國立中央大學
商業智慧期末報告

銀行貸款違約情況預測

壹、商業理解

一、商業背景分析

銀行的主要利潤來自對外放款，但通常也伴隨著風險，尤其是近期的新冠肺炎疫情導致，貸款人違約的機會提升，為了有效緩解此種問題，許多銀行打算使用機器學習來解決人工分析時容易遇到的盲點，並且藉此補足因貸款人數增加所導致的放款效率下降等問題，銀行收集了過去眾多貸款人的相關資料與信用評等，希望能透過機器學習的模型來分判定客戶是否會出現貸款違約的情形，使銀行能夠有效的減少損失，並提升放款效率。

二、數據分析的目標

本研究主要用於建立銀行新客戶貸款的風險分類模型，用以識別客戶是否可能會為約的機率，並分析出導致違約相關因素中各項因素的重要程度，並藉由分析出的結果決定出該客戶的貸款申請是否通過，同時也能促進銀行對老客戶推銷貸款時的精準度，使銀行業務能夠更有效率的鎖定顧客中潛在的貸款需求者。

貳、資料理解

一、資料集介紹

本研究中所使用的資料集來自 kaggle.com，一個數據建模和數據分析競賽平台，本資料等級為 CC0，公眾領域貢獻宣告，意即任何人可以任何目的自由地以該著作為基礎，從事創作、提升或再使用等行為，而不受著作權或是資料庫相關法律的限制。

資料集網址: https://www.kaggle.com/datasets/yasserh/loan-default-dataset?resource=download

二、資料集欄位介紹

Loan Default Dataset [1]資料集中含有大量由多種決定性因素所組成的資料，如：貸款者收入、性別、貸款目的等…，詳細欄位說明如下表

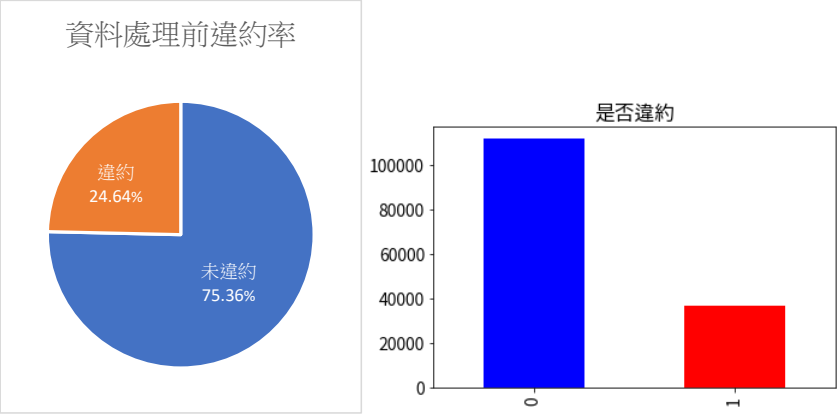
欄位編號	欄位名稱	說明
1	ID	客戶 ID
2	year	年
3	loan_limit	貸款上限
4	Gender	性別
5	approv_in_adv	預先批准
6	loan_type	貸款類型
7	loan_purpose	貸款目的
8	Credit_Worthiness	信用價值等級
9	open_credit	一般信用狀
10	business_or_commercial	是否商用
11	loan_amount	貸款金額
12	rate_of_interest	利率
13	Interest_rate_spread	利率差距
14	Upfront_charges	預付費用
15	term	貸款周期
16	Neg_ammortization	負攤還借款
17	interest_only	利息
18	lump_sum_payment	一次性支付
19	property_value	財產價值
20	construction_type	建築型態
21	occupancy_type	入住類型

欄位編號	欄位名稱	說明
22	Secured_by	擔保方式
23	total_units	總單位數
24	income	收入
25	credit_type	信用類型
26	Credit_Score	信用分數
27	co-applicant_credit_type	共同申請人信用類型
28	age	年齡
29	submission_of_application	申請次要目的
30	LTV	顧客終身價值
31	Region	地區
32	Security_Type	證券型態
33	Status	狀態
34	dtirl	資產負債比

參、資料前處理

一、資料處理說明

本資料中含有遺失值與極值，需先進行資料清洗，將無法使用(包含過多的錯誤值)、遺失值過多或不具分析價值(對於分析不具參考價值或影響力)之欄位優先去除，如年分、id 等…，此後將資料分割為訓練資料集與測試資料集，詳細說明如下：



資料在尚未進行任何處理時的違約情形，約有 24.64%的違約率，

二、資料欄位刪除以及遺失值處理

根據資料來源所描述，我們必須先對資料進行清洗，去除無參考價值與有缺失值的欄位，

(1)首先針對違約資料進行篩選並觀察遺失值分布情況

實做部分如下圖程式碼：

```
#  確認缺失值
df1=df_all[df_all['Status']==1]
print(df1.isnull().sum())  #找出違約資料並觀察遺失值狀況
```

```
ID                          0
year                        0
loan_limit                881
Gender                      0
approv_in_adv           241
loan_type                   0
loan_purpose               35
Credit_Worthiness           0
open_credit                 0
business_or_commercial      0
loan_amount                 0
rate_of_interest        36439
Interest_rate_spread    36639
Upfront_charges         36486
term                       15
Neg_ammortization          32
interest_only               0
lump_sum_payment            0
property_value          15096
construction_type           0
occupancy_type              0
Secured_by                  0
total_units                 0
income                   1239
credit_type                 0
Credit_Score                0
co-applicant_credit_type    0
age                       200
submission_of_application 200
LTV                     15096
Region                      0
Security_Type               0
Status                      0
dtir1                   16310
```

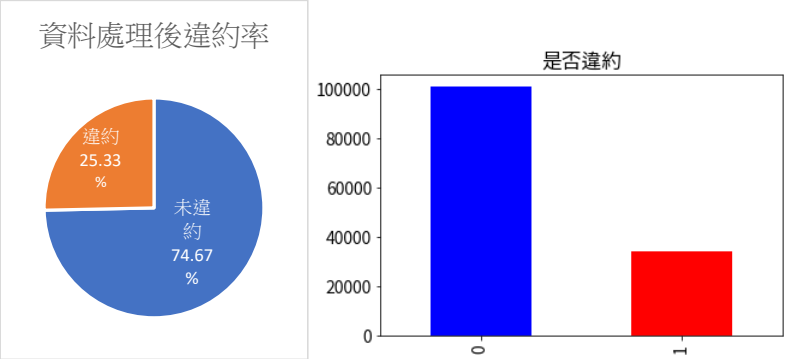
(2) 刪除違約資料中遺失值過多的欄位以及 id、年份這兩個不具分析價值的欄位

```
df_all=df_all.drop(['rate_of_interest','Interest_rate_spread','Upfront_charges','property_value','dtir1','ID','LTV','year'],axis=1)
print(df_all.isnull().sum())  #刪除任一欄位含有遺失值的資料
```

```
df_all=df_all.replace("Unknow",np.nan)
df_all=df_all.replace("Unknow",np.nan).dropna()
print(df_all.isnull().sum())
```

(3)針對清洗後的資料，再次進行分析：

```
#  確認訓練資料的筆數與項目數
print(df_all.shape)
print()

#  確認「違約率」的分佈
print(df_all['Status'].value_counts())
print()

#  違約率
rate  =  df_all['Status'].value_counts()[1]/len(df_all)
print(f'違約率  :  {rate:.4f}')
```



已註解 [E1]: 缺少長條統計圖

經過資料清洗後我們可以得到 25.33%的違約率。

三、將資料進行 One-Hot 編碼

這部份我們將非連續型的資料欄位進行 One-Hot 編碼

```
# 用於對項目進行 One-Hot 編碼之函式
def enc(df, column):
        df_dummy = pd.get_dummies(df[column], prefix=column)
        df = pd.concat([df.drop([column],axis=1),df_dummy],axis=1)
        return df

df_a112 = df_a11.copy()
df_a112 = enc(df_a112, 'loan_limit')
df_a112 = enc(df_a112, 'Gender')
df_a112 = enc(df_a112, 'approv_in_adv')
df_a112 = enc(df_a112, 'loan_type')
df_a112 = enc(df_a112, 'loan_purpose')
df_a112 = enc(df_a112, 'Credit_Worthiness')
df_a112 = enc(df_a112, 'open_credit')
df_a112 = enc(df_a112, 'business_or_commercial')
df_a112 = enc(df_a112, 'age')
df_a112 = enc(df_a112, 'Neg_ammortization')
df_a112 = enc(df_a112, 'interest_only')
df_a112 = enc(df_a112, 'lump_sum_payment')
df_a112 = enc(df_a112, 'construction_type')
df_a112 = enc(df_a112, 'occupancy_type')
df_a112 = enc(df_a112, 'Secured_by')
df_a112 = enc(df_a112, 'total_units')
df_a112 = enc(df_a112, 'credit_type')
df_a112 = enc(df_a112, 'co-applicant_credit_type')
df_a112 = enc(df_a112, 'submission_of_application')
df_a112 = enc(df_a112, 'Region')
df_a112 = enc(df_a112, 'Security_Type')
# 確認結果
display(df_a112.head())
```

產生結果之範例如下圖：

	loan_amount	term	income	Credit_Score	Status	loan_limit_cf	loan_limit_ncf	Gender_Female	Gender_Joint	Gender_Male
0	116500	360.0000	1740.0000	758	1	1	0	0	0	0
1	206500	360.0000	4980.0000	552	1	1	0	0	0	1
2	406500	360.0000	9480.0000	834	0	1	0	0	0	1
3	456500	360.0000	11880.0000	587	0	1	0	0	0	1
4	696500	360.0000	10440.0000	602	0	1	0	0	1	0

四、資料切割

最後我們將資料進行分割，劃分出 60%的訓練資料與 40%的測試資料，並設定 random seed 來確保每次切割結果相同，實作如下圖：

```
# 分割輸入資料與標準答案
x = df_all2.drop('Status', axis=1)
y = df_all2['Status'].values

# 分割訓練資料與驗證資料
# 以訓練資料 60%、驗證資料 40% 的比例分割
test_size = 0.4

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=test_size, random_state=random_seed,
    stratify=y)
```

肆、模型訓練

在此部分，我們使測試了許多不同的演算法，包括邏輯斯迴歸、SVM、決策樹、隨機森林及 XGBoost，並針對不同演算法進行比較，從中將選出最佳的演算法，進行後續處理。

一、邏輯斯回歸

邏輯斯迴歸主要是用來找出目標變數與自變數之間的關係，通常目標變數主要是可以分成兩類的二元類別型變數。相較於傳統迴歸分析方法如果碰到自變數類型是非連續性變數就無法處理，邏輯斯迴歸則可以針對類別型變數使用 odds ratio 來判斷其對於目標變數的影響強度。
邏輯斯迴歸實作過程：

```
#邏輯斯迴歸
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
algorithm = LogisticRegression(random_state=random_seed)
algorithm =algorithm.fit(x_train, y_train)
y_predicted=algorithm.predict(x_test)
print('分類正確率',metrics.accuracy_score(y_test,y_predicted))
print(metrics.classification_report(y_test,y_predicted))
```

訓練後可得到正確率約為 74.67%：

	precision	recall	f1-score	support
0	0.75	1.00	0.85	40353
1	0.00	0.00	0.00	13691
accuracy			0.75	54044
macro avg	0.37	0.50	0.43	54044
weighted avg	0.56	0.75	0.64	54044
分類正確率：0.7466693805047739				

二、SVM

SVM 支援向量機是一種線性分類器，同時也可以解決非線性的分割問題，主要是將在低微度空間無法線性分割的樣本投影到高為度空間，來尋找一個平面可以有效地將這些樣本作切割。

SVM 實作過程：

```
#SVM
from sklearn import metrics
from sklearn.svm import SVC
algorithm = SVC(kernel='rbf',random_state=random_seed)
algorithm=algorithm.fit(x_train,y_train)
y_predicted=algorithm.predict(x_test)
print('分類正確率',metrics.accuracy_score(y_test,y_predicted))
print(metrics.classification_report(y_test,y_predicted))
```

訓練後可得到正確率約為 74.73%：

	precision	recall	f1-score	support
0	0.75	1.00	0.86	40353
1	0.68	0.00	0.01	13691
accuracy			0.75	54044
macro_avg	0.71	0.50	0.43	54044
weighted_avg	0.73	0.75	0.64	54044
分類正確率 0.7472799940788987				

三、決策樹

決策樹是一種樹狀結構的分類器，其中非葉子節點的其他結點表示一個特徵屬性上的測試，而每個分支則代表這個特徵屬性在某個條件判斷後的輸出，並且以遞迴的方式從根節點開始生成判斷條件來產生新的節點直到滿足條件後產生葉子結點並以此做為決策結果。

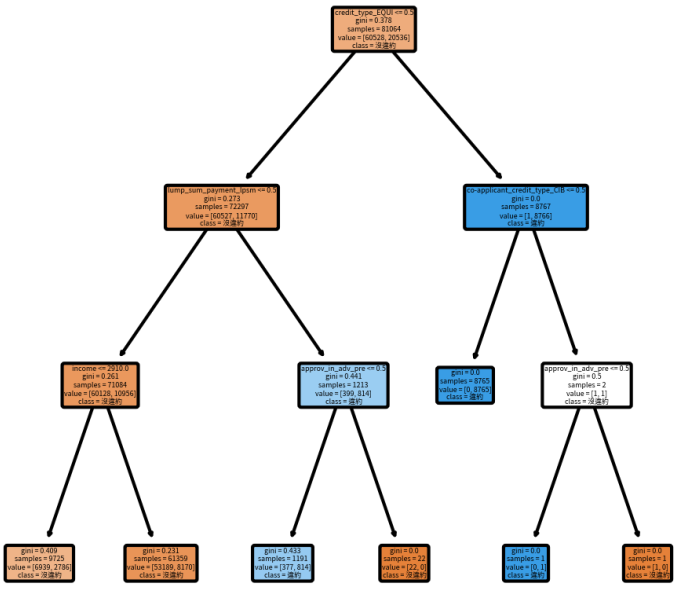
決策樹實作過程：

```
#決策樹
#  訓練
from  sklearn.tree  import  DecisionTreeClassifier
fig,  axes  =  plt.subplots(nrows  =  1,ncols  =  1,figsize  =  (4,4),  dpi=300)
algorithm  =  DecisionTreeClassifier(max_depth=3,random_state=random_seed)
algorithm  =algorithm.fit(x_train,y_train)

#  繪製決策樹的樹狀結構
from  sklearn  import  tree

import  pydotplus
from  IPython.display  import  Image
iclass=['沒違約','違約']
tree.plot_tree(algorithm,feature_names=x_train.columns,class_names=iclass,filled=True,rounded=True)
fig.savefig('imagename.png')
```

訓練後可得到正確率約為 85.93%：

	precision	recall	f1-score	support
0	0.85	0.99	0.91	40353
1	0.96	0.47	0.63	13691
accuracy			0.86	54044
macro_avg	0.90	0.73	0.77	54044
weighted_avg	0.87	0.86	0.84	54044
分類正確率：0.8592813263266967				

詳細決策樹如下圖所示：



已註解 [E2]: 建議重新畫過決策樹，字太小了幾乎看不見

四、隨機森林

隨機森林是一種包含多棵決策樹的模型，在森林裡面建構一棵棵各自獨立的決策樹，最後以投票方式來決定最終的結果。當進行分類任務時，有新的資料輸入，就讓森林中的每一顆決策樹分別進行分類，並統計各個決策樹的分類結果將該筆資料分成最多的那一種類別，隨機森林就會把這個類別當成最終結果。

隨機森林實作過程：

```
# 隨機森林
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
algorithm = RandomForestClassifier(random_state=random_seed)
algorithm =algorithm.fit(x_train,y_train)
y_predicted=algorithm.predict(x_test)
print('分類正確率',metrics.accuracy_score(y_test,y_predicted))
print(metrics.classification_report(y_test,y_predicted))
```

訓練後可得到正確率約為 86.60%：

	precision	recall	f1-score	support
0	0.86	0.98	0.92	40353
1	0.90	0.53	0.67	13691
accuracy			0.87	54044
macro_avg	0.88	0.76	0.79	54044
weighted_avg	0.87	0.87	0.85	54044
分類正確率：0.8660350825253497				

五、XGBoost

XGBoost 全名是 eXtreme Gradient Boosting，是 boosting 演算法的其中一種，XGBoost 是一種將許多樹模型集成再一起形成的強分類器。使得後面產生的樹可以透過改進前一個生成樹的缺失，來提升最終的模型結果。

XGBoost 實作過程：

```
# XGBoost
from xgboost import XGBClassifier
from sklearn import metrics
algorithm = XGBClassifier(random_state=random_seed)
algorithm=algorithm.fit(x_train,y_train)
y_predicted=algorithm.predict(x_test)
print('分類正確率',metrics.accuracy_score(y_test,y_predicted))
print(metrics.classification_report(y_test,y_predicted))
```

訓練後可得到正確率約為 87.06%：

	precision	recall	f1-score	support
0	0.86	0.99	0.92	40353
1	0.96	0.51	0.67	13691
accuracy			0.87	54044
macro_avg	0.91	0.75	0.79	54044
weighted_avg	0.88	0.87	0.86	54044
分類正確率：0.8705684257271853				

伍、模型評估

一、模型 AUC 比較

這部份我們利用 K-fold 交叉驗證個別計算多種演算法的平均 AUC 值來挑選最佳的模型，我們將結果透過下列程式碼轉換為分析資料，以列表方式呈現如下：

```
# 將候選演算法建立成列表

# 邏輯斯迴歸
from sklearn.linear_model import LogisticRegression
algorithm1 = LogisticRegression(random_state=random_seed)

# 決策樹
from sklearn.tree import DecisionTreeClassifier
algorithm2 = DecisionTreeClassifier(random_state=random_seed)

# 隨機森林
from sklearn.ensemble import RandomForestClassifier
algorithm3 = RandomForestClassifier(random_state=random_seed)

# XGBoost
from xgboost import XGBClassifier
algorithm4 = XGBClassifier(random_state=random_seed)

# SVM
from sklearn.svm import SVC
algorithm5 = SVC(kernel='rbf',random_state=random_seed)

algorithms = [algorithm1, algorithm2, algorithm3, algorithm4,algorithm5]
```

```
# 利用交叉驗證選擇最佳演算法
from sklearn.model_selection import StratifiedKFold
stratifiedkfold = StratifiedKFold(n_splits=3)

from sklearn.model_selection import cross_val_score
for algorithm in algorithms:
        scores = cross_val_score(algorithm , x_train, y_train,
                cv=stratifiedkfold, scoring='roc_auc')
        score = scores.mean()
        name = algorithm.__class__.__name__
        print(f'平均分數 : {score:.4f} 個別分數 : {scores} {name}')
```

比較表如下：

	平均分數	個別分數
LogisticRegression	0.5738	[0.5678 0.5739 0.5797]
DecisionTreeClassifier	0.7336	[0.7311 0.7354 0.7343]
RandomForestClassifier	0.8356	[0.8327 0.8358 0.8384]
XGBClassifier	0.8537	[0.8545 0.8539 0.8528]
SVC	0.5138	[0.503 0.5077 0.5305]

二、模型選擇以及後續訓練

經過上表的比較後，發現 XGBboost 是 4 個候選演算法中 AUC 最高的，因此接下來的步驟使用 XGBoost 演算法對資料進行訓練、預測及評估

訓練、預測、評估

在評估部分，我們使用混淆矩陣來表示預測結果與標準答案

```
from sklearn.metrics import confusion_matrix
df_matrix = make_cm(
        confusion_matrix(y_test, y_pred), ['沒違約', '違約'])
display(df_matrix)


from sklearn.metrics import precision_recall_fscore_support
precision, recall, fscore, _ = precision_recall_fscore_support(
        y_test, y_pred, average='binary')
print(f'精確性 : {precision:.4f}  召回率 : {recall:.4f}  F 分數 : {fscore:.4f}')
```

混淆矩陣的結果如下：

		預測結果	
		沒違約	違約
標準答案	沒違約	40081	272
	違約	6723	6968

精確性 : 0.9624 召回率 : 0.5089 F 分數 : 0.6658

三、調整模型閾值

在取得 XGBboost 分析結果後，我們對模型透過調整閾值來提高預測的精確度：透過分析預測閾值為 0.5 以外時的情形來將模型優化，並根據違約預測數、精準性、召回率、F 分數做為調整的基準：

```
def pred(algorithm, x, thres):
        y_proba = algorithm.predict_proba(x)
        y_probal = y_proba[:,1]
        y_pred = (y_probal > thres).astype(int)
        return y_pred
```

```
# 以 0.05 為間距逐次改變閾值，並計算精確性、召回率及 F 分數
thres_list = np.arange(0.5, 0, -0.05)

for thres in thres_list:
        y_pred = pred(algorithm, x_test, thres)
        pred_sum = y_pred.sum()
        precision, recall, fscore, _ = precision_recall_fscore_support(
                y_test, y_pred, average='binary')
        print(f'閾值 : {thres:.2f} 違約預測數 : {pred_sum}\
  精確性 : {precision:.4f} 召回率 : {recall:.4f} F 分數 : {fscore:.4f})')
```

經過計算後可得到以下結果：

閾值	違約預測數	精確性	召回率	F 分數
0.50	7240	0.9624	0.5089	0.6658
0.45	7482	0.9491	0.5187	0.6708
0.40	7931	0.9235	0.5349	0.6775
0.35	8825	0.8767	0.5651	0.6872
0.30	10006	0.8142	0.5951	0.6876
0.25	12014	0.7289	0.6396	0.6813
0.20	16271	0.6009	0.7142	0.6527
0.15	24929	0.4493	0.8181	0.5800
0.10	40585	0.3193	0.9465	0.4775
0.05	52318	0.2608	0.9965	0.4134

由上表我們可以得出，閾值在 0.3 時 F 分數最大，因此我們使用閾值為 0.3 的結果再次繪製出混淆矩陣：

```
1  # 最大化 F 分數的閾值為 0.3
2  y_final = pred(algorithm, x_test, 0.3)
3
4  # 輸出混淆矩陣
5  df_matrix2 = make_cm(
6          confusion_matrix(y_test, y_final), ['沒違約', '違約'])
7  display(df_matrix2)
8
9  # 計算精確性、召回率與 f1 值
10 precision, recall, fscore, _ = precision_recall_fscore_support(
11         y_test, y_final, average='binary')
12 print(f'精確性 : {precision:.4f} 召回率 : {recall:.4f}\
13     F 分數: {fscore:.4f}')
```

更新後的混淆矩陣如下：

		預測結果	
		沒違約	違約
標準答案	沒違約	38494	1859
	違約	5544	8147

精確性：0.8142 召回率：0.5951 F 分數: 0.6876

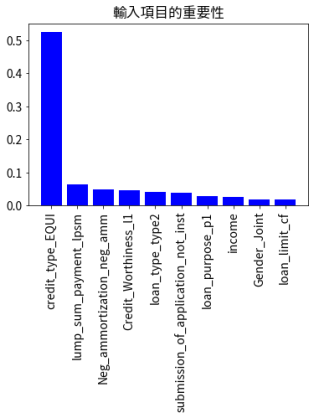
根據結果顯示模型有 59.51%的 Recall 值與 81.42%的 Precision 值，這代表模型整體有 81.42%針對貸款人是否會違約的預測正確率，而原本違約的人會被正確預測成會違約也有 59.51%的正確率，對於銀行評估貸款人潛在違約可能性有相當程度的貢獻，可以大幅度降低發生貸款人違約的風險。

最後在所有欄位中篩選出前 10 大影響貸款人是否會違約的重要因素：

```
#  重要性分析
importances  =  algorithm.feature_importances_
w  =  pd.Series(importances,  index=x.columns)

#  按值的大小排序
u  =  w.sort_values(ascending=False)

#  取出前10個個欄位
v  =  u[:10]
#直方圖
plt.title('輸入項目的重要性')
plt.bar(range(len(v)),  v,  color='b',  align='center')
plt.xticks(range(len(v)),  v.index,  rotation=90)
plt.show()
```

下表顯示了前十大影響貸款者為約與否的潛在因素：可以看出信用類型是否為 EQUI 為貸款人會違約的最重要影響因素，另外像是貸款的類型與目的以及貸款人的收入對於違約情況也有一定的預測能力。

已註解 [E3]: 這幾項目且協助在旁邊加上相對應的說明，感謝



陸、Deployment

經過了對貸款人的各項指標分析後，我們可以輕易的分辨出貸款人違約的機率，如此一來可以透過機器學習的模型來分析，減少所需的人力資源，銀行可以增加放款的速度，同時減少放款風險、增加銀行利潤，但我們仍必須要考慮到在不同的經濟環境與不同國家的文化之下，相同的機器學習模型是否可以適用，因此我們建議在不同國家要使用不同的分類模型以提高準確率。

程式碼連結:
https://colab.research.google.com/drive/1TLlfQPbumRKYSXwoWrSBnPN_D_YGnBh0#scrollTo=3o4jUNMrDD6L