统计分析分为*描述性统计*和*推断统计*

量化中常用的是描述性统计

描述性统计

描述性统计，是指运用制表和分类，图形以及计算概括性数据来描述数据特征的各项活动。描述性统计分析要对调查总体所有变量的有关数据进行统计性描述，主要包括数据的频数分析、集中趋势分析、离散程度分析、分布以及一些基本的统计图形。

- 频数分析：检验异常值
- 集中趋势分析：反映数据的一般水平（平均值，中位数和众数）
- 离散程度分析：反映数据的差异程度（方差和标准差）
- 分布：偏度和峰度两个指标来检查样本数据是否符合正态分布

1. 均值

In [0]:

```
from WindPy import *
w.start()

error_code, data_df = w.wsd("000001.SZ", "open,high,low,close,pct_chg,volume,amt", "2017-05-01", "2018-05-01", "",usedf=True)
data_df.head(5)
```

```
COPYRIGHT (C) 2017 Wind Information Co., Ltd. ALL RIGHTS RESERVED.
IN NO CIRCUMSTANCE SHALL WIND BE RESPONSIBLE FOR ANY DAMAGES OR LOSSES
CAUSED BY USING WIND QUANT API FOR PYTHON.
```

Out[0]:

	OPEN	HIGH	LOW	CLOSE	PCT_CHG	VOLUME	AMT
2017-05-02 00:00:00.005	8.96	8.96	8.90	8.94	-0.556174	31102610.0	277665881.0
2017-05-03 00:00:00.005	8.92	8.93	8.89	8.91	-0.335570	28031077.0	249660877.0
2017-05-04 00:00:00.005	8.89	8.89	8.72	8.74	-1.907969	69651707.0	613783434.0
2017-05-05 00:00:00.005	8.74	8.76	8.58	8.63	-1.258581	62370085.0	539541739.0
2017-05-08 00:00:00.005	8.60	8.62	8.54	8.57	-0.695249	46008989.0	394424117.0

In [0]:

```
import scipy.stats as stats
import numpy as np

print('000001.SZ 收盘价的均值：', sum(data_df['CLOSE']), "/", len(data_df['CLOSE']) , '= ', np.mean(data_df['CLOSE']))
```

```
000001.SZ 收盘价的均值：  2800.1000000000013 / 244 =  11.475819672131147
```

In [0]:

```
data_df['CLOSE'].mean()
```

Out[0]:

```
11.475819672131147
```

加权平均

1. 成交量加权平均价格（VWAP Volume-Weighted Average Price)

VWAP策略是一种拆分大额委托单，在约定时间段内分批执行，以期使得最终买入或卖出成交均价尽量接近这段时间内整个市场成交均价的交易策略

$$VWAP = \frac{\sum_{i=1}^{n} price_i * volume_i}{\sum_{i=1}^{n} volume_i}$$

VWAP模型必须把母单分割成为许多小的子单，并在一个指定时间段内逐步送出去。降低大单对市场的冲击，增加大单的隐秘性。

如果订单非常大，譬如超过市场日交易量的1%的话，即便VWAP可以在相当大的程度上改善市场冲击，但市场冲击仍然会以积累的方式改变市场，最终使得模型的效果差于预期。

VWAP算法交易的目的是最小化冲击成本，并不寻求最小化所有成本。理论上，在没有额外的信息，也没有针对股票价格趋势的预测的情况下，VWAP 是最优的算法交易策略。

In [0]:

```
vwap = np.average(data_df['CLOSE'], weights = data_df['VOLUME'])
print(vwap)
```

```
12.020335949847505
```

时间加权平均价格（TWAP）

TWAP模型是把一个母单的数量平均地分配到一个交易时段上。该模型将交易时间进行均匀分割，并在每个分割节点上将拆分的订单进行提交。例如，可以将某个交易日的交易时间平均分为 N 段，TWAP 策略会将该交易日需要执行的订单均匀分配在这 N 个时间段上去执行

$$TWAP = \frac{\sum_{i=1}^{n} price_i}{n}$$

TWAP模型的目的是使交易对市场影响减小的同时提供一个较低的平均成交价格，从而达到减小交易成本的目的。在分时成交量无法准确估计的情况下，该模型可以较好地实现算法交易的基本目的。

潜在问题：

- 订单规模大的情况下，均匀分配到每个节点上的下单量仍然会对市场造成一定的冲击
- 真是市场的成交量总是在波动变化的，将所有的订单均匀分配到每个节点上显然是不够合理的

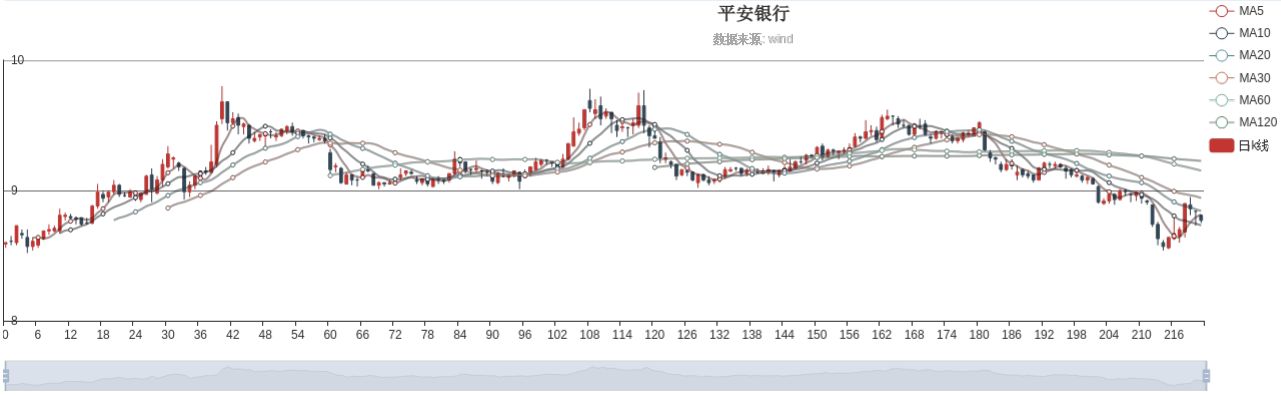
In [0]:

```
t = np.arange(len(data_df['CLOSE']))
#获取收盘价的长度，依次建立自然增长数列
twap = np.average(data_df['CLOSE'], weights = t)
print(twap)
```

```
12.125264116575595
```

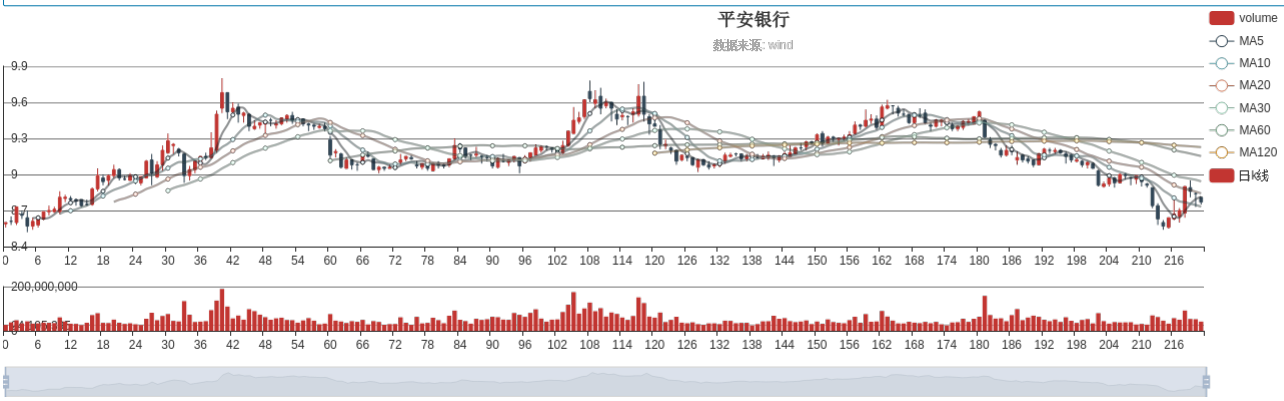
In [0]:

```
#该只股票的时间段走势

from pandas import DataFrame
from WindCharts import *
w.start(show_welcome = False)

wsd_data = w.wsd("000001.SZ", "open,high,low,close", "2016-06-18", "2017-05-17", "")
data_df = DataFrame(wsd_data.Data, index = ['open', 'high', 'low', 'close'])
data_df = data_df.T
data_df['date'] = [str(date_temp.date()) for date_temp in wsd_data.Times]

chart = WCandlestick("平安银行", "数据来源：wind", data=data_df, ma=[5,10,20,30,60,120])
chart.plot()
```



In [0]:

```
#添加Volume
wsd_data = w.wsd("000001.SZ", "open,high,low,close, volume", "2016-06-18", "2017-05-17", "")
data_df = DataFrame(wsd_data.Data, index = ['open', 'high', 'low', 'close', 'volume'])
data_df = data_df.T
data_df['date'] = [str(date_temp.date()) for date_temp in wsd_data.Times]

chart = WCandlestick("平安银行", "数据来源：wind", data=data_df, ma=[5,10,20,30,60,120])
chart.plot()
```



2. 取值范围

In [0]:

```
print("highest = ", np.max(data_df['high']))
print("lowest = ", np.min(data_df['low']))
```

```
highest =  9.8
lowest =  8.52
```

In [0]:

```
# numpy.ptp可以计算数组的取值范围，返回数组元素的最大值和最小值之间的差值

print("最高价的最大值与最小值的差：", np.ptp(data_df['high']))
print("最低价的最大值与最小值的差：", np.ptp(data_df['low']))
```

```
最高价的最大值与最小值的差： 1.200000000000001
最低价的最大值与最小值的差： 1.08
```

3. 中位数

In [0]:

```
print("Median:", np.median(data_df['close']))
```

```
Median: 9.16
```

In [0]:

```
a = pd.Series([2, 4, 5, 10])
a
print(sorted(a))
a.median()
```

```
[2, 4, 5, 10]
```

Out[0]:

```
4.5
```

4.众数

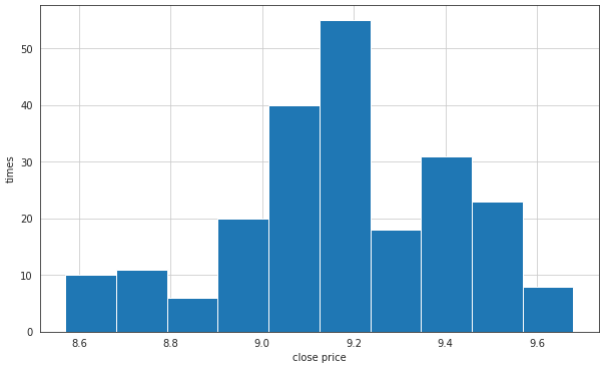
In [0]:

```
def mode(l):
    counts = {}
    for e in l:
        if e in counts:
            counts[e] += 1
        else:
            counts[e] = 1

    maxcount = 0
    modes = {}
    for (key, value) in counts.items():
        if value > maxcount:
            maxcount = value
            modes = {key}
        elif value == maxcount:
            modes.add(key)

        if maxcount > 1 or len(l) == 1:
            return list(modes)
        return "没有众数"

print("收盘价的众数: ", mode(data_df['close']))
```

收盘价的众数: 没有众数

In [0]:

```
## 频数 & pinlv

import matplotlib.pyplot as plt
import seaborn as sns

sns.set_style('white')
fig, ax = plt.subplots(figsize = (10, 6));
data_df['close'].hist()
plt.xlabel('close price')
plt.ylabel('times');
```



In [0]:

```
# histogram() & enumerate()

hist, bins = np.histogram(data_df['close'])
print('频数: ', '\n', hist)
print()
print('频率', '\n', bins)
```

频数:
 [10 11 6 20 40 55 18 31 23 8]

频率
 [8.57 8.681 8.792 8.903 9.014 9.125 9.236 9.347 9.458 9.569 9.68]

In [0]:

```
for i, j in enumerate(hist):
    print(i, j)
```

```
0 10
1 11
2 6
3 20
4 40
5 55
6 18
7 31
8 23
9 8
```

In [0]:

```
print ('观测样本数值最集中的范围是: ', [(bins[i], bins[i+1]) for i, j in enumerate(hist) if j == max(hist)])
```

观测样本数值最集中的范围是: [(9.125, 9.236)]

5. 几何平均值

$G = \sqrt[n]{X_1 X_2 \dots X_n}$

$\ln G = \frac{\sum_{i=1}^{n} \ln X_i}{n}$

几何平均值总是小于或等于算术平均值（所观测的数据为非负时），当所有数据都一样时等号成立

In [0]:

```
print("几何平均值", stats.gmean(data_df['close']))
```

几何平均值 9.171666789379701

收益率存在负数

$R_G = \sqrt[n]{(1+R_1)\dots(1+R_T)} - 1$

In [0]:

```
ratios = data_df['PCT_CHG'] / 100 + np.ones(len(data_df['PCT_CHG']))
R_G = stats.gmeans(ratios) - 1
print(R_G)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/opt/conda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
 2654 try:
-> 2655 return self._engine.get_loc(key)
 2656 except KeyError:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

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

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

KeyError: 'PCT_CHG'

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<76ba63339c2445d4a08d3c6bdddb4fe8> in <module>
      1 import ppdb
      2 ppdb.set_trace();from ppdb import self_print as print
----> 3 ratios = data_df['PCT_CHG'] / 100 + np.ones(len(data_df['PCT_CHG']))
      4 R_G = stats.gmeans(ratios) - 1
      5 print(R_G)

/opt/conda/lib/python3.6/site-packages/pandas/core/frame.py in __getitem__(self, key)
   2925             if self.columns.nlevels > 1:
   2926                 return self._getitem_multilevel(key)
-> 2927             indexer = self.columns.get_loc(key)
   2928             if is_integer(indexer):
   2929                 indexer = [indexer]

/opt/conda/lib/python3.6/site-packages/pandas/core/indexes/base.py in get_loc(self, key, method, tolerance)
   2657                 return self._engine.get_loc(key)
   2658             except KeyError:
-> 2659                 return self._engine.get_loc(self._maybe_cast_indexer(key))
   2660         indexer = self.get_indexer([key], method=method, tolerance=tolerance)
   2661         if indexer.ndim > 1 or indexer.size > 1:

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

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

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

KeyError: 'PCT_CHG'
```

In [0]:

```
print(np.ones(len(data_df['close'])))
```

```
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
```

6. 调和平均数

调和平均数又称倒数平均数，是总体各统计变量倒数的算术平均数的倒数。

$$H = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}}$$

In [0]:

```
print(stats.hmean(data_df['close']))
```

```
9.168355726263067
```