

22

Manipulating Pandas DataFrames

Pandas 规整

concat()、join()、merge()、pivot()、stack()、groupby() ...



希望，是一个醒来的梦想。

Hope is a waking dream.

—— 亚里士多德 (Aristotle) | 古希腊哲学家 | 384 ~ 322 BC



- ◀ pandas.concat() 将多个数据帧在特定轴（行、列）方向进行拼接
- ◀ pandas.DataFrame.apply() 将一个自定义函数或者 lambda 函数应用到数据帧的行或列上，实现数据的转换和处理
- ◀ pandas.DataFrame.drop() 删除数据帧特定列
- ◀ pandas.DataFrame.groupby() 在分组后的数据上执行聚合、转换和其他操作，从而对数据进行更深入的分析 and 处理
- ◀ pandas.DataFrame.join() 将两个数据集按照索引或指定列进行合并
- ◀ pandas.DataFrame.merge() 按照指定的列标签或索引进行数据库风格的合并
- ◀ pandas.DataFrame.pivot() 用于将数据透视成新的行和列形式的函数
- ◀ pandas.DataFrame.stack() 将 DataFrame 中的列转换为多级索引的行形式的函数
- ◀ pandas.DataFrame.unstack() 将 DataFrame 中的多级索引行转换为列形式的函数
- ◀ pandas.melt() 将宽格式数据转换为长格式数据的函数，将多个列“融化”成一列
- ◀ pandas.pivot_table() 根据指定的索引和列对数据进行透视，并使用聚合函数合并重复值的函数
- ◀ pandas.wide_to_long() 将宽格式数据转换为长格式数据的函数，类似于 melt()，但可以处理多个标识符列和前缀



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

22.1 Pandas 数据帧规整

Pandas 是一种用于数据处理和分析的 Python 库，它提供了多种数据规整方法来整理和准备数据，使之能够更方便地进行分析和可视化。本章将介绍如下三种数据帧的拼接合并方法。

- ▶ 方法 `concat()` 将多个数据帧在特定轴方向进行拼接。
- ▶ 方法 `join()` 将两个数据集按照索引或指定列进行合并。
- ▶ 方法 `merge()` 按照指定的列标签或索引进行数据库风格的合并。

此外，在 Pandas 中，数据帧的重塑和透视操作是指通过重新组织数据的方式，使数据呈现出不同的结构，以满足特定的分析需求。

具体来说，数据帧重塑（reshaping）是指改变数据的行和列的排列方式。数据帧透视（pivoting）是指通过旋转数据的行和列，以重新排列数据，并根据指定的聚合函数来生成新的数据帧。这样做可以更好地展示数据的结构和统计特征。

长格式、宽格式是本章重要概念。如图 1 所示，长格式（long format）和宽格式（wide format）是两种不同的数据存储形式。

如图 1（a）所示，长格式类似流水账，每一行代表一个观察值，比如某个学生某科目期中考试成绩。

如图 1（b）所示，宽格式更像是“矩阵”，每一行代表一个特定观察条件，比如某个特定学生的学号。此外，宽格式数据的列用于表示不同的特征或维度，比如特定科目。

显然，长格式、宽格式之间可以很容易相互转化。Pandas 提供很多方法用来完成数据帧的重塑和透视。

(a) long format			(b) wide format			
Student ID	Subject	Midterm	Subject	Art	Math	Science
1	Math	3	Student ID			
1	Art	4	1	5	4	NaN
2	Science	5	2	5	NaN	3
2	Art	3	3	NaN	4	5
3	Math	4	4	3	5	NaN
3	Science	4				
4	Art	4				
4	Math	5				

图 1. 比较长格式、宽格式

本章将介绍的重塑和透视操作如下。

- ▶ 方法 `pivot()` 用于根据一个或多个列创建一个新的数据透视表。`pivot_table()` 与 `pivot()` 类似，它也可以执行透视操作，但是它允许对重复的索引值进行聚合，产生一个透视表。它对于处理有重复数据的情况更加适用。

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>

本书配套微课程视频均发布在 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

- ▶ 方法 `stack()` 用于将数据帧从宽格式转换为长格式。方法 `melt()` 也可以用于将数据从宽格式转换为长格式，类似于 `stack()`。
- ▶ 方法 `unstack()` 是 `stack()` 的逆操作，用于将数据从长格式转换为宽格式，也就是将数据从索引转换为列。

本章将展开介绍上述方数据帧方法。

22.2 拼接: `pandas.concat()`

`pandas.concat()` 是 `pandas` 库中的一个函数，用于将多个数据结构按照行或列的方向进行合并。它可以将数据连接在一起，形成一个新的 `DataFrame`。

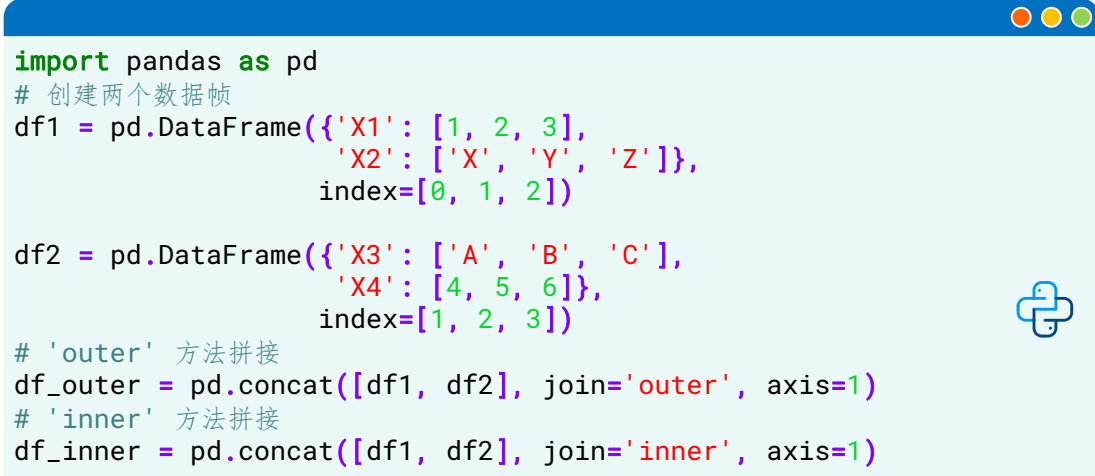
这个函数的主要参数为 `pandas.concat(objs, axis=0, join='outer', ignore_index=False)`。

参数 `objs`：这是一个需要连接的对象列表，比如 `[df1, df2, df3]`。

参数 `axis` 指定连接的轴向，可以是 0 或 1，默认为 0；0 表示按行连接（如图 3 所示），1 表示按列连接（如图 4 所示）。

参数 `join` 指定拼接的方式，可以是 'inner'、'outer'，默认是 'outer'。'inner' 表示内连接，只保留两个数据集中共有的列/行。'outer' 表示外连接，保留所有列/行，缺失值用 `NaN` 填充。

图 2 给出的代码比较 'outer' 和 'inner' 和两种拼接方式。



```
import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'X1': [1, 2, 3],
                    'X2': ['X', 'Y', 'Z']},
                    index=[0, 1, 2])

df2 = pd.DataFrame({'X3': ['A', 'B', 'C'],
                    'X4': [4, 5, 6]},
                    index=[1, 2, 3])

# 'outer' 方法拼接
a df_outer = pd.concat([df1, df2], join='outer', axis=1)
# 'inner' 方法拼接
b df_inner = pd.concat([df1, df2], join='inner', axis=1)
```

图 2. 用 `concat()` 拼接，比较 'outer' 和 'inner'；

a 的结果如图 5 所示，图中 `x` 代表 `NaN` 缺失值。**b** 的结果如图 6 所示。

参数 `ignore_index` 为布尔值，默认为 `False`；如果设置为 `True`，将会重新生成索引，忽略原来的索引。

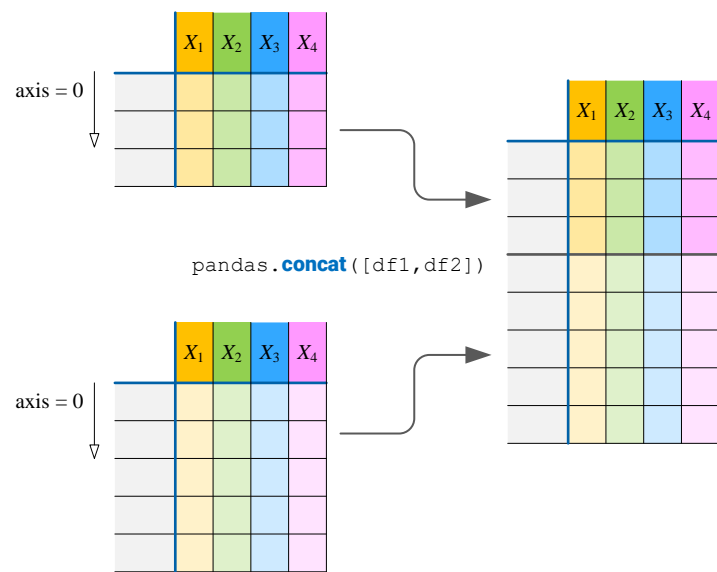


图 3. 利用 `pandas.concat()` 完成轴方向拼接，`axis = 0`（默认）

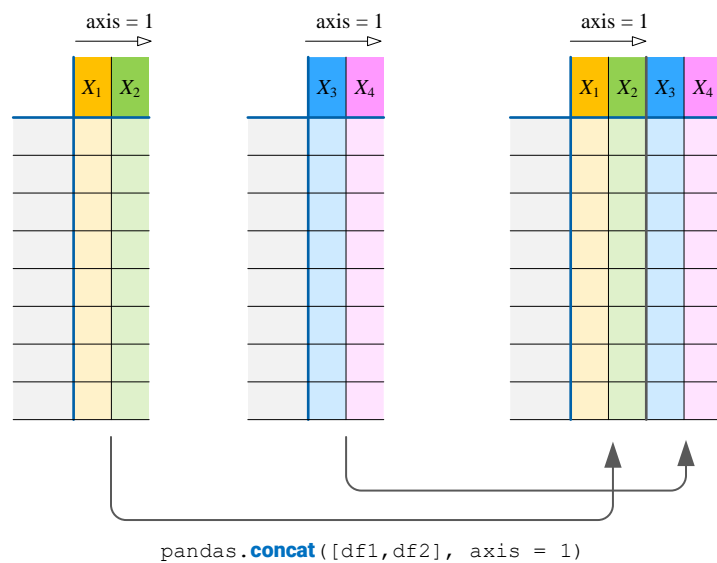
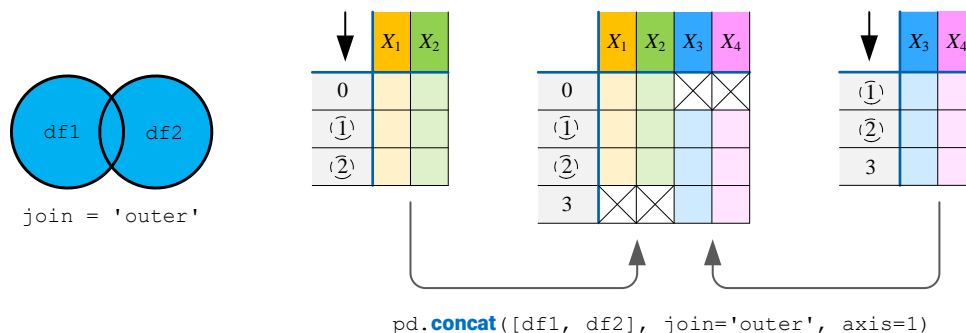
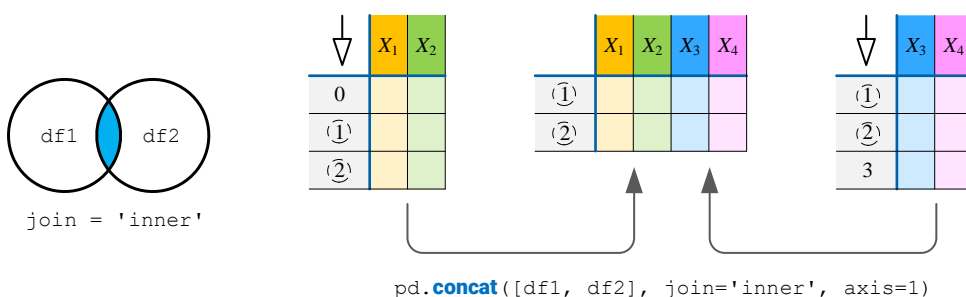


图 4. 利用 `pandas.concat()` 完成轴方向拼接，`axis = 1`

图 5. 利用 `pandas.concat()` 完成合并, `join = 'outer'`图 6. 利用 `pandas.concat()` 完成合并, `join = 'inner'`

22.3 合并: `pandas.join()`

在 Pandas 中, `join` 是 `DataFrame` 对象的一个方法, 用于按照索引 (默认) 或指定列合并两个 `DataFrame`。

这个函数的主要参数为 `DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='')`。

参数 `other` 是要连接的另一个 `DataFrame`。

参数 `on` 是指定连接的列名或列标签级别 (多级列标签的情况) 的名称。如果不指定, 将会以两个 `DataFrame` 的索引为连接依据。

参数 `how` 指定连接方式, 可以是 'left' (左连接)、'right' (右连接)、'outer' (外连接)、'inner' (内连接) 或 'cross' (交叉连接), 默认是 'left'。图 7 代表比较 'left'、'right'、'outer'、'inner' 这四种方法。

如图 8 所示, 'left' 使用左侧 `DataFrame` 的索引或指定列进行合并, 对应图 7 中 **a**。

如图 9 所示, 'right' 使用右侧 `DataFrame` 的索引或指定列进行合并, 对应图 7 中 **b**。

如图 10 所示, 'outer' 使用两个 `DataFrame` 的并集索引或指定列进行合并, 缺失值用 `NaN` 填充, 对应图 7 中 **c**。

如图 11 所示, 'inner' 使用两个 `DataFrame` 的交集索引或指定列进行合并, 对应图 7 中 **d**。

```

import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'X1': [1, 2, 3],
                    'X2': ['X', 'Y', 'Z']},
                    index=[0, 1, 2])

df2 = pd.DataFrame({'X3': ['A', 'B', 'C'],
                    'X4': [4, 5, 6]},
                    index=[1, 2, 3])

# 'left' 方法合并
a df_left = df1.join(df2, how='left')
# 'right' 方法合并
b df_right = df1.join(df2, how='right')
# 'outer' 方法合并
c df_outer = df1.join(df2, how='outer')
# 'inner' 方法合并
d df_inner = df1.join(df2, how='inner')

```

图 7. 用 join() 合并, 比较 'left'、'right'、'outer'、'inner';

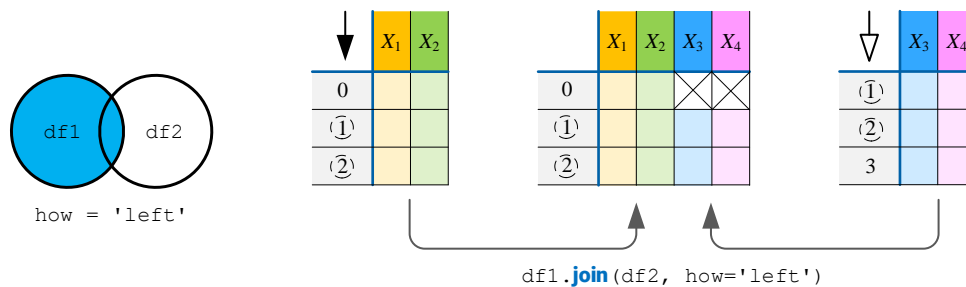


图 8. 利用 pandas.join() 完成合并, join = 'left'

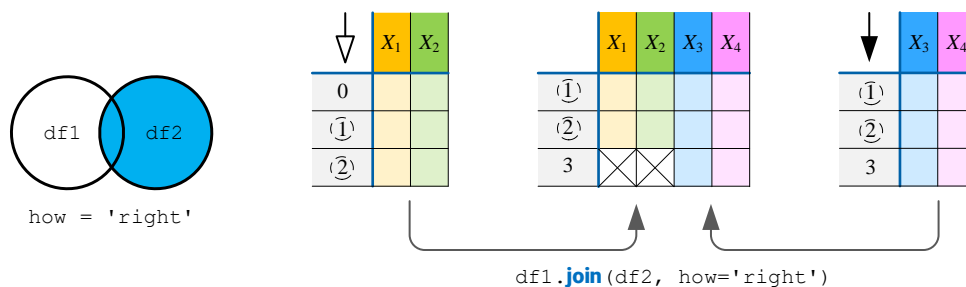


图 9. 利用 pandas.join() 完成合并, join = 'right'

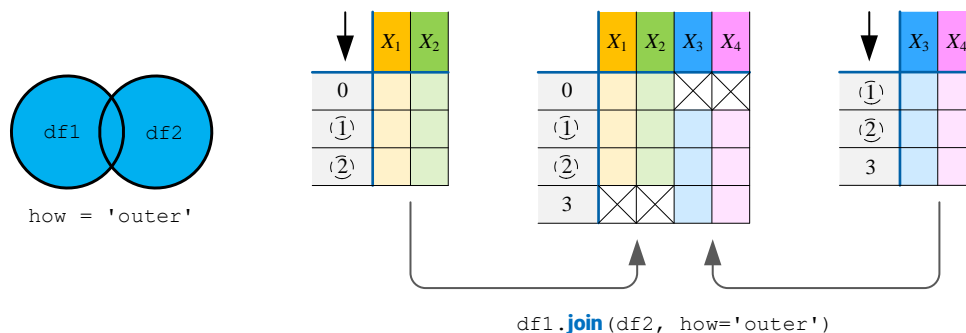


图 10. 利用 pandas.join() 完成合并, join = 'outer'

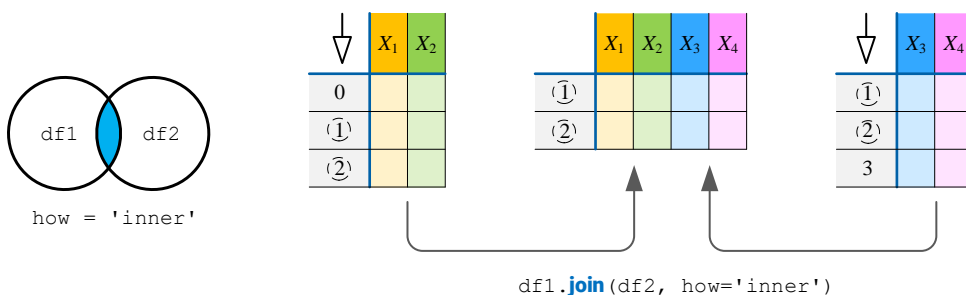


图 11. 利用 pandas.join() 完成合并, join = 'inner'

如图 12 代码所示, 'cross' 连接是一种笛卡尔积的连接方式, 它会将两个 DataFrame 的所有行进行组合, 从而得到两个 DataFrame 之间的所有可能组合。图 13 给出这种合并方法的图解。

'cross' 这种连接方式在 SQL 中称为 "CROSS JOIN"。'cross' 连接方式适用于较小的 DataFrame, 因为连接后的结果行数会呈指数增长。如果 DataFrame 较大, 这种连接方式可能会导致非常庞大的结果, 从而占用大量的内存和计算资源。因此, 在使用 'cross' 连接时, 应该谨慎操作, 确保不会导致资源耗尽。

当连接的两个 DataFrame 中存在同名的列时, 可以通过 lsuffix 和 rsuffix 这两个参数为左边和右边的列名添加后缀 (suffix), 避免列名冲突。

```
import pandas as pd
# 创建两个数据帧
df1 = pd.DataFrame({'A': ['X', 'Y', 'Z']})
df2 = pd.DataFrame({'B': [1, 2]})

# 使用 'cross' 连接
df_cross = df1.join(df2, how='cross')
```

图 12. 用 join() 合并, how = 'cross';

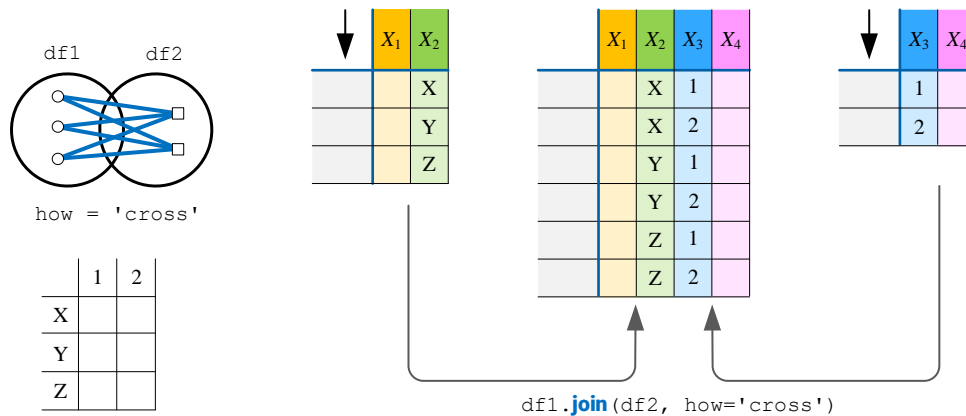


图 13. 利用 pandas.join() 完成合并, join = 'cross'

22.4 合并: pandas.merge()

实践中, 相较本章前文介绍的两种方法, merge() 更灵活, 可以处理更多种合并情况。merge() 可以通过指定列标签合并 (参数 left_on 和 right_on, 或 on), 可以指定索引 (left_index 和 right_index) 合并。merge() 还支持 'left'、'right'、'outer'、'inner' 或 'cross' 五种合并方法。

基于单个列合并

图 14 所示为 merge() 通过参数 on 指定同名列标签, 完成 df_left 和 df_right 两个数据帧合并, 合并方法为 how = 'left'。如图 15 所示, 当两个数据帧有同名列标签时, 合并后同名标签会加后缀以便区分, 默认标签为 (“_x”, “_y”)。

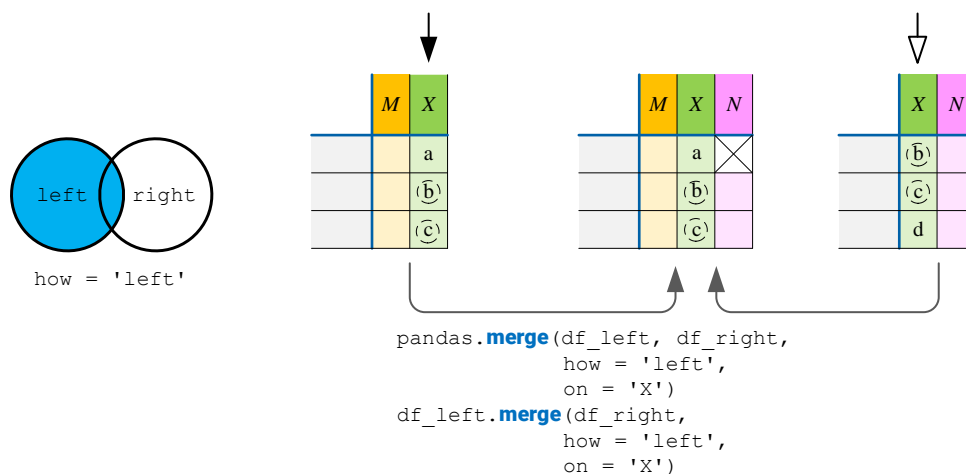


图 14. 利用 pandas.merge() 完成合并, how = 'left'

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

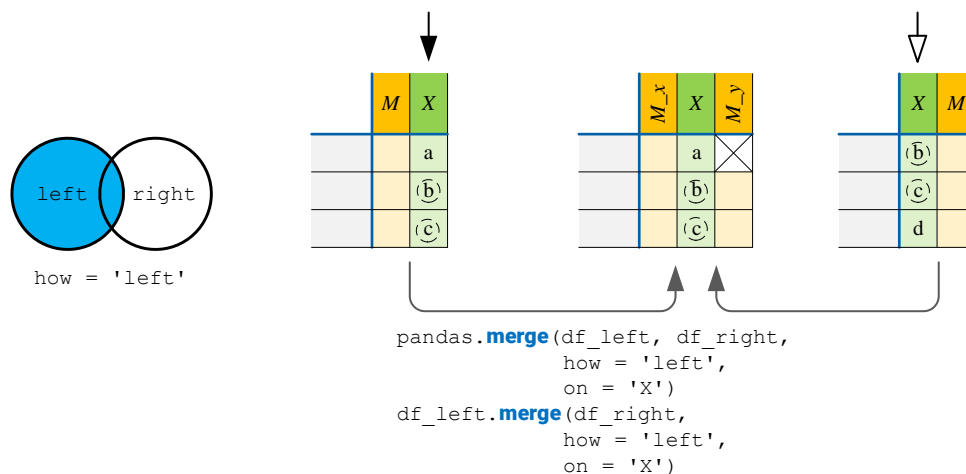


图 15. 利用 pandas.merge() 完成合并, how = 'left', 有列标签重名的情况

基于左右列合并

图 16 ~ 图 19 所示为 merge() 通过指定左右数据帧的列标签 (left_on 和 right_on) 完成合并。此外, merge() 还可以指定多个列标签进行合并操作。

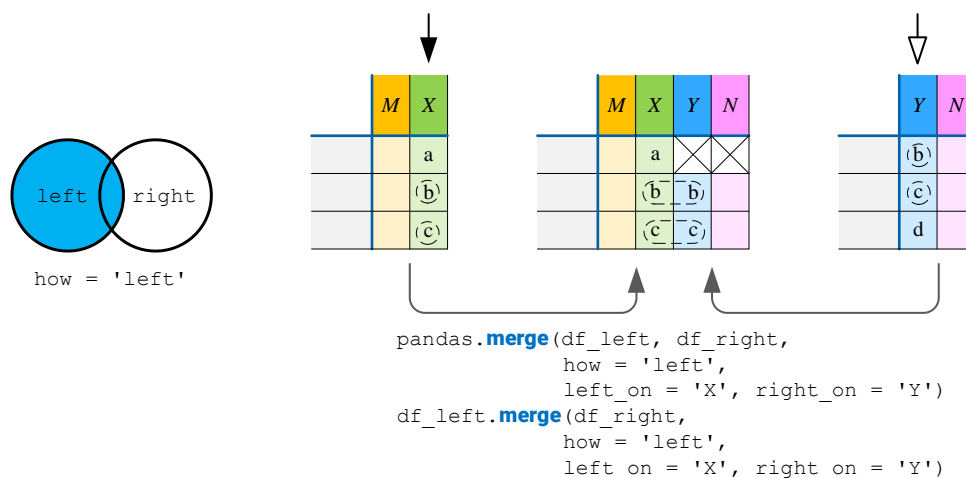


图 16. 利用 pandas.merge() 完成合并, how = 'left'

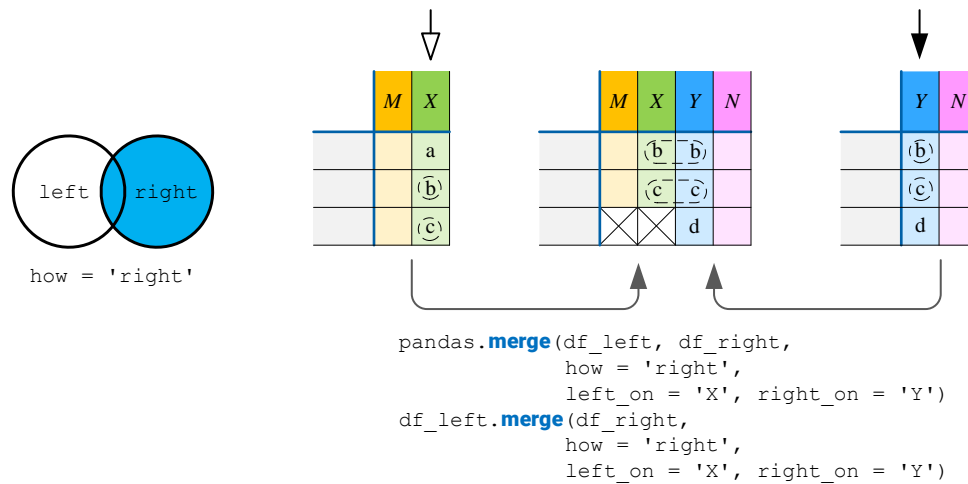


图 17. 利用 pandas.merge() 完成合并, how = 'right'

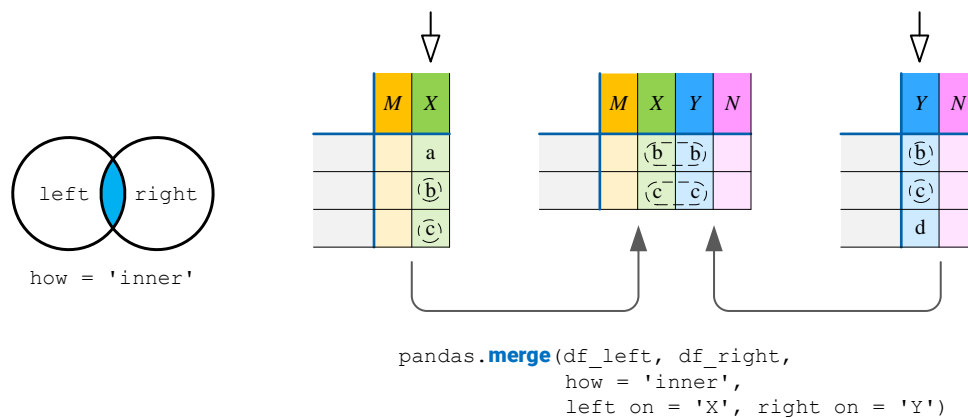


图 18. 利用 pandas.merge() 完成合并, how = 'inner'

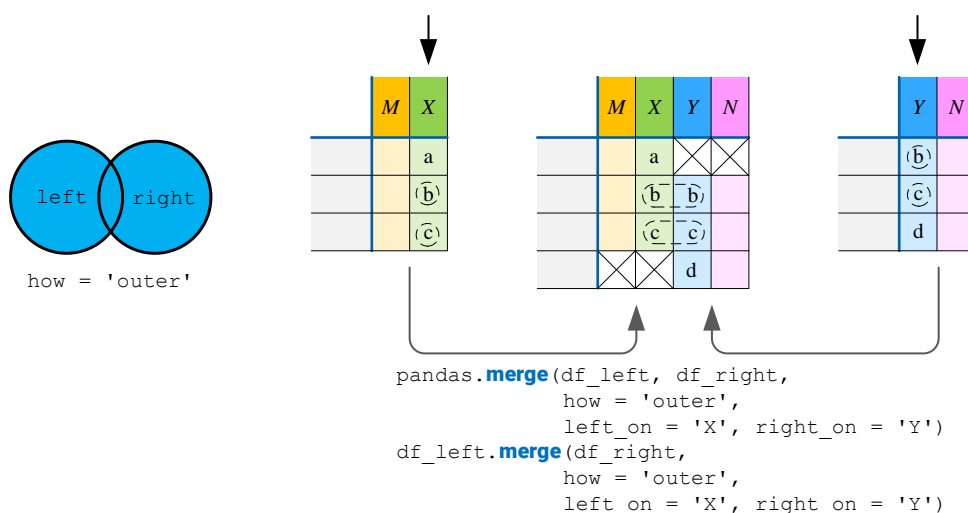


图 19. 利用 pandas.merge() 完成合并, how = 'outer'

独有

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

图 20 总结常用几种合并几何运算，`merge()` 可以直接完成前 5 种，目前 `merge()` 暂不直接支持剩下 3 种。这 3 种合并集合运算为：

左侧独有 (left exclusive)：只保留左侧 DataFrame 中存在，而右侧 DataFrame 中不存在的行。

右侧独有 (right exclusive)：只保留右侧 DataFrame 中存在，而左侧 DataFrame 中不存在的行。

全外独有 (full outer exclusive)：保留左侧 DataFrame 中不存在于右侧 DataFrame，同时右侧 DataFrame 中不存在于左侧 DataFrame 的行。

但是，我们可以利用 `merge()` 完成图 20，具体代码如图 21 所示。

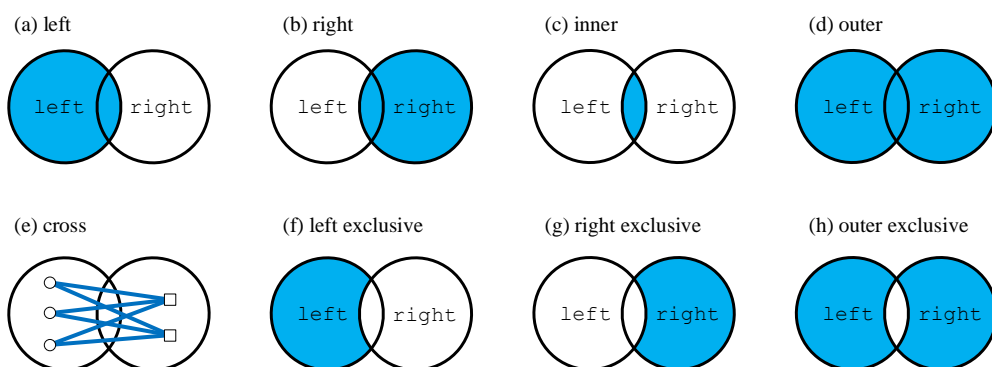


图 20. 总结常用合并集合运算

图 21 中的 **a** 首先利用 `merge()` 完成左连接合并。在 `pandas` 的 `merge()` 方法中，`indicator` 参数用于指定是否添加一个特殊的列，该列记录了每行的合并方式。这个特殊的列名可以通过 `indicator` 参数进行自定义，默认为 `"_merge"`。"`_merge`" 列可以取三个值：

"left_only": 表示该行只在左边的 DataFrame 中存在，即左连接中独有的行。

"right_only": 表示该行只在右边的 DataFrame 中存在，即右连接中独有的行。

"both": 表示该行在两个 DataFrame 中都存在，即连接方式中共有的行。

在 **b** 中，通过设定筛选条件，`left_exl['_merge'] == 'left_only'`，我们可以保留合并后的“左侧独有”行。结果如图 22 所示。

同理，**c** 完成右连接合并，**d** 通过设定筛选条件保留数据帧中“右侧独有”行，结果如图 23 所示。类似地，**e** 完成外连接合并，**f** 通过设定筛选条件保留“全外独有”行，结果如图 24 所示。

```

import pandas as pd
# 创建两个数据帧
left_data = {
    'M': [1, 2, 3],
    'X': ['a', 'b', 'c']}
left_df = pd.DataFrame(left_data)

right_data = {
    'X': ['b', 'c', 'd'],
    'N': [22, 33, 44]}
right_df = pd.DataFrame(right_data)

# LEFT EXCLUSIVE
a left_exl = left_df.merge(right_df,
                           on='X',
                           how='left',
                           indicator=True)

b left_exl = left_exl[
    left_exl['_merge'] == 'left_only'].drop(
    columns=['_merge'])

# RIGHT EXCLUSIVE
c right_exl = left_df.merge(right_df,
                           on='X',
                           how='right',
                           indicator=True)

d right_exl = right_exl[
    right_exl['_merge'] == 'right_only'].drop(
    columns=['_merge'])

# FULL OUTER EXCLUSIVE
e outer_exl = left_df.merge(right_df,
                           on='X',
                           how='outer',
                           indicator=True)

f outer_exl = outer_exl[
    outer_exl['_merge'] != 'both'].drop(
    columns=['_merge'])

```

图 21. 利用 merge() 完成左侧独有、右侧独有、全外独有；

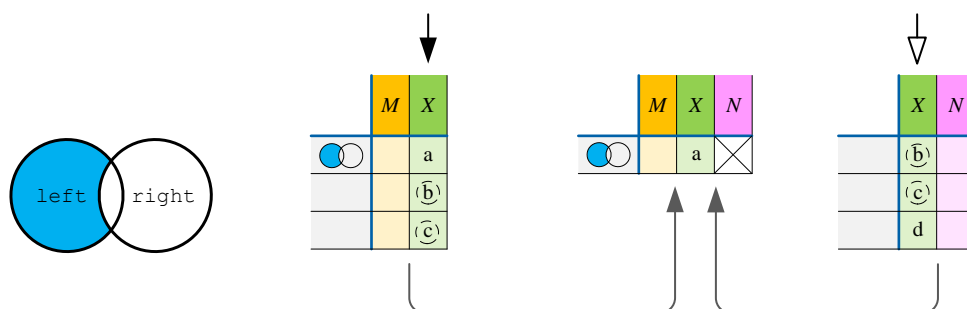
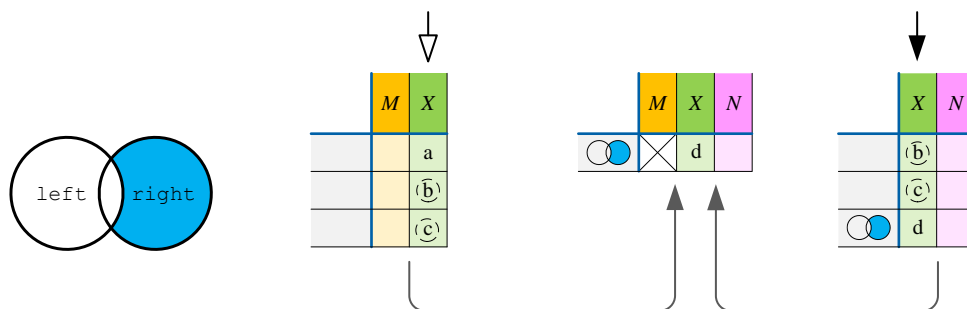
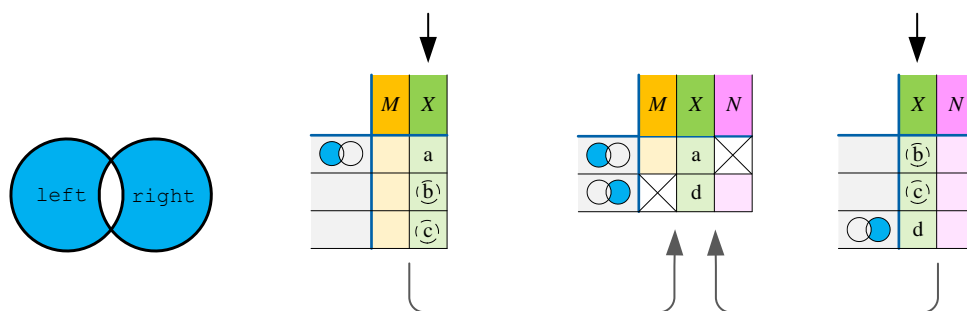


图 22. 利用 pandas.merge() 完成合并，左侧独有

图 23. 利用 `pandas.merge()` 完成合并，右侧独有图 24. 利用 `pandas.merge()` 完成合并，全外独有

22.5 长格式转换为宽格式：pivot()

`pivot()` 可以理解为一种长格式转换为宽格式的特殊情况。`pivot()` 需要指定三个参数：`index`、`columns` 和 `values`，它们分别代表新 `DataFrame` 的行索引、列索引和填充数据的值。

举个例子，图 25 左图表格为一个班级四名学生（学号分别为 1、2、3、4）的各科（Math、Art、Science）期中、期末成绩，这个表格就是所谓的长格式，相当于“流水账”。

图 25 右图则是期中考试成绩“矩阵”，行标签（`index`）为学生学号 'ID'，列标签（`columns`）为三门科目 'Subject'，数据（`values`）为期中考试成绩 'Midterm'。

由于每名学生仅仅选修两门科目，因此在图 25 右图中会看到 NaN。

进一步，图 25 右图数据帧横向求和，得到学生总成绩；而纵向求平均值，便是各科平均成绩。这是下一章要介绍的操作。

图 26 对应上述操作的代码。请大家自行提取同学各科期末考试成绩，科目为行标签，学号为列标签。

注意，使用 `pivot()` 时，必须指定 `index` 和 `columns`，这两列的值将用于创建新的行和列。

此外，请大家思考如果，如果参数 `values = ['Midterm', 'Final']`，结果会怎样？

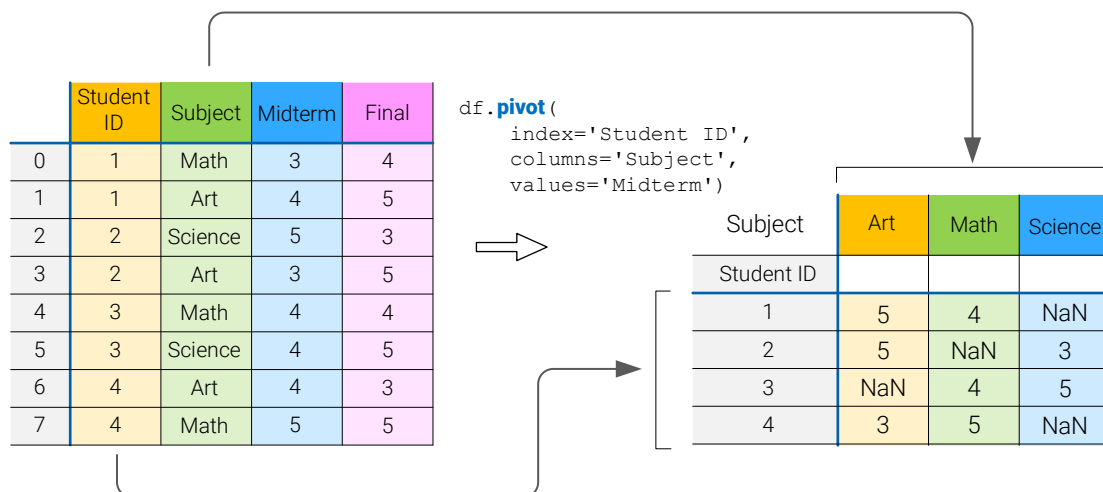


图 25. 利用 pivot() 提取学生各科期中考试成绩，学号为行标签，科目为列标签

```
import pandas as pd

data = {
    'Student ID': ['1', '1', '2', '2', '3', '3', '4', '4'],
    'Subject':    ['Math', 'Art', 'Science', 'Art',
                  'Math', 'Science', 'Art', 'Math'],
    'Midterm':    [4, 5, 3, 5, 4, 5, 3, 5],
    'Final':      [3, 4, 5, 3, 4, 4, 4, 5]}

df = pd.DataFrame(data)
df.pivot(index='Student ID',
         columns='Subject',
         values='Midterm')
```

图 26. 利用 pivot() 将长格式转换为宽格式，代码；

我们可以用 pivot_table() 完成和图 25 一样的操作，df.pivot_table(index='Student ID', columns = 'Subject', values='Midterm')。

和 pivot() 不同的是，pivot_table() 可以不用指定 columns。如图 27 所示。利用 pivot_table()，我们可以把数据帧学号、科目转化为双层行索引。

```
df.pivot_table(index=['Subject', 'Student ID'],
                values=['Midterm', 'Final'])
```

	Student ID	Subject	Midterm	Final
0	1	Math	3	4
1	1	Art	4	5
2	2	Science	5	3
3	2	Art	3	5
4	3	Math	4	4
5	3	Science	4	5
6	4	Art	4	3
7	4	Math	5	5

	Subject	Student ID	Final	Midterm
Art		1	4	5
		2	3	5
		4	4	3
Math		1	3	4
		3	4	4
		4	5	5
Science		2	5	3
		3	4	5

图 27. 利用 pivot_table() 将学号、科目转化为双层行索引

```
import pandas as pd

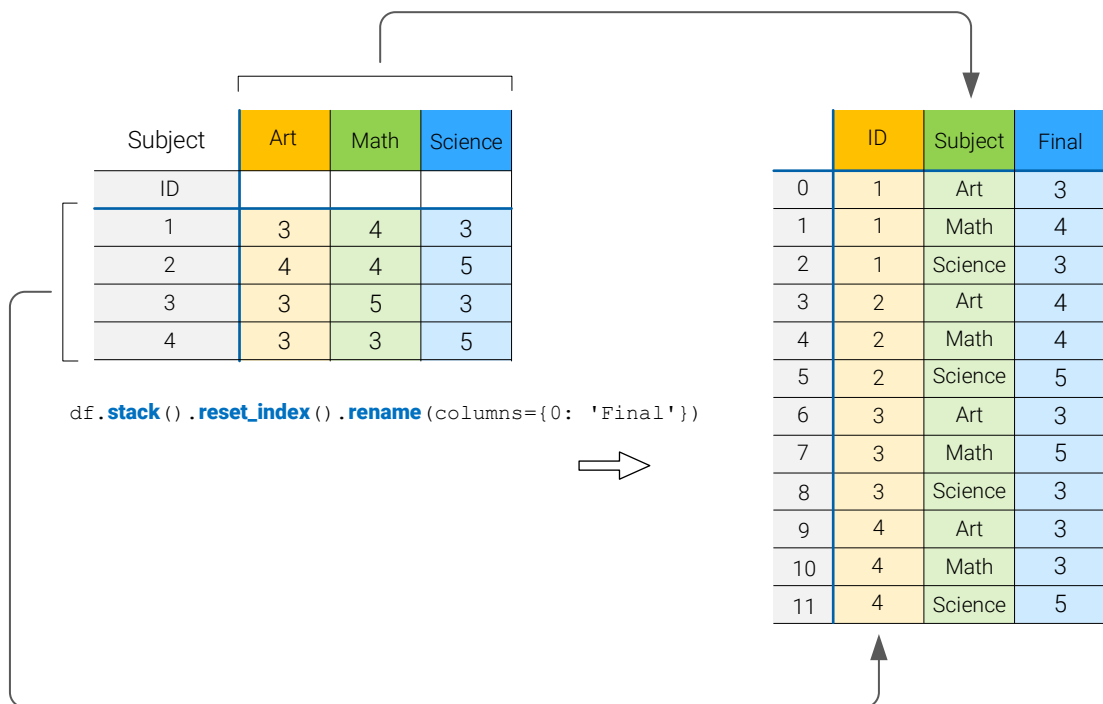
data = {
    'Student ID': ['1', '1', '2', '2',
                  '3', '3', '4', '4'],
    'Subject':    ['Math', 'Art', 'Science', 'Art',
                  'Math', 'Science', 'Art', 'Math'],
    'Midterm':    [4, 5, 3, 5, 4, 5, 3, 5],
    'Final':      [3, 4, 5, 3, 4, 4, 4, 5]}

df = pd.DataFrame(data)
df.pivot_table(index=['Subject', 'Student ID'],
                values=['Midterm', 'Final'])
```

图 28. 利用 pivot_table() 将长格式转换为宽格式，代码；

22.6 宽格式转换为长格式：stack()

方法 `stack()` 是一种将列逐级转换为层次化索引的操作。如果 `DataFrame` 的列是层次化索引，那么 `stack()` 会将最内层的列转换为最内层的索引。该函数返回一个 `Series` 或 `DataFrame`，具体取决于原始数据的维度。

图 29. 利用 `stack()` 将宽格式转换为长格式

```
import pandas as pd
import numpy as np

student_ids = [1, 2, 3, 4]
subjects = ['Art', 'Math', 'Science']
np.random.seed(0)
# 使用随机数生成成绩数据
scores = np.random.randint(3, 6,
                           size=(len(student_ids), len(subjects)))

# 创建数据帧
df = pd.DataFrame(scores, index=student_ids,
                  columns=subjects)

# 修改行列名称
df.columns.names = ['Subject']
df.index.names = ['Student ID']

# 将长格式转化为宽格式
a df.stack().reset_index().rename(columns={0: 'Final'})
```

图 30. 利用 `stack()` 将宽格式转换为长格式，代码；

`melt()` 将原始数据中的多列合并为一列，并根据其他列的值对新列进行重复。可以理解为 `stack()` 的一种泛化形式。`melt()` 需要指定 `id_vars` 参数，表示保持不变的列，同时还可以选择 `value_vars` 参数来指定哪些列需要被转换。请大家自行练习图 31 给出的示例。


```
import pandas as pd

data = {
    'Student ID': ['1', '2', '3', '4'],
    'Art': [4, 3, 5, 4],
    'Math': [3, 4, 5, 3],
    'Science': [5, 4, 3, 4]}

df = pd.DataFrame(data)
df.columns.names = ['Subject']
a melted_df = df.melt(id_vars='Student ID',
                      var_name='Subject',
                      value_vars=['Art', 'Math', 'Science'],
                      value_name='Score')
```

图 31. 利用 melt() 将宽格式转换为长格式，代码；

多层列标签

如果数据帧有多层列标签，可以有选择地选取特定级别列标签完成 stack() 操作。

数据帧中 A、B 代表两个班级，每个班级 Class 有 4 名同学（学号 1、2、3、4），这些同学都选了 3 门课程（Art、Math、Science）。数据帧的数据部分为同学们的期末成绩。

请大家思考如果采用 df.stack(level=["Subject"]), 结果会怎样？

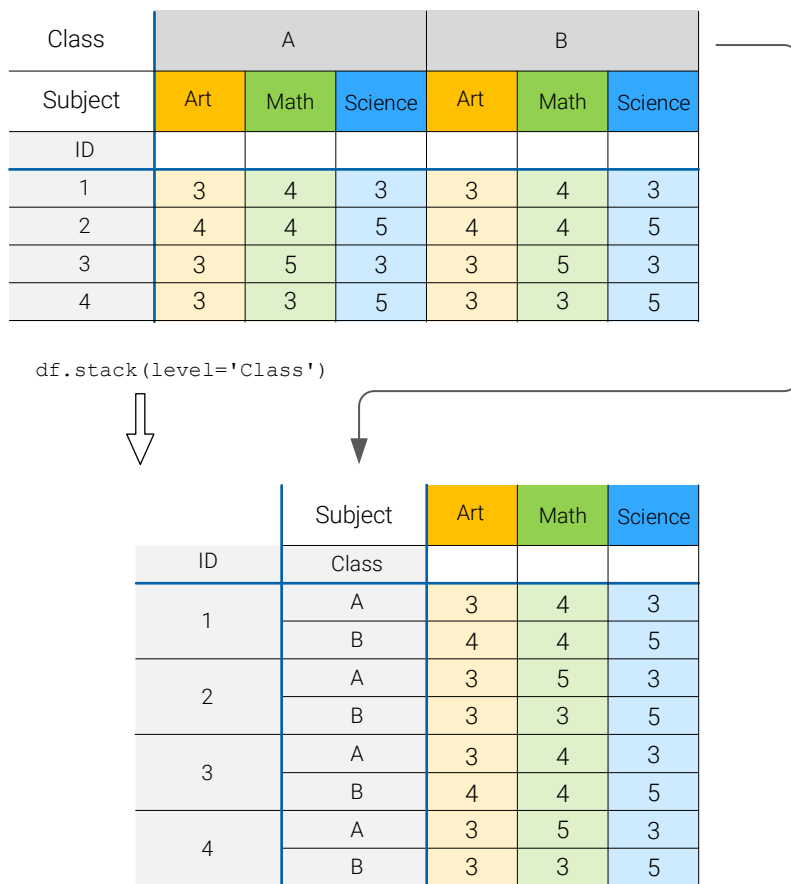
```
import pandas as pd

data = {
    ('A', 'Art'): [4, 3, 5, 4],
    ('A', 'Math'): [3, 4, 5, 3],
    ('A', 'Science'): [5, 4, 3, 4],
    ('B', 'Art'): [3, 4, 5, 4],
    ('B', 'Math'): [4, 5, 3, 3],
    ('B', 'Science'): [5, 3, 4, 5]}

# 创建多层行标签数据帧
df = pd.DataFrame(data, index=[1, 2, 3, 4])

# 添加行标签名称
df.columns.names = ['Class', 'Subject']
df.index.names = ['Student ID']
# 选择 'Class' 进行 stack() 操作
a stacked_df = df.stack(level='Class')
# stacked_df = df.stack(level=0)
```

图 32. 利用 stack() 将宽格式转换为长格式，选择特定列级别，代码；

图 33. 利用 `stack()` 将宽格式转换为长格式，选择特定列级别

22.7 长格式转换为宽格式: `unstack()`

在 Pandas 中，`unstack()` 是一个用于数据透视的方法，它用于将一个多级索引的 Series 或 DataFrame 中的其中选定级别转换为列。这在处理分层索引数据时非常有用。

如图 34 所示，左侧的数据帧 `df` 有 3 层行索引。第 0 层为 `Class`，第 1 层为 `Student ID`，第 2 层为 `Subject`。第 0 层 `Class` 有两个值 A、B，代表有两个班级。第 1 层 `Student ID` 有四个值 1、2、3、4，代表每个班级学生的学号。第 2 层有三个值 `Art`、`Math`、`Science`，代表三个科目。

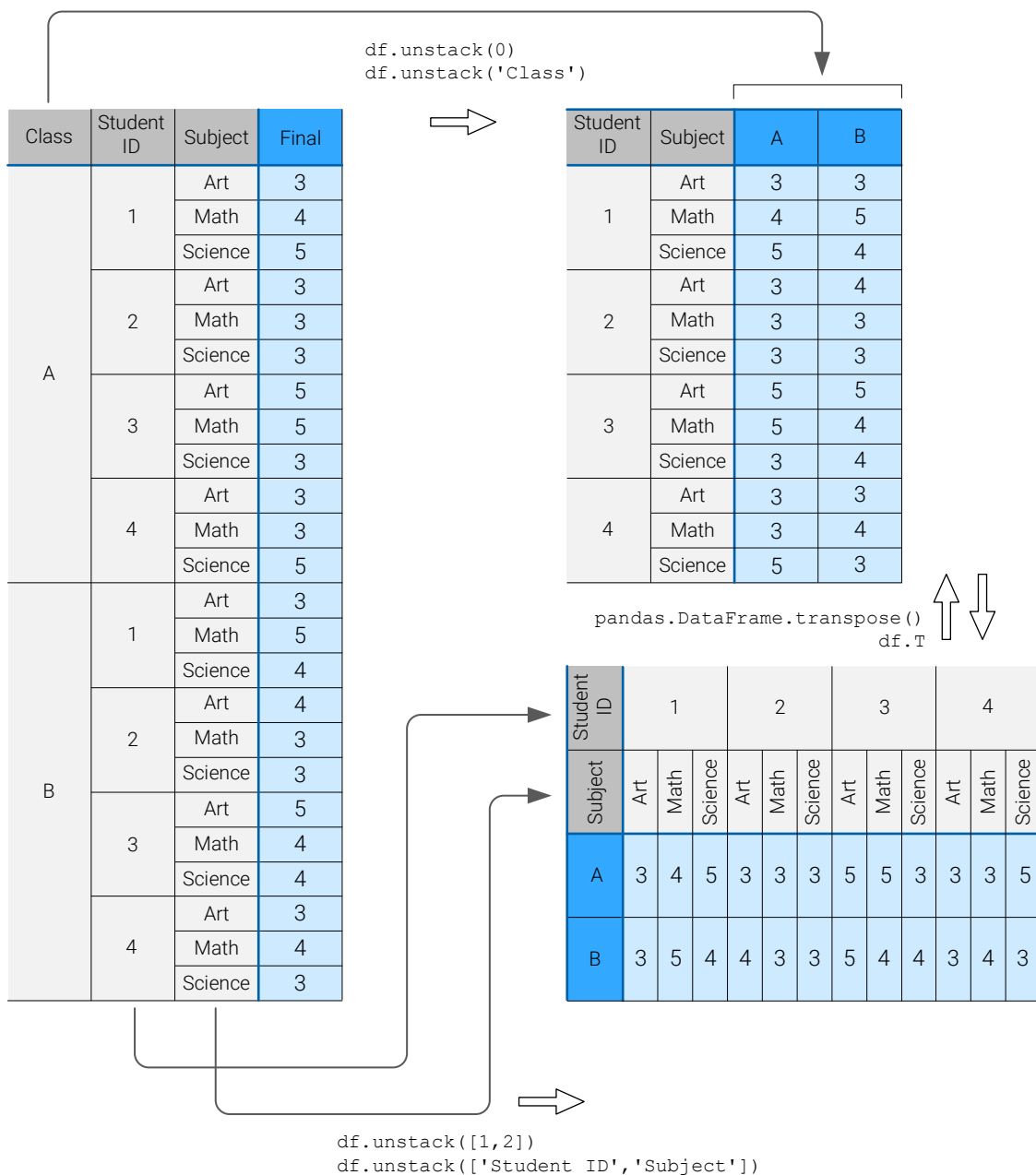


图 34. 利用 unstack() 将长格式转换为宽格式

`df.unstack(0)` 或 `df.unstack('Class')` 将第 0 层 Class 行索引转换成两列——A、B。请大家尝试，`df.unstack(1)`、`df.unstack('Student ID')`、`df.unstack(2)`、`df.unstack('Subject')`，并比较结果。

`df.unstack([1,2])` 或 `df.unstack(['Student ID', 'Subject'])` 将第 1、2 层行索引转换成两层列标签。请大家尝试 `df.unstack([2,1])` 或 `df.unstack(['Subject', 'Student ID'])`，以及尝试其他组合，比如 `[0, 2]`、`[2, 0]`、`[0, 1]`、`[1, 0]`，并比较结果。

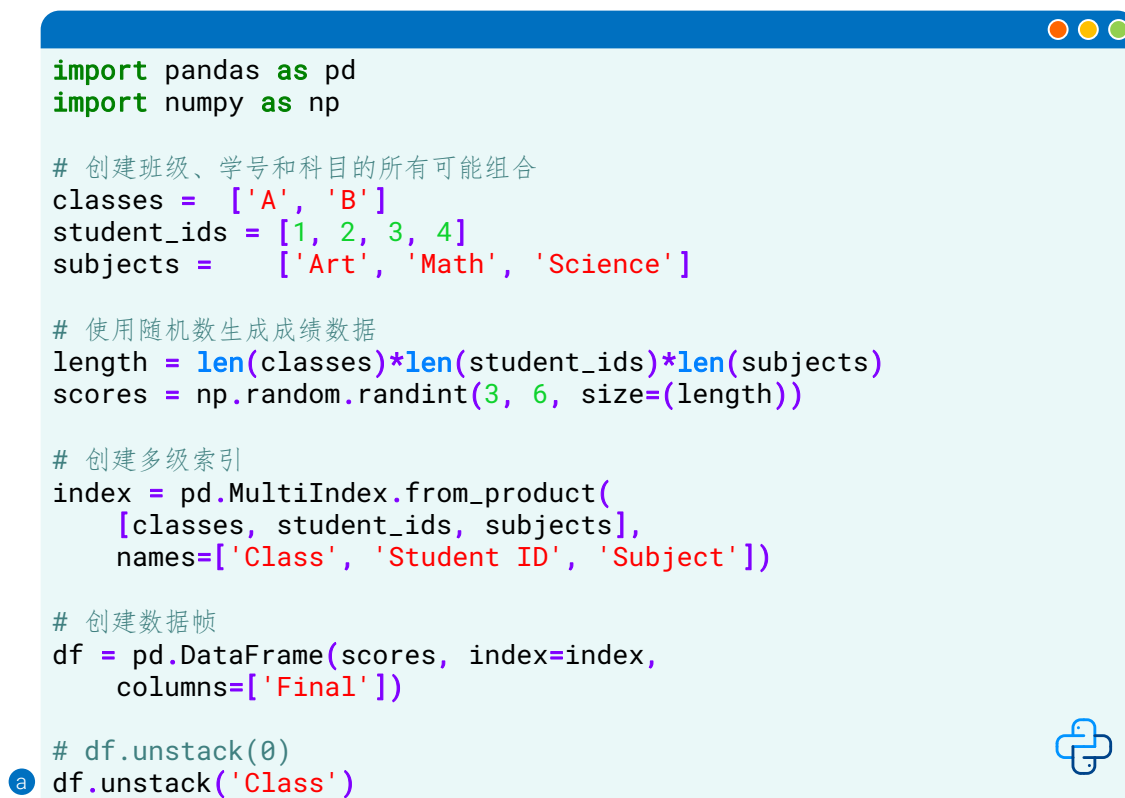


图 35. 利用 unstack() 将长格式转换为宽格式，代码；

22.8 分组聚合：groupby()

在 Pandas 中，`groupby()` 是一种非常有用的数据分组聚合计算方法。`groupby()` 按照某个或多个列的值对数据进行分组，然后对每个分组进行聚合操作。图 36 代码介绍如何使用 `groupby()` 方法，并结合 `mean()`、`std()`、`var()`、`cov()` 和 `corr()` 对分组后的数据进行聚合操作。

图 37、图 38 所示为考虑鸢尾花分类的协方差矩阵、相关性系数矩阵热图。其中，`groupby(['species']).cov()` 得到的数据帧为两层行索引。

根据前文介绍的多层行索引数据帧切片方法，`groupby_cov.loc['setosa']` 提取鸢尾花类别为 'setosa' 的协方差矩阵。也可以用 `groupby_cov.xs('setosa')` 提取相同数据。此外，我们也可以用 `iris_df.loc[iris_df['species'] == 'setosa'].cov()` 专门计算鸢尾花类别为 'setosa' 的协方差矩阵。

```
import seaborn as sns
import pandas as pd

iris_df = sns.load_dataset("iris")
# 从Seaborn中导入鸢尾花数据帧
# 分组计算统计量
a groupby_mean = iris_df.groupby(['species']).mean()
b groupby_std = iris_df.groupby(['species']).std()
c groupby_var = iris_df.groupby(['species']).var()
d groupby_cov = iris_df.groupby(['species']).cov()
e groupby_corr = iris_df.groupby(['species']).corr()
```

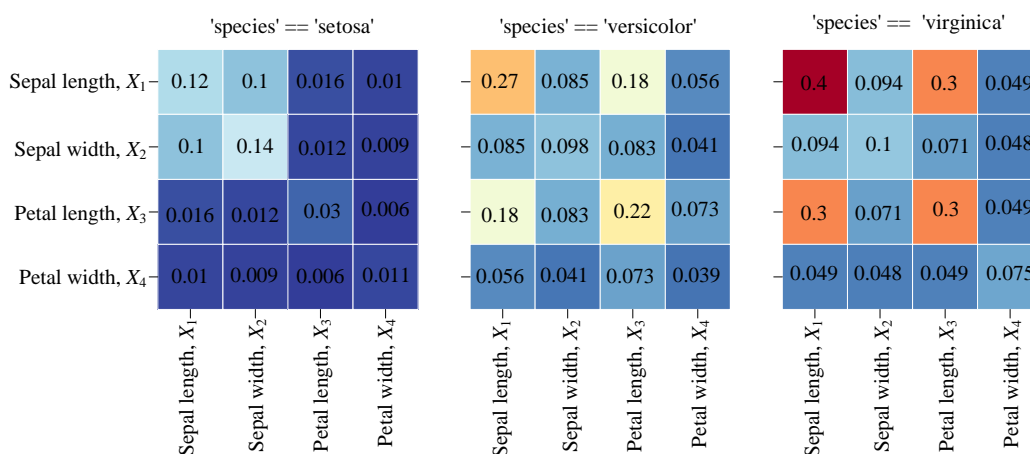
图 36. 鸢尾花数据帧 `groupby(['species'])` 计算统计量；

图 37. 协方差矩阵热图，考虑分类

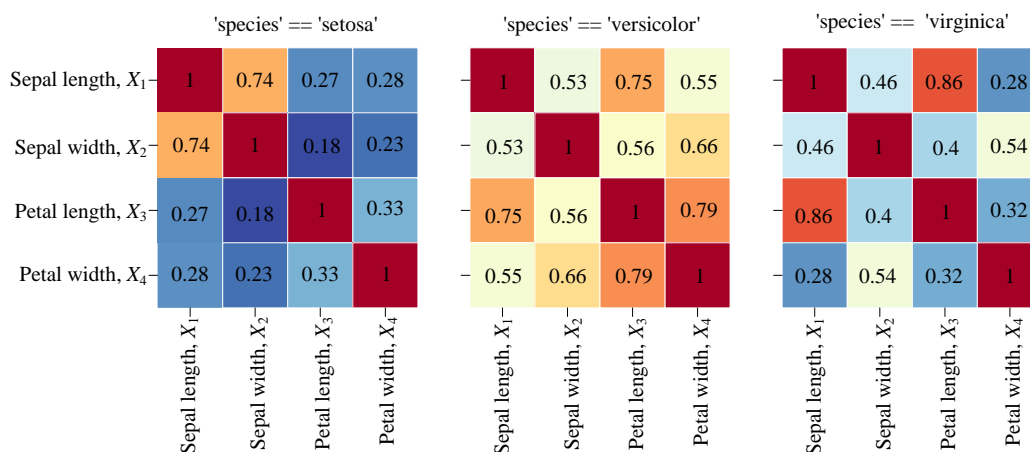


图 38. 相关性系数矩阵热图，考虑分类标签

下面介绍如何用 `groupby()` 汇总学生成绩。

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

```

import pandas as pd
import numpy as np

# 创建班级、学号和科目的所有可能组合
classes = ['A', 'B']
stu_ids = [1, 2, 3, 4]
subjects = ['Art', 'Math', 'Science']

# 使用随机数生成成绩数据
np.random.seed(0)
length = len(classes) * len(stu_ids) * len(subjects)
data = np.random.randint(3, 6, size=(length))

# 创建多行标签数据帧
index = pd.MultiIndex.from_product(
    [classes, stu_ids, subjects],
    names=['Class', 'Student ID', 'Subject'])
df = pd.DataFrame(data, index=index, columns=['Score'])

# 1) 每个班级各个科目平均成绩
a class_subject_avg = df.groupby(
    ['Class', 'Subject'])['Score'].mean()

# 2) 每个班级各个学生的平均成绩
b class_student_avg = df.groupby(
    ['Class', 'Student ID'])['Score'].mean()

# 3) 两个班级放在一起各个科目平均成绩
c both_class_avg = df.groupby(
    'Subject')['Score'].mean()

# 4) 两个班级每个同学总成绩，并排名
d student_total_score = df.groupby(
    ['Class', 'Student ID'])['Score'].sum().sort_values(
    ascending=False)

```



图 39. 利用 groupby() 汇总学生成绩，代码；

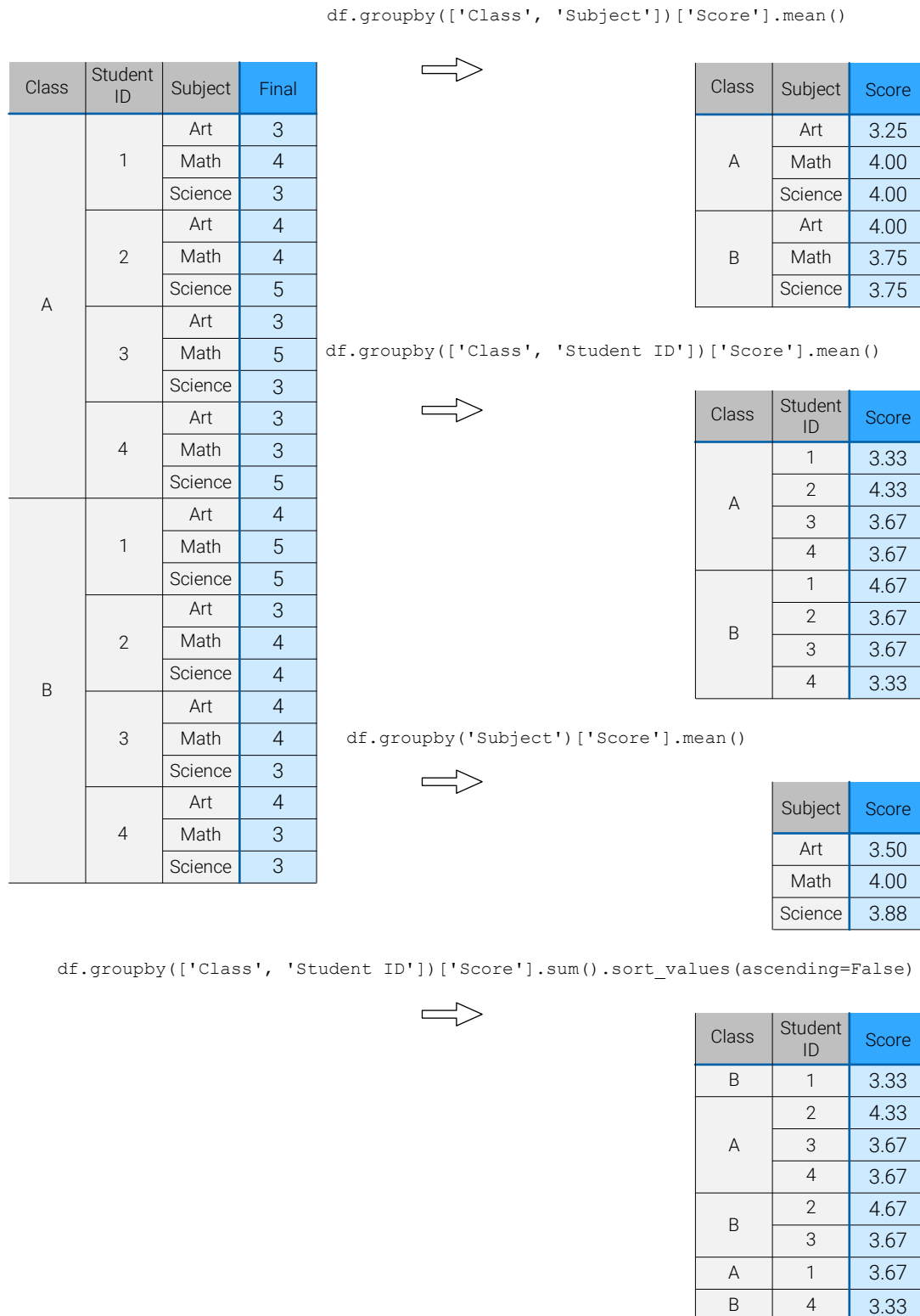
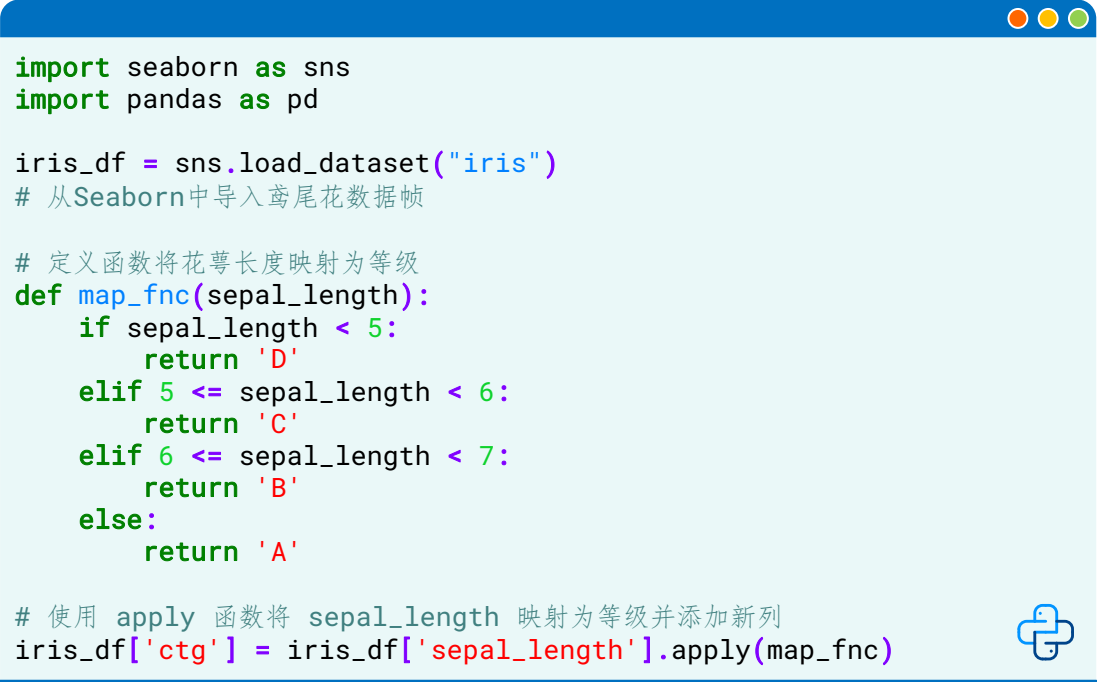


图 40. 利用 groupby() 汇总学生成绩

22.9 自定义操作：apply()

在 Pandas 中，可以使用 `apply()` 方法对 `DataFrame` 的行或列进行自定义函数的运算。`apply()` 方法是 Pandas 中最重要和最有用的方法之一，它可以实现 `DataFrame` 数据的处理和转换，也可以实现计算和数据清洗等功能。

如图 41 代码所示，^a 定义函数 `map_fnc()`，这个函数的目的是将花萼长度 `sepal_length` 转化为等级。转化的规则为，如果 `sepal_length < 5`，等级为 D；如果 `5 <= sepal_length < 6`，等级为 C；如果 `6 <= sepal_length < 7`，等级为 B；其余情况（`sepal_length > 6`），等级为 A。^b 利用 `apply()` 将自定义函数用在数据帧 `iris_df['sepal_length']` 上。



```
import seaborn as sns
import pandas as pd

iris_df = sns.load_dataset("iris")
# 从Seaborn中导入鸢尾花数据帧

# 定义函数将花萼长度映射为等级
def map_fnc(sepal_length):
    if sepal_length < 5:
        return 'D'
    elif 5 <= sepal_length < 6:
        return 'C'
    elif 6 <= sepal_length < 7:
        return 'B'
    else:
        return 'A'

# 使用 apply 函数将 sepal_length 映射为等级并添加新列
iris_df['ctg'] = iris_df['sepal_length'].apply(map_fnc)
```

图 41. 鸢尾花数据帧使用 `apply()` 自定义函数，对于特定一列；

`apply()` 方法可以接受一个函数作为参数，这个函数将会被应用到 `DataFrame` 的每一行或每一列上。这个函数可以是 Pandas 中已经定义好的函数，可以是自定义函数，也可以是匿名 `lambda` 函数。

比如，图 42 代码使用 `apply()` 和 `lambda` 函数计算鸢尾花数据集中每个类别中最小的花瓣宽度。

^a 等价于 `iris_df.groupby('species')['sepal_length'].min()`。

图 43 中 `apply()` 的输入先是匿名 `lambda` 函数，对象定义为 `row`，代表数据帧的每一行。而 `lambda` 函数调用自定义函数 `map_petal_width()`，这个函数有两个输入。

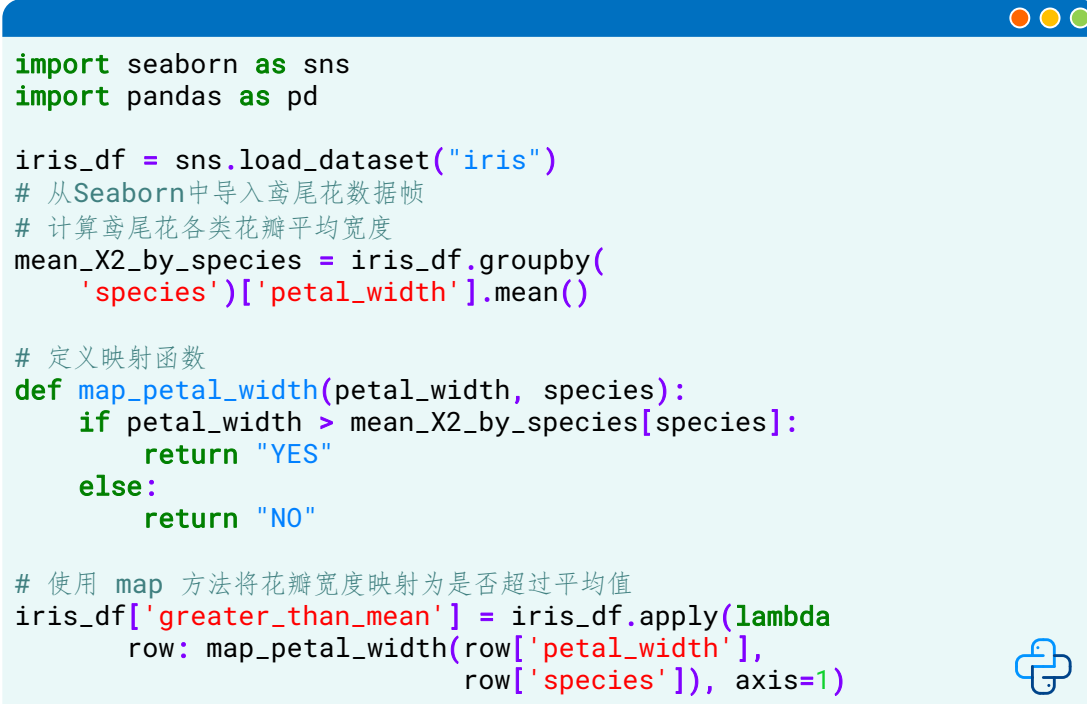


```
import seaborn as sns
import pandas as pd

iris_df = sns.load_dataset("iris")
# 从Seaborn中导入鸢尾花数据帧

# 使用apply() 和lambda函数计算每个类别中最小的花瓣宽度
iris_df.groupby('species')['sepal_length'].apply(
    lambda x: x.min())
# iris_df.groupby('species')['sepal_length'].min()
```

图 42. 鸢尾花数据帧使用 apply() 匿名 lambda 函数，对于特定一列；



```
import seaborn as sns
import pandas as pd

iris_df = sns.load_dataset("iris")
# 从Seaborn中导入鸢尾花数据帧
# 计算鸢尾花各类花瓣平均宽度
mean_X2_by_species = iris_df.groupby(
    'species')['petal_width'].mean()

# 定义映射函数
def map_petal_width(petal_width, species):
    if petal_width > mean_X2_by_species[species]:
        return "YES"
    else:
        return "NO"

# 使用 map 方法将花瓣宽度映射为是否超过平均值
iris_df['greater_than_mean'] = iris_df.apply(lambda
    row: map_petal_width(row['petal_width'],
        row['species']), axis=1)
```

图 43. 鸢尾花数据帧使用 apply() 匿名 lambda 函数，对于特定两列；

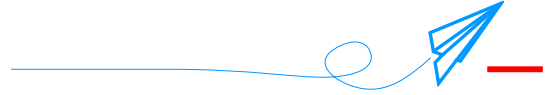
此外，在 Pandas 中，可以使用 map() 方法对 Series 或 DataFrame 特定列进行自定义函数的运算。这个映射关系可以由用户自己定义，也可以使用 Pandas 中已经定义好的函数。

除了 apply() 和 map() 方法之外，Pandas DataFrame 还提供 applymap()、transform() 等方法，请大家自行学习使用。需要大家注意，applymap() 用于对 DataFrame 中的每个元素应用同一个函数，返回一个新的 DataFrame。



Pandas 中重塑和透视操作灵活多样，本章介绍的方法仅仅是冰山一角而已。实践中，大家可以根据需求自行学习使用其他方法操作，建议大家继续阅读如下链接。

https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html



下两章将利用本章介绍的一些方法，结合 Plotly 库中可视化函数，让大家看到两者结合后“讲故事”的力量！