

Python 数据分析实验指导

目录

Python 数据分析实验指导	1
实验一 Python 语言基础实验	3
实验二 程序控制结构实验	11
实验三 函数实验	15
实验四 正则表达式和文件操作实验	18
实验五 数据可视化实验	23
实验六 numpy 库实验	28
7.1 ndarray 多维数组	28
7.1.1 创建 ndarray 数组	28
7.1.2 创建特殊的 ndarray 数组	29
7.1.3 ndarray 对象的数据类型	34
7.1.4 ndarray 数组对象的属性	34
7.2 数组元素的索引、切片和选择	35
7.2.1 索引和切片	35
7.2.2 选择数组元素的方法	36
7.2.3 ndarray 数组的形状变换	40
7.3 随机数数组	42
7.3.1 简单随机数	42
7.3.2 随机分布	43
7.3.3 随机排列	45
7.3.4 随机数生成器	46
7.4 数组的运算	47
7.4.1 算术运算与函数运算	47
7.4.2 统计计算	50
7.4.3 线性代数运算	52
7.4.4 排序	55
7.4.5 数组拼接与切分	56
7.5 读写数据文件	58
7.5.1 读写二进制文件	58
7.5.2 读写文本文件	58
实验七 pandas 库实验	60
8.1 Series 对象	60
8.1.1 Series 对象创建	60
8.1.2 Series 对象的属性	61
8.1.3 Series 对象的数据的查看和修改	62
8.2 Series 对象的基本运算	63
8.2.1 算术运算与函数运算	63
8.2.2 Series 对象之间的运算	65
8.3 DataFrame 对象	66

8.3.1 DataFrame 对象创建	66
8.3.2 DataFrame 对象的属性	68
8.3.3 查看和修改 DataFrame 对象的元素	70
8.3.4 判断元素是否属于 DataFrame 对象	72
8.4 DataFrame 对象的基本运算	73
8.4.1 数据筛选	73
8.4.2 数据预处理	75
8.4.3 数据运算与排序	91
8.4.4 数学统计	97
8.4.5 数据分组与聚合	103
8.5 pandas 数据可视化	111
8.5.1 绘制折线图	111
8.5.2 绘制条形图	113
8.5.3 绘制直方图	114
8.5.4 绘制箱线图	115
8.5.5 绘制区域图	116
8.5.6 绘制散点图	117
8.5.7 绘制饼状图	118
8.6 pandas 读写数据	119
8.6.1 读写 csv 文件	119
8.6.2 读取 txt 文件	122
8.6.3 读写 Excel 文件	123
8.7 筛选和排序数据实例	126
实验八 数据预处理实验	129
10.1 数据清洗	129
10.1.1 处理缺失值	129
10.2 数据集成	136
10.3 数据规范化	136
10.3.1 最小-最大规范化	136
10.3.2 z 分数规范化	137
10.3.3 小数定标规范化	138
10.4 数据离散化	138
10.4.1 无监督离散化	138
10.4.2 监督离散化	138
10.5 数据归约	138
10.5.1 过滤法	138
10.5.2 包装法	141
10.5.3 嵌入法	141
10.6 数据降维	142
10.6.1 主成分分析	142
10.6.2 线性判别分析法	142
10.7 数据预处理举例	143
实验九 数据分析方法实验	152

实验一 Python 语言基础实验

一、实验目的

掌握 Python 的编辑器安装和使用； Python 运算符与表达式及常用 Python 内置函数和模块的导入与使用； Python 代码的编写规范。掌握：列表、元组、字典、集合等数据结构的异同以及对它们访问、切片、计算等。

二、实验过程

1. Python 固定语法

```
hello_world = 'hello world'    # 变量名无需提前声明
print(hello_world)
```

```
'''
```

```
hello_world = 'hello world'    # 变量名无需提前声明
print(hello_world)
```

```
hello_world = 'hello world'    # 变量名无需提前声明
print(hello_world)
```

```
'''
```

```
机器学习 = ['决策树', '神经网络', '聚类分析']
```

```
for i in 机器学习:
```

```
    print(i)
```

```
    print('hello world')
```

2. 字符串与数值

```
# 创建字符串
```

```
string1 = 'Python1'
```

```
string2 = "Python2"
```

```
string3 = """1.35
```

```
Python3
```

```
Python4
```

```
2.44"""
```

```
print(string3)
```

字符串的基本操作

string1 + string2 # 合并字符串

string1 * 3 # 复制字符串

int('9') # 将字符串转化成数值

字符串的索引及切片操作

print(string1)

string1[0] # 正序索引，序号从 0 开始

string1[-7] # 逆序索引，序号从-1 开始

string1[1]

string1[1:3] # 字符串的切片操作，切片时左闭右开

string1[:3]

string1[3:]

任务实现

1.创建一个字符串变量“Apple's unit price is 9 yuan.”。

applePriceString = "Apple's unit price is 9 yuan."

2.提取出里面的数字 9 并赋值给新的变量。

applePrice = applePriceString[-7]

3.查看新变量的数据类型。

type(applePrice)

4.将提取的数字 9 转成整型（int）。

applePriceInt = int(applePrice)

5.确认数据类型是否转换成功。

type(applePriceInt)

3. 计算圆形的各参数

任务实现 1：给定圆的半径，计算圆的周长和面积

pi = 3.14

r = 3

C = 2 * pi * r # 计算圆周长

S = pi * r ** 2 # 计算圆的面积

任务实现 2: 给定圆的周长, 计算圆的半径和面积

C = 5

r = C / (2 * pi)

S = pi * r ** 2

任务实现 3: 给定圆的面积, 计算圆的半径和周长

S = 5

r = (S/pi)**0.5

C = 2 * pi * r

4. 创建列表并进行增删改查操作

创建列表

all_in_list = [0.25, 'hello', True, [2.3, 1.5]]

list_example = list('ABCD')

print(all_in_list)

列表的索引及切片

print(all_in_list)

all_in_list[1] # 索引操作

all_in_list[-3]

all_in_list[0:3] # 切片操作

all_in_list[-4:-1]

all_in_list[:3]

all_in_list[:]

all_in_list[0:1] # 注意, 切片操作返回的值会保留原来的数据结构

为列表新增元素

print(all_in_list)

all_in_list.append(0.78) # 在列表末尾追加一个元素

all_in_list.append([1, 2]) # 将待添加元素作为一个整体追加至目标列表的末尾

all_in_list.extend([1, 2]) # 将待添加元素的各个元素分别追加至目标列表的末尾

```

all_in_list.insert(1, 'world')    # 在指定位置插入相应元素
[1, 2] + [3, 4]                  # 将两个列表中的元素进行合并

# 删除列表中的元素
print(all_in_list)
all_in_list.remove('hello')      # 删除列表中的指定元素
del all_in_list[0:2]              # 删除列表中的多个元素
del all_in_list                  # 删除列表本身

# 修改列表中的元素
all_in_list = [0.25, 'hello', True, [2.3, 1.5]]
all_in_list[0] = 125             # 通过赋值来修改列表中的元素

# 列表推导式
x = []                           # 构建一个空列表
for i in range(1, 11):
    x.append(i)
print(x)
X_new = [i for i in range(1, 11)] # 通过列表推导式来构建一个具有特定规则的列表
print(X_new)
print([i**2 for i in range(1, 11)])

# 求解曲边图形的面积
import math
n = 10000                        # 划分的小矩形个数
# 1.将图形等份划分，得到若干小矩形（构建 x 序列）。
width = 2 * math.pi / n        # 每个小矩形的宽度
# x = [0*width, 1*width, ...(n-1)*width]
x = [i * width for i in range(0, n)] # x 序列
# 2.求出各小矩形的面积。
s = [abs(math.sin(i)) * width for i in x]
# 3.最后求和。

```

sum(s)

5. 创建字典并进行增删改查操作

创建字典

```
dict_first = {  
    'the': 2,  
    3.4: [3.5, 4],  
    'hello': 'world'  
}
```

字典的增删改查操作

dict_first['the'] # 通过键来索引对应的值

dict_first['the'] = 101 # 修改字典中的值

dict_first['world'] = 2.5 # 通过赋值来新增键值对

print(dict_first)

dict_first.update({'hello world': 3, 4.5: [2.3, 1.2]})

del dict_first['world'] # 删除字典中的指定的键值对

print(dict_first[3.4])

dict_first.keys() # 访问字典的所有键

dict_first.values() # 访问字典的所有值

dict_first.items() # 访问字典的所有元素

second_dict = {i: i ** 2 for i in range(1, 11)} # 字典推导式

任务实现：统计单词词频

lyric = 'The night begin to shine, the night begin to shine'

lyric = lyric.lower() # 将所有字母转为小写形式

words = lyric.split() # 将句子拆分成多个单词

word_freq = {} # 构建一个空字典,用于后续记录各单词的频次

for word in words:

```

        if word in word_freq.keys():      # 判断当前访问的单词是否在字典中
            word_freq[word] += 1          # 若在，则将该单词对应键的值加一
        else:
            word_freq[word] = 1           # 若不在则以该单词为键创建一个键值对，且赋值为1

```

6. 实现一组数的连加与连乘

```

# for 循环语句
for i in range(10):
    print(i)      # 代码块（循环体）通过缩进进行限制
print(0.001)

```

```

s = 0
while s < 10:
    print(s)
    s += 1

```

```

# 任务实现
# 实现一组数的连加操作
vec = list(range(1, 11))    # 创建列表
m = 0
for i in vec:
    m += i
print(m)

```

```

# 实现一组数的连乘操作
x = list(range(1, 11))
n = 1
for i in x:
    n *= i
print(n)

```

7. 考试成绩等级划分

```

# 条件判定语句
if 1 < 2:

```



```

    print('hello')

if 1 > 2:
    print('hello')
else:
    print('hello world')

# 多路分支语句
if 1 > 2:
    pass
elif 2 < 3:
    print('world')
else:
    print('hello')

try:
    print(hello)
except:
    print('hello')

# 任务实现
# 1.创建一个变量，输入任意数值作为成绩并赋予该变量。
# 2.检测输入的内容是否为数值型的数据。
# 3.设置条件分支判断成绩属于哪个等级。
# 4.打印结果。

try:
    score = input('请输入考试成绩: ')
    score = float(score)    # 将数据转为浮点类型
    if score >= 90:
        print('A')
    elif 80 <= score < 90:

```

```

        print('B')
    elif 70 <= score < 80:
        print('C')
    elif 60 <= score < 70:
        print('D')
    else:
        print('E')
except:
    print('输入的成绩是非数值型的！')

```

8. 冒泡排序法排序

```

# 使用冒泡排序法对指定序列进行排序
# 1.创建一个列表对象[1,2,6,0.3,2,0.5,-1,2.4] 。
vec = [1,2,6,0.3,2,0.5,-1,2.4]
# 2.编写嵌套循环，外循环 i 的取值为 0 到列表对象的长度，内循环 j 的取值
    为 0 到 i。
for i in range(len(vec)):
    for j in range(i):
        if vec[j] > vec[i]:
            vec[j], vec[i] = vec[i], vec[j]    # 3.当遍历的列表对象的前一个
            元素比后一个元素小时，两个元素的位置互换。
# 4.打印结果。
print(vec)

```

实验二 程序控制结构实验

一、实验目的

掌握 if 选择、for 循环、while 循环；range 对象在循环中的使用，成员测试符 in 在循环语句中的使用。熟悉 break 和 continue 语句的作用。

二、实验过程

1.

```
import random
lottery = random.randint(100,999)
guess=eval(input("输入你想要的三位彩票号码: "))
lotteryD1 = lottery//100
lotteryD2 = (lottery//10)%10
lotteryD3 = lottery%10
guessD1 = guess//100
guessD2 = (guess//10)%10
guessD3 = guess%10
print("开奖号码是: ",lottery)
if guess== lottery:
    print("号码完全相同: 奖金为 3000 美元")
elif (lotteryD1==guessD1 and lotteryD2==guessD2) or (lotteryD3==guessD3 and
lotteryD2==guessD2):
    print("有两位号码连着相同: 奖金为 2000 美元")
elif len((set(str(lottery))&set(str(guess))))==1:
    print("有一号码相同: 奖金为 500 美元")
else:
    print("对不起, 这次没中奖! ")
```

2.

```
x=input("请输入用户名:")
y=input("请输入密码:")
z=input("请输入性别('男' or '女'):")
if y=="Python3.6.0":
    if z=="男":
        print("祝贺你, %s 先生, 你已成功登录!"%x)
    if z=="女":
        print("祝贺你, %s 女士, 你已登录成功!"%x)
else:
    print("对不起, 密码错误, 登录失败!")
```

3.

```
i = 100          #为变量 i 赋初始值
print('所有的水仙花数是: ',end='')
while i <= 999:  #循环继续的条件
    c = i%10     #获得个位数
```

```

    b = i//10%10      #获得十位数
    a = i//100        #获得百位数
    if a**3+b**3+c**3==i:    #判断是否是“水仙花数”
        print(i,end=' ')    #打印水仙花数
    i = i+1            #变量 i 增加 1
4.
numbers=[1,2,4,6,7,8,9,10,13,14,17,21,26,29]
even_number=[]
odd_number=[]
while len(numbers) > 0:
    number=numbers.pop()
    if(number%2 == 0):
        even_number.append(number)
    else:
        odd_number.append(number)
print('列表中的偶数有', even_number)
print('列表中的奇数有', odd_number)
5.
import random
import time
correctCount=0        #记录正确答对数
count=0               #记录回答的问题数
continueLoop='y'      #让用户来决定是否继续答题
startTime=time.time()  #记录开始时间
while continueLoop=='y':
    number1=random.randint(0,50)
    number2=random.randint(0,50)
    answer=eval(input(str(number1)+'+'+str(number2)+'='+'?'))
    if number1+number2==answer:
        print('你的回答是正确的! ')
        correctCount+=1
    else:
        print('你的回答是错误的。')
        print(number1,'+',number2,'=',number1+number2)
    count+=1
    continueLoop=input('输入 y 继续答题, 输入 n 退出答题: ')
endTime=time.time()    #记录结束时间
testTime=int(endTime-startTime)
print("正确率: %.2f%%\n 测验用时: %d 秒" % ((correctCount/count)*100,testTime))
6.
Names = ['宋爱梅','王志芳','于光','贾隽仙','贾燕青','刘振杰','郭卫东','崔红宇','马福平']
print("-----添加之前, 列表 A 的数据-----")
for Name in Names:

```

```

    print( Name,end=' ')
print(' ')
continueLoop='y'    #让用户来决定是否继续添加
while continueLoop=='y':
    temp = input('请输入要添加的学生姓名:')    #提示、并添加姓名
    Names.append(temp)
    continueLoop=input('输入 y 继续添加, 输入 n 退出添加: ')
print ("-----添加之后, 列表 A 的数据-----")
for Name in Names:
    print(Name, end=' ')

```

```

7.
a=1
b=1
n=int(input('请输入斐波那契数列的项数(>2 的整数): '))
print('前%d 项斐波那契数列为:'%(n),end='')
print(a,b,end=' ')
for k in range(3,n+1):
    c=a+b
    print(c,end=' ')
    a=b
    b=c

```

```

8.
import math
NUMBER_OF_PRIMES = 50
NUMBER_OF_PRIMES_PER_LINE = 10
count=0    #记录找到的素数个数
i=2
while count< NUMBER_OF_PRIMES:
    j=2
    for j in range(2,int(math.sqrt(i))+1):
        if(i%j==0):
            break
    else:
        print('{:>4}'.format(i),end='')    #格式化输出: 右对齐, 宽度为 4
        count += 1
        if(count%NUMBER_OF_PRIMES_PER_LINE==0):    #输出 10 个换行
            print(end='\n')
    i += 1

```

```

9.
s = input('Please input a string:\n')
letter = 0

```

```
space = 0
digit = 0
other = 0
for c in s:
    if c.isalpha():
        letter += 1
    elif c.isspace():
        space += 1
    elif c.isdigit():
        digit += 1
    else:
        other += 1
print('char=%d,space=%d,digit=%d,others=%d'%(letter,space,digit,other))
```

实验三 函数实验

一、实验目的

掌握 Python 函数的定义方式；return 语句的使用；正则表达式元字符、re 模块常用方法。熟悉 lambda 表达式声明匿名函数，在 lambda 表达式中调用函数，map()、reduce()、filter() 的使用；类的定义与使用，及其属性的定义与使用。了解局部作用域与全局作用域的区别。

二、实验过程

1. 利用 def 定义函数

Python 内建函数

```
print('hello world')
```

```
int('101')
```

```
def def_sum(x, y):    # 自定义函数
```

```
    z = x + y
```

```
    print('hello world')
```

```
    return z
```

```
def_sum(3, 4)        # 调用自定义的函数
```

任务实现：使用 def 关键字定义一个求列表均值的自定义函数

```
vec = [1, 2, 6, 0.3, 2, 0.5, -1, 2.4]
```

```
def def_mean(x):    # 定义函数
```

```
    m = 0
```

```
    for i in x:
```

```
        m += i
```

```
    return m/len(x)
```

```
def_mean(x=vec)    # 调用自定义的函数
```

2. 使用 lambda 创建匿名函数

```
y = lambda x: x ** 2    # 创建匿名函数
```

```
y(10)
```

```
y(x=5)
```

3. 存储并导入函数库

```
from def_mdoule import def_mean, y, pi    # 导入模块中的目标函数
```

```
vec = [1, 2, 6, 0.3, 2, 0.5, -1, 2.4] # 目标列表
def _mean(vec) # 调用函数求解均值
y(100)
pi
```

4.创建类

```
class Human:
    def __init__(self, age, gender): # 构造函数
        self.age = age # 类的属性
        self.gender = gender

    def sqrt(self, x):
        return x**2
```

对象 zhangfei = Human(age=23, gender='男') # 类的实例化, 得到一个具体对象

```
zhangfei.age # 对象的属性
zhangfei.gender
zhangfei.sqrt(10) # 调用对象的方法
```

对象 caocao = Human(age=36, gender='男') # 类的实例化, 得到一个具体对象

```
all_in_list = [0.2, 0.25]
all_in_list.append(3.56)
# zhangfei.append(0.3) # 非法操作, Human 对象没有 append 方法
print(zhangfei)
print(all_in_list)
```

5.统计词频

```
from functools import reduce
import re

str1="Youth is not a time of life; it is a state of mind; it is not a matter of rosy
cheeks, red lips and supple knees; it is a matter of the will, a quality of the
imagination, a vigor of the emotions; it is the freshness of the deep springs of life. "
words=str1.split() #以空字符为分隔符对 str1 进行分割
words1=[re.sub('\W',",",i) for i in words]#将字符串中的非单词字符替换为"
def fun(x,y):
    if y in x:
        x[y]=x[y]+1
    else:
        x[y]=1
```



```

        return x
result=reduce(fun, words1, {})      #统计词频
print("词频为: ",result)
6. 求最大公约数
def gcd(x, y):
    """该函数返回两个数的最大公约数"""
    if x > y:      #求出两个数的最小值
        smaller = y
    else:
        smaller = x
    i,gcd=2,1
    while i<=smaller:
        if((x % i == 0) and (y % i == 0)):
            gcd=i
            i+=1
    return gcd
# 用户输入两个数字
num1 = int(input("输入第一个数字: "))
num2 = int(input("输入第二个数字: "))
print( num1,"和", num2,"的最大公约数为", gcd(num1, num2))

```

实验四 正则表达式和文件操作实验

一、实验目的

掌握正则表达式的构成、边界匹配、分组、选择和引用匹配以及正则表达式模块 re、对象和 Match 对象；掌握文本文件的打开、读写以及文件指针的定位，二进制文件的打开与读写。Os、os.path、shutil 对文件与文件夹的操作，csv 文件的读取和写入。

二、实验过程

1.正则表达式

```
import re

string = '1. A small sentence. - 2. Another tiny sentence. '
re.findall('sentence', string)
re.search('sentence', string)
re.match('1. A small sentence', string)
re.sub(pattern='small', repl='large', string=string)
re.sub(pattern='small', repl='', string=string)

string = 'small smell smll smsmll sm3ll sm.ll sm?ll sm\\nll sm\\tll'
re.findall('sm.ll', string)
re.findall('sm[asdbf]ll', string)
re.findall('sm[a-zA-Z0-9]ll', string)
re.findall('sm\\.ll', string)
re.findall('sm[.\\- ?]ll', string)
re.findall('small|smell', string)
re.findall('sm\\wll', string)

re.findall('sm..ll', string)
re.findall('sm.{2}ll', string)
re.findall('sm.{1,2}ll', string)
re.findall('sm.{1,}ll', string)
re.findall('sm.?ll', string) # {0,1}
print(re.findall('sm.+ll', string)) # {0,}
print(re.findall('sm.*ll', string)) # {1,}
```

```
re.findall('sm\?ll', string)
```

```
rawdata = '555-1239
```

```
Moe Szyslak
```

```
(636) 555-0113
```

```
Burns, C.Montgomery
```

```
555-6542
```

```
Rev. Timothy Lovejoy
```

```
555 8904
```

```
Ned Flanders
```

```
636-555-3226
```

```
Simpson,Homer
```

```
5553642
```

```
Dr. Julius Hibbert'
```

```
import pandas as pd
```

```
rawdata = '555-1239Moe Szyslak(636) 555-0113Burns, C.Montgomery555-6542Rev.
```

```
Timothy Lovejoy555 8904Ned Flanders636-555-3226Simpson,Homer5553642Dr.
```

```
Julius Hibbert'
```

```
names = re.findall('[A-Z][A-Za-z,. ]*', rawdata)
```

```
tels = re.findall('\(?[0-9]{0,3}\)?[ \-]?[0-9]{3}[ \-]?[0-9]{4}', rawdata)
```

```
pd.DataFrame({'Names': names, 'TelPhone': tels})
```

```
import requests
```

```
import re
```

```
url = 'http://www.tipdm.com/tipdm/index.html'
```

```
rqq = requests.get(url)
```

```
rqq.encoding = 'utf-8'
```

```
rqq.text
```

2.

'使用 writer 写入 csv 文件'

```
import csv
```

```
with open('consumer.csv', 'w', newline='') as csvfile: #写入的数据将覆盖
```

```
consumer.csv 文件
```

```
    spamwriter = csv.writer(csvfile)          #生成 csv.writer 文件对象
```

```

        spamwriter.writerow(['55','555','55'])    #写入一行数据
        spamwriter.writerows([(['35','355','35'),('18','188','18')])
with open('consumer.csv',newline='') as csvfile:    #重新打开文件
    spamreader = csv.reader(csvfile)
    for row in spamreader:        #输出用 writer 对象的写入方法写入数据后
        的文件
        print(row)

```

3.

'使用 writer 向 csv 文件追加数据'

```

import csv
with open('consumer.csv', 'a+', newline='') as csvfile:
    spamwriter = csv.writer(csvfile)
    spamwriter.writerow(['55','555','55'])
    spamwriter.writerows([(['35','355','35'),('18','188','18')])
with open('consumer.csv',newline='') as csvfile:    #重新打开文件
    spamreader = csv.reader(csvfile)
    for row in spamreader:        #输出用 writer 对象的写入方法写入数据后
        的文件
        print(row)

```

4.

'使用 csv.DictReader 读取 csv 文件'

```

import csv
with open('consumer.csv', 'r') as csvfile:
    dict_reader = csv.DictReader(csvfile)
    for row in dict_reader:
        print(row)

```

5.

'使用 csv.DictReader 读取 csv 文件，并为输出的数据指定新的字段名'

```

import csv
print_dict_name=['年龄','消费金额','消费频率']
with open('consumer.csv', 'r') as csvfile:
    dict_reader = csv.DictReader(csvfile,fieldnames=print_dict_name)
    for row in dict_reader:
        print(row)
print("\nconsumer.csv 文件内容: ")
with open('consumer.csv',newline='') as csvfile:    #重新打开文件
    spamreader = csv.reader(csvfile)
    for row in spamreader:
        print(row)

```

6.'使用 csv.DictWriter()写入 csv 文件'

```

import math
import csv
dict_record = [{'客户年龄': 23, '平均每次消费金额': 318, '平均消费周期': 10}, {'客户年龄': 22, '平均每次消费金额': 147, '平均消费周期': 13}]
keys = ['客户年龄', '平均每次消费金额', '平均消费周期']
#在该程序文件所在目录下创建 consumer1.csv 文件
with open('consumer1.csv', 'w+', newline='') as csvfile:
    #文件头以列表的形式传入函数，列表的每个元素表示每一列的标识
    dictwriter = csv.DictWriter(csvfile, fieldnames=keys)
    #若此时直接写入内容，会导致没有数据名，需先执行 writeheader()将文件头写入
    # writeheader()没有参数，因为在建立对象 dictwriter 时，已设定了参数 fieldnames
    dictwriter.writeheader()
    for item in dict_record:
        dictwriter.writerow(item)

print("以 csv.DictReader()方式读取 consumer1.csv: ")
with open('consumer1.csv', 'r') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row)

print("\n 以 csv.reader()方式读取 consumer1.csv: ")
with open('consumer1.csv', newline='') as csvfile: #重新打开文件
    spamreader = csv.reader(csvfile)
    for row in spamreader:
        print(row)

```

7.

```

import csv
def read(file):
    with open(file, 'r+', newline='') as csvfile:
        reader = csv.reader(csvfile)
        return [row for row in reader]

def write(file, lst):
    with open(file, 'w+', newline='') as csvfile:
        # delimiter=';'指定写入文件的分隔符，quoting 指定双引号的规则
        writer = csv.writer(csvfile, delimiter=';', quoting=csv.QUOTE_ALL)
        for row in lst:
            writer.writerow(row)

def main():

```

```

columns = int(input("请输入要输入的列数: "))
input_list = []
i=1
with open('consumer.csv', 'r', newline='') as csvfile:
    spamreader = csv.reader(csvfile)
    for row in spamreader:
        if i<=columns+1:
            input_list.append(row)
        else:
            break
    i+=1
print(input_list)
write('consumer1.csv', input_list)
written_value = read('consumer1.csv')
print(written_value)

main()

8
import os
def rename_files(filepath):
    os.chdir(filepath)          #改变当前目录
    print('更名前%s 目录下的文件列表'%filepath)
    print(os.listdir())
    filelist = os.listdir()      #获取当前文件夹中所有文件的名称列表
    for item in filelist:
        if item[item.rfind('.')+1:]=='txt':
            #rfind('.')返回'.'最后一次出现在字符串中的位置
            newname = item[:item.rfind('.')+1] + 'html'
            os.rename(item, newname)

def main():
    while True:
        filepath = input('请输入路径:').strip()
        if os.path.isdir(filepath) == True:
            break
    rename_files(filepath)
    print('更名后%s 目录下的文件列表'%filepath)
    print(os.listdir(filepath))

main()

```

实验五 数据可视化实验

一、实验目的

学会使用 matplotlib 的 pyplot 子库绘制线形图、直方图、条形图、饼图以及散点图。

二、实验过程

1.

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(0.0, 5.0, 0.02)      #生成一个序列
plt.plot(a, np.sin(2*np.pi*a), 'k--')
plt.xlabel('横轴：时间', fontproperties='KaiTi', fontsize=18)
plt.ylabel('纵轴：振幅', fontproperties='KaiTi', fontsize=18)
plt.title("正弦线", fontproperties='LiSu', fontsize=18)
plt.show()
```

2.

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(0.0, 5.0, 0.02)
plt.plot(a, np.sin(2*np.pi*a), 'k--')
# fontproperties 也可用 fontname 代替
plt.ylabel('纵轴：振幅', fontproperties='Kaiti', fontsize=20)
plt.xlabel('横轴：时间', fontproperties='Kaiti', fontsize=20)
plt.title(r'正弦波实例:  $y=\sin(2\pi x)$ ', fontproperties='Kaiti', fontsize=20)
""" xy=(2.25,1)指定箭头的位置，xytext=(3, 1.5)指定箭头的注解文本的位置，
facecolor='black'指定箭头填充的颜色，shrink=0.1 指定箭头的长度，width=1 指定
箭头的宽度"""
plt.annotate(r' $\mu=100$ ', fontsize=15, xy=(2.25,1), xytext=(3, 1.5), arrowprops =
dict(facecolor='black', shrink=0.1, width=1))
# text()可以在图中的任意位置添加文字，1, 1.5 为文本在图像中的坐标
plt.text(1, 1.5, '正弦波曲线', fontproperties='Kaiti', fontsize=20) #添加文本'正弦波
```

曲线'

```
plt.axis([0, 5, -2, 2]) #指定 x 轴和 y 轴的取值范围
```

```
plt.grid(True) #在绘图区域添加网格线
```

```
plt.show()
```

3.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.arange(-2*np.pi, 2*np.pi, 0.01)
```

```
y1 = np.sin(2*x)/x
```

```
y2 = np.sin(3*x)/x
```

```
y3 = np.sin(4*x)/x
```

```
plt.plot(x, y1, 'k--')
```

```
plt.plot(x, y2, 'k-.')
```

```
plt.plot(x, y3, 'k')
```

```
plt.show()
```

4.

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
x = np.arange(-2*np.pi, 2*np.pi, 0.01)
```

```
y1 = np.sin(2*x)/x
```

```
y2 = np.sin(3*x)/x
```

```
y3 = np.sin(4*x)/x
```

```
plt.plot(x, y1, 'k--')
```

```
plt.plot(x, y2, 'k-.')
```

```
plt.plot(x, y3, 'k')
```

```
plt.xticks([-2*np.pi,-np.pi,0,np.pi,2*np.pi],[r'$-2\pi$',r'$-\pi$',r'$0$',r'$+\pi$',r'$+2\pi$'])
```

```
#设置 y 轴范围及标注刻度值
```

```
plt.yticks([-1,0,1,2,3,4],[r'$-1$',r'$0$',r'$1$',r'$2$',r'$3$', r'$4$'])
```

```
ax = plt.gca()
```

```
ax.spines['right'].set_color('none') #设置右边框的颜色为'none'
```

```
ax.spines['top'].set_color('none')
```

```
ax.xaxis.set_ticks_position('bottom') #将底边框设为 x 轴
```



```
ax.spines['bottom'].set_position(('data',0)) #移动底边框
ax.yaxis.set_ticks_position('left') #将左边框设为 y 轴
ax.spines['left'].set_position(('data',0)) #移动左边框
plt.show()
```

5.

```
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(-5,5,0.1)
y=x**2
'''使用 figure 创建一块自定义大小的画布(窗口)，使得后面的图形输出在这块规定了大小的画布上，其中参数 figsize 设置画布大小'''
plt.figure(figsize=(8,8))
'''将 figure 设置的画布分成多个部分，参数‘221’表示将画布分成两行两列的 4 块区域，1 表示选择 4 块区域中的第一块作为输出区域，如果参数设置为 subplot(111)，则表示图形直接输出在整块画布上，画布不分割成小块区域'''
plt.subplot(221)
plt.plot(x,y) #在 2x2 画布中第一块区域绘制线形图
plt.subplot(222)
plt.plot(x,y) #在 2x2 画布中第二块区域绘制线形图
plt.subplot(223) #在 2x2 画布中第三块区域绘制线形图
plt.plot(x,y)
plt.subplot(224) # 在 2x2 画布中第四块区域绘制线形图
plt.plot(x,y)
plt.show()
```

6.

```
import matplotlib
import matplotlib.pyplot as plt
xvalues = [0,1,2,3] #条形图在 x 轴上的起始位置
GDP = [13908.57,12406.8,9891.48,9709.02]
# 设置图表的中文显示方式
matplotlib.rcParams['font.family'] = 'FangSong' #设置字体为 FangSong 中文仿宋
matplotlib.rcParams['font.size'] = 15 #设置字体的大小
plt.bar(range(4), GDP, align = 'center', color='black') # 绘图
plt.ylabel( 'GDP') # 添加 y 轴标签
```

```

plt.title( 'GDP--TOP4 的城市')    # 添加标题
plt.xticks(range( 4), ['上海市', '北京市', '广州市', '深圳市'])    #设置 x 轴刻度标签
plt.ylim([9000, 15000])    # 设置 Y 轴的刻度范围
# 为每个条形图添加数值标签
for x,y in enumerate(GDP):
    plt.text(x, y+ 100, '%s'%round(y, 1), ha= 'center')    # ha= 'center'表示居中对齐
plt.show()    # 显示图形

```

7.

```

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
matplotlib.rcParams['font.family'] = 'FangSong'    #设置字体为 FangSong 中文仿宋
matplotlib.rcParams['font.size'] = 15    #设置字体的大小
label = ['上海市', '北京市', '广州市', '深圳市']
GDP = [13908.57, 12406.8, 9891.48, 9709.02]
index =np.arange(len(GDP))
plt.barh(index, GDP, color='black')
plt.yticks(index, label)    #设置 y 轴刻度标签
plt.xlabel( 'GDP')    # 添加 x 轴标签
plt.ylabel('Top4 城市')
plt.title( 'GDP--TOP4 的城市')    # 添加标题
plt.grid(axis='x')
plt.show()    #显示图形

```

8.

```

import matplotlib.pyplot as plt
labels = ('Java','C','C++','Python')
sizes = [15,30,45,10]
explode = (0,0.1,0,0)    #0.1 表示将'C'那一块离开中心的距离
#startangle 表示饼图的起始角度
plt.pie(sizes,explode=explode,labels=labels,autopct='%1.1f%%',shadow=False,startangle=90)
plt.show()

```

9.

```

from matplotlib import pyplot as plt

```

```

import matplotlib
matplotlib.rcParams['font.family'] = 'FangSong' #显示 FangSong 中文仿宋字体
matplotlib.rcParams['font.size'] = '12' #设置字体大小
plt.figure(figsize=(7,7))#创建一个绘图对象(窗口)，指定绘图对象的宽度和高度
#定义饼状图每块旁边的标签
labels = ('六分水','三分山','一分田')
#定义饼图中每块的大小
sizes = (6,3,1)
colors = ['red','yellowgreen','lightskyblue']
explode = (0,0,0.05) #0.05 表示'一分田'那一块离开中心的距离
plt.pie(sizes,explode=explode,labels=labels,colors=colors, labeldistance = 1.1,
autopct = '%4.2f%%',shadow = True, startangle = 90,pctdistance = 0.5)
#labeldistance，文本的位置离饼的中心点有多远，1.1 指 1.1 倍半径的位置
#autopct，圆里面的文本格式，%4.2f%%表示数字显示的宽度有四位，小数点后
有 2 位
#shadow，饼是否有阴影，取 False 没有阴影，取 True 有阴影
#startangle，饼图的起始绘制角度，一般选择从 90 度开始
#pctdistance，百分比的 text 离圆心的距离，0.5 指 0.5 倍半径的位置
plt.legend(loc="best") #为饼图添加图例，loc="best"用来设置图例的位置
plt.show()

```

实验六 numpy 库实验

一、实验目的

掌握 ndarray 数组的创建，特殊的 ndarray 数组的创建，ndarray 数组的索引、切片和选择，ndarray 数组的统计计算以及随机数数组、数组的基本运算和数组数据文件的读写。

二、实验过程

7.1 ndarray 多维数组

7.1.1 创建 ndarray 数组

1. 使用 numpy 的 array()函数创建 ndarray 数组

numpy 的 array() 函数的语法格式如下。

`numpy.array(object, dtype=None)`

作用：返回满足要求的数组对象。

参数说明：

object：指定生成数组的数据序列，可以是列表、元组。

dtype：数据类型，指定数组元素的数据类型。

```
>>> import numpy as np          #本章中出现的 np 默认均是这个含义
>>> a=np.array([1,2,3])         #以列表作为参数创建一维数组
>>> a
array([1, 2, 3])
>>> b = np.array([[1,2],[3,4]]) #以列表作为参数创建 2×2 的二维数
组
>>> b
array([[1, 2],
       [3, 4]])
>>> c=np.array(((1,3,5),(2,4,6))) #以嵌套元组作为参数创建数组
>>> c
array([[1, 3, 5],
       [2, 4, 6]])
>>> d=np.array([(1,3,5),(2,4,6)]) #以元组所组成的列表作为参数创
建数组
>>> d
```

```
array([[1, 3, 5],
       [2, 4, 6]])
```

2. 使用 numpy 的 ndarray() 函数创建 ndarray 数组

```
>>> e=np.ndarray(shape=(2,3),dtype=int,
buffer=np.array([0,1,2,3,4,5,6,7,9]), offset=0, order="C") #buffer 中
的数据按行的顺序存入将要创建的数组 e 中
```

```
>>> e
array([[0, 1, 2],
       [3, 4, 5]])
```

```
>>> f=np.ndarray(shape=(2,3),
dtype=int,buffer=np.array([0,1,2,3,4,5,6,7,9]), offset=0, order="F")
#buffer 中的数据按列的顺序存入将要创建的数组 f 中
```

```
>>> f
array([[0, 2, 4],
       [1, 3, 5]])
```

```
>>> g=np.ndarray(shape=(2,3),
dtype=int,buffer=np.array([0,1,2,3,4,5,6,7,9]), offset=4, order="C")
#首个数据的偏移量 offset 为 4，为 4 的整数倍
```

```
>>> g
array([[1, 2, 3],
       [4, 5, 6]])
```

#首个数据的偏移量 offset 的值为 8，为 8 的整数倍

```
>>> h=np.ndarray(shape=(2,3), dtype=float, buffer= np.array
([0.1,1.1,2.1,3.1, 4.1, 5.1, 6.1, 7.1, 8.1]), offset=8, order="C")
```

```
>>> h
array([[ 1.1,  2.1,  3.1],
       [ 4.1,  5.1,  6.1]])
```

7.1.2 创建特殊的 ndarray 数组

1. 使用 ones()函数创建一个元素全为 1 的数组

```
>>> import numpy as np
>>> a = np.ones(shape = (3, 3))    #通过 shape 指定生成 3×3 的全为 1
的数组
```

```

>>> a
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])
>>> a.dtype      #通过数组的属性 dtype 返回数组元素的类型
dtype('float64')
>>> np.ones((2, 2))
array([[1., 1.],
       [1., 1.]])

```

此外，通过 `ones_like()` 函数可以创建与已知数组的 `shape` 相同的元素全为 1 数组。

```

>>> a = np.array([[1, 2, 3], [3, 4, 5]])
>>> a
array([[1, 2, 3],
       [3, 4, 5]])
>>> b = np.ones_like(a)
>>> b
array([[1, 1, 1],
       [1, 1, 1]])

```

2. 使用 `zeros()` 函数创建一个元素全为 0 的数组

`>>> b=np.zeros(shape=(3,3), dtype=int)` #通过 `shape` 指定生成 3×3 的全为 0 的数组

```

>>> b
array([[0, 0, 0],
       [0, 0, 0],
       [0, 0, 0]])

```

此外，通过 `zeros_like()` 函数可以创建与已知数组 `shape` 相同的元素全为 0 数组。

```

>>> a = np.array([[1, 2, 3], [3, 4, 5]])
>>> b = np.zeros_like(a)
>>> b
array([[0, 0, 0],
       [0, 0, 0]])

```

3. 使用 `empty()` 函数创建一个随机数组

```
>>> c = np.empty(shape=(2,3), dtype=int)
>>> c
array([[ 308,    0,    0],
       [1713398048, 1952673397, 745434985]])
```

此外，通过 `empty_like()` 函数可以创建与已知数组 `shape` 相同的随机数组。

4. 使用 `arange()` 函数创建均匀间隔的一维数组

```
>>> import numpy as np
>>> range(1,10)
range(1, 10)
>>> list(range(1,10))
[1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> np.arange(1,10)
array([1, 2, 3, 4, 5, 6, 7, 8, 9])    #不包括 10
>>> np.arange(1, 10, 0.5)            #步长为 0.5
array([ 1. ,  1.5,  2. ,  2.5,  3. ,  3.5,  4. ,  4.5,  5. ,  5.5,
        6. ,
        6.5,  7. ,  7.5,  8. ,  8.5,  9. ,  9.5])
>>> A=np.arange(1,10).reshape(3,3)#把 np.arange() 生成的一维数组拆
为二维数组
>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> B=np.arange(1, 28).reshape(3,3,3) #把 arange() 生成的一维数组拆
分三维数组
>>> B
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],
       [[10, 11, 12],
        [13, 14, 15],
```

```
[16, 17, 18]],
```

```
[[19, 20, 21],  
 [22, 23, 24],  
 [25, 26, 27]]])
```

5. 使用 `linspace()` 函数创建等差数组

①使用三个参数，第一个参数表示起始点，第二个参数表示终止点，第三个参数表示将要创建的数组包含的元素个数。

```
>>> import numpy as np  
>>> np.linspace(1,10,10)  
array([ 1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9.,  
10.])  
>>> np.linspace(0,10,6,dtype=int) #通过 dtype=int 指定数组的元素类  
型
```

```
array([ 0,  2,  4,  6,  8, 10])
```

②创建一个元素全部是 1 的等差数列，也可以让所有元素为 0。

```
>>> np.linspace(1,1,10,dtype=int)  
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

③通过将参数 `endpoint` 设置为 `False` 来指定所生成的数组不包含结尾值 `stop`。

```
>>> np.linspace(1,10,10,endpoint=False)  
array([ 1. ,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  
9.1])
```

④将可选参数 `retstep` 的值设置为 `True`，注意观察返回值的形式。

```
>>> np.linspace(2.0, 3.0, num=5, retstep=True)  
(array([2., 2.25, 2.5, 2.75, 3.]), 0.25) #0.25 为数组中相邻两个元素  
的间隔值
```

6. 使用 `logspace()` 函数创建等比数组

```
>>> import numpy as np  
>>> np.logspace(1, 3, num=3)  
array([ 10.,  100., 1000.])  
>>> np.logspace(1, 3, num=3, base=2)  
array([ 2.,  4.,  8.])
```


7. 使用 `eye()`函数创建对角线全为 1、其余位置全是 0 的二维数组

```
>>> np.eye(3, k = 0)
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
>>> np.eye(3, k = 1)
array([[ 0.,  1.,  0.],
       [ 0.,  0.,  1.],
       [ 0.,  0.,  0.]])
>>> np.eye(3, k = -1)
array([[ 0.,  0.,  0.],
       [ 1.,  0.,  0.],
       [ 0.,  1.,  0.]])
```

8. 使用 `identity()`函数创建 $n \times n$ 单位数组

```
>>> np.identity(3, dtype=int)
array([[1, 0, 0],
       [0, 1, 0],
       [0, 0, 1]])
```

9. 使用 `full()`函数创建由固定值填充的数组

```
>>> np.full((2, 3), 10)
array([[10, 10, 10],
       [10, 10, 10]])
```

此外，通过 `full_like()`函数可以创建与已知数组 `shape` 相同的由 `fill_value` 指定的值填充的数组。

```
>>> a = np.array([[1, 2, 3], [3, 4, 5]])
>>> a
array([[1, 2, 3],
       [3, 4, 5]])
>>> np.full_like(a, 1)
array([[1, 1, 1],
       [1, 1, 1]])
```

7.1.3 ndarray 对象的数据类型

可以使用 `dtype` 参数来定义一个复数数组：

```
>>> import numpy as np
>>> a=np.array([[1, 2, 3], [4, 5, 6]], dtype=complex)
>>> a
array([[ 1.+0. j,   2.+0. j,   3.+0. j],
       [ 4.+0. j,   5.+0. j,   6.+0. j]])
```

7.1.4 ndarray 数组对象的属性

```
>>> import numpy as np
>>> a = np.array([[ 0, 1, 2, 3, 4], [ 5, 6, 7, 8, 9], [10, 11, 12, 13,
14]])
>>> a.T                                #返回数组的转置
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
>>> a.size                            #返回数组中元素的个数
15
>>> a.itemsize                        #返回数组中的单个元素在内存所占字节数
4
>>> a.ndim                            #返回数组的维度
2
>>> a.shape                           #返回数组的型
(3, 5)
>>> a.flat                            #返回数组的迭代器
<numpy.flatiter object at 0x0000000003728C20>
>>> for x in a.flat:
    print(x, end=', ')

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
>>> a.flat=[1, 2, 3, 4, 5]
>>> a
```

```

array([[1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5]])
>>> a = np.array([[ 0, 1, 2, 3, 4], [ 5, 6, 7, 8, 9], [10, 11, 12, 13,
14]])
>>> a.flat=np.arange(1, 16)
>>> a
array([[ 1,  2,  3,  4,  5],
       [ 6,  7,  8,  9, 10],
       [11, 12, 13, 14, 15]])

```

7.2 数组元素的索引、切片和选择

7.2.1 索引和切片

1. 单个元素索引

```

>>> import numpy as np
>>> x = np.arange(10)
>>> x[5]          #索引为非负值，获取第 i 个值，从 0 开始计数
5
>>> x[-2]         #索引为负值，从末尾开始索引，倒数第一个索引为-1
8
>>> x[2:6]        #切片
array([2, 3, 4, 5])
>>> x[:7]
array([0, 1, 2])
>>> x[0:10:2]     #切片，2 为所取元素的间隔，一共取出 5 个元素
array([0, 2, 4, 6, 8])

>>> x.shape = (2,5) #改变数组的形状
>>> x
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
>>> x[(1,3)]      #用逗号分隔的索引元组获取元素
8

```

2. 使用列表索引数组

```
>>> x = np.arange(10,1,-1)
>>> x
array([10,  9,  8,  7,  6,  5,  4,  3,  2])
>>> x[[2, 2, 1, 6]] #用列表[2,2,1,6]取出 x 中的第 2,2,1,6 的四个元素组成一个数组
array([8, 8, 9, 4])
```

```
>>> y = np.arange(35).reshape(5,7) #产生一个 5×7 的数组
>>> y[1:4, 2] #第 1 行到第 3 行中第 2 列的元素
array([ 9, 16, 23])
>>> y[[0,2,4], 1:3] #使用列表索引行，使用切片索引列
array([[ 1,  2],
       [15, 16],
       [29, 30]])
```

3. 布尔值索引数组

```
>>> y = np.arange(30)
>>> b = y>20
>>> y[b]
array([21, 22, 23, 24, 25, 26, 27, 28, 29])
```

7.2.2 选择数组元素的方法

(1) `ndarray.take(indices[, axis=None, out=None, mode='raise'])`: 根据指定的索引 `indices` 从数组对象 `ndarray` 中获取对应元素。

```
>>> import numpy as np
>>> x = np.arange(0, 20, 2)
>>> x
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
>>> x.take([0, 2, 4]) #获取 0, 2, 4 索引处的元素
array([0, 4, 8])
>>> x.take([[2, 5], [3, 6]]) #返回数组的形状与索引的形状相同
array([[ 4, 10],
       [ 6, 12]])
```

```
>>> y=np.array([[0, 5, 10, 15], [20, 25, 30, 35], [40, 45, 50, 55], [60, 65, 70, 75]])
```

```
>>> y.take([[1, 2], [2, 3]])    #take() 默认情况下把数组 y 当成一个一维数组
```

```
array([[ 5, 10],
       [10, 15]])
```

```
>>> y.take([[1, 2], [2, 3]], axis=0)    #axis=0 表示按行选取元素
```

```
array([[20, 25, 30, 35],
       [40, 45, 50, 55]],
```

```
       [[40, 45, 50, 55],
       [60, 65, 70, 75]])
```

```
>>> y.take([[1, 2], [2, 3]], axis=1)
```

```
array([[ 5, 10],
       [10, 15]],
```

```
       [[25, 30],
       [30, 35]],
```

```
       [[45, 50],
       [50, 55]],
```

```
       [[65, 70],
       [70, 75]])
```

(2) `ndarray.put(indices, values[, mode])`: 将数组中索引 `indices` 指定的位置处的元素值设置为 `values` 中对应的元素值。

```
>>> x = np.arange(0, 20, 2)
```

```
>>> x.put([0, 1], [1, 3]) #将 x 中索引[0, 1]处的值设置为列表[1, 3]中对应的值
```

```
>>> x
```

```
array([ 1,  3,  4,  6,  8, 10, 12, 14, 16, 18])
```

(3) `ndarray.searchsorted(v, side='left', sorter=None)`: 将 `v` 插入到当前有序的数组中, 返回插入的位置索引。

```
>>> w = np.array([1, 2, 3, 3, 3, 3, 6, 7, 9, 10, 12])
```

```
>>> w.searchsorted(3)
```

```
2
```

```
>>> w.searchsorted(3, side='right')          # side='right'
```

```
6
```

(4) `ndarray.partition(kth[, axis, kind, order])`: 将数组重新排列, 所有小于 `kth` 的值在 `kth` 的左侧, 所有大于或等于 `kth` 的值在 `kth` 的右侧。

```
>>> x = np.array([3, 4, 2, 1])
```

#将数组 `x` 重新排列, 所有小于 3 的值在 3 的左侧, 所有大于或等于 3 的值在 3 的右侧

```
>>> x.partition(3)
```

```
>>> x
```

```
array([2, 1, 3, 4])
```

(5) `ndarray.argpartition(kth[, axis, kind, order])`: 返回对数组执行 `partition` 之后的元素索引。

```
>>> x = np.array([3, 4, 2, 1])
```

```
>>> x.argpartition(2)    #对数组执行 partition 之后的元素索引
```

```
array([3, 2, 0, 1], dtype=int64)
```

>>> `x[x.argpartition(2)]` #对数组执行 `partition` 之后的元素索引对应的元素

```
array([1, 2, 3, 4])
```

```
>>> x
```

```
array([3, 4, 2, 1])
```

(6) `ndarray.diagonal(offset=0, axis1=0, axis2=1)`: 返回指定的对角线。

```
>>> k = np.array([[ 0,  1,  2,  3], [ 4,  5,  6,  7], [ 8,  9, 10, 11]])
```

```
>>> k
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>> k.diagonal()    #返回前 3 行 3 列所对应的主对角线
```

```
array([ 0,  5, 10])
```

(7) `ndarray.item(*args)`: 复制数组中的一个元素, 并返回。

```
>>> a=np.arange(9).reshape(3,3)
```

```
>>> a
```

```

array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> a.item(3)           #按行存储的顺序获取数组中序号为 3 的元素
3
>>> a.item((2, 2))      #获取行号为 2 列号为 2 处的元素
8

```

(8) `ndarray.itemset(*args)`: 修改数组中某个元素的值。

```

>>> a.itemset(3, 33)    #将序号为 3 的元素修改为 33
>>> a
array([[ 0,  1,  2],
       [33,  4,  5],
       [ 6,  7,  8]])
>>> a.itemset((1, 2), 12) #将 a 数组中(1, 2)处的元素修改为 12
>>> a
array([[ 0,  1,  2],
       [33,  4, 12],
       [ 6,  7,  8]])

```

(9) `ndarray.tolist()`: 将数组转换成 Python 标准 list。

```

>>> a.tolist()          #将数组转 a 换成 Python 标准 list
[[0, 1, 2], [33, 4, 12], [6, 7, 8]]

```

(10) `ndarray.tostring()`: 构建一个包含 `ndarray` 的原始字节数据的字节字符串。

```

>>> a.tostring()        #构建一个包含 a 的原始字节数据的字节字符串
b'\x00\x00\x00\x00\x01\x00\x00\x00\x02\x00\x00\x00!\x00\x00\x00\x04\x00\x00\x00\x0c\x00\x00\x00\x06\x00\x00\x00\x07\x00\x00\x00\x08\x00\x00\x00'

```

(11) `ndarray.copy([order])`: 复制数组并返回（深拷贝）。

```

>>> b = a.copy()        #深拷贝，b 与 a 是两个无关的数组
>>> b[0,0] = 10
>>> b
array([[10,  1,  2],
       [33,  4, 12],
       [ 6,  7,  8]])

```

```
>>> a
array([[ 0,  1,  2],
       [33,  4, 12],
       [ 6,  7,  8]])
```

(12) `ndarray.fill(value)`: 使用值 `value` 填充数组。

```
>>> b.fill(6)  #使用值 6 填充数组 b
>>> b
array([[6, 6, 6],
       [6, 6, 6],
       [6, 6, 6]])
```

7.2.3 ndarray 数组的形状变换

(1) `ndarray.reshape(shape,order)`: 返回一个具有相同数据域, 但 `shape` 不一样的视图。

```
>>> x = np.arange(0,12)
>>> x
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> y = x.reshape((3,4))
>>> y
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> y[0][0]=20  #修改 y 数组的元素, 直接影响原数组中的元素值
>>> y
array([[20,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
>>> x          #x(0,0)下标元素的值也发生了相应的改变
array([20,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

(2) `ndarray.resize(new_shape)`: 原地修改数组的形状, 需要保持元素个数前后相同。

```
>>> x.resize((3,4))  #resize 没有返回值, 会直接修 x 改数组的 shape
>>> x                #x 数组的形状发生了改变
array([[20,  1,  2,  3],
```



```
[ 4,  5,  6,  7],
 [ 8,  9, 10, 11]])
```

(3) **ndarray.transpose(*axes)**: 返回数组针对某一轴进行转置后的数组，对于二维 **ndarray**，**transpose** 在不指定参数时默认是矩阵转置。

```
>>> x = np.arange(4).reshape((2,2))
```

```
>>> x
```

```
array([[0, 1],
       [2, 3]])
```

```
>>> x_transpose = x.transpose() #对于二维数组，默认返回数组的转置
```

```
>>> x_transpose
```

```
array([[0, 2],
       [1, 3]])
```

>>> x.transpose((0,1)) # (0,1)表示按照原坐标轴改变序列，也就是保持不变

```
array([[0, 1],
       [2, 3]])
```

说明：第一个方括号（第一维）为 0 轴，第二个方括号为（第二维）1 轴。

>>> x.transpose((1,0)) # x.transpose((1,0))表示交换“0 轴”和“1 轴”

```
array([[0, 2],
       [1, 3]])
```

(4) **ndarray.flatten(order)**: 返回将原数组展平后的一维数组的拷贝（全新的数组）。

```
>>> x = np.arange(0,12).reshape((3,4))
```

```
>>> x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
>>> y = x.flatten() #返回一个全新的数组
```

```
>>> y
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

```
>>> y[0] = 300 #y 中下标为 0 的元素的值的改变，不会影响到 x 数组
```

```
>>> x #修改 y 中元素的值，不影响 x 中的元素
```

```
array([[ 0,  1,  2,  3],
```

```
[ 4,  5,  6,  7],
 [ 8,  9, 10, 11]])
```

(5) `ndarray.ravel(order)`: 返回将原数组展平后的一维数组的视图。

```
>>> x=np. array([[1, 2], [3, 4]])
>>> x
array([[1, 2],
       [3, 4]])
>>> x.ravel()           #行序优先展平
array([1, 2, 3, 4])
>>> x.ravel('F')        #列序优先展平
array([1, 3, 2, 4])
```

7.3 随机数数组

7.3.1 简单随机数

(1) `rand(d0,d1,...,dn)`: 生成一个(d0, d1, ..., dn)维的均匀分布的随机数数组。

```
>>> from numpy import random
>>> random.rand()       #生成[0, 1)之间均匀分布的随机数
0.26091475743474
>>> random.rand(5)      #生成一个形状为 5 的一维数组
array([0.39347271, 0.48445215, 0.75060248, 0.56305932, 0.36042016])
>>> random.rand(2, 3)    #生成 2×3 的二维数组
array([[0.14348196, 0.4153655 , 0.43341674],
       [0.83781023, 0.99925649, 0.39293883]])
```

(2) `randn(d0, d1, ..., dn)`: 生成一个(d0,d1,...,dn)维的标准正态分布随机数数组。

```
>>> random.randn()      #无参
1.4798279036179214
>>> random.randn(2, 3)   #生成 2×3 数组
array([[ 0.32426548,  1.67166582,  0.61398562],
       [-0.80949354,  0.38630344,  0.60296898]])
```

(3) `randint(low, high=None, size=None, dtype='T')`: 生成 size 个随机整数, 取值区间为[low, high), 若没有输入参数 high, 则取值区间为[0,low)。

```
>>> random.randint(3, size=5) #生成 5 个 [0, 3) 之间的随机整数
array([0, 0, 2, 2, 1])
```

>>> random.randint(2, 6, (2, 3)) #生成 2×3 的随机整数数组, 整数取值范围为[2, 6)

```
array([[2, 3, 3],
       [4, 2, 2]])
```

(4) **random_sample(size=None)**: 生成一个[0,1)之间的随机数或指定维的随机数组。

```
>>> random.random_sample()          #生成一个[0, 1)之间随机浮点数
0.6834957797352343
```

```
>>> random.random_sample(2)          #生成 shape=2 的一维数组
array([0.89888107, 0.44997489])
```

```
>>> random.random_sample((2, 3))     #生成  $2 \times 3$  数组
array([[0.95211226, 0.10665191, 0.50584127],
       [0.72724414, 0.85263057, 0.46839588]])
```

(5) **choice(a,size=None, replace=True, p=None)**: 从 a (数组) 中选取 size 个 (维度) 随机数, **replace=True** 表示可重复抽取, p 是 a 中每个数出现的概率, 若 a 是整数, 则 a 代表的数组是 **np.arange(a)**。

```
>>> import numpy as np
```

```
>>> np.random.choice(3) #a 为整数, size 为 None, 生成一个 range(3)
中的随机数
```

```
1
```

```
>>> np.random.choice(2, 2) #a 为整数, size 为整数, 生成一个 shape=2 的
一维数组
```

```
array([0, 1])
```

```
# a 为数组, size 为整数元组, 生成  $2 \times 3$  数组
```

```
>>> np.random.choice(np.array(['a', 'b', 'c', 'f']), (2, 3))
```

```
array([[ 'c', 'a', 'f'],
       [ 'b', 'b', 'f']], dtype='<U1')
```

```
#生成 shape=3 的一维数组, 元素取值为 1 或 2 的随机数
```

```
>>> np.random.choice(5, 3, p=[0, 0.5, 0.5, 0, 0])
```

```
array([2, 1, 1], dtype=int64)
```

7.3.2 随机分布

(1) **binomial(n, p, size=None)**: 产生 size 个二项分布的样本值, n 表示 n 次试验, p 表示每次实验发生的概率, 其中 $n > 0$ 且 p 在区间[0,1]中。

```
>>> from numpy import random
>>> n, p = 10, 0.6
>>> random.binomial(n, p, size=20) #取样 20 个, 每个值为 10 次实验中发
生的次数
array([4, 5, 6, 7, 6, 7, 5, 7, 7, 6, 5, 7, 8, 7, 8, 5, 7, 5, 6, 2])
(2) normal(loc=0.0, scale=1.0, size=None): 产生 size 个正态 (高斯) 分布
的样本值。
#从某一正态分布 (由均值和标准差标识) 中获得样本
>>> from numpy import random
>>> import numpy as np
>>> mu, sigma = 0, 1
>>> s = random.normal(loc=mu, scale=sigma, size=1000) #获取 1000
个样本值
#绘制样本的直方图, 以及概率密度函数
>>> import matplotlib.pyplot as plt
>>> count, bins, patches = plt.hist(s, 30, density=True)
>>> plt.plot(bins, 1/(sigma * np.sqrt(2 * np.pi))*np.exp( - (bins -
mu)**2/(2 * sigma**2)), linewidth=2, color='r')
[<matplotlib.lines.Line2D object at 0x0000000000AB745C0>]
>>> plt.show() #绘制的直方图以及概率密度函数曲线如图 8-1 所示
```

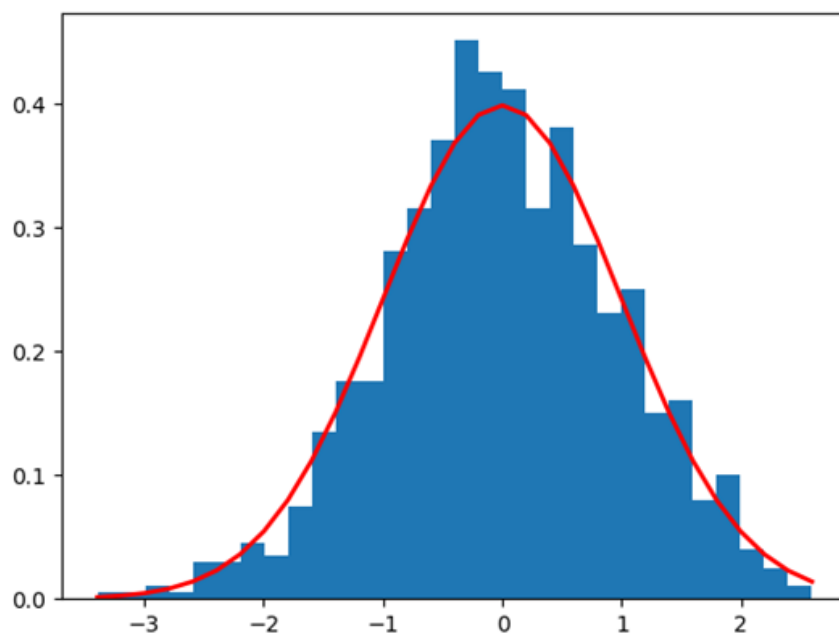


图 8-1 绘制的直方图以及概率密度函数曲线

(3) `poisson(lam=1.0, size=None)`: 从泊松分布中生成随机数, `lam` 是单位时间内事件的平均发生次数。

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> s=np.random.poisson(100, 10000)
>>> count,bins,patches = plt.hist(s,100) #绘制样本数据的直方图
>>> plt.show() #泊松分布样本数据的直方图如图 8-2 所示
```

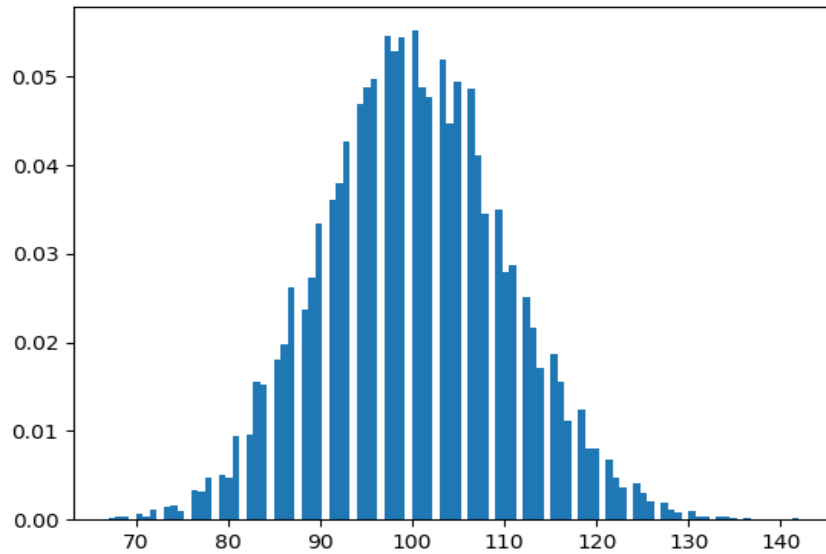


图 8-2 泊松分布样本数据的直方图

7.3.3 随机排列

(1) `shuffle(x)`: 打乱对象 `x` (多维数组按照第一维打乱), 直接在原来的数组上进行操作, 改变原来数组元素的顺序, 无返回值, `x` 可以是数组或者列表。

```
>>> import numpy as np
>>> arr = np.arange(10)
>>> np.random.shuffle(arr)
>>> arr
array([5, 8, 7, 3, 6, 4, 0, 1, 2, 9])
>>> arr1 = np.arange(9).reshape((3, 3))
>>> arr1
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> np.random.shuffle(arr1)
```

```
>>> arr1                                     #多维矩阵按照第一维打乱，以行为单位进行打乱
```

```
array([[6, 7, 8],  
       [0, 1, 2],  
       [3, 4, 5]])
```

(2) **permutation(x)**: 打乱并返回新对象（多维数组按照第一维打乱），不直接在原数组上进行操作，而是返回一个新的打乱元素顺序的数组，并不改变原来的数组。

```
>>> import numpy as np  
>>> np.random.permutation(10) #返回一个随机排列  
array([2, 3, 9, 7, 5, 8, 4, 0, 1, 6])
```

7.3.4 随机数生成器

举例 1。

```
import numpy as np  
for i in range(3):  
    np.random.seed()    # seed() 无参数  
    perm = np.random.permutation(10)  
    print(perm)
```

上述代码在 IDLE 中运行的结果如下：

```
[3 1 0 4 8 9 5 7 6 2]  
[8 3 0 4 2 1 5 9 6 7]  
[4 7 3 6 1 9 5 2 0 8]
```

举例 2。

```
import numpy as np  
for i in range(3):  
    np.random.seed(10) #每次 seed() 函数的参数值都是 10  
    perm = np.random.permutation(10)  
    print(perm)
```

上述代码在 IDLE 中运行的结果如下：

```
[8 2 5 6 3 1 0 7 4 9]  
[8 2 5 6 3 1 0 7 4 9]  
[8 2 5 6 3 1 0 7 4 9]
```

举例 3。

```
import numpy as np
for i in range(3):
    np.random.seed(i)    #每次 seed() 函数的参数值都不同
    perm = np.random.permutation(10)
    print(perm)
```

上述代码在 **IDLE** 中运行的结果如下：

```
[2 8 4 9 1 6 7 3 0 5]
[2 9 6 4 0 3 1 7 8 5]
[4 1 5 0 7 2 3 6 9 8]
```

举例 4。

```
import numpy as np
for i in range(3):
    np.random.seed(10)    #每次 seed() 函数的参数值都相同
    perm = np.random.permutation(10)
    print(perm)
    perm = np.random.permutation(10)
    print(perm)
```

上述代码在 **IDLE** 中运行的结果如下：

```
[8 2 5 6 3 1 0 7 4 9]
[5 3 4 7 6 8 9 2 1 0]
[8 2 5 6 3 1 0 7 4 9]
[5 3 4 7 6 8 9 2 1 0]
[8 2 5 6 3 1 0 7 4 9]
[5 3 4 7 6 8 9 2 1 0]
```

7.4 数组的运算

7.4.1 算术运算与函数运算

```
>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]])
>>> a
array([[1, 2, 3],
       [4, 5, 6]])
>>> a + 6    #数组 a 中的每个元素+6
array([[ 7,  8,  9],
```

```

        [10, 11, 12]))
>>> a*2      #数组 a 中的每个元素*2
array([[ 2,  4,  6],
       [ 8, 10, 12]])
>>> b=np.array([[1,2,3],[4,5,6]])
>>> a+b      #a、b 两个数组对应位置上的元素相加
array([[ 2,  4,  6],
       [ 8, 10, 12]])
>>> a-b
array([[0, 0, 0],
       [0, 0, 0]])
>>> a*b      #a、b 两个数组对应位置上的元素相乘
array([[ 1,  4,  9],
       [16, 25, 36]])
>>> a/b      #a、b 两个数组对应位置上的元素相除
array([[ 1.,  1.,  1.],
       [ 1.,  1.,  1.]])

```

此外，通过在数组和数字之间使用条件运算符，比如大于号，将会得到由布尔值组成的数组，对于原数组中满足条件的元素，布尔数组中处于同等位置的元素为 **True**。

```

>>> a=np.random.random((4,4))
>>> a
array([[ 0.87946123,  0.18813443,  0.40653531,  0.16681976],
       [ 0.47983569,  0.8154622 ,  0.60523502,  0.63313401],
       [ 0.26820056,  0.01313951,  0.40508807,  0.27654031],
       [ 0.12925672,  0.16174955,  0.54596369,  0.0266615 ]])
>>> a>0.5
array([[ True, False, False, False],
       [False,  True,  True,  True],
       [False, False, False, False],
       [False, False,  True, False]], dtype=bool)

```

直接把 `a>0.5` 的条件表达式置于方括号中，能抽取所有大于 0.5 的元素，组成一个新数组。

```

>>> b=a[a>0.5]

```



```

>>> b
array([ 0.87946123,  0.8154622 ,  0.60523502,  0.63313401,
0.54596369])

>>> import numpy as np
>>> a=np.array([[1,2,3],[4,5,6]])
>>> np.sqrt(a)          #计算各元素的平方根
array([[ 1.          ,  1.41421356,  1.73205081],
       [ 2.          ,  2.23606798,  2.44948974]])
>>> np.square(a)        #计算各元素的平方
array([[ 1,  4,  9],
       [16, 25, 36]], dtype=int32)
>>> b=np.array([[1.44,4.84,9],[4,22.5,25]])
>>> b
array([[ 1.44,  4.84,  9.  ],
       [ 4.  , 22.5 , 25.  ]])
>>> np.floor(b)         #向下取整
array([[ 1.,  4.,  9.],
       [ 4., 22., 25.]])
>>> np.ceil(b)          #向上取整
array([[ 2.,  5.,  9.],
       [ 4., 23., 25.]])
>>> np.modf(b)          #将数组的小数和整数部分以两个独立数组的形式
返回
(array([[ 0.44,  0.84,  0.  ],
       [ 0.  ,  0.5 ,  0.  ]]), array([[ 1.,  4.,  9.],
       [ 4., 22., 25.])))

(2)
>>> a=np.array([[1,2,3],[4,5,6]])
>>> b=np.array([[1,2,3],[4,5,6]])
>>> np.add(a,b)
array([[ 2,  4,  6],
       [ 8, 10, 12]])
>>> np.multiply(a,b)

```

```

array([[ 1,  4,  9],
       [16, 25, 36]])
>>> np.equal(a,b)
array([[ True,  True,  True],
       [ True,  True,  True]], dtype=bool)
>>> np.logical_and(a,b)
array([[ True,  True,  True],
       [ True,  True,  True]], dtype=bool)

```

7.4.2 统计计算

ndarray 数组对象的常用统计计算方法使用举例。

(1) **ndarray.max(axis=None, out=None)**: 返回根据指定的 **axis** 计算最大值, **axis=0** 表示求各 **column** 的最大值, **axis=1** 表示求各 **row** 的最大值, **out** 是 **ndarray** 对象, 用来存放函数返回值, 要求其 **shape** 必须与函数返回值的 **shape** 一致。

```

>>> import numpy as np
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a
array([[2, 3, 4, 9],
       [8, 7, 6, 5],
       [4, 3, 5, 8]])
>>> a.max()                                #默认将数组看成一维, 返回最大值元素
9
>>> o=np.ndarray(shape=3)
>>> a.max(axis=1,out=o)                    # axis=1 表示求各 row 的最大值
array([9., 8., 8.])
>>> print(o)
[9. 8. 8.]
>>> a.max(axis=0)                          # axis=0 表示求各列的最大值
array([8, 7, 6, 9])

```

(2) **ndarray.argmax(axis=None, out)**: 返回根据指定 **axis** 计算最大值的索引。

```

>>> a.argmax(axis=0)                      #按列求每列最大值元素的索引
array([1, 1, 1, 0], dtype=int32)
>>> a.argmax(axis=1)                      #按行求每行最大值元素的索引
array([3, 0, 3], dtype=int32)

```

```
>>> a.argmax()           #默认将数组看成一维，返回最大值元素的索引
```

3

(3) `ndarray.ptp([axis, out])`: 返回根据指定 `axis` 计算最大值与最小值的差。

```
>>> a.ptp(axis=1)         #返回按行计算每行最大值与最小值的差
array([7, 3, 5])
```

```
>>> a.ptp()              #返回整个数组中最大值与最小值的差
```

7

(4) `ndarray.clip(min, max, out)`: 返回数组元素限制在`[min, max]`之间的新数组（小于 `min` 的转为 `min`，大于 `max` 的转为 `max`）。

#返回数组元素限制在`[5, 8]`之间的新数组，小于 5 的转为 5，大于 8 的转为 8

```
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a.clip(5, 8)
array([[5, 5, 5, 8],
       [8, 7, 6, 5],
       [5, 5, 5, 8]])
```

(5) `ndarray.sum(axis=None, dtype=None, out=None)`: 返回指定 `axis` 轴的所有元素的和，默认求所有元素的和。

```
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a.sum(axis=1)         #求每行所有元素的和
array([18, 26, 20])
```

(6) `ndarray.cumsum(axis=None, dtype=None, out=None)`: 按照所给定的轴参数返回元素的累计和。

```
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a.cumsum(axis=1)      #按行求累计和
array([[ 2,  5,  9, 18],
       [ 8, 15, 21, 26],
       [ 4,  7, 12, 20]], dtype=int32)
```

(7) `ndarray.cumprod(axis=None, dtype=None, out=None)`: 返回指定轴的累积。

```
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a.cumprod(axis=1)     #得到每行元素的累积
array([[ 2,  6, 24, 216],
```

```

        [ 8, 56, 336, 1680],
        [ 4, 12, 60, 480]], dtype=int32)
(8)ndarray.all(axis, dtype, out): 根据指定 axis 判断所有元素是否全部为真。
>>> a=np.array([[2, 3, 4, 9],[8, 7, 6, 5],[4, 3, 5, 8]])
>>> a.all()
True                                #表明数组 a 的所有元素全为真

```

7.4.3 线性代数运算

```

>>> import numpy as np
>>> A=np.arange(1,10).reshape((3,3))
>>> A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
>>> B=np.ones(shape=(3,3), dtype=int) #通过 shape 指定生成 3×3 的全
为 1 的数组

```

```

>>> B
array([[1, 1, 1],
       [1, 1, 1],
       [1, 1, 1]])
>>> np.dot(A,B)                #执行两个矩阵的乘积运算
array([[ 6,  6,  6],
       [15, 15, 15],
       [24, 24, 24]])

```

注意：A、B 都是一维数组时，dot()函数返回一个标量。

```

>>> a=np.array([1,2,3])
>>> b=np.array([1,2,3])
>>> np.dot(a,b)

```

14

矩阵乘积的另外一种写法是把 dot()函数当作其中一个矩阵对象的方法。

```

>>> A.dot(B)
array([[ 6,  6,  6],
       [15, 15, 15],
       [24, 24, 24]])

```

1. 求方阵逆矩阵

设 A 是数域上的一个 n 阶矩阵, 若存在 n 阶矩阵 B 使得 $AB=BA=E$ (单位矩阵), 则称 A 是可逆矩阵, 称 B 是 A 的逆矩阵。

```
>>> import numpy as np
>>> A=np.array([[1,2,3],[1,0,-1],[0,1,1]])
>>> A
array([[ 1,  2,  3],
       [ 1,  0, -1],
       [ 0,  1,  1]])
>>> B=np.linalg.inv(A)
>>> B
array([[ 0.5,  0.5, -1. ],
       [-0.5,  0.5,  2. ],
       [ 0.5, -0.5, -1. ]])
>>> np.dot(A,B)      #检查原矩阵 A 和求得的逆矩阵 B 相乘的结果是否为
单位矩阵
array([[ 1.,  0.,  0.],
       [ 0.,  1.,  0.],
       [ 0.,  0.,  1.]])
```

2. 求解线性方程组

```
>>> import numpy as np
>>> A=np.array([[2,3,-5],[1,-2,1],[3,1,3]])
>>> A
array([[ 2,  3, -5],
       [ 1, -2,  1],
       [ 3,  1,  3]])
>>> b=np.array([3,0,7])
>>> b
array([3, 0, 7])
>>> c = np.linalg.solve(A,b)      #用 c 存储函数的返回值
>>> c                               #展示求出的解
array([ 1.42857143,  1.          ,  0.57142857])
```

3. 求解特征值和特征向量

```
>>> A=np.array([[1,2,2],[2,1,2],[2,2,1]])
```

```

>>> A
array([[1, 2, 2],
       [2, 1, 2],
       [2, 2, 1]])
>>> np.linalg.eigvals(A) #调用 eigvals 函数求解特征值
array([-1.,  5., -1.])
>>> np.linalg.eig(A)      #调用 eig 函数求解特征值和特征向量
(array([-1.,  5., -1.]), array([[ -0.81649658,  0.57735027,
0.          ],
[ 0.40824829,  0.57735027, -0.70710678],
[ 0.40824829,  0.57735027,  0.70710678]]))

```

4. 奇异值分解

```

>>> import numpy as np
>>> A=np.array([[0,1],[1,1],[1,0]])
>>> A
array([[0, 1],
       [1, 1],
       [1, 0]])
''' 使用 svd 函数分解矩阵，返回 U、 $\Sigma$  和 V 这 3 个矩阵（由数组表示），
但这里的  $\Sigma$  是由奇异值构成的一维数组，可使用 diag 函数生成完整的奇异值矩
阵'''

```

```

>>> np.linalg.svd(A)
(array([[ -0.40824829,  0.70710678,  0.57735027],
       [-0.81649658,  0.          , -0.57735027],
       [-0.40824829, -0.70710678,  0.57735027]]),
array([1.73205081, 1.          ]), array([[ -0.70710678,
-0.70710678],
       [-0.70710678,  0.70710678]]))
>>> U,Sigma,V = np.linalg.svd(A)
>>> U
array([[ -0.40824829,  0.70710678,  0.57735027],
       [-0.81649658,  0.          , -0.57735027],
       [-0.40824829, -0.70710678,  0.57735027]])
>>> V

```

```

array([[ -0.70710678, -0.70710678],
       [-0.70710678,  0.70710678]])
>>> Sigma
array([[1.73205081, 1.          ]])
>>> np.diag(Sigma)      #使用 diag 函数生成完整的奇异值矩阵，忽略
全为 0 的行
array([[1.73205081, 0.          ],
       [0.          , 1.          ]])

```

7.4.4 排序

(1) `ndarray.sort(axis=-1, kind='quicksort', order=None)`: 原地对数组元素进行排序。

```

>>> y = np.array([1, 3, 4, 9, 8, 7, 6, 5, 3, 10, 2])
>>> y.sort()  #原地排序
>>> y
array([ 1,  2,  3,  3,  4,  5,  6,  7,  8,  9, 10])
>>> y1=np.array([[0,15,10,5],[25, 22, 3, 2],[55, 45, 59, 50]])
>>> y1
array([[ 0, 15, 10,  5],
       [25, 22,  3,  2],
       [55, 45, 59, 50]])
>>> y1.sort()
>>> y1
array([[ 0,  5, 10, 15],
       [ 2,  3, 22, 25],
       [45, 50, 55, 59]])

```

(2) `ndarray.argsort(axis=-1, kind='quicksort', order=None)`: 返回对数组进行升序排序之后的数组元素在原数组中的索引。

```

>>> z = np.array([1, 3, 4, 9, 8, 7, 6, 5, 3, 10, 2])
>>> z.argsort()      #返回对数组进行升序排序之后的数组元素在原数组
中的索引
array([ 0, 10,  1,  8,  2,  7,  6,  5,  4,  3,  9], dtype=int32)

```

7.4.5 数组拼接与切分

1. 数组拼接

(1) 垂直拼接

```
>>> a = np.array([1, 2, 3])
>>> b = np.array([2, 3, 4])
>>> np.vstack((a,b))
array([[1, 2, 3],
       [2, 3, 4]])
```

可将两个数组通过 `reshape` 转换为列数相同的数组后再进行 `vstack` 拼接。

```
>>> a=np.arange(16).reshape(4,4)
>>> b=np.arange(12).reshape(3,4)
>>> np.vstack((a,b))
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

(2) 水平拼接

```
>>> a = np.array([1, 2, 3]).reshape(3,1)
>>> b = np.array([4, 5, 6]).reshape(3,1)
>>> a
array([[1],
       [2],
       [3]])
>>> b
array([[4],
       [5],
       [6]])
>>> np.hstack((a,b))
array([[1, 4],
       [2, 5],
       [3, 6]])
```


2. 数组切分

(1) 水平切分

```
>>> a=np.arange(16).reshape(4,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> x,y,z=np.hsplit(a,[2,3]) #返回三个子数组
>>> x
array([[ 0,  1],
       [ 4,  5],
       [ 8,  9],
       [12, 13]])
>>> y
array([[ 2],
       [ 6],
       [10],
       [14]])
>>> z
array([[ 3],
       [ 7],
       [11],
       [15]])
```

(2) 竖直切分

```
>>> a=np.arange(16).reshape(4,4)
>>> a
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
>>> np.vsplit(a, 2)
[array([[0, 1, 2, 3],
       [4, 5, 6, 7]]), array([[ 8,  9, 10, 11],
       [12, 13, 14, 15]])]
```

```
[12, 13, 14, 15]])]
```

7.5 读写数据文件

7.5.1 读写二进制文件

```
>>> import numpy as np
>>> A = np.arange(16).reshape(2, 8)
>>> A
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
>>> np.save("C:/workspace/Python/A.npy", A)
#如果文件路径末尾没有扩展名.npy, 系统会自动添加该扩展名
>>> B=np.load("C:/workspace/Python/A.npy") #load 用来读取二进制文件
```

```
>>> B
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
>>> import numpy as np
>>> A = np.arange(16).reshape(2, 8)
>>> B = np.arange(15).reshape(3, 5)
>>> np.savez("C:/workspace/Python/C.npz", A, B)
>>> D=np.load("C:/workspace/Python/C.npz")
>>> D['arr_0']
array([[ 0,  1,  2,  3,  4,  5,  6,  7],
       [ 8,  9, 10, 11, 12, 13, 14, 15]])
>>> D['arr_1']
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

7.5.2 读写文本文件

```
>>> a=np.arange(0, 10).reshape(2, 5)
>>> a
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
#以空格分隔将数组 a 存放到文本文件中
```

```

>>> np.savetxt("C:/workspace/Python/a.txt", a)
>>> np.loadtxt("C:/workspace/Python/a.txt")
array([[ 0.,  1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.,  9.]])
>>> b=np.arange(0,10,0.5).reshape(2,10)
#将数组元素保存为浮点数，以逗号分隔
>>>
np.savetxt("C:/workspace/Python/b.txt", b, fmt="%f", delimiter=",")
#load 时也要指定以逗号分隔，指定要读取的数据类型为浮点型
>>>
np.loadtxt("C:/workspace/Python/b.txt", dtype="f", delimiter=",")
array([[0. , 0.5, 1. , 1.5, 2. , 2.5, 3. , 3.5, 4. , 4.5],
       [5. , 5.5, 6. , 6.5, 7. , 7.5, 8. , 8.5, 9. , 9.5]],
dtype=float32)

```

实验七 pandas 库实验

一、实验目的

掌握 Series 对象的创建，Series 对象的基本运算，DataFrame 对象的创建，DataFrame 对象的元素的查看和修改，DataFrame 对象的基本运算，pandas 数据可视化，pandas 读写数据。

二、实验过程

8.1 Series 对象

8.1.1 Series 对象创建

1. 用一维 ndarray 数组创建 Series 对象

```
>>> import numpy as np    #本章中出现的 np 默认均是这个含义
>>> import pandas as pd   #本章中出现的 pd 默认均是这个含义
>>> s=pd.Series(data=np.arange(0,5,2))
>>> s
0    0
1    2
2    4
dtype: int32
```

从 Series 对象的输出可以看到，左侧是一列索引，右侧是索引对应的数据。创建 Series 对象时，若不用 index 明确指定索引，pandas 默认使用从 0 开始依次递增的整数值作为索引。

```
>>> s1=pd.Series(np.arange(0,5,2),index=['a','b','c']) #通过
index 指定索引
```

```
>>> s1
a    0
b    2
c    4
dtype: int32
```

如果想分别查看组成 Series 对象的两个数组，可通过调用它的两个属性 index（索引）和 values（数据）来得到。

```
>>> s1.values
array([0, 2, 4])
```

```
>>> s1.index  
Index(['a', 'b', 'c'], dtype='object')
```

2. 用标量值创建 Series 对象

```
>>> import pandas as pd  
>>> s2 = pd.Series(25, index = ['a', 'b', 'c'])  
>>> s2  
a      25  
b      25  
c      25  
dtype: int64
```

3. 用字典创建 Series 对象

键值对中的“键”是用来作为 Series 对象的索引，键值对中的“值”作为 Series 对象的数据。

```
>>> dict1 = {'Alice': '2341', 'Beth': '9102', 'Cecil': '3258'}  
>>> sd = pd.Series(dict1)  
>>> sd  
Alice      2341  
Beth       9102  
Cecil      3258  
dtype: object
```

4. 用列表创建 Series 对象

```
>>> import pandas as pd  
>>> s3=pd.Series([1,2,3], index = ['Java', 'C', 'Python'])  
>>> s3  
Java        1  
C           2  
Python      3  
dtype: int64
```

8.1.2 Series 对象的属性

1. shape 属性获取 Series 对象的形状

```
>>> s = pd.Series([1,3,5], index=['a', 'b', 'c'])
```

```
>>> s.shape  
(3,)
```

2. dtype 属性获取 Series 对象的数据数组中的数据的数据类型

```
>>> s.dtype  
dtype('int64')
```

3. values 属性获取 Series 对象的数据数组

```
>>> s.values  
array([1, 3, 5], dtype=int64)
```

4. index 属性获取 Series 对象的数据数组的索引

```
>>> s.index  
Index(['a', 'b', 'c'], dtype=object)
```

5. Series 对象本身及索引的 name 属性

```
>>> s.name='data' #为 Series 对象 s 命名'data'  
>>> s.name  
'data'  
>>> s.index.name='idx'  
>>> s.index.name  
'idx'
```

8.1.3 Series 对象的数据的查看和修改

1. 通过索引和切片查看 Series 对象的数据

```
>>> s=pd.Series(data=[1,2,3],index = ['Java','C','Python'])  
>>> s['C']  
2
```

可通过默认索引来读取。

```
>>> s[1]  
2
```

通过截取（切片）的方式读取多个元素。

```
>>> s[0:2]  
Java    1  
C        2
```

```
dtype: int64
```

使用多个数据对应的索引来一次读取多个元素，注意索引要放在一个列表中。

```
>>> s[['Python','C','Java']]
```

```
Python    3
```

```
C          2
```

```
Java       1
```

```
dtype: int64
```

根据筛选条件读取数据。

```
>>> s[s > 1]          #获取数据值>1 的元素
```

```
C          2
```

```
Python     3
```

```
dtype: int64
```

2. Series 对象中数据的修改

```
>>> s=pd.Series([1,2,3],index = ['Java','C','Python'])
```

```
>>> s['Python']=5
```

```
>>> s
```

```
Java       1
```

```
C          2
```

```
Python     5
```

```
dtype: int64
```

8.2 Series 对象的基本运算

8.2.1 算术运算与函数运算

1. 算术运算

```
>>> import pandas as pd
```

```
>>> s = pd.Series([2,4,6],index = ["a","b","c"])
```

```
>>> s
```

```
a      2
```

```
b      4
```

```
c      6
```

```
dtype: int64
```

```
>>> s+2
```

```
a      4
```

```

b      6
c      8
dtype: int64
>>> s*2
a      4
b      8
c     12
dtype: int64

```

2. 函数运算

```

>>> import pandas as pd
>>> import numpy as np
>>> s = pd.Series([2,4,6], index = ["a","b","c"])
>>> np.sqrt(s)           #计算各数据的平方根
a      1.414214
b      2.000000
c      2.449490
dtype: float64
>>> np.square(s)         #计算各数据的平方
a      4
b     16
c     36
dtype: int64
>>> s1 = pd.Series([2,4,6,8,2,4], index = ["a","b","c","d",'e','f'])
>>> s1
a      2
b      4
c      6
d      8
e      2
f      4
dtype: int64
>>> s1.unique()           #返回 s1 包含的不同元素
array([2, 4, 6, 8], dtype=int64)

```



```

>>> s1.count()                #返回 s1 包含的元素个数
6
>>> s1.divide(2)              #返回 s1 除以 2 的结果
a    1.0
b    2.0
c    3.0
d    4.0
e    1.0
f    2.0
dtype: float64
>>> s1.isin([2,4])            #判断给定的一系列数据[2,4]是否在 s1 中
a    True
b    True
c    False
d    False
e    True
f    True
dtype: bool
>>> s1.drop(labels=['a','c','f']) #删除 s1 中索引为 'a', 'c', 'f' 元素, s1 不变
b    4
d    8
e    2
dtype: int64

```

从返回结果可以看出：将剩下的元素作为 Series 对象返回。

8.2.2 Series 对象之间的运算

```

>>> s5=pd.Series([10,20],index=['c','d'])
>>> s6=pd.Series([2,4,6,8],index=['a','b','c','d'])
>>> s5+s6                #相同索引值的元素相加
a    NaN
b    NaN
c    16.0
d    28.0

```

```
dtype: float64
```

8.3 DataFrame 对象

8.3.1 DataFrame 对象创建

(1)

```
>>> import pandas as pd
```

```
>>>
```

```
data={'course':['C','Java','Python','Hadoop'],'scores':[82,96,92,88],  
'grade':['B','A','A','B']}
```

```
>>> df=pd.DataFrame(data)
```

```
>>> df
```

	course	grade	scores
0	C	B	82
1	Java	A	96
2	Python	A	92
3	Hadoop	B	88

(2)

```
>>> df1=pd.DataFrame (data,columns=['course','grade'])
```

```
>>> df1
```

	course	grade
0	C	B
1	Java	A
2	Python	A
3	Hadoop	B

(3)

```
>>> df2=pd.DataFrame (data,index=['一','二','三','四'])
```

```
>>> df2
```

	course	grade	scores
一	C	B	82
二	Java	A	96
三	Python	A	92
四	Hadoop	B	88

(4)

```
>>> df3=pd.DataFrame
([ [1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]], index=['一', '二', '三', '四'], columns=['A', 'B', 'C', 'D'])
```

```
>>> df3
```

	A	B	C	D
一	1	2	3	4
二	5	6	7	8
三	9	10	11	12
四	13	14	15	16

(5)

```
>>> import pandas as pd
```

```
>>>
```

```
data={ "name":{"one":"Jack", 'two':"Mary", 'three':"John", 'four':"Alice"},
```

```
      "age":{"one":10, 'two':20, 'three':30, 'four':40},
```

```
      "weight":{"one":30, 'two':40, 'three':50, 'four':65}}
```

```
>>> df=pd.DataFrame(data)
```

```
>>> df
```

	age	name	weight
four	40	Alice	65
one	10	Jack	30
three	30	John	50
two	20	Mary	40

用键值为 Series 的字典创建 DataFrame:

```
>>> import pandas as pd
```

```
>>> import numpy as np
```

```
>>> d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']), 'two' :
pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
```

```
>>> df = pd.DataFrame(d)
```

```
>>> df
```

	one	two
a	1.0	1.0
b	2.0	2.0
c	3.0	3.0

```
d NaN 4.0
```

(6)

```
>>> data1={ "name":["Jack","Mary","John"],
            "age":[10,20,30],
            "weight":[30,40,50]}
```

```
>>> df=pd.DataFrame(data1)
```

```
>>> df
```

	age	name	weight
0	10	Jack	30
1	20	Mary	40
2	30	John	50

(7)

```
>>> d = [{'one' : 1, 'two' : 1}, {'one' : 2, 'two' : 2}, {'one' : 3, 'two' : 3}, {'two' : 4}]
```

```
>>> df =
```

```
pd.DataFrame(d, index=['a','b','c','d'], columns=['one','two'])
```

```
>>> df
```

	one	two
a	1.0	1
b	2.0	2
c	3.0	3
d	NaN	4

8.3.2 DataFrame 对象的属性

```
>>> import pandas as pd
```

```
>>> df = pd.DataFrame({'city':['上海市','北京市','广州市','深圳市'],
                        'GDP':[13908.57,12406.8,9891.48,9709.02]}, columns=['city','GDP'])
```

```
>>> df
```

	city	GDP
0	上海市	13908.57
1	北京市	12406.80
2	广州市	9891.48
3	深圳市	9709.02

```

>>> df.T          #转置
              0          1          2          3
city    上海市    北京市    广州市    深圳市
GDP    13908.6  12406.8  9891.48  9709.02
>>> df.index      #查看行索引名
RangeIndex(start=0, stop=4, step=1)
>>> df.columns    #查看行列引名
Index(['city', 'GDP'], dtype='object')
>>> df.shape      #查看 DataFrame 对象的形状
(4, 2)
>>> df.values     #查看 DataFrame 对象的数据
array([[ '上海市', 13908.57],
       [ '北京市', 12406.8],
       [ '广州市', 9891.48],
       [ '深圳市', 9709.02]], dtype=object)
#要想获取 DataFrame 对象某一行内容，只需把这一行的名称作为索引
>>> df['city']
0    上海市
1    北京市
2    广州市
3    深圳市
Name: city, dtype: object
#也可以通过将列名称作为 DataFrame 对象的属性来获取该列的内容
>>> df.GDP
0    13908.57
1    12406.80
2     9891.48
3     9709.02
Name: GDP, dtype: float64
>>> df.ix[2]      #获取行号为 2 这一行的内容
city    广州市
GDP     9891.48
Name: 2, dtype: object
>>> df.ix[[2, 3]] #用一个列表指定多个索引就可获取多行的内容

```

```

    city      GDP
2  广州市  9891.48
3  深圳市  9709.02
>>> df[1:3]  #通过指定索引范围来获取多行内容，1 为起始索引值，3 为
结束索引值
    city      GDP
1  北京市  12406.80
2  广州市  9891.48
>>> df.loc[1]  #通过行索引获取行数据
city      北京市
GDP      12406.8
Name: 1, dtype: object
>>> df.size  #返回 DataFrame 对象包含的元素个数
8
#对行重新索引，然后对列重新索引
>>> df.ix[[ 3, 2, 1, 0], ['GDP', 'city']]
      GDP city
3  9709.02  深圳市
2  9891.48  广州市
1  12406.80  北京市
0  13908.57  上海市
>>> df.iloc[0:2,:2]  #获取前 2 行、前 2 列的数据
    city      GDP
0  上海市  13908.57
1  北京市  12406.80

```

8.3.3 查看和修改 DataFrame 对象的元素

1. 查看 DataFrame 对象中的元素

```

>>> import pandas as pd
>>> data={ "name":["Jack","Mary","John","Alice"],
           "age":[10,20,30,40],
           "weight":[30,40,55,65] }
>>> df=pd.DataFrame(data)
>>> df

```

	age	name	weight
0	10	Jack	30
1	20	Mary	40
2	30	John	55
3	40	Alice	65

```
>>> df['age'][1]
```

```
20
```

可以通过指定条件筛选 **DataFrame** 对象的元素。

```
>>> df[df.weight>35]    #获取 weight 大于 35 的行
```

	age	name	weight
1	20	Mary	40
2	30	John	55
3	40	Alice	65

```
>>> df[df>35]          #获取 DataFrame 对象中数值大于 35 的所有元素
```

	age	name	weight
0	NaN	Jack	NaN
1	NaN	Mary	40.0
2	NaN	John	55.0
3	40.0	Alice	65.0

2. 修改 **DataFrame** 对象中的元素

```
>>> df.index.name='id'
```

```
>>> df.columns.name='item'
```

```
>>> df
```

item	age	name	weight
id			
0	10	Jack	30
1	20	Mary	40
2	30	John	55
3	40	Alice	65

可以为 **DataFrame** 对象添加新的列，指定新列的名称，以及为新列赋值。

```
>>> df['new']=10    #添加新列，并将新列的所有元素都赋值为 10
```

```
>>> df
```

item	age	name	weight	new
------	-----	------	--------	-----

```

id
0      10   Jack      30   10
1      20   Mary      40   10
2      30   John      55   10
3      40  Alice      65   10

```

从显示结果可以看出，**DataFrame** 对象新增了名称为 “new” 的列，它的各个元素都为 10。

如果想更新一列的内容，需要把一列数赋给这一列。

```

>>> df['new']=[11,12,13,14]
>>> df

```

```

item  age  name  weight  new
id
0      10   Jack      30   11
1      20   Mary      40   12
2      30   John      55   13
3      40  Alice      65   14

```

修改单个元素的方法是：选择元素，为其赋新值即可。

```

>>> df['weight'][0]=25
>>> df

```

```

item  age  name  weight  new
id
0      10   Jack      25   11
1      20   Mary      40   12
2      30   John      55   13
3      40  Alice      65   15

```

8.3.4 判断元素是否属于 **DataFrame** 对象

可通过 **DataFrame** 对象的方法 `isin()` 判断一组元素是否属于 **DataFrame** 对象。

```

>>> df

```

```

      age  name  weight
four   40  Alice      65
one    10   Jack      30
three  30   John      50
two    20   Mary      40

```



```
>>> df.isin(['Jack', 30])
```

	age	name	weight
four	False	False	False
one	False	True	True
three	True	False	False
two	False	False	False

```
>>> df[df.isin(['Jack', 30])]
```

	age	name	weight
four	NaN	NaN	NaN
one	NaN	Jack	30.0
three	30.0	NaN	NaN
two	NaN	NaN	NaN

8.4 DataFrame 对象的基本运算

8.4.1 数据筛选

8.4.1 数据筛选二维码

```
>>> import pandas as pd
>>> import numpy as np
>>> data={'index':[1,2,3,4,5], 'year':[2012,2013,2014,2015,2016],
'status':['good','good','well','well','wonderful']}
```

```
>>> df=pd.DataFrame(data, columns=['status','year','index'],
index=['one','two','three','four','five'])
```

```
>>> df
```

	status	year	index
one	good	2012	1
two	good	2013	2
three	well	2014	3
four	well	2015	4
five	wonderful	2016	5

```
>>> df.head(3)      #获取前 3 行
```

	status	year	index
one	good	2012	1
two	good	2013	2
three	well	2014	3

```
>>> df.tail(2)      #获取后 2 行
```

```

        status year index
four      well 2015     4
five wonderful 2016     5
>>> df[2:4]    #获取 2~3 行
        status year index
three  well 2014     3
four   well 2015     4
>>> df[df['year'] > 2014]    #获取'year'列值大于 2014 的行
        status year index
four      well 2015     4
five wonderful 2016     5
>>> df.query('year > 2014')    #获取'year'列值大于 2014 的行
        status year index
four      well 2015     4
five wonderful 2016     5
>>> df.query('year%2000>index*5')    #选取满足条件的行
        status year index
one   good 2012     1
two   good 2013     2
>>> df.query('year==[2013,2014]')    #选取满足条件的行
        status year index
two    good 2013     2
three  well 2014     3
>>> df.ix[1]    #获取行号为 1 的行，采用默认的行索引
status    good
year      2013
index      2
Name: two, dtype: object
>>> df.ix['two']    #获取行索引为 two 的行，采用自定义的行索引
status    good
year      2013
index      2
Name: two, dtype: object

```

```

>>> df.ix[1:4,'year']    #获取'year'列的1~3行
two      2013
three    2014
four     2015
Name: year, dtype: int64
>>> df.ix[1:4]           #获取1~3行
      status  year  index
two     good  2013     2
three   well  2014     3
four    well  2015     4
>>> df.ix[1,2]           #获取默认行索引为1，默认列索引为2的元素
2
>>> df['year']           #返回'year'列
one      2012
two      2013
three    2014
four     2015
five     2016
Name: year, dtype: int64

```

8.4.2 数据预处理

(1) 重复行处理

① df.duplicated(subset=None, keep='first')

```

>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame({'col1': ['one', 'one', 'two', 'two', 'two',
' three', 'four'], 'col2': [3, 2, 3, 2, 3, 3, 5], 'col3': ['一', '二', '三',
', '四', '五', '六', '七']}, index=['a', 'a', 'b', 'c', 'b', 'a', 'c'])
>>> df
      col1  col2  col3
a     one     3     一
a     one     2     二
b     two     3     三
c     two     2     四

```

```

b    two    3    五
a  three    3    六
c   four    5    七

```

```
>>> df.duplicated() #默认所有列，标记重复行
```

```

a    False
a    False
b    False
c    False
b    False
a    False
c    False

```

```
dtype: bool
```

#针对'col1'列标记重复的行，subset='col1'与'col1'的效果等同

```
>>> df.duplicated('col1')
```

```

a    False
a     True
b    False
c     True
b     True
a    False
c    False

```

```
dtype: bool
```

```
>>> df.duplicated(['col1','col2']) #第5行被标记为重复
```

```

a    False
a    False
b    False
c    False
b     True
a    False
c    False

```

```
dtype: bool
```

```
>>> df.duplicated(['col1','col2'],keep='last') #第3行被标记为重
```

复

```
a    False
```

a False

b True

c False

b False

a False

c False

dtype: bool

② df.index.duplicated(keep='last')

作用：根据行索引标记重复行。

```
>>> df.index.duplicated(keep='last')#第 1、2、3、4 四被标记为重复
array([ True,  True,  True,  True, False, False, False],
      dtype=bool)
```

③ df.drop_duplicates(subset=None, keep='first', inplace=False)

>>> df.drop_duplicates('col1') #删除 df.duplicated('col1') 标记的重复记录

	col1	col2	col3
a	one	3	一
b	two	3	三
a	three	3	六
c	four	5	七

#inplace=False 返回一个删除重复行的副本

```
>>> df.drop_duplicates('col2', keep='last', inplace=False)
```

	col1	col2	col3
c	two	2	四
a	three	3	六
c	four	5	七

(2) 缺失值处理

```
>>> import numpy as np
```

```
>>> import pandas as pd
```

```
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0], [5, 4, np.nan, 1], [np.nan, np.nan, np.nan, 3], [np.nan, 5, np.nan, 4]], columns=list('ABCD'))
```

```
>>> df
```

	A	B	C	D
--	---	---	---	---

```

0 NaN 2.0 NaN 0
1 5.0 4.0 NaN 1
2 NaN NaN NaN 3
3 NaN 5.0 NaN 4

```

```
>>> df.fillna('missing') #用字符串'missing' 填充缺失值
```

```

      A      B      C D
0 missing      2 missing 0
1      5      4 missing 1
2 missing missing missing 3
3 missing      5 missing 4

```

```
>>> df.fillna(method='ffill') #ffill: 用前一个非缺失值去填充该缺失值
```

```

      A      B      C D
0 NaN 2.0 NaN 0
1 5.0 4.0 NaN 1
2 5.0 4.0 NaN 3
3 5.0 5.0 NaN 4

```

```
>>> values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
```

```
#将'A'、'B'、'C'和'D'列中的NaN元素分别替换为0、1、2和3
```

```
>>> df.fillna(value=values)
```

```

      A      B      C D
0 0.0 2.0 2.0 0
1 5.0 4.0 2.0 1
2 0.0 1.0 2.0 3
3 0.0 5.0 2.0 4

```

```
>>> df.fillna(value=values, limit=1) #替换各列的第一个NaN元素
```

```

      A      B      C D
0 0.0 2.0 2.0 0
1 5.0 4.0 NaN 1
2 NaN 1.0 NaN 3
3 NaN 5.0 NaN 4

```

(3) 删除指定的行或列

```
>>> data={'index':[1,2,3,4,5], 'year':[2012,2013,2014,2015,2016],
'status':['good','very good','well','very well','wonderful']}
```

```

>>> df=pd.DataFrame(data, columns=['status','year','index'],
index=['one','two','three','four','five'])
>>> df

```

	status	year	index
one	good	2012	1
two	very good	2013	2
three	well	2014	3
four	very well	2015	4
five	wonderful	2016	5

```

>>> df.drop('two', axis=0) #删除行索引为'two'的行

```

	status	year	index
one	good	2012	1
three	well	2014	3
four	very well	2015	4
five	wonderful	2016	5

```

>>> df.drop('year', axis=1) #删除列索引为'year'的列, 要指定
axis=1

```

	status	index
one	good	1
two	very good	2
three	well	3
four	very well	4
five	wonderful	5

```

>>> df.drop(['year','index'], axis=1) #删除两列

```

	status
one	good
two	very good
three	well
four	very well
five	wonderful

```

>>> df.drop(['one','two','three']) #删除前3行

```

	status	year	index
four	very well	2015	4
five	wonderful	2016	5

(4) 删除缺失值

```
>>> df = pd.DataFrame({"name": ['ZhangSan', 'LiSi',  
'WangWu', np.nan], "sex": [np.nan, 'male', 'female', np.nan], "age":  
[np.nan, 26, 21, np.nan]})
```

```
>>> df
```

	age	name	sex
0	NaN	ZhangSan	NaN
1	26.0	LiSi	male
2	21.0	WangWu	female
3	NaN	NaN	NaN

```
>>> df.dropna() #删除含有缺失值的行
```

	age	name	sex
1	26.0	LiSi	male
2	21.0	WangWu	female

```
>>> df.dropna(how='all') #删除全为缺失值的行
```

	age	name	sex
0	NaN	ZhangSan	NaN
1	26.0	LiSi	male
2	21.0	WangWu	female

```
>>> df.dropna(thresh=2) #删除至少含有两个缺失值的行
```

	age	name	sex
1	26.0	LiSi	male
2	21.0	WangWu	female

```
>>> df.dropna(subset=['name', 'sex']) #删除时只在'name'和'sex'列中  
查看缺失值
```

	age	name	sex
1	26.0	LiSi	male
2	21.0	WangWu	female

(5) 重新命名列名

```
>>> df.columns = ['年龄', '姓名', '性别']
```

```
>>> df
```

	年龄	姓名	性别
0	NaN	ZhangSan	NaN
1	26.0	LiSi	male


```

2    21.0    WangWu    female
3     NaN         NaN         NaN

```

(6) 重命名行名和列名

```
>>> df.rename(index={0:'A',1:'B',2:'C',3:'D'}, columns={'年龄':'
年龄 age','姓名':'姓名 name','性别':'性别 sex'})
```

```

      年龄 age    姓名 name    性别 sex
A     NaN  ZhangSan     NaN
B    26.0    LiSi     male
C    21.0    WangWu  female
D     NaN         NaN     NaN

```

(7) 重新索引

```
>>> data={'index':[1,2,3,4,5], 'year':[2012,2013,2014,2015,2016],
'status':['good','very good','well','very well','wonderful']}
```

```
>>> df=pd.DataFrame(data, columns=['status','year','index'],
index=['one','two','three','four','five'])
```

```
>>> df
```

	status	year	index
one	good	2012	1
two	very good	2013	2
three	well	2014	3
four	very well	2015	4
five	wonderful	2016	5

#对 df 重新行索引

```
>>> df.reindex(index=['two','four','one','five','three','six'],
fill_value='NaN')
```

```

      status  year  index
two  very good  2013     2
four  very well  2015     4
one    good    2012     1
five  wonderful  2016     5
three    well   2014     3
six         NaN    NaN    NaN

```

#同时改变行索引和列索引

```
>>> df.reindex(index= ['two','four','one','five','three'],
columns= ['year','status','index'])
```

	year	status	index
two	2013	very good	2
four	2015	very well	4
one	2012	good	1
five	2016	wonderful	5
three	2014	well	3

(8) 数据替换

```
>>> import numpy as np
>>> import pandas as pd
>>> df = pd.DataFrame({'A': [0, 1, 2, 3, 4], 'B': [5, 6, 7, 8, 9], 'C':
['a', 'b', 'c', 'd', 'e']})
```

```
>>> df
```

	A	B	C
0	0	5	a
1	1	6	b
2	2	7	c
3	3	8	d
4	4	9	e

#单值替换

```
>>> df.replace(0,np.nan) #用 np.nan 替换 0
```

	A	B	C
0	NaN	5	a
1	1.0	6	b
2	2.0	7	c
3	3.0	8	d
4	4.0	9	e

```
>>> df.replace([0, 1, 2, 3], np.nan) #把 df 出现在列表中的元素用
np.nan 替换
```

	A	B	C
0	NaN	5	a
1	NaN	6	b
2	NaN	7	c

```

3 NaN 8 d
4 4.0 9 e
#把 df 出现在第一个列表中的元素用第二个列表中的相应元素替换
>>> df.replace([1, 2, 3], [11, 22, 33]) #1 替换为 11, 2 替换为 22, 3
替换为 33

```

```

      A B C
0    0 5 a
1   11 6 b
2   22 7 c
3   33 8 d
4    4 9 e

```

#用字典表示 to_replace 参数，将 0 替换为 10, 1 替换为 100

```
>>> df.replace({0:10, 1:100})
```

```

      A B C
0   10 5 a
1  100 6 b
2    2 7 c
3    3 8 d
4    4 9 e

```

#将 A 列的 0、B 列的 5 替换为 100

```
>>> df.replace({'A': 0, 'B': 5}, 100)
```

```

      A B C
0  100 100 a
1    1  6 b
2    2  7 c
3    3  8 d
4    4  9 e

```

#将 A 列的 0 替换为 100、4 替换为 400

```
>>> df.replace({'A': {0: 100, 4: 400}})
```

```

      A B C
0  100 5 a
1    1 6 b
2    2 7 c
3    3 8 d

```

```

4 400 9 e
>>> df1 = pd.DataFrame({'A': ['a1e', 'a2e', 'a0e'], 'B':
['abe', 'ace', 'ade']})
>>> df1
   A    B
0 a1e  abe
1 a2e  ace
2 a0e  ade
# 'a\de' 可以匹配 'a1e'、'a2e' 等, regex=True 将 'a\de' 解释为正则表达式
# \d 匹配 0 到 9 之间的任一数字
>>> df1.replace(to_replace='a\de', value='good', regex=True) #
   A    B
0 good  abe
1 good  ace
2 good  ade
# 把 regex 匹配到的元素用 'new' 替换
>>> df1.replace(regex='a\de', value='new')
   A    B
0 new  abe
1 new  ace
2 new  ade
# 把 regex 中的正则表达式匹配到的元素都用 'good' 替换
>>> df1.replace(regex=['a\de', 'a[bcd]e'], value='good')
   A    B
0 good good
1 good good
2 good good
# 把 regex 中的两个正则表达式匹配到的元素分别用 'good' 和 'better' 替
换
>>> df1.replace(regex={'a\de': 'good', 'a[bcd]e': 'better'})
   A      B
0 good better
1 good better
2 good better

```

(9) 两个 DataFrame 对象的连接

```
>>> df1 = pd.DataFrame({'key1': ['k1', 'k1', 'k2', 'k3'], 'key2':  
['k1', 'k2', 'k1', 'k2'], 'A': ['a1', 'a2', 'a3', 'a4'], 'B': ['b1', 'b2',  
'b3', 'b4']})
```

```
>>> df2 = pd.DataFrame({'key1': ['k1', 'k2', 'k2', 'k3'], 'key2':  
['k1', 'k1', 'k1', 'k1'], 'C': ['c1', 'c2', 'c3', 'c4'], 'D': ['d1', 'd2',  
'd3', 'd4']})
```

```
>>> df1
```

	A	B	key1	key2
0	a1	b1	k1	k1
1	a2	b2	k1	k2
2	a3	b3	k2	k1
3	a4	b4	k3	k2

```
>>> df2
```

	C	D	key1	key2
0	c1	d1	k1	k1
1	c2	d2	k2	k1
2	c3	d3	k2	k1
3	c4	d4	k3	k1

#how 没指定，默认使用 inner，使用 'key1'，'key2' 作为键进行内连接

```
>>> df1.merge(df2, on=['key1', 'key2']) #只保留两个表中公共部分的  
信息
```

	A	B	key1	key2	C	D
0	a1	b1	k1	k1	c1	d1
1	a3	b3	k2	k1	c2	d2
2	a3	b3	k2	k1	c3	d3

#how='left'，只保留 df1 的所有数据

```
>>> df1.merge(df2, how='left', on=['key1', 'key2'])
```

	A	B	key1	key2	C	D
0	a1	b1	k1	k1	c1	d1
1	a2	b2	k1	k2	NaN	NaN
2	a3	b3	k2	k1	c2	d2
3	a3	b3	k2	k1	c3	d3
4	a4	b4	k3	k2	NaN	NaN

```

# how='right', 只保留 df2 的所有数据
>>> df1.merge(df2, how='right', on=['key1', 'key2'])
   A    B key1 key2   C   D
0  a1   b1   k1   k1  c1  d1
1  a3   b3   k2   k1  c2  d2
2  a3   b3   k2   k1  c3  d3
3  NaN  NaN   k3   k1  c4  d4
# how='outer', 保留 df1 和 df2 的所有数据
>>> df1.merge(df2, how='outer', on=['key1', 'key2'])
   A    B key1 key2   C   D
0  a1   b1   k1   k1  c1  d1
1  a2   b2   k1   k2  NaN  NaN
2  a3   b3   k2   k1  c2  d2
3  a3   b3   k2   k1  c3  d3
4  a4   b4   k3   k2  NaN  NaN
5  NaN  NaN   k3   k1  c4  d4
(10)
>>> import pandas as pd
>>> df1=pd.DataFrame({'lkey':['Java','C','C++','Python'],'value':
[1,2,3,4]})
>>> df2=pd.DataFrame({'rkey':['Python','Java','C','Basic'],
'value':[5,6,7,8]})
>>> df1
   lkey  value
0   Java      1
1      C      2
2   C++      3
3 Python      4
>>> df2
   rkey  value
0 Python      5
1   Java      6
2      C      7
3  Basic      8

```

```
>>> pd.merge(df1, df2, on='value') #没用公共部分，返回空 DataFrame 对象
```

```
Empty DataFrame
```

```
Columns: [lkey, value, rkey]
```

```
Index: []
```

```
>>> pd.merge(df1, df2, left_on='lkey', right_on='rkey', how='outer')
```

	lkey	value_x	rkey	value_y
0	Java	1.0	Java	6.0
1	C	2.0	C	7.0
2	C++	3.0	NaN	NaN
3	Python	4.0	Python	5.0
4	NaN	NaN	Basic	8.0

#sort='True' 表示对合并后的 DataFrame 对象，根据合并关键字按字典顺序对行排序

```
>>> pd.merge(df1, df2, left_on='lkey', right_on='rkey',  
how='outer', sort=True)
```

	lkey	value_x	rkey	value_y
0	NaN	NaN	Basic	8.0
1	C	2.0	C	7.0
2	C++	3.0	NaN	NaN
3	Java	1.0	Java	6.0
4	Python	4.0	Python	5.0

#how='inner' 内连接，只保留两个表中公共部分的信息

```
>>> pd.merge(df1, df2, left_on='lkey', right_on='rkey', how='inner')
```

	lkey	value_x	rkey	value_y
0	Java	1	Java	6
1	C	2	C	7
2	Python	4	Python	5

(11)

```
>>> import pandas as pd
```

```
>>> df1=pd.DataFrame({'A':['A0','A1','A2','A3'],'B':  
['B0','B1','B2','B3'],'C':['C0','C1','C2','C3']})
```

```
>>> df2=pd.DataFrame({'A':['A0','A4','A5'],'B':  
['B0','B4','B5'],'C':['C0','C4','C5']})
```

```

>>> df3=pd.DataFrame({'A':['A5','A6'],'B':['B5','B6'],'C':
['C5','C6']})
>>> pd.concat([df2,df3])
   A  B  C
0  A0 B0 C0
1  A4 B4 C4
2  A5 B5 C5
0  A5 B5 C5
1  A6 B6 C6
# ignore_index=True, 忽略 df2、df3 的行索引，重新索引
>>> pd.concat([df2,df3],ignore_index=True)
   A  B  C
0  A0 B0 C0
1  A4 B4 C4
2  A5 B5 C5
3  A5 B5 C5
4  A6 B6 C6
#参数 key 增加层次索引，以标识数据源自于哪张表
>>> pd.concat([df2,df3],keys=['x','y'])
   A  B  C
x 0  A0 B0 C0
  1  A4 B4 C4
  2  A5 B5 C5
y 0  A5 B5 C5
  1  A6 B6 C6
# axis=1, 横向表拼接
>>> pd.concat([df1, df2,df3], axis=1)
   A  B  C  A  B  C  A  B  C
0  A0 B0 C0 A0 B0 C0 A5 B5 C5
1  A1 B1 C1 A4 B4 C4 A6 B6 C6
2  A2 B2 C2 A5 B5 C5 NaN NaN NaN
3  A3 B3 C3 NaN NaN NaN NaN NaN NaN
#join 为'inner'时得到的是两表的交集，是'outer'时得到的是两表的并集
>>> pd.concat([df1,df2], axis=1,join='inner')

```


	A	B	C	A	B	C
0	A0	B0	C0	A0	B0	C0
1	A1	B1	C1	A4	B4	C4
2	A2	B2	C2	A5	B5	C5

```
>>> pd.concat([df1, df2], axis=1, join='outer')
```

	A	B	C	A	B	C
0	A0	B0	C0	A0	B0	C0
1	A1	B1	C1	A4	B4	C4
2	A2	B2	C2	A5	B5	C5
3	A3	B3	C3	NaN	NaN	NaN

#join_axes=[df2.index], 保留与 df2 的行索引一样的数据, 配合 axis=1 一起用

```
>>> pd.concat([df1, df2], axis=1, join_axes=[df2.index])
```

	A	B	C	A	B	C
0	A0	B0	C0	A0	B0	C0
1	A1	B1	C1	A4	B4	C4
2	A2	B2	C2	A5	B5	C5

(12) 列旋转为行

```
>>> df1 = pd.DataFrame([[69, 175], [50, 170]], index=['ZhangSan', 'LiSi'], columns=['weight', 'height']) #单层次的列, 即列名只有一层
```

```
>>> df1
```

	weight	height
ZhangSan	69	175
LiSi	50	170

```
>>> df1.stack()
```

ZhangSan	weight	69
	height	175
LiSi	weight	50
	height	170

```
dtype: int64
```

```
>>> multicol = pd.MultiIndex.from_tuples([('weight', 'kg'), ('height', 'cm')])
```

```
>>> df2=pd.DataFrame([[69, 175], [50, 170]], index=['ZhangSan', 'LiSi'], columns=multicol) #得到含有两层列名的 DataFrame 对象 df2
```

```

>>> df2
      weight height
      kg      cm
ZhangSan   69   175
LiSi       50   170
>>> df2.stack(0)    # height、weight 层转换为行
      cm      kg
ZhangSan height  175.0   NaN
      weight   NaN   69.0
LiSi     height  170.0   NaN
      weight   NaN   50.0
>>> df2.stack([0, 1]) #两个层次的列都分别转换为行
ZhangSan height cm      175.0
      weight kg       69.0
LiSi     height cm      170.0
      weight kg       50.0

dtype: float64
>>> df3=pd.DataFrame([[69, 175], [50,
170], [60,np.nan]], index=['ZhangSan',
'LiSi','WangWu'], columns=['weight', 'height'])
>>> df3.stack(dropna=False)
ZhangSan weight      69.0
      height     175.0
LiSi     weight      50.0
      height     170.0
WangWu   weight      60.0
      height      NaN

dtype: float64
>>> df3.stack(dropna=True)    #删除含有缺失值的行
ZhangSan weight      69.0
      height     175.0
LiSi     weight      50.0
      height     170.0
WangWu   weight      60.0

```

```

dtype: float64
(13) 行旋转为列
>>> import pandas as pd
>>> index = pd.MultiIndex.from_tuples([('one', 'a'), ('one',
'b'), ('two', 'a'), ('two', 'b')]) #生成多层索引标签
>>> index
MultiIndex(levels=[['one', 'two'], ['a', 'b']],
            labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
#得到含有两层行名的 DataFrame 对象 s
>>> s = pd.Series(np.arange(1.0, 5.0), index=index)
>>> s
one  a    1.0
     b    2.0
two  a    3.0
     b    4.0
dtype: float64
>>> s.unstack()           #内层行转换为列
      a    b
one  1.0  2.0
two  3.0  4.0
>>> s.unstack(level=0)   #one、two 层转换为列
      one  two
a    1.0  3.0
b    2.0  4.0

```

8.4.3 数据运算与排序

(1) df.T

作用：df 的行列转置。

```

>>> import pandas as pd
>>> d1 = {'姓名': ['李明', '王华'], '物理成绩': [89, 85], '化学成绩':
[92, 94]}
>>> df = pd.DataFrame(d1, columns=['姓名', '物理成绩', '化学成绩'])
>>> df
   姓名  物理成绩  化学成绩

```

```

0  李明      89      92
1  王华      85      94
>>> df.T

```

```

      0      1
姓名  李明  王华
物理成绩  89   85
化学成绩  92   94

```

(2) df1 + df2

作用：将 df1 和 df2 的行名和列名都相同的元素相加，其它位置的元素用 NaN 填充。

```

>>> import pandas as pd
>>> import numpy as np
>>>
df1=pd.DataFrame(np.arange(9).reshape((3,3)),columns=list('bcd'),
index=['A','B','C'])
>>>
df2=pd.DataFrame(np.arange(12).reshape((4,3)),columns=list('bde'),
index=['A','D','B','E'])
>>> df1
   b  c  d
A  0  1  2
B  3  4  5
C  6  7  8
>>> df2
   b  d  e
A  0  1  2
D  3  4  5
B  6  7  8
E  9 10 11
>>> df1+df2  #行名和列名都相同的元素相加，其它位置的元素用 NaN 填充

```

```

      b  c      d  e
A  0.0 NaN  3.0 NaN
B  9.0 NaN 12.0 NaN

```

```
C NaN NaN NaN NaN
D NaN NaN NaN NaN
E NaN NaN NaN NaN
```

(3) `df1.add(other, axis='columns', level=None, fill_value=None)`

```
>>> df1.add(10) #other 为标量 5
```

```
      b    c    d
A  10  11  12
B  13  14  15
C  16  17  18
```

```
>>> df1.add([5, 5, 5]) #other 为列表
```

```
      b    c    d
A    5    6    7
B    8    9   10
C   11   12   13
```

```
>>> df1.add(df2) #与 df1+df2 的计算结果一样
```

```
      b    c    d    e
A  0.0 NaN  3.0 NaN
B  9.0 NaN 12.0 NaN
C  NaN NaN  NaN NaN
D  NaN NaN  NaN NaN
E  NaN NaN  NaN NaN
```

```
>>> df1.add(df2, fill_value=100)
```

```
      b      c      d      e
A    0.0 101.0    3.0 102.0
B    9.0 104.0   12.0 108.0
C 106.0 107.0 108.0   NaN
D 103.0   NaN 104.0 105.0
E 109.0   NaN 110.0 111.0
```

(4) `df.apply(func,axis=0)`

```
>>>
```

```
df=pd.DataFrame(np.arange(16).reshape((4,4)),columns=list('abcd'),
index=['A','B','C','D'])
```

```
>>> df
```

```
      a    b    c    d
```

```

A    0    1    2    3
B    4    5    6    7
C    8    9   10   11
D   12   13   14   15

```

```

>>> def f(x):                #计算数组元素的取值间隔
    return x.max()-x.min()

```

```

>>> df.apply(f,axis=0)      #求每一列元素的取值间隔

```

```

a    12
b    12
c    12
d    12

```

```

dtype: int64

```

```

>>> df.apply(f,axis=1)

```

```

A     3
B     3
C     3
D     3

```

```

dtype: int64

```

apply()函数可一次执行多个函数，作用于行或列时，一次返回多个结果。

```

>>> def f1(x):
    return pd.Series([x.max(),x.min()],index=['max','min'])

```

```

>>> df.apply(f1,axis=0) #返回一个DataFrame 对象

```

```

      a    b    c    d
max  12   13   14   15
min   0    1    2    3

```

(5) df.applymap(func)

作用：将 func 函数应用到各个元素上。

```

>>> df

```

```

      a    b    c    d
A    0    1    2    3
B    4    5    6    7
C    8    9   10   11
D   12   13   14   15

```

```

>>> def f2(x):

```

```

    return 2*x+1
>>> df.applymap(f2)
      a   b   c   d
A    1   3   5   7
B    9  11  13  15
C   17  19  21  23
D   25  27  29  31

(6) df.sort_index(axis=0, ascending=True)
>>> df = pd.DataFrame({'col1' : ['A', 'A', 'B', 'D', 'C'], 'col2' :
[2, 9, 8, 7, 4], 'col3' : [1, 9, 4, 2, 3], })
>>> df
   col1  col2  col3
0    A     2     1
1    A     9     9
2    B     8     4
3    D     7     2
4    C     4     3
>>> df.sort_index(ascending=False) #按行索引进行降序排序
   col1  col2  col3
4    C     4     3
3    D     7     2
2    B     8     4
1    A     9     9
0    A     2     1
>>> df.sort_index(axis=1, ascending=False) #按列索引进行降序排列
   col3  col2  col1
0     1     2    A
1     9     9    A
2     4     8    B
3     2     7    D
4     3     4    C

(7) df.sort_values(by, axis=0, ascending=True)
axis: axis=0, 对行进行排序; axis=1, 对列进行排序。
>>> df.sort_values(by=['col1']) #按 col1 进行排序

```

	col1	col2	col3
0	A	2	1
1	A	9	9
2	B	8	4
4	C	4	3
3	D	7	2

```
>>> df.sort_values(by=['col1','col2']) #按 col1、col2 进行排序
```

	col1	col2	col3
0	A	2	1
1	A	9	9
2	B	8	4
4	C	4	3
3	D	7	2

```
(8) df.rank(axis=0,method='average',ascending=True)
```

```
>>> df =
```

```
pd.DataFrame({'a':[4,2,3,2], 'b':[15,21,10,13]}, index=['one','two',
' three',' four'])
```

```
>>> df
```

	a	b
one	4	15
two	2	21
three	3	10
four	2	13

```
>>> df.rank(method = 'first') #为每个位置分配从小到大排序后其元素
对应的序号
```

	a	b
one	4.0	3.0
two	1.0	4.0
three	3.0	1.0
four	2.0	2.0

```
#将在两个 2 这一分组的排名 1 和 2 的最大排名 2 作为两个 2 的排名
```

```
>>> df.rank(method = 'max')
```

	a	b
one	4.0	3.0


```

two      2.0  4.0
three    3.0  1.0
four     2.0  2.0
#将两个 2 这一分组的排名 1 和 2 的平均值 1.5 作为两个 2 的排名
>>> df.rank(method = 'average')
      a    b
one    4.0  3.0
two     1.5  4.0
three   3.0  1.0
four    1.5  2.0

```

8.4.4 数学统计

```

(1) df.count(axis=0,level=None)
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame({"Name":["John", "Myla", None, "John", "Myla"],
"Age": [24., np.nan, 21., 33, 26], "Grade": ['A', 'A', 'B', 'B',
'A']}, columns=["Name", "Age", "Grade"])
>>> df
   Name  Age Grade
0  John  24.0    A
1  Myla   NaN    A
2  None  21.0    B
3  John  33.0    B
4  Myla  26.0    A
>>> df.count()           #统计每列非 NaN 的元素个数
Name      4
Age       4
Grade     5
>>> df.count(axis=1)    #统计每行非 NaN 的元素个数
0      3
1      2
2      2
3      3

```

4 3

(2) `df.describe(percentiles=None,include=None,exclude=None)`

```
>>> data={'index':[1,2,3,4,5], 'year':[2012,2013,2014,2015,2016],  
'status':['good','very good','well','very well','wonderful']}
```

```
>>> df=pd.DataFrame(data, columns=['status','year','index'],  
index=['one','two','three','four','five'])
```

```
>>> df
```

	status	year	index
one	good	2012	1
two	very good	2013	2
three	well	2014	3
four	very well	2015	4
five	wonderful	2016	5

```
>>> df.describe() #一次性产生多个汇总统计
```

	year	index
count	5.000000	5.000000
mean	2014.000000	3.000000
std	1.581139	1.581139
min	2012.000000	1.000000
25%	2013.000000	2.000000
50%	2014.000000	3.000000
75%	2015.000000	4.000000
max	2016.000000	5.000000

```
>>> df.describe(include='all')
```

	status	year	index
count	5	5.000000	5.000000
unique	5	NaN	NaN
top	very good	NaN	NaN
freq	1	NaN	NaN
mean	NaN	2014.000000	3.000000
std	NaN	1.581139	1.581139
min	NaN	2012.000000	1.000000
25%	NaN	2013.000000	2.000000
50%	NaN	2014.000000	3.000000

```
75%          NaN  2015.000000  4.000000
```

```
max          NaN  2016.000000  5.000000
```

```
(3) df.sum(axis=None, skipna=None, level=None)
```

```
>>> df.sum()      #默认返回 df 对象每列元素的和
```

```
status      goodvery goodwellvery wellwonderful
```

```
year                                               10070
```

```
index                                              15
```

```
dtype: object
```

```
>>> df.sum(axis=1)  #返回 df 对象每行可求和元素的和
```

```
one      2013      #1 和 2012 的和
```

```
two      2015
```

```
three    2017
```

```
four     2019
```

```
five     2021      #5 和 2016 的和
```

```
(4) df.max(axis=0)
```

```
>>>
```

```
df1=pd.DataFrame(np.arange(20).reshape(5,4), index=list('abcde'),  
columns=['one', 'two', 'three', 'four'])
```

```
>>> df1
```

```
      one  two  three  four
```

```
a      0   1     2     3
```

```
b      4   5     6     7
```

```
c      8   9    10    11
```

```
d     12  13    14    15
```

```
e     16  17    18    19
```

```
>>> df1.max(axis=1)  #返回每行的最大值
```

```
a      3
```

```
b      7
```

```
c     11
```

```
d     15
```

```
e     19
```

```
dtype: int32
```

```
>>> df1.max(axis=0)  #返回每列的最大值
```

```
one     16
```

```
two      17
three    18
four     19
dtype: int32
```

(5) `df.var(axis=None, skipna=None, level=None)`

作用：`df.var()`返回指定轴上的元素值的均方差。

```
>>> df1.var(axis=0)          #按列计算方差
one      40.0
two      40.0
three    40.0
four     40.0
dtype: float64
```

```
>>> df1.var(axis=1)          #按行计算方差
```

```
a      1.666667
b      1.666667
c      1.666667
d      1.666667
e      1.666667
dtype: float64
```

(6) `df.std(axis=None, skipna=None, level=None)`

作用：`df.std()`返回指定轴上的元素值的标准差。

```
>>> df1.std()
one      6.324555
two      6.324555
three    6.324555
four     6.324555
dtype: float64
```

(7) `df.cov()`

```
>>> df2 = pd.DataFrame([(1, 2), (0, 3), (2, 0), (1,
1)], columns=['dogs', 'cats'])
```

```
>>> df2
   dogs  cats
0     1     2
1     0     3
```

```

2      2      0
3      1      1
>>> df2.cov() #计算 df2 的列列间的协方差
           dogs      cats
dogs  0.666667 -1.000000
cats -1.000000  1.666667
(8) df.corr(method='pearson')
>>> import pandas as pd
>>> data = pd.DataFrame({ 'x':[0,1,2,4,7,10], 'y':[0,3,2,4,5,7],
's':[0,1,2,3,4,5], 'c':[5,4,3,2,1,0]}, index =
['p1','p2','p3','p4','p5','p6'])
>>> data
      c  s  x  y
p1  5  0  0  0
p2  4  1  1  3
p3  3  2  2  2
p4  2  3  4  4
p5  1  4  7  5
p6  0  5 10  7
>>> data.corr() #计算 pearson 相关系数
           c      s      x      y
c  1.000000 -1.000000 -0.972598 -0.946256
s -1.000000  1.000000  0.972598  0.946256
x -0.972598  0.972598  1.000000  0.941729
y -0.946256  0.946256  0.941729  1.000000
(9) df.corrwith(other, axis=0)
>>> import pandas as pd
>>> df2 =
pd.DataFrame({'a':[1,2,3,6], 'b':[5,6,8,10], 'c':[9,10,12,13],
'd':[13,14,15,18]})
>>> df2
      a  b  c  d
0  1  5  9 13
1  2  6 10 14

```

```

2  3   8  12  15
3  6  10  13  18
>>> df2.corrwith(pd.Series([1, 2, 3, 4]))#计算 df2 的列与 Series 对象之
间的相关性
a    0.956183
b    0.989778
c    0.989949
d    0.956183
dtype: float64
>>> df3 =
pd.DataFrame({'a': [1, 2, 3, 4], 'b': [5, 6, 7, 8], 'c': [9, 10, 11, 12],
'd': [13, 14, 15, 16]})
>>> df3
   a  b  c  d
0  1  5  9 13
1  2  6 10 14
2  3  7 11 15
3  4  8 12 16
>>> df2.corrwith(df3)      #计算 df2 与 df3 列列间的相关性
a    0.956183
b    0.989778
c    0.989949
d    0.956183
dtype: float64
>>> df2.corrwith(df3, axis=1)  #传入 axis=1 可按行进行计算
0    1.000000
1    1.000000
2    0.993808
3    0.995429
dtype: float64
(10) df.cumsum(axis=0, skipna=True)
>>> df3.cumsum(axis=0)      #对列求累加和
   a  b  c  d
0  1  5  9 13

```

```

1   3  11  19  27
2   6  18  30  42
3  10  26  42  58

```

```
>>> df3.cumsum(axis=1)    #对行求累加和
```

```

      a   b   c   d
0   1   6  15  28
1   2   8  18  32
2   3  10  21  36
3   4  12  24  40

```

(11) df.cumprod(axis=None, skipna=True)

```
>>> df3.cumprod(axis=0)    #按列求累计积
```

```

      a      b      c      d
0   1      5      9     13
1   2     30     90    182
2   6    210    990   2730
3  24   1680   11880  43680

```

```
>>> df3.cumprod(axis=1)    #按行求累计积
```

```

      a   b   c   d
0   1   5  45  585
1   2  12 120 1680
2   3  21 231 3465
3   4  32 384 6144

```

8.4.5 数据分组与聚合

(1) 按列分组

```
>>> import pandas as pd
```

```
>>> from numpy import random
```

```
>>>
```

```
df=pd.DataFrame({'key1':['a','a','b','b','a'],'key2':['one','two','one',
'e',
'two','one'],'data1':random.randint(1,6,5),'data2':random.randint(1,6
,5)})
```

```
>>> df
```

```
data1 data2 key1 key2
```

```

0      5      2    a  one
1      5      4    a  two
2      3      5    b  one
3      4      3    b  two
4      1      2    a  one
>>> group = df.groupby('key1')      #按'key1'进行分组
>>> type(group)
<class 'pandas.core.groupby.DataFrameGroupBy'>
>>> group.groups
{'a': Int64Index([0, 1, 4], dtype='int64'), 'b': Int64Index([2, 3],
dtype='int64')}

```

如上所示，每个分组都指明了它所包含的行。

```

>>> for x in group:    #显示分组内容
    print(x)

```

```

('a',      data1  data2 key1 key2
0      5      2    a  one
1      5      4    a  two
4      1      2    a  one)
('b',      data1  data2 key1 key2
2      3      5    b  one
3      4      3    b  two)

```

既可依据单个列名'key1'进行为分组，也可依据多个列名进行分组。

```

>>> group1= df.groupby(['key1','key2']) #依据两个列名
['key1','key2']进行分组
>>> for x in group1:    #对数据进行迭代输出
    print(x)

```

```

(('a', 'one'),      data1  data2 key1 key2
0      5      2    a  one
4      1      2    a  one)
(('a', 'two'),      data1  data2 key1 key2
1      5      4    a  two)
(('b', 'one'),      data1  data2 key1 key2

```



```

2      3      5      b one)
(('b', 'two'),      data1 data2 key1 key2
3      4      3      b two)

```

(2) 通过字典进行分组

```

>>> people = pd.DataFrame(np.random.rand(5, 5), columns=['a', 'b',
'c', 'd', 'e'], index=['Joe', 'Steve', 'Wes', 'Jim', 'Travis'])

```

```

>>> people

```

	a	b	c	d	e
Joe	0.848172	0.974520	0.273986	0.149582	0.896612
Steve	0.827622	0.050503	0.262741	0.936242	0.470282
Wes	0.118550	0.243571	0.419201	0.617766	0.717359
Jim	0.429525	0.579634	0.895218	0.072886	0.990689
Travis	0.552533	0.715262	0.116545	0.049255	0.547945

#对列名建立字典

```

>>> mapping = {'a': 'red', 'b': 'red', 'c': 'blue', 'd': 'blue', 'e':
'red'}

```

```

>>> by_columns=people.groupby(mapping, axis=1) #依据 mapping 进行分
组

```

```

>>> for x in by_columns:      #显示分组内容
    print(x)

```

```

('blue',      c      d
Joe      0.273986  0.149582
Steve    0.262741  0.936242
Wes      0.419201  0.617766
Jim      0.895218  0.072886
Travis   0.116545  0.049255)
('red',      a      b      e
Joe      0.848172  0.974520  0.896612
Steve    0.827622  0.050503  0.470282
Wes      0.118550  0.243571  0.717359
Jim      0.429525  0.579634  0.990689
Travis   0.552533  0.715262  0.547945)

```

```

>>> by_columns.mean()      #对每行的各个分组求平均值

```

	blue	red
Joe	0.211784	0.906435
Steve	0.599492	0.449469
Wes	0.518484	0.359827
Jim	0.484052	0.666616
Travis	0.082900	0.605247

(3) 将 **groupby** 分类结果转化成字典

```
>>> list(group1)
[ (('a', 'one'),      data1  data2 key1 key2
0      5      2      a  one
4      1      2      a  one), (('a', 'two'),      data1  data2 key1
key2
1      5      4      a  two), (('b', 'one'),      data1  data2 key1
key2
2      3      5      b  one), (('b', 'two'),      data1  data2 key1
key2
3      4      3      b  two)
>>> pieces=dict(list(group1))    #将 groupby 分类结果转化成字典
>>> pieces
{('a', 'one'):      data1  data2 key1 key2
0      5      2      a  one
4      1      2      a  one, ('a', 'two'):      data1  data2 key1 key2
1      5      4      a  two, ('b', 'one'):      data1  data2 key1 key2
2      3      5      b  one, ('b', 'two'):      data1  data2 key1 key2
3      4      3      b  two}
>>> pieces[('a', 'one')]
      data1  data2 key1 key2
0      5      2      a  one
4      1      2      a  one
>>> pieces[('b', 'one')]
      data1  data2 key1 key2
2      3      5      b  one
```

(4) 按照列的数据类型进行分组

```
>>> df.dtypes
```

```

data1      int32
data2      int32
key1       object
key2       object
dtype: object
>>> group3=df.groupby(df.dtypes,axis=1) #按照列的数据类型进行分组
>>> for x in group3:    #显示分组内容
    print(x)

```

```

(dtype('int32'),    data1  data2
0      5      2
1      5      4
2      3      5
3      4      3
4      1      2)

```

```

(dtype('O'),    key1 key2
0    a  one
1    a  two
2    b  one
3    b  two
4    a  one)

```

(5) 通过函数进行分组

```

>>> student =
pd.DataFrame(np.arange(16).reshape((4,4)),columns=['a', 'b', 'c', 'd'],
index=['LiHua', 'XiaoLi', 'Jack', 'LiMing'])

```

```

>>> student
      a  b  c  d
LiHua  0  1  2  3
XiaoLi  4  5  6  7
Jack   8  9 10 11
LiMing 12 13 14 15

```

```

>>> group4=student.groupby(len)
>>> for x in group4:    #显示分组内容
    print(x)

```

```
(4,      a  b  c  d
Jack  8  9 10 11)
(5,      a  b  c  d
LiHua  0  1  2  3)
(6,      a  b  c  d
XiaoLi  4  5  6  7
LiMing 12 13 14 15)
```

(6) 按分组统计

```
>>> group.size()    #统计分组数量
key1
a      3
b      2
dtype: int64
>>> group.sum()      #求不同列的分组和
      data1  data2
key1
a          11      8
b           7      8
>>> group.count()    #求不同列的分组数量
      data1  data2  key2
key1
a           3      3      3
b           2      2      2
>>> group.mean()     #求分组不同列的平均值
      data1      data2
key1
a      3.666667  2.666667
b      3.500000  4.000000
#按 key1 进行分组，并计算 data1 列的平均值
>>> grouped = df['data1'].groupby(df['key1']).mean()
>>> print(grouped)
key1
a      3.666667
```

```
b      3.500000
```

```
Name: data1, dtype: float64
```

2. 数据聚合

(1) 在 **DataFrame** 对象的行或列上执行聚合操作

```
>>> df3 = pd.DataFrame([[1, 2, 3], [4, 5, 6], [7, 8, 9], [np.nan, np.nan, np.nan]], columns=['A', 'B', 'C'])
```

```
>>> df3
```

	A	B	C
0	1.0	2.0	3.0
1	4.0	5.0	6.0
2	7.0	8.0	9.0
3	NaN	NaN	NaN

```
>>> df3.agg(['sum', 'min']) #在 df 的各列上执行 'sum' 和 'min' 聚合操作
```

作

	A	B	C
sum	12.0	15.0	18.0
min	1.0	2.0	3.0

#在不同列上执行不同的聚合操作

```
>>> df3.agg({'A': ['sum', 'min'], 'B': ['min', 'max']})
```

	A	B
max	NaN	8.0
min	1.0	2.0
sum	12.0	NaN

```
>>> df3.agg("mean", axis=1) #在行上执行 "mean" 操作
```

0	2.0
1	5.0
2	8.0
3	NaN

```
dtype: float64
```

(2) 在 **df.groupby()** 所生成的分组上应用 **agg()**

```
>>> dict_data = {'key1': ['a', 'b', 'c', 'd', 'a', 'b', 'c', 'd'],
'key2': ['one', 'two', 'three', 'one', 'two', 'three', 'one', 'two'],
'data1': np.random.randint(1, 10, 8), 'data2': np.random.randint(1, 10, 8)}
```

```
>>> df4 = pd.DataFrame(dict_data)
```

```
>>> df4
```

	data1	data2	key1	key2
0	1	7	a	one
1	6	2	b	two
2	8	3	c	three
3	7	3	d	one
4	7	4	a	two
5	2	4	b	three
6	8	5	c	one
7	7	4	d	two

```
>>> group5=df4.groupby('key1')
>>> group5.agg('mean')
```

	data1	data2
key1		
a	4.0	5.5
b	4.0	3.0
c	8.0	4.0
d	7.0	3.5

```
>>> df4.groupby('key2').agg(['mean','sum']) #在每列上使用两个函数
```

	data1		data2	
	mean	sum	mean	sum
key2				
one	5.333333	16	5.000000	15
three	5.000000	10	3.500000	7
two	6.666667	20	3.333333	10

```
>>> group['data1','data2'].agg(['mean','sum'])#指定作用的列并用多个函数
```

	data1		data2	
	mean	sum	mean	sum
key1				
a	3.666667	11	2.666667	8
b	3.500000	7	4.000000	8

#自定义聚合函数，用来求每列的最大值与最小值的差

```
>>> def value_range(df):      #定义求每列的最大值和最小值差的函数
    return df.max()-df.min()
>>> df4.groupby('key1')['data2','data1'].agg(value_range)
      data2  data1
key1
a         3     6
b         2     4
c         2     0
d         1     0
>>> df4.groupby('key1').agg(lambda df:df.max()-df.min()) #使用匿名函数
```

```
      data1  data2
key1
a         6     3
b         4     2
c         0     2
d         0     1
```

(3) 应用 apply() 函数执行聚合操作

```
>>> df4.groupby('key2').apply(sum)
      data1  data2 key1      key2
key2
one       16    15  adc  oneoneone
three     10     7   cb  threethree
two       20    10  bad  twotwotwo
```

8.5 pandas 数据可视化

8.5.1 绘制折线图

```
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> list_1 = [[1, 3, 3, 5, 4], [11, 7, 15, 13, 9], [4, 2, 7, 9, 3],
[15, 11, 12, 6, 11]]
>>> date_range = pd.date_range(start='20180101',end='20180104')
```

```
>>> df = pd.DataFrame(list_1, index=date_range,
columns=list("abcde"))
>>> df
```

	a	b	c	d	e
2018-01-01	1	3	3	5	4
2018-01-02	11	7	15	13	9
2018-01-03	4	2	7	9	3
2018-01-04	15	11	12	6	11

```
# title='fenbu' 用来设置图片的标题, figsize=[5,5]用来设置图片尺寸大小
>>> df.plot(kind='line',figsize=[5,5],legend=True,title='fenbu')
<matplotlib.axes._subplots.AxesSubplot object at
0x000000000BFEECF8>
>>> plt.show() #显示 df.plot()绘制的'line'图如图 8-3 所示
```

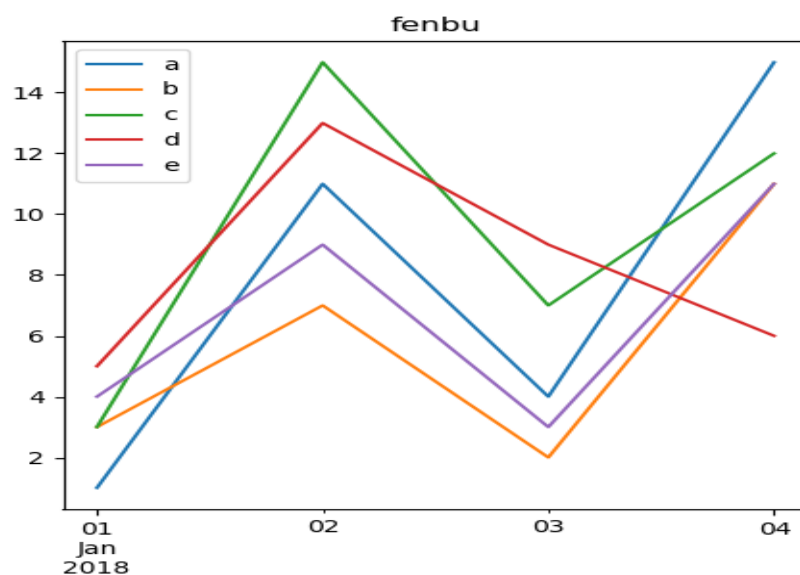


图 8-3 df.plot()绘制的'line'图

可以选择 df 中的部分列进行图像绘制, 绘制 df 的'a', 'c'列的程序代码如下所示:

```
>>>
df[['a','c']].plot(kind='line',figsize=[5,5],legend=True,title='fenbu')
<matplotlib.axes._subplots.AxesSubplot object at
0x000000000107E0438>
```



```
>>> plt.show()    #显示用 df 中的'a'、'c' 列绘制的折线图，如图 8-4 所示
```

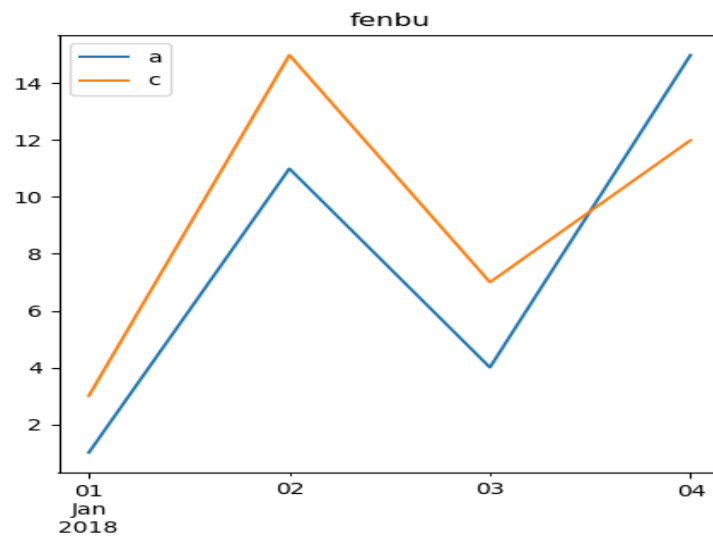


图 8-4 用 df 中的'a'、'c'列绘制的折线图

8.5.2 绘制条形图

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.rand(6, 4),
                  index=['one', 'two', 'three', 'four', 'five',
                        'six'],
                  columns=pd.Index(['A', 'B', 'C', 'D'],
                                  name='classification'))
df.plot(kind='bar', figsize=(10, 6), fontsize=15, rot=45)
plt.xlabel('classification', fontsize=15)    #添加 x 轴标签并指定标
签字体大小
plt.ylabel('sizes of the numbers', fontsize=15) # 添加 y 轴标签
plt.title('Bar', fontsize=15) #指定条形图的标题
plt.show()                                #显示绘制的垂直条形图
运行上述代码得到的条形图如图 8-5 所示。
```

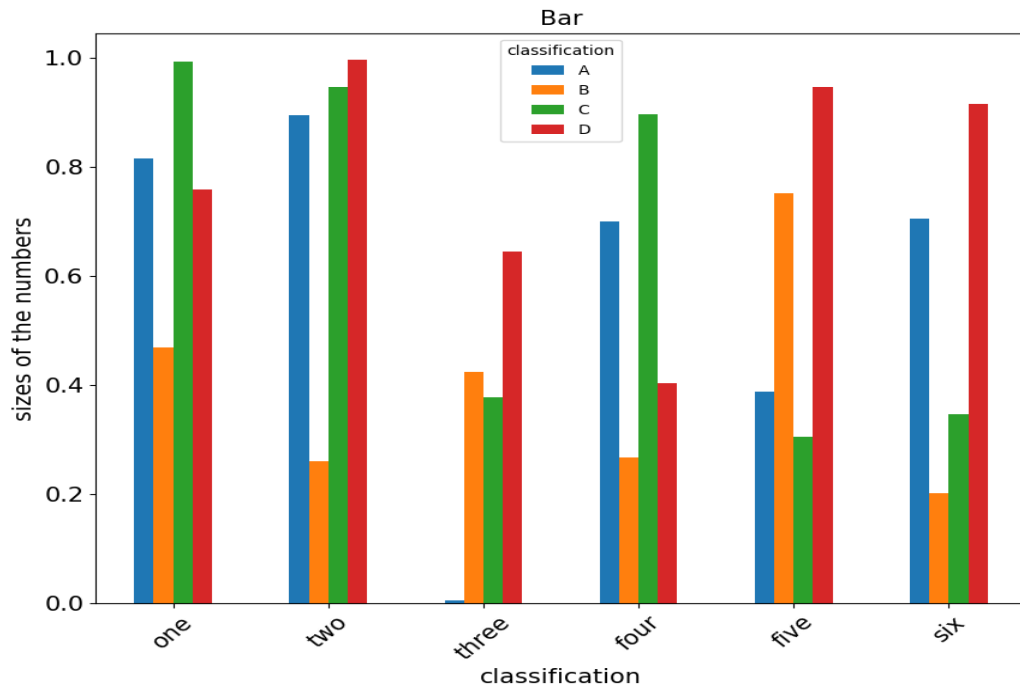


图 8-5 绘制的垂直条形图

8.5.3 绘制直方图

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

df =
pd.DataFrame({'name': ['LiHua', 'WangMing', 'ZhengLi', 'SunFei', 'ZhangFei'],
'maths': [82, 85, 88, 92, 94], 'physics': [89, 75, 83, 82, 86], 'chemistry': [86, 87, 80, 82, 92]}, columns = ['name', 'maths', 'physics', 'chemistry'])

df.plot(kind = 'hist', figsize=(10, 6), bins=10, alpha=0.8,
stacked=True, color=['coral', 'darkslateblue', 'mediumseagreen'])

#stacked=True 表示叠加直方图
plt.title('Histogram of score') #指定直方图的标题
plt.xlabel('score') #给 x 轴添加标签
plt.show()
```

运行上述代码得到的直方图如图 8-7 所示。

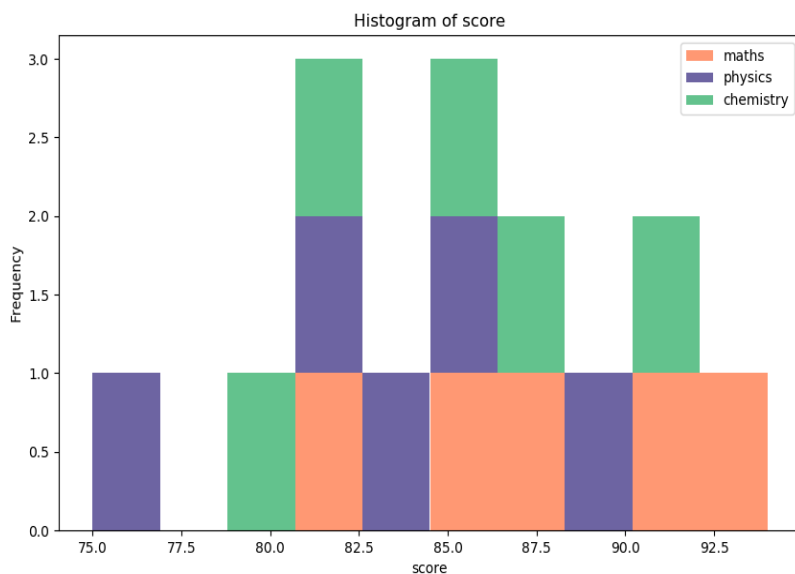


图 8-7 直方图

此外，可通过 `df.hist()` 方法为每列绘制不同的直方图，代码如下。

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

df=pd.DataFrame({'a':np.random.randn(1000)+2,'b':np.random.randn(
1000)+1,'c':np.random.randn(1000),'d':np.random.randn(1000)-1},
columns=['a', 'b', 'c','d'])

df.hist(bins=20)

plt.show()
```

8.5.4 绘制箱线图

```
>>> df=pd.DataFrame({'a':np.random.randn(1000)+2,
'b':np.random.randn(1000)+1,'c':np.random.randn(1000),'d':np.random.r
andn(1000)-1}, columns=['a', 'b', 'c','d'])

>>> df.plot(kind="box")

<matplotlib.axes._subplots.AxesSubplot object at
0x000000000C0D1CF8>

>>> plt.show()    #显示绘制的箱线图如图 8-9 所示
```

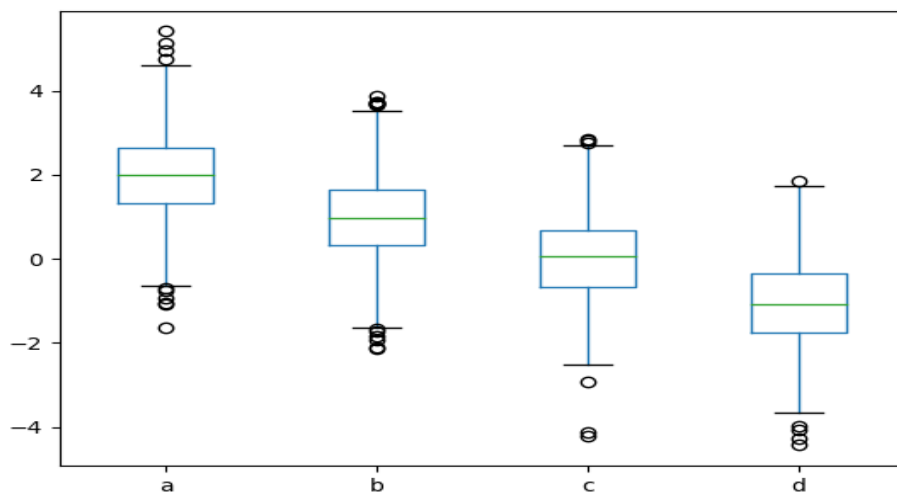


图 8-9 绘制的箱线图

8.5.5 绘制区域图

区域图是一种折线图，其中线和轴之间的区域用颜色标记为阴影，这些图表通常用于表示累计合计。

```
>>> import matplotlib.pyplot as plt
>>> import pandas as pd
>>> import numpy as np
>>> df = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c',
'd'])
>>> df
```

	a	b	c	d
0	0.721873	0.744056	0.922115	0.347943
1	0.886042	0.574766	0.132610	0.097605
2	0.674016	0.269607	0.777125	0.494149
3	0.437274	0.715205	0.291754	0.429157
4	0.436593	0.087703	0.100466	0.107748
5	0.375575	0.281472	0.362715	0.707687
6	0.366597	0.016144	0.645673	0.730062
7	0.771116	0.696150	0.611636	0.861009
8	0.589853	0.565190	0.574012	0.914027
9	0.586576	0.384761	0.081383	0.245726

```
>>> df.plot(kind='area') #生成堆积图
```

```
<matplotlib.axes._subplots.AxesSubplot object at  
0x000000000C5C0588>
```

```
>>> plt.show()      #显示绘制的堆积区域图如图 8-10 所示
```

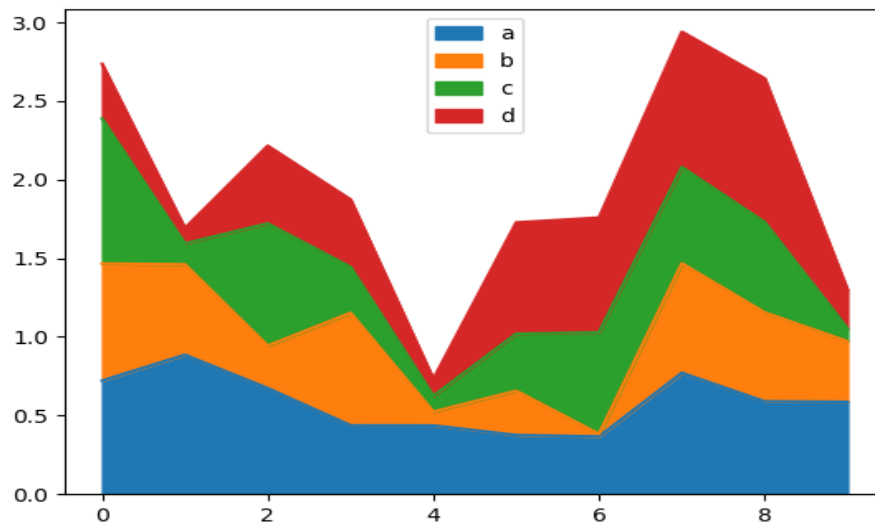


图 8-10 堆积区域图

```
>>> df.plot(kind='area', stacked=False) #生成非堆积区域图
```

```
<matplotlib.axes._subplots.AxesSubplot object at 0x0000000010C59438>
```

```
>>> plt.show()      #显示绘制的非堆积区域图如图 8-11 所示
```

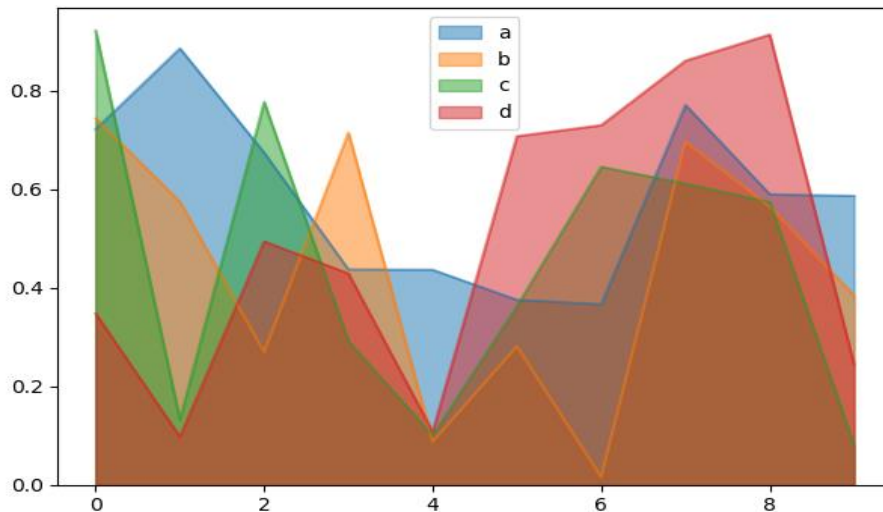


图 8-11 非堆积区域图

8.5.6 绘制散点图

```
>>> import matplotlib.pyplot as plt
```

```
>>> import pandas as pd
```

```
>>> import numpy as np
```

```
>>> df = pd.DataFrame(np.random.rand(300,2), columns=['A', 'B'])
```

```
>>> df.plot(kind='scatter', x='A', y='B')
<matplotlib.axes._subplots.AxesSubplot object at
0x00000000BFE71D0>
>>> plt.show() #显示绘制的散点图如图 8-12 所示
```

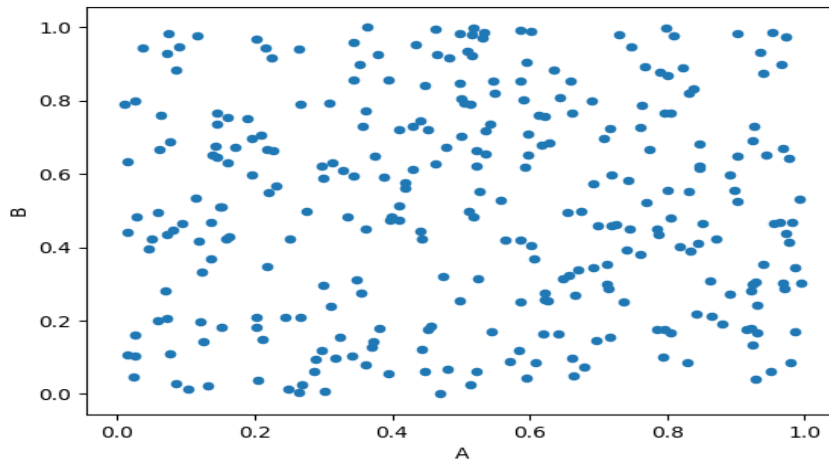


图 8-12 绘制的散点图

8.5.7 绘制饼状图

```
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> df = pd.DataFrame(3 * np.random.rand(4, 2), index=['a', 'b', 'c',
'd'], columns=['x', 'y'])
>>> df
           x           y
a  0.934451  1.415920
b  1.667728  1.467860
c  2.080061  1.875914
d  1.926433  1.688255
>>> df.plot(kind='pie', subplots=True, autopct='%2.0f%%', figsize=(8,
4))
>>> plt.show() #显示绘制的饼图如图 8-13 所示
```

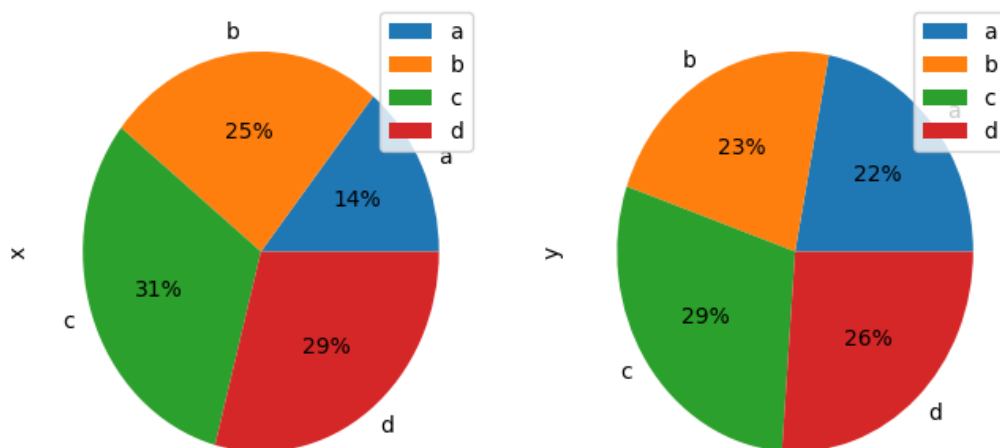


图 8-13 绘制的饼图

8.6 pandas 读写数据

8.6.1 读写 csv 文件

1. 读取 csv 文件中的数据

student.csv 文件内容如下：

Name, Math, Physics, Chemistry

WangLi, 93, 88, 90

ZhangHua, 97, 86, 92

LiMing, 84, 72, 77

ZhouBin, 97, 94, 80

```
>>> csvframe = pd.read_csv('student.csv') #从 csv 中读取数据
```

```
>>> type(csvframe)
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
>>> csvframe
```

	Name	Math	Physics	Chemistry
0	WangLi	93	88	90
1	ZhangHua	97	86	92
2	LiMing	84	72	77
3	ZhouBin	97	94	80

```
>>> pd.read_table('student.csv', sep=',')
```

	Name	Math	Physics	Chemistry
0	WangLi	93	88	90
1	ZhangHua	97	86	92

2	LiMing	84	72	77
3	ZhouBin	97	94	80

#指定 csv 文件中的行号为 0、2 的行为列标题

```
>>> csvframe = pd.read_csv('student.csv', header=[0, 2])
```

```
>>> csvframe
```

	Name	Math	Physics	Chemistry
	ZhangHua	97	86	92
0	LiMing	84	72	77
1	ZhouBin	97	94	80

```
>>> pd.read_csv('student.csv', usecols=[1, 2]) #读取第 2 列和第 3 列
```

	Math	Physics
0	93	88
1	97	86
2	84	72
3	97	94

#设置 header=0, names=['name', 'maths', 'physical', 'chemistry']实现表头定制

```
>>>
```

```
pd.read_csv('student.csv', header=0, names=['name', 'maths', 'physical', 'chemistry'])
```

	name	maths	physical	chemistry
0	WangLi	93	88	90
1	ZhangHua	97	86	92
2	LiMing	84	72	77
3	ZhouBin	97	94	80

```
>>> pd.read_csv('student.csv', index_col=[0, 1]) #指定前两列作为行索引
```

引

		Physics	Chemistry
Name	Math		
WangLi	93	88	90
ZhangHua	97	86	92
LiMing	84	72	77
ZhouBin	97	94	80

2. 往 csv 文件写入数据

```
>>> import pandas as pd
>>> date_range = pd.date_range(start="20180801", periods=4)
>>> df=pd.DataFrame({'book':[12, 13, 15, 22], 'box':[3, 8, 13, 18], 'pen':
[5, 7, 12, 15]}, index=date_range)
>>> df
```

	book	box	pen
2018-08-01	12	3	5
2018-08-02	13	8	7
2018-08-03	15	13	12
2018-08-04	22	18	15

```
>>> df.to_csv('bbp.csv') #把 df 中的数据写入默认工作目录下的 bbp.csv
文件
```

生成的 **bbp.csv** 文件的内容如下:

, book, box, pen

2018-08-01, 12, 3, 5

2018-08-02, 13, 8, 7

2018-08-03, 15, 13, 12

2018-08-04, 22, 18, 15

```
>>> df.to_csv('bbp1.csv', index=False, header=False)
```

生成的 **bbp1.csv** 文件的内容如下:

12, 3, 5

13, 8, 7

15, 13, 12

22, 18, 15

#写入时，为行索引指定列标签名

```
>>> df.to_csv("bbp2.csv", index_label="index_label")
```

bbp2.csv 文件内容:

index_label, book, box, pen

2018-08-01, 12, 3, 5

2018-08-02, 13, 8, 7

2018-08-03, 15, 13, 12

2018-08-04, 22, 18, 15

8.6.2 读取 txt 文件

1.txt 文本文件内容如下：

```
C Python Java
```

```
1 4      5
```

```
3 3      4
```

```
4 2      3
```

```
2 1      1
```

```
>>> pd.read_table('1.txt') #读取 1.txt 文本文件
```

```
C Python Java
```

```
0 1 4      5
```

```
1 3 3      4
```

```
2 4 2      3
```

```
3 2 1      1
```

```
>>> pd.read_table('1.txt', sep='\\s*')
```

```
C Python Java
```

```
0 1      4      5
```

```
1 3      3      4
```

```
2 4      2      3
```

```
3 2      1      1
```

```
>>> pd.read_table('1.txt', sep='\\s*', skiprows=[1], nrows=2)
```

```
C Python Java
```

```
0 3      3      4
```

```
1 4      2      3
```

在接下来这个例子中，2.txt 文件中数字和字母杂糅在一起，需要从中抽取数字部分，2.txt 文件的内容如下：

```
0BEGIN11NEXT22A32
```

```
1BEGIN12NEXT23A33
```

```
2BEGIN13NEXT23A34
```

2.txt 文件显然没有表头，用 read_table 读取时需要将 header 选项设置为 None。

```
>>> pd.read_table('2.txt', sep='\\D*', header=None)
```

```
0 1 2 3
```

```
0 0 11 22 32
```

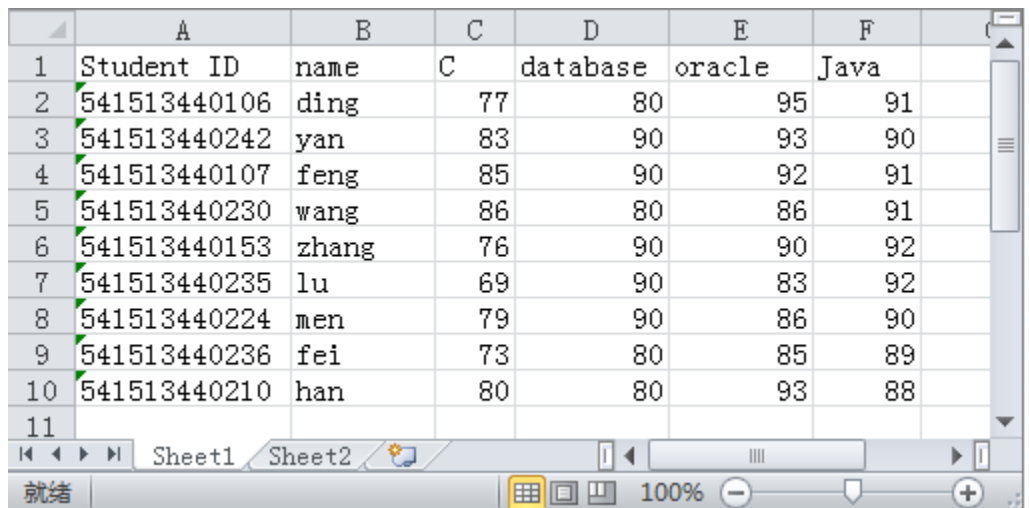
```
1 1 12 23 33
```

```
2 2 13 23 34
```

8.6.3 读写 Excel 文件

1. 读取 Excel 文件中的数据

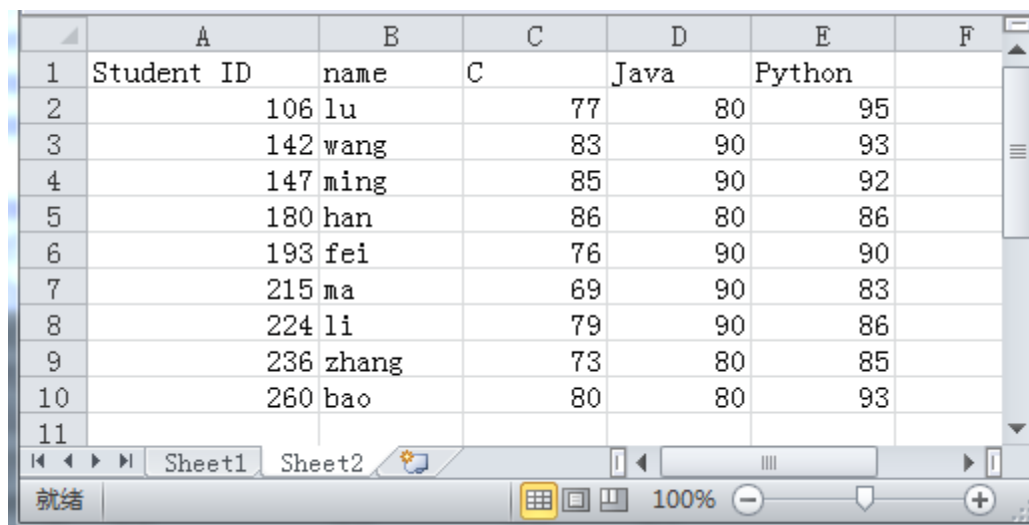
表 8-8 Sheet1 的内容

A screenshot of an Excel spreadsheet showing Sheet1. The spreadsheet has columns A through F. Row 1 contains headers: 'Student ID', 'name', 'C', 'database', 'oracle', and 'Java'. Rows 2 through 10 contain data for students with IDs 541513440106, 541513440242, 541513440107, 541513440230, 541513440153, 541513440235, 541513440224, 541513440236, and 541513440210 respectively. The 'C' column contains scores ranging from 73 to 86. The 'database' column contains scores ranging from 80 to 90. The 'oracle' column contains scores ranging from 83 to 95. The 'Java' column contains scores ranging from 88 to 93. The Excel interface shows 'Sheet1' and 'Sheet2' tabs at the bottom, with 'Sheet1' selected. The status bar at the bottom indicates '就绪' (Ready) and '100%' zoom.

	A	B	C	D	E	F
1	Student ID	name	C	database	oracle	Java
2	541513440106	ding	77	80	95	91
3	541513440242	yan	83	90	93	90
4	541513440107	feng	85	90	92	91
5	541513440230	wang	86	80	86	91
6	541513440153	zhang	76	90	90	92
7	541513440235	lu	69	90	83	92
8	541513440224	men	79	90	86	90
9	541513440236	fei	73	80	85	89
10	541513440210	han	80	80	93	88

Sheet2 的内容表 8-9 所示。

表 8-9 Sheet2 的内容

A screenshot of an Excel spreadsheet showing Sheet2. The spreadsheet has columns A through F. Row 1 contains headers: 'Student ID', 'name', 'C', 'Java', and 'Python'. Rows 2 through 10 contain data for students with IDs 106, 142, 147, 180, 193, 215, 224, 236, and 260 respectively. The 'C' column contains scores ranging from 69 to 86. The 'Java' column contains scores ranging from 80 to 90. The 'Python' column contains scores ranging from 83 to 95. The Excel interface shows 'Sheet1' and 'Sheet2' tabs at the bottom, with 'Sheet2' selected. The status bar at the bottom indicates '就绪' (Ready) and '100%' zoom.

	A	B	C	D	E	F
1	Student ID	name	C	Java	Python	
2	106	lu	77	80	95	
3	142	wang	83	90	93	
4	147	ming	85	90	92	
5	180	han	86	80	86	
6	193	fei	76	90	90	
7	215	ma	69	90	83	
8	224	li	79	90	86	
9	236	zhang	73	80	85	
10	260	bao	80	80	93	

接下来通过 pandas 的 read_excel 方法来读取 chengji.xlsx 文件。

```
>>> pd.read_excel('chengji.xlsx')
```

```
      Student ID  name  C  database  oracle  Java
0  541513440106  ding  77      80      95    91
1  541513440242   yan  83      90      93    90
2  541513440107  feng  85      90      92    91
3  541513440230  wang  86      80      86    91
4  541513440153  zhang  76      90      90    92
```

5	541513440235	lu	69	90	83	92
6	541513440224	men	79	90	86	90
7	541513440236	fei	73	80	85	89
8	541513440210	han	80	80	93	88

#将 chengji.xlsx 的列名作为所生成的 DataFrame 对象的第一行数据，并重新生成索引

```
>>> pd.read_excel('chengji.xlsx', header=None)
      0      1      2      3      4      5
0  Student ID  name  C  database  oracle  Java
1  541513440106  ding  77      80      95      91
2  541513440242   yan  83      90      93      90
3  541513440107  feng  85      90      92      91
4  541513440230  wang  86      80      86      91
5  541513440153  zhang  76      90      90      92
6  541513440235   lu   69      90      83      92
7  541513440224   men  79      90      86      90
8  541513440236   fei  73      80      85      89
9  541513440210   han  80      80      93      88
```

skiprows 指定读取数据时要忽略的行，这里忽略第 1, 2, 3 行

```
>>> pd.read_excel('chengji.xlsx', skiprows = [1, 2, 3])
      Student ID  name  C  database  oracle  Java
0  541513440230  wang  86      80      86      91
1  541513440153  zhang  76      90      90      92
2  541513440235   lu   69      90      83      92
3  541513440224   men  79      90      86      90
4  541513440236   fei  73      80      85      89
5  541513440210   han  80      80      93      88
```

skip_footer=4, 表示读取数据时忽略最后 4 行

```
>>> pd.read_excel('chengji.xlsx', skip_footer=4)
      Student ID  name  C  database  oracle  Java
0  541513440106  ding  77      80      95      91
1  541513440242   yan  83      90      93      90
2  541513440107  feng  85      90      92      91
3  541513440230  wang  86      80      86      91
```

```

4 541513440153 zhang 76          90      90      92
# index_col="Student ID"表示指定 Student ID 为行索引
>>> pd.read_excel('chengji.xlsx', skip_footer=4, index_col="Student
ID")

```

	name	C	database	oracle	Java
Student ID					
541513440106	ding	77		80	95 91
541513440242	yan	83		90	93 90
541513440107	feng	85		90	92 91
541513440230	wang	86		80	86 91
541513440153	zhang	76		90	90 92

```

#names 参数用来重新命名列名称
>>>

```

```

pd.read_excel('chengji.xlsx', skip_footer=5, names=["a", "b", "c", "d", "e",
"f"])

```

	a	b	c	d	e	f
0	541513440106	ding	77	80	95	91
1	541513440242	yan	83	90	93	90
2	541513440107	feng	85	90	92	91
3	541513440230	wang	86	80	86	91

```

# sheet_name= [0,1]表示同时读取 Sheet1 和 Sheet2

```

```

>>> pd.read_excel('chengji.xlsx', skip_footer=5, sheet_name= [0,1])
OrderedDict([(0,      Student ID  name    C  database  oracle

```

Java

0	541513440106	ding	77		80	95	91
1	541513440242	yan	83		90	93	90
2	541513440107	feng	85		90	92	91
3	541513440230	wang	86		80	86	91), (1,

Student ID name C Java Python

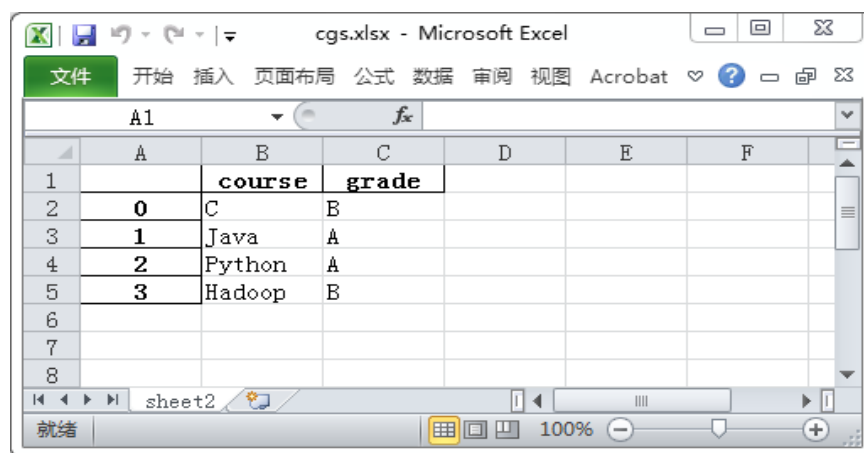
0	106	lu	77	80	95
1	142	wang	83	90	93
2	147	ming	85	90	92
3	180	han	86	80	86)]

2. 往 Excel 文件写入数据

```
>>>
df=pd.DataFrame({'course':['C','Java','Python','Hadoop'],'scores':
[82,96,92,88], 'grade':['B','A','A','B']})
>>> df
      course grade  scores
0         C     B      82
1      Java     A      96
2    Python     A      92
3    Hadoop     B      88
'''sheet_name="sheet2"表示将 df 存储在 Excel 的 sheet2 页面, columns
=["course","grade"]表示选择"course","grade"两列进行输出'''
>>>
df.to_excel(excel_writer='cgs.xlsx',sheet_name="sheet2",columns
=["course","grade"])
```

生成的 cgs.xlsx 文件表其内容如表 8-10 所示。

表 8-10 生成的 cgs.xlsx 文件表



	A	B	C	D	E	F
1		course	grade			
2	0	C	B			
3	1	Java	A			
4	2	Python	A			
5	3	Hadoop	B			
6						
7						
8						

8.7 筛选和排序数据实例

```
>>> import pandas as pd
>>> df=pd.DataFrame(pd.read_excel('chengji.xlsx'))
>>> df
```

	Student ID	name	C	database	oracle	Java
0	541513440106	ding	77	80	95	91
1	541513440242	yan	83	90	93	90
2	541513440107	feng	85	90	92	91
3	541513440230	wang	86	80	86	91

4	541513440153	zhang	76	90	90	92
5	541513440235	lu	69	90	83	92
6	541513440224	men	79	90	86	90
7	541513440236	fei	73	80	85	89
8	541513440210	han	80	80	93	88

创建 DataFrame 对象 df 后，使用 df 的 sort_values() 方法对 df 的数据进行排序操作，

```
>>> df.sort_values(by='Student ID', ascending=True) #按 Student ID 升序排序
```

	Student ID	name	C	database	oracle	Java
0	541513440106	ding	77	80	95	91
2	541513440107	feng	85	90	92	91
4	541513440153	zhang	76	90	90	92
8	541513440210	han	80	80	93	88
6	541513440224	men	79	90	86	90
3	541513440230	wang	86	80	86	91
5	541513440235	lu	69	90	83	92
7	541513440236	fei	73	80	85	89
1	541513440242	yan	83	90	93	90

除了对单列数据进行排序以外，sort_values 方法还可以对多列数据进行排序操作。下面我们对 database 和 Java 列进行升序排列，以下是具体的代码和排序结果，与单列数据排序的代码相比，将包含两个列名称['database', 'Java']列表赋值给 by。

```
>>> df.sort_values(by=['database', 'Java']) #按 database 和 Java 进行升序排序
```

	Student ID	name	C	database	oracle	Java
8	541513440210	han	80	80	93	88
7	541513440236	fei	73	80	85	89
0	541513440106	ding	77	80	95	91
3	541513440230	wang	86	80	86	91
1	541513440242	yan	83	90	93	90
6	541513440224	men	79	90	86	90
2	541513440107	feng	85	90	92	91
4	541513440153	zhang	76	90	90	92

```
5 541513440235 lu 69 90 83 92
```

在完成了对数据表排序的操作后，我们可以对数据表进行简单的筛选，例如获取 C 分数最小的前 5 名数据。具体的方法是先对 df 数据表按 C 升序排列，然后取前 5 名的数据。与前面单列升序排列的代码相比只在结尾增加了 head() 方法。

```
>>> df.sort_values(by='C').head(5) #获取 C 分数最小的前 5 名数据
```

	Student ID	name	C	database	data structure	oracle	Java
5	541513440235	lu	69	90		88	83
7	541513440236	fei	73	80		87	85
4	541513440153	zhang	76	90		85	90
0	541513440106	ding	77	80		92	95
6	541513440224	men	79	90		83	86

```
>>> df.query('C > 80').head(2) #获取 C 分数大于 80 的最小的前 2 名数据
```

	Student ID	name	C	database	oracle	Java
1	541513440242	yan	83	90	93	90
2	541513440107	feng	85	90	92	91

#获取 C 分数大于 80 的最小的前 3 名数据，显示 Student ID、C、oracle、Java 列

```
>>> df[['Student ID', 'C', 'oracle', 'Java']].query('C > 80').head(3)
```

	Student ID	C	oracle	Java
1	541513440242	83	93	90
2	541513440107	85	92	91
3	541513440230	86	86	91

实验八 数据预处理实验

一、实验目的

掌握数据清洗、数据集成、数据规范化、数据离散化、数据规约、数据降维。

二、实验过程

10.1 数据清洗

10.1.1 处理缺失值

① 直线拟合

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> x = np.linspace(100, 200, 30) #返回 30 个[100, 200]内均匀间隔的数字序列
#random_integers(5, 20, 30)生成[5, 20]上离散均匀分布的 30 个整数值
>>> y = x + np.random.random_integers(5, 20, 30)
>>> p = np.polyfit(x, y, deg=1) #直线拟合, p 为一次多项式的系数
>>> p
array([ 1.05535484,  4.83010753])
>>> q = np.polyval(p, x) #计算 p 所指定的一次多项式在 x 处的函数值
>>> plt.plot(x, y, 'o') #o 表示数据点用实心圈标记
[<matplotlib.lines.Line2D object at 0x000000000ED29048>]
>>> plt.plot(x, q, 'k') #绘制拟合的直线
[<matplotlib.lines.Line2D object at 0x000000000B5726A0>]
>>> plt.show() #显示直线拟合的绘图结果如图 10-1 所示
```

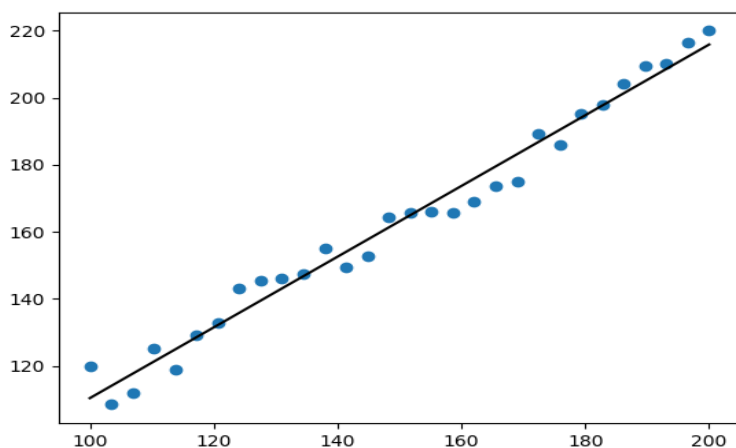


图 10-1 直线拟合的绘图结果

② 抛物线拟合

```
>>> p1 = np.polyfit(x,y,deg=2) #抛物线拟合
```

```
>>> q1 = np.polyval(p1, x)      #计算 p1 所对应的二次多项式在 x 处的函数
```

值

```
>>> plt.plot(x, y, 'o')
```

```
[<matplotlib.lines.Line2D object at 0x000000000EF74FD0>]
```

```
>>> plt.plot(x, q1, 'k')
```

```
[<matplotlib.lines.Line2D object at 0x000000000DA5F908>]
```

```
>>> plt.show()      #显示抛物线拟合的绘图结果如图 10-2 所示
```

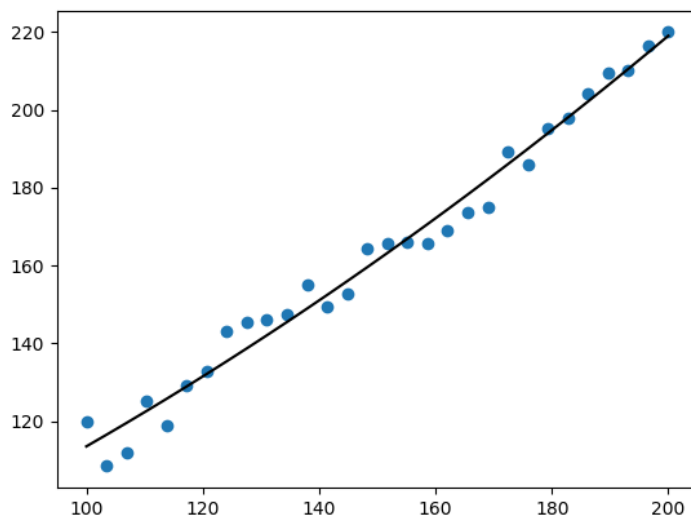


图 10-2 抛物线拟合的绘图结果

③ 3 阶多项式拟合

下面给出进行 3 阶多项式拟合的程序代码：

```
>>> p2 = np.polyfit(x, y, deg=3)  #3 阶多项式拟合
>>> q2 = np.polyval(p2, x)        #计算 p2 所指定的三次多项式在 x 处的函数
值
>>> plt.plot(x, y, 'o')
[<matplotlib.lines.Line2D object at 0x000000000EFE1B38>]
>>> plt.plot(x, q2, 'k')
[<matplotlib.lines.Line2D object at 0x000000000E8C9B00>]
>>> plt.show()                    #显示 3 阶多项式拟合的绘图结果如图 10-3 所示
```

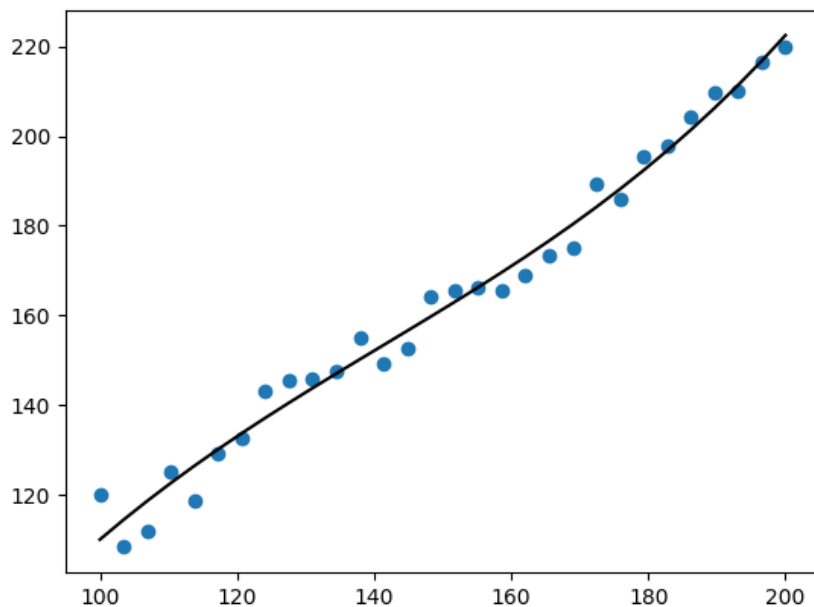


图 10-3 3 阶多项式拟合的绘图结果

(2) 各种函数的拟合

① e 的 b/x 次方拟合

第 1 步，定义需要拟合的函数，如：

```
def func(x, a, b):
    return a*np.exp(b/x)
```

第 2 步，进行函数拟合，获取 popt 里面的拟合系数：

```
popt, pcov = curve_fit(func, x, y) #进行函数拟合
```

得到的拟合系数存储在 popt 中，a 的值存储在 popt[0] 中，b 的值存储在 popt[1] 中。pcov 存储的是最优参数的协方差估计矩阵。

```
>>> import numpy as np
```

```

>>> import matplotlib.pyplot as plt
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b):
    return a*np.exp(b/x)
>>> x = np.arange(1, 11, 1)          #定义 x、y 散点坐标
>>> y = np.array([3.98, 5.1, 5.85, 6.4, 7.4, 8.6, 10, 10.2, 13.1, 14.5])
>>> popt, pcov = curve_fit(func, x, y) #函数拟合
>>> a = popt[0]                      #获取 popt 里面的拟合系数
>>> b = popt[1]
>>> y1 = func(x,a,b)                 #获取拟合值
>>> print('系数 a:', a)
系数 a: 16.036555526
>>> print('系数 b:', b)
系数 b: -2.9088756676
>>> plt.plot(x, y, 'o',label='original values') #绘制(x、y)点
[<matplotlib.lines.Line2D object at 0x00000000143064E0>]
>>> plt.plot(x, y1, 'k',label='polyfit values') #绘制拟合曲线
[<matplotlib.lines.Line2D object at 0x000000000EFA8D68>]
>>> plt.xlabel('x')
Text(0.5,0,'x')
>>> plt.ylabel('y')
Text(0.5,0,'y')
>>> plt.title('curve_fit')
Text(0.5,1,'curve_fit')
>>> plt.legend(loc=4) #指定 legend 的位置在右下角
<matplotlib.legend.Legend object at 0x0000000014306F98>
>>> plt.show() #显示 e 的 b/x 次方拟合的绘图结果如图 10-4 所示

```

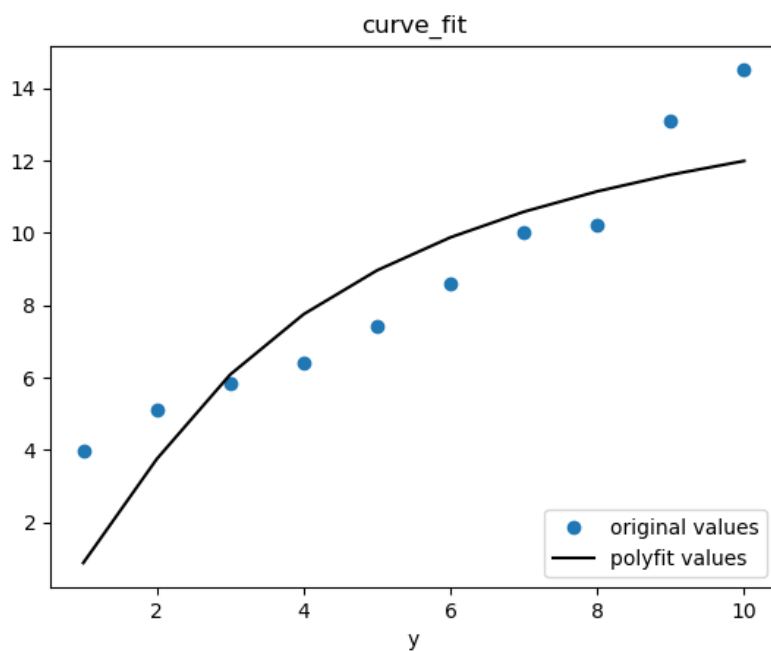


图 10-4 e 的 b/x 次方拟合的绘图结果

② $a * e^{b/x} + c$ 的拟合

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> from scipy.optimize import curve_fit
>>> def func(x, a, b, c):
    return a * np.exp(-b * x) + c
>>> x = np.linspace(0, 4, 50)
>>> y = func(x, 2.5, 1.3, 0.5)
>>> y1 = y + 0.2 * np.random.normal(size=len(x)) #为数据加入噪声
>>> plt.plot(x, y1, 'o', label='original values')
[<matplotlib.lines.Line2D object at 0x0000000013B9E710>]
>>> popt, pcov = curve_fit(func, x, y1)
>>> plt.plot(x, func(x, *popt), 'k--', label='fit')
[<matplotlib.lines.Line2D object at 0x000000000ED56B70>]
>>> plt.xlabel('x')
Text(0.5,0,'x')
>>> plt.ylabel('y')
Text(0,0.5,'y')
>>> plt.legend()
<matplotlib.legend.Legend object at 0x0000000013B9EDD8>
```

>>> plt.show() #显示 $a \cdot e^{b/x} + c$ 的拟合的绘图结果如图 10-5 所示

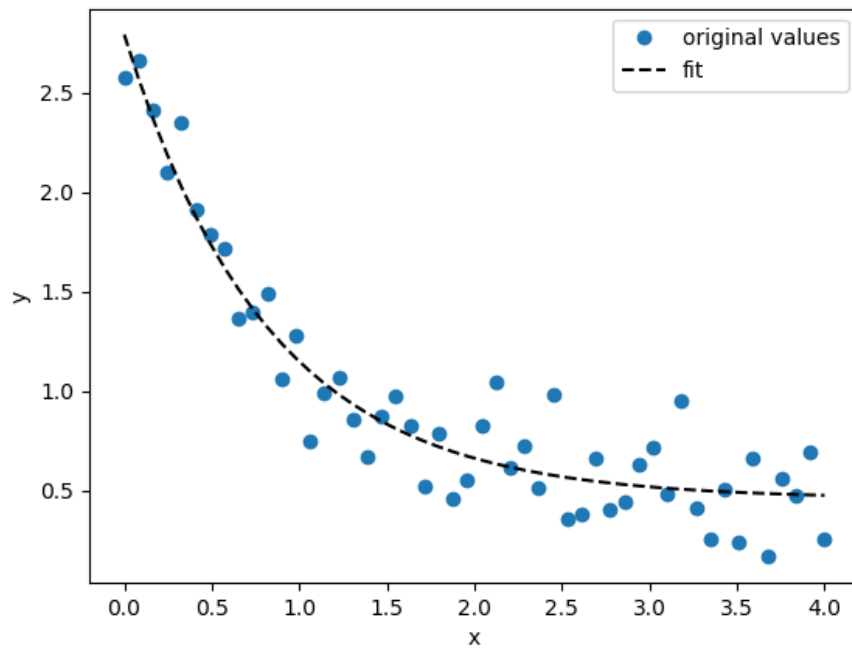


图 10-5 $a \cdot e^{b/x} + c$ 的拟合的绘图结果

③ $a \cdot \sin(x) + b$ 的拟合

```
import numpy as np
from matplotlib import pyplot as plt
from scipy.optimize import curve_fit
def f(x):
    return 2*np.sin(x)+3
def f_fit(x,a,b):
    return a*np.sin(x)+b
x=np.linspace(-2*np.pi,2*np.pi)
y=f(x)+0.5*np.random.randn(len(x))    #加入了噪声
popt,pcov=curve_fit(f_fit,x,y)        #曲线拟合
print('最优参数:',popt)                #最优参数
print(pcov)                            #输出最优参数的协方差估计矩
```

阵

```
a = popt[0]
b = popt[1]
y1 = f_fit(x,a,b) #获取拟合值
plt.plot(x,f(x),'r',label='original')
```

```
plt.scatter(x,y,c='g',label='original values') #散点图
plt.plot(x,y1,'b--',label='fitting')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show() #显示绘制  $a*\sin(x)+b$  的拟合图如图
```

10-6 所示

运行上述代码得到的输出结果如下：

最优参数：[1.95980894 2.96039244]

最优参数的协方差估计矩阵：

```
[[1.19127360e-02 4.32976874e-13]
```

```
[4.32976874e-13 5.83724065e-03]]
```

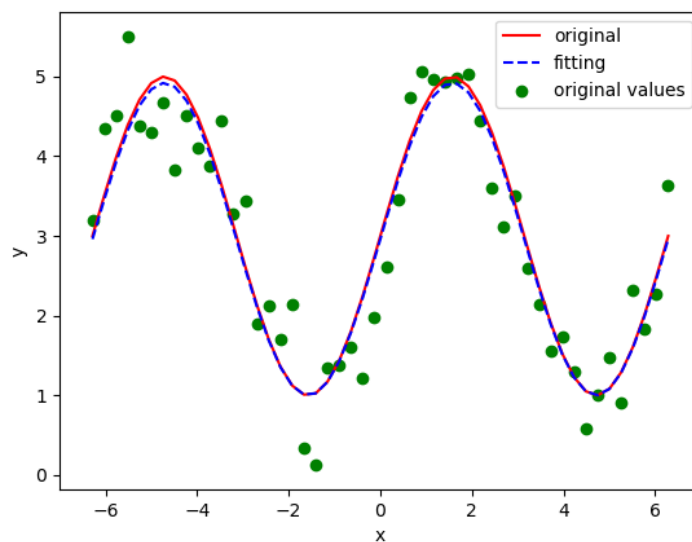


图 10-6 $a*\sin(x)+b$ 的拟合图

5) 插值补齐

下面是拉格朗日的使用举例：

```
>>> x = [-2, 0, 1, 2] #生成已知点的 x 坐标
>>> y = [17, 1, 2, 17] #生成已知点的 y 坐标
>>> def Larange(x, y, a):
    t = 0.0
    for i in range(len(y)):
        c = y[i]
        for j in range(len(y)):
```

```

        if j != i:
            c *= (a-x[j])/(x[i]-x[j])
        t += c
    return t
>>> y2 = Larange(x, y, 0.6)      #求插值函数在 0.6 处的函数值
>>> print(y2)
0.256

```

Python 的 `scipy` 库提供了拉格朗日插值函数，因此可通过直接调用拉格朗日插值函数来实现插值计算。

```

from scipy.interpolate import lagrange
x = [-2, 0, 1, 2]    #生成已知点的 x 坐标
y = [17, 1, 2, 17]   #生成已知点的 y 坐标
a=lagrange(x, y)     #四个点返回 3 阶拉格朗日插值多项式
print('插值函数的阶: '+str(a.order))
print('插值函数的系数:
'+str(a[3])+':'+str(a[2])+':'+str(a[1])+':'+ str(a[0]))
print(a)
print(a(0.6)) #插值函数在 0.6 处的值

```

执行上述代码得到的输出如下：

插值函数的阶：3

插值函数的系数：1.0:4.0:-4.0:1.0

$$1x^3 + 4x^2 - 4x + 1$$

0.25599999999999999

10.2 数据集成

10.3 数据规范化

10.3.1 最小-最大规范化

对 Iris 数据集的数据进行最小-最大规范化处理的代码如下。

```

>>> from sklearn.preprocessing import MinMaxScaler
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()

```

#获取 IRIS(鸢尾花)数据集前 6 行数据，每行数据为花萼长度、宽度，花瓣长度、宽度


```

>>> data=iris.data[0:6]
>>> data
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2],
       [5.4, 3.9, 1.7, 0.4]])
#返回值为缩放到[0,1]区间的数据
>>> MinMaxScaler().fit_transform(data)
array([[0.625      , 0.55555556, 0.25      , 0.        ],
       [0.375      , 0.        , 0.25      , 0.        ],
       [0.125      , 0.22222222, 0.        , 0.        ],
       [0.        , 0.11111111, 0.5       , 0.        ],
       [0.5       , 0.66666667, 0.25      , 0.        ],
       [1.        , 1.        , 1.        , 1.        ]])

```

10.3.2 z 分数规范化

使用 `preprocessing` 库的 `StandardScaler` 类对数据进行标准化的代码如下：

```

>>> from sklearn.preprocessing import StandardScaler
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
#获取 IRIS(鸢尾花)数据集前 6 行数据，每行数据为花萼长度、宽度，花瓣
长度、宽度
>>> data=iris.data[0:6]
#标准化，返回值为标准化后的数据
>>> StandardScaler().fit_transform(data)
array([[ 0.57035183,  0.37257241, -0.39735971, -0.4472136 ],
       [-0.19011728, -1.22416648, -0.39735971, -0.4472136 ],
       [-0.95058638, -0.58547092, -1.19207912, -0.4472136 ],
       [-1.33082093, -0.9048187 ,  0.39735971, -0.4472136 ],
       [ 0.19011728,  0.69192018, -0.39735971, -0.4472136 ],
       [ 1.71105548,  1.64996352,  1.98679854,  2.23606798]])

```

10.3.3 小数定标规范化

10.4 数据离散化

10.4.1 无监督离散化

1. 等宽离散化

```
>>> import pandas as pd
>>> x=[1, 2, 5, 10, 12, 14, 17, 19, 3, 21, 18, 28, 7]
>>> x=pd.Series(x)
>>> s=pd.cut(x, bins=[0, 10, 20, 30])#此处是等宽离散化方法，bin 表示区间的间距
```

```
>>> s          #获取每个数据的类标号
0      (0, 10]
1      (0, 10]
2      (0, 10]
3      (0, 10]
4     (10, 20]
5     (10, 20]
6     (10, 20]
7     (10, 20]
8      (0, 10]
9     (20, 30]
10     (10, 20]
11     (20, 30]
12     (0, 10]
dtype: category
Categories (3, interval[int64]): [(0, 10] < (10, 20] < (20, 30]]
```

10.4.2 监督离散化

10.5 数据归约

10.5.1 过滤法

1. 方差选择法

```
>>> from sklearn.feature_selection import VarianceThreshold
>>> from sklearn.datasets import load_iris
```

```
>>> iris = load_iris()
#方差选择, 返回值为特征选择后的数据, 参数 threshold 为方差的阈值
>>>
VarianceThreshold(threshold=0.2).fit_transform(iris.data)[0:5]
array([[5.1, 1.4, 0.2],
       [4.9, 1.4, 0.2],
       [4.7, 1.3, 0.2],
       [4.6, 1.5, 0.2],
       [5. , 1.4, 0.2]])
```

2. 相关系数法

```
>>> from sklearn.feature_selection import SelectKBest
>>> import numpy as np
>>> from scipy.stats import pearsonr
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>> iris.data[0:5]    #显示前 5 行花的特征数据
array([[5.1, 3.5, 1.4, 0.2],
       [4.9, 3. , 1.4, 0.2],
       [4.7, 3.2, 1.3, 0.2],
       [4.6, 3.1, 1.5, 0.2],
       [5. , 3.6, 1.4, 0.2]])
>>> iris.target[0:5] #显示前 5 行花的类别数据
array([0, 0, 0, 0, 0])
```

"选择 k 个最好的特征, 返回选择特征后的数据; 第一个参数为计算评估特征是否好的函数, 该函数输入特征矩阵和目标向量, 输出二元组 (评分, P 值) 的数组, 数组第 i 项为第 i 个特征的评分和 P 值, 在此定义为计算相关系数; 参数 k 为选择的特征个数"

```
>>> m=SelectKBest(lambda X,Y:np.array(list(map(lambda x:pearsonr(x,
Y), X.T))).T[0], k=2).fit_transform(iris.data, iris.target)
>>> m[0:5]          #获取选择的特征
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
```

```
[1.5, 0.2],  
[1.4, 0.2]])
```

3. 卡方检验法

```
>>> from sklearn.datasets import load_iris  
>>> from sklearn.feature_selection import SelectKBest  
>>> from sklearn.feature_selection import chi2  
>>> iris = load_iris()  
#选择 K 个最好的特征，返回选择特征后的数据，这里只显示前 5 行数据  
>>> SelectKBest(chi2, k=2).fit_transform(iris.data,  
iris.target)[0:5]  
array([[1.4, 0.2],  
       [1.4, 0.2],  
       [1.3, 0.2],  
       [1.5, 0.2],  
       [1.4, 0.2]])
```

从返回结果可以看出选择出的两个特征是花瓣长度、花瓣宽度。

4. 最大信息系数法

```
>>> from sklearn.feature_selection import SelectKBest  
>>> from minepy import MINE  
>>> from sklearn.datasets import load_iris  
>>> iris = load_iris()  
''' 由于 MINE 的设计不是函数式的，定义 mic 方法将其为函数式的，返回一个二元组，二元组的第 2 项设置成固定的 P 值 0.5 '''  
>>> def mic(x, y):  
    m = MINE()  
    m.compute_score(x, y)  
    return (m.mic(), 0.5)  
#选择 k 个最好的特征，返回特征选择后的数据，这里只显示前 5 行数据  
>>> SelectKBest(lambda X, Y: np.array(list(map(lambda x:mic(x, Y),  
X.T))).T[0], k=2).fit_transform(iris.data, iris.target)[0:5]  
array([[1.4, 0.2],  
       [1.4, 0.2],  
       [1.3, 0.2],  
       [1.5, 0.2],  
       [1.4, 0.2]])
```

```

[1.3, 0.2],
[1.5, 0.2],
[1.4, 0.2]])

```

从返回结果可以看出选择出的两个特征是花瓣长度、花瓣宽度。

10.5.2 包装法

```

>>> from sklearn.feature_selection import RFE
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()

#递归特征消除法，返回特征选择后的数据，参数 estimator 用来指定学习
模型

#参数 n_features_to_select 为选择的特征个数
>>> RFE(estimator=LogisticRegression(),
n_features_to_select=2).fit_transform(iris.data, iris.target)[0:5]
array([[3.5, 0.2],
       [3. , 0.2],
       [3.2, 0.2],
       [3.1, 0.2],
       [3.6, 0.2]])

```

10.5.3 嵌入法

1. 基于惩罚项的特征选择法

```

>>> from sklearn.feature_selection import SelectFromModel
>>> from sklearn.linear_model import LogisticRegression
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()

#带 L1 惩罚项的逻辑回归作为基模型的特征选择
>>> SelectFromModel(LogisticRegression(penalty="l1",
C=0.1)).fit_transform(iris.data, iris.target)[0:5]
array([[5.1, 3.5, 1.4],
       [4.9, 3. , 1.4],
       [4.7, 3.2, 1.3],
       [4.6, 3.1, 1.5],

```

```
[5. , 3.6, 1.4]])
```

实际上，L1 惩罚项降维的原理在于保留多个对目标值具有同等相关性的特征中的一个，没选到的特征不代表不重要。

2. 基于树模型的特征选择法

```
>>> from sklearn.feature_selection import SelectFromModel
>>> from sklearn.ensemble import GradientBoostingClassifier
>>> from sklearn.datasets import load_iris
>>> iris = load_iris()
>>>
SelectFromModel(GradientBoostingClassifier()).fit_transform(iris.data,
iris.target)[0:5]
array([[1.4, 0.2],
       [1.4, 0.2],
       [1.3, 0.2],
       [1.5, 0.2],
       [1.4, 0.2]])
```

10.6 数据降维

10.6.1 主成分分析

使用 `decomposition` 库的 `PCA` 类选择特征降维的代码如下：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.decomposition import PCA
>>> iris = load_iris()
#主成分分析法，返回降维后的数据，参数 n_components 为主成分数目
>>> PCA(n_components=2).fit_transform(iris.data)[0:5]
array([[ -2.68420713,  0.32660731],
       [-2.71539062, -0.16955685],
       [-2.88981954, -0.13734561],
       [-2.7464372 , -0.31112432],
       [-2.72859298,  0.33392456]])
```

10.6.2 线性判别分析法

```
>>> import matplotlib.pyplot as plt
>>> from sklearn.datasets import load_iris
```

```
>>> from sklearn.discriminant_analysis import
LinearDiscriminantAnalysis as LDA
>>> iris = load_iris()
#利用 LDA 将原始数据降至 2 维, 因为 LDA 要求降维后的维数<=分类数-1
>>> X_lda = LDA(n_components=2).fit_transform(iris.data,
iris.target)
>>> X_lda[0:5] #显示降维后的前 5 行数据
array([[ 8.0849532 ,  0.32845422],
       [ 7.1471629 , -0.75547326],
       [ 7.51137789, -0.23807832],
       [ 6.83767561, -0.64288476],
       [ 8.15781367,  0.54063935]])
#将降至 2 维的数据进行绘图
>>> fig = plt.figure()
>>> plt.scatter(X_lda[:, 0], X_lda[:, 1], marker='o',c=iris.target)
<matplotlib.collections.PathCollection object at
0x000000001B6F06D8>
>>> plt.show() #显示对降至 2 维的数据进行绘图的绘图结果如图 10-7
所示
```

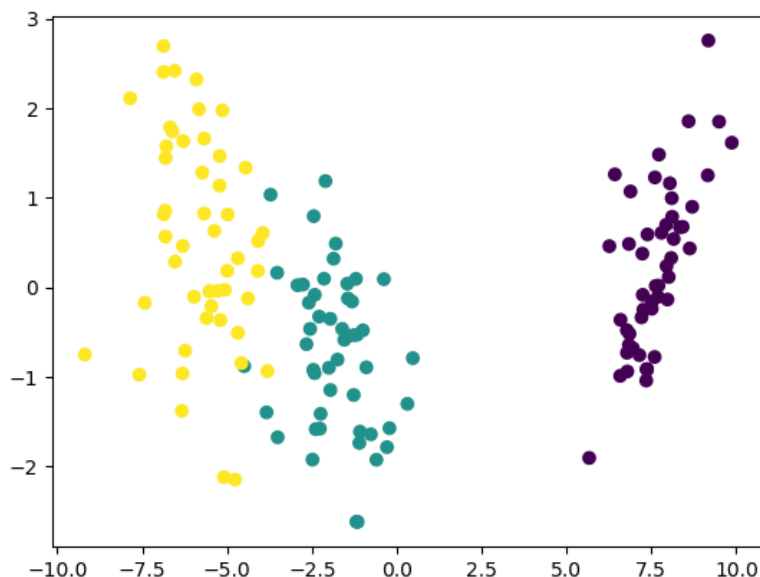


图 10-7 对降至 2 维的数据进行绘图的绘图结果

10.7 数据预处理举例

【例 10-2】数据规范化。

```

>>> import pandas as pd
>>> import numpy as np
>>> data = pd.read_excel(r'D:\Python\chengji.xlsx', header=None) #
读取数据
>>> data

```

	0	1	2	3	4	5
0	Student ID	name	C	database	oracle	Java
1	541513440106	ding	77	80	95	91
2	541513440242	yan	83	90	93	90
3	541513440107	feng	85	90	92	91
4	541513440230	wang	86	80	86	91
5	541513440153	zhang	76	90	90	92
6	541513440235	lu	69	90	83	92
7	541513440224	men	79	90	86	90
8	541513440236	fei	73	80	85	89
9	541513440210	han	80	80	93	88

```

>>> data=data.iloc[:,2:] #获取第2列之后的所有列
>>> data

```

	2	3	4	5
0	C	database	oracle	Java
1	77	80	95	91
2	83	90	93	90
...
8	73	80	85	89
9	80	80	93	88

```

>>> data=data[1:10] #获取第1行至第9行数据
>>> data

```

	2	3	4	5
1	77	80	95	91
2	83	90	93	90
...
8	73	80	85	89
9	80	80	93	88


```
>>> print((data - data.min())/(data.max() - data.min())) #最小-最大规范化
```

	2	3	4	5
1	0.470588	0	1	0.75
2	0.823529	1	0.833333	0.5
...
8	0.235294	0	0.166667	0.25
9	0.647059	0	0.833333	0

```
>>> print((data - data.mean())/data.std()) #标准差标准化
```

	2	3	4	5
1	-0.298142	-1.05409	1.34533	0.416667
2	0.77517	0.843274	0.879637	-0.333333
...
8	-1.01368	-1.05409	-0.983124	-1.08333
9	0.238514	-1.05409	0.879637	-1.83333

```
>>> print(data/10**np.ceil(np.log10(data.abs().max()))) #小数定标规范化
```

	2	3	4	5
1	0.77	0.8	0.95	0.91
2	0.83	0.9	0.93	0.9
...
8	0.73	0.8	0.85	0.89
9	0.8	0.8	0.93	0.88

【例 10-3】数据预处理综合实战

```
>>> import pandas as pd
```

```
>>> import numpy as np
```

```
>>> data = pd.read_csv(r'D:\Python\chengji.csv') #读取数据
```

```
>>> data
```

	name	sex	C	database	oracle	Java
0	ding	female	77	80	95	91.0
1	yan	female	83	90	93	90.0
2	feng	female	85	90	92	91.0
3	wang	male	86	80	86	91.0
4	zhang	male	76	90	90	92.0

```

5      lu  female  69          90          83  92.0
6  meng    male  79          90          86  NaN
7   fei  female  73          80          85  89.0
8   han    male  80          80          93  88.0

>>> data_statistics=data.describe().T  #产生多个列的汇总统计，T 表示转置

>>> data_statistics
           count          mean          std   min    25%    50%    75%
max
C           9.0  78.666667  5.590170  69.0  76.00  79.0  83.00
86.0
database    9.0  85.555556  5.270463  80.0  80.00  90.0  90.00
90.0
oracle      9.0  89.222222  4.294700  83.0  86.00  90.0  93.00
95.0
Java        8.0  90.500000  1.414214  88.0  89.75  91.0  91.25
92.0

>>> data_statistics['null']=len(data)-data_statistics['count']  #
统计空值记录

>>> data_statistics
           count          mean          std   min    25%    50%    75%
max  null
C           9.0  78.666667  5.590170  69.0  76.00  79.0  83.00
86.0    0.0
database    9.0  85.555556  5.270463  80.0  80.00  90.0  90.00
90.0    0.0
oracle      9.0  89.222222  4.294700  83.0  86.00  90.0  93.00
95.0    0.0
Java        8.0  90.500000  1.414214  88.0  89.75  91.0  91.25
92.0    1.0

>>> data_max_min=data_statistics[['max','min']]  #获取'max','min'
两列的内容

>>> data_max_min
           max    min

```

```
C          86.0  69.0
```

```
database   90.0  80.0
```

```
oracle     95.0  83.0
```

```
Java       92.0  88.0
```

#选取 oracle 成绩大于 85 且 Java 成绩大于 90 的学生

```
>>> data[(data['oracle']>85)&(data['Java']>90)]
```

	name	sex	C	database	oracle	Java
0	ding	female	77	80	95	91.0
2	feng	female	85	90	92	91.0
3	wang	male	86	80	86	91.0
4	zhang	male	76	90	90	92.0

```
>>> data.sort_values(['C','Java'],ascending=True) #按'C'、'Java'
```

进行升序排列

	name	sex	C	database	oracle	Java
5	lu	female	69	90	83	92.0
7	fei	female	73	80	85	89.0
4	zhang	male	76	90	90	92.0
0	ding	female	77	80	95	91.0
6	meng	male	79	90	86	NaN
8	han	male	80	80	93	88.0
1	yan	female	83	90	93	90.0
2	feng	female	85	90	92	91.0
3	wang	male	86	80	86	91.0

```
>>> data.groupby('sex').size() #按'sex'列分组
```

```
sex
```

```
female    5
```

```
male      4
```

```
dtype: int64
```

```
>>> data.groupby('sex').count() #按'sex'列分组
```

	name	C	database	oracle	Java
sex					
female	5	5	5	5	5
male	4	4	4	4	3

```
>>> data.groupby('sex').agg({'C': np.sum}) #按'sex'列分组并对'C'
列求和
```

```
      C
sex
female 387
male    321
```

```
>>> data.groupby('sex').agg({'C': np.max})
      C
```

```
sex
female 85
male    86
```

```
>>> sex_mapping = { 'female': 1, 'male': 2}
```

```
>>> data['sex'] = data['sex'].map(sex_mapping) #应用 map 函数
```

```
>>> data
```

	name	sex	C	database	oracle	Java
0	ding	1	77	80	95	91.0
1	yan	1	83	90	93	90.0
...
6	meng	2	79	90	86	NaN
7	fei	1	73	80	85	89.0
8	han	2	80	80	93	88.0

```
#应用 apply 函数
```

```
>>> data['C']=data['C'].apply(lambda x: x+10 if x >= 85 else x)
```

```
>>> data
```

	name	sex	C	database	oracle	Java
0	ding	1	77	80	95	91.0
1	yan	1	83	90	93	90.0
...
6	meng	2	79	90	86	NaN
7	fei	1	73	80	85	89.0
8	han	2	80	80	93	88.0

```
>>> df=data.dropna() #删除含有缺失值的行
```

```
>>> df
```

	name	sex	C	database	oracle	Java
--	------	-----	---	----------	--------	------

```

0   ding    1  77      80      95  91.0
...   ...    ...  ...      ...    ...  ...

5     lu    1  69      90      83  92.0
7    fei    1  73      80      85  89.0
8    han    2  80      80      93  88.0
>>> df1=data.fillna(0)      #用 0 填补所有缺失值
>>> df1
      name  sex   C  database  oracle  Java
0   ding    1  77      80      95  91.0
...   ...    ...  ...      ...    ...  ...
6   meng    2  79      90      86   0.0
7    fei    1  73      80      85  89.0
8    han    2  80      80      93  88.0
>>> df2=data.fillna(method='ffill')      #使用前一个观察值填充缺失
值
>>> df2
      name  sex   C  database  oracle  Java
0   ding    1  77      80      95  91.0
...   ...    ...  ...      ...    ...  ...
6   meng    2  79      90      86  92.0
7    fei    1  73      80      85  89.0
8    han    2  80      80      93  88.0
#使用均值填充指定列的缺失值
>>> df3=data.fillna({'Java':int(data['Java'].mean())})
>>> df3
      name  sex   C  database  oracle  Java
0   ding    1  77      80      95  91.0
...   ...    ...  ...      ...    ...  ...
6   meng    2  79      90      86  90.0
7    fei    1  73      80      85  89.0
8    han    2  80      80      93  88.0
#数据分箱（离散化）
>>> bins = [60, 70, 80, 90,100]      #分箱的边界

```

```

>>> cats = pd.cut(list(data['C']), bins) #使用 cut 函数进行数据分箱
箱

>>> cats #显示分箱结果
[(70, 80], (80, 90], (90, 100], (90, 100], (70, 80], (60, 70], (70,
80], (70, 80], (70, 80]]
Categories (4, interval[int64]): [(60, 70] < (70, 80] < (80, 90] <
(90, 100]]

>>> cats.codes #获取分箱编码
array([1, 2, 3, 3, 1, 0, 1, 1, 1], dtype=int8)

>>> cats.categories #返回分箱便捷索引
IntervalIndex([(60, 70], (70, 80], (80, 90], (90, 100]]
                closed='right',
                dtype='interval[int64]')

>>> pd.value_counts(cats) #统计箱中元素的个数
(70, 80]      5
(90, 100]      2
(80, 90]       1
(60, 70]       1
dtype: int64
#进行带标签的分箱
>>> group_names = ['pass', 'medium', 'good', 'excellent']
>>> cats1 = pd.cut(list(data['C']), bins, labels = group_names)
>>> cats1 #查看带标签的分箱结果
[medium, good, excellent, excellent, medium, pass, medium, medium,
medium]
Categories (4, object): [pass < medium < good < excellent]
>>> cats1.get_values()
array(['medium', 'good', 'excellent', 'excellent', 'medium',
'pass',
      'medium', 'medium', 'medium'], dtype=object)

```


实验九 数据分析方法实验

一、实验目的

掌握相似度和相异度的度量、分类分析方法、回归分析方法和聚类分析方法。

二、实验过程

1.DBSCAN 算法应用举例

```
from sklearn.cluster import DBSCAN
from sklearn.datasets.samples_generator import make_blobs
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
centers = [[1, 1], [-1, -1], [1, -1]]
# 生成样本数据
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4)
db = DBSCAN(eps=0.3, min_samples=10, metric='euclidean')
y_db = db.fit_predict(X)
plt.scatter(X[y_db==0,0],X[y_db==0,1],c='', marker='o', edgecolors='k', s=40,
label='cluster 1')
plt.scatter(X[y_db==1,0],X[y_db==1,1],c='',marker='s',edgecolors='k',s=40,label='cluster 2')
plt.scatter(X[y_db==2,0],X[y_db==2,1],c='',marker='*',edgecolors='k',s=40,label='cluster 3')
plt.legend()
plt.show()    #显示 DBSCAN 聚类的结果
```

2.LinearRegression 模型拟合 boston 房价数据集

1. 数据集的数据结构分析

```
>>> from sklearn.datasets import load_boston
>>> import pandas as pd
>>> import numpy as np
>>> import matplotlib.pyplot as plt    #python 中的绘图模块
>>> from sklearn.linear_model import LinearRegression    #导入线性回归模型
>>> boston=load_boston()    #加载波士顿房价数据集
>>> x=boston.data    #加载波士顿房价属性数据集
```



```

>>> y=boston.target          #加载波士顿房价房价数据集
>>> boston.keys()
>>> x.shape
>>> boston_df=pd.DataFrame(boston['data'],columns=boston.feature_names)
>>> boston_df['Target']=pd.DataFrame(boston['target'],columns=['Target'])
>>> boston_df.head(3)  #显示完整数据集的前 3 行数据
2. 分析数据并可视化
>>> boston_df.corr().sort_values(by=['Target'],ascending=False)
>>> import matplotlib
>>> matplotlib.rcParams['font.family'] = 'FangSong'  #设置中文字体格式为仿宋
>>> plt.scatter(boston_df['RM'],y)
<matplotlib.collections.PathCollection object at 0x000000001924C550>
>>> plt.xlabel('房间数(RM)',fontsize=15)
Text(0.5,0,'房间数(RM)')
>>> plt.ylabel('房屋价格(MEDV)',fontsize=15)
Text(0,0.5,'房屋价格(MEDV)')
>>> plt.title('房间数(RM)与房屋价格(MEDV)的关系',fontsize=15)
Text(0.5,1,'房间数(RM)与房屋价格(MEDV)的关系')
>>> plt.show()  #显示绘制的房间数与房屋价格的散点图

```

3. 一元线性回归

（1）去掉一些脏数据，比如去掉房价大于等于 50 的数据和房价小于等于 10 的数据。

```

>>> X=boston.data
>>> y=boston.target
>>> X=X[y<50]
>>> y=y[y<50]
>>> X=X[y>10]
>>> y=y[y>10]
>>> X.shape
(466, 13)
>>> y.shape
(466,)

```

（2）构建线性回归模型

```

>>> from sklearn.model_selection import train_test_split

```

#切分数据集，取数据集的 75%作为训练数据，25%作为测试数据

```
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
```

```
>>> LR =LinearRegression()
```

```
>>> LR.fit(X_train,y_train)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

(3) 算法评估

```
>>> pre = LR.predict(X_test)
```

```
>>> print("预测结果", pre[3:8])          #选取 5 个结果进行显示
```

```
预测结果 [27.48834701 21.58192891 20.36438243 22.980885   24.35103277]
```

```
>>> print(u"真实结果", y_test[3:8])      #选取 5 个结果进行显示
```

```
真实结果 [22.   22.   24.3 22.2 21.9]
```

```
>>> LR.score(X_test,y_test)              #模型评分
```

```
0.7155555361911698
```

这个模型的准确率只有 71.5%。

3. 簇间最大距离的凝聚层次聚类

```
import pandas as pd
```

```
import numpy as np
```

```
from scipy.spatial.distance import pdist, squareform
```

```
from scipy.cluster.hierarchy import linkage
```

```
from scipy.cluster.hierarchy import dendrogram
```

```
import matplotlib.pyplot as plt
```

```
np.random.seed(150)
```

```
features=['f1','f2','f3']          #设置特征的名称
```

```
labels = ["s0","s1","s2","s3","s4"] #设置数据样本编号
```

```
X = np.random.random_sample([5,3])*10 #生成一个(5,3)的数组
```

```
#通过 pandas 将数组转换成一个 DataFrame 类型
```

```
df=pd.DataFrame(X,columns=features,index=labels)
```

```
print(df)    #查看生成的数据
```

```
dist_matrix = pd.DataFrame(squareform(pdist(df,metric='euclidean')),
```

```
columns=labels, index=labels)
```

```
print(dist_matrix) #查看距离矩阵
```

#linkage()以簇间最大距离作为距离判断标准，得到一个关系矩阵，实现层次聚类

#linkage()返回长度为 n-1 的数组，其包含每一步合并簇的信息，n 为数据集的样

本数

```
row_clusters = linkage(pdist(df,metric='euclidean'),method="complete")
print(row_clusters) #输出合并簇的过程信息
#将关系矩阵转换成一个 DataFrame 对象
clusters = pd.DataFrame(row_clusters,columns=["label 1","label
2","distance","sample size"],index=["cluster %d"%(i+1) for i in
range(row_clusters.shape[0])])
print(clusters)
dendrogram(row_clusters,labels=labels)
plt.tight_layout()
plt.ylabel('Euclidean distance')
plt.show() #显示层次聚类的树状图
```

4. 对 iris 数据进行逻辑回归分析

```
from sklearn.datasets import load_iris
from sklearn.linear_model import LogisticRegression as LR
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
from sklearn.cross_validation import train_test_split #这里是引用了交叉验证
matplotlib.rcParams['font.family'] = 'Kaiti' #Kaiti 是中文楷体
#加载数据
iris = load_iris()
data = iris.data
target = iris.target
X = data[0:100,[0,2]] #获取前 100 条数据的前两列
y = target[0:100] #获取类别属性数据的前 100 条数据
label = np.array(y)
index_0 = np.where(label==0) #获取 label 中数据值为 0 的索引
#按选取的两个特征绘制散点图
plt.scatter(X[index_0,0],X[index_0,1],marker='x',color = 'k',label = '0')
index_1 = np.where(label==1) #获取 label 中数据值为 1 的索引
plt.scatter(X[index_1,0],X[index_1,1],marker='o',color = 'k',label = '1')
plt.xlabel('花萼长度',fontsize=15)
```

```

plt.ylabel('花萼宽度',fontsize=15)
plt.legend(loc = 'lower right')
plt.show()
#切分数据集，取数据集的 75%作为训练数据，25%作为测试数据
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
lr=LR()          #建立逻辑回归模型
lr.fit(X_train,y_train)  #训练模型
print('模型在(X_test, y_test)上的预测准确率为: ', lr.score(X_test, y_test))

```

5. 广告媒体与销售额之间的关系

```

import pandas as pd
import matplotlib
matplotlib.rcParams['font.family'] = 'Kaiti' #Kaiti 是中文楷体
from sklearn import linear_model
from sklearn.cross_validation import train_test_split #这里是引用了交叉验证
data = pd.read_csv('Advertising.csv')
feature_cols = ['TV', 'radio', 'newspaper'] #指定特征属性
X = data[feature_cols] #得到数据集的三个属性'TV', 'radio', 'newspaper'列
y = data['sales']      #得到数据集的目标列，即 sales 列
#切分数据集，取数据集的 75%作为训练数据，25%作为测试数据
X_train,X_test, y_train, y_test = train_test_split(X, y, random_state=1)
clf = linear_model.LinearRegression()    #建立线性回归模型
clf.fit(X_train,y_train)                 #训练模型
print('回归方程的非常数项系数 coef_值为: ',clf.coef_)
print('回归方程的常数项 intercept_值为: ',clf.intercept_)
print(list(zip(feature_cols, clf.coef_))) #输出每个特征相应的回归系数
#模型评价
y_pred = clf.predict(X_test)
print('预测效果评分: ',clf.score(X_test, y_test))
#以图形的方式表示所得到的模型质量
import matplotlib.pyplot as plt
plt.figure()
plt.plot(range(len(y_pred)),y_pred,'k',label="预测值")
plt.plot(range(len(y_pred)),y_test,'k--',label="测试值")

```

```
plt.legend(loc="upper right") #显示图中的标签
plt.xlabel("测试数据序号",fontsize=15)
plt.ylabel('销售额',fontsize=15)
plt.show()          #绘制的预测值与测试值的线性图
```

6. 使用 k-means 对鸢尾花数据集聚类

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import matplotlib

from sklearn.cross_validation import train_test_split #这里是引用了交叉验证
matplotlib.rcParams['font.family'] = 'Kaiti'        #Kaiti 是中文楷体
#加载数据
iris = load_iris()
data = iris.data
target = iris.target
X = data[:,[0,2]]    #获取第 1 列和第 3 列数据
y = iris.target      #获取类别属性数据
label = np.array(y)
index_0 = np.where(label==0) #获取类别属性数据中类别为 0 的数据索引
#按选取的两个特征绘制散点
plt.scatter(X[index_0,0],X[index_0,1],marker='o',color = '', edgecolors='k', label = '0')
index_1 = np.where(label==1) #获取类别属性数据中类别为 1 的数据索引
plt.scatter(X[index_1,0],X[index_1,1], marker='*', color = 'k', label = '1')
index_2 = np.where(label==2) #获取类别属性数据中类别为 2 的数据索引
plt.scatter(X[index_2,0],X[index_2,1], marker='o', color = 'k', label = '2')
plt.xlabel('花萼长度', fontsize=15)
plt.ylabel('花萼宽度',fontsize=15)
plt.legend(loc = 'lower right')
plt.show()    #显示按鸢尾花数据集的两个特征绘制的散点图
#切分数据集，取数据集的 75%作为训练数据，25%作为测试数据
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
kms = KMeans(n_clusters=3) #构造聚类模型,设定生成的聚类数为 3
```

```

kms.fit(X_train)          #在数据集 X_train 上进行 k-means 聚类
label_pred = kms.labels_  #获取每个样本点对应的类别
#绘制 k-means 结果
x0 = X_train[label_pred == 0]
x1 = X_train[label_pred == 1]
x2 = X_train[label_pred == 2]
plt.scatter(x0[:, 0], x0[:, 1], c = "", marker='o', edgecolors='k', label='label0')
plt.scatter(x1[:, 0], x1[:, 1], c = "", marker='*', edgecolors='k', label='label1')
plt.scatter(x2[:, 0], x2[:, 1], c = "k", marker='o', label='label2')
plt.xlabel('花萼长度', fontsize=15)
plt.ylabel('花萼宽度', fontsize=15)
plt.legend(loc=2)
plt.show()               #显示鸢尾花数据集 k-means 聚类的结果

```

7. 线性支持向量机

```

from sklearn import svm
import numpy as np
import matplotlib.pyplot as plt  #python 中的绘图模块
#平面上的 8 个点
X = [[0.39,0.17],[0.49,0.71],[0.92,0.61],[0.74,0.89],[0.18,0.06], [0.41,0.26],[0.94,0.81],
[0.21,0.01]]
Y = [1,-1,-1,-1,1,1,-1,1]      #标记数据点属于的类
clf = svm.SVC(kernel='linear') #建立模型， linear 为小写， 线性核函数
clf.fit(X,Y)                    #训练模型
w = clf.coef_[0]                #取得 w 值， w 是二维的
a = -w[0]/w[1]                  #计算直线斜率
x = np.linspace(0,1,50)         #随机产生连续 x 值
y = a*x-(clf.intercept_[0])/w[1] #根据随机 x 得到 y 值
#计算与直线相平行的两条直线
b = clf.support_vectors_[0]      #获取 1 个支持向量
y_down = a*x+(b[1]-a*b[0])
c = clf.support_vectors_[-1]     #获取 1 个支持向量
y_up = a*x+(c[1]-a*c[0])
print('模型参数 w:',w)

```

```

print('边缘直线斜率:',a)
print('打印出支持向量:',clf.support_vectors_)
#画出三条直线
plt.plot(x,y,'k-')
plt.plot(x,y_down,'k--')
plt.plot(x,y_up,'k--')
#绘制散点图
plt.scatter([s[0] for s in X],[s[1] for s in X],c=Y, cmap=plt.cm.Paired)
plt.show()
8. 一元线性回归
import matplotlib.pyplot as plt
import matplotlib
import numpy as np
matplotlib.rcParams['font.family'] = 'FangSong' #指定字体的中文格式
#定义一个画图函数
def runplt():
    plt.figure()
    plt.title('10 个厂家的投入和产出',fontsize=15)
    plt.xlabel('投入',fontsize=15)
    plt.ylabel('产出',fontsize=15)
    plt.axis([0,50,0,80])
    plt.grid(True)
    return plt
#投入、产出训练数据
X = [[20],[40],[20],[30],[10],[10],[20],[20],[20],[30]]
y = [[30],[60],[40],[60],[30],[40],[40],[50],[30],[70]]
from sklearn.linear_model import LinearRegression
model = LinearRegression() #建立线性回归模型
model.fit(X,y) #用训练数据进行模型训练
runplt()
X2 = [[0],[20],[25],[30],[35],[50]]
#利用通过 fit()训练的模型对输入值的产出值进行预测
y2 = model.predict(X2) #预测数据
plt.plot(X,y,'k.') #根据观察到的投入、产出值绘制点

```

```
plt.plot(X2,y2,'k-')          #根据 X2、 y2 绘制拟合的回归直线
plt.show()                    #显示绘制的一元线性回归图如图 11-12 所示
#输出  $\beta_0$  的估计值
print('求得的一元线性回归方程的 b0 值为: %.2f'%model.intercept_)
print('求得的一元线性回归方程的 b1 值为: %.2f'%model.coef_) #输出  $\beta_1$  的估计
值
print('预测投入 25 的产出值: %.2f'%model.predict([[25]]))#输出投入 25 的预测值
```