《大数据开发技术（Hadoop）》

实训报告

主题：招聘网站职位分析实战

组长：曾森林

成员：王亚徽 刘欣粤

完成日期：2024.12.31

目 录

[第1部分 项目背景 1](#_Toc1057)

[第2部分 项目实现技术 2-3](#_Toc27978)

[第3部分 项目实现思路介绍 3](#_Toc20657)

[第4部分 项目实现流程介绍 3-11](#_Toc22979)

[第5部分 项目分析结果说明 11-12](#_Toc13545)

[第6部分 项目代码介绍文档 12-39](#_Toc1485)

# 项目背景

招聘网站职位分析实战项目的背景可以从以下几个方面进行阐述：

一、招聘行业现状与发展趋势

随着互联网技术的快速发展，招聘行业规模不断扩大，线上招聘平台逐渐成为主流。招聘网站作为线上招聘平台的重要组成部分，为求职者和招聘方提供了一个便捷、高效的交流平台。同时，招聘行业也面临着一些挑战，如信息真实性难以保障、招聘流程繁琐、人才匹配精准度不高等问题。因此，打造一个高效、便捷、真实的招聘平台，提高招聘效率和人才匹配精准度，成为当前招聘网站发展的重要方向。

二、市场需求与竞争态势

市场需求：随着人才流动和就业市场竞争的加剧，企业对招聘网站的需求持续增长。招聘网站能够提供丰富的职位信息，帮助企业快速找到合适的人才。同时，求职者也可以通过招聘网站了解最新的招聘信息，提高求职效率。

竞争态势：目前市场上主要的招聘网站包括前程无忧、智联招聘、拉勾网、BOSS直聘等。这些网站在品牌知名度、用户规模、技术实力等方面具有一定优势。然而，随着市场竞争的加剧，一些招聘网站也面临着产品同质化、服务单一等问题。因此，如何在激烈的市场竞争中脱颖而出，成为当前招聘网站需要思考的重要问题。

三、项目目标与意义

项目目标：招聘网站职位分析实战项目的目标是通过采集招聘网站大数据职位信息，利用数据分析、数据挖掘等技术手段，对职位信息进行深入分析，揭示最新的岗位需求趋势、薪资状况以及求职者的技能需求。同时，通过构建精准的职位和人才推荐算法，提升用户体验和匹配效率。

项目意义：该项目不仅有助于招聘网站提升服务质量和用户体验，还能够为企业提供更精准的人才匹配服务，降低招聘成本。此外，该项目还能够为求职者提供更全面的职位信息和求职建议，提高求职成功率。

四、技术实现与数据分析

技术实现：招聘网站职位分析实战项目通常采用先进的网络爬虫技术（如Selenium）和数据分析工具（如Python、PySpark、jieba分词等）进行数据采集和分析。通过构建自动化数据抓取系统，实现招聘信息的高效采集和清洗。同时，利用数据挖掘和机器学习技术构建精准的职位和人才推荐算法。

数据分析：在数据分析阶段，项目团队会对采集到的招聘信息进行深入分析，包括职位类型、薪资水平、工作经验要求、学历要求等方面的统计和分析。通过制作各种图表（如饼状图、直方图等）直观展示数据分析结果，帮助用户更好地理解数据背后的意义。

五、实际应用与效果评估

实际应用：招聘网站职位分析实战项目的成果可以应用于多个领域。例如，招聘网站可以利用这些成果优化职位推荐算法，提高用户匹配度；企业可以利用这些成果了解最新的岗位需求趋势和薪资状况，制定更合理的招聘计划；求职者可以利用这些成果了解不同行业和职位的薪资水平和职业发展前景，做出更明智的求职决策。

效果评估：为了评估项目的实际效果，可以通过对比实验、用户调查等方式进行效果评估。例如，可以对比使用项目成果前后的用户匹配度、招聘效率等指标的变化情况；同时，也可以通过用户调查了解用户对项目的满意度和改进建议。

综上所述，招聘网站职位分析实战项目具有重要的现实意义和应用价值。通过深入分析招聘信息和构建精准的职位和人才推荐算法，该项目能够为招聘网站、企业和求职者提供全方位的服务和支持。

# 项目实现技术

1. **编程语言**：Python是常用的编程语言，因为它具有丰富的数据处理和分析库（如Pandas、NumPy等）。
2. **爬虫工具**：BeautifulSoup、Scrapy等用于网页解析和数据抓取。
3. **数据分析工具**：Pandas、NumPy、jiaba，pychark等用于数据处理和分析。
4. **数据可视化工具**：Matplotlib、Django Echarts等用于数据可视化。
5. **机器学习库**：LDA模型用于建立预测模型和进行数据挖掘。

# 项目实现思路介绍

整体实现思路：本项目通过采集招聘网站大数据职位信息，利用数据清洗，数据分析，结巴分词，数据挖掘，完成整体项目的开发工作。任务包含爬取招聘网站大数据职位信息，使用BeautifulSoup清洗职位信息网页，使用PySpark对智联数据进行分析，对招聘职位信息进行探索分析，使用结巴分词对岗位描述进行分词并将关键词统计，利用Echarts将职位分析结果可视化，建立职位模型对应应聘人员进行相似度的计算。

实验步骤：

#### 爬取招聘网站大数据职位信息。

#### 使用BeautifulSoup清洗职位信息网页。

#### 使用PySpark对职位数据进行分析。

#### 对招聘职位信息进行探索分析。

#### 使用结巴分词对岗位描述进行分词并将关键词统计。

#### 利用Django Echarts将职位分析结果进行可视化。

#### 建立LDA模型对职位描述进行相似度的计算

# 项目实现流程介绍

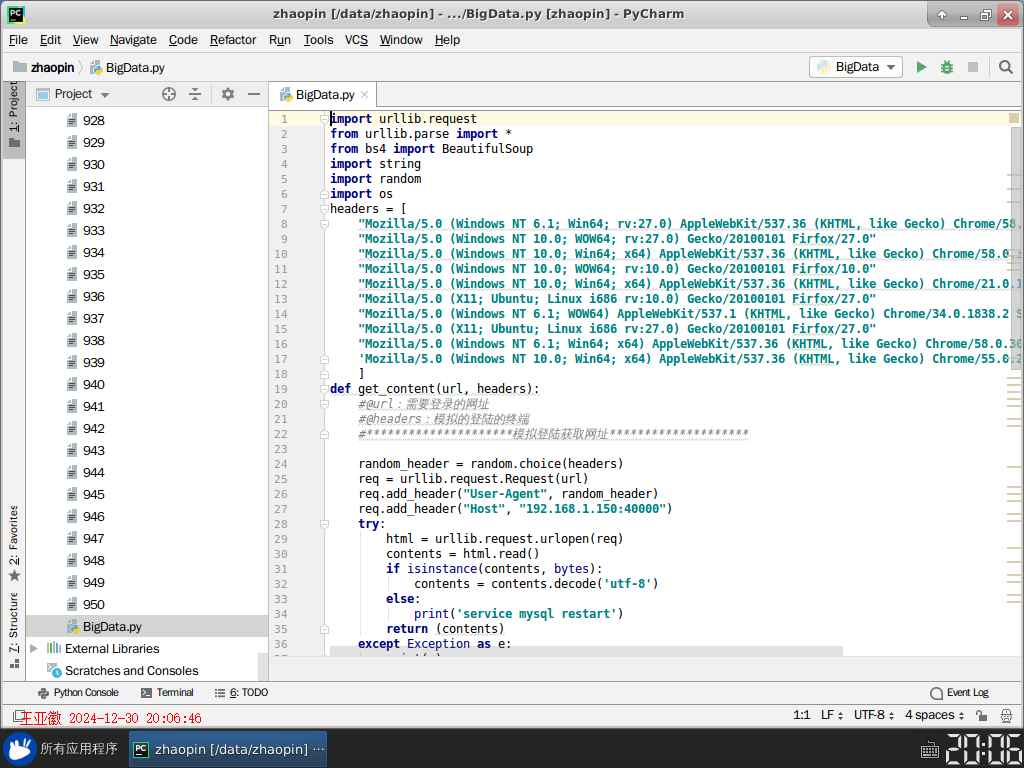
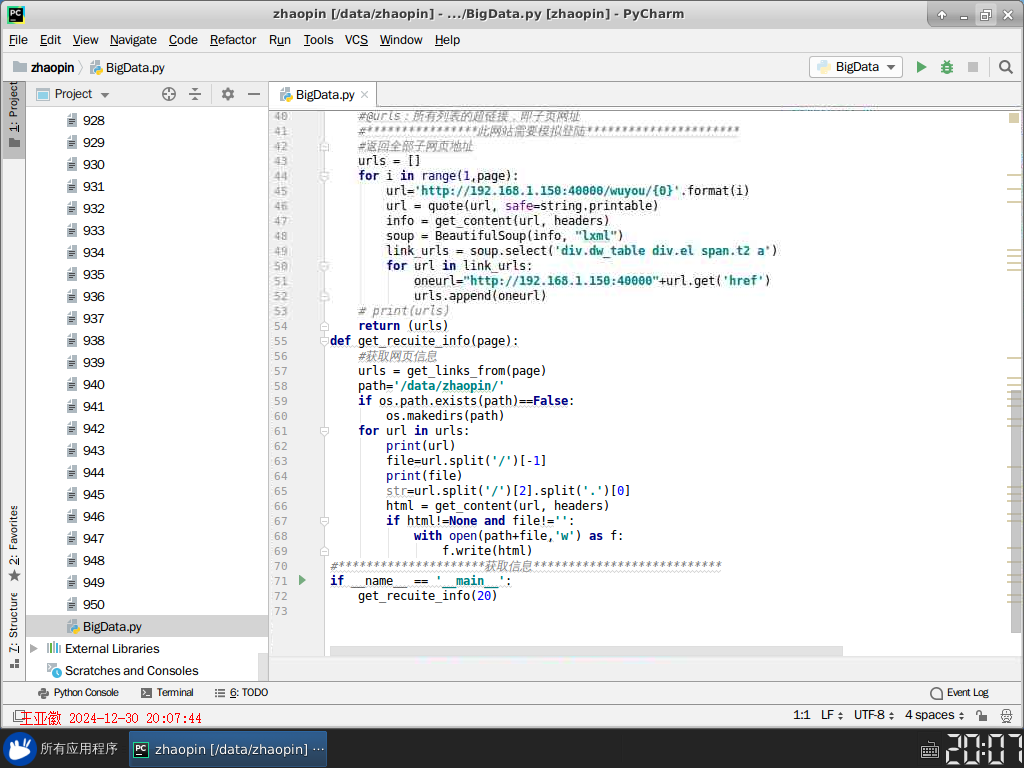
数据收集部分：

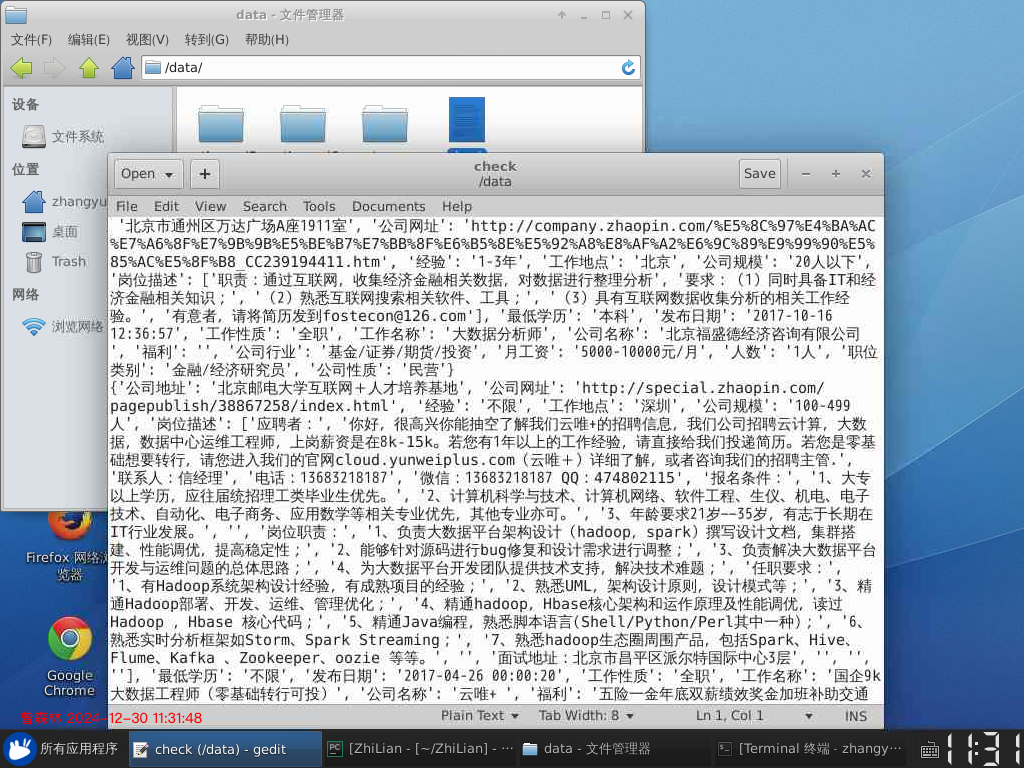
#### 第一步：爬取招聘网站大数据职位信息。

#### 将爬虫伪装成浏览器，防止网站针对爬虫的限制。

#### 模拟登陆获取网址

3.获取全部子网页地址

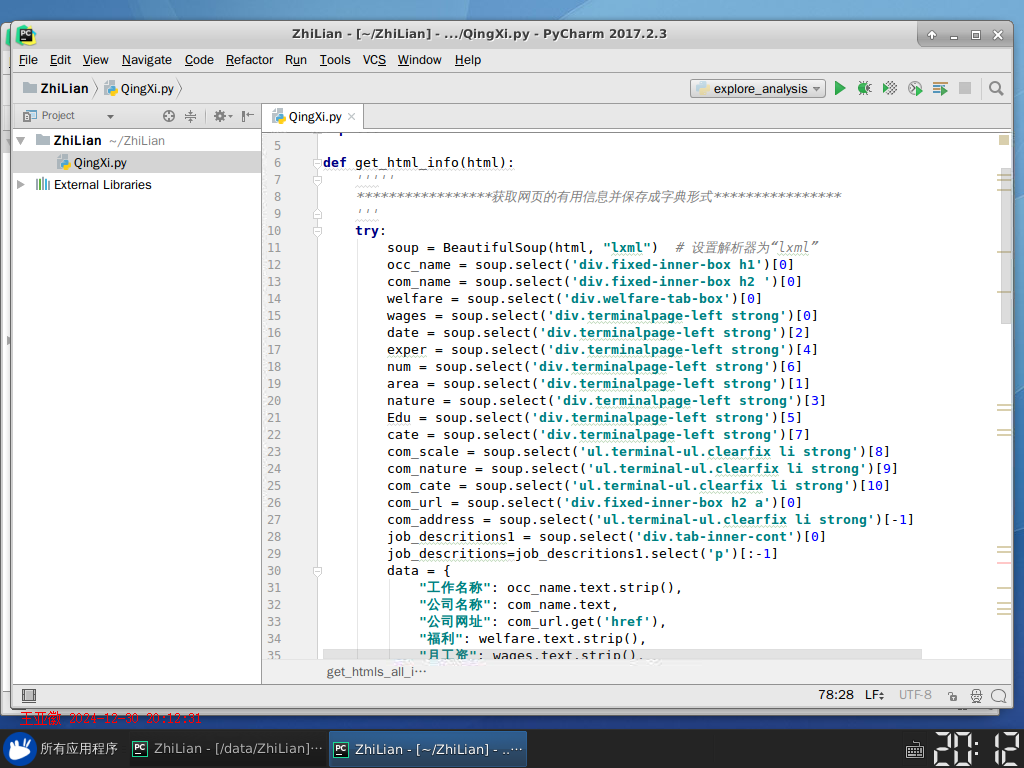


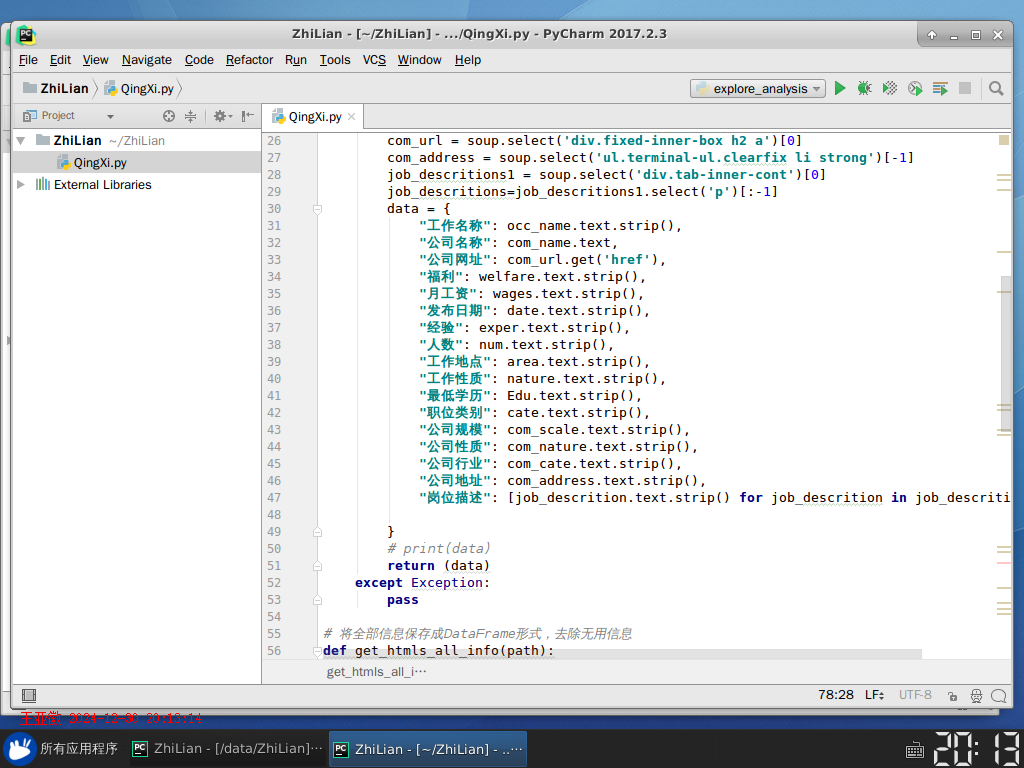
4. 获取网页信息，并保存

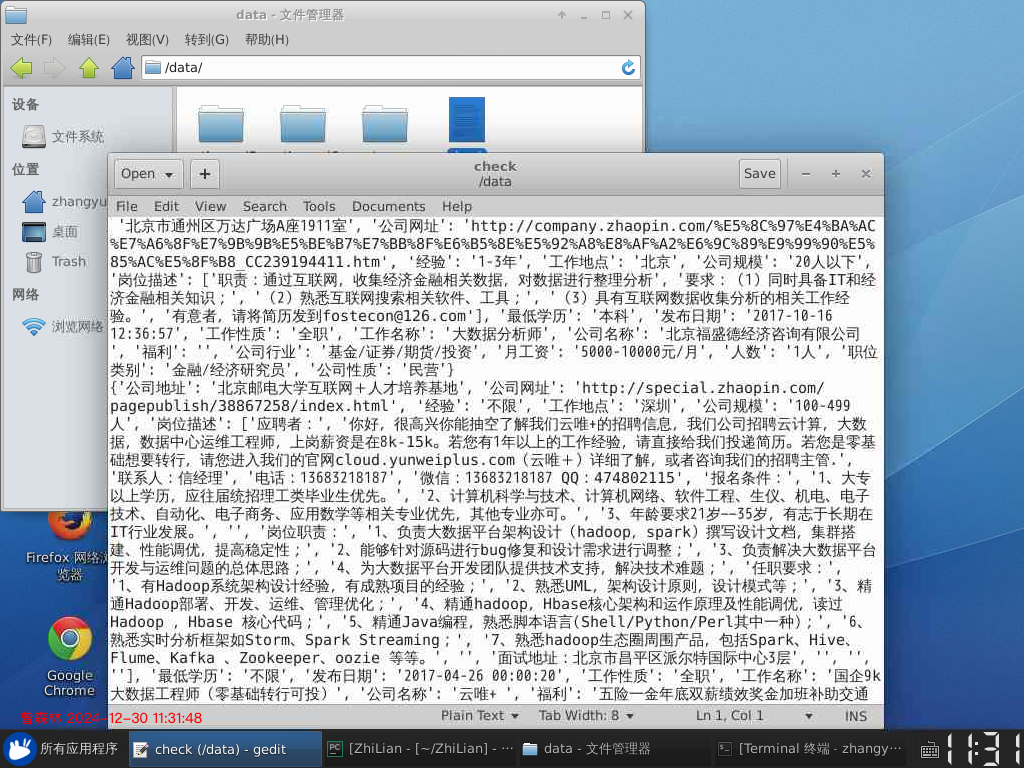
数据清洗部分：

#### 第二步：使用BeautifulSoup清洗职位信息网页。

1. 获取网页的有用信息并保存成字典形式
2. 将全部信息保存成矩阵形式，去除无用信息







数据分析部分：

#### 第三步：使用PySpark对职位数据进行分析。

##### 先启动mysql服务

##### 再开启PySpark

##### 创建RDD

##### 创建DateFrame并将从网站中筛选的信息的字段名与数据类型存入，并将信息存入

##### 利用DataFrame统计公司性质及数量

##### 利用Spark SQL统计经验要求及数量

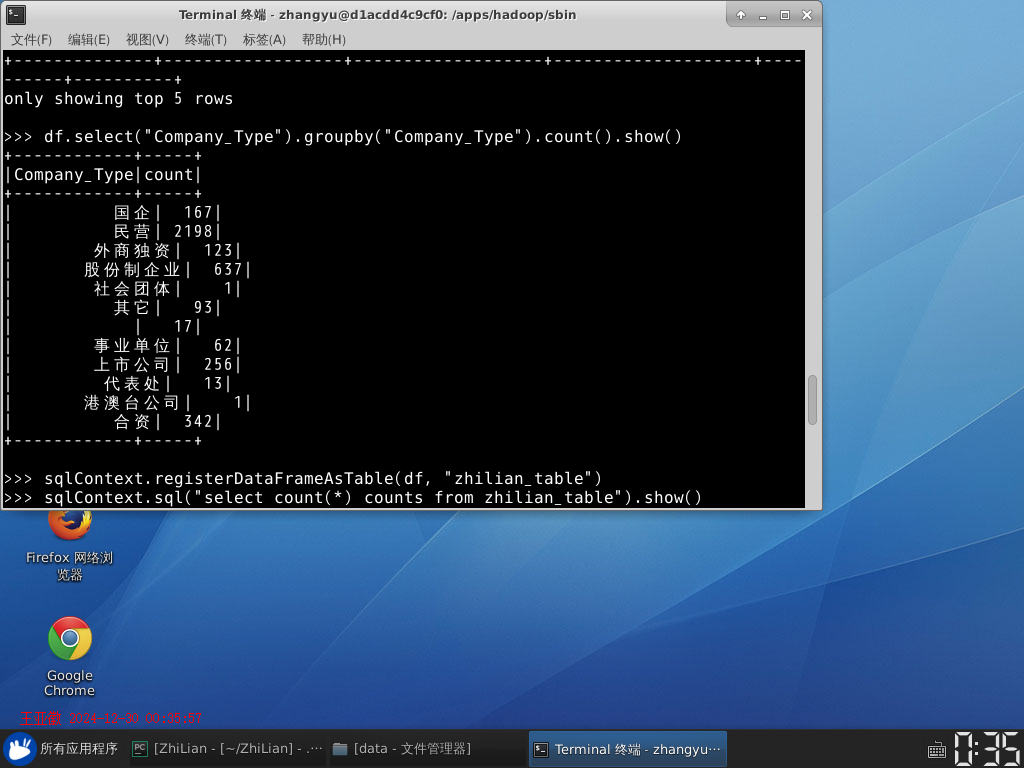
利用Spark SQL统计公司规模及数量

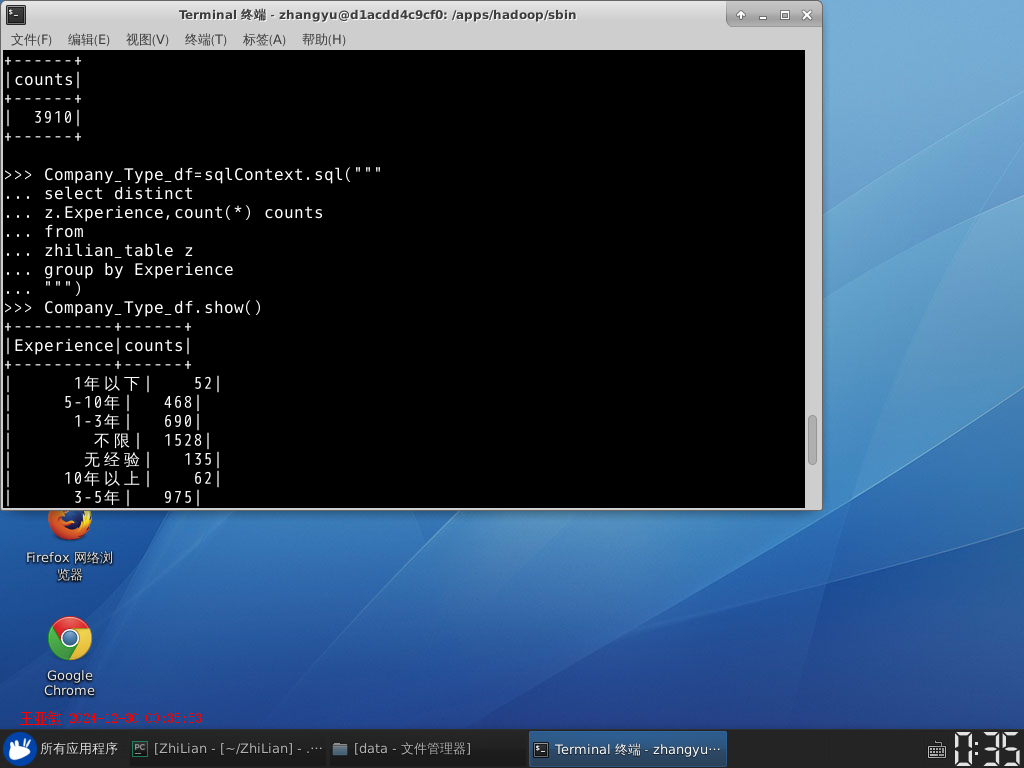
创新点：利用Spark SQL统计最低学历及数量

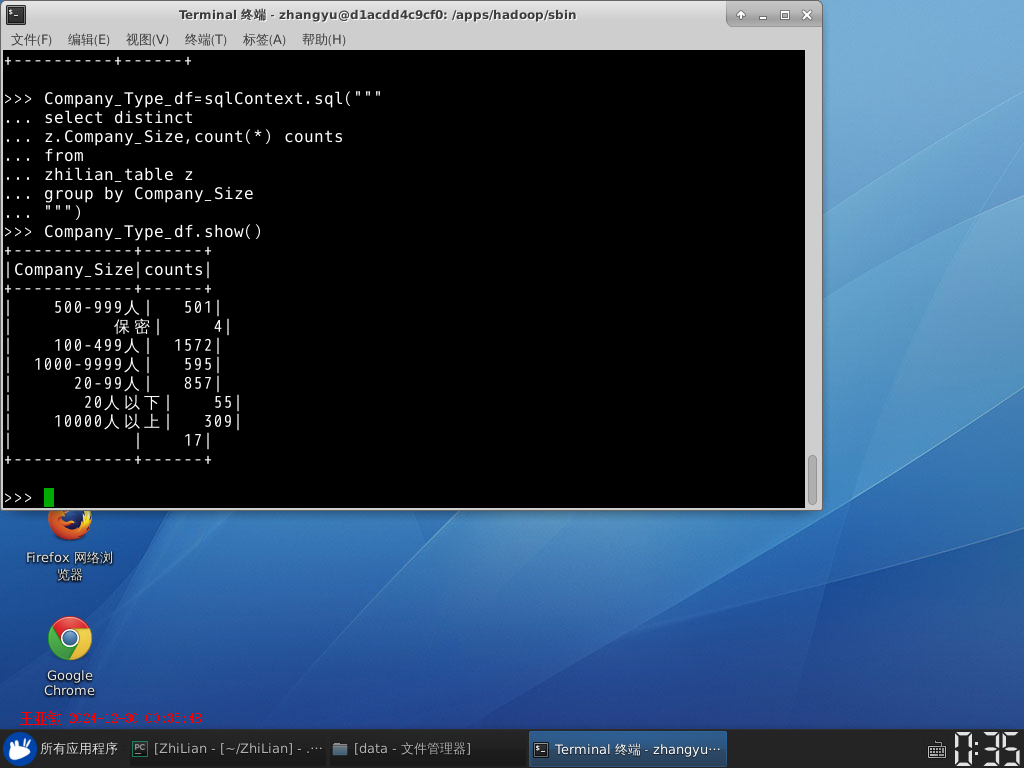
利用Spark SQL 统计工作地点及数量

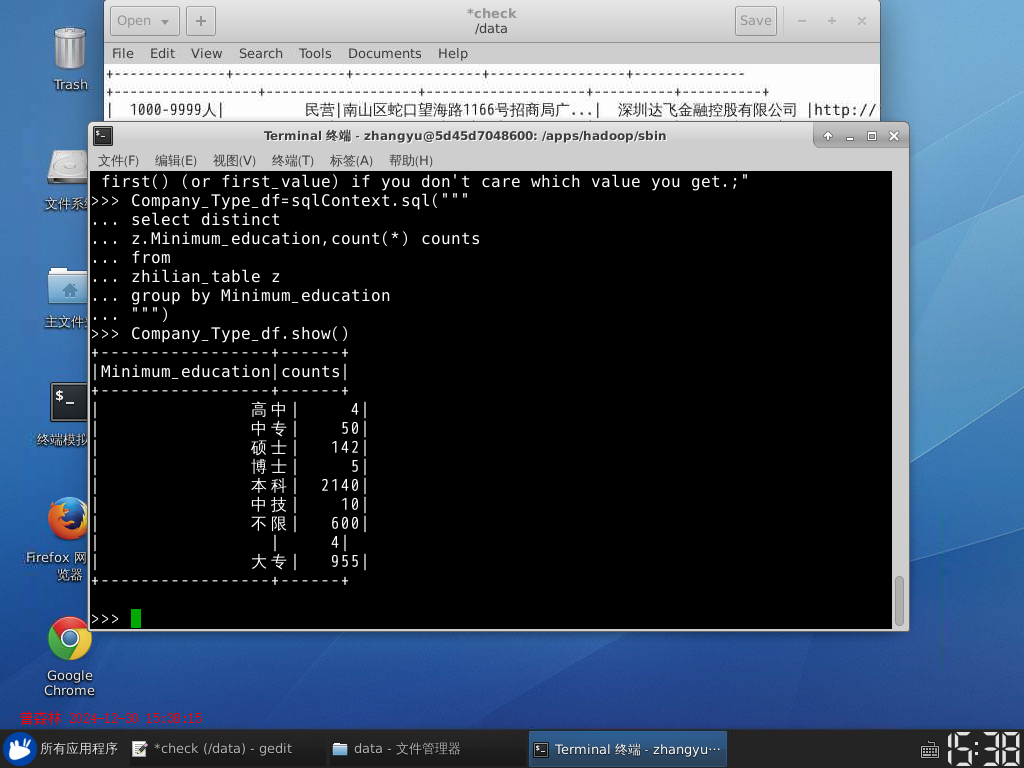
利用Spark SQL统计职位类别及数量

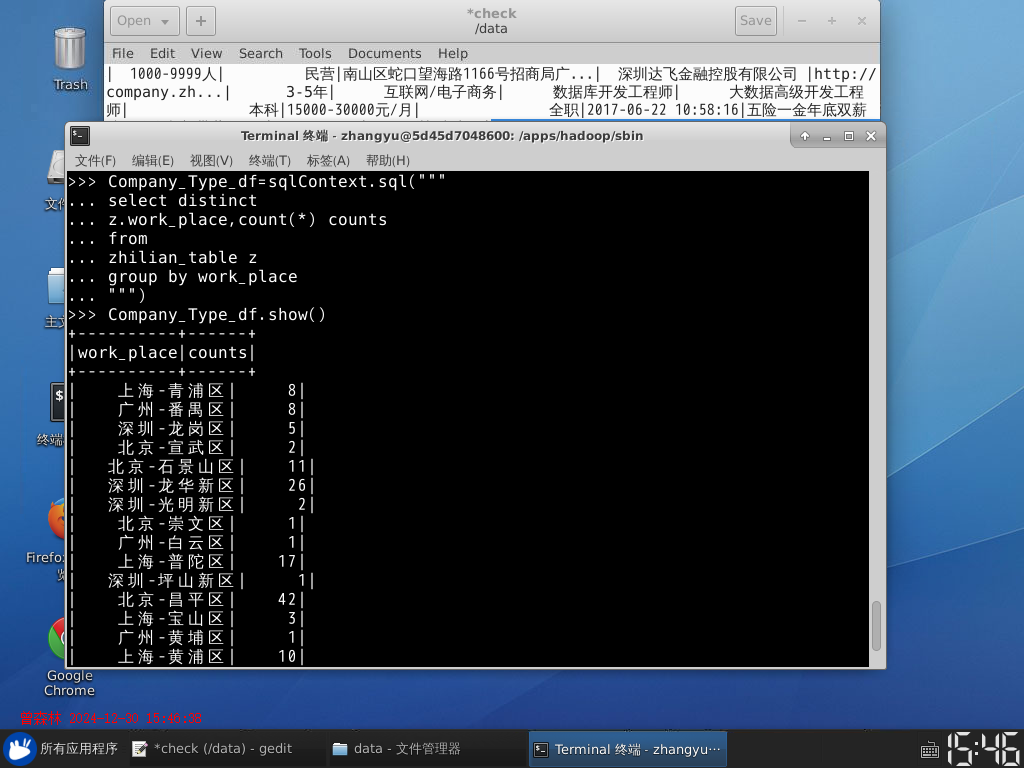
利用Spark SQL统计福利及数量



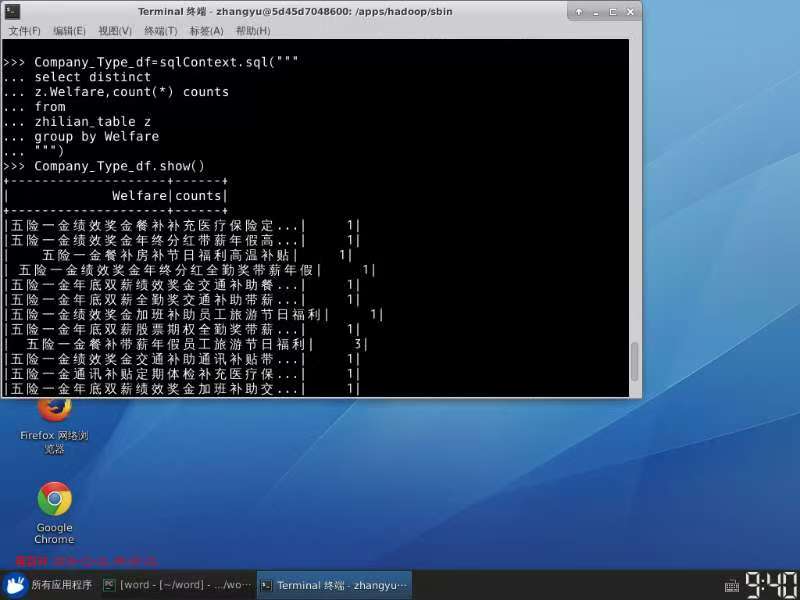












数据可视化部分：

#### 第四步：对招聘职位信息进行探索分析。

1.分析招聘信息中，公司规模的分布

2.分析招聘信息中公司性质的分布

3.分析招聘信息中职业类别的分布

4.分析招聘信息中公司主营行业的分布

5.分析招聘信息中经验的分布

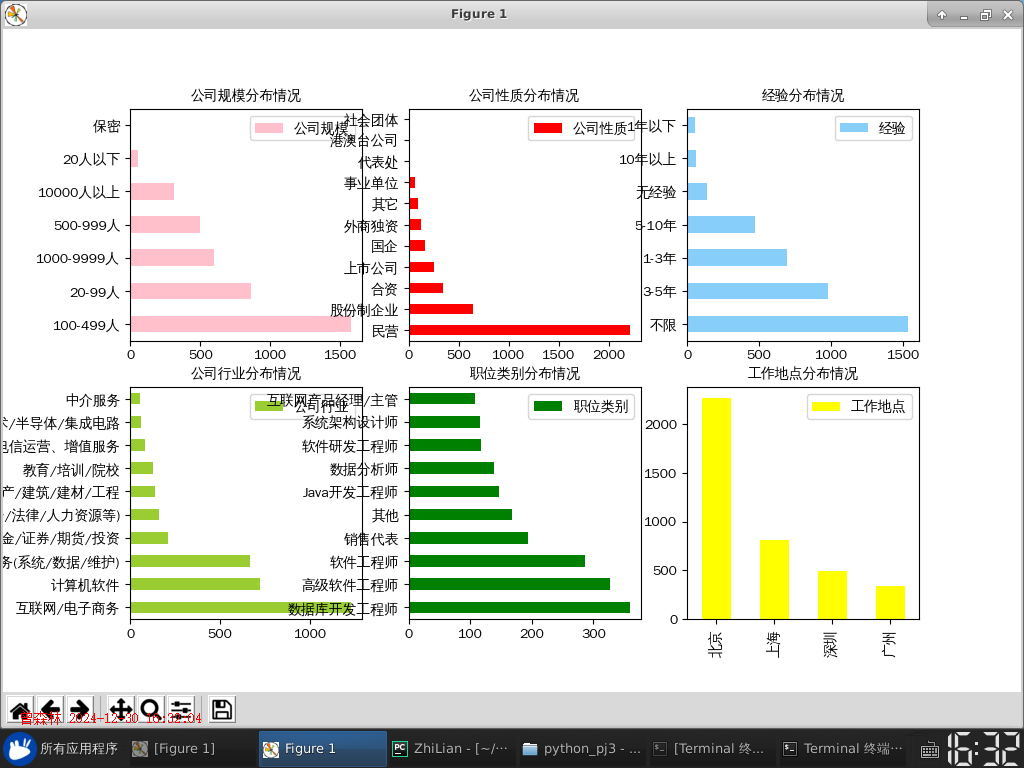
6.分析招聘信息中要求的最低学历分布

7.分析不同城市对于大数据岗位需求，即在该城市是否容易找到工作

8.分析不同城市的大数据岗位薪资的分布

9.分析不同学历要求薪资的分布

10.分析不同经验薪资的分布



由上图可知：

公司规模在100-499人的公司招聘的大数据岗位最多。

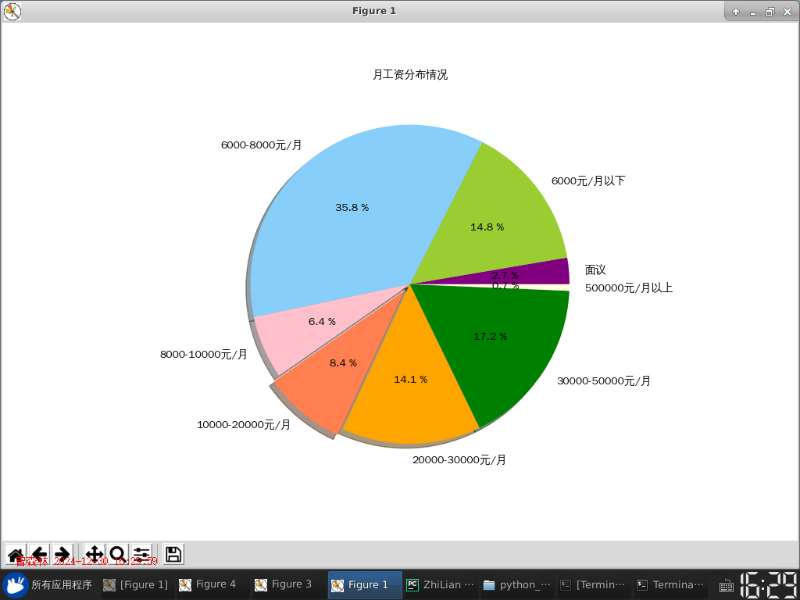
公司性质为民营企业招聘的大数据岗位最多

经验要求大部分没有明确说明，剩下的基本上集中在1-5年之间

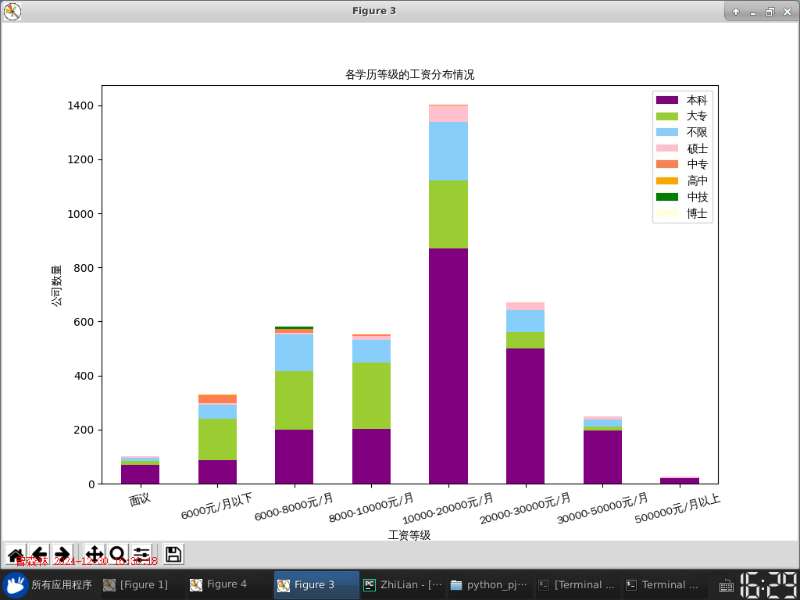
招聘公司主营行业主要集中在互联网、计算机、IT服务等行业

职位类别主要侧重于数据库开发，软件工程师等岗位

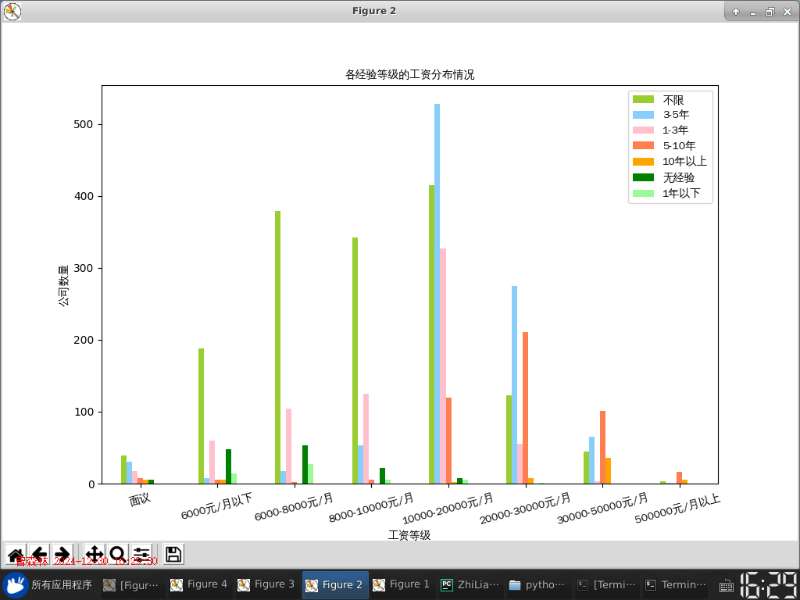
工作地点主要分布在北京，广东最少



由上图可知：公司招聘时，给的月工资主要集中在10000-20000元/月为35.8%，其次是20000-30000，10000-8000或6000-8000的也占很大比重，50000元/月以上的所占比例最小为0.7%，几乎可以省略

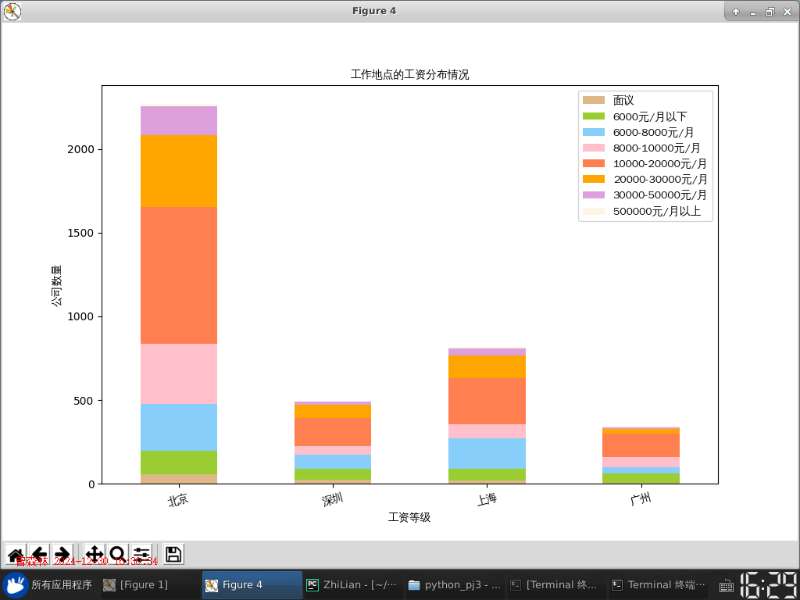


由上图可知：在各薪资范围内要求最低学历为本科的公司最多，中专、中技、高中几乎不存在



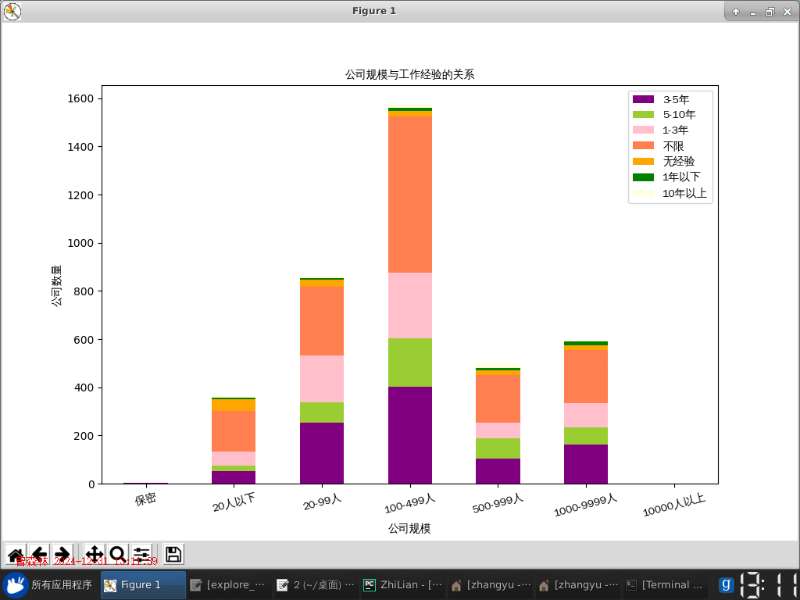
由上图可知：

* 薪资在1万元/月以下，经验要求由高到低的是1-3年，1-3年、无经验
* 薪资在1-2万元/月，经验要求由高到低的是3-5年，1-3年、5-10年
* 薪资在2-3万元/月，经验要求由高到低的是3-5年，5-10年、1-3年
* 薪资在3-5万元/月，经验要求由高到低的是5-10年，3-5年、10年以上
* 薪资在5万元/月以上，经验要求由高到低的是5-10年，3-5年、10年以上。



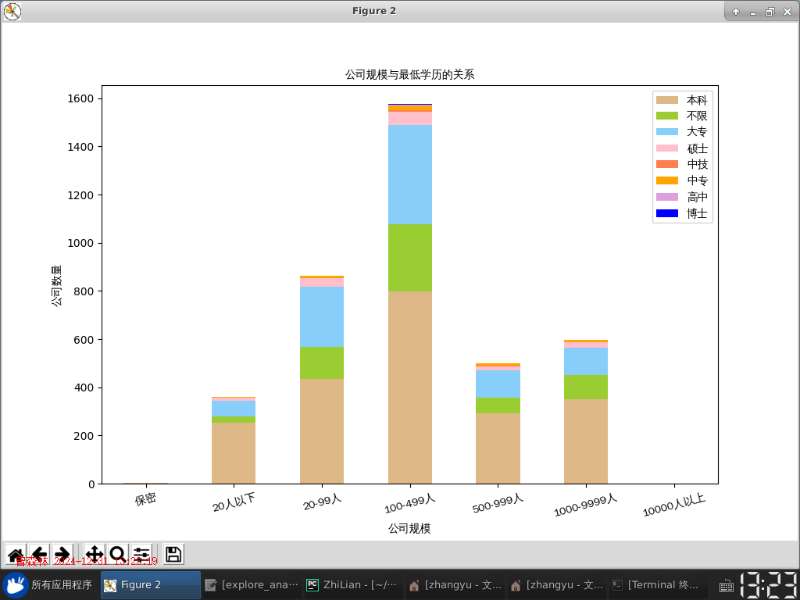
由图可知：各城市公司招聘给定的薪资10000-20000元/月所占比例最大，不同薪资分布比例基本较相似。

创新点：1，通过可视化分析公司规模与工作经验的关系



由图可知，无论公司规模大小要求工作经验不限的最多，不同工作经验在不同规模的公司中所占比列相似

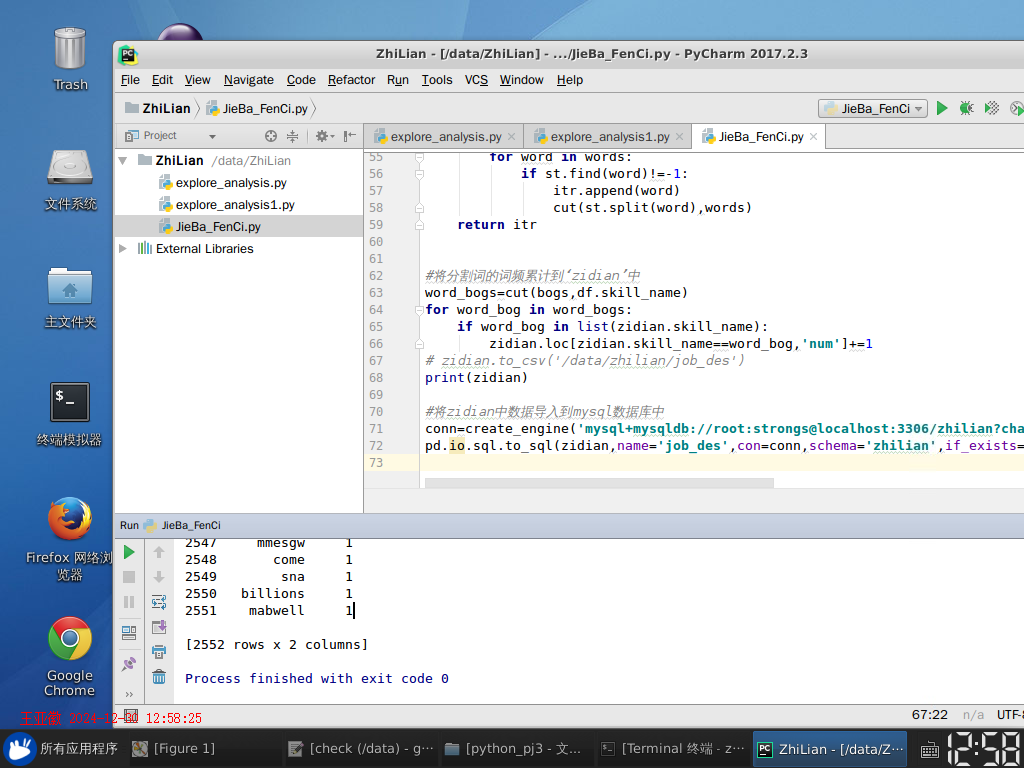
2，通过可视化分析公司规模与最低学历的关系

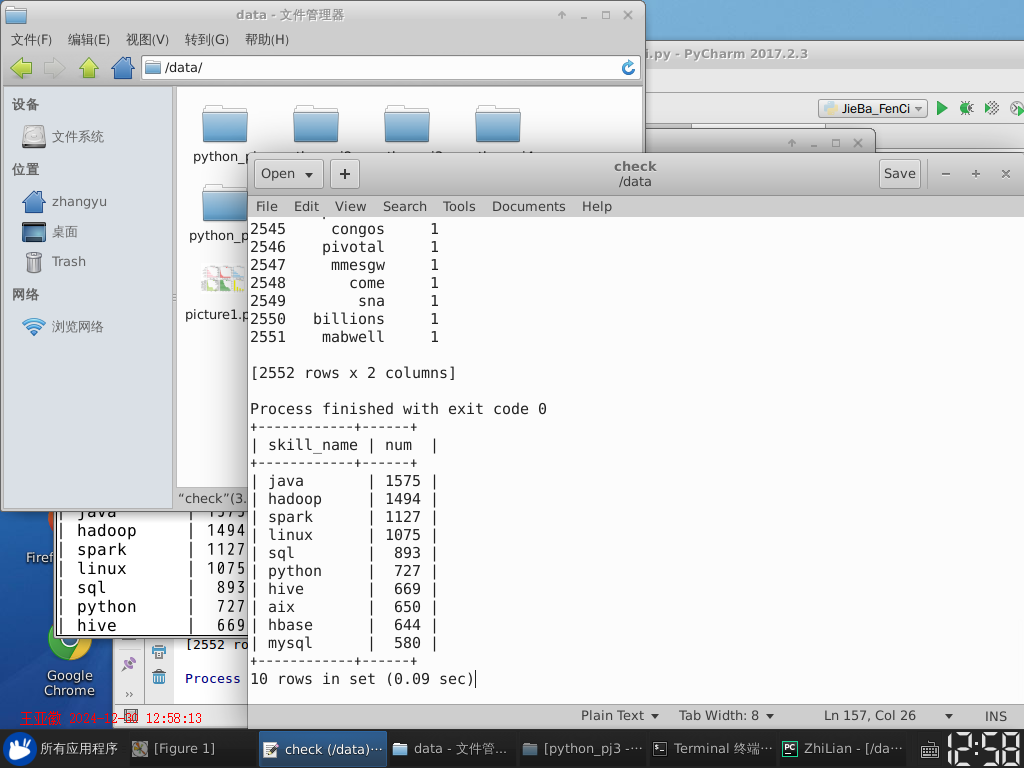


由图可知，各种不同的公司规模中要求最低学历为本科的最多，且公司规模越大对最低学历的要求越高

#### 第五步：使用结巴分词对岗位描述进行分词并将关键词统计。

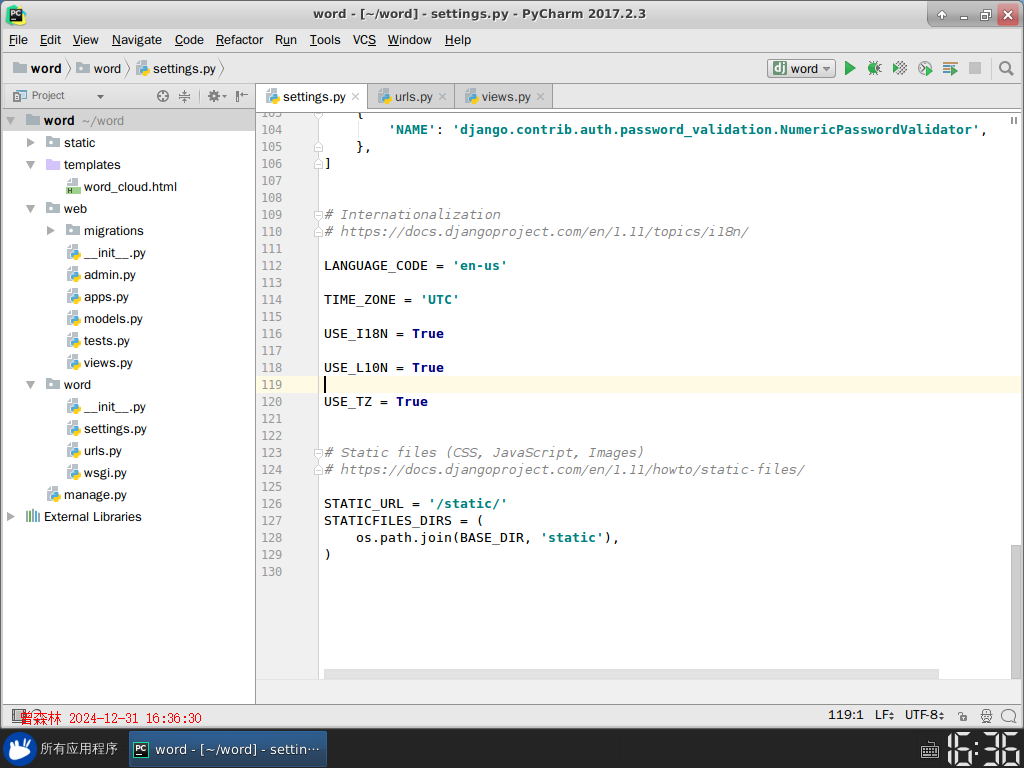
* 1. 先在pycharm中写入jieba切词代码
  2. 创建zhilian库并创建job\_des表
  3. 运行jieba对岗位描述分词获取热门的前十编程语言

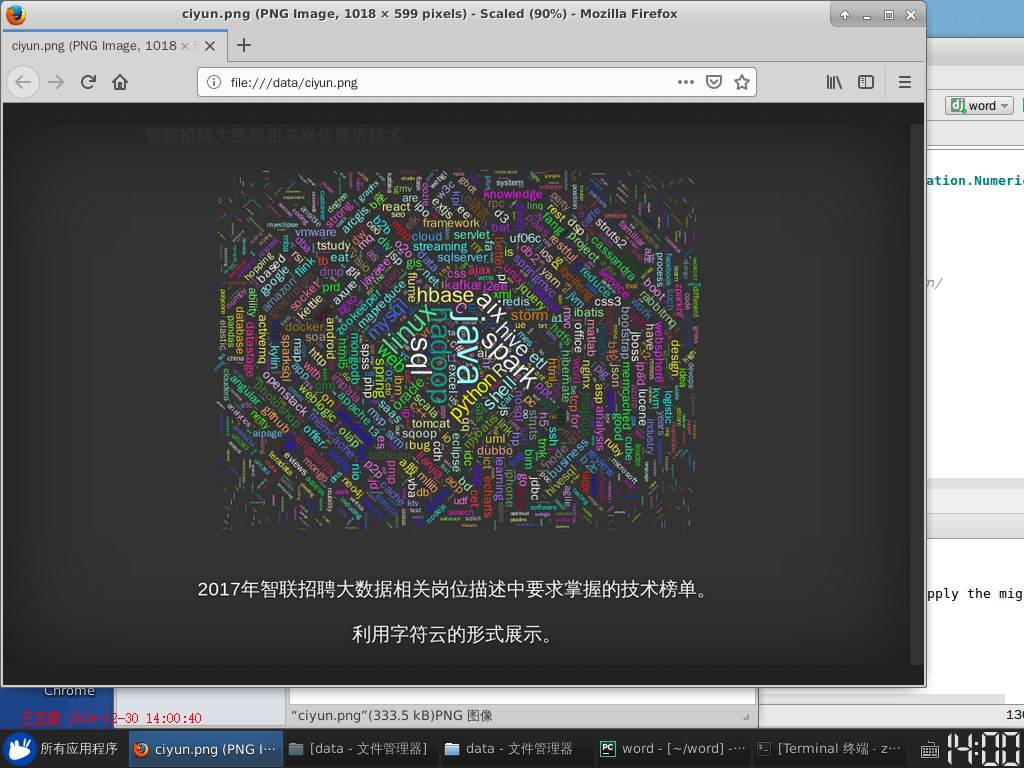




#### 第六步：利用Django Echarts将职位分析结果进行可视化。

1. 打开pycharm，创建Django项目



1. 配置好相应的代码，运行并点击链接

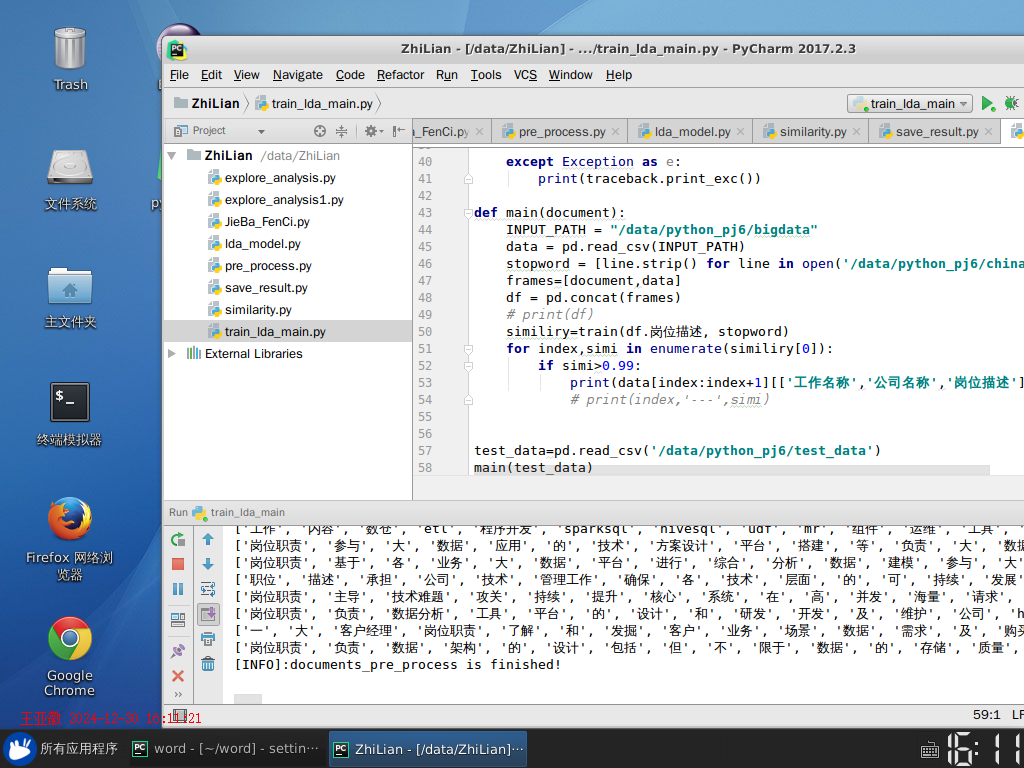
#### 第七步：建立LDA模型对职位描述进行相似度的计算

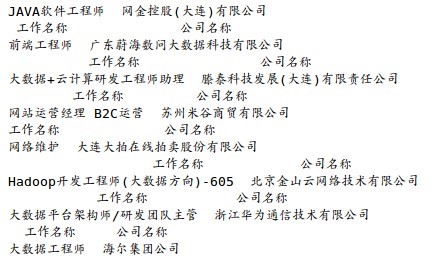
1，文本预处理过程(分词、去停用词)

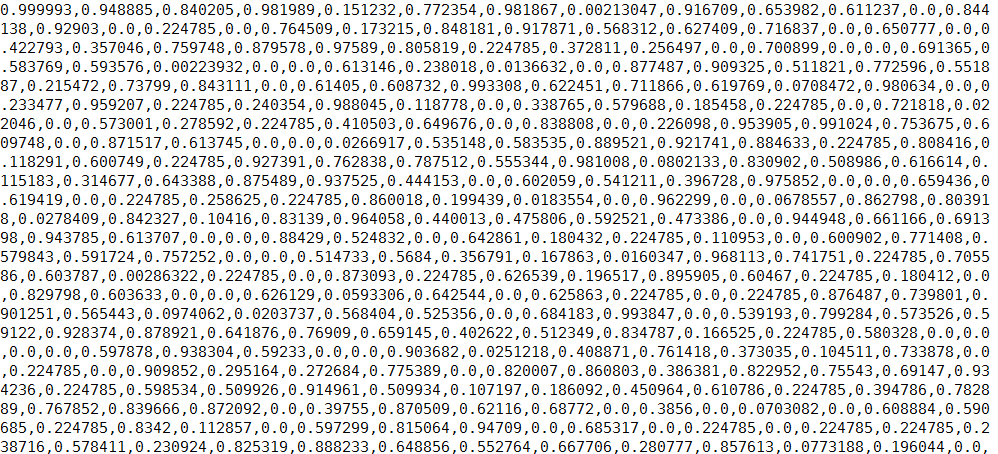
2，LDA模型向量化文本

3，计算相似度

4，结果保存







# 项目分析结果说明

通过对招聘网站职位分析，得到了相关的信息，其中统计了公司性质，经验，最低学历，月工资，职位类别，工作地点，公司规模等，得到它们的数量信息，再通过统计图将两个属性与企业数量相连，得出不同的结论，从而分析出月工资与最低学历的关系，月工资与经验的关系以及月工资与地区的关系，此外还有公司规模与最低学历，经验的关系，可以为应聘者提供很好的参考价值，还找到了其中有关数据专业最热门的前十的编程语言，并通过可视化将其表示出来。最后通过训练LDA 模型对岗位描述并进行相似度的计算，得到其相似度结果。

通过这次招聘网站职位分析实战，我们学习到了如何对类似的网站进行数据爬取数据清洗，数据挖掘，数据探索，可视化以及模型训练，最重要的是对hadoop，spark等有更深一步的了解和更好的将其应用。

# 项目代码介绍文档

**爬取招聘网站大数据职位信息：**

import urllib.request

from urllib.parse import \*

from bs4 import BeautifulSoup

import string

import random

import os

headers = [

"Mozilla/5.0 (Windows NT 6.1; Win64; rv:27.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"

"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:27.0) Gecko/20100101 Firfox/27.0"

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"

"Mozilla/5.0 (Windows NT 10.0; WOW64; rv:10.0) Gecko/20100101 Firfox/10.0"

"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/21.0.1180.110 Safari/537.36"

"Mozilla/5.0 (X11; Ubuntu; Linux i686 rv:10.0) Gecko/20100101 Firfox/27.0"

"Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.1 (KHTML, like Gecko) Chrome/34.0.1838.2 Safari/537.36"

"Mozilla/5.0 (X11; Ubuntu; Linux i686 rv:27.0) Gecko/20100101 Firfox/27.0"

"Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36"

'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36'

]

def get\_content(url, headers):

#@url：需要登录的网址

#@headers：模拟的登陆的终端

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*模拟登陆获取网址\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

random\_header = random.choice(headers)

req = urllib.request.Request(url)

req.add\_header("User-Agent", random\_header)

req.add\_header("Host", "192.168.1.150:40000")

try:

html = urllib.request.urlopen(req)

contents = html.read()

if isinstance(contents, bytes):

contents = contents.decode('utf-8')

else:

print('service mysql restart')

return (contents)

except Exception as e:

print(e)

def get\_links\_from(page):

#@page：表示第几页信息

#@urls：所有列表的超链接，即子页网址

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*此网站需要模拟登陆\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

#返回全部子网页地址

urls = []

for i in range(1,page):

url='http://192.168.1.150:40000/wuyou/{0}'.format(i)

url = quote(url, safe=string.printable)

info = get\_content(url, headers)

soup = BeautifulSoup(info, "lxml")

link\_urls = soup.select('div.dw\_table div.el span.t2 a')

for url in link\_urls:

oneurl="http://192.168.1.150:40000"+url.get('href')

urls.append(oneurl)

# print(urls)

return (urls)

def get\_recuite\_info(page):

#获取网页信息

urls = get\_links\_from(page)

path='/data/zhaopin/'

if os.path.exists(path)==False:

os.makedirs(path)

for url in urls:

print(url)

file=url.split('/')[-1]

print(file)

str=url.split('/')[2].split('.')[0]

html = get\_content(url, headers)

if html!=None and file!='':

with open(path+file,'w') as f:

f.write(html)

#\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*获取信息\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

if \_\_name\_\_ == '\_\_main\_\_':

get\_recuite\_info(20)

使用BeautifulSoup清洗职位信息网页：

# -\*- coding: utf-8 -\*-

from bs4 import BeautifulSoup

import pandas as pd

import os

def get\_html\_info(html):

'''''

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*获取网页的有用信息并保存成字典形式\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

'''

try:

soup = BeautifulSoup(html, "lxml") # 设置解析器为“lxml”

occ\_name = soup.select('div.fixed-inner-box h1')[0]

com\_name = soup.select('div.fixed-inner-box h2 ')[0]

welfare = soup.select('div.welfare-tab-box')[0]

wages = soup.select('div.terminalpage-left strong')[0]

date = soup.select('div.terminalpage-left strong')[2]

exper = soup.select('div.terminalpage-left strong')[4]

num = soup.select('div.terminalpage-left strong')[6]

area = soup.select('div.terminalpage-left strong')[1]

nature = soup.select('div.terminalpage-left strong')[3]

Edu = soup.select('div.terminalpage-left strong')[5]

cate = soup.select('div.terminalpage-left strong')[7]

com\_scale = soup.select('ul.terminal-ul.clearfix li strong')[8]

com\_nature = soup.select('ul.terminal-ul.clearfix li strong')[9]

com\_cate = soup.select('ul.terminal-ul.clearfix li strong')[10]

com\_url = soup.select('div.fixed-inner-box h2 a')[0]

com\_address = soup.select('ul.terminal-ul.clearfix li strong')[-1]

job\_descritions1 = soup.select('div.tab-inner-cont')[0]

job\_descritions=job\_descritions1.select('p')[:-1]

data = {

"工作名称": occ\_name.text.strip(),

"公司名称": com\_name.text,

"公司网址": com\_url.get('href'),

"福利": welfare.text.strip(),

"月工资": wages.text.strip(),

"发布日期": date.text.strip(),

"经验": exper.text.strip(),

"人数": num.text.strip(),

"工作地点": area.text.strip(),

"工作性质": nature.text.strip(),

"最低学历": Edu.text.strip(),

"职位类别": cate.text.strip(),

"公司规模": com\_scale.text.strip(),

"公司性质": com\_nature.text.strip(),

"公司行业": com\_cate.text.strip(),

"公司地址": com\_address.text.strip(),

"岗位描述": [job\_descrition.text.strip() for job\_descrition in job\_descritions],

}

# print(data)

return (data)

except Exception:

pass

# 将全部信息保存成DataFrame形式，去除无用信息

def get\_htmls\_all\_info(path):

df = pd.DataFrame({

"工作名称": [],

"公司名称": [],

"公司网址": [],

"福利": [],

"月工资": [],

"发布日期": [],

"经验": [],

"人数": [],

"工作地点": [],

"工作性质": [],

"最低学历": [],

"职位类别": [],

"公司规模": [],

"公司性质": [],

"公司行业": [],

"公司地址": [],

"岗位描述": [],

})

dirs = os.listdir(path)

for dir in dirs:

#print(dir)

if dir.find('swp'):

dir=dir.strip('.'and '.swp')

p = os.path.join(path, dir)

html = open(p).read()

data = get\_html\_info(html)

print(data)

df = df.append(data, ignore\_index=True)

return df

if \_\_name\_\_ == '\_\_main\_\_':

path='/data/python\_pj2/jobs.zhaopin.com'

df=get\_htmls\_all\_info(path)

df.to\_csv('/data/python\_pj2/bigdata', index=False)

print(df)

使用PySpark对职位数据进行分析

RDD1 = sc.textFile("/bigdata")

RDD1.count()#使用textFile()方法读取HDFS上的bigdata文件，赋值给RDD1并统计文件内容有多少行

RDD2=RDD1.map(lambda line:line.split(","))#使用map函数处理每一项数据，用lambda语句创建匿名函数传入line参数，在匿名函数中，line.split(“,”)表示按照逗号分隔获取每一个字段

from pyspark.sql import Row

zhilian\_Rows = RDD2.map(lambda p:

Row(

num\_people=p[0],

Company\_name=p[1],

Company\_address=p[2],

Company\_Type=p[3],

Company\_website=p[4],

Industry=p[5],

Company\_Size=p[6],

Release\_date=p[7],

Job\_name=p[8],

work\_place=p[9],

Nature\_of\_the\_work=p[10],

Minimum\_education=p[11],

Monthly\_salary=p[12],

Welfare=p[13],

Experience=p[14],

Job\_Categories=p[15]# 导入row模块，通过RDD2创建DataFrame，定义DataFrame的每一个字段名与数据类型

)

)

from pyspark.sql import SQLContext

sqlContext = SQLContext(sc)

zhilian\_df = sqlContext.createDataFrame(zhilian\_Rows)

zhilian\_df.printSchema()

#创建了zhilian\_Rows之后，使用sqlContext.createDataFrame()方法写入zhilian\_Rows数据，创建DataFrame，然后使用.printSchema()方法查看DataFrames的Schema

zhilian\_df.show(5)# 接下来，我们可以使用.show()方法来查看前5行数据

df=zhilian\_df.alias("df")

df.show(5)# 我们也可以使用.alias()方法来为DadaFrame创建别名

df.select("Company\_Type").groupby("Company\_Type").count().show()

#使用DataFrame统计公司性质及数量

sqlContext.registerDataFrameAsTable(df, "zhilian\_table")#我们之前创建DataFrame，下面我们使用registerTempTable方法将df转换为zhilian\_table表

sqlContext.sql("select count(\*) counts from zhilian\_table").show()#接下来，我们可以使用sqlContext.sql()输入sql语句，使用select关键字查询文件内容行数，并使用from关键字指定要查询的表，最后使用show()方法显示查询结果

sqlContext.sq

Company\_Type\_df=sqlContext.sql("""

select distinct

z.Experience,count(\*) counts

from

zhilian\_table z

group by Experience

""")

Company\_Type\_df.show()

#使用PySpark SQL统计经验要求及数量

Company\_Type\_df=sqlContext.sql("""

select distinct

z.Company\_Size,count(\*) counts

from

zhilian\_table z

group by Company\_Size

""")

Company\_Type\_df.show()

#使用PySpark SQL统计公司规模及数量

Company\_Type\_df=sqlContext.sql("""

select distinct

z.Minimum\_education,count(\*) counts

from

zhilian\_table z

group by Minimum\_education

""")

Company\_Type\_df.show()

#使用PySpark SQL统计最低学历及数量

Company\_Type\_df=sqlContext.sql("""

select distinct

z.work\_place,count(\*) counts

from

zhilian\_table z

group by work\_place

""")

Company\_Type\_df.show()

#使用PySpark SQL统计工作地区及数量

Company\_Type\_df=sqlContext.sql("""

select distinct

z.Job\_Categories,count(\*) counts

from

zhilian\_table z

group by Job\_Categories

""")

Company\_Type\_df.show()

#使用PySpark SQL统计职业类别及数量

对招聘职位信息进行探索分析

import pandas as pd

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.font\_manager as fm

fontPath ="/usr/share/fonts/truetype/wqy/wqy-zenhei.ttc"

font = fm.FontProperties(fname=fontPath, size=10)

data=pd.read\_csv('/data/python\_pj3/bigdata',)

print(data.shape,data.columns)

data.loc[(data.经验=='3年以上'),'经验']='3-5年'

#公司规模分布情况

plt.figure(figsize=(12,10))

plt.subplot2grid((2,3),(0,0))

a=data['公司规模'].value\_counts().plot(kind='barh',title='公司规模分布情况',color='pink')

a.xaxis.get\_label().set\_fontproperties(font)

a.yaxis.get\_label().set\_fontproperties(font)

a.legend(loc='best',prop=font)

for label in ([a.title]+a.get\_xticklabels()+a.get\_yticklabels()):

label.set\_fontproperties(font)

#公司性质分布情况

plt.subplot2grid((2,3),(0,1))

b=data['公司性质'].value\_counts().plot(kind='barh',title='公司性质分布情况',color='red')

b.xaxis.get\_label().set\_fontproperties(font)

b.yaxis.get\_label().set\_fontproperties(font)

b.legend(loc='best',prop=font)

for label in ([b.title]+b.get\_xticklabels()+b.get\_yticklabels()):

label.set\_fontproperties(font)

#经验分布情况

# plt.subplot2grid((2,2),(1,0),colspan=2)

plt.subplot2grid((2,3),(0,2))

c=data['经验'].value\_counts().plot(kind='barh',title='经验分布情况',color='lightskyblue')

c.xaxis.get\_label().set\_fontproperties(font)

c.yaxis.get\_label().set\_fontproperties(font)

c.legend(loc='best',prop=font)

for label in ([c.title]+c.get\_xticklabels()+c.get\_yticklabels()):

label.set\_fontproperties(font)

#公司行业分布情况

plt.subplot2grid((2,3),(1,0))

d=data['公司行业'].value\_counts().sort\_values(ascending=False).head(10).plot(kind='barh',title='公司行业分布情况',color='yellowgreen')

d.xaxis.get\_label().set\_fontproperties(font)

d.yaxis.get\_label().set\_fontproperties(font)

d.legend(loc='best',prop=font)

for label in ([d.title]+d.get\_xticklabels()+d.get\_yticklabels()):

label.set\_fontproperties(font)

#职位类别分布情况

plt.subplot2grid((2,3),(1,1))

d=data['职位类别'].value\_counts().sort\_values(ascending=False).head(10).plot(kind='barh',title='职位类别分布情况',color='green')

d.xaxis.get\_label().set\_fontproperties(font)

d.yaxis.get\_label().set\_fontproperties(font)

d.legend(loc='best',prop=font)

for label in ([d.title]+d.get\_xticklabels()+d.get\_yticklabels()):

label.set\_fontproperties(font)

#工作地点分布情况

plt.subplot2grid((2,3),(1,2))

d=data['工作地点'].str.split('-',expand=True)[0].value\_counts().plot(kind='bar',title='工作地点分布情况',color='yellow',label='工作地点')

d.xaxis.get\_label().set\_fontproperties(font)

d.yaxis.get\_label().set\_fontproperties(font)

d.legend(loc='best',prop=font)

# print(d.get\_legend\_handles\_labels())

for label in ([d.title]+d.get\_xticklabels()+d.get\_yticklabels()):

label.set\_fontproperties(font)

plt.show()

#月工资与其他因素的关系

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.font\_manager as fm

fontPath ="/usr/share/fonts/truetype/wqy/wqy-zenhei.ttc"

font = fm.FontProperties(fname=fontPath, size=10)

data=pd.read\_csv('/data/python\_pj3/bigdata')

print(data.shape)

print(data.columns)

# print([data.职位类别.value\_counts().index if data.职位类别.value\_counts()<10==False:])

data.loc[((data.职位类别=='客户代表')|(data.职位类别=='电话销售')|(data.职位类别=='大客户销售代表')),'职位类别']='销售代表'

月工资=data['月工资'].str.strip('元/月').str.split('-',expand=True)

月工资.columns=['月工资\_min','月工资\_max']

data['月工资\_min']=月工资['月工资\_min']

data['月工资\_max']=月工资['月工资\_max']

data.loc[(data.月工资\_min=='面议'),'月工资\_min']=0

data.loc[(data.月工资\_min=='1000元/月以下' ),'月工资\_min']=1

data.loc[(data.月工资\_min=='100000元/月以上'),'月工资\_min']=7

data.loc[(data.月工资\_max.isnull()),'月工资\_max']=0

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

data.月工资\_min=data.月工资\_min.astype(int)

data.月工资\_max=data.月工资\_max.astype(int)

月工资\_mean=(data.月工资\_min+data.月工资\_max)/2

data.loc[((data.月工资\_min==0) & (data.月工资\_max==0) & (data.月工资.notnull()) ),'月工资']=0

data.loc[((data.月工资\_min==1) & (data.月工资\_max==0) & (data.月工资.notnull()) ),'月工资']=1

data.loc[((data.月工资\_min<6000) & (data.月工资\_min>8)),'月工资'] = 1

data.loc[((((data.月工资\_min>=6000) & (data.月工资\_max<=8000)) | ((6000<月工资\_mean)&(月工资\_mean<8000))) & (data.月工资.notnull())),'月工资'] = 2

data.loc[((((data.月工资\_min>=8000) & (data.月工资\_max<=10000)) | ((8000<=月工资\_mean)&(月工资\_mean<10000))) & (data.月工资.notnull()) ),'月工资']=3

data.loc[((((data.月工资\_min>=10000) & (data.月工资\_max<=20000)) | ((10000<=月工资\_mean)&(月工资\_mean<20000))) & (data.月工资.notnull())),'月工资']=4

# data.loc[((data.月工资\_min>=15000) & (data.月工资\_max<=20000) & (data.月工资.notnull()) ),'月工资']=5

data.loc[((((data.月工资\_min>=20000) & (data.月工资\_max<=30000)) | ((20000<=月工资\_mean)&(月工资\_mean<30000))) & (data.月工资.notnull()) ),'月工资']=5

data.loc[((((data.月工资\_min>=30000) & (data.月工资\_max<=50000)) | ((30000<=月工资\_mean)&(月工资\_mean<50000))) & (data.月工资.notnull())),'月工资']=6

data.loc[(((data.月工资\_min>=50000) | (50000<=月工资\_mean)) & (data.月工资.notnull()) ),'月工资']=7

data.loc[((data.月工资\_min==7) & (data.月工资\_max==0) & (data.月工资.notnull()) ),'月工资']=7

print(data.月工资.value\_counts(sort=False))

#月工资分布情况

fig=plt.figure()

fracs=data.月工资.value\_counts(sort=False)

labels=['面议','6000元/月以下','6000-8000元/月','8000-10000元/月','10000-20000元/月','20000-30000元/月','30000-50000元/月','500000元/月以上 ']

colors = ['purple','yellowgreen','lightskyblue','pink','coral','orange','green','lightyellow']

explode=[0,0,0,0,0.05,0,0,0]

patchs,l\_text,p\_text=plt.pie(x=fracs,explode=explode, labels=labels,colors=colors, autopct='%3.1f %%',

shadow=True, labeldistance=1.1, startangle=0, pctdistance=0.6)

plt.title('月工资分布情况',fontproperties=font)

for t in (l\_text+p\_text):

t.set\_fontproperties(font)

#各经验等级的月工资情况

data.loc[(data.经验=='3年以上'),'经验']='3-5年'

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

df=pd.DataFrame([data.经验[data.月工资==i].value\_counts().rename(i) for i in range(8)])

print(df)

df.plot(kind='bar',colors=['yellowgreen','lightskyblue','pink','coral','orange','green','palegreen'])

plt.legend(loc='best',prop=font)

plt.xticks(df.index,labels,rotation=15,fontproperties=font)

plt.title('各经验等级的工资分布情况',fontproperties=font)

plt.xlabel('工资等级',fontproperties=font)

plt.ylabel('公司数量',fontproperties=font)

#月工资与学历

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

df=pd.DataFrame([data.最低学历[data.月工资==i].value\_counts().rename(i) for i in range(8)])

print(df)

df.plot(kind='bar',stacked=True,colors=colors)

plt.legend(loc='best',prop=font)

plt.xticks(df.index,labels,rotation=15,fontproperties=font)

plt.title('各学历等级的工资分布情况',fontproperties=font)

plt.xlabel('工资等级',fontproperties=font)

plt.ylabel('公司数量',fontproperties=font)

#月工资与工作地点

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

df=pd.DataFrame([data.工作地点.str.split('-',expand=True)[0][data.月工资==i].value\_counts().rename(i) for i in range(8)]).T

print(df)

colors=['burlywood','yellowgreen','lightskyblue','pink','coral','orange','plum','oldlace']

d=df.plot(kind='bar',stacked=True,colors=colors)

plt.legend(loc='best',prop=font,labels=labels)

print(d.get\_legend\_handles\_labels())

plt.xticks([i for i in range(len(df.index))],df.index,rotation=15,fontproperties=font)

plt.title('工作地点的工资分布情况',fontproperties=font)

plt.xlabel('工资等级',fontproperties=font)

plt.ylabel('公司数量',fontproperties=font)

print(df.values)

plt.show()

#公司规模与经验和公司规模与最低学历的关系

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

import matplotlib.font\_manager as fm

fontPath ="/usr/share/fonts/truetype/wqy/wqy-zenhei.ttc"

font = fm.FontProperties(fname=fontPath, size=10)

data=pd.read\_csv('/data/python\_pj3/bigdata')

print(data.shape)

print(data.columns)

# print([data.职位类别.value\_counts().index if data.职位类别.value\_counts()<10==False:])

data.loc[((data.职位类别=='客户代表')|(data.职位类别=='电话销售')|(data.职位类别=='大客户销售代表')),'职位类别']='销售代表'

公司规模 =data['公司规模'].str.strip('人').str.split('-',expand=True)

公司规模.columns=['公司规模\_min','公司规模\_max']

data['公司规模\_min']=公司规模['公司规模\_min']

data['公司规模\_max']=公司规模['公司规模\_max']

data.loc[(data.公司规模\_min=='保密'),'公司规模\_min']=0

data.loc[(data.公司规模\_min=='20人以下' ),'公司规模\_min']=1

data.loc[(data.公司规模\_min=='10000人以上'),'公司规模\_min']=6

data.loc[(data.公司规模\_max.isnull()),'公司规模\_max']=0

data.loc[(data.公司规模\_min.isnull()),'公司规模\_min']=0

print(data.公司规模\_min)

print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

data.公司规模\_min=data.公司规模\_min.astype(int)

data.公司规模\_max=data.公司规模\_max.astype(int)

公司规模\_mean=(data.公司规模\_min+data.公司规模\_max)/2

data.loc[((data.公司规模\_min==0) & (data.公司规模\_max==0) & (data.公司规模.notnull()) ),'公司规模']=0

data.loc[((data.公司规模\_min==1) & (data.公司规模\_max==0) & (data.公司规模.notnull()) ),'公司规模']=1

data.loc[((data.公司规模\_min<20) & (data.公司规模\_min>1)),'公司规模'] = 1

data.loc[((((data.公司规模\_min>=20) & (data.公司规模\_max<=99)) | ((20<公司规模\_mean)&(公司规模\_mean<99))) & (data.公司规模.notnull())),'公司规模'] = 2

data.loc[((((data.公司规模\_min>=100) & (data.公司规模\_max<=499)) | ((100<=公司规模\_mean)&(公司规模\_mean<499))) & (data.公司规模.notnull()) ),'公司规模']=3

data.loc[((((data.公司规模\_min>=500) & (data.公司规模\_max<=999)) | ((500<=公司规模\_mean)&(公司规模\_mean<999))) & (data.公司规模.notnull())),'公司规模']=4

data.loc[((((data.公司规模\_min>=1000) & (data.公司规模\_max<=9999)) | ((1000<=公司规模\_mean)&(公司规模\_mean<9999))) & (data.公司规模.notnull()) ),'公司规模']=5

data.loc[(((data.公司规模\_min>=10000) | (10000<=公司规模\_mean)) & (data.公司规模.notnull()) ),'公司规模']=6

data.loc[((data.公司规模\_min==7) & (data.公司规模\_max==0) & (data.公司规模.notnull()) ),'公司规模']=6

print(data.公司规模.value\_counts(sort=False))

colors=['purple','yellowgreen','pink','coral','orange','green','lightyellow']

labels=['保密','20人以下','20-99人','100-499人','500-999人','1000-9999人','10000人以上']

data.loc[(data.经验=='3年以上'),'经验']='3-5年'

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

df=pd.DataFrame([data.经验[data.公司规模==i].value\_counts().rename(i) for i in range(7)])

print(df)

df.plot(kind='bar',stacked=True,colors=colors)

plt.legend(loc='best',prop=font)

plt.xticks(df.index,labels,rotation=15,fontproperties=font)

plt.title('公司规模与工作经验的关系',fontproperties=font)

plt.xlabel('公司规模',fontproperties=font)

plt.ylabel('公司数量',fontproperties=font)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

df=pd.DataFrame([data.最低学历[data.公司规模==i].value\_counts().rename(i) for i in range(7)])

print(df)

colors=['burlywood','yellowgreen','lightskyblue','pink','coral','orange','plum','blue']

df.plot(kind='bar',stacked=True,colors=colors)

plt.legend(loc='best',prop=font)

plt.xticks(df.index,labels,rotation=15,fontproperties=font)

plt.title('公司规模与最低学历的关系',fontproperties=font)

plt.xlabel('公司规模',fontproperties=font)

plt.ylabel('公司数量',fontproperties=font)

print(df.values)

plt.show()

使用结巴分词对岗位描述进行分词并将关键词统计

import pandas as pd

import re

import numpy as np

import jieba

import jieba.analyse

import pymysql

from sqlalchemy import create\_engine

pymysql.install\_as\_MySQLdb()

#加载数据

data=pd.read\_csv('/data/python\_pj4/bigdata',header=None,names=[ '人数', '公司名称', '公司地址', '公司性质', '公司网址', '公司行业', '公司规模',

'发布日期', '岗位描述', '工作名称', '工作地点', '工作性质', '最低学历', '月工资', '福利', '经验', '网址',

'职位类别'])

data=data[1:]

# 使用pandas中的方法,取出岗位描述列中符合的内容

job\_des=data.岗位描述.str.strip("[]").str.replace(',','').str.replace("'","").str.replace(r'\\xa0','').str.replace(' '\*3 or ' '\*4,'|').str.split('|',expand=True)[0]

# print(job\_des)

#jieba分词，并统计词频

zidian={}

r1=re.compile(r'\w') #使用正则表达式,筛选[A-Za-z0-9\_]

r2=re.compile(r'[^\d]') #使用正则表达式,筛选[0-9\_]

r4=re.compile(r'[^\_]')

r3=re.compile(r'[\u4e00-\u9fa5]') #使用正则表达式,筛选纯中文

stopkeyword=[line.strip() for line in open('/home/zhangyu/stopword').readlines()] #加载停用词

bogs=[] #用于存储长的英文语句

for i in job\_des:

seg\_list = jieba.cut(i)

# print('Dafault Mode:', ' '.join(seg\_list))

for word in seg\_list:

if word not in stopkeyword and r1.match(word) and r2.match(word)and r3.match(word)==None and r4.match(word): #筛选语句中的英文

word=word.lower() #小写化所有词

if len(word)>=10:

bogs.append(word)

elif len(word)!=1 or word in ['c','r']:

#统计词频

if word in zidian:

zidian[word]+=1

else:

zidian[word]=1

#将字典的值逆序

zidian=sorted(zidian.items(),key=lambda item:item[1],reverse=True)

zidian=pd.DataFrame(zidian,columns=['skill\_name','num'])

zidian.loc[(zidian.skill\_name=='r'),'skill\_name']="R"

zidian.loc[(zidian.skill\_name=='c'),'skill\_name']="C"

df=zidian[zidian.num>70]

#用于分割长语句中的关键词

def cut(sentences,words):

itr=[]

for st in sentences:

for word in words:

if st.find(word)!=-1:

itr.append(word)

cut(st.split(word),words)

return itr

#将分割词的词频累计到‘zidian’中

word\_bogs=cut(bogs,df.skill\_name)

for word\_bog in word\_bogs:

if word\_bog in list(zidian.skill\_name):

zidian.loc[zidian.skill\_name==word\_bog,'num']+=1

# zidian.to\_csv('/data/zhilian/job\_des')

print(zidian)

#将zidian中数据导入到mysql数据库中

conn=create\_engine('mysql+mysqldb://root:strongs@localhost:3306/zhilian?charset=utf8')

pd.io.sql.to\_sql(zidian,name='job\_des',con=conn,schema='zhilian',if\_exists='append',index=False)

利用Django Echarts将职位分析结果进行可视化

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.mysql',

'NAME': 'zhilian',

'USER': 'root',

'PASSWORD': 'strongs',

'HOST': 'localhost',

'PORT': '3306',

}

}# Django项目建成后, 默认设置了使用SQLite数据库，把DATABASES修改MySQL的数据库设置, 在PyCharm打开word/word/settings.py 中可以查看和修改数据库设置

from django.conf.urls import url

from web import views

urlpatterns = [

url(r'^word$', views.word),

url(r'^word\_data/$', views.word\_data),

]

#url设置

from web import models

import json

from django.db import connection

import pymysql

from django.shortcuts import render, HttpResponse

def word(request):

return render(request, 'word\_cloud.html')

def word\_data(request):

# select

# conn = pymysql.connect(host='localhost', user='root', passwd='123456', db='zhilian', port=3306, charset='utf8')

cur = connection.cursor()

cur.execute('select \* from job\_des order by num desc ;')

data\_obj = cur.fetchall()

data\_list = []

for i in data\_obj:

data\_list.append({'name': i[0], 'value': i[1]})

cur.close()

connection.close()

return HttpResponse(json.dumps(data\_list))

#views函数

MIDDLEWARE = [

'django.middleware.security.SecurityMiddleware',

'django.contrib.sessions.middleware.SessionMiddleware',

'django.middleware.common.CommonMiddleware',

# 'django.middleware.csrf.CsrfViewMiddleware',

'django.contrib.auth.middleware.AuthenticationMiddleware',

'django.contrib.messages.middleware.MessageMiddleware',

'django.middleware.clickjacking.XFrameOptionsMiddleware',

]#csrf设置

建立LDA模型对职位描述进行相似度的计算

#-\*- coding:utf-8

'''

preprocess.py

这个文件的作用是做文档预处理，

讲每篇文档，生成相应的token\_list

只需执行最后documents\_pre\_process函数即可。

'''

from nltk.tokenize import WordPunctTokenizer

import traceback

import jieba

from nltk.corpus import stopwords

from nltk.stem.lancaster import LancasterStemmer

from collections import defaultdict

import re

# 分词 - 英文

def tokenize(document):

try:

token\_list = WordPunctTokenizer().tokenize(document)

#print("[INFO]: tokenize is finished!")

return token\_list

except Exception as e:

print(traceback.print\_exc())

# 分词 - 中文

def tokenize\_chinese(document):

try:

token\_list = jieba.cut( document, cut\_all=False )

#print("[INFO]: tokenize\_chinese is finished!")

return token\_list

except Exception as e:

print(traceback.print\_exc())

# 去除停用词 -英文

def filtered\_stopwords\_en(token\_list):

try:

token\_list\_without\_stopwords = [ word for word in token\_list if word not in stopwords.words("english")]

#print("[INFO]: filtered\_words is finished!")

return token\_list\_without\_stopwords

except Exception as e:

print(traceback.print\_exc())

# 去除停用词 - 中文

def filtered\_stopwords\_ch(token\_list,stopwords):

try:

token\_list\_without\_stopwords = [word for word in token\_list if word not in stopwords]

# print("[INFO]: filtered\_words is finished!")

return token\_list\_without\_stopwords

except Exception as e:

print(traceback.print\_exc())

# 去除标点

def filtered\_punctuations(token\_list):

try:

punctuations = ['', '\n', '\t', ',', '.', ':', ';', '?', '(', ')', '[', ']', '&', '!', '\*', '@', '#', '$', '%','xa0','，']

token\_list\_without\_punctuations = [word for word in token\_list if word not in punctuations]

#print("[INFO]: filtered\_punctuations is finished!")

return token\_list\_without\_punctuations

except Exception as e:

print(traceback.print\_exc())

# 过滤出中 - 英文

def filtered\_chinese\_english\_words(token\_list):

try:

r1 = re.compile(r'\w') # 使用正则表达式,筛选[A-Za-z0-9\_]

r2 = re.compile(r'[^\d]') # 使用正则表达式,筛选[0-9\_]

r4 = re.compile(r'[^\_]')

r3 = re.compile(r'[\u4e00-\u9fa5]')

token\_list = [word.lower() for word in token\_list if (r3.match(word) != None) or (r3.match(word) == None and r1.match(word) and r2.match(word) and r4.match(word))]

# print("[INFO]: filtered\_punctuations is finished!")

return token\_list

except Exception as e:

print(traceback.print\_exc())

# 词干化 -英文

def stemming( filterd\_token\_list ):

try:

st = LancasterStemmer()

stemming\_token\_list = [ st.stem(word) for word in filterd\_token\_list ]

#print("[INFO]: stemming is finished")

return stemming\_token\_list

except Exception as e:

print(traceback.print\_exc())

# 去除低频单词

def low\_frequence\_filter( token\_list ):

try:

word\_counter = defaultdict(int)

for word in token\_list:

word\_counter[word] += 1

threshold = 0

token\_list\_without\_low\_frequence = [ word

for word in token\_list

if word\_counter[word] > threshold]

#print "[INFO]: low\_frequence\_filter is finished!"

return token\_list\_without\_low\_frequence

except Exception as e:

print(traceback.print\_exc())

"""

功能：预处理

@ document: 文档

@ token\_list: 预处理之后文档对应的单词列表

"""

def pre\_process( document,ch\_stopwords ):

try:

# token\_list = tokenize(document)

token\_list = tokenize\_chinese(document)

token\_list=filtered\_chinese\_english\_words(token\_list )

token\_list = filtered\_stopwords\_ch(token\_list, ch\_stopwords)

token\_list= filtered\_punctuations(token\_list)

#print("[INFO]: pre\_process is finished!")

return token\_list

except Exception as e:

print(traceback.print\_exc())

"""

功能：预处理

@ document: 文档集合

@ token\_list: 预处理之后文档集合对应的单词列表

"""

def documents\_pre\_process( documents,ch\_stopwords ):

try:

documents\_token\_list = []

for document in documents:

token\_list = pre\_process(document,ch\_stopwords)

documents\_token\_list.append(token\_list)

print("[INFO]:documents\_pre\_process is finished!")

return documents\_token\_list

except Exception as e:

print(traceback.print\_exc())

#-----------------------------------------------------------------------

def test\_pre\_process(documents,ch\_stopwords):

# documents = ["he,he,he,we are happy!",

# "he,he,we are happy!",

# "you work!"]

documents\_token\_list = []

for document in documents:

token\_list = pre\_process(document,ch\_stopwords)

documents\_token\_list.append(token\_list)

for token\_list in documents\_token\_list:

print(token\_list)

# test\_pre\_process()

import pandas as pd

INPUT\_PATH = "/data/python\_pj6/bigdata"

ch\_stopkeyword=[line.strip() for line in open('/data/python\_pj6/stopword').readlines()] #加载停用词

data=pd.read\_csv(INPUT\_PATH)

test\_pre\_process(data.岗位描述,ch\_stopkeyword)

#-\*- coding:utf-8

'''

lda\_model.py

这个文件的作用是lda模型的训练

根据预处理的结果，训练lda模型

'''

from pre\_process import documents\_pre\_process

from gensim import corpora, models, similarities

import traceback

# 训练tf\_idf模型

def tf\_idf\_trainning(documents\_token\_list):

try:

# 将所有文章的token\_list映射为 vsm空间

dictionary = corpora.Dictionary(documents\_token\_list)

# 每篇document在vsm上的tf表示

corpus\_tf = [ dictionary.doc2bow(token\_list) for token\_list in documents\_token\_list ]

# 用corpus\_tf作为特征，训练tf\_idf\_model

tf\_idf\_model = models.TfidfModel(corpus\_tf)

# 每篇document在vsm上的tf-idf表示

corpus\_tfidf = tf\_idf\_model[corpus\_tf]

#print "[INFO]: tf\_idf\_trainning is finished!"

return dictionary, corpus\_tf, corpus\_tfidf

except Exception as e:

print(traceback.print\_exc())

# 训练lda模型

def lda\_trainning( dictionary, corpus\_tfidf, K ):

try:

# 用corpus\_tfidf作为特征，训练lda\_model

lda\_model = models.LdaModel( corpus\_tfidf, id2word=dictionary, num\_topics = K )

# 每篇document在K维空间上表示

corpus\_lda = lda\_model[corpus\_tfidf]

#print "[INFO]: lda\_trainning is finished!"

return lda\_model, corpus\_lda

except Exception as e:

print(traceback.print\_exc())

'''

功能:根据文档来训练一个lda模型，以及文档的lda表示

训练lda模型的用处是来了query之后，用lda模型将queru映射为query\_lda

@documents:原始文档raw material

@K:number of topics

@lda\_model:训练之后的lda\_model

@corpus\_lda:语料的lda表示

'''

def get\_lda\_model( documents, K , stopkeyword):

try:

# 文档预处理

documents\_token\_list = documents\_pre\_process( documents, stopkeyword)

# print(documents\_token\_list)

# 获取文档的字典vsm空间,文档vsm\_tf表示,文档vsm\_tfidf表示

dict, corpus\_tf, corpus\_tfidf = tf\_idf\_trainning( documents\_token\_list)

# print(corpus\_tfidf)

# 获取lda模型,以及文档vsm\_lda表示

lda\_model, corpus\_lda = lda\_trainning( dict, corpus\_tfidf, K )

print("[INFO]:get\_lda\_model is finished!")

return lda\_model, corpus\_lda, dict, corpus\_tf, corpus\_tfidf

except Exception as e:

print(traceback.print\_exc())

#-\*- coding:utf-8

'''

similarity.py

这个文件的作用是训练后的的lda模型，对语料进行相似度的计算

'''

from gensim import corpora, models, similarities

import traceback

'''

这个函数没有用到

'''

# 基于lda模型的相似度计算

def lda\_similarity( query\_token\_list, dictionary, corpus\_tf, lda\_model ):

try:

# 建立索引

index = similarities.MatrixSimilarity( lda\_model[corpus\_tf] )

# 在dictionary建立query的vsm\_tf表示

query\_bow = dictionary.doc2bow( query\_token\_list )

# 查询在K维空间的表示

query\_lda = lda\_model[query\_bow]

# 计算相似度

# simi保存的是 query\_lda和corpus\_lda的相似度

simi = index[query\_lda]

query\_simi\_list = [ item for \_, item in enumerate(simi) ]

return query\_simi\_list

except Exception as e:

print(traceback.print\_exc())

'''

功能：语聊基于lda模型的相似度计算

@ corpus\_tf:语聊的vsm\_tf表示

@ lda\_model:训练好的lda模型

'''

def lda\_similarity\_corpus( corpus\_tf, lda\_model ):

try:

# 语料库相似度矩阵

lda\_similarity\_matrix = []

# 建立索引

index = similarities.MatrixSimilarity( lda\_model[corpus\_tf] )

# 计算相似度

for query\_bow in corpus\_tf:

# K维空间表示

query\_lda = lda\_model[query\_bow]

# 计算相似度

simi = index[query\_lda]

# 保存

query\_simi\_list = [item for \_, item in enumerate(simi)]

lda\_similarity\_matrix.append(query\_simi\_list)

print("[INFO]:lda\_similarity\_corpus is finished!")

return lda\_similarity\_matrix

except Exception as e:

print(traceback.print\_exc())

#-\*- coding:utf-8

'''

save\_result.py

这个文件的作用是保存结果

'''

import traceback

def save\_similarity\_matrix(matrix, output\_path):

try:

outfile = open( output\_path, "w" )

for row\_list in matrix:

line = ""

for value in row\_list:

line += ( str(value) + ',' )

outfile.write(line + '\n')

outfile.close()

print("[INFO]:save\_similarity\_matrix is finished!")

except Exception as e:

print(traceback.print\_exc())

#-\*- coding:utf-8

'''

train\_lda\_main.py

这个文件的作用是汇总前面各部分代码，对文档进行基于lda的相似度计算

'''

from lda\_model import get\_lda\_model

from similarity import lda\_similarity\_corpus

from save\_result import save\_similarity\_matrix

import traceback

import pandas as pd

INPUT\_PATH = "/data/python\_pj6/bigdata"

OUTPUT\_PATH = "/data/python\_pj6/lda\_simi\_matrix.txt"

# data=pd.read\_csv(INPUT\_PATH)

# print(data.岗位描述)

# print(data.info())

def train(documents,stopword):

try:

# 语料

# documents = ["Shipment of gold damaged in a fire",

# "Delivery of silver arrived in a silver truck",

# "Shipment of gold arrived in a truck"]

# 训练lda模型

K = 2 # number of topics

lda\_model, \_, \_,corpus\_tf, \_ = get\_lda\_model(documents, 50,stopword )

# 计算语聊相似度

lda\_similarity\_matrix = lda\_similarity\_corpus( corpus\_tf, lda\_model )

# 保存结果

save\_similarity\_matrix( lda\_similarity\_matrix, OUTPUT\_PATH )

return lda\_similarity\_matrix

except Exception as e:

print(traceback.print\_exc())

def main(document):

INPUT\_PATH = "/data/python\_pj6/bigdata"

data = pd.read\_csv(INPUT\_PATH)

stopword = [line.strip() for line in open('/data/python\_pj6/china\_stopword').readlines()] # 加载停用词

frames=[document,data]

df = pd.concat(frames)

# print(df)

similiry=train(df.岗位描述, stopword)

for index,simi in enumerate(similiry[0]):

if simi>0.99:

print(data[index:index+1][['工作名称','公司名称','岗位描述']])

# print(index,'---',simi)

test\_data=pd.read\_csv('/data/python\_pj6/test\_data')

main(test\_data)