

使用以下语句导入 Pandas 库:

```
> import pandas as pd
```

1. Pandas 数据结构

Series - 序列

存储任意类型数据的一维数组

```
> s = pd.Series([3, -5, 7, 4], index=['a', 'b', 'c', 'd'])
```

DataFrame - 数据帧

```
> data = {'Country': ['Belgium', 'India', 'Brazil'],
          'Capital': ['Brussels', 'New Delhi', 'Brasília'],
          'Population': [11190846, 1303171035, 207847528]}
```

```
> df = pd.DataFrame(data, columns=['Country', 'Capital', 'Population'])
```

2. 输入 / 输出

读取 / 写入 CSV

```
> df.to_csv('myDataFrame.csv', index=False)
> pd.read_csv('myDataFrame.csv', nrows=5)
```

读取 / 写入 Excel

```
> pd.to_excel('myDataFrame.xlsx', index=False, sheet_name='Sheet1')
> pd.read_excel('myDataFrame.xlsx')
> xlsx = pd.ExcelFile('myDataFrame.xls') # 读取内含多个表的 Excel
> df = pd.read_excel(xlsx, 'Sheet1') # 读取多表 Excel 中的 Sheet1 表
```

读取和写入 SQL 查询及数据库表

```
> from sqlalchemy import create_engine
> engine = create_engine('sqlite:///memory:')
> pd.read_sql("SELECT * FROM my_table;", engine)
> pd.read_sql_table('my_table', engine)
> pd.read_sql_query("SELECT * FROM my_table;", engine)
```

read_sql() 是 read_sql_table() 与 read_sql_query() 的便捷打包器

```
> pd.to_sql('myDf', engine)
```

3. 筛选数据

取值

```
> s['b'] # 取序列的值
```

```
-5
```

```
> df[1:] # 取数据帧的子集
```

Country Capital Population

1 India New Delhi 1303171035

2 Brazil Brasília 207847528

选取、布尔索引及设置值

按位置

按行与列的位置选择某值

```
> df.iloc[[0], [0]]
```

```
'Belgium'
```

```
> df.iat[0, 0]
```

```
'Belgium'
```

按标签

按行与列的名称选择某值

```
> df.loc[[0], ['Country']]
```

```
'Belgium'
```

按行与列的名称选择某值

```
> df.at[0, 'Country']
```

```
'Belgium'
```

按标签 / 位置

选择某行

```
> df.loc[2]
```

```
Country Brazil
```

```
Capital Brasília
```

```
Population 207847528
```

选择某列

```
> df.loc[:, 'Capital']
```

```
0 Brussels
```

```
1 New Delhi
```

```
2 Brasília
```

按行列取值

```
> df.loc[1, 'Capital']
```

```
'New Delhi'
```

布尔索引

```
> s[~(s > 1)] # 序列 S 中没有大于 1 的值
```

```
> s[(s < -1) | (s > 2)] # 序列 S 中小于 -1 或大于 2 的值
```

选择数据帧中 Population 大于 12 亿的数据

```
> df[df['Population']>1200000000]
```

选择数据帧中人口大于 12 亿的数据 'Country' 和 'Capital' 字段

```
> df.loc[df['Population']>1200000000, ['Country', 'Capital']]
```

设置值

```
> s['a'] = 6 # 将序列 s 中索引为 a 的值设为 6
```



Pandas 是一个构建于 Numpy 之上的 Python 库，它提供了高性能的数据操作。Pandas 提供了大量能使我们快速便捷地处理数据的函数和方法，经常在数据科学中用于数据处理、数据变换、特征工程、数据探索分析与呈现等过程中。



4. 删除数据

通过 **drop** 函数删除数据

```
> s.drop(['a', 'c']) # 按索引删除序列的值 (axis=0)
> df.drop('Country', axis=1) # 按列名删除数据帧的列 (axis=1)
```

5. 排序和排名

根据索引或者值进行排序

```
> df.sort_index() # 按索引排序
> df.sort_values(by='Country') # 按某列的值排序
> df.rank() # 数据帧排名
```

6. 查询信息与计算

基本信息

```
> df.shape # (行, 列)
> df.index # 获取索引
> df.columns # 获取列名
> df.info() # 获取数据帧基本信息
> df.count() # 非 Na 值的数量
```

汇总

```
> df.sum() # 合计
> df.cumsum() # 累计
> df.min()/df.max() # 最小值除以最大值
> df.idxmin()/df.idxmax() # 索引最小值除以索引最大值
> df.describe() # 基础统计数据
> df.mean() # 平均值
> df.median() # 中位数
```

7. 应用函数

通过 **apply** 函数应用变换

```
> f = lambda x: x*2 # 应用匿名函数 lambda
> df.apply(f) # 应用函数
> df.applymap(f) # 对每个单元格应用函数
```

8. 数据对齐

内部数据对齐

如有不一致的索引，则使用 NA 值：

```
> s3 = pd.Series([7, -2, 3], index=['a', 'c', 'd'])
> s + s3
a    10.0
b     NaN
c     5.0
d     7.0
```

使用 **Fill** 方法运算

还可以使用 **Fill** 方法补齐缺失后再运算：

```
> s.add(s3, fill_value=0)
a    10.0
b    -5.0
c     5.0
d     7.0
> s.sub(s3, fill_value=2)
> s.div(s3, fill_value=4)
> s.mul(s3, fill_value=3)
```

```
> help(pd.Series.loc)
```

调用帮助

作者 | 韩信子 @ShowMeAI

设计 | 南乔 @ShowMeAI

参考 | datacamp cheatsheet



扫码回复“速查表”

下载最新全套资料