##

简单网络爬虫实现 ####小组分工

姓名	分工
齐新新	网页下载
罗俊斌	网页解析
罗庆鸣	数据清洗
王杏婷	数据分析

####Step1：获取URL　　给定URL：http://www.shfe.com.cn/statements/dataview.html?paramid=delaymarket_all ####Step2：网页下载（by 齐新新） ##### 直接访问

```
def http_direct():
    """
    function:
该函数利用库函数直接打开所需要的url，但是存在两个弊端，无法处理反爬虫机制，无法获取动态网页，只能得到静态HTML
    Disadvantage:
    can not handle anti-spider mechanism
    """
    url = 'http://www.shfe.com.cn/statements/dataview.html?paramid=delaymarket_all'
    response = urllib2.urlopen(url)
    page = response.read()
    page = page.decode('utf-8')
    return page
```

模拟浏览器get请求

```
def http_get():
    """
    function:
this function simulates the browser to request a static HTML page from the web server.
    Disadvantage:
    can not get attribute "src" of iframe, which unables to solve the problem of dynamic page acquisition
    """
    url = 'http://www.shfe.com.cn/statements/delaymarket_all.html'
    headers = {
'User-Agent': 'Mozilla/5.0 (X11; Linux x86_64; rv:52.0) Gecko/20100101 Firefox/52.0',
 'Referer': 'http://www.shfe.com.cn/statements/delaymarket_all.html',
 'Connecttion': 'keep-alive'
```

```
                }
        req = urllib2.Request(url, headers=headers)
            page = urllib2.urlopen(req).read()
                    return page
```

利用selenium+PantomJS工具

```
                def get(url):
                    """
                    function:
        this function gets the "src" with the help of selenium and PantomJS,
              which enables spider to get the dynamic webpage
                    """
                driver = webdriver.PhantomJS()
                        driver.get(url)
                         time.sleep(1)
                driver.switch_to.frame("smsg")
            content = driver.page_source.encode('utf-8')
                        return content
```

####Step3：网页解析（by 罗俊斌）

网页数据获取

通过step1和step2能够获取得到我们需要的数据网页"delaymarket_all.html"，读取该文件之后可以发现该文件中其实仅存在着三种类型的对象：table, tr和td。基于以上发现，可编写以下获取数据代码：

```
                def get_dtframe(page_str):
            bs = BeautifulSoup(page_str, "html.parser")
                    table = bs.find_all('table')
                    tr = table[0].find_all('tr')

                        #the first table
                           final = []
                         for cnt in tr:
                       td = cnt.find_all('td')
                              temp = []
                           for cntt in td:
                        temp.append(cntt.string)
                         final.append(temp)
                       colHead = final[1]
                        final = final[2:]
        table_of_Data = DataFrame(final, columns=colHead)

                        #the second table
                           final = []
```

```
            tr = table[1].find_all('tr')
                for cnt in tr:
                td = cnt.find_all('td')
                    temp = []
                  for cntt in td:
                  temp.append(cntt.string)
                  final.append(temp)
                colHead = final[0]
                final = final[1:]
        table_Of_End = DataFrame(final,columns = colHead)
    table_Of_End.to_csv('Data_Of_Footer.csv',encoding="utf-8")

                return table_of_Data
```

- 由于使用正则表达式没有得到一个很令人满意的结果（代码量太大），因此使用了BeautifulSoup包，并使用了find_all函数获取到页面当中所有的table, tr和td对象。使用append()分别针对两个table生成两个二维list，最后生成DataFrame对象。
- 简单来说，Beautiful Soup是python的一个库，最主要的功能是从网页抓取数据，Beautiful Soup提供一些简单的、Python式的函数用来处理导航、搜索、修改分析树等功能。 ####Step4：数据清洗（by 罗庆鸣）

###对解析网页得到的dataframe进行排序处理并存储

- 将的到的dataframe按照对应的列进行排序 设计函数Data_Sort，传入参数为之前解析得到的dataframe、列名对应的list、要转化为float排序的列，对对应的列的每个元素类型转化为float进行排序处理，如果其中包含None，则将None替换为0，这里主要针对成交量这列可能存在None类型的值，函数代码如下：

```
#different strategy of sorting the dataframe
    def Data_Sort(dtframe, colHead, col):
   for i in range(len(dtframe[colHead[col]])):
     if str(dtframe[colHead[col]][i]) == 'None':
             dtframe[colHead[col]][i] = '0'
dtframe[colHead[col]][i] = float(dtframe[colHead[col]][i])

    dtframe = dtframe.sort_values(by=colHead[col])
                return dtframe
```

- 首先调用上面的函数按照涨跌幅进行排序，传入参数为得到的dataframe，以及列数3也就是涨跌幅对应的列，再将得到的表格存为"涨跌.csv".
- 然后调用上面的函数按照成交量进行排序，传入参数为得到的dataframe，以及列数5也就是涨跌幅对应的列，再将得到的表格存为"成交量.csv".

###对解析页面得到的dataframe进行去除None值并存储

- 对开盘，最低价，最高价三列去除空值然后将产生的新的dataframe存储为csv，主要是使用dataframe中的loc函数，筛选出对应列中不为空的行组成新的dataframe然后储存：

```
tableOfKP = table.loc[table[colHead[8]].notnull()]
tableOfKP.to_csv(table_list[2],encoding="utf-8")

tableOfL = table.loc[table[colHead[9]].notnull()]
tableOfL.to_csv(table_list[3],encoding="utf-8")


tableOfH = table.loc[table[colHead[10]].notnull()]
tableOfH.to_csv(table_list[4],encoding="utf-8")
```

###从解析页面得到的dataframe中得到每一种期货的12个月的数据表格

- 设计metal_to_csv函数，传入参数为dataframe，以及要得到的期货类型的英文缩写，筛选出对应类型期货的行，存为对应名字csv，代码如下：

```
#each kind of metal create a csv
def metal_to_csv(dtframe,metal_name):
    Head = dtframe.columns.values.tolist()
dtframe = dtframe.loc[table[Head[0]].str.contains(metal_name)]
dtframe.to_csv("metal_"+metal_name+"_data.csv",encoding="utf-8")

####Step5：数据分析（by 王杏婷）

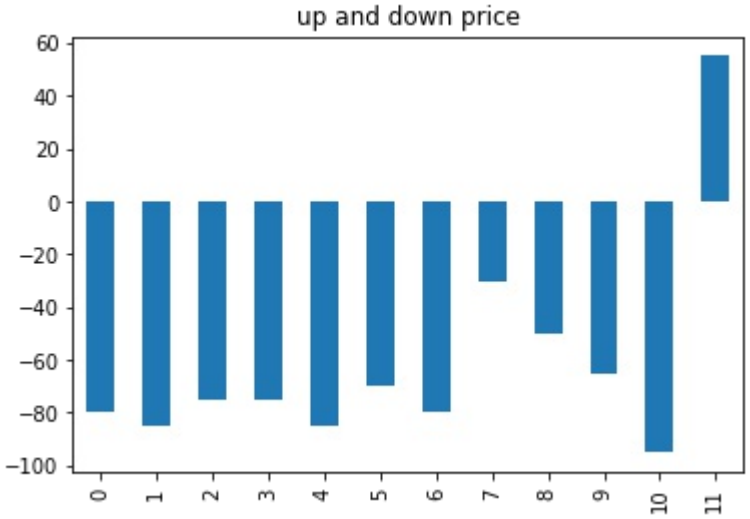
对不同的期货取出涨跌、开盘、最低、最高分析，读取每一种期货的.csv文件，用`pandas.read_csv`读取，拿到`dataframe`，然后用`matplotlib.pyplot`画图分析，将代码写成一个函数，方便拿每一种期货数据分析，具体代码如下：
    ```python
    import numpy as np
    import pandas as pd
import matplotlib.pyplot as plt
    def totalfunc(url):
    data = pd.read_csv(url)

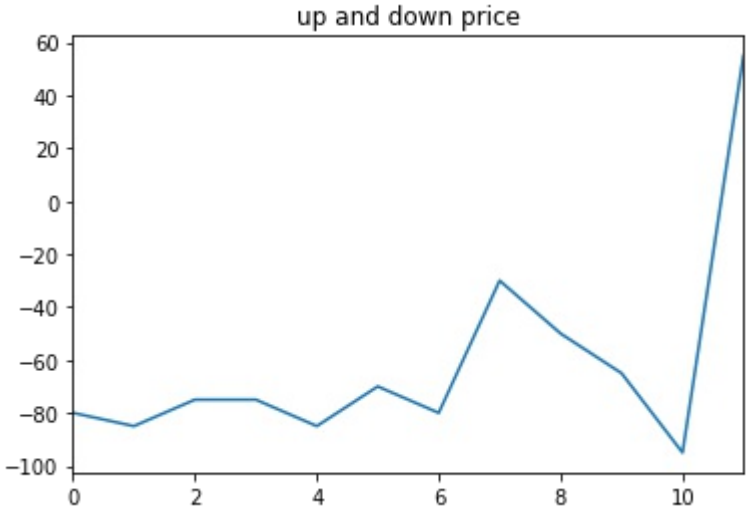
colHead = data.columns.values.tolist()
   #print type(data[colHead[1]][1])
        #data[colHead[3]]

   data[colHead[3]].plot(kind='bar')
    plt.title('up and down price')
        plt.show()
   data[colHead[9]].plot(kind='bar')
    plt.title('open_price')
        plt.show()
   data[colHead[10]].plot(kind='bar')
      plt.title('lowest')
        plt.show()
   data[colHead[11]].plot(kind='bar')
      plt.title('highest')
        plt.show()
```
```

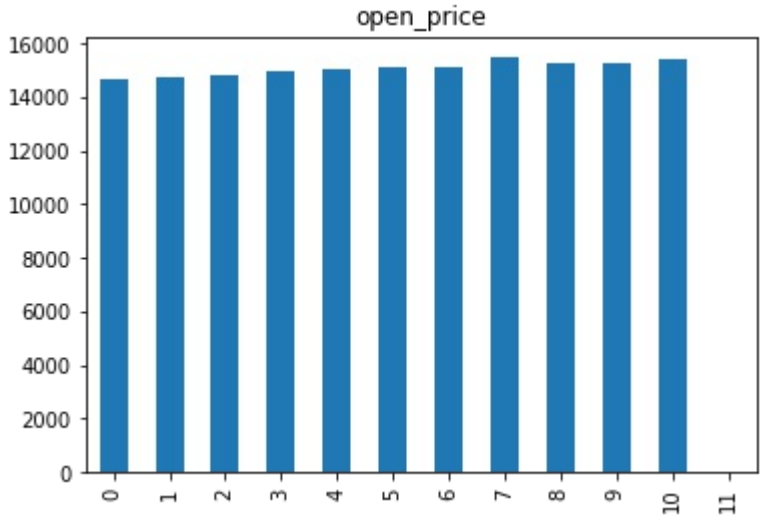
```
if __name__ == "__main__":
    url = "metal_ag_data.csv"
        totalfunc(url)
```

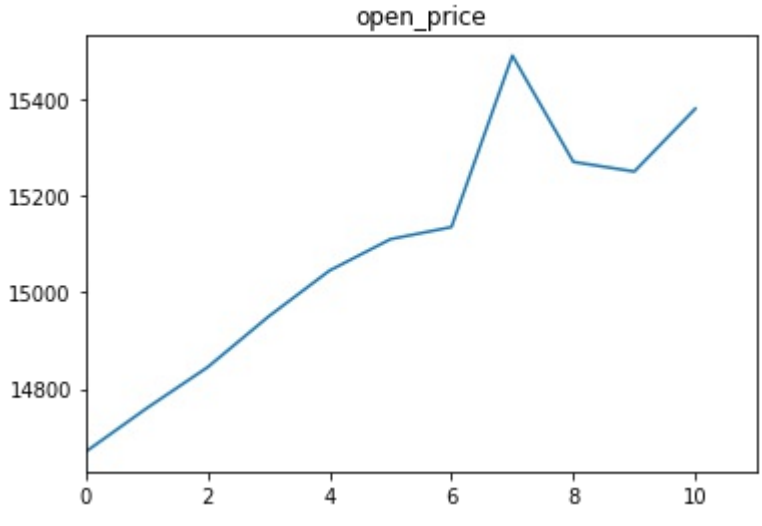
取al这种期货分析：下图柱状图和折线图分别是涨跌变化图：



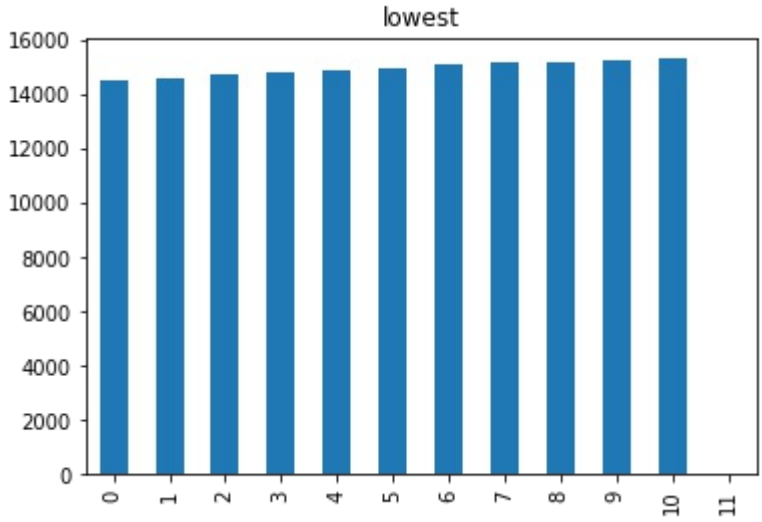


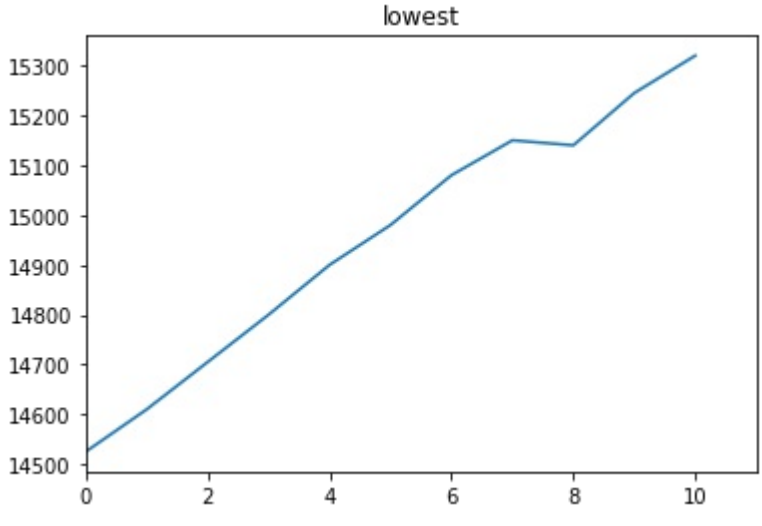
开盘价



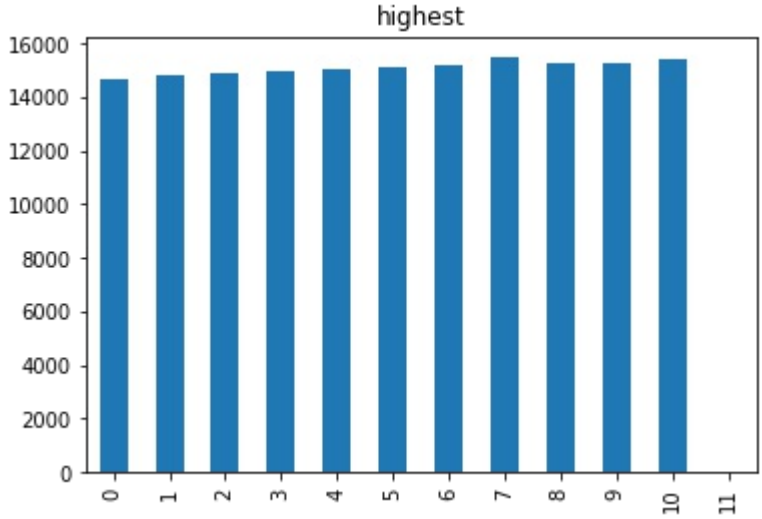

open_price
15400
15200
15000
14800
0
2
4
6
8
10

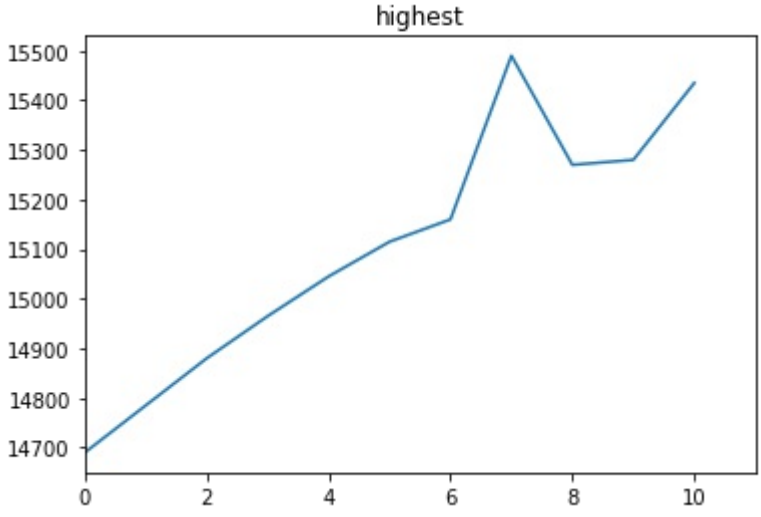
最低价格


lowest
16000
14000
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
8
9
10
11


lowest
15300
15200
15100
15000
14900
14800
14700
14600
14500
0
2
4
6
8
10

最高价格


highest
16000
14000
12000
10000
8000
6000
4000
2000
0
0
1
2
3
4
5
6
7
8
9
10
11


highest
15500
15400
15300
15200
15100
15000
14900
14800
14700
0
2
4
6
8
10