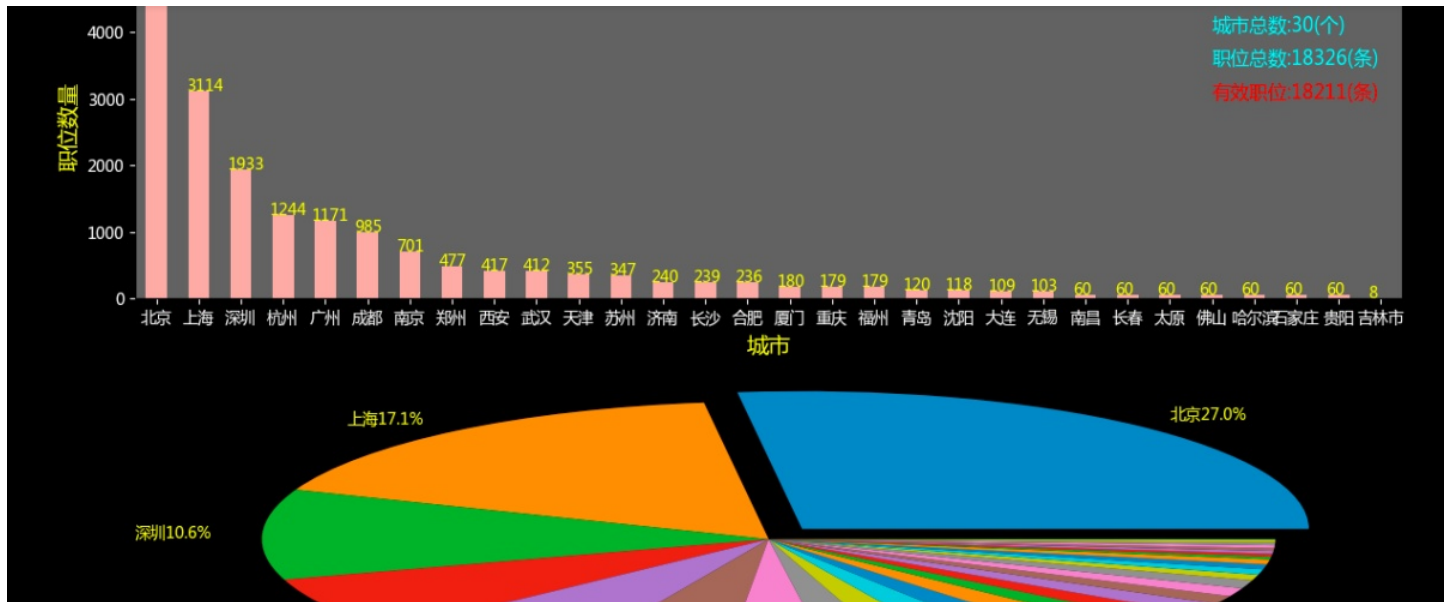


知乎



首发于

Python数据从入门到实践



智联Python相关职位的数据分析及可视化-Pandas&Matplotlib篇



Lyon

Keep balance, Be a better man!

48 人赞了该文章

上一篇，我用了Excel对爬虫采集到的智联招聘数据进行了数据分析及可视化，用到软件是Excel，这一篇，我们打算完全用Python来做同样的事。用到的库有Pandas、Matplotlib。np、pd、plt分别是numpy、pandas、matplotlib.pyplot的常用缩写。

Numpy (Numerical Python的简称) 是Python科学计算的基础包。它提供了以下功能：

1. 快速高效的多维数组对象ndarray。
2. 用于对数组执行元素级计算以及直接对数组执行数学运算的函数。
3. 用于读写硬盘上基于数组的数据集的工具。
4. 线性代数运算、傅里叶变换，以及随机数生成。
5. 用于将C、C++、Fortran代码集成到Python的工具。

除了为Python提供快速的数组处理能力，Numpy在数据分析方面还有另外一个主要作用，即作为在算法之间传递数据的容器。对于数值型数据，Numpy数组在存储和处理数据时要比内置的Python数据结构高效的多。此外，由低级语言（比如C和Fortran）编写的库可以直接操作Numpy数组中的数据，无需进行任何数据复制工作。

Pandas这个名字本身源于panel data（面板数据，这是计量经济学中关于多维结构化数据集的一个术语）以及Python data analysis. par

▲ 赞同 48 ▼

● 21 条评论

➤ 分享

★ 收藏

量数据结构和函数。Pandas中用的最多的是DataFrame，它是一个面向列的二维表结构，且行标和列标。pandas兼具numpy高性能的数组计算功能以及电子表格和关系型数据库(如SQL)灵活的数据处理功能。它提供了复杂精细的索引功能，以便更为便捷地完成重塑、切片和切块、聚合以及选取数据子集等操作。

Matplotlib是Python中常用的可视化绘图库，可以通过简单的几行代码生成直方图，功率谱，条形图，错误图，散点图等。Seaborn、ggplot、等诸多Python可视化库均是在此基础上开发的，所以学会matplotlib的基础操作还是很有必要的！它和Ipython结合的很好，提供了一种非常好用的交互式数据绘图环境。绘制的图表也是交互式的，你可以利用绘图窗口中的工具栏放大图表中的某个区域或对整个图表进行平移浏览。

数据来源：

Python爬虫爬取了智联招聘关键词：【Python】、全国30个主要城市的搜索结果，总职位条数：18326条(行)，其中包括【职位月薪】、【公司链接】、【工作地点】、【岗位职责描述】等14个字段列，和一个索引列【ZL_Job_id】共计15列。数据存储在本地MySQL服务器上，从服务器上导出json格式的文件，再用Python进行数据读取分析和可视化。

数据简单清洗：

1.首先在终端中打开输入ipython --pylab。在Ipython的shell界面里导入常用的包numpy、pandas、matplotlib.pyplot。用pandas的read_json()方法读取json文件，并转化为用df命名的DataFrame格式文件。（DataFrame格式是Pandas中非常常用且重要的一种数据存储格式、类似于Mysql和Excel中的表。）

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_json('/Users/zhaoluyang/Desktop/Python_全国JSON.json')

#查看df的信息
df.info()
df.columns
```

赞同 48

21 条评论

分享

收藏

```

Lucifers-MacBook-Air:~ zhaoluyang$ ipython --pylab
Python 3.6.0 (default, Jan 24 2017, 16:44:16)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.
Using matplotlib backend: MacOSX

[In [1]: import numpy as np

[In [2]: import pandas as pd

[In [3]: import matplotlib.pyplot as plt

In [4]: df = pd.read_json('/Users/zhaoluyang/Desktop/智联招聘-作图/Python_全国JS
...: ON.json')

[In [5]: df.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18326 entries, 0 to 18325
Data columns (total 15 columns):
ZL_Job_id      18326 non-null int64
公司名称      18326 non-null object
公司链接      18326 non-null object
发布日期      18326 non-null object
岗位职责描述   18326 non-null object
工作地点      18326 non-null object
工作性质      18326 non-null object
工作经验      18326 non-null object
招聘人数      18326 non-null object
最低学历      18326 non-null object
福利标签      18326 non-null object
职位名称      18326 non-null object
职位月薪      18326 non-null object
职位类别      18326 non-null object
职位链接      18326 non-null object
dtypes: int64(1), object(14)
memory usage: 2.2+ MB

[In [6]: df.columns
Out[6]:
Index(['ZL_Job_id', '公司名称', '公司链接', '发布日期', '岗位职责描述', '工作地点', '工作性质',
'工作经验',
'招聘人数', '最低学历', '福利标签', '职位名称', '职位月薪', '职位类别', '职位链接'],
      dtype='object')

```

可以看到读取的df格式文件共有15列，18326行，pandas默认分配了索引值从0~18325。还有一点值得注意的：全部的15列都有18326个非空值，因为当初写爬虫代码时设置了，如果是空值，譬如：有一条招聘信息其中【福利标签】空着没写，那么就用字符串代替，如“found no element”。

2.读取JSON文件时pandas默认分配了从0开始的索引，由于文件'ZL_Job_id'列中自带索引，故将其替换!替换后，用sort_index()给索引重新排列。

```

df.index = df['ZL_Job_id']#索引列用'ZL_Job_id'列替换。
del(df['ZL_Job_id'])#删除原文件中'ZL_Job_id'列。
df_sort = df.sort_index()#给索引列重新排序。
df = df_sort
df[['工作地点', '职位月薪']].head(10)

```

3.下面，将进行【职位月薪】列的分列操作，新增三列【bottom】、【top】、【average】分别存放最低月薪、最高月薪和平均月薪。其

8000-10000元/月，为此需要对【职位月薪】

赞同 48

21 条评论

分享

收藏

处理后bottom = 8000,top = 10000,average = 9000. 其中不同语句用于处理不同的情况, 譬如【职位月薪】= '面议'、'found no element'等。对于字符形式的'面议'、'found no element'处理后保持原字符不变, 即bottom = top = average = 职位月薪。
q1,q2,q3,q4用来统计各个语句执行次数.其中q1统计职位月薪形如'6000-8000元/月'的次数; q2统计形如月收入'10000元/月以下'; q3代表其他情况如'found no element', '面议'的次数; q4统计失败的特殊情况。

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

for i in range(len(df['职位月薪'])):
    item = df['职位月薪'].iloc[i].strip()
    result = re.findall(pattern,item)
    try:
        if result:
            try:
                #此语句执行成功则表示result[0],result[1]都存在, 即职位月薪形如'6000-8000元/月'
                df['bottom'].iloc[i],df['top'].iloc[i] = result[0],result[1]
                df['average'].iloc[i] = str((int(result[0])+int(result[1]))/2)
                q1+=1

            except:
                #此语句执行成功则表示result[0]存在, result[1]不存在, 职位月薪形如'10000元/月以下'
                df['bottom'].iloc[i] = df['top'].iloc[i] = result[0]
                df['average'].iloc[i] = str((int(result[0])+int(result[0]))/2)
                q2+=1

        else:
            #此语句执行成功则表示【职位月薪】中并无数字形式存在, 可能是'面议'、'found no element'
            df['bottom'].iloc[i] = df['top'].iloc[i] = df['average'].iloc[i] = item
            q3+=1

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

for i in range(100):#测试一下看看职位月薪和bottom、top是否对的上号
    print(df.iloc[i][['职位月薪','bottom','top','average']])#或者df[['职位月薪','bottom'

df[['职位月薪','bottom','top','average']].head(10)
```

赞同 48

21 条评论

分享

收藏


```

职位月薪      8001-10000元 /月
bottom        8001
top           10000
average       9000.5
Name: 163, dtype: object
职位月薪      6000-12000元 /月
bottom        6000
top           12000
average       9000.0
Name: 164, dtype: object

[In [34]: df[['职位月薪','bottom','top','average']].head(10)
Out[34]:
```

ZL_Job_id	职位月薪	bottom	top	average
1	面议	面议	面议	面议
2	6001-8000元 /月	6001	8000	7000.5
3	4001-6000元 /月	4001	6000	5000.5
4	2001-4000元 /月	2001	4000	3000.5
5	4001-6000元 /月	4001	6000	5000.5
6	2001-4000元 /月	2001	4000	3000.5
7	15001-20000元 /月	15001	20000	17500.5
8	4001-6000元 /月	4001	6000	5000.5
9	5000-8000元 /月	5000	8000	6500.0
10	8001-10000元 /月	8001	10000	9000.5

```

[In [35]: print(q1,q2,q3,q4,q1+q2+q3+q4)
16905 61 1360 0 18326

```

经过检查，可以发现【职位月薪】和新增的bottom、top、average列是能对的上。其中形如‘6000-8000元/月’的有16905条、形如‘10000元以下’的有61条、‘found no element’和‘面议’加起来有1360条，总数18326条，可见是正确的。

4.进行【工作地点】列的处理，新增【工作城市】列，将工作地点中如‘苏州-姑苏区’、‘苏州-工业园区’等统统转化为‘苏州’存放在【工作城市】列。

```

df['工作城市'] = df['工作地点']
pattern2 = re.compile('(.?)(\-)')
df_city = df['工作地点'].copy()

for i in range(len(df_city)):
    item = df_city.iloc[i].strip()
    result = re.search(pattern2,item)
    if result:
        print(result.group(1).strip())
        df_city.iloc[i] = result.group(1).strip()
    else:
        print(item.strip())
        df_city.iloc[i] = item.strip()

```

赞同 48

21 条评论

分享

收藏

```
df['工作城市'] = df_city  
df[['工作地点', '工作城市']].head(20)
```

```
[In [48]: df[['工作地点', '工作城市']].head(20)  
Out[48]:
```

	工作地点	工作城市
ZL_Job_id		
1	苏州	苏州
2	苏州-工业园区	苏州
3	苏州	苏州
4	苏州-工业园区	苏州
5	苏州	苏州
6	苏州	苏州
7	苏州-工业园区	苏州
8	苏州	苏州
9	苏州	苏州
10	苏州-工业园区	苏州
11	苏州-工业园区	苏州
12	苏州	苏州
13	苏州-姑苏区	苏州
14	苏州	苏州
15	苏州	苏州
16	苏州-工业园区	苏州
17	苏州-工业园区	苏州
18	苏州	苏州
19	苏州	苏州
20	苏州	苏州

检查一下，没有错误，可以进行下一步的操作了！

数据分析和可视化

从可读性来看，应该是先进行数据清洗，然后进行分析及可视化，但是实际过程中，往往是交织在一起的，所有下面让我们一步步来，完成所有的清洗、分析和可视化工作。除了具体的公司和职位名称以外，我们还比较关心几个关键词：平均月薪、工作经验、工作城市、最低学历和岗位职责描述，这里岗位职责描述以后会用python分词做词云图，所以目前筛选出【平均月薪】、【工作经验】、【工作城市】、【最低学历】这四个标签，这些标签可以两两组合产生各种数据。譬如我想知道各个城市的招聘数量分布情况，会不会大部分的工作机会都集中在北上广深？是不是北上广深的平均工资也高于其他城市？我想知道Python这个关键词的18000多条招聘数据中 对学历的要求和对工作经验的要求，以及它们分别占比多少？我还想知道平均月薪和工作经验的关系？最低学历和平均月薪的关系？和上一篇（Excel篇）类似，不同的是，这次我们完全用Python实现同样的操作。

1.各个城市职位数量及分布

赞同 48

21 条评论

分享

收藏

根据猜想，北上广深，一定占据了Python这个关键词下大部分的工作机会，会不会符合28定律？20%的城市占据了80%的岗位？有可能！我们先用df.工作城市.value_counts()看一下究竟有多少个城市，以及他们各自有多少条工作数据？

```
df.工作城市.value_counts()#等价于df['工作城市'].value_counts()
```

```
#再用count()来看一下统计出来的城市数量
```

```
df.工作城市.value_counts().count()
```

```
type(df.工作城市.value_counts())#用type()查看下类型。
```

```
福州          179
青岛          120
沈阳          118
大连          109
无锡          103
found no element 95
石家庄         60
哈尔滨         60
长春           60
贵阳           60
佛山           60
南昌           60
太原市         60
吉林市         8
辽源           3
松原           3
白山           2
延边           2
通化           2
公主岭         2
四平           2
白城           2
珲春           2
Name: 工作城市, dtype: int64

[In [8]: df.工作城市.value_counts().count()
Out[8]: 40

[In [9]: print(type(df.工作城市.value_counts()))
<class 'pandas.core.series.Series'>

In [10]:
```

可以看到，明明设置的是搜索30个城市，怎么变成了40？像延边、珲春、白山。。。是什么鬼？想了一下，这些城市是搜索关键词城市‘吉林市’时，自动冒出来的；还有95个‘found no element’，是这些职位链接本身就没有填写工作城市，为了避免干扰，要把他们统统替换成空值。用df_工作城市 = df['工作城市'].replace()

[赞同 48](#)[21 条评论](#)[分享](#)[收藏](#)

#将原来df['工作城市']列中选定的字段替换成空值nan

```
df_工作城市 = df['工作城市'].replace(['found no element','松原','辽源','珲春','白山','公主岭','白城','延边','四平','通化'],np.nan)
```

#查看替换后各个城市职位计数

```
df_工作城市.value_counts()
```

#查看替换后城市所包含的职位总数；查看替换后的城市数量，是否等于30。

```
df_工作城市
```

#将新的[df_工作城市]列添加到df表中，留作备用

```
df['df_工作城市'] = df_工作城市
```

```
In [46]: df_工作城市 = df['工作城市'].replace(['found no element','松原','辽源','珲春','白山','公主岭','白城','延边','四平','通化'],np.nan)

In [47]: df_工作城市.value_counts()
Out[47]:
北京      4924
上海      3114
深圳      1933
杭州      1244
广州      1171
成都       985
南京       701
郑州       477
西安       417
武汉       412
天津       355
苏州       347
济南       240
长沙       239
合肥       236
厦门       180
重庆       179
福州       179
青岛       120
沈阳       118
大连       109
无锡       103
太原        60
贵阳        60
长春        60
佛山        60
哈尔滨      60
石家庄      60
南昌        60
吉林市        8
Name: 工作城市, dtype: int64

In [48]: print(df_工作城市.value_counts().sum(),df_工作城市.value_counts().count())
18211 30
```

看了一下，没有问题，现在df_工作城市中筛选出了30个城市，合计18211条职位数据。为了数据完整性，df表保持原样，我们用df_工作城市，再一一解释下。

赞同 48

21 条评论

分享

收藏


```

fig1 = plt.figure(1,facecolor = 'black')#设置视图画布1
ax1 = fig1.add_subplot(2,1,1,facecolor='#4f4f4f',alpha=0.3)#在视图1中设置子图1,背景色灰色
plt.tick_params(colors='white')#设置轴的颜色为白色
df_工作城市.value_counts().plot(kind='bar',rot=0,color='#ef9d9a')#画直方图图

#设置图标题, x和y轴标题
title = plt.title('城市—职位数分布图',fontsize=18,color='yellow')#设置标题
xlabel = plt.xlabel('城市',fontsize=14,color='yellow')#设置X轴轴标题
ylabel = plt.ylabel('职位数量',fontsize=14,color='yellow')#设置Y轴轴标题

#设置说明, 位置在图的右上角
text1 = ax1.text(25,4500,'城市总数:30(个)',fontsize=12, color='cyan')#设置说明, 位置在图
text2 = ax1.text(25,4000,'职位总数:18326(条)',fontsize=12, color='cyan')
text3 = ax1.text(25,3500,'有效职位:18211(条)',fontsize=12, color='red')

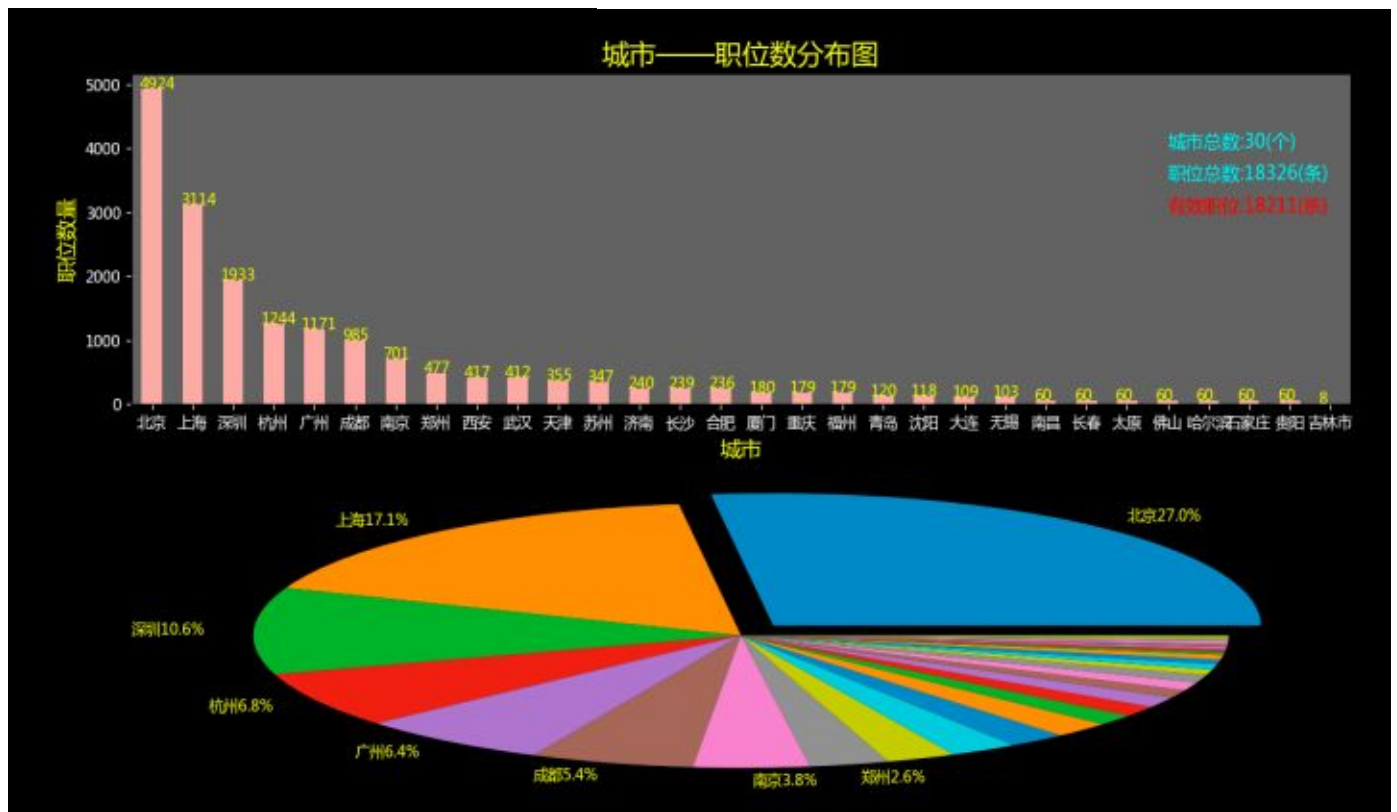
#添加每一个城市的坐标值
for i in range(len(list_1)):
    ax1.text(i-0.3,list_1[i],str(list_1[i]),color='yellow')

#可以用plt.grid(True)添加栅格线
#可以用下面语句添加注释箭头。指向上海, xy为坐标值、xytext为注释坐标值, facecolor为箭头颜色。
#arrow = plt.annotate('职位数:3107', xy=(1,3107), xytext=(3, 4000),color='blue',arrowpi
ax2 = fig1.add_subplot(2,1,2)#设置子图2, 是位于子图1下面的饼状图
#为了方便, 显示前8个城市的城市名称和比例、其余的不显示, 用空字符列表替代, 为此需要构造列表label
x = df_工作城市.value_counts().values#x是数值列表, pie图的比例根据数值占整体的比例而划分
label_list = []#label_list是构造的列表, 装的是前8个城市的名称+职位占比。
for i in range(8):
    t = df_工作城市.value_counts().values[i]/df_工作城市.value_counts().sum()*100
    city = df_工作城市.value_counts().index[i]
    percent = str('%.1f%%'%t)
    label_list.append(city+percent)

#labels参数原本是与数值对应的标签列表, 此处30个城市过多, 所以只取了前8个城市显示。
#explode即饼图中分裂的效果explode=(0.1, 1, 1, ...)表示第一块图片显示为分裂效果
labels = label_list + ['']*22
explode = tuple([0.1]+[0]*29)
plt.pie(x,explode=explode,labels=labels,textprops={'color':'yellow'})

#可加参数autopct='%.1f%%'来显示饼图中每一块的比例, 但是此处30个城市, 如果全显示的话会非常拥挤
#若要显示标准圆形, 可以添加: plt.axis('equal')

```



可以看见，这个曲线下降的弧度还是挺美的，北上深杭广5个城市占据了超过60%以上的职位数。其中北京当之无愧的占据了四分之一的Python工作数量，不愧为帝都。上海以3107条职位排名第二，可见上海虽然经济超越北京，在互联网环境和工作机遇方面还需努力！深圳作为中国的科技中心，排名第三我是没疑问的，杭州竟然超过广州排名第四！不过也可以想到，阿里巴巴、百草味等等电商产业带动了整个杭州的互联网文化！

【北上深杭广】+成都、南京、郑州，这8个城市占据了全国30座城市中，近80%的工作机会！剩下的22个城市合起来只占据了20%，果然，是基本符合28定律的。。。

2.工作经验-职位数量及分布

Python虽然是一名比较老的语言，但是在人们的印象中火起来也就最近几年，Python相关的工作对于【工作经验】是怎样要求的呢？让我们来看看！

```
df.工作经验.value_counts()#统计【工作经验】下各个字段的累计和
```

赞同 48

21 条评论

分享

收藏

```
In [54]: df.工作经验.value_counts()
Out[54]:
不限                6183
3-5年              5164
1-3年              4841
5-10年             1516
无经验              366
1年以下            111
found no element    95
10年以上            34
3年以上              9
1年以上              3
5年以上              2
2年以上              1
1-2年                1
Name: 工作经验, dtype: int64
```

可以看见出现了一些很数字少量的字段譬如“5年以上”，“2年以上”，“1-2年”，“1年以上”等，这些标签下职位的数量都在10以内，不太具备统计意义，所以我们作图的时候不想让他们出现，必须筛选掉。下面我们还是通过同样的步骤来清除掉此类数据。

```
df_工作经验 = df['工作经验'].replace(['found no element','3年以上','1年以上','5年以上','2
df_工作经验.value_counts()
df_工作经验.value_counts().sum()
```

```
In [44]: df_工作经验 = df['工作经验'].replace(['found no element','3年以上','1年以上','5
...: 年以上','2年以上','1-2年'],np.nan)
In [45]: df_工作经验.value_counts()
Out[45]:
不限                6183
3-5年              5164
1-3年              4841
5-10年             1516
无经验              366
1年以下            111
10年以上            34
Name: 工作经验, dtype: int64
In [46]: df_工作经验.value_counts().sum()
Out[46]: 18215
In [47]:
```

现在，可以进行下一步可视化了，还是做2个图，一个是不同工作经验的要求占比，如

赞同 48

21 条评论

分享

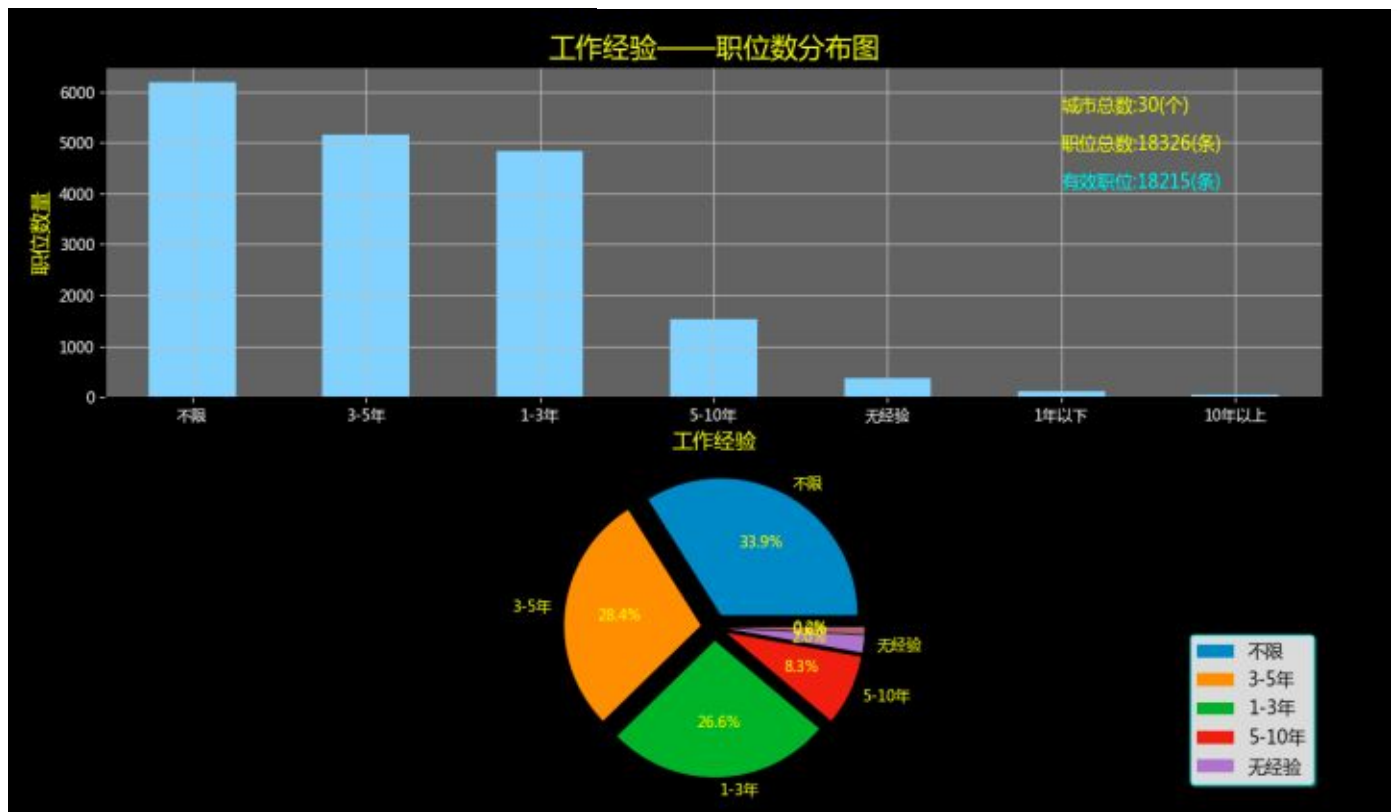
收藏

```
fig2 = plt.figure(2,facecolor = 'black')
ax2_1 = fig2.add_subplot(2,1,1,facecolor='#4f4f4f',alpha=0.3)
plt.tick_params(colors='white')
df_工作经验.value_counts().plot(kind = 'bar',rot = 0,color='#7fc8ff')
title = plt.title('工作经验—职位数分布图',fontsize = 18,color = 'yellow')
xlabel = plt.xlabel('工作经验',fontsize = 14,color = 'yellow')
ylabel = plt.ylabel('职位数量',fontsize = 14,color = 'yellow')
plt.grid(True)
text1_ = ax2_1.text(5,5600,'城市总数:30(个)',fontsize=12, color='yellow')
text2 = ax2_1.text(5,4850,'职位总数:18326(条)',fontsize=12, color='yellow')
text3 = ax2_1.text(5,4100,'有效职位:18215(条)',fontsize=12, color='cyan')

#设置子图2，是位于子图1下面的饼状图
ax2_2 = fig2.add_subplot(2,1,2)

#x是数值列表，pie图的比例根据数值占整体的比例而划分
x2 = df_工作经验.value_counts().values
labels = list(df_工作经验.value_counts().index[:5])+ ['']*2
explode = tuple([0.1,0.1,0.1,0.1,0.1,0.1,0.1])
plt.pie(x2,explode=explode,labels=labels,autopct='%1.1f%%',textprops={'color':'yellow'})
plt.axis('equal')#显示为等比例圆形

#设置图例，方位为右下角
legend = ax2_2.legend(loc='lower right',shadow=True,fontsize=12,edgecolor='cyan')
```



总共得到18215条职位。从直方图里可以明显看出工作机会集中在'不限'、'1-3年'、'3-5年'，其中工作经验要求3年以下的（【无经验】+【不限】+【1年以下】+【1-3年】）合计11501条职位，占比超过63%，看来即使是初入门者，大家的机会也还是有不少的！（PS:最后,在df表中添加一列'df_工作经验'，以后筛选时就可以直接用了，df['df_工作经验']=df_工作经验）

3.工作经验-平均月薪

这个嘛，大家闭着眼都能想到！肯定是工作经验越久的拿钱越多了！再猜猜？无经验的和5-10年经验的收入差距有多大？这个，嘿嘿就不好猜了，让我们来看看吧！

1.第一步，要想统计工作经验和平均月薪的关系，那么我们先看看df中对应的列df.工作经验和df.average。之前我们构造了一列df_工作经验，把df.工作经验中几个样本容量小于10的值和'found no element'全筛选掉了，故df_工作经验还能继续使用。现在，让我们看看df.average的信息。

```
df.average.value_counts()
```

可以看到，其中有1265个值是'面议'，有95个值是'found no element'，这些值需要替换成空值，不然会影响下一步工资的计算。

```
df_平均月薪 = df['average'].replace(['面议','found no element'],np.nan)
```

2.好了，第一步的简单数据清洗完成了，我们可以思考下一步了，现在我们想要得到的是不同工作经验字段下的平均月薪

赞同 48

21 条评论

分享

收藏

- A. 首先我需要把df_工作经验和df_平均月薪这两列元素放在一起，构造一个DataFrame用于存放df_工作经验和df_平均月薪这两列元素,且方便进一步的groupby操作。
- B. 其次我需要把df_平均月薪列根据df_工作经验进行分组(用groupby),分组后我可以求得df_工作经验下各个字段的月薪的计数、最大值最小值、累加和、平均值等一系列数据。
- C. 当然此处我只需要平均值。对分组后的grouped用mean()方法，就可以轻松统计分组内各项的平均值了。

```
df3=pd.DataFrame(data={'工作经验':df['df_工作经验'],'平均月薪':df_平均月薪})
df3.info()
grouped3 = df3['平均月薪'].groupby(df3['工作经验'])
grouped3.mean()
```

```
[In [65]: df_平均月薪.value_counts().sum()
Out[65]: 16966

[In [66]: df3.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18326 entries, 1 to 19033
Data columns (total 2 columns):
工作经验      18215 non-null object
平均月薪      16966 non-null object
dtypes: object(2)
memory usage: 1.0+ MB

[In [67]: grouped3 = df3['平均月薪'].groupby(df3['工作经验'])

[In [68]: grouped3.mean()
-----
DataError                                Traceback (most recent call last)
<ipython-input-68-ad84a1924c36> in <module>()
----> 1 grouped3.mean()
```

在进行grouped3.mean()时，我们发现报错了：DataError: No numeric types to aggregate，看一下，原来df_平均月薪列里的值都是字符型str，并不是数值型的float，因为前面的步骤没有做好，留下了这个bug，无奈我们需要对值类型做个转换。

#构造一个list存放转化后float型的‘平均月薪’

```
import re
pattern = re.compile('([0-9]+)')
listi = []
for i in range(len(df.average)):
    item = df.average.iloc[i].strip()
    result = re.findall(pattern,item)
    try:
        if result:
            listi.append(float(result[0]))
```

赞同 48

21 条评论

分享

收藏

```

elif (item.strip()=='found no element' or item.strip()=='面议'):
    listi.append(np.nan)
else:
    print(item)
except Exception as e:
    print(item,type(item),repr(e))

```

#将df3.平均月薪列替换掉,同时给df新增一列'df_平均月薪'做备用。

```
df3['平均月薪'] = listi
```

```
df['df_平均月薪'] = df3['平均月薪']
```

#看看更新后的数据是否正确

```
df3['平均月薪'].value_counts()#统计每个月薪字段的个数
```

```
df3['平均月薪'][:10]#查看前10个值
```

```
type(df3['平均月薪'][1])#看看现在月薪的类型是不是浮点型
```

```
df3['平均月薪'].value_counts().sum()#看看月薪样本总数
```

```
df3['平均月薪'].mean()#看看这16966个月薪样本的平均值是多少?
```

```

In [76]: df3['平均月薪'][:10]
Out[76]:
ZL_Job_id
1      NaN
2    7000.0
3    5000.0
4    3000.0
5    5000.0
6    3000.0
7   17500.0
8    5000.0
9    6500.0
10   9000.0
Name: 平均月薪, dtype: float64

In [77]: type(df3['平均月薪'][1])
Out[77]: numpy.float64

In [78]: df3['平均月薪'].value_counts().sum()
Out[78]: 16966

In [79]: df3['平均月薪'].mean()
Out[79]: 14197.406459978782

```

可以看到, 替换后的df3['平均月薪']值从str变为了可以计算的float, 月薪样本总数16966个, 样本的平均月薪14197元。好, 现在终于OK了, 让我们再回到之前的步骤:

```

grouped3 = df3['平均月薪'].groupby(df)
grouped3.mean()

```

赞同 48

21 条评论

分享

收藏

```
[In [80]: grouped3 = df3['平均月薪'].groupby(df3['工作经验'])

[In [81]: grouped3.mean()
Out[81]:
工作经验
1-3年      11986.983896
10年以上   34890.625000
1年以下    7579.990476
3-5年      16385.524958
5-10年     23638.390007
不限       12373.108474
无经验     5842.827869
Name: 平均月薪, dtype: float64
```

好了，完美，格式对了，数据有了，现在可以来画图了！但是再看看，还不是那么完美，数据大小排列很乱，而且小数点那么多。。。好吧，让我们再简单处理下

#新增一个平均值，即所有非空df3['平均月薪']的平均值

```
s3 = pd.Series(data = {'平均值':df3['平均月薪'].mean()})
```

```
result3 = grouped3.mean().append(s3)
```

#sort_values()方法可以对值进行排序，默认按照升序，round（1）表示小数点后保留1位小数。

```
result3.sort_values(ascending=False).round(1)
```

```
[In [82]: s3 = pd.Series(data = {'平均值':df3['平均月薪'].mean()})

[In [83]: result3 = grouped3.mean().append(s3)

[In [84]: result3.sort_values(ascending=False).round(1)
Out[84]:
10年以上      34890.6
5-10年       23638.4
3-5年        16385.5
平均值       14197.4
不限         12373.1
1-3年        11987.0
1年以下       7580.0
无经验        5842.8
dtype: float64
```

3.数据可视化

这次我们画一个躺倒的柱状图(barh)，用g

赞同 48

21 条评论

分享

收藏

```
matplotlib.style.use('ggplot')
fig3 = plt.figure(3,facecolor = 'black')
ax3 = fig3.add_subplot(1,1,1,facecolor='#4f4f4f',alpha=0.3)
result3.sort_values(ascending=False).round(1).plot(kind='barh',rot=0)

#设置标题、x轴、y轴的标签文本
title = plt.title('工作经验—平均月薪分布图',fontsize = 18,color = 'yellow')
xlabel= plt.xlabel('平均月薪',fontsize = 14,color = 'yellow')
ylabel = plt.ylabel('工作经验',fontsize = 14,color = 'yellow')

#添加值标签
list3 = result3.sort_values(ascending=False).values
for i in range(len(list3)):
    ax3.text(list3[i],i,str(int(list3[i])),color='yellow')

#设置标识箭头
arrow = plt.annotate('Python平均月薪:14197元/月', xy=(14197,3.25), xytext=(20000,4.05),

#设置图例注释（16966来源: df2['平均月薪'].value_counts().sum()）
text= ax3.text(27500,6.05,'月薪样本数:16966(个)',fontsize=16, color='cyan')

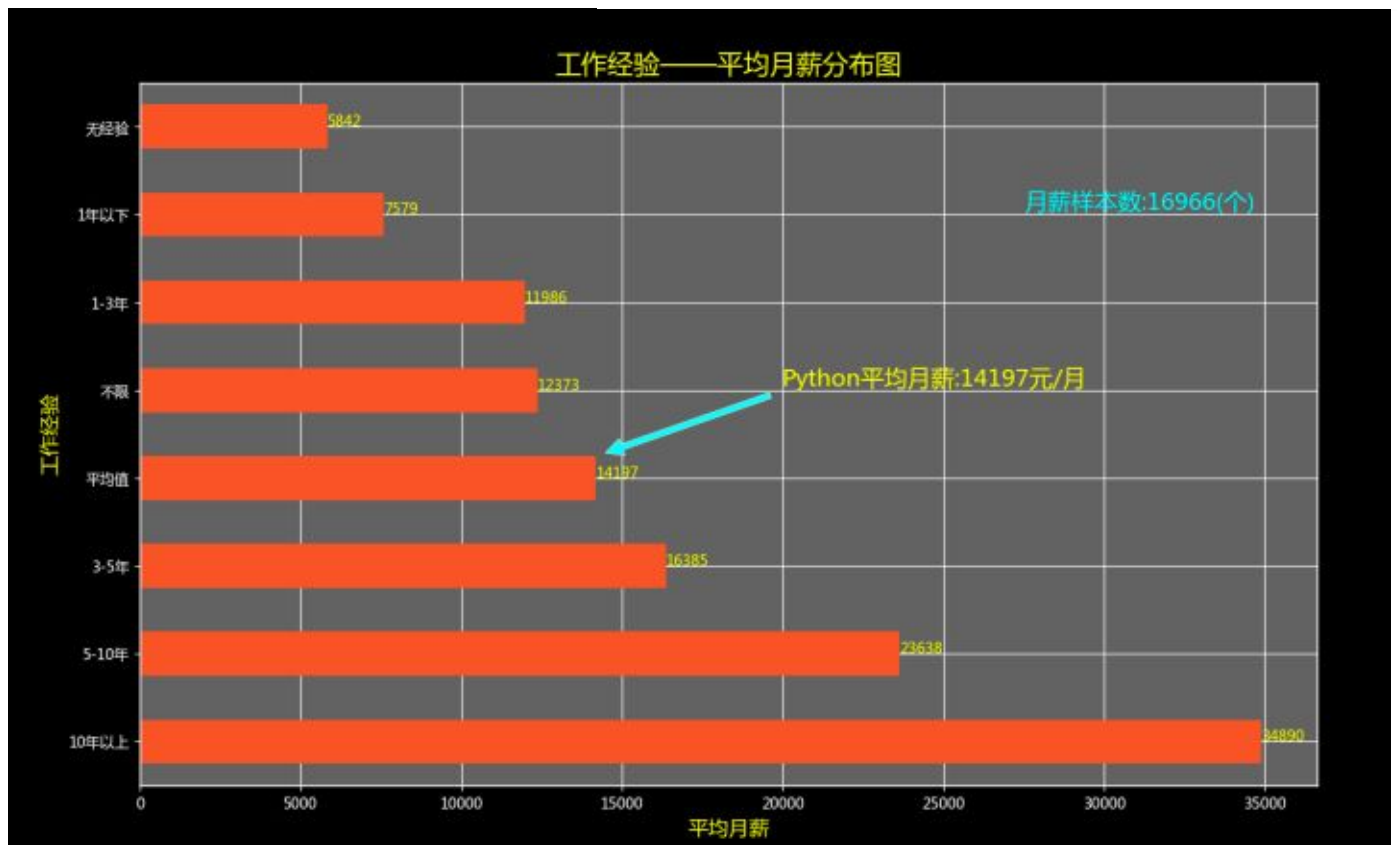
#设置轴刻度文字颜色为白色
plt.tick_params(colors='white')
```

赞同 48

21 条评论

分享

收藏



通过图表，我们可以直观地看到，Python关键词下的职位月薪是随着工作经验增长而递增的（这不是说了一句废话么？！囧）其中【无经验】的平均月薪最低，只有5842，相比之下【10年以上】经验的，平均月薪达到了恐怖的34890，约达到了【无经验】月薪的6倍之多！！！【1年以下】的平均月薪7579，还勉强凑合，【1-3年】的已经破万了，达到了近12000元/月的水准。最后让我们看看平均值吧，由于‘被平均’的缘故，16966条月薪样本的均值是14197元，有没有让你满意呢？

4.工作城市-平均月薪

对了，刚才说到北上广深占据了全国大部分的工作机会，那么北上广深的平均月薪如何呢？会不会也碾压小城市？让我们来看看！和之前的套路一样，我们还是要构造一个DataFrame，包含两列，一列是【平均月薪】，一列是【工作城市】，然后对df4进行groupby操作，还是很简单的！不过，经过上次的教训，平均月薪一定要是数值型的，str型的计算不了。

```
#此处df['df_工作城市']是之前经过筛选后的30个城市数据
df4=pd.DataFrame(data={'工作城市':df['df_工作城市'],'平均月薪':df['df_平均月薪']})
df4.info()
grouped4 = df4['平均月薪'].groupby(df4['工作城市'])
grouped4.mean()#查看对30个城市分组后，各个城市月薪的平均值
grouped4.count().sum()#查看对30个城市分组后筛选出的平均月薪样本数
```

```
#新增一个平均值，即所有非空df2['平均月薪']
s4 = pd.Series(data = {'平均值':df['df_平均月薪']})
```

赞同 48

21 条评论

分享

收藏


```
result4 = grouped4.mean().append(s4)
```

#sort_values()方法可以对值进行排序，默认按照升序，round（1）表示小数点后保留1位小数。

```
result4.sort_values(ascending=False).round(1)
```

```
[In [92]: df4.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18326 entries, 1 to 19033
Data columns (total 2 columns):
工作城市    18211 non-null object
平均月薪    16966 non-null float64
dtypes: float64(1), object(1)
memory usage: 1.0+ MB

[In [93]: grouped4 = df4['平均月薪'].groupby(df4['工作城市'])

[In [94]: grouped4.mean()
Out[94]:
工作城市
上海      16496.360096
佛山      11834.821429
北京      17177.895796
南京      12742.367733
南昌       9652.500000
厦门      11016.949153
合肥      10445.394737
吉林市    10125.000000
哈尔滨     8908.333333
大连      10702.380952
天津       9461.048159
太原       9425.000000
广州      12962.815552
成都      10348.099156
无锡       9780.612245
杭州      14299.906593
武汉      11339.549751
沈阳      10340.090090
济南      10220.905172
深圳      15876.068031
石家庄     7059.322034
福州      10429.775281
苏州      11146.484950
西安       9509.849754
贵阳       7510.550000
```

```
[In [95]: s4 = pd.Series(data = {'平均值':df['df_平均月薪'].mean()})

[In [96]: result4 = grouped4.mean().append(s4)

[In [97]: result4.sort_values(ascending=False).round(1)
Out[97]:
北京      17177.9
上海      16496.4
深圳      15876.1
杭州      14299.9
平均值    14197.4
广州      12962.8
南京      12742.4
佛山      11834.8
武汉      11339.5
苏州      11146.5
厦门      11016.9
大连      10702.4
合肥      10445.4
福州      10429.8
成都      10348.1
沈阳      10340.1
济南      10220.9
吉林市    10125.0
长沙      9954.8
无锡      9780.6
南昌      9652.5
西安      9509.8
天津      9461.0
太原      9425.0
长春      9241.6
重庆      9222.5
哈尔滨    8908.3
郑州      8838.4
青岛      8836.9
贵阳      7563.6
石家庄    7059.3
dtype: float64
```

数据构造好了，进行下一步，可视化。

#可以通过style.available查看可用的绘图风格，总有一款适合你

```
matplotlib.style.use('dark_background')
```

```
fig4 = plt.figure(4)
```

```
ax4 = fig4.add_subplot(1,1,1)#可选facecolor='#4f4f4f',alpha=0.3, 设置子图,背景色灰色,透I
result4.sort_values(ascending=False).round(1).plot(kind='bar',rot=30)#可选color='#ef9d'
```

#设置图标题，x和y轴标题

```
title = plt.title(u'城市—平均月薪分布图',fontsize=18,color='yellow')#设置标题
```

```
xlabel = plt.xlabel(u'城市',fontsize=14,color='yellow')#设置X轴轴标题
```

```
ylabel = plt.ylabel(u'平均月薪',fontstyle='italic',color='yellow')#设置Y轴轴标题
```

赞同 48

21 条评论

分享

收藏

#设置说明，位置在图的右上角

```
text1 = ax4.text(25,16250,u'城市总数:30(个)',fontsize=12, color='#FF00FF')#设置说明，位置
```

```
text2 = ax4.text(25,15100,u'月薪样本数:16946(条)',fontsize=12, color='#FF00FF')
```

#添加每一个城市的坐标值

```
list_4 = result4.sort_values(ascending=False).values
```

```
for i in range(len(list_4)):
```

```
    ax4.text(i-0.5,list_4[i],int(list_4[i]),color='yellow')
```

#设置箭头注释

```
arrow = plt.annotate(u'全国月薪平均值:14197元/月', xy=(4.5,14197), xytext=(7,15000),color
```

#设置轴刻度文字颜色为粉色

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



可以看见，Python这个关键词下，全国16946条样本的月薪平均值为14197元/月，平均月薪排名前5的城市分别是：北京、上海、深圳、杭州、广州。哎，记得之前城市—职位数分布图么？全国30个城市中，职位数排名前5的也是这5座城市！看来北上广深杭不仅集中了全国大部分的职位数量、连平均工资也是领跑全国的！不禁让人觉得越大越强！但是在超级大城市奋斗，买房总是遥遥无期，房子在中国人的概念里，有着特殊的情节，意味着家，老小妻儿生活的地方，给人一种安全感！我们可以看到还有不少城市的平均月薪也破万了，在这些超一线城市之外，选择一个二线城市，买房还是有希望的，哈哈！譬如南京、

赞同 48

21 条评论

分享

收藏

5.学历-职位数量

直觉来看Python这类工作职位，应该是本科及以上学历要求居多吧？那么工作经验【不限】和【大专】的机会占比多少呢？让我们来看看！首先，还是用df['最低学历'].value_counts()来看一下有哪些字段，以及各个字段的统计值。

```
df['最低学历'].value_counts()
df_最低学历=df['最低学历'].replace(['中技','其他','高中','found no element'],np.nan)
df_最低学历.value_counts()
df_最低学历.value_counts().sum()
df['df_最低学历'] = df_最低学历 #留作备用
```

```
[In [7]: df.最低学历.value_counts()
Out[7]:
本科          9954
大专          3704
不限          3205
硕士          1137
found no element    95
博士           88
中专           31
中技            8
其他            7
高中            2
Name: 最低学历, dtype: int64

In [8]: df_最低学历=df['最低学历'].replace(['中技','其他','高中','found no eleme
...: nt'],np.nan)

[In [9]: df_最低学历.value_counts()
Out[9]:
本科          9954
大专          3704
不限          3205
硕士          1137
博士           88
中专           31
Name: 最低学历, dtype: int64

[In [10]: df_最低学历.value_counts().sum()
Out[10]: 18119

[In [11]: df['df_最低学历'] = df_最低学历
```

可以看到对于学历要求，最多的集中在大专、本科、硕士、不限还有极少数的博士和中专学历。至于中技、其他、高中则连10个都不到，对

赞同 48

21 条评论

分享

收藏

将其排除（并没有歧视低学历的意思啊囧！）

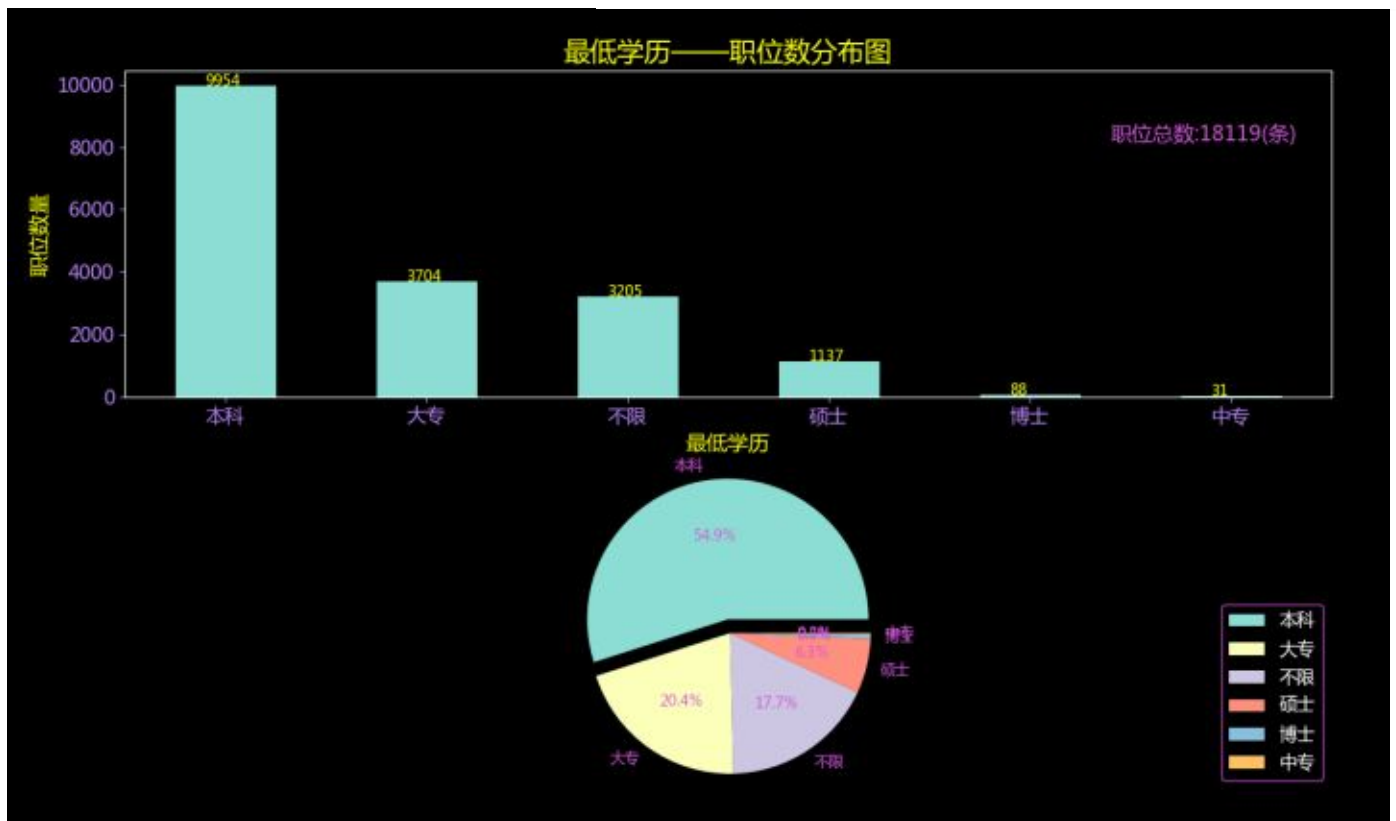
可以看到排除后还剩余6个字段，共计18119个职位，下一步，还是来经典的条形分布图和饼图！

```
fig5 = plt.figure(5)
ax5_1 = fig5.add_subplot(2,1,1) #可选facecolor='#4f4f4f',alpha=0.3
df_最低学历.value_counts().plot(kind = 'bar',rot=0)    #color='#7fc8ff'

#设置标题、x轴和y轴标题、图例文字
title = plt.title(u'最低学历—职位数分布图',fontsize = 18,color = 'yellow')
xlabel = plt.xlabel(u'最低学历',fontsize = 14,color = 'yellow')
ylabel = plt.ylabel(u'职位数量',fontsize = 14,color = 'yellow')
text1 = ax5_1.text(4.4,8200,u'职位总数:18119(条)',fontsize=14, color='#B452CD')

#设置坐标轴的颜色和文字大小
plt.tick_params(colors='#9F79EE',labelsize=13)

#设置坐标值文字
list5 = df_最低学历.value_counts().values
for i in range(len(list5)):
    ax5_1.text(i-0.1,list5[i],int(list5[i]),color='yellow')
ax5_2=fig5.add_subplot(2,1,2)
x1 = df_最低学历.value_counts().values
labels = list(df_最低学历.value_counts().index)
explode = tuple([0.1,0,0,0,0,0])
plt.pie(x1,explode=explode,labels=labels,autopct='%1.1f%%',textprops={'color': '#B452CD'})
plt.axis('equal')
legend = ax5_2.legend(loc='lower right',shadow=True,fontsize=12,edgecolor='#B452CD')
plt.tick_params(colors='#9F79EE',labelsize=13)
```

可见【本科】独占鳌头，占据了超过50%的市场！【不限】和【大专】也合计占比38%不容小觑！看起来，只要技术过硬，学历从来都不是问题！！！作为对比【硕士】占比6%，【博士】更是少到只有1%，果然稀缺到百里挑一！

6.最低学历-平均月薪

按道理学历越高，平均月薪越高，类似工作经验一样都是正相关，到底是不是呢？来看一下！构造一个DataFrame(df6), 包含两列最低学历和平均月薪，我们直接用之前构造好的df中的【df_最低学历】和【df_平均月薪】即可，然后还是熟悉的groupby(df_最低学历)

```
df6=pd.DataFrame(data={'最低学历':df['df_最低学历'],'平均月薪':df['df_平均月薪']})
df6.info()
grouped6 = df6['平均月薪'].groupby(df6['最低学历'])
```

#查看grouped6的信息

```
grouped6.mean()
grouped6.count()
grouped6.count().sum()
matplotlib.style.use('ggplot')
fig = plt.figure(6,facecolor = 'black')
ax6 = fig6.add_subplot(1,1,1,facecolor='#4f4f4f',alpha=0.3)
grouped6.mean().round(1).sort_values().plot(color = 'cyan')#在条形图上叠加一个折线图
grouped6.mean().round(1).sort_values
```

赞同 48

21 条评论

分享

收藏

```
#设置标题、x轴、y轴的标签文本
title = plt.title(u'最低学历—平均月薪分布图',fontsize = 18,color = 'yellow')
xlabel= plt.xlabel(u'最低学历',fontsize = 14,color = 'yellow')
ylabel = plt.ylabel(u'平均月薪',fontsize = 14,color = 'yellow')

#添加值标签(坐标值文字)
list6 = grouped6.mean().round(1).sort_values().values
for i in range(len(list6)):
    ax6.text(i-0.1,list6[i],int(list6[i]),color='yellow')

#设置图例注释
text= ax6.text(0,27000,u'月薪样本数:16956(个)',fontsize=16, color='cyan')

#设置轴刻度的文字颜色
plt.tick_params(colors='#9F79EE')
```

平均月薪14139元，可以看到学历越高果然工资越高，博士级别的更是碾压，达到了29562元。只要学历在【大专】以上，那么平均月薪都已经过万了。 BUT，重点来了，学历并不是万能的，一个【中专】学历，有超过5年经验的，工资一定超过【本科】毕业无工作经验的。所以大家看看就好，不要当真，哈哈！

7.最低学历-工作经验-平均月薪

看了前面的图表，大家都知道了，学历越高平均月薪越高，工作经验越高平均月薪越高，但是我想看看更细粒度的情形呢？譬如我想知道【大学+无经验】和【大学+1-3年】工资的差别，我想看看【大专+3-5年】和【硕士+无经验】工资的对比究竟谁高？现在，我不知道，但是接下来让我们把这些情况用图表呈现出来，大家就会一目了然！

```
df7 = pd.DataFrame(data = {'平均月薪':df['df_平均月薪'],'最低学历':df['df_最低学历'],'工作经验':df['df_工作经验']})
df7.info()
grouped7 = df7['平均月薪'].groupby([df7['最低学历'],df7['工作经验']])

#查看grouped7的信息
grouped7.mean().round(1)
grouped7.count()
grouped7.count().sum()
```

[赞同 48](#)[21 条评论](#)[分享](#)[收藏](#)

```
[In [6]: df7.info()
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18326 entries, 0 to 18325
Data columns (total 3 columns):
工作经验      18215 non-null object
平均月薪      16966 non-null float64
最低学历      18119 non-null object
dtypes: float64(1), object(2)
memory usage: 572.7+ KB

[In [7]: grouped7 = df7['平均月薪'].groupby([df7['最低学历'],df7['工作经验']])

[In [8]: grouped7.mean().round(1)
Out[8]:
最低学历  工作经验
不限      1-3年      11133.0
          10年以上     55000.0
          1年以下      6916.7
          3-5年      15541.5
          5-10年     21362.6
          不限      13578.4
          无经验      4846.8
中专      1-3年      7600.0
          1年以下      7000.0
          3-5年      7500.0
          5-10年     22500.0
          不限      5166.7
          无经验      4333.3
博士      1-3年     22538.5
          3-5年     26800.0
          5-10年     43842.3
```

其实我们输入`type(grouped7.mean())`，会发现它是一个包含了层次化索引的Series结构。其中第一层索引是【最低学历】 第二层索引是【工作经验】，数值列【平均月薪】被这两层索引所分配！下面我们开始准备可视化，还是画一个bar柱状图，不过这次画的是多列一起呈现的形式，Y轴表示职位月薪、X轴表示最低学历，在每个学历字段下，又分别添加不同工作经验的列！

```
grouped7.mean().round(1)[:,'1-3年']
grouped7.mean().round(1)[:,'1-3年'].sort_values()
xlist = list(grouped7.mean().round(1)[:,'1-3年'].sort_values().index)
grouped7.mean().round(1)[:,'1-3年'].reindex(xlist)
print(xlist)
```

```

In [184]: grouped7.mean().round(1)[:,'1-3年']
Out[184]:
最低学历
不限      11133.0
中专      7600.0
博士     22538.5
大专      9811.3
本科     12419.9
硕士     17564.8
Name: 平均月薪, dtype: float64

In [185]: grouped7.mean().round(1)[:,'1-3年'].sort_values()
Out[185]:
最低学历
中专      7600.0
大专      9811.3
不限     11133.0
本科     12419.9
硕士     17564.8
博士     22538.5
Name: 平均月薪, dtype: float64

In [186]: xlist = list(grouped7.mean().round(1)[:,'1-3年'].sort_values().index)

In [187]: grouped7.mean().round(1)[:,'1-3年'].reindex(xlist)
Out[187]:
最低学历
中专      7600.0
大专      9811.3
不限     11133.0
本科     12419.9
硕士     17564.8
博士     22538.5
Name: 平均月薪, dtype: float64

In [188]: xlist
Out[188]: ['中专', '大专', '不限', '本科', '硕士', '博士']

```

grouped7.mean()将会显示各组的平均值，round(1)表示小数点保留1位。[:, '1-3年']是对层次化索引的一种操作，表示选取 grouped7.mean()中索引名字为'工作经验'下'1-3年'字段的所有值。此处构造了列表xlist，值是筛选后的'最低学历'索引，xlist将用于画条形图时X轴坐标的标签文本（表示最低学历），Y轴相对应的是平均月薪。工作经验则用条形图和图例展示。

#开始画图，设置基本参数

```

matplotlib.style.use('dark_background')
fig7 = plt.figure(7, facecolor = 'black')
ax7 = fig7.add_subplot(1,1,1, facecolor='#4f4f4f', alpha=0.3)
title = plt.title(u'最低学历-工作经验-平均月薪分布图', fontsize = 18, color = 'yellow')
xlabel = plt.xlabel(u'最低学历', fontsize = 14, color = 'yellow')
ylabel = plt.ylabel(u'平均月薪', fontsize = 14, color = 'yellow')
plt.tick_params(colors='cyan')

```

#ylist1~7分别是7种条形图的Y值列表

赞同 48

21 条评论

分享

收藏


```
ylist1 = grouped7.mean().round(1)[:,'无经验'].reindex(xlist).values
ylist2 = grouped7.mean().round(1)[:,'1年以下'].reindex(xlist).values
ylist3 = grouped7.mean().round(1)[:,'不限'].reindex(xlist).values
ylist4 = grouped7.mean().round(1)[:,'1-3年'].reindex(xlist).values
ylist5 = grouped7.mean().round(1)[:,'3-5年'].reindex(xlist).values
ylist6 = grouped7.mean().round(1)[:,'5-10年'].reindex(xlist).values
ylist7 = grouped7.mean().round(1)[:,'10年以上'].reindex(xlist).values

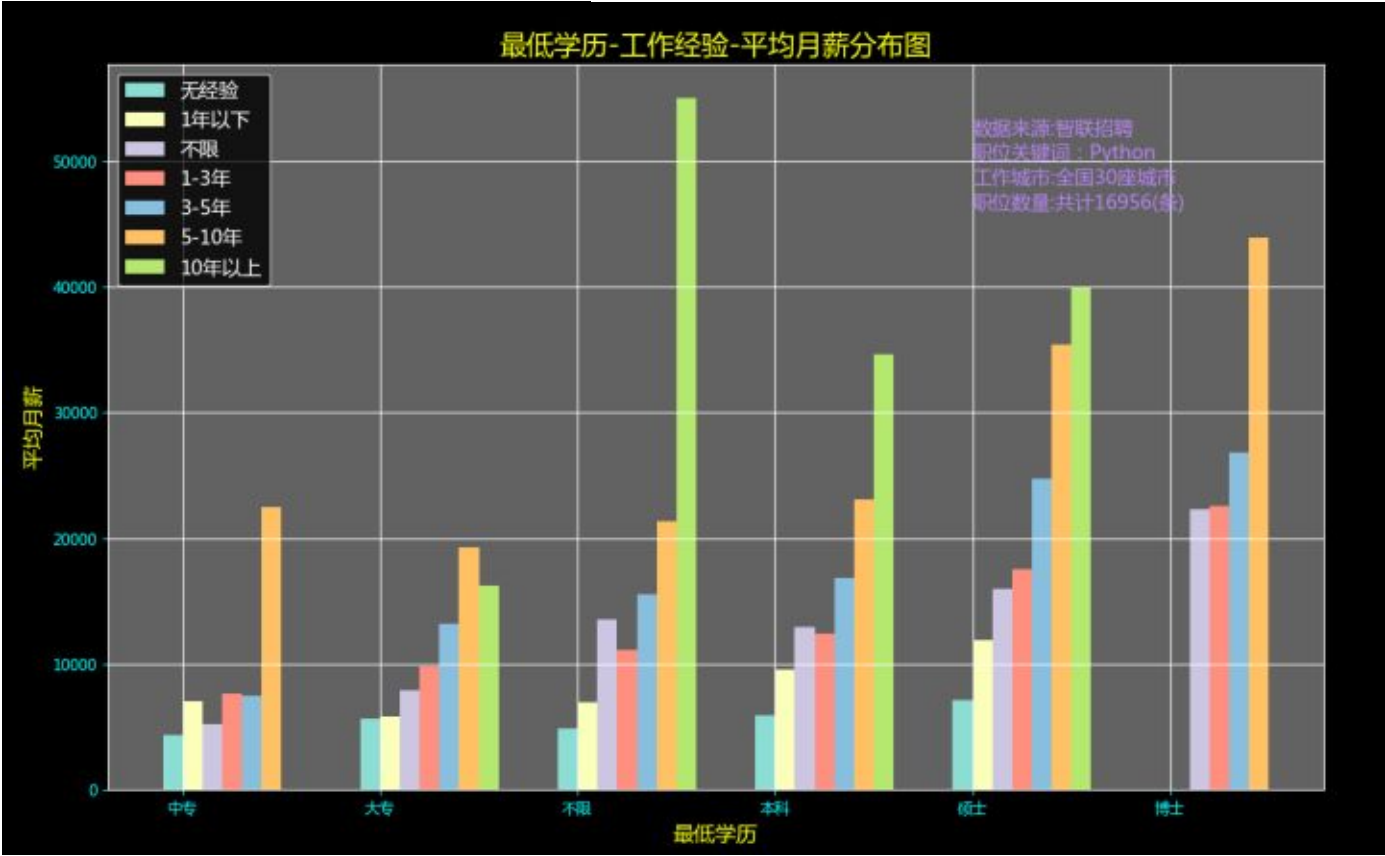
#img1~img7分别表示7种条形图
ind = np.arange(6)#ind为x轴宽度，用numpy的array形式表示
width = 0.1#条形图的宽度，要合理设置否则太宽会摆不下
img1 = ax7.bar(ind,ylist1,width)
img2 = ax7.bar(ind+width,ylist2,width)
img3 = ax7.bar(ind+width*2,ylist3,width)
img4 = ax7.bar(ind+width*3,ylist4,width)
img5 = ax7.bar(ind+width*4,ylist5,width)
img6 = ax7.bar(ind+width*5,ylist6,width)
img7 = ax7.bar(ind+width*6,ylist7,width)

#设置X轴文本和位置调整
ax7.set_xticklabels(xlist)
ax7.set_xticks(ind + width / 2)

#设置文字说明
text1 = ax7.text(4.05,52100,u'数据来源:智联招聘',fontsize=13, color='#9F79EE')
text2 = ax7.text(4.05,50200,u'职位关键词: Python',fontsize=13, color='#9F79EE')
text3 = ax7.text(4.05,48200,u'工作城市:全国30座城市',fontsize=13, color='#9F79EE')
text4 = ax7.text(4.05,46200,u'职位数量:共计16956(条)',fontsize=13, color='#9F79EE')

#设置图例
ax7.legend((img1[0],img2[0],img3[0],img4[0],img5[0],img6[0],img7[0]), (u'无经验',u'1年!
```





最后，上一张简单词云图给大家看看，用的BDP傻瓜式制作，看看就好！