Pandas

Pandas是一个开源的数据处理与分析的工具。可以与`numpy`进行联合使用。

Pandas中主要有两种数据结构：DataFrame和Series

Series： 类似于一维数组或列表，是由一组数据以及与之相关的数据标签（索引）构成。Series 可以看作是 DataFrame 中的一列，也可以是单独存在的一维数据结构。

DataFrame：可视为由多个 Series 组成的数据结构

一、Series

Series是一个序列，类似于表格中的一个列，也可以理解为一个一维数组，可以保存任何数据类型。

创建Series

使用列表进行创建

```
pandas.Series( data, index, dtype, name, copy)
```

> **参数说明：**
>
> - data：一组数据(ndarray 类型)。
> - index：数据索引标签，如果不指定，默认从 0 开始。
> - dtype：数据类型，默认会自己判断。
> - name：设置名称。
> - copy：拷贝数据，默认为 False。

```
import pandas as pd

a = [1, 2, 3]

myvar = pd.Series(a)

print(myvar)
'''
输出：
0    1
1    2
2    3
dtype: int64
'''
```

默认的索引是从0开始的，我们可以使用中括号的方式来指定元素`[]`

使用字典进行创建

```
sites = {1: "Google", 2: "Runoob", 3: "Wiki"}

myvar = pd.Series(sites)

print(myvar)
'''
输出：
1    Google
2    Runoob
3      Wiki
dtype: object
'''
```

如果我们只需字典中的一部分数据，只需要指定需要数据的索引即可。

```
sites = {1: "Google", 2: "Runoob", 3: "Wiki"}

myvar = pd.Series(sites, index = [1, 2])

print(myvar)

'''
输出：
1    Google
2    Runoob
dtype: object
'''
```

指定索引

我们可以自己指定索引

```
a = ["Google", "Runoob", "Wiki"]

myvar = pd.Series(a, index = ["x", "y", "z"])

print(myvar)
'''
输出:
x    Google
y    Runoob
z      Wiki
dtype: object
'''
```

设置 Series 名称参数:

```
import pandas as pd

sites = {1: "Google", 2: "Runoob", 3: "Wiki"}

myvar = pd.Series(sites, index = [1, 2], name="RUNOOB-Series-TEST" )

print(myvar)

'''
输出:
1    Google
2    Runoob
Name: RUNOOB-Series-TEST, dtype: object
'''
```

切片操作

```
var1 = myvar[0:1]
print(var1)
'''
输出:
1    Google
Name: RUNOOB-Series-TEST, dtype: object
'''
```

获取索引或值

```
index = myvar.index
values = myvar.values
print(index)
print(values)
'''
输出:
Index([1, 2], dtype='int64')
['Google' 'Runoob']
'''
```

获取统计信息

```
myvar1 = pd.Series([1, 2, 3, 4, 5, 6, 7, 8, 9])
print(myvar1.describe())

'''
输出:
count    9.000000
mean     5.000000
std      2.738613
min      1.000000
25%      3.000000
50%      5.000000
75%      7.000000
max      9.000000
dtype: float64
'''
```

二、DataFrame

`DataFrame` 是一个表格型的数据结构。

创建DataFrame

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

> 参数说明:
>
> - data：一组数据`(ndarray、series, map, lists, dict 等类型)`。
> - index：索引值，或者可以称为行标签。
> - columns：列标签，默认为 `RangeIndex (0, 1, 2, …, n)` 。
> - dtype：数据类型。

> - copy：拷贝数据，默认为 `False`。

使用列表进行创建

```
data = [['Google', 10], ['Runoob', 12], ['Wiki', 13]]

# 创建DataFrame
df = pd.DataFrame(data, columns=['Site', 'Age'])

# 使用astype方法设置每列的数据类型
df['Site'] = df['Site'].astype(str)
df['Age'] = df['Age'].astype(float)

print(df)
'''
输出：
     Site   Age
0  Google  10.0
1  Runoob  12.0
2    Wiki  13.0
'''
```

使用字典进行创建

```
data = {'Site':['Google', 'Runoob', 'Wiki'], 'Age':[10, 12, 13]}

df = pd.DataFrame(data)

print (df)
'''
输出：
     Site  Age
0  Google   10
1  Runoob   12
2    Wiki   13
'''
```

```
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

df = pd.DataFrame(data)

print (df)
'''
输出:
   a   b     c
0  1   2   NaN
1  5  10  20.0
'''
```

使用ndarray进行创建

```
import numpy as np
# 创建一个包含网站和年龄的二维ndarray
ndarray_data = np.array([
    ['Google', 10],
    ['Runoob', 12],
    ['Wiki', 13]
])

# 使用DataFrame构造函数创建数据帧
df = pd.DataFrame(ndarray_data, columns=['Site', 'Age'])

# 打印数据帧
print(df)

'''
输出:
     Site Age
0  Google  10
1  Runoob  12
2    Wiki  13
'''
```

找到指定行的数据

Pandas 可以使用loc属性返回指定行的数据，如果没有设置索引，第一行索引为 **0**，第二行索引为 **1**，以此类推。

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

# 数据载入到 DataFrame 对象
```

```
df = pd.DataFrame(data)

# 返回第一行
print(df.loc[0])
# 返回第二行
print(df.loc[1])
'''
输出：
calories    420
duration     50
Name: 0, dtype: int64
calories    380
duration     40
Name: 1, dtype: int64
'''
```

返回多行数据

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

# 数据载入到 DataFrame 对象
df = pd.DataFrame(data)

# 返回第一行和第二行
print(df.loc[[0, 1]])

'''
输出：
   calories  duration
0       420        50
1       380        40
'''
```

指定索引

```
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

print(df)
```

```
'''
输出：
      calories  duration
day1       420        50
day2       380        40
day3       390        45
'''
```

更多操作

```
# 获取列
name_column = df['Name']

# 获取行
first_row = df.loc[0]

# 选择多列
subset = df[['Name', 'Age']]

# 过滤行
filtered_rows = df[df['Age'] > 30]

# 获取列名
columns = df.columns

# 获取形状（行数和列数）
shape = df.shape

# 获取索引
index = df.index

# 获取描述统计信息
stats = df.describe()

# 添加新列
df['Salary'] = [50000, 60000, 70000]

# 删除列
df.drop('City', axis=1, inplace=True)

# 排序
df.sort_values(by='Age', ascending=False, inplace=True)

# 重命名列
df.rename(columns={'Name': 'Full Name'}, inplace=True)

# 从CSV文件创建 DataFrame
```

```
df_csv = pd.read_csv('example.csv')

# 从Excel文件创建 DataFrame
df_excel = pd.read_excel('example.xlsx')

# 从字典列表创建 DataFrame
data_list = [{'Name': 'Alice', 'Age': 25}, {'Name': 'Bob', 'Age': 30}]
df_from_list = pd.DataFrame(data_list)
```

三、数据的读取

CSV格式

读取文件

> 方法：`df = pd.read_csv(path)`
>
> 返回的是一个`DataFrame`格式的数据

如果需要展示完整的数据，需要使用`to_string()`来进行展示

```
df = pd.read_csv('./access/nba.csv')
print(df.to_string())  # 加上to_string() 会将文件以DataFrame的格式进行呈现
# print(df)  # 这样做只会显示前五行和后五行，其它行以...的形式进行省略
```

查看数据

`head()`函数提供了显示表格的前n行（默认为前5行）。

`tail()`函数提供了显示表格的最后n行（默认为后5行）。

`info()`函数提供了显示表格的具体信息。

```
df1 = pd.read_csv('./access/nba.csv')
print(df1.head(10))  # 读取前n行，默认为前5行

print(df1.tail(10))  # 读取后n行，默认为读取后5行

print(df1.info())
```

json格式

读取文件

与读取`csv`格式的数据类似，使用`pd.read_json(path)`进行读取

```
df2 = pd.read_json('./access/sites.json')
# print(df2)
print(df2.to_string())  # 以DataFrame格式显示完整的数据
```

从`url`中读取json数据

```
URL = 'https://static.jyshare.com/download/sites.json'
df = pd.read_json(URL)
print(df)
```

处理内嵌的json数据

对于下面这种json格式的数据，在读取的时候无法直接读取到嵌套的内容，会将嵌套的内容当作一个整体进行展示

```
{
    "school_name": "ABC primary school",
    "class": "Year 1",
    "students": [
    {
        "id": "A001",
        "name": "Tom",
        "math": 60,
        "physics": 66,
        "chemistry": 61
    },
    {
        "id": "A002",
        "name": "James",
        "math": 89,
        "physics": 76,
        "chemistry": 51
    },
    {
        "id": "A003",
        "name": "Jenny",
        "math": 79,
        "physics": 90,
        "chemistry": 78
    }]
}
```

```
df3 = pd.read_json('./access/nels.json')
print(df3.to_string())  # 由于有内嵌的数据，不能直接展示出来

'''
输出：
          school_name   class
                              students
0  ABC primary school  Year 1    {'id': 'A001', 'name': 'Tom', 'math':
60, 'physics': 66, 'chemistry': 61}
1  ABC primary school  Year 1  {'id': 'A002', 'name': 'James', 'math':
89, 'physics': 76, 'chemistry': 51}
2  ABC primary school  Year 1  {'id': 'A003', 'name': 'Jenny', 'math':
79, 'physics': 90, 'chemistry': 78}
'''
```

为此我么需要使用其它方法来进行处理，`json_normalize()`可以很方便的处理这种数据

```
# 使用json_normalize()函数将其完全显示出来
import json
with open('./access/nels.json') as f:
    data = json.loads(f.read())

# 展平数据
df_ne_ls = pd.json_normalize(data, record_path = ['students'])  # 表示
展开students下没有展开的数据，这会导致不包含原来展开的数据
print(df_ne_ls)

'''
输出：
     id   name  math  physics  chemistry
0  A001    Tom    60       66         61
1  A002  James    89       76         51
2  A003  Jenny    79       90         78
'''
```

如果还要包含原来已经展开的数据，我们需要使用一个参数：`meta`进行指定。meta的值是一个列表，如果可以表示某一个父级，也可以在嵌套一个列表表示某一个父级下的子级，如：

```
df = pd.json_normalize(
  data,
  record_path =['students'],
  meta=[
    'class',
    ['info', 'president'],    # 表示info下的president中的数据
    ['info', 'contacts', 'tel']   # 表示info下的contaces下的tel中的数据
  ]
)
```

```
# 包含原来展开的数据，需要使用meta参数来进行设置要展示的数据字段
# 使用 Python JSON 模块载入数据
with open('./access/nels.json','r') as f:
    data = json.loads(f.read())

# 展平数据
df_nested_list = pd.json_normalize(
    data,
    record_path =['students'],
    meta=['school_name', 'class']
)
print(df_nested_list)

'''
输出:
     id   name  math  physics  chemistry         school_name   class
0  A001    Tom    60       66         61  ABC primary school  Year 1
1  A002  James    89       76         51  ABC primary school  Year 1
2  A003  Jenny    79       90         78  ABC primary school  Year 1
'''
```

四、空值处理

清洗空值

如果我们要删除包含空字段的行，可以使用 **dropna()** 方法，语法格式如下：

```
DataFrame.dropna(axis=0, how='any', thresh=None, subset=None,
inplace=False)
```

> **参数说明：**
>
> - `axis`：默认为 **0**，表示逢空值剔除整行，如果设置参数 **axis = 1** 表示逢空值去掉整列。

> - `how`：默认为 **'any'** 如果一行（或一列）里任何一个数据有出现 `NA` 就去掉整行，如果设置 **how='all'** 一行（或列）都是 `NA`才去掉这整行。
> - `thresh`：设置需要多少非空值的数据才可以保留下来的。
> - `subset`：设置想要检查的列。如果是多个列，可以使用列名的 `list` 作为参数。
> - `inplace`：如果设置 `True`，将计算得到的值直接覆盖之前的值并返回 `None`，修改的是源数据。

判断单元格是否为空

使用 **isnull()** 判断各个单元格是否为空，如果为空，则标记为True，默认显示的`NaN`即为空数据。

```
df = pd.read_csv('./access/property-data.csv')

print (df['NUM_BEDROOMS'])
print (df['NUM_BEDROOMS'].isnull())  # 判断是否为空值，标记空数据的行为True

'''
输出：

0      3
1      3
2    NaN
3      1
4      3
5    NaN
6      2
7      1
8     na
Name: NUM_BEDROOMS, dtype: object
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7    False
8    False
Name: NUM_BEDROOMS, dtype: bool
'''
```

我们也可以指定某些数据是属于空值的，如上面第`8`行`na`这个数据，我们也可以理解为是空数据，也要进行标记，需要使用参数：`na_values`来指定某些值是属于空数据类型的。

```
missing_values = ["n/a", "na", "--"]
df = pd.read_csv('./access/property-data.csv', na_values =
missing_values)   # 直接指定了那些数据是始于空值的

print (df['NUM_BEDROOMS'])
print (df['NUM_BEDROOMS'].isnull())
'''
输出:
0    3.0
1    3.0
2    NaN
3    1.0
4    3.0
5    NaN
6    2.0
7    1.0
8    NaN
Name: NUM_BEDROOMS, dtype: float64
0    False
1    False
2     True
3    False
4    False
5     True
6    False
7    False
8     True
Name: NUM_BEDROOMS, dtype: bool
'''
```

删除空的数据行

函数：`dropna()`用来删除含有空数据的行，没有参数时，默认返回删除后的结果，原数据不发生改变。

```
df = pd.read_csv('./access/property-data.csv')

new_df = df.dropna()    # 这个函数可以返回将空数据的行进行删除后的数据，原来的数据不发生改变

print(new_df.to_string())

'''
输出：
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM            Y            3        1
1000
1  100002000.0   197.0   LEXINGTON            N            3      1.5
--
8  100009000.0   215.0     TREMONT            Y           na        2
1800
'''
```

如果要让原数据发生改变，需要添加参数`inplace = True`

```
df = pd.read_csv('./access/property-data.csv')
df.dropna(inplace = True)

print(df.to_string())

'''
输出：
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM            Y            3        1
1000
1  100002000.0   197.0   LEXINGTON            N            3      1.5
--
8  100009000.0   215.0     TREMONT            Y           na        2
1800
'''
```

指定某些字段为空的行进行删除

```
df = pd.read_csv('./access/property-data.csv')

df.dropna(subset=['ST_NUM'], inplace = True)  # 指定删除ST_NUM列中是空值的行

print(df.to_string())
'''
```

```
输出:
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM            Y            3        1
1000
1  100002000.0   197.0   LEXINGTON            N            3      1.5
--
3  100004000.0   201.0    BERKELEY           12            1      NaN
700
4          NaN   203.0    BERKELEY            Y            3        2
1600
5  100006000.0   207.0    BERKELEY            Y          NaN        1
800
7  100008000.0   213.0     TREMONT            Y            1        1
NaN
8  100009000.0   215.0     TREMONT            Y           na        2
1800
'''
```

替换空值

```
df = pd.read_csv('./access/property-data.csv')

df.fillna(12345, inplace = True)    # 使用fillna函数来替换原来的空值

print(df.to_string())
'''
输出:
           PID    ST_NUM      ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0     104.0       PUTNAM            Y            3        1
  1000
1  100002000.0     197.0    LEXINGTON            N            3      1.5
    --
2  100003000.0   12345.0    LEXINGTON            N        12345        1
   850
3  100004000.0     201.0     BERKELEY           12            1    12345
   700
4      12345.0     203.0     BERKELEY            Y            3        2
  1600
5  100006000.0     207.0     BERKELEY            Y        12345        1
   800
6  100007000.0   12345.0   WASHINGTON        12345            2   HURLEY
   950
7  100008000.0     213.0      TREMONT            Y            1        1
12345
```

```
8  100009000.0    215.0     TREMONT            Y           na         2
  1800
'''
```

替换指定列的空值

```
df = pd.read_csv('./access/property-data.csv')

df['PID'].fillna(12345, inplace = True)  # 只替换PID这一列的空值

print(df.to_string())
'''
输出 :
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM            Y            3         1
1000
1  100002000.0   197.0   LEXINGTON            N            3       1.5
   --
2  100003000.0     NaN   LEXINGTON            N          NaN         1
  850
3  100004000.0   201.0    BERKELEY           12            1       NaN
  700
4      12345.0   203.0    BERKELEY            Y            3         2
1600
5  100006000.0   207.0    BERKELEY            Y          NaN         1
  800
6  100007000.0     NaN  WASHINGTON          NaN            2   HURLEY
  950
7  100008000.0   213.0     TREMONT            Y            1         1
  NaN
8  100009000.0   215.0     TREMONT            Y           na         2
1800
'''
```

五、统计

均值

函数：`mean()`用于返回指定字段的均值

```
# mean()函数
df = pd.read_csv('./access/property-data.csv')

x = df["ST_NUM"].mean()  # 计算指定列的均值

df["ST_NUM"].fillna(x, inplace = True) # 将该列的空值全部替换为均值
```

```
print(df.to_string())

'''
输出:
           PID      ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS
NUM_BATH SQ_FT
0  100001000.0  104.000000      PUTNAM            Y            3
1  1000
1  100002000.0  197.000000   LEXINGTON            N            3
1.5    --
2  100003000.0  191.428571   LEXINGTON            N          NaN
1   850
3  100004000.0  201.000000    BERKELEY           12            1
NaN   700
4          NaN  203.000000    BERKELEY            Y            3
2  1600
5  100006000.0  207.000000    BERKELEY            Y          NaN
1   800
6  100007000.0  191.428571  WASHINGTON          NaN            2
HURLEY   950
7  100008000.0  213.000000     TREMONT            Y            1
1   NaN
8  100009000.0  215.000000     TREMONT            Y           na
2  1800
'''
```

中位数

函数：`median()`用于返回指定字段的中位数

```
# 可以使用median()方法来计算中位数
df = pd.read_csv('./access/property-data.csv')

x = df["ST_NUM"].median()

df["ST_NUM"].fillna(x, inplace = True)  # 将ST_NUM列中的控制替换为中位数

print(df.to_string())

'''
输出:
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM            Y            3        1
1000
1  100002000.0   197.0   LEXINGTON            N            3      1.5
   --
```

```
2  100003000.0    203.0    LEXINGTON             N           NaN         1
  850
3  100004000.0    201.0     BERKELEY            12             1       NaN
  700
4           NaN    203.0     BERKELEY             Y             3         2
1600
5  100006000.0    207.0     BERKELEY             Y           NaN         1
  800
6  100007000.0    203.0   WASHINGTON           NaN             2    HURLEY
  950
7  100008000.0    213.0      TREMONT             Y             1         1
  NaN
8  100009000.0    215.0      TREMONT             Y            na         2
1800
'''
```

众数

函数：`mode()`用于返回指定字段的众数

```
# 使用mode()方法来计算众数
df = pd.read_csv('./access/property-data.csv')

x = df["ST_NUM"].mode()

df["ST_NUM"].fillna(x, inplace = True)  # 将ST_NUM列的空值替换为改列的众数

print(df.to_string())

'''
输出:
           PID  ST_NUM     ST_NAME OWN_OCCUPIED NUM_BEDROOMS NUM_BATH
SQ_FT
0  100001000.0   104.0      PUTNAM             Y             3         1
1000
1  100002000.0   197.0   LEXINGTON             N             3       1.5
   --
2  100003000.0   201.0   LEXINGTON             N           NaN         1
  850
3  100004000.0   201.0     BERKELEY            12             1       NaN
  700
4           NaN   203.0     BERKELEY             Y             3         2
1600
5  100006000.0   207.0     BERKELEY             Y           NaN         1
  800
6  100007000.0   215.0   WASHINGTON           NaN             2    HURLEY
  950
```

```
7  100008000.0   213.0     TREMONT            Y            1        1
  NaN
8  100009000.0   215.0     TREMONT            Y           na        2
1800
'''
```

六、数据清洗

格式错误的数据

```
# 第三个日期格式错误
data = {
  "Date": ['2020/12/01', '2020/12/02' , '20201226'],
  "duration": [50, 40, 45]
}

df = pd.DataFrame(data, index = ["day1", "day2", "day3"])

df['Date'] = pd.to_datetime(df['Date'], format='mixed')  # 将该列的数据类型全部指定为日期格式的数据

print(df.to_string())
'''
输出:
           Date  duration
day1 2020-12-01        50
day2 2020-12-02        40
day3 2020-12-26        45
'''
```

数据错误的数据

直接使用`loc`索引到指定的坐标，然后进行更改。

```
person = {
  "name": ['Google', 'Runoob' , 'Taobao'],
  "age": [50, 40, 12345]    # 12345 年龄数据是错误的
}

df = pd.DataFrame(person)

df.loc[2, 'age'] = 30 # 修改数据， 将第2行age列的数据指定为30

print(df.to_string())
'''
输出:
     name  age
```

```
0  Google   50
1  Runoob   40
2  Taobao   30
'''
```

```
# 设置条件语句，设定数据
person = {
  "name": ['Google', 'Runoob' , 'Taobao'],
  "age": [50, 200, 12345]
}

df = pd.DataFrame(person)

for x in df.index:
  if df.loc[x, "age"] > 120:
    df.loc[x, "age"] = 120

print(df.to_string())
'''
输出:
     name  age
0  Google   50
1  Runoob  120
2  Taobao  120
'''
```

```
# 对指定的数据进行删除
person = {
  "name": ['Google', 'Runoob' , 'Taobao'],
  "age": [50, 40, 12345]    # 12345 年龄数据是错误的
}

df = pd.DataFrame(person)

for x in df.index:
  if df.loc[x, "age"] > 120:
    df.drop(x, inplace = True)  # 删除指定的数据行

print(df.to_string())
'''
输出:
     name  age
0  Google   50
1  Runoob   40
'''
```

清洗重复的数据

标记

函数：`duplicated()`用于标记重复的数据，第二次出现即为重复，返回一个`DataFrame`类型的数据，其中`True`表示这个数据已经重复了。

```
person = {
  "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
  "age": [50, 40, 40, 23]
}
df = pd.DataFrame(person)

print(df.duplicated())  # 返回的数据中，标记为True的则为重复的数据

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

删除

函数：`drop_duplicates(inplace = Ture)`用于删除重复的数据。

```
persons = {
  "name": ['Google', 'Runoob', 'Runoob', 'Taobao'],
  "age": [50, 40, 40, 23]
}

df = pd.DataFrame(persons)

df.drop_duplicates(inplace = True)  # 可以直接删除重复的数据
print(df)

'''
输出：
     name  age
0  Google   50
1  Runoob   40
3  Taobao   23
'''
```

七、后记

读取数据的函数

函数	说明
`pd.read_csv(filename)`	读取 CSV 文件；
`pd.read_excel(filename)`	读取 Excel 文件；
`pd.read_sql(query, connection_object)`	从 SQL 数据库读取数据；
`pd.read_json(json_string)`	从 JSON 字符串中读取数据；
`pd.read_html(url)`	从 HTML 页面中读取数据。

数据清洗

函数	说明
`df.dropna()`	删除包含缺失值的行或列；
`df.fillna(value)`	将缺失值替换为指定的值；
`df.replace(old_value, new_value)`	将指定值替换为新值；
`df.duplicated()`	检查是否有重复的数据；
`df.drop_duplicates()`	删除重复的数据。

数据选择和切片

函数	说明
`df[column_name]`	选择指定的列；
`df.loc[row_index, column_name]`	通过标签选择数据；
`df.iloc[row_index, column_index]`	通过位置选择数据；
`df.ix[row_index, column_name]`	通过标签或位置选择数据；
`df.filter(items=[column_name1, column_name2])`	选择指定的列；
`df.filter(regex='regex')`	选择列名匹配正则表达式的列；

函数	说明
`df.sample(n)`	随机选择 n 行数据。

数据排序

函数	说明
`df.sort_values(column_name)`	按照指定列的值排序；
`df.sort_values([column_name1, column_name2], ascending=[True, False])`	按照多个列的值排序；
`df.sort_index()`	按照索引排序。

数据分组和聚合

函数	说明
`df.groupby(column_name)`	按照指定列进行分组；
`df.aggregate(function_name)`	对分组后的数据进行聚合操作；
`df.pivot_table(values, index, columns, aggfunc)`	生成透视表。

数据合并

函数	说明
`pd.concat([df1, df2])`	将多个数据框按照行或列进行合并；
`pd.merge(df1, df2, on=column_name)`	按照指定列将两个数据框进行合并。

数据选择和过滤

函数	说明
`df.loc[row_indexer, column_indexer]`	按标签选择行和列。

函数	说明
`df.iloc[row_indexer, column_indexer]`	按位置选择行和列。
`df[df['column_name'] > value]`	选择列中满足条件的行。
`df.query('column_name > value')`	使用字符串表达式选择列中满足条件的行。

数据统计和描述

函数	说明
`df.describe()`	计算基本统计信息，如均值、标准差、最小值、最大值等。
`df.mean()`	计算每列的平均值。
`df.median()`	计算每列的中位数。
`df.mode()`	计算每列的众数。
`df.count()`	计算每列非缺失值的数量。