# Predicting Client's Repayment Ability

This python notebook includes our data merging and modeling process, for data cleaning of the mair

# BA 888 - Capstone Project

Team 7

# Import libraries

↳ *1 cell hidden*

# Data Merging

In this section, we extract important features from the 5 small tables that we think might be relevant t application and merge these features to the main application table.

## Main tables

Static data for all applications. One row represents one loan in our data sample.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=

Enter your authorization code:
··········
Mounted at /content/drive
```

```
df = pd.read_csv("drive/My Drive/888/dummy.csv")
```

```
df.head()
```

| | Unnamed: 0 | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE.Revolving.loans | CODE_GENDER. |
|---|---|---|---|---|---|
| **0** | 2 | 100031.0 | 1 | 0 | |
| **1** | 3 | 100047.0 | 1 | 0 | |
| **2** | 4 | 100049.0 | 1 | 0 | |
| **3** | 5 | 100096.0 | 1 | 0 | |
| **4** | 6 | 100112.0 | 1 | 0 | |

5 rows × 143 columns

## POS_CASH_balance Data

Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had wi
This table has one row for each month of history of every prev[link text]ious credit in Home Credit (con
loans in main data

```
#Group by SK_ID_CURR and aggregate
# Input and display the cleaned POS_CASH_balance dataset
pos_cash= pd.read_csv('drive/My Drive/888/df_POS_CASH.csv',sep="\t")
```

```
pos_cash.head()
```

| | SK_ID_CURR | defalut_rate_POS |
|---|---|---|
| **0** | 100001.0 | 0.111111 |
| **1** | 100002.0 | 0.000000 |
| **2** | 100003.0 | 0.000000 |
| **3** | 100004.0 | 0.000000 |
| **4** | 100005.0 | 0.000000 |

- default_rate_POS = $\frac{number\ of\ past\ due\ (with\ tolerance)}{total\ number\ of\ months}$

## Installment Payment Data

Repayment history for the previously disbursed credits in Home Credit related to the loans in our sam
One row is equivalent to one payment of one installment from previous loan application.

```
# SELECT
```

```
#   COUNT(DISTINCT SK_ID_PREV) AS NUM_PREV,
#   COUNT(*) AS NUM_PAY,
#   SK_ID_CURR,
#   AVG(DAYS_PAST_DUE_INS) AS AVG_PASS_DUE_INS,
#   SUM(CASE
#       WHEN DAYS_PAST_DUE_INS <= 0 THEN 1
#     ELSE
#     0
#   END
#     ) AS NUM_PRE_PAY,
#   SUM(CASE
#       WHEN DAYS_PAST_DUE_INS > 0 THEN 1
#     ELSE
#     0
#   END
#     ) AS NUM_LATE_PAY,
#   MAX(AMT_INSTALMENT)AS MAX_AMT_INS,
#   SUM(UNPAY_AMOUNT) AS TOTAL_UNPAY,
# FROM (
#   SELECT
#     SK_ID_PREV,
#     SK_ID_CURR,
#     (DAYS_ENTRY_PAYMENT - DAYS_INSTALMENT) AS DAYS_PAST_DUE_INS,
#     (AMT_INSTALMENT- AMT_PAYMENT)AS UNPAY_AMOUNT,
#     AMT_INSTALMENT,
#     AMT_PAYMENT
#   FROM
#     `ba888-team7.Original_dataset.installments_payments` )
# GROUP BY
#   SK_ID_CURR

#Group by SK_ID_CURR and agregate
# Input and display the cleaned installment_payment dataset
install_pay = pd.read_csv('drive/My Drive/888/installment_payment.csv')
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: Dty
  interactivity=interactivity, compiler=compiler, result=result)
```

```
install_pay.head()
```

```
install_pay["ins_tunpay_ratio"] = pd.to_numeric(install_pay["ins_tunpay_ratio"],error
```

- NUM_PREV : Count of the previous applications
- SK_ID_CURR : ID of loan, key to join back to main dataset
- AVG_PASS_DUE_INS : AVG(days_entry_payment - days_instalment) (Negative means pay before
- ins_nprepay_ratio : $\frac{number\ of\ times\ the\ applicants\ pay\ before\ due}{Total\ number\ of\ payments}$
- ins_tunpay_ratio : $\frac{Sum(installment\ amount)}{Sum(suppose\ to\ paid\ installment\ amount - actual\ paid\ amount)}$

## Credit_Card_Balance Data

Monthly balance snapshots of previous credit cards that the applicant has with Home Credit. This tal history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in o

```
#Group by SK_ID_CURR and agregate
# Input and display the cleaned credit_card_balance dataset
credit_balance = pd.read_csv('drive/My Drive/888/cc_card_data.csv',sep="\t")
```

```
credit_balance.head()
```

| | SK_ID_CURR | day_past_due | day_past_due_t | credit_limit | total_balance | total_p |
|---|---|---|---|---|---|---|
| **0** | 100006 | 0 | 0 | 270000.000000 | 0.000000 | 0. |
| **1** | 100011 | 0 | 0 | 164189.189189 | 54482.111149 | 4843. |
| **2** | 100013 | 1 | 1 | 131718.750000 | 18159.919219 | 7168. |
| **3** | 100021 | 0 | 0 | 675000.000000 | 0.000000 | 0. |
| **4** | 100023 | 0 | 0 | 135000.000000 | 0.000000 | 0. |

- SK_ID_CURR : ID of loan, key to join back to main dataset
- day_past_due_t : total count of past due month (with tolerance)
- credit_limit : mean(AMT_CREDIT_LIMIT_ACTUAL)
- total_balance : mean(AMT_BALANCE)
- total_payment : mean(AMT_PAYMENT_CURRENT)
- total_AMT_drawing: sum()

## Bureau Data

All client's previous credits provided by other financial institutions that were reported to Credit Bureau sample).

```
### SQL code for future edit
# SELECT *
# FROM (
#   SELECT
#     SK_ID_CURR,
#     COUNT(SK_ID_CURR) bureau_record_number,
#     ROUND(SUM(AMT_CREDIT_SUM_DEBT) / (SUM(AMT_CREDIT_SUM) + 0.0001), 4) bureau_debt
#     COALESCE(ROUND(SUM(AMT_CREDIT_MAX_OVERDUE), 2),
#       0) bureau_credit_amount_sum_overdue,
#     COALESCE( SUM(CREDIT_DAY_OVERDUE),
#       0) bureau_credit_days_sum_overdue,
#     ROUND(AVG(DAYS_CREDIT), 0) bureau_average_days_credit,
#   FROM `ba888-team7.Original_dataset.bureau`
#   WHERE SK_ID_CURR IN (
#     SELECT SK_ID_CURR
#     FROM `ba888-team7.Original_dataset.application_train`)
#     AND CREDIT_CURRENCY = 'currency 1'
#   GROUP BY SK_ID_CURR)
# LEFT JOIN (
#   SELECT
#     SK_ID_CURR,
#     MAX(DAYS_CREDIT) bureau_mostrecent_days_credit
#   FROM `ba888-team7.Original_dataset.bureau`
#   WHERE CREDIT_ACTIVE = 'Active'
#   GROUP BY SK_ID_CURR)
# USING (SK_ID_CURR)
# ORDER BY SK_ID_CURR


# Group by SK_ID_CURR and aggregate
# Input and display the cleaned bureau dataset
bureau= pd.read_csv('drive/My Drive/888/bureau.csv')


bureau.head()
```

| | SK_ID_CURR | bureau_record_number | bureau_debt_credit_ratio | bureau_credit_amou |
|---|---|---|---|---|
| **0** | 100002 | 8 | 0.2841 | |
| **1** | 100003 | 4 | 0.0000 | |
| **2** | 100004 | 2 | 0.0000 | |
| **3** | 100007 | 1 | 0.0000 | |
| **4** | 100008 | 3 | 0.5125 | |

- SK_ID_CURR : ID of loan, key to join back to main dataset
- bureau_record_number: The number of records provided by other financial institutions that were
- bureau_debt_credit_ratio : $\frac{sum\ of\ debt}{sum\ of\ all\ credit}$ Note: to avoid error, here we add a small amount to de
- bureau_credit_amount_sum_overdue: The sum of amount overdue reported to Credit Bureau
- bureau_credit_days_sum_overdue: The number of total days overdue reported to Credit Bureau
- bureau_average_days_credit: the average days for all loans record reported to Credit Bureau
- bureau_mostrecent_days_credit: From the active loans in bureau, the most recent DAYS CREDIT now

## previous_application Data

```
#Group by SK_ID_CURR and agregate
# Input and display the cleaned credit_card_balance dataset
previous_application = pd.read_csv('drive/My Drive/888/previous_application.csv')
```

```
previous_application.head(6)
```

| | SK_ID_CURR | reject_ratio | avg_approved_amt |
|---|---|---|---|
| **0** | 100001 | 0.000000 | 23787.0 |
| **1** | 100002 | 0.000000 | 179055.0 |
| **2** | 100003 | 0.000000 | 484191.0 |
| **3** | 100004 | 0.000000 | 20106.0 |
| **4** | 100005 | 0.000000 | 40153.5 |
| **5** | 100006 | 0.111111 | 343728.9 |

- SK_ID_CURR : ID of loan, key to join back to main dataset
- Avg_amt : the average loan amount of the applicant's previous loan application that have been a
- reject_ratio: $\frac{total\ number\ of\ rejection}{total\ number\ of\ loan\ applications}$

## Merge the data

In this step, we merge all the datasets together using the merge function in Python.

## left join

```
df2 = df.merge(bureau, how="left", on="SK_ID_CURR")
```

```
df2 = df2.merge(credit_balance, how="left", on="SK_ID_CURR")
```

```
df2 = df2.merge(install_pay, how="left", on="SK_ID_CURR")
```

```
df2 = df2.merge(pos_cash, how="left", on="SK_ID_CURR")
```

```
df2 = df2.merge(previous_application, how="left", on="SK_ID_CURR")
```

```
df2.head()
```

| | Unnamed: 0 | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE.Revolving.loans | CODE_GENDER. |
|---|---|---|---|---|---|
| **0** | 2 | 100031.0 | 1 | 0 | |
| **1** | 3 | 100047.0 | 1 | 0 | |
| **2** | 4 | 100049.0 | 1 | 0 | |
| **3** | 5 | 100096.0 | 1 | 0 | |
| **4** | 6 | 100112.0 | 1 | 0 | |

5 rows × 164 columns

## Deal with NA

Since the 5 small tables are mainly focusing on applicants who might have previous loan history with
possibility that there are some applicants who do not have their bureau data, credit card or previous a
Therefore, in this step, we are going to deal with the NA's.

```
df2.drop(['Unnamed: 0'], axis = 1, inplace=True)
```

```
df2.dtypes
```

```
SK_ID_CURR                                float64
TARGET                                      int64
NAME_CONTRACT_TYPE.Revolving.loans          int64
CODE_GENDER.M                               int64
FLAG_OWN_CAR.Y                              int64
                                           ...
ins_nprepay_ratio                         float64
ins_tunpay_ratio                          float64
defalut_rate_POS                          float64
reject_ratio                              float64
avg_approved_amt                          float64
Length: 163, dtype: object
```

```
NAcolumn = df2.isna().sum()
```

```
nac = pd.DataFrame(NAcolumn[NAcolumn>0])
```

```
nacn=list(nac.index.values.tolist())
```

We thought that replacing 'day_past_due' with any value or simply deleting them is not an optimal opt variable name 'no_credit' that represents the applicant's previous credit history. 1 means that they hav don't.

```
df2['no_credit'] = df2['day_past_due'].apply(lambda x : 1 if np.isnan(x) else 0)
```

## k-means Cluster

For other columns that contain NA's, we want to use KMeans to segment the customers into 2 groups pick up any characteristics that could differentiate the default and non-default customers. Then, we r median of the cluster that the customer belongs to.

```
x = df.iloc[:,3:] #exclude id and target columns
```

```
from sklearn.cluster import KMeans

# Number of clusters
kmeans = KMeans(n_clusters=2)
# Fitting the input data
kmeans = kmeans.fit(x)
# Getting the cluster labels
labels = kmeans.predict(x)
# Centroid values
centroids = kmeans.cluster_centers_
```

```
df2['cluster'] = labels
```

```
dftemp = df2.loc[df2['cluster'] == 0,:]
for i in nacn:
  med = dftemp[i].median()
  dftemp[i] = dftemp[i].apply(lambda x: med if np.isnan(x) else x)

dftemp1 = df2.loc[df2['cluster'] == 1,:]
for i in nacn:
  med = dftemp1[i].median()
  dftemp1[i] = dftemp1[i].apply(lambda x: med if np.isnan(x) else x)
```

```
df3 = dftemp.append([dftemp1])
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:4: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab]
  after removing the cwd from sys.path.
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:9: SettingWithCopyWa
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab]
  if __name__ == '__main__':
```

```
df3.cluster.describe()
```

```
count    244475.000000
mean          0.643129
std           0.479077
min           0.000000
25%           0.000000
50%           1.000000
75%           1.000000
max           1.000000
Name: cluster, dtype: float64
```

```
df3['TARGET'].describe()
```

```
count    244475.000000
mean          0.080781
std           0.272499
min           0.000000
25%           0.000000
50%           0.000000
75%           0.000000
max           1.000000
Name: TARGET, dtype: float64
```

```
df3
```

| | SK_ID_CURR | TARGET | NAME_CONTRACT_TYPE.Revolving.loans | CODE_GENDER.M | FL |
|---|---|---|---|---|---|
| **0** | 100031.0 | 1 | 0 | 0 | |
| **1** | 100047.0 | 1 | 0 | 1 | |
| **4** | 100112.0 | 1 | 0 | 1 | |
| **11** | 100273.0 | 1 | 0 | 0 | |
| **14** | 100295.0 | 1 | 0 | 1 | |
| **...** | ... | ... | ... | ... | |
| **244469** | 456248.0 | 0 | 0 | 0 | |
| **244470** | 456249.0 | 0 | 0 | 0 | |
| **244471** | 456251.0 | 0 | 0 | 1 | |
| **244472** | 456252.0 | 0 | 0 | 0 | |
| **244473** | 456253.0 | 0 | 0 | 0 | |

244475 rows × 165 columns

## Imbalance dataset issue - sampling

After cleaning the NA's, we want to deal with the imbalanced label issues in our dataset. Since our da significantly low number of default: 1's), we want to use SMOTE technique to help us with the imbalar

```
from collections import Counter
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
```

```
X = df3.iloc[:,2:]
y = df3['TARGET']
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_stat
```

```
counter = Counter(y_train)
print(counter)
```

```
Counter({0: 179747, 1: 15833})
```

```
over = SMOTE(sampling_strategy=0.2)
under = RandomUnderSampler(sampling_strategy=0.5)
```

```
under = RandomUnderSampler(sampling_strategy=0.5)
steps = [('o', over), ('u', under)]
pipeline = Pipeline(steps=steps)

X_train, y_train = pipeline.fit_resample(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWa
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:87: FutureWa
  warnings.warn(msg, category=FutureWarning)
```

```
print(Counter(y_train))
```

```
Counter({0: 71898, 1: 35949})
```

After trying random oversampling of the minority group (0), SMOTE oversampling of minority group, a group and undersampling of the majority group, we found that the combination of oversampling and 0.1 and 0.5 gave us the best predicted result.

# Modeling

## Logistic Regression

↳ 1 cell hidden

## Random Forest

↳ 7 cells hidden

## XGBoost

```
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

xg_boost = XGBClassifier(seed=27)

xg_boost.fit(X_train,y_train)

predict_test_xg_prob = xg_boost.predict_proba(X_test.values)

print("XGBoost Forest AUC Score (Test): %f" % roc_auc_score(y_test, predict_test_xg_p
```

```
XGBoost Forest AUC Score (Test): 0.748209
```

```
imp= xg_boost.feature_importances_

imp = {'n': range(len(random_forest.feature_importances_)), 'importance': imp}
imp = pd.DataFrame(imp)

imp.sort_values('importance', ascending=False).head(7)
```

| | n | importance |
|---|---|---|
| **22** | 22 | 0.115004 |
| **26** | 26 | 0.065702 |
| **27** | 27 | 0.061244 |
| **1** | 1 | 0.056865 |
| **58** | 58 | 0.055954 |
| **61** | 61 | 0.055866 |
| **159** | 159 | 0.051531 |

```
df3.iloc[:,[24,28,29,3,60,161]].columns
```

```
Index(['NAME_INCOME_TYPE.Working',
       'NAME_EDUCATION_TYPE.Secondary...secondary.special',
       'NAME_FAMILY_STATUS.Married', 'CODE_GENDER.M',
       'OCCUPATION_TYPE.Not.Provided', 'reject_ratio'],
      dtype='object')
```

## XGBoost Model Tuning

```
def modelfit(alg,useTrainCV=True, cv_folds=3, early_stopping_rounds=30):
  if useTrainCV:
    xgb_param = alg.get_xgb_params()
    xgtrain = xgb.DMatrix(X_train, label=y_train)
    cvresult = xgb.cv(xgb_param, xgtrain, num_boost_round=alg.get_params()['n_estimat
        metrics='auc', early_stopping_rounds=early_stopping_rounds)
    print("There are {} trees in our model. CV-mean: {:.4f}, CV-std: {:.4f}.".format(
        cvresult.shape[0], cvresult.iloc[cvresult.shape[0]-1, 0],
        cvresult.iloc[cvresult.shape[0]-1, 1]))
    alg.set_params(n_estimators=cvresult.shape[0])

  #Fit the algorithm on the data
  alg.fit(X_train, y_train,eval_metric='auc')

  #Predict training set:
  dtrain_predictions = alg.predict(X_test.values)
  dtrain_predprob = alg.predict_proba(X_test.values)
```

```
    #Print model report:
    print ("\nModel Report")
    print ("Accuracy : %.4g" % accuracy_score(y_test, dtrain_predictions))
    print ("F1 Score (Test): %f" % f1_score(y_test, dtrain_predictions))
    print ("AUC Score (Test): %f" % roc_auc_score(y_test, dtrain_predprob[:,1]))
```

## Depth and weight

↳ *3 cells hidden*

## Gamma

↳ *2 cells hidden*

## Subsample

↳ *4 cells hidden*

## Final XGBoost model

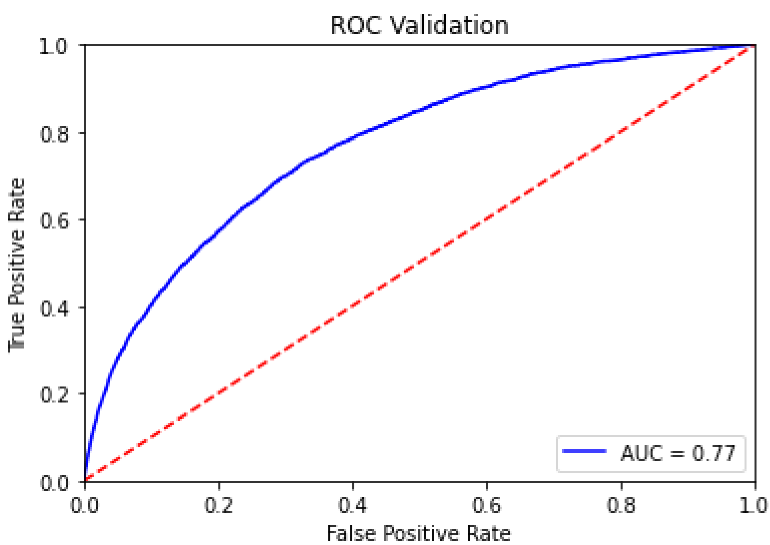
```
xgb4 = XGBClassifier(
 learning_rate =0.01,
 n_estimators=2000,
 max_depth=5,
 min_child_weight=3,
 gamma=0,
 subsample=0.8,
 colsample_bytree=0.8,
 reg_alpha=0,
 objective= 'binary:logistic',
 nthread=4,
 scale_pos_weight=1,
 seed=27)
modelfit(xgb4)
```

```
There are 2000 trees in our model. CV-mean: 0.9271, CV-std: 0.0007.

Model Report
Accuracy : 0.9118
F1 Score (Test): 0.233789
AUC Score (Test): 0.765433
```

## graph

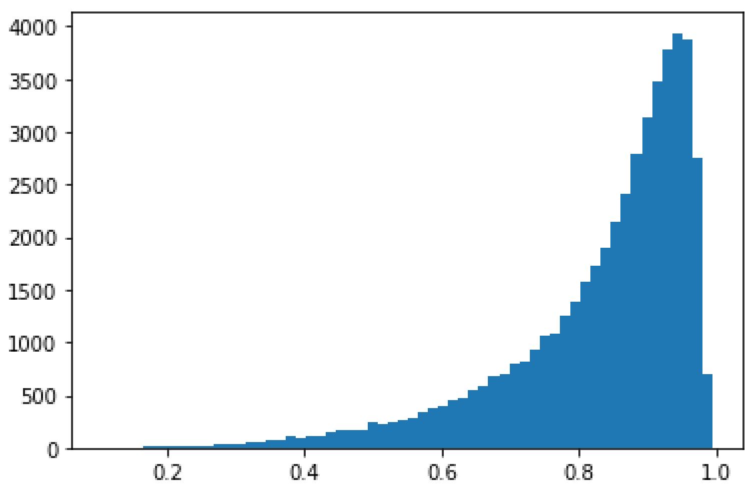
```
predict_test_xg_prob = xgb4.predict_proba(X_test.values)
```

```
fpr, tpr, _ = roc_curve(y_test, predict_test_xg_prob[:, 1])
roc_auc = auc(fpr, tpr)
plt.title('ROC Validation')
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])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
plt.hist(predict_test_xg_prob[:, 0], bins = 60)
```

```
(array([1.000e+00, 1.000e+00, 2.000e+00, 5.000e+00, 7.000e+00, 8.000e+00,
        6.000e+00, 1.600e+01, 1.300e+01, 1.800e+01, 2.100e+01, 3.300e+01,
        3.900e+01, 3.000e+01, 5.500e+01, 5.700e+01, 6.700e+01, 8.000e+01,
        1.020e+02, 9.700e+01, 1.130e+02, 1.170e+02, 1.430e+02, 1.660e+02,
        1.750e+02, 1.700e+02, 2.430e+02, 2.280e+02, 2.430e+02, 2.680e+02,
        2.880e+02, 3.290e+02, 3.770e+02, 3.980e+02, 4.500e+02, 4.800e+02,
        5.560e+02, 5.810e+02, 6.790e+02, 6.910e+02, 7.940e+02, 8.060e+02,
        9.230e+02, 1.053e+03, 1.084e+03, 1.255e+03, 1.392e+03, 1.567e+03,
        1.728e+03, 1.901e+03, 2.149e+03, 2.411e+03, 2.799e+03, 3.136e+03,
        3.479e+03, 3.784e+03, 3.938e+03, 3.885e+03, 2.763e+03, 6.950e+02]),
 array([0.1055302 , 0.12035343, 0.13517666, 0.14999989, 0.16482313,
        0.17964636, 0.19446959, 0.20929281, 0.22411604, 0.23893927,
        0.2537625 , 0.26858574, 0.28340897, 0.2982322 , 0.31305543,
        0.32787865, 0.34270188, 0.3575251 , 0.37234834, 0.38717157,
        0.4019948 , 0.41681802, 0.43164125, 0.4464645 , 0.46128774,
        0.47611097, 0.4909342 , 0.5057574 , 0.52058065, 0.53540385,
        0.5502271 , 0.56505036, 0.57987356, 0.5946968 , 0.60952   ,
        0.6243433 , 0.6391665 , 0.65398973, 0.66881293, 0.6836362 ,
        0.6984594 , 0.71328264, 0.72810584, 0.7429291 , 0.7577523 ,
        0.77257556, 0.7873988 , 0.802222  , 0.8170453 , 0.83186847,
        0.8466917 , 0.8615149 , 0.8763382 , 0.8911614 , 0.90598464,
        0.92080784, 0.9356311 , 0.9504543 , 0.96527755, 0.98010075,
        0.994924  ], dtype=float32),
 <a list of 60 Patch objects>)
```




```
###
models = {"xgb": xgb4,
          "rf": random_forest,
          "lr": lr_clf}

for name, model in models.items():
  model_probs = model.predict_proba(X_test.values)[:, 1:]
  model_auc_score = roc_auc_score(y_test, model_probs)
  fpr, tpr, _ = roc_curve(y_test, model_probs)
  precision, recall, _ = precision_recall_curve(y_test, model_probs)
  axes[0].plot(fpr, tpr, label=f"{name}, auc = {model_auc_score:.3f}")
  axes[1].plot(recall, precision, label=f"{name}")
```

## Cut off and f1 score

```
from sklearn.metrics import precision_recall_curve
presicion, recall, threshold = precision_recall_curve(y_test, predict_test_xg_prob[:,
```

```
d = {'precision': presicion[0:-1], 'recall': recall[0:-1], 'threshold' : threshold}
f1df = pd.DataFrame(data=d)
f1df
```

| | precision | recall | threshold |
|---|---|---|---|
| **0** | 0.079785 | 1.000000 | 0.011061 |
| **1** | 0.079767 | 0.999743 | 0.011098 |
| **2** | 0.079768 | 0.999743 | 0.011112 |
| **3** | 0.079770 | 0.999743 | 0.011127 |
| **4** | 0.079771 | 0.999743 | 0.011152 |
| **...** | ... | ... | ... |
| **48754** | 1.000000 | 0.001283 | 0.849189 |
| **48755** | 1.000000 | 0.001027 | 0.858475 |
| **48756** | 1.000000 | 0.000770 | 0.859527 |
| **48757** | 1.000000 | 0.000513 | 0.873891 |
| **48758** | 1.000000 | 0.000257 | 0.894470 |

48759 rows × 3 columns

```
f1df['f1score'] = 2*f1df['precision']*f1df['recall']/(f1df['recall']+f1df['precision'
```

```
f1df['f1score'].describe()
```

```
count    48759.000000
mean         0.227151
std          0.056279
min          0.000513
25%          0.179308
50%          0.222073
75%          0.277714
max          0.316635
Name: f1score, dtype: float64
```

```
f1df.sort_values('f1score',ascending=False)
```

| | precision | recall | threshold | f1score |
|---|---|---|---|---|
| **43817** | 0.283000 | 0.359343 | 0.358436 | 0.316635 |
| **43816** | 0.282943 | 0.359343 | 0.358392 | 0.316599 |
| **43815** | 0.282885 | 0.359343 | 0.358369 | 0.316563 |
| **43814** | 0.282828 | 0.359343 | 0.358318 | 0.316527 |
| **43820** | 0.282969 | 0.359086 | 0.358589 | 0.316516 |
| **...** | ... | ... | ... | ... |
| **48754** | 1.000000 | 0.001283 | 0.849189 | 0.002563 |
| **48755** | 1.000000 | 0.001027 | 0.858475 | 0.002051 |
| **48756** | 1.000000 | 0.000770 | 0.859527 | 0.001539 |
| **48757** | 1.000000 | 0.000513 | 0.873891 | 0.001026 |
| **48758** | 1.000000 | 0.000257 | 0.894470 | 0.000513 |

48759 rows × 4 columns

```
df_prob = pd.DataFrame(predict_test_xg_prob[:,1])
```

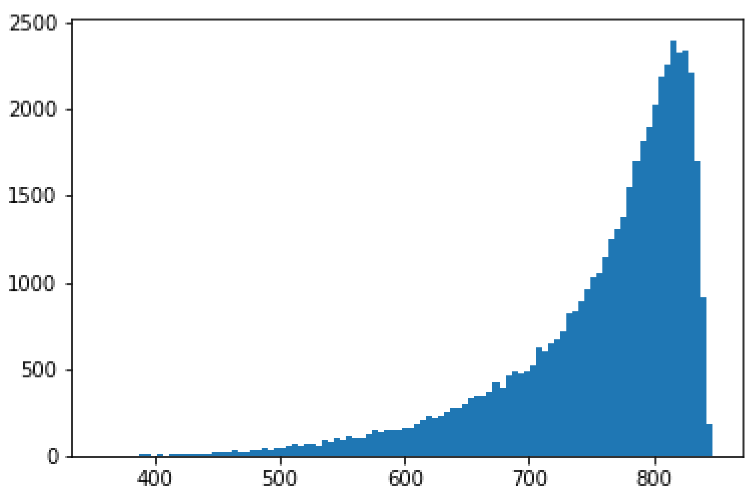
```
y_predict_optimal = df_prob[0].apply(lambda x : 0 if  x < 0.358436  else 1)

print(confusion_matrix(y_test, y_predict_optimal))
```

```
[[41452  3547]
 [ 2497  1399]]
```

```
df_prob['credit_score'] = (1-df_prob[0])* 550 + 300
```

```
plt.hist(df_prob['credit_score'], bins = 100)
```

```
(array([1.000e+00, 0.000e+00, 1.000e+00, 1.000e+00, 1.000e+00, 2.000e+00,
        5.000e+00, 4.000e+00, 2.000e+00, 7.000e+00, 3.000e+00, 8.000e+00,
        9.000e+00, 6.000e+00, 9.000e+00, 1.200e+01, 1.000e+01, 1.200e+01,
        1.600e+01, 2.200e+01, 1.900e+01, 2.700e+01, 1.900e+01, 2.500e+01,
        3.400e+01, 3.500e+01, 3.900e+01, 3.600e+01, 4.600e+01, 4.800e+01,
        5.600e+01, 6.700e+01, 5.600e+01, 6.900e+01, 6.400e+01, 5.800e+01,
        8.700e+01, 7.800e+01, 1.070e+02, 9.600e+01, 1.110e+02, 1.010e+02,
        1.020e+02, 1.280e+02, 1.460e+02, 1.370e+02, 1.430e+02, 1.490e+02,
        1.510e+02, 1.590e+02, 1.560e+02, 1.880e+02, 2.060e+02, 2.240e+02,
        2.200e+02, 2.310e+02, 2.520e+02, 2.740e+02, 2.740e+02, 2.970e+02,
        3.300e+02, 3.480e+02, 3.440e+02, 3.650e+02, 4.290e+02, 3.870e+02,
        4.570e+02, 4.880e+02, 4.710e+02, 4.880e+02, 5.210e+02, 6.220e+02,
        6.050e+02, 6.470e+02, 6.650e+02, 7.220e+02, 8.210e+02, 8.270e+02,
        8.850e+02, 9.590e+02, 1.025e+03, 1.049e+03, 1.148e+03, 1.245e+03,
        1.311e+03, 1.378e+03, 1.551e+03, 1.702e+03, 1.816e+03, 1.899e+03,
        2.024e+03, 2.187e+03, 2.260e+03, 2.400e+03, 2.330e+03, 2.333e+03,
        2.216e+03, 1.703e+03, 9.080e+02, 1.830e+02]),
 array([358.04163, 362.9333 , 367.82495, 372.7166 , 377.60828, 382.49994,
        387.39163, 392.2833 , 397.17496, 402.06662, 406.95828, 411.84995,
        416.7416 , 421.63327, 426.52493, 431.41663, 436.3083 , 441.19995,
        446.0916 , 450.98328, 455.87494, 460.7666 , 465.65826, 470.54993,
        475.4416 , 480.33325, 485.22495, 490.1166 , 495.00827, 499.89993,
        504.7916 , 509.68326, 514.57495, 519.4666 , 524.3583 , 529.24994,
        534.1416 , 539.03326, 543.9249 , 548.8166 , 553.70825, 558.5999 ,
        563.4916 , 568.38324, 573.2749 , 578.16656, 583.0582 , 587.9499 ,
        592.84155, 597.7332 , 602.6249 , 607.5166 , 612.40826, 617.2999 ,
        622.1916 , 627.08325, 631.9749 , 636.8666 , 641.75824, 646.6499 ,
        651.54156, 656.4332 , 661.3249 , 666.21655, 671.1082 , 675.9999 ,
        680.89154, 685.7832 , 690.67487, 695.5665 , 700.45825, 705.3499 ,
        710.2416 , 715.13324, 720.0249 , 724.91656, 729.8082 , 734.6999 ,
        739.59155, 744.4832 , 749.3749 , 754.26654, 759.1582 , 764.04987,
        768.9415 , 773.8332 , 778.72485, 783.6165 , 788.5082 , 793.39984,
        798.2915 , 803.1832 , 808.0749 , 812.96655, 817.8582 , 822.7499 ,
        827.64154, 832.5332 , 837.42487, 842.3165 , 847.2082 ],
       dtype=float32),
 <a list of 100 Patch objects>)
```




```
f1df['f1score'].describe()
```

```
count    48759.000000
mean         0.227151
std          0.056279
min          0.000513
25%          0.179308
50%          0.222073
75%          0.277714
max          0.316635
Name: f1score, dtype: float64
```

```
f1df.loc[f1df['f1score']>=0.277714]
```

| | precision | recall | threshold | f1score |
|---|---|---|---|---|
| **43817** | 0.283000 | 0.359343 | 0.358436 | 0.316635 |
| **43816** | 0.282943 | 0.359343 | 0.358392 | 0.316599 |
| **43815** | 0.282885 | 0.359343 | 0.358369 | 0.316563 |
| **43814** | 0.282828 | 0.359343 | 0.358318 | 0.316527 |
| **43820** | 0.282969 | 0.359086 | 0.358589 | 0.316516 |
| **...** | ... | ... | ... | ... |
| **34022** | 0.175522 | 0.664784 | 0.196916 | 0.277718 |
| **34058** | 0.175611 | 0.663501 | 0.197183 | 0.277718 |
| **34166** | 0.175883 | 0.659651 | 0.198419 | 0.277718 |
| **46226** | 0.352268 | 0.229209 | 0.448997 | 0.277717 |
| **34029** | 0.175537 | 0.664528 | 0.197009 | 0.277715 |

12190 rows × 4 columns

```
# 50% percentile
(1-0.358436)*550+300
```

```
652.8602000000001
```

```
f1df['credit_score'] = f1df['threshold']*550+300
```