

## 1. Pandas 数据结构

### 1.1 Series – 序列

```
import pandas as pd # 导入 Pandas 库

# 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'])
```

### 1.2 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

### 2.1 读取 / 写入 CSV

```
df.to_csv('myDataFrame.csv', index=False)

pd.read_csv('myDataFrame.csv', nrows=5)
```

### 2.2 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
```

### 2.3 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

### 3.1 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
```

### 3.2 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

# 按行列表值

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

'New Delhi'

#### 布尔索引

# 序列 S 中没有大于 1 的值

```
s[~(s>1)]
```

# 序列 S 中小于 -1 或大于 2 的值

```
s[(s<-1)|(s>2)]
```

# 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

# 将序列 s 中索引为 a 的值设为 6

```
s['a'] = 6
```

Show Me AI

## 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
```

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。



Pandas 速查表

获取最新版 | <http://www.showmeai.tech/>

作者 | 韩信子 @ShowMeAI

设计 | 南乔 @ShowMeAI

扫码回复“**工控科学**”参考 | DataCamp Cheatsheet

下载**最新**全套速查表

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

Show Me AI

## 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:

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)
s.sub(s3, fill_value=2)
s.div(s3, fill_value=4)
s.mul(s3, fill_value=3)
```

a	10.0
b	-5.0
c	5.0
d	7.0

## 9. 数据重塑

```
import pandas as pd
df2 = pd.DataFrame({'Date': ['2021-12-25', '2021-12-26', '2021-12-25', '2021-12-27', '2021-12-26', '2021-12-27'], \
                    'Type': ['a', 'b', 'c', 'a', 'a', 'c'], \
                    'Value': [1.34, 10.2, 20.43, 50.31, 0.26, 20.64]})
df3 = df2.pivot(index='Date', columns='Type', values='Value') # 将行变为列
df4 = pd.pivot_table(df2, values='Value', index='Date', columns='Type') # 将行变为列
stacked = df2.stack() # 透视列标签
stacked.unstack() # 透视索引标签
pd.melt(df2, id_vars=['Date'], value_vars=['Type', 'Value'], value_name='Observations') # 将列转
```

透视

透视表

堆叠(轴旋转)

融合 / Melt

## 10. 迭代

迭代遍历数据帧

`df2.iteritems()` # (列索引, 序列) 键值对  
`df2.iterrows()` # (行索引, 序列) 键值对

## 11. 高级索引

基础选择

```
df3.loc[:, (df3>1).any()] # 选择任一值大于1的列
df3.loc[:, (df3>1).all()] # 选择所有值大于1的列
df3.loc[:, df3.isnull().any()] # 选择含NaN值的列
df3.loc[:, df3.notnull().all()] # 选择不含NaN值的列
```

通过 isin 选择

# 选择指定列为某一类型的数值

```
df2[(df2.Type.isin(['b', 'c']))]
```

# 选择特定值

```
df3.filter(items=['a', 'b'])
```

通过 where 选择

# 选择子集

```
s.where(s>0)
```

通过 query 选择

# 查询 DataFrame

```
df2.query('Value > 10')
```

	Country	Capital	Population
0	Belgium	Brussels	11190846
1	India	New Delhi	1303171035
2	Brazil	Brasília	207847528
3	Brazil	Brasília	207847528
4			

### 11.1 设置 / 取消索引

```
df.set_index('Country') # 设置索引
df4 = df.reset_index() # 重置索引 0-n
# 重命名 DataFrame 列名
df = df.rename(index=str, \
               columns={'Country': 'cntry', \
                        'Capital': 'cpt1', \
                        'Population': 'ppltn'})
```

### 11.2 重设索引

```
s2 = s.reindex(['a', 'c', 'd', 'e', 'b'])
前向填充
df.reindex(range(4), method='ffill')
后向填充
s3 = s.reindex(range(5), method='bfill')
```

### 11.3 多重索引

```
arrays = [np.array([1, 2, 3]), np.array([5, 4, 3])]
df5 = pd.DataFrame(np.random.rand(3, 2), index=arrays)
tuples = list(zip(*arrays))

index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
df6 = pd.DataFrame(np.random.rand(3, 2), index=index)
df2.set_index(["Date", "Type"])
```

## 12. 数据滤重

数据帧自带一系列函数对数据重复值进行处理

```
s3.unique()           # 返回唯一值
df2.duplicated('Type') # 查找重复值
df2.drop_duplicates('Type', keep='last') # 去除重复值
df.index.duplicated()  # 查找重复索引
```

## 13. 数据分组

分组聚合

```
df2.groupby(by=['Date', 'Type']).mean() # 分组求均值
df4.groupby(level=0).sum()
df4.groupby(level=0).agg({'a': lambda x: sum(x)/len(x), 'b': np.sum})
```

转换

```
customSum = lambda x: (x+x%2)
df4.groupby(level=0).transform(customSum)
```

## 14. 缺失值

```
df.dropna()           # 去除缺失值 NaN
df3.fillna(df3.mean()) # 用预设值填充缺失值 NaN
df2.replace("a", "f") # 用一个值替换另一个值
```

## 15. 合并数据

### 15.1 合并 – Merge

```
data1 = pd.DataFrame({'key': ['K0', 'K1', 'K2', 'K3'], 'A': ['A0', 'A1', 'A2', 'A3'], 'B': ['B0', 'B1', 'B2', 'B3']})
data2 = pd.DataFrame({'key': ['K0', 'K1', 'K3', 'K4'], 'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']})
pd.merge(data1, data2, how='left', on='key')
pd.merge(data1, data2, how='right', on='key')
pd.merge(data1, data2, how='inner', on='key')
pd.merge(data1, data2, how='outer', on='key')
```



### 15.2 拼接 – Concatenate

纵向

```
s.append(s2)
```

横向 / 纵向

```
pd.concat([s, s2], axis=1, keys=['One', 'Two'])
pd.concat([data1, data2], axis=1, join='inner')
```

### 15.3 连接 – Join

```
data1.join(data2, how='right', lsuffix='_1', rsuffix='_2')
```

Show Me AI

## 16. 日期转换

pandas 包含对时间型数据变换与处理的函数

```
df2['Date'] = pd.to_datetime(df2['Date'])
df2['Date'] = pd.date_range('2021-12-25', periods=6, freq='M')
import datetime
dates = [datetime.date(2021, 12, 25), datetime.date(2021, 12, 26)]
index = pd.DatetimeIndex(dates)
index = pd.date_range(datetime.date(2021, 12, 25), end=datetime.date(2022, 12, 26), freq='BM')
```

## 17. 可视化

Series 和 Dataframe 都自带 plot 绘图功能

```
import matplotlib.pyplot as plt
s.plot()
plt.show()
df2.plot()
plt.show()
```

## 18. 调用帮助

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

## 数据科学工具库速查表



**NumPy** 是 Python 数据科学计算的核心库，提供了高性能多维数组对象及处理数组的工具。使用以下语句导入 NumPy 库：

```
import numpy as np
```



**SciPy** 是基于 NumPy 创建的 Python 科学计算核心库，提供了众多数学算法与函数。



**Pandas** 是基于 NumPy 创建的 Python 库，为 Python 提供了易于使用的的数据结构和数据分析工具。使用以下语句导入：

```
import pandas as pd
```



**Matplotlib** 是 Python 的二维绘图库，用于生成符合出版质量或跨平台交互环境的各类图形。

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



**Seaborn** 是基于 matplotlib 开发的高阶 Python 数据可视图库，用于绘制优雅、美观的统计图形。使用下列别名导入该库：

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

```
import seaborn as sns
```



**Bokeh** 是 Python 的交互式可视图库，用于生成在浏览器里显示的大规模数据集高性能可视图。Bokeh 的中间层通用 **bokeh.plotting** 界面主要为两个组件：数据与图示例。

```
from bokeh.plotting import figure
```

```
from bokeh.io import output_file, show
```



**PySpark** 是 Spark 的 Python API，允许 Python 调用 Spark 编程模型。Spark SQL 是 Apache Spark 处理结构化数据模块。

## AI 垂直领域工具库速查表



**Scikit-learn** 是开源的 Python 库，通过统一的界面实现机器学习、预处理、交叉验证及可视化算法。



**Keras** 是强大、易用的深度学习库，基于 Theano 和 TensorFlow 提供了高阶神经网络 API，用于开发和评估深度学习模型。



“TensorFlow™ is an open source software library for numerical computation using data flow graphs.” **TensorFlow** 是 Google 公司开发的机器学习架构，兼顾灵活性和扩展性，既适合用于工业生产也适合用于科学研究。



**PyTorch** 是 Facebook 团队 2017 年初发布的深度学习框架，有利于研究人员、爱好者、小规模项目等快速搞出原型。**PyTorch** 也是 Python 程序员最容易上手的深度学习框架。



**Hugging Face** 以开源的 NLP 预训练模型库 **Transformers** 而广为人知，目前 GitHub Star 已超过 54000+。**Transformers** 提供 100+ 种语言的 32 种预训练语言模型，简单，强大，高性能，是新手入门的不二选择。



**OpenCV** 是一个跨平台计算机视觉库，由 C 函数 /C++ 类构成，提供了 Python、MATLAB 等语言的接口。**OpenCV** 实现了图像处理和计算机视觉领域的很多通用算法。

## 编程语言速查表



**SQL** 是管理关系数据库的结构化查询语言，包括数据的增删查改等。作为数据分析的必备技能、岗位 JD 的重要关键词，SQL 是技术及相关岗位同学一定要掌握的语言。



**Python** 编程语言简洁快速、入门简单且功能强大，拥有丰富的第三方库，已经成为大数据和人工智能领域的主流编程语言。

More...

## AI 知识技能速查表



**Jupyter Notebook** 交互式计算环境，支持运行 40+ 种编程语言，可以用来编写漂亮的交互式文档。这个教程把常用的基础功能讲解得很清楚，对新手非常友好。



**正则表达式** 非常强大，能匹配很多规则的文本，常用于文本提取和爬虫处理。这也是一门令人难以捉摸的语言，字母、数字和符号堆在一起，像极了“火星文”。

More...



ShowMeAI 速查表 (©2021)

获取最新版 | <http://www.showmeai.tech/>

作者 | 韩信子 @ShowMeAI

设计 | 南乔 @ShowMeAI

数据科学工具库速查表

扫码回复“数据科学”

获取最新全套速查表

AI 垂直领域工具库速查表

扫码回复“工具库”

获取最新全套速查表

编程语言速查表

扫码回复“编程语言”

获取最新全套速查表

AI 知识技能速查表

扫码回复“知识技能”

获取最新全套速查表