Numpy 和 Pandas 基础

Table of Contents

0.1 Numpy 基础

0.1.1 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 1 所示：

Table 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
`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)
```

```
[[-0.00637526  0.1398649  -0.32859433]
 [-2.15813855 -0.76865789  0.43287538]
 [-0.84919002 -0.45378588 -0.28117915]
 [-0.87308064  1.43878943 -1.63986651]]
[[2 9 4 2 4 2 9 2 2 5]
 [7 1 9 6 1 4 2 8 9 1]]
```

0.1.2 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])
```

0.1.3 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])
```

0.1.4 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])
```

0.1.5 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)
```

0.1.6 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。

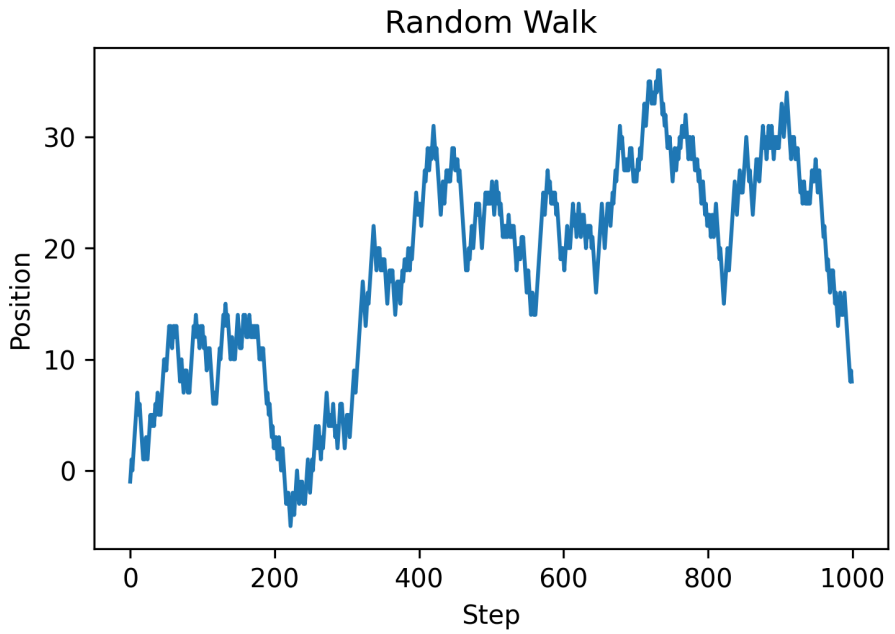
```
import numpy as np

# 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)
```

```
banana
```

0.1.6.1 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()
```



0.1.6.2 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。

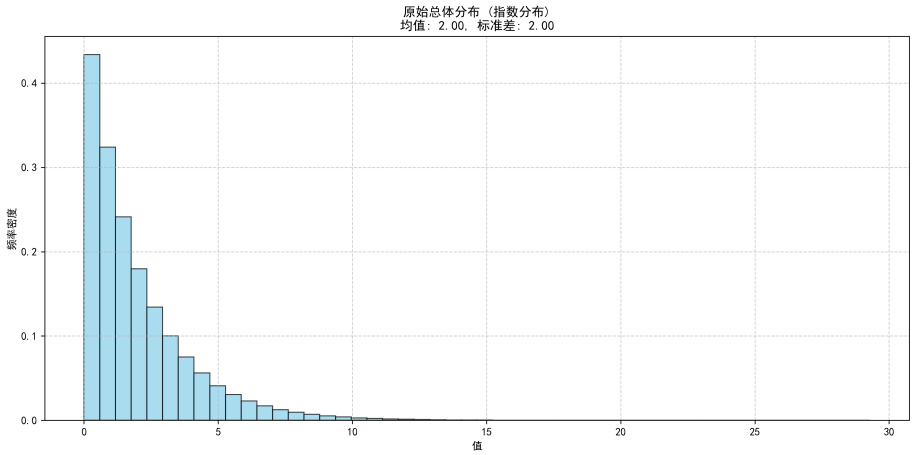
我们将通过以下步骤来模拟验证 CLT：

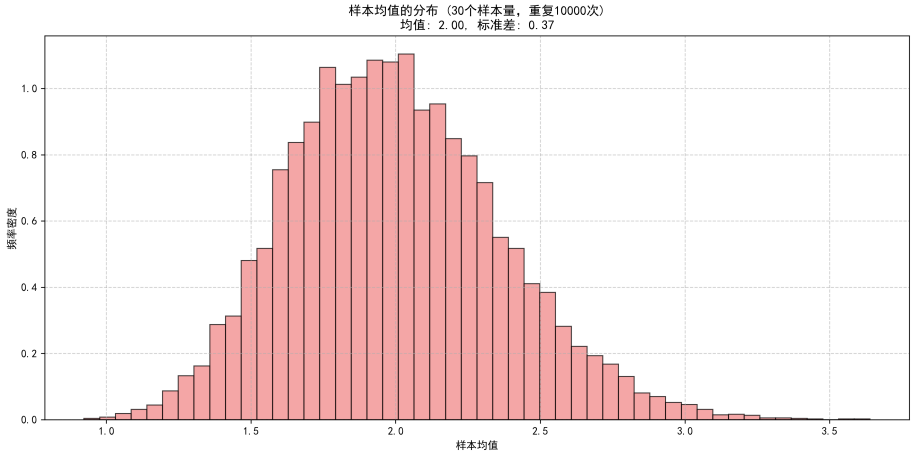
- 选择一个非正态分布的总体: 比如，一个指数分布或均匀分布，它们的形状都不是钟形的。
- 设置样本参数: 定义每次抽样的样本大小 (sample_size) 和重复抽样的次数 (num_samples)。
- 重复抽样并计算均值: 从总体中抽取 num_samples 次样本，每次抽取 sample_size 个数据点，并计算每次抽样的平均值。
- 可视化: 绘制样本均值的直方图，并与原始总体分布的直方图进行对比。

```
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_size)

# --- 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
plt.title(f'原始总体分布 (指数分布)\n均值: {np.mean(population_data):.2f}, 标准差: {
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
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.me
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
```

0.1.7 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 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

0.1.8 例：通用函数

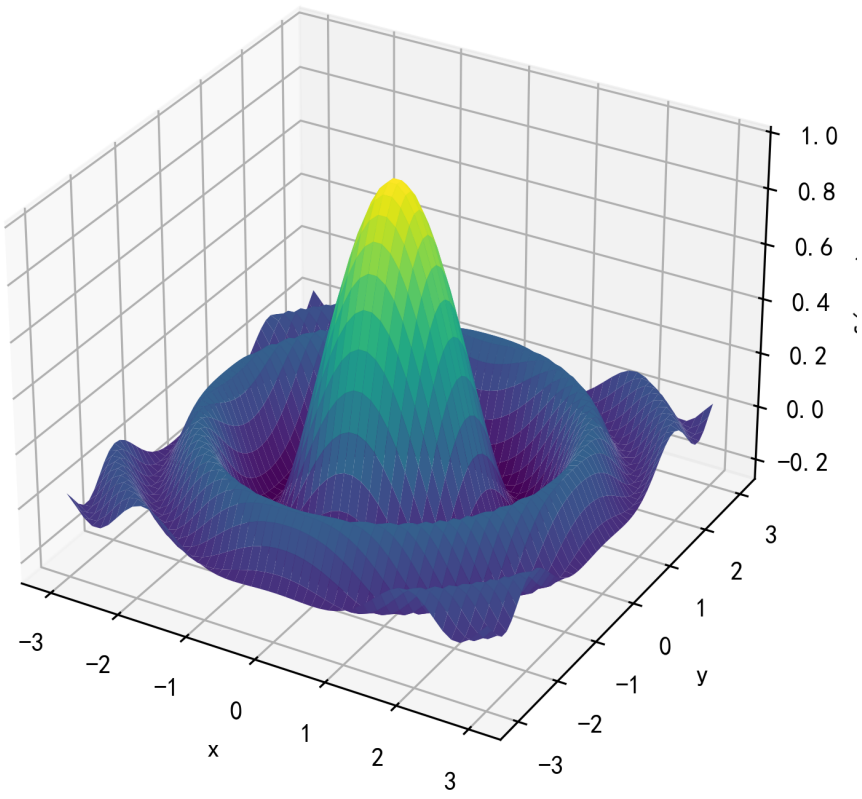
考察最大化函数 $f(x, y)$ 在区间 $ [−a,a] ×[−a, a]$ 上的最大值：

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

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
```



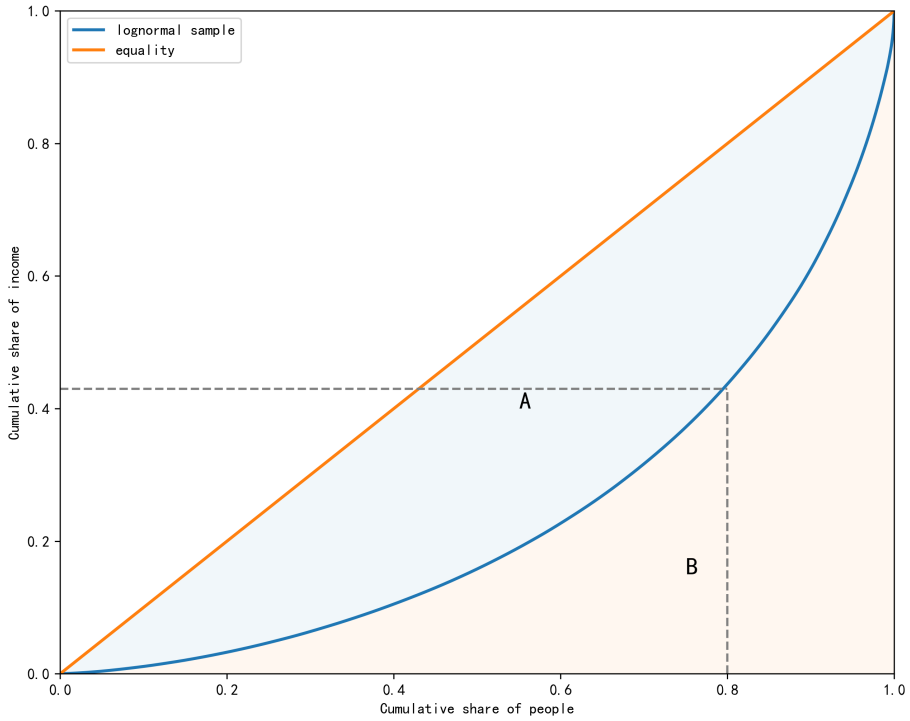
图示

0.1.9 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()
```



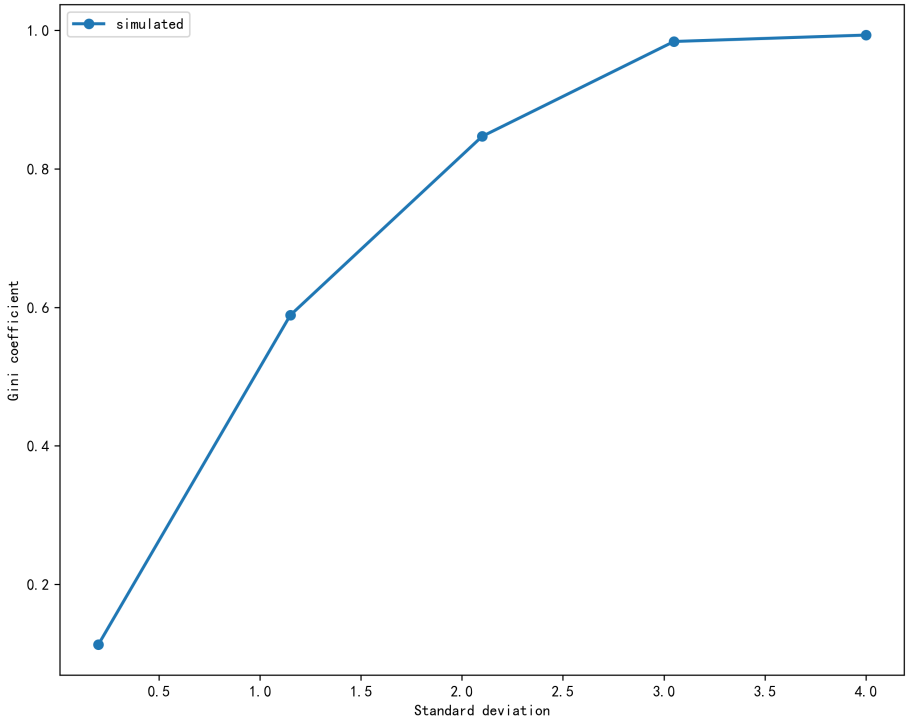
0.1.10 基尼系数

```
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()
```



0.2 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。

0.2.1 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；

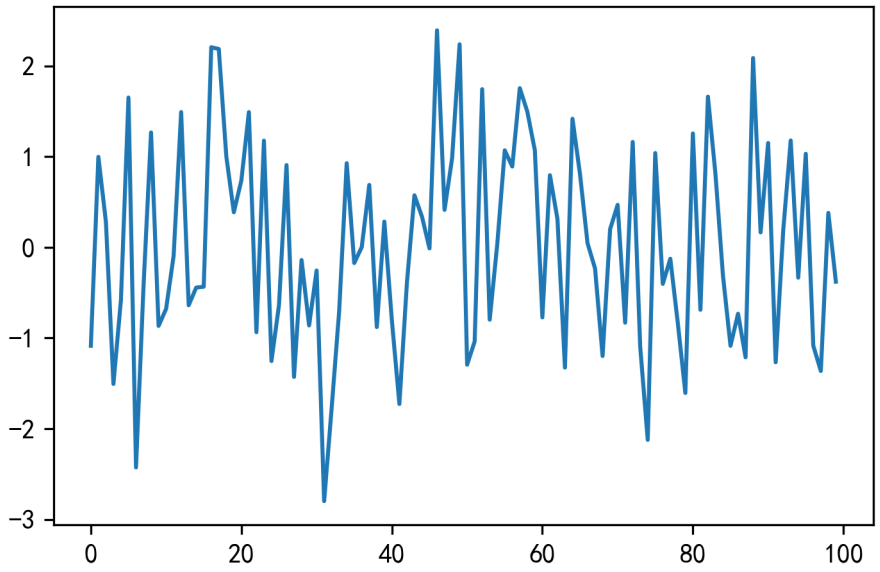
Series 可以看作一 “列” 数据；

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；

0.3 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。

0.3.1 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
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.2995
AGO	Angola	Kwanza	2019	228151.015625	227855.71
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

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')
```

0.3.2 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	pop
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

要选择**列**，可以用包含列名字的列表：

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

0.3.2.1 `.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
CHN	China	Yuan Renminbi	2019	20056066.0	2.025766e+07	1433.7
BRA	Brazil	Brazilian Real	2019	3089273.5	3.080048e+06	211.04
RUS	Russian Federation	Russian Ruble	2019	4197222.5	4.161194e+06	145.87
IND	India	Indian Rupee	2019	8945547.0	9.170555e+06	1366.4
ZAF	South Africa	Rand	2019	748940.0	7.340944e+05	58.558

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

0.3.2.2 `.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。

0.3.2.3 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
BRA	Brazil	Brazilian Real	2019	3.089274e+06	3.080048e+06	211.04952
CHN	China	Yuan Renminbi	2019	2.005607e+07	2.025766e+07	1433.7836
IDN	Indonesia	Rupiah	2019	3.104439e+06	3.137931e+06	270.62556
IND	India	Indian Rupee	2019	8.945547e+06	9.170555e+06	1366.4177
NGA	Nigeria	Naira	2019	9.834982e+05	1.001537e+06	200.96359
PAK	Pakistan	Pakistan Rupee	2019	1.036800e+06	1.088502e+06	216.56531
USA	United States	US Dollar	2019	2.086051e+07	2.059584e+07	329.06491

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	avh
CHN	China	Yuan Renminbi	2019	20056066.0	20257660.0	1433.783686	798.807739	2168
IND	India	Indian Rupee	2019	8945547.0	9170555.0	1366.417754	497.615723	2122

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	e
DEU	Germany	Euro	2019	4.308862e+06	4275312.00	83.517045	4
FRA	France	Euro	2019	3.018885e+06	2946958.25	67.351247	2
GBR	United Kingdom	Pound Sterling	2019	3.118991e+06	2989895.50	67.530172	3
ITA	Italy	Euro	2019	2.508404e+06	2466327.50	60.550075	2
JPN	Japan	Yen	2019	5.028348e+06	5036891.00	126.860301	6
KOR	Republic of Korea	Won	2019	2.090946e+06	2162705.25	51.225308	2
RUS	Russian Federation	Russian Ruble	2019	4.197222e+06	4161194.50	145.872256	7
TUR	Turkey	New Turkish Lira	2019	2.227538e+06	2248225.75	83.429615	2
USA	United States	US Dollar	2019	2.086051e+07	20595844.00	329.064917	1

0.3.3 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。

0.3.4 检测和处理缺失值

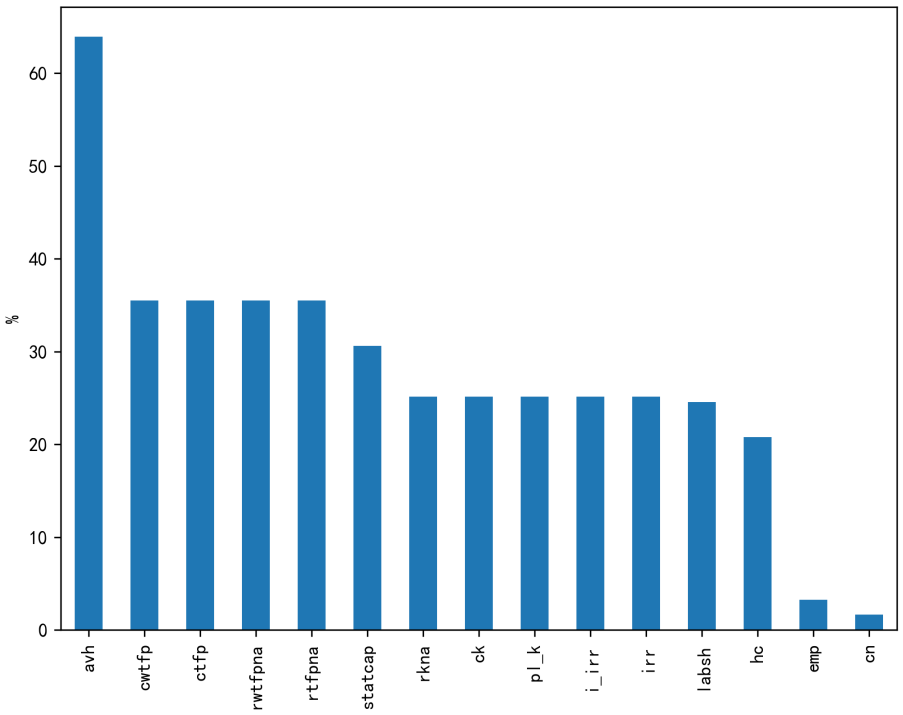
Pandas 中最常用的缺失值表示是 `NaN`（Not a Number）。可以使用 `isnull()` 或 `isna()` 函数检测缺失值，返回一个布尔型的 DataFrame，其中 `True` 表示缺失值：

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

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

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：

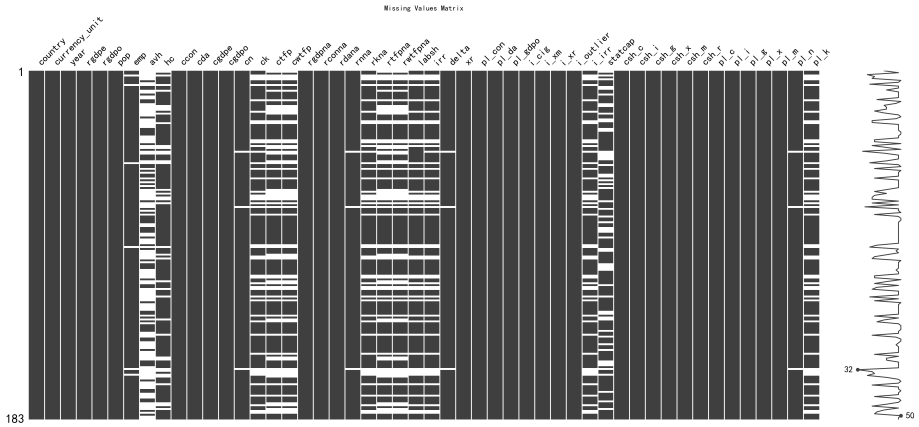
```
fig, ax = plt.subplots(figsize=(8, 6))
(pwt2019.isna().sum()/pwt2019.shape[0]*100).sort_values(ascending=False)[:15].plot(kind='bar
ax.set_ylabel("%")
plt.show()
```



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
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.2995
AGO	Angola	Kwanza	2019	228151.015625	227855.71
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
...	...	...	...	...	...
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

country	currency_unit	year	rgdpe	rgdpo	pop

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_first(pwt20
pwt2019.select_dtypes(np.number).fillna(pwt2019.median(numeric_only=True)).combine_first(pwt
```

countrycode	avh	ccon	cda	cgdpe	cgdpo	ck	cr
ABW	1818.281597	3023.694824	3877.659668	3912.334717	3466.241943	0.000209	1.
AGO	1818.281597	155943.718750	198750.421875	227771.609375	223289.312500	0.016624	1.
AIA	1818.281597	438.470032	509.044983	375.136444	241.384537	0.005160	2.
ALB	1818.281597	33399.167969	40868.316406	35808.343750	36288.328125	0.005160	2.
ARE	1818.281597	306771.156250	515623.312500	678241.187500	635332.812500	0.005160	4.
...	...	...	...	...	...	...	...
VNM	2131.968232	582677.062500	758821.937500	747853.750000	723142.687500	0.005160	1.
YEM	1818.281597	49266.472656	67992.531250	49937.042969	51983.429688	0.005160	5.
ZAF	2191.363362	623669.562500	741675.937500	748245.937500	735067.062500	0.033228	2.
ZMB	1818.281597	38698.402344	56536.863281	57695.066406	56811.105469	0.004523	2.
ZWE	1818.281597	43961.839844	47128.785156	42325.117188	41081.722656	0.000733	5.

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

```
C:\Users\admin\AppData\Local\Temp\ipykernel_17816\1604694572.py:2: FutureWarning:
  pwt2019.fillna(method='bfill')
```

countrycode	country	currency_unit	year	rgdpe	rgdpo
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.2995
AGO	Angola	Kwanza	2019	228151.015625	227855.71
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
...	...	...	...	...	...
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

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_17816\33985437.py:1: FutureWarning: Da
  pwt2019.interpolate(method="linear")
```

countrycode	country	currency_unit	year	rgdpe	rgdpo
ABW	Aruba	Aruban Guilder	2019	3921.261230	3467.2995
AGO	Angola	Kwanza	2019	228151.015625	227855.71
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
...	...	...	...	...	...

countrycode	country	currency_unit	year	rgdpe	rgdpo	pop
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

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=pwt2019.s
```

	year	rgdpe	rgdpo	pop	emp	avh	hc	ccon
0	2019.0	3921.261230	3467.299561	0.106314	0.047601	1849.981084	2.709271	3023.69482
1	2019.0	228151.015625	227855.718750	31.825295	16.644962	1849.981084	1.481984	155943.718
2	2019.0	376.634979	225.680527	0.014869	18.736708	1849.981084	2.709271	438.470032
3	2019.0	35890.019531	36103.042969	2.880917	1.075898	1849.981084	2.964992	33399.1679
4	2019.0	681525.812500	645956.250000	9.770529	5.808834	1849.981084	2.746695	306771.156
...	...	...	...	...	...	...	...	...
178	2019.0	750726.750000	724123.375000	96.462106	50.399563	2131.968232	2.869998	582677.062
179	2019.0	50052.933594	51828.058594	29.161922	5.531877	1849.981084	1.842989	49266.4726
180	2019.0	748940.000000	734094.375000	58.558270	18.642710	2191.363362	2.908202	623669.562
181	2019.0	57956.183594	56783.714844	17.861030	5.225448	1849.981084	2.686845	38698.4023
182	2019.0	42296.062500	40826.570312	14.645468	6.831017	1849.981084	2.713408	43961.8398

0.3.5 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
```

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

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	rgdpo
LUX	Luxembourg	Euro	2019	69541.328125	55710.792969
MAC	China, Macao SAR	Pataca	2019	67463.125000	59874.164062
QAT	Qatar	Qatari Rial	2019	292963.531250	323141.15625
IRL	Ireland	Euro	2019	499741.093750	501053.59375
SGP	Singapore	Singapore Dollar	2019	514376.312500	477907.87500
...	...	...	...	...	...
MWI	Malawi	Kwacha	2019	20362.392578	21635.066406
COD	D.R. of the Congo	Franc Congolais	2019	89061.671875	88673.375000
CAF	Central African Republic	CFA Franc BEAC	2019	4532.561035	4642.448730
BDI	Burundi	Burundi Franc	2019	8664.988281	9109.688477

countrycode	country	currency_unit	year	rgdpe	r
VEN	Venezuela (Bolivarian Republic of)	Bolivar Fuerte	2019	7166.571777	7

0.3.7 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

0.3.8 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	currency_un
	1992	Aruba	Aruban Guil
	1993	Aruba	Aruban Guil
ABW	1994	Aruba	Aruban Guil
	1995	Aruba	Aruban Guil
	1996	Aruba	Aruban Guil
...	...	...	...
	2015	Zimbabwe	US Dollar
	2016	Zimbabwe	US Dollar
ZWE	2017	Zimbabwe	US Dollar
	2018	Zimbabwe	US Dollar
	2019	Zimbabwe	US Dollar

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]
```

countrycode	year	country	c
	1993	Aruba	A
	1994	Aruba	A
ABW	1995	Aruba	A
	1996	Aruba	A
	1997	Aruba	A
...	...	...	...
	2015	Zimbabwe	U
	2016	Zimbabwe	U
ZWE	2017	Zimbabwe	U
	2018	Zimbabwe	U
	2019	Zimbabwe	U

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	rgdpo
CHN	2016	18611202.0	18591710.0
	2019	20056066.0	20257660.0
USA	2016	19285252.0	19095196.0
	2019	20860506.0	20595844.0

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

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

0.3.9 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)
```

countrycode	year	rgdpe	pop
CHN	2015	1.786628e+07	1406.8478
	2016	1.861120e+07	1414.0493
	2017	1.950114e+07	1421.0217
	2018	1.950871e+07	1427.6477
	2019	2.005607e+07	1433.7836
KOR	2015	1.928057e+06	50.823093
	2016	1.999700e+06	50.983457
	2017	2.070936e+06	51.096415
	2018	2.102052e+06	51.171706
	2019	2.090946e+06	51.225308
USA	2015	1.890512e+07	320.87831
	2016	1.928525e+07	323.01599
	2017	1.975475e+07	325.08475
	2018	2.036944e+07	327.09626
	2019	2.086051e+07	329.06491

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	2017
ABW	NY.GDP.PCAP.CD	2.844005e-
	SP.POP.TOTL	1.087350e-
AFE	NY.GDP.PCAP.CD	1.520212e-

economy	series
AFG	SP.POP.TOTL
	NY.GDP.PCAP.CD

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.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

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
df_long.pivot_table(index=['economy', 'year'], columns='series', values='value')
```

economy	series year	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

0.3.10 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。

0.3.10.1 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：

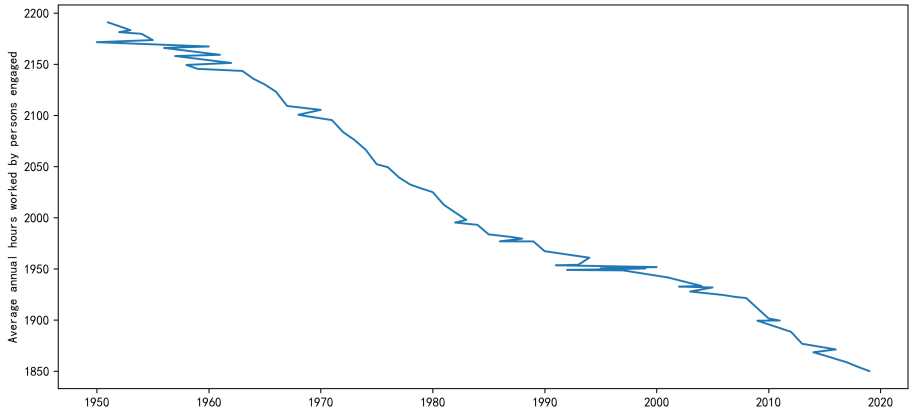
```
pwt.groupby(level=1)['pop'].sum()
```

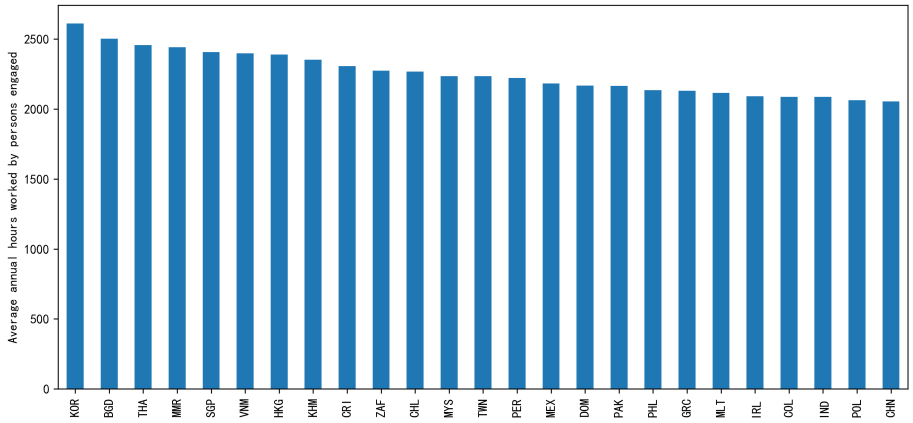
```
year
1950    1297.363356
1951    1345.648916
1952    1948.874249
1953    2005.091897
1954    2048.591355
           ...
2015    7254.659556
2016    7336.956076
2017    7418.960776
2018    7500.383052
2019    7580.896719
Name: pop, Length: 70, dtype: float64
```

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[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='ba
```

```
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	pop	e
Albania	Lek	35890.019531	36103.042969	2.880917	1
Algeria	Algerian Dinar	488952.375000	507487.562500	43.053054	1

country	currency_unit	rgdpe	rgdpo	pc
Angola	Kwanza	228151.015625	227855.718750	31
Anguilla	East Caribbean Dollar	376.634979	225.680527	0.
Antigua and Barbuda	East Caribbean Dollar	1986.163208	1603.854492	0.
...	...	...	...	...
Venezuela (Bolivarian Republic of)	Bolivar Fuerte	7166.571777	7160.106934	28
Viet Nam	Dong	750726.750000	724123.375000	96
Yemen	Yemeni Rial	50052.933594	51828.058594	29
Zambia	Kwacha	57956.183594	56783.714844	17
Zimbabwe	US Dollar	42296.062500	40826.570312	14

0.3.10.2 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.std())
```

		rgdpe	rgdpo	po
countrycode	year			
	1950	NaN	NaN	Na
	1951	NaN	NaN	Na
ABW	1952	NaN	NaN	Na
	1953	NaN	NaN	Na
	1954	NaN	NaN	Na
...	...	...	...	...
	2015	0.557612	0.538962	1.3
	2016	0.653705	0.602805	1.4
ZWE	2017	0.808743	0.786654	1.4
	2018	0.789505	0.737542	1.5
	2019	0.677034	0.595315	1.5

0.3.10.3 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
	1950	Bangladesh
	1951	Bangladesh
BGD	1952	Bangladesh
	1953	Bangladesh
	1954	Bangladesh
...	...	...
	2015	Viet Nam
	2016	Viet Nam
VNM	2017	Viet Nam
	2018	Viet Nam
	2019	Viet Nam

0.3.10.4 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。

Bibliography

[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.