第3章

Python数据挖掘中的常用模块

Python具有强大的扩展能力，其中的数据分析与挖掘常用模块几乎可以满足人们的各种需求。例如，科学计算模块NumPy提供了矩阵运算；基于NumPy的数据分析处理模块Pandas提供了一些数据挖掘工具；数据可视化模块Matplotlib具有类似MATLAB的绘图工具；针对Python编程软件免费版的机器学习模块Scikit-learn具有常用的分类、回归和聚类等算法。

3.1 NumPy模块

NumPy(Numerical Python的简称)是Python的一个开源数值计算扩展模块，可以用来存储和处理大型矩阵。NumPy要比Python自身的嵌套列表结构高效得多，支持大维度数组与矩阵运算，并且针对数组运算提供了大量的数学函数库。

3.1.1 NumPy数据类型

NumPy提供了一个n维数组类型ndarray，描述了相同类型的“items”的集合。n维数组(ndarray)是NumPy主要的数据类型，数组的下标从0开始。

1. ndarray对象

ndarray对象可以通过一个常规的Python列表或者使用array()函数的元组来构建，通过调用生成后数组的dtype属性来了解该数组的元素类型。

导入NumPy库的语句形式主要有以下3种。

- import numpy as np：在这个方式下使用NumPy函数或属性时以np.开头。
- import numpy：在这个方式下使用NumPy函数或属性时以numpy.开头。
- from numpy import *：在这个方式下使用NumPy函数或属性时可以直接引用。

一般常用第一种方法，尽量不用第三种方法。以后如果没有特殊说明，都是默认使用第一种方法。

(1) 利用列表创建数组，其常用形式为np.array(object[,dtype][,ndmin])。

参数说明：

① object为同类型元素的列表或元组。

② dtype表示数组所需的数据类型，默认为None。

③ ndmin 为 int 类型，指定生成数组应该具有的最小维数，默认为 None。

例 3.1 用列表生成数组示例。

程序代码如下：

```
import numpy as np
data = [3, - 4,7,12]
x = np.array(data) #使用列表生成一维数组
print(x)
data = [[1,2],[3,4],[5,6]]
y = np.array(data) #使用列表生成二维数组
print(y)
```

(2) 利用 range()和 arange()函数生成一维数组，其常用形式如下。

① range(start，end，step)：返回一个 list 对象，起始值为 start，终止值为 end，但不含终止值，步长为 step。该函数只能创建 int 型 list。

② arange(start，end，step)：与 range()类似，但是返回一个 ndarray 对象，并且可以使用 float 型数据。

(3) 利用 arange()函数和 reshape()函数创建多维数组，其常用形式为：

```
np.arange(<elements_num>).reshape(<dimension_1>,<dimension_2>,…,<dimension_m>)
```

说明：该方法利用 arange()函数生成一维数组，而利用 reshape()函数将一维数组转换为多维数组。

例 3.2 创建数组示例。

程序代码如下：

```
import numpy as np
a = np.arange(0,1,0.1) #生成一维数组
print(a)
b = np.arange(10).reshape(2,5) #生成 2×5 数组
print(b)
```

2. matrix 对象

matrix 是 ndarray 的一个小分支，它拥有 ndarray 的所有特性。在 NumPy 中 matrix 的主要优势是具有简单的乘法运算符号。例如 a 和 b 是两个 matrix 类型的对象，则 a * b 就是矩阵积。

利用 mat()函数或 matrix()函数可以创建矩阵，其常用形式为 np.mat|matrix(<object>)，其中<object>可以是①字符串，例如 np.mat('1 2 3；4 5 3')；②嵌套列表，例如 np.mat([[1,5,10]，[1.5,6,14]])；③数组，例如 np.mat(np.random.rand(3,4))。

以下是创建矩阵的常见例句。

data1＝np.mat(np.zeros((3,3)))：创建一个 3×3 的零矩阵，其中 zeros()函数的参数是一个 tuple 类型。

data2＝ np.mat(np.ones((2,4)))：创建一个 2×4 的元素为 1 的矩阵，默认是浮点型的数据，如果需要是 int 类型，可以使用 dtype＝int。

data3＝np.mat(np.random.rand(2,2))：其中 random 使用的是 NumPy 中的 random 模块，random.rand(2,2)创建的是一个二维数组，需要将其转换成 matrix 类型。

data4＝np.mat(np.random.randint(10,size＝(3,3)))：生成一个 3×3 的 0～10 的随机整数矩阵，如果需要指定下界，则可以多加一个参数。

data5＝np.mat(np.random.randint(2,8,size＝(2,5)))：产生一个 2～8 的随机整数

矩阵。

data6＝np.mat(np.diag([1,2,3]))：生成一个对角线元素为1、2、3的对角矩阵。

3. Python列表与NumPy数组的区别

虽然Python列表与NumPy数组在形式和应用上类似，但是它们之间有着本质的区别，在数据科学领域中应用NumPy多维数组要比应用Python列表更有效。NumPy数组也支持“向量化”操作，可以有效地处理大型稀疏数据集。

Python列表与NumPy数组的区别如下：

(1) Python列表不需要指定长度、数据类型，可以进行索引、更新、删除、切片等操作，但创建NumPy数组时必须指定数组长度和数据类型。

(2) 列表是Python中内置的数据类型，使用方括号[]和逗号分隔符，其元素的类型可以不同，但NumPy数组中元素的类型必须相同。

(3) Python列表中所有元素的内存地址可以不是连续的，它是通过每个元素记录上一个元素的内存地址和下一个元素的内存地址来排列的，而NumPy数组是一个连续的内存空间，每个元素都按照先后顺序排列在内存中，所以存放数据还是尽量使用数组。

(4) Python列表中有append的方法，可以进行追加，而NumPy数组没有类似的方法。

(5) 用print()函数打印的结果不同，Python列表元素之间用逗号分隔，而NumPy数组元素之间用空格分隔。

3.1.2　NumPy基本运算

NumPy提供了一些用于算术运算的方法，使用起来会比Python提供的运算符灵活一些。

1. ndarray对象的基本运算

NumPy基本运算都是按元素操作的，下面给出一些基本运算。

假设a＝np.arange(3,11,2)，b＝np.array([1,4,3,2])，ndarray对象的基本运算如表3.1所示。

表3.1　ndarray对象的基本运算

运　算　符	说　　明	示　　例	结　　果
＋	加法	a＋b	[4　9 10 11]
－	减法	a－b	[2 1 4 7]
*	乘法	a*b,3*a	[3 20 21 18],[9 15 21 27]
/	除法	a/b	[3. 1.25　2.33333333 4.5]
%	取余数	b%a	[0 1 1 1]
**	幂运算	a**3	[27 125 343 729]

例3.3　多维数组基本运算示例。

程序代码如下：

```
import numpy as np
a = np.array([[1,2,3],[3,4,1]])
b = np.arange(6).reshape(2,3)
print(a,'\n',b)
print(a + b)
print(a - b)
print(a * b)
```

```
print(a**3)
print(2*a)
```

dot()函数返回的是两个数组(向量)的点积(dot product)。

例 3.4 数组对象乘法和点积运算示例。

程序代码如下:

```
import numpy as np
a = np.array([0,1,2,3])
b = np.array([2,1,3,-2])
c = np.array([[4,3],[2,1]])
d = np.array([[1,2],[3,4]])
print('一维数组乘法:',a*b)
print('一维数组点积:',np.dot(a,b))
print('二维数组乘法:\n',c*d)
print('二维数组点积:\n',np.dot(c,d))
```

2. matrix 对象的基本运算

NumPy 中的 ndarray 对象重载了许多运算符,使用这些运算符可以完成矩阵间对应元素的运算。matrix 对象的优势在于能够实现部分矩阵运算,例如矩阵乘法、逆矩阵等运算。

1) 矩阵乘法

假设 a、b 表示矩阵,a*b、np.dot(a,b)表示数学中的矩阵乘法运算,np.multiply(a,b)表示矩阵对应元素的乘积。

2) 矩阵转置

矩阵转置的形式为<matrix_variate>.transpose()|T。

例 3.5 矩阵转置示例。

程序代码如下:

```
import numpy as np
a = np.mat([[4,3,5,-4],[2,1,4,0]])      #定义 2×4 矩阵 a
print(a.transpose())                    #输出 a 的转置矩阵
print(a.T)                              #输出 a 的转置矩阵
```

3) 逆矩阵

求矩阵的逆矩阵需要先导入 numpy.linalg,用 linalg 的 inv()函数来求逆矩阵,形式如下:linalg.inv(<matrix_variate>)或<matrix_variate>.I

例 3.6 求逆矩阵示例。

程序代码如下:

```
import numpy as np
a = np.mat([[1,0,0],[3,4,0],[1,2,3]])   #定义 3×3 矩阵 a
print(np.linalg.inv(a))                 #输出 a 的逆矩阵
print(a.I)                              #输出 a 的逆矩阵
```

3.1.3 生成随机数的常用函数

在 NumPy 的 random 模块中含有两类随机数生成函数,一类是浮点型的,常以 uniform()为代表;另一类是整数型的,常以 randint()为代表。

1. uniform()函数

使用 uniform()函数可以生成给定范围内的浮点型随机数,可以是单个值,也可以是一维数组,还可以是多维数组。其常用语法形式如下:

```
np.random.uniform([<low>,<high>[,<size>]])
```

功能：从一个浮点数均匀分布[low,high]中随机采样，注意定义域是左闭右开，即包含low，不包含high。

参数说明：

(1) low为采样下界，float类型，默认值为0。

(2) high为采样上界，float类型，默认值为1。

(3) size为输出样本数目，int或元组(tuple)类型。例如size=(m,n,k)，则表示输出m*n*k个样本，该参数省略时输出一个值。

返回值：ndarray类型，其形状和size参数中的描述一致。

例3.7　使用uniform()函数生成随机数示例。

程序代码如下：

```
import numpy as np
print(np.random.uniform())                    #默认为0到1
print(np.random.uniform(1,5))                 #生成1～5的float数
print(np.random.uniform(1,5,4))               #生成一维数组
print(np.random.uniform(1,5,(4,3)))           #生成4×3的数组
print(np.random.uniform([1,5],[5,10]))        #生成两个元素的一维数组
```

2. randint()函数

使用randint()函数可以生成给定范围内的整型随机数，可以是单个值，也可以是一维数组，还可以是多维数组。其常用语法形式如下：

```
np.random.randint(<low>[,high=None],size=None,dtype='l')
```

功能：用于生成一个指定范围内的整数。其中low是下限、high是上限，生成的随机数n有low<=n<high，即[low,high)。

参数说明：

(1) low为int类型，是随机数的下限。

(2) high为int类型，默认为空，是随机数的上限，当该值为空时，函数生成[0,<low>]区间内的随机数。

(3) size为int类型或元组，指明生成模式。

(4) dtype表示元素的数据类型，可选'int'、'int64'，默认为'l'。

例3.8　randint()函数应用示例。

程序代码如下：

```
import numpy as np
print(np.random.randint(5))                                  #生成0～5的整数
print(np.random.randint(5,size=4))                           #生成0～5的4个元素的数组
print(np.random.randint(5,10,size=6))                        #生成5～10的6个元素的数组
print(np.random.randint(5,10,size=(2,3),dtype='int'))        #生成5～10的2×3数组
```

例3.9　随机函数生成验证码示例。

程序代码如下：

```
import random
#从一个字符串中随机生成若干个字符
def gen_code(n):
    s='er0dfsdfxcvbn7f989fd'
    code=''
    for i in range(n):
```

```
            r = random.randint(0,len(s) - 1)
            code += s[r]
        return code
username = input("输入用户名: ")
passwd = input("输入密码:")
code = gen_code(5)
print("验证码是: ",code)
code1 = input("输入验证码: ")
if code.lower() == code1.lower():
    if username == 'knn' and passwd == 'abc':
        print("Login success!")
    else:
        print("username or password error!")
else:
    print("check code error!")
```

3.1.4 对象转换

在使用字符串、列表、数组和矩阵的过程中经常需要进行相互转换，用户可以使用 type() 函数查看对象的类型。

1. 列表和字符串相互转换

1) 列表转换为字符串

如果列表中的各元素都是字符串，一般通过 join()函数转换成字符串，但是当列表中含有数字类型的元素时，需要先将数字类型的元素转换为字符串。

例 3.10 列表转换为字符串示例。

程序代码如下：

```
lst1 = ['This','is','an','apple']
print(' '.join(lst1))
#如果列表中包含数字类型的元素,需要先转换为字符串
lst2 = ['S1','许文秀','女',20,'计算机系']
print(' '.join([str(x) for x in lst2]))
```

2) 字符串转换为列表

字符串转换为列表的方法有两种，一是利用列表直接转换，即 list(< string >)；二是通过字符串的 split()方法转换。

例 3.11 字符串转换为列表示例。

程序代码如下：

```
str1 = 'Python program'
result1 = list(str1)                #用 list()转换
print(result1)
result2 = str1.split()              #默认以空格分隔
print(result2)
result3 = str1.split(',')           #以逗号分隔
print(result3)
```

2. 数组和字符串相互转换

将数组转换为字符串的方法和将列表转换为字符串的方法是一样的。

例 3.12 数组转换为字符串示例。

程序代码如下：

```
import numpy as np
lst = ['This','is','a','program']
```

```
arr = np.array(lst)
str1 = ' '.join(arr)
print(str1)
```

将字符串转换为数组可以先将字符串转换为列表，再将列表转换为数组。

例 3.13　字符串转换为数组示例。

程序代码如下：

```
import numpy as np
str1 = '567232'
lst = list(str1)
arr1 = np.array(lst)
print(arr1)
```

3. 列表和数组相互转换

使用 np.array()方法可以将列表转换为数组。

例 3.14　列表转换为数组示例。

程序代码如下：

```
import numpy as np
lst1 = [ - 3,4,8,6]
lst2 = [[2,3,4],[4,7,1]]
arr1 = np.array(lst1)
arr2 = np.array(lst2)
print(arr1)
print(arr2)
```

使用 tolist()或 list()方法可以将数组转换为列表。

例 3.15　数组转换为列表示例。

程序代码如下：

```
import numpy as np
arr1 = np.array([ - 3,4,8,6])
arr2 = np.array([[2,3,4],[4,7,1]])
lst1 = arr1.tolist()
lst2 = arr2.tolist()
print(lst1)
print(lst2)
```

4. 列表和矩阵相互转换

使用 np.mat()方法可以将列表转换为矩阵。

例 3.16　列表转换为矩阵示例。

程序代码如下：

```
import numpy as np
lst = [[1,2,3],[4,5,6]]
matr = np.mat(lst)
print(type(matr))
print(matr)
```

将矩阵转换为列表和将数组转换为列表的方法相同。

5. 数组和矩阵相互转换

将数组转换为矩阵和将列表转换为矩阵的方法相同，将矩阵转换为数组和将列表转换为数组的方法相同。

3.1.5　数组元素和切片

NumPy 多维数组和列表的类型非常类似，同样有访问数组元素和切片的功能，访问数组

元素是指获取数组中特定位置元素的过程，切片是指获取数组元素子集的过程。

1. 一维数组元素和切片

一维数组元素和切片与 Python 列表类似。

例 3.17 一维数组元素和切片示例。

程序代码如下：

```
import numpy as np
arr = np.array([ - 4,8,12,6,9,11,25])
print(arr[4])                    #输出下标为 4 的元素
print(arr[2:5])                  #输出下标为 2、3、4 的元素
print(arr[1:5:2])                #输出下标为 1、3 的元素
```

2. 多维数组元素和切片

多维数组元素的不同维度用逗号隔开，取切片用冒号，多维数组取步长用冒号。

例 3.18 多维数组元素和切片示例。

程序代码如下：

```
import numpy as np
arr = np.arange(12).reshape([3,4])
print(arr[1,2])                  #输出下标为[1,2]的元素
print(arr[1:,2:])                #输出行标为 1 和 2、列标为 2 和 3 的元素
print(arr[::2,1])                #输出行标为 0 和 2、列标为 1 的元素
```

3.2 Pandas 模块

Pandas 是基于 NumPy 的一种工具，该工具是为了解决数据分析任务而创建的。它采用的是矩阵运算，要比 Python 自带的字典或者列表的效率高得多。

3.2.1 Pandas 中的数据结构

Pandas 有两大数据结构——Series 和 DataFrame。数据分析的相关操作都围绕着这两种结构进行，它们需要用 import pandas as pd 语句导入。Series 是一种一维数组对象，DataFrame 是一个二维的表结构，类似 Excel。

1. Series 对象

Series 对象包含一组数据和一组索引，可以理解为一组带索引的数组。它由两个相关联的数组组合在一起，即主元素数组和 index 数组。index 数组可以是数字或者字符串。创建 Series 对象的常用形式如下：

(1) 通过列表创建，形式为 pd.Series(<list>[,index=<index_array>])，其中<list>为数据列表；<index_array>为 Series 的索引，如果没有定义，则默认为(0,1,2,…,n)。

(2) 通过字典创建，形式为 pd.Series({<key_1>:<value_1>,<key_2>:<value_2>,…,<key_n>:<value_n>})。

例 3.19 Series 对象创建示例。

程序代码如下：

```
import pandas as pd
s1 = pd.Series([12,5,7,21],index = [4,2,3,1])
s2 = pd.Series([12,5,7,21],index = ['a','b','c','d'])
s3 = pd.Series({'a':21,'b':213,'c':309,'d':210,'e':111})
print(s1)
print(s2)
print(s3)
```

2. DataFrame 对象

DataFrame 是一个表格型的数据结构，包含一组有序的列，每列可以是不同的值类型（数值、字符串、布尔型等）。它既有行索引又有列索引，可以被看作由 Series 组成的字典。

（1）用二维列表创建 DataFrame 对象，常用的形式为：

```
pd.DataFrame(<two_dimension_list>[,index=<line_index>][,columns=<column_index>])
```

参数说明：

① <two_dimension_list>表示二维列表数据。

② index 表示 DataFrame 的行索引，默认为(0,1,2,…,m)。

③ columns 表示 DataFrame 的列索引，默认为(0,1,2,…,n)。

例 3.20　用二维列表创建 DataFrame 对象示例。

程序代码如下：

```
import pandas as pd
datas=[['许文秀','女',20,'计算机系'],['刘世元','男',21,'电信系'],['刘德峰','男',22,'统计系'],
['于金凤','女',20,'计算机系']]
line_index=['S1','S2','S3','S4']
column_index=['姓名','性别','年龄','系部']
df=pd.DataFrame(datas,index=line_index,columns=column_index)
print(df)
```

（2）用字典方式创建 DataFrame 对象，形式为：

```
pd.DataFrame(<dict>[,index=<line_index>])
```

参数说明：

① dict 表示字典数据。

② index 表示 DataFrame 的行索引，默认为(0,1,2,…,m)。

例 3.21　用字典方式创建 DataFrame 对象示例。

程序代码如下：

```
import pandas as pd
datas ={'姓名':['许文秀','刘世元','刘德峰','于金凤'],'性别':['女','男','男','女'],'年龄':[20,21,
22,20],'系部':['计算机系','电信系','统计系','计算机系']}
line_index=['S1','S2','S3','S4']
df=pd.DataFrame(datas,index=line_index)
print(df)
```

3.2.2　DataFrame 的基本属性

DataFrame 的基础属性有 values、index、columns、ndim 和 shape 等，分别用来获取 DataFrame 的元素、索引、列名、维度和形状。假设 df=pd.DataFrame({'姓名':['许文秀','刘世元'],'性别':['女','男'],'年龄':[20,21],'系部':['计算机系','电信系']},index=['S1','S2'])，DataFrame 的基本属性如表 3.2 所示。

表 3.2　DataFrame 的基本属性

属性和方法	说　明	示　例	结　果
values	获取 ndarray 类型的元素	df.values	[['许文秀' '女' 20 '计算机系'] ['刘世元' '男' 21 '电信系']]
index	获取行索引	df.index	index(['S1','S2'],dtype='object')

续表

属性和方法	说　　明	示　　例	结　　果
axes	获取行索引及列索引	df. axes	[index(['S1' 'S2'],dtype='object'),index(['姓名','性别','年龄','系部'],dtype='object')]
columns	获取列名列表	df. columns	index(['姓名','性别','年龄','系部'],dtype='object')
size	获取元素个数	df. size	8
ndim	获取维度	df. ndim	2
shape	获取形状	df. shape	(2,4)

3.2.3 DataFrame 的常用方法

继续利用 3.2.2 节中的例子,DataFrame 的常用方法如表 3.3 所示。

表 3.3 DataFrame 的常用方法

属性和方法	说　　明	示　　例	结　　果
iloc[<行序>,<列序>]	按序号获得元素	df. iloc[:,0:2]	姓名　性别 S1　许文秀　女 S2　刘世元　男
loc[<行索引>,<列索引>]	按索引获得元素	df. loc['S2','姓名']	刘世元
df. head(i)	显示前 i 行数据	df. head(1)	姓名　性别　年龄　系部 S1　许文秀　女　20　计算机系
df. tail(i)	显示后 i 行数据	df. tail(1)	姓名　性别　年龄　系部 S2　刘世元　男　21　电信系
df. describe()	查看数据列的统计信息	df. describe()	年龄列的 count、mean、std、min、max、25%(下四分位数)、50%(中位数)、75%(上四分位数)的值
del	删除指定列	del df['性别']	删除性别列
drop(columns=[<列名 1>,<列名 2>,…,<列名 n>])	同时删除多个列	df. drop(columns=['性别','系部'])	姓名　年龄 S1　许文秀　20 S2　刘世元　21
rename(columns={<原名 1>:<新名 1>,<原名 2>:<新名 2>,…,<原名 n>:<新名 n>})	同时修改多个列名	df. rename(columns={'姓名':'Name','性别':'Sex','年龄':'Age','系部':'Dept'})	Name Sex Age Dept S1　许文秀　女　20　计算机系 S2　刘世元　男　21　电信系
df. copy()	复制 df	df1=df. copy()	df1 中为 df 内容

3.2.4 DataFrame 的数据查询与编辑

1. 数据查询

数据查询一般都是通过索引来操作的。

1) 查询列数据

通过列索引标签或者属性的方式可以单独获取 DataFrame 的列数据,返回的数据类型为 Series。注意在选取列时不能使用切片的方式,若超过一个列名,用 df[[< column_name_1 >,< column_name_2 >,…,< column_name_k >]]形式。

例 3.22　查询列数据示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas,index = ['S1','S2','S3','S4'])
print('查询姓名列: \n',df[['姓名']])
print('查询姓名和年龄列\n: ',df[['姓名','年龄']])
```

2）查询行数据

通过行索引或者行索引位置的切片形式获取行数据(从 0 开始,左闭右开)。DataFrame 提供的 head()和 tail()方法分别用于获取开始和末尾的连续多行数据。

例 3.23　查询行数据示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas,index = ['S1','S2','S3','S4'])
print('查询前两行: \n',df[:2])
print('查询第 2 行: \n',df[1:2])
print('查询前 3 行: \n',df.head(3))
print('查询后两行: \n',df.tail(2))
```

3）同时查询行和列

切片查询行的限制比较大,查询单独的几行数据可以采用 Pandas 提供的 iloc[]和 loc[]方法实现。

例 3.24　同时选择行和列示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas,index = ['S1','S2','S3','S4'])
print('查询序号为 S1 和 S3 的同学的姓名和系部: \n',df.loc[['S1','S3'],['姓名','系部']])
print('查询第 1 行和第 3 行的第 2 列: \n',df.iloc[[1,3],[1]])
```

4）条件查询

条件查询由逻辑表达式构成查询条件,获取符合条件的记录。

例 3.25　条件查询示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas,index = ['S1','S2','S3','S4'])
```

```
print('查询姓名为刘德峰的行数据\n',df[df['姓名'] == '刘德峰'])
print('查询计算机系的女同学: \n',df[(df['性别'] == '女') & (df['系部'] == '计算机系')])
```

2. 数据编辑

DataFrame 对象的数据编辑方法有很多,这里只介绍常用的几种。

1) 添加数据

添加一行数据可以通过 append()方法实现,创建一个新的数据列的方法类似于在字典中添加项的方法。

例 3.26 添加新的行或列示例。

程序代码如下:

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas)
data_1 = {'姓名':'王文庆','性别':'男','年龄':21,'系部':'电信系'}
df1 = df.append(data_1,ignore_index = True) #添加一行数据
print('添加新的数据行: \n',df1)
df1['籍贯'] = ['河北省','天津市','河北省','重庆市','江苏省']  #添加一列数据
print('添加新的数据列: \n',df1)
```

2) 删除数据

删除数据直接用 drop()方法,行、列数据通过 axis 参数设置,其中 0 默认为删除行,1 默认为删除列。默认删除数据不修改原数据,inplace=True 表示在原数据上删除。

例 3.27 删除行或列示例。

程序代码如下:

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas)
df1 = df.drop([2],axis = 0,inplace = False) #删除序号为 2 的行,即第 3 行
print('删除第 3 行: \n',df1)
df.drop('系部',axis = 1,inplace = True) #删除系部列
print('删除系部列: \n',df)
```

3) 修改数据

修改行、列标题使用 df.rename()方法,修改数据只需要对选择的数据进行赋值。

例 3.28 修改行、列标题示例。

程序代码如下:

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas)
print('原数据: \n',df)
df1 = df.rename({0:'S1',1:'S2',2:'S3',3:'S4'}) #修改行标题
print('修改行标题: \n',df1)
df1.rename(columns = {'姓名':'name','性别':'sex','年龄':'age','系部':'dept'},inplace = True)
```

```
print('修改列标题: \n',df1)
```

例 3.29　数据修改示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas)
df.loc[2,'姓名'] = '刘得锋'      #将姓名刘德峰改成刘得锋,或用 df.iloc[2,0]
print('修改一个元素的值: \n',df)
df.loc[1,['姓名','性别']] = ['陈晓晴','女'] #修改第 2 行的姓名和性别
print('修改某行几个元素的值: \n',df)
df.iloc[:,2] = [21,22,23,21]   #修改第 3 列的内容
print('修改某列的值: \n',df)
```

3.2.5　Pandas 数据的四则运算

两个 DataFrame 对象可以进行四则运算，运算法则是对应元素进行运算，对于不同维度的对象，无对应元素结果取 NaN 值，也可以利用 fill_value=< data_values >先填充，后计算。

例 3.30　两个 DataFrame 对象的四则运算示例。

程序代码如下：

```
import numpy as np
import pandas as pd
df1 = pd.DataFrame(np.arange(12).reshape((3,4)),columns = list('abcd'))
df2 = pd.DataFrame(np.arange(20).reshape((4,5)),columns = list('abcde'))
print('A = \n',df1,'\nB = \n',df2) #打印 DataFrame 对象
print('A + B = \n',df1.add(df2,fill_value = 0)) #(或用 df1 + df2)输出 df1 与 df2 的和
print('A - B = \n',df1.sub(df2,fill_value = 0)) #(或用 df1 - df2)输出 df1 与 df2 的差
print('A * B = \n',df1.mul(df2,fill_value = 0)) #(或用 df1 * df2)输出 df1 与 df2 的积
print('A/B = \n',df1.div(df2,fill_value = 0)) #(或用 df1/df2)输出 df1 与 df2 的商
```

3.2.6　函数变换

在数据清洗工作中经常需要对数据进行各种函数变换，为此 Pandas 提供了 map()、apply()和 applymap()等函数。

1. map()函数

map()函数主要用在 Series 对象中，用来对 Series 对象中的元素进行变换。

例 3.31　map()函数变换示例。

程序代码如下：

```
import pandas as pd
datas = {'姓名':['许文秀','刘世元','刘德峰','于金凤'],
        '性别':['女','男','男','女'],
        '年龄':[20,21,22,20],
        '系部':['计算机系','电信系','统计系','计算机系']}
df = pd.DataFrame(datas)
def sex_map(x): #将性别'男'改为'M'、'女'改为'F'
    sex = 'F'
    if x == '男': sex  = 'M'
    return sex #该函数也可以用字典 sex_map = {'男':'M','女':'F'}替代
```

```
df['性别'] = df['性别'].map(sex_map)
print(df)
```

2. apply()函数

apply()是 Pandas 模块的一个很重要的函数,可以直接用于 DataFrame 和 Series 对象,它经常结合 NumPy 库和隐函数 lambda 来使用。

例 3.32 apply()函数应用示例。

程序代码如下:

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.arange(12).reshape((3,4)),columns = list('ABCD'))
print('df 中原始数据为: \n',df)
print('计算每个元素的平方根: \n',df.apply(np.sqrt))
print('计算每一列元素的平均值:\n',df.apply(np.mean))
print('计算每一行元素的平均值:\n',df.apply(np.mean,axis = 1))
print('增加第 E 列,为前面 4 列元素之和: ')
def Add_a(x):
    return x.A + x.B + x.C + x.D
df['E'] = df.apply(Add_a,axis = 1)
print(df)
print('列 E 中的所有元素加 5: ')
df.E = df.E.apply(lambda x:x + 5)
print(df)
print('第 E 列元素被 3 整除的均赋值 Yes,否则赋值 No: ')
df.E = df.E.apply(lambda x: 'Yes' if x % 3 == 0 else'No')
print(df)
```

3. applymap()函数

applymap()函数是元素级别的操作,返回结果是对 DataFrame 中的每个元素执行指定的函数操作。

例 3.33 applymap()函数应用示例。

程序代码如下:

```
import pandas as pd
df = pd.DataFrame([[1,2.12,3.245],[3.356,4.56,2.1101]],columns = list('abc'),index = [2,3])
def f(x):
   return len(str(x)) #对每一个元素求长度,其中 1 按 1.0 计算
print(df.applymap(f))
```

4. map()、apply()和 applymap()在应用上的区别

简单总结起来,这 3 个函数在应用上有以下几点区别:

(1) map()函数只能作用于 Series 对象中的每个元素。

(2) apply()函数既可以作用于 Series 对象中的每个元素,也可以作用于 DataFrame 对象中的行或列。

(3) applymap()只能作用于 DataFrame 对象中的每个元素。

3.2.7 排序

在数据分析过程中,有时需要根据索引的大小或者值的大小对 Series 对象和 DataFrame 对象进行排序,使用 sort_index()函数可以根据行或列的索引进行排序,使用 sort_values()函数可以根据行或列的值进行排序。

例 3.34　Series 对象按索引进行排序示例。

程序代码如下：

```
import pandas as pd
s = pd.Series([1,2,3],index = ['a','c','b'])
print('按 Series 对象的索引进行升序排序：\n',s.sort_index()) #默认是升序排序
print('按 Series 对象的索引进行降序排序：\n',s.sort_index(ascending = False))
```

例 3.35　Series 对象按值进行排序示例。

程序代码如下：

```
import pandas as pd
import numpy as np
s = pd.Series([np.nan,1,7,2,3],index = ['a','c','e','b','d'])
print('按 Series 对象的值进行升序排序：\n',s.sort_values()) #默认是升序排序
print('按 Series 对象的值进行降序排序：\n',s.sort_values(ascending = False))
```

说明：在对值进行排序的时候，无论是升序还是降序，缺失值(NaN)都会排在最后。

例 3.36　DataFrame 对象按索引排序示例。

程序代码如下：

```
import numpy as np
import pandas as pd
a = np.arange(9).reshape(3,3)
data = pd.DataFrame(a,index = ['0','2','1'],columns = ['c','a','b'])
print('按行的索引进行升序排序:\n',data.sort_index())  #默认按行升序排序
print('按行的索引进行降序排序:\n',data.sort_index(ascending =  False))
print('按列的索引进行升序排序:\n',data.sort_index(axis = 1))  #默认按列升序排序
print('按列的索引进行降序排序：\n',data.sort_index(axis = 1, ascending = False))
```

例 3.37　DataFrame 对象按值排序示例。

程序代码如下：

```
import pandas as pd
import numpy as np
data = [[9,3,1],[1,2,8],[1,0,5]]
df = pd.DataFrame(data,index = ['S1','S2','S3'],columns = ['c','a', 'b'])
print('按指定列的值大小顺序进行排序：\n',df.sort_values(by = 'c')) #默认升序
print('按指定多列的值大小顺序进行排序：\n',df.sort_values(by = ['c','a']))
#在对 DataFrame 对象的值进行排序时,要使用 by 指定某一行(列)或者某几行(列)
print('按指定行值进行排序：\n',df.sort_values(by = 'S1',axis = 1))
#在指定行值进行排序的时候必须设置 axis = 1
```

3.2.8　汇总与统计

Pandas 对象拥有一组常用的数学和统计方法，和 NumPy 数组相比，它们只对真实值(非缺失数据)进行统计。

1. 常用方法

假设 df＝pd. DataFrame([[np. nan,1,3],[4,5,6]],index＝{'S1','S2'},columns＝['c','a','b'])，Pandas 对象的常用方法及示例如表 3.4 所示。

2. 协方差和相关系数

协方差(Covariance)反映两个样本/变量 X、Y 之间的相互关系以及相关程度。如果协方差为正，说明 X、Y 同向变化，协方差越大，说明同向程度越高；如果协方差为负，说明 X、Y 反向变化，协方差越小，说明反向程度越高。

表 3.4 Pandas 对象的常用方法及示例

方　法	说　明	示　例	结　果
count	按列统计非 NaN 值的数量	df.count()	输出列名及各列非 NaN 值的数量
describe	针对 Series 或各 DataFrame 列计算汇总统计	df.describe()	输出各列的 count、mean、std、min、max 等值
max、min	最大值和最小值	df.max()	输出列名及各列的最大值
sum	值的总和	df.sum()	输出列名及各列的和
mean	值的平均数	df.mean()	输出列名及各列的平均值
median	值的算术中位数(二分位数)	df.median()	输出列名及各列的中位数
var	样本值的方差	df.var()	输出列名及各列的方差
std	样本值的标准差	df.std()	输出列名及各列的标准差
diff	一阶差分	df.diff()	输出列名及各列的一阶差分

相关系数(Correlation-coefficient)反映两个样本/变量之间的相关程度。相关系数也可以看成是两个变量的量化影响、标准化后的特殊协方差。相关系数在－1～＋1 变化：

(1) 当相关系数为 1 的时候两者相似度最大，同向正相关。

(2) 当相关系数为 0 的时候两者没有相似度，两个变量无关。

(3) 当相关系数为－1 的时候两者变化的反向相似度最大，完全反向负相关。

使用 cov()方法和 corr()方法分别计算两个 Series 对象或 DataFrame 对象的协方差和相关系数。

例 3.38 协方差和相关系数示例。

程序代码如下：

```
import pandas as pd
df = pd.DataFrame({
        "年龄":[8,9,10,11,12],
        "身高":[130,135,140,141,150]})
print('\n 年龄和身高的协方差为: ',df['年龄'].cov(df['身高']))
print('\n 年龄和身高的相关系数为: ',df['年龄'].corr(df['身高']))
```

3.2.9 数据的分组与统计

Pandas 提供了较好的分组与统计功能，可以很方便地对数据进行不同维度的分组与统计操作。

1. 分组

分组(Groupby)就是对数据集进行分组，然后对每组数据进行统计分析。Pandas 使用 groupby()进行分组，它没有进行实际运算，只是包含分组的中间数据。

按列名分组的常用形式为<DataFrame_object>.groupby(<column_name>)。

例 3.39 groupby()分组运算示例。

程序代码如下：

```
import pandas as pd
import numpy as np
dict = {'key1' : ['a', 'b', 'a', 'b','a', 'b', 'a', 'a'],
       'key2' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],
       'data1': np.random.randn(8), 'data2': np.random.randn(8)}
df = pd.DataFrame(dict)
print('\n 创建的 DataFrame 对象: \n',df)
grouped1 = df.groupby('key1')
```

```
print('\n 第一分组均值: \n',grouped1.mean())      # DataFrame 根据 key1 进行分组
grouped2 = df['data1'].groupby(df['key1'])        # DataFrame 的 data1 列根据 key1 进行分组
print('\n 第二分组均值: \n',grouped2.mean())
print('\n 分组一元素个数: \n',grouped1.size())
print('\n 分组二元素个数: \n',grouped2.size())
```

用户可以将分组关键字定义为列表或多层列表,也可以选定多个关键字分组。

例 3.40　自定义 key 分组及多层分组示例。

程序代码如下:

```
import pandas as pd
import numpy as np
dict = {'key1' : ['a', 'b', 'a', 'b','a', 'b', 'a', 'a'],
        'key2' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],
        'data1': np.random.randn(8),'data2': np.random.randn(8)}
df = pd.DataFrame(dict)
print(df)
self_def_key = [0, 1, 2, 3, 3, 4, 5, 7]
print(df.groupby(self_def_key).size())              # 按自定义关键字分组,列表
print(df.groupby([df['key1'], df['key2']]).size())  # 按自定义关键字分组,多层列表
grouped2 = df.groupby(['key1', 'key2'])             # 按多个列进行多层分组
print(grouped2.size())
grouped3 = df.groupby(['key2', 'key1'])             # 多层分组按关键字顺序进行
print(grouped3.mean())
# unstack()可以将多层索引结果转换成单层的 DataFrame
print(grouped3.mean().unstack())
```

2. 统计

Pandas 提供了基于行和列的统计操作,统计运算方法有 sum()、mean()、max()、min()、size()、describe()等。

例 3.41　分组后应用统计函数示例。

程序代码如下:

```
import pandas as pd
import numpy as np
dict = {'key1' : ['a', 'b', 'a', 'b','a', 'b', 'a', 'a'],
       'key2' : ['one', 'one', 'two', 'three','two','two', 'one', 'three'],
       'data1': np.random.randn(8),'data2': np.random.randn(8)}
df = pd.DataFrame(dict)
print(df)
# 使用统计函数
print(df.groupby('key1').sum())
print(df.groupby('key1').max())
print(df.groupby('key1').min())
print(df.groupby('key1').mean())
print(df.groupby('key1').size())
print(df.groupby('key1').count())
print(df.groupby('key1').describe())
```

3.2.10　Pandas 数据的读取与存储

数据大部分存储在文件中,因此 Pandas 支持复杂的 I/O 操作,它的 API 支持众多的文件格式,例如 CSV、TXT、Excel、MySQL 等。

1. CSV 文件

CSV(Comma Separated Values)文件是最通用的一种文件格式,它可以被非常容易地导

入各种计算机表格及数据库中。此文件一行即为数据表的一行，生成的数据表字段用半角逗号隔开。Pandas 利用 writerow()方法写入 CSV 文件数据，利用 read_csv()方法读取 CSV 文件数据。

例 3.42 在指定路径下创建 CSV 文件示例。

程序代码如下：

```
import csv
import os
os.chdir('D:\\Data_Mining') #改变当前路径
head = ['学号','姓名','性别','年龄','系部'] #定义文件头
lst = [['S1','许文秀','女',20,'计算机系'],
       ['S2','刘世元','男',21,'电信系'],
       ['S3','刘德峰','男',22,'统计系'],
       ['S4','于金凤','女',20,'计算机系']]
with open ('test.csv', 'a', newline = '') as f: #以追加方式打开或创建
   f_csv = csv.writer(f)
   f_csv.writerow(head) #写入文件头
   for i in range(4): #按行写入文件
      f_csv.writerow(lst[i])
```

例 3.43 Pandas 读取 CSV 文件示例。

程序代码如下：

```
import pandas as pd
import os
os.chdir('D:\\Data_Mining') #设置当前路径
data = pd.read_csv('test.csv',encoding = 'gb18030') #读取文件数据
print(data)
```

2. Excel 文件

Pandas 依赖 xlrd 模块处理 Excel 文件，因此需要提前安装该模块，其安装命令为 pip install xlrd。注意 xlrd 1.2.0 之后的版本不支持 XLSX 格式，仅支持 XLS 格式。

Pandas 利用 read_excel()方法读取 Excel 文件数据后返回 DataFrame 对象。

例 3.44 Pandas 读取 Excel 数据文件示例。

程序代码如下：

```
import pandas as pd
file = 'D:\\Data_Mining\stud.xls'
df = pd.read_excel(file)
print('获得 Excel 文件数据:\n',df)
```

如果是将整个 DataFrame 写入 Excel，则调用 to_excel()方法即可实现。

例 3.45 Pandas 将 DataFrame 对象数据写入 Excel 文件示例。

程序代码如下：

```
import pandas as pd
head = ['学号','姓名','性别','年龄','系部']
data = [['S1','许文秀','女',20,'计算机系'],
       ['S2','刘世元','男',21,'电信系'],
       ['S3','刘德峰','男',22,'统计系'],
       ['S4','于金凤','女',20,'计算机系']]
df = pd.DataFrame(data,columns = head)
file = 'D:\\Data_Mining\stud11.xlsx'
```

```
df.to_excel(file)
```

3. 读取 MySQL 数据

Pandas 利用 read_sql()方法读取 MySQL 文件。

例 3.46　读取 MySQL 数据示例。

程序代码如下：

```
import pymysql
con = pymysql.connect(host = 'localhost',user = 'root', password = 'root', database = 'test',port =
3306,charset = 'utf - 8')
sql_select = 'select * from a'df = pd.read_sql(sql_select, con)
```

其中，host 是主机名，一般填写 IP 地址，user 代表数据库用户名，password 代表数据库密码，database 代表需要连接的数据库名称，port 代表端口，charset 代表编码格式，一般使用 utf-8。

3.3　Matplotlib 图表绘制基础

Matplotlib 是一个 Python 工具箱，用于科学计算的数据可视化，它是一个非常好用的库，无论是写论文需要画图，还是在数据调研中显示数据相关性，都是一个很好的选择。

3.3.1　Matplotlib 简介

Matplotlib 是 Python 中基于 NumPy 的一套绘图工具包。Matplotlib 提供了一整套在 Python 下实现类似 MATLAB 的纯 Python 第三方库，其风格与 MATLAB 相似，并且继承了 Python 简单明了的优点。近年来，在开源社区的推动下，Matplotlib 在科学计算领域得到了广泛应用，成为 Python 中应用非常广的绘图工具包之一。

3.3.2　Matplotlib 绘图基础

Matplotlib 绘图流程如图 3.1 所示。

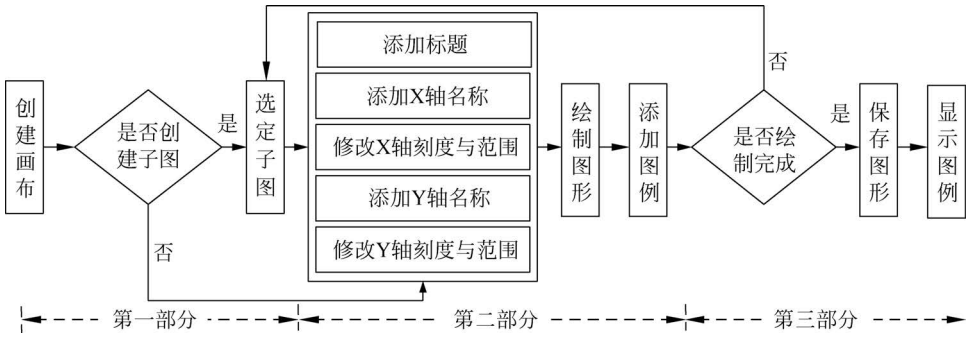

图 3.1　Matplotlib 绘图流程

使用 Matplotlib 模块进行绘图主要有 3 个步骤，第一步创建画布与子图；第二步准备数据，进行绘图及修饰；第三步保存与显示图形。

1. 创建画布与子图

该部分的主要作用是创建一张空白画布，并可以选择是否将整个画布划分为多个部分，方便在同一幅图上绘制多个图形。最简单的绘图可以省略第一部分，然后直接在默认的画布上进行图形绘制。创建画布与子图涉及的函数如表 3.5 所示。

表 3.5 创建画布与子图涉及的函数

函 数 名	功 能
figure()	创建一个空白画布,可以指定画布的大小、像素
figure.add_subplot()	创建并选中子图,可以指定子图的行数、列数
subplots_adjust()	调整子图之间的间距,wspace 为调整宽度,hspace 为调整高度

创建子图的常用步骤如下:

(1) 利用 plt.figure()函数创建画布(只绘制一幅图时,取默认画布,所以可以省略这一步)。

(2) 利用 plt.subplot()函数创建子图,需要传入行、列、索引等参数。

(3) 利用 axi=plt.subplot(m,n,i)(i=1,2,…,m×n)在画布中创建 m×n 个图形。

(4) 给子图 axi 绘制图形。

(5) 利用 plt.show()展示图片,释放内存。

例 3.47 创建多个子图示例。

程序代码如下:

```
import numpy as np
import matplotlib.pyplot as plt
x1 = np.arange( - 2,2,0.01)
p1 = plt.figure(figsize = (8,4),dpi = 80) # 确定画布大小
ax1 = p1.add_subplot(1,2,1) # 创建一个 1 行 2 列的子图,并开始绘制第一幅
plt.title('Power Function')
plt.plot(x1,x1 ** 2)
plt.plot(x1,x1 ** 4)
plt.legend(['y = x^2','y = x^4'])
ax2 = p1.add_subplot(1,2,2)
plt.title('e^x/log(x)')
x2 = np.arange(1,4,0.01)
plt.plot(x2,np.exp(x2))
plt.plot(x2,np.log(x2))
plt.legend(['y = e^x','y = log(x)'])
plt.show()          # 显示绘制出的图
```

程序运行结果如图 3.2 所示。

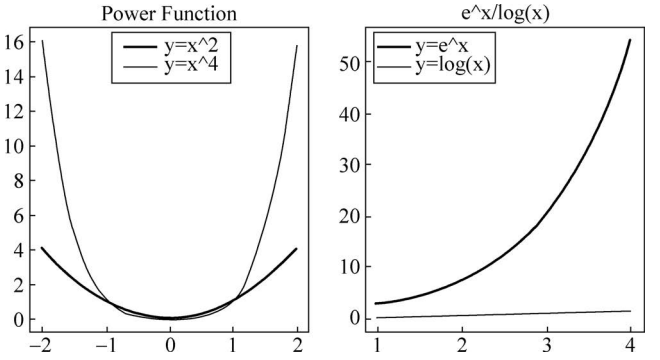

图 3.2 例 3.47 的运行结果

2. 绘图及修饰

该部分是绘图的主体部分,其中添加标题、坐标轴名称,绘制图形等步骤是并列的,没有先后顺序,可以先绘制图形,也可以先添加各类标签,但是添加图例一定要在绘制图形之后。绘图及修饰涉及的函数如表 3.6 所示。

表 3.6　绘图及修饰涉及的函数

函　数　名	功　　能
plt.title()	在当前图形中添加标题，可以指定标题的名称、位置、颜色、字体大小等参数
plt.xlabel()	在当前图形中添加 X 轴名称，可以指定位置、颜色、字体大小等参数
plt.ylabel()	在当前图形中添加 Y 轴名称，可以指定位置、颜色、字体大小等参数
plt.xlim()	指定当前图形的 X 轴的范围，只能确定一个数值区间，而无法使用字符串标识
plt.ylim()	指定当前图形的 Y 轴的范围，只能确定一个数值区间，而无法使用字符串标识
plt.xticks()	指定 X 轴刻度的数目与取值
plt.yticks()	指定 Y 轴刻度的数目与取值
plt.legend()	指定当前图形的图例，可以指定图例的大小、位置、标签

3. 保存与显示图形

该部分用于保存与显示图形，主要涉及的函数如表 3.7 所示。

表 3.7　保存与显示图形涉及的函数

函　数　名	功　　能
plt.savafig()	保存绘制的图形，可以指定图片的分辨率、边缘的颜色等参数
plt.show()	在本机显示图形

例 3.48　某市某周每天的最高气温如表 3.8 所示。

表 3.8　某市某周每天的最高气温

日期	周一	周二	周三	周四	周五	周六	周日
最高气温/℃	15	20	22	23	20	18	16

绘制该周的温度变化曲线。

程序代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(1,8)                    #绘图及修饰
y = np.array([15,20,22,23,20,18,16])
plt.plot(x,y)
plt.show()                            #显示图形
```

程序运行结果如图 3.3 所示。

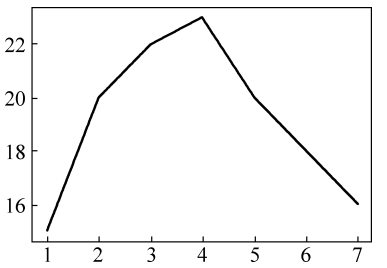

图 3.3　例 3.48 的运行结果

3.3.3　使用 Matplotlib 简单绘图

Matplotlib 是 Python 中的一个简单又完善的开源绘图库，用户只需要编写几行代码就可以生成绘图。Matplotlib 含有 pyplot 和 pylab 两个绘图模块，其中 pyplot 提供了一套和 MATLAB 类似的绘图 API，将众多绘图对象所构成的复杂结构隐藏在这套 API 内；pylab 包含了许多 NumPy 和 pyplot 模块中常用的函数，方便用户快速进行计算和绘图，非常适合在 Python 交互式环境中使用。

1. pyplot 模块

Matplotlib.pyplot 是一个有命令风格的函数集合，各种状态通过函数调用保存起来，以便随时跟踪当前图像和绘图区域等对象。利用绘图函数可以绘制折线图、散点图、柱状图、饼图、直方图等图像。引入 pyplot 模块的常用语句为 import matplotlib.pyplot as plt。

1）plot()函数

使用 Matplotlib 提供的 plot()函数绘制二维图像，展现变量的变化趋势。plot()函数的

常用形式如下：

```
plt.plot(<x>,<y>,<style>,<line_width>,<label>)
```

参数说明：

（1）x 是 X 轴上的有效坐标数组。

（2）y 是 Y 轴上的有效坐标数组。

（3）style 为线条风格，可选，它由表 3.9～表 3.11 所示的颜色字符、风格字符和标识字符组成。

（4）line_width 为折线图的线条宽度。

（5）label 是标识图内容的标签文本。

表 3.9　常用颜色字符表

颜色字符	说　明	颜色字符	说　明
'b'	蓝色	'm'	洋红色(magenta)
'g'	绿色	'y'	黄色
'r'	红色	'k'	黑色
'c'	青色(cyan)	'w'	白色
'#008000'	RGB 某颜色	'0.8'	灰度值字符串

表 3.10　常用风格字符表

风格字符	说　明	风格字符	说　明
'-'	实线	'-.'	点画线
'--'	破折线	":"	虚线

表 3.11　常用标识字符表

标识字符	说　明	标识字符	说　明
'o'	实心圈标识	'x'	x 标识
'v'	倒三角形标识	'D'	菱形标识
'^'	上三角形标识	'd'	瘦菱形标识
'+'	十字标识	'*'	星形标识

例 3.49　plot()函数线条风格应用示例。

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(10)
plt.plot(x,x*1.5,'g+:',x,x*2.5,'ro-.',x,x*3.5,'x--',x,x*4.5,'bd-')
plt.show()
```

程序运行结果如图 3.4 所示。

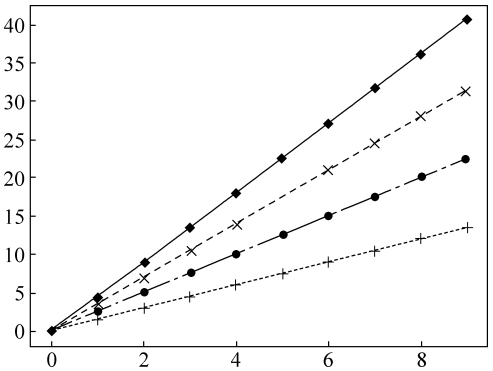

图 3.4　例 3.49 的运行结果

例 3.50　plot()函数绘图示例。

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(0.01,np.e, 0.01)
y1 = np.exp(-x)
y2 = np.log(x)
fig = plt.figure()
ax1 = fig.add_subplot(111)
ax1.plot(x, y1,'r',label = 'exp(-x)')
ax1.legend(bbox_to_anchor = (1,0.5))
```

```
ax1.set_ylabel('Y values for exp(-x)',color='r')
ax1.set_xlabel('X values')
ax1.set_title('Double Y axis')
ax2=ax1.twinx()
ax2.plot(x,y2,'g',label='log(x)')
ax2.legend(bbox_to_anchor=(1,0.6))
ax2.set_ylabel('Y values for log(x)',color='g')
plt.show()
```

程序运行结果如图 3.5 所示。

用户可以利用 plot()函数绘制简单的图形、图像。

例 3.51　绘制直线示例。

程序代码如下：

```
import matplotlib.pyplot as plt
x=[1,2]                          #横坐标区域
y=[3,6]                          #纵坐标区域
plt.plot(x,y)                    #当前绘图对象进行绘图
plt.show()                       #结果展示
```

例 3.52　绘制折线图示例。

程序代码如下：

```
import matplotlib.pyplot as plt
x=[0,1,2,3,4,5,6]                #横坐标数据
y=[0.3,0.4,2,5,3,4.5,4]          #纵坐标数据
plt.figure(figsize=(8,4))        #创建绘图对象
plt.plot(x,y,'b-',linewidth=1)   #当前对象绘图
plt.show()                       #结果展示
```

程序运行结果如图 3.6 所示。

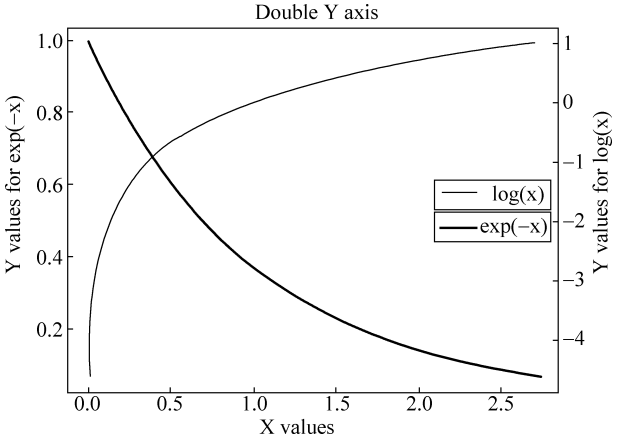

图 3.5　例 3.50 的运行结果

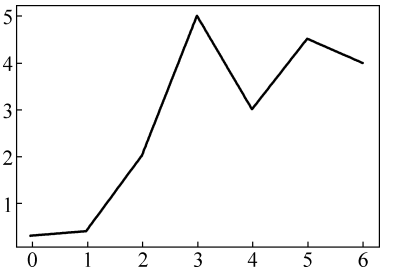

图 3.6　例 3.52 的运行结果

例 3.53　绘制正弦和余弦曲线图。

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x=np.linspace(0,2*np.pi,1000)
y1=np.sin(x)
y2=np.cos(x)
plt.plot(x, y1, ls="-", lw=2, label="sin(x) figure")
plt.plot(x, y2, ls="-", lw=2, label="cos(x) figure")
```

```
plt.legend() #给图加上图例
plt.show()
```

程序运行结果如图 3.7 所示。

2）scatter()函数

用户可以使用 Matplotlib 提供的 scatter()函数绘制散点图(Scatter Diagram)。散点图又称散点分布图，是以一个特征为横坐标，另一个特征为纵坐标，利用坐标点(散点)的分布形态反映特征间统计关系的一种图形。它的值由点在图表中的位置表示，类别由图表中的不同标识表示，通常用于比较不同类别的数据。scatter()函数的常用形式如下：

```
plt.scatter(<x>,<y>[,s=20][,c='b'][,marker='o'])
```

参数说明：

(1) x、y 分别为相同长度的有效坐标数组。

(2) s 给出散点面积大小，可选，默认为 20。

(3) c 颜色或颜色序列，为散点的颜色，可选，值默认为'b'。

(4) marker 表示 MarkerStyle(见表 3.11)，可选，值默认为'o'。

例 3.54 绘制散点图示例。

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.uniform(0,1,200)
y = np.random.uniform(0,1,200)
size = np.random.uniform(0,1,200) * 30
color = np.random.uniform(0,1,200)
plt.scatter(x,y,size,color)
plt.colorbar()
plt.show()
```

程序运行结果如图 3.8 所示。

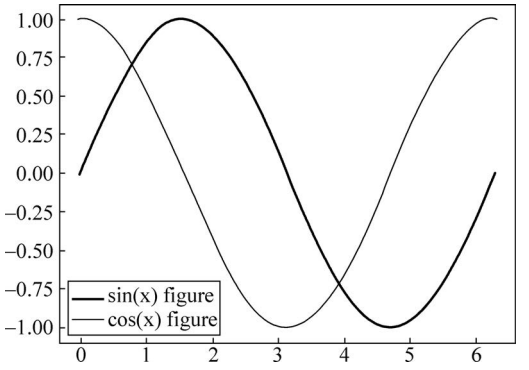

图 3.7 例 3.53 的运行结果

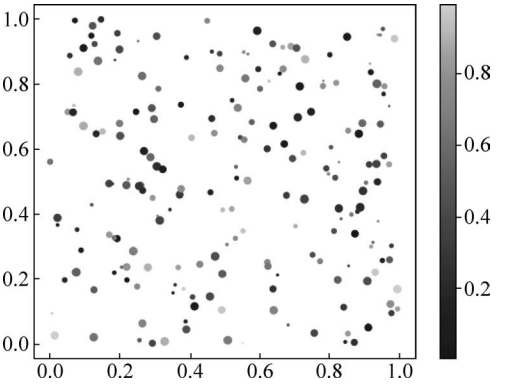

图 3.8 例 3.54 的运行结果

3）bar()函数

用户可以使用 Matplotlib 提供的 bar()函数绘制条形图(Bar Chart)。条形图是一种以长方形长度为变量表达图形的统计报告图，它由一系列高度不等的纵向条纹表示数据分布的情况，用于分析数据内部的分布状态或分散状态，适用于对较小数据集的分析。bar()函数的常用形式如下：

```
plt.bar(<x>,<height>,label='label_string',width=0.8,bottom=None,color='b',edgecolor=
```

```
'b',linewidth = None,orientation = 'vertical')
```

参数说明：

(1) x 表示每一个条形左侧的横坐标。

(2) height 表示每一个条形的高度，它与 x 具有相同长度的有效坐标数组。

(3) label 表示图形标签。

(4) width 表示条形的宽度，取值范围为 0～1，默认为 0.8。

(5) bottom 表示条形的起始位置，默认为 None。

(6) color 表示条形的颜色，取值为'r'、'b'、'g'等，默认为'b'。

(7) edgecolor 表示边框的颜色，取值同上。

(8) linewidth 表示边框的宽度，为 int 类型，单位是像素，默认为 None。

(9) orientation 表示是竖直条还是水平条，取值为'vertical'(竖直条)或'horizontal'(水平条)。

例 3.55　条形图示例。

程序代码如下：

```
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5]
y = [222,42,455,664,454,334]
plt.bar(x,y,0.4,color = 'g')
plt.show() #结果展示
```

例 3.56　利用条形图将表 3.12 中 2017—2023 年度的数据库技术和数据挖掘教材销售数据进行可视化展示。

表 3.12　2017—2023 年度的数据库技术和数据挖掘教材销售数据

年度	2017	2018	2019	2020	2021	2022	2023
数据库技术	58 000	60 200	63 000	71 000	84 000	90 500	107 000
数据挖掘	52 000	54 200	51 500	58 300	56 800	59 500	62 700

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x_data = ['2017','2018','2019','2020','2021','2022','2023',]
y_data1 = [58000,60200,63000,71000,84000,90500,107000]
y_data2 = [52000,54200,51500,58300,56800,59500,62700]
bar_width = 0.3
plt.rcParams['font.family'] = 'STSong' #图形中显示汉字
plt.rcParams['font.size'] = 12 #显示汉字字体
plt.bar(x = np.arange(len(x_data)),height = y_data1,label = '数据库技术',color = 'blue',width =
bar_width)
plt.bar(x = np.arange(len(x_data)) + bar_width,height = y_data2,label = '数据挖掘',color = 'red',
width = bar_width)
for x,y in enumerate(y_data1):
    plt.text(x,y + 100,'%s' % y,ha = 'center',va = 'bottom')
for x,y in enumerate(y_data2):
    plt.text(x + bar_width,y + 100,'%s' % y,ha = 'center',va = 'top')
plt.title('数据库技术与数据挖掘销售对比')
plt.xlabel('2017—2023 年度')
plt.ylabel('销量')
plt.legend()
plt.show()
```

程序运行结果如图 3.9 所示。

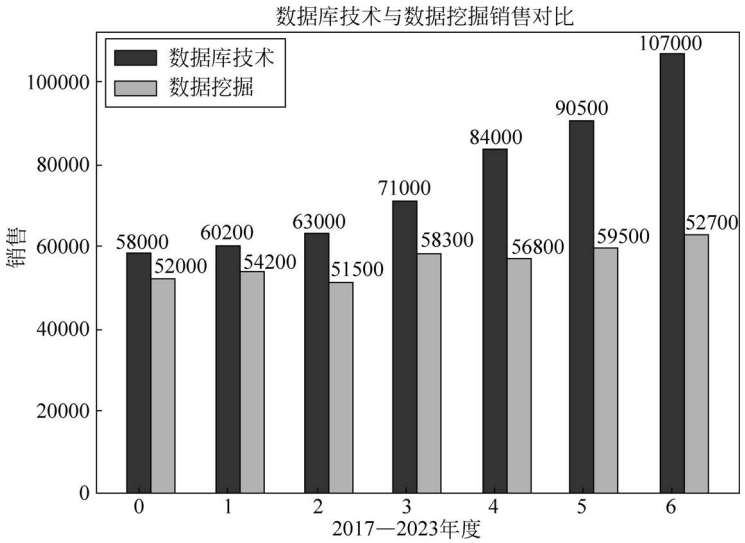

图 3.9　例 3.56 的运行结果

4）pie()函数

用户可以使用 pyplot 模块中的 pie()函数绘制饼图(Pie-graph)，它的功能是沿着逆时针方向排列饼图中的饼形或楔形。饼图是将各项的大小与各项总和的比例显示在一张“饼”中，以“饼”的大小来确定每一项的占比。饼图可以比较清楚地反映出部分与部分、部分与整体之间的比例关系，易于显示每组数据相对于总数的大小，而且显示方式直观。pie()函数的常用形式如下：

```
pie(< x >, labels = None, explode = separation_value, colors = ('b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'), autopct =
None, labeldistance = 1.1, pctdistance = 0.6, shadow = False)
```

参数说明：

(1) x 表示每一块的比例，如果 sum(x)> 1，会使用 sum(x)归一化。

(2) labels 表示每一块饼图外侧显示的说明文字。例如 labels=['苹果','香蕉','橘子','火龙果']。

(3) explode 表示每一块离开中心的距离。

(4) colors 表示每一块显示的颜色。

(5) autopct 表示饼图内的百分比设置，可以使用 format 字符串或者 format function'%3.1f%%'指明小数点前后的位数(没有时用空格补齐)。

(6) labeldistance 表示 label 绘制位置，相对于半径的比例，如小于 1 则绘制在饼图内侧。

(7) pctdistance 类似于 labeldistance，表示 autopct 的位置刻度，radius 控制饼图半径。

(8) shadow 表示是否添加阴影，以增加立体感。

例 3.57　绘制饼图示例。

程序代码如下：

```
import matplotlib.pyplot as plt
fracs = [15, 30.6, 44.4, 10]
A = [0,0.1,0,0]
label = ['A','B','C','D']
plt.pie(x = fracs, autopct = '%3.1f%%', explode = A, labels = label, shadow = True)
plt.show()
```

程序运行结果如图 3.10 所示。

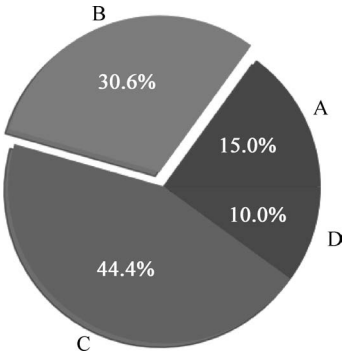

图 3.10　例 3.57 的运行结果

5）hist()函数

用户可以使用 pyplot 模块中的 hist()函数绘制直方图(Histogram)。直方图又称质量分布图，是统计报告图的一种，由一系列高度不等的纵向条纹或线段表示数据分布情况，一般用横轴表示数据所属类别，用纵轴表示数量或者占比。用直方图可以比较直观地看出产品质量特性的分布状态，以便于判断其总体质量的分布情况。通过直方图可以发现分布表无法发现的数据模式、样本的频率分布和总体分布。hist()函数的常用形式如下：

```
hist(< x >, bins = None, range = None, normed = False, align = 'mid',
orientation = 'vertical', color = None)
```

参数说明：

(1) x 表示每个箱子(bin)分布的数据，可以是一个数组或多个数组，对应于 X 轴。

(2) bins 表示箱子(bin)的个数，即共有多少个箱子。

(3) range 表示显示范围，范围以外的将被舍弃，取值为 tuple 或 None。

(4) normed 表示得到的直方图向量是否归一化，为布尔值。

(5) align 表示对齐方式，取值为'left'、'mid'、'right'。

(6) orientation 表示直方图方向，取值为'horizontal'、'vertical'。

(7) color 表示设置的颜色，默认值为 None。

例 3.58　绘制直方图示例。

程序代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.normal(0,1,10000)
n,bins,patches = plt.hist(data,50)
plt.show()
```

程序运行结果如图 3.11 所示。

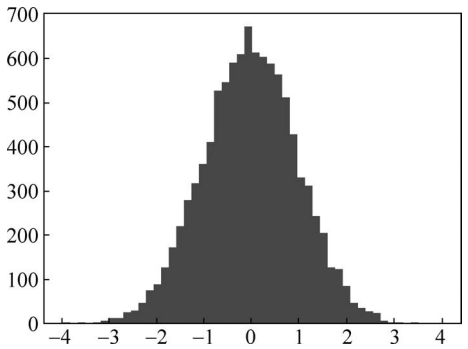

图 3.11　例 3.58 的运行结果

6）boxplot()函数

用户可以使用 boxplot()函数绘制箱线图(Boxplot)。箱线图也称箱须图，其绘制需要使用常用的统计量，能提供有关数据位置和分散情况的关键信息，尤其在比较不同特征时更可表现其分散程度的差异。箱线图利用数据中的 5 个统计量(最小值、下四分位数、中位数、上四分位数和最大值)来描述数据，通过它可以粗略地看出数据是否具有对称性以及分布的分散程度等信息，尤其是用于对几个样本的比较。boxplot()函数的常用形式如下：

```
boxplot(< x >, notch = None, sym = None, vert = None, whis = None, positions = None, widths = None,
boxprops = None)
```

参数说明：

(1) x 表示要绘制箱线图的数据。

(2) notch 表示是否以凹口形式展现箱线图,默认为非凹口。

(3) sym 表示异常点的形状,默认以+号显示。

(4) vert 表示是否需要将箱线图垂直摆放,默认为垂直摆放。

(5) whis 表示上下与上下四分位的距离,默认为 1.5 倍的四分位差。

(6) positions 表示箱线图的位置,默认为[0,1,2,…]。

(7) widths 表示箱线图的宽度,默认为 0.5。

(8) boxprops 表示箱体的属性,例如边框色、填充色等。

例 3.59 绘制箱线图示例。

程序代码如下:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
dt = pd.DataFrame(np.random.rand(4,4))
plt.boxplot(x = dt.values,labels = dt.columns,whis = 1.5)
plt.show()
```

程序运行结果如图 3.12 所示。

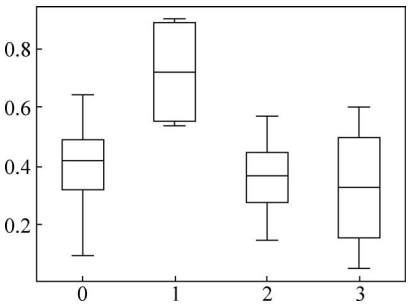

图 3.12 例 3.59 的运行结果

2. pylab 模块

Matplotlib 还提供了一个 pylab 模块,它的语法和 MATLAB 十分相近,主要的绘图命令和 MATLAB 对应的命令有相似参数,pylab 结合了 pyplot 和 NumPy,对交互式使用来说比较方便,既可以画图又可以进行简单的计算。

在实际应用中建议分别导入使用,即:

```
import numpy as np
import matplotlib.pyplot as plt
```

而不是 import pylab as pl(即便使用这一条语句也能满足需要,如例 3.59)。

例 3.60 pylab 绘图示例。

程序代码如下:

```
import pylab as pl
x = range(10)                                    # 横轴的数据
y = [i * i for i in x]                           # 纵轴的数据
pl.rcParams['font.family'] = 'STSong'            # 图形中显示汉字
pl.rcParams['font.size'] = 12                    # 显示汉字字体
pl.plot(x,y,'ob - ',label = 'y = x^2 曲线图')    # 调用 pylab 的 plot()函数绘制曲线
pl.xlabel('X 轴')
pl.ylabel('Y 轴')
pl.legend()
pl.show()
```

3.3.4 文本注解

绘图有时需要在图表中添加文本注解,Python 通过 text()函数在指定的位置(x,y)添加图形内容细节的无指向型注释文本,也可以利用 annotate()函数根据(x,y)坐标完成指向型注释。

1. text()函数

text()函数的常用形式如下:

```
text(< x >,< y >,< string >,weight = 'bold',color = 'b')
```

参数说明：

（1）x 表示注释文本内容所在位置的横坐标。

（2）y 表示注释文本内容所在位置的纵坐标。

（3）string 表示注释文本内容。

（4）weight 表示注释文本内容的粗细风格。

（5）color 表示注释文本内容的字体颜色。

例 3.61　text()函数应用示例。

程序代码如下：

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange( - 10,11,1)
y = x ** 2
plt.plot(x,y)
plt.text( - 3,20,'Function: y = x^2',size = 12)
plt.show()
```

程序运行结果如图 3.13 所示。

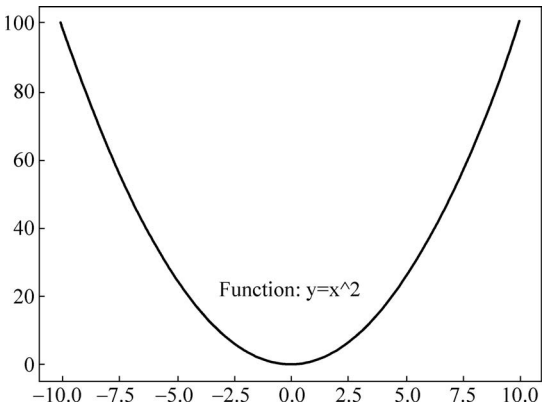

图 3.13　例 3.61 的运行结果

2. annotate()函数

在 Matplotlib 中可以实现用一个箭头指向要注释的地方，再加上一段话的效果。因为要控制的细节比较多，这里仅介绍最简单的实现方法。annotate()函数的简单形式如下：

```
annotate(< text >, xy = xycoords, xytext = note_coordinates, xycoords = coordinates_property,
textcoords = coordinates_property2, arrowprops = arrow_shape)
```

参数说明：

（1）text 表示注释文本的内容。

（2）xy 表示被注释的坐标点，二维元组形如(x,y)。

（3）xytext 表示注释文本的坐标点，也是二维元组，默认与 xy 相同。

（4）xycoords 表示被注释点的坐标系属性，允许输入的值如表 3.13 所示。

（5）textcoords 表示注释文本的坐标系属性，默认与 xycoords 属性的值相同，也可设为 offset points 和 offset pixels 两种属性值，其中 offset points 为相对于被注释点 xy 的偏移量，单位是点；offset pixels 也为相对于被注释点 xy 的偏移量，但单位是像素。

（6）arrowprops 表示箭头的样式，取值主要有'->'、'-['、'|-|'、'-|>'、'<-'、'<->'、'<|-'。

表 3.13 坐标系属性的取值

属 性 值	说 明
figure points	以绘图区的左下角为参考,单位是点
figure pixels	以绘图区的左下角为参考,单位是像素
figure fraction	以绘图区的左下角为参考,单位是百分比
axes points	以子绘图区的左下角为参考,单位是点(一个 figure 可以有多个 axex,默认为一个)
axes pixels	以子绘图区的左下角为参考,单位是像素
axes fraction	以子绘图区的左下角为参考,单位是百分比
data	以被注释的坐标点 xy 为参考(默认值)
polar	不使用本地数据坐标系,使用极坐标系

例 3.62 annotate()函数应用示例。

程序代码如下:

```
import numpy as np
import matplotlib.pyplot as plt
x = np.arange(0,10,0.005) #以步长 0.005 绘制一个曲线
y = np.exp( - x/2.0) * np.sin(2 * np.pi * x)
fig,ax = plt.subplots()
ax.plot(x,y)
ax.set_xlim(0,10)
ax.set_ylim( - 1,1)
#注释点的数据轴坐标和所在的像素
xdata,ydata = 5,0
xdis,ydis = ax.transData.transform_point((xdata,ydata))
#设置注释文本的样式和箭头的样式
bbox1 = dict(boxstyle = 'round',fc = '0.8')
arrow1 = dict(arrowstyle = ' - >')
arrow2 = dict(arrowstyle = ' - >',connectionstyle = 'angle,angleA = 0,angleB = 90,rad = 10')
#用参数 connectionstyle 控制箭头弯曲
offset = 72 #设置偏移量
#xycoords 默认为'data'数据轴坐标,对坐标点(5,0)添加注释
#注释文本参考被注释点设置偏移量,向左 2×72points,向上 72points
ax.annotate('data = ( %.1f, %.1f)' % (xdata,ydata),(xdata,ydata),xytext = ( - 2 * offset,offset),
textcoords = 'offset points',bbox = bbox1,arrowprops = arrow1)
#xycoords 以绘图区的左下角为参考,单位为像素
#注释文本参考被注释点设置偏移量,向左 0.5×72points,向下 72points
disp = ax.annotate('display = ( %.1f, %.1f)' % (xdis,ydis),(xdis,ydis),xytext = (0.5 * offset,
 - offset),xycoords = 'figure pixels',textcoords = 'offset points',bbox = bbox1,arrowprops = arrow2)
plt.show()
```

程序运行结果如图 3.14 所示。

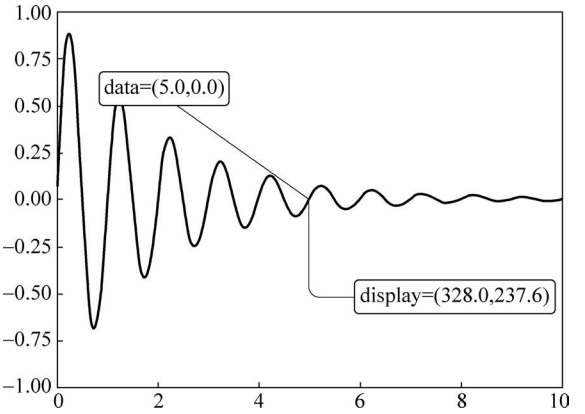

图 3.14 例 3.62 的运行结果

3.4　Scikit-learn

Scikit-learn 简称 sklearn，它是用 Python 实现的机器学习算法库。Scikit-learn 依赖于 Python 的 NumPy、Pandas 和 Matplotlib 库，封装了大量经典以及最新的机器学习模型。作为一款用于机器学习和实践的 Python 第三方开源数据库，Scikit-learn 具备出色的接口设计和高效的学习能力。如果用户已经安装过 NumPy 和 Pandas，安装 Scikit-learn 的最简单方法是使用 Python 包管理工具 pip 直接安装，即在命令提示符下输入 pip install -U scikit-learn。

3.4.1　Scikit-learn 的主要功能

Scikit-learn 包含众多的机器学习算法，主要有六大基本功能，分别是分类、回归、聚类、数据降维、模型选择和数据预处理，如表 3.14 所示。

表 3.14　Scikit-learn 的基本功能

基本功能	说　明	算　法
分类	对给定对象指定所属类别	支持向量机(SVM)、K 最近邻算法(KNN)等
回归	通过建立模型研究因变量和自变量之间的显著关系	向量回归(SVR)、Lasso 回归、贝叶斯回归等
聚类	自动识别具有相似属性的给定对象，并将其分组为簇类	K-均值聚类、分层聚类、DBSCAN 聚类等
数据降维	用来减少随机数个数的方法，主要用在可视化处理、效率提升的应用场景中	主成分分析(PCA)、非负矩阵分解(NMF)等
模型选择	对给定参数和模型进行比较、验证和选择的方法，目的是通过调整参数来提升精度	交叉验证和各种针对预测误差评估的度量函数等
数据预处理	在数据分析处理之前对数据进行的一些处理	数据清理、数据集成、数据归约、数据规范化等

3.4.2　Scikit-learn 自带的小规模数据集

在机器学习过程中需要使用各种各样的数据集，Scikit-learn 内置了一些小规模数据集，小规模数据集包含在 datasets 模块中，大规模数据集需要用户从网络上下载。本书只介绍常用的小规模数据集，如表 3.15 所示。

表 3.15　常用的小规模数据集

名　称	说　明	主要用途	样本×属性
iris	鸢尾花数据集	用于多分类任务的数据集	150×4
boston	波士顿房价数据集	用于回归任务的经典数据集	506×13
breast_cancer	乳腺癌数据集	用于二分类任务的数据集	569×30
linnerud	健身数据集	用于多变量回归的数据集	20×3
wine	红酒数据集	用于分类的数据集	178×13
diabetes	糖尿病数据集	用于回归分析的数据集	442×10
digits	手写数字数据集	用于分类的数据集	1797×64

调用 sklearn 中的小规模数据集首先要导入 datasets 模块，语句形式为 from sklearn import datasets，调用形式为 datasets.load_< name >。其中< name >为被调用的小规模数据集的名称。下面仅介绍常用的 iris、boston 和 digits 小规模数据集的数据结构。

1. iris

iris 数据集有 150 个数据样本，共分为 3 类，每类 50 个样本，每个样本有 4 个特征。

样本包含的 4 个特征分别为 Sepal. Length(花萼长度)、Sepal. Width(花萼宽度)、Petal. Length(花瓣长度)和 Petal. Width(花瓣宽度)。特征值都是正浮点数，单位为厘米。鸢尾花示意图如图 3.15 所示。

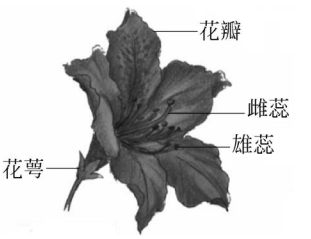

图 3.15　鸢尾花示意图

用户可以通过这 4 个特征预测鸢尾花属于 iris-setosa(山鸢尾)、iris-versicolour(杂色鸢尾)和 iris-virginica(维吉尼亚鸢尾)3 类中的哪一类。

例 3.63　查看 iris 数据集的数据结构示例。

程序代码如下：

```
from sklearn.datasets import load_iris
data = load_iris()                          #加载 iris 数据集
print(data.feature_names)                   #查看数据的特征名
print('*'*80)
print(data.target_names)                    #查看数据的分类名
print('*'*80)
print(data.target)                          #目标标签值
print('*'*80)
print(data.target[[1,10,100]])              #查看第 2、11、101 个样本的目标值
```

2. boston

boston(波士顿房价)数据集包含 506 条数据，每条数据包含房屋以及房屋周围的详细信息，其中包括城镇犯罪率、一氧化氮浓度、住宅平均房间数、到中心区域的加权距离以及自住房平均房价等。boston 数据集的属性描述如表 3.16 所示。

表 3.16　boston 数据集的属性描述

属　性	说　明	属　性	说　明
CRIM	城镇人均犯罪率	DIS	到波士顿 5 个中心区域的加权距离
ZN	住宅用地超过 25 000 平方英尺的比例	RAD	径向高速公路的可达性指数
INDUS	城镇非零售商用土地的比例	TAX	每 10 000 美元的不动产税
CHAS	查尔斯河虚拟变量	PTRATIO	城镇师生比例
NOX	一氧化氮浓度	B	城镇中黑人的比例
RM	住宅平均房间数	LSTAT	人口中地位低收入者的比例
AGE	1940 年之前建成的自用房屋比例		

例 3.64　查看 boston 数据集的数据结构示例。

程序代码如下：

```
from sklearn.datasets import load_boston
boston = load_boston()
print(boston.data.shape)
print(boston.data)
```

3. digits

digits(手写数字)数据集包括 1797 个 0～9 的手写数字数据，每个数字由 8×8 大小的矩阵构成，矩阵中值的范围是 0～16，代表颜色的深度。加载该数据集的语句中有 return_X_y 和 n_class 两个参数。

(1) return_X_y 若为 True，则以(data，target)形式返回数据。其默认为 False，表示以字

典形式返回数据的全部信息(包括 data 和 target)。

(2) n_class 表示返回数据的类别数,default=10,例如 n_class=5,返回 0 到 4 的数据样本。

例 3.65　查看 digits 数据集的数据结构示例。

程序代码如下:

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
data = load_digits(n_class = 5,return_X_y = False)
print(data.target[0:10]) #查看第 1～10 个样本的目标值
print(data.data)
print('*'*80)
print(data.feature_names)
print('*'*80)
print(data.target_names)
print('*'*80)
print(data.target)
print('*'*80)
print(data.target[[2,20,200]]) #查看第 3、21、201 个样本的目标值
print('*'*80)
print(data.images.shape) #查看数字图形的形式
print('*'*80)
plt.matshow(data.images[1]) #查看数字图像
plt.show()
```

3.4.3　使用 Scikit-learn 生成数据集

在机器学习算法中经常需要数据集来验证算法、调试参数,但是找到一组十分适合某种特定算法类型的数据样本不容易。除 NumPy 模块提供了随机数生成的功能外,使用 Scikit-learn 也可以生成指定模式和复杂形状的数据样本集。

1. 生成聚类和分类数据集

使用 make_blobs()和 make_classification()都可以创建多个类别的数据集,但 make_blobs()主要用来创建聚类的数据集,对每个聚类的中心、标准差都能很好地控制; make_classification()通过冗杂、无效的特征等方法引入噪声,用来创建分类的数据集。

1) make_blobs()函数

make_blobs()的常用形式如下:

```
X,y = make_blobs(n_features = 2,n_samples = 100,centers = 3,random_state = num_value,cluster_std = [0.8,2,5])
```

参数说明:

(1) n_features 表示每个样本的特征(或属性)数,也是数据的维度,默认值为 2。

(2) n_samples 表示数据样本点的个数,默认值为 100。

(3) centers 表示类别数(标签的种类数),默认值为 3。

(4) random_state 表示随机生成器的种子,给定数值之后,每次生成的数据集就是固定的;若不给定数值,则由于随机性将导致每次运行程序所获得的结果可能有所不同。

(5) cluster_std 表示每个类别的方差,例如希望生成两类数据,其中一类比另一类具有更大的方差,可以将 cluster_std 设置为[1.0,3.0],其默认值为 1.0。

返回值:

(1) X 为产生的相应数据集。

(2) y 为产生的相应类别标签。

例 3.66 make_blobs()函数应用示例。

程序代码如下：

```
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
data,target = make_blobs(n_samples = 100,n_features = 2,centers = 3)
#在二维图中绘制数据样本,每个样本的颜色不同
plt.scatter(data[:,0],data[:,1],c = target);
plt.show()
```

程序运行结果如图 3.16 所示。

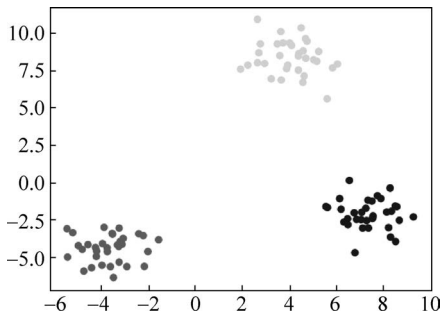

图 3.16 例 3.66 的运行结果

2) make_classification()函数

make_classification()函数的常用形式如下：

```
X, y = make_classification(n_samples = 100,
n_features = 2,n_informative = 2,n_redundant = 0,
n_classes = 2,n_clusters_per_class = 2,weights =
None)
```

参数说明：

(1) n_samples 为 int 类型或数组，如果是数组，则序列中的每个元素表示相应聚簇的样本数量，该参数可选，默认值为 100。

(2) n_features 为 int 类型，表示每个数据样本的特征(或属性)数量，该参数可选，默认值为 2。

(3) n_informative 表示多信息特征(或属性)的个数。

(4) n_redundant 表示冗余信息，informative 特征的随机线性组合。

(5) n_classes 表示分类类别数。

(6) n_clusters_per_class 表示某一个类别是由几个 cluster 构成的。

(7) weights 表示列表类型，权重比，默认值为 None。

返回值：

(1) X 为产生的相应数据集。

(2) y 为产生的相应类别标签。

例 3.67 make_classification()函数应用示例。

程序代码如下：

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
#X 为样本特征,y 为样本类别输出,共 400 个样本,每个样本两个特征,输出有 3 个类别,没有冗余特
#征,每个类别一个簇
X,y = make_classification(n_samples = 400,n_features = 2,n_redundant = 0,n_clusters_per_class =
1,n_classes = 3)
plt.scatter(X[:,0],X[:,1],marker = 'o',c = y)
plt.show()
```

程序运行结果如图 3.17 所示。

2. 生成环形形状和月亮形状数据集

make_circles()函数和 make_moons()函数在二维图中分别创建一个大圆包含小圆的环形形状和月亮形状的数据样本集，用于可视化聚类和分类算法。

1）make_circles()函数

该函数的常用形式为：

```
X,y = make_circles(n_samples = 100,factor = 0.8,noise = None)
```

参数说明：

（1）n_samples 表示生成的样本数量，默认值为 100。

（2）factor 为(0,1)的浮点数，默认值为 0.8，表示内、外圆之间的比例因子。

（3）noise 表示样本的随机噪声。

返回值：

（1）X 为产生的相应数据集。

（2）y 为产生的相应类别标签。

例 3.68　make_circles()函数应用示例。

程序代码如下：

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_circles
# 生成环形形状数据集
X,y = make_circles(n_samples = 1000,factor = 0.6,noise = 0.1)
plt.scatter(X[:,0],X[:,1],c = y,s = 25)
plt.show()
```

程序运行结果如图 3.18 所示。

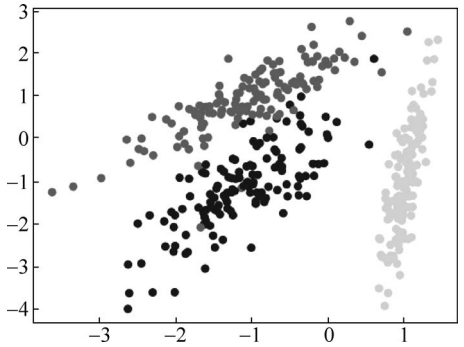

图 3.17　例 3.67 的运行结果

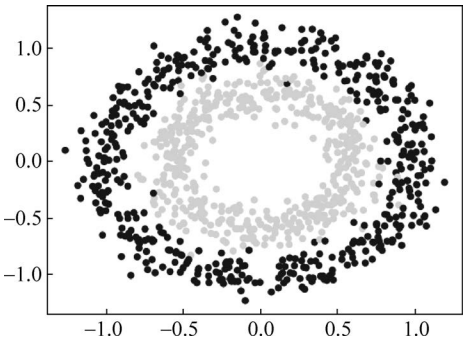

图 3.18　例 3.68 的运行结果

2）make_moons()函数

该函数的常用形式为：

```
X,y = make_moons(n_samples = 100,shuffle = True,noise = None)
```

参数说明：

（1）n_samples 为 int 类型，表示产生总样本的数量，该参数可选，默认值为 100。

（2）shuffle 为 boolean 类型，表示是否对数据样本进行重新洗牌，该参数可选，默认值为 True。

（3）noise 为 float 类型，表示增加到数据里面的高斯噪声标准差，默认值为 None。

返回值：

（1）X 为产生的相应数据集。

（2）y 为产生的相应类别标签。

例 3.69　make_moons()函数应用示例。

程序代码如下：

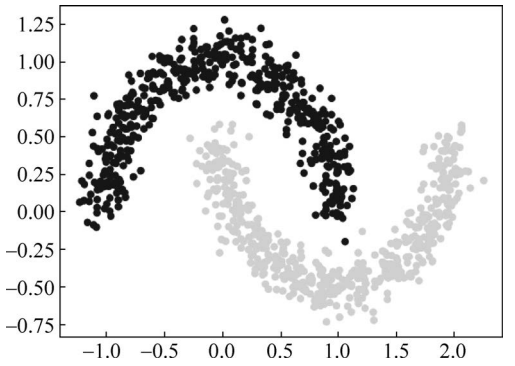

图 3.19　例 3.69 的运行结果

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_moons
#生成月亮数据集
X,y = make_moons(n_samples = 1000,noise = 0.1)
plt.scatter(X[:,0],X[:,1],c = y,s = 25)
plt.show()
```

程序运行结果如图 3.19 所示。

3. make_gaussian_quantiles()函数

make_gaussian_quantiles()函数可以随机生成多类别的多维正态分布数据集。其常用形式如下：

```
X,y = make_gaussian_quantiles(n_samples = 100,
n_features = 2,n_classes = 4,mean = [1,2],cov = 2)
```

参数说明：

(1) n_samples 表示生成数据样本集的样本数，默认为 100。

(2) n_features 表示正态分布的维数，默认为 2。

(3) n_classes 表示数据在正态分布中按分位数分配的类别数，默认为 4。

(4) mean 表示特征均值。

(5) cov 表示样本协方差的系数。

返回值：

(1) X 为产生的相应数据集。

(2) y 为产生的相应类别标签。

例 3.70　make_gaussian_quantiles()函数应用示例。

程序代码如下：

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_gaussian_quantiles
#生成二维正态分布,生成的数据按分位数分成 3 组,1000 个样本,样本特征均值为 1 和 2,协方差系数
#为 2
X,y = make_gaussian_quantiles(n_samples = 1000, n_features = 2, n_classes = 3, mean = [1,2],cov = 2)
plt.scatter(X[:,0],X[:,1], marker = 'o', c = y)
plt.show()
```

程序运行结果如图 3.20 所示。

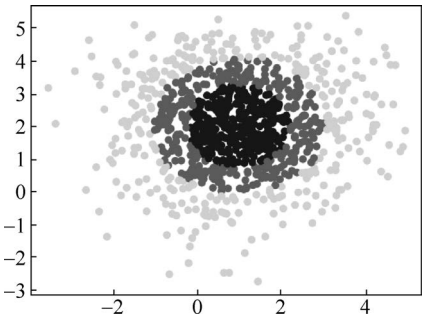

图 3.20　例 3.70 的运行结果

4. make_regression()函数

使用 make_regression()函数可以生成回归模型数据集，其常用形式如下：

```
X,y,coef = make_regression(n_samples = num_value1,n_features = num,noise =  num_value2,coef =
True)
```

参数说明：

(1) n_samples 表示生成的样本个数。

(2) n_features 表示样本的特征(或属性)个数。

(3) noise 表示样本的随机噪声。

(4) coef 表示是否返回回归系数。

返回值：

(1) X 为产生的相应数据集。

(2) y 为产生的相应类别标签。

例 3.71　make_regression()函数应用示例。

程序代码如下：

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_regression
X, y, coef = make_regression(n_samples = 1000, n_features = 1, noise = 20, coef = True) # 共 1000
                                                                                    # 个样本,每个样本一个特征
plt.scatter(X, y, color = 'black')
plt.plot(X, X * coef, color = 'blue',linewidth = 3)
plt.show()
```

程序运行结果如图 3.21 所示。

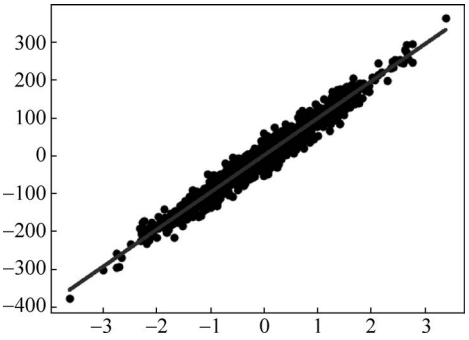

图 3.21　例 3.71 的运行结果

3.5　股票数据的简单分析

分析股票数据可以了解股市的大盘走势，进行股票行情分析，以掌握股票的涨跌趋势，提供股票解套方案，提示股票买卖关键点。

3.5.1　抓取股票数据

获取股票数据是股票数据分析中必不可少的一部分，而网络爬虫是获取股票数据的一个重要渠道，该部分获取的是证券之星网站上某网页中的 A 股数据。该程序主要分为 3 个部分，即网页源代码的获取、所需内容的提取、所得结果的整理。程序代码如下：

```
import urllib.request
import re
import csv
import os
url = 'http://quote.stockstar.com/stock/ranklist_a_3_1_1.html'      # 目标网址
```

```
headers = {"User - Agent":"Mozilla/5.0 (Windows NT 10.0; WOW64)"}    #伪装浏览器请求报头
request = urllib.request.Request(url = url,headers = headers)    #请求服务器
response = urllib.request.urlopen(request)                        #服务器应答
content = response.read().decode('gbk')                           #以一定的编码方式查看源代码
print(content)                                                    #打印页面源代码
pattern = re.compile('<tbody[\s\S]*</tbody>')
body = re.findall(pattern,str(content))    #匹配<tbody>和</tbody>之间的所有代码
pattern = re.compile('>(.*?)<')
stock_page = re.findall(pattern,body[0])                          #匹配>和<之间的所有信息
stock_total = stock_page
stock_last = stock_total[:]      #stock_total: 匹配出的股票数据
for data in stock_total:         #stock_last: 整理后的股票数据
    if data == '':
        stock_last.remove('')
head = ['代码','简称','最新价','涨跌幅','涨跌额','5 分钟涨幅']
lst = []
for i in range(0,len(stock_last),6):                              #网页中共有 6 列数据
    lst.append([stock_last[i],stock_last[i+1],stock_last[i+2], stock_last[i+3],stock_last
[i+4],stock_last[i+5]])
    os.chdir('D:\\Data_Mining\gupiaoshujufenxi')                  #改变当前路径
    with open ('test.csv', 'a', newline = '') as f :              #以追加方式打开或创建
        f_csv = csv.writer(f)
        f_csv.writerow(head) #写入文件头
        for i in range(len(lst)): #按行写入文件
            f_csv.writerow(lst[i])
```

程序运行结果是在指定路径 D:\\Data_Mining\gupiaoshujufenxi 下创建了数据文件 test.csv，文件的部分数据如图 3.22 所示。

图 3.22　获取证券之星网站上某天的所有 A 股数据

3.5.2　股票数据的各指标折线图

以顺序号为横坐标，最新价、涨跌幅、涨跌额为纵坐标，绘制折线图，可以观察各股票的大致情况。程序代码如下：

```
import pandas as pd
import matplotlib.pyplot as plt
import os
```

```
os.chdir('D:\\Data_Mining\gupiaoshujufenxi')          #设置当前路径
data = pd.read_csv('test.csv',encoding = 'gb18030')   #读取文件数据
plt.rcParams['font.family'] = 'STSong'                #图形中显示汉字字体
plt.rcParams['font.size'] = 12                        #显示汉字字号
data['最新价'].plot(grid = True)
data['涨跌幅'].plot(grid = True)
data['涨跌额'].plot(grid = True)
plt.legend(['最新价','涨跌幅','涨跌额'])
plt.show()
```

程序运行结果如图 3.23 所示。

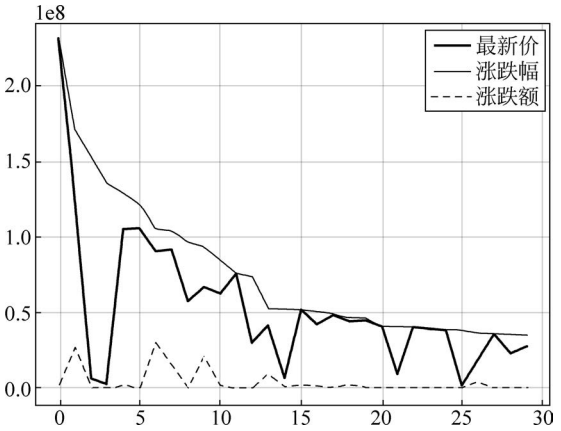

图 3.23　以顺序号为横坐标，最新价、涨跌幅、涨跌额为纵坐标的折线图

3.5.3　各股票的 5 分钟涨幅柱状图

由于篇幅原因，本书设计每 6 股股票为一张图。程序代码如下：

```
import pandas as pd
import matplotlib.pyplot as plt
import os
os.chdir('D:\\Data_Mining\gupiaoshujufenxi')          #设置当前路径
data = pd.read_csv('test.csv',encoding = 'gb18030')   #读取文件数据
plt.rcParams['font.family'] = 'STSong'                #图形中显示汉字字体
plt.rcParams['font.size'] = 12                        #显示汉字字号
lst = []
lst1 = []
for i in range(0,len(data['简称']) - 6,6):
    lst.append([data['简称'][i + 0],data['简称'][i + 1],data['简称'][i + 2],data['简称'][i + 3],
data['简称'][i + 4],data['简称'][i + 5]])
    lst1.append([data['5 分钟涨幅'][i + 0],data['5 分钟涨幅'][i + 1],data['5 分钟涨幅'][i + 2],
data['5 分钟涨幅'][i + 3],data['5 分钟涨幅'][i + 4],data['5 分钟涨幅'][i + 5]])
for i in range(len(lst)):
    x = range(len(lst[i]))
    y = lst1[i]
    plt.bar(x,y,width = 0.3)
    plt.xticks(x,lst[i])
plt.title('第' + str(i + 1) + '张图')
plt.legend(['5 分钟涨幅'])
plt.show()
```

程序运行结果如图 3.24 所示。

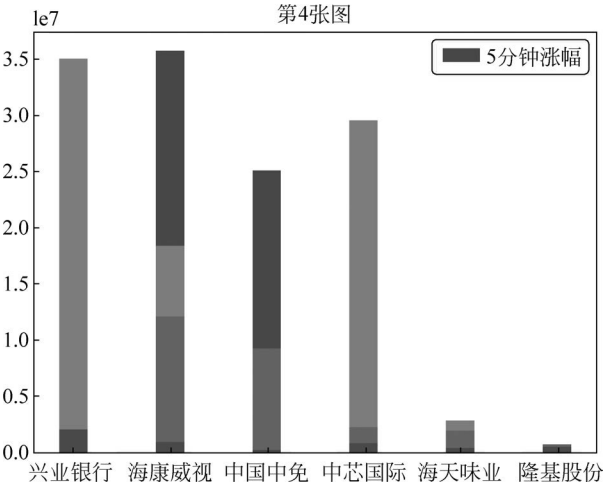

图 3.24　各股票 5 分钟涨幅柱状图

3.5.4　股票各指标之间的关系对比

选取股票部分具有代表性的指标，使用 pd.plotting.scatter_matrix()函数将各项指标数据两两关联做散点图，对角线是每个指标数据的直方图。程序代码如下：

```
import pandas as pd
import matplotlib.pyplot as plt
import os
os.chdir('D:\\Data_Mining\gupiaoshujufenxi') #设置当前路径
data = pd.read_csv('test.csv',encoding = 'gb18030') #读取文件数据
plt.rcParams['font.family'] = 'STSong' #图形中显示汉字字体
plt.rcParams['font.size'] = 12 #显示汉字字号
small = data[['最新价', '涨跌幅', '涨跌额','5 分钟涨幅']]
st = pd.plotting.scatter_matrix(small)
plt.show()
```

程序运行结果如图 3.25 所示。

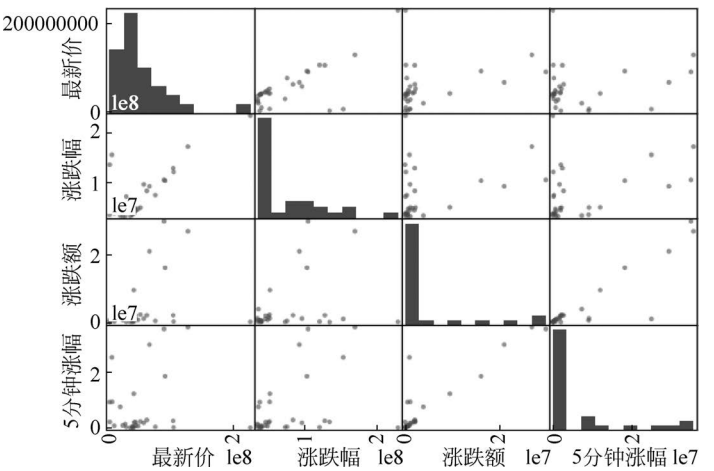

图 3.25　股票各指标之间关系的对比图

扫一扫

自测题

习题 3

3-1　选择题：

(1) 创建一个 3×3 零矩阵的 Python 语句为(　　)。

A. np.mat(np.zeros(3,3))　　B. np.mat(np.zeros(3))

C. np.mat(np.zeros((3,3)))　　D. np.mat(np.zeros((3)))

(2) 生成一个 3×3 的 0～10 的随机整数矩阵的 Python 语句为(　　)。

A. np.random.randint(10,size=(3,3))

B. np.random.randint([0,10],size=(3,3))

C. np.mat(np.random.randint(10,size=(3,3)))

D. np.mat(np.random.randint([0,10],size=(3,3)))

(3) Python 语句 np.mat(np.random.randint(2,8,size=(2,5)))执行后的结果是(　　)。

A. 产生一个 2～8 的 2×5 的随机整数矩阵

B. 产生一个 2～5 的 2×8 的随机整数矩阵

C. 产生一个 2～8 共有 2×5 个整数的随机数序列

D. 产生一个 2～5 共有 2×8 个整数的随机数序列

(4) 执行下列 Python 语句序列后，使得 a 和 b 进行相应元素相乘的运算是(　　)。

```
import numpy as np
a = mat([[12,3],[2,14]])
b = np.mat([[1,1],[0,0]])
```

A. np.dot(a,b)　　B. np.multiply(a,b)

C. a * b　　D. a · b

(5) Python 语句 np.random.uniform([1,5],[5,10])执行后的结果为(　　)。

A. 生成 1～5 和 5～10 的二维整型数组

B. 生成 1～5 和 5～10 的二维浮点型数组

C. 生成 1～5 和 5～10 的两个整型数的一维数组

D. 生成 1～5 和 5～10 的两个浮点型数的一维数组

(6) Pandas 利用 read_excel()方法读取 Excel 文件数据后返回(　　)对象。

A. Excel　　B. DataFrame　　C. CSV　　D. Series

(7) 下列不属于 Scikit-learn 处理的功能是(　　)。

A. 数据降维　　B. 模型选择　　C. 数据预处理　　D. 数据可视化

(8) Scikit-learn 模块自带的 iris(鸢尾花)小规模数据集(　　)。

A. 有 150 个数据样本，共分为 3 类，每个样本有 4 个特征

B. 有 50 个数据样本，共分为 3 类，每个样本有 4 个特征

C. 有 150 个数据样本，共分为 5 类，每个样本有 4 个特征

D. 有 150 个数据样本，共分为 3 类，每个样本有两个特征

(9) 饼图利用圆形及圆内扇形的(　　)表示数值大小。

A. 面积　　B. 弧线长度　　C. 角度　　D. 颜色

(10) 条形图利用宽度相同的条形的(　　)表述数据多少的图形。

A. 面积　　B. 高度或长度　　C. 频数　　D. 类别

3-2 填空题：

(1) NumPy 是高性能科学计算和(　　)的基础包。

(2) Python 语句 np.random.randint(5,10,size=6)生成(　　)的 6 个元素的数组。

(3) 在使用字符串、列表、数组和矩阵的过程中经常需要相互转换，可以用(　　)函数查看对象的类型。

(4) 执行下列 Python 语句序列后输出(　　)。

```
import numpy as np
arr = np.arange(1 2).reshape([3,4])
    print(arr[::2,1])
```

(5) 通过(　　)或者属性的方式可以单独获取 DataFrame 的列数据，返回数据的类型为 Series。

(6) 在 DataFrame 下查询单独的几行数据可以采用 Pandas 提供的(　　)和 loc()方法实现。

(7) 在 DataFrame 下对值进行排序的时候，无论是升序还是降序，缺失值(NaN)都会排在(　　)。

(8) CSV 是最通用的一种文件格式，它可以被非常容易地导入各种计算机表格及(　　)中。

(9) Scikit-learn 模块的波士顿房价数据集包含(　　)条数据，每条数据包含房屋以及房屋周围的详细信息。

3-3 简述 map()、apply()和 mapapply()三者在用法上的区别。

3-4 利用 plot()函数绘制二次函数 $y=2x^2-3x+1$ 在[−2,3.5]上的图像。

3-5 2022 年 4 月某国总统大选第一轮投票结果如表 3.17 所示，用饼图展现得票结果，并将最高得分显著展示出来。

表 3.17 2022 年 4 月某国总统大选第一轮投票结果

候选人	马西龙	勒庞让	吕克里	埃里克	瓦莱丽	卓丽雅
得票率/%	29.6	24.5	22.4	12.3	7.9	3.2

3-6 运行以下程序，解释程序运行结果。

```
from sklearn.datasets import make_circles
from sklearn.datasets import make_moons
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure(1)
x1,y1 = make_circles(n_samples = 1000,factor = 0.5,noise = 0.1)
plt.subplot(121)
plt.title('make_circles function example')
plt.scatter(x1[:,0],x1[:,1],marker = 'o',c = y1)
plt.subplot(122)
x1,y1 = make_moons(n_samples = 1000,noise = 0.1)
plt.title('make_moons function example')
plt.scatter(x1[:,0],x1[:,1],marker = 'o',c = y1)
plt.show()
```