基于 Python 的经济数据分析与应用

于明超

2025-07-28

Table of Contents

前言

本书是为南京师范大学商学院国际商务、数字经济专业硕士准备的课程讲义。目的是掌握基本的数据分析方法，为从事科学研究、论文写作奠定基础。

本书特色：

- 将 Python 与经济分析紧密结合；
- 应用实际数据集。采用的 Penn World Table、CFPS 数据集、CHNS 等，无论是宏观还是微观数据，都是实际学术研究中常使用的数据集。
- 适合高年级本科生和研究生使用。分析方法是学术期刊经常出现，如熵权法、泰尔指数分解、出口产品复杂度等。

本书内容来自教学实践，囿于作者水平，难免出现疏漏错误，欢迎批评指正。

第一章　引言

《基于 Python 的经济分析与应用》课程专为国际商务专业硕士研究生设计，注重理论与实践相结合。课程内容涵盖 Python 编程基础、数据采集与处理、经济数据分析、可视化展示及实际案例应用。通过本课程，学生能够掌握利用 Python 进行经济数据分析的方法，提高数据处理与决策支持能力，增强在国际商务环境中的实际应用能力，为未来从事数据驱动的经济分析和国际商务决策打下坚实基础。

1.1　课程是关于什么的？

1.1.1　涉及课程

- 理论模型？
- 实证分析？
- 数据分析？
- 经济学？
- 产业经济学
- 国际贸易？

1.1.2　经济数据分析

- 宏观经济形式分析。如北京大学汇丰商学院的经济分析。毕马威的中国经济观察季度报告等。

- 中国宏观经济论坛
- 美国经济分析局（Bureau of Economic Analysis，BEA）：The Bureau of Economic Analysis (BEA) promotes a better understanding of the U.S. economy by providing the most timely, relevant, and accurate economic accounts data in an objective and cost-effective manner.
- 数据科学
- 数据服务商

1.1.3 本课程的主要内容

围绕经济学、国际贸易中有关主题展开，主要包括：

- 经济数据分析：如增长、不平等等、通货膨胀等宏观数据；
- 统计分析方法：t 检验、方差分析等；
- 线性回归方法
- 蒙特卡洛模拟分析
- 机器学习基础方法
- 投入产出模型；
- 网络分析方法；

1.1.4 主要参考书

会用到部分 Python 有关的内容，如：

- McKinney [2]，在线阅读
- VanderPlas [3]，在线阅读
- Python Programming for Economics and Finance

1.2 分析工具：Python

我们使用 Python 作为主要的分析工具。根据TIOBE Index，Python 是目前最流行的编程语言。

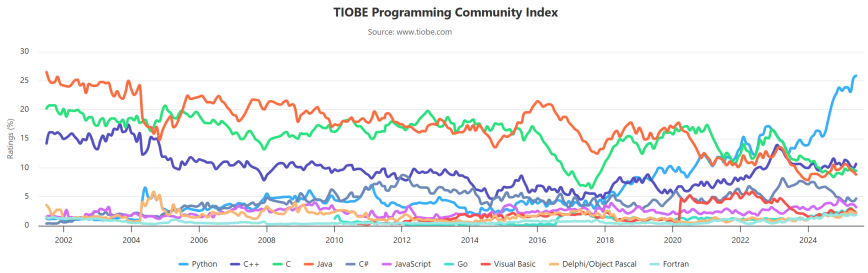


Figure 1.1: TIOBE Programming Community Index

1.2.1 功能强大，应用广泛

Python 广泛应用于机器学习、科学计算等各个领域：

1. 机器学习
2. 数据科学
3. 通讯
4. 网页开发
5. CGI and GUI
6. 自然语言处理
7. 游戏开发
8. 等等

1.2.2 Python 的特点

Python 具有许多优点：

1. 易读、易写和易调试；
2. 核心内容易学；
3. 众多库的支持；
4. 初学者友好
5. 支持多平台
6. 网络资源众多

1.3 安装 Python

1.3.1 下载安装 Python

1. 自官方网站下载 Python，当前版本 3.13.5
2. 在点击 “Install Now” 安装程序之前，Windows 系统**注意**勾选：Add Python to PATH，将 Python 的安装路径添加到操作系统的环境变量 Path 中；

Figure 1.2: 将 Python 添加至环境

3. 在 “命令提示符” 输入 `python --version` 查看安装版本。
4. 安装JupyterLab。通过命令提示符（或 Mac OS 的终端）安装：`pip install jupyterlab`。另外，需要经常使用 pip 安装，建议配置镜像源为清华大学开源软件镜像站。
5. 推荐安装Visual Code Studio，然后在扩展（Extensions Marketplace）搜索安装：

- Python
- Jupyter

- Quarto

1. 如果只使用 JupyterLab，在命令提示符输入 `jupyter lab` 即可启动；
2. 另外，安装Anaconda也是非常常见的方式，会同时方便的安装常用 Python 程序包。
3. 需要安装第三方程序包时，在命令提示符使用 pip 安装，例如：

- Numpy: `pip install numpy`
- Pandas: `pip install pandas`
- Matplotlib: `pip install matplotlib`

1.4 脚本模式和交互模式

1.4.1 交互式

用户输入代码，回车运行。在如 IDLE，Ipython 都可以方便地进行交互式操作。例如：

- 在命令提示符输入 `ipython`，将打开 Ipython 的界面，输入 `3 + 3`，回车，将显示计算结果；
- 在应用程序中打开 IDLE Shell，也可以方便的进行交互式操作；
- 在 Jupyter Notebook 的代码单元格内，输入代码，点击运行显示结果；

1.4.2 脚本模式

将代码保存在.py 格式文件中，使用命令提示符运行；

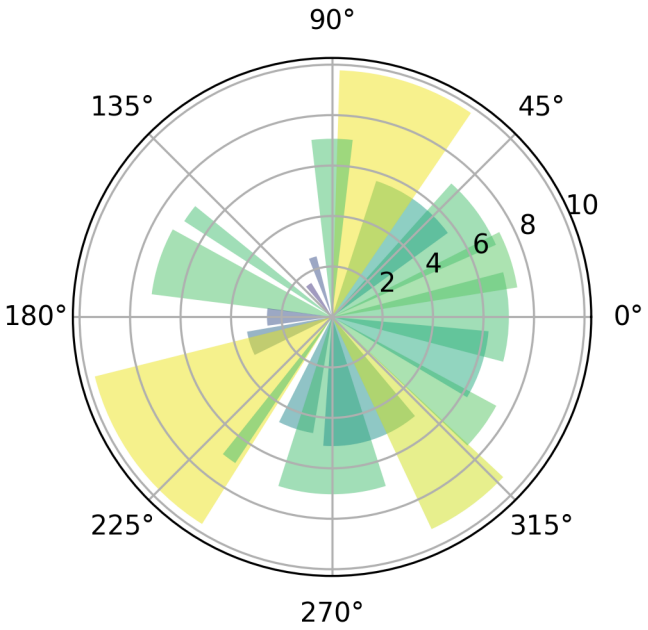
例如，在文件夹 pyfiles 中保存有一个文件，存储了下面绘制圆形柱状图的代码。在命令提示符或终端中运行：

```
python pyfiles/polar_plot.py
```

当然，也可以先改变当前文件夹至 pyfiles 文件夹，就可以省略路径。### Jupyter notebook 兼有

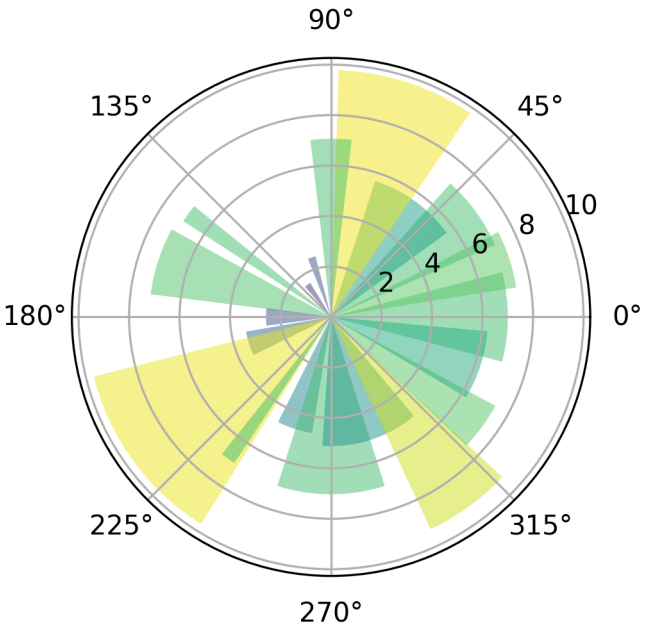
在 Jupyter Notebook 中试验是否成功安装，打开链接中的例子，将代码复制到一个单元格中，绘制一个圆形柱状图：

```
import numpy as np
import matplotlib.pyplot as plt


# Fixing random state for reproducibility
np.random.seed(19680801)

# Compute pie slices
N = 20
theta = np.linspace(0.0, 2 * np.pi, N, endpoint=False)
radii = 10 * np.random.rand(N)
width = np.pi / 4 * np.random.rand(N)
colors = plt.cm.viridis(radii / 10.)

ax = plt.subplot(111, projection='polar')
ax.bar(theta, radii, width=width, bottom=0.0, color=colors, alpha=0.5)

plt.show()
```



Jupyter Notebook 延续了 ipython 中的`%run` 命令，可以脚本模式运行：

```
%run pyfiles/polar_plot.py
```



1.5 文档与帮助

1.5.1 help() 和?

Python 有非常详细的官方帮助文档，帮助新用户快速的熟悉其用法。

例如，Python 有一个内置函数 `help()`，可以查看定义的文档，例如对函数 `len()`：

```
help(len)
```

```
Help on built-in function len in module builtins:

len(obj, /)
    Return the number of items in a container.
```

由于其重要性，Ipython 和 Jupyter 中可以使用? 作为缩写：

```
len?
```

当然，对自定义的对象也是适用的。下面定义的函数，有一段函数的说明文字（docstring）：

```
def square(x):
    """
    Calculates the square of a given number.

    Args:
        x (int or float): The number to be squared.

    Returns:
        int or float: The square of the input number.
    """
    return x**2
```

如果输入 help() 函数：

```
help(square)
```

```
Help on function square in module __main__:

square(x)
    Calculates the square of a given number.

    Args:
        x (int or float): The number to be squared.

    Returns:
        int or float: The square of the input number.
```

第二章　Python 基础

这一部分，将简要回归 Python 语言的基本内容，包括变量的类型、条件语句、循环、自定义函数和类等内容。

2.1　变量与数据类型

变量是用来存储数据的 “容器”，可以赋不同类型的值。Python 常见数据类型有：整数 (int)、浮点数 (float)、字符串 (str)、布尔 (bool) 等。

2.1.1　变量赋值

变量赋值把一个具体的值存储到一个变量中，方便后续使用和操作。要注意变量命名的规则：

- 变量名只能包含字母、数字和下划线，且不能以数字开头
- 区分大小写（如：age 和 Age 是不同变量）
- 不可使用 Python 关键字作为变量名（如：if, for, class 等）
- 建议使用有意义的英文单词，遵循小写加下划线的风格（如：student_name）

如果命名方式不符合要求，软件将返回错误。将下面例子中表示注释的去掉，试运行语句，看提示的错误类型是什么？

```
# 1student = "Bob"      # 不能以数字开头
# class = "Math"        # 不能使用关键字
```

```
# student-name = "Tom"  # 不能包含减号
# t&2 = 30              # 特殊符号
```

例如，正确的变量名：

```
name = "Alice"
age = 21
province = "Jiang Su"
```

2.1.2 保留关键词

上面的 class，属于软件内部保留的 33 个关键词之一，注意在命名时应避免与关键词冲突：

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class
```

2.1.3 常见数据类型：

整数（int）：用于表示整数类型的数据。

可以通过 type() 函数查看一个对象的类型。

```
a = -5
print(type(a))
#
b = 10
c = b/a
type(c)
```

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

```
float
```

浮点数（float）：用于表示带小数点的数值

```
pi = 3.14159
radius = 1.75
area = pi * radius **2
print(type(pi))
print(area)
```

```
<class 'float'>
9.621119375
```

字符串（str）：用于表示文本数据

可以用双引号或者单引号定义字符串。当字符串较长时，可以用三引号进行定义。例如：

```
sentence = "Python's syntax is easy to learn."
multiline = """ 这个句子
被我
分成了几行。这样
看起来，像诗一样。
"""
```

字符串有许多属性和方法，如 Table 2.1 所示：

Table 2.1: 常见的字符串方法

方法	功能说明	示例
`s.lower()`	转换为小写	`"Hello".lower()` → `"hello"`
`s.upper()`	转换为大写	`"hello".upper()` → `"HELLO"`
`s.capitalize()`	首字母大写，其余小写	`"hello world".capitalize()` → `"Hello world"`

方法	功能说明	示例
s.title()	每个单词首字母大写	"hello world".title() → "Hello World"
s.strip()	去除字符串首尾空白字符	" hello ".strip() → "hello"
s.lstrip()	去除左侧空白	" hello".lstrip() → "hello"
s.rstrip()	去除右侧空白	"hello ".rstrip() → "hello"
s.replace(old, new)	替换子串	"banana".replace("a", "o") → "bonono"
s.find(sub)	查找子串位置，找不到返回 -1	"hello".find("l") → 2
s.count(sub)	统计子串出现次数	"banana".count("a") → 3
s.startswith(prefix)	是否以指定前缀开头	"hello".startswith("he") → True
s.endswith(suffix)	是否以指定后缀结尾	"hello".endswith("lo") → True
s.split(sep)	按分隔符拆分字符串	"a,b,c".split(",") → ["a", "b", "c"]
sep.join(iterable)	使用分隔符连接字符串序列	",".join(["a", "b", "c"]) → "a,b,c"

注意，当对字符串使用某一方法时，虽然结果显示了变化，并没有改变原本

对字符串。如果要保存结果可以赋值。

```
# 例：将日期字符串按“-”分割成年、月、日
date_str = "2024-06-01"
parts = date_str.split("-")
year, month, day = parts
print(f" 年份：{year}，月份：{month}，日期：{day}")

# 例：用 join 方法将列表中的单词拼接成一句话
words = ["Python", "is", "fun"]
sentence = "$#$_".join(words)
print(sentence)
```

```
年份：2024，月份：06，日期：01
Python$#$_is$#$_fun
```

布林类型（bool）：只有 True 和 False 两个值，常用于条件判断

```
name = "Jane"
age = 20
score = 57
is_adult = age >= 18
has_passed = score >= 60

print(type(is_adult))  # <class 'bool'>
print(f" 是否成年：{is_adult}")
print(f" 是否及格：{has_passed}")
```

```
<class 'bool'>
是否成年：True
是否及格：False
```

2.2 print 与 f-string

2.2.1 带引号的字符

上面的例子中，常用 print() 函数将内容输出到屏幕，它是 Python 中最常用的输出语句，它可以输出字符串、变量、表达式等，并支持格式化输出。

在 Python 语言中，字符可以使用单引号或者双引号表示，在输出带引号的字符串时，可以利用该特点。例如

```
# 输出带有引号的字符串
print('She said, "Hello!"')       # 外单内双
print("It's a nice day.")         # 外双内单

# 如果字符串本身包含同样的引号，可以用转义符\避免冲突
print('It\'s a nice day.')        # 单引号中包含单引号
print("She said, \"Hello!\"")     # 双引号中包含双引号

# 三引号可以包含单双引号和多行内容
print(""" 他说: "It's OK!" """)
```

```
She said, "Hello!"
It's a nice day.
It's a nice day.
She said, "Hello!"
他说: "It's OK!"
```

2.2.2 格式化输出 f-string

f-string（格式化字符串字面量）是 Python 3.6 及以上版本提供的一种字符串格式化方式，它让我们可以在字符串中直接嵌入变量或表达式。

输出结果时也可以进行数学运算：

```
year = 2024
GDP_per_capita = 95749
growth = 0.051

my_string = f"{year}年，人均国内生产总值为{GDP_per_capita/10000:.3f}万元，比去年增长{growth:.3%
print(my_string)
```

2024年，人均国内生产总值为9.575万元，比去年增长5.100%

2.3 运算符与表达式

运算符用于对数据进行各种操作，主要包括以下几类：

- 算术运算符：+（加），-（减），*（乘），/（除），//（整除），%（取余），**（幂）
- 比较运算符：==（等于），!=（不等于），>（大于），<（小于），>=（大于等于），<=（小于等于）
- 逻辑运算符：and（与），or（或），not（非）
- 赋值运算符：=（赋值），+=，-=，*=，/= 等
- 成员运算符：in，not in（判断元素是否属于序列）
- 身份运算符：is，is not（判断两个对象是否为同一对象）

下面分别介绍常用的运算符及其用法。

2.3.1 算术运算符

算术运算符包括：加法（+）、减法（-）、乘法（*）、除法（/）、整除（//）、取余（%）、幂运算（**）：

```
a = 15
b = 4

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

```
print("a % b =", a % b)
print("a ** b =", a ** b)
```

```
a // b = 3
a % b = 3
a ** b = 50625
```

2.3.2 比较运算符

比较运算符用于比较两个值，结果为布尔类型（True 或 False）

```
print("a > b:", a > b)
print("a == b:", a == b)
print("a != b:", a != b)
```

```
a > b: True
a == b: False
a != b: True
```

2.3.2.1 逻辑运算符

逻辑运算符用于连接多个条件表达式，常用于复合条件判断。例如：

```
x = 8
y = 3
#
print((x > 5) and (y < 5))
print((x < 5) or (y < 5))
print(not (x > y))
```

```
True
True
False
```

2.3.3 赋值运算符

赋值运算符用于给变量赋值或在原有基础上进行运算后赋值:

```
x = 10
x += 5
x -= 3
x *= 2
x /= 4
x //= 2
x %= 2
x **= 3
#
print(x)
```

```
1.0
```

2.3.4 成员运算符

成员运算符用于判断某个元素是否属于某个序列，如列表、元组、字符串等。例如：

```
for i in range(5):
  print(i)
```

```
0
1
2
3
4
```

2.4 控制结构：条件与循环

2.4.1 条件语句

条件语句（if 语句）用于根据条件判断执行不同的代码块，基本结构：

```
# if 条件:
#     代码块 1
# elif 其他条件:
#     代码块 2
# else:
#     代码块 3
```

以世界银行经济体收入分组标准为例：

世界银行按照人均国民收入把世界各经济体分成四组，如 Table 2.2 所示，中、低收入国家被称为发展中国家，高收入国家被称为发达国家。

Table 2.2: 世界银行经济体收入分组标准

经济体分组	划分标准（人均国民总收入）
低收入经济体	1145 美元以下
中等偏下收入经济体	1146—4515 美元
中等偏上收入经济体	4516—14005 美元
高收入经济体	14005 美元以上

可以使用嵌套条件语句来进行判别：

```
gni = 13660

if gni <= 1135:
    economy = " 低收入经济体"
elif gni <= 4465:
    economy =  " 中等偏下收入经济体"
elif gni <= 13845:
```

```
    economy = " 中等偏上收入经济体"
else:
    economy = " 高收入经济体"

print(f" 人均国民总收入为 {gni} 美元，属于：{economy}")
```

```
人均国民总收入为 13660 美元，属于：中等偏上收入经济体
```

2.4.2 例：BMI 指数

再来看一个计算 BMI（Body Mass Index）指数的例子：

$$BMI = \frac{weight(kg)}{height(m)^2}$$

```
height = 1.75
weight = 75

# 计算 BMI
bmi = weight / (height ** 2)

if bmi < 18.5:
    status = " 偏瘦"
elif bmi < 24:
    status = " 正常"
elif bmi < 28:
    status = " 超重"
else:
    status = " 肥胖"
print(f"BMI = {bmi:.2f}, {status}")
```

```
BMI = 24.49, 超重
```

2.4.3 for 循环与 range

for 循环用于遍历序列（如列表、字符串、元组等）或按照一定范围循环，经常与 range() 一起使用。

```
fruits = ["apple", "banana", "cherry"]
for fruit in fruits:
    print(fruit)
#
total = 0
for i in range(1, 101):
    total += i
print("1 到 100 的和为: ", total)
```

```
apple
banana
cherry
1到100的和为： 5050
```

2.4.4 while 循环

while 循环是一种基于条件判断的循环结构。当条件为 True 时，循环体会反复执行，直到条件变为 False 为止。

while 循环要注意循环变量的更新，否则可能会造成死循环。

```
# 例 1: 输出 1 到 5
i = 1
while i <= 5:
    print(i)
    i += 1

# 例 2: 计算 1 到 100 的和
total = 0
n = 1
```

```
while n <= 100:
    total += n
    n += 1
print("1 到 100 的和为：", total)
```

```
1
2
3
4
5
1到100的和为： 5050
```

2.5 容器类型：列表、字典、元组、集合

列表（list）、字典（dict）、元组（tuple）、集合（set）是 Python 中常用的容器类型。

- 列表：用 [] 定义，有序可变
- 字典：用 {} 定义，键值对结构
- 元组：用 () 定义，有序不可变
- 集合：用 set() 或 {} 定义，无序不重复

2.5.1 列表

Table 2.3 中列出了常用的列表方法，假设初始列表为 `lst = [1, 2, 3]`：

Table 2.3: 列表的常用方法

方法	功能说明	示例	结果
`append(x)`	在列表末尾添加元素 x	`lst.append(4)`	`[1, 2, 3, 4]`
`extend(iterable)`	将可迭代对象中的元素添加到列表末尾	`lst.extend([4, 5])`	`[1, 2, 3, 4, 5]`

方法	功能说明	示例	结果
`insert(i, x)`	在索引 i 处插入元素 x	`lst.insert(1, 10)`	`[1, 10, 2, 3]`
`remove(x)`	删除列表中第一个值为 x 的元素	`lst.remove(2)`	`[1, 3]`
`pop([i])`	移除并返回指定索引的元素，默认删除最后一个	`lst.pop()` / `lst.pop(0)`	3 / 1
`clear()`	清空列表所有元素	`lst.clear()`	`[]`
`index(x)`	返回第一个值为 x 的索引	`lst.index(3)`	2
`count(x)`	返回元素 x 在列表中出现的次数	`lst.count(2)`	1
`sort()`	原地排序（默认升序）	`lst.sort()`	`[1, 2, 3]`
`sort(reverse=True)`	原地降序排序	`lst.sort(reverse=True)`	`[3, 2, 1]`
`reverse()`	原地反转列表顺序	`lst.reverse()`	`[3, 2, 1]`
`copy()`	返回列表的浅拷贝	`new_lst = lst.copy()`	`[1, 2, 3]`

```
# 列表（list）：有序、可变的元素集合，用 [] 表示
numbers = [10, 20, 30, 40]
fruits = ["apple", "banana", "cherry"]

# 访问元素（索引从 0 开始）
print(numbers[0])      # 10
print(fruits[-1])      # cherry（倒数第 1 个）

# 修改元素
numbers[1] = 25
print(numbers)         # [10, 25, 30, 40]

# 添加元素
```

```
fruits.append("orange")          # 末尾添加
fruits.insert(1, "pear")         # 指定位置插入
print(fruits)                    # ['apple', 'pear', 'banana', 'cherry', 'orange']

# 删除元素
del numbers[2]                   # 按索引删除
fruits.remove("banana")          # 按值删除
print(numbers)                   # [10, 25, 40]
print(fruits)                    # ['apple', 'pear', 'cherry', 'orange']

# 列表切片
print(numbers[1:])               # [25, 40]
print(fruits[:2])                # ['apple', 'pear']

# 遍历列表
for fruit in fruits:
    print(fruit)

# 列表常用方法
print(len(numbers))              # 长度
print(max(numbers))              # 最大值
print(min(numbers))              # 最小值
print(numbers.count(25))         # 出现次数
numbers.sort()                   # 排序
print(numbers)
```

```
10
cherry
[10, 25, 30, 40]
['apple', 'pear', 'banana', 'cherry', 'orange']
[10, 25, 40]
['apple', 'pear', 'cherry', 'orange']
[25, 40]
```

```
['apple', 'pear']
apple
pear
cherry
orange
3
40
10
1
[10, 25, 40]
```

```python
# 元组（tuple）：有序、不可变的元素集合，用 () 表示
point = (3, 4)
colors = ("red", "green", "blue")

# 访问元素
print(point[0])                 # 3
print(colors[-1])               # blue

# 元组不可修改
# point[1] = 5  # 会报错

# 单元素元组要加逗号
single = (5,)
print(type(single))             # <class 'tuple'>

# 元组可以用于多变量赋值
x, y = point
print(x, y)                     # 3 4

# 遍历元组
for color in colors:
    print(color)
```

```
3
blue
<class 'tuple'>
3 4
red
green
blue
```

2.5.2 例：计算净现值

如果某项资产在多个时间周期内支付一系列收益流，那么我们可以使用**贴现率**来计算这整个收益序列对消费者的当前价值。

更一般地说，我们用索引 t 来表示每一个离散的时间周期（例如年、月、日），其中“今天”为第 $t = 0$ 期，资产一共存在 T 个时间周期。

我们用 y_t 表示第 t 期的收益，**并假设这些收益现在是已知的**。

如果贴现因子为 $r \geq 0$，那么消费者对第 t 期收到的收益 y_t 的当前“价值”是：

$$\frac{1}{(1+r)^t} y_t$$

需要注意的是，当 $t = 0$ 时，当前价值就是 y_0。

基于这一逻辑，我们可以用一个求和表达式来表示整条收益序列的总价值：

$$P_0 = \sum_{t=0}^{T} (\frac{1}{1+r})^t y_t$$

我们将假设的收益序列保存为列表：`y = [-100, 80, 60, 50]`，贴现率假设为 0.05：

```
y = [-100, 80, 60, 50]
r = 0.05
```

```
P0 = 0
for t in range(len(y)):
    P0 = P0 + y[t] / (1 + r) ** t
print(f"The Total Present Value = : {P0:.2f}")
```

```
The Total Present Value = : 73.80
```

2.5.3 字典的键值对操作

字典（dict）是一种用于存储键值对的数据结构。每个元素由“键”（key）和“值”（value）组成，键必须唯一且不可变，值可以是任意类型。字典用大括号 {} 表示，键和值之间用冒号: 分隔，多个键值对之间用逗号, 分隔。

```
# 定义字典
person = {"name": "Tom", "age": 18, "gender": "male"}
print(person)

# 访问字典的值（通过键）
print(person["name"])      # 输出: Tom

# 添加或修改元素
person["city"] = "Beijing" # 添加新键值对
person["age"] = 20         # 修改已有键的值
print(person)

# 遍历字典
for key in person:
    print(key, person[key])

# 常用方法
print(person.keys())       # 所有键
print(person.values())     # 所有值
print(person.items())      # 所有键值对
```

```
print("name" in person)    # 判断键是否存在
```

```
{'name': 'Tom', 'age': 18, 'gender': 'male'}
Tom
{'name': 'Tom', 'age': 20, 'gender': 'male', 'city': 'Beijing'}
name Tom
age 20
gender male
city Beijing
dict_keys(['name', 'age', 'gender', 'city'])
dict_values(['Tom', 20, 'male', 'Beijing'])
dict_items([('name', 'Tom'), ('age', 20), ('gender', 'male'), ('city', 'Beijing')])
True
```

2.5.4 应用：利用字典进行词频统计

统计一段文本中每个单词出现的次数，例如统计下面英文歌词代词出现的次数。首先将字母转化为小写字母，然后利用字符的切分方法`.split()`，将结果保存在字典 `freq` 之中，然后按词频排序，注意使用了 lambda 函数的方法。

```
text = """Generals gathered in their masses
Just like witches at black masses
Evil minds that plot destruction
Sorcerer of death's construction
In the fields, the bodies burning
As the war machine keeps turning
Death and hatred to mankind
Poisoning their brainwashed minds
Oh, Lord, yeah
Politicians hide themselves away
They only started the war
Why should they go out to fight?
```

```
They leave that role to the poor, yeah
Time will tell on their power minds
Making war just for fun
Treating people just like pawns in chess
Wait 'til their judgement day comes, yeah
Now in darkness, world stops turning
Ashes where their bodies burning
No more war pigs have the power
Hand of God has struck the hour
Day of judgement, God is calling
On their knees, the war pigs crawling
Begging mercy for their sins
Satan laughing, spreads his wings
Oh, Lord, yeah
"""

words = text.lower().split()
freq = {}

for word in words:
    if word in freq:          # 判断单词是否已在字典中
        freq[word] += 1
    else:
        freq[word] = 1

sorted_freq = sorted(freq.items(),
                     key=lambda item: item[1],
                     reverse=True)

for word, count in sorted_freq[:10]:
    print(f"{word}: {count}")
```

```
the: 8
```

```
their: 7
war: 5
in: 4
yeah: 4
just: 3
minds: 3
of: 3
to: 3
they: 3
```

2.6 函数与模块

2.6.1 内置函数的调用

Python 内置函数是系统自带的、可以直接使用的函数，无需导入模块。常见内置函数有：abs(), len(),max(), min(), sum(), type(), int(), float(), str(), list(), dict(), range() 等

```python
# 例 1: abs()
print(abs(-10))

# 例 2: len() 计算长度
lst = [1, 2, 3, 4]
print(len(lst))

# 例 3: max() 和 min() 求最大最小值
print(max(5, 8, 2))
print(min([7, 3, 9]))

# 例 4: sum() 求和
print(sum([1, 2, 3]))

# 例 5: type() 查看类型
```

```
print(type("hello"))

# 例 6: int(), float(), str() 类型转换
print(int("123"))
print(float("3.14"))
print(str(456))
```

```
10
4
8
3
6
<class 'str'>
123
3.14
456
```

2.6.2 自定义函数

自定义函数使用 def 关键字，指定函数名、参数列表和函数体。下面是几个简单的自定义函数的例子。

```
def greet(name):
    print(f"Hello, {name}!")
greet("Alice")
```

```
Hello, Alice!
```

带返回值的函数，如计算平均数

```
def mean(numbers):
    total = sum(numbers)
    N = len(numbers)
    answer = total / N
```

```
    return answer
nums = [3, 8, 1, 6]
print(mean(nums))
```

```
4.5
```

带默认参数的函数

```
def power(base, exponent=2):
    return base ** exponent

print(power(4))
print(power(2, 3))
```

```
16
8
```

函数也可以有多个返回值（返回元组）

```
def min_max(numbers):
    return min(numbers), max(numbers)

nums = [3, 8, 1, 6]
min, max = min_max(nums)
```

另外一个例子计算商和余数:

```
def divide(a, b):
    quotient = a // b
    remainder = a % b
    return quotient, remainder

q, r = divide(17, 5)
```

可变参数： `*args` 接收任意数量的位置参数，类型为元组

```
def total(*args):
    return sum(args)

print(total(1, 2, 3))
print(total(5, 10))
```

```
6
15
```

关键字参数：**kwargs 接收任意数量的关键字参数，类型为字典

```
def show_info(**kwargs):
    for key, value in kwargs.items():
        print(f"{key}: {value}")

show_info(name="Alice", age=20)
```

```
name: Alice
age: 20
```

2.6.3 例：世界银行经济体分组（续）

将前面例子中的 `if` 语句代码，赋值粘贴至某一个 AI 大模型平台，输入提示："将下面的 Python 代码定义为一个函数"。

```
def classify_economy(gni):
    if gni <= 1135:
        economy = " 低收入经济体"
    elif gni <= 4465:
        economy = " 中等偏下收入经济体"
    elif gni <= 13845:
        economy = " 中等偏上收入经济体"
    else:
        economy = " 高收入经济体"
    return economy
```

```
e = classify_economy(13500)
print(e)
```

```
中等偏上收入经济体
```

自定义函数常与其他功能结合使用，例如 Pandas 库的 `df.apply()` 方法。下面的例子首先从 World Bank 数据库下载 2024 年GDP per capita (current US$)数据，删除缺失值，然后应用自定义的 `classify_economy()` 函数：

```
import pandas as pd
import wbgapi as wb
gdp_pc_2024 = wb.data.DataFrame("NY.GDP.PCAP.CD", time = 2024)
gdp_pc_2024 = gdp_pc_2024.dropna()
gdp_pc_2024['NY.GDP.PCAP.CD'].apply(classify_economy)
```

```
economy
AFE    中等偏下收入经济体
AFW    中等偏下收入经济体
AGO    中等偏下收入经济体
ALB    中等偏上收入经济体
AND      高收入经济体
         ...
XKX    中等偏上收入经济体
YEM      低收入经济体
ZAF    中等偏上收入经济体
ZMB    中等偏下收入经济体
ZWE    中等偏下收入经济体
Name: NY.GDP.PCAP.CD, Length: 232, dtype: object
```

2.6.4 例：BMI 指数函数

同样，也可以将上面计算 BMI 指数的过程，定义为一个函数：

```
def bmi(height, weight):
    """ 计算 BMI 指数并返回数值和健康状况"""
```

```
    bmi = weight / (height ** 2)
    if bmi < 18.5:
        status = " 偏瘦"
    elif bmi < 24:
        status = " 正常"
    elif bmi < 28:
        status = " 超重"
    else:
        status = " 肥胖"
    return bmi, status

# 示例调用
bmi, status = bmi(1.75, 75)
print(f"BMI = : {bmi:.2f}, {status}")
```

```
BMI = : 24.49, 超重
```

2.6.5 例：定义一个集中度函数

令 s_i 表示企业 i 的市场份额，定义一个函数，计算：

产业集中度指数（CR_n ）

$$CR_4 = \sum_{i=1}^{4} s_i$$

和赫芬达尔指数

$$H_i = \sum_{i=1}^{n} s_i^2$$

```
def concentration_index(sales, top_n=4):
    """
    计算前 N 家企业的集中度（Cn）和赫芬达尔-赫希曼指数（HHI）。
    sales: 销售收入列表
    top_n: 前 N 家，默认为 4（C4），可设为 8（C8）
    返回: (Cn, HHI)
```

```
    """
    total = sum(sales)
    if total == 0:
        return 0, 0
    sorted_sales = sorted(sales, reverse=True)
    cn = sum(sorted_sales[:top_n]) / total
    hhi = sum((s / total) ** 2 for s in sales)
    return cn, hhi

# 示例
sales = [100, 80,85, 60, 40, 30, 20, 10, 5, 3, 2]
concentration_index(sales, top_n=4)
```

```
(0.7471264367816092, 0.15993129871845688)
```

2.6.6 例：定义 Cobb-Douglas 函数

Cobb-Douglas 生产函数是最常见的设定，例如一个规模报酬不变的 CD 函数表示为：

$$Y = AK^{\alpha}L^{1-\alpha}$$

其中，α 表示资本的产出弹性，A 表示全要素生产率。

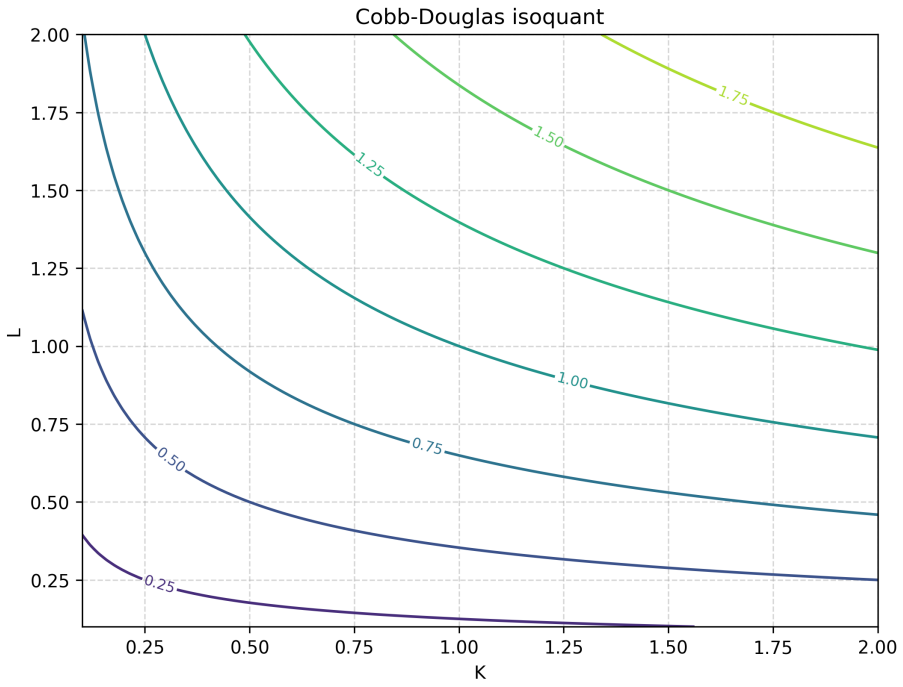
```
import numpy as np
import matplotlib.pyplot as plt

def cobb_douglas(K, L, alpha = 1/3, A = 1):
    output = A * K**alpha * L**(1 - alpha)
    return output
cobb_douglas(K=1, L=0.5)
```

```
0.6299605249474366
```

从微观经济学内容我们知道，等产量曲线是带来相同产出的要素组合 (K, L) 形成的曲线，可以应用 matplotlib 中的 `contour` 函数绘制上面函数定义的

等产量曲线：

```
# 生成 K 和 L 的网格
K = np.linspace(0.1, 2, 100)
L = np.linspace(0.1, 2, 100)
K_grid, L_grid = np.meshgrid(K, L)

# 利用函数计算产量
Y = cobb_douglas(K_grid, L_grid)
# 绘图
fig, ax = plt.subplots(figsize=(8, 6))
contours = ax.contour(K_grid, L_grid, Y, levels=8, cmap='viridis')
ax.clabel(contours, inline=True, fontsize=8)
ax.set_xlabel('K')
ax.set_ylabel('L')
ax.set_title('Cobb-Douglas isoquant')
ax.grid(True, linestyle='--', alpha=0.5)
plt.show()
```



2.6.7 模块的导入与使用

Python 的模块（module）是包含一组功能的代码文件，可以通过 import 语句导入并使用其中的函数、变量等。分为三类：- python 自带模块，不需要安装，直接 import 载入就可以了，如 math 包；- 第三方模块，通常需要下载安装，然后载入，如 numpy，scipy 等；- 自定义模块，自己写的实现某些功能的 py 文件的集合。

```
import math
print(math.sqrt(16))
print(math.pi)
#
from random import randint
print(randint(1, 10))


#
```

```
import datetime as dt
now = dt.datetime.now()
print(now)
```

```
4.0
3.141592653589793
2
2025-07-28 00:47:14.979060
```

可以把自定义的函数等保存在.py 格式等脚本文件中，自定义模块。例如，在文件夹 pyfiles 中 my_module.py 文件保存了一个计算 BMI 的函数，我们载入该函数然后进行计算：

```
from pyfiles.my_module import bmi_index

# 示例调用
bmi, status = bmi_index(1.75, 75)
print(f"BMI = {bmi:.2f}")
print(f" 健康状况: {status}")
```

```
BMI = 24.49
健康状况: 超重
```

2.7 面向对象的编程

2.7.1 什么是类

类（class）是面向对象编程（OOP）的核心概念，用于描述具有相同属性和方法的一类对象的模板或蓝图。对象（object）是类的实例，拥有类定义的属性和方法。

2.7.2 定义一个简单的类

下面定义一个类 Student，它具有两个属性 name、age，和一个方法 introduce()：

```
class Student:
    # 构造方法，初始化属性
    def __init__(self, name, age):
        self.name = name
        self.age = age

    # 方法
    def introduce(self):
        print(f"Hello, My name is {self.name}, I'm {self.age} years old.")

#
stu1 = Student("Alice", 20)
stu2 = Student("Bob", 22)

#
stu1.introduce()
stu2.introduce()
```

```
Hello, My name is Alice, I'm 20 years old.
Hello, My name is Bob, I'm 22 years old.
```

2.7.3 例：定义一个局部均衡分析模型

微观经济学中，简单的局部均衡市场模型表示为:

$$\begin{cases} Q_D = a - b \times P \\ Q_S = c + d \times P \\ Q_D = Q_S \end{cases}$$

我们定义一个类 Market，将参数定义为属性，将需求、供给和均衡条件定义为三个方法：

- demand 计算给定价格的需求量；
- supply 计算给定价格的供给量
- equilibrium 计算均衡价格和均衡数量;

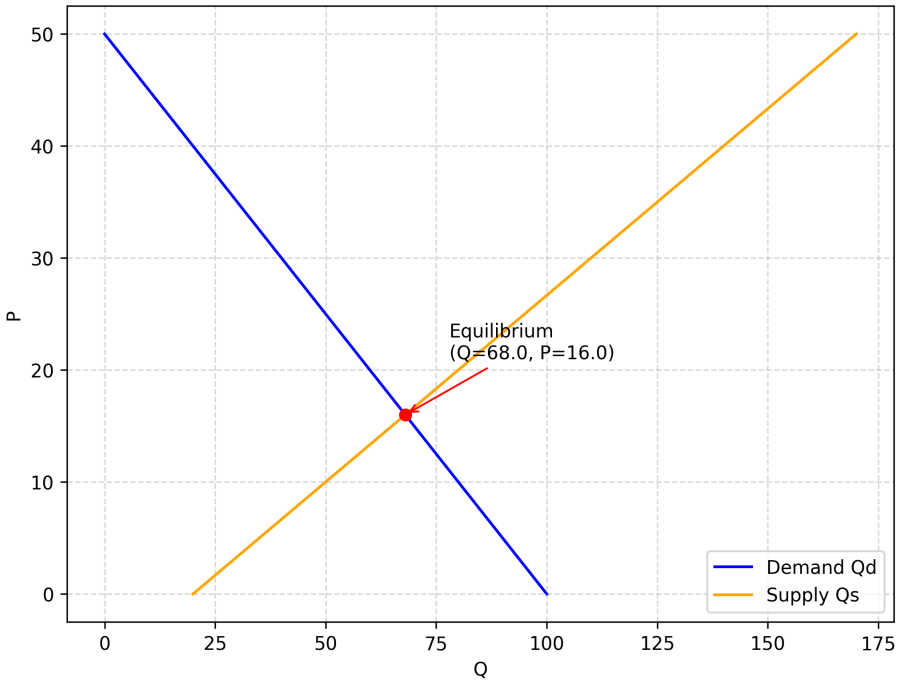
```
class Market:
    def __init__(self, a, b, c, d):
        """
        Qd = a - b*P
        Qs = c + d*P
        """
        self.a = a
        self.b = b
        self.c = c
        self.d = d

    def demand(self, P):
        return self.a - self.b * P

    def supply(self, P):
        return self.c + self.d * P

    def equilibrium(self):
        """
        Qd = Qs
        """
        # a - b*P = c + d*P
        # (a - c) = (b + d)*P
        P_eq = (self.a - self.c) / (self.b + self.d)
        Q_eq = self.demand(P_eq)
        return P_eq, Q_eq
```

```
#
market = Market(a=100, b=2, c=20, d=3)
P_star, Q_star = market.equilibrium()
print(f"Equilibrium Price = {P_star:.2f}")
print(f"Equilibrium Quantity = {Q_star:.2f}")
```

```
Equilibrium Price = 16.00
Equilibrium Quantity = 68.00
```

可以利用定义的 market 类中的方法绘制需求曲线和供给曲线：

```
import numpy as np
import matplotlib.pyplot as plt

# 生成价格区间
P = np.linspace(0, 50, 200)
Qd = [market.demand(p) for p in P]
Qs = [market.supply(p) for p in P]

# 绘图
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(Qd, P, label="Demand Qd", color="blue")
ax.plot(Qs, P, label="Supply Qs", color="orange")
ax.scatter(Q_star, P_star, color="red", zorder=5)
ax.annotate(f"Equilibrium\n(Q={Q_star:.1f}, P={P_star:.1f})",
            xy=(Q_star, P_star), xytext=(Q_star+10, P_star+5),
            arrowprops=dict(arrowstyle="->", color="red"))
ax.set_xlabel("Q")
ax.set_ylabel("P")
ax.legend()
ax.grid(True, linestyle="--", alpha=0.5)
plt.show()
```


50
40
30
20
10
0
P
0
25
50
75
100
125
150
175
Q
Equilibrium
(Q=68.0, P=16.0)
Demand Qd
Supply Qs

第三章　Numpy 和 Pandas 基础

3.1　Numpy 基础

3.1.1　创建数组

有许多种方法创建数组，下面是一些简单的例子，使用 `np.array()` 函数，将列表、元组转化为数组：

```
import numpy as np

a = np.array([1, 2, 3, 4])
print(a)
```

```
[1 2 3 4]
```

注意，与列表不同，Numpy 数组只能包含相同类型的数据，下面的例子中，`np.array()` 函数自动将列表中的整数转换为浮点数：

```
b = np.array([3.14, 4, 2, 3])
b
```

```
array([3.14, 4.  , 2.  , 3.  ])
```

列表总是一维的，Numpy 数组可以是多维的，例如下面的例子使用：

```
data = np.array([[1.5, -0.1, 3],
                 [0, -3, 6.5]])
print(data)
```

```
[[ 1.5 -0.1  3. ]
 [ 0.  -3.   6.5]]
```

数组 data 是二维数组，可以查看属性 ndim 和 shape：

```
data.ndim
data.shape
```

```
(2, 3)
```

可以对 data 进行通常的数学运算：

```
print(data * 10)
print(data + data)
```

```
[[ 15.  -1.  30.]
 [  0. -30.  65.]]
[[ 3.  -0.2  6. ]
 [ 0.  -6.  13. ]]
```

Numpy 也有函数来生成一些特定格式的数组, 如表 Table 3.1 所示：

Table 3.1: Numpy 中生成数组的函数

函数名	描述
array	将输入数据（列表、元组、数组或其他序列类型）转换为 ndarray，可以自动推断或显式指定数据类型；默认会复制输入数据
asarray	将输入转换为 ndarray，如果输入已经是 ndarray，则不会进行复制
arange	类似于内置的 range，但返回的是 ndarray 而不是列表
ones, ones_like	生成给定形状和数据类型的全 1 数组；ones_like 以另一个数组为模板，生成相同形状和数据类型的全 1 数组
zeros, zeros_like	类似于 ones 和 ones_like，但生成的是全 0 数组
empty, empty_like	通过分配新内存创建新数组，但不会像 ones 和 zeros 那样填充值

函数名	描述
full, full_like	生成给定形状和数据类型的数组，所有值都设置为指定的“填充值”；full_like 以另一个数组为模板，生成相同形状和数据类型的填充值数组
eye,identity	生成单位矩阵（对角线为 1，其余为 0）

```
zeros = np.zeros(10)
print(zeros)
ones = np.ones((2,3), dtype=float)
print(ones)
# 单位矩阵
idents = np.identity(3)
print(idents)

evens = np.arange(0, 20, 2)
print(evens)
grids = np.linspace(0, 1, 21)
print(grids)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[[1. 1. 1.]
 [1. 1. 1.]]
[[1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]]
[ 0  2  4  6  8 10 12 14 16 18]
[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0.65
 0.7  0.75 0.8  0.85 0.9  0.95 1.  ]
```

Numpy 中 random 子库包含丰富的生成随机数的函数，例如：

```
# 生成正态分布
nums_norm = np.random.normal(loc=0, scale=1, size=(4, 3))
print(nums_norm)
```

```
nums_int = np.random.randint(low=1, high=11, size=(2, 10))
print(nums_int)
```

```
[[-1.99099513  0.77175133 -0.4985406 ]
 [ 0.15565743 -1.19637674 -1.28048606]
 [-0.74699724 -0.48749577  1.52576719]
 [-0.87959326 -1.29629248 -1.42161959]]
[[ 2  5 10  3  5  1  8  7  6  7]
 [ 5  3  8 10  1  6  5  1  4  7]]
```

3.1.2 数组的索引

注意索引与列表一样，从 0 开始；选择元素时不包括右侧。

```
z = np.array((1,2,3,4,5))
z[0]
z[0:2]
z[-1]
z[::2]
z[::-1]
# 2D arrays
z = np.array([[1,2],
              [3, 4]])
z[0,0]
z[0,:]
z[:,1]
```

```
array([2, 4])
```

3.1.3 数组方法

数组方法众多，例如：

代码段

```
a = np.array((4,3,2,1))
a.sort()

a.sum()
a.mean()
a.max()
a.min()
a.var()
a.std()
a.argmax()
a.cumsum()
a.cumprod()
```

```
array([ 1,  2,  6, 24])
```

3.1.4 数组的数学运算

注意，运算符 +, - , *, / 和 **，都是逐元素运算。例如：

```
a = np.array([1,2,3,4])
b = np.array([5,6,7,8])
a + b
a * b
a + 10
a * 10
# 2D array
A = np.ones((2,2))
B = np.ones((2,2))
A + B
A+10
A * B
(A+1) ** 2
```

```
array([[4., 4.],
```

```
       [4., 4.]])
```

可以使用 @ 或 np.dot() 进行矩阵乘法。如果是向量则计算内积。

```
A = np.array([[1,2],
              [3,4]])
B = np.array([[5,6,],
              [7,8]])
A@B
#or
np.dot(A,B)
#
b = np.array([0, 1])
A@b
```

```
array([2, 4])
```

3.1.5 例：多项式计算

Numpy 中有一些列简便运算的函数。例如 `np.poly1d()`，多项式求和:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_N x^N = \sum_{n=0}^{N} a_n x^n$$

```
p = np.poly1d([1,2,3])
print(p)
print(p(2))
```

```
   2
1 x + 2 x + 3
11
```

注意，`np.poly1d()` 函数高阶项在前面。

利用向量计算，自定义一个函数：

```
def poly1d(x, coef):
    X = np.ones_like(coef)
    X[1:] = x
    y = np.cumprod(X) # y = [1,x,x**2,...]
    return coef @ y[::-1]

coef = [1, 2, 3]
poly1d(2, coef=coef)
```

```
np.int64(11)
```

3.1.6 Random 子库

Numpy 中有大量的与随机数生成器有关的函数。

下面是一个例子，注意，没有设定随机种子数，因此每次运行结果会不同。

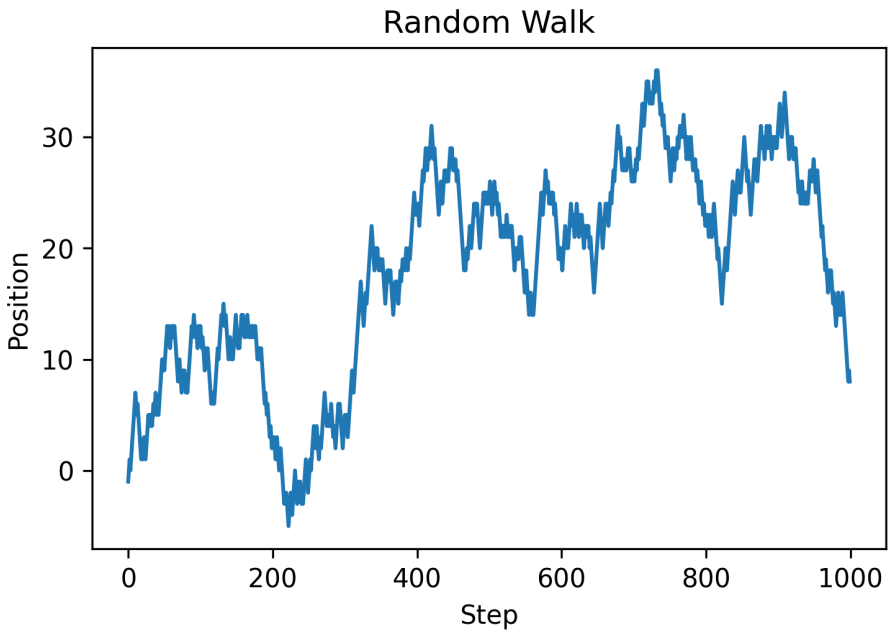
```
import numpy as np

# Define an array of choices
choices = np.array(['apple', 'banana', 'orange', 'grape', 'kiwi'])

# Perform random choice
random_choice = np.random.choice(choices)

# Print the random choice
print(random_choice)
```

```
kiwi
```

3.1.6.1 例：简单的随机游走

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# 设置随机种子以便复现
np.random.seed(0)

# 步数
n_steps = 1000

# 生成每一步的随机步长（-1 或 1）
steps = np.random.choice([-1, 1], size=n_steps)

# 计算随机游走序列
walk = np.cumsum(steps)

# 绘制线形图
plt.plot(walk)
plt.title('Random Walk')
plt.xlabel('Step')
plt.ylabel('Position')
plt.show()
```



3.1.6.2 例：利用随机数模拟中心极限定理

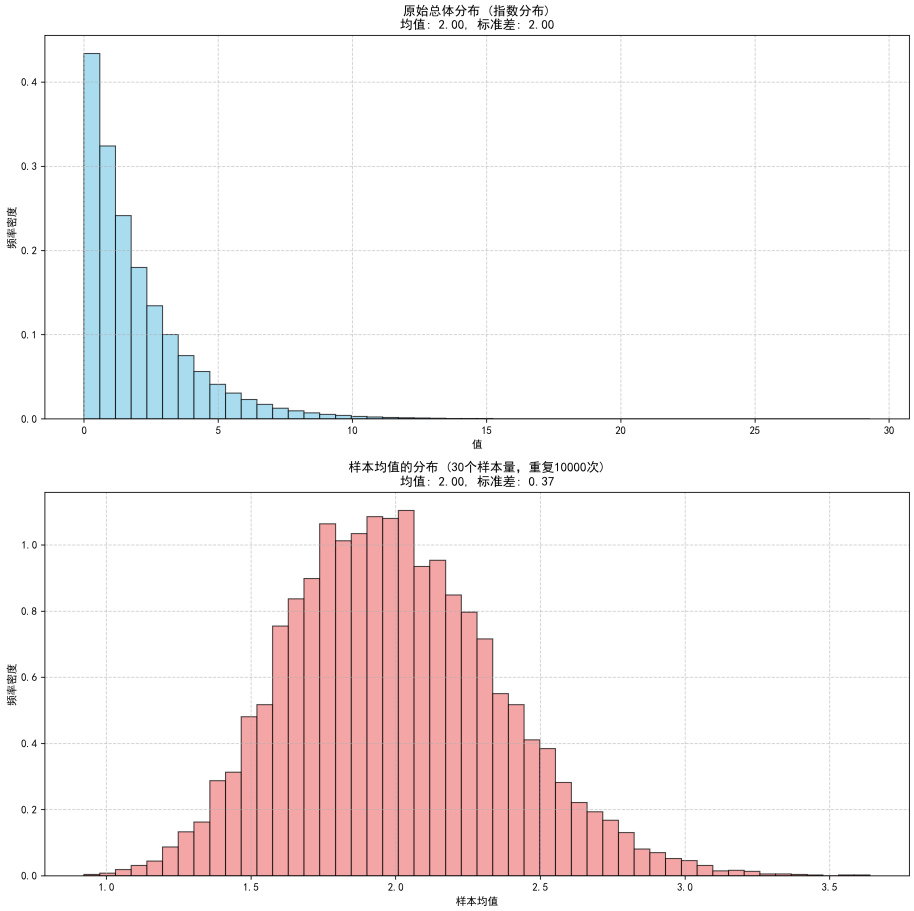
中心极限定理 (Central Limit Theorem, CLT) 是概率论中一个非常强大的定理。它指出，当从任何形状的总体中抽取足够大的独立同分布 (i.i.d.) 样本时，这些样本均值的分布将近似于正态分布，无论原始总体分布如何。样本量越大，近似程度越好。

我们将通过以下步骤来模拟验证 CLT：

- 选择一个非正态分布的总体: 比如，一个指数分布或均匀分布，它们的形状都不是钟形的。
- 设置样本参数: 定义每次抽样的样本大小 (sample_size) 和重复抽样的次数 (num_samples)。
- 重复抽样并计算均值: 从总体中抽取 num_samples 次样本，每次抽取 sample_size 个数据点，并计算每次抽样的平均值。
- 可视化: 绘制样本均值的直方图，并与原始总体分布的直方图进行对比。

```
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['font.family'] = 'SimHei'
plt.rcParams['axes.unicode_minus'] = False

# --- 1. 设置模拟参数 ---
population_size = 1000000  # 原始总体的大小
sample_size = 30           # 每次抽样的样本量（通常大于 30 就被认为是“大样本”）
num_samples = 10000        # 重复抽样的次数，即我们将有多少个样本均值
np.random.seed(123)

# --- 2. 选择一个非正态分布的总体（例如：指数分布）---
# 指数分布 (Exponential Distribution) 是一种偏态分布，非常适合验证 CLT
# numpy.random.exponential(scale=1.0, size=None)
# scale 参数是均值，这里我们设置均值为 2.0
population_data = np.random.exponential(scale=2.0, size=population_size)
# 也可以用均匀分布作为总体进行验证
# population_data_uniform = np.random.uniform(low=0.0, high=10.0, size=population_

# --- 3. 重复抽样并计算均值 ---
sample_means = []
for _ in np.arange(num_samples):
    # 从总体中随机抽取 sample_size 个数据点
    sample = np.random.choice(population_data, size=sample_size, replace=True)
    # 计算样本的均值并添加到列表中
    sample_means.append(np.mean(sample))

# 将样本均值列表转换为 NumPy 数组，方便后续处理和绘图
sample_means = np.array(sample_means)

# --- 4. 可视化结果 ---
plt.figure(figsize=(12, 12))
```

```
# 绘制原始总体分布的直方图
plt.subplot(2, 1, 1) # 1 行 2 列的第一个图
plt.hist(population_data, bins=50, density=True, color='skyblue', edgecolor='black', alpha=0
plt.title(f'原始总体分布 (指数分布)\n均值: {np.mean(population_data):.2f}, 标准差: {np.std(pop
plt.xlabel('值')
plt.ylabel('频率密度')
plt.grid(True, linestyle='--', alpha=0.6)

# 绘制样本均值分布的直方图
plt.subplot(2, 1, 2) # 1 行 2 列的第二个图
plt.hist(sample_means, bins=50, density=True, color='lightcoral', edgecolor='black', alpha=0
plt.title(f'样本均值的分布 ({sample_size}个样本量, 重复{num_samples}次)\n均值: {np.mean(sample
plt.xlabel('样本均值')
plt.ylabel('频率密度')
plt.grid(True, linestyle='--', alpha=0.6)

plt.tight_layout() # 调整子图布局, 避免重叠
plt.show()

# --- 5. 额外验证: 比较均值和标准差 ---
print("\n--- 模拟结果验证 ---")
print(f" 原始总体的均值 ( ): {np.mean(population_data):.4f}")
print(f" 原始总体的标准差 ( ): {np.std(population_data):.4f}")
print(f" 样本均值的均值 ( _x̄): {np.mean(sample_means):.4f}")
# 根据中心极限定理, 样本均值的标准差 (标准误差) 应该约等于 总体标准差 / sqrt(样本量)
expected_std_of_means = np.std(population_data) / np.sqrt(sample_size)
print(f" 样本均值的标准差 ( _x̄): {np.std(sample_means):.4f}")
print(f" 理论上样本均值的标准差 (  / sqrt(n)): {expected_std_of_means:.4f}")
```




```
--- 模拟结果验证 ---
原始总体的均值 ( ): 1.9988
原始总体的标准差 ( ): 1.9992
样本均值的均值 ( _x̄): 1.9984
样本均值的标准差 ( _x̄): 0.3653
理论上样本均值的标准差 (  / sqrt(n)): 0.3650
```

3.1.7 通用函数

Numpy 中许多函数是通用函数 (universal functions)，是一种在 ndarray 数据中进行逐元素操作的函数，大多数数学函数属于此类。

例如 `np.cos()` 函数：

```
np.cos(1.0)
np.cos(np.linspace(0, 1, 3))
```

```
array([1.        , 0.87758256, 0.54030231])
```

例如，我们想计算 $\frac{0}{1}, \frac{1}{2}, \cdots, \frac{4}{5}$:

```
np.arange(5) / np.arange(1, 6)
```

```
array([0.        , 0.5       , 0.66666667, 0.75      , 0.8       ])
```

Table 3.2: Numpy 中算术运算子和函数

运算符	对应的 ufunc	描述	示例
+	np.add	加法	1 + 1 = 2
-	np.subtract	减法	3 - 2 = 1
-	np.negative	一元取反	-2
*	np.multiply	乘法	2 * 3 = 6
/	np.divide	除法	3 / 2 = 1.5
//	np.floor_divide	向下取整除法	3 // 2 = 1
**	np.power	幂运算	2 ** 3 = 8
%	np.mod	取模/余数	9 % 4 = 1

3.1.8 例：通用函数

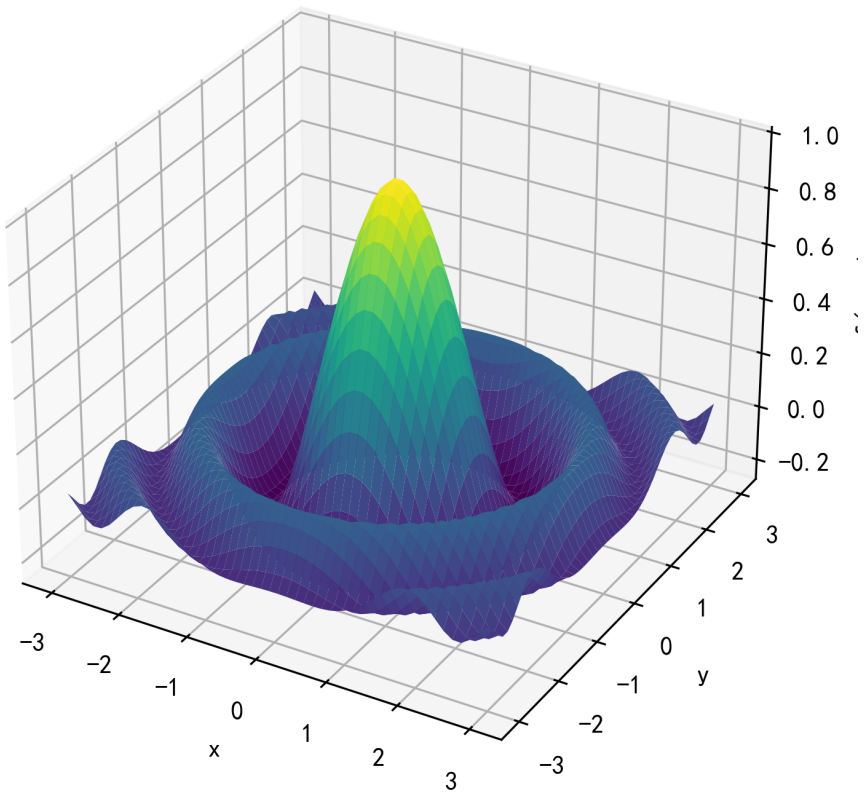
考察最大化函数 $f(x, y)$ 在区间 $ [−a,a] ×[−a, a]$ 上的最大值：

$$f(x, y) = \frac{cos(x^2 + y^2)}{1 + x^2 + y^2}$$

令 $a = 3$。我们定义一个函数，然后生成数组，计算对应的-值，通过栅格（grid）搜索最大值（等于 1）。

```
import numpy as np
from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt

def f(x, y):
    return np.cos(x**2 + y**2) / (1 + x**2 + y**2)

grid = np.linspace(-3, 3, 50)
x, y = np.meshgrid(grid, grid)
z = f(x, y)

# 最大值
max_value = np.max(z)
print(" 函数的最大值:", max_value)

# 绘制 3D 图像
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='viridis')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
plt.show()
```

```
函数的最大值: 0.9925310162998334
```



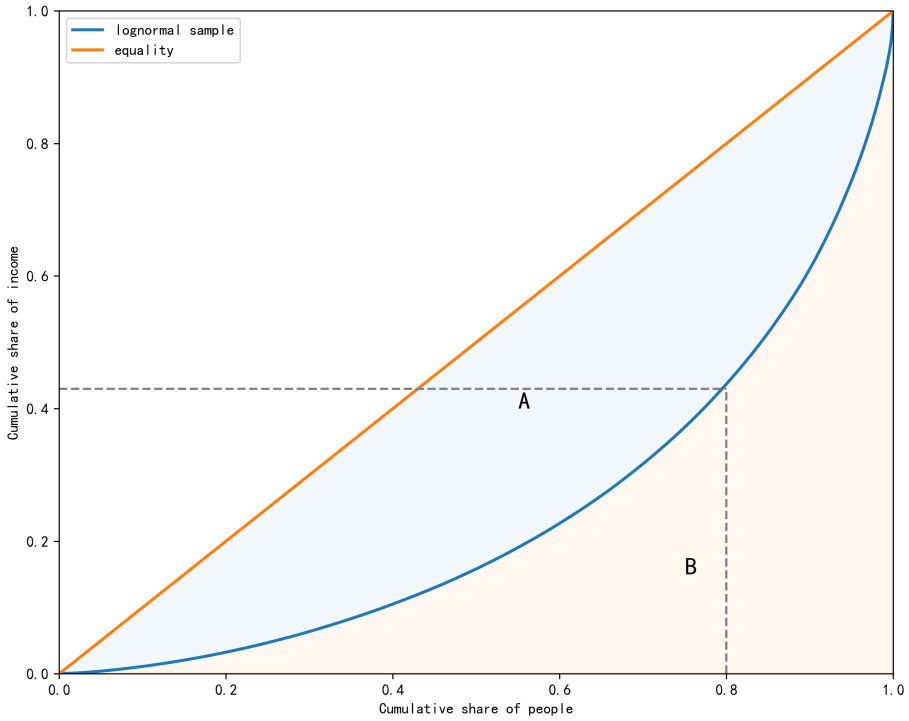
图示

3.1.9 例：洛伦茨曲线和基尼系数

```
import numpy as np # 载入 numpy 库
def lorenz_curve(y):
    n = len(y)
    y = np.sort(y) # 从小到大排序
    s = np.zeros(n + 1) # 生成 n+1 个数值零
    s[1:] = np.cumsum(y) # 从第 2 个数（索引 1）累计求和，使第一个数据点为（0，0）
    cum_people = np.linspace(0, 1, n + 1)
```

```
    cum_income = s / s[n] # s[n] 为最后的值，即所有值的和
    return cum_people, cum_income
```

```
n = 2000
np.random.seed(1)
sample = np.exp(np.random.randn(n))
f_vals, l_vals = lorenz_curve(sample)
#
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(f_vals, l_vals, label=f'lognormal sample', lw = 2)
ax.plot([0, 1], [0, 1], label='equality', lw = 2)
ax.fill_between(f_vals,l_vals, f_vals, alpha=0.06)
ax.fill_between(f_vals, l_vals, np.zeros_like(f_vals),alpha=0.06)
ax.vlines([0.8], [0], [0.43], linestyles='--', colors='gray')
ax.hlines([0.43], [0], [0.8], linestyles='--', colors='gray')
ax.set_xlim((0,1))
ax.set_ylim((0,1))
ax.text(0.55, 0.4,"A", fontsize=16)
ax.text(0.75,0.15,"B",fontsize=16)
ax.set_xlabel('Cumulative share of people')
ax.set_ylabel('Cumulative share of income')
ax.legend()
plt.show()
```



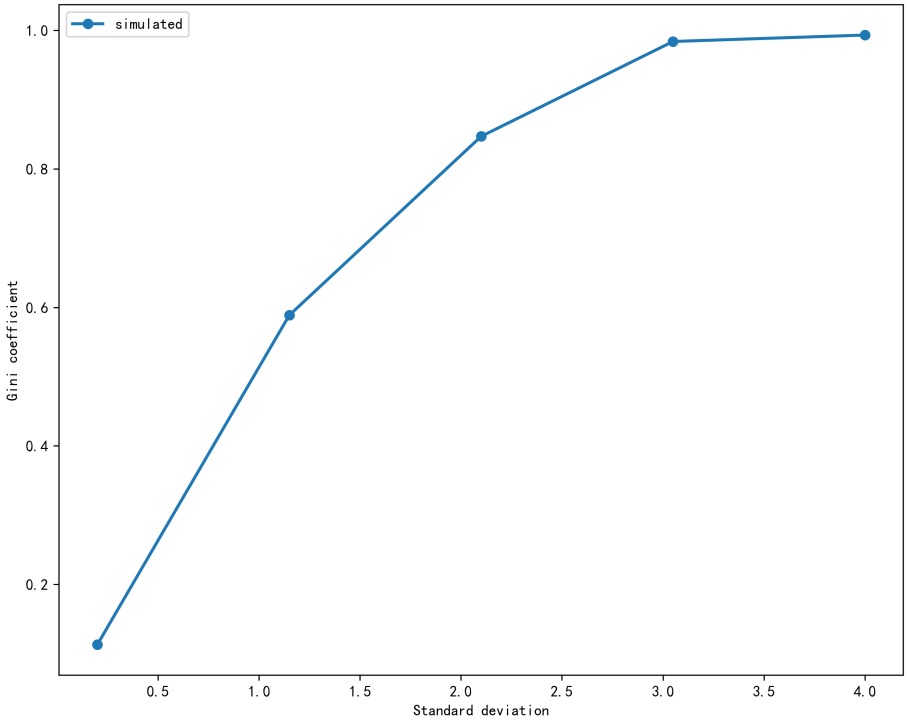
3.1.10 基尼系数

```
def gini(y):
    n = len(y)
    y_1 = np.reshape(y, (n, 1))
    y_2 = np.reshape(y, (1, n))
    g_sum = np.sum(np.abs(y_1 - y_2)) # 利用了 numpy 的广播（broadcasting）
    return g_sum / (2 * n * np.sum(y))
```

```
# 模拟对数正态数据
np.random.seed(1)
k = 5
sigmas = np.linspace(0.2, 4, k)
n = 2000
ginis = []
```

```
for sigma in sigmas:
    mu = -sigma ** 2 / 2
    y = np.exp(mu + sigma * np.random.randn(n))
    ginis.append(gini(y))
```

```
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(sigmas, ginis, marker = 'o',label='simulated', lw = 2)
ax.set_xlabel('Standard deviation')
ax.set_ylabel('Gini coefficient')
ax.legend()
plt.show()
```



3.2 Pandas 基础

Pandas 是数据分析最常用的包:

- Pandas 定义了处理数据的结构；
- 数据处理：读取、调整指数、日期和时间序列、排序、分组、处理缺失值；
- 一些更复杂的统计功能，如 statsmodels 和 scikit-learn，也是建立在 pandas 基础上。

3.2.1 Pandas 中的序列和数据框

Pandas 中两类数据，Series 和 DataFrame；

Series 基于 Numpy 数组，支持许多类似运算；

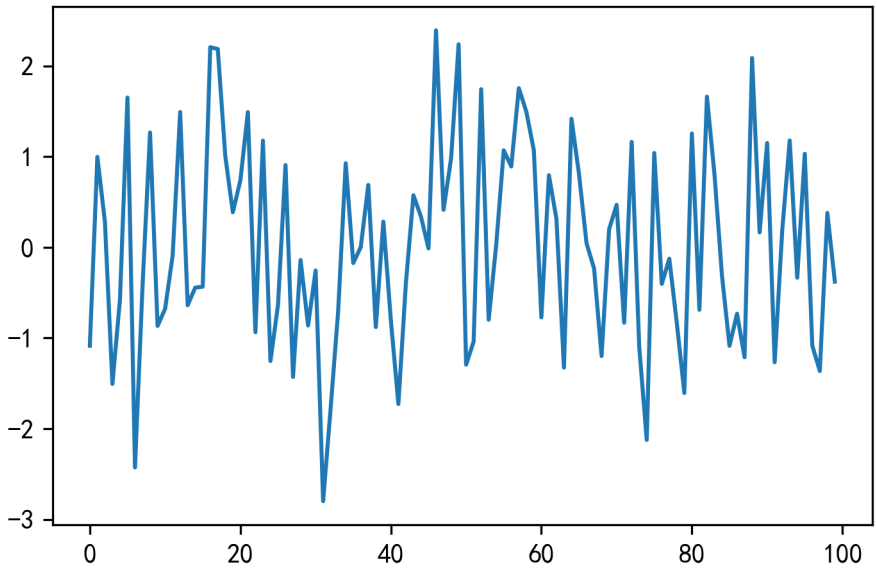
Series 可以看作一 “列” 数据；

DataFrame 可以看作储存相应列数据的二维对象；类似 Excel 表单；

Series 一些方法

```
import pandas as pd
np.random.seed(123)
s = pd.Series(np.random.randn(100), name="daily return")
s.plot();
np.abs(s)
s.describe()
```

```
count    100.000000
mean       0.027109
std        1.133924
min       -2.798589
25%       -0.832745
50%       -0.053270
75%        0.983388
max        2.392365
Name: daily return, dtype: float64
```



DataFrames 是几列数据组成，每一列对应一个变量；

用来方便的处理行和列组织的数据；索引（index）对应行，变量列名（columns）对应列；

可以读取各类软件格式存储数据，csv, excel, stata, html, json,sql 等；

3.3 应用：Penn World Table

这一部分应用Penn World Table介绍对原始数据的一些常见处理方法。该数据集当前版本为 PWT 10.01，包含 183 个国家 1950-2019 年的收入、产出、投入和生产率等指标，详细介绍可参见User Guide to PWT 10.0 data files。数据背后的方法、理论及使用建议，可参见 Feenstra, Inklaar, and Timmer [1]。

网站提供了 Stata 和 Excel 格式数据，这里我们下载了后者。数据本身是一个面板数据（Panel Data），“国家 - 年” 唯一识别一个观测值。我们从截面数据入手先只保留 2019 年数据，然后再看更复杂的情况。

3.3.1 导入数据

假设数据保存在当前路径的 datasets 子文件中：

```
import pandas as pd
pwt = pd.read_excel(io = "datasets/pwt1001.xlsx",
                    header=0,
                    sheet_name="Data")
# 保留 2019 年数据
pwt2019 = pwt[pwt['year'] == 2019].copy().drop(labels='cor_exp',axis=1)
```

注意其中的几个参数，io 是文件路径；header 表明列标题行，这里是第一行；sheet_name 是数据所在表单名；将载入的数据赋值给 pwt 数据框。我们只保留 2019 年的观测值，变量 cor_exp 在这一年全部为缺失值，这里直接删除了。

先为 pwt2019 数据框设置索引变量，这里使用国家名代码变量（countrycode）：

```
pwt2019.set_index('countrycode', inplace=True)
```

可以 df.info() 概率数据集，或者使用 df.head() 或 df.tail() 查看头部和尾部观测值：

```
pwt2019.info()
pwt2019.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 183 entries, ABW to ZWE
Data columns (total 50 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   country        183 non-null    object
 1   currency_unit  183 non-null    object
 2   year           183 non-null    int64
 3   rgdpe          183 non-null    float64
 4   rgdpo          183 non-null    float64
```

```
5   pop           183 non-null    float64
6   emp           177 non-null    float64
7   avh           66 non-null     float64
8   hc            145 non-null    float64
9   ccon          183 non-null    float64
10  cda           183 non-null    float64
11  cgdpe         183 non-null    float64
12  cgdpo         183 non-null    float64
13  cn            180 non-null    float64
14  ck            137 non-null    float64
15  ctfp          118 non-null    float64
16  cwtfp         118 non-null    float64
17  rgdpna        183 non-null    float64
18  rconna        183 non-null    float64
19  rdana         183 non-null    float64
20  rnna          180 non-null    float64
21  rkna          137 non-null    float64
22  rtfpna        118 non-null    float64
23  rwtfpna       118 non-null    float64
24  labsh         138 non-null    float64
25  irr           137 non-null    float64
26  delta         180 non-null    float64
27  xr            183 non-null    float64
28  pl_con        183 non-null    float64
29  pl_da         183 non-null    float64
30  pl_gdpo       183 non-null    float64
31  i_cig         183 non-null    object
32  i_xm          183 non-null    object
33  i_xr          183 non-null    object
34  i_outlier     183 non-null    object
35  i_irr         137 non-null    object
36  statcap       127 non-null    float64
37  csh_c         183 non-null    float64
```

```
 38  csh_i           183 non-null    float64
 39  csh_g           183 non-null    float64
 40  csh_x           183 non-null    float64
 41  csh_m           183 non-null    float64
 42  csh_r           183 non-null    float64
 43  pl_c            183 non-null    float64
 44  pl_i            183 non-null    float64
 45  pl_g            183 non-null    float64
 46  pl_x            183 non-null    float64
 47  pl_m            183 non-null    float64
 48  pl_n            180 non-null    float64
 49  pl_k            137 non-null    float64
dtypes: float64(42), int64(1), object(7)
memory usage: 72.9+ KB
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.299561	0.10
AGO	Angola	Kwanza	2019	228151.015625	227855.718750	31.8
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.680527	0.01
ALB	Albania	Lek	2019	35890.019531	36103.042969	2.88
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.250000	9.77

默认显示 5 条观测值，如果希望看到更多观测值，可以使用 `df.tail(n=10)` 修改数值。

可以应用`.shape`, `.ndim`,`.columns` 等属性查看基本信息，可以看到数据集包含 51 个变量共 183 个观测值。

```
print(pwt2019.shape)
print(pwt2019.columns)
```

```
(183, 50)
Index(['country', 'currency_unit', 'year', 'rgdpe', 'rgdpo', 'pop', 'emp',
```

```
       'avh', 'hc', 'ccon', 'cda', 'cgdpe', 'cgdpo', 'cn', 'ck', 'ctfp',
       'cwtfp', 'rgdpna', 'rconna', 'rdana', 'rnna', 'rkna', 'rtfpna',
       'rwtfpna', 'labsh', 'irr', 'delta', 'xr', 'pl_con', 'pl_da', 'pl_gdpo',
       'i_cig', 'i_xm', 'i_xr', 'i_outlier', 'i_irr', 'statcap', 'csh_c',
       'csh_i', 'csh_g', 'csh_x', 'csh_m', 'csh_r', 'pl_c', 'pl_i', 'pl_g',
       'pl_x', 'pl_m', 'pl_n', 'pl_k'],
      dtype='object')
```

3.3.2 选择观测值和变量

应用中经常对某些观测值或特定子集进行操作，因此很重要的一步是选择观测值和变量。

最基本的方法可以通过 Python 数组的切片（slicing）方式选择特定的**行**。例如，选择第 3 至 5 个观测值：

```
pwt2019[2:5]
```

countrycode	country	currency_unit	year	rgdpe	rgdpo
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.68052
ALB	Albania	Lek	2019	35890.019531	36103.042
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.25

要选择**列**，可以用包含列名字的列表：

```
vars_selected = ['country', 'rgdpe', 'rgdpo', 'pop', 'emp', 'cgdpe', 'cgdpo', 'ctf
df = pwt2019[vars_selected]
```

3.3.2.1 `.loc` 方法

`.loc` 是基于标签（label-based）的数据选择方法。这意味着你使用行和列的实际标签名来选择数据，而不是它们的整数位置。

例如，要选择金砖国家（BRICKS）的观测值：

```
bricks = ['CHN', 'BRA', 'RUS', 'IND', 'ZAF']
pwt2019.loc[bricks]
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp
CHN	China	Yuan Renminbi	2019	20056066.0	2.025766e+07	1433.783686	798
BRA	Brazil	Brazilian Real	2019	3089273.5	3.080048e+06	211.049527	93.9
RUS	Russian Federation	Russian Ruble	2019	4197222.5	4.161194e+06	145.872256	71.6
IND	India	Indian Rupee	2019	8945547.0	9.170555e+06	1366.417754	497
ZAF	South Africa	Rand	2019	748940.0	7.340944e+05	58.558270	18.6

或者选择列：

```
variables = ['country', 'rgdpe', 'pop']
pwt2019.loc[:, variables]
```

countrycode	country	rgdpe	pop
ABW	Aruba	3921.261230	0.106314
AGO	Angola	228151.015625	31.825295
AIA	Anguilla	376.634979	0.014869
ALB	Albania	35890.019531	2.880917
ARE	United Arab Emirates	681525.812500	9.770529
...	...	...	...
VNM	Viet Nam	750726.750000	96.462106
YEM	Yemen	50052.933594	29.161922
ZAF	South Africa	748940.000000	58.558270
ZMB	Zambia	57956.183594	17.861030
ZWE	Zimbabwe	42296.062500	14.645468

或者同时指定行和列：

```
pwt2019.loc[bricks, variables]
```

countrycode	country	rgdpe	pop
CHN	China	20056066.0	1433.783686
BRA	Brazil	3089273.5	211.049527
RUS	Russian Federation	4197222.5	145.872256
IND	India	8945547.0	1366.417754
ZAF	South Africa	748940.0	58.558270

3.3.2.2 `.iloc` 方法

相应的，`.iloc` 是基于整数位置（integer-location based）的，使用行和列的整数位置（从 0 开始）来选择数据。例如：

```
# 选择第 2 行数据（索引位置为 1）
pwt2019.iloc[1]
# 选择第 1 行（索引为 0）、第 3 行（索引为 2）和第 5 行（索引为 4）
pwt2019.iloc[[0, 2, 4]]
# 选择前 5 行、第 4 至第 6 列观测值
pwt2019.iloc[:5, 3:6]
```

countrycode	rgdpe	rgdpo	pop
ABW	3921.261230	3467.299561	0.106314
AGO	228151.015625	227855.718750	31.825295
AIA	376.634979	225.680527	0.014869
ALB	35890.019531	36103.042969	2.880917
ARE	681525.812500	645956.250000	9.770529

这里需要注意 Python 中索引位置。Python 中进行切片（slicing）操作时，语法通常类似 `[start:end]`，要注意：

- `start`：切片的起始索引，对应的元素会被包含。
- `end`：切片的结束索引，对应的元素不会被包含。

3.3.2.3 根据条件筛选

除了根据索引或位置选择数据外，也可以利用条件来筛选观测值。例如，根据人口变量（`pop`，单位：百万）选择 2019 年总人口超过 2 亿的观测值：

```
pwt2019[pwt2019['pop'] >= 200]
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp
BRA	Brazil	Brazilian Real	2019	3.089274e+06	3.080048e+06	211.049527	93.956
CHN	China	Yuan Renminbi	2019	2.005607e+07	2.025766e+07	1433.783686	798.80
IDN	Indonesia	Rupiah	2019	3.104439e+06	3.137931e+06	270.625568	131.17
IND	India	Indian Rupee	2019	8.945547e+06	9.170555e+06	1366.417754	497.61
NGA	Nigeria	Naira	2019	9.834982e+05	1.001537e+06	200.963599	73.020
PAK	Pakistan	Pakistan Rupee	2019	1.036800e+06	1.088502e+06	216.565318	63.085
USA	United States	US Dollar	2019	2.086051e+07	2.059584e+07	329.064917	158.29

注意，`pwt2019['pop'] >= 200` 的结果是一列布林值，然后 `pwt2019[]` 选择返回取值为 `True` 的观测值。

再例如，下面的代码包含了两个条件：

- 国家名属于金砖国家。注意这里使用了 Pandas 中的 `df.isin()` 函数；
- 2019 年人口超过 10 亿。

当有不止一个条件时，我们用 `&`, `|` 表示 `and` 和 `or` 运算符；

```
BRICKS = ['China','Brazil',  ' Russian Federation', 'India', 'South Africa']
#
pwt2019[(pwt2019['country'].isin(BRICKS)) & (pwt2019['pop'] > 1000)]
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp
CHN	China	Yuan Renminbi	2019	20056066.0	20257660.0	1433.783686	798.80
IND	India	Indian Rupee	2019	8945547.0	9170555.0	1366.417754	497.61

更复杂的情况，可以在条件语句中加入数学表达式。例如，下面的代码筛选了人均实际 GDP 超过 2 万美元和人口超过 5000 万的国家的观测值，这里人均实际 GDP 是购买力平价调整后支出法衡量的实际 GDP 与人口的比值：

```
pwt2019[(pwt2019['rgdpe']/pwt2019['pop'] > 20000) & (pwt2019['pop'] > 50)]
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
DEU	Germany	Euro	2019	4.308862e+06	4275312.00	83.5
FRA	France	Euro	2019	3.018885e+06	2946958.25	67.3
GBR	United Kingdom	Pound Sterling	2019	3.118991e+06	2989895.50	67.5
ITA	Italy	Euro	2019	2.508404e+06	2466327.50	60.5
JPN	Japan	Yen	2019	5.028348e+06	5036891.00	126
KOR	Republic of Korea	Won	2019	2.090946e+06	2162705.25	51.2
RUS	Russian Federation	Russian Ruble	2019	4.197222e+06	4161194.50	145
TUR	Turkey	New Turkish Lira	2019	2.227538e+06	2248225.75	83.4
USA	United States	US Dollar	2019	2.086051e+07	20595844.00	329

3.3.3 apply 方法

Pandas 中一个广泛应用的方法是 `df.apply()`，它将一个函数应用到每一行/列，返回一个序列；

函数可以是内嵌的（built in）也可以是自定义的，例如，计算每一列的最大值，为了节省输出空间，使用子集 `df` 数据框：

```
df.apply(np.max, axis=0)
```

```
country       Zimbabwe
rgdpe       20860506.0
rgdpo       20595844.0
pop        1433.783686
emp         798.807739
cgdpe       20791364.0
cgdpo       20566034.0
ctfp          1.276913
dtype: object
```

或者，自定义一个函数 `range(x)` 计算极差：

```
import numpy as np
def range(x):
    return np.max(x) - np.min(x)
df.select_dtypes(np.number).apply(range)
```

```
rgdpe    2.086041e+07
rgdpo    2.059577e+07
pop      1.433779e+03
emp      7.988052e+02
cgdpe    2.079126e+07
cgdpo    2.056595e+07
ctfp     1.222178e+00
dtype: float64
```

再例如，归一化（normalization）经常使用 minmax 方法：

$$Y = \frac{X_i - \min(X_i)}{\max(X_i) - \min(X_i)}$$

我们定义一个函数 `minmax()`，然后应用 `apply()` 方法：

```
def minmax(S):
    return (S-S.min())/(S.max() - S.min())
```

```
pwt2019[['pop','rgdpe', 'emp']].apply(minmax)
```

countrycode	pop	rgdpe	emp
ABW	0.000071	0.000183	0.000056
AGO	0.022193	0.010932	0.020834
AIA	0.000007	0.000013	NaN
ALB	0.002006	0.001716	0.001344
ARE	0.006811	0.032666	0.007269
...	...	...	...
VNM	0.067275	0.035983	0.063091
YEM	0.020336	0.002395	0.006922
ZAF	0.040838	0.035898	0.023335
ZMB	0.012454	0.002774	0.006538
ZWE	0.010211	0.002023	0.008548

经常将 `lambda` 函数方法与 `df.apply()` 方法相结合。例如，数据集中有 4 个指标度量 GDP，分别是 `['rgdpe', 'rgdpo','cgdpe','cgdpo']`，假设我们希望计算一个加权平均数，权重为（0.3，0.2，0.3，0.2）：

```
variables = ['rgdpe', 'rgdpo','cgdpe','cgdpo']
df[variables].apply(lambda row:
    row['rgdpe']*0.3 + row['rgdpo']*0.2 + row['cgdpe']*0.3 + row['cgdpo']*0.2,
    axis=1)
```

```
countrycode
ABW      3736.787085
AGO    227005.793750
AIA       318.944440
ALB     35987.783203
ARE    664187.912500
           ...
VNM    739027.362500
```

```
YEM      50759.290625
ZAF     742988.068750
ZMB      57414.339062
ZWE      41768.012500
Length: 183, dtype: float64
```

注意，z 选项 `axis = 1` ，将函数应用至每一行，默认值为 0。

3.3.4 检测和处理缺失值

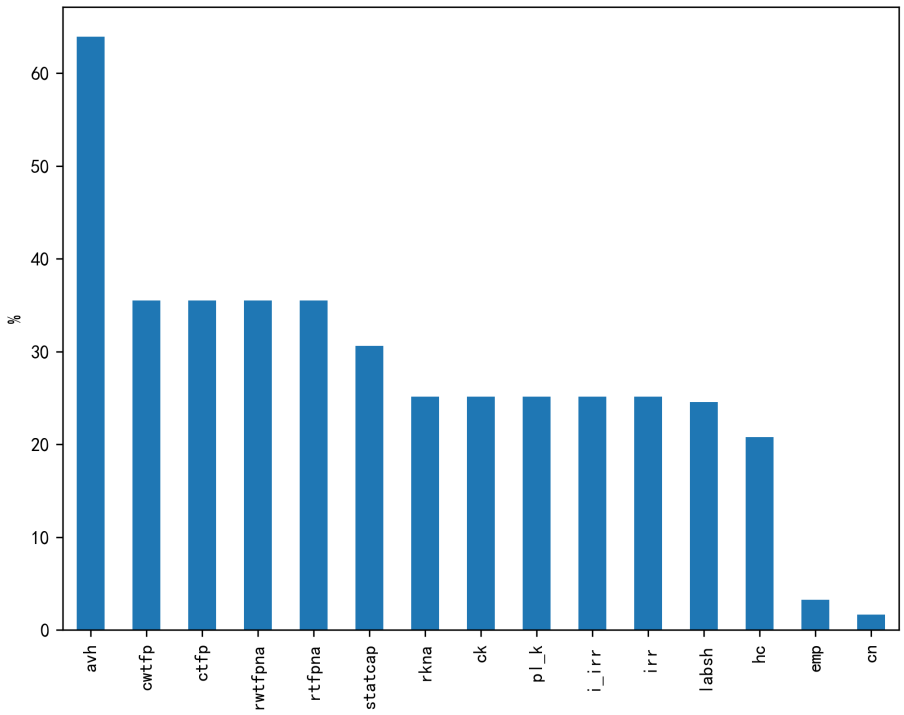
Pandas 中最常用的缺失值表示是 `NaN`（Not a Number）。可以使用 `isnull()` 或 `isna()` 函数检测缺失值，返回一个布尔型的 DataFrame，其中 `True` 表示缺失值：

```
pwt2019.isna()
#pwt2019.isnull()
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp	avh	hc	ccon	...	cs
ABW	False	False	False	False	False	False	False	True	True	False	...	F
AGO	False	False	False	False	False	False	False	True	False	False	...	F
AIA	False	False	False	False	False	False	True	True	True	False	...	F
ALB	False	False	False	False	False	False	False	True	False	False	...	F
ARE	False	False	False	False	False	False	False	True	False	False	...	F
...	...	...	...	...	...	...	...	...	...	...	...	..
VNM	False	False	False	False	False	False	False	False	False	False	...	F
YEM	False	False	False	False	False	False	False	True	False	False	...	F
ZAF	False	False	False	False	False	False	False	False	False	False	...	F
ZMB	False	False	False	False	False	False	False	True	False	False	...	F
ZWE	False	False	False	False	False	False	False	True	False	False	...	F

下面的的代码计算了缺失值的数量，将其除以样本容量得到缺失值比例，然后按照降序排序，并将比例最高的前 15 个变量绘制柱形图：

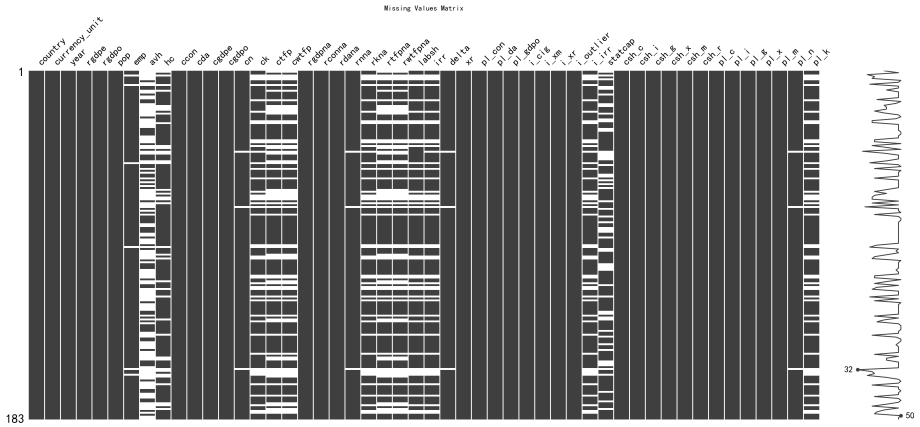
```
fig, ax = plt.subplots(figsize=(8, 6))
(pwt2019.isna().sum()/pwt2019.shape[0]*100).sort_values(ascending=False)[:15].plot
ax.set_ylabel("%")
plt.show()
```



另一种图示的方法是类似矩阵绘图的方式，将缺失值标记出来，missingno 库有简单的命令实现：

```
import missingno as msno
plt.figure(figsize=(12, 6))
msno.matrix(pwt2019)
plt.title("Missing Values Matrix")
plt.show()
```

```
<Figure size 3600x1800 with 0 Axes>
```



删除缺失值

处理缺失值的方法有很多种，选择哪种方法取决于你的数据特性、缺失原因以及分析目标。最直接的方法是使用 `df.dropna()` 函数删除包含缺失值的行或列：

```
# 删除含缺失值的行
pwt2019.dropna()
# 删除含缺失值的列
pwt2019.dropna(axis=1)
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.299561	0.10
AGO	Angola	Kwanza	2019	228151.015625	227855.718750	31.8
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.680527	0.01
ALB	Albania	Lek	2019	35890.019531	36103.042969	2.88
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.250000	9.77
...	...	...	...	...	...	...
VNM	Viet Nam	Dong	2019	750726.750000	724123.375000	96.4
YEM	Yemen	Yemeni Rial	2019	50052.933594	51828.058594	29.1
ZAF	South Africa	Rand	2019	748940.000000	734094.375000	58.5
ZMB	Zambia	Kwacha	2019	57956.183594	56783.714844	17.8
ZWE	Zimbabwe	US Dollar	2019	42296.062500	40826.570312	14.6

country	currency_unit	year	rgdpe	rgdpo
countrycode				

另外，上面的命令并没有改变原数据框，可以通过赋值方式保存。或者加上选项 `df.dropna(inplace=True)`，即在原数据框中生效。

填充

`df.fillna()` 是用于填充缺失值的核心函数。

```
#
pwt2019.fillna(0)
#
pwt2019.select_dtypes(np.number).fillna(0).combine_first(pwt2019)
pwt2019.select_dtypes(np.number).fillna(pwt2019.mean(numeric_only=True)).combine_f
pwt2019.select_dtypes(np.number).fillna(pwt2019.median(numeric_only=True)).combine
```

	avh	ccon	cda	cgdpe	cgdpo	c
countrycode						
ABW	1818.281597	3023.694824	3877.659668	3912.334717	3466.241943	0
AGO	1818.281597	155943.718750	198750.421875	227771.609375	223289.312500	0
AIA	1818.281597	438.470032	509.044983	375.136444	241.384537	0
ALB	1818.281597	33399.167969	40868.316406	35808.343750	36288.328125	0
ARE	1818.281597	306771.156250	515623.312500	678241.187500	635332.812500	0
...	...	...	...	...	...	..
VNM	2131.968232	582677.062500	758821.937500	747853.750000	723142.687500	0
YEM	1818.281597	49266.472656	67992.531250	49937.042969	51983.429688	0
ZAF	2191.363362	623669.562500	741675.937500	748245.937500	735067.062500	0
ZMB	1818.281597	38698.402344	56536.863281	57695.066406	56811.105469	0
ZWE	1818.281597	43961.839844	47128.785156	42325.117188	41081.722656	0

```
#pwt2019.fillna(method='ffill')
pwt2019.fillna(method='bfill')
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_8680\1604694572.py:2: FutureWarning: DataFrame.f
  pwt2019.fillna(method='bfill')
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.299561	0.10
AGO	Angola	Kwanza	2019	228151.015625	227855.718750	31.8
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.680527	0.01
ALB	Albania	Lek	2019	35890.019531	36103.042969	2.88
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.250000	9.77
...	...	...	...	...	...	...
VNM	Viet Nam	Dong	2019	750726.750000	724123.375000	96.4
YEM	Yemen	Yemeni Rial	2019	50052.933594	51828.058594	29.1
ZAF	South Africa	Rand	2019	748940.000000	734094.375000	58.5
ZMB	Zambia	Kwacha	2019	57956.183594	56783.714844	17.8
ZWE	Zimbabwe	US Dollar	2019	42296.062500	40826.570312	14.6

插值法（Interpolation）

除了填充给定值以外，也有更复杂的插值法。

```
pwt2019.interpolate(method="linear")
```

```
C:\Users\admin\AppData\Local\Temp\ipykernel_8680\33985437.py:1: FutureWarning: DataFrame.int
  pwt2019.interpolate(method="linear")
```

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.299561	0.10
AGO	Angola	Kwanza	2019	228151.015625	227855.718750	31.8
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.680527	0.01
ALB	Albania	Lek	2019	35890.019531	36103.042969	2.88
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.250000	9.77
...	...	...	...	...	...	...

countrycode	country	currency_unit	year	rgdpe	rgdpo
VNM	Viet Nam	Dong	2019	750726.750000	724123.37
YEM	Yemen	Yemeni Rial	2019	50052.933594	51828.058
ZAF	South Africa	Rand	2019	748940.000000	734094.37
ZMB	Zambia	Kwacha	2019	57956.183594	56783.714
ZWE	Zimbabwe	US Dollar	2019	42296.062500	40826.570

更复杂的方法涉及到模型估计问题，如 KNN 预测等。Scikit-learn 库有专门的方法，这里就不多涉及。

```
from sklearn.impute import SimpleImputer
imputer_mean = SimpleImputer(strategy='mean')
pd.DataFrame(imputer_mean.fit_transform(pwt2019.select_dtypes(np.number)), columns
```

	year	rgdpe	rgdpo	pop	emp	avh	hc
0	2019.0	3921.261230	3467.299561	0.106314	0.047601	1849.981084	2.709271
1	2019.0	228151.015625	227855.718750	31.825295	16.644962	1849.981084	1.481984
2	2019.0	376.634979	225.680527	0.014869	18.736708	1849.981084	2.709271
3	2019.0	35890.019531	36103.042969	2.880917	1.075898	1849.981084	2.964992
4	2019.0	681525.812500	645956.250000	9.770529	5.808834	1849.981084	2.746695
...	...	...	...	...	...	...	...
178	2019.0	750726.750000	724123.375000	96.462106	50.399563	2131.968232	2.869998
179	2019.0	50052.933594	51828.058594	29.161922	5.531877	1849.981084	1.842989
180	2019.0	748940.000000	734094.375000	58.558270	18.642710	2191.363362	2.908202
181	2019.0	57956.183594	56783.714844	17.861030	5.225448	1849.981084	2.686845
182	2019.0	42296.062500	40826.570312	14.645468	6.831017	1849.981084	2.713408

3.3.5 缩尾处理

应用中，常需要对异常值进行一定的处理，其中一种方法是缩尾处理（Winsorize），将极端值替换为某个百分位数的值，例如，将上限设为 99 百分位

数，下限设为 1 百分位数。

可以使用 `df.clip()` 函数实现，例如全要素生产率水平 `ctfp`：

```
q95 = pwt2019['ctfp'].quantile(0.95)
q05 = pwt2019['ctfp'].quantile(0.05)

pwt2019['ctfp'].dropna().clip(lower=q05, upper=q95, inplace=False)
```

```
countrycode
AGO    0.387996
ARG    0.828559
ARM    0.838301
AUS    0.837649
AUT    0.829206
         ...
USA    1.000000
VEN    0.266597
ZAF    0.547630
ZMB    0.266597
ZWE    0.374524
Name: ctfp, Length: 118, dtype: float64
```

3.3.6 观测值排序

有时候需要对数据集进行一定的排序，Pandas 中可以按索引 (`df.sort_index`) 和值（`df.sort_values`）排序。

例如，将索引按降序排序，这里的索引是国家代码，因此升序/降序是按照字母顺序：

```
pwt2019.sort_index(ascending=False)
```

countrycode	country	currency_unit	year	rgdpe	rgdpo
ZWE	Zimbabwe	US Dollar	2019	42296.062500	40826.570
ZMB	Zambia	Kwacha	2019	57956.183594	56783.714
ZAF	South Africa	Rand	2019	748940.000000	734094.37
YEM	Yemen	Yemeni Rial	2019	50052.933594	51828.058
VNM	Viet Nam	Dong	2019	750726.750000	724123.37
...	...	...	...	...	...
ARE	United Arab Emirates	UAE Dirham	2019	681525.812500	645956.25
ALB	Albania	Lek	2019	35890.019531	36103.042
AIA	Anguilla	East Caribbean Dollar	2019	376.634979	225.68052
AGO	Angola	Kwanza	2019	228151.015625	227855.71
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.2995

来看 `df.sort_values` 的例子，假设我们希望按 2019 年的人均 GDP（PPP 链式调整后）降序排列：

```
pwt2019['rgdp_per'] = pwt2019['rgdpe']/pwt2019['pop']
pwt2019.sort_values(by='rgdp_per', ascending=False)
```

countrycode	country	currency_unit	year	rgdpe	r
LUX	Luxembourg	Euro	2019	69541.328125	5
MAC	China, Macao SAR	Pataca	2019	67463.125000	5
QAT	Qatar	Qatari Rial	2019	292963.531250	3
IRL	Ireland	Euro	2019	499741.093750	5
SGP	Singapore	Singapore Dollar	2019	514376.312500	4
...	...	...	...	...	..
MWI	Malawi	Kwacha	2019	20362.392578	2
COD	D.R. of the Congo	Franc Congolais	2019	89061.671875	8
CAF	Central African Republic	CFA Franc BEAC	2019	4532.561035	4
BDI	Burundi	Burundi Franc	2019	8664.988281	9

countrycode	country	currency_unit	year	rgdpe	rgdpo
VEN	Venezuela (Bolivarian Republic of)	Bolivar Fuerte	2019	7166.571777	7160.106934

3.3.7 数据集合并

实际应用中，数据可能来自不同的来源，经常需要合并数据集，`pd.merge()`函数

```
import wbgapi as wb
inf = wb.data.DataFrame(series='NY.GDP.DEFL.KD.ZG', time='2019')
pd.merge(df[['country','pop','emp']], inf, left_index=True, right_index=True)
```

	country	pop	emp	NY.GDP.DEFL.KD.ZG
ABW	Aruba	0.106314	0.047601	6.017818
AGO	Angola	31.825295	16.644962	19.187004
ALB	Albania	2.880917	1.075898	1.000633
ARE	United Arab Emirates	9.770529	5.808834	-3.194409
ARG	Argentina	44.780677	20.643215	49.195579
...	...	...	...	...
VNM	Viet Nam	96.462106	50.399563	2.423227
YEM	Yemen	29.161922	5.531877	NaN
ZAF	South Africa	58.558270	18.642710	4.613525
ZMB	Zambia	17.861030	5.225448	7.633470
ZWE	Zimbabwe	14.645468	6.831017	225.394837

3.3.8 多级索引

这里的数据是一个面板数据，“国家-年” 对应一个观测值，可以利用 Pandas 的多级索引功能，详见 Pandas 文档MultiIndex / advanced indexing。

```
pwt = pd.read_excel(io = "datasets/pwt1001.xlsx",
                    header=0,
                    sheet_name="Data")
pwt.set_index(['countrycode','year'], inplace=True)
```

我们可以使用.loc() 方法选择需要的数据，例如：

```
# 中国子集
df_china = pwt.loc['CHN']
# 中国、美国子集
df_china_us = pwt.loc[['CHN','USA']]
# 变量子集
df_sub_china_us = pwt.loc[['CHN', 'USA']][['rgdpe','rgdpo']]
```

如果需要选择某一年的截面数据：

```
pwt.loc[(slice(None), [2019]), :]
# 1992 年之后的数据
pwt.loc[(slice(None), slice(1992, None)), :]
```

countrycode	year	country	c
	1992	Aruba	A
	1993	Aruba	A
ABW	1994	Aruba	A
	1995	Aruba	A
	1996	Aruba	A
...	...	...	...
	2015	Zimbabwe	U
	2016	Zimbabwe	U
ZWE	2017	Zimbabwe	U
	2018	Zimbabwe	U
	2019	Zimbabwe	U

这里使用了 `df.loc` 结合 `slice` 函数的方法，注意：

- `slice(None)`: 这表示选择所有 `countrycode`。
- `slice(1992, None)`: 这表示从 `year` 的 1992 年开始，选择到 **所有**后续年份。由于索引是排序的（通常情况下），这有效地选择了所有 `year > 1992` 的数据。
- `:` 表示选择所有列。

上面的例子使用 `slice` 函数不是那么直观,也可以使用 `df.index.get_level_values('year')` 提取索引 `year` 的值，形成一个序列（可以另存为一个变量），然后利用表达式生成一个布尔序列，对数据框进行筛选：

```
pwt[pwt.index.get_level_values('year') > 1992]
```

		country	currency_un
countrycode	year		
	1993	Aruba	Aruban Guil
	1994	Aruba	Aruban Guil
ABW	1995	Aruba	Aruban Guil
	1996	Aruba	Aruban Guil
	1997	Aruba	Aruban Guil
...	...	...	...
	2015	Zimbabwe	US Dollar
	2016	Zimbabwe	US Dollar
ZWE	2017	Zimbabwe	US Dollar
	2018	Zimbabwe	US Dollar
	2019	Zimbabwe	US Dollar

当然，可以同时选择指定的变量和年份，例如：

```
pwt.loc[(slice(None),[2016,2019]), ['rgdpe','rgdpo']]
#
pwt.loc[((["CHN", "USA"], [2016,2019])), ['rgdpe','rgdpo']]
```

countrycode	year	rgdpe
CHN	2016	18611202.0
	2019	20056066.0
USA	2016	19285252.0
	2019	20860506.0

除了通常的排序以外，由于有了二级索引，如果按索引排序，两级索引变量是同时排序的：

```
pwt.sort_index()
```

countrycode	year	country	c
ABW	1950	Aruba	A
	1951	Aruba	A
	1952	Aruba	A
	1953	Aruba	A
	1954	Aruba	A
...	...	...	...
ZWE	2015	Zimbabwe	U
	2016	Zimbabwe	U
	2017	Zimbabwe	U
	2018	Zimbabwe	U
	2019	Zimbabwe	U

可以对两级索引以列表的形式分别设定排序的顺序。例如，先将国家代码按字母升序，然后将年降序：

```
pwt.sort_index(ascending=[True, False])
```

countrycode	year	country	currency_un
	2019	Aruba	Aruban Guil
	2018	Aruba	Aruban Guil
ABW	2017	Aruba	Aruban Guil
	2016	Aruba	Aruban Guil
	2015	Aruba	Aruban Guil
...	...	...	...
	1954	Zimbabwe	US Dollar
	1953	Zimbabwe	US Dollar
ZWE	1952	Zimbabwe	US Dollar
	1951	Zimbabwe	US Dollar
	1950	Zimbabwe	US Dollar

3.3.9 stack 和 unstack

数据有“长（long）”和“宽（wide）”两种组织方式，Penn World Table 是以“长”的形式保存的。有时候需要在两种数据格式之间进行转换，就需要用到 `df.stack()` 和 `df.unstack()` 函数。

注意，`df.unstack()` 函数的参数 `level=`，设置为哪一级索引，便生成为列。默认在最后一级索引上转换，即年，因此列便为年，行为国家，反之，列为国家，行为年。如下面例子所示，为了简便只保留了三个国家 5 年的数据：

```
pwt_sub = pwt.loc[(["CHN", "KOR", "USA"], slice(2015, None)), ["rgdpe", "pop"]]
#
pwt_sub_wide = pwt_sub.unstack(level=-1)
# pwt_sub.unstack(level=0)
```

要获得长格式的数据，使用 `df.stack()` 即可：

```
pwt_sub_wide.stack(future_stack=True)
```

		rgdpe
countrycode	year	
CHN	2015	1.786628e+07
	2016	1.861120e+07
	2017	1.950114e+07
	2018	1.950871e+07
	2019	2.005607e+07
KOR	2015	1.928057e+06
	2016	1.999700e+06
	2017	2.070936e+06
	2018	2.102052e+06
	2019	2.090946e+06
USA	2015	1.890512e+07
	2016	1.928525e+07
	2017	1.975475e+07
	2018	2.036944e+07
	2019	2.086051e+07

当我们从一些数据库下载数据时，常见形式为列为不同时期相同变量的值。例如，从世界银行下载人均 GDP 和人口数据：

```
import wbgapi as wb
df = wb.data.DataFrame(series=['NY.GDP.PCAP.CD', "SP.POP.TOTL"],
                                #time=range(2017,2020),
                                time=['YR2017','YR2018','YR2019'],
                                 numericTimeKeys=True)
df.head()
```

economy	series
ABW	NY.GDP.PCAP.CD
	SP.POP.TOTL
AFE	NY.GDP.PCAP.CD

economy	series	2017
AFG	SP.POP.TOTL	6.400587e-
	NY.GDP.PCAP.CD	5.254698e-

下载的数据 `df` 索引是 “economy - series”，每一年数据一列。我们希望序列成为列变量，时间成为索引。我们可以先对数据进行转置成宽格式的数据，然后再在国家层面堆叠，使其成为索引，再交换索引排序得到通常的情况：

```
df.T.stack(level=0, future_stack=True).swaplevel().sort_index()
```

economy	series	NY.GDP.PCAP.CD	SI
ABW	2017	28440.051964	10
	2018	30082.127645	10
	2019	31096.205074	10
AFE	2017	1520.212231	64
	2018	1538.901679	65
...	...	...	...
ZMB	2018	1463.899979	17
	2019	1258.986198	18
ZWE	2017	3448.086991	14
	2018	2271.852504	15
	2019	1683.913136	15

另外，stack 不是唯一的方法，也可以使用 `df.melt()` 结合 `df.pivot_table()` 函数来实现：

```
df_reset = df.reset_index()
df_long = df_reset.melt(id_vars=['economy', 'series'], var_name='year', value_name='value')
df_long.pivot_table(index=['economy', 'year'], columns='series', values='value')
```

economy	series year	NY.GDP.PC
ABW	2017	28440.051964
	2018	30082.127645
	2019	31096.205074
AFE	2017	1520.212231
	2018	1538.901679
...	...	...
ZMB	2018	1463.899979
	2019	1258.986198
ZWE	2017	3448.086991
	2018	2271.852504
	2019	1683.913136

3.3.10 Pandas 中的分组计算（groupby）

Pandas 的分组（`groupby()`）方法按照“分割-应用-组合（split-apply-combine）”的原理，创建一个 groupby 对象，可以应用各种方法来聚合、转换或过滤数据。更多介绍参见 Pandas 官方文档Group by: split-apply-combine。

选择合适的方法：

- 如果你的操作只是简单的统计（如求和、平均值），优先使用聚合方法，它们通常效率最高。
- 如果需要返回与原始 DataFrame 相同长度的结果，例如进行组内标准化，使用转换方法。
- 如果需要根据组的属性来决定保留或丢弃整个组，使用过滤方法。
- 当以上方法都无法满足需求时，或者需要执行更复杂的自定义逻辑时，使用 **apply()** 方法。

3.3.10.1 聚合方法（Aggregation Methods）

聚合方法将每个组的数据压缩成一个单一的值，是最常用的 groupby 操作，例 如 mean(),sum(),count(),size(),min(),max(),std(),var(),median() 等常见的统计量，或者 first(),last(),nth(n) 等获取第一个、最好一个或第 n 个值：

索引

例如，根据索引计算世界人口，先在索引上分组，然后使用.sum() 函数：

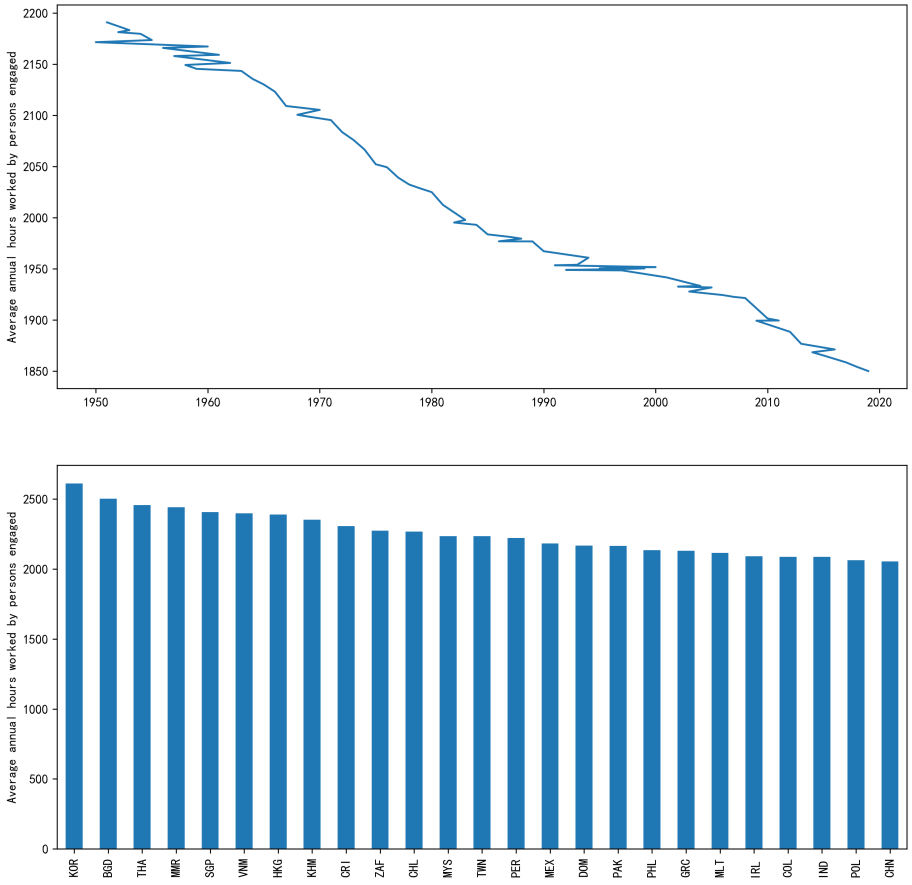
```
pwt.groupby(level=1)['pop'].sum()
```

```
year
1950    1297.363356
1951    1345.648916
1952    1948.874249
1953    2005.091897
1954    2048.591355
           ...
2015    7254.659556
2016    7336.956076
2017    7418.960776
2018    7500.383052
2019    7580.896719
Name: pop, Length: 70, dtype: float64
```

avh 变量度量了 “Average annual hours worked by persons engaged”, 让我们分组计算平均，得到按年和按国家平均

```
avh = pwt[pwt['avh'].notna()]
fig, ax = plt.subplots(2, 1, figsize=(12, 12))
avh.groupby(level=1)['avh'].mean().sort_values(ascending=False).plot(kind='line', ax=ax[0])
ax[0].set_xlabel("")
ax[0].set_ylabel("Average annual hours worked by persons engaged")
avh.groupby(level=0)['avh'].mean().sort_values(ascending=False)[:25].plot(kind='bar', ax=ax[
```

```
ax[1].set_xlabel("")
ax[1].set_ylabel("Average annual hours worked by persons engaged")
plt.show()
```



最常见的是按变量进行分组，例如，按国家名 `country` 分组，最后一个观测值：

```
pwt.groupby(by=['country']).last()
```

country	currency_unit	rgdpe	rgdpo	po
Albania	Lek	35890.019531	36103.042969	2.
Algeria	Algerian Dinar	488952.375000	507487.562500	43

country	currency_unit	rgdpe	rgdpo	pop	e
Angola	Kwanza	228151.015625	227855.718750	31.825295	1
Anguilla	East Caribbean Dollar	376.634979	225.680527	0.014869	0
Antigua and Barbuda	East Caribbean Dollar	1986.163208	1603.854492	0.097118	0
...	...	...	...	...	..
Venezuela (Bolivarian Republic of)	Bolivar Fuerte	7166.571777	7160.106934	28.515829	1
Viet Nam	Dong	750726.750000	724123.375000	96.462106	5
Yemen	Yemeni Rial	50052.933594	51828.058594	29.161922	5
Zambia	Kwacha	57956.183594	56783.714844	17.861030	5
Zimbabwe	US Dollar	42296.062500	40826.570312	14.645468	6

3.3.10.2 转换方法（Transformation Methods）

- `transform(func)`: 对每个组应用函数，并将结果广播回原始 DataFrame 的形状。
- `rank(method='average')`: 计算组内排名。
- `fillna(value)`: 在组内填充缺失值。

```
avh.groupby(level=1)['avh'].transform('mean')
avh.groupby(level=1)['avh'].mean()
```

```
year
1950    2171.439158
1951    2190.832106
1952    2181.242069
1953    2183.205302
1954    2179.603764
           ...
2015    1865.220762
2016    1871.137771
2017    1858.542897
2018    1854.065910
```

```
2019    1849.981084
Name: avh, Length: 70, dtype: float64
```

注意，转换与聚合的区别, 转换将生成的值与原数据观测值一样多，这里是 3492 个，而聚合只有 70 个。

`.transform()` 方法可以与 `lambda` 函数相结合，例如：

```
pwt.select_dtypes(np.number).groupby(level=0).transform(lambda x: (x - x.mean())/x
```

		rgdpe	rg
countrycode	year		
	1950	NaN	Na
	1951	NaN	Na
ABW	1952	NaN	Na
	1953	NaN	Na
	1954	NaN	Na
...	...	...	...
	2015	0.557612	0.5
	2016	0.653705	0.6
ZWE	2017	0.808743	0.7
	2018	0.789505	0.7
	2019	0.677034	0.5

3.3.10.3 过滤方法（Filtration Methods）

过滤方法会根据每个组的某个条件来排除整个组。

- filter(func): 根据一个返回布尔值的函数来过滤组。如果函数对一个组返回 True，则保留该组；否则，删除该组。

```
pwt.groupby(level=0).filter(lambda x: x['pop'].mean() > 50)
```

countrycode	year	country	currency_u
	1950	Bangladesh	Taka
	1951	Bangladesh	Taka
BGD	1952	Bangladesh	Taka
	1953	Bangladesh	Taka
	1954	Bangladesh	Taka
...	...	...	...
	2015	Viet Nam	Dong
	2016	Viet Nam	Dong
VNM	2017	Viet Nam	Dong
	2018	Viet Nam	Dong
	2019	Viet Nam	Dong

3.3.10.4 应用方法（Application Methods）

apply() 方法是最通用的方法，它允许你对每个组应用任何自定义函数。这个函数可以执行聚合、转换或过滤操作，或者任何更复杂的逻辑。

- apply(func): 将一个自定义函数应用于每个组。函数的返回值可以是 Series、DataFrame 或标量。

第四章　后记

In summary, this book has no content whatsoever.

参考文献

[1] Robert C Feenstra, Robert Inklaar, and Marcel P Timmer. “The next generation of the Penn World Table”. In: *American economic review* 105.10 (2015), pp. 3150–3182.

[2] Wes McKinney. *Python for data analysis: Data wrangling with pandas, numpy, and jupyter*. ” O’Reilly Media, Inc.”, 2022.

[3] Jake VanderPlas. *Python data science handbook: Essential tools for working with data*. ” O’Reilly Media, Inc.”, 2016.