**题目：招聘网爬取、可视化与机器学习应用**

**一、完成人**

|  |  |  |  |
| --- | --- | --- | --- |
| **姓名** | **学号** | **分工** | **占比** |
| 康育鑫 | SA19225203 | 写爬虫，爬取网页 | 1/3 |
| 周天逸 | SA19225509 | 数据可视化分析 | 1/3 |
| 苟小飞 | SA19225149 | 机器学习应用 | 1/3 |

**二、设计背景**

作为工程硕士，大部分同学的毕业去向主要是就业，但是有些同学对就业市场的了解还不够充分。因此，本项目爬取了“前程无忧”网站“Java开发”岗位的招聘信息，对数据进行可视化展示及分析，并建立了预测期望薪资的机器学习模型，来帮助同学们加深对就业形势的认识。

**三、设计目标**

本项目旨在设计出爬取效率较高的爬虫程序、充分而生动的可视化展示、误差较低的期望薪资预测模型。

**四、技术路线**

使用threading和queue进行多线程爬取

使用lxml中的xpath进行定位；

使用pyecharts及matplotlib.pyplot对数据进行可视化操作

使用pandas对数据进行处理；

使用sklearn中操作数据集的函数及机器学习算法函数；

使用matplotlib画出描述数据的图像。

**五、具体实现**

**5.1 网络爬虫模块**

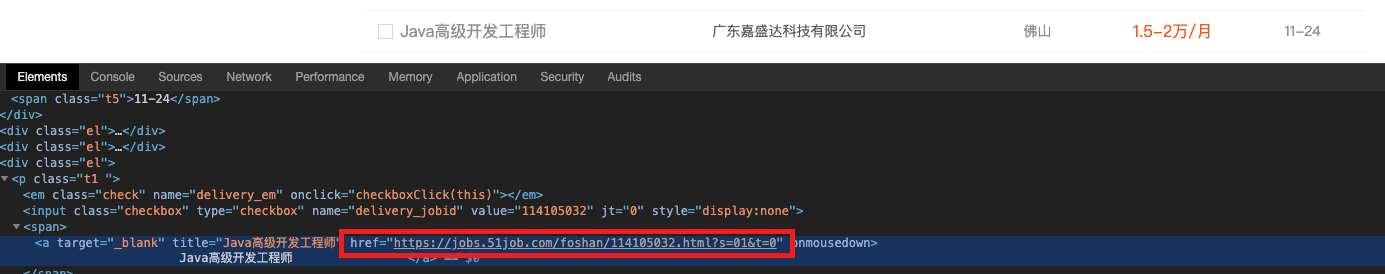
起初使用的是单线程爬虫爬取前程无忧网站前200页关于Java职位的数据（虽然有1457页，但发现245页以后就不是Java职位了，所以就爬前200页数据），每页大概有50条，总共将近一万条数据，如下图所示。





**总体思路：**

由于搜索首页只显示“职位名，公司名，工作地点，薪资，发布时间”这几个数据，而要给后面数据可视化和分析爬取的数据应该有“职位名，公司名，工作地点，薪资，经验要求，教育要求，招聘人数，公司性质和公司规模”。所有我要先爬取职位名的链接（即下图所示），再打开这个链接，爬取我所需要的全部数据如图所示。





**5.1.1 单线程爬虫**

思路：

Page url list

发送请求获取响应

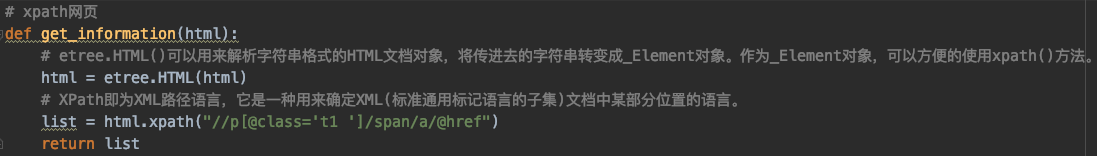
提取数据

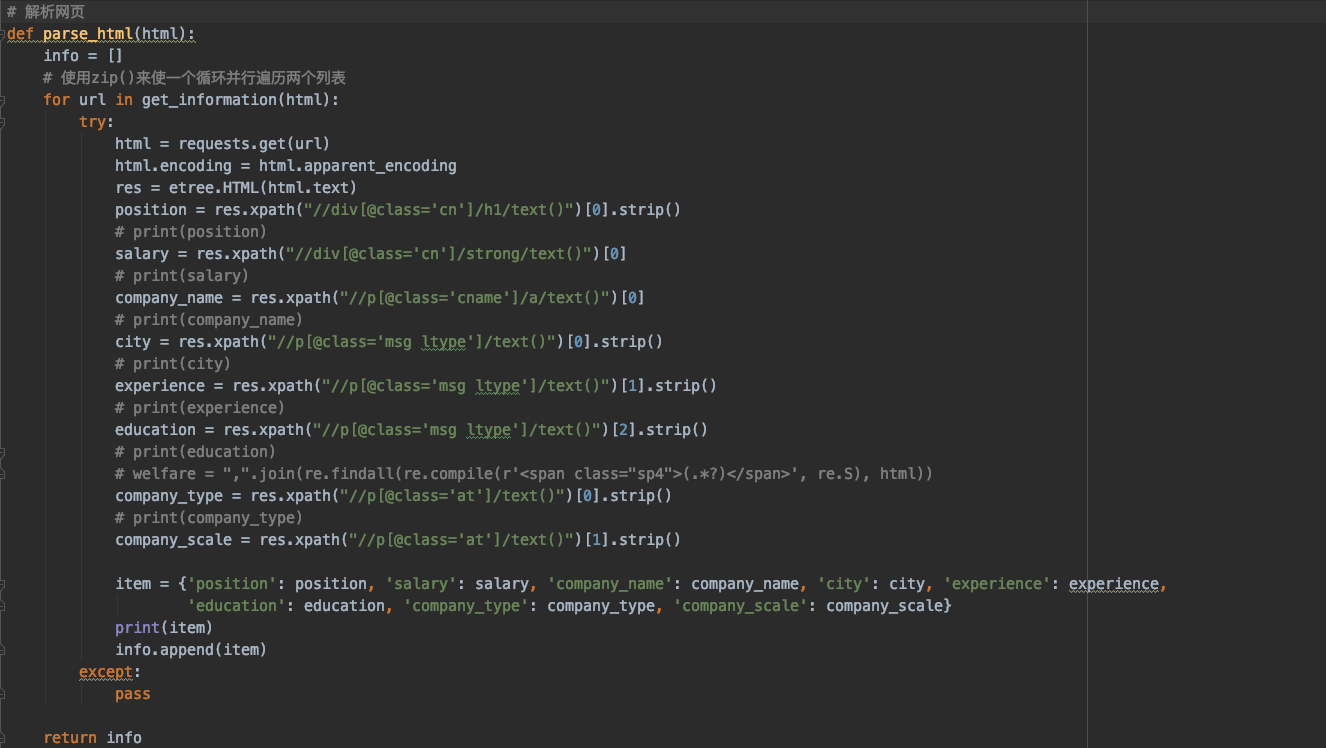
保存

用到的库：requests，lxml的etree，pandas，datetime

代码实现：（只贴关键代码）





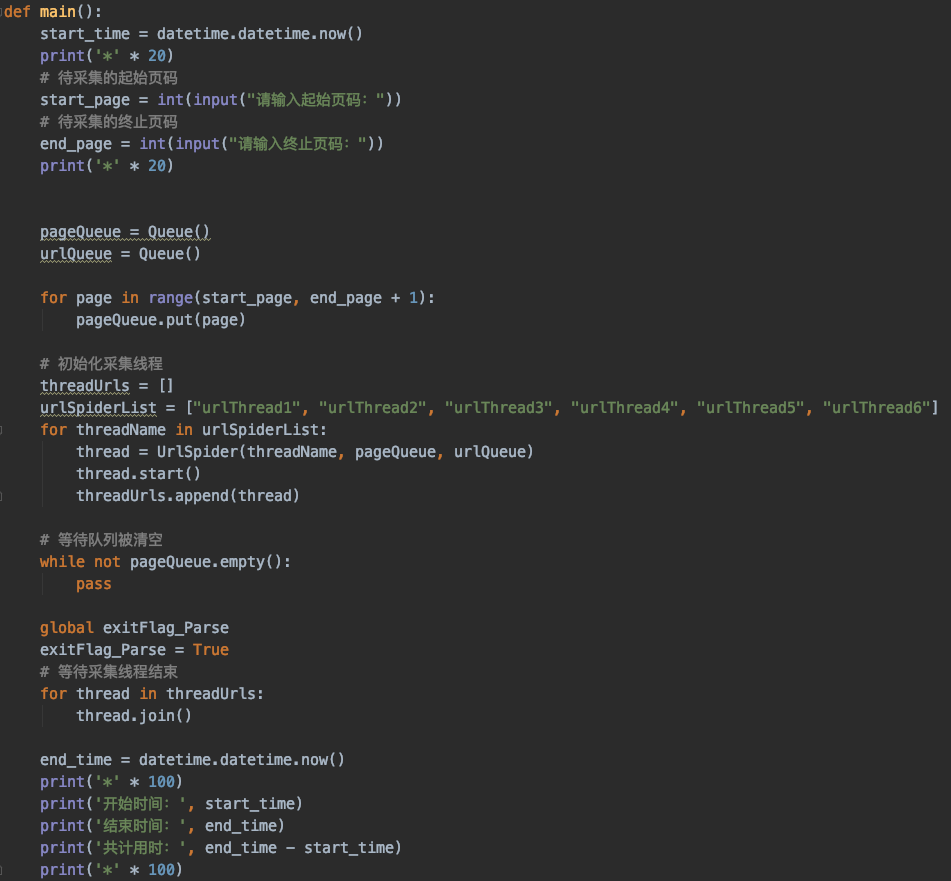


5.1.2 多线程爬取职位URL检验多进程是否能提速

用到的库：threading，queue，lxml，requests，datetime

代码实现：

****



5.1.3 多线程拆分为两个类的爬虫

思路：定义两个类，一个用来爬取职位URL，一个用来解析获取数据并保存。

Page url list

发送请求获取响应

提取数据

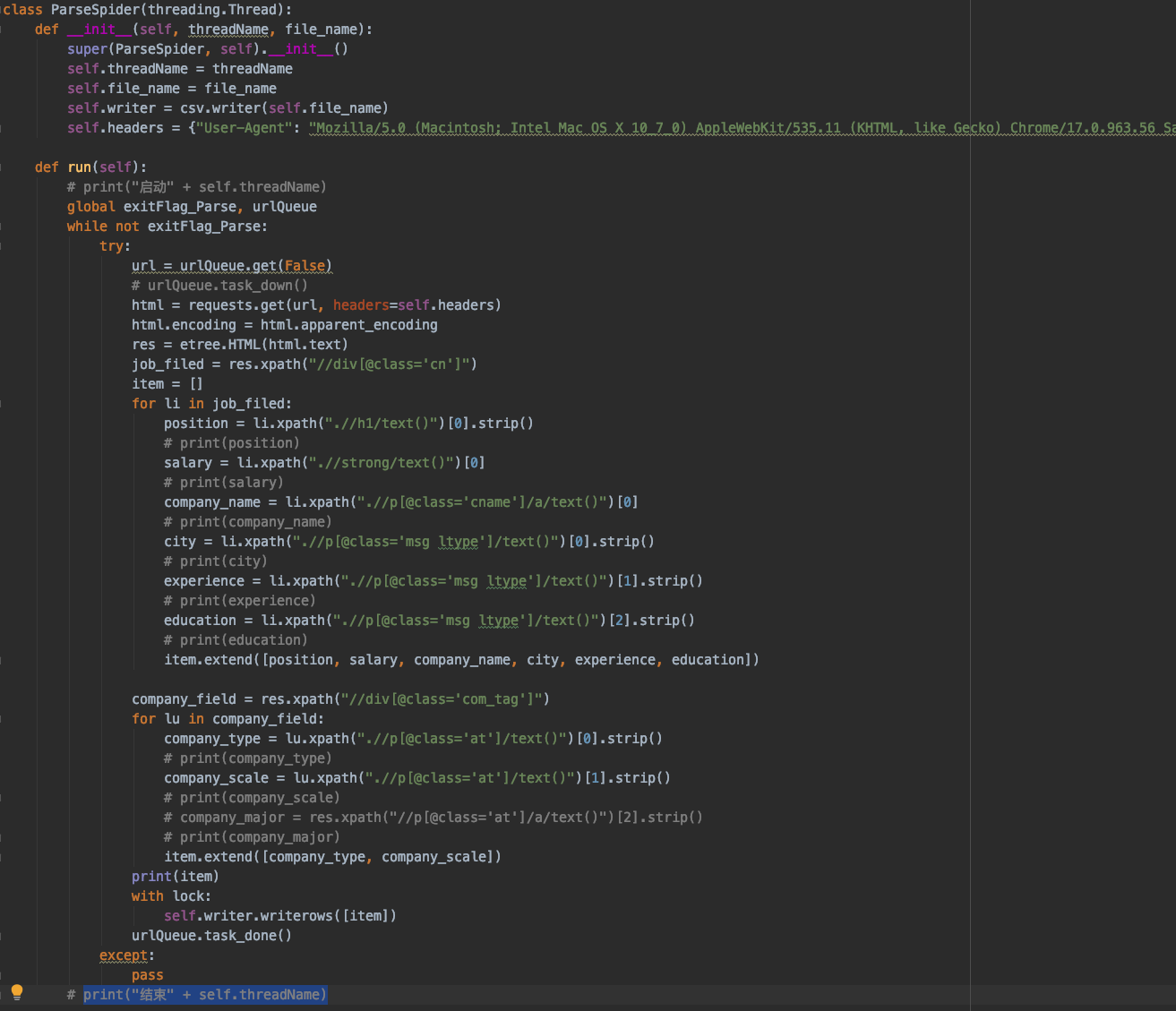
保存

职位URL队列

用到的库：thread，queue，lxml， requests，csv，datetime

代码实现：





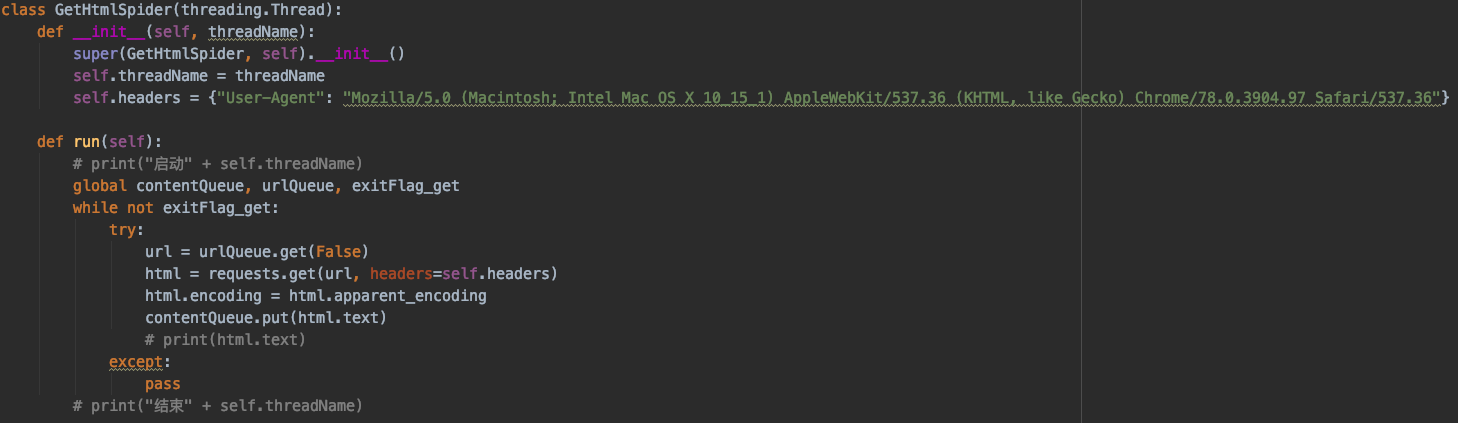
5.1.4 多线程拆分为三个类的爬虫

代码实现：就把页面解析类拆分成：页面解析+写入保存类

****

5.1.5 多线程拆分为四个类的爬虫

代码实现：获取职位URL类，获取HTML内容类，页面解析类，写入保存类

****

5.2 数据清洗

在获取爬虫从网站爬取的数据后，首先进行数据分析，若不进行仔细全面的数据分析直接进入清洗阶段很容易出现遗漏导致后期工作出错，拖慢工作进程。

直接爬取的数据关系如表5-1所示，但刚爬取的数据格式及类型不一定完全符合，下图是清洗后的数据目标。

表5-1 数据关系表

|  |  |  |  |
| --- | --- | --- | --- |
| 字段名称 | 类型 | 允许为空 | 说明 |
| index | int | NOT NULL | 编号 |
| position | varchar | NOT NULL | 职位 |
| salary | varchar | NOT NULL | 薪水 |
| company\_name | varchar | NOT NULL | 公司名称 |
| city | varchar | NOT NULL | 所在地 |
| exprience | varchar | NOT NULL | 工作经历要求 |
| education | varchar | NOT NULL | 学历要求 |
| company\_type | varchar | NULL | 公司类型 |
| company\_scale | varchar | NULL | 公司规模 |

(1) index，编号字段，不必保留，对数据进行清洗时必定会删除一些不合的条目导序号不连续。

(2) position,职位字段，由于各大招聘网站上招聘职位具有很大的随意性，如：JAVA工程师，Java开发工程师，Java高级开发工程师等。上述三个岗位其实都是同一个职位，故根据统计各职位出现数量，将职位进行统一化，以便后期的数据统计分析与预测，统一格式化职位如表5-2所示。

表5-2 统一格式化职位表

|  |
| --- |
| JAVA实习生 |
| JAVA架构师 |
| JAVA项目经理 |
| JAVA培训讲师 |
| JAVA高级/资深/全栈开发工程师 |
| JAVA中级开发工程师 |
| JAVA开发工程师 |

清洗数据前期工作，将info.csv（爬取后的粗数据）读入，并将格式转换为DataFrame格式，代码如下：

data =pd.read\_csv(r'info.csv',header = 0,encoding="utf-8")

result = pd.DataFrame(data)

Position字段部分清洗实现代码，依据权值对各职位确定先后，以防原职位中出现非权重字段导致岗位区分错误，如JAVA实习架构师，最终将其划分为实习生而不是架构师，代码如下：

a1 = u"实习"

a2 = u"架构"

a31=u"经理"

a32=u"主管"

a41=u"讲师"

a42=u"老师"

a51=u"资深"

a52=u"高"

a53=u"全栈"

a6=u"中"

li\_SX = result['position']

for i in range(0,len(li\_SX)):

try:

if a1 in li\_SX[i]:

li\_SX[i] = format("JAVA实习生")

elif a2 in li\_SX[i]:

li\_SX[i] = format("JAVA架构师")

elif a31 in li\_SX[i] or a32 in li\_SX[i]:

li\_SX[i] = format("JAVA项目经理")

elif a41 in li\_SX[i] or a42 in li\_SX[i]:

li\_SX[i] = format("JAVA培训讲师")

elif a51 in li\_SX[i] or a52 in li\_SX[i] or a53 in li\_SX[i]:

li\_SX[i] = format("JAVA高级/资深/全栈开发工程师")

elif a6 in li\_SX[i]:

li\_SX[i] = format("JAVA中级开发工程师")

else:

li\_SX[i] = format("JAVA开发工程师")

except:

pass

(3)salary，薪水字段，因数据未统一化，爬取的数据单位共有四种，万/月、千/月、万/年以及元/日，我们以万/月为标准，将数据进行格式统一化，代码如下：

b0 = u'万/月'

b1 = u'千/月'

b2 = u'万/年'

b3 = u'元/天'

li\_salary = result['salary']

for i in range(0,len(li\_salary)):

try:

if b0 in li\_salary[i]:

x=re.findall(r'\d\*\.?\d+',li\_salary[i])

li\_salary[i] = str(format(float(x[0]),'.2f')+'-'+format(float(x[1]),'.2f'))

if b1 in li\_salary[i]:

x=re.findall(r'\d\*\.?\d+',li\_salary[i])

#print('1.'+x[0]+'-'+x[1])

min\_ = format(float(x[0])/10,'.2f')

max\_ = format(float(x[1])/10,'.2f')

li\_salary[i] = str(min\_+'-'+max\_)

if b2 in li\_salary[i]:

x=re.findall(r'\d\*\.?\d+',li\_salary[i])

#print('2.'+x[0]+'-'+x[1])

min\_ = format(float(x[0])/12,'.2f')

max\_ = format(float(x[1])/12,'.2f')

li\_salary[i] = str(min\_+'-'+max\_)

if b3 in li\_salary[i]:

x = re.findall(r'\d\*\.?\d+',li\_salary[i])

min\_ = format(float(x[0])\*28/10000,'.2f')

max\_ = format(float(x[0])\*31/10000,'.2f')

li\_salary[i] = str(min\_+'-'+max\_)

except:

pass

(4)company\_name，公司名称字段，若为空，则直接删除该条目，实现代码如下：

indexs = list(result[np.isnan(result['company\_name'])].index)

result = result.drop(indexs)

(5)city,所在地字段，爬取的数据有的只出现省份，有的只是一个市，有的还详细到了所在区所在街道，我们将以市为标准进行格式统一化，如：杭州-余杭区-文一西街道转换为杭州，以市为单位，实现代码如下：

b1 = u'省'

b2 = u'雄安新区'

li\_city = result['city']

for i in range(0,len(li\_city)):

try:

x=re.findall(r'^\w+',li\_city[i])

li\_city[i] = str(x[0])

if b1 in li\_city[i]:

result = result.drop(i,axis=0)

if b2 in li\_city[i]：

li\_city[i] = str('雄安')

except:

pass

for i in range(0,len(li\_city)):

print(li\_city[i])

(6) experience，工作经历字段，若为空，则直接删除该条目，实现代码如下：

indexs = list(result[np.isnan(result['exprience'])].index)

result = result.drop(indexs)

(7) education,学历字段，在爬取的数据中该字段出现‘招多少人’等与之无关的信息，我们要将之删除，代码如下：

b = u'人'

li\_education = result['education']

for i in range(0,len(li\_education)):

try:

if b in li\_education[i]:

result = result.drop(i,axis = 0)

except:

pass

最后将清洗的数据导出：

result.to\_csv('info\_cleaning.csv',header = 1 ,encoding="utf-8",index=0\

,columns=['position','salary','company\_name','city','experience','education',\

'welfare','company\_type','company\_scale','company\_major'])

5.3数据可视化

对数据进行清洗后，将数据进行可视化，以便于后面对数据进行分析。

导入包：

import pandas as pd

import numpy as np

import re

from pyecharts import Pie,Geo,WordCloud,Bar3D,Line

import matplotlib.pyplot as plt

5.3.1数据预处理

file =pd.read\_csv(r'info\_cleaning.csv',header = 0,encoding="utf-8")

f = pd.DataFrame(file)

pd.set\_option('display.max\_rows',None)

pos = f['position']

sal = f['salary']

cN= f['company\_name']

ct = f['city']

exp= f['experience']

edu = f['education']

wel = f['welfare']

cT=f['company\_type']

cS = f['company\_scale']

cM = f['company\_major']

#初始化操作

position =[]

salary = []

companyName = []

city = []

experience =[]

education =[]

welfare = []

companyType = []

companyScale =[]

companyMajor = []

for i in range(0,len(f)):

try:

position.append(pos[i])

companyName.append(cN[i])

city.append(ct[i])

experience.append(exp[i])

education.append(edu[i])

welfare.append(wel[i])

companyType.append(cT[i])

companyScale.append(cS[i])

companyMajor.append(cM[i])

#print(position[i])

s = re.findall(r'\d\*\.?\d+',sal[i])

salary.append([float(s[0]),float(s[1])])

except:

pass

因为从文件中读取的数据都是pandas.core.series.Series类型，若想要操作必须将之转换为list类型,用多个list来单独存放各字段信息。

5.3.2 matplotlib.pyplot 线状图

def linearGraph(str,object,order):

plt.rcParams['font.family'] = 'STSong'

plt.rcParams['font.size'] =10

min\_s=[]

max\_s=[]

for i in range(0,len(experience)):

min\_s.append(salary[i][0])

max\_s.append(salary[i][1])

my\_df = pd.DataFrame({str:object,'min\_salary':min\_s,"max\_salary":max\_s})

data = my\_df.groupby(str).mean()['min\_salary']

data = data[order]

data.plot(kind='line')

plt.savefig(str+"-工资线状图.jpg")

plt.show()

order=['无工作经验','1年经验','2年经验','3-4年经验','5-7年经验','8-9年经验','10年以上经验']

linearGraph('experience',experience,order)

order2=['初中及以下','中专','大专','高中','本科','硕士']

linearGraph('education',education,order2)

linearGraph()函数中输入str为关系表中任意字段，object为str所对应的list，order需为str字段所有value，使折线图x轴按order顺序。该函数中先设置了pyplot的设置，并获取了所有薪水，因为某一职位薪水的格式为12-13，其中12为最低，13为最高薪水，使用Dataframe.groupby分组方法对最低薪水进行求期望并根据str进行聚合分组。最后通过plt.savefig保存至当前文件中。

5.3.3 pyecharts.Pie环图

#显示列属性值以及出现的数量

def columnsAndCount(list):

list\_temp={}

for i in set(list):

list\_temp[i]=list.count(i)

return list\_temp

#3-玫瑰环图

def ringFigure(str,object):

obj=columnsAndCount(object)

attr = obj.keys()

value = obj.values()

pie =Pie(str,width=1000,height=500)

pie.add(str,attr,value,\

center=[70,50],is\_random=False,\

radius=[30,75],rosetype='radius',\

is\_lengend\_show=True,is\_label\_show=True,is\_legend\_orient='vertical')

pie.render(str+'玫瑰环图.html')

ringFigure('学历要求',education)

columnsAndCount(list)函数中输入list,通过统计list中value的属性以及使用count()方法对该属性的出现次数进行计数，最后返回一个属性与次数的分组list。

ringFigure(str,object)函数中输入str为关系表中任意字段，object为str所对应的list。在函数中将object的key以及value进行了分离并调用pyecharts.Pie.add函数进行绘图，该函数中第一个参数为图的名称，其次为每个属性名、属性值大小、饼图的中心（圆心）坐标，数组的第一项是横坐标，第二项是纵坐标、饼图的半径，数组的第一项是内半径，第二项是外半径，最后通过pie.render方法保存至当前文件中

5.3.4 pyecharts.WordCloud词云

def wordloud(str,x,y):

wordcloud = WordCloud(background\_color="#white",\

width = 1300,height = 800)

wordcloud.add(str,x,y,word\_size\_range=[10,20],rotate\_step=45,shape="star")

wordcloud.render(str+"词云.html")

comValue = []

comValue = columnsAndCount(companyName)

x = []

y = []

for i in set(comValue):

x.append(i)

y.append(comValue[i])

wordloud('公司',x,y)

wordloud(str,x,y)函数中输入str为关系表中任意字段，x为对应字段中各属性值，y则为权重值这里用出现次数替代。在函数中将直接调用WordCloud（）函数进行初始化背景颜色、宽高等，在接着调用add（）函数设置各属性值，最后通过wordcloud.render方法保存至当前文件中。

5.3.5 pyecharts.Line折线图

matplotlib.pyplot 线状图的表现不如pyecharts.Line折线图，pyecharts.Line折线图可以进行多条折线对比以及外观更加好看。

def Linef(str,order,data):

line = Line(str)

line.add(str,order,data,is\_symbol\_show=True,

is\_smooth=False,

is\_stack=True,

is\_step=False,

is\_fill=True,)

line.render(str+'.html')

#5-end

min\_s=[]

max\_s=[]

for i in range(0,len(experience)):

min\_s.append(salary[i][0])

max\_s.append(salary[i][1])

my\_df = pd.DataFrame({'experience':experience,'min\_salary':min\_s,"max\_salary":max\_s})

data1 = my\_df.groupby('experience').mean()['min\_salary']

order1 =['无工作经验','1年经验','2年经验','3-4年经验','5-7年经验','8-9年经验','10年以上经验']

data1 =data1[order1]

Linef("工作经验工作线状图",order1,data1)

my\_df2 = pd.DataFrame({'education':education,'min\_salary':min\_s,"max\_salary":max\_s})

data2 = my\_df2.groupby('education').mean()['min\_salary']

order2=['初中及以下','中专','大专','高中','本科','硕士']

data2 =data2[order2]

Linef("学历线状图",order2,data2)

Linef(str,x,y)函数输入str为关系表中任意字段，data为str所对应的list，order需为str字段所有value，使折线图x轴按order顺序。

5.3.6 pyecharts.Geo热点地图

#6-Geo地图

def geoprint(str,attr,value):

geo = Geo(str,title\_color="#fff",\

title\_text\_size=24,title\_top=20,title\_pos="center",\

width=1300,height=600)

geo.add(str,attr,value,type="effectScatter",is\_random=True,visual\_range=[0,1000],\

maptype="china",symbol\_size=8,effect\_scale=5,border\_color="#000000",is\_visualmap=True)

geo.render(str+".html")

#6-end

elements\_city=[]

elements\_city=pd.DataFrame({'city':city}).groupby('city').count().index

address=elements\_city

min\_s=[]

max\_s=[]

for i in range(0,len(city)):

min\_s.append(salary[i][0])

max\_s.append(salary[i][1])

my\_df = pd.DataFrame({'city':city,'min\_salary':min\_s,"max\_salary":max\_s})

data1 = my\_df.groupby('city').mean()['min\_salary']

data2 = my\_df.groupby('city').mean()['max\_salary']

data = []

for i in range(0,len(data1)):

data.append(format(data2[i],'.2f'))

value = data

geoprint("java需求薪资分布",address,value)

geoprint(str,attr,value) 函数输入str为关系表中任意字段，attr则为地理城市，value为各城市的权值大小。通过pd.DataFrame({'city':city}).groupby('city').count().index获取了出现的各城市的名称以及次数，接着通过循环统计各岗位薪资，并通过

pd.DataFrame.groupby('city').mean()['min\_salary']来获取与各城市对应的平均薪资。

5.3.7 pyecharts. 3D柱状图

#7-3D柱状图

def threeD(str,x\_order,y\_order,z\_order):

bar3d = Bar3D(background\_color="#white",width = 1300,height = 800)

bar3d.add("城市-职位-薪水3D柱状图",\

x\_order,y\_order,z\_order,grid3d\_opacity=0.8,grid3d\_shading="realistic")

bar3d.render("城市职位薪水3D柱状图.html")

#数据处理

min\_s = []

max\_s = []

for i in range(0,len(f)):

min\_s.append(salary[i][0])

max\_s.append(salary[i][1])

poistion\_salaryMin = pd.DataFrame({'city':city,'position':position,'min\_salary':min\_s})

data1 = poistion\_salaryMin.groupby('position').mean()['min\_salary']

poistion\_salaryMax = pd.DataFrame({'position':position,'max\_salary':max\_s})

data2 = poistion\_salaryMax.groupby('position').mean()['max\_salary']

elements\_poistion = []

elements\_poistion = data1.keys()

elements\_city=[]

elements\_city=pd.DataFrame({'city':city}).groupby('city').count().index

elements\_education=[]

elements\_education=pd.DataFrame({'education':education}).groupby('education').count().index

city\_position\_salarySum=[]

for i in range(0,len(elements\_city)):

for j in range(0,len(elements\_poistion)):

city\_position\_salarySum.append([0,0.00,0.00,elements\_city[i],elements\_poistion [j]])

for i in range(0,len(f)):

for j in range(0,len(city\_position\_salarySum)):

if city[i] in city\_position\_salarySum[j] and position[i] in city\_position\_salarySum[j]:

city\_position\_salarySum[j][0] = city\_position\_salarySum[j][0]+1

city\_position\_salarySum[j][1] = city\_position\_salarySum[j][1]+min\_s[i]

city\_position\_salarySum[j][2] = city\_position\_salarySum[j][2]+max\_s[i]

break

for i in range(0,len(city\_position\_salarySum)):

if city\_position\_salarySum[i][0] != 0:

city\_position\_salarySum[i][1] = \

format(city\_position\_salarySum[i][1] / city\_position\_salarySum[i][0],'.2f')

city\_position\_salarySum[i][2] = \

format(city\_position\_salarySum[i][2] / city\_position\_salarySum[i][0],'.2f')

education\_position\_salarySum=[]

for i in range(0,len(elements\_education)):

for j in range(0,len(elements\_poistion)):

education\_position\_salarySum.append([0,0.00,0.00,

elements\_education[i],elements\_poistion[j]])

for i in range(0,len(f)):

for j in range(0,len(education\_position\_salarySum)):

if education[i] in education\_position\_salarySum[j] and \

position[i] in education\_position\_salarySum[j]:

education\_position\_salarySum[j][0] = education\_position\_salarySum[j][0]+1

education\_position\_salarySum[j][1] = education\_position\_salarySum[j][1]+min\_s[i]

education\_position\_salarySum[j][2] = education\_position\_salarySum[j][2]+max\_s[i]

break

for i in range(0,len(education\_position\_salarySum)):

if education\_position\_salarySum[i][0] != 0:

education\_position\_salarySum[i][1] = \

format(education\_position\_salarySum[i][1] / education\_position\_salarySum[i][0],'.2f')

education\_position\_salarySum[i][2] = \

format(education\_position\_salarySum[i][2] / education\_position\_salarySum[i][0],'.2f')

z\_order=[]

tem =[]

for i in range(0,len(elements\_city)):

for j in range(0,len(elements\_poistion)):

for m in range(0,len(city\_position\_salarySum)):

if elements\_city[i] in city\_position\_salarySum[m] and \

elements\_poistion[j] in city\_position\_salarySum[m]:

z\_order.append([elements\_city[i],elements\_poistion[j]\

,city\_position\_salarySum[m][1]])

break

threeD(str,elements\_city,elements\_poistion,z\_order)

首先做笛卡尔积，将数据集进行分类，获取每个城市中每种职业对应的平均工资，这是最难的，首先我们通过elements\_poistion = data1.keys()获取职位的全部种类，通过elements\_city=pd.DataFrame({'city':city}).groupby('city').count().index获取城市的全部种类，接着通过两层循环初始化city\_position\_salarySum，其类型为list,其中元素也为list，元素list中元素为([0,0.00,0.00,elements\_city[i],elements\_poistion [j]])，即职位数量、最低工资、最高工资、所在城市及职位。通过循环遍历来填写city\_position\_salarySum信息并用职位数量求最低工资、最高工资平均值，最后带入threeD(str,x\_order,y\_order,z\_order)函数建立3D柱状图。

5.4 机器学习应用部分具体实现

5.4.1. 概述

(1) 数据集为爬取的java职位招聘信息，本部分将 salary 属性作为标签，将其他属性作为输入，训练出一个预测期望薪资的模型。

(2) 本项目主要使用 sklearn 框架。

(3) 本项目以均方根差作为模型预测性能的评价标准。

5.4.2. 数据获取并可视化展示

原始数据通过爬虫程序爬取“前程无忧”招聘网站获取，数据获取及可视化展示以及在前面的部分详细说明，此处不再赘述。

5.4.3. 为机器学习算法准备数据

(1) 定义各个属性处理的函数：

salary属性: 该属性为机器学习算法的标签，由于要对其进行预测，而原始数据都是薪资的区间范围，因此我们需要将薪资区间范围转换为一个确切的值。我们使用的方法是取区间端点的平均值，单位为（万元/月），并从数据集中分离出来作为标签。

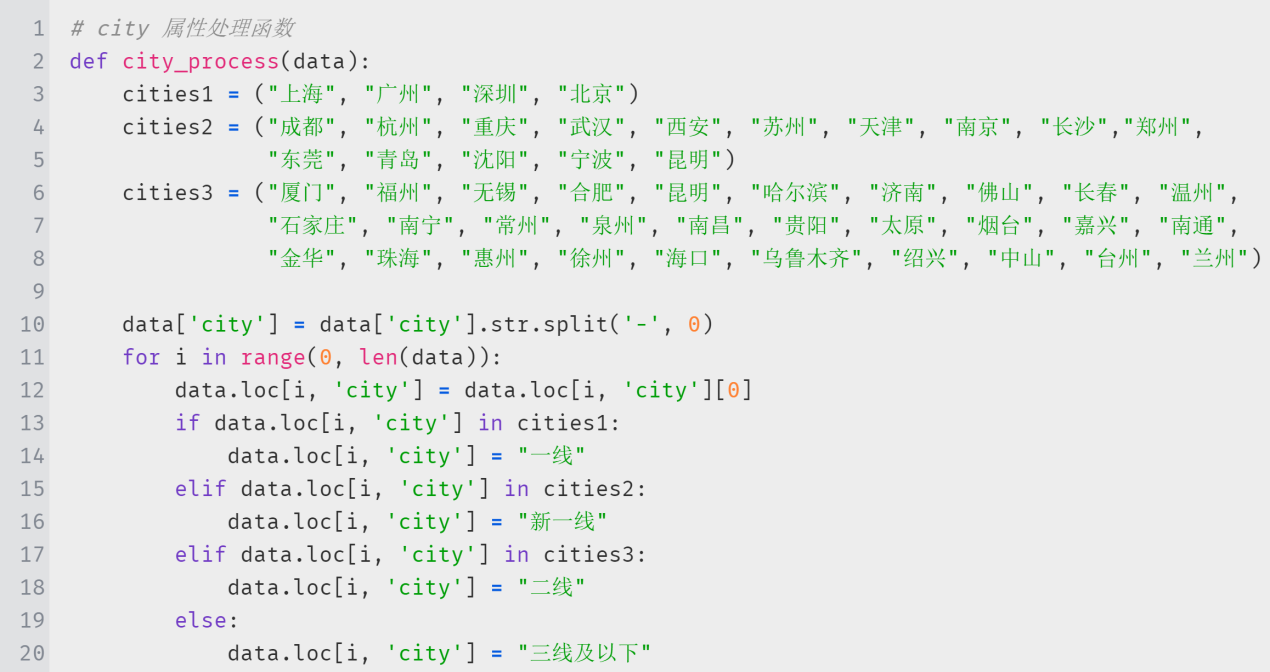


position属性: 职位属性直接影响薪资的高低，原始的职位名称五花八门，现在将其分类为：java资深, java高级, java中级, java开发, java实习。



company\_name属性: 公司名称显然与薪资高低相关性很小，直接删掉。

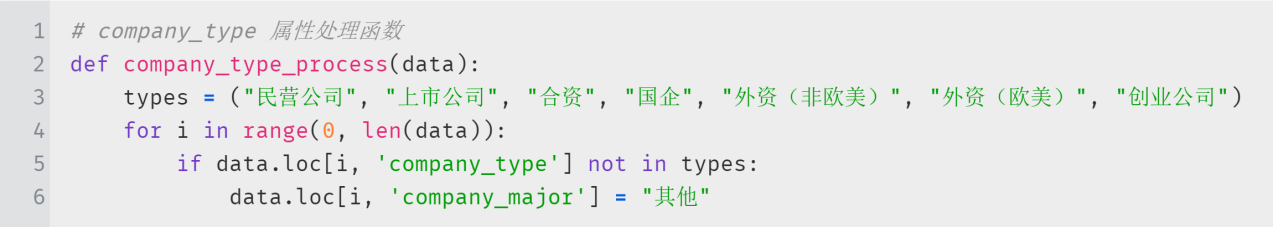
city属性: 城市属性对薪资高低也有很大的影响，我们提取出城市名，将城市名分类为：一线、新一线、二线、三线及以下。



education属性: 教育信息属性比较规整，但是出现了个别不是教育信息的内容，将其填充为众数：本科。

welfare属性: 福利属性应该算是标签salary的一小部分，为了简化模型，我们这里只考虑薪资，忽略掉福利，将该属性直接删掉。

company\_type属性：由于我们之后要对所有的离散型变量独热编码，该属性下类型太多，我们将频数较小的一些类型统一为“其他”。



company\_major属性: 该属性下类型也非常多，考虑到职位类型为Java开发，故我们将其分类为：计算机硬件、计算机软件、其他行业。

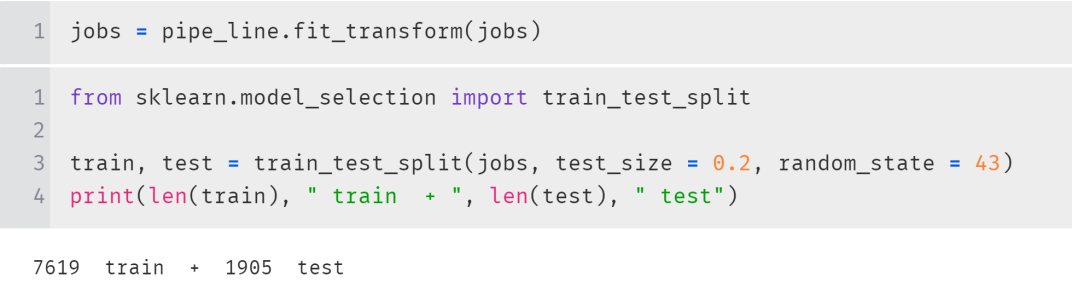


(2) 为了使测试集能够有和训练集完全相同的数据处理，定义一个来处理数据的管道 pipe\_line。在管道函数中对离线变量使用了独热编码。



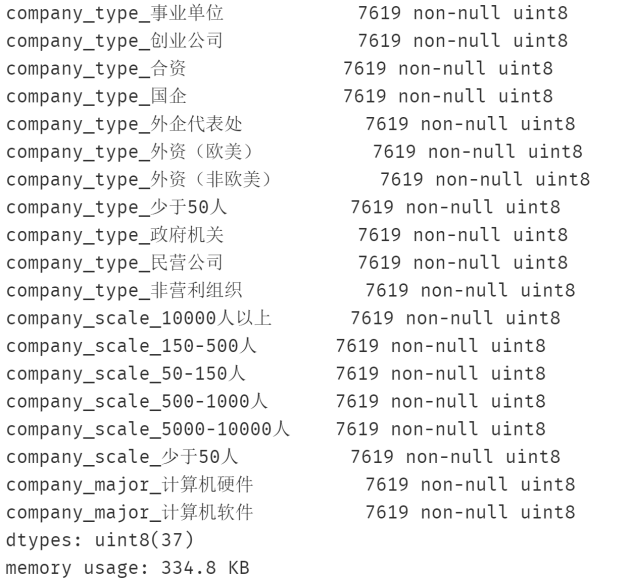
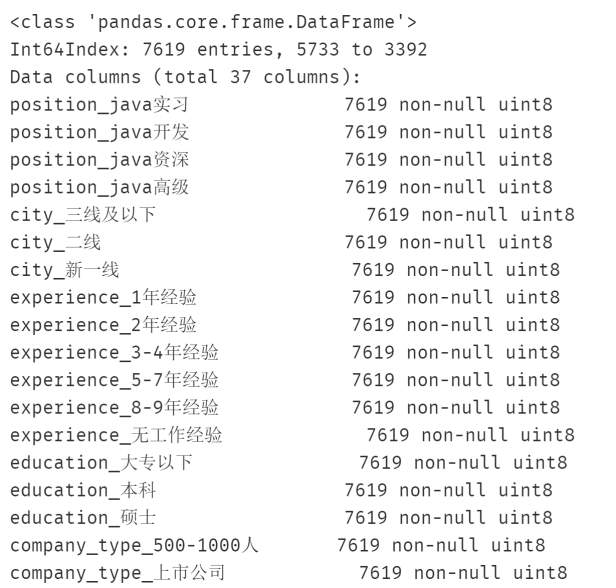


(3) 分离出测试集及标签并进行处理。



处理后的数据如图所示：



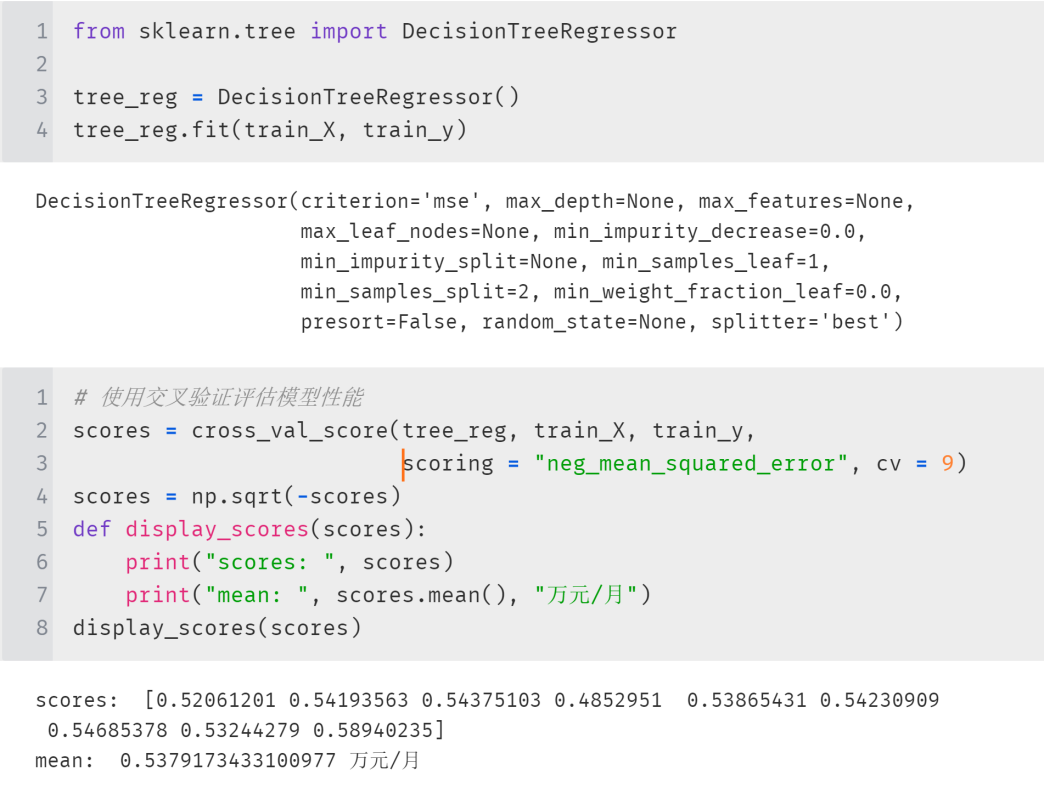


5.4.4 训练模型

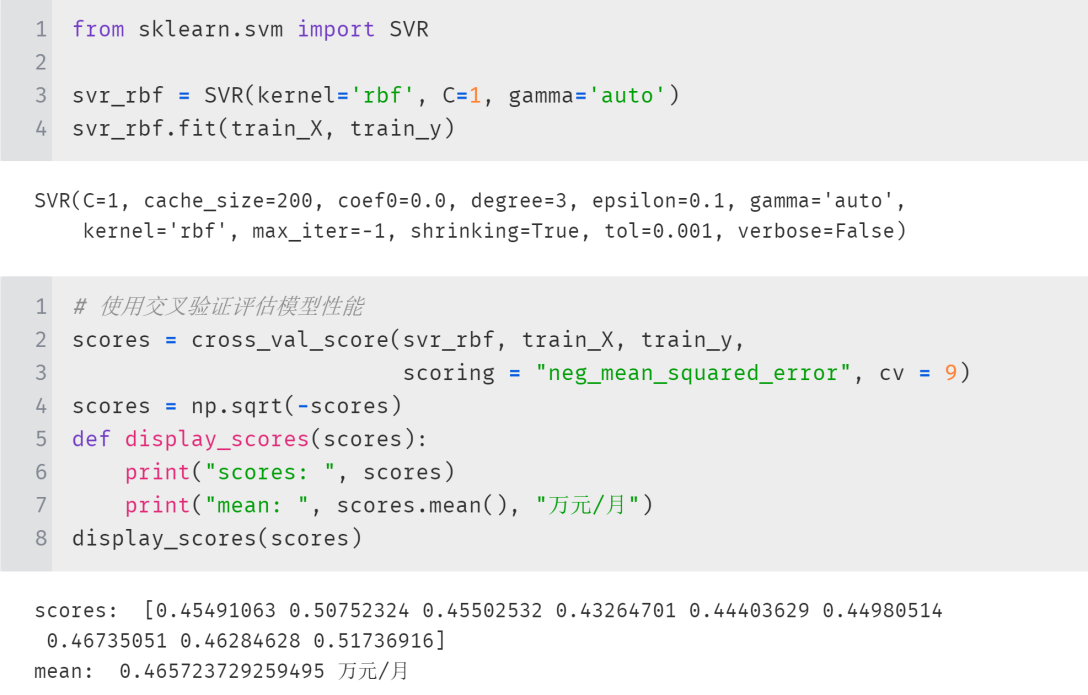
(1) 线性回归：



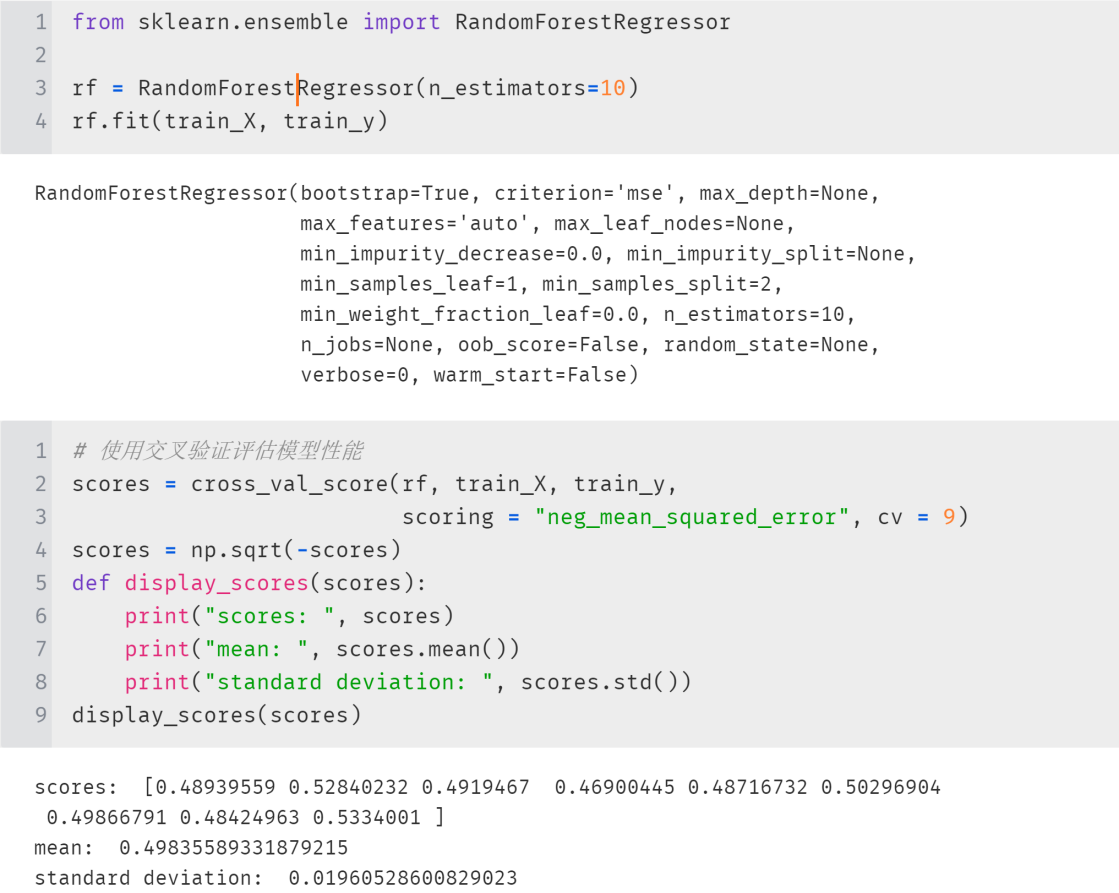
(2) 决策树



(3) 支持向量机



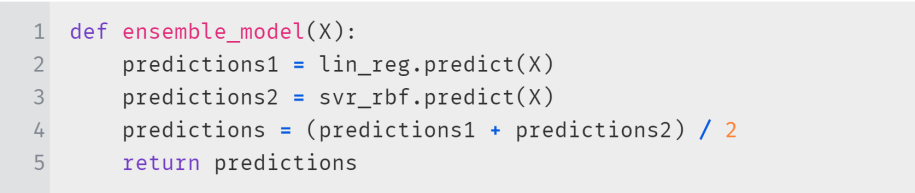
(4) 随机森林



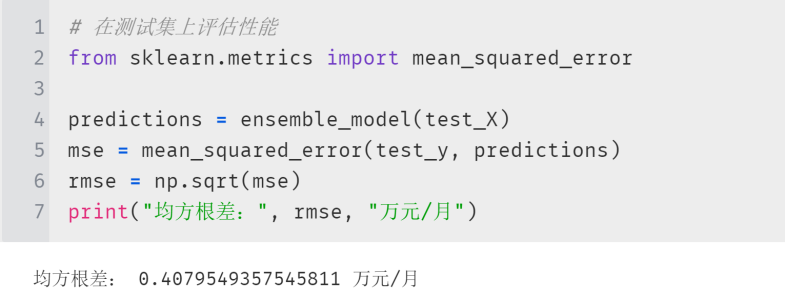
通过对以上几个模型的比较分析，我们发现线性回归和支持向量机两个模型表现较好，而决策树出现了过拟合现象。

5.4.5. 模型融合及评估性能

由于本项目是回归问题，故采用了比较简单的平均法对预测结果进行融合，选取的基预测器是表现较好的线性回归与支持向量机。



我们将融合后的模型在测试集上测试其性能，结果如下，融合后的模型误差更小，模型融合起到了一定效果。

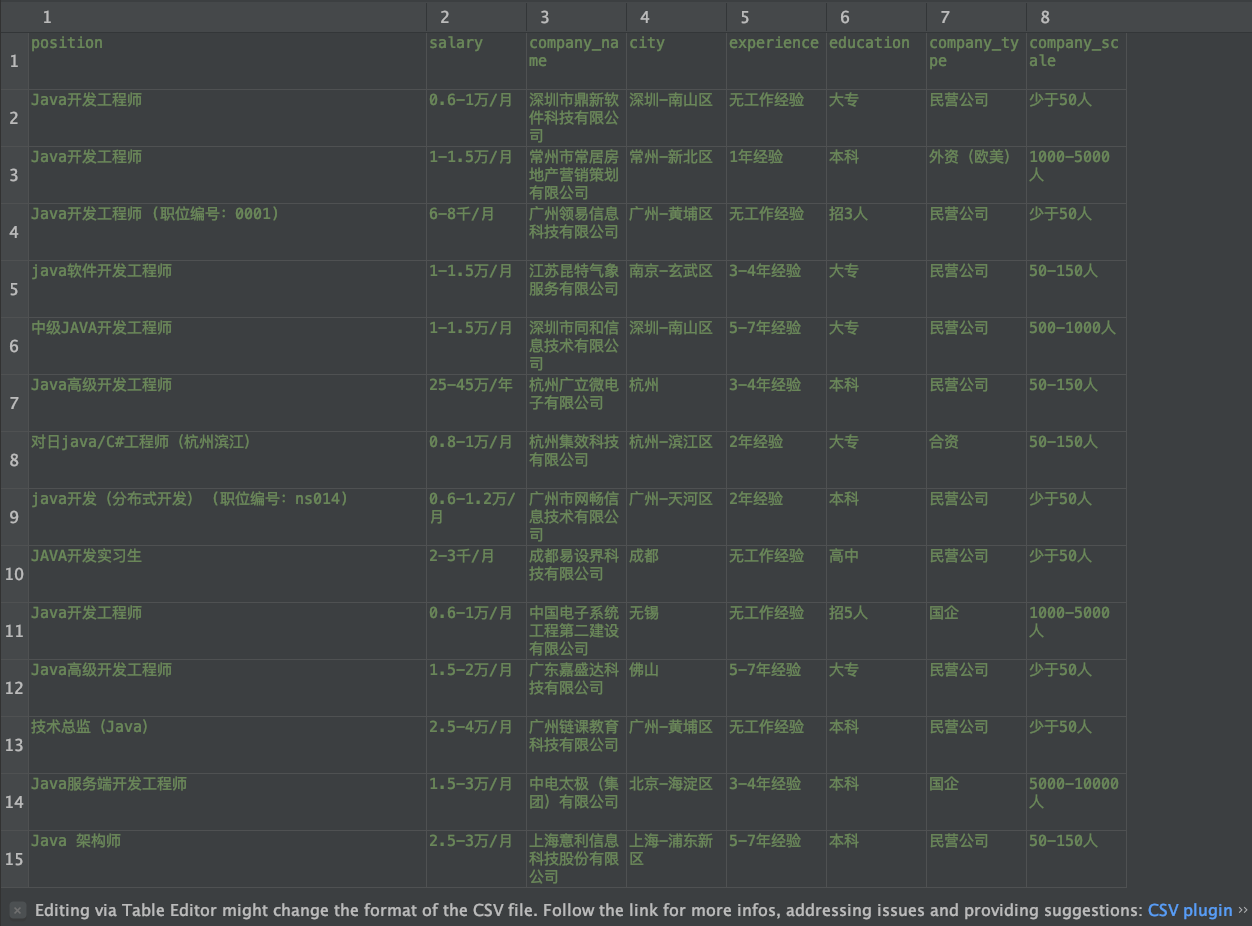


**六、结果展示**

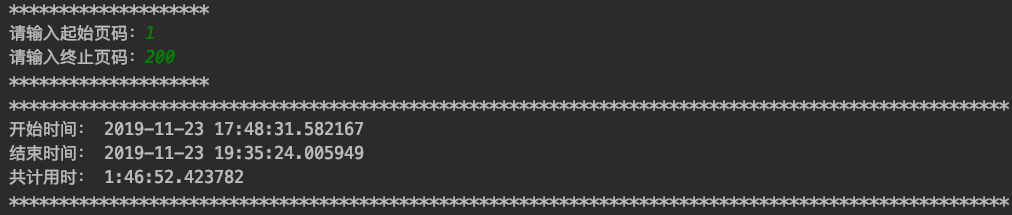
6.1 爬虫模块结果展示

6.1.1 单线程爬虫结果

CSV文件：

****

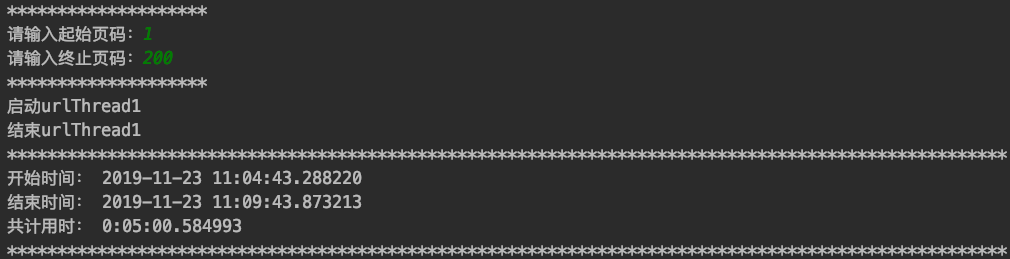
爬取200页将近一万数据花费时间：

****

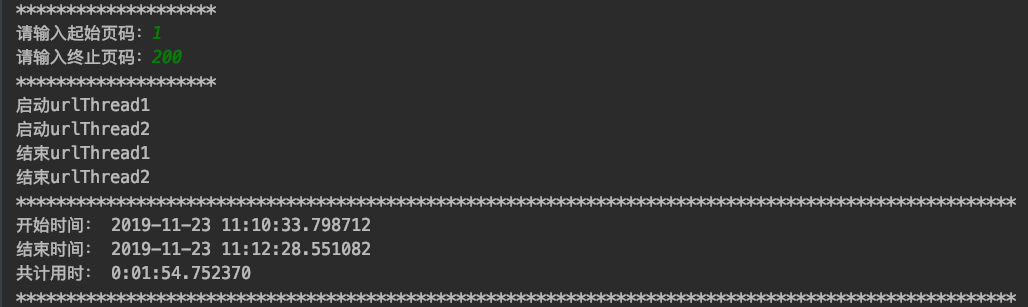
**分析**：只使用单线程爬虫速度不理想，下面尝试使用多进程

6.1.2 多线程爬取职位URL结果

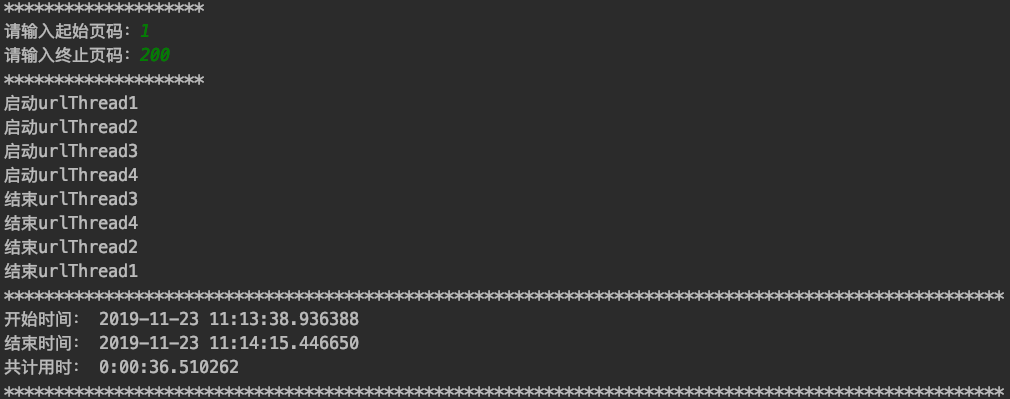
1. 分配1个线程

****

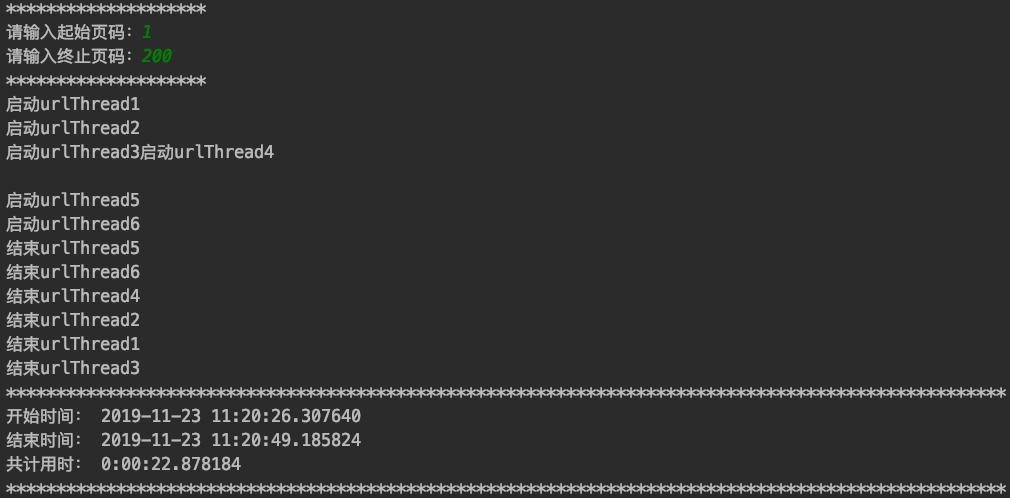
2.分配2个线程

****

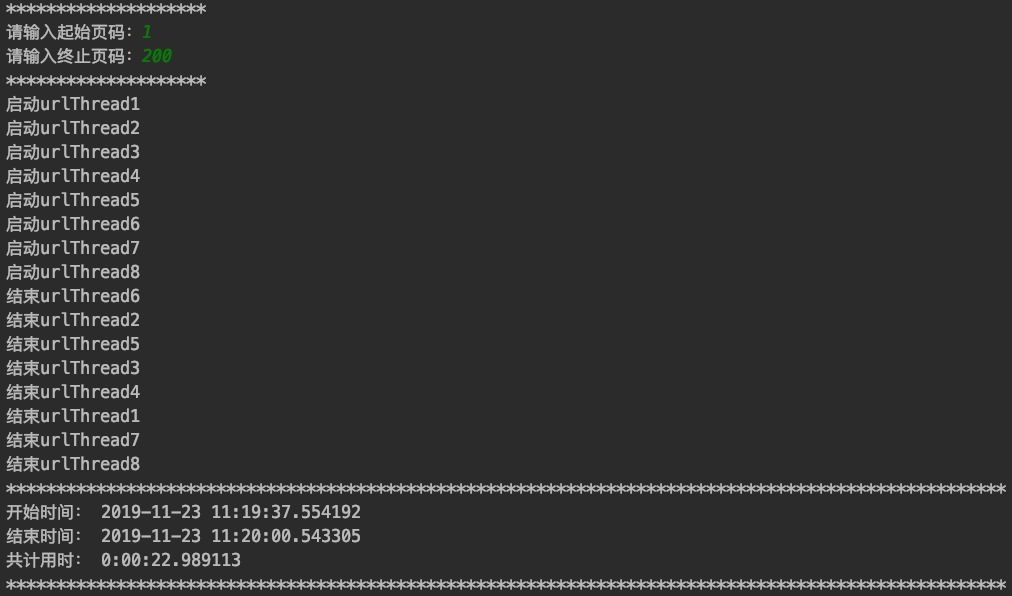
3.分配4个线程

****

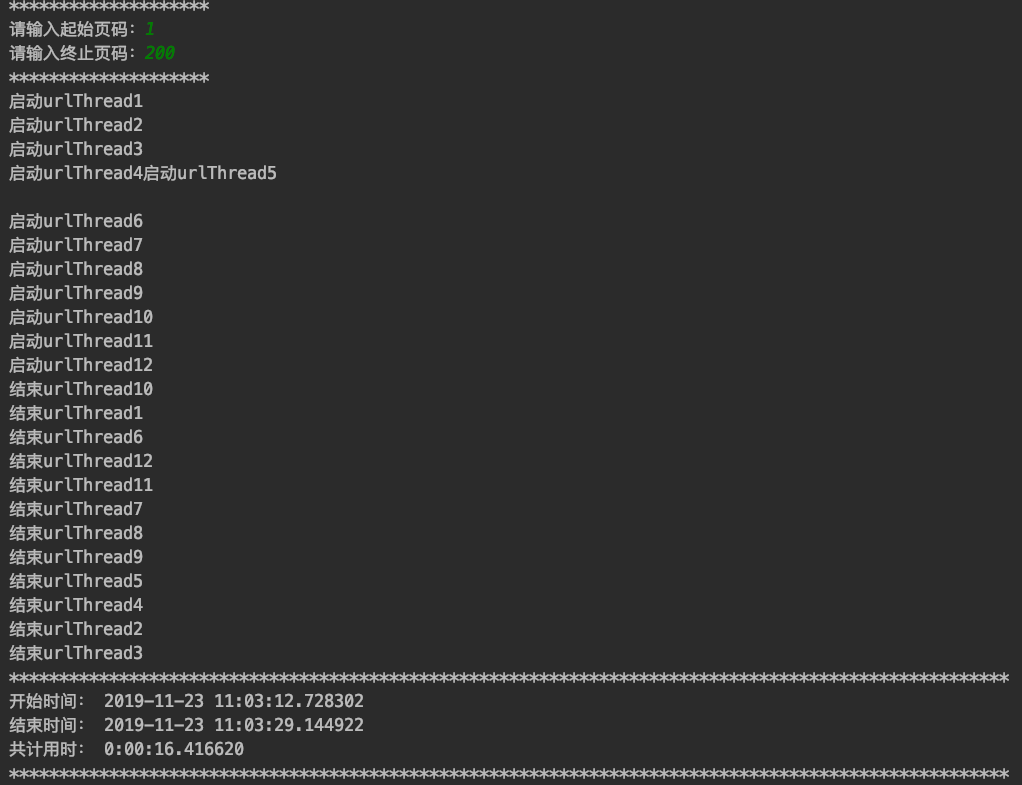
4.分配6个线程

****

5.分配8个线程

****

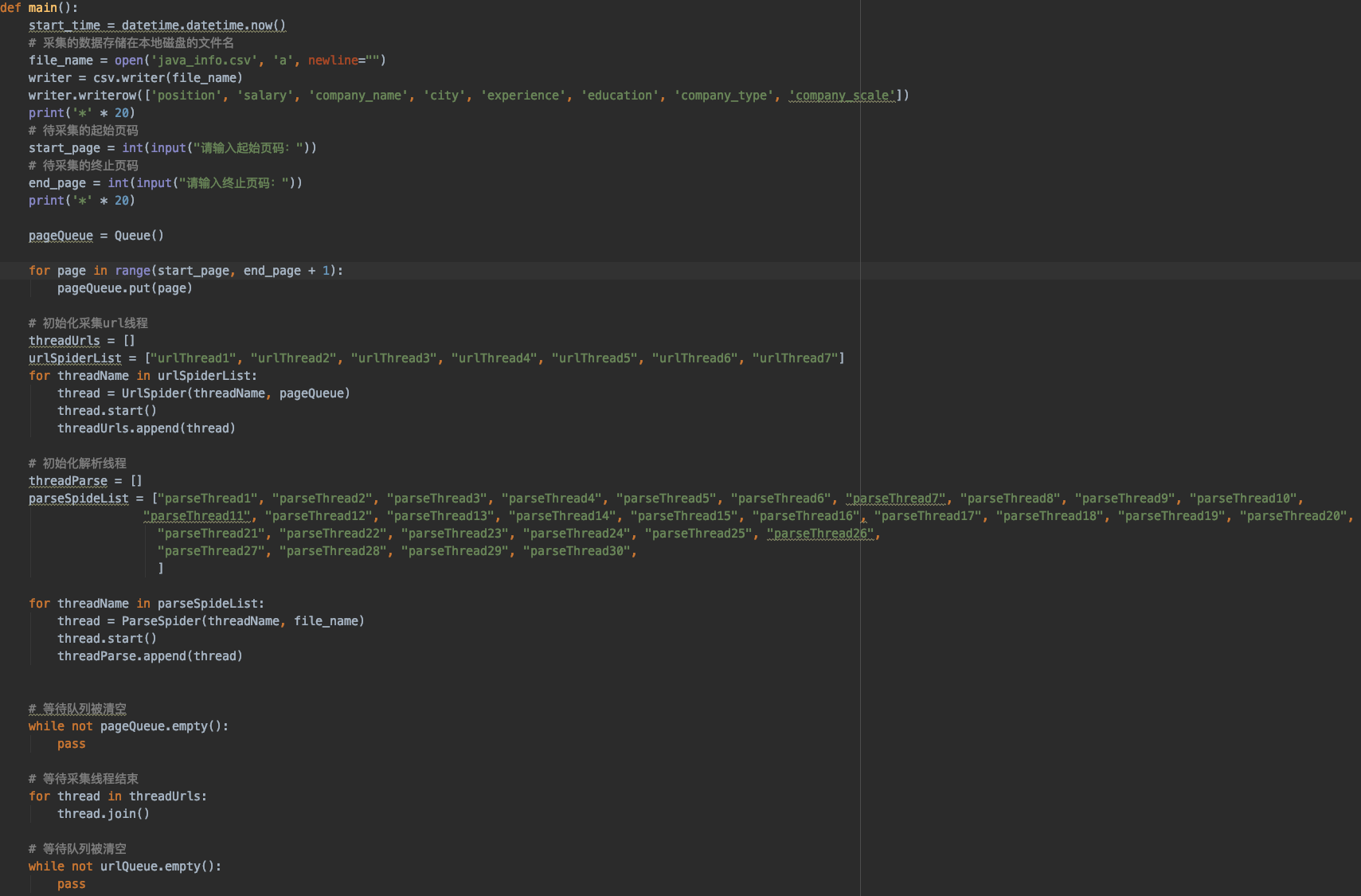
6.分配12个线程

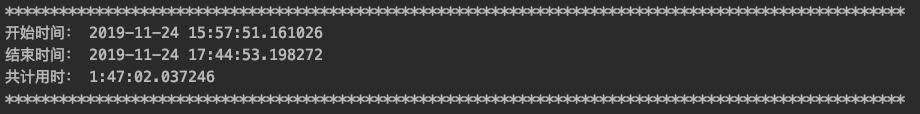
****

**分析：**从运行结果，可以了解到多线程确实可以极大加快爬取速度，线程数在4到8个为宜，再多增幅不明显，且占用资源。

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 分配线程数 | 1 | 2 | 4 | 6 | 8 | 12 |
| 花费时间(s) | 300 | 114 | 36 | 22 | 22 | 18 |

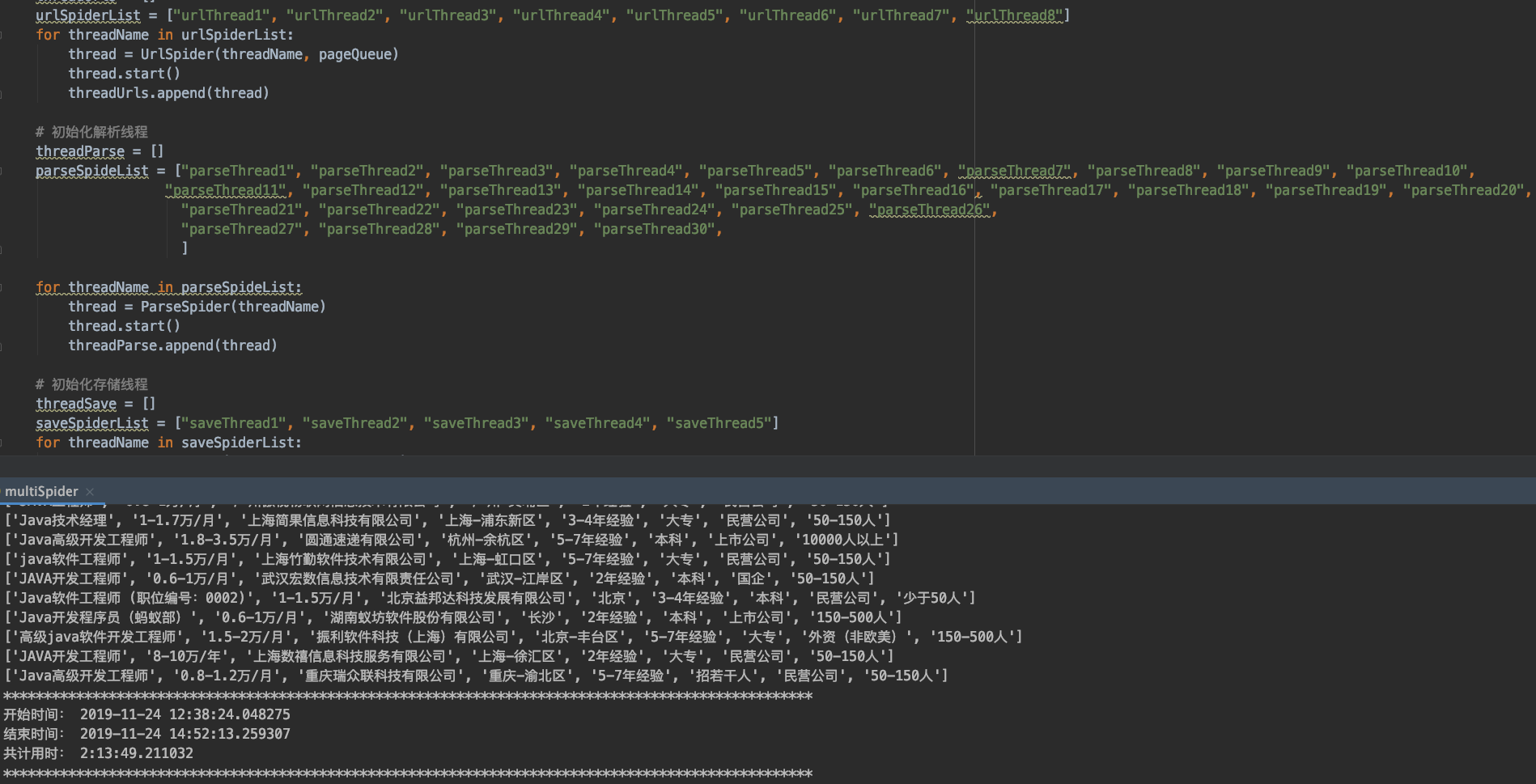
6.1.3 多线程拆分为两个类的爬取结果

****

****

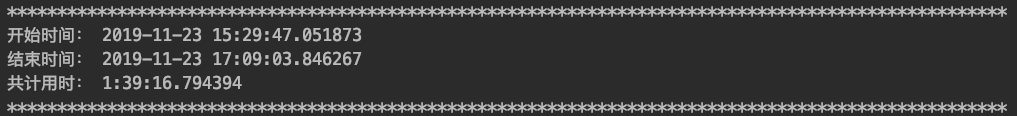
**分析：**分配了6个线程给提取URL类，30个线程给页面解析类。速度并没有任何提升，猜想是不是解析类中写入csv文件那块占用时间比较多影响了速度。

6.1.4 多线程拆分为三个类的爬取结果

****

**分析：**用了两个小时出头，反而时间更慢了

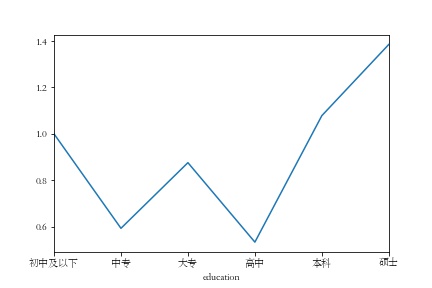
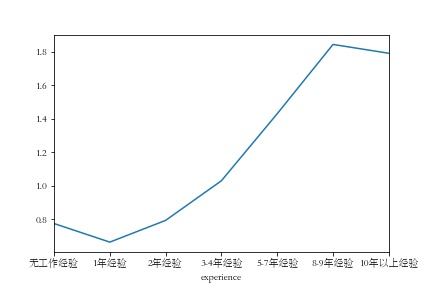
6.1.5 多线程拆分为四个类的爬取结果

****

**分析：**就比单线程加速了几分钟，百思不得其解，能拆分的功能都给拆分一个个类再给它们分配线程，怎么提速这么少。实在解决不了。

6.2 数据可视化

6.2.1 matplotlib.pyplot 线状图



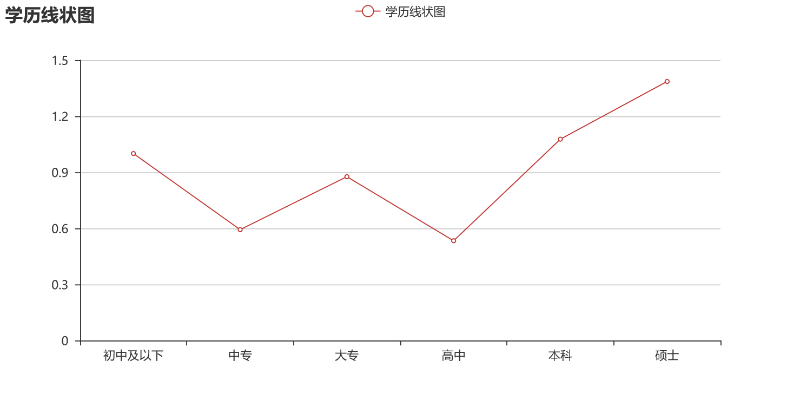
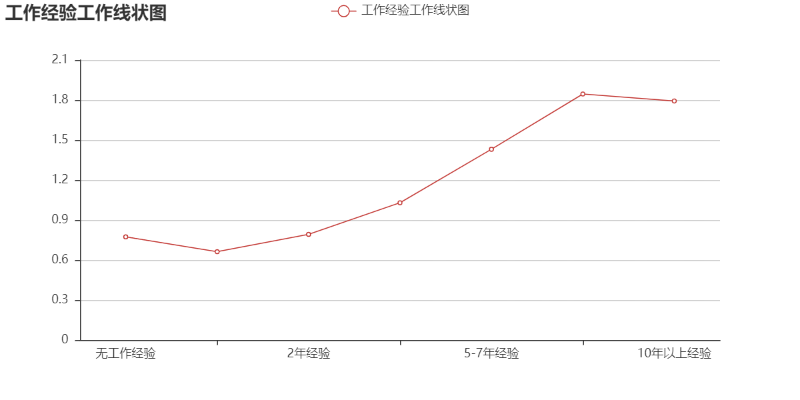
6.2.2 pyecharts.Pie环图



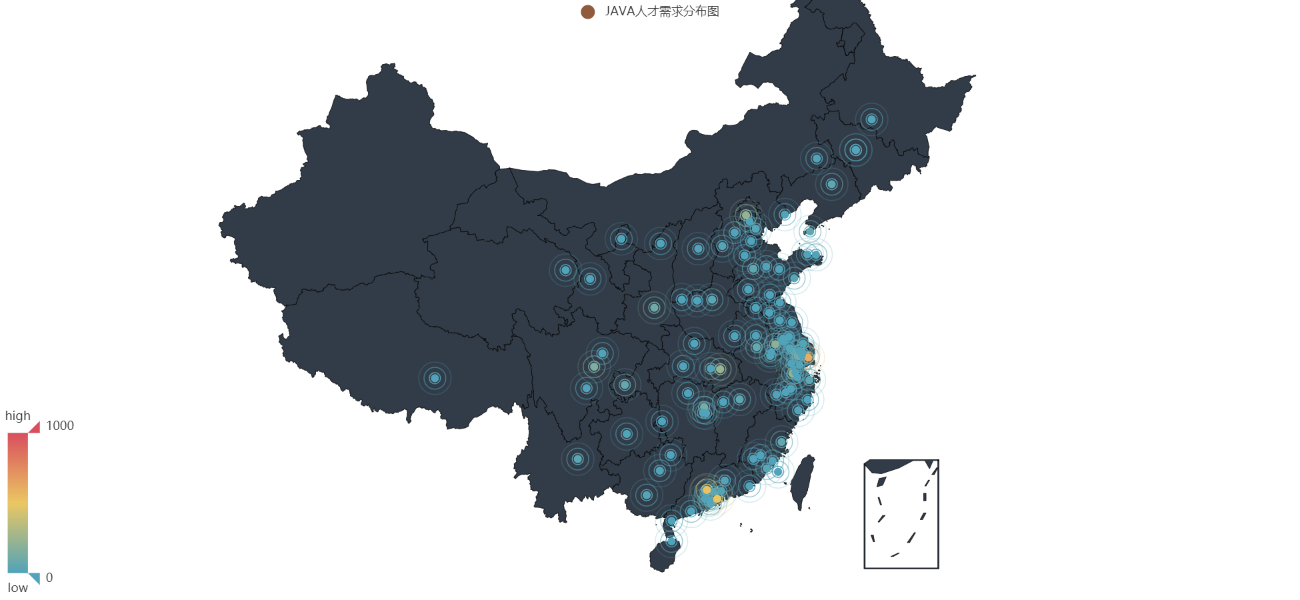
6.2.3 pyecharts.WordCloud词云



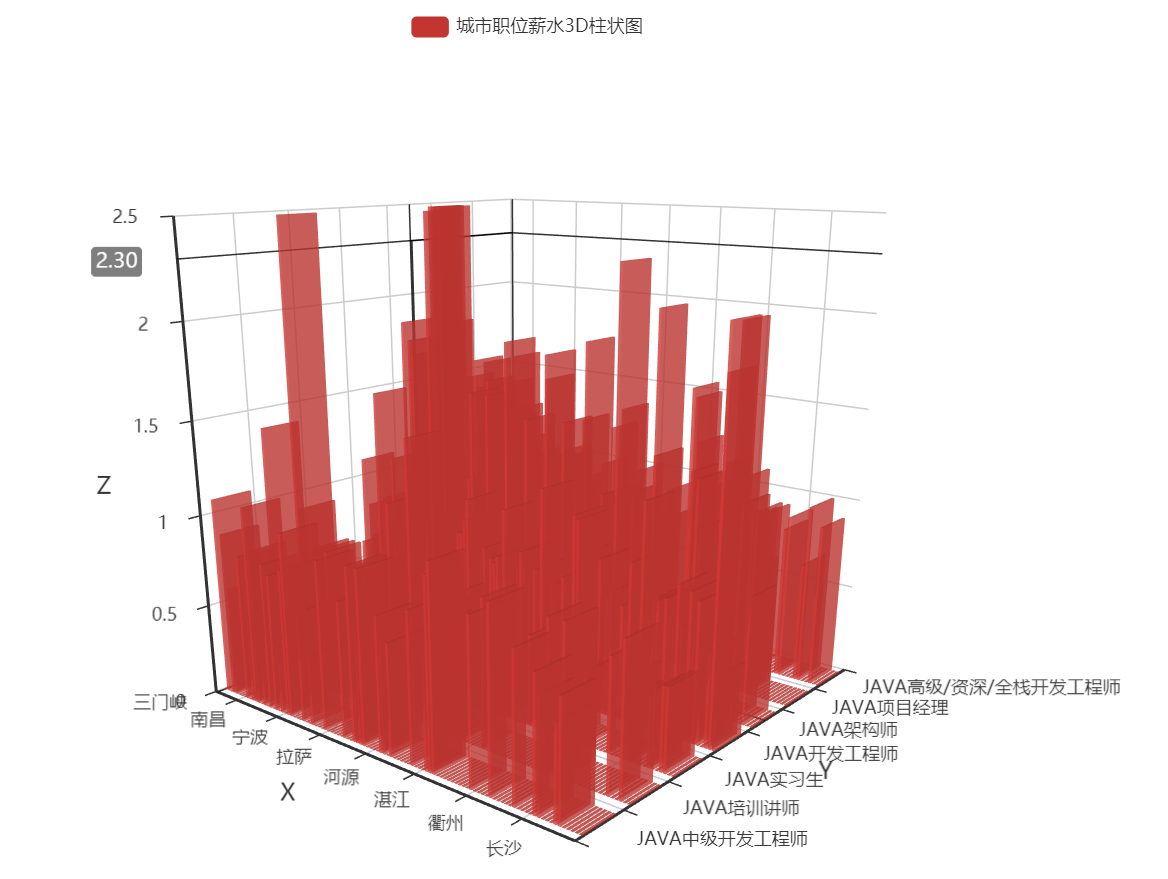
6.2.4 pyecharts.Line折线图



6.2.5 pyecharts.Geo热点地图



6.2.6 pyecharts. 3D柱状图



根据以上可视化可以分析出以下结论：

无工作经验的薪资会比1年经验的要高，这是因为刚出校门的应届生能够参加学校组织的校招，待遇普遍不错，而工作经历1年又刚刚失去应届身份的人只能参加社招，这些人一没工作经历竞争力二无法享受校招待遇。故刚毕业后一年是一段处于最困难的时期，除非有非常完美的计划，不建议跳槽或者离职。工作10年以上工作经历足够，但因为年纪大精力不足无法与年轻人去比较，再者一般这么长工作经历的员工在公司福利待遇都是最高的，性价比较低。

根据热点地图可以发现，高薪资都聚集在长江三角以及港珠澳湾区，这两个地区。南方的需求明显大于北方以及西南方向，当今社会企业对JAVA的需求还是很高，分布也很广泛。同时我们也能发现，java相关工作岗位数量及薪资，超一线城市 > 一线城市 > 二三线城市，至于小城市一般都是极个别的有需求。我们还是能从中发现，互联网企业一般还是聚集在大城市之中。

通过学历玫瑰环图，我们能发现大部分需求要求都是本科生以上，虽然都说JAVA工作好找，且门槛低，无论什么学历什么专业都能应聘。但是通过以上的图我们能看到未达到本科的话，工作还是很难找的，而且薪资也比平均薪资要低。在IT人才,甚至是程序员以及达到饱和的今天，职位需求开始精英化，很多开发工程师岗位都是本科以上，高级甚至是架构师都必须要求为硕士以上了。图中缺少博士研究生的岗位，不是因为不招博士，也不是因为博士工作难找，主要问题在于，博士相对于其他学历更加稀少，而且专业性很强，一般都是由内推或者引荐，故社招要求博士学位在市场上一般很少见。

6.3 机器学习应用部分的结果展示

我们在测试集上随机查看部分样本的表现如下图所示。



**七、存在的问题**

7.1 网络爬虫存在的问题

在将单线程爬虫写成多线程爬虫后时间提速不明显或根本没什么提速，但从6.1.2节单单只爬取1万条职位URL时，六线程所花费的时间是单线程的将近14分之1，提速非常明显。所有整合起来却没什么提升。

**可能的原因：**

1.只爬取1万条职位URL时，每个pageurl可以爬50条职位URL，也就是说只需要跳转200个页面；而整体爬取时，每条职位URL只能爬一条职位信息，也就是说要跳转10000个页面。跳转页面的时间是大头，所以造成时间提升不明显。

2．我的笔记本配置不够好；

3．算法内部逻辑结构可能出现我没找出的问题，出现多个线程互相等待的情况

8.2 数据可视化存在的问题

对数据的展示还是不够充分，差异性没有得到很好的展示，且数据与数据之间的关联性也并没有很充分的展示出来。

对数据的分析只是停留在最表面的层次上，并没有深入的去分析其原因，甚至没有提出相应的见解。

7.3 机器学习应用部分存在的问题

分离出测试集的时候更好的步骤是先分离出测试集，再对训练集调用fit\_transform()函数进行数据处理，对测试集调用transform()进行处理。

但是调用sklearn的train\_test\_split()函数后，所有的样本都被打乱顺序，导致使用fit\_transform()和transform()处理数据时出现如下报错，目前还未找到有效解决方案。

