# Regression Model

**Prof. Chia-Yu Lin**

**Yuan Ze University**

**2022 Spring**

Thanks to the slides of Prof. P. Domingos from Washington University, Prof. H.-T. Lin and Prof. Lee Hung-Yi Lee from NTU.

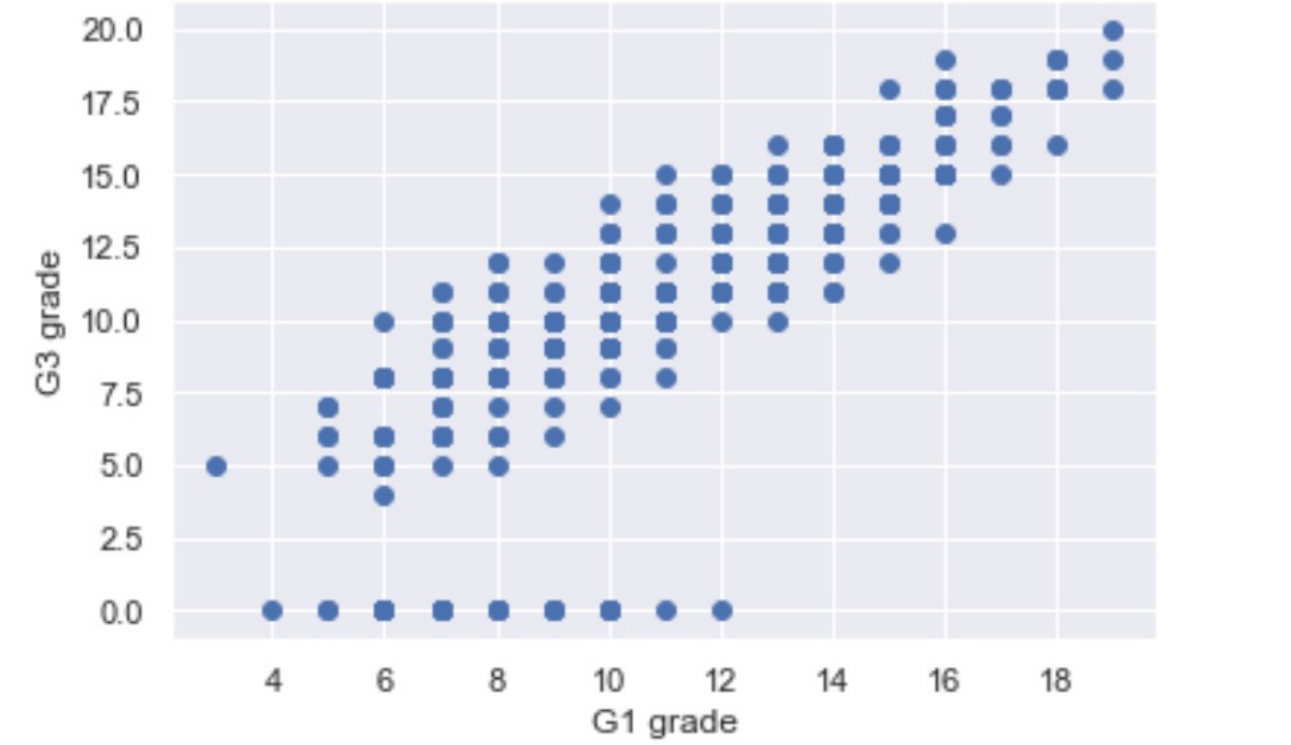
# 簡單線性迴歸
# (Simple Linear Regression)

# 迴歸分析

- 迴歸分析是預測數值的分析
- 機器學習會預測資料，成為機器學習基礎的就是接下來要介紹的迴歸分析

# 觀察

- G1與G3似乎有關係



# 迴歸問題

- 在迴歸問題裡，從給予的資料來假設關係式，逐步求取最合適的係數
- 例如：已知G1的成績，預測G3的成績
- 將G3視為目標(目標變數)，G1為解釋變數，來進行預測
- 這就是之前提到的「監督式學習」之一：在學習時的資料逐一給予正確解答，作為計算關聯性的基礎

# 迴歸分析

- 對於輸出(目標變數)與輸入(解釋變數)的關係，若是輸入一個變數則為「簡單迴歸分析」
- 若是輸入2個以上的變數，則為「多元迴歸分析」

# 簡單線性迴歸分析

- 輸出與輸入之間成立線性關係(y=ax+b)，則為簡單線性迴歸分析
- scikit-learn為機器學習的套件
- 我們可以使用scikit-learn引入linear_model進行簡單線性迴歸分析

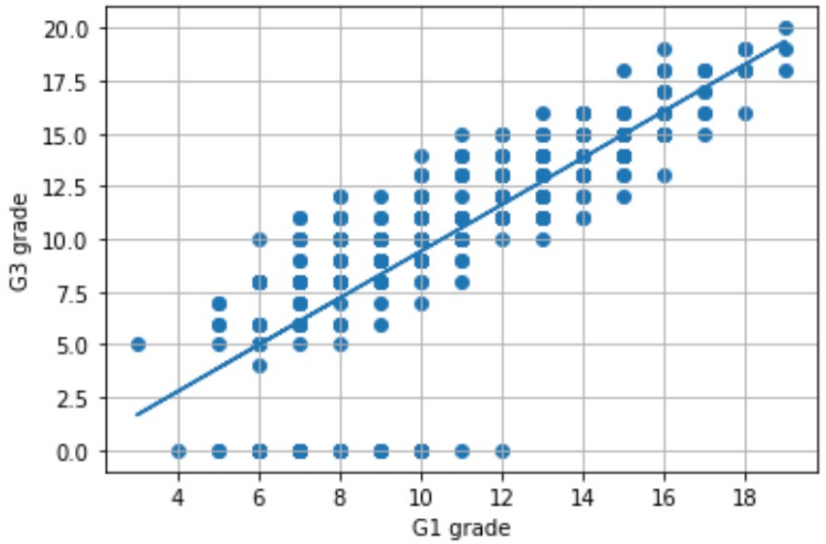
```
from sklearn import linear_model
#生成線性迴歸模型
reg = linear_model.LinearRegression()
```

# fit函式

- 解釋變數(X)和目標變數(Y)資料，使用線性迴歸的fit函式功能，計算預測模型
- 使用最小平方法來計算迴歸係數a與截距b

```
# 解釋變數使用第一學期的成績
#loc從DataFrame取出指定的列與行
#loc[:,['G1']]會取出G1行的所有列
X = student_data_math.loc[:, ['G1']].values

# 目標函數使用最後一學期的數學成績
Y = student_data_math['G3'].values

# 計算預測模型，在此算出a,b
reg.fit(X, Y)

# 迴歸係數
print('迴歸係數:', reg.coef_)

# 截距
print('截距 :', reg.intercept_)
```

```
迴歸係數: [1.106]
截距 : -1.6528038288004616
```



# 決定係數

- 從上頁的圖形，預測式似乎能很好地預測出實際值，但是否客觀無從判斷
- 因此我們需要「決定係數」（Coefficient of determination）
- 決定係數稱為貢獻度，以$R^2$表示
- $R^2$為1，越接近1是越好的模型
- 多高才叫做好？ 教科書寫0.9，實務上很難達到，是情況而定
- 0.64雖不高，但是堪用

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - f(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

```
# 決定係數
print('決定係數:', reg.score(X, Y))
```

```
決定係數: 0.64235084605227
```

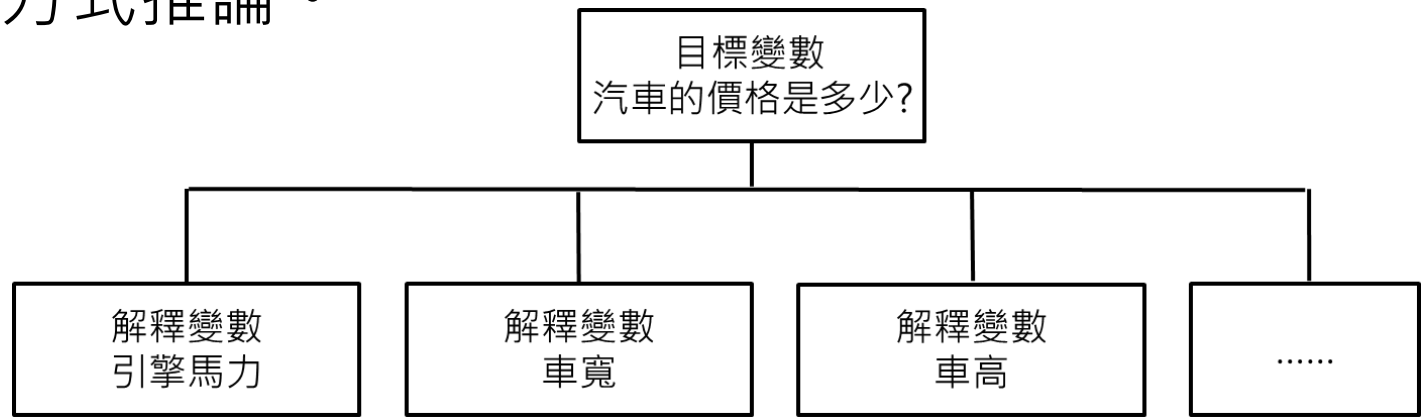
# 多元線性迴歸
# (Multiple Linear Regression)

# 線性迴歸

- 一般線性迴歸: 對於目標變數只有一個解釋變數。

  $y=ax+b$
- 多元線性迴歸: 解釋變數有多個。

  $y=a_1x_1+a_2x_2+a_3x_3+.....+b$
- 藉由多元線性迴歸，可以計算出各個解釋變數之係數(迴歸係數)的推論預測值。
- 迴歸係數是以讓預測值與目標變數的平方誤差最小化的方式推論。



# 建置預測汽車價格的模型

- 汽車價格和汽車的一些屬性(汽車的大小等)有相關聯

- 目標：利用多元線性回歸，建構出能從這些屬性預測汽車價格的模型。

# 引入資料處理library

```
#先導入資料處理會用到的模組
import numpy as np
import numpy.random as random
import scipy as sp
from pandas import Series, DataFrame
import pandas as pd

# 可視化模組
import matplotlib.pyplot as plt
import matplotlib as mpl
import seaborn as sns
%matplotlib inline

# 機器學習模組
import sklearn

# 表示到小數第三位
%precision 3
```

# 讀取汽車售價資料 (1/4)

- 汽車售價資料：
  - http://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data

```
import requests, zipfile
import io

# 取得汽車價格資料
url = 'http://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data'
res = requests.get(url).content

# 將取得的資料作為DataFrame讀取
auto = pd.read_csv(io.StringIO(res.decode('utf-8')), header=None)

# 在資料的行裡設定標籤
auto.columns =['symboling','normalized-losses','make','fuel-type' ,'aspiration','num-of-doors',
                           'body-style','drive-wheels','engine-location','wheel-base','length','width','height',
                           'curb-weight','engine-type','num-of-cylinders','engine-size','fuel-system','bore',
                           'stroke','compression-ratio','horsepower','peak-rpm','city-mpg','highway-mpg','price']
```

# 讀取汽車售價資料 (2/4)

- 觀察是什麼樣的資料

```
print('汽車資料的形式:{}'.format(auto.shape))
```

```
汽車資料的形式:(205, 26)
```

- 為205列26行的資料

# 讀取汽車售價資料 (3/4)

- 用head()顯示最開始的五列

```
#用head()顯示最開始的五列
auto.head()
```

| | symboling | normalized-losses | make | fuel-type | aspiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 |
| **1** | 3 | ? | alfa-romero | gas | std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 |
| **2** | 1 | ? | alfa-romero | gas | std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 |
| **3** | 2 | 164 | audi | gas | std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 |
| **4** | 2 | 164 | audi | gas | std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 |

5 rows × 26 columns

| spiration | num-of-doors | body-style | drive-wheels | engine-location | wheel-base | ... | engine-size | fuel-system | bore | stroke | compression-ratio | horsepower | peak-rpm | city-mpg | highway-mpg | price |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 | 5000 | 21 | 27 | 13495 |
| std | two | convertible | rwd | front | 88.6 | ... | 130 | mpfi | 3.47 | 2.68 | 9.0 | 111 | 5000 | 21 | 27 | 16500 |
| std | two | hatchback | rwd | front | 94.5 | ... | 152 | mpfi | 2.68 | 3.47 | 9.0 | 154 | 5000 | 19 | 26 | 16500 |
| std | four | sedan | fwd | front | 99.8 | ... | 109 | mpfi | 3.19 | 3.40 | 10.0 | 102 | 5500 | 24 | 30 | 13950 |
| std | four | sedan | 4wd | front | 99.4 | ... | 136 | mpfi | 3.19 | 3.40 | 8.0 | 115 | 5500 | 18 | 22 | 17450 |

# 讀取汽車售價資料 (4/4)

- 在這個資料裡，汽車的價格訂於「price」中。
- 所以我們的模型就是要從price以外的欄位值來預測price這個欄位。

- 因為從全部的欄位來解釋price會太複雜，以下的範例使用「horsepower、width、height」這三個解釋變數來預測模型。

# 資料前處理

- 收到資料第一步要做什麼？
- 確認資料的「正確性」。

- 接下來呢？
- 尋找資料之間的「相關性」。

# 確認資料正確性 (1/3)

- 資料有時會包含不適當的東西，所以要進行確認，去除不適當的資料。

- 剛剛使用head()之後發現有「?」的資料，可以怎麼處理？
- 如何處理遺漏值？
  - 去除那一列
  - 補0
  - 補前一列的值

# 確認資料正確性 (2/3)

- 先計算各個行(列)裡面有多少個「？」

```
#計算各個行(列)裡面有多少個「?」
auto = auto[['price','horsepower','width','height']]
auto.isin(['?']).sum()
```

```
price          4
horsepower     2
width          0
height         0
dtype: int64
```

# 確認資料正確性 (3/3)

- 去除那一列
- 將「？」轉換回遺漏值(NaN)之後，去除那一列

```
# 將「?」取代成NaN，刪除有NaN的列
auto = auto.replace('?', np.nan).dropna()
print('汽車資料的形式:{}'.format(auto.shape))
```

```
汽車資料的形式:(199, 4)
```

- 刪除之後變成199列, 4行

# 資料型別的轉換 (1/2)

- 確認資料的型別

```
#確認資料的型別
print('資料型別的確認 (型別轉換前)\n{}\n'.format(auto.dtypes))
```

```
資料型別的確認 (型別轉換前)
price             object
horsepower        object
width            float64
height           float64
dtype: object
```

- 如此確認便可得知price和horsepower並非數值型別

# 資料型別的轉換 (2/2)

- 使用to_numeric將price和horsepower轉換為數值型別

```
auto = auto.assign(price=pd.to_numeric(auto.price))
auto = auto.assign(horsepower=pd.to_numeric(auto.horsepower))
print('資料型別的確認（型別轉換後）\n{}'.format(auto.dtypes))
```

```
資料型別的確認（型別轉換後）
price             int64
horsepower        int64
width           float64
height          float64
dtype: object
```

# 相關性確認 (1/2)

- 經過上述已經將目標變數、解釋變數的所有列加工為沒有遺漏且為數執行別的資料形式
- 接下來要確認各個變數的相關性
- 使用corr

```
auto.corr()
```

| | price | horsepower | width | height |
|---|---|---|---|---|
| **price** | 1.000000 | 0.810533 | 0.753871 | 0.134990 |
| **horsepower** | 0.810533 | 1.000000 | 0.615315 | -0.087407 |
| **width** | 0.753871 | 0.615315 | 1.000000 | 0.309223 |
| **height** | 0.134990 | -0.087407 | 0.309223 | 1.000000 |

# 相關性確認 (2/2)

```
auto.corr()
```

| | price | horsepower | width | height |
|---|---|---|---|---|
| **price** | 1.000000 | 0.810533 | 0.753871 | 0.134990 |
| **horsepower** | 0.810533 | 1.000000 | 0.615315 | -0.087407 |
| **width** | 0.753871 | 0.615315 | 1.000000 | 0.309223 |
| **height** | 0.134990 | -0.087407 | 0.309223 | 1.000000 |

- 目標變數為price，觀察其他三個變數
- 發現horsepower和width的相關性為0.6，稍微偏高
- 相關性較高的變數同時作為多元線性迴歸的解釋變數，可能發生「多元共線性(multi-collinearity)」

# 多元共線性

- 多元共線性是由於變數間的高相關性，迴歸係數的變異數變大，失去了係數的顯著性。
- 這樣的現象應該避免，所以建構多元線性迴歸的模型時，通常只挑出能代表高相關性變數群的變數來用於模型。
- 不過因為這裡是實驗，並未嚴密考慮，所以接下來還是先將width, horsepower兩者和height一起留下來建構模型。

# 引入函式

- 使用scikit-learn的model_selection模組的train_test_split函式，將資料分為訓練資料與測試資料

```
# 為了資料分割(訓練資料與測試資料)的匯入
from sklearn.model_selection import train_test_split

# 為了多元線性迴歸模型建構的導入
from sklearn.linear_model import LinearRegression
```

# 指定目標變數與解釋變數

- 指定目標變數為price、其他為解釋變數

```
# 指定目標變數為price、其他為解釋變數
X = auto.drop('price', axis=1)
y = auto['price']
```

# 分為訓練資料與測試資料

- train_test_split將資料分為訓練資料與測試資料
- 以何種比例來分取決於test_size，這裡將test_size比例設為0.5，資料會分一半
- 若是設成0.4，則是test data: training data=4:6
  - random_state用以控制亂數的生成
  - 將random_state固定(設為0)， 則是無論執行幾次都能一樣地分離資料
  - 如果不指定為任意的值，每次執行時某列不一定會被歸類於訓練資料或是測試資料，無法得到相同的結果
  - 因此模型效能的驗證時，會將random_state固定

```
# 分為訓練資料與測試資料
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
```

# 建構多元線性迴歸模型

- 使用LinearRegression類別來進行
- 以「 model = LinearRegression() 」產生物件
- 將訓練資料以「 model.fit(X_train,y_train) 」的方式讀取，完成學習

```
# 多元線性迴歸的初始化學習
model = LinearRegression()
model.fit(X_train,y_train)
```

# 決定係數與相關係數確認

- 學習好之後，可將決定係數與相關係數以截距確認
- 決定係數是用來表示對於目標變數來說，預測值與實際的目標變數之值有多接近
- 決定係數可用score方法取得

```
# 顯示決定係數
print('決定係數(train):{:.3f}'.format(model.score(X_train,y_train)))
print('決定係數(test):{:.3f}'.format(model.score(X_test,y_test)))

# 顯示迴歸係數與截距
print('\n迴歸係數\n{}'.format(pd.Series(model.coef_, index=X.columns)))
print('截距: {:.3f}'.format(model.intercept_))
```

# 過度學習 (Overfitting)

- 機器學習的目的是獲得高度泛用性，也就是建構出來的模型，對於未知的資料也能正確地預測。
- 儘管追求對於訓練資料的吻合度看似能得到較好的模型，實際上並非如此，經常出現對於訓練資料準確度很高，但對於測試資料準確度卻降低的情形。
- 此為過度學習或是過擬合(Overfitting)。
- 建構模型時需要特別注意是否有此現象。

# 多元線性迴歸結果

- Train為0.733
- Test為0.737
- 訓練資料與測試資料的分數很接近，可判斷此模型並沒有陷入過度學習的情形

```
決定係數(train):0.733
決定係數(test):0.737

迴歸係數
horsepower       81.651078
width          1829.174506
height          229.510077
dtype: float64
截距: -128409.046
```