.to_timedelta('67:15:45.454')
Out[25]: Timedelta('2 days 19:15:45.454000')

In[26]: s = pd.Series([10, 100])
         pd.to_timedelta(s, unit='s')
Out[26]: 0    00:00:10
         1    00:01:40
         dtype: timedelta64[ns]

In[27]: time_strings = ['2 days 24 minutes 89.67 seconds', '00:
        45:23.6']
         pd.to_timedelta(time_strings)
Out[27]: TimedeltaIndex(['2 days 00:25:29.670000', '0 days 00:45
        :23.600000'], dtype='timedelta64[ns]', freq=None)

```

```

# Timedeltas对象可以和Timestamps互相加减，甚至可以相除返回一个浮点数
In[28]: pd.Timedelta('12 days 5 hours 3 minutes') * 2
Out[28]: Timedelta('24 days 10:06:00')

In[29]: pd.Timestamp('1/1/2017') + pd.Timedelta('12 days 5 hour
s 3 minutes') * 2
Out[29]: Timestamp('2017-01-25 10:06:00')

In[30]: td1 = pd.to_timedelta([10, 100], unit='s')
         td2 = pd.to_timedelta(['3 hours', '4 hours'])
         td1 + td2
Out[30]: TimedeltaIndex(['03:00:10', '04:01:40'], dtype='timedelta64[ns]', freq=None)

In[31]: pd.Timedelta('12 days') / pd.Timedelta('3 days')
Out[31]: 4.0

```

```
# Timestamps 和 Timedeltas有许多可用的属性和方法，下面列举了一些：
In[32]: ts = pd.Timestamp('2016-10-1 4:23:23.9')
In[33]: ts.ceil('h')
Out[33]: Timestamp('2016-10-01 05:00:00')

In[34]: ts.year, ts.month, ts.day, ts.hour, ts.minute, ts.second
Out[34]: (2016, 10, 1, 4, 23, 23)

In[35]: ts.dayofweek, ts.dayofyear, ts.daysinmonth
Out[35]: (5, 275, 31)

In[36]: ts.to_pydatetime()
Out[36]: datetime.datetime(2016, 10, 1, 4, 23, 23, 900000)

In[37]: td = pd.Timedelta(125.8723, unit='h')
td
Out[37]: Timedelta('5 days 05:52:20.280000')

In[38]: td.round('min')
Out[38]: Timedelta('5 days 05:52:00')

In[39]: td.components
Out[39]: Components(days=5, hours=5, minutes=52, seconds=20, milliseconds=280, microseconds=0, nanoseconds=0)

In[40]: td.total_seconds()
Out[40]: 453140.28
```

更多

```
# 对比一下，在使用和没使用格式指令的条件下，将字符串转换为Timestamps对象的速度
In[41]: date_string_list = ['Sep 30 1984'] * 10000

In[42]: %timeit pd.to_datetime(date_string_list, format='%b %d %Y')
37.8 ms ± 556 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)

In[43]: %timeit pd.to_datetime(date_string_list)
1.33 s ± 57.9 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

2. 智能切分时间序列

从hdf5文件crime.h5读取丹佛市的crimes数据集，输出列数据的数据类型和数据的前几行

```
In[44]: crime = pd.read_hdf('data/crime.h5', 'crime')
crime.dtypes
Out[44]:
OFFENSE_TYPE_ID           category
OFFENSE_CATEGORY_ID       category
REPORTED_DATE            datetime64[ns]
GEO_LON                   float64
GEO_LAT                   float64
NEIGHBORHOOD_ID          category
IS_CRIME                  int64
IS_TRAFFIC                int64
dtype: object
```

```
In[45]: crime = crime.set_index('REPORTED_DATE')
crime.head()
```

Out[45]:

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2014-06-29 02:01:00	traffic-accident-dui-duid	traffic-accident	-105.000149	39.745753	cbd	0	
2014-06-29 01:54:00	vehicular-eluding-no-chase	all-other-crimes	-104.884660	39.738702	east-colfax	1	
2014-06-29 02:00:00	disturbing-the-peace	public-disorder	-105.020719	39.706674	athmar-park	1	
2014-06-29 02:18:00	curfew	public-disorder	-105.001552	39.769505	sunnyside	1	
2014-06-29 04:17:00	aggravated-assault	aggravated-assault	-105.018557	39.679229	college-view-south-platte	1	

注意到有三个类型列和一个Timestamp对象列，这些数据的数据类型在创建时就建立了对应的数据类型。

这和CSV文件非常不同，CSV文件保存的只是字符串。

由于前面已经将REPORTED_DATE设为了行索引，所以就可以进行智能Timestamp对象切分。

```
In[46]: pd.options.display.max_rows = 4
```

```
In[47]: crime.loc['2016-05-12 16:45:00']
```

Out[47]:

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2016-05-12 16:45:00	traffic-accident	traffic-accident	-104.847024	39.779596	montbello	0	
2016-05-12 16:45:00	traffic-accident	traffic-accident	-105.049180	39.769296	west-highland	0	
2016-05-12 16:45:00	fraud-identity-theft	white-collar-crime	-104.931971	39.717359	hilltop	1	

可以进行时间部分匹配

In[48]: crime.loc['2016-05-12']

Out[48]:

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2016-05-12 23:51:00	criminal-mischief-other	public-disorder	-105.017241	39.705845	athmar-park	1	
2016-05-12 18:40:00	liquor-possession	drug-alcohol	-104.995692	39.747875	cbd	1	
...
2016-05-12 15:59:00	menacing-felony-w-weap	aggravated-assault	-104.935172	39.723703	hilltop	1	
2016-05-12 16:39:00	assault-dv	other-crimes-against-persons	-104.974700	39.740555	north-capitol-hill	1	

243 rows × 7 columns



也可以选取一整月、一整年或某天的某小时

In[49]: crime.loc['2016-05'].shape

Out[49]: (8012, 7)

In[50]: crime.loc['2016'].shape

Out[50]: (91076, 7)

In[51]: crime.loc['2016-05-12 03'].shape

Out[51]: (4, 7)

也可以包含月的名字

In[52]: crime.loc['Dec 2015'].sort_index()

Out[52]:

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2015-12-01 00:48:00	drug-cocaine-possess	drug-alcohol	-104.891681	39.740155	east-colfax	1	
2015-12-01 00:48:00	theft-of-motor-vehicle	auto-theft	-104.891681	39.740155	east-colfax	1	
...
2015-12-31 23:45:00	violation-of-restraining-order	all-other-crimes	-105.034887	39.741827	west-colfax	1	
2015-12-31 23:50:00	weapon-poss-illegal-dangerous	all-other-crimes	-105.032769	39.709188	westwood	1	

6907 rows × 7 columns



```
# 其它一些字符串的格式也可行
In[53]: crime.loc['2016 Sep, 15'].shape
Out[53]: (252, 7)

In[54]: crime.loc['21st October 2014 05'].shape
Out[54]: (4, 7)
```

```
# 可以进行切片
In[55]: crime.loc['2015-3-4':'2016-1-1'].sort_index()
Out[55]:
```

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2015-03-04 00:11:00	assault-dv	other-crimes-against-persons	-105.021966	39.770883	sunnyside	1	
2015-03-04 00:19:00	assault-dv	other-crimes-against-persons	-104.978988	39.748799	five-points	1	
...
2016-01-01 23:45:00	drug-cocaine-possess	drug-alcohol	-104.987310	39.753598	five-points	1	
2016-01-01 23:48:00	drug-poss-paraphernalia	drug-alcohol	-104.986020	39.752541	five-points	1	

75403 rows × 7 columns

```
# 提供更为精确的时间
In[56]: crime.loc['2015-3-4 22':'2016-1-1 23:45:00'].sort_index()
()
Out[56]:
```

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2015-03-04 22:25:00	traffic-accident-hit-and-run	traffic-accident	-104.973896	39.769064	five-points	0	
2015-03-04 22:30:00	traffic-accident	traffic-accident	-104.906412	39.632816	hampden-south	0	
...
2015-03-04 23:40:00	robbery-business	robbery	-105.039236	39.726157	villa-park	1	
2016-01-01 23:45:00	drug-cocaine-possess	drug-alcohol	-104.987310	39.753598	five-points	1	

75175 rows × 7 columns

原理

```
# hdf5文件可以保存每一列的数据类型，可以极大减少内存的使用。
# 在上面的例子中，三个列被存成了类型，而不是对象。存成对象的话，消耗的内存会
变为之前的四倍。
In[57]: mem_cat = crime.memory_usage().sum()
         mem_obj = crime.astype({'OFFENSE_TYPE_ID':'object',
                                  'OFFENSE_CATEGORY_ID':'object',
                                  'NEIGHBORHOOD_ID':'object'}).me
         mory_usage(deep=True)\ \
                               .sum()
         mb = 2 ** 20
         round(mem_cat / mb, 1), round(mem_obj / mb, 1)
Out[57]: (29.4, 122.7)
```

```
# 为了用日期智能选取和切分，行索引必须包含日期。
# 在前面的例子中，REPORTED_DATE被设成了行索引，行索引从而成了DatetimeIndex对象。
In[58]: crime.index[:2]
Out[58]: DatetimeIndex(['2014-06-29 02:01:00', '2014-06-29 01:54
:00'], dtype='datetime64[ns]', name='REPORTED_DATE', freq=None)
```

更多

```
# 对行索引进行排序，可以极大地提高速度
In[59]: %timeit crime.loc['2015-3-4':'2016-1-1']
          42.4 ms ± 865 µs per loop (mean ± std. dev. of 7 runs,
          10 loops each)

In[60]: crime_sort = crime.sort_index()

In[61]: %timeit crime_sort.loc['2015-3-4':'2016-1-1']
          840 µs ± 32.3 µs per loop (mean ± std. dev. of 7 runs,
          1000 loops each)

In[62]: pd.options.display.max_rows = 60
```

3. 只使用适用于DatetimeIndex的方法

```
# 读取crime hdf5数据集，行索引设为REPORTED_DATE，检查其数据类型
In[63]: crime = pd.read_hdf('data/crime.h5', 'crime').set_index
         ('REPORTED_DATE')
         print(type(crime.index))
<class 'pandas.core.indexes.datetimes.DatetimeIndex'>
```

```
# 用between_time方法选取发生在凌晨2点到5点的案件
In[64]: crime.between_time('2:00', '5:00', include_end=False).head()
Out[64]:
```

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2014-06-29 02:01:00	traffic-accident-dui-duid	traffic-accident	-105.000149	39.745753	cbd	0	
2014-06-29 02:00:00	disturbing-the-peace	public-disorder	-105.020719	39.706674	athmar-park	1	
2014-06-29 02:18:00	curfew	public-disorder	-105.001552	39.769505	sunnyside	1	
2014-06-29 04:17:00	aggravated-assault	aggravated-assault	-105.018557	39.679229	college-view-south-platte	1	
2014-06-29 04:22:00	violation-of-restraining-order	all-other-crimes	-104.972447	39.739449	cheesman-park	1	

```
# 用at_time方法选取特定时间
In[65]: crime.at_time('5:47').head()
Out[65]:
```

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
REPORTED_DATE							
2013-11-26 05:47:00	criminal-mischief-other	public-disorder	-104.991476	39.751536	cbd	1	
2017-04-09 05:47:00	criminal-mischief-mtr-veh	public-disorder	-104.959394	39.678425	university	1	
2017-02-19 05:47:00	criminal-mischief-other	public-disorder	-104.986767	39.741336	north-capitol-hill	1	
2017-02-16 05:47:00	aggravated-assault	aggravated-assault	-104.934029	39.732320	hale	1	
2017-02-12 05:47:00	police-interference	all-other-crimes	-104.976306	39.722644	speer	1	

```
# first方法可以选取排在前面的n个时间
# 首先将时间索引排序，然后使用pd.offsets模块
In[66]: crime_sort = crime.sort_index()
In[67]: pd.options.display.max_rows = 6
In[68]: crime_sort.first(pd.offsets.MonthBegin(6))
Out[68]:
```

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TF
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...	
2012-06-30 23:50:00	criminal-mischief-mtr-veh	public-disorder	-104.838271	39.788683	montbello	1	
2012-06-30 23:54:00	traffic-accident-hit-and-run	traffic-accident	-105.014162	39.740439	lincoln-park	0	
2012-07-01 00:01:00	robbery-street	robbery	-104.924292	39.767585	northeast-park-hill	1	

```
# 前面的结果最后一条是7月的数据，这是因为pandas使用的是行索引中的第一个值，也就是2012-01-02 00:06:00
```

```
# 下面使用MonthEnd
```

```
In[69]: crime_sort.first(pd.offsets.MonthEnd(6))
```

```
Out[69]:
```

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TF
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...	
2012-06-29 23:41:00	robbery-street	robbery	-104.991912	39.756163	five-points	1	
2012-06-29 23:57:00	assault-simple	other-crimes-against-persons	-104.987360	39.715162	speer	1	
2012-06-30 00:04:00	traffic-accident	traffic-accident	-104.894697	39.628902	hampden-south	0	

```
# 上面的结果中，6月30日的数据只有一条，这也是因为第一个时间值的原因。
```

```
# 所有的DateOffsets对象都有一个normalize参数，当其设为True时，会将所有时间归零。
```

```
# 下面就是我们想获得的结果
```

```
In[70]: crime_sort.first(pd.offsets.MonthBegin(6, normalize=True))
```

```
Out[70]:
```

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...
2012-06-30 23:44:00	traffic-accident	traffic-accident	-104.987578	39.711158	baker	0	
2012-06-30 23:50:00	criminal-mischief-mtr-veh	public-disorder	-104.838271	39.788683	montbello	1	
2012-06-30 23:54:00	traffic-accident-hit-and-run	traffic-accident	-105.014162	39.740439	lincoln-park	0	

27488 rows × 7 columns



选取2012-06的数据

In[71]: crime_sort.loc[:'2012-06']

Out[71]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...
2012-06-30 23:44:00	traffic-accident	traffic-accident	-104.987578	39.711158	baker	0	
2012-06-30 23:50:00	criminal-mischief-mtr-veh	public-disorder	-104.838271	39.788683	montbello	1	
2012-06-30 23:54:00	traffic-accident-hit-and-run	traffic-accident	-105.014162	39.740439	lincoln-park	0	

一些时间差的别名 <http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases>

5天

In[72]: crime_sort.first('5D')

Out[72]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TF
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...	
2012-01-06 23:30:00	assault-dv	other-crimes-against-persons	-104.958983	39.674135	university-park	1	
2012-01-06 23:44:00	theft-of-motor-vehicle	auto-theft	-104.845356	39.794035	montbello	1	
2012-01-06 23:55:00	threats-to-injure	public-disorder	-105.004788	39.708714	athmar-park	1	

5个工作日

In[73]: crime_sort.first('5B')

Out[73]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TF
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...	
2012-01-08 23:52:00	theft-other	larceny	-104.968227	39.739752	cheesman-park	1	
2012-01-09 00:04:00	traffic-accident-hit-and-run	traffic-accident	-104.973343	39.760757	five-points	0	
2012-01-09 00:05:00	fraud-criminal-impersonation	white-collar-crime	-105.024676	39.712702	valverde	1	

7周

In[74]: crime_sort.first('7W')

Out[74]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...
2012-02-18 22:20:00	traffic-accident-dui-duid	traffic-accident	-104.919946	39.761917	north-park-hill	0	
2012-02-18 22:44:00	criminal-mischief-mtr-veh	public-disorder	-105.044984	39.736776	west-colfax	1	
2012-02-18 23:27:00	theft-items-from-vehicle	theft-from-motor-vehicle	-105.009018	39.708701	athmar-park	1	

第3季度开始

In[75]: crime_sort.first('3QS')

Out[75]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TR
2012-01-02 00:06:00	aggravated-assault	aggravated-assault	-104.816860	39.796717	montbello	1	
2012-01-02 00:06:00	violation-of-restraining-order	all-other-crimes	-104.816860	39.796717	montbello	1	
2012-01-02 00:16:00	traffic-accident-dui-duid	traffic-accident	-104.971851	39.736874	cheesman-park	0	
...
2012-09-30 23:29:00	theft-of-motor-vehicle	auto-theft	-104.988838	39.686925	overland	1	
2012-09-30 23:41:00	traffic-accident-hit-and-run	traffic-accident	-105.087598	39.638462	marston	0	
2012-09-30 23:43:00	robbery-business	robbery	-104.772712	39.781966	gateway-green-valley-ranch	1	

43045 rows × 7 columns

原理

使用datetime模块的time对象

In[76]: import datetime
crime.between_time(datetime.time(2,0), datetime.time(5,0), include_end=False)

Out[76]:

REPORTED_DATE	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_TF
2014-06-29 02:01:00	traffic-accident-dui-duid	traffic-accident	-105.000149	39.745753	cbd	0	
2014-06-29 02:00:00	disturbing-the-peace	public-disorder	-105.020719	39.706674	athmar-park	1	
2014-06-29 02:18:00	curfew	public-disorder	-105.001552	39.769505	sunnyside	1	
...
2017-09-13 02:21:00	assault-simple	other-crimes-against-persons	-104.925733	39.654184	university-hills	1	
2017-09-13 03:21:00	traffic-accident-dui-duid	traffic-accident	-105.010711	39.757385	highland	0	
2017-09-13 02:15:00	traffic-accident-hit-and-run	traffic-accident	-105.043950	39.787436	regis	0	

```
# 选取第一个时间
# 用两种方法加六个月
In[77]: first_date = crime_sort.index[0]
first_date
Out[77]: Timestamp('2012-01-02 00:06:00')

In[78]: first_date + pd.offsets.MonthBegin(6)
Out[78]: Timestamp('2012-07-01 00:06:00')

In[79]: first_date + pd.offsets.MonthEnd(6)
Out[79]: Timestamp('2012-06-30 00:06:00')
```

更多

```
# 使用自定义的DateOffset对象
In[80]: dt = pd.Timestamp('2012-1-16 13:40')
dt + pd.DateOffset(months=1)
Out[80]: Timestamp('2012-02-16 13:40:00')
```

```
# 一个使用更多日期和时间的例子
In[81]: do = pd.DateOffset(years=2, months=5, days=3, hours=8,
seconds=10)
pd.Timestamp('2012-1-22 03:22') + do
Out[81]: Timestamp('2014-06-25 11:22:10')
```

```
In[82]: pd.options.display.max_rows=60
```

4. 计算每周的犯罪数

```
# 读取crime数据集，行索引设定为REPORTED_DATE，然后对行索引排序，以提高后续运算的速度
```

```
In[83]: crime_sort = pd.read_hdf('data/crime.h5', 'crime') \
    .set_index('REPORTED_DATE') \
    .sort_index()
```

```
# 为了统计每周的犯罪数，需要按周分组
# resample方法可以用DateOffset对象或别名，即可以在所有返回的对象分组上操作
```

```
In[84]: crime_sort.resample('W')
Out[84]: DatetimeIndexResampler [freq=<Week: weekday=6>, axis=0,
closed=right, label=right, convention=start, base=0]
```

```
# size()可以查看每个分组的大小
In[85]: weekly_crimes = crime_sort.resample('W').size()
weekly_crimes.head()
Out[85]: REPORTED_DATE
2012-01-08    877
2012-01-15   1071
2012-01-22    991
2012-01-29    988
2012-02-05    888
Freq: W-SUN, dtype: int64
```

```
# len()也可以查看大小
```

```
In[86]: len(crime_sort.loc[:'2012-1-8'])
Out[86]: 877
```

```
In[87]: len(crime_sort.loc['2012-1-9':'2012-1-15'])
Out[87]: 1071
```

```
# 用周四作为每周的结束
```

```
In[88]: crime_sort.resample('W-THU').size().head()
Out[88]: REPORTED_DATE
2012-01-05    462
2012-01-12   1116
2012-01-19    924
2012-01-26   1061
2012-02-02    926
Freq: W-THU, dtype: int64
```

```
# groupby方法可以重现上面的resample，唯一的不同是要在pd.Grouper对象中传入抵消值
In[89]: weekly_crimes_gby = crime_sort.groupby(pd.Grouper(freq='W')).size()
         weekly_crimes_gby.head()
Out[89]: REPORTED_DATE
2012-01-08      877
2012-01-15     1071
2012-01-22      991
2012-01-29      988
2012-02-05      888
Freq: W-SUN, dtype: int64
```

```
# 判断两个方法是否相同
In[90]: weekly_crimes.equals(weekly_crimes_gby)
Out[90]: True
```

原理

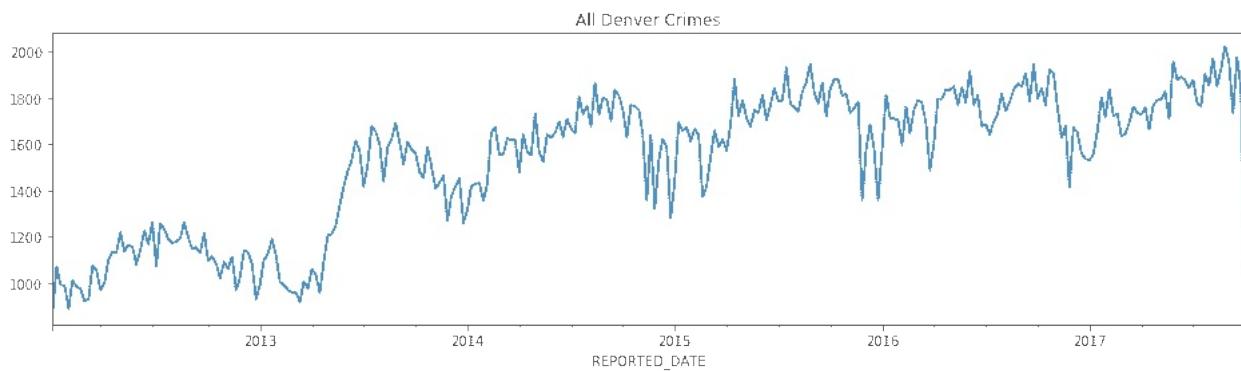
```
# 输出resample对象的所有方法
In[91]: r = crime_sort.resample('W')
         resample_methods = [attr for attr in dir(r) if attr[0].islower()]
         print(resample_methods)
['agg', 'aggregate', 'apply', 'asfreq', 'ax', 'backfill', 'bfill',
 'count', 'ffill', 'fillna', 'first', 'get_group', 'groups', 'indices',
 'interpolate', 'last', 'max', 'mean', 'median', 'min',
 'ndim', 'ngroups', 'nunique', 'obj', 'ohlc', 'pad', 'plot', 'prod',
 'sem', 'size', 'std', 'sum', 'transform', 'var']
```

更多

```
# 可以通过resample方法的on参数，来进行分组
In[92]: crime = pd.read_hdf('data/crime.h5', 'crime')
         weekly_crimes2 = crime.resample('W', on='REPORTED_DATE').size()
         weekly_crimes2.equals(weekly_crimes)
Out[92]: True
```

```
# 也可以通过pd.Grouper的参数key设为Timestamp，来进行分组
In[93]: weekly_crimes_gby2 = crime.groupby(pd.Grouper(key='REPORTED_DATE', freq='W')).size()
         weekly_crimes_gby2.equals(weekly_crimes_gby)
Out[93]: True
```

```
# 可以很方便地利用这个数据画出线图
In[94]: weekly_crimes.plot(figsize=(16,4), title='All Denver Crimes')
Out[94]: <matplotlib.axes._subplots.AxesSubplot at 0x10b8d3240>
```



5. 分别聚合每周犯罪和交通事故数据

```
# 读取crime数据集，行索引设为REPORTED_DATE，对行索引排序
In[95]: crime_sort = pd.read_hdf('data/crime.h5', 'crime') \
          .set_index('REPORTED_DATE') \
          .sort_index()
```

```
# 按季度分组，分别统计'IS_CRIME'和'IS_TRAFFIC'的和
In[96]: crime_quarterly = crime_sort.resample('Q')[['IS_CRIME', 'IS_TRAFFIC']].sum()
          crime_quarterly.head()
Out[96]:
```

	IS_CRIME	IS_TRAFFIC
REPORTED_DATE		
2012-03-31	7882	4726
2012-06-30	9641	5255
2012-09-30	10566	5003
2012-12-31	9197	4802
2013-03-31	8730	4442

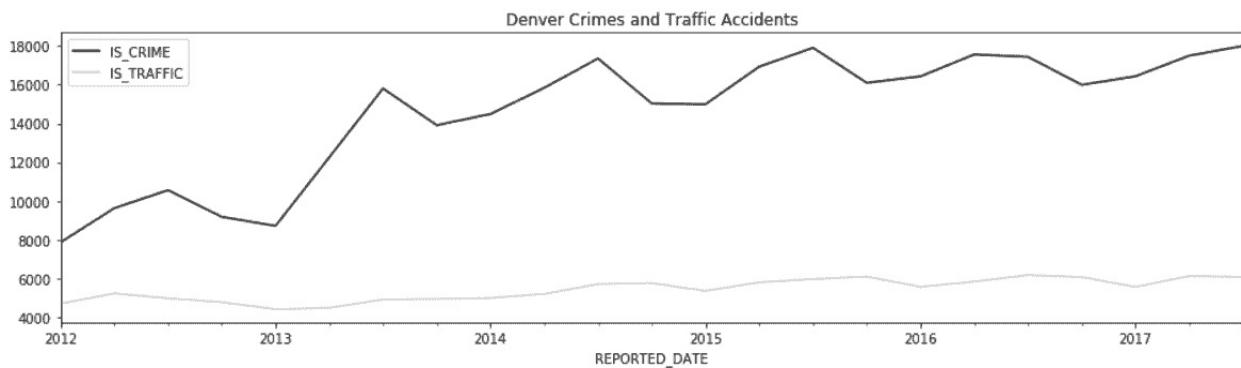
```
# 所有日期都是该季度的最后一天，使用QS来生成每季度的第一天
In[97]: crime_sort.resample('QS')[['IS_CRIME', 'IS_TRAFFIC']].sum()
().head()
Out[97]:
```

	IS_CRIME	IS_TRAFFIC
REPORTED_DATE		
2012-01-01	7882	4726
2012-04-01	9641	5255
2012-07-01	10566	5003
2012-10-01	9197	4802
2013-01-01	8730	4442

```
# 通过检查第二季度的数据，验证结果是否正确
In[98]: crime_sort.loc['2012-4-1':'2012-6-30', ['IS_CRIME', 'IS_TRAFFIC']].sum()
Out[98]: IS_CRIME      9641
          IS_TRAFFIC    5255
          dtype: int64
```

```
# 用groupby重现上述方法
In[99]: crime_quarterly2 = crime_sort.groupby(pd.Grouper(freq='Q'))[['IS_CRIME', 'IS_TRAFFIC']].sum()
         crime_quarterly2.equals(crime_quarterly)
Out[99]: True
```

```
# 作图来分析犯罪和交通事故的趋势
In[100]: plot_kwarg = dict(figsize=(16,4),
                           color=['black', 'lightgrey'],
                           title='Denver Crimes and Traffic Accidents')
           crime_quarterly.plot(**plot_kwarg)
Out[100]: <matplotlib.axes._subplots.AxesSubplot at 0x10b8d12e8>
```



原理

```
# 如果不选择IS_CRIME和IS_TRAFFIC两列，则所有的数值列都会求和
In[101]: crime_sort.resample('Q').sum().head()
Out[101]:
```

	GEO_LON	GEO_LAT	IS_CRIME	IS_TRAFFIC
REPORTED_DATE				
2012-03-31	-1.313006e+06	496960.237747	7882	4726
2012-06-30	-1.547274e+06	585656.789182	9641	5255
2012-09-30	-1.615835e+06	611604.800384	10566	5003
2012-12-31	-1.458177e+06	551923.040048	9197	4802
2013-03-31	-1.368931e+06	518159.721947	8730	4442

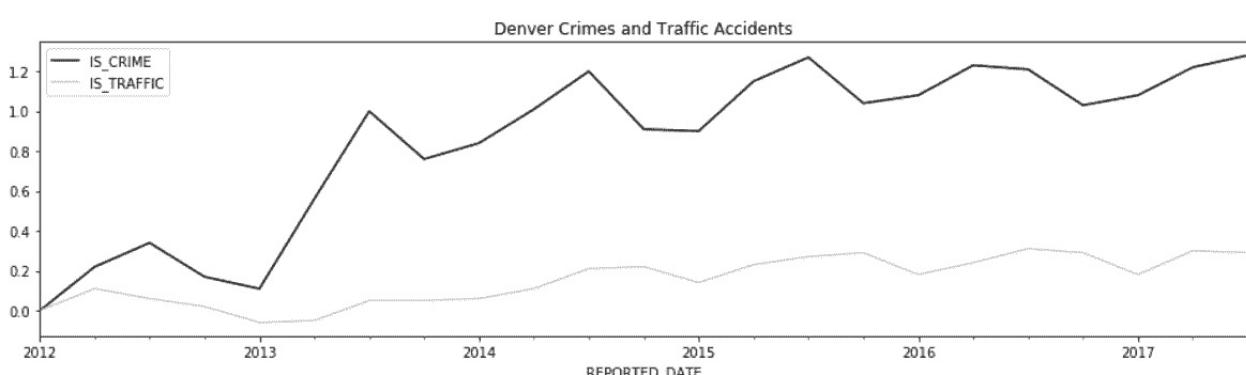
```
# 如果想用5月1日作为季度开始，可以使用别名QS-MAR
In[102]: crime_sort.resample('QS-MAR')[['IS_CRIME', 'IS_TRAFFIC']].sum().head()
Out[102]:
```

REPORTED_DATE	IS_CRIME	IS_TRAFFIC
2011-12-01	5013	3198
2012-03-01	9260	4954
2012-06-01	10524	5190
2012-09-01	9450	4777
2012-12-01	9003	4652

更多

```
# 画出犯罪和交通事故的增长率，通过除以第一行数据
In[103]: crime_begin = crime_quarterly.iloc[0]
          crime_begin
Out[103]: IS_CRIME      7882
           IS_TRAFFIC    4726
           Name: 2012-03-31 00:00:00, dtype: int64

In[104]: crime_quarterly.div(crime_begin) \
          .sub(1) \
          .round(2) \
          .plot(**plot_kwargs)
Out[104]: <matplotlib.axes._subplots.AxesSubplot at 0x1158850b8>
```



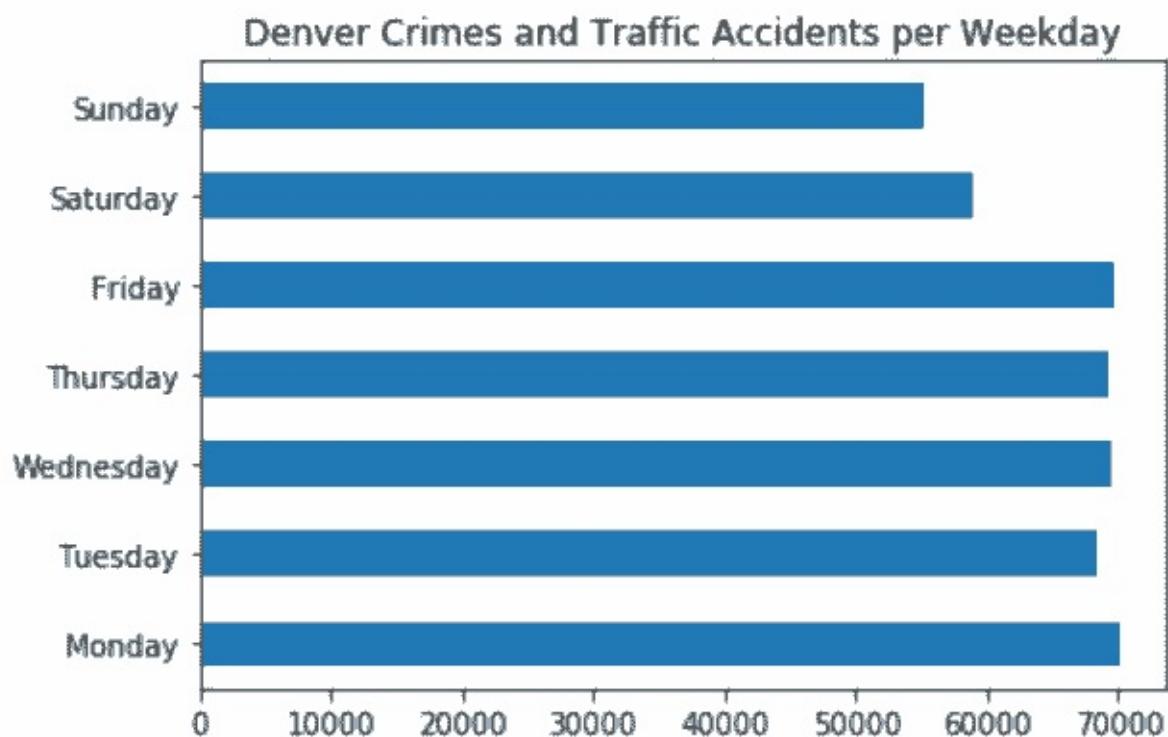
6. 按工作日和年测量犯罪

```
# 读取crime数据集，将REPORTED_DATE作为一列
In[105]: crime = pd.read_hdf('data/crime.h5', 'crime')
          crime.head()
Out[105]:
```

	OFFENSE_TYPE_ID	OFFENSE_CATEGORY_ID	REPORTED_DATE	GEO_LON	GEO_LAT	NEIGHBORHOOD_ID	IS_CRIME	IS_
0	traffic-accident-dui-duid	traffic-accident	2014-06-29 02:01:00	-105.000149	39.745753	cbd	0	
1	vehicular-eluding-no-chase	all-other-crimes	2014-06-29 01:54:00	-104.884660	39.738702	east-colfax	1	
2	disturbing-the-peace	public-disorder	2014-06-29 02:00:00	-105.020719	39.706674	athmar-park	1	
3	curfew	public-disorder	2014-06-29 02:18:00	-105.001552	39.769505	sunnyside	1	
4	aggravated-assault	aggravated-assault	2014-06-29 04:17:00	-105.018557	39.679229	college-view-south-platte	1	

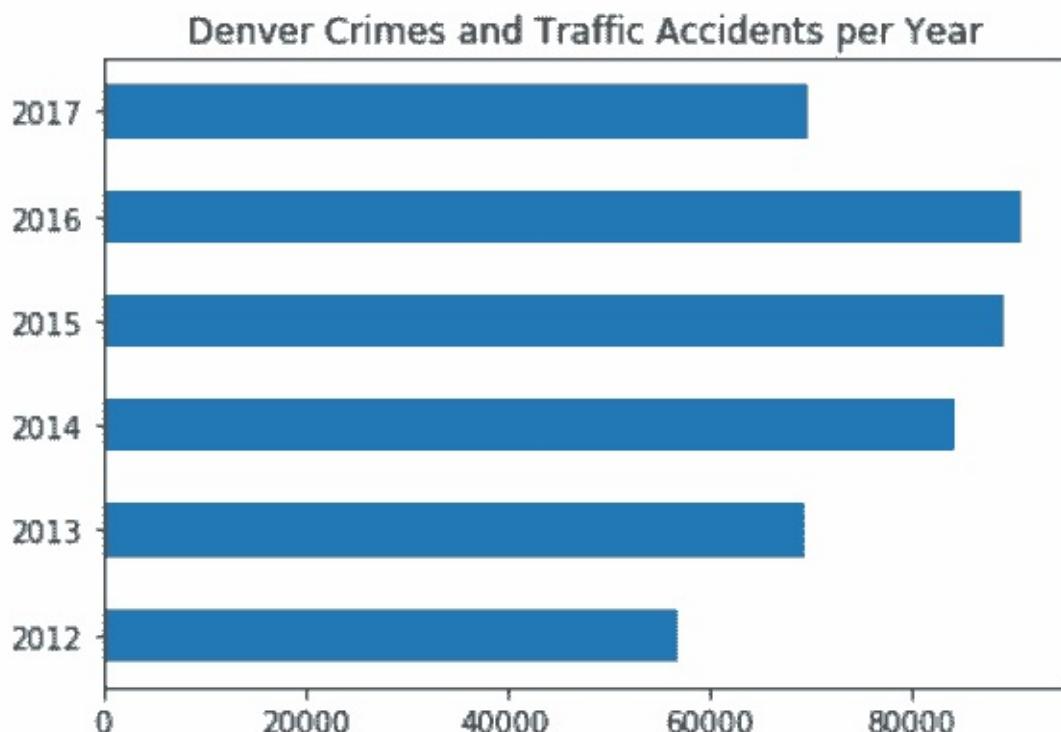
```
# 可以通过Timestamp的dt属性得到周几，然后统计
In[106]: wd_counts = crime['REPORTED_DATE'].dt.weekday_name.value_counts()
          wd_counts
Out[106]: Monday      70024
          Friday       69621
          Wednesday    69538
          Thursday     69287
          Tuesday      68394
          Saturday     58834
          Sunday       55213
          Name: REPORTED_DATE, dtype: int64
```

```
# 画一张水平柱状图
In[107]: days = ['Monday', 'Tuesday', 'Wednesday', 'Thursday',
                 'Friday', 'Saturday', 'Sunday']
          title = 'Denver Crimes and Traffic Accidents per Weekday'
          wd_counts.reindex(days).plot(kind='barh', title=title)
Out[107]: <matplotlib.axes._subplots.AxesSubplot at 0x117e39e48>
```



```
# 相似的，也可以画出每年的水平柱状图
In[108]: title = 'Denver Crimes and Traffic Accidents per Year'

        crime['REPORTED_DATE'].dt.year.value_counts() \
            .sort_index() \
            .plot(kind='barh', title
=title)
Out[108]: <matplotlib.axes._subplots.AxesSubplot at 0x11b1c6d68>
```



```
# 将年和星期按两列分组聚合
```

```
In[109]: weekday = crime['REPORTED_DATE'].dt.weekday_name
year = crime['REPORTED_DATE'].dt.year

crime_wd_y = crime.groupby([year, weekday]).size()
crime_wd_y.head(10)
```

```
Out[109]: REPORTED_DATE  REPORTED_DATE
```

2012	Friday	8549
	Monday	8786
	Saturday	7442
	Sunday	7189
	Thursday	8440
	Tuesday	8191
	Wednesday	8440
2013	Friday	10380
	Monday	10627
	Saturday	8875

```
dtype: int64
```

```
# 重命名索引名，然后对Weekday做unstack
```

```
In[110]: crime_table = crime_wd_y.rename_axis(['Year', 'Weekday']).unstack('Weekday')
crime_table
```

```
Out[110]:
```

Weekday	Friday	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
Year							
2012	8549	8786	7442	7189	8440	8191	8440
2013	10380	10627	8875	8444	10431	10416	10354
2014	12683	12813	10950	10278	12309	12440	12948
2015	13273	13452	11586	10624	13512	13381	13320
2016	14059	13708	11467	10554	14050	13338	13900
2017	10677	10638	8514	8124	10545	10628	10576

```
# 找到数据中2017年的最后一天
```

```
In[111]: criteria = crime['REPORTED_DATE'].dt.year == 2017
         crime.loc[criteria, 'REPORTED_DATE'].dt.dayofyear.max()
)
Out[111]: 272
```

```
# 计算这272天的平均犯罪率
```

```
In[112]: round(272 / 365, 3)
Out[112]: 0.745
```

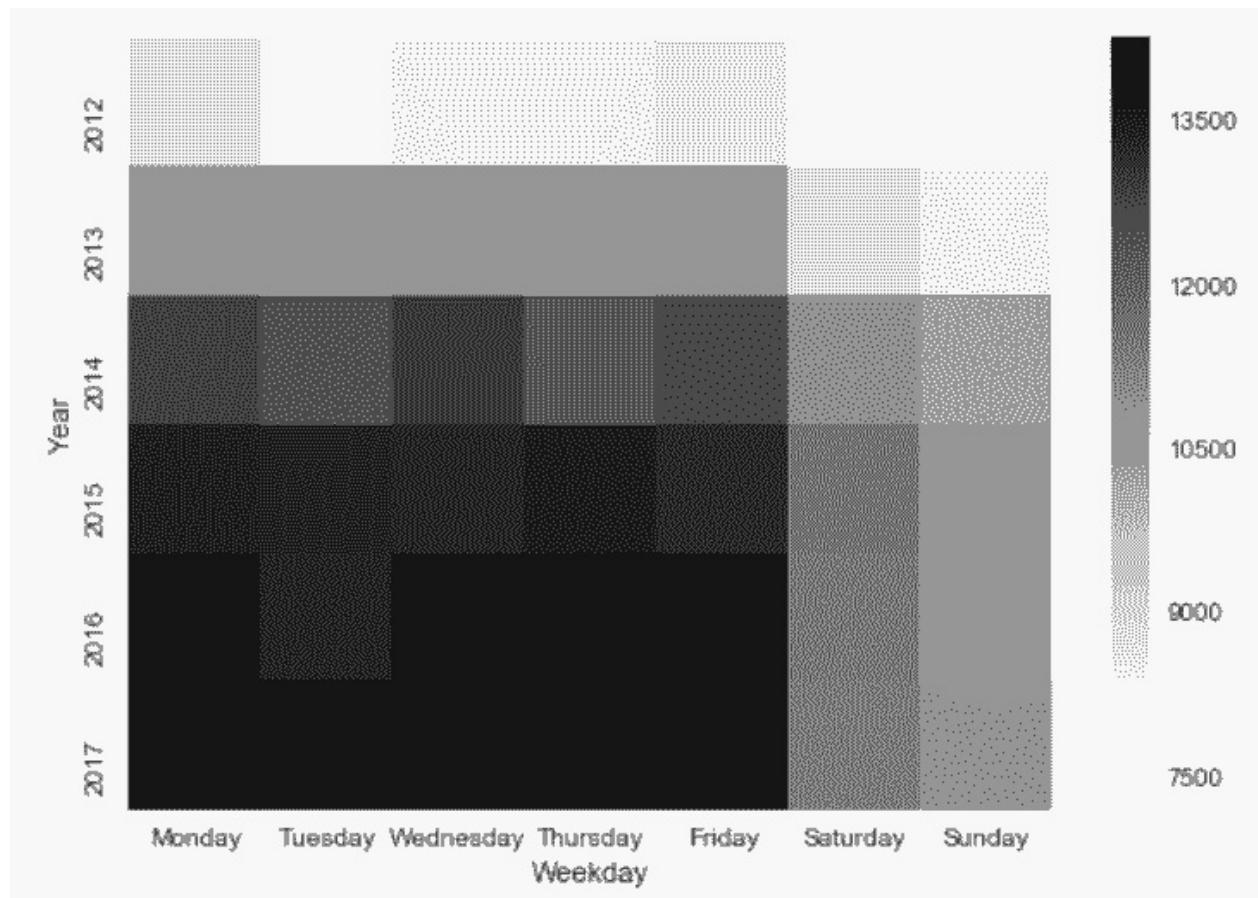
```
In[113]: crime_pct = crime['REPORTED_DATE'].dt.dayofyear.le(272)
          .groupby(year) \
          .mean() \
          .round(3)
          crime_pct
Out[113]: REPORTED_DATE
          2012    0.748
          2013    0.725
          2014    0.751
          2015    0.748
          2016    0.752
          2017    1.000
Name: REPORTED_DATE, dtype: float64
```

```
In[114]: crime_pct.loc[2012:2016].median()
Out[114]: 0.748
```

```
# 更新2017年的数据，并将星期排序
In[115]: crime_table.loc[2017] = crime_table.loc[2017].div(.748)
          .astype('int')
          crime_table = crime_table.reindex(columns=days)
          crime_table
Out[115]:
```

Weekday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Year							
2012	8786	8191	8440	8440	8549	7442	7189
2013	10627	10416	10354	10431	10380	8875	8444
2014	12813	12440	12948	12309	12683	10950	10278
2015	13452	13381	13320	13512	13273	11586	10624
2016	13708	13338	13900	14050	14059	11467	10554
2017	14221	14208	14139	14097	14274	11382	10860

```
# 用seaborn画热力图
In[116]: import seaborn as sns
          sns.heatmap(crime_table, cmap='Greys')
Out[116]: <matplotlib.axes._subplots.AxesSubplot at 0x117a37ba8>
```



```
# 犯罪貌似每年都在增加，但这个数据没有考虑每年的新增人口。
```

```
# 读取丹佛市人口denver_pop数据集
```

```
In[117]: denver_pop = pd.read_csv('data/denver_pop.csv', index_col='Year')
denver_pop
```

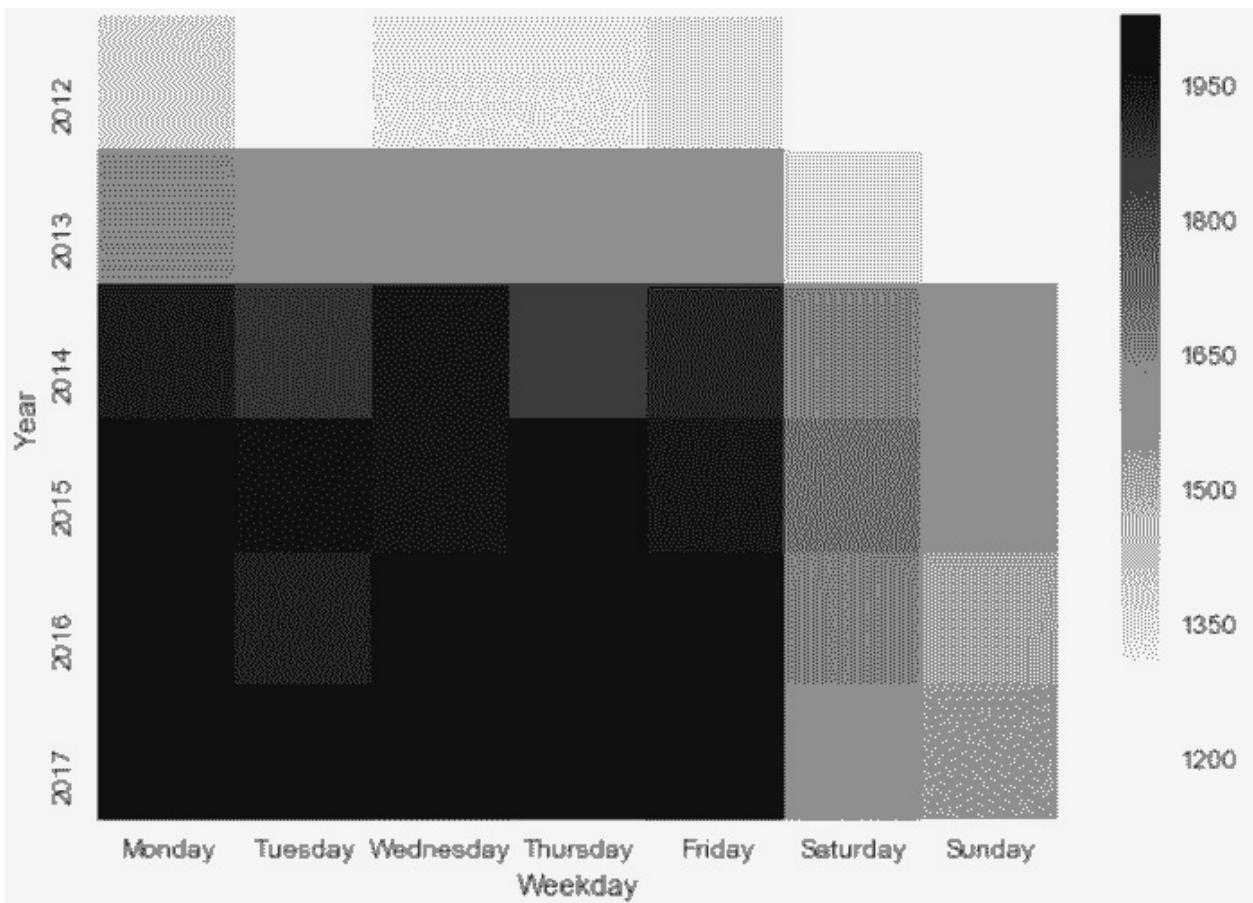
```
Out[117]:
```

Population	
Year	
2017	705000
2016	693000
2015	680000
2014	662000
2013	647000
2012	634000

```
# 计算每10万人的犯罪率
In[118]: den_100k = denver_pop.div(100000).squeeze()
          crime_table2 = crime_table.div(den_100k, axis='index')
          .astype('int')
          crime_table2
Out[118]:
```

Weekday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
Year							
2012	1385	1291	1331	1331	1348	1173	1133
2013	1642	1609	1600	1612	1604	1371	1305
2014	1935	1879	1955	1859	1915	1654	1552
2015	1978	1967	1958	1987	1951	1703	1562
2016	1978	1924	2005	2027	2028	1654	1522
2017	2017	2015	2005	1999	2024	1614	1540

```
# 再画一张热力图
In[119]: sns.heatmap(crime_table2, cmap='Greys')
Out[119]: <matplotlib.axes._subplots.AxesSubplot at 0x1203024e0>
```



原理

```
# loc接收一个排好序的列表，也可以实现reindex同样的功能
```

```
In[120]: wd_counts.loc[days]
Out[120]: Monday      70024
          Tuesday     68394
          Wednesday   69538
          Thursday    69287
          Friday      69621
          Saturday    58834
          Sunday      55213
Name: REPORTED_DATE, dtype: int64
```

```
# DataFrame和Series相除，会使用DataFrame的列和Series的行索引对齐
```

```
In[121]: crime_table / den_100k
/Users/Ted/anaconda/lib/python3.6/site-packages/pandas/core/indexes/base.py:3033: RuntimeWarning: '<' not supported between instances of 'str' and 'int', sort order is undefined for incomparable objects
      return this.join(other, how=how, return_indexers=return_indexers)
Out[121]:
```

	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday	2017	2016	2015	2014	2013	2012
Year													
2012	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2013	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2014	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2015	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

更多

```
# 将之前的操作打包成一个函数，并且可以根据犯罪类型筛选数据
In[122]: ADJ_2017 = .748

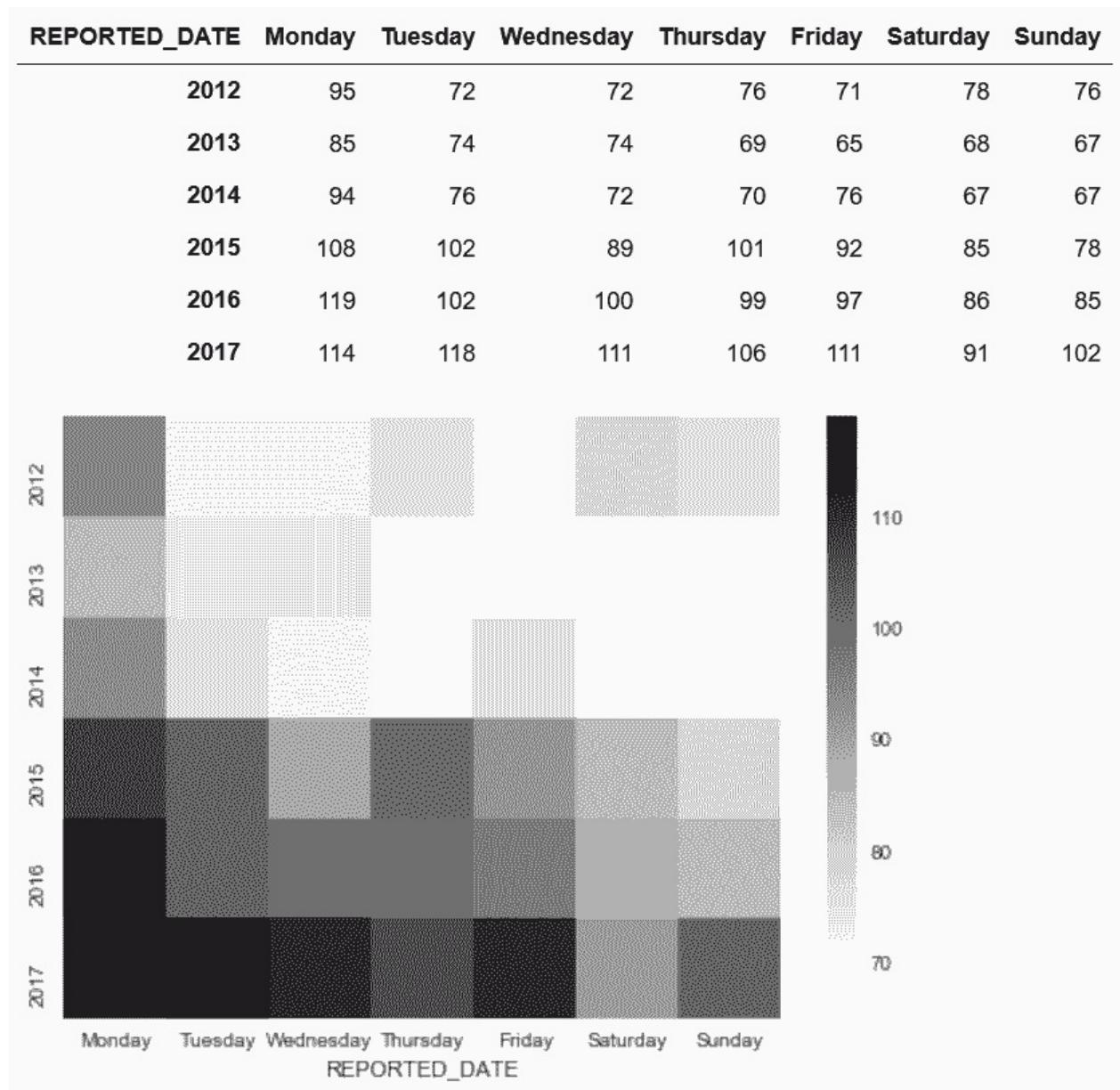
def count_crime(df, offense_cat):
    df = df[df['OFFENSE_CATEGORY_ID'] == offense_cat]
    weekday = df['REPORTED_DATE'].dt.weekday_name
    year = df['REPORTED_DATE'].dt.year

    ct = df.groupby([year, weekday]).size().unstack()
    ct.loc[2017] = ct.loc[2017].div(ADJ_2017).astype('int')

    pop = pd.read_csv('data/denver_pop.csv', index_col='Year')
    pop = pop.squeeze().div(100000)

    ct = ct.div(pop, axis=0).astype('int')
    ct = ct.reindex(columns=days)
    sns.heatmap(ct, cmap='Greys')
    return ct

In[123]: count_crime(crime, 'auto-theft')
Out[123]:
```



7. 用带有DatetimeIndex的匿名函数做分组

```
# 读取crime数据集，行索引设为REPORTED_DATE，并排序
In[124]: crime_sort = pd.read_hdf('data/crime.h5', 'crime') \
           .set_index('REPORTED_DATE') \
           .sort_index()

# 输出DatetimeIndex的可用属性和方法
In[125]: common_attrs = set(dir(crime_sort.index)) & set(dir(pd.Timestamp))
          print([attr for attr in common_attrs if attr[0] != '_'])

['nanosecond', 'month', 'daysinmonth', 'year', 'is_quarter_start',
 'offset', 'tz_localize', 'second', 'is_leap_year', 'resolution',
 'freqstr', 'week', 'weekday', 'is_month_start', 'normalize',
 'to_julian_date', 'to_period', 'freq', 'tzinfo', 'weekday_name',
 'microsecond', 'days_in_month', 'date', 'is_quarter_end', 'tz',
 'to_datetime', 'tz_convert', 'weekofyear', 'time', 'hour', 'min',
 'max', 'floor', 'is_year_start', 'ceil', 'dayofweek', 'day',
 'quarter', 'dayofyear', 'round', 'strftime', 'is_month_end', 'min
ute', 'is_year_end', 'to_pydatetime']

# 用index找到星期名
In[126]: crime_sort.index.weekday_name.value_counts()
Out[126]: Monday      70024
          Friday     69621
          Wednesday   69538
          Thursday    69287
          Tuesday     68394
          Saturday    58834
          Sunday      55213
          Name: REPORTED_DATE, dtype: int64
```

```
# groupby可以接收函数作为参数。
# 用函数将行索引变为周几，然后按照犯罪和交通事故统计
In[127]: crime_sort.groupby(lambda x: x.weekday_name)[['IS_CRIME',
          'IS_TRAFFIC']].sum()
Out[127]:
```

	IS_CRIME	IS_TRAFFIC
Friday	48833	20814
Monday	52158	17895
Saturday	43363	15516
Sunday	42315	12968
Thursday	49470	19845
Tuesday	49658	18755
Wednesday	50054	19508

```
# 可以用函数列表，用天的小时时间和年做分组，然后对表做重构型
```

```
In[128]: funcs = [lambda x: x.round('2h').hour, lambda x: x.year]
cr_group = crime_sort.groupby(funcs)[['IS_CRIME', 'IS_TRAFFIC']).sum()
cr_final = cr_group.unstack()
cr_final.style.highlight_max(color='lightgrey')
```

```
Out[128]:
```

	IS_CRIME							IS_TRAFFIC						
	2012	2013	2014	2015	2016	2017	2012	2013	2014	2015	2016	2017		
0	2422	4040	5649	5649	5377	3811	919	792	978	1136	980	782		
2	1888	3214	4245	4050	4091	3041	718	652	779	773	718	537		
4	1472	2181	2956	2959	3044	2255	399	378	424	471	464	313		
6	1067	1365	1750	2167	2108	1567	411	399	479	494	593	462		
8	2998	3445	3727	4161	4488	3251	1957	1955	2210	2331	2372	1828		
10	4305	5035	5658	6205	6218	4993	1979	1901	2139	2320	2303	1873		
12	4496	5524	6434	6841	7226	5463	2200	2138	2379	2631	2760	1986		
14	4266	5698	6708	7218	6896	5396	2241	2245	2630	2840	2763	1990		
16	4113	5889	7351	7643	7926	6338	2714	2562	3002	3160	3527	2784		
18	3660	5094	6586	7015	7407	6157	3118	2704	3217	3412	3608	2718		
20	3521	4895	6130	6360	6963	5272	1787	1806	1994	2071	2184	1491		

更多

```
# xs方法可以从任意索引层选出一个唯一值
In[129]: cr_final.xs('IS_TRAFFIC', axis='columns', level=0).head()
Out[129]:
```

	2012	2013	2014	2015	2016	2017
0	919	792	978	1136	980	782
2	718	652	779	773	718	537
4	399	378	424	471	464	313
6	411	399	479	494	593	462
8	1957	1955	2210	2331	2372	1828

```
# 用xs只从2016年选择数据，层级是1
In[130]: cr_final.xs(2016, axis='columns', level=1).head()
Out[130]:
```

	IS_CRIME	IS_TRAFFIC
0	5377	980
2	4091	718
4	3044	464
6	2108	593
8	4488	2372

8. 用时间戳和另一列分组

```
# 读取employee数据集，用HIRE_DATE列创造一个DatetimeIndex
In[131]: employee = pd.read_csv('data/employee.csv',
                                 parse_dates=['JOB_DATE', 'HIRE_DATE'],
                                 index_col='HIRE_DATE')
employee.head()
Out[131]:
```

	UNIQUE_ID	POSITION_TITLE	DEPARTMENT	BASE_SALARY	RACE	EMPLOYMENT_TYPE	GENDER
HIRE_DATE							
2006-06-12	0	ASSISTANT DIRECTOR (EX LVL)	Municipal Courts Department	121862.0	Hispanic/Latino	Full Time	Female
2000-07-19	1	LIBRARY ASSISTANT	Library	26125.0	Hispanic/Latino	Full Time	Female
2015-02-03	2	POLICE OFFICER	Houston Police Department-HPD	45279.0	White	Full Time	Male
1982-02-08	3	ENGINEER/OPERATOR	Houston Fire Department (HFD)	63166.0	White	Full Time	Male
1980-06-10	4	ELECTRICIAN	General Services	56347.0	White	Full Time	Male

```
# 对性别做分组，查看二者的工资
In[132]: employee.groupby('GENDER')['BASE_SALARY'].mean().round(-2)
Out[132]: GENDER
Female    52200.0
Male      57400.0
Name: BASE_SALARY, dtype: float64
```

```
# 根据聘用日期，每10年分一组，查看工资情况
In[133]: employee.resample('10AS')[['BASE_SALARY']].mean().round(-2)
Out[133]: HIRE_DATE
1958-01-01    81200.0
1968-01-01   106500.0
1978-01-01    69600.0
1988-01-01    62300.0
1998-01-01    58200.0
2008-01-01    47200.0
Freq: 10AS-JAN, Name: BASE_SALARY, dtype: float64
```

```
# 如果要按性别和五年分组，可以在groupby后面调用resample
In[134]: sal_avg = employee.groupby('GENDER').resample('10AS')[  
'BASE_SALARY'].mean().round(-2)
          sal_avg
Out[134]: GENDER  HIRE_DATE
           Female  1975-01-01    51600.0
                      1985-01-01    57600.0
                      1995-01-01    55500.0
                      2005-01-01    51700.0
                      2015-01-01    38600.0
           Male    1958-01-01    81200.0
                      1968-01-01   106500.0
                      1978-01-01    72300.0
                      1988-01-01    64600.0
                      1998-01-01    59700.0
                      2008-01-01    47200.0
Name: BASE_SALARY, dtype: float64
```

```
# 对性别unstack
In[135]: sal_avg.unstack('GENDER')
Out[135]:
```

GENDER	Female	Male
HIRE_DATE		
1958-01-01	NaN	81200.0
1968-01-01	NaN	106500.0
1975-01-01	51600.0	NaN
1978-01-01	NaN	72300.0
1985-01-01	57600.0	NaN
1988-01-01	NaN	64600.0
1995-01-01	55500.0	NaN
1998-01-01	NaN	59700.0
2005-01-01	51700.0	NaN
2008-01-01	NaN	47200.0
2015-01-01	38600.0	NaN

```
# 上面数据的问题，是分组不恰当造成的。
# 第一名男性受聘于1958年
In[136]: employee[employee['GENDER'] == 'Male'].index.min()
Out[136]: Timestamp('1958-12-29 00:00:00')

# 第一名女性受聘于1975年
In[137]: employee[employee['GENDER'] == 'Female'].index.min()
Out[137]: Timestamp('1975-06-09 00:00:00')
```

```
# 为了解决前面的分组问题，必须将日期和性别同时分组
In[138]: sal_avg2 = employee.groupby(['GENDER', pd.Grouper(freq='10AS')])['BASE_SALARY'].mean().round(-2)
sal_avg2
Out[138]: GENDER   HIRE_DATE
Female    1968-01-01      NaN
           1978-01-01    57100.0
           1988-01-01    57100.0
           1998-01-01    54700.0
           2008-01-01    47300.0
Male      1958-01-01    81200.0
           1968-01-01   106500.0
           1978-01-01    72300.0
           1988-01-01    64600.0
           1998-01-01    59700.0
           2008-01-01    47200.0
Name: BASE_SALARY, dtype: float64
```

```
# 再对性别做unstack
In[139]: sal_final = sal_avg2.unstack('GENDER')
sal_final
Out[139]:
```

GENDER	Female	Male
HIRE_DATE		
1958-01-01	NaN	81200.0
1968-01-01	NaN	106500.0
1978-01-01	57100.0	72300.0
1988-01-01	57100.0	64600.0
1998-01-01	54700.0	59700.0
2008-01-01	47300.0	47200.0

原理

```
# groupby返回对象包含resample方法，但相反却不成立
In[140]: 'resample' in dir(employee.groupby('GENDER'))
Out[140]: True

In[141]: 'groupby' in dir(employee.resample('10AS'))
Out[141]: False
```

更多

```
# 通过加9，手工创造时间区间
In[142]: years = sal_final.index.year
          years_right = years + 9
          sal_final.index = years.astype(str) + '-' + years_right
          sal_final
Out[142]:
```

GENDER	Female	Male
1958-1967	NaN	81200.0
1968-1977	NaN	106500.0
1978-1987	57100.0	72300.0
1988-1997	57100.0	64600.0
1998-2007	54700.0	59700.0
2008-2017	47300.0	47200.0

也可以使用cut函数创造基于每名员工受聘年份的等宽间隔

```
In[143]: cuts = pd.cut(employee.index.year, bins=5, precision=0)
)
cuts.categories.values
Out[143]: array([Interval(1958.0, 1970.0, closed='right'),
                   Interval(1970.0, 1981.0, closed='right'),
                   Interval(1981.0, 1993.0, closed='right'),
                   Interval(1993.0, 2004.0, closed='right'),
                   Interval(2004.0, 2016.0, closed='right')], dtype
e=object)

In[144]: employee.groupby([cuts, 'GENDER'])['BASE_SALARY'].mean()
().unstack('GENDER').round(-2)
Out[144]:
```

GENDER	Female	Male
(1958.0, 1970.0]	NaN	85400.0
(1970.0, 1981.0]	54400.0	72700.0
(1981.0, 1993.0]	55700.0	69300.0
(1993.0, 2004.0]	56500.0	62300.0
(2004.0, 2016.0]	49100.0	49800.0

9. 用merge_asof找到上次低20%犯罪率

```
# 读取crime数据集，行索引设为REPORTED_DATE，并排序
In[145]: crime_sort = pd.read_hdf('data/crime.h5', 'crime') \
           .set_index('REPORTED_DATE') \
           .sort_index()
```

```
# 找到最后一个完整月
In[146]: crime_sort.index.max()
Out[146]: Timestamp('2017-09-29 06:16:00')
```

```
# 因为9月份的数据不完整，所以只取到8月份的
In[147]: crime_sort = crime_sort[:'2017-8']
           crime_sort.index.max()
Out[147]: Timestamp('2017-08-31 23:52:00')
```

```
# 统计每月的犯罪和交通事故数量
In[148]: all_data = crime_sort.groupby([pd.Grouper(freq='M'), 'OFFENSE_CATEGORY_ID']).size()
           all_data.head()
Out[148]: REPORTED_DATE  OFFENSE_CATEGORY_ID
          2012-01-31    aggravated-assault      113
                           all-other-crimes     124
                           arson                  5
                           auto-theft            275
                           burglary            343
dtype: int64
```

```
# 重新设置索引
In[149]: all_data = all_data.sort_values().reset_index(name='Total')
           all_data.head()
Out[149]:
```

	REPORTED_DATE	OFFENSE_CATEGORY_ID	Total
0	2014-12-31	murder	1
1	2013-01-31	arson	1
2	2016-05-31	murder	1
3	2012-12-31	murder	1
4	2016-12-31	murder	1

```
# 用当前月的统计数乘以0.8，生成一个新的目标列
In[150]: goal = all_data[all_data['REPORTED_DATE'] == '2017-8-31'].reset_index(drop=True)
          goal['Total_Goal'] = goal['Total'].mul(.8).astype(int)
          goal.head()
Out[150]:
```

	REPORTED_DATE	OFFENSE_CATEGORY_ID	Total	Total_Goal
0	2017-08-31	murder	7	5
1	2017-08-31	arson	7	5
2	2017-08-31	sexual-assault	57	45
3	2017-08-31	robbery	108	86
4	2017-08-31	white-collar-crime	138	110

```
# 用merge_asof函数，找到上次每个犯罪类别低于目标值的月份
In[151]: pd.merge_asof(goal, all_data, left_on='Total_Goal', right_on='Total',
                      by='OFFENSE_CATEGORY_ID', suffixes=('_Current', '_Last'))
Out[151]:
```

REPORTED_DATE_Current	OFFENSE_CATEGORY_ID	Total_Current	Total_Goal	REPORTED_DATE_Last	Total_Last	
0	2017-08-31	murder	7	5	2017-01-31	5
1	2017-08-31	arson	7	5	2012-01-31	5
2	2017-08-31	sexual-assault	57	45	2013-01-31	45
3	2017-08-31	robbery	108	86	2015-03-31	86
4	2017-08-31	white-collar-crime	138	110	2016-10-31	110
5	2017-08-31	aggravated-assault	195	156	2016-05-31	154
6	2017-08-31	other-crimes-against-persons	376	300	2014-04-30	285
7	2017-08-31	burglary	432	345	2012-01-31	343
8	2017-08-31	auto-theft	599	479	2017-07-31	477
9	2017-08-31	drug-alcohol	636	508	2015-05-31	505
10	2017-08-31	theft-from-motor-vehicle	675	540	2015-03-31	535
11	2017-08-31	larceny	877	701	2015-01-31	697

更多

```
# 手动创建一个Periods  
In[152]: pd.Period(year=2012, month=5, day=17, hour=14, minute=20, freq='T')  
Out[152]: Period('2012-05-17 14:20', 'T')
```

```
# 具有DatetimeIndex的DataFrame有to_period方法，可以将Timestamps转换为Periods
In[153]: crime_sort.index.to_period('M')
Out[153]: PeriodIndex(['2012-01', '2012-01', '2012-01', '2012-01',
, '2012-01', '2012-01',
, '2012-01', '2012-01', '2012-01',
,
,
, '2017-08', '2017-08', '2017-08', '2017-08',
, '2017-08', '2017-08', '2017-08', '2017-08'],
],
dtype='period[M]', name='REPORTED_DATE',
length=453568, freq='M')

In[154]: ad_period = crime_sort.groupby([lambda x: x.to_period('M'),
,
'OFFENSE_CATEGORY_ID'])
.size()
ad_period = ad_period.sort_values() \
.reset_index(name='Total') \
.rename(columns={'level_0': 'REPOR
TED_DATE'})
ad_period.head()
Out[154]:
```

	REPORTED_DATE	OFFENSE_CATEGORY_ID	Total
0	2014-12	murder	1
1	2013-01	arson	1
2	2016-05	murder	1
3	2012-12	murder	1
4	2016-12	murder	1

```
# 判断ad_period的最后两列和之前的all_data是否相同
In[155]: cols = ['OFFENSE_CATEGORY_ID', 'Total']
          all_data[cols].equals(ad_period[cols])
Out[155]: True
```

```
# 用同样的方法，也可以重构正文中的最后两步
In[156]: aug_2018 = pd.Period('2017-8', freq='M')
          goal_period = ad_period[ad_period['REPORTED_DATE'] == aug_2018].reset_index(drop=True)
          goal_period['Total_Goal'] = goal_period['Total'].mul(.8).astype(int)

          pd.merge_asof(goal_period, ad_period, left_on='Total_Goal', right_on='Total',
                         by='OFFENSE_CATEGORY_ID', suffixes=( '_Current', '_Last')).head()
Out[156]:
```

	REPORTED_DATE_Current	OFFENSE_CATEGORY_ID	Total_Current	Total_Goal	REPORTED_DATE_Last	Total_Last
0	2017-08	murder	7	5	2017-01	5
1	2017-08	arson	7	5	2012-01	5
2	2017-08	sexual-assault	57	45	2013-01	45
3	2017-08	robbery	108	86	2015-03	86
4	2017-08	white-collar-crime	138	110	2016-10	110

第11章 用Matplotlib、Pandas、Seaborn进行可视化

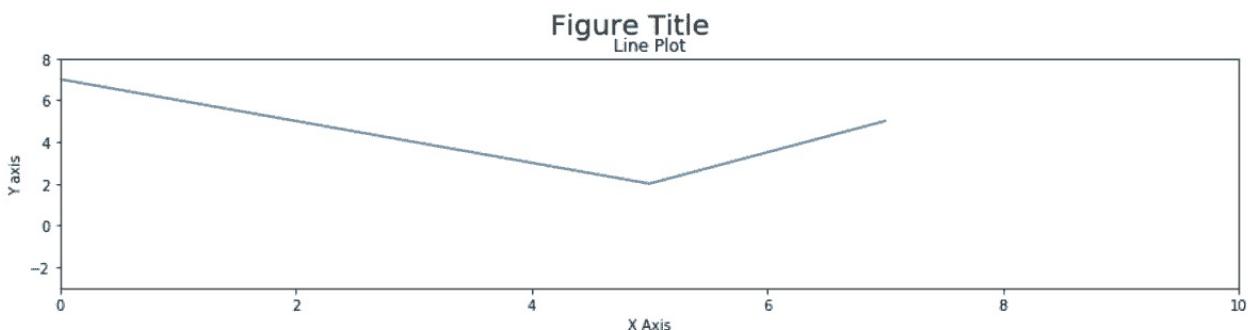
一章内容介绍三块内容，感觉哪个都没说清。

```
In[1]: import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
  
%matplotlib inline
```

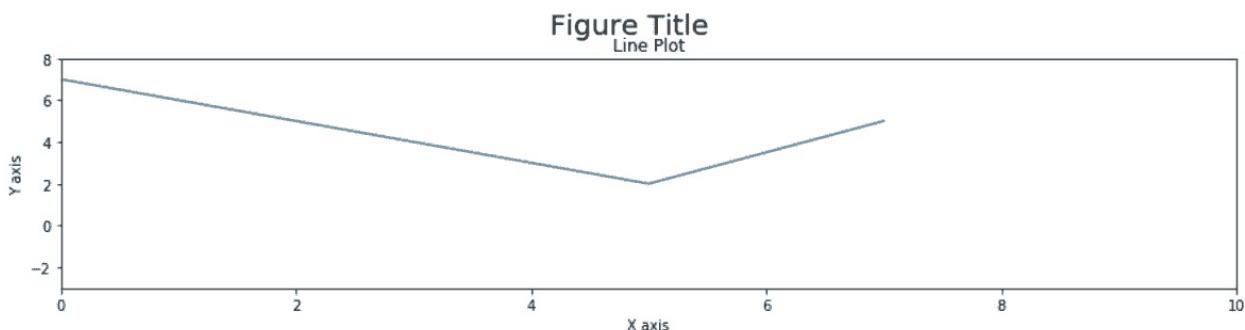
1. matplotlib入门

Matplotlib提供了两种方法来作图：状态接口和面向对象。

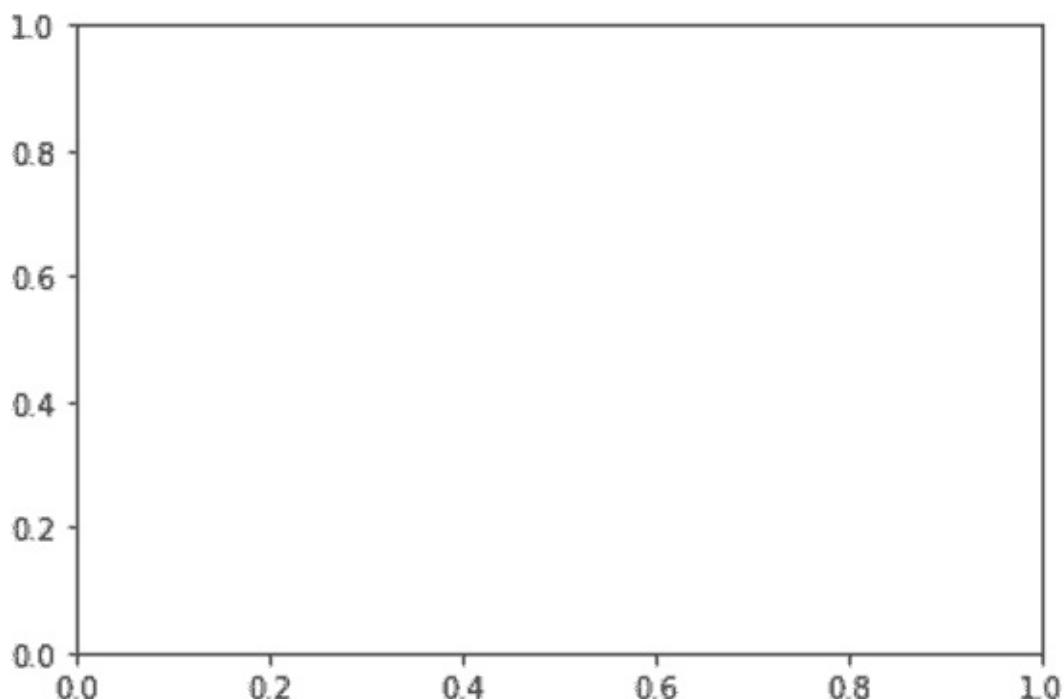
```
# 状态接口是通过pyplot模块来实现的，matplotlib会追踪绘图环境的当前状态  
# 这种方法适合快速画一些简单的图，但是对于多图和多轴会不方便  
In[2]: x = [-3, 5, 7]  
y = [10, 2, 5]  
  
plt.figure(figsize=(15, 3))  
plt.plot(x, y)  
plt.xlim(0, 10)  
plt.ylim(-3, 8)  
plt.xlabel('X Axis')  
plt.ylabel('Y axis')  
plt.title('Line Plot')  
plt.suptitle('Figure Title', size=20, y=1.03)  
Out[2]: Text(0.5, 1.03, 'Figure Title')
```



```
# 面向对象的方法更易懂，修改的是哪个对象非常清晰  
# 而且代码更加pythonic，与pandas的交互方式更相似  
In[3]: fig, ax = plt.subplots(figsize=(15, 3))  
        ax.plot(x, y)  
        ax.set_xlim(0, 10)  
        ax.set_ylim(-3, 8)  
        ax.set_xlabel('X axis')  
        ax.set_ylabel('Y axis')  
        ax.set_title('Line Plot')  
        fig.suptitle('Figure Title', size=20, y=1.03)  
Out[3]: Text(0.5, 1.03, 'Figure Title')
```



```
# 用subplots函数创建一个带有一个Axes的Figure  
In[4]: fig, ax = plt.subplots(nrows=1, ncols=1)
```

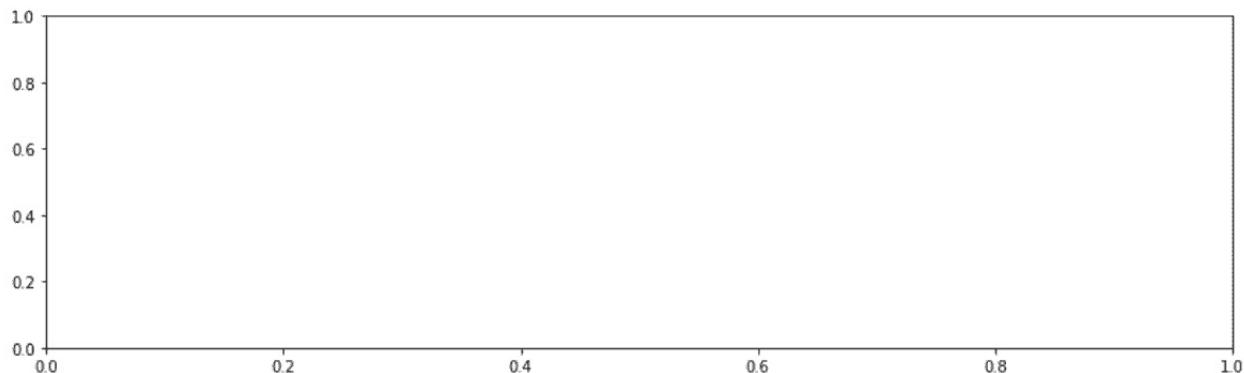


```
# 查看fig和ax的数据类型
In[5]: type(fig)
Out[5]: matplotlib.figure.Figure

In[6]: type(ax)
Out[6]: matplotlib.axes._subplots.AxesSubplot
```

```
# 查询Figure的大小，并放大
In[7]: fig.get_size_inches()
Out[7]: array([ 6.,  4.])

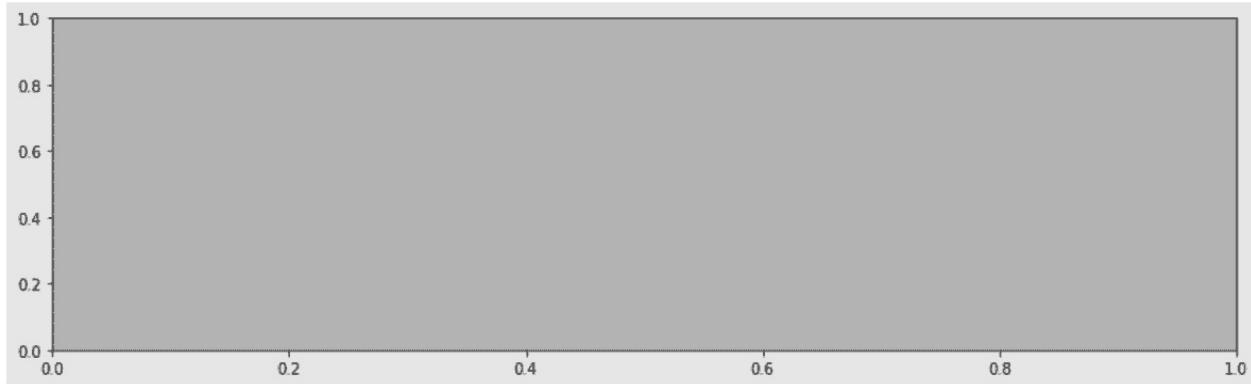
In[8]: fig.set_size_inches(14, 4)
fig
Out[8]:
```



```
# axes属性可以获取Figure的所有Axes
In[9]: fig.axes
Out[9]: [<matplotlib.axes._subplots.AxesSubplot at 0x1134202b0>]
```

```
# 判断Axes列表中的第一个元素和之前定义的ax是否相同
In[10]: fig.axes[0] is ax
Out[10]: True
```

```
# 用浮点数显示不同的灰度
In[11]: fig.set_facecolor('.9')
         ax.set_facecolor('.7')
         fig
Out[11]:
```



```
# 检查Axes的子元素，每个基本的图都有四个spine和两个axis
```

```
# spine是数据边界，即四条边
```

```
# x和y轴对象包含了更多的绘图对象，比如刻度、标签
```

```
In[12]: ax_children = ax.get_children()
ax_children
```

```
Out[12]: [<matplotlib.spines.Spine at 0x11145b358>,
<matplotlib.spines.Spine at 0x11145b0f0>,
<matplotlib.spines.Spine at 0x11145ae80>,
<matplotlib.spines.Spine at 0x11145ac50>,
<matplotlib.axis.XAxis at 0x11145aa90>,
<matplotlib.axis.YAxis at 0x110fa8d30>,
...]
```

```
# 通过spines属性直接访问spines
```

```
>>> spines = ax.spines
```

```
>>> spines
```

```
OrderedDict([('left', <matplotlib.spines.Spine at 0x11279e320>),
('right', <matplotlib.spines.Spine at 0x11279e0b8>)
,
('bottom', <matplotlib.spines.Spine at 0x11279e048>),
('top', <matplotlib.spines.Spine at 0x1127eb5c0>)])
```

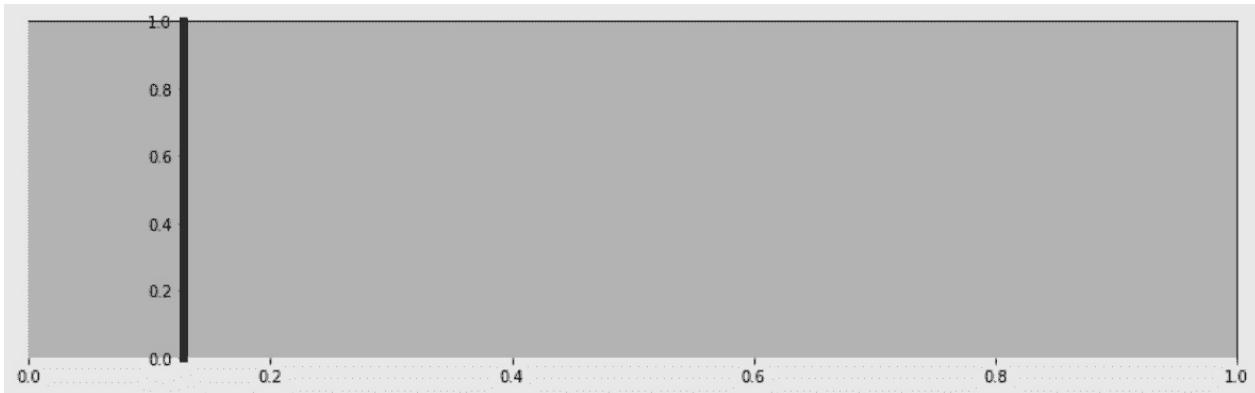
```
# 选中左边，改变它的位置和宽度，使底边不可见
```

```
In[13]: spine_left = spines['left']
```

```
spine_left.set_position(('outward', -100))
spine_left.set_linewidth(5)
```

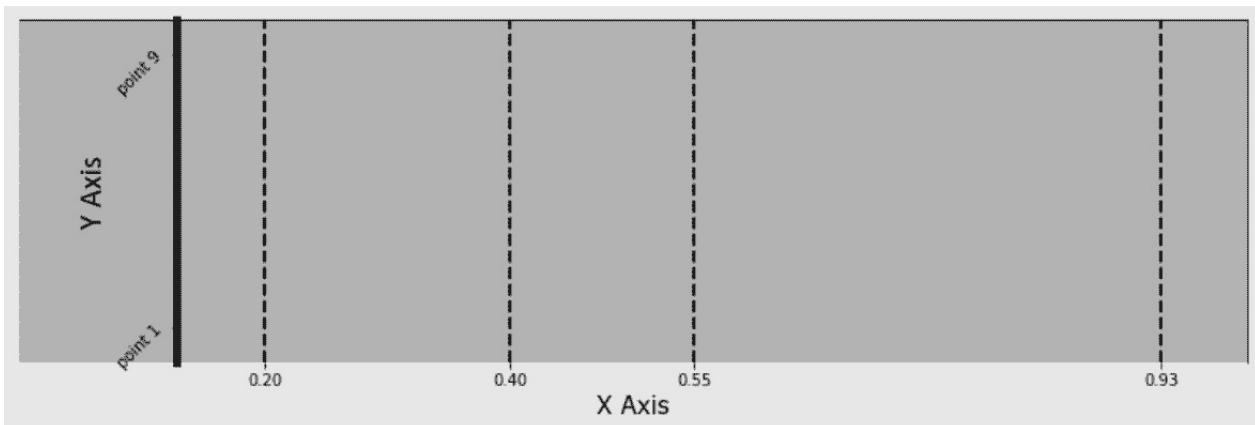
```
spine_bottom = spines['bottom']
spine_bottom.set_visible(False)
fig
```

```
Out[13]:
```



```
# 通过属性xaxis和yaxis可以修改属性，有些属性也可以通过Axes对象直接修改
In[14]: ax.xaxis.grid(True, which='major', linewidth=2, color='black', linestyle='--')
         ax.xaxis.set_ticks([.2, .4, .55, .93])
         ax.xaxis.set_label_text('X Axis', family='Verdana', fontsize=15)

         ax.set_ylabel('Y Axis', family='Calibri', fontsize=20)
         ax.set_yticks([.1, .9])
         ax.set_yticklabels(['point 1', 'point 9'], rotation=45)
fig
Out[14]:
```



原理

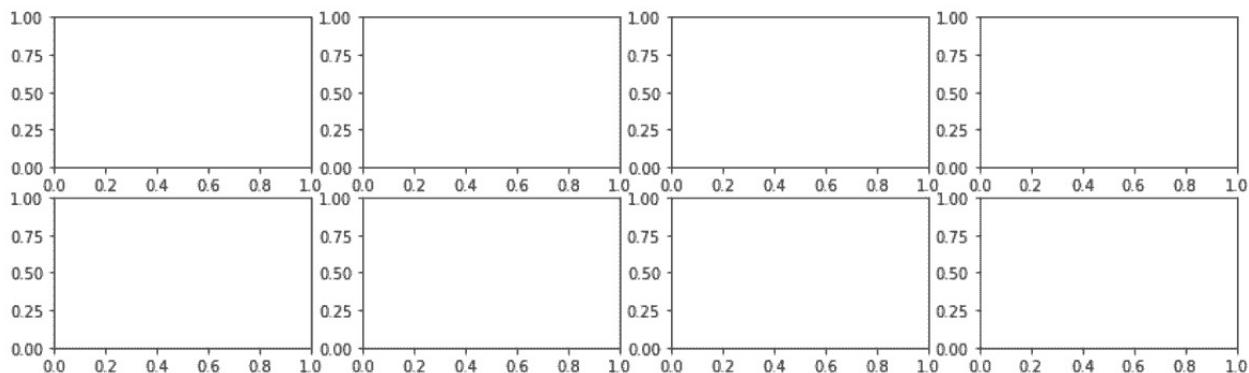
```
# plt.subplots函数返回的是一个元组
In[22]: plot_objects = plt.subplots()

In[23]: type(plot_objects)
Out[23]: tuple

In[24]: fig = plot_objects[0]
         ax = plot_objects[1]
```

```
# 如果创建了多个轴，则元组的第二个元素是一个包含所有轴的NumPy数组
```

```
In[25]: plot_objects = plt.subplots(2, 4, figsize=(14, 4))
```



```
In[26]: plot_objects[1]
```

```
Out[26]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x113eefaa0>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x113f7ccc0>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x11413ed68>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x114213e48>],
  [<matplotlib.axes._subplots.AxesSubplot object at 0x11424ce80>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x1142807b8>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x1142b8898>,
   <matplotlib.axes._subplots.AxesSubplot object at 0x1142f2898>]], dtype=object)
```

```
# 一些属性和与其等价的get方法
```

```
In[27]: fig.get_axes() == fig.axes
Out[27]: True
```

```
In[29]: fig.axes == fig.get_axes()
Out[29]: True
```

```
In[30]: ax.xaxis == ax.get_xaxis()
Out[30]: True
```

```
In[31]: ax.yaxis == ax.get_yaxis()
Out[31]: True
```

更多

```
# 查询xaxis的所有属性
In[15]: ax.xaxis.properties()
Out[15]:
{'agg_filter': None,
 'alpha': None,
 'animated': False,
 'children': [Text(0.5, 22.2, 'X Axis'),
  Text(1, 23.2, ''),
  <matplotlib.axis.XTick at 0x113371fd0>,
  <matplotlib.axis.XTick at 0x113514240>,
  <matplotlib.axis.XTick at 0x1136387b8>,
  <matplotlib.axis.XTick at 0x113638f60>],
 'clip_box': TransformedBbox(Bbox([[0.0, 0.0], [1.0, 1.0]]), CompositeGenericTransform(CompositeGenericTransform(BboxTransformTo(Bbox([[0.0, 0.0], [1.0, 1.0]]))), Affine2D(array([[ 1.,  0.,  0.],
   [ 0.,  1.,  0.],
   [ 0.,  0.,  1.]]))), BboxTransformTo(TransformedBbox(Bbox([[0.125, 0.125], [0.9, 0.88]]), BboxTransformTo(TransformedBbox(Bbox([[0.0, 0.0], [14.0, 4.0]]), Affine2D(array([[ 72.,  0.,
  0.],
  [ 0.,  72.,  0.],
  [ 0.,  0.,  1.]]))))))),
 'clip_on': True,
 'clip_path': None,
 'contains': None,
 'data_interval': array([ inf, -inf]),
 'figure': <matplotlib.figure.Figure at 0x11332abe0>,
 'gid': None,
 'gridlines': <a list of 4 Line2D gridline objects>,
 'label': Text(0.5, 22.2, 'X Axis'),
 'label_position': 'bottom',
 'label_text': 'X Axis',
 'major_formatter': <matplotlib.ticker.ScalarFormatter at 0x113543780>,
 'major_locator': <matplotlib.ticker.FixedLocator at 0x113648ba8>,
 'major_ticks': [<matplotlib.axis.XTick at 0x113371fd0>,
  <matplotlib.axis.XTick at 0x113514240>,
  <matplotlib.axis.XTick at 0x1136387b8>,
  <matplotlib.axis.XTick at 0x113638f60>],
 'majorticklabels': <a list of 4 Text major ticklabel objects>,
 'majorticklines': <a list of 8 Line2D ticklines objects>,
 'majorticklocs': array([ 0.2,  0.4,  0.55,  0.93]),
 'minor_formatter': <matplotlib.ticker.NullFormatter at 0x11341a518>,
 'minor_locator': <matplotlib.ticker.NullLocator at 0x113624198>,
 'minor_ticks': [],
 'minorticklabels': <a list of 0 Text minor ticklabel objects>,
 'minorticklines': <a list of 0 Line2D ticklines objects>,
 'minorticklocs': []},
```

```
'minpos': inf,
'offset_text': Text(1, 23.2, ''),
'path_effects': [],
'picker': None,
'pickradius': 15,
'rasterized': None,
'scale': 'linear',
'sketch_params': None,
'smart_bounds': False,
'snap': None,
'tick_padding': 3.5,
'tick_space': 26,
'ticklabels': <a list of 4 Text major ticklabel objects>,
'ticklines': <a list of 8 Line2D ticklines objects>,
'ticklocs': array([ 0.2 , 0.4 , 0.55, 0.93]),
'ticks_direction': array(['out', 'out', 'out', 'out'],
    dtype='<U3'),
'ticks_position': 'bottom',
'transform': IdentityTransform(),
'transformed_clip_path_and_affine': (None, None),
'units': None,
'url': None,
'vew_interval': array([ 0., 1.]),
'veisible': True,
'zorder': 1.5}
```

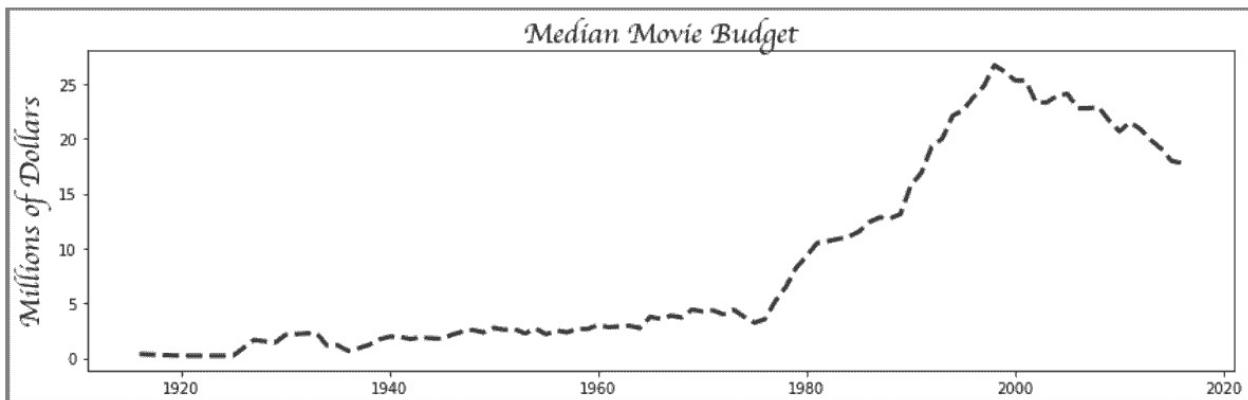
2. 用matplotlib做数据可视化

```
# 读取movie数据集，计算每年的预算中位数，再计算五年滚动均值以平滑数据
In[32]: movie = pd.read_csv('data/movie.csv')
med_budget = movie.groupby('title_year')['budget'].median() / 1e6
med_budget_roll = med_budget.rolling(5, min_periods=1).mean()
med_budget_roll.tail()
Out[32]: title_year
2012.0    20.893
2013.0    19.893
2014.0    19.100
2015.0    17.980
2016.0    17.780
Name: budget, dtype: float64
```

```
# 将数据变为NumPy数组
In[33]: years = med_budget_roll.index.values
years[-5:]
Out[33]: array([ 2012.,  2013.,  2014.,  2015.,  2016.])

In[34]: budget = med_budget_roll.values
budget[-5:]
Out[34]: array([ 20.893,  19.893,  19.1 ,  17.98 ,  17.78 ])
```

```
# plot方法可以用来画线图
In[35]: fig, ax = plt.subplots(figsize=(14,4), linewidth=5, edgecolor='.5')
        ax.plot(years, budget, linestyle='--', linewidth=3, color='.2', label='All Movies')
        text_kw_args=dict(fontsize=20, family='cursive')
        ax.set_title('Median Movie Budget', **text_kw_args)
        ax.set_ylabel('Millions of Dollars', **text_kw_args)
Out[35]: Text(0,0.5,'Millions of Dollars')
```



```
# 每年的电影产量
In[36]: movie_count = movie.groupby('title_year')['budget'].count()
          movie_count.tail()
Out[36]: title_year
          2012.0    191
          2013.0    208
          2014.0    221
          2015.0    192
          2016.0     86
Name: budget, dtype: int64
```

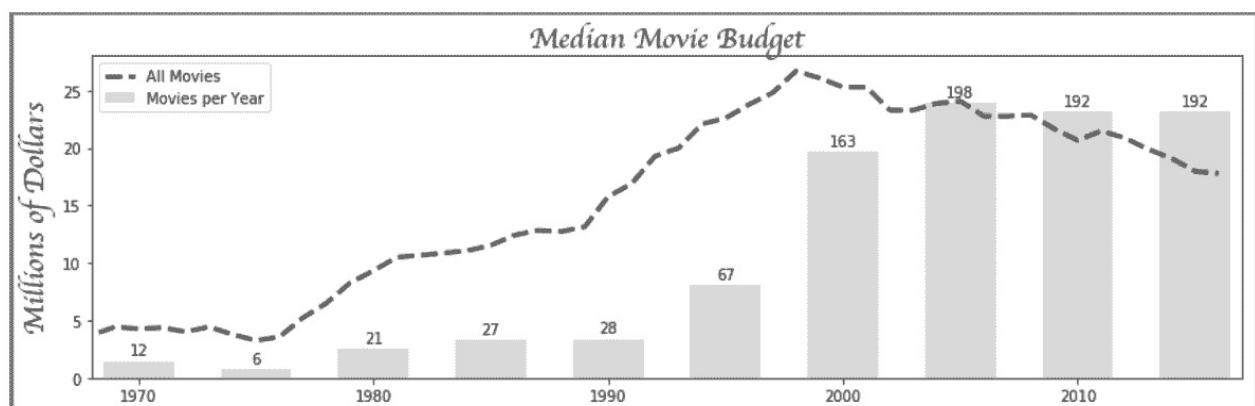
```
# 在前图的基础上，将每年的电影产量画成一个柱状图，因为大部分电影都近年的，所以将起始的年设为1970
```

```
In[37]: ct = movie_count.values
ct_norm = ct / ct.max() * budget.max()

fifth_year = (years % 5 == 0) & (years >= 1970)
years_5 = years[fifth_year]
ct_5 = ct[fifth_year]
ct_norm_5 = ct_norm[fifth_year]

ax.bar(years_5, ct_norm_5, 3, facecolor='0.5', alpha=.3,
label='Movies per Year')
ax.set_xlim(1968, 2017)
for x, y, v in zip(years_5, ct_norm_5, ct_5):
    ax.text(x, y + .5, str(v), ha='center')
ax.legend()
fig
```

```
Out[37]:
```



```
# 找到每年的前10部电影的五年滚动中位数
```

```
In[38]: top10 = movie.sort_values('budget', ascending=False) \
    .groupby('title_year')['budget'] \
    .apply(lambda x: x.iloc[:10].median() / 1e6
)
```

```
top10_roll = top10.rolling(5, min_periods=1).mean()
top10_roll.tail()
```

```
Out[38]: title_year
```

```
2012.0    192.9
2013.0    195.9
2014.0    191.7
2015.0    186.8
2016.0    189.1
```

```
Name: budget, dtype: float64
```

```
# 将上面的数据画到另一张子图中
In[39]: fig2, ax_array = plt.subplots(2, 1, figsize=(14,6), sharex=True)
         ax1 = ax_array[0]
         ax2 = ax_array[1]

         ax1.plot(years, budget, linestyle='--', linewidth=3, color='.2', label='All Movies')
         ax1.bar(years_5, ct_norm_5, 3, facecolor='.5', alpha=.3, label='Movies per Year')
         ax1.legend(loc='upper left')
         ax1.set_xlim(1968, 2017)
         plt.setp(ax1.get_xticklines(), visible=False)

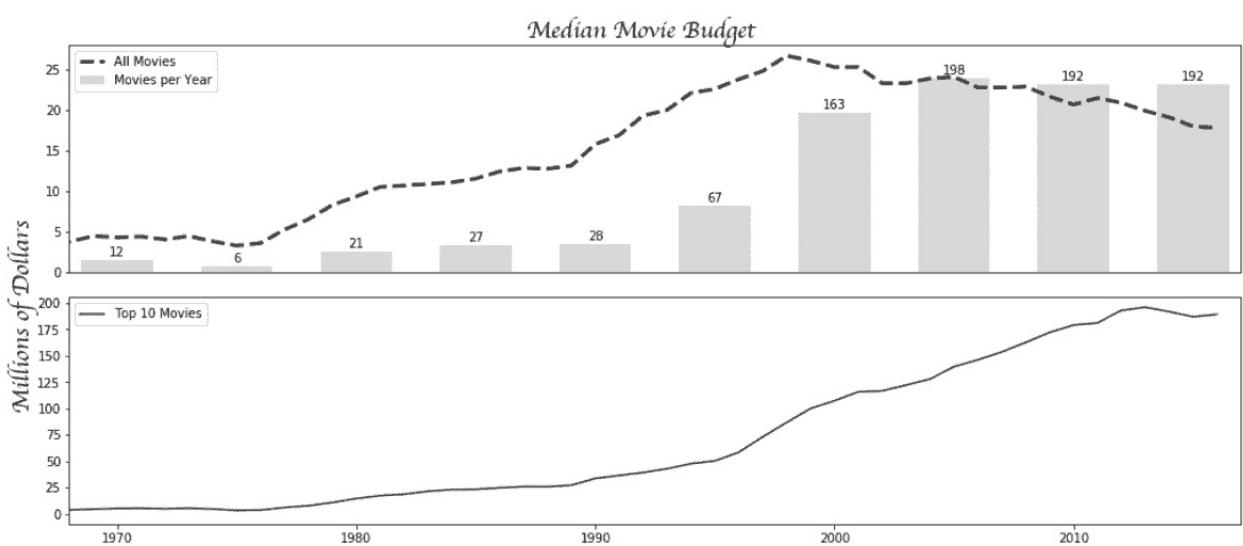
         for x, y, v in zip(years_5, ct_norm_5, ct_5):
             ax1.text(x, y + .5, str(v), ha='center')

         ax2.plot(years, top10_roll.values, color='.2', label='Top 10 Movies')
         ax2.legend(loc='upper left')

         fig2.tight_layout()
         fig2.suptitle('Median Movie Budget', y=1.02, **text_kwargs)
         fig2.text(0, .6, 'Millions of Dollars', rotation='vertical', ha='center', **text_kwargs)

import os
path = os.path.expanduser('~/Desktop/movie_budget.png')
fig2.savefig(path, bbox_inches='tight')

Out[39]:
```



原理

```
In[40]: med_budget_roll.tail()  
Out[40]: title_year  
2012.0    20.893  
2013.0    19.893  
2014.0    19.100  
2015.0    17.980  
2016.0    17.780  
Name: budget, dtype: float64
```

```
# 手动确认一下rolling方法  
In[41]: med_budget.loc[2012:2016].mean()  
Out[41]: 17.78  
  
In[42]: med_budget.loc[2011:2015].mean()  
Out[42]: 17.98  
  
In[43]: med_budget.loc[2010:2014].mean()  
Out[43]: 19.1
```

```
# 必须使用expanduser创建完整路径  
In[44]: os.path.expanduser('~/Desktop/movie_budget.png')  
Out[44]: '/Users/Ted/Desktop/movie_budget.png'
```

更多

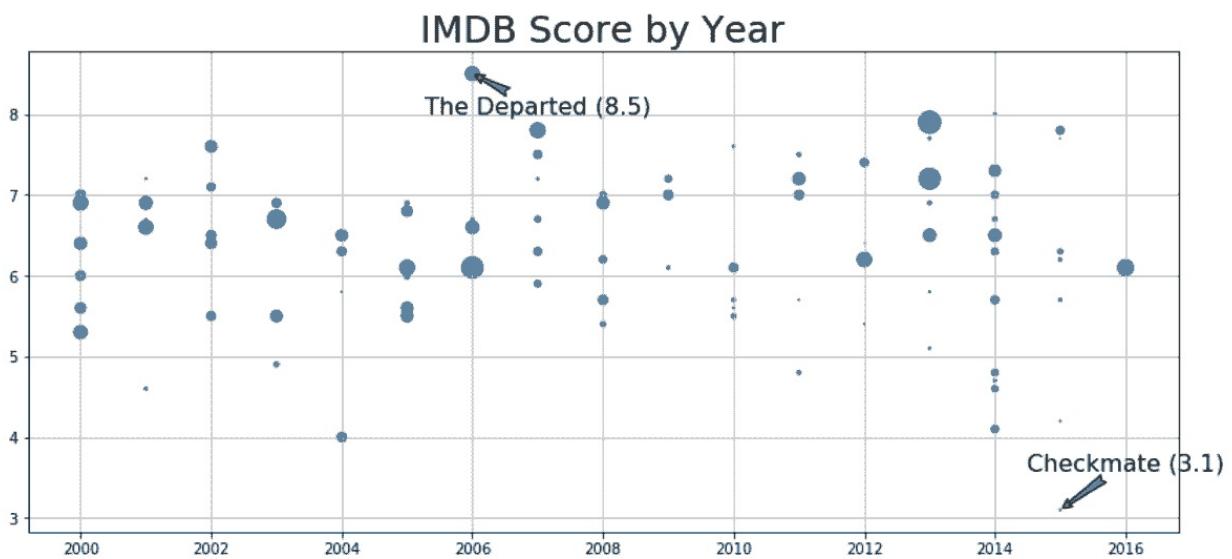
```
In[45]: cols = ['budget', 'title_year', 'imdb_score', 'movie_title']
m = movie[cols].dropna()

# m = movie[['budget', 'title_year', 'imdb_score', 'movie_title']].dropna()
m['budget2'] = m['budget'] / 1e6
np.random.seed(0)
movie_samp = m.query('title_year >= 2000').sample(100)

fig, ax = plt.subplots(figsize=(14, 6))
ax.scatter(x='title_year', y='imdb_score', s='budget2',
           data=movie_samp)

idx_min = movie_samp['imdb_score'].idxmin()
idx_max = movie_samp['imdb_score'].idxmax()
for idx, offset in zip([idx_min, idx_max], [.5, -.5]):
    year = movie_samp.loc[idx, 'title_year']
    score = movie_samp.loc[idx, 'imdb_score']
    title = movie_samp.loc[idx, 'movie_title']
    ax.annotate(xy=(year, score),
                xytext=(year + 1, score + offset),
                s=title + ' ({})'.format(score),
                ha='center',
                size=16,
                arrowprops=dict(arrowstyle="fancy"))
ax.set_title('IMDB Score by Year', size=25)
ax.grid(True)
```

Out[45]:

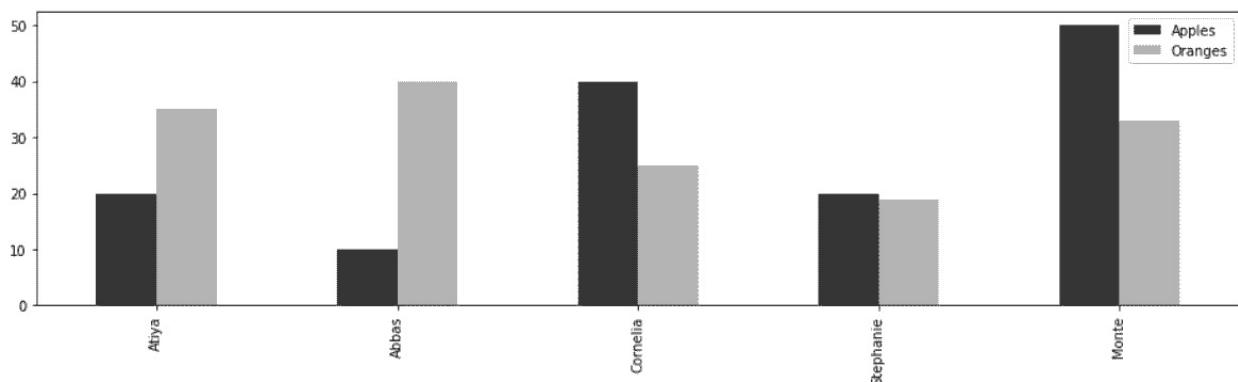


3. Pandas绘图基础

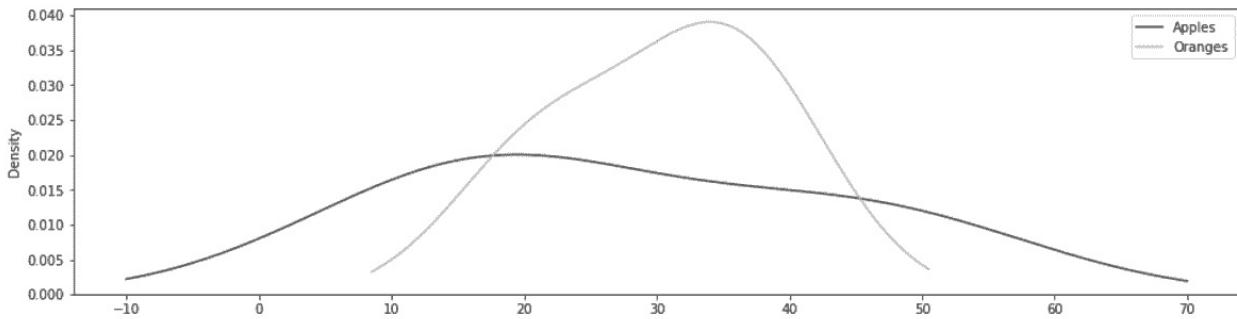
```
# 创建一个小DataFrame
In[46]: df = pd.DataFrame(index=['Atiya', 'Abbas', 'Cornelia',
'Stephanie', 'Monte'],
                           data={'Apples':[20, 10, 40, 20, 50],
                           'Oranges':[35, 40, 25, 19, 33]})
)
df
Out[46]:
```

	Apples	Oranges
Atiya	20	35
Abbas	10	40
Cornelia	40	25
Stephanie	20	19
Monte	50	33

```
# 画柱状图，使用行索引做x轴，列的值做高度，使用plot方法，参数kind设为bar
In[47]: color = ['.2', '.7']
df.plot(kind='bar', color=color, figsize=(16, 4))
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1143cae10>
```

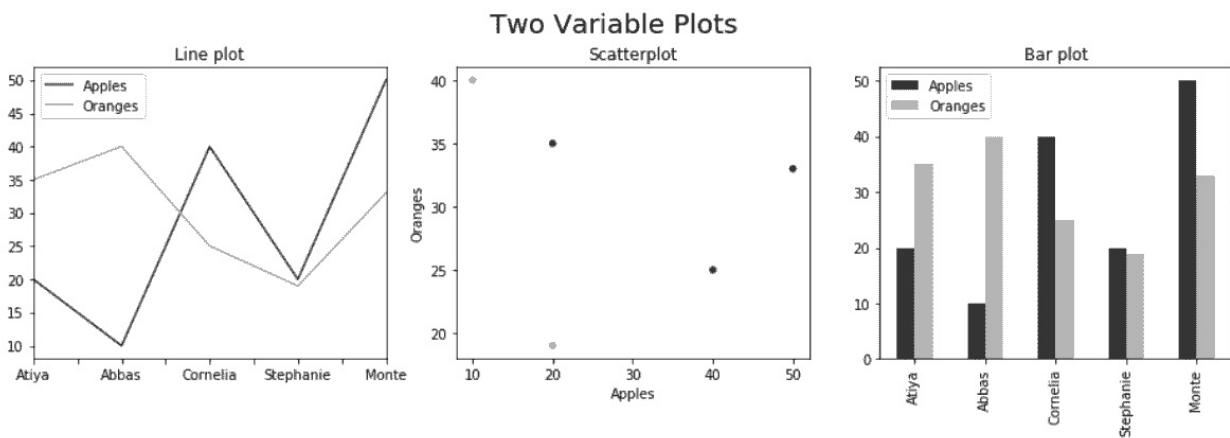


```
# KDE图忽略行索引，使用每列的值作为x轴，并计算y值得概率密度
In[48]: df.plot(kind='kde', color=color, figsize=(16, 4))
Out[48]: <matplotlib.axes._subplots.AxesSubplot at 0x11503ec50>
```



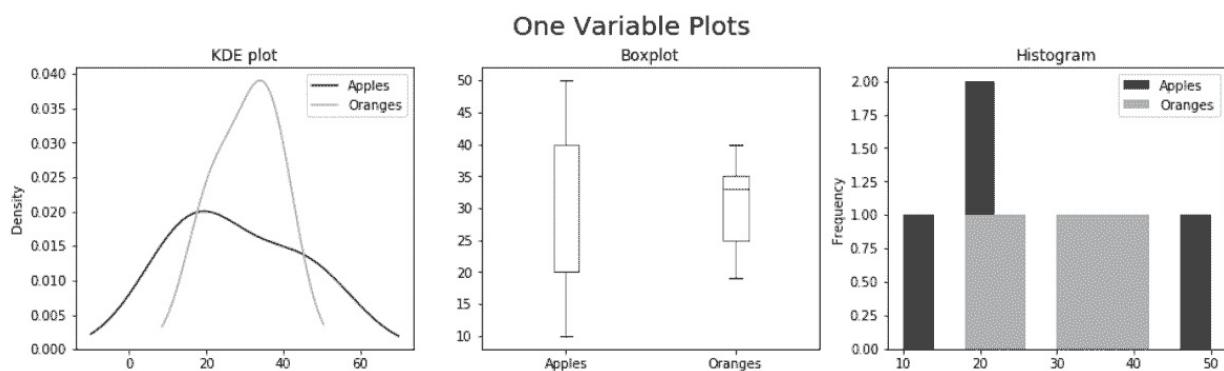
```
# 画三张双变量子图。散点图是唯一需要指定x和y轴的列，如果想在散点图中使用行索引，可以使用方法reset_index。
```

```
In[49]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
fig.suptitle('Two Variable Plots', size=20, y=1.02)
df.plot(kind='line', color=color, ax=ax1, title='Line plot')
df.plot(x='Apples', y='Oranges', kind='scatter', color=color,
        ax=ax2, title='Scatterplot')
df.plot(kind='bar', color=color, ax=ax3, title='Bar plot')
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0x119ccb5f8>
```



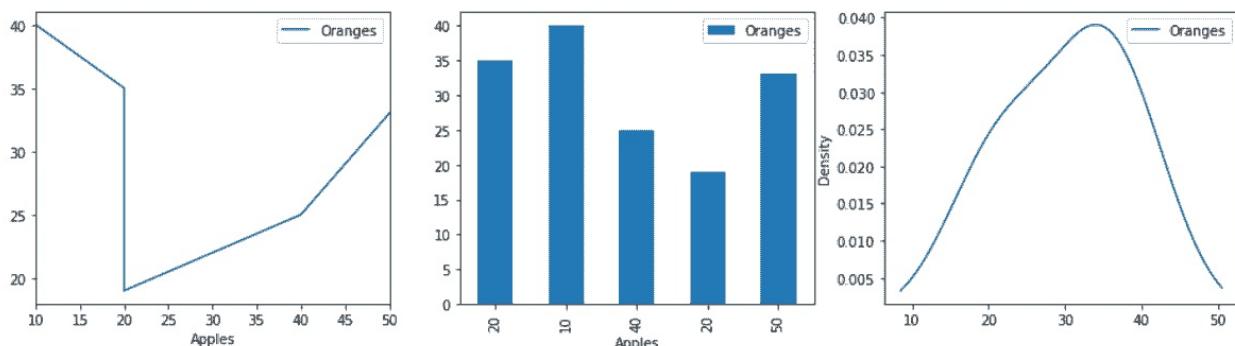
```
# 将单变量图也画在同一张图中
```

```
In[50]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
fig.suptitle('One Variable Plots', size=20, y=1.02)
df.plot(kind='kde', color=color, ax=ax1, title='KDE plot')
df.plot(kind='box', ax=ax2, title='Boxplot')
df.plot(kind='hist', color=color, ax=ax3, title='Histogram')
Out[50]: <matplotlib.axes._subplots.AxesSubplot at 0x119f475f8>
```



更多

```
# matplotlib允许手动指定x和y的值
In[51]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(16, 4))
          df.sort_values('Apples').plot(x='Apples', y='Oranges',
                                         kind='line', ax=ax1)
          df.plot(x='Apples', y='Oranges', kind='bar', ax=ax2)
          df.plot(x='Apples', kind='kde', ax=ax3)
Out[51]: <matplotlib.axes._subplots.AxesSubplot at 0x11a1bc438>
```



4. 可视化flights数据集

```
# 读取flights数据集
In[52]: flights = pd.read_csv('data/flights.csv')
          flights.head()
Out[52]:
```

MONTH	DAY	WEEKDAY	AIRLINE	ORG_AIR	DEST_AIR	SCHED_DEP	DEP_DELAY	AIR_TIME	DIST	SCHED_ARR	ARR_D
0	1	1	4	WN	LAX	SLC	1625	58.0	94.0	590	1905
1	1	1	4	UA	DEN	IAD	823	7.0	154.0	1452	1333
2	1	1	4	MQ	DFW	VPS	1305	36.0	85.0	641	1453
3	1	1	4	AA	DFW	DCA	1555	7.0	126.0	1192	1935
4	1	1	4	WN	LAX	MCI	1720	48.0	166.0	1363	2225

```
# 创建两列，表示延迟和准时
In[53]: flights['DELAYED'] = flights['ARR_DELAY'].ge(15).astype(int)
          cols = ['DIVERTED', 'CANCELLED', 'DELAYED']
          flights['ON_TIME'] = 1 - flights[cols].any(axis=1)

          cols.append('ON_TIME')
          status = flights[cols].sum()
          status
Out[53]: DIVERTED      137
          CANCELLED     881
          DELAYED     11685
          ON_TIME     45789
          dtype: int64
```

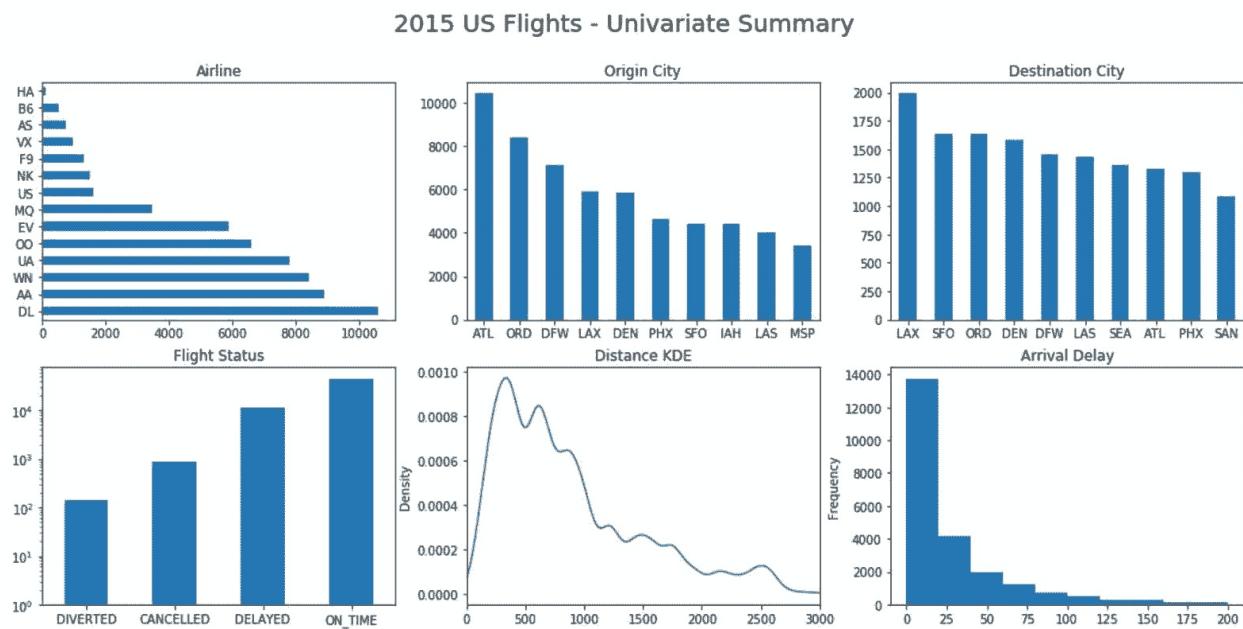
```
# 对类型值和连续值列作图
In[54]: fig, ax_array = plt.subplots(2, 3, figsize=(18,8))
          (ax1, ax2, ax3), (ax4, ax5, ax6) = ax_array
          fig.suptitle('2015 US Flights - Univariate Summary', size=20)

          ac = flights['AIRLINE'].value_counts()
          ac.plot(kind='barh', ax=ax1, title ='Airline')

          oc = flights['ORG_AIR'].value_counts()
          oc.plot(kind='bar', ax=ax2, rot=0, title='Origin City')

          dc = flights['DEST_AIR'].value_counts().head(10)
          dc.plot(kind='bar', ax=ax3, rot=0, title='Destination City')

          status.plot(kind='bar', ax=ax4, rot=0, log=True, title='Flight Status')
          flights['DIST'].plot(kind='kde', ax=ax5, xlim=(0, 3000),
          ,
                           title='Distance KDE')
          flights['ARR_DELAY'].plot(kind='hist', ax=ax6,
                                     title='Arrival Delay', range=(0,200))
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x11a67e3c8>
```



```
# 添加关于年的列，用起飞时间得到小时和分钟
```

```
In[55]: hour = flights['SCHED_DEP'] // 100
        minute = flights['SCHED_DEP'] % 100
        df_date = flights[['MONTH', 'DAY']].assign(YEAR=2015, HOUR=hour, MINUTE=minute)
        df_date.head()
```

```
Out[55]:
```

	MONTH	DAY	HOUR	MINUTE	YEAR
0	1	1	16	25	2015
1	1	1	8	23	2015
2	1	1	13	5	2015
3	1	1	15	55	2015
4	1	1	17	20	2015

```
# 用to_datetime函数，将df_date变为Timestamps对象
```

```
In[56]: flight_dep = pd.to_datetime(df_date)
        flight_dep.head()
```

```
Out[56]: 0    2015-01-01 16:25:00
        1    2015-01-01 08:23:00
        2    2015-01-01 13:05:00
        3    2015-01-01 15:55:00
        4    2015-01-01 17:20:00
        dtype: datetime64[ns]
```

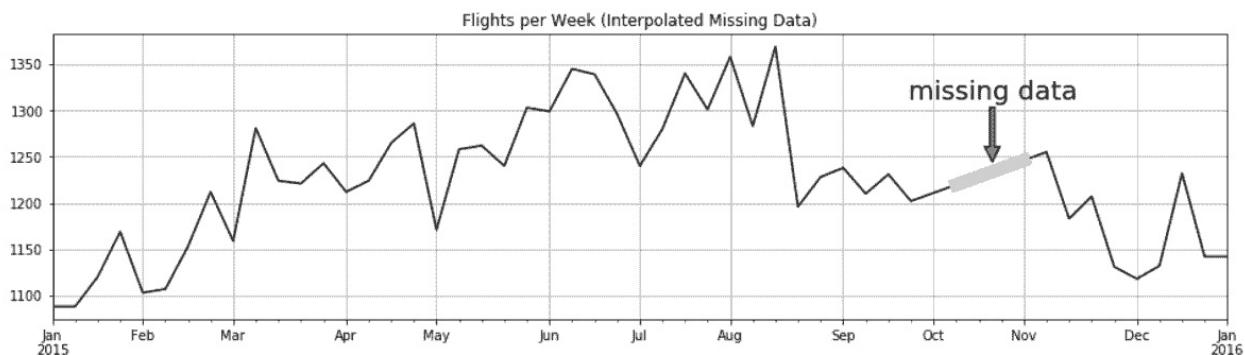
```
# 用flight_dep作为新的行索引，并根据它统计每周的航班数
In[57]: flights.index = flight_dep
         fc = flights.resample('W').size()
         fc.plot(figsize=(12,3), title='Flights per Week', grid=True)
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x109d116d8>
```



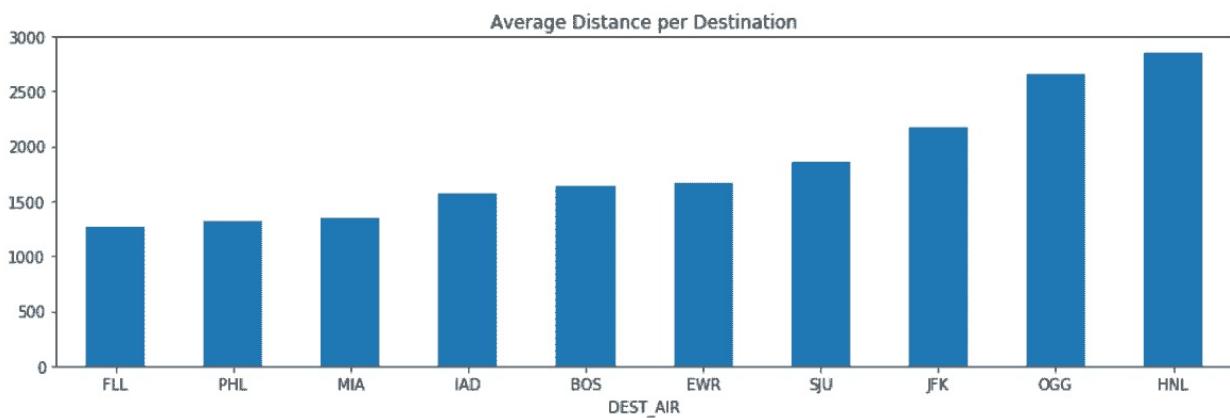
```
# 如果航班数小于1000，则将其当做缺失值。然后用interpolate方法填补缺失值
In[58]: fc_miss = fc.where(fc > 1000)
         fc_intp = fc_miss.interpolate(limit_direction='both')

         ax = fc_intp.plot(color='black', figsize=(16,4))
         fc_intp[fc < 500].plot(linewidth=10, grid=True,
                               color='.8', ax=ax)

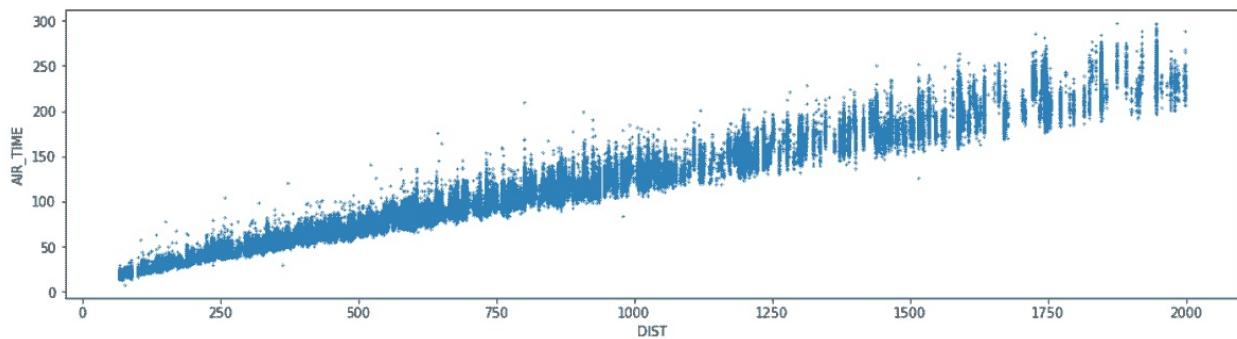
         ax.annotate(xy=(.8, .55), xytext=(.8, .77),
                     xycoords='axes fraction', s='missing data',
                     ha='center', size=20, arrowprops=dict())
         ax.set_title('Flights per Week (Interpolated Missing Data)')
Out[58]: Text(0.5,1,'Flights per Week (Interpolated Missing Data)')
```



```
# 找到10个有最长平均入境航班航程、最少100航次的机场
In[59]: flights.groupby('DEST_AIR')['DIST'] \
    .agg(['mean', 'count']) \
    .query('count > 100') \
    .sort_values('mean') \
    .tail(10) \
    .plot(kind='bar', y='mean', legend=False,
          rot=0, figsize=(14,4),
          title='Average Distance per Destination')
Out[59]: <matplotlib.axes._subplots.AxesSubplot at 0x11a480dd8>
```



```
# 画出航班时间和航程的散点图
In[60]: fs = flights.reset_index(drop=True)[['DIST', 'AIR_TIME']] \
    .query('DIST <= 2000').dropna()
    fs.plot(x='DIST', y='AIR_TIME', kind='scatter', s=1, \
    figsize=(16,4))
Out[60]: <matplotlib.axes._subplots.AxesSubplot at 0x11a49b860>
```



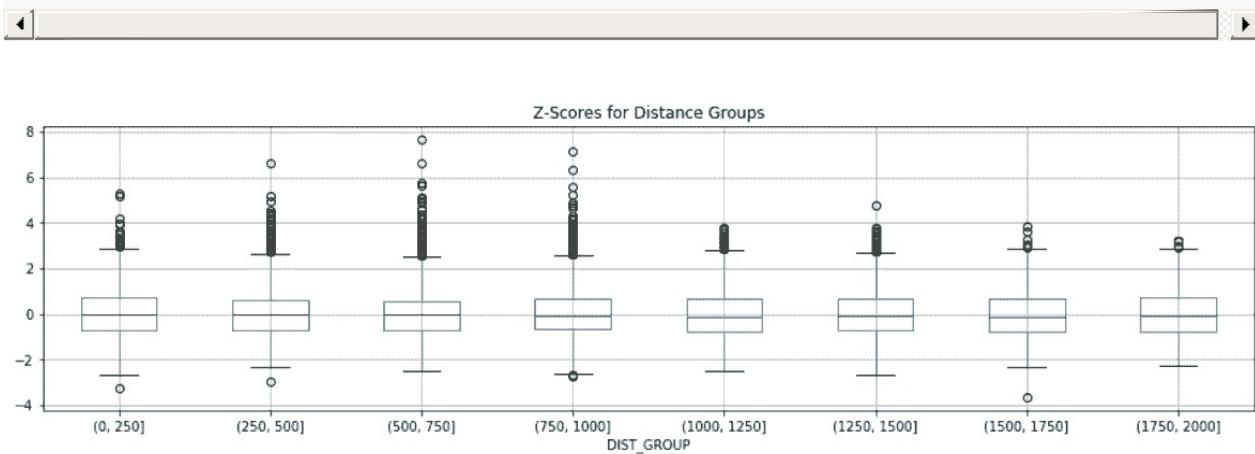
```
# 用cut函数，将航班距离分成八组
In[61]: fs['DIST_GROUP'] = pd.cut(fs['DIST'], bins=range(0, 2001
, 250))
fs['DIST_GROUP'].value_counts().sort_index()
Out[61]: (0, 250]           6529
          (250, 500]         12631
          (500, 750]         11506
          (750, 1000]        8832
          (1000, 1250]       5071
          (1250, 1500]       3198
          (1500, 1750]       3885
          (1750, 2000]       1815
Name: DIST_GROUP, dtype: int64
```

```
# 计算每组的标准差
In[62]: normalize = lambda x: (x - x.mean()) / x.std()
         fs['TIME_SCORE'] = fs.groupby('DIST_GROUP')['AIR_TIME']
                                \
                                .transform(normalize)

         fs.head()
Out[62]:
```

	DIST	AIR_TIME	DIST_GROUP	TIME_SCORE
0	590	94.0	(500, 750]	0.490966
1	1452	154.0	(1250, 1500]	-1.267551
2	641	85.0	(500, 750]	-0.296749
3	1192	126.0	(1000, 1250]	-1.211020
4	1363	166.0	(1250, 1500]	-0.521999

```
# 用boxplot方法画出异常值
In[63]: ax = fs.boxplot(by='DIST_GROUP', column='TIME_SCORE', figsize=(16,4))
          ax.set_title('Z-Scores for Distance Groups')
          ax.figure.suptitle('')
/Users/Ted/anaconda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:57: FutureWarning: reshape is deprecated and will raise
  in a subsequent release. Please use .values.reshape(...) instead
  return getattr(obj, method)(*args, **kwds)
Out[63]: Text(0.5,0.98,'')
```



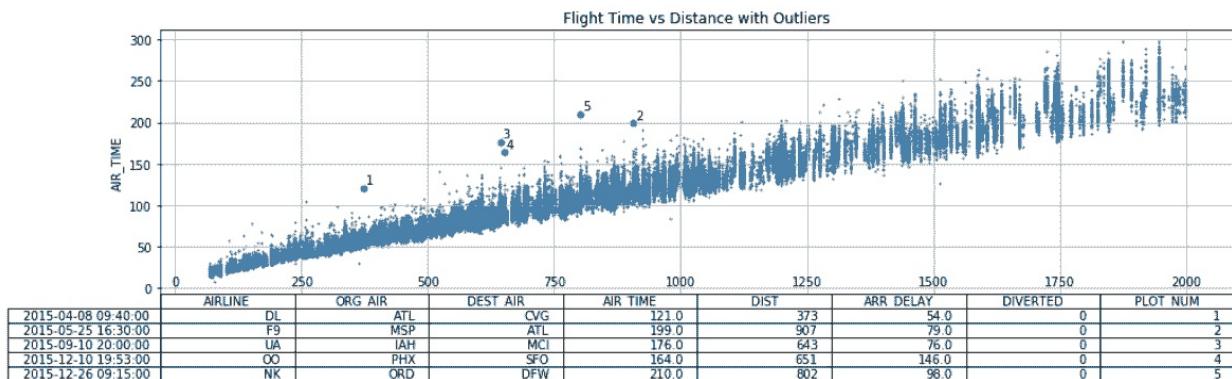
```
# 检查超出6个标准偏差的点。用一个DataFrame记录异常点。
In[64]: outliers = flights.iloc[fs[fs['TIME_SCORE'] > 6].index]
         outliers = outliers[['AIRLINE', 'ORG_AIR', 'DEST_AIR',
         'AIR_TIME',
         'DIST', 'ARR_DELAY', 'DIVERTED']]
         outliers['PLOT_NUM'] = range(1, len(outliers) + 1)
         outliers
Out[64]:
```

	AIRLINE	ORG_AIR	DEST_AIR	AIR_TIME	DIST	ARR_DELAY	DIVERTED	PLOT_NUM
2015-04-08 09:40:00	DL	ATL	CVG	121.0	373	54.0	0	1
2015-05-25 16:30:00	F9	MSP	ATL	199.0	907	79.0	0	2
2015-09-10 20:00:00	UA	IAH	MCI	176.0	643	76.0	0	3
2015-12-10 19:53:00	OO	PHX	SFO	164.0	651	146.0	0	4
2015-12-26 09:15:00	NK	ORD	DFW	210.0	802	98.0	0	5

```
# 可以这张表的数据确定异常值。pandas提供了将表格附加于图片底部的方法。
In[65]: ax = fs.plot(x='DIST', y='AIR_TIME',
                     kind='scatter', s=1,
                     figsize=(16,4), table=outliers)
outliers.plot(x='DIST', y='AIR_TIME',
               kind='scatter', s=25, ax=ax, grid=True)

outs = outliers[['AIR_TIME', 'DIST', 'PLOT_NUM']]
for t, d, n in outs.itertuples(index=False):
    ax.text(d + 5, t + 5, str(n))

plt.setp(ax.get_xticklabels(), y=.1)
plt.setp(ax.get_xticklines(), visible=False)
ax.set_xlabel('DIST')
ax.set_title('Flight Time vs Distance with Outliers')
Out[65]: Text(0.5,1,'Flight Time vs Distance with Outliers')
```



5. 堆叠面积图，以发现趋势

```
# 读取meetup_groups数据集
In[66]: meetup = pd.read_csv('data/meetup_groups.csv',
                           parse_dates=['join_date'],
                           index_col='join_date')
meetup.head()
Out[66]:
```

		group	city	state	country
	join_date				
2016-11-18 02:41:29	houston machine learning	Houston	TX	us	
2017-05-09 14:16:37	houston machine learning	Houston	TX	us	
2016-12-30 02:34:16	houston machine learning	Houston	TX	us	
2016-07-18 00:48:17	houston machine learning	Houston	TX	us	
2017-05-25 12:58:16	houston machine learning	Houston	TX	us	

```
# 算出每周加入每个组的人
```

```
In[67]: group_count = meetup.groupby([pd.Grouper(freq='W'), 'group']).size()
group_count.head()
Out[67]: join_date      group
2010-11-07    houstonr     5
2010-11-14    houstonr    11
2010-11-21    houstonr     2
2010-12-05    houstonr     1
2011-01-16    houstonr     2
dtype: int64
```

```
# 将数据表unstack
```

```
In[68]: gc2 = group_count.unstack('group', fill_value=0)
gc2.tail()
Out[68]:
```

	group	houston data science	houston data visualization	houston energy data science	houston machine learning	houstonr
join_date						
2017-09-17		16	2	6	5	0
2017-09-24		19	4	16	12	7
2017-10-01		20	6	6	20	1
2017-10-08		22	10	10	4	2
2017-10-15		14	13	9	11	2

```
# 做累积求和
```

```
In[69]: group_total = gc2.cumsum()
group_total.tail()
Out[69]:
```

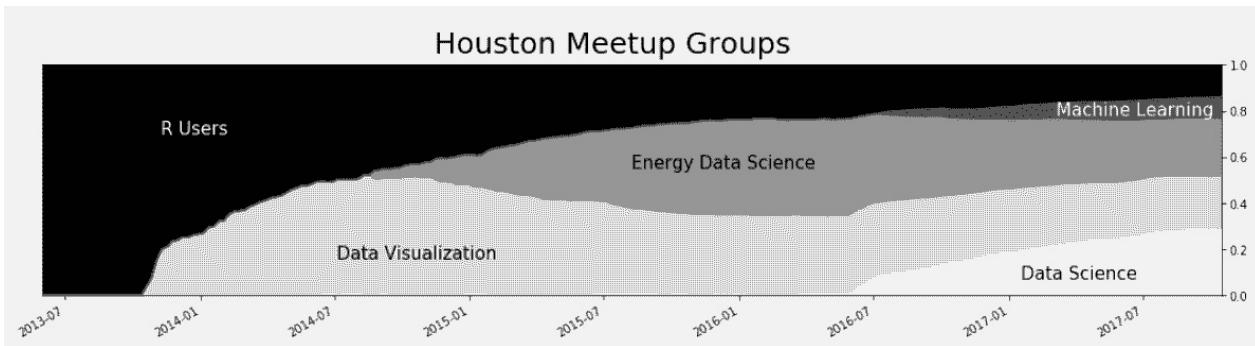
group	houston data science	houston data visualization	houston energy data science	houston machine learning	houstonr
join_date					
2017-09-17	2105	1708	1886	708	1056
2017-09-24	2124	1712	1902	720	1063
2017-10-01	2144	1718	1908	740	1064
2017-10-08	2166	1728	1918	744	1066
2017-10-15	2180	1741	1927	755	1068

```
# 将每行分开，已找到其在总数中的百分比
In[70]: row_total = group_total.sum(axis='columns')
          group_cum_pct = group_total.div(row_total, axis='index')
        )
        group_cum_pct.tail()
Out[70]:
```

group	houston data science	houston data visualization	houston energy data science	houston machine learning	houstonr
join_date					
2017-09-17	0.282058	0.228862	0.252713	0.094868	0.141498
2017-09-24	0.282409	0.227629	0.252892	0.095732	0.141338
2017-10-01	0.283074	0.226829	0.251914	0.097703	0.140481
2017-10-08	0.284177	0.226712	0.251640	0.097612	0.139858
2017-10-15	0.284187	0.226959	0.251206	0.098423	0.139226

```
# 话堆叠面积图
In[71]: ax = group_cum_pct.plot(kind='area', figsize=(18, 4),
                                 cmap='Greys', xlim=('2013-6', N
one),
                                 ylim=(0, 1), legend=False)
ax.figure.suptitle('Houston Meetup Groups', size=25)
ax.set_xlabel('')
ax.yaxis.tick_right()

plot_kwargs = dict(xycoords='axes fraction', size=15)
ax.annotate(xy=(.1, .7), s='R Users', color='w', **plot
_kw_args)
ax.annotate(xy=(.25, .16), s='Data Visualization', colo
r='k', **plot_kw_args)
ax.annotate(xy=(.5, .55), s='Energy Data Science', colo
r='k', **plot_kw_args)
ax.annotate(xy=(.83, .07), s='Data Science', color='k',
**plot_kw_args)
ax.annotate(xy=(.86, .78), s='Machine Learning', color=
'w', **plot_kw_args)
Out[71]: Text(0.86, 0.78, 'Machine Learning')
```



更多

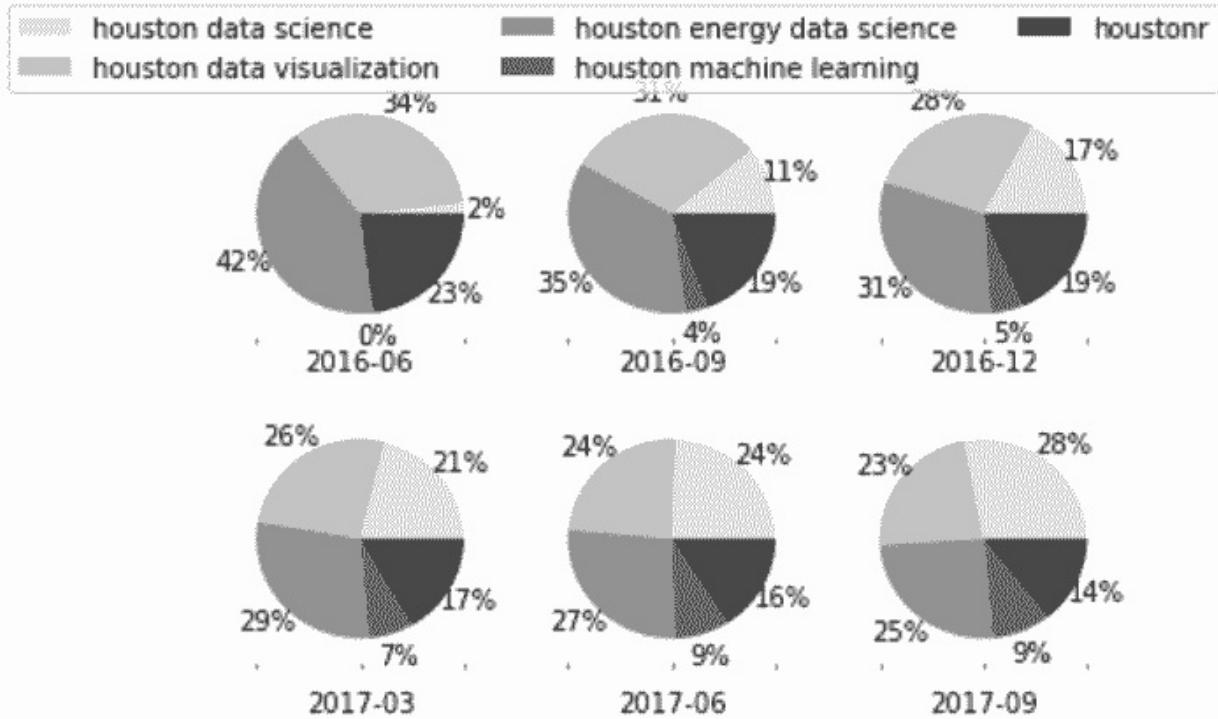
用饼图查看每组随时间的分布情况

```
In[72]: pie_data = group_cum_pct.asfreq('3MS', method='bfill')
\ .tail(6).to_period('M').T
pie_data
Out[72]:
```

	join_date	2016-06	2016-09	2016-12	2017-03	2017-06	2017-09
	group						
	houston data science	0.016949	0.110375	0.171245	0.212289	0.244033	0.280162
	houston data visualization	0.337827	0.306052	0.277244	0.261103	0.242085	0.230332
	houston energy data science	0.416025	0.354467	0.312271	0.288859	0.267576	0.253758
	houston machine learning	0.000000	0.037176	0.051969	0.071593	0.087839	0.093026
	houstonr	0.229199	0.191931	0.187271	0.166156	0.158467	0.142722

```
In[73]: from matplotlib.cm import Greys
greys = Greys(np.arange(50, 250, 40))
```

```
ax_array = pie_data.plot(kind='pie', subplots=True,
                           layout=(2, 3), labels=None,
                           autopct='%1.0f%%', pctdistance=
                           1.22,
                           colors=greys)
ax1 = ax_array[0, 0]
ax1.figure.legend(ax1.patches, pie_data.index, ncol=3)
for ax in ax_array.flatten():
    ax.xaxis.label.set_visible(True)
    ax.set_xlabel(ax.get_ylabel())
    ax.set_ylabel('')
ax1.figure.subplots_adjust(hspace=.3)
Out[73]:
```

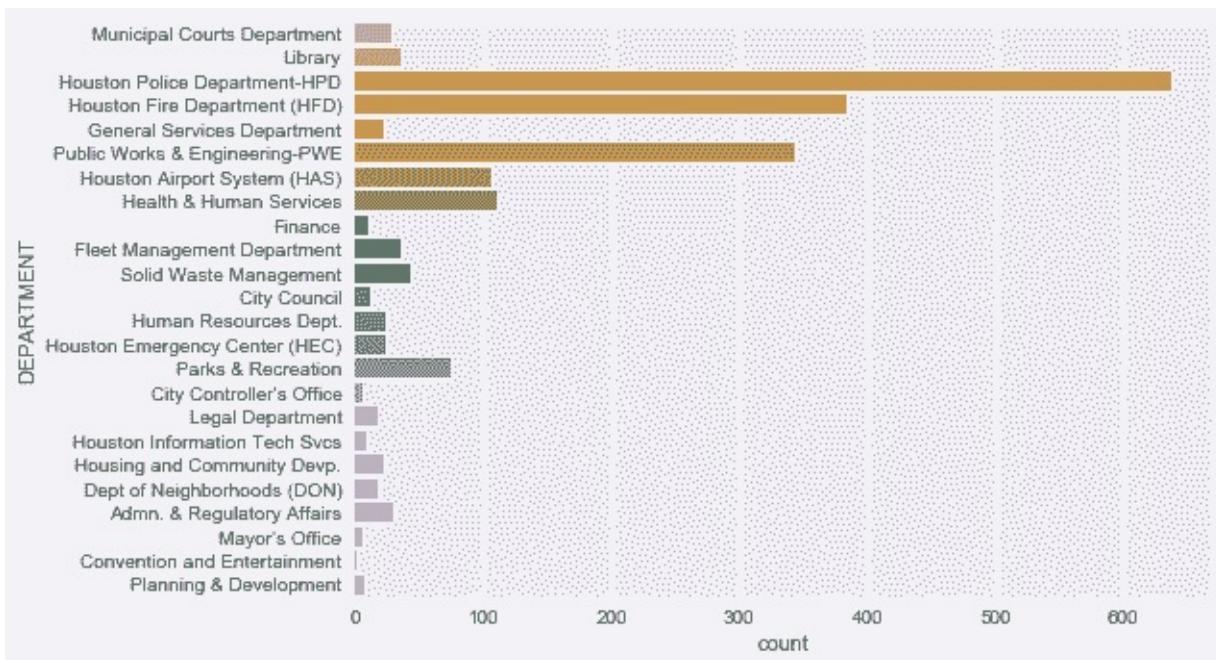


6. Seaborn和Pandas的不同点

```
# 读取employee数据集
In[74]: employee = pd.read_csv('data/employee.csv',
                               parse_dates=['HIRE_DATE', 'JOB_DATE'])
          employee.head()
Out[74]:
```

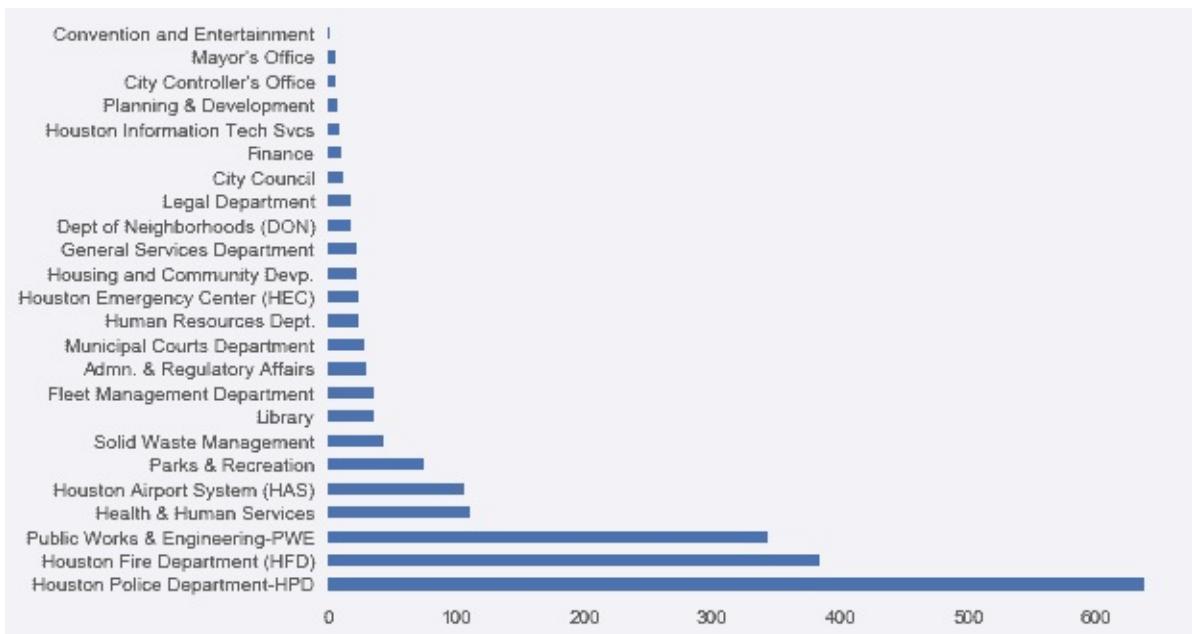
	UNIQUE_ID	POSITION_TITLE	DEPARTMENT	BASE_SALARY	RACE	EMPLOYMENT_TYPE	GENDER	EMPLOYN
0	0	ASSISTANT DIRECTOR (EX LVL)	Municipal Courts Department	121862.0	Hispanic/Latino	Full Time	Female	
1	1	LIBRARY ASSISTANT	Library	26125.0	Hispanic/Latino	Full Time	Female	
2	2	POLICE OFFICER	Houston Police Department-HPD	45279.0	White	Full Time	Male	
3	3	ENGINEER/OPERATOR	Houston Fire Department (HFD)	63166.0	White	Full Time	Male	
4	4	ELECTRICIAN	General Services Department	56347.0	White	Full Time	Male	

```
# 用seaborn画出每个部门的柱状图
In[75]: import seaborn as sns
In[76]: sns.countplot(y='DEPARTMENT', data=employee)
Out[76]: <matplotlib.axes._subplots.AxesSubplot at 0x11e287128>
```



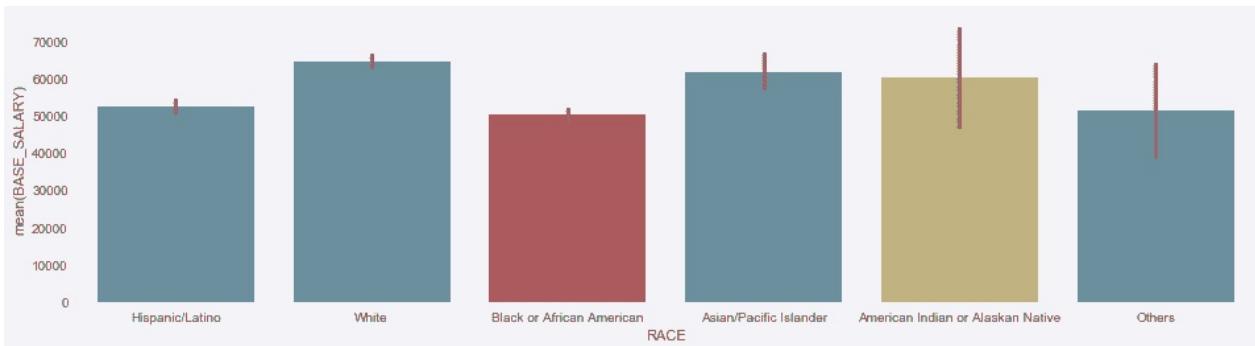
```
# 要是用pandas来做，需要先聚合数据
```

```
In[77]: employee['DEPARTMENT'].value_counts().plot('barh')
Out[77]: <matplotlib.axes._subplots.AxesSubplot at 0x11e30a240>
```



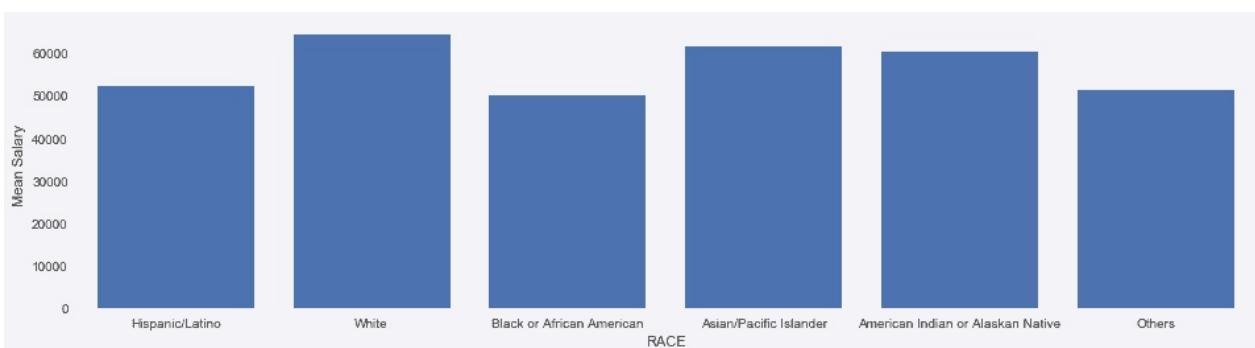
```
# 用seaborn找到每个种族的平均工资
```

```
In[78]: ax = sns.barplot(x='RACE', y='BASE_SALARY', data=employee)
                  ax.figure.set_size_inches(16, 4)
Out[78]:
```



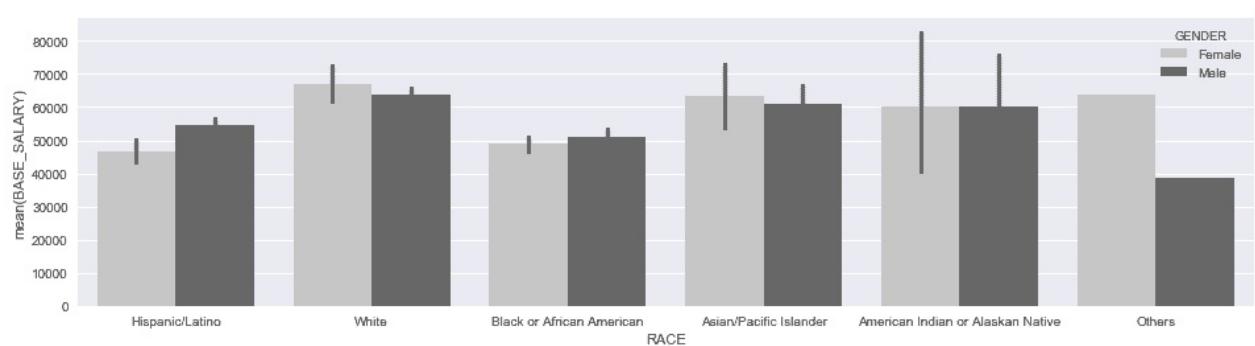
用pandas来做，需要先按照race分组

```
In[79]: avg_sal = employee.groupby('RACE', sort=False)[['BASE_SALARY']].mean()
        ax = avg_sal.plot(kind='bar', rot=0, figsize=(16, 4), width=.8)
        ax.set_xlim(-.5, 5.5)
        ax.set_ylabel('Mean Salary')
Out[79]: Text(0, 0.5, 'Mean Salary')
```

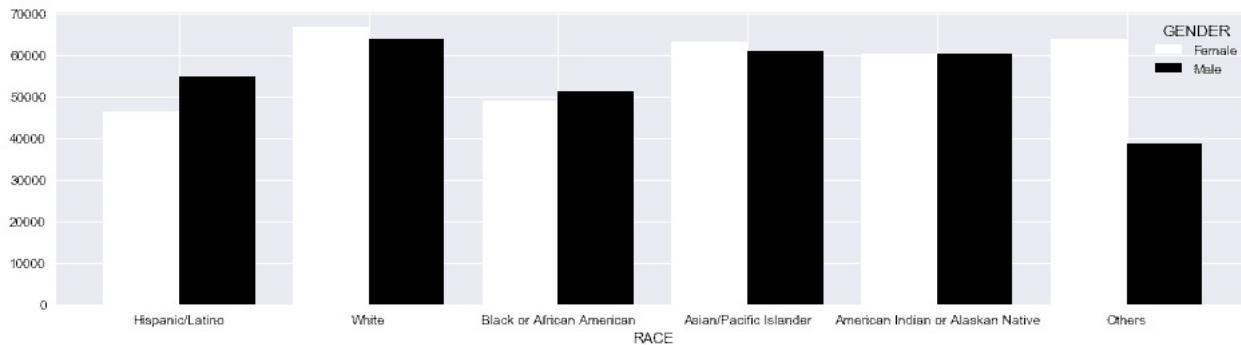


seaborn还支持在分组内使用第三个参数

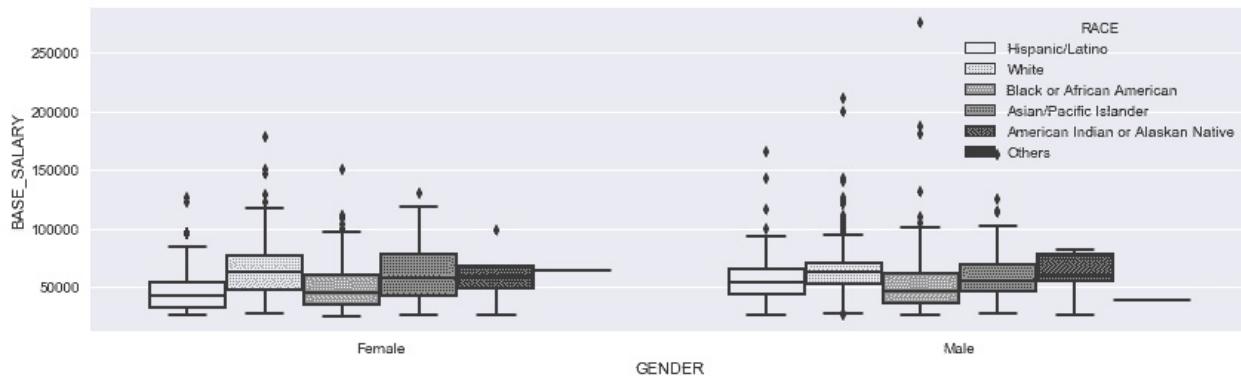
```
In[80]: ax = sns.barplot(x='RACE', y='BASE_SALARY', hue='GENDER',
                      data=employee, palette='Greys')
        ax.figure.set_size_inches(16, 4)
Out[80]:
```



```
# pandas则要对race和gender同时分组，并对gender做unstack
In[81]: employee.groupby(['RACE', 'GENDER'], sort=False)['BASE_SALARY'] \
          .mean().unstack('GENDER') \
          .plot(kind='bar', figsize=(16,4), rot=0,
                width=.8, cmap='Greys')
Out[81]: <matplotlib.axes._subplots.AxesSubplot at 0x11ecf45c0>
```

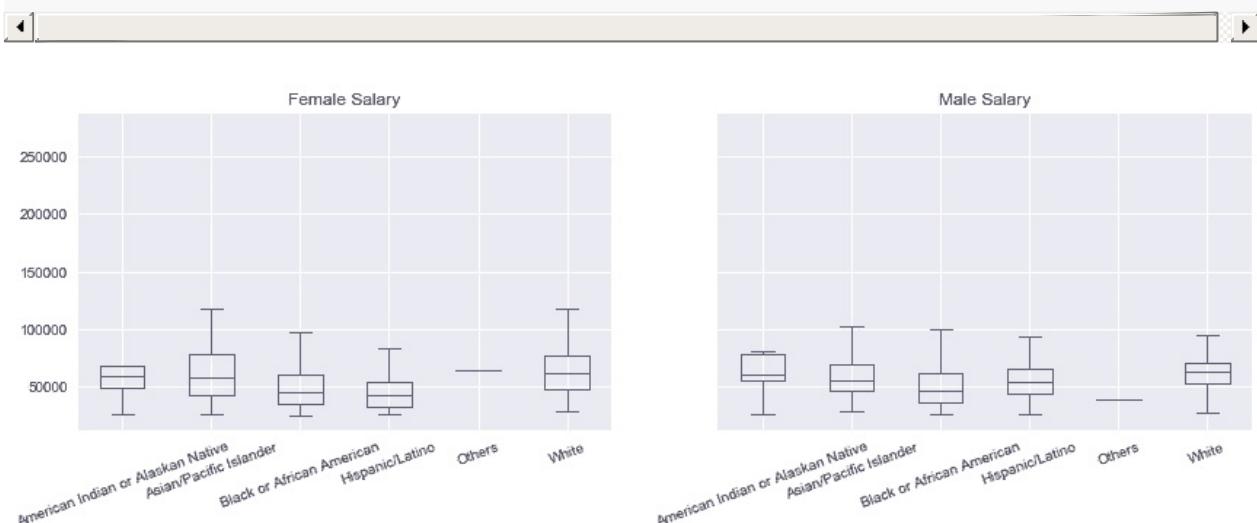


```
# 用seaborn话race和gender的盒图
In[82]: ax = sns.boxplot(x='GENDER', y='BASE_SALARY', data=employee,
                      hue='RACE', palette='Greys')
          ax.figure.set_size_inches(14,4)
Out[82]:
```



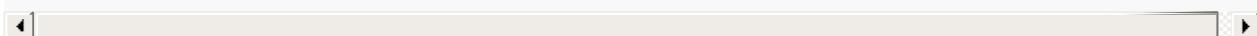
```
# pandas则要为gender创建两个独立的Axes，然后根据race画盒图
In[83]: fig, ax_array = plt.subplots(1, 2, figsize=(14,4), sharey=True)
        for g, ax in zip(['Female', 'Male'], ax_array):
            employee.query('GENDER== @g') \
                .boxplot(by='RACE', column='BASE_SALARY', ax=ax, rot=20)
            ax.set_title(g + ' Salary')
            ax.set_xlabel('')
        fig.suptitle('')

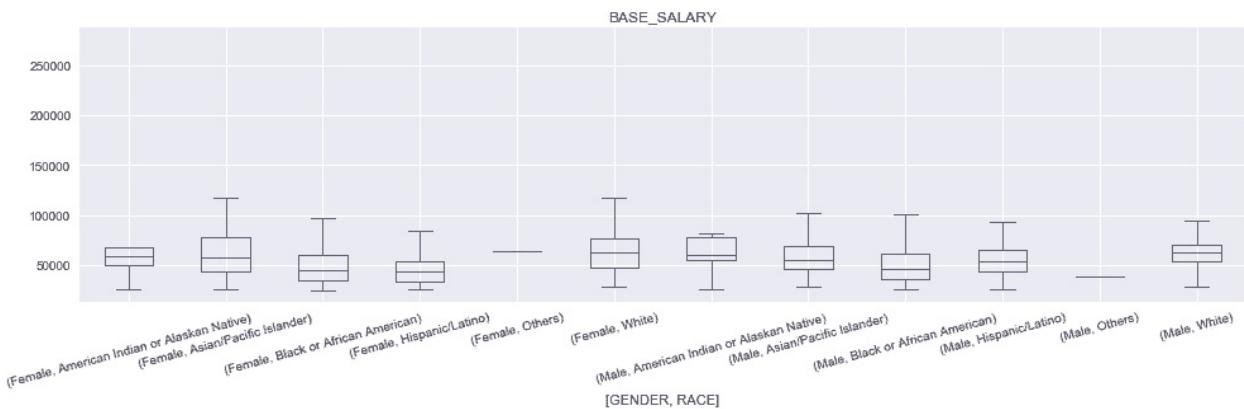
/Users/Ted/anaconda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:57: FutureWarning: reshape is deprecated and will raise
    in a subsequent release. Please use .values.reshape(...) instead
d
    return getattr(obj, method)(*args, **kwds)
Out[83]: Text(0.5,0.98, '')
```



```
# pandas也可以列表分离多个变量，但是画的图不优雅
In[84]: ax = employee.boxplot(by=['GENDER', 'RACE'],
                               column='BASE_SALARY',
                               figsize=(16,4), rot=15)
        ax.figure.suptitle('')

/Users/Ted/anaconda/lib/python3.6/site-packages/numpy/core/fromnumeric.py:57: FutureWarning: reshape is deprecated and will raise
    in a subsequent release. Please use .values.reshape(...) instead
d
    return getattr(obj, method)(*args, **kwds)
Out[84]: Text(0.5,0.98, '')
```





7. 使用Seaborn网格做多变量分析

```
# 读取employee数据集，创建工龄的列
In[85]: employee = pd.read_csv('data/employee.csv',
                               parse_dates=['HIRE_DATE', 'JOB_DATE'])
          days_hired = (pd.to_datetime('12-1-2016') - employee['HIRE_DATE'])

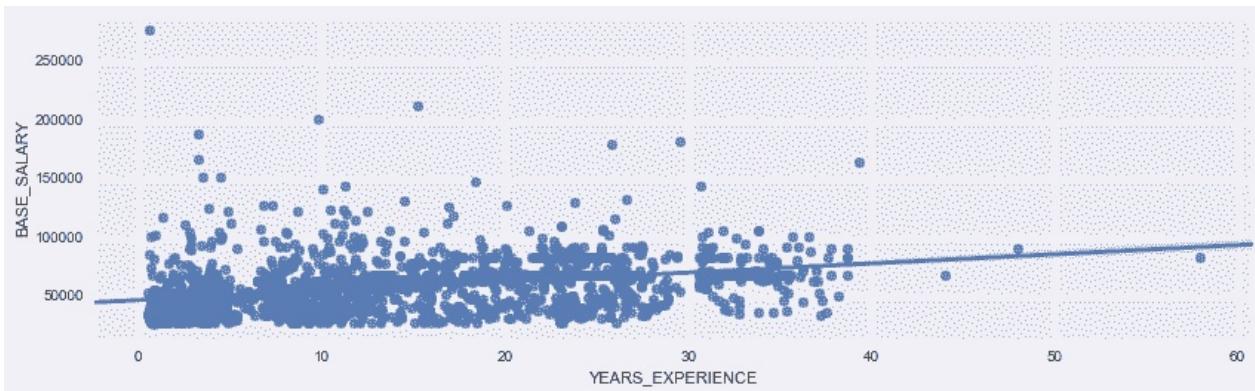
          one_year = pd.Timedelta(1, unit='Y')
          employee['YEARS_EXPERIENCE'] = days_hired / one_year
          employee[['HIRE_DATE', 'YEARS_EXPERIENCE']].head()

Out[85]:
```

	HIRE_DATE	YEARS_EXPERIENCE
0	2006-06-12	10.472494
1	2000-07-19	16.369946
2	2015-02-03	1.826184
3	1982-02-08	34.812488
4	1989-06-19	27.452994

```
# 画一个基本的带有回归线的散点图
In[86]: import seaborn as sns
In[87]: ax = sns.regplot(x='YEARS_EXPERIENCE', y='BASE_SALARY',
                      data=employee)
          ax.figure.set_size_inches(14, 4)

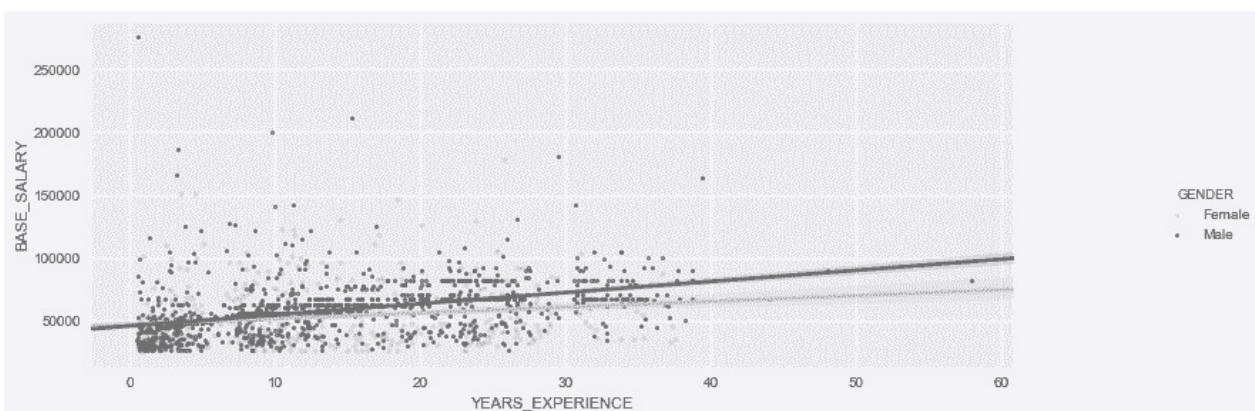
Out[87]:
```



用regplot的上层函数lmplot，画出不同性别的回归线

```
In[88]: grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY'
,
hue='GENDER', palette='Greys',
scatter_kws={'s':10}, data=employee)
grid.fig.set_size_inches(14, 4)
type(grid)
```

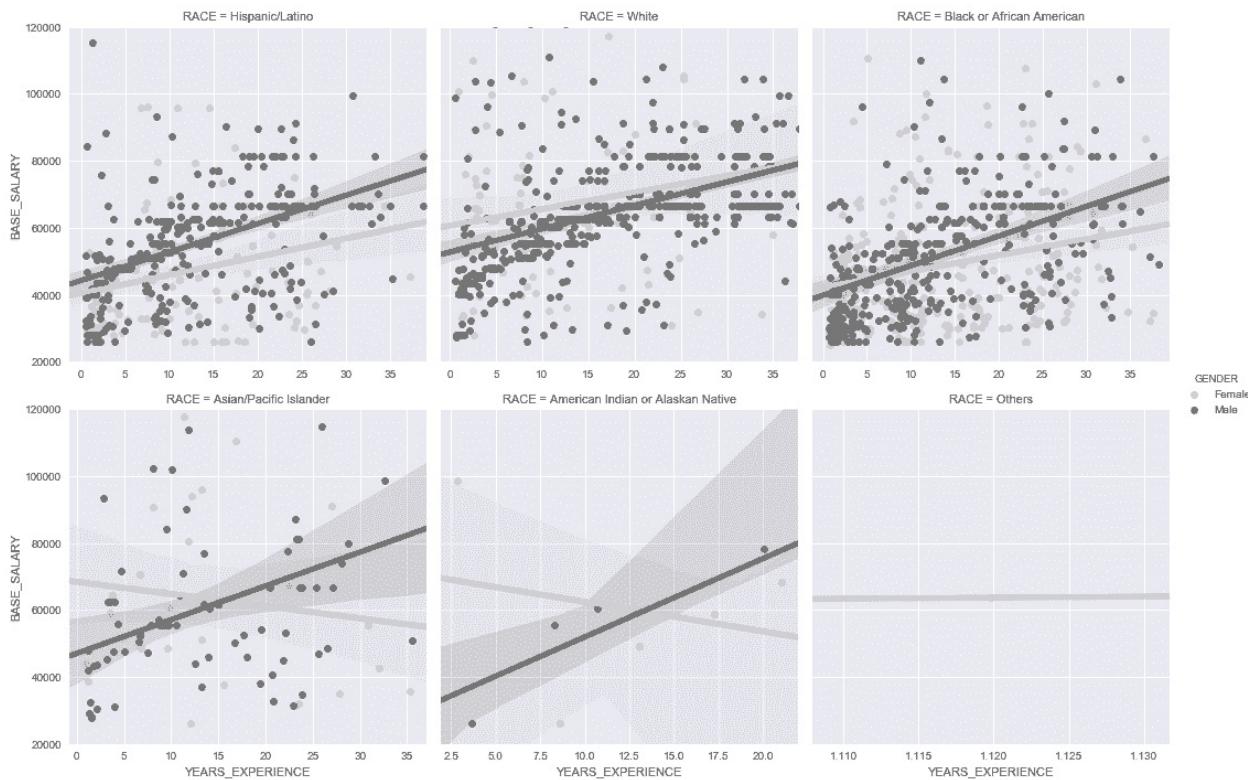
Out[88]: seaborn.axisgrid.FacetGrid



为每个种族创建子图，同时保留回归线

```
In[89]: grid = sns.lmplot(x='YEARS_EXPERIENCE', y='BASE_SALARY'
,
hue='GENDER', col='RACE', col_wrap=3,
palette='Greys', sharex=False,
line_kws = {'linewidth':5},
data=employee)
grid.set(ylim=(20000, 120000))
```

Out[89]: <seaborn.axisgrid.FacetGrid at 0x11e7ce470>



将类型值的层级减小到二，将部门的层级减小到三

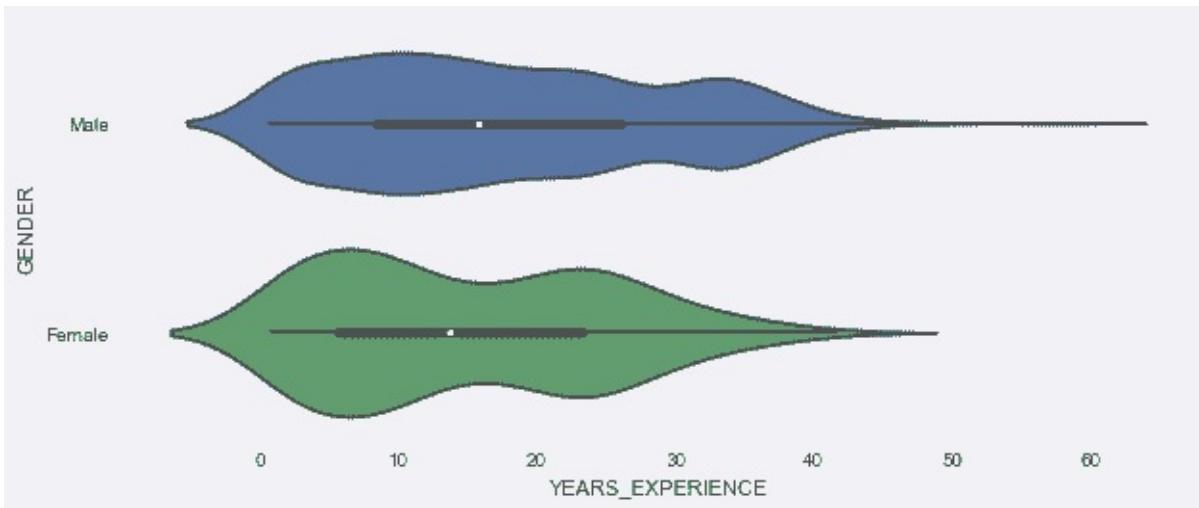
```
In[90]: deps = employee['DEPARTMENT'].value_counts().index[:2]
races = employee['RACE'].value_counts().index[:3]
is_dep = employee['DEPARTMENT'].isin(deps)
is_race = employee['RACE'].isin(races)
emp2 = employee[is_dep & is_race].copy()
emp2['DEPARTMENT'] = emp2.DEPARTMENT.str.extract('(HPD|HFD)', expand=True)
emp2.shape
Out[90]: (968, 11)
```

```
In[91]: emp2['DEPARTMENT'].value_counts()
Out[91]: HPD      591
          HFD      377
          Name: DEPARTMENT, dtype: int64
```

```
In[92]: emp2['RACE'].value_counts()
Out[92]: White                  478
          Hispanic/Latino        250
          Black or African American 240
          Name: RACE, dtype: int64
```

用Axe层函数，比如violinplot来画出工龄和性别的分布

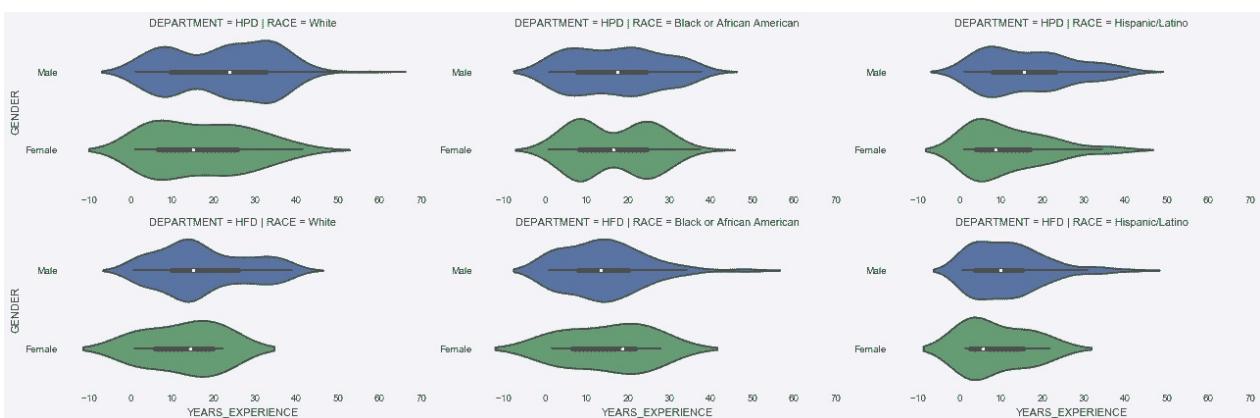
```
In[93]: ax = sns.violinplot(x = 'YEARS_EXPERIENCE', y='GENDER',
                           data=emp2)
                           ax.figure.set_size_inches(10, 4)
Out[93]:
```



```
# 用factorplot函数，为每个部门和种族的组合画图
```

```
In[94]: sns.factorplot(x ='YEARS_EXPERIENCE', y='GENDER',
                      col='RACE', row='DEPARTMENT',
                      size=3, aspect=2,
                      data=emp2, kind='violin')
```

```
Out[94]: <seaborn.axisgrid.FacetGrid at 0x11e40ec50>
```



8. 用Seaborn破解diamonds数据集的辛普森悖论

```
In[95]: pd.DataFrame(index=['Student A', 'Student B'],
                      data={'Raw Score': ['50/100', '80/100'],
                            'Percent Correct':[50, 80]}, columns=[
    'Raw Score', 'Percent Correct'])
```

```
Out[95]:
```

	Raw Score	Percent Correct
Student A	50/100	50
Student B	80/100	80

```
In[96]: pd.DataFrame(index=['Student A', 'Student B'],
                     data={'Difficult': ['45/95', '2/5'],
                            'Easy': ['5/5', '78/95'],
                            'Difficult Percent': [47, 40],
                            'Easy Percent': [100, 82],
                            'Total Percent':[50, 80]},
                     columns=['Difficult', 'Easy', 'Difficult Percent',
                            'Easy Percent', 'Total Percent'])
Out[96]:
```

	Difficult	Easy	Difficult Percent	Easy Percent	Total Percent
Student A	45/95	5/5	47	100	50
Student B	2/5	78/95	40	82	80

```
# 读取diamonds数据集
In[97]: diamonds = pd.read_csv('data/diamonds.csv')
          diamonds.head()
Out[97]:
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

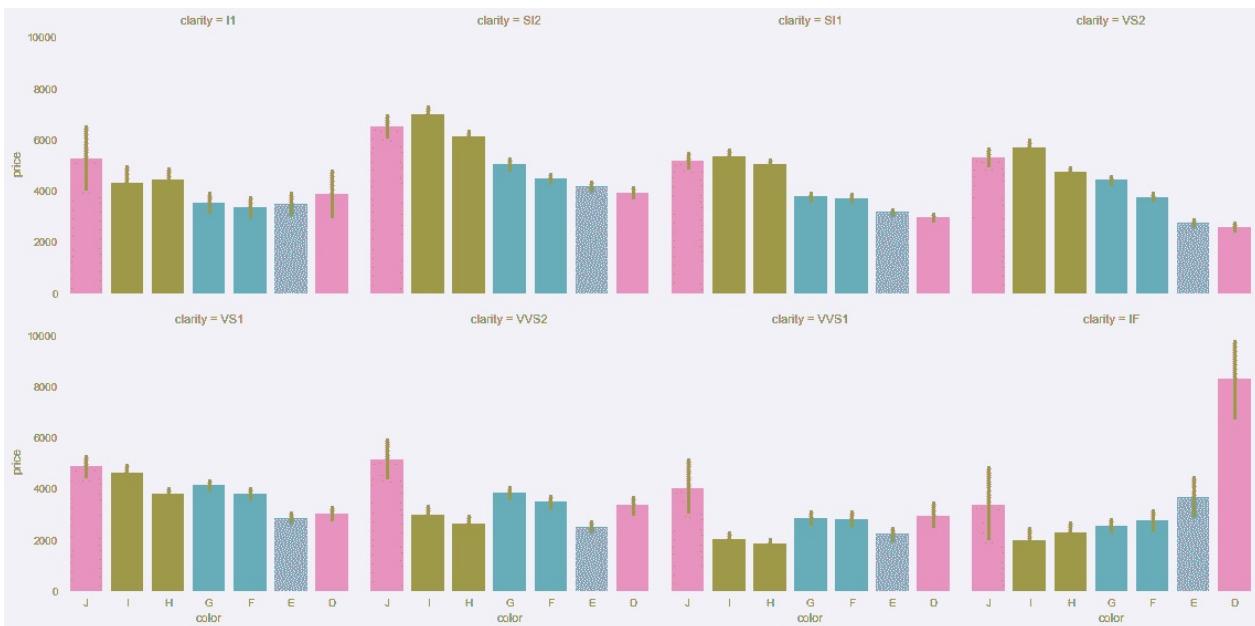
```
# 将cut、color、clarity列变为有序类型
In[98]: cut_cats = ['Fair', 'Good', 'Very Good', 'Premium', 'Ideal']
          color_cats = ['J', 'I', 'H', 'G', 'F', 'E', 'D']
          clarity_cats = ['I1', 'SI2', 'SI1', 'VS2', 'VS1', 'VVS2',
          , 'VVS1', 'IF']
          diamonds['cut'] = pd.Categorical(diamonds['cut'],
                                              categories=cut_cats,
                                              ordered=True)

          diamonds['color'] = pd.Categorical(diamonds['color'],
                                              categories=color_cat
          s,
                                              ordered=True)

          diamonds['clarity'] = pd.Categorical(diamonds['clarity'],
          ],,
                                              categories=clarity
          _cats,
                                              ordered=True)
In[99]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(14,4))
          sns.barplot(x='color', y='price', data=diamonds, ax=ax1)
          sns.barplot(x='cut', y='price', data=diamonds, ax=ax2)
          sns.barplot(x='clarity', y='price', data=diamonds, ax=a
          x3)
          fig.suptitle('Price Decreasing with Increasing Quality?')
Out[98]: Text(0.5,0.98,'Price Decreasing with Increasing Quality
?')
```

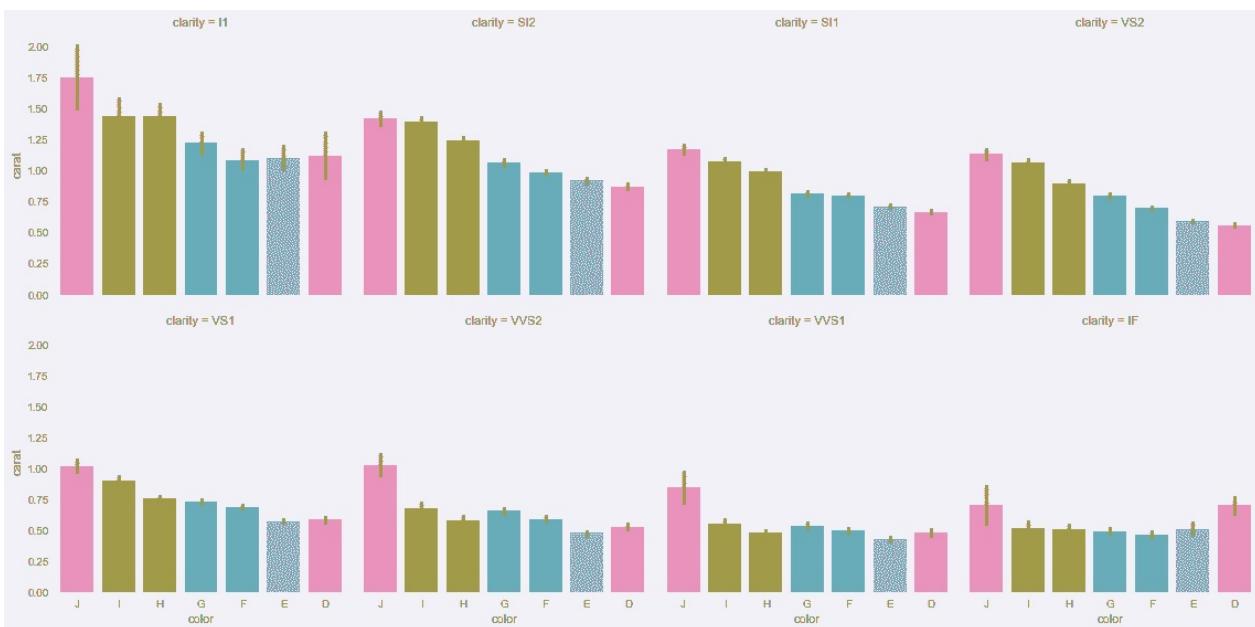


```
# 画出每种钻石颜色和价格的关系
In[100]: sns.factorplot(x='color', y='price', col='clarity',
                         col_wrap=4, data=diamonds, kind='bar')
Out[100]: <seaborn.axisgrid.FacetGrid at 0x11b61d5f8>
```

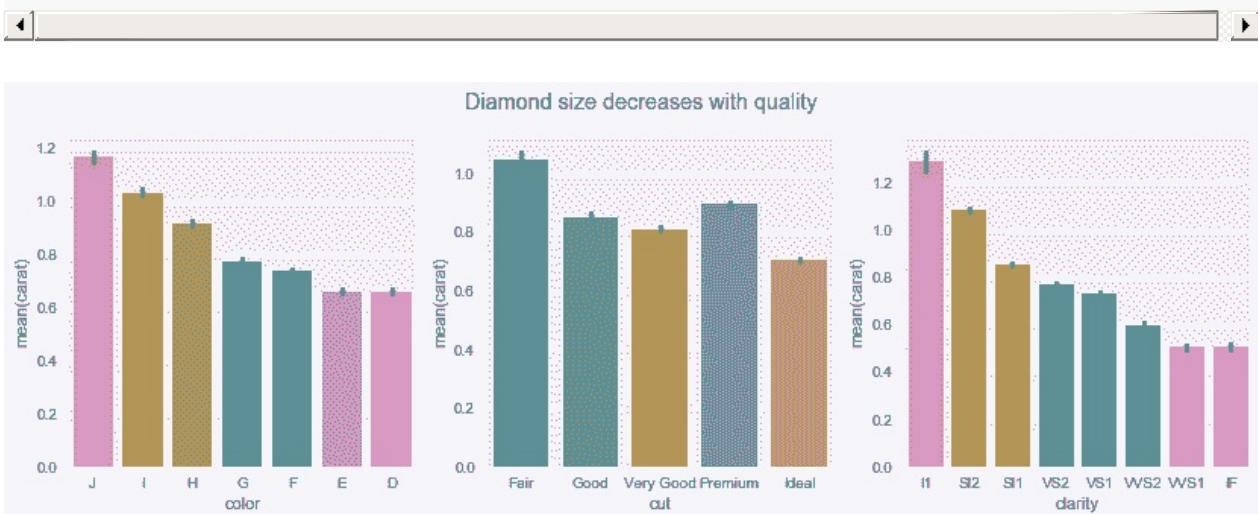


```
# 用克拉值取代价格
```

```
In[101]: sns.factorplot(x='color', y='carat', col='clarity',
                      col_wrap=4, data=diamonds, kind='bar')
Out[101]: <seaborn.axisgrid.FacetGrid at 0x11e42eeef0>
```



```
In[102]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(14, 4))
        sns.barplot(x='color', y='carat', data=diamonds, ax=ax1)
        sns.barplot(x='cut', y='carat', data=diamonds, ax=ax2)
        sns.barplot(x='clarity', y='carat', data=diamonds, ax=ax3)
        fig.suptitle('Diamond size decreases with quality')
Out[102]: Text(0.5, 0.98, 'Diamond size decreases with quality')
```



下图显示钻石越大，价格越高

```
In[103]: diamonds['carat_category'] = pd.qcut(diamonds.carat, 5)

from matplotlib.cm import Greys
greys = Greys(np.arange(50, 250, 40))

g = sns.factorplot(x='clarity', y='price', data=diamonds,
                    hue='carat_category', col='color',
                    col_wrap=4, kind='point') #, palette=greys)
g.fig.suptitle('Diamond price by size, color and clarity',
                y=1.02, size=20)
Out[103]: Text(0.5, 1.02, 'Diamond price by size, color and clarity')
```



更多

```
# 用seaborn更高级的PairGrid构造器，对二元变量作图
In[104]: g = sns.PairGrid(diamonds, size=5,
                           x_vars=["color", "cut", "clarity"],
                           y_vars=["price"])
g.map(sns.barplot)
g.fig.suptitle('Replication of Step 3 with PairGrid',
y=1.02)
Out[104]: Text(0.5, 1.02, 'Replication of Step 3 with PairGrid')
```

