國 立 清 華 大 學
工業工程與工程管理學系

智慧化企業整合

Intelligent Integration of Enterprise

Project 2

駕駛安全風險評估預測

第 2 組

111034540 陳柔卉

111034552 陳政翔

111034519 李泓逸

111034545 簡楷峰

指導教授：邱銘傳 博士

中華民國 112 年 6 月

目錄

圖目錄

表目錄

一、 背景介紹與問題分析

1. 背景介紹

現今社會中，人們利用汽機車或是大眾運輸上下班已成常態，然據交通部統計，每月平均車禍事故總件數高達一萬件以上，因此社會對於交通道路安全的意識逐漸高漲，包含在今年 5 月行政院通過的行人優先交通安全行動綱領等法規都是對於此議題重視的體現。

雖具有相關法規的制定，但對於駕駛人的要求卻不盡完善，尤其在於職業駕駛如計程車、公車司機或是客運司機等等，承擔了整車乘客的安全風險，在出勤前，僅有簡單的體溫及酒精濃度測試，但對於職業駕駛人的淺在風險沒有相關的量化方式。

基於此問題，本組發想了「駕駛安全風險評估」系統，根據 Kaggle 上心血管疾病預測資料集，透過機器學習的方式對駕駛員進行預測，並配合每月一次的血液健康檢查，在網頁上輸入對應的健康資訊，來建議該駕駛員是否適合出勤，以此期望能降低事故發生的可能性，使乘客的安全更具有保障。

2. 問題分析(5WIH)

根據此議題，首先進行 5WIH 問題分析。

表 1. 5W1H 表

Who	● 客運司機
When	● 每月一次
Why	● 提供多樣的評估指標 ● 保障乘客乘車安全
What	● 駕駛員風險評估
Where	● 客運出發地
How	● 網站結合機器學習

二、 方法介紹

1. 方法 1：Logistic Regression 邏輯迴歸

邏輯回歸是一種監督式機器學習演算法，用於二元分類任務。它被廣泛用於根據一組輸入變數或特徵預測二元結果。儘管名為邏輯回歸，但它實際上是一種分類算法，而非迴歸算法。邏輯回歸的目標是基於輸入變數估計二元結果發生的概率。它通過將一個邏輯函數（也稱為 sigmoid 函數）擬合到訓練數據上來實現這一目標。邏輯函數將任何實數映射到 0 到 1 之間的值，可以解釋為概率。邏輯回歸模型計算輸入特徵的加權和，並將邏輯函數應用於該和以獲得預測的概率。與每個輸入特徵相關的權重或係數在訓練過程中通過最大概然估計等優化技術學習。

以下是邏輯回歸中常見的超參數：

- C 正則化參數：
  它控制對模型應用的正則化程度。正則化通過在損失函數中添加一個懲罰項來幫助防止過擬合。較高的 C 值減少了正則化的程度，使模型更貼近訓練數據。
- Penalty 懲罰類型：
  邏輯回歸可以使用不同的正則化技術，如 L1 正則化（Lasso）或 L2 正則化（Ridge）。L1 正則化鼓勵模型權重的稀疏性，從而進行特徵選擇，而 L2 正則化將權重收縮到零附近。
- solver 優化算法：
  優化算法決定用於找到邏輯回歸模型的最佳權重的優化算法。不同的求解器具有不同的優點和計算效率。常見的選項包括"lbfgs"（BFGS 限制記憶體算法）、"liblinear"和"newton-cg"。
- max_iter 最大迭代次數：
  該超參數確定求解器應該運行的最大迭代次數或時期數，以找到最佳權重。如果求解器在此限制內無法收斂，它將停止並返回當前的權重。
- class_weight 類別權重：
  在不平衡的數據集中，其中一個類別的示例數量超過另一個類別，為每個類別分配不同的權重有助於在訓練過程中更重視少數類別。

2. 方法 2：K Neighbors Classifier

KNeighbors 分類器是一種用於分類任務的監督式機器學習演算法。它是一種基於實例或記憶的學習演算法，意味著它根據新數據點與訓練數據的相似性進行預測。該演算法通過在特徵空間中找到給定數據點的 k 個最近鄰居，並根據這些鄰居的多數投票來分配類別標籤。k 的值是一個超參數，用於確定進行分類時要考慮的鄰居數量。

以下是 KNeighbors 分類器的主要超參數：

- n_neighbors 鄰居數量 (k)：
  它定義了在進行預測時要考慮的最近鄰居數量。較高的 k 值考慮了更多的鄰居，可能會得到更平滑的決策邊界，但也增加了計算複雜性。相反，較低的 k 值可能導致更具局部敏感性的決策邊界。
- distance_metrics 距離度量：
  該超參數確定用於計算數據點之間相似度的距離度量。常見的選項包括歐式距離、曼哈頓距離和明可夫斯基距離。距離度量的選擇取決於數據的性質和問題的特點。
- weights 加權方案：
  KNeighbors 分類器允許在投票過程中為鄰居分配權重。每個鄰居的權重可以基於其與查詢點的距離或相似度來決定。常見的加權方案包括均勻權重（所有鄰居具有相等權重）和基於距離的權重（較近的鄰居具有較高權重）。
- algorithm 演算法：
  該參數指定用於計算最近鄰居的演算法。可用的選項包括 'auto'、'ball_tree'、'kd_tree' 和 'brute'。'auto' 選項將根據輸入數據自動選擇最適合的演算法。
- leaf_size 葉節點大小：
  此參數僅適用於 'ball_tree' 或 'kd_tree' 演算法。它控制樹結構中葉節點的大小，影響演算法的速度和記憶體使用。

3. 方法 3：Support Vector Classification 支援向量分類

支援向量分類是一種用於分類任務的監督式機器學習演算法。它基於支援向量機（Support Vector Machine，SVM）方法，通常用於二元分類問題，也可以擴展到多類別分類。支援向量分類的目標是找到一個最佳的超平面，能夠將不同類別的數據點有效地分隔開來。該超平面在特徵空間中具有最大的間隔，稱為最大間隔分類器。支持向量指的是位於超平面上的訓練數據點，它們對於確定超平面的位置和形狀起著關鍵作用。

以下是支援向量分類器的主要超參數：

- C 正則化參數：
  C 是一個正則化參數，控制分類器的容忍度。較小的 C 值表示容忍更多的分類錯誤，可能導致更大的間隔，但也可能使分類器更具泛化能力。較大的 C 值將迫使分類器盡量正確分類每個訓練樣本，但可能會導致過度擬合。
- kernel 核函數：
  kernel 參數指定在特徵空間中計算兩個數據點之間距離的方法。常用的 kernel 函數包括線性核函數、多項式核函數、高斯（RBF）核函數等。選擇適當的 kernel 函數取決於數據的性質和分類問題的特點。
- gamma：
  gamma 參數用於控制核函數的影響範圍。較大的 gamma 值表示核函數具有較小的影響範圍，使得分類器更加關注鄰近的訓練數據點。較小的 gamma 值使得核函數具有較大的影響範圍，使分類器更加平滑。
- class_weight 權重分配：
  class_weight 參數用於處理類別不平衡的情況。通常在類別樣本數量差異較大時使用，它可以為不同的類別指定不同的權重，以平衡不同類別的重要性。

4. 方法 4：Random Forest 隨機森林

隨機森林是一種集成學習方法，用於解決分類和回歸問題。它由多個決策樹組成，每個決策樹都進行獨立訓練並投票或平均化來做出最終預測。

隨機森林的工作原理如下：

(1). 從訓練數據中隨機選擇一部分樣本（bootstrap 抽樣），形成一個被稱為訓練集的子集。

(2). 使用這個子集訓練一個決策樹模型。在每個節點上，從所有特徵中隨機選擇一個子集進行分割。

(3). 重複步驟 1 和 2 多次，形成多個獨立的決策樹。

(4). 對於分類問題，隨機森林根據所有決策樹的預測結果進行投票，選擇得票最多的類別作為最終預測結果。對於回歸問題，隨機森林取多個決策樹預測值的平均值。

以下是隨機森林的主要超參數：

- n_estimators 指定要構建的決策樹的數量：
  較大的數值可以提升模型性能，但同時也增加了計算成本。
- max_depth 決策樹的最大深度：
  限制決策樹的增長，以避免過度擬合。較小的深度通常有助於簡化模型並提高泛化能力。
- min_samples_split 節點分割的最小樣本數：
  當節點上的樣本數小於此值時，停止進一步分割。較大的值有助於防止過度擬合。
- min_samples_leaf 葉子節點的最小樣本數：
  當葉子節點上的樣本數小於此值時，停止分割。與 min_samples_split 一樣，它也是用於控制模型的複雜度和防止過度擬合。
- max_features 特徵數量：
  每個決策樹在分割節點時考慮的特徵數量

三、 模型訓練與績效

1. 資料集介紹

心血管疾病（CVDs）是全球首要的死因，每年估計造成 1,790 萬人死亡，佔全球死亡人數的 31%。

心力衰竭是由心血管疾病引起的常見病例，本資料集包含 12 個特徵，可用於預測心力衰竭的死亡率。

大多數心血管疾病可以通過針對行為風險因素進行預防，如使用煙草、不健康的飲食和肥胖、缺乏體力活動以及有害酗酒，利用全民策略來解決。

患有心血管疾病或具有高心血管風險（由於存在高血壓、糖尿病、高血脂症或已確診疾病等一個或多個風險因素）的人需要早期檢測和管理，機器學習模型可以提供很大的幫助。

表 2. 資料集欄位表

名稱	說明
age	Age
anaemia	Decrease of red blood cells or hemoglobin (boolean)
creatinine_phosphokinase	Level of the CPK enzyme in the blood (mcg/L)
diabetes	If the patient has diabetes (boolean)
ejection_fraction	Percentage of blood leaving the heart at each contraction (percentage)
high_blood_pressure	If the patient has hypertension (boolean)
platelets	Platelets in the blood (kiloplatelets/mL)
serum_creatinine	Level of serum creatinine in the blood (mg/dL)
serum_sodium	Level of serum sodium in the blood (mEq/L)
sex	Woman or man (binary)
smoking	If the patient smokes or not (boolean)
time	Follow-up period (days)
DEATH_EVENT	If the patient deceased during the follow-up period (boolean)

2. 資料前處理

首先，我們先確認這筆資料集是否有缺失值。

```
[ ]  # 檢查是否有缺失值
     df.isnull().sum()
```

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
time                        0
DEATH_EVENT                 0
dtype: int64
```

圖 1. 確認資料集缺失情況

確認無缺失值之後，了解整體資料各個欄位的屬性如何。

```
# 了解整體資料狀況
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 299 entries, 0 to 298
Data columns (total 13 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   age                       299 non-null    float64
 1   anaemia                   299 non-null    int64
 2   creatinine_phosphokinase  299 non-null    int64
 3   diabetes                  299 non-null    int64
 4   ejection_fraction         299 non-null    int64
 5   high_blood_pressure       299 non-null    int64
 6   platelets                 299 non-null    float64
 7   serum_creatinine          299 non-null    float64
 8   serum_sodium              299 non-null    int64
 9   sex                       299 non-null    int64
 10  smoking                   299 non-null    int64
 11  time                      299 non-null    int64
 12  DEATH_EVENT               299 non-null    int64
dtypes: float64(3), int64(10)
memory usage: 30.5 KB
```

圖 2. 確認資料集整體情況

接著，我們再由各個變數之間的相互關係去了解各變數對於 DEATH_EVENT 的影響狀況為何。

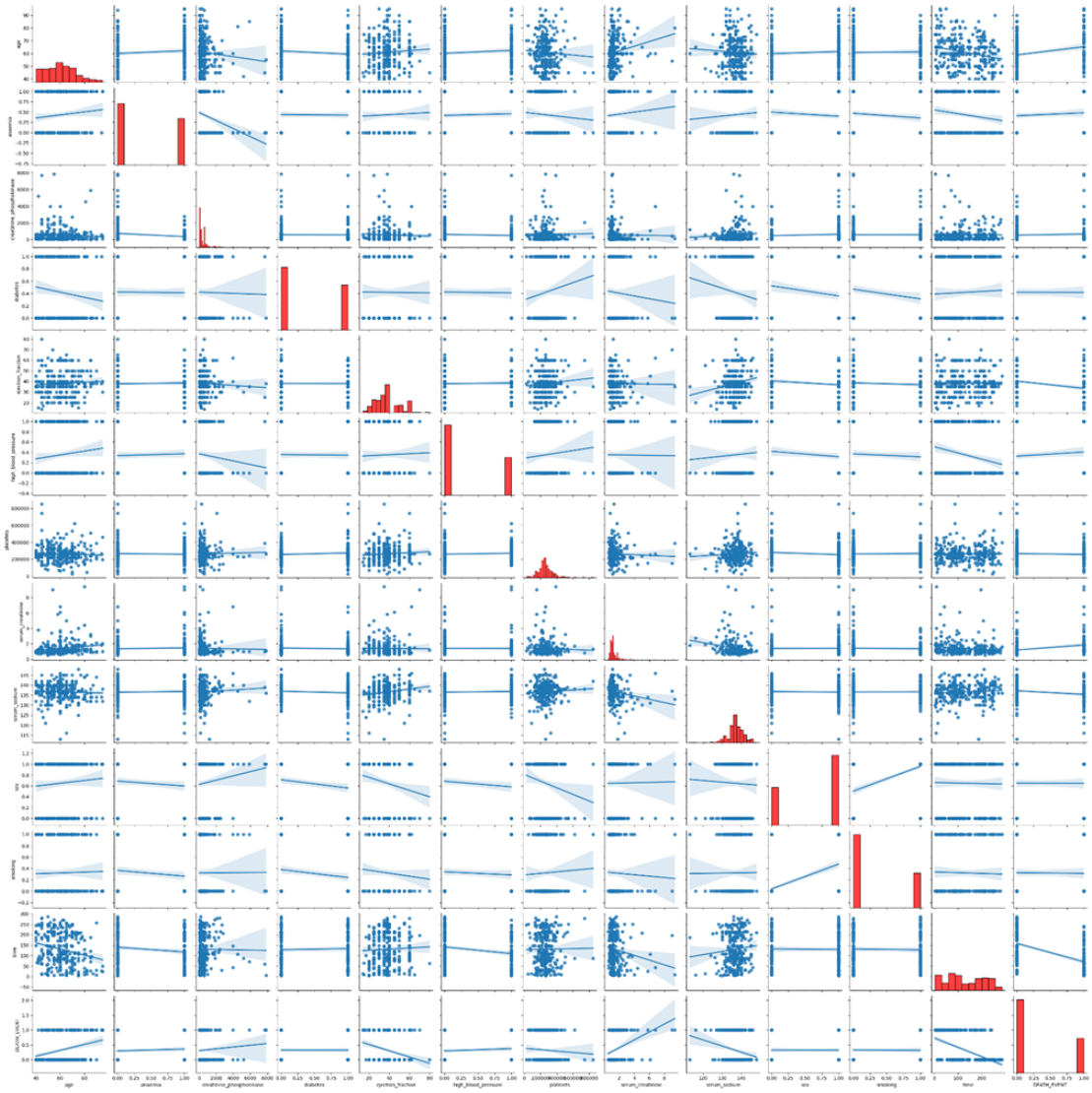

圖 3. 變數間相互關係資料散布圖

也去確認每個變數的直方圖為何。

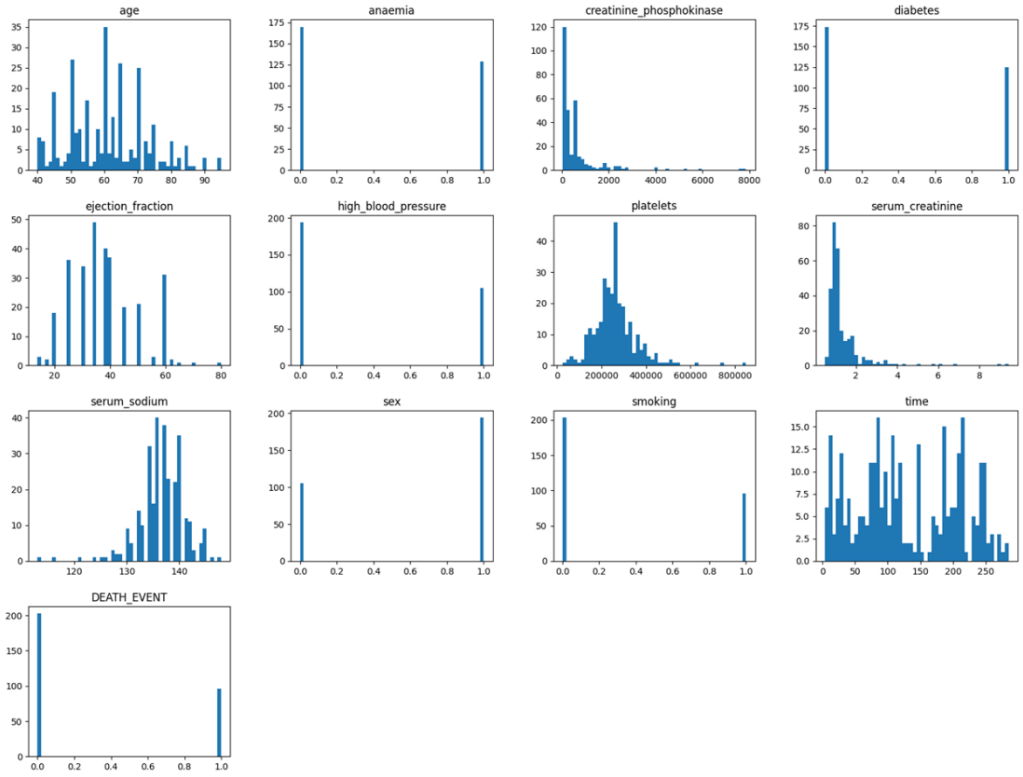

圖 4. 每個變數的直方圖

接著，再以各變數間的相關係數圖(圖 5)確認是否有與 DEATH_EVENT 強列相關的變數。

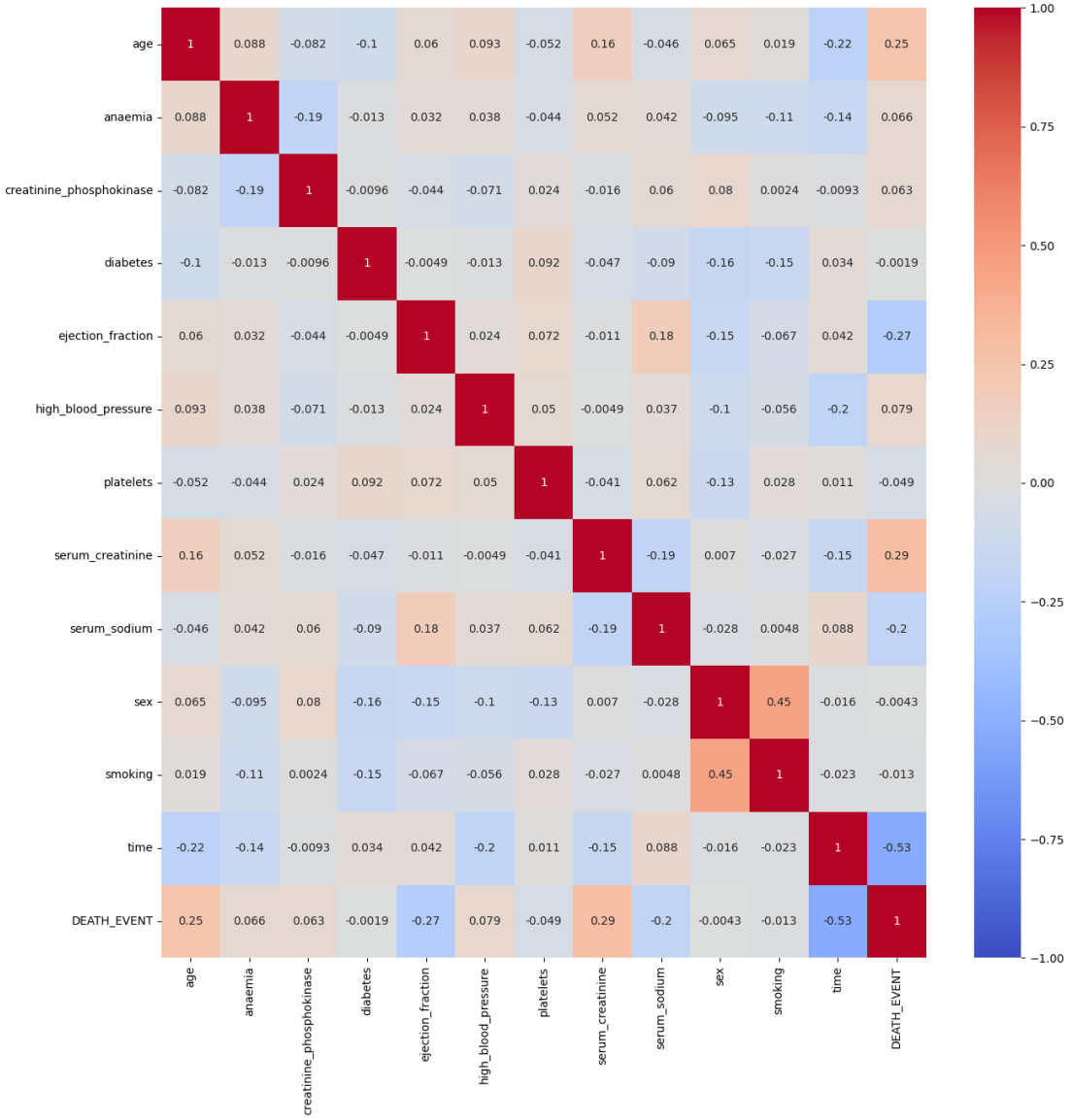

圖 5. 各變數間的相關係數圖

以圖 6 的數值方式呈現，找出重要的變數為：age, ejection_fraction, serum_creatinine, serum_sodium, time 共五個重要變數，我們再以試誤法的方式，找出 age 對正確性助益不大，不將 age 變數納入考量。

最後，將資料集裡的資料做正規化，並區分出測試集與驗證集。

```
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
#sc = preprocessing.MinMaxScaler()
sc = preprocessing.StandardScaler()
x_scale=sc.fit_transform(x)

x_train,x_test,y_train,y_test=train_test_split(x_scale,y,test_size=0.3,random_state=56)
```

圖 6. 以數值方式呈現重要變數的相關係數

3. 訓練與參數優化

我們利用兩種參數優化的方式，分別為：Grid Search 跟 Random Search 兩種方式。Grid Search 方法為將給定的各種參數組合全部試過一次，提供最好的參數組合給使用者。而 Random Search 則為在所有可能的參數組合中隨機挑選一組嘗試。

(1). Logistic Regression 邏輯迴歸

我們利用 Random Search 下去尋找，如圖 7，找到表現優良的參數組合為：solver = 'newton-cg', random_state = 72, penalty = 'l2', max_iter = 100, C = 0.2。並將此參數組合帶入測試集測試，得到正確性為 80%。

```
from sklearn.linear_model import LogisticRegression
import numpy as np
import sklearn
from sklearn.model_selection import RandomizedSearchCV


param_dist = {
        'penalty':['l1','l2'],
        'C': [0.2, 0.5, 0.8, 1.0, 1.5],
        'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
        'max_iter' : [100, 200, 300, 400],
        'random_state':range(0,100,1)
        }
clf = LogisticRegression(random_state = 1)

LR_random = RandomizedSearchCV(estimator = clf, param_distributions = param_dist,
                               n_iter=100, cv=3, verbose=2, random_state=42, n_jobs=-1)

LR_random.fit(x_train,y_train)
LR_random.best_params_
```

圖 7. 訓練邏輯迴歸的程式碼

```
from sklearn.linear_model import LogisticRegression

# 建立Logistic模型
clf = LogisticRegression(solver='newton-cg', penalty= 'l2', max_iter= 200, C = 0.2, random_state=72)
# 使用訓練資料訓練模型
clf.fit(x_train, y_train)
# 使用訓練資料預測分類
y_pred = clf.predict(x_train)

clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
```

```
Accuracy: 80.0
```

圖 8. 將選定的邏輯迴歸參數組合代入測試集驗證

(2). K Neighbors Classifier

首先，以 KNeighbors 的預設參數進行模型訓練，觀察其準確度。

```
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
print(classification_report(y_test, y_pred))
```

```
Accuracy: 85.0
```

圖 9. KNeighbors 預設參數訓練

其次，進行 GridSearchCV 得到最佳超參數。

```
k_range = list(range(1, 31))
param_grid = dict(n_neighbors=k_range, weights=['uniform', 'distance'], algorithm=['auto', 'ball_tree', 'kd_tree', 'brute'])

# defining parameter range
grid = GridSearchCV(KNeighborsClassifier(), param_grid, cv=10, scoring='accuracy', return_train_score=False,verbose=1)
# fitting the model for grid search
grid_search=grid.fit(x_train, y_train)
# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
Fitting 10 folds for each of 240 candidates, totalling 2400 fits
{'algorithm': 'auto', 'n_neighbors': 6, 'weights': 'uniform'}
KNeighborsClassifier(n_neighbors=6)
```

圖 10. KNeighbors 參數優化

最終確認準確度。

```
grid_predictions = grid.predict(x_test)

accuracy = accuracy_score(y_test, grid_predictions)
print("Accuracy:", accuracy*100)
# print classification report
print(classification_report(y_test, grid_predictions))
```

```
Accuracy: 88.33333333333333
```

圖 11. KNeighbors 參數優化後準確度

結果：用參數優化使 KNeighbors 模型準確度提升，將以調整參數後訓練的模型進行最後比較。

(3). Support Vector Classification 支援向量分類

首先，以 SVC 的預設參數進行模型訓練，觀察其準確度。

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=45)
clf = svm.SVC()

clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)

k = clf._gamma
print(k)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
print(classification_report(y_test, y_pred))
```

```
4.9978391596233096e-05
Accuracy: 90.0
```

圖 12. SVC 預設參數訓練

其次，進行 GridSearchCV 得到最佳超參數。

```
# defining parameter range
param_grid = {'C': [0.1, 1, 10, 100, 1000],
              'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'kernel': ['linear', 'rbf', 'sigmoid']}

grid = GridSearchCV(svm.SVC(), param_grid, refit = True, verbose = 3)

# fitting the model for grid search
grid.fit(x_train, y_train)

# print best parameter after tuning
print(grid.best_params_)

# print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'C': 10, 'gamma': 0.0001, 'kernel': 'rbf'}
SVC(C=10, gamma=0.0001)
```

圖 13. SVC 參數優化

最終確認準確度。

```
grid_predictions = grid.predict(x_test)

accuracy = accuracy_score(y_test, grid_predictions)
print("Accuracy:", accuracy*100)
# print classification report
print(classification_report(y_test, grid_predictions))
```

```
Accuracy: 88.33333333333333
```

圖 14. SVC 參數優化後準確度

結果：使用參數優化並未使 SCV 模型準確度提升，將以預設模型進行最後比較。

(4). Random Forest 隨機森林

我們利用 Random Search 下去尋找，如圖 X，找到表現優良的參數組合為：random_state = 91, n_estimators = 117, min_samples_split = 5, min_samples_leaf = 10, max_features = 'sqrt', max_depth = 17, bootstrap = True，並將此參數組合代入測試集驗證，得到正確性為 93.3%。

```
from sklearn.ensemble import RandomForestClassifier
import numpy as np
import sklearn
from sklearn.model_selection import RandomizedSearchCV

param_dist = {
        'n_estimators':range(20,200,1),
        'max_features': ['auto', 'sqrt'],
        'max_depth':range(1,20,1),
        'min_samples_split': range(1,20,1),
        'min_samples_leaf' : range(1,20,1),
        'bootstrap': [True, False],
        'random_state': range(1,100,1)
        }
clf = RandomForestClassifier(n_estimators = 1000, random_state = 1)

rf_random = RandomizedSearchCV(estimator = clf, param_distributions = param_dist,
                               n_iter=100, cv=3, verbose=2, random_state=42, n_jobs=-1)

rf_random.fit(x_train,y_train)
rf_random.best_params_
```

圖 15. 訓練隨機森林的程式碼

```
{'random_state': 91,
 'n_estimators': 117,
 'min_samples_split': 5,
 'min_samples_leaf': 10,
 'max_features': 'sqrt',
 'max_depth': 17,
 'bootstrap': True}
```

圖 16. 訓練隨機森林所得到的參數組合

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

clf = RandomForestClassifier(bootstrap=True, max_depth=17, max_features='sqrt', min_samples_leaf=10, min_samples_split=5,
                             n_estimators=117, random_state = 91)


clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)


accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy*100)
```

```
Accuracy: 93.33333333333333
```

圖 17. 將選定的隨機森林參數組合代入測試集驗證

4. 方法績效比較

綜合以上第三章的四個模型訓練結果，我們整理出表 X，發現隨機森林模型的正確性高達 93.3%，是這四個模型裡表現最好的，因此挑選此訓練完成的模型結合後續在第四章所欲介紹的網頁內容裡。

表 3. 各種模型表現比較

使用模型	正確性
Logistic Regression	80.0%
K-Neighbors	88.3%
SVM	88.3%
Random Forest	**93.3%**

四、Web

1.Web 架構圖

本網頁架構圖如圖 18 所示，主要分為網站基本功能及會員功能，網站基本功能可提供首頁(最新消息、介紹、未來展望、Chatbot、外部連結)、網路訂票、乘車資訊(票價、座位、路線時刻)、會員登入及乘客服務(Q&A、意見聯絡)之相關功能；會員功能包括加入會員、密碼找回、修改刪除會員資料、確認訂單、修改刪除訂單。

另外，透過會員登入管理員帳號能進入後台系統，有訂單管理、會員管理、意見管理以及駕駛安全風險評估四大功能區塊，前三者可以查看現有的資訊，直接對資料庫進行修改刪除(意見管理僅刪除功能)，後者可進行駕駛員出勤前的風險評估。

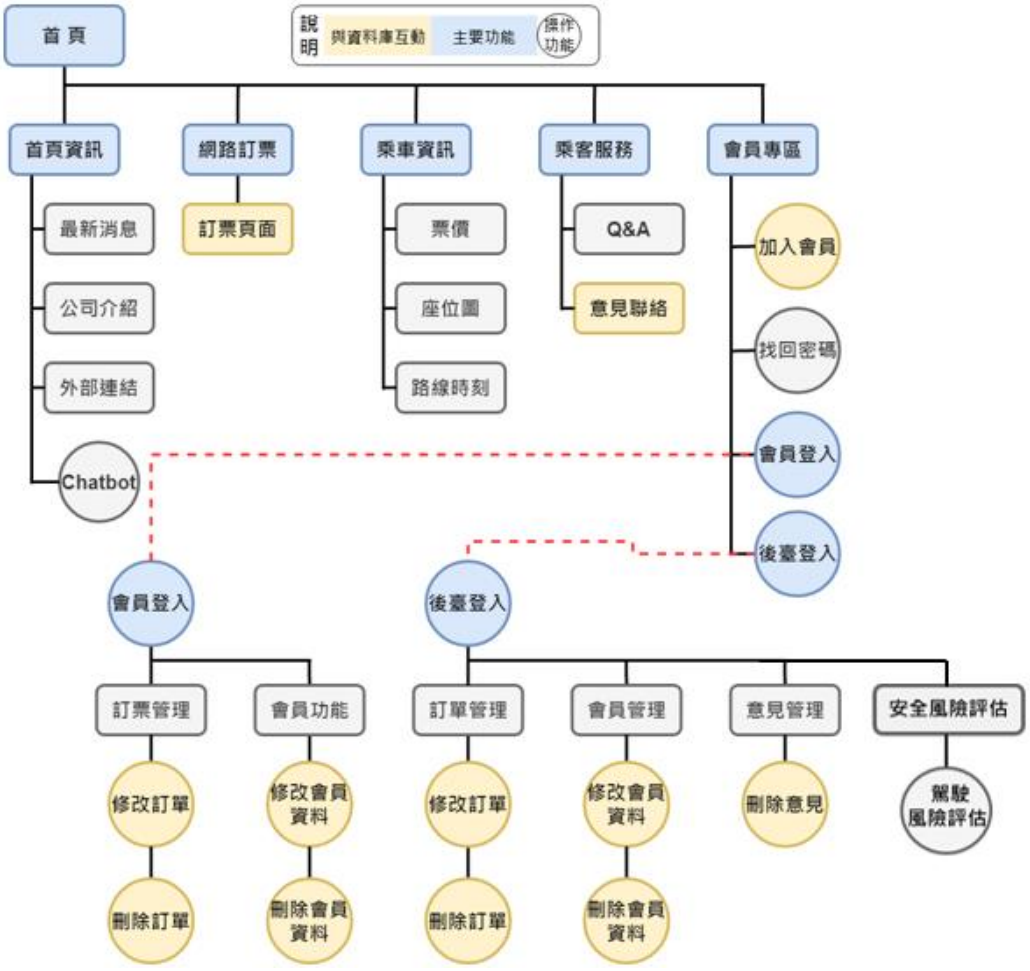

圖 18. 網站架構圖

2.ER-model

本組利用 PhpMySql 作為後端資料庫，並建立三個資料表儲存對應之資料，分別為會員資料表(member)、訂單資料表(order)，和與我們聯絡表(contact_us)，其 E-R model 如圖 19 所示。

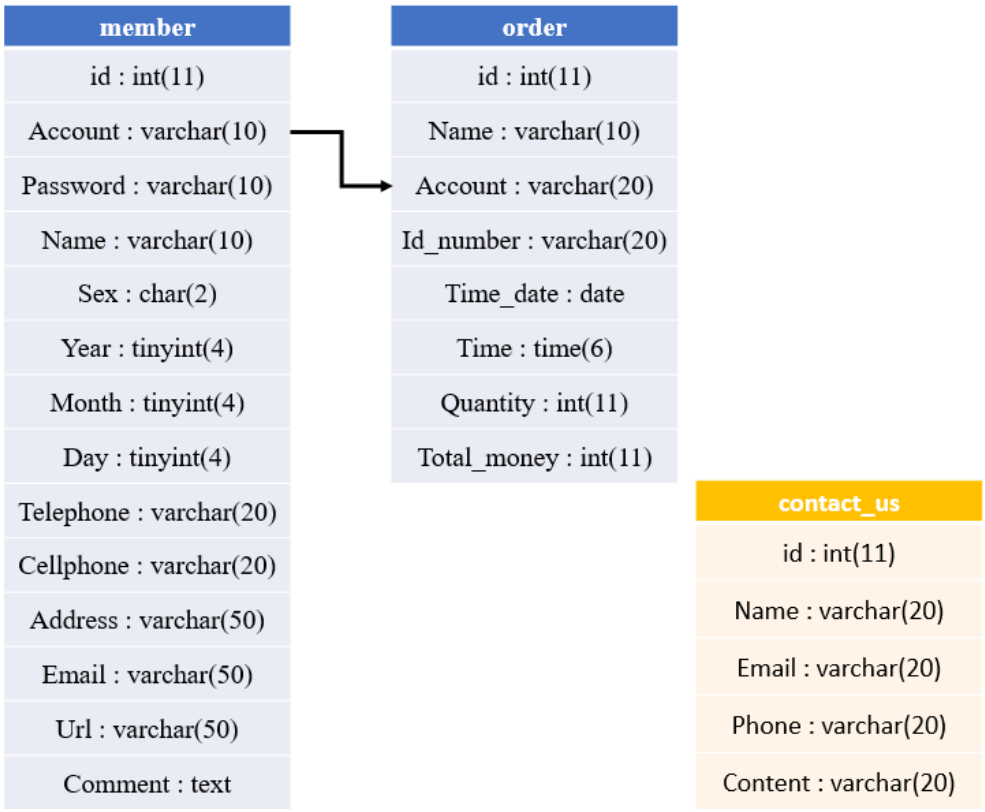

圖 19. E-R model

3.網站功能介紹

3.1 網路訂票

乘客填寫個人資訊並選擇對應之車票與數量(如圖 20 所示)。完成訂票後，會跳至完成訂票之頁面(如圖 21)，並在畫面中顯示訂購人之名字與本筆訂單之總金額。

圖 20. 網路訂票介面

圖 21.完成訂票頁面

3.2 會員專區

網站中設有會員專區，再會員專區的介面中可以進行會員登錄、加入會員和找回密碼。

圖 22.會員專區介面

圖 23.加入會員表單

圖 24.找回密碼功能

進行會員登入後，即可使用會員特殊功能，包括帳號管理和訂票訂單管理。

圖 25.帳號管理訂票訂單與功能

圖 26.改會員資料

圖 27.刪除會員資料

圖 28.訂票資訊修改與刪除

3.3 後臺管理

在會員專區登入管理員帳號後，即可進入後台管理系統，有訂單管理、會員管理、意見管理以及 3.4 節的駕駛安全風險評估四大功能區塊，前三者可以查看現有的資訊，並直接對資料庫進行修改刪除(意見管理僅刪除功能)。

圖 29. 後臺訂單管理

圖 30.後臺訂單修改刪除

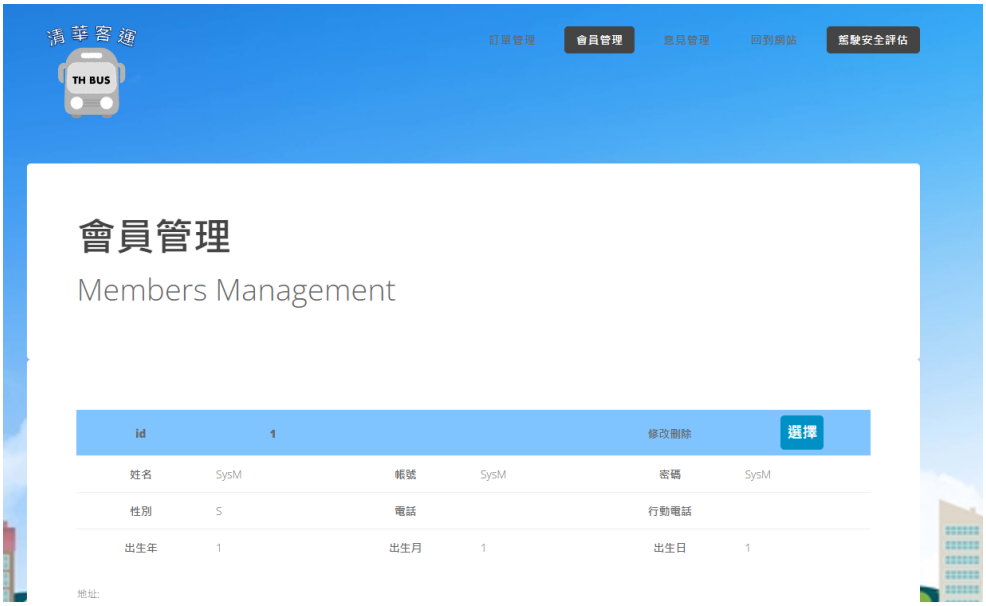

圖 31.後臺會員管理

圖 32.後臺會員修改刪除

圖 33.後臺意見管理

3.4 網站執行 AI -駕駛安全風險評估

本次 project 結合網站與 AI，讓使用者可以在網站上輸入駕駛員相關資料，透過隨機森林模型即可在網站上顯示安全風險評估預測結果，供管理者排班時的參考依據。

承 3.3，當管理者進入後台管理後，在導航欄中點選「駕駛安全風險評估」，即可進入駕駛安全風險評估頁面。

圖 34.駕駛安全風險評估

而本次 project 是使用 Flask 來部署機器學習模型，Flask 是一個輕量級的 Python 語言編寫之 Web 開發框架，使用者可以透過 POST 協定從前端網頁發送駕駛員資訊(如年齡、隨訪時間等)。後端程式收到數值後送給事先打包好的模型，並將模型預測結果透過 JSON 格式回傳到前端使用者。此部分架構主要有 model.py、HTML/CSS 以及 app.py，以下將分別描述。

- model.py

  此檔案詳細內容在三、模型訓練與績效有介紹過，主要為機器學習模型的程式碼，而訓練好的模型會利用 Python 的 pickle 模組來保存為一個 model.pkl 檔案(如圖 35)。

```
##  模型參數存檔
import  pickle
pickle.dump(clf,  open('modle.pkl','wb'))
```

圖 35.模型保存

- HTML/CSS

  此部分主要為在.html 檔案中加入<form action="{{ url_for('predict')}}" method="post" name="myform"> 以及呈現預測結果的區域 {{ prediction_display_area }}，此方法主要是使用 POST 請求讓網頁可以在接收到使用者輸入的資料後，提交表單，得到風險評估預測結果。(如圖 36)

圖 36.HTML 程式碼(紅框區)

- app.py

  此部分為建立 Flask API，首先要先載入前述中所提及之訓練好的模型(model.pkl)，接著建立一個 GET 的 route @app.route('/')，去呼叫已經傳寫好的 HTML 檔案。再來是建立第二個 route，路徑設定為 /predict 以及 HTTP Requesst 的方法指定 methods=['POST']，主要用途是接收使用者輸入的數值，並將數值放到 list 中，用前面所載入的模型進行預測，並將預測的結果透過 JSON 格式回應給前端使用者。若預測結果為 1，代表高風險駕駛，目前身體狀態不適合駕駛；若預測結果為 0，則屬於低風險駕駛，目前身體狀態良好，適合駕駛。圖 37 為 app.py 程式碼。

```python
import numpy as np
from flask import Flask, request, jsonify, render_template
import pickle

app = Flask(__name__)
model = pickle.load(open('model.pkl','rb'))

@app.route('/')
def home():
    return render_template('safety_prediction_test1.html')

@app.route('/predict', methods=['POST'])
def predict():
    features_list = []

    ejection_fraction = int(request.form['ejection_fraction'])
    serum_creatinine = float(request.form['serum_creatinine'])
    serum_sodium = int(request.form['serum_sodium'])
    time = int(request.form['time'])


    features_list.extend([
        ejection_fraction, serum_creatinine, serum_sodium, time
    ])

    features = np.array(features_list).reshape(1,-1)
    predict_outcome_list = model.predict(features)
    predict_outcome = predict_outcome_list[0]

    if predict_outcome == 1:
        prediction_text = "1-->屬於高風險駕駛，目前身體狀態不適合駕駛"
    else:
        prediction_text = "0-->屬於低風險駕駛，目前身體狀態良好，適合駕駛"

    return render_template('safety_prediction_test1.html', prediction_display_area='  風險評估：{}'.format(prediction_text), prediction_text_color='color: black;


if __name__ == "__main__":
    app.run(debug = True)
```

圖 37.app.py 程式碼

說明完架構後，接著是執行 API，開啟 Python prompt 後，輸入 python app.py 指令即可。(如圖 38)

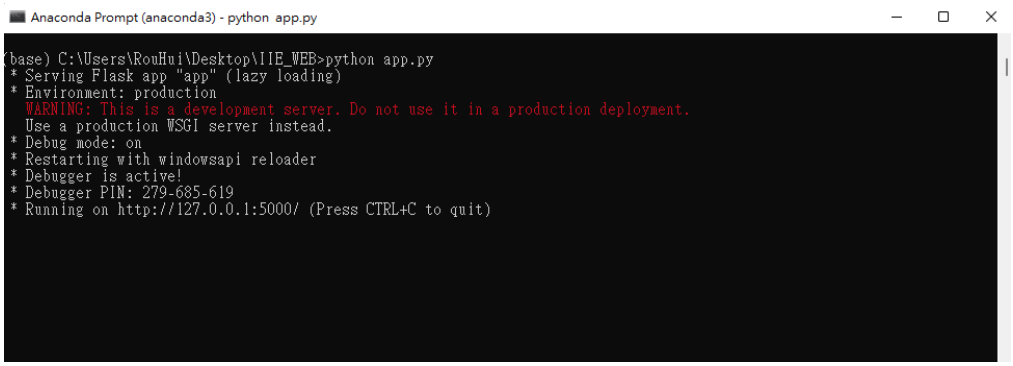

圖 38.執行 API

執行完後會產生一串網址 http://127.0.0.1:5000/，此網址只能在本機端運行，透過瀏覽器訪問該網址，在駕駛員資訊中輸入要預測的相關資訊，接著點擊風險預估按鈕後，就會出現風險預測結果。如圖 39 所示。

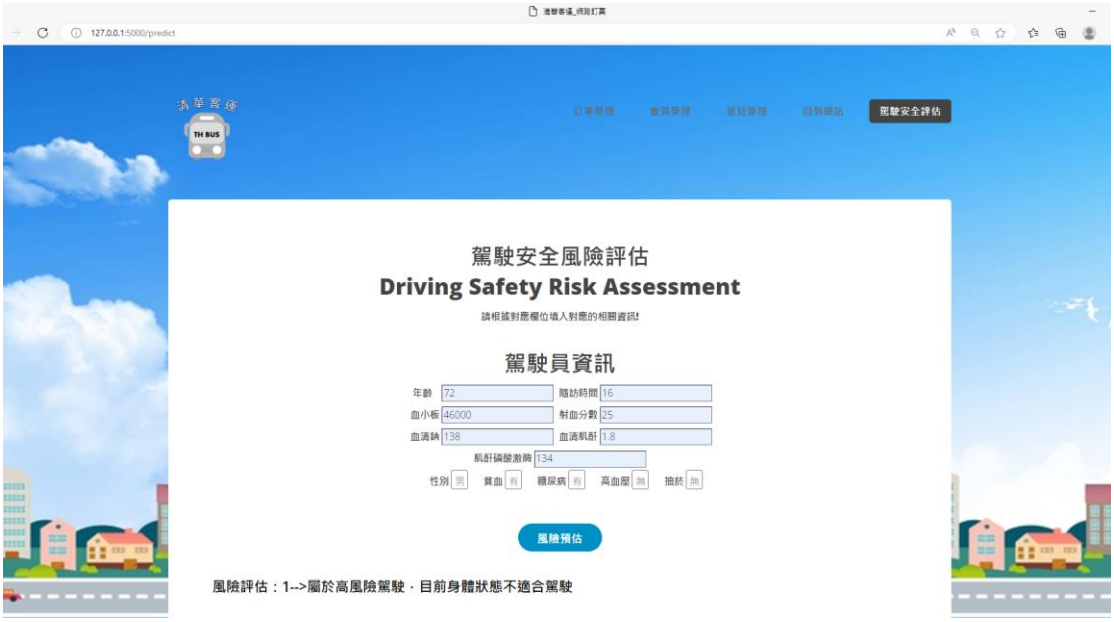

圖 39.駕駛安全風險評估網站

五、結論與未來展望

1.整體貢獻

透過網頁結合機器學習發想出的「駕駛安全風險評估」系統，在嘗試不同的模型與超參數組合後，最終採用隨機森林，且在準確率上可高達93%，駕駛員在頁面上輸入對應的欄位資訊，系統就可對該司機提出是否建議出勤的提示，提供駕駛員在出勤前對自身狀況的另一種評估方式，期望能透過此系統的建立，提升社會大眾對於交通安全的重視，營造安全的交通環境。

2.未來展望

目前網頁上所需輸入的資料欄位有些取得不易，未來若有更多相關的資料蒐集，替換所需輸入的資訊，將可對駕駛安全風險評估系統的便利性更為提升，此外，未來也可增加記錄該司機過往的輸入資料並提供相關的健康注意事項，將此系統的應用層面更為周全，不只是對駕駛員更是對乘客的一個保障。

六、參考資料

1. 使用 Flask 部署機器學習模型 https://zhuanlan.zhihu.com/p/363303013
2. How to Easily Deploy Machine Learning Models Using Flask https://towardsdatascience.com/how-to-easily-deploy-machine-learning-models-using-flask-b95af8fe34d4
3. Grid Search 與 Random Search 的差別 https://ithelp.ithome.com.tw/articles/10275842