



# 大數據資料分析實作



## 預處理資料與標準化

博雅(科技)課程

# 預處理資料

- 建立模型的第一步就是**確認資料**。
- 確認完訓練資料，再將資料送入模型前，下一個步驟需先進行**預處理**。
- **預處理**是實務上**最花時間**，且**最重要**的步驟程序，否則「Garbage in, Garbage out」，因此需多加與耐心學習。
- 本次練習一樣使用「鐵達尼號資料集」，因為此資料集有許多種資料**型態**的欄位，非常適合來講解資料的預處理。



# 實作練習：鐵達尼號資料集

欄位名稱	代表意思	說明
survival	生還	0：死亡；1：存活
pclass	艙等	1：一等艙/2：二等艙/3：三等艙
sex	性別	male：男性/female：女性
age	年齡	
sibsp	手足與配偶數	同乘的兄弟姊妹與配偶數
parch	父母與子女數	同乘的父母與子女數
fare	票價	
embarked	乘船港代碼	C：Cherbourg/Q：Queenstown/S：Southampton
class	艙等名	First：一等艙/Second：二等艙/Third：三等艙
who	男女兒童	man：男性/woman：女性/child：兒童
adult_male	成人男子	True/False
deck	甲板	房艙號碼首字母（A到G）
embark_town	乘船港	Southampton/Cherbourg/Queenstown
alive	生還與否	yes/no
alone	單身	True/False

# 預處理資料 - (1)刪除多餘資料欄位

- 資料集中有些欄位名稱雖不同，但意義卻相同，因此可檢視欄位資料，並將多餘欄位進行刪除。
- 在進行數據分析與建立模型前，最好每個欄位都具有其**獨特性**，若有某幾個欄位彼此相關，即會有多餘欄位，此狀況稱為「**多元共線性**」，若將這些欄位資料送入模型，則會造成正確性下降。
- 鐵達尼號資料集中，「艙等」與「艙等名」、「乘船港代碼」與「乘船港」、「生還」與「生還與否」這3組資料欄位意義相同，只需保留1個欄位即可。

# 預處理資料 - (1)刪除多餘資料欄位

- 保留欄位的標準：

1. 具有**順序關係**的欄位：因為順序可用數值表示，應優先保留數值欄位。
2. **只有兩種值**的欄位：若欄位的值只有0、1兩種，則優先保留該欄位。  
因為保留「生還與否」欄位，而後仍需轉換成0、1才能送入模型中。
3. 以**實作方便性和易理解**來選擇欄位：以「乘船港代碼」與「乘船港」為例，二選一皆可，但為方便將欄位做數值編碼，建議選擇欄位值較短的「乘船港代碼」欄位。

# 預處理資料 - (2)處理缺失值

- 預處理資料工作在刪除多餘欄位後，接著要確認各欄位缺失值情況。

而針對**不同缺失值情況**，分別有不同的處理方式：

1. 缺失值很少的欄位，可**逐列刪除**。
2. 缺失值是數值時，可用**補值**的方式。
3. 缺失值的欄位只有固定幾類，可用**虛擬碼補值**。





# 預處理資料 - (2)處理缺失值

- 針對不同缺失值情況，總結三種策略處理方式：
  1. 乘船港代碼：資料型態是**字串**。缺失值**很少**，只有2筆。→**逐列刪除**
  2. 年齡：資料型態為**數值**(float64)。缺失值**很多**，有177筆。→以資料的**平均值代替**
  3. 甲板：資料型態為**類別**(category)。缺失值**很多**，有688筆。→利用額外的**虛擬碼取代**缺失值



# 預處理資料 - (2)處理缺失值

- 處理缺失值的程式方式：

1. **逐列刪除**：刪除含有缺失值的一整列資料，使用**dropna** ( drop NA or NaN ) 函式
2. 以資料的**平均值代替**：填入缺失值，使用**fillna** ( fill NA or NaN ) 函式
3. 利用額外的**虛擬碼取代**缺失值：用虛擬碼“N”取代缺失值，使用**replace**函式

※處理「甲板」欄位缺失值不使用**fillna**函式，是因為「甲板」資料型態是**Category**，而不是字串，因此不能直接填入“N”



# 預處理資料 - (3)處理二元資料數值化

- 當欄位資料只有2種類別，即為**二元資料**。
- 鐵達尼號資料集中，二元資料欄位分別有「**性別**」(male / female)、  
「**成年男子**」(True / False)、  
「**單身**」(True / False)。
- 進行欄位內容**數值化轉換**，可利用資料框的**map**函式

male	→ 1	;	True	→ 1
female	→ 0	;	False	→ 0



# 預處理資料 - (4)處理多元資料數值化

- **多元資料**數值化一般會採用「**One-Hot 編碼**」。主要是根據欄位值有幾種來進行編碼。
- 若將多元資料數值化，採直覺依欄位資料中各分類進行數值分配（例如0、1、2），則將不利於模型計算。

【舉例：假設要將動物欄位中「老鼠」、「長頸鹿」、「大象」這3個資料數值化，則將會因**思考所設定的標準不同**，而有不同的排列順序。若以“體重”為標準，則順序為老鼠=0、長頸鹿=1、大象=2；若以“身高”為標準，則順序為老鼠=0、大象=1、長頸鹿=2】



# 預處理資料 - (4)處理多元資料數值化

- 「**One-Hot 編碼**」主要是根據欄位值有幾種來進行編碼。
- 鐵達尼號資料集中，有「男女兒童」、「乘船港代碼」、「甲板」這三個欄位都有多元資料。
- 以「男女兒童」為例，其欄位有child、man、woman這3種欄位值，因此可訂出一個3維編碼，讓每一種欄位值只會對應到其中1維(其值設為1)，其他維的值則設為0，此為One-Hot 編碼。



# 預處理資料 - (4)處理多元資料數值化

男女兒童	男女兒童_child	男女兒童_man	男女兒童_woman
man	0	1	0
woman	0	0	1
woman	0	0	1
woman	0	0	1
man	0	1	0
child	1	0	0
man	0	1	0

- 注意：「**One-Hot 編碼**」的**維度很高**時（例如有1000維），則資料中會出現大量的0（即1000維的資料，只會有1000個1，但會有 $1000 \times 999$ 個0），將這樣的資料送入模型中訓練，會造成**無謂的運算成本**。

# 預處理資料 - (4)處理多元資料數值化

- 「**One-Hot 編碼**」可應用資料框的“**get\_dummies**”函式。
- `get_dummies` 函式可指定要將哪個欄位進行 One-Hot 編碼，並透過 **prefix** 參數來將**指定的字串**新增到自動生成的 One-Hot **欄位名稱開頭**。
- 以「男女兒童」為例，應用`get_dummies` 函式與其**prefix**參數，會將原本「男女兒童」欄位，經編碼後產生出3個新欄位，分別為「男女兒童\_child」、「男女兒童\_man」、「男女兒童\_woman」。

# 預處理資料 - (5)資料標準化

- 資料中若有離群值，或因量測單位不同而產生數值範圍差異較大，將會造成模型訓練問題。
- 解決離群值或數值範圍差異大的問題，一般有兩種方式：
  - ① **Min-max正規化**(Normalization)：將數值範圍縮放到0~1之間。
  - ② **Z值標準化**(Standardization)：將數值範圍轉換成常態分布。
- 【補充說明】資料標準化對線性迴歸、邏輯斯迴歸、支持向量機等演算法有效用，但對於決策樹型演算法就沒有必要性。



# 預處理資料 - (5)資料標準化

- **Min-max正規化** ( Normalization ) : 其轉換數值方式是先找出數值範圍的**最大值**和**最小值**，然後再將每個數值**依比例縮放**到 **0~1** 之間。

轉換公式為：

$$\hat{x} = \frac{x - x_{min}}{x_{max} - x_{min}}$$





# 預處理資料 - (5)資料標準化

- **Z值標準化** ( Standization ) : 主要是將資料的**平均數移動到 0** 的位置。  
若資料為常態分布，將資料經 Z 值標準化轉換到「**平均值為 0**，**標準差為 1**」的**標準常態分布**。

轉換公式為：

$$\hat{x} = \frac{x - m}{\sigma}$$

# 預處理資料 - (5)資料標準化

## 【補充說明】

- Min-max 正規化容易受離群值影響。因此針對資料標準化處理選擇：
- 若資料中可能**含有較大離群值** → 選擇“**Z值標準化**”
- 若資料事先**已知最大值與最小值** → 選擇“**Min-max 正規化**”



# 課堂實作範例操作流程



# 實作步驟：安裝與設定

- 此行程式碼會**安裝** **japanize-matplotlib** **套件**，用來在 Matplotlib 圖表中顯示日文字符的套件。
- `tail -n 1` 則會只顯示安裝過程的最後一行。

```
!pip install japanize-matplotlib | tail -n 1
```



# 實作步驟：安裝與設定

- 這兩行程式碼是為了**關閉警告訊息**。
  - ✓ **import warnings**：匯入 Python 的警告管理模組。
  - ✓ **warnings.filterwarnings('ignore')**：告訴 Python 忽略所有警告訊息。這在進行資料處理和繪圖時，常見警告如 deprecation（過時警告）或數據不一致警告會被隱藏。

```
import warnings  
warnings.filterwarnings('ignore')
```



# 實作步驟：載入所需的 Python 模組

- pandas：用於**資料處理和操作**的資料框架。
- numpy：用於**數學運算**，特別是數組處理。
- matplotlib.pyplot：用於**繪製圖表**。
- japanize\_matplotlib：用於在 Matplotlib 圖表中顯示日文字符。
- IPython.display：用於在 Jupyter Notebook 中顯示物件。

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import japanize_matplotlib
from IPython.display import display
```



# 實作步驟：設定numpy與pandas顯示選項

- 設定 **numpy** 顯示數值時的格式：

- ✓ `suppress=True`：避免顯示過多的數字（例如浮點數很小的數字）。
- ✓ `precision=4`：將數字顯示為四位小數。

```
np.set_printoptions(suppress=True, precision=4)
```

- `pd.options.display.float_format = '{:.4f}'.format`：設定 **pandas** 顯示浮點數的格式為小數點後 4 位，這樣在 DataFrame 中顯示的浮點數會更整潔。

```
pd.options.display.float_format = '{:.4f}'.format
```



# 實作步驟：設定pandas顯示選項

- `pd.set_option("display.max_columns", None)`：告訴 pandas 在顯示 DataFrame 時不限制列數，這樣可以顯示所有欄位。

```
pd.set_option("display.max_columns", None)
```

- `plt.rcParams["font.size"] = 14`：設定 Matplotlib 圖表中的字體大小為 14，這使得圖表中的文字更大更易讀。

```
plt.rcParams["font.size"] = 14
```



# 實作步驟：資料讀取與初步處理

- `import seaborn as sns`：匯入 **seaborn** 庫，這是一個建立在 `matplotlib` 基礎上的視覺化庫，並且提供了一些便捷的資料集。
- `df_titanic = sns.load_dataset("titanic")`：載入 **seaborn** 提供的 **Titanic 數據集**，這個數據集包含了Titanic乘客的資訊，如是否生還、艙等、性別、年齡等。

```
import seaborn as sns  
df_titanic = sns.load_dataset("titanic")
```



# 實作步驟：更改資料集的欄位名稱

```
columns_t = ['生還', '艙等', '性別', '年齡', '手足與配偶數',  
             '父母與子女數', '票價', '乘船港代碼', '艙等名', '男女兒童',  
             '成人男子', '甲板', '乘船港', '生還與否', '單身']  
  
df_titanic.columns = columns_t  
print(df_titanic.head())
```

- `columns_t = [...]`：這行**定義了一個新的欄位名稱列表**，將原本的英文欄位名稱翻譯為中文，讓資料更容易理解。
- `df_titanic.columns = columns_t`：將 `df_titanic` 的欄位名稱更改為上述定義的中文名稱。

# 實作步驟：了解資料的結構與大小

- `display(df_titanic.head())`：顯示 `df_titanic DataFrame` 的前五行，這可以幫助我們快速了解資料的結構。
- `print(df_titanic.shape)`：顯示 `df_titanic DataFrame` 的形狀（行數和列數），用於了解資料的大小。

```
display(df_titanic.head())  
print(df_titanic.shape)
```



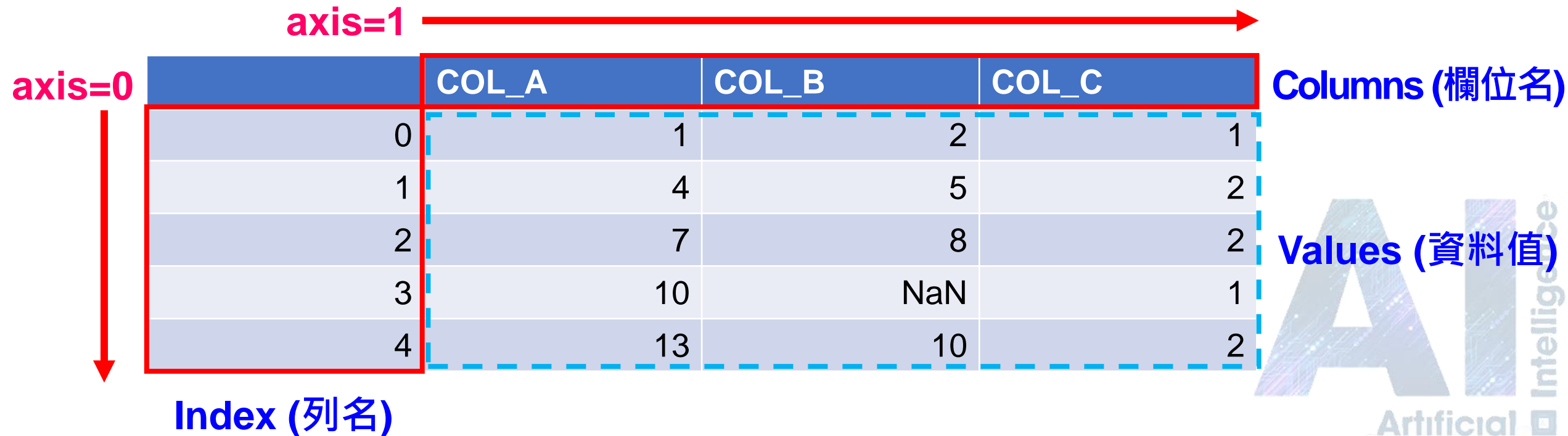
# 實作步驟：刪除多餘資料欄位

- 鐵達尼號資料集中，「艙等」與「艙等名」、「乘船港代碼」與「乘船港」、「生還」與「生還與否」這3組資料欄位意義相同，只需保留1個欄位即可。
- `df1 = df_titanic.drop('艙等名', axis=1)`：這行程式碼刪除了 '艙等名' 這個欄位。`axis=1` 表示刪除欄，若 `axis=0` 則是刪除行。

```
df1 = df_titanic.drop('艙等名', axis=1)
```

# 實作步驟：【補充說明】資料處理

- DataFrame與NumPy都可對表格資料進行處理，若是垂直操作（預設），可以指定 **axis=0**，若為水平操作，則指定 **axis=1**。



The diagram illustrates a DataFrame with its axes and components. A red arrow labeled 'axis=1' points horizontally across the top of the table. A red arrow labeled 'axis=0' points vertically down the left side of the table. The table has a blue header row with columns 'COL\_A', 'COL\_B', and 'COL\_C'. The first column contains index values 0, 1, 2, 3, 4. The data cells are highlighted with a dashed blue border. To the right of the table, the text 'Columns (欄位名)' is next to the header, and 'Values (資料值)' is next to the data rows. A large 'AI Intelligence' watermark is visible in the background.

	COL_A	COL_B	COL_C
0	1	2	1
1	4	5	2
2	7	8	2
3	10	NaN	1
4	13	10	2

Columns (欄位名)

Values (資料值)

Index (列名)

# 實作步驟：刪除多餘資料欄位

- 刪除了 '乘船港' 欄位。

```
df2 = df1.drop('乘船港', axis=1)
```

- 刪除 '生還與否' 欄位，因為這個欄位在後續處理中已經不需要。

```
df3 = df2.drop('生還與否', axis=1)
```

- 顯示處理過後的 df3 的前五行，**檢查刪除欄位後的資料狀態**。

```
display(df3.head())
```





# 實作步驟：處理缺失值

- `df3.isnull().sum()`：這會顯示每個欄位中缺失值的數量。 `isnull()` 返回一個布林值的 `DataFrame`，`sum()` 計算每列中的 `True`（即缺失值）個數。

```
display(df3.isnull().sum())
```

- `df3['甲板'].value_counts()`：顯示‘甲板’欄位的值計數。這用來檢查是否有缺失值，或哪些值的頻次較高。查看甲板的資料分佈。

```
display(df3['甲板'].value_counts())
```

# 實作步驟：處理缺失值

- `df3.dropna(subset=['乘船港代碼'])`：刪除 ['乘船港代碼'] 欄位中存在缺失值的行。這保證在處理後的數據中，這個欄位不會有任何缺失值。

```
df4 = df3.dropna(subset=['乘船港代碼'])
```



# 實作步驟：處理缺失值

- `df4['年齡'].mean()`：計算 '年齡' 欄位的**平均值**。
- `df5 = df4.fillna({'年齡': age_average})`：使用 '年齡'欄位的平均值**填充所有缺失**的 '年齡'值。

```
age_average = df4['年齡'].mean()  
df5 = df4.fillna({'年齡': age_average})
```



# 實作步驟：處理缺失值

```
df6 = df5.replace({'甲板': {np.nan: 'N'}})
```

- df5：原本的 DataFrame。
- replace**：是 Pandas 提供的一個方法，用來**取代** DataFrame 中的某些值。
- {'甲板': {np.nan: 'N'}}：指定要在欄位 '甲板' 中，**將 NaN（缺失值）**  
**取代為 'N'。**
- df6 =：把處理後的結果存成新的 DataFrame，命名為 df6。

# 實作步驟：【補充說明】用fillna函式填入字串

- df5['甲板'].**astype**(object)：將 '甲板' 欄位的數據類型轉換為 **object 類型**（即字符型），因為它是類別數據。
- df6 = df5.**fillna**({'甲板': 'N'})：用 'N' 填充 '甲板' 欄位的**所有缺失值**，這可能表示缺失或未知的艙位。

```
df5['甲板'] = df5['甲板'].astype(object)
df6 = df5.fillna({'甲板': 'N'})
```

# 實作步驟：處理缺失值

- 顯示 df6 中每個欄位的缺失值數量，**檢查是否還有缺失值**。

```
display(df6.isnull().sum())
```

- 顯示 df6 的前五行，**檢查處理後的資料**。

```
display(df6.head())
```



# 實作步驟：性別轉換

- 顯示 df6 中 '性別' 欄位的值計數，[查看男性與女性的數量分佈](#)。

```
display(df6['性別'].value_counts())
```





# 實作步驟：性別轉換

- `mf_map = {'male': 1, 'female': 0}`：建立一個映射字典，將 'male' 轉換為 1，'female' 轉換為 0。
- `df7['性別'] = df7['性別'].map(mf_map)`：將 '性別' 欄位的 male 和 female 轉換為 1 和 0。

```
mf_map = {'male': 1, 'female': 0}
df7 = df6.copy()
df7['性別'] = df7['性別'].map(mf_map)
display(df7.head())
```



# 實作步驟：布林值轉換

- 應用**map**函式將“成年男子”、“單身”欄位中的「**True**」轉換為 **1**，「**False**」轉換為 **0**。

```
display(df7['成人男子'].value_counts())  
tf_map = {True: 1, False: 0}  
df8 = df7.copy()  
df8['成人男子'] = df8['成人男子'].map(tf_map)  
df9 = df8.copy()  
df9['單身'] = df8['單身'].map(tf_map)  
display(df9.head())
```

# 實作步驟：布林值轉換

- 顯示‘成人男子’欄位中不同布林值（True 和 False）的數量。

```
display(df7['成人男子'].value_counts())
```

- tf\_map** = {True: 1, False: 0}：建立布林值映射字典，將True轉換為 1，False 轉換為 0。

- df8['成人男子'] = df8['成人男子'].map(**tf\_map**)：將‘成人男子’欄位的布林值轉換為 1 和 0。

```
tf_map = {True: 1, False: 0}
df8 = df7.copy()
df8['成人男子'] = df8['成人男子'].map(tf_map)
```



# 實作步驟：布林值轉換

- 將‘單身’欄位中的布林值轉換為 1 和 0。

```
df9 = df8.copy()  
df9['單身'] = df8['單身'].map(tf_map)
```

- 顯示更新後的 df9 前五行。

```
display(df9.head())
```



# 實作步驟：處理類別欄位

- 應用 **get\_dummies** 函式 與 **prefix** 參數，會將原本「男女兒童」欄位，經編碼後產生出3個新欄位，分別為「男女兒童\_child」、「男女兒童\_man」、「男女兒童\_woman」。
- 顯示「男女兒童」欄位的前 10 行。

```
display(df9[['男女兒童']].head(10))
```



# 實作步驟：處理類別欄位

- `pd.get_dummies(df9['男女兒童'], prefix='男女兒童')`：使用 pandas 的 `get_dummies()` 方法將 '男女兒童' 欄位進行 one-hot 編碼，將其轉換為數值型欄位。
- `prefix='男女兒童'`：指定新欄位的前綴名。
- `display(w.head(10))`：顯示編碼後的前 10 行。

```
w = pd.get_dummies(df9['男女兒童'], prefix='男女兒童')  
display(w.head(10))
```

# 實作步驟：處理類別欄位

- 定義 **enc 函數**，這個函數對指定的類別欄位進行 **one-hot 編碼**：
  - `df[column]` 取出指定的欄位。
  - `pd.get_dummies()` 創建 one-hot 編碼。
  - `df.drop([column], axis=1)` 刪除原始的類別欄位。
  - `pd.concat([df_drop, df_dummy], axis=1)` 將編碼後的欄位與原來的 DataFrame 合併。

```
def enc(df, column):  
    df_dummy = pd.get_dummies(df[column], prefix=column)  
    df_drop = df.drop([column], axis=1)  
    df1 = pd.concat([df_drop, df_dummy], axis=1)  
    return df1
```

# 實作步驟：處理類別欄位

- 使用 `enc` 函式對 '男女兒童' 欄位進行 **one-hot 編碼**，並顯示處理後的 DataFrame。

```
df10 = enc(df9, '男女兒童')  
display(df10.head())
```

- 對 '乘船港代碼' 和 '甲板' 欄位進行 **one-hot 編碼**，並顯示結果。

```
df11 = enc(df10, '乘船港代碼')  
df12 = enc(df11, '甲板')  
display(df12.head())
```





# 實作步驟：數據標準化

- **StandardScaler** 用於將數據標準化，使其具有均值為 0，標準差為 1。  
這對於某些機器學習演算法（如 **SVM**、**KNN**）非常有用。
- 使用 **StandardScaler** 對 '年齡' 和 '票價' 欄位進行標準化，使這些欄位的值的均值為 0，標準差為 1。

```
df13 = df12.copy()
```

```
from sklearn.preprocessing import StandardScaler
```

# 實作步驟：數據標準化

- **fit\_transform()** 用於擬合數據並轉換其範圍。

```
stdsc = StandardScaler()  
df13[['年齡', '票價']] = stdsc.fit_transform(df13[['年齡', '票價']])
```

- 顯示標準化後的資料。

```
display(df13.head())
```

- 這樣就完成了資料的預處理與清理工作。



# 「大數據資料分析實作」課程

