

第 5 章

实现简易平行志愿填报系统

——字典、While 循环及跳转语句、变量的作用域、模糊查询、异常处理

本章使命

解决“使用 Python 语言实现高考平行志愿录取算法”这一任务的关键处理,平行志愿填报;同时,掌握字典数据类型、由 DataFrame 对象创建字典、while 循环及跳转语句、变量的作用域等 Python 相关知识。

5.1 Excitation——提出问题

我们已经使用 Python 模拟生成了考生高考原始数据,对原始数据进行了赋分,并根据排序规则给出了考生的位次。接下来,考生通过查询院校的招生计划,就可以进行平行志愿的填报。由于我们无法获取各院校招生计划的实际数据,因此,可以参考前面的方法模拟生成各院校的招生计划数据,并存储在“招生计划.csv”文件中。

如何编写平行志愿的填报程序呢?

5.2 What——探索问题本质

关于院校的招生计划,我们直接使用模拟好的数据,包括院校代码、院校名称、专业代码、专业名称和招生人数 5 列,包含了 25 所院校、125 个专业、共计 163 条院校招生计划信息。数据存储在模拟生成的“招生计划.csv”文件中,用 Excel 打开可以看到部分招生计划数据如图 5-1 所示。

考生进行志愿填报时,首先需要查询各院校、各专业的招生计划。这个查询不是简单地把所有信息呈现出来,而是考生给出感兴趣的查询条件,程序筛选出满足条件的结果。例如,某考生想了解北京大学的各专业招生情况,输入“北京大学”,就可以获得北京大学所有招生专业及招生人数情况。由于考生往往无法准确记住院校名称或专业名称,因此还要考虑根据部分关键字进行模糊查询的功能,即能够按照给出的部分信息进行检

	A	B	C	D	E
1	院校代码	院校名称	专业代码	专业名称	招收人数
2	100001	星际探索学院	101010	太空工程	2
3	100001	星际探索学院	101011	星际航行	29
4	100001	星际探索学院	101012	外星人研究	3
5	100001	星际探索学院	101013	逻辑学	14
6	100001	星际探索学院	101014	行星地质学	7
7	100001	星际探索学院	101015	星际法律与政策	2
8	100001	星际探索学院	101016	神经重构	63
9	100002	未来医疗技术学府	101013	逻辑学	7
10	100002	未来医疗技术学府	101017	基因编辑	3

图 5-1 模拟的部分院校招生计划数据

索。例如,某考生对医疗专业感兴趣,要查询所有院校的招生计划,输入“医疗”两个字,模糊查询就可以检索出所有包含“医疗”两个字的所有专业信息。当考生已经了解了感兴趣的院校和专业的招生计划,就可以结合自身排名位次进行平行志愿填报了,每名考生可以填报多个平行志愿。

因此,本章的任务就是完成为考生提供按条件查询招生计划的功能和平行志愿填报功能,并保存填报结果。

同样利用“分而治之”的思想,将任务分为以下两个子任务:

- (1) 院校招生计划查询。
- (2) 平行志愿填报。

5.3 How——拓展求解问题必备的知识 and 能力

5.3.1 AI 助学

我们已经知道查询招生计划要实现条件检索和模糊查询功能,但是如何使用 Python 实现还无从下手。此时就可以先考虑寻求 AI 的帮助。由于院校招生数据是一个二维数据,可以用 Pandas 的 DataFrame 对象来存储。我们先看看 Pandas 是不是有办法。

我的问题: 如何对 DataFrame 按条件进行检索和模糊查询?

文心一言回答: 在 Python 的 pandas 库中,DataFrame 是一个非常强大的数据结构,它允许你按条件检索和进行模糊查询。下面是一些基本的例子来说明如何做到这一点。

1. 按条件检索

假设你有一个名为 df 的 DataFrame,并且你想找出所有 column_name 列的值大于某个特定值 value 的行。你可以使用 loc 函数或者布尔索引来实现这一点。

```
import pandas as pd
# 创建一个示例
DataFrame data = {'column_name': [1, 2, 3, 4, 5]}
df = pd.DataFrame(data)
# 定义你的条件
value = 3
```

```
#使用 loc 函数按条件检索
result = df.loc[df['column_name'] > value] print(result)
```

2. 模糊查询

对于模糊查询,你可以使用 pandas 的 `str.contains()` 方法。这个方法允许你使用正则表达式来进行模糊匹配。

```
import pandas as pd
#创建一个示例 DataFrame
data = {'column_name': ['apple', 'banana', 'cherry', 'date']}
df = pd.DataFrame(data)
#定义你的模糊查询条件
pattern = 'an'
#使用 str.contains() 进行模糊查询
result = df[df['column_name'].str.contains(pattern, case=False)]
print(result)
```

在这个例子中, `result` 将包含所有 `column_name` 列的值包含子字符串 'an'(不区分大小写)的行。注意, `str.contains()` 方法返回的是一个布尔序列,你可以用它来索引你的 `DataFrame`。

注意,这些只是基本示例,你可以根据自己的需求对它们进行扩展和修改。例如,你可以组合多个条件,或者使用更复杂的正则表达式进行模糊查询。

从 AI 的回答中,我们很容易地找到相关方法,在如何解决数据检索和模糊查询问题上,有了拓展学习的方向。

提示: 在向 AI 工具提问的过程中,你的问题越明确越聚焦,得到的答案离你想要的结果越接近,很多已经就是问题的解。此时,我们应该清醒地意识到,人类已经步入智能化时代, AI 工具已经能够帮助人类解决很多问题。5E 是人类探索未知和创新的基本过程,一个人在 `Excitation`(提出问题)和 `Exploration`(发现问题本质)阶段的能力将成为核心竞争力。

5.3.2 查询招生计划的方法

5.3.2.1 按条件筛选

我们已经学习了使用索引运算符“`[]`”筛选(访问) `DataFrame` 中数据的方法。Pandas 还提供了一种按条件对 `DataFrame` 的数据进行筛选的操作方法,具体语法如下:

```
df[筛选条件]
```

其中筛选条件的常见形式为: `df['标签']` 满足某个条件,如 `df['标签']` 等于某个值,大于某个值或小于某个值,筛选条件则返回一个布尔型 `Series`,通过 `df[布尔 Series]` 把 `DataFrame` 中对应标签,布尔值为 `True` 的数据筛选出来,得到的是 `DataFrame` 对象。

筛选条件既可以是单个条件,也可以是多个条件,多个条件之间使用 Pandas 的逻辑运算符与(`&`)、或(`|`)、非(`~`)连接。注意,每个筛选条件需要用小括号括起来。

【例 5-1】 按条件筛选 DataFrame 中的数据。

```

1  import pandas as pd
2  major_list1 = [
3      ['太空工程',2],
4      ['星际航行',29],
5      ['外星人研究',3],
6      ['逻辑学',14],
7      ['行星地质学',7],
8      ['星际法律与政策',2],
9      ['神经重构',63]
10 ]
11 df = pd.DataFrame(major_list1,columns=['专业名称','招生人数'])
12 print('--筛选条件:专业名称为逻辑学--')
13 print(df[df['专业名称']=='逻辑学'])
14 print('--筛选条件:招生人数超过 8--')
15 print(df[df['招生人数']>8])
16 print('--筛选条件:招生人数在 8~30 的专业--')
17 print(df[(df['招生人数']>=8) & (df['招生人数']<=30)])

```

在上面代码中:

第 11 行代码,创建了一个 DataFrame 对象,存储的是专业名称和招生人数。

第 13 行代码,筛选条件为“df[‘专业名称’]==‘逻辑学’”,为单个筛选条件,返回一个布尔型 Series,通过 df[布尔 Series]把筛选结果为 True(即专业名称为“逻辑学”)的数据筛选出来。

第 15 行代码,筛选条件为“df[‘招生人数’]>8”,也是单个筛选条件,返回一个布尔型 Series,通过 df[布尔 Series]把筛选结果为 True(即“招生人数”>8)的数据筛选出来。

第 17 行代码,筛选条件为“df[‘招生人数’]>=8”和“df[‘招生人数’]<=30”,为多条件筛选,多个条件同时满足,使用了逻辑与运算符“&.”进行连接。同样返回一个布尔型 Series,通过 df[布尔 Series]把筛选结果为 True(即“招生人数”>=8 并且“招生人数”<=30)的数据筛选出来。

程序运行结果如图 5-2 所示。

DataFrame 的筛选方式还有很多,比如 AI 工具给出的 df.loc[df[筛选条件]];使用 df.between(a,b) 方法筛选取值范围在[a,b]的数据,包括边界 a 和 b;使用 df.isin(ls 列表)方法可以筛选出 ls 列表列举出来的所有数据等。读者在需要时再去学习。

5.3.2.2 模糊查询

模糊查询是一种非常有用的技术,可以帮助我们快速检索出数据中是否包含某些特定字符或字符串。在 Pandas 中,可以使用 str.contains() 方法对 DataFrame 进行模糊查询。str.contains() 方法接收一个包含要搜索的关键词字符串,返回一个布尔型的 Series 对象,用于指示每个元素是否包含这个关键词。

```

--筛选条件: 专业名称为逻辑学--
专业名称  招生人数
3  逻辑学      14
--筛选条件: 招生人数超过8--
专业名称  招生人数
1  星际航行      29
3  逻辑学        14
6  神经重构      63
--筛选条件: 招生人数在8~30的专业--
专业名称  招生人数
1  星际航行      29
3  逻辑学        14

```

图 5-2 例 5-2 运行结果

【例 5-2】 使用 `str.contains()` 方法进行模糊查询。

```
1 import pandas as pd
2 major_list2= [
3     ['101010','太空工程'],
4     ['101011','星际航行'],
5     ['101012','外星人研究'],
6     ['101013','逻辑学'],
7     ['101014','行星地质学'],
8     ['101015','星际法律与政策'],
9     ['101016','神经重构'],
10    ['101017','基因编辑'],
11 ]
12 df = pd.DataFrame(major_list2,columns=['专业代码','专业名称'])
13 find_major = input('请输入你要查询的专业:')
14 df['专业名称'].str.contains(find_major)
```

在上面代码中：

第 12 行代码，创建了一个 DataFrame 对象，存储的是专业代码和专业名称。

第 13 行代码，`find_major` 接收用户输入的要查询的专业名称，用户输入的可能不正确或只包含部分信息。

第 14 行代码，使用 `str.contains()` 方法查询 `df['专业名称']` 这一列数据是否包含 `find_major`，返回一个布尔型的 Series。`str.contains()` 依次检查 `df['专业名称']` 列中的每一个数据是否包含 `find_major`，包含则返回 `True`，不包含则返回 `False`。

假设用户输入的是“星际”，程序运行结果如图 5-3 所示。由于第 2 个和第 6 个专业中包含“星际”，因此结果为 `True`。

我们可以将使用 `str.contains()` 方法模糊查询的结果作为筛选条件，对数据进行查询。

【例 5-3】 将 `str.contains()` 作为筛选方法，将筛选结果作为条件进行条件检索。

```
1 import pandas as pd
2 major_list2 = [
3     ['101010','太空工程'],
4     ['101011','星际航行'],
5     ['101012','外星人研究'],
6     ['101013','逻辑学'],
7     ['101014','行星地质学'],
8     ['101015','星际法律与政策'],
9     ['101016','神经重构'],
10    ['101017','基因编辑'],
11 ]
12 df = pd.DataFrame(major_list2,columns=['专业代码','专业名称'])
13 find_major = input('请输入你要查询的专业')
14 result = df[df['专业名称'].str.contains(find_major)]
15 if result.empty:
16     print('你查询的专业不存在!')
```

请输入你要查询的专业：星际

```
0    False
1     True
2    False
3    False
4    False
5     True
6     False
7    False
Name: 专业名称, dtype: bool
```

图 5-3 例 5-2 运行结果

```

17 else:
18     print(result)

```

在上述代码中：

第 14 行代码，result 为筛选后得到的 DataFrame 对象，筛选结果存在两种可能，一种是存在满足筛选条件的数据，即 `df['专业名称'].str.contains(find_major)` 模糊查询返回的 Series 中存在 True；另一种是不存在满足筛选条件的数据，此时 result 对象无数据。

第 15 行代码，是一个 if 分支的判断，`DataFrame.empty` 属性用来检查 DataFrame 对象数据是否为空，若空则返回 True，若不空则返回 False。

第 16 号代码，执行 True 的分值处理，输出“你查询的专业不存在！”。

第 18 行代码，执行 False 的分支处理，输出 result。

运行程序，假设输入“星际”，程序将返回专业名称中包含“星际”两个字的所有专业，程序运行结果如图 5-4 所示。

运行程序，假设输入“计算机”，程序将输出“您查询的专业不存在！”。运行结果如图 5-5 所示。

```

请输入您要查询的专业： 星际
      专业代码      专业名称
1  101011      星际航空
5  101015  星际法律与政策

```

图 5-4 例 5-3 输入“星际”的运行结果

```

请输入您要查询的专业： 计算机
您查询的专业不存在！

```

图 5-5 例 5-3 输入“计算机”的运行结果

5.3.2.3 多次查询的方法

考生进行招生专业查询时，应允许进行多次查询，即考生完成一次查询后还可以再进行下一次查询，直到考生不查询时再退出查询程序。这个过程是重复进行查询操作的过程。因此，可以用循环语句实现这个多次查询的处理。

1. while 循环

在 Python 中除了 for 循环以外，还提供了另外一种 while 循环，其语法格式如下：

```

while 循环条件：
    语句序列

```

当遇到 While 语句时，首先判断循环条件是否为 True，若为 True，则先执行语句序列，然后再去判断循环条件；否则，就退出 While 循环。

【例 5-4】 使用 while 循环计算 $1+2+\dots+100$ 。

```

1  n=100
2  i,sum=1,0      # i 和 sum 分别赋值为 1 和 0
3  while i<=100:  # 当 i<=100 成立时继续循环, 否则退出循环
4      sum=sum+i
5      i=i+1      # 注意该行也是 while 循环语句序列中的代码, 要与第 4 行有相同缩进
6  print(sum)     # 输出求和结果

```

程序运行结果如图 5-6 所示,其中 While 语句的循环条件为 $i \leq 100$,循环语句序列为 $sum = sum + i$ 和 $i = i + 1$ 。

2. break

break 语句用于跳出 for 循环或 while 循环。

提示: 对于多重循环(即循环嵌套循环),break 语句只能跳出它所在的那重循环。

【例 5-5】 输出 100 以内能整除 3 的第一个整数。

```
1 i=1
2 while i<=100:
3     if i%3==0:
4         print(i)
5         break
6     i=i+1
```

程序运行结果如图 5-7 所示。在 while 循环中,如果判断出当前的数 i 能被 3 整除($i \% 3 == 0$ 的结果为真),就将其输出;同时也不需要再判断下面的数了。因此,直接使用 break 语句退出循环即可。

5050

图 5-6 例 5-4 运行结果

3

图 5-7 例 5-5 运行结果

3. continue

continue 语句用于结束本次循环并开始下一次循环。

提示: 与 break 语句类似,对于多重循环情况,continue 语句仅作用于它所在的那重循环。

【例 5-6】 输出 100 以内能整除 3 的所有整数。

```
1 i=1
2 while i<=100:
3     if i%3!=0:
4         i=i+1
5         continue
6     print(i,end=' ')
7     i=i+1
```

程序运行结果如图 5-8 所示。在 while 循环中,如果判断出当前处理的数 i 不能整除 3($i \% 3 != 0$ 的结果为真),就直接使用 continue 继续判断下一个数;否则,这个 i 就是要找的数,将其输出。

3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57 60 63 66 69 72 75 78 81 84 87 90 93 96 99

图 5-8 例 5-6 运行结果

【例 5-7】 根据用户选择进行多次查询或退出查询(假设: 1 是继续查询,2 退出查询)。

```
1 import pandas as pd
2 major_list2 = [
3     ['101010', '太空工程'],
4     ['101011', '星际航行'],
```

```

5     ['101012', '外星人研究'],
6     ['101013', '逻辑学'],
7     ['101014', '行星地质学'],
8     ['101015', '星际法律与政策'],
9     ['101016', '神经重构'],
10    ['101017', '基因编辑'],
11 ]
12 df = pd.DataFrame(major_list2, columns=['专业代码', '专业名称'])
13 while True:
14     choose=eval(input('欢迎使用专业查询系统!输入数字进行选择:\n 1.查询专业 \n
2.退出系统 \n'))
15     if choose==1:
16         find_major = input('请输入你要查询的专业:')
17         result = df[df['专业名称'].str.contains(find_major)]
18         if result.empty:
19             print('您查询的专业不存在!')
20         else:
21             print(result)
22     elif choose==2:
23         break
24     else:
25         print('你的输入非法,请重新输入!')

```

在上述代码中,第 14~25 行代码中用 choose 接收用户的输入。若用户输入数字 1,表示用户想要查询专业,根据用户的输入进行模糊查询;若用户选择数字 2,表示用户要退出查询,直接使用 break 语句退出查询程序。

假设用户要查询包括“星际”或“计算机”的专业,程序运行结果如图 5-9 所示。用户的输入顺序依次为 1、星际、1、计算机、2。

```

欢迎使用专业查询系统! 输入数字进行选择:
1. 查询专业
2. 退出系统
1
请输入你要查询的专业: 星际
      专业代码      专业名称
1  101011      星际航行
5  101015  星际法律与政策
欢迎使用专业查询系统! 输入数字进行选择:
1. 查询专业
2. 退出系统
1
请输入你要查询的专业: 计算机
你查询的专业不存在!
欢迎使用专业查询系统! 输入数字进行选择:
1. 查询专业
2. 退出系统
2

```

图 5-9 例 5-7 运行结果

5.3.2.4 数据的快速检索

在进行志愿填报时,需要先选择院校,然后选择该院校的具体某个专业,院校和专业之间具有映射关系;有了院校名称,需要快速定位到该院校的所有专业,这涉及数据的快速检

索(查找)。通过关键字(如学校名称)快速访问到相应的数据(如专业和招生人数)是我们经常遇到的情景。Python 提供了一种能够存储数据对应关系,并能通过关键字就能够快速检索到数据的数据类型——字典。

Python 中的字典(Dictionary)的每个元素由一个键(key)和一个对应的值(value)组成,简称“键值对”。键值对的键和值之间使用冒号“:”分隔,每个键值对之间用逗号“,”分隔。字典中的键必须是唯一的,值可以是任意类型的对象,即还可以是字典。字典中的值还是字典的情况,称为字典嵌套。不包含任何元素的字典称为空字典。

1. 创建字典

创建字典时,既可以使用花括号“{}”,也可以使用 dict()方法。

【例 5-8】 创建字典。

```
1 #使用{}创建字典
2 my_dict1={'太空工程':2,'星际航行':29,'外星人研究':3}
3 print(my_dict1)
4 #使用dict函数创建字典
5 my_dict2=dict(逻辑学=14,行星地质学=7,星际法律与政策=2)
6 print(my_dict2)
7 #创建嵌套字典
8 my_dict3={
9     '星际探索学院':{'太空工程':2,'星际航行':29,'外星人研究':3},
10    '未来医疗技术学府':{'逻辑学':7,'基因编辑':3},
11    '全息艺术学院':{'光影画作':10,'虚拟雕刻':7,'全息音乐制作':96,'虚拟现实舞蹈':5}
12 }
13 print(my_dict3)
```

在上面代码中:

第2行代码,使用花括号“{}”创建了一个字典对象 my_dict1,“太空工程”和“星际航行”、“外星人研究”是字典的键;2、29、3 是字典的值。

第5行代码,使用 dict 函数创建了一个字典对象 my_dict2,“逻辑学”“行星地质学”和“星际法律与政策”是字典的键;14、7、2 是字典的值。

第8~12行代码,创建了一个嵌套的字典对象 my_dict3,外层字典中“星际探索学院”“未来医疗技术学府”和“全息艺术学院”是字典的键,内层的字典是具体的值。

程序运行结果如图 5-10 所示。

```
{'太空工程': 2, '星际航行': 29, '外星人研究': 3}
{'逻辑学': 14, '行星地质学': 7, '星际法律与政策': 2}
{'星际探索学院': {'太空工程': 2, '星际航行': 29, '外星人研究': 3}, '未来医疗技术学府': {'逻辑学': 7, '基因编辑': 3}, '全息艺术学院': {'光影画作': 10, '虚拟雕刻': 7, '全息音乐制作': 96, '虚拟现实舞蹈': 5}}
```

图 5-10 例 5-8 运行结果

2. 访问字典的值

可以通过方括号“[]”和键的方式访问字典中与键对应的值,如果键不存在,会报 KeyError 错误;也可以通过 get()方法根据给定的键获取对应的值,其语法格式如下:

```
dict.get(key,default=None)
```

get()方法是从字典 dict 中获取键为 key 的元素的值并返回。如果字典中不存在键为 key 的元素,则返回 default 参数的值(默认为 None)。

【例 5-9】 访问字典。

```
1 my_dict1={'太空工程':2,'星际航行':29,'外星人研究':3}
2 print(my_dict1['太空工程'])
3 print(my_dict1.get('星际航行'))
4 print(my_dict1.get('基因编辑'))
```

程序运行结果如图 5-11 所示。由于字典 my_dict1 中不存在“基因编辑”这一键,因此 my_dict1.get('基因编辑')返回的是 None。

```
2
29
None
```

图 5-11 例 5-9 运行结果

3. 修改字典的值

通过键才能修改字典中的值,如果该键在字典中已存在,则会将该键对应的值修改;如果该键在字典中不存在,则会在字典中插入一个新元素。

假设 dictionary 是一个字典,修改字典的值的语法格式如下:

```
dictionary[键]=新值
```

【例 5-10】 修改字典的值。

```
1 my_dict2=dict(逻辑学=14,行星地质学=7,星际法律与政策=2)
2 #修改字典中逻辑学对应的值
3 my_dict2['逻辑学']=17
4 print(my_dict2)
5 #字典中不存在键为神经重构的元素,所以会插入一个新元素
6 my_dict2['神经重构']=63
7 print(my_dict2)
```

程序运行结果如图 5-12 所示。

```
{'逻辑学': 17, '行星地质学': 7, '星际法律与政策': 2}
{'逻辑学': 17, '行星地质学': 7, '星际法律与政策': 2, '神经重构': 63}
```

图 5-12 例 5-10 运行结果

4. 获取字典的键、值和键值对

假设 dictionary 是一个字典,可以获取字典中所有的键、所有的值和所有的键值对。

(1) 使用字典中的 keys()方法可以获取字典所有的键。keys()方法的语法格式如下:

```
dictionary.keys()
```

(2) 使用字典中的 values()方法可以获取字典所有的值。values()的语法格式如下:

```
dictionary.values()
```

(3) 使用字典中的 items()方法可以返回一个可按(键,值)方式遍历的对象。items()方法的语法格式如下:

```
dictionary.items()
```