Pandas tutorial

Pandas 是 Python 语言下的一个用于数据分析的工具类库。使用 Pandas 可以方便的对数据进行处理和分析。

1. Data Structures

Pandas 处理数据靠的是两个核心数据结构，Series 和 DataFrame，将会贯穿于整个数据分析过程。

Series 用来处理一维的序列数据，而 DataFrame 用来处理更复杂的多维数据。

1.1. Series

Series 是 Pandas 中用来处理一维数据的结构，有点类似于数组，但是增加了许多额外的特性。其数据结构如下所示：

index	value
0	12
1	-4
2	7
3	9

Series 包含两个数组，一个是存储的实际的值，另一个存储的是值的索引。Series 中存储的值可以是所有的 NumPy 中的数据结构。

1.1.1. 创建 Series

1.1.1.1. 从 list 中创建

```
1.  import numpy as np
2.  import pandas as pd
```

```
3.
4.  s = pd.Series([12, -4, 7, 9])
5.  print(s)
```

打印结果：

```
1.  0    12
2.  1    -4
3.  2     7
4.  3     9
5.  dtype: int64
```

1.1.1.2. 创建指定索引的 Series

```
1.  import numpy as np
2.  import pandas as pd
3.
4.  s = pd.Series([12, -4, 7, 9], index = ['a', 'b', 'c', 'd'])
5.  print(s)
```

打印结果：

```
1.  a    12
2.  b    -4
3.  c     7
4.  d     9
5.  dtype: int64
```

1.1.1.3. 从 Numpy 数组创建

```
1.  arr = np.array([1, 2, 3, 4])
2.  ser = pd.Series(arr)
3.  print(ser)
```

打印结果：

```
1.  0    1
2.  1    2
3.  2    3
4.  3    4
5.  dtype: int32
```

需要注意的是，从 Numpy 中创建的 Series，只是引用，对 Series 中值操作的影响会直接反应到原始的 Numpy 中

```
print(arr)
ser[0] = 0
print(arr)
```

打印结果：

```
[1 2 3 4]
[0 2 3 4]
```

1.1.1.4. 从 dict 创建

```
dic = {'red': 2000, 'blue': 1000, 'yellow': 500, 'orange': 1000}
ser = pd.Series(dic)
print(ser)
```

打印结果：

```
red       2000
blue      1000
yellow     500
orange    1000
dtype: int64
```

1.1.2. 查看 Series

1.1.2.1. 访问元素&查看Series

使用指定标签创建的 Series 既可以用下标访问元素，也可以用标签访问元素：

```
print("s[1]: " + str(s[1]))
print("s['b']: " + str(s['b']))
```

打印结果：

```
s[1]: -4
s['b']: -4
```

另外还可以直接查看 Series 的索引和值：

```
print("s.values: " + str(s.values))
print("s.index: " + str(s.index))
```

打印结果：

```
s.values: [12 -4  7  9]
s.index: Index(['a', 'b', 'c', 'd'], dtype='object')
```

1.1.2.2. 选取值

从 Series 中选取值与 NumPy 中类似，可以直接使用切片的方式选取。

```
print(s[0:2])
```

打印结果：

```
a    12
b    -4
dtype: int64
```

另外 Series 还支持使用标签的形式来选取对应的值：

```
print(s[['b', 'd']])
```

注意，这里的标签是一个数组，打印结果：

```
b   -4
d    9
dtype: int64
```

使用表达式选择值：

```
print(s[s > 8])
```

打印结果：

```
a    12
d     9
dtype: int64
```

1.1.2.3. 赋值

可以直接使用标签或索引，类似于数组进行赋值。

```
s['b'] = 1
print(s)
```

打印结果：

```
a    12
b     1
c     7
d     9
dtype: int64
```

1.1.3. 数学运算

类似于 Numpy 中对数学运算的支持，可以使用 Series 直接与数值进行加减乘除。

1.1.4. 常用操作

1.1.4.1. 去重

```
ser = pd.Series([1, 0, 2, 1, 2, 3])
print(ser.unique())
```

打印结果：

```
[1 0 2 3]
```

1.1.4.2. 统计

```
ser = pd.Series([1, 0, 2, 1, 2, 3])
print(ser.value_counts())
```

打印结果：

```
2    2
1    2
3    1
0    1
dtype: int64
```

其中第一列表示的是 Series 中的值，第二列表示的是在 Series 中出现的次数。

1.1.4.3. 是否存在

```
ser = pd.Series([1, 0, 2, 1, 2, 3])
print(ser.isin([0, 3]))
```

打印结果：

```
0    False
1     True
2    False
3    False
4    False
5     True
dtype: bool
```

直接将结果返回回来：

```
print(ser[ser.isin([0, 3])])
```

打印结果：

```
1    0
```

```
5    3
dtype: int64
```

1.1.4.4. 空值

在 Pandas 中使用 Numpy 中的 NaN 表示空值。可以使用 `isnull()` 和 `notnull()` 方法筛选结果。

```
ser = pd.Series([5, -3, np.NaN, 15])
print(ser)

print(ser.isnull())
```

打印结果：

```
0     5.0
1    -3.0
2     NaN
3    15.0
dtype: float64

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

1.2. DataFrame

DataFrame 是一种类似于表格的结构，用于处理多维数据。

index	color	object	price
0	blue	ball	1.2
1	green	pen	1.0
2	yellow	pencil	0.5
3	red	paper	0.8

index	color	object	price
4	white	mug	1.5

不同于 Series，DataFrame 有两列索引，第一个索引是行索引，每个索引关联一行的数据；第二个索引包含的是一系列的标签，关联的是每个特定的列。

我们一般把行索引称为索引（index），把列索引称为标签（label）。

1.2.1. 创建 DataFrame

1.2.1.1. 从字典中创建

```
import numpy as np
import pandas as pd

myDict = {
    'color': ['blue', 'green', 'yellow', 'red', 'white'],
    'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
    'price': [1.2, 1.0, 0.5, 0.8, 1.5]
}

df = pd.DataFrame(myDict)
print(df)
```

打印结果：

```
    color  object  price
0    blue    ball    1.2
1   green     pen    1.0
2  yellow  pencil    0.5
3     red   paper    0.8
4   white     mug    1.5
```

在创建时还可以指定需要的列，以及指定索引。

```
df = pd.DataFrame(myDict, columns=['object','price'],
    index=['one','two','three','four','five'])
print(df)
```

打印结果：

```
       object  price
one      ball    1.2
two       pen    1.0
three  pencil    0.5
four    paper    0.8
five      mug    1.5
```

1.2.1.2. 从 Numpy 数组中创建

```
arr = np.arange(16).reshape((4, 4))
df = pd.DataFrame(arr, columns=['colA','colB', 'colC', 'colD'])
print(df)
```

打印结果：

```
   colA  colB  colC  colD
0     0     1     2     3
1     4     5     6     7
2     8     9    10    11
3    12    13    14    15
```

1.2.2. 查看 DataFrame

1.2.2.1. 基本信息

查看索引：

```
print(df.index)
```

打印结果：

```
RangeIndex(start=0, stop=4, step=1)
```

查看标签：

```
print(df.columns)
```

打印结果：

```
Index(['colA', 'colB', 'colC', 'colD'], dtype='object')
```

查看值：

```
df.values
```

1.2.2.2. 选择列

使用标签选择对应列的值：

```
myDict = {
    'color': ['blue', 'green', 'yellow', 'red', 'white'],
    'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
    'price': [1.2, 1.0, 0.5, 0.8, 1.5]
}

df = pd.DataFrame(myDict)
print(df[["object", "price"]])
```

打印结果：

```
   object  price
0    ball    1.2
1     pen    1.0
2  pencil    0.5
3   paper    0.8
4     mug    1.5
```

1.2.2.3. 选择行

使用索引选择对应行的记录。其中 loc[] 中接受的可以是一个索引值，也可以是一个索引数组。

```
myDict = {
    'color': ['blue', 'green', 'yellow', 'red', 'white'],
    'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
    'price': [1.2, 1.0, 0.5, 0.8, 1.5]
}
```

```
6.
7.  df = pd.DataFrame(myDict)
8.  print(df.loc[2])
```

打印结果：

```
1.  color     yellow
2.  object    pencil
3.  price        0.5
4.  Name: 2, dtype: object
```

1.2.2.4. 切片

DataFrame 也支持切片，不过只支持行切片。

```
1.  myDict = {
2.      'color': ['blue', 'green', 'yellow', 'red', 'white'],
3.      'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
4.      'price': [1.2, 1.0, 0.5, 0.8, 1.5]
5.  }
6.
7.  df = pd.DataFrame(myDict)
8.  print(df[1:3])
```

打印结果：

```
1.      color  object  price
2.  1   green     pen    1.0
3.  2  yellow  pencil    0.5
```

1.2.2.5. 过滤

```
1.  myDict = {
2.      'color': ['blue', 'green', 'yellow', 'red', 'white'],
3.      'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
4.      'price': [1.2, 1.0, 0.5, 0.8, 1.5]
5.  }
6.
7.  df = pd.DataFrame(myDict)
8.  print(df[df['price'] > 1])
```

打印结果：

```
   color object  price
0   blue   ball    1.2
4  white    mug    1.5
```

1.2.3. 修改 DataFrame

1.2.3.1. 增加新列

```
myDict = {
    'color': ['blue', 'green', 'yellow', 'red', 'white'],
    'object': ['ball', 'pen', 'pencil', 'paper', 'mug'],
    'price': [1.2, 1.0, 0.5, 0.8, 1.5]
}

df = pd.DataFrame(myDict)
df['new'] = 12
print(df)
```

打印结果：

```
    color  object  price  new
0    blue    ball    1.2   12
1   green     pen    1.0   12
2  yellow  pencil    0.5   12
3     red   paper    0.8   12
4   white     mug    1.5   12
```

也可以使用一个数组作为新的列：

```
df['new'] = [3, 4, 5, 6, 7]
print(df)
```

打印结果：

```
    color  object  price  new
0    blue    ball    1.2    3
1   green     pen    1.0    4
```

```
2  yellow  pencil    0.5    5
3     red   paper    0.8    6
4   white     mug    1.5    7
```

这里的数组还可以换成 Series。

1.2.3.2. 删除新列

```
del df['price']
print(df)
```

打印结果：

```
    color  object  new
0    blue    ball    3
1   green     pen    4
2  yellow  pencil    5
3     red   paper    6
4   white     mug    7
```

Index Object

Index Object 是 Pandas 中表示索引的对象，在前面的 Series 和 DataFrame 中实际上已经接触过这个对象了。

一般不会涉及到这个对象的操作，知道有这么个东西即可。

2. 数据操作

2.1. Lambda 表达式

首先创建一个三行四列的 DataFrame：

```
arr = np.arange(12).reshape((3, 4)) ** 2
df = pd.DataFrame(arr)
```

```
print(df)
```

打印结果：

```
    0   1    2    3
0   0   1    4    9
1  16  25   36   49
2  64  81  100  121
```

然后利用 lambda 表达式对其中的每个元素开平方：

```
df = df.apply(lambda x: np.sqrt(x))
print(df)
```

打印结果：

```
     0    1     2     3
0  0.0  1.0   2.0   3.0
1  4.0  5.0   6.0   7.0
2  8.0  9.0  10.0  11.0
```

2.2. 统计信息

sum()

```
arr = np.arange(12).reshape((3, 4))
df = pd.DataFrame(arr, columns=['colA','colB', 'colC', 'colD'])
print(df.sum())
```

打印结果：

```
colA    12
colB    15
colC    18
colD    21
dtype: int64
```

mean()

```
arr = np.arange(12).reshape((3, 4))
df = pd.DataFrame(arr, columns=['colA','colB', 'colC', 'colD'])
print(df.mean())
```

打印结果：

```
colA    4.0
colB    5.0
colC    6.0
colD    7.0
dtype: float64
```

describe()

```
arr = np.arange(12).reshape((3, 4))
df = pd.DataFrame(arr, columns=['colA','colB', 'colC', 'colD'])
print(df.describe())
```

打印结果：

```
        colA  colB  colC  colD
count   3.0   3.0   3.0   3.0
mean    4.0   5.0   6.0   7.0
std     4.0   4.0   4.0   4.0
min     0.0   1.0   2.0   3.0
25%     2.0   3.0   4.0   5.0
50%     4.0   5.0   6.0   7.0
75%     6.0   7.0   8.0   9.0
max     8.0   9.0  10.0  11.0
```

2.3. 排序

2.3.1. Series

```
ser = pd.Series([1, 5, 2, 8, 3], index=['one','two','three','four','fiv
```

```
e'])
print(ser.sort_index())

print(ser.sort_values())
```

打印结果：

```
# sort by index
five     3
four     8
one      1
three    2
two      5
dtype: int64

# sort by values
one      1
three    2
five     3
two      5
four     8
dtype: int64
```

2.3.2. DataFrame

根据 colA 列的值进行倒序排序。

```
arr = np.arange(12).reshape((3, 4))
df = pd.DataFrame(arr, columns=['colA','colB', 'colC', 'colD'])
print(df.sort_values(by=['colA'], ascending=False))
```

打印结果：

```
   colA  colB  colC  colD
2     8     9    10    11
1     4     5     6     7
0     0     1     2     3
```

3. 读写数据

3.1. CSV

读取CSV

```
df = pd.read_csv("./somefile.csv")
```

写入CSV

```
pd.to_csv("./somefile.csv")
```

4. 数据分析

4.1. 数据准备

4.1.1. Merging

根据键进行拼接，有点类似于 SQL 语句中的 join 连接操作。

4.1.1.1. 简单连接

```
import numpy as np
import pandas as pd

myDict1 = {
    'id':['ball','pencil','pen','mug','ashtray'],
    'price': [12.33,11.44,33.21,13.23,33.62]
}

myDict2 = {
    'id':['pencil','pencil','ball','pen'],
    'color': ['white','red','red','black']
}
```

```
13.
14.  df1 = pd.DataFrame(myDict1)
15.  df2 = pd.DataFrame(myDict2)
16.
17.  print(pd.merge(df1, df2))
```

打印结果：

```
1.         id  price  color
2.  0    ball  12.33    red
3.  1  pencil  11.44  white
4.  2  pencil  11.44    red
5.  3     pen  33.21  black
```

这两个 DataFrame 通过 ID 这一列进行连接。

4.1.1.2. 使用 on 指定 key 字段

```
1.  myDict1 = {
2.      'id':['ball','pencil','pen','mug','ashtray'],
3.      'color': ['white','red','red','black','green'],
4.      'brand': ['OMG','ABC','ABC','POD','POD']
5.  }
6.
7.  myDict2 = {
8.      'id':['pencil','pencil','ball','pen'],
9.      'brand': ['OMG','POD','ABC','POD']
10. }
11.
12. df1 = pd.DataFrame(myDict1)
13. df2 = pd.DataFrame(myDict2)
14.
15. print(pd.merge(df1, df2, on = 'id'))
```

打印结果：

```
1.         id  color brand_x brand_y
2.  0    ball  white     OMG     ABC
3.  1  pencil    red     ABC     OMG
4.  2  pencil    red     ABC     POD
5.  3     pen    red     ABC     POD
```

4.1.1.3. 指定不同字段连接

以上几个实例中连接时是假设两个 DataFrame 作为 key 的字段的列名是相同的，很多时候可能会存在不同的情况，那么应该如何处理？

```
myDict1 = {
    'id':['ball','pencil','pen','mug','ashtray'],
    'color': ['white','red','red','black','green'],
    'brand': ['OMG','ABC','ABC','POD','POD']
}

myDict2 = {
    'sid':['pencil','pencil','ball','pen'],
    'brand': ['OMG','POD','ABC','POD']
}

df1 = pd.DataFrame(myDict1)
df2 = pd.DataFrame(myDict2)

print(pd.merge(df1, df2, left_on = 'id', right_on = 'sid'))
```

打印结果：

```
       id  color brand_x brand_y
0    ball  white     OMG     ABC
1  pencil    red     ABC     OMG
2  pencil    red     ABC     POD
3     pen    red     ABC     POD
```

4.1.1.4. 连接方式

在 SQL 中进行表的连接时，有几种连接方式，包括 inner join, outer join, left join and right join. 在 Pandas 中默认是内连接，也可以通过指定参数采用不同的连接方式。

```
myDict1 = {
    'id':['ball','pencil','pen','mug','ashtray'],
    'color': ['white','red','red','black','green'],
    'brand': ['OMG','ABC','ABC','POD','POD']
}

myDict2 = {
    'id':['pencil','pencil','ball','pen'],
```

```
    'brand': ['OMG','POD','ABC','POD']
}

df1 = pd.DataFrame(myDict1)
df2 = pd.DataFrame(myDict2)

print(pd.merge(df1, df2, on='id'))  # 默认内连接
print(pd.merge(df1, df2, on='id', how='outer'))
print(pd.merge(df1, df2, on='id', how='left'))
print(pd.merge(df1, df2, on='id', how='right'))
```

打印结果：

```
# 内连接：两边都有值的记录进行连接
       id  color brand_x brand_y
0    ball  white     OMG     ABC
1  pencil    red     ABC     OMG
2  pencil    red     ABC     POD
3     pen    red     ABC     POD

# 外连接：将所有的记录都进行连接
        id  color brand_x brand_y
0     ball  white     OMG     ABC
1   pencil    red     ABC     OMG
2   pencil    red     ABC     POD
3      pen    red     ABC     POD
4      mug  black     POD     NaN
5  ashtray  green     POD     NaN

# 左连接：以第一张表为主进行连接
        id  color brand_x brand_y
0     ball  white     OMG     ABC
1   pencil    red     ABC     OMG
2   pencil    red     ABC     POD
3      pen    red     ABC     POD
4      mug  black     POD     NaN
5  ashtray  green     POD     NaN

# 右连接：以第二张表为主进行连接
       id  color brand_x brand_y
0    ball  white     OMG     ABC
1  pencil    red     ABC     OMG
2  pencil    red     ABC     POD
3     pen    red     ABC     POD
```

4.1.1.5. 基于索引连接

Pandas 还提供了一个 `join()` 方法，用于基于索引的连接。

4.1.2. Concatenating

将多个表拼接成一张表。

```
df1 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[1,2,3], columns=['A','B','C'])
df2 = pd.DataFrame(np.random.rand(9).reshape(3,3), index=[4,5,6], columns=['A','B','C'])

print(pd.concat([df1, df2]))
```

打印结果：

```
          A         B         C
1  0.110074  0.360846  0.817533
2  0.464365  0.804425  0.372668
3  0.288729  0.783474  0.649644
4  0.941699  0.804263  0.493243
5  0.292398  0.573224  0.307815
6  0.447318  0.134989  0.667292
```

4.1.3. Combining

如果碰到需要合并的两个 DataFrame 中有重复的索引存在，而又不希望被第二个 DataFrame 覆盖相同索引的记录时，可以使用这个方法合并。

```
ser1 = pd.Series(np.random.rand(5),index=[1,2,3,4,5])
ser2 = pd.Series(np.random.rand(4),index=[2,4,5,6])

print(ser1)
print(ser2)
print(ser1.combine_first(ser2))
```

打印结果：

```
1    0.515438
2    0.371625
3    0.927717
4    0.321709
5    0.813122
dtype: float64

2    0.564350
4    0.718179
5    0.741726
6    0.108383
dtype: float64

1    0.515438
2    0.371625
3    0.927717
4    0.321709
5    0.813122
6    0.108383
dtype: float64
```

4.1.3. 行转列

在处理独热编码时，会碰到这样的需求，直接看例子。

```
myDict = {
    'color':['white','white','white', 'red','red','red', 'black','black','black'],
    'item':['ball','pen','mug', 'ball','pen','mug', 'ball','pen','mug'],
    'value': np.random.rand(9)
}
df = pd.DataFrame(myDict)
print(df)

pdf = df.pivot('color', 'item')
print(pdf)
```

打印结果：

```
   color  item     value
```

```
0  white  ball  0.583582
1  white   pen  0.413616
2  white   mug  0.389276
3    red  ball  0.541863
4    red   pen  0.413578
5    red   mug  0.942618
6  black  ball  0.850345
7  black   pen  0.230536
8  black   mug  0.612583

          value
item       ball       mug       pen
color
black  0.850345  0.612583  0.230536
red    0.541863  0.942618  0.413578
white  0.583582  0.389276  0.413616
```

4.2. 数据转换

4.2.1. 去重

```
myDict = {
    'color': ['white','white','red','red','white'],
    'value': [2,1,3,3,2]
}

df = pd.DataFrame(myDict)
print(df.duplicated())

print(df[df.duplicated()])
```

打印结果：

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

```
    color  value
3     red      3
4   white      2
```

这里需要注意的是，False 表示没有重复，那么后面打印出来的两个就是重复的值，而不是去重后的结果。

4.2.2. map

map 或者说是字典，利用字典可以进行数据替换、添加等操作。

4.2.2.1. 基于 map 的替换

```
myDict = {
    'item':['ball','mug','pen','pencil','ashtray'],
    'color':['white','rosso','verde','black','yellow'],
    'price':[5.56,4.20,1.30,0.56,2.75]
}

df = pd.DataFrame(myDict)

newColor = {
    'rosso': 'red',
    'verde': 'green'
}

print(df)

print(df.replace(newColor))
```

打印结果：

```
       item   color  price
0     ball   white   5.56
1      mug   rosso   4.20
2      pen   verde   1.30
3   pencil   black   0.56
4  ashtray  yellow   2.75

       item   color  price
0     ball   white   5.56
```

```
1       mug     red   4.20
2       pen   green   1.30
3    pencil   black   0.56
4  ashtray  yellow   2.75
```

4.2.2.2. 基于 map 的添加操作

```
myDict = {
    'item':['ball','mug','pen','pencil','ashtray'],
    'color':['white','rosso','verde','black','yellow']
}

df = pd.DataFrame(myDict)

prices = {
    'ball' : 5.56,
    'mug' : 4.20,
    'bottle' : 1.30,
    'scissors' : 3.41,
    'pen' : 1.30,
    'pencil' : 0.56,
    'ashtray' : 2.75
}

print(df)

df['price'] = df['item'].map(prices)
print(df)
```

打印结果：

```
      item   color
0     ball   white
1      mug   rosso
2      pen   verde
3   pencil   black
4  ashtray  yellow

      item   color  price
0     ball   white   5.56
1      mug   rosso   4.20
2      pen   verde   1.30
3   pencil   black   0.56
4  ashtray  yellow   2.75
```

4.2.3. 离散化

有时候在处理一个连续型值的列时，需要将值划分成几个区间，然后转换成离散型变量。

4.2.3.1. 指定区间切分

```
results = [12, 34, 67, 55, 28, 90, 99, 12, 3, 56, 74, 44, 87, 23, 49, 89, 87]
bins = [0, 25, 50, 75, 100]

cat = pd.cut(results, bins)

print(cat)
```

打印结果：

```
[(0, 25], (25, 50], (50, 75], (50, 75], (25, 50], ..., (75, 100], (0, 25], (25, 50], (75, 100], (75, 100]]
Length: 17
Categories (4, interval[int64]): [(0, 25] < (25, 50] < (50, 75] < (75, 100]]
```

这里有三行结果，分别来看每行的含义：
- 第一行是转换后的结果，即每个值被划分到哪个区间，这里就被替换成这个区间了
- 第二行是总数，即处理的数据的条数
- 第三行是划分的区间，这里一共被划分成四个区间

查看所划分的分类：

```
print(cat.categories)
```

打印结果：

```
IntervalIndex([(0, 25], (25, 50], (50, 75], (75, 100]]
              closed='right',
              dtype='interval[int64]')
```

查看划分结果对应区间的代码：

```
print(cat.codes)
```

打印结果：

```
[0 1 2 2 1 3 3 0 0 2 2 1 3 0 1 3 3]
```

统计划分后的结果：

```
print(pd.value_counts(cat))
```

打印结果：

```
(75, 100]    5
(50, 75]     4
(25, 50]     4
(0, 25]      4
dtype: int64
```

可以对区间设置名称：

```
results = [12, 34, 67, 55, 28, 90, 99, 12, 3, 56, 74, 44, 87, 23, 49, 89, 87]
bins = [0, 25, 50, 75, 100]
bin_names = ['unlikely','less likely','likely','highly likely']

cat = pd.cut(results, bins, labels = bin_names)

print(cat)
```

打印结果：

```
[unlikely, less likely, likely, likely, less likely, ..., highly likely, unlikely, less likely, highly likely, highly likely]
Length: 17
Categories (4, object): [unlikely < less likely < likely < highly likely]
```

4.2.3.2. 指定区间数量等距切分

根据指定区间的数量，以及该列值的最大值和最小值，切割成若干等分，然后进行划分。

```
results = [12, 34, 67, 55, 28, 90, 99, 12, 3, 56, 74, 44, 87, 23, 49, 89, 87]
bin_names = ['unlikely','less likely','likely','highly likely']

cat = pd.cut(results, 4, labels = bin_names)

print(cat)
```

打印结果：

```
[unlikely, less likely, likely, likely, less likely, ..., highly likely, unlikely, less likely, highly likely, highly likely]
Length: 17
Categories (4, object): [unlikely < less likely < likely < highly likely]
```

4.2.3.3. 指定区间数量根据密度划分

指定区间数量，根据数据散布的特点，确保每个区间最终划分的值的数量相同，至于区间边界，不一定是等分的。

```
results = [12, 34, 67, 55, 28, 90, 99, 12, 3, 56, 74, 44, 87, 23, 49, 89, 87, 20]
bin_names = ['unlikely','likely','highly likely']

cat1 = pd.cut(results, 3, labels = bin_names)
cat2 = pd.qcut(results, 3, labels = bin_names)

print(pd.value_counts(cat1))
print(pd.value_counts(cat2))
```

打印结果：

```
# pd.cut()
unlikely         7
highly likely    6
likely           5
```

```
dtype: int64

# pd.qcut()
highly likely    6
likely           6
unlikely         6
dtype: int64
```

4.2.4. 随机采样

```
myDict = {
    'item':['ball','mug','pen','pencil','ashtray'],
    'color':['white','rosso','verde','black','yellow'],
    'price':[5.56,4.20,1.30,0.56,2.75]
}

df = pd.DataFrame(myDict)
sample = np.random.randint(0, len(df), size=3)
print(df.take(sample))
```

打印结果：

```
     item   color  price
2     pen   verde   1.30
0    ball   white   5.56
3  pencil   black   0.56
```

4.3. 数据聚合

数据聚合是数据分析中很重要的一部分内容，比如常见的求和、求平均等都是属于聚合。聚合中很重要的一部分就是分组处理。

我们可以把分组处理拆分为三个阶段：
- 拆分 —— 将数据集拆分成不同的分组
- 处理 —— 对每个分组分别进行处理
- 合并 —— 将每个分组处理的结果合并成最终的结果

给定一个 DataFrame：

```
myDict = {
    'color': ['white','red','green','red','green'],
    'object': ['pen','pencil','pencil','ashtray','pen'],
    'price1': [5.56,4.20,1.30,0.56,2.75],
    'price2': [4.75,4.12,1.60,0.75,3.15]
}

df = pd.DataFrame(myDict)
print(df)
```

打印结果：

```
   color   object  price1  price2
0  white      pen    5.56    4.75
1    red   pencil    4.20    4.12
2  green   pencil    1.30    1.60
3    red  ashtray    0.56    0.75
4  green      pen    2.75    3.15
```

现在要求按照颜色一列统计价格1的平均值。

首先按照颜色进行分组：

```
group = df['price1'].groupby(df['color'])
print(group.groups)
```

打印结果：

```
{
    'green': Int64Index([2, 4], dtype='int64'),
    'red': Int64Index([1, 3], dtype='int64'),
    'white': Int64Index([0], dtype='int64')
}
```

然后计算每组的平均值：

```
print(group.mean())
```

打印结果：

```
color
green    2.025
red      2.380
white    5.560
Name: price1, dtype: float64
```