《数据采集与预处理》

教材官网： http://dblab.xmu.edu.cn/post/data-collection/

温馨提示：编辑幻灯片母版，可以修改每页PPT的厦大校徽和底部文字

第8章 使用pandas进行数据清洗

（PPT版本号：2022年1月版本）

林子雨 副教授

厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn

主页：**http://dblab.xmu.edu.cn/linziyu**

提纲

8.1 NumPy的基本使用方法
8.2 pandas数据结构
8.3 基本功能
8.4 汇总和描述统计
8.5 处理缺失数据
8.6 综合实例

本PPT是以下教材的配套讲义
林子雨编著《数据采集与预处理》
人民邮电出版社

教材官网：
http://dblab.xmu.edu.cn/post/data-collection

8.1 NumPy的基本使用方法

NumPy是Python语言的一个扩充程序库，支持高级的数组与矩阵运算，此外也针对数组运算提供了大量的数学函数库，包括线性代数运算、傅立叶变换和随机数生成等。如果没有安装NumPy，可以在Windows系统的cmd窗口中执行如下命令安装：

```
> pip install numpy
```

8.1.1 数组创建
8.1.2 数组索引和切片
8.1.3 数组运算

8.1.1 数组创建

下面是数组创建的一些具体实例：

```
>>> import numpy as np
>>> a = [1,2,3,4,5]     # 创建简单的列表
>>> b = np.array(a)    # 将列表转换为数组
>>> b
array([1, 2, 3, 4, 5])
>>> b.size  # 数组的元素个数
5
>>> b.shape  # 数组的形状
(5,)
>>> b.ndim  # 数组的维度
1
```

8.1.1 数组创建

```
>>> b.dtype  # 数据的元素类型
dtype('int32')
>>> print(b[0],b[1],b[2])  #访问数组元素
1 2 3
>>> b[4] = 6  # 修改数组元素
>>> b
array([1, 2, 3, 4, 6])
>>> c = np.array([[1,2,3],[4,5,6]])  # 创建二维数组
>>> c.shape     # 数组的形状
(2, 3)
>>> print(c[0,0],c[0,1],c[0,2],c[1,0],c[1,1],c[1,2])
1 2 3 4 5 6
```

8.1.1 数组创建

Python做数据处理的时候经常要初始化高维矩阵，常用的函数包括zeros()、ones()、empty()、eye()、full()、random.random()、random.randint()、random.rand()、random.randn()等，具体如下：

（1）zeros()：创建一个矩阵，内部元素均为0，第一个参数提供维度，第二个参数提供类型。

```
>>> a = np.zeros([2,3],int)
>>> a
array([[0, 0, 0],
       [0, 0, 0]])
```

8.1.1 数组创建

（2）ones()：创建一个矩阵，内部元素均为1，第一个参数提供维度，第二个参数提供类型。

```
>>> a = np.ones([2,3],int)
>>> a
array([[1, 1, 1],
       [1, 1, 1]])
```

（3）empty()：创建一个矩阵，内部是无意义的数值，第一个参数提供维度，第二个参数提供类型。

```
>>> a = np.empty([2,3],int)
>>> a
array([[0, 0, 0],
       [0, 0, 0]])
```

8.1.1 数组创建

（4）eye()：创建一个对角矩阵，第一个参数提供矩阵规模，对于第二个参数而言，如果为0则对角线全为“1”，大于0则右上方第K条对角线全为“1”，小于0则左下方第K条对角线全为“1”，第三个参数提供类型。

```
>>> a = np.eye(3,k=1,dtype=int)
>>> a
array([[0, 1, 0],
       [0, 0, 1],
       [0, 0, 0]])
>>> a = np.eye(4,k=-2,dtype=int)
>>> a
array([[0, 0, 0, 0],
       [0, 0, 0, 0],
       [1, 0, 0, 0],
       [0, 1, 0, 0]])
```

8.1.1 数组创建

（5）full()：full((m,n),c)可以生成一个m×n的元素全为c的矩阵。

```
>>> a = np.full((2,3),4)
>>> a
array([[4, 4, 4],
       [4, 4, 4]])
```

（6）random.random()：random.random((m,n))生成一个m×n的元素为0~1之间随机数的矩阵。

```
>>> a = np.random.random((2,3))
>>> a
array([[0.46657535, 0.2398773 , 0.18675721],
       [0.30525201, 0.66826887, 0.5708038 ]])
```

8.1.1 数组创建

（7）random.randint()：numpy.random.randint(low, high=None, size=None, dtype='l') 函数的作用是，返回一个随机整型数，范围从低（包括）到高（不包括），即[low, high)。如果没有写参数high的值，则返回[0,low)的值。

```
>>> a = np.random.randint(2, size=10)
>>> a
array([0, 1, 0, 0, 1, 1, 0, 0, 1, 1])
>>> b = np.random.randint(5, size=(2, 4))
>>> b
array([[1, 2, 3, 3],
       [0, 0, 2, 4]])
```

8.1.1 数组创建

（8）random.rand()：random.rand(d0,d1,…,dn) 函数根据给定维度生成[0,1)之间的数据，其中，dn表示每个维度的元素个数。

```
>>> a = np.random.rand(4,2)
>>> a
array([[0.22225254, 0.25555882],
       [0.69250455, 0.62957494],
       [0.567664  , 0.30459249],
       [0.16394031, 0.00900947]])
```

（9）random.randn()：random.randn(d0,d1,…,dn) 函数返回一个或一组样本，具有标准正态分布，其中，dn表示每个维度的元素个数。

```
>>> a = np.random.randn(2,4)
>>> a
array([[-0.28183753, -0.4931384 , -2.11355842,  0.17782074],
       [-1.14089585,  0.816798  ,  0.39287532, -0.19339946]])
```

8.1.2 数组索引和切片

与Python列表类似，可以对NumPy数组进行索引和切片。由于数组可能是多维的，因此，必须为数组的每个维度指定一个索引或切片。具体实例如下：

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[5]
5
>>> a[5:8]
array([5, 6, 7])
>>> a[5:8] = 12
>>> a
array([ 0,  1,  2,  3,  4, 12, 12, 12,  8,  9])
>>> a = np.arange(10)
>>> a_slice = a[5:8]
```

8.1.2 数组索引和切片

```
>>> a_slice[0] = -1
>>> a_slice
array([-1,  6,  7])
>>> a
array([ 0,  1,  2,  3,  4, -1,  6,  7,  8,  9])
>>> b = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> b[2]
array([7, 8, 9])
>>> b[0][2]
3
>>> b[0,2]
3
>>> b[:2]
array([[1, 2, 3],
       [4, 5, 6]])
```

8.1.2 数组索引和切片

```
>>> b[:2,1:]
array([[2, 3],
       [5, 6]])
>>> b[1,:2]
array([4, 5])
>>> b[:2,2]
array([3, 6])
>>> b[:,:1]
array([[1],
       [4],
       [7]])
```

8.1.3 数组运算

数组运算实质上是数组对应位置的元素进行运算，常见的是加、减、乘、除、开方等运算。具体实例如下：

```
>>> a = np.array([[1,2,3],[4,5,6]])
>>> a*a
array([[ 1,  4,  9],
       [16, 25, 36]])
>>> a-a
array([[0, 0, 0],
       [0, 0, 0]])
>>> 1/a
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
>>> a+a
array([[ 2,  4,  6],
          [ 8, 10, 12]])
```

8.1.3 数组运算

```
>>> np.exp(a)  # e的幂次方
array([[  2.71828183,   7.3890561 ,  20.08553692],
       [ 54.59815003, 148.4131591 , 403.42879349]])
>>> np.sqrt(a)
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974]])
>>> a**2
array([[ 1,  4,  9],
       [16, 25, 36]], dtype=int32)
```

8.2 pandas数据结构

本节介绍pandas数据结构，包括Series、DataFrame和索引对象。在开展具体实验之前，首先要打开一个cmd窗口执行如下命令安装pandas：

> pip install pandas

8.2.1Series
8.2.2 DataFrame
8.2.3索引对象

8.2.1Series

Series是一种类似于一维数组的对象，它由一维数组以及一组与之相关的数据标签（即索引）组成，仅由一组数据即可产生最简单的Series。Series的字符串表现形式为：索引在左边，值在右边。如果没有为数据指定索引，就会自动创建一个0到N-1(N为数据的长度)的整数型索引。可以通过Series的values和index属性获取其数组表现形式和索引对象。

下面是具体实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> obj=Series([3,5,6,8,9,2])
>>> obj
0    3
1    5
2    6
3    8
4    9
5    2
dtype: int64
>>> obj.index
RangeIndex(start=0, stop=6, step=1)
```

8.2.1Series

上面的代码中，我们没有为数据指定索引，因此，pandas会自动创建一个整型索引。现在，我们创建对数据点进行标记的索引，作为演示，下面继续执行如下代码：

```
>>> obj2=Series([3,5,6,8,9,2],index=['a','b','c','d','e','f'])
>>> obj2
a    3
b    5
c    6
d    8
e    9
f    2
dtype: int64
>>> obj2.index
Index(['a', 'b', 'c', 'd', 'e', 'f'], dtype='object')
```

8.2.1Series

创建好Series以后，可以利用索引的方式选取Series的单个或一组值。作为演示，下面继续执行如下代码：

```
>>> obj2['a']
3
>>> obj2[['b','d','f']]
b    5
d    8
f    2
dtype: int64
```

8.2.1Series

可以对Series进行NumPy数组运算。作为演示，下面继续执行如下代码：

```
>>> obj2[obj2>5]
c    6
d    8
e    9
dtype: int64
>>> obj2*2    #乘以2
a     6
b    10
c    12
d    16
e    18
f     4
dtype: int64
```

8.2.1Series

```
>>> np.exp(obj2)    #求e的幂次方
a      20.085537
b     148.413159
c     403.428793
d    2980.957987
e    8103.083928
f       7.389056
dtype: float64
```

8.2.1Series

可以将Series看成是一个有定长的有序字典，因为它是索引值到数据值的一个映射。因此，一些字典函数也可以在这里使用。作为演示，下面继续执行如下代码：

```
>>> 'b' in obj2
True
>>> 'm' in obj2
False
```

此外，也可以用字典创建Series。作为演示，下面继续执行如下代码：

```
>>> dic={'m':4,'n':5,'p':6}
>>> obj3=Series(dic)
>>> obj3
m    4
n    5
p    6
dtype: int64
```

8.2.1Series

使用字典生成Series时，可以指定额外的索引。如果额外的索引与字典中的键不匹配，则不匹配的索引部分数据为NaN。作为演示，下面继续执行如下代码：

```
>>> ind=['m','n','p','a']
>>> obj4=Series(dic,index=ind)
>>> obj4
m    4.0
n    5.0
p    6.0
a    NaN
dtype: float64
```

8.2.1Series

pandas提供了isnull()和notnull()函数用于检测缺失数据。作为演示，下面继续执行如下代码：

```
>>> pd.isnull(obj4)
m    False
n    False
p    False
a     True
dtype: bool
>>> pd.notnull(obj4)
m     True
n     True
p     True
a    False
dtype: bool
```

8.2.1Series

可以对不同的Series进行算术运算，在运算过程中，pandas会自动对齐不同索引的数据。作为演示，下面继续执行如下代码：

```
>>> obj3+obj4
a     NaN
m     8.0
n    10.0
p    12.0
dtype: float64
```

Series对象本身及其索引都有一个name属性。作为演示，下面继续执行如下代码：

```
>>> obj4.name='sereis_a'
>>> obj4.index.name='letter'
>>> obj4
letter
m    4.0
n    5.0
p    6.0
a    NaN
Name: sereis_a, dtype: float64
```

8.2.1Series

Series的索引可以通过赋值的方式进行改变。作为演示，下面继续执行如下代码：

```
>>> obj4.index=['u','v','w','a']
>>> obj4
u    4.0
v    5.0
w    6.0
a    NaN
Name: sereis_a, dtype: float64
```

8.2.2 DataFrame

DataFrame是一个表格型的数据结构，它含有一组有序的列，每列可以是不同的值类型（数值、字符串、布尔值等）。DataFrame既有行索引也有列索引，它可以被看作由Series组成的字典（公用同一个索引）。跟其他类似的数据结构相比，DataFrame中面向行和面向列的操作基本是平衡的。其实，DataFrame中的数据是以一个或多个二维块存放的（而不是列表、字典或者别的一维数据结构）。

8.2.2 DataFrame

下面是具体实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> data = {'sno':['95001', '95002', '95003', '95004'],
            'name':['Xiaoming','Zhangsan','Lisi','Wangwu'],
            'sex':['M','F','F','M'],
            'age':[22,25,24,23]}
>>> frame=DataFrame(data)
>>> frame
    sno     name  sex  age
0  95001  Xiaoming   M   22
1  95002  Zhangsan   F   25
2  95003      Lisi   F   24
3  95004    Wangwu   M   23
```

8.2.2 DataFrame

从执行结果可以看出，虽然没有指定行索引，但是，pandas会自动添加索引。如果指定列索引，则会按照指定顺序排列。作为演示，继续执行如下代码：

```
>>> frame=DataFrame(data,columns=['name','sno','sex','age'])
>>> frame
      name    sno  sex  age
0  Xiaoming  95001    M   22
1  Zhangsan  95002    F   25
2      Lisi  95003    F   24
3    Wangwu  95004    M   23
```

8.2.2 DataFrame

在制定列索引时，如果存在不匹配的列，则不匹配的列的值为NaN：

```
>>> frame=DataFrame(data,columns=['sno','name','sex','age','grade'])
>>> frame
    sno      name sex  age grade
0  95001  Xiaoming   M   22   NaN
1  95002  Zhangsan   F   25   NaN
2  95003      Lisi   F   24   NaN
3  95004    Wangwu   M   23   NaN
```

8.2.2 DataFrame

可以同时指定行索引和列索引：

```
>>>
frame=DataFrame(data,columns=['sno','name','sex','age','grade'],index=['a',
'b','c','d'])
>>> frame
    sno      name sex  age grade
a  95001  Xiaoming   M   22   NaN
b  95002  Zhangsan   F   25   NaN
c  95003      Lisi   F   24   NaN
d  95004    Wangwu   M   23   NaN
```

8.2.2 DataFrame

通过类似字典标记或属性的方式，可以获取Series（列数据）：

```
>>> frame['sno']
a    95001
b    95002
c    95003
d    95004
Name: sno, dtype: object
>>> frame.name
a    Xiaoming
b    Zhangsan
c        Lisi
d      Wangwu
Name: name, dtype: object
```

8.2.2 DataFrame

行也可以通过位置或名称获取：

```
>>> frame.loc['b']
sno         95002
name     Zhangsan
sex             F
age            25
grade         NaN
Name: b, dtype: object
>>> frame.iloc[1]
sno         95002
name     Zhangsan
sex             F
age            25
grade         NaN
Name: b, dtype: object
```

8.2.2 DataFrame

或者，也可以采用“切片”的方式一次获取多个行：

```
>>> frame.loc['b':'c']
     sno      name sex  age grade
b  95002  Zhangsan   F   25   NaN
c  95003      Lisi   F   24   NaN
>>> frame.iloc[2:4]
     sno    name sex  age grade
c  95003    Lisi   F   24   NaN
d  95004  Wangwu   M   23   NaN
```

8.2.2 DataFrame

可以用“切片”的方式使用列名称获取1个列：

```
>>> frame.loc[:,['sex']]
  sex
a   M
b   F
c   F
d   M
```

也可以用“切片”的方式使用列名称获取多个列：

```
>>> frame.loc[:,'sex':]
  sex  age grade
a   M   22   NaN
b   F   25   NaN
c   F   24   NaN
d   M   23   NaN
```

上面的代码在截取列时，是从sex列开始截取，并把sex之后的所有列都截取出来。

8.2.2 DataFrame

还可以用“切片”的方式使用列索引获取多个列：

```
>>> frame.iloc[:,1:4]
      name sex  age
a  Xiaoming   M   22
b  Zhangsan   F   25
c      Lisi   F   24
d    Wangwu   M   23
```

上面的代码在截取列时，从索引号为1的列开始截取，也就是从name列开始截取，一直截取到索引号为4的列之前（不含索引号为4的列）。

8.2.2 DataFrame

可以给列赋值，赋值是列表的时候，列表中元素的个数必须和数据的行数匹配：

```
>>> frame['grade']=[93,89,72,84]
>>> frame
    sno      name sex  age  grade
a  95001  Xiaoming   M   22     93
b  95002  Zhangsan   F   25     89
c  95003      Lisi   F   24     72
d  95004    Wangwu   M   23     84
```

8.2.2 DataFrame

可以用一个Series修改一个DataFrame的值，将精确匹配DataFrame的索引，空位将补上缺失值：

```
>>> frame['grade']=Series([67,89],index=['a','c'])
>>> frame
    sno      name sex  age  grade
a  95001  Xiaoming   M   22   67.0
b  95002  Zhangsan   F   25    NaN
c  95003      Lisi   F   24   89.0
d  95004    Wangwu   M   23    NaN
```

8.2.2 DataFrame

可以增加一个新的列：

```
>>> frame['province']=['ZheJiang','FuJian','Beijing','ShangHai']
>>> frame
    sno      name sex  age  grade  province
a  95001  Xiaoming   M   22   67.0  ZheJiang
b  95002  Zhangsan   F   25    NaN   FuJian
c  95003      Lisi   F   24   89.0   Beijing
d  95004    Wangwu   M   23    NaN  ShangHai
```

当不再需要一个列时，可以删除该列：

```
>>> del frame['province']
>>> frame
    sno      name sex  age  grade
a  95001  Xiaoming   M   22   67.0
b  95002  Zhangsan   F   25    NaN
c  95003      Lisi   F   24   89.0
d  95004    Wangwu   M   23    NaN
```

8.2.2 DataFrame

可以把嵌套字典(字典的字典)作为参数，传入DataFrame，其中，外层字典的键作为列（column），内层键作为行索引(index)：

```
>>>
dic={'computer':{2020:78,2021:82},'math':{2019:76,2020:78,2021:81}}
>>> frame1=DataFrame(dic)
>>> frame1
     computer  math
2020      78.0    78
2021      82.0    81
2019       NaN    76
```

可以对结果进行转置：

```
>>> frame1.T
          2020  2021  2019
computer  78.0  82.0   NaN
math      78.0  81.0  76.0
```

8.2.2 DataFrame

还可以指定行索引，对于不匹配的行会返回NaN：

```
>>> frame2=DataFrame(dic,index=[2020,2021,2022])
>>> frame2
      computer  math
2020      78.0  78.0
2021      82.0  81.0
2022       NaN   NaN
```

8.2.2 DataFrame

下面用NumPy的相关模块来生成DataFrame：

```
>>> import numpy as np
>>> import pandas as pd
>>> #用顺序数np.arange(12).reshape(3,4)
>>> df1=pd.DataFrame(np.arange(12).reshape(3,4),columns=['a','b','c','d'])
>>> df1
   a  b   c   d
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
>>> #用随机数np.random.randint(20,size=(2,3))
>>> df2=pd.DataFrame(np.random.randint(20,size=(2,3)),columns=['b','d','a'])
>>> df2
    b   d  a
0   0  19  4
1  10   2  5
```

8.2.2 DataFrame

```
>>> #用随机数np.random.randn(5,3)
>>>
df3=pd.DataFrame(np.random.randn(5,3),index=list('abcde'),columns=['one','two','three'])
>>> df3
        one       two     three
a -0.204225 -0.402101 -0.528857
b  0.070463 -1.203973 -1.271088
c -1.210856  0.438507  1.442583
d -0.101521  1.283724 -0.101034
e -1.256007 -0.112633 -1.590732
```

8.2.2 DataFrame

此外，也可以从表格型数据文件中读取数据生成DataFrame。pandas提供了一些用于将表格型数据读取为DataFrame对象的函数，其中常用的函数包括read_csv()和read_table()，具体用法如下：

```
>>> import pandas as pd
>>> from pandas import DataFrame
>>> csv_df = pd.read_csv('C:\\Python38\my_file.csv')
>>> table_df = pd.read_table('C:\\Python38\my_table.txt')
```

8.2.3索引对象

pandas的索引（Index）对象负责管理轴标签和轴名称等。构建Series或DataFrame时，所用到的任何数组或其他序列的标签都会被转换成一个Index对象。 Index对象是不可修改的，Series和DataFrame中的索引都是Index对象。

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame,Index
>>> # 获取Index对象
>>> x = Series(range(3), index = ['a', 'b', 'c'])
>>> index = x.index
>>> index
Index(['a', 'b', 'c'], dtype='object')
>>> index[0:2]
Index(['a', 'b'], dtype='object')
```

8.2.3索引对象

```
>>> # 构造/使用Index对象
>>> index = Index(np.arange(3))
>>> obj2 = Series([2.5, -3.5, 0], index = index)
>>> obj2
0    2.5
1   -3.5
2    0.0
dtype: float64
>>> obj2.index is index
True
>>> # 判断列/行索引是否存在
>>> data = {'pop':[2.3,2.6],
            'year':[2020,2021]}
>>> frame = DataFrame(data)
>>> frame
   pop  year
0  2.3  2020
1  2.6  2021
```

8.2.3索引对象

```
>>> 'pop' in frame.columns
True
>>> 1 in frame.index
True
```

8.3 基本功能

8.3.1 重新索引
8.3.2 丢弃指定轴上的项
8.3.3 索引、选取和过滤
8.3.4 算术运算
8.3.5 DataFrame和Series之间的运算
8.3.6 函数应用和映射
8.3.7 排序和排名
8.3.8分组
8.3.9 shape函数
8.3.10 info()函数
8.3.11 cut()函数

8.3.1 重新索引

pandas中的reindex方法可以为Series和DataFrame添加或者删除索引。如果新添加的索引没有对应的值，则默认为NaN。如果减少索引，就相当于一个切片操作。

下面是对Series使用reindex方法的实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
    >>> s1 = Series([1, 2, 3, 4], index=['A', 'B', 'C', 'D'])
>>> s1
A    1
B    2
C    3
D    4
dtype: int64
```

8.3.1 重新索引

```
>>> # 重新指定index， 多出来的index，可以使用fill_value填充
>>> s1.reindex(index=['A', 'B', 'C', 'D', 'E'], fill_value = 10)
A     1
B     2
C     3
D     4
E    10
dtype: int64
>>> s2 = Series(['A', 'B', 'C'], index = [1, 5, 10])
```

8.3.1 重新索引

```
>>> # 修改索引，将s2的索引增加到15个，如果新增加的索引值不存在，默认为NaN
>>> s2.reindex(index=range(15))
0    NaN
1      A
2    NaN
3    NaN
4    NaN
5      B
6    NaN
7    NaN
8    NaN
9    NaN
10     C
11   NaN
12   NaN
13   NaN
14   NaN
dtype: object
```

8.3.1 重新索引

```
>>> # ffill： 表示forward fill，向前填充
>>> # 如果新增加索引的值不存在，那么按照前一个非NaN的值填充进去
>>> s2.reindex(index=range(15), method='ffill')
0    NaN
1      A
2      A
3      A
4      A
5      B
6      B
7      B
8      B
9      B
10     C
11     C
12     C
13     C
14     C
dtype: object
```

8.3.1 重新索引

```
>>> # 减少index
>>> s1.reindex(['A', 'B'])
A    1
B    2
dtype: int64
```

8.3.1 重新索引

下面是对DataFrame使用reindex方法的实例：

```
>>> df1 = DataFrame(np.random.rand(25).reshape([5, 5]), index=['A', 'B', 'D',
'E', 'F'], columns=['c1', 'c2', 'c3', 'c4', 'c5'])
>>> df1
        c1        c2        c3        c4        c5
A  0.077539  0.574105  0.868985  0.305669  0.738754
B  0.939470  0.464108  0.951791  0.277599  0.091289
D  0.019077  0.850392  0.069981  0.397684  0.526270
E  0.564420  0.723089  0.971805  0.501211  0.641450
F  0.308109  0.831558  0.215271  0.729247  0.944689
```

8.3.1 重新索引

```
>>> # 为DataFrame添加一个新的索引
>>> # 可以看到自动扩充为NaN
>>> df1.reindex(index=['A', 'B', 'C', 'D', 'E', 'F'])
         c1        c2        c3        c4        c5
A  0.077539  0.574105  0.868985  0.305669  0.738754
B  0.939470  0.464108  0.951791  0.277599  0.091289
C       NaN       NaN       NaN       NaN       NaN
D  0.019077  0.850392  0.069981  0.397684  0.526270
E  0.564420  0.723089  0.971805  0.501211  0.641450
F  0.308109  0.831558  0.215271  0.729247  0.944689
```

8.3.1 重新索引

```
>>> # 扩充列
>>> df1.reindex(columns=['c1', 'c2', 'c3', 'c4', 'c5', 'c6'])
         c1        c2        c3        c4        c5  c6
A  0.077539  0.574105  0.868985  0.305669  0.738754 NaN
B  0.939470  0.464108  0.951791  0.277599  0.091289 NaN
D  0.019077  0.850392  0.069981  0.397684  0.526270 NaN
E  0.564420  0.723089  0.971805  0.501211  0.641450 NaN
F  0.308109  0.831558  0.215271  0.729247  0.944689 NaN
>>> # 减少index
>>> df1.reindex(index=['A', 'B'])
         c1        c2        c3        c4        c5
A  0.077539  0.574105  0.868985  0.305669  0.738754
B  0.939470  0.464108  0.951791  0.277599  0.091289
```

8.3.2 丢弃指定轴上的项

可以使用drop()方法丢弃指定轴上的项，drop()方法返回的是一个在指定轴上删除了指定值的新对象。具体实例如下：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> # Series根据行索引删除行
>>> s1 = Series(np.arange(4), index = ['a', 'b', 'c','d'])
>>> s1
a    0
b    1
c    2
d    3
dtype: int32
>>> s1.drop(['a', 'b'])
c    2
d    3
dtype: int32
```

8.3.2 丢弃指定轴上的项

```
>>> # DataFrame根据索引行/列删除行/列
>>> df1 = DataFrame(np.arange(16).reshape((4, 4)),
                    index = ['a', 'b', 'c', 'd'],
                    columns = ['A', 'B', 'C', 'D'])
>>> df1
    A   B   C   D
a   0   1   2   3
b   4   5   6   7
c   8   9  10  11
d  12  13  14  15
```

8.3.2 丢弃指定轴上的项

```
>>> df1.drop(['A','B'],axis=1)   #在列的维度上删除AB两行，axis值为1表示列的维度
   C   D
a  2   3
b  6   7
c  10  11
d  14  15
>>> df1.drop('a', axis = 0)   #在行的维度上删除行，axis值为0表示行的维度
   A   B   C   D
b  4   5   6   7
c  8   9  10  11
d  12  13  14  15
>>> df1.drop(['a', 'b'], axis = 0)
   A   B   C   D
c  8   9  10  11
d  12  13  14  15
```

8.3.3 索引、选取和过滤

下面是关于DataFrame的索引、选取和过滤的一些实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> # DataFrame的索引
>>> data = DataFrame(np.arange(16).reshape((4, 4)),
                     index = ['a', 'b', 'c', 'd'],
                     columns = ['A', 'B', 'C', 'D'])
>>> data
    A   B   C   D
a   0   1   2   3
b   4   5   6   7
c   8   9  10  11
d  12  13  14  15
```

8.3.3 索引、选取和过滤

```
>>> data['A']  #打印列
a     0
b     4
c     8
d    12
Name: A, dtype: int32
>>> data[['A', 'B']]   #花式索引
    A   B
a   0   1
b   4   5
c   8   9
d  12  13
```

8.3.3 索引、选取和过滤

```
>>> data[:2]   #切片索引,选择行
   A  B  C  D
a  0  1  2  3
b  4  5  6  7
>>> # 根据条件选择
>>> data
    A   B   C   D
a   0   1   2   3
b   4   5   6   7
c   8   9  10  11
d  12  13  14  15
>>> data[data.A > 5]  #根据条件选择行
    A   B   C   D
c   8   9  10  11
d  12  13  14  15
```

8.3.3 索引、选取和过滤

```
>>> data < 5    #打印True或者False
       A      B      C      D
a   True   True   True   True
b   True  False  False  False
c  False  False  False  False
d  False  False  False  False
>>> data[data < 5] = 0  #条件索引
>>> data
    A   B   C   D
a   0   0   0   0
b   0   5   6   7
c   8   9  10  11
d  12  13  14  15
```

8.3.4 算术运算

下面是关于DataFrame算术运算的实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
    >>> df1 = DataFrame(np.arange(12).reshape((3,4)),columns=list("abcd"))
>>> df2 = DataFrame(np.arange(20).reshape((4,5)),columns=list("abcde"))
>>> df1
   a  b   c   d
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11
```

8.3.4 算术运算

```
>>> df2
   a   b   c   d   e
0  0   1   2   3   4
1  5   6   7   8   9
2  10  11  12  13  14
3   15  16  17  18  19

>>> df1+df2
     a     b     c     d   e
0   0.0   2.0   4.0   6.0 NaN
1   9.0  11.0  13.0  15.0 NaN
2  18.0  20.0  22.0  24.0 NaN
3   NaN   NaN   NaN   NaN NaN
```

8.3.4 算术运算

```
>>> df1.add(df2,fill_value=0)  #为df1添加第3行和e这一列
     a     b     c     d     e
0   0.0   2.0   4.0   6.0   4.0
1   9.0  11.0  13.0  15.0   9.0
2  18.0  20.0  22.0  24.0  14.0
3  15.0  16.0  17.0  18.0  19.0
>>> df1.add(df2).fillna(0)  #按照正常方式将df1和df2相加，然后将NaN值填充为0
     a     b     c     d    e
0   0.0   2.0   4.0   6.0  0.0
1   9.0  11.0  13.0  15.0  0.0
2  18.0  20.0  22.0  24.0  0.0
3   0.0   0.0   0.0   0.0  0.0
```

8.3.5 DataFrame和Series之间的运算

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
    >>> frame = DataFrame(np.arange(12).reshape((4,3)),columns=list("bde"),
                index=["Beijing","Shanghai","Shenzhen","Xiamen"])
>>> frame
          b   d   e
Beijing   0   1   2
Shanghai  3   4   5
Shenzhen  6   7   8
Xiamen    9  10  11
>>> frame.iloc[1]  # 获取某一行数据
b    3
d    4
e    5
Name: Shanghai, dtype: int32
```

8.3.5 DataFrame和Series之间的运算

```
>>> frame.index  #获取索引
Index(['Beijing', 'Shanghai', 'Shenzhen', 'Xiamen'], dtype='object')
>>> frame.loc["Xiamen"]    #根据行索引提取数据
b     9
d    10
e    11
Name: Xiamen, dtype: int32
>>> series = frame.iloc[0]
>>> series
b    0
d    1
e    2
Name: Beijing, dtype: int32
```

8.3.5 DataFrame和Series之间的运算

```
>>> frame - series
        b  d  e
Beijing   0  0  0
Shanghai  3  3  3
Shenzhen  6  6  6
Xiamen    9  9  9
```

8.3.6 函数应用和映射

用apply()将一个规则应用到DataFrame的行或者列上，实例如下：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
   >>> frame = DataFrame(np.arange(12).reshape((4,3)),columns=list("bde"),
              index=["Beijing","Shanghai","Shenzhen","Xiamen"])
>>> frame
         b   d   e
Beijing   0   1   2
Shanghai  3   4   5
Shenzhen  6   7   8
Xiamen    9  10  11
```

8.3.6 函数应用和映射

```
>>> f = lambda x : x.max() - x.min()  # 匿名函数
>>> frame.apply(f)  #apply默认第二个参数axis=0,作用于列方向上，
axis=1时作用于行方向上
b    9
d    9
e    9
dtype: int64
>>> frame.apply(f,axis=1)
Beijing     2
Shanghai    2
Shenzhen    2
Xiamen      2
dtype: int64
```

8.3.6 函数应用和映射

可以使用applymap()将一个规则应用到DataFrame中的每一个元素，实例如下：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
    >>> frame = DataFrame(np.arange(12).reshape((4,3)),columns=list("bde"),
                index=["Beijing","Shanghai","Shenzhen","Xiamen"])
>>> frame
          b   d   e
Beijing   0   1   2
Shanghai  3   4   5
Shenzhen  6   7   8
Xiamen    9  10  11
```

8.3.6 函数应用和映射

```
>>> f = lambda num : "%.2f"%num  #匿名函数
>>> # 将匿名函数f应用到frame中的每一元素中
>>> strFrame = frame.applymap(f)
>>> strFrame
             b      d      e
Beijing   0.00   1.00   2.00
Shanghai  3.00   4.00   5.00
Shenzhen  6.00   7.00   8.00
Xiamen    9.00  10.00  11.00
>>> frame.dtypes  # 获取DataFrame中每一列的数据类型
b    int32
d    int32
e    int32
dtype: object
```

8.3.6 函数应用和映射

```
>>> strFrame.dtypes
b    object
d    object
e    object
dtype: object
>>> # 将一个规则应用到某一列上
>>> frame["d"].map(lambda x :x+10)
Beijing     11
Shanghai    14
Shenzhen    17
Xiamen      20
Name: d, dtype: int64
```

8.3.7 排序和排名

1.排序

下面是对Series和DataFrame进行排序的实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> series = Series(range(4),index=list("dabc"))
>>> series
d    0
a    1
b    2
c    3
dtype: int64
>>> series.sort_index() #索引按字母顺序排序
a    1
b    2
c    3
d    0
dtype: int64
```

8.3.7 排序和排名

```
>>> frame = DataFrame(np.arange(8).reshape((2,4)),
                      index=["three","one"],
                      columns=list("dabc"))
>>> frame
       d  a  b  c
three  0  1  2  3
one    4  5  6  7
>>> frame.sort_index()
       d  a  b  c
one    4  5  6  7
three  0  1  2  3
>>> frame.sort_index(axis=1,ascending=False)
       d  c  b  a
three  0  3  2  1
one    4  7  6  5
```

8.3.7 排序和排名

```
>>> # 按照DataFrame中某一列的值进行排序
>>> df = DataFrame({"a":[4,7,-3,2],"b":[0,1,0,1]})
>>> df
   a  b
0  4  0
1  7  1
2 -3  0
3  2  1
>>> # 按照b这一列的数据值进行排序
>>> df.sort_values(by="b")
   a  b
0  4  0
2 -3  0
1  7  1
3  2  1
```

8.3.7 排序和排名

2.排名

排名（rank）是指根据值的大小/出现次数来进行排名，得到一组排名值。排名跟排序关系密切，且它会增设一个排名值（从1开始，一直到数组中有效数据的数量）。默认情况下，rank()通过将平均排名分配到每个组打破平级关系。也就是说，如果有两组数值一样，那他们的排名将会被加在一起再除以2。

8.3.7 排序和排名

下面是一些具体实例：

```
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> obj=Series([7,-4,7,3,2,0,5])
>>> obj.rank()
0    6.5
1    1.0
2    6.5
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

8.3.7 排序和排名

在上面的代码中，rank()没有任何参数，所以method是采用默认值'average'。这时，对于obj中的7,-4,7,3,2,0,5，我们可以手工进行排名，-4排第1名，0排第2名，2排第3名，3排第4名，5排第5名，第1个7排第6名，第2个7排第7名，出现了两个值相等的7，也就是出现了平级关系，因此，取二者排名的平均值6.5来破坏平级关系。因此，在obj.rank()的结果中，第0行是6.5（说明这一行的7排名是6.5），第1行是1.0（说明这一行的-4排名是1.0），第2行是6.5（说明这一行的7排名是6.5），第3行是4.0（说明这一行的3排名是4.0），以此类推。

8.3.7 排序和排名

然后，继续执行如下代码：

```
>>> obj.rank(method='first')
0    6.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

在上面的代码中，method的取值为'first'，这时，如果出现平级关系，就按值在原始数据中的出现顺序分配排名。可以看到，obj中出现了两个7，也就是出现了平级关系，这时，谁先出现，谁就排在前面，因此，第1个7排第6名，第2个7排第7名。

8.3.7 排序和排名

然后，继续执行如下代码：

```
>>> obj.rank(method='min')
0    6.0
1    1.0
2    6.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

在上面的代码中，method的取值为'min'，这时如果出现平级关系，就使用整个分组的最小值排名。可以看到，obj中出现了两个7，也就是出现了平级关系，这时，第1个7排第6名，第2个7排第7名，我们就取二者较小的排名作为二者的排名，因此，第0行的值是6.0，第2行也是6.0。

8.3.7 排序和排名

然后，继续执行如下代码：

```
>>> obj.rank(method='max')
0    7.0
1    1.0
2    7.0
3    4.0
4    3.0
5    2.0
6    5.0
dtype: float64
```

在上面的代码中，method的取值为'max'，这时如果出现平级关系，就使用整个分组的最小值排名。可以看到，obj中出现了两个7，也就是出现了平级关系，这时，第1个7排第6名，第2个7排第7名，我们就取二者较大的排名作为二者的排名，因此，第0行的值是7.0，第2行也是7.0。

8.3.7 排序和排名

也可以对DataFrame使用rank()，实例如下：

```
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> frame=DataFrame({'b':[3,1,5,2],'a':[8,4,3,7],'c':[2,7,9,4]})
>>> frame
   b  a  c
0  3  8  2
1  1  4  7
2  5  3  9
3  2  7  4
>>> frame.rank(axis=1)  # axis=0时作用于列方向上，axis=1时作用于行方向上
     b    a    c
0  2.0  3.0  1.0
1  1.0  2.0  3.0
2  2.0  1.0  3.0
3  1.0  3.0  2.0
```

8.3.8分组

1.分组操作

下面是分组操作的实例：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> dict_obj = {'key1' : ['a', 'b', 'a', 'b', 'a', 'b', 'a', 'a'],
                'key2' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],
                'data1': np.random.randn(8),
                'data2': np.random.randn(8)}
>>> df_obj = DataFrame(dict_obj)
```

8.3.8分组

```
>>> df_obj
  key1   key2     data1     data2
0    a    one -0.026042  0.051420
1    b    one -0.214902 -1.245808
2    a    two -0.626813  0.313240
3    b  three -1.074137  0.245969
4    a    two  0.106360 -0.344038
5    b    two -0.719663 -0.877795
6    a    one -0.248008 -0.650183
7    a  three  0.861269  1.388312
>>> df_obj.groupby('key1')
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000000037E93D0>
>>> type(df_obj.groupby('key1'))
<class 'pandas.core.groupby.generic.DataFrameGroupBy'>
>>> df_obj['data1'].groupby(df_obj['key1'])
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000000000B4E7D00>
>>> type(df_obj['data1'].groupby(df_obj['key1']))
<class 'pandas.core.groupby.generic.SeriesGroupBy'>
```

8.3.8分组

2.分组运算

下面是分组运算的实例：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> dict_obj = {'key1' : ['a', 'b', 'a', 'b', 'a', 'b', 'a', 'a'],
                'key2' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],
                'data1': np.random.randn(8),
                'data2': np.random.randn(8)}
>>> df_obj = DataFrame(dict_obj)
>>> df_obj
  key1   key2     data1     data2
0    a    one -0.026042  0.051420
1    b    one -0.214902 -1.245808
2    a    two -0.626813  0.313240
3    b  three -1.074137  0.245969
4    a    two  0.106360 -0.344038
5    b    two -0.719663 -0.877795
6    a    one -0.248008 -0.650183
7    a  three  0.861269  1.388312
```

8.3.8分组

```
>>> grouped1 = df_obj.groupby('key1')
>>> grouped1.mean()
         data1     data2
key1
a     0.013353  0.151750
b    -0.669567 -0.625878
>>> grouped2 = df_obj['data1'].groupby(df_obj['key1'])
>>>grouped2.mean()
key1
a    0.013353
b   -0.669567
Name: data1, dtype: float64
>>> grouped1.size()  #返回每个分组的元素个数
key1
a    5
b    3
dtype: int64
```

8.3.8分组

```
>>> grouped2.size()  #返回每个分组的元素个数
key1
a    5
b    3
Name: data1, dtype: int64
>>> df_obj.groupby([df_obj['key1'], df_obj['key2']]).size()
key1  key2
a     one      2
      three    1
      two      2
b     one      1
      three    1
      two      1
dtype: int64
```

8.3.8分组

```
>>> grouped3 = df_obj.groupby(['key1', 'key2'])
>>> grouped3.size()
key1  key2
a     one      2
      three    1
      two      2
b     one      1
      three    1
      two      1
dtype: int64
>>> grouped3.mean()
              data1     data2
key1 key2
a    one   -0.137025 -0.299382
     three  0.861269  1.388312
     two   -0.260226 -0.015399
b    one   -0.214902 -1.245808
     three -1.074137  0.245969
     two   -0.719663 -0.877795
```

8.3.8分组

3.按照自定义的key分组

pandas支持按照自定义的key进行分组，具体实例如下：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> dict_obj = {'key1' : ['a', 'b', 'a', 'b', 'a', 'b', 'a', 'a'],
                'key2' : ['one', 'one', 'two', 'three','two', 'two', 'one', 'three'],
                'data1': np.random.randn(8),
                'data2': np.random.randn(8)}
>>> df_obj = DataFrame(dict_obj)
```

8.3.8分组

```
>>> df_obj
  key1   key2     data1     data2
0    a    one -0.026042  0.051420
1    b    one -0.214902 -1.245808
2    a    two -0.626813  0.313240
3    b  three -1.074137  0.245969
4    a    two  0.106360 -0.344038
5    b    two -0.719663 -0.877795
6    a    one -0.248008 -0.650183
7    a  three  0.861269  1.388312
>>> self_def_key = [0, 1, 2, 3, 3, 4, 5, 7]
>>> df_obj.groupby(self_def_key).size()
0    1
1    1
2    1
3    2
4    1
5    1
7    1
```

8.3.9 shape函数

DataFrame的shape函数用于返回DataFrame的形状，具体用法如下：

- shape:返回DataFrame包含几行几列；
- shape[0]：返回DataFrame包含几行；
- shape[1]：返回DataFrame包含几列。

8.3.9 shape函数

具体实例如下：

```
>>> import pandas as pd
>>> from pandas import DataFrame
>>> frame=DataFrame({'b':[3,1,5,2],'a':[8,4,3,7],'c':[2,7,9,4]})
>>> frame
   b  a  c
0  3  8  2
1  1  4  7
2  5  3  9
3  2  7  4
>>> frame.shape
(4, 3)
>>> frame.shape[0]
4
>>> frame.shape[1]
3
```

8.3.10 info()函数

info()函数用于返回DataFrame的基本信息（维度、列名称、数据格式、所占空间等）。具体实例如下：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import DataFrame
>>> df=
DataFrame({'id':[1,np.nan,3,4],'name':['asx',np.nan,'wes','asd'],'score':[78,90,np.nan,88]},index=list('abcd'))
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
Index: 4 entries, a to d
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   id      3 non-null      float64
 1   name    3 non-null      object
 2   score   3 non-null      float64
dtypes: float64(2), object(1)
memory usage: 128.0+ bytes
```

8.3.11 cut()函数

cut()函数用于将数据进行离散化，将连续变量进行分段汇总，该方法仅适用于一维数组状对象。如果我们有大量标量数据并对其进行一些统计分析, 则可以使用cut()方法。其语法形式如下：

pandas.cut(x，bins，right = True,labels = None,retbins = False,precision = 3,include_lowest = False)

其中，各个参数的含义如下：

•x：一维数组；

•bins：是指一个整型或标量序列，这些值定义用于分割的箱子的边缘。如果是整数，则表示将x划分为多少个等距的区间；如果是序列，则表示将x划分在指定序列中，若不在该序列中，则是NaN；

•right：是否包含右端点；

•labels：是否用标记来代替返回的箱子；

•precision：精度；

•include_lowest：是否包含左端点。

8.3.11 cut()函数

下面是具体实例：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import DataFrame
>>> info_nums = DataFrame({'num': np.random.randint(1, 50, 11)})
>>> info_nums
    num
0    43
1     7
2    13
3    47
4    23
5    10
6    44
7     2
8    31
9    21
10   47
```

8.3.11 cut()函数

```
>>> info_nums['num_bins'] = pd.cut(x=info_nums['num'], bins=[1, 25, 50])
>>> info_nums
    num  num_bins
0    43  (25, 50]
1     7   (1, 25]
2    13   (1, 25]
3    47  (25, 50]
4    23   (1, 25]
5    10   (1, 25]
6    44  (25, 50]
7     2   (1, 25]
8    31  (25, 50]
9    21   (1, 25]
10   47  (25, 50]
>>> info_nums['num_bins'].unique()
[(25, 50], (1, 25]]
Categories (2, interval[int64]): [(1, 25] < (25, 50]]
```

8.3.11 cut()函数

下面演示如何向箱子添加标签：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import DataFrame
>>> info_nums = DataFrame({'num': np.random.randint(1, 10, 7)})
>>> info_nums
   num
0    4
1    5
2    6
3    8
4    6
5    2
6    3
```

8.3.11 cut()函数

```
>>> info_nums['nums_labels'] = pd.cut(x=info_nums['num'], bins=[1, 7,
10], labels=['Lows', 'Highs'], right=False)
>>> info_nums
   num nums_labels
0    4        Lows
1    5        Lows
2    6        Lows
3    8       Highs
4    6        Lows
5    2        Lows
6    3        Lows
>>> info_nums['nums_labels'].unique()
['Lows', 'Highs']
Categories (2, object): ['Lows' < 'Highs']
```

8.4 汇总和描述统计

8.4.1 与描述统计相关的函数
8.4.2 唯一值、值计数以及成员资格

8.4.1 与描述统计相关的函数

下面是一些具体实例：

```
>>> import numpy as np
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> df=DataFrame([[1.3,np.nan],[6.2,-3.4],[np.nan,np.nan],[0.65,-1.4]],columns=['one','two'])
>>> df.sum()  #计算每列的和，默认排除NaN
one    8.15
two   -4.80
dtype: float64
>>> df.sum(axis=1)  #计算每行的和，默认排除NaN
0    1.30
1    2.80
2    0.00
3   -0.75
dtype: float64
```

8.4.1 与描述统计相关的函数

```
>>> #计算每行的和，设置skipna=False，NaN参与计算，结果仍为NaN
>>> df.sum(axis=1,skipna=False)
0     NaN
1    2.80
2     NaN
3   -0.75
dtype: float64
>>> df.mean(axis=1)
0    1.300
1    1.400
2      NaN
3   -0.375
dtype: float64
>>> df.mean(axis=1,skipna=False)  #计算每行的平均值，NaN参与运算
0     NaN
1    1.400
2     NaN
3   -0.375
dtype: float64
```

8.4.1 与描述统计相关的函数

```
>>> df.cumsum()   #求样本值的累计和
    one  two
0  1.30  NaN
1  7.50 -3.4
2   NaN  NaN
3  8.15 -4.8
>>> df.describe()    #针对列计算汇总统计
            one       two
count  3.000000  2.000000
mean   2.716667 -2.400000
std    3.034112  1.414214
min    0.650000 -3.400000
25%    0.975000 -2.900000
50%    1.300000 -2.400000
75%    3.750000 -1.900000
max    6.200000 -1.400000
```

8.4.2 唯一值、值计数以及成员资格

1.唯一值和值计数

```
>>> import pandas as pd
>>> from pandas import Series
    >>> s = Series([3,3,1,2,4,3,4,6,5,6])
>>> #判断Series中的值是否重复,False表示重复
>>> print(s.is_unique)
False
>>> #输出Series中不重复的值,返回值没有排序，返回值的类型为数组
>>> s.unique()
array([3, 1, 2, 4, 6, 5], dtype=int64)
>>> #统计Series中重复值出现的次数,默认是按出现次数降序排序
>>> s.value_counts()
3    3
4    2
6    2
1    1
2    1
5    1
dtype: int64
```

8.4.2 唯一值、值计数以及成员资格

```
>>> #按照重复值的大小排序输出频率
>>> s.value_counts(sort=False)
1    1
2    1
3    3
4    2
5    1
6    2
dtype: int64
```

8.4.2 唯一值、值计数以及成员资格

2.成员资格判断

下面是关于成员资格判断的具体实例：

```
>>> import pandas as pd
>>> from pandas import Series,DataFrame
>>> s = Series([6,6,7,2,2])
>>> s
0    6
1    6
2    7
3    2
4    2
dtype: int64
>>> #判断矢量化集合的成员资格,返回一个布尔类型的Series
>>> s.isin([6])
0     True
1     True
2    False
3    False
4    False
dtype: bool
```

8.4.2 唯一值、值计数以及成员资格

```
>>> type(s.isin([6]))
<class 'pandas.core.series.Series'>
>>> #通过成员资格方法选取Series中的数据子集
>>> s[s.isin([6])]
0    6
1    6
dtype: int64
>>> data = [[4,3,7],[3,2,5],[7,3,6]]
>>> df = DataFrame(data,index=["a","b","c"],columns=["one","two","three"])
>>> df
   one  two  three
a    4    3      7
b    3    2      5
c    7    3      6
```

8.4.2 唯一值、值计数以及成员资格

```
>>> #返回一个布尔型的DataFrame
>>> df.isin([2])
     one    two  three
a  False  False  False
b  False   True  False
c  False  False  False
>>> #选取DataFrame中值为2的数，其他的为NaN
>>> df[df.isin([2])]
   one  two  three
a  NaN  NaN    NaN
b  NaN  2.0    NaN
c  NaN  NaN    NaN
>>> #选取DataFrame中值为2的数，将NaN用0进行填充
>>> df[df.isin([2])].fillna(0)
   one  two  three
a  0.0  0.0    0.0
b  0.0  2.0    0.0
c  0.0  0.0    0.0
```

8.5 处理缺失数据

8.5.1检查缺失值
8.5.2清理/填充缺失值
8.5.3 丢失缺少的值

8.5.1检查缺失值

为了更容易地检测缺失值，pandas提供了isnull()和notnull()函数，它们也是Series和DataFrame对象的方法。下面是具体实例：

```
    >>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> df = DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e',
'f','h'],columns=['one', 'two', 'three'])
>>> df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
>>> df['one'].isnull()
a    False
b     True
c    False
d     True
e    False
f    False
g     True
h    False
Name: one, dtype: bool
```

```
>>> df['one'].notnull()
a     True
b    False
c     True
d    False
e     True
f     True
g    False
h     True
Name: one, dtype: bool
```

8.5.2清理/填充缺失值

pandas提供了各种方法来清除缺失的值。fillna()函数可以通过几种方法用非空数据“填充”缺失值。下面是具体实例：

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> df = pd.DataFrame(np.random.randn(3, 3), index=['a', 'c',
'e'],columns=['one','two', 'three'])
>>> df = df.reindex(['a', 'b', 'c'])
>>> df
        one       two     three
a -0.963024 -0.284216 -1.762598
b       NaN       NaN       NaN
c  0.677290  0.320812 -0.145247
```

```
>>> df.fillna(0)  #用0填充缺失值
        one       two     three
a -0.963024 -0.284216 -1.762598
b  0.000000  0.000000  0.000000
c  0.677290  0.320812 -0.145247
>>> df.fillna(method='pad')  #填充时和前一行的数据相同
        one       two     three
a -0.963024 -0.284216 -1.762598
b -0.963024 -0.284216 -1.762598
c  0.677290  0.320812 -0.145247
>>> df.fillna(method='backfill')  #填充时和后一行的数据相同
        one       two     three
a -0.963024 -0.284216 -1.762598
b  0.677290  0.320812 -0.145247
c  0.677290  0.320812 -0.145247
```

8.5.3 丢失缺少的值

如果只想排除缺少的值，则使用dropna()函数和axis()参数。 默认情况下，axis = 0，即在行上应用，这意味着如果行内的任何值缺失，那么整个行被排除。

```
>>> import pandas as pd
>>> import numpy as np
>>> from pandas import Series,DataFrame
>>> df = DataFrame(np.random.randn(5, 3), index=['a', 'c', 'e',
'f','h'],columns=['one', 'two', 'three'])
>>> df = df.reindex(['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'])
>>> df
        one       two     three
a -0.249220 -0.003033 -0.615404
b       NaN       NaN       NaN
c  0.034787 -0.056103 -0.389375
d       NaN       NaN       NaN
e -0.453844  1.131537  0.273852
f -0.895511 -0.306457 -0.135208
g       NaN       NaN       NaN
h  0.701194  0.556521 -0.341591
```

8.5.3 丢失缺少的值

```
>>> df.dropna()  #默认情况下，axis = 0，即在行上应用
        one       two     three
a -0.249220 -0.003033 -0.615404
c  0.034787 -0.056103 -0.389375
e -0.453844  1.131537  0.273852
f -0.895511 -0.306457 -0.135208
h  0.701194  0.556521 -0.341591
>>> df.dropna(axis=1)  # axis = 1时在列上应用
Empty DataFrame
Columns: []
Index: [a, b, c, d, e, f, g, h]
```

8.5.3 丢失缺少的值

```
>>> # 可以用一些具体的值取代一个通用的值
>>> df = DataFrame({'one':[1,2,3,4,5,300],'two':[200,0,3,4,5,6]})
>>> df
   one  two
0    1  200
1    2    0
2    3    3
3    4    4
4    5    5
5  300    6
>>> df.replace({200:10,300:60})
   one  two
0    1   10
1    2    0
2    3    3
3    4    4
4    5    5
5   60    6
```

8.6 综合实例

8.6.1 Matplotlib的使用方法
8.6.2实例1：对一个数据集进行基本操作
8.6.3实例2：百度搜索指数分析
8.6.4实例3：电影评分数据分析
8.6.5实例4：APP行为数据预处理（请直接参考教材）

8.6.1 Matplotlib的使用方法

Matplotlib是Python最著名的绘图库，它提供了一整套和Matlab相似的命令API，十分适合交互式地进行制图。而且也可以方便地将它作为绘图控件，嵌入到GUI应用程序中。Matplotlib能够创建多数类型的图表，如条形图、散点图、条形图、饼图、堆叠图、3D图和地图图表。

Python安装好以后，默认是没有安装Matplotlib库的，需要单独安装。在Windows系统中打开一个cmd窗口，执行如下命令安装Matplotlib库：

```
> pip install matplotlib
```

8.6.1 Matplotlib的使用方法

下面介绍如何使用Matplotlib绘制一些简单的图表。
首先要导入pyplot模块：

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

接下来，我们调用plot方法绘制一些坐标：

```
>>> plt.plot([1,2,3],[4,8,5])
```

plot()方法需要很多参数，但是最主要的是前2个参数，分别表示x坐标和y坐标，比如，上面语句中放入了两个列表[1,2,3]和[4,8,5]，就表示生成了3个坐标(1,4)、(2,8)和(3,5)。
下面可以把图表显示到屏幕上（如图8-1所示）：

```
>>> plt.show()
```

8.6.1 Matplotlib的使用方法

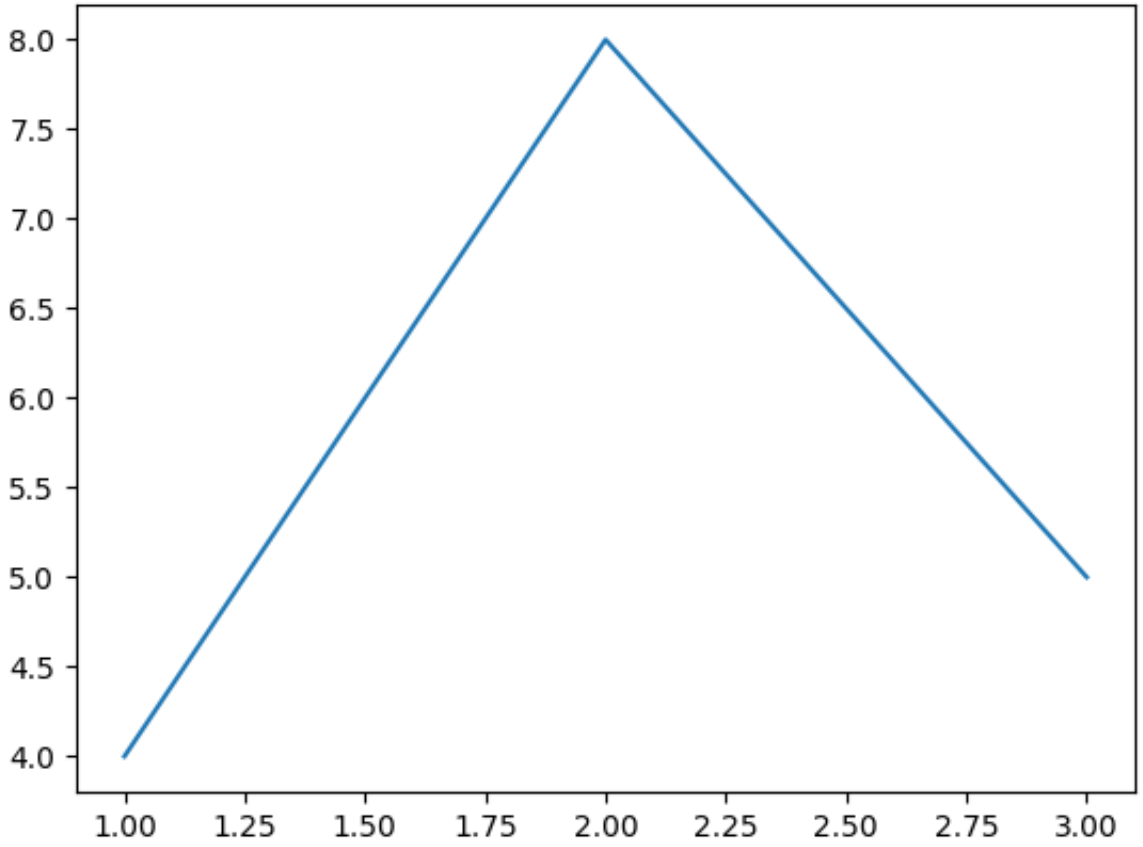


图8-1 三个坐标点生成的折线图

8.6.1 Matplotlib的使用方法

下面画出两条折线，并且给每条折线一个名称：

```
>>> x = [1,2,3]    #第1条折线的横坐标
>>> y = [4,8,5]    #第1条折线的纵坐标
>>> x2 = [1,2,3]      #第2条折线的横坐标
>>> y2 = [11,15,13]    #第2条折线的纵坐标
>>> plt.plot(x, y, label='First Line')    #绘制第1条折线，给折线一个名称'First Line'
>>> plt.plot(x2, y2, label='Second Line')  #绘制第2条折线，给折线一个名称'Second Line'
>>> plt.xlabel('Plot Number')    #给横坐标轴添加名称
>>> plt.ylabel('Important var')    #给纵坐标轴添加名称
>>> plt.title('Graph Example\nTwo lines')  #添加标题
>>> plt.legend()   #添加图例
>>> plt.show()   #显示到屏幕上（如图8-2所示）
```

8.6.1 Matplotlib的使用方法

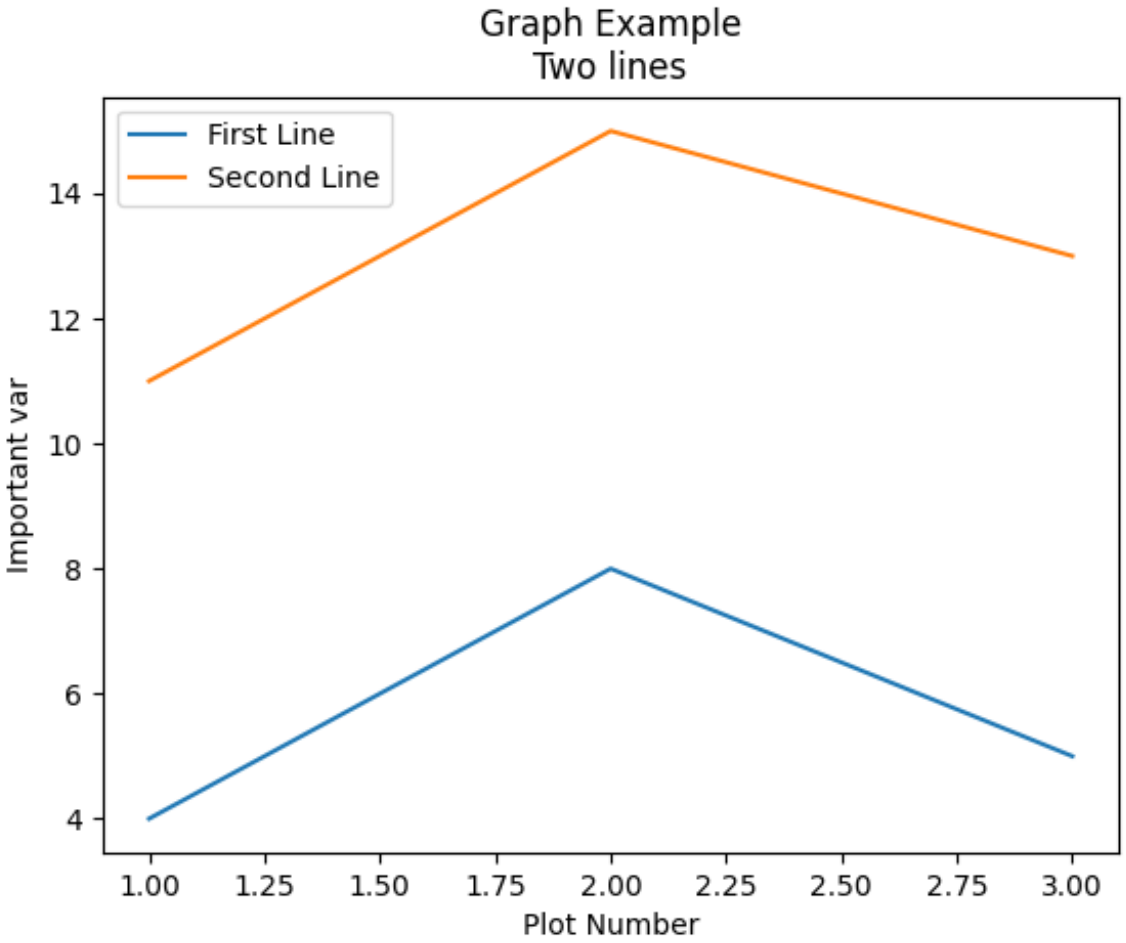


图8-2 绘制两条折线

8.6.1 Matplotlib的使用方法

下面介绍条形图的绘制方法。

```
>>> plt.bar([1,3,5,7,9],[6,3,8,9,2], label="First Bar")  #第1个数据系列
>>> #下面的color='g'，表示设置颜色为绿色
>>> plt.bar([2,4,6,8,10],[9,7,3,6,7], label="Second Bar", color='g')  #第2个数据系列
>>> plt.legend()  #添加图例
>>> plt.xlabel('bar number')  #给横坐标轴添加名称
>>> plt.ylabel('bar height')  #给纵坐标轴添加名称
>>> plt.title('Bar Example\nTwo bars!')  #添加标题
>>> plt.show()  #显示到屏幕上（如图8-3所示）
```

8.6.1 Matplotlib的使用方法

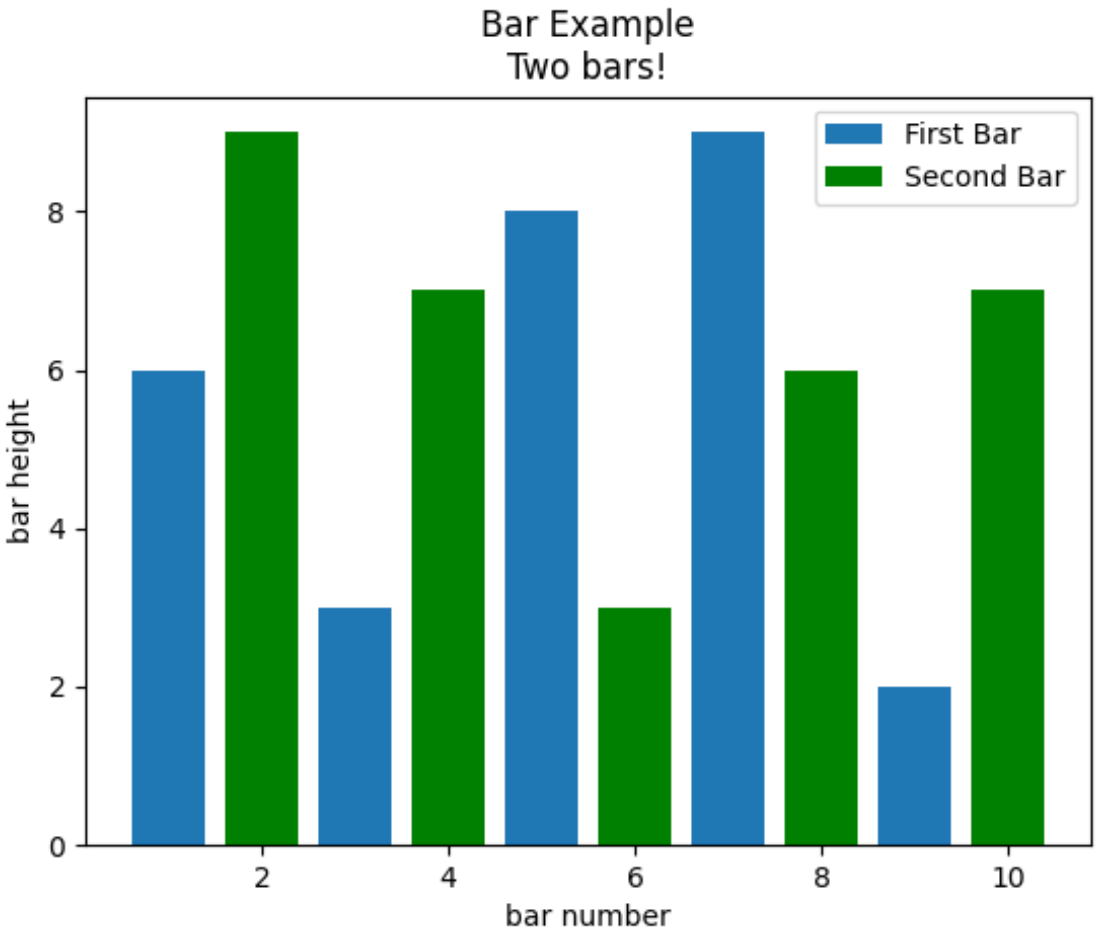


图8-3 条形图

8.6.1 Matplotlib的使用方法

下面介绍直方图的绘制方法。

```
>>> population_ages = [21,57,61,47,25,21,33,41,41,5,96,103,108,
                       121,122,123,131,112,114,113,82,77,67,56,46,44,45,47]
>>> bins = [0,10,20,30,40,50,60,70,80,90,100,110,120,130]
>>> plt.hist(population_ages, bins, histtype='bar', rwidth=0.8)
>>> plt.xlabel('x')
>>> plt.ylabel('y')
>>> plt.title('Graph Example\n Histogram')
>>> plt.show()  #显示到屏幕上（如图8-4所示）
```

8.6.1 Matplotlib的使用方法

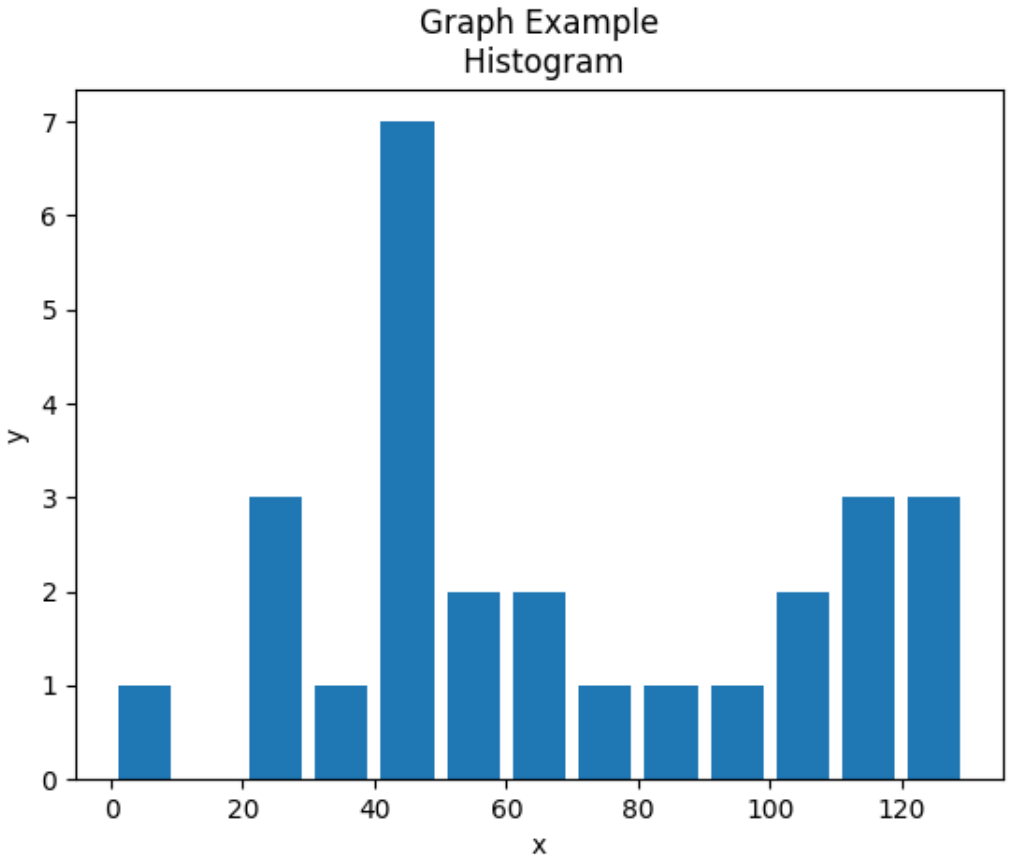


图8-4 直方图

8.6.1 Matplotlib的使用方法

下面介绍饼图的绘制方法。

```
>>> slices = [7,2,2,13]  #即activities分别占比7/24,2/,2/24,13/24
>>> activities = ['sleeping','eating','working','playing']
>>> cols = ['c','m','r','b']
>>> plt.pie(slices,
                labels=activities,
colors=cols,
startangle=90,
                shadow= True,
                explode=(0,0.1,0,0),
                autopct='%1.1f%%')
>>> plt.title('Graph Example\n Pie chart')
>>> plt.show()    #显示到屏幕上（如图8-5所示）
```

8.6.1 Matplotlib的使用方法

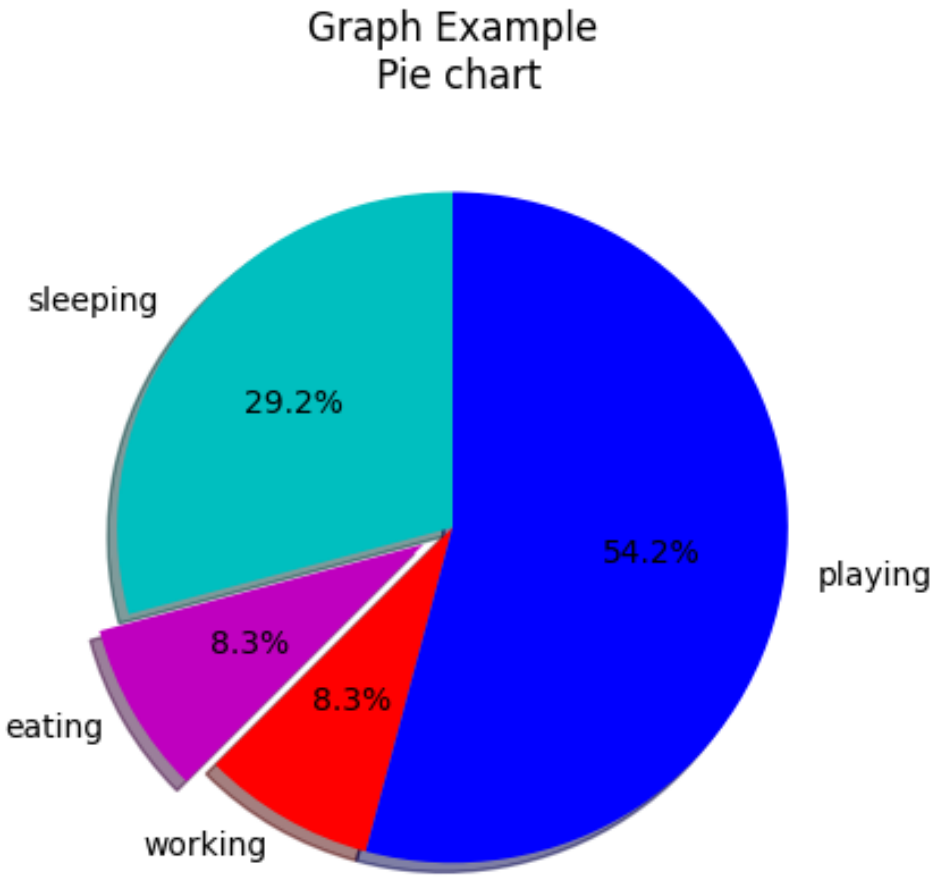


图8-5 饼图

8.6.2实例1：对一个数据集进行基本操作

假设有一个关于食品信息的数据集food_info.csv，下面利用这个数据集进行一些基本的数据分析操作：

```
    >>> import pandas
>>> food_info = pandas.read_csv("D:\\food_info.csv")
>>> #使用head()方法读取前几行数据，参数为空时默认展示5行数据，可以传入其他数字，如4、9等；
>>> food_info.head()
>>> #使用tail()方法倒着从后读取后几行数据，参数为空时默认展示5行数据，可以传入其他数字，如4、9等；
>>> food_info.tail()
>>> #使用columns方法，打印列名，作用是可以看到每一列的数据代表的含义
>>> food_info.columns
>>> #使用shape方法，打印数据的维度，一共有几行几列
>>> food_info.shape
```

8.6.2实例1：对一个数据集进行基本操作

```
>>> #也可以使用切片操作，例如取第3-5行的数据
>>> food_info.loc[3:5]
>>> #也可以传进去一个列表，例如打印 4，6，9 行的数据
>>> food_info.loc[[4,6,9]]
>>> #通过列名取一列数据
>>> food_info['Water_(g)']
>>> #通过好几个列名取好几列数据，参数是一个含有多个列名的列表
>>> food_info[['Water_(g)','Ash_(g)']]
>>> #找出以“(g)”结尾的列，取前3行数据打印出来
>>> col_names = food_info.columns.tolist()
>>> gram_columns = []
>>> for i in col_names:
        if i.endswith("(g)"):
            gram_columns.append(i)
>>> gram_df = food_info[gram_columns]
>>> print(gram_df.head(3))
```

8.6.2实例1：对一个数据集进行基本操作

可以给DataFrame添加一列新的特征。可能对某个列的数值进行一些简单的数学运算，又可以得到一列新的特征，比如，某一列的数据是以mg为单位的，现在想要该列数据以g为单位，加入到特征列中。继续执行如下代码：

```
>>> iron_gram = food_info["Iron_(mg)"]/1000
>>> food_info["Iron_(g)"] = iron_gram
>>> food_info.shape
>>> #对某一列进行归一化操作，比如列中的每个元素都除以该列的最大值
>>> normalized_fat = food_info["Lipid_Tot_(g)"]/
food_info["Lipid_Tot_(g)"].max()
>>> print(normalized_fat)
```

8.6.2实例1：对一个数据集进行基本操作

还可以对某一列的值进行排序操作，调用sort_values()方法，传入参数是列名，inplace属性决定了是再产生一个新列，还是在原列基础上排序，ascending属性决定了是正序还是倒序排列，默认为True，正序排列。
继续执行如下代码：

```
>>> food_info.sort_values("Sodium_(mg)",inplace = True, ascending = False)
>>> print(food_info['Sodium_(mg)'])
```

8.6.3实例2：百度搜索指数分析

给定一个百度搜索指数表baidu_index.xls，里面包含了id、keyword、index、date四个字段（如图8-6所示），每行数据记录了某个关键词在某天被搜索的次数，比如，第1行数据的含义是，“缤智”这个关键词在2018年12月1日一共被搜索了2699次。要求计算出每个车型每个月的搜索指数（即一个月总共被搜索的次数）。

	A	B	C	D
1	id	keyword	index	date
2	1	缤智	2699	2018-12-1
3	2	缤智	2767	2018-12-2
4	3	缤智	2866	2018-12-3
5	4	缤智	2872	2018-12-4
6	5	缤智	2739	2018-12-5

图8-6 百度指数趋势表

8.6.3实例2：百度搜索指数分析

为了让pandas能够顺利读取Excel表格文件，需要安装第三方库xlrd和openpyxl。打开一个cmd窗口，执行如下命令安装第三方库：

```
> pip install xlrd
> pip install openpyxl
```

打开百度指数趋势表baidu_index.xls，发现有如下问题需要处理：

- 对于个别车型，近期才有数据，之前没有数据，需要对缺失值进行处理；
- 结果是需要月级数据，但是原始数据是按天的，需要对日期进行处理；
- 对于原始数据关键词keyword字段，为防止合并时出现大小写区别而合并错误，需要对关键词进行统一处理。

8.6.3实例2：百度搜索指数分析

在IDLE中执行如下命令：

```
>>> import numpy as np
>>> import pandas as pd
>>> index=pd.read_excel('D:\\baidu\\baidu_index.xls')
>>> # 处理缺失值
>>> index = index.fillna(0)
```

下面查看index中的“date”字段类型：

```
>>> index['date'].head()
0   2018-12-01
1   2018-12-02
2   2018-12-03
3   2018-12-04
4   2018-12-05
Name: date, dtype: datetime64[ns]
```

从返回结果“dtype: datetime64[ns]”可以看出，“date”字段属于日期类型，注意，如果这里不是日期类型，而是字符串类型（这时返回的信息会是“dtype:object”），则必须使用to_datetime()函数进行转换。

8.6.3实例2：百度搜索指数分析

下面对日期进行转换，只保留月份：

```
>>> index['date']
0      2018-12-01
1      2018-12-02
2      2018-12-03
3      2018-12-04
4      2018-12-05
          ...
Name: date, Length: 6344, dtype: datetime64[ns]
>>> index['date'] = index['date'].dt.strftime('%B')
>>> index['date']
0      December
1      December
2      December
3      December
4      December
          ...
Name: date, Length: 6344, dtype: object
```

8.6.3实例2：百度搜索指数分析

上面的语句中使用了DataFrame的列数据的dt接口，这个接口可以帮我们快速实现特定的功能，这里调用了dt接口下的strftime()函数，用于对日期进行格式化处理，格式化字符串'%B'表示返回月份的英文单词，如“一月”则返回“January”。

8.6.3实例2：百度搜索指数分析

下面对“keyword”字段进行数据处理，删除字段中的所有空白符，并且把英文字母全部转化为大写字母：

```
>>> index['keyword']
        ...
6339    T-cross
6340    T-cross
6341    T-cross
6342    T-cross
6343    T-cross
Name: keyword, Length: 6344, dtype: object
>>> index['keyword'] = index['keyword'].apply(lambda x: x.strip('
\r\n\t').upper())
>>> index['keyword']
        ...
6339    T-CROSS
6340    T-CROSS
6341    T-CROSS
6342    T-CROSS
6343    T-CROSS
Name: keyword, Length: 6344, dtype: object
```

8.6.3实例2：百度搜索指数分析

下面根据“keyword”和“date”字段对搜索指数进行分类汇总求和：

```
>>> new_index_mean = index.groupby(['keyword','date'])['index'].sum()
>>> new_index_mean
keyword   date
IX25      April       29144.0
          December    32422.0
          February    28511.0
          January     32204.0
          June          882.0
                        ...
雪铁龙C3-XR  June          184.0
          March        9967.0
          May          6419.0
          November     6346.0
          October      7757.0
Name: index, Length: 234, dtype: float64
```

8.6.4实例3：电影评分数据分析

有一个电影评分数据集IMDB-Movie-Data.csv，里面包含了电影标题、类型、导演、演员、上映年份、电影时长、评分、收入等信息，下面使用pandas、NumPy和Matplotlib对数据集进行分析。

```
>>> import matplotlib.pyplot as plt
>>> import numpy as np
>>> import pandas as pd
>>> # 读取数据
>>> movie = pd.read_csv("D:\\IMDB-Movie-Data.csv")
>>> #查看前5条数据
>>> movie.head()
   >>> #求出电影评分的平均分
>>> movie['Rating'].mean()
```

8.6.4实例3：电影评分数据分析

下面要求出导演人数信息，导演人数可能重复，因此需要使用np.unique()进行数据去重，求出唯一值，然后使用shape方法获取导演人数。

```
>>> np.unique(movie['Director']).shape[0]
```

下面以直方图的形式呈现电影评分的数据分布。

```
>>> # 创建画布
>>> plt.figure(figsize=(20, 8), dpi=100)
>>> # 绘制图像
>>> plt.hist(movie["Rating"].values, bins=20)
>>> # 添加刻度
>>> max_ = movie["Rating"].max()
>>> min_ = movie["Rating"].min()
>>> t1 = np.linspace(min_, max_, num=21)
>>> plt.xticks(t1)
>>> # 添加网格
>>> plt.grid()
>>> # 显示
>>> plt.show()   #显示到屏幕上（如图8-7所示）
```

8.6.4实例3：电影评分数据分析

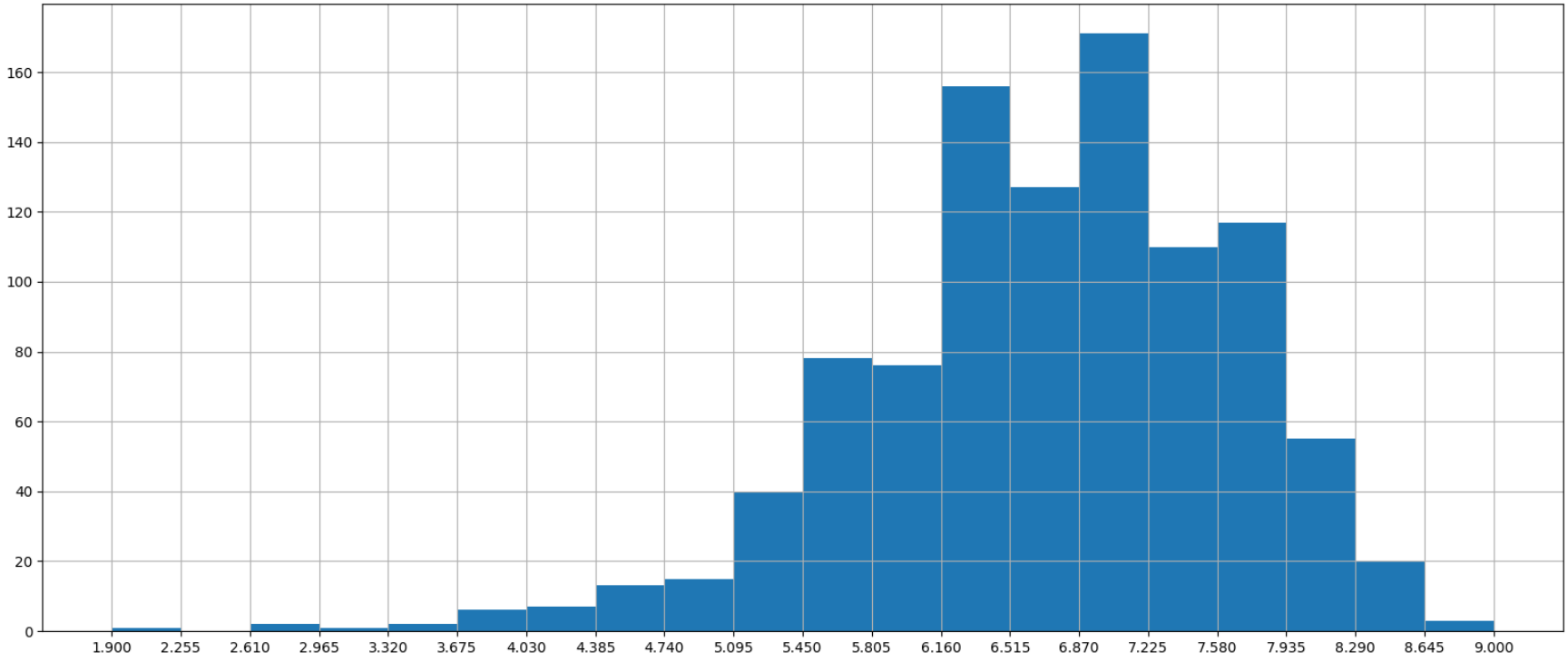


图8-7 电影评分分布图

8.6.4实例3：电影评分数据分析

下面以直方图的形式呈现电影时长的数据分布。

```
>>> # 查看电影时长
>>> runtime_data = movie["Runtime (Minutes)"]
>>> # 创建画布
>>> plt.figure(figsize=(20,8),dpi=80)
>>> # 求出最大值和最小值
>>> max_ = runtime_data.max()
>>> min_ = runtime_data.min()
>>> num_bin = (max_-min_)//5
>>> # 绘制图像
>>> plt.hist(runtime_data,num_bin)
>>> # 添加刻度
>>> plt.xticks(range(min_,max_+5,5))
>>> # 添加网格
>>> plt.grid()
>>> plt.show()    #显示到屏幕上（如图8-8所示）
```

8.6.4实例3：电影评分数据分析

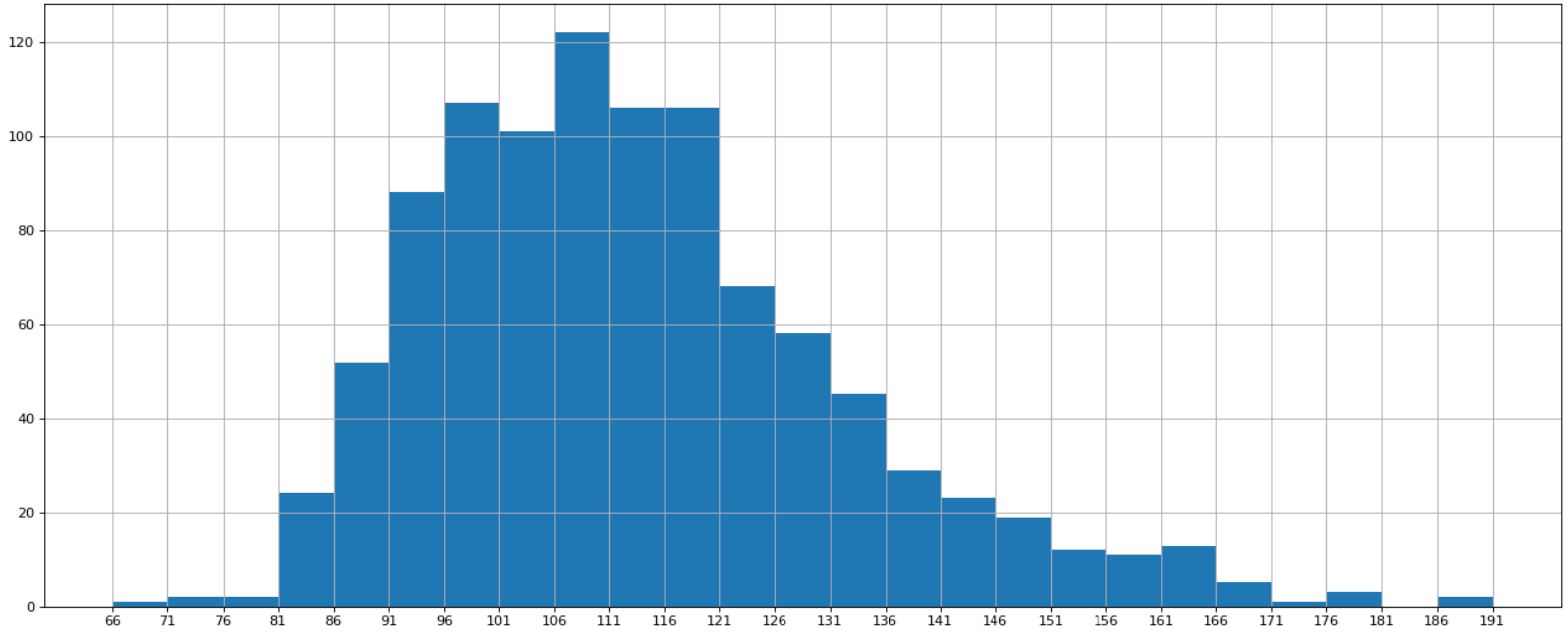


图8-8 电影时长分布图

8.6.4实例3：电影评分数据分析

下面继续求评分平均数、导演人数、演员人数。

```
>>> # 查看评分平均数
>>> movie["Rating"].mean()
>>> # 查看导演人数
>>> np.unique(movie["Director"]).shape[0]
>>> len(set(movie["Director"].tolist()))
>>> # 查看演员人数
>>> num = movie["Actors"].str.split(',').tolist()
>>> actor_nums = [j for i in num for j in i]
>>> len(set(actor_nums))
```

8.6.4实例3：电影评分数据分析

下面统计电影分类情况。

```
>>> movie["Genre"].head()
0     Action,Adventure,Sci-Fi
1    Adventure,Mystery,Sci-Fi
2              Horror,Thriller
3     Animation,Comedy,Family
4    Action,Adventure,Fantasy
Name: Genre, dtype: object
```

从上面执行结果可以看出，每部电影会属于多个分类。因此，统计每个分类电影的个数的基本思路是（如图8-9所示），创建一个DataFrame，取每一个分类名为列名，行填充为0，当某部电影属于某个分类时，对应的0替换成1。最后，对每个列的1进行求和，就可以计算出每个分类的电影个数。

8.6.4实例3：电影评分数据分析

	Genre
1	“b,c,d”
2	“c,e,f”
3	“a,b,g”
4	“d,e,f”
5	“a,c,f”

a	b	c	d	e	f	g
0	1	1	1	0	0	0
0	0	1	0	1	1	0
1	1	0	0	0	0	1
0	0	0	1	1	1	0
1	0	1	0	0	1	0

图8-9 统计电影分类个数的思路

8.6.4实例3：电影评分数据分析

```
>>> # 将'Genre'转化为列表
>>> temp_list = [i for i in movie['Genre']]
>>> # 去除分隔符，变成二维数组
>>> temp_list = [i.split(sep=',') for i in movie['Genre']]
>>> # 提取二维数组中元素
>>> [i for j in temp_list for i in j]
>>> # 去重，得到所有电影类别
>>> array_list = np.unique([i for j in temp_list for i in j])
>>> #创建一个全为0的DataFrame，列索引置为电影的分类
>>> array_list.shape
>>> movie.shape
>>> np.zeros((movie.shape[0], array_list.shape[0]))
>>> genre_zero = pd.DataFrame(np.zeros((movie.shape[0],
array_list.shape[0])),
                                    columns=array_list,
                                    index=movie["Title"])
```

8.6.4实例3：电影评分数据分析

```
>>> #遍历每一部电影，DataFrame中把分类出现的列的值置为1
>>> for i  in range(movie.shape[0]):
        genre_zero.iloc[i, genre_zero.columns.get_indexer(temp_list[i])] = 1
>>> genre_zero
>>> # 对每个分类求和
>>> genre_zero.sum(axis=0)
>>> # 排序、画图
>>> new_zeros = genre_zero.sum(axis=0)
>>> new_zeros
>>> genre_count = new_zeros.sort_values(ascending=False)
>>> x_ = genre_count.index
>>> y_ = genre_count.values
>>> plt.figure(figsize=(20,8),dpi=80)
>>> plt.bar(range(len(x_)),y_,width=0.4,color="orange")
>>> plt.xticks(range(len(x_)),x_)
>>> plt.show()    #显示到屏幕上（如图8-10所示）
```

8.6.4实例3：电影评分数据分析

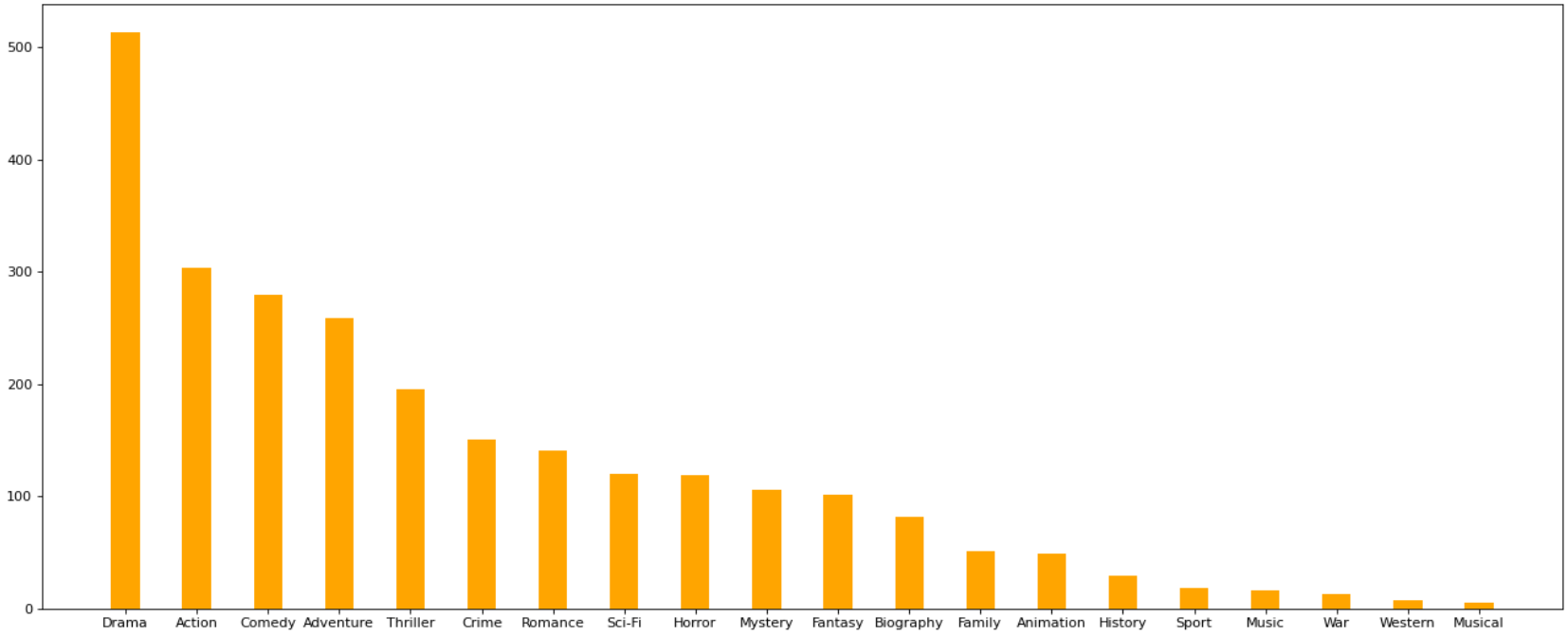


图8-10 每个分类的电影个数分布图

8.7 本章小结

pandas是基于NumPy 的一种工具，该工具是为解决数据分析任务而创建的，它是使Python成为强大而高效的数据分析环境的重要因素之一。本章内容首先介绍了Series和DataFrame这两种数据结构，然后，介绍了pandas的一些基本功能，包括重新索引、丢弃指定轴上的项、索引过滤和选取、算术运算、函数应用和映射、排序和排名等，接下来介绍了与描述统计相关的函数、唯一值、值计数以及成员资格判断等，同时也介绍了缺失数据的处理，最后，通过4个综合实例展示了pandas的应用方法。

附录A：主讲教师林子雨简介

主讲教师：林子雨

单位：厦门大学计算机科学与技术系
E-mail: ziyulin@xmu.edu.cn
个人网页：http://dblab.xmu.edu.cn/linziyu
数据库实验室网站：http://dblab.xmu.edu.cn

扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，厦门大学信息学院实验教学中心主任，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学青年基金项目(No.61303004)、1项福建省自然科学青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过400万次，累计访问量超过1500万次。

附录B：大数据学习路线图

大数据学习路线图访问地址：http://dblab.xmu.edu.cn/post/10164/

附录C：林子雨大数据系列教材

了解全部教材信息：http://dblab.xmu.edu.cn/post/bigdatabook/

附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材

本课程旨在实现以下几个培养目标：

- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：http://dblab.xmu.edu.cn/post/bigdataintroduction/

附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：http://dblab.xmu.edu.cn/post/bigdata-introduction/

开设大数据专业导论课的优质教材

扫一扫访问教材官网

附录F：《大数据技术原理与应用（第3版）》教材

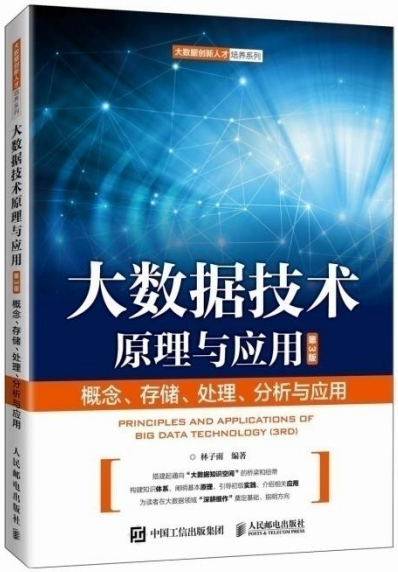
《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社　ISBN:978-7-115-54405-6　定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据 库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：
http://dblab.xmu.edu.cn/post/bigdata3

扫一扫访问教材官网



附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材

1+1黄金组合

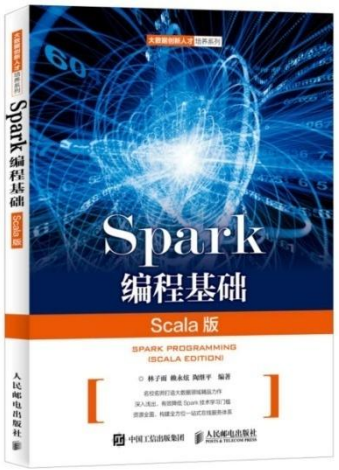
厦门大学林子雨编著

配套实验指导书

- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》
清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版

附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

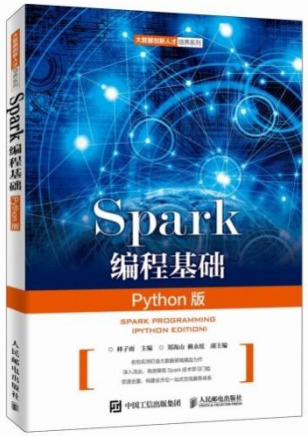
披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9

教材官网：http://dblab.xmu.edu.cn/post/spark/

本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。

附录I：《Spark编程基础（Python版）》



《Spark编程基础（Python版）》

厦门大学 林子雨，郑海山，赖永炫 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-52439-3

教材官网： http://dblab.xmu.edu.cn/post/spark-python/

本书以Python作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。

附录J：高校大数据课程公共服务平台

高校大数据课程

公 共 服 务 平 台

http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/

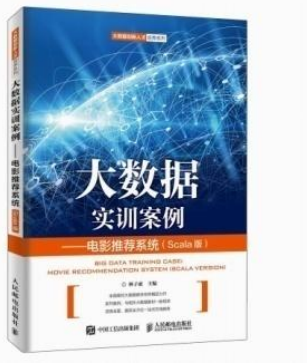
扫一扫访问平台主页

扫一扫观看3分钟FLASH动画宣传片

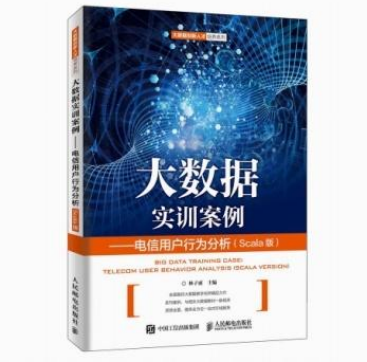
附录K：高校大数据实训课程系列案例教材

为了更好满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

《电影推荐系统》（已经于2019年5月出版）
《电信用户行为分析》（已经于2019年5月出版）
《实时日志流处理分析》
《微博用户情感分析》
《互联网广告预测分析》
《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！
http://dblab.xmu.edu.cn/post/shixunkecheng/



扫一扫访问大数据实训课程系列案例教材主页


Thank You!