# 第二十五章 Python 进行数据分析

## 25.1 导入外部数据

导入数据主要用到的是 Pandas 里的 read_x()方法，x 表示待导入文件的格式。

### 25.1.1 导入.xlsx 文件

Python 中导入.xlsx 文件的方法是 read_excel()。

**1) 基本导入**

使用 read_excel()方法导入文件，首先要指定文件的路径，也就是这个文件在电脑中的哪个文件夹下存着。

**【示例 25-1】导入.xlsx 文件**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df
```

执行结果如图 25-1 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

**图 25-1 示例 25-1 运行效果图**

电脑中的文件默认路径使用\，这个时候需要在路径前面加一个 r（转义符）避免路径里面的\被转义。也可以不加 r，但是需要把路径里面的所有\转换成/，这个规则在导入其他格式文件时也是一样的，一般在路径前面加 r。

**2) 指定导入哪个 Sheet**

.xlsx 格式的文件有多个 Sheet，可以通过设定 sheet_name 参数来指定要导入哪个 Sheet 的文件。

**【示例 25-2】导入.xlsx 文件时，指定导入哪个 Sheet**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name='Sheet1')
df
```

执行结果如图 25-2 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

**图 25-2 示例 25-2 运行效果图**

除了可以指定具体 Sheet 的名字，还可以传入 Sheet 的顺序，从 0 开始计数。

**【示例 25-3】导入.xlsx 文件时，传入 Sheet 的顺序**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name=0)
df
```

执行结果如图 25-3 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

**图 25-3 示例 25-3 运行效果图**

如果不指定 sheet_name 参数时，那么默认导入的都是第一个 Sheet 的文件。

**3) 指定行索引**

将本地文件导入 DataFrame 时，行索引使用的从 0 开始的默认索引，可以通过设置 index_col 参数来设置。

**【示例 25-4】导入.xlsx 文件时，通过 index_col 指定行索引**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",sheet_name=0,index_col=0)
df
```

执行结果如图 25-4 所示：

| 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|
| 张三 | 30 | CEO | 50000 | 2010-03-11 |
| 李四 | 24 | 主管 | 30000 | 2018-08-19 |
| 王五 | 25 | 经理 | 40000 | 2015-10-11 |

**图 25-4 示例 25-4 运行效果图**

index_col 表示用.xlsx 文件中的第几列做行索引，从 0 开始计数。

**4) 指定列索引**

将本地文件导入 DataFrame 时，默认使用源数据表的第一行作为列索引，也可以通过设置 header 参数来设置列索引。header 参数值默认为 0，即用第一行作为列索引；也可以是其他行，只需要传入具体的那一行即可；

**【示例 25-5】导入.xlsx 文件时，通过 header 指定行索引**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",header=1)
df
```

执行结果如图 25-5 所示：

| | 张三 | 30 | CEO | 50000 | 2010-03-11 00:00:00 |
|---|---|---|---|---|---|
| 0 | 李四 | 24 | 主管 | 30000 | 2018-08-19 |
| 1 | 王五 | 25 | 经理 | 40000 | 2015-10-11 |

**图 25-5 示例 25-5 运行效果图**

**【示例 25-6】导入.xlsx 文件时，通过 header 指定行索引**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",header=None)
df
```

执行结果如图 25-6 所示：

| | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
| 1 | 张三 | 30 | CEO | 50000 | 2010-03-11 00:00:00 |
| 2 | 李四 | 24 | 主管 | 30000 | 2018-08-19 00:00:00 |
| 3 | 王五 | 25 | 经理 | 40000 | 2015-10-11 00:00:00 |

**图 25-6 示例 25-6 运行效果图**

**5) 指定导入列**

有时候本地文件的列数太多，而我们又不需要那么多列时，我们就可以通过设置 usecols 参数来指定要导入的列。

**【示例 25-7】导入.xlsx 文件时，通过 usecols 指定列索引**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx",usecols=1)
df
```

执行结果如图 25-7 所示：

| | 姓名 | 年龄 |
|---|---|---|
| 0 | 张三 | 30 |
| 1 | 李四 | 24 |
| 2 | 王五 | 25 |

**图 25-7 示例 25-7 运行效果图**

## 25.1.2 导入.csv 文件

**1) 直接导入，指定编码格式**

导入时除了指明文件路径，还需要设置编码格式。Python 中用得比较多的两种编码格式是 UTF-8 和 gbk，默认编码格式是 UTF-8。我们要根据导入文件本身的编码格式进行设置，通过设置参数 encoding 来设置导入的编码格式。

**【示例 25-8】导入.csv 文件，文件编码格式是 gbk**

```
import pandas as pd
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk')
df
```

执行结果如图 25-8 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

**图 25-8 示例 25-8 运行效果图**

**2) 指明分隔符**

在 Excel 和 DataFrame 中的数据都是很规整的排列的，这都是工具在后台根据某条规则进行切分的。read_csv()默认文件中的数据都是以逗号分开的，但是有的文件不是用逗号分开的，这时候就需要人为指定分隔符号，否则就会报错。

**【示例 25-9】导入.csv 文件，指明分隔符**

```
import pandas as pd
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=' ')
df
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=', ')
df
```

执行结果如图 25-9 所示：

```
import pandas as pd
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=' ')
df
```

| | 姓名,年龄,工作,薪水,入职日期 |
|---|---|
| 0 | 张三,30,CEO,50000,2010/3/11 |
| 1 | 李四,24,主管,30000,2018/8/19 |
| 2 | 王五,25,经理,40000,2015/10/11 |

```
df = pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=',')
df
```

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

**图 25-9 示例 25-9 运行效果图**

从上面的示例可以看到，如果使用空格进行分隔，获取到的数据还是一个整体，并没有分开。用默认的逗号作为分隔符号进行分隔，数据被规整地分好了。常用的分隔符除了逗号、空格，还有制表符（\t）。

**3) 指明读取行数**

如果有一个几百兆的文件，想了解一席按这个文件里有哪些数据，那么这时候就没有必要把全部数据都导入，只要看到前面几行即可，可以通过设置 nrows 参数。

**【示例 25-10】导入.csv 文件，nrows 参数的使用**

```
import pandas as pd
pd.read_csv(r"E:\excelTest.csv",encoding='gbk',sep=',',nrows=1)
```

执行结果如图 25-10 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |

图 25-10 示例 25-10 运行效果图

**4) engine 指定**

当文件路径或者文件名中包含中文时，如果还用上面的导入方式就会报错。因为当调用 read_csv()方法时，默认使用 C 语言作为解析语言，可以通过设置 engine 参数，把默认值 C 改为 Python 就可以了，如果文件格式是 CSV UTF-8（逗号分隔）(*.csv)，那么编码格式也需要跟着变为 utf-8-sig，如果文件格式是 CSV（逗号分隔）(*.csv）格式，对应的编码格式则为 gbk。

**【示例 25-11】导入.csv 文件，engine 参数的使用**

```
import pandas as pd
pd.read_csv(r"E:\文件\excelTest.csv",engine='python',encoding='gbk')
```

执行结果如图 25-11 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

图 25-11 示例 25-11 运行效果图

.csv 文件也涉及行、列索引设置及指定导入某列或某几列，设定方法与导入.xlsx 文件一致。

## 25.1.3 导入.txt 文件

导入.txt 文件用得方法时 read_table()，read_table()是将利用分隔符分开的文件导入 DataFrame 的通用函数。它不仅仅可以导入.txt 文件，还可以导入.csv 文件。

**【示例 25-12】导入.txt 文件**

```
import pandas as pd
pd.read_table(r"E:\testFile.txt",encoding='gbk',sep='\t')
```

执行结果如图 25-12 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

图 25-12 示例 25-12 运行效果图

【示例 25-13】使用 read_table()方法导入.csv 文件

```
import pandas as pd
pd.read_table(r"E:\excelTest.csv",encoding='gbk',sep=',')
```

执行结果如图 25-13 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | 张三 | 30 | CEO | 50000 | 2010/3/11 |
| 1 | 李四 | 24 | 主管 | 30000 | 2018/8/19 |
| 2 | 王五 | 25 | 经理 | 40000 | 2015/10/11 |

图 25-13 示例 25-13 运行效果图

## 25.2 查看数据

有了数据以后，先查看数据，只有对数据充分熟悉后，才能更好地进行分析。

### 25.2.1 预览前、后几行

当数据表中包含数据行数过多时，我们只想看一下每一列数据都是什么样，可以只把数据的前几行或者后几行进行查看。

导入一个文件后，可以用 head()方法来控制要显示的前几行，tail()方法来控制要显示的后几行。在 head()、tail()方法中直接设置行数，默认展示 5 行。

【示例 25-14】预览前、后几行

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
display(df,df.head(),df.tail(2))
```

执行结果如图 25-14 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | A1 | 30 | CEO | 50000 | 2010-03-11 |
| 1 | B1 | 24 | 主管 | 30000 | 2018-08-19 |
| 2 | C1 | 25 | 经理 | 40000 | 2015-10-11 |
| 3 | A2 | 23 | 员工 | 2000 | 2017-04-23 |
| 4 | B2 | 28 | 主任 | 3200 | 2013-11-13 |
| 5 | C2 | 25 | 员工 | 2200 | 2016-09-12 |
| 6 | D2 | 26 | 员工 | 2400 | 2018-10-12 |

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | A1 | 30 | CEO | 50000 | 2010-03-11 |
| 1 | B1 | 24 | 主管 | 30000 | 2018-08-19 |
| 2 | C1 | 25 | 经理 | 40000 | 2015-10-11 |
| 3 | A2 | 23 | 员工 | 2000 | 2017-04-23 |
| 4 | B2 | 28 | 主任 | 3200 | 2013-11-13 |

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 5 | C2 | 25 | 员工 | 2200 | 2016-09-12 |
| 6 | D2 | 26 | 员工 | 2400 | 2018-10-12 |

**图 25-14 示例 25-14 运行效果图**

## 25.2.2 调用 shape 获取数据表的大小

调用 shape 获取数据表的行、列数。

**【示例 25-15】调用 shape 获取数据表的行、列数**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
display(df,df.shape)
```

执行结果如图 25-15 所示：

| | 姓名 | 年龄 | 工作 | 薪水 | 入职日期 |
|---|---|---|---|---|---|
| 0 | A1 | 30 | CEO | 50000 | 2010-03-11 |
| 1 | B1 | 24 | 主管 | 30000 | 2018-08-19 |
| 2 | C1 | 25 | 经理 | 40000 | 2015-10-11 |
| 3 | A2 | 23 | 员工 | 2000 | 2017-04-23 |
| 4 | B2 | 28 | 主任 | 3200 | 2013-11-13 |
| 5 | C2 | 25 | 员工 | 2200 | 2016-09-12 |
| 6 | D2 | 26 | 员工 | 2400 | 2018-10-12 |

(7, 5)

**图 25-15 示例 25-15 运行效果图**

shape 方法会以元组的形式返回行、列数，上面代码中的（7,5）表示 df 表有 7 行 5 列数据。这里需要注意的是，Python 中利用 shape 方法获取行数和列数时不会把行索引和列索引计算在内，而 Excel 中是把行索引和列索引计算在内的。

## 25.2.3 调用 info()获取数据类型

查看数据的第二点就是看一下数据类型，不同的数据类型的分析思路是不一样的，比如数值类型的数据可以求均值，但是字符串类型的数据就没法求均值了。

利用 info()查看数据表中的数据类型，而且不需要一列一列查看，在调用 info()方法以后就会输出整个表中所有列的数据类型。

**【示例 25-16】调用 info()获取数据类型**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df.info()
```

执行结果如图 25-16 所示：

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
姓名      7 non-null object
年龄      7 non-null int64
工作      7 non-null object
薪水      7 non-null int64
入职日期    7 non-null datetime64[ns]
dtypes: datetime64[ns](1), int64(2), object(2)
memory usage: 360.0+ bytes
```

**图 25-16 示例 25-16 运行效果图**

通过 info()方法可以看出表 df 的行索引 index 是 0-6，共 5columns，分别是姓名、年龄、工作、薪水及入职时间，且 5columns 中年龄和薪水是 int 类型，入职日期是 datetime 类型，其他 columns 都是 object 类型，共占用内存 360bytes。

## 25.2.4 调用 describe()获取数值分布情况

查看数据的第三点就是掌握数值的分布情况，即均值是多少，最大最小是多少，方差即分位数分别又是多少。

调用 describe()方法就可以获取所有数值类型字段的分布值。

**【示例 25-17】调用 describe()获取数值分布情况**

```
import pandas as pd
df = pd.read_excel(r"E:\excelTest.xlsx")
df.describe()
```

执行结果如图 25-17 所示：

| | 年龄 | 薪水 |
|---|---|---|
| count | 7.000000 | 7.000000 |
| mean | 25.857143 | 18542.857143 |
| std | 2.410295 | 20888.502192 |
| min | 23.000000 | 2000.000000 |
| 25% | 24.500000 | 2300.000000 |
| 50% | 25.000000 | 3200.000000 |
| 75% | 27.000000 | 35000.000000 |
| max | 30.000000 | 50000.000000 |

**图 25-17 示例 25-17 运行效果图**

# 25.3 足球运动员数据分析

## 25.3.1 加载数据

**【示例 25-18】加载足球运动员数据**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
#支持中文
#解决中文显示问题
plt.rcParams['font.sans-serif'] = ['KaiTi'] # 指定默认字体
plt.rcParams['axes.unicode_minus'] = False # 解决保存图像是负号'-'显示为方块的问题
player = pd.read_csv('./data/FullData.csv')
```

## 25.3.2 查看数据

**【示例 25-19】查看数据前 5 行**

```
player.head()
```

执行结果如图 25-18 所示：

| | Name | Nationality | National_Position | National_Kit | Club | Club_Position | Club_Kit | Club_Joining | Contract_Expiry | Rating | ... | Long_Shots |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Cristiano Ronaldo | Portugal | LS | 7.0 | Real Madrid | LW | 7.0 | 07/01/2009 | 2021.0 | 94 | ... | 90 |
| 1 | Lionel Messi | Argentina | RW | 10.0 | FC Barcelona | RW | 10.0 | 07/01/2004 | 2018.0 | 93 | ... | 88 |
| 2 | Neymar | Brazil | LW | 10.0 | FC Barcelona | LW | 11.0 | 07/01/2013 | 2021.0 | 92 | ... | 77 |
| 3 | Luis Suárez | Uruguay | LS | 9.0 | FC Barcelona | ST | 9.0 | 07/11/2014 | 2021.0 | 92 | ... | 86 |
| 4 | Manuel Neuer | Germany | GK | 1.0 | FC Bayern | GK | 1.0 | 07/01/2011 | 2021.0 | 92 | ... | 16 |

5 rows × 53 columns

**图 25-18 示例 25-19 运行效果图**

**【示例 25-20】查看数据的数据类型**

```
player.info()
```

执行结果如下所示：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17588 entries, 0 to 17587
Data columns (total 53 columns):
Name                  17588 non-null object
Nationality           17588 non-null object
National_Position     1075 non-null object
National_Kit          1075 non-null float64
Club                  17588 non-null object
Club_Position         17587 non-null object
Club_Kit              17587 non-null float64
Club_Joining          17587 non-null object
Contract_Expiry       17587 non-null float64
Rating                17588 non-null int64
Height                17588 non-null object
Weight                17588 non-null object
Preffered_Foot        17588 non-null object
Birth_Date            17588 non-null object
Age                   17588 non-null int64
Preffered_Position    17588 non-null object
Work_Rate             17588 non-null object
Weak_foot             17588 non-null int64
Skill_Moves           17588 non-null int64
Ball_Control          17588 non-null int64
```

```
Dribbling                17588 non-null int64
Marking                  17588 non-null int64
Sliding_Tackle           17588 non-null int64
Standing_Tackle          17588 non-null int64
Aggression               17588 non-null int64
Reactions                17588 non-null int64
Attacking_Position       17588 non-null int64
Interceptions            17588 non-null int64
Vision                   17588 non-null int64
Composure                17588 non-null int64
Crossing                 17588 non-null int64
Short_Pass               17588 non-null int64
Long_Pass                17588 non-null int64
Acceleration             17588 non-null int64
Speed                    17588 non-null int64
Stamina                  17588 non-null int64
Strength                 17588 non-null int64
Balance                  17588 non-null int64
Agility                  17588 non-null int64
Jumping                  17588 non-null int64
Heading                  17588 non-null int64
Shot_Power               17588 non-null int64
Finishing                17588 non-null int64
Long_Shots               17588 non-null int64
Curve                    17588 non-null int64
Freekick_Accuracy        17588 non-null int64
Penalties                17588 non-null int64
Volleys                  17588 non-null int64
GK_Positioning           17588 non-null int64
GK_Diving                17588 non-null int64
GK_Kicking               17588 non-null int64
GK_Handling              17588 non-null int64
GK_Reflexes              17588 non-null int64
dtypes: float64(3), int64(38), object(12)
```

```
memory usage: 7.1+ MB
```

### 25.3.3 缺值处理

从上述示例可以看到总共 17588 行，但 National_Position（国家队位置） 是 1075 行，Club_Position （俱乐部位置）17587 行。我们知道有的足球运动员是没有进入国家队的，所以 National_Position 缺值是正常情况。但 Club_Position 缺值需要处理。

**【示例 25-21】删除 Club_Position 的缺失值**

```
player = player[player['Club_Position'].notnull()]
player.info()
```

执行结果如下所示：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 17587 entries, 0 to 17587
Data columns (total 53 columns):
Name                    17587 non-null object
Nationality             17587 non-null object
National_Position       1075 non-null object
National_Kit            1075 non-null float64
Club                    17587 non-null object
Club_Position           17587 non-null object
Club_Kit                17587 non-null float64
Club_Joining            17587 non-null object
Contract_Expiry         17587 non-null float64
Rating                  17587 non-null int64
Height                  17587 non-null object
Weight                  17587 non-null object
Preffered_Foot          17587 non-null object
Birth_Date              17587 non-null object
Age                     17587 non-null int64
Preffered_Position      17587 non-null object
Work_Rate               17587 non-null object
Weak_foot               17587 non-null int64
Skill_Moves             17587 non-null int64
Ball_Control            17587 non-null int64
```

```
Dribbling                 17587 non-null int64
Marking                   17587 non-null int64
Sliding_Tackle            17587 non-null int64
Standing_Tackle           17587 non-null int64
Aggression                17587 non-null int64
Reactions                 17587 non-null int64
Attacking_Position        17587 non-null int64
Interceptions             17587 non-null int64
Vision                    17587 non-null int64
Composure                 17587 non-null int64
Crossing                  17587 non-null int64
Short_Pass                17587 non-null int64
Long_Pass                 17587 non-null int64
Acceleration              17587 non-null int64
Speed                     17587 non-null int64
Stamina                   17587 non-null int64
Strength                  17587 non-null int64
Balance                   17587 non-null int64
Agility                   17587 non-null int64
Jumping                   17587 non-null int64
Heading                   17587 non-null int64
Shot_Power                17587 non-null int64
Finishing                 17587 non-null int64
Long_Shots                17587 non-null int64
Curve                     17587 non-null int64
Freekick_Accuracy         17587 non-null int64
Penalties                 17587 non-null int64
Volleys                   17587 non-null int64
GK_Positioning            17587 non-null int64
GK_Diving                 17587 non-null int64
GK_Kicking                17587 non-null int64
GK_Handling               17587 non-null int64
GK_Reflexes               17587 non-null int64
dtypes: float64(3), int64(38), object(12)
```

```
memory usage: 7.2+ MB
```

**【示例 25-22】查看是否有重复数据**

```
player.duplicated().any()
```

执行结果如图 25-19 所示：

```
#查看是否有重复数据
player.duplicated().any()
```

```
False
```

**图 25-19 示例 25-22 运行效果图**

## 25.3.4 运动员身高和体重分布

从查看数据结果可以看到运动员身高 Height、体重 Weight 的数据后都添加了相应的单位。要分析运动员身高和体重的分布，首先需要将身高 Height 和 Weight 数据的单位去掉。

**【示例 25-23】去掉 Height 和 Weight 数据的单位**

```
player['Height'] = player['Height'].str.replace('cm','')
player['Weight'] = player['Weight'].str.replace('kg','')
player['Height'] = player['Height'].astype('int')
player['Weight'] = player['Weight'].astype('int')
display(player['Height'].head(),player['Weight'].head())
```

执行结果如图 25-20 所示：

```
display(player['Height'].head(),player['Weight'].head())
```

```
0    185
1    170
2    174
3    182
4    193
Name: Height, dtype: int32

0    80
1    72
2    68
3    85
4    92
Name: Weight, dtype: int32
```

图 25-20 示例 25-23 运行效果图

【示例 25-24】运动员 Height 数值分布

```
player['Height'].describe()
```

执行结果如图 25-21 所示，运动员 Height 数值分布的直方图 25-22 所示：

```
player['Height'].describe()
```

```
count    17587.000000
mean       181.105021
std          6.675084
min        155.000000
25%        176.000000
50%        181.000000
75%        186.000000
max        207.000000
Name: Height, dtype: float64
```

图 25-21 示例 25-24 运行效果图

```
player['Height'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bb9376c50>
```

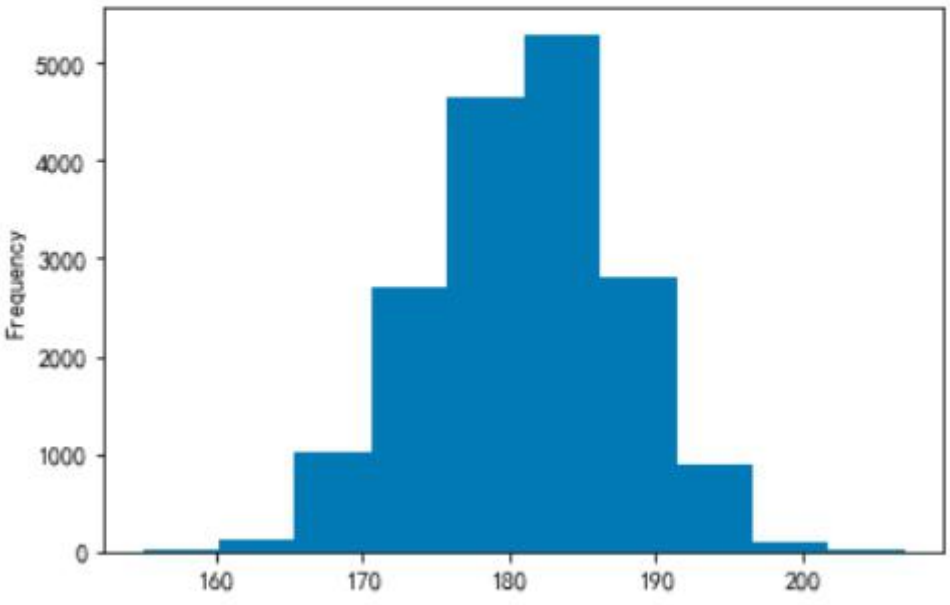


图 25-22 运动员 Height 直方图

【示例 25-25】运动员 Weight 数值分布

```
player['Weight'].describe()
```

执行结果如图 25-23 所示，运动员 Weight 数值分布的直方图 25-24 所示：

```
player['Weight'].describe()
```

```
count    17587.000000
mean        75.253085
std          6.898051
min         48.000000
25%         70.000000
50%         75.000000
75%         80.000000
max        110.000000
Name: Weight, dtype: float64
```

图 25-23 示例 25-25 运行效果图

```
player['Weight'].plot(kind='hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1c05793dfd0>
```

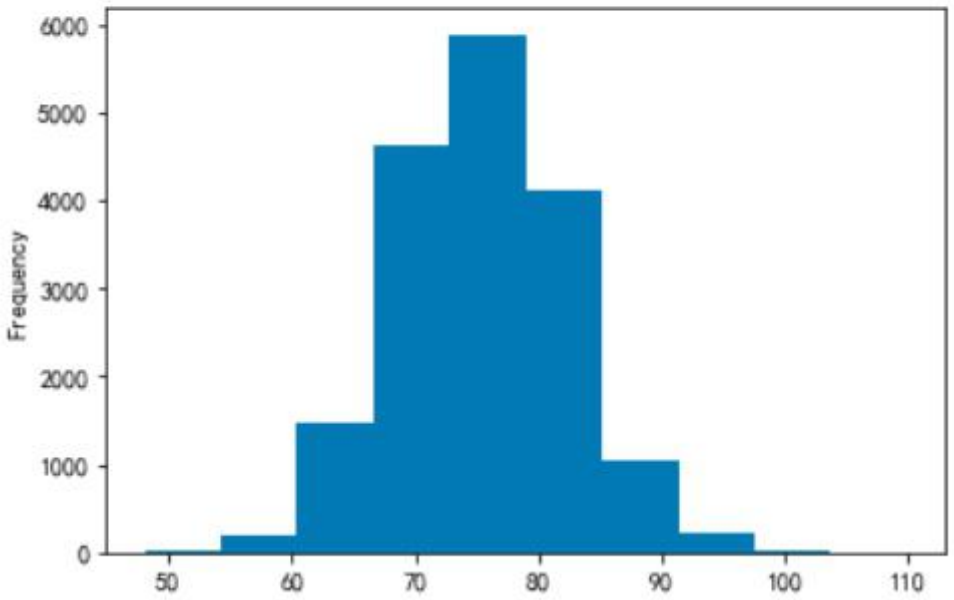


图 25-24 运动员 Weight 直方图

## 25.3.5 左脚右脚使用数量

**【示例 25-26】查看足球运动员左脚右脚使用情况**

```
player['Preffered_Foot'].head(10)
```

执行结果如图 25-25 所示：

```
player['Preffered_Foot'].head(10)
0    Right
1     Left
2    Right
3    Right
4    Right
5    Right
6    Right
7     Left
8    Right
9     Left
Name: Preffered_Foot, dtype: object
```

**图 25-25 示例 25-26 运行效果图**

从上述示例可以看到，足球运动员踢球有使用左脚也有使用右脚。要统计使用左脚和右脚的数量，需要按 Preffered_Foot 进行分组，计算其 count()值。我们可以使用饼状图来显示左脚右脚选手数量的差别。

**【示例 25-27】使用饼状图来显示左脚右脚选手数量的差别**

```
g = player.groupby('Preffered_Foot')
s = g['Preffered_Foot'].count()
s.plot(kind='pie',autopct='%.2f')
```

执行结果如图 25-26 所示：

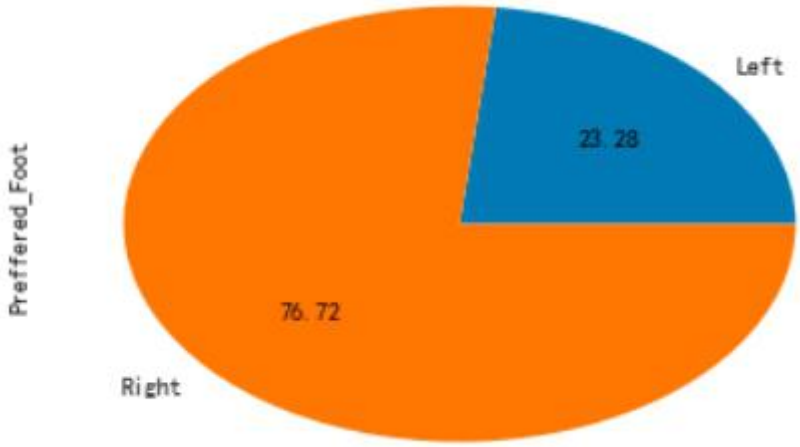


**图 25-26 示例 25-27 运行效果图**

按某列分组计算 count()值可以直接使用 value_counts()方法，示例如下：

**【示例 25-28】value_counts()方法的使用**

```
foot_counts = player['Preffered_Foot'].value_counts()
foot_counts.name=' '       #将左侧 Preffered_Foot 去掉
foot_counts.plot(kind='pie',autopct='%.2f')
```

执行结果如图 25-27 所示：


**图 25-27 示例 25-28 运行效果图**

## 25.3.6 相关性分析

通过散点图查看变量之间关系。

**【示例 25-29】分析足球运动员的身高和体重是否相关**

```
player.plot(kind='scatter',x='Height',y='Weight')
```

执行结果如图 25-28 所示：

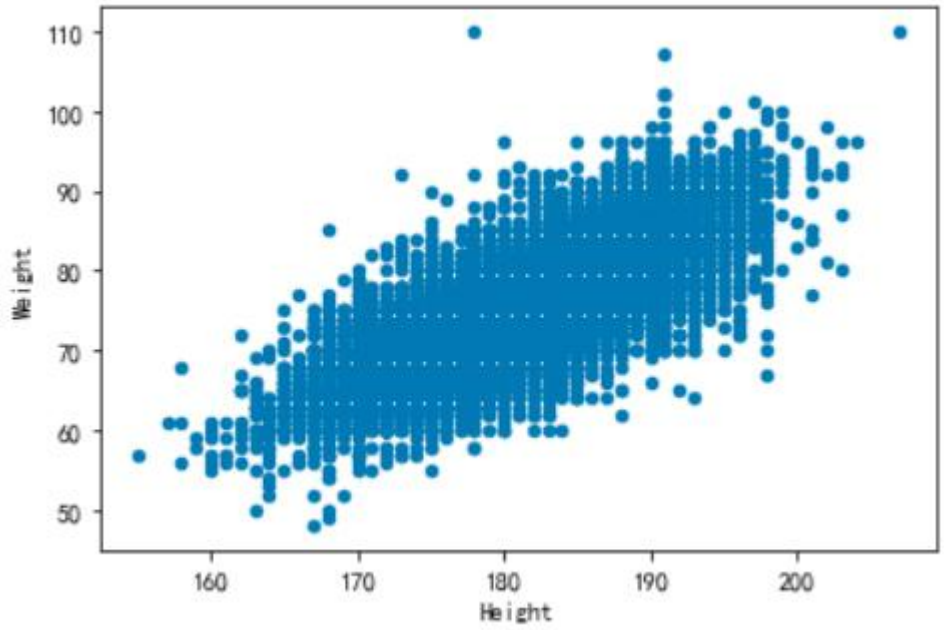


**图 25-28 示例 25-29 运行效果图**

从上面示例可以看到，身高和体重有一定的相关性，身高越高体重越重。

**【示例 25-30】分析足球运动员的身高和评分是否相关**

```
player.plot(kind='scatter',x='Height',y='Rating')
```

执行结果如图 25-29 所示：

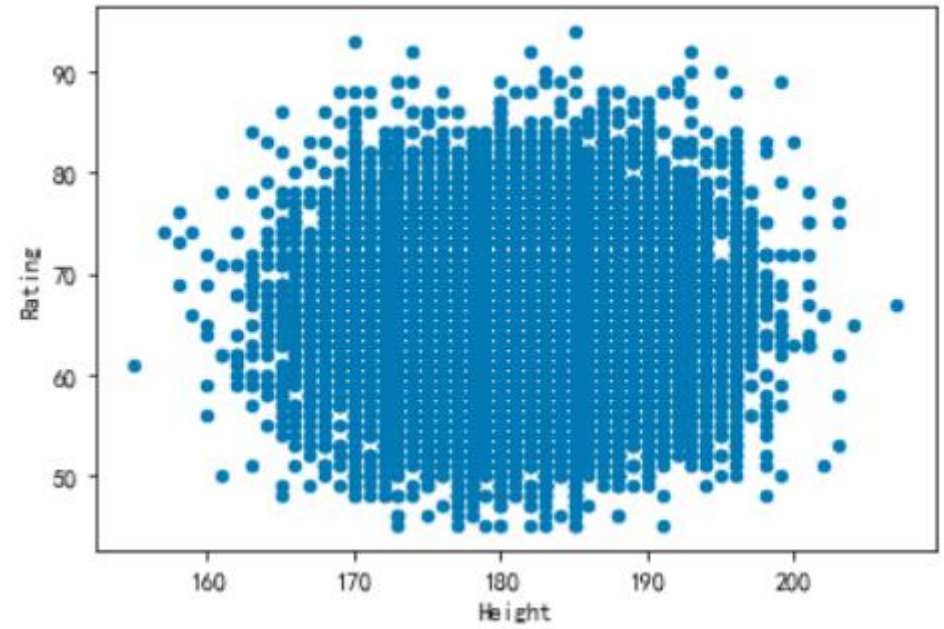

图 25-29 示例 25-30 运行效果图

从上面示例可以看到，身高和评分并没有一定的相关性。查看两个变量是否有相关性，可以通过 corr()方法，该方法返回两个变量的相关系数值，相关系数值越大表示这两个变量相关性越强。相关系数的最大值为 1，即获变量自己和自己的相关系数就是 1。

**【示例 25-31】通过相关系数查看两个变量是否相关**

```
print('身高与身高的相关系数：',player['Height'].corr(player['Height']))
print('身高与体重的相关系数：',player['Height'].corr(player['Weight']))
print('身高与评分的相关系数：',player['Height'].corr(player['Rating']))
```

执行结果如图 25-30 所示：

```
身高与身高的相关系数：  1.0
身高与体重的相关系数：  0.7582075774488368
身高与评分的相关系数：  0.046937095897011116
```

图 25-30 示例 25-31 运行效果图

**【示例 25-32】分析对评分影响前 10 的变量**

```
player.corr()['Rating'].sort_values(ascending=False).head(10)
```

执行结果如图 25-31 所示：

```
player.corr()['Rating'].sort_values(ascending=False).head(10)

Rating          1.000000
Reactions       0.828329
Composure       0.613612
Short_Pass      0.496239
Vision          0.489277
Long_Pass       0.483217
Ball_Control    0.463211
Age             0.458098
Shot_Power      0.441773
Curve           0.420796
Name: Rating, dtype: float64
```

图 25-31 示例 25-32 运行效果图

在上述示例中，首先获取影响评分的相关系数值，再对其进行排序。按照某列排序需要调用 sort_values()方法，在 sort_values()方法中通过 ascending 参数指定是升序还是降序。ascending 参数默认值为 True，表示升序排序，如果想降序需要将 ascending 参数设置为 False。

## 25.4 航班准点分析

数据集为美国各州机场的航班信息，包含出发地，目的地，是否出发延迟 15 分钟，是

否到达延迟 15 分钟等。打开 https://www.transtats.bts.gov/Fields.asp?Table_ID=246 网址进行下载数据集。

### 24.4.1 数据探索和清洗

**【示例 25-33】加载数据集**

```
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
#支持中文显示
mpl.rcParams['font.family']='Kaiti'
# 使用非 unicode 的负号，当使用中文时候要设置
mpl.rcParams['axes.unicode_minus']=False
%matplotlib inline
data = pd.read_csv('data/airport-ontime.csv')
data.info()
```

执行结果如图 25-32 所示：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502617 entries, 0 to 502616
Data columns (total 17 columns):
FL_DATE                 502617 non-null object
UNIQUE_CARRIER          502617 non-null object
ORIGIN_AIRPORT_ID       502617 non-null int64
ORIGIN_AIRPORT_SEQ_ID   502617 non-null int64
ORIGIN_CITY_MARKET_ID   502617 non-null int64
ORIGIN_STATE_ABR        502617 non-null object
DEST_AIRPORT_ID         502617 non-null int64
DEST_AIRPORT_SEQ_ID     502617 non-null int64
DEST_CITY_MARKET_ID     502617 non-null int64
DEST_STATE_ABR          502617 non-null object
DEP_DELAY_NEW           492974 non-null float64
DEP_DEL15               492974 non-null float64
ARR_DELAY_NEW           490716 non-null float64
ARR_DEL15               490716 non-null float64
DISTANCE                502617 non-null float64
DISTANCE_GROUP          502617 non-null int64
Unnamed: 16             0 non-null float64
dtypes: float64(6), int64(7), object(4)
memory usage: 65.2+ MB
```

**图 25-32 示例 25-33 运行效果图**

从上述示例可以看到总共 502617 行、17 列。其中 DEP_DEL15(起飞是否延迟)非空值是 492974 行，ARR_DEL15（到达是否延迟）非空值是 490716 行，Unnamed 这列全部为空值。

**【示例 25-34】删除空值 Unnamed 列**

```
data.dropna(axis=1,how='all',inplace=True)
data.info()
```

执行结果如图 25-33 所示：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 502617 entries, 0 to 502616
Data columns (total 16 columns):
FL_DATE                 502617 non-null object
UNIQUE_CARRIER          502617 non-null object
ORIGIN_AIRPORT_ID       502617 non-null int64
ORIGIN_AIRPORT_SEQ_ID   502617 non-null int64
ORIGIN_CITY_MARKET_ID   502617 non-null int64
ORIGIN_STATE_ABR        502617 non-null object
DEST_AIRPORT_ID         502617 non-null int64
DEST_AIRPORT_SEQ_ID     502617 non-null int64
DEST_CITY_MARKET_ID     502617 non-null int64
DEST_STATE_ABR          502617 non-null object
DEP_DELAY_NEW           492974 non-null float64
DEP_DEL15               492974 non-null float64
ARR_DELAY_NEW           490716 non-null float64
ARR_DEL15               490716 non-null float64
DISTANCE                502617 non-null float64
DISTANCE_GROUP          502617 non-null int64
dtypes: float64(5), int64(7), object(4)
memory usage: 61.4+ MB
```

**图 25-33 示例 25-34 运行效果图**

**【示例 25-35】查看及删除重复数据**

```
#data.duplicated().any()
data.drop_duplicates(inplace=True)
data.info()
```

执行结果如图 25-34 所示：

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 394113 entries, 0 to 502616
Data columns (total 16 columns):
FL_DATE                 394113 non-null object
UNIQUE_CARRIER          394113 non-null object
ORIGIN_AIRPORT_ID       394113 non-null int64
ORIGIN_AIRPORT_SEQ_ID   394113 non-null int64
ORIGIN_CITY_MARKET_ID   394113 non-null int64
ORIGIN_STATE_ABR        394113 non-null object
DEST_AIRPORT_ID         394113 non-null int64
DEST_AIRPORT_SEQ_ID     394113 non-null int64
DEST_CITY_MARKET_ID     394113 non-null int64
DEST_STATE_ABR          394113 non-null object
DEP_DELAY_NEW           386058 non-null float64
DEP_DEL15               386058 non-null float64
ARR_DELAY_NEW           383812 non-null float64
ARR_DEL15               383812 non-null float64
DISTANCE                394113 non-null float64
DISTANCE_GROUP          394113 non-null int64
dtypes: float64(5), int64(7), object(4)
memory usage: 51.1+ MB
```

**图 25-34 示例 25-35 运行效果图**

## 25.4.2 统计起飞是否延迟情况

**【示例 25-36】查询起飞是否延迟**

```
data['DEP_DEL15'].head()
```

执行结果如图 25-35 所示：

```
data['DEP_DEL15'].head()

0    0.0
1    0.0
2    0.0
3    1.0
4    0.0
Name: DEP_DEL15, dtype: float64
```

**图 25-35 示例 25-36 运行效果图**

从上述示例执行结果可以看到，起飞是否延迟通过 0 和 1 来表示。其中 0 表示不延迟，1 表示延迟。要统计起飞延迟与不延迟的数量，我们需要按 DEP_DEL15 列分组。

【示例 25-37】统计起飞延迟与不延迟，使用饼状图来显示

```
    s = data['DEP_DEL15'].dropna()
    delays = s.value_counts()
    display(delays)
    delays.name=''
    delays.plot(kind='pie',labels=['起飞不延迟','起飞延迟'],autopct='%.2f',title='起飞延迟总体情况')
```

执行结果如图 25-36 所示：

```
0.0    260977
1.0    125081
Name: DEP_DEL15, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x17fd02d51d0>
```

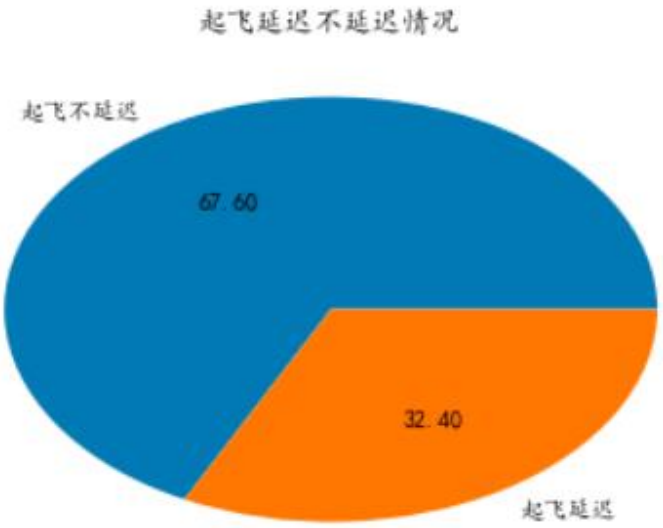


图 25-36 示例 25-37 运行效果图

## 25.4.3 统计到达是否延迟情况

【示例 25-38】统计到达延迟与不延迟，使用饼状图显示

```
    s = data['ARR_DEL15'].dropna()
    delays = s.value_counts()
    display(delays)
    delays.name=''
    delays.plot(kind='pie',labels=['到达不延迟','到达延迟'],autopct='%.2f',title='到达延迟总体
```

```
情况')
```

执行结果如图 25-37 所示：

```
0.0     254200
1.0     129612
Name: ARR_DEL15, dtype: int64

<matplotlib.axes._subplots.AxesSubplot at 0x17fd28e85c0>
```

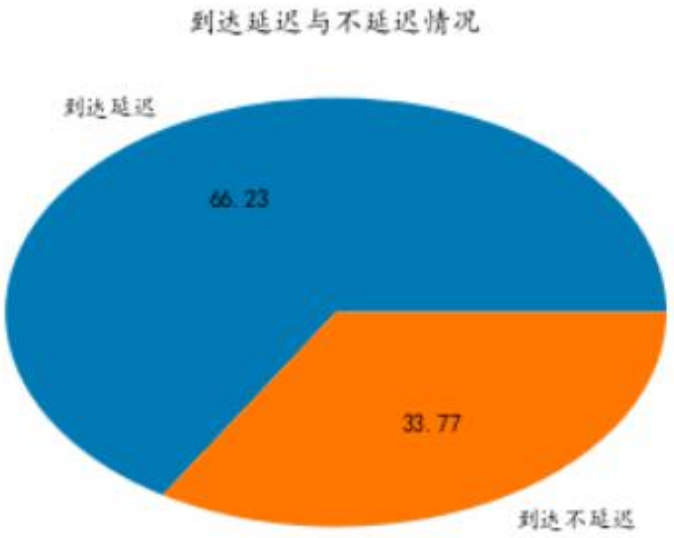


**图 25-37 示例 25-38 运行效果图**

## 25.4.4 统计机场航班起飞、到达延迟数量

**【示例 25-39】统计机场航班起飞延迟数量，使用柱状图显示**

```
# 缺失值处理
d = data[['ORIGIN_STATE_ABR','DEP_DEL15']].dropna()
depart_delay_couots = d.groupby('ORIGIN_STATE_ABR')['DEP_DEL15'].sum()
# 设置画布大小 figsize=(a,b) a 表示画布宽，b 表示画布高，单位英寸
depart_delay_couots.sort_values(ascending=False).plot(kind='bar',figsize=(14,6))
```

执行结果如图 25-38 所示：

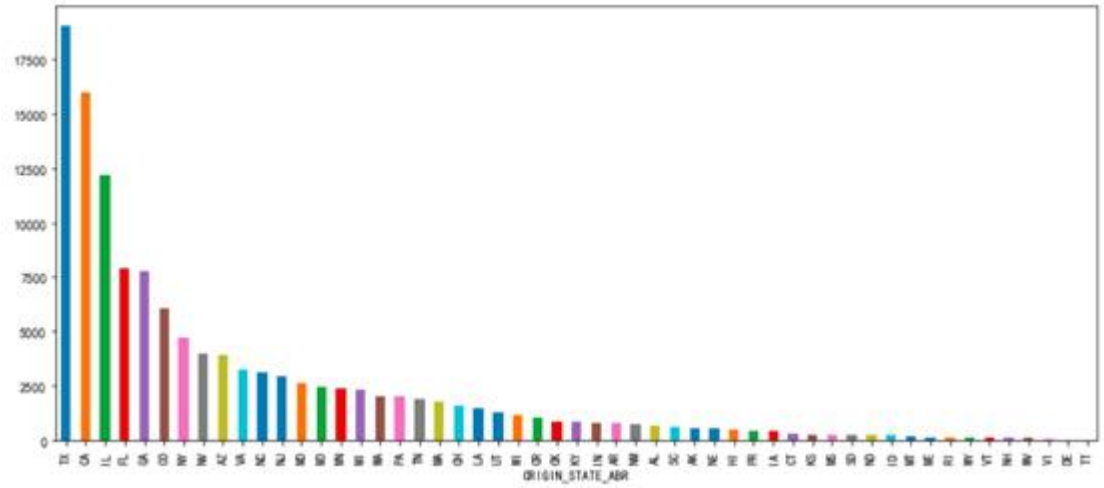


**图 25-38 示例 25-39 运行效果图**

**【示例 25-40】统计机场航班到达延迟数量，使用柱状图显示**

```
# 缺失值处理
d = data[['DEST_STATE_ABR','ARR_DEL15']].dropna()
arrive_delay_couots = d.groupby('DEST_STATE_ABR')['ARR_DEL15'].sum()
# 设置画布大小 figsize=(a,b) a 表示画布宽，b 表示画布高，单位英寸
arrive_delay_couots.sort_values(ascending=False).plot(kind='bar',figsize=(14,6))
```

执行结果如图 25-39 所示：

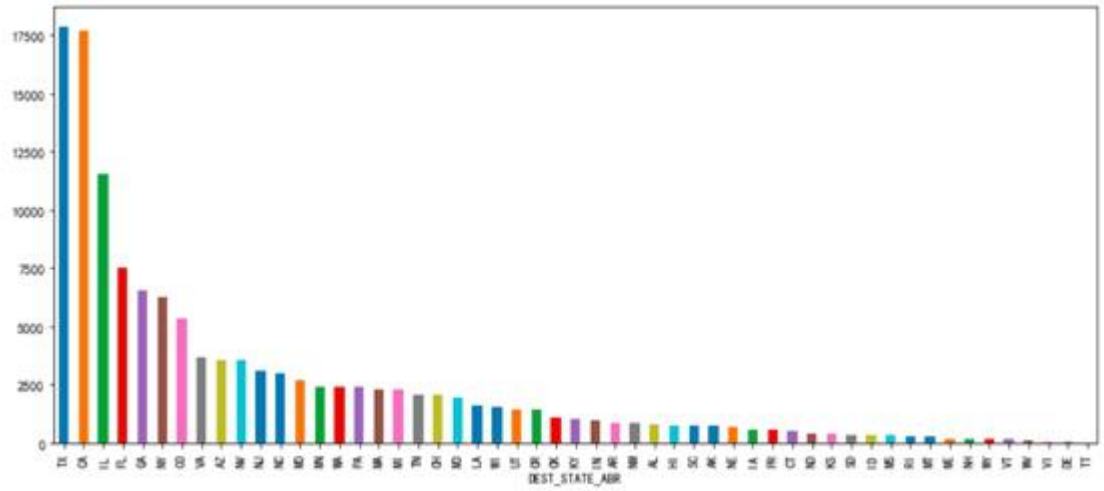


**图 25-39 示例 25-40 运行效果图**

**【示例 25-41】合并机场航班起飞和到达延迟**

```
delay_df=pd.DataFrame([depart_delay_couots,arrive_delay_couots]).T
delay_df.columns=['起飞延迟','到达延迟']
delay_df.sort_values('起飞延迟',ascending=False).plot(kind='bar',figsize=(14,6),title='机场
```

```
起飞到达延迟状况')
```

执行结果如图 25-40 所示：

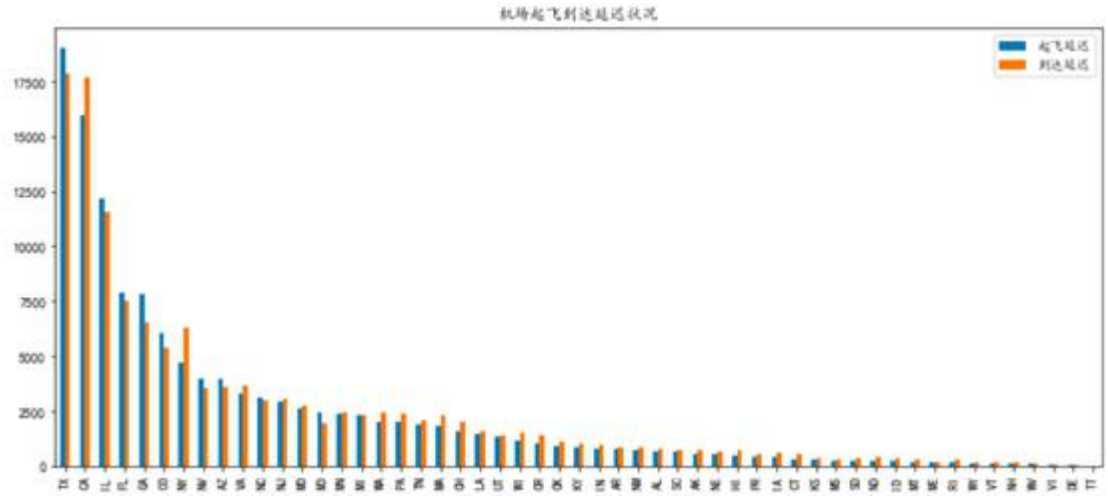


**图 25-40 示例 25-41 运行效果图**

## 25.4.5 统计机场航班起飞、到达延迟的百分比

**1) 机场航班起飞延迟的百分比**

机场航班起飞延迟百分比=机场起飞延迟的航班数/机场航班总起飞数。机场起飞延迟的航班数前面已经获取到变量 depart_delay_couots 中，还需要获取机场总航班，示例如下：

**【示例 25-42】机场起飞总航班数**

```
d = data[['ORIGIN_STATE_ABR','DEP_DEL15']].dropna()
departs = d['ORIGIN_STATE_ABR'].value_counts()
```

**【示例 25-43】机场航班起飞延迟的百分比**

```
pct_departure_delays = depart_delay_couots/departs
```

**2) 机场航班到达延迟的百分比**

同理，机场航班到达延迟百分比=机场到达延迟的航班数/机场航班达数总数。机场到达延迟航班数前面已经获取到变量 arrive_delay_couots 中，还需要获取机场航班到达总数，示例如下：

**【示例 25-44】机场航班到达延迟的百分比**

```
d = data[['DEST_STATE_ABR','ARR_DEL15']].dropna()
#  计算到达航班的数量
arrives = d['DEST_STATE_ABR'].value_counts()
# arrive_delay_couots  机场到达延迟航班数
```

```
    pct_arrive_delays = arrive_delay_couots/arrives
```

**【示例 25-45】组合起飞延迟和到达延迟百分比，柱状图描述**

```
    # 将起飞延迟和到达延迟组合成 DataFrame，柱状图描述
    pct_delay_df=pd.DataFrame([pct_departure_delays,pct_arrive_delays]).T
    pct_delay_df.columns=['起飞延迟比例','到达延迟比例']
    # display(pct_departure_delays,pct_arrive_delays)
    pct_delay_df.sort_values('起飞延迟比例',ascending=False).plot(kind='bar',title='机场起飞到达延迟百分比',figsize=(14,6))
```

执行结果如图 25-41 所示：

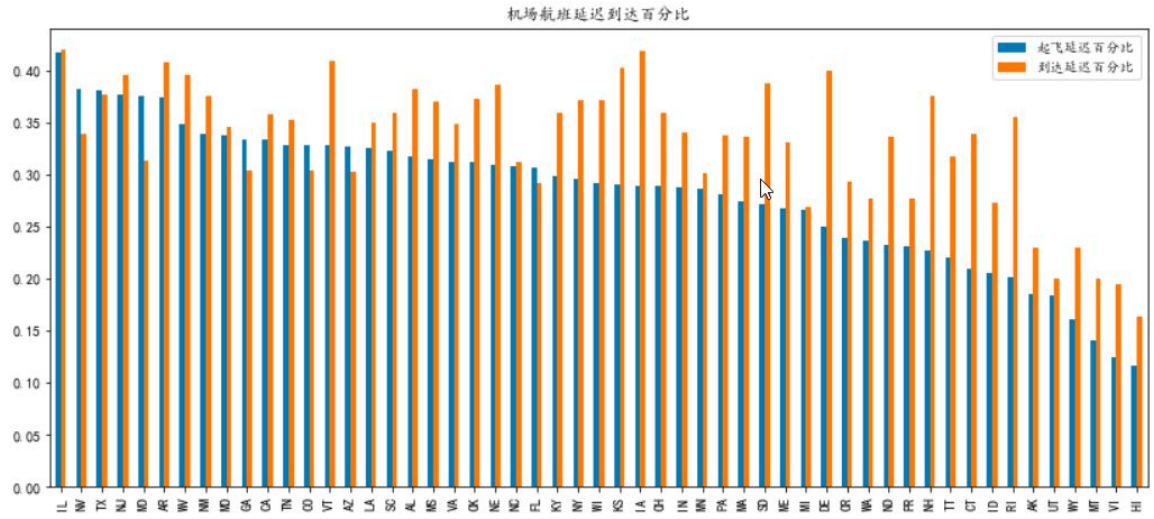


**图 25-41 示例 25-45 运行效果图**

# 习题

## 一、编码题

1. **Python 操作 Excel，完成对 Excel 的读和写。**
2. **导入.xlsx 文件，并完成如下操作：**
   a. 指定导入哪个 Sheet。
   b. 指定行索引。
   c. 指定列索引。
   d. 指定导入列。
3. **设有如下数据集，完成统计分析**

```
import pandas as pd
df = pd.DataFrame({'applied_from': [1,1,1,1,1,2,2,2,2,2,2],
                   'applied_type': [0,0,0,1,1,0,0,0,1,1,1],
                   'refused_id':     [100,100,100,102,102,102,102,102,102,102,100],
                                                                    'name':
['mark','python','C','HAHA','XX','XIXI','ROSE','RICHAR','BINGO','BUFF','YEVON_OU'],
                   'age':            [18,19,22,12,22,24,33,44,55,66,77],
                                                                    'country':
['china','china','USA','japan','japan','USA','uk','uk','uk','uk','USA']})
df
```

   a. 获取 applied_from 字段。
   b. country 中的类别个数计算。
   c. 对 country 分组再求和。
   d. 对 countrys 中 age 按照降序排列。
   e. 求 age 最大的索引值。
   f. 查看 age 年龄最小的这一行数据。
   g. 计算年龄标准差。
   h. df 中描述性统计信息查看。