第5章

数据分析与可视化

本章概要

当获得数据后，如果想从数据中挖掘出一些关键信息，就先要了解这份数据，即本节将要介绍的数据分析。在 Python 语言中，Pandas 是高性能计算和数据分析的基础包，在数据工程中已得到广泛应用。

而从数据中提取出来有吸引力的图表更是一件非常重要的事情，这就是数据可视化的工作。数据可视化是一个非常直观的查看数据手段。Matplotlib 是 Python 主要的科学绘图库，其功能为生成可发布的可视化内容，如折线图、直方图、散点图等。将数据及各种分析可视化，可以让用户产生深刻的理解。

本章将根据数据分析流程，依次介绍数据获取、数据预处理、数据分析的操作方法，并借助 Python 的 NumPy、Pandas、Matplotlib 等模块所提供的计算、分析、统计和绘图功能，来总体介绍实用数据分析和数据可视化方法。

学习目标

通过本章的学习，要求达到以下目标：

(1) 了解数据分析的目标。

(2) 掌握 Pandas 数据分析模块。

(3) 掌握 DataFrame 数据结构的使用。

(4) 掌握 Matplotlib 的数据可视化方法。

(5) 理解 Seaborn 高级数据可视化手段。

5.1 Python 数据分析

Pandas(Python Data Analysis Library)是 Python 的一个数据分析包，是基于 NumPy 的一种工具，为了解决数据分析任务而创建。

Pandas 使用强大的数据结构提供高性能的数据操作和分析工具。模块提供了大量的能便捷处理数据的函数、方法和模型，还包括操作大型数据集的工具，从而高效分析数据。

Pandas 主要处理以下三种数据结构。

(1) Series：一维数组，与 NumPy 中一维的 ndarray 类似。数据结构接近 Python 中的 List 列表，数据元素可以是不同的数据类型。

(2) DataFrame：二维数据结构。DataFrame 可以理解成 Series 的容器，其内部的每项元素都可以看作一个 Series。DataFrame 是重要的数据结构，在机器学习中经常使用。

(3) Panel：三维数组，可以理解为 DataFrame 的容器，其内部的每项元素都可以看作一个 DataFrame。

这些数据结构都是构建在 NumPy 数组的基础之上，运算速度很快。

5.1.1　Series 数据结构

Series 是一种类似于一维数组的对象，由一组数据以及一组与之相关的数据标签(即索引)组成，数据可以是任何 NumPy 数据类型(如整数、字符串、浮点数、Python 对象等)。

1. 创建 Series 对象

创建 Series 对象可以使用函数 Series(data, index)，参数 data 表示数据值，index 是索引，默认情况下会自动创建一个 0～$N-1$(N 为数据的长度)的整数型索引。访问 Series 对象的成员的方法类似 narray 数组，使用索引或索引名进行访问。

例 5-1：创建一个 Series 对象。

```
import pandas as pd
s = pd.Series([1,3,5,9,6,8])
print(s)
```

例 5-2：为一个地理位置数据创建 Series 对象。

```
import pandas as pd
#使用列表创建,索引值为默认值
print('--------   列表创建 series   ----------')
s1=pd.Series([1,1,1,1,1])
print(s1)
print('--------   字典创建 series   ----------')
#使用字典创建,索引值为字典的 key 值
s2=pd.Series({'Longitude':39,'Latitude':116,'Temperature':23})
print('First value in s2:',s2['Longitude'])
print('--------- 用序列作 series 索引 ----------')
#使用 range()函数生成的迭代序列设置索引值
s3=pd.Series([3.4,0.8,2.1,0.3,1.5],range(5,10))
print('First value in s3:',s3[5])
```

2. 访问 Series 数据对象

1) 修改数据

可以通过赋值操作直接修改 Series 对象成员的值，还可以为多个对象成员批量修改数据。

例 5-3：对例 5-2 中创建的 s2，将温度增加 2，设置城市为“Beijing”。

```
#温度增加 2 度,设置城市为 Beijing
```

```
s2["City"]="Beijing"
s2['Temperature']+=2
s2
```

运行结果如图 5-1 所示。

2）按条件表达式筛选数据

例 5-4：找出例 5-2 的 s3 中大于 2 的数据。

```
s3[s3>2]
```

输出结果如图 5-2 所示。

```
Longitude          39
Latitude          116
Temperature        25
City          Beijing
dtype: object
```

图 5-1　例 5-3 运行结果

```
5    3.4
7    2.1
dtype: float64
```

图 5-2　例 5-4 输出结果

3）增加对象成员

两个 Series 对象可以通过 append()函数进行拼接，从而产生一个新的 Series 对象。进行拼接操作时，原来的 Series 对象内容保持不变。

例 5-5：为 s2 添加一项湿度数据。

```
stiny=pd.Series({'humidity':84})
s4=s2.append(stiny)
print('--------原 Series: --------\n',s2)
print('--------新 Series: --------\n',s4)
```

输出结果如图 5-3 所示。

```
-------原 Series: -------
Longitude          39
Latitude          116
Temperature        25
City          Beijing
dtype: object
-------新 Series: -------
Longitude          39
Latitude          116
Temperature        25
City          Beijing
humidity           84
dtype: object
```

图 5-3　例 5-5 输出结果

可以看到，合并操作不影响原 Series。结果中原 s2 数据没有变化，新创建的 s4 对象接收了合并后的新数据。

4）删除对象成员

可以通过 drop()函数删除对象成员，可以删除一个或多个对象成员。与 append()函数

一样，drop()函数也不改变原对象的内容，返回一个新的 Series 对象。

例 5-6：删除重量数据。

```
s2=s2.drop('City')
s2
```

输出结果如图 5-4 所示。

```
Longitude          39
Latitude          116
Temperature        25
dtype: object
```

图 5-4　例 5-6 输出结果

5.1.2　DataFrame 对象

DataFrame 是一个表格型的数据结构，包含一组有序数列。列索引(columns)对应表格的字段名，行索引(index)对应表格的行号，值(values)是一个二维数组。每一列表示一个独立的属性，各个列的数据类型(数值、字符串、布尔值等)可以不同。

DataFrame 既有行索引也有列索引，所以 DataFrame 也可以看成是 Series 的容器。

1. 创建 DataFrame 对象

构建 DataFrame 的办法有很多，基本方法是使用 DataFrame()函数构造，格式如下。

```
DataFrame([data, index, columns, dtype, copy])
```

1) 从字典构建 DataFrame

例 5-7：从字典数据创建 DataFrame。

```
import pandas as pd
dict1 = {'col1':[1,2,5,7],'col2':['a','b','c','d']}
df = pd.DataFrame(dict1)
df
```

运行结果如图 5-5 所示。

	col1	col2
0	1	a
1	2	b
2	5	c
3	7	d

图 5-5　例 5-7 运行结果

例 5-8：由列表组成 DataFrame。

```
lista = [1,2,5,7]
listb = ['a','b','c','d']
df = pd.DataFrame({'col1':lista,'col2':listb})
df
```

2）从数组创建 DataFrame

可以使用 Python 的二维数组作为数值，通过 columns 参数指定列名，构建 DataFrame。

例 5-9：二维数组和 columns 构建 DataFrame。

```
import pandas as pd
a = pd.DataFrame([[1,0.1,5],
                  [2,0.5,6],
                  [4,0.8,5]],columns = ["t1", "t2", "p1"])
a
```

运行结果如图 5-6 所示。

也可以从 NumPy 提供的 ndarray 结构创建 DataFrame。

例 5-10：从二维 ndarray 创建 DataFrame。

```
a = np.array([[1,2,3], [4,5,6],[7,8,9]])
b=pd.DataFrame(a)
b
```

运行结果如图 5-7 所示。

	t1	t2	p1
0	1	0.1	5
1	2	0.5	6
2	4	0.8	5

图 5-6　例 5-9 运行结果

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

图 5-7　例 5-10 运行结果

3）从 csv 文件中读取数据到 DataFrame

通过第 4 章内容我们了解到，Pandas 还提供读写 csv 文件功能。例如，read_csv()函数可以读取 csv 文件的数据，返回 DataFrame 对象。

2. 访问 DataFrame 对象

从本质上来说，Series 或 DataFrame 的索引是一个 Index 对象，负责管理轴标签等。在构建 Series 或 DataFrame 时，所使用的数组或序列的标签会转换成索引对象。因此，Series 的索引不只是数字，也包括字符等。对 DataFrame 进行索引，可以获取其中的一个或多个列，列可以通过索引进行访问。

例 5-11：对 Series 和 DataFrame 进行索引。

```
import numpy as np
import pandas as pd
ser=pd.Series(np.arange(4),index=['A','B','C','D']) #构造一个 Series
data=pd.DataFrame(np.arange(16).reshape(4,4),
                  index=['BJ','SH','GZ','SZ'],
                  columns=['q','r','s','t'])
print("ser['C']:",ser['C'])
print("ser[2]:",ser[2])
print("data['q']:",data['q'])
print("data[['q','t']]:",data[['q','t']])
```

所构成的二维 DataFrame 数组 data 的内容如图 5-8 所示。

运行结果如图 5-9 所示。

	q	r	s	t
BJ	0	1	2	3
SH	4	5	6	7
GZ	8	9	10	11
SZ	12	13	14	15

图 5-8　二维数组 data

```
ser['C']: 2
ser[2]: 2
data['q']: BJ     0
SH     4
GZ     8
SZ    12
Name: q, dtype: int32
data[['q','t']]:      q   t
BJ   0   3
SH   4   7
GZ   8  11
SZ  12  15
```

图 5-9　例 5-11 运行结果

也可以通过切片或条件筛选进行数据过滤。

例 5-12：数据切片与筛选。

```
data[:2]
data[data['s']<=10]
```

运行结果如图 5-10 所示。

In [5]: data[:2]

Out[5]:

	q	r	s	t
BJ	0	1	2	3
SH	4	5	6	7

In [4]: data[data['s']<=10]

Out[4]:

	q	r	s	t
BJ	0	1	2	3
SH	4	5	6	7
GZ	8	9	10	11

图 5-10　例 5-12 运行结果

还可以使用 loc()和 iloc()函数进行切片。

其中，按索引名关键字抽取指定行列的数据使用 loc()函数，标准格式如下。

```
<DataFrame 对象>.loc[<行索引名或行索引名列表>,<列索引名或列索引名列表>]
```

按位置访问指定行列时，可以使用 iloc()函数，同样有行、列两个参数，可以是下标值，可以是列表，还可以是切片，标准格式如下。

```
<DataFrame 对象>.iloc[<行参数>,<列参数>]
```

例 5-13：抽取指定行列的数据。

```
data.loc[['SH','GZ'],['r','s']]
```

或：

```
data.iloc[:-1,1:3]
```

运行结果如图 5-11 所示。

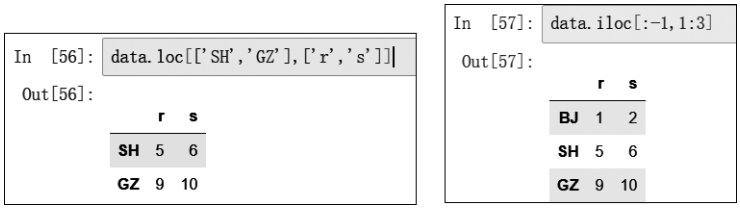

图 5-11　例 5-13 运行结果

3. 修改 DataFrame 数据

1）修改数据

通过赋值语句修改数据，可以修改指定行、列的数据，还可以把要修改的数据查询筛选出来，或重新赋值。

例 5-14：修改数组中的某个数据。

```
import numpy as np
import pandas as pd
data=pd.DataFrame(np.arange(16).reshape(4,4),
                  index=['BJ','SH','GZ','SZ'],
                  columns=['q','r','s','t'])
data['q']['BJ']=8
data['t']=8
data['s']['SZ']=8
data
```

运行结果如图 5-12 所示。

2）增加列

可以为 DataFrame 对象添加新的列，通过赋值语句赋值时，只要列索引名不存在，就添加新列，否则就修改列值，这与字典的特性相似。

例 5-15：为 data 增加一列“u”，值为 9。

```
data['u']=9
data
```

运行后 data 如图 5-13 所示。

	q	r	s	t
BJ	8	1	2	8
SH	4	5	6	8
GZ	8	9	10	8
SZ	12	13	8	8

图 5-12　例 5-14 运行结果

	q	r	s	t	u
BJ	8	1	2	8	9
SH	4	5	6	8	9
GZ	8	9	10	8	9
SZ	12	13	8	8	9

图 5-13　例 5-15 运行结果

3）合并添加数据

DataFrame 对象可以增加新列，但与 Series 对象一样不能直接增加新行。如果需要增加几行数据，需要将数据存入一个新 DataFrame 对象，然后将两个 DataFrame 对象进行合并。

两个 DataFrame 对象的合并可以使用 Pandas 的 concat()方法，通过 axis 参数的选择，能够按不同的轴向连接两个 DataFrame 对象。

4）删除 DataFrame 对象的数据

drop()函数可以按行列删除数据，drop()函数基本格式如下。

```
<DataFrame 对象>.drop(索引值或索引列表,axis=0, inplace=False,…)
```

主要参数如下。

- axis：默认为 0，为行索引值或列索引列表；值为 0 表示删除行，值为 1 表示删除列。
- inplace：逻辑型，表示操作是否对原数据生效。默认为 False，产生新对象，原 DataFrame 对象内容不变。

例 5-16：DataFrame 对象的删除操作。

```
dt1=data.drop('SZ',axis=0)          #删除 index 值为'SZ'的行
dt2=data.drop(["r","u"],axis=1)     #删除"r","u"列
data.drop('SZ',inplace=True)        #从原数据中删除一行
```

运行结果如图 5-14 所示。

	q	r	s	t	u
BJ	8	1	2	8	9
SH	4	5	6	8	9
GZ	8	9	10	8	9

	q	s	t
BJ	8	2	8
SH	4	6	8
GZ	8	10	8
SZ	12	8	8

	q	r	s	t
BJ	8	1	2	8
SH	4	5	6	8
GZ	8	9	10	8

图 5-14　例 5-16 运行结果

4. 汇总和描述性统计计算

Pandas 的 Series 对象和 DataFrame 对象都继承了 NumPy 的统计函数，常用的数学和统计方法如表 5-1 所示，通常可以对一列或多列数据进行统计分析。

表 5-1　常用的描述和汇总统计函数

函　数　名	功 能 说 明
count	统计数据值的数量，不包括 NA 值
describe	对 Series、DataFrame 进行汇总统计
min，max	计算最小值、最大值
argmin，argmax	计算最小值、最大值的索引位置
idxmin，idxmax	计算最小值、最大值的索引值
sum	计算总和
mean	计算平均值
median	返回中位数
var	计算样本值的方差
std	计算样本值的标准差
cumsum	计算样本值的累计和
diff	计算一阶差分

例 5-17：一个简单的 DataFrame。

```
df=pd.DataFrame(np.arange(16).reshape(4,4),
                index=['BJ','SH','GZ','SZ'],
                columns=['q','r','s','t'])
df
```

DataFrame 的跨行用 0 轴表示，跨列用 1 轴表示。例如，按 0 轴求和的操作如下。

```
df.sum()
```

或：

```
df.sum(axis=0)
```

运行结果如图 5-15 所示。

按 1 轴求和操作如下。

```
df.sum(axis=1)
```

运行结果如图 5-16 所示。

```
Out[80]: q    24
         r    28
         s    32
         t    36
         dtype: int64
```

图 5-15　按 0 轴求和

```
Out[81]: BJ     6
         SH    22
         GZ    38
         SZ    54
         dtype: int64
```

图 5-16　按 1 轴求和

求平均值的操作如下。

```
df.mean(axis=1)
```

```
Out[82]: BJ     1.5
         SH     5.5
         GZ     9.5
         SZ    13.5
         dtype: float64
```

图 5-17　求平均值

运行结果如图 5-17 所示。

求最大值和最小值也很方便，分别使用 df.max()、df.min()方法。

例 5-18：综合示例——英文分词。

在文本处理中，分词是一项基本任务，能够表达内容相关性、提取页面关键词、主题标签等。下面对英文句子进行基本的单词频率提取。

```
#使用变量 p 保存英文句子
p='life can be good,life can be sad,life is mostly cheerful,but sometimes sad.'
pList=p.split() #使用 split()函数进行分隔
pdict={}
#对分隔后的词语进行统计
for item in pList:
    if item[-1] in ',.':
        item=item[:-1]
    if item not in pdict:
```

```
            pdict[item]=1
        else:
            pdict[item]+=1
    print(pdict)
```

输出的分词结果如图 5-18 所示。

```
{'life': 1, 'can': 2, 'be': 2, 'good,life': 1, 'sad,life': 1, 'is': 1, 'mostly': 1, 'cheerful,but': 1,
'sometimes': 1, 'sad': 1}
```

图 5-18　分词结果

5. 分组统计

在日常的数据分析中，经常需要将数据根据一个或多个字段划分为不同的组进行分析，如电商领域将全国的总销售额按省份进行分组，并分析各省销售额的变化情况；或者社交网站将用户根据性别、年龄等特征进行细分，据此研究用户画像和偏好等。

在 Pandas 中，上述分组操作主要运用 groupby()函数完成，然后进行基本的分组操作，如组内计数、求和、求均值和求方差等。

例 5-19：对电影评分数据集合 IMDB300 进行处理，按职业进行分组，统计各职业的用户人数。

分析：按职业进行分组，可以使用 groupby('uOccup')进行分组。还通常使用 groupby 分组对象的 size()方法以返回组大小(Series 结构)，使用 count()方法进行组内特征计数(Series 或 Dataframe 结构)。

```
import pandas as pd
f = open('score.txt',encoding='UTF-8')
df=pd.read_csv(f,sep='\t',header=None,names=['Index','uNo','uAge','uOccup',
'filmNo', 'filmName','url','score','timestamp'])
#分组统计
df.groupby('uOccup').size()        #查看组大小结果
```

从运行结果可以看出，size()方法返回的是当前组的大小，如图 5-19 所示。

```
uOccup
administrator    40
artist            7
doctor            9
educator         29
engineer         24
entertainment     6
executive        22
```

图 5-19　返回当前组的大小

再增加下面的语句：

```
df.groupby('uOccup').count()          #查看计数结果
```

可以得到如图 5-20 所示结果。

从运行结果可以看到，count()方法返回的是一个 DataFrame 结构，内容为各组的全部特征的计数。