資料探勘應用－以LendingClub為例

Data mining on LendingClub dataset

指導老師：徐立群 教授
第二組：蘇佳成、林予捷、張文彥、涂祐誠

目錄

第壹章 研究背景與動機

近年來隨著科技快速滲透金融產業，**P2P借貸**這種商業模式因應而生，P2P借貸又稱互聯網借貸，不同於以往向銀行申請貸款的常規模式，P2P借貸帶著Web2.0的社群特色，讓使用者根據自身需求選擇擔任債權人或債務人的角色，在網路平台上借款、放款以及還款，目前知名的借貸平台有英國的Zopa，美國的Prosper Marketplace以及為本文提供資料集的Lending Club，現代的P2P借貸快速又方便，同時亦存在比傳統信貸還高的風險，本文試著藉由資料集提供的不具名用戶特徵，利用四種機器學習模型(XGB、Gradient Boosting、Random Forest、Logistic Regression)預測出債務人是否會違約，提供未來想要擔任債權人的用戶一個降低風險的參考。

第貳章 模型方法介紹

一、羅吉斯回歸(Logistic Regression)

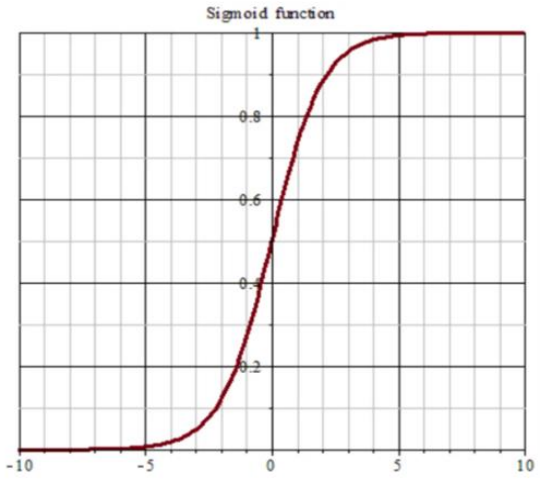
羅吉斯回歸屬於監督式學習，也是線性分類器的一種，適合用來處理二元分類問題。其運作原理是用特徵對應變數建立迴歸方程式，與一般迴歸不同的是，羅吉斯回歸改用Odds Ratio取對數建立回歸方程式。

$$logit(Odds) = \ln(\frac{p}{1-p}) = w_0 + w_1 X$$

$$where\ Odds = \frac{p}{1-p}$$

而後經過Sigmoid函數的轉換，讓結果介於0-1之間

$$S(X) = \frac{1}{1+e^{-X}}$$

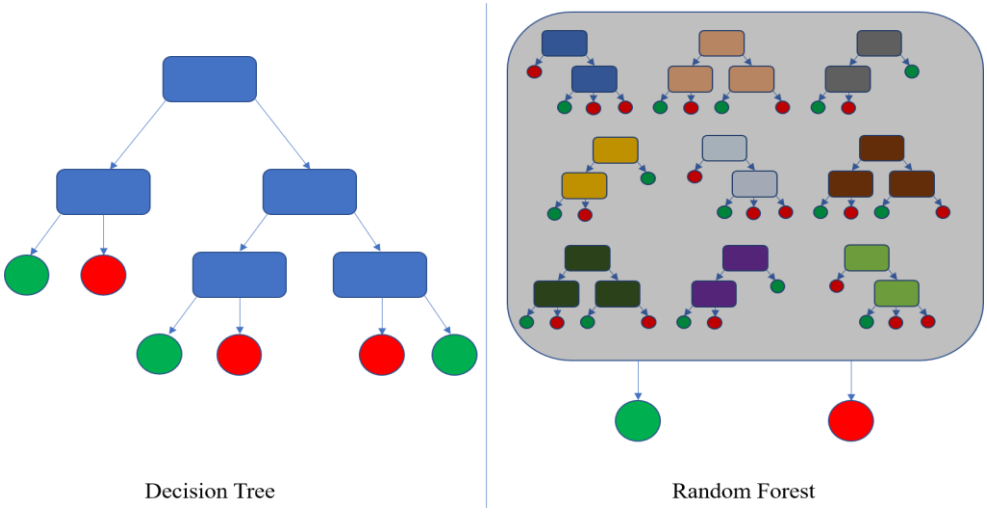


將每個原始變數X得到的預測值放入上式Sigmoid函數中變數X的部分，即可得每個預測值相對應的值，將事件發生機率>50%的分類為1，發生機率<50%則為0。

羅吉斯回歸之優缺點	
優點	**缺點**
● 運算速度較快，也較不佔用儲存空間 ● 容易理解，容易使用與解釋	● 預測結果呈“S”型，因此從log(odds)向概率轉化的過程是非線性的，在兩端隨著log(odds)值的變化，機率變化很小，但中間機率的變化很大，導致中間區間的變數變化對目標機率的影響較無區分度，預測容易失準 ● 當特徵很多時，羅吉斯回歸的表現不是很好 ● 容易underfitting

二、隨機森林(Random Forest)

由多棵決策樹所組成的模型(model)，生成指定的決策樹數量，再由所有決策樹所估計的結果投票或取平均來做最後的結果預測。隨機取樣多個決策樹的隨機森林在大數法則下，樹的結果會趨向一致，雖然單棵樹會過度配適，但只要一開始抽樣的隨機性是足夠的，考慮各種組合的集合會是更接近真實的狀況，預測錯誤率大幅降低。

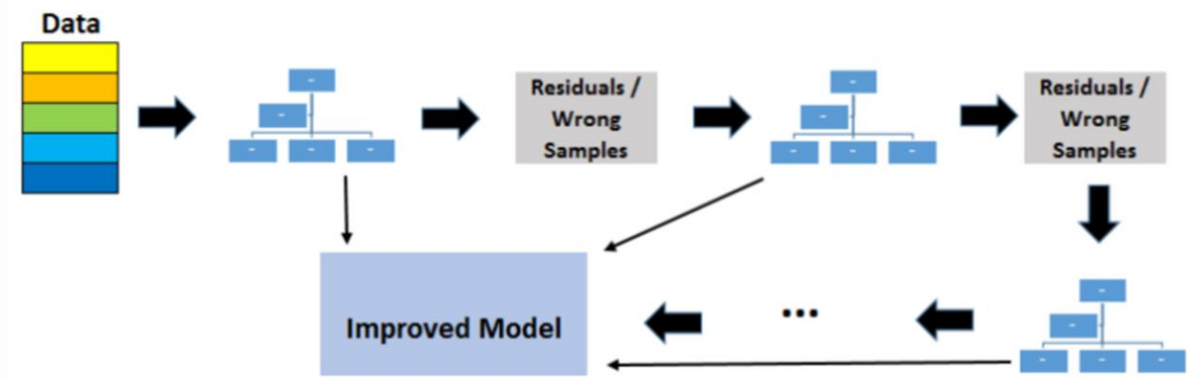


Decision Tree　　Random Forest

隨機森林之優缺點	
優點	**缺點**
● 抗Overfitting能力強 ● 能處理特徵值數量大的資料 ● 和其他模型相比效能高，速度快	● 可能有很多相似的決策樹，會掩蓋結果 ● 在處理小資料方面，預測能力較不好

三、GBDT(Gradient Boosting Decision Tree)

是一種基於決策樹(Decision Tree)的機器學習方法，而決策樹具有分類速度快，且容易解釋的特性，不過也很容易發生過度配適，雖然能夠進行剪枝，可是效果可能仍不如預期好。因此GBDT藉助Boosting方法控制learning rate每次訓練的樣本量，並透過每次訓練後進行模型修正，得以避免Decision Tree容易產生過度配適的問題。

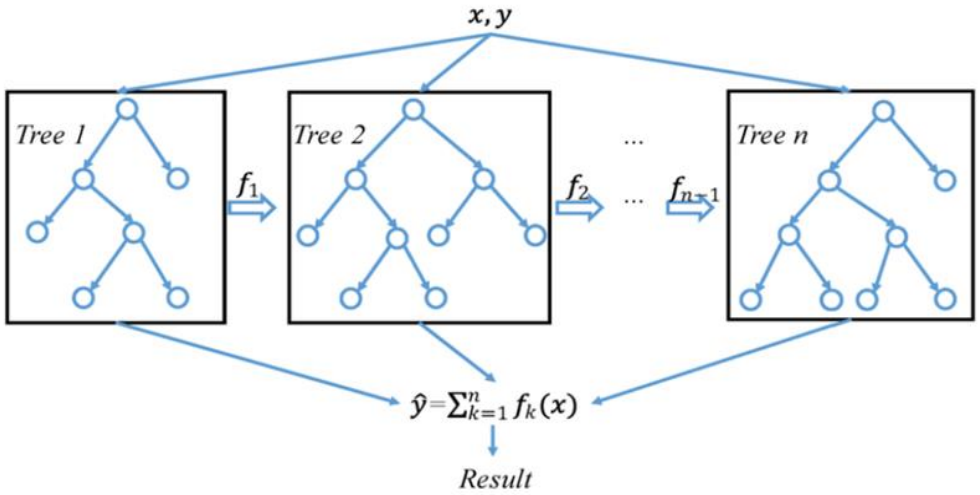


Source from: An Introduction to Gradient Boosting Decision Trees

GBDT之優缺點	
優點	**缺點**
● 可以有效避免Overfitting ● 主要還是基於決策樹的運作原理，因此也比較容易理解	● 不適合處理高維度資料

四、XGBoost(eXtreme Gradient Boosting)

XGboost的全名為eXtreme Gradient Boosting，由華盛頓大學博士生陳天奇所提出，是一種Gradient Boosted Tree(GBDT)，原理是透過不斷迭代生成新的樹，將成百上千個數模型組合成一個準確率很高的模型，主要用於監督式學習中的分類問題或回歸問題上。與其他演算法相比，XGBoost除了精準度很高之外，訓練的速度也非常快，是目前資料科學競賽中最常見到的算法之一，同時也是多數得獎者所使用的模型。下圖為XGBoost示意圖[Wang et al. 2019]：



XGBoost之優缺點	
優點	**缺點**
● 訓練時間短 ● 在低維度的特徵資料下，通常都有不錯的表現 ● 多種防止過度配適的機制，包括Shrinkage、Column Subsampling等 ● 支援使用者自定義目標函式和評估函式	● 不適合處理高維度特徵資料 ● 調整參數不易，需要對原理十分清楚，才比較好上手

第參章 變數選擇

一、資料集介紹

本研究使用的資料集為”accepted_2007_to_2018Q4.csv”，資料來源為Kaggle，資料年度從2007年Q1至2018年Q4，總樣本數為2,260,701筆樣本，特徵屬性有151個。

```
loans.shape
```

```
(2260701, 151)
```

二、資料前處理

loan_stasus欄位中的Fully Paid代表個案已繳清本金及利息款項；Charged Off代表個案逾期120天以上沒有繳款，判斷為舞弊。本研究的目的為預測個案是否為舞弊，因此在loan_stasus欄位中，僅保留Fully Paid及Charged Off兩種狀態，其餘刪除。樣本數降至1,345,310筆，特徵數維持151個。

```
loans['loan_status'].value_counts(dropna=False)
```

```
Fully Paid                                         1076751
Current                                             878317
Charged Off                                         268559
Late (31-120 days)                                   21467
In Grace Period                                       8436
Late (16-30 days)                                     4349
Does not meet the credit policy. Status:Fully Paid    1988
Does not meet the credit policy. Status:Charged Off    761
Default                                                 40
NaN                                                     33
Name: loan_status, dtype: int64
```

```
# keep the loans with status "Fully Paid" or "Charged Off."
loans = loans.loc[loans['loan_status'].isin(['Fully Paid', 'Charged Off'])]
```

```
loans.shape
```

```
(1345310, 151)
```

計算各欄位中的缺失數量，並刪除缺失值過多的欄位，樣本數維持1,345,310筆，特徵數降至56個。

```
loans.shape
```

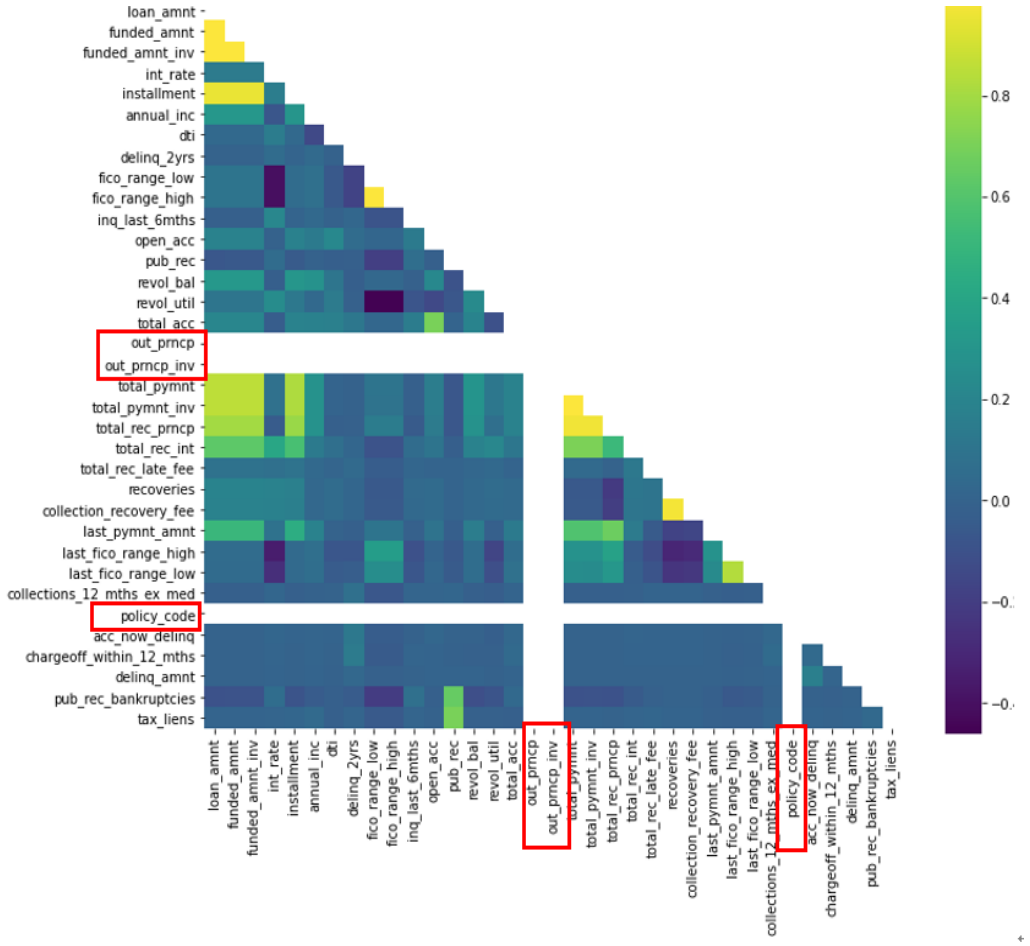
```
(1345310, 56)
```

```
member_id                                1345310
next_pymnt_d                             1345310
orig_projected_additional_accrued_interest  1341551
hardship_start_date                      1339556
hardship_end_date                        1339556
payment_plan_start_date                  1339556
hardship_length                          1339556
hardship_dpd                             1339556
hardship_loan_status                     1339556
hardship_last_payment_amount             1339556
hardship_payoff_balance_amount           1339556
                                           ⋮
total_bal_ex_mort                          47281
total_bc_limit                             47281
title                                      16659
last_pymnt_d                                2313
revol_util                                   857
pub_rec_bankruptcies                         697
dti                                          374
collections_12_mths_ex_med                    56
```

整筆刪除含有空值欄位的樣本，樣本數降至1,341,026筆，特徵數維持56個。

```
loans = loans.dropna()
loans.shape
```

```
(1341026, 56)
```

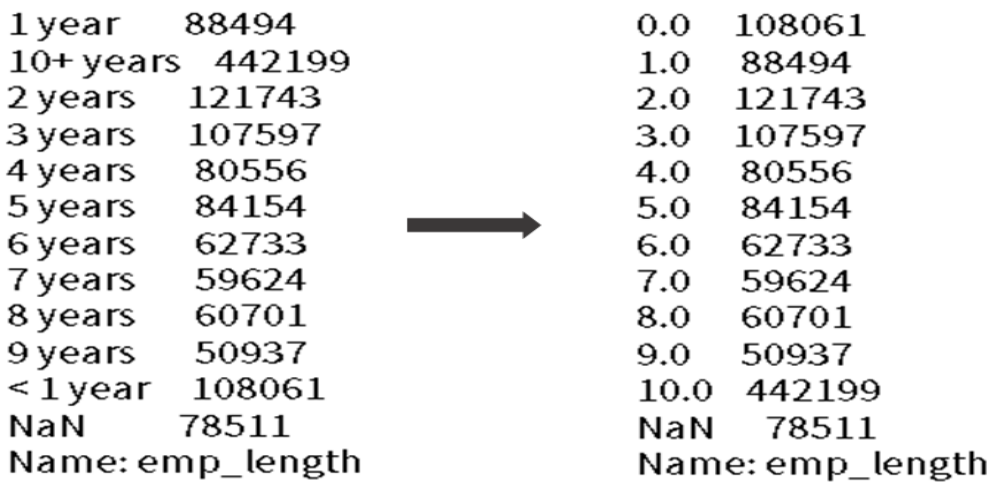


在此資料集中，有許多重複意義的特徵欄位，為了提升後續訓練模型的效率，我們決定刪除這些相同意義的欄位。經由我們人工判斷，我們也刪除了一些不重要之屬性特徵。繪製熱力圖之後發現，欄位'out_prncp'、'out_prncp_inv'、 'policy_code'與其它屬性沒有相關性，判斷為不重要之特徵，因此也一併刪除。樣本數維持1,341,026筆，特徵數降至34個。

```
loans.shape
```

```
(1341026, 34)
```

接著對一些欄位做型態轉換，此過程不影響樣本數及特徵數。下圖以欄位'emp_length'為例。



最後對類別型態的屬性欄位進行處理，樣本數維持1,341,026筆，特徵數上升至83個。

```
loans = pd.get_dummies(loans, columns = ['sub_grade', 'home_ownership',
                                         'initial_list_status', 'application_type', 'disbursement_me
```

```
loans.shape
```

```
(1341026, 83)
```

三、敘述型統計

下表為本組所列之重要變數，其中標底色為在各模型中feature importance較高的變數，各模型feature importance的圖表會在後面章節呈現。

變數名稱	變數說明
loan_amnt	申請的貸款金額
term	貸款償還期數，通常是36或者60
int_rate	貸款利率
installment	如果貸款發生，每月的欠款
grade	LC指定的貸款等級
sub_grade	LC指定的貸款子等級

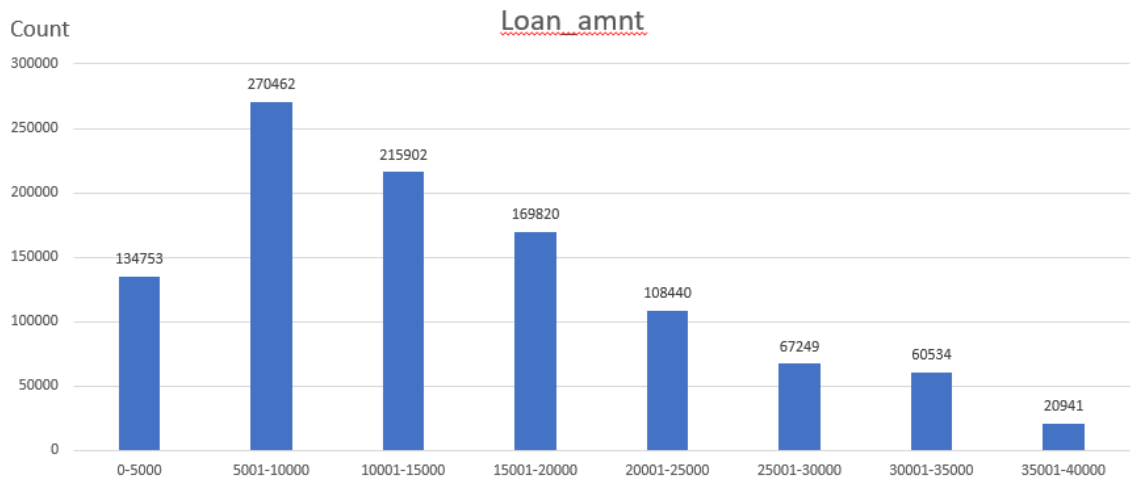
變數名稱	變數說明
emp_title	借款人在申請貸款時提供的職位名稱
emp_length	工作年限以年為單位。取值範圍為0~10，其中0表示少於1年，10表示10年或10年以上
recoveries	貸款回收金額
last pymnt amnt	最近一次支付金額
home_ownership	借款人在登記時提供或在信用報告中取得的自置居所狀況：RENT, OWN,MORTGAGE, OTHER
last fico range high	最近的fico分數區間上緣
annual_inc	借款人在登記時自行申報的年收入
verification_status	收入是否被LC驗證、未驗證或收入來源是否被驗證
issue_d	貸款發放的月份
loan_status	貸款的當前狀態
purpose	借款人為貸款請求提供類別
zip_code	借款人在貸款申請中提供的郵遞區號的前3個數字
addr_state	借款人在貸款申請中提供的狀態
dti	一種比率，是指借款人每月就債務總額所繳付的債務總額除以借款人自行申報的每月收入而計算的比率
earliest_cr_line	借款人最早公佈的信用額度開通的月份
open_acc	借款人信用檔案中的信用額度
pub_rec	不良公共記錄數量
total rec int	至今收到的利息金額
revol_util	迴圈額度利用率，或借款人使用的信貸金額相對於所有可用的迴圈信貸
total_acc	借款人信用檔案中當前信用額度的總數
initial_list_status	貸款的初始上市狀態。可能的值是W, F
application_type	指示貸款是單獨申請還是與兩個共同借款人的聯合申請

變數名稱	變數說明
pub_rec_bankruptcies	公開破產記錄數量
Y	借款人是否與債務清算公司合作

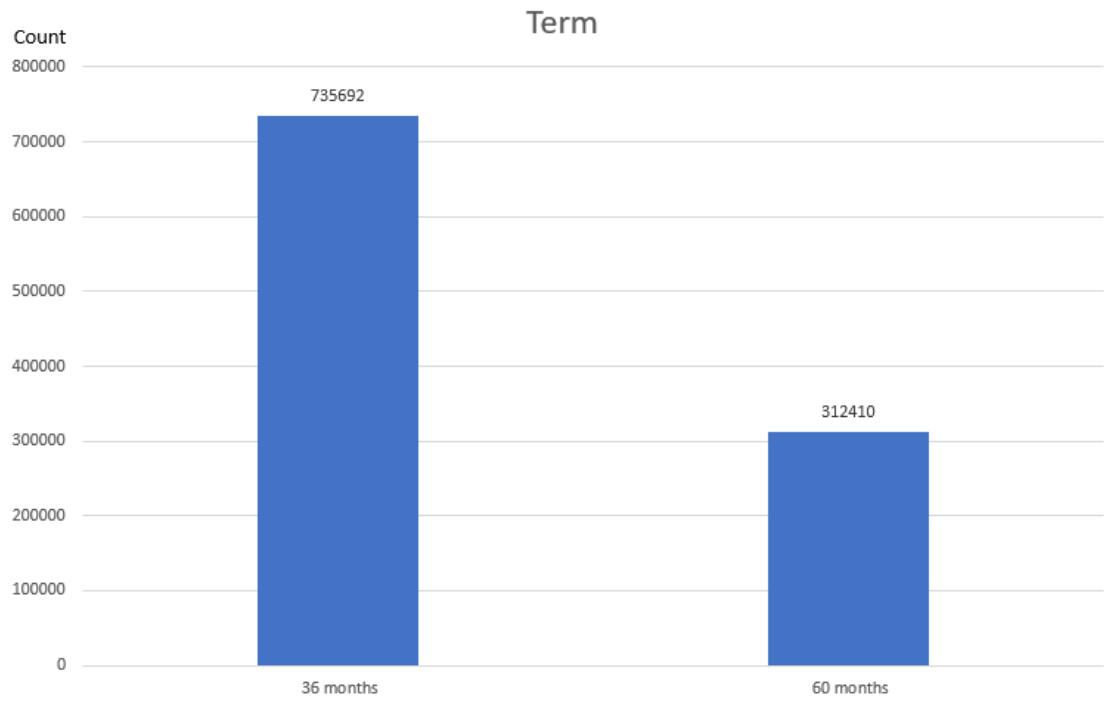
feature	count	mean	std	min	25%	50%	75%	max
loan_amnt	269360.0	15547.816955	8814.732243	5.0000e+02	9000.0000	14300.0000	20400.0000	4.0000e+04
recoveries	269360.0	1207.539935	1847.546420	0	0	588.6200	1720.2650	3.985955e+04
int_rate	269360.0	15.707638	4.906016	5.310000e+00	12.2900	15.0500	18.5500	3.099000e+01
last_pymnt_amnt	269360.0	471.984375	570.752665	0	255.0300	389.3800	599.8700	3.989829e+04
last_fico_range_high	269360.0	568.572824	54.263500	0	524.0000	559.0000	604.0000	8.500000e+02
dti	269289.0	20.154296	11.825032	0	13.5400	19.7500	26.2900	9.990000e+02
total_rec_int	269360.0	2685.433899	2725.692738	0	852.9850	1795.6100	3568.1150	2.682238e+04

四、描述型分析

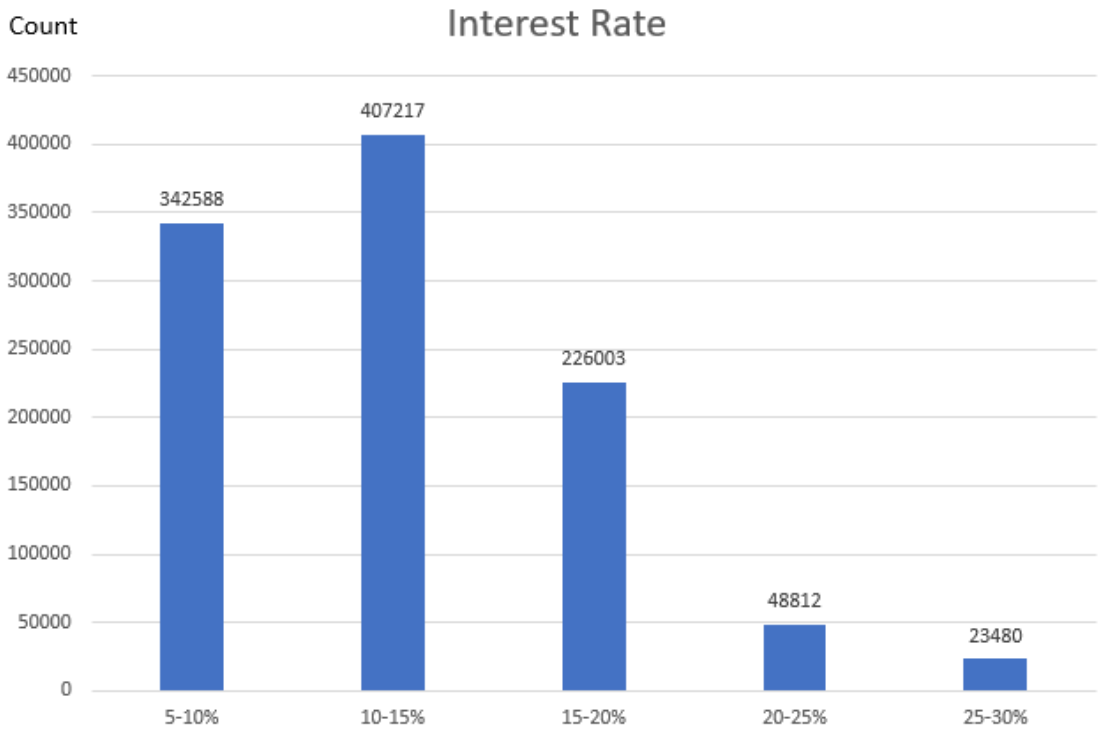
由下圖可知，一個案件的貸款金額多落在5001-10000元，而後隨金額增加，數量呈現遞減趨勢。



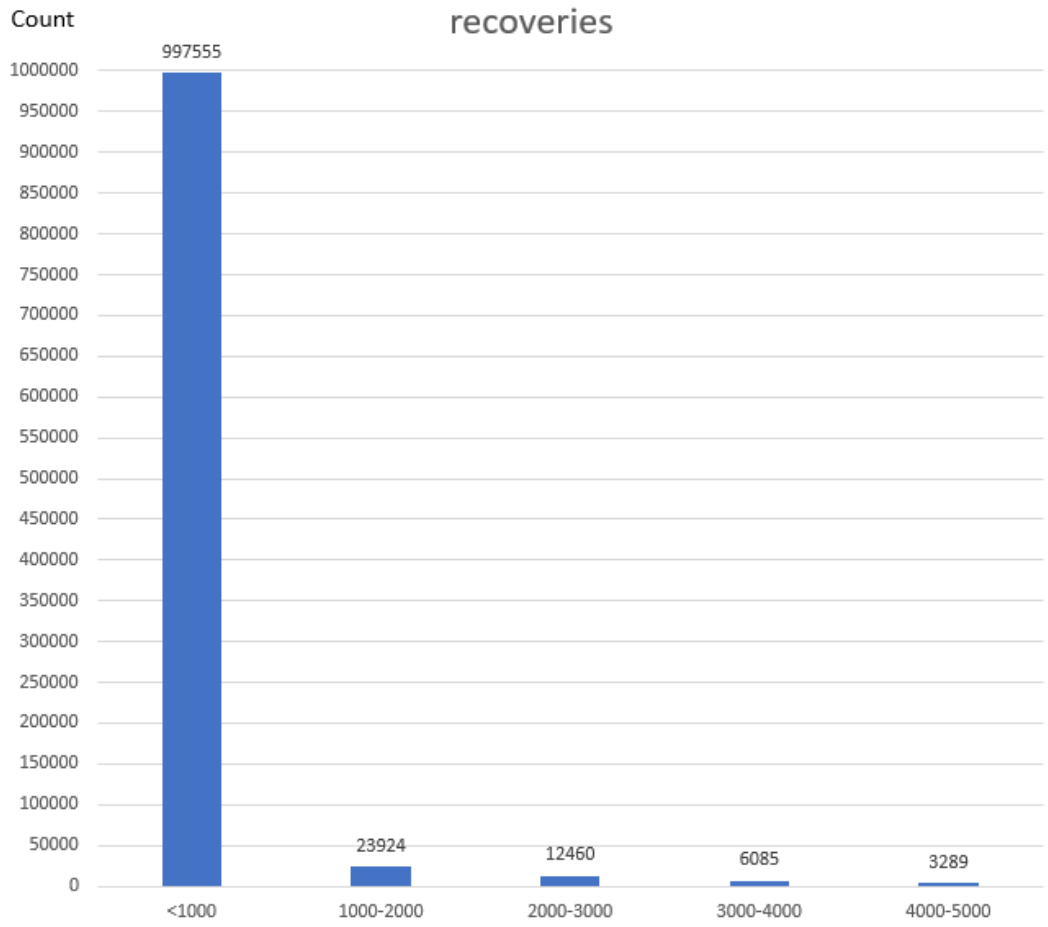
貸還償還期數多落在較短的36個月，這對於貸款公司也是比較有利的。



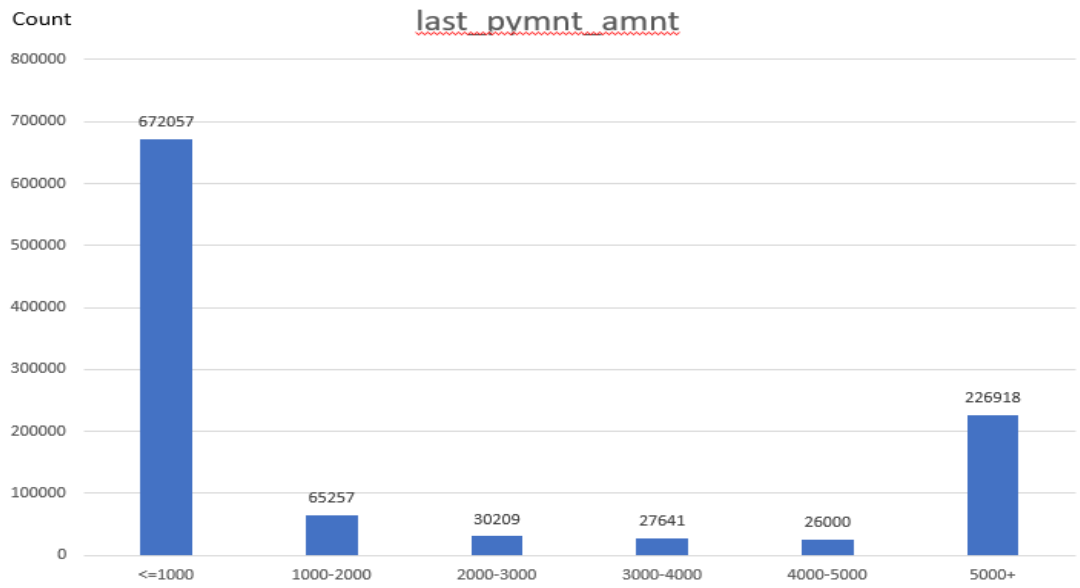
貸款利率多落在5-10%及10-15%的區間，最高落在30%左右。



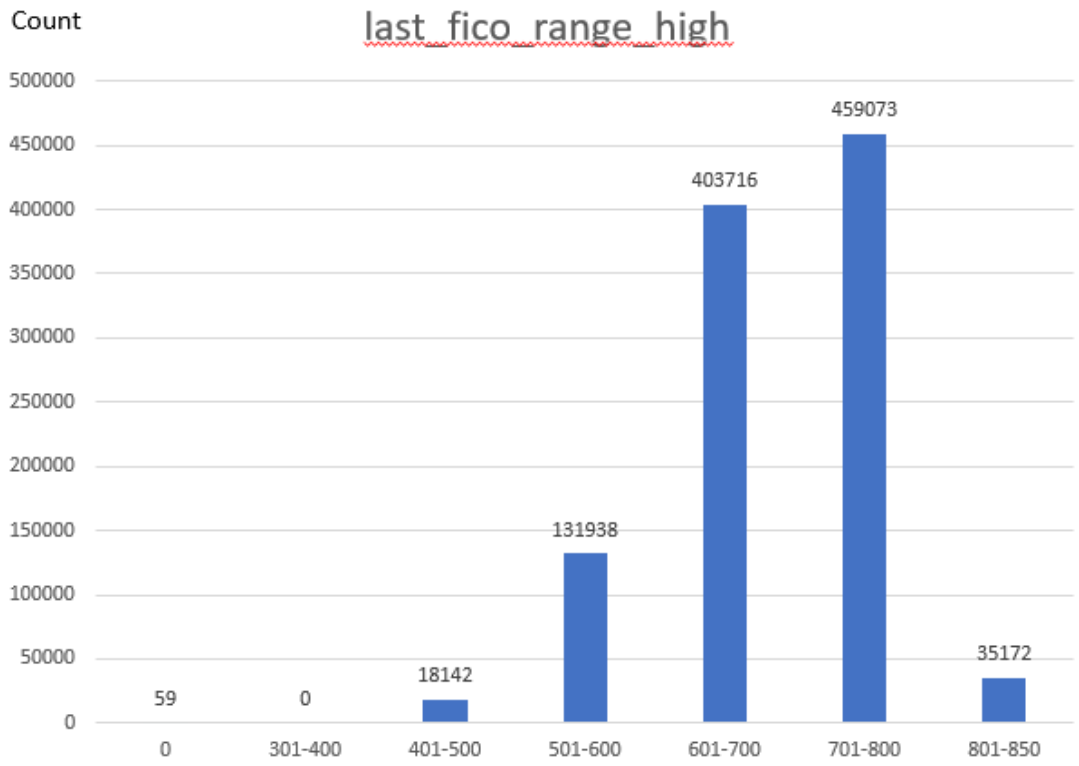
recoveries是貸款回收的金額，多落在<1000元，占了9成以上。



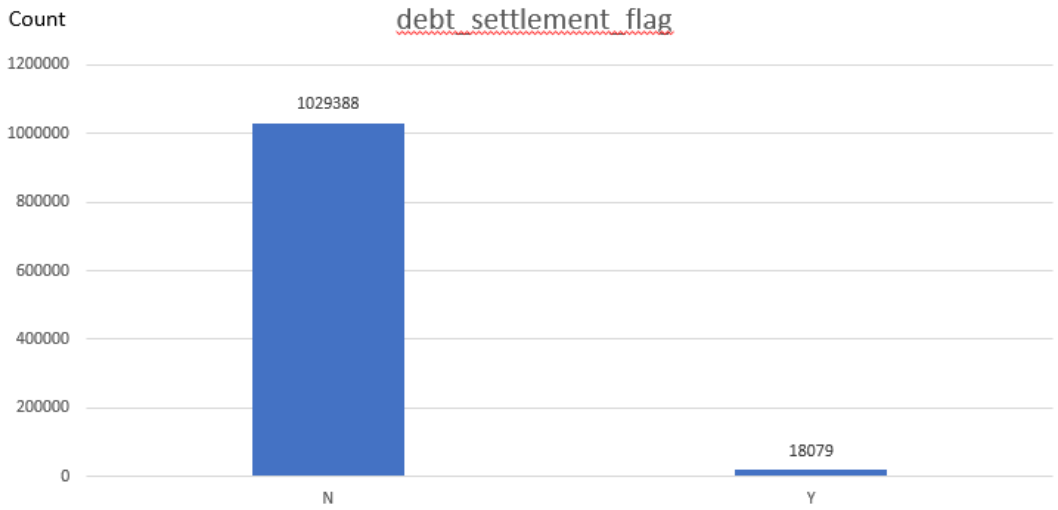
last payment amount是最近一次支付的金額，多落在小於1000元與大於5000元。



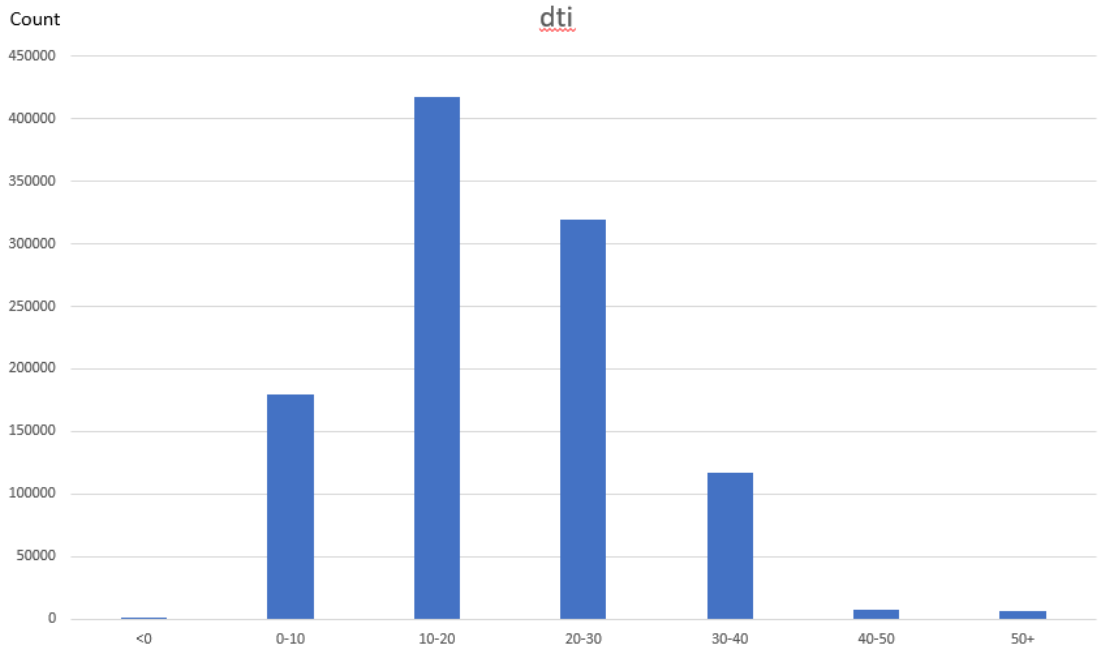
下圖是申請貸款人最近一次的Fico分數上緣，分數越高代表信用越好。而樣本的Fico分數多落在701-800之間。



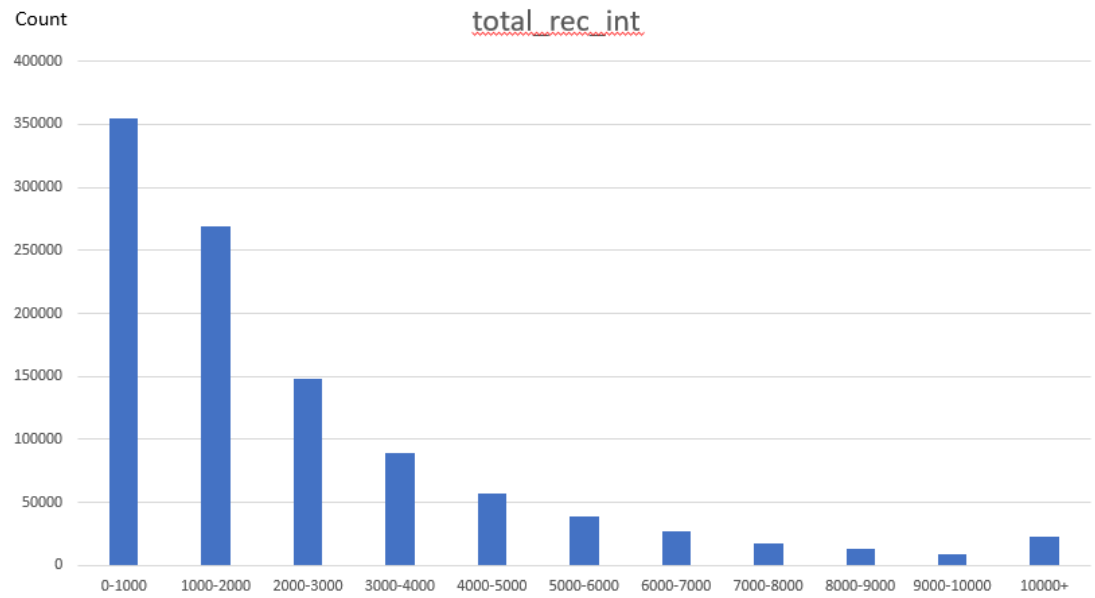
下圖是申請貸款人有無跟貸款清算公司合作，若有合作的話，對債權人是不利的，其違約風險也較高。



下圖是dfi分布圖，dti是一種比率，是指借款人每月就債務總額所繳付的債務總額除以借款人 自行申報的每月收入而計算的比率。dti越高，對債權人越有利。



total_rec_int是債權人至今收到的利息金額，大多數樣本皆落在0-1000元，樣本數分布隨金額提高而遞減。



第肆章 實證結果

一、模型的設計

在訓練模型之前，我們創建欄位'charged_off'儲存loan_status欄位中的Fully Paid及Charged Off。[charged_off = 1 (charged-off)；charged_off = 0 (fully paid)]，接著刪除掉欄位loan_status，樣本數及特徵數皆維持不變。

```
# 0 indicates fully paid and 1 indicates charge-off
loans['charged_off'] = (loans['loan_status'] == 'Charged Off').apply(np.uint8)
loans.drop('loan_status', axis=1, inplace=True)
```

接續使用交叉驗證切割訓練及測試資料，k設定為10，random state設定為42，shuffle=True。將charged_off以外的features作為自變數預測應變數charged_off的值。

```
X, y = loans.drop('charged_off', axis=1), loans['charged_off']
kf = KFold(n_splits=10, random_state=42, shuffle=True)

for train_index, test_index in kf.split(X):
  X_train, X_test = X.iloc[train_index], X.iloc[test_index],
  y_train, y_test = y.iloc[train_index], y.iloc[test_index],
```

1.羅吉斯回歸(Logistic Regression)

```
from sklearn.linear_model  import LogisticRegression
```

model=LogisticRegression(penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None)

其中較為重要的參數有:

1.penalty:指定懲罰的形式，有不加懲罰項、L1、L2及兩種懲罰項皆添加

2.max_iter:演算法收斂的最大迭代數

2.隨機森林(Random Forest)

我們的參數調整專注在n_estimators上，此參數代表森林中取樣樹的數目，在預設100棵附近測試了80、85、90、95、100，發現取樣100棵樹的時候訓練結果和測試結果的準確度最接近，因此我們選擇100棵做為最佳參數。為避免節點過度擴張導致執行時間過長，我們分別測試max_depth=6和7，最終7對於模型適配表現較佳。

```
In [68]:
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier(max_depth = 7)
model.fit(X_train, y_train)

Out[68]:
RandomForestClassifier(max_depth=7)
```

3.GBDT(Gradient Boosting Decision Tree)

GradientBoostingClassifier有非常多參數，主要設定其中三個：

gradient_booster = GradientBoostingClassifier(learning_rate=0.1, n_estimators=200, max_depth=7,…)

這樣設定的原因是經過幾次嘗試後，得出預測結果為最佳。測試期間有使用n_estimators = 100、150、200，以及max_depth=3、6、7，雖然結果相距不遠，不過n_estimators =200與max_depth=7則是相對較佳的設定方式。

```
gradient_booster = GradientBoostingClassifier(max_depth=7, n_estimators=200,learning_rate=0.1)
gradient_booster.get_params()
```

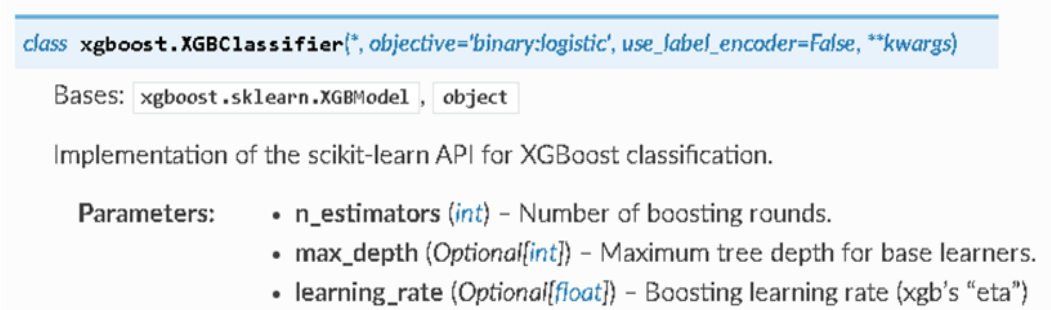
4.XGBoost (eXtreme Gradient Boosting)

由於XGBoost較為特殊，大部分的參數都已經最佳化，參數調整不易，因此本研究採用預設參數來訓練模型。

```
from xgboost.sklearn import XGBClassifier

xgb_clf = XGBClassifier(use_label_encoder=False)
xgb_clf.fit(X_train,y_train)

y_train_pred = xgb_clf.predict(X_train)
y_test_pred = xgb_clf.predict(X_test)

print_score(y_train,y_train_pred,train=True)
print_score(y_test,y_test_pred,train=False)
```

在XGBClassifier中，重要的參數有：

1. n_estimators：樹的數量
2. max_depth：每顆樹的最大深度
3. learning_rate：學習率，範圍通常在0.01-0.2之間



class **xgboost.XGBClassifier**(**, objective='binary:logistic', use_label_encoder=False, **kwargs*)

Bases: `xgboost.sklearn.XGBModel`, `object`

Implementation of the scikit-learn API for XGBoost classification.

Parameters:
- **n_estimators** (*int*) – Number of boosting rounds.
- **max_depth** (*Optional[int]*) – Maximum tree depth for base learners.
- **learning_rate** (*Optional[float]*) – Boosting learning rate (xgb's "eta")

XGBClassifier 參數說明

二、訓練資料與測試資料之結果

1.羅吉斯回歸(Logistic Regression)

```
Accuracy Score: 96.93%

_____________________________________________
CLASSIFICATION REPORT:
                     0              1  accuracy     macro avg  weighted avg
precision     0.976138       0.940355  0.969322  9.582465e-01  9.690373e-01
recall        0.985826       0.902654  0.969322  9.442401e-01  9.693220e-01
f1-score      0.980958       0.921119  0.969322  9.510385e-01  9.690840e-01
support    967428.000000  239496.000000  0.969322  1.206924e+06  1.206924e+06
_____________________________________________
Confusion Matrix:
 [[953716  13712]
 [ 23314 216182]]

Test Result:
================================================
Accuracy Score: 88.36%

_____________________________________________
CLASSIFICATION REPORT:
                     0              1  accuracy      macro avg   weighted avg
precision     0.951662       0.667694  0.883618       0.809678       0.895792
recall        0.900871       0.813182  0.883618       0.857027       0.883618
f1-score      0.925570       0.733291  0.883618       0.829431       0.887740
support    107718.000000  26384.000000  0.883618  134102.000000  134102.000000
_____________________________________________
Confusion Matrix:
 [[97040 10678]
 [ 4929 21455]]
```

2.隨機森林(Random Forest)

```
Accuracy Score: 95.94%
_____________________________________________
CLASSIFICATION REPORT:
                      0              1  accuracy     macro avg  weighted avg
precision      0.953174       0.991958  0.959395  9.725658e-01  9.608699e-01
recall         0.998391       0.801876  0.959395  9.001331e-01  9.593951e-01
f1-score       0.975258       0.886846  0.959395  9.310520e-01  9.577142e-01
support    967428.000000  239496.000000  0.959395  1.206924e+06  1.206924e+06
_____________________________________________
Confusion Matrix:
 [[965871   1557]
 [ 47450 192046]]

Test Result:
================================================
Accuracy Score: 95.95%
_____________________________________________
CLASSIFICATION REPORT:
                      0              1  accuracy      macro avg   weighted avg
precision      0.953533       0.991048  0.959501       0.972290       0.960913
recall         0.998227       0.801395  0.959501       0.899811       0.959501
f1-score       0.975368       0.886188  0.959501       0.930778       0.957822
support    107718.000000  26384.000000  0.959501  134102.000000  134102.000000
_____________________________________________
Confusion Matrix:
 [[107527    191]
 [  5240  21144]]
```

3.GBDT(Gradient Boosting Decision Tree)

```
Accuracy Score: 99.09%
_____________________________________________
CLASSIFICATION REPORT:
                      0              1  accuracy     macro avg  weighted avg
precision      0.992312       0.985177   0.99092  9.887448e-01  9.908964e-01
recall         0.996391       0.968818   0.99092  9.826047e-01  9.909199e-01
f1-score       0.994348       0.976929   0.99092  9.856384e-01  9.908912e-01
support    967428.000000  239496.000000   0.99092  1.206924e+06  1.206924e+06
_____________________________________________
Confusion Matrix:
 [[963937   3491]
 [  7468 232028]]

Test Result:
================================================
Accuracy Score: 98.94%
_____________________________________________
CLASSIFICATION REPORT:
                      0              1  accuracy      macro avg   weighted avg
precision      0.991528       0.980330  0.989359       0.985929       0.989325
recall         0.995256       0.965282  0.989359       0.980269       0.989359
f1-score       0.993389       0.972748  0.989359       0.983068       0.989328
support    107718.000000  26384.000000  0.989359  134102.000000  134102.000000
_____________________________________________
Confusion Matrix:
 [[107207    511]
 [   916  25468]]
```

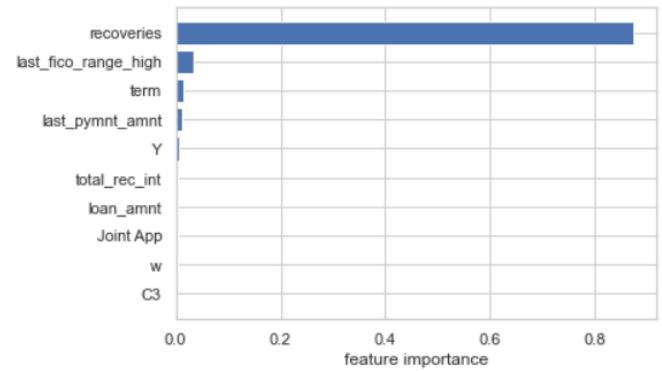
4. XGBoost (eXtreme Gradient Boosting)

```
Accuracy Score: 99.12%
_____________________________________________
CLASSIFICATION REPORT:
                     0             1  accuracy     macro avg  weighted avg
precision     0.992588      0.985495  0.991202  9.890414e-01  9.911802e-01
recall        0.996466      0.969941  0.991202  9.832035e-01  9.912024e-01
f1-score      0.994523      0.977656  0.991202  9.860896e-01  9.911760e-01
support    967428.000000 239496.000000  0.991202  1.206924e+06  1.206924e+06
_____________________________________________
Confusion Matrix:
 [[964009   3419]
 [  7199 232297]]

Test Result:
================================================
Accuracy Score: 98.99%
_____________________________________________
CLASSIFICATION REPORT:
                     0             1  accuracy      macro avg   weighted avg
precision     0.991888      0.981454  0.989866       0.986671       0.989835
recall        0.995525      0.966760  0.989866       0.981143       0.989866
f1-score      0.993703      0.974052  0.989866       0.983877       0.989837
support    107718.000000  26384.000000  0.989866  134102.000000  134102.000000
_____________________________________________
Confusion Matrix:
 [[107236    482]
 [   877  25507]]
```

三、特徵重要性呈現

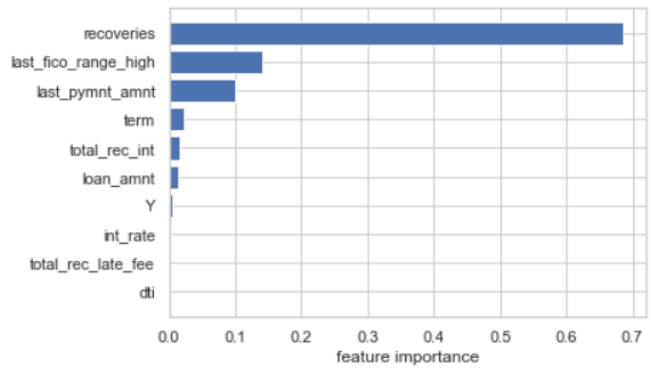
Feature importance是在預測模型進行預測後衡量每個特徵的貢獻度，屬於事後可解釋性，並通用於多種模型(model-agnostic)。衡量特徵重要度的方法有不少，例如：Gini importance、Permutation feature importance、SHAP等。Gini importance是藉由計算子節點的Gini係數，再進行加權而得出特徵重要度；Permutation feature importance則是透過打亂特徵的排序，藉以得知特徵重要程度；SHAP是一種XAI方法，也有人使用在衡量特徵重要度上。以下是我們對於每種建模方法的Feature importance結果呈現：

XGBoost



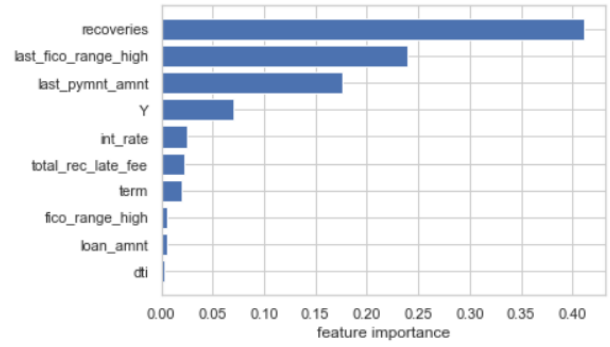
	feature	feature_importance
35	C3	0.002257
78	w	0.003503
79	Joint App	0.003538
0	loan_amnt	0.003562
13	total_rec_int	0.003803
81	Y	0.005295
16	last_pymnt_amnt	0.013358
1	term	0.015433
17	last_fico_range_high	0.034265
15	recoveries	0.874213

GBDT



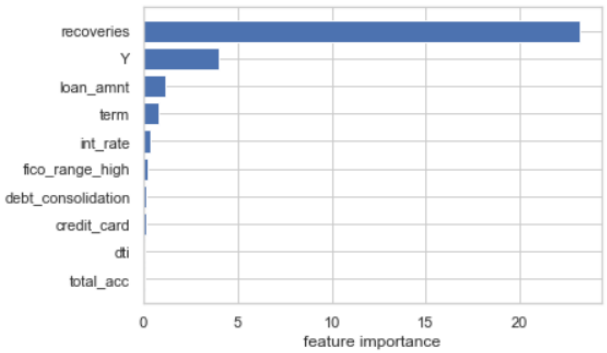
	feature	feature_importance
4	dti	0.001727
14	total_rec_late_fee	0.001983
2	int_rate	0.003958
81	Y	0.005559
0	loan_amnt	0.013192
13	total_rec_int	0.016514
1	term	0.021978
16	last_pymnt_amnt	0.099207
17	last_fico_range_high	0.140354
15	recoveries	0.686680

RandomForest



	feature	feature_importance
4	dti	0.003501
0	loan_amnt	0.003964
6	fico_range_high	0.004028
14	total_rec_late_fee	0.016689
2	int_rate	0.020156
1	term	0.021323
81	Y	0.048771
16	last_pymnt_amnt	0.168398
17	last_fico_range_high	0.287150
15	recoveries	0.409740

LogisticRegression



	feature	feature_importance
12	total_acc	0.113762
4	dti	0.115083
65	credit_card	0.135306
66	debt_consolidation	0.164897
6	fico_range_high	0.207438
2	int_rate	0.399551
1	term	0.842581
0	loan_amnt	1.210943
81	Y	3.991150
15	recoveries	23.158323

可以從上表中觀察到四個模型的特徵重要程度都不太一致，特別是羅吉斯回歸。而XGBoost、GBDT、RandomForest三個特徵重要度的前幾名較為相向，其中比較重要的特徵有recoveries、last_fica_range_high、last_pymnt_amnt，特別是recoveries重要程度特別高(藍色長條愈長，表示重要程度愈高)。

第伍章 結論與建議

一、總結

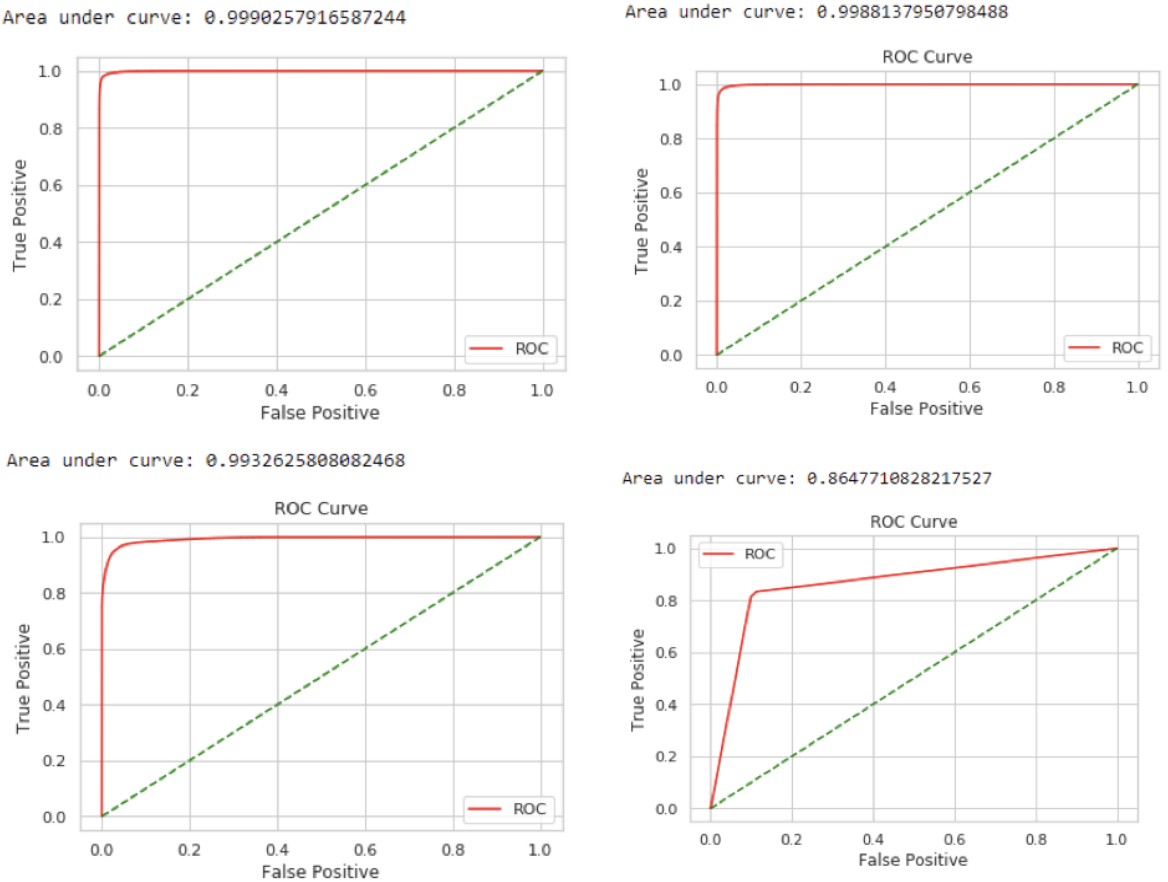
1. Accuracy

經過資料前處理後，使用XGB、GB和隨機森林的準確率在測試資料及訓練資料上皆能達到九成以上，另外我們發現羅吉斯回歸有發生過度配適的情形，屬於四者之中較差的模型。從分類報告(CLASSIFICATION REPORT)中，可以看出訓練資料0的Support為967428，1的Support為239496，顯示資料集有稍微不平衡的現象，但是兩者在XGB和GB的精確率和召回率皆接近99%，表示模型的預測能力極高，而隨機森林在數次調整參數後，精確率也都能達到九成五以上，惟1的召回率略遜於前兩者，落在0.80左右。這也使得隨機森林1的f1-score偏低，但預測能力不會影響太多，準確度仍有95%。

2. ROC Curve

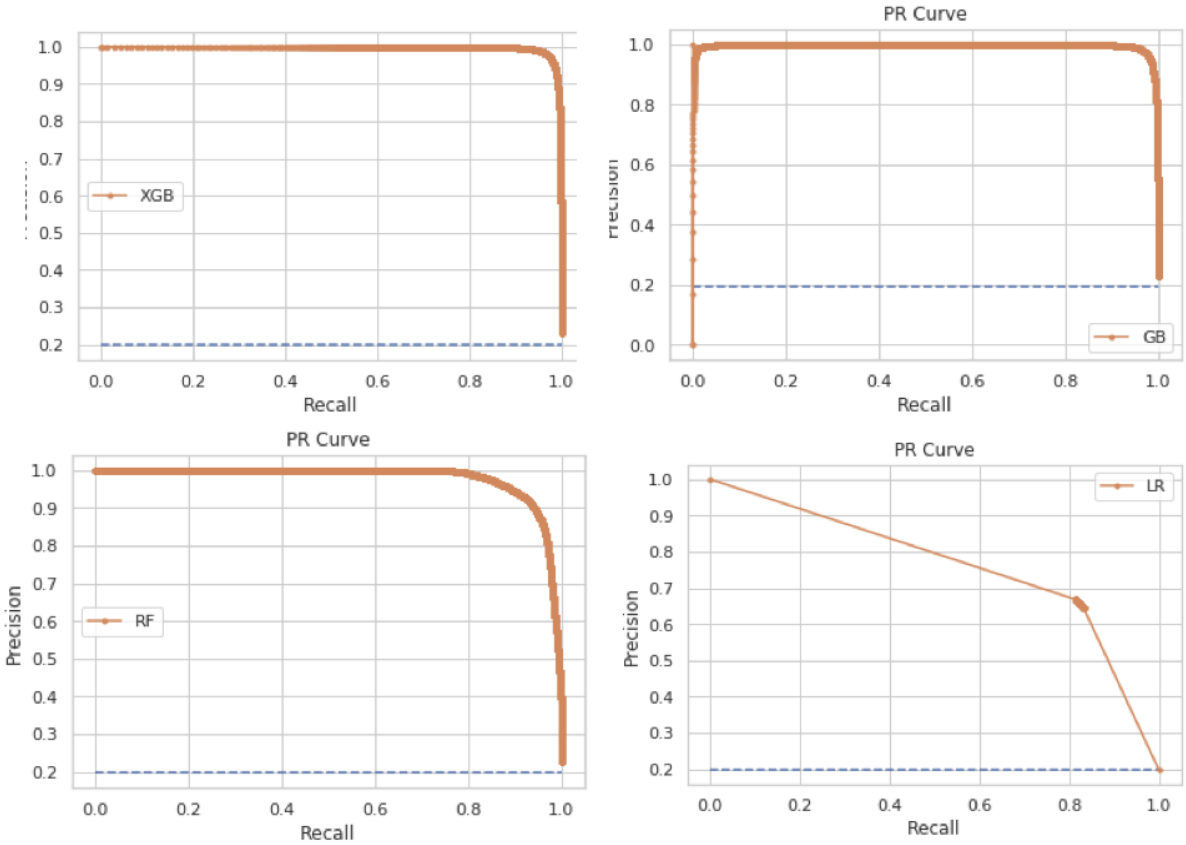
ROC曲線同時考慮陽性和陰性資料，能幫助我們評估分類模型整體的效能，我們使用曲線底下面積(area under curve)作為最直觀的判斷基準，面積愈大代表模型的表現愈好。

(下圖順時針方向分別為XGB、GB、Logistic Regression、Random Forest。)



3. PR Curve

PR曲線類似於ROC曲線，但只考慮陽性資料，因此較適合用在不平衡的資料集，一般來說精確度和召回度皆達到1的時候，模型將被視為擁有完美的預測能力。可以觀察到在實驗中，XGB、GB、隨機森林三者的右上角皆呈現接近直角的形狀，佐證其效能。（下圖順時針方向分別為XGB、GB、Logistic Regression、Random Forest。）



二、結論

對銀行而言評估貸款人會不會違約是非常重要的一件事情，如果貸款人有很大機率違約，那再將前借出，就等於把錢拿去打水瓢。那我們藉由Kaggle網站上P2P借貸的真實資料進行四種模型的建模與預測，預測結果發現不管是哪一種模型，預測精準度都非常高，但羅吉斯回歸則偏低一些，且有發生過度配適的情況，不過理解羅吉斯回歸的運作原理後，預測結果有過度配適也不讓人意外。綜合以上分析與預測結果，我們人為利用XGB、Gradient Boosting、Random Forest預測出債務人是否會違約，是有相當程度地可靠度可以提供Lending club P2P借貸評估風險的依據。

第陸章 額外分析

在使用accepted資料集(通過核貸)做完貸款人違約預測分類問題的分析後，我們結合了rejected資料集(拒絕貸款)進行另外一個分類問題的分析，我們想知道:若有一個人來申請貸款，此人的申請資料是否可讓他通過核貸?

accepted資料集與rejected資料集共通的特徵非常少，特徵有三，如下所示:

1.Amount Requested：貸款人申請的貸款金額

2.Risk Score：介於300-800之間，分數越高，信用越好

3.Employment Length：貸款人已工作年數

將兩個資料集的特徵數值合併後，我們添加一個新的應變數欄位:在accepted資料集中的樣本，我們給1，rejected資料集中的樣本則給0(添加是否通過貸款的label)。

總樣本數：194,790

label為1樣本數:96,670

label為0樣本數:98,120

做完資料前處理後，我們使用XGB、GBDT、隨機森林和羅吉斯回歸建立預測模型，模型最終表現如下：

模型	Accuracy	Precision	Recall	F1-Score
XGB	0.9609836	0.9968742	0.9298648	0.9622043
GBDT	0.9131029	0.9836499	0.8615481	0.9185591
隨機森林	0.9303865	0.9673985	0.9003229	0.9326563
羅吉斯回歸	0.7598268	0.8169546	0.7320406	0.7721702

XGB：

In [11]:
```
model = XGBClassifier()
model.fit(X_train,y_train)

X_test = scaler.transform(X_test)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)

print(df['Y'].value_counts())
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
```

```
[18:02:46] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.3.0/src/learner.cc:1061: Starting in XGBoost 1.3.
0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly se
t eval_metric if you'd like to restore the old behavior.
0    98120
1    96670
Name: Y, dtype: int64
number of correct sample: 56157
accuracy: 0.960983623389291
```

In [12]:
```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)
con_matrix = confusion_matrix(y_test, y_pred)


print("Mean squared error: {}".format(mse))
print('r2 score: {}'.format(r2))
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
print('confusion matrix: {}'.format(con_matrix))
```

```
Mean squared error: 0.03901637661070897
r2 score: 0.8439324588511251
number of correct sample: 56157
accuracy: 0.960983623389291
confusion matrix: [[27135  2189]
 [   91 29022]]
```

GBDT：

In [14]:
```
model = GradientBoostingClassifier()
model.fit(X_train,y_train)

X_test = scaler.transform(X_test)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)

print(df['Y'].value_counts())
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
```

```
0    98120
1    96670
Name: Y, dtype: int64
number of correct sample: 53359
accuracy: 0.9131029998117631
```

In [15]:
```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)
con_matrix = confusion_matrix(y_test, y_pred)


print("Mean squared error: {}".format(mse))
print('r2 score: {}'.format(r2))
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
print('confusion matrix: {}'.format(con_matrix))
```

```
Mean squared error: 0.0868970001882369
r2 score: 0.652407467564041
number of correct sample: 53359
accuracy: 0.9131029998117631
confusion matrix: [[24722  4602]
 [  476 28637]]
```

隨機森林：

In [5]:
```
model = RandomForestClassifier()
model.fit(X_train,y_train)

X_test = scaler.transform(X_test)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)

print(df['Y'].value_counts())
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
```

```
0    98120
1    96670
Name: Y, dtype: int64
number of correct sample: 54369
accuracy: 0.930386570152472
```

In [6]:
```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)
con_matrix = confusion_matrix(y_test, y_pred)


print("Mean squared error: {}".format(mse))
print('r2 score: {}'.format(r2))
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
print('confusion matrix: {}'.format(con_matrix))
```

```
Mean squared error: 0.06961342984752811
r2 score: 0.7215426502659548
number of correct sample: 54369
accuracy: 0.930386570152472
confusion matrix: [[26206  3118]
 [  950 28163]]
```

羅吉斯回歸：

In [8]:
```
model=LogisticRegression()
model.fit(X_train,y_train)

X_test = scaler.transform(X_test)
y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)

print(df['Y'].value_counts())
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
```

```
0    98120
1    96670
Name: Y, dtype: int64
number of correct sample: 44402
accuracy: 0.7598268220476753
```

In [9]:
```
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score, confusion_matrix
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)
num_correct_samples = accuracy_score(y_test, y_pred, normalize=False)
con_matrix = confusion_matrix(y_test, y_pred)


print("Mean squared error: {}".format(mse))
print('r2 score: {}'.format(r2))
print('number of correct sample: {}'.format(num_correct_samples))
print('accuracy: {}'.format(accuracy))
print('confusion matrix: {}'.format(con_matrix))
```

```
Mean squared error: 0.24017317795232473
r2 score: 0.03929476314716707
number of correct sample: 44402
accuracy: 0.7598268220476753
confusion matrix: [[20618  8706]
 [ 5329 23784]]
```

額外分析中，各項模型的表現，不論是Accuracy、Precision、recall或是F1-Score，都是XGB的表現最好。

第柒章 參考資料

- All Lending Club loan data
  https://www.kaggle.com/wordsforthewise/lending-club/code?datasetId=902&sortBy=commentCount
- Yuanchao Wang, Zhichen Pan, Jianhua Zheng, Lei Qian and Mingtao Li. A hybrid ensemble method for pulsar candidate classification. arXiv:1909.05303v2, 2019
- EDA with Python
  https://www.kaggle.com/wordsforthewise/eda-with-python
- Python API Reference - xgboost
  https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn
- 隨機森林算法參數解釋及調整優化
- 隨機森林(RANDOM FOREST)的底層概念、操作細節，與推薦相關資源
- sklearn.ensemble.RandomForestClassifier
- 隨機森林 - 維基百科
- sklearn.ensemble.sklearn.linear_model.LogisticRegression
- 羅吉斯回歸說明