

# AI



# 深度學習

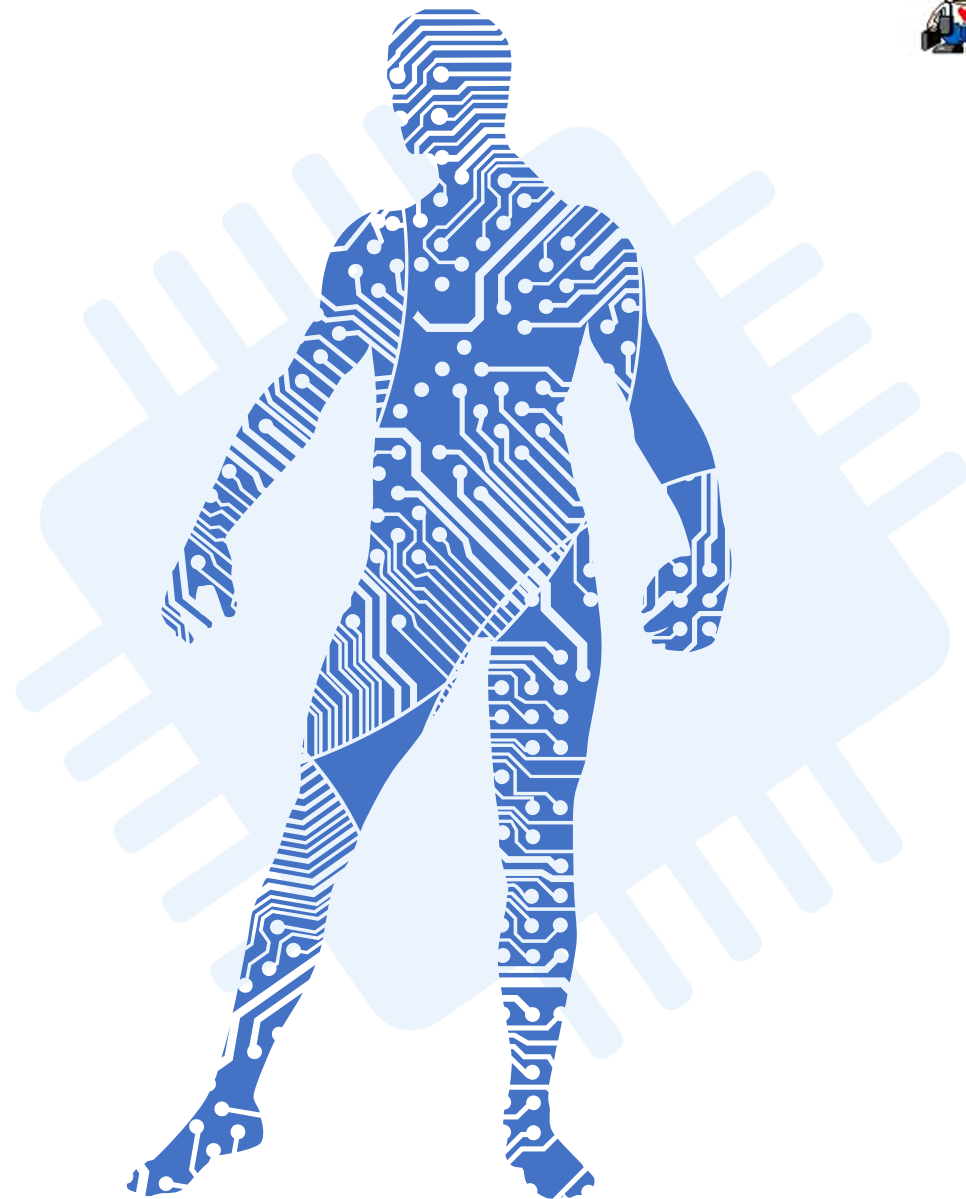
## 第 4 章 資料前處理

講師：紀俊男



# 本章大綱

- 資料前處理簡介
- 載入資料集
- 切分自變數、應變數
- 處理缺失資料
- 類別資料數位化
- 切分訓練集、測試集
- 特徵縮放





# AI

## 資料前處理簡介

# 何謂資料前處理 ( Data Pre-processing ) ?



- 將資料處理到可以代入人工智慧模型的步驟

做菜



前處理



人工智慧

- ✓ 讀入資料
- ✓ 資料補遺
- ✓ 格式統一
- ✓ ...

前處理





# 資料前處理要做哪些事？



- 載入資料集
- 切分自變數、應變數
- 處理缺失資料
- 類別資料數位化
- 切分訓練集、測試集
- 特徵縮放



標準版

快樂版



## 載入資料集

標準版完整原始碼：  
<https://ishort.ink/Gwrr>

快樂版完整原始碼：  
<https://ishort.ink/PQEy>



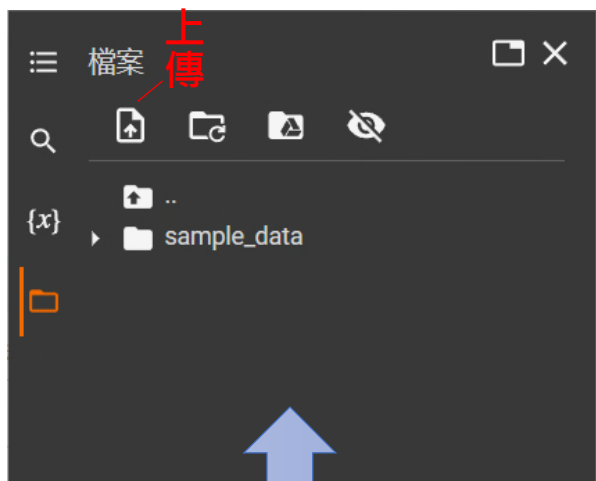
# 備妥資料集給 Colab



- 下載後上傳給 Colab

- 要求 Colab 直接從源頭下載

- 使用 **wget** 指令 + 下載之網址



CarEvaluation.csv

```
1 # 不檢查檔案是否存在，暴力下載
2 # ! wget https://raw.githubusercontent.com/cnchi/datasets/master/CarEvaluation.csv
3
4 # 先檢查檔案是否存在，再決定是否下載
5 import os
6
7 Dataset_File = "CarEvaluation.csv"
8 if not os.path.isfile(Dataset_File):
9     os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)
```

好處：按下「**執行**」→ 自動完成，不必人為介入。

- 請依照講師指示，下載 **CarEvaluation.csv**，放置在您的**本機磁碟**。
- 將 **CarEvaluation.csv** 上載至 Colab，雙擊看看，能否看到檔案的內容。
- 將 **CarEvaluation.csv** 刪除，開啟一個新的 Colab 頁面，假設叫做 **Preprocessing.ipynb** 並撰寫好下列程式碼。執行之後，看能否在 Colab 看到**資料檔**已經下載？
- 雙擊**資料檔**，看看能否觀察到檔案的內容？
- 透過此練習，希望您能掌握如何在 Colab 內，將**資料檔**備妥的技巧。

```
1 import os
2
3 Dataset_File = "CarEvaluation.csv"
4 if not os.path.isfile(Dataset_File):
5     os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)
```





撰寫程式碼

```
1 import numpy as np
2 import pandas as pd
3
4 dataset = pd.read_csv("CarEvaluation.csv")
```



利用草稿儲存格觀看讀入結果

草稿儲存格 ×

✓ 0 秒

1 dataset

	City	Children	Age	Salary	ToBuy
0	Taipei	NaN	44.0	72000.0	No
1	Taichung	0.0	27.0	48000.0	Yes
2	Kaohsiung	0.0	30.0	54000.0	No
3	Taichung	1.0	38.0	61000.0	No
4	Kaohsiung	2.0	40.0	NaN	Yes
5	Taipei	2.0	35.0	58000.0	Yes
6	Taichung	1.0	NaN	52000.0	No
7	Taipei	2.0	48.0	79000.0	Y
8	Kaohsiung	1.0	50.0	83000.0	
9	Taipei	2.0	37.0	67000.0	Yes

「缺失值」變成  
NaN (Not a Number)

- 請撰寫以下的程式碼，儲存到 `Preprocessing.ipynb`，並執行它。
- 插入「草稿程式碼儲存格」，觀察讀取到的 dataset 變數內容。
- 你能注意到所有「缺失值」被標記為 `NaN (Not a Number)` 嗎？

```
1 import numpy as np
2 import pandas as pd
3
4 dataset = pd.read_csv("CarEvaluation.csv")
```



- 何謂「快樂版」函式庫？
  - 講師自製、讓你以更短時間，完成相同工作的函式庫

使用「標準版」函式庫做前處理（28 行）

使用「快樂版」函式庫做前處理（14 行）

```
1 import numpy as np
2 import pandas as pd
3
4 dataset = pd.read_csv("CarEvaluation.csv")
5
6 X = dataset.iloc[:, :-1].values
7 Y = dataset.iloc[:, 4].values
8
9 from sklearn.impute import SimpleImputer
10
11 imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
12 imputer = imputer.fit(X[:, 1:4])
13 X[:, 1:4] = imputer.transform(X[:, 1:4])
14
15 from sklearn.preprocessing import LabelEncoder
16 labelEncoder = LabelEncoder()
17 Y = labelEncoder.fit_transform(Y).astype("float64")
18
19 ary_dummies = pd.get_dummies(X[:, 0]).values
20 X = np.concatenate((ary_dummies, X[:, 1:4]), axis=1).astype("float64")
21
22 from sklearn.model_selection import train_test_split
23 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
24
25 from sklearn.preprocessing import StandardScaler
26 sc_X = StandardScaler().fit(X_train)
27 X_train = sc_X.transform(X_train)
28 X_test = sc_X.transform(X_test)
```

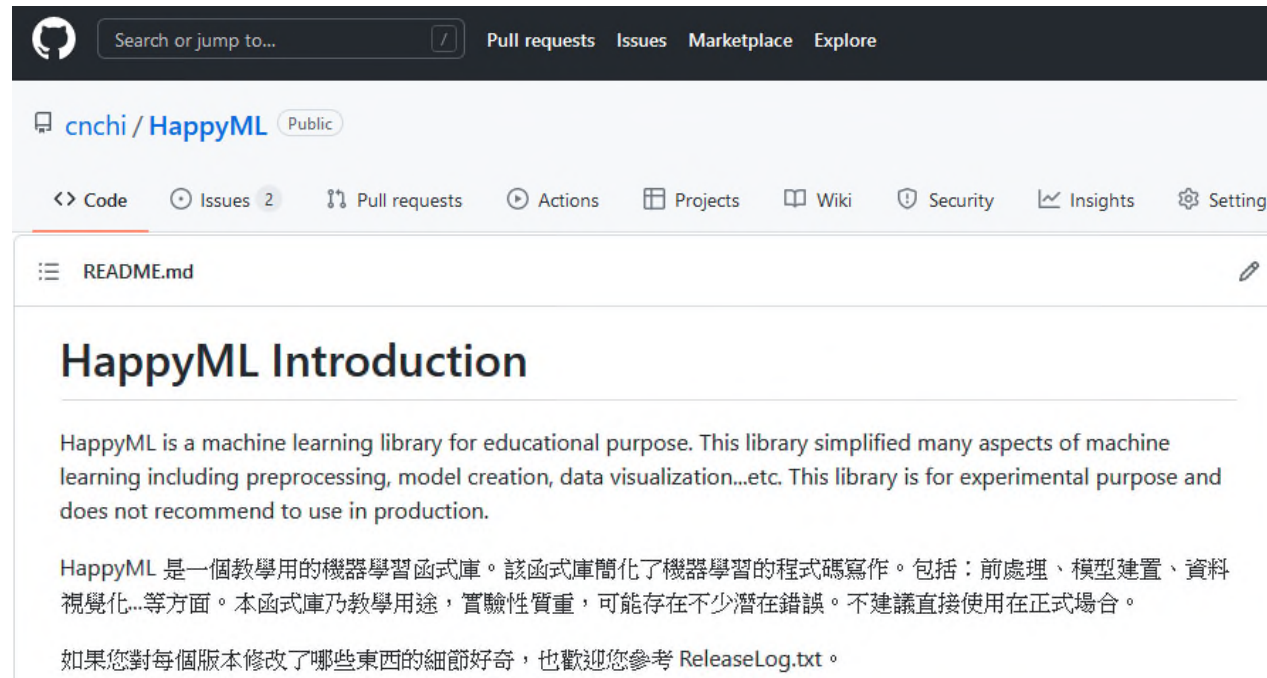
```
1 import HappyML.preprocessor as pp
2
3 dataset = pp.dataset("CarEvaluation.csv")
4
5 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(4)], y_columns=[4])
6
7 X = pp.missing_data(X, strategy="mean")
8
9 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
10 X = pp.onehot_encoder(X, columns=[0])
11
12 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y, train_size=0.8, random_state=0)
13
14 X_train, X_test = pp.feature_scaling(X_train, transform_arys=(X_train, X_test))
```

# A 我可以任意使用「快樂版」函式庫嗎？



- GitHub 公開原始碼。自由修改、自由使用，註明出處

<https://github.com/cnchi/HappyML>





# 安裝「快樂版」函式庫



```
1 # 先檢查是否存在 HappyML 這個資料夾，若沒有，則下載
2 import os
3
4 if not os.path.isdir("HappyML"):
5     os.system("git clone https://github.com/cnchi/HappyML.git")
```

# 隨堂練習：安裝「快樂版」函式庫



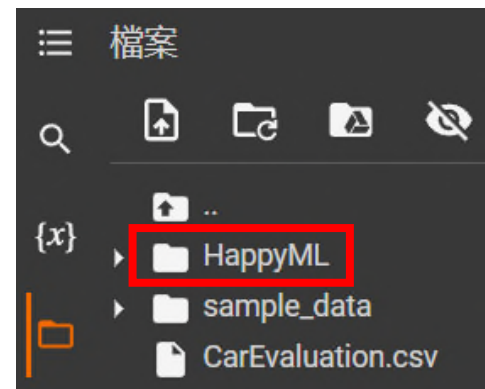
- 請開啟一個新的頁面，假設叫做 `Preprocessing_Happy.ipynb`。
- 將 `Preprocessor.ipynb` 內的「取得資料集」程式碼拷貝過來。
- 將前一頁的原始碼撰寫好。完成的程式碼應該如下所示。
- 執行整個程式碼，並觀察 `HappyML` 資料夾是否已經下載下來了？

## 取得資料集

```
[ ] 1 # 不檢查檔案是否存在，暴力下載
2 # ! wget https://raw.githubusercontent.com/cnchi/datasets/master/CarEvaluation.csv
3
4 # 先檢查檔案是否存在，再決定是否下載
5 import os
6
7 Dataset_File = "CarEvaluation.csv"
8 if not os.path.isfile(Dataset_File):
9     os.system("wget https://raw.githubusercontent.com/cnchi/datasets/master/" + Dataset_File)
```

## 安裝快樂版函式庫

```
[ ] 1 # 先檢查是否存在 HappyML 這個資料夾，若沒有，則下載
2 if not os.path.isdir("HappyML"):
3     os.system("git clone https://github.com/cnchi/HappyML.git")
```





- 「快樂版」原始碼講解

```
15 def dataset(file=""):
16     if file != "":
17         dataset = pd.read_csv(file)
18     else:
19         dataset = None
20
21     return dataset
```

- 「快樂版」的使用方法

```
1 import HappyML.preprocessor as pp
2
3 dataset = pp.dataset(file="CarEvaluation.csv")
```

注意！

- dataset() 函數傳回來的資料型態，是 pandas 的 DataFrame。

- 將 HappyML 裡面的 `preprocessor.py` 打開，瀏覽一下 `dataset()` 函數的原始碼。
- 在 `Preprocessing_Happy.ipynb` 中，撰寫「載入資料集」的程式碼如下，並且執行看看。
- 用「草稿程式碼儲存格」，觀察讀取到的 `dataset` 變數內容。
- 你能注意到所有「缺失值」被標記為 `NaN (Not a Number)` 嗎？

```
1 import HappyML.preprocessor as pp
2
3 dataset = pp.dataset(file="CarEvaluation.csv")
```





切分  
自變數、應變數

# 為何要切分「自變數、應變數」



	A	B	C	D
1	國別	年齡	薪資	是否購買
2	France	44	72000	No
3	Spain	27	48000	Yes
4	Germany	30	54000	No
5	Spain	38	61000	No
6	Germany	40		Yes
7	France	35	58000	Yes
8	Spain		52000	No
9	France	48	79000	Yes
10	Germany	50	83000	No
11	France	37	67000	Yes

自變數

應變數

$$\text{購買} = a \cdot (\text{國別}) + b \cdot (\text{年齡}) + c \cdot (\text{薪資}) \quad a, b, c = \text{權重}$$

# 使用標準函式庫切分「自變數、應變數」



## 原始碼

```
1 X = dataset.iloc[:, :-1].values
2 Y = dataset.iloc[:, 4].values
```



	A	B	C	D	E
1	City	Children	Age	Salary	ToBuy
2	Taipei		44	72000	No
3	Taichung	0	27	48000	Yes
4	Kaohsiung	0	30	54000	No
5	Taichung	1	38	61000	No
6	Kaohsiung	2	40		Yes
7	Taipei	2	35	58000	Yes
8	Taichung	1		52000	No
9	Taipei	2	48	79000	Yes
10	Kaohsiung	1	50	83000	No
11	Taipei	2	37	67000	Yes

自變數 X  
[0:9, 0:3]

應變數 Y  
[0:9, 4:4]

## 注意：.values 的意義

- .values 可以將 DataFrame 中的值取出，變成 NDArray。
- 人工智慧底層函式庫大多只接受傳入 NDArray，而非 DataFrame。

- 請到 `Preprocessing.ipynb` 裡面，撰寫下列程式碼：

```
1 X = dataset.iloc[:, :-1].values
2 Y = dataset.iloc[:, 4].values
```

- 重新執行整個程式。
- 用「草稿程式碼儲存格」，觀察讀取到的自變數 `X` 與應變數 `Y` 的內容：

自變數 `X`

```
array([[ 'Taipei', nan, 44.0, 72000.0],
       [ 'Taichung', 0.0, 27.0, 48000.0],
       [ 'Kaohsiung', 0.0, 30.0, 54000.0],
       [ 'Taichung', 1.0, 38.0, 61000.0],
       [ 'Kaohsiung', 2.0, 40.0, nan],
       [ 'Taipei', 2.0, 35.0, 58000.0],
       [ 'Taichung', 1.0, nan, 52000.0],
       [ 'Taipei', 2.0, 48.0, 79000.0],
       [ 'Kaohsiung', 1.0, 50.0, 83000.0],
       [ 'Taipei', 2.0, 37.0, 67000.0]], dtype=object)
```

應變數 `Y`

```
array([ 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

注意！

- `NDArray` 是沒有「列名」與「欄名」的。





• 原始碼解說

```
26 def decomposition(dataset, x_columns, y_columns=[]):
27     X = dataset.iloc[:, x_columns]
28     Y = dataset.iloc[:, y_columns]
29
30     if len(y_columns) > 0:
31         return X, Y
32     else:
33         return X
```

• 使用方法

```
1 import HappyML.preprocessor as pp
2
3 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(4)], y_columns=[4])
```

- 使用「快樂版」優點：
- 回傳值是 DataFrame，不是 ndarray。
  - DataFrame 能保留列名、欄名，方便理解。
  - 不必記憶過多的引入套件（import packages）名稱。

- 請到 `Preprocessing_Happy.ipynb` 裡面，撰寫下列程式碼：

```
1 import HappyML.preprocessor as pp
2
3 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(4)], y_columns=[4])
```

- 重新執行整個程式。
- 用「草稿程式碼儲存格」，觀察讀取到的自變數 `X` 與應變數 `Y` 的內容。

自變數 X

	City	Children	Age	Salary
0	Taipei	NaN	44.0	72000.0
1	Taichung	0.0	27.0	48000.0
2	Kaohsiung	0.0	30.0	54000.0
3	Taichung	1.0	38.0	61000.0
4	Kaohsiung	2.0	40.0	NaN
5	Taipei	2.0	35.0	58000.0
6	Taichung	1.0	NaN	52000.0
7	Taipei	2.0	48.0	79000.0
8	Kaohsiung	1.0	50.0	83000.0
9	Taipei	2.0	37.0	67000.0

應變數 Y

	ToBuy
0	No
1	Yes
2	No
3	No
4	Yes
5	Yes
6	No
7	Yes
8	No
9	Yes

注意！

- `DataFrame` 的列名與欄名會保留！閱讀容易！





處理缺失資料

• 何謂**缺失資料**

	A	B	C	D	E
1	City	Children	Age	Salary	ToBuy
2	Taipei		44	72000	No
3	Taichung	0	27	48000	Yes
4	Kaohsiung	0	30	54000	No
5	Taichung	1	38	61000	No
6	Kaohsiung	2	40		Yes
7	Taipei	2	35	58000	Yes
8	Taichung	1		52000	No
9	Taipei	2	48	79000	Yes
10	Kaohsiung	1	50	83000	No
11	Taipei	2	37	67000	Yes

• 缺失資料**補遺**辦法

- 取**欄平均**填入
- 取**欄中位數**填入
- 取**欄眾數**填入

dataset.isnull()

	City	Children	Age	Salary	ToBuy
0	False	True	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	False	False	False	False
4	False	False	False	True	False
5	False	False	False	False	False
6	False	False	True	False	False
7	False	False	False	False	False
8	False	False	False	False	False
9	False	False	False	False	False

 DataFrame 格式

dataset.isnull().sum()

City	0
Children	1
Age	1
Salary	1
ToBuy	0
dtype:	int64

Series 格式

dataset.isnull().any()

City	False
Children	True
Age	True
Salary	True
ToBuy	False
dtype:	bool

Series 格式

sum(dataset.isnull().sum())

3



只要 isnull 個數總和 > 0  
→ 就需要做「缺失資料補遺」



- 請先任選 `Preprocessing.ipynb` 或 `Preprocessing_Happy.ipynb` 任一檔案。
- 插入「草稿程式碼儲存格」。並試用下列指令，來檢查資料集有無缺失資料：
  - `dataset.isnull()`
  - `dataset.isnull().sum()`
  - `dataset.isnull().any()`
  - `sum(dataset.isnull().sum())`





# 缺失資料填補法



```
1 from sklearn.impute import SimpleImputer
2
3 imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
4 imputer = imputer.fit(X[:, 1:4])
5 X[:, 1:4] = imputer.transform(X[:, 1:4])
```

1. `impute` = 責難、推算

## 2. `SimpleImputer`

- `missing_values = np.nan`  
所有標示為 NaN 的欄位
- `strategy =`
  - `"mean"` : 欄平均
  - `"median"` : 欄中位數
  - `"most_frequent"` : 眾數
  - `"constant"` : 特定常數
- `fill_value = ( 某數字 )`
  - 當 `strategy = "constant"` 時才有用
  - 會把 `fill_value` 後面的數字，填入所有缺失資料中

## 3. `.fit`(陣列切片)

- 根據「陣列切片」出來的結果，去計算：
  - 「平均」 ( `strategy= "mean"` )
  - 「中位數」 ( `strategy= "median"` )
  - 「眾數」 ( `strategy= "most_frequent"` )
  - 「特定常數」 ( `strategy= "constant"` )

## 4. `.transform`(陣列切片)

- 將陣列切片內的「缺失資料」，轉化為計算出來的值。

### 注意：

- `sklearn.preprocessing.Imputer` 已經建議停用 ( **Deprecated** )



# 隨堂練習：填補缺失資料

- 請先於 `Preprocessing.ipynb` 內，引入 `SimpleImputer` 套件
  - `from sklearn.impute import SimpleImputer`
- 指定**缺失值**填入策略
  - 使用**欄平均值**，作為填入所有缺失資料的替代值
  - `imputer = SimpleImputer(missing_values=np.nan, strategy="mean")`
- 針對有缺失的「**欄1**、**欄2**、**欄3**」，計算各欄平均值
  - `imputer = imputer.fit(X[:, 1:4])`
- 將所有缺失資料，轉化為各欄平均值
  - `X[:, 1:4] = imputer.transform(X[:, 1:4])`
- 請用「**草稿程式碼儲存格**」，觀看程式執行過後，自變數 `X` 的缺失資料，是否已經補齊了？
- 參考程式碼如下所示：

```
1 from sklearn.impute import SimpleImputer
2
3 imputer = SimpleImputer(missing_values=np.nan, strategy="mean")
4 imputer = imputer.fit(X[:, 1:4])
5 X[:, 1:4] = imputer.transform(X[:, 1:4])
```



• 原始碼解說

```
39 from sklearn.impute import SimpleImputer
40
41 def missing_data(dataset, strategy="mean"):
42     if strategy not in ("mean", "median", "most_frequent"):
43         strategy = "mean"
44
45     if (type(dataset) is pd.DataFrame) and (sum(dataset.isnull().sum()) > 0):
46         ary = dataset.values
47         missing_cols = [i for i, j in enumerate(dataset.isnull().any()) if j]
48         imputer = SimpleImputer(missing_values=np.nan, strategy=strategy)
49         imputer = imputer.fit(ary[:, missing_cols])
50         ary[:, missing_cols] = imputer.transform(ary[:, missing_cols])
51         return pd.DataFrame(ary, index=dataset.index, columns=dataset.columns)
52     else:
53         return dataset
```

• 使用方法

```
1 import HappyML.preprocessor as pp
2
3 X = pp.missing_data(X, strategy="mean")
```

使用「快樂版」的好處：

- 傳回值為 DataFrame，能保持欄位名稱。
- 不用管有無缺失資料，或缺失資料是哪幾欄。直接丟入函數中，叫用就對了！



- 請於 **Preprocessing\_Happy.ipynb** 之下，撰寫下列程式碼，並執行看看：

```
1 import HappyML.preprocessor as pp
2
3 X = pp.missing_data(X, strategy="mean")
```

- 重新**執行**整個程式。
- 到**草稿程式碼儲存格**，檢查一下，所有缺失資料是否已經補齊了？





# AI

## 類別資料數位化

• 原始資料

City	Children	Age	Salary	ToBuy
Taipei	nan	44	72000	No
Taichung	0	27	48000	Yes
Kaohsiung	0	30	54000	No
Taichung	1	38	61000	No
Kaohsiung	2	40	nan	Yes
Taipei	2	35	58000	Yes
Taichung	1	nan	52000	No
Taipei	2	48	79000	Yes
Kaohsiung	1	50	83000	No
Taipei	2	37	67000	Yes

「文字」無法作為「方程式輸入值」

• 數位化資料

0
0
1
0
0
1
1
0
1
0
1

「數字」才能作為「方程式輸入值」



• 法一：使用「**標籤編碼器 (LabelEncoder)**」

```
1 from sklearn.preprocessing import LabelEncoder
2
3 labelEncoder = LabelEncoder()
4 Y = labelEncoder.fit_transform(Y).astype("float64")
```

3

4

1. 引入 **LabelEncoder**
2. 由 **LabelEncoder** 類別，產生一個物件，交由 **labelEncoder** 變數保存起來。
3. **.fit\_transform(Y)**
  - 計算指定各標籤的數值（**fit** 部分）
  - 將各標籤轉化為數值（**transform** 部分）
4. **.astype("float64")**
  - 為了與 X 陣列的浮點數運算時，**型態相同**而作。
  - **非必要**。不做的話，運算時頂多產生「警告」訊息。



- 請在 `Preprocessing.ipynb` 引入 `LabelEncoder`
  - `from sklearn.preprocessing import LabelEncoder`
- 用 `LabelEncoder` 類別，產生一個物件，交給 `labelEncoder` 變數保存起來：
  - `labelEncoder = LabelEncoder()`
- 將文字計算、轉化為數字，並以 64 bits 浮點數保存之
  - `Y = labelEncoder.fit_transform(Y).astype("float64")`
- 參考程式碼如下所示：

```
1 from sklearn.preprocessing import LabelEncoder
2
3 labelEncoder = LabelEncoder()
4 Y = labelEncoder.fit_transform(Y).astype("float64")
```



- 使用「快樂版」，做「標籤編碼器 (LabelEncoder)」

原始碼說明

要被轉換的 DataFrame

是否要將 No=0, Yes=1... 整理成一個對照表傳回

```
62 def label_encoder(ary, mapping=False):
63     encoder = LabelEncoder()
64     columns = ary.columns
65     index = ary.index
66     encoder.fit(ary.values.ravel())
67     mapper = {k:v for k, v in enumerate(list(encoder.classes_))}
68     encoded_ary = pd.DataFrame(encoder.transform(ary.values.ravel()), index=index, columns=columns)
69
70     if mapping:
71         return encoded_ary, mapper
72     else:
73         return encoded_ary
```

← 標準函式庫的「標籤編碼器」

保存 DataFrame 的欄列名稱

← (1) ary 轉成 NDAarray，並弭平維度 (2) 用 .fit() 計算各標籤應對應的數值

← (1) 標籤轉 list 並加上索引值 (2) 排列成字典

真正將算出來的對應值，寫回原始之 NDAarray 中

將欄列名稱合併回去

(1) 如果 mapping = True：傳回編碼結果 + 對照表  
(2) 如果 mapping = False：僅傳回編碼結果

# 如何做「類別資料數位化」？



- 使用「快樂版」，做「標籤編碼器（LabelEncoder）」

使用方法（`mapping = False`）

```
1 import HappyML.preprocessor as pp
2
3 Y = pp.label_encoder(Y)
```

Y	
Index	ToBuy
0	No
1	Yes
2	No
3	No
4	Yes




Y	
Index	ToBuy
0	0
1	1
2	0
3	0
4	1

使用方法（`mapping = True`）

```
1 import HappyML.preprocessor as pp
2
3 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
```

Y	
Index	ToBuy
0	No
1	Yes
2	No
3	No
4	Yes



Y	
Index	ToBuy
0	0
1	1
2	0
3	0
4	1

+

Y_mapping			
Key	Type	Size	Value
0	str	1	No
1	str	1	Yes

好處：(1) 使用 DataFrame，能保留欄位名稱。(2) 能自由傳回轉換用對照表（`mapping = True`）。

- 請在 `Preprocessing_Happy.ipynb` 撰寫下列程式碼，並且執行看看：

```
1 import HappyML.preprocessor as pp
2
3 Y = pp.label_encoder(Y)
```

- 請用草稿程式碼儲存格，觀察下列變數的值：
  - `Y`：是否已經從 `Yes/No` 變成 `1/0` 了？
- 請再把程式碼改成下列這個樣子，再執行看看：

```
1 import HappyML.preprocessor as pp
2
3 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
```

- 請用草稿程式碼儲存格，看看是否多出下列這個變數？
  - `Y_mapping`：是否為一個字典，裡面包含 `Yes/No` 對 `1/0` 的對應值？



# 如何做「類別資料數位化」？



- 法二：使用「One-Hot Encoder」：

City	Children	Age	Salary
Taipei	nan	44	72000
Taichung	0	27	48000
Kaohsiung	0	30	54000
Taichung	1	38	61000
Kaohsiung	2	40	nan
Taipei	2	35	58000
Taichung	1	nan	52000
Taipei	2	48	79000
Kaohsiung	1	50	83000
Taipei	2	37	67000



City			Children	Age	Salary
Kaohsiung	Taichung	Taipei	Children	Age	Salary
0	0	1	1.22222	44	72000
0	1	0	0	27	48000
1	0	0	0	30	54000
0	1	0	1	38	61000
1	0	0	2	40	63777.8
0	0	1	2	35	58000
0	1	0	1	38.7778	52000
0	0	1	2	48	79000
1	0	0	1	50	83000
0	0	1	2	37	67000

1 個標籤，1 個欄位 = 獨熱編碼器 (One Hot Encoder)

# 為何不用「LabelEncoder」就好？



- 不想賦予「**訓練模型**」  $\text{Taipei}(2) > \text{Taichung}(1) > \text{Kaohsiung}(0)$  的數學意義。
- 萬一**混成**資料集時，發生  $\text{Taipei}$  同時等於**兩種值**的情況時，比較好**矯正**回來。

LabelEncoder 1 :

City	Children	Age
Taipei =0	nan	44
Taichung =1	0	27
Kaohsiung =2	0	30

+

LabelEncoder 2 :

Taichung =0	1	38
Kaohsiung =1	2	40
Taipei =2	2	35

標籤數字不同  
不太好合併

OneHotEncoder 1

[0]	[1]	[2]
Kaohsiung =2	Taichung =1	Taipei =0
0	0	1
0	1	0
1	0	0

OneHotEncoder 2

[0]	[1]	[2]
Taipei =2	Kaohsiung =1	Taichung =0
0	0	1
0	1	0
1	0	0

只要交換欄位順序，就可合併



# 應變數 Y 為何不用「OneHotEncoder」？



City	Children	Age	Salary	ToBuy
Taipei	nan	44	72000	0
Taichung	0	27	48000	1
Kaohsiung	0	30	54000	0
Taichung	1	38	61000	0
Kaohsiung	2	40	nan	1
Taipei	2	35	58000	1
Taichung	1	nan	52000	0
Taipei	2	48	79000	1
Kaohsiung	1	50	83000	0
Taipei	2	37	67000	1

- 應變數的「數字」不會參與計算。
- 應變數的值，僅作為答案、代表某種分類結果而已。
- 如果要套用 **OneHotEncoder**，應變數會變成兩欄以上，程式比對「答案」時，較不好寫。



# 如何做「One-Hot Encoder」？



- 使用 pandas 的「`get_dummies()`」函數

City	Children	Age	Salary
Taipei	nan	44	72000
Taichung	0	27	48000
Kaohsiung	0	30	54000
Taichung	1	38	61000
Kaohsiung	2	40	nan
Taipei	2	35	58000
Taichung	1	nan	52000
Taipei	2	48	79000
Kaohsiung	1	50	83000
Taipei	2	37	67000

`pd.get_dummies(X[:, 0])`

City		
Kaohsiung	Taichung	Taipei
0	0	1
0	1	0
1	0	0
0	1	0
1	0	0
0	0	1
0	1	0
0	0	1
1	0	0
0	0	1

• 原始程式碼

```
1 ary_dummies = pd.get_dummies(X[:, 0]).values
2 X = np.concatenate((ary_dummies, X[:, 1:4]), axis=1).astype("float64")
```

- 1. `.get_dummies()`：一次傳入「一欄」，並且針對那一欄做「One-Hot Encoding」後，以 `DataFrame` 傳回。
- 2. `.values`：將 `DataFrame` 轉成 `NDArray` 傳回來。
- 3. `.concatenate()`：合併多個 `NDArray`。
  - `(ary_dummies, X[:, 1:4])`：將放在第一參數、`Tuple` 內的所有 `NDArray` 合併。
  - `axis = 1`：=0 代表「以『列』為方向合併」。=1 代表「以『欄』為方向合併」。
- 4. `.astype("float64")`：將合併後的 `NDArray`，全數改成 `float64`（8 Bytes 雙精確度浮點數）來儲存。  
（這也是人工智慧最常使用的 `NDArray` 資料型態）

- 請在 [Preprocessing.ipynb](#) 撰寫下列程式碼，並且執行看看：

```
1 ary_dummies = pd.get_dummies(X[:, 0]).values
2 X = np.concatenate((ary_dummies, X[:, 1:4]), axis=1).astype("float64")
```

- 請用**草稿程式碼儲存格**，觀察 X 是否已經被 One-Hot Encoding 過了？



• 原始碼解說：

- `ary`：整個陣列，預設為 `DataFrame` 型態。
- `columns`：你要針對哪幾欄做 One-Hot Encoding。如：`[1, 3, 7]`。
- `remove_trap`：是否移除「共線性」陷阱（以後會教，目前先不要使用）

```
77 def onehot_encoder(ary, columns=[], remove_trap=False):
78     df_results = pd.DataFrame() ← 存放最終結果的「空 DataFrame」
79
80     # Iterate each column in DataFrame ary
81     for i in range(ary.shape[1]): ← 針對整個 DataFrame，每一個欄位做迭代
82         # if this column (i) is dummy column
83         if i in columns: ← 如果迭代出來的欄位，就是要做 One-Hot Encoding 的欄位
84             base_name = ary.columns[i]
85             this_column = pd.get_dummies(ary.iloc[:, i])
86             this_column = this_column.rename(columns={n:"{}_{}".format(base_name, n) for n in this_column.columns})
87             # Remove Dummy Variable Trap if needed
88             if remove_trap: ← 如果需要移除「共線性」陷阱，就挑「第 0 欄」移除（以後會教）
89                 this_column = this_column.drop(this_column.columns[0], axis=1)
90         # else this column is normal column
91         else: ← 如果迭代出來的欄位，不用做 One-Hot Encoding，就直接保留原欄位不做任何變更
92             this_column = ary.iloc[:, i]
93         # Append this column to the Result DataFrame ← 將做好的欄位（不管有無 Encoding），附加到最終結果上
94         df_results = pd.concat([df_results, this_column], axis=1)
95
96     return df_results
```

• 使用方法

```
1 import HappyML.preprocessor as pp
2
3 X = pp.onehot_encoder(X, columns=[0])
```

• 執行結果

Index	City_Kaohsiung	City-Taichung	City-Taipei	Children	Age	Salary
0	0	0	1	1.22222	44	72000
1	0	1	0	0	27	48000
2	1	0	0	0	30	54000
3	0	1	0	1	38	61000
4	1	0	0	2	40	63777.8
5	0	0	1	2	35	58000
6	0	1	0	1	38.7778	52000
7	0	0	1	2	48	79000
8	1	0	0	1	50	83000
9	0	0	1	2	37	67000

「快樂版」優點：

- 使用 DataFrame，能保持欄位名稱。
- 能一口氣指定多個需要做 One-Hot Encoding 的欄位，並省卻合併的煩惱。



- 請在 `Preprocessing_Happy.ipynb` 撰寫下列程式碼，並且執行看看：

```
1 import HappyML.preprocessor as pp
2
3 X = pp.onehot_encoder(X, columns=[0])
```

- 請用**草稿程式碼儲存格**，觀察 X 是否已經被 One-Hot Encoding 過了？





# 切分 訓練集、測試集

( Training Set 、 Testing Set )

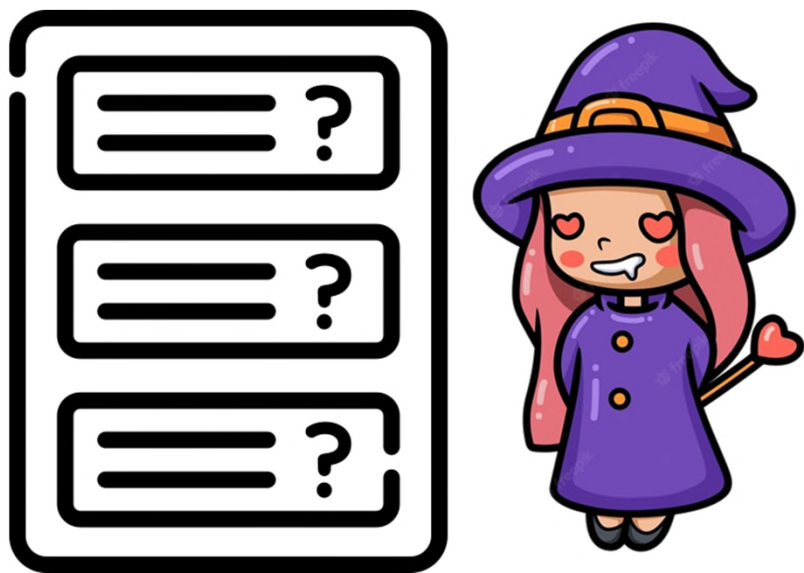




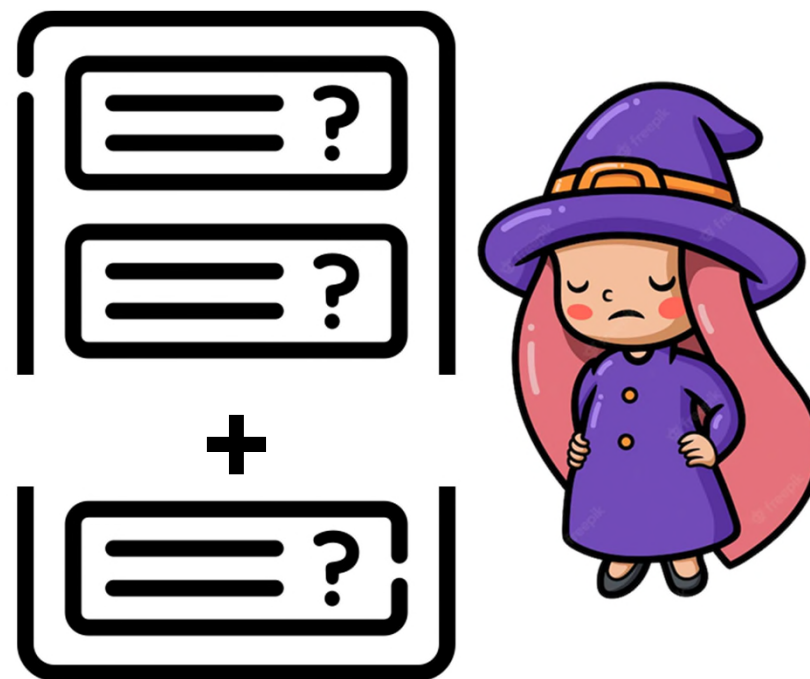
# 為何要切分「訓練集」與「測試集」？



- 哪一種方式，最能測驗出學習成效？



練習所有題目，從中抽考



練習部分題目，保留部分試題抽考

• 使用「`train_test_split`」物件

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

4

1

2

3

1. `X, Y`

- 自變數、應變數矩陣
2. `test_size=0.2`

- 測試集佔比
  - 也可用訓練集佔比 `train_size=0.8` 來指定比例
3. `random_state=0`

- 若為整數：作為亂數產生器的「亂數種子」（亂數序列的起點）此時，每次執行程式，每次切分「訓練集、測試集」的方法會一樣。有助於維持每次執行結果相同。
  - 若為 `None`：使用預設的 `np.random.RandomState` 來控制亂數如何產生
4. 依序產生自變數「訓練、測試集」，與應變數「訓練、測試集」



- 請在 `Preprocessing.ipynb` 引入下列套件：
  - `from sklearn.model_selection import train_test_split`
- 用下列程式碼切分出「訓練集、測試集」：
  - `X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)`
- 請在執行完畢後，用「**草稿程式碼儲存格**」，觀看 `X_train, X_test, Y_train, Y_test` 四個變數的內容。
- 參考程式碼如下所示：

```
1 from sklearn.model_selection import train_test_split
2
3 X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```



• 原始碼解說

```
101 from sklearn.model_selection import train_test_split
102 import time
103
104 def split_train_test(x_ary, y_ary, train_size=0.75, random_state=int(time.time())):
105     return train_test_split(x_ary, y_ary, test_size=(1-train_size), random_state=random_state)
```

- 1. **x\_ary, y\_ary**：要被切分成「訓練集」、「測試集」的**自變數**、**應變數**。
- 2. **train\_size**：訓練集大小。預設值是 **75%**。
- 3. **random\_state**：抽選「訓練集」、「測試集」的「**亂數種子**」。
  - = **特定數字**：每次執行此程式時，會切出一模一樣的「訓練集」與「測試集」。常用的數字是 **=0**。適合撰寫論文時，希望每次執行、每次結果相同時使用。
  - = **int(time.time())**：**time.time()** 乃取系統時鐘的秒數。並用 **int()** 將它轉為純整數。用這種方法切出來的「訓練集」與「測試集」會每次執行、每次不同。

• 使用方法

```
1 import HappyML.preprocessor as pp
2
3 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y, train_size=0.8, random_state=0)
```

• 執行結果

X\_train

Index	City_Kaohsiung	City-Taichung	City_Taipei	Children	Age	Salary
4	1	0	0	2	40	63777.8
9	0	0	1	2	37	67000
1	0	1	0	0	27	48000
6	0	1	0	1	38.7778	52000
7	0	0	1	2	48	79000
3	0	1	0	1	38	61000
0	0	0	1	1.22222	44	72000
5	0	0	1	2	35	58000

X\_test

Index	City_Kaohsiung	City-Taichung	City_Taipei	Children	Age	Salary
2	1	0	0	0	30	54000
8	1	0	0	1	50	83000

Y\_train

Index	ToBuy
4	1
9	1
1	1
6	0
7	1
3	0
0	0
5	1

Y\_test

Index	ToBuy
2	0
8	0

- 請在 `Preprocessing_Happy.ipynb` 撰寫下列程式碼，並且執行看看：

```
1 import HappyML.preprocessor as pp
2
3 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y, train_size=0.8, random_state=0)
```

- 請用草稿程式碼儲存格，觀察系統是否已經產生下列資料集了：
  - `X_train`
  - `X_test`
  - `Y_train`
  - `Y_test`

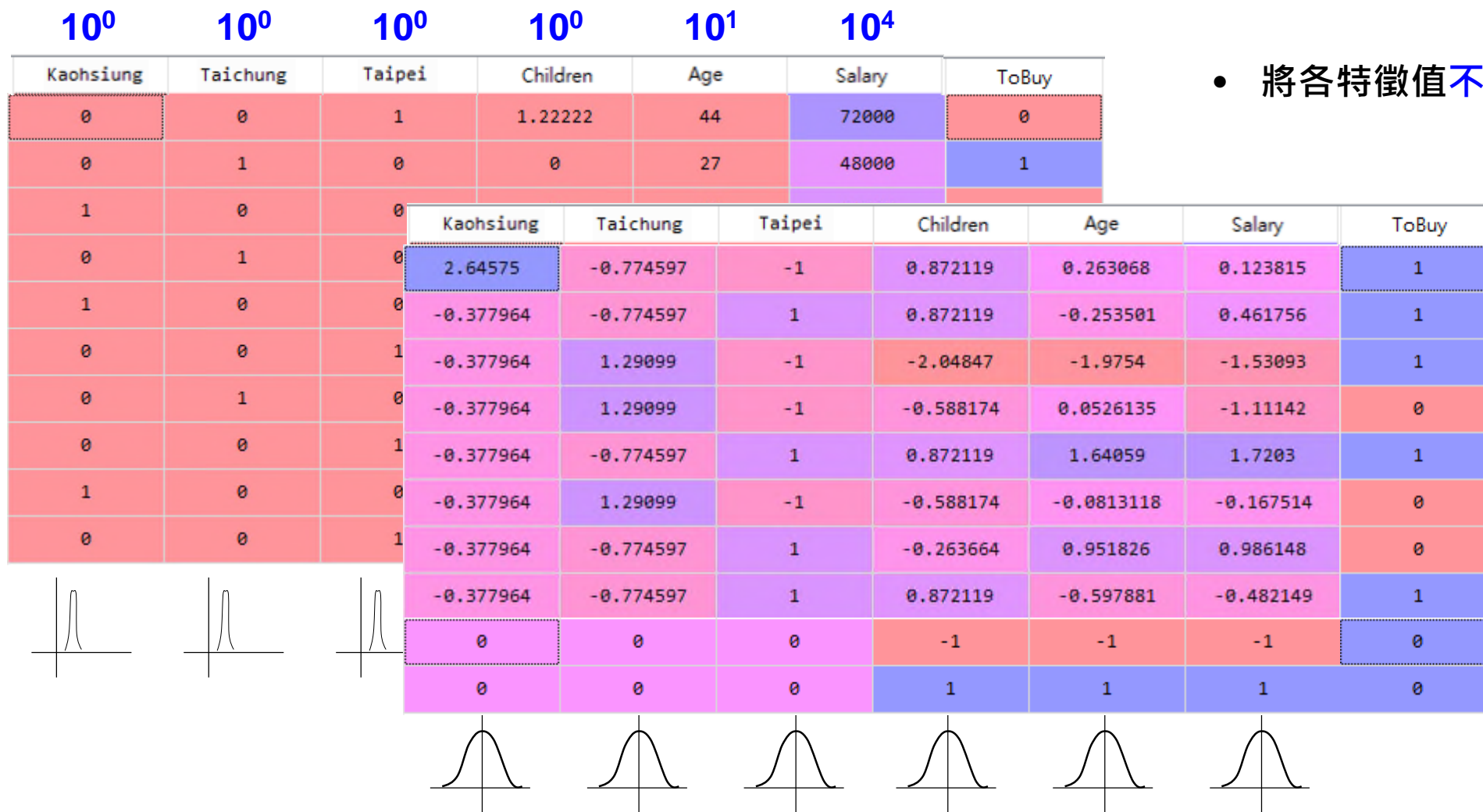




# 特徵縮放

( Feature Scaling )

# 什麼是「特徵縮放」



- 將各特徵值不同的比例尺

- 用統計學的常態分布正規化

$$\hat{x} = \frac{x - \text{Avg}(x)}{\text{Std}(x)}$$

- 將比例尺化為相同

$$\hat{x} \sim (\mu = 0 \quad \sigma = 1)$$



$X_1, X_2, \dots, X_6$  若比例尺相同  
 $\rightarrow C_i$  單純，收斂快！

Kaohsiung	Taichung	Taipei	Children	Age	Salary	ToBuy
0	0	1	1.22222	44	72000	0
0	1	0	0	27	48000	1
1	0	0	0	30	54000	0
0	1	0	1	38	61000	0
1	0	0	2	40	63777.8	1
0	1	0	1	35	58000	1
0	1	0	1	38.7778	52000	0
0	0	1	2	48	79000	1
1	0	0	1	50	83000	0
0	0	1	2	37	67000	1

$$Y = C_1X_1 + C_2X_2 + C_3X_3 + C_4X_4 + C_5X_5 + C_6X_6$$

1

1

1

1

1

1

-1

0

+1

0

-1

+3

...

+0.1

-0.44

-1.58

-2.33

-3.5

+3.2x10<sup>-4</sup>

$C_1X = \hat{Y}_1$   
 $C_2X = \hat{Y}_2$   
 $C_nX = \hat{Y}_n$

透過調整  $C_i$ ，慢慢收斂至正確答案

# 何時該做「特徵縮放」



## 並非所有人工智慧演算法都需要特徵縮放

有明確公式可求最佳解的演算法

Kaohsiung	Taichung	Taipei	Children	Age	Salary	ToBuy
0	0	1	1.22222	44	72000	0
0	1	0	0	27	48000	1
1	0	0	0	30	54000	0
0	1	0	1	38	61000	0
1	0	0	2	40	63777.8	1
0	0	1	2	35	58000	1
0	1	0	1	38.7778	52000	0
0	0	1	2	48	79000	1
1	0	0	1	50	83000	0
0	0	1	2	37	67000	1

$$\frac{d}{dX} Y = \frac{d}{dX} (C_1 X_1 + \dots + C_n X_n) = 0$$

可做可不做

只能用試誤法慢慢逼近最佳解的演算法

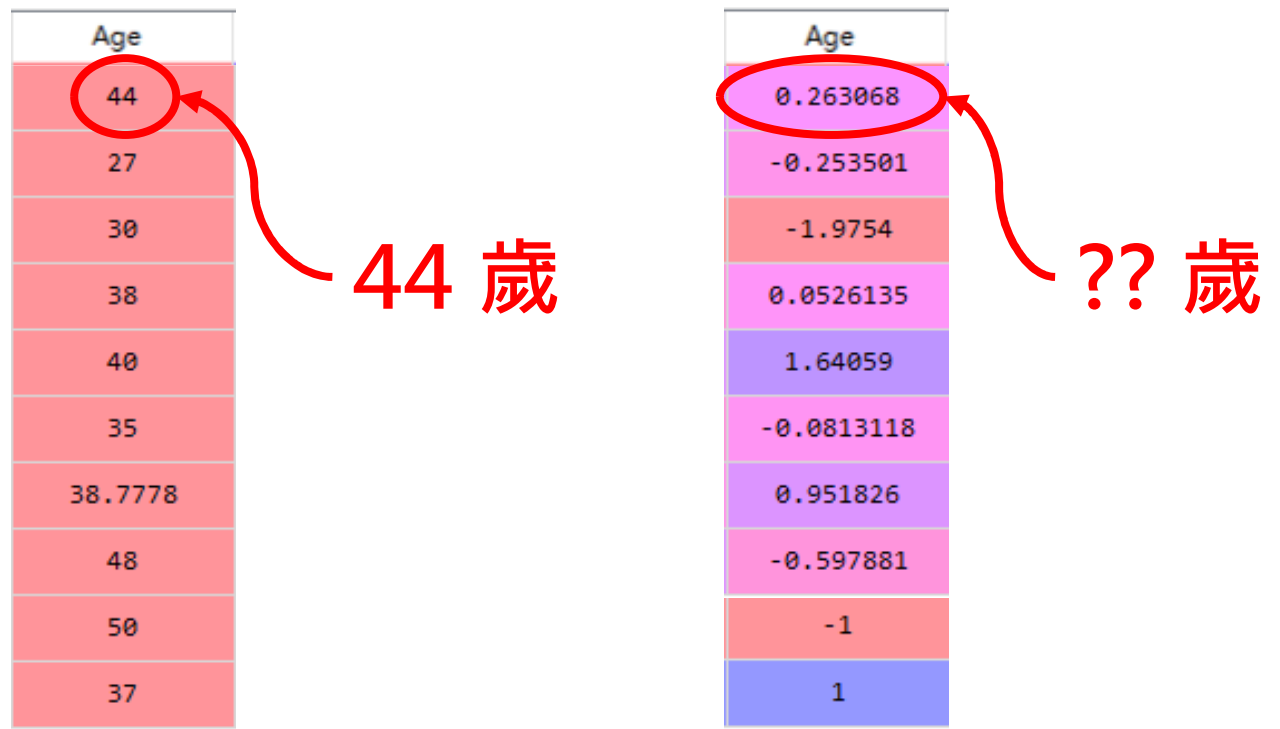
Kaohsiung	Taichung	Taipei	Children	Age	Salary	ToBuy
2.64575	-0.774597	-1	0.872119	0.263068	0.123815	1
-0.377964	-0.774597	1	0.872119	-0.253501	0.461756	1
-0.377964	1.29099	-1	-2.04847	-1.9754	-1.53093	1
-0.377964	1.29099	-1	-0.588174	0.0526135	-1.11142	0
-0.377964	-0.774597	1	0.872119	1.64059	1.7203	1
-0.377964	1.29099	-1	-0.588174	-0.0813118	-0.167514	0
-0.377964	-0.774597	1	-0.263664	0.951826	0.986148	0
-0.377964	-0.774597	1	0.872119	-0.597881	-0.482149	1
0	0	0	-1	-1	-1	
0	0	0	1	1	1	

$$Y = C_1 X_1 + C_2 X_2 + C_3 X_3 + C_4 X_4 + C_5 X_5 + C_6 X_6$$

+0.1      -0.44      -1.58      -2.33      -3.5      +3.2x10<sup>-4</sup>

強烈建議要做

- 使得數字失去「原始意義」



**解決方法**

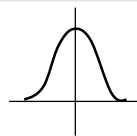
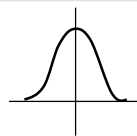
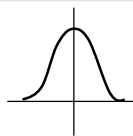
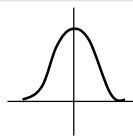
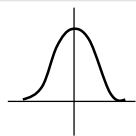
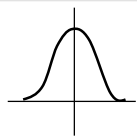
- 保留轉換前的數字，不要覆蓋掉，便於日後對照、追查。

# 應變數需要做「特徵縮放」嗎？



Kaohsiung	Taichung	Taipei	Children	Age	Salary	ToBuy
2.64575	-0.774597	-1	0.872119	0.263068	0.123815	1
-0.377964	-0.774597	1	0.872119	-0.253501	0.461756	1
-0.377964	1.29099	-1	-2.04847	-1.9754	-1.53093	1
-0.377964	1.29099	-1	-0.588174	0.0526135	-1.11142	0
-0.377964	-0.774597	1	0.872119	1.64059	1.7203	?
-0.377964	1.29099	-1	-0.588174	-0.0813118	-0.167514	?
-0.377964	-0.774597	1	-0.263664	0.951826	0.986148	0
-0.377964	-0.774597	1	0.872119	-0.597881	-0.482149	1
0	0	0	-1	-1	-1	0
0	0	0	1	1	1	0

- 可做，可不做
- 特徵縮放是讓「自變數」對「應變數」的影響力一致！
- 反正「應變數」只是某種「答案」的代表符號而已！
- 若應變數做了特徵縮放，將來有新進資料，平均值與標準差會變動，得再重算。
- 所以應變數一般而言，很少做特徵縮放。



• 使用「標準縮放器 ( StandardScaler ) 」物件

```
1 from sklearn.preprocessing import StandardScaler
2
1 3 sc_X = StandardScaler().fit(X_train)
2 4 X_train = sc_X.transform(X_train)
3 5 X_test = sc_X.transform(X_test)
```

1. StandardScaler().fit(X\_train)
  - 產生一個「StandardScaler」物件，並產生縮放計算模型。
2. sc\_X.transform(X\_train)
  - 用縮放後的值，替換 X\_train 內的原值。
3. sc\_X.transform(X\_test)
  - 用縮放後的值，替換 X\_test 內的原值。

$$\hat{x} = \frac{x - Avg(x)}{Std(x)}$$

Age		Age
44		0.263068
27		-0.253501
30		-1.9754
38	X_train	0.0526135
40		1.64059
35		-0.0813118
38.7778		0.951826
48		-0.597881
50		-1
37		
	X_test	1

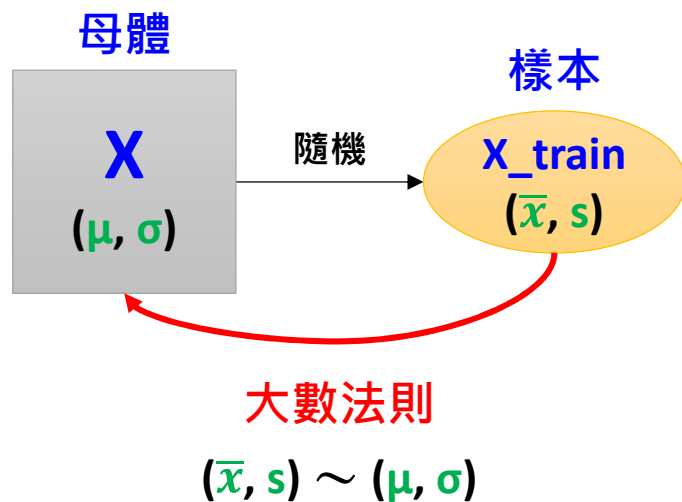
$\bar{x} = 37.60$   
 $s = 6.90$

# 使用「標準函式庫」來做特徵縮放



- 為何使用 `.fit(X_train)`，而不是 `.fit(X)`

```
1 from sklearn.preprocessing import StandardScaler
2
3 sc_X = StandardScaler().fit(X_train)
4 X_train = sc_X.transform(X_train)
5 X_test = sc_X.transform(X_test)
```



Age	Age	Age
44	0.263068	44
27	-0.253501	27
30	-1.9754	30
38	0.0526135	38
40	1.64059	40
35	-0.0813118	35
38.7778	0.951826	38.7778
48	-0.597881	48
50	-1	50
37	-1	37

$X_{\text{train}}$   
 $(\bar{x}, s)$

$X_{\text{train}}$   
 $(\mu, \sigma)$

$X_{\text{test}}$   
 $(\mu, \sigma)$

- 請在 `Preprocessing.ipynb` 引入 `StandardScaler` 物件
  - `from sklearn.preprocessing import StandardScaler`
- 請產生一個 `StandardScaler` 物件，並順便產生縮放計算模型
  - `sc_X = StandardScaler().fit(X_train)`
- 用下列程式碼，用產生出來的縮放計算模型，計算新值，取代原值
  - `X_train = sc_X.transform(X_train)`
  - `X_test = sc_X.transform(X_test)`
- 參考程式碼如下所示：

```
1 from sklearn.preprocessing import StandardScaler
2
3 sc_X = StandardScaler().fit(X_train)
4 X_train = sc_X.transform(X_train)
5 X_test = sc_X.transform(X_test)
```





```

116 from sklearn.preprocessing import StandardScaler
117
118 def feature_scaling(fit_ary, transform_arrys=None):
119     scaler = StandardScaler()
120     scaler.fit(fit_ary.astype("float64"))
121
122     if type(transform_arrys) is tuple:    ← transform_arrys = (X_train, X_test) 的情況
123         return (pd.DataFrame(scaler.transform(arry.astype("float64")), index=arry.index, columns=arry.columns) for arry in transform_arrys)
124     else:                                ← transform_arrys = X_train 的情況 ( 免除使用 transform_arrys = (X_train,) 這樣的語法 )
125         return pd.DataFrame(scaler.transform(transform_arrys.astype("float64"), index=transform_arrys.index, columns=transform_arrys.columns))

```

```
1 import HappyML.preprocessor as pp
2
3 X_train, X_test = pp.feature_scaling(X_train, transform_arrys=(X_train, X_test))
```



- 請在 **Preprocessing\_Happy.ipynb** 撰寫下列程式碼，並且執行看看：

```
1 import HappyML.preprocessor as pp
2
3 X_train, X_test = pp.feature_scaling(X_train, transform_arrys=(X_train, X_test))
```

- 請用**草稿程式碼儲存格**，觀察系統是否已經針對下列兩個**資料集**做**特徵縮放**了：
  - X\_train
  - X\_test



# 課後作業：健康檢查資料前處理



## • 要求

- 請至 Moodle 下載 **HealthCheck.csv** 這份資料集，再上傳至 Colab。或者利用 **wget** 指令，透過下列網址，直接下載資料集至 Colab：

<https://raw.githubusercontent.com/cnchi/datasets/master/HealthCheck.csv>

- 針對資料集，執行下列「**資料前處理**」動作：
  - 補足**空白資料**。
  - 為所有「**類別資料**」編碼，包括**自變數**與**應變數**。
  - 切出「**訓練集**」與「**測試集**」。
  - 針對自變數，以**標準常態分佈**模型，執行「**特徵縮放**」。

- 最後印出下列四個陣列：

- 自變數**訓練集、**應變數**訓練集
- 自變數**測試集、**應變數**測試集

- 輸出結果如下所示：

自變數訓練集：

```
[[ 1.15470054 -0.63245553 -0.63245553 1.30941777 1.89111602]
 [-0.8660254 1.58113883 -0.63245553 -1.19037979 0.07416141]
 [ 1.15470054 -0.63245553 -0.63245553 -1.07134181 -0.30076573]
 [-0.8660254 -0.63245553 1.58113883 0.47615192 -0.76221452]
 [ 1.15470054 -0.63245553 -0.63245553 -0.47615192 -0.76221452]
 [-0.8660254 -0.63245553 1.58113883 1.42845575 0.96821844]
 [-0.8660254 1.58113883 -0.63245553 -0.47615192 -1.10830111]]
```

應變數訓練集：[0. 0. 1. 0. 1. 1. 1.]

自變數測試集：

```
[[ -0.8660254 -0.63245553 1.58113883 0.01487975 0.85285625]
 [-0.8660254 1.58113883 -0.63245553 0.11903798 -0.18540353]]
```

應變數測試集：[1. 0.]



• 以下是我們資料前處理的模版：

```
1 import HappyML.preprocessor as pp
2
3 # In[] Load data
4 dataset = pp.dataset(file="CarEvaluation.csv")
5
6 # In[] Decomposition
7 X, Y = pp.decomposition(dataset, x_columns=[i for i in range(4)], y_columns=[4])
8
9 # In[] Missing Data
10 X = pp.missing_data(X, strategy="mean")
11
12 # In[] Categorical Data Encoding
13 # Label Encoding
14 Y, Y_mapping = pp.label_encoder(Y, mapping=True)
15 # One-Hot Encoding
16 X = pp.onehot_encoder(X, columns=[0])
17
18 # In[] Split Training Set, Testing Set
19 X_train, X_test, Y_train, Y_test = pp.split_train_test(X, Y, train_size=0.8, random_state=0)
20
21 # In[] Feature Scaling for X_train, X_test
22 X_train, X_test = pp.feature_scaling(X_train, transform_arys=(X_train, X_test))
```

- 1 載入資料
- 2 切分自變數、應變數
- 3 補足缺失資料
- 4 類別資料數位化
- 5 切分訓練集、測試集
- 6 特徵縮放



標準版

- 載入資料
  - pandas.read\_csv("檔案路徑 & 名稱")
- 切分「自變數、應變數」
  - DataFrame.iloc[列切片, 欄切片]
- 缺失資料補足
  - sklearn.impute.SimpleImputer(strategy="mean")
- 類別資料數位化
  - sklearn.preprocessing.LabelEncoder
  - pandas.get\_dummies()
- 切分「訓練集、測試集」
  - sklearn.model\_selection.train\_test\_split
- 特徵縮放
  - sklearn.preprocessing.StandardScaler

快樂版

- 載入資料
  - pp.dataset("檔案路徑 & 名稱")
- 切分「自變數、應變數」
  - pp.decomposition(dataset, x\_columns=[...], y\_columns=[...])
- 缺失資料補足
  - pp.missing\_data(X, strategy="mean")
- 類別資料數位化
  - pp.label\_encoder(Y, mapping=True)
  - pp.onehot\_encoder(X, columns=[...])
- 切分「訓練集、測試集」
  - pp.split\_train\_test(X, Y, train\_size=0.8, random\_state=0)
- 特徵縮放
  - pp.feature\_scaling(X\_train, transform\_ays=(X\_train, X\_test))