清华大学数据库技术与应用

Pandas 第一部分

授课教师：计算机系王健楠

授课学期：2026年（春季）

本讲目标

这是关于 pandas 三讲序列中的第一讲。

核心知识点

pandas 库介绍：处理表格数据的重要 Python 库

关键数据结构： DataFrames、 Series、 Indices

数据提取方法： 使用 .loc、.iloc 和 [] 进行高效查询

课堂讲解

侧重介绍高层概念与语法逻辑

实验与作业

提供实际编码与动手的机会

数据科学家偏爱表格数据

"表格数据" = 以表格形式组织的数据

Student_ID	Name	Course	Score
20250101	张伟	**数据科学基础**	92
20250102	李娜	**数据科学基础**	95
20250103	王芳	**计算机程序设计基础**	88
20250104	刘洋	**统计学导论**	90
20250105	陈静	**数据结构**	85

↔ **行 (Row)**

代表一个 观测值 (Observation)

例：一名学生的单门课程记录（如 李娜 的记录）

↕ **列 (Column)**

代表该观测的某个 **特征** (Feature)

例：记录的属性（如 Course 课程名称列）

互动问题

从这个表格中，你最想回答的**有趣问题**是什么？

	Year	Candidate	Party	Popular vote	Result	%
0	1824	Andrew Jackson	Democratic-Republican	151,271	loss	57.210122
1	1824	John Quincy Adams	Democratic-Republican	113,142	win	42.789878
2	1828	Andrew Jackson	Democratic	642,806	win	56.203927
3	1828	John Quincy Adams	National Republican	500,897	loss	43.796073
4	1832	Andrew Jackson	Democratic	702,735	win	54.574789
...	...	...	...	...	...	...
182	2024	Donald Trump	Republican	77,303,568	win	49.808629
183	2024	Kamala Harris	Democratic	75,019,230	loss	48.336772
184	2024	Jill Stein	Green	861,155	loss	0.554864
185	2024	Robert Kennedy	Independent	756,383	loss	0.487357
186	2024	Chase Oliver	Libertarian Party	650,130	loss	0.418895

这个图还引发了你哪些新的观察或问题?

1900-2020年美国总统选举两党得票率趋势

% 得票率

65
60
55
50
45
40
35
30
25
20

1900 1912 1924 1936 1948 1960 1972 1984 1996 2008 2020

Year (年份)

Democratic (民主党) Republican (共和党)

我们需要什么数据来制作这张图表？

标准 Python 数据科学工具：pandas

THE PYTHON DATA ANALYSIS LIBRARY

pandas

研究与工业界处理表格数据的事实标准工具

名称来源

名称 "pandas" 并非来自可爱的熊猫 (Panda)， 而是源于计量经济学术语：

panel data analysis

（面板数据 / 多维数据）

核心定义

全称：Python Data Analysis Library

它是一个开源的、BSD 许可的库，为 Python 编程语言提供高性能、易用的数据结构和数据分析工具。

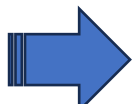
业界地位

数据科学领域的通用标准。

- 替代 Excel 进行自动化处理
- 与 NumPy, Matplotlib 深度集成

通常的导入方式： `import pandas as pd`

提纲



Series, DataFrames, and Indices

Pandas 的核心数据结构

数据提取

使用 .loc, .iloc, 和 [ ] 进行数据筛选与操作

Series 概念

什么是 Series?

Series 是 pandas 中的 一维数组型对象

它由两部分紧密构成： Values (值序列) Index (索引标签)

包含的数据类型通常为同一 dtype；也可能是 object 存放混合类型，但会影响性能/语义

基本构造与访问

```
import pandas as pd

# 创建一个包含字符串的 Series
s = pd.Series(["welcome","to","data world"])

# 访问属性
s.values # 获取值数组
s.index # 获取索引对象
```

对象结构图示

Index	Values
0	"welcome"
1	"to"
2	"data world"

↑ s.index　　s.values ↑

构造 Series

```
# 导入 pandas 库
import pandas as pd

# 创建带自定义索引的Series
s = pd.Series([-1, 10, 2],
    index = ["a", "b", "c"])

# 查看索引
s.index

# 修改索引
s.index = ["first", "second", "third"]
```

指定索引标签

在创建 Series 时，可以通过 index 参数提供自定义义的标签列表。

修改现有索引

Series 的索引可以被整体替换，通过直接赋值给给 s.index 来更改整个索引。

OUTPUT (S.INDEX)

```
Index(['first', 'second', 'third'], dtype='object')
```

Series 中的选择（概览）

```
# 示例 Series
s = pd.Series([4, -2, 0, 6], index=["a", "b", "c", "d"])

# 1. 单个标签选择
val = s["a"]

# 2. 标签列表选择
subset = s[["a", "c"]]

# 3. 布尔条件过滤
mask = s > 0
filtered = s[mask]
```

单个标签 (Single Label)

使用单个字符串作为索引键。如果标签存在，返回该标签对应的单个标量值。

标签列表 (List of Labels)

传入一个包含多个标签的列表。返回一个新的子 Series，保留列表中指定的顺序。

布尔条件 (Boolean Mask)

先生成一个布尔 Series (Mask)，再将其作为索引，筛筛选出所有为 True 的行。

Series 中的选择：单个标签

```
# 创建演示 Series
s = pd.Series([4, -2, 0, 6],
    index = ["a", "b", "c", "d"])

# 使用单个字符串标签选择
s["a"]

# 返回结果为标量值 4
```

基于标签定位

使用索引中的标签名（如 "a"）而非整数位置来访问数据，这使得代码更具可读性。

返回标量值

当传入单个标签时，pandas 会返回该位置对应的实际数值（标量），而不是返回一个新的 Series 对象。

注意事项

如果请求的标签在索引中不存在，pandas 将会抛出 KeyError 异常。

```
OUTPUT (s["a"])
4
```

Series 中的选择：标签列表

```
# 演示用Series
s = pd.Series([4, -2, 0, 6],
    index = ["a", "b", "c", "d"])

# 使用标签列表进行选择
subset = s[["a", "c"]]

# 查看结果
subset
```

传入列表作为参数

使用方括号内的列表 [...] 来指定多个标签。注意语法中出现了双重方括号：s[[...]]。

返回子 Series

与选择单个标签不同，使用列表选择会返回一个新的 Series 对象，而非单个标量值。

OUTPUT (SUBSET)

```
a 4
c 0
dtype: int64
```

Series 中的选择：条件过滤

```
# 演示用Series
s = pd.Series([4, -2, 0, 6], index=["a", "b", "c", "d"])

# 1. 创建布尔掩码 (Boolean Mask)
mask = s > 0
# mask: [True, False, False, True]

# 2. 使用掩码索引原 Series
result = s[mask]

# 3. 常用一行写法
s[s > 0]
```

创建布尔掩码

首先应用条件（如 s > 0），pandas 会生成一个由 True 和 False 组成的同长度 Series。

逻辑筛选

当使用布尔掩码进行索引时，pandas 仅保留对应位位置为 True 的数据行，自动过滤掉 False 的行。

OUTPUT (S[S > 0])

```
a 4
d 6
dtype: int64
```

DataFrame 是 Series 的集合

核心概念

在 pandas 中，DataFrame 本质上是多个命名列（Series）的有序集合。

所有列（Series）共享同一个 Index（索引）。

行 (Row)：代表一次观测（如：一条学生记录）。

列 (Column)：代表一个特征（如：姓名、成绩）。

组件拆解

以学生成绩表 grades 为例：

Student_ID 是一个 Series **Name** 是一个 Series

Course 是一个 Series **Score** 是一个 Series

注：Series 的复数形式仍然是 Series。

组合在一起构成 DataFrame

↓ 共享的 Index　　　多个 Series 列 ↓

Index	Student_ID (Series)	Name (Series)	Course (Series)	Score (Series)
0	20250101	张伟	数据科学基础	92
1	20250102	李娜	数据科学基础	95
2	20250103	王芳	程序设计基础	88
3	...	...	...	...
5	20250105	陈静	数据结构	85

创建 DataFrame 的方法 (概览)

核心构造函数语法

```
pandas.DataFrame(data, index, columns)
```

通常根据 data 的来源选择不同的方法

1. 从 CSV 文件读取

最常用的方法。直接将外部数据文件加载为 DataFrame， 自动推断列名。

pd.read_csv("filename.csv")

2. 使用列表创建

适合小型数据集或手动输入数据。通常需要显式指定列名(columns)。

pd.DataFrame(list, columns=[...])

3. 从字典创建

非常灵活。支持"列字典"（键为列名，值为列表）或"行记录列表"（类似 JSON）。

pd.DataFrame({"col": [1, 2]})

4. 从 Series 组装

将现有的多个 Series 组合成一个 DataFrame。它们会自动根据索引对齐。

pd.DataFrame({"a": s1, "b": s2})

学习建议： 理解想要做什么比死记硬背语法更重要。优秀的数据科学家也经常查阅文档！

从 CSV 文件创建 DataFrame

```
import pandas as pd

# 读取清华学生成绩数据文件
# read_csv 默认将第一行作为列名
grades = pd.read_csv("data/grades.csv")

# 指定某一列作为索引（可选）
# grades = pd.read_csv("...", index_col="Student_ID")

# 查看 DataFrame 的前几行
grades.head()
```

读取 CSV 文件

pd.read_csv() 是最常用的加载数据方式。它会自动推断数据据类型并解析表头。

索引列 (Index Column)

如果不指定 index_col，pandas 会默认创建一个从 0 开始的整整数索引。

OUTPUT (GRADES.HEAD())

Student_ID	Name	Course	Score
20250101	张伟	数据科学基础	92
20250102	李娜	数据科学基础	95
20250103	王芳	计算机程序设计...	88

使用列表和列名创建 DataFrame

```python
import pandas as pd

# 方式一：一维列表（单列）
scores = pd.DataFrame([92, 95, 88], columns=['Score'])

# 方式二：嵌套列表（List of Lists）- 每行一个子列表
grades = pd.DataFrame([
[20250101, '张伟', '数据科学基础', 92],
[20250102, '李娜', '数据科学基础', 95],
[20250103, '王芳', '计算机程序设计基础', 88]
], columns=['Student_ID', 'Name', 'Course', 'Score'])

# 查看结果
grades
```

一维列表创建单列

如果传递简单的列表，pandas 默认创建单列。务必指定 columns 参数赋予列名。

嵌套列表代表"行"

在嵌套列表中，每个内部列表代表 DataFrame 的一行 (Row)。数据的顺序必须严格对应 columns 列表中的列名顺序。

OUTPUT (GRADES)

	Student_ID	Name	Course	Score
0	20250101	张伟	数据科学基础	92
1	20250102	李娜	数据科学基础	95
2	20250103	王芳	计算机程序设计基础	88

从字典创建 DataFrame（按列）

```
import pandas as pd

# 字典：键为列名，值为数据列表（长度需一致）
grades = pd.DataFrame({
    'Student_ID': [20250101, 20250102, 20250103],
    'Name': ['张伟', '李娜', '王芳'],
    'Course': ['数据科学基础', '数据科学基础','计算机...'],
    'Score': [92, 95, 88]
})

# 查看结果
grades
```

直观的列定义

这是最常用的构造方式。字典的 Key 直接对应 DataFrame 的列名，Value 则是列是列数据的列表。

数据对齐要求

字典中所有列表的长度必须严格一致。如果长度不匹配，pandas 会抛出 ValueError。

默认索引

如果不指定 index 参数，会自动生成从 0 开始的整数索引。

从字典创建 DataFrame（按行）

```
# 列表中每个元素是一个“记录”（字典）
import pandas as pd

grades = pd.DataFrame([
{
'Student_ID': 20250101, 'Name': '张伟',
'Course': '数据科学基础', 'Score': 92
},
{
'Student_ID': 20250102, 'Name': '李娜',
'Course': '数据科学基础', 'Score': 95
},
{
'Student_ID': 20250103, 'Name': '王芳',
'Course': 'C++程序设计', 'Score': 88
}
]).set_index("Student_ID")

grades
```

类 JSON 结构

这种列表嵌套字典的结构与 JSON 数据格式非常相似，适合直接处理从 API 接口获取的数据。

键即列名

字典的键（Key）会自动转换为 DataFrame 的列名。无需显式指定 columns 参数。

DATAFRAME OUTPUT

Student_ID	Name	Course	Score
20250101	张伟	数据科学基础	92
20250102	李娜	数据科学基础	95
...	...	...	...

从 Series 创建 DataFrame

```
# 准备 Series 数据 (共享索引)
s_name = pd.Series(["张伟", "李娜"], index=["r1", "r2"])
s_score = pd.Series([92, 95], index=["r1", "r2"])

# 方式一：字典组装多个 Series
df = pd.DataFrame({
    "Name": s_name,
    "Score": s_score
})

# 方式二：单个 Series 转 DataFrame
name_df = s_name.to_frame(name="Student")

print(df)
```

索引自动对齐

当使用 Series 字典创建 DataFrame 时，pandas 会自会自动根据 Series 的 index 进行对齐。缺失的索引引位置会自动填充 NaN。

to_frame() 方法

Series 是单维对象。使用 s.to_frame() 方法可以将其其快速转换为单列的 DataFrame，常用于数据重塑。
。

OUTPUT (DF)

Name Score r1 张伟 92 r2 李娜 95

创建 DataFrame 的方法小结

核心回顾：DataFrame 的本质

多个 Series + 共享索引 (Shared Index)

常用构造途径

CSV 文件

数据加载

`pd.read_csv()`

列表 (List)

简单构造

`pd.DataFrame(list)`

字典 (Dict)

结构化数据/API

`from_dict / records`

Series 组装

列对齐合并

`{col: series}`

选型建议

文件读写： 首选 read_csv / to_csv， **最常用的数据入口。**

结构化数据 (JSON/API)： 优先使用 **字典 或** 记录列表，结构清晰。

已有数据列： **使用** Series 组装，利用 pandas 自动对齐索引的特性。

索引不一定是行号

核心概念

索引(Index)不仅仅是默认的 0..n-1 行号。

索引可以是任意标签：字符串、日期、ID等。

索引具有语义：**通常用于**标识行（最好唯一）

可以通过 index.name **属性**为索引命名。

思考

在清华教务系统中，查找一名学生是按照"**第300个注册的**的人"查找，还是按照"**学号**(Student_ID)"查找？显然，学号作为索引更具实际意义。

默认索引 (RangeIndex) 位置编号

	Student_ID	Name	Course	Score
0	20250101	张伟	数据科学基础	92
1	20250102	李娜	数据科学基础	95

```
set_index("Student_ID")
```

↓

自定义索引 (Index) 实际标签

Student_ID	Name	Course	Score
20250101	张伟	数据科学基础	92
20250102	李娜	数据科学基础	95

** 索引变成了学号，更便于检索*

索引不一定唯一

```python
import pandas as pd

# 创建成绩单，同一学生多门课
grades = pd.DataFrame([
    {"Name": "张伟", "Course": "数据科学", "Score": 92},
    {"Name": "张伟", "Course": "数据结构", "Score": 88},
    {"Name": "李娜", "Course": "统计学", "Score": 95},
])

# 将姓名设为索引 -> 产生非唯一索引
by_name = grades.set_index("Name")

# 查看结果
by_name
```

允许重复标签

pandas 的索引不要求唯一。在实际数据中，一个实体（如学生"张伟"）可能有多个观测记录。

选择操作的变化

当索引不唯一时，基于标签的选择（如 .loc["张伟"]）将返回多行数据（DataFrame），而非单行（Series）。

OUTPUT (BY_NAME)

	Course	Score
张伟	数据科学	92
张伟	数据结构	88
李娜	统计学	95

修改索引 (set_index)

```
import pandas as pd
# 假设 grades DataFrame 已加载（含学号、姓名、成绩）

# 将某一列设为索引（返回新 DataFrame）
by_id = grades.set_index("Student_ID")
by_id.index.name    # 查看新的索引名

# 保留原列：drop=False
# 默认 drop=True 会把该列从数据区移除
by_name_keep = grades.set_index("Name",
                drop=False)

# 原地修改（不推荐初学者使用）：
# grades.set_index("Student_ID", inplace=True)
```

索引变换

set_index **方法将**现有的列转换为行索引（Index）。这允许我们通过数据的语义标签（如学号或姓名）**而非数字位置来**访问行。

保留原列

默认情况下，成为索引的列会从数据列中删除。使用 drop=False **可以同**时保留该列在数据区域中。

OUTPUT (BY_ID.INDEX.NAME)

'Student_ID'

STRUCTURE PREVIEW

Index: 20250101, ... | Cols: Name, Course, Score

重置索引 (reset_index)

```
# 假设 grades 已将 Student_ID 设为索引
grades = grades.set_index("Student_ID")

# 1. 恢复默认索引 (旧索引变回列)
reset_df = grades.reset_index()

# 2. 恢复索引并丢弃旧索引列
reset_drop = grades.reset_index(
    drop=True)

# 3. 链式操作：修改后再重置
df = df.set_index("Name").reset_index()
```

恢复默认 RangeIndex

reset_index() 将索引重置为默认的 0 到 n-1 序列。这在索引变得混乱或需要用数字位置访问行时非常有用。

处理旧索引列

默认情况下，旧的索引会作为新列添加到 DataFrame 中。若希望彻底丢弃旧索引，需设置参数 drop=True。

OUTPUT (RESET_DF.INDEX)

```
RangeIndex(start=0, stop=3, step=1)
```

列名通常唯一

为什么要保持唯一性？

避免歧义：如果有两列都叫 "Score"， pandas 将无法区分你要操作哪一列。

选择语义：非唯一列名会导致 df['Score'] 返回一个 DataFrame 而不是 Series， 破坏后续代码逻辑 。

命名规范建议

使用英文与下划线：避免中文、特殊字符或空格。 Final Score 成绩 Final_Score

风格统一：推荐使用 snake_case（蛇形命名法）。

避免关键字：不要使用 python 关键字或 pandas 方法名作为列名。

常用操作

重命名特定列

使用 rename 方法映射旧名到新名

```
# 将 "Score" 改为 "Final_Score"
df = df. rename( columns={
            "Score"  : "Final_Score"     })
```

批量设置列名

直接覆盖 columns 属性（长度需一致）

```
# 重新设置所有列名
df. columns
= [ "Student_ID"  , "Name" , "Course"  , "Score"  ]
```

获取索引、列名与形状

```
# 准备数据 (假设已设置 Student_ID 为索引)
grades = pd.DataFrame([...])
.set_index("Student_ID")

# 1. 提取行标签 (Index 对象)
row_idx = grades.index

# 2. 提取列标签 (Index 对象)
col_names = grades.columns

# 3. 获取数据维度 (行数, 列数)
shape = grades.shape
```

索引与列名

.index 和 .columns 属性分别返回行标签和列标签的 Index 对象。

数据形状

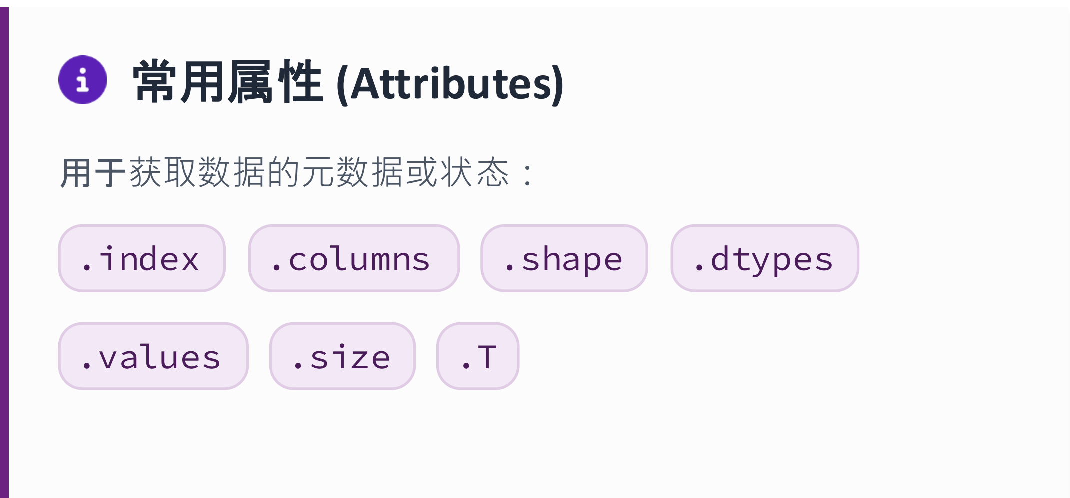
.shape 返回一个元组 (n_rows, n_cols)，用于快速查看数据规模。

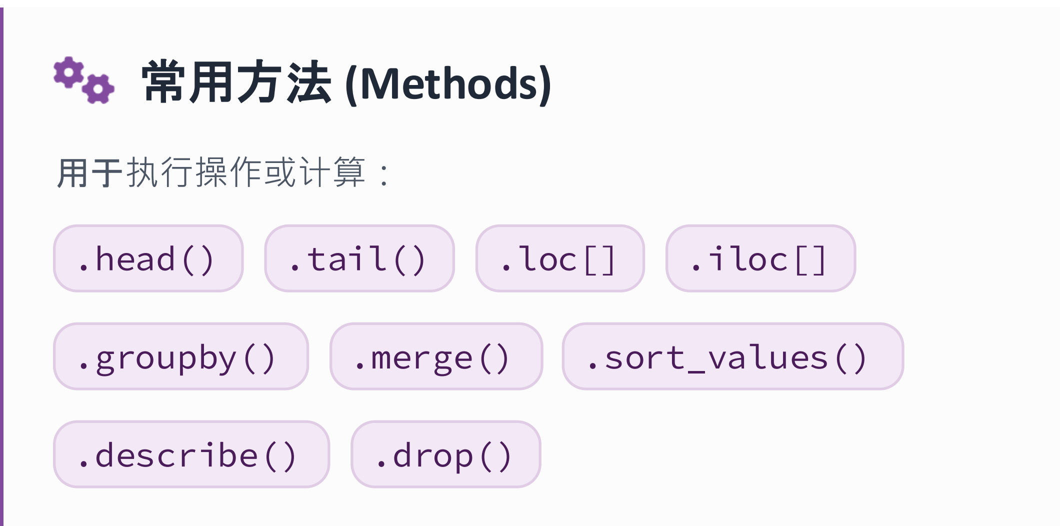
INTERACTIVE OUTPUT

```
>>> row_idx
Index([20250101, 20250102, ...],
name='Student_ID')

>>> col_names
Index(['Name', 'Course', 'Score'], dtype='object')
dtype='object')

>>> shape
(3, 3)
```

DataFrame API 简介

DataFrame 类的 API（应用程序接口）极其丰富，包含了大量的属性和方法，赋予了 pandas 强大的数据处理能力。





官方文档

pandas.pydata.org/docs/reference/api/pandas.DataFrame.html

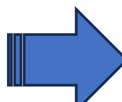
学习建议

在实际科研或工程中，随时查阅文档是常态。相比死记硬背每个方法的参数，理解核心概念（如索引对齐、向量化操作）更为重要。

提纲

Series, DataFrames, and Indices

Pandas 的核心数据结构



数据提取

使用 .loc, .iloc, 和 [ ] 进行数据筛选与操作

数据提取简介

在 pandas 中，我们主要使用三种方式来访问和提取 DataFrame 中的数据。选择合适的工具对于代码的可读性和稳健性至关重要。

快速浏览

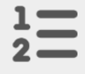
`.head(n) / .tail(n)`

用于快速查看数据的前几行或后几行，了解数据概貌、列名和基本结构。

基于标签 (Label)

`.loc[row, col]`

最推荐使用。根据索引标签 (Index）和列名（Columns）进行提取，语义清晰，不易出错。



基于位置 (Position)

`.iloc[row, col]`

根据行和列的整数位置（下标 0 到 n-1）进行提取，类似于 Python 原生列表切片。

本节示例数据设定

我们将继续使用 “清华学生成绩” DataFrame (grades)。 关键设定：已将 Student_ID (学号) 设置为索引 (Index)， Name, Course, Score 为列。
Score 为列。

目标：在不同情景下，选择最清晰、最稳健的数据提取方式。

数据提取：.head() 与 .tail()

```python
import pandas as pd

# 准备数据(假设已加载并设置Student_ID 为索引)
grades = pd.DataFrame([...]).set_index("Student_ID")

# 1. 提取前3 行(头部)
grades.head(3)

# 2. 提取最后2 行(尾部)
grades.tail(2)

# 3. 默认提取前5 行
grades.head()
```

快速预览 (.head)

使用 .head(n) 返回前 n 行。若不指定参数，默认返回前 5 行。常用于快速检查数据结构、列名和数据类型。

检查尾部 (.tail)

使用 .tail(n) 返回最后 n 行。在处理时间序列数据或检查文件读取是否完整时非常有用。

INTERACTIVE OUTPUT (GRADES.HEAD(3))

Student_ID	Name	Course	Score
20250101	张伟	数据科学基础	92
20250102	李娜	数据科学基础	95
20250103	王芳	程序设计基础	88

.loc 简介（基于标签的选择）

核心语法

df.loc[行标签, 列标签]

↑ 来自 df.index　　↑ 来自 df.columns

支持的参数形式

列表
['a', 'b', 'c']
选定特定多个

切片
'a':'c'
⚠ **包含右端点！**

单个值
'a'
单个标签

冒号
:
选择所有行或列

.loc vs .iloc

.loc — **标签 (Labels)**
索引名、列名（如 "Name", 20250101）

.iloc — **位置 (Integers)**
从 0 开始的整数下标（如 0, 1, 2）

关键提示

使用 .loc 进行切片时（如 "A":"C"），结果包含结束标签 "C"。这与 Python 标准列表切片（不包含右端点）完全不同，是 pandas 的特殊设计。

.loc：使用列表（多行多列）

```
import pandas MŒpd

# 假设 grades 索引为 Student_ID （行标签 ）
# 同时传入行标签列表和列标签列表
subset =grades. loc [
    [ 20250101 , 20250103 ], # 行标签列表
    [ "Name" , "Score" ] # 列标签列表
]

subset
```

精确点名

通过向 BÕŃLĹ 传入两个列表，**可以同**时指定需要的特**定行和特定列**。这是从大型数据集中提取不连续数**据的最直接方法**。

保留列表顺序

返回结果的顺序将严格遵循你传入列表的顺序，而**不是原始数据的**顺序。这使得你可以方便地按需重**新排列数据**。

OUTPUT (SUBSET)

```
          Name  Score
20250101  张伟     92
20250103  王芳     88
```

.loc：使用切片 (右端点包含)

```
import p andas as pd

# 对行标签做闭区间切片
# 对列标签做闭区间切片
part =grades.loc[
    20250101 : 20250103 ,
    # 行: 20250101 到 20250103

    "Course" : "Score"
    # 列: Course 到 Score
]

part
```

右端点包含 (Inclusive)

基于标签的切片操作会包含结束标签。这与 Python 原生的列表切片（不包含右端点）是完全不同的。

连续区间选择

非常适合选取连续的数据块，例如按时间范围、ID 序列或表格中相邻的列名区间进行批量提取。

OUTPUT (PART)

```
St u dent _ID      Cou r s e   Scor e
20250101   数据科学基础    92
20250102   数据科学基础    95
20250103   程序设计基础    88
```

.loc：提取所有行或所有列

```
import pandas as pd
grades =... # 假设已设置 Student_ID 为索引

# 1. 选取所有行，指定列
# 语法：df.loc[:, 列标签列表]
a = grades.loc[:, ["Name", "Score"]]

# 2. 指定行，选取所有列
# 语法：df.loc[行标签列表, :]
b = grades.loc[[20250102, 20250105], :]

# 3. 简写形式（隐式选取所有列）
# 当省略第二个参数时，另一维(列)默认为全选
c = grades.loc[[20250102, 20250105]]
```

全选操作符

冒号 : 在 pandas 切片语法中代表"全选"。它可以用于行维度，也可以用于列维度。

隐式列选择

如果只提供一个参数（行选择器），pandas 会默认选取该该行对应的所有列。即 df.loc[rows] 等价于 df.loc[rows, :]。。

注意维度顺序

第一个参数始终控制行，第二个参数控制列。若要A取所有行、特定列A，必须显式写出第一个冒号。

.loc：单个值与单列选择

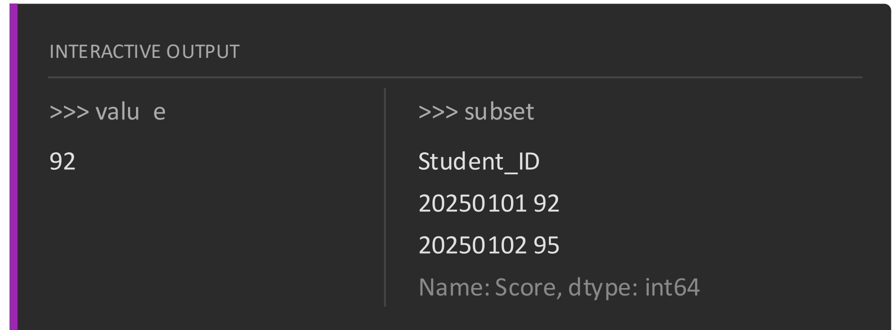
```
import pandas as pd
# 假设 grades 索引为 Student_ID

Á ĈB提取单个标量值 Ǻ行标签 Ą 列标签Å
value = grades.loc[20250101, "Score"]

Á ČB提取单列多行 Æ 返回 Ĭ ÑØÒÑŒ
subset = grades.loc[
    [20250101, 20250102],
    "Score"
]
```

返回标量 (Scalar)

当同时指定单个行标签和单个列标签时，.loc 返回该单元格的基础数据类型值（如整数 92）。

返回 Series

如果其中一个维度是单个标签，而另一个维度是列表或切片（多项），结果将降维返回一维的 Series 对象。



BOON 实践演示

```
import pandas as pd

# 1. 指定索引区间，选择关心的列 (切片包含右端点)
demo1 = grades.loc[20250101 : 20250105,
    ["Name", "Course", "Score"]]

# 2. 按学号列表取子集 (精确点名)
top_ids = [20250101, 20250103]
demo2 = grades.loc[top_ids]

# 3. 按列名区间批量选择 (全选行)
demo3 = grades.loc[:, "Name" : "Score"]

# 建议：先预测返回结果，再运行验证
```

标签选择更稳健

即使数据行重新排序，基于标签的代码（如 loc[ID]）仍然指向正确的数据实体，而基于位置的索引可能会指向错误的行。

预测返回类型

如果是切片或列表，结果通常是 DataFrame（保留二维结构）；如果是单个标签，结果可能是 Series 或标量。

数据提取小结（.loc）

</> 核心语法与参数

语法：df.loc[row_labels, column_labels]

- 列表 List：精确选择多个标签 (['a', 'c'])
- 切片 Slice：右端点包含 ('a':'c' 取 a, b, c)
- 单个值 Scalar：单个标签 ('a')
- 冒号 : 代表该维度“全选”

返回类型速查

标量 Scalar
单个行标签 + 单个列标签

Series (1D)
单行多列 或 多行单列

DataFrame (2D)
多行 + 多列 (或切片)

为什么推荐优先使用 BÕŃ?

特性	.loc (Label)	.iloc (Integer)
可读性	高 (语义明确)	低 (数字索引)
稳健性	高 (不受排序影响)	低 (依赖顺序)

建议： 在科研和工程实践中，为了代码的长期维护性和可读性，应优先使用 .loc 表达“语义明确”的数据提取逻辑。

.iloc 简介 (基于整数位置)

CORE SYNTAX

```
df.iloc[row_integers, column_integers]
```

通过整数位置 (Integer Position) 来选择数据，完全忽略标签。

核心特性

从 Ć 开始计数

行和列的索引都从 0 开始（Python 标准习惯），与具体的 Index 名称无关。

切片右端点不包含

遵循 Python 列表切片规则：start:stop 包含 start， 不包含 stop。

灵活的返回类型

根据选择维度返回：标量 (单值)、Series (单行/列) 或 DataFrame (多行多列)。

.loc vs .iloc 对比

特性	.loc (Label)	[illegible]
依据	标签 (Index/Columns)	整数位置 (0, 1, 2...)
切片规则	包含右端点 Inclusive	不包含右端点 Exclusive (Pythonic)
适用场景	业务逻辑清晰， 语义明确时使用	基于固定位置， 或与切片/循环配合时使用

可视演示: [0:3]　0　1　2　3

.iloc：使用列表

```python
# 假设已存在 grades， 列顺序为 [Name, Course, Score]
# 0: 张伟, 1: 李娜, 2: 王芳

# 选择第1、3 行 (索引 0, 2)
# 选择第1、3 列 (索引 0, 2 -> Name, Score)
subset = grades.iloc[[0, 2], [0, 2]]

# 查看结果
subset
```

精确点名 (整数位置)

行和列都可以使用整数列表来精确指定。与 .loc 不同同，这里使用的是位置下标（0, 1, 2...），而非标签名。

返回 DataFrame

当行和列都使用列表选择时，结果是一个二维的 DataFrame。列表中的顺序决定了结果中行列的显示示顺序。

INTERACTIVE OUTPUT

	Name	Score
0	张伟	92
2	王芳	88

.iloc：使用切片（右端点不包含）

```
# 行与列均可使用切片
# 注意：右端点不包含（遵循 Python 标准）

# 取第1-3 行（位置索引0, 1, 2）
# 取第1-2 列（位置索引0, 1 -> Name, Course）

part = grades.iloc[0:3, 0:2]

part
```

半开区间规则

.iloc 的切片语法与 Python 原生列表切片完全一致：start:stop，不包含结束位置 stop。

批量位置选择

非常适合处理不需要关心具体列名，只需提取á 前í 列â 或á 中间某段行â 的场景，这与 ВŐŃ的闭区间行为形成鲜明对比。

OUTPUT (PART)

Name Course 0 张伟 数据科学基础 1 李娜 数据科学基础 2 王芳 计算机程序设计基础

.iloc：提取所有行或所有列

```
# 所有行, 取前两列 (Name, Course)
a = grades.iloc[:, 0:2]

# 指定行区间，取所有列
b = grades.iloc[1:4, :]

# 只有一个参数时，另一维默认为冒号（所有列）
c = grades.iloc[[1, 3]]

# 等价于 grades.iloc[[1, 3], :]
```

冒号 "全选"

冒号：单独使用时表示选中该维度上的所有元素。常用于固定一维（如取所有行）而筛选另一维。

简写形式

如果只提供一个参数，pandas 默认将其视为行选择器，并隐含地选择所有列。

OUTPUT (C)

Name Course Score 1 李娜 数据科学基础 95 3 刘洋 统计学导论 90

BÒÕÕŃ：单个值与单列选择

```
# 1. 单个标量值 (行位置 + 列位置)
# 获取第1行(索引0)第3列(Score,索引2)的值
val  =grades.  iloc  [ 0, 2]

# 2. 单列 + 多行 -> 返回 Series
# 获取第2-3行的第2列(Course)
# 注意：使用列表 [1, 2] 指定多行
sub =grades.  iloc  [[ 1, 2], 1]

# 3. 整列 -> 返回 Series
# 获取所有行，第1列(Name)
col  =grades.  iloc  [:,  0]
```

返回标量 (Scalar)

当行和列都指定为单个整数时，pandas 会进行“降维”操作操作，直接提取出该位置的具体数值（如 int, float, str）。

。

返回 Series (一维)

如果其中一个维度是单个整数（锁定某一行或某一列），而另一个维度是列表、切片或冒号，则返回 Series 对象。

INTERACTIVE OUTPUT

```
>>> val
92
>>> type(sub)
pandas.core.series.Series
```

.iloc 实践演示

```
# 1. 取中间一行 (基于长度动态计算)
mid = grades.iloc[ len (grades) // 2]

# 2. 取最后两行 (负索引切片)
last_two = grades.iloc[ -2 :]

# 3. 间隔抽样 (每隔一行取一行)
ever y_ot her = grades.iloc[:: 2]

# 4. 组合选择 (前3行 + 指定列位置)
subset = gr ades .iloc[ 0: 3, [ 0, 2]]
```

位置计算

.iloc 使得基于位置的数学计算变得简单，例如提取中位数位置、四分位数位置的数据。

灵活切片

完美支持 Python 原生切片语法：start:stop:step，轻松实现反转、间隔抽样等操作。

INTERACTIVE OUTPUT (LAST_TWO)

```
Name Course Score
Student_ID
20250104 刘洋 统计学导论 90
20250105 陈静 数据结构 85
```

.loc vs .iloc 对比与选择建议

.loc

标签驱动

选择维度

基于标签 Ă MNÑŎŒĂĤŌŇŇŘ Ä FŎŎÞÖ Ō
Í MÖ ÑŒ

切片规则

右端点 包含 (Inclusive)['a':'c'] -> a, b, c

可读性

高语义明确，易于理解业务逻辑

稳健性

高不怕行/列顺序打乱

.iloc

位置驱动

选择维度

基于整数位置 (Integers)0, 1, 2...
(Position)

切片规则

右端点 不包含 (Exclusive)[0:3] -> 0, 1, 2

可读性

中等需要知道数据具体位置

稳健性

较低依赖数据排序， 易受插入C删除影响

实践建议 (Best Practices)

首选语义明确的操作

如果数据有明确的业务标签（如学号、姓名、日期），.loc 是最佳选择，代码更易读且不易出错。

特定位置操作

当需要"取前10行"、"取倒数第2行"、"每隔5行取样"时，使用 .iloc 更为自然和高效。

方括号 [] 提取（上下文相关）

> [] 是 pandas 中最常用的选择操作符，但它是上下文相关的：它的具体行为取决于你传入参数的类型。

方法对比

.loc 显式基于标签 (Row + Col)

.iloc 显式基于整数位置 (Row + Col)

[] 隐式推断：切片→行位置；列表/值→列标签

⚠ 新手易错点：双层方括号

如果要选择多列，必须传入一个列表，因此需要两层括号：

df[["A", "B"]]

df["A", "B"]

[] 使用切片

```
# 假设 grades 索引为 Student_ID

# 1. 取第 2-4 行 (位置 1, 2, 3)

top_mid  =ŊǾM̌ŇŒ 1: 4]

# 2. 负索引：取最后 2 行

last_two   =ŊǾM̌ŇŒ - 2:]

# 3. 步长：每隔一行取一行

every_other   =grades[::    Č]
```

基于整数位置的切片

[] 中的切片被解释为对行进行整数位置切片，行为与 Python 列表一致。

右端点不包含

重要：区间 [a:b] 包含位置 a，但不包含位置 b。这与 .loc（右端包含）的行为完全相反！

灵活性（负索引与步长）

支持负索引（如 -2:）和切片步长（如 ::2），非常适合快速浏览数据。

[] 使用列表（指定列）

```
# 1. 准备列名列表
cols = [ "Name" , ÀĬ ŃŎǾÑÀ ]
subset = grades[cols]

# 2. 直接写法：注意双层方括号
subset2 = ŊǾMŇ ÑŒL "Score" , "Name" ]]

# ❌ 常见错误：忘记内层列表括号
# grades["Name", "Score"]        # KeyError
```

提取多列

传入一个字符串列表 list，返回一个新的 DataFrame。

双层方括号 LL BBĽĽ

外层 [] 是 pandas 的索引操作符，内层 [] 是 Python 的列表语法。

顺序保留

结果中的列顺序将严格遵循列表中指定的顺序。

[] 单列选择：Series vs DataFrame

```
Á ĈB单个方括号Æ 返回ĬÑØÒÑŒ Ă一维Å
name_s =grades [ "Name" ]

# 2. 双重方括号 -> 返回 DataFrame (二维)
Á 这里的Ш 是提取操作符
# 里面的[] 是列名列表 ["Name"]
name_df =grades [[ "Name" ]]

Á 区别：
# Series 用于一维向量计算
# DataFrame 用于保持表格结构
```

返回 Series (一维)

使用 ["col"]。得到一维数组对象，适合进行数学运运算或统计分析。

返回 DataFrame (二维)

使用 [["col"]]。即使只有一列，也保留了二维表格结构，方便与其他 DataFrame 合并。

CHECK TYPES

```
t yp e(name_s ) p andas .Ser ies
type(name_df) pandas.DataFrame
```

[] 实践演示

```
# 1. 快速抽样浏览 (前3 行)
head3 = grades[: Ċ]

# 2. 中间区间 (位置 1 到 3， 右不含)
mid = grades[ 1: Ç]

# 3. 指定多列 (注意双层括号)
name_score = grades[[ "Name" , ÀĬ ŃŎǾÑÀ ]]

# 4. 布尔条件筛选 (分数 >= 90)
high_score = grades[grades[ "Score" ] Ë Ě90 ]
```

行切片 (Slicing)

最常用的快速浏览方式。[] 内为整数切片时，行为类似列表切片（右端不含）。

列选择 (Columns)

提取特征子集。传入列表时，返回 DataFrame；传入单字符串时，返回 Series。

条件筛选 (Filtering)

虽然 Lľ 接受布尔掩码，但复杂逻辑建议使用 ĕŐŃ 以提高可读性。

为什么使用 [] ？

简洁高效

相比 [illegible] 和 [illegible]， 方括号语法更加简洁， 显著减少代码输入量
。

推荐： `grades["Name"]`

繁琐： `grades.loc[:, "Name"]`

探索性数据分析 (EDA)

在数据探索阶段，我们通常需要快速查看数据分布、抽取特定**列**进行统计。

- [] 是 Jupyter Notebook 中最常用的交互式操作符。

注意事项与实践建议

⚠ 语义依赖上下文

初学者容易混淆：

- 输入切片 1:5 → 行选择
- 输入列表 ["Name"] → 列选择

💡 最佳实践

列操作：优先使用 []， 简单直观。

复杂逻辑：涉及精确的 **行+列 控制**时，请使用 .loc **提升代**码可读性与稳健**性**。

课程总结：Pandas 第一部分

Pandas 核心介绍

- Ĩ ŘPOŎŌ 数据科学的标准工具库
- 核心能力：表格数据组织、清洗与分析
- 基于 NumPy 构建，支持向量化计算
- 支持多种数据源：CSV, Excel, SQL, JSON

核心数据结构

- Series 一维数组 + 索引（值序列+标签）
- DataFrame 二维表格（Series 的有序集合）
- 共享索引机制：行标签 (Index) 标识观测
- 列标签 ĂF ŎŌÞ Ö ŌŒÅ 标识特征变量

G MPMĞØMÖ Ñ 创建

- 文件读取：pd.read_csv()
- 从字典(按列)：{'Col': [val1, val2]}
- 从字典(按行)：[{'Col': val}, ...]
- 从列表：需手动指定 columns 参数

索引操作

- 索引特性：不一定是数字，不一定唯一
- ŒÑPŁŎÕŇÑŔĂÅ: 将某列设为索引（语义化）
- ØŃŒÑPŁŎÕŇÑŔĂÅ: 恢复默认 ĆBŌÆ 索引
- 索引是数据的"行标签"，而非简单的行号

数据提取方法

- .loc[]：基于标签 (Label-based)
- .iloc[]：基于整数位置 (Integer-based)
- []：上下文相关 (Context-dependent)
- 切片差异：.loc 包含右端点，.iloc 不包含

方法选择建议

- 优先使用 ĦŎŎŃ：语义明确，代码稳健
- 仅在位置处理时使用 .iloc
- 简单列选择可用 [] 提高效率

最佳实践：明确你的意图是"标签"还是"位置"！