会计大数据管理分析

暨南大学 MPAcc 2025 级

王新路

2025 年 11 月 04 日

Table of contents

课程信息

教师信息

- 王新路 (8) + 谭有超 (4) = 12
- Office: 管理学院 603 室
- Office Hour: 预约

课程内容

1. Python 基础
2. Pandas 数据分析
3. SQL 应用
4. 数据可视化
5. 简单文本分析
6. 网络爬虫
7. AI 自动化工具的应用

课程考核

形式	说明	分值
教学过程	考勤、讨论、回答问题	15%
课后作业	认真完成	20%
数据分析案例	小组报告	65%

微助教课堂

关注微助教服务号 -> 学生（S）-> 全部（A）-> 加入课堂 -> 输入课堂编号

- 暨南大学会计大数据管理分析 (周一下午): **PN087**
- 暨南大学会计大数据管理分析 (周二下午): **PN088**
- 暨南大学会计大数据管理分析 (周二晚上): **PN089**

课件下载

课程介绍 ppt 下载

1 Python 环境配置

1.1 安装 Python

`Anaconda` 是 Python 发行版本，包含了常用的 Python 包，可以快速安装和管理 Python 环境，适合于数据科学、机器学习、深度学习等领域的用户。`Miniconda` 是 `Anaconda` 的轻量级版本，包含 Python、conda、pip 三个基本包，适合于服务器环境。

- 推荐使用 `Anaconda` 安装 下载，时间比较久耐心等待。
  - 安装的路径上（位置）不要有中文或空格, 每一层文件夹名只能有字母数字和下划线。

1.2 安装 VS Code

- 安装 VS Code: 下载
- 安装 VS code 的 Python 插件:
  - Python Extension 点击安装
  - Jupyter Extension 点击安装

1.3 测试安装

- 打开 VS Code -> `File` -> `New File` -> `Jupyter Notebook`
  - 右上角:`[Select Kernel]` -> `[Select Another Kernel]` -> `[Python Environment]` -> 选择你的 Python 环境
  - 点击 `+ Code` -> 输入 `print("Hello, World!")` 并运行。

1.4 pip 设置阿里云镜像

```
pip config set global.index-url https://mirrors.aliyun.com/pypi/simple/
```

1.5 UV 安装 (可选)

安装uv用于管理 packages 和 python 版本:

1.5.1 simple install

```
pip install uv
```

1.5.2 Or, native install

On macOS and Linux.

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

On Windows

```
powershell -ExecutionPolicy ByPass -c "irm
↪ https://astral.sh/uv/install.ps1 | iex"
```

或者

```
winget install --id=astral-sh.uv -e
```

或者

uv release page: https://github.com/astral-sh/uv/releases 可以下载最新版本的.exe

1.5.3 设置 project 镜像

```
[[tool.uv.index]]
url = "https://mirrors.aliyun.com/pypi/simple/"
default = true
```

1.5.4 设置 User 镜像

C:\Users\sean\AppData\Roaming\uv\uv.toml

~/.config/uv/uv.toml

```
[[index]]
url = "https://mirrors.aliyun.com/pypi/simple/"
default = true
```

1.5.5 uv 简单实用

```
uv init project_name # 创建项目
uv venv # 创建虚拟环境
uv add <package> # 安装包
uv remove <package> # 删除包
uv sync # 同步
```

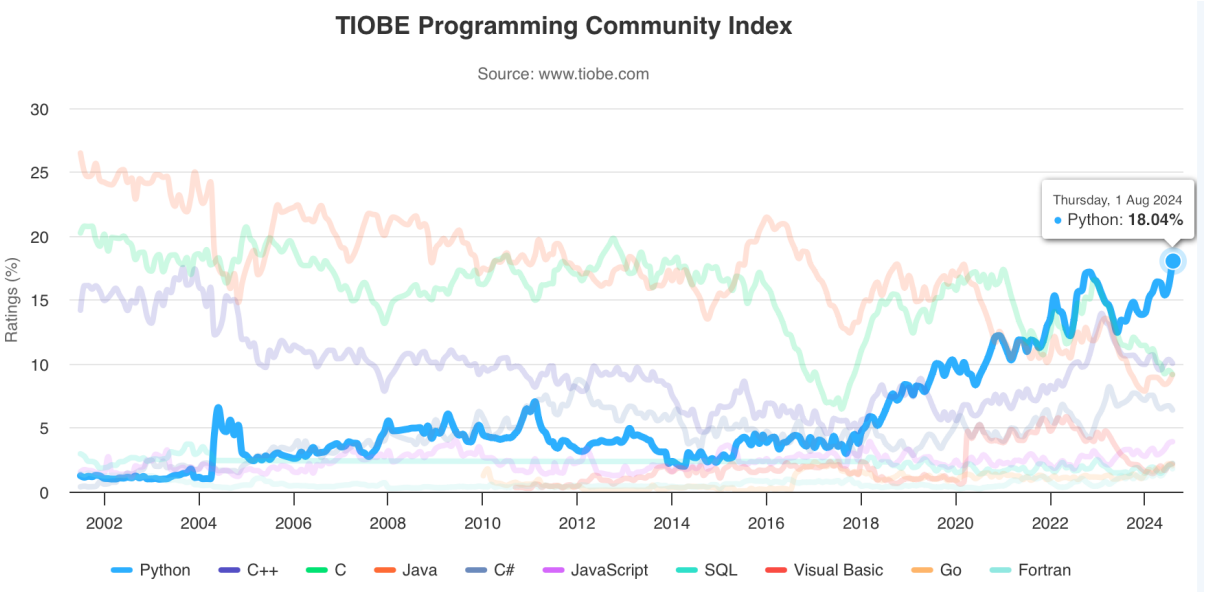
1.6 AI 编程工具 (可选)

- vs code extension: Cline 点击安装
- vs code extension: GitHub.copilot 点击安装
- vs code extension: GitHub.copilot chat 点击安装
- vs code extension: LingMa 点击安装

2 Python 入门

点击下载本章节代码

2.1 Why Python



2.1.1 最受欢迎的编程语言

- 支持性社区
- ChatGPT 等 AI 工具更可靠

2.2 Hello World Demo

让我们开始第一个 Python 程序：

```
1 + 1
```

2

2.2.1 Jupyter Notebook 快捷键

- 执行单元格程序 `Shift + Enter` (`Shift + Return` on a Mac)
- 插入新单元格 `B` (below) 或 `A` (above)
- 删除单元格 `D + D` (按两次 `D`)
- 保存 `Ctrl + S` (`Cmd + S` on a Mac)
- `#` 开始的行为注释，不会被执行

2.3 编程：让机器执行你的指令

2.3.1 现实生活中的指令（煎蛋步骤）

1. 将一个鸡蛋打入小碗中，确保没有蛋壳，放在一旁备用。
2. 在不粘锅中加入黄油或其他油脂，用中高火加热一分钟。
3. 将鸡蛋倒入锅中，煮三到四分钟。
4. 上桌，根据个人口味添加盐和胡椒。

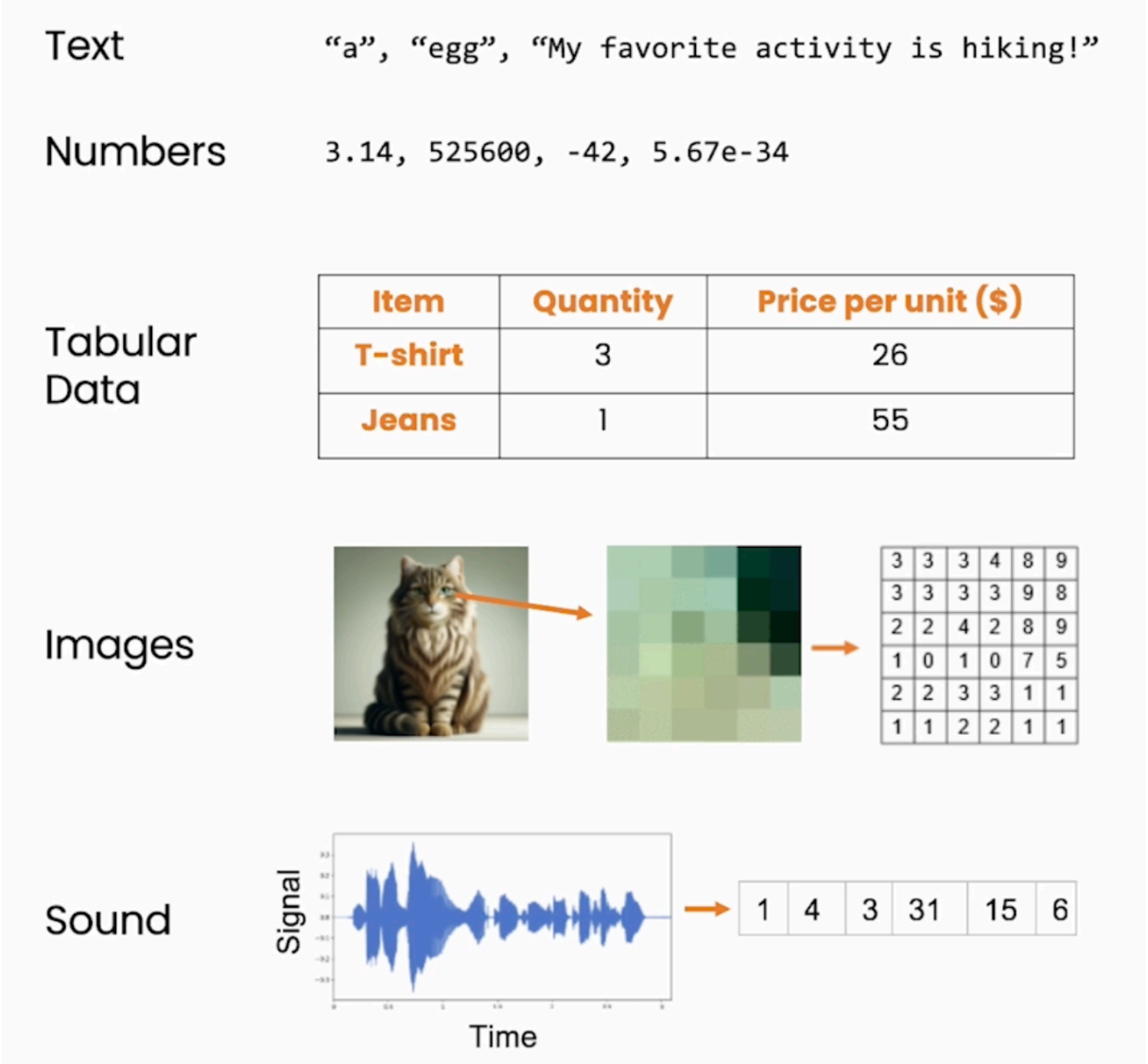
2.3.2 编程中的指令

```
# 赋值给 num1 和 num2
num1 = 37
num2 = 5

# 计算和
sum = num1 + num2

# 展示结果
print(sum)
```

42

A set of instructions to perform a task

2.4 数据类型



编程本质上就是在处理文字和数字。常见的数据类型比如日期、时间、货币、百分比、电话号码、邮件地址、网址等等。

2.4.1 字符串和数字的区别

- 数字用来加减乘除，并且是有意义的
- 字符串用于切片、提取一部分、粘贴一起、打印消息等等

2.5 字符串 Strings 介绍

- Python 中的字符串是由**文本、数字**或**符号字符**组成的任意组合，并且用**双引号**或者**单引号**括起来。
- **引号之间的所有内容**都是字符串的一部分，包括空格。

例如:

- "Hello, world"
- "My favorite drink is Earl Grey tea"
- "_（ツ）_! "
- "2.99"
- " 大数据 Big Data 》。! 你好"

注意: 引号必须是英文的引号 " "，而不是中文的引号 “”。

2.5.1 特殊字符串

```
print("Hello World!")
```

```
Hello World!
```

```
type("Hello World!")
```

```
str
```

```
print(""" 这是一个多行字符串的实例
也可以使用换行符。""")
```

```
这是一个多行字符串的实例
也可以使用换行符。
```

特殊字符

- \n 换行符，例如 print("Hello\nWorld")
  - Windows 系统中, 会使用 \r\n（回车加换行）作为新行的标记。
- \t 制表符; \" 双引号; \' 单引号

注意事项

函数的概念会在后面继续讲。我们可以用 """ 来创建多行字符串。

如果写成 print("Hello"World") 会出现错误:SyntaxError: EOL while scanning string literal。

2.5.2 字符串索引切片

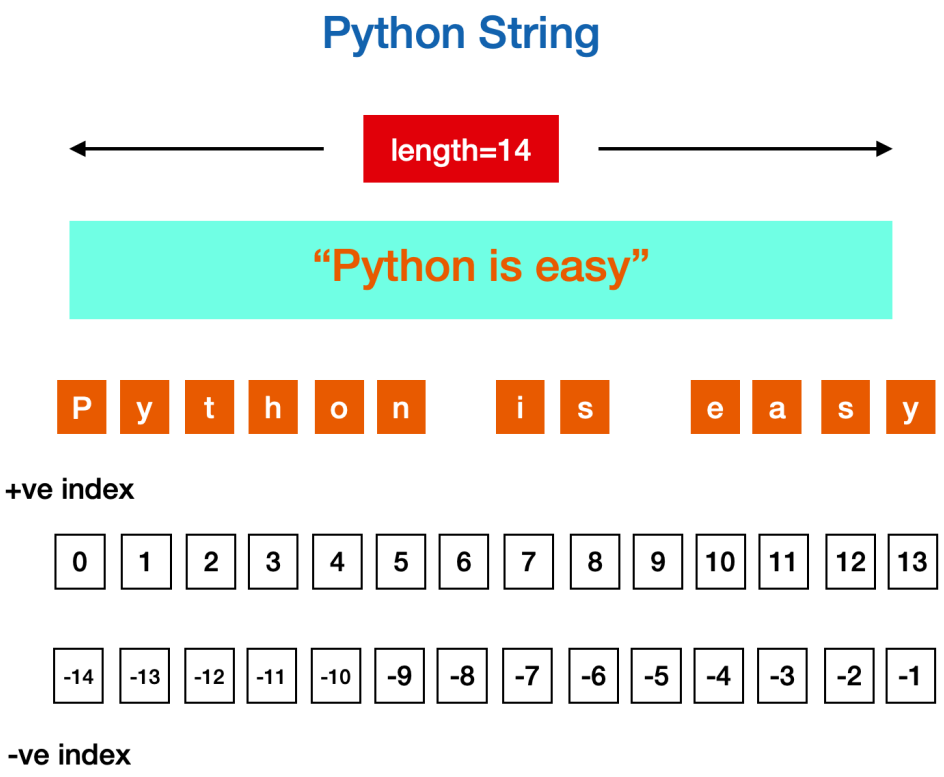


图 2.1: Python String Indexing

语法: string[start:stop:step]

```
a = "Python is easy"
len(a)
```

```
14
```

```
print(a[1])  # y
```

```
y
```

```
print(a[1:4])  # yth
```

```
yth
```

```
print(a[1:4:2])  # yt
```

```
yh
```

2.5.3 字符串基本操作

```
"Hello" in "Hello World!"
```

```
True
```

```
"hello" not in "Hello World!"
```

```
True
```

```
"hello" + " World!"
```

```
'hello World!'
```

```
"hello" * 2
```

```
'hellohello'
```

```
"hello".upper()
```

```
'HELLO'
```

2.6 数字类型

Python 中的数字类型有三种：**整数 (int)**、**浮点数 (float)** 和复数。

- int: `3`，`400`，`-2`，`0`
- float: `3.0`，`3.14`，`-2.5`

2.6.1 数字运算

```
print (10 % 3) # Modulus
print (2 ** 3) # Exponentiation
print (10 // 3) # Floor division
```

```
1
8
3
```

```
print(3 + 2 * (1 - 1) + 2 ** 2 )
```

```
7
```

2.6.2 数字类型检查

```
print(type(3)) # Int
print(type(3.0)) # Float
```

```
<class 'int'>
<class 'float'>
```

2.6.3 类型转换

```
print(float(3)) # Int to Float
print(int(3.0)) # Float to Int
```

```
3.0
3
```

2.6.4 数字字符转换

```
n = 3
print(type(n))
```

```
<class 'int'>
```

```
n_str = str(n)
print(type(n_str))
```

```
<class 'str'>
```

```
s = "3.5"
print(type(s))
```

```
<class 'str'>
```

```
s_float = float(s)
print(type(s_float))
```

```
<class 'float'>
```

2.7 变量

1. 给一个值 (对象 Object) 赋予一个名称以供以后使用
2. 使用相同的名称来引用不断变化的值

2.7.1 变量示例

- `x = 5`
- `y = "Hello, World!"`

2.7.2 变量命名规则

- 只能是一个词，变量名中不能够有 **空格**以及 **标点符号**，习惯上用 _ 表示空格
- 不能以数字开头
- 不能使用 Python 保留字（关键字）e.g., *False True if else import None del,* etc.
- 建议只包含 **字母、数字和下划线**
- Snake Case : `my_variable_name`
- Camel Case : `myVariableName`
- 变量名是区分大小写的

2.7.3 练习：判断合法的变量名?

`balance, current-balance, current balance, true, currentBalance, current_balance, _spam, SPAM, 4account, account4, 100, total_$um, 'guanli'`

2.7.4 注意事项

Python 的变量是动态类型的，这意味着变量的数据类型是根据赋给它的值来确定的。

Python 关键字: `False await else import pass None break except in raise True class finally is return and continue for lambda try as def from nonlocal while assert del global not with async elif if or yield`

2.7.5 变量赋值

```
num = 5
num = 32
```

```
print(num)
```

32

```
num = num + 1
print(num)
```

33

```
num = "Hello"
print(num)
```

Hello

2.8 字符串格式化

```
name = " 小李"
print(" 你好, " + name + "!")
```

你好, 小李!

2.8.1 这样会出错

```
year = 1998
print(" 小李出生于" + year + " 年!")
```

2.8.2 正确的方式

```
year = 1998
print(" 小李出生于" + str(year) + " 年!")
```

小李出生于1998年!

2.8.3 Format string 方式

```
year = 1998
name = " 小李"
print(f" 我是{name}, 出生于{year}年!")
```

我是小李，出生于1998年!

2.8.4 Format String 更多例子

```
print(f"Isabel is {28/7} years old.")
```

```
Isabel is 4.0 years old.
```

格式化小数

问题：python format string 里面如何小数取整?

```
print(f"Isabel is {28/7:.0f} years old.")
```

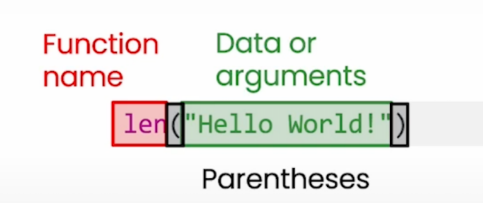
```
Isabel is 4 years old.
```

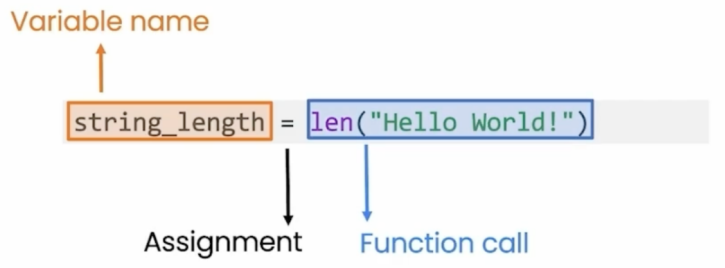
Format String 的巧妙用法

- 对齐：`f"|{name:<10}|{name:^10}|{name:>10}|"`
- 补零：`f"{number:04}"`
- 百分比：`f"{percentage:.2%}"`

2.9 函数

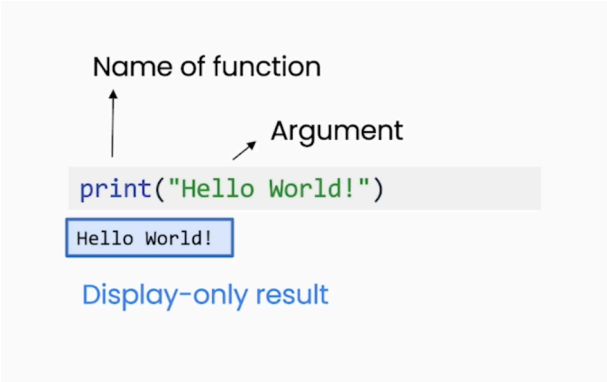
- 函数允许你对数据执行特定 **操作**，并提供（或返回 `return`）结果。
  - `len("Hello World!")`
  - `round(3.14159)`





2.9.1 无返回值函数

- 有些函数不返回结果，只是执行操作。
  - `print("Hello")`



2.9.2 自定义函数

带返回值的函数

```
def add(x, y):
  z = x + y * 2
  return z
```

无参数函数

```
def greet():
  print('Hello World!')
```

带参数函数

```
def greet(name):
  print(f'你好，{name}!')

greet('小李')
```

你好，小李!

注意事项

- 函数也是一个对象，是一个代码块，一个过程，可以重复使用。

- 匿名函数 `lambda x: (x + 1) ** 2`
- `type(greet)` 可以查看函数类型

2.9.3 Lambda 函数（可选）

```
lambda x: (x + 1) ** 2
```

```
<function __main__.<lambda>(x)>
```

```
f = lambda x: (x + 1) ** 2
```

```
type(f)
```

```
function
```

2.10 容器数据类型（container）

类型	例子
列表 list	my_list = [1, 2, 3]
元组 tuple	my_tuple = (1, 2, 3)
集合 set	my_set = {1,3,5}
字典 dict	my_dict = {'David': 25, 'Mark':30}

2.10.1 列表 List

- 列表是一个有序的集合，可以包含不同类型的元素。
- 列表是可变的，可以通过索引进行修改。string 不可以。

基本操作

```
example_list = [1, 2, 3, 4, 5]
print(example_list)
```

```
[1, 2, 3, 4, 5]
```

```
print(example_list[0])
```

```
1
```

```
print(example_list[1:3])
```

```
[2, 3]
```

```
print(example_list[::2])
```

```
[1, 3, 5]
```

修改列表

```
example_list.append(1)
example_list
```

```
[1, 2, 3, 4, 5, 1]
```

```
example_list.insert(0, 111)
example_list
```

```
[111, 1, 2, 3, 4, 5, 1]
```

```
example_list[2] = 10
example_list
```

```
[111, 1, 10, 3, 4, 5, 1]
```

列表练习

```
L = [['Apple', 'Google', 'Microsoft'],
     ['Java', 'Python', 'Ruby', 'PHP'],
     ['Adam', 'Bart', 'Lisa']]

# 打印出 Apple:
# 打印出 Python:
# 打印出 Lisa:
```

解决方案:

```
print(L[0][0])
print(L[1][1])
print(L[2][2])
```

```
Apple
Python
Lisa
```

2.10.2 元组 Tuple

```
example_tuple = (4,5,"test",8)
type(example_tuple)
```

```
tuple
```

```
example_tuple[0]
```

```
4
```

```
example_tuple[0] = 10
```

2.10.3 集合 Set

```
lp = [1,2,3,4,5,5,5,5,5,5,5,5]
set(lp)
```

```
{1, 2, 3, 4, 5}
```

```
set1 = {" 上证综指"," 深圳成指"," 恒生指数",
  " 日经 225 指数"," 道琼斯指数"}
type(set1)
```

```
set
```

```
set2 = {" 标普 500 指数"," 道琼斯指数"," 沪深 300 指数",
  " 日经 225 指数"," 发过 CAC40 指数"," 德国 DAX 指数"}
type(set2)
```

```
set
```

```
set1 | set2 # 并集
```

```
{'上证综指',
 '发过CAC40指数',
 '德国DAX指数',
 '恒生指数',
 '日经225指数',
 '标普500指数',
 '沪深300指数',
 '深圳成指',
 '道琼斯指数'}
```

```
print(set1 & set2)
print(set1.intersection(set2))
```

```
{'日经225指数', '道琼斯指数'}
{'日经225指数', '道琼斯指数'}
```

```
print(set1 - set2) # set1 对 set2 的差集
print(set2 - set1) # set2 对 set1 的差集
```

```
{'深圳成指', '上证综指', '恒生指数'}
{'沪深300指数', '德国DAX指数', '发过CAC40指数', '标普500指数'}
```

```
set1.add(" 德国 DAX 指数")
set1
```

```
{'上证综指', '德国DAX指数', '恒生指数', '日经225指数', '深圳成指', '道琼斯指数'}
```

```
set1.discard(" 日经 225 指数")
set1
```

```
{'上证综指', '德国DAX指数', '恒生指数', '深圳成指', '道琼斯指数'}
```

2.10.4 字典 Dict

```
{key1:value1, key2:value2, key3:value2 ,...}
```

1. 字典中的元素必须以键 (**key**) 和值 (**value**) 的形式成对出现，也就是所谓的键-值存储；
2. **key** 不可以重复，但是 **value** 可以重复；
3. **key** 不可以修改，但是 **value** 可以修改，并且可以是任意的类型。

```
dict1 = {" 指数名称":" 沪深 300",
  " 证券代码":"000300",
  " 交易日期":"2019-01-08",
  " 涨跌幅":-0.0022} # 直接法创建
print(type(dict1))
```

```
<class 'dict'>
```

```
dict2 = {} # 间接法创建
dict2[" 指数名称"] = " 沪深 300"
dict2[" 证券代码"] = "000300"
dict2[" 交易日期"] = "2019-01-08"
dict2[" 涨跌幅"] = -0.0022

print(type(dict2))
```

```
<class 'dict'>
```

```
dict1.keys()
```

```
dict_keys(['指数名称', '证券代码', '交易日期', '涨跌幅'])
```

```
dict1.values()
```

```
dict_values(['沪深300', '000300', '2019-01-08', -0.0022])
```

```
dict1.items()
```

```
dict_items([('指数名称', '沪深300'), ('证券代码', '000300'), ('交易日期', '2019-01-08
```

```
dict1['交易日期'] = "2019-01-07"
dict1
```

```
{'指数名称': '沪深300', '证券代码': '000300', '交易日期': '2019-01-07', '涨跌幅': -0.
```

2.11 流程控制

2.11.1 for 循环

常用的循环对象有:

- 容器数据类型如 list、tuple、dict、set、str 等
- range(),enumerate() 函数生成对象

```
names = ['Michael', 'Bob', 'Tracy']

for name in names:
  print(f'你好, {name}!')
```

```
你好, Michael!
你好, Bob!
你好, Tracy!
```

```
# 进行一定目的或逻辑的 for 循环
result = 0
for i in range(1, 101):
    result = result + i
result
```

```
5050
```

```
grades = {'Michael': 95,
          'Bob': 75,
          'Tracy': 85}

for name, grade in grades.items():
    print(f'{name}的分数是: {grade}')
```

```
Michael的分数是: 95
Bob的分数是: 75
Tracy的分数是: 85
```

2.11.2 布尔类型

```
print(True)
print(False)
# no error ? 没有引号
```

```
True
False
```

- 布尔类型只有两个值：True 和 False。

```
print(10 == 9)
```

```
False
```

```
print(10 < 9)
```

```
False
```

```
type(10 < 9)
```

```
bool
```

```
type(True)
```

```
bool
```

```
x = 10 == 9
```

```
print(x)
```

```
False
```

2.11.3 条件判断

- *Comparison Operators*: >, <, >=, <=, ==, !=
- *Membership Operators*: `in`, `not in`
- *Equality Operator*: `and`, `or`, `not`

```
"He" in "Hello"
```

```
True
```

```
1 in [1, 3, 3, 4]
```

```
True
```

```
3 >= 2 and 3 < 2
```

```
False
```

2.11.4 if 语句

```
age = 17
if age >= 18:
    print('your age is', age)
    print('adult')
```

```
age = 3
if age >= 18:
    print('your age is', age)
    print('adult')
else:
    print('your age is', age)
    print('teenager')
```

```
your age is 3
teenager
```

> 注意不要少写了冒号:。

```
age = 20

if 0<= age <= 3:
    print('your age is', age)
    print('baby')
elif 3< age < 18:
    print('your age is', age)
    print('teenager')
else:
    print('your age is', age)
    print('adult')
```

```
your age is 20
adult
```

注意： None、 []、 ''、 0 与 False 功效几乎完全等同！

如果 condition 为 None、 []、 ''、 0、 False，dosomething 是不执行的！

2.11.5 循环控制关键字

- break: 终止当前循环，且跳出整个循环
- continue: 终止当次循环，跳出该次循环，直接执行下一次循环
- pass 不执行任何操作，一般用于占位
- else: 当 for 循环正常执行完毕（即没有通过 break 语句提前结束）时，else 块会被执行

```
r_list = [-0.0137, 0.024, 0.0061, -0.0022,
  0.0101, -0.0087, 0.0196, 0.0002, -0.0055 ]

for i in r_list:
    if i > 0.02:
        break
    print(" 收益率数据: ",i)
```

```
收益率数据:  -0.0137
```

```
for i in r_list:
    if i < 0:
        continue
    print(" 非负收益率数据: ", i)
```

```
非负收益率数据:  0.024
非负收益率数据:  0.0061
非负收益率数据:  0.0101
非负收益率数据:  0.0196
非负收益率数据:  0.0002
```

```
for i in r_list:
    if i < 0:
        pass # 涨跌幅数据为负数时，不执行任何操作
    else:
        print(" 非负收益率数据: ",i)
```

```
非负收益率数据:  0.024
非负收益率数据:  0.0061
非负收益率数据:  0.0101
非负收益率数据:  0.0196
非负收益率数据:  0.0002
```

```
for i in range(5):
    if i == 6:
        print(f"Found {i}")
        break
else:
    print("Not found")
```

```
Not found
```

2.11.6 while 循环

```
n = 1
while n <= 5:
    print(n)
    n = n + 1
print('END')
```

```
1
2
3
```

```
4
5
END
```

```
n = 0
while True:
  n = n + 1
  print(n)
  if n >= 5:
     break
```

```
1
2
3
4
5
```

2.12 模块和包

2.12.1 包 Package

- Python 有一些自带的 packages，如 `os`, `pathlib`, `requests` 等
- 还可以通过 `pip` 安装更多的 packages
  - 在线安装: `pip install 库名`
  - 离线安装: 下载`.whl` 文件，然后 `pip install 库文件.whl`

```
%pip install --upgrade pip
%pip config set global.index-url
 ↪  https://mirrors.tuna.tsinghua.edu.cn/pypi/web/simple
```

```
%pip install dfply
```

```
import os
os.getcwd()
```

```
'/Users/xinlu/_project/AI4MPAcc-Course'
```

```
import pandas as pd
pd.DataFrame()
```

```
from os import getcwd
getcwd()
```

```
'/Users/xinlu/_project/AI4MPAcc-Course'
```

```
from os import getcwd as pwd
pwd()
```

```
'/Users/xinlu/_project/AI4MPAcc-Course'
```

2.12.2 自己的模块 Module

2.13 错误处理

- 如果代码中遇到 错误 **Error**， 那么程序会在 错误所在的代码行停止。而错误后面的代码将无法执行。
- try 和 except 语句

```
a = 10
nums = [3,4,5,2,0,1,7,8]
for b in nums:
```

```
    print(a, b, a/b)
print(" 继续")
```

```
a = 10
nums = [3,4,5,2,0,1,7,8]
for b in nums:
    try:
        print(a, b, a/b)
    except:
        pass
print(" 继续")
```

```
10 3 3.3333333333333335
10 4 2.5
10 5 2.0
10 2 5.0
10 1 10.0
10 7 1.4285714285714286
10 8 1.25
继续
```

2.14 MarkDown 基本语法

Markdown 是一种轻量级标记语言，用于格式化纯文本。

- Jupyter notebook Markdown 单元格
- 各种 Chat Bot 的交互界面
- 技术文档

2.14.1 标题

使用 # 表示标题，# 的数量表示标题的级别，最多支持六级标题。

```
# 一级标题
## 二级标题
### 三级标题
#### 四级标题
##### 五级标题
###### 六级标题
```

2.14.2 强调

使用 * 或 _ 包围文字表示斜体，使用 ** 或 __ 包围文字表示粗体，使用 ~~ 包围文字表示删除线。

```
* 斜体 * 或 _ 斜体 _

** 粗体 ** 或 __ 粗体 __

~~ 删除线 ~~
```

2.14.3 列表

- **无序列表** 使用 -、+ 或 * 表示。
- **有序列表** 使用数字加 . 表示。

```
- 项目 1
- 项目 2
  - 子项目 2.1
  - 子项目 2.2

1. 项目 1
2. 项目 2
   1. 子项目 2.1
   2. 子项目 2.2
```

2.14.4 链接

使用 [文本](链接) 创建链接，使用 [文本](链接 " 标题") 创建带有标题的链接。

```
[Google](https://www.google.com)

[Google](https://www.google.com " 访问 Google")
```

2.14.5 图片

使用 ![替代文字](图片链接) 插入图片。

```
![示例图片](https://example.com/image.jpg)
```

2.14.6 引用

使用 > 表示引用，可以嵌套使用。

```
> 这是一个引用
>> 这是一个嵌套引用
```

2.14.7 代码

- **行内代码** 使用反引号 ` 包围代码。
- **代码块** 使用三个反引号 ``` 或四个空格缩进代码。

````
`行内代码`

```python
def hello():
    print("Hello, World!")
```
````

2.14.8 分割线

使用三个或以上的 -、* 或 _ 表示分割线。

```
---

***
---
```

2.14.9 表格

使用 | 创建表格，使用 - 定义表头。

```
| 头 1 | 头 2 | 头 3 |
| --- | --- | --- |
| 内容 1 | 内容 2 | 内容 3 |
| 内容 A | 内容 B | 内容 C |
```

头 1	头 2	头 3
内容 1	内容 2	内容 3
内容 A	内容 B	内容 C

2.14.10 公式

使用 $ 包围公式，支持 LaTeX 语法。

inline formula `$ f\left(Y_{i t}\right)+R_{i t} g\left(Y_{i t}\right)+R_{i t} D_{i t} c\left(Y_{i t}\right)+D_{i t} d\left(Y_{i t}\right) + \epsilon_{i t} $`

inline formula $f\left(Y_{it}\right)+R_{it} g\left(Y_{it}\right)+R_{it} D_{it} c\left(Y_{it}\right)+D_{it} d\left(Y_{it}\right)+\epsilon_{it}$

block formula

```
$$
X_{i t}=\underbrace{f\left(Y_{i t}\right)+R_{i t} g\left(Y_{i
↪ t}\right)+R_{i t} D_{i t} c\left(Y_{i t}\right)+D_{i t} d\left(Y_{i
↪ t}\right)}_{=m\left(Y_{i t}, R_{i t}\right)}+\epsilon_{i t},
$$
```

$$X_{it}=\underbrace{f\left(Y_{it}\right)+R_{it} g\left(Y_{it}\right)+R_{it} D_{it} c\left(Y_{it}\right)+D_{it} d\left(Y_{it}\right)}_{=m\left(Y_{it}, R_{it}\right)}+\epsilon_{it},$$

2.14.11 Markdown Display

```
from IPython.display import display, Markdown
display(Markdown("### 测试三级标题"))
```

```
for x in range(5):
    # Print in Markdown format
    display(Markdown(f"${x}^2$ <red style='color:red'> = </red>
↪   **{x**2}**"))
```

2.14.12 Makrdown 编辑器

- VS Code
  - VS Code Markdown All in One 插件
  - VS Code Markdown Preview Enhanced 插件
- zettlr
- Obsidian

3 python 调用 LLM

点击下载本章节代码

3.1 配置环境变量 KEY

```
%env DASHSCOPE_API_KEY="sk-xxx"
```

3.2 不同模型调用

3.2.1 zhipu AI

```
import os
from openai import OpenAI
os.environ["ZAI_API_KEY"]
client = OpenAI(api_key=os.environ["ZAI_API_KEY"],
                    base_url="https://open.bigmodel.cn/api/paas/v4/")

completion = client.chat.completions.create(
    model="glm-4.5-flash",
    messages=[
    {"role": "user", "content": "Hello!"}
    ]
)

print(completion.choices[0].message.content)
```

```
Hello!  How can I help you today? Feel free to ask anything or start a conversation!
```

3.2.2 deepseek

```python
from openai import OpenAI
client = OpenAI(
    # 若没有配置环境变量，请用百炼 API Key 将下行替换为：api_key="sk-xxx",
    base_url="https://api.deepseek.com",
    api_key=os.environ["DEEPSEEK_API_KEY"],
)

completion = client.chat.completions.create(
    model="deepseek-chat",
    messages=[
    {"role": "user", "content": "Hello!"}
    ]
)

print(completion.choices[0].message.content)
```

```
Hello! How can I help you today?
```

3.2.3 阿里千问

```python
from openai import OpenAI
client = OpenAI(
    # 若没有配置环境变量，请用百炼 API Key 将下行替换为：api_key="sk-xxx",
    api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1",
)
```

```
completion = client.chat.completions.create(
    model="qwen-turbo",
    messages=[
    {"role": "user", "content": "Hello!"}
    ]
)

print(completion.choices[0].message.content)
```

```
Hello! How can I assist you today?
```

3.3 模型回复结构

```
from pprint import pprint
pprint(completion.to_dict())
```

```
{'choices': [{'finish_reason': 'stop',
              'index': 0,
              'logprobs': None,
              'message': {'content': 'Hello! How can I assist you today?  ',
                          'role': 'assistant'}}],
 'created': 1758093186,
 'id': 'chatcmpl-eecb7ef2-94c5-433a-8812-65f09cd86115',
 'model': 'qwen-turbo',
 'object': 'chat.completion',
 'system_fingerprint': None,
 'usage': {'completion_tokens': 11,
           'prompt_tokens': 14,
           'prompt_tokens_details': {'cached_tokens': 0},
           'total_tokens': 25}}
```

3.4 简单问答

```
client = OpenAI(api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1")

messages = [{"role": "user", "content": " 财务会计和管理会计的区别是什么?
↪  "}]

response = client.chat.completions.create(
    model="qwen-turbo",
    messages=messages
)
print(response.choices[0].message.content)
```

财务会计（Financial Accounting）和管理会计（Managerial Accounting）是会计学的两个重要

一、定义不同

项目	财务会计	管理会计
定义	为企业外部利益相关者提供财务信息的会计系统	为内部管理者提供决策支持信息的会

二、服务对象不同

项目	财务会计	管理会计
服务对象	外部：股东、债权人、政府、公众等	内部：企业管理人员、部门负责人、管理层
使用者	非会计人员（如投资者、监管机构）	会计人员和管理人员（专业性较强）

三、报告内容和时间不同

项目	财务会计	管理会计
报告内容	以历史数据为主，反映企业过去的经营成果和财务状况	包括历史数据、预测、预
报告时间	按固定周期（如月、季、年）编制报表（如资产负债表、利润表）	按需随时编制

四、信息特征不同

项目	财务会计	管理会计
信息性质	历史性、总结性、综合性	预测性、前瞻性、细节性
信息形式	标准化、统一化	非标准化、多样化
信息范围	全面、综合（整个企业）	针对性、局部性（某个部门、产品或项目）

五、依据和规范不同

项目	财务会计	管理会计
依据	会计准则（如中国《企业会计准则》、国际财务报告准则IFRS）	企业内部规定、管理
保密性	一般公开（如年报、财报）	通常不公开，属于企业内部机密

六、目的不同

项目	财务会计	管理会计
目的	向外部提供真实、公允的财务信息，供其进行投资、信贷等决策	为内部管理提供信息

七、举例说明

- **财务会计**:
  - 编制季度利润表
  - 提供年度审计报告
  - 发布年度财务报表给股东和监管部门

- **管理会计**:
  - 分析某产品的成本与利润
  - 制定部门预算
  - 进行绩效评估和成本控制

总结对比表:

对比项	财务会计	管理会计
服务对象	外部	内部
时间性	历史数据	预测与决策
内容	综合性、标准化	局部性、灵活性
规范	会计准则	企业内部制度
报告频率	定期	按需
保密性	公开	保密

如果你是学生或刚接触会计知识，理解这两者的区别有助于你更好地把握会计工作的不同应用场景

3.4.1 display response markdown

```
from IPython.display import display, Markdown
display(Markdown(response.choices[0].message.content))
```

3.4.2 模型列表

https://bailian.console.aliyun.com/?tab=model#/model-market

3.5 自定义函数

```
def ask_qwen(prompt, model="qwen-turbo"):

    client = OpenAI(api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1"
    )

    messages = [{"role": "user", "content": prompt}]

    response = client.chat.completions.create(
        model=model,
        messages=messages,
        extra_body={"enable_thinking": False,"enable_search": True,
                    "search_options": {"search_strategy": "pro"}}
    )
    return response.choices[0].message.content
```

```
ask_qwen(" 中国 2024 年一季度的 GDP 是多少，只回答阿拉伯数字，不要其他信
↪ 息，单位是亿元人民币")
```

```
'304761.8'
```

```
ask_qwen(" 中国 2023 年一季度的 GDP 是多少，只回答阿拉伯数字，不要其他信
↪ 息，单位是亿元人民币")
```

```
'11713'
```

> 思考
>
> 如何用 For Loop 获取过去 5 年的 GDP 数据?

3.6 JSON output 格式化

```
import json
def extract_json(text):
    """Extract JSON from text response"""
    try:
        # Look for JSON between backticks or just anywhere in the text
        import re
        json_match = re.search(r'```json\s*(.*?)\s*```', text, re.DOTALL)
        if json_match:
            json_str = json_match.group(1)
            return json.loads(json_str)
        else:
            # Try to find JSON directly
            return json.loads(text)
    except Exception as e:
        print(f"Error extracting JSON: {e}")
        print(f"Original text: {text}")
        return None
```

```
function_string = """
```json
{"title": "First Int", "type": "integer"}
```
"""

extract_json(function_string)
```

```
{'title': 'First Int', 'type': 'integer'}
```

```
prompt = """
请生成包括书名、作者和类别的三本虚构书籍清单，\
并以 JSON 格式提供，其中包含以下键:book_id、title、author、genre:

{"books": [ ... ] }
"""

response = ask_qwen(prompt, model="qwen-turbo")
print(response)
```

```
{
  "books": [
    {
      "book_id": 1,
      "title": "星尘之梦",
      "author": "艾琳·维斯特",
      "genre": "奇幻"
    },
    {
      "book_id": 2,
      "title": "寂静回声",
```

```
      "author": "马克斯 · 格雷",
      "genre": "悬疑"
    },
    {
      "book_id": 3,
      "title": "时间的碎片",
      "author": "莉娅 · 陈",
      "genre": "科幻"
    }
  ]
}
```

```
extract_json(response)
```

```
{'books': [{'book_id': 1, 'title': '星尘之梦', 'author': '艾琳 · 维斯特', 'genre': '奇
  {'book_id': 2, 'title': '寂静回声', 'author': '马克斯 · 格雷', 'genre': '悬疑'},
  {'book_id': 3, 'title': '时间的碎片', 'author': '莉娅 · 陈', 'genre': '科幻'}]}
```

```
import pandas as pd
df = pd.DataFrame(extract_json(response)["books"])
df
```

	book_id	title	author	genre
0	1	星尘之梦	艾琳 · 维斯特	奇幻
1	2	寂静回声	马克斯 · 格雷	悬疑
2	3	时间的碎片	莉娅 · 陈	科幻

```
df.to_csv("books.csv", index=False)
```

```
import os
os.remove("books.csv")
```

3.7 财务数据 sentiment analysis

```
prompt = """
请生成 10 个关于上市公司的财务数据的新闻标题，5 个正面，5 个负面 \
并以 JSON 格式提供，其中包含以下键:stock_code、title:

{"news": [ ... ] }
"""

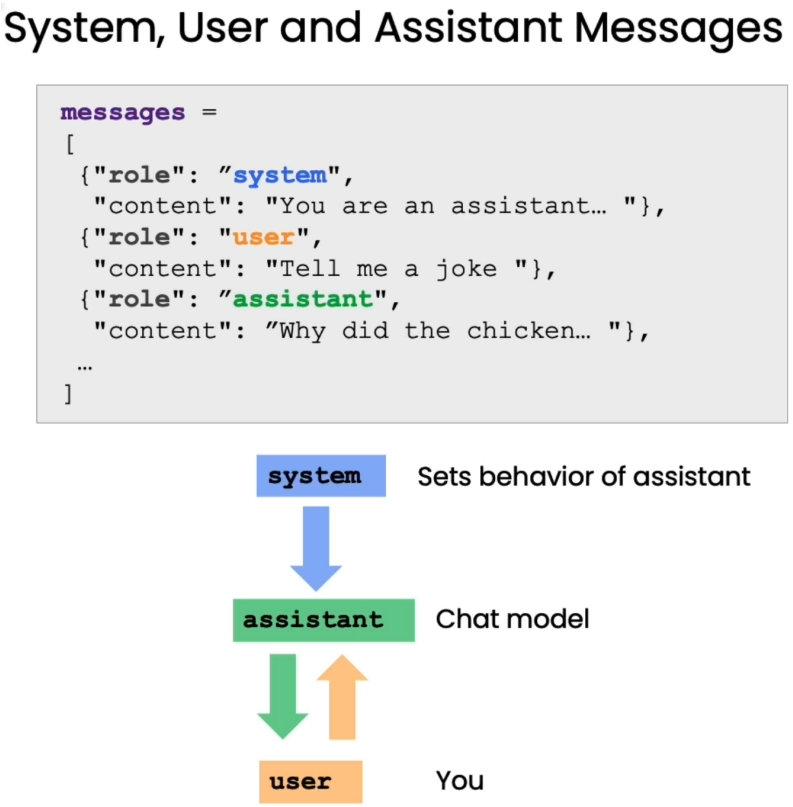
response = ask_qwen(prompt)
print(response)
```


```
{
  "news": [
    {
      "stock_code": "601995.SH",
      "title": "中金公司2025年中期业绩稳健增长，归母净利润同比增长94%"
    },
    {
      "stock_code": "002635.SZ",
      "title": "TCL科技扣非净利润大增179%，显示业务盈利能力显著提升"
    },
    {
      "stock_code": "002493.SZ",
      "title": "联发股份上半年经营性现金流净额创历史新高，盈利倍增"
    },
    {
      "stock_code": "300510.SZ",
      "title": "金冠股份上半年亏损扩大，但智能电网设备需求持续增长"
    },
    {
      "stock_code": "300510.SZ",
```

```
      "title": "金冠股份第二季度亏损扩大，但储能业务发展潜力巨大"
    },
    {
      "stock_code": "002493.SZ",
      "title": "联发股份毛利率提升，主业造血能力明显增强"
    },
    {
      "stock_code": "300510.SZ",
      "title": "金冠股份上半年亏损扩大，资产规模有所下降"
    },
    {
      "stock_code": "002635.SZ",
      "title": "TCL科技上半年营收稳健增长，净利润显著提升"
    },
    {
      "stock_code": "002493.SZ",
      "title": "联发股份绿色转型加速，入选江苏省‘绿色工厂’示范名单"
    },
    {
      "stock_code": "300510.SZ",
      "title": "金冠股份上半年亏损扩大，现金流有所改善"
    }
  ]
}
```


```
df = pd.DataFrame(extract_json(response)["news"])
df
```

	stock_code	title
0	601995.SH	中金公司 2025 年中期业绩稳健增长，归母净利润同比增长 94%
1	002635.SZ	TCL 科技扣非净利润大增 179%，显示业务盈利能力显著提升
2	002493.SZ	联发股份上半年经营性现金流净额创历史新高，盈利倍增

	stock_code	title
3	300510.SZ	金冠股份上半年亏损扩大，但智能电网设备需求持续增长
4	300510.SZ	金冠股份第二季度亏损扩大，但储能业务发展潜力巨大
5	002493.SZ	联发股份毛利率提升，主业造血能力明显增强
6	300510.SZ	金冠股份上半年亏损扩大，资产规模有所下降
7	002635.SZ	TCL 科技上半年营收稳健增长，净利润显著提升
8	002493.SZ	联发股份绿色转型加速，入选江苏省‘绿色工厂’示范名单
9	300510.SZ	金冠股份上半年亏损扩大，现金流有所改善

```
def sentiment_analysis(text):
    prompt = f"""
    请判断下面的新闻标题为 正面还是负面:
    ```{text}```
    仅回答“正面”或“负面”即可。
    """
    response = ask_qwen(prompt)
    return response
```

```
from tqdm.notebook import tqdm
tqdm.pandas()

df["sentiment"] = df["title"].progress_apply(sentiment_analysis)
df
```

```
  0%|          | 0/10 [00:00<?, ?it/s]
```

	stock_code	title	sentiment
0	601995.SH	中金公司 2025 年中期业绩稳健增长，归母净利润同比增长 94%	正面
1	002635.SZ	TCL 科技扣非净利润大增 179%，显示业务盈利能力显著提升	正面
2	002493.SZ	联发股份上半年经营性现金流净额创历史新高，盈利倍增	正面
3	300510.SZ	金冠股份上半年亏损扩大，但智能电网设备需求持续增长	负面
```

	stock_code	title	sentiment
4	300510.SZ	金冠股份第二季度亏损扩大，但储能业务发展潜力巨大	负面
5	002493.SZ	联发股份毛利率提升，主业造血能力明显增强	正面
6	300510.SZ	金冠股份上半年亏损扩大，资产规模有所下降	负面
7	002635.SZ	TCL 科技上半年营收稳健增长，净利润显著提升	正面
8	002493.SZ	联发股份绿色转型加速，入选江苏省‘绿色工厂’示范名单	正面
9	300510.SZ	金冠股份上半年亏损扩大，现金流有所改善	负面

3.8 订餐系统

3.8.1 messages 结构




```
qa_messages = [{'role':'system', 'content':"""
你是订餐机器人，为披萨餐厅自动收集订单信息。
你要首先问候顾客。然后等待用户回复收集订单信息。收集完信息需确认顾客是否还
↪ 需要添加其他内容。
最后需要询问是否自取或外送，如果是外送，你要询问地址。
```

```
最后告诉顾客订单总金额，并送上祝福。

请确保明确所有选项、附加项和尺寸，以便从菜单中识别出该项唯一的内容。
你的回应应该以简短、非常随意和友好的风格呈现。

菜单包括:

菜品:
意式辣香肠披萨（大、中、小） 12.95、10.00、7.00
芝士披萨（大、中、小） 10.95、9.25、6.50
茄子披萨（大、中、小） 11.95、9.75、6.75
薯条（大、小） 4.50、3.50
希腊沙拉 7.25

配料:
奶酪 2.00
蘑菇 1.50
香肠 3.00
加拿大熏肉 3.50
AI 酱 1.50
辣椒 1.00

饮料:
可乐（大、中、小） 3.00、2.00、1.00
雪碧（大、中、小） 3.00、2.00、1.00
瓶装水 5.00
"""},
{"role": "user",
"content": " 我要一个大可乐，一个大的意大利辣香肠披萨，加半根香肠，再要一个
↪  扬州炒饭，直接回答总价"
}
]
```

```python
def ask_qwen(messages, model="qwen-turbo"):

    client = OpenAI(api_key=os.getenv("DASHSCOPE_API_KEY"),
    base_url="https://dashscope.aliyuncs.com/compatible-mode/v1"
    )

    response = client.chat.completions.create(
        model=model,
        messages=messages
    )
    return response.choices[0].message.content
```

```python
res = ask_qwen(qa_messages)
print(res)
```

你好呀！让我来帮你算一下总价：

- 大可乐：3.00
- 意大利辣香肠披萨（大）：12.95
- 半根香肠：1.50
- 扬州炒饭：这个我们没有在菜单上，是不是要点别的？比如薯条或者沙拉？

如果你确定要扬州炒饭，那我先按这个计算，总价是 **17.45**。你确认一下吗？

> 思考
>
> 我们是否可以自定义一个更好的 print 函数，利用 markdown 语法来打印结果？

4 pandas 数据框

点击下载本章节代码

4.1 Pandas 处理表格数据

参考: https://realpython.com/pandas-dataframe/

pandas DataFrame 是一种包含**二维数据**及其对应**标签**的**数据结构**。

DataFrame 类似于 Excel 或 SQL 中使用的电子表格。在很多情况下，**DataFrame** 比表格或电子表格更快、更易于使用且功能更强大。

4.1.1 安装所需包

```
%pip install pandas openpyxl pyarrow dfply matplotlib seaborn --upgrade
```

4.1.2 导入 pandas

```
import pandas as pd
```

4.1.3 创建示例数据

让我们创建一个会计科目表作为示例:

```
# 创建会计科目数据
data = {
    'account_name': ['现金', '银行存款', '应收账款', '存货', '固定资产',
    ↪  '应付账款', '实收资本'],
```

```
    'account_type': ['资产', '资产', '资产', '资产', '资产', '负债', '所
    ↪ 有者权益'],
    'balance': [50000.0, 200000.0, 150000.0, 180000.0, 300000.0,
    ↪ 100000.0, 680000.0],
    'balance_direction': ['借', '借', '借', '借', '借', '贷', '贷']
}

# 使用会计科目编号作为索引
account_codes = [1001, 1002, 1122, 1123, 1601, 2202, 4001]

df = pd.DataFrame(data, index = account_codes)
```

> **Tip**
>
> **会计背景说明:** - **账户编号:** 通常使用数字编码, 如 1 开头表示资产,2 开头表示负债等 - **借贷方向:** 资产类账户余额在借方, 负债和所有者权益账户余额在贷方 - **余额:** 表示该科目的期末余额

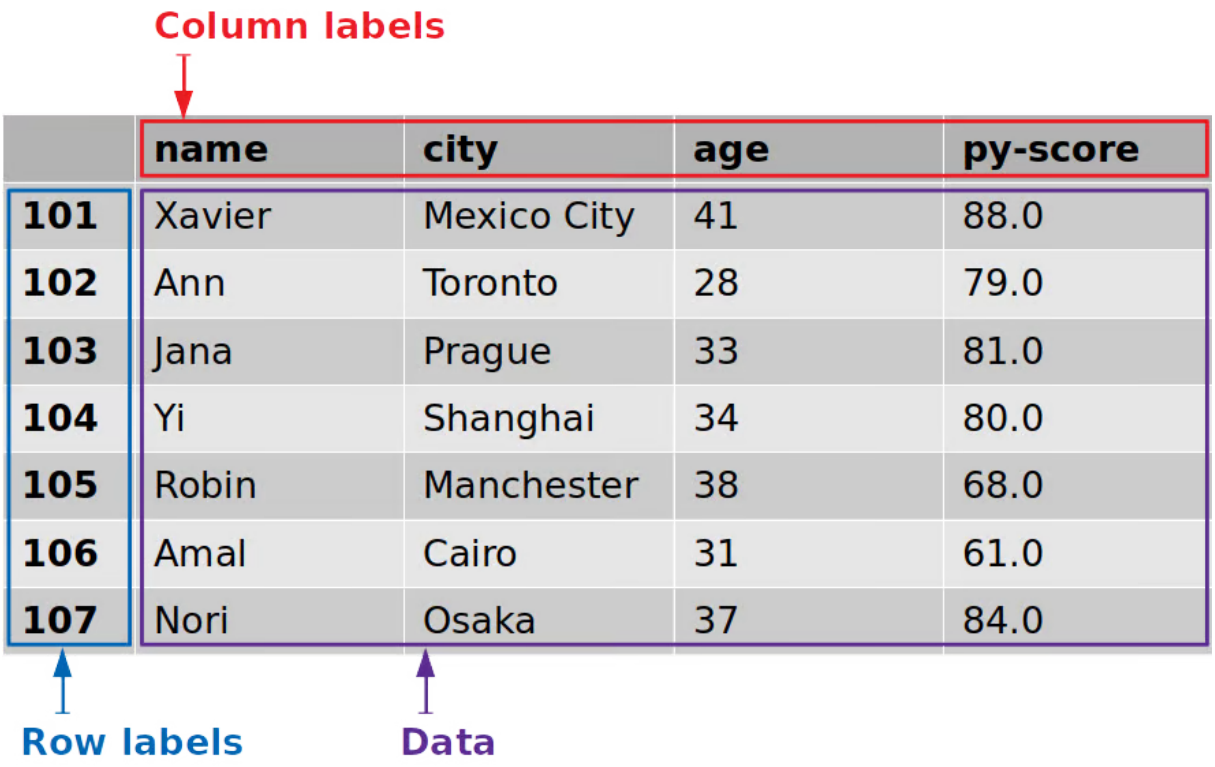
4.2 DataFrame 结构与属性

查看 DataFrame 类型和结构:

```
type(df)
```

```
pandas.core.frame.DataFrame
```

```
df
```



> **Note**
>
> 可以删掉 labels 的信息，系统会自己给一个编号。

4.2.1 索引 (index)

```
df.index
```

```
Index([1001, 1002, 1122, 1123, 1601, 2202, 4001], dtype='int64')
```

4.2.2 列名 (columns)

```
df.columns
```

```
Index(['account_name', 'account_type', 'balance', 'balance_direction'], dtype='object
```

4.2.3 值 (values)

```
df.values
```

```
array([['现金', '资产', 50000.0, '借'],
       ['银行存款', '资产', 200000.0, '借'],
       ['应收账款', '资产', 150000.0, '借'],
       ['存货', '资产', 180000.0, '借'],
       ['固定资产', '资产', 300000.0, '借'],
       ['应付账款', '负债', 100000.0, '贷'],
       ['实收资本', '所有者权益', 680000.0, '贷']], dtype=object)
```

4.2.4 形状 (shape)

```
df.shape
```

```
(7, 4)
```

4.2.5 数据类型 (dtypes)

```
df.dtypes
```

```
account_name          object
account_type          object
balance              float64
balance_direction     object
dtype: object
```

4.2.6 基本信息 (info)

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7 entries, 1001 to 4001
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   account_name       7 non-null      object 
 1   account_type       7 non-null      object 
 2   balance            7 non-null      float64
 3   balance_direction  7 non-null      object 
dtypes: float64(1), object(3)
memory usage: 280.0+ bytes
```

4.3 DataFrame 行列操作

4.3.1 访问列

通过列名访问列数据:

```
# 显示账户名称列
df.account_name
```

```
# 将账户名称列赋值给变量
accounts = df['account_name']
accounts
```

```
1001      现金
1002    银行存款
1122    应收账款
1123      存货
1601    固定资产
```

```
2202    应付账款
4001    实收资本
Name: account_name, dtype: object
```

查看列的类型:

```
type(df.account_name)
```

```
pandas.core.series.Series
```

访问特定位置的值:

```
# 访问账户编号为 1002 的账户名称
accounts[1002]
```

```
'银行存款'
```

> **Note**
>
> **说明:** DataFrame 的每一列都是一个 Series 对象。Series 是 pandas 的一维数组, 类似于 Excel 中的单独一列。

4.3.2 访问行

使用.loc 访问行:

```
# 访问账户编号为 1122 的科目信息
account_row = df.loc[1122]
account_row
```

```
account_name              应收账款
account_type                资产
balance                150000.0
balance_direction             借
Name: 1122, dtype: object
```

访问行中的特定列:

```
# 获取该科目的账户类型
account_row['account_type']
```

```
'资产'
```

查看行的类型:

```
type(account_row)
```

```
pandas.core.series.Series
```

> **Note**
>
> **说明:** 访问 DataFrame 的一行会返回一个 Series 对象, 其索引是原 DataFrame 的列名。

4.3.3 转置

原始数据:

```
df
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

转置后:

```
df.T
```

	1001	1002	1122	1123	1601	2202	4001
account_name	现金	银行存款	应收账款	存货	固定资产	应付账款	实收资本
account_type	资产	资产	资产	资产	资产	负债	所有者权益
balance	50000.0	200000.0	150000.0	180000.0	300000.0	100000.0	680000.0
balance_direction	借	借	借	借	借	贷	贷

4.3.4 查看头尾数据

查看前 n 行:

```
df.head(n=3)
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借

查看后 n 行 (默认 5 行):

```
df.tail()
```

	account_name	account_type	balance	balance_direction
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

4.3.5 创建 DataFrame

从字典列表创建:

```
l = [{'x': 1, 'y': 2, 'z': 100},
     {'x': 2, 'y': 4, 'z': 100},
     {'x': 3, 'y': 8, 'z': 100}]

pd.DataFrame(l)
```

	x	y	z
0	1	2	100
1	2	4	100
2	3	8	100

从二维数组创建 - 指定列名和索引:

```
# 从数组构建 DataFrame
data = [['Alice', 25], ['Bob', 30], ['Charlie', 35]]
pd.DataFrame(data, columns=['Name', 'Age'], index = [1,2,9])
```

	Name	Age
1	Alice	25
2	Bob	30
9	Charlie	35

使用默认列名和索引:

```
pd.DataFrame(data)
```

	0	1
0	Alice	25
1	Bob	30
2	Charlie	35

4.4 DataFrame 导入与导出

学习目标: - 掌握从不同文件格式读取数据 - 了解常见数据导入问题的处理方法 - 学会将处理后的数据导出为专业格式

pandas 可以将各种数据格式导入为 DataFrame:

- 支持的格式: `CSV`, `XLSX`, `SQL`, `JSON`, `HTML`, `HDF5`, `SAS`, `STATA`, `Parquet`, `SQL` 等
- 参考文档: https://pandas.pydata.org/pandas-docs/stable/reference/io.html

4.4.1 读取 CSV 文件

CSV(逗号分隔值) 是会计数据最常见的格式之一:

```
# 基本读取
df = pd.read_csv('financial_data.csv')

# 指定编码 (处理中文)
df = pd.read_csv('financial_data.csv', encoding='utf-8')

# 指定数据类型 (提高性能)
df = pd.read_csv('financial_data.csv',
                 dtype={'account_number': str, 'amount': float})

# 处理日期列
df = pd.read_csv('financial_data.csv',
                 parse_dates=['transaction_date'])
```

> **Tip**
>
> **会计应用:** 从 ERP 系统或财务软件导出的数据通常是 CSV 格式。正确设置编码和数据类型可以避免很多后续问题。

4.4.2 读取 Excel 文件

```
# 读取第一个工作表
df = pd.read_excel("example.xlsx", sheet_name=0)
```

```
# 读取特定工作表
df = pd.read_excel('financial_statements.xlsx', sheet_name='Balance
↪ Sheet')

# 跳过某些行 (如标题行)
df = pd.read_excel('financial_statements.xlsx', skiprows=[0, 1])

# 指定索引列
df = pd.read_excel('financial_statements.xlsx', index_col=0)
```

4.4.3 处理大文件

对于大型财务数据文件, 可以分块读取:

```
# 分块读取 CSV
chunk_size = 10000
chunks = []
for chunk in pd.read_csv('large_transaction_file.csv',
↪ chunksize=chunk_size):
    # 对每个块进行处理
    chunks.append(chunk)
df = pd.concat(chunks, ignore_index=True)
```

4.4.4 导出数据

导出到 Excel:

```
df.to_excel("example.xlsx", index=False)
```

```
# 导出到 CSV(不包含索引)
df.to_csv('output.csv', index=False, encoding='utf-8-sig')

# 导出到 Excel with 格式化
with pd.ExcelWriter('financial_report.xlsx', engine='openpyxl') as
 ↪  writer:
    df.to_excel(writer, sheet_name='Summary', index=False)
```

> **Warning**
>
> **常见错误:** 忘记设置 `index=False` 会导致导出时包含行号列, 这在财务报表中通常是不需要的。

4.4.5 错误处理

```
# 安全地读取文件
try:
    df = pd.read_excel('financial_data.xlsx')
    print(f" 成功读取 {len(df)} 行数据")
except FileNotFoundError:
    print(" 文件未找到, 请检查文件路径")
except Exception as e:
    print(f" 读取文件时出错: {e}")
```

4.5 数据清洗 (Data Cleaning)

学习目标: - 识别和处理缺失值 - 检测和删除重复数据 - 转换数据类型 - 处理会计数据中的特殊格式

> **Important**
>
> **为什么数据清洗很重要?** 会计数据常常来自不同系统, 可能包含缺失值、重复记录、格式不统一等问题。数据清洗是数据分析的第一步, 决定了分析结果的准确性。

4.5.1 检查缺失值

创建示例财务数据 (包含缺失值):

```
import numpy as np

# 创建包含缺失值的账目数据
accounts_data = {
    'account_number': ['1001', '1002', '1003', '1004', '1005'],
    'account_name': ['现金', '应收账款', '存货', None, '固定资产'],
    'amount': [50000, 120000, np.nan, 80000, 200000],
    'category': ['资产', '资产', '资产', '资产', None]
}
accounts_df = pd.DataFrame(accounts_data)
accounts_df
```

	account_number	account_name	amount	category
0	1001	现金	50000.0	资产
1	1002	应收账款	120000.0	资产
2	1003	存货	NaN	资产
3	1004	None	80000.0	资产
4	1005	固定资产	200000.0	None

检查缺失值:

```
# 检查每列的缺失值数量
accounts_df.isnull().sum()
```

```
account_number    0
account_name      1
amount            1
category          1
dtype: int64
```

```
# 检查是否有任何缺失值
accounts_df.isnull().any()
```

```
account_number    False
account_name       True
amount             True
category           True
dtype: bool
```

```
# 查看包含缺失值的行
accounts_df[accounts_df.isnull().any(axis=1)]
```

	account_number	account_name	amount	category
2	1003	存货	NaN	资产
3	1004	None	80000.0	资产
4	1005	固定资产	200000.0	None

4.5.2 处理缺失值

不同的处理方法:

```
# 方法 1: 删除包含缺失值的行
df_dropped = accounts_df.dropna()
print(f" 删除后剩余 {len(df_dropped)} 行")
df_dropped
```

删除后剩余 2 行

	account_number	account_name	amount	category
0	1001	现金	50000.0	资产
1	1002	应收账款	120000.0	资产

```
# 方法 2: 用特定值填充 (金额用 0 填充)
df_filled = accounts_df.copy()
df_filled['amount'] = df_filled['amount'].fillna(0)
df_filled
```

	account_number	account_name	amount	category
0	1001	现金	50000.0	资产
1	1002	应收账款	120000.0	资产
2	1003	存货	0.0	资产
3	1004	None	80000.0	资产
4	1005	固定资产	200000.0	None

```
# 方法 3: 用平均值填充 (适用于数值列)
df_mean = accounts_df.copy()
df_mean['amount'] =
 ↪  df_mean['amount'].fillna(accounts_df['amount'].mean())
df_mean
```

	account_number	account_name	amount	category
0	1001	现金	50000.0	资产
1	1002	应收账款	120000.0	资产
2	1003	存货	112500.0	资产
3	1004	None	80000.0	资产
4	1005	固定资产	200000.0	None

```
# 方法 4: 用前一个值填充 (forward fill)
df_ffill = accounts_df.copy()
df_ffill = df_ffill.fillna(method='ffill')
df_ffill
```

	account_number	account_name	amount	category
0	1001	现金	50000.0	资产
1	1002	应收账款	120000.0	资产
2	1003	存货	120000.0	资产
3	1004	存货	80000.0	资产
4	1005	固定资产	200000.0	资产

Tip

会计应用: 对于财务金额, 通常用 0 填充缺失值。对于分类数据 (如账户类别), 可能需要用”未分类”或最频繁的类别填充。

4.5.3 检测重复数据

创建包含重复记录的交易数据:

```
# 创建包含重复的交易数据
transactions = pd.DataFrame({
    'transaction_id': ['T001', 'T002', 'T003', 'T002', 'T004'],
    'date': ['2024-01-01', '2024-01-02', '2024-01-03', '2024-01-02',
    ↪  '2024-01-04'],
```

```
    'amount': [1000, 2000, 1500, 2000, 3000],
    'description': ['销售收入', '采购支出', '销售收入', '采购支出', '销售
    ↪  收入']
})
transactions
```

	transaction_id	date	amount	description
0	T001	2024-01-01	1000	销售收入
1	T002	2024-01-02	2000	采购支出
2	T003	2024-01-03	1500	销售收入
3	T002	2024-01-02	2000	采购支出
4	T004	2024-01-04	3000	销售收入

检测重复:

```
# 检查完全重复的行
transactions.duplicated()
```

```
0    False
1    False
2    False
3     True
4    False
dtype: bool
```

```
# 查看重复的行
transactions[transactions.duplicated(keep=False)]
```

	transaction_id	date	amount	description
1	T002	2024-01-02	2000	采购支出
3	T002	2024-01-02	2000	采购支出

```
# 基于特定列检查重复（交易 ID 应该是唯一的）
transactions.duplicated(subset=['transaction_id'])
```

```
0    False
1    False
2    False
3     True
4    False
dtype: bool
```

删除重复:

```
# 删除重复行，保留第一次出现的
transactions_clean =
↪  transactions.drop_duplicates(subset=['transaction_id'], keep='first')
transactions_clean
```

	transaction_id	date	amount	description
0	T001	2024-01-01	1000	销售收入
1	T002	2024-01-02	2000	采购支出
2	T003	2024-01-03	1500	销售收入
4	T004	2024-01-04	3000	销售收入

> **Warning**
>
> **常见错误:** 删除重复数据前，要确认哪些字段组合应该是唯一的。有时看似重复的记录可能是合法的 (如相同客户的多次交易)。

4.5.4 数据类型转换

创建需要类型转换的数据:

```
# 创建混合类型的财务数据
messy_data = pd.DataFrame({
    'account': ['1001', '1002', '1003', '1004'],
    'revenue': ['$1,200', '$2,500', '$1,800', '$3,200'],
    'cost': ['(500)', '1200', '(800)', '1500'],
    'quantity': ['100', '200', '150', '250']
})
messy_data
```

	account	revenue	cost	quantity
0	1001	$1,200	(500)	100
1	1002	$2,500	1200	200
2	1003	$1,800	(800)	150
3	1004	$3,200	1500	250

```
# 查看当前数据类型
messy_data.dtypes
```

```
account     object
revenue     object
cost        object
quantity    object
dtype: object
```

处理货币符号和逗号:

```
# 清理 revenue 列（移除 $ 和逗号）
messy_data['revenue_clean'] = messy_data['revenue'].str.replace('$',
 ↪  '').str.replace(',', '').astype(float)
messy_data
```

	account	revenue	cost	quantity	revenue_clean
0	1001	$1,200	(500)	100	1200.0
1	1002	$2,500	1200	200	2500.0
2	1003	$1,800	(800)	150	1800.0
3	1004	$3,200	1500	250	3200.0

处理括号表示的负数 (会计中常用):

```
# 处理括号表示的负数
def parse_accounting_number(value):
    value = str(value).strip()
    if '(' in value:
        # 移除括号并转为负数
        return -float(value.replace('(', '').replace(')', ''))
    else:
        return float(value)

messy_data['cost_clean'] =
 ↪  messy_data['cost'].apply(parse_accounting_number)
messy_data[['cost', 'cost_clean']]
```

	cost	cost_clean
0	(500)	-500.0
1	1200	1200.0
2	(800)	-800.0
3	1500	1500.0

转换为数值类型:

```
# 转换 quantity 为整数
messy_data['quantity'] = pd.to_numeric(messy_data['quantity'],
 ↪  errors='coerce')
```

```
messy_data.dtypes
```

```
account           object
revenue           object
cost              object
quantity           int64
revenue_clean    float64
cost_clean       float64
dtype: object
```

> **Tip**
>
> **会计应用:** - 货币金额常包含货币符号 ($, ¥) 和千位分隔符 - 负数可能用括号表示, 如 (1,000) 表示 -1000 - 账户编号应保持为字符串类型, 避免前导零丢失

4.5.5 数据验证

创建需要验证的账目数据:

```
# 创建试算平衡表数据
trial_balance = pd.DataFrame({
    'account': ['现金', '应收账款', '应付账款', '资本'],
    'debit': [10000, 5000, 0, 0],
    'credit': [0, 0, 3000, 12000]
})
trial_balance
```

	account	debit	credit
0	现金	10000	0
1	应收账款	5000	0
2	应付账款	0	3000
3	资本	0	12000

验证借贷平衡:

```
# 计算借贷总额
total_debit = trial_balance['debit'].sum()
total_credit = trial_balance['credit'].sum()

print(f" 借方总额: ¥{total_debit:,.2f}")
print(f" 贷方总额: ¥{total_credit:,.2f}")

# 验证是否平衡
if total_debit == total_credit:
    print(" 借贷平衡")
else:
    print(f" 不平衡, 差额: ¥{abs(total_debit - total_credit):,.2f}")
```

```
借方总额: ¥15,000.00
贷方总额: ¥15,000.00
 借贷平衡
```

检查异常值:

```
# 创建包含异常值的销售数据
sales_data = pd.DataFrame({
    'product': ['A', 'B', 'C', 'D', 'E'],
    'units_sold': [100, 150, 200, 1000000, 180],  # D 产品的数量明显异常
    'unit_price': [10, 15, 12, 11, 13]
})

# 使用 describe() 查看统计信息
sales_data['units_sold'].describe()
```

```
count          5.000000
mean      200126.000000
std       447143.160945
min          100.000000
```

```
25%           150.000000
50%           180.000000
75%           200.000000
max       1000000.000000
Name: units_sold, dtype: float64
```

```python
# 识别异常值（使用 IQR 方法）
Q1 = sales_data['units_sold'].quantile(0.25)
Q3 = sales_data['units_sold'].quantile(0.75)
IQR = Q3 - Q1

# 定义异常值范围
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 找出异常值
outliers = sales_data[(sales_data['units_sold'] < lower_bound) |
                      (sales_data['units_sold'] > upper_bound)]
outliers
```

	product	units_sold	unit_price
3	D	1000000	11

> **Note**
>
> 练习: 1. 创建一个包含缺失值的应收账款数据集 2. 检查并处理缺失的金额和客户名称 3. 验证没有重复的发票号 4. 确保所有金额都是正数

4.6 DataFrame 筛选与条件过滤

学习目标: - 掌握多种数据筛选方法 - 学会条件过滤 - 应用于财务数据查询

4.6.1 切片操作

像 list 和 string 一样切片行数据:

```
# 前 3 行数据
df[:3]
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借

通过 Label(索引) 切片:

```
# 筛选账户编号 1001 到 1123 的科目
df.loc[1001:1123]
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借

同时筛选行和列:

```
# 只查看特定科目的名称和余额
df.loc[1001:1123, ['account_name','balance']]
```

	account_name	balance
1001	现金	50000.0
1002	银行存款	200000.0
1122	应收账款	150000.0

	account_name	balance
1123	存货	180000.0

通过位置 (position) 筛选:

```
# 前 3 行, 选择第 1,3,2,4 列 (按位置)
df.iloc[:3, [0,2,1,3]]
```

	account_name	balance	account_type	balance_direction
1001	现金	50000.0	资产	借
1002	银行存款	200000.0	资产	借
1122	应收账款	150000.0	资产	借

访问单个元素:

```
# 访问第一行第二列的值
df.iat[0, 1]
```

```
'资产'
```

4.6.2 条件筛选

筛选余额大于 150000 的科目:

```
# 筛选高额余额账户
high_balance_accounts = df[df['balance'] > 150000]
high_balance_accounts
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1123	存货	资产	180000.0	借

	account_name	account_type	balance	balance_direction
1601	固定资产	资产	300000.0	借
4001	实收资本	所有者权益	680000.0	贷

筛选资产类科目:

```
# 筛选所有资产类账户
asset_accounts = df[df['account_type'] == '资产']
asset_accounts
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借

多条件筛选 (使用 & 和 |):

```
# 筛选余额大于 100000 且为资产类的科目
df[(df['balance'] > 100000) & (df['account_type'] == '资产')]
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借

```
# 筛选负债类或所有者权益类科目
df[(df['account_type'] == '负债') | (df['account_type'] == '所有者权益')]
```

	account_name	account_type	balance	balance_direction
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

> **Warning**
>
> 注意: - 使用 & 表示" 且"(AND) - 使用 | 表示" 或"(OR) - 每个条件必须用括号括起来 - 不能使用 Python 的 and 和 or 关键字

4.6.3 使用 isin 筛选

筛选特定账户:

```
# 筛选现金和银行存款账户
df[df['account_name'].isin(['现金', '银行存款'])]
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借

排除特定账户 (使用 ~ 取反):

```
# 筛选除现金外的所有账户
df[~df['account_name'].isin(['现金'])]
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

4.6.4 query() 方法

更简洁的筛选方式:

```
# 使用 query 方法筛选
df.query('balance > 150000')
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
4001	实收资本	所有者权益	680000.0	贷

```
# 复杂条件查询
df.query('account_type == " 资产" and balance > 100000')
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借

> **Tip**
>
> **会计应用**: 条件筛选常用于: - 筛选特定类别的科目 (资产、负债等) - 查找大额交易或余额 - 识别异常账户 - 生成特定范围的财务报表

4.7 数据操作

4.7.1 赋值

首先创建副本:

```
df2 = df.copy()
df2
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

修改单个值和批量赋值:

```
# 修改单个值
df2.iat[0, 0] = " 库存现金"

# 批量修改特定行的某列值
df2.loc[1001:1123, 'balance'] = [55000, 210000, 160000, 190000]

# 创建新列（余额的万元单位）
df2['balance_ 万元'] = df2['balance'] / 10000
df2
```

	account_name	account_type	balance	balance_direction	balance_ 万元
1001	库存现金	资产	55000.0	借	5.5
1002	银行存款	资产	210000.0	借	21.0
1122	应收账款	资产	160000.0	借	16.0
1123	存货	资产	190000.0	借	19.0
1601	固定资产	资产	300000.0	借	30.0
2202	应付账款	负债	100000.0	贷	10.0

	account_name	account_type	balance	balance_direction	balance_ 万元
4001	实收资本	所有者权益	680000.0	贷	68.0

4.7.2 排序

按索引排序:

```
# 按账户编号（索引）排序
df.sort_index()
```

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1123	存货	资产	180000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

4.7.3 按值排序

```
# 按余额排序
df.sort_values('balance', ascending=False)
```

	account_name	account_type	balance	balance_direction
4001	实收资本	所有者权益	680000.0	贷
1601	固定资产	资产	300000.0	借
1002	银行存款	资产	200000.0	借
1123	存货	资产	180000.0	借
1122	应收账款	资产	150000.0	借

	account_name	account_type	balance	balance_direction
2202	应付账款	负债	100000.0	贷
1001	现金	资产	50000.0	借

4.7.4 重命名

批量重命名列名:

使用列表推导式:

```
df2 = df.copy()

df2.columns = [x.upper() for x in df.columns]
df2.head(2)
```

	ACCOUNT_NAME	ACCOUNT_TYPE	BALANCE	BALANCE_DIRECTION
1001	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借

4.7.5 选择性重命名

使用 rename 方法:

```
df2 = df.copy()
df2 = df2.rename(columns={"account_name": " 科目名称", "balance": " 余
↪  额"},
                 index={1001: " 现金科目"})
df2.head(2)
```

	科目名称	account_type	余额	balance_direction
现金科目	现金	资产	50000.0	借
1002	银行存款	资产	200000.0	借

4.7.6 删除行/列

删除行:

```
# 删除特定账户编号的行
df2 = df.drop(index=[1001, 1123])
df2
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1122	应收账款	资产	150000.0	借
1601	固定资产	资产	300000.0	借
2202	应付账款	负债	100000.0	贷
4001	实收资本	所有者权益	680000.0	贷

4.7.7 删除列

```
# 删除特定列
df2 = df.drop(columns=['balance_direction', 'account_type'])
df2
```

	account_name	balance
1001	现金	50000.0
1002	银行存款	200000.0
1122	应收账款	150000.0
1123	存货	180000.0
1601	固定资产	300000.0
2202	应付账款	100000.0
4001	实收资本	680000.0

4.8 统计计算与自定义函数

4.8.1 统计量

均值:

```
# 计算余额的平均值
df['balance'].mean()
```

```
np.float64(237142.85714285713)
```

4.8.2 标准差

```
# 计算余额的标准差
df['balance'].std()
```

```
210611.3550052391
```

4.8.3 描述性统计

```
# 余额的描述性统计
df['balance'].describe()
```

```
count         7.000000
mean     237142.857143
std      210611.355005
min       50000.000000
25%      125000.000000
50%      180000.000000
75%      250000.000000
max      680000.000000
Name: balance, dtype: float64
```

4.8.4 财务计算

学习目标: - 掌握累计计算 (running totals) - 学会计算百分比变化 - 了解移动平均和滚动计算 - 应用财务比率分析

累计求和 (Running Totals)

创建月度收入数据:

```
# 创建月度收入数据
monthly_revenue = pd.DataFrame({
    'month': ['2024-01', '2024-02', '2024-03', '2024-04', '2024-05',
    ↪  '2024-06'],
    'revenue': [100000, 120000, 115000, 135000, 140000, 150000],
    'cost': [60000, 70000, 68000, 80000, 82000, 88000]
})
monthly_revenue
```

	month	revenue	cost
0	2024-01	100000	60000
1	2024-02	120000	70000
2	2024-03	115000	68000
3	2024-04	135000	80000
4	2024-05	140000	82000
5	2024-06	150000	88000

计算累计收入 (年初至今 YTD):

```
# 计算累计收入
monthly_revenue['累计收入'] = monthly_revenue['revenue'].cumsum()
monthly_revenue
```

	month	revenue	cost	累计收入
0	2024-01	100000	60000	100000
1	2024-02	120000	70000	220000
2	2024-03	115000	68000	335000
3	2024-04	135000	80000	470000
4	2024-05	140000	82000	610000
5	2024-06	150000	88000	760000

计算累计利润:

```
# 先计算每月利润
monthly_revenue['profit'] = monthly_revenue['revenue'] -
 ↪  monthly_revenue['cost']
# 再计算累计利润
monthly_revenue['累计利润'] = monthly_revenue['profit'].cumsum()
monthly_revenue[['month', 'profit', '累计利润']]
```

	month	profit	累计利润
0	2024-01	40000	40000
1	2024-02	50000	90000
2	2024-03	47000	137000
3	2024-04	55000	192000
4	2024-05	58000	250000
5	2024-06	62000	312000

> **Tip**
>
> **会计应用:** 累计求和常用于: - 年初至今 (YTD) 收入/支出 - 现金流量表中的期末余额 - 累计折旧计算

百分比计算

计算环比增长率 (Month-over-Month):

```
# 计算环比增长率
monthly_revenue['环比增长率'] = monthly_revenue['revenue'].pct_change() *
 ↪  100
monthly_revenue[['month', 'revenue', '环比增长率']]
```

	month	revenue	环比增长率
0	2024-01	100000	NaN
1	2024-02	120000	20.000000
2	2024-03	115000	-4.166667
3	2024-04	135000	17.391304
4	2024-05	140000	3.703704
5	2024-06	150000	7.142857

计算同比增长率 (需要上年同期数据):

```
# 创建两年数据
two_years_data = pd.DataFrame({
    'month': ['2023-01', '2023-02', '2023-03', '2024-01', '2024-02',
     ↪  '2024-03'],
    'revenue': [80000, 90000, 85000, 100000, 120000, 115000]
})

# 计算同比增长率 (与 3 个月前比较, 假设季度数据)
# 实际应用中通常是 12 个月
two_years_data['同比增长率'] =
 ↪  two_years_data['revenue'].pct_change(periods=3) * 100
two_years_data
```

	month	revenue	同比增长率
0	2023-01	80000	NaN
1	2023-02	90000	NaN

	month	revenue	同比增长率
2	2023-03	85000	NaN
3	2024-01	100000	25.000000
4	2024-02	120000	33.333333
5	2024-03	115000	35.294118

计算占比 (Percentage of Total):

```
# 创建部门收入数据
dept_revenue = pd.DataFrame({
    'department': ['销售部', '服务部', '咨询部', '培训部'],
    'revenue': [500000, 300000, 150000, 50000]
})

# 计算各部门收入占比
total_revenue = dept_revenue['revenue'].sum()
dept_revenue['收入占比%'] = (dept_revenue['revenue'] / total_revenue *
 ↪  100).round(2)
dept_revenue
```

	department	revenue	收入占比%
0	销售部	500000	50.0
1	服务部	300000	30.0
2	咨询部	150000	15.0
3	培训部	50000	5.0

移动平均 (Moving Average)

计算 3 个月移动平均:

```
# 创建更长时间序列
sales_data = pd.DataFrame({
```

```
    'month': pd.date_range('2024-01', periods=12, freq='MS'),
    'sales': [100, 110, 105, 120, 115, 125, 130, 135, 128, 140, 145, 150]
})

# 计算 3 个月移动平均
sales_data['MA_3'] = sales_data['sales'].rolling(window=3).mean()
sales_data[['month', 'sales', 'MA_3']]
```

	month	sales	MA_3
0	2024-01-01	100	NaN
1	2024-02-01	110	NaN
2	2024-03-01	105	105.000000
3	2024-04-01	120	111.666667
4	2024-05-01	115	113.333333
5	2024-06-01	125	120.000000
6	2024-07-01	130	123.333333
7	2024-08-01	135	130.000000
8	2024-09-01	128	131.000000
9	2024-10-01	140	134.333333
10	2024-11-01	145	137.666667
11	2024-12-01	150	145.000000

> **Tip**
>
> **会计应用**: 移动平均可以: - 平滑季节性波动 - 识别长期趋势 - 预测未来表现

4.8.5 财务比率计算

创建财务报表数据:

```
# 创建资产负债表和利润表数据
financial_data = pd.DataFrame({
    'period': ['Q1', 'Q2', 'Q3', 'Q4'],
```

```
    'current_assets': [500000, 520000, 480000, 550000],
    'current_liabilities': [300000, 310000, 290000, 320000],
    'total_assets': [1000000, 1050000, 1020000, 1100000],
    'total_liabilities': [600000, 620000, 610000, 650000],
    'revenue': [400000, 450000, 420000, 480000],
    'net_income': [50000, 60000, 55000, 70000]
})
financial_data
```

	period	current_assets	current_liabilities	total_assets	total_liabilities	revenue	net_incon
0	Q1	500000	300000	1000000	600000	400000	50000
1	Q2	520000	310000	1050000	620000	450000	60000
2	Q3	480000	290000	1020000	610000	420000	55000
3	Q4	550000	320000	1100000	650000	480000	70000

计算流动比率 (Current Ratio):

```
# 流动比率 = 流动资产 / 流动负债
financial_data['流动比率'] = (financial_data['current_assets'] /

 ↪  financial_data['current_liabilities']).round(2)
financial_data[['period', '流动比率']]
```

	period	流动比率
0	Q1	1.67
1	Q2	1.68
2	Q3	1.66
3	Q4	1.72

计算资产负债率 (Debt-to-Asset Ratio):

```
# 资产负债率 = 总负债 / 总资产
financial_data['资产负债率%'] = (financial_data['total_liabilities'] /
                                 financial_data['total_assets'] *
 ↪  100).round(2)
financial_data[['period', '资产负债率%']]
```

	period	资产负债率%
0	Q1	60.00
1	Q2	59.05
2	Q3	59.80
3	Q4	59.09

计算净利润率 (Net Profit Margin):

```
# 净利润率 = 净利润 / 营业收入
financial_data['净利润率%'] = (financial_data['net_income'] /
                               financial_data['revenue'] * 100).round(2)
financial_data[['period', 'revenue', 'net_income', '净利润率%']]
```

	period	revenue	net_income	净利润率%
0	Q1	400000	50000	12.50
1	Q2	450000	60000	13.33
2	Q3	420000	55000	13.10
3	Q4	480000	70000	14.58

现金流量计算

创建现金流量数据:

```
# 创建现金流量数据
cash_flow = pd.DataFrame({
    'date': pd.date_range('2024-01-01', periods=10, freq='D'),
```

```
    'cash_in': [50000, 0, 30000, 45000, 0, 60000, 0, 40000, 55000, 0],
    'cash_out': [20000, 15000, 25000, 10000, 30000, 20000, 15000, 25000,
    ↪  20000, 18000]
})

# 计算每日净现金流
cash_flow['net_cash_flow'] = cash_flow['cash_in'] - cash_flow['cash_out']

# 计算现金余额（假设期初余额为 100000）
opening_balance = 100000
cash_flow['cash_balance'] = opening_balance +
↪  cash_flow['net_cash_flow'].cumsum()

cash_flow
```

	date	cash_in	cash_out	net_cash_flow	cash_balance
0	2024-01-01	50000	20000	30000	130000
1	2024-01-02	0	15000	-15000	115000
2	2024-01-03	30000	25000	5000	120000
3	2024-01-04	45000	10000	35000	155000
4	2024-01-05	0	30000	-30000	125000
5	2024-01-06	60000	20000	40000	165000
6	2024-01-07	0	15000	-15000	150000
7	2024-01-08	40000	25000	15000	165000
8	2024-01-09	55000	20000	35000	200000
9	2024-01-10	0	18000	-18000	182000

Note

练习: 1. 创建一个 12 个月的销售数据集 2. 计算每月的环比增长率 3. 计算 6 个月移动平均 4. 找出销售额低于移动平均的月份

4.9 实用工具函数

4.9.1 自定义函数

定义函数并应用到 DataFrame:

```
# 定义一个函数将余额转换为万元并四舍五入
def to_wan_yuan(x):
    return round(x / 10000, 2)

# 应用到余额列
df['balance'].transform(to_wan_yuan)
```

```
1001     5.0
1002    20.0
1122    15.0
1123    18.0
1601    30.0
2202    10.0
4001    68.0
Name: balance, dtype: float64
```

4.9.2 字符串操作

转换为小写:

```
# 将账户类型转换为小写
df['account_type'].str.lower()
```

```
1001    资产
1002    资产
1122    资产
1123    资产
1601    资产
```

```
2202        负债
4001    所有者权益
Name: account_type, dtype: object
```

4.9.3 获取字符串长度

```
# 获取账户名称的长度
df['account_name'].str.len()
```

```
1001    2
1002    4
1122    4
1123    2
1601    4
2202    4
4001    4
Name: account_name, dtype: int64
```

4.9.4 判断是否包含特定字符

```
# 判断账户名称是否包含" 账款"
df['account_name'].str.contains('账款')
```

```
1001    False
1002    False
1122     True
1123    False
1601    False
2202     True
4001    False
Name: account_name, dtype: bool
```

4.9.5 字符串索引

```
# 获取账户名称的第一个字
df['account_name'].str[0]
```

```
1001    现
1002    银
1122    应
1123    存
1601    固
2202    应
4001    实
Name: account_name, dtype: object
```

4.9.6 随机抽样

抽取指定数量的样本:

```
df.sample(2)
```

	account_name	account_type	balance	balance_direction
1002	银行存款	资产	200000.0	借
1123	存货	资产	180000.0	借

4.9.7 按比例抽样

```
df.sample(frac = 0.4)
```

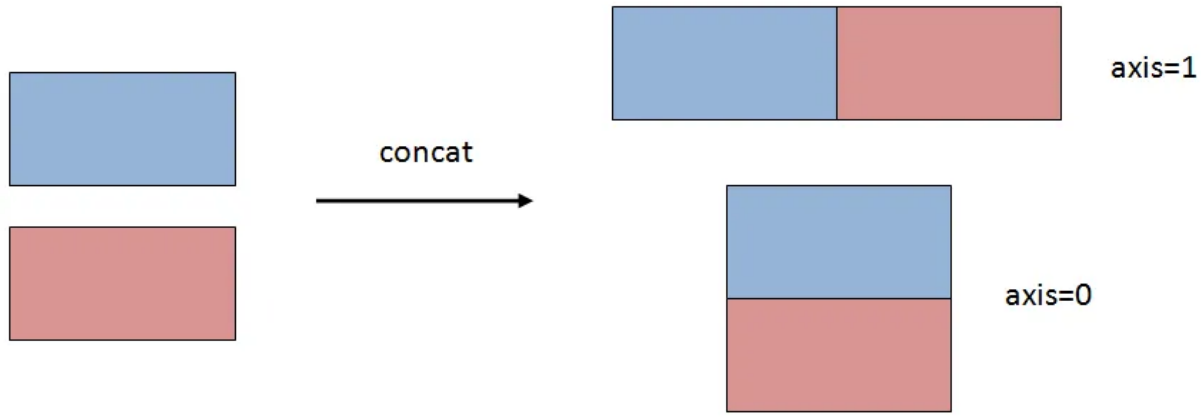
	account_name	account_type	balance	balance_direction
1601	固定资产	资产	300000.0	借
4001	实收资本	所有者权益	680000.0	贷

	account_name	account_type	balance	balance_direction
1001	现金	资产	50000.0	借

4.10 数据合并

4.10.1 concat 合并

用标签对齐合并数据 (index / columns)



创建示例数据:

```
df1 = pd.DataFrame(
  {'A': ['A0', 'A1', 'A2', 'A3'],
    'B': ['B0', 'B1', 'B2', 'B3'],
    'C': ['C0', 'C1', 'C2', 'C3'],
    'D': ['D0', 'D1', 'D2', 'D3']})


df2 = pd.DataFrame(
  {'A': ['A4', 'A5', 'A6', 'A7'],
    'B': ['B4', 'B5', 'B6', 'B7'],
    'C': ['C4', 'C5', 'C6', 'C7'],
    'D': ['D4', 'D5', 'D6', 'D7']},
```

```
  index=[4, 5, 6, 7])
```

纵向合并 (axis=0):

```
pd.concat([df1, df2], axis= 0, sort=False)
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

横向合并 (axis=1) - 创建第三个 DataFrame:

```
df3 = pd.DataFrame(
  {'B': ['B2', 'B3', 'B6', 'B7'],
    'D': ['D2', 'D3', 'D6', 'D7'],
    'F': ['F2', 'F3', 'F6', 'F7']},
  index=[2, 3, 6, 7])
df3
```

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

横向合并:

```
pd.concat([df1, df3], axis= 1, sort=False)
```

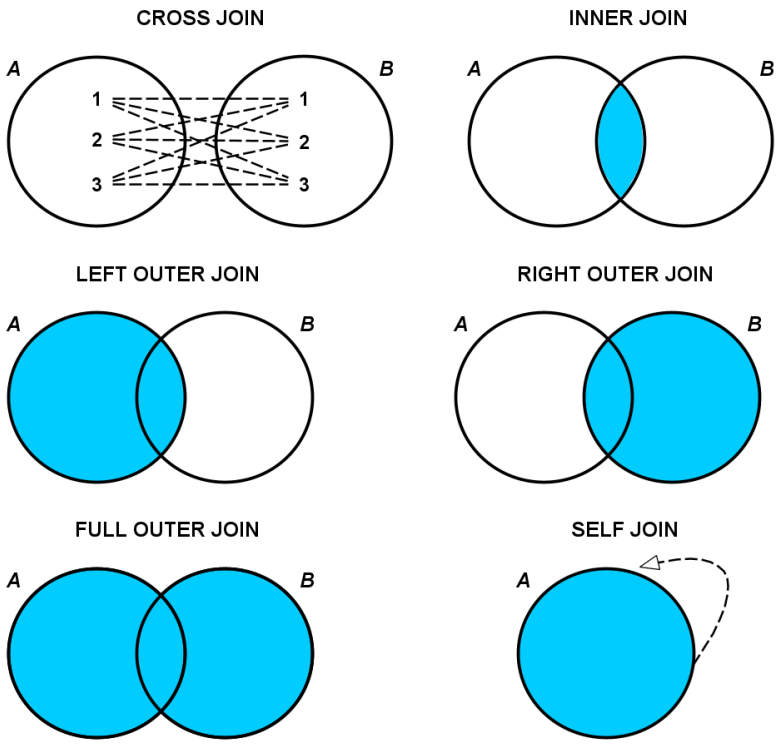
	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

4.10.2 merge 合并

用 values 合并数据, 参考文档: pandas.merge()

准备数据:

```
df1['n1'] = df1.A.str[-1]
df3['n3'] = df3.B.str[-1]
```



执行 merge 操作:

```
df1.merge(df3, left_on='n1', right_on='n3', how='inner')
```

	A	B_x	C	D_x	n1	B_y	D_y	F	n3
0	A2	B2	C2	D2	2	B2	D2	F2	2
1	A3	B3	C3	D3	3	B3	D3	F3	3

4.11 时间日期处理

创建日期数据:

```
data = {
    'date': ['2020-08-25', '2021-07-26',
    '2022-06-27', '2023-05-28', '2024-04-29'],
    'value': [10, 15, 7, 25, 30]
}

df = pd.DataFrame(data)
# Convert the 'date' column to datetime format
df['date'] = pd.to_datetime(df['date'])
df.dtypes
```

```
date     datetime64[ns]
value             int64
dtype: object
```

提取年份:

```
df.date.dt.year
```

```
0    2020
1    2021
```

```
2    2022
3    2023
4    2024
Name: date, dtype: int32
```

提取月份:

```
df.date.dt.month
```

```
0    8
1    7
2    6
3    5
4    4
Name: date, dtype: int32
```

提取周数:

```
df.date.dt.isocalendar().week
```

```
0    35
1    30
2    26
3    21
4    18
Name: week, dtype: UInt32
```

提取日:

```
df.date.dt.day
```

```
0    25
1    26
2    27
```

```
3    28
4    29
Name: date, dtype: int32
```

星期几:

```
df.date.dt.day_of_week
# Monday=0, Sunday=6
# df.date.dt.day_name()
```

```
0    1
1    0
2    0
3    6
4    0
Name: date, dtype: int32
```

一年中的第几天:

```
df.date.dt.day_of_year
```

```
0    238
1    207
2    178
3    148
4    120
Name: date, dtype: int32
```

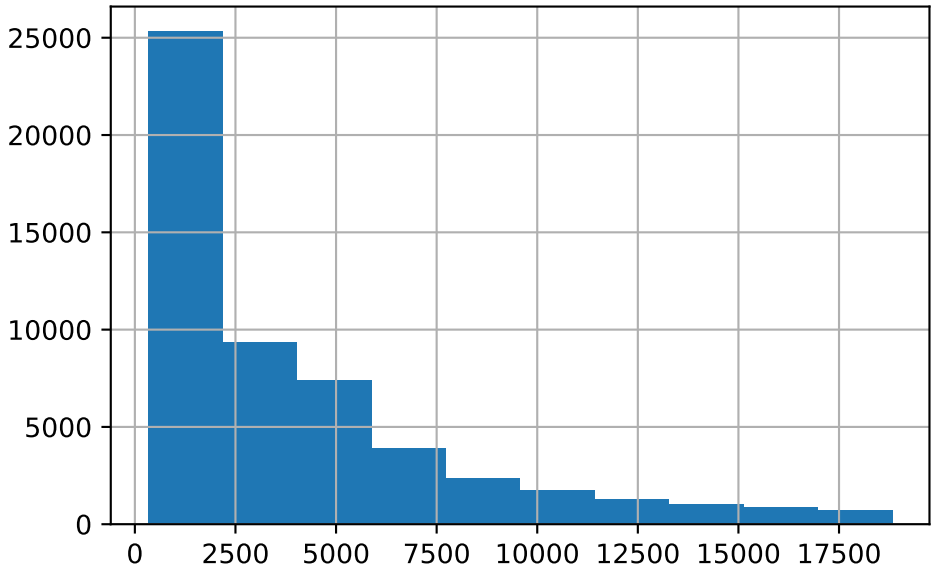
4.12 数据可视化

启用 matplotlib 内联显示:

```
%matplotlib inline
```

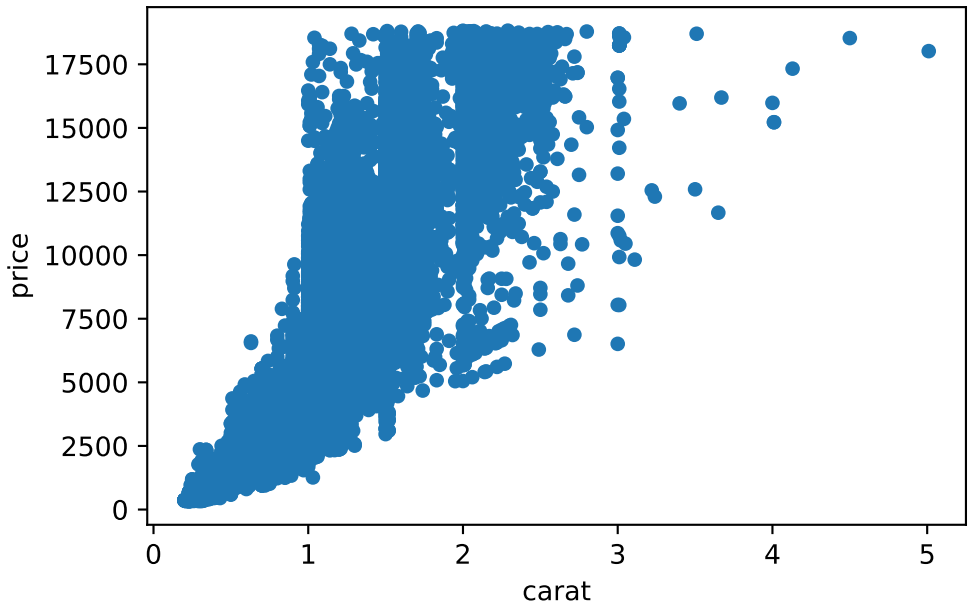
直方图:

```
from dfply import diamonds
diamonds.price.hist()
```



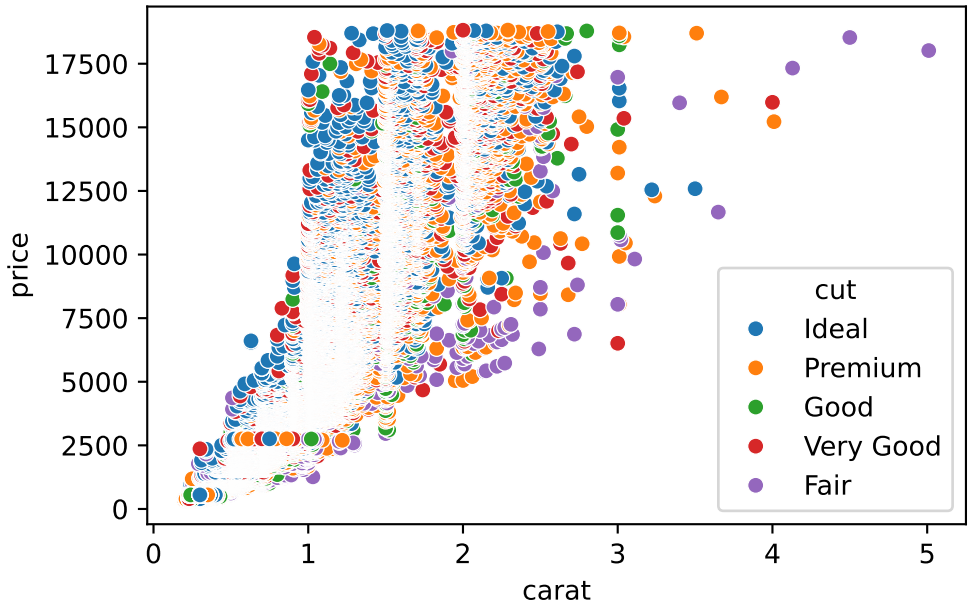
散点图:

```
from dfply import diamonds
diamonds.plot.scatter('carat', 'price')
```



seaborn 散点图:

```
import seaborn as sns
sns.scatterplot(x="carat", y="price", hue="cut", data=diamonds)
```



4.13 分组与聚合 (Grouping & Aggregation)

学习目标: - 掌握按类别分组汇总数据 - 学会多维度聚合 - 创建会计汇总报表 - 理解透视表在财务分析中的应用

4.13.1 基本分组

创建交易明细数据:

```
# 创建会计分录数据
ledger_data = pd.DataFrame({
    'date': ['2024-01-05', '2024-01-05', '2024-01-06', '2024-01-06',
    ↪  '2024-01-07', '2024-01-07'],
    'account': ['现金', '应收账款', '现金', '应收账款', '现金', '销售收
    ↪  入'],
    'category': ['资产', '资产', '资产', '资产', '资产', '收入'],
```

```
    'debit': [10000, 5000, 8000, 3000, 6000, 0],
    'credit': [0, 0, 0, 0, 0, 6000]
})
ledger_data
```

	date	account	category	debit	credit
0	2024-01-05	现金	资产	10000	0
1	2024-01-05	应收账款	资产	5000	0
2	2024-01-06	现金	资产	8000	0
3	2024-01-06	应收账款	资产	3000	0
4	2024-01-07	现金	资产	6000	0
5	2024-01-07	销售收入	收入	0	6000

按账户分组求和:

```
# 按账户汇总借贷金额
ledger_data.groupby('account').agg({
    'debit': 'sum',
    'credit': 'sum'
}).reset_index()
```

	account	debit	credit
0	应收账款	8000	0
1	现金	24000	0
2	销售收入	0	6000

按类别分组:

```
# 按账户类别汇总
ledger_data.groupby('category').agg({
    'debit': 'sum',
```

```
    'credit': 'sum'
}).reset_index()
```

	category	debit	credit
0	收入	0	6000
1	资产	32000	0

4.13.2 多维度聚合

创建更复杂的销售数据:

```
# 创建销售明细数据
sales_detail = pd.DataFrame({
    'date': pd.to_datetime(['2024-01-15', '2024-01-16', '2024-01-15',
    ↪  '2024-01-16',
                            '2024-02-15', '2024-02-16', '2024-02-15',
                            ↪  '2024-02-16']),
    'region': ['华东', '华东', '华南', '华南', '华东', '华东', '华南',
    ↪  '华南'],
    'product': ['A 产品', 'B 产品', 'A 产品', 'B 产品', 'A 产品', 'B 产
    ↪  品', 'A 产品', 'B 产品'],
    'quantity': [100, 150, 120, 80, 110, 160, 130, 90],
    'unit_price': [100, 150, 100, 150, 100, 150, 100, 150]
})

# 计算销售金额
sales_detail['amount'] = sales_detail['quantity'] *
↪  sales_detail['unit_price']
sales_detail
```

	date	region	product	quantity	unit_price	amount
0	2024-01-15	华东	A 产品	100	100	10000
1	2024-01-16	华东	B 产品	150	150	22500
2	2024-01-15	华南	A 产品	120	100	12000
3	2024-01-16	华南	B 产品	80	150	12000
4	2024-02-15	华东	A 产品	110	100	11000
5	2024-02-16	华东	B 产品	160	150	24000
6	2024-02-15	华南	A 产品	130	100	13000
7	2024-02-16	华南	B 产品	90	150	13500

多列聚合:

```
# 按地区和产品分组，计算多个统计量
sales_summary = sales_detail.groupby(['region', 'product']).agg({
    'quantity': 'sum',
    'amount': ['sum', 'mean', 'count']
}).round(2)
sales_summary
```

region	product	q s
华东	A 产品	2
	B 产品	3
华南	A 产品	2
	B 产品	1

自定义聚合函数:

```
# 按地区汇总，使用自定义聚合
regional_summary = sales_detail.groupby('region').agg({
    'amount': ['sum', 'mean', 'max', 'min'],
```

```
    'quantity': 'sum'
})
regional_summary.columns = ['总金额', '平均金额', '最大单笔', '最小单笔',
↪  '总数量']
regional_summary
```

region	总金额	平均金额	最大单笔	最小单笔	总数量
华东	67500	16875.0	24000	10000	520
华南	50500	12625.0	13500	12000	420

> **Tip**
>
> **会计应用:** 分组聚合常用于: - 按科目汇总账目 - 按部门/项目汇总费用 - 按期间汇总收入成本 - 生成试算平衡表

4.13.3 创建汇总表 (带小计和合计)

```
# 创建费用明细数据
expense_data = pd.DataFrame({
    'department': ['销售部', '销售部', '财务部', '财务部', '人力部', '人力
    ↪  部'],
    'category': ['差旅费', '办公费', '差旅费', '办公费', '差旅费', '办公
    ↪  费'],
    'amount': [5000, 2000, 3000, 1500, 2000, 1000]
})

# 按部门和类别分组
expense_summary = expense_data.groupby(['department',
↪  'category'])['amount'].sum()
print(" 明细汇总:")
print(expense_summary)
```

```
print("\n部门小计:")
print(expense_summary.groupby('department').sum())
print(f"\n总计: ¥{expense_data['amount'].sum():,.2f}")
```

```
明细汇总:
department  category
人力部         办公费         1000
            差旅费         2000
财务部         办公费         1500
            差旅费         3000
销售部         办公费         2000
            差旅费         5000
Name: amount, dtype: int64

部门小计:
department
人力部    3000
财务部    4500
销售部    7000
Name: amount, dtype: int64

总计: ¥14,500.00
```

4.13.4 条件聚合

```
# 创建应收账款账龄数据
receivables = pd.DataFrame({
    'customer': ['客户 A', '客户 B', '客户 C', '客户 D', '客户 E'],
    'invoice_date': pd.to_datetime(['2024-01-01', '2024-02-15',
    ↪  '2024-03-10',
                                    '2024-01-20', '2024-02-28']),
    'amount': [50000, 30000, 40000, 25000, 35000],
```

```
    'paid': [True, False, False, True, False]
})

# 计算账龄（假设今天是 2024-04-01）
current_date = pd.to_datetime('2024-04-01')
receivables['days_outstanding'] = (current_date -
 ↪  receivables['invoice_date']).dt.days

receivables
```

	customer	invoice_date	amount	paid	days_outstanding
0	客户 A	2024-01-01	50000	True	91
1	客户 B	2024-02-15	30000	False	46
2	客户 C	2024-03-10	40000	False	22
3	客户 D	2024-01-20	25000	True	72
4	客户 E	2024-02-28	35000	False	33

按付款状态分组:

```
# 按付款状态汇总
payment_summary = receivables.groupby('paid').agg({
    'amount': ['sum', 'count'],
    'days_outstanding': 'mean'
})
payment_summary.columns = ['总金额', '笔数', '平均账龄']
payment_summary
```

paid	总金额	笔数	平均账龄
False	105000	3	33.666667
True	75000	2	81.500000

创建账龄分析:

```
# 创建账龄区间
receivables['aging_bucket'] = pd.cut(receivables['days_outstanding'],
                                     bins=[0, 30, 60, 90, float('inf')],
                                     labels=['0-30 天', '31-60 天',
 ↪  '61-90 天', '90 天以上'])

# 按账龄区间汇总
aging_analysis =
 ↪  receivables.groupby('aging_bucket')['amount'].agg(['sum', 'count'])
aging_analysis.columns = ['金额', '笔数']
aging_analysis
```

	金额	笔数
aging_bucket		
0-30 天	40000	1
31-60 天	65000	2
61-90 天	25000	1
90 天以上	50000	1

> **Note**
>
> **练习:** 1. 创建一个包含多个月销售数据的 DataFrame 2. 按月份和产品分组, 计算销售总额 3. 找出每月销售最高的产品 4. 计算每个产品的月平均销售额

4.13.5 Excel 风格的对比

会计师常用的 Excel 功能在 pandas 中的对应操作:

Excel 功能	Pandas 方法	说明
SUMIF	groupby().sum()	条件求和
COUNTIF	groupby().count()	条件计数

Excel 功能	Pandas 方法	说明
AVERAGEIF	groupby().mean()	条件平均
Subtotal	groupby() + agg()	分类小计
Pivot Table	pivot_table()	数据透视表

> **Tip**
>
> 从 **Excel** 到 **Pandas**: 如果你熟悉 Excel 的数据透视表,pandas 的 `pivot_table()` 和 `groupby()` 可以实现相同甚至更强大的功能。

4.13.6 数据透视表 (Pivot Tables)

学习目标: - 掌握创建数据透视表 - 学会多维度数据分析 - 创建财务交叉报表 - 生成专业的财务分析报告

pivot() 基础

创建销售数据:

```
# 创建简单的销售数据
simple_sales = pd.DataFrame({
    'date': ['2024-01', '2024-01', '2024-02', '2024-02'],
    'region': ['华东', '华南', '华东', '华南'],
    'sales': [200000, 150000, 220000, 180000]
})
simple_sales
```

	date	region	sales
0	2024-01	华东	200000
1	2024-01	华南	150000
2	2024-02	华东	220000
3	2024-02	华南	180000

创建数据透视表:

```
# 将数据重塑为表格形式
pivot_df = simple_sales.pivot(
    index='date',
    columns='region',
    values='sales')
pivot_df
```

region	华东	华南
date		
2024-01	200000	150000
2024-02	220000	180000

- **pivot()**: 通过将一列的唯一值转换为新列来重塑数据。适用于没有重复索引的情况。
- **pivot_table()**: 用于对数据进行分组、汇总和聚合，可以根据数据的不同维度计算聚合值。可以处理重复值。

pivot_table() 高级应用

创建更复杂的财务数据:

```
# 创建包含重复值的月度财务数据
financial_trans = pd.DataFrame({
    'month': ['1 月', '1 月', '1 月', '2 月', '2 月', '2 月', '3 月', '3
    ↪  月', '3 月'] * 2,
    'category': ['收入', '成本', '费用'] * 6,
    'department': ['销售部']*9 + ['服务部']*9,
    'amount': [500000, 300000, 80000, 520000, 310000, 85000, 480000,
    ↪  290000, 75000,
               300000, 180000, 50000, 310000, 185000, 52000, 290000,
                ↪  175000, 48000]
})
financial_trans.head(10)
```

	month	category	department	amount
0	1 月	收入	销售部	500000
1	1 月	成本	销售部	300000
2	1 月	费用	销售部	80000
3	2 月	收入	销售部	520000
4	2 月	成本	销售部	310000
5	2 月	费用	销售部	85000
6	3 月	收入	销售部	480000
7	3 月	成本	销售部	290000
8	3 月	费用	销售部	75000
9	1 月	收入	服务部	300000

创建部门-类别透视表:

```
# 创建透视表，按部门和类别汇总
dept_pivot = pd.pivot_table(financial_trans,
                            values='amount',
                            index='department',
                            columns='category',
                            aggfunc='sum',
                            fill_value=0)
dept_pivot
```

category department	成本	收入	费用
服务部	540000	900000	150000
销售部	900000	1500000	240000

添加行列汇总:

```
# 添加小计和合计
dept_pivot_total = pd.pivot_table(financial_trans,
                                  values='amount',
                                  index='department',
                                  columns='category',
                                  aggfunc='sum',
                                  fill_value=0,
                                  margins=True,  # 添加汇总行和列
                                  margins_name='总计')
dept_pivot_total
```

category department	成本	收入	费用	总计
服务部	540000	900000	150000	1590000
销售部	900000	1500000	240000	2640000
总计	1440000	2400000	390000	4230000

计算利润:

```
# 创建一个包含计算列的透视表副本
profit_analysis = dept_pivot.copy()
profit_analysis['利润'] = profit_analysis['收入'] - profit_analysis['成
↪  本'] - profit_analysis['费用']
profit_analysis['利润率%'] = (profit_analysis['利润'] /
↪  profit_analysis['收入'] * 100).round(2)
profit_analysis
```

category department	成本	收入	费用	利润	利润率%
服务部	540000	900000	150000	210000	23.33
销售部	900000	1500000	240000	360000	24.00

> **Tip**
>
> **会计应用**: 数据透视表在财务分析中常用于: - 多维度收入分析 (按产品、地区、时间) - 成本费用分析 (按部门、类别、期间) - 预算 vs 实际对比 - 盈利能力分析

多级透视表

创建更详细的数据:

```
# 创建季度-月度-产品销售数据
detailed_sales = pd.DataFrame({
    'quarter': ['Q1', 'Q1', 'Q1', 'Q2', 'Q2', 'Q2'] * 4,
    'month': ['1 月', '2 月', '3 月', '4 月', '5 月', '6 月'] * 4,
    'product': ['产品 A']*6 + ['产品 B']*6 + ['产品 C']*6 + ['产品 D']*6,
    'revenue': [100, 110, 105, 120, 115, 125,
                80, 85, 82, 90, 88, 95,
                60, 65, 62, 70, 68, 72,
                50, 55, 52, 60, 58, 62]
})

# 创建多级索引的透视表
multi_pivot = pd.pivot_table(detailed_sales,
                             values='revenue',
                             index=['quarter', 'product'],
                             columns='month',
                             aggfunc='sum',
                             fill_value=0)
multi_pivot
```

	month	1
quarter	product	
Q1	产品 A	1
	产品 B	8

quarter	month product	1
Q2	产品 C	6
	产品 D	5
	产品 A	0
	产品 B	0
	产品 C	0
	产品 D	0

交叉表 (Cross-tabulation)

交叉表用于统计频次:

```
# 创建交易批准数据
approval_data = pd.DataFrame({
    'department': ['销售部', '采购部', '销售部', '财务部', '采购部',
                   '销售部', '财务部', '采购部', '销售部', '财务部'],
    'status': ['已批准', '已批准', '待审批', '已批准', '已拒绝',
               '已批准', '待审批', '已批准', '已拒绝', '已批准'],
    'amount': [50000, 30000, 45000, 25000, 20000,
               55000, 28000, 35000, 40000, 22000]
})

# 创建交叉表统计各状态的笔数
status_crosstab = pd.crosstab(approval_data['department'],
                              approval_data['status'],
                              margins=True,
                              margins_name='总计')
status_crosstab
```

status department	已批准	已拒绝	待审批	总计
财务部	2	0	1	3
采购部	2	1	0	3
销售部	2	1	1	4
总计	6	2	2	10

交叉表 with 汇总金额:

```
# 统计各状态的总金额
amount_crosstab = pd.crosstab(approval_data['department'],
                              approval_data['status'],
                              values=approval_data['amount'],
                              aggfunc='sum',
                              margins=True,
                              margins_name='总计')
amount_crosstab.fillna(0)
```

status department	已批准	已拒绝	待审批	总计
财务部	47000.0	0.0	28000.0	75000
采购部	65000.0	20000.0	0.0	85000
销售部	105000.0	40000.0	45000.0	190000
总计	217000.0	60000.0	73000.0	350000

> **Note**
>
> 练习: 1. 创建一个包含多个产品线、多个地区、多个月份的销售数据 2. 使用 pivot_table 创建地区 × 产品的销售金额透视表 3. 添加 margins 参数显示小计 4. 计算每个地区各产品的销售占比

stack 和 unstack

stack 操作 (列转行):

```
pivot_df.stack()
```

```
date       region
2024-01    华东        200000
           华南        150000
2024-02    华东        220000
           华南        180000
dtype: int64
```

unstack 操作 (行转列):

```
pivot_df.stack().unstack()
```

region	华东	华南
date		
2024-01	200000	150000
2024-02	220000	180000

- **unstack()**: 将最内层的行索引转换为列。
- **stack()**: 将最内层的列索引转换为行索引，创建一个具有多重索引的 Series。

4.14 综合案例：销售收入分析

案例背景: 某零售公司需要分析 2024 年第一季度的销售情况, 包括不同产品线、不同地区的销售表现, 以及应收账款的管理情况。

4.14.1 步骤 1: 准备数据

```
# 创建销售交易数据
sales_transactions = pd.DataFrame({
    'transaction_id': ['T001', 'T002', 'T003', 'T004', 'T005', 'T006',
    ↪  'T007', 'T008',
```

```
                    'T009', 'T010', 'T011', 'T012', 'T013', 'T014',
                    ↪ 'T015'],
    'date': pd.to_datetime(['2024-01-05', '2024-01-08', '2024-01-12',
    ↪ '2024-01-15', '2024-01-20',
                            '2024-02-03', '2024-02-07', '2024-02-14',
                            ↪ '2024-02-18', '2024-02-25',
                            '2024-03-05', '2024-03-10', '2024-03-15',
                            ↪ '2024-03-20', '2024-03-28']),
    'customer': ['客户 A', '客户 B', '客户 A', '客户 C', '客户 B',
                 '客户 A', '客户 D', '客户 C', '客户 B', '客户 E',
                 '客户 A', '客户 D', '客户 C', '客户 E', '客户 B'],
    'product_line': ['电子产品', '家居用品', '电子产品', '服装', '家居用
    ↪ 品',
                     '电子产品', '服装', '服装', '电子产品', '家居用品',
                     '电子产品', '家居用品', '服装', '电子产品', '家居用
                     ↪ 品'],
    'region': ['华东', '华南', '华东', '华北', '华南',
               '华东', '华北', '华北', '华南', '华中',
               '华东', '华北', '华北', '华中', '华南'],
    'quantity': [50, 30, 45, 60, 35, 55, 40, 50, 38, 42, 52, 45, 55, 48,
    ↪ 40],
    'unit_price': [2000, 500, 2000, 300, 500, 2000, 300, 300, 2000, 500,
    ↪ 2000, 500, 300, 2000, 500],
    'payment_status': ['已付', '未付', '已付', '未付', '已付',
                       '已付', '未付', '已付', '未付', '已付',
                       '已付', '未付', '未付', '已付', '未付']
})

# 计算销售金额
sales_transactions['amount'] = sales_transactions['quantity'] *
↪ sales_transactions['unit_price']
```

```
print(f" 共{len(sales_transactions)}笔交易")
sales_transactions.head()
```

共15笔交易

	transaction_id	date	customer	product_line	region	quantity	unit_price	paymen
0	T001	2024-01-05	客户 A	电子产品	华东	50	2000	已付
1	T002	2024-01-08	客户 B	家居用品	华南	30	500	未付
2	T003	2024-01-12	客户 A	电子产品	华东	45	2000	已付
3	T004	2024-01-15	客户 C	服装	华北	60	300	未付
4	T005	2024-01-20	客户 B	家居用品	华南	35	500	已付

4.14.2 步骤 2: 数据清洗与验证

```
# 检查数据质量
print("=== 数据质量检查 ===")
print(f" 缺失值检查:")
print(sales_transactions.isnull().sum())

print(f"\n重复交易 ID 检查:")
duplicate_ids = sales_transactions['transaction_id'].duplicated().sum()
print(f" 重复的交易 ID 数量: {duplicate_ids}")

print(f"\n金额异常值检查:")
print(sales_transactions['amount'].describe())

# 验证数据完整性
assert sales_transactions['transaction_id'].is_unique, " 存在重复的交易
 ↪  ID!"
print("\n  数据验证通过")
```

```
=== 数据质量检查 ===
缺失值检查:
transaction_id    0
date              0
customer          0
product_line      0
region            0
quantity          0
unit_price        0
payment_status    0
amount            0
dtype: int64

重复交易ID检查:
重复的交易ID数量: 0

金额异常值检查:
count        15.000000
mean      48900.000000
std       40517.720991
min       12000.000000
25%       17000.000000
50%       21000.000000
75%       93000.000000
max      110000.000000
Name: amount, dtype: float64
```

数据验证通过

4.14.3 步骤 3: 财务分析

3.1 总体销售概况

```
# 计算总体指标
total_revenue = sales_transactions['amount'].sum()
total_transactions = len(sales_transactions)
average_transaction = sales_transactions['amount'].mean()

print("=== 第一季度销售概况 ===")
print(f" 总销售额: ¥{total_revenue:,.2f}")
print(f" 交易笔数: {total_transactions}")
print(f" 平均交易额: ¥{average_transaction:,.2f}")

# 按付款状态分析
payment_summary =
 ↪  sales_transactions.groupby('payment_status')['amount'].agg(['sum',
 ↪  'count'])
payment_summary.columns = ['金额', '笔数']
print("\n按付款状态分类:")
print(payment_summary)

收款率 = payment_summary.loc['已付', '金额'] / total_revenue * 100
print(f"\n收款率: {收款率:.2f}%")
```

```
=== 第一季度销售概况 ===
总销售额: ¥733,500.00
交易笔数: 15
平均交易额: ¥48,900.00

按付款状态分类:
                金额  笔数
payment_status
已付            553500   8
未付            180000   7
```

收款率：75.46%

3.2 产品线分析

```
# 产品线业绩分析
product_analysis = sales_transactions.groupby('product_line').agg({
    'amount': ['sum', 'mean', 'count'],
    'quantity': 'sum'
}).round(2)

product_analysis.columns = ['销售总额', '平均订单额', '订单数', '销售数
↪  量']

# 计算各产品线占比
product_analysis['收入占比%'] = (product_analysis['销售总额'] /
↪  total_revenue * 100).round(2)

# 按销售额排序
product_analysis = product_analysis.sort_values('销售总额',
↪  ascending=False)

print("=== 产品线业绩分析 ===")
print(product_analysis)
```

```
=== 产品线业绩分析 ===
              销售总额     平均订单额  订单数  销售数量  收入占比%
product_line
电子产品          576000  96000.0     6   288  78.53
家居用品           96000  19200.0     5   192  13.09
服装               61500  15375.0     4   205   8.38
```

3.3 地区分析

```python
# 创建地区-产品线透视表
regional_pivot = pd.pivot_table(sales_transactions,
                                values='amount',
                                index='region',
                                columns='product_line',
                                aggfunc='sum',
                                fill_value=0,
                                margins=True,
                                margins_name='总计')

print("=== 地区 × 产品线销售矩阵 ===")
print(regional_pivot)

# 找出各地区的主打产品
print("\n各地区销售最高的产品线:")
for region in regional_pivot.index[:-1]:  # 排除总计行
    top_product = regional_pivot.loc[region].drop('总计').idxmax()
    top_amount = regional_pivot.loc[region, top_product]
    print(f"{region}: {top_product} (¥{top_amount:,.2f})")
```

```
=== 地区×产品线销售矩阵 ===
product_line   家居用品      服装     电子产品       总计
region
华东                0       0   404000   404000
华中            21000       0    96000   117000
华北            22500   61500        0    84000
华南            52500       0    76000   128500
总计            96000   61500   576000   733500

各地区销售最高的产品线:
华东: 电子产品 (¥404,000.00)
华中: 电子产品 (¥96,000.00)
```

华北: 服装 (¥61,500.00)

华南: 电子产品 (¥76,000.00)

3.4 时间趋势分析

```
# 添加月份列
sales_transactions['month'] =
 ↪  sales_transactions['date'].dt.to_period('M')

# 按月统计
monthly_revenue = sales_transactions.groupby('month').agg({
    'amount': 'sum',
    'transaction_id': 'count'
})
monthly_revenue.columns = ['销售额', '交易笔数']

# 计算环比增长率
monthly_revenue['环比增长率%'] = monthly_revenue['销售额'].pct_change() *
 ↪  100

print("=== 月度销售趋势 ===")
print(monthly_revenue)

# 计算累计收入
monthly_revenue['累计收入'] = monthly_revenue['销售额'].cumsum()
print("\n包含累计收入:")
print(monthly_revenue[['销售额', '累计收入']])
```

```
=== 月度销售趋势 ===
            销售额  交易笔数      环比增长率%
month
2024-01  240500     5        NaN
2024-02  234000     5  -2.702703
2024-03  259000     5  10.683761
```

包含累计收入:

```
            销售额     累计收入
month
2024-01   240500   240500
2024-02   234000   474500
2024-03   259000   733500
```

3.5 客户分析

```
# 客户价值分析
customer_analysis = sales_transactions.groupby('customer').agg({
    'amount': 'sum',
    'transaction_id': 'count'
})
customer_analysis.columns = ['总购买额', '购买次数']

# 计算平均订单价值
customer_analysis['平均订单额'] = customer_analysis['总购买额'] /
 ↪  customer_analysis['购买次数']

# 按总购买额排序
customer_analysis = customer_analysis.sort_values('总购买额',
 ↪  ascending=False)

# 计算客户贡献度
customer_analysis['收入占比%'] = (customer_analysis['总购买额'] /
 ↪  total_revenue * 100).round(2)

print("=== 客户价值分析 ===")
print(customer_analysis)

# 识别 VIP 客户 (前 20%)
```

```
vip_threshold = customer_analysis['总购买额'].quantile(0.8)
vip_customers = customer_analysis[customer_analysis['总购买额'] >=
 ↪  vip_threshold]
print(f"\nVIP 客户 (前 20%):")
print(vip_customers)
```

```
=== 客户价值分析 ===
            总购买额  购买次数      平均订单额  收入占比%
customer
客户A       404000     4  101000.0  55.08
客户B       128500     4   32125.0  17.52
客户E       117000     2   58500.0  15.95
客户C        49500     3   16500.0   6.75
客户D        34500     2   17250.0   4.70

VIP客户(前20%):
            总购买额  购买次数      平均订单额  收入占比%
customer
客户A       404000     4  101000.0  55.08
```

4.14.4 步骤 4: 应收账款分析

```
# 筛选未付款交易
receivables = sales_transactions[sales_transactions['payment_status'] ==
 ↪  '未付'].copy()

# 计算账龄 (截至 2024-04-01)
current_date = pd.to_datetime('2024-04-01')
receivables['days_outstanding'] = (current_date -
 ↪  receivables['date']).dt.days

# 创建账龄区间
```

```
receivables['aging_bucket'] = pd.cut(receivables['days_outstanding'],
                                     bins=[0, 30, 60, 90, float('inf')],
                                     labels=['0-30 天', '31-60 天',
 ↪  '61-90 天', '90 天以上'])

print("=== 应收账款账龄分析 ===")
print(f" 应收账款总额: ¥{receivables['amount'].sum():,.2f}")
print(f" 应收笔数: {len(receivables)}")

# 账龄分布
aging_analysis =
 ↪  receivables.groupby('aging_bucket')['amount'].agg(['sum', 'count'])
aging_analysis.columns = ['金额', '笔数']
aging_analysis['占比%'] = (aging_analysis['金额'] /
 ↪  receivables['amount'].sum() * 100).round(2)
print("\n账龄分布:")
print(aging_analysis)

# 按客户统计应收
customer_receivables = receivables.groupby('customer').agg({
    'amount': 'sum',
    'days_outstanding': 'mean'
}).round(2)
customer_receivables.columns = ['应收金额', '平均账龄']
customer_receivables = customer_receivables.sort_values('应收金额',
 ↪  ascending=False)

print("\n按客户应收账款:")
print(customer_receivables)
```

```
=== 应收账款账龄分析 ===
应收账款总额: ¥180,000.00
```

应收笔数: 7

账龄分布:

aging_bucket	金额	笔数	占比%
0-30天	59000	3	32.78
31-60天	88000	2	48.89
61-90天	33000	2	18.33
90天以上	0	0	0.00

按客户应收账款:

customer	应收金额	平均账龄
客户B	111000	43.67
客户C	34500	47.00
客户D	34500	38.00

4.14.5 步骤 5: 生成管理报告

```
print("=" * 60)
print("          2024 年第一季度销售分析报告")
print("=" * 60)

print(f"\n一、总体业绩")
print(f"   销售总额: ¥{total_revenue:,.2f}")
print(f"   交易笔数: {total_transactions}")
print(f"   平均订单: ¥{average_transaction:,.2f}")
print(f"   收款率: {收款率:.2f}%")

print(f"\n二、产品线表现 (按销售额排序)")
for idx, row in product_analysis.head(3).iterrows():
    print(f"   {idx}: ¥{row['销售总额']:,.2f} ({row['收入占比%']}%)")
```

```
print(f"\n三、地区表现（销售总额排序）")
regional_total = regional_pivot.iloc[:-1,
 ↪  -1].sort_values(ascending=False)
for region, amount in regional_total.head(3).items():
    pct = amount / total_revenue * 100
    print(f"   {region}: ¥{amount:,.2f} ({pct:.2f}%)")

print(f"\n四、TOP3 客户")
for idx, row in customer_analysis.head(3).iterrows():
    print(f"   {idx}: ¥{row['总购买额']:,.2f} ({row['收入占比%']}%),
     ↪  {row['购买次数']}次购买")

print(f"\n五、应收账款预警")
print(f"   应收总额: ¥{receivables['amount'].sum():,.2f}")
print(f"   超 60 天应收: ¥{aging_analysis.loc[['61-90 天', '90 天以上'],
 ↪  '金额'].sum():,.2f}")
over_60_pct = aging_analysis.loc[['61-90 天', '90 天以上'], '金额'].sum()
 ↪  / receivables['amount'].sum() * 100
print(f"   账龄风险: {over_60_pct:.2f}% 超过 60 天")

print("\n" + "=" * 60)
```

```
============================================================
          2024年第一季度销售分析报告
============================================================

一、总体业绩
   销售总额: ¥733,500.00
   交易笔数: 15
   平均订单: ¥48,900.00
   收款率: 75.46%
```

二、产品线表现(按销售额排序)

电子产品: ¥576,000.00 (78.53%)

家居用品: ¥96,000.00 (13.09%)

服装: ¥61,500.00 (8.38%)

三、地区表现(销售总额排序)

华东: ¥404,000.00 (55.08%)

华南: ¥128,500.00 (17.52%)

华中: ¥117,000.00 (15.95%)

四、TOP3客户

客户A: ¥404,000.00 (55.08%), 4.0次购买

客户B: ¥128,500.00 (17.52%), 4.0次购买

客户E: ¥117,000.00 (15.95%), 2.0次购买

五、应收账款预警

应收总额: ¥180,000.00

超60天应收: ¥33,000.00

账龄风险: 18.33% 超过60天

==========

Note

案例小结:

这个综合案例展示了如何使用 pandas 进行完整的财务数据分析:

1. **数据准备:** 创建和导入交易数据
2. **数据清洗:** 检查缺失值、重复值、异常值
3. **多维分析:** 产品线、地区、时间、客户等多个维度
4. **财务计算:** 累计、占比、增长率等关键指标
5. **风险评估:** 应收账款账龄分析
6. **报告生成:** 汇总关键发现

实践建议: - 尝试修改数据, 观察分析结果的变化 - 添加更多维度的分析 (如利润分析、成本分析) - 导出结果到 Excel, 制作可视化图表 - 结合实际业务场景, 思考如何应用这些技术

4.15 总结与下一步

4.15.1 Pandas 核心概念回顾

通过本章学习, 你已经掌握了:

基础操作: - DataFrame 和 Series 的创建和基本属性 - 数据导入导出 (CSV, Excel) - 数据选择和筛选 (loc, iloc, 条件筛选)

数据处理: - 数据清洗 (缺失值、重复值、类型转换) - 数据验证和质量检查 - 数据变形 (pivot, melt, stack/unstack)

财务计算: - 累计求和 (cumsum) - 百分比计算 (pct_change) - 移动平均 (rolling) - 财务比率计算

数据聚合: - 分组统计 (groupby) - 多维聚合 (pivot_table) - 交叉表分析 (crosstab)

4.15.2 与 Excel 的对比

任务	Excel	Pandas	优势
数据处理能力	百万行限制	几乎无限制	Pandas 处理大数据更快
重复性任务	需要手动	可编程自动化	Pandas 节省时间
数据清洗	需要多步操作	一行代码完成	Pandas 更高效
复杂计算	公式复杂	代码清晰	Pandas 更易维护
版本控制	困难	代码易于追踪	Pandas 更专业

4.15.3 学习建议

1. **多练习:** 用实际财务数据练习, 理解更深刻
2. **查文档:** pandas 官方文档非常详细, 遇到问题先查文档
3. **循序渐进:** 从简单操作开始, 逐步掌握复杂功能
4. **结合业务:** 思考如何将这些技术应用到实际会计工作中

4.15.4 下一步学习方向

- **数据可视化:** 使用 matplotlib 和 seaborn 创建财务图表
- **数据库操作:** 学习 SQL, 直接从数据库读取财务数据
- **自动化报表:** 结合 Excel 库 (openpyxl) 生成格式化的财务报表
- **高级分析:** 时间序列分析、预测模型等

4.15.5 推荐资源

- Pandas 官方文档
- Pandas 速查表
- 10 minutes to pandas

> **Important**
>
> **记住:** Pandas 是一个强大的工具, 但工具只是手段, 重要的是理解财务数据背后的业务逻辑和会计原理。技术与专业知识相结合, 才能发挥最大价值。

5 SQL 入门与实战

点击下载本章节代码

5.1 SQL 语言简介

SQL（Structured Query Language）是一种用于管理关系型数据库的标准语言。它允许用户创建、查询、更新和删除数据库中的数据。SQL 是数据库管理系统（如 MySQL、PostgreSQL、SQLite 等）的核心工具。

5.1.1 数据库基础概念

- 数据库（**Database**）：数据的集合。
- 表（**Table**）：数据库中的数据结构，由行和列组成。
- 行（**Row**）：表中的一条记录。
- 列（**Column**）：表中的一个字段。
- 主键（**Primary Key**）：唯一标识每行的列。
- 外键（**Foreign Key**）：引用另一个表的主键。

> Tip
>
> 会计应用: 在会计系统中，数据库用于存储：- 会计科目表 (Chart of Accounts) - 记账凭证 (Journal Entries) - 总分类账 (General Ledger) - 明细账 (Subsidiary Ledgers) - 财务报表数据

5.1.2 SQLite3 介绍

SQLite3 是一个轻量级的、嵌入式的关系型数据库管理系统。它不需要单独的服务器进程，直接在应用程序中运行，非常适合小型项目、原型开发和嵌入式系统。SQLite3 支持标准的 SQL 语法，并且是 ACID 兼容的。

SQLite3 的优势

- 无需安装服务器
- 零配置
- 单个文件数据库
- 跨平台兼容
- 免费开源

5.2 DB Browser for SQLite GUI 工具

DB Browser for SQLite 是一个免费的开源 GUI 工具，用于查看和编辑 SQLite 数据库文件。

5.2.1 安装和使用

1. 下载并安装 DB Browser for SQLite：https://sqlitebrowser.org/
2. 打开工具，选择 “Open Database” 并选择你的 `.db` 文件
3. 在 GUI 中可以：
    - 浏览表结构
    - 查看和编辑数据
    - 执行 SQL 查询
    - 导入/导出数据

5.2.2 优势

- 直观的图形界面
- 无需编写代码即可操作数据库
- 支持 SQL 查询执行
- 可以导出数据为 CSV、SQL 等格式

5.3 使用 Python 接入 SQLite3

Python 标准库中包含了 `sqlite3` 模块，可以直接使用，无需额外安装。

5.3.1 创建数据库和表

学习目标: - 理解 SQL 建表语句 - 掌握数据类型定义 - 学会设置主键和约束

让我们创建一个会计科目表 (Chart of Accounts):

```
import sqlite3

# 连接到数据库（如果不存在会自动创建）
conn = sqlite3.connect('accounting.db')
cursor = conn.cursor()

# 创建会计科目表
cursor.execute('''
CREATE TABLE IF NOT EXISTS chart_of_accounts (
    account_code TEXT PRIMARY KEY,
    account_name TEXT NOT NULL,
    account_type TEXT NOT NULL,
    parent_code TEXT,
    balance_direction TEXT,
    is_active INTEGER DEFAULT 1
)
''')

conn.commit()
print(" 会计科目表创建成功")
```

会计科目表创建成功

> **Note**
>
> **字段说明:** - `account_code`: 科目编码 (主键) - `account_name`: 科目名称 - `account_type`: 科目类型 (资产/负债/所有者权益/收入/费用) - `parent_code`: 上级科目编码 - `balance_direction`: 余额方向 (借/贷) - `is_active`: 是否启用 (1= 启用, 0= 停用)

插入数据

```
# 插入会计科目数据
accounts = [
    ('1001', '库存现金', '资产', None, '借', 1),
    ('1002', '银行存款', '资产', None, '借', 1),
    ('1122', '应收账款', '资产', None, '借', 1),
    ('1123', '预付账款', '资产', None, '借', 1),
    ('1401', '存货', '资产', None, '借', 1),
    ('2202', '应付账款', '负债', None, '贷', 1),
    ('2203', '预收账款', '负债', None, '贷', 1),
    ('4001', '实收资本', '所有者权益', None, '贷', 1),
    ('6001', '主营业务收入', '收入', None, '贷', 1),
    ('6401', '主营业务成本', '费用', None, '借', 1)
]

cursor.executemany('''
INSERT OR REPLACE INTO chart_of_accounts
(account_code, account_name, account_type, parent_code,
 ↪  balance_direction, is_active)
VALUES (?, ?, ?, ?, ?, ?)
''', accounts)

conn.commit()
print(f" 成功插入{len(accounts)}个会计科目")
```

成功插入10个会计科目

查询数据

```
# 查询所有科目
cursor.execute("SELECT * FROM chart_of_accounts")
```

```
rows = cursor.fetchall()
print(" 所有会计科目: ")
for row in rows[:5]:  # 只显示前 5 条
    print(row)

# 条件查询: 查询资产类科目
cursor.execute("""
SELECT account_code, account_name, balance_direction
FROM chart_of_accounts
WHERE account_type = ?
""", ('资产',))
rows = cursor.fetchall()
print("\n资产类科目: ")
for row in rows:
    print(f" 科目编码: {row[0]}, 科目名称: {row[1]}, 余额方向: {row[2]}")
```

```
所有会计科目:
('1001', '库存现金', '资产', None, '借', 1)
('1002', '银行存款', '资产', None, '借', 1)
('1122', '应收账款', '资产', None, '借', 1)
('1123', '预付账款', '资产', None, '借', 1)
('1401', '存货', '资产', None, '借', 1)

资产类科目:
科目编码: 1001, 科目名称: 库存现金, 余额方向: 借
科目编码: 1002, 科目名称: 银行存款, 余额方向: 借
科目编码: 1122, 科目名称: 应收账款, 余额方向: 借
科目编码: 1123, 科目名称: 预付账款, 余额方向: 借
科目编码: 1401, 科目名称: 存货, 余额方向: 借
```

更新和删除数据

```
# 更新数据: 修改科目名称
cursor.execute("""
UPDATE chart_of_accounts
SET account_name = ?
WHERE account_code = ?
""", ('现金', '1001'))
conn.commit()
print(" 库存现金科目名称已更新为'现金'")

# 停用某个科目（软删除，不是物理删除）
cursor.execute("""
UPDATE chart_of_accounts
SET is_active = 0
WHERE account_code = ?
""", ('1123',))
conn.commit()
print(" 预付账款科目已停用")

# 再次查询启用的科目
cursor.execute("""
SELECT account_code, account_name, account_type, is_active
FROM chart_of_accounts
ORDER BY account_code
""")
rows = cursor.fetchall()
print("\n更新后的科目列表: ")
for row in rows[:7]:
    status = " " if row[3] == 1 else " "
    print(f"{status} {row[0]} - {row[1]} ({row[2]})")
```

```
库存现金科目名称已更新为'现金'
```

预付账款科目已停用

更新后的科目列表:
1001 - 现金 (资产)
1002 - 银行存款 (资产)
1122 - 应收账款 (资产)
1123 - 预付账款 (资产)
1401 - 存货 (资产)
2202 - 应付账款 (负债)
2203 - 预收账款 (负债)

> **Warning**
>
> **重要提示:** 在会计系统中，通常不会物理删除科目，而是使用"软删除"(设置 is_active=0)。这样可以保留历史数据的完整性。

```
PRAGMA wal_checkpoint;

PRAGMA integrity_check;

PRAGMA optimize;
```

5.4 使用 Pandas 与 SQLite3 交互

学习目标: - 掌握 pandas 与 SQL 的集成使用 - 学会将 DataFrame 写入数据库 - 理解 SQL 查询与 pandas 的结合优势

Pandas 提供了便捷的方法来读取和写入数据库，这对财务数据分析非常有用。

5.4.1 创建会计分录表

让我们创建一个记账凭证表来演示:

```
import pandas as pd

# 创建记账凭证数据
journal_entries = pd.DataFrame({
    'entry_id': ['JE001', 'JE001', 'JE002', 'JE002', 'JE003', 'JE003',
    ↪  'JE003', 'JE004', 'JE004'],
    'entry_date': ['2024-01-15', '2024-01-15', '2024-01-20',
    ↪  '2024-01-20',
                   '2024-02-05', '2024-02-05', '2024-02-05',
                    ↪  '2024-02-10', '2024-02-10'],
    'account_code': ['1002', '6001', '1122', '6001', '1001', '1002',
    ↪  '6001', '2202', '6401'],
    'description': ['销售收款', '销售收入', '赊销收入', '销售收入', '现金
    ↪  销售', '银行转账', '销售收入',
                   '采购商品', '采购成本'],
    'debit': [50000, 0, 30000, 0, 20000, 0, 0, 0, 15000],
    'credit': [0, 50000, 0, 30000, 0, 20000, 40000, 25000, 0]
})

# 查看数据结构
print(" 记账凭证数据前 5 行: ")
print(journal_entries.head())

print("\n数据集信息: ")
print(journal_entries.info())
```

记账凭证数据前5行:

	entry_id	entry_date	account_code	description	debit	credit
0	JE001	2024-01-15	1002	销售收款	50000	0
1	JE001	2024-01-15	6001	销售收入	0	50000
2	JE002	2024-01-20	1122	赊销收入	30000	0
3	JE002	2024-01-20	6001	销售收入	0	30000

```
4     JE003   2024-02-05          1001         现金销售   20000        0

数据集信息:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9 entries, 0 to 8
Data columns (total 6 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   entry_id      9 non-null      object
 1   entry_date    9 non-null      object
 2   account_code  9 non-null      object
 3   description   9 non-null      object
 4   debit         9 non-null      int64
 5   credit        9 non-null      int64
dtypes: int64(2), object(4)
memory usage: 564.0+ bytes
None
```

> **Note**
>
> **双记账说明:** - 每笔会计分录必须借贷平衡 - JE001: 银行存款 (借) = 销售收入 (贷) = 50,000 - JE002: 应收账款 (借) = 销售收入 (贷) = 30,000 - JE003: 现金 (借) + 销售收入 (贷) = 银行存款 (借) + 销售收入 (贷) = 40,000

将 DataFrame 写入 SQLite3

```
# 将记账凭证数据写入数据库
journal_entries.to_sql('journal_entries', conn, if_exists='replace',
 ↪  index=False)
print(" 记账凭证数据已写入数据库")

# 验证数据
cursor.execute("SELECT COUNT(*) FROM journal_entries")
```

```
count = cursor.fetchone()[0]
print(f" 数据库中共有 {count} 条分录记录")
```

记账凭证数据已写入数据库
数据库中共有 9 条分录记录

从 SQLite3 读取数据

```
# 从数据库读取数据
df_from_db = pd.read_sql_query("SELECT * FROM journal_entries LIMIT 10",
 ↪  conn)
print(" 从数据库读取的前 10 行: ")
print(df_from_db)
```

从数据库读取的前10行:

	entry_id	entry_date	account_code	description	debit	credit
0	JE001	2024-01-15	1002	销售收款	50000	0
1	JE001	2024-01-15	6001	销售收入	0	50000
2	JE002	2024-01-20	1122	赊销收入	30000	0
3	JE002	2024-01-20	6001	销售收入	0	30000
4	JE003	2024-02-05	1001	现金销售	20000	0
5	JE003	2024-02-05	1002	银行转账	0	20000
6	JE003	2024-02-05	6001	销售收入	0	40000
7	JE004	2024-02-10	2202	采购商品	0	25000
8	JE004	2024-02-10	6401	采购成本	15000	0

SQL 查询示例

```
# 复杂查询: 按科目汇总借贷金额
query = """
SELECT
    j.account_code,
```

```
    c.account_name,
    c.account_type,
    SUM(j.debit) as total_debit,
    SUM(j.credit) as total_credit,
    SUM(j.debit) - SUM(j.credit) as net_balance
FROM journal_entries j
LEFT JOIN chart_of_accounts c ON j.account_code = c.account_code
GROUP BY j.account_code, c.account_name, c.account_type
ORDER BY j.account_code
"""
result = pd.read_sql_query(query, conn)
print(" 按科目汇总的借贷金额: ")
print(result)
```

按科目汇总的借贷金额:

	account_code	account_name	account_type	total_debit	total_credit	net_balance
0	1001	现金	资产	20000	0	20000
1	1002	银行存款	资产	50000	20000	30000
2	1122	应收账款	资产	30000	0	30000
3	2202	应付账款	负债	0	25000	-25000
4	6001	主营业务收入	收入	0	120000	-120000
5	6401	主营业务成本	费用	15000	0	15000

> **Tip**
>
> **会计应用:** 这个查询实际上生成了一个简单的**试算平衡表**的基础数据。试算平衡表是验证账目借贷平衡的重要工具。

验证借贷平衡

```
# 检查整体借贷是否平衡
query = """
SELECT
    SUM(debit) as total_debits,
    SUM(credit) as total_credits,
    SUM(debit) - SUM(credit) as difference
FROM journal_entries
"""
balance_check = pd.read_sql_query(query, conn)
print("\n借贷平衡检查: ")
print(balance_check)

if balance_check['difference'][0] == 0:
    print("  借贷平衡")
else:
    print(f"  不平衡, 差额: {balance_check['difference'][0]}")
```

借贷平衡检查:

```
   total_debits  total_credits  difference
0        115000         165000      -50000
  不平衡, 差额: -50000
```

5.4.2 高级 SQL 操作

学习目标: - 掌握 SQL 聚合函数在财务分析中的应用 - 学会使用 GROUP BY 进行分类汇总 - 理解 JOIN 操作在多表关联中的作用 - 应用窗口函数进行高级财务计算

本节将详细介绍常见的 SQL 操作，使用会计数据进行演示。

聚合函数

聚合函数对一组值进行计算，返回单个值。在会计中常用于汇总分析。

常用聚合函数: - COUNT(*): 计数 - SUM(column): 求和 - AVG(column): 平均值 - MIN(column): 最小值 - MAX(column): 最大值

```
# 聚合函数示例: 分析记账凭证
query = """
SELECT
    COUNT(DISTINCT entry_id) as total_entries,
    COUNT(*) as total_lines,
    SUM(debit) as total_debits,
    SUM(credit) as total_credits,
    AVG(debit + credit) as avg_amount_per_line,
    MAX(debit) as max_debit,
    MAX(credit) as max_credit
FROM journal_entries
"""
result = pd.read_sql_query(query, conn)
print(" 记账凭证聚合统计: ")
print(result)
```

记账凭证聚合统计:

```
   total_entries  total_lines  total_debits  total_credits  \
0              4            9        115000         165000

   avg_amount_per_line  max_debit  max_credit
0         31111.111111      50000       50000
```

> **Tip**
>
> **会计应用:** - total_debits 应该等于 total_credits(借贷平衡原则) - total_entries 表示凭证数量 - total_lines 表示分录行数

GROUP BY 和 HAVING

GROUP BY 用于将结果按指定列分组，HAVING 用于过滤分组后的结果。

```
# GROUP BY 示例：按科目类型分组统计
query = """
SELECT
    c.account_type,
    COUNT(DISTINCT j.account_code) as accounts_used,
    COUNT(*) as transaction_count,
    SUM(j.debit) as total_debits,
    SUM(j.credit) as total_credits
FROM journal_entries j
LEFT JOIN chart_of_accounts c ON j.account_code = c.account_code
GROUP BY c.account_type
ORDER BY c.account_type
"""
result = pd.read_sql_query(query, conn)
print(" 按科目类型分组的统计：")
print(result)
```

按科目类型分组的统计：

	account_type	accounts_used	transaction_count	total_debits	total_credits
0	收入	1	3	0	120000
1	负债	1	1	0	25000
2	费用	1	1	15000	0
3	资产	3	4	100000	20000

```
# HAVING 示例：找出交易金额超过 30000 的科目
query = """
SELECT
    j.account_code,
    c.account_name,
```

```
    SUM(j.debit + j.credit) as total_amount,
    COUNT(*) as transaction_count
FROM journal_entries j
LEFT JOIN chart_of_accounts c ON j.account_code = c.account_code
GROUP BY j.account_code, c.account_name
HAVING SUM(j.debit + j.credit) > 30000
ORDER BY total_amount DESC
"""
result = pd.read_sql_query(query, conn)
print("\n交易金额超过 30000 的科目: ")
print(result)
```

交易金额超过30000的科目:

	account_code	account_name	total_amount	transaction_count
0	6001	主营业务收入	120000	3
1	1002	银行存款	70000	2

> **Note**
>
> **HAVING vs WHERE**: - WHERE: 过滤行数据 (在分组之前) - HAVING: 过滤分组结果 (在分组之后)

JOIN 操作

JOIN 用于连接多个表，在会计中常用于关联科目表、凭证表、客户表等。

JOIN 类型: - **INNER JOIN**: 只返回匹配的行 (两表都有的数据) - **LEFT JOIN**: 返回左表所有行和右表匹配行 (保留主表全部数据) - **CROSS JOIN**: 笛卡尔积 (两表的所有组合)

我们已经在前面的查询中使用了 JOIN。让我们创建客户主数据表来演示更多 JOIN 应用:

```
# 创建客户主数据表
cursor.execute('''
CREATE TABLE IF NOT EXISTS customers (
```

```
    customer_id TEXT PRIMARY KEY,
    customer_name TEXT NOT NULL,
    customer_type TEXT,
    credit_limit REAL
)
''')

# 插入客户数据
customers_data = [
    ('C001', '甲公司', 'VIP', 100000),
    ('C002', '乙公司', '普通', 50000),
    ('C003', '丙公司', 'VIP', 150000),
    ('C004', '丁公司', '普通', 30000)
]

cursor.executemany("INSERT OR REPLACE INTO customers VALUES (?, ?, ?,
 ↪  ?)", customers_data)
conn.commit()
print(" 客户主数据表创建成功")

# 创建应收账款明细表
receivables_data = pd.DataFrame({
    'invoice_id': ['INV001', 'INV002', 'INV003', 'INV004', 'INV005'],
    'customer_id': ['C001', 'C001', 'C002', 'C003', 'C999'],  # C999 不存
     ↪  在，用于演示 JOIN 差异
    'invoice_date': ['2024-01-15', '2024-02-10', '2024-01-25',
     ↪  '2024-02-20', '2024-03-01'],
    'amount': [30000, 25000, 15000, 50000, 10000],
    'paid': [1, 0, 1, 0, 0]
})

receivables_data.to_sql('receivables', conn, if_exists='replace',
 ↪  index=False)
```

```
print(" 应收账款明细表创建成功")
```

客户主数据表创建成功
应收账款明细表创建成功

INNER JOIN 示例 - 查询有应收账款的客户:

```
# INNER JOIN: 只显示有应收账款的客户
query = """
SELECT
    c.customer_id,
    c.customer_name,
    c.customer_type,
    r.invoice_id,
    r.invoice_date,
    r.amount,
    CASE WHEN r.paid = 1 THEN '已付' ELSE '未付' END as payment_status
FROM customers c
INNER JOIN receivables r ON c.customer_id = r.customer_id
ORDER BY c.customer_id, r.invoice_date
"""
result = pd.read_sql_query(query, conn)
print("INNER JOIN 示例 - 有应收账款的客户: ")
print(result)
```

```
INNER JOIN 示例 - 有应收账款的客户:
  customer_id customer_name customer_type invoice_id invoice_date  amount  \
0        C001           甲公司           VIP     INV001   2024-01-15   30000
1        C001           甲公司           VIP     INV002   2024-02-10   25000
2        C002           乙公司            普通     INV003   2024-01-25   15000
3        C003           丙公司           VIP     INV004   2024-02-20   50000

  payment_status
```

```
0             已付
1             未付
2             已付
3             未付
```

LEFT JOIN 示例 - 查询所有客户的应收情况:

```
# LEFT JOIN: 显示所有客户，包括没有应收账款的
query = """
SELECT
    c.customer_id,
    c.customer_name,
    c.customer_type,
    c.credit_limit,
    COUNT(r.invoice_id) as invoice_count,
    COALESCE(SUM(r.amount), 0) as total_receivables,
    COALESCE(SUM(CASE WHEN r.paid = 0 THEN r.amount ELSE 0 END), 0) as
 ↪  outstanding_amount
FROM customers c
LEFT JOIN receivables r ON c.customer_id = r.customer_id
GROUP BY c.customer_id, c.customer_name, c.customer_type, c.credit_limit
ORDER BY total_receivables DESC
"""
result = pd.read_sql_query(query, conn)
print("\nLEFT JOIN 示例 - 所有客户应收汇总: ")
print(result)
```

```
LEFT JOIN 示例 - 所有客户应收汇总:
  customer_id customer_name customer_type  credit_limit  invoice_count  \
0        C001           甲公司           VIP      100000.0              2
1        C003           丙公司           VIP      150000.0              1
2        C002           乙公司            普通       50000.0              1
3        C004           丁公司            普通       30000.0              0
```

```
   total_receivables  outstanding_amount
0              55000               25000
1              50000               50000
2              15000                   0
3                  0                   0
```

> **Tip**
>
> **会计应用:** - **INNER JOIN**: 用于查询有交易的客户/供应商 - **LEFT JOIN**: 用于生成完整的客户列表 (包括无交易的) - **COALESCE**: 处理 NULL 值, 在汇总时很有用

查找数据质量问题:

```
# 找出应收账款表中不存在于客户主数据的记录
query = """
SELECT
    r.invoice_id,
    r.customer_id,
    r.amount,
    '客户不存在' as issue
FROM receivables r
LEFT JOIN customers c ON r.customer_id = c.customer_id
WHERE c.customer_id IS NULL
"""
result = pd.read_sql_query(query, conn)
print("\n数据质量检查 - 客户主数据缺失: ")
print(result)
```

```
数据质量检查 - 客户主数据缺失:
  invoice_id customer_id  amount  issue
0     INV005        C999   10000  客户不存在
```

子查询

子查询是嵌套在另一个查询中的查询，在会计分析中常用于比较分析和数据筛选。

> **Note**
>
> **学习目标:** - 掌握标量子查询 (返回单个值) - 掌握 IN 子查询 (返回值列表) - 理解子查询在会计分析中的应用

```
# 子查询示例: 查找金额高于平均值的交易
query = """
SELECT
    entry_id,
    entry_date,
    account_code,
    description,
    debit,
    credit,
    (debit + credit) as amount
FROM journal_entries
WHERE (debit + credit) > (
    SELECT AVG(debit + credit)
    FROM journal_entries
    WHERE debit + credit > 0
)
ORDER BY (debit + credit) DESC
"""
result = pd.read_sql_query(query, conn)
print(" 金额高于平均值的记账分录: ")
print(result)

# 子查询用于 IN 操作: 找出有大额交易的科目
query = """
SELECT
```

```
    c.account_code,
    c.account_name,
    c.account_type
FROM chart_of_accounts c
WHERE c.account_code IN (
    SELECT account_code
    FROM journal_entries
    GROUP BY account_code
    HAVING MAX(debit + credit) > 40000
)
ORDER BY c.account_type, c.account_code
"""
result = pd.read_sql_query(query, conn)
print("\n存在大额交易 (>40000) 的会计科目: ")
print(result)

# EXISTS 子查询: 找出有未付款发票的客户
query = """
SELECT
    c.customer_id,
    c.customer_name,
    c.customer_type,
    c.credit_limit
FROM customers c
WHERE EXISTS (
    SELECT 1
    FROM receivables r
    WHERE r.customer_id = c.customer_id
    AND r.paid = 0
)
"""
result = pd.read_sql_query(query, conn)
```

```
print("\n存在未付款发票的客户: ")
print(result)
```

金额高于平均值的记账分录:

```
  entry_id  entry_date account_code description  debit  credit  amount
0    JE001  2024-01-15         1002        销售收款  50000       0   50000
1    JE001  2024-01-15         6001        销售收入      0   50000   50000
2    JE003  2024-02-05         6001        销售收入      0   40000   40000
```

存在大额交易(>40000)的会计科目:

```
  account_code account_name account_type
0         6001       主营业务收入           收入
1         1002         银行存款           资产
```

存在未付款发票的客户:

```
  customer_id customer_name customer_type  credit_limit
0        C001           甲公司           VIP      100000.0
1        C003           丙公司           VIP      150000.0
```

窗口函数

窗口函数对结果集的子集（窗口）进行计算, 在会计分析中用于排名、趋势分析和累计计算。

> **Note**
>
> **学习目标:** - 掌握 ROW_NUMBER()、RANK() 等排名函数 - 掌握 LAG()、LEAD() 等偏移函数用于环比分析 - 掌握 PARTITION BY 进行分组窗口计算 - 理解窗口函数在财务分析中的应用

SQLite3 支持以下窗口函数:

- `ROW_NUMBER()`: 行号
- `RANK()`: 排名（跳跃）
- `DENSE_RANK()`: 密集排名

- LAG(): 前一行值
- LEAD(): 后一行值
- SUM() OVER(): 窗口内求和

```
# 创建月度收入数据用于演示
monthly_revenue = pd.DataFrame({
    'month': ['2024-01', '2024-02', '2024-03', '2024-04', '2024-05',
    ↪  '2024-06',
             '2024-07', '2024-08', '2024-09', '2024-10', '2024-11',
             ↪  '2024-12'],
    'revenue': [850000, 920000, 880000, 950000, 990000, 1050000,
              980000, 1020000, 1100000, 1080000, 1150000, 1200000],
    'cost': [600000, 650000, 620000, 670000, 700000, 740000,
            690000, 720000, 770000, 760000, 810000, 850000]
})
monthly_revenue.to_sql('monthly_revenue', conn, if_exists='replace',
↪  index=False)
print(" 月度收入成本数据已创建")

# ROW_NUMBER 和 RANK 示例: 客户销售额排名
query = """
SELECT
    customer_id,
    customer_name,
    SUM(amount) as total_sales,
    ROW_NUMBER() OVER (ORDER BY SUM(amount) DESC) as row_num,
    RANK() OVER (ORDER BY SUM(amount) DESC) as rank,
    DENSE_RANK() OVER (ORDER BY SUM(amount) DESC) as dense_rank
FROM (
    SELECT
        c.customer_id,
        c.customer_name,
        r.amount
```

```
    FROM customers c
    JOIN receivables r ON c.customer_id = r.customer_id
)
GROUP BY customer_id, customer_name
ORDER BY total_sales DESC
"""
result = pd.read_sql_query(query, conn)
print("\n窗口函数: 客户销售额排名: ")
print(result)

# LAG 和 LEAD 示例: 环比分析
query = """
SELECT
    month,
    revenue,
    LAG(revenue) OVER (ORDER BY month) as prev_month_revenue,
    LEAD(revenue) OVER (ORDER BY month) as next_month_revenue,
    revenue - LAG(revenue) OVER (ORDER BY month) as mom_change,
    ROUND(100.0 * (revenue - LAG(revenue) OVER (ORDER BY month)) /
          LAG(revenue) OVER (ORDER BY month), 2) as mom_growth_rate
FROM monthly_revenue
ORDER BY month
"""
result = pd.read_sql_query(query, conn)
print("\nLAG 和 LEAD 函数: 月度环比分析: ")
print(result)

# 累计求和: 计算年度累计收入
query = """
SELECT
    month,
    revenue,
```

```
    SUM(revenue) OVER (ORDER BY month ROWS BETWEEN UNBOUNDED PRECEDING
 ↪  AND CURRENT ROW) as ytd_revenue,
    ROUND(100.0 * revenue / SUM(revenue) OVER (ORDER BY month ROWS
 ↪  BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW), 2) as pct_of_ytd
FROM monthly_revenue
ORDER BY month
"""
result = pd.read_sql_query(query, conn)
print("\n累计求和: 年度累计收入 (YTD): ")
print(result)

# 分组窗口函数: 按科目类型分析
query = """
SELECT
    c.account_type,
    c.account_code,
    c.account_name,
    SUM(j.debit) as total_debit,
    SUM(j.credit) as total_credit,
    AVG(SUM(j.debit + j.credit)) OVER (PARTITION BY c.account_type) as
 ↪  type_avg_amount,
    SUM(SUM(j.debit + j.credit)) OVER (PARTITION BY c.account_type) as
 ↪  type_total_amount
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
GROUP BY c.account_type, c.account_code, c.account_name
ORDER BY c.account_type, total_debit + total_credit DESC
"""
result = pd.read_sql_query(query, conn)
print("\n分组窗口函数: 按科目类型的交易分析: ")
print(result)
```

```
# 移动平均: 3 个月移动平均
query = """
SELECT
    month,
    revenue,
    ROUND(AVG(revenue) OVER (
        ORDER BY month
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
    ), 2) as ma_3month
FROM monthly_revenue
ORDER BY month
"""
result = pd.read_sql_query(query, conn)
print("\n移动平均: 3 个月收入移动平均: ")
print(result)
```

```
月度收入成本数据已创建

窗口函数: 客户销售额排名:
  customer_id customer_name  total_sales  row_num  rank  dense_rank
0        C001           甲公司        55000        1     1           1
1        C003           丙公司        50000        2     2           2
2        C002           乙公司        15000        3     3           3

LAG 和 LEAD 函数: 月度环比分析:
      month  revenue  prev_month_revenue  next_month_revenue  mom_change  \
0   2024-01   850000                 NaN            920000.0         NaN
1   2024-02   920000            850000.0            880000.0     70000.0
2   2024-03   880000            920000.0            950000.0    -40000.0
3   2024-04   950000            880000.0            990000.0     70000.0
4   2024-05   990000            950000.0           1050000.0     40000.0
5   2024-06  1050000            990000.0            980000.0     60000.0
```

6	2024-07	980000	1050000.0	1020000.0	-70000.0
7	2024-08	1020000	980000.0	1100000.0	40000.0
8	2024-09	1100000	1020000.0	1080000.0	80000.0
9	2024-10	1080000	1100000.0	1150000.0	-20000.0
10	2024-11	1150000	1080000.0	1200000.0	70000.0
11	2024-12	1200000	1150000.0	NaN	50000.0

	mom_growth_rate
0	NaN
1	8.24
2	-4.35
3	7.95
4	4.21
5	6.06
6	-6.67
7	4.08
8	7.84
9	-1.82
10	6.48
11	4.35

累计求和: 年度累计收入(YTD):

	month	revenue	ytd_revenue	pct_of_ytd
0	2024-01	850000	850000	100.00
1	2024-02	920000	1770000	51.98
2	2024-03	880000	2650000	33.21
3	2024-04	950000	3600000	26.39
4	2024-05	990000	4590000	21.57
5	2024-06	1050000	5640000	18.62
6	2024-07	980000	6620000	14.80
7	2024-08	1020000	7640000	13.35
8	2024-09	1100000	8740000	12.59

9	2024-10	1080000	9820000	11.00
10	2024-11	1150000	10970000	10.48
11	2024-12	1200000	12170000	9.86

分组窗口函数：按科目类型的交易分析：

	account_type	account_code	account_name	total_debit	total_credit	\
0	收入	6001	主营业务收入	0	120000	
1	负债	2202	应付账款	0	25000	
2	费用	6401	主营业务成本	15000	0	
3	资产	1002	银行存款	50000	20000	
4	资产	1122	应收账款	30000	0	
5	资产	1001	现金	20000	0	

	type_avg_amount	type_total_amount
0	120000.0	120000
1	25000.0	25000
2	15000.0	15000
3	40000.0	120000
4	40000.0	120000
5	40000.0	120000

移动平均：3个月收入移动平均：

	month	revenue	ma_3month
0	2024-01	850000	850000.00
1	2024-02	920000	885000.00
2	2024-03	880000	883333.33
3	2024-04	950000	916666.67
4	2024-05	990000	940000.00
5	2024-06	1050000	996666.67
6	2024-07	980000	1006666.67
7	2024-08	1020000	1016666.67
8	2024-09	1100000	1033333.33

```
9   2024-10   1080000   1066666.67
10  2024-11   1150000   1110000.00
11  2024-12   1200000   1143333.33
```

> **Tip**
>
> **会计应用:** 窗口函数在财务分析中的常见应用：- **环比分析**：使用 LAG/LEAD 计算月度、季度环比增长 - **累计计算**：YTD 收入、累计利润等 - **移动平均**：平滑波动，识别趋势 - **排名分析**：客户/产品/部门业绩排名 - **占比分析**：各项目占总额的比例

CTE (Common Table Expressions)

CTE 使用 `WITH` 关键字定义临时结果集，使复杂查询更清晰、可读。在会计报表编制中特别有用。

> **Note**
>
> **学习目标:** - 掌握 WITH 语句创建 CTE - 掌握多个 CTE 的组合使用 - 理解 CTE 在财务报表编制中的应用

```
# CTE 示例：分析各科目类型的交易统计
query = """
WITH account_type_stats AS (
    SELECT
        c.account_type,
        COUNT(DISTINCT j.entry_id) as entry_count,
        COUNT(*) as line_count,
        SUM(j.debit) as total_debit,
        SUM(j.credit) as total_credit,
        AVG(j.debit + j.credit) as avg_amount
    FROM chart_of_accounts c
    JOIN journal_entries j ON c.account_code = j.account_code
    GROUP BY c.account_type
),
ranked_types AS (
```

```
    SELECT
        *,
        RANK() OVER (ORDER BY total_debit + total_credit DESC) as
↪   activity_rank,
        ROUND(100.0 * (total_debit + total_credit) /
              SUM(total_debit + total_credit) OVER (), 2) as pct_of_total
    FROM account_type_stats
)
SELECT * FROM ranked_types
ORDER BY activity_rank
"""
result = pd.read_sql_query(query, conn)
print("CTE 示例: 科目类型交易统计和排名: ")
print(result)

# 递归 CTE 示例: 利润表层级结构 (简化版)

query = """
WITH RECURSIVE income_statement AS (
    -- 收入部分
    SELECT 1 as line_order, '营业收入' as line_item, SUM(credit) as
↪   amount
    FROM journal_entries
    WHERE account_code LIKE '6001%'

    UNION ALL

    -- 成本部分
    SELECT 2, '营业成本', SUM(debit)
    FROM journal_entries
    WHERE account_code LIKE '6401%'
```

```
    UNION ALL

    -- 毛利
    SELECT 3, '毛利', (
        SELECT SUM(credit) FROM journal_entries WHERE account_code LIKE
↪   '6001%'
    ) - (
        SELECT SUM(debit) FROM journal_entries WHERE account_code LIKE
↪   '6401%'
    )
)
SELECT line_order, line_item, ROUND(amount, 2) as amount
FROM income_statement
ORDER BY line_order
"""
result = pd.read_sql_query(query, conn)
print("\n递归 CTE: 简化利润表: ")
print(result)

# 多个 CTE 组合: 客户价值分析
query = """
WITH customer_sales AS (
    SELECT
        c.customer_id,
        c.customer_name,
        c.customer_type,
        COUNT(r.invoice_id) as invoice_count,
        SUM(r.amount) as total_sales,
        AVG(r.amount) as avg_invoice_amount
    FROM customers c
    LEFT JOIN receivables r ON c.customer_id = r.customer_id
    GROUP BY c.customer_id, c.customer_name, c.customer_type
```

```
),
customer_payments AS (
    SELECT
        customer_id,
        SUM(CASE WHEN paid = 1 THEN amount ELSE 0 END) as paid_amount,
        SUM(CASE WHEN paid = 0 THEN amount ELSE 0 END) as unpaid_amount,
        ROUND(100.0 * SUM(CASE WHEN paid = 1 THEN 1 ELSE 0 END) /
 ↪  COUNT(*), 2) as payment_rate
    FROM receivables
    GROUP BY customer_id
),
customer_classification AS (
    SELECT
        cs.*,
        cp.paid_amount,
        cp.unpaid_amount,
        cp.payment_rate,
        CASE
            WHEN cs.total_sales > 60000 THEN '大客户'
            WHEN cs.total_sales > 30000 THEN '中客户'
            ELSE '小客户'
        END as customer_class,
        CASE
            WHEN cp.payment_rate >= 80 THEN '优质'
            WHEN cp.payment_rate >= 50 THEN '良好'
            ELSE '需关注'
        END as credit_rating
    FROM customer_sales cs
    LEFT JOIN customer_payments cp ON cs.customer_id = cp.customer_id
)
SELECT * FROM customer_classification
ORDER BY total_sales DESC
```

```
"""
result = pd.read_sql_query(query, conn)
print("\n多 CTE 组合: 客户价值与信用评级分析: ")
print(result)
```

```
CTE 示例: 科目类型交易统计和排名:
  account_type  entry_count  line_count  total_debit  total_credit  \
0           收入            3           3            0        120000
1           资产            3           4       100000         20000
2           负债            1           1            0         25000
3           费用            1           1        15000             0

   avg_amount  activity_rank  pct_of_total
0     40000.0              1         42.86
1     30000.0              1         42.86
2     25000.0              3          8.93
3     15000.0              4          5.36

递归CTE: 简化利润表:
   line_order line_item    amount
0           1      营业收入  120000.0
1           2      营业成本   15000.0
2           3        毛利  105000.0

多CTE组合: 客户价值与信用评级分析:
  customer_id customer_name customer_type  invoice_count  total_sales  \
0        C001           甲公司           VIP              2      55000.0
1        C003           丙公司           VIP              1      50000.0
2        C002           乙公司            普通              1      15000.0
3        C004           丁公司            普通              0          NaN

   avg_invoice_amount  paid_amount  unpaid_amount  payment_rate  \
```

```
0              27500.0       30000.0        25000.0          50.0
1              50000.0           0.0        50000.0           0.0
2              15000.0       15000.0            0.0         100.0
3                  NaN           NaN            NaN           NaN

  customer_class credit_rating
0            中客户            良好
1            中客户          需关注
2            小客户            优质
3            小客户          需关注
```

> **Tip**
>
> **会计应用**: CTE 在财务报表中的应用：- **分步计算**：将复杂的报表计算分解为清晰的步骤 - **递归查询**：处理科目体系的层级结构 - **中间结果**：保存计算结果供后续使用 - **提高可读性**：使复杂的 SQL 逻辑更易理解和维护

日期和时间处理

SQLite 支持多种日期和时间函数，可以处理日期的存储、格式化和计算。

创建包含日期的表

```
# 创建包含日期字段的表
cursor.execute('''
CREATE TABLE IF NOT EXISTS sales (
    id INTEGER PRIMARY KEY,
    product_name TEXT,
    sale_date TEXT,  -- SQLite 中日期通常存储为 TEXT
    quantity INTEGER,
    price REAL
)
''')

# 插入包含日期的数据
```

```
sales_data = [
    ('钻石戒指', '2024-01-15', 2, 1500.00),
    ('项链', '2024-02-20', 1, 800.00),
    ('耳环', '2024-03-10', 3, 450.00),
    ('手镯', '2024-01-25', 1, 1200.00),
    ('钻石项链', '2024-02-28', 1, 2500.00)
]

cursor.executemany("INSERT OR REPLACE INTO sales (product_name,
↪  sale_date, quantity, price) VALUES (?, ?, ?, ?)", sales_data)
conn.commit()
print(" 销售数据表创建并插入数据成功")
```

```
销售数据表创建并插入数据成功
```

SQLite 日期函数

```
# 使用日期函数查询
query = """
SELECT
    product_name,
    sale_date,
    quantity,
    price,
    -- 提取年月日
    strftime('%Y', sale_date) as year,
    strftime('%m', sale_date) as month,
    strftime('%d', sale_date) as day,
    -- 计算总价
    quantity * price as total_amount
FROM sales
ORDER BY sale_date
```

```
"""
result = pd.read_sql_query(query, conn)
print(" 销售数据及日期处理: ")
print(result)

# 按月份统计销售额
query = """
SELECT
    strftime('%Y-%m', sale_date) as month,
    COUNT(*) as sales_count,
    SUM(quantity * price) as total_revenue,
    AVG(quantity * price) as avg_sale_amount
FROM sales
GROUP BY strftime('%Y-%m', sale_date)
ORDER BY month
"""
result = pd.read_sql_query(query, conn)
print("\n按月份统计销售额: ")
print(result)
```

销售数据及日期处理:

	product_name	sale_date	quantity	price	year	month	day	total_amount
0	钻石戒指	2024-01-15	2	1500.0	2024	01	15	3000.0
1	钻石戒指	2024-01-15	2	1500.0	2024	01	15	3000.0
2	钻石戒指	2024-01-15	2	1500.0	2024	01	15	3000.0
3	钻石戒指	2024-01-15	2	1500.0	2024	01	15	3000.0
4	钻石戒指	2024-01-15	2	1500.0	2024	01	15	3000.0
..	...	...	...	...	...	...	..	...
60	限时优惠品	2025-10-19	1	999.0	2025	10	19	999.0
61	限时优惠品	2025-10-19	1	999.0	2025	10	19	999.0
62	限时优惠品	2025-10-19	1	999.0	2025	10	19	999.0
63	限时优惠品	2025-10-19	1	999.0	2025	10	19	999.0

```
64         限时优惠品  2025-10-19         1    999.0  2025     10  19          999.0

[65 rows x 8 columns]
```

按月份统计销售额:

	month	sales_count	total_revenue	avg_sale_amount
0	2024-01	22	46200.0	2100.0
1	2024-02	22	36300.0	1650.0
2	2024-03	11	14850.0	1350.0
3	2025-10	10	9990.0	999.0

日期计算和过滤

```
# 查询最近 30 天的销售
query = """
SELECT
    product_name,
    sale_date,
    quantity * price as amount
FROM sales
WHERE date(sale_date) >= date('now', '-30 days')
ORDER BY sale_date DESC
"""
result = pd.read_sql_query(query, conn)
print(" 最近 30 天的销售记录: ")
print(result)

# 计算两个日期之间的天数差
query = """
SELECT
    product_name,
    sale_date,
    -- 计算距今天的天数
```

```
    julianday('now') - julianday(sale_date) as days_since_sale,
    -- 计算距 2024 年 1 月 1 日的天数
    julianday(sale_date) - julianday('2024-01-01') as days_from_start
FROM sales
ORDER BY sale_date
"""
result = pd.read_sql_query(query, conn)
print("\n日期计算示例: ")
print(result)
```

最近30天的销售记录:

	product_name	sale_date	amount
0	限时优惠品	2025-10-19	999.0
1	限时优惠品	2025-10-19	999.0
2	限时优惠品	2025-10-19	999.0
3	限时优惠品	2025-10-19	999.0
4	限时优惠品	2025-10-19	999.0
5	限时优惠品	2025-10-19	999.0
6	限时优惠品	2025-10-19	999.0
7	限时优惠品	2025-10-19	999.0
8	限时优惠品	2025-10-19	999.0
9	限时优惠品	2025-10-19	999.0

日期计算示例:

	product_name	sale_date	days_since_sale	days_from_start
0	钻石戒指	2024-01-15	643.330782	14.0
1	钻石戒指	2024-01-15	643.330782	14.0
2	钻石戒指	2024-01-15	643.330782	14.0
3	钻石戒指	2024-01-15	643.330782	14.0
4	钻石戒指	2024-01-15	643.330782	14.0
..	...	...	...	...
60	限时优惠品	2025-10-19	0.330782	657.0

```
61          限时优惠品  2025-10-19          0.330782            657.0
62          限时优惠品  2025-10-19          0.330782            657.0
63          限时优惠品  2025-10-19          0.330782            657.0
64          限时优惠品  2025-10-19          0.330782            657.0

[65 rows x 4 columns]
```

在 Python 中处理日期

```
import datetime

# 使用 Python 的 datetime 处理日期
current_date = datetime.datetime.now()
print(f" 当前日期时间: {current_date}")

# 格式化日期
formatted_date = current_date.strftime("%Y-%m-%d %H:%M:%S")
print(f" 格式化后的日期: {formatted_date}")

# 插入当前日期的数据
cursor.execute("""
INSERT INTO sales (product_name, sale_date, quantity, price)
VALUES (?, ?, ?, ?)
""", ('限时优惠品', current_date.strftime("%Y-%m-%d"), 1, 999.00))
conn.commit()
print(" 插入了当前日期的销售记录")

# 查询今天的销售
today = datetime.date.today().strftime("%Y-%m-%d")
query = f"""
SELECT * FROM sales
WHERE sale_date = '{today}'
"""
```

```
result = pd.read_sql_query(query, conn)
print(f"\n今天的销售记录 ({today}):")
print(result)
```

```
当前日期时间: 2025-10-19 15:56:19.526581
格式化后的日期: 2025-10-19 15:56:19
插入了当前日期的销售记录

今天的销售记录 (2025-10-19):
    id product_name   sale_date  quantity  price
0    6        限时优惠品  2025-10-19         1  999.0
1   12        限时优惠品  2025-10-19         1  999.0
2   18        限时优惠品  2025-10-19         1  999.0
3   24        限时优惠品  2025-10-19         1  999.0
4   30        限时优惠品  2025-10-19         1  999.0
5   36        限时优惠品  2025-10-19         1  999.0
6   42        限时优惠品  2025-10-19         1  999.0
7   48        限时优惠品  2025-10-19         1  999.0
8   54        限时优惠品  2025-10-19         1  999.0
9   60        限时优惠品  2025-10-19         1  999.0
10  66        限时优惠品  2025-10-19         1  999.0
```

Pandas 中的日期处理

```
# 将字符串日期转换为 datetime
df_sales = pd.read_sql_query("SELECT * FROM sales", conn)
print(" 原始销售数据: ")
print(df_sales)

# 转换日期列
df_sales['sale_date'] = pd.to_datetime(df_sales['sale_date'])
print("\n转换日期类型后的数据: ")
```

```
print(df_sales.dtypes)

# 日期相关的操作
print("\n日期统计: ")
print(f" 最早销售日期: {df_sales['sale_date'].min()}")
print(f" 最晚销售日期: {df_sales['sale_date'].max()}")
print(f" 销售天数: {(df_sales['sale_date'].max() -
 ↪  df_sales['sale_date'].min()).days} 天")

# 按月份分组统计
df_sales['month'] = df_sales['sale_date'].dt.to_period('M')
monthly_sales = df_sales.groupby('month').agg({
    'quantity': 'sum',
    'price': 'sum',
    'id': 'count'
}).rename(columns={'id': 'sales_count'})
print("\n月度销售统计: ")
print(monthly_sales)
```

原始销售数据:

```
    id product_name   sale_date  quantity   price
0    1         钻石戒指  2024-01-15         2  1500.0
1    2           项链  2024-02-20         1   800.0
2    3           耳环  2024-03-10         3   450.0
3    4           手镯  2024-01-25         1  1200.0
4    5         钻石项链  2024-02-28         1  2500.0
..  ..          ...         ...       ...     ...
61  62           项链  2024-02-20         1   800.0
62  63           耳环  2024-03-10         3   450.0
63  64           手镯  2024-01-25         1  1200.0
64  65         钻石项链  2024-02-28         1  2500.0
65  66        限时优惠品  2025-10-19         1   999.0
```

```
[66 rows x 5 columns]

转换日期类型后的数据:
id                          int64
product_name               object
sale_date       datetime64[ns]
quantity                    int64
price                     float64
dtype: object

日期统计:
最早销售日期: 2024-01-15 00:00:00
最晚销售日期: 2025-10-19 00:00:00
销售天数: 643 天

月度销售统计:
         quantity    price  sales_count
month
2024-01        33  29700.0           22
2024-02        22  36300.0           22
2024-03        33   4950.0           11
2025-10        11  10989.0           11
```

其他常见 SQL 操作

5.4.3 DISTINCT - 去重查询

DISTINCT 用于返回唯一不重复的值, 在会计中用于统计科目类型、客户类别等。

```
# DISTINCT 示例: 查看所有科目类型
query = """
SELECT DISTINCT account_type
FROM chart_of_accounts
```

```
WHERE is_active = 1
ORDER BY account_type
"""
result = pd.read_sql_query(query, conn)
print("DISTINCT 示例: 活跃的科目类型: ")
print(result)

# 组合 DISTINCT: 科目类型和余额方向组合
query = """
SELECT DISTINCT
    account_type,
    balance_direction
FROM chart_of_accounts
ORDER BY account_type, balance_direction
"""
result = pd.read_sql_query(query, conn)
print("\n科目类型与余额方向的唯一组合: ")
print(result)

# COUNT DISTINCT: 统计唯一值数量
query = """
SELECT
    COUNT(DISTINCT customer_id) as unique_customers,
    COUNT(DISTINCT account_code) as unique_accounts,
    COUNT(DISTINCT entry_id) as unique_entries
FROM journal_entries j
LEFT JOIN receivables r ON DATE(j.entry_date) = DATE(r.invoice_date)
"""
result = pd.read_sql_query(query, conn)
print("\n唯一值计数统计: ")
print(result)
```

DISTINCT 示例：活跃的科目类型：

```
  account_type
0        所有者权益
1           收入
2           负债
3           费用
4           资产
```

科目类型与余额方向的唯一组合：

```
  account_type balance_direction
0        所有者权益                 贷
1           收入                 贷
2           负债                 贷
3           费用                 借
4           资产                 借
```

唯一值计数统计：

```
   unique_customers  unique_accounts  unique_entries
0                 1                6               4
```

5.4.4 CASE 语句 - 条件逻辑

CASE 语句用于条件判断和分类，在会计中用于账龄分析、科目分类、风险评级等。

```
# CASE 语句示例：应收账款账龄分析
query = """
SELECT
    invoice_id,
    customer_id,
    invoice_date,
    amount,
    paid,
    CASE
```

```
        WHEN paid = 1 THEN '已付款'
        WHEN julianday('now') - julianday(invoice_date) <= 30 THEN '30 天
↪  内'
        WHEN julianday('now') - julianday(invoice_date) <= 60 THEN '31-60
↪  天'
        WHEN julianday('now') - julianday(invoice_date) <= 90 THEN '61-90
↪  天'
        ELSE '90 天以上'
    END as aging_category,
    CASE
        WHEN paid = 0 AND julianday('now') - julianday(invoice_date) > 90
↪  THEN '高风险'
        WHEN paid = 0 AND julianday('now') - julianday(invoice_date) > 60
↪  THEN '中风险'
        WHEN paid = 0 THEN '低风险'
        ELSE '无风险'
    END as risk_level
FROM receivables
ORDER BY invoice_date
"""
result = pd.read_sql_query(query, conn)
print("CASE 语句示例: 应收账款账龄与风险分析: ")
print(result)

# 嵌套 CASE: 交易金额分级
query = """
SELECT
    entry_id,
    account_code,
    debit,
    credit,
    CASE
```

```
        WHEN debit > 0 THEN
            CASE
                WHEN debit >= 50000 THEN '大额借方'
                WHEN debit >= 20000 THEN '中额借方'
                ELSE '小额借方'
            END
        WHEN credit > 0 THEN
            CASE
                WHEN credit >= 50000 THEN '大额贷方'
                WHEN credit >= 20000 THEN '中额贷方'
                ELSE '小额贷方'
            END
        ELSE '无金额'
    END as transaction_class
FROM journal_entries
ORDER BY (debit + credit) DESC
"""
result = pd.read_sql_query(query, conn)
print("\n嵌套 CASE: 交易金额分类: ")
print(result)
```

CASE 语句示例: 应收账款账龄与风险分析:

	invoice_id	customer_id	invoice_date	amount	paid	aging_category	risk_level
0	INV001	C001	2024-01-15	30000	1	已付款	无风险
1	INV003	C002	2024-01-25	15000	1	已付款	无风险
2	INV002	C001	2024-02-10	25000	0	90天以上	高风险
3	INV004	C003	2024-02-20	50000	0	90天以上	高风险
4	INV005	C999	2024-03-01	10000	0	90天以上	高风险

嵌套CASE: 交易金额分类:

	entry_id	account_code	debit	credit	transaction_class
0	JE001	1002	50000	0	大额借方

1	JE001	6001	0	50000	大额贷方
2	JE003	6001	0	40000	中额贷方
3	JE002	1122	30000	0	中额借方
4	JE002	6001	0	30000	中额贷方
5	JE004	2202	0	25000	中额贷方
6	JE003	1001	20000	0	中额借方
7	JE003	1002	0	20000	中额贷方
8	JE004	6401	15000	0	小额借方

Tip

会计应用: CASE 语句的典型应用场景：- **账龄分析**：根据日期计算账龄区间 - **科目分类**：将明细科目归类到报表项目 - **异常标识**：标记异常交易或超限额交易 - **KPI 评级**：根据财务指标进行等级评定

5.4.5 UNION - 合并查询结果

UNION 用于合并多个查询结果，常用于编制合并报表或组合不同来源的数据。

```
# UNION 示例：合并收入和支出科目
query = """
SELECT
    '收入' as category,
    c.account_code,
    c.account_name,
    SUM(j.credit) as amount
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
WHERE c.account_type = '收入'
GROUP BY c.account_code, c.account_name

UNION ALL

SELECT
```

```
    '成本费用' as category,
    c.account_code,
    c.account_name,
    SUM(j.debit) as amount
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
WHERE c.account_type IN ('成本', '费用')
GROUP BY c.account_code, c.account_name

ORDER BY category, amount DESC
"""
result = pd.read_sql_query(query, conn)
print("UNION 示例: 收入与成本费用汇总: ")
print(result)

# UNION vs UNION ALL: 资产负债表简化示例
query = """
SELECT 1 as sort_order, '资产' as section, SUM(debit - credit) as balance
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
WHERE c.account_type = '资产'

UNION ALL

SELECT 2, '负债', SUM(credit - debit)
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
WHERE c.account_type = '负债'

UNION ALL

SELECT 3, '所有者权益', SUM(credit - debit)
```

```
FROM chart_of_accounts c
JOIN journal_entries j ON c.account_code = j.account_code
WHERE c.account_type = '所有者权益'

ORDER BY sort_order
"""
result = pd.read_sql_query(query, conn)
print("\nUNION ALL: 资产负债表简化结构: ")
print(result)
```

```
UNION 示例: 收入与成本费用汇总:
  category account_code account_name  amount
0     成本费用         6401       主营业务成本   15000
1       收入         6001       主营业务收入  120000

UNION ALL: 资产负债表简化结构:
   sort_order section  balance
0           1      资产  80000.0
1           2      负债  25000.0
2           3   所有者权益      NaN
```

> **Note**
>
> **UNION vs UNION ALL**: - `UNION`: 自动去除重复行, 性能较慢 - `UNION ALL`: 保留所有行包括重复, 性能较快 - 在财务报表中通常使用 `UNION ALL`, 因为各部分数据不会重复

5.4.6 处理大数据: 使用 chunksize

当数据量很大时（如数百万条交易记录），可以使用 `chunksize` 参数分块读取数据, 避免内存溢出。

```
# 分块读取大型凭证数据
query = "SELECT * FROM journal_entries"
chunks = pd.read_sql_query(query, conn, chunksize=3)

print(" 分块读取凭证数据: ")
total_rows = 0
total_debit = 0
total_credit = 0

for i, chunk in enumerate(chunks):
    chunk_debit = chunk['debit'].sum()
    chunk_credit = chunk['credit'].sum()
    total_debit += chunk_debit
    total_credit += chunk_credit
    total_rows += len(chunk)
    print(f" 块 {i+1}: {len(chunk)} 行, 借方合计: {chunk_debit}, 贷方合
    ↪ 计: {chunk_credit}")

print(f"\n总行数: {total_rows}")
print(f" 总借方: {total_debit}, 总贷方: {total_credit}")
print(f" 借贷平衡检查: {'平衡' if abs(total_debit - total_credit) < 0.01
↪ else '不平衡'}")
```

```
分块读取凭证数据:
块 1: 3 行, 借方合计: 80000, 贷方合计: 50000
块 2: 3 行, 借方合计: 20000, 贷方合计: 50000
块 3: 3 行, 借方合计: 15000, 贷方合计: 65000

总行数: 9
总借方: 115000, 总贷方: 165000
借贷平衡检查: 不平衡
```

> **Tip**
>
> **大数据处理建议:** - 对于超过 100 万条记录的数据, 建议使用 chunksize 分块处理 - 每块大小可设置为 10000-50000 行, 根据内存情况调整 - 可在每块处理后进行聚合, 避免一次性加载全部数据

5.5 综合案例：企业财务分析仪表盘

本案例综合运用前面学习的各种 SQL 技术, 构建一个完整的财务分析仪表盘。

> **Note**
>
> **案例背景:** 某制造企业需要每月生成财务分析报告, 包括：1. 收入成本分析 2. 客户价值排名 3. 应收账款健康度 4. 月度趋势分析 5. 财务健康度评分

5.5.1 步骤 1：综合数据准备

```
# 创建更完整的测试数据集
print("=== 综合案例: 财务分析仪表盘 ===\n")

# 添加更多月度数据（如果尚未添加）
additional_months = pd.DataFrame({
    'month': ['2024-01', '2024-02', '2024-03'],
    'revenue': [850000, 920000, 880000],
    'cost': [600000, 650000, 620000]
})

# 检查月度收入表是否存在足够数据
existing_count = pd.read_sql_query("SELECT COUNT(*) as cnt FROM
 ↪  monthly_revenue", conn).iloc[0]['cnt']
if existing_count < 3:
    additional_months.to_sql('monthly_revenue', conn,
 ↪  if_exists='replace', index=False)
```

```
print(" 财务数据准备完成")
```

```
=== 综合案例: 财务分析仪表盘 ===
```

```
财务数据准备完成
```

5.5.2 步骤 2: 收入成本分析

```
# 分析收入成本结构和趋势
query = """
WITH monthly_analysis AS (
    SELECT
        month,
        revenue,
        cost,
        revenue - cost as gross_profit,
        ROUND(100.0 * (revenue - cost) / revenue, 2) as gross_margin,
        LAG(revenue) OVER (ORDER BY month) as prev_revenue,
        ROUND(100.0 * (revenue - LAG(revenue) OVER (ORDER BY month)) /
              LAG(revenue) OVER (ORDER BY month), 2) as
↪   revenue_growth_rate
    FROM monthly_revenue
)
SELECT
    month,
    revenue,
    cost,
    gross_profit,
    gross_margin,
    COALESCE(revenue_growth_rate, 0) as revenue_growth_rate,
    CASE
        WHEN gross_margin >= 35 THEN '优秀'
```

```
        WHEN gross_margin >= 25 THEN '良好'
        WHEN gross_margin >= 15 THEN '一般'
        ELSE '需改进'
    END as profitability_rating
FROM monthly_analysis
ORDER BY month
"""
revenue_analysis = pd.read_sql_query(query, conn)
print("1. 收入成本分析: ")
print(revenue_analysis)
```

1. 收入成本分析:

	month	revenue	cost	gross_profit	gross_margin	revenue_growth_rate	\
0	2024-01	850000	600000	250000	29.41	0.00	
1	2024-02	920000	650000	270000	29.35	8.24	
2	2024-03	880000	620000	260000	29.55	-4.35	
3	2024-04	950000	670000	280000	29.47	7.95	
4	2024-05	990000	700000	290000	29.29	4.21	
5	2024-06	1050000	740000	310000	29.52	6.06	
6	2024-07	980000	690000	290000	29.59	-6.67	
7	2024-08	1020000	720000	300000	29.41	4.08	
8	2024-09	1100000	770000	330000	30.00	7.84	
9	2024-10	1080000	760000	320000	29.63	-1.82	
10	2024-11	1150000	810000	340000	29.57	6.48	
11	2024-12	1200000	850000	350000	29.17	4.35	

	profitability_rating
0	良好
1	良好
2	良好
3	良好
4	良好

5	良好
6	良好
7	良好
8	良好
9	良好
10	良好
11	良好

5.5.3 步骤 3：客户价值与信用分析

```
# 客户分级和价值分析
query = """
WITH customer_metrics AS (
    SELECT
        c.customer_id,
        c.customer_name,
        c.customer_type,
        c.credit_limit,
        COUNT(r.invoice_id) as total_invoices,
        SUM(r.amount) as total_sales,
        SUM(CASE WHEN r.paid = 1 THEN r.amount ELSE 0 END) as
↪   paid_amount,
        SUM(CASE WHEN r.paid = 0 THEN r.amount ELSE 0 END) as
↪   outstanding_amount,
        ROUND(100.0 * SUM(CASE WHEN r.paid = 1 THEN 1 ELSE 0 END) /
↪   COUNT(*), 2) as payment_rate,
        AVG(r.amount) as avg_invoice_amount
    FROM customers c
    LEFT JOIN receivables r ON c.customer_id = r.customer_id
    GROUP BY c.customer_id, c.customer_name, c.customer_type,
↪   c.credit_limit
),
```

```
customer_ranking AS (
    SELECT
        *,
        RANK() OVER (ORDER BY total_sales DESC) as sales_rank,
        CASE
            WHEN total_sales >= 60000 THEN 'A 类-大客户'
            WHEN total_sales >= 30000 THEN 'B 类-中客户'
            ELSE 'C 类-小客户'
        END as customer_class,
        CASE
            WHEN payment_rate >= 80 THEN '优质'
            WHEN payment_rate >= 50 THEN '良好'
            WHEN payment_rate >= 20 THEN '需关注'
            ELSE '高风险'
        END as credit_rating
    FROM customer_metrics
)
SELECT * FROM customer_ranking
ORDER BY sales_rank
"""
customer_analysis = pd.read_sql_query(query, conn)
print("\n2. 客户价值与信用分析: ")
print(customer_analysis)
```

2. 客户价值与信用分析:

```
  customer_id customer_name customer_type  credit_limit  total_invoices  \
0        C001           甲公司           VIP      100000.0               2
1        C003           丙公司           VIP      150000.0               1
2        C002           乙公司            普通       50000.0               1
3        C004           丁公司            普通       30000.0               0
```

```
   total_sales  paid_amount  outstanding_amount  payment_rate  \
0      55000.0        30000               25000          50.0
1      50000.0            0               50000           0.0
2      15000.0        15000                   0         100.0
3          NaN            0                   0           0.0

   avg_invoice_amount  sales_rank customer_class credit_rating
0             27500.0           1         B类-中客户            良好
1             50000.0           2         B类-中客户          高风险
2             15000.0           3         C类-小客户            优质
3                 NaN           4         C类-小客户          高风险
```

5.5.4 步骤 4：应收账款健康度分析

```
# 应收账款账龄和风险分析
query = """
SELECT
    COUNT(*) as total_invoices,
    SUM(amount) as total_receivables,
    SUM(CASE WHEN paid = 1 THEN amount ELSE 0 END) as collected,
    SUM(CASE WHEN paid = 0 THEN amount ELSE 0 END) as outstanding,
    ROUND(100.0 * SUM(CASE WHEN paid = 1 THEN amount ELSE 0 END) /
↪   SUM(amount), 2) as collection_rate,
    -- 账龄分析
    SUM(CASE WHEN paid = 0 AND julianday('now') - julianday(invoice_date)
↪   <= 30 THEN amount ELSE 0 END) as current,
    SUM(CASE WHEN paid = 0 AND julianday('now') - julianday(invoice_date)
↪   BETWEEN 31 AND 60 THEN amount ELSE 0 END) as aged_31_60,
    SUM(CASE WHEN paid = 0 AND julianday('now') - julianday(invoice_date)
↪   BETWEEN 61 AND 90 THEN amount ELSE 0 END) as aged_61_90,
    SUM(CASE WHEN paid = 0 AND julianday('now') - julianday(invoice_date)
↪   > 90 THEN amount ELSE 0 END) as aged_over_90,
```

```
    -- 风险评估
    CASE
        WHEN 100.0 * SUM(CASE WHEN paid = 0 AND julianday('now') -
↪   julianday(invoice_date) > 90 THEN amount ELSE 0 END) /
            NULLIF(SUM(CASE WHEN paid = 0 THEN amount ELSE 0 END), 0) <
↪   10 THEN '低风险'
        WHEN 100.0 * SUM(CASE WHEN paid = 0 AND julianday('now') -
↪   julianday(invoice_date) > 90 THEN amount ELSE 0 END) /
            NULLIF(SUM(CASE WHEN paid = 0 THEN amount ELSE 0 END), 0) <
↪   25 THEN '中风险'
        ELSE '高风险'
    END as ar_risk_level
FROM receivables
"""
ar_health = pd.read_sql_query(query, conn)
print("\n3. 应收账款健康度分析: ")
print(ar_health)
```

3. 应收账款健康度分析:

```
   total_invoices  total_receivables  collected  outstanding  collection_rate  \
0               5             130000      45000        85000            34.62

   current  aged_31_60  aged_61_90  aged_over_90 ar_risk_level
0        0           0           0         85000          高风险
```

5.5.5 步骤 5: 财务健康度综合评分

```
# 计算企业财务健康度综合评分
query = """
WITH monthly_growth AS (
    SELECT
```

```
        month,
        revenue,
        cost,
        100.0 * (revenue - cost) / revenue as gross_margin,
        100.0 * (revenue - LAG(revenue) OVER (ORDER BY month)) /
              NULLIF(LAG(revenue) OVER (ORDER BY month), 0) as
 ↪  growth_rate
    FROM monthly_revenue
),
metrics AS (
    SELECT
        -- 盈利能力指标
        (SELECT AVG(gross_margin) FROM monthly_growth) as
 ↪  avg_gross_margin,
        -- 收款能力指标
        (SELECT 100.0 * SUM(CASE WHEN paid = 1 THEN amount ELSE 0 END) /
 ↪  SUM(amount) FROM receivables) as collection_rate,
        -- 增长能力指标
        (SELECT AVG(growth_rate) FROM monthly_growth WHERE growth_rate IS
 ↪  NOT NULL) as avg_growth_rate
),
scores AS (
    SELECT
        avg_gross_margin,
        collection_rate,
        COALESCE(avg_growth_rate, 0) as avg_growth_rate,
        -- 各维度评分（满分 100）
        CASE
            WHEN avg_gross_margin >= 35 THEN 100
            WHEN avg_gross_margin >= 25 THEN 80
            WHEN avg_gross_margin >= 15 THEN 60
            ELSE 40
```

```
        END as profitability_score,
        CASE
            WHEN collection_rate >= 80 THEN 100
            WHEN collection_rate >= 60 THEN 80
            WHEN collection_rate >= 40 THEN 60
            ELSE 40
        END as collection_score,
        CASE
            WHEN COALESCE(avg_growth_rate, 0) >= 10 THEN 100
            WHEN COALESCE(avg_growth_rate, 0) >= 5 THEN 80
            WHEN COALESCE(avg_growth_rate, 0) >= 0 THEN 60
            ELSE 40
        END as growth_score
    FROM metrics
)
SELECT
    ROUND(avg_gross_margin, 2) as gross_margin_pct,
    ROUND(collection_rate, 2) as collection_rate_pct,
    ROUND(avg_growth_rate, 2) as avg_growth_rate_pct,
    profitability_score,
    collection_score,
    growth_score,
    ROUND((profitability_score * 0.4 + collection_score * 0.3 +
↪   growth_score * 0.3), 2) as overall_score,
    CASE
        WHEN (profitability_score * 0.4 + collection_score * 0.3 +
↪   growth_score * 0.3) >= 90 THEN 'A-优秀'
        WHEN (profitability_score * 0.4 + collection_score * 0.3 +
↪   growth_score * 0.3) >= 80 THEN 'B-良好'
        WHEN (profitability_score * 0.4 + collection_score * 0.3 +
↪   growth_score * 0.3) >= 70 THEN 'C-中等'
        WHEN (profitability_score * 0.4 + collection_score * 0.3 +
↪   growth_score * 0.3) >= 60 THEN 'D-及格'
```

```
        ELSE 'F-需改进'
    END as health_grade
FROM scores
"""
health_score = pd.read_sql_query(query, conn)
print("\n4. 财务健康度综合评分: ")
print(health_score)
```

```
4. 财务健康度综合评分:
   gross_margin_pct  collection_rate_pct  avg_growth_rate_pct  \
0              29.5                34.62                 3.31

   profitability_score  collection_score  growth_score  overall_score  \
0                   80                40            60           62.0

  health_grade
0         D-及格
```

5.5.6 案例总结

```
print("\n" + "="*60)
print(" 财务分析仪表盘总结")
print("="*60)

# 输出关键发现
print("\n【关键发现】")
print(f"1. 平均毛利率: {health_score['gross_margin_pct'].iloc[0]:.2f}%")
print(f"2. 应收账款回款率:
↪ {health_score['collection_rate_pct'].iloc[0]:.2f}%")
print(f"3. 平均增长率:
↪ {health_score['avg_growth_rate_pct'].iloc[0]:.2f}%")
```

```
print(f"4. 财务健康度评分:
 ↪  {health_score['overall_score'].iloc[0]:.2f}分")
print(f"5. 综合评级: {health_score['health_grade'].iloc[0]}")

print("\n【管理建议】")
margin_pct = health_score['gross_margin_pct'].iloc[0]
if margin_pct < 20:
    print("• 毛利率偏低，建议优化成本结构或提升产品定价")
elif margin_pct < 30:
    print("• 毛利率处于合理区间，可继续保持")
else:
    print("• 毛利率表现优秀，盈利能力强")

collection_pct = health_score['collection_rate_pct'].iloc[0]
if collection_pct < 60:
    print("• 应收账款回款率较低，需加强催收管理")
elif collection_pct < 80:
    print("• 应收账款回款情况一般，建议优化信用政策")
else:
    print("• 应收账款回款情况良好，现金流健康")

print("\n" + "="*60)
```

```
============================================================
财务分析仪表盘总结
============================================================

【关键发现】
1. 平均毛利率: 29.50%
2. 应收账款回款率: 34.62%
3. 平均增长率: 3.31%
```

4. 财务健康度评分：62.00分

5. 综合评级：D-及格

【管理建议】

- 毛利率处于合理区间,可继续保持
- 应收账款回款率较低,需加强催收管理

==========

Tip

案例要点总结：

本案例展示了 SQL 在财务分析中的强大能力：

1. **数据整合：** 通过 JOIN 连接多个数据表
2. **指标计算：** 使用聚合函数和窗口函数计算各类财务指标
3. **趋势分析：** 使用 LAG/LEAD 函数进行环比分析
4. **分类评级：** 使用 CASE 语句进行多维度分级
5. **综合评分：** 构建加权评分模型
6. **自动化报告：** 一键生成完整的财务分析仪表盘

在实际工作中，这些 SQL 查询可以：- 保存为视图 (VIEW) 供反复使用 - 定时执行生成定期报告 - 导出为 Excel/CSV 供进一步分析 - 集成到 BI 工具 (如 Tableau、Power BI) 中可视化展示

5.5.7 关闭数据库连接

```
# 关闭连接
conn.close()
print(" 数据库连接已关闭")
```

数据库连接已关闭

5.6 总结

5.6.1 本章学习内容回顾

通过本章的学习, 你已经掌握了 SQL 在会计领域的核心应用技能:

1. SQL 基础知识

- 数据库的基本概念 (表、列、行、主键、外键)
- 会计科目表、凭证表的数据库设计
- 双重记账法在数据库中的实现

2. 核心 SQL 操作

- **数据查询**: SELECT、WHERE、ORDER BY
- **数据聚合**: COUNT、SUM、AVG、MAX、MIN
- **多表连接**: INNER JOIN、LEFT JOIN 用于关联科目表、客户表、凭证表
- **分组统计**: GROUP BY、HAVING 进行科目汇总和分类统计

3. 高级 SQL 技术

- **子查询**: 用于比较分析和条件筛选
- **窗口函数**: ROW_NUMBER、RANK、LAG/LEAD 实现排名和环比分析
- **CTE**: WITH 语句编写清晰的多步骤查询
- **CASE 语句**: 实现账龄分析、风险评级、科目分类
- **UNION**: 合并报表数据

4. 会计实务应用

- **试算平衡表**: 验证借贷平衡, 汇总科目发生额
- **资产负债表**: 编制财务状况报表, 验证会计等式
- **利润表**: 计算收入、成本、利润及财务比率
- **应收账款分析**: 账龄分析、回款率计算、风险评估
- **客户价值分析**: 客户分级、信用评级
- **财务健康度评分**: 多维度综合评价体系

5. Python 与 SQL 集成

- 使用 `sqlite3` 模块操作数据库
- 使用 `pandas.read_sql_query()` 读取查询结果
- 使用 `DataFrame.to_sql()` 将数据写入数据库
- 使用 `chunksize` 处理大数据集

5.6.2 SQL 在会计工作中的价值

> **!** Important
>
> **为什么会计人员需要掌握 SQL?**
>
> 1. **数据量爆炸：** 现代企业的交易数据以 TB 为单位,Excel 已无法胜任
> 2. **自动化需求：** 月度、季度报表需要快速生成,SQL 可实现一键出报表
> 3. **深度分析：** 从海量数据中挖掘洞察, 如客户价值分析、异常交易识别
> 4. **系统对接：** ERP、财务软件底层都是数据库, 掌握 SQL 可直接查询原始数据
> 5. **职业竞争力：** SQL 已成为高级会计、财务分析师的必备技能

5.6.3 下一步学习建议

1. **练习实战**
   - 下载真实的财务数据集进行练习
   - 尝试编写自己公司的财务报表 SQL
   - 参与 Kaggle 等平台的财务数据分析竞赛
2. **深入学习**
   - 学习 PostgreSQL、MySQL 等生产级数据库
   - 了解数据库设计范式和索引优化
   - 学习存储过程和触发器
3. **工具整合**
   - 将 SQL 与 Excel Power Query 结合
   - 学习 Tableau、Power BI 等可视化工具
   - 探索 Python 自动化脚本定期生成报表
4. **持续实践**

- 建立个人财务数据库项目
- 为家庭或小微企业建立简单的财务系统
- 在工作中主动寻找 SQL 应用场景

记住：SQL 是一门”做中学”的技能，只有通过大量实践才能真正掌握。从今天开始，在你的会计工作中寻找 SQL 的应用场景，将学习转化为生产力！

6 文件路径与数据持久化

点击下载本章节代码

> **本章学习目标**
>
> - 掌握现代 Python 文件路径操作（pathlib）
> - 学会高效的文件读写方法
> - 了解 Python 对象序列化与反序列化
> - 掌握大规模数据文件处理技巧

6.1 环境准备

```
%pip install joblib pandas openpyxl pyarrow fastparquet tqdm --upgrade
```

> **为什么使用 pathlib?**
>
> Python 3.4+ 引入的 `pathlib.Path` 相比传统的 `os.path` 有诸多优势：
>
> - 面向对象的 API，更直观易用
> - 跨平台兼容性更好（自动处理 Windows/Linux/Mac 路径差异）
> - 支持链式调用，代码更简洁
> - 集成了常用文件操作方法（读写、复制、删除等）

6.2 路径操作基础

```
from pathlib import Path
import pandas as pd
from tqdm import tqdm
```

```
Path.cwd()
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course')
```

```
file = Path("./data/example.txt")
type(file)
```

```
pathlib._local.PosixPath
```

```
file.exists()
```

```
True
```

```
file.absolute()
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data/example.txt')
```

6.3 路径拆分

```
file = file.absolute()
file
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data/example.txt')
```

```
file.parts
```

```
('/', 'Users', 'xinlu', '_project', 'AI4MPAcc-Course', 'data', 'example.txt')
```

```
file.is_dir()
```

```
False
```

```
file.is_file()
```

```
True
```

```
file.name
```

```
'example.txt'
```

```
file.stem
```

```
'example'
```

```
file.suffix
```

```
'.txt'
```

```
file.parent
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data')
```

```
list(file.parents)
```

```
[PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data'),
 PosixPath('/Users/xinlu/_project/AI4MPAcc-Course'),
 PosixPath('/Users/xinlu/_project'),
 PosixPath('/Users/xinlu'),
 PosixPath('/Users'),
 PosixPath('/')]
```

6.4 文本文件读写

Path 的便捷读写方法

pathlib.Path 提供了简洁的文本读写方法：

- Path.read_text() - 读取整个文件为字符串
- Path.write_text() - 将字符串写入文件
- Path.read_bytes() - 读取二进制文件
- Path.write_bytes() - 写入二进制数据

这些方法自动处理文件的打开和关闭，比传统的 open() 更简洁。

```
# 创建目录（如果不存在）
Path("./data/").mkdir(parents=True, exist_ok=True)
```

```
# 写入文本文件（推荐始终指定 encoding="utf-8"）
file.write_text('暨南', encoding="utf-8")
```

2

```
# 读取文本文件
file.read_text(encoding="utf-8")
```

'暨南'

```
# 编码错误演示：用错误的编码读取
try:
    file.read_text(encoding="gbk")
except Exception as e:
    print(f" 编码错误：{e}")
```

```
# 删除文件
file.unlink()
```

⚠ 字符编码陷阱

在处理中文文本时，务必：

1. **写入时明确指定 encoding="utf-8"**
2. **读取时使用相同编码**
3. **避免使用 gbk 或系统默认编码（容易产生乱码）**

推荐统一使用 UTF-8 编码。

```
try:
    file.write_text('暨南大学管理学院', encoding="utf-8")
    # 尝试用错误的编码读取会报错
    print(file.read_text(encoding="gbk"))
except Exception as e:
    print(f" 错误: {e}")
finally:
    # 正确读取
    print(file.read_text(encoding="utf-8"))
```

```
错误: 'gbk' codec can't decode byte 0xad in position 10: illegal multibyte sequence
暨南大学管理学院
```

6.5 文件重命名

```
file.with_suffix(".json")
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data/example.json')
```

```
file.rename("./data/example_new_name.txt")
```

```
PosixPath('data/example_new_name.txt')
```

```
print(file)
```

```
file.exists()
```

```
/Users/xinlu/_project/AI4MPAcc-Course/data/example.txt
```

```
False
```

```
Path("./data/example_new_name.txt").rename("./data/example.txt")
```

```
PosixPath('data/example.txt')
```

6.6 COPY/MOVE/DELETE 文件/文件夹

```
Path("./make/new/folder/").mkdir(parents=True, exist_ok=True)
```

```
import shutil
shutil.copy(file, Path("./make/new/folder/"))
```

```
'make/new/folder/example.txt'
```

```
new_file = file.with_suffix(".json")
new_file
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data/example.json')
```

```
shutil.copyfile(file, new_file)
```

```
PosixPath('/Users/xinlu/_project/AI4MPAcc-Course/data/example.json')
```

```
shutil.move(new_file, Path("./make/new/folder/"))
```

```
'make/new/folder/example.json'
```

```
shutil.copytree("./make", "./data/make_copy")
```

```
'./data/make_copy'
```

```
shutil.rmtree("./make", ignore_errors=True)
shutil.rmtree("./data/make_copy", ignore_errors=True)
```

6.7 文件搜索与批量处理

> **glob vs rglob**
>
> - `glob(pattern)` - 只搜索当前目录
> - `rglob(pattern)` - 递归搜索所有子目录（r = recursive）
>
> 常用通配符：- `*` - 匹配任意字符 - `**` - 匹配任意级别的目录 - `?` - 匹配单个字符 - `[abc]` - 匹配括号内的任一字符

```
import os

# 获取当前 Python 环境路径
env_path = os.environ.get('CONDA_PREFIX', '.')
print(f" 环境路径: {env_path}")
```

```
环境路径: /Users/xinlu/miniconda3
```

```
# 搜索当前目录下的所有 .py 文件（不含子目录）
py_files = list(Path(env_path).glob('*.py'))
print(f" 找到 {len(py_files)} 个 .py 文件")
py_files[:3]  # 显示前 3 个
```

找到 0 个 .py 文件

[]

```
# 递归搜索所有子目录中的 .py 文件
all_py_files = list(Path(env_path).rglob('*.py'))
print(f" 递归搜索找到 {len(all_py_files)} 个 .py 文件")
all_py_files[:3]
```

递归搜索找到 48937 个 .py 文件

```
[PosixPath('/Users/xinlu/miniconda3/bin/pdf2txt.py'),
 PosixPath('/Users/xinlu/miniconda3/bin/dumppdf.py'),
 PosixPath('/Users/xinlu/miniconda3/bin/pwiz.py')]
```

```
# 批量复制图片文件示例
png_files = Path(env_path).rglob('*.png')

# 创建目标文件夹
output_dir = Path("./data/collected_images/")
output_dir.mkdir(parents=True, exist_ok=True)

# 使用 tqdm 显示进度条
from shutil import copy2

for i, img_path in enumerate(tqdm(list(png_files)[:10], desc=" 复制图
↪  片")):
    if img_path.is_file():
        # 为避免文件名冲突，添加序号
        new_name = f"{i:03d}_{img_path.name}"
        copy2(img_path, output_dir / new_name)

print(f" 已复制到: {output_dir.absolute()}")
```

已复制到: /Users/xinlu/_project/AI4MPAcc-Course/data/collected_images

```
# 筛选文件和目录
items = list(Path(env_path).glob('*'))

files = [x for x in items if x.is_file()]
dirs = [x for x in items if x.is_dir()]

print(f" 文件数量: {len(files)}")
print(f" 目录数量: {len(dirs)}")
print(f" 目录示例: {dirs[:3]}")
```

文件数量: 6
目录数量: 15
目录示例: [PosixPath('/Users/xinlu/miniconda3/man'), PosixPath('/Users/xinlu/minicond

6.8 保存和加载 Python 单个对象

```
import joblib
# 创建一个示例对象，例如一个简单的字典
data = {'name': 'Alice', 'age': 25, 'city': 'New York'}

# 使用 joblib 保存对象到文件
joblib.dump(data, 'data.joblib')
```

['data.joblib']

```
# 加载保存的对象
loaded_data = joblib.load('data.joblib')
loaded_data
```

{'name': 'Alice', 'age': 25, 'city': 'New York'}

```
Path('data.joblib').unlink()
```

6.9 保存和加载 Python 多个对象

```
import joblib

# 创建多个示例对象
model = {'type': 'LinearRegression', 'parameters': {'intercept': 0.5,
 ↪  'slope': 2.3}}
config = {'batch_size': 32, 'epochs': 100}
data = [1, 2, 3, 4, 5]

# 使用 joblib 保存多个对象到一个文件
joblib.dump((model, config, data), 'multiple_objects.joblib')

# 加载保存的多个对象
loaded_model, loaded_config, loaded_data =
 ↪  joblib.load('multiple_objects.joblib')

print(loaded_model)
print(loaded_config)
print(loaded_data)
```

```
{'type': 'LinearRegression', 'parameters': {'intercept': 0.5, 'slope': 2.3}}
{'batch_size': 32, 'epochs': 100}
[1, 2, 3, 4, 5]
```

```
Path('multiple_objects.joblib').unlink()
```

...

- 把数据转化为 DataFrame，然后利用 pandas 的强大 IO 接口存为 xlsx, csv 等数据

6.10 DataFrame 拆分例子

```
from dfply import diamonds
```

```
Path('./temp/').mkdir()
```

```
clarity_set = set(diamonds.clarity.tolist())
clarity_set
```

```
{'I1', 'IF', 'SI1', 'SI2', 'VS1', 'VS2', 'VVS1', 'VVS2'}
```

```
for c in clarity_set:
  df_subset = diamonds[diamonds.clarity == c]
  df_subset.to_excel(f'./temp/clarity_{c}.xlsx', index = False)
  print(f'clarity = {c} save completed!')
```

```
clarity = VVS2 save completed!
clarity = I1 save completed!
clarity = VS2 save completed!
clarity = VVS1 save completed!
clarity = VS1 save completed!
clarity = SI2 save completed!
clarity = SI1 save completed!
clarity = IF save completed!
```

6.11 DataFrame 读取例子

```
file_list = [x for x in Path('./temp/').glob("*.xlsx")]
file_list
```

```
[PosixPath('temp/clarity_VVS1.xlsx'),
 PosixPath('temp/clarity_SI2.xlsx'),
 PosixPath('temp/clarity_VS1.xlsx'),
 PosixPath('temp/clarity_I1.xlsx'),
 PosixPath('temp/clarity_IF.xlsx'),
 PosixPath('temp/clarity_VS2.xlsx'),
 PosixPath('temp/clarity_SI1.xlsx'),
 PosixPath('temp/clarity_VVS2.xlsx')]
```

```
import pandas as pd
df_list = [pd.read_excel(x) for x in file_list]
df_list[0].head(2)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.24	Very Good	I	VVS1	62.3	57.0	336	3.95	3.98	2.47
1	0.32	Ideal	I	VVS1	62.0	55.3	553	4.39	4.42	2.73

```
df_all = pd.concat(df_list)
df_all.shape
```

```
(53940, 10)
```

```
import shutil
shutil.rmtree("./temp/", ignore_errors=True)
```

7 文本分析入门

点击下载本章节代码

7.1 Spacy 模型

安装依赖包

```
%pip install spacy wordcloud gensim pyLDAvis tqdm --upgrade
```

下载中文语言模型（3.8.0 版本）

```
%pip install
↪ https://github.com/explosion/spacy-models/releases/download/zh_core_web_md-3.8.0/
```

或使用命令行下载：

```
python -m spacy download zh_core_web_md
```

7.2 加载模型

```
import spacy
nlp = spacy.load("zh_core_web_md")
```

```
text = " 暨南大学是中国第一所由政府创办的华侨学府。\
学校目前是中央统战部、教育部、广东省共建的国家“双一流”建设高校，直属中央
↪ 统战部管理。"
```

```
print(text)
doc = nlp(text)
```

暨南大学是中国第一所由政府创办的华侨学府。学校目前是中央统战部、教育部、广东省共建的国

```
for x in doc:
  print(x)
```

暨南
大学
是
中国
第一
所
由
政府
创办
的
华侨
学府
。
学校
目前
是
中央
统战部
、
教育部
、
广东省
共建

的
国家
“
双一流
”
建设
高校
，
直属
中央
统战部
管理
。

```
for i, sent in enumerate(doc.sents):
    print(i, sent)
```

0 暨南大学是中国第一所由政府创办的华侨学府。
1 学校目前是中央统战部、教育部、广东省共建的国家“双一流”建设高校，直属中央统战部管理

```
for i, sent in enumerate(doc.sents):
  print(i, "-----")
  for x in sent:
    print(x)
```

0 -----
暨南
大学
是
中国
第一
所
由

政府
创办
的
华侨
学府
。
1 -----
学校
目前
是
中央
统战部
、
教育部
、
广东省
共建
的
国家
“
双一流
”
建设
高校
，
直属
中央
统战部
管理
。

```
token = doc[0]
print(token)
print(" 1", token.text)
print(" 2", token.ent_type_)
print(" 4", token.is_alpha)
print(" 5", token.is_space)
print(" 6", token.is_stop)
print(" 7", token.is_punct)
print(" 8", token.is_currency)
print(" 9", token.is_upper)
print("10", token.pos_) # Part of Speech
```

```
暨南
 1 暨南
 2 ORG
 4 True
 5 False
 6 False
 7 False
 8 False
 9 False
10 PROPN
```

```
def clean_text(text):
  text = "".join(text.split())
  doc = nlp(text)
  token_list = [token.text for token in doc if token.is_alpha and not
↪  token.is_stop]
  return " ".join(token_list)
```

```
clean_text(" 暨南大学是中国第一所由政府创办的华侨学府。")
```

'暨南 大学 中国 第一 政府 创办 华侨 学府'

7.2.1 自定义分词

```
proper_nouns = ['暨南大学','华侨学府']
nlp.tokenizer.pkuseg_update_user_dict(proper_nouns)
```

```
clean_text(" 暨南大学是中国第一所由政府创办的华侨学府。")
```

'暨南大学 中国 第一 政府 创办 华侨学府'

7.2.2 自定义 stopwords

```
from spacy.lang.zh.stop_words import STOP_WORDS
#  set of Spacy's default stop words

for word in [" 第一", " 学府"]:
    STOP_WORDS.add(word) # 增加 stop words
    lexeme = nlp.vocab[word]
    lexeme.is_stop = True
```

```
clean_text(" 暨南大学是中国第一所由政府创办的华侨学府。")
```

'暨南大学 中国 政府 创办 华侨学府'

```
# 删除 stopwords
for word in [" 第一", " 学府", " 是"]:
    STOP_WORDS.remove(word) # 剔除 stop words
    lexeme = nlp.vocab[word]
    lexeme.is_stop = False
```

```
clean_text(" 暨南大学是中国第一所由政府创办的华侨学府。")
```

```
'暨南大学 是 中国 第一 政府 创办 华侨学府'
```

7.3 加载审计数据

```
import pandas as pd
df = pd.read_parquet(
    ↪ "https://ai4biz.oss-cn-guangzhou.aliyuncs.com/data/audit_descrpiption.parquet
   )

df = df[df.year == 2023].sample(100)
df.head()
```

	stkcd	year	audit_description
16589	600977	2023	中影股份主营业务包括创作、发行、放映、科技、服务及创新六大板块，涵盖电
22653	830832	2023	2023 年度营业收入为 37,170.14 万元。（1）鉴于齐鲁华信销售业务客户较为集中
1260	719	2023	中原传媒公司及其子公司主要从事出版物的印刷生产和销售，以及与印刷相关的
10703	300706	2023	截至 2023 年 12 月 31 日，阿石创公司应收账款余额 25,188.91 万元，坏账准备
12683	301421	2023	收入是波长光电的关键业绩指标之一，波长光电主要从事光机电产品和激光、红

7.4 清洗文本

```
from tqdm.notebook import tqdm
tqdm.pandas()

df["audit_description_seg"] =
 ↪ df["audit_description"].progress_apply(clean_text)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]
```

```
df.audit_description_seg.str.len().describe()
```

```
count    100.000000
mean     193.700000
std      113.019664
min       38.000000
25%      118.250000
50%      166.500000
75%      232.250000
max      618.000000
Name: audit_description_seg, dtype: float64
```

```
df['n_words'] = df.audit_description_seg.apply(lambda x: len(x.split()))
df['n_words'].describe()
```

```
count    100.000000
mean      64.700000
std       37.144952
min       13.000000
25%       41.000000
50%       56.000000
75%       78.250000
max      205.000000
Name: n_words, dtype: float64
```

7.5 词频向量

```
import pandas as pd
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
```

```
cv = CountVectorizer(
    # max_features=1800,
    min_df=0.01, # lower
    max_df = 0.99,
    ngram_range=(1, 1)
)

word_count_vector =
 ↪  cv.fit_transform(df["audit_description_seg"].tolist())
```

```
word_count_vector
```

```
<Compressed Sparse Row sparse matrix of dtype 'int64'
    with 3296 stored elements and shape (100, 945)>
```

```
cv.get_feature_names_out()
```

```
array(['cfr', 'cif', 'cip', 'cnf', 'cpt', 'dap', 'ddp', 'ddu',
       'dematicgrou', 'dematicgroup', 'exw', 'fca', 'fob', 'group', 'it',
       'kion', 'poct', 'p所', 'r组件', '一体化', '一时点', '万向', '三力', '三十', '三
       '三十五', '三十八', '三阶段', '上年', '上年度', '上年末', '上期', '上海', '上
       '不尽相同', '不确定性', '专家', '世荣', '业务', '业绩', '东亚', '东利', '东方'
       '中原', '中影', '中科', '中船', '中钢', '中铁', '中间体', '为经', '主体', '主
       '主观性', '义务', '之间', '书面', '买入', '二十', '二十三', '二十五', '二级',
       '互联网', '亚华', '亚虹', '交互', '交付', '交易', '交易性', '交易量', '交易额'
       '产品', '产线', '享有', '人民币', '亿联', '亿通', '仓储', '仓库', '代理', '仪器
       '价格', '价款', '企业', '休闲', '众多', '众辰', '会计', '会议', '传媒', '传感器
       '低于', '余额', '作出', '佣金', '供应链', '依赖', '依赖于', '保税区', '保证',
       '信宇人', '信息', '信用', '信质', '修订', '借款', '假设', '偏向', '做出', '储存
       '充值', '充分性', '兆业', '先款', '光机电', '光电', '入账', '全球', '全程', '公
       '公路', '六大', '关联', '关联方', '其他应收款', '内容', '内蒙', '内部', '内酰'
       '军工', '冰压机', '冰鲜', '冷链', '净值', '净额', '准则', '准确', '准确性', '冰
```

'减少', '减震器', '凭据', '凭证', '出口', '出库单', '出版', '出版物', '击量',
'分为', '分摊', '分散', '分析', '分部', '分配', '划分', '列为', '列报', '创作'
'判断', '利润', '利润表', '利益', '到期', '制作', '制动', '制品', '制片', '制
'前瞻性', '剧集', '剩余', '办理', '加工', '加权', '动力', '包含', '包括', '北
'十六', '华信', '华培', '华米', '华菱', '华闻', '协助', '协议', '单个', '单位'
'单时', '单核', '单笔', '单项', '南京', '博闻', '占期', '印制', '印刷', '历史'
'原值', '原则', '原料药', '厨柜', '参数', '参阅', '发出', '发动', '发展', '发
'发电', '发行', '发货', '受偿', '受限', '变化', '变更', '变现', '口径', '可变'
'各类', '各项', '合力', '合同', '合并', '合并利', '合成', '合理性', '合肥', '
'同比', '后台', '后续', '后货', '含有', '周期', '品牌', '售价', '商品', '商官网
'商誉', '喹喏', '四十三', '四十四', '回单', '回性', '回报', '回收', '因子', '
'固有', '国中', '国产', '国内', '国博', '国际', '圣阳', '地区', '地并', '地点'
'垫款', '基础', '境内', '境内线', '境外', '境外线', '增加', '增材', '增长', '
'处置', '复杂性', '外线', '外部', '外销', '多且', '多种', '大件', '大宗', '大
'妆品', '始确', '姚记', '子公司', '存续', '存续期', '存货', '季度', '宁波', '
'安徽', '安排', '安装', '完工', '完整', '完毕', '宏源', '宏观', '定义', '定制
'实时', '审价', '审定', '客户', '宣传', '家具', '家居', '家电', '家畜', '对价'
'导致', '导航', '导购', '寿命', '封件', '射频', '尚未', '尤其是', '层于', '展
'履行', '工业', '工具', '工程', '差异', '差异性', '差额', '市场', '带来', '常
'平均', '年内', '年度', '年限', '并经', '广告', '广告位', '应收', '应计', '店
'建立', '建设', '建造', '开关', '开发', '开设', '引导', '弘元', '当年', '当月'
'形车', '影响', '影视', '影院', '德邦', '快运', '快递', '快速', '总收入', '总
'恒帅', '恰当', '悬架', '情况', '情景', '成交', '成品油', '成本', '成果', '截
'房地产', '所在地', '所示', '所述', '手续', '扑克牌', '执行', '执行器', '执行
'承租人', '承诺', '承运人', '技术', '投入', '投放', '投资', '投资者', '抗真',
'抗菌药', '折现率', '报关', '报关单', '报告', '报表', '报表日', '披露', '抵押
'抽水', '担任', '担保人', '拥有', '持续', '指定', '指标', '按点', '损失', '损
'授予', '授权', '接单', '接受', '接口', '接收方', '控制', '控制权', '推广', '
'提前', '提单', '提存', '摊销', '摩托车', '播放', '操控', '操纵', '支付', '收
'收到', '收取', '收回', '收方', '收款', '收益', '收租', '收融', '收货', '收购'
'放映', '政务', '政府', '政策', '敞口', '数字', '数字化', '数据', '数智', '数
'文件', '新华', '新材', '新百', '方式', '施工', '无形', '无线', '无误', '无需'

'时段', '时点', '时计', '时间', '春风', '晋西', '普莱', '普莱得', '智能', '智
'暂定', '暴露', '最为', '最佳', '月度', '有效性', '有线', '有限', '服务', '服
'服务费', '朗博', '期内', '期初', '期望', '期末', '期间', '期限', '木质', '未
'本期', '机制', '机器', '机密', '机床', '机械', '机载', '权利', '权力', '权益
'条件', '条款', '来源', '来源于', '来自于', '杰华特', '松茸', '板块', '标准',
'核算', '根据线', '梦天', '概率', '模具', '模块', '模型', '模式', '模组', '款
'正威', '正确', '正裕', '母公司', '每月', '比例', '比重', '水务', '水平', '永
'汇总', '汉光', '江苏', '污水', '汽车', '沥青', '沪电', '治理', '波长', '注册
'洛耐', '活动', '活鸡', '派林', '流入', '流动', '流程', '流贴', '流转', '流量
'浙江', '海关', '海外', '海格', '海通', '消耗', '消费者', '涉及', '润表', '涵
'渠道', '港口', '游戏', '湖雪', '湘佳', '湘邮', '潍柴', '潜在', '激光', '激励
'热电', '焊割', '煤炭', '版块', '物流', '物资', '特定', '特征', '特许', '特货
'玩家', '环境', '现值', '现时', '现法', '现金', '现金流', '珀莱雅', '瑞凌', '
'生态', '生物', '生物质', '生猪', '用于', '用户', '田野', '申万', '电力', '电
'电商', '电子', '电影', '电新', '电机', '电池', '电源', '电视', '电路板', '白
'皖通', '皮肤', '益佰', '监控', '目标', '目的', '直营', '相关', '相当于', '相
'真实性', '研发', '硬件', '确认', '碱药', '票务', '离子', '离岸', '禽肉', '科
'租售', '租赁', '租赁物', '程序', '程度', '税费', '站到站', '笔数', '第三方',
'策略', '签字', '签收', '签署', '签订', '简称', '管理', '管理层', '管理者', '
'精密', '精工', '精机', '系统', '索赔', '累计', '红外', '约定', '纳入', '线下
'组合', '终了', '终端', '经常性', '经接', '经济', '经营', '经营权', '经购', '
'经销商', '结构化', '结算', '统称', '绩效', '维修', '维生素', '综合', '绿能',
'网络', '联产', '联合', '联方', '联盟', '聘请', '股份', '股权', '胺类', '能力
'自主', '自初', '自动化', '自有', '自营', '自行', '至少', '航天', '航科', '航
'芯片', '若干', '药业', '药品', '莱赛', '获取', '菌制品', '菌药', '菲利华', '
'蒸汽', '蓄能', '虚增', '虚拟', '蚕丝', '融出', '融合', '融资', '血液', '行业
'补价', '表中', '表日', '表现', '装备', '装运港', '视为', '视觉', '角色', '解
'计入', '计提', '计算', '计费', '计量', '订单', '认后', '认定', '记录', '设备
'访问', '证券', '评估', '评估师', '识别', '诊断', '该类', '详见', '调整', '调
'负债', '财信', '财务', '账务', '账户', '账方式', '账款', '账确', '账面', '账
'货方', '货款', '货物', '购买', '购子', '购货方', '贵州', '贷款', '贸易', '费
'资产', '资产组', '资本', '资本化', '资源', '资租', '资金', '超卓', '超市', '

```
       '跌价', '跨期', '车站', '车轴', '转移', '软件', '软硬件', '载明', '输入', '输
       '运作', '运动', '运至', '运营', '运营商', '运行', '运输', '运输交', '运送', '
       '进口', '进度', '违约', '迹象', '追踪', '送货', '选取', '选择', '通信', '通知
       '逻辑', '邮件', '部品', '部门', '配件', '配合', '配电', '酮类', '采用', '重要
       '金徽酒', '金牌', '金租', '金融', '金证', '金额', '钱潮', '铁路', '铅蓄', '铝
       '销售', '销售化', '销售密', '销售量', '错报', '错误', '镇海', '长城', '长期',
       '长高电新', '门到门', '阀门', '阶段', '阿石创', '附注', '院线', '集团', '集成
       '零部件', '非t', '音飞', '项目', '顺序', '预估', '预期', '预测', '预计', '领域
       '频繁', '风险', '飞机', '食品', '食用', '验收', '高度', '高电', '高者', '高速
       '黄金', '齐鲁', '龙利', '龙利得'], dtype=object)
```

```
word_count_vector.shape
```

```
(100, 945)
```

```
word_count_vector.toarray() # type: ignore
```

```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 3, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], shape=(100, 945))
```

```
word_tfidf_vector = TfidfTransformer().fit_transform(word_count_vector)
word_tfidf_vector
```

```
<Compressed Sparse Row sparse matrix of dtype 'float64'
    with 3296 stored elements and shape (100, 945)>
```

```
word_tfidf_vector.toarray() # type: ignore
```

```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.54476285, 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]], shape=(100, 945))
```

```
word_count = pd.DataFrame(word_count_vector.toarray(), # type: ignore
                          columns = cv.get_feature_names_out(),
                          index = df['stkcd'])


word_count
```

stkcd	cfr	cif	cip	cnf	cpt	dap	ddp	ddu	dematicgrou	dematicgroup	...	验收	高度
600977	0	0	0	0	0	0	0	0	0	0	...	0	0
830832	0	0	0	0	0	0	0	0	0	0	...	0	0
719	0	0	0	0	0	0	0	0	0	0	...	0	0
300706	0	0	0	0	0	0	0	0	0	0	...	0	0
301421	0	0	0	0	0	0	0	0	0	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2853	0	0	0	0	0	0	0	0	0	0	...	0	0

stkcd	cfr	cif	cip	cnf	cpt	dap	ddp	ddu	dematicgrou	dematicgroup	...	验收	高度
338	0	0	0	0	0	0	0	0	2	1	...	0	0
603180	0	0	0	0	0	0	0	0	0	0	...	0	0
603056	0	0	0	0	0	0	0	0	0	0	...	0	0
793	0	0	0	0	0	0	0	0	0	0	...	0	0

7.5.1 基于词典的方法

```
word_count
```

stkcd	cfr	cif	cip	cnf	cpt	dap	ddp	ddu	dematicgrou	dematicgroup	...	验收	高度
600977	0	0	0	0	0	0	0	0	0	0	...	0	0
830832	0	0	0	0	0	0	0	0	0	0	...	0	0
719	0	0	0	0	0	0	0	0	0	0	...	0	0
300706	0	0	0	0	0	0	0	0	0	0	...	0	0
301421	0	0	0	0	0	0	0	0	0	0	...	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
2853	0	0	0	0	0	0	0	0	0	0	...	0	0
338	0	0	0	0	0	0	0	0	2	1	...	0	0
603180	0	0	0	0	0	0	0	0	0	0	...	0	0
603056	0	0	0	0	0	0	0	0	0	0	...	0	0
793	0	0	0	0	0	0	0	0	0	0	...	0	0

```
wcdf = word_count.stack().reset_index()
wcdf
```

	stkcd	level_1	0
0	600977	cfr	0
1	600977	cif	0
2	600977	cip	0
3	600977	cnf	0
4	600977	cpt	0
...	...	...	...
94495	793	鱼微	0
94496	793	黄金	0
94497	793	齐鲁	0
94498	793	龙利	0
94499	793	龙利得	0

```
wcdf.columns = ["stkcd", "word", "N"]
wcdf = wcdf[wcdf.N > 0].sort_values(by=['stkcd',
 ↪  "N"]).reset_index(drop=True)
wcdf
```

	stkcd	word	N
0	166	包括	1
1	166	影响	1
2	166	投资	1
3	166	投资者	1
4	166	担任	1
...	...	...	...
3291	871396	执行器	4
3292	871396	控制	4
3293	871396	阀门	4
3294	871396	收入	5
3295	871396	销售	5

7.6 词云

```
from wordcloud import WordCloud
```

```
wordclouddf =
 ↪  wcdf.groupby('word').N.sum().reset_index().sort_values(by='N',
 ↪  ascending=False).set_index('word')
wordclouddf
```

word	N
收入	457
确认	215
公司	204
营业	151
销售	129
...	...
地并	1
版块	1
煤炭	1
焊割	1
cfr	1

```
worddic = wordclouddf["N"].to_dict()
worddic
```

```
{'收入': 457,
 '确认': 215,
 '公司': 204,
 '营业': 151,
 '销售': 129,
```

'管理层': 98,
'业务': 91,
'年度': 90,
'风险': 76,
'指标': 66,
'产品': 65,
'业绩': 61,
'金额': 59,
'财务': 56,
'人民币': 54,
'固有': 54,
'客户': 50,
'报表': 48,
'特定': 48,
'目标': 48,
'资产': 47,
'减值': 46,
'预期': 41,
'信用': 40,
'股份': 38,
'判断': 35,
'影响': 34,
'合同': 32,
'相关': 32,
'识别': 32,
'商誉': 31,
'合并': 31,
'服务': 31,
'时点': 30,
'科技': 29,
'应收': 28,
'操纵': 28,

'估计': 27,
'主营': 27,
'期望': 27,
'损失': 27,
'恰当': 27,
'模式': 27,
'账款': 26,
'会计': 25,
'涉及': 25,
'账面': 24,
'系统': 24,
'生产': 23,
'价值': 22,
'项目': 20,
'包括': 19,
'评估': 17,
'设备': 17,
'计量': 17,
'简称': 17,
'控制': 17,
'信息': 17,
'经营': 16,
'货物': 16,
'工程': 16,
'成本': 15,
'金融': 15,
'假设': 15,
'收回': 14,
'预计': 14,
'附注': 14,
'出口': 14,
'余额': 14,

'来自于': 14,
'模型': 13,
'测试': 13,
'履约': 13,
'约定': 13,
'错报': 13,
'平台': 12,
'商品': 12,
'比例': 12,
'发生': 12,
'坏账': 12,
'参数': 12,
'集团': 11,
'控制权': 11,
'外销': 11,
'开发': 11,
'收款': 11,
'资产组': 11,
'情况': 10,
'指定': 10,
'提供': 10,
'主体': 10,
'特许': 10,
'制药': 10,
'结构化': 10,
'公路': 10,
'运输': 10,
'管理': 10,
'报关': 9,
'未来': 9,
'物流': 9,
'药业': 9,

'子公司': 9,
'现金': 9,
'利润': 9,
'研发': 9,
'运营': 9,
'交付': 9,
'计提': 8,
'赁款': 8,
'经销商': 8,
'经营权': 8,
'地点': 8,
'采用': 8,
'汽车': 8,
'房地产': 8,
'签收': 8,
'义务': 8,
'网络': 8,
'游戏': 8,
'收取': 8,
'增加': 7,
'认定': 7,
'快运': 7,
'快递': 7,
'收租': 7,
'因素': 7,
'总额': 7,
'期间': 7,
'交易': 7,
'发货': 7,
'进度': 7,
'通信': 7,
'无形': 7,

'状况': 7,
'技术': 7,
'违约': 7,
'过程': 7,
'长期': 7,
'收费': 7,
'来源于': 7,
'存货': 7,
'权利': 7,
'流量': 7,
'政策': 7,
'电源': 6,
'经济': 6,
'原值': 6,
'前瞻性': 6,
'能源': 6,
'电商': 6,
'益佰': 6,
'发行': 6,
'结算': 6,
'组合': 6,
'验收': 6,
'收到': 6,
'潜在': 6,
'投资': 6,
'负债': 6,
'设计': 6,
'合计': 6,
'it': 6,
'境外': 6,
'本期': 6,
'内销': 6,

'履行': 6,
'境内': 6,
'普莱': 6,
'宏源': 5,
'金租': 5,
'动力': 5,
'皖通': 5,
'润表': 5,
'运至': 5,
'江苏': 5,
'圣阳': 5,
'综合': 5,
'注释': 5,
'湘佳': 5,
'合并利': 5,
'智能': 5,
'承包': 5,
'方式': 5,
'所述': 5,
'瑞凌': 5,
'报告': 5,
'截止': 5,
'成果': 5,
'日发': 5,
'零部件': 5,
'净值': 5,
'国博': 5,
'准确性': 5,
'电子': 5,
'信质': 5,
'收购': 5,
'有限': 5,

'依赖': 5,
'重要性': 5,
'之间': 5,
'上年': 5,
'转移': 5,
'广告': 5,
'航空': 5,
'机械': 5,
'条款': 5,
'股权': 5,
'来源': 5,
'镇海': 5,
'数据': 5,
'中铁': 4,
'珀莱雅': 4,
'激光': 4,
'现值': 4,
'特货': 4,
'固定': 4,
'增长': 4,
'购货方': 4,
'杰华特': 4,
'中原': 4,
'一时点': 4,
'弘元': 4,
'当虹': 4,
'智造': 4,
'普莱得': 4,
'德邦': 4,
'总收入': 4,
'所示': 4,
'大宗': 4,

'执行器': 4,
'新材': 4,
'kion': 4,
'提单': 4,
'推广': 4,
'损失率': 4,
'阶段': 4,
'阀门': 4,
'工业': 4,
'用户': 4,
'梦天': 4,
'正威': 4,
'家居': 4,
'正裕': 4,
'世荣': 4,
'东亚': 4,
'比重': 4,
'汇宇': 4,
'污水': 4,
'波长': 4,
'派林': 4,
'东利': 4,
'海格': 4,
'生物': 4,
'国内': 4,
'申万': 4,
'光电': 4,
'变现': 4,
'传媒': 4,
'企业': 4,
'超卓': 4,
'签订': 4,

'关联': 4,
'订单': 4,
'价格': 4,
'航天': 4,
'包含': 4,
'航科': 4,
'芯片': 4,
'博闻': 4,
'准则': 4,
'表日': 4,
'历史': 4,
'减去': 4,
'联合': 4,
'程序': 4,
'兆业': 4,
'费用': 4,
'产线': 4,
'贷款': 4,
'充值': 4,
'作出': 4,
'货款': 4,
'制品': 4,
'精机': 4,
'各类': 4,
'电影': 4,
'绿能': 4,
'货币': 4,
'概率': 3,
'交货': 3,
'莱赛': 3,
'不确定性': 3,
'获取': 3,

'常辅': 3,
'导购': 3,
'权重': 3,
'条件': 3,
'市场': 3,
'错误': 3,
'亿通': 3,
'期末': 3,
'利益': 3,
'集成': 3,
'代理': 3,
'证券': 3,
'龙利': 3,
'齐鲁': 3,
'高速': 3,
'贵州': 3,
'提前': 3,
'贸易': 3,
'预测': 3,
'财信': 3,
'折现率': 3,
'数字化': 3,
'资金': 3,
'三力': 3,
'施工': 3,
'手续': 3,
'截止性': 3,
'计算': 3,
'计入': 3,
'时间': 3,
'春风': 3,
'恒帅': 3,

'装备': 3,
'跌价': 3,
'办理': 3,
'工具': 3,
'中影': 3,
'真实': 3,
'依赖于': 3,
'百货': 3,
'迹象': 3,
'基础': 3,
'流程': 3,
'租赁物': 3,
'宁波': 3,
'流入': 3,
'安排': 3,
'电池': 3,
'亚华': 3,
'回收': 3,
'海通': 3,
'发展': 3,
'运行': 3,
'印刷': 3,
'现时': 3,
'华信': 3,
'合肥': 3,
'现金流': 3,
'组件': 3,
'海外': 3,
'真实性': 3,
'环境': 3,
'终端': 3,
'运营商': 3,

'计费': 2,
'音飞': 2,
'认后': 2,
'相当于': 2,
'火腿': 2,
'湘邮': 2,
'选择': 2,
'金牌': 2,
'物资': 2,
'白酒': 2,
'记录': 2,
'接受': 2,
'金徽酒': 2,
'该类': 2,
'玩家': 2,
'电新': 2,
'配件': 2,
'电机': 2,
'高者': 2,
'部门': 2,
'操控': 2,
'食用': 2,
'支付': 2,
'食品': 2,
'邮件': 2,
'政府': 2,
'预估': 2,
'电视': 2,
'电路板': 2,
'调试': 2,
'科目': 2,
'调整': 2,

'特征': 2,
'详见': 2,
'政务': 2,
'运送': 2,
'经销': 2,
'湖雪': 2,
'程度': 2,
'第三方': 2,
'自动化': 2,
'权力': 2,
'长城': 2,
'自主': 2,
'软件': 2,
'电力': 2,
'治理': 2,
'沪电': 2,
'精工': 2,
'销售量': 2,
'松茸': 2,
'板块': 2,
'标准': 2,
'核算': 2,
'汉光': 2,
'终了': 2,
'模具': 2,
'水务': 2,
'钱潮': 2,
'模组': 2,
'流转': 2,
'机床': 2,
'计价': 2,
'药品': 2,

'租赁': 2,
'晋西': 2,
'渠道': 2,
'清洗': 2,
'消费者': 2,
'表中': 2,
'补价': 2,
'衡量': 2,
'行业': 2,
'消耗': 2,
'月度': 2,
'虚拟': 2,
'虚增': 2,
'阿石创': 2,
'朗博': 2,
'车轴': 2,
'期内': 2,
'菌制品': 2,
'期限': 2,
'长高': 2,
'金证': 2,
'收货': 2,
'龙利得': 2,
'庞大': 2,
'出库单': 2,
'冷链': 2,
'价款': 2,
'发电': 2,
'军工': 2,
'当月': 2,
'审价': 2,
'分摊': 2,

'中间体': 2,
'中钢': 2,
'中船': 2,
'中科': 2,
'影视': 2,
'内部': 2,
'东方': 2,
'恒华': 2,
'厨柜': 2,
'回性': 2,
'众多': 2,
'回报': 2,
'定制化': 2,
'原料药': 2,
'仪器': 2,
'变更': 2,
'下降': 2,
'储存': 2,
'准确': 2,
'射频': 2,
'减少': 2,
'享有': 2,
'净额': 2,
'亿联': 2,
'差异': 2,
'凭证': 2,
'交易量': 2,
'交易性': 2,
'后续': 2,
'亚虹': 2,
'导致': 2,
'后台': 2,

'同比': 2,
'同期': 2,
'对价': 2,
'合力': 2,
'可收': 2,
'可变': 2,
'口径': 2,
'传感器': 2,
'会议': 2,
'宏观': 2,
'安徽': 2,
'安居宝': 2,
'上海': 2,
'抗菌药': 2,
'入账': 2,
'披露': 2,
'存续': 2,
'姚记': 2,
'光机电': 2,
'dematicgrou': 2,
'天味': 2,
'大恒': 2,
'制造': 2,
'大件': 2,
'持续': 2,
'剩余': 2,
'多种': 2,
'华菱': 2,
'处置': 2,
'外部': 2,
'p所': 2,
'封件': 2,

'r组件': 2,
'万向': 2,
'执行': 2,
'印制': 2,
'上年度': 2,
'完毕': 2,
'国中': 2,
'承租人': 2,
'列报': 2,
'投入': 2,
'单项': 2,
'公允': 2,
'其他应收款': 1,
'冰压机': 1,
'表现': 1,
'调节': 1,
'内蒙': 1,
'解痉药': 1,
'访问': 1,
'内酰': 1,
'全程': 1,
'内容': 1,
'评估师': 1,
'视觉': 1,
'全球': 1,
'充分性': 1,
'装运港': 1,
'先款': 1,
'冰鲜': 1,
'诊断': 1,
'视为': 1,
'关联方': 1,

'六大': 1,
'角色': 1,
'接单': 1,
'减震器': 1,
'制作': 1,
'自初': 1,
'创作': 1,
'能力': 1,
'胺类': 1,
'创新': 1,
'利润表': 1,
'聘请': 1,
'联盟': 1,
'联方': 1,
'到期': 1,
'联产': 1,
'编辑': 1,
'凭据': 1,
'制动': 1,
'制片': 1,
'维生素': 1,
'维修': 1,
'绩效': 1,
'统称': 1,
'剧集': 1,
'加工': 1,
'经购': 1,
'加权': 1,
'北斗': 1,
'列为': 1,
'自有': 1,
'自营': 1,

'自行': 1,
'血液': 1,
'融资': 1,
'融合': 1,
'融出': 1,
'蚕丝': 1,
'出版': 1,
'蓄能': 1,
'蒸汽': 1,
'营运': 1,
'出版物': 1,
'菲利华': 1,
'菌药': 1,
'击量': 1,
'分t': 1,
'分为': 1,
'分散': 1,
'若干': 1,
'分析': 1,
'船舷': 1,
'分部': 1,
'分配': 1,
'划分': 1,
'至少': 1,
'储能': 1,
'保税区': 1,
'做出': 1,
'铝合金': 1,
'三十八': 1,
'门到门': 1,
'长高电新': 1,
'三阶段': 1,

'上年末': 1,
'上期': 1,
'上涨': 1,
'销售密': 1,
'销售化': 1,
'不尽相同': 1,
'银泰': 1,
'铅蓄': 1,
'三十一': 1,
'铁路': 1,
'专家': 1,
'丝绸': 1,
'金徽': 1,
'为经': 1,
'主观': 1,
'酮类': 1,
'配电': 1,
'配合': 1,
'主观性': 1,
'书面': 1,
'三十五': 1,
'三十': 1,
'买入': 1,
'dematicgroup': 1,
'cip': 1,
'cnf': 1,
'黄金': 1,
'鱼微': 1,
'cpt': 1,
'dap': 1,
'高电': 1,
'高度': 1,

'ddp': 1,
'ddu': 1,
'飞机': 1,
'频繁': 1,
'院线': 1,
'领用': 1,
'领域': 1,
'exw': 1,
'fca': 1,
'fob': 1,
'group': 1,
'顺序': 1,
'poct': 1,
'非t': 1,
'零售': 1,
'一体化': 1,
'部品': 1,
'逻辑': 1,
'偏向': 1,
'购子': 1,
'超市': 1,
'休闲': 1,
'资租': 1,
'资源': 1,
'资本化': 1,
'资本': 1,
'众辰': 1,
'低于': 1,
'佣金': 1,
'供应链': 1,
'经接': 1,
'购买': 1,

'越过': 1,
'保证': 1,
'信号': 1,
'货方': 1,
'信宇人': 1,
'账龄': 1,
'修订': 1,
'账确': 1,
'借款': 1,
'账方式': 1,
'账户': 1,
'账务': 1,
'超过': 1,
'跨期': 1,
'通讯': 1,
'返售': 1,
'通知': 1,
'二十': 1,
'二十三': 1,
'选取': 1,
'送货': 1,
'追踪': 1,
'二十五': 1,
'二级': 1,
'云办公': 1,
'进口': 1,
'还款': 1,
'互联网': 1,
'车站': 1,
'运输交': 1,
'交互': 1,
'交易额': 1,

'运动': 1,
'运作': 1,
'输配电': 1,
'输入': 1,
'载明': 1,
'软硬件': 1,
'仓储': 1,
'仓库': 1,
'医院': 1,
'单笔': 1,
'经常性': 1,
'暴露': 1,
'年内': 1,
'年限': 1,
'机载': 1,
'并经': 1,
'广告位': 1,
'机密': 1,
'机器': 1,
'机制': 1,
'应计': 1,
'未计': 1,
'店铺': 1,
'木质': 1,
'建立': 1,
'建设': 1,
'建造': 1,
'期初': 1,
'开关': 1,
'开设': 1,
'服务费': 1,
'服务方': 1,

'引导': 1,
'当年': 1,
'有线': 1,
'有效性': 1,
'最佳': 1,
'权益': 1,
'平均': 1,
'带来': 1,
'款外': 1,
'实时': 1,
'永续': 1,
'水平': 1,
'审定': 1,
'宣传': 1,
'每月': 1,
'母公司': 1,
'家具': 1,
'正确': 1,
'家电': 1,
'款确': 1,
'家畜': 1,
'差额': 1,
'对账': 1,
'模块': 1,
'导航': 1,
'寿命': 1,
'根据线': 1,
'核心': 1,
'尚未': 1,
'尤其是': 1,
'层于': 1,
'展示': 1,

'差异性': 1,
'最为': 1,
'暂定': 1,
'十六': 1,
'形车': 1,
'抗真': 1,
'cif': 1,
'收融': 1,
'抗胆': 1,
'收益': 1,
'收方': 1,
'报关单': 1,
'报表日': 1,
'收入区': 1,
'抵押物': 1,
'抵销': 1,
'抽水': 1,
'担任': 1,
'播放': 1,
'摩托车': 1,
'摊销': 1,
'提存': 1,
'担保人': 1,
'拥有': 1,
'推迟': 1,
'按点': 1,
'损益': 1,
'授予': 1,
'接收方': 1,
'接口': 1,
'投资者': 1,
'放映': 1,

'投放': 1,
'无线': 1,
'智能化': 1,
'影院': 1,
'快速': 1,
'悬架': 1,
'情景': 1,
'时计': 1,
'成交': 1,
'时段': 1,
'成品油': 1,
'无需': 1,
'无误': 1,
'所在地': 1,
'承运人': 1,
'扑克牌': 1,
'新百': 1,
'执行性': 1,
'新华': 1,
'文件': 1,
'整体': 1,
'数量': 1,
'数智': 1,
'承诺': 1,
'数字': 1,
'敞口': 1,
'汇总': 1,
'实施': 1,
'定义': 1,
'沥青': 1,
'占期': 1,
'历次': 1,

'租售': 1,
'原则': 1,
'禽肉': 1,
'离岸': 1,
'离子': 1,
'票务': 1,
'碱药': 1,
'硬件': 1,
'参阅': 1,
'发出': 1,
'发动': 1,
'相比': 1,
'发放': 1,
'直营': 1,
'目的': 1,
'受偿': 1,
'监控': 1,
'受限': 1,
'皮肤': 1,
'变化': 1,
'各项': 1,
'合成': 1,
'合理性': 1,
'南京': 1,
'税费': 1,
'站到站': 1,
'精密': 1,
'华培': 1,
'华米': 1,
'线下': 1,
'纳入': 1,
'华闻': 1,

'红外': 1,
'累计': 1,
'索赔': 1,
'协助': 1,
'协议': 1,
'单个': 1,
'类型': 1,
'笔数': 1,
'类似': 1,
'管理者': 1,
'单位': 1,
'单据': 1,
'单时': 1,
'单核': 1,
'签署': 1,
'授权': 1,
'签字': 1,
'策略': 1,
'第四': 1,
'电动车': 1,
'电动': 1,
'后货': 1,
'海关': 1,
'境外线': 1,
'潍柴': 1,
'增材': 1,
'增长率': 1,
'复杂性': 1,
'港口': 1,
'外线': 1,
'涵盖': 1,
'多且': 1,

'妆品': 1,
'始确': 1,
'浙江': 1,
'激励': 1,
'存续期': 1,
'流贴': 1,
'季度': 1,
'流动': 1,
'活鸡': 1,
'活动': 1,
'洛耐': 1,
'安装': 1,
'注册量': 1,
'完工': 1,
'完整': 1,
'境内线': 1,
'垫款': 1,
'田野': 1,
'喹啉': 1,
'含有': 1,
'用于': 1,
'生猪': 1,
'生物质': 1,
'周期': 1,
'生态': 1,
'生命': 1,
'品牌': 1,
'售价': 1,
'商官网': 1,
'商平台': 1,
'现法': 1,
'热电': 1,

```
 '四十三': 1,
 '四十四': 1,
 '回单': 1,
 '因子': 1,
 '国产': 1,
 '国际': 1,
 '地区': 1,
 '地并': 1,
 '版块': 1,
 '煤炭': 1,
 '焊割': 1,
 'cfr': 1}
```

open source font: https://fonts.google.com/

```
wordcloud = WordCloud(font_path="data/cangerr03-W03.ttf",
                      background_color="white", max_words=100,
                  max_font_size=250, random_state=42, width=1200,
                      height=600,
↪   margin=2).generate_from_frequencies(worddic)
```

```
import matplotlib.pyplot as plt
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

```
wordcloud = WordCloud(font_path="data/MaShanZheng-Regular.ttf",
                      background_color="white", max_words=100,
                    max_font_size=250, random_state=42, width=1200,
                      height=600,
 ↪  margin=2).generate_from_frequencies(worddic)
```

```
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
```

7.7 词表占比

```
total = wcdf.groupby("stkcd").N.sum().reset_index().rename(columns =
 ↪  {"N":"total"})
total.head()
```

	stkcd	total
0	166	73
1	338	96
2	403	80
3	417	89

	stkcd	total
4	559	50

```
# define your own word list
wordlist = [" 收入"]
```

```
wordlistN =
 ↪  wcdf[wcdf.word.isin(wordlist)].groupby("stkcd").N.sum().reset_index().rename(colu
 ↪  = {"N":"listN"})
wordlistN
```

	stkcd	listN
0	403	7
1	417	1
2	719	9
3	799	9
4	838	6
...	...	...
77	688819	4
78	830832	4
79	832023	4
80	838262	4
81	871396	5

```
wordlistN = total.merge(wordlistN, how="left", on = "stkcd").fillna(0)
wordlistN
```

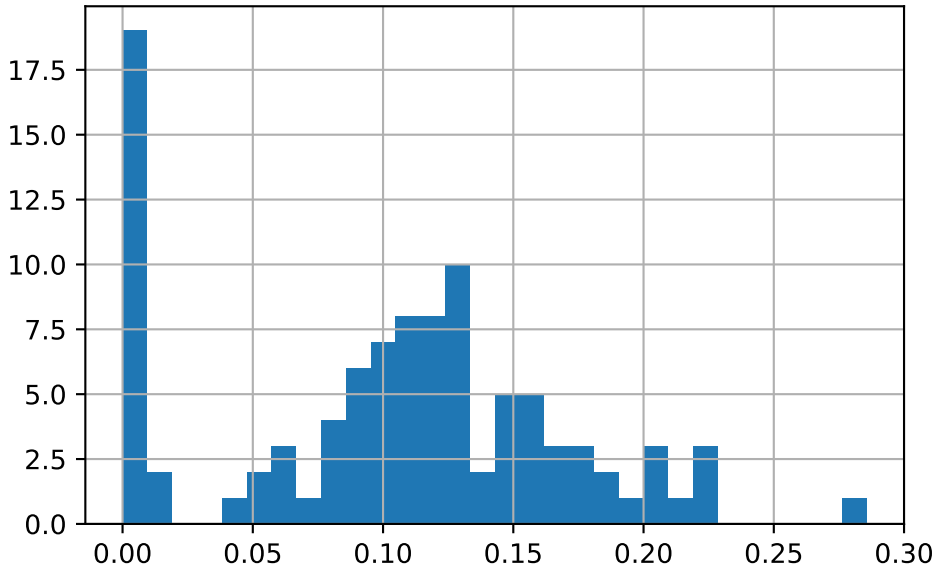
	stkcd	total	listN
0	166	73	0.0

	stkcd	total	listN
1	338	96	0.0
2	403	80	7.0
3	417	89	1.0
4	559	50	0.0
...	...	...	...
95	830832	27	4.0
96	832023	24	4.0
97	838262	33	4.0
98	871263	39	0.0
99	871396	53	5.0

```
wordlistN['wordrate'] = wordlistN["listN"]/wordlistN["total"]
wordlistN.head()
```

	stkcd	total	listN	wordrate
0	166	73	0.0	0.000000
1	338	96	0.0	0.000000
2	403	80	7.0	0.087500
3	417	89	1.0	0.011236
4	559	50	0.0	0.000000

```
wordlistN.wordrate.hist(bins = 30)
```




```
wordlistN.to_excel("data/wordlist_ratio.xlsx", index = False)
```

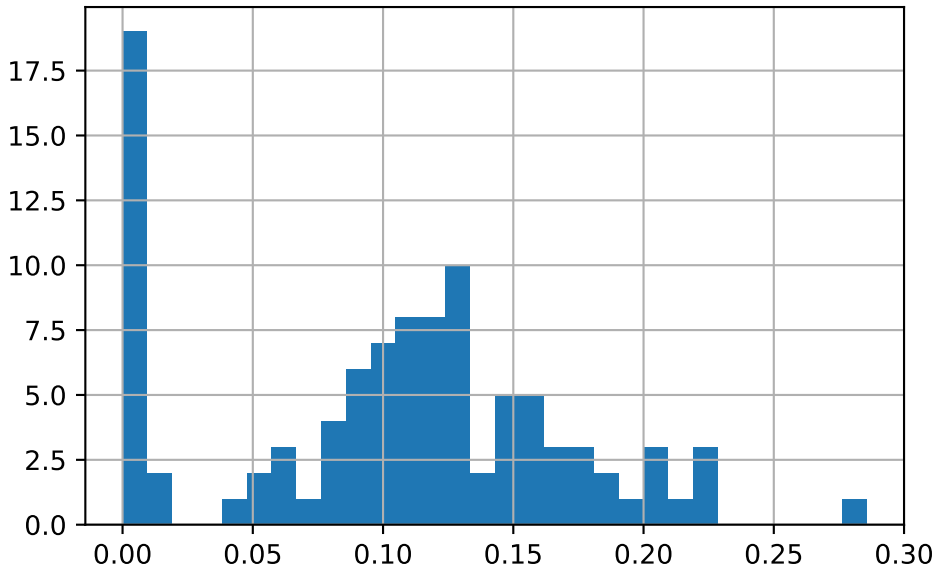
```
wordlistN = wordlistN.assign(sentiment =
↪  wordlistN["listN"]/wordlistN["total"])
wordlistN.query("sentiment < 0")
```

stkcd	total	listN	wordrate	sentiment

```
wordlistN.sentiment.describe()
```

```
count    100.000000
mean       0.102567
std        0.066533
min        0.000000
25%        0.061800
50%        0.110243
75%        0.148148
max        0.285714
Name: sentiment, dtype: float64
```

```
wordlistN.sentiment.hist(bins = 30)
```



7.8 相似度

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
xx = cosine_similarity(word_tfidf_vector)
```

```
xx.shape
```

```
(100, 100)
```

```
pd.DataFrame(xx, columns =  df.stkcd,
                            index = df.stkcd)
```

stkcd	600977	830832	719	300706	301421	688553	799	603070	3012
stkcd									
600977	1.000000	0.048218	0.125220	0.004306	0.073676	0.063892	0.132036	0.087896	0.06

stkcd stkcd	600977	830832	719	300706	301421	688553	799	603070	3012
830832	0.048218	1.000000	0.130587	0.000000	0.060503	0.072498	0.230335	0.086199	0.05
719	0.125220	0.130587	1.000000	0.010078	0.083489	0.105940	0.390918	0.151922	0.09
300706	0.004306	0.000000	0.010078	1.000000	0.003765	0.023940	0.025468	0.032614	0.02
301421	0.073676	0.060503	0.083489	0.003765	1.000000	0.102491	0.224208	0.116393	0.09
...	...	...	...	...	...	...	...	...	...
2853	0.135564	0.234815	0.480994	0.024555	0.158285	0.206910	0.707412	0.269446	0.18
338	0.006073	0.003409	0.008334	0.117279	0.007342	0.023701	0.030085	0.028313	0.02
603180	0.092044	0.108199	0.213851	0.009752	0.104312	0.136175	0.258173	0.160411	0.11
603056	0.120057	0.059891	0.111101	0.009871	0.037954	0.042566	0.122016	0.062185	0.03
793	0.048434	0.000000	0.004514	0.076536	0.008403	0.019549	0.012863	0.025381	0.01

```
simi_pair = pd.DataFrame(xx, columns =  df.stkcd, index =
 ↪  df.stkcd).stack()
simi_pair
```

```
stkcd   stkcd
600977  600977    1.000000
        830832    0.048218
        719       0.125220
        300706    0.004306
        301421    0.073676
                     ...
793     2853      0.010998
        338       0.125898
        603180    0.003733
        603056    0.010922
        793       1.000000
Length: 10000, dtype: float64
```

```
simi_pair.index.names = ['stkcd_x', 'stkcd_y']
simi_pair.reset_index().rename(columns = {0: "similarity"})
```

	stkcd_x	stkcd_y	similarity
0	600977	600977	1.000000
1	600977	830832	0.048218
2	600977	719	0.125220
3	600977	300706	0.004306
4	600977	301421	0.073676
...	...	...	...
9995	793	2853	0.010998
9996	793	338	0.125898
9997	793	603180	0.003733
9998	793	603056	0.010922
9999	793	793	1.000000

8 可视化

点击下载本章节代码

8.1 Matplotlib 数据可视化

Matplotlib 模块是 Python 最基础的可视化库，可以绘制多种形式的图形，包括线图、直方图、饼状图、散点图、误差线图等等。在我们的工作和学习中，将数据图形化展示是一项非常重要的任务，可以让数据更加直观。

Matplotlib 最重要的特性之一就是具有良好的操作系统兼容性和图形显示底层接口兼容性。

```
%pip install matplotlib pandas numpy seaborn plotly --upgrade
```

```
import matplotlib.pyplot as plt
from matplotlib import animation
import pandas as pd
import numpy as np
import math
```

```
t = np.linspace(0, math.pi, 1000)
x = np.sin(t)
y = np.cos(t) + np.power(x, 2.0/3)
plt.plot(x, y, color='red', linewidth=2, label='h')
plt.plot(-x, y, color='red', linewidth=2, label='-h')
plt.xlabel('t')
plt.ylabel('h')
```

```
plt.ylim(-2, 2)
plt.xlim(-2, 2)


# plt.show()
```

8.1.1 设置绘图样式

使用 `plt.style` 来选择图形的绘图风格。

> **Matplotlib 样式说明**
>
> 从 Matplotlib 3.6 版本开始，`seaborn` 样式已被移除。可以使用以下替代方案：
>
> - `seaborn-v0_8` - 使用旧版 seaborn 样式
> - `default` - Matplotlib 默认样式
> - `ggplot` - ggplot2 风格
> - `bmh` - Bayesian Methods for Hackers 样式

```
# 查看所有可用样式
print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bm
```

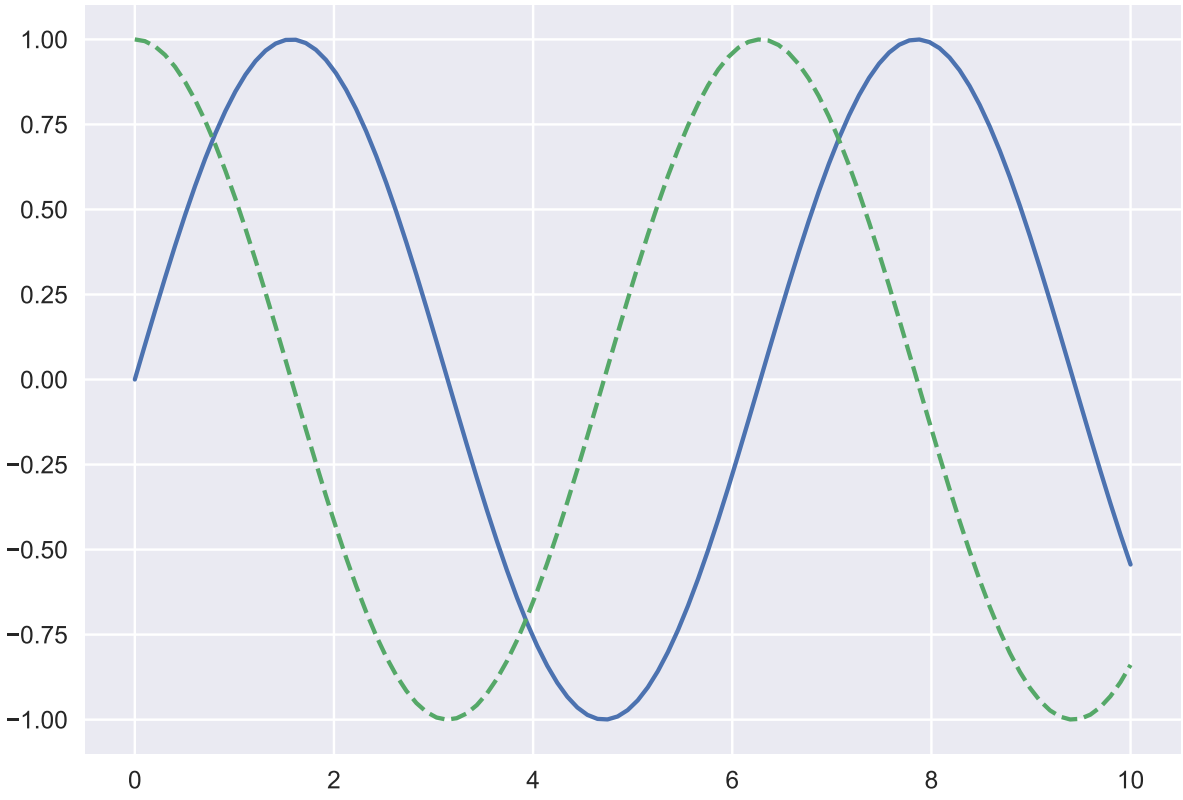
```
# 使用 seaborn-v0_8 样式（兼容旧代码）
plt.style.use('seaborn-v0_8')
```

8.1.2 关于 plt.show()

如果你在一个脚本文件中使用 Matplotlib，那么显示图形的时候必须使用 `plt.show()`，而在 notebook 中可以不要，特别注意在 notebook 中画图：

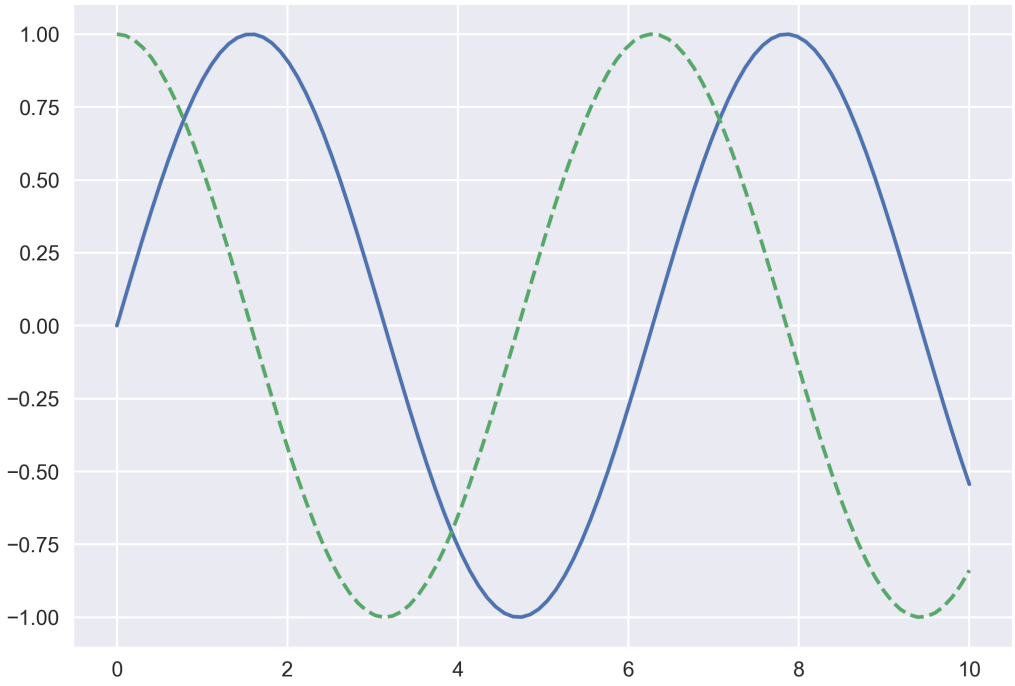
- `%matplotlib notebook` 会在 Notebook 中启动交互式图形。
- `%matplotlib inline` 会在 Notebook 中启动静态图形。

```
import numpy as np
x = np.linspace(0, 10, 100)
fig = plt.figure()
plt.plot(x, np.sin(x), '-')
plt.plot(x, np.cos(x), '--')
```



```
fig.savefig('my_figure.png')
```

```
from IPython.display import Image
Image('my_figure.png')
```



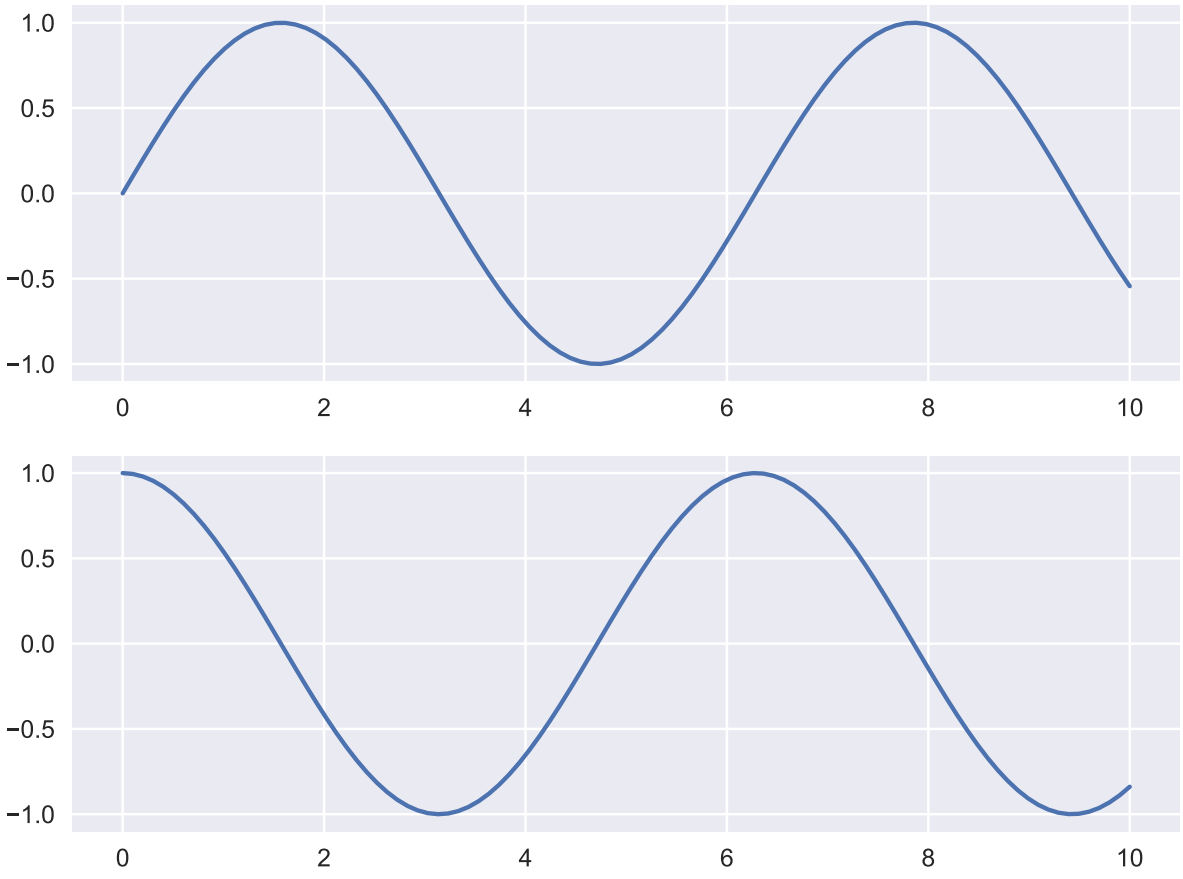

```
fig.savefig('my_figure.pdf')
```

```
fig.canvas.get_supported_filetypes()
```

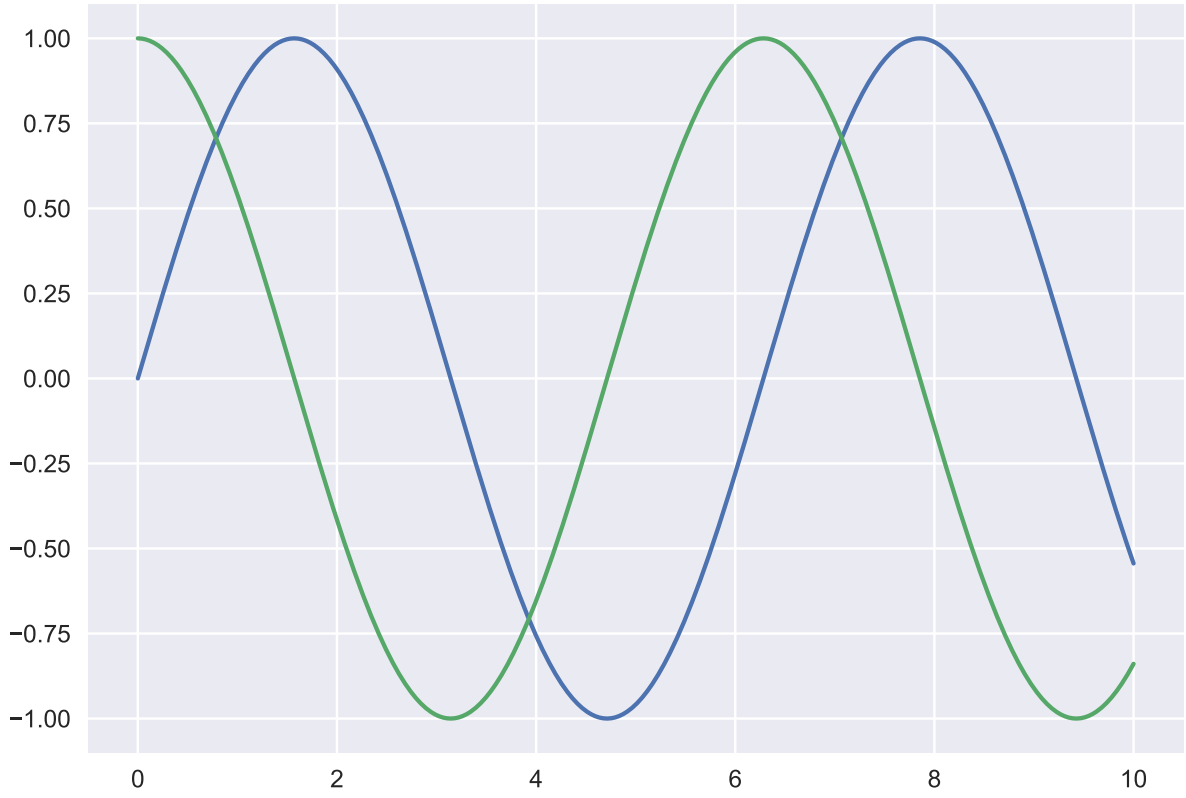
```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
```

```
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format',
 'webp': 'WebP Image Format'}
```

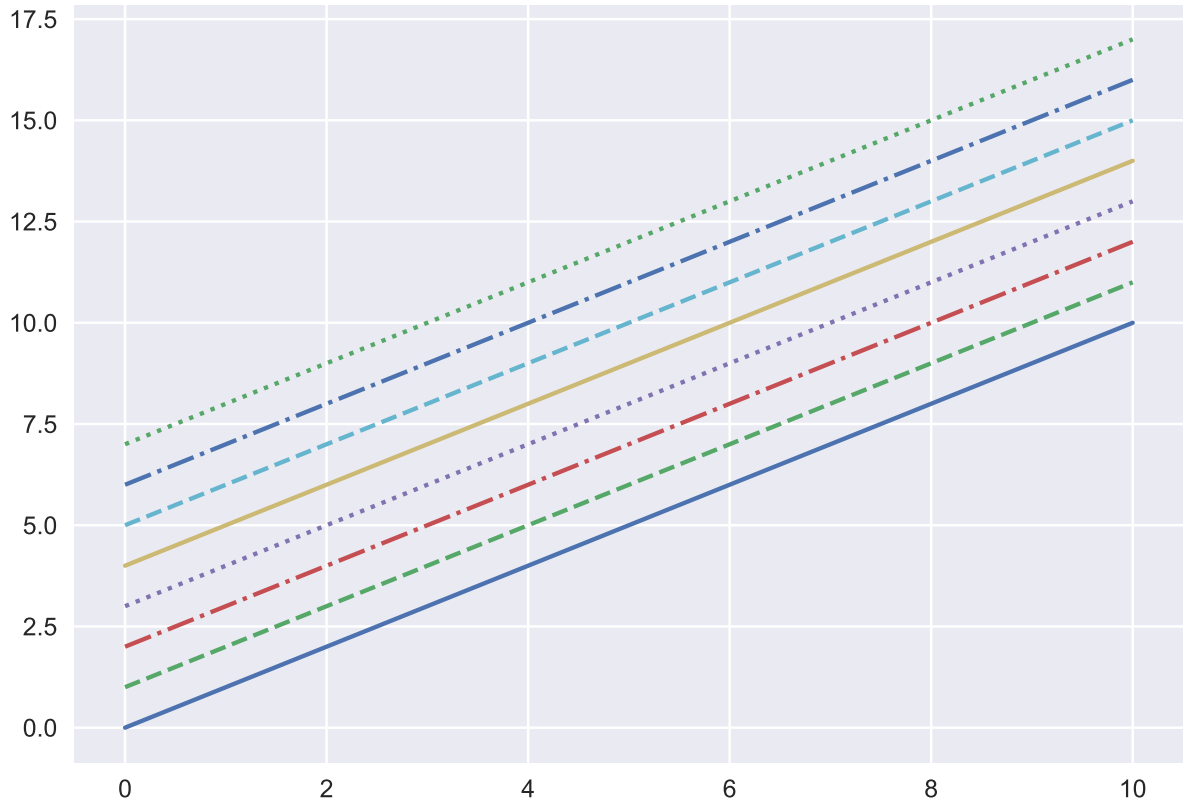
```
plt.figure(figsize= (8,6)) # 创建图形
# 创建两个子图中的第一个，设置坐标轴
plt.subplot(2, 1, 1) # （行、列、子图编号）
plt.plot(x, np.sin(x))
# 创建两个子图中的第二个，设置坐标轴
plt.subplot(2, 1, 2)
plt.plot(x, np.cos(x));
```



```
fig = plt.figure()
ax = plt.axes()
x = np.linspace(0, 10, 1000)
ax.plot(x, np.sin(x))
ax.plot(x, np.cos(x))
```

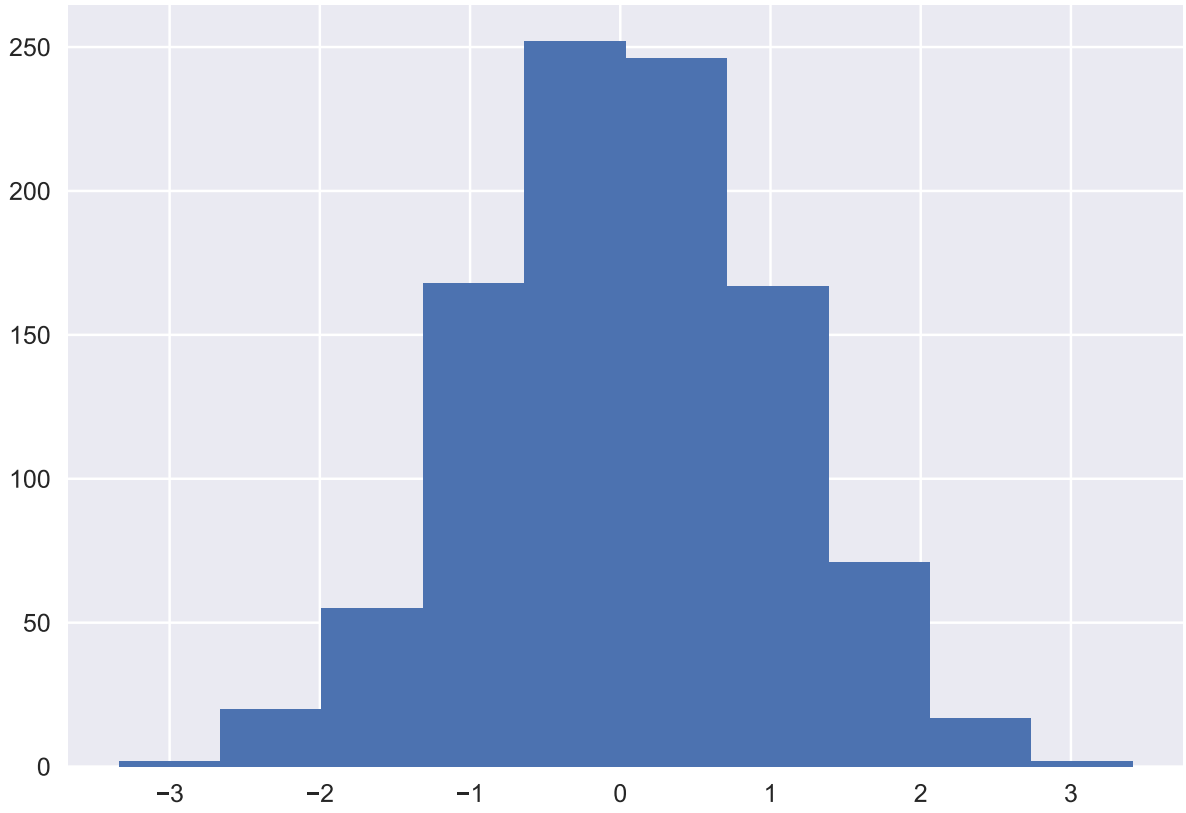



```
plt.plot(x, x + 0, linestyle='solid')
plt.plot(x, x + 1, linestyle='dashed')
plt.plot(x, x + 2, linestyle='dashdot')
plt.plot(x, x + 3, linestyle='dotted');
# 你可以用下面的简写形式
plt.plot(x, x + 4, linestyle='-') # 实线
plt.plot(x, x + 5, linestyle='--') # 虚线
plt.plot(x, x + 6, linestyle='-.') # 点划线
plt.plot(x, x + 7, linestyle=':'); # 实点线
```




```
import numpy as np
import matplotlib.pyplot as plt
data = np.random.randn(1000)
plt.hist(data)
```

```
(array([  2.,   20.,   55., 168., 252., 246., 167.,   71.,   17.,    2.]),
 array([-3.33782825, -2.66315348, -1.98847871, -1.31380394, -0.63912918,
         0.03554559,  0.71022036,  1.38489513,  2.0595699 ,  2.73424466,
         3.40891943]),
 <BarContainer object of 10 artists>)
```



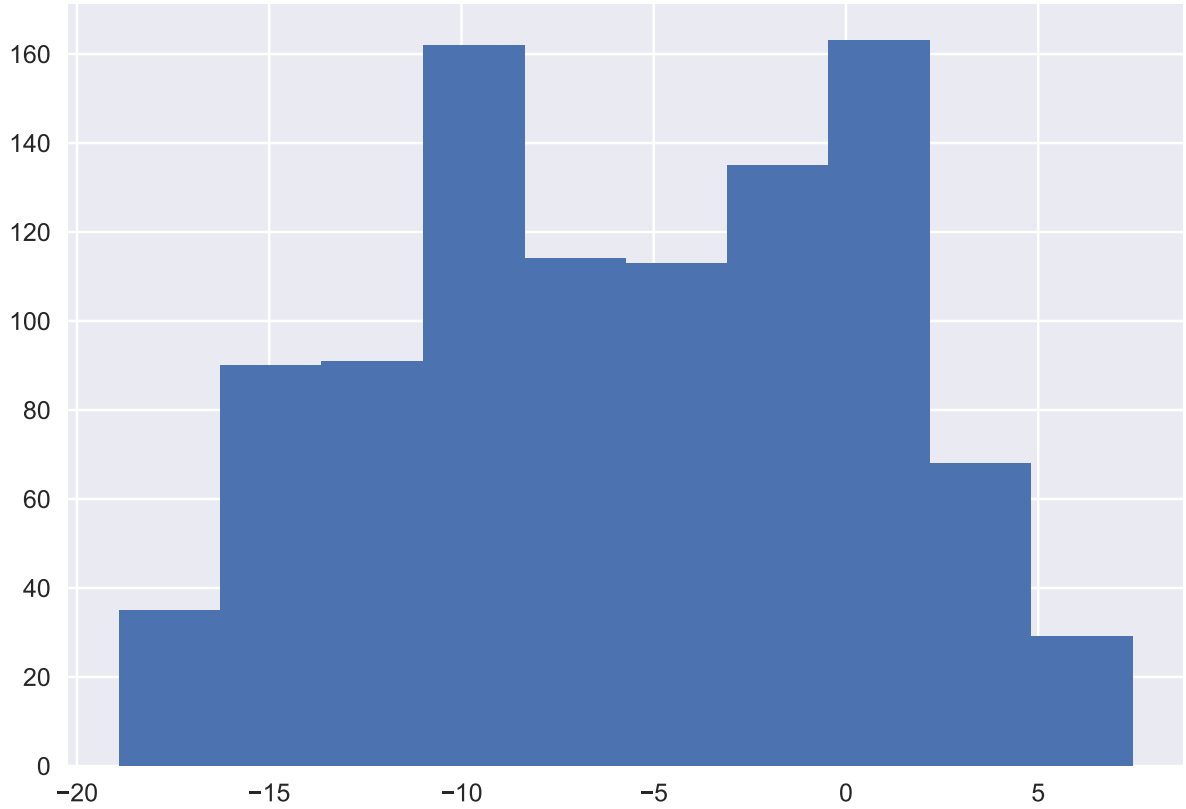
8.2 Pandas plot

```
df = pd.DataFrame(np.random.randn(1000, 4),
                  index=pd.date_range('1/1/2000', periods=1000),
                  columns=list('ABCD'))

df = df.cumsum()
df.head()
```

	A	B	C	D
2000-01-01	0.201344	-0.077865	-0.551789	-0.278203
2000-01-02	-0.472940	-1.402981	-1.647169	-1.118338
2000-01-03	-1.396229	-1.398226	-0.723251	-1.250755

	A	B	C	D
2000-01-04	-2.445071	-0.838639	-0.386590	0.097072
2000-01-05	-3.407625	-1.322600	-0.804589	-1.288955

```
df.A.hist()
```




```
df["gp"] =  df.A // 15
```

```
df.gp.drop_duplicates()
```

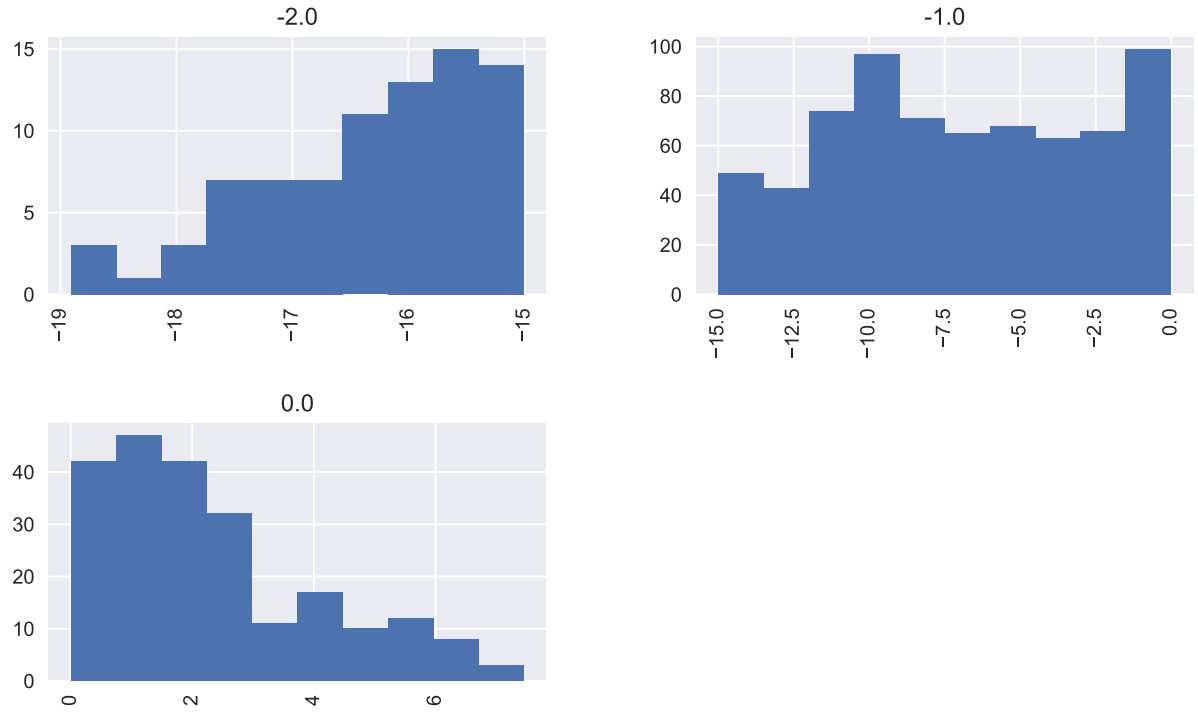
```
2000-01-01    0.0
2000-01-02   -1.0
2000-07-09   -2.0
Name: gp, dtype: float64
```

```
df.head()
```

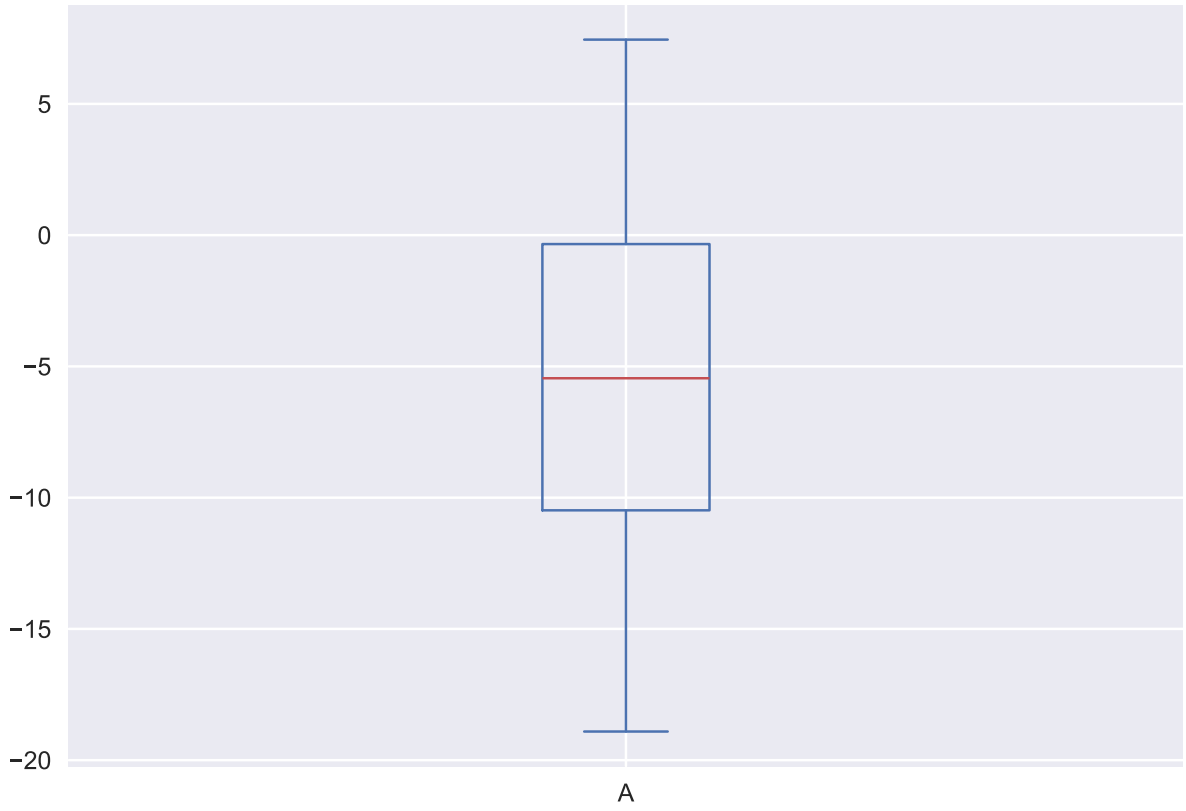
	A	B	C	D	gp
2000-01-01	0.201344	-0.077865	-0.551789	-0.278203	0.0
2000-01-02	-0.472940	-1.402981	-1.647169	-1.118338	-1.0
2000-01-03	-1.396229	-1.398226	-0.723251	-1.250755	-1.0
2000-01-04	-2.445071	-0.838639	-0.386590	0.097072	-1.0
2000-01-05	-3.407625	-1.322600	-0.804589	-1.288955	-1.0

```
df.A.hist(by= df.gp, figsize=(10, 6))
```

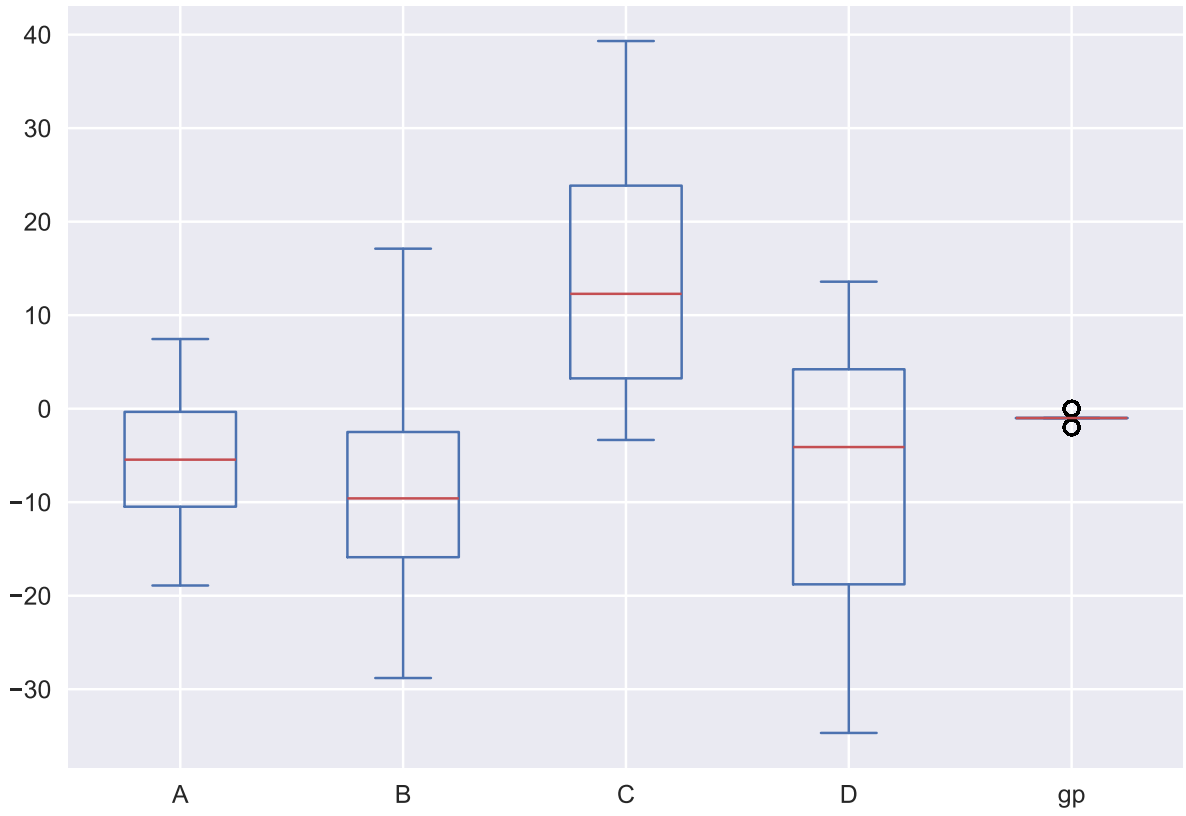
```
array([[<Axes: title={'center': '-2.0'}>,
        <Axes: title={'center': '-1.0'}>],
       [<Axes: title={'center': '0.0'}>, <Axes: >]], dtype=object)
```



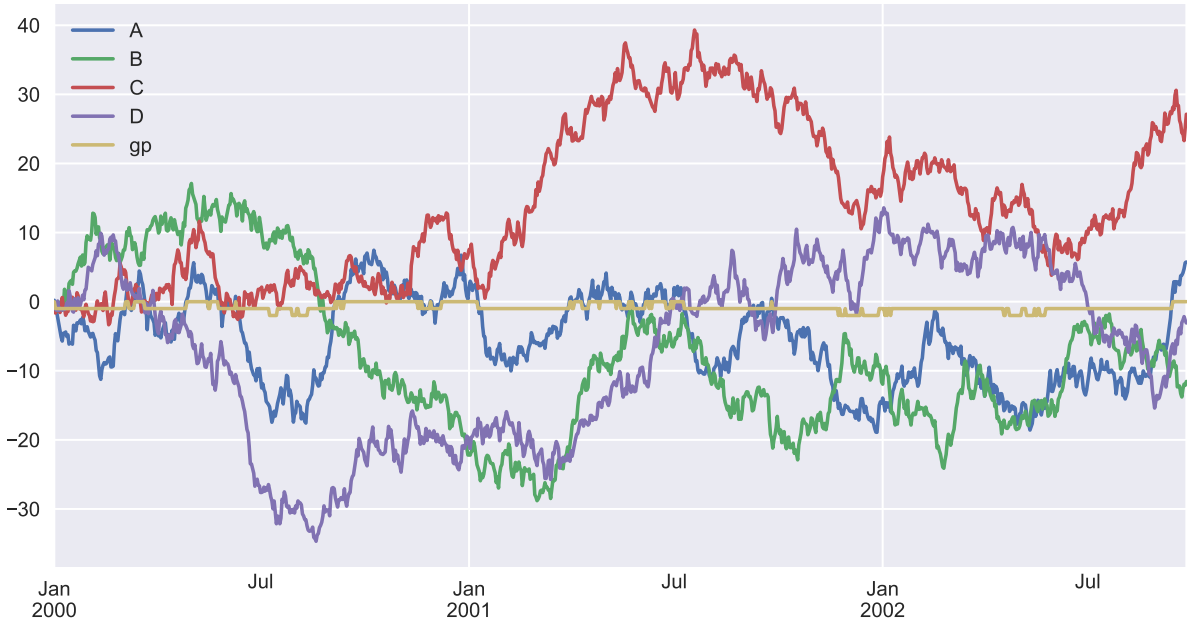

```
df.A.plot(kind='box')
```




```
df.plot(kind='box')
```



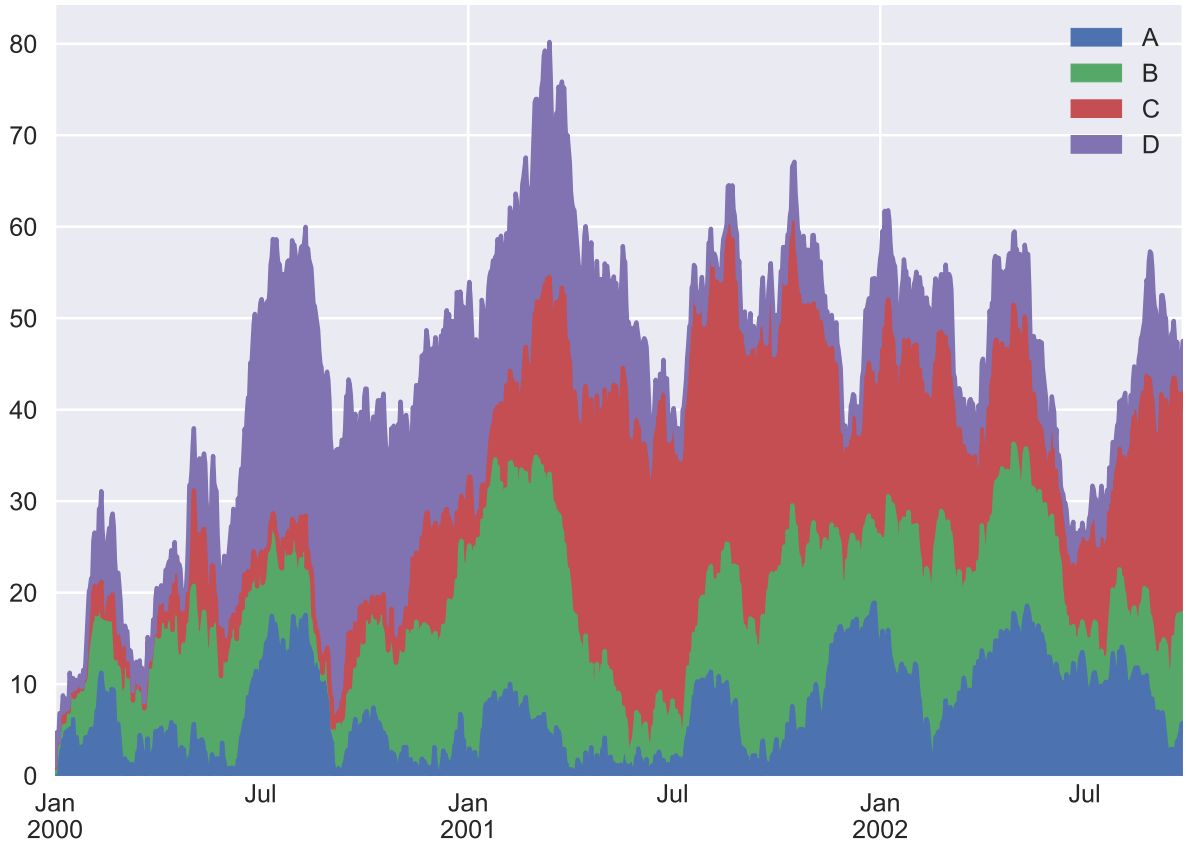
```
df.plot(figsize= (10,5))
```



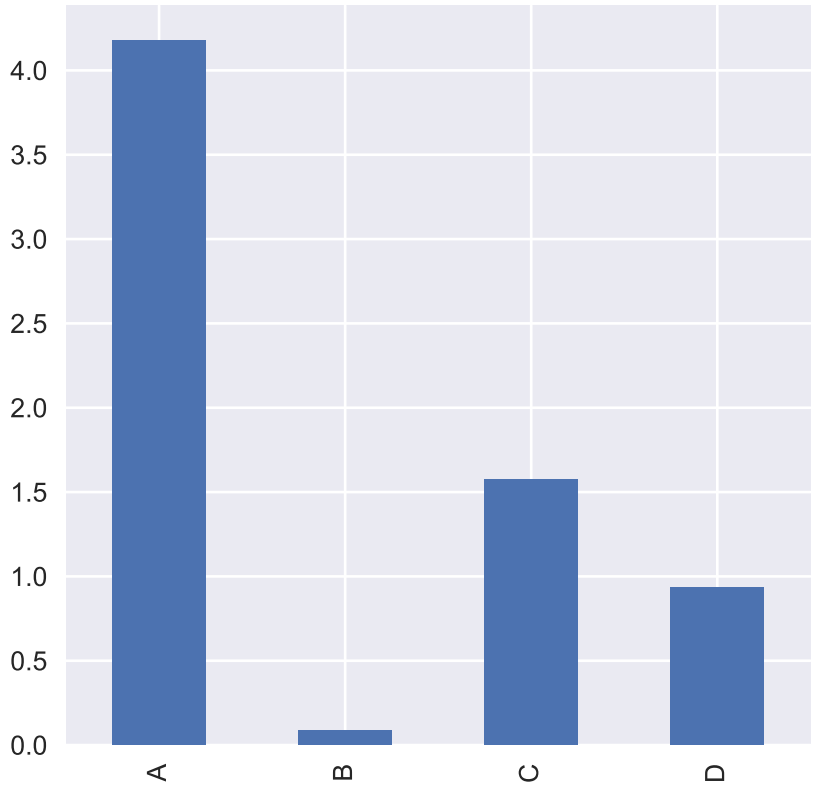

```
df = abs(df)
```

```
del df["gp"]
```

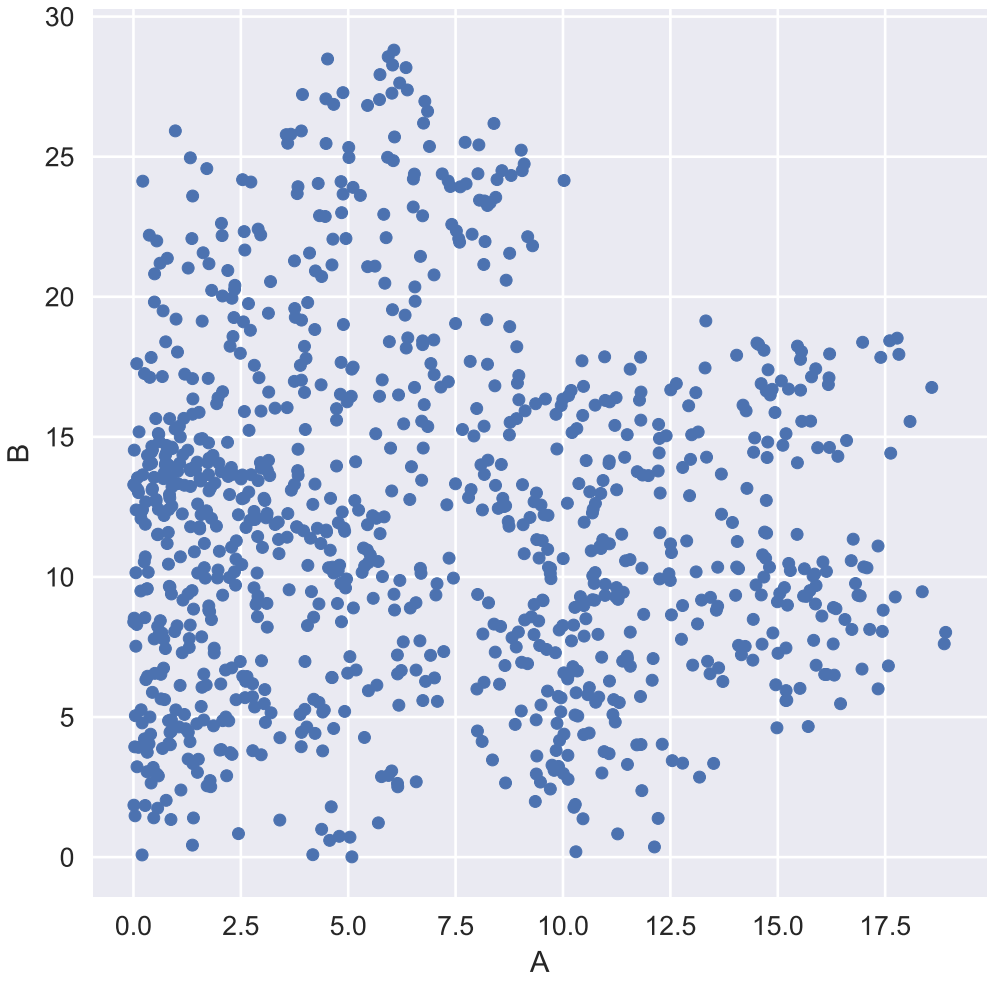
```
df.plot.area()
```



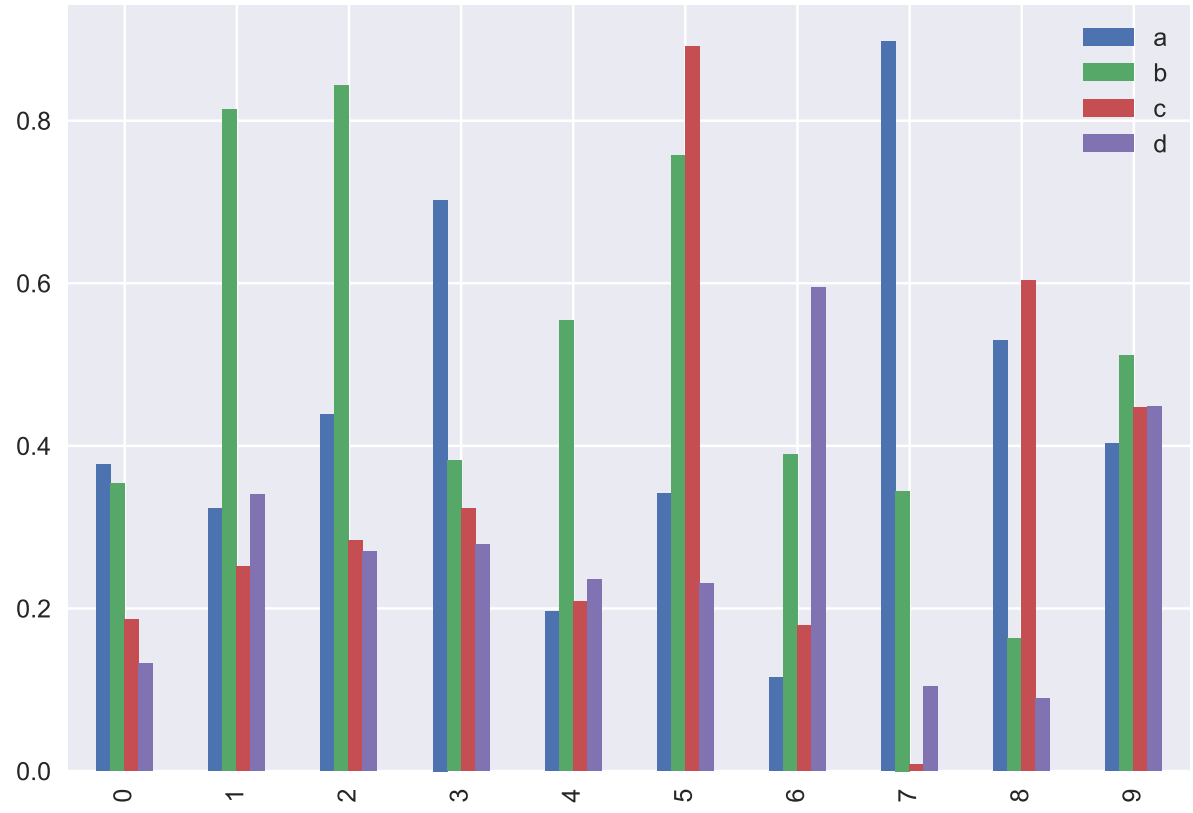

```
df.iloc[5].plot(kind='bar', figsize= (5,5))
```



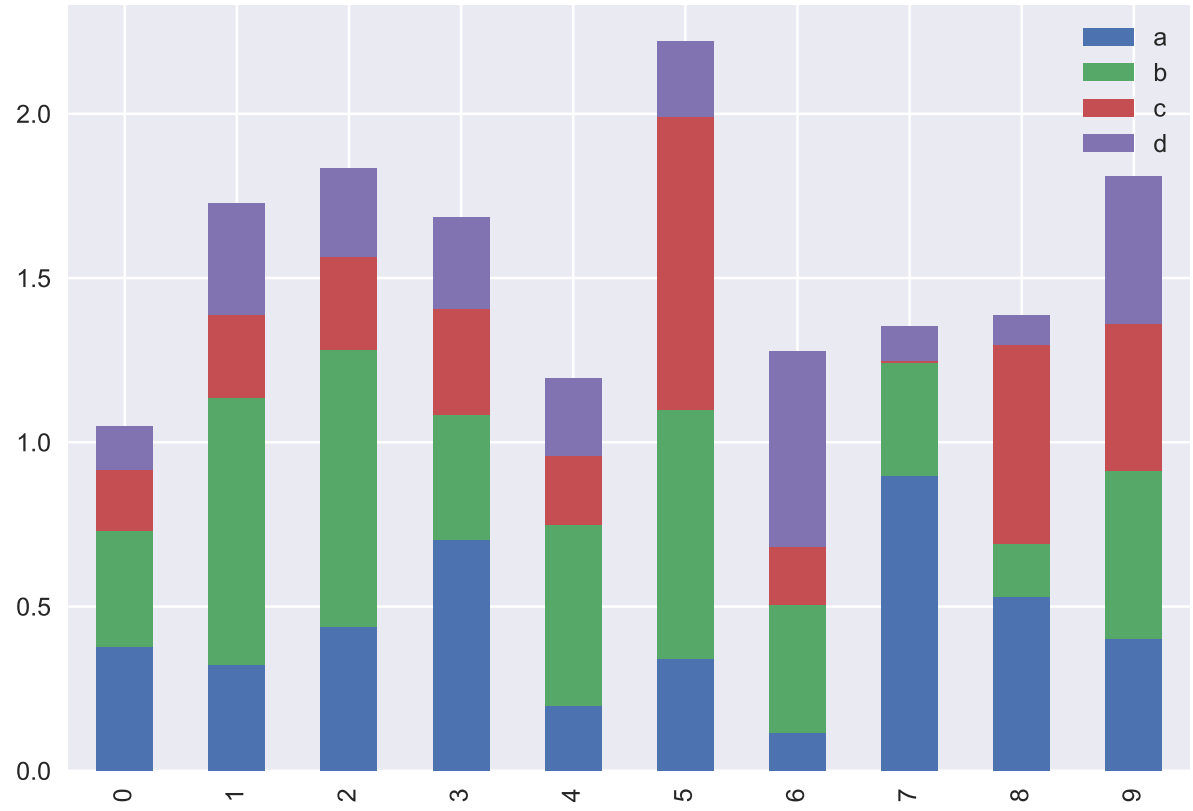
```
df.plot.scatter(x = "A", y = "B", figsize= (6,6))
```



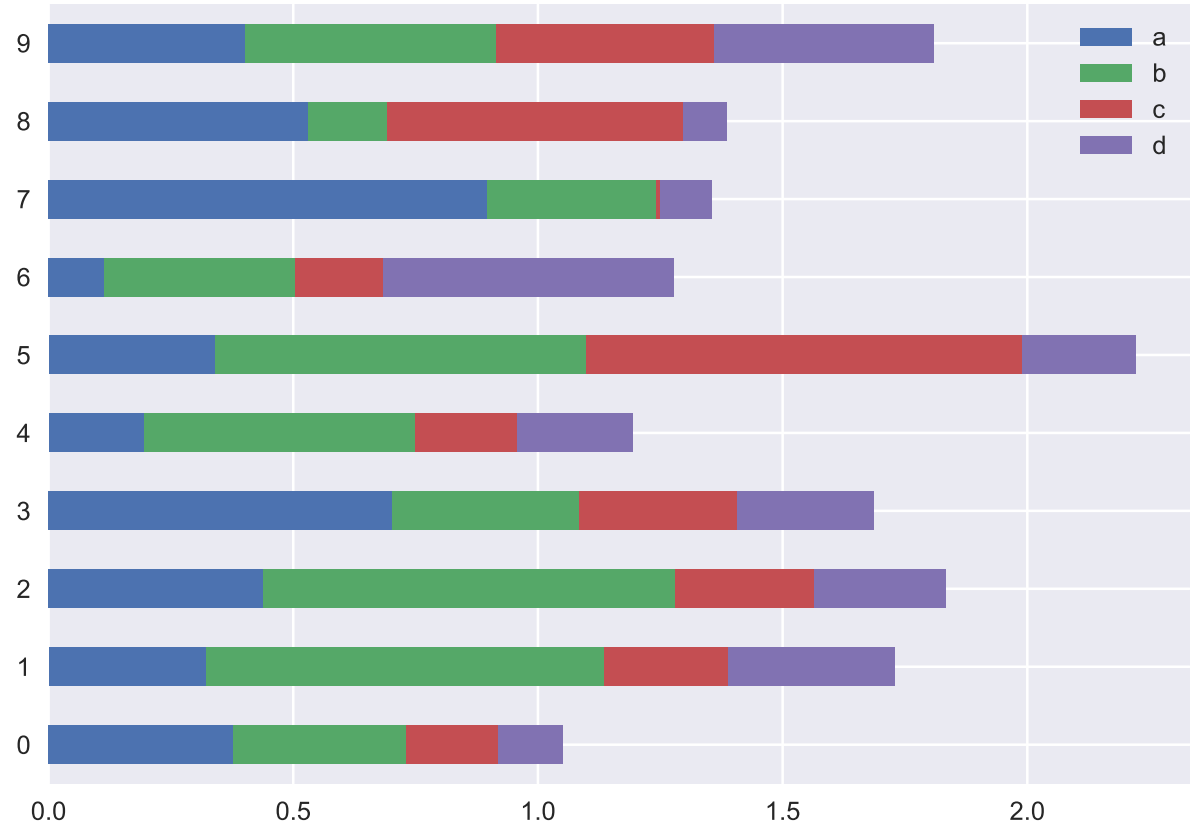

```
df2 = pd.DataFrame(np.random.rand(10, 4), columns=['a', 'b', 'c', 'd'])
df2.plot.bar()
```




```
df2.plot.bar(stacked=True)
```

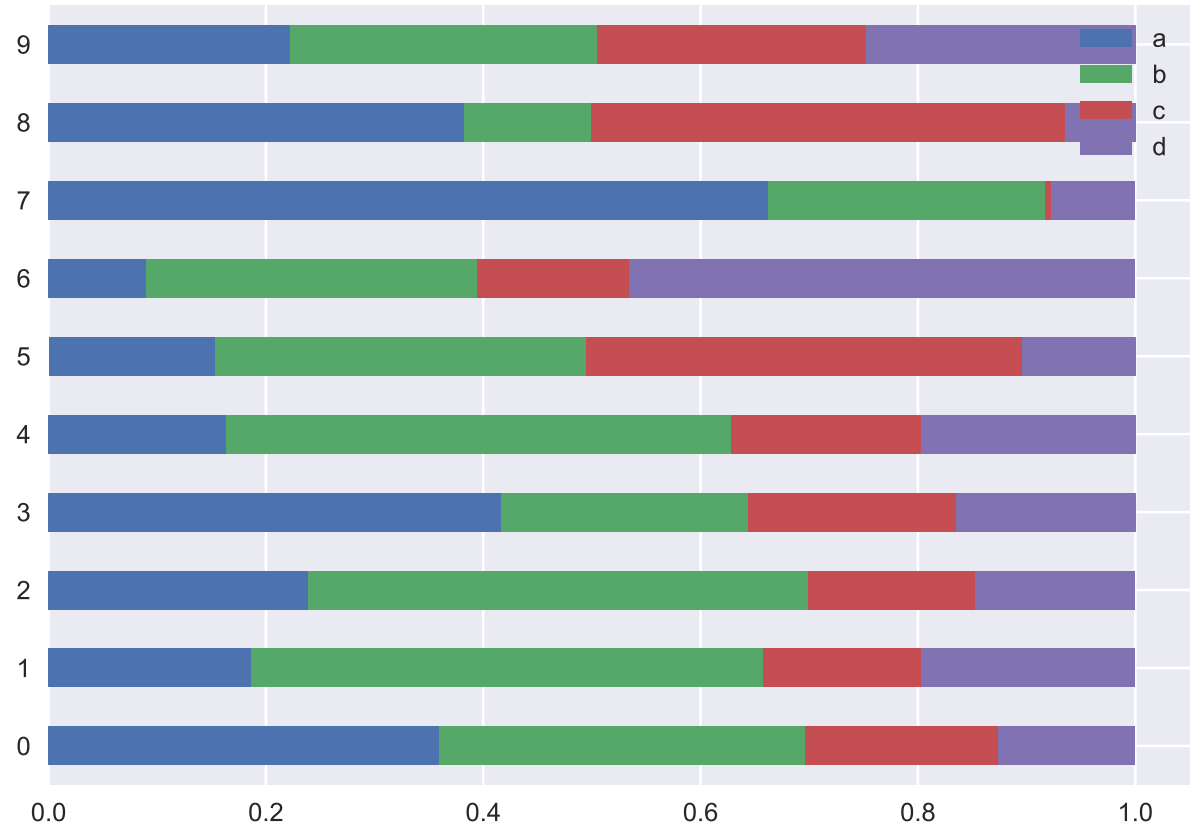


```
df2.plot.barh(stacked=True)
```




```
df2 = df2.div(df2.sum("columns"), axis = "rows")
```

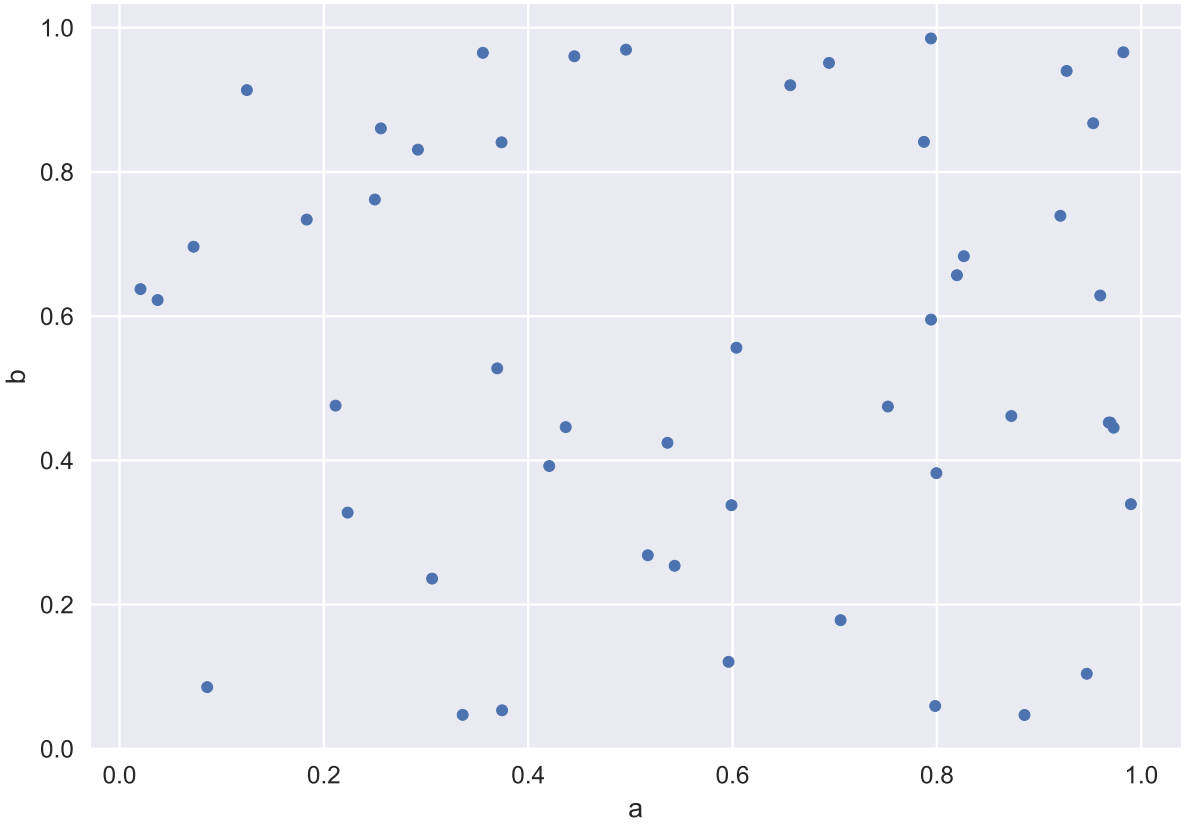
```
df2.plot.barh(stacked=True)
```



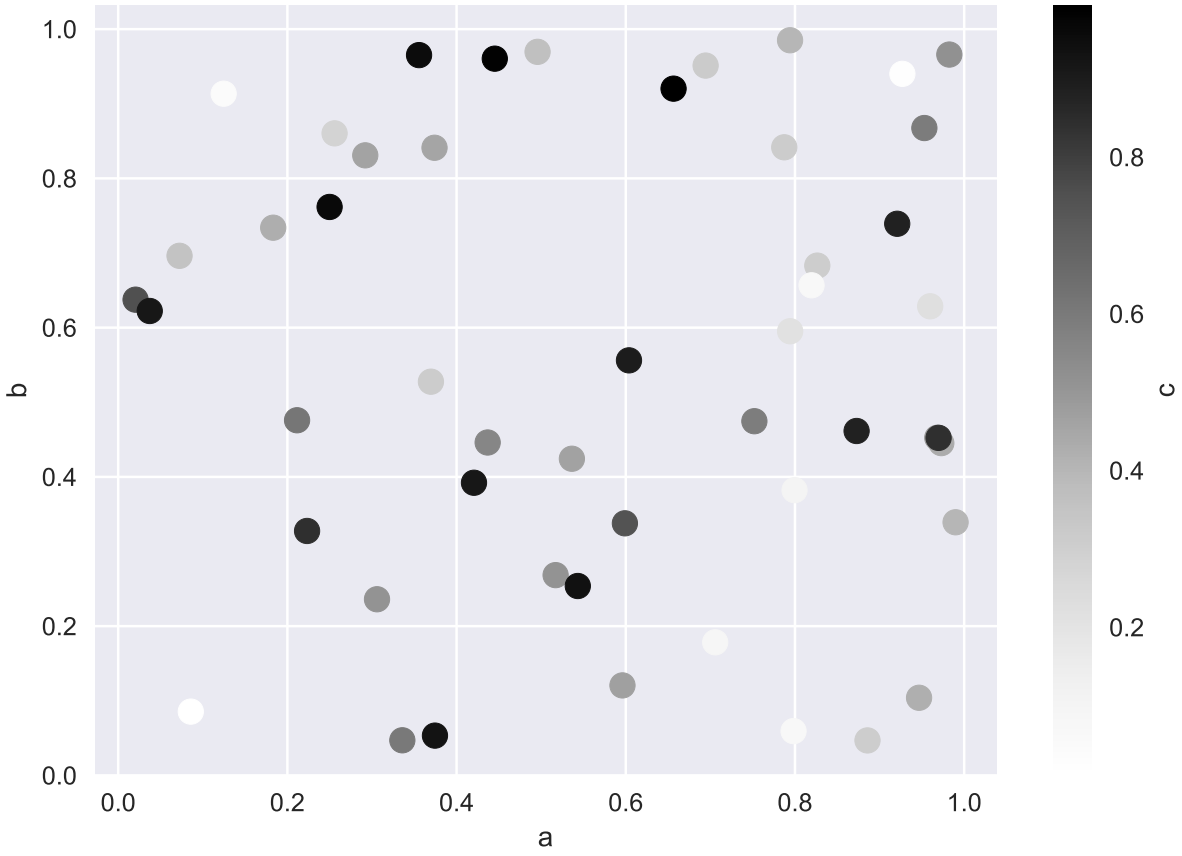

```
df = pd.DataFrame(np.random.rand(50, 4), columns=['a', 'b', 'c', 'd'])
df.head()
```

	a	b	c	d
0	0.656485	0.920260	0.993871	0.065119
1	0.305955	0.236032	0.513419	0.289175
2	0.959830	0.628607	0.228175	0.735164
3	0.752016	0.474612	0.588254	0.899860
4	0.124703	0.913492	0.048444	0.969139

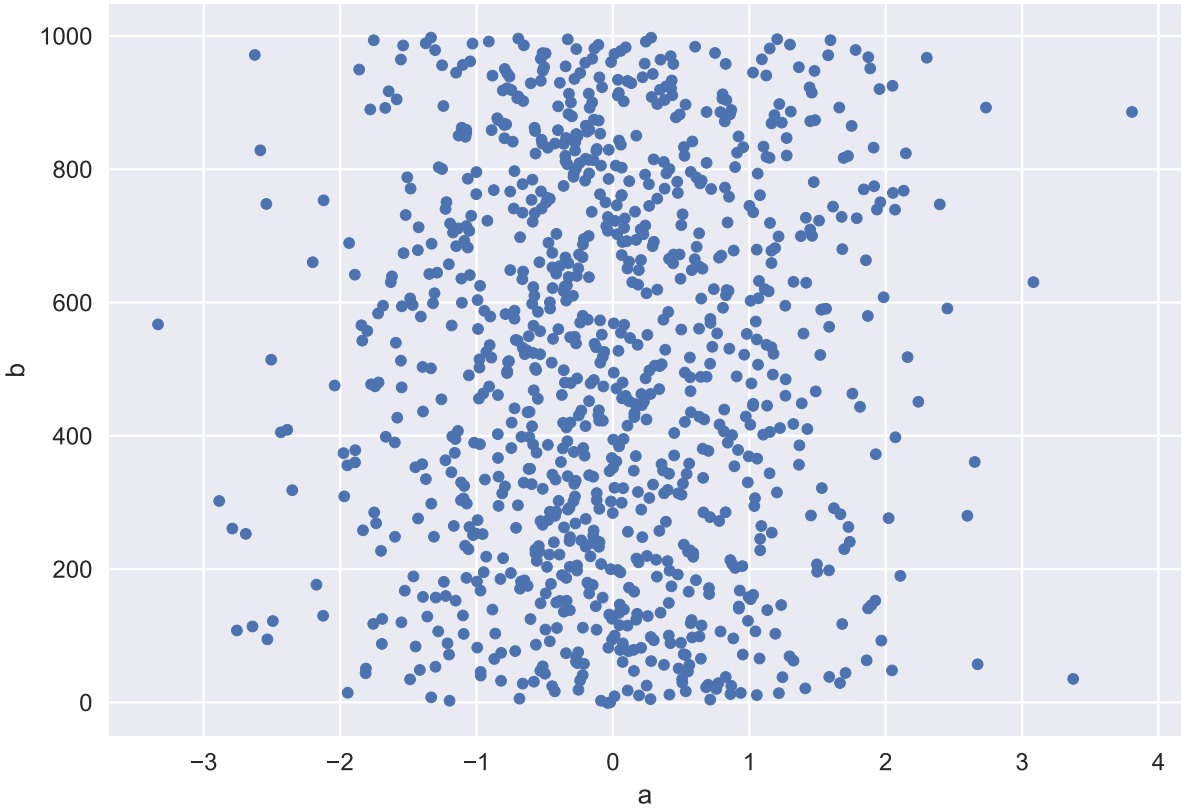
```
df.plot.scatter(x='a', y='b')
```



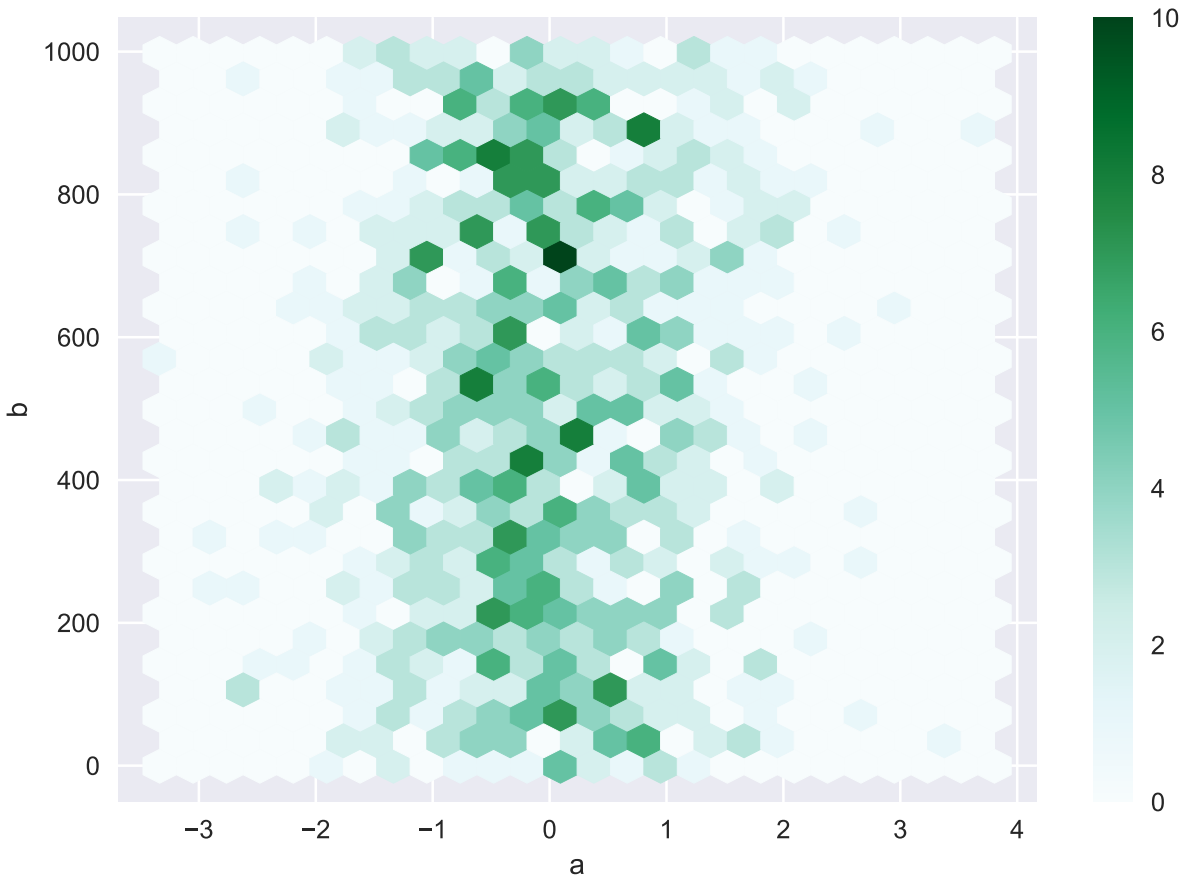

```
df.plot.scatter(x='a', y='b', c='c', s=100)
```




```
df = pd.DataFrame(np.random.randn(1000, 2), columns=['a', 'b'])

df['b'] = df['b'] + np.arange(1000)

df.plot.scatter(x='a', y='b')
```

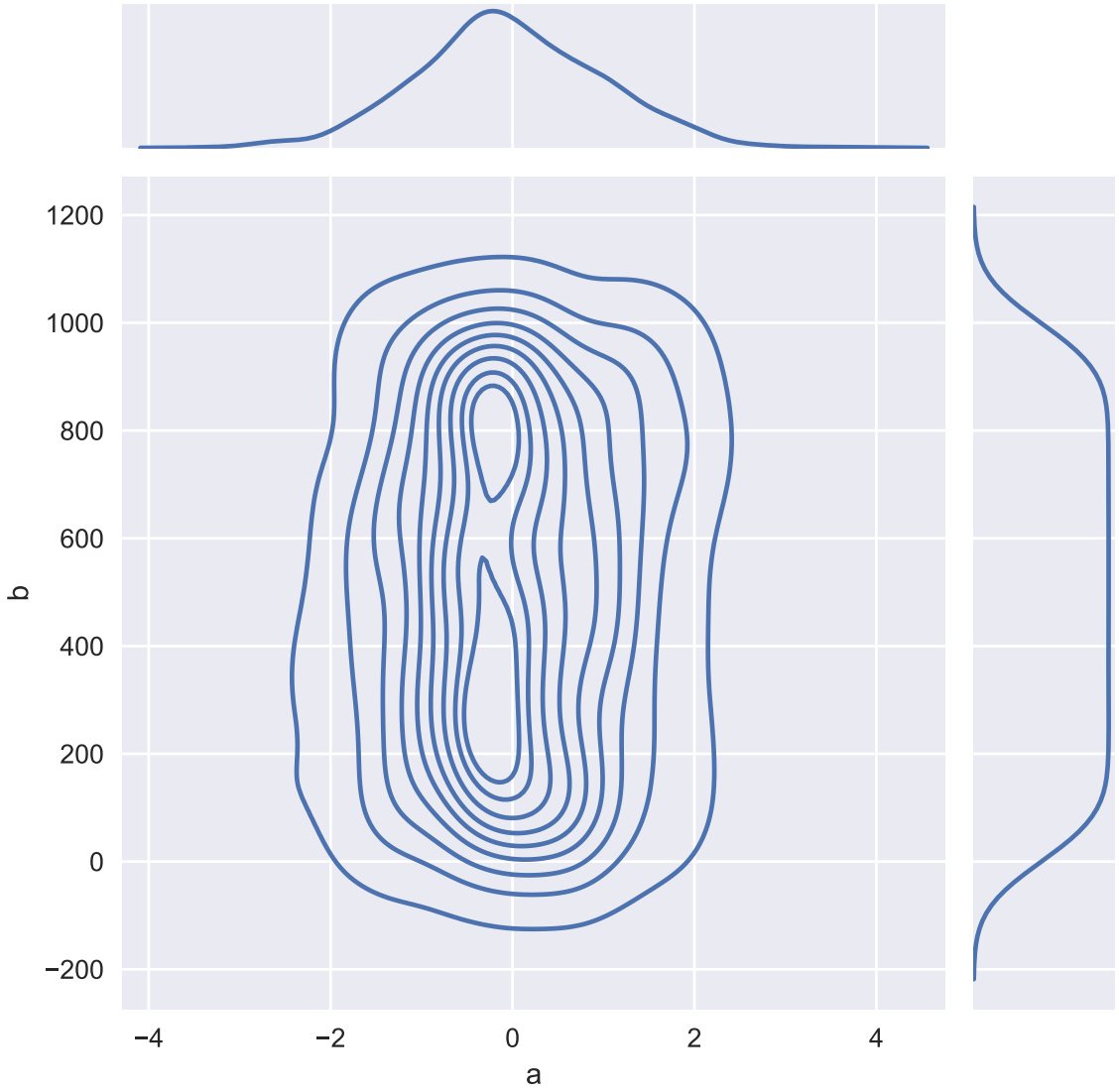



```
df.plot.hexbin(x='a', y='b', gridsize=25)
```




```
import seaborn as sns
```

```
import seaborn as sns
sns.jointplot(x="a", y="b", data=df, kind="kde")
```



8.3 Plotly 交互式可视化

> **!** 关于 cufflinks
>
> **cufflinks** 库已停止维护，且与新版本的 plotly/numpy 存在兼容性问题。
>
> 推荐使用 **Plotly Express**，它是现代化的、官方支持的交互式可视化库。

8.3.1 安装 Plotly

```
%pip install plotly kaleido --upgrade
```

8.3.2 Plotly Express 基础示例

Plotly Express 提供了简洁的 API 来创建交互式图表。

```
import plotly.express as px
import pandas as pd
import numpy as np

# 创建示例数据
df = pd.DataFrame({
    'x': np.random.randn(100),
    'y': np.random.randn(100),
    'category': np.random.choice(['A', 'B', 'C'], 100)
})

df.head()
```

	x	y	category
0	0.627366	-0.499237	B
1	-1.045043	0.234597	B
2	0.068052	-0.707555	A
3	2.302555	0.124455	A
4	-1.340164	-0.690232	B

```
# 散点图
fig = px.scatter(df, x='x', y='y', color='category',
                 title='交互式散点图',
                 labels={'x': 'X 轴', 'y': 'Y 轴'})
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
Unable to display output for mime type(s): text/html
```

```
# 创建时间序列数据
ts_df = pd.DataFrame({
    'date': pd.date_range('2024-01-01', periods=100),
    'value': np.cumsum(np.random.randn(100))
})

# 线图
fig = px.line(ts_df, x='date', y='value',
              title='时间序列图',
              labels={'date': '日期', 'value': '数值'})
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
# 直方图
fig = px.histogram(df, x='x', color='category',
                   title='分组直方图',
                   nbins=20)
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

```
# 箱线图
fig = px.box(df, x='category', y='y',
             title='箱线图',
             labels={'category': '类别', 'y': '数值'})
fig.show()
```

```
Unable to display output for mime type(s): text/html
```

Plotly 优势

- **交互式：** 可以缩放、平移、悬停查看数据
- **现代化：** API 简洁，文档完善
- **导出方便：** 支持导出为 HTML、PNG、PDF 等格式
- **样式丰富：** 内置多种主题和配色方案

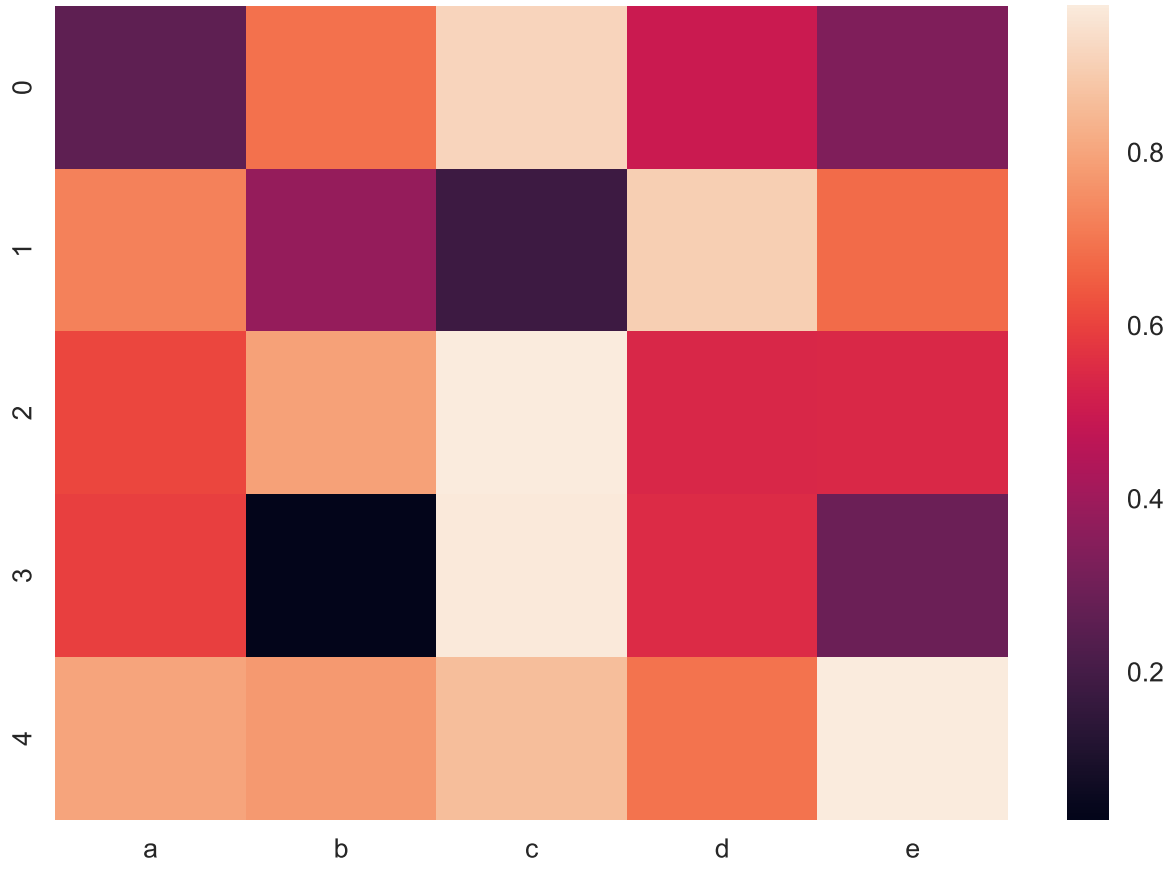
8.4 Seaborn 可视化库

https://seaborn.pydata.org/index.html

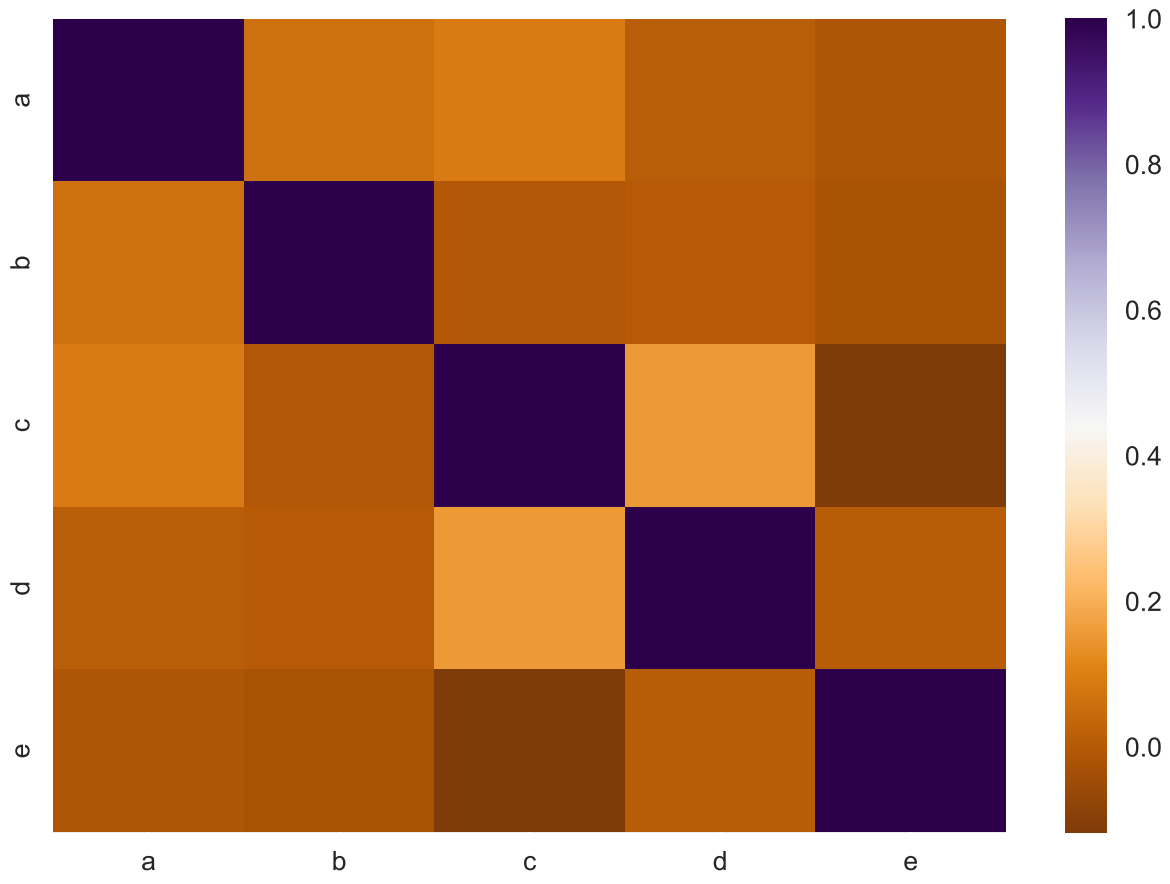
Seaborn 是基于 matplotlib 的图形可视化 python 包。它提供了一种高度交互式界面，便于用户能够做出各种有吸引力的统计图表。

https://seaborn.pydata.org/examples/index.html

```
# library
import seaborn as sns
import pandas as pd
import numpy as np

# Create a dataset (fake)
df = pd.DataFrame(np.random.random((5,5)), columns=["a","b","c","d","e"])

# Default heatmap: just a visualization of this square matrix
p1 = sns.heatmap(df)
```

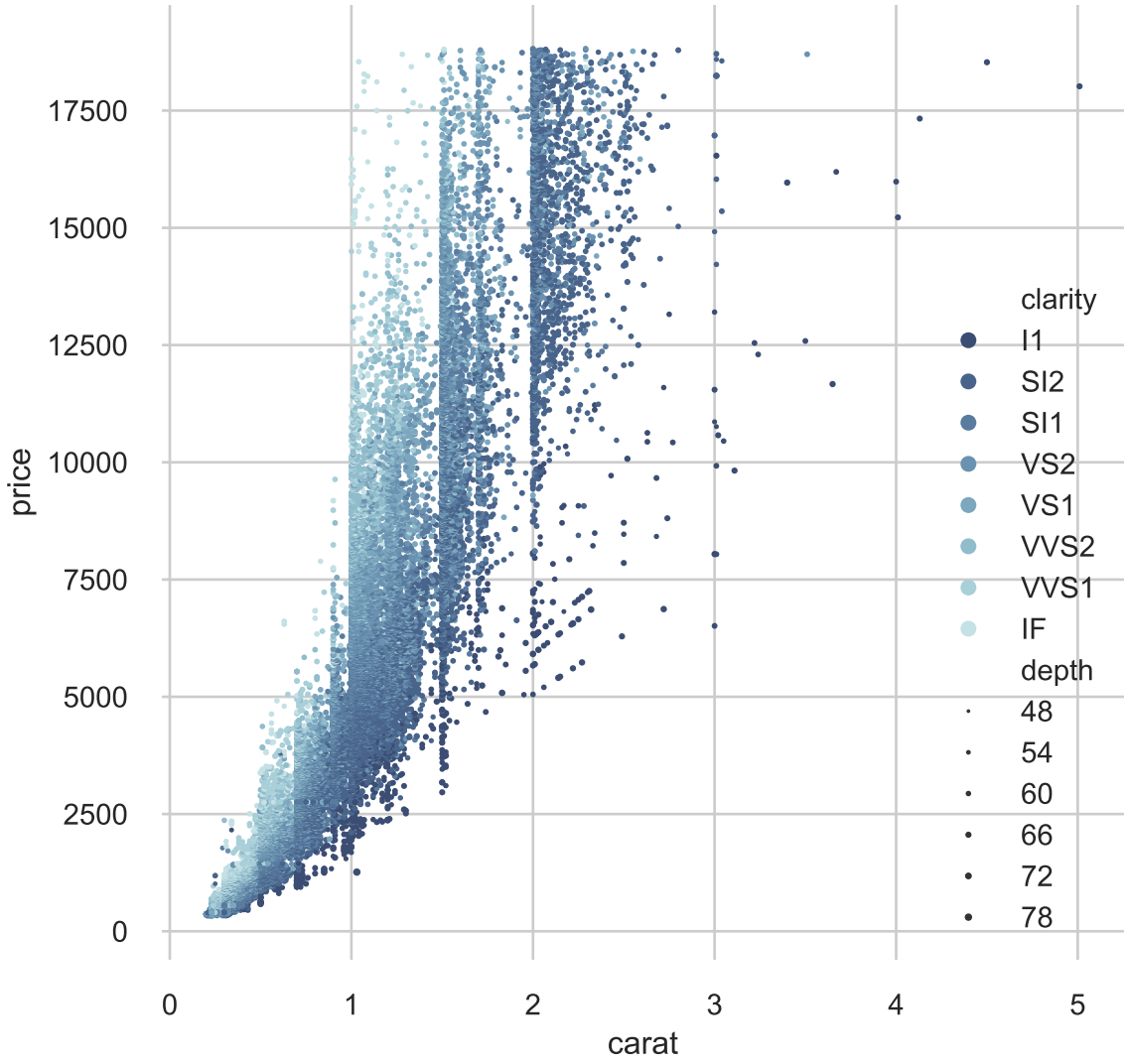



```
# library
import seaborn as sns
import pandas as pd
import numpy as np

# Create a dataset (fake)
df = pd.DataFrame(np.random.random((100,5)),
 ↪  columns=["a","b","c","d","e"])

# Calculate correlation between each pair of variable
corr_matrix=df.corr()

# plot it
sns.heatmap(corr_matrix, cmap='PuOr')
```



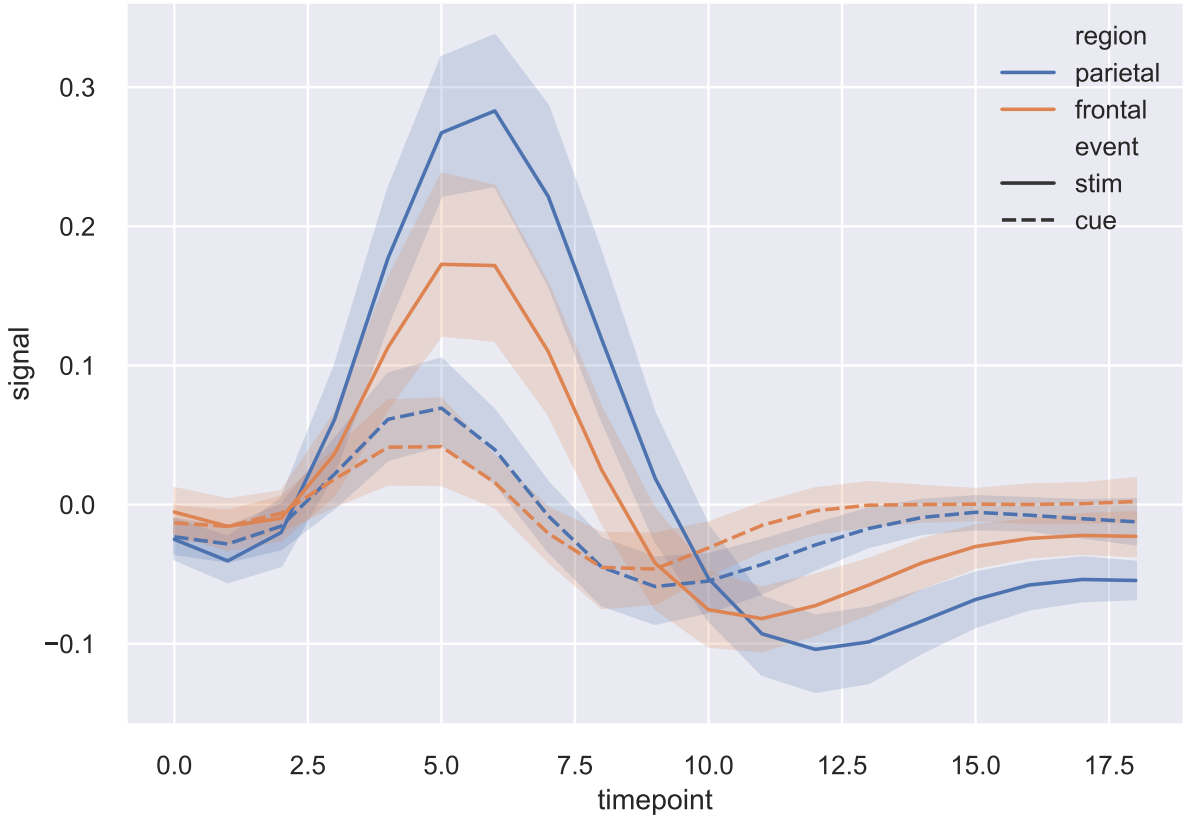

```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="whitegrid")

# Load the example diamonds dataset
diamonds = sns.load_dataset("diamonds")

# Draw a scatter plot while assigning point colors and sizes to different
# variables in the dataset
f, ax = plt.subplots(figsize=(6.5, 6.5))
sns.despine(f, left=True, bottom=True)
clarity_ranking = ["I1", "SI2", "SI1", "VS2", "VS1", "VVS2", "VVS1",
 ↪  "IF"]
sns.scatterplot(x="carat", y="price",
                hue="clarity", size="depth",
```

```
                palette="ch:r=-.2,d=.3_r",
                hue_order=clarity_ranking,
                sizes=(1, 8), linewidth=0,
                data=diamonds, ax=ax).get_figure().savefig("output.png",
↪  dpi=600)
```




```
import seaborn as sns
import matplotlib.pyplot as plt
sns.set_theme(style="darkgrid")

# Load an example dataset with long-form data
fmri = sns.load_dataset("fmri")
```

```
# Plot the responses for different events and regions
sns.lineplot(x="timepoint", y="signal",
                hue="region", style="event",
                data=fmri).get_figure()

plt.savefig('saving-a-seaborn-plot-as-png-file-transparent.png',
            transparent=True, dpi = 600)
```



8.5 folium 地图可视化

https://python-visualization.github.io/folium/

```
%pip install folium --upgrade
```

```
import folium
local = [23.132, 113.345]
# local = [23.155, 113.327]
# local = [23.019, 113.411]
m = folium.Map(location=local,zoom_start=16)
m
```

```
<folium.folium.Map at 0x11ce2b230>
```

```
local = [23.087, 113.37]
# 使用 OpenStreetMap 默认底图（推荐）
m = folium.Map(
    location=local,
    zoom_start=12
)

folium.Circle(
    radius=800,
    location=[23.133, 113.342],
    popup='暨南大学石牌校区',
    color='crimson',
    fill=False,
).add_to(m)

folium.CircleMarker(
    location=[23.155, 113.327],
    radius=15,
    popup='暨南大学华文校区',
    color='#3186cc',
    fill=True,
    fill_color='#3186cc'
).add_to(m)
```

```
folium.CircleMarker(
    location=[23.02829562968323, 113.41585435512872],
    radius=20,
    popup='暨南大学番禺校区',
    color='#3186cc',
    fill=True,
    fill_color='#3186cc'
).add_to(m)



m
```

```
<folium.folium.Map at 0x11ba59310>
```

```
# generated data
import numpy as np
data = (
    np.random.normal(size=(300, 3)) *
    np.array([[0.05, 0.05, 0.05]]) +
    np.array([[23.087, 113.3, 1]])
).tolist()
data
```

```
[[23.12292567475108, 113.31809261106721, 0.9512346127155135],
 [23.041480981566522, 113.27505139810303, 1.0014561027021551],
 [23.110695281822395, 113.18403315916512, 0.9318375272376518],
 [23.09015812185156, 113.29098970584552, 0.947261809937028],
 [23.125066284225852, 113.2412566302634, 0.9936659017200175],
 [23.0954231884294, 113.39268341322936, 0.9409524132677773],
 [23.076206805399124, 113.3092817998647, 1.0755670093995926],
 [23.08886483885913, 113.29449376179657, 1.073226387462056],
```

[23.156695116674182, 113.31020011468533, 1.020796497479569],
[23.134805461434812, 113.3221590889464, 1.1198492637777768],
[23.09212528354766, 113.3286074318886, 1.0147310800921767],
[23.130694765251995, 113.3162900943547, 0.9661963141438934],
[23.075351474581307, 113.33722577131158, 0.9755031633158013],
[22.972608530006, 113.40229425729164, 0.8756159305541846],
[23.15044704874554, 113.31430078675666, 0.9463823071198089],
[23.06513758525095, 113.32697030164556, 0.921914060110518],
[23.05153121843535, 113.30091381712379, 1.0006419441635177],
[23.025794479210848, 113.39099042839162, 1.0961652796852122],
[23.112030712037267, 113.3788489958498, 1.012073202829108],
[23.07953633211426, 113.34716647159757, 0.9936592812261629],
[23.139570208994762, 113.285401897008, 0.9872969515019017],
[23.051343255187913, 113.33695792088785, 1.0917263739890593],
[23.097304538933965, 113.28139877991453, 0.9992924654582637],
[23.14512611667325, 113.31463752631878, 0.9137752941933122],
[23.054589373217766, 113.3438156510386, 1.1095838617572624],
[23.085193414095166, 113.27282343645759, 0.991227732968647],
[23.04444550661176, 113.29261571096772, 1.0884383400706585],
[23.126469545037004, 113.29379422496584, 0.9950141219218092],
[23.16791937627841, 113.32066195555569, 1.097353598590821],
[23.084368287822745, 113.38645441751699, 1.0006209889297821],
[23.137260903962407, 113.37090679580558, 1.0327426991374837],
[23.18274299039718, 113.22943369468808, 0.968816356160638],
[23.071299767658658, 113.26832362514449, 1.0526259755276077],
[23.078726175885194, 113.33032814775468, 0.9871838871859377],
[23.164499776347228, 113.27613841888774, 0.9831172547378841],
[23.08717660000457, 113.2958480141386, 1.0197082057281965],
[23.074763998802222, 113.2681607221764, 1.0313599422889252],
[23.020664668731182, 113.31658825862529, 1.0214707785175974],
[23.003087013029866, 113.30526938021956, 1.00431206667352],
[23.071556998953913, 113.35323157123479, 1.0802903370424324],

[23.08600201916953, 113.3194171393517, 0.9009619505636194],
[23.17410926529449, 113.31945303347577, 1.0127410889218496],
[23.0907295903923, 113.37367801803224, 0.9747770184339344],
[22.95096957529456, 113.2573105512718, 1.0709426552931949],
[23.133050780284957, 113.40678506511614, 0.9710102127702553],
[23.103403176667694, 113.24245030272121, 1.0341702627979958],
[23.19895759617069, 113.21834348342354, 0.9522490882116766],
[23.09798261222306, 113.30418117662032, 0.9791823244296058],
[23.007328434250343, 113.40031473166533, 1.029881128305673],
[23.07566222810024, 113.24170776518467, 1.0634749888731827],
[23.156741602689266, 113.3095073072332, 1.0031023188314387],
[23.07726907285033, 113.4004272717788, 1.0197222625996527],
[23.040623362283043, 113.28614658196811, 0.9975025863737166],
[23.10262035764237, 113.24102439897791, 1.0290853582832504],
[23.127858343876927, 113.33058772052696, 0.9902143488522896],
[23.087728940980334, 113.27100555684953, 1.0539794987326014],
[23.008814303713372, 113.32201821706005, 0.9551414985218852],
[23.082233795456343, 113.30194055689061, 0.9657589639402723],
[23.041896914643907, 113.31723263463309, 1.0394398132117124],
[23.1224638991438, 113.32172613820511, 0.9568114416174717],
[23.146654880529777, 113.22812518016103, 1.0038200866173963],
[23.0975864533372, 113.34684012282491, 0.9683882167872174],
[23.089788402788614, 113.28752529075138, 0.9340373173381542],
[22.995836791976014, 113.30142615078746, 0.9629764047587978],
[23.003966523163946, 113.28457824481265, 0.9974131719999153],
[23.084525661563358, 113.35218560634391, 1.063595808791513],
[23.044687235416173, 113.23858897936852, 1.0206296596624913],
[23.14218103928034, 113.31843292594432, 0.9731828391542027],
[23.00185879732825, 113.24931714611772, 1.0532470158418312],
[23.071254111600165, 113.25771011122946, 0.9981969962138197],
[23.162371094979214, 113.28414356620009, 1.0338432641090314],
[23.047355414556744, 113.28052642007968, 0.9768681049053216],

[23.051297336293757, 113.27761609786553, 0.9759883987156982],
[23.102498025655056, 113.2844402229352, 0.9612772593739601],
[23.06641312912853, 113.26452200167576, 0.9554492381139845],
[22.986649257705917, 113.43406363511278, 1.0105403000507227],
[23.11281963296153, 113.31941999249435, 0.9839878843692701],
[23.129075443928663, 113.3421156036694, 1.047942380726218],
[23.041486049586045, 113.2204967313712, 0.949280909265336],
[23.21177627548372, 113.38316471859672, 0.9256469145588638],
[23.108231492091345, 113.35882904528813, 1.066215003547937],
[23.120619551085966, 113.27506705696995, 0.994967877121682],
[23.096050360411947, 113.32513760857503, 1.1003517155110996],
[23.128120591704246, 113.32032980412413, 0.9327065831130598],
[23.069894640732066, 113.29039879496638, 0.9141795265980605],
[23.098494116284062, 113.24325348576602, 1.0170363797536706],
[23.121697667224577, 113.34340381375044, 1.037742532290055],
[23.038806377916277, 113.25721686668791, 0.994931491491079],
[22.999223596270056, 113.2934426669681, 1.049197861487439],
[23.104323307031247, 113.26526525834568, 0.9530484433712318],
[23.1387809667552, 113.32706768524493, 0.989258130616403],
[22.996414889133398, 113.33839286539919, 0.9537001386655031],
[23.07448440537044, 113.29332386459518, 0.9986984686341231],
[23.06781378216937, 113.34201710406913, 1.0328437857808297],
[23.03617280481243, 113.34417150957852, 0.9485877592584417],
[23.1458776298731, 113.37550386091327, 1.0138418763405146],
[23.1298319124015, 113.32211853247115, 0.9584238834911811],
[23.06618384924404, 113.1840113196497, 1.1002199282375327],
[23.05928269870583, 113.31012935616161, 0.9758915125782275],
[23.073494284809, 113.26217409986687, 0.9838915982295962],
[23.078787238754064, 113.26146865730784, 0.9680667685124081],
[23.10626428089576, 113.23983719610032, 0.9935165600423754],
[23.082647996953735, 113.308209656826, 0.965444915621199],
[23.10123192857074, 113.33730147248562, 1.129757863707711],

[23.04795512117888, 113.32913020104714, 1.003476636674064],
[23.11057886697128, 113.4271198055599, 0.9967332536172089],
[23.06117462823625, 113.1870854826703, 1.044796739723515],
[23.052049673249314, 113.32336552803926, 0.9433316011860172],
[23.092545236891564, 113.39011186348878, 0.9863813717996438],
[23.124719163520435, 113.2462614664115, 1.022928402214007],
[23.023084338403375, 113.26946832128952, 1.0362699616454099],
[23.077580441388186, 113.27253666607423, 0.9837941157446796],
[23.062890758168997, 113.36763215916993, 0.9890372151314588],
[23.104786849333387, 113.20978417600725, 1.0478071273897254],
[23.114018249625932, 113.25134029764436, 0.9747826065417896],
[23.153550432573386, 113.21794829404622, 0.9636764217168993],
[23.045360725144544, 113.37187084265922, 1.0263292074909873],
[23.04842621155067, 113.38470546997445, 0.9211024234734683],
[22.981004845735384, 113.34405638395721, 0.9202682577348917],
[23.087771977063532, 113.3227173780349, 1.0136750501679086],
[22.985984637096774, 113.30347140582688, 1.0669938980296503],
[23.050709843770502, 113.35887477226288, 1.02573719754271],
[22.99742354171756, 113.24888218591104, 0.9504541072600879],
[23.05023101311708, 113.37683191834329, 1.0479384919392547],
[23.077204786163, 113.29512751595561, 1.0286484183562605],
[22.969160848071894, 113.33886443147135, 0.9276116139548678],
[23.07096926789701, 113.29140424465719, 0.9879327706259524],
[23.079665285527327, 113.33353340890324, 1.1255368063283484],
[23.034212275934664, 113.23035225634703, 1.0181725018603602],
[23.113186152632373, 113.27903339473215, 1.0431089008628824],
[23.01040294763731, 113.376599829609, 0.973446870371315],
[23.064284443604908, 113.27975522644063, 1.0293959739538052],
[23.171650923764687, 113.31444346388845, 0.9949925578913307],
[23.007602118667847, 113.22430437405451, 1.057509012413103],
[23.07903579319951, 113.26999200112336, 1.1139912753640133],
[23.114486758176113, 113.12956197321343, 0.991093562220511],

[23.132626920236408, 113.25896646702371, 1.0115296312068003],
[23.069011842770095, 113.29593823135492, 1.0284655783407466],
[23.055037906881083, 113.33453502940509, 0.9339961143041128],
[23.096198994421272, 113.33987068178011, 0.9656040715007674],
[23.003605723643055, 113.36536639010548, 0.9550579592802125],
[22.97709721866362, 113.30188341391961, 1.0023084610162543],
[23.172023636774703, 113.32397024317054, 1.0423898906684972],
[23.05130137437062, 113.33179687128145, 0.9711534620666442],
[23.103178951354266, 113.28099687999223, 1.0138104363486076],
[23.11716812946778, 113.3223397644357, 0.9253814715723656],
[23.147678969874594, 113.26430415487924, 0.910272688919386],
[23.105772870624897, 113.27330393757747, 0.9592355319781251],
[23.040658092751706, 113.276122185894, 1.0532973867324158],
[23.020549688806398, 113.34614126627224, 0.9798801434221969],
[23.16892821315353, 113.30843415856708, 0.9966871351606071],
[23.131013837480403, 113.30613252257513, 1.0916744033777226],
[23.14848371965029, 113.35901316590264, 1.0481371367225172],
[23.168109777378152, 113.32530231844041, 1.0639136615545666],
[23.104891393044248, 113.2467825705774, 1.0383617181697853],
[23.040166705705083, 113.16162576546051, 0.9717354318318634],
[23.06063419390737, 113.33796831285375, 0.9738527183327209],
[23.0299233728093, 113.30680414009801, 0.9920796128258713],
[23.102123496530478, 113.20536059492618, 0.9671166918987479],
[23.05007677019235, 113.36109629139268, 1.0034781655933736],
[23.110615328951056, 113.34031091240281, 1.0436867714054057],
[23.063493394175413, 113.38046956768963, 0.9411821497616826],
[23.036581327271097, 113.29690560025129, 0.9107429049876126],
[23.038392351904587, 113.32248579483938, 0.931044160477816],
[23.102190941950134, 113.2609104006009, 1.005136804097656],
[23.1341250673337, 113.26523704926888, 0.9687817844978972],
[23.167180637766545, 113.35914723035545, 1.0162737339381747],
[23.083677766772404, 113.30681337071519, 0.9655823335620674],

[23.175136817316684, 113.19189359501969, 1.0471277217801767],
[23.1492107608909, 113.44210183575827, 1.0301354492223733],
[23.127186171568624, 113.30593923497145, 1.0280133659885327],
[23.033934060950056, 113.25968864517034, 0.9455019836596257],
[23.144665101590146, 113.32049001835182, 1.0095401422061794],
[23.033907688282518, 113.34924501962671, 0.9527465100546084],
[22.982492332603798, 113.32713527512823, 0.9249915866485441],
[23.103679023992267, 113.29136881108371, 0.9865558276660872],
[23.04382898418947, 113.34558598697292, 0.9761868655546372],
[23.09541912886149, 113.26737295498934, 1.038372349810079],
[23.133578422328558, 113.2789178147626, 0.9815678041713445],
[23.137297964465656, 113.28922095283336, 0.9372065395239505],
[23.171181050822984, 113.37956900974099, 1.0890442885068372],
[23.205473377766886, 113.22507707219653, 1.027814317290653],
[23.109384961551115, 113.33525830688913, 0.9840579128199375],
[23.04273709056374, 113.33023167722227, 0.9284830627815504],
[23.123747222217098, 113.22151373781539, 0.9938482047406878],
[23.080453493522132, 113.25941777907177, 1.0030941149751162],
[23.082607634200734, 113.25199585137314, 0.9025227004427396],
[23.037229196607385, 113.29354365306872, 0.9493449971738958],
[23.111686618621526, 113.41884888889875, 0.9679566969587095],
[23.08368646811695, 113.29536095653303, 1.0537445372183525],
[23.03449307912378, 113.30491823690242, 0.977923142848577],
[22.998420297984705, 113.4201682538007, 0.8815339810421324],
[23.080917669082503, 113.35129289842445, 1.0197517587869973],
[23.11482063874529, 113.40915895042686, 0.9461470785747483],
[23.090328957611145, 113.32348035671063, 1.037866286006159],
[23.024919059823144, 113.21566393889339, 0.92059232648922],
[23.11066331082251, 113.32689642933225, 0.9789650400454358],
[23.075105588097852, 113.27043109124348, 1.0104678853073994],
[23.06832391460701, 113.32425146760727, 0.9979330854351578],
[23.13060978915038, 113.34583015432841, 0.9438880321913511],

[23.062444101491927, 113.36329255241412, 0.9857168512673958],
[23.01715970979868, 113.34486795965574, 0.9396929893350328],
[22.99258675859923, 113.28296997666338, 0.9134284388330829],
[23.044958970789985, 113.25759845367584, 0.9440999133477848],
[23.12792063927098, 113.22027378939943, 0.9719477821953935],
[23.058731091146914, 113.38111896052935, 0.98553700125542],
[23.138650724662842, 113.36398398256469, 1.0020789606280478],
[23.073932367858315, 113.24337191417074, 1.005833617098707],
[23.12413331271888, 113.35009835708352, 1.0609536685996555],
[23.06113848503462, 113.28044949706566, 0.9698245744039555],
[23.076074513029813, 113.2944823481715, 0.9125046692961192],
[23.054729425462934, 113.3173596569306, 1.0562035532928036],
[23.13241763531783, 113.26745036287632, 0.9897997443571512],
[23.036160887718385, 113.33445699208183, 1.0649388685737],
[23.105538025975665, 113.26230608223963, 1.0544525447651114],
[23.193746228321803, 113.33321568350172, 1.0110967306532803],
[23.05261489695624, 113.34286870541185, 1.0065826912760452],
[23.087026039190597, 113.30816138359104, 0.9892350993807316],
[23.085789155891767, 113.25965442837615, 0.9737294321608534],
[23.061689142955476, 113.36729753503141, 1.0292209732711293],
[23.102335416841647, 113.30299963325102, 0.9249983969836697],
[23.06941056471305, 113.26101296617104, 1.0888640438972836],
[23.05325394808547, 113.36692529114484, 1.0391889285797389],
[23.112604859402875, 113.2325176729842, 0.8853144688706669],
[23.048108831694538, 113.36513376941906, 1.0255553056677393],
[23.047472664654315, 113.25282242998384, 1.09044728363618],
[22.964981579734918, 113.30456825549108, 0.9460953778071548],
[23.15937445868103, 113.35809310361283, 0.9733278209284137],
[23.009000179362978, 113.28253406265881, 1.0326927147576812],
[23.131638450677762, 113.25648212899264, 0.9495265721925205],
[22.997272868240877, 113.25246082123445, 1.1095879427158188],
[23.17425496606531, 113.37475031693936, 1.0260024308492381],

[23.047037146605142, 113.32168068234373, 0.9985534845248806],
[23.044573642663554, 113.3002333400525, 0.9567043240003137],
[23.016253534427598, 113.27155263810711, 0.9240905368947935],
[23.071561793358434, 113.27246812206441, 1.0146094757216226],
[23.10089962927346, 113.34380591516414, 0.9870438801374537],
[23.122700358916344, 113.28923755693053, 0.981689092385444],
[23.200754882929566, 113.23907569011978, 1.0736132143541073],
[23.05671729294272, 113.24642174272074, 0.9692499528211079],
[23.167807593499635, 113.29851390138822, 1.0071695327149592],
[23.19136111254222, 113.30882594653231, 0.9125699229562564],
[23.046133447345536, 113.26348105252589, 1.03978639515402],
[23.097530233575515, 113.31678261747464, 0.9091412181823305],
[23.158675513862832, 113.2615309240117, 1.0299487778761662],
[23.185145839082082, 113.34828191107871, 1.0025543147642881],
[23.083570028900922, 113.32150832429629, 1.0389021268202652],
[23.156604860217367, 113.3162292559205, 1.0228526696579607],
[23.096316578215287, 113.27016004829684, 0.9505642196076789],
[23.025555850233598, 113.30214536268608, 1.0078529572945947],
[23.062132859741347, 113.30445163105126, 0.9969475879579656],
[23.03153547786772, 113.32888725908839, 1.0888461193400105],
[23.082141900890342, 113.25891160644882, 0.955259765104159],
[23.143349132065488, 113.242494827586, 0.9722884190071142],
[23.087927341444058, 113.25519751013817, 1.00122227462315],
[23.115135527124963, 113.29294594556991, 0.9845582022259945],
[23.221179964722534, 113.3260404961154, 0.9534455353958193],
[23.037307893990537, 113.33594319927164, 1.0164118946833003],
[22.999228959890033, 113.24970715913389, 0.9589220366117504],
[23.096801025800882, 113.33199633866889, 1.0770371514698034],
[23.012642977079754, 113.31347590510333, 0.9806021725707501],
[23.105027615653626, 113.25852398480747, 1.0013523119866372],
[23.065268830752185, 113.28842446685438, 1.0028081790139378],
[23.017506270583866, 113.22491653765282, 0.9944864864541447],

[23.12629992454365, 113.37958220773633, 1.100685226484174],
[23.126055945805724, 113.28401858381856, 0.9486954539886417],
[23.057901547697778, 113.28648228878693, 1.0391973752529808],
[23.083635136684972, 113.3856954646631, 1.023335494180312],
[23.106299851195743, 113.35951305057117, 1.0302995529608698],
[23.096894140300634, 113.24120418732883, 0.9961796101627634],
[23.142976882739866, 113.35503749152537, 0.9084820811782881],
[23.06715875993821, 113.26409727061159, 1.0049097499701758],
[23.10243444560605, 113.33849679912996, 1.057151442241974],
[23.134683177792613, 113.2307063641154, 1.1305059610778272],
[23.10919096962813, 113.36936740118813, 0.9371372444604422],
[23.065299291835842, 113.36137768639006, 0.9813025219542669],
[23.103077926922897, 113.21470289176665, 0.9787826944062553],
[23.11193358743528, 113.25444093940071, 1.0259783994161253],
[23.107091104104892, 113.29697053174657, 0.9459933496786161],
[23.09066164435428, 113.34389972389853, 1.000195849170243],
[23.008421107753463, 113.2455672346012, 1.0049417644408518],
[23.062317380800778, 113.30664066812115, 1.0673273326550967],
[23.073789624105586, 113.30980088686773, 0.955515988226136],
[23.13182146001401, 113.3719943767497, 1.0273175324681623],
[23.079624306906567, 113.25209601362968, 0.9701390813946531],
[23.04024156804249, 113.23588257560213, 1.0543141198897426],
[23.132801163346954, 113.33248803940424, 0.9822636866450936],
[23.091146672562882, 113.31807210595011, 1.0870583428927847],
[23.11779544895673, 113.26879578218016, 0.9015952114800665],
[23.129750244312568, 113.30546749688361, 1.0773147162456953],
[23.045895642418674, 113.35038822875588, 0.9079177149086665],
[23.047365655385324, 113.28427890230681, 1.0460633743681738],
[23.066444217155958, 113.2356766566315, 1.0379761377716792],
[23.02945551767962, 113.28140420937177, 0.9189601933999842],
[23.087559170199537, 113.31509792730053, 0.9751871810170482],
[23.066930940600745, 113.27258025336806, 0.9707835515964236],

```
 [23.127008142124485, 113.34711601904912, 0.9945290058488773],
 [23.00988761104292, 113.25435234197204, 1.0518326649244096],
 [23.189407458899797, 113.34642333613476, 1.0259994701721056],
 [23.067016086343415, 113.245147780456, 1.0835103203591014]]
```

```python
# HeatMap（热力图）
from folium.plugins import HeatMap

# 创建地图（使用默认 OpenStreetMap 底图）
m = folium.Map([23.087, 113.3], zoom_start=12)

# 添加热力图层
HeatMap(data).add_to(m)

# 保存为 HTML 文件
m.save('Heatmap.html')
print(" 热力图已保存到 Heatmap.html")

# 显示地图
m
```

```
热力图已保存到 Heatmap.html
```

```
<folium.folium.Map at 0x11b987c50>
```

> **Folium 底图选择**
>
> 新版本 folium 推荐使用以下方式指定底图：

```
# 默认 OpenStreetMap（推荐）
folium.Map(location=[lat, lon])

# 使用其他底图提供商
folium.Map(location=[lat, lon], tiles='OpenStreetMap')
folium.Map(location=[lat, lon], tiles='CartoDB positron')
folium.Map(location=[lat, lon], tiles='CartoDB dark_matter')
```

注意：Stamen 底图已不再免费提供，建议使用 OpenStreetMap 或 CartoDB。

9 网络爬虫

点击下载本章节代码

9.1 网页基础知识

9.1.1 核心概念

1. 什么是网站?

   - 网站是通过互联网提供的一组相互链接的网页。
   - 网站的组成部分：
     - 前端：用户可见的部分，包括网页结构（HTML）、样式（CSS）和交互功能（JavaScript）。
     - 后端：运行在服务器上的部分，负责逻辑处理、数据存储和提供 API 接口。

2. 网络如何工作

   - 域名和 IP 地址：
     - 域名（如 `example.com`）是方便人类记忆的网络地址，通过 DNS（域名系统）映射到具体的 IP 地址。
   - **HTTP** 和 **HTTPS** 协议：
     - 超文本传输协议（HTTP）定义了客户端和服务器之间的通信规则。
     - HTTPS 是加密版本，确保数据传输的安全性。
   - 客户端请求和服务器响应：
     - 请求包含方法（如 `GET` 或 `POST`）、URL、头部信息等。
     - 响应包含状态码（如 200 表示成功）、内容类型（如 HTML）和具体的数据。

3. 基本网页技术

   - **HTML**（超文本标记语言）：

- 定义网页的结构，包括标题、段落、链接、图片等。
- 示例：

```
<html>
<head>
    <title>示例网页</title>
</head>
<body>
    <h1>欢迎来到示例网页</h1>
    <p>这是一个段落。</p>
</body>
</html>
```

- **CSS**（层叠样式表）：
  - 控制网页的外观样式（颜色、字体、布局等）。
  - 示例：

```
    <style>
        h1 {
        color: red;
        }
        p {
        font-size: 30px;
        }
    </style>
```

- **JavaScript**：
  - 为网页添加交互性（如按钮点击、动态加载内容）。
  - 示例：

```
    <button id="myButton">Click Me</button>
    <p id="message"></p>
    <script>
        // Get references to the button and the paragraph
        const button = document.getElementById("myButton");
```

```
        const message = document.getElementById("message");

        // Add an event listener to the button
        button.addEventListener("click", () => {
            // Change the text content of the paragraph
            message.textContent = "You clicked the button!";
        });
    </script>
```

4. **客户端-服务器模型**

- **客户端：**
  - 发送请求到服务器。请求 `requests`
  - 示例：浏览器、Python 爬虫脚本。
- **服务器：**
  - 接收请求并返回响应。响应 `response`
  - 示例：托管网页的服务器、提供 API 服务的服务器。
  - 响应 response
  - 响应的内容类型：
    * `text`，如 `HTML`、CSS、JavaScript
    * `image`，如 JPEG、PNG
    * `pdf`，如 PDF 文档
    * `json`，书写格式类似于 python 的字典 `dict`
    * `xml`，类似于 html 的标记语言

HTTP 状态码简介

- **200**：请求成功。
- **404**：资源未找到。
- **500**：服务器内部错误。

浏览器开发者工具

- **检查网页元素**：右键点击网页，选择“检查”可以打开开发者工具。
- **Network 标签**：查看页面加载过程中所有请求和响应。

- **Elements 标签**：查看网页的 HTML 和 CSS 结构。

演示

1. 打开浏览器并访问一个示例网站，例如 `https://www.jnu.edu.cn`。
2. 使用开发者工具查看：

- HTML 元素结构。
- 页面中加载的 CSS 和 JavaScript 文件。
- 通过 Network 标签查看页面加载时的网络请求。
- 豆瓣读书 **https://book.douban.com/tag/小说**

9.2 爬虫基础

安装所需的库:

```
%pip install requests httpx parsel playwright --upgrade
```

安装 Playwright 浏览器（在终端运行）：

```
playwright install chromium
```

9.2.1 分析网址规律

1. 网址包含信息
2. 是否需要 load `javascript`
3. response 是否为 json 数据
4. 是否需要用模拟浏览器

9.2.2 requests 库

找到网页 url 规律后，我们需要将这些 url 对应的网页数据下载下来，这里就用到**requests 库文档链接**。

requests 两种访问方法，两者都返回 Response 对象：

requests 常用函数	参数解读
requests.get(url, params, verify)	发起 get 访问，返回 **Response 对象**；除非需要传参，否则不需要用 **params 和 verify 参数**
requests.post(url, data)	发起 post 访问，返回 **Response 对象**；除非需要传参，否则不需要用 **data 参数**

```
import requests

url = "http://www.ruanyifeng.com/blog/archives.html"

try:
    r = requests.get(url)
    print(type(r))
    print(r.status_code)
except Exception as e:
    print(e)
```

```
<class 'requests.models.Response'>
200
```

```
import requests

headers = {'User-Agent':
            'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
            ↪  (KHTML, like Gecko) Chrome/106.0.0.0 Safari/537.36'
            }

url = 'https://book.douban.com/tag/小说?start=20&type=T'

try:
    r = requests.get(url, headers = headers)
```

```
    print(type(r))
    print(r.status_code)
except Exception as e:
    print(e)
```

```
<class 'requests.models.Response'>
200
```

注意 Response 后面带有的状态码： - 2 开头表示访问正常 - 4 开头，比如 403 表示爬虫被网站封锁 - 5 开头表示服务器出问题

9.2.3 Response 响应对象

Response 对象的方法	作用
Response.json()	获得 json 格式网页数据
Response.text	获得 html 网页数据
Response.content	获得网页**二进制文件数据**，常常用该方法爬取**图片、视频**等数据
Response.encoding	更改网页数据的编码方式。除非使用 Response.text 得到的 html 中文本是乱码的，否则一般不用 Response.encoding
Response.status_code	查看当前访问的状态码，200 表示访问正常，4 开头的状态码表示访问出问题，5 开头的状态码表示服务器出问题

以上三种方法，最常用的是 `text` 和 `json` 方法，分别对应于 `html` 网页数据和 `json` 网页数据

```
r.status_code
```

```
200
```

```
print(r.text[:1000])
```

```
<!DOCTYPE html>
<html lang="zh-cmn-Hans" class="ua-windows ua-webkit book-new-nav">
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <title>
  豆瓣图书标签: 小说
</title>

<script>!function(e){var o=function(o,n,t){var c,i,r=new Date;n=n||30,t=t||"/",r.setT
```

```
with open("sample.html", "w+", encoding=r.encoding) as file:
    file.write(r.text)
```

```
from pathlib import Path
Path("sample.html").write_text(r.text)
```

```
54094
```

```
r.encoding
```

```
'utf-8'
```

```
import requests

r = requests.get('http://httpbin.org/get')
print(r.text)
```

```
{
```

```
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.32.5",
    "X-Amzn-Trace-Id": "Root=1-69071ea9-404f98341e9f9d7a2ab0c878"
  },
  "origin": "129.227.235.26",
  "url": "http://httpbin.org/get"
}
```


```
x = r.json()
x
```


```
{'args': {},
 'headers': {'Accept': '*/*',
  'Accept-Encoding': 'gzip, deflate',
  'Host': 'httpbin.org',
  'User-Agent': 'python-requests/2.32.5',
  'X-Amzn-Trace-Id': 'Root=1-69071ea9-404f98341e9f9d7a2ab0c878'},
 'origin': '129.227.235.26',
 'url': 'http://httpbin.org/get'}
```


```
x['url']
```

```
'http://httpbin.org/get'
```

9.2.4 巨潮资讯网

科创板的公告:

http://www.cninfo.com.cn/new/commonUrl?url=disclosure/list/notice#sseKcp

```
import requests
import pandas as pd

url = "http://www.cninfo.com.cn/new/hisAnnouncement/query"

headers = {
    'content-type': 'application/json',
    'Accept-Encoding': 'gzip, deflate',
    'User-Agent': 'Mozilla/5.0 (Windows NT 6.1; WOW64; rv:53.0)
    ↪  Gecko/20100101 Firefox/53.0'
    }
```

```
params = {'column': 'szse',
            'pageNum': 1,
            'plate': 'sz',
            'searchkey': '辞职;', # key1 or key2
            'pageSize': 30,
            'seDate': '2023-10-01 ~ 2023-10-22',
            'tabName': 'fulltext'}
r = requests.post(url, params=params, headers=headers)
```

```
r.json()
```

```
{'classifiedAnnouncements': None,
 'totalSecurities': 0,
 'totalAnnouncement': 95,
 'totalRecordNum': 95,
 'announcements': [{'id': None,
   'secCode': '002939',
   'secName': '长城证券',
   'orgId': 'qsgn0000030',
   'announcementId': '1218108610',
```

```
 'announcementTitle': '关于公司董事辞职的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218108610.PDF',
 'adjunctSize': 211,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '长城证券',
 'shortTitle': '关于公司董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300801',
 'secName': '泰和科技',
 'orgId': '9900031706',
 'announcementId': '1218108177',
 'announcementTitle': '关于独立董事辞职暨补选独立董事的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218108177.PDF',
 'adjunctSize': 210,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
```

```
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '泰和科技',
 'shortTitle': '关于独立董事辞职暨补选独立董事的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000420',
 'secName': '吉林化纤',
 'orgId': 'gssz0000420',
 'announcementId': '1218108160',
 'announcementTitle': '关于董事、董事会秘书辞职的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218108160.PDF',
 'adjunctSize': 149,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '吉林化纤',
 'shortTitle': '关于董事、董事会秘书辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
```

{'id': None,
 'secCode': '001965',
 'secName': '招商公路',
 'orgId': 'GD008065',
 'announcementId': '1218108158',
 'announcementTitle': '关于高级管理人员辞职的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218108158.PDF',
 'adjunctSize': 105,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '招商公路',
 'shortTitle': '关于高级管理人员辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002075',
 'secName': '沙钢股份',
 'orgId': '9900001101',
 'announcementId': '1218107902',
 'announcementTitle': '关于职工代表监事辞职及补选的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218107902.PDF',
 'adjunctSize': 77,

```
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '沙钢股份',
 'shortTitle': '关于职工代表监事辞职及补选的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000532',
 'secName': '华金资本',
 'orgId': 'gssz0000532',
 'announcementId': '1218106993',
 'announcementTitle': '关于公司董事辞职的公告',
 'announcementTime': 1697817600000,
 'adjunctUrl': 'finalpage/2023-10-21/1218106993.PDF',
 'adjunctSize': 119,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
```

```
 'orgName': None,
 'tileSecName': '华金资本',
 'shortTitle': '关于公司董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300053',
 'secName': '航宇微',
 'orgId': '9900010151',
 'announcementId': '1218109594',
 'announcementTitle': '关于副总经理辞职的公告',
 'announcementTime': 1697798665000,
 'adjunctUrl': 'finalpage/2023-10-20/1218109594.PDF',
 'adjunctSize': 99,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '航宇微',
 'shortTitle': '关于副总经理辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300650',
 'secName': '太龙股份',
 'orgId': '9900031733',
```

```
 'announcementId': '1218109108',
 'announcementTitle': '关于副总经理辞职的公告',
 'announcementTime': 1697796629000,
 'adjunctUrl': 'finalpage/2023-10-20/1218109108.PDF',
 'adjunctSize': 62,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '太龙股份',
 'shortTitle': '关于副总经理辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002761',
 'secName': '浙江建投',
 'orgId': '9900023671',
 'announcementId': '1218108414',
 'announcementTitle': '中国国际金融股份有限公司关于浙江省建设投资集团股份有限公司董
 'announcementTime': 1697793563000,
 'adjunctUrl': 'finalpage/2023-10-20/1218108414.PDF',
 'adjunctSize': 315,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
```

```
 'announcementType': '01010501||010112||013999',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '浙江建投',
 'shortTitle': '中国国际金融股份有限公司关于浙江省建设投资集团股份有限公司董事长辞职
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002761',
 'secName': '浙江建投',
 'orgId': '9900023671',
 'announcementId': '1218108413',
 'announcementTitle': '招商证券股份有限公司关于浙江省建设投资集团股份有限公司董事长
 'announcementTime': 1697793434000,
 'adjunctUrl': 'finalpage/2023-10-20/1218108413.PDF',
 'adjunctSize': 397,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010501||010112||013999',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '浙江建投',
 'shortTitle': '招商证券股份有限公司关于浙江省建设投资集团股份有限公司董事长辞职暨
 'announcementTypeName': None,
```

```
 'secNameList': None},
{'id': None,
 'secCode': '000584',
 'secName': 'ST工智',
 'orgId': 'gssz0000584',
 'announcementId': '1218097684',
 'announcementTitle': '关于公司监事辞职暨选举非职工监事的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218097684.PDF',
 'adjunctSize': 221,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': 'ST工智',
 'shortTitle': '关于公司监事辞职暨选举非职工监事的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000826',
 'secName': '启迪环境',
 'orgId': 'gssz0000826',
 'announcementId': '1218097065',
 'announcementTitle': '关于公司高级管理人员辞职的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218097065.PDF',
```

```
 'adjunctSize': 163,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '启迪环境',
 'shortTitle': '关于公司高级管理人员辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000030',
 'secName': '富奥股份',
 'orgId': 'gssz0000030',
 'announcementId': '1218095768',
 'announcementTitle': '关于公司董事辞职的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218095768.PDF',
 'adjunctSize': 56,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
```

```
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '富奥股份',
 'shortTitle': '关于公司董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000158',
 'secName': '常山北明',
 'orgId': 'gssz0000158',
 'announcementId': '1218095021',
 'announcementTitle': '关于职工代表监事辞职及补选职工代表监事的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218095021.PDF',
 'adjunctSize': 165,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '常山北明',
 'shortTitle': '关于职工代表监事辞职及补选职工代表监事的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002238',
 'secName': '天威视讯',
```

```
 'orgId': '9900004647',
 'announcementId': '1218094640',
 'announcementTitle': '天威视讯关于高级管理人员辞职的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218094640.PDF',
 'adjunctSize': 152,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '天威视讯',
 'shortTitle': '天威视讯关于高级管理人员辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300686',
 'secName': '智动力',
 'orgId': '9900029630',
 'announcementId': '1218093982',
 'announcementTitle': '2023-039 智动力: 关于独立董事辞职的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218093982.PDF',
 'adjunctSize': 68,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
```

```
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '智动力',
 'shortTitle': '2023-039 智动力: 关于独立董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300513',
 'secName': '恒实科技',
 'orgId': '9900026672',
 'announcementId': '1218093902',
 'announcementTitle': '关于独立董事辞职及提名独立董事候选人、监事辞职及补选监事的公
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218093902.PDF',
 'adjunctSize': 100,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '恒实科技',
 'shortTitle': '关于独立董事辞职及提名独立董事候选人、监事辞职及补选监事的公告',
```

```
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002818',
 'secName': '富森美',
 'orgId': '9900026774',
 'announcementId': '1218093786',
 'announcementTitle': '关于职工代表监事辞职及补选职工代表监事的公告',
 'announcementTime': 1697731200000,
 'adjunctUrl': 'finalpage/2023-10-20/1218093786.PDF',
 'adjunctSize': 233,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '富森美',
 'shortTitle': '关于职工代表监事辞职及补选职工代表监事的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '301370',
 'secName': '国科恒泰',
 'orgId': '9900035522',
 'announcementId': '1218093153',
 'announcementTitle': '关于监事辞职的公告',
 'announcementTime': 1697709263000,
```

```
 'adjunctUrl': 'finalpage/2023-10-19/1218093153.PDF',
 'adjunctSize': 105,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '国科恒泰',
 'shortTitle': '关于监事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300337',
 'secName': '银邦股份',
 'orgId': '9900022204',
 'announcementId': '1218092322',
 'announcementTitle': '银邦股份关于独立董事辞职的公告',
 'announcementTime': 1697707584000,
 'adjunctUrl': 'finalpage/2023-10-19/1218092322.PDF',
 'adjunctSize': 87,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
```

```
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '银邦股份',
 'shortTitle': '银邦股份关于独立董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300849',
 'secName': '锦盛新材',
 'orgId': '9900035518',
 'announcementId': '1218094124',
 'announcementTitle': '关于公司非独立董事辞职的公告',
 'announcementTime': 1697707223000,
 'adjunctUrl': 'finalpage/2023-10-19/1218094124.PDF',
 'adjunctSize': 107,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '锦盛新材',
 'shortTitle': '关于公司非独立董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002761',
```

```
 'secName': '浙江建投',
 'orgId': '9900023671',
 'announcementId': '1218078979',
 'announcementTitle': '浙江省建设投资集团股份有限公司关于公司董事长辞职暨推举董事代
 'announcementTime': 1697644800000,
 'adjunctUrl': 'finalpage/2023-10-19/1218078979.PDF',
 'adjunctSize': 139,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '浙江建投',
 'shortTitle': '浙江省建设投资集团股份有限公司关于公司董事长辞职暨推举董事代行董事长
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '001286',
 'secName': '陕西能源',
 'orgId': '9900048369',
 'announcementId': '1218074786',
 'announcementTitle': '陕西能源投资股份有限公司关于董事辞职的公告',
 'announcementTime': 1697644800000,
 'adjunctUrl': 'finalpage/2023-10-19/1218074786.PDF',
 'adjunctSize': 150,
 'adjunctType': 'PDF',
 'storageTime': None,
```

```
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '陕西能源',
 'shortTitle': '陕西能源投资股份有限公司关于董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '300157',
 'secName': '新锦动力',
 'orgId': '9900015787',
 'announcementId': '1218076867',
 'announcementTitle': '关于公司董事辞职的公告',
 'announcementTime': 1697622747000,
 'adjunctUrl': 'finalpage/2023-10-18/1218076867.PDF',
 'adjunctSize': 110,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '新锦动力',
```

```
 'shortTitle': '关于公司董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002201',
 'secName': '正威新材',
 'orgId': '9900003944',
 'announcementId': '1218072521',
 'announcementTitle': '关于董事会秘书辞职的公告',
 'announcementTime': 1697600664000,
 'adjunctUrl': 'finalpage/2023-10-18/1218072521.PDF',
 'adjunctSize': 144,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '正威新材',
 'shortTitle': '关于董事会秘书辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '000755',
 'secName': '山西路桥',
 'orgId': 'gssz0000755',
 'announcementId': '1218061388',
 'announcementTitle': '关于独立董事辞职的公告',
```

```
 'announcementTime': 1697558400000,
 'adjunctUrl': 'finalpage/2023-10-18/1218061388.PDF',
 'adjunctSize': 221,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '山西路桥',
 'shortTitle': '关于独立董事辞职的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '301188',
 'secName': '力诺特玻',
 'orgId': 'gfbj0833017',
 'announcementId': '1218059956',
 'announcementTitle': '关于公司总经理、部分董事、董事长辞职暨聘任总经理、补选董事、
 'announcementTime': 1697558400000,
 'adjunctUrl': 'finalpage/2023-10-18/1218059956.PDF',
 'adjunctSize': 198,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012301||012303',
 'associateAnnouncement': None,
```

```
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '力诺特玻',
 'shortTitle': '关于公司总经理、部分董事、董事长辞职暨聘任总经理、补选董事、选举董事
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '301188',
 'secName': '力诺特玻',
 'orgId': 'gfbj0833017',
 'announcementId': '1218059949',
 'announcementTitle': '关于公司监事会主席辞职暨补选监事的公告',
 'announcementTime': 1697558400000,
 'adjunctUrl': 'finalpage/2023-10-18/1218059949.PDF',
 'adjunctSize': 94,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||160203||250301||251302',
 'pageColumn': 'SZCY',
 'announcementType': '01010503||010112||010115||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '力诺特玻',
 'shortTitle': '关于公司监事会主席辞职暨补选监事的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
```

```
 'secCode': '002833',
 'secName': '弘亚数控',
 'orgId': '9900029438',
 'announcementId': '1218059468',
 'announcementTitle': '关于高级管理人员辞职暨聘任高级管理人员的公告',
 'announcementTime': 1697558400000,
 'adjunctUrl': 'finalpage/2023-10-18/1218059468.PDF',
 'adjunctSize': 252,
 'adjunctType': 'PDF',
 'storageTime': None,
 'columnId': '09020202||250101||251302',
 'pageColumn': 'SZZB',
 'announcementType': '01010503||010112||012303',
 'associateAnnouncement': None,
 'important': None,
 'batchNum': None,
 'announcementContent': '',
 'orgName': None,
 'tileSecName': '弘亚数控',
 'shortTitle': '关于高级管理人员辞职暨聘任高级管理人员的公告',
 'announcementTypeName': None,
 'secNameList': None},
{'id': None,
 'secCode': '002771',
 'secName': '真视通',
 'orgId': '9900023511',
 'announcementId': '1218059327',
 'announcementTitle': '关于公司副总经理辞职的公告',
 'announcementTime': 1697558400000,
 'adjunctUrl': 'finalpage/2023-10-18/1218059327.PDF',
 'adjunctSize': 255,
 'adjunctType': 'PDF',
```

```
    'storageTime': None,
    'columnId': '09020202||250101||251302',
    'pageColumn': 'SZZB',
    'announcementType': '01010503||010112||012303',
    'associateAnnouncement': None,
    'important': None,
    'batchNum': None,
    'announcementContent': '',
    'orgName': None,
    'tileSecName': '真视通',
    'shortTitle': '关于公司副总经理辞职的公告',
    'announcementTypeName': None,
    'secNameList': None}],
  'categoryList': None,
  'hasMore': True,
  'totalpages': 3}
```

```
content = r.json()
record = content["totalAnnouncement"]
record
```

```
95
```

```
df = pd.DataFrame(content["announcements"])
df.head()
```

	id	secCode	secName	orgId	announcementId	announcementTitle
0	None	002939	长城证券	qsgn0000030	1218108610	关于公司董事辞职的公告
1	None	300801	泰和科技	9900031706	1218108177	关于独立董事辞职暨补选独立董事
2	None	000420	吉林化纤	gssz0000420	1218108160	关于董事、董事会秘书辞职的公告
3	None	001965	招商公路	GD008065	1218108158	关于高级管理人员辞职的公告
4	None	002075	沙钢股份	9900001101	1218107902	关于职工代表监事辞职及补选的公

http://static.cninfo.com.cn/finalpage/2021-11-22/1211672004.PDF

```
row = df.loc[0,:]
print(type(row))
row
```

```
<class 'pandas.core.series.Series'>

id                                                    None
secCode                                             002939
secName                                               长城证券
orgId                                          qsgn0000030
announcementId                                  1218108610
announcementTitle                               关于公司董事辞职的公告
announcementTime                           1697817600000
adjunctUrl              finalpage/2023-10-21/1218108610.PDF
adjunctSize                                           211
adjunctType                                           PDF
storageTime                                         None
columnId                          09020202||250101||251302
pageColumn                                          SZZB
announcementType                  01010503||010112||012303
associateAnnouncement                               None
important                                           None
batchNum                                            None
announcementContent
orgName                                             None
tileSecName                                           长城证券
shortTitle                                      关于公司董事辞职的公告
announcementTypeName                                None
secNameList                                         None
Name: 0, dtype: object
```

```
Path("pdf").mkdir(exist_ok=True)

pdf_url = f"http://static.cninfo.com.cn/{row.adjunctUrl}"
local_pdf = f'pdf/{row.secName}_{row.announcementId}.pdf'
```

```
from urllib.request import urlretrieve

urlretrieve(pdf_url, local_pdf)

print(f"download from {pdf_url} to local file: {local_pdf}")
```

```
download from http://static.cninfo.com.cn/finalpage/2023-10-21/1218108610.PDF to loca
```

9.3 模拟浏览器

Playwright 文档

- Playwright Python 入门
- API 文档
- 库模式使用

Playwright 是一个强大的浏览器自动化工具，支持 Chromium、Firefox 和 WebKit。

Jupyter Notebook 中的注意事项

在 Jupyter/Quarto 环境中，由于已存在 asyncio 事件循环，需要使用 异步 **API**。
在普通 Python 脚本中，可以使用同步 API。

9.3.1 异步 API 示例（Notebook 推荐）

```
from playwright.async_api import async_playwright

async def browse_example():
```

```
    async with async_playwright() as p:
        # 启动浏览器 (headless=True 表示无界面模式)
        browser = await p.chromium.launch(headless=True)
        page = await browser.new_page()

        # 访问网页
        await page.goto("https://playwright.dev")

        # 获取标题
        title = await page.title()
        print(f" 页面标题: {title}")

        # 截图
        await page.screenshot(path="./data/playwright_example.png")
        print(" 截图已保存")

        # 关闭浏览器
        await browser.close()

# 在 Jupyter 中直接运行异步函数
await browse_example()
```

9.3.2 完整的网页自动化示例

```
from playwright.async_api import async_playwright
from pathlib import Path

async def search_and_save():
    async with async_playwright() as p:
        # 启动浏览器
        browser = await p.chromium.launch(headless=True)
```

```
        page = await browser.new_page()

        # 访问百度首页
        await page.goto('https://www.baidu.com')
        print("  已打开百度首页")

        # 搜索" 暨南大学管理学院"
        await page.fill('input[name="wd"]', '暨南大学管理学院')
        await page.click('input[type="submit"]')
        print("  已提交搜索")

        # 等待搜索结果加载
        await page.wait_for_load_state('networkidle')

        # 获取页面标题
        title = await page.title()
        print(f" 页面标题: {title}")

        # 保存页面内容
        html_content = await page.content()
        Path('search_results.html').write_text(html_content,
↪  encoding='utf-8')
        print("  页面内容已保存到 search_results.html")

        # 截图
        await page.screenshot(path='search_screenshot.png')
        print("  截图已保存到 search_screenshot.png")

        # 关闭浏览器
        await browser.close()

# 运行
```

```
await search_and_save()
```

9.3.3 同步 API（仅供参考，不在 Notebook 中使用）

> **Warning**
>
> 以下代码仅适用于普通 Python 脚本，不要在 **Jupyter Notebook** 中运行。

```
# 仅在普通 .py 脚本中使用
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch(headless=False)
    page = browser.new_page()
    page.goto("https://playwright.dev")
    print(page.title())
    browser.close()
```

9.4 解析 HTML

```
import requests
headers = {'User-Agent':
           'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
            ↪  (KHTML, like Gecko) Chrome/79.0.3945.130 Safari/537.36
            ↪  OPR/66.0.3515.115'}
url = 'https://book.douban.com/tag/小说?start=0&type=T'
resp = requests.get(url, headers = headers)   # 极少数情况会用到
 ↪  verify=False 这参数
resp
```

```
<Response [200]>
```

```
Path("sample.html").write_text(resp.text)
```

```
53589
```

```
html_string = Path("sample.html").read_text() # local html
```

```
type(html_string)
```

```
str
```

> **HTML 解析库选择**
>
> 常用的 HTML 解析库：
>
> - **Parsel** - 支持 CSS 和 XPath，Scrapy 框架使用的解析库（推荐）
> - **lxml** - 速度快，功能强大
> - **BeautifulSoup** - 功能全面，语法简单
>
> 本教程使用 **Parsel** 库，演示两种选择器用法：CSS 和 XPath。

9.4.1 方法一：使用 Parsel + CSS 选择器

Parsel 是 Scrapy 框架使用的解析库，同时支持 CSS 选择器和 XPath。

```
from parsel import Selector

# 创建 Selector 对象
selector = Selector(text=html_string)
print(f"Selector 类型: {type(selector)}")
```

```
Selector 类型: <class 'parsel.selector.Selector'>
```

CSS 选择器语法

为了定位 html 中对应的节点及其属性和含有的信息，我们需要使用选择器。

CSS 选择器	例子	解释
.class	.intro	选出 class="intro" 的节点
#id	#firstname	选出 id="firstname" 的节点
element	p	选出所有 p 标签的节点
element element	div p	选出 div 节点后代所有 p 节点
[attribute]	[target]	选出带有 target 属性所有节点
[attribute=value]	[target=_blank]	选出 target="_blank" 的所有节点

> **SelectorGadget 工具**
>
> 使用 Chrome 扩展 SelectorGadget 可以快速获取元素的 CSS 选择器。

Parsel CSS 选择器示例

```
# 选择所有书籍条目
items = selector.css('.subject-item')
print(f" 找到 {len(items)} 个书籍条目")
```

```
找到 20 个书籍条目
```

```
# 获取第一个书籍条目
first_item = selector.css('.subject-item')[0]
print(first_item)
```

```
<li class="subject-item">
    <div class="pic">
      <a class="nbg" href="https://book.douban.com/subject/37339619/" onclick="moreur
        <img class="" src="https://img2.doubanio.com/view/subject/s/public/s35162221.
```

```
    </a>
  </div>
  <div class="info">
    <h2 class="">


<a href="https://book.douban.com/subject/37339619/" title="桃花源没事儿" onclick="m

  桃花源没事儿




</a>

    </h2>
    <div class="pub">


马伯庸 / 湖南文艺出版社 / 2025-6 / 48.00元

    </div>




<div class="star clearfix">
      <span class="allstar40"></span>
      <span class="rating_nums">7.6</span>
```

```
  <span class="pl">
      (12983人评价)
  </span>
</div>
```

<p>小道士玄穹是天生穷命，只要一捞横财，必有天雷劈下。他啥钱也不敢挣，只能去桃花源

```
    <div class="ft">

<div class="collect-info">
</div>

        <div class="cart-actions">

  <span class="buy-info">
    <a href="https://book.douban.com/subject/37339619/buylinks">
      纸质版 35.00元
```

```
      </a>
    </span>

          </div>
```

```
  <div class="ebook-link">
    <a target="_blank" href="https://read.douban.com/ebook/648940719/?dcs=tag-buylink
  </div>
```

```
      </div>

    </div>
  </li>
```

```
# 遍历所有书籍，提取信息
for item in items[:3]:  # 只看前 3 个
    # 提取书名（使用 ::text 伪元素获取文本）
    title = item.css('a[title]::attr(title)').get()
    # 提取评分
    rating = item.css('.rating_nums::text').get()
    # 提取评价人数
    pl = item.css('.pl::text').get()

    print(f" 书名: {title}")
    print(f" 评分: {rating}")
    print(f" 评价: {pl}")
```

```
    print("-" * 50)
```

书名：桃花源没事儿
评分：7.6
评价：
        (12983人评价)

书名：长安的荔枝
评分：8.5
评价：
        (261869人评价)

书名：太白金星有点烦
评分：9.0
评价：
        (139370人评价)

```
# 提取图片 URL
img_url = selector.css('.subject-item img::attr(src)').get()
print(f" 图片 URL: {img_url}")
```

图片URL: https://img2.doubanio.com/view/subject/s/public/s35162221.jpg

```
# 提取所有书名（返回列表）
all_titles = selector.css('.subject-item a[title]::attr(title)').getall()
print(f" 共找到 {len(all_titles)} 本书")
print(all_titles[:5])  # 显示前 5 本
```

```
共找到 20 本书
['桃花源没事儿', '长安的荔枝', '太白金星有点烦', '食南之徒', '一句顶一万句']
```

9.4.2 方法二：使用 Parsel + XPath

XPath 是更强大的选择器语言，可以进行复杂的节点查询。

> **XPath vs CSS**
>
> - **CSS 选择器**：简洁直观，适合简单查询
> - **XPath**：功能更强大，支持复杂的逻辑判断和层级关系
>
> 推荐：简单场景用 CSS，复杂场景用 XPath。

XPath 基础语法

XPath 表达式	说明
/	从根节点选取
//	从任意位置选取
.	当前节点
..	父节点
@	选取属性
text()	选取文本内容

```
# 使用 XPath 选择所有书籍条目
items_xpath = selector.xpath('//li[@class="subject-item"]')
print(f"XPath 找到 {len(items_xpath)} 个书籍条目")
```

```
XPath 找到 20 个书籍条目
```

```
# 使用 XPath 提取第一本书的信息
first_book = items_xpath[0]

# 提取书名
```

```
title_xpath = first_book.xpath('.//a[@title]/@title').get()
# 提取评分
rating_xpath =
 ↪  first_book.xpath('.//span[@class="rating_nums"]/text()').get()
# 提取出版信息
pub_xpath = first_book.xpath('.//div[@class="pub"]/text()').get()

print(f" 书名: {title_xpath}")
print(f" 评分: {rating_xpath}")
print(f" 出版信息: {pub_xpath}")
```

书名: 桃花源没事儿
评分: 7.6
出版信息:

马伯庸 / 湖南文艺出版社 / 2025-6 / 48.00元

```
# XPath 高级用法: 条件筛选
# 选择评分大于等于 8.0 的书籍 (需要将评分转为数字比较)
high_rated_books = []

for item in items_xpath:
    rating = item.xpath('.//span[@class="rating_nums"]/text()').get()
    title = item.xpath('.//a[@title]/@title').get()

    if rating and float(rating) >= 8.0:
        high_rated_books.append({
            'title': title,
            'rating': rating
```

```
        })

print(f" 评分 >= 8.0 的书籍共 {len(high_rated_books)} 本:")
for book in high_rated_books[:5]:
    print(f"  {book['title']}: {book['rating']}")
```

```
评分 >= 8.0 的书籍共 18 本:
  长安的荔枝: 8.5
  太白金星有点烦: 9.0
  食南之徒: 8.2
  一句顶一万句: 9.0
  素食者: 8.1
```

```
# XPath 提取所有图片 URL
img_urls_xpath =
 ↪  selector.xpath('//li[@class="subject-item"]//img/@src').getall()
print(f" 共找到 {len(img_urls_xpath)} 个图片")
print(img_urls_xpath[:3])
```

```
共找到 20 个图片
['https://img2.doubanio.com/view/subject/s/public/s35162221.jpg', 'https://img3.douba
```

CSS vs XPath 对比示例

```
import pandas as pd

# 创建对比数据
comparison_data = []

for i, item in enumerate(items[:5], 1):
    # CSS 方式
    title_css = item.css('a[title]::attr(title)').get()
```

```
    rating_css = item.css('.rating_nums::text').get()

    # XPath 方式
    title_xpath = items_xpath[i-1].xpath('.//a[@title]/@title').get()
    rating_xpath =
↪   items_xpath[i-1].xpath('.//span[@class="rating_nums"]/text()').get()

    comparison_data.append({
        '序号': i,
        '书名 (CSS)': title_css,
        '评分 (CSS)': rating_css,
        '书名 (XPath)': title_xpath,
        '评分 (XPath)': rating_xpath,
        '结果一致': title_css == title_xpath and rating_css ==
        ↪ rating_xpath
    })

df_comparison = pd.DataFrame(comparison_data)
print(df_comparison)
```

```
   序号  书名(CSS) 评分(CSS) 书名(XPath) 评分(XPath)  结果一致
0   1   桃花源没事儿     7.6    桃花源没事儿       7.6  True
1   2    长安的荔枝     8.5     长安的荔枝       8.5  True
2   3  太白金星有点烦     9.0   太白金星有点烦       9.0  True
3   4     食南之徒     8.2      食南之徒       8.2  True
4   5   一句顶一万句     9.0    一句顶一万句       9.0  True
```

Parsel 方法速查

CSS 选择器方法： - `selector.css('div')` - 选择所有 div 元素 - `selector.css('div::text').get()` - 获取第一个 div 的文本 - `selector.css('div::text').getall()` - 获取所有 div 的文本 - `selector.css('a::attr(href)').get()` - 获取第一个 a 标签的 href 属性

XPath 方法： - `selector.xpath('//div')` - 选择所有 div 元素 - `selector.xpath('//div/text()').get()` - 获取第一个 div 的文本 - `selector.xpath('//div/text()').getall()` - 获取所有 div 的文本 - `selector.xpath('//a/@href').get()` - 获取第一个 a 标签的 href 属性

9.4.3 下载图片示例

```
url = selector.css('.subject-item img::attr(src)').get()
print(f" 图片 URL: {url}")
```

图片 URL: https://img2.doubanio.com/view/subject/s/public/s35162221.jpg

```
from urllib.request import urlretrieve
from pathlib import Path

# 设置本地文件名
local = f"{Path(url).name}"
print(f" 保存路径: {local}")
```

保存路径: s35162221.jpg

```
# 下载图片
urlretrieve(url, local)
print(f"  图片已下载到: {local}")
```

图片已下载到: s35162221.jpg

9.5 总结

本章介绍了网页爬虫的基础知识：

1. **HTTP 请求**：使用 `requests` 库获取网页内容

2. **HTML 解析**：使用 `parsel` 库的两种方式
    - **CSS 选择器**：简洁直观，适合大多数场景
    - **XPath**：功能强大，适合复杂查询
3. **浏览器自动化**：使用 `playwright` 处理 JavaScript 动态内容

爬虫注意事项

1. **遵守 robots.txt**：检查网站的爬虫协议
2. **控制请求频率**：添加延时，避免给服务器造成压力
3. **尊重版权**：仅用于学习和研究，不用于商业用途
4. **数据隐私**：不爬取个人隐私信息

推荐学习资源

- Parsel 官方文档
- XPath 教程
- CSS 选择器参考
- Scrapy 框架（基于 Parsel 的强大爬虫框架）