第5章 Python数据表分析

在读者具备了 pandas 库和 numpy 库的基础知识以后，本章学习如何使用它们对数据表(即结构化数据)进行分析。5.1 节介绍如何使用 pandas 库实现数据概览及预处理，5.2 节在讲解数据排序后介绍 pandas 库和 numpy 库中常用的数据计算函数和方法，以便于进行数据的描述性统计分析，5.3 节介绍如何实现数据的分组统计分析，5.4 节以豆瓣读书排行榜的数据为例进行对应任务的实战分析。

5.1 数据概览及预处理

扫一扫

视频讲解

5.1.1 数据概览分析

数据概览是在数据分析之前对数据的规模、数据的类型及数据的质量等进行概览性的分析。使用 pandas 库中 DataFrame 对象的常用属性和方法可以进行数据概览。表 5.1 中列出 DataFrame 的常用属性，用于查看数据的基本信息以及数据的规模。

表 5.1 DataFrame 的常用属性

属　性	作　用	类　别
index	行名(索引)	数据的基本信息
columns	列名	
dtypes	数据的类型	
values	数据值	
shape	数据的形状	数据的规模
ndim	数据的维度	
size	数据中元素的个数	
Index. size	行数	
columns. size	列数	

下面以存储在 cj. xlsx 文件中的“成绩表”数据集为例进行数据的概览性分析，部分数据的示例如图 5.1 所示。

首先引入 pandas 库，读取 cj. xlsx 文件中的数据集，将其存储在 DataFrame 对象 df 中。

```
import pandas as pd
#读取数据
df = pd.read_excel("tdata/cj.xlsx")
```

注意：这里的 cj. xlsx 文件存储在当前路径下的 tdata 文件夹中。

学号	姓名	性别	专业	英语	数学	Python	选修	管理学
2020802045	魏天	男	信息管理与信息系统	67.12	90.8	93.0	95	106.0
2020844001	郭夏	男	国际贸易	91.05	83.4	86.0	100	99.0
2020844002	王晓加	男		54.20	83.4	74.0		90.0
2020844003	黄婷婷	女	国际贸易	87.80	91.4	79.7	95	92.7
2020844004	赵小瑜		国际贸易	61.15	82.2	84.7	100	97.7
2020844005	辛禧	男	国际贸易	65.13	88.6	68.0	80	81.0
2020844007	王晨	男	国际贸易	62.40	80.0	65.0	90	78.0
2020844008	韩天	男	国际贸易	96.25	91.0	85.0	97	98.0
2020844009	刘玉	女	国际贸易	89.05	91.4	80.3	100	93.3

图 5.1　数据集示例

【例 5.1】 使用基础属性查看 df 数据集的基本信息。

```
print("索引:",df.index)
print("列名:",df.columns)
print("数据元素:",df.values[:5])
print("数据类型:\n",df.dtypes)
```

显示结果如下：

```
索引: RangeIndex(start = 0, stop = 57, step = 1)
列名: Index(['学号', '姓名', '性别', '专业', '英语', '数学', 'Python', '选修',
       '管理学'], dtype = 'object')
数据元素: [[2020802045 '魏天' '男' '信息管理与信息系统' 67.12 90.80
  93.0 95.0 106.0]
[2020844001 '郭夏' '男' '国际贸易' 91.05 83.4 86.0 100.0 99.0]
[2020844002 '王晓加' '男' nan 54.2 83.4 74.0 nan 90.0]
[2020844003 '黄婷婷' '女' '国际贸易' 87.8 91.4 79.66 95.0 92.66]
[2020844004 '赵小瑜' nan '国际贸易' 61.15 82.2 84.66 100.0 97.66]]
数据类型:
学号          int64
姓名          object
性别          object
专业          object
英语          float64
数学          float64
Python        float64
选修          float64
管理学        float64
dtype: object
```

【例 5.2】 使用基础属性查看 df 数据集的规模。

```
print("元素个数:",df.size)
print("维度数:",df.ndim)
print("形状:",df.shape)
print("行数:",df.index.size)
print("列数:",df.columns.size)
```

显示结果如下：

```
元素个数: 513
维度数: 2
形状: (57, 9)
```

```
行数: 57
列数: 9
```

使用 DataFrame 的一些方法可以查看数据、查看数据的总体信息、查看数据的缺失情况以及判断是否有重复的数据，如表 5.2 所示。

表 5.2 使用 DataFrame 进行概览分析的常用方法

方 法	作 用
head(n)	查看前 n 行数据，不指定 n，默认为 5 行
tail(n)	查看后 n 行数据，不指定 n，默认为 5 行
info()	查看数据的总体信息，包括数据类型、行/列名、行/列数、每列的数据类型、所占内存等
isnull()/isna() notnull()/notna()	查看数据的缺失情况
duplicated()	判断是否有重复的数据

【例 5.3】 查看 df 数据集的样本数据。

```
#查看数据集的前两条样本
df.head(2)
```

结果如图 5.2 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
0	2020802045	魏天	男	信息管理与信息系统	67.116667	90.8	93.0	95.0	106.0
1	2020844001	郭夏	男	国际贸易	91.050000	83.4	86.0	100.0	99.0

图 5.2 数据集的前两条样本

【例 5.4】 查看数据集 df 中各个特征的缺失情况。

```
print("df 中每个特征的缺失情况:\n", df.isna() .sum())
```

显示结果如下：

```
df 中每个特征的缺失情况:
学号        0
姓名        0
性别        3
专业        3
英语        0
数学        0
Python      0
选修        4
管理学      0
dtype: int64
```

isna 方法判断每个样本值是否缺失，如果缺失，返回 True。在进行概览性分析时直接使用 isna 方法不利于掌握数据集样本缺失的整体情况，因此通常在 isna 方法后要再使用 sum 方法汇总每列样本值缺失的总个数。

【例 5.5】 查看数据集 df 的完整信息摘要。

```
df.info()
```

显示结果如下：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57 entries, 0 to 56
Data columns (total 9 columns) :
学号          57 non-null int64
姓名          57 non-null object
性别          54 non-null object
专业          54 non-null object
英语          57 non-null float64
数学          57 non-null float64
Python        57 non-null float64
选修          53 non-null float64
管理学        57 non-null float64
dtypes: float64(5) , int64(1) , object(3)
memory usage: 4.1+ KB
```

从输出结果可以看出，完整的数据集摘要信息包括数据集的数据类型，行/列索引信息，各个列的名称、非空值的数量以及该列的数据类型，所有列的数据类别的信息汇总以及DataFrame元素的内存使用情况。

5.1.2 数据清洗

数据缺失会导致样本数据信息减少，增加数据分析的难度，也会导致数据分析的结果产生偏差；数据重复会导致数据之间的差异变小，数据的分布发生较大变化；数据集中有异常值会产生“伪回归”，因此需要在数据概览后对这些数据进行适当的处理。

数据清洗是通过预处理去除数据中的噪声，恢复数据的完整性和一致性。数据清洗通常包括缺失值处理、重复值处理以及异常值处理。

(1) 在处理缺失值时，可以采用删除缺失值法或者用其他值替换缺失值的方法。

(2) 在处理重复值时，一般采用删除的方法。

(3) 异常值的处理方法和缺失值相同，可以采用删除或替换的方式，只是对于如何判定异常值(也就是异常值检测)，需要根据数据的具体情况采用不同的方法。

本节以5.1.1节中存储到DataFrame对象df中的数据集为例进行数据清洗的相关操作的介绍。

```
import pandas as pd
#解决数据输出时列名不对齐的问题
pd.set_option('display.unicode.east_asian_width', True)
#读取数据
df = pd.read_excel("tdata/cj.xlsx")
```

1. 缺失值处理

在pandas中专门提供了删除缺失值的dropna方法和替换缺失值的fillna方法。

dropna方法的语法格式如下：

```
DataFrame.dropna(axis = 0, how = "any", subset = None, inplace = False)
```

常见参数的说明如下。

(1) axis：表示轴向，其中 0 为删除行、1 为删除列，默认为 0。

(2) how：表示删除形式，其中"any"表示只要有缺失值就删除，"all"表示全部为缺失值时才删除，默认为"any"。

(3) subset：表示进行去重的列/行，接收 array，默认为 None。

(4) inplace：表示是否在原表上进行操作，接收 boolean，默认为 False。

dropna 方法由 DataFrame 直接调用。

注意：为了有效地保护原始数据，pandas 提供了一种策略，即涉及删除、修改等操作时，默认都会在备份数据上进行，所以在涉及 DataFrame 数据修改和删除的方法中都有 inplace 参数，其作用是相同的。

【例 5.6】 数据集 df 的缺失值的删除处理示例。

```
#存在任一缺失值即删除
df1 = df.dropna()
print("删除前:",df.shape)
print("删除后:",df1.shape)
#所有列均为缺失值即删除
df1 = df.dropna(how = "all")
print("删除前:",df.shape)
print("删除后:",df1.shape)
#指定列均为缺失值即删除
df1 = df.dropna(how = "all",subset = ["专业","选修"])
print("删除前:",df.shape)
print("删除后:",df1.shape)
```

显示结果如下：

```
删除前: (57, 9)
删除后: (48, 9)

删除前: (57, 9)
删除后: (57, 9)

删除前: (57, 9)
删除后: (56, 9)
```

fillna 方法的语法格式如下：

```
DataFrame.fillna(value, method = None,axis = 1, inplace = False,limit = None)
```

常见参数的说明如下。

(1) value：接收替换缺失值的值。

(2) method：表示缺失值的填充方法，默认为 None。其中，"backfill"或"bfill"表示用下一个非缺失值来填充；"pad"或"ffill"为使用上一个非缺失值来填充。

(3) axis：表示轴向，其中 0 为行、1 为列，默认为 0。

(4) inplace：表示是否在原表上进行操作，接收 boolean，默认为 False。

(5) limit：表示填补缺失值个数的上限，接收 int，默认为 None。

fillna 方法由 DataFrame 直接调用,其中在进行缺失值替换的时候会根据数据集的特点采取相应策略,如果想用数据集中缺失值之前或之后的值进行替换,则可以设置 method 填充方法。

【例 5.7】 数据集 df 的缺失值的填充处理示例。

```
#将缺失值 NaN 填充为 0
df['选修'].fillna(0)
#将缺失值 NaN 填充为后面的值
df['选修'].fillna(method = "bfill")
#将缺失值 NaN 填充为选修课的平均分
df['选修'].fillna(np.mean(df["选修"]))
```

2. 重复值处理

处理重复值可以使用 drop_duplicates 方法。

drop_duplicates 方法的语法格式如下:

```
DataFrame.drop_duplicates(subset = None,keep = "first",inplace = False)
```

常见参数的说明如下。

(1) subset: 表示进行去重的列,默认为 None,表示全部列。

(2) keep: 表示重复时保留第几个数据,默认为"first"。其中,"first"为保留第一个,"last"为保留最后一个,False 为均不保留重复值。

(3) inplace: 表示是否在原表上进行操作,接收 boolean,默认为 False。

调用 drop_duplicates 方法,如果直接使用无参数的方法,表示要去除全部的重复数据; 如果设置 subset 参数,则去除指定列的重复数据,keep 参数表示去重时保留值的方法。

【例 5.8】 数据集 df 的重复值的处理示例。

```
#去除全部重复数据
df1 = df.drop_duplicates()
print("去重前:",df.shape)
print("去重后:",df1.shape)
#去除指定列中的重复数据
df1 = df.drop_duplicates(["专业"])
print("去重前:",df.shape)
print("去重后:",df1.shape)
#去除指定列中的重复数据,设置 keep 参数
df1 = df.drop_duplicates(["专业"],keep = "last")
print("去重前:",df.shape)
print("去重后:",df1.shape)
#去除指定若干列中的重复数据
df1 = df.drop_duplicates(["学号","姓名"])
print("去重前:",df.shape)
print("去重后:",df1.shape)
```

显示结果如下:

```
去重前: (57, 9)
```

```
去重后：(54, 9)

去重前：(57, 9)
去重后：(6, 9)

去重前：(57, 9)
去重后：(6, 9)

去重前：(57, 9)
去重后：(54, 9)
```

3. 异常值处理

在数据分析中，异常值是指超出或低于正常范围的值，比如成绩高于 100 分、身高超过 3 米、商品价格为负值等。一般可以简单地根据给定的数据范围进行判断，将不在范围内的数据视为异常值。另外，也可以使用统计学均方差检查数据是不是在标准差范围内，如图 5.3 所示，这个方法的局限性在于需要数据具有正态分布特征。后续在可视化分析中学习箱形图的绘制后，也可以使用箱形图识别异常值，如图 5.4 所示。

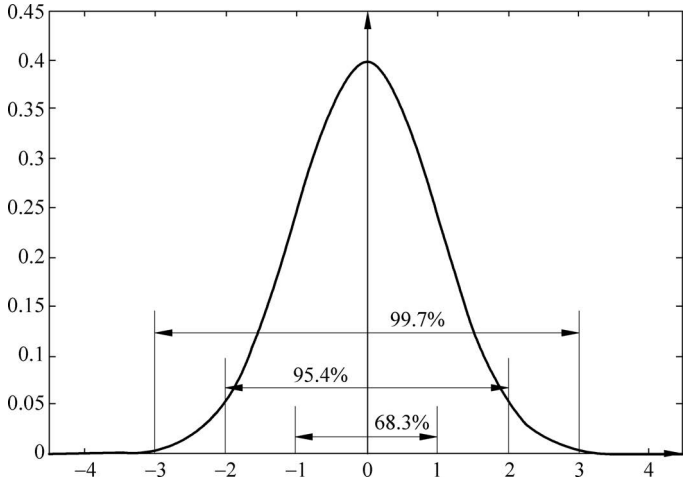


图 5.3 用统计学标准差检测异常值

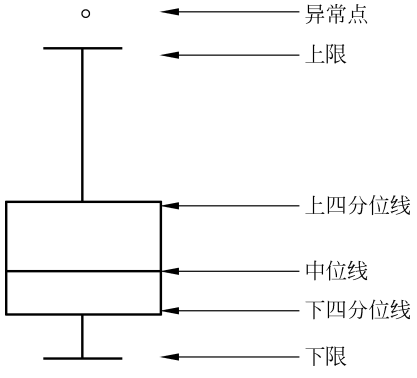


图 5.4 用箱形图检测异常值

当然,针对不同的情形还有更多异常值的检测方法,这里不再介绍。

在异常值检测出来以后,最常见的处理方式是删除,也可以把它们当作缺失值进行替换填充处理。另外在一些场景中,比如针对信用卡使用记录的分析,还会专门把异常值当成特殊情况进行分析,研究其出现的原因。

5.1.3 数据的抽取与合并

在进行数据分析时,并不是所有数据集中的数据都是必需的,此时可以进行数据抽取;有时要分析的数据来源于不同的数据表,例如想对排行榜上的书籍信息进行分析,而所需要的数据在两张表中,这时就要进行数据合并。数据抽取和数据合并是进行结构化数据分析预处理中常见的操作,pandas 库对这两项操作提供了灵活、便利的实现方法。

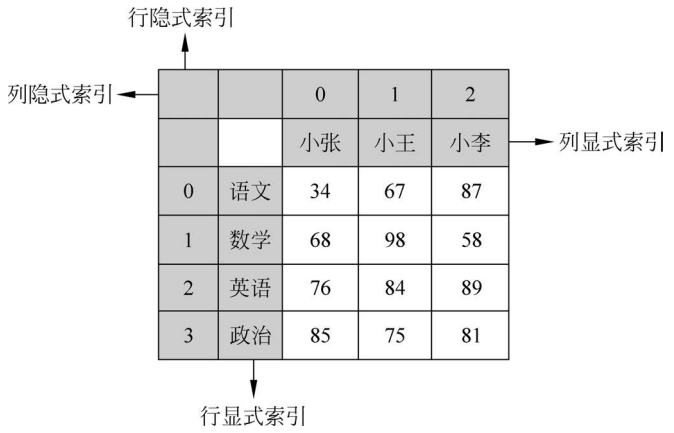
1. 数据抽取

对数据集进行抽取,既可以抽取所需要的列,也就是数据集的特征,也可以抽取所需要的行,也就是数据集的样本。

1) DataFrame 的索引

DataFrame 数据结构有行和列两种索引,DataFrame 的行、列索引都有两种表示方法:一是隐式索引,系统默认赋值的索引编号,下标从 0 开始;二是显式索引,是用户通过 index 和 columns 属性设置的行和列的名称。

隐式索引符合计算机用户的使用习惯,数据下标从 0 开始,但是显式索引更符合数据分析用户的习惯,因为带有标签的数据更有意义,比如 34 这个数据通过行、列显式索引可以看出是小张的语文成绩,而仅用隐式索引表示时,只能说 34 是第 0 行第 0 列的数据,如图 5.5 所示。



		0	1	2
		小张	小王	小李
0	语文	34	67	87
1	数学	68	98	58
2	英语	76	84	89
3	政治	85	75	81

图 5.5 DataFrame 的索引

2) DataFrame 的数据抽取方法

在 DataFrame 中隐式索引和显式索引表示方式是共存的,在抽取数据时,用 iloc 属性可以按照系统默认的隐式索引抽取数据,用 loc 属性可以根据用户定义的显式索引抽取数据。此外,抽取数据还可以用 DataFrame 的列名属性和索引下标的方法。

在实际应用中,因为行、列抽取的需求不同,大家在初学时可能会觉得 pandas 中对 DataFrame 的数据抽取相对复杂,但总结起来一共有列名属性、索引下标、iloc 属性、loc 属性 4 种方法,具体如表 5.3 所示。

表 5.3 数据抽取方法

方法	用法	作用	示 例	说 明
列名属性	.列名	抽取单列	df.学号	抽取学号列,返回 Series 类型
索引下标	[]	抽取单列或多列	df["学号"] df[["学号"]] df[["学号","姓名"]]	抽取学号列,返回 Series 类型 抽取学号列,返回 DataFrame 类型 抽取学号、姓名两列
iloc 属性	.iloc[]	抽取任意列、任意行或指定行/列	df.iloc[:,[0]] df.iloc[1:20,] df.iloc[1:10,2:5]	抽取第 0 列 抽取第 1 到第 19 行 抽取第 1 到第 9 行,第 2 到第 4 列
loc 属性	.loc[]	抽取任意列、任意行或指定行/列或根据条件进行抽取	df.loc[:,["学号"]] df.loc[1:20,] df.loc[1:10,["学号"]] df.loc[df.英语>90,]	抽取“学号”列 抽取第 1 到第 20 行 抽取第 1 到第 10 行的学号列 抽取英语成绩超过 90 的所有行

3) 数据抽取示例

下面仍以 cj.xlsx 中的数据集为例来看使用这些方法抽取列、抽取行以及抽取行和列的示例。首先将数据集读取到 Dataframe 对象 df 中。

```
import pandas as pd
pd.set_option('display.unicode.east_asian_width', True)
df = pd.read_excel("tdata/cj.xlsx")
```

(1) 抽取列的示例。

```
>>> df.学号
```

结果显示为:

```
0     2020802045
1     2020844001
2     2020844002
…
```

使用 DataFrame 属性的方法可以抽取一列,其返回的数据是 Series 类型,这和使用[]索引下标引用单列的方法的作用相同,此时[]索引下标中的列名以字符串形式给出。

```
>>> df["学号"]
```

结果显示为:

```
0     2020802045
1     2020844001
2     2020844002
…
```

在[]索引下标中也可以放置要抽取列名构成的列表,其中列名同样以字符串形式给出,用这种方式抽取的结果是 DataFrame 类型。

在列表中可以是单个列名:

```
>>> df[["学号"]]
```

结果显示为：

```
    学号
0  2020802045
1  2020844001
2  2020844002
    …
```

在列表中也可以是多个列名：

```
>>> df[["学号","姓名","专业"]]
```

结果显示为：

```
    学号        姓名          专业
0  2020802045    魏天  信息管理与信息系统
1  2020844001    郭夏        国际贸易
2  2020844002   王晓加         NaN
    …
```

用 iloc 属性和 loc 属性进行数据抽取时，需要在属性后跟[ ]索引下标，[ ]里面是先行后列，用逗号“,”分隔。行和列的表示形式可以是单行、单列、切片(切片表示连续区域或列表)。

注意：iloc 属性和 loc 属性的区别如下。

- iloc 属性中行列是隐式索引，使用整数的位置访问 DataFrame 元素。
- loc 属性中行列是显式索引，使用标签的索引访问 DataFrame 元素。
- 使用切片表示连续区域时，loc 属性抽取到包括切片结束点本身数据，而 iloc 属性抽取到结束点－1 的数据。

下面的代码同样可以分别实现对学号、姓名和专业列的数据抽取。

```
>>> df.iloc[:,[0,1,3]]
>>> df.loc[:,["学号","姓名","专业"]]
```

(2) 抽取行的示例。

抽取连续行数据可以使用 iloc 属性或 loc 属性。

```
>>> df.loc[1:5,]
```

结果如图 5.6 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
1	2020844001	郭夏	男	国际贸易	91.050	83.4	86.00	100.0	99.00
2	2020844002	王晓加	男	NaN	54.200	83.4	74.00	NaN	90.00
3	2020844003	黄婷婷	女	国际贸易	87.800	91.4	79.66	95.0	92.66
4	2020844004	赵小瑜	NaN	国际贸易	61.150	82.2	84.66	100.0	97.66
5	2020844005	辛禧	男	国际贸易	65.125	88.6	68.00	80.0	81.00

图 5.6　loc 属性切片抽取结果

注意：使用 loc 属性时，此处[ ]中的 1 和 5 被看作行标签，切片数据包括标签 5 所在行；使用 iloc 属性时，1 和 5 则被看作位置，切片数据不包括数字 5 所在的行。读者可以自行尝试

df.iloc[1:5,],体会二者的区别。

在抽取行时,除了可以使用切片抽取连续行,还可以抽取用列表表示的不连续的行。

```
>>> df.iloc[[1,2, 6,7],]
```

结果如图 5.7 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
1	2020844001	郭夏	男	国际贸易	91.05	83.4	86.0	100.0	99.0
2	2020844002	王晓加	男	NaN	54.20	83.4	74.0	NaN	90.0
6	2020844007	王晨	男	国际贸易	62.40	80.0	65.0	90.0	78.0
7	2020844008	韩天	男	国际贸易	96.25	91.0	85.0	97.0	98.0

图 5.7 iloc 属性列表抽取结果

另外,如果将 DataFrame 类型数据看作序列,可以使用切片的方法来抽取指定的连续行。

```
>>> df[5:7]
```

结果如图 5.8 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
5	2020844005	辛禧	男	国际贸易	65.125	88.6	68.0	80.0	81.0
6	2020844007	王晨	男	国际贸易	62.400	80.0	65.0	90.0	78.0

图 5.8 序列切片抽取结果

注意:这种方法实际上较少使用,而且它不能抽取单行或不连续的行。

在根据用户自定义的条件对行数据进行抽取时只能使用 loc 属性,例如下面的代码抽取了英语列大于 90 的行数据。

```
>>> df.loc[df.英语>90,]
```

结果如图 5.9 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
1	2020844001	郭夏	男	国际贸易	91.05	83.4	86.00	100.0	99.00
7	2020844008	韩天	男	国际贸易	96.25	91.0	85.00	97.0	98.00
17	2020844019	陈雨涵	男	市场营销	95.20	95.0	88.00	100.0	101.00
18	2020844020	张家齐	男	市场营销	95.45	91.0	96.00	100.0	109.00
33	2020848003	张淳	女	会计学	91.30	92.2	81.32	100.0	94.32
37	2020848007	苏远	女	信息管理与信息系统	90.25	89.2	79.32	68.0	92.32
38	2020848008	方雨桃	女	信息管理与信息系统	93.10	86.2	83.00	100.0	96.00
40	2020848011	张田田	女	信息管理与信息系统	91.20	89.6	96.32	77.0	109.32
44	2020848016	杨帆	男	信息管理与信息系统	98.70	87.6	95.00	NaN	108.00
53	2020848027	热孜耶·买买提	女	金融学	92.70	93.2	86.32	100.0	99.32

图 5.9 loc 属性条件抽取结果

(3) 抽取行和列的示例。

抽取指定行和列可以采用以下两种方法。

方法一:用 DataFrame 数据框类型的索引下标法先抽取指定的列,再使用切片法抽取指定连续行的方法。

方法二：用 iloc 或 loc 属性的方法在[ ]中同时指定要抽取的行、列索引。

其中 iloc 和 loc 属性的方法比较常用。

使用方法一进行抽取的示例如下。

这里先抽取指定的学号、姓名、专业列，再用切片法抽取前两行：

```
>>> df[["学号","姓名","专业"]][:2]          #用切片法抽取连续行
```

结果显示为：

```
      学号      姓名          专业
0  2020802045     魏天   信息管理与信息系统
1  2020844001     郭夏        国际贸易
```

注意：这种方法不适用于抽取单行或不连续的行，但可以在抽取列后通过自定义条件来抽取行，例如抽取学号、姓名、专业列，且数学成绩大于 90 分的样本记录。

```
>>> df[["学号","姓名","专业"]][df.数学> 90]
```

结果显示为：

```
      学号      姓名          专业
0  2020802045     魏天   信息管理与信息系统
3  2020844003    黄婷婷        国际贸易
7  2020844008     韩天        国际贸易
                  …
```

在方法二中可以用 iloc 属性或 loc 属性同时指定行、列索引进行抽取。在抽取指定行和列的时候常用这两个属性，并且这两个属性在指定行时比较灵活，可以是单行、切片表示的连续行、列表表示的不连续行。其中 iloc 属性在指定行、列时均使用隐式索引，loc 属性在抽取列时使用显式索引。

首先来看 iloc 属性进行行、列抽取的示例。

```
>>> df.iloc[3,2:5]                  #抽取单行
```

结果显示为：

```
性别          女
专业     国际贸易
英语         87.8
Name: 3, dtype: object
```

这里返回的是 Series 序列类型的数据。

```
>>> df.iloc[[3],2:5]                #抽取单行
```

结果显示为：

```
   性别     专业      英语
3   女    国际贸易   87.8
```

这里返回的是 DataFrame 数据框类型的数据。

```
>>> df.iloc[:3,2:5]                    #抽取连续行
```

结果显示为：

```
    性别          专业          英语
0    男    信息管理与信息系统  67.116667
1    男       国际贸易       91.050000
2    男         NaN          54.200000
```

另外还可以指定不连续的行和不连续的列，例如：

```
>>> df.iloc[[3,13],[0,2]]              #抽取不连续的行
```

结果显示为：

```
    学号      性别
3    男        女
13   男        男
```

用 loc 属性方法则可以显式指定要抽取的列索引，例如：

```
>>> df.loc[:2,["学号","姓名","专业"]]
```

结果显示为：

```
       学号        姓名          专业
0  2020802045     魏天      信息管理与信息系统
1  2020844001     郭夏         国际贸易
2  2020844002    王晓加          NaN
```

用 loc 属性方法也可以自定义条件抽取指定的行，例如：

```
>>> df.loc[df.数学>90,["学号","姓名","专业"]]
```

结果显示为：

```
       学号        姓名          专业
0  2020802045     魏天      信息管理与信息系统
3  2020844003    黄婷婷         国际贸易
7  2020844008     韩天         国际贸易
                  …
```

2. 数据合并

在进行数据分析时需要的数据可能来源于不同的表，为了便于分析，通常会对这些数据进行合并。数据合并涉及按列合并特征还是按行合并样本。表 5.4～表 5.7 所示的是 4 个数据

集,其中 df1 可以分别和 df2、df3 按列合并形成特征更丰富的数据集,这里 df1 和 df2 有重叠的隐式行索引,df1 和 df3 有重叠的“学号”列。df1 和 df4 可以按行合并,增加样本量。

表 5.4 数据集 df1

序 号	学 号	姓 名	专 业
0	2020802045	魏天	信息管理与信息系统
1	2020844001	郭夏	国际贸易
2	2020844002	王晓加	NaN
3	2020844003	黄婷婷	国际贸易
4	2020844004	赵小瑜	国际贸易

表 5.5 数据集 df2

序 号	数 学	选 修
0	90.8	95.0
1	83.4	100.0
2	83.4	NaN
3	91.4	95.0
4	82.2	100.0

表 5.6 数据集 df3

序 号	学 号	Python
0	2020802045	93.00
1	2020844001	86.00
2	2020844002	74.00
3	2020844003	79.66
4	2020844004	84.66

表 5.7 数据集 df4

序 号	学 号	姓 名	专 业
20	2020844022	关帅	会计学
21	2020844023	刘嘉雯	会计学
22	2020844024	刘浩天	会计学
23	2020844025	刘宇	NaN
24	2020844026	胡童	会计学
25	2020844027	丁灿	会计学

上述 4 个 DataFrame 数据集由“成绩表”数据集抽取而成,具体生成程序的代码如下:

```
import pandas as pd
pd.set_option('display.unicode.east_asian_width', True)
df = pd.read_excel("tdata/cj.xlsx")                    #读取数据
df1 = df[["学号","姓名","专业"]][:5]                    #生成 df1
df2 = df[["学号","Python"]][:5]                        #生成 df2
df3 = df[["数学","选修"]][:5]                           #生成 df3
df4 = df.loc[20:25,["学号","姓名","专业"]]              #生成 df4
```

在 pandas 库中主要有 4 个方法用于特征和样本,即列和行的合并。

- 按列合并特征,可以采用 DataFrame 的 join 方法,或 pandas 中的 merge 和 concat 方法。
- 按行合并样本,可以采用 DataFrame 的 append 方法,或 pandas 中的 concat 方法。

1）合并列

合并列常用的方法是 join 和 merge 方法，其中 join 方法默认是按照 DataFrame 的行索引、merge 方法默认是按照 DataFrame 中同名的列来实现列的合并功能。

下面是 join 方法的语法格式：

```
DataFrame.join(other, on = None, how = 'inner', lsuffix = '', rsuffix = '', sort = False)
```

主要参数的说明如下。

（1）other：接收 DataFrame、Series 或者包含了多个 DataFrame 的 list，表示参与连接的其他 DataFrame，无默认值。

（2）on：接收列名或者包含列名的 list 或 tuple，表示用于连接的列名，默认为 None。

（3）how：接收特定 string。inner 代表内连接，outer 代表外连接，left 和 right 分别代表左连接和右连接，默认为 inner。

（4）lsuffix：接收 string，表示用于追加到左侧重叠列名的末尾，无默认值。

（5）rsuffix：接收 string，表示用于追加到右侧重叠列名的末尾，无默认值。

（6）sort：根据连接主键对合并后的数据进行排序，默认为 True。

join 方法在合并数据集的列时，默认是按照 DataFrame 的行索引来实现按列合并的功能，因此不允许待合并的数据集中有重叠（同名）列，如果数据集中有同名列，在进行合并时需要指明 lsuffix 或 rsuffix 参数，即给同名列起一个别名。

【例 5.9】 使用 join 方法按列合并数据集。

按列合并 df1 和 df3：

```
df1.join(df3)
```

df1 和 df3 默认以 index 为连接主键合并列，结果如图 5.10 所示。

	学号	姓名	专业	数学	选修
0	2020802045	魏天	信息管理与信息系统	90.8	95.0
1	2020844001	郭夏	国际贸易	83.4	100.0
2	2020844002	王晓加	NaN	83.4	NaN
3	2020844003	黄婷婷	国际贸易	91.4	95.0
4	2020844004	赵小瑜	国际贸易	82.2	100.0

图 5.10 df1 和 df3 使用 join 方法合并结果

在默认情况下，如果待合并的数据集中有同名的列，用 join 方法合并会报错，例如：

```
df1.join(df2)                          #有同名列,无法区分报错
```

报错信息为：

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\reshape\merge.py in _items_overlap_
with_suffix(left, right, suffixes)
   2176
   2177     if not lsuffix and not rsuffix:
-> 2178         raise ValueError(f"columns overlap but no suffix specified: {to_rename}")
   2179
   2180     def renamer(x, suffix):

ValueError: columns overlap but no suffix specified: Index(['学号'], dtype = 'object')
```

此时,为了合并成功,可以在 join 方法中使用 lsuffix 或 rsuffix 参数,例如:

```
df1.join(df2,lsuffix = "x")                    #给同名列起别名
```

结果如图 5.11 所示。

	学号x	姓名	专业	学号	Python
0	2020802045	魏天	信息管理与信息系统	2020802045	93.00
1	2020844001	郭夏	国际贸易	2020844001	86.00
2	2020844002	王晓加	NaN	2020844002	74.00
3	2020844003	黄婷婷	国际贸易	2020844003	79.66
4	2020844004	赵小瑜	国际贸易	2020844004	84.66

图 5.11 df1 和 df2 使用 join 方法合并结果

从结果可以看出,df1 作为待合并的左侧数据集,在其原“学号”列后增加了指定的字符“x”作为后缀,避免了在合并后的数据集中出现相同的“学号”列。

merge 方法是按照两个 DataFrame 对象的同名列连接合并。和 join 方法不同,它默认要求待合并的两个 DataFrame 必须有相同的列,如果没有同名列,在合并时要设置 left_index 和 right_index 参数的值为 True。下面是 merge 方法的语法格式:

```
pandas.merge(left,right,how = 'inner',on = None,left_on = None,right_on = None,left_index =
False, right_index = False, sort = False, suffixes = ('_x', '_y'))
```

主要参数的说明如下。

(1) left:接收 DataFrame 或 Series,表示要添加的新数据,无默认值。

(2) right:接收 DataFrame 或 Series,表示要添加的新数据,无默认值。

(3) how:接收 inner、outer、left、right,表示数据的连接方式,默认为 inner。

(4) on:接收 string 或 sequence,表示两个数据合并的主键(必须一致),默认为 None。

(5) left_on:接收 string 或 sequence,表示 left 参数接收数据用于合并的主键,默认为 None。

(6) right_on:接收 string 或 sequence,表示 right 参数接收数据用于合并的主键,默认为 None。

(7) left_index:接收 boolean,表示是否将 left 参数接收数据的 index 作为连接主键,默认为 False。

(8) right_index:接收 boolean,表示是否将 right 参数接收数据的 index 作为连接主键,默认为 False。

(9) sort:接收 boolean,表示是否根据连接主键对合并后的数据进行排序,默认为 False。

(10) suffixes:接收 tuple,表示用于追加到 left 和 right 参数接收数据重叠列名的尾缀默认为('_x', '_y')。

【例 5.10】 用 merge 方法按列合并数据集。

按列合并 df1 和 df2:

```
df1.merge(df2)
```

df1 和 df2 按照同名列进行连接,自动删除同名列,结果如图 5.12 所示。

在默认情况下,如果待合并的数据集中没有同名的列,用 merge 方法合并会报错,例如:

```
df1.merge(df3)
```

报错信息如下：

```
...
-> 1035     lidx = self.left_index, ridx = self.right_index))
    1036   if not common_cols.is_unique:
    1037   raise MergeError("Data columns not unique: {common!r}"
MergeError: No common columns to perform merge on. Merge options: left_on = None, right_on = None,
left_index = False, right_index = False
```

如果要解决这个问题，可以设置 left_index 和 right_index 参数的值为 True，例如：

```
df1.merge(df3,left_index = True,right_index = True)
```

结果如图 5.13 所示。

	学号	姓名	专业	Python
0	2020802045	魏天	信息管理与信息系统	93.00
1	2020844001	郭夏	国际贸易	86.00
2	2020844002	王晓加	NaN	74.00
3	2020844003	黄婷婷	国际贸易	79.66
4	2020844004	赵小瑜	国际贸易	84.66

图 5.12 df1 和 df2 使用 merge 方法合并结果

	学号	姓名	专业	数学	选修
0	2020802045	魏天	信息管理与信息系统	90.8	95.0
1	2020844001	郭夏	国际贸易	83.4	100.0
2	2020844002	王晓加	NaN	83.4	NaN
3	2020844003	黄婷婷	国际贸易	91.4	95.0
4	2020844004	赵小瑜	国际贸易	82.2	100.0

图 5.13 df1 和 df3 使用 merge 方法合并结果

2）合并行

两个数据集按行合并，可以用 DataFrame 的 append 方法，下面是 append 方法的语法格式：

```
DataFrame.append(self, other, ignore_index = False, verify_integrity = False)
```

主要参数的说明如下。

（1）other：接收 DataFrame 或 Series，表示要添加的新数据，无默认值。

（2）ignore_index：接收 boolean，如果输入 True，会对新生成的 DataFrame 使用新的索引（自动产生）而忽略原来数据的索引。其默认为 False。

（3）verify_integrity：接收 boolean，如果输入 True，那么当 ignore_index 为 False 时会检查添加的数据索引是否冲突，如果冲突，则会添加失败。其默认为 False。

【例 5.11】 用 append 方法合并行。

按行合并 df1 和 df4：

```
df1.append(df4)
```

df1 和 df4 有相同列，合并结果如图 5.14 所示。

当待合并的两个数据集的列不相同时，使用 append 方法实现的是行、列并集的拼接，例如合并 df1 和 df3：

```
df1.append(df3)                    #列不相同,实现并集拼接
```

合并结果如图 5.15 所示。

	学号	姓名	专业
0	2020802045	魏天	信息管理与信息系统
1	2020844001	郭夏	国际贸易
2	2020844002	王晓加	NaN
3	2020844003	黄婷婷	国际贸易
4	2020844004	赵小瑜	国际贸易
20	2020844022	关帅	会计学
21	2020844023	刘嘉雯	会计学
22	2020844024	刘浩天	会计学
23	2020844025	刘宇	NaN
24	2020844026	胡童	会计学
25	2020844027	丁灿	会计学

图 5.14 df1 和 df4 使用 append 方法合并结果

	学号	姓名	专业	数学	选修
0	2.020802e+09	魏天	信息管理与信息系统	NaN	NaN
1	2.020844e+09	郭夏	国际贸易	NaN	NaN
2	2.020844e+09	王晓加	NaN	NaN	NaN
3	2.020844e+09	黄婷婷	国际贸易	NaN	NaN
4	2.020844e+09	赵小瑜	国际贸易	NaN	NaN
0	NaN	NaN	NaN	90.8	95.0
1	NaN	NaN	NaN	83.4	100.0
2	NaN	NaN	NaN	83.4	NaN
3	NaN	NaN	NaN	91.4	95.0
4	NaN	NaN	NaN	82.2	100.0

图 5.15 df1 和 df3 使用 append 方法合并结果

3) 拼接合并数据

如果不考虑待合并的数据集是否有相同的行或列索引,也就是实现数据集的拼接,可以使用 concat 方法。concat 方法的使用非常灵活,可以按行或按列合并。concat 方法的语法格式如下:

```
pandas.concat(objs, axis = 0, join = 'outer', join_axes = None, ignore_index = False, keys = None,
levels = None, names = None, verify_integrity = False, copy = True)
```

主要参数的说明如下。

(1) objs: 接收多个 Series、DataFrame 的组合,表示参与拼接的 pandas 对象的列表组合,无默认值。

(2) axis: 接收 0 或 1,表示连接的轴向,默认为 0,当 axis 为 0 时表示按行拼接,当 axis 为 1 时表示按列拼接。

(3) join: 接收 inner 或 outer,表示其他轴向上的索引是按交集(inner)还是按并集(outer)进行合并,其默认为 outer。

(4) join_axes: 接收 Index 对象,表示用于其他 n−1 条轴的索引,不执行并集/交集运算。

(5) ignore_index: 接收 boolean,表示是否不保留连接轴上的索引,产生一组新索引。其默认为 False。

(6) keys: 接收 sequence,表示与连接对象有关的值,用于形成连接轴向上的层次化索引。其默认为 None。

(7) levels: 接收包含多个 sequence 的 list,表示在指定 keys 参数后,指定用作层次化索引各级别上的索引。其默认为 None。

(8) names: 接收 list,表示在设置了 keys 和 levels 参数后用于创建分层级别的名称。其默认为 None。

(9) verify_integrity: 接收 boolean,表示是否检查结果对象新轴上的重复情况,如果发现则引发异常。其默认为 False。

【例 5.12】 用 concat 方法实现数据集的拼接。

df1 和 df2 按列拼接:

```
pd.concat([df1,df2],axis = 1)                    #按列拼接
```

拼接结果如图 5.16 所示。

	学号	姓名	专业	学号	Python
0	2020802045	魏天	信息管理与信息系统	2020802045	93.00
1	2020844001	郭夏	国际贸易	2020844001	86.00
2	2020844002	王晓加	NaN	2020844002	74.00
3	2020844003	黄婷婷	国际贸易	2020844003	79.66
4	2020844004	赵小瑜	国际贸易	2020844004	84.66

图 5.16 df1 和 df2 使用 concat 方法按列拼接结果

在 concat 方法中将要拼接的数据集以列表形式呈现，可以同时拼接两个以上的数据集，例如同时按行拼接 df1、df2、df3 数据集：

```
pd.concat([df1,df2,df3],axis = 0)                    # 按行拼接
```

拼接以后部分结果如图 5.17 所示。

	学号	姓名	专业	Python	数学	选修
0	2.020802e+09	魏天	信息管理与信息系统	NaN	NaN	NaN
1	2.020844e+09	郭夏	国际贸易	NaN	NaN	NaN
2	2.020844e+09	王晓加	NaN	NaN	NaN	NaN
3	2.020844e+09	黄婷婷	国际贸易	NaN	NaN	NaN
4	2.020844e+09	赵小瑜	国际贸易	NaN	NaN	NaN
0	2.020802e+09	NaN	NaN	93.00	NaN	NaN
1	2.020844e+09	NaN	NaN	86.00	NaN	NaN
2	2.020844e+09	NaN	NaN	74.00	NaN	NaN
3	2.020844e+09	NaN	NaN	79.66	NaN	NaN
4	2.020844e+09	NaN	NaN	84.66	NaN	NaN
0	NaN	NaN	NaN	NaN	90.8	95.0

图 5.17 df1、df2 和 df3 使用 concat 方法按行拼接结果

扫一扫

视频讲解

5.1.4 数据的增、删、改

对数据进行预处理时经常会涉及数据的增、删、改操作。

1. 数据增加

数据增加就是在原始表中增加不存在的列或者行。

1）增加列

增加列常用的方法有新建的列索引赋值方法和 insert 方法。

新建的列索引赋值方法是通过为 DataFrame 对象新建一个列索引，并对该列索引下的数据进行赋值操作，使用该方法可以实现在 DataFrame 的末尾新增一列。

【例 5.13】 在成绩表中增加“团员否”列，将其值设为 True。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df["团员否"] = True
df.head()
```

结果如图 5.18 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学	团员否
0	2020802045	魏天	男	信息管理与信息系统	67.116667	90.8	93.00	95.0	106.00	True
1	2020844001	郭夏	男	国际贸易	91.050000	83.4	86.00	100.0	99.00	True
2	2020844002	王晓加	男	NaN	54.200000	83.4	74.00	NaN	90.00	True
3	2020844003	黄婷婷	女	国际贸易	87.800000	91.4	79.66	95.0	92.66	True
4	2020844004	赵小瑜	NaN	国际贸易	61.150000	82.2	84.66	100.0	97.66	True

图 5.18 增加“团员否”列后的结果

如果要在指定位置增加列,可以使用 insert 方法,该方法的语法格式如下:

```
DataFrame.insert(loc,column,value,allow_duplicates = False)
```

常见参数的说明如下。

(1) loc: 接收 int,表示第几列,如果在第一列插入数据,则 loc=0。

(2) column: 接收 string,插入列的名称。

(3) value: 接收 boolean、string、array、Series 等类型数据,插入列的值。

(4) allow_duplicates:接收 boolean,表示是否允许列名重复,若设为 True,则允许新插入的列名和已存在的列名重复。

【例 5.14】 在成绩表的第 3 列增加“年龄”列,将其值设为 18。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df.insert(2,"年龄",18)
df.head()
```

结果如图 5.19 所示。

	学号	姓名	年龄	性别	专业	英语	数学	Python	选修	管理学
0	2020802045	魏天	18	男	信息管理与信息系统	67.116667	90.8	93.00	95.0	106.00
1	2020844001	郭夏	18	男	国际贸易	91.050000	83.4	86.00	100.0	99.00
2	2020844002	王晓加	18	男	NaN	54.200000	83.4	74.00	NaN	90.00
3	2020844003	黄婷婷	18	女	国际贸易	87.800000	91.4	79.66	95.0	92.66
4	2020844004	赵小瑜	18	NaN	国际贸易	61.150000	82.2	84.66	100.0	97.66

图 5.19 在指定位置增加“年龄”列后的结果

2) 增加行

用 loc 属性可以为 DataFrame 对象新建一个行索引,并对其赋值; 增加多行可以使用 append 方法。

【例 5.15】 为成绩表新增一行数据。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df.loc[57] = ["20200848045","王芳",10,"女","金融学",55,66,77,90]
df.tail()
```

结果如图 5.20 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
53	2020848027	热孜耶·买买提	女	金融学	92.700	93.2	86.32	100.0	99.32
54	2020848028	奴热艾力·雪艾力	女	金融学	15.000	75.0	63.32	100.0	76.32
55	2020848029	林可新	女	金融学	89.300	87.4	95.00	100.0	108.00
56	2020848031	任旭	女	金融学	83.425	85.4	71.66	100.0	84.66
57	20200848045	王芳	女	金融学	55.000	66.0	77.00	90.0	95.00

图 5.20 新增一行数据后的结果

【例 5.16】 为成绩表新增 3 行数据。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#提取前 3 行数据的学号、姓名和专业列作为数据集 df1
df1 = df[["学号","姓名","专业"]][:3]
#将 df1 添加到 df 中并查看后 5 行数据
df.append(df1).tail()
```

结果如图 5.21 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
55	2020848029	林可新	女	金融学	89.300	87.4	95.00	100.0	108.00
56	2020848031	任旭	女	金融学	83.425	85.4	71.66	100.0	84.66
0	2020802045	魏天	NaN	信息管理与信息系统	NaN	NaN	NaN	NaN	NaN
1	2020844001	郭夏	NaN	国际贸易	NaN	NaN	NaN	NaN	NaN
2	2020844002	王晓加	NaN	NaN	NaN	NaN	NaN	NaN	NaN

图 5.21 新增 3 行数据后的结果

注意：使用 append 方法不会直接在原数据集 df 中增加新列，如果需要使用新增后的数据集，建议将其保存为新的 DataFrame 对象。

2. 数据修改

数据修改通过抽取 DataFrame 要修改的列或行直接赋新值。

【例 5.17】 修改成绩表的数据。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#修改年龄列
df["年龄"] = 25
#修改第 57 行
df.loc[57] = ["20200848043","刘云","女","金融学",75,86,87,96,95]
#修改前 10 行的年龄列为 20
df["年龄"][:10] = 20
```

另外，如果要修改 DataFrame 的列名，可以用给 columns 属性直接赋新列名的方法，也可以用 rename 方法修改指定列名。rename 方法的语法格式如下：

```
DataFrame.rename(index = None, columns = None)
```

主要参数的说明如下。

(1) index：行标签参数,用于修改行名,需要传入一个字典或函数。

(2) columns：列标签参数,用于修改列名,需要传入一个字典或函数。

【例 5.18】 修改成绩表的列名。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#修改 columns 属性
df.columns = ['id', '姓名', 'age', '性别', '专业', '英语', '数学', 'Python', '选修', '团员否']
#使用 rename 方法修改指定列名
df1 = df.rename(columns = {"id":"我的学号"})
```

注意：在修改列名时用 columns 属性赋值的方法需要给出全部列名,包括要修改的列名和未修改的列名；用 rename 方法可以修改部分列名,也可以修改部分行名。

3. 数据删除

删除数据可以使用 drop 方法,其语法格式如下：

```
DataFrame.drop (labels = None,axis = 0,index = None,columns = None,inplace = False)
```

常见参数的说明如下。

(1) labels：表示行标签或列标签。

(2) axis：表示轴向,0 为按行删除,1 为按列删除,默认值为 0。

(3) index：表示删除行,默认值为 None。

(4) columns：表示删除列,默认值为 None。

(5) inplace：表示是否在原表上进行操作,接收 boolean,默认值为 False。

注意：指定 index 是按行删除,指定 columns 是按列删除,通过同时指定 labels 和 axis 参数也可以删除行或列。

【例 5.19】 删除成绩表中的数据。

```
import pandas as pd
pd.set_option('display.unicode.east_asian_width', True)
df = pd.read_excel("tdata/cj.xlsx")                    #读取数据
#增加“团员否”列
df["团员否"] = True
#删除第 1、3、5 行数据
df.drop(index = [1,3,5])
#删除“团员否”列
df.drop(columns = ["团员否"])
#使用 labels 参数删除第 1 到第 19 行,axis 参数默认为 0
df.drop(labels = range(1,20))
```

注意：上述数据删除都是在 df 的副本上进行的,并未删除 df 本身的数据。如果要在 df 上进行删除操作,需要指定 inplace 参数值为 True,例如：

```
#使用 labels 参数,设置 axis 参数为 1,就地删除"选修"列
df.drop(labels = "选修",axis = 1,inplace = True)
```

5.1.5 数据转换

处理数据有时会遇到数据转换的问题，例如转换指定列的数据类型、将整个 DataFrame 或 Series 序列转换成列表或字典等类型。

1. 转换指定列的数据类型

转换 DataFrame 中某列的数据类型可以用 astype 方法，其语法格式如下：

```
DataFrame.astype(dtype, copy = True)
```

常见参数的说明如下。

(1) dtype：用一个 numpy.dtype 或 Python 类型将整个 pandas 对象转换为相同类型；或使用{col:dtype,...}将 DataFrame 的一个或多个列转换为 dtype 类型，其中 col 是列标签，而 dtype 是 numpy.dtype 或 Python 类型。

(2) copy：为 True 时返回一个副本，为 False 时在原始数据集上修改。

【例 5.20】 修改成绩表中指定列的数据类型。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#查看学号列的数据类型
print("学号列转换前:\n",df["学号"].dtypes)
#转换为 object 类型,并赋值给学号列
df["学号"] = df["学号"].astype(object)
print("学号列转换后:\n",df["学号"].dtypes)
#修改 df 中英语、数学列的数据类型为 int 类型
df = df.astype({"英语":int,"数学":int})
print("转换后 df 各列的数据类型:")
df.dtypes
```

结果显示为：

```
学号列转换前:
 int64
学号列转换后:
 object
转换后 df 各列的数据类型:
学号        object
姓名        object
性别        object
专业        object
英语        int32
数学        int32
Python     float64
选修        float64
管理学     float64
dtype: object
```

2. 将 DataFrame 转换成其他数据类型

DataFrame 可以整体转换成列表或字典。

(1) 转换成列表：使用 DataFrame 的 values 属性的 tolist 方法。

(2) 转换成字典：使用 DataFrame 的 to_dict 方法。

【例 5.21】 将 DataFrame 转换成列表和字典。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#将 df 转换成列表
list = df.values.tolist()
#查看转换后列表中第 5 到第 9 个元素
list[5:10]
```

显示结果如下：

```
[[2020844005, '辛禧', '男', '国际贸易', 65.125, 88.6, 68.0, 80.0, 81.0],
[2020844007, '王晨', '男', '国际贸易', 62.4, 80.0, 65.0, 90.0, 78.0],
[2020844008, '韩天', '男', '国际贸易', 96.25, 91.0, 85.0, 97.0, 98.0],
[2020844009, '刘玉', '女', '国际贸易', 89.05, 91.4, 80.32, 100.0, 93.32],
[2020844010, '谢亚鹏', '男', '市场营销', 70.5, 85.2, 60.0, 90.0, 73.0]]
```

将 df 转换成字典类型。

```
df.to_dict()
```

显示结果如下：

```
{'学号': {0: 2020802045,
  1: 2020844001,
  2: 2020844002,
...}
```

将 df 中的姓名和 Python 列转换为字典类型。

```
df[["姓名","Python"]].to_dict()
```

显示结果如下：

```
{'姓名': {0: '魏天',
  1: '郭夏',
  2: '王晓加',
...},
'Python': {0: 93.0,
  1: 86.0,
...}}
```

5.2 数据的描述性统计分析

数据的描述性统计分析是采用一些统计计算方法来概括事物的整体状况以及事物间的关联和类属关系。在对数据进行描述性统计分析之前按照某种依据对数据进行排序或排名，有助于从有序的角度把握数据的整体状况。

5.2.1 数据排序和排名

1. 数据排序

DataFrame 数据排序时主要采用的方法是 sort_values 方法，可以按照指定的行/列进行排序，语法格式如下：

```
DataFrame.sort_values(by,axis = 0,ascending = True, inplace = False, na_position = 'last')
```

主要参数的说明如下。

(1) by：接收 list、string，要排序的依据，无默认值。

(2) axis：接收 int，表示轴向，其中 0 表示行、1 表示列，默认为 0，即按行进行操作。

(3) ascending：接收 boolean 值或其列表，指定升序或降序，当指定多个排序时可以使用布尔值列表，默认为降序。

(4) inplace：接收 boolean，默认为 False，如果设置为 True 表示就地排序。

(5) na_position：空值 NaN 的位置，值为 first 表明空值显示在数据的开头，值为 last 表明空值显示在数据的最后。

【例 5.22】 对成绩表进行排序。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
```

按照选修课的成绩升序排序：

```
df.sort_values(by = "选修")
```

结果显示为：

```
    学号         姓名   性别   专业              英语    数学   Python   选修
37  2020848007   苏远   女     信息管理与信息系统  90.25  89.2   79.32    68.0
40  2020848011   张田田 女     信息管理与信息系统  91.20  89.6   96.32    77.0
                 …
```

按照选修课的成绩降序排序：

```
df.sort_values(by = "选修",ascending = False)
```

结果显示为：

```
    学号         姓名   性别   专业    英语    数学    Python   选修
29  2020848001   王春杨 女     会计学  88.10   89.8    84.00    100.0
27  2020844029   金耀   男     会计学  79.45   87.2    68.00    100.0
                 …
```

按照选修课和 Python 的成绩升序排序：

```
df.sort_values(by = ["选修","Python"])
```

结果显示为：

```
    学号        姓名   性别  专业              英语     数学   Python   选修
37  2020848007  苏远   女    信息管理与信息系统  90.25   89.2   79.32    68.0
40  2020848011  张田田 女    信息管理与信息系统  91.20   89.6   96.32    77.0
5   2020844005  辛禧   男    国际贸易          65.125   88.6   68.00    80.0
13  2020844014  刘欣语 男    市场营销          48.718   83.8   86.00    80.0
                …
```

按照选修升序排序,如该列中有空值,显示在开头:

```
df.sort_values(by = "选修",na_position = "first")
```

结果显示为:

```
    学号        姓名   性别  专业      英语    数学   Python   选修
2   2020844002  王晓加 男    NaN       54.20   83.4   74.00    NaN
10  2020844011  娄天楠 男    市场营销  58.80   84.6   60.00    NaN
                …
```

按行进行排序,则列之间应该是可比较的,这里提取 df 中的英语、数学、Python 和选修 4 列,按行进行排序:

```
dfs = df[["英语","数学","Python","选修"]]
dfs.sort_values(by = 1,axis = 1)
```

结果显示为:

```
    数学   Python   英语        选修
0   90.8   93.00    67.116667   95.0
1   83.4   86.00    91.050000   100.0
2   83.4   74.00    54.200000   NaN
           …
```

2. 数据排名

DataFrame 的 rank 方法可以实现按照 DataFrame 对象中的某些列进行排名,语法格式如下:

```
DataFrame.rank(axis = 0,method = 'average', numeric_only = None, ascending = True)
```

主要参数的说明。

(1) axis:接收 int,表示轴向,其中 0 表示行、1 表示列,默认为 0,即按行进行操作。

(2) method:在相同值情况下采用的排名方法,默认是 average(平均值),也可以是 max(最大值)、min(最小值)、first(原始顺序)、dense(密集)。

(3) numeric_only:接收 boolean,若设为 True,表示只对数字列进行排名。

(4) ascending:接收 boolean 值或其列表,指定升序或降序,若指定多个排序可以使用布尔值列表,默认为降序。

【例 5.23】 对成绩表进行排名操作。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
```

按选修列进行排名：

```
df[["选修"]].rank()
```

结果显示为：

```
   选修
0  12.0
1  34.0
2  NaN
3  12.0
4  34.0
…
```

按选修和 Python 列进行排名：

```
df[["选修","Python"]].rank()
```

结果显示为：

```
    选修    Python
0   12.0    52.5
1   34.0    37.0
2   NaN     13.5
3   12.0    21.0
4   34.0    32.0
…
```

rank 排名方法在相同值的情况下默认按照平均值进行排名，通过设置 method 参数可以选择不同的排名方法：

```
df["选修 Average 排名"] = df["选修"].rank()
df["选修 First 排名"] = df["选修"].rank(method = "first")
df["选修 Max 排名"] = df["选修"].rank(method = "max")
df["选修 Min 排名"] = df["选修"].rank(method = "min")
df[["学号","选修 Average 排名","选修 First 排名","选修 Max 排名","选修 Min 排名"]]
```

结果显示为：

```
    学号          选修 Average 排名   选修 First 排名   选修 Max 排名   选修 Min 排名
0   2020802045          12.0              11.0              13.0              11.0
1   2020844001          34.0              15.0              53.0              15.0
2   2020844002          NaN               NaN               NaN               NaN
3   2020844003          12.0              12.0              13.0              11.0
4   2020844004          34.0              16.0              53.0              15.0
…
```

3. 设置索引

设置索引能够快速地查询数据，实现数据的快速排序，也能在合并数据时实现数据的自动对齐。在 pandas 中提供了一系列方法设置和修改 DataFrame 数据对象的索引。

- reindex 方法：重设索引。
- set_index 方法：将某列设置为索引。
- reset_index 方法：重新设置连续索引。

1）reindex 方法

reindex 方法通过修改索引操作完成对现有数据的重新排序。

【例 5.24】 按选修列排序后修改 DataFrame 索引为 0～4 或 10～14。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
dfs = df.sort_values(by = "选修")
dfs.head()
```

按照选修列排序后结果如图 5.22 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
37	2020848007	苏远	女	信息管理与信息系统	90.250000	89.2	79.32	68.0	92.32
40	2020848011	张田田	女	信息管理与信息系统	91.200000	89.6	96.32	77.0	109.32
5	2020844005	辛禧	男	国际贸易	65.125000	88.6	68.00	80.0	81.00
13	2020844014	刘欣语	男	市场营销	48.718333	83.8	86.00	80.0	99.00
19	2020844021	李赫桐	男	会计学	88.276667	86.8	83.00	87.0	96.00

图 5.22 排序后的结果

修改行索引为 0～4：

```
dfs.reindex(range(0,5))
```

显示结果如图 5.23 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
0	2020802045	魏天	男	信息管理与信息系统	67.116667	90.8	93.00	95.0	106.00
1	2020844001	郭夏	男	国际贸易	91.050000	83.4	86.00	100.0	99.00
2	2020844002	王晓加	男	NaN	54.200000	83.4	74.00	NaN	90.00
3	2020844003	黄婷婷	女	国际贸易	87.800000	91.4	79.66	95.0	92.66
4	2020844004	赵小瑜	NaN	国际贸易	61.150000	82.2	84.66	100.0	97.66

图 5.23 修改行索引为 0～4 后的结果

修改索引为 10～14：

```
dfs.reindex(range(10,15))
```

显示结果如图 5.24 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
10	2020844011	娄天楠	男	市场营销	58.800000	84.6	60.00	NaN	73.00
11	2020844012	唐喆	男	市场营销	80.233333	87.4	64.00	100.0	77.00
12	2020844013	史昀	男	市场营销	82.733333	82.2	73.32	100.0	86.32
13	2020844014	刘欣语	男	市场营销	48.718333	83.8	86.00	80.0	99.00
14	2020844015	王同	男	市场营销	74.200000	92.2	92.00	100.0	115.00

图 5.24 修改行索引为 10～14 后的结果

2）set_index 方法

set_index 方法用于将 DataFrame 中的列转化为行索引，在转换之后，默认方法中的 drop 参数为 True，即原有的列会被删除，可以通过设置 drop 参数为 False 保留原来的列。

【例 5.25】 将姓名列设置为 DataFrame 的索引。

```
dfs.set_index("姓名",drop = False)
```

部分结果如图 5.25 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修
姓名								
苏远	2020848007	苏远	女	信息管理与信息系统	90.250000	89.2	79.32	68.0
张田田	2020848011	张田田	女	信息管理与信息系统	91.200000	89.6	96.32	77.0
辛禧	2020844005	辛禧	男	国际贸易	65.125000	88.6	68.00	80.0
刘欣语	2020844014	刘欣语	男	市场营销	48.718333	83.8	86.00	80.0
李赫桐	2020844021	李赫桐	男	会计学	88.276667	86.8	83.00	87.0

图 5.25 将姓名列设置为行索引后的结果

3）reset_index 方法

reset_index 方法用于重新设置 DataFrame 索引。在对数据表进行清洗操作之后，例如去重或删除了空值所在的行，行索引本身是不会发生变化的，会出现不连续情形；在对数据进行排序后，原始行索引也不会变化，会出现乱序情形，此时可以用 reset_index 方法重新设置索引。

【例 5.26】 在按选修列排序后重新设置 DataFrame 索引。

```
dfs = df.sort_values(by = "选修")
dfs.reset_index() .head()
```

结果如图 5.26 所示。

	index	学号	姓名	性别	专业	英语	数学	Python	选修
0	37	2020848007	苏远	女	信息管理与信息系统	90.250000	89.2	79.32	68.0
1	40	2020848011	张田田	女	信息管理与信息系统	91.200000	89.6	96.32	77.0
2	5	2020844005	辛禧	男	国际贸易	65.125000	88.6	68.00	80.0
3	13	2020844014	刘欣语	男	市场营销	48.718333	83.8	86.00	80.0
4	19	2020844021	李赫桐	男	会计学	88.276667	86.8	83.00	87.0

图 5.26 重新设置索引后的结果

5.2.2 常见的数据计算方法

本节介绍描述性统计分析中常用的数据计算方法。描述性统计分析是采用一些统计计算方法来概括事物的整体状况以及事物间的关联和类属关系。

从分析的角度而言，数据可以分为数值型数据和类别型数据两种。

- 数值型数据：比如表示成绩、年龄的数据，对其的描述统计主要包括了计算数值型数据的最小值、均值、中位数、最大值、方差、标准差等。
- 类别型数据：比如表示性别、年级的数据，对其的描述统计主要是分布情况，可以使用频数统计。

1. 数值型数据的描述统计

进行数值型数据的描述性统计分析既可以采用表 5.8 所示 numpy 库中的数值计算方法，也可以采用表 5.9 所示的 pandas 提供的数值计算方法。

表 5.8 numpy 库中常用的数值计算方法

方　法	说　明	方　法	说　明
np.min()	计算最小值	np.max()	计算最大值
np.mean()	计算均值	np.ptp()	计算极差
np.median()	计算中位数	np.std()	计算标准差
np.var()	计算方法	np.cov()	计算协方差

表 5.9 pandas 库中常用的数值计算方法

方　法	说　明	方　法	说　明
min()	计算最小值	max()	计算最大值
mean()	计算均值	quantile()	计算四分位数
median()	计算中位数	std()	计算标准差
var()	计算方差	cov()	计算协方差
count()	计算非空值数量	mode()	计算众数
skew()	计算样本偏度	kurt()	计算样本峰度

【例 5.27】 对成绩表中的数值列进行计算。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
dfs = df[["英语","数学","Python","选修"]]
```

计算各列的均值：

```
print("按列求均值\n",dfs.mean())
```

结果显示为：

```
按列求均值
英语        80.440877
数学        87.845614
Python     81.625965
选修        96.679245
dtype: float64
```

计算各行的均值，显示前 5 条记录：

```
print("按行求均值\n",dfs.mean(axis = 1) [:5])
```

结果显示为：

```
按行求均值
0    86.479167
1    90.112500
2    70.533333
```

```
3    88.465000
4    82.002500
dtype: float64
```

计算各列的中位数：

```
print("中位数:\n",dfs.median())
```

结果显示为：

```
中位数:
英语       83.75
数学       87.40
Python     83.00
选修       100.00
dtype: float64
```

计算各列的四分位数：

```
print("四分位数:\n",dfs.quantile(0.75))
```

结果显示为：

```
四分位数:
英语       89.55
数学       91.40
Python     89.00
选修       100.00
Name: 0.75, dtype: float64
```

计算各列及单独某数据列的众数：

```
print("四科成绩的众数\n",dfs.mode())
print("数学成绩的众数\n",dfs["数学"].mode())
```

结果显示为：

```
四科成绩的众数
     英语    数学    Python    选修
0    83.75   87.4    83.0      100.0
1    NaN     NaN     85.0      NaN
2    NaN     NaN     86.0      NaN
3    NaN     NaN     89.0      NaN
4    NaN     NaN     92.0      NaN
数学成绩的众数
0    87.4
dtype: float64
```

计算各列的方差：

```
print("方差:\n",dfs.var())
```

结果显示为：

```
方差:
英语        211.639194
数学         17.961454
Python      90.198842
选修         48.491292
dtype: float64
```

计算各列的标准差：

```
print("标准差:\n",dfs.std())
```

结果显示为：

```
标准差:
英语        14.547824
数学         4.238096
Python      9.497307
选修         6.963569
dtype: float64
```

计算各列的偏度和峰度：

```
print("样本偏度:\n",dfs.skew())              #计算各列的偏度
print("样本峰度:\n",dfs.kurt())              #计算各列的峰度
```

结果显示为：

```
样本偏度:
英语        -2.200623
数学        -0.393805
Python      -0.599544
选修        -2.454543
dtype: float64
样本峰度:
英语         6.655603
数学         0.103636
Python      -0.371648
选修         6.041652
dtype: float64
```

2. 类别型数据的描述统计

在 pandas 中，表示类别的特征可以用 object 字符串类型，比如性别是“男”还是“女”，专业是“市场营销”还是“金融”等，此时描述类别型数据的分布情况可以使用 pandas 库中的 value_counts 方法。

【例 5.28】 统计各个专业的人数。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df["专业"].value_counts()
```

结果显示为：

```
信息管理与信息系统    14
会计学                14
市场营销              10
金融学                9
国际贸易              7
Name: 专业, dtype: int64
```

另外，pandas 专门提供了一种特殊的类别类型——category 类型，以便于进行统计分类。可以用 astype 方法将 object 类型的数据转换为 category 类型，例如：

```
df["专业"].astype("category")
```

3. describe 方法

pandas 还提供了对数据进行描述性统计分析的 describe 方法，既适用于 Series 对象，也适用于 DataFrame 对象。

describe 方法是以列为单位进行统计分析，如果列是数值类型，进行的统计运算包括 count(求列元素的个数)、mean(求列数据均值)、std(求标准差)、min(求列数据的最小值)、max(求列数据的最大值)，以及求前 25%的数据均值、前 50%的数据均值、前 75%的数据均值；如果列是 object 类型或 category 类型，进行的统计运算包括 count(求列数据的元素个数)、unique(求列数据中元素的种类)、top(求列元素中出现频率最高的元素)、freq(求列元素中出现频率最高的元素的个数)。

【例 5.29】 对成绩表中的所有数值列进行描述性统计分析。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df.describe()
```

显示结果如图 5.27 所示。

	学号	英语	数学	Python	选修	管理学
count	5.700000e+01	57.000000	57.000000	57.000000	53.000000	57.000000
mean	2.020845e+09	80.440877	87.845614	81.625965	96.679245	95.117193
std	6.157525e+03	14.547824	4.238096	9.497307	6.963569	10.189566
min	2.020802e+09	15.000000	75.000000	60.000000	68.000000	73.000000
25%	2.020844e+09	75.625000	85.200000	75.000000	97.000000	89.000000
50%	2.020844e+09	83.750000	87.400000	83.000000	100.000000	96.000000
75%	2.020848e+09	89.550000	91.400000	89.000000	100.000000	102.000000
max	2.020848e+09	98.700000	95.000000	96.320000	100.000000	119.000000

图 5.27 成绩表中数值列的描述性统计结果

DataFrame 的 describe 方法默认只对数值型列进行统计分析，如果想对 object 类型或 category 类型进行统计分析，可以设置 describe 方法中的 include 参数，或者使用 Series 列调用 describe 方法。

【例 5.30】 对成绩表中所有的列进行描述性统计分析。

```
import pandas as pd
```

```
df = pd.read_excel("tdata/cj.xlsx")
df.describe(include = "all")
```

显示结果如图 5.28 所示。

	学号	姓名	性别	专业	英语	数学	Python	选修	管理学
count	5.700000e+01	57	54	54	57.000000	57.000000	57.000000	53.000000	57.000000
unique	NaN	54	2	5	NaN	NaN	NaN	NaN	NaN
top	NaN	陈小恬	男	信息管理与信息系统	NaN	NaN	NaN	NaN	NaN
freq	NaN	3	32	14	NaN	NaN	NaN	NaN	NaN
mean	2.020845e+09	NaN	NaN	NaN	80.440877	87.845614	81.625965	96.679245	95.117193
std	6.157525e+03	NaN	NaN	NaN	14.547824	4.238096	9.497307	6.963569	10.189566
min	2.020802e+09	NaN	NaN	NaN	15.000000	75.000000	60.000000	68.000000	73.000000
25%	2.020844e+09	NaN	NaN	NaN	75.625000	85.200000	75.000000	97.000000	89.000000
50%	2.020844e+09	NaN	NaN	NaN	83.750000	87.400000	83.000000	100.000000	96.000000
75%	2.020848e+09	NaN	NaN	NaN	89.550000	91.400000	89.000000	100.000000	102.000000
max	2.020848e+09	NaN	NaN	NaN	98.700000	95.000000	96.320000	100.000000	119.000000

图 5.28　成绩表中所有列的描述性统计结果

设置 include 参数为“all”,可以对 DataFrame 的全部列数据进行分析,也可以通过设置 include 参数为 number、category 或 np.object 指定对 DataFrame 中的数值类别列、类别类型列或字符串类型列的数据进行分析。

【例 5.31】 对成绩表中的 object 类型列数据进行描述性分析。

```
import numpy as np
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df.describe(include = np.object)
```

结果显示为:

```
          姓名    性别
count     57      54
unique    54      2
top       陈小恬  男
freq      3       32
```

对指定列的描述性统计分析可以先抽取列,再调用 Series 的 describe 方法。

【例 5.32】 对成绩表中的专业列进行描述性统计分析。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
df["专业"].describe()
```

结果显示为:

```
count                    54
unique                    5
top       信息管理与信息系统
freq                     14
Name: 专业, dtype: object
```

5.3 分组统计

数据的分组统计也叫分组聚合，是根据某个或某几个字段先对数据集进行分组，然后对各组应用一个函数进行聚合计算，是数据分析中的常见操作。pandas 提供了一个灵活、高效的 groupby 方法进行分组，配合 agg 或者 apply 方法，能够实现分组统计。图 5.29 显示了分组统计操作的实现原理，具体包括拆分、应用和合并 3 个步骤。

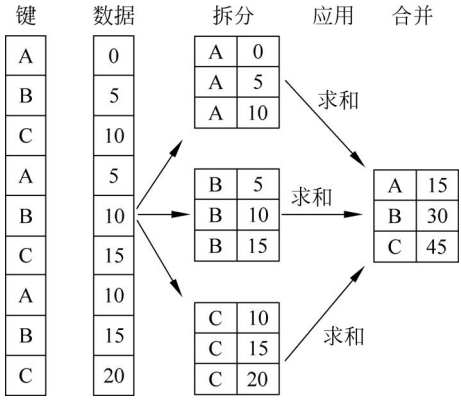


图 5.29 分组统计操作的实现原理

① 拆分：根据分组原则对某个或某几个字段进行拆分，将相同属性值的字段分成一组，例如按照 3 个属性值 A、B、C 拆分成 3 组。

② 应用：对拆分后的各组应用函数进行相应的转换操作，例如应用 sum 函数进行各组内的求和操作。

③ 合并：将结果合并成一个数据结构，例如汇总各组求和后的结果。

扫一扫

视频讲解

5.3.1 数据分组

按照一列或多列对数据进行分组（即拆分功能）是由 groupby 方法实现的，该方法的语法格式如下：

```
DataFrame.groupby(by, axis = 0, level = None, as_index = True, sort = True, group_keys = True,
squeeze = False, ** kwargs)
```

主要参数的说明如下。

（1）by：接收 list、string、mapping 或 generator，用于确定进行分组的依据，无默认值。

（2）axis：接收 int，表示操作的轴向，默认为 0，对列进行操作。

（3）level：接收 int 或者索引名，代表标签所在的级别，默认为 None。

（4）as_index：接收 boolean，表示聚合后的聚合标签是否以 DataFrame 索引形式输出，默认为 True。

（5）sort：接收 boolean，表示是否对分组依据的分组标签进行排序，默认为 True。

（6）group_keys：接收 boolean，表示是否显示分组标签的名称，默认为 True。

（7）squeeze：接收 boolean，表示是否在允许的情况下对返回的数据进行降维，默认为 False。

by 表示分组依据,是必填参数,常用的是给出列名表示的单个字符串或字符串列表。

分组后的结果是 groupby 对象,该对象存在于内存中,无法直接查看,如果输出,显示的是 groupby 对象的内存地址。

【例 5.33】 对成绩表进行分组操作,并查看分组对象。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
#按照性别列进行分组
g1 = df.groupby("性别")
print(g1)
#按照性别和专业列进行分组
g2 = df.groupby(["性别","专业"])
print(g2)
```

结果显示为:

```
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x118cf65c0>
<pandas.core.groupby.groupby.DataFrameGroupBy object at 0x1091fe748>
```

groupby 对象可以调用描述性统计方法,以便于返回各组的统计值。

【例 5.34】 对分组对象进行求和运算。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
g1 = df.groupby("性别")
g2 = df.groupby(["性别","专业"])
```

对按性别分组后的对象 g1 做求和运算:

```
g1.sum()
```

显示结果如图 5.30 所示。

对按性别和专业分组后的对象 g2 做求和运算:

```
g2.sum()
```

显示结果如图 5.31 所示。

性别	英语	数学	Python	选修	管理学
女	1830.675	1974.8	1835.88	2130.0	2121.88
男	2521.055	2770.2	2583.14	2694.0	3027.14

图 5.30 按性别分组求和后的结果

性别	专业	英语	数学	Python	选修	管理学
女	会计学	430.650000	466.4	432.32	500.0	497.32
	信息管理与信息系统	363.450000	355.0	336.64	345.0	388.64
	国际贸易	176.850000	182.8	159.98	195.0	185.98
	市场营销	82.750000	92.0	92.00	100.0	105.00
	金融学	692.525000	785.6	732.28	890.0	849.28
男	会计学	694.451667	694.8	666.00	687.0	785.00
	信息管理与信息系统	703.326667	778.8	764.82	775.0	881.82
	国际贸易	314.825000	343.0	304.00	367.0	356.00
	市场营销	679.051667	784.6	698.32	765.0	825.32

图 5.31 按性别和专业分组求和后的结果

5.3.2 分组聚合

agg 方法和 apply 方法是两个常见的聚合方法，它们既可以作用于整个 DataFrame 对象，也可以作用于 groupby 分组对象。当作用于 groupby 分组对象时具有分组聚合的功能。

1. agg 方法

agg 方法是 aggregate 方法的简写形式，支持对每个分组应用某个函数，不同的分组可以应用不同的函数，该方法的语法格式如下：

```
DataFrame.agg(func, axis = 0, *args, **kwargs)
```

主要参数的说明如下。

(1) func：接收 list、dict、function，表示应用于每行/每列的函数，无默认值。

(2) axis：接收 0 或 1，代表操作的轴向，默认为 0，对列进行操作。

agg 方法中接收的分组聚合函数可以是以下之一：

- Python 内置的数学函数；
- numpy 库中提供的数学运算函数；
- 用户自定义函数。

下面以成绩表的性别分组对象 g1 为例了解 agg 方法的具体用法。

【例 5.35】 使用 agg 方法实现分组聚合运算。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
g1 = df.groupby("性别")
```

对分组对象进行聚合运算，将聚合函数名以字符串形式作为 agg 方法的参数：

```
g1.agg("sum")                #求男、女分组各科成绩的总和
```

显示结果如图 5.32 所示。

性别	英语	数学	Python	选修	管理学
女	1830.675	1974.8	1835.88	2130.0	2121.88
男	2521.055	2770.2	2583.14	2694.0	3027.14

图 5.32 男、女分组各科成绩的求和结果

对分组对象进行多个聚合运算，将多个聚合函数名以列表元素的形式作为 agg 方法的参数，如果使用的是内置的函数，则函数名以字符串形式给出，例如：

```
g1.agg(["sum","mean","max"])          #求男、女分组各科成绩的总和、均值和最大值
```

如果使用的是 numpy 库中的数据运算函数，则函数名直接给出，例如：

```
g1.agg([np.sum,np.mean,np.max])
```

上述显示结果均如图 5.33 所示。

对分组对象的不同列进行不同运算，例如对某个列求均值，对另一列求最大值，可以使用字典的方法，将两个列名分别作为 key，把聚合函数分别作为 value，例如：

```
funcDict = {"英语":np.mean ,"数学":max}
g1.agg(funcDict)
```

	英语			数学			Python			选修			管理学		
	sum	mean	max	sum	mean	max	sum	mean	max	sum	mean	max	sum	mean	max
性别															
女	1830.675	83.212500	93.1	1974.8	89.763636	94.8	1835.88	83.449091	96.32	2130.0	96.818182	100.0	2121.88	96.449091	109.32
男	2521.055	78.782969	98.7	2770.2	86.568750	95.0	2583.14	80.723125	96.00	2694.0	96.214286	100.0	3027.14	94.598125	119.00

图 5.33 男、女分组各科成绩求总和、均值和最大值的结果

显示结果如图 5.34 所示。

如果对分组对象的某个列求多个统计值,对另一列求一个统计值,此时只需要将字典对应 key 的 value 变为列表,列表元素为多个目标的聚合函数即可,例如:

```
funcDict = {"英语":[np.mean,max],"数学":max}
g1.agg(funcDict)
```

结果如图 5.35 所示。

	英语	数学
	mean	max
性别		
女	83.212500	94.8
男	78.782969	95.0

图 5.34 男、女分组对英语求均值、对数学求最大值的结果

	英语		数学
	mean	max	max
性别			
女	83.212500	93.1	94.8
男	78.782969	98.7	95.0

图 5.35 男、女分组对英语求均值和最大值、对数学求最大值的结果

agg 方法可传入自定义的函数,例如:

```
def doubleSum(data) :                    #求和的两倍
    return data.sum() * 2
g1.agg(doubleSum)
```

结果如图 5.36 所示。

	学号	姓名	英语	数学	Python	选修	管理学
性别							
女	88917288614	黄婷婷刘玉张析王春杨陈小恬陈小恬陈小恬张淳郑彤苏远方雨桃张田田贾晶晶张雨桐孟德坤王少祖黄金雨...	3661.35	3949.6	3671.76	4260.0	4243.76
男	129333997064	魏天郭夏王晓加辛禧王晨韩天谢亚鹏娄天楠唐喆史昀刘欣语王同武天一陈雨涵张家齐李赫桐刘嘉雯刘浩天...	5042.11	5540.4	5166.28	5388.0	6054.28

图 5.36 男、女分组各字段两倍求和的结果

2. apply 方法

apply 方法支持对每个分组应用某个函数,该方法的语法格式如下:

```
DataFrame.apply(func, axis = 0, broadcast = False, raw = False, reduce = None, ** kwds)
```

主要参数的说明如下。

(1) func:接收 functions,表示应用于每行/列的函数,无默认值。

(2) axis:接收 0 或 1,代表操作的轴向,默认为 0,对列进行操作。

(3) broadcast:接收 boolean,表示是否进行广播,默认为 False。

(4) raw:接收 boolean,表示是否直接将 ndarray 对象传递给函数,默认为 False。

(5) reduce:接收 boolean 或者 None,表示返回值的格式,默认为 None。

apply 方法将当前分组后的数据一起传入，可以返回多维数据。下面仍以成绩表的性别分组对象 g1 为例了解 apply 方法的具体用法。

【例 5.36】 使用 apply 方法实现分组聚合运算。

```
import pandas as pd
df = pd.read_excel("tdata/cj.xlsx")
g1.apply(sum)
```

结果如图 5.37 所示。

	学号	姓名	性别	英语	数学	Python	选修	管理学
性别								
女	44458644307	黄婷婷刘玉张析王春杨陈小恬陈小恬陈小恬张淳郑彤苏远方雨桃张田田贾晶晶张雨桐孟德坤王少祖黄金雨...	女女女女女女女女女女女女女女女女女女女女女	1830.675	1974.8	1835.88	2130.0	2121.88
男	64666998532	魏天郭夏王晓加辛禧王晨韩天谢亚鹏娄天楠唐喆史昀刘欣语王同武天一陈雨涵张家齐李赫桐刘嘉雯刘浩天...	男男男男男男男男男男男男男男男男男男男男男男男男男男男男男男...	2521.055	2770.2	2583.14	2694.0	3027.14

图 5.37　使用 apply 方法进行 sum 聚合后的结果

从上述结果可以看出，apply 将 sum 聚合函数作用在分组后的每一列，对于数值型的数据，计算每个元素的累计和；对于字符串类型的数据，则将每个元素进行连接。

如果只想显示数值类型列，例如英语、数学、Python 列的聚合结果，可以针对 g1 对象做抽取，也可以在应用 apply 方法进行聚合以后做抽取，例如：

```
g1[["英语","数学","Python"]].apply(sum)               #先抽取再聚合
g1.apply(sum)[["英语","数学","Python"]]               #先聚合再抽取
```

上述结果显示如图 5.38 所示。

	英语	数学	Python
性别			
女	1830.675	1974.8	1835.88
男	2521.055	2770.2	2583.14

图 5.38　抽取数值型数据用 apply 方法聚合后的结果

另外，apply 方法同样可以使用自定义函数进行聚合运算，其用法和 agg 方法相同。

5.4　实战：豆瓣读书 Top250 的数据表分析

本节以从互联网上爬取并存储在 CSV 文件中的豆瓣读书排行榜 Top250 的数据为例进行数据表的分析实战，图 5.39 所示为数据集的示例。

A	B	C	D	E	F	G	H	I	J	K	L	M
	书名	封面	出版信息	评分	评价人数	书籍详情	备注	作者	译作者	出版社	出版时间	定价
0	红楼梦	https://img	[清] 曹雪	9.6	344508人评价	https://boo	都云作者痴	[清] 曹雪芹 著		人民文学出版社	1996-12	59.7
1	活着	https://img	余华 / 作	9.4	616965人评价	https://boo	生的苦难与	余华		作家出版社	2012-8-1	20
2	百年孤独	https://img	[哥伦比亚	9.3	345801人评价	https://boo	魔幻现实主	[哥伦比亚	范晔	南海出版公司	2011-6	39.5
3	1984	https://img	[英] 乔治	9.4	189562人评价	https://boo	栗树荫下，	[英] 乔治	刘绍铭	北京十月文艺出版社	2010-4-1	28
4	飘	https://img	[美国] 玛	9.3	181850人评价	https://boo	革命时期的	[美国] 玛	李美华	译林出版社	2000-9	40
5	三体全集	https://img	刘慈欣 /	9.4	103274人评价	https://boo	地球往事三	刘慈欣		重庆出版社	2012-1-1	168
6	三国演义	https://img	[明] 罗贯	9.3	140077人评价	https://boo	是非成败转	[明] 罗贯中		人民文学出版社	1998-05	39.5
7	白夜行	https://img	[日] 东野	9.1	360187人评价	https://boo	一宗离奇命	[日] 东野	刘姿君	南海出版公司	2013-1-1	39.5
8	房思琪的初	https://img	林奕含 /	9.2	263416人评价	https://boo	向死而生的	林奕含		北京联合出版公司	2018-2	45
9	福尔摩斯探	https://img	[英] 阿·	9.3	108607人评价	https://boo	名侦探的代	[英] 阿·	丁钟华 等	群众出版社		
10	小王子	https://img	[法] 圣埃	9.0	649580人评价	https://boo	献给长成了	[法] 圣埃	马振聘	人民文学出版社	2003-8	22

图 5.39　豆瓣读书排行榜 Top250 数据集

针对豆瓣读书排行榜的数据,大家会有一些分析需求,例如想了解排行榜上图书的平均评分、想查看最受关注的图书信息或者想了解各个出版社各有多少上榜图书等。由于数据是从互联网上爬取而来的,数据往往含有噪声,也存在不一致的情况,一般不能直接用来进行数据分析,在分析前通常需要对数据进行预处理。

扫一扫

视频讲解

5.4.1 数据预处理

对爬取下来的豆瓣读书排行榜的数据集进行预处理操作,包括数据清洗、合并、拆分等。目前该数据存在出版信息列包括多特征信息、评价人数含有冗余字符等问题。

【预处理思路】:拆分出版信息列,将拆分后的列合并到原始数据集中,删除评价人数列中的冗余字符,转换具有数据特征的列,例如评价人数列的数据类型,具体的处理步骤如下。

(1) 读取数据集。
(2) 数据概览性分析。
(3) 拆分列。
(4) 合并列。
(5) 删除冗余列。
(6) 删除冗余字符。
(7) 数据类型转换。

1. 读取数据集

首先使用 pandas 库读取数据集,将其保存在 DataFrame 的对象 df 中,具体代码如下:

```
import pandas as pd
df = pd.read_excel("豆瓣.xlsx")                    #读取数据
```

2. 数据概览性分析

在进行数据预处理前,使用 DataFrame 的 info 和 head 方法对数据集 df 进行概览性分析,明确需要对数据集进行哪些预处理操作。

```
df.info()
df.head()
```

上述代码的显示结果如图 5.40 和图 5.41 所示。

通过概览性分析的显示结果可以看出,该数据集共有 250 个样本、8 列特征,其中 Unnamed 列为自动保存下来的数据集的行索引列,为 int64 类型,评分列为 float64 类型,其余列为 object 类型,除备注列以外,其他列不存在缺失值。通过观察数据,发现需要针对该数据集进行拆分列、合并列、删除列元素的冗余字符、转换数据类型以及删除冗余列等操作。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  250 non-null    int64
 1   书名          250 non-null    object
 2   封面          250 non-null    object
 3   出版信息        250 non-null    object
 4   评分          250 non-null    float64
 5   评价人数        250 non-null    object
 6   书籍详情        250 non-null    object
 7   备注          229 non-null    object
dtypes: float64(1), int64(1), object(6)
memory usage: 15.8+ KB
```

图 5.40 数据集的概览信息

3. 拆分列

出版信息列包括有关图书的多种信息,一般包括作者、译作者、出版社、出版时间和定价,这些信息之间多用“/”分隔,但具体图书的信息并不一致,大概能够覆盖 3 种情形,即包括 5 种

	Unnamed: 0	书名	封面	出版信息	评分	评价人数	书籍详情	备注
0	0	红楼梦	https://img1.doubanio.com/view/subject/s/publi...	[清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元...	9.6	344508人评价	https://book.douban.com/subject/1007305/	都云作者痴，谁解其中味?
1	1	活着	https://img9.doubanio.com/view/subject/s/publi...	余华 / 作家出版社 / 2012-8-1 / 20.00元	9.4	616965人评价	https://book.douban.com/subject/4913064/	生的苦难与伟大
2	2	百年孤独	https://img3.doubanio.com/view/subject/s/publi...	[哥伦比亚] 加西亚·马尔克斯 / 范晔 / 南海出版公司 / 2011-6 / 39.50...	9.3	345801人评价	https://book.douban.com/subject/6082808/	魔幻现实主义文学代表作
3	3	1984	https://img1.doubanio.com/view/subject/s/publi...	[英] 乔治·奥威尔 / 刘绍铭 / 北京十月文艺出版社 / 2010-4-1 / 28.0...	9.4	189562人评价	https://book.douban.com/subject/4820710/	栗树荫下，我出卖你，你出卖我
4	4	飘	https://img1.doubanio.com/view/subject/s/publi...	[美国] 玛格丽特·米切尔 / 李美华 / 译林出版社 / 2000-9 / 40.00元...	9.3	181850人评价	https://book.douban.com/subject/1068920/	革命时期的爱情，随风而逝

图 5.41 数据集的前 5 条数据

信息的、4 种信息的和 3 种信息的，因此需要对这些不同的情形进行分别处理。从通用性出发，将出版信息拆分为 5 列，如果有的图书信息缺失，则将其内容设置为空。其具体实现代码如下：

```
items = []                                  #存放每本图书的列表
for str in df["出版信息"]:
    item = []                               #存放一本图书信息的列表
    infos = str.split("/")                  #以"/"为分隔符进行拆分
    if len(infos) == 5:                     #具备 5 种信息
        item.append(infos[0])
        item.append(infos[1])
        item.append(infos[2])
        item.append(infos[3])
        item.append(infos[4])
    elif(len(infos) == 4) :                 #具备 4 种信息,缺失译作者
        item.append(infos[0])
        item.append("")
        item.append(infos[1])
        item.append(infos[2])
        item.append(infos[3])
    else:                                   #具备 3 种信息,缺失作者、译作者
        item.append("")
        item.append("")
        item.append(infos[0])
        item.append(infos[1])
        item.append(infos[2])
    items.append(item)
```

提取列表的前 5 条信息：

```
items[:5]
```

可以看到拆分后的结果为：

```
[['[清] 曹雪芹 著 ', '', '人民文学出版社 ', '1996-12 ', '59.70 元'],
['余华 ', '', '作家出版社 ', '2012-8-1 ', '20.00 元'],
['[哥伦比亚] 加西亚·马尔克斯 ', '范晔 ', '南海出版公司 ', '2011-6 ', '39.50 元'],
['[英] 乔治·奥威尔 ', '刘绍铭 ', '北京十月文艺出版社 ', '2010-4-1 ', '28.00'],
['[美国] 玛格丽特·米切尔 ', '李美华 ', '译林出版社 ', '2000-9 ', '40.00 元']]
```

4. 合并列

将拆分出来的信息合并到原始数据集中，先将二维列表 items 转换成带有列标题的 DataFrame 数据结构：

```
infoT = ["作者","译作者","出版社","出版时间","定价"]
dfinfo = pd.DataFrame(items,columns = infoT)
```

查看 dfinfo 的前 5 条数据：

```
dfinfo.head()
```

结果如图 5.42 所示。

	作者	译作者	出版社	出版时间	定价
0	[清] 曹雪芹 著		人民文学出版社	1996-12	59.70元
1	余华		作家出版社	2012-8-1	20.00元
2	[哥伦比亚] 加西亚·马尔克斯	范晔	南海出版公司	2011-6	39.50元
3	[英] 乔治·奥威尔	刘绍铭	北京十月文艺出版社	2010-4-1	28.00
4	[美国] 玛格丽特·米切尔	李美华	译林出版社	2000-9	40.00元

图 5.42 拆分后数据的结果

因为拆分后的 dfinfo 和 df 具有相同的行索引，所以可以使用 join 方法将 dfinfo 合并到 df 中，并查看合并结果：

```
df = df.join(dfinfo)
df.head(2)
```

查看前两条记录，合并后的结果如图 5.43 所示。

	书名	封面	出版信息	评分	评价人数	书籍详情	备注	作者	译作者	出版社	出版时间	定价
0	红楼梦	https://img1.doubanio.com/view/subject/s/publi...	[清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元...	9.6	344508人评价	https://book.douban.com/subject/1007305/	都云作者痴，谁解其中味?	[清] 曹雪芹 著		人民文学出版社	1996-12	59.70元
1	活着	https://img9.doubanio.com/view/subject/s/publi...	余华 / 作家出版社 / 2012-8-1 / 20.00元	9.4	616965人评价	https://book.douban.com/subject/4913064/	生的苦难与伟大	余华		作家出版社	2012-8-1	20.00元

图 5.43 合并列后的结果

5. 删除冗余列

在合并列后，数据集中原始的出版信息列成为冗余列，另外读取数据集时自动生成的Unnamed:0列也是冗余列，都可以使用drop方法删除。

```
df.drop(labels = "出版信息",axis = 1,inplace = True)
df.drop(labels = "Unnamed: 0",axis = 1,inplace = True)
```

注意：在drop方法中设置了inplace参数为True，表明在df数据集中直接删除。

6. 删除冗余字符

为便于后续的数据分析，需要修改一些列，包括去掉“评价人数”列中的“人评价”3个字、去掉“定价”列中的非数字和小数点的其他字符。

(1) “评价人数”列中所有的数据元素都统一成以“人评价”结尾，可以直接使用字符串的replace方法将冗余字符串替换为空。

(2) “定价”列中有些数据元素的后面带有“元”字符，有些没有，还有些数据元素的前面带有“CNY”字符，为便于统一操作，使用字符串的extract方法书写正则表达式提取数字和小数点字符。

其具体代码如下：

```
df["评价人数"] = df["评价人数"].str.replace("人评价"," ")
df["定价"] = df["定价"].str.extract(r"(\d + .\d + ) ")
df.head(2)
```

查看前两条记录，结果如图5.44所示。

	书名	封面	出版信息	评分	评价人数	书籍详情	备注	作者	译作者	出版社	出版时间	定价
0	红楼梦	https://img1.doubanio.com/view/subject/s/publi...	[清] 曹雪芹 著 / 人民文学出版社 / 1996-12 / 59.70元...	9.6	344508	https://book.douban.com/subject/1007305/	都云作者痴，谁解其中味?	[清]曹雪芹著		人民文学出版社	1996-12	59.70
1	活着	https://img9.doubanio.com/view/subject/s/publi...	余华 / 作家出版社 / 2012-8-1 / 20.00元	9.4	616965	https://book.douban.com/subject/4913064/	生的苦难与伟大	余华		作家出版社	2012-8-1	20.00

图5.44 删除冗余字符后的结果

7. 数据类型转换

在完成上述处理操作后，再次调用df的info方法对数据进行概览性分析。

```
df.info()
```

显示结果如图5.45所示。

可以看出评价人数列和定价列均为object类型，根据分析需求，应将这两列的数据类型转换为数值类型，这里转换为float64类型。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   书名      250 non-null    object
 1   封面      250 non-null    object
 2   评分      250 non-null    float64
 3   评价人数    250 non-null    object
 4   书籍详情    250 non-null    object
 5   备注      229 non-null    object
 6   作者      250 non-null    object
 7   译作者     250 non-null    object
 8   出版社     250 non-null    object
 9   出版时间    250 non-null    object
 10  定价      246 non-null    object
dtypes: float64(1), object(10)
memory usage: 21.6+ KB
```

图 5.45　处理后的数据集概览

```
df["评价人数"] = df["评价人数"].astype("float64")
df["定价"] = df["定价"].astype("float64")
```

至此完成了数据集的清洗和转换工作,为方便今后分析使用,可以将预处理后的数据保存下来,例如将其保存为 douban250.xlsx 文件。

```
df.to_excel("douban250.xlsx")
```

【实战案例代码 5.1】 豆瓣读书数据集的预处理。

```
import pandas as pd
#读取数据
df = pd.read_excel("豆瓣.xlsx")
#概览性分析
df.info()
df.head()
#拆分列
items = []
for str in df["出版信息"]:
    item = []
    infos = str.split("/")
    if len(infos) == 5:
        item.append(infos[0])
        item.append(infos[1])
        item.append(infos[2])
        item.append(infos[3])
        item.append(infos[4])
    elif(len(infos) == 4):
        item.append(infos[0])
        item.append("")
        item.append(infos[1])
        item.append(infos[2])
        item.append(infos[3])
    else:
        item.append("")
        item.append("")
        item.append(infos[0])
```

```
            item.append(infos[1])
            item.append(infos[2])
        items.append(item)
items[:5]
#合并列
infoT = ["作者","译作者","出版社","出版时间","定价"]
dfinfo = pd.DataFrame(items,columns = infoT)
df = df.join(dfinfo)
df.head()
#删除冗余列
df.drop(labels = "出版信息",axis = 1,inplace = True)
df.drop(labels = "Unnamed: 0",axis = 1,inplace = True)
#删除冗余字符
df["评价人数"] = df["评价人数"].str.replace("人评价"," ")
df["定价"] = df["定价"].str.extract(r"(\d+.\d+)")
df.head()
#数据类型转换
df.info()
df["评价人数"] = df["评价人数"].astype("float64")
df["定价"] = df["定价"].astype("float64")
#存储预处理后的数据
df.to_excel("douban250.xlsx")
```

扫一扫

视频讲解

5.4.2 数据分析

数据预处理后，如果用户想更深入地掌握排行榜上榜图书的信息，可以进行相关分析。

【分析思路】 数据分析往往要围绕用户需求。针对豆瓣读书排行榜的数据，读者可能会关心评分高的图书的信息，例如用户评分最高的图书；出版社可能会关心同行的相关信息，例如各个出版社上榜图书的平均单价等。此处模拟一些用户需求进行相关分析，具体步骤如下：

（1）抽取数据。

（2）根据评分进行图书信息分析。

（3）根据出版社进行分组分析。

1. 抽取数据

考虑满足用户需求并不需要数据集的全部列，在进行数据分析之前首先进行特征抽取，将抽取后的数据保存在数据集 dfs 中。

```
df = pd.read_excel("douban250.xlsx")
dfs = df[["书名","评分","评价人数","作者","译作者","出版社","出版时间","定价"]]
```

2. 根据评分进行图书信息分析

针对 dfs 数据集，围绕评分列进行分析处理，匹配读者的需求，例如进行以下分析：

（1）查看排行榜上榜图书的用户平均评分；

（2）查看所有高于平均分的图书的信息；

(3) 查看用户评分最高的图书的信息；

(4) 按照评分和评价人数降序查看排行榜上的数据。

查看排行榜图书的平均评分即为计算评分列的平均值。

```
print("排行榜平均评分",dfs["评分"].mean())
```

结果显示为：

```
排行榜平均评分 8.918799999999996
```

查看所有高于平均分的图书的信息即为筛选出大于平均分评分的所有行。

```
dfs.loc[dfs["评分"]>dfs["评分"].mean() ,]
```

结果如图 5.46 所示。

	书名	评分	评价人数	作者	译作者	出版社	出版时间	定价
0	红楼梦	9.6	344508	[清] 曹雪芹 著	NaN	人民文学出版社	1996-12	59.7
1	活着	9.4	616965	余华	NaN	作家出版社	2012-8-1	20.0
2	百年孤独	9.3	345801	[哥伦比亚] 加西亚·马尔克斯	范晔	南海出版公司	2011-6	39.5
3	1984	9.4	189562	[英] 乔治·奥威尔	刘绍铭	北京十月文艺出版社	2010-4-1	28.0
4	飘	9.3	181850	[美国] 玛格丽特·米切尔	李美华	译林出版社	2000-9	40.0
...	...	...	...	...	...	...	...	...
221	众病之王	9.1	10420	[美] 悉达多·穆克吉	李虎	中信出版社	2013-2	42.0
222	象棋的故事	9.1	10464	[奥] 斯蒂芬·茨威格	张玉书	上海译文出版社	2007-7	23.0
233	毛姆短篇小说精选集	9.1	10428	[英] 威廉·萨默塞特·毛姆	冯亦代	译林出版社	2012-11	36.0
234	长袜子皮皮	9.0	11202	[瑞典] 阿斯特丽德·林格伦	李之义	中国少年儿童出版社	1999-3	17.8
237	牡丹亭	9.0	11549	汤显祖	NaN	人民文学出版社	1963-4-1	14.5

115 rows × 8 columns

图 5.46 高于平均分的图书的信息

查看用户评分最高的图书信息也是根据评分列对行进行筛选。

```
dfs.loc[dfs["评分"] == dfs["评分"].max() ,]
```

结果如图 5.47 所示。

	书名	评分	评价人数	作者	译作者	出版社	出版时间	定价
21	哈利·波特	9.7	53570	J.K.罗琳 (J.K.Rowling)	苏农	人民文学出版社	2008-12-1	498.0

图 5.47 用户评分最高的图书信息

按照评分和评价人数降序查看排行榜上的图书数据，同时为了让排序后的数据索引有序，使用 reset_index 方法重新设置索引。

```
dfs.sort_values(by = ["评分","评价人数"],ascending = False) .reset_index()
```

结果如图 5.48 所示。

3. 根据出版社进行分组分析

针对 dfs 数据集，按出版社进行分组后进行分析处理，匹配出版社的需求，例如进行以下分析：

	index	书名	评分	评价人数	作者	译作者	出版社	出版时间	定价
0	21	哈利·波特	9.7	53570	J.K.罗琳 (J.K.Rowling)	苏农	人民文学出版社	2008-12-1	498.0
1	0	红楼梦	9.6	344508	[清] 曹雪芹 著	NaN	人民文学出版社	1996-12	59.7
2	63	艺术的故事	9.6	19358	[英] 贡布里希 (Sir E.H.Gombrich)	范景中	广西美术出版社	2008-04	280.0
3	57	史记 (全十册)	9.5	20046	司马迁	(索隐) 司马贞, (正义) 张守节	中华书局	1982-11	125.0
4	1	活着	9.4	616965	余华	NaN	作家出版社	2012-8-1	20.0
...	...	...	...	...	...	...	...	...	...
245	241	心是孤独的猎手	8.5	29655	[美] 卡森·麦卡勒斯	陈笑黎	上海三联书店	2005-8	25.0
246	230	告白	8.5	13584	[日]凑佳苗	丁世佳	哈尔滨出版社	2010-7	26.0
247	218	骆驼祥子	8.4	173428	老舍	NaN	人民文学出版社	2000-3-1	12.0
248	201	老人与海	8.4	171474	海明威	吴劳	上海译文出版社	1999-10	8.2
249	236	呼啸山庄	8.4	108578	艾米莉·勃朗特	张扬	人民文学出版社	1999-01-01	27.3

250 rows × 9 columns

图 5.48 排序后的图书信息

(1) 查看各个出版社上榜图书的情况；

(2) 了解各个出版社出版图书的平均单价；

(3) 总体查看各个出版社上榜图书的数量和平均单价。

查看各个出版社各有多少上榜图书，可以在分组后使用 count 方法统计每组中图书的数量，这里显示上榜图书数量最多的前十个出版社的信息。

```
dfs.groupby("出版社")["书名"].count().sort_values(ascending=False)[:10]
```

结果如图 5.49 所示。

了解各出版社上榜图书的平均单价，先按照出版社分组，再抽取定价列，对分组后的定价求均值，为便于查看，此处对分组聚合后的结果进行降序排序。

```
dfs.groupby("出版社")["定价"].mean().sort_values(ascending=False)
```

结果如图 5.50 所示。

```
出版社
人民文学出版社          39
上海译文出版社          27
生活·读书·新知三联书店    19
译林出版社              15
南海出版公司            14
北京十月文艺出版社      11
广西师范大学出版社      10
上海人民出版社           6
哈尔滨出版社             6
作家出版社               5
Name: 书名, dtype: int64
```

图 5.49 上榜图书数量最多的前十个出版社的信息

```
出版社
中国海关出版社          358.20
广西美术出版社          280.00
浙江教育出版社          168.00
重庆出版社              118.00
广州出版社 花城出版社    108.00
                         ...
漓江出版社                3.95
安徽文艺出版社            NaN
湖南文艺出版社            NaN
S.A.阿列克谢耶维奇        NaN
[英] 阿·柯南道尔          NaN
Name: 定价, Length: 81, dtype: float64
```

图 5.50 各出版社上榜图书的平均单价

查看各个出版社上榜图书的数量和平均单价，采用分组后对书名和定价做聚合运算的方法来实现。

```
dfs.groupby("出版社").agg({"书名":"count","定价":"mean"}).sort_values(by="书名",
ascending=False)
```

显示结果如图 5.51 所示。

出版社	书名	定价
人民文学出版社	39	37.972821
上海译文出版社	27	23.315926
生活·读书·新知三联书店	19	39.105263
译林出版社	15	27.320000
南海出版公司	14	30.307143
...	...	...
广州出版社 花城出版社	1	108.000000
广西美术出版社	1	280.000000
当代世界出版社	1	20.000000
文汇出版社	1	25.000000
少年儿童出版社	1	30.000000

81 rows × 2 columns

图 5.51 查看各个出版社上榜图书的数量和平均单价

【实战案例代码 5.2】 豆瓣读书排行榜的数据分析。

```
import pandas as pd
df = pd.read_excel("douban250.xlsx")
#抽取列
dfs = df[["书名","评分","评价人数","作者","译作者","出版社","出版时间","定价"]]
#了解排行榜图书的平均评分
print("排行榜平均评分",df["评分"].mean())
#查看所有高于平均分的图书的信息
df.loc[df["评分"]> df["评分"].mean(),]
#查看最受关注的图书的信息
df.loc[df["评分"] == df["评分"].max(),]
#根据评分和评价人数对排行榜的数据进行重新排序
df.sort_values(by = ["评分","评价人数"],ascending = False).reset_index()
#查看各个出版社各有多少上榜图书
df.groupby("出版社")["书名"].count().sort_values(ascending = False)[:10]
#查看各个出版社所出版图书的平均单价
df.groupby("出版社")["定价"].mean().sort_values(ascending = False)
#查看各个出版社的图书的数量和平均单价
df.groupby("出版社").agg({"书名":"count","定价":"mean"}).sort_values(by = "书名",ascending =
False)
```

本章小结

本章首先介绍了如何使用 pandas 库进行数据概览及预处理,包括数据概览分析的属性和方法、数据清洗的方法、抽取与合并的方法、数据的增/删/改和数据类型转换的方法;然后介绍了数据描述性统计分析的方法,包括如何进行数据的排序和排名、常用的数据计算方法;在介绍完数据的分组统计(包括数据分组和分组聚合运算)后,以豆瓣读书 Top250 的数据集为例进行了数据表分析,包括数据预处理和模拟用户需求进行的相关数据分析。

习题 5

扫一扫

习题

扫一扫

自测题