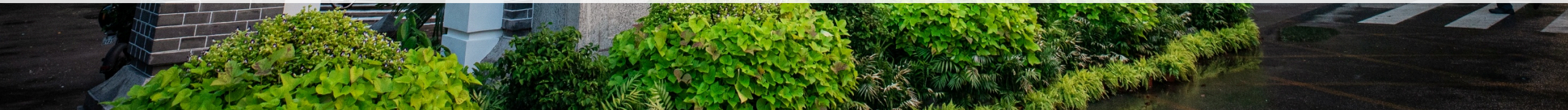
清华大学数据库技术与应用

Pandas 第二部分

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 条件选择

学习如何筛选满足特定条件的数据行与子集

2 列的操作

掌握添加、删除、重命名及修改列数据的技巧

3 实用函数

介绍 Pandas 中高效的数据分析常用工具函数

4 自定义排序

基于复杂逻辑或自定义规则进行灵活排序

知识回顾

清華大學 Tsinghua University

两种数据提取方式

`.iloc` 基于**整数位置**进行索引

`.loc` 基于**标签** 进行索引

```
# 提取 students_df 的前 5 行数据
# 使用 .loc（包含结束标签）
first_5_rows = students_df.loc[:4, :]
# 使用 .iloc（不包含结束位置）
first_5_rows = students_df.iloc[:5, :]
```

Index	Name	StudentID	Major	Score
0	张伟	20230101	计算机科学	92
1	李娜	20230102	自动化	88
2	王强	20230103	电子工程	95
3	刘洋	20230104	物理学	89
4	陈静	20230105	经济管理	94

条件筛选行

如何基于条件筛选行?

除了整数位置和标签，`.loc` 和 `[ ]` 还接受**布尔数组**作为输入。

筛选成绩 > 90 的学生?

Bool Mask	Index	Name	Major	Score
True	0	张伟	计算机科学	92
False	1	李娜	自动化	88
True	2	王强	电子工程	95
False	3	刘洋	物理学	89
True	4	陈静	经济管理	94

布尔数组输入

.loc 筛选机制

我们也可以使用 `.loc` 来执行完全相同的布尔筛选操作。

- 输入是一个由 `True` 和 `False` 组成的列表或数组。
- `True` 对应的行会被保留。
- `False` 对应的行会被丢弃。

关键要求

布尔数组的长度必须与 DataFrame 的行数**严格一致**，否则 Pandas 会报错。

```python
# 假设 students_df 有 5 行数据

# 创建一个布尔列表，指定要保留第 1, 3, 5 行
bool_list = [True, False, True, False, True]

# 传入 .loc 进行筛选（这里选取所有列 : ）
selected_students = students_df.loc[bool_list, ":"]
```

筛选过程可视化

逻辑运算输入

逻辑运算原理

我们可以通过对 **Series** 应用逻辑运算符来生成布尔数组。

- **逐元素比较**: 运算符会分别应用于 Series 中的每一个元素。
- **返回值**: 一个新的 Series，长度与原 Series 相同，包含 `True` 或 `False`。

案例目标：

找出所有 **成绩 > 90** 的优秀学生记录。

```
# 生成布尔数组：筛选成绩大于 90 分的学生
is_high_score = students_df["Score"] > 90
print(is_high_score)
```

students_df["Score"]

Idx	Value
0	92
1	88
2	95
3	89
4	94

> 90

Result (Boolean Series)

Idx	Value
0	True
1	False
2	True
3	False
4	True

注意：生成的 Series 长度与原数据框行数完全一致

布尔掩码过滤

清華大學 Tsinghua University

布尔掩码 (Boolean Mask)

我们可以使用生成的布尔数组作为“掩码”来过滤 DataFrame 中的行。

- 对应 `True` 的行被保留
- 对应 `False` 的行被丢弃

⚠ **注意：**布尔掩码的长度必须与 DataFrame 的行数完全一致。

```python
# 任务：筛选所有成绩 > 90 分的学生数据

# 1. 创建布尔掩码 (Length: 5)
mask = students_df["Score"] > 90

# 2. 应用掩码进行筛选
high_score_students = students_df[mask]

---

# 或者一步到位的写法：
high_score_students = students_df[students_df["Score"] > 90]
```

Index	Name	Gender	Major	Score
0	张伟	男	计算机科学	92
2	王强	男	电子工程	95
4	陈静	女	经济管理	94

组合布尔筛选

清華大學 Tsinghua University

案例目标

从学生成绩表中筛选出所有**成绩优秀且性别为女**的学生记录：

- 条件1：成绩 (Score) **> 90** 分
- 条件2：性别 (Gender) **== "女"**

组合条件的关键

使用位运算符 `&` (AND) 连接两个条件。

注意：每个条件必须用圆括号()包裹！

```python
# 1. 构建组合布尔掩码（成绩 > 90 AND 性别 == "女"）

# 注意：pandas 中必须使用 & 运算符，且每个条件需加括号
mask = (students_df["Score"] > 90) & (students_df["Gender"] == "女")

# 2. 使用 .loc 进行筛选
excellent_female_students = students_df.loc[mask, :]

# 打印结果
print(excellent_female_students)
```

Index	Name	Gender	Major	Score
4	陈静	女	经济管理	94

组合布尔筛选

清華大學 Tsinghua University

案例目标

从学生成绩表中筛选出所有 **成绩优秀 OR 性别为女** 的学生记录：

条件 1
成绩 (Score) > 90 分

--- OR (或) ---

条件 2
性别 (Gender) == "女"

组合条件的关键：OR

使用位运算符 `|` (OR) 连接两个条件。

相比 AND，OR 更宽松，只要满足其中一个条件即可。
注意：每个条件必须用圆括号 () 包裹！

```
# 1. 构建组合布尔掩码（成绩 > 90 OR 性别 = "女"）# 注意：必须使用 | 运算符，且每个条件需加括号
mask = (students_df["Score"] > 90) | (students_df["Gender"] == "女")

# 2. 使用 .loc 进行筛选
result_students = students_df.loc[mask, :]
```

Index	Name	Gender	Major	Score
0	**张伟**	男	计算机科学	**92** ✓
1	**李娜**	**女** ✓	自动化	88
2	**王强**	男	电子工程	**95** ✓
3	刘洋	男	物理学	89 ×
4	**陈静**	**女**	经济管理	**94** ✓

Length: 4 rows, Columns: 5

位运算符

在 Pandas 中，我们使用位运算符来组合多个布尔条件。假设 `p` 和 `q` 是布尔数组或 Series：

符号	用法	含义	说明
~	~p	NOT p	逻辑非 (取反)
|	p | q	p OR q	逻辑或 (满足其一即可)
&	p & q	p AND q	逻辑与 (需同时满足)
^	p ^ q	p XOR q	逻辑异或 (互斥)

课堂小测试

综合应用

下列哪个 Pandas 语句能够正确返回 前 3 个 成绩大于 85 分的学生记录?

任务拆解

- 条件：`Score > 85`
- 数量：前 3 行数据
- 注意：操作顺序很重要

请选择正确的代码实现：

A

```
students_df[students_df["Score"] > 85].head(3)
```

B

```
students_df.head(3)[students_df["Score"] > 85]
```

C

```
students_df[students_df > 85].head(3)
```

D

```
students_df.head(3) & students_df["Score"] > 85
```

测试答案解析

Q1 正确选项

A

```
students_df[students_df["Score"] > 85].head(3)
```

正确。 逻辑顺序：**先筛选 → 后截取**。
先通过布尔索引选出所有成绩大于85分的学生，然后从这个结果集中取前3行。这符合题目"前3个成绩大于85分的学生"的要求。

选项逐一解析

B **错误：逻辑顺序颠倒**

代码 `.head(3)[... > 85]` 会先取原始数据的前3行。如果这3个人里没有或只有1个人成绩大于85，结果就会少于3行，甚至为空。这是最常见的逻辑陷阱。

C **错误：语法错误**

缺少列名索引。代码 `students_df > 85` 试图将整个 DataFrame（包含姓名、专业等字符串列）与数字比较，会导致报错。

D **错误：运算符使用错误**

位运算符 `&` 用于连接两个布尔序列，不能用于连接 DataFrame 切片和布尔序列。此外逻辑也是错的。

操作顺序图解

布尔选择的替代方法

为什么需要替代方法?

直接使用布尔数组虽然功能强大，但在处理多个条件（尤其是 OR 逻辑）时，代码容易变得**冗长且难以维护**。

Pandas 提供的简洁方案

`.isin(list)`

用于检查值是否存在于列表或集合中。替代多个 OR 条件。

`.str.startswith()`

字符串访问器方法。用于高效的字符串模式匹配。

`.groupby().filter()`

基于组的属性过滤行（将在第4讲详细介绍）。

案例 1: 筛选多个专业

Verbose Approach（繁琐）

```
students_df[
    (students_df["Major"] == "计算机科学") |
    (students_df["Major"] == "自动化") |
    (students_df["Major"] == "物理")
]
```

Better Approach（推荐）

```
# 定义目标列表
target_majors = ["计算机科学", "自动化", "物理"]
students_df[students_df["Major"].isin(target_majors)]
```

案例 2: 筛选姓“张”的学生

```
# 返回一个布尔Series，当名字以"张"开头时为True
students_df[students_df["Name"].str.startswith("张")]
```

布尔选择的替代方法：.isin()

清華大學 Tsinghua University

.isin() 的返回值

`.isin()` 本质上是一个**逐元素比较**操作。

它检查 Series 中的每个值是否存在于目标列表中，并返回一个长度相同的 **布尔 Series (True/False)**。

代码实现

```
# 1. 定义目标专业列表
target_majors = ["计算机科学", "自动化", "物理学"]

# 2. 生成布尔掩码 (Mask)
mask = students_df["Major"].isin(target_majors)

# 3. 应用筛选
result = students_df[mask]
```

可视化流程

1 生成布尔 Series (Mask)

students_df["Major"]

Idx	Value
0	计算机科学
1	自动化
2	电子工程
3	物理学
4	经济管理

→ `.isin([...])`

mask (Boolean Series)

Idx	Value
0	True
1	True
2	False
3	True
4	False

↓

2 筛选结果 DataFrame (仅保留 True 的行)

Idx	Name	Major	Score
0	张伟	计算机科学	92
1	李娜	自动化	88
3	刘洋	物理学	89

布尔选择的替代方法：.str.startswith()

清華大學 Tsinghua University

.str.startswith() 的返回值

`.str.startswith()` 同样是一个向量化操作。

它检查 Series 中的每个字符串是否以指定字符开头，并返回一个长度相同的 **布尔 Series (True/False)**。

</> 代码实现

```
# 1. 定义筛选条件：姓氏以"张"开头 # 案例：筛选所有姓"张"的学生

# 2. 生成布尔掩码 (Mask)
mask =
  students_df["Name"].str.startswith("张")

# 3. 应用筛选
result = students_df[mask]
```

可视化流程

1 生成布尔 Series (Mask)

students_df["Name"]

Idx	Value
0	**张伟**
1	李娜
2	王强
3	**张敏**
4	陈静

→ `.str.startswith("张")`

mask (Boolean Series)

Idx	Value
0	**True**
1	False
2	False
3	**True**
4	False

2 筛选结果 DataFrame (仅保留 True 的行)

Idx	Name	Major	Score
0	**张伟**	计算机科学	92
3	**张敏**	物理学	89

** Index 1, 2 和 4 被过滤掉了*

本讲议程

清華大學 Tsinghua University

1 **条件选择**

学习如何筛选满足特定条件的数据行与子集

2 **列的操作**

掌握添加、删除、重命名及修改列数据的技巧

3 **实用函数**

介绍 Pandas 中高效的数据分析常用工具函数

4 **自定义排序**

基于复杂逻辑或自定义规则进行灵活排序

添加列的语法

清華大學 Tsinghua University

基本语法

像给字典赋值一样，将一个 Series 赋值给 DataFrame 的新键（列名）。

```
df["新列名"] = Series或值
```

```python
# 示例：创建一个新列 "name_length"，存储学生名字的字符数

# 1. 计算每个名字的长度（Series）
name_lengths = students_df["Name"].str.len()

# 2. 将 Series 赋值给新列 "name_length"
students_df["name_length"] = name_lengths
```

操作步骤

1. **生成数据**：创建一个与 DataFrame 索引匹配的 Series（例如计算名字长度）。
2. **赋值**：将该 Series 赋给一个新的列名。

Index	Name	Major	Score	name_length
0	张伟	计算机科学	92	5
1	李娜	自动化	88	3
2	王强	电子工程	95	4
3	欧阳震华	物理学	89	3

修改列的语法

修改现有数据

修改列的语法与创建新列非常相似。如果我们对一个**已存在的列名**进行赋值，Pandas 将会更新该列中的值。

```
df["现有列"] = 新值或运算结果
```

向量化运算

对列的操作会自动应用到该列的每一行数据（Broadcasting），无需编写循环。

```
# 示例：将 name_length 列的值减 1
# 修改前："张伟"（2）→ 修改后：1
students_df["name_length"] = students_df["name_length"] - 1
```

students_df (Modified)

Index	Name	Major	Original Length	name_length
0	张伟	计算机科学	5	**4**
1	Li Na	自动化	3	**2**
2	王强	电子工程	4	**3**
3	Liu Yang	物理学	3	**2**
4	陈静	经济管理	4	**3**

重命名列的语法

使用 .rename() 方法

使用 `.rename()` 方法并传入 `columns` 字典参数来重命名列。

```
df.rename(columns={"旧名": "新名"})
```

> ⚠️ **常见错误提示 (Common Typo)**
>
> 非常容易混淆字典中键（Key）和值（Value）的顺序！
>
> - **Key (键)** 是 **当前列名**
> - **Value (值)** 是 **想要的新列名**
>
> *"Forgive yourself if you mix them up! ❤️"*

Python Code

```python
# 将 "name_length" 列重命名为 "Length"
students_df = students_df.rename(
    columns={"name_length": "Length"}
)
```

执行结果 Result

Index	Name	Major	Score	Length
0	张伟	计算机科学	92	2
1	李娜	自动化	88	2
2	欧阳修	人文学院	95	3

** Data 100/200, Fall 2025 @ Tsinghua University*

删除列的语法

清華大學 Tsinghua University

核心方法

使用 `.drop()` 方法移除数据。

关键参数

- 默认情况下，`.drop()` 删除的是**行**。
- 必须指定 `axis="columns"` (或 axis=1) 来删除列。

提示：就像把列"丢掉"一样简单，但要告诉 Pandas 丢的是列！ 👋

```python
# 删除 "name_length" 列
students_df = students_df.drop(
    "name_length", axis="columns"
)
```

结果 DataFrame

Index	Name	Major	~~name_length~~	Score
0	张伟	计算机科学	2	92
1	李娜	自动化	2	88
2	王强	电子工程	2	95

列 `name_length` 已被成功移除

重要提示：DataFrame 副本

清華大學 Tsinghua University

默认行为

在上一页中，我们使用 `.drop()` 删除了列。

关键点：Pandas 默认会创建一个**新的副本 (New Copy)**，而**不会**修改原始的 DataFrame。

⚠ 常见误区

如果只是调用方法而不进行赋值，你的更改将会丢失！

"Our change was not applied!"

解决方案：将结果重新赋值给原变量。

错误做法 (无变化)

```
# 尝试删除 "name_length" 列
students_df.drop("name_length", axis="columns")
# students_df 保持原样，"name_length" 仍然存在！
```

>>> 更改未保存

正确做法 (重新赋值)

```
# 将返回的新 DataFrame 重新赋值给变量
students_df = students_df.drop("name_length",
axis="columns")
# 现在 students_df 已经更新了
```

本讲议程

清華大學 Tsinghua University

1 **条件选择**

学习如何筛选满足特定条件的数据行与子集

2 **列的操作**

掌握添加、删除、重命名及修改列数据的技巧

3 **实用函数**

介绍 Pandas 中高效的数据分析常用工具函数

4 **自定义排序**

基于复杂逻辑或自定义规则进行灵活排序

NumPy

Pandas Series 和 DataFrame 支持许多操作，包括 **NumPy 操作**，只要底层数据是数值型的。

常用 NumPy 函数

- `np.mean()` 计算平均值
- `np.max()` 计算最大值
- `np.log()` 计算对数

Pro Tip: 精度控制

只要 Python 显示了很多小数位，并不意味着你需要汇报所有位数。在报告中保持适当的精度（如 89.4）通常更专业。

```python
# 1. 筛选出计算机系(CS)学生的成绩
cs_scores = students_df[students_df["Major"] == "计算机科学"]["Score"]

# 2. 使用 NumPy 计算平均分
import numpy as np
np.mean(cs_scores)
```

```
89.42857142857143
```

```python
# 3. 计算最高分
np.max(cs_scores)
```

```
98
```

Pandas Series 兼容 NumPy 数组接口

Pandas 内置方法

Pandas 提供了极其丰富的内置工具函数用于数据探索和操作。如果想要对数据进行某种处理，通常已经有现成的函数可以实现。

ATTRIBUTES

`.shape / .size`

查看 DataFrame 的维度（行数, 列数）或元素总数。

STATISTICS

`.describe()`

快速生成数据的统计摘要（均值、标准差、分位数等）。

SAMPLING

`.sample()`

从 DataFrame 中随机抽取行，支持有放回或无放回抽样。

COUNTING

`.value_counts()`

计算 Series 中每个唯一值出现的频率，结果默认降序排列。

UNIQUE VALUES

`.unique()`

返回 Series 中的所有唯一值组成的数组。

SORTING

`.sort_values()`

根据指定列的值对 DataFrame 进行升序或降序排序。

学习建议

如果你想以某种方式处理数据，Pandas 大概率已经提供了内置函数。请善用 **官方文档**、 Google 搜索或询问 LLM (Large Language Model)!

.shape 和 .size

清華大學 Tsinghua University

.shape

返回一个表示维度的**元组 (tuple)**。

```
(# rows, # columns) = (行数, 列数)
```

.size

返回 DataFrame 中的**元素总数**。

```
# rows × # columns = 行数 × 列数
```

示例数据：students_df

假设我们有一个包含清华大学学生成绩的 DataFrame，共有 **2,500** 名学生，每条记录包含 **6** 个字段（姓名、学号、性别、专业、成绩、年级）。

```
# 查看 DataFrame 的维度
students_df.shape
> (2500, 6)
```

```
# 查看总元素个数 (2500 * 6)
students_df.size
> 15000
```

注意：这些是属性(attributes)而非方法(methods)，不需要加括号()

.describe()

清華大學 Tsinghua University

功能描述

`.describe()` 方法用于生成 DataFrame 或 Series 的描述性统计摘要。它能快速帮助我们了解数据的分布情况。

包含的统计指标：

- **count** 非空值的数量
- **mean** 平均值
- **std** 标准差 (离散程度)
- **min/max** 最小值 / 最大值
- **25/50/75%** 四分位数 (50%即中位数)

> **注意：**
> 默认情况下，该方法**仅汇总数值型列** (Numeric Columns)！
> 非数值列（如姓名、专业）将被自动忽略。

```
# 查看 students_df 的统计摘要
students_df.describe()
```

Output DataFrame

	Score	GPA	Credits
count	120.000000	120.000000	120.000000
mean	88.450000	3.652000	82.500000
std	6.230000	0.320000	12.400000
min	65.000000	2.500000	45.000000
25%	85.000000	3.500000	72.000000
50%	89.500000	3.700000	84.000000
75%	93.000000	3.900000	92.000000
max	100.000000	4.000000	110.000000

.describe() 用于 Series

非数值型数据的统计

当 `.describe()` 应用于非数值型（如 object, string, category）的 Series 时，Pandas 会报告一组不同的统计指标：

- `count` 非空值的总数量
- `unique` 不同取值（唯一值）的数量
- `top` 出现频率最高的值（众数）
- `freq` 最高频值出现的次数

```
# 查看 "Major"（专业）列的统计信息
students_df["Major"].describe()
```

Output: Series Summary

count	100
unique	15
top	计算机科学
freq	28

注意： 数值型列默认返回 mean, std 等指标，除非强制转换为 categorical 类型。

随机抽样

.sample() 方法

用于从 DataFrame 或 Series 中随机抽取行。

- **n**: 抽取的行数（默认为 1）。
- **frac**: 抽取的比例（如 0.5 表示 50%）。
- **replace**: 是否有放回抽样。
  - `False` (默认): 不放回，不可重复。
  - `True`: 有放回，可能重复抽取。

数据分析提示

由于数据框的头部 (`.head()`) 往往具有某种排序或规律，不能代表整体分布。
在探索数据时，优先使用 `df.sample()` 来随机检查数据。

示例 1 随机抽取 3 名学生 (无放回)

```
# 默认 replace=False
students_df.sample(n=3, random_state=42)
```

Index	Name	Major	Score
8	赵敏	新闻学	89
2	王强	电子工程	95
15	孙悟空	生命科学	92

示例 2 随机抽取 3 名学生 (有放回)

```
# replace=True 允许行被重复选中
students_df.sample(n=3, replace=True)
```

Index	Name	Major	Score
5	周杰	建筑学	87
5	周杰	建筑学	87
11	吴京	体育	90

方法链接

清華大學 Tsinghua University

任务：筛选2023级学生 → 随机抽取3人 → 提取姓名和成绩

students_df

APPROACH 1: 不使用方法链接

特点：使用中间变量，代码冗长，占用内存

```
# 1. 筛选行（中间变量1）
students_2023 = students_df[students_df["Year"] == 2023]

# 2. 随机采样（中间变量2）
sampled_students = students_2023.sample(3, replace=True)

# 3. 选择列（最终结果）
result = sampled_students.loc[:, ["Name", "Score"]]
```

APPROACH 2: 使用方法链接 (推荐)

特点：无需中间变量，逻辑连贯，代码紧凑

```
result = (
    students_df[students_df["Year"] == 2023] # 1. 筛选行
      .sample(3, replace=True) # 2. 随机采样
      .loc[:, ["Name", "Score"]] # 3. 选择列
)
```

为什么推荐链接?

- 避免创建不必要的中间变量名
- 阅读顺序与思维逻辑一致（从上到下）
- 利用括号 () 自动换行，无需反斜杠 \

运行结果 (Result Preview)

type: DataFrame

Index	Name	Score
42	林浩	88
15	周薇	95
8	王建国	91

.unique() 方法

清華大學 Tsinghua University

功能说明

`Series.unique()` 方法返回一个包含 Series 中所有唯一值的 NumPy 数组。

- **返回类型：** NumPy Array (ndarray)
- **主要用途：** 快速查看某列（如类别变量）包含哪些不同的取值。
- **注意：** 返回的结果是**未排序**的（按出现顺序）。

相关提示

如果只想知道唯一值的**数量**而不是具体内容，可以使用 `.nunique()` 方法，它返回一个整数。

案例：查看清华学生数据中的所有专业

Original Column: "Major"

计算机科学

自动化

计算机科学

电子工程

自动化

...

```
# 获取所有出现的不同专业
majors_array =
students_df["Major"].unique()
```

执行结果：

Output (NumPy Array)

```
array(['计算机科学', '自动化', '电子工程',
       '物理学', '经济管理', '建筑学'],
      dtype=object)
```

去除了重复项，仅保留唯一值

.value_counts() 方法

功能概述

`Series.value_counts()` 返回一个包含唯一值计数的 Series。

- **默认排序：**按计数降序排列。
- **索引：**唯一值作为结果的索引。
- **缺失值：**默认忽略 `NaN`（除非 `dropna=False`）。

常用参数

`normalize=True`

返回相对频率（百分比）而非绝对计数。

`ascending=True`

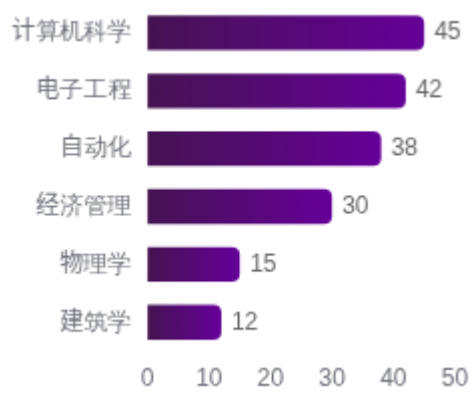
改为升序排列（计数最少的在前）。

```
# 统计清华学生样本中各专业的学生人数
major_counts = students_df["Major"].value_counts()
print(major_counts)
```

CONSOLE OUTPUT

```
计算机科学    45
电子工程      42
自动化        38
经济管理      30
物理学        15
建筑学        12
...
Name: Major, dtype: int64
```

VISUAL REPRESENTATION



.sort_values() 方法

清華大學 Tsinghua University

核心功能

对 DataFrame 或 Series 进行值排序。

- **DataFrame:** 必须指定 `by` 参数（列名）。
- **Series:** 直接调用，无需指定列。

关键参数

`ascending`	默认为 `True` (升序) 设置为 `False` (降序)

重要提示：Index 也会改变！

排序操作不仅仅是移动数据值，整行（包括索引）都会随之移动。

```
# 按"Score"列降序排列（分数从高到低）
students_df.sort_values(by="Score", ascending=False)

# Series 排序（默认升序）
students_df["Name"].sort_values()
```

结果: 按照分数降序排列

注意看索引也随之改变

Index	Name	Major	Score ↓
2	王强	电子工程	95
4	陈静	经济管理	94
0	张伟	计算机科学	92
3	刘洋	物理学	89
1	李娜	自动化	88

常见错误：忘记 `ascending` 的默认值。建议显式写出参数以提高代码可读性。

课堂小测试

列间比较与聚合

下列哪个 Pandas 语句能够正确提取 A列的值 至少 与 B列最小值 一样大的行?

解题关键点

- 获取B列最小值：`.min()`
- 比较条件："至少" 意味着 `≥`
- 筛选目标：提取符合条件的行

请选择正确的代码实现：

A `df[df['A'] ⩾ df['B'].min()]`

B `df[df['A'] ⩾ df['B']]`

C `df[df['A'] > df['B'].min()]`

D `df.loc[df['A'] ⩾ df['B'].min]`

测试答案解析

清華大學 Tsinghua University

正确选项：A

```
df[df['A'] ⩾ df['B'].min()]
```

该语句正确实现了题目要求：先计算B列的最小值，然后筛选出A列值大于等于该最小值的行。

解题关键点

- **聚合函数**：`.min()` 用于计算列的最小值，返回一个标量。
- **比较逻辑**："至少一样大" 对应数学上的 `⩾` (大于等于)。
- **布尔索引**：将比较产生的布尔 Series 放入 `df[]` 中进行筛选。

选项逐一解析

A `df[df['A'] ⩾ df['B'].min()]` ✓

正确。先计算标量最小值，再进行矢量化比较，逻辑完全符合题意。

B `df[df['A'] ⩾ df['B']]` ✗

错误。这是两列之间的**逐行比较**（Element-wise），而不是与B列的最小值比较。

C `df[df['A'] > df['B'].min()]` ✗

错误。使用了 > 而不是 >=。题目要求"至少一样大"包含等于的情况。

D `df.loc[df['A'] ⩾ df['B'].min]` ✗

错误。`.min` 缺少括号 `()`，这是方法对象而不是调用结果，会导致语法错误。

按长度排序

核心问题

在 Pandas 中，默认的 `.sort_values()` 是按照**字典序**进行排序的。

如果我们想要按照姓名长度（字符数）来排序，应该怎么做？

思考时间

回忆一下 Python 的 `sorted()` 函数或者 Pandas 的其他功能，有没有什么方法可以让我们自定义排序规则？

原始数据 (默认排序)

Name	Length
陈静	2
李华	2
欧阳娜娜	4
王建国	3
张伟	2

按拼音/字典序排列

期望结果 (按长度降序)

Name	Length
欧阳娜娜	**4**
王建国	**3**
陈静	2
李华	2
张伟	2

按字符数从多到少排列

如何实现这种排序？

```
students_df["Name"].sort_values()
```

方法 1：创建新列并排序

清華大學 Tsinghua University

操作步骤流程：

1 计算辅助数据

创建一个新列（例如 `name_length`），其中存储每个名字的长度。

2 执行排序

调用 `.sort_values()` 方法，指定新列作为排序依据。

特点： 这种方法直观易懂，适合需要保留中间计算结果（如名字长度）用于后续分析的场景。但会改变 DataFrame 的结构（增加了一列）。

```
# 1. 创建包含名字长度的新列
students_df["name_length"] = students_df["Name"].str.len()

# 2. 根据新列进行降序排序
students_df = students_df.sort_values(by="name_length", ascending=False)

# 查看结果
students_df.head(5)
```

Index	Name	Major	Score	name_length
35	欧阳娜娜	艺术设计	92	4
12	诸葛孔明	历史学	98	4
8	王小明	土木工程	85	3
0	张伟	计算机科学	92	2
4	李华	英语	89	2

方法 2：使用 key 参数

更加简洁的排序

我们可以直接在 `.sort_values()` 中使用 **key** 参数，而无需先创建中间列。

- 避免了修改原 DataFrame 结构
- 代码更加紧凑、可读性更强

Lambda 函数

`lambda x: x.str.len()`

这是一个**匿名函数**（即没有名称、临时的、一次性使用的函数）。它会被应用到排序列的每一个元素上，生成的转换值将用于排序。

```
# 按名字长度降序排序（无需创建新列！）
students_df.sort_values(
    "Name",
    key=lambda x: x.str.len(),
    ascending=False
).head()
```

RESULT

按 "Name" 长度降序排列

Index	Name	Major	Score
4	欧阳震华 4	建筑学	92
2	司马相如 4	中文系	89
1	王小明 3	自动化	88
0	张伟 2	计算机	95
3	李娜 2	经济学	94

方法 3：使用 Categorical 类型

适用场景

当我们需要按照**非字母顺序**的自定义逻辑（例如星期、年级、特定业务顺序）进行排序时，可以将列转换为 Categorical 类型，并明确指定类别的顺序。Pandas 会自动按照这个顺序进行排序。

自定义年级顺序：

大一 > 大二 > 大三 > 大四

```
# 1. 定义自定义顺序列表
custom_order = ["大一", "大二", "大三", "大四"]

# 2. 将 'Year' 列转换为有序的 Categorical 类型
students_df["Year"] = pd.Categorical( students_df["Year"], categories=custom_order, ordered=True )

# 3. 直接按照 'Year' 列排序
sorted_df = students_df.sort_values("Year")
```

Name	Major	Year (Sorted)	Score
李华	建筑学	大一	85
张伟	计算机	大一	92
王强	电子工程	大二	89
陈静	经管	大三	94
刘洋	物理系	大三	88
赵敏	中文系	大四	91

底层原理：Category Codes

Pandas 在底层将这些分类数据存储为整数代码（Codes）。"大一"映射为 0，"大二"映射为 1，以此类推。排序时，Pandas 实际上是根据这些整数代码进行排序，从而实现了我们期望的非字典序排序。

多列排序

清華大學 Tsinghua University

排序逻辑与优先级

- **排序键列表：**在 `by` 参数中传入列名列表。
- **优先级规则：**列表中的第一个列是**主排序键**，随后的列依次为次级排序键。
- **独立升降序：**`ascending` 参数接受布尔值列表，分别控制每一列的排序方向。

```
# 任务：按"专业"升序排列，
# 同专业内按"成绩"降序排列

sorted_df = students_df.sort_values(
  by=["Major", "Score"],
  ascending=[True, False]
)
```

排序结果预览

主键: Major ↑　次键: Score ↓

Index	Name	Major 1	Score 2
2	王强	计算机科学	95
0	张伟	计算机科学	92
5	赵敏	计算机科学	85
3	刘洋	物理学	89
6	孙策	物理学	88
1	李娜	自动化	88

观察结果：
首先按照 **Major** 拼音/字典序分组排列（计算机 → 物理 → 自动化），然后在每个专业内部，**Score** 按照从高到低排列。

空值排序

清華大學 Tsinghua University

na_position 参数

在使用 `.sort_values()` 排序时，默认情况下空值 (NaN) 会被放在最后。我们可以通过 `na_position` 参数来控制空值的位置。

`'last'`（默认） **`'first'`（置顶）**

```
# 场景：按成绩降序排序，但希望缺考(NaN)的学生排在最前面
# na_position='first': 将空值放在结果的开头
sorted_df = students_df.sort_values(
    by="Score",
    ascending=False,
    na_position='first'
)
```

实用技巧

这在需要优先处理缺失数据，或者想把"特殊情况"（如未评分）置顶时非常有用。

排序结果对比

按 Score 降序排列

默认行为 (na_position='last')

Name	Score
王强	95.0
张伟	92.0
赵敏	85.0
李娜	NaN
孙策	NaN

↓ 空值沉底

自定义 (na_position='first')

Name	Score
李娜	NaN
孙策	NaN
王强	95.0
张伟	92.0
赵敏	85.0

↑ 空值置顶

本讲总结

清華大學 Tsinghua University

条件选择

- **布尔筛选：** 使用布尔数组作为掩码，通过 `df[mask]` 或 `.loc` 精确提取数据。
- **逻辑组合：** 必须使用位运算符 `&` (AND), `|` (OR), `~` (NOT) 并加括号。
- **高效方法：** 使用 `.isin()` 匹配列表，使用 `.str` 访问器处理字符串。

列的操作

- **增删改名：** 字典式赋值添加列，`.drop()` 删除列，`.rename()` 重命名。
- **向量化运算：** 对整列进行运算无需循环，使用 `df["col"] + 1` 等表达式高效处理。
- **副本机制：** Pandas 操作默认返回新副本，需重新赋值才能保存更改。

实用函数

- **统计探索：** `.describe()` 概览，`.value_counts()` 频次，`.unique()` 唯一值。
- **数据抽样：** 使用 `.sample()` 进行随机抽样，支持有放回抽样。
- **方法链接：** 使用圆括号 `( )` 将多个操作链式连接，代码更优雅。

排序技巧

- **多维排序：** `.sort_values(by=[...])` 支持多列排序及独立升降序。
- **自定义排序：** 灵活运用 `key=lambda` 或 **Categorical** 类型定义顺序。
- **空值处理：** 使用 `na_position` 参数控制空值置顶或沉底。