TEAM PROJECT

上海交通大學

PROJECT TITLE

JOB DATA ANALYSIS AND VISUALIZATION BASED ON ZPIN

TEAM NAME

QUATRAY KILL

TEAM MEMBERS

关若萱 517120910171

曹怡惠 517120910169

徐心娴 517120910181

夏瑞欣 517120910180

11TH MAY 2018

CONTENT

1 Program Introduction

1.1 ZPIN Introduction

ZPIN, established in1997, a large-scale company providing human resources information and service for various companies in diverse fields, is a significant reference for post-graduates as well as hundreds of thousands of job hunters, in which contains numerous information such as job introduction, required minimum monthly salaries, lowest required degree…and so on. Not to mention, ZPIN, of which average daily browsing reaches 68million, is an official and professional organization that has both talent service license*(人才服务许可证)* and labor dispatch license*(劳务派遣许可证)* issued by the government.

1.2 Problem Description

Plenteous though these information is, it still lacks systematic classification and analysis, thus makes it troublesome to find targeted information. As a result, we want to write a python program to organize information based on ZPIN, including company name, job location, recruiting numbers, job name, job monthly-salary , as well as give the job hunters a clear visualization of job-related issues, for instance, How are recruitment quantities distributed among each city?, How average salaries differ between economically developed regions and under-developed ones?...

1.3 Preparation

(1) Installation of pip packages

Scrapy: to crawl data from ZPIN and store it into a .json document.

Numpy: to provide fast precompiled functions for mathematical and numerical routines.

Pandas: to carry out entire data analysis workflow in Python without having to switch to a more domain specific language .

Matplotlib: to provide both a very quick way to visualize data from Python and publication-quality figures in many formats.

(2) Data crawling & processing

We first use scrapy to crawl four columns of data from ZPIN. On account that the data we crawled previously is still not ready for direct analysis, to better assist further analysis, we plan to process these data for accurate use, such as Invalid data deletion, data subdivision & segmentation, during which process we may apply pip packages like numpy and pandas.

(3)Data analysis & visualization

We thus plan to pair every two of these key words we've categorized above and use python(matplotlib) to draw a histogram and pie chart, taking them as dependent variable and independent variable separately, thus giving a clear sketch of current job situation.

2 Key Word Categorization

2.1 Introduction of Scrapy

There're various methods to crawl data on a web, such as requests, BeautifulSoup and scrapy. As novices here we use scrapy to grab data on ZPIN, which is a fast and high-level screen grabbing framework developed by Python.

The advantage of Scrapy is that it is a framework that anyone can easily modify it according to their needs. It also provides the basic classes of plenty kinds of reptiles, including BaseSpider, sitemap crawler, and so on.

- The universal step of using scrapy

(1) Select a web where you want to crawl data.

(2) Define grabbing Items.

(3) Write a spider to grab the data. (Including data crawl rules)

(4) Run the spider you write to grab the data.

2.2 Crawl data by Scrapy

- Select a web:

  We plan to analyze the data of employment information on investment banking from ZPIN. So we choose a web that contains the searching result: [http://sou.zhaopin.com/jobs/searchresult.ashx?jl= 全 国 &kw= 投 行 &sm=0&p=1&isfilter=0&fl=489&isadv=0]

- Open terminal

```
pip install scrapy
cd Desktop/    #change the path to find the Folder more easily.
scrapy startproject zpin    #start a project named "zpin"
```

- Then you'll find a folder named "zpin" on your desktop. Use PyCharm to open the [items.py] in the folder to definite our 4 aimed grabbing items.

```
import scrapy
class ZpinItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()
    salary = scrapy.Field()
    job = scrapy.Field ()
    company = scrapy.Field ()
    place = scrapy.Field ()
    pass
```

- After the definition of grabbing items, we need to write a spider to grab the data.

```
import scrapy
import string
from zpin.items import ZpinItem
# input the url
class zpinnn(scrapy.Spider):
    name = "zpinnn"
    allowed_domains = ["zhaopin.com"]
    start_urls = [
        "http://sou.zhaopin.com/jobs/searchresult.ashx?jl=全国&kw=投行
&sm=0&p=1&isfilter=0&fl=489&isadv=0"
    ]

    def parse(self, response):
        '''
        爬虫程序响应函数
        :param response:
        :return:
        '''
        url = response.urljoin (self.start_urls[0])
        yield scrapy.Request (url, callback=self.parse_response)

# Filter to get the work list
    def parse_response(self, response):
        job_list = response.xpath
("//div[@id='newlist_list_content_table']/table[position()>1]/tr[1]")
# Filter to get the aimed data
        for job_msg in job_list:
```

```
            job = job_msg.xpath ("td[@class='zwmc']/div/a").xpath
("string(.)").extract ()[0]
            company = job_msg.xpath ("td[@class='gsmc']/a").xpath
("string(.)").extract ()[0]
            salary = job_msg.xpath ("td[@class='zwyx']").xpath
("string(.)").extract ()[0]
            place =  job_msg.xpath ("td[@class='gzdd']").xpath
("string(.)").extract ()[0]
# Encapsulation into item objects
            item = ZpinItem ()
            item['job'] = job
            item['company'] = company
            item['salary'] = salary
            item['place'] = place
            yield item
# Turn the page of the web
        next_page = response.xpath
("//div[@class='pagesDown']/ul/li/a/@href").extract ()
        for page in next_page:
            page = response.urljoin (page)
            yield scrapy.Request (page, callback=self.parse_response)
```

2.3 Data Storage

So far, we've finished writing all the code of data crawl. However, to put it into further use, we need to store the data into a json file, which can be more convenient to analyze.

- Open terminal

```
cd Desktop/
cd zpin    # change the path
scrapy crawl zpinnn –o zpin.json –t json
```

Then we'll find a file named "zpin.json" in the "zpin" folder.("json" is a lightweight data exchange format. It is easy to read and write. It is also easy for machine to parse and generate.)

zpin.json

3 Data Processing

3.1 Purpose of Processing

Because the data we crawled previously is still not ready for direct analysis, for example, salary data has different forms(*e.g. maximum or minimum or simply face-to-face consultation*), place data are not yet categorized into cities…To better assist further analysis, we plan to process these data for accurate use.

3.2 Salary Rearrangement

- Open terminal

```
pip install pandas
pip install numpy
pip install matplotlib
```

- Open Pycharm

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Import the packages we need
```

- Check information of df

```
df = pd.read_json("zpin.json")
# Use read_json() in pandas to read the document
 and transfer it into a DataFrame document named df.
df.info()   # Let's check the information in df.
```

- Output

```
Int64Index: 4737 entries, 0 to 4736
Data columns (total 4 columns):
company    4737 non-null object
job        4737 non-null object
place      4737 non-null object
salary     4737 non-null object
dtypes: object(4)
```

- We can see that the file has 4 columns and 4737 rolls, representing company, job, place, and salary respectively.

- Serialize 'salary' column and add 3 columns 'bottom', 'top', 'average' to store lowest salary, highest salary, and average salary respectively. Use q1, q2, q3, q4 to count the execution times of each statement in which q1 counts the times of salary such as '6000-8000 RMB/month', and q2, 'not much than 10000 RMB/month'; q3 represents other circumstances such as 'found no element' or 'discuss personally'.

- Codes:

```
import re
df['bottom'] = df['top'] = df['average'] = df['salary']
pattern = re.compile('([0-9]+)')
q1=q2=q3=q4=0
```

- Use 'try' first to execute common circumstances: salary such as '6000-8000 RMB/month' and deal with 'salary' column one by one with 'regular expression'*(正则表达式)*, storing them into three new columns.

- Use 'else' to execute other circumstances when there's no numbers such as 'found no element' or 'discuss personally', and keep the original string, that is to keep bottom=top=average='discuss personally'*(面议)*

- Codes:

```
for i in range(len(df['salary'])):
    item = df['salary'].iloc[i].strip()
    # iloc[i] can select roll 'i' data in DataFrame
    # strip() can remove blank space and line break of a string
        result = re.findall(pattern,item)
        try:
        if result:
            try:
```

```
            #If this sentence is correctly executed, result[0],result[1]
exist, such as    '6000-8000 RMB/month'
                df['bottom'].iloc[i],df['top'].iloc[i] = result[0],result[1]
                df['average'].iloc[i] =
str((int(result[0])+int(result[1]))/2)
                q1+=1

            except:
            # If this sentence is correctly executed, result[0] exists,
result[1]does not, such as'<=10000 RMB/month'
                df['bottom'].iloc[i] = df['top'].iloc[i] = result[0]
                df['average'].iloc[i] =
str((int(result[0])+int(result[0]))/2)
                q2+=1

      else:
      # If this sentence is correctly executed, there's  no numbers in
【salary】, such as 'found no element' or 'discuss personally' .
          df['bottom'].iloc[i] = df['top'].iloc[i] = df['average'].iloc[i] =
item
              q3+=1

      except Exception as e:
          q4+=1
          print(q4,item,repr(e))
```

- Test if salary matches 'top' and 'bottom'.

```
for i in range(100):#
    print(df.iloc[i][['salary','bottom','top','average']])
```

- Output:

```
salary     2001-4000
bottom          2001
top             4000
average       3000.5
Name: 0, dtype: object
```

Obviously, we get 99 results as above.

- Next, we print the result.

Code:

```
df[['salary','bottom','top','average']].head(99)
```

- Output:

	salary	bottom	top	average
0	2001-4000	2001	4000	3000.5
1	8001-10000	8001	10000	9000.5
2	3500-4500	3500	4500	4000.0
3	6000-9000	6000	9000	7500.0
4	8001-10000	8001	10000	9000.5
5	20001-30000	20001	30000	25000.5
6	4001-6000	4001	6000	5000.5
7	6001-8000	6001	8000	7000.5
8	100001-150000	100001	150000	125000.5
9	15000-30000	15000	30000	22500.0
24	6001-8000	6001	8000	7000.5
25	10000-15000	10000	15000	12500.0
26	8000-15000	8000	15000	11500.0
27	面议	面议	面议	面议
28	8000-15000	8000	15000	11500.0
29	5000-8000	5000	8000	6500.0
...	...	...	...	...
69	4001-6000	4001	6000	5000.5
70	15000-30000	15000	30000	22500.0
71	15000-30000	15000	30000	22500.0
72	5000-8000	5000	8000	6500.0

3.3 City Rearrangement

Next comes to the process of working place rearrangement.

- First take a look at the [place] column and the related working data amounts.

```
df.place.value_counts()
```

- Output:

```
北京          858
上海          525
深圳          324
广州          247
北京-朝阳区     141
杭州          126
成都          124
郑州           89
济南           73
武汉           71
南京           67
深圳-福田区      62
天津           57
苏州           56
厦门           51
青岛           48
广州-天河区      47
北京-海淀区      45
上海-浦东新区    44
西安           43
合肥           40
重庆           37
大连           34
福州           32
石家庄          31
长沙           30
深圳-南山区      29
南昌           28
北京-西城区      28
佛山           27
...
上海-金山区      1
平凉             1
双城             1
青岛-黄岛区（新行政区）    1
梧州             1
印度尼西亚       1
南通-港闸区      1
咸宁             1
无锡-南长区      1
晋中             1
邵阳             1
西安-莲湖区      1
盐城-亭湖区      1
成都-天府新区    1
宜宾             1
伊春             1
乌兰察布         1
淮安-清河区      1
安阳             1
松原             1
黄石             1
昆明-盘龙区      1
成都-郫都区      1
成都-龙泉驿区    1
烟台-芝罘区      1
兰州-七里河区    1
Name: place, Length: 404, dtype: int64
```

Since the outputs include both working cities and working city-districts. For further detailed analysis, we add to a new column named ‘city’ to transfer the data like ‘Beijing-Chaoyang district’, ’Beijing-Xicheng district’ into ‘Beijing’ as a whole in the ‘city’ column.

- Add new [city] column

```
import re
df['city'] = df['place']    #add the 【city】 numpy
pattern2 = re.compile('(.*?)(\-)')     #compile regular expression patter
df_city = df['place'].copy()
for i in range(len(df_city)):
    item = df_city.iloc[i].strip()    #index by line number
    result = re.search(pattern2,item)
    #re.search function match pattern in the string, and will return when finding the first matching string, if no string matches, it will return None
    if result:
        print(result.group(1).strip())
        df_city.iloc[i] = result.group(1).strip()
    else:
        print(item.strip())
        df_city.iloc[i] = item.strip()
df['city'] = df_city
```

- Examine

To examine if the new column is correct, we print both the new [city] column and the original [place] column to see if they match each other or not.

```
df[['city',"place"]]
```

- Outputs:

	city	place
0	呼和浩特	呼和浩特
1	武汉	武汉
2	武汉	武汉
3	北京	北京
4	广州	广州
5	杭州	杭州
6	南京	南京
7	上海	上海
8	成都	成都
9	武汉	武汉-江岸区
10	上海	上海-普陀区
11	郑州	郑州
12	成都	成都-青羊区
13	郑州	郑州
14	北京	北京
15	天津	天津
16	广州	广州
17	广州	广州
18	广州	广州
19	广州	广州
20	上海	上海
21	成都	成都
22	成都	成都
23	福州	福州-台江区
24	成都	成都
25	深圳	深圳
26	上海	上海-浦东新区
27	深圳	深圳
28	南京	南京
29	重庆	重庆
...	...	...
4724	南通	南通
4725	深圳	深圳-南山区
4726	吉安	吉安
4727	北京	北京
4728	上海	上海
4729	深圳	深圳-福田区
4730	青岛	青岛
4731	吉林市	吉林市
4732	杭州	杭州
4733	上海	上海-闵行区
4734	北京	北京
4735	北京	北京
4736	天津	天津

4737 rows × 2 columns

4 Data Analysis & Visualization

Apart from the concrete company names and job names, the key words we pay attention to mainly include: average monthly salary, working city and related job descriptions (which will later turn into a related word cloud by using python). So far, we have processed two columns about [average salary] and [city], and try to combine them with jobs separately to produce a variety of data for analysis. For instance, how job opportunities concerning investment banks distributed among the country, whether most of the job recruitments concentrate in Beijing, Shanghai, Guangzhou and Shenzhen, and whether the average monthly salary in these four cities is higher than others as well?

4.1 Job Distribution

Although from the previous process of data crawling, we have obtained over 4 thousand job-related data in nearly 3 hundred cities, we later come to a conclusion by integrating the data that the number of related jobs in Wuhan which ranks eight is already smaller than one hundred, and the number is even smaller than fifty when it comes to the cities ranking after 15. As a consequence, we decide to select the top 15 cities to make further visualization.

- Data slicing

First we slice the previous [city] column and print the sorted numpy to get the data we want. The city ends up to Xi'an, the last city of which the number of job opportunities is above 50.

```
from pylab import mpl
mpl.rcParams['font.sans-serif'] = ['SimHei']
```

```
df['city'].value_counts()[:15].keys().tolist()
print(df['city'].value_counts()[:15].index.tolist())
print(df['city'].value_counts()[:15].values.tolist())
```

- Outputs:

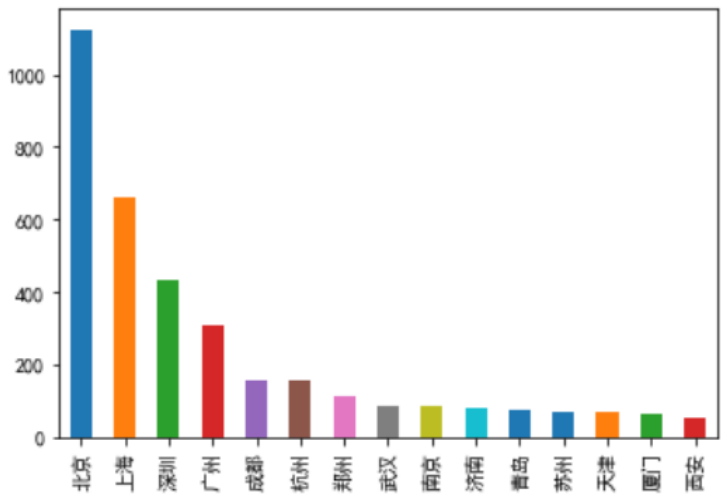
```
['北京', '上海', '深圳', '广州', '成都', '杭州', '郑州', '武汉', '南京', '济南'
, '青岛', '苏州', '天津', '厦门', '西安']
[1124, 662, 432, 311, 156, 154, 113, 87, 86, 79, 75, 72, 72, 66, 54]
```

- Visualization-Bar graph

Then we use the plot statement to make the corresponding bar graph directly.

```
df['city'].value_counts()[:15].plot.bar()
```

- Output:



北京	1124
上海	662
深圳	432
广州	311
成都	156
杭州	154
郑州	113
武汉	87
南京	86
济南	79
青岛	75
苏州	72
天津	72
厦门	66
西安	54

- Visulization-Pie chart

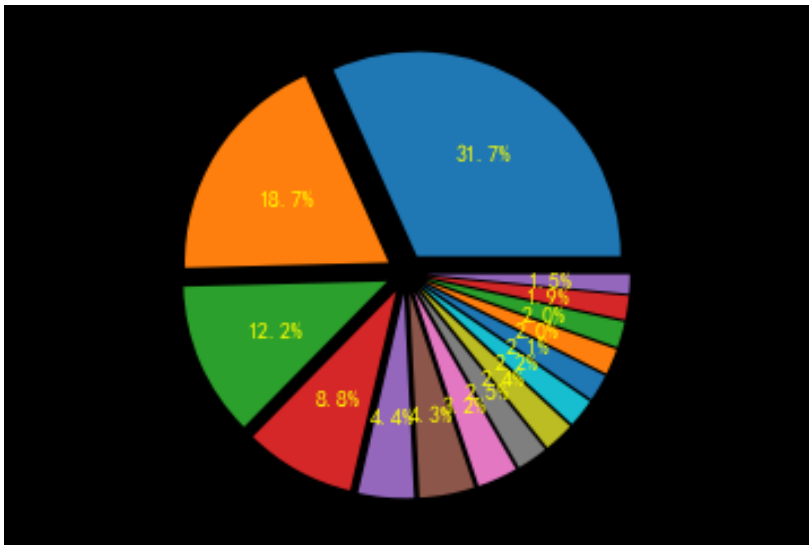
As an additional chart, we also make a pie chart to vividly demonstrate the proportion of job opportunities concerning investment banks of each city. We directly set a canvas , and draw the pie chart on the canvas (we choose a round pie chart as to observe conveniently) to show the proportion of each city.

```
fig3 = plt.figure(1,facecolor = 'white')
#set a canvas for our diagram

ax3 = fig3.add_subplot(1,1,1)
x3 = df['city'].value_counts()[:15].values.tolist()
#x is the list of numerical value, the proportion of pie chart depends on
the proportion of the values of df['city']
```

```
explode =
tuple([0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1])
plt.pie(x3,explode=explode,autopct='%1.1f%%',textprops={'color':'yellow'})
plt.axis('equal') #exhibit the pie chart as a rotundity
```

- Output:



4.2 Average monthly salary

Now that Beijing, Shanghai, Guangzhou and Shenzhen have accounted for most of the job opportunities in the country, what we take into account next is whether the average monthly salary will also overtly overcome other small cities. In this part of visual operation, we need to construct a DataFrame including two columns [city] and [average salary], and process it with groupby operation.

- DataFrame constructing

```
df_average = pd.to_numeric(df_average)
#change the data form of df_average to 'float'

df4=pd.DataFrame(data={'city':df_city,'average':df_average})
df4.info()

grouped4 = df4['average'].groupby(df4['city'])
```

```
#divide the average salary into groups, according to the corresponding
cities
grouped4.mean()
#check the average salary of each city

s4 = pd.Series(data = {'average':df_average.mean()})
result4 = grouped4.mean().append(s4)
#add a new average number which represents all the average salary that
meaningful, that is to say, higher than zero

df_ranking=result4.sort_values(ascending=False).round(1)[:30]
print(df_ranking)
#sort the average salary (rounded up to one decimal place only) in the
ascending sort, and remain the cities whose average salary rank the top 30
```

- Output:

```
孝感      67500.2
松原      60000.5
四平      60000.5
公主岭     60000.5
菏泽      60000.5
双城      60000.5
邢台      60000.5
咸宁      60000.5
舟山      60000.5
张家口     60000.5
十堰      60000.5
黄冈      60000.5
湖州      60000.5
满洲里     60000.5
黄石      60000.5
鄂州      60000.5
赣州      48125.4
三亚      45000.2
宜春      43125.4
新余      42750.2
莱芜      42500.5
德州      42500.5
晋城      42500.5
景德镇     42500.5
朔州      42500.5
赤峰      42500.5
枣庄      42500.5
牡丹江     42500.5
dtype: float64
```

- Visualization-Histogram

Like the previous step, we directly set a canvas, and the canvas is further provided with a subgraph, coordinate axis and its title, a general chart and some related illustrations in the form of histogram.

```
plt.style.use('dark_background')
fig4 = plt.figure(4)
ax4 = fig4.add_subplot(1,1,1)
#set the canvas for the following diagram
df_ranking.sort_values(ascending=False).round(1).plot(kind='bar',rot=30)
```

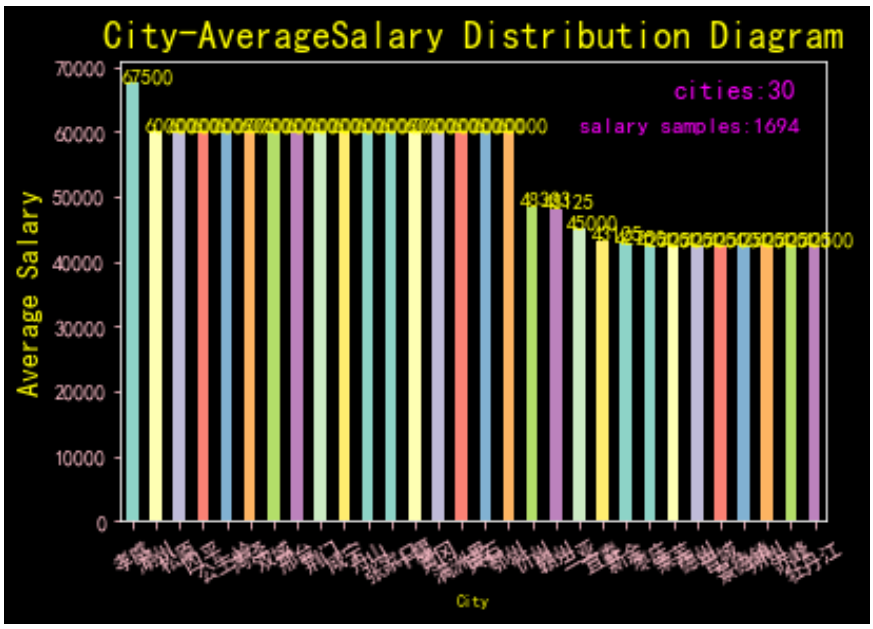
```
#set the title of the diagram, axis x and axis y
title = plt.title(u'City-AverageSalary Distribution
Diagram',fontsize=18,color='yellow')
xlabel = plt.xlabel(u'City',fontsize=8,color='yellow')
ylabel = plt.ylabel(u'Average Salary',fontsize=14,color='yellow')

#set the explanation, located in the top right corner
text1 = ax4.text(23,65000,u'cities:30',fontsize=13, color='#FF00FF')
text2 = ax4.text(19,60000,u'salary samples:1694',fontsize=11,
color='#FF00FF')

#add the coordinate position of each city
list_4 = df_ranking.sort_values(ascending=False).values
for i in range(len(df_ranking)):
    ax4.text(i-0.5,list_4[i],int(list_4[i]),color='yellow')

plt.tick_params(colors='pink')
```

- Output:



孝感	67500.2
松原	60000.5
四平	60000.5
公主岭	60000.5
菏泽	60000.5
双城	60000.5
邢台	60000.5
咸宁	60000.5
舟山	60000.5
张家口	60000.5
十堰	60000.5
黄冈	60000.5
湖州	60000.5
满洲里	60000.5
黄石	60000.5
鄂州	60000.5
赣州	48125.4
三亚	45000.2
宜春	43125.4
新余	42750.2
莱芜	42500.5
德州	42500.5
晋城	42500.5
景德镇	42500.5
朔州	42500.5
赤峰	42500.5
枣庄	42500.5
牡丹江	42500.5

dtype: float64

5 Analysis of result

5.1 Job & Cities

The graph and the chart confirm the previous guess: the job opportunities in Beijing, Shanghai, Guangzhou and Shenzhen occupy almost 71.4% of the total. Among all, Beijing undoubtedly tops the list with a surprising figure of 1124, the only city in the country that occupies more than one thousand investment bank-related jobs; Shanghai follows, but the number of jobs is less than six out ten than those in Beijing.

5.2 Salary & Cities

As we can see, under the key word "investment bank"(投行), the average monthly salary of 249 samples around the country varies distinctly. The cities with the top 5 average monthly salary are xiaogan(孝感), songyuan(松原), siping(四平), gongzhuling(公主岭) and heze(菏泽). Thus we come to the conclusion that may surprise us all: Beijing, Shanghai, Guangzhou and Shenzhen only have an absolute advantage in terms of job opportunities, but they fail to continue leading the nation in average monthly salary. At the same time, we also find a number of unexpected cities of relatively considerable average salaries like the top five (even the top 30), and we may draw the conclusion that this strange phenomenon may arise from the fact that our datum is limited ; that is to say, we may only have scraped one or two sample from city like heze related to the key word "investment bank", while the only opportunity there offers an extremely high salary. However, as we may see from the data, the average salary in shanghai is more than 20,000, which is obviously a decent pay among all the works in such high-level city. So our analysis does exhibit its weakness as for few would like to work in small cities for the sake of higher salary, but it also raises another opportunity for those who are always seeking for jobs in first-line cities.

6 Conclusion

With the help of python, the powerful and accessible programming language, we are able to solve problems more complex and practical than we could imagine, e.g. large sum of data analysis. In addition, our teamwork skill and our organization and coordination capabilities are also improved during the project. By the way, we would thank our tutor, Mr. Bao, for giving us abundant and useful suggestions and support whenever we have problems, his profound knowledge of computing sparks our confidence and interest in python.

Appendix:

Distribution of duties:

关若萱 (517120910171) : Code/Report/PPT/Presentation

Ability to work independently， mature and resourceful.

曹怡惠 (517120910169): Code/Report/PPT/Presentation

A stable personality and high sense of responsibility.

徐心娴 (517120910181) : Code/Report/PPT/Presentation

Mature, self-motivated and good ability of systematical thinking.

夏瑞欣 (517120910180): Code/Report/PPT/Presentation

Initiative， independent and good communication skill.