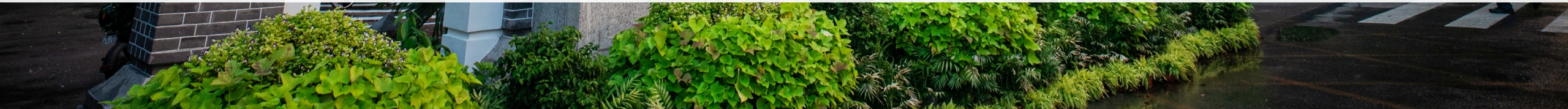
清华大学数据库技术与应用

Pandas 第三部分

授课教师：计算机系王健楠

授课学期：2026年（春季）



本讲议程

清華大學 Tsinghua University

1

分组

groupby 与聚合操作的核心框架

2

数据透视表

数据重塑与多维分析

3

表连接

merge 与 join 操作详解

为什么要分组?

在数据科学中，分组（Grouping）是一种强大的工具，用于发现数据中的模式和趋势。

- **归类：**将同一类别（如同一院系、同一年份）的数据行归并在一个组中。
- **聚合：**对每个组执行统计操作（如求和、平均值），将大量细节数据压缩为关键指标。

价值所在

分组让我们能够从“微观”的个体数据（如单个学生成绩）上升到“宏观”的群体洞察（如各院系教学质量对比）。

分组过程图解：按院系统计平均分

姓名	院系	成绩
张三	计算机	95
李四	数学系	88
王五	计算机	92
赵六	数学系	90

↓

院系	平均成绩
计算机	93.5
数学系	89.0

groupby() 基础概念

Split - Apply - Combine

.groupby() 操作涉及拆分对象、应用函数和合并结果的组合过程。

Split（拆分）：
根据提供的键（如“院系”）将数据拆分为多个独立的组。

DataFrameGroupBy 对象： 调用 `.groupby()` 后生成的是一个中间对象。它在逻辑上保留了完整的 DataFrame 结构，但内部建立了分组索引映射。

原始 DataFrame (students)

姓名	院系	成绩
张三	计算机	95
李四	数学系	88
王五	计算机	92
赵六	数学系	90

`.groupby("院系")`

↓

DataFrameGroupBy Object (逻辑视图)

姓名	院系		成绩
张三	计算机	GROUP 1	95
王五	计算机	GROUP 1	92
李四	数学系	GROUP 2	88
赵六	数学系	GROUP 2	90

本质上仍是表格数据，但已按 Key 建立了分组映射

.groupby().agg() 数据流转

从原始 DataFrame 到最终聚合结果的完整流程解析

原始 DataFrame

院系	姓名	成绩
计算机	张三	95
数学系	李四	88
计算机	王五	92
数学系	赵六	90

`.groupby("院系")` →

GroupBy 对象 (中间状态)

DataFrameGroupBy Object

院系	姓名	成绩
Grp1 计算机	张三	95
Grp1 计算机	王五	92
Grp2 数学系	李四	88
Grp2 数学系	赵六	90

逻辑分组视图：保留完整结构，建立分组索引

`.agg("mean")` →

结果 DataFrame

院系	成绩
计算机	93.5
数学系	89.0

思考：非数值列去哪里了？

在原始数据中，"姓名"列包含文本数据（张三、李四...）。
为什么在最终的结果 DataFrame 中，"姓名"列消失了？

把前面的内容整合起来

数据处理全流程回顾

1. 原始 DataFrame

院系	姓名	成绩
计算机	张三	95
计算机	王五	92
数学系	李四	88
数学系	赵六	90

`.groupby("院系")` →

2. GroupBy Object

Grp1: 计算机

计算机	张三	95
计算机	王五	92

Grp2: 数学系

数学系	李四	88
数学系	赵六	90

`.agg("mean")` →

3. 结果 DataFrame

院系	成绩
计算机	93.5
数学系	89.0

标准语法公式

将分组与聚合组合成一行代码：

`df.groupby(列名).agg(函数)`

实际代码示例

Python

```python
students[["院系", "成绩"]].groupby("院系").agg("mean")
```

聚合函数语法详解

清華大學 Tsinghua University

.agg() 括号里到底能填什么?

Python 内置函数

如 `sum`, `max`。简单但处理大数据慢。

NumPy 函数

如 `np.mean`。数值计算极快，不可直接用于字符串。

Pandas 字符串别名

如 `"mean"`。**最推荐**，自动处理空值且优化最好。

常用写法对照表

功能	Python / NumPy 写法	Pandas 字符串写法 推荐
求和	`.agg(sum)` `.agg(np.sum)`	`.agg("sum")`
平均值	`.agg(np.mean)`	`.agg("mean")`
最大值	`.agg(max)` `.agg(np.max)`	`.agg("max")`
首位值	*(无直接简单对应)*	`.agg("first")`
计数	`.agg(len)` `.agg(np.size)`	`.agg("count")`

为什么推荐字符串形式?

例如 `.agg("mean")` 不仅代码简洁，底层还针对 Pandas 数据结构进行了 Cython 级别的性能优化，并能智能忽略 NaN 缺失值。

等价写法

目标：统计各课程平均成绩

Python

```python
# 方式 A（推荐）：先选列，再 agg
res1 = students.groupby("课程")[["成绩"]].agg("mean")

# 方式 B（简洁）：先选列，直接调用方法
res2 = students.groupby("课程")[["成绩"]].mean()

# 方式 C（变体）：对所有列操作，忽略非数值列
res3 = students.groupby("课程").mean(numeric_only=True)

print(res1.equals(res2)) # → True
```

课程	成绩
数据结构	88.5
线性代数	92.0
计算机网络	85.4

提示： 只要结果符合需求，选择团队一致的代码风格即可。

groupby 总结

Split - Apply - Combine 框架

Split (拆分)

根据一个或多个键（Key）将数据表拆分为若干个子 DataFrame（组）。

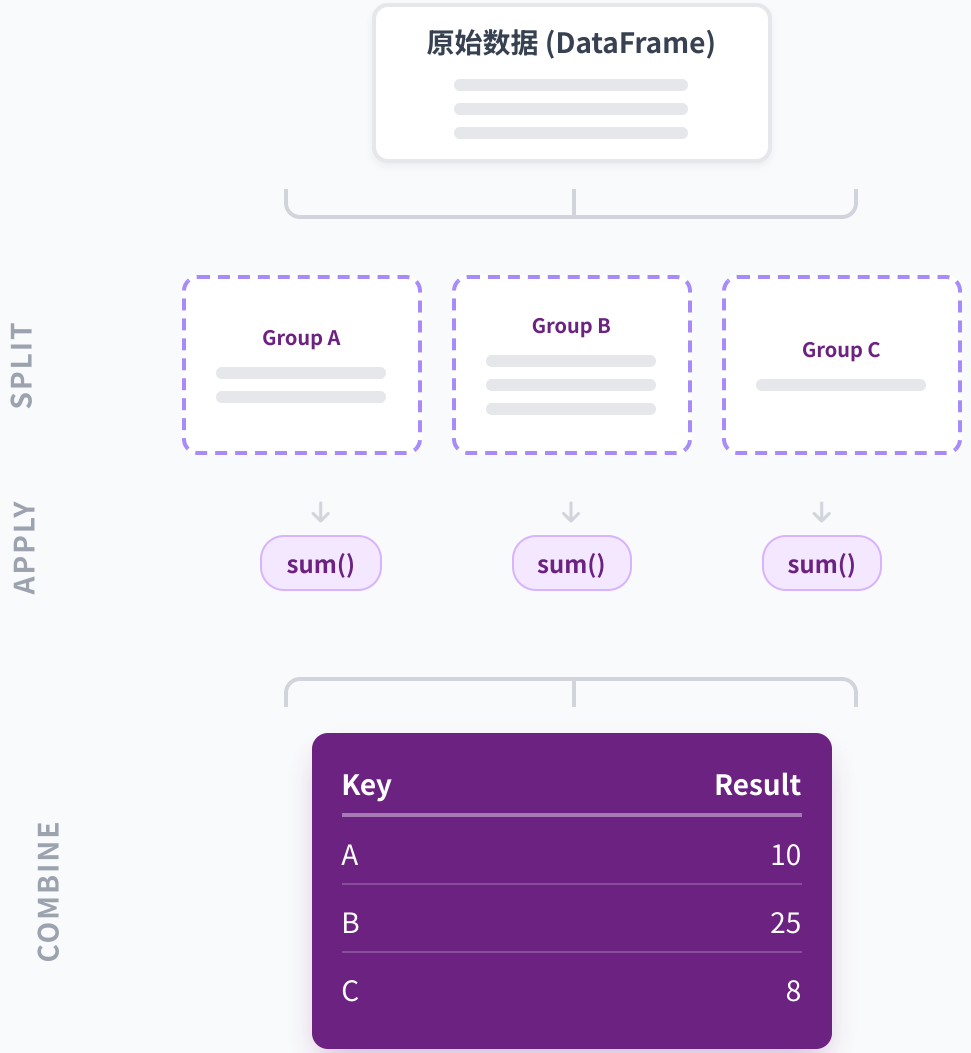
Apply (应用)

对每个组独立应用函数。最常见的是聚合（Aggregation），如求和、平均值。

Combine (合并)

将每个组的计算结果重新组合成一个新的数据结构（DataFrame 或 Series）。

```
df.groupby("Key") # Split
  .agg("sum")       # Apply
  # Combine （自动完成）
```



课堂小测试

原始数据 (df)

班级	分数	备注
A	83	ak
A	91	ak
A	88	hi
B	75	tx
A	79	ca

```
# 目标：对“班级”列分组并求最大值
result = df.groupby("班级").agg(max)
```

班级 A 的结果行是什么?

执行代码后，在结果 DataFrame 中，**班级 A** 对应的 **[分数]** 和 **[备注]** 分别是多少?

选项 A

91, "ak"

来自最高分那一行

选项 B

91, "hi"

独立最大值

选项 C

88, "hi"

来自最大备注那一行

选项 D

TypeError

无法对字符串求最大值

答案与解析

课堂测试回顾

回顾第10页的问题：对“班级”列分组并求最大值，班级 A 的结果行是什么？

正确答案

选项 B：91, "hi"

核心要点

Pandas 的聚合操作是**按列独立处理**的，每列单独计算最大值，因此结果可能来自原始数据中的不同行（导致“数据错位”）。

1. 筛选出班级 A 的原始数据

班级	分数	备注
A	83	ak
A	91	ak
A	88	hi
A	79	ca

2. 按列独立求最大值 (.max)

分数列

83
91
88
79

Max: 91

备注列 (字母顺序)

ak
ak
hi
ca

Max: "hi"

冗余列处理说明

什么是“干扰列”问题?

在执行 .agg() 时，如果聚合函数不能应用于所有列（例如尝试对“姓名”或“院系”字符串计算数值平均值），Pandas 可能会：

潜在风险

1. 抛出 **TypeError** 异常（在较新版本中更常见）。
2. 产生**被自动忽略的结果**（默默过滤掉非数值列，可能让你误以为操作成功了，但容易忽略逻辑错误）。

最佳实践

显式选列： 在调用 .agg() 之前，显式选择需要聚合的列。这不仅能避免警告，还能防止数据意外丢失或产生逻辑错误。

Nuisance_Columns.py

```
# ❌ 错误示范：隐式聚合所有列
# 这会尝试对 "姓名" 等所有列应用 mean，可能导致报错
bad_result = students.groupby("课程").agg("mean")
# 结果：可能报错 TypeError: Could not convert string to float

# ------------------------------------------------

# ✅ 推荐做法：先选列，后聚合
# 明确只对 "成绩" 列计算平均值
avg_table = (
    students
    .groupby("课程")[["成绩"]] # 关键步骤：显式选择数值列
    .agg("mean")
)
```

清洗后的正确结果 (avg_table)：

课程 (Index)	成绩 (Mean)
数据结构	88.5
微积分 A(1)	79.0

重命名聚合后的列（提升可读性）

清華大學 Tsinghua University

为什么要重命名？

默认情况下，.groupby 聚合操作不会更改列名。

例如，对“成绩”列计算平均值后，列名仍然是“成绩”，这会造成语义歧义。

> **清晰度原则**
>
> 列名应准确反映数据含义。将“成绩”改为“平均成绩”可以避免误解，明确该列是统计结果。

两种常用方法

1. **链式调用 .rename()**
   在聚合操作后直接修改列名。
2. **命名聚合 (Named Aggregation)**
   在 .agg() 中直接指定新列名和聚合逻辑。

Python

```python
# 方法一：聚合后使用 .rename()
avg_table = (
students.groupby("课程")[["成绩"]]
.agg("mean")
.rename(columns={"成绩": "平均成绩"})
)

# 方法二：命名聚合 (Named Aggregation) - 推荐写法
avg_table = (
students.groupby("课程")
.agg(平均成绩=("成绩", "mean"))
)
print(avg_table.head(3))
```

>_ 输出结果 (DataFrame)：

课程 (Index)	平均成绩
数据结构	88.5
操作系统	90.2
计算机网络	85.3

GroupBy 对象与其他方法

Raw GroupBy Objects

groupby 操作的结果是一个 `DataFrameGroupBy` 对象。

注意：它不是一个 DataFrame!

```
grouped_by_year = elections.groupby("Year")
type(grouped_by_year)
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

核心理解

给定一个 DataFrameGroupBy 对象，我们可以使用多种函数来生成新的 DataFrames（或 Series）。

`agg()` 只是众多选择中的一种。

常用 GroupBy 方法列表：

```
df.groupby(col).mean()
```

```
df.groupby(col).sum()
```

```
df.groupby(col).min()
```

```
df.groupby(col).max()
```

```
df.groupby(col).first()
```

```
df.groupby(col).last()
```

```
df.groupby(col).size()
```

❓ 区别是什么?

```
df.groupby(col).count()
```

❓ 区别是什么?

```
df.groupby(col).filter()
```

groupby.size() vs groupby.count()

清華大學 Tsinghua University

一句话总结

size() 计数每组有多少**行**（不管是否有数据缺失）；
count() 计数每组每列有多少**非空数据**（排除 NaN）。

代码示例

```
students.groupby("院系").size()  # 对应右侧蓝色结果表
students.groupby("院系").count()  # 对应右侧粉色结果表
```

1. 原始数据 (含 NAN)

院系	姓名	成绩
计算机	张三	95
计算机	*NaN*	88
数学系	李四	*NaN*
数学系	赵六	90

2. 结果对比

.size() (计数行数)

院系	Size
计算机	2
数学系	2

注：无论列中是否有NaN，只统计行数

.count() (计数非空值)

院系	姓名	成绩
计算机	1	2
数学系	2	1

注：分别计算每列非空值数量

按组过滤 (Filtering by Group)

工作原理

groupby.filter() 允许我们根据分组后的整体属性来筛选数据，而不是基于单个行。

核心流程

1. **Split**
   将 DataFrame 按键分组。
2. **Apply Filter**
   对每组应用函数，返回 True/False。
3. **Combine**
   保留返回 True 的组的所有行，丢弃 False 的组。

1 原始数据 DataFrame (4行 × 3列)

课程	学生	成绩
数据结构	张三	95
数据结构	李四	88
微积分	王五	70
微积分	赵六	85

↓

2 分组计算：保留总分 > 180 的组

课程：数据结构 ✓

张三	95
李四	88

`Sum: 183 > 180 (True)`

课程：微积分 ✗

王五	70
赵六	85

`Sum: 155 < 180 (False)`

↓ `.filter(lambda x: x['成绩'].sum() > 180)`

3 过滤结果：保留整组的所有行（包含学生列）

课程	学生	成绩
数据结构	张三	95
数据结构	李四	88

groupby.filter() 示例

常见过滤模式

1. **基于组大小过滤**
   例如：仅保留选课人数超过 100 人的热门课程。

2. **基于聚合属性过滤**
   例如：仅保留平均分高于 85 分的绩优课程。

3. **综合复杂条件**
   同时满足人数和成绩要求的课程。

Python

```python
# 示例 1: 保留选课人数 ≥ 100 的课程
popular = students.groupby("课程").filter(lambda sf: len(sf) ≥ 100)

# 示例 2: 保留平均分 > 85 的课程（sf 代表 sub-frame）
high_score = students.groupby("课程").filter(lambda sf: sf["成绩"].mean()
> 85)

# 示例 3: 综合条件（人数 ≥ 80 且 平均分 > 82）
elite_courses = students.groupby("课程").filter(
lambda sf: (len(sf) ≥ 80) and (sf["成绩"].mean() > 82)
)
```

groupby.filter() 示例： 低分课程预警案例

清華大學 Tsinghua University

应用场景：教学预警

在教务管理中，我们需要快速识别出“教学效果异常”的课程。例如，如果某门课程的所有学生最高分都低于 60 分，这可能意味着试卷难度过大或教学环节存在问题。

目标

筛选出所有“最高分 < 60”的课程组，生成预警清单。

操作步骤

1. 按课程分组：将学生成绩按"课程"进行拆分。
2. 应用过滤条件：对每组检查max() < 60。
3. 查看结果：对筛选出的“问题课程”进行统计汇总。

Python

```python
# 1. 使用 filter 保留"全班最高分都不到60分"的课程记录
low_perf_courses = students.groupby("课程").filter(
    lambda sf: sf["成绩"].max() < 60
)

# 2. 验证结果：按课程汇总统计（计数、最高分、平均分）
summary = low_perf_courses.groupby("课程")["成绩"] \
    .agg(["count", "max", "mean"]) \
    .sort_values("mean")

print(summary.head())
```

输出结果（预警清单）：

课程	count	max	mean
高维拓扑学	15	58	42.5
量子场论进阶	22	55	45.0
古希腊语语法	8	59	51.2

课堂小测试：如何找到每个院系平均分最高的学期？

原始数据 (students) 需提取粉色行

院系	年份	学期	平均分
计算机	2023	秋季	88
计算机	**2024**	**春季**	**95**
计算机	2022	春季	91
数学系	**2022**	**秋季**	**92**
数学系	2023	春季	85

↓

期望输出 完整保留整行信息

院系	年份	学期	平均分
计算机	**2024**	**春季**	**95**
数学系	**2022**	**秋季**	**92**

哪段代码可以正确实现左侧的数据转换？

选项 A

```
students.groupby("院系").max()
```

选项 B

```
students.sort_values("平均分",
ascending=False)
        .groupby("院系").first()
```

选项 C

```
students.groupby("院系").agg(lambda x:
x.iloc[0])
```

选项 D

```
students.groupby("院系")["平均分"].max()
```

答案与解析

为什么选项 A 错误?

`groupby().max()` 会对每一列独立计算最大值。

结果中的“学期”是字母顺序最后的学期，而“成绩”是数值最高的成绩——它们可能来自不同的行！

正确答案：选项 B

先按成绩降序排序，然后对每个院系分组并取第一行（first）。

sort_values + groupby + first

其他选项问题

选项 C: `agg(lambda x: x.iloc[0])` 默认未排序，取到的是随机或索引顺序的第一行。

选项 D: 仅返回了最大成绩，丢失了“学期”和“课程”等关联信息。

其他等价方法：lambda、idxmax、drop_duplicates

多种实现路径

在 Pandas 中，条条大路通罗马。除了“排序+First”方法外，还有以下三种常见方式来获取分组中的最佳行。

方法 A: Lambda 函数

利用 agg 和自定义 lambda。灵活但速度较慢。需先对数据进行排序。

方法 B: idxmax (推荐)

直接找到目标值（如最高分）对应的索引，然后通过 .loc 提取。通常性能最佳。

方法 C: drop_duplicates

排序后直接利用去重功能保留每组的第一条。代码最简洁。

Python - Three Alternatives

```python
# ═══ 方法 A: 使用 agg 和 lambda ═══
# 链式调用：先排序，再分组并对子集取第一行
best_A = (students
    .sort_values("成绩", ascending=False)
    .groupby("院系", as_index=False)
    .agg(lambda x: x.iloc[0])
)

# ═══ 方法 B: 使用 idxmax（无需预先排序）═══
# 1. 找到每个院系成绩最大值的行索引
idx = students.groupby("院系")["成绩"].idxmax()
# 2. 根据索引提取整行数据
best_B = students.loc[idx]

# ═══ 方法 C: 使用 drop_duplicates ═══
# 链式调用：排序后对"院系"列去重，保留第一条
best_C = (students
    .sort_values("成绩", ascending=False)
    .drop_duplicates(subset=["院系"], keep="first")
)
```

DataFrameGroupBy 对象的更多操作

DataFrameGroupby 对象的本质

GroupBy 操作返回的不是 DataFrame，而是一个中间对象（Plan）。我们可以通过它来检查分组细节，而不仅仅是聚合。

.groups 属性

返回一个字典，键是组名，值是该组对应的原始行索引（Index）。

.get_group() 方法

允许你提取出属于特定分组的所有行，返回一个标准的 DataFrame。这对调试非常有帮助。

Python

```python
# 创建按照"院系"分组的对象
grouped = students.groupby("院系")

# 1. 检查对象类型
type(grouped)
# 输出: <class 'pandas.core.groupby.generic.DataFrameGroupBy'>

# 2. 查看所有分组的名称 (Keys)
print(grouped.groups.keys())

# 3. 提取"计算机系"这一组的所有数据
cs_dept = grouped.get_group("计算机系")
cs_dept.head(3)
```

输出结果：

```
dict_keys(['计算机系', '自动化系', '电子系', '经管学院'])
```

学号	姓名	院系	课程	成绩
20230101	王小明	计算机系	数据结构	92
20230105	李华	计算机系	操作系统	88
20230108	张伟	计算机系	算法设计	95

数据透视表

Pivot Tables

多维数据分析

Why：为什么需要透视表?

What：透视表定义

How：透视表工作机制

为什么需要透视表?

业务场景：教务数据多维分析

需求：我们需要直观地对比**不同院系**在**不同年份**的平均成绩趋势。

传统方法：多列分组 (GroupBy)

```
df.groupby(["年份", "院系"])["成绩"].mean()
```

年份	院系	成绩
2024	计算机	90.5
	自动化	88.2
2025	计算机	92.1
	自动化	89.5

难以横向对比

高效方案：数据透视表 (Pivot Table)

```
df.pivot_table(index="年份", columns="院系", values="成绩")
```

年份 \ 院系	计算机	自动化
2024	90.5	88.2
2025	92.1	89.5

直观清晰

透视表定义

基本语法

```
df.pivot_table(
    index="行分类",
    columns="列分类",
    values="数值列",
    aggfunc="mean"
)
```

三个核心参数

Index (行索引)

你想把哪些类别放在侧边（左侧）作为行标签?

Columns (列索引)

你想把哪些类别放在顶端作为列标签?

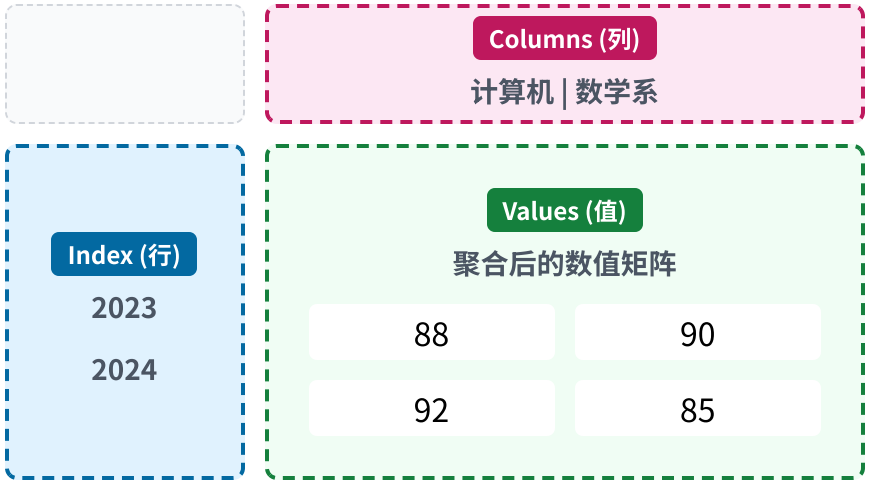
Values (数值)

你想对哪些数值进行计算（求和、平均值等）?

1. 原始数据 (Long Format)

年份	院系	平均分
2023	计算机	88
2023	数学系	90
2024	计算机	92
2024	数学系	85

2. 透视表结果 (Wide Format)



透视表工作机制

核心流程

透视表通过重塑数据结构，将"长格式"数据转换为易于阅读的"宽格式"交叉表。

1. 分组

根据行索引(Index)和列索引(Columns)将数据切分为小块。

2. 聚合

对每个数据块应用聚合函数(aggfunc)，如求均值。

3. 重塑

将聚合结果填入对应的行和列位置，形成二维表格。

● 行索引 ● 列索引 ● 聚合值

1 原始"长表"数据

年份 (Index)	院系 (Columns)	学生	成绩 (Values)
2023	CS	张三	95
2023	CS	李四	85
2023	EE	王五	90
2024	CS	赵六	88

↓

2 分组与聚合

2023 + CS	2023 + EE	2024 + CS	2024 + EE
95 (张三) 85 (李四) Mean: 90.0	90 (王五) Mean: 90.0	88 (赵六) Mean: 88.0	(无数据) NaN

↓

3 最终透视表

年份 \ 院系	CS	EE
2023	90.0	90.0
2024	88.0	NaN

透视表示例：多级索引与多值聚合

清華大學 Tsinghua University

Python

```python
pivot_result = students.pivot_table(
    index=["院系", "年级"], # 多级行索引：先按院系，再按年级
    columns=["学期", "性别"], # 多级列索引：先按学期，再按性别
    values=["成绩", "学分"], # 聚合多个数值列
    aggfunc="mean" # 计算平均值
)
```

↓ 生成结果

		成绩 (Mean Score)				学分 (Mean Credits)			
		2023 秋季		2024 春季		2023 秋季		2024 春季	
院系	年级	男	女	男	女	男	女	男	女
计算机系	2022级	88.5	90.2	**92.4**	**93.8**	3.5	3.5	3.0	3.0
	2023级	86.0	89.5	**88.2**	**91.0**	4.0	4.0	3.5	3.5
数学系	2022级	91.5	93.0	**94.5**	**95.2**	3.5	3.5	3.0	3.0
	2023级	85.5	88.0	**89.5**	**90.5**	4.0	4.0	3.5	3.5

表连接

Joining Tables

合并与连接操作

- 为什么需要表连接?
- 什么是键值?
- 连接类型详解

为什么需要表连接?

业务场景：教务数据分散存储

在真实的数据库设计中，为了减少数据冗余和维护一致性，我们通常将不同维度的信息存储在不同的表中。

学生表：存储学号、姓名、院系等相对稳定的基本信息。
成绩表：存储每个学期的课程、分数等动态数据。

> **⚠ 核心痛点**
>
> 数据分散在多个表中，无法直接分析。例如：我们知道"学号202301"考了95分，但不知道他叫什么名字，属于哪个院系。

→ **解决方案**：通过一个共同的列（键/Key）将两张表"连接"起来。

表1：学生基本信息 (students) Shape: (3, 3)

学号	姓名	院系
202301	张三	计算机系
202302	李四	电子系
202303	王五	自动化系

独立存储，信息断层

表2：课程成绩 (grades) Shape: (4, 3)

学号	课程	成绩
202301	数据结构	95
202301	线性代数	88
202302	信号与系统	92
202304	微积分	78

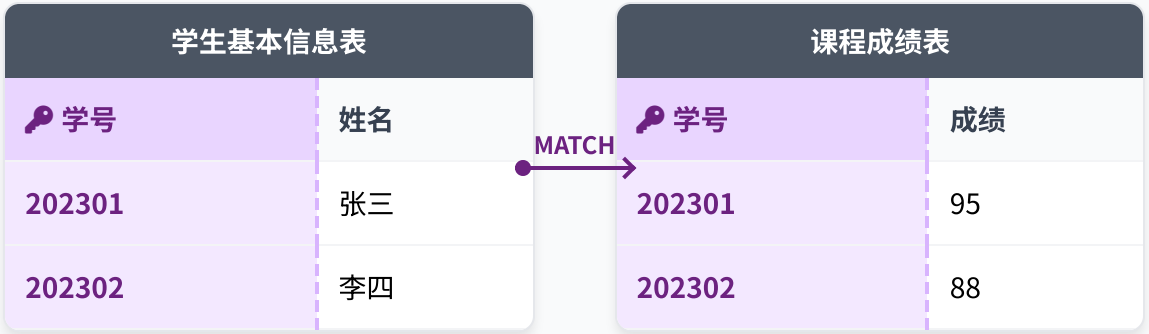
Key Insight: 两张表都有 **"学号"** 列，这是连接它们的桥梁。

什么是键?

键的核心概念

在表连接中，**键 (Key)** 是两个表之间共有的列，用于确定哪些行应该匹配在一起。它是连接不同数据孤岛的桥梁。

- **匹配依据：** Pandas 根据键列中的值是否相等来决定如何合并行。
- **唯一性：** 理想情况下，键在至少一个表中应该是唯一的（如学号、ID），以避免产生笛卡尔积。



键的类型 (Types of Keys)

单键 (Single Key)

使用单个列作为连接依据

例如：`on="学号"`

复合键 (Composite Key)

使用多个列的组合作为唯一标识

例如：`on=["姓名", "院系"]`

不同名键

两表中含义相同但列名不同

例如：`left_on="ID", right_on="学号"`

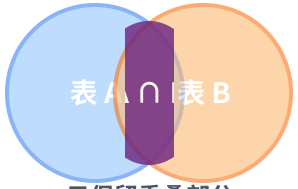
内连接（Inner Join） - 只保留交集

核心概念

内连接（Inner Join）是表连接中最基础、最常用的类型。它类似于集合论中的“交集”。

关键规则

只有当两个表中的连接键（Key）完全匹配时，记录才会被保留。



只保留重叠部分

```python
# 内连接语法（默认为 inner）
merged = pd.merge(
  students, scores,
  on="学号",
  how="inner" # 显式指定
)
```

数据流转演示

表1：学生表 (Left)

学号 (Key)	姓名	院系
202301	张三	CS
202302	李四	EE
202303	王五	Math

表2：成绩表 (Right)

学号 (Key)	课程	成绩
202301	数据结构	95
202302	微积分	88
202301	算法	92
202304	赵六	85

匹配 Key = "学号"

结果表：内连接 (Inner Join)

学号	姓名	院系	课程	成绩
202301	张三	CS	数据结构	95
202301	张三	CS	算法	92
202302	李四	EE	微积分	88

左连接 (Left Join)

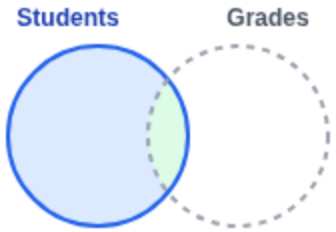
左表: Students (学生表)

学号	姓名	院系
202301	张三	CS
202302	李四	EE
202303	王五	Math

右表: Grades (成绩表)

学号	课程	分数
202301	数据结构	95
202301	算法	92
202302	微积分	88
202304	赵六	85

```
pd.merge(
  left=students,
  right=grades,
  on="学号",
  how="left"
)
```



逻辑：保留左表所有行

右表无匹配时填 NaN

合并结果 (Result DataFrame)

学号	姓名	院系	课程	分数
202301	张三	CS	数据结构	95
202301	张三	CS	算法	92
202302	李四	EE	微积分	88
202303	王五	Math	NaN	NaN

Right Join (右连接) 示例

左表: Students (学生)

学号	姓名	院系
202301	张三	CS
202302	李四	EE
~~202303~~	~~王五~~	~~Math~~

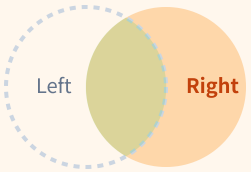
* 王五 (202303) 不在右表，被丢弃

右表: Grades (成绩)

学号	课程	分数
202301	数据结构	95
202301	算法	92
202302	微积分	88
202304	赵六	85

* 赵六 (202304) 仅在右表，保留

连接逻辑




```
pd.merge(
  left=students,
  right=grades,
  on="学号",
  how="right"
)
```

匹配部分 + 右表独有

结果表: Result DataFrame

学号	姓名	院系	课程	分数
202301	张三	CS	数据结构	95
202301	张三	CS	算法	92
202302	李四	EE	微积分	88
202304	NaN	NaN	赵六	85

外连接 (Outer Join) - 保留并集

1 输入数据

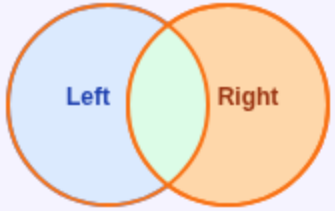
学生表 (Students)

学号	姓名	院系
202301	张三	CS
202302	李四	EE
202303	王五	Math

成绩表 (Grades)

学号	课程	分数
202301	数据结构	95
202301	算法	92
202302	微积分	88
202304	赵六	85

2 连接逻辑



A ∪ B (并集)
保留两表所有数据

```
pd.merge( students, grades,
on="学号", how="outer" )
```

3 结果数据

结果表 (Result) - 5行数据

学号	姓名	院系	课程	分数
202301	张三	CS	数据结构	95
202301	张三	CS	算法	92
202302	李四	EE	微积分	88
202303	王五	Math	*NaN*	*NaN*
202304	*NaN*	*NaN*	赵六	85

连接表的两种方式

调用风格对比

Pandas 提供了两种等价的语法来实现表连接操作，您可以根据团队代码风格进行选择。

1. 函数式 (Functional)

`pd.merge(left, right)`

显式指定左表和右表，逻辑清晰，适合数据管道的起始步骤。

2. 面向对象 (OOP)

`left.merge(right)`

直接在 DataFrame 上调用，更适合链式调用 (Method Chaining)。

Python

```python
# 写法 A：使用 pandas 顶级函数（推荐用于管道开始）
merged = pd.merge(
    left=students_info, # 左表：学生信息
    right=scores, # 右表：成绩记录
    on="学号", # 连接键
    how="inner" # 连接类型（默认）
)

# 写法 B：使用 DataFrame 方法（推荐用于链式调用）
merged2 = scores.merge(
    students_info, # 右表（左表是 self）
    on="学号",
    how="left" # 保留左表全部行
)
```

连接结果展示与检查

质量检查清单

合并数据后，必须进行以下检查以确保数据的完整性和准确性：

形状检查 (.shape)

确认行数是否符合预期（是否因重复键导致爆炸或因 inner join 导致丢失）。

缺失值检查 (.isna)

查看合并过程中是否引入了意外的空值。

随机抽查 (.sample)

人工核对几条记录，验证列对齐是否正确。

Python

```
# 1. 基本检查：查看行数变化与缺失比例
print(f"Shape: {merged.shape}")
print(merged.isna().mean().round(3))

# 2. 典型分析：计算每门课的平均分（带院系维度）
analysis = (merged.groupby(["院系", "课程"])["成绩"]
                  .mean().reset_index())
analysis.head(5)
```

控制台输出：

```
Shape: (2450, 7)
学号: 0.000, 姓名: 0.000, 院系: 0.000, 课程: 0.000, 成绩: 0.000, 学分: 0.000, 年份: 0.000
```

analysis.head(5) 运行结果：

索引	院系	课程	成绩
0	计算机系	数据结构	85.4
1	计算机系	操作系统	82.1
2	自动化系	电路原理	88.5
3	自动化系	自动控制原理	84.2
4	电子系	信号与系统	86.7

本讲核心内容总结

清華大學 Tsinghua University

分组聚合

Split-Apply-Combine　.agg()　.filter()

数据透视表

index (行)　columns (列)　values (值)

表连接

Inner　Left / Right　Outer

关键语法速查

```
# 1. 分组聚合
df.groupby("Key").agg(["sum", "mean"])
# 2. 透视表
df.pivot_table(index="R", columns="C", values="V")
# 3. 表连接
pd.merge(left, right, on="Key", how="inner")
```