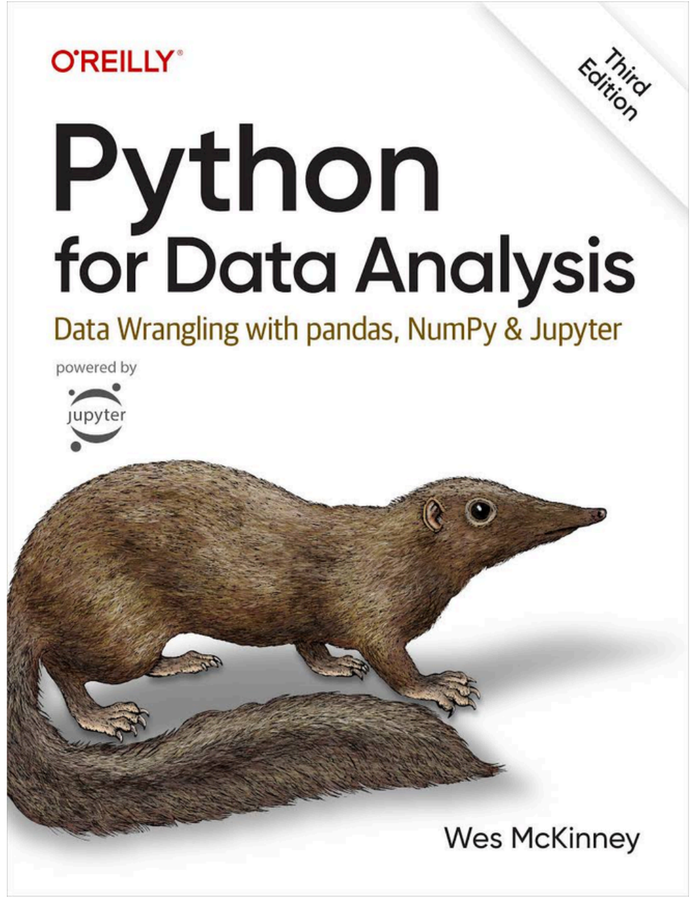
# 專書報告

## Python資料分析之資料清理與準備

組長:陳鈺昕
組員:賴兆信, 黃子晏, 宋苡瑄, 黃天芸




Python for Data Analysis

# 本書結構

| | 內容範疇 | 說明 |
|---|---|---|
| **基礎技能** | Python 語言、IPython、Jupyter、NumPy | 建立數據分析需要的 Python 程式能力與計算基礎 |
| **資料操作** | pandas、文件讀寫、清理、合併、分組、重塑、時間序列 | 教你用 pandas 高效進行資料清理、轉換與準備 |
| **分析與表達** | 繪圖、案例分析、展示結果 | 完整演練數據分析流程，並將結果以圖表、報表或數據模型呈現 |

# 此書的總目標

不是教你統計學，也不是教你機器學習，而是教你如何用 Python 建立乾淨、結構良好且可以被後續分析的數據表

這是一本教你用 Python 和 pandas，從混亂資料中萃取出清晰、可操作知識的實戰指南

**標準流程化**　讀取→清理→轉換→分析→可視化 這一條龍的標準數據處理流程

# 章節主題：資料清洗與轉換

# 書內容項目

處理缺失資料

資料轉換

延伸資料型別

字串操作

類別資料

# 資料清洗與轉換

本章強調『清洗與轉換』的核心任務包括：
將原始數據中不一致、不完整、不正確或不適合分析的部分，做處理

這裡面分成兩大動作：

清洗（Cleaning）：找出並處理缺失值、重複、錯誤、不合理值。

轉換（Transforming）：變更資料型態、結構、語意，讓資料適配後續分析。

# 缺失處理 (Missing Data)

在真實世界資料中，缺失值普遍且結構化，常見於：
問卷、醫療紀錄、交易系統、IoT感測器等。

| 缺失機制 | 說明 | 處理建議 |
| --- | --- | --- |
| **MCAR** | 完全隨機缺失 | 可刪除樣本 |
| **MAR** | 條件性缺失 | 模型插補 |
| **MNAR** | 非隨機缺失 | 應建模處理，<br>**不可直接補值** |

# 缺失處理

pandas 從設計之初，即支援：

.isna() / .notna() 缺失檢測

.dropna() / .fillna() 缺失處理

允許橫向與縱向多種策略

可用 df.isna().astype('int') 產生 缺失指標，加入特徵集合以評估缺失本身的預測力。

4 rows × 3 columns

| | feature1 | feature2 | target |
|---|---|---|---|
| 0 | 1.0 | NaN | 0 |
| 1 | 2.0 | 2.0 | 1 |
| 2 | NaN | 3.0 | 0 |
| 3 | 4.0 | 4.0 | 1 |

```
# 處理
X = df.drop('target', axis=1)
X_miss_ind = X.isna().add_prefix('isna_').astype('int')
X_filled = X.fillna(-999)
X_aug = pd.concat([X_filled, X_miss_ind], axis=1)
```

保留原始特徵

增加缺失指標欄位

讓模型學習「缺失模式」與預測目標的關聯

4 rows × 4 columns

| | feature1 | feature2 | isna_feature1 | isna_feature2 |
|---|---|---|---|---|
| 0 | 1.0 | -999.0 | 0 | 1 |
| 1 | 2.0 | 2.0 | 0 | 0 |
| 2 | -999.0 | 3.0 | 1 | 0 |
| 3 | 4.0 | 4.0 | 0 | 0 |

# 資料轉換 (Data Transformation)

「轉換」的三種動機

正規化：消除輸入異質性（log、Box-Cox）。

闡釋化：將連續量化為語意分區（cut/qcut）。

去偏強化：處理極端值（Winsorize）。

常見錯誤：

闡釋化錯誤區間：固定寬度 cut 可能導致資訊斷層；建議用 qcut 先看分布再微調。

解法：

建議先用 qcut()，根據資料的實際分布切成「每組人數接近」的區間，接著再微調邊界，就能保留資料的結構，又能有語意清楚的分群。

# 正規化：讓數值在同一個起跑線上

資料裡有些欄位數字很大（如年收入），有些很小（如利率），如果直接拿來做統計或機器學習，會變得不公平。

所以我們會透過像是「取對數（log）」或「Box-Cox轉換」這種方法，
把不同量級的數值拉到同樣尺度下，這樣資料才有「公平的發聲權」。

| | income |
|---|---|
| 0 | 30000 |
| 1 | 50000 |
| 2 | 100000 |
| 3 | 300000 |
| 4 | 1000000 |

```
# 取對數後 log以自然數e為底
df['log_income'] = np.log(df['income'])
```

| | income | log_income |
|---|---|---|
| 0 | 30000 | 10.308953 |
| 1 | 50000 | 10.819778 |
| 2 | 100000 | 11.512925 |
| 3 | 300000 | 12.611538 |
| 4 | 1000000 | 13.815511 |

# 闡釋化：把數字變成有意義的分類

闡釋化：把數字變成有**意義的分類**
不是所有的數值都要當連續數字用。
像年齡 18、29、32，雖然是連續，但在分析中我們

更在意的是：
「他是年輕人？中年？還是高齡？」
所以我們會用 cut() 或 qcut() 把這些連續數據分成有意義的區間，
讓模型或人類都能更容易解釋資料的意涵。

| | age |
|---|---|
| 0 | 18 |
| 1 | 25 |
| 2 | 35 |
| 3 | 45 |
| 4 | 60 |

```
# 分箱
df['age_band'] = pd.qcut(df['age'], q=3, labels=['young', 'middle', 'old'])
```

| | age | age_band |
|---|---|---|
| 0 | 18 | young |
| 1 | 25 | young |
| 2 | 35 | middle |
| 3 | 45 | old |
| 4 | 60 | old |

# 延伸型別 (Extension Dtypes)

為什麼 pandas 要引入延伸型別？用 Int64 替代 float64？

原本是整數 1, 2, 3，因為 None 存在，被強制轉成 float64
對「年齡、次數、排名」這類資料失去語意準確性
NaN 只能用 float 表示，導致型別升級 + 運算誤差風險

```
s1 = pd.Series([1, 2, 3, None])                    # 傳統 float64
s2 = pd.Series([1, 2, 3, None], dtype="Int64")     # 延伸型 Int64
```

保持原本的「整數」語意不變
None ➡ <NA>：統一的缺失值表示（非 NaN）
支援 .isna()、.sum()

| | 123 <unnamed> |
|---|---|
| 0 | 1.0 |
| 1 | 2.0 |
| 2 | 3.0 |
| 3 | NaN |

| | 123 <unnamed> |
|---|---|
| 0 | 1 |
| 1 | 2 |
| 2 | 3 |
| 3 | <NA> |

延伸型別不是多個型別，而是讓「有缺值的整數欄位」變得自然、合理、不中斷！

# 字串操作 (String Manipulation)

「向量化字串」的威力

相比逐行 for/regex(字串處理正規化)，pandas .str 背後是 Cython (非 C Python)+ re2，在百萬級資料可快 30 ~ 50×。

關鍵範例：多階段解析 E-mail

| | email |
|---|---|
| 0 | alice@gmail.com |
| 1 | bob@yahoo.com |
| 2 | carol@ntu.edu.tw |

利用 命名捕獲組 (?P<name>) 直接生成欄位 user/domain。

對缺失值 <NA> 自動傳播，免去特別判段。

```
email_pat = r"(?P<user>[^@]+)@(?P<domain>[^\.]+)"
parts = df['email'].str.extract(email_pat, expand=True)
df = pd.concat([df, parts], axis=1)
```

三步驟清洗框架

標準化：大小寫、去空白、unicode normalize。

結構拆解：正則 extract → 多欄。

語意比對：groupby('domain').size() 找出主流與異常值。

| | email | user | domain |
|---|---|---|---|
| 0 | alice@gmail.com | alice | gmail |
| 1 | bob@yahoo.com | bob | yahoo |
| 2 | carol@ntu.edu.tw | carol | ntu |

# 類別資料 (Categorical Data)

效能與語意雙贏
像性別（ Male / Female ）、地區（ North / South ）、客戶類型（ A / B / C ）這種
只會在幾個選項中出現的欄位，我們就叫它「類別欄位」。
這種groupby分析會變得更快，常常可以快上 5 到 10 倍以上！

像是寫成 Parquet 或壓縮儲存時，
類別型別會用「編號 + 對照表」的方式壓縮，
能讓檔案縮小 超過 70%！

```
N = 5_000_000
s_obj = pd.Series(np.random.choice(['foo', 'bar', 'baz'], N))  # object array
s_cat = s_obj.astype('category')  # category dtype

print(s_obj.memory_usage(deep=True)/1e6)  # Object型記憶體用量
print(s_cat.memory_usage(deep=True)/1e6)  # Category型記憶體用量

[1]
```

```
260.000132
5.000396
```

one-hot encoding 進階優化
對高基數欄位進行頻次截斷：先 value_counts()，只保留 top N，其餘編為 __OTHER__，減少維度爆炸。
透過 sparse=True 生成稀疏矩陣 → 直接餵 scikit-learn/LightGBM。

# 感謝聆聽