**前程无忧岗位数据可视化分析报告**

# 1 数据爬取

## 1.1爬取网站内容

以前程无忧 51job 求职网站（https: //www.51job.com/）为对象，选取了与信管专业相关的 30 个岗位（数据分析，产品经理，产品助理，交互设计，前端开发，软件设计，IOS 开发，业务分析，安卓开发，PHP 开发，业务咨询，需求分析，流程设计，售后经理，售前经理，技术支持，ERP 实施，实施工程师，IT 项目经理，IT 项目助理，信息咨询，数据挖掘，数据运营，网络营销，物流与供应链，渠道管理，电商运营，客户关系管理，新媒体运营，产品运营），通过 51job 官网首页的关键词搜索栏依次爬取岗位名称、公司名称、公司规模、工作地点、薪资、工作要求、工作待遇等相关信息内容。

相关的30个岗位都将爬取30页数据，每页50条，一共45000条数据（其中“IT项目助理”岗位仅有27页，缺少3页150条数据），将所有数据通过python的相应接口添加到Mysql数据库的jobs表中进行储存。

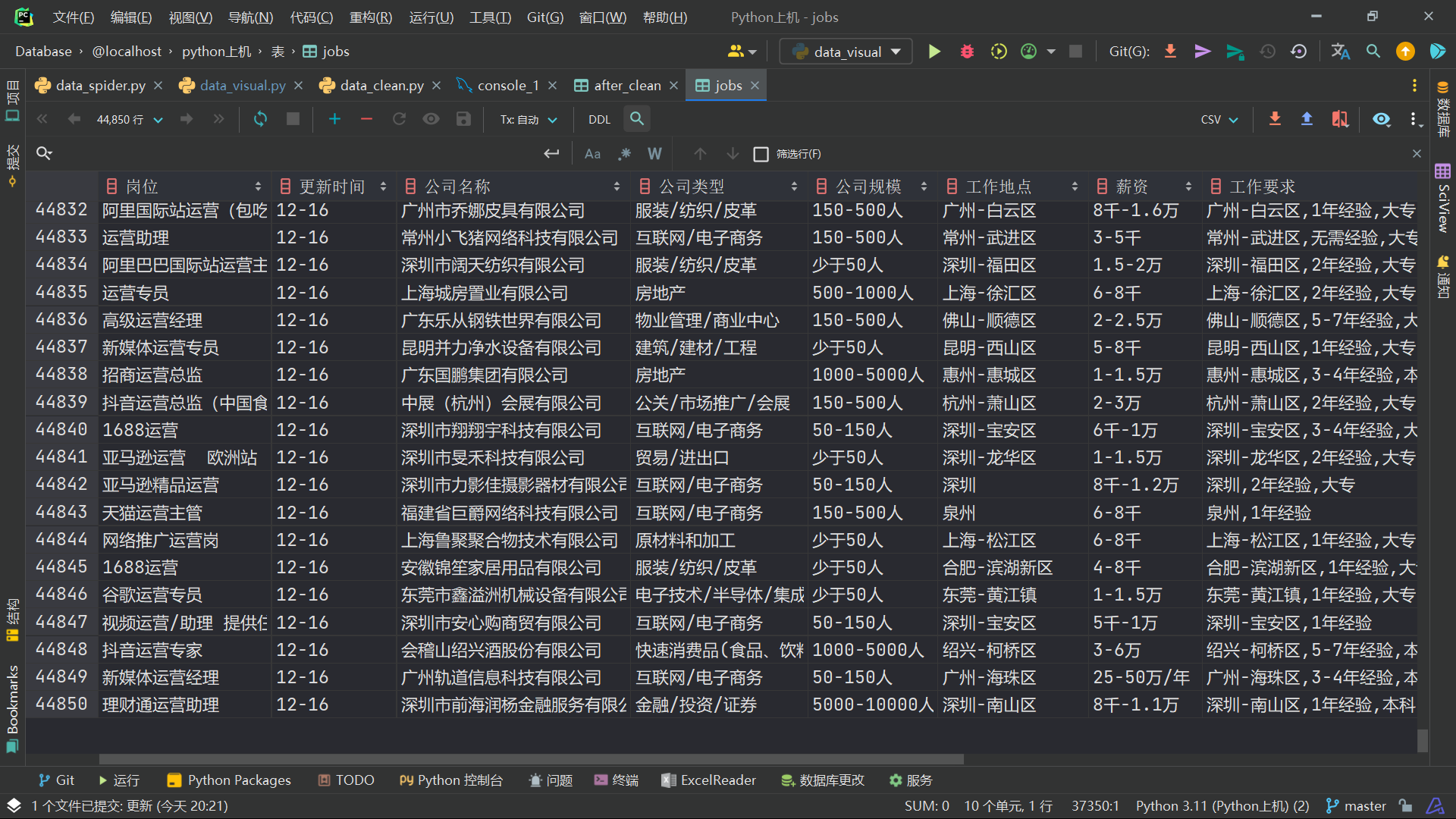


图1 共爬取44850条数据

## 1.2完整代码

data\_spider.py爬取数据文件完整代码

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

from bs4 import BeautifulSoup

import pymysql

import random

# 定义 jiexi()函数，用于解析得到的 html

def jiexi(html, info, name):

# print(html)

soup = BeautifulSoup(html, "html.parser")

text = soup.find\_all("script", type="text/javascript") # [2].string

text = str(text).split('window.\_\_SEARCH\_RESULT\_\_')[1]

text = text.split("</script>, ")[0]

# print('————————————————————————————————————————')

# print(text)

# 观察原始代码发现我们需要的数据在 engine\_jds 后

data = eval(text.split("=", 1)[1])["engine\_jds"]

for d in data:

try:

job\_name = d["job\_name"].replace("\\", "") # 岗位名称

except:

job\_name = " "

try:

company\_name = d["company\_name"].replace("\\", "") # 公司名称

except:

company\_name = " "

try:

providesalary\_text = d["providesalary\_text"].replace("\\", "") # 薪资

except:

providesalary\_text = " "

try:

workarea\_text = d["workarea\_text"].replace("\\", "") # 工作地点

except:

workarea\_text = " "

try:

updatedate = d["updatedate"].replace("\\", "") # 更新时间

except:

updatedate = " "

try:

jobwelf = d["jobwelf"].replace("\\", "") # 工作待遇

except:

jobwelf = " "

try:

companyind\_text = d["companyind\_text"].replace("\\", "")# 公司类型

except:

companyind\_text = " "

try:

companysize\_text = d["companysize\_text"].replace("\\","")#公司规模

except:

companysize\_text = " "

try:

at = d["attribute\_text"] # 工作要求

s = ''

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

s = s + at[i] + ','

attribute\_text = s[:-1]

except:

attribute\_text = " "

# 将每一条岗位数据爬取下的内容以及传入参数name作为一个列表，依此加入到info列表中

info.append([name, job\_name, updatedate, company\_name, companyind\_text, companysize\_text, workarea\_text,

providesalary\_text, attribute\_text, jobwelf])

# 将数据存到 MySQL 中名为“51job”的数据库中

def save(data):

db = pymysql.connect( # 连接数据库

host="127.0.0.1", # MySQL 服务器名

user="root", # 用户名

password="123456", # 密码

database="Python上机", # 操作的数据库名称

charset="utf8"

)

cursor = db.cursor()

# 将数据保存到数据库表中对应的列

for each\_data in data:

present\_job = each\_data[0] # 当前爬取岗位

job\_name = each\_data[1] # 岗位

updatedate = each\_data[2] # 更新时间

company\_name = each\_data[3] # 公司名称

companyind\_text = each\_data[4] # 公司类型

companysize\_text = each\_data[5] # 公司规模

workarea\_text = each\_data[6] # 工作地点

providesalary\_text = each\_data[7] # 薪资

attribute\_text = each\_data[8] # 工作要求

jobwelf = each\_data[9] # 工作待遇

# 创建 sql 语句

sql = "insert into jobs(当前爬取岗位, 岗位, 更新时间,公司名称,公司类型,公司规模,工作地点,薪资,工作要求,工作待遇) values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"

# 执行 sql 语句

cursor.execute(sql, [present\_job, job\_name, updatedate, company\_name, companyind\_text, companysize\_text, workarea\_text, providesalary\_text, attribute\_text, jobwelf])

db.commit() # 提交数据库

cursor.close() # 关闭游标

db.close() # 关闭数据库

if \_\_name\_\_ == '\_\_main\_\_': # 主函数

# searchword = input('请输入你想查询的岗位：') # 自行输入 30 个岗位名称搜索

# pages = int(input('请输入需要爬取的总页数：')) # 在观察后输入数据页面爬取数据

searchword = '产品运营' # 自行输入 30 个岗位名称搜索

pages = 30 # 在观察后输入数据页面爬取数据

mainurl = 'https://www.51job.com' # 设置 51jobs 首页的url

option = webdriver.ChromeOptions()

option.add\_experimental\_option('excludeSwitches', ['enable-automation'])

option.add\_experimental\_option('useAutomationExtension', False)

browser = webdriver.Chrome(options=option)

browser.execute\_cdp\_cmd("Page.addScriptToEvaluateOnNewDocument",

{"source":

"""

Object.defineProperty(navigator, 'webdriver', {

get: () => undefined

})

"""

})

browser.get(mainurl) # 打开 51jobs 首页

browser.implicitly\_wait(random.randint(6, 15))

browser.find\_element(by=By.XPATH, value='/html/body/div[3]/div/div[1]/div/div/p[1]/input').send\_keys(

searchword) # 在搜索框输入要查询的岗位

browser.implicitly\_wait(random.randint(6, 15))

button1 = browser.find\_element(by=By.XPATH, value='/html/body/div[3]/div/div[1]/div/button') # 寻找搜索按钮

button1.click() # 点击搜索按钮

time.sleep(random.randint(8, 15))

windows = browser.window\_handles

browser.switch\_to.window(windows[-1])

info = [] # 空列表，传入 jiexi 函数用于存储每一条岗位的数据

for i in range(1, pages + 1):

# html = browser.find\_element(by=By.XPATH, value='/html/body/div[2]/div[3]/div/div[2]/div[4]/div[1]').text

# browser.refresh()

html = browser.page\_source # 获取网页的 html

browser.implicitly\_wait(random.randint(8, 14))

jiexi(html, info, searchword)

if i <= 5:

button2 = browser.find\_element(By.XPATH,

value='/html/body/div[2]/div[3]/div/div[2]/div[4]/div[2]/div/div/div/ul/li[{}]/a'.format(i + 7))

else:

button2 = browser.find\_element(By.XPATH,

value='/html/body/div[2]/div[3]/div/div[2]/div[4]/div[2]/div/div/div/ul/li[13]/a') # 寻找下一页的按钮

button2.click() # 翻到下一页

time.sleep(10)

windows = browser.window\_handles

browser.switch\_to.window(windows[-1])

browser.get(browser.current\_url)

time.sleep(6)

save(info)

print('{}岗位的{}页数据已经爬取成功！请进入 Mysql 数据库检查！'.format(searchword, pages))

# 2 数据清洗

在本案例中所做的数据清洗工作主要包括：1）匹配各个岗位，2）清洗工作地点，3）平均化各个岗位的薪资并统一量纲，4）精简工作要求。

## 2.1匹配工作岗位

由于岗位爬取过程是以整页为单位进行的数据爬取，在爬取的最后一页岗位信息内容中会包含非关键词岗位搜索内容，为了确保爬取岗位信息的准确性，进行岗位匹配的数据清洗过程，针对不同岗位的清洗过程将存在一定差异。

具体的操作代码如下

# 引入 pymysql 包

import pymysql

# 连接 MySQL 数据库

db = pymysql.connect(

host="127.0.0.1",

user="root",

password="123456",

database="Python上机",

charset="utf8"

)

def pipei():

cursor = db.cursor() # 获取操作游标

cursor.execute("select \* from jobs") # 从 jobs 表中查询所有内容并保存

results = cursor.fetchall() # 接受全部的返回结果

after\_pipei = [] # 建立一个空列表，用来存储匹配后数据

for each\_result in results:

if each\_result[-1] == '物流与供应链':

if '物流' in each\_result[0] or '供应链' in each\_result[0]:

after\_pipei.append(each\_result)

elif each\_result[-1] == '新媒体运营' or each\_result[-1] == '电商运营':

if '运营' in each\_result[0]:

after\_pipei.append(each\_result)

# 由于在 以关键词“电商运营”或“新媒体运营”搜索的岗位信息中包含大量具体电商或新媒体平台名称的岗位名称，如“拼多多运营”“抖音运营”等，因此在这两类 岗位名称匹配时我们认为只要岗位名称中包含“运营”就算匹配成功。

elif each\_result[-1] == '客户关系管理':

if '客户关系' in each\_result[0]:

after\_pipei.append(each\_result)

elif each\_result[-1] == '安卓开发':

if '安卓' in each\_result[0] or 'Android' in each\_result[0]:

after\_pipei.append(each\_result)

# 由于在很多公司的招聘岗位中“安卓”会以“Android”英文形式出现，因此，在以“安卓开发”为关键词进行搜索时，我们认为只要包含“安卓”或“Android”开发 就算匹配成功。

elif each\_result[-1][:-2] in each\_result[0] and each\_result[-1][-2:] in each\_result[0]:

after\_pipei.append(each\_result)

# 剩余岗位需要两个关键词都存在岗位名称中，例如包含“数据”或“分析”在以“数据分析”

# 为关键词搜索的岗位名称种，我们就认为匹配成功。

cursor.close() # 关闭游标

return after\_pipei # 返回匹配后的列表

## 2.2 工作地点筛选

由于爬取到的工作地点信息格式不统一，有的地点只给出一级地名，有的给出了两级，我们将岗位对应的工作地点统一保留到给出的最高级别地名，例如“上海—普陀区”只保留“上海”。

def split\_city(data):

after\_split\_city = [] # 建立一个空列表，用来存储匹配后数据

for each\_date in data:

each\_date\_list = list(each\_date)

each\_date\_list[5] = each\_date\_list[5].split('-')[0] # 将数据表中工作地点列以'-'进行切割，选取第一个元素替换

# print(each\_date\_list)

after\_split\_city.append(each\_date\_list)

return after\_split\_city # 返回筛除后的数据

## 2.3 清洗薪资内容

由于爬取到的薪资信息大都是以区间数据形式出现，例如“5000-6000/月”，并且薪资单位不一致，有“千/月”、“万/年”等形式，因此对薪资数据的清洗主要包括两个方面：

1）对区间数值取组中值。如“5000-6000/月”处理为“5500/月”；

2）统一量纲。将薪资的单位统一为“千/月”，如“5500/月”处理为“5.5 千/月”，“24 万/年”处理为“20 千/月”。

def salary\_1(data): # 清除工作薪资内容的“·xx 薪”

after\_salary\_1 = []

for each\_data in data:

if '千' in each\_data[6] and '万' in each\_data[6]:

if '千' in each\_data[6]:

each\_data[6] = str((float(each\_data[6].split('千')[0]) / 10)) + each\_data[6].split('千')[1]

if '及以下' in each\_data[6]:

each\_data[6] ='30.0千/月'

if '薪' in each\_data[6]:

each\_data[6] = each\_data[6].split('·')[0]

after\_salary\_1.append(each\_data)

return after\_salary\_1

def salary(data):

after\_salary = [] # 建立一个空列表，用来存储匹配后数据

for each\_data in data:

if each\_data[6] != '': # 筛除缺失值，以小时 计费，给出的薪资表达为在“……以下”及“……以上”等难以计算数据的工作岗位#统一量纲（单位:千/月）

if each\_data[6][-1] == '年':

try:

each\_data[6] = str((float(each\_data[6].split('万')[0].split('-')[0]) + float(each\_data[6].split('万')[0].split('-')[1])) \* 5 / 12) + '千/月'

except:

each\_data[6] = each\_data[6]

elif each\_data[6][-1] == '天':

try:

each\_data[6] = str(float(each\_data[6].split('元')[0]) \* 30 / 1000) + '千/月'

except:

each\_data[6] = each\_data[6]

elif each\_data[6][-1] == '万':

try:

each\_data[6] = str((float(each\_data[6].split('万')[0].split('-')[0]) + float(each\_data[6].split('万')[0].split('-')[1])) \* 5) + '千/月'

except:

each\_data[6] = each\_data[6]

else:

try:

each\_data[6] = str((float(each\_data[6].split('千')[0].split('-')[0]) + float(each\_data[6].split('千')[0].split('-')[1])) /2) + '千/月'

except:

each\_data[6] = each\_data[6]

after\_salary.append(each\_data)

# print(after\_salary)

return after\_salary # 返回平均工资后的数据

## 2.4 清洗“工作要求”

由于爬取到的“工作要求”内容中一般包含“地点，经验，学历，招收人数”四项内容，在本次实验中只用到“经验”和“学历”两项内容，因此此步骤需要清洗掉其他两项内容。

def job\_attribute\_text(data):

for each\_data in data:

# 因为爬取到的工作要求内容中的情况多样，如“东莞-松山湖区,5-7 年经验,大专”等，需要做出多种判断。最后只保留“经验，学历”形式的数据内容，若经验或学历为空，保留形式为“，学历”或“经验，”方便后续选用。以“，”切割后的列表长度为 3，若包含了“经验”元素，则保留“经验，”形式内容

if len(each\_data[7].split(',')) == 3:

each\_data[7] = each\_data[7].split(',')[1] + ',' + each\_data[7].split(',')[2]

elif len(each\_data[7].split(',')) == 2:

each\_data[7] = each\_data[7].split(',')[1] + ','

else:

each\_data[7] = ''

# 返回筛除后的数据 #将清洗后的数据保存到数据库中 after\_clean表中，代码和保存爬取数据时类似

return data

## 2.5 保存数据和去除重复项

以上环节完成后，数据仍有一定的重复项，可能是爬取过程中重复刷新同一页面导致，故此需要去除重复数据项，保证数据的可靠性。在清洗完成后将数据导入after\_clean表中储存，共17582条可用数据。

def save(data):

cursor = db.cursor()

for each\_data in data:

job\_name = each\_data[0]

updatedate = each\_data[1]

company\_name = each\_data[2]

companyind\_text = each\_data[3]

companysize\_text = each\_data[4]

workarea\_text = each\_data[5]

providesalary\_text = each\_data[6]

attribute\_text = each\_data[7]

jobwelf = each\_data[8]

present\_job = each\_data[9]

sql = "insert into after\_clean(当前爬取岗位,岗位,更新时间,公司名称,公司类型,公司规模,工作地点,薪资,工作要求,工作待遇)values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"

cursor.execute(sql, [present\_job, job\_name, updatedate, company\_name, companyind\_text, companysize\_text, workarea\_text, providesalary\_text, attribute\_text, jobwelf])

db.commit()

cursor.close()

db.close()

# 定义主函数，用以执行以上四步数据清洗过程

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

data = pipei()

data1 = split\_city(data)

data2 = salary\_1(data1)

data3 = salary(data2)

data4 = job\_attribute\_text(data3)

data5 = list(set(tuple(i) for i in data4)) # 转化为元组然后利用集合去重

save(data5)

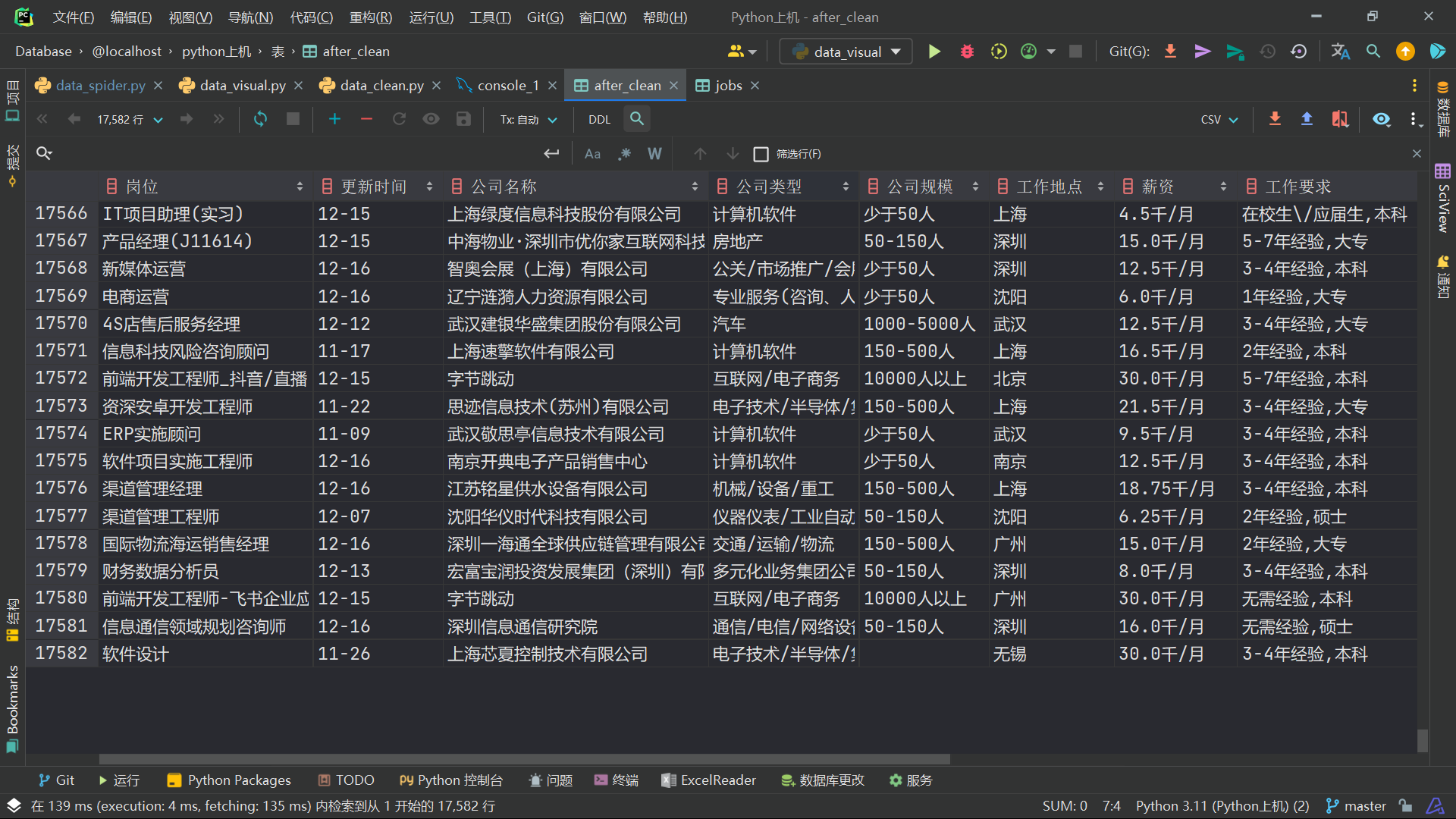


图2 清洗完成的17582条数据

# 3 数据可视化

数据可视化主要选取了柱状图、饼图、横状条形图、热力图、箱线图、桑基图、词云图和堆叠面积图进行描述性统计。

## 3.1 柱状图

**代码：**

# 描绘九小类岗位的市场需求的数量特征

def gangweishuliang\_hist(): # 主函数调用的 gangweishuliang\_hist()函数

cursor = db.cursor()

cursor.execute("select `当前爬取岗位` from after\_clean")

# 选出 30 个岗位关键词所在的列数据

results = cursor.fetchall()

jobs = ['产品经理', '产品助理', '交互设计', '前端开发', '软件设计', 'IOS开发', '业务分析', '安卓开发', 'PHP开发', '业务咨询', '需求分析', '流程设计', '售后经理', '售前经理', '技术支持', 'ERP实施', '实施工程师', 'IT项目经理', 'IT项目助理', '信息咨询', '数据挖掘', '数据运营', '数据分析', '网络营销', '物流与供应链', '渠道管理', '电商运营', '客户关系管理', '新媒体运营', '产品运营']

count = [] # 创建一个空列表，用于存储每种岗位的数量值

for i in range(len(jobs)):

count.append(0) # 先在空列表中创建 30 个 0 元素

for each\_result in results:

for i in range(0, 30):

if each\_result[0] == jobs[i]:

count[i] += 1

continue # 计算每种岗位的数量

jobs\_classification = ['技术管理类', 'IT运维类', '技术开发类', '业务咨询类', '技术支持类', '数据运营类', '市场职能类', '产品运营类', '数据管理类'] # 将 30 种岗位划分为九小类

counts = [] # 创建一个空列表，用于存储每小类岗位的数量值

for i in range(len(jobs\_classification)):

counts.append(0) # 先在空列表中创建 9 个 0 元素

# 根据大纲中给出的分类表，依据 30 种岗位数量值，分别计算出九小类岗位的数量值

counts[0] = count[0] + count[1] + count[2]

counts[1] = count[17] + count[18]

counts[2] = count[3] + count[4] + count[5] + count[6] + count[7] + count[8]

counts[3] = count[9] + count[10] + count[11]

counts[4] = count[12] + count[13] + count[14] + count[15] + count[16]

counts[5] = count[21] + count[22]

counts[6] = count[23] + count[24] + count[25]

counts[7] = count[26] + count[27] + count[28] + count[29]

counts[8] = count[19] + count[20]

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号

# 画柱状图

fig, ax = plt.subplots(figsize=(10, 7)) # 定义图片的大小

ax.bar(x=jobs\_classification, height=counts,

color='r')

# x是所有柱子的下标的列表 jobs\_classification，height 包含所有柱子的高度值的列表counts，width 每个柱子的宽度，本代码使用默认值。align 柱子对齐方式，有两个可选值：center 和 edge，本代码使用默认值“center”。color 为每根柱子呈现的颜色。

ax.set\_title("岗位数量柱状图", fontsize=15) # 为柱状图命名并设置字体大小

for x, y in enumerate(counts):

plt.text(x, y + 5, '%s' % y, ha='center') # 为每根柱子加上数值

plt.savefig(r'./岗位数量柱状图.png')

plt.show() # 展示图片

cursor.close() # 关闭操作游标

**绘制图形：**

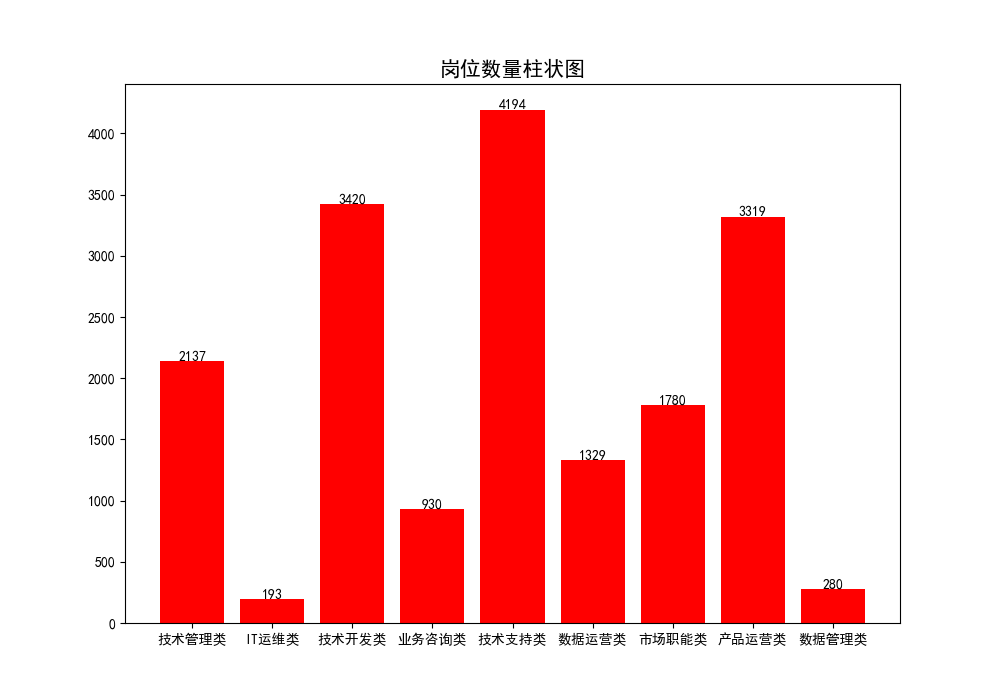


图3 岗位数量柱状图

**分析：**在清洗过后的17582条数据中，分别统计了九类岗位数据，因为每个岗位都爬取了1500条数据，岗位数量差异主要存在于数据清洗过程中被筛选掉的数据。

1. 从图中看出最多的三种岗位分别是技术支持类、技术开发类和产品运营类，一共10933条数据，占到了总数的近三分之二。这几类主要包括对专业技术和互联网方面的能力，能看出市场对于信管专业的技术和相应的互联网知识的需要。
2. 在IT运维和数据管理方面的岗位较少，这可能是因为企业对数据挖掘和数据管理岗位的定位在于少而精，导致需求量不多，但是要求高。

## 3.2 饼状图

**代码：**

# 描绘所有岗位学历要求饼图

def xueli\_pie(): # 主函数调用的 xueli\_pie()函数

cursor = db.cursor()

cursor.execute("select `工作要求` from after\_clean") # 选出工作要求的列数据

results = cursor.fetchall()

xueli = [] # 创建一个空列表，用来装工作要求种的“学历“数据

for each\_result in results:

if each\_result[0] != '' and each\_result[0].split(',')[1] != '':

xueli.append(each\_result[0].split(',')[1])

after\_quchong\_xueli = list(set(xueli)) # 对学历去重复值，由于元组不能更改，转换为列表类型，便于后续操作

after\_quchong\_xueli = ['在校生/应届生' if each == '在校生\\/应届生' else each for each in after\_quchong\_xueli]

counts = [] # 创建一个空列表，用于装所有岗位信息中每一种学历的数量值

for i in range(len(after\_quchong\_xueli)):

counts.append(0)

for each in xueli:

for i in range(len(after\_quchong\_xueli)):

if each == after\_quchong\_xueli[i]:

counts[i] += 1 # 计算每种学历的数量值

# 在画图前，先设置参数 explode，计算每个城市数据分析岗位占比，让占比小于 5%的城市在饼图中突出一部分

a = []

for i in range(0, len(after\_quchong\_xueli)):

a.append(0)

for i in range(0, len(after\_quchong\_xueli)):

total = sum(counts[:len(after\_quchong\_xueli)])

if (counts[i] / total) < 0.05:

a[i] = 0.2 # 让占比小于 5%的城市突出

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号

# x 为基本数据，labels 为给各个部分添加标签的列表，autopct 显示各部分比例，本例中调用%1.2f%%。

plt.pie(x=counts, explode=a, labels=after\_quchong\_xueli, autopct='%1.2f%%')

plt.title('学历要求饼状图') # 为饼图添加标题

plt.savefig(r'./学历要求饼状图.png')

plt.show() # 展示图片

cursor.close() # 关闭操作游标

**绘制图形：**

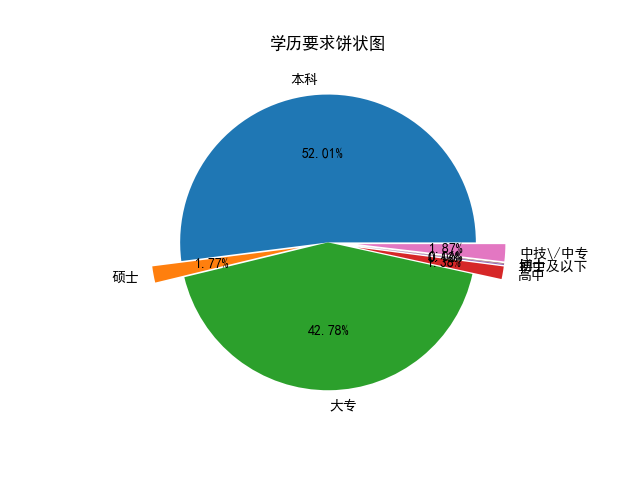


图4 学历要求饼状图

**分析：**从图中可以看出，市场对信管专业同学的学历还是有一定要求的。

1. 学历要求主要是大专生和本科生，这两类学历占比接近了 95%。即对于找工作来说，本科学历是一个明显的门槛，拥有本科学历后可以选择就业市场上接近 98%的工作，就业前景十分良好。
2. 大专以下学历的岗位数量只占 3%左右，这体现了市场对信管专业同学的要求是起码需要经过专业的训练，是一个有门槛的行业。这也符合信管专业既包括专业技术学习，也包括管理理论学习的专业特色。
3. 硕士与博士的行业需求只占 2%左右。这首先与硕博士的市场供给有关，即就业市场上求职的硕博士并不多，导致需求也不多。其次，这也说明就业市场上只需要硕博士的高要求岗位并不多，绝大部分岗位对学历要求没这么高。

综上，对于就业来说，信管专业同学只要达到本科学历就可以胜任绝大部分工作，继续深造对就业选择数量的增加不多。

## 3.3 条形图

**代码：**

# 描绘所有岗位公司规模的数量特征

def company\_size(): # 主函数调用的 company\_size()函数

cursor = db.cursor()

cursor.execute("SELECT `公司规模` FROM `after\_clean`") # 选出公司规模列

results = cursor.fetchall()

# 筛出非空数据，装进 size 列表中

size = []

for each\_result in results:

if each\_result[0] != '' and each\_result[0].split(',')[0] != '':

size.append(each\_result[0].split(',')[0])

# 将公司规模类型去重复值，得到例如['少于 50 人', '1000-5000 人', '5000-10000 人', '50-150 人',

# '150-500 人', '10000 人以上', '500-1000 人']的after\_quchong 列表

after\_quchong = list(set(size))

# 计算公司规模的类型数量 type\_size

type\_size = len(after\_quchong)

# 计算每种公司规模的岗位数

# 创建一个起始值为 0，有 type\_size 各元素的 count\_each\_size 列表

count\_each\_size = []

for i in range(type\_size):

count\_each\_size.append(0)

# 计算不同规模公司的岗位数量

for each\_size in size:

for each in range(0, type\_size):

if each\_size == after\_quchong[each]:

count\_each\_size[each] = count\_each\_size[each] + 1

# 创建一个公司规模类型和数量一一对应的字典

dic = {}

for i in range(0, type\_size):

dic[count\_each\_size[i]] = after\_quchong[i]

# 本例希望画出经过排序后的图形，由于 sort 排序会直接改变列表原来的元素，因此本例先新建一个与 count\_each\_size 相同的 order 列表，再进行排序

order = count\_each\_size

order.sort()

scale = []

for i in range(0, type\_size):

scale.append(dic.get(count\_each\_size[i])) # 通过字典，用新建的order 列表找到对应的公司规模类型，生成新的 scale 列表，此时两个列表中的元素是一一对应关系

# 画图

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号

plt.barh(scale, count\_each\_size) # 横放条形图函数 barh

plt.xlabel('数量') # x 轴名称

plt.ylabel('规模') # y 轴名称

plt.title('公司规模') # 图形名称

for y, x in enumerate(count\_each\_size):

plt.text(x + 0.1, y, "%s" % round(x, 1), va='center') # 为每个柱子加上对应的数值，其中 round(x,1)是将 x 值四舍五入到一个小数位

plt.savefig(r'./公司规模横向直方图.png')

plt.show() # 展示图形

cursor.close() # 关闭操作游标

**绘制图形：**

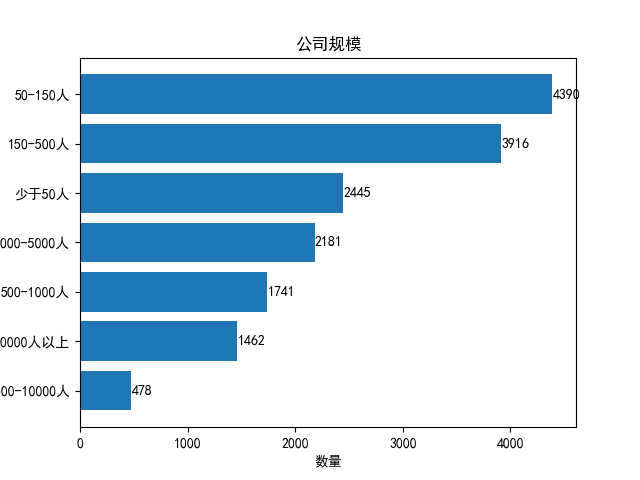


图5 公司规模横向条形图

**分析：**从整体上可以看出，该图是一个公司规模与岗位数量的一个右偏分布的图形，即信管专业的就业需求主要集中在公司规模小于 500 人的中小微企业。

值得注意的是，公司规模在 50人以下的企业提供的就业需求是第三多的，这说明大量的初创微型企业对信管专业同学有着需求。我们发现，实际上公司规模在 5000人以上的大厂对信管专业同学的需求只占总需求的 11%左右，这说明信管专业学生毕业后想进入大厂的难度很高，竞争十分激烈。

总的来说，中小微型企业占据了大多数信管专业人才的需求，以后择业可以根据自身的能力水平考虑去中小型企业还是大企业。

## 3.4 热力图

**代码：**

# 所有岗位在各个城市的数量分布热力图

def gangweishuliang\_heatmap(): # 主函数调用的 gangweishuliang\_heatmap()函数

cursor = db.cursor()

cursor.execute("SELECT `工作地点` FROM `after\_clean` ")

all = cursor.fetchall()

cities = [] # 创建一个空列表，用来装各个城市名称

for each in all:

if len(each[0]) == 2:

cities.append(each[0])

names = list(set(cities)) # 将城市名称去重复值，便于计算

# 获取各个城市的岗位数量值，装在 final 列表中

final = []

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

final.append(0)

for each in all:

for each\_name in range(0, len(names)):

if each[0] == names[each\_name]:

final[each\_name] += 1

break

# 此处需要将资料中给出的 city\_coordinates.json 文件放在 python 目录下，获取各个城市的经纬度信息

aa = [list(z) for z in zip(names, final)]

geo = (

Geo()

.add\_schema(maptype="china")

.add(

"岗位-城市数量分布热力图", # 图题

aa,

type\_=ChartType.HEATMAP, # 地图类型

)

.set\_series\_opts(label\_opts=opts.LabelOpts(is\_show=False))

# 设置是否显示标签

.set\_global\_opts(

visualmap\_opts=opts.VisualMapOpts(max\_=260),#设置 legend显示的最大值 )

)

geo.render("cities\_heatmap.html") # 以 html 类型保存，名称为cities\_heatmap

cursor.close() # 关闭操作游标

**绘制图形：**

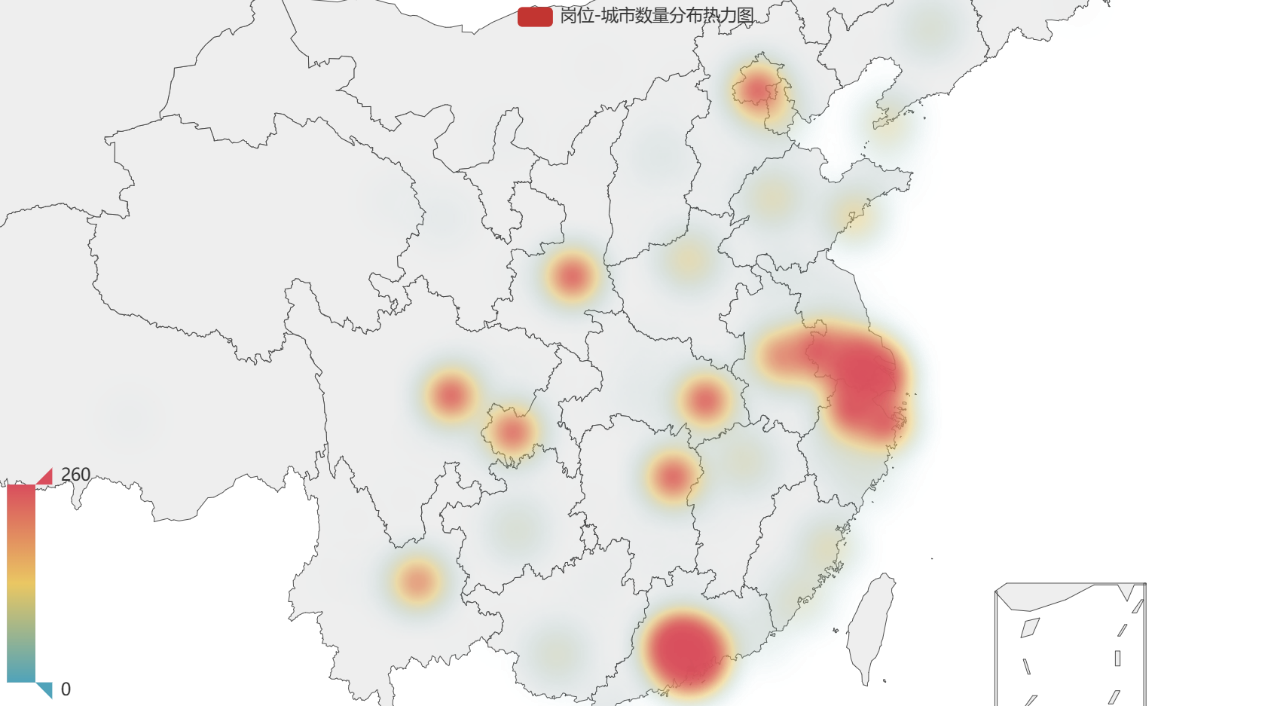


图6 岗位-城市数量分布热力图

**分析：**热力图是通过密度函数进行可视化用于表示地图中点的密度的热图，以特殊高亮的形式显示热度较强的地理区域。从图中可以看出，信管专业的市场需求主要分布在东南沿海以及除西北地区外的各个省会城市。

首先，以上海为中心的长三角地区以及广州、深圳等沿海城市的岗位数量明显多于其他各城市，这说明信管专业的岗位需求与经济发展水平有着明显的正向关系，同学们想要获得更好的职业发展机会应该优先考虑一线沿海城市。

其次，北京、武汉、长沙等内陆省会城市的岗位数量虽不及一线沿海城市，但对信管专业的同学也有一定的需求，若来自这些地方的同学想回家乡工作，各自的省会城市也能提供不错的发展机会。

最后，从图中可以看到西北城市提供的信管专业岗位数量很少，几乎没有，

这说明信管专业的就业还是有一定的地区聚集效应，来自这些地方的同学若想要

继续从事信管相关工作，还是需要考虑中东部发达城市。

## 3.5 箱线图

**代码：**

def salary\_xueli\_boxplot(): # 主函数调用的salary\_xueli\_boxplot()函数

cursor = db.cursor()

cursor.execute("SELECT `薪资`,`工作要求` FROM after\_clean")

all = cursor.fetchall()

# 创建两个空列表，用来存储筛选后的学历和薪资

xueli = []

salary = []

for each in all:

if each[1] != '' and each[1].split(',')[1] != '': # 筛除学历的缺失值以及部分异常值

xueli.append(each[1].split(',')[1])

# 筛除薪资的缺失值

salary.append(each[0].split('千')[0])

# print(each[1].split(',')[1],"+",each[0].split('千/月')[0])

# 对学历去重复值，并转换为列表形式，得到学历类型列表 xueli\_after\_quchong

xueli\_after\_quchong = list(set(xueli))

# 创建一个包含学历类型数个空列表的列表

final = []

for each in range(len(xueli\_after\_quchong)):

final.append([])

# 将不同类型学历的所有薪资先由字符串类型转换为浮点数类型，通过 index 索引，放进final 列表中对应的空列表，得到每种学历对应的薪资列表 final

for each in xueli\_after\_quchong:

for i in range(len(xueli)):

if xueli[i] == each:

final[xueli\_after\_quchong.index(each)].append(float(salary[i]))

# print(final)

# 画图

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号

plt.boxplot(final, labels=xueli\_after\_quchong)

plt.title('学历-薪资水平箱线图', fontsize=15)

plt.ylabel('薪资(单位：千/月)', fontsize=12)

plt.savefig(r'./学历-薪资水平箱线图.png')

plt.show() # 展示图形

cursor.close() # 关闭操作游标

**绘制图形：**

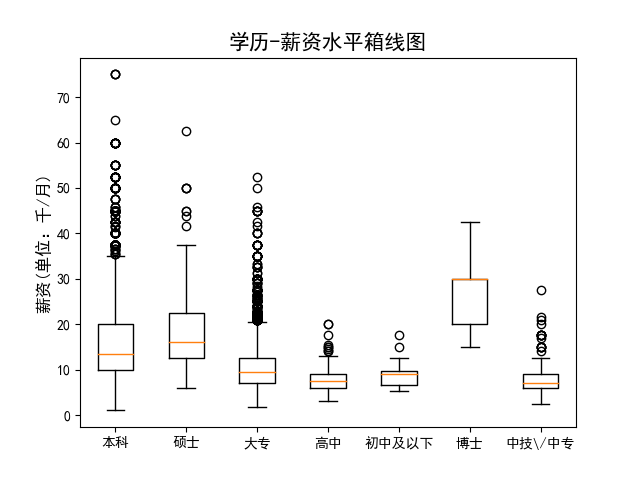


图7 学历-薪资水平箱线图

**分析：**箱线图，是利用数据中的五个统计量：上边缘、下边缘、中位数和两个四分位数来描述数据的一种方法，它也可以粗略地看出数据是否具有对称性，分布的分散程度等信息，特别可以用于对几个样本的比较。从图中可以看出，市场对于不同学历的求职者给与的薪资水平是存在差异的。

首先，随着学历水平的提高，薪资水平也有着相应的提高，从中位数来看，博士生的薪资是要明显高于其他学历求职者的。

其次，对于学历要求为本科和硕士的岗位，其薪资水平十分接近，这说明可能对于信管专业学生来说，本科学历已经足以胜任大部分工作。

再者，可以发现学历要求为大专的岗位异常值最多，这说明该学历的薪资分歧严重，这可能是以为某些小企业对于数据分析人才的急缺，导致它们在降低学历要求的同时给出了更高的薪资待遇。

最后，由于学历要求为高中、中技、中专和初中及以下的岗位数较少，因此其各数据几乎重叠且数值较低，不具代表性。

总的来说，除去一些极端情况，学历和薪资还是存在一定正相关关系的。

## 3.6 Sankey图

**代码：**

def post\_salary(): # 主函数调用的 post\_salary()函数

cursor = db.cursor()

# 从数据库表中选择这 7 种职务的薪资和名称

cursor.execute("SELECT `当前爬取岗位`,`薪资` FROM `after\_clean` ")

results = cursor.fetchall()

# 根据实验课程大纲中的表格分类信息，将 30 种岗位分为九小类

post = ['技术管理类', '技术开发类', '业务咨询类', '技术支持类', 'IT运维类', '数据管理类', '数据运营类', '市场职能类', '产品运营类']

T1 = ['产品经理', '产品助理', '交互设计']

T2 = ['前端开发', '软件设计', 'IOS开发', '业务分析', '安卓开发', 'PHP开发']

C1 = ['业务咨询', '需求分析', '流程设计']

C2 = ['售后经理', '售前经理', '技术支持', 'ERP实施', '实施工程师']

C3 = ['IT项目经理', 'IT项目助理']

D1 = ['信息咨询', '数据挖掘']

D2 = ['数据运营', '数据分析']

P1 = ['网络营销', '物流与供应链', '渠道管理']

P2 = ['电商运营', '客户关系管理', '新媒体运营', '产品运营']

clasify = [T1, T2, C1, C2, C3, D1, D2, P1, P2]

post1 = []

for each\_post in post:

for i in range(7):

post1.append(each\_post) # 将九小类岗位重复 7 次，放进 post1 列表中，用于后续计算数值和画图

# 薪资水平划分

salary = ['5千/月以下', '5-10千/月', '10-15千/月', '15-20千/月', '20-25千/月', '25-30 千/月', '30 千/月以上']

# 为了能对应九小类职务，将每种水平薪资取 7 次

salary1 = salary \* 9

# 构建一个包含 7\*9 个元素的列表，初始值为 0，用于存储职务与薪资水平的一对一岗位数量

count = []

for i in range(7 \* 9):

count.append(0)

# 计算 count 列表种每个元素的值，即职务与薪资水平的一对一岗位数量

for each\_result in results:

for i in range(9):

if each\_result[0] in clasify[i] and each\_result[1] != '':

if float(each\_result[1].split('千')[0]) < 5:

count[i \* 7] += 1

elif 5 <= float(each\_result[1].split('千')[0]) < 10:

count[i \* 7 + 1] += 1

elif 10 <= float(each\_result[1].split('千')[0]) < 15:

count[i \* 7 + 2] += 1

elif 15 <= float(each\_result[1].split('千')[0]) < 20:

count[i \* 7 + 3] += 1

elif 20 <= float(each\_result[1].split('千')[0]) < 25:

count[i \* 7 + 4] += 1

elif 25 <= float(each\_result[1].split('千')[0]) < 30:

count[i \* 7 + 5] += 1

elif 30 <= float(each\_result[1].split('千')[0]):

count[i \* 7 + 6] += 1

# 整理数据

df = pd.DataFrame({

'职位': post1,

'薪资': salary1,

'数量': count

})

# 把所有涉及到的节点去重规整到一起，即把“职位”列的'数据分析','产品经理', '产品助理', '交互设计', '前端开发', '软件设计', 'IOS 开发'和“薪资”

# 列中的'5 千/月以下','5-10 千/月','10-15 千/月','15-20 千/月','20-25 千/月','25-30 千/月','30 千/月以上'以列表内嵌套字典的形式去重汇总

nodes = []

for i in range(2):

values = df.iloc[:, i].unique()

for value in values:

dic = {}

dic['name'] = value

nodes.append(dic)

# 定义边和流量，用 Source-target-value 字典格式，能清晰描述数据的流转情况

linkes = []

for i in df.values:

dic = {}

dic['source'] = i[0]

dic['target'] = i[1]

dic['value'] = i[2]

linkes.append(dic)

# 画图

pic = (

Sankey().add(

'职位\_薪资桑基图', # 图例名称

nodes, # 传入节点数据

linkes, # 传入边和流量数据

# 设置透明度、弯曲度

linestyle\_opt=opts.LineStyleOpts(opacity=0.3, curve=0.5),

# 标签显示位置

label\_opts=opts.LabelOpts(position='right'),

# 节点之间的距离

node\_gap=30,

)

.set\_global\_opts(title\_opts=opts.TitleOpts(title='职位\_薪资桑基图'))

)

pic.render('Sankey.html') # 默认保存在本代码同文件夹下

**绘制图形：**

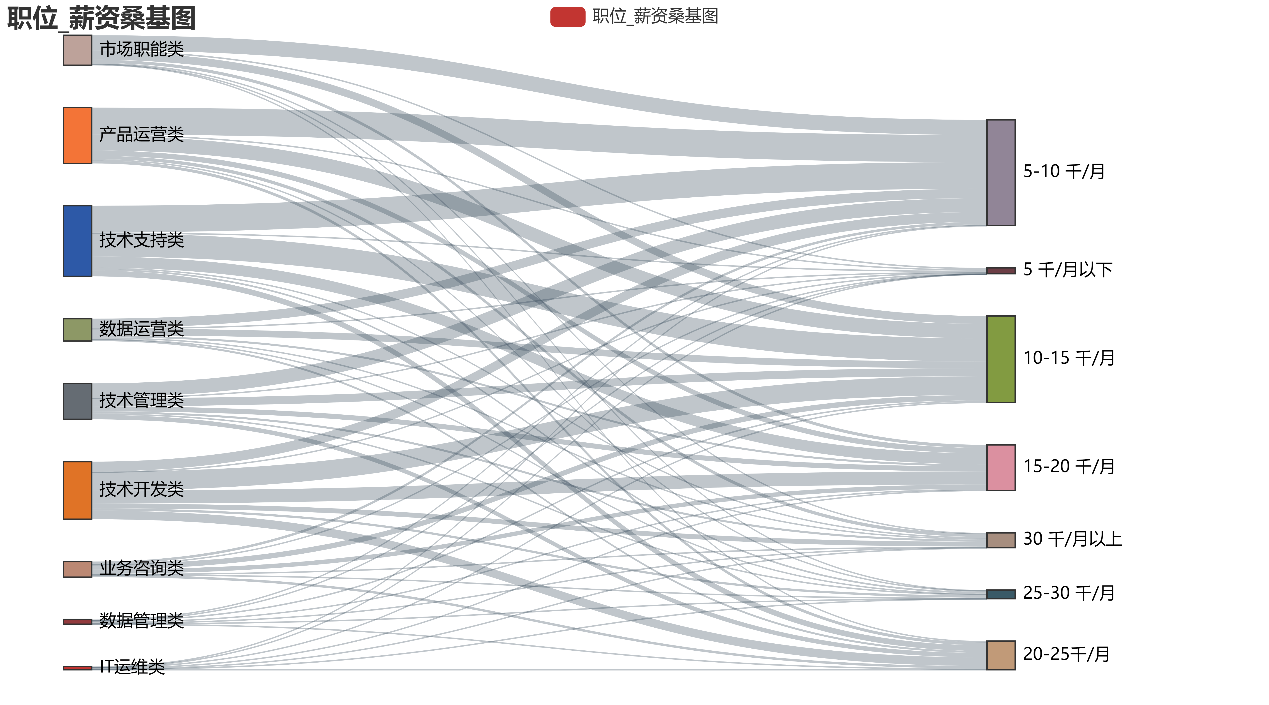


图8 职位-薪资桑基图

**分析：**桑基图，也叫桑基能量分流图或者桑基能量平衡图。它是一种特定类型的流程图，图中延伸的分支的宽度对应数据流量的大小，所有主支宽度的总和应与所有分出去的分支宽度的总和相等，保持能量的平衡。

从 Sankey 图中可以很清晰的看出这九小类岗位类型大致的薪酬水平大致的分布情况和主要的分布区间。

首先，可以清晰地看出信管专业的总体薪资水平主要集中在 5-10k/月、10-15k/月和 15-20k/月这三个区间，每月25k以上的岗位数量非常少。

其次，市场需求较大的产品运营类、技术支持类、技术开发类和技术管理类岗位的薪资主要集中在5-10k/月和10-15k/月两个区间水平，并且这几类岗位中还有不少岗位的薪资能达到每月15-20k/月的薪资水平，总体薪资还是很不错的。

从这个图可以很直观地看出自己想从事的岗位类型的薪酬水平分布，并可以分析各种岗位类型的薪酬水平结构是比较稳定的。无论从事何种岗位工作，其薪酬水平分布都不会过于极端。

## 3.7 词云图

**代码：**

# 用来描绘所有岗位的工作待遇热词共现

def wordcloud\_welfare(): # 主函数调用的 wordcloud\_welfare()函数

cursor = db.cursor()

cursor.execute("SELECT `工作待遇` FROM `after\_clean`")

results = cursor.fetchall()

txt = ''

for each\_result in results:

txt = txt + each\_result[0]

# 统计词频的字典

word\_freq = dict()

# 装载停用词,此处需将资料中给出的 hit\_stopwords.txt 文件放到本代码所在路径下

with open("D:/Python\_Program/python数据爬虫及可视化实验/实验所需输入文件/stopwords.txt", "r", encoding='utf-8') as f1: # 读取我们的待处理本文

txt1 = f1.readlines()

stoplist = []

for line in txt1:

stoplist.append(line.strip('\n'))

# 切分、停用词过滤、统计词频

for w in list(jieba.cut(txt)):

if len(w) > 1 and w not in stoplist:

if w not in word\_freq:

word\_freq[w] = 1

else:

word\_freq[w] = word\_freq[w] + 1

# 指定背景模式图片

back\_color = imread("D:/Python\_Program/python数据爬虫及可视化实验/实验所需输入文件/地大.png")

# 构造 WordCloud 对象

wc = WordCloud(background\_color='white', max\_words=100, collocations=False, width=1000, height=1000,

font\_path='simhei.ttf', mask=back\_color)

# 调用方法生成词云图

wc = wc.generate\_from\_frequencies(word\_freq)

# 保存图片

wc.to\_file('WordCloud.png')

plt.imshow(wc)

plt.axis("off")

plt.savefig(r'./工作待遇词云图.png')

plt.show()

**绘制图形：**



图9 工作待遇词云图

**分析：**词云图是由词汇、颜色、字体大小和图形四个要素构成的，它浓缩了文本数据的内容，通过文字、色彩、图形的搭配，产生了有冲击力地视觉效果。词云图直观的表示了每个词汇在相应文本数据中的词频分布，通过使用不同的颜色和大小来表示不同级别的相对重要性，字体越大越显眼，对应的词汇被提及频率越高。词云图过滤掉了大量的文本信息，使阅读者只要一眼扫过词云图就可以大致领略到文本所表达的主旨。

从图中可以非常清晰直观地看出岗位提供的各种福利待遇和一些基本的待遇类型。首先，字体最大的是绩效奖金和年终奖金，说明这些待遇是跟实际工资挂钩的，关系着这份工作能带来的收入，相当多的公司通过这些来吸引求职。

其次，还包括专业培训、旅游、餐饮、体检、医疗保险、周末和全勤奖等的字体也较大，大多数的公司也会给予员工这些福利待遇，可以较基本待遇更吸引求职者。

还有一些待遇较好的公司还会提供住宿、包吃、出国机会、班车、带薪休假等等福利，但这种待遇在图中的字体较小，说明这并不是普遍存在于各个公司，要视公司而定。这些福利待遇将更多地吸引要求较高福利待遇的求职者并起到积极的激励作用。

总体而言，公司一般都会提供补贴、绩效奖和年终奖这些基本福利待遇，较好的公司还会提供更多更吸引人的福利，同学们可以根据自己的实际情况，在求职时对公司有更多的了解和要求

# 4 其他分析

**简介：**堆叠面积图（stacked area chart）可以用来比较在一个区域内的多个变量，适合展示整体数据的变化趋势。堆叠面积图和普通的面积图的区别是每个数据值序列映射的区域起点都是上一个数据值序列顶端。如果要绘制的数据有多个序列，并想要分析整体变化的趋势，使用堆叠面积图会是一个不错的选择。

**代码：**

# 画岗位数量与薪资之间的堆叠面积图

def post\_salary\_stacked\_area():

cursor = db.cursor()

# 从数据库表中选择职务的薪资和名称

cursor.execute("SELECT `当前爬取岗位`,`薪资` FROM `after\_clean` ")

results = cursor.fetchall()

# 选取的六个岗位作图

T1 = '产品经理'

C1 = '需求分析'

C2 = 'ERP实施'

C3 = 'IT项目经理'

D2 = '数据分析'

P1 = '网络营销'

clasify = [T1, C1, C2, C3, D2, P1]

# 薪资水平划分

salary = ['5千/月以下', '5-10千/月', '10-15千/月', '15-20千/月', '20-25千/月', '25-30千/月', '30 千/月以上']

# 构建一个包含 6 个元素的列表，初始值为 0，用于存储岗位与薪资水平的一对一岗位数量

count = [[], [], [], [], [], []]

for x in range(6):

for y in range(7):

count[x].append(0)

# 计算 count 列表种每个元素的值，即职务与薪资水平的一对一岗位数量

for each\_result in results:

for i in range(6):

if each\_result[0] == clasify[i] and each\_result[1] != '':

s = float(each\_result[1].split('千')[0])

if s < 5:

count[i][0] += 1

elif 5 <= s < 10:

count[i][1] += 1

elif 10 <= s < 15:

count[i][2] += 1

elif 15 <= s < 20:

count[i][3] += 1

elif 20 <= s < 25:

count[i][4] += 1

elif 25 <= s < 30:

count[i][5] += 1

elif 30 <= s:

count[i][6] += 1

# 计算每个薪资区间的岗位总数

all\_count = [0, 0, 0, 0, 0, 0, 0]

for j in range(6):

for i in range(7):

all\_count[i] += count[j][i]

# 画图

pic = (

Line()

.add\_xaxis(salary)

.add\_yaxis(T1, count[0], color='#5470c6',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.add\_yaxis(C1, count[1], color='#91cc75',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.add\_yaxis(C2, count[2], color='#fac858',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.add\_yaxis(C3, count[3], color='#ee6666',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.add\_yaxis(D2, count[4], color='#73c0de',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.add\_yaxis(P1, count[5], color='#3ba272',

stack='stack', symbol='circle', is\_smooth=True,

areastyle\_opts=opts.AreaStyleOpts(opacity=0.4)

)

.set\_global\_opts(title\_opts=opts.TitleOpts(title='职位\_薪资堆叠面积图'),

xaxis\_opts=opts.AxisOpts(axislabel\_opts=opts.LabelOpts(rotate=-30)), # 设置x轴标签旋转角度

yaxis\_opts=opts.AxisOpts(name='岗位数量'),

)

.set\_series\_opts(label\_opts=opts.LabelOpts(font\_size=8,position='inside'))

)

pic.render('Stacked\_Area.html') # 默认保存在本代码同文件夹下

**绘制图形：**

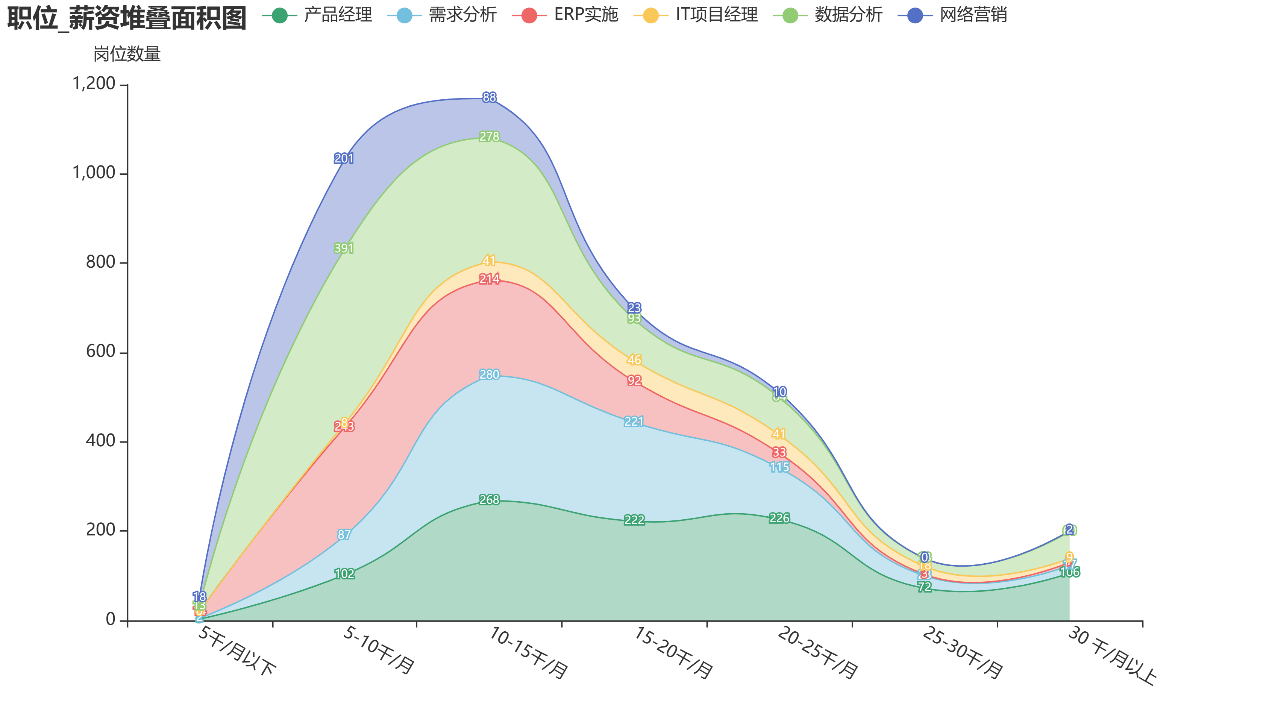


图10 职位-薪资堆叠面积图

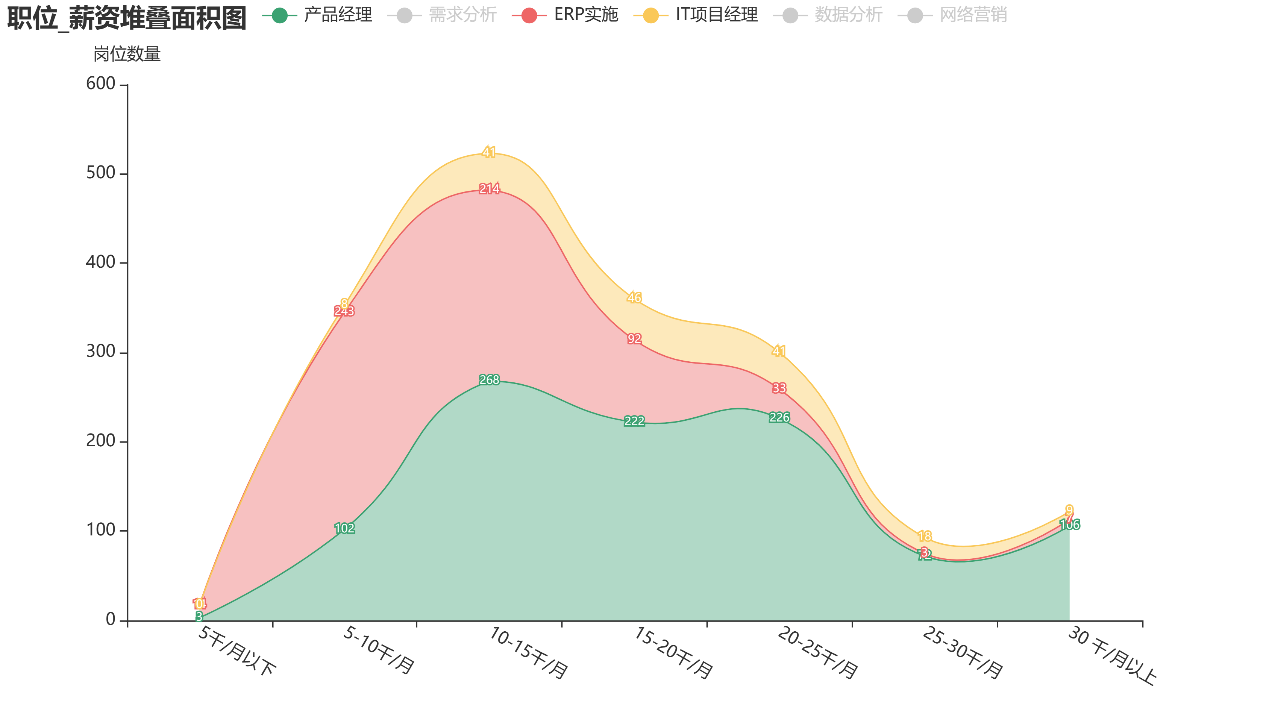


图11 “产品经理”“ERP实施”“IT项目经理”堆叠面积图

**分析：** 职位-薪资堆叠面积图展现了不同职位随着薪资变化，清晰反映了不同岗位在薪资方面的变化趋势，而且能在不同岗位之间进行方便对比。

首先，在图10中可以看出这六个岗位薪资主要集中在5-10千/月和10-15千/月的区间内，而5千/月以下、25-30千/月和30千/月的岗位数量相对较少。

其次，IT项目经理的岗位相当少，且薪资集中在10-25千/月这个区间，说明这个职位级别较高，需求少，薪资高。在ERP实施岗位和产品经理中，岗位数量很多，市场需求旺盛，其中ERP实施岗位薪资集中在5-15千/月，而产品经理岗位薪资则在5-30千以上的分布很广，30千以上数量最多，说明这个职位薪资对个人能力要求很高，做的好的话薪资相当可观。

这个堆叠面积图可以很直观的看出岗位数量随薪资的变化情况，同时在不同岗位之间进行对比，方便我们选择自己理想的岗位和薪资。也反映了薪资的趋势很大程度取决于所处岗位，岗位的选择决定着薪资的上下限。