第 5 章

Python 数据分析与可视化

大数据、数据挖掘、机器学习等前沿技术早已应用在我们生活中的各个领域，例如智慧出行，通过大数据获取最佳的出行线路。量化交易，通过数据挖掘及数据分析来获得投资的建议。无人驾驶，通过机器学习实现自动避障、最优线路的选择等。这些前沿领域的应用能够正常地运行都离不开大量的数据支持，通过对大量的数据进行分析，可以让我们更加精准、轻松地把握趋势及预测未来。

但是在获取时这些大量的数据都是非统一格式的、非统一类型的甚至是非连续性的数据，想要直接将这些数据投入生产环境进行使用无疑是徒劳的。要想使这些数据能够被有效地利用，则在使用前需要对这些数据进行加工。这些数据往往需要经过迁移、压缩、清洗、打散、分片、分块及其他各种转换处理，使对这些数据进行进一步利用提供良好的环境。

数据分析与统计学密切相关，想要成为一名优秀的数据分析师，掌握统计学的知识是必不可少的。在统计学中数据分析按类型可以分为探索性数据分析、定性型数据分析、离线数据分析、在线数据分析等，探索性数据分析是指为了形成值得假设的检验而对数据进行分析的一种方法。定性数据分析是指对目标对象观察结果之类的非数值型数据的分析。离线数据分析用于较复杂和比较耗时的数据分析及处理。通过云平台可以实现大量的数据运算。在线数据分析用于处理在线请求，对响应的时间要求比较高。与离线数据分析相比，在线数据分析能够实时处理用户的请求。

数据分析中 5 个常用的统计学概念分别为特征统计、概率分布、降维、采样及贝叶斯统计。特征统计是比较常用的统计学概念，可能大部分人在生活中会经常碰到，其应用场景例如统计地区人均收入、中位收入等。特征统计包括偏差、方差、平均值、中位数、百分比等。概率分布表示为所有可能值出现的概率的函数，常见的概率分布有均匀分布、正态分布、泊松分布等。降维顾名思义将高维度的数据转换成为低维度的数据，通过降低数据的维度以提高计算量。采样即从总体中抽取个体或者样品的过程，通过采样对总体进行评测。贝叶斯的统计思想为一种归纳推理的理论，后被一些统计学发展成为一种系统的统计推断方法，用于描述两个条件概率之间的关系。

除了统计学以外，良好的数据分析工具也能为数据分析提高效率。数据分析工具的使用也没有统一的标准，在不同的数据量的情况下，使用的技术手段是不尽相同的，例如要对班级学生每个阶段的测试结果进行统计并预测，则仅仅使用 Excel 就可以实现，而想要实现无人驾驶，则需要对海量的数据进行分析，此时就不能使用 Excel 来完成了，所以对于数据分析所使用的技术栈，也取决于需要分析的数据量的大小。

将分析出的结果通过图表的方式展现出来可以减少人们对数据的理解时间。一个专业的图表，会让人更容易理解和发现数据中有价值的信息，这也是数据可视化的根本价值所在。Python 对数据可视化方面也有着非常成熟的模块，可以快速地构建出非常强大美观的可视化界面。

5.1　NumPy

14min

5.1.1　NumPy 简介及安装

NumPy 是使用 Python 进行科学计算的基础软件包，它包括功能强大的 N 维数组对象、精密广播功能函数、集成 C/C++和 FORTRAN 代码的工具及强大的线性代数、傅里叶变换和随机数功能等。

NumPy 包的核心是 ndarray 对象。它封装了 Python 原生的、同数据类型的 n 维数组，为了保证其性能优良，其中很多操作是将代码在本地进行编译后执行的。

NumPy 数组在创建时有固定的大小，更改 ndarray 的大小将会创建一个新的数组并且删除原来的数组。NumPy 数组中的元素具有相同的数据类型。NumPy 数组有助于对大量数据进行高级数学和其他类型的操作。

NumPy 的官方网站的网址为 https://numpy.org/，如图 5-1 所示。

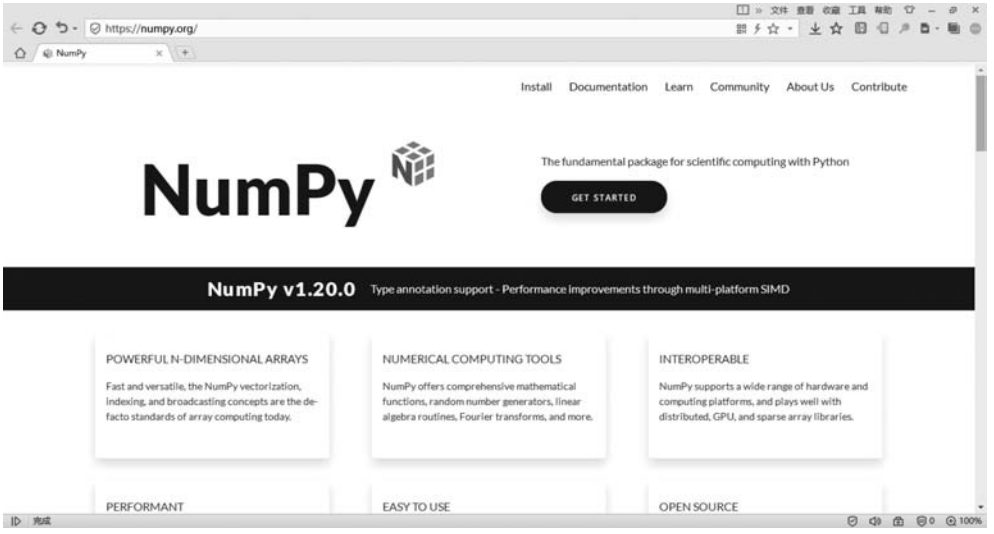

图 5-1　NumPy 官方网站

可以使用 pip 安装 NumPy 模块，打开 PyCharm→终端，在终端输入的命令如下：

```
pip install numpy
```

出现 Successfully 后即表示安装成功。使用如下代码测试是否工作正常，代码如下：

```
import numpy
help(numpy)
```

上面代码会打印出 NumPy 的文档，如果正确显示了文档，则表示安装正常，可以进一步使用。

5.1.2 NumPy 数组属性

NumPy 的核心是 ndarray 对象，它封装了 Python 原生的、同数据类型的 n 维数组，数组中的元素的类型通常情况下为数字类型。在 NumPy 数组中没有负索引，只有正索引。在 NumPy 维度中称为轴。例如对于一个空间坐标[1,2,3]具有一个轴，该轴有 3 个元素，所以我们说它的长度为 3，如图 5-2 所示。

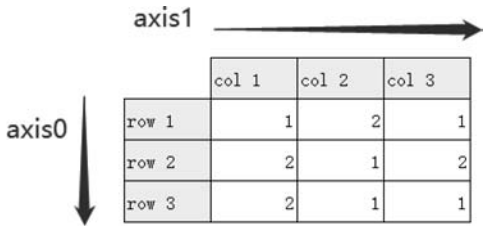

图 5-2 NumPy 轴的概念

ndarray 的常用属性为 ndarray. ndim、ndarray. shape、ndarray. size、ndarray. dtype、ndarray. itemsize、ndarray. data，其中 ndarray. ndim 为数组的轴(维度)的个数，在 Python 中，维度的数量被称为 rank。ndarray. shape 为数组的维度。返回值为整数类型的元组，表示每个维度中数组的大小。对于有 n 行和 m 列的矩阵，shape 返回值为 (n,m)，因此 shape 元组的长度就是 rank 或维度的个数 ndim。ndarray. size 用于返回数组元素的总数。ndarray. dtype 用于描述数组中元素的类型。ndarray. itemsize 用于返回数组中元素的大小。ndarray. data 表示该缓冲区包含数组的实际元素。通常情况下使用索引访问元素，而不是通过此属性。ndarray 示例代码如下：

```
#第 5 章//nmp.py
import numpy
#创建一个数组
arr = numpy.array([(1,2,1),(2,2,1),(2,1,1)])
#返回轴的个数
print(arr.ndim)
#返回组的维度
print(arr.shape)
#返回元素的总数
print(arr.size)
#返回数组中元素的类型
print(arr.dtype)
#返回数组中元素的大小
print(arr.itemsize)
#访问数组元素
```

```
print(arr[0][1])
#返回数组类型
print(type(arr))

#输出结果为
2
(3, 3)
9
int32
4
2
<class 'numpy.ndarray'>
```

5.1.3　NumPy 创建数组

创建数组有多种方法，例如可以通过 array 函数显式地创建指定的数组，示例代码如下：

```
import numpy
#创建一个数组
arr = numpy.array([(1,2,1),(2,2,1),(2,1,1)])
print(arr)

#输出结果为
[[1 2 1]
[2 2 1]
[2 1 1]]
```

上面创建了一个多维数组，如想要创建一个一维数组，其示例代码如下：

```
import numpy
#创建一个数组
arr = numpy.array([1,2,1])
print(arr)

#输出结果为[1 2 1]
```

如果想要在创建时指定数组的类型，则可以使用 dtype 参数进行指定，示例代码如下：

```
import numpy
#创建一个数组
arr = numpy.array([1,2,1],dtype = float)
print(arr)

#输出结果为[1. 2. 1.]
```

大多数情况下在创建数组时知道数组的大小，但是并不知道要放置在数组中的内容究竟是什么，因此 NumPy 提供了多种函数用于创建具有初始占位符的数组，这就减少了数组

的资源开销。能够创建具有初始占位符的函数有 zeros()、ones()、empty()等。

zeros()函数用于创建指定大小的数组，数组元素以 0 来填充，该函数包含 3 个主要参数，分别为 shape、dtype、order，shape 代表要创建的数组的形状，也就是行与列。dtype 为可选参数，表示创建的数组的数据类型，默认为浮点数。order 为'C'用于 C 的行数组，或者'F'用于 FORTRAN 的列数组。其示例代码如下：

```
import numpy
#创建一个 1 行 0 列的数组，元素长度为 5
arr = numpy.zeros(5)
print(arr)
#创建一个 1 行 0 列的数组，元素长度为 5，类型为 int
arr1 = numpy.zeros((5,), dtype = int)
print(arr1)

#输出结果为
[0. 0. 0. 0. 0.]
[0 0 0 0 0]
```

ones()函数用于创建指定大小的数组，数组元素以 1 来填充，该函数包含了 3 个主要参数，分别为 shape、dtype、order，shape 代表要创建的数组的形状，也就是行与列。dtype 为可选参数，表示创建的数组的数据类型，默认为浮点数。order 为'C'用于 C 的行数组，或者'F'用于 FORTRAN 的列数组。其示例代码如下：

```
import numpy
#创建一个 1 行 0 列的数组，元素长度为 5
arr = numpy.ones(5)
print(arr)
#创建一个 1 行 0 列的数组，元素长度为 5，类型为 int
arr1 = numpy.ones((5,), dtype = int)
print(arr1)

#输出结果为
[1. 1. 1. 1. 1.]
[1 1 1 1 1]
```

empty()函数用于创建指定大小的数组，数组元素以随机值来填充，该函数包含了 3 个主要参数，分别为 shape、dtype、order，shape 代表要创建的数组的形状，也就是行与列。dtype 为可选参数，表示创建的数组的数据类型，默认为浮点数。order 为'C'用于 C 的行数组，或者'F'用于 FORTRAN 的列数组。其示例代码如下：

```
import numpy
#创建一个 1 行 0 列的数组，元素长度为 5
arr = numpy.empty(5)
print(arr)
#创建一个 1 行 0 列的数组，元素长度为 5，类型为 int
arr1 = numpy.empty((5,), dtype = int)
```

```
print(arr1)

#输出结果为
[5.e-324 4.e-323 4.e-323 4.e-323 0.e+000]
[1066 0 1 35 1]
```

arange()函数可以创建指定数值范围的数组，该函数包含 4 个主要参数，分别为 start、stop、step、dtype，start 为起始值，默认值为 0。stop 为终止值，终止的数值是不包含该值的。step 为步长，即每次的增量数，默认为 1。dtype 为数据类型，默认为 int。arange 创建数组的示例代码如下：

```
import numpy
#创建一个数组，起始为 2，终止为 10，步长为 2
arr = numpy.arange(2,11,2)
print(arr)

#输出结果为[ 2 4 6 8 10]
```

linspace()用于创建一个一维数组，数组为一个等差数列，该函数包含 6 个主要参数，分别为 start、stop、num、endpoint、retstep、dtype，start 为序列的起始值。stop 为序列的终止值，如果 endpoint 为 True，则该值包含于数列中。num 为要生成的等步长的样本数量，默认为 50。retstep 为 True 时生成的数组会显示间距，反之不显示。dtype 为数据类型。linspace()创建数组的示例代码如下：

```
import numpy
arr = numpy.linspace(2,12,11)
print(arr)

#输出结果为[ 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.]
```

logspace()用于创建一个等比数列数组，该函数包含 6 个主要参数，分别为 start、stop、num、endpoint、base、dtype，start 为序列的起始值。stop 为序列的终止值，如果 endpoint 为 True，则该值包含于数列中。num 为要生成的等步长的样本数量，默认为 50。base 为对数 log 的底数。dtype 为数据类型。logspace()创建数组的示例代码如下：

```
import numpy
arr = numpy.logspace(1,2,10)
print(arr)

#输出结果为
[ 10. 12.91549665 16.68100537 21.5443469 27.82559402
  35.93813664 46.41588834 59.94842503 77.42636827 100. ]
```

5.1.4　NumPy 切片索引及迭代

与 Python 的其他序列类型一样，NumPy 也可以进行切片或者通过索引进行访问。例

如对一个一维数组进行切片的代码如下：

```
import numpy
arr = numpy.arange(2,11,1)
print("切片前:",arr)
#从索引 1 开始切片,间隔为 2
print("切片后:",arr[1::2])

#输出结果为
切片前: [ 2 3 4 5 6 7 8 9 10]
切片后: [3 5 7 9]
```

对于多维数组的切片同样也可以使用此类方式进行，示例代码如下：

```
import numpy
arr = numpy.array([[2,3,4,5],[1,2,3,4],[5,2,7,4],[9,5,1,6]])
print("切片前:",arr)
#从索引 1 开始切片,间隔为 2
print("切片后:",arr[1::2])

#输出结果为
切片前: [[2 3 4 5]
[1 2 3 4]
[5 2 7 4]
[9 5 1 6]]
切片后: [[1 2 3 4]
[9 5 1 6]]
```

除了基本的索引方式外，NumPy 还提供了更多的索引方式，包括整数数组索引、运算符索引等。在方括号内传入多个索引值，可以同时选择多个元素，示例代码如下：

```
import numpy
arr = numpy.array([1,2,3,4,5,6,7,8])
indexs = [1,3,5]
print(arr[indexs])

#输出结果为[2 4 6]
```

多维数组同样可以使用此方式进行选择，示例代码如下：

```
import numpy
arr = numpy.array([[1,2,3],
                   [2,3,4],
                   [3,4,5],
                   [4,5,6],
                   [5,6,7]])
```

```
#表示行索引
r = [0,1,2]
#表示列索引
c = [1,2,2]
y = arr[r,c]
print(y)

#输出结果为[2 4 5]
```

NumPy 还支持运算符索引，例如选择数组中所有大于 2 的元素，示例代码如下：

```
import numpy
arr = numpy.array([[1,2,3],[2,3,4],[3,4,5],[4,5,6],[5,6,7]])
print(arr[arr > 2])

#输出结果为[3 3 4 3 4 5 4 5 6 5 6 7]
```

对数组迭代的示例代码如下：

```
import numpy
arr = numpy.array([[1,2,3],[2,3,4],[3,4,5],[4,5,6],[5,6,7]])
for i in arr:
   print(i)

#输出结果为
[1 2 3]
[2 3 4]
[3 4 5]
[4 5 6]
[5 6 7]
```

由上面的代码可见，数组的迭代是基于轴的，如果要迭代数组中的每个元素，则需要使用 flat 属性，该属性是数组的所有元素的迭代器，示例代码如下：

```
import numpy
arr = numpy.array([[1,2,3],[2,3,4],[3,4,5],[4,5,6],[5,6,7]])
for i in arr.flat:
   print(i)

#输出结果为
1
2
3
2
3
4
3
4
```

```
5
4
5
6
5
6
7
```

5.1.5 操作数组

数组中的运算符操作可执行元素级别的操作，两个数组相乘的示例代码如下：

```
import numpy
arr = numpy.array([[1,2,3],[2,3,4]])
nrr = numpy.array([[3,4,5],[4,5,6]])
newarr = arr * nrr
print(newarr)

#输出结果为
[[ 3 8 15]
 [ 8 15 24]]
```

矩阵乘积可以使用@运算符或 dot()函数或方法执行，dot()函数仅支持 Python 3.5 以上的版本，示例代码如下：

```
import numpy
arr = numpy.array([[1,2],[2,3]])
nrr = numpy.array([[3,4],[4,5]])
newarr = arr@nrr
print(newarr)

#输出结果为
[[11 14]
 [18 23]]
```

当使用不同类型的数组进行操作时，数组的类型会向上转换，即其类型属于更加精确的一方，示例代码如下：

```
import numpy
arr = numpy.array([[1,2],[2,3]])
nrr = numpy.array([[3.,4],[4.,5]])
newarr = arr + nrr
print(newarr)

#输出结果为
[[4. 6.]
 [6. 8.]]
```

NumPy 提供了大量的数学计算函数，例如 sin()、cos()、tan()等，三角函数运算的示例代码如下：

```
import numpy
arr = numpy.array([30,90])
#获取 30°及 90°正弦值
print(numpy.sin(arr * numpy.pi/180))
#获取 30°及 90°余弦值
print(numpy.cos(arr * numpy.pi/180))
#获取 30°及 90°正切值
print(numpy.tan(arr * numpy.pi/180))

#输出结果为
[0.5 1. ]
[8.66025404e-01 6.12323400e-17]
[5.77350269e-01 1.63312394e+16]
```

舍入函数包括 around()、floor()、ceil()等，around()函数的示例代码如下：

```
#第 5 章//nmp2.py
import numpy
newarr = numpy.array([12.223,402.255,31.164])
#不保留小数
print(numpy.around(newarr))
#保留 1 位小数
print(numpy.around(newarr,decimals = 1))
#decimals 为负,将四舍五入到小数点左侧
print(numpy.around(newarr,decimals = -1))
#decimals 为负,将四舍五入到小数点左侧
print(numpy.around(newarr,decimals = -2))

#输出结果为
[ 12. 402. 31.]
[ 12.2 402.3 31.2]
[ 10. 400. 30.]
[ 0. 400. 0.]
```

floor()函数，向下取整，即返回指定表达式的最大整数，示例代码如下：

```
import numpy
newarr = numpy.array([12.4,1.6,3.5])
print(numpy.floor(newarr))

#输出结果为
[12. 1. 3.]
```

ceil()函数向上取整，返回指定表达式的最小整数，示例代码如下：

```
import numpy
newarr = numpy.array([12.4,1.6,3.5])
print(numpy.ceil(newarr))

#输出结果为
[13. 2. 4.]
```

算数函数包括 add()、subtract()、multiply()、divide()等，示例代码如下：

```
import numpy
arr1 = numpy.array([12.4,1.6,3.5])
arr2 = numpy.array([4,2,3])
#相加
print(numpy.add(arr1,arr2))
#相减
print(numpy.subtract(arr1,arr2))
#相乘
print(numpy.multiply(arr1,arr2))
#相除
print(numpy.divide(arr1,arr2))

#输出结果为
[16.4  3.6  6.5]
[ 8.4  -0.4  0.5]
[49.6  3.2  10.5]
[3.1      0.8      1.16666667]
```

统计函数 amin()用于计算指定轴的最小值、amax()用于计算指定轴的最大值、ptp()用于计算元素最大值与最小值的差、percentile()表示小于指定值的占比、median()用于计算中位数、mean()用于计算算数平均值、average()用于计算加权平均值等，示例代码如下：

```
#第 5 章//nmp3.py
import numpy
arr = numpy.array([[9,2,3],[2,3,4],[3,4,5],[4,5,6]])
#所有数组中最小的值
print(numpy.amin(arr))
#沿纵轴判断最小的值
print(numpy.amin(arr,1))
#沿横轴判断最小的值
print(numpy.amin(arr,0))
#所有数组中最大的值
print(numpy.amax(arr))
#沿横轴判断最大的值
print(numpy.amax(arr,0))
#所有数组中最大与最小的差值
print(numpy.ptp(arr))
#沿纵轴判断差值
```

```
print(numpy.ptp(arr,1))
#50% 的分位数,也就是 a 中排序后的中位数
print(numpy.percentile(arr,50))
#沿横轴
print(numpy.percentile(arr,50,0))
#取中位数
print(numpy.median(arr))
#沿横轴取中位数
print(numpy.median(arr,0))
#返回算数平均值
print(numpy.mean(arr))
#返回加权平均值
print(numpy.average(arr))

#输出结果为
2
[2 2 3 4]
[2 2 3]
9
[9 5 6]
7
[7 2 2 2]
4.0
[3.5 3.5 4.5]
4.0
[3.5 3.5 4.5]
4.166666666666667
4.166666666666667
```

5.1.6　NumPyIO

NumPy 可以将文件存储在磁盘上,与 Python 文件类似,NumPy 的文件格式为 npy,npy 文件用于存储 NumPy 所需的数据、图形、dtype 和其他信息。存储的数据可以使用 load()与 save()函数对文件进行读取和存储,数组是以二进制格式保存在扩展名为.npy 的文件中。savez()函数用于将多个数组写入文件,以二进制格式保存在扩展名为.npz 的文件中。loadtxt()与 savetxt()函数用于处理正常的文本文件。

save()函数将数组保存至扩展名为.npy 的文件中,save()函数包含 4 个主要参数,分别为 file 要保存的文件名,如果没有加扩展名,则自动加上.npy; arr 为要保存的数组; allow_pickle 为可选参数,表示是否允许使用 Python pickles 保存数组; fix_imports 为可选参数,为了向下兼容。save()函数的示例代码如下:

```
import numpy
arr = numpy.array([[9,2,3],[2,3,4],[3,4,5],[4,5,6]])
numpy.save('newfile',arr)
```

上面的代码执行后,会在当前目录下生成一个新的文件,名为 newfile.npy,由于是以二

进制方式存储的，所以直接打开此文件会显示乱码，如图 5-3 所示。

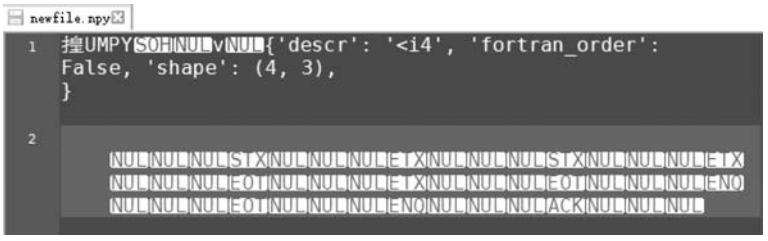

图 5-3　NumPy 保存后生成的扩展名为.npy 的文件

load()函数将从.npy 文件中读取数组，load()函数包含 5 个主要的参数，参数 file 表示要读取的文件名。参数 mmap_mode 为读取的模式，例如 r＋、r、w＋、c 等，一般无须指定。参数 allow_pickle 表示允许使用 pickle 反序列化。参数 fix_imports 表示向下兼容。参数 encoding 表示编码，默认为 ASCLL。load()函数的示例代码如下：

```
import numpy
arr = numpy.array([[9,2,3],[2,3,4],[3,4,5],[4,5,6]])
newarr = numpy.load('newfile.npy')
print(newarr)

#输出结果为
[[9 2 3]
 [2 3 4]
 [3 4 5]
 [4 5 6]]
```

savez()函数包含 3 个主要参数，参数 file 表示要保存的文件。参数 args 为要保存的数组。参数 kwds 为要保存的数组所使用的关键字名称。示例代码如下：

```
#第 5 章//nmp4.py
import numpy
arr = numpy.array([[9,2,3],[2,3,4],[3,4,5],[4,5,6]])
arr1 = numpy.array([[1.,2.,3.],[2.,3.,4.],[2.,1.,6.]])
arr2 = numpy.array([1,2,3])
numpy.savez("arrs.npz",arr,arr1,key1 = arr2)

reads = numpy.load("arrs.npz")
print(reads.files)
print(reads["arr_0"])
print(reads["arr_1"])
print(reads["key1"])

#输出结果为
['key1', 'arr_0', 'arr_1']
[[9 2 3]
 [2 3 4]
 [3 4 5]
```

```
[4 5 6]]

[[1. 2. 3.]
 [2. 3. 4.]
 [2. 1. 6.]]

[1 2 3]
```

savetxt()将数组以字符串的形式保存在文本内。savetxt()包含了 9 个主要的参数，参数 fname 为要保存的文件名。参数 X 为要存储的数组。参数 fmt 为存储的数据格式。参数 delimiter 为数据列之间的分隔符。参数 newline 为数据行之间的分隔符。参数 header 为文件头部写入的字符串。参数 footer 为文件底部写入的字符串。参数 comments 为文件头部或者尾部字符串的开头字符，默认为#。参数 encoding 为编码。savetxt()函数的示例代码如下：

```
import numpy
arr = numpy.array([[9,2,3],[2,3,4],[3,4,5],[4,5,6]])
numpy.savetxt("arrs.npz",arr)

reads = numpy.loadtxt("arrs.npz")
print(reads)
```

5.2　Pandas

5.2.1　Pandas 简介及安装

Pandas 是开放源代码的数据处理及分析利器，它提供了快速、灵活、明确的数据结构，旨在简单、直观地处理关系型、标记型数据。Pandas 适用于处理含异构列的表格数据、有序和无序(非固定频率)的时间序列数据、带行列标签的矩阵数据、包括同构或异构型数据及任意其他形式的观测统计数据集。

Pandas 的主要数据结构有 Series (一维数据)与 DataFrame(二维数据)，这两种数据结构满足了大多数应用场景的需求。Pandas 是基于 NumPy 开发的，与 Pandas 的主要区别为 NumPy 的重点是在数值计算方面，NumPy 包含了大量的数值计算工具，可以很方便地对数值进行运算，而 Pandas 则主要用于数据处理，以及对数据进行分析。Pandas 提供了大量的库和一些标准的数据模型，极大地提高了操作大型数据的效率，并且 Pandas 提供了大量便捷地处理数据的函数和方法，使 Pandas 在数据处理方面极其强大。NumPy 与 Pandas 两者在数据领域的主要应用方向不同。

Pandas 的主要特点是拥有快速高效的 DataFrame 对象，并且具有默认和自定义的索引。能够轻松处理浮点数据中的丢失数据(以 NaN 表示)及非浮点数据。强大灵活的分组功能可方便地对数据进行拆分、组合及转换。可以轻松地将其他 Python 和 NumPy 数据结构中的不同索引的数据转换为 DataFrame 对象等。

Pandas 已广泛地应用在金融、统计、社会科学及工程领域。

Pandas 的官方网站网址为 https://pandas.pydata.org/，如图 5-4 所示。

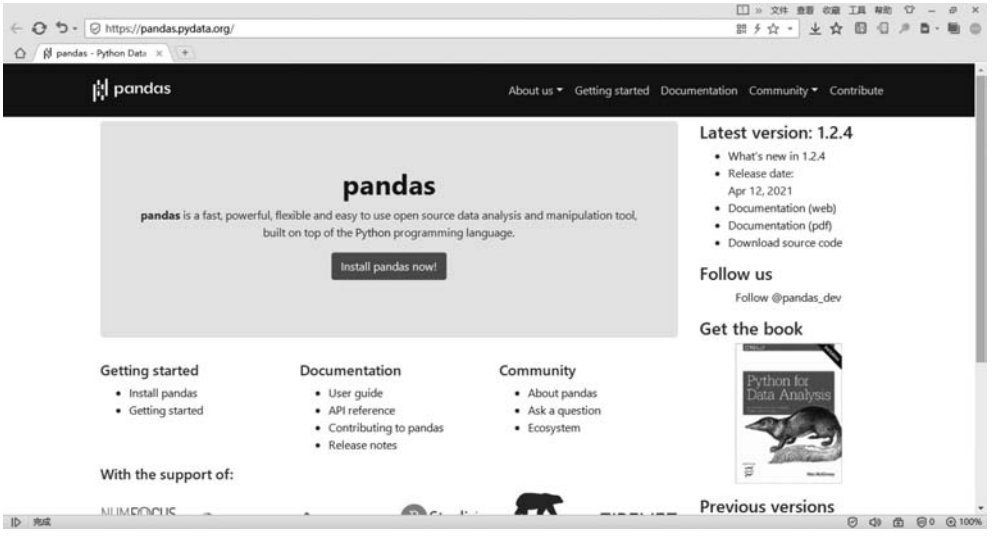

图 5-4　Pandas 官方网站

可以使用 pip 安装 Pandas 模块，打开 PyCharm→终端，在终端输入的命令如下：

```
pip install Pandas
```

出现 Successfully 后表示安装成功。使用如下代码测试是否可工作正常，代码如下：

```
import pandas
help(pandas)
```

上面的代码会打印出 Pandas 的文档，如果正确显示了文档，则表示安装正常，可以进一步使用。

5.2.2　Series

Series 是带标签的一维数组，它可以存储整数型、字符串等各种 NumPy 可存储的数据，其轴标签称为索引。使用 Series()函数即可创建 Series，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6])
print(pd)

#输出结果为
0    1
1    2
2    3
3    4
```

```
4    5
5    6
dtype: int64
```

Series 的表现形式为左边列为索引，右边列为值。由于在上文的代码中没有为数据指定索引，所以系统会自动地创建一个从 0 开始的索引。可以通过 values 属性与 index 属性获取其值与索引，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6])
print(pd.values)
print(pd.index)

#输出结果为
[1 2 3 4 5 6]
RangeIndex(start = 0, stop = 6, step = 1)
```

创建一个带索引标签的 Series，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6],index = ["a","b","c","d","e","f"])
print(pd)

a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

Series 可通过索引进行访问，支持切片，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6],index = ["a","b","c","d","e","f"])
#通过索引访问单个元素
print(pd["c"])
#通过索引访问多个元素
print(pd[["c","b","f"]])
#访问
print(pd[4:])

#输出结果为
3
c    3
b    2
f    6
```

```
dtype: int64
e    5
f    6
dtype: int64
```

Series 支持 NumPy 的大多数函数，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6], index = ["a","b","c","d","e","f"])
#取大于中位数的值
print(pd[pd > pd.median()])
#取大于平均数的值
print(pd[pd > pd.mean()])

#输出结果为
d    4
e    5
f    6
dtype: int64
d    4
e    5
f    6
dtype: int64
```

由于 Series 带标签索引的特点，所以它与字典类型十分相似，可以将其看作一个定长的字典，用于许多原本需要字典参与的函数中，示例代码如下：

```
import pandas
pd = pandas.Series([1,2,3,4,5,6], index = ["a","b","c","d","e","f"])
bol = "f" in pd
print(bol)

#输出结果为 True
```

使用字典来创建 Series，示例代码如下：

```
import pandas
dic = {"a":1,"b":2,"c":3,"d":4}
pd = pandas.Series(dic)
print(pd)

#输出结果为
a    1
b    2
c    3
d    4
dtype: int64
```

通过传入不同顺序的索引来为 Series 进行排序，示例代码如下：

```
import pandas
dic = {"a":1,"b":2,"c":3,"d":4}
indexs = ["c","b","d","f"]
pd = pandas.Series(dic,index = indexs)
print(pd)

#输出结果为
c     3.0
b     2.0
d     4.0
f     NaN
dtype: float64
```

在上文的代码中，由于索引 f 没有对应的值，所以会使用 NaN 来表示缺失的数据，而索引 a 不在 indexs 中，所以 a 被从结果中除去。这是 Series 的一个重要的特性，该特性可以用作数据。使其与其他应用更好地结合，例如数据库的多表查询。

可以使用 isnull()函数来检测当前 Series 中是否含缺失数据，示例代码如下：

```
import pandas
dic = {"a":1,"b":2,"c":3,"d":4}
indexs = ["c","b","d","f"]
pd = pandas.Series(dic,index = indexs)
print(pd.isnull())
print(pandas.isnull(pd))

#输出结果为
c     False
b     False
d     False
f     True
dtype: bool
c     False
b     False
d     False
f     True
dtype: bool
```

Series 与索引支持 name 属性，通常情况下 Series 自动分配 name，示例代码如下：

```
import pandas
dic = {"a":1,"b":2,"c":3,"d":4}
indexs = ["c","b","d","f"]
pd = pandas.Series(dic,index = indexs)
pd.name = "newseries"
pd.index.name = "newindexseries"
print(pd)

#输出结果为
```

```
newindexseries
c    3.0
b    2.0
d    4.0
f    NaN
Name: newseries, dtype: float64
```

5.2.3 DataFrame

DataFrame是一个由多种类型的列构成的二维标签数据结构，其数据结构类似于Excel表格。DataFrame是最常用的Pandas对象，与Series一样，DataFrame支持多种类型的数据。

使用DataFrame()函数创建DataFrame，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic)
print(df)

#输出结果为
   user  pass  age
0  jack  1234  23
1  marry 4567  24
2  tom   7890  25
```

在上面的代码中，创建了一个DataFrame，可以看到其输出结果类似一张SQL表格，此表格的字段名分别为user、pass、age，index为自增索引，如图5-5所示。

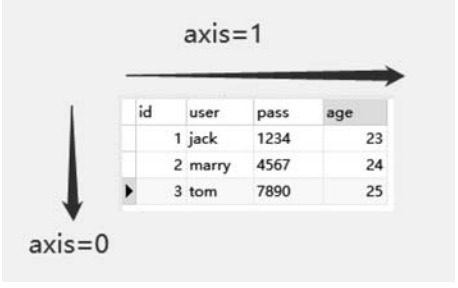

图5-5 DataFrame表格

DataFrame()函数是DataFrame类的构造函数，其包含data、index、columns、dtype、copy等参数，参数data为要创建的DataFrame数据。参数index为DataFrame的行标签。参数columns为DataFrame的列标签。参数dtype为每列的类型。参数copy为从input中复制数据，默认值为False。index、columns用于指定行列标签的示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic)
print(df.index)
print(df.columns)

#输出结果为
RangeIndex(start = 0, stop = 3, step = 1)
Index(['user', 'pass', 'age'], dtype = 'object')
```

DataFrame()函数的 columns 参数可用于对标签进行排序，示例代码如下：

```
#第 5 章//pds.py
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic)
print(df)
newdf = pandas.DataFrame(df,columns = ["age","user","pass"])
print(newdf)

#输出结果为
   user  pass  age
0  jack  1234  23
1  marry 4567  24
2  tom   7890  25

   age  user  pass
0  23   jack  1234
1  24   marry 4567
2  25   tom   7890
```

如果传入的 columns 参数在列中找不到，则会使用缺失数据补全，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic,columns = ["user","pass","age","sex"])
print(df)
```

```
#输出结果为
    user  pass  age  sex
0   jack  1234  23   NaN
1   marry 4567  24   NaN
2   tom   7890  25   NaN
```

head(n)方法用于获取 DataFrame 前 n 行数据，n 的默认值为 5，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom","user1","user2","user3","user4"],
    "pass":["1234","4567","7890","1234","1234","1234","1234"],
    "age":[23,24,25,26,22,21,27]
}
df = pandas.DataFrame(dic)
print(df.head(3))                                    #输出前 3 行数据

#输出结果为
    user  pass  age
0   jack  1234  23
1   marry 4567  24
2   tom   7890  25
```

通过列标记可以很容易地获取一个列 Series，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic)
print(df.user)
print(df["user"])

#输出结果为
0     jack
1     marry
2     tom
Name: user, dtype: object
0     jack
1     marry
2     tom
Name: user, dtype: object
```

使用 loc 与 iloc 属性获取一个行 Series，loc 与 iloc 支持多列访问，且支持切片及对象降维，loc 与 iloc 的区别为 iloc 通过下标进行查找，而 loc 通过 index 标识进行查找。示例代码如下：

```
#第 5 章//pds.py
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic,index = ["one","two","three"])
print(df.iloc[2])
print(df.iloc[1:2,0:2])
print(df.loc["three"])
print(df.loc["one":"two",["user","pass"]])

#输出结果为
user    tom
pass    7890
age    25
Name: three, dtype: object
      user  pass
two  marry  4567
user    tom
pass    7890
age    25
Name: three, dtype: object
      user  pass
one  jack  1234
two  marry  4567
```

可以通过赋值的方式对列进行修改，示例代码如下：

```
#第 5 章//pds.py
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic,index = ["one","two","three"],columns = ["user","pass","age",
"other"])
print(df)
df["other"] = "data"                          #赋值给 other 列
print(df)

#输出结果为
      user  pass  age  other
one    jack  1234  23  NaN
two  marry  4567  24  NaN
three  tom  7890  25  NaN
```

```
      user  pass  age  other
one   jack  1234  23   data
two   marry 4567  24   data
three  tom  7890  25   data
```

也可以使用列表为 DataFrame 列赋值，如果使用列表进行赋值，则要保证列表的长度与 DataFrame 列的长度一致。除列表外，也可以使用 Series 进行赋值，使用 Series 赋值，则 Series 索引与 DataFrame 索引会进行精准匹配。在使用 Series 精准匹配时，缺失的索引对应的数据将会用 NaN 代替，示例代码如下：

```
#第 5 章//pds.py
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df)
list = [1,2,3]
df["other"] = list                          #使用列表赋值
print(df)
s = pandas.Series(["one","three"], index = ["one","three"])
df["other"] = s                             #使用 Series 赋值
print(df)

#输出结果为
      user  pass  age  other
one    jack  1234   23    NaN
two   marry  4567   24    NaN
three  tom   7890   25    NaN

      user  pass  age  other
one    jack  1234   23      1
two   marry  4567   24      2
three  tom   7890   25      3

      user  pass  age  other
one    jack  1234   23    one
two   marry  4567   24    NaN
three  tom   7890   25  three
```

del 方法用于删除一列，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
```

```
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
del df["user"]                          #删除 user 列
print(df)

#输出结果为
       pass  age  other
one    1234   23    NaN
two    4567   24    NaN
three  7890   25    NaN
```

与 Series 一样，DataFrame 中可以使用 values 属性返回所有的数据，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df.values)

#输出结果为
[['jack' '1234' 23 nan]
['marry' '4567' 24 nan]
['tom' '7890' 25 nan]]
```

DataFrame 也可以使用切片的方式进行访问，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df[0:2])

#输出结果为
      user  pass  age  other
one   jack  1234   23    NaN
two  marry  4567   24    NaN
```

5.2.4 常用操作

to_numpy()方法可用于将 DataFrame 数据类型转换为 NumPy 数据类型，需要注意的是，当 DataFrame 由多种数据类型的列组成时，该操作所消耗的资源比较大。示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df.to_numpy())

#输出结果为
[['jack' '1234' 23 nan]
 ['marry' '4567' 24 nan]
 ['tom' '7890' 25 nan]]
```

describle()方法可以快速地查看数据的统计摘要，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df.describe())

#输出结果为
        age
count    3.0
mean    24.0
std      1.0
min     23.0
25%     23.5
50%     24.0
75%     24.5
max     25.0
```

行与列转换，可以使用装饰器 T 来完成，实际执行的是 transpose()方法，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df.T)

#输出结果为
        one    two    three
user    jack   marry  tom
pass    1234   4567   7890
age     23     24     25
other   NaN    NaN    NaN
```

sort_values()函数可以按值进行排序，sort_index()函数可以按轴进行排序，示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
print(df.sort_index(axis = 1))                          #按纵轴排序,0为横轴
print(df.sort_values(by = "pass"))                      #按值进行排序

#输出结果为
       age  other  pass  user
one    23   NaN    1234  jack
two    24   NaN    4567  marry
three  25   NaN    7890  tom
       user  pass  age  other
one    jack  1234  23   NaN
two    marry 4567  24   NaN
three  tom   7890  25   NaN
```

选择满足条件的值，需要注意的是，使用条件筛选整个 DataFrame 数据，需要满足数据类型与判断目标数据类型一致，否则会报错。示例代码如下：

```
import pandas
dic = {
    "pass":[1234,4567,7890],
    "age":[23,24,25]
```

```
}
df = pandas.DataFrame(dic,index = ["one","two","three"])
print(df[df.age > 24])                          #使用 age 单例进行匹配
print(df[df > 24])                              #使用全部数据进行匹配

#输出结果为
       pass  age
three  7890  25
       pass  age
one    1234  NaN
two    4567  NaN
three  7890  25.0
```

apply()方法作用于 DataFrame 中的每个行或者列。apply() 方法允许用户传入函数，传入的函数可以是 Python 内置的函数，也可以是用户自定义的函数。例如沿着纵轴进行求和，示例代码如下：

```
import pandas
dic = {
    "pass":[11,22,33],
    "age":[23,24,25]
}
#定义要传入的函数
def func(x,n):
    return x + n
df = pandas.DataFrame(dic,index = ["one","two","three"],columns = ["pass","age"])
#DataFrame 中 pass 列每个值减 3 并返回 totle 列
df["totle"] = df["pass"].apply(func,args = ( - 3,))#参数使用元组进行传入
print(df)

#输出结果为
       pass  age  totle
one    11    23   8
two    22    24   19
three  33    25   30
```

applymap()用于对 DataFrame 中每个单元格执行指定的函数操作，示例代码如下：

```
import pandas
dic = {
    "pass":[11,22,33],
    "age":[23,24,25]
}
#定义要传入的函数
def func(x):
    return " * " + str(x) + " * "
df = pandas.DataFrame(dic,index = ["one","two","three"],columns = ["pass","age"])
df2 = df.applymap(func)
```

```
print(df2)
#输出结果为
       pass  age
one     *11* *23*
two     *22* *24*
three   *33* *25*
```

map()用于对 Series 的每个数据进行操作，示例代码如下：

```
import pandas
s1 = pandas.Series([1,2,3,4,5,6])
def func(x):
   return x * 2
s2 = s1.map(func)
print(s2)

#输出结果为
0     2
1     4
2     6
3     8
4     10
5     12
dtype: int64
```

5.2.5　读写 Excel

Pandas 对 xlrd 等模块进行了封装，可以很方便地处理 Excel 文件，支持 xls 和 xlsx 等格式，需要提前安装模块，其命令为 pip install xlrd，安装 xlrd 模块可参考 4.2.3 节。

read_excel()函数用于读取 Excel 表，read_excel()函数包含 4 个常用的参数，分别为 filename、sep、header、encoding、sheet_name，参数 filename 为要打开的 Excel 表的路径。参数 sep 为分隔符。参数 header 为表头，即列名。参数 encodeing 为文档的编码格式。参数 sheet_name 为要打开的 sheet 索引，默认值为 0，即打开第 1 个 sheet。

使用 read_excel()函数前应先在当前项目目录下创建一个名为 test.xls 的 Excel 表格，包含 2 个 Sheet，表格内容可随意填写，如图 5-6 所示。

使用 read_excel()读取 test.xls 文件，默认读取 Sheet1。示例代码如下：

```
import pandas
result = pandas.read_excel("test.xls")
print(result)

#输出结果为
    name city  age sex       tel
0  jack   杭州   23   男  12222222
1  marry  武汉   22   女  13333333
2  tom    北京   23   男  14444444
```

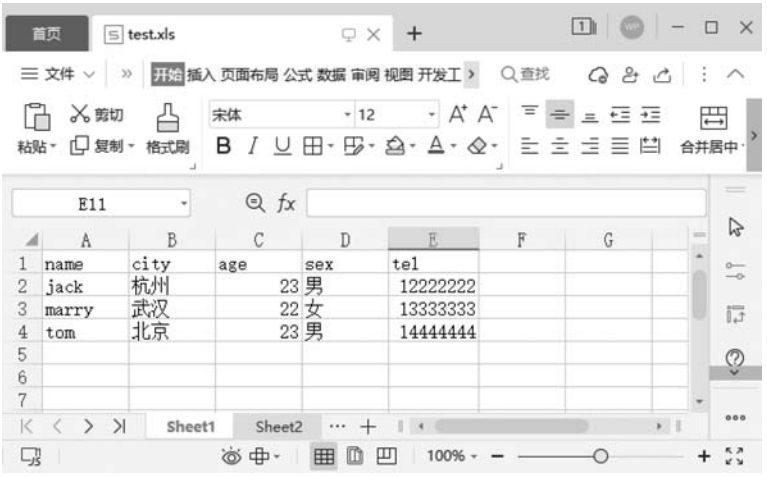

图 5-6 DataFrame 表格

读取 Sheet2 传入 sheet_name=1 参数即可，示例代码如下：

```
import pandas
result = pandas.read_excel("test.xls",sheet_name = 1)
print(result)

#输出结果为
      name  class  number
0   jack    语文    100
1   marry   数学    100
2   tom     英语    100
```

使用 to_excel()方法将 DataFrame 写入 Excel 表格中。如需存储为 xlsx 格式，则在使用 to_excel()方法前需安装 openpyxl 模块，命令如下：

```
pip install openpyxl
```

注意 使用 to_excel()方法将文件存储为 xls 格式时调用的是第三方模块 xlwt，由于模块 xlwt 不再维护，所以保存为 xls 文件时会有警告，如需保存为 xlsx 文件，则需要先安装 openpyxl 模块。

to_excel()方法的常用参数有 excel_writer、sheet_name、na_rep、encoding，参数 excel_writer 为要保存的 xlsx 文件。参数 sheet_name 为 sheet 名称。参数 na_rep 为缺失数据的表示方式。参数 encoding 为文档的编码格式。to_excel()方法的示例代码如下：

```
import pandas
dic = {
    "user":["jack","marry","tom"],
```

```
    "pass":["1234","4567","7890"],
    "age":[23,24,25]
}
df = pandas.DataFrame(dic, index = ["one","two","three"], columns = ["user","pass","age",
"other"])
df.to_excel("newtest.xlsx",sheet_name = "sheetone")
```

5.3　Matplotlib

Matplotlib 是 Python 最著名的绘图库，它提供了一整套和 MATLAB 相似的命令 API，十分适合交互式地进行制图，而且也可以方便地将它作为绘图控件，嵌入 GUI 应用程序中。Matplotlib 官方网站为 https://matplotlib.org/，如图 5-7 所示。

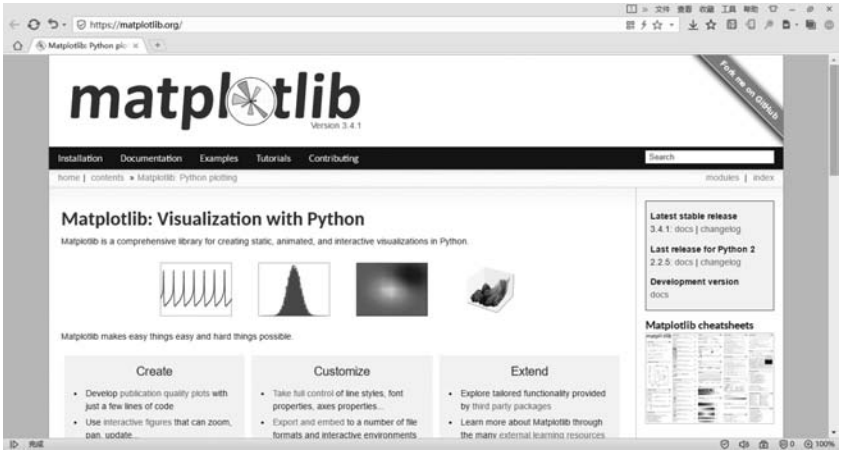

图 5-7　Matplotlib 官网

通过 pip 安装 Matplotlib，命令如下：

```
pip install matplotlib
```

出现 Successfully installed 则表示安装成功。

在使用 Matplotlib 进行绘图前，先了解一下画图的基本知识。在现实生活中画图，首先需要一张纸，也就是画布，然后才在画布上进行作画。Matplotlib 也有画布的概念，在使用 Matplotlib 画画前，使用 figure()方法创建面板，figure()方法包含 6 个常用参数，分别为 num、figsize、dpi、facecolor、edgecolor、frameon，参数 num 为图像编号或名称，如果使用数字则为编号，如果使用字符串则为名称。参数 figsize 为指定画布的宽和高。参数 dpi 为指定绘图对象的分辨率。参数 facecolor 为画布的背景颜色。参数 edgecolor 为画布的边框颜色。参数 frameon 为是否显示边框，默认不显示。

Matplotlib 中所有的图像都位于 figure 对象之中。1 张图像只能有 1 个 figure 对象。创建 figure 的示例代码如下：

```
import matplotlib.pyplot as plt
fig = plt.figure()
```

拥有了 figure 对象后，就可以在 figure 下创建一个或多个 subplot 对象(axes)，用于绘制图像，使用 add_subplot()添加一个子图，add_subplot()方法包含 2 个常用参数，分别为 * args、projection。参数 * args 使用 3 个整数或者 3 个独立的整数来描述子图的位置信息。3 个整数分别代表行数、列数和索引值，子图将分布在索引位置上。索引从 1 开始，从右上角增加到右下角。参数 projection 为可选参数，可以选择子图的类型，例如选择 polar，就是一个极点图。默认为 none，就是一个线性图。使用 add_subplot()方法绘制图像的示例代码如下：

```
#第 5 章//mbs.py
import matplotlib.pyplot as plt
fig = plt.figure("figure name")
ax = fig.add_subplot(221)
ax.set(xlim = [0.5, 4.5], ylim = [1, 10], title = 'Axis title',ylabel = 'Y - Axis', xlabel = 'X - Axis')
ax = fig.add_subplot(222)
ax.set(xlim = [0.5, 4.5], ylim = [1, 10], title = 'Axis title',ylabel = 'Y - Axis', xlabel = 'X - Axis')
plt.show()
```

上面的代码会在 figure 对象中绘制两幅子图，add_subplot(221)、add_subplot(222)即将 figure 对象分割为 2 行 2 列，并在 figure 对象的位置 1 与位置 2 处添加两幅子图，如图 5-8 所示。

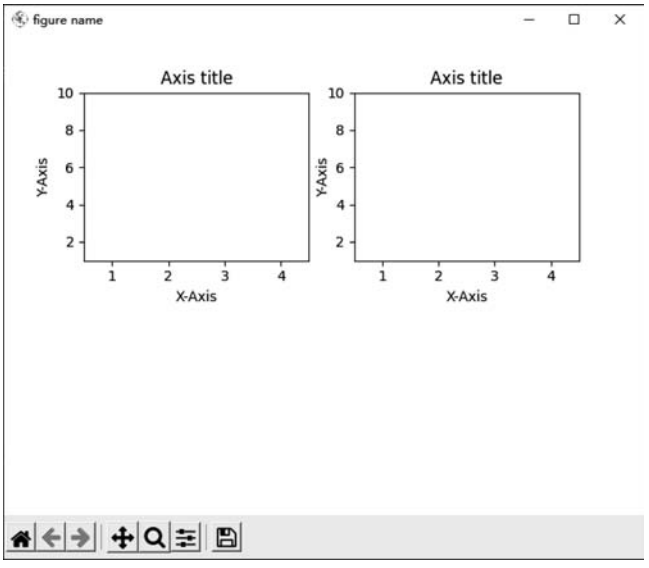

图 5-8　在 figure 对象上添加 2 个子图

了解了 Matplotlib 的绘图流程，接下来进一步了解 Matplotlib 中的常用的图形类型，包括折线图、散点图、条形图、直方图、饼图、泡泡图、等高线等。

5.3.1　折线图

折线图是数据分析中最常用的图形，折线图用于分析自变量和因变量之间的趋势关系，最适合用于显示随着时间而变化的连续数据，同时还可以看出数量的差异，以及增长情况。

Matplotlib 中使用 plot()方法绘制折线图，plot()方法常用的参数有 8 个，分别为 x、y、color、marker、linestyle、linewidth、alpha、label，其中参数 x 与参数 y 分别代表 x 轴与 y 轴对应的数据。参数 color 表示折线的颜色。参数 marker 表示这条线上数据点的类型。参数 linestyle 表示折线的类型。参数 linewidth 表示线条的粗细。参数 alpha 表示点的透明度。参数 label 表示数据图例的内容。plot()画折线图的示例代码如下：

```
#第 5 章//zx.py
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure("figure name")                    #设置画布名称
ax = fig.add_subplot(111)                          #设置子图位置
x = np.linspace(0, np.pi)
y = np.sin(x)
y2 = np.cos(x) + 6
y3 = np.cos(x) + 3
ax.plot(x,y,color = "#1cba28",linestyle = '-- ',linewidth = 2,label = 'first', marker = 'o')
                                                   #添加第一条线
ax.plot(x,y2,color = "#0598ff",linestyle = ':',linewidth = 2,label = 'second', marker = '+')
                                                   #添加第二条线
ax.plot(x,y3,color = "#ffc105",linestyle = '-',linewidth = 2,label = 'three', marker = '*')
                                                   #添加第三条线
plt.legend(loc = 2)                                #显示图例并设置图例位置,loc 可取 1、2、3、4
plt.show()
```

上面的代码画了 3 条折线，分别使用不同的颜色及样式进行作画，并添加了一张图例，显示在左上角，画出的图形如图 5-9 所示。

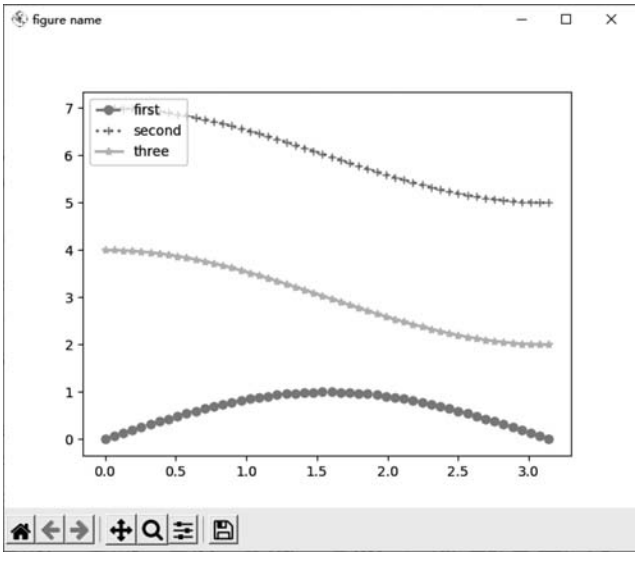

图 5-9　画三条折线并在左上角显示图例

参数 linestyle 用于控制线条的样式，参数 marker 用于控制点的样式，参数 color 用户控制线条的颜色，这 3 个参数的取值范围如表 5-1 所示。

表 5-1 参数的取值范围

linestyle 取值	说　明
—	实线(默认)
— —	双画线
:	虚线
:.	点画线
marker 取值	说　明
+	加号
○	空心圆
*	星号
.	实心圆
×	叉符号
s	正方形
d	菱形
^	上三角
v	下三角
>	右三角
<	左三角
p	五角星
h	六边形
color 取值	说　明
r	红色
g	绿色
b	蓝色
c	青绿色
m	洋红色
y	黄色
k	黑色
w	白色
#000000～#FFFFFF	可取颜色范围为#000000～#FFFFFF

legend()方法用于为图表进行标注，legend()方法常用的参数为 loc，参数 loc 用于设置图例的位置，其取值范围为 1～4，分别代表右上角、左上角、左下角及右下角。

5.3.2 散点图

使用 scatter()方法可以绘制散点图，其参数与 plot()方法的参数类似，散点图不需要 linestyle 与 linewidth 参数，如需绘制泡泡图，则需设置 s 参数及 c 参数，其中 s 参数为散点标记的大小，c 参数为散点标记的颜色。如绘制散点图，则无须设置 s 参数与 c 参数。示例代码如下：

```
#第 5 章//sd.py
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure("figure name")
ax = fig.add_subplot(111)
x = np.arange(10)
y = np.random.randn(10)
ax.scatter(x,y,color = "red", marker = 'o')
plt.show()
```

代码绘制的散点图，如图 5-10 所示。

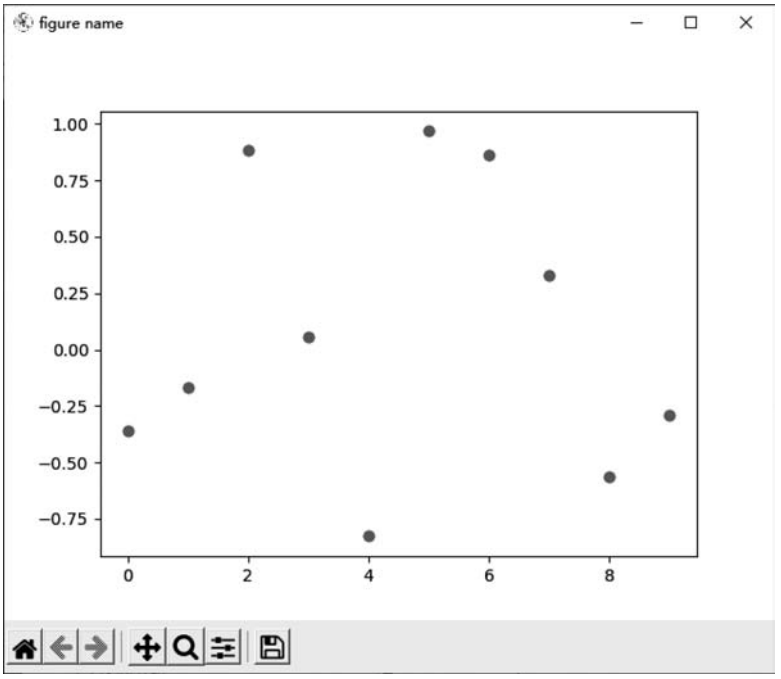

图 5-10　散点图

5.3.3　柱状图

使用 bar()方法可绘制柱状图，bar()方法包含 8 个常用参数，分别为 left、height、alpha、width、color、edgecolor、label、linewidth，其中参数 left 为 x 轴。参数 height 为柱形图的高度。参数 alpha 为柱形图颜色的透明度。参数 width 为柱形图的宽度。参数 color 为柱形图填充的颜色。参数 edgecolor 为图形边缘的颜色。参数 label 为数据图例的内容。参数 linewidth 为边缘或者线的宽度。使用 bar()方法绘制柱状图的示例代码如下：

```
#第 5 章//zz.py
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure("figure name")
```

```
ax = fig.add_subplot(111)
x = np.arange(10)
y = np.random.randn(10)
ax.bar(x,y,color = "#c248be",label = "bar",)
ax.legend(loc = 2)
plt.show()
```

代码绘制的柱状图，如图 5-11 所示。

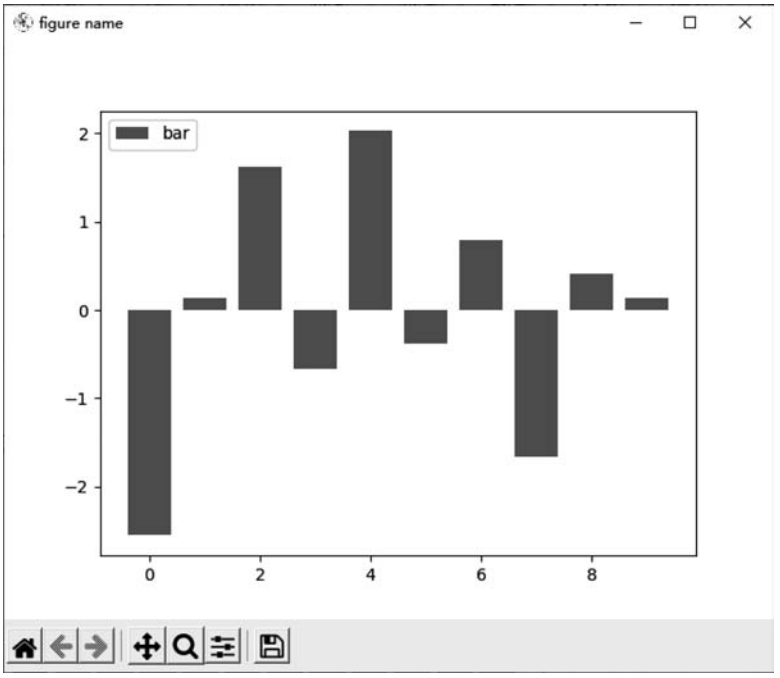

图 5-11　柱状图

5.3.4　饼图

使用 pie()方法可绘制饼图，pie()方法包含 15 个常用参数，分别为 x、labels、explode、startangle、shadow、labeldistance、autopct、pctdistance、radius、counterclock、wedgeprops、textprops、center、frame、rotatelabels。其中参数 x 为每一块的百分比。参数 labels 为每一块外侧显示的说明文字。参数 explode 为每一块离开中心的距离。参数 startangle 为起始绘制的角度，默认为从 x 轴正方向逆时针画起。参数 shadow 为阴影，默认值为 False，即不画阴影。参数 labeldistance 为 label 标记的绘制位置。参数 autopct 为控制饼图内百分比设置。参数 pctdistance 指定 autopct 的位置刻度，默认值为 0.6。参数 radius 为控制饼图半径，默认值为 1。参数 counterclock 为指定指针方向，默认为逆时针。参数 wedgeprops 将字典传递给 wedge 对象，用来画一个饼图。参数 textprops 用于设置标签和比例文字的格式。参数 center 为图标中心位置。参数 frame 为绘制带有表的轴框架，默认不绘制。参数 rotatelabels 将每个 label 旋转到指定的角度，默认不旋转。

使用 pie()绘制饼图，示例代码如下：

```
#第 5 章//bt.py
import matplotlib.pyplot as plt
fig = plt.figure("figure name")
ax = fig.add_subplot(111)
labels = 'A','B','C','D'
sizes = [10,10,20,60]
explode = (0,0,0.1,0)
colors = ['r','g','y','b']
ax.pie(sizes, explode = explode, labels = labels, colors = colors, autopct = ' % 1.2f % % ',
pctdistance = 0.4,shadow = True,labeldistance = 0.8,startangle = 30,radius = 1.3,counterclock
 = False,textprops = {'fontsize':20,'color':'black'})
plt.show()
```

代码绘制的饼图,如图 5-12 所示。

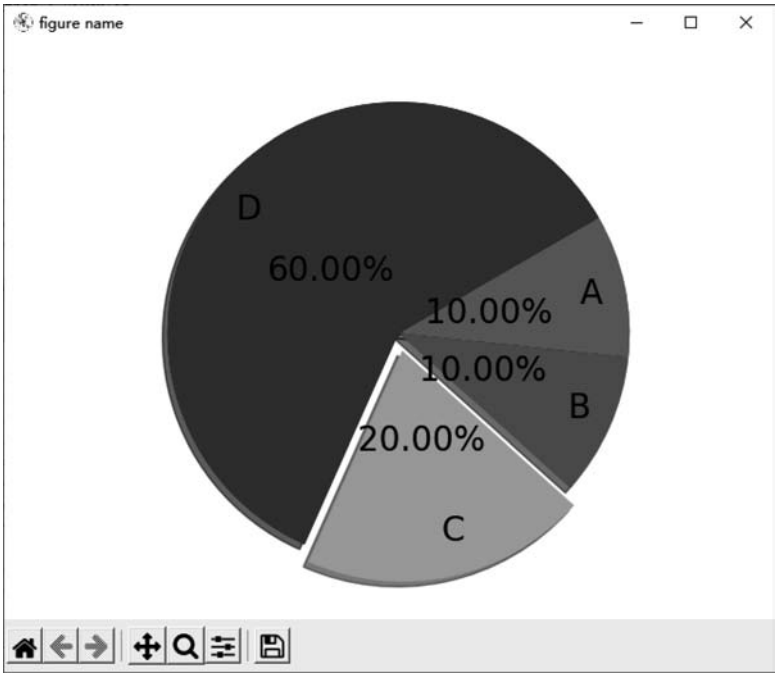

图 5-12　饼图

5.3.5　泡泡图

使用 scatter()方法可绘制散点图,通过设置 s 参数与 c 参数可将散点图变化为泡泡图,s 参数为散点标记的大小,c 参数为散点标记的颜色。散点图的示例代码如下:

```
#第 5 章//pp.py
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure("figure name")
ax = fig.add_subplot(111)
```

```
x = np.random.rand(50)
y = np.random.rand(50)
area = (20 * x + 20 * y) ** 2
ax.scatter(x,y,s = area,c = np.random.rand(50), marker = "o",alpha = 0.6)
plt.show()
```

代码绘制的泡泡图，如图 5-13 所示。

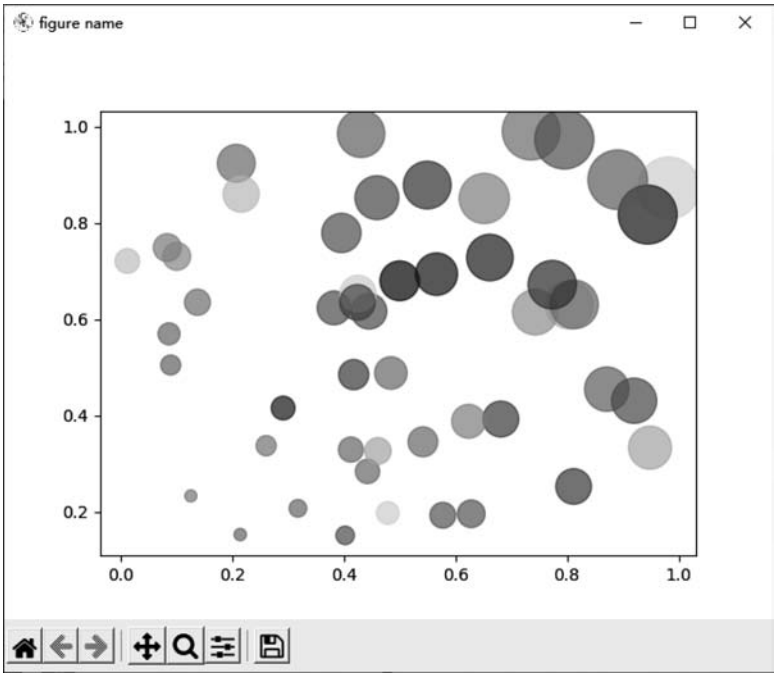

图 5-13　泡泡图

5.3.6　等高线

使用 contour()与 contourf()方法可绘制等高线，contour()与 contourf()方法的区别为 contour()方法只绘制轮廓线，而 contourf()方法除了绘制轮廓线，还会对绘制的轮廓线进行颜色填充。两种方法的参数是相同的，都包含了 8 个常用的参数，分别为 x、y、z、colors、alpha、cmap、linewidths、linestyles，其中参数 x 为 x 轴数据。参数 y 为 y 轴的数据。参数 z 为高度数据。参数 colors 为指定不同高度的等高线颜色。参数 alpha 为透明度。参数 cmap 为用不同颜色区分不同高度区域。参数 linewidths 为指定等高线的宽度。参数 linestyles 为指定等高线的样式。

绘制等高线的示例代码如下：

```
#第 5 章//dgx.py
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure("figure name")
ax = fig.add_subplot(211)
```

```
ax2 = fig.add_subplot(212)
n = 128
x = np.linspace( - 2,2,n)
y = np.linspace( - 2,5,n)
x,y = np.meshgrid(x,y)
def func(x,y):
    return (1 - x/2 + x ** 2 + y ** 3) * np.exp( - x ** 2 - y ** 4)
ax.contourf(x,y,func(x,y))
ax2.contour(x,y,func(x,y))
plt.show()
```

代码绘制的等高线如图 5-14 所示，其中上半部分使用了 contourf()方法进行绘制，下半部分使用了 contour()方法进行绘制。

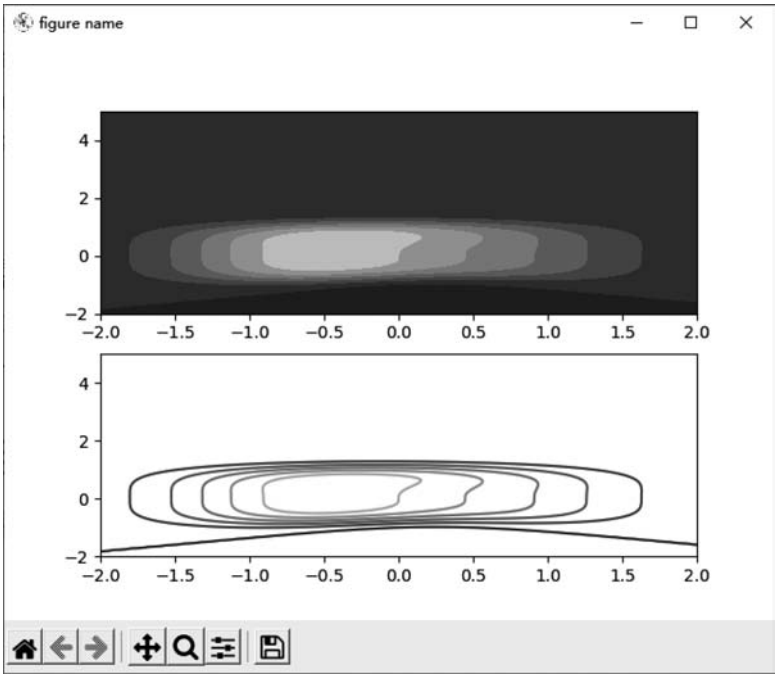

图 5-14　等高线