pandas数据处理120题

1.DataFrame基本操作

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

1.将下面的字典创建为DataFrame

```
data = {"grammer":["Python","C","Java","GO",np.nan,"SQL","PHP","Python"],
        "score":[1,2,np.nan,4,5,6,7,10]}
```

```
df=pd.DataFrame(data)
df
```

	grammer	score
0	Python	1.0
1	C	2.0
2	Java	NaN
3	GO	4.0
4	NaN	5.0
5	SQL	6.0
6	PHP	7.0
7	Python	10.0

2.提取数据–提取含有字符串“Python”的行

```
# 方法一:
df[df['grammer']=='Python']
```

	grammer	score
0	Python	1.0
7	Python	10.0

```
# 方法二:
results = df['grammer'].str.contains('Python')
results.fillna(value=False,inplace=True)
df[results]
```

	grammer	score
0	Python	1.0
7	Python	10.0

3. 提取列名

```
# 输出df所有的列
df.columns
```

```
Index(['grammer', 'score'], dtype='object')
```

4.修改第二列列名为‘popularity’

```
# 修改第二列列名为'popularity'
df.rename(columns={'score':'popularity'},inplace=True)
```

```
df
```

	grammer	popularity
0	Python	1.0
1	C	2.0
2	Java	NaN
3	GO	4.0
4	NaN	5.0
5	SQL	6.0
6	PHP	7.0
7	Python	10.0

5.统计grammer列中每种编程语言出现的次数

```
# 统计grammer列中每种编程语言出现的次数
df['grammer'].value_counts()
```

```
Python    2
C         1
Java      1
PHP       1
GO        1
SQL       1
Name: grammer, dtype: int64
```

```
# 统计一共出现多少个不重复的编程语言
# df['grammer'].nunique()
```

```
6
```

6.缺失值用上下值均值填充

```
#缺失值用上下值均值填充
df['popularity']=df['popularity'].fillna(df['popularity'].interpolate())
```

```
df
```

	grammer	popularity
0	Python	1.0
1	C	2.0
2	Java	3.0
3	GO	4.0
4	NaN	5.0
5	SQL	6.0
6	PHP	7.0
7	Python	10.0

7.提取popularity列中值大于3的行

```
#提取popularity列中值大于3的行
df[df['popularity']>3]
```

	grammer	popularity
3	GO	4.0
4	NaN	5.0
5	SQL	6.0
6	PHP	7.0
7	Python	10.0

8.按照grammer列进行去重

```
# 按照grammer列进行去重
df.drop_duplicates(['grammer'])
```

	grammer	popularity
0	Python	1.0
1	C	2.0
2	Java	3.0
3	GO	4.0
4	NaN	5.0
5	SQL	6.0
6	PHP	7.0

9.计算popularity列平均值

```
# 计算popularity列平均值
df['popularity'].mean()
```

```
4.75
```

10.将grammer列转换为list

```
#将grammer列转换为list
df['grammer'].to_list()
```

```
['Python', 'C', 'Java', 'GO', nan, 'SQL', 'PHP', 'Python']
```

11.将dataframe保存为excel

```
#将dataframe保存为excel
df.to_excel('f1.xlsx')
```

12.查看数据行列数 形状

```
# 查看数据行列数 形状
df.shape
```

```
(8, 2)
```

13.提取popularity列值大于3小于7的行

```
# 提取popularity列值大于3小于7的行
df[(df['popularity']>3) & (df['popularity']<7)]
```

	grammer	popularity
3	GO	4.0
4	NaN	5.0
5	SQL	6.0

14.交换两列的位置

```
## 交换两列的位置
#方法一:
df=df[['popularity','grammer']]
```

```
temp = df['popularity']
df.drop(labels=['popularity'], axis=1,inplace = True)
df.insert(0, 'popularity', temp)
```

```
df
```

	popularity	grammer
0	1.0	Python
1	2.0	C
2	3.0	Java
3	4.0	GO
4	5.0	NaN
5	6.0	SQL
6	7.0	PHP
7	10.0	Python

15.提取popularity列最大值所在的行

```
# 提取popularity列最大值所在的行
df[df['popularity']==df['popularity'].max()]
```

	popularity	grammer
7	10.0	Python

16.查看数据最后5行

```
# 查看数据最后5行
df.tail()
```

	popularity	grammer
3	4.0	GO
4	5.0	NaN
5	6.0	SQL
6	7.0	PHP
7	10.0	Python

17. 删除最后一行数据

```
# 删除最后一行数据
df=df.drop(labels=df.shape[0]-1)
```

18. 添加一行数据['Perl',6.6]

```
##  添加一行数据['Perl',6.6]
row = {'grammer':'Perl','popularity':6.6}
df = df.append(row,ignore_index=True)
```

19.对数据按照"popularity"列值的大小进行排序

```
# 对数据按照"popularity"列值的大小进行排序
df.sort_values("popularity",inplace=True)
```

```
df
```

	popularity	grammer
0	1.0	Python
1	2.0	C
2	3.0	Java
3	4.0	GO
4	5.0	NaN
5	6.0	SQL
7	6.6	Perl
6	7.0	PHP

20.统计grammer列每个字符串的长度

```
df['grammer']=df['grammer'].fillna('R')
df['len_str']=df['grammer'].map(lambda x:len(x))
df
```

	popularity	grammer	len_str
0	1.0	Python	6
1	2.0	C	1
2	3.0	Java	4
3	4.0	GO	2
4	5.0	R	1
5	6.0	SQL	3
7	6.6	Perl	4
6	7.0	PHP	3

2. Pandas数据处理

21.读取本地EXCEL数据

```
df = pd.read_excel('pandas120/21-50数据.xlsx')
```

22.查看df数据前5行

```
df.head()
```

	createTime	education	salary
0	2020-03-16 11:30:18	本科	20k-35k
1	2020-03-16 10:58:48	本科	20k-40k
2	2020-03-16 10:46:39	不限	20k-35k
3	2020-03-16 10:45:44	本科	13k-20k
4	2020-03-16 10:20:41	本科	10k-20k

23.将salary列数据转换为最大值与最小值的平均值

```
#因为我们的数据中是20k-35k这种字符串，所以需要先用正则表达式提取数字
import re
# 方法一：apply + 自定义函数
def func(df):
    lst = df['salary'].split('-')
    smin = int(lst[0].strip('k'))
    smax = int(lst[1].strip('k'))
    df['salary'] = int((smin + smax) / 2 * 1000)
    return df

df = df.apply(func,axis=1)
```

```
df
```

	createTime	education	salary
0	2020-03-16 11:30:18	本科	27500
1	2020-03-16 10:58:48	本科	30000
2	2020-03-16 10:46:39	不限	27500
3	2020-03-16 10:45:44	本科	16500
4	2020-03-16 10:20:41	本科	15000
...	...	...	...
130	2020-03-16 11:36:07	本科	14000
131	2020-03-16 09:54:47	硕士	37500

	createTime	education	salary
132	2020-03-16 10:48:32	本科	30000
133	2020-03-16 10:46:31	本科	19000
134	2020-03-16 11:19:38	本科	30000

135 rows × 3 columns

```
# 方法二：iterrows + 正则
import re
for index,row in df.iterrows():
    nums = re.findall('\d+',row[2])
    df.iloc[index,2] = int(eval(f'({nums[0]} + {nums[1]}) / 2 * 1000'))
```

24.将数据根据学历进行分组并计算平均薪资

```
df.groupby('education').mean()
```

	salary
education	
不限	19600.000000
大专	10000.000000
本科	19361.344538
硕士	20642.857143

25.将createTime列时间转换为月-日

```
for i in range(len(df)):
    df.ix[i,0] = df.ix[i,0].to_pydatetime().strftime("%m-%d")
df.head()
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecated

D:\Anaconda3\lib\site-packages\pandas\core\indexing.py:961: FutureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecated
  return getattr(section, self.name)[new_key]
```

	createTime	education	salary
0	03-16	本科	27500
1	03-16	本科	30000
2	03-16	不限	27500
3	03-16	本科	16500
4	03-16	本科	15000

```
# iloc 同ix
for i in range(len(df)):
    df.iloc[i,0] = df.iloc[i,0].to_pydatetime().strftime("%m-%d")
df.head()
```

	createTime	education	salary
0	03-16	本科	20k-35k
1	03-16	本科	20k-40k
2	03-16	不限	20k-35k
3	03-16	本科	13k-20k
4	03-16	本科	10k-20k

26.查看索引、数据类型和内存信息

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 135 entries, 0 to 134
Data columns (total 3 columns):
createTime    135 non-null object
education     135 non-null object
salary        135 non-null int64
dtypes: int64(1), object(2)
memory usage: 3.3+ KB
```

27.查看数值型列的汇总统计

```
df.describe()
```

	salary
count	135.000000
mean	19159.259259
std	8661.686922
min	3500.000000
25%	14000.000000
50%	17500.000000
75%	25000.000000
max	45000.000000

28.新增一列根据**salary**将数据分为三组

```
bins = [0,5000, 20000, 50000]
group_names = ['低', '中', '高']
df['categories'] = pd.cut(df['salary'], bins, labels=group_names)
df
```

	createTime	education	salary	categories
0	03-16	本科	27500	高
1	03-16	本科	30000	高
2	03-16	不限	27500	高
3	03-16	本科	16500	中
4	03-16	本科	15000	中
...	...	...	...	...
130	03-16	本科	14000	中
131	03-16	硕士	37500	高
132	03-16	本科	30000	高
133	03-16	本科	19000	中
134	03-16	本科	30000	高

135 rows × 4 columns

29.按照**salary**列对数据降序排列

```
df.sort_values('salary', ascending=False)
```

	createTime	education	salary	categories
53	03-16	本科	45000	高
37	03-16	本科	40000	高
101	03-16	本科	37500	高
16	03-16	本科	37500	高
131	03-16	硕士	37500	高
...	...	...	...	...
123	03-16	本科	4500	低
126	03-16	本科	4000	低
110	03-16	本科	4000	低
96	03-16	不限	3500	低
113	03-16	本科	3500	低

135 rows × 4 columns

30.取出第**33**行数据

```
df.loc[32]
```

```
createTime    03-16
education       硕士
salary         22500
categories         高
Name: 32, dtype: object
```

31.计算**salary**列的中位数

```
np.median(df['salary'])
```

```
17500.0
```

32.绘制薪资水平频率分布直方图

```
df.salary.plot(kind='hist');
```

33.绘制薪资水平密度曲线

```
df.salary.plot(kind='kde',xlim=(0,80000));
```

34.删除最后一列categories

```
del df['categories']
# 等价于
# df.drop(columns=['categories'], inplace=True)
```

35.将df的第一列与第二列合并为新的一列

```
df['test'] = df['education']+df['createTime']
df
```

	createTime	education	salary	test
0	03-16	本科	27500	本科03-16
1	03-16	本科	30000	本科03-16
2	03-16	不限	27500	不限03-16
3	03-16	本科	16500	本科03-16
4	03-16	本科	15000	本科03-16
...	...	...	...	...
130	03-16	本科	14000	本科03-16
131	03-16	硕士	37500	硕士03-16
132	03-16	本科	30000	本科03-16
133	03-16	本科	19000	本科03-16
134	03-16	本科	30000	本科03-16

135 rows × 4 columns

36.将education列与salary列合并为新的一列

```
# salary为int类型，操作与35题有所不同
df["test1"] = df["salary"].map(str) + df['education']
df
```

	createTime	education	salary	test	test1
0	03-16	本科	27500	本科03-16	27500本科
1	03-16	本科	30000	本科03-16	30000本科
2	03-16	不限	27500	不限03-16	27500不限
3	03-16	本科	16500	本科03-16	16500本科
4	03-16	本科	15000	本科03-16	15000本科
...	...	...	...	...	...
130	03-16	本科	14000	本科03-16	14000本科
131	03-16	硕士	37500	硕士03-16	37500硕士
132	03-16	本科	30000	本科03-16	30000本科
133	03-16	本科	19000	本科03-16	19000本科
134	03-16	本科	30000	本科03-16	30000本科

135 rows × 5 columns

37.计算salary最大值与最小值之差

```
df[['salary']].apply(lambda x: x.max() - x.min())
```

```
salary    41500
dtype: int64
```

38.将第一行与最后一行拼接

```
pd.concat([df[:1], df[-2:-1]])
```

	createTime	education	salary	test	test1
0	03-16	本科	27500	本科03-16	27500本科

133	03-16 createTime	本科 education	19000 salary	本科03-16 test	19000本科 test1

39.将第**8**行数据添加至末尾¶

```
df.append(df.iloc[7])
```

	createTime	education	salary	test	test1
0	03-16	本科	27500	本科03-16	27500本科
1	03-16	本科	30000	本科03-16	30000本科
2	03-16	不限	27500	不限03-16	27500不限
3	03-16	本科	16500	本科03-16	16500本科
4	03-16	本科	15000	本科03-16	15000本科
...	...	...	...	...	...
131	03-16	硕士	37500	硕士03-16	37500硕士
132	03-16	本科	30000	本科03-16	30000本科
133	03-16	本科	19000	本科03-16	19000本科
134	03-16	本科	30000	本科03-16	30000本科
7	03-16	本科	12500	本科03-16	12500本科

136 rows × 5 columns

40.查看每列的数据类型¶

```
df.dtypes
```

```
createTime    object
education     object
salary         int64
test          object
test1         object
dtype: object
```

41.将**createTime**列设置为索引

```
df.set_index("createTime")
```

	education	salary	test	test1
createTime				
03-16	本科	27500	本科03-16	27500本科
03-16	本科	30000	本科03-16	30000本科
03-16	不限	27500	不限03-16	27500不限
03-16	本科	16500	本科03-16	16500本科
03-16	本科	15000	本科03-16	15000本科
...	...	...	...	...
03-16	本科	14000	本科03-16	14000本科
03-16	硕士	37500	硕士03-16	37500硕士
03-16	本科	30000	本科03-16	30000本科
03-16	本科	19000	本科03-16	19000本科
03-16	本科	30000	本科03-16	30000本科

135 rows × 4 columns

42.生成一个和**df**长度相同的随机数**dataframe**

```
df1 = pd.DataFrame(pd.Series(np.random.randint(1, 10,len(df))))
df1
```

	0
0	4
1	1
2	5
3	4
4	8
...	...
130	5
131	4
132	4

133	4
134	6

135 rows × 1 columns

43.将上一题生成的**dataframe**与**df**合并

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

	createTime	education	salary	test	test1	0
0	03-16	本科	27500	本科03-16	27500本科	4
1	03-16	本科	30000	本科03-16	30000本科	1
2	03-16	不限	27500	不限03-16	27500不限	5
3	03-16	本科	16500	本科03-16	16500本科	4
4	03-16	本科	15000	本科03-16	15000本科	8
...	...	...	...	...	...	...
130	03-16	本科	14000	本科03-16	14000本科	5
131	03-16	硕士	37500	硕士03-16	37500硕士	4
132	03-16	本科	30000	本科03-16	30000本科	4
133	03-16	本科	19000	本科03-16	19000本科	4
134	03-16	本科	30000	本科03-16	30000本科	6

135 rows × 6 columns

44.生成新的一列**new**为**salary**列减去之前生成随机数列

```
df["new"] = df["salary"] - df[0]
df
```

	createTime	education	salary	test	test1	0	new
0	03-16	本科	27500	本科03-16	27500本科	4	27496
1	03-16	本科	30000	本科03-16	30000本科	1	29999
2	03-16	不限	27500	不限03-16	27500不限	5	27495
3	03-16	本科	16500	本科03-16	16500本科	4	16496
4	03-16	本科	15000	本科03-16	15000本科	8	14992
...	...	...	...	...	...	...	...
130	03-16	本科	14000	本科03-16	14000本科	5	13995
131	03-16	硕士	37500	硕士03-16	37500硕士	4	37496
132	03-16	本科	30000	本科03-16	30000本科	4	29996
133	03-16	本科	19000	本科03-16	19000本科	4	18996
134	03-16	本科	30000	本科03-16	30000本科	6	29994

135 rows × 7 columns

45.检查数据中是否含有任何缺失值

```
df.isnull().values.any()
```

```
False
```

46.将**salary**列类型转换为浮点数

```
df['salary'].astype(np.float64)
```

```
0     27500.0
1     30000.0
2     27500.0
3     16500.0
4     15000.0
       ...
130   14000.0
131   37500.0
132   30000.0
133   19000.0
134   30000.0
Name: salary, Length: 135, dtype: float64
```

47.计算**salary**大于**10000**的次数

```
len(df[df.salary>10000])
```

```
119
```

48.查看每种学历出现的次数

```
df.education.value_counts()
```

```
本科    119
硕士     7
不限     5
大专     4
Name: education, dtype: int64
```

49.查看education列共有几种学历¶

```
df.education.nunique()
```

```
4
```

50.提取salary与new列的和大于60000的最后3行

```
# 方法一:
df[(df.salary+df.new>60000)].tail(3)
```

	createTime	education	salary	test	test1	0	new
92	03-16	本科	35000	本科03-16	35000本科	8	34992
101	03-16	本科	37500	本科03-16	37500本科	2	37498
131	03-16	硕士	37500	硕士03-16	37500硕士	4	37496

```
#方法二:
df1 = df[['salary','new']]
rowsums = df1.apply(np.sum, axis=1)
res = df.iloc[np.where(rowsums > 60000)[0][-3:], :]
res
```

	createTime	education	salary	test	test1	0	new
92	03-16	本科	35000	本科03-16	35000本科	8	34992
101	03-16	本科	37500	本科03-16	37500本科	2	37498
131	03-16	硕士	37500	硕士03-16	37500硕士	4	37496

3.金融数据处理

51.使用绝对路径读取本地Excel数据

```
data = pd.read_excel("pandas120/51-80数据.xls")
```

52.查看数据前三行

```
data.head(3)
```

	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流(股
0	600000.SH	浦发银行	2016-01-04	16.1356	16.1444	16.1444	15.4997	15.7205	42240610	754425783	-0.4151	-2.5725	17.8602	0.2264	3.320318e+11	3.320318e+11	1.8653
1	600000.SH	浦发银行	2016-01-05	15.7205	15.4644	15.9501	15.3672	15.8618	58054793	1034181474	0.1413	0.8989	17.8139	0.3112	3.350163e+11	3.350163e+11	1.8653
2	600000.SH	浦发银行	2016-01-06	15.8618	15.8088	16.0208	15.6234	15.9855	46772653	838667398	0.1236	0.7795	17.9307	0.2507	3.376278e+11	3.376278e+11	1.8653

53.查看每列数据缺失值情况

```
# 每一列的缺失个数
data.isnull().sum()
```

```
代码          1
简称          2
日期          2
前收盘价(元)      2
开盘价(元)       2
最高价(元)       2
最低价(元)       2
收盘价(元)       2
成交量(股)       2
成交金额(元)      2
涨跌(元)        2
涨跌幅(%)       2
均价(元)        2
换手率(%)       2
A股流通市值(元)    2
总市值(元)       2
A股流通股本(股)    2
市盈率          2
dtype: int64
```

54.提取日期列含有空值的行

```
data[data['日期'].isnull()]
```

	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)	市盈率
327	NaN	NaN	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
328	数据来源：Wind资讯	NaN	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

55.输出每列缺失值具体行数

```
for columname in data.columns:
    if data[columname].count() != len(data):
        loc = data[columname][data[columname].isnull().values==True].index.tolist()
        print('列名："{}", 第{}行位置有缺失值'.format(columname,loc))
```

```
列名："代码", 第[327]行位置有缺失值
列名："简称", 第[327, 328]行位置有缺失值
列名："日期", 第[327, 328]行位置有缺失值
列名："前收盘价(元)", 第[327, 328]行位置有缺失值
列名："开盘价(元)", 第[327, 328]行位置有缺失值
列名："最高价(元)", 第[327, 328]行位置有缺失值
列名："最低价(元)", 第[327, 328]行位置有缺失值
列名："收盘价(元)", 第[327, 328]行位置有缺失值
列名："成交量(股)", 第[327, 328]行位置有缺失值
列名："成交金额(元)", 第[327, 328]行位置有缺失值
列名："涨跌(元)", 第[327, 328]行位置有缺失值
列名："涨跌幅(%)", 第[327, 328]行位置有缺失值
列名："均价(元)", 第[327, 328]行位置有缺失值
列名："换手率(%)", 第[327, 328]行位置有缺失值
列名："A股流通市值(元)", 第[327, 328]行位置有缺失值
列名："总市值(元)", 第[327, 328]行位置有缺失值
列名："A股流通股本(股)", 第[327, 328]行位置有缺失值
列名："市盈率", 第[327, 328]行位置有缺失值
```

56.删除所有存在缺失值的行

```
'''
备注
axis：0-行操作（默认），1-列操作
how：any-只要有空值就删除（默认），all-全部为空值才删除
inplace：False-返回新的数据集（默认），True-在原数据集上操作
'''
data.dropna(axis=0, how='any', inplace=True)
```

```
data.isnull().sum()
```

```
代码          0
简称          0
日期          0
前收盘价(元)      0
开盘价(元)       0
最高价(元)       0
最低价(元)       0
收盘价(元)       0
成交量(股)       0
成交金额(元)      0
涨跌(元)        0
涨跌幅(%)       0
均价(元)        0
换手率(%)       0
A股流通市值(元)    0
总市值(元)       0
A股流通股本(股)    0
市盈率          0
dtype: int64
```

57.绘制收盘价的折线图

```
plt.style.use('seaborn-darkgrid') # 设置画图的风格
plt.rc('font',  size=6) #设置图中字体和大小
plt.rc('figure', figsize=(4,3), dpi=150) # 设置图的大小
data['收盘价(元)'].plot();
```

```
plt.style.use('seaborn-darkgrid') # 设置画图的风格
plt.rc('font',size=8) #设置图中字体和大小
plt.rc('figure', figsize=(4,3), dpi=180) # 设置图的大小
# data['收盘价(元)'].plot()
plt.plot(data["收盘价(元)"]);
```

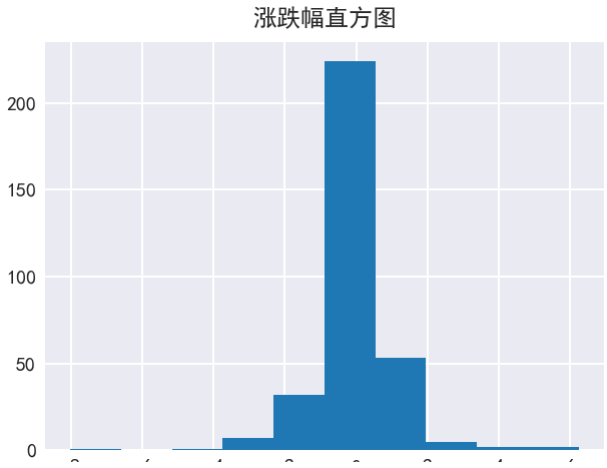
58.同时绘制开盘价与收盘价¶

```
plt.rcParams['font.sans-serif']=['Simhei']  #显示中文
plt.rcParams['axes.unicode_minus']=False    #设置显示中文后,负号显示受影响,显示负号
```

```
data[['收盘价(元)','开盘价(元)']].plot();
```

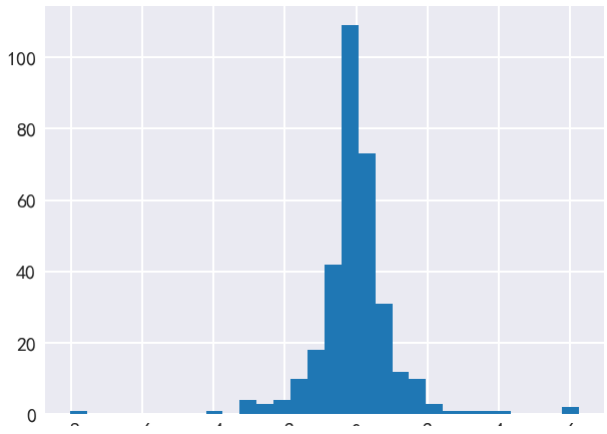
59.绘制涨跌幅的直方图

```
plt.title('涨跌幅直方图')
plt.hist(data['涨跌幅(%)']);
# 等价于
# df['涨跌幅(%)'].hist()
```



60.让直方图更细致

```
data['涨跌幅(%)'].hist(bins = 30);
```



61.以**data**的列名创建一个**dataframe**

```
temp = pd.DataFrame(columns = data.columns.to_list())
```

```
temp
```

	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)	市盈率

62.打印所有换手率不是数字的行

```
for i in range(len(data)):
    if type(data.iloc[i,13]) != float:
        temp = temp.append(data.loc[i])
temp
```

	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)	市盈率
26	600000.SH	浦发银行	2016-02-16	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
27	600000.SH	浦发银行	2016-02-17	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
28	600000.SH	浦发银行	2016-02-18	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
29	600000.SH	浦发银行	2016-02-19	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
30	600000.SH	浦发银行	2016-02-22	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
31	600000.SH	浦发银行	2016-02-23	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
32	600000.SH	浦发银行	2016-02-24	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
33	600000.SH	浦发银行	2016-02-25	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
34	600000.SH	浦发银行	2016-02-26	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
35	600000.SH	浦发银行	2016-02-29	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
36	600000.SH	浦发银行	2016-03-01	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
37	600000.SH	浦发银行	2016-03-02	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
38	600000.SH	浦发银行	2016-03-03	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
39	600000.SH	浦发银行	2016-03-04	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
40	600000.SH	浦发银行	2016-03-07	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
41	600000.SH	浦发银行	2016-03-08	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
42	600000.SH	浦发银行	2016-03-09	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
43	600000.SH	浦发	2016-	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801

	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)	市盈率
26	600000.SH	浦发银行	2016-02-16	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
27	600000.SH	浦发银行	2016-02-17	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
28	600000.SH	浦发银行	2016-02-18	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
29	600000.SH	浦发银行	2016-02-19	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
30	600000.SH	浦发银行	2016-02-22	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
31	600000.SH	浦发银行	2016-02-23	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
32	600000.SH	浦发银行	2016-02-24	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
33	600000.SH	浦发银行	2016-02-25	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
34	600000.SH	浦发银行	2016-02-26	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
35	600000.SH	浦发银行	2016-02-29	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
36	600000.SH	浦发银行	2016-03-01	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
37	600000.SH	浦发银行	2016-03-02	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
38	600000.SH	浦发银行	2016-03-03	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
39	600000.SH	浦发银行	2016-03-04	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
40	600000.SH	浦发银行	2016-03-07	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
41	600000.SH	浦发银行	2016-03-08	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
42	600000.SH	浦发银行	2016-03-09	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801

63.打印所有换手率为**--**的行

```
data[data['换手率(%)'].isin(['--'])]
```


	代码	简称	日期	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)	市盈率
26	600000.SH	浦发银行	2016-02-16	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
27	600000.SH	浦发银行	2016-02-17	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
28	600000.SH	浦发银行	2016-02-18	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
29	600000.SH	浦发银行	2016-02-19	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
30	600000.SH	浦发银行	2016-02-22	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
31	600000.SH	浦发银行	2016-02-23	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
32	600000.SH	浦发银行	2016-02-24	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
33	600000.SH	浦发银行	2016-02-25	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
34	600000.SH	浦发银行	2016-02-26	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
35	600000.SH	浦发银行	2016-02-29	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
36	600000.SH	浦发银行	2016-03-01	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
37	600000.SH	浦发银行	2016-03-02	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
38	600000.SH	浦发银行	2016-03-03	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
39	600000.SH	浦发银行	2016-03-04	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
40	600000.SH	浦发银行	2016-03-07	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
41	600000.SH	浦发银行	2016-03-08	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801
42	600000.SH	浦发银行	2016-03-09	16.2946	16.2946	16.2946	16.2946	16.2946	--	--	0.0	0.0	--	--	3.441565e+11	3.441565e+11	1.865347e+10	6.801

43	600000.SH 代码	浦发 銀称	2016-03-10 日期	16.2946 前收盘价(元)	16.2946 开盘价(元)	16.2946 最高价(元)	16.2946 最低价(元)	16.2946 收盘价(元)	成交量(股)	成交金额(元)	0.0 涨跌(元)	0.0 涨跌幅(%)	均价(元)	换手率(%)	3.441565e+11 A股流通市值(元)	3.441565e+11 总市值(元)	1.865347e+10 A股流通股本(股)	6.801 市盈率

64.重置**data**的行号

```
data=data.reset_index()
```

65.删除所有换手率为非数字的行

```
k =[]
for i in range(len(data)):
    if type(data.iloc[i,13]) != float:
        k.append(i)
data.drop(labels=k,inplace=True)
```

66.绘制换手率的密度曲线

```
data['换手率(%)'].plot(kind='kde');
```

67.计算前一天与后一天收盘价的差值

```
data['收盘价(元)'].diff()
```

```
0        NaN
1     0.1413
2     0.1237
3    -0.5211
4    -0.0177
      ...
322  -0.0800
323  -0.1000
324  -0.0600
325  -0.0600
326  -0.1000
Name: 收盘价(元), Length: 309, dtype: float64
```

68.计算前一天与后一天收盘价变化率

```
data['收盘价(元)'].pct_change()
```

```
0        NaN
1     0.008988
2     0.007799
3    -0.032598
4    -0.001145
       ...
322  -0.005277
323  -0.006631
324  -0.004005
325  -0.004021
326  -0.006729
Name: 收盘价(元), Length: 309, dtype: float64
```

69.设置日期为索引

```
data = data.set_index('日期')
```

```
data.head()
```

	index	代码	简称	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A
日期																	
2016-01-04	0	600000.SH	浦发银行	16.1356	16.1444	16.1444	15.4997	15.7205	42240610	754425783	-0.4151	-2.5725	17.8602	0.2264	3.320318e+11	3.320318e+11	1
2016-01-05	1	600000.SH	浦发银行	15.7205	15.4644	15.9501	15.3672	15.8618	58054793	1034181474	0.1413	0.8989	17.8139	0.3112	3.350163e+11	3.350163e+11	1
2016-01-06	2	600000.SH	浦发银行	15.8618	15.8088	16.0208	15.6234	15.9855	46772653	838667398	0.1236	0.7795	17.9307	0.2507	3.376278e+11	3.376278e+11	1
2016-01-07	3	600000.SH	浦发银行	15.9855	15.7205	15.8088	15.3672	15.4644	11350479	199502702	-0.5211	-3.2597	17.5766	0.0608	3.266223e+11	3.266223e+11	1
2016-01-08	4	600000.SH	浦发银行	15.4644	15.6675	15.7912	14.9345	15.4467	71918296	1262105060	-0.0177	-0.1142	17.5492	0.3855	3.262492e+11	3.262492e+11	1

70.以**5**个数据作为一个数据滑动窗口，在这个**5**个数据上取均值(收盘价)

```
data['收盘价(元)'].rolling(5).mean()
```

```
日期
2016-01-04       NaN
2016-01-05       NaN
2016-01-06       NaN
2016-01-07       NaN
2016-01-08   15.69578
                ...
2017-05-03   15.14200
2017-05-04   15.12800
2017-05-05   15.07000
2017-05-08   15.00000
2017-05-09   14.92000
Name: 收盘价(元), Length: 309, dtype: float64
```

71.以**5**个数据作为一个数据滑动窗口，计算这五个数据总和**(**收盘价**)**

```
data['收盘价(元)'].rolling(5).sum()
```

```
日期
2016-01-04       NaN
2016-01-05       NaN
2016-01-06       NaN
2016-01-07       NaN
2016-01-08   78.4789
                ...
2017-05-03   75.7100
2017-05-04   75.6400
2017-05-05   75.3500
2017-05-08   75.0000
2017-05-09   74.6000
Name: 收盘价(元), Length: 309, dtype: float64
```

72.将收盘价**5**日均线、**20**日均线与原始数据绘制在同一个图上

```
data['收盘价(元)'].plot()
data['收盘价(元)'].rolling(5).mean().plot()
data['收盘价(元)'].rolling(20).mean().plot();
```

73.按周为采样规则，取一周收盘价最大值

```
data['收盘价(元)'].resample('W').max()
```

```
日期
2016-01-10   15.9855
2016-01-17   15.8265
2016-01-24   15.6940
2016-01-31   15.0405
2016-02-07   16.2328
                ...
2017-04-16   15.9700
2017-04-23   15.5600
2017-04-30   15.2100
2017-05-07   15.1600
2017-05-14   14.8600
Freq: W-SUN, Name: 收盘价(元), Length: 71, dtype: float64
```

74.绘制重采样数据与原始数据

```
data['收盘价(元)'].plot()
data['收盘价(元)'].resample('7D').max().plot();
```

75.将数据往后移动**5**天

```
data.shift(5)
```

	index	代码	简称	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)
日期																	
2016-01-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-01-05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-01-06	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-01-07	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2016-01-08	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-05-03	317.0	600000.SH	浦发银行	15.00	15.02	15.10	14.99	15.05	12975919	195296862	0.05	0.3333	15.0507	0.06	3.253551e+11	3.253551e+11	2.161828e+10

日期	index	代码	简称	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)
2017-05-04	318.0	600000.SH	浦发银行	15.05	15.06	15.11	15.00	15.05	14939871	[illegible]	[illegible]	[illegible]	15.0619	[illegible]	[illegible]	3.253551e+11	[illegible]
2017-05-05	319.0	600000.SH	浦发银行	15.05	15.05	15.25	15.03	15.21	22887645	345791526	0.16	1.0631	15.1082	0.1059	3.288140e+11	3.288140e+11	2.161828e+10
2017-05-08	320.0	600000.SH	浦发银行	15.21	15.15	15.22	15.08	15.21	15718509	238419161	0.00	0.0000	15.1681	0.0727	3.288140e+11	3.288140e+11	2.161828e+10
2017-05-09	321.0	600000.SH	浦发银行	15.21	15.21	15.22	15.13	15.16	12607509	191225527	-0.05	-0.3287	15.1676	0.0583	3.277331e+11	3.277331e+11	2.161828e+10

309 rows × 18 columns

76.将数据向前移动5天

```
data.shift(-5)
```

	index	代码	简称	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	涨跌(元)	涨跌幅(%)	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)
日期																
2016-01-04	5.0	600000.SH	浦发银行	15.4467	15.1994	15.4114	14.9786	15.0581	90177135	1550155933	-0.3886	-2.5157	17.1901	0.4834	3.180417e+11	3.180417e+11
2016-01-05	6.0	600000.SH	浦发银行	15.0581	15.1641	15.4732	15.0846	15.4114	55374454	964061502	0.3533	2.3460	17.4099	0.2969	3.255031e+11	3.255031e+11
2016-01-06	7.0	600000.SH	浦发银行	15.4114	15.5174	15.8088	15.3231	15.3584	47869312	843717365	-0.0530	-0.3438	17.6254	0.2566	3.243839e+11	3.243839e+11
2016-01-07	8.0	600000.SH	浦发银行	15.3584	15.0140	15.8883	14.9168	15.8265	54838833	966117848	0.4681	3.0477	17.6174	0.294	3.342702e+11	3.342702e+11
2016-01-08	9.0	600000.SH	浦发银行	15.8265	15.7205	16.0296	15.4732	15.5262	46723139	836146426	-0.3003	-1.8973	17.8958	0.2505	3.279280e+11	3.279280e+11
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-05-03	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-05-04	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-05-05	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-05-08	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2017-05-09	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

309 rows × 18 columns

77.使用expending函数计算开盘价的移动窗口均值

```
data['开盘价(元)'].expanding(min_periods=1).mean()
```

```
日期
2016-01-04    16.144400
2016-01-05    15.804400
2016-01-06    15.805867
2016-01-07    15.784525
2016-01-08    15.761120
                ...
2017-05-03    16.041489
2017-05-04    16.038314
2017-05-05    16.034769
2017-05-08    16.030695
2017-05-09    16.026356
Name: 开盘价(元), Length: 309, dtype: float64
```

78.绘制上一题的移动均值与原始数据折线图

```
data['expanding Open mean']=data['开盘价(元)'].expanding(min_periods=1).mean()
data[['开盘价(元)', 'expanding Open mean']].plot(figsize=(16, 6));
```

79.计算布林指标

```
data['former 30 days rolling Close mean']=data['收盘价(元)'].rolling(20).mean()
data['upper bound']=data['former 30 days rolling Close mean']+2*data['收盘价(元)'].rolling(20).std()#在这里我们取20天内的标准差
data['lower bound']=data['former 30 days rolling Close mean']-2*data['收盘价(元)'].rolling(20).std()
```

```
data
```

	index	代码	简称	前收盘价(元)	开盘价(元)	最高价(元)	最低价(元)	收盘价(元)	成交量(股)	成交金额(元)	...	均价(元)	换手率(%)	A股流通市值(元)	总市值(元)	A股流通股本(股)
日期																
2016-01-04	0	600000.SH	浦发银行	16.1356	16.1444	16.1444	15.4997	15.7205	42240610	754425783	...	17.8602	0.2264	3.320318e+11	3.320318e+11	1.865347e+10
2016-01-05	1	600000.SH	浦发银行	15.7205	15.4644	15.9501	15.3672	15.8618	58054793	1034181474	...	17.8139	0.3112	3.350163e+11	3.350163e+11	1.865347e+10
2016-01-06	2	600000.SH	浦发银行	15.8618	15.8088	16.0208	15.6234	15.9855	46772653	838667398	...	17.9307	0.2507	3.376278e+11	3.376278e+11	1.865347e+10
2016-01-07	3	600000.SH	浦发银行	15.9855	15.7205	15.8088	15.3672	15.4644	11350479	199502702	...	17.5766	0.0608	3.266223e+11	3.266223e+11	1.865347e+10
2016-01-08	4	600000.SH	浦发银行	15.4644	15.6675	15.7912	14.9345	15.4467	71918296	1262105060	...	17.5492	0.3855	3.262492e+11	3.262492e+11	1.865347e+10
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2017-05-03	322	600000.SH	浦发银行	15.1600	15.1600	15.1600	15.0500	15.0800	14247943	215130847	...	15.0991	0.0659	3.260037e+11	3.260037e+11	2.161828e+10
2017-05-04	323	600000.SH	浦发银行	15.0800	15.0700	15.0700	14.9000	14.9800	19477788	291839737	...	14.9832	0.0901	3.238418e+11	3.238418e+11	2.161828e+10
2017-05-05	324	600000.SH	浦发银行	14.9800	14.9500	14.9800	14.5200	14.9200	40194577	592160198	...	14.7323	0.1859	3.225447e+11	3.225447e+11	2.161828e+10
2017-05-08	325	600000.SH	浦发银行	14.9200	14.7800	14.9000	14.5100	14.8600	43568576	638781010	...	14.6615	0.2015	3.212476e+11	3.212476e+11	2.161828e+10
2017-05-09	326	600000.SH	浦发银行	14.8600	14.6900	14.8400	14.6600	14.7600	19225492	283864640	...	14.765	0.0889	3.190858e+11	3.190858e+11	2.161828e+10

309 rows × 22 columns

80.计算布林线并绘制

```
data[['收盘价(元)', 'former 30 days rolling Close mean','upper bound','lower bound' ]].plot(figsize=(16, 6));
```

4.当pandas遇上Numpy

81.查看pandas与numpy版本

```
import pandas as pd
import numpy as np
print(np.__version__)
print(pd.__version__)
```

```
1.16.5
0.25.3
```

82.使用numpy生成20个0-100随机数构成df

```
#使用numpy生成20个0-100随机数
tem = np.random.randint(1,100,20)
df1 = pd.DataFrame(tem)
df1
```

	0
0	52
1	42
2	26
3	70
4	11
5	90
6	6
7	66
8	90
9	2
10	67
11	40
12	68
13	26
14	22
15	49
16	63
17	69
18	21
19	5

83.使用numpy生成20个0-100固定步长的数

```
#使用numpy生成20个0-100固定步长的数
tem = np.arange(0,100,5)
df2 = pd.DataFrame(tem)
df2
```

	0
0	0
1	5
2	10
3	15
4	20
5	25
6	30
7	35
8	40
9	45
10	50
11	55
12	60
13	65

14	70
15	75
16	80
17	85
18	90
19	95

84.使用**numpy**生成**20**个指定分布**(**如标准正态分布**)**的数

```
tem = np.random.normal(0, 1, 20)
df3 = pd.DataFrame(tem)
df3
```

	0
0	-1.046510
1	0.112247
2	0.063541
3	1.561601
4	0.271020
5	-0.054702
6	0.949461
7	-1.594941
8	-1.410419
9	-0.166459
10	-0.404137
11	1.395874
12	0.717067
13	0.228148
14	0.348453
15	-0.301339
16	1.738995
17	1.235685
18	-0.205439
19	0.804149

85.将**df1**，**df2**，**df3**按照行合并为新**DataFrame**

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

	0
0	52.000000
1	42.000000
2	26.000000
3	70.000000
4	11.000000
5	90.000000
6	6.000000
7	66.000000
8	90.000000
9	2.000000
10	67.000000
11	40.000000
12	68.000000
13	26.000000
14	22.000000
15	49.000000
16	63.000000
17	69.000000

	0
18	21.000000
19	5.000000
20	0.000000
21	5.000000
22	10.000000
23	15.000000
24	20.000000
25	25.000000
26	30.000000
27	35.000000
28	40.000000
29	45.000000
30	50.000000
31	55.000000
32	60.000000
33	65.000000
34	70.000000
35	75.000000
36	80.000000
37	85.000000
38	90.000000
39	95.000000
40	-1.046510
41	0.112247
42	0.063541
43	1.561601
44	0.271020
45	-0.054702
46	0.949461
47	-1.594941
48	-1.410419
49	-0.166459
50	-0.404137
51	1.395874
52	0.717067
53	0.228148
54	0.348453
55	-0.301339
56	1.738995
57	1.235685
58	-0.205439
59	0.804149

86.将df1，df2，df3按照列合并为新DataFrame

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

	0	1	2
0	52	0	-1.046510
1	42	5	0.112247
2	26	10	0.063541
3	70	15	1.561601
4	11	20	0.271020
5	90	25	-0.054702
6	6	30	0.949461

7	66	35	-1.594941
8	90	40	-1.410419
9	2	45	-0.166459
10	67	50	-0.404137
11	40	55	1.395874
12	68	60	0.717067
13	26	65	0.228148
14	22	70	0.348453
15	49	75	-0.301339
16	63	80	1.738995
17	69	85	1.235685
18	21	90	-0.205439
19	5	95	0.804149

87.查看**df**所有数据的最小值、**25%**分位数、中位数、**75%**分位数、最大值

```
np.percentile(df, q=[0, 25, 50, 75, 100])
```

```
array([-1.59494057,  0.78237889, 21.5       , 60.75      , 95.        ])
```

88.修改列名为**col1,col2,col3**

```
df.columns = ['col1','col2','col3']
```

89.提取第一列中不在第二列出现的数字

```
df['col1'][~df['col1'].isin(df['col2'])]
```

```
0     52
1     42
2     26
4     11
6      6
7     66
9      2
10    67
12    68
13    26
14    22
15    49
16    63
17    69
18    21
Name: col1, dtype: int32
```

90.提取第一列和第二列出现频率最高的三个数字

```
temp = df['col1'].append(df['col2'])
temp.value_counts().index[:3]
```

```
Int64Index([90, 26, 40], dtype='int64')
```

91.提取第一列中可以整除**5**的数字位置

```
np.argwhere(df['col1']%5==0)
```

```
D:\Anaconda3\lib\site-packages\numpy\core\fromnumeric.py:56: FutureWarning: Series.nonzero() is deprecated and will be removed in a future version.Use Series.to_numpy().nonzero() instead
  return getattr(obj, method)(*args, **kwds)
```

```
array([[ 3],
       [ 5],
       [ 8],
       [11],
       [19]], dtype=int64)
```

```
df[df['col1']%5==0].index.tolist()
```

```
[3, 5, 8, 11, 19]
```

92.计算第一列数字前一个与后一个的差值

```
df['col1'].diff().tolist()
```

```
[nan,
-10.0,
-16.0,
44.0,
-59.0,
79.0,
-84.0,
60.0,
24.0,
-88.0,
65.0,
-27.0,
28.0,
-42.0,
-4.0,
27.0,
14.0,
6.0,
-48.0,
-16.0]
```

93.将col1,col2,clo3三列顺序颠倒¶

```
df.ix[:,::-1]
```

```
D:\Anaconda3\lib\site-packages\ipykernel_launcher.py:1: FutureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecated
  """Entry point for launching an IPython kernel.
D:\Anaconda3\lib\site-packages\pandas\core\indexing.py:822: FutureWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ix-indexer-is-deprecated
  retval = getattr(retval, self.name)._getitem_axis(key, axis=i)
```

	col3	col2	col1
0	-1.046510	0	52
1	0.112247	5	42
2	0.063541	10	26
3	1.561601	15	70
4	0.271020	20	11
5	-0.054702	25	90
6	0.949461	30	6
7	-1.594941	35	66
8	-1.410419	40	90
9	-0.166459	45	2
10	-0.404137	50	67
11	1.395874	55	40
12	0.717067	60	68
13	0.228148	65	26
14	0.348453	70	22
15	-0.301339	75	49
16	1.738995	80	63
17	1.235685	85	69
18	-0.205439	90	21
19	0.804149	95	5

```
df.iloc[:,::-1]
```

	col3	col2	col1
0	-1.046510	0	52
1	0.112247	5	42
2	0.063541	10	26
3	1.561601	15	70
4	0.271020	20	11
5	-0.054702	25	90

6	0.949461	30	6
7	-1.594941	35	66
8	-1.410419	40	90
9	-0.166459	45	2
10	-0.404137	50	67
11	1.395874	55	40
12	0.717067	60	68
13	0.228148	65	26
14	0.348453	70	22
15	-0.301339	75	49
16	1.738995	80	63
17	1.235685	85	69
18	-0.205439	90	21
19	0.804149	95	5

```
df.loc[:,df.columns.values.tolist()[::-1]]
```

	col3	col2	col1
0	-1.046510	0	52
1	0.112247	5	42
2	0.063541	10	26
3	1.561601	15	70
4	0.271020	20	11
5	-0.054702	25	90
6	0.949461	30	6
7	-1.594941	35	66
8	-1.410419	40	90
9	-0.166459	45	2
10	-0.404137	50	67
11	1.395874	55	40
12	0.717067	60	68
13	0.228148	65	26
14	0.348453	70	22
15	-0.301339	75	49
16	1.738995	80	63
17	1.235685	85	69
18	-0.205439	90	21
19	0.804149	95	5

94.提取第一列位置在1,10,15的数字

```
df.iloc[[1,10,15],0]
```

```
1    42
10   67
15   49
Name: col1, dtype: int32
```

```
df['col1'].take([1,10,15])
```

```
1    42
10   67
15   49
Name: col1, dtype: int32
```

95.查找第一列的局部最大值位置

```
# 即比它前一个与后一个数字的都大的数字
tem = np.diff(np.sign(np.diff(df['col1'])))
np.where(tem == -2)[0] + 1
```

```
array([ 3,  5,  8, 10, 12, 17], dtype=int64)
```

```
df['col1']
```

```
0    52
1    42
2    26
3    70
4    11
5    90
6     6
7    66
8    90
9     2
10   67
11   40
12   68
13   26
14   22
15   49
16   63
17   69
18   21
19    5
Name: col1, dtype: int32
```

```
np.diff(df['col1'])
```

```
array([-10, -16,  44, -59,  79, -84,  60,  24, -88,  65, -27,  28, -42,
        -4,  27,  14,   6, -48, -16])
```

```
np.sign(np.diff(df['col1']))
```

```
array([-1, -1,  1, -1,  1, -1,  1,  1, -1,  1, -1,  1, -1, -1,  1,  1,  1,
       -1, -1])
```

```
np.diff(np.sign(np.diff(df['col1'])))
```

```
array([ 0,  2, -2,  2, -2,  2,  0, -2,  2, -2,  2, -2,  0,  2,  0,  0, -2,
        0])
```

96.按行计算df的每一行均值

```
df.mean(axis=1)
```

```
0    16.984497
1    15.704082
2    12.021180
3    28.853867
4    10.423673
5    38.315099
6    12.316487
7    33.135020
8    42.863194
9    15.611180
10   38.865288
11   32.131958
12   42.905689
13   30.409383
14   30.782818
15   41.232887
16   48.246332
17   51.745228
18   36.931520
19   33.601383
dtype: float64
```

```
df[['col1','col2','col3']].mean(axis=1)
```

```
0    16.984497
1    15.704082
2    12.021180
3    28.853867
4    10.423673
5    38.315099
6    12.316487
7    33.135020
8    42.863194
9    15.611180
10   38.865288
11   32.131958
12   42.905689
13   30.409383
14   30.782818
15   41.232887
16   48.246332
17   51.745228
18   36.931520
19   33.601383
dtype: float64
```

97.对第二列计算移动平均值

```
df['col2']
```

```
0     0
1     5
2    10
3    15
4    20
5    25
6    30
7    35
8    40
9    45
10   50
11   55
12   60
13   65
14   70
15   75
16   80
17   85
18   90
19   95
Name: col2, dtype: int32
```

```
#每次移动三个位置，不可以使用自定义函数
np.convolve(df['col2'], np.ones(3)/3, mode='valid')
```

```
array([ 5., 10., 15., 20., 25., 30., 35., 40., 45., 50., 55., 60., 65.,
       70., 75., 80., 85., 90.])
```

98.将数据按照第三列值的大小升序排列

```
df.sort_values("col3",inplace=True)
df
```

	col1	col2	col3
7	66	35	-1.594941
8	90	40	-1.410419
0	52	0	-1.046510
10	67	50	-0.404137
15	49	75	-0.301339
18	21	90	-0.205439
9	2	45	-0.166459
5	90	25	-0.054702
2	26	10	0.063541
1	42	5	0.112247
13	26	65	0.228148
4	11	20	0.271020
14	22	70	0.348453
12	68	60	0.717067
19	5	95	0.804149
6	6	30	0.949461
17	69	85	1.235685
11	40	55	1.395874
3	70	15	1.561601
16	63	80	1.738995

99.将第一列大于50的数字修改为'高'

```
df.col1[df['col1'] > 50]= '高'
df
```

	col1	col2	col3
7	高	35	-1.594941
8	高	40	-1.410419
0	高	0	-1.046510
10	高	50	-0.404137
15	49	75	-0.301339
18	21	90	-0.205439
9	2	45	-0.166459
5	高	25	-0.054702
2	26	10	0.063541

	col1	col2	col3
1	42	5	0.112247
13	26	65	0.228148
4	11	20	0.271020
14	22	70	0.348453
12	高	60	0.717067
19	5	95	0.804149
6	6	30	0.949461
17	高	85	1.235685
11	40	55	1.395874
3	高	15	1.561601
16	高	80	1.738995

100.计算第二列与第三列之间的欧式距离

```
np.linalg.norm(df['col2']-df['col3'])
```

```
247.09477496805883
```

5.其他操作

101.从CSV文件中读取指定数据

```
# 从数据1中的前10行中读取positionName, salary两列
# usecols 设置导入的列
# encoding 解码方式
# nrows 导入的行数
df = pd.read_csv('pandas120/数据1_101-120涉及.csv',encoding='gbk', usecols=['positionName', 'salary'],nrows = 10)
df
```

	positionName	salary
0	数据分析	37500
1	数据建模	15000
2	数据分析	3500
3	数据分析	45000
4	数据分析	30000
5	数据分析	50000
6	数据分析	30000
7	数据建模工程师	35000
8	数据分析专家	60000
9	数据分析师	40000

102.从CSV文件中读取指定数据

```
# 从数据2中读取数据并在读取数据时将薪资大于10000的为改为高
df = pd.read_csv('E:/桌面/为了工作/面试题/pandas120/数据2_101-120涉及.csv',converters={'薪资水平': lambda x: '高' if float(x) > 10000 else '低'} )
df
```

	学历要求	薪资水平
0	本科	高
1	硕士	高
2	本科	低
3	本科	高
4	不限	高
...	...	...
1149	硕士	高
1150	本科	高
1151	本科	高
1152	本科	高
1153	本科	高

1154 rows × 2 columns

103.从上一题数据中，对薪资水平列每隔20行进行一次抽样

```
df.iloc[::20, :][['薪资水平']]
```

	薪资水平

	薪资水平
0	高
20	高
40	高
60	高
80	高
100	高
120	高
140	高
160	高
180	高
200	高
220	高
240	高
260	高
280	低
300	高
320	高
340	低
360	高
380	高
400	高
420	高
440	高
460	低
480	高
500	高
520	高
540	高
560	高
580	高
600	高
620	高
640	高
660	低
680	低
700	高
720	高
740	高
760	高
780	高
800	高
820	高
840	高
860	低
880	高
900	高
920	高
940	高
960	高
980	高
1000	高
1020	高
1040	高
1060	高

1080	薪资水平 高
1100	高
1120	高
1140	高

104.将数据取消使用科学计数法

```
df = pd.DataFrame(np.random.random(10)**10, columns=['data'])
df
```

	data
0	1.462932e-08
1	6.941200e-01
2	1.455299e-09
3	1.857895e-04
4	2.401570e-09
5	7.926270e-01
6	7.644857e-03
7	2.117843e-02
8	2.422171e-02
9	1.510634e-01

```
df.round(3)
```

	data
0	0.000
1	0.694
2	0.000
3	0.000
4	0.000
5	0.793
6	0.008
7	0.021
8	0.024
9	0.151

105.将上一题的数据转换为百分数

```
df.style.format({'data': '{0:.2%}'.format})
```

```
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row0" class="row_heading level0 row0" >0</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row0_col0" class="data row0 col0" >0.00%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row1" class="row_heading level0 row1" >1</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row1_col0" class="data row1 col0" >69.41%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row2" class="row_heading level0 row2" >2</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row2_col0" class="data row2 col0" >0.00%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row3" class="row_heading level0 row3" >3</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row3_col0" class="data row3 col0" >0.02%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row4" class="row_heading level0 row4" >4</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row4_col0" class="data row4 col0" >0.00%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row5" class="row_heading level0 row5" >5</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row5_col0" class="data row5 col0" >79.26%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row6" class="row_heading level0 row6" >6</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row6_col0" class="data row6 col0" >0.76%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row7" class="row_heading level0 row7" >7</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row7_col0" class="data row7 col0" >2.12%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row8" class="row_heading level0 row8" >8</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row8_col0" class="data row8 col0" >2.42%</td>
        </tr>
        <tr>
                <th id="T_8ad8d580_8633_11ea_9800_f834414b6977level0_row9" class="row_heading level0 row9" >9</th>
                <td id="T_8ad8d580_8633_11ea_9800_f834414b6977row9_col0" class="data row9 col0" >15.11%</td>
        </tr>
</tbody></table>
```

	data

106.查找数据中第3大值的行号

```
temp= pd.DataFrame(np.random.randint(1,30,15),columns=['a'])
temp
```

	a
0	4
1	8
2	15
3	5
4	3
5	28
6	14
7	17
8	26
9	6
10	9
11	6
12	20
13	13
14	13

```
temp['a'].argsort()[::-1].values.tolist()[2]
```

```
12
```

107.反转df的行

```
df.iloc[::-1, :]
```

	data
9	1.510634e-01
8	2.422171e-02
7	2.117843e-02

	data
6	7.644857e-03
5	7.926270e-01
4	2.401570e-09
3	1.857895e-04
2	1.455299e-09
1	6.941200e-01
0	1.462932e-08

108.按照多列对数据进行合并

```
#输入
df1 = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
'key2': ['K0', 'K1', 'K0', 'K1'],
'A': ['A0', 'A1', 'A2', 'A3'],
'B': ['B0', 'B1', 'B2', 'B3']})

df2 = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
'key2': ['K0', 'K0', 'K0', 'K0'],
'C': ['C0', 'C1', 'C2', 'C3'],
'D': ['D0', 'D1', 'D2', 'D3']})
```

```
df1
df2
```

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

```
# 默认是内连接
pd.merge(df1,df2,on=['key1','key2'])
```

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K1	K0	A2	B2	C1	D1
2	K1	K0	A2	B2	C2	D2

109.按照多列对数据进行合并（左连接）

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

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN

110.再次读取数据1并显示所有的列

```
df = pd.read_csv('pandas120/数据1_101-120涉及.csv',encoding='gbk')
pd.set_option("display.max.columns", None)
df
```

	positionId	positionName	companyId	companyLogo	companySize	industryField	financeStage	companyLabelList
0	6802721	数据分析	475770	i/image2/M01/B7/3E/CgoB5lwPfEaAdn8WAABWQ0Jgl5s...	50-150人	移动互联网,电商	A轮	['绩效奖金', '带薪年假', '定期体检', '弹性工作']
1	5204912	数据建模	50735	image1/M00/00/85/CgYXBlTUXeeAR0ljAABbroUk-dw97...	150-500人	电商	B轮	['年终奖金', '做五休二', '六险一金', '子女福利']
2	6877668	数据分析	100125	image2/M00/0C/57/CgqLKVYcOA2ADcFuAAAE8MukIKA74...	2000人以上	移动互联网,企业服务	上市公司	['节日礼物', '年底双薪', '股票期权', '带薪年假']

	positionId	positionName	companyId	companyLogo	companySize	industryField	financeStage	companyLabelList
3	6496141	数据分析	26564	i/image2/M01/F7/3F/CgoB5lyGAQGAZeI-AAAdOqXecnw...	500-2000人	电商	D轮及以上	['生日趴', '每月腐败基金', '每月补贴', '年度旅游']
4	6467417	数据分析	29211	i/image2/M01/77/B8/CgoB5l1WDyGATNP5AAAIY3h88SY...	2000人以上	物流丨运输	上市公司	['技能培训', '免费班车', '专项奖金', '岗位晋升']
...	...	...	...	...	...	...	...	...
100	6884346	数据分析师	21236	i/image/M00/43/F6/CgqKkVeEh76AUVPoAAA2Bj747wU6...	500-2000人	移动互联网,医疗丨健康	C轮	['技能培训', '年底双薪', '节日礼物', '绩效奖金']
101	6849100	商业数据分析	72076	i/image2/M01/92/A4/CgotOV2LPUmAR_8dAAB_DlDMiXA...	500-2000人	移动互联网,电商	C轮	['节日礼物', '股票期权', '带薪年假', '年度旅游']
102	6803432	奔驰・耀出行-BI数据分析专家	751158	i/image3/M01/64/93/Cgq2xl48z2mAeYRoAAD6Qf_Jeq8...	150-500人	移动互联网	不需要融资	[]
103	6704835	BI数据分析师	52840	i/image2/M00/26/CA/CgoB5lofsguAfk9ZAACoL3r4p24...	2000人以上	电商	上市公司	['技能培训', '年底双薪', '节日礼物', '绩效奖金']
104	6728058	数据分析专家-LQ(J181203029)	2474	i/image2/M01/14/4D/CgoB5lyq5fqAAHHzAAAa148hbk8...	2000人以上	汽车丨出行	不需要融资	['弹性工作', '节日礼物', '岗位晋升', '技能培训']

105 rows × 53 columns

111.查找secondType与thirdType值相等的行号¶

```
np.where(df.secondType == df.thirdType)
```

```
(array([  0,   2,   4,   5,   6,  10,  14,  23,  25,  27,  28,  29,  30,
         33,  37,  38,  39,  40,  41,  48,  49,  52,  53,  55,  57,  61,
         65,  66,  67,  71,  73,  74,  75,  79,  80,  82,  85,  88,  89,
         91,  96, 100], dtype=int64),)
```

112.查找薪资大于平均薪资的第三个数据

```
np.argwhere(df['salary'] > df['salary'].mean())[2]
```

```
array([5], dtype=int64)
```

113.将上一题数据的salary列开根号

```
df[['salary']].apply(np.sqrt)
```

	salary
0	193.649167
1	122.474487
2	59.160798
3	212.132034
4	173.205081
...	...
100	158.113883
101	187.082869
102	173.205081
103	141.421356
104	146.628783

105 rows × 1 columns

114.将数据的linestaion列按_拆分

```
df['split'] = df['linestaion'].str.split('_')
```

```
df['split']
```

```
0                                NaN
1                                NaN
2           [4号线, 城星路;4号线, 市民中心;4号线, 江锦路]
3                          [1号线, 文泽路]
4                                NaN
              ...
100                              NaN
101                              NaN
102    [1号线, 滨和路;1号线, 江陵路;1号线, 滨和路;1号线, 江陵路]
103                              NaN
104                              NaN
Name: split, Length: 105, dtype: object
```

115.查看数据中一共有多少列

```
df.shape[1]
```

```
54
```

116.提取industryField列以’数据’开头的行

```
#df[df['industryField'].str.startswith('数据')]
df[df.industryField.str.startswith('数据')]
```

	positionId	positionName	companyId	companyLogo	companySize	industryField	financeStage	companyLabelList
8	6458372	数据分析专家	34132	i/image2/M01/F8/DE/CgoB5lyHTJeAP7v9AAFXUt4zJo4...	150-500人	数据服务,广告营销	A轮	['开放式办公', '扁平管理', '带薪假期', '弹性工作时间']
10	6804629	数据分析师	34132	i/image2/M01/F8/DE/CgoB5lyHTJeAP7v9AAFXUt4zJo4...	150-500人	数据服务,广告营销	A轮	['开放式办公', '扁平管理', '带薪假期', '弹性工作时间']
13	6804489	资深数据分析师	34132	i/image2/M01/F8/DE/CgoB5lyHTJeAP7v9AAFXUt4zJo4...	150-500人	数据服务,广告营销	A轮	['开放式办公', '扁平管理', '带薪假期', '弹性工作时间']
21	6267370	数据分析专家	31544	image1/M00/00/48/CgYXBlTUXOaADKooAABjQoD_n1w50...	150-500人	数据服务	不需要融资	['专业红娘牵线', '节日礼物', '技能培训', '岗位晋升']
32	6804489	资深数据分析师	34132	i/image2/M01/F8/DE/CgoB5lyHTJeAP7v9AAFXUt4zJo4...	150-500人	数据服务,广告营销	A轮	['开放式办公', '扁平管理', '带薪假期', '弹性工作时间']
37	6242470	数据分析师	31544	image1/M00/00/48/CgYXBlTUXOaADKooAABjQoD_n1w50...	150-500人	数据服务	不需要融资	['专业红娘牵线', '节日礼物', '技能培训', '岗位晋升']
50	6680900	数据分析师(MJ000250)	114335	i/image2/M00/17/C2/CgoB5ln5lUuAM8oSAADO2Rz54hQ...	150-500人	数据服务	B轮	['股票期权', '弹性工作', '领导好', '五险一金']
63	6680900	数据分析师(MJ000250)	114335	i/image2/M00/17/C2/CgoB5ln5lUuAM8oSAADO2Rz54hQ...	150-500人	数据服务	B轮	['股票期权', '弹性工作', '领导好', '五险一金']
78	5683671	数据分析实习生(MJ000087)	114335	i/image2/M00/17/C2/CgoB5ln5lUuAM8oSAADO2Rz54hQ...	150-500人	数据服务	B轮	['股票期权', '弹性工作', '领导好', '五险一金']
79	6046866	数据分析师	543802	i/image2/M01/63/3C/CgotOV0ulwOAU8KWAAAsMECc53M...	15-50人	数据服务	不需要融资	[]
92	6813626	资深数据分析专员	165939	i/image3/M01/65/71/CgpOIF5CFp2ACoo9AAD3lkKwlv8...	150-500人	数据服务	不需要融资	['年底双薪', '带薪年假', '午餐补助', '定期体检']
94	6818950	资深数据分析师	165939	i/image3/M01/65/71/CgpOIF5CFp2ACoo9AAD3lkKwlv8...	150-500人	数据服务	不需要融资	['年底双薪', '带薪年假', '午餐补助', '定期体检']
97	6718750	旅游大数据分析师（杭州）	122019	i/image/M00/1A/4A/CgqKkVb583WABT4BAABM5RuPCmk9...	50-150人	数据服务,企业服务	A轮	['年底双薪', '股票期权', '午餐补助', '定期体检']

	positionId	positionName	companyId	companyLogo	companySize	industryField	financeStage	companyLabelList
98	6655562	数据分析建模工程师	117422215	i/image2/M01/AF/6D/CgotOV3ki4iAOuo3AABbiII8DfA...	50-150人	数据服务,信息安全	A轮	['午餐补助', '带薪年假', '16到18薪', '法定节假日']
99	6677939	数据分析建模工程师（校招）	117422215	i/image2/M01/AF/6D/CgotOV3ki4iAOuo3AABbiII8DfA...	50-150人	数据服务,信息安全	A轮	['午餐补助', '带薪年假', '16到18薪', '法定节假日']

117.按列制作数据透视表

```
pd.pivot_table(df,values=["salary","score"],index="positionId")
```

	salary	score
positionId		
5203054	30000	4.0
5204912	15000	176.0
5269002	37500	1.0
5453691	30000	4.0
5519962	37500	14.0
...	...	...
6882983	27500	15.0
6884346	25000	0.0
6886661	37500	5.0
6888169	42500	1.0
6896403	30000	3.0

95 rows × 2 columns

118.同时对**salary**、**score**两列进行计算¶

```
df[["salary","score"]].agg([np.sum,np.mean,np.min])
```

	salary	score
sum	3.331000e+06	1335.000000
mean	3.172381e+04	12.714286
amin	3.500000e+03	0.000000

119.对**salary**求平均，对**score**列求和

```
df.agg({"salary":np.sum,"score":np.mean})
```

```
salary    3.331000e+06
score     1.271429e+01
dtype: float64
```

120.计算并提取平均薪资最高的区

```
df[['district','salary']].groupby(by='district').mean().sort_values('salary',ascending=False).head(1)
```

	salary
district	
萧山区	36250.0