实验课 09

实验9-1 CSV文件的读写

CSV的英语全称为 Comma-Separated Values，即逗号分隔值。其文件以纯文本形式存储表格数据，由任意数目的记录组成，记录之间通过换行符分隔；每条记录由字段构成，字段间的分隔符最常见的是逗号或者是制表符。

在前面的实验练习中我们曾经简单接触过CSV文件，并使用传统的Python文件处理方法对CSV文件进行相关处理。不过当时我们读取文件的一行后先要去掉行末的换行符，再将一行字符串通过逗号分割成由多个元素构成的列表。

Python中提供了CSV库，可以让我们更加方便地处理CSV文件。

步骤1 读取CSV文件

我们以鸢尾花数据集`iris.csv`为例，学习如何利用CSV库读取该文件，并显示每一行数据。

CSV库是Python自带的库，无需手动安装。

`reader`方法用于读取CSV文件，其参数为一个文件描述符，也就是说我们首先需要利用`open`方法打开这个文件，并将该文件对象作为`reader`方法的实参。

```
import csv

with open('data/iris.csv','r') as f:
    csv_file = csv.reader(f)
    for line in csv_file:
        print(line)
```

可以看到从第1行开始每一行都会返回一个列表，列表中的元素就是一个个已经分割好的字段，这样省去了我们原来用`split`方法这一步骤。

请注意：列表中的元素全部都是str类型，若要进行相关数学运算，请不要忘记将相应元素转换为int型或float型再使用。

使用CSV库的另一好处就是对含有字符串的字段有着良好的分割效果，请看以下例子。

假如我们有一个csv文件，每一行的内容分别为用户ID，商品ID，用户对商品的评分，用户对商品的评价。其中用户评价中避免不了含有标点符号（肯定含有逗号）。例如有一行内容如下：

```
12478,13694,3,"I don't like this bag, it's too big for me!"
```

为了不产生分割上的歧义，我们将用户评论以双引号包裹，评论内部的逗号只是用户实际的评论字符，并不是字段的分隔符。

如果我们用传统的`split`方法，Python会将`bag, it's`之间的逗号作为分隔符，把原来的用户评价分成了两个元素，并且句子的前后双引号仍旧保留，即

```
['12478', '13694', '3', '"I don\'t like this bag', ' it\'s too big for me!"']
```

而使用CSV库的情况下，双引号被认为是**一个整体**，其内部的逗号不具有分隔符的含义，在读取记录并返回列表时并不会对双引号内部的逗号进行分割，并且在返回的列表中会自动去除双引号，即

```
['12478', '13694', '3', "I don't like this bag, it's too big for me!"]
```

步骤2 处理“方言”

有的时候我们获取到的CSV文件中并不是以逗号作为分隔符，有可能以管道符`|`；或者是冒号`:`；或者是双冒号`::`等等。我们称这种CSV文件为CSV的方言。

假设我们有一个csv文件，各字段之间以管道符分隔，内容如下：

```
user_id|item_id|rank|comment
12348|2247|1.5|"I don't like this bag, it's too big for me!"
12345|3321|4.5|"I like it !"
12346|3320|3.0|"Just so so."
```

对于这种以非逗号分隔的CSV文件，我们可以使用`register_dialect`方法注册一个新的方言，然后在使用`reader`方法时通过`delimiter`参数指定方言。

```
import csv

csv.register_dialect('pipe',delimiter='|')
with open('data/dialect.csv','r') as f:
    csv_file = csv.reader(f,dialect='pipe')
    for line in csv_file:
        print(line)
```

```
['user_id', 'item_id', 'rank', 'comment']
['12348', '2247', '1.5', "I don't like this bag, it's too big for me!"]
['12345', '3321', '4.5', 'I like it !']
['12346', '3320', '3.0', 'Just so so.']
```

步骤3 将读取的结果转换为字典

如果一个CSV文件含有大量的字段，那么想确认一条记录中的某一个数据属于第几个字段是很麻烦的事情。CSV模块提供了以字典结构返回数据的方式，即使用CSV模块中的`DictReader`方法。

```
import csv

csv.register_dialect('pipe',delimiter='|')
with open('data/dialect.csv','r') as f:
    csv_file = csv.DictReader(f,dialect='pipe')
    for line in csv_file:
        print(line)
```

```
{'user_id': '12348', 'item_id': '2247', 'rank': '1.5', 'comment': "I don't like
this bag, it's too big for me!"}
{'user_id': '12345', 'item_id': '3321', 'rank': '4.5', 'comment': 'I like it !'}
{'user_id': '12346', 'item_id': '3320', 'rank': '3.0', 'comment': 'Just so so.'}
```

步骤4 创建CSV文件

以前我们写入文件时，将所有待写入的数据转换成字符串类型，再拼接成一个大字符串通过`write`方法写入文件中。

现在我们使用CSV库就可以直接把待写入的记录通过`writerow`方法逐行地写入。

例如我们现在将2名学生的学号、语数英成绩以及教师评语写入到CSV文件中。

```
import csv

stu_1 = ['202001', 72, 92.5, 85, 'Need to keep working hard for chinese.']
stu_2 = ['202002', 91, 90.5, 89, 'Good, keep going!']
stu_list = [stu_1, stu_2]

with open('stuGrade2.csv','w') as f:
    writer = csv.writer(f)
    writer.writerow(['id','chinese','math','english','teacher_comment'])
    for stu in stu_list:
        writer.writerow(stu)
```

实验9-2 Pandas库初步（上）

Pandas是Python编程语言的基于NumPy的用于数据操纵和分析的软件库。它特别提供操纵数值表格和时间序列的数据结构和运算操作。

它的名字衍生自术语"面板数据"（panel data），这是计量经济学的数据集术语，它们包括了对同一个体的在多个时期上的观测。

当然Pandas和'熊猫'并没有什么联系.....

步骤1 安装Pandas

Pandas是第三方提供的自由软件库，我们在使用之前首先要安装Pandas库。

在Anaconda Powershell Prompt命令行中切换到你的环境（若有），输入`conda install pandas`后回车。

步骤2 Series对象

Pandas的Series对象是一个带索引数据构成的一维数组，我们可以用一个普通的一维数组创建一个Series对象。

```
import pandas as pd

data_1 = pd.Series([0.75,0.25,0.5,1.0])
print(data_1)
```

```
0    0.75
1    0.25
2    0.50
3    1.00
dtype: float64
```

我们可以看到Series对象包含一组数据和一组索引，我们可以通过`values`属性和`index`属性分别获取数据和索引。

```
print(data_1.values)
```

```
[0.75 0.25 0.5  1.  ]
```

```
print(data_1.index)
```

```
RangeIndex(start=0, stop=4, step=1)
```

和NumPy数组一样，我们可以通过中括号内引用下标获取某个（或通过切片获取多个）元素。

```
print(data_1[0])
```

```
0.75
```

```
print(data_1[1:3])
```

```
1    0.25
2    0.50
dtype: float64
```

你可能觉得这和NumPy数组有什么不同？请看下例，我们将显式定义数组的索引，区别于传统的默认从0开始的索引：

```
data_2 = pd.Series([1.38,2.45,3.56,4.78],index=['a','b','c','d'])
print(data_2)
```

```
a    1.38
b    2.45
c    3.56
d    4.78
dtype: float64
```

你会发现`data_2`Series对象的索引不再是默认的从0开始的递增序列，现在我们可以通过显式定义的索引调用Series的元素。

```
print(data_2['b'])
```

```
2.45
```

```
print(data_2['b':'d'])  # 显式定义字符索引后切片操作不再满足左闭右开的原则
```

```
b    2.45
c    3.56
d    4.78
dtype: float64
```

当然我们还可以继续使用传统的数组下标方式调用，如果显式定义的索引中含有整型数据，则仅能调用曾经定义的索引（切片仍可以使用隐式整数索引）。

```
print(data_2[0])
```

```
1.38
```

```
data_3 = pd.Series([100,200,300,400],index=['aa',2,'bb','cc'])
print(data_3[2])
```

```
200
```

```
print(data_3[0])  # 错误示范!
```

```
print(data_3[2:4])
```

```
bb    300
cc    400
dtype: int64
```

可能你被这显式隐式的索引搞晕了……幸好Pandas提供了一些**索引器**帮助我们准确地选取某个（某些）数据。

我们现在创建两个Series对象`data_4`和`data_5`。

```
data_4 = pd.Series([100,200,300,400],index=['a','b','c','d'])
data_5 = pd.Series([500,600,700,800],index=[2,4,6,8])
print(data_4)
print('++++++++++++++++++')
print(data_5)
```

```
a    100
b    200
c    300
d    400
dtype: int64
++++++++++++++++++++
2    500
4    600
6    700
8    800
dtype: int64
```

第一种索引器是`loc`属性，表示取值和切片都是显式（我们自定义）的：

```
print(data_4.loc['a'])
```

```
100
```

```
print(data_4.loc['a':'c'])
```

```
a    100
b    200
c    300
dtype: int64
```

```
print(data_5.loc[2])
```

```
500
```

```
# 错误示范!!!
print(data_5.loc[3])
```

第二种索引器是`iloc`属性，表示取值和切片都是隐式（类似传统数组）的：

```
print(data_4.iloc[0])
```

```
100
```

```
print(data_4.iloc[1:3])
```

```
b    200
c    300
dtype: int64
```

```
print(data_5.iloc[2])  # 取传统数组意义上的第2个元素（700），不是取自定义下标为2的元素（500）
```

```
700
```

最后要说的是，使用字典创建一个Series对象是最完美的方法！

```
city_weather = {'Beijing':7.2, 'Tianjin':7.5, 'Shanghai':13.2, 'Chongqing':15.7}
data_6 = pd.Series(city_weather)
print(data_6)
```

```
Beijing       7.2
Tianjin       7.5
Shanghai     13.2
Chongqing    15.7
dtype: float64
```

步骤3 DataFrame对象

Pandas的另一个基础数据结构是DataFrame。

如果将Series类比成带有自定义索引的一维数组，那么DataFrame就可以类比为既有灵活的行索引，又有灵活列索引的二维数组。在C语言中，我们把二维数组看成由多个一维数组组成的数组；那么在Pandas中我们可以把DataFrame看作有序排列的若干Series对象。

现在我们由两个Series对象：我国四个直辖市的人口（万人，数据来自第六次人口普查）及面积（平方千米），创建一个DataFrame对象。

```
population_dict = {'Beijing':1961.2, 'Tianjin':1293.9, 'Shanghai':2301.9,
'Chongqing':2884.6}
area_dict = {'Beijing':16406, 'Tianjin':11917, 'Shanghai':8359,
'Chongqing':82374}
pop_area_data = pd.DataFrame({'Population':population_dict, 'Area':area_dict})
print(pop_area_data)
```

```
           Population   Area
Beijing        1961.2  16406
Tianjin        1293.9  11917
Shanghai       2301.9   8359
Chongqing      2884.6  82374
```

我们可以通过`index`属性获取行索引标签：

```
print(pop_area_data.index)
```

```
Index(['Beijing', 'Tianjin', 'Shanghai', 'Chongqing'], dtype='object')
```

可以通过`columns`属性获取列索引标签：

```
print(pop_area_data.columns)
```

```
Index(['Population', 'Area'], dtype='object')
```

我们可以像操作Python中字典那样在DataFrame对象中添加列，例如我们通过人口和面积计算出四个城市的人口密度。

```
pop_area_data['Density'] = pop_area_data['Population'] / pop_area_data['Area']
print(pop_area_data)
```

```
           Population   Area   Density
Beijing        1961.2  16406  0.119542
Tianjin        1293.9  11917  0.108576
Shanghai       2301.9   8359  0.275380
Chongqing      2884.6  82374  0.035018
```

由于我们创建DataFrame对象的时候是通过两个Series对象创建的，而这两个Series对象构成了DataFrame对象的两列。所以与我们以前惯例不同，通过DataFrame变量加中括号只能获取列。

```
print(pop_area_data['Area'])
```

```
Beijing      16406
Tianjin      11917
Shanghai      8359
Chongqing    82374
Name: Area, dtype: int64
```

```
# 错误示范
print(pop_area_data['Beijing'])
```

如果我们想获取某一行内容，一种方法是通过索引器像操作传统二维数组那样对DataFrame对象进行获取，例如我们想获取上海市的信息。

```
print(pop_area_data.loc['Shanghai'])  # 显式索引
```

```
Population    2301.90000
Area          8359.00000
Density          0.27538
Name: Shanghai, dtype: float64
```

```
print(pop_area_data.iloc[2,0:2])  # 隐式索引，只获取前两列内容
```

```
Population    2301.9
Area          8359.0
Name: Shanghai, dtype: float64
```

我们可以通过二维NumPy数组创建一个DataFrame对象，当然最好创建时就指定行索引和列索引。

```
import numpy as np
arr = np.random.randn(3,2)
data_7 = pd.DataFrame(arr,index=['Row1','Row2','Row3'],columns=['ColA','ColB'])
print(data_7)
```

```
          ColA      ColB
Row1 -1.006579  1.289408
Row2 -1.226821  2.188305
Row3  0.471258 -0.740075
```

当然也可以将DataFrame对象转换为二维NumPy数组（这样取出一行元素就方便多了）。

```
arr_2 = data_7.values
print(type(arr_2))
print(arr_2)
```

```
<class 'numpy.ndarray'>
[[-1.00657933  1.28940761]
 [-1.22682106  2.18830503]
 [ 0.4712584  -0.74007516]]
```

步骤4 从文件中读取数据

Pandas可以从csv或者Excel文件中读取数据，并转换DataFrame对象。

我们首先读取鸢尾花数据集的csv文件，该csv文件的第1行为列名，可以直接使用`read_csv`方法。

```
iris_data = pd.read_csv('data/iris.csv')
print(iris_data)
```

```
     Sepal_Length  Sepal_Width  Petal_Length  Petal_Width    Species
0             5.1          3.5           1.4          0.2     setosa
1             4.9          3.0           1.4          0.2     setosa
2             4.7          3.2           1.3          0.2     setosa
3             4.6          3.1           1.5          0.2     setosa
4             5.0          3.6           1.4          0.2     setosa
..            ...          ...           ...          ...        ...
145           6.7          3.0           5.2          2.3  virginica
146           6.3          2.5           5.0          1.9  virginica
147           6.5          3.0           5.2          2.0  virginica
148           6.2          3.4           5.4          2.3  virginica
149           5.9          3.0           5.1          1.8  virginica

[150 rows x 5 columns]
```

我们也可以通过`index_col`参数指定某一列的内容作为行索引，例如学生成绩csv中我们指定`stuID`作为行索引。

```
stu_data = pd.read_csv('data/stuGrade.csv',index_col='stuID')
print(stu_data)
```

```
       chinese  math  english
stuID
10381       74    87       82
10382       92    88       90
10383       87    82       81
10384       98    97       96
10385       62    51       60
```

接下来我们读取香港京士柏每日日照小时数的csv文件（已去除文件最后3行的注释内容），该csv文件的第3行为列名，我们需要指定`header`参数，将第3行内容作为列名。

```
sun_data = pd.read_csv('data/daily_KP_SUN_2020-Copy1.csv',header=2)  # 从0行开始计数
print(sun_data)
```

```
     年/Year  月/Month  日/Day  数值/Value 數據完整性/data Completeness
0      2020        1      1       0.2                        C
1      2020        1      2       3.6                        C
2      2020        1      3       8.9                        C
3      2020        1      4       8.6                        C
4      2020        1      5       6.6                        C
..      ...      ...    ...       ...                      ...
269    2020        9     26       2.4                        C
270    2020        9     27       2.7                        C
271    2020        9     28       0.1                        C
272    2020        9     29       2.1                        C
273    2020        9     30       4.2                        C

[274 rows x 5 columns]
```

如果读取的csv文件没有列名，我们需要手动设置`names`参数，该参数接收一个列表，为列索引内容。

读取Excel文件的方法为`read_excel()`，Excel文件称为**工作簿**里面包含多张**工作表**，Pandas默认读取`Sheet1`工作表，如果要指定读取的工作表，需要指定`sheet_name`参数。

步骤5 将数据保存到文件中

我们可以通过`to_csv`方法将DataFrame对象保存为csv文件。

例如我们把`iris_data`的前50条数据保存为csv文件，由于读取数据集文件时并没有指定某列元素作为行索引，Pandas默认以从0开始的序列作为DataFrame对象的行索引，我们在保存csv文件时并不希望写入没有意义的行索引，需要指定`index = 0`.

```
data_8 = iris_data.iloc[:50]
data_8.to_csv('ttt.csv',index=0)
```

你也可以通过`columns`参数指定输出的列，设置`header=0`参数指定不输出列索引等功能。