

21

Indexing and Slicing Pandas DataFrame

Pandas 索引和切片

利用 DataFrame 的行列标签、整数索引



生命就是一个实验。实验做的越多，越好。

All life is an experiment. The more experiments you make, the better.

—— 拉尔夫·沃尔多·爱默生 (Ralph Waldo Emerson) | 美国思想家、文学家 | 1942 ~ 2018



- ▶ `pandas.dataframe.iloc()` 通过整数索引来选择 DataFrame 的行和列的索引器
- ▶ `pandas.DataFrame.isin()` 用于检查 DataFrame 中的元素是否在给定的值序列中
- ▶ `pandas.dataframe.loc()` 通过标签索引来选择 DataFrame 的行和列的索引器
- ▶ `pandas.DataFrame.query()` 筛选和过滤 DataFrame 数据的方法
- ▶ `pandas.DataFrame.where()` 在 DataFrame 中根据条件对元素进行筛选和替换的方法
- ▶ `pandas.MultiIndex.from_arrays()` 用于从多个数组创建多级索引的方法
- ▶ `pandas.MultiIndex.from_frame()` 用于从 DataFrame 创建多级索引的方法
- ▶ `pandas.MultiIndex.from_product()` 用于从多个可迭代对象的笛卡尔积创建多级索引的方法
- ▶ `pandas.MultiIndex.from_tuples()` 用于从元组列表创建多级索引的方法



21.1 数据帧的索引和切片

Pandas 的数据帧和 NumPy 数组这两种数据结构在 Python 数据科学生态系统中都扮演着重要的角色，但它们在索引和切片上有一些异同之处。

NumPy 数组一般是一个多维的、同质的数据结构，意味着 NumPy 数组通常包含相同数据类型的元素，并且维度是固定的。NumPy 数组使用基于 0 的整数索引。

Pandas 数据帧一般是一个二维的、异质的数据结构，可以包含不同数据类型的列，并且可以拥有拥有灵活的行和列标签。

NumPy 数组使用整数索引来访问元素，类似于 Python 的列表索引。例如，对于二维数组 `array`，可以使用 `array[row_index, column_index]` 来获取元素。

上一章提过，行标签、列标签特指数据帧的标签；而对于数据帧，行索引、列索引则是指行列整数索引，这一点类似 NumPy 二维数组。默认情况下，数据帧行标签、列标签均为基于 0 的整数索引。

Pandas 数据帧使用行列标签来进行索引和切片。类似 NumPy 数组，Pandas 数据帧还可以使用 `.iloc[]` 属性可以通过整数索引完成索引、切片。

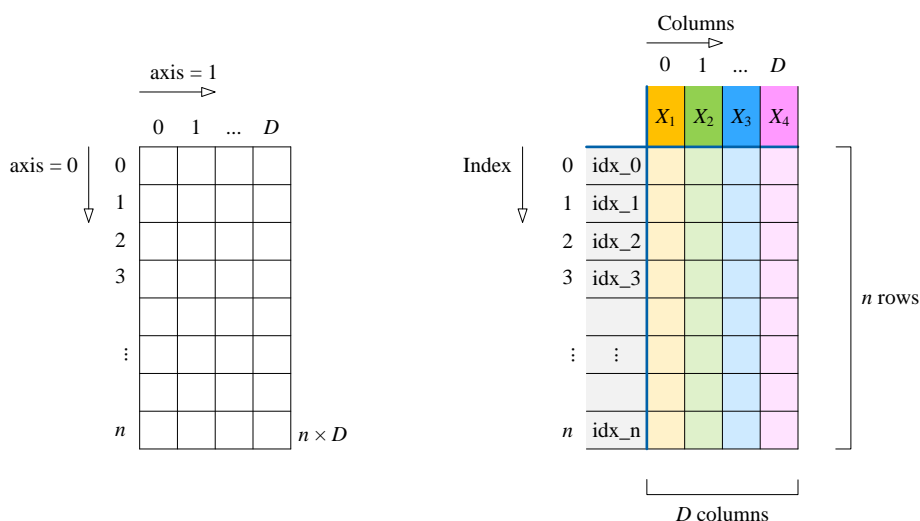
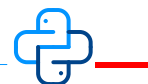


图 1. 比较 NumPy array 和 Pandas DataFrame 索引



本节配套的 Jupyter Notebook 文件是 `Bk1_Ch21_01.ipynb`。请大家一边阅读本章内容，一边在 JupyterLab 中实践。

21.2 提取特定列

图 2 所示为从数据帧取出特定一列的几种方法。

其中，`pandas.DataFrame.loc[]` 是 Pandas 中用于基于标签进行索引和切片重要工具，允许通过指定行标签、列标签来选择数据帧中的特定行和列，或者获取特定行或列上的值。

而 `pandas.DataFrame.iloc[]` 是 Pandas 中用于基于整数位置进行索引和切片的工具，方括号内的索引规则和 NumPy 二维数组完全一致。`pandas.DataFrame.iloc[]` 允许通过指定行的整数位置和列的整数位置来选择数据帧中的特定行和列，或者获取特定行或列上的值。

特别需要大家注意的是左侧的方法返回的是 Pandas Series（相当于一维数组），而右边的方法返回的是 Pandas DataFrame（相当于二维数组）。

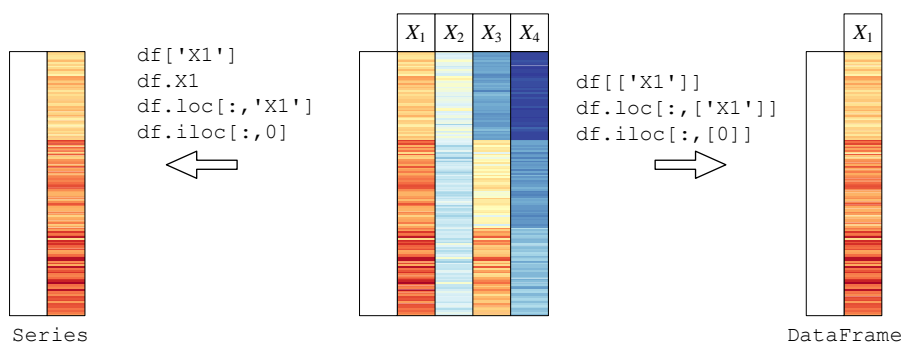


图 2. 提取一列

图 3 所示为从数据帧中取出连续多列的几种方法。相比而言，采用 `pandas.DataFrame.iloc[]` 取出连续多列最方便。类似 NumPy 数组，还可以利用 `pandas.DataFrame.iloc[]` 等间隔提取特定列，比如 `df.iloc[:, ::2]` 从第 0 行开始每 2 列取一列。图 4 则展示从数据帧取出不连续多列的方法。

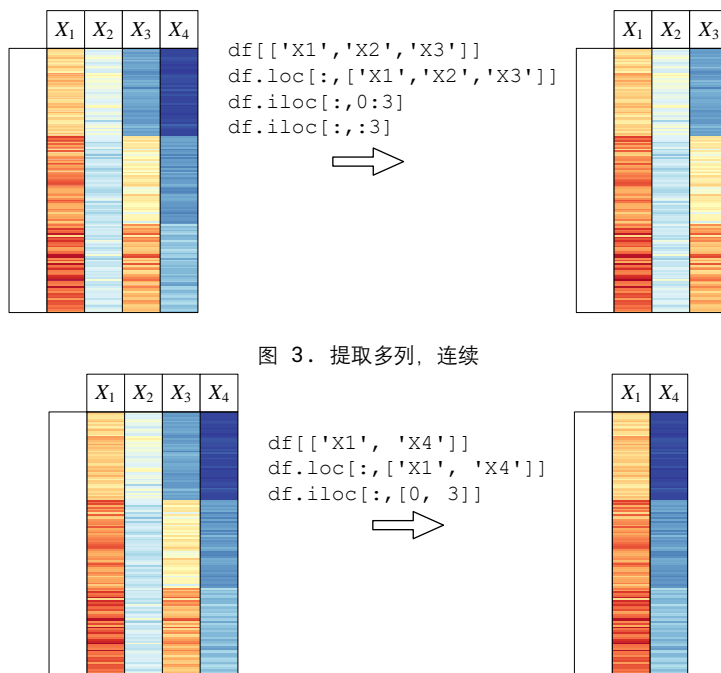


图 3. 提取多列，连续

图 4. 提取多列，不连续

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

21.3 提取特定行

图 5 所示为提取特定一行的几种方法。也需要大家注意的是，左侧提取结果为 Pandas Series，右侧提取结果为 Pandas DataFrame。此外，'idx_0' 为认为设定的行标签；数据帧采用的是默认从 0 开始的整数索引，则其行标签、行整数索引都是 0。

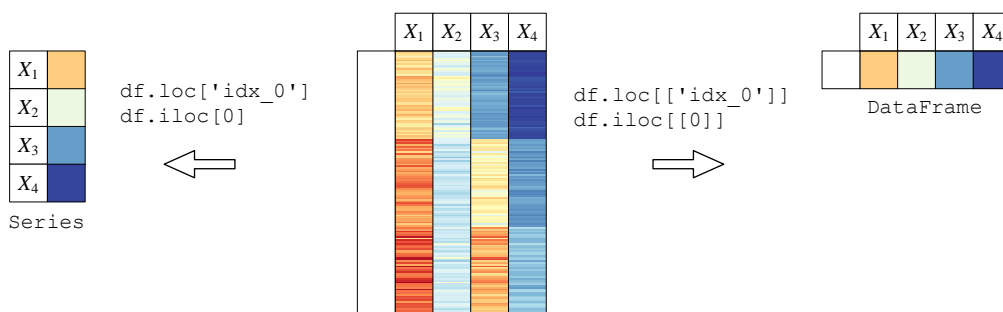


图 5. 提取一行

图 6 所示为从数据帧中取出连续多行的几种方法。相比而言，采用 `pandas.DataFrame.iloc[]` 取出连续多行比较容易。类似 NumPy 数组，还可以利用 `pandas.DataFrame.iloc[]` 等间隔提取特定行，比如 `df.iloc[:,2]` 从第 0 行开始每 2 行取一行。图 7 则展示从数据帧取出不连续多行的方法。

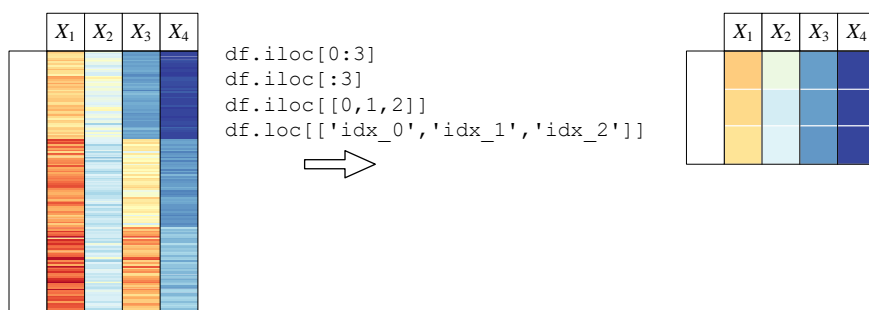


图 6. 提取多行，连续

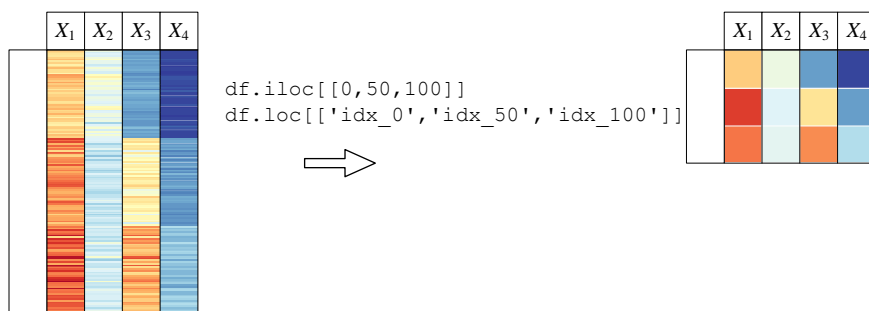


图 7. 提取多行，不连续

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。

版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微视频均发布在 B 站——生姜 DrGinger：<https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

21.4 提取特定元素

利用 `pandas.DataFrame.iloc[row_position, column_position]`，我们可以取得数据帧的特定位置元素，这一点和 NumPy 二维数组相同；本章配套代码提供若干示例，请大家自行学习。

本节要特别介绍 `at` 和 `iat` 方法。它俩是 Pandas DataFrame 中的快速访问器，用于在 DataFrame 中访问单个元素。

`at` 是基于标签的访问器，可以通过标签（行标签、列标签）快速获取数据帧单个元素，速度比 `loc` 快。

`iat` 是基于整数索引的访问器，可以通过整数索引（行索引、列索引）快速获取单个元素，速度比 `iloc` 快。

注意，使用 `at` 和 `iat` 访问器，只能访问单个元素，返回结果为具体元素。如果需要访问多个元素，应该使用 `loc` 或 `iloc`。

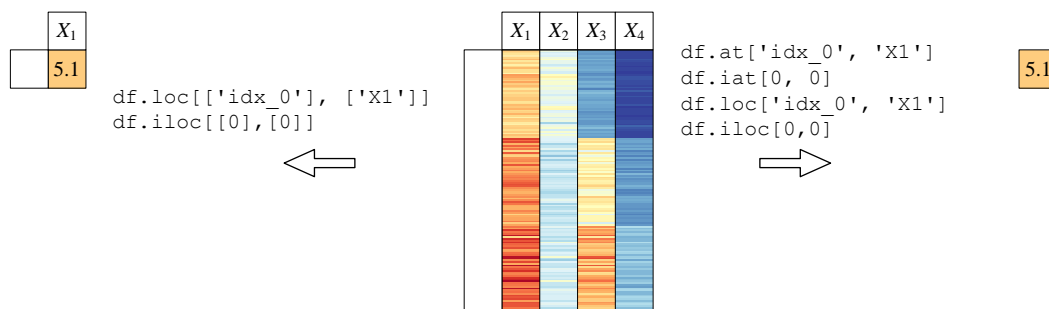


图 8. 提取特定元素

21.5 条件索引

在 Pandas 中，条件索引是通过布尔条件（Boolean expression）筛选数据帧中的行的一种技术。这意味着可以基于某些条件从数据帧中选择满足这些条件的特定行。条件索引使用布尔运算，如 `>`、`<`、`==`、`!=`、`&`、`|` 等等，来生成布尔值的数据帧，然后根据这些布尔值来筛选数据帧。

布尔条件

如图 9 所示，左侧的 `df` 为鸢尾花数据集前 4 列构成的数据帧。布尔运算 `(df > 6) | (df < 1.5)` 通过 `|` 或运算结合两个不等式，含义是数据帧中满足大于 6 或小于 1.5 的元素设为 `True`，否则设为 `False`。图 9 右侧的热图中深蓝色色块代表 `True`，浅蓝色色块代表 `False`。图 9 右侧这种方案还会用在可视化数据帧中缺失值。

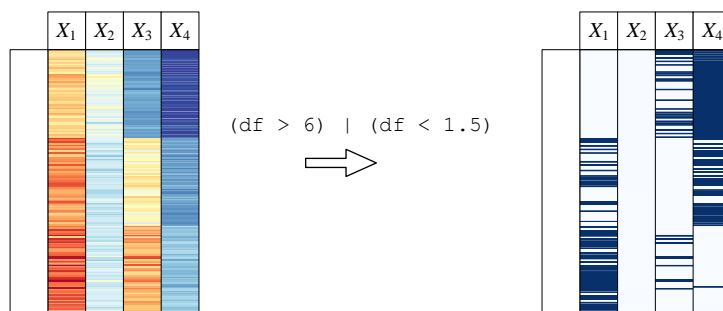


图 9. 满足条件的布尔数据帧

图 10 利用布尔数据帧筛选满足条件的行。其中，^a 创建了一个布尔条件数据帧，用于筛选 `iris_df` 中 "sepal_length" 列大于等于 7 的行。^b 使用上面创建的布尔条件 `condition` 对 `iris_df` 进行筛选，得到一个新的 DataFrame `iris_df_filtered`，其中只包含 "sepal_length" 列大于等于 7 的行，具体如图 11 所示。

```
import pandas as pd
import seaborn as sns

iris_df = sns.load_dataset("iris")
# 从Seaborn中导入鸢尾花数据帧
# 使用 drop(..., inplace=True) 删除一列
iris_df.drop(columns='species', inplace=True)
a condition = iris_df['sepal_length'] >= 7
# 创建了一个布尔条件condition数据帧
b iris_df_filtered = iris_df[condition]
# 只包含"sepal_length"列大于等于7的行
```

图 10. 利用布尔条件筛选数据帧; Bk1_Ch21_01.ipynb

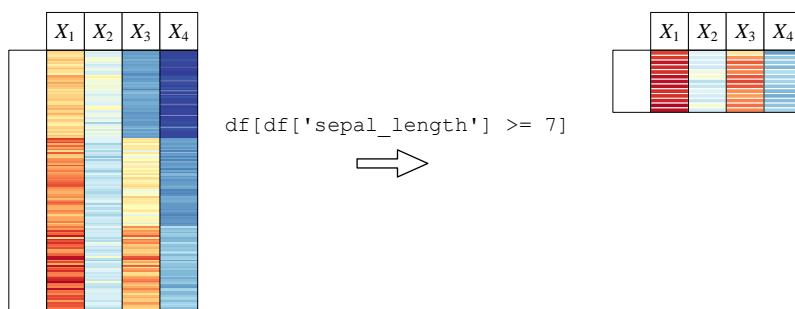


图 11. 满足条件的布尔数据帧

loc[]

实践中，一般更常用 `loc[]` 筛选满足条件的数据帧。举个例子，图 11 筛选可以通过这句话完成 `df.loc[df.loc[:, 'sepal_length'] >= 7, :]`。

本 PDF 文件为作者草稿，发布目的为方便读者在移动终端学习，终稿内容以清华大学出版社纸质出版物为准。


版权归清华大学出版社所有，请勿商用，引用请注明出处。

代码及 PDF 文件下载：<https://github.com/Visualize-ML>

本书配套微课视频均发布在 B 站——生姜 DrGinger: <https://space.bilibili.com/513194466>

欢迎大家批评指教，本书专属邮箱：jiang.visualize.ml@gmail.com

本章配套 Jupyter Notebook 给出更多实例，请大家自行学习。

表 1. 利用 loc[] 筛选示例;  Bk1_Ch21_01.ipynb

| 示例（假设 df 为鸢尾花数据集） | 说明 |
|--|---|
| <pre>df.loc[df.loc[:, 'species'] == 'versicolor', :] df.loc[df.species == 'versicolor', :]</pre> | 条件：鸢尾花种类 'species' 为 (==) 'versicolor' |
| <pre>df.loc[(df.sepal_length < 6.5) & (df.sepal_length > 6)] df.loc[(df.loc[:, 'sepal_length'] < 6.5) & (df.loc[:, 'sepal_length'] > 6)]</pre> | 条件：鸢尾花花萼长度 'sepal_length' 小于 (<) 6.5 且 (&) 大于 (>) 6 |
| <pre>df.loc[(df.loc[:, 'sepal_length'] < 6.5) & (df.loc[:, 'sepal_length'] > 6), ['petal_length', 'petal_width']]</pre> | 条件：鸢尾花花萼长度 'sepal_length' 小于 (<) 6.5 且 (&) 大于 (>) 6 返回：df 中 'petal_length' 和 'petal_width' 两列，同时满足两个条件 |
| <pre>df.loc[df['species'] != 'virginica']</pre> | 条件：鸢尾花种类 'species' 不是 (!=) 'virginica' |
| <pre>df.loc[df['species'].isin(['virginica', 'setosa'])] df.loc[df.species.isin(['virginica', 'setosa'])]</pre> | 条件：鸢尾花种类 'species' 在 (isin) 列表 ['virginica', 'setosa'] 之中 |
| <pre>df.loc[~df.species.isin(['virginica', 'setosa']), ['petal_length', 'petal_width']]</pre> | 条件：鸢尾花种类 'species' 不在 (~ ... isin) 列表 ['virginica', 'setosa'] 之中 返回：df 中 'petal_length' 和 'petal_width' 两列，满足条件所有行 |

query()

`query(expression)` 是 Pandas 中的一个方法，用于对数据帧进行查询操作。它允许通过指定一定的查询条件来筛选数据，并返回满足条件的行。其中，`expression` 是一个字符串，表示查询表达式，描述了筛选条件。通常，`expression` 由列名和运算符组成，可以使用布尔运算符，如 `==`、`!=`、`>`、`<`、`>=`、`<=` 等，来指定条件。还可以使用 `and`、`or` 和 `not` 等逻辑运算符来组合多个条件。默认 `inplace = False`，即不在原地修改数据帧。如果 `inplace=True`，则会直接在原始数据帧上进行修改，不返回一个新的数据帧。

表 2 给出若干示例，`query()` 内的条件很容易理解，请大家自行学习。

表 2. 利用 query() 筛选示例;  Bk1_Ch21_01.ipynb

示例（假设 df 为鸢尾花数据集）

| |
|---|
| <code>df.query('sepal_length > 2*sepal_width')</code> |
| <code>df.query("species == 'versicolor'")</code> |
| <code>df.query("not (sepal_length > 7 and petal_width > 0.5)")</code> |
| <code>df.query("species != 'versicolor'")</code> |
| <code>df.query("abs(sepal_length-6) > 1")</code> |
| <code>df.query("species in ('versicolor', 'virginica')")</code> |
| <code>df.query("sepal_length >= 6.5 or sepal_length <= 4.5")</code> |
| <code>df.query("sepal_length <= 6.5 and sepal_length >= 4.5")</code> |

21.6 多层索引

在 Pandas 中，多级索引（multi-index）是一种特殊的索引类型，允许在数据帧的行或列上具有多个层次的索引。这使得我们可以在更复杂的高维数据集上进行分层操作和查询。

多层行标签

图 12 所示为用列表创建两层行标签数据帧。

a 利用 `pandas.MultiIndex.from_arrays()` 构造两层行标签，结果为：

```
MultiIndex([( 'A', 1),
            ( 'A', 2),
            ( 'B', 3),
            ( 'B', 4),
            ( 'C', 5),
            ( 'C', 6),
            ( 'D', 7),
            ( 'D', 8)],
           names=[ 'I', 'II'])
```

b 构造两层行标签数据帧，如图 13 左图所示。图 14 所示为用 `loc[]` 对两层行标签索引、切片。类似地，我们还可以利用 `pandas.MultiIndex.from_tuples()` 从元组列表创建多级索引。此外，`pandas.MultiIndex.from_frame()` 是用于从 `DataFrame` 创建多级索引的方法。请大家参考本章配套的 Jupyter Notebook 自行学习。


```

import pandas as pd
import numpy as np
# 创建列表、数据
index_arrays = [['A', 'A', 'B', 'B', 'C', 'C', 'D', 'D'],
                 range(1, 9)]
data = np.random.randint(0, 9, size=(8, 4))

# 创建多层行索引
a row_idx = pd.MultiIndex.from_arrays(index_arrays,
                                     names=['I', 'II'])

# 创建DataFrame
b df = pd.DataFrame(data,
                    index=row_idx,
                    columns=['X1', 'X2', 'X3', 'X4'])

```

图 12. 用列表构造多层行标签; Bk1_Ch21_02.ipynb

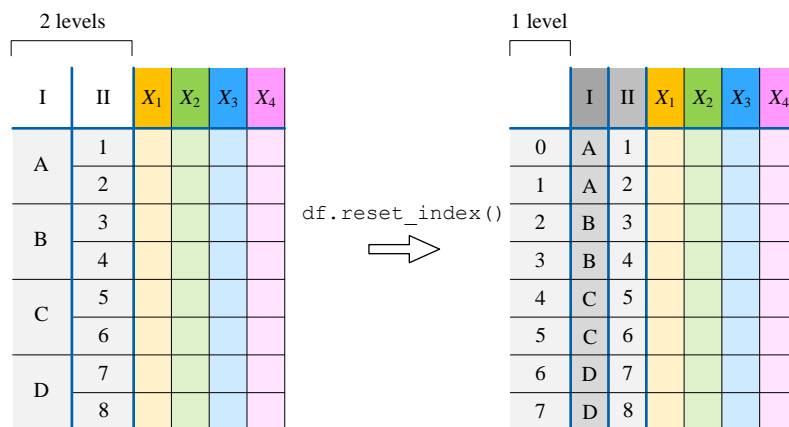


图 13. 两层行标签

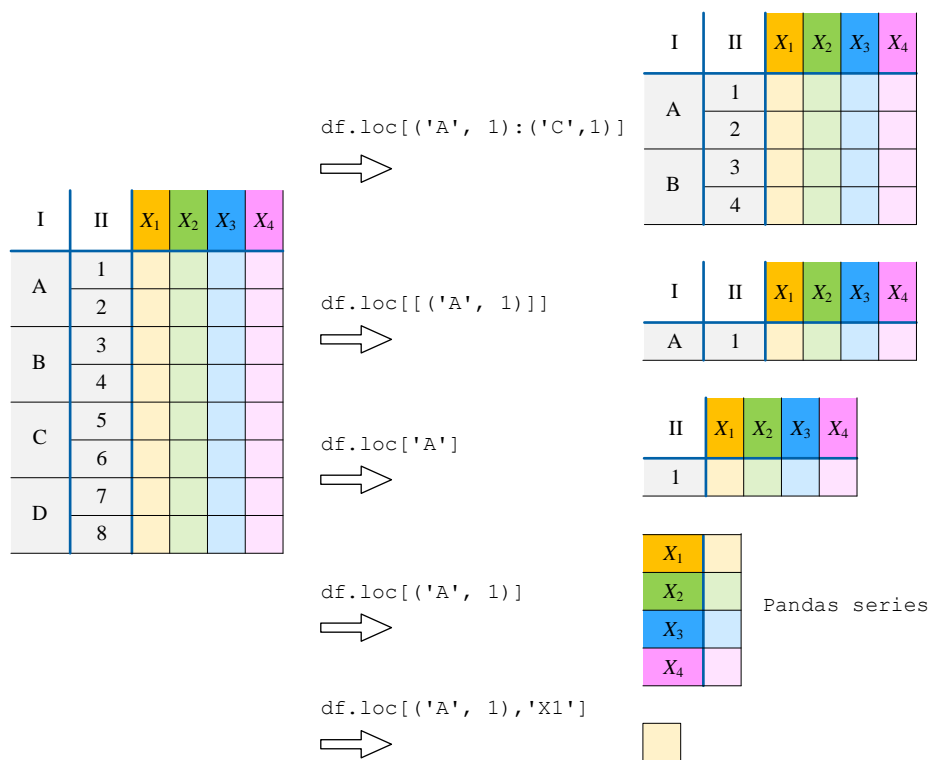


图 14. 两层行标签, 索引、切片

图 15 利用中利用 `pandas.MultiIndex.from_product()` 从多个可迭代对象的笛卡尔积创建多层行索引。图 15 代码产生如图 16 两个双层行标签数据帧。

`df_2.index.set_names('Level_0_idx', level=0, inplace=True)` 将第 0 级索引的名称设置为 'Level_0_idx'。

然后, 再用 `df_2.index.set_names('Level_1_idx', level=1, inplace=True)` 将第 1 级索引的名称设置为 'Level_1_idx'。

```

import pandas as pd
import numpy as np
# 示例数据
data = np.random.randint(0,9,size=(8,4))

# 两组列表
categories = ['A','B','C','D']
types = ['X', 'Y']

# 创建多层行索引, 先categories, 再types
a idx_1 = pd.MultiIndex.from_product([categories, types],
                                   names=['I', 'II'])
df_1 = pd.DataFrame(data, index=idx_1,
                    columns=['X1', 'X2', 'X3', 'X4'])

# 创建多层行索引, 先types, 再categories
b idx_2 = pd.MultiIndex.from_product([types, categories],
                                   names=['I', 'II'])
df_2 = pd.DataFrame(data, index=idx_2,
                    columns=['X1', 'X2', 'X3', 'X4'])

# 将第0级索引的名称设置为 'Level_0_idx'
c df_2.index.set_names('Level_0_idx', level=0, inplace=True)
# 将第1级索引的名称设置为 'Level_1_idx'
d df_2.index.set_names('Level_1_idx', level=1, inplace=True)

# 获取 DataFrame 中多级索引的第0级别 (level=0) 的所有标签值
e df_2.index.get_level_values(0)
# 获取 DataFrame 中多级索引的第1级别 (level=1) 的所有标签值
f df_2.index.get_level_values(1)

```

图 15. 用多个可迭代对象的笛卡尔积构造多层行标签; Bk1_Ch21_03.ipynb

| 2 levels | | | | | |
|----------|----|----------------|----------------|----------------|----------------|
| I | II | X ₁ | X ₂ | X ₃ | X ₄ |
| A | X | | | | |
| | Y | | | | |
| B | X | | | | |
| | Y | | | | |
| C | X | | | | |
| | Y | | | | |
| D | X | | | | |
| | Y | | | | |

| 2 levels | | | | | |
|----------|----|----------------|----------------|----------------|----------------|
| I | II | X ₁ | X ₂ | X ₃ | X ₄ |
| X | A | | | | |
| | B | | | | |
| | C | | | | |
| | D | | | | |
| Y | A | | | | |
| | B | | | | |
| | C | | | | |
| | D | | | | |

图 16. 两层行标签, 笛卡尔积

```

a df_2.xs('X', level='Level_0_idx')
# df_2.xs('X')
# 获取 Level_0_idx 等于 'X' 的所有行

b df_2.xs('A', level='Level_1_idx')
# 获取 Level_1_idx 等于 'A' 的所有行

c df_2.xs(('X', 'A'), level=['Level_0_idx', 'Level_1_idx'])
# df_2.xs(('X', 'A'))
# 获取 Level_0_idx 等于 'X' 且 Level_1_idx 等于 'A' 的所有行

```



图 17. 用 `pandas.xs()` 访问多级索引 `DataFrame` 中的数据, 使用时配合前文代码; Bk1_Ch21_03.ipynb

图 18 所示为将 `DataFrame` 的索引转换为字符串类型, 并且每个索引元素中的多个级别值用下划线连接成一个字符串。

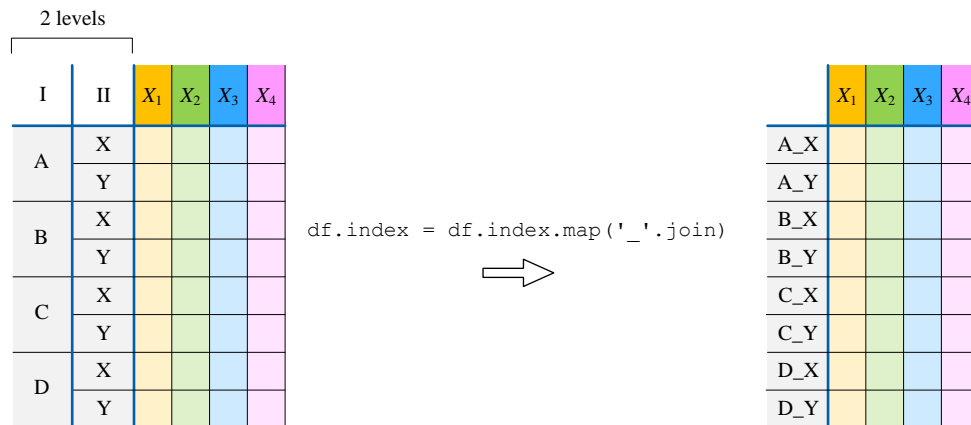


图 18. 两层行标签降为一层

多层列标签

图 19 所示为用列表创建两层列标签数据帧, 如图 20 左图所示。本章配套的 Jupyter Notebook 还介绍了利用笛卡尔积、元组、数据帧创建多层列标签数据帧, 请大家自行学习。图 21 所示为利用 `loc[]` 对多层列标签进行索引、切片。

```

import pandas as pd
import numpy as np
# 示例数据
data = np.random.randint(0,9,size=(8,4))

# 创建两层列标签列表
col_arrays = [['A', 'A', 'B', 'B'],
              ['X1', 'X2', 'X3', 'X4']]

# 创建两层列索引
a multi_col = pd.MultiIndex.from_arrays(col_arrays,
                                       names=['I', 'II'])

# 创建DataFrame
df = pd.DataFrame(data, columns=multi_col)

```

图 19. 用列表构造多层列标签; Bk1_Ch21_04.ipynb

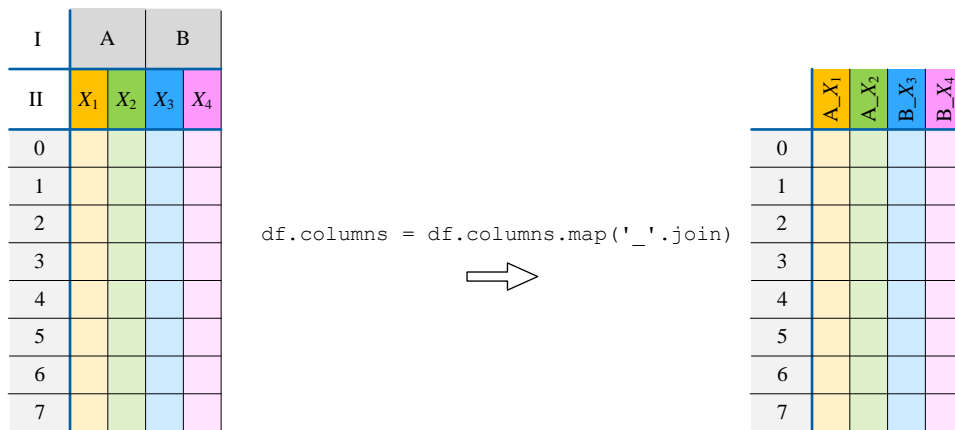


图 20. 两层行标签降为一层

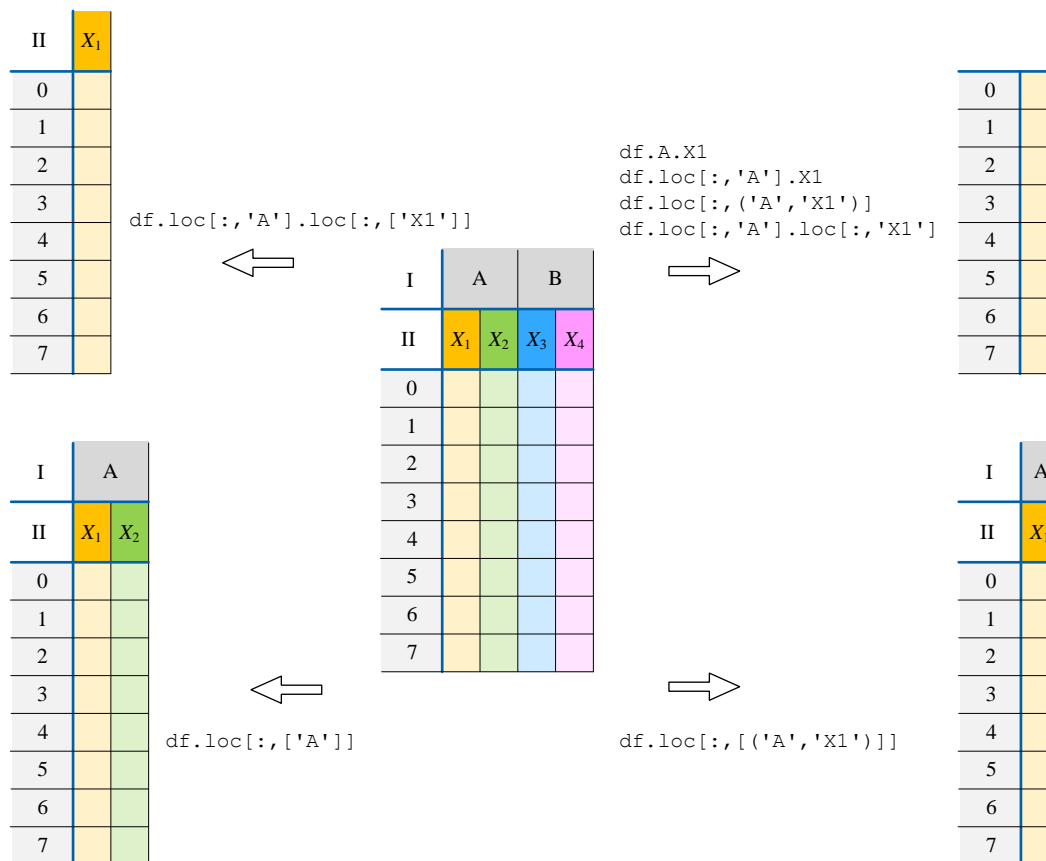


图 21. 两层列标签，索引、切片

21.7 时间序列数据帧索引和切片

本章最后简单聊聊如何对时间序列数据帧进行索引切片。

图 22 所示为利用蒙特卡洛模拟 (Monte Carlo simulation) 生成的随机行走，横轴为日期，纵轴为随机点位置。图中一共有 50 条轨迹，721 个日期点（包括起始点 0）。图 22 对应的数据类型为 Pandas DataFrame。

简单来说，蒙特卡罗模拟是通过随机抽样的方法进行数值计算的一种技术。它基于随机抽样的思想，通过生成大量的随机样本来估算数学问题的解。蒙特卡罗模拟广泛应用于金融、物理、工程等领域，用于解决复杂的概率、统计和优化问题。

如图 24 所示，我们利用数据帧切片方法从 50 条轨迹取出前两条（索引为 0 和 1）。以图 24 中任意一条轨迹为例，它的每天变化量都是来自于服从标准正态分布的随机数。

如图 25 所示，我们可以进一步在时间索引上切片，取出部分数据。



鸢尾花书《统计至简》将介绍如何用蒙特卡洛模拟估算面积、积分、圆周率，以及产生满足特性相关性的随机数。《数据有道》会深入一步，介绍有关随机过程的基础知识。

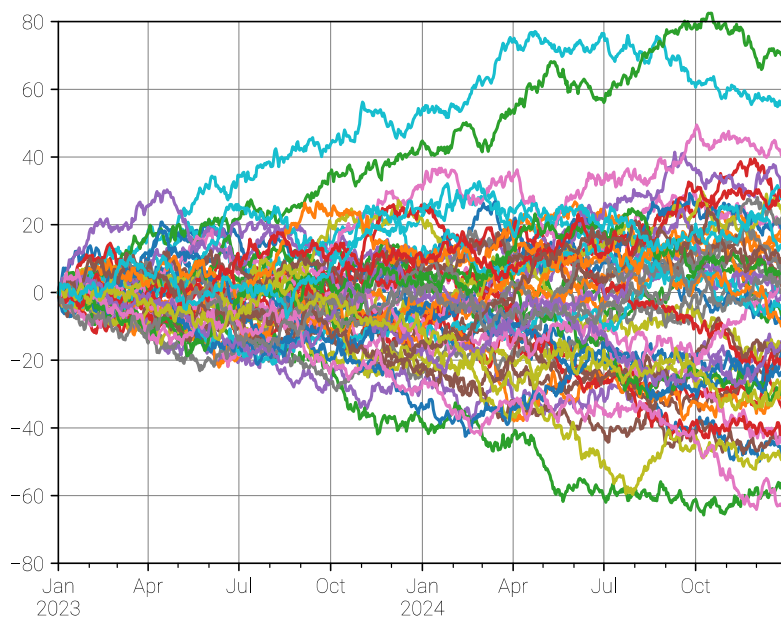


图 22. 50 条随机行走轨迹，时间为 2 年 720 天

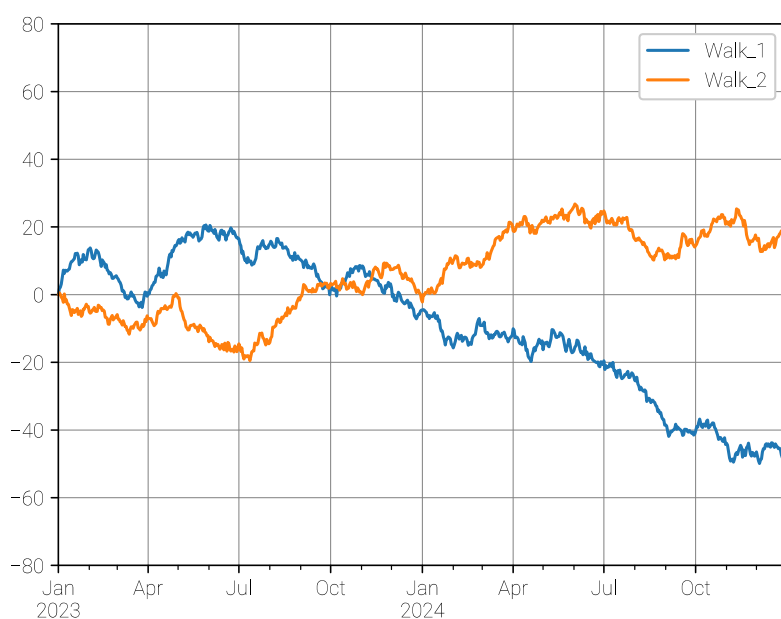


图 23. 前 2 条随机行走轨迹，时间为 2 年 720 天

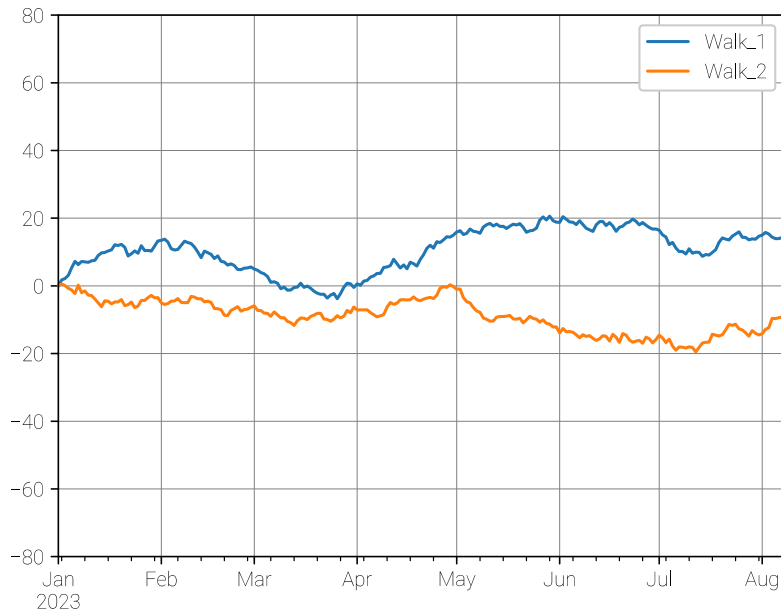


图 24. 前 2 条随机行走轨迹，时间为前 220 天

图 25 代码生成图 22、图 23、图 24，下面聊聊其中关键语句。

a 利用 `pandas.date_range()` 创建了一个从 2023 年 1 月 1 日开始，为期两年的每一天的日期范围，结果存储在 `date_range` 变量中。

参数 `start='2023-01-01'` 指定了日期范围的起始日期。建议使用 `YYYY-MM-DD` 这种日期格式。

参数 `periods=365*2` 指定了日期范围的长度，即两年的长度。

参数 `freq='D'` 指定了日期范围的频率，'D' 表示每天。

b 创建一个空数据帧，指定行索引为日期。

c 设定 NumPy 随机数发生器的种子。

d 用 `numpy.random.normal()` 生成满足标准正态分布随机数，用来代表每天行走的步长。

e 添加 0 作为初始状态。当然大家可以在空数据帧中直接加一行 0。

f 用 `cumsum()` 计算累加，代表随机行走位置随时间变化。

g 将 for 循环中每次迭代得到的随机路径保存在 DataFrame 中，并设置特定列标签。

? 请大家想办法去掉这个 for 循环。

h 用 `plot()` 方法绘制线图可视化随机行走。参数 `legend = False` 代表不展示图例。

i 用 `iloc[]` 方法取出前两条随机行走路径，这一句被故意注释掉了。

j 用 `df[['Walk_1', 'Walk_2']]` 取出两条随机行走轨迹，并可视化。

k 用 `loc[]` 方法取出特定时间切片，时间序列索引为 `'2023-01-01':'2023-08-08'`。

l 用 `iloc[]` 方法完成和上一句想听操作，这一句也被故意注释掉了。


```

# 导入包
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# 创建日期范围，假设为2年（365*2天）
a date_range = pd.date_range(start='2023-01-01',
                             periods=365*2, freq='D')

# 创建一个空的 DataFrame，用于存储随机行走数据
b df = pd.DataFrame(index=date_range)

# 模拟50个随机行走
num_path = 50
# 设置随机种子以保证结果可重复
c np.random.seed(0)

for i in range(num_path):
    # 生成随机步长，每天行走步长服从标准正态分布
    d step_idx = np.random.normal(loc=0.0, scale=1.0,
                                  size=len(date_range) - 1)

    # 增加初始状态
    e step_idx = np.append(0, step_idx)

    # 计算累积步数
    f walk_idx = step_idx.cumsum()

    # 将行走路径存储在DataFrame中，列名为随机行走编号
    g df[f'Walk_{i + 1}'] = walk_idx

# 请大家想办法去掉for循环

# 绘制所有随机行走轨迹
h df.plot(legend = False)
plt.grid(True)
plt.ylim(-80, 80)

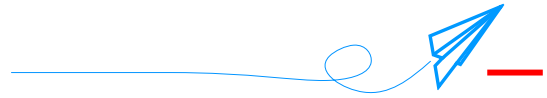
# 绘制前两条随机行走
i # df.iloc[:, [1, 0]].plot(legend = True)
j df[['Walk_1', 'Walk_2']].plot(legend = True)
plt.grid(True)
plt.ylim(-80, 80)

# 绘制前两条随机行走，特定时间段
k df.loc['2023-01-01':'2023-08-08',
        ['Walk_1', 'Walk_2']].plot(legend = True)

l # df.iloc[0:220, 0:2].plot(legend = True)
plt.grid(True)
plt.ylim(-80, 80)

```

图 25. 蒙特卡罗模拟时间序列切片; Bk1_Ch21_05.ipynb



本章介绍了常见的 Pandas 数据帧索引和切片方法，其中稍有难度的是条件索引、多层索引，请大家格外注意。

Pandas 中有关时间序列数据帧的操作有很多。限于篇幅，我们不会展开介绍。大家如果感兴趣的话，请参考。

https://pandas.pydata.org/docs/user_guide/timeseries.html