第 5 章

Pandas 数据获取与清洗

对于数据分析而言，数据大部分来源于外部数据，如常用的 CSV 文件、Excel 文件和数据库文件等。Pandas 库将外部数据转换为 DataFrame 数据格式，处理完成后再存储到相应的外部文件中。

前期采集到的数据，或多或少都存在一些瑕疵和不足，如数据缺失、极端值、数据格式不统一等问题。数据预处理不仅可以提高初始数据的质量，保留与分析目标联系紧密的数据，而且可以优化数据的表现形式，有助于提高数据分析或数据挖掘工作的效率和准确率。

数据清洗主要是将“脏”数据变成“干净”数据的过程，该过程中会通过一系列的方法对“脏”数据进行处理，以达到清除冗余数据、规范数据、纠正错误数据的目的。

接下来，本节将针对 Pandas 中数据获取、清洗与格式化处理的内容进行详细讲解。

5.1 数据获取操作

在对数据进行分析时，通常不会将需要分析的数据直接写入程序中，这样不仅造成程序代码臃肿，而且可用率很低。常用的方法是将需要分析的数据存储到本地，之后再对存储文件进行读取。

初始数据获取是预处理的第一步，该步骤主要负责从文件、数据库、网页等众多渠道中获取数据，以得到预处理的初始数据，为后续的处理工作做好数据准备。

针对不同的存储文件，Pandas 读取数据的方式是不同的。Pandas 将数据加载到 DataFrame 后，就可以使用 DataFrame 对象的属性和方法进行操作。这些操作有的是完成数据分析中的常规统计工作，有的是对数据的加工处理。

接下来，本节将针对常用存储格式文件的读写进行介绍。

5.1.1 读取文本(CSV 和 TXT)文件

文本文件是一种由若干行字符构成的计算机文件，它是一种典型的顺序文件。CSV(Comma-Separated Values)是一种逗号分隔的文件格式，因为其分隔符不一定是逗号，又被称为字符分隔文件，文件以纯文本形式存储表格数据(数字和文本)。CSV 不仅可以是一个实体文件，还可以是字符形式(如 URL+data.csv)，以便于在网络上传输。

CSV 文件是一种纯文本文件，可以使用任何文本编辑器进行编辑，它支持追加模式，节省内存开销。因为 CSV 文件具有诸多的优点，所以在很多时候会将数据保存到 CSV 文件中。

CSV 不带数据样式，标准化较强，是最为常见的数据格式。

Pandas 中提供了 read_csv()函数用于读取 CSV 文件，关于它们的具体介绍如下。

1. 通过 read_csv()函数读取 CSV 文件的数据

read_csv()函数的作用是将 CSV 文件的数据读取出来，并转换成 DataFrame 对象。read_csv()函数的语法格式如下。

```
read_csv(filepath_or_buffer,sep=',', delimiter=None, header='infer', names=
None, index_col=None, usecols=None, prefix=None,nrows=None, …)
```

参数说明如下。

- filepath_or_buffer：表示文件路径，可以为 URL 字符串。没有默认值，也不能为空，根据 Python 的语法，第一个参数传参时可以不写参数名。
- sep：接收 string，代表每行数据内容的分隔符。read_csv 默认为“,”，read_table 默认为制表符“[Tab]”，如果分隔符指定错误，在读取数据的时候，每一行数据将连成一片。read_csv()函数还提供了一个参数名为 delimiter 的定界符，这是一个备选分隔符，是 sep 的别名，效果和 sep 一样。如果指定该参数，则 sep 参数失效。常见参数的有以下形式。

```
pd.read_csv(r'./data.csv',sep='\t')          #指定制表符分隔 Tab
pd.read_csv(r'./data.csv',sep='(?<!a)\|(?!1)',engine='python')
                                             #使用正则表达式
```

- header：指定行数作为列名，指定第几行是表头，默认会自动推断把第一行作为表头。常见参数有以下形式。

```
pd.read_csv(r'./data.csv',header=0)          #第一行
pd.read_csv(r'./data.csv',header=None)       #没有表头
pd.read_csv(r'./data.csv',header=[0,1,3])    #多层索引 MulitIndex
```

- names：指定列的名称，是一个类似列表的序列，与数据一一对应。如果文件不包含列名，应该设置 header＝None。列名列表中不允许有重复值。常见参数有以下形式。

```
pd.read_csv(r'./data.csv',names=['列名_1','列名_'])      #指定列名
```

- index_col：指定索引列，可以是行索引的列编号或者列名，接收 int、sequence 或 False，表示索引列的位置。若取值为 sequence 则代表多重索引，默认为 None。如果给定一个序列，则有多个行索引。Pandas 不会自动将第一列作为索引，不指定时会自动使用以 0 开始的自然索引。常见参数有以下形式。

```
pd.read_csv(r'./data.csv',index_col=False)              #不再使用首列作为索引
pd.read_csv(r'./data.csv',index_col=0)                  #第几列作为索引
pd.read_csv(r'./data.csv',index_col='列名')             #指定列名作为索引
pd.read_csv(r'./data.csv',index_col=['列名 1','列名 2']) #多个索引
pd.read_csv(r'./data.csv',index_col=[0,3])              #按列索引指定多个索引
```

- 如果只使用数据的部分列，可以用 usecols 来指定，这样可以加快加载速度并降低内存消耗。常见参数有以下形式。

```
pd.read_csv(r'./data.csv',usecols=[1,5,2])        #按照索引只读取指定列，与顺序无关
pd.read_csv(r'./data.csv',usecols=['列名 1','列名 3','列名 2'])
                                                  #按列名，列名必须存在
```

- nrows：指定需要读取的行数，从文件第一行算起，经常用于较大的数据，先取部分

数据进行代码编写。

需要注意的是，在读取文件时，如果传入的是文件的路径，而不是文件名，则会出现报错，具体的解决方法是先切换到该文件的目录下，使用 OS 模块获取该文件的文件名。

一般情况下，会将读取到的数据返回一个 DataFrame。

【例 5-1】 读取“记账凭证清单.csv”文件。

```
import pandas as pd
pd.set_option('display.unicode.east_asian_width',True)
#读取指定目录下的 csv 格式的文件
df=pd.read_csv(r'./记账凭证清单.csv',encoding='gbk')
print(df.head())                                    #输出前 5 条
```

运行结果：

```
   凭证号    年    月   日  ...   科目名称    明细   科目名称金额     方向金额
0    1    2013   1   5   ...   银行存款    NaN     借方金额   5,151,450.00
1    1    2013   1   5   ...   实收资本    NaN     贷方金额   5,151,450.00
2    2    2013   1   7   ...   销售费用   差旅费    借方金额     4,200.00
3    2    2013   1   7   ...   库存现金    NaN     贷方金额     1,200.00
4    2    2013   1   7   ...  其他应收款   NaN     贷方金额     3,000.00
 [5 rows x 10 columns]
```

在上述代码中指定了编码格式，即 encoding='gbk'。Python 常用的编码格式是 UTF-8 和 GBK 格式，默认编码格式为 UTF-8。读取.csv 文件时，需要通过 encoding 参数指定编码格式。

2. 读取.txt 文件格式

Text 格式的文件也是比较常见的存储数据的方式，扩展名为“.txt”，它与上面提到的 CSV 文件都属于文本文件。如果希望读取 Text 文件，既可以使用前面提到的 read_csv()函数，也可以使用 read_table()函数。读取时需要指定 sep 参数(如制表符\t)。

read_table()函数的语法格式如下。

```
read_table(filepath_or_buffer,sep='\t', delimiter=None, header='infer', names
=None, index_col=None, usecols=None, prefix=None, …)
```

参数说明同 read_csv()函数的参数。

【例 5-2】 读取“记账凭证清单.txt”文件。

```
import pandas as pd
pd.set_option('display.unicode.east_asian_width',True)
#读取指定目录下的 csv 格式的文件
df=pd.read_csv(r'./记账凭证清单.txt',sep='\t',encoding='gbk')
print(df.head())                                    #输出前 5 条
```

运行结果：

```
   凭证号    年    月   日  ...   科目名称    明细   科目名称金额     方向金额
0    1    2013   1   5   ...   银行存款    NaN     借方金额   5,151,450.00
1    1    2013   1   5   ...   实收资本    NaN     贷方金额   5,151,450.00
2    2    2013   1   7   ...   销售费用   差旅费    借方金额     4,200.00
3    2    2013   1   7   ...   库存现金    NaN     贷方金额     1,200.00
4    2    2013   1   7   ...  其他应收款   NaN     贷方金额     3,000.00
 [5 rows x 10 columns]
```

注意：read_csv()函数与 read_table()函数的区别在于使用的分隔符不同，前者使用“,”作为分隔符，而后者使用“\t”作为分隔符。

5.1.2 读取 Excel 文件

Excel 文件也是比较常见的用于存储数据的方式，它里面的数据均是以二维表格的形式显示的。从内容角度来说，Excel 可以分为以文字为主的文字或者信息结构化和以数字为核心的统计报表，因此可以对数据进行统计、分析等操作。

Excel 的文件扩展名有.xls 和.xlsx 两种。Pandas 中提供了对 Excel 文件进行读取操作的方法为 read_excel()函数，关于它们的操作具体如下。

1. 使用 read_excel()函数读取 Excel 文件

read_excel()函数的作用是将 Excel 文件中的数据读取出来，并转换成 DataFrame 对象，其语法格式如下。

```
pandas.read_excel(io,sheet_name=0,header=0,names=None,index_col=None, **kwds)
```

参数说明如下。

- io：接收字符串，表示.xls 或.xlsx 文件路径或类文件对象。
- sheet_name：指定要读取的工作表，可接收 None、字符串、整数、字符串列表或整数列表，默认为 0(表示第一个 Sheet 页中的数据作为 DataFrame 对象)，其他参数值请见表 5-1。字符串指工作表名称；整数类型为索引，表示工作表位置；字符串列表或整数列表用于请求多个工作表；为 None 时则获取所有的工作表。

表 5-1 sheet_name 参数值及说明

值	说 明
sheet_name=0	第一个 Sheet 页中的数据作为 DataFrame 对象
sheet_name=1	第二个 Sheet 页中的数据作为 DataFrame 对象
sheet_name="Sheet1"	名为“Sheet1”的 Sheet 页中的数据作为 DataFrame 对象
sheet_name=[0,1, "Sheet3"]	第一个、第二个和名为“Sheet3”的 Sheet 页中的数据作为 DataFrame 对象

- header：指定作为列名的行，默认为 0，即取第一行的值为列名。数据为除列名以外的数据；若数据不包括列名，则设置为 header=None。
- names：默认为 None，表示要使用的列名列表。如不指定，默认为表头的名称。
- index_col：指定列为索引列，默认为 None，索引 0 是 DataFrame 对象的行标签。

特别提醒，当使用 read_excel()函数读取 Excel 文件时，若出现 importError 异常，说明当前 Python 环境中缺少读取 Excel 文件的依赖库 xlrd，需要手动安装依赖库 xlrd(pip install xlrd)进行解决。

2. 读取指定 Sheet 页的数据

一个 Excel 文件有时会包含多个 Sheet 页，通过设置 sheet_name 参数就可以读取指定 Sheet 页的数据。

【例 5-3】 通过 read_excel()函数读取“高中班学生成绩.xlsx”文件中“高一(1)班成绩”

Sheet 的数据。

```
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)
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩')
print(df.head())                                    #输出前 5 条
```

运行结果：

```
     学号     姓名    语文   数学   英语   物理   化学   地理   历史
0  G210101   申志凡   99.0    98  101.0     95     91     95     78
1  G210102   冯默风   78.0    95   94.0     82     90     93     94
2  G210103   石双英   84.0   100   97.0     87     78     89     93
3  G210104   史伯威  101.0   110  102.0     93     95     92     88
4  G210105   王家骏   91.5    89   94.0     92     91     86     86
```

除了指定 Sheet 页的名字，还可以指定 Sheet 页的顺序，从 0 开始。例如，“sheet_name＝0”表示读取第一个 Sheet 页的数据；“sheet_name＝1”表示导入第二个 Sheet 页的数据，以此类推。

如果不指定 sheet_name 参数，则默认导入第一个 Sheet 页的数据。

特别提醒，在读取文件时，要注意绝对路径和相对路径的问题。在 Python 中则需要在路径前面加一个“r”，以避免路径里的反斜杠“\”被转义。

3. 通过行列索引读取指定行列数据

DataFrame 是二维数据结构，因此它既有行索引又有列索引。当读取 Excel 文件时，行索引会自动生成，如 0，1，2，…，而列索引则默认将第 0 行作为列索引。

如果通过指定行索引读取 Excel 文件，则需要设置 index_col 参数。

一个 Excel 表有多列数据，若只需其中几列数据，可以通过 usecols 参数指定需要的列，一般从 0 开始（表示第 1 列，以此类推）。如果导入多列，则可以在列表中指定多个值，也可以指定列名称（要么都为值，要么都为列名称）。

【例 5-4】 以第 0 列作为行索引，选取姓名、数学和物理列数据。

```
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)
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',index_col=
0,usecols=[0,1,3,5])                                #设置"学号"为行索引
print(df.head())                                    #输出前 5 条
```

运行结果：

```
     学号     姓名   数学   物理
G210101   申志凡    98     95
G210102   冯默风    95     82
G210103   石双英   100     87
G210104   史伯威   110     93
G210105   王家骏    89     92
```

如果通过指定列索引导入 Excel 数据，则需要设置 header 参数，关键代码如下。

```
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',header=1)
#设置第 1 行为列索引
```

如果将数字作为列索引,可以设置 header 参数为 None,关键代码如下。

```
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',header=
None) #列索引为数字
```

那么,为什么要指定索引呢? 因为通过索引可以快速地检索数据,例如,根据 df[1],就可以快速检索到“姓名”这一列数据。

4. 读取指定列数据

一个 Excel 表中往往包含多列数据,如果只需要其中的几列,可以通过 usecols 参数指定需要的列,从 0 开始(表示第 1 列,以此类推)。

【例 5-5】 读取“高中班学生成绩.xlsx”第 1 列数据的前 5 条。

```
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)
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',usecols=[1])
                                                    #读取第 1 列
print(df.head())                                    #输出前 5 条
```

运行结果:

```
    姓名
0  申志凡
1  冯默风
2  石双英
3  史伯威
4  王家骏
```

如果导入多列,则可以在列表中指定多个值。例如,导入第 1 列和第 4 列,关键代码如下。

```
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',usecols=
[0,3])
```

也可以指定列名称,关键代码如下。

```
df=pd.read_excel(r'./高中班学生成绩.xlsx',sheet_name='高一(1)班成绩',usecols=
['姓名'])
```

5.1.3 读取 JSON 数据文件

JSON(JavaScript Object Notation)是互联网上非常通用的轻量级数据交换格式,是 HTTP 请求中数据的标准格式之一。Pandas 提供的 JSON 读取方法在解析网络爬虫数据时,可以极大地提高效率,广泛应用于 Web 数据交互。

JSON 采用独立于编程语言的文本格式来存储数据,其文件的扩展名为.json,可通过文本编辑工具查看。

JSON 格式简洁、结构清晰,使用键值对(key:value)的格式存储数据对象。

key 是数据对象的属性,value 是数据对象属性的对应值。JSON 数据使用大括号来区分表示并存储。例如,“性别”:“男”就是一个 key:value 结构的数据。

例如:

```
{
" fruit ":"apple" ,
"color": red ,
"productioninformation":"AnHui",
"farmer ":{"name":"Muzi" , "age":40 , "sex":"male"}
}
```

对象在 JSON 中是使用大括号{}括起来的内容，数据结构为{key1：value1，key2：value2，…}的键值对结构。

数组（理解为 Python 中的列表，形式一致）在 JSON 中是使用方括号[]括起来的内容，可以存放多个对象。

JSON 文件格式广泛应用于互联网服务器 API 中，易于机器解析和生成，文件体积小。但是 JSON 文件格式存储单一，只能存储文本，不如 Excel 容易阅读。

JSON 文件格式与 Excel 文件一样，编码格式为 ANSI、Unicode 和 UTF-8。在数据加载时，可以根据需要进行相应的处理，其中，UTF-8 格式的 JSON 文件应用最为广泛。

Pandas 读取 JSON 数据的 read_json()函数如下：

```
pandas.read_json(path_or_buf=None, orient=None, typ='frame', dtype=True,
convert_axes=True, convert_dates=True, keep_default_dates=True,
numpy=False, precise_float=False, date_unit=None, encoding=None,
lines=False, chunksize=None, compression='infer')
```

【示例 5-6】 Pandas 通过 read_json()函数读取 JSON 数据。

```
import pandas as pd
df = pd.read_json(r"./json_data.json",encoding="utf8")
print(df)
```

运行结果：

```
        sno   sname ssex  sage
0  202101002   Marry    F    18
1  202102001  Strong    M    19
```

Pandas 还提供了 pd.json normalize(data)方法来读取半结构化的 JSON 数据。

5.1.4　读取 HTML 表格数据

在浏览网页时，有些数据会在 HTML（Hyper Text Markup Language，超文本标记语言）网页中以表格的形式进行展示，对于这部分数据，可以使用 Pandas 中的 read_html()函数接收 HTML 字符串、HTML 文件、URL，并将 HTML 中的＜table＞标签表格数据解析为 DataFrame。如返回有多个 df 的列表，则可以通过索引指定取第几个。如果页面里只有一个表格，那么这个列表就只有一个 DataFrame。

read_html()函数的语法格式如下。

```
pandas.read_html(io, match='.+', flavor=None,
header=None, index_col=None,skiprows=None, encoding=None,attrs=None)
```

参数说明如下。

- io：字符串，文件路径，也可以是 URL 链接。如果网址不接受 https，可以尝试去掉 https 中的 s 后爬取。

- match：正则表达式，返回与正则表达式匹配的表格。
- flavor：解析器默认为“lxml”。
- header：指定列标题所在的行，列表 list 为多重索引。
- index_col：指定行标题对应的列，列表 list 为多重索引。
- encoding：字符串，默认为 None，文件的编码格式。
- attrs：默认为 None，用于表示表格的属性值。

在使用 read_html()函数时，首先要确定网页表格是否为＜table＞标签。可以通过在网页中单击右键，在弹出的菜单中选择“查看源文件”，查看代码是否含有表格标签“＜table＞…＜/table＞”的字样，然后才使用 read_html()函数。

【例 5-7】 读取新浪网上的大学部分专业信息。

```
import pandas as pd
import requests
html_data = requests.get('http://kaoshi.edu.sina.com.cn/college/majorlist?page=1')
html_table_data = pd.read_html(html_data.content,header=0,encoding='utf-8')
columns = ['专业名称','专业代码','专业大类','专业小类']
df = pd.DataFrame(data=html_table_data[1],columns=columns)
print(df.head())
```

运行结果：

```
   专业名称  专业代码  专业大类  专业小类
0  哲学类     101      哲学     哲学类
1   哲学     10101     哲学     哲学类
2  逻辑学    10102     哲学     哲学类
3  宗教学    10103     哲学     哲学类
4  伦理学    10104     哲学     哲学类
```

值得一提的是，在使用 read_html()函数读取网页中的表格数据时，需要注意网页的编码格式。运行程序，如果出现“ImportError：lxml not found，please install it”的错误提示信息，则需要安装 lxml 模块。

5.1.5 读取 MySQL 数据库中数据

大多数情况下，海量的数据是使用数据库进行存储的，这主要是依赖于数据库的数据结构化、数据共享性、独立性等特点。因此，在实际生产环境中，绝大多数的数据都是存储在数据库中。

Pandas 支持 MySQL、Oracle、SQLite 等主流数据库的读写操作。

为了高效地对数据库中的数据进行读取，这里需要引入 SQLAlchemy。SQLAlchemy 是使用 Python 编写的一款开源软件，它提供的 SQL 工具包和对象映射工具能够高效地访问数据库。在使用 SQLAlchemy 时需要使用相应的连接工具包，如 MySQL 需要安装 mysqlconnector，Oracle 则需要安装 cx_oracle。

Pandas 的 io.sql 模块中提供了常用的读写数据库函数，read_sql_table()函数与 read_sql_query()函数都可以将读取的数据转换为 DataFrame 对象，前者表示将整张表的数据转换成 DataFrame，后者则表示将执行 SQL 语句的结果转换为 DataFrame 对象。而 read_sql()

函数同时支持 read_sql_table()函数与 read_sql_query()函数两者的功能。to_sql()方法则是把记录数据写到数据库里。

在连接 MySQL 数据库时，这里使用的是 mysqlconnector 驱动，如果当前的 Python 环境中没有该模块，则需要使用 pip install mysql-connector 命令安装该模块。下面以 read_sql()函数和 to_sql()方法为例，分别介绍如何读写数据库中的数据，具体内容如下。

1. 使用 read_sql()函数读取数据

read_sql()函数既可以读取整张数据表，又可以执行 SQL 语句，其语法格式如下。

```
pandas.read_sql(sql,con,index_col=None,coerce_float=True,params=None,parse_
dates=None, columns=None, chunksize=None)
```

参数说明如下。

- sql：表示被执行的 SQL 语句。
- con：接收数据库连接，表示数据库的连接信息。
- index_col：默认为 None，如果传入一个列表，则表示为层次化索引。
- coerce_float：将非字符串、非数字对象的值转换为浮点数类型。
- params：传递给执行方法的参数列表，如 params ={'name':'value'}。
- columns：接收 list 表示读取数据的列名，默认为 None。

如果发现数据中存在空值，则会使用 NaN 进行补全。

【例 5-8】 使用 read_sql()函数读取数据库中的数据表 specialty。

```
import pandas as pd
from sqlalchemy import create_engine
#mysql 账号为 root,密码为 123456,数据名为 jxgl
#数据表名称:specialty
engine = create_engine('mysql+pymysql://'
                'root:123456@127.0.0.1:3306/jxgl')
#通过数据表名读取数据库的数据
#category_data = pd.read_sql('specialty', engine)
#也可以通过 SQL 语句读取数据库的数据
sql = 'SELECT * FROM specialty'
df_data = pd.read_sql(sql, engine)
print(df_data)
```

运行结果：

```
    zno        zname
0  1102  数据科学与大数据技术
1  1103       人工智能
2  1201      网络与新媒体
3  1214    区块链科学与工程
4  1407    健康服务与管理
5  1409      智能医学工程
6  1601      供应链管理
7  1805      智能感知工程
8  1807     智能装备与系统
```

上述示例中，首先导入了 sqlalchemy 模块，通过 create_engine()函数创建连接数据库的信息，然后调用 read_sql()函数读取数据库中的 specialty 数据表，并转换成 DataFrame 对象。

注意：在使用 create_engine()函数创建连接时，其格式如下：'数据库类型＋数据库驱动名称://用户名：密码@机器地址：端口号/数据库名'。

需要强调的是，这里的 SQL 语句不仅是用于筛选的 SQL 语句，其他用于增删改查的 SQL 语句都是可以执行的。

2. 使用 to_sql()方法将数据写入数据库中

to_sql()方法的功能是将 Series 或 DataFrame 对象以数据表的形式写入数据库中，其语法格式如下。

```
to_sql(name, con, schema = None, if_exists = 'fail', index = True, index_label =
None, chunksize = None, dtype = None)
```

参数说明如下。

- name：表示数据库表的名称。
- con：表示数据库的连接信息。
- if_exists：可以取值为 fail、replace 或 ap pend，默认为 fail。每个取值代表的含义如下。

  fail：如果表存在，则不执行写入操作。

  replace：如果表存在，则将源数据库表删除再重新创建。

  append：如果表存在，那么在原数据库表的基础上追加数据。
- index：表示是否将 DataFrame 行索引作为数据传入数据库，默认为 True。
- index_label：表示是否引用索引名称。如果 index 设为 True，此参数为 None，则使用默认名称；如果 index 为层次化索引，则必须使用序列类型。

接下来，通过一个示例程序来演示如何使用 Pandas 向数据库中写入数据。

首先，创建一个名称为 students_info 的数据库，具体的 SQL 语句如下。

```
CREATE DATABASE students_info CHARASET=utf8
```

然后，创建一个 DataFrame 对象，它统计了每个年级中男生和女生的人数。

接着，调用 to_sql()方法将 DataFrame 对象写入名称为 students 的数据表中，具体代码如下。

```
from pandas import DataFrame, Series
import pandas as pd
from sqlalchemy import create_engine
from sqlalchemy.types import *
df = DataFrame({"班级":["一年级","二年级","三年级","四年级"],
                              "男生人数":[25,23,27,30],
                              "女生人数":[19,17,20,20]})
#创建数据库引擎
#mysql+pymysql 表示使用 MySQL 数据库的 pymysql 驱动
#账号:root,密码:123456,数据库名:studnets_info
#数据表的名称: students
engine=create_engine('mysql+mysqlconnector://root:123456@127.0.0.1/students_
info')
df.to_sql('students',engine)
```

当程序执行结束后，可以在数据库中查看是否成功创建了数据表，以及数据是否保存成功，这里使用命令行的方式进行验证。