人工智能程序设计

M2 科学计算与数据分析基础

2.3 pandas与数据分析

张 莉

表格型数据

Date	Open	High	Low	Close	Adj Close	Volume
2018/8/6	25437.43	25540.02	25381.38	25502.18	25502.18	238990000
2018/8/7	25551.65	25692.72	25551.65	25628.91	25628.91	239910000
2018/8/8	25615.72	25634.11	25557.48	25583.75	25583.75	217770000
2018/8/9	25589.79	25613.31	25492.69	25509.23	25509.23	214970000
2018/8/10	25401.19	25401.19	25222.88	25313.14	25313.14	234480000
2018/8/13	25327.19	25381.39	25153.93	25187.7	25187.7	219990000
2018/8/14	25215.69	25339.51	25201.87	25299.92	25299.92	219210000
2018/8/15	25235.37	25235.37	24965.77	25162.41	25162.41	295810000
2018/8/16	25294.97	25607.34	25294.97	25558.73	25558.73	342430000
2018/8/17	25550.8	25728.16	25521.66	25669.32	25669.32	284160000

pandas

1. Series
2. DataFrame
3. 基于Series和DataFrame的数据统计和分析

1 SERIES

人工智能程序设计

Series

- **基本特征**
  - 类似一维数组的对象
  - 由数据和索引组成（有序字典，称变长字典）

Series()函数

Source

```
import pandas as pd
>>> aSer = pd.Series([1, 2.0, 'a'])
>>> aSer
0   1
1   2
2   a
dtype: object
```

自定义Series的index

Source

```
>>> bSer = pd.Series(['apple','peach','lemon'], index = [1,2,3])
>>> bSer
1    apple
2    peach
3    lemon
dtype: object
>>> bSer.index        # 常进行单独赋值
Int64Index([1, 2, 3], dtype = 'int64')
>>> bSer.values
array(['apple', 'peach', 'lemon'], dtype = object)
```

Series的基本运算

Source

```
>>> cSer = pd.Series([3, 5, 7], index = ['a', 'b', 'c'])
>>> cSer['b']
5
>>> cSer * 2
a     6
b    10
c    14
dtype: int64
>>> import numpy as np
>>> np.exp(cSer)
a       20.085537
b      148.413159
c     1096.633158
dtype: float64
```

Series的基本运算

切片

基于位置

基于索引

Source

```
>>> cSer = pd.Series([3, 5, 7], index = ['a', 'b', 'c'])
>>> cSer[1: 2]
b   5
dtype: int64
>>> cSer['a': 'b']
a   3
b   5
dtype: int64
```

人工智能程序设计

2 DATAFRAME

DataFrame

- **基本特征**
  - 一个表格型的数据结构（称数据框）
  - 含有一组有序的列（类似于index）
  - 大致可看成共享同一个index的Series集合

	name	pay
0	Mayue	3000
1	Lilin	4500
2	Wuyun	8000

创建DataFrame

DataFrame()函数

Source

```
>>> data = {'name': ['Mayue', 'Lilin', 'Wuyun'], 'pay': [3000, 4500, 8000]}
>>> aDF = pd.DataFrame(data)
>>> aDF
     name   pay
0   Mayue  3000
1   Lilin  4500
2   Wuyun  8000
```

DataFrame的索引和值

Source

```
>>> data = np.array([('Mayue', 3000), ('Lilin', 4500), ('Wuyun', 8000)])
>>> bDF =pd.DataFrame(data, index = range(1, 4), columns = ['name', 'pay'])
>>> bDF
     name   pay
1   Mayue  3000
2   Lilin  4500
3   Wuyun  8000
>>> bDF.index        # 重新赋值即为修改行索引
RangeIndex(start=1, stop=4, step=1)
>>> bDF.columns      # 重新赋值即为修改列索引
Index(['name', 'pay'], dtype='object')
>>> bDF.values
array([['Mayue', '3000'],
       ['Lilin', '4500'],
       ['Wuyun', '8000']], dtype=object)
```

修改DataFrame-添加列

```
>>> aDF
   name   pay
0 Mayue  3000
1  Lilin 4500
2 Wuyun  8000
```

Source

```
>>> aDF['tax'] = [0.05, 0.05, 0.1]
>>> aDF
   name   pay   tax
0 Mayue  3000  0.05
1  Lilin 4500  0.05
2 Wuyun  8000   0.1
```

修改DataFrame-添加行

```
>>> aDF
      name   pay   tax
0    Mayue  3000  0.05
1    Lilin  4500  0.05
2   Wuyun   8000   0.1
```

Source

```
>>> aDF.loc[5] = {'name': 'Liuxi', 'pay': 5000, 'tax': 0.05}
>>> aDF
      name   pay   tax
0    Mayue  3000  0.05
1    Lilin  4500  0.05
2   Wuyun   8000   0.1
5    Liuxi  5000  0.05
```

修改DataFrame-添加行

```
>>> aDF
   name   pay   tax
0 Mayue  3000  0.05
1 Lilin  4500  0.05
2 Wuyun  8000   0.1
5 Liuxi  5000  0.05
>>> tempDF
   name   pay   tax
7 Yeqing 7000   0.1
9 Qianjie 9500  0.1
```

Source

```
>>> aDF.append(tempDF)
   name   pay   tax
0 Mayue  3000  0.05
1 Lilin  4500  0.05
2 Wuyun  8000   0.1
5 Liuxi  5000  0.05
7 Yeqing 7000   0.1
9 Qianjie 9500  0.1
```

修改DataFrame-添加行

```
>>> aDF
      name   pay   tax
0    Mayue  3000  0.05
1    Lilin  4500  0.05
2    Wuyun  8000   0.1
5    Liuxi  5000  0.05
>>> tempDF
      name   pay   tax
7   Yeqing  7000   0.1
9  Qianjie  9500   0.1
```

Source

```
>>> pieces = [aDF, tempDF]
>>> pd.concat(pieces)
      name   pay   tax
0    Mayue  3000  0.05
1    Lilin  4500  0.05
2    Wuyun  8000   0.1
5    Liuxi  5000  0.05
7   Yeqing  7000   0.1
9  Qianjie  9500   0.1
```

删除

```
>>> aDF
   name   pay   tax
0 Mayue  3000  0.05
1  Lilin  4500  0.05
2 Wuyun  8000   0.1
5  Liuxi  5000  0.05
```

Source

```
>>> aDF.drop(5)
   name   pay   tax
0 Mayue  3000  0.05
1  Lilin  4500  0.05
2 Wuyun  8000   0.1
>>> aDF.drop('tax', axis = 1)
   name   pay
0 Mayue  3000
1  Lilin  4500
2 Wuyun  8000
5  Liuxi  5000
```

inplace = True

修改DataFrame

```
>>> aDF
      name   pay   tax
0    Mayue  3000  0.05
1    Lilin  4500  0.05
2   Wuyun   8000   0.1
5    Liuxi  5000  0.05
```

Source

```
>>> aDF['tax'] = 0.03
>>> aDF
     name   pay   tax
0   Mayue  3000  0.03
1    Lilin  4500  0.03
2  Wuyun   8000  0.03
5    Liuxi  5000  0.03
>>> aDF.loc[5] = ['Liuxi', 9800, 0.05]
     name   pay   tax
0   Mayue  3000  0.03
1    Lilin  4500  0.03
2  Wuyun   8000  0.03
5    Liuxi  9800  0.05
>>> aDF['pay'] = aDF['pay'].astype('float32')
```

交换DataFrame元素

```
>>> df = pd.DataFrame(np.arange(1,10).reshape(3,3), index = ['a','b','c'], columns = ['F1','F2','F3'])
>>> df
   F1  F2  F3
a   1   2   3
b   4   5   6
c   7   8   9
>>> ind = ['c', 'b', 'a']
>>> col = ['F1', 'F3', 'F2']
>>> df.reindex(index = ind)
 F1  F2  F3
b   4   5   6
a   1   2   3
c   7   8   9
>>> df.reindex(columns = col)
 F1  F3  F2
a   1   3   2
b   4   6   5
c   7   9   8
```

df.reindex(ind, axis=0)

df.reindex(col, axis=1)
df.loc[:, col]
df.iloc[:, [0,2,1]]

DataFrame数据选择

	code	name	price
0	MMM	3M	155.82
1	AXP	American Express	114.41
2	AAPL	Apple	227.01
3	BA	Boeing	375.70
4	CAT	Caterpillar	121.04
5	CVX	Chevron	113.85
6	CSCO	Cisco	47.52
7	KO	Coca-Cola	54.54
8	DIS	Disney	130.27
9	DOW	Dow Chemical	45.34
10	XOM	Exxon Mobil	68.97
11	GS	Goldman Sachs	200.80
12	HD	Home Depot	227.93
13	IBM	IBM	142.99
14	INTC	Intel	50.92
15	JNJ	Johnson & Johnson	133.66
16	JPM	JPMorgan Chase	114.62
17	MCD	McDonald's	211.69
18	MRK	Merck	85.00
19	MSFT	Microsoft	138.12
20	NKE	Nike	93.07
21	PFE	Pfizer	35.93
22	PG	Procter & Gamble	124.00
23	TRV	Travelers Companies Inc	144.96
24	UTX	United Technologies	133.21
25	UNH	UnitedHealth	219.80
26	VZ	Verizon	59.90
27	V	Visa	175.98
28	WMT	Wal-Mart	118.16
29	WBA	Walgreen	52.97

选择方式

- 选择行
- 选择列
- 选择区域
- 筛选（条件选择）

	close	high	low	open	volume
2018-10-19	106.730003	107.550003	104.059998	104.059998	5726300
2018-10-22	104.510002	106.959999	104.449997	106.610001	5003100
2018-10-23	104.379997	104.519997	101.839996	102.410004	4223800
2018-10-24	101.839996	104.949997	101.510002	104.430000	4056700
2018-10-25	103.599998	104.169998	101.800003	102.480003	3378900
2018-10-26	101.250000	102.660004	100.139999	102.540001	5395700
2018-10-29	101.190002	103.250000	100.040001	102.470001	4238700
2018-10-30	102.080002	102.389999	100.410004	101.599998	3778200
2018-10-31	102.730003	103.709999	102.550003	103.059998	4511300
2018-11-01	104.040001	104.269997	103.019997	103.260002	2786800
2018-11-02	103.709999	105.050003	102.889999	104.930000	4322200
2018-11-05	105.209999	105.400002	103.800003	104.040001	2697700
2018-11-06	104.980003	105.660004	104.370003	104.980003	2856000
2018-11-07	107.309998	107.480003	104.900002	105.730003	3606900
2018-11-08	108.500000	108.629997	107.029999	107.029999	2896700
2018-11-09	108.279999	109.330002	107.349998	108.379997	4444000
2018-11-12	106.489998	108.440002	106.300003	108.160004	3154600
2018-11-13	107.860001	108.199997	106.470001	106.650002	3021800
2018-11-14	107.769997	109.330002	106.889999	108.610001	4978100
2018-11-15	109.599998	109.699997	106.339996	106.680000	3742600

DataFrame数据选择-选择行

```
>>> df
      姓名  语文  数学  英语  总分
a     陈纯  88  87  85  260
b  方小磊  93  88  90  271
c     王妤  82  99  96  277
d  彭子晖  97  94  84  275
e  丁海斌  97  94  76  267
```

- **选择行**
  - 索引
  - 切片
  - 专门的方法

Source

```
>>> df['a': 'c']
>>> df[0: 3]
>>> df.head(3)
```

DataFrame数据选择-选择列

```
>>> df
    姓名  语文  数学  英语  总分
a   陈纯  88   87   85   260
b  方小磊  93   88   90   271
c   王妤  82   99   96   277
d  彭子晖  97   94   84   275
e  丁海斌  97   94   76   267
```

- **选择列**
  - 列名

Source

```
>>> df['姓名']
>>> df.姓名
```

df['姓名', '语文'] ×
df['语文': '英语'] ×
df[['姓名', '语文']] √
df[['语文': '英语']] ×

DataFrame数据选择-选择区域

```
>>> df
    姓名  语文  数学  英语  总分
a   陈纯   88   87   85  260
b 方小磊   93   88   90  271
c   王妤   82   99   96  277
d 彭子晖   97   94   84  275
e 丁海斌   97   94   76  267
```

- **选择区域**
  - 标签（loc）
  - 位置（iloc）

Source

```
>>> df.loc['b': 'd', '语文': '英语']
>>> df.iloc[1: 4, 1: 4]
```

DataFrame数据选择-选择区域

```
>>> df
     姓名  语文  数学  英语  总分
a    陈纯  88    87    85    260
b  方小磊  93    88    90    271
c    王妤  82    99    96    277
d  彭子晖  97    94    84    275
e  丁海斌  97    94    76    267
```

- **选择区域-行或列**
  - 标签（loc）
  - 位置（iloc）

Source

```
>>> df.loc['a': 'c',]
>>> df.loc[:, ['语文', '数学']]
>>> df.iloc[:, [1, 2, 3]]
```

DataFrame数据选择-选择区域

```
>>> df
    姓名  语文  数学  英语  总分
a   陈纯   88   87   85   260
b  方小磊   93   88   90   271
c   王妤   82   99   96   277
d  彭子晖   97   94   84   275
e  丁海斌   97   94   76   267
```

- **选择区域-单个值**
  - 标签（loc或at)
  - 位置（iloc或iat)

Source

```
>>> df.at['b', '数学']
>>> df.iat[1, 2]
```

ix-选择区域

- ix 不推荐使用
  - loc和iloc的混合

```
>>> df
    姓名  语文  数学  英语  总分
a   陈纯    88    87    85   260
b  方小磊    93    88    90   271
c   王妤    82    99    96   277
d  彭子晖    97    94    84   275
e  丁海斌    97    94    76   267
```

Source

```
>>> df.ix[:, '语文': '英语']
   语文  数学  英语
a   88    87    85
b   93    88    90
c   82    99    96
d   97    94    84
e   97    94    76
```

df.ix[:,1:4]

DataFrame数据选择-条件筛选

```
>>> df
     姓名  语文  数学  英语  总分
a    陈纯  88  87  85  260
b  方小磊  93  88  90  271
c    王妤  82  99  96  277
d  彭子晖  97  94  84  275
e  丁海斌  97  94  76  267
```

找出索引值在'b'~'d'之间（包括'b'和'd'）并且数学成绩大于等于90的学生记录

Source

```
>>> df[(df.index >= 'b') & (df.index <= 'd') & (df.数学 >= 90)]
```

DataFrame数据选择-条件筛选

```
>>> df
      姓名  语文  数学  英语   总分
a     陈纯   88    87    85   260
b  方小磊   93    88    90   271
c     王妤   82    99    96   277
d  彭子晖   97    94    84   275
e  丁海斌   97    94    76   267
```

1. 查找陈纯和彭子晖的成绩记录；
2. 查找所有陈姓同学的成绩记录

Source

```
>>> df[df.姓名.isin(['陈纯', '彭子晖'])]
>>> df[df.姓名.str.contains('陈')]
```

人工智能程序设计

3 基于SERIES和DATAFRAME的数据统计和分析

数据统计与分析

Source

```
import pandas as pd
>>> dir(pd.Series)
[..., 'head', ..., 'index', ..., 'stack', 'std', ..., 'where', ...]
>>> dir(pd.DataFrame)
[..., 'head', ..., 'index', ..., 'stack', 'std', ..., 'to_csv', ...]
```

数据统计与分析-简单统计

```
>>> df
     姓名  语文  数学  英语  总分
a    陈纯   88   87   85   260
b  方小磊   93   88   90   271
c    王妤   82   99   96   277
d  彭子晖   97   94   84   275
e  丁海斌   97   94   76   267
```

Source

```
>>> df.mean()
语文    91.4
数学    92.4
英语    86.2
总分   270.0
dtype: float64
>>> df.数学.mean()
92.4
```

数据统计与分析-排序

Source

```
>>> df.sort_values(by = '总分')
    姓名  语文  数学  英语  总分
a   陈纯    88    87    85   260
e  丁海斌    97    94    76   267
b  方小磊    93    88    90   271
d  彭子晖    97    94    84   275
c   王妤    82    99    96   277
>>> df.sort_values(by = '总分')[:3].姓名
a   陈纯
e   丁海斌
b   方小磊
Name: 姓名, dtype: object
```

sort_index()

数据统计与分析-简单统计与筛选

统计数学成绩大于等于90的学生每门课程（包括总分）的平均值

统计总分大于等于270的学生人数

Source

```
>>> df[df.数学 >= 90].mean()
语文    92.000000
数学    95.666667
英语    85.333333
总分   273.000000
dtype: float64
>>> len(df[df.总分 >= 270])
3
```

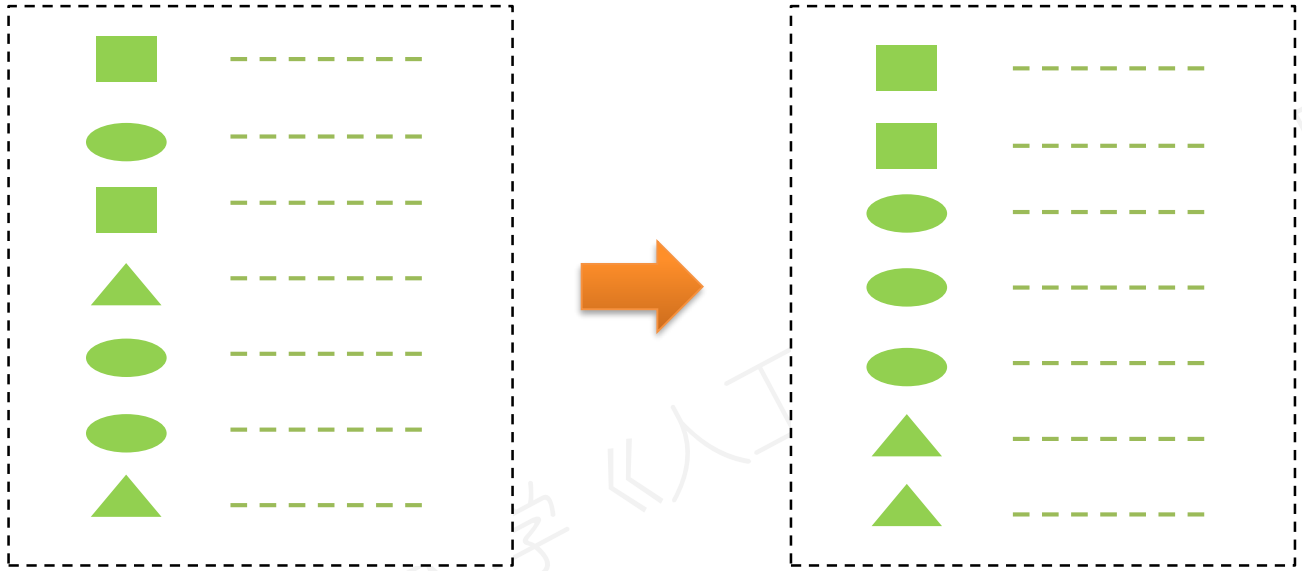
数据统计与分析-简单统计与筛选

按总分是否大于等于270为界将等级分为A和B两级

Source

```
>>> mark = ['A' if item >= 270 else 'B' for item in df.总分]
>>> df['等级'] = mark
>>> df
     姓名  语文  数学  英语   总分 等级
a    陈纯    88    87    85    260    B
b  方小磊    93    88    90    271    A
...
>>> df.groupby('等级').姓名.count()  # 或groupby(mark)
等级
A   3
B   2
Name: 姓名, dtype: int64
```

分组



Grouping的顺序

① Splitting

② Applying

③ Combining

Python财经数据接口包Tushare

http://tushare.org

数据统计与分析-简单统计与筛选

统计股票(代码600068)2019年上半年每个月的股票开盘天数

Source

```
>>> import tushare as ts
>>> df = ts.get_hist_data('600068', ...)
>>> # 条件筛选
>>> month = [...]
>>> df.groupby(month).open.count()
```

数据统计与分析-groupby&apply

apply方法可对DataFrame对象进行操作，既可作用于行或列，也可作用于Series的每一个元素上

Source

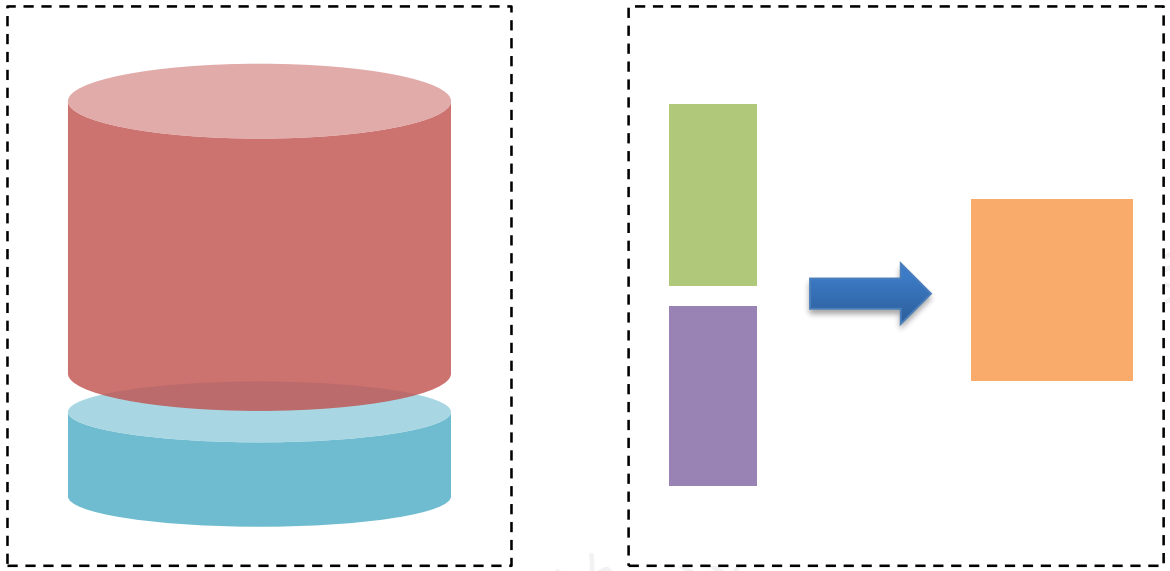
```
>>> month = [...]
>>> df.groupby(month).apply(len)
>>> df.groupby(month).close.mean()
>>> df.groupby(month).close.apply(np.mean)
>>> df.groupby(month)['open', 'close'].mean()
```

多函数聚合agg([m1, m2])方法

apply方法

```
>>> df.max()
>>> df.max(axis = 1)
>>> df.loc[:, ['close', 'open']].astype(int)
>>> df.apply(max)
>>> df.apply(max, axis = 1)
>>> df.loc[:, ['close', 'open']].apply(np.int32)
>>> df.loc[:, ['close', 'open']].apply(int)
>>> df.loc[:, ['close', 'open']].applymap(int)
>>> df.volume.apply(int)
>>> df.loc[:, ['close', 'open']]= df.loc[:, ['close', 'open']].apply(np.int32)
```

合并



Merge的形式

- Append
  - 加行到DataFrame
- Concat
  - 连接pandas对象
- Join
  - SQL类型的连接

Join

code	name
AXP	
KO	

volume	code	month
	AXP	
	AXP	
	KO	
	KO	

code	name	volume	month
AXP			
AXP			
KO			
KO			

Join

```
SUMMARY & USAGE LICENSE
================================================

MovieLens data sets were collected by the GroupLens Research Project
at the University of Minnesota.

This data set consists of:
    * 100,000 ratings (1-5) from 943 users on 1682 movies.
    * Each user has rated at least 20 movies.
        * Simple demographic info for the users (age, gender, occupation, zip)

The data was collected through the MovieLens web site
(movielens.umn.edu) during the seven-month period from September 19th,
1997 through April 22nd, 1998. This data has been cleaned up - users
who had less than 20 ratings or did not have complete demographic
information were removed from this data set. Detailed descriptions of
the data file can be found at the end of this file.

Neither the University of Minnesota nor any of the researchers
involved can guarantee the correctness of the data, its suitability
for any particular purpose, or the validity of results based on the
use of the data set.  The data set may be used for any research
purposes under the following conditions:

     * The user may not state or imply any endorsement from the
       University of Minnesota or the GroupLens Research Group.

     * The user must acknowledge the use of the data set in
       publications resulting from the use of the data set
       (see below for citation information).

     * The user may not redistribute the data without separate
       permission.

     * The user may not use this information for any commercial or
       revenue-bearing purposes without first obtaining permission
       from a faculty member of the GroupLens Research Project at the
       University of Minnesota.

If you have any further questions or comments, please contact GroupLens
```

基于pandas的男女电影评分差异分析

MovieLens data sets

http://files.grouplens.org/datasets/movielens/ml-100k.zip

Join

```
unames = ['user id', 'age', 'gender', 'occupation', 'zip code']
users = pd.read_csv('ml-100k/u.user', sep = '|', names = unames)
rnames = ['user id', 'item id', 'rating', 'timestamp']
ratings = pd.read_csv('ml-100k/u.data', sep = '\t', names = rnames)
users_df = users.loc[:, ['user id', 'gender']]
ratings_df = ratings.loc[:, ['user id', 'rating']]
rating_df = pd.merge(users_df, ratings_df)
```

```
# Way 1 - groupby()
result = rating_df.groupby('gender').rating.apply(pd.Series.std)
print(result)

# Way 2 - groupby()
df_temp = rating_df.groupby(['user id', 'gender']).apply(np.mean)
result = df_temp.groupby('gender').rating.apply(pd.Series.std)
print(result)
# Way 2 - pivot_table()
gender_table = pd.pivot_table(rating_df, index = ['gender', 'user id'], values = 'rating')
Female_df = gender_table.query("gender == ['F']")
Male_df = gender_table.query("gender == ['M']")
Female_std = pd.Series.std(Female_df)
Male_std = pd.Series.std(Male_df)
print('Gender', '\nF\t%.6f' % Female_std, '\nM\t%.6f' % Male_std)
```

数据统计与分析—数据描述

>>> df.info()

```
<class 'pandas.core.frame.DataFrame'>
Index: 610 entries, 2020-04-24 to 2017-10-25
Data columns (total 13 columns):
open            610 non-null float64
high            610 non-null float64
close           610 non-null float64
low             610 non-null float64
volume          610 non-null float64
price_change    610 non-null float64
p_change        610 non-null float64
ma5             610 non-null float64
ma10            610 non-null float64
ma20            610 non-null float64
v_ma5           610 non-null float64
v_ma10          610 non-null float64
v_ma20          610 non-null float64
dtypes: float64(13)
memory usage: 66.7+ KB
```

>>> df.describe()

	open	high	close	
count	610.000000	610.000000	610.000000	6
mean	7.091738	7.185033	7.092311	
std	1.131810	1.149434	1.124954	
min	5.300000	5.410000	5.400000	
25%	6.220000	6.282500	6.220000	
50%	6.735000	6.820000	6.735000	
75%	7.800000	7.950000	7.817500	
max	10.340000	10.460000	10.340000	

[8 rows x 13 columns]

数据统计与分析—相关分析

皮尔逊(Pearson)

$$r_{xy} = \frac{\sum (X-\bar{X})(Y-\bar{Y})}{(\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2})(\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2})}$$

约束条件：

1. 两个变量间有线性关系
2. 均是连续变量
3. 变量均符合正态分布,且二元分布也符合正态分布
4. 两个变量独立

- [维基百科] 假设五个国家的国民生产总值分别是1、2、3、5、8（单位10亿美元），又假设这五个国家的贫困比例分别是11%、12%、13%、15%、18%。

x均值：3.8

y均值：0.138

(1-3.8)*(0.11-0.138)=0.0784

(2-3.8)*(0.12-0.138)=0.0324

(3-3.8)*(0.13-0.138)=0.0064

(5-3.8)*(0.15-0.138)=0.0144

(8-3.8)*(0.18-0.138)=0.1764

0.0784+0.0324+0.0064+0.0144+0.1764=0.308

(1-3.8)^2=7.84

(2-3.8)^2=3.24

(3-3.8)^2=0.64

(5-3.8)^2=1.44

(8-3.8)^2=17.64

7.84+3.24+0.64+1.44+17.64=30.8

30.8^0.5=5.549775

0.00308^0.5=0.05549775

0.308/(5.549775*0.05549775)=1

皮尔逊(Pearson)相关分析

$$\frac{\sum(X-\bar{X})(Y-\bar{Y})}{(\sqrt{\sum_{i=1}^{n}(X_i-\bar{X})^2})(\sqrt{\sum_{i=1}^{n}(Y_i-\bar{Y})^2})}$$

```
x = np.array([1,2,3,5,8])
y = np.array([0.11,0.12,0.13,0.15,0.18])
x_mean = np.mean(x)
y_mean = np.mean(y)
```

```
u = np.sum((x-x_mean)*(y-y_mean))
l = np.sqrt(np.sum((x-x_mean)**2))*np.sqrt(np.sum((y-y_mean)**2))
print(u/l)
```

皮尔逊(Pearson)相关分析

```
import pandas as pd

x = [1,2,3,5,8]
y = [0.11,0.12,0.13,0.15,0.18]
df = pd.DataFrame()
df['x'] = x
df['y'] = y
print(df.corr())
```