# Project报告

## 小组成员及分工

11612833 尚刘方剑 Coding

11612903 赵金平 设计输入输出数据形式，设计图表; Coding

11612929 杨洽 Design, Abstract Code; Code Review

11613028 何雨京 Coding; 撰写和翻译报告

## 主程序说明和使用方法

### 说明

使用语言: Python 主要依赖:

* click: 实现相对友好的命令行操作
* xlwt: 输出数据到excel
* matplotlib: 将数据生成图表

文件结构

* main.py: main program
* requirements.txt: dependencies
* data
  + ProjectData.csv: the file to read
  + graph: the dir to store graph
  + excel: the dir to store xls file
* Model
  + \_\_init\_\_.py
  + Statistics.py: the Class includes all statistics methods
  + Student.py: the Class defines the properties of students

requirements.txt

Click==7.0  
cycler==0.10.0  
kiwisolver==1.0.1  
matplotlib==3.0.2  
numpy==1.15.4  
pyparsing==2.3.0  
python-dateutil==2.7.5  
six==1.12.0  
xlwt==1.3.0

main.py 读取csv

@click.command()  
@click.option('--file', default='data/ProjectData.csv')  
def run(file):  
 # The fields of students  
 keys = [  
 'id',  
 'name',  
 'sex',  
 'province',  
 'city',  
 'district',  
 'gaokao',  
 'sustech',  
 'gpa',  
 'dream',  
 'abroadCountry',  
 'abroadUniversity',  
 'major1',  
 'domesticCity',  
 'domesticUniversity',  
 'major2',  
 'workProvince',  
 'workCity',  
 'degree',  
 'workPlace',  
 'salary'  
 ]  
 data = {}  
 students = []  
 with open(file, newline='', encoding ='utf-8') as csv\_file:  
 csv\_reader = csv.reader(csv\_file)  
 count = 0  
 print('=== Reading Data... ===')  
 for row in csv\_reader:  
 # Skip the row of titles  
 if count != 0:  
 for i, key in enumerate(keys):  
 data[key] = row[i]  
 student = Student(data)  
 students.append(student)  
 print('ID: {}, Name: {}'.format(student.id, student.name))  
 count += 1  
 print('=== Read Process Finished, Total: {} students ==='.format(count))

### 使用方法

python main.py

可选参数

--file= : the csv file location, default='data/ProjectData.csv'

## 主要代码和思路

题目要求统计和学生相关的数据，因此我们先创建一个Student类，用来存放每个学生的数据，同时创建一个Statistics类，来放置各种统计方法，方便调用。

# encoding=utf-8  
  
class Student:  
 '''  
 Parameters:  
 data - a dict contains the data of each student  
 '''  
 def \_\_init\_\_(self, data):  
 self.id = data['id']  
 self.name = data['name']  
 self.sex = data['sex']  
 # 其他属性省略

class Statistics:  
 '''  
 Parameters:  
 students - a list of type Student  
 '''  
 def \_\_init\_\_(self, students):  
 self.students = students  
  
 # def somefunc:  
 # ...

### 数据预处理

首先不管要处理什么数据，都要先把需要的数据取出来。这里先定义一个 pluck 函数，用来取出需要字段的数据。这里使用了可变参数 \*args, 即可以传入可选个数的所需要的字段名字，来取出对应的数据。

这里还有一种情况是，我们可能只想取出某些固定值的数据进行统计，比如：只对毕业去向是“毕业工作”的同学进行统计，所以我们把输入的参数做相应的调整，如果输入的参数是字段名字符串，就取出该字段的所有数据；如果输入的参数是 {'field': ['value1', 'value2']} , 则我们只取出该字段下值符合value数组的数据。

'''  
Pluck the data of some fields  
Parameters:  
 \*fields (str,dict): the name of the fields to be pluck  
Output:  
 [  
 {field1: value},  
 {field2: value},  
 ...  
 ]  
'''  
def pluck(self, \*fields):  
 result = []  
 # Store into dict  
 # result = [{field1: value, field2: value}, {...}, ...]  
 row = {}  
 for student in self.students:  
 for field in fields:  
 expects = []  
 if isinstance(field, dict):  
 expects = list(field.values())[0]  
 field = list(field.keys())[0]  
 value = getattr(student, field).strip()  
 if expects and not value in expects:  
 break  
 row[field] = value  
 if row:  
 result.append(row.copy())  
 return result

然后我们发现数据中有一些不合理的地方，需要进行处理，来帮助我们更好的统计。我们统一把处理函数命名为 process{FieldName}, 这样做的好处在于可以在统计函数中动态调用，而不是硬编码。调用代码如下:

process = 'process{}'.format(field.capitalize()) # processId, processName, ...  
if hasattr(self, process) and callable(getattr(self, process)):  
 value = getattr(self, process)(value)

### process修正的具体方法

* processAbroadcountry 由于统计的表中内容格式不统一，如目标同为去美国留学，有的同学填写“美国”，有的同学填写“美”，这里统一将abroadCountry中的“国”字删去；
* processSallary 统一sallary单位，对于误填造成的明显不符合逻辑的月薪大于1000k的项，记录其除以1000以后的值；

### 求和部分统计

#### 思路

通过分析题目，可以发现很多的统计项，属于统计人数的项目，可以理解成不同学生相同项的求和。于是我们希望有一个通用的函数来完成所有和求和相关的统计项。 我们主要的处理思路是： 输入字段名，即可返回该字段的统计结果，返回结果用一个dict来表示，其中key是不同的字段数据，value是人数。比如统计月薪的情况，最终返回的结果可能是：

{  
 "30k": 3,  
 "16k": 20,  
 "10k": 35,  
 //...  
}

处理的过程是取得该同学月薪的数额，和结果dict进行比对，如果存在该数额的key，我们把value递增，如果不存在，就新建一个以该数额命名的key，value为0。 同时我们发现有的统计项目需要分层级计算，这样我们的统计方法可以优化，统计方法的参数同样使用 \*args 可以传入多个字段名称，然后按顺序分级，第一个字段作为父层级，第二个是子层级，以此类推。处理过程与上述类似。最终返回的结果可能如下:

{  
 "广东": {  
 "data": 3,  
 "深圳": {  
 "data": 3,  
 "南山": {"data": 2},  
 "福田": {"data": 1},  
 }  
 },  
 // ...  
}

更进一步我们发现统计项“统计毕业去向”的情况是，存在四种独立的情况“毕业工作”，“出国留学”，“国内读研”，“香港读研”，假设毕业去向是“出国留学”，则会有“留学国家”，“留学大学”，“专业1”等相关数据，而“读研学校”等其他选项相关的数据会留空。于是我们在统计过程中过滤数据为空的项，最终可以生成如下类似结果:

{  
 "出国留学": {  
 "data": 20,  
 "美": {  
 "data": 10,  
 "斯坦福": {  
 "data": 5,  
 "计算机": {"data": 2},  
 }  
 }  
 },  
 "国内读研": {  
 "data": 15,  
 "北京大学": // ...  
 }  
}

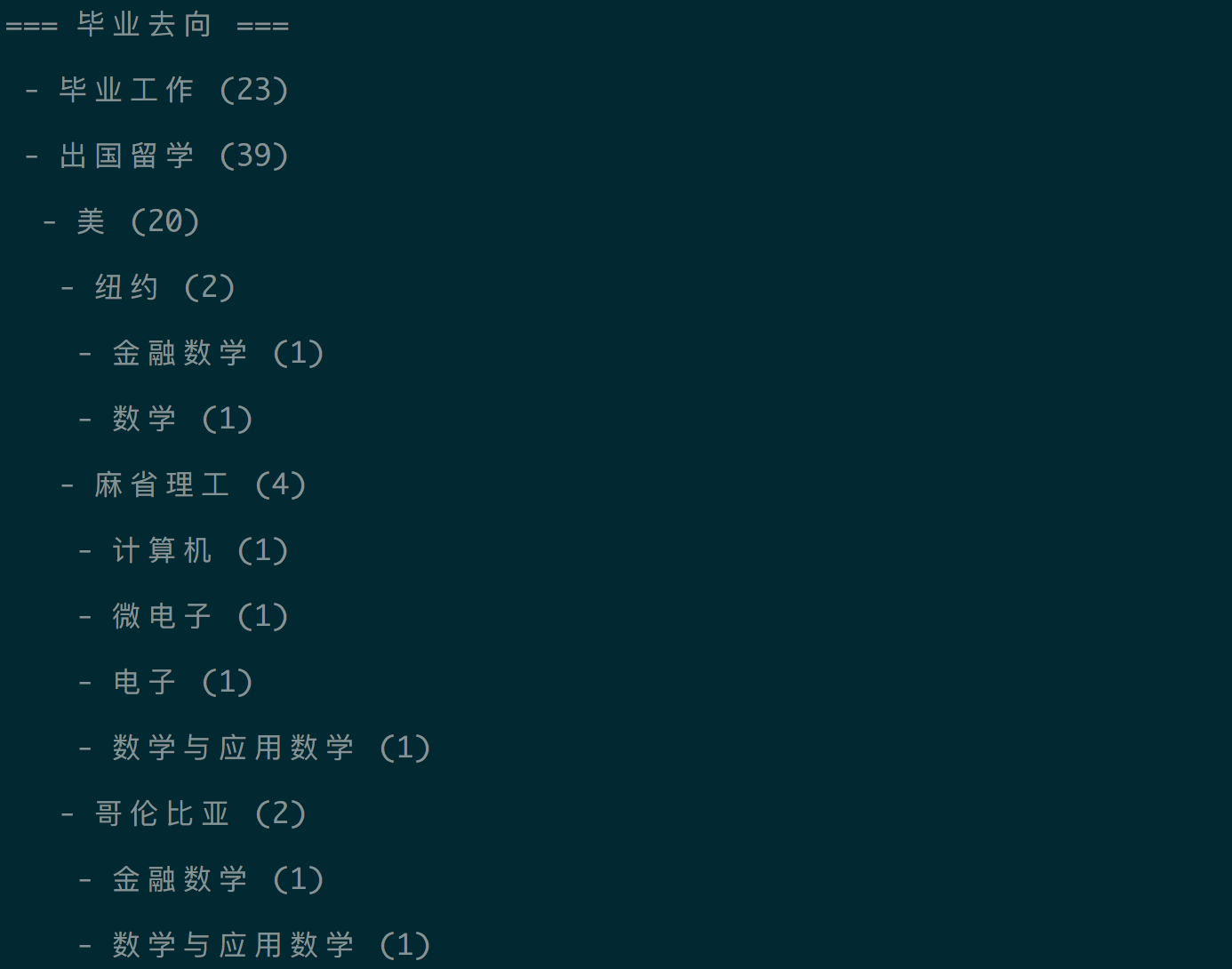
#### 代码

'''  
Parameters:  
 \*fields (str,dict): the name of the fields to be pluck  
Output:  
 [  
 {  
 field1: {  
 data: ...,  
 subfield1: { ... }  
 }  
 },  
 {  
 field2: ...  
 },  
 ...  
 ]  
'''  
def calculate(self, \*fields):  
 result = {}  
 data = self.pluck(\*fields)  
 for row in data:  
 parent = result  
 final = ''  
 last\_parent = {}  
 for field in fields:  
 if isinstance(field, dict):  
 field = list(field.keys())[0]  
 value = row[field]  
 if value != '':  
 # If there exists a process method, call it  
 process = 'process{}'.format(field.capitalize())  
 if hasattr(self, process) and callable(getattr(self, process)):  
 value = getattr(self, process)(value)  
 final = value  
 last\_parent = parent  
 if value not in parent:  
 parent[value] = {'data': 0}  
 parent[value]['data'] += 1  
 parent = parent[value]  
  
 return result

#### 运行截图

分层级统计

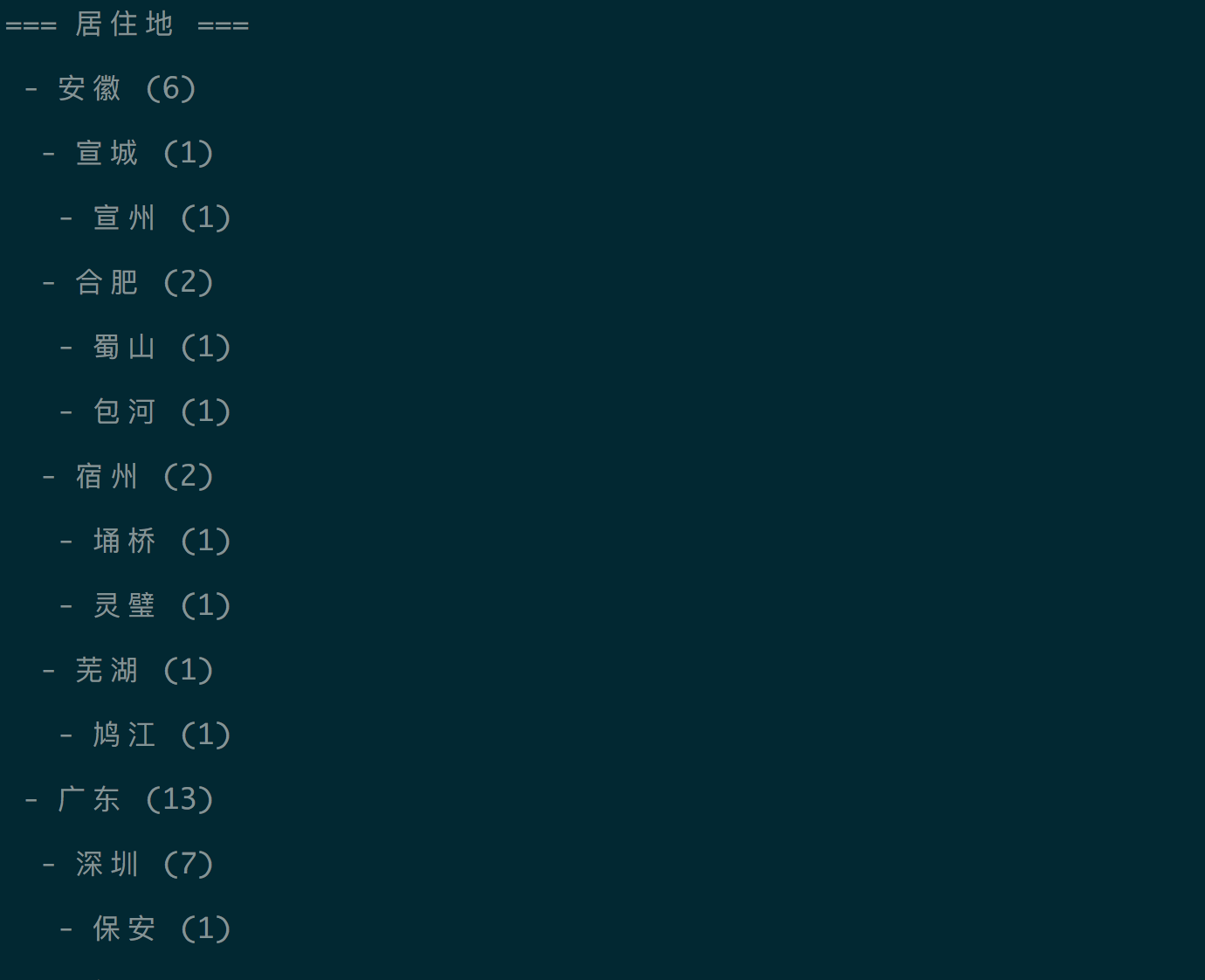
statistics.calculate(  
 'province',  
 'city',  
 'district'  
)



sum1

含并列项分层级统计

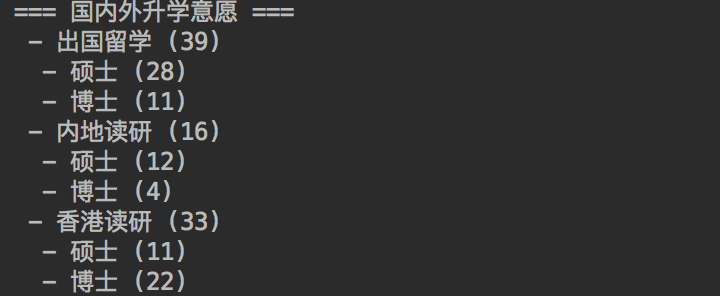
statistics.calculate(  
 'dream',  
 'abroadCountry',  
 'abroadUniversity',  
 'major1',  
 'domesticCity',  
 'domesticUniversity',  
 'major2',  
)



sum2

升学意向分层级统计

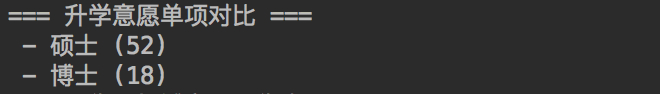
statistics.calculate(  
 {'dream': ['出国留学', '香港读研', '内地读研']},  
 'degree'  
)



sum3

升学意愿单项对比

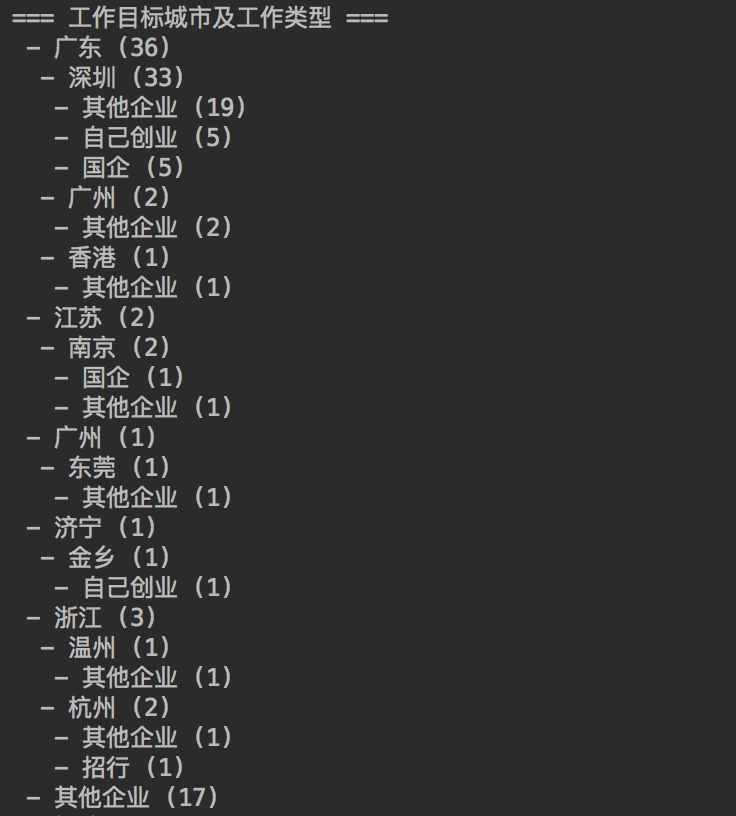
statistics.calculate('degree')



sum4

工作意向分层对比

statistics.calculate(  
 'workProvince',  
 'workCity',  
 'workPlace'  
)



png5

月薪情况

statistics.calculate('salary')



sum6

#### 代码

def processAbroadcountry(self, val):  
 return val.replace('国', '')  
  
def processSalary(self,val):  
 int\_value = int(val)  
 if int\_value>1000:  
 return str(int(int\_value/1000))  
 else :  
 return val

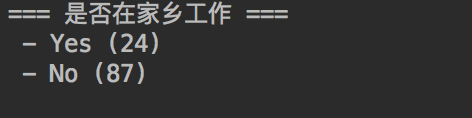
### 横向判断方法

对于判断一个同学未来是否在家乡工作的问题，我们需要在程序中进行一个横向的判断. 为了方便利用Statistics中的output函数进行输出，我们规定该方法的输出格式为 {'Yes':{'data':int},'No':{'data':int}}; 我们将需要横向对比的两项数据名称作为参数输入，利用Statistics.pluck()方法得到需要的数据，然后通过判断两个key对应的value是否相等来对结果中的int进行调整；

#### 代码

def compare(self, field1, field2):  
 result = {}  
 parent = {  
 'Yes': {'data': 0},  
 'No': {'data': 0}  
 }  
 data = self.pluck(field1, field2)  
 for row in data:  
 if row[field1] == row[field2]:  
 parent['Yes']['data'] += 1  
 else:  
 parent['No']['data'] += 1  
 result = parent  
 return result

#### 运行截图



png3

### 结果输出

我们程序中的函数返回结果基本都是如下dict嵌套的形式

{  
 "field1": {  
 "data": 2,  
 "subfield1": {  
 "data": 2  
 }  
 }  
}

在生成excel文件和输出成图表的时候，处理起来不是很方便，于是我们使用了一个将多层嵌套的dict扁平化的函数，这部分代码参考自网上，参考链接 <https://codereview.stackexchange.com/questions/173439/pythonic-way-to-flatten-nested-dictionarys>，结合我们实际情况修改得到如下代码：

def unpack(parent\_key, parent\_value):  
 try:  
 items = parent\_value.items()  
 except AttributeError:  
 # parent\_value was not a dict, no need to flatten  
 yield (parent\_key, parent\_value)  
 else:  
 for key, value in items:  
 if key != 'data':  
 yield (parent\_key + (key,), value)  
 else:  
 yield (parent\_key, value)  
  
# Flatten the multilevel dict  
def flatten(self, dictionary):  
 # Put each key into a tuple to initiate building a tuple of subkeys  
 dictionary = {(key,): value for key, value in dictionary.items()}  
 while True:  
 # Keep unpacking the dictionary until all value's are not dictionary's  
 dictionary = dict(chain.from\_iterable(starmap(unpack, dictionary.items())))  
 if not any(isinstance(value, dict) for value in dictionary.values()):  
 break  
  
 return dictionary

#### 打印到控制台

打印到控制台比较简单，这里使用一个简单的递归遍历多层dict，每一层输出的时候进行缩进。

def output(self, data, level=0):  
 for k, v in data.items():  
 if not isinstance(v, dict):  
 pass  
 else:  
 print('{} - {} ({})'.format(' ' \* level, k, v['data']))  
 self.output(v, level + 1)

#### 导出为图表

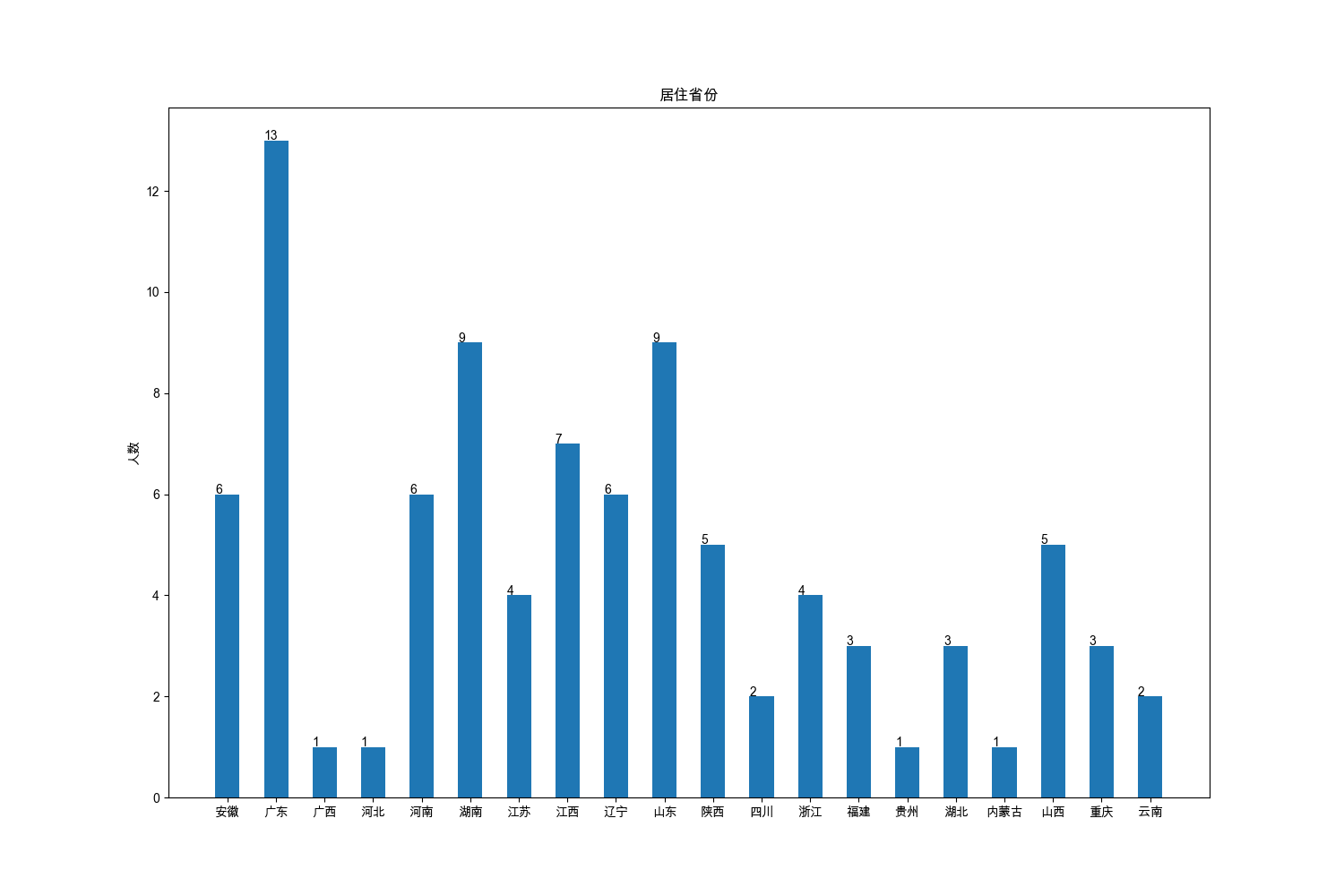
生成图表我们使用了 matplotlib 库。经过分析，我们认为绝大部分数据都可以通过条形图和饼图来表示，所以我们把生成图表的函数抽象为两个，一个用来生成条形图，一个用来生成饼图。

生成条形图的函数

'''  
Export data to a bar char  
Parameters:  
 data (dict): a non-nested dict like {field: {data: value}, ...}  
 title (str)  
 yLable (str)  
'''  
def exportToBarchart(self, data, title='', yLabel=''):  
 yData = ()  
 xLabel = ()  
 for k, v in data.items():  
 xLabel += (u'{}'.format(k),)  
 yData += (v['data'],)  
  
 f = plt.figure(figsize=(15, 10))  
 y\_pos = np.arange(len(xLabel))  
 ax = f.add\_subplot(111)  
 bars = ax.bar(y\_pos, yData, align='center', width=0.5)  
 ax.set\_xticks(y\_pos, xLabel)  
 ax.set\_ylabel(u'{}'.format(yLabel))  
 ax.set\_title(u'{}'.format(title))  
  
 for bar in bars:  
 yval = bar.get\_height()  
 ax.text(bar.get\_x(), yval + .01, yval)  
  
 t = time.time()  
 name = str(int(t))  
 f.savefig('./data/graph/{}-{}.png'.format(title.replace(' ', ''), name))

示例结果

statistics.exportToBarchart(  
 statistics.calculate('province'),  
 '居住省份',  
 '人数'  
)



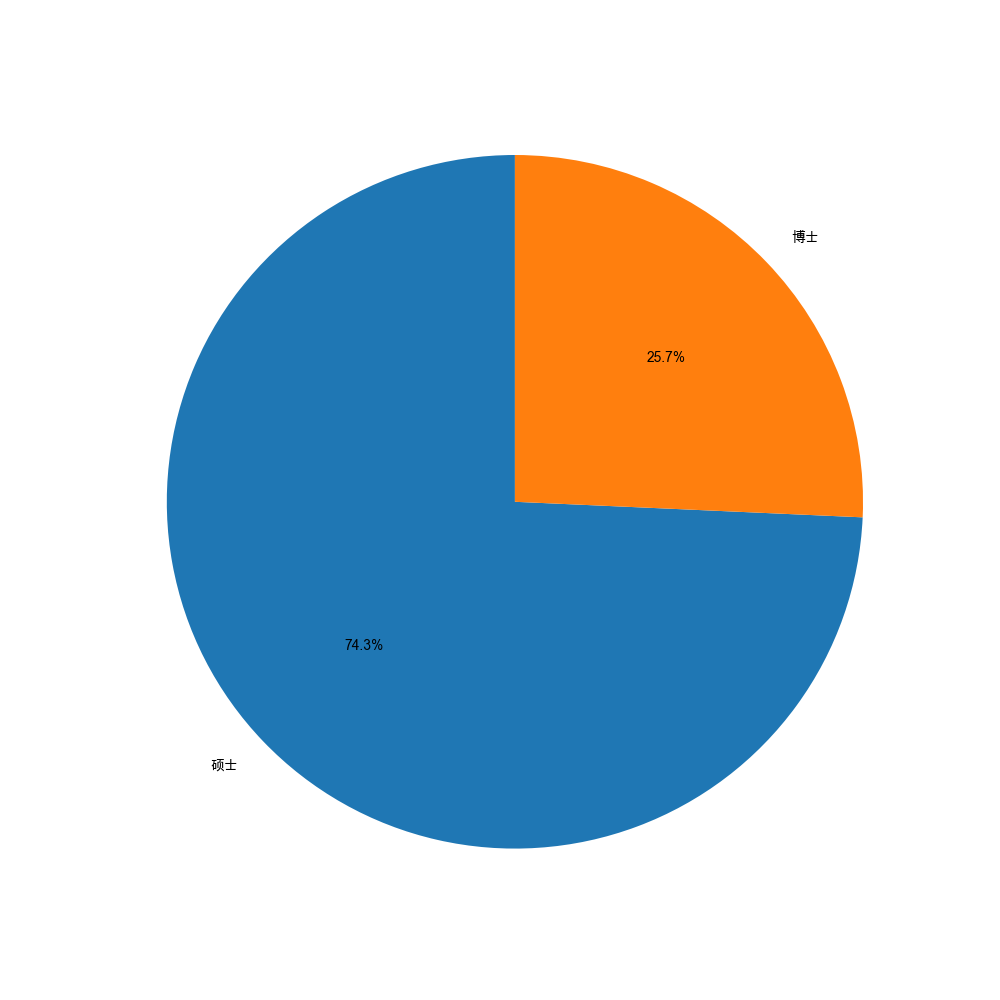
screeshot

生成饼图的函数

'''  
Export data to a pie char  
Parameters:  
 data (dict): a non-nested dict like {field: {data: value}, ...}  
 yLable (str)  
'''  
def exportToPiechart(self, data, title=''):  
 pieData = ()  
 labels = ()  
 for k, v in data.items():  
 labels += (u'{}'.format(k),)  
 pieData += (v['data'],)  
  
 f = plt.figure(figsize=(10, 10))  
 ax = f.add\_subplot(111)  
 ax.pie(pieData, labels=labels, autopct='%1.1f%%', startangle=90)  
 ax.axis('equal')  
   
 t = time.time()  
 name = str(int(t))  
 f.savefig('./data/graph/{}-{}.png'.format(title.replace(' ', ''), name))

示例结果

statistics.exportToPiechart(  
 statistics.calculate('degree'),  
 '升学意愿'  
)



screeshot