第 3 章　数据清洗

数据的获取是复杂而艰苦的，数据的来源也是多种多样的。数据可视化过程中的大部分工作是将大量数据可视化，因此，数据中的图片、声音、视频等多媒体基本不需清洗。但大量数字和文字的数据清理（data cleansing，有时候也用 data cleaning 或 data scrubbing 表示）往往需要占用整个工作量的 80% [1]。数据清洗是一个漫长而复杂的过程，虽然有前人总结的方法和技巧，但是每次数据清洗都可能遇到新的问题。每个人对数据的理解不同，处理方案也不一样，所以理论上说，不同的人清洗同一个数据集，最后得到的数据集是不一样的。为了更好地理解干净的数据（tidy data），首先规范以下定义。

变量（Variable）是指一个度量或一个属性，如身高、体重、性别等。

值（Value）是指实际的测量或属性值，如 1.75 米、72.3 kg、男性等。

观察（Observation）是指在同一个个体上测量的所有值，如每个人的全部变量值。

干净的数据必须具备以下三个特征。其一，每个变量构成一列并包含非空列值；其二，每次观察构成一行；其三，每类观察个体组成一个表。如多个人组成一个包含 3 列（身高、体重、性别）的表，表中的缺失值、重复行记录、单位不统一（如身高的单位有的是米，有的是厘米）、格式错误、无列头等数据均是“脏数据（messy data）”。

来源复杂的数据不仅需要格式转换，还需要数据清洗后才能进行数据分析、数据诠释和数据可视化。数据清洗是数据处理过程中最重要的环节，如果无法保证数据的正确性，则基于“脏数据”的任何后续工作都是毫无意义的。数据清洗是将数据中的“脏数据”清洗掉，变成干净的数据，提高数据质量。对数据进行重新审查和校验的目的在于处理缺失数据、规范数据格式内容、避免逻辑错误、删除非需求数据、对来自多个数据源的数据进行分组或合并转换、保证数据内部和外部的一致性等。数据清洗是对源数据的不可逆转修改，无法恢复，所以数据清洗前务必备份源数据。

本章使用 Python 的 Pandas 包完成上述操作，一般用 Jupyter Notebook 进行 Python 代码的编写和运行。本章主要探究数值和文本数据的清洗方法和技巧，需要用到 Python 的 Pandas 包[2]，具体安装方法见 2.8.1 节，也可以到其官方网站[3]查看最新版本的系统要求、安装方法和详细步骤。本章所有案例基于 Windows 系统。在 Mac 系统上运行，格式可能有所不同。

3.1　Jupyter Notebook

Jupyter Notebook 是一个交互式应用程序，在网页浏览器环境中运行，其界面主要以“单

[1] Megan Squire．干净的数据：数据清洗入门与实践．任政委译．人民邮电出版社，2016.

[2] 下载 Pandas 库的官方网站 http://pandas.pydata.org

[3] 安装 Python 库的官方网站 https://packaging.python.org/tutorials/installing-packages

元格”（cell）为基础。用户可以直接在网页中编写代码、运行代码，也可以在同一个页面中直接编写说明和注释文档，代码的运行结果也直接显示在同一个页面的代码行下面。

Jupyter Notebook 的优势是内核不需运行 Python，用户在 Web 应用中编写代码，代码通过服务器发送给内核，内核运行代码，并将结果发送回该服务器，任何输出都会返回到浏览器中。Jupyter Notebook 页面简洁大方，便于提升用户工作效率。

3.1.1 安装 Jupyter Notebook

安装 Jupyter Notebook [4]前，需要安装 Python，最好安装 3.3 版本及以上。注意，Python 3 对 Python 2 的兼容性较差，几乎所有的 Python 2 程序都需要一些修改才能正常运行在 Python 3 环境下。Python 官方建议直接学习 Python 3，且 Python 2 只维护到 2020 年。

首先，把 pip 升级到最新版本，如果是 Python 3.x，则语句如下：

```
pip3 install --upgrade pip
```

如果是 Python 2.x，则语句如下：

```
pip install --upgrade pip
```

Python 3.x 安装 Jupyter Notebook 的语句如下：

```
pip3 install jupyter
```

Python 2.x 安装 Jupyter Notebook 的语句如下：

```
pip install jupyter
```

3.1.2 启动、关闭 notebook 服务器

启动 notebook 服务器的方法是在终端输入“jupyter notebook”，终端会显示一系列 notebook 的服务器数据，同时浏览器自动启动 Jupyter Notebook。Windows 操作系统界面见图 3.1，Mac 操作系统界面见图 3.2。注意，在 Jupyter Notebook 操作过程中必须保持终端的运行状态，不能关闭。

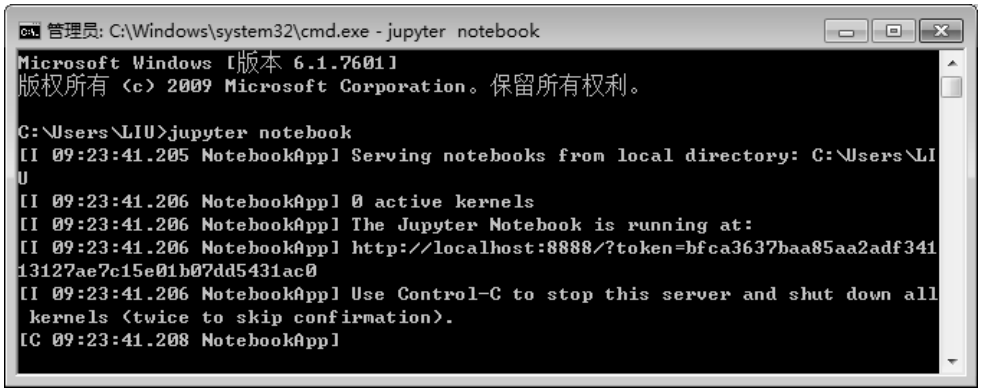


图 3.1　Windows 操作系统启动 Jupyter Notebook

[4]　官网文档资料 https://jupyter-notebook.readthedocs.io/en/stable/notebook.html

图 3.2　Mac 操作系统启动 Jupyter Notebook

若对“jupyter notebook”命令有疑问，可以查看官方帮助文档，命令如下：

```
jupyter notebook -help
```

或

```
jupyter notebook -h
```

正常启动服务器后，浏览器的地址栏默认显示 http://localhost:8888，运行界面见图 3.3。其中，“localhost”指本地计算机，“8888”是端口号。注意，虽然默认端口号是“8888”，但是若启动了多个 Jupyter Notebook，或者通过用户自定义等方法，均可修改端口号。

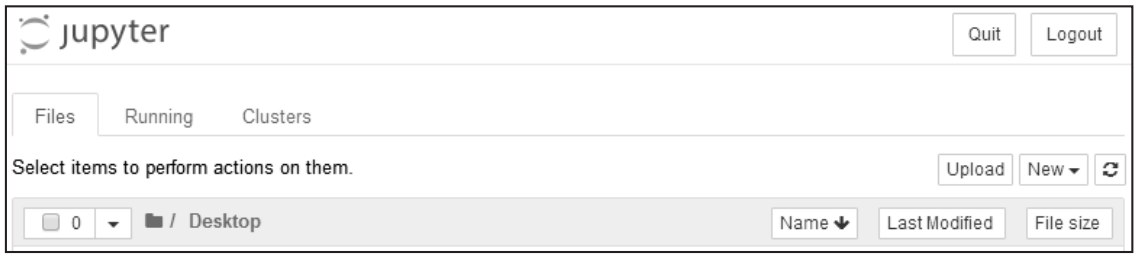


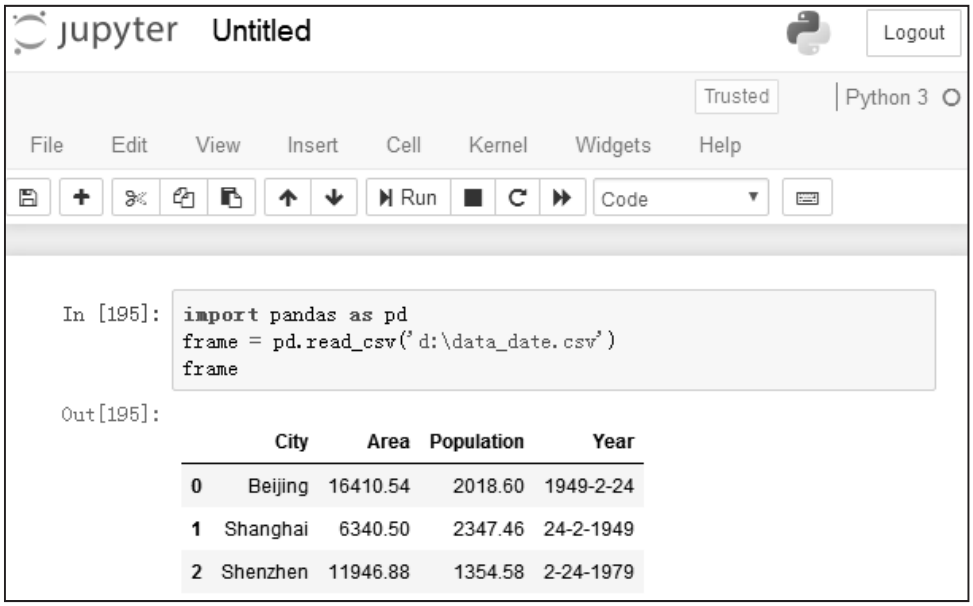
图 3.3　运行 Jupyter Notebook

单击“New”按钮，选择“Python”，即可新建一个终端。在终端输入语句后，单击“Run”按钮，运行后可见运行结果，见图 3.4。

在“Files”页面关闭 notebook，进入“Files”页面，见图 3.5。正在运行的 notebook 图标为绿色，见图 3.5 中的“Untitled.ipynb”，其右侧还有 notebook 的运行状态“Running”。然后勾选要关闭的 notebook，单击左上方的黄色按钮“Shutdown”，即可关闭 notebook。关闭后的 notebook 图标为灰色。

“Files”页面只能关闭 notebook，无法关闭终端。在“Running”页面中可以同时关闭 notebook 和终端。进入“Running”页面（见图 3.6），“Terminals”栏显示正在运行的所有终端，“Notebooks”栏显示正在运行的所有 notebook。单击终端或 notebook 右侧的黄色“Shutdown”按钮，可以关闭终端或 notebook。

无论是关闭 notebook 还是关闭终端，Jupyter Notebook 服务器都在运行中，若想关闭服务器，要在 Windows 操作系统终端按组合键 Ctrl+C，在 Mac 操作系统终端上按组合键 Ctrl+Q。



	City	Area	Population	Year
0	Beijing	16410.54	2018.60	1949-2-24
1	Shanghai	6340.50	2347.46	24-2-1949
2	Shenzhen	11946.88	1354.58	2-24-1979

图 3.4　Jupyter Notebook 的“Files”页面

图 3.5　在“Files”页面关闭 notebook

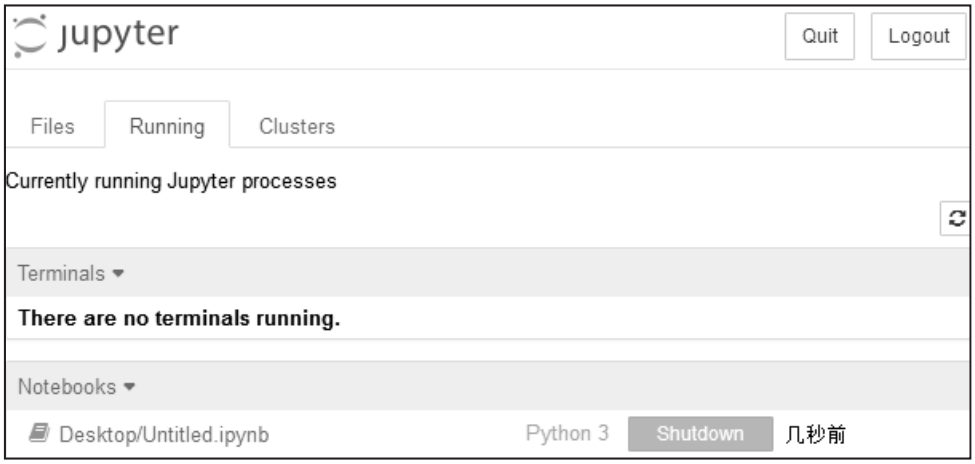


图 3.6　在“Running”页面关闭 notebook 和终端

3.1.3 保存 notebook

在 Jupyter Notebook 中编写的代码可以保存成文档，默认的后缀名为 .ipynb 的 JSON 格式，在终端页面的“File”菜单中可以保存文件或另存为其他格式。也可以根据需要，导出成 PY、HTML 和 PDF 等文档格式。

如有需要，可以设置 Jupyter Notebook 文件存放位置，具体方法见官方网站。

3.2 Pandas 包

Pandas 是 Python 的一个数据分析包，2009 年底由 AQR Capital Management 推出，是开源的，现在的最高版本是 0.25.1（至 2019 年 10 月[5]）。

Pandas 是一个提供快速、灵活和表达性数据结构的 Python 包，可以处理系列（Series）、数据帧（DataFrame）和面板（Panel）三种数据结构，见表 3.1。Series 是一维数据。DataFrame 是二维数据，可以理解为 Series 的容器。Panel 是三维数组，可以理解为 DataFrame 的容器。Pandas 包使用最广泛的是 Series 和 DataFrame。

表 3.1 Pandas 包可以处理的三种数据结构

数据结构	维 数	描 述
系列（Series）	1	均匀数组，大小不变
数据帧（DataFrame）	2	大小可变的表结构与潜在的异质类型的列
面板（Panel）	3	大小可变数组

3.2.1 系列（Series）

系列（Series）是带有标签的一维数组，可以保存任何数据类型（如整数、字符串、浮点数、对象等），标签统称为索引（Index）。系列可以直接输入数据，也可以通过 list 数据赋值。Series 可以赋值索引，也可以由系统自动给定。

1．自动索引 Series 语句

若没有自定义索引，则 Series 语句自动添加索引，默认情况下，Series 索引从 0 开始，步长为 1，连续递增。新建一个程序文件 test.py，编辑运行代码，或者启动 Jupyter Notebook 服务器查看运行代码。例如：

```
import pandas as pd
s = pd.Series([1,2,3,4,5])
print(s)
```

程序运行结果如下：

```
0    1
1    2
2    3
```

[5] http://pandas.pydata.org

```
3      4
4      5
dtype: int64
```

说明：

① 第 1 条语句的含义是导入 Pandas 包，并用别名“pd”代替它。使用别名主要是因为有些包名比较长，书写不便，用简短的别名代替可以提高编程效率。

② 第 2 条语句是为 Series 对象赋元素值，无索引值。

③ 输出结果中的左列表示每个元素的自动索引，右列表示相应的元素值。最后一行输出显示元素的数据类型是 64 位的整型数据。

2．自定义索引 Series 语句

索引是可以自定义的，索引个数必须与元素个数相同。例如：

```
import pandas as pd
s = pd.Series([1,2,3,4,5], index=['a', 'b', 'c', 'd', 'e'])
print(s)
```

程序运行结果如下：

```
a      1
b      2
c      3
d      4
e      5
dtype: int64
```

说明：

① 第 2 条语句不仅为元素赋值，还自定义了元素的索引值。

② 最后一行输出显示的是元素的数据类型，而非索引的数据类型。

思考：修改上述代码，将索引个数减少到 4 个，元素个数不变，查看运行结果；再将元素个数减少到 3 个，查看运行结果。

3．通过 list 创建 Series 语句

可以将 list 直接赋值给 Series，相当于 Series 只有元素值，默认的索引值。例如：

```
import pandas as pd
list_test = [99,87,72,86]
s = pd.Series(list_test)
print(s)
```

程序运行结果如下：

```
0      99
1      87
2      72
3      86
dtype: int64
```

说明：第 2 条语句是为 list 赋值，第 3 条语句是将 list 值赋值给 Series。

4．从标量值创建 Series 语句

通过标量值创建 Series，虽然标量值只有一个，但是序列的个数决定了标量值的个数。例如：

```
import pandas as pd
s = pd.Series(99,index =['a','b','c','d'])
print(s)
```

程序运行结果如下：

```
a      99
b      99
c      99
d      99
dtype: int64
```

说明：第 2 条语句设置了 4 个索引和 1 个标量值，根据索引和元素个数必须相同的原则，此标量被赋值了 4 次。

5．从字典类型创建 Series 语句

通过字典类型创建 Series，键值对中的“键”是索引。例如：

```
import pandas as pd
s = pd.Series({'a':1,'b':2,'c':3})
print(s)
```

程序运行结果如下：

```
a      1
b      2
c      3
dtype: int64
```

说明：键值对必须是一一对应的，一个索引对应一个元素值。

6．Series 类型的基本操作

Series 包括索引和元素两部分，可以使用 index 和 values 单独引用。例如：

```
import pandas as pd
s = pd.Series({'a':1, 'b':2, 'c':3})
print(s.index)
print(s.values)
```

程序运行结果如下：

```
Index(['a', 'b', 'c'], dtype = 'object')
[1 2 3]
```

说明：

① 第 3 条语句输出 Series 的索引。

② 第 4 条语句输出 Series 的索引元素，即一个数组对象。

3.2.2 数据帧（DataFrame）

DataFrame 是表格型的数据类型，每列数据值的类型均可以不同，既有行索引也有列索引，所以 DataFrame 对象包含数据、横轴和竖轴三部分。数据可以手工输入，也可以来自 Series 对象，更多来自 Excel 或 CSV 文件，被读取到 DataFrame 对象后再进行计算和处理。

1．直接赋值的无行列索引 DataFrame 语句

例如：

```
import pandas as pd
frame = pd.DataFrame([["北京",16410.54],["上海",6340.5],["深圳",11946.88]])
print(frame)
```

程序运行结果如下：

```
     0          1
0  北京   16410.54
1  上海    6340.50
2  深圳   11946.88
```

说明：

① 第 2 条语句是为 DataFrame 对象赋值，方式为水平赋值，即赋值语句的“,”前后均是一条完整的数据记录，本例中包含 3 条记录，每条记录包含 2 个元素。

② 本例中的“北京”“上海”和“深圳”要加双引号，表明是字符串而非变量名。

2．用 Series 赋值的 DataFrame 语句

例如：

```
import pandas as pd
frame = pd.DataFrame({"城市":pd.Series(["北京", "上海", "深圳"]), "面积":pd.Series([16410.54,\
    6340.5, 11946.88])})
print(frame)
```

程序运行结果如下：

```
    城市        面积
0  北京   16410.54
1  上海    6340.50
2  深圳   11946.88
```

说明：

① 第 2 条语句为 DataFrame 对象赋值，方式为垂直赋值，即赋值语句的“,”前后均是一列完整的数据，本例中包含“城市”和“面积”2 列，每列包含 3 个值，即 3 条记录。

② 语句 pd.Series(["北京", "上海", "深圳"]是从字典类型创建一个 Series 对象，列索引值是“城市”。

3．DataFrame 数据的切片和转置

例如：

```
import pandas as pd
frame = pd.DataFrame({"城市":pd.Series(["北京","上海","深圳"]),"面积":pd.Series([16410.54, \
```

```
   6340.5, 11946.88])})
print(frame.城市)
print(frame.T)
```

程序运行结果如下：

```
0  北京
1  上海
2  深圳
Name: 城市, dtype: object
       0           1           2
城市    北京          上海          深圳
面积    16410.5     6340.5      11946.9
```

说明：

① 语句 frame.城市等同于语句 frame["城市"]，功能是返回 frame 中“城市”列的内容，内容返回到一个 Series 对象。

② T 是转置操作，将 DataFrame 数据集解释为矩阵，再执行矩阵的转置计算。DataFrame 对象的常见操作见表 3.2。

表 3.2　DataFrame 对象的常见操作

操　作	功　能	举　例	结　果
head(int n)	显示数据的前 *n* 行	frame.head(2)	城市　面积 0　北京　16410.54 1　上海　6340.50
tail(int n)	显示数据的后 *n* 行	frame.tail(2)	城市　面积 1　上海　6340.50 2　深圳　11946.88
describe( )	对每列数据进行统计，包括计数、均值、标准差、各分位数等	frame.describe( )	面积 Count　3.000000 mean　11565.973333 std　5045.814485 min　6340.500000 25%　9143.690000 50%　11946.880000 75%　14178.710000 max　16410.540000

4．通过 Excel 文件创建 DataFrame 对象

例如：

```
import pandas as pd
frame = pd.read_excel('D:\data.xlsx')
frame
```

程序运行结果如下：

```
   城市      面积        人口
0  北京    16410.54    2018.60
1  上海     6340.50    2347.46
2  深圳    11946.88    1354.58
```

说明：

① 第 2 条语句将 Excel 文件读取到 DataFrame 对象。若运行时出现“No module named 'xlrd'”错误，说明未安装 xlrd 模块。xlrd 是第三方导入 Excel 表格的模块，需要自行安装。若计算机安装的是 Python 2.x 版本，则需在终端执行“pip install xlrd”命令安装；若计算机安装的是 Python 3.x 版本，则需在终端执行“pip3 install xlrd”命令安装。

② 第 2 条语句在读取 Excel 文件时要写清文件的盘符、路径、文件名和文件扩展名。语句 pd.read_excel('D:\data.xlsx')指明文件在 D 盘根目录下。

③ Excel 包含多个表，若没有特别说明，默认情况下读取表 Sheet1，且返回表 Sheet1 的全部数据。如果 Sheet1 表的数据文件首行不是列元素，则读取数据的代码如下：

```
pd.read_excel('D:\data.xlsx', header = None)
```

5．通过 CSV 文件创建 DataFrame 对象

例如：

```
import pandas as pd
frame = pd.read_csv('D:\data.csv')
frame
```

程序运行结果如下：

```
      City         Area       Population
0    Beijing     16410.54     2018.60
1   Shanghai      6340.50     2347.46
2   Shenzhen     11946.88     1354.58
```

说明：

① 第 2 条语句在读取 CSV 文件时，默认情况下使用“,”分隔符解析。若 CSV 文件使用其他分隔符，需要为 sep 参数赋值。例如，读取空格分隔符的 CSV 文件时，语句为

```
pd.read_csv('D:\data.csv', sep = ' ')
```

② 若 CSV 文件读取时编码有误，可以通过参数设置，如语句 encoding='utf8'指定字符集类型是“utf-8”。

3.3 清洗缺失值

缺失值是最常见的“脏数据”。缺失值的存在会影响数据分析结果，不完整的记录在分析和可视化时缺失实际意义。

有的数据工程师习惯先清洗缺失值，有的习惯先删除重复记录。建议初学者先清洗缺失值，常见方法包括 isnull( )、notnull( )、dropna( )和 fillna( )。方法 isnull( )和 notnull( )用于判断元素中的缺失值。方法 dropna( )用于删除缺失值。方法 fillna( )用于填充缺失值。

3.3.1 检查缺失值

Pandas 使用 isnull( )和 notnull( )方法（函数）检查缺失值，返回一个布尔型数组。为了使

用这两个方法，先创建一个包含有缺失值的 DataFrame 对象。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
print(frame)
```

程序运行结果如下：

```
     0    1     2
0  1.0  2.0  None
1  3.0  NaN  None
2  NaN  NaN  None
3  4.0  5.0  None
```

说明：

① 第 2 条语句为 DataFrame 对象赋值，包含 7 个 None 值。但程序的运行结果中不仅包含 None 值，还包含 NaN 值。虽然 None 和 NaN 都用来表示空缺的数据，但二者的行为在不同的场景下有相当大的差异。在 Pandas 中，如果其他数据都是数值类型，Pandas 会把 None 自动替换成 NaN。本例中，第一列和第二列包含其他数值类型，所以其中的 None 自动被替换成 NaN。

② None 值源自 Python，是 Python 的一种特殊的数据类型。NaN 源自 Pandas 和 Numpy 包，是一种特殊的浮点（float）数据类型。

方法 isnull( )与 notnull( )的使用方式类似，功能相反。例如，使用 isnull( )方法检查数据是否有缺失值：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
frame.isnull()
```

程序运行结果如下：

```
       0      1     2
0  False  False  True
1  False   True  True
2   True   True  True
3  False  False  True
```

isnull( )方法是对各元素的遍历，返回 DataFrame 对象中每个元素是否是空值。若返回 True，表示数据是空值。如果数据多，则返回的阵列也多，很难快速发现缺失值，可以使用 any( )方法定位存在缺失值的列。先为 DataFrame 对象追加两条记录，再定位包含缺失值的列。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [99,None,None], [4,5,None]])
frame_new = pd.DataFrame([[6,7,8], [9,10,11]])
frame = frame.append(frame_new)
print(frame)
print(frame.isnull().any())
```

程序运行结果如下：

```
     0     1     2
0    1   2.0  None
1    3   NaN  None
2   99   NaN  None
3    4   5.0  None
0    6   7.0     8
1    9  10.0    11

0    False
1     True
2     True
dtype: bool
```

说明：

① 结果的第一部分是输出追加两条记录后的 DataFrame 对象。

② 结果的第二部分是使用 any( )方法定位存在缺失值的列，本例中数据包含 3 列，只有第 0 列返回 False，不存在缺失数据，其他两列返回 True，均存在缺失数据。

3.3.2 删除含缺失值的行或列

若行数据包含缺失值，可以使用 dropna( )方法删除包含缺失值的记录行。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
frame_new = pd.DataFrame([[6,7,8], [9,10,11]])
frame = frame.append(frame_new)
print(frame.dropna(how = 'all'))
print(frame.dropna())
```

程序运行结果如下：

```
     0     1     2
0  1.0   2.0  None
1  3.0   NaN  None
3  4.0   5.0  None
0  6.0   7.0     8
1  9.0  10.0    11

     0     1     2
0  6.0   7.0     8
1  9.0  10.0    11
```

说明：

① 结果的第一部分显示删除全部为空值的行记录。方法 dropna(how='all') 是 dropna(how='all', axis=0)的简写，表示删除全部是空值的行记录。方法 dropna(how='all', axis=1) 表示删除全部是空值的列，这种情况相对较少。

② 第二部分使用 dropna( )方法，删除所有含缺失值的行，即仅输出不包含缺失值的行。

3.3.3 填充缺失值

方法 fillna( )用来填充缺失值。方法中的参数可以设置不同的标量值、字典、用数据本身

的值或统计值填充。

1．标量值填充缺失值

将“NaN”替换为 0 值是一种常见的填充缺失值的方法。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
frame.fillna(0)
```

程序运行结果如下：

```
     0    1    2
0  1.0  2.0    0
1  3.0  0.0    0
2  0.0  0.0    0
3  4.0  5.0    0
```

说明：第 2 条语句为 DataFrame 对象赋值 7 个 None，包含 4 个 0 和 3 个 0.0，具体显示为 0 或者 0.0 则与列的数据类型有关。

2．字典填充缺失值

方法 fillna( )传递一个字典说明对某一列填充的具体值。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
frame.fillna({0:0,1:1, 2:2})
```

程序运行结果如下：

```
     0    1    2
0  1.0  2.0    2
1  3.0  1.0    2
2  0.0  1.0    2
3  4.0  5.0    2
```

说明：

① 第 3 条语句为 DataFrame 对象第 0 列缺失值赋值为 0，第 1 列缺失值赋值为 1，第 2 列缺失值赋值为 2。

② 输出结果中，第 0 列缺失值显示为 0.0，第 1 列缺失值赋值为 1.0，第 2 列缺失值赋值为 2。显示值是否包含小数与列的数据类型相关。

3．用数据本身的值填充缺失值

方法 fillna(method='ffill')将前一行的记录值赋值给下一行记录的缺失值。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None], [3,None,None], [None,None,None], [4,5,None]])
frame.fillna(method = 'ffill')
```

程序运行结果如下：

```
     0    1     2
0  1.0  2.0  None
1  3.0  2.0  None
2  3.0  2.0  None
3  4.0  5.0  None
```

说明：行索引为 2，列索引为 0 的缺失值，用其前一行记录的相应值“3.0”替代。其他缺失值的替代方法类似。

4．用统计值填充缺失值

也可以使用统计值对缺失值进行填充，如均值 mean( )、中位数 median( )、众数 most_frequent( )等，默认统计值是 mean( )，axis=0。例如：

```
import pandas as pd
frame = pd.DataFrame([[1,2,None],[3,None,None], [None,None,None],[4,5,None]])
print(frame.fillna(frame.mean()))
```

程序运行结果如下：

```
          0       1      2
0   1.00000     2.0    NaN
1   3.00000     3.5    NaN
2   2.66667     3.5    NaN
3   4.00000     5.0    NaN
```

说明：

① 行索引为 2、列索引为 0 的缺失值用列索引为 0 的非缺失值的均值替代，即(1+3+4)/3 = 2.666667。

② 行索引为 1、列索引为 1 的缺失值和行索引为 2、列索引为 1 的缺失值用列索引为 1 的非缺失值的均值替代，即(2+5)/2 = 3.5。

③ 行索引为 2 的整个列没有非缺失值，因此该列依旧为缺失值。

3.4 清洗格式内容

人工收集、手工录入，或者来源不同的数据通常在格式和内容方面会存在一些问题，如不同国家或地区的日期型数据格式可能是不同的，“2020-02-24”“24-02-2020”和“02-24-2020”均表示 2020 年 2 月 24 日；也可能存在全半角的问题，如“Beijing”和“Ｂｅｉｊｉｎｇ”都表示同一个城市；也可能存在空格的问题，如“　北京　”和“北　京”都表示同一个城市“北京”。虽然用户可以理解数据的这些格式和内容问题，但计算机会将这类仅仅是格式不同内容相同的数据认为是不同的数据，在做分类汇总等数据分析时出现错误。

Python 使用函数和方法可以删除字符串中的空格、转换大小写和更改数据格式等。

3.4.1 删除字符串中的空格

数据的字符串前后或中间可能存在空格，如“北京”可能保存为“北　京”“　北京”或“北京　”等。数据空格主要是指字符串的头部、尾部和中间的空格。Python 包含 3 个去除字符串空格的内建函数，分别是：lstrip( )函数，功能是去掉字符串左边的空格或指定字符；rstrip( )函数，功能是去掉字符串末尾的空格；strip( )函数，功能是在字符串上执行 lstrip( )函数和 rstrip( )函数。

函数 map(function)将应用于 Series 对象中的每个元素，参数 function 是 Python 的函数

名。实现删除字符串空格的语句如下：

```
import pandas as pd
frame = pd.DataFrame({"城市":pd.Series([" 北 京 ","上海 "," 深圳"]), \
    "面积":pd.Series([16410.54, 6340.5,11946.88])})
print(frame)
print(frame["城市"].map(str.lstrip))
print(frame["城市"].map(str.rstrip))
print(frame["城市"].map(str.strip))
```

程序运行结果如下：

```
      城市       面积
0   北 京   16410.54
1     上海    6340.50
2      深圳  11946.88
0   北 京
1     上海
2      深圳
Name: 城市, dtype: object
0    北 京
1      上海
2      深圳
Name: 城市, dtype: object
0   北 京
1     上海
2    深圳
Name: 城市, dtype: object
```

说明：语句 frame["城市"].map(str.lstrip))的含义是去掉“城市”列的左边空格。

3.4.2 大小写转换

英文字符的大小写混用会在数据分类汇总分析时导致错误。清洗大小写混用是数据清洗的必要步骤之一。Python 包含 4 个字符大小转换的内建函数：upper( )函数的功能是将小写字母转换为大写；lower( )函数的功能是将大写字母转换为小写；swapcase( )函数的功能是将大写转换为小写，小写转换为大写；title( )函数的功能是返回“标题化”的字符串，就是所有单词都是以大写开始，其余字母均为小写。实现字符全部大写转换的语句如下：

```
import pandas as pd
frame = pd.read_csv('D:\data.csv')
print(frame)
frame['City'] = frame['City'].map(str.upper)
print(frame)
```

程序运行结果如下：

```
       City      Area  Population
0   Beijing  16410.54     2018.60
1  Shanghai   6340.50     2347.46
2  Shenzhen  11946.88     1354.58
       City      Area  Population
0   BEIJING  16410.54     2018.60
1  SHANGHAI   6340.50     2347.46
2  SHENZHEN  11946.88     1354.58
```

思考：用 lower( )函数、swapcase( )函数和 title( )函数替代上例中的 upper( )函数，程序运行效果是什么？

Python 提供判断字符大小写的函数。函数 islower( )判断字符串中的字符是否都是小写，若是，则返回 True，否则返回 False。函数 istitle( )判断字符串是否是标题化的，若是，则返回 True，否则返回 False。函数 isupper( )判断字符串中的字符是否都是大写，若是，则返回 True，否则返回 False。判断字符是否全部是小写的语句如下：

```
import pandas as pd
frame = pd.read_csv('D:\data.csv')
frame['City'] = frame['City'].map(str.lower)
print(frame)
print(frame['City'].apply(lambda x: x.islower()))
```

程序运行结果如下：

```
       City      Area  Population
0   beijing  16410.54     2018.60
1  shanghai   6340.50     2347.46
2  shenzhen  11946.88     1354.58
0    True
1    True
2    True
Name: City, dtype: bool
```

说明：

① 方法 apply( )的参数较多，具体格式是

```
apply(func, axis=0, broadcast=False, raw=False, reduce=None, args=( ), **kwargs)
```

功能是当函数参数已经存在于一个元组或字典中时，间接地调用函数，元素的参数是有序的，必须与 object( )形式参数的顺序一致，返回值是 object( )的返回值。其中，参数 args 是一个包含将要提供给函数的按位置传递的参数的元组。如果省略 args，任何参数都不会被传递。参数 kwargs 是一个包含关键字参数的字典。第 5 条语句将参数 islower( )方法的运行结果返回，即判断 DataFrame 对象的“City”列数据是否为小写的。

② 对 DataFrame 对象中的某些行或列，或者对 DataFrame 对象中的所有元素进行某种运算或操作时，Python 不用编写循环语句，可以使用 DataFrame 对象提供的方法 apply( )、map( )和 applymap( )实现相同的功能。apply( )方法用于 DataFrame 对象时，是对 DataFrame 对象的行或列进行计算；map( )方法和 applymap( )方法是元素级别的操作，即对 DataFrame 对象的每个元素的操作。

③ 语句 lambda x: x.islower( )使用了 lambda 匿名表达式。lambda 表达式的一般形式是关键字 lambda 后面跟一个或多个参数，紧跟一个“:”，最后是一个表达式。lambda 表达式可以实现函数编程，即将函数作为参数传递给其他函数。它能够出现在 Python 语法不允许 def（定义函数）出现的地方。作为表达式，lambda 的功能与函数相同，均有返回值。

思考：（1）用 istitle( )函数和 isupper( )函数替代上述案例中的 islower( )函数，查看程序运行效果。

（2）将最后一条语句 frame['City'].apply(lambda x: x.islower( ))替换为语句 frame['City'].map(str.islower)，查看程序运行效果。

3.4.3 规范数据格式

Pandas 在读取数据文件时，可以规范化数据类型。如在读取 CSV 文件时，将某列规范为数值类型或字符类型，实现规范数据格式的语句如下：

```
import pandas as pd
frame = pd.read_csv('D:\data_format.csv', dtype={'Year': str})
frame['Year'] = frame['Year']+'000'
print(frame)
```

程序运行结果如下：

```
       City      Area  Population     Year
0   Beijing  16410.54     2018.60  1949000
1  Shanghai   6340.50     2347.46  1949000
2  Shenzhen  11946.88     1354.58  1979000
```

说明：

① 第 2 条语句在读取 CSV 文件的时候，通过 dtype 参数设置将“Year”列更改为字符串类型。

② 第 3 条语句将“Year”列的每个元素分别与字符串“000”相加。

思考：（1）案例是否改变了 frame 对象的“Year”列数据？为什么？

（2）文件 data_format.csv 的“Year”列数据是否改变了？为什么？

to_datetime( )方法可以解析多种不同的日期表示形式。例如：

```
import pandas as pd
frame = pd.read_csv('D:\data_date.csv')
print(frame)
frame['Year'] = pd.to_datetime(frame['Year'])
print(frame)
```

程序运行结果如下：

```
       City      Area  Population       Year
0   Beijing  16410.54     2018.60  1949-2-24
1  Shanghai   6340.50     2347.46  24-2-1949
2  Shenzhen  11946.88     1354.58  2-24-1979
       City      Area  Population       Year
0   Beijing  16410.54     2018.60 1949-02-24
1  Shanghai   6340.50     2347.46 1949-02-24
2  Shenzhen  11946.88     1354.58 1979-02-24
```

3.4.4 字符型数据判断

Python 可以对字符型数据的内容进行检查，主要函数如下。

函数 isalnum( )的功能是，如果字符串至少包含一个字符，且所有字符都是字母或数字，则返回 True，否则返回 False。

函数 isalpha( )的功能是，如果字符串至少包含一个字符并且所有字符都是字母，则返回 True，否则返回 False。

函数 isdigit( )的功能是，如果字符串只包含数字（如 Unicode 数字、半角数字、全角数字和罗马数字等），则返回 True，否则返回 False。

函数 isnumeric( )的功能是，如果字符串只包含数字字符（如 Unicode 数字、全角数字、罗马数字和汉字数字等），则返回 True，否则返回 False。

函数 isspace( )的功能是，如果字符串只包含空格，则返回 True，否则返回 False。实现字符型数据判断的语句如下：

```
import pandas as pd
frame = pd.read_csv('D:\data.csv')
print(frame['City'].apply(lambda x: x.isalpha()))
```

程序运行结果如下：

```
0    True
1    True
2    True
Name: City, dtype: bool
```

思考：用其他函数替代上述案例中 isalpha( )函数，查看程序运行效果。

3.5 清洗逻辑错误

逻辑错误就是不符合逻辑的数据问题，如重复记录、异常值和极端值（如工资高的不符合常理，学生成绩出现过多的零分）等。我们应尽早发现逻辑错误并清洗，以防止数据分析结果走偏。

3.5.1 删除重复记录

数据表中经常会出现重复的数据，可以使用 DataFrame 对象的 duplicated( )方法和 drop_duplicates( )方法查找并删除重复的行数据。方法 duplicated( )用来查找数据表中的重复行数据，返回一个布尔型的 Series，只有当两条记录完全一样时才会判断为重复值，返回值是 True，否则返回值是 False。方法 drop_duplicates( )用来删除数据表中的重复行数据。例如：

```
import pandas as pd
frame = pd.DataFrame({"城市":["北京", "上海"]*2 + ["深圳"], "面积":[16410.54, 6340.5, \
    16410.54, 6340.5, 11946.88]})
print(frame)
frame.duplicated()
print(frame.duplicated())
frame.drop_duplicates()
```

程序运行结果如下：

```
   城市       面积
0  北京  16410.54
1  上海   6340.50
2  北京  16410.54
3  上海   6340.50
```

```
4  深圳  11946.88
0    False
1    False
2     True
3     True
4    False
dtype: bool
```

	城市	面积
0	北京	16410.54
1	上海	6340.50
4	深圳	11946.88

说明：

① 输出结果的第一部分是 DataFrame 对象的数据，共 5 行 2 列。

② 输出结果的第二部分是判断 5 行数据是否有重复，结果为 True 的行是重复行。本例中的行索引为 2 和 3 的两行是重复行。

③ 输出结果的第三部分是删除重复记录行后的数据，共 3 行 2 列。默认情况下，保留第一个非重复的记录行，本例中保留行索引为 0 和 1 的两行，删除了行索引为 2 和 3 的两行重复行。默认情况下，duplicated( )方法和 drop_duplicates( )方法判断记录的全部列值是否重复，也可以增加参数指定某列或某行判断是否重复，语句是 frame.drop_duplicates(['列名'])。

思考：尝试使用 drop_duplicates(['列名'])方法修改代码，功能是判断出“列名”重复的记录然后删除，如“面积”列重复即删除整行。

3.5.2 替换不合理值

不合理数据主要是指异常值和极端值。统计学概念中，异常值是指一组测定值中与平均值的偏差超过 2 倍标准差的测定值 [6]。在数据挖掘领域，异常值又称为离群点，显著不同于其他数据，好像是它被不同机制产生的一样 [7]。极端值是指一个函数的极大值或极小值。由于异常值和极端值与正常值差异较大，在数据分析时容易产生较大的误差，影响数据的有效性和稳健性。由于数据的不同，对异常值和极端值的认定也有区别，如医学上对异常值和极端值的认定和精确度要求往往高于其他学科。

方法 replace(old, new[, max])的功能是把 old 值替换成 new 值，如果指定第 3 个参数 max，则替换不超过 max 次；若不指定第 3 个参数 max，则用 new 替换全部 old。例如：

```
import pandas as pd
frame = pd.read_csv('D:\data_date.csv')
print(frame)
print(frame.replace(16410.54, frame['Area'].mean()))
```

程序运行结果如下：

[6] 郑家亨．统计大辞典．北京：中国统计出版社，1995.
[7] 郑家炜．数据挖掘概念与技术．北京：机械工业出版社，2012.

```
       City      Area  Population       Year
0   Beijing  16410.54     2018.60  1949-2-24
1  Shanghai   6340.50     2347.46  24-2-1949
2  Shenzhen  11946.88     1354.58  2-24-1979
       City          Area  Population       Year
0   Beijing  11565.973333     2018.60  1949-2-24
1  Shanghai   6340.500000     2347.46  24-2-1949
2  Shenzhen  11946.880000     1354.58  2-24-1979
```

说明：最后一行代码用 frame['Area'].mean( )的值替换了 16410.54，详见输出结果。

3.6 删除非需求数据

非需求数据是指已经明确的不做数据分析和后期可视化的行数据或列数据。为了保证分析速度，我们可以对非需求数据先进行备份，然后删除所有非需求数据。

3.6.1 删除非需求行

3.3.2 节中使用了删除含缺失值的所有行记录的 dropna(axis=0)方法，删除含缺失值的所有列的 dropna(axis=1)方法。删除某一行的格式是 drop(i)，参数 i 是行索引值。例如：

```
import pandas as pd
frame = pd.read_csv('D:\data_date.csv')
print(frame)
frame = frame.drop(1)
print(frame)
```

程序运行结果如下：

```
       City      Area  Population       Year
0   Beijing  16410.54     2018.60  1949-2-24
1  Shanghai   6340.50     2347.46  24-2-1949
2  Shenzhen  11946.88     1354.58  2-24-1979
       City      Area  Population       Year
0   Beijing  16410.54     2018.60  1949-2-24
2  Shenzhen  11946.88     1354.58  2-24-1979
```

说明：

① 输出结果的第一部分是 DataFrame 对象的数据，共 3 行。

② 语句 frame.drop(1)执行后，删除了行索引是“1”的行，输出结果只有 2 行。

思考：若语句 frame.drop(1)改为语句 frame.drop(3)，因为不存在行索引值是“3”的记录，程序是否会出错？为什么？

3.6.2 删除非需求列

删除列的格式如下：

```
del 对象名['列名']
```