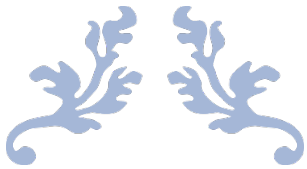


機器學習導論-期末報告

Kaggle 競賽：WSDM - KKBox's Music Recommendation Challenge

組員：

巨資四 B　06170203　劉馨瑄

巨資四 B　06170245　楊怡芊

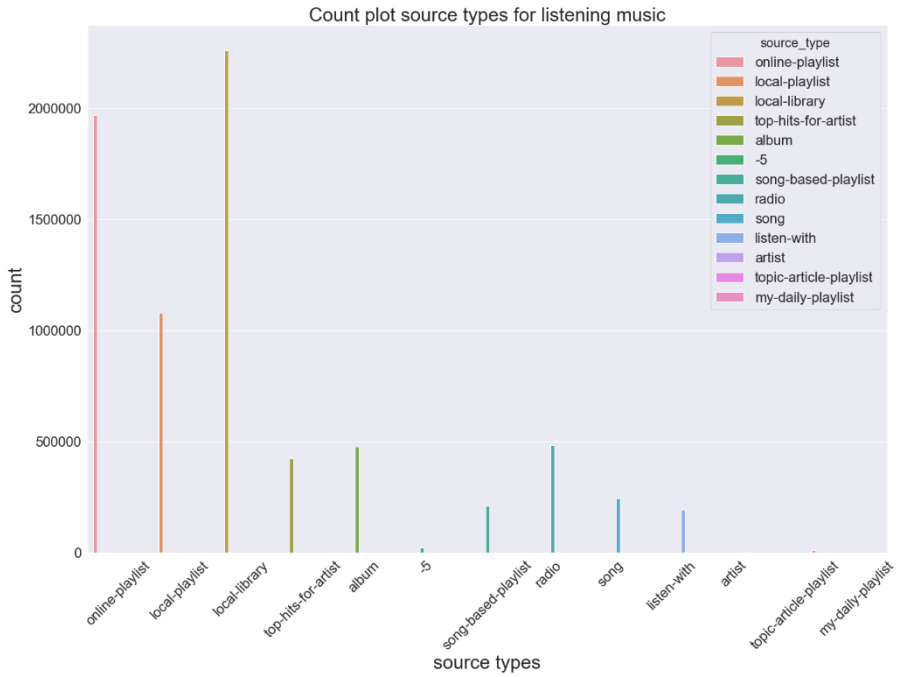
目錄

壹、資料探索式分析與視覺化

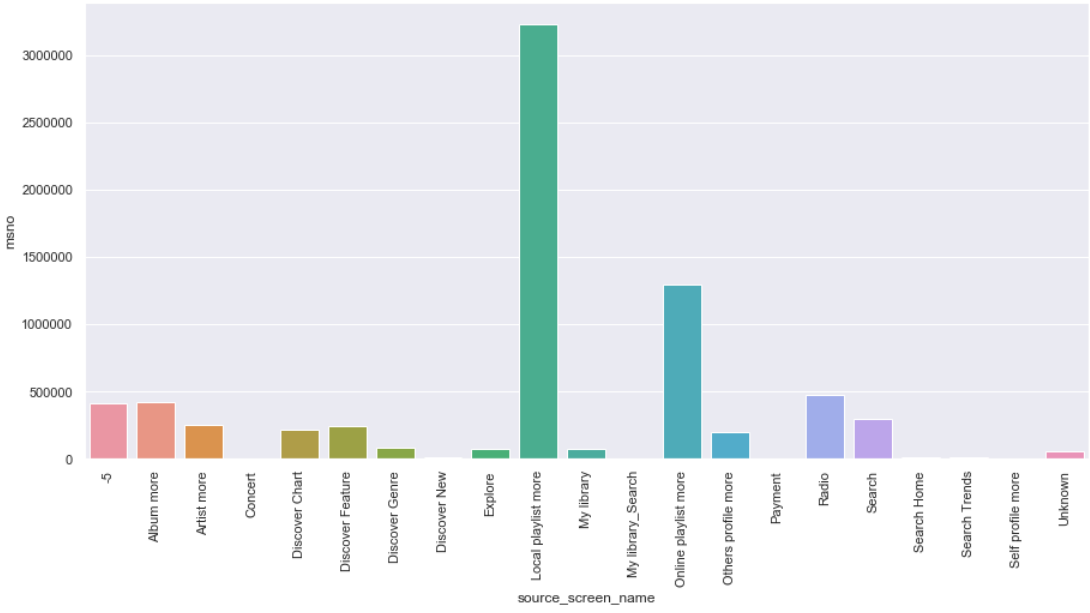
先將 train.csv、songs.csv、member.csv 資料集合併之後，首先觀察所有欄位 NA 值的數量，可以發現 gender、lyricist、composer 等欄位空值個數較多。先將 NA 值用(-5) 補上以利視覺化分析。

```
Out[129]: msno                          0
          song_id                       0
          source_system_tab         24849
          source_screen_name       414796
          source_type               21539
          target                        0
          song_length                   0
          genre_ids                118341
          artist_name                   0
          composer                1675592
          lyricist                3178684
          language                     36
          city                          0
          bd                            0
          gender                  2961436
          registered_via                0
          registration_init_time        0
          expiration_date               0
          registration_init_time_year   0
          registration_init_time_month  0
          expiration_date_year          0
          expiration_date_month         0
          registration_init_time_day    0
          expiration_date_day           0
          registration_init_time_new    0
          expiration_date_new           0
          dtype: int64
```

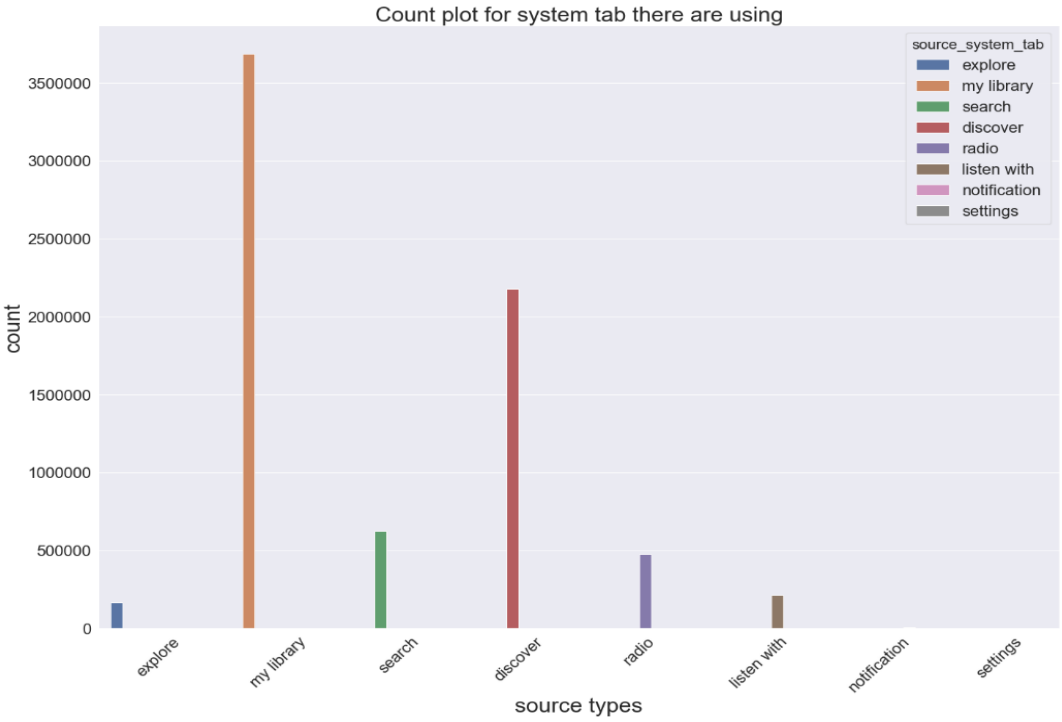
→ 在 source types 中 local-library 最多其次是 online playlist 和 local playlist 還有部分資料為空值(-5)。



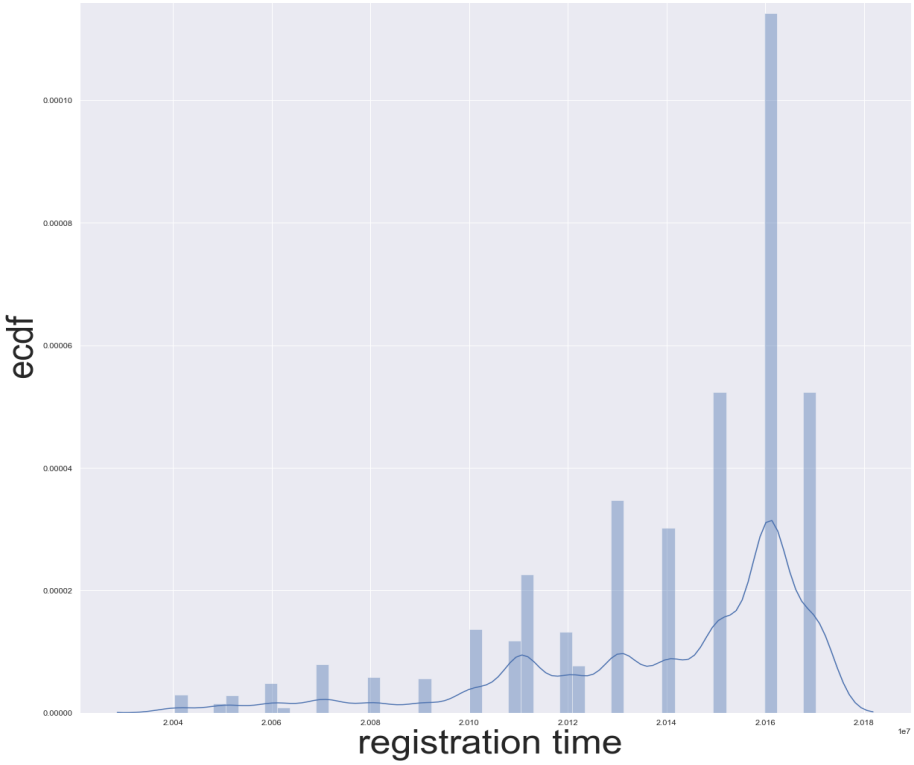
觀察 source screen name 可以看到大多數用戶都點選 Local playlist more，且部分用戶會通過 Online playlist(在線播放列表)返回歌曲來源。



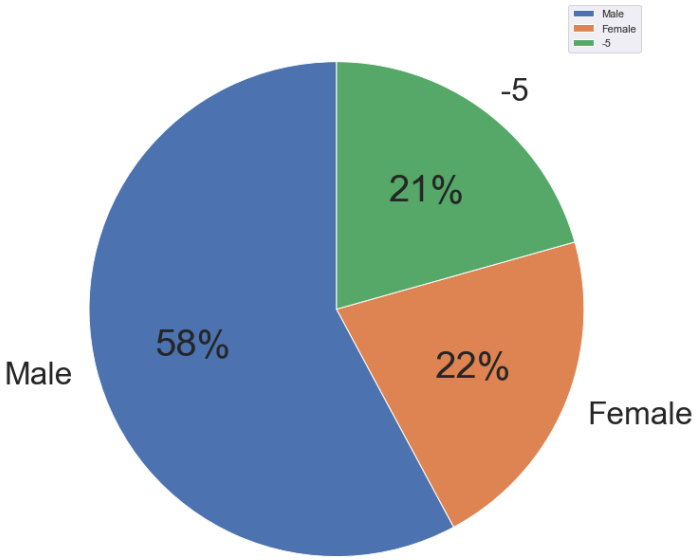
從 source system tab 中可以看出 my library (我的音樂庫)依然是最高最受歡迎的方式。



觀察註冊時間的經驗累積分佈函數圖可以看到 2012 年至 2016 年期間為註冊量的高峰，尤其在 2016 年。

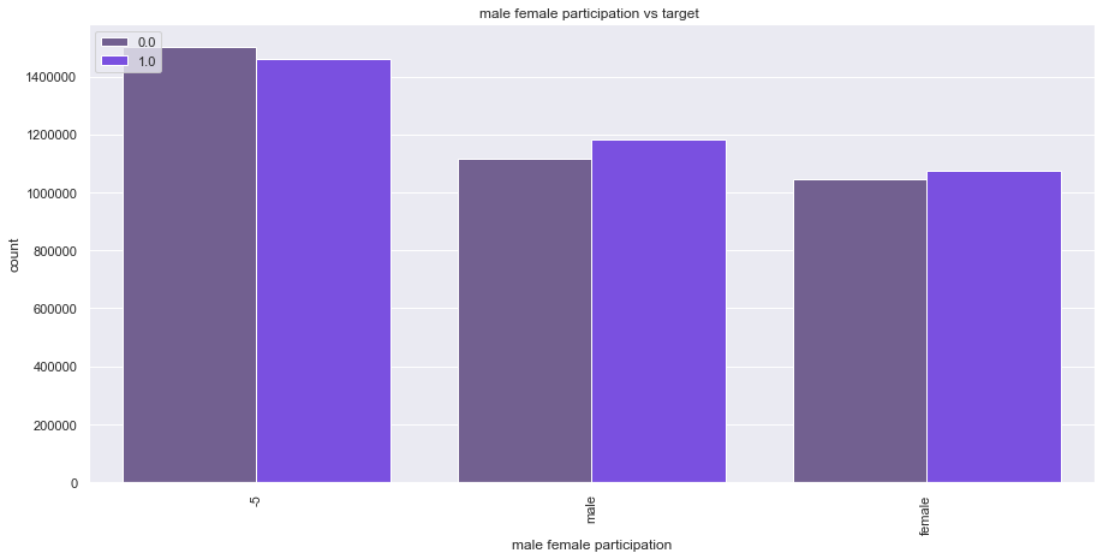


觀察 genders 可以看出男性較多，且 gender 的空值(-5)比例偏高。可以更進階的觀察男性與女性在註冊方式、收聽習慣上有何差異。

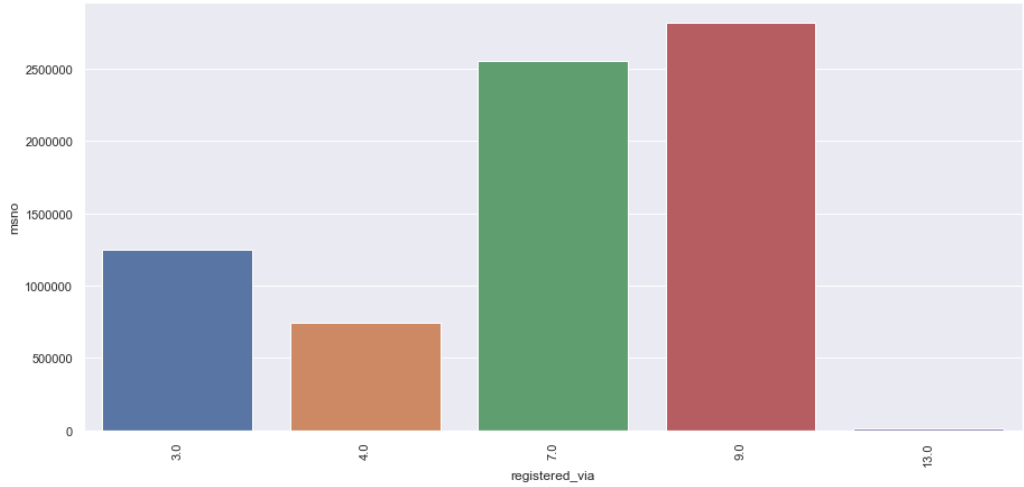


從以下的圖可以看出 男性在選擇尋找音樂時都只使用一種方式達成，而女性則是以多種方式搜尋。這也與現實世界中男性對於事件多以一個方向鑽研，而女性通常關注所有可能發展的面向相符合。

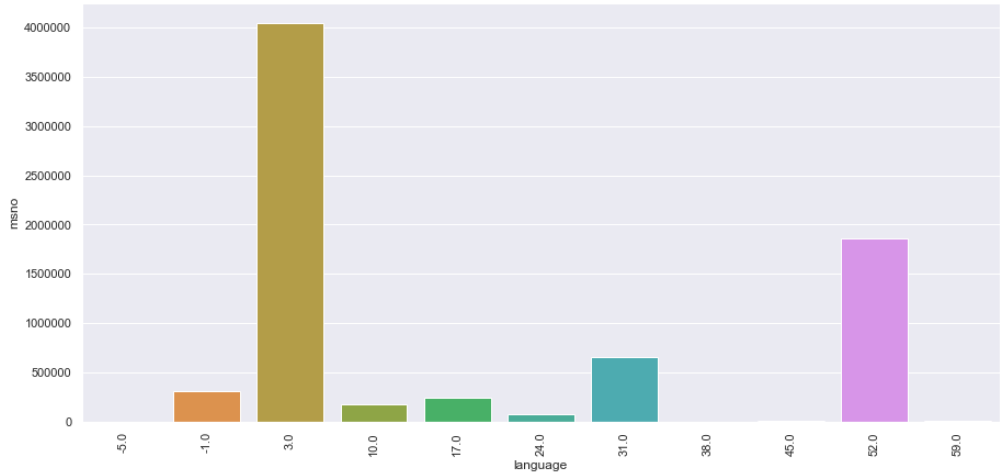
觀察下圖可以發現不論 target=1or0 男性用戶(重複收聽用戶)都較女性來得多。另外，兩個性別的 target=1 用戶數皆高於 target=0。



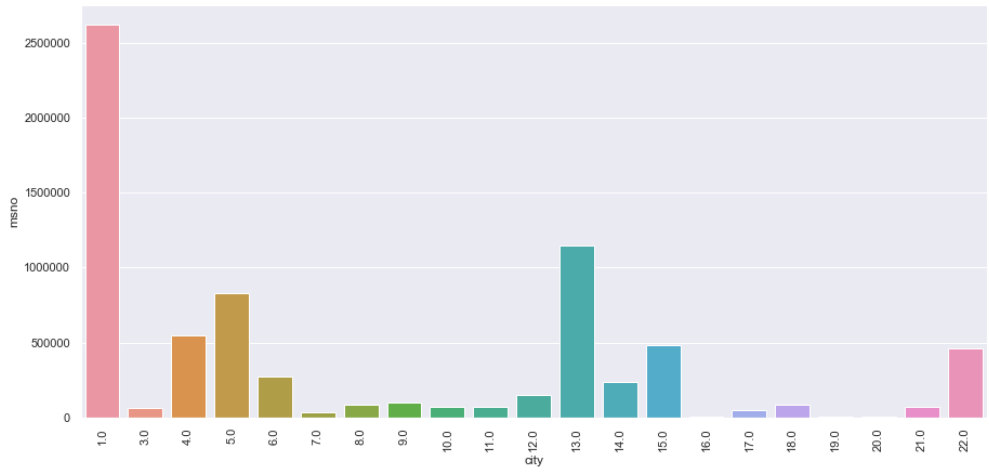
註冊方式則以 7&9 為主要方式。



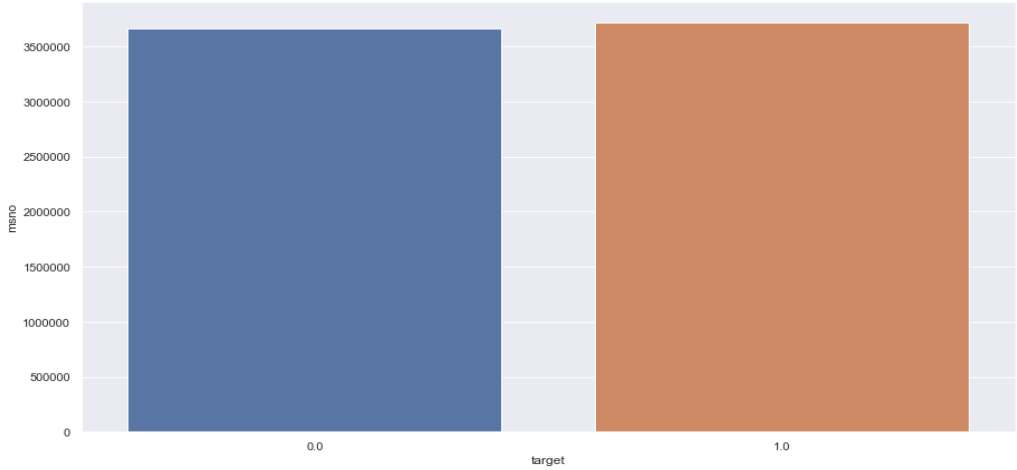
觀察 language 可以看出用戶使用語言明顯是語言 3 與語言 52 佔多數



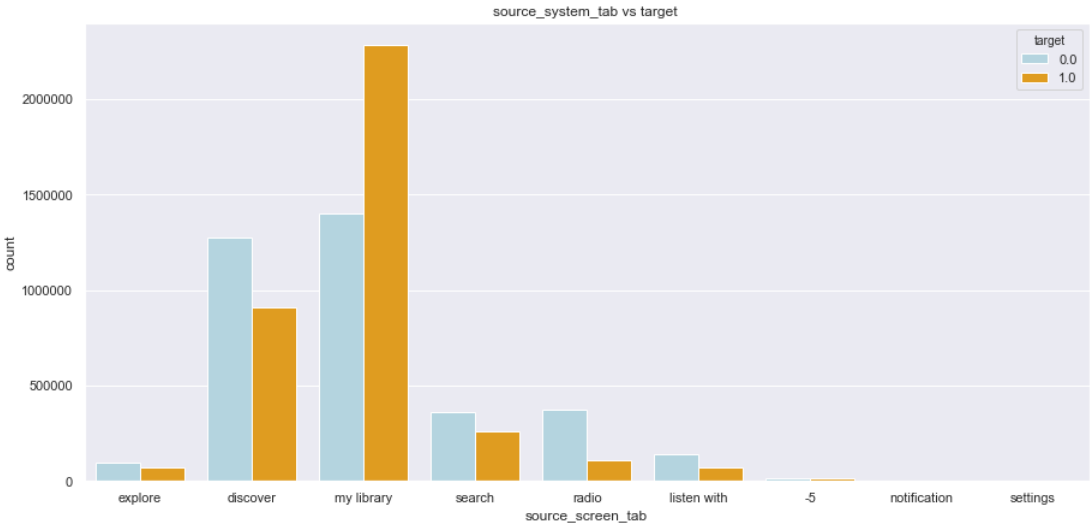
觀察 city 可以看出 city1 的使用者站資料中的多數。



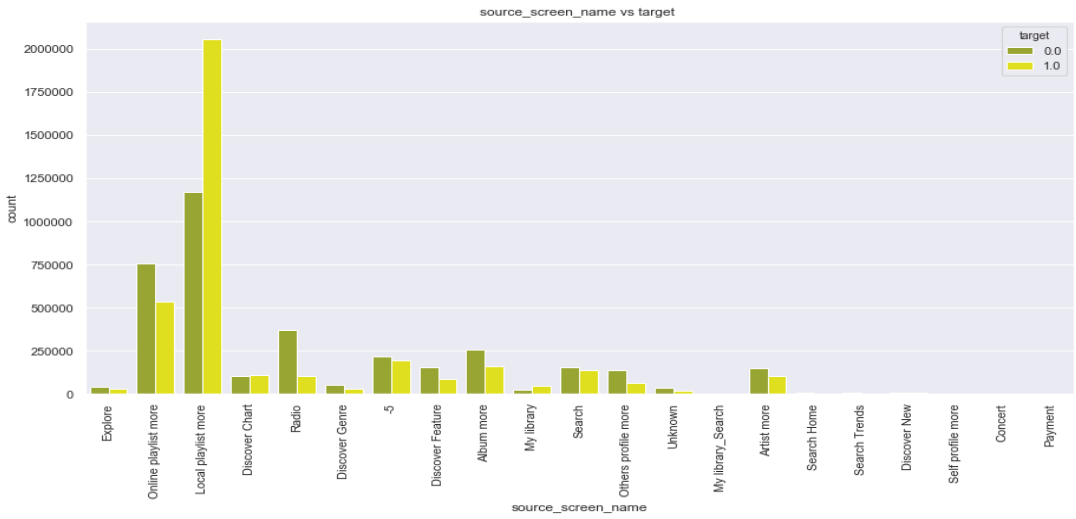
下圖為 target 1 與 0 差距為(還沒算)



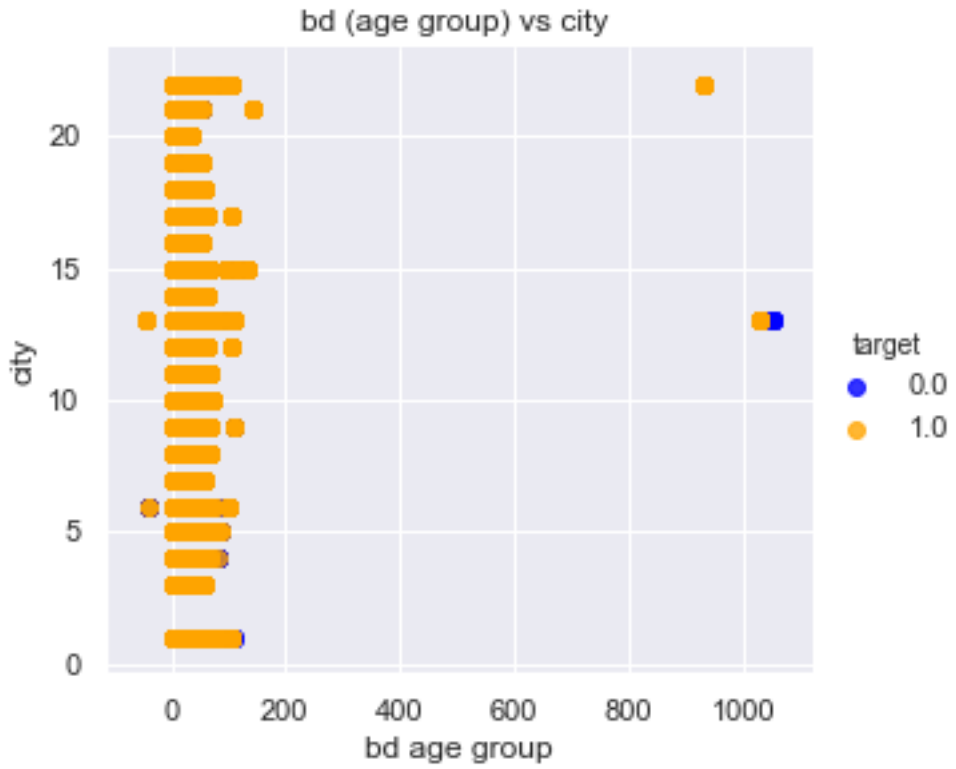
觀察 source system tab 與 target 可以看出一個月內沒有重複收聽的用戶大部分會從 discover 和 my library 收聽音樂，而重複收聽者多從 my library。



觀察下圖可以發現不論 target=0 or1 Local playlist more 是他們收聽歌曲最常使用的方式。



觀察下圖我們可以 bd 資料多半在 0~100 變動，可以從此明顯觀察出 bd 的離群值。



貳、資料前處理

一、讀取資料表並合併資料欄位

```
df_train = pd.read_csv('train.csv')
df_song = pd.read_csv('songs.csv')
df_members = pd.read_csv('members.csv')
df_test = pd.read_csv('test.csv')
# df_target = pd.read_csv('sample_submission.csv')
```

→ train 資料與 song 資料依據 song_id 合併資料、與 member 資料依據 msno 合併。新資料為 df_train_new。

```
df_train_new = pd.merge(df_train, df_song, on='song_id')
df_train_new = pd.merge(df_train_new,df_members, on='msno')
df_train_new.head()
```

檢視 df_train_new 的欄位確認合併成功與其欄位為何。

```
df_train_new.columns
```

```
Index(['msno', 'song_id', 'source_system_tab', 'source_screen_name',
       'source_type', 'target', 'song_length', 'genre_ids', 'artist_name',
       'composer', 'lyricist', 'language', 'city', 'bd', 'gender',
       'registered_via', 'registration_init_time', 'expiration_date'],
      dtype='object')
```

test 資料則與 song、member 資料依據 test 資料所有欄位與 song 資料合併(若無資料則顯示空值)，新資料為 df_test_new

```
df_test_new = pd.merge(df_test, df_song, how = 'left')
df_test_new = pd.merge(df_test_new,df_members, how = 'left')
df_test_new.head()
```

檢視 df_test_new 的欄位確認合併成功與其欄位為何。

```
df_test_new.columns
```

```
Index(['id', 'msno', 'song_id', 'source_system_tab', 'source_screen_name',
       'source_type', 'song_length', 'genre_ids', 'artist_name', 'composer',
       'lyricist', 'language', 'city', 'bd', 'gender', 'registered_via',
       'registration_init_time', 'expiration_date'],
      dtype='object')
```

二、轉換日期欄位格式

資料中包含時間的欄位有:registration_init_time(開始註冊時間)與expiration_date(用戶到期時間) 將資料先轉換回時間格式並且切分新增年、月欄位。

```
df_train_new['registration_init_time_year'] = df_train_new['registration_init_time'].dt.year
df_train_new['registration_init_time_month'] = df_train_new['registration_init_time'].dt.month
# df_train_new['registration_init_time_day'] = df_train_new['registration_init_time'].dt.day

df_test_new['registration_init_time_year'] = df_test_new['registration_init_time'].dt.year
df_test_new['registration_init_time_month'] = df_test_new['registration_init_time'].dt.month

df_train_new['expiration_date_year'] = df_train_new['expiration_date'].dt.year
df_train_new['expiration_date_month'] = df_train_new['expiration_date'].dt.month
# df_train_new['expiration_date_day'] = df_train_new['expiration_date'].dt.day

df_test_new['expiration_date_year'] = df_test_new['expiration_date'].dt.year
df_test_new['expiration_date_month'] = df_test_new['expiration_date'].dt.month
```

新增後成果如下：

registration_init_time	expiration_date	registration_init_time_year	registration_init_time_month	expiration_date_year	expiration_date_month
2012-01-02	2017-10-05	2012	1	2017	10

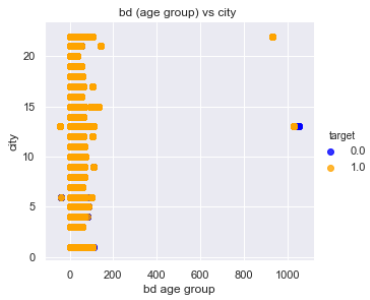
為了讓資料能放入模型必須再將資料轉換為時間戳，使用下列方式轉換：

```
import time

df_train_new['registration_init_time_new'] = df_train_new['registration_init_time'].apply(lambda x:int(time.mktime(x.timetup
df_train_new['expiration_date_new'] = df_train_new['expiration_date'].apply(lambda x:int(time.mktime(x.timetuple())))

df_test_new['registration_init_time_new'] = df_test_new['registration_init_time'].apply(lambda x:int(time.mktime(x.timetuple
df_test_new['expiration_date_new'] = df_test_new['expiration_date'].apply(lambda x:int(time.mktime(x.timetuple())))
```

轉換後成果如下：

registration_init_time_new	expiration_date_new
1423497600	1524153600
1423497600	1524153600

三、刪除異常值欄位



由左圖可以看出 bd 有離群值且是異常值，正常人類壽命應界於 0~100 歲之間。因此觀察 bd<0 有 195 筆、bd>100 的資料有 6508 筆，將異常值刪除。

```
df_train_new[df_train_new['bd']<0]
```

195 rows × 24 columns

```
df_train_new[df_train_new['bd']>100]
```

6508 rows × 24 columns

```
df_train_new = df_train_new[df_train_new['bd']<=100]
df_train_new = df_train_new[df_train_new['bd']>=0]
df_train_new
```

（僅保留 0<=bd<=100 的列數）

四、資料內容

欄位名稱	內容	個數
source_system_tab	my library,nan,discover,explore,settings,listen with,notification,search,radio	9
source_screen_name	nan, Search Trends,Search Home, Online playlist more, Discover Feature, Payment, Album more, Explore, Search, My library_Search, Self profile more, Discover New, Artist more, Unknown, Local playlist more, Radio, Discover Chart, My library, Others profile more, Discover Genre, Concert	21
source_type	nan, song-based-playlist, listen-with,online-playlist, topic-article-playlist, local-library, top-hits-for-artist, song, album, local-playlist, my-daily-playlist, radio,artist	13
target	0, 1	2
language	3, 10, 17, 24, 31, 38, 45, 52, 59,-1	10
city	1, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22	21
gender	nan, male, female	3
registered_via	3, 4, 7, 9, 13	5

五、NA 缺失值觀察與處理

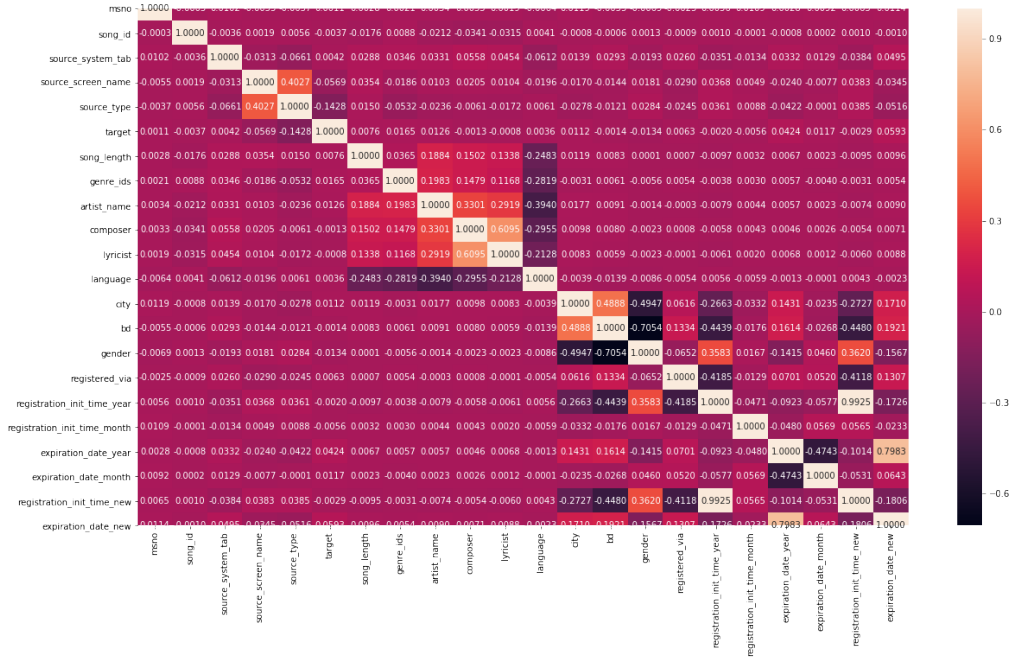
```
df_train_new.isnull().sum(axis=0)
```

```
index                          0
msno                           0
song_id                        0
source_system_tab          24845
source_screen_name         414719
source_type                21535
target                         0
song_length                    0
genre_ids                  118252
artist_name                    0
composer                 1674052
lyricist                 3175831
language                      36
city                           0
bd                             0
gender                   2961241
registered_via                 0
registration_init_time         0
expiration_date                0
registration_init_time_year    0
registration_init_time_month   0
expiration_date_year           0
expiration_date_month          0
registration_init_time_new     0
expiration_date_new            0
dtype: int64
```

```
df_test_new.isnull().sum(axis=0)
```

```
index                          0
id                             0
msno                           0
song_id                        0
source_system_tab           8442
source_screen_name        162883
source_type                 7297
song_length                   25
genre_ids                  42110
artist_name                   25
composer                  619304
lyricist                 1224744
language                      42
city                           0
bd                             0
gender                   1052224
registered_via                 0
registration_init_time         0
expiration_date                0
registration_init_time_year    0
registration_init_time_month   0
expiration_date_year           0
expiration_date_month          0
registration_init_time_new     0
expiration_date_new            0
dtype: int64
```

可以看出 train 與 test 資料中 gender、lyricist 欄位有許多空值。



從熱度圖觀察 gender 與 source_type 與其他為相關度為負值，考量到這兩個欄位相關度為負且有許多空值，因此將 gender、source_type 從 train 與 test 中刪除。

六、One-hot Encoding

為了要模型進行預測，必須將類別 (categorical)或是文字(text)的資料轉換成數字，讓程式能夠更好的去理解及運算。LabelEncoding 及 One hot encoder 都是常見的方式。先觀察 df_train_new 與 df_test_new 的資料筆數：

```
print(df_train_new.shape)
print(df_test_new.shape)
```

```
(7370601, 23)
(2556790, 23)
```

為了防止特徵數不一樣丟入模型會出現錯誤，因此必須將 df_train 與 df_test 先合併成 all_data 等到完成編碼後再重新切割。

```
all_data = pd.concat([df_train_new,df_test_new])
all_data.shape
```

考慮到電腦效能（欄位過多模型電腦會跑不動）所以選擇 Language、registered_via 這兩個類別較少的欄位進行 one-hot Encoding 其餘需要編碼的欄位退而求其次使用 LabelEncoder 來編碼。

```
data_dummies = pd.get_dummies(all_data[['source_system_tab',#'city','source_screen_name',#'source_type','gender','Language',
                                        'registered_via']],dummy_na=True)#, drop_first = True)
data_dummies.head()
```

tings	source_system_tab_nan	registered_via_13	registered_via_16	registered_via_3	registered_via_4	registered_via_7	registered_via_9	registered_via_nan
0	0	0	0	0	0	1	0	0

觀察編碼後的欄位列舉如下：

```
data_dummies.columns
```

```
Index(['source_system_tab_discover', 'source_system_tab_explore',
       'source_system_tab_listen with', 'source_system_tab_my library',
       'source_system_tab_notification', 'source_system_tab_radio',
       'source_system_tab_search', 'source_system_tab_settings',
       'source_system_tab_nan', 'registered_via_13', 'registered_via_16',
       'registered_via_3', 'registered_via_4', 'registered_via_7',
       'registered_via_9', 'registered_via_nan'],
      dtype='object')
```

接著，刪除編碼之後已經不需要的 Language、registered_via 欄位，並且將 all_data 與 data_dummies 合併成 df_new。

```
all_data = df_new.drop(['source_system_tab','registered_via'], axis = 1)
all_data
```

```
df_new = pd.concat([all_data,data_dummies],axis=1)
df_new
```

七、LabelEncoder

msno, song_id, genre_ids, source_screen_name, artist_name, composer, lyricist, language, city → 將以上幾個欄位進行 labelEncoding。

因為 language 與 registered_via 的值雖然是數值但是並不是連號的，因此先將上述兩個欄位內容的格式轉換為文字格式，才能順利 labelEncoder。

```
df_train_new['language'] = df_train_new['language'].astype(str)
df_test_new['language'] = df_test_new['language'].astype(str)
```

```
df_train_new['registered_via'] = df_train_new['registered_via'].astype(str)
df_test_new['registered_via'] = df_test_new['registered_via'].astype(str)
```

```
from sklearn.preprocessing import LabelEncoder
enc = LabelEncoder()

for col in [
    'msno', 'song_id', 'genre_ids', 'source_screen_name',
    'artist_name', 'composer', 'lyricist','language','city' #,'gender'
]:
    all_data[col] = enc.fit_transform(all_data[col].fillna('nan'))
```

最後將 all_data 切割回 df_train_new、df_test_new

```
df_train_new = all_data[:7370601]
df_test_new = all_data[7370601:]
```

並確認資料筆數正確無誤。

```
print(df_train_new.shape)
print(df_test_new.shape)
```

```
(7370601, 23)
(2556790, 23)
```

八、切出 ID 欄位

由於 df_train 沒有 id 這個欄位，之前因為合併 df_train、df_test 後來切分資料時使得 df_train 多出了 id 欄位必須刪除。

```
df_train_new = df_train_new.drop(['id'], axis = 1)
df_train_new
```

另外必須將 df_test 的 id 這個欄位切割出來成為 test_id

```
test_id = df_test_new['id']
df_test_new = df_test_new.drop(['id'],axis=1)
df_test_new
```

```
test_id
```

```
0                0.0
1                1.0
2                2.0
3                3.0
4                4.0
              ...
2556785    2556785.0
2556786    2556786.0
2556787    2556787.0
2556788    2556788.0
2556789    2556789.0
Name: id, Length: 2556790, dtype: float64
```

同理，由於 df_test 沒有 target 這個欄位，之前因為合併 df_train、df_test 後來切分資料時使得 df_test 多出了 target 欄位必須刪除。

```
df_test_new = df_test_new.drop(['target'],axis=1)
```

下方資訊為最終丟入模型的欄位名稱與資料行列數：

```
df_train_new.columns
```

```
Index(['artist_name', 'bd', 'city', 'composer', 'expiration_date',
       'expiration_date_month', 'expiration_date_new', 'expiration_date_year',
       'genre_ids', 'index', 'language', 'lyricist', 'msno',
       'registration_init_time', 'registration_init_time_month',
       'registration_init_time_new', 'registration_init_time_year', 'song_id',
       'song_length', 'source_screen_name', 'target',
       'source_system_tab_discover', 'source_system_tab_explore',
       'source_system_tab_listen with', 'source_system_tab_my library',
       'source_system_tab_notification', 'source_system_tab_radio',
       'source_system_tab_search', 'source_system_tab_settings',
       'source_system_tab_nan', 'registered_via_13', 'registered_via_16',
       'registered_via_3', 'registered_via_4', 'registered_via_7',
       'registered_via_9', 'registered_via_nan'],
      dtype='object')
```

```
df_train_new.shape
```

```
(7370601, 37)
```

```
df_test_new.shape
```

```
(2556790, 36)
```

九、轉換為丟入模型的資料格式

```
df_x = df_train_new.drop(["target",'registration_init_time','expiration_date'], axis = 1)
# df_x
df_y = df_train_new['target']
df_y = pd.DataFrame(df_y)
# df_y

X = df_x.iloc[:,:].values
y = df_y.iloc[:,-1].values
```

```
df_x.columns    #丟入模型的欄位
```

```
Index(['artist_name', 'bd', 'city', 'composer', 'expiration_date_month',
       'expiration_date_new', 'expiration_date_year', 'genre_ids', 'index',
       'language', 'lyricist', 'msno', 'registration_init_time_month',
       'registration_init_time_new', 'registration_init_time_year', 'song_id',
       'song_length', 'source_screen_name', 'source_system_tab_discover',
       'source_system_tab_explore', 'source_system_tab_listen with',
       'source_system_tab_my library', 'source_system_tab_notification',
       'source_system_tab_radio', 'source_system_tab_search',
       'source_system_tab_settings', 'source_system_tab_nan',
       'registered_via_13', 'registered_via_16', 'registered_via_3',
       'registered_via_4', 'registered_via_7', 'registered_via_9',
       'registered_via_nan'],
      dtype='object')
```

```
len(df_x.columns)
```

```
34
```

```
df_x_test = df_test_new.drop(['registration_init_time','expiration_date'], axis = 1)
X_ = df_x_test.iloc[:,:].values
```

```
df_x_test.columns
```

```
Index(['artist_name', 'bd', 'city', 'composer', 'expiration_date_month',
       'expiration_date_new', 'expiration_date_year', 'genre_ids', 'index',
       'language', 'lyricist', 'msno', 'registration_init_time_month',
       'registration_init_time_new', 'registration_init_time_year', 'song_id',
       'song_length', 'source_screen_name', 'source_system_tab_discover',
       'source_system_tab_explore', 'source_system_tab_listen with',
       'source_system_tab_my library', 'source_system_tab_notification',
       'source_system_tab_radio', 'source_system_tab_search',
       'source_system_tab_settings', 'source_system_tab_nan',
       'registered_via_13', 'registered_via_16', 'registered_via_3',
       'registered_via_4', 'registered_via_7', 'registered_via_9',
       'registered_via_nan'],
      dtype='object')
```

```
len(df_x_test.columns)
```

```
34
```

將資料切分為訓練資料集與測試資料集(7:3)

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)#,random_state=3)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)


all_test_std = sc.transform(X_)
```

```
X.shape
```

```
(7370601, 34)
```

```
X_.shape
```

```
(2556790, 34)
```

十、輸出 csv

```
np.any(np.isnan(X_))
```

```
True
```

```
np.all(np.isfinite(X_))
```

```
False
```

```
df_test_new = df_test_new.fillna(-100)
```

```
df_x_test = df_test_new.drop(['registration_init_time','expiration_date'], axis = 1)
X_ = df_x_test.iloc[:,:].values
all_test_std = sc.transform(X_)
```

```
print(np.any(np.isnan(X_)))
print(np.all(np.isfinite(X_)))
```

```
False
True
```

```
df_train_new.to_csv("機器學習期末kkbox.csv")
```

```
df_test_new.to_csv("機器學習期末kkbox_test.csv")
```

參、模型建置與初步比較

下方二 def 為評估模型時所使用的 def：包含混淆矩陣、F1-score、precision、recall、accuracy 等指標。

```
#confusion matrix
from sklearn.metrics import confusion_matrix
#ROC & AUC
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

def evaluate_model(predicted, actual):

    # Table-Type Plotting
    #print('Confusion Matrix:\n{}'.format(confusion_matrix(predicted, actual)))
    print('Classification Report:\n{}'.format(classification_report(predicted, actual)))
    print('Accuracy: {}'.format(accuracy_score(predicted, actual)))
    print('Precision: {}'.format(precision_score(predicted, actual)))
    print('Recall: {}'.format(recall_score(predicted, actual)))
    print('F-1: {}'.format(f1_score(predicted, actual)))
    print('AUC: {}'.format(roc_auc_score(predicted, actual)))

    # ROC Curve Plotting
    fpr, tpr, thresh = roc_curve(actual, predicted)
    roc_auc = roc_auc_score(predicted, actual)
    plt.title('ROC')
    plt.plot(fpr, tpr, 'b',
    label='AUC = %0.2f'% roc_auc)
    plt.legend(loc='lower right')
    plt.plot([0,1],[0,1],'r--')
    plt.xlim([-0.1,1.2])
    plt.ylim([-0.1,1.2])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()

    print('Confusion Matrix:\n{}'.format(confusion_matrix(predicted, actual)))
    tp, fp, fn, tn  = confusion_matrix(actual, predicted).ravel()
    print("True positives: " + str(tp))
    print("False positives: " + str(fp))
    print("True negatives: " + str(tn))
    print("False negatives: " + str(fn))
    print('\n')
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix

def printCM(y,y_pred):
    print('Confusion Matrix:\n{}'.format(confusion_matrix(y, y_pred)))
    tp, fp, fn, tn  = confusion_matrix(y, y_pred).ravel()
    print("True positives: " + str(tp))
    print("False positives: " + str(fp))
    print("True negatives: " + str(tn))
    print("False negatives: " + str(fn))
    print('\n')
```

```
import pandas as pd
import csv
import numpy as np


df = pd.read_csv('機器學習期末kkbox.csv', index_col=0)
df_test = pd.read_csv('機器學習期末kkbox_test.csv', index_col=0)

df_x = df.drop(["target",'registration_init_time','expiration_date'], axis = 1)
df_y = df['target']
df_y = pd.DataFrame(df_y)

X = df_x.iloc[:,:].values
y = df_y.iloc[:,-1].values


df_x_test = df_test.drop(['registration_init_time','expiration_date'], axis = 1)
X_ = df_x_test.iloc[:,:].values

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)#,random_state=3)

sc = StandardScaler()
sc.fit(X_train)
X_train_std = sc.transform(X_train)
X_test_std = sc.transform(X_test)


all_test_std = sc.transform(X_)
```

```
print(X.shape)
print(X_.shape)
print(np.any(np.isnan(X_)))
print(np.all(np.isfinite(X_)))
```

```
(7370601, 34)
(2556790, 34)
False
True
```

先使用以下幾種模型測試一下，觀察尚未調參之前的結果。再依據結果選擇表現最好的幾個模型進行參數的調整。(因為本來可能就較 fit 該資料集，推測可能預測成果也會較佳)

XGBoost

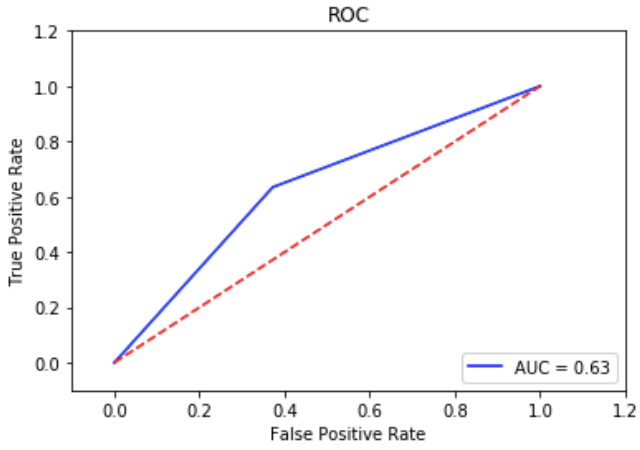
```
import warnings
warnings.filterwarnings('ignore')

from xgboost import XGBClassifier
xgb = XGBClassifier()

xgb.fit(X_train_std, y_train)
pred_y_test = xgb.predict(X_test_std)
```

```
[01:30:07] WARNING: src/learner.cc:686:
havior (exact greedy algorithm on singl
```

```
xgb.score(X_train_std, y_train)
```

```
0.6316770102065736
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.63      0.63      0.63   1097202
         1.0       0.63      0.63      0.63   1113979

    accuracy                           0.63   2211181
   macro avg       0.63      0.63      0.63   2211181
weighted avg       0.63      0.63      0.63   2211181

Accuracy: 0.6312726095240507
Precision: 0.6344740017650343
Recall: 0.63246973237377
F-1: 0.6334702817204697
AUC: 0.6312634570935731
```




```
Confusion Matrix:
[[691300 405902]
 [409421 704558]]
True positives: 691300
False positives: 409421
True negatives: 704558
False negatives: 405902
```

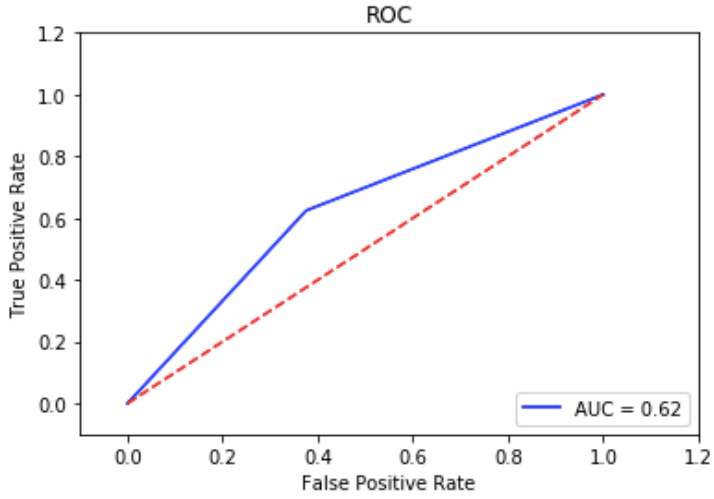
AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()

ada.fit(X_train_std, y_train)
pred_y_test = ada.predict(X_test_std)
```

```
ada.score(X_train_std, y_train)
```

```
0.6248706249927317
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.62      0.61      0.62   1097202
         1.0       0.62      0.64      0.63   1113979

    accuracy                           0.62   2211181
   macro avg       0.62      0.62      0.62   2211181
weighted avg       0.62      0.62      0.62   2211181

Accuracy: 0.6244350869512717
Precision: 0.6243250709240071
Recall: 0.6390793722323311
F-1: 0.6316160696700149
AUC: 0.6243231261755166
```




```
Confusion Matrix:
[[668818 428384]
 [402058 711921]]
True positives: 668818
False positives: 402058
True negatives: 711921
False negatives: 428384
```

RandomForest

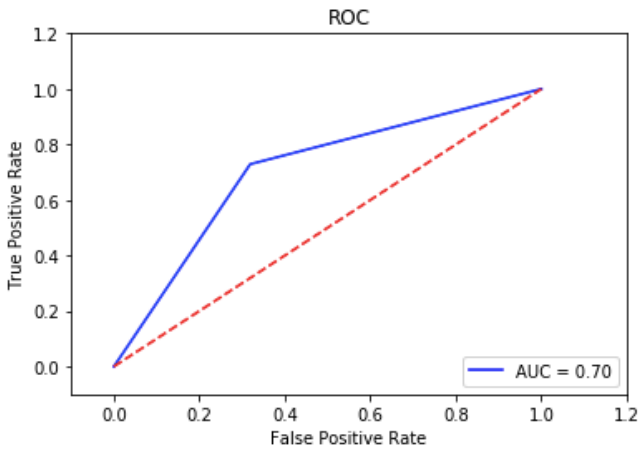
```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()

rf.fit(X_train_std, y_train)
pred_y_test = rf.predict(X_test_std)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/e
will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarnin
```

```
rf.score(X_train_std, y_train)
```

```
0.9862224823720496
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.68      0.75      0.72   1098649
         1.0       0.73      0.65      0.69   1112532

    accuracy                           0.70   2211181
   macro avg       0.71      0.70      0.70   2211181
weighted avg       0.71      0.70      0.70   2211181

Accuracy: 0.7026498509167726
Precision: 0.7289399883879005
Recall: 0.6511399222674045
F-1: 0.6878470206136476
AUC: 0.7029753016928798
```




```
Confusion Matrix:
[[829272 269377]
 [388118 724414]]
True positives: 829272
False positives: 388118
True negatives: 724414
False negatives: 269377
```

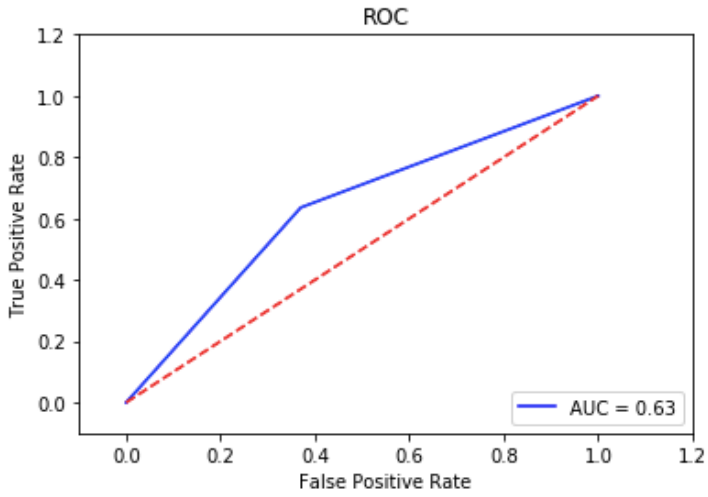
Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()

gb.fit(X_train_std, y_train)
pred_y_test = gb.predict(X_test_std)
```

```
gb.score(X_train_std, y_train)
```

```
0.6332519934411233
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.63      0.63      0.63   1097202
         1.0       0.64      0.64      0.64   1113979

    accuracy                           0.63   2211181
   macro avg       0.63      0.63      0.63   2211181
weighted avg       0.63      0.63      0.63   2211181

Accuracy: 0.6329640133485228
Precision: 0.6357099645821029
Recall: 0.6357938524873449
F-1: 0.6357519057674571
AUC: 0.6329423782206101
```




```
Confusion Matrix:
[[691337 405865]
 [405718 708261]]
True positives: 691337
False positives: 405718
True negatives: 708261
False negatives: 405865
```

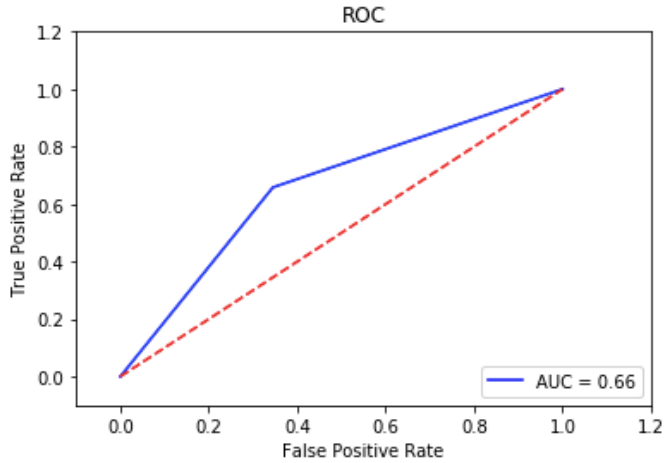
DecisionTree Classifier

```
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()

tree.fit(X_train_std, y_train)
pred_y_test = tree.predict(X_test_std)
```

```
tree.score(X_train_std, y_train)
```

```
1.0
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.66      0.65      0.65   1097202
         1.0       0.66      0.66      0.66   1113979

    accuracy                           0.66   2211181
   macro avg       0.66      0.66      0.66   2211181
weighted avg       0.66      0.66      0.66   2211181

Accuracy: 0.6566522595843579
Precision: 0.6581488504870954
Recall: 0.6626803557338155
F-1: 0.660406829758797
AUC: 0.6566061726427102
```




```
Confusion Matrix:
[[713765 383437]
 [375767 738212]]
True positives: 713765
False positives: 375767
True negatives: 738212
False negatives: 383437
```

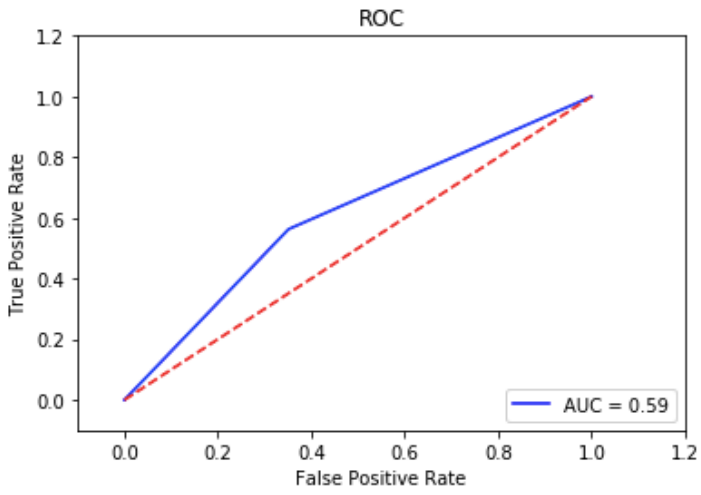
Navie Bayes

```
from sklearn.naive_bayes import GaussianNB

gnb_model = GaussianNB()

gnb_model.fit (X_train_std, y_train)
pred_y_test = gnb_model.predict(X_test_std)
```

```
gnb_model.score(X_train_std, y_train)
```

```
0.5865595357617717
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.65      0.37      0.47   1098279
         1.0       0.56      0.80      0.66   1112902

    accuracy                           0.59   2211181
   macro avg       0.61      0.59      0.57   2211181
weighted avg       0.61      0.59      0.57   2211181

Accuracy: 0.5868207984782793
Precision: 0.5626520811300563
Recall: 0.804078885652106
F-1: 0.6620420103161426
AUC: 0.5853744606129723
```




```
Confusion Matrix:
[[402706 695573]
 [218041 894861]]
True positives: 402706
False positives: 218041
True negatives: 894861
False negatives: 695573
```

SVM

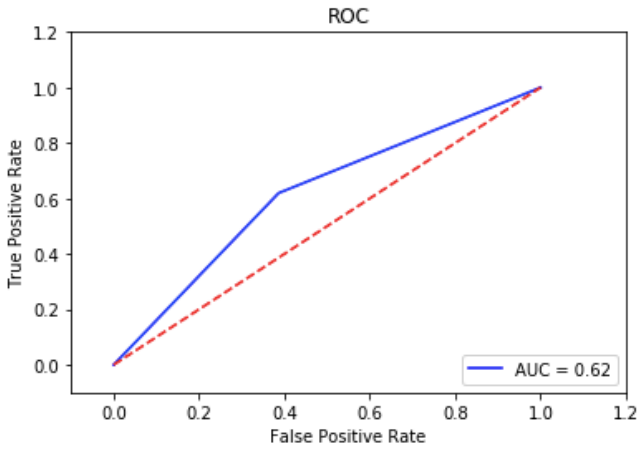
```
from sklearn.svm import LinearSVC

svm_model = LinearSVC()

svm_model.fit(X_train_std, y_train)
pred_y_test = svm_model.predict(X_test_std)
```

```
/opt/anaconda3/lib/python3.7/site-packages/sklearn/
se the number of iterations.
  "the number of iterations.", ConvergenceWarning)
```

```
svm_model.score(X_train_std, y_train)
```

```
0.6158378267324621
```

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.61      0.61      0.61   1098279
         1.0       0.62      0.62      0.62   1112902

    accuracy                           0.62   2211181
   macro avg       0.62      0.62      0.62   2211181
weighted avg       0.62      0.62      0.62   2211181

Accuracy: 0.6165040311037405
Precision: 0.6192140744094038
Recall: 0.6182242461600392
F-1: 0.6187187644023785
AUC: 0.6164925792300506
```




```
Confusion Matrix:
[[675179 423100]
 [424879 688023]]
True positives: 675179
False positives: 424879
True negatives: 688023
False negatives: 423100
```

模型比較總表（未調參）

模型	score	precision	recall	F1-score	Confusion Matrix	AUC
XGBoost	0.6316	0.6344	0.6324	0.6334	[[691300 405902] [409421 704558]]	0.6312
AdaBoost	0.6248	0.6243	0.639	0.6316	[[668818 428384] [402058 711921]]	0.6243
RandomForest	0.9862	0.7289	0.6511	0.6878	[[829272 269377] [388118 724414]]	0.7029
Gradient Boost	0.6332	0.6357	0.6357	0.6357	[[691337 405865] [405718 708261]]	0.6329
DecisionTree classifier	1	0.6581	0.6626	0.6604	[[713765 383437] [375767 738212]]	0.6566
Navie Bayes	0.5865	0.5626	0.804	0.662	[[402706 695573] [218041 894861]]	0.5853
SVM	0.6158	0.6192	0.6182	0.6187	[[675179 423100] [424879 688023]]	0.6164

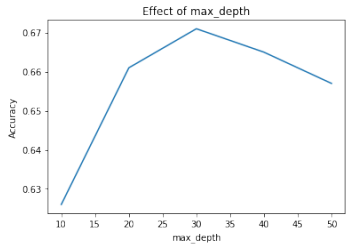
觀察以上幾個模型的預測結果，我們選擇使用 XGBoost、RandomForest、DecisionTree 三種模型進行參數的調整

四、DecisionTree 調參

→主要調整的參數: Max_depth、min_samples_leaf、max_feature、min_samples_split

- **Max_depth：**

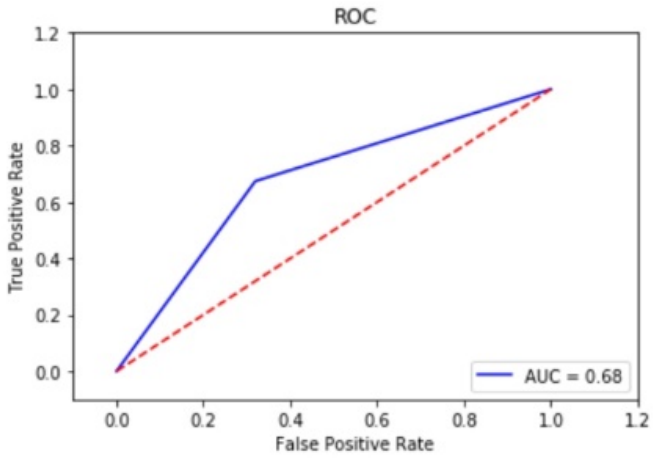
若直接不調整，決策樹在建立子樹的時候就不會限制子樹的深度。
資料或特徵較少時可以不管這個值，如果模型樣本與特徵較多的情況下最好限制最大深度，取值取決於資料的分佈。常用的取值界於 10~100 之間，常用來解決過度擬合問題。

Max_depth	Accuracy	折線圖
10	0.626	
20	0.661	
30	0.671	
40	0.665	
50	0.657	

根據上表，選擇調整 max_depth =30，調整參數之後的分數約 0.838

```
dt = DecisionTreeClassifier(max_depth = 30, splitter = 'random')

dt.fit(X_train_std, y_train)
pred_y_test = dt.predict(X_test_std)
print(dt.score(X_train_std, y_train))
```

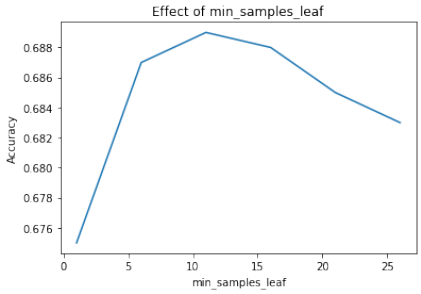
```
0.838033344833334
```



- **min_samples_leaf:**

預設值=1

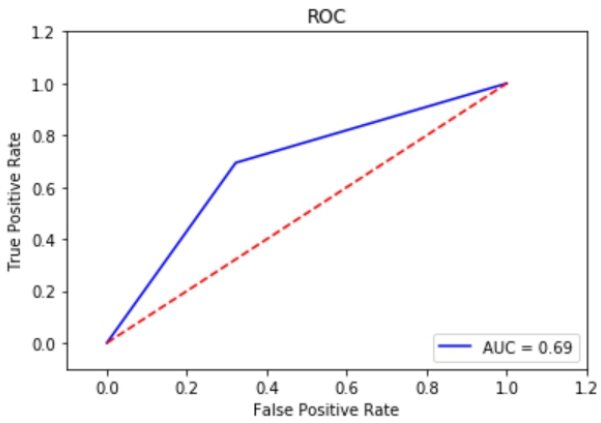
如這個值限制了葉子節點最少的樣本數，如果某葉子節點數目小於樣本數，則會和兄弟節點一起被修枝，可以輸入最少的樣本數的整數，或者最少樣本數佔樣本總數的百分比。如果樣本量不大，則不需理會這個值。如果樣本量數量非常大，可以增大該值。

min samples leaf	Accuracy	折線圖
1	0.675	
6	0.687	
11	0.689	
16	0.688	
21	0.685	
26	0.683	

根據上表，選擇調整 min_samples_leaf=11，調整參數之後的分數約 0.7324

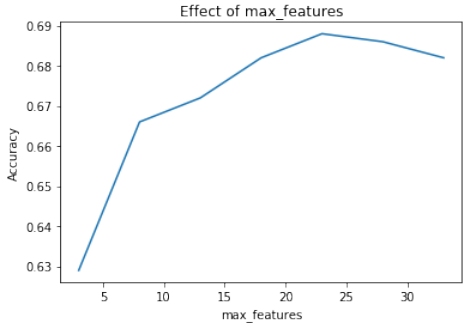
```
dt = DecisionTreeClassifier(max_depth = 30, splitter = 'random', min_samples_leaf = 11)

dt.fit(X_train_std, y_train)
pred_y_test = dt.predict(X_test_std)
print(dt.score(X_train_std, y_train))
```

```
0.7324660523857333
```



- **max_feature:**

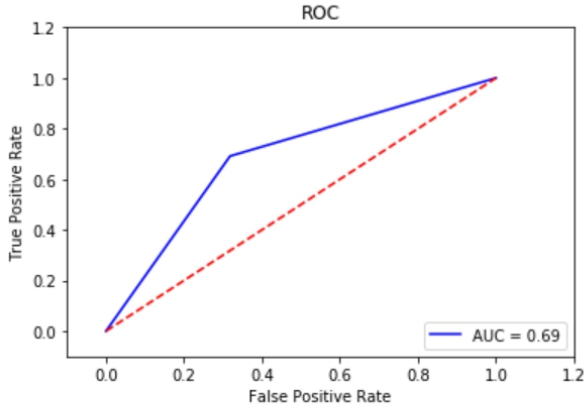
預設="None"
可以使用很多種類型的值，劃分時要考慮所有的特徵數。
"log2"表示劃分時最多考慮 log2N 個特徵，"sqrt"或"auto"表示劃分時最多考慮 sqrt(N)個特徵。如果是整數，代表考慮的特徵絕對數。如果是浮點數，代表考慮特徵百分比，就是考慮(百分比 xN)取整數後的特徵數。其中 N 爲樣本總特徵數。如果樣本特徵數不多，比如小於 50，用預設的"None"就可以了，如果特徵數非常多，可以嘗試用上述值作為控制劃分時考慮的最大特徵數來控制決策樹的生成時間。

min samples leaf	Accuracy	折線圖
3	0.629	
8	0.666	
13	0.672	
18	0.682	
23	0.688	
28	0.686	
33	0.682	

根據上表，選擇調整 max features=33，調整參數之後的分數約 0.7361

```
dt = DecisionTreeClassifier(max_depth = 30, splitter = 'random', min_samples_leaf = 11, max_features = 33)

dt.fit(X_train_std, y_train)
pred_y_test = dt.predict(X_test_std)
print(dt.score(X_train_std, y_train))
```

```
0.7361439851766284
```



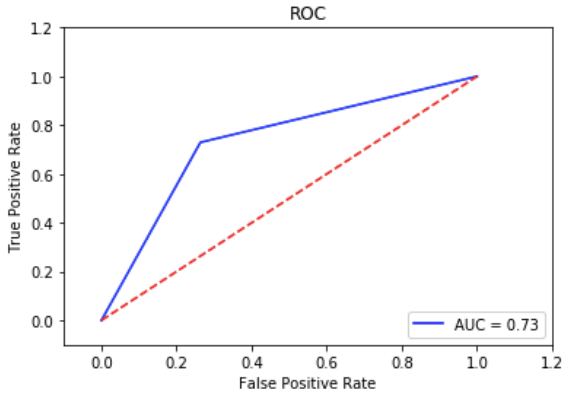
→ DecisionTreen 調參過程：

	precision	recall	F1-score	Confusion Matrix	AUC
dt 調參 1 max_depth =30	0.6741	0.6936	0.6837	[[725474 373092] [340844 771771]]	0.677
dt 調參 2 min_samples_leaf=11	0.6941	0.6712	0.6825	[[769428 329138] [365744 746871]]	0.6858
dt 調參 3 max_feature=33	0.6913	0.6812	0.6862	[[760153 338413] [354671 757944]]	0.6865

→最終使用的參數為：

max_depth=30, splitter= 'random', min_samples_leaf= 11, max_features= 33

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.74      0.72      0.73   1098273
         1.0       0.73      0.75      0.74   1112908

    accuracy                           0.73   2211181
   macro avg       0.73      0.73      0.73   2211181
weighted avg       0.73      0.73      0.73   2211181

Accuracy: 0.7328527153588964
Precision: 0.7293091440510578
Recall: 0.7461659005057022
F-1: 0.7376412311756537
AUC: 0.7327640131579758
```




```
Confusion Matrix:
[[790056 308217]
 [282494 830414]]
True positives: 790056
False positives: 282494
True negatives: 830414
False negatives: 308217
```

→ 在 kaggle 上獲得的分數：

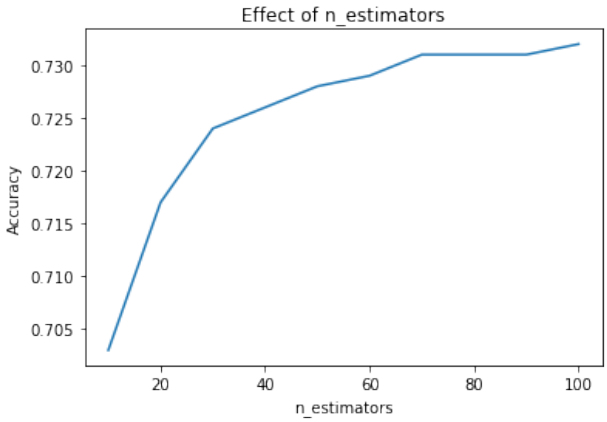
Submission and Description	Private Score	Public Score	Use for Final Score
submission_decisiontree3.csv 5 minutes ago by chinghsuan dt調參3 proba	0.58802	0.58583	☐

五、RandomForest 調參

→ 主要調整的參數:n_estimators、max_depth、oob_score

- **n_estimators:**

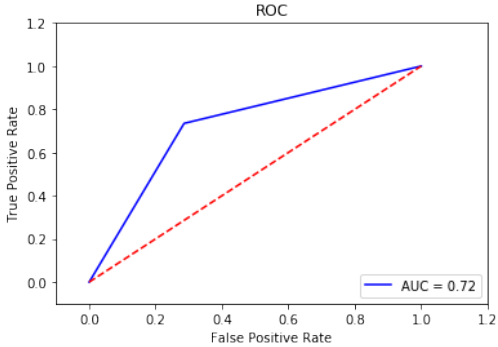
預設值=100
森林中樹木的數量(base estimator 的數量)
通常此值越大，模型效果往往越好，但是電腦的計算量、內存消耗量越大所以訓練所需時間也越長。在調整的時候需要在訓練時間和模型效果之間取一個平衡。另外，如果 n_estimators 太小，容易欠擬合，n_estimators 太大，又容易過擬合，所以要選擇一個適中的數值。

n_estimators	Accuracy	折線圖
10	0.703	
20	0.717	
30	0.724	
40	0.726	
50	0.728	
60	0.729	
70	0.731	
80	0.731	
90	0.731	
100	0.732	
110	0.703	

根據上表，調整 n_estimators=30，調整參數之後的分數約 0.9991

```
rf.score(X_train_std, y_train)
```

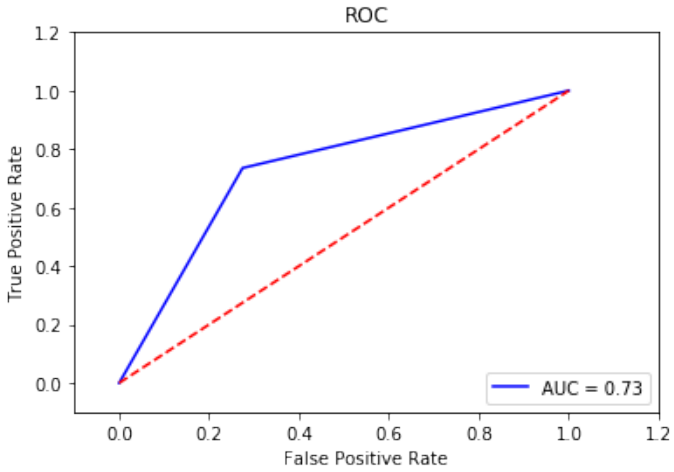
```
0.9991028061293711
```



根據上表，選擇調整 n_estimators=70，調整參數之後的分數約 0.9999，相較於 n_estimators=30 的各項數值表現都較優異因此選擇調整參數 n_estimators=70

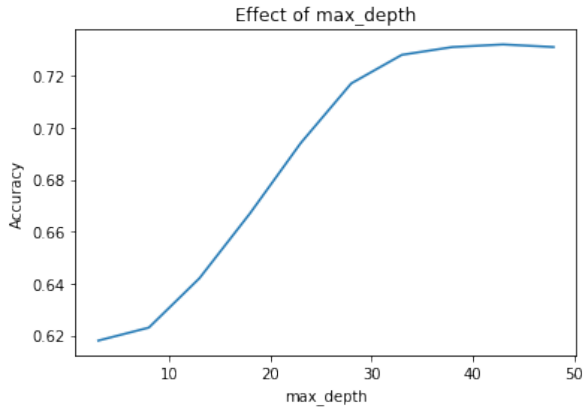
```
rf.score(X_train_std, y_train)
```

```
0.9999614297731141
```



- **max_depth:**

Max_depth	Accuracy	折線圖
3	0.618	
8	0.623	
13	0.642	
18	0.667	
23	0.694	
28	0.717	
33	0.728	
38	0.731	
43	0.732	
48	0.731	

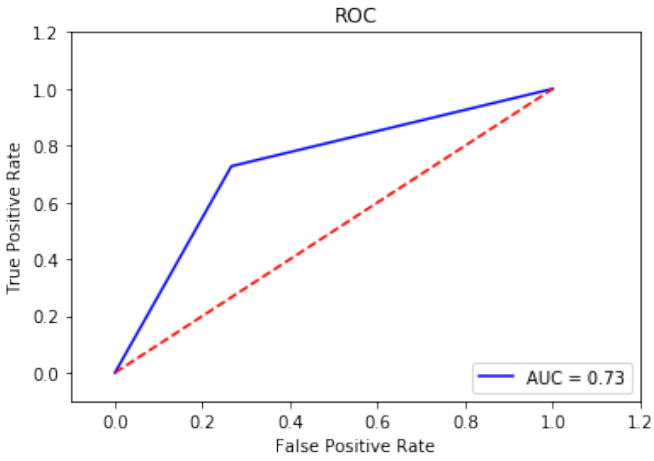


根據上表，調整 max_depth=38，調整參數之後的分數約 0.9886

```
rf = RandomForestClassifier(n_estimators = 70,max_depth = 38)

rf.fit(X_train_std, y_train)
pred_y_test = rf.predict(X_test_std)
```

```
rf.score(X_train_std, y_train)
```

```
0.988610153854503
```



- **oob_score:**

預設值=False

袋外資料(Out Of Bag,簡稱 OOB):放回取樣中大約 36.8%的沒有被取樣到的資料，這些資料沒有參與訓練集模型的擬合，可以用來檢測模型的泛化能力。

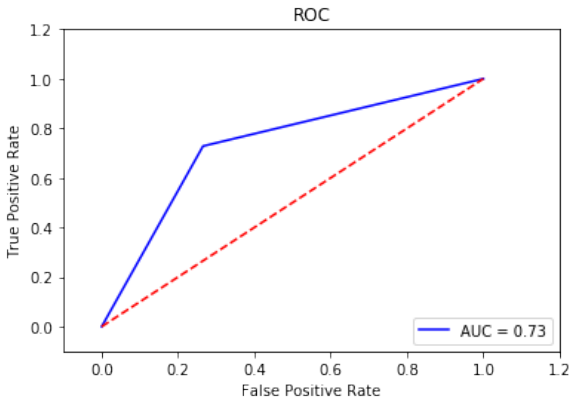
調整 oob_score=Ture or False 代表是否採用袋外樣本來評估模型的好壞。嘗試設為 True(為 True 時不須再劃分訓練集與測試集)，因為袋外分數反應了一個模型擬合後的泛化能力。針對單個模型引數訓練，可以用 cross validation(cv)來進行，但會特別消耗時間，且對於這種 RandomForest 情況也沒有大的必要，所以就用這個資料對決策樹模型進行驗證，算是一個簡單的交叉驗證，效能消耗小，但是效果不錯。

oob_score=Ture 調整 oob_score=Ture，調整參數之後的分數約 0.9889

→將 oob_score 調為 Ture 之後可以看出各項表現都有成長。

```
rf = RandomForestClassifier(n_estimators = 70,max_depth = 38, oob_score = True)

rf.fit(X_train_std, y_train)
pred_y_test = rf.predict(X_test_std)
print(rf.score(X_train_std, y_train))
```

```
0.9889464319632827
```



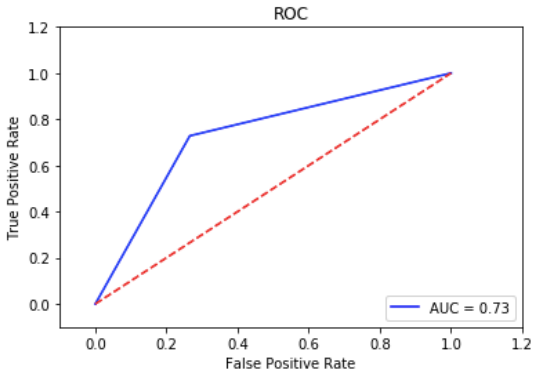
→ RandomForest 調參過程：

	precision	recall	F1-score	Confusion Matrix	AUC
rf 調參 1 n_estimators=30	0.7346	0.7075	0.7208	[[813520 284449] [325597 787615]]	0.7242
rf 調參 2(選用) n_estimators=70	0.7351	0.7436	0.7304	[[806817 291152] [305151 808061]]	0.7303
rf 調參 3 max_depth=38	0.7273	0.6812	0.7353	[[787986 310287] [285306 827602]]	0.7305
rf 調參 4 oob_score=Ture	0.7281	0.7435	0.7357	[[789345 308928] [285395 827513]]	0.7311

→最終使用的參數為：

n_estimators = 70,max_depth = 38, oob_score = True

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.73      0.72      0.73   1098273
         1.0       0.73      0.74      0.74   1112908

    accuracy                           0.73   2211181
   macro avg       0.73      0.73      0.73   2211181
weighted avg       0.73      0.73      0.73   2211181

Accuracy: 0.7312191991519464
Precision: 0.7281618667401123
Recall: 0.7435592160358269
F-1: 0.7357799967901825
AUC: 0.7311369809115382
```




```
Confusion Matrix:
[[789345 308928]
 [285395 827513]]
True positives: 789345
False positives: 285395
True negatives: 827513
False negatives: 308928
```

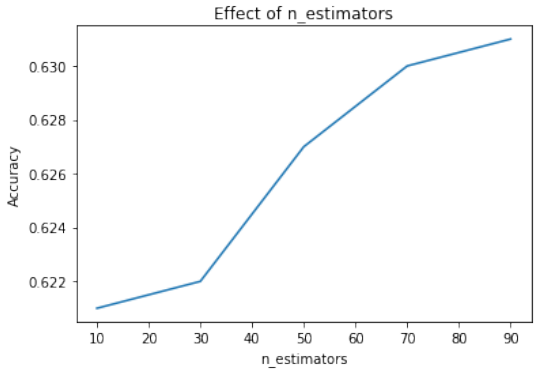
→ 在 kaggle 上獲得的分數:

Submission and Description	Private Score	Public Score	Use for Final Score
submission_rf4.csv a minute ago by chinghsuan rf調參4_proba	0.63765	0.63636	☐

六、XGBoost 調參

→ 主要調整的參數: n_estimators、max_depth

- **n_estimators:**

n_estimators	Accuracy	折線圖
10	0.621	
30	0.622	
50	0.627	
70	0.63	
90	0.631	

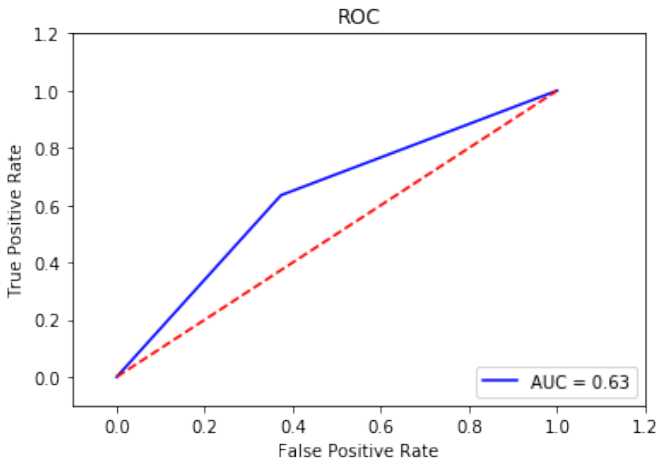
根據上表，調整 n_estimators=90，調整參數之後的分數約 0.631

```
xgb = XGBClassifier(n_estimators = 90)

xgb.fit(X_train_std, y_train)
pred_y_test = xgb.predict(X_test_std)
```

```
[01:48:47] WARNING: src/learner.cc:686: Tree method
havior (exact greedy algorithm on single machine),
```

```
xgb.score(X_train_std, y_train)
```

```
0.6310374034290676
```



- **max_depth**

Max_depth	Accuracy
3	0.631
8	0.655
13	0.7
18	0.733
23	0.737

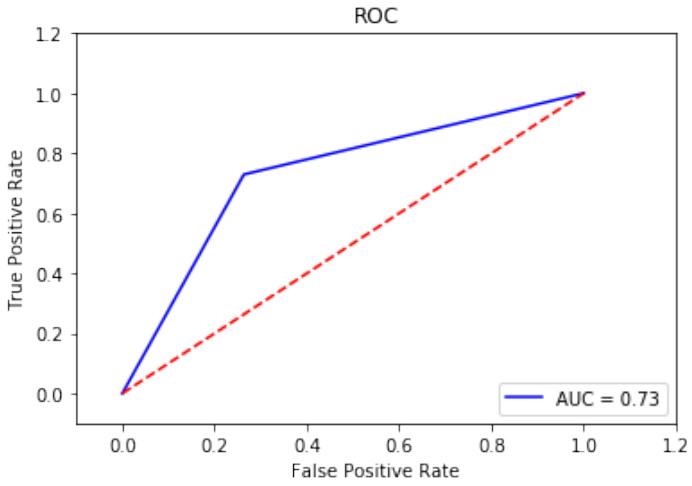
max_depth=18 調整參數之後的分數約 0.8327

```
xgb = XGBClassifier(n_estimators = 90, max_depth=18)

xgb.fit(X_train_std, y_train)
pred_y_test = xgb.predict(X_test_std)
```

```
[18:09:41] WARNING: src/learner.cc:686: Tree method is automatically selected t
havior (exact greedy algorithm on single machine), set tree_method to 'exact'.
```

```
xgb.score(X_train_std, y_train)
```

```
0.8327393389179404
```



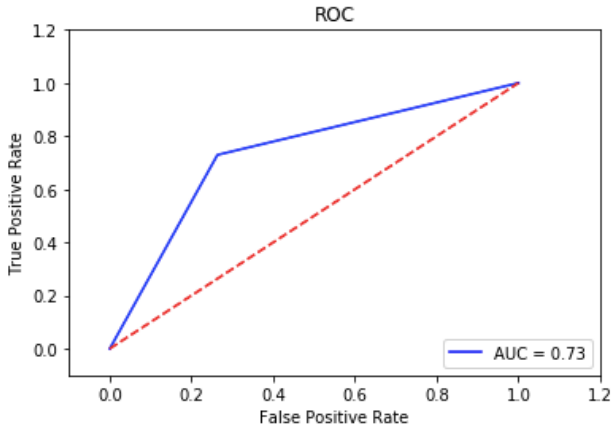
→ XGBoost 調參過程：

	precision	recall	F1-score	Confusion Matrix	AUC
xgb 調參 1 n_estimators=90	0.6345	0.6281	0.6313	[[695733 402540] [413849 699059]]	0.6308
xgb 調參 2 max_depth =18	0.7293	0.7461	0.7376	[[790056 308217] [282494 830414]]	0.7327

→最終使用的參數為：

n_estimators = 90, max_depth=18

```
Classification Report:
              precision    recall  f1-score   support

         0.0       0.74      0.72      0.73   1098273
         1.0       0.73      0.75      0.74   1112908

    accuracy                           0.73   2211181
   macro avg       0.73      0.73      0.73   2211181
weighted avg       0.73      0.73      0.73   2211181

Accuracy: 0.7328527153588964
Precision: 0.7293091440510578
Recall: 0.7461659005057022
F-1: 0.7376412311756537
AUC: 0.7327640131579758
```




```
Confusion Matrix:
[[790056 308217]
 [282494 830414]]
True positives: 790056
False positives: 282494
True negatives: 830414
False negatives: 308217
```

→ 在 kaggle 上獲得的分數：

Submission and Description	Private Score	Public Score	Use for Final Score
submission_xgb2.csv 12 minutes ago by chinghsuan xgb調參2_proba	0.64064	0.63757	☐

● 總結

	XGBoost	RandomForest	DecisionTree
precision	0.73	0.73	0.69
recall	0.75	0.74	0.68
F1-score	0.74	0.74	0.69
Confusion Matrix	[[790056 308217] [282494 830414]]	[[789345 308928] [285395 827513]]	[[760153 338413] [354671 757944]]
AUC	**0.73**	**0.73**	**0.69**
Kaggle: Private Score	0.6406	0.6376	0.588
Kaggle: Public Score	0.6376	0.6363	0.5858

從上表結果可以看出，雖然三者皆為未調參時表現就相對其他模型較佳的演算法，但原本在未調參前表現就較其他兩者相對較差的 DecisionTree Classifier，在調參後表現也尚未明顯超越其他模型。

值得注意的是，一開始未調參時我們廣泛測試了 RandomForest、DecisionTree、XGBoost、AdaBoost、SVM、Gradient Boosting、Naïve Bayes 等模型，其中表現較突出的三者（RandomForest、DecisionTree、XGBoost）皆為有關於「樹」的演算法模型。三者模型的特點各自不同，我們也在電腦效能能夠接受的情況下（模型跑得動的情況下），找到了他們較佳的參數組合。最後可以觀察出，三者模型**表現優劣排序依序為：XGBoost→RandomForest→DecisionTree**。另外，我們也發現，XGBoost 在越高的 n_estimators 的情況下有機會繼續增加 AUC 分數，然而受限於電腦效能不佳，本次無法測試到更高的參數值。

組員分工表

	資料清整	資料視覺化	模型建置	調整參數	文本撰寫
劉謦瑄	v		v	v	v
楊怡芊		v			v