Statistical Learning and Data Mining

Credit Card Loan Approval

Mahima Chadha- 21060641024

Raunak Guha- 21060641036

Contents

Problem Statement

Who should be given the loans?

Personal Loan offer company is a peer-to-peer lender that allows investors to lend money to borrowers without an intermediary being involved.

Problem1:
Good loans are defined as those listed as “Current” and defaulting loans as those listed as “Charged Off”.
We define positive outcomes as those that lead to good loans and negative outcomes as those that lead to defaults.

Problem2:
Good loans are defined as those listed as “Fully Paid” and defaulting loans as those listed as “Charged Off”.
We define positive outcomes as those that lead to good loans and negative outcomes as those that lead to defaults.

Task

1.To provide a solution to assist the company to arrive at a suitable criteria to sanction loan . Try at least two models and choose the better one (specify what criterion you will use to select your choice).

2. Identify the most important attributes this company should use in future to consider the loan approval.

3. Interpret various measures of accuracy that you will be providing with your chosen model.

Data Description and Tools

Data Description

- The dataset consists of loan borrowers from a personal loan offering company.
- The dataset has 25000 records and 135 fields of loan offering.
- Dataset has following fields member id, interest rate, loan status, loan amount, term, home ownership, hardship flags.

Tools Used: Python
Libraries Used: NumPy, Pandas, Matplotlib, Seaborn, Scikit-Learn

Methodology

1. DATA CLEANING AND PREPROCESSING
2. FEATURE SELECTION
3. VISUALIZING THE DATA
4. APPLYING CLASSIFICATION ALGORITHMS
5. EVALUATION OF MODELS

CASE-1

DATA CLEANING AND PREPROCESSING

Data Information

```
In [347]: credit.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Columns: 135 entries, Unnamed: 0 to settlement_term
dtypes: float64(100), int64(2), object(33)
memory usage: 25.7+ MB
```

After omitting the columns which have more than or equal to 45% null values the dataset has been reduced from 135 to 78 columns

```
In [233]: # Considering only those columns which have null values less than 45% in that particular column
          credit = credit[credit.columns[((credit.isnull().sum())/25000) < 0.45]]
          credit.shape

Out[233]: (25000, 78)

In [348]: credit.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Columns: 135 entries, Unnamed: 0 to settlement_term
dtypes: float64(100), int64(2), object(33)
memory usage: 25.7+ MB
```

Missing data have been dealt with by filling the categorical variables with the mode values and by filling the numeric variables with the mean values.

```
In [256]: credit.fillna(cred_num.mean(),inplace=True)

In [257]: credit.fillna(mode(cred_obj),inplace=True)

In [258]: credit.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25000 entries, 0 to 24999
Data columns (total 76 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   member_id                    25000 non-null  object
 1   loan_amnt                    25000 non-null  float64
 2   term                         25000 non-null  object
 3   int_rate                     25000 non-null  float64
 4   installment                  25000 non-null  float64
 5   grade                        25000 non-null  object
 6   sub_grade                    25000 non-null  object
 7   emp_title                    25000 non-null  object
 8   emp_length                   25000 non-null  object
 9   home_ownership               25000 non-null  object
 10  annual_inc                   25000 non-null  float64
 11  verification_status          25000 non-null  object
 12  loan_status                  25000 non-null  object
 13  pymnt_plan                   25000 non-null  object
 14  purpose                      25000 non-null  object
```

Unimportant columns like ‘id’, 'member id’ have been deleted.
Next label encoding have been done for ease of feature selection.

Feature Selection

Feature selection is done to select the optimal attributes thereby reducing the complexity and noise of the dataset making it easy for machine learning models to apply

The feature selection techniques that have been applied are:-
Variance Threshold: This is done to remove columns that have 0 variance
Pearson Correlation: This is done to find out the columns which are highly correlated and the ones which have more than 85% correlation have been removed(reducing multicollinearity).
Mutual Information Estimation: This method was applied to get the columns which contain most information about the predictor variable. From here the 20 best columns were selected.

The dataset was split into train and test with 75:25 ratio before applying these techniques in order to avoid any overfitting.

Information about the new dataset after selecting the features:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 14620 entries, 0 to 24999
Data columns (total 20 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   int_rate                 14620 non-null  float64
 1   emp_title                14620 non-null  object 
 2   title                    14620 non-null  object 
 3   initial_list_status      14620 non-null  object 
 4   last_credit_pull_d       14620 non-null  object 
 5   last_fico_range_high     14620 non-null  float64
 6   tot_coll_amt             14620 non-null  float64
 7   tot_cur_bal              14620 non-null  float64
 8   mo_sin_rcnt_rev_tl_op    14620 non-null  float64
 9   mo_sin_rcnt_tl           14620 non-null  float64
 10  mort_acc                 14620 non-null  float64
 11  num_accts_ever_120_pd    14620 non-null  float64
 12  num_actv_bc_tl           14620 non-null  float64
 13  num_bc_tl                14620 non-null  float64
 14  num_il_tl                14620 non-null  float64
 15  num_tl_30dpd             14620 non-null  float64
 16  num_tl_90g_dpd_24m       14620 non-null  float64
 17  num_tl_op_past_12m       14620 non-null  float64
 18  pct_tl_nvr_dlq           14620 non-null  float64
 19  loan_status              14620 non-null  object 
dtypes: float64(15), object(5)
memory usage: 2.3+ MB
```

Now we have to select only two values for the predictor variable “loan_status”: “Current” as positive outcome and “Charged Off” as negative outcome and have been label encoded.

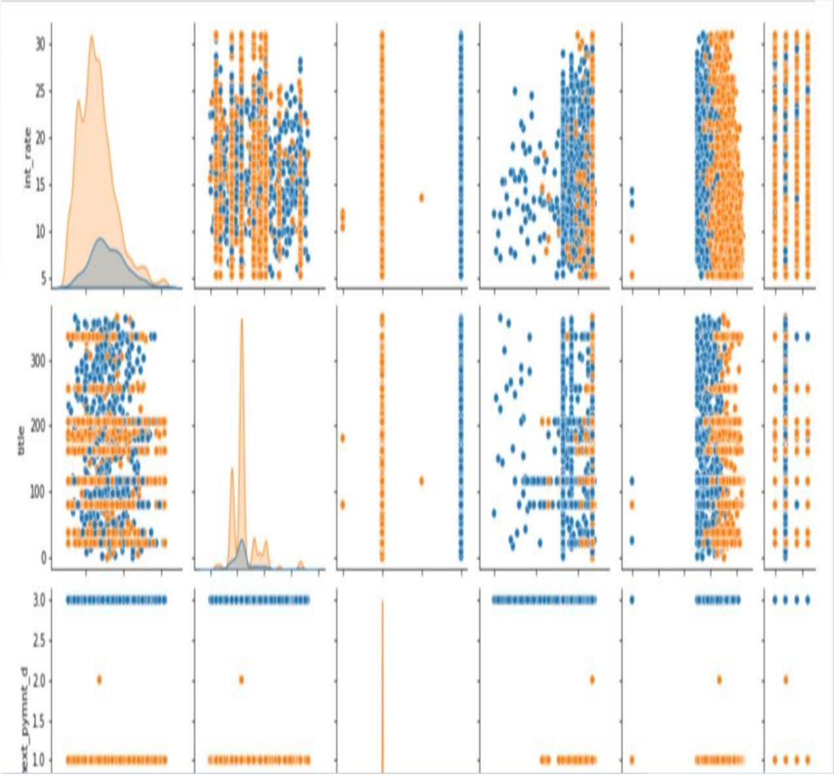
```
credit_final['loan_status'].unique()
```

```
array(['Charged Off', 'Current'], dtype=object)
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for col in credit_final.select_dtypes(include='object').columns:
    credit_final[col]=le.fit_transform(credit_final[col].astype(str))
```

Data Visualization

The dataset has been visualized to check the linearity columns have been plotted pairwise.



Selecting Classification Algorithms:

From the plot it is being found that there is too much overlapping in the data making it non-linear. Thus we cannot choose models like Logistic Regression and Support Vector Machine.
KNN and Naïve Bayes can be used for non-linear datasets but they will work well on small datasets. Moreover KNN is sensitive to outliers.
Therefore we have decided to select Tree-Based Models like Random Forest Classifier and Extreme Gradient Boosting.

Analysis

Machine Learning Techniques:

1. Random Forest:
2. XGBoost:

Model Evaluation Techniques:

1. Confusion Matrix: is a table that is used to define the performance of a classification algorithm to visualize and summarize.

2. Precision: Percentage of correct positive predictions relative to total positive predictions.
Recall: Percentage of correct positive predictions relative to total actual positives.
F1-Score: A weighted harmonic mean of precision and recall. The closer to 1, the better the model.
F1 Score=2 * (Precision * Recall) / (Precision + Recall)

3. ROC-AUC Score: It tells how much the model is capable of distinguishing between classes.

HYPERPARAMETER TUNING AND IDENTIFICATION OF OVERFITTING:

```
In [135]: #Hyperparameter Tuning  Random Forest Classifier
          model = RandomForestClassifier()
          n_estimators = [10, 100, 1000]

In [136]: grid = dict(n_estimators=n_estimators)

In [137]: grid_search_rf = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                                     scoring='r2',error_score=0,verbose=2,cv=2)

In [138]: grid_search_rf.fit(X_train, Y_train)

          Fitting 2 folds for each of 3 candidates, totalling 6 fits

Out[138]: GridSearchCV(cv=2, error_score=0, estimator=RandomForestClassifier(), n_jobs=-1,
                       param_grid={'n_estimators': [10, 100, 1000]}, scoring='r2',
                       verbose=2)

In [139]: print(f"Best: {grid_search_rf.best_score_:.3f} using {grid_search_rf.best_params_}")

          Best: 0.715 using {'n_estimators': 1000}

In [140]: means = grid_search_rf.cv_results_['mean_test_score']
          stds = grid_search_rf.cv_results_['std_test_score']
          params = grid_search_rf.cv_results_['params']

In [141]: for mean, stdev, param in zip(means, stds, params):
              print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

          0.688 (0.000) with: {'n_estimators': 10}
          0.706 (0.014) with: {'n_estimators': 100}
          0.715 (0.010) with: {'n_estimators': 1000}
```

Hyperparameter Tuning was done on both the algorithms -Random Forest Classifier and XG Boost to select the best models.

```
In [127]: #Hyperparameter Tuning For XGB
          from sklearn.model_selection import GridSearchCV

In [128]: model = xgb_cl
          n_estimators = [10, 100, 1000]

In [129]: grid = dict(n_estimators=n_estimators)

In [130]: grid_search_xgb = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                                         scoring='r2',error_score=0,verbose=2,cv=2)

In [131]: grid_search_xgb.fit(X_train, Y_train)

          Fitting 2 folds for each of 3 candidates, totalling 6 fits

Out[131]: GridSearchCV(cv=2, error_score=0,
                       estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                               callbacks=None, colsample_bylevel=1,
                                               colsample_bynode=1, colsample_bytree=1,
                                               early_stopping_rounds=None,
                                               enable_categorical=False, eval_metric=None,
                                               gamma=0, gpu_id=-1,
                                               grow_policy='depthwise',
                                               importance_type=None,
                                               interaction_constraints='',
                                               learning_rate=0.300000012, max_bin=256,
                                               max_cat_to_onehot=4, max_delta_step=0,
```

In [132]:
```
print(f"Best: {grid_search_xgb.best_score_:.3f} using {grid_search_xgb.best_params_}")
```
```
Best: 0.708 using {'n_estimators': 10}
```

In [133]:
```
means = grid_search_xgb.cv_results_['mean_test_score']
stds = grid_search_xgb.cv_results_['std_test_score']
params = grid_search_xgb.cv_results_['params']
```

In [134]:
```
for mean, stdev, param in zip(means, stds, params):
    print(f"{mean:.3f} ({stdev:.3f}) with: {param}")
```
```
0.708 (0.005) with: {'n_estimators': 10}
0.706 (0.002) with: {'n_estimators': 100}
0.697 (0.002) with: {'n_estimators': 1000}
```

In [117]:
```
#Identifying Overfitting in RF
```

In [121]:
```
print("training accuracy in RF=",accuracy_score(Y_train,rf.predict(X_train)))
print("testing accuracy in RF=",accuracy_score(Y_test,rf.predict(X_test)))
```
```
training accuracy in RF= 1.0
testing accuracy in RF= 0.9517783857729138
```

```
In [145]: print("training accuracy in XGBoost=",accuracy_score(Y_train,xgb_cl.predict(X_train)))
          print("testing accuracy in XGBoost=",accuracy_score(Y_test,xgb_cl.predict(X_test)))

training accuracy in XGBoost= 0.9718707250341997
testing accuracy in XGBoost= 0.9514363885088919
```

For both the models we get very high accuracy rate that is very low error rate for both training and test data. Thus it says that the models are good fit and we don't have to worry about overfitting or underfitting

Model Evaluations

Random Forest Classifier:

```
In [210]: print(accuracy_score(Y_test,Y_pred_rf))
          print(mean_absolute_error(Y_test,Y_pred_rf))
          print(np.sqrt(mean_squared_error(Y_test,Y_pred_rf)))

0.9504103967168263
0.04958960328317374
0.2226872319716012З
```

The Random Forest Classifier model fits 95.04% of the data. The MAE and the RMSE values are 0.0496 and 0.22 respectively.

The Confusion Matrix is shown below:

```
In [224]: # Calculate the confusion matrix
          print(confusion_matrix(Y_test, Y_pred_rf))

[[ 467   89]
 [  56 2312]]
```

The Confusion Matrix tell us that 2779 outcomes can be predicted correctly and 145 outcomes will be predicted incorrectly.

From the precision value we can infer that out of all the members that the model predicted have been sanctioned good loan,96% actually got. From the recall value we can infer that out of all the members who have been actually sanctioned good loan, the model predicted this outcome correctly for 98% of those members.
F1-Score =0.97 which is very close to 1 which tells us that the model does a very good job in predicting if the members have been sanctioned good loans.

In [225]:
```
# Calculate the classification report
print(classification_report(Y_test, Y_pred_rf))
```

	precision	recall	f1-score	support
0	0.89	0.84	0.87	556
1	0.96	0.98	0.97	2368
accuracy			0.95	2924
macro avg	0.93	0.91	0.92	2924
weighted avg	0.95	0.95	0.95	2924

Model Evaluations

XG Boost

```
In [205]: print(accuracy_score(Y_test,Y_pred_xgb))
          print(mean_absolute_error(Y_test,Y_pred_xgb))
          print(np.sqrt(mean_squared_error(Y_test,Y_pred_xgb)))

0.9514363885088919
0.04856361149110807
0.22037153058212414
```

The XG Boost model fits 95.14% of the data. The MAE and the RMSE values are 0.0486 and 0.22 respectively.

The confusion matrix is shown below.

```
In [227]: # Calculate the confusion matrix
          print(confusion_matrix(Y_test,Y_pred_xgb))

[[ 470   86]
 [  56 2312]]
```

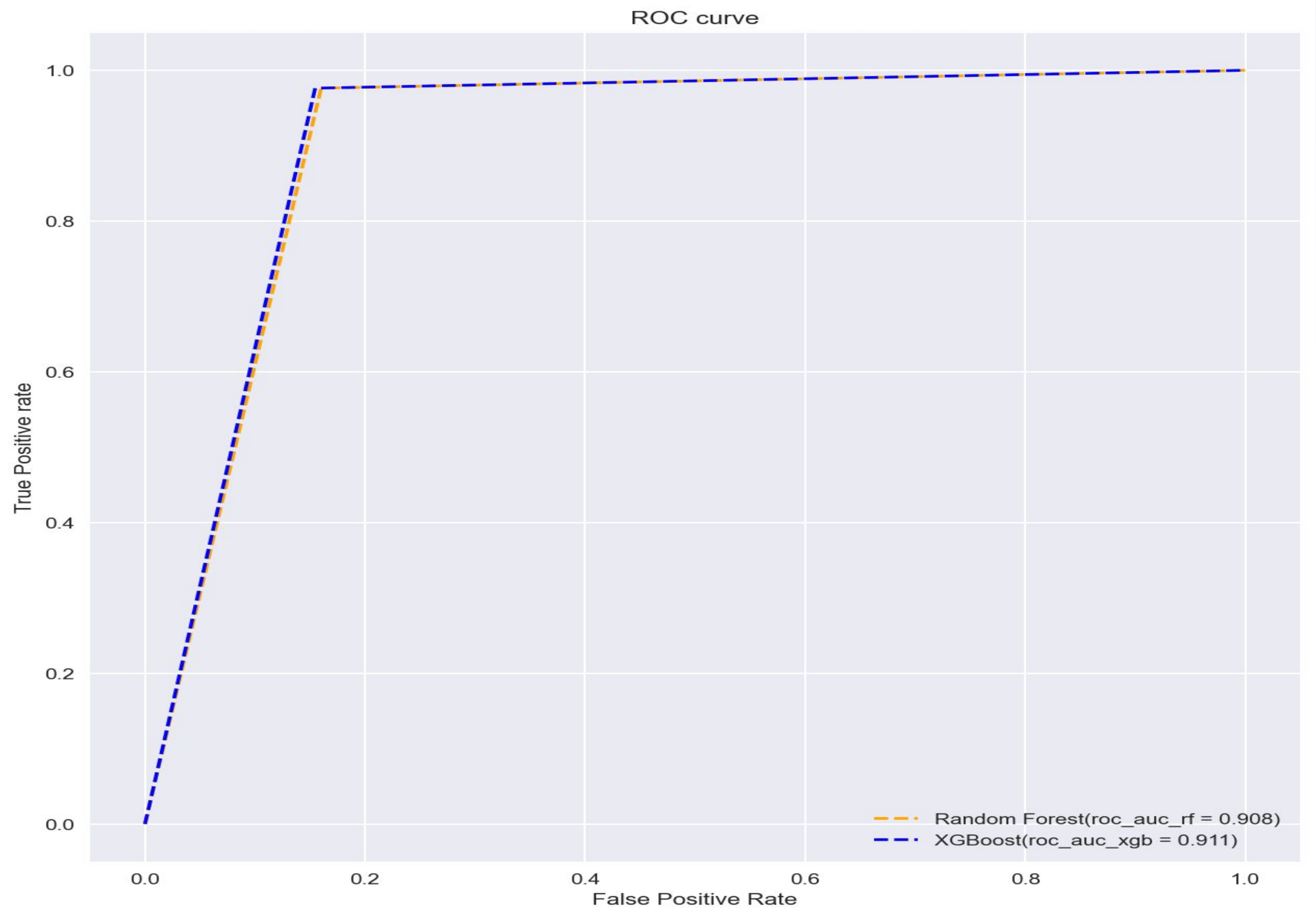
The Confusion Matrix tell us that 2782 outcomes can be predicted correctly and 142 outcomes will be predicted incorrectly.

From the precision value we can infer that out of all the members that the model predicted have been sanctioned good loan,96% actually got. From the recall value we can infer that out of all the members who have been actually sanctioned good loan, the model predicted this outcome correctly for 98% of those members.F1-score =0.97 which is very close to 1 which tells us that the model does a good job in predicting if the members have been sanctioned good loans.

In [228]:
```
# Calculate the classification report
print(classification_report(Y_test, Y_pred_xgb))
```

	precision	recall	f1-score	support
0	0.89	0.85	0.87	556
1	0.96	0.98	0.97	2368
accuracy			0.95	2924
macro avg	0.93	0.91	0.92	2924
weighted avg	0.95	0.95	0.95	2924

ROC Curve of model



CASE-2

Loan status - 'Fully Paid' as Good loans and bad loans as "Charged Off"

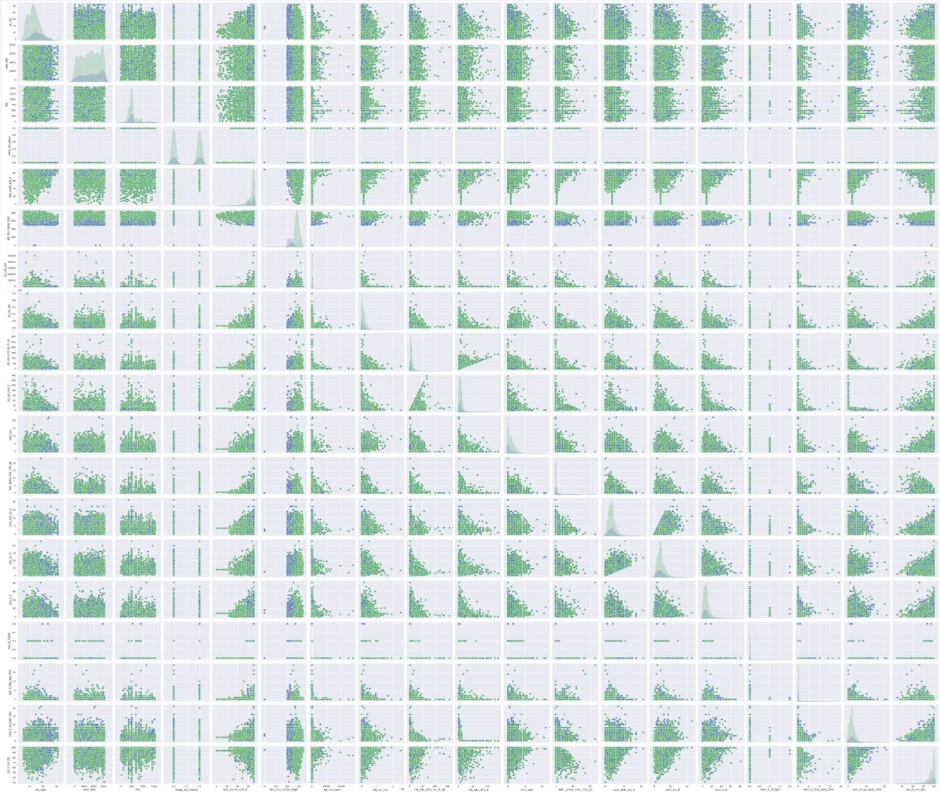
So here the value of the positive outcome got changed to "Fully Paid" from "Current". We have used the same data cleaning and pre-processing, feature selection techniques. The information about the required dataset is:

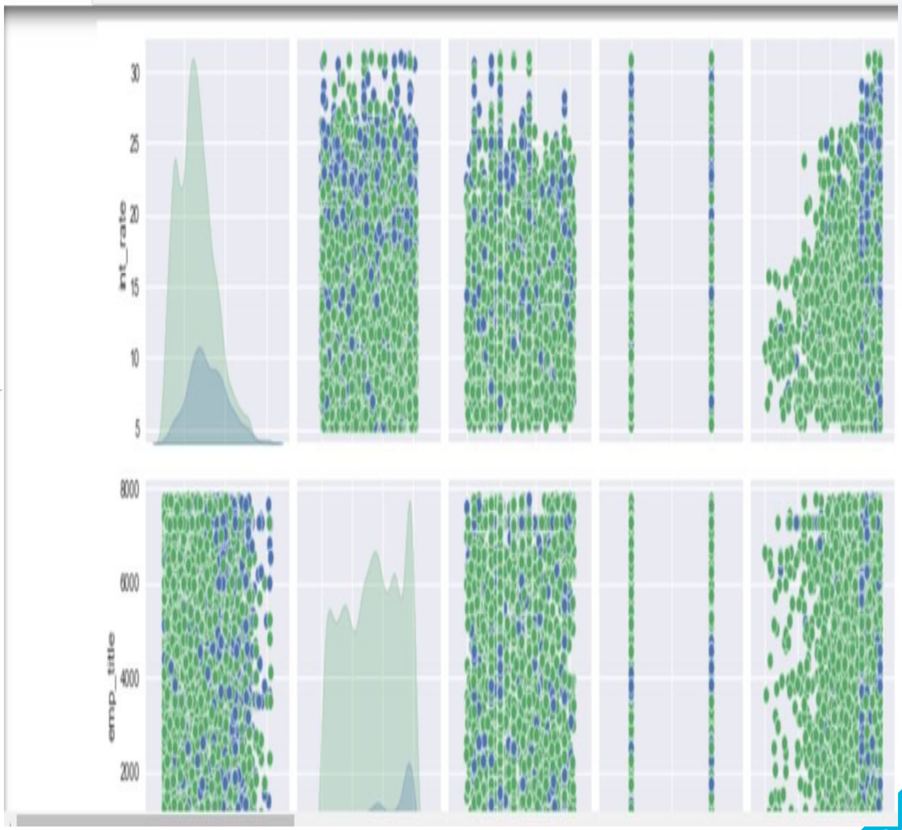
```
In [92]: credit_final['loan_status'].unique()

Out[92]: array(['Charged Off', 'Fully Paid'], dtype=object)
```

```
In [154]: credit_final.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 12290 entries, 0 to 24996
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   int_rate                12290 non-null  float64
 1   emp_title               12290 non-null  int32  
 2   title                   12290 non-null  int32  
 3   initial_list_status     12290 non-null  int32  
 4   last_credit_pull_d      12290 non-null  int32  
 5   last_fico_range_high    12290 non-null  float64
 6   tot_coll_amt            12290 non-null  float64
 7   tot_cur_bal             12290 non-null  float64
 8   mo_sin_rcnt_rev_tl_op   12290 non-null  float64
 9   mo_sin_rcnt_tl          12290 non-null  float64
 10  mort_acc                12290 non-null  float64
 11  num_accts_ever_120_pd   12290 non-null  float64
 12  num_actv_bc_tl          12290 non-null  float64
 13  num_bc_tl               12290 non-null  float64
 14  num_il_tl               12290 non-null  float64
 15  num_tl_30dpd            12290 non-null  float64
 16  num_tl_90g_dpd_24m      12290 non-null  float64
 17  num_tl_op_past_12m      12290 non-null  float64
 18  pct_tl_nvr_dlq          12290 non-null  float64
 19  loan_status             12290 non-null  int32  
dtypes: float64(15), int32(5)
memory usage: 1.7 MB
```

Data Visualization

The dataset has been visualized to check the linearity columns have been plotted pairwise





Selecting Classification Algorithms For Case 2:

From the plot it is being found again that there is too much overlapping in the data making it non-linear. Thus we cannot choose models like Logistic Regression and Support Vector Machine.
KNN and Naïve Bayes can be used for non-linear datasets but they will work well on small datasets. Moreover KNN is sensitive to outliers.
Therefore we have again decided to select Tree-Based Models like Random Forest Classifier and Extreme Gradient Boosting.

HYPERPARAMETER TUNING AND IDENTIFICATION OF OVERFITTING:

Hyperparameter Tuning was done on both the algorithms -Random Forest Classifier and XG Boost to select the best models.

```
In [134]: #Hyperparameter Tuning  Random Forest Classifier
          model = RandomForestClassifier()
          n_estimators = [10, 100, 1000]

In [135]: grid = dict(n_estimators=n_estimators)

In [136]: grid_search_rf = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                                        scoring='r2',error_score=0,verbose=2,cv=2)

In [137]: grid_search_rf.fit(X_train, Y_train)

          Fitting 2 folds for each of 3 candidates, totalling 6 fits

Out[137]: GridSearchCV(cv=2, error_score=0, estimator=RandomForestClassifier(), n_jobs=-1,
                       param_grid={'n_estimators': [10, 100, 1000]}, scoring='r2',
                       verbose=2)

In [138]: print(f"Best: {grid_search_rf.best_score_:.3f} using {grid_search_rf.best_params_}")

          Best: 0.471 using {'n_estimators': 100}

In [139]: means = grid_search_rf.cv_results_['mean_test_score']
          stds = grid_search_rf.cv_results_['std_test_score']
          params = grid_search_rf.cv_results_['params']
```

```
In [140]: for mean, stdev, param in zip(means, stds, params):
              print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

          0.432 (0.006) with: {'n_estimators': 10}
          0.471 (0.010) with: {'n_estimators': 100}
          0.470 (0.012) with: {'n_estimators': 1000}
```

```
In [126]: #Hyperparameter Tuning For XGB
          from sklearn.model_selection import GridSearchCV

In [127]: model = xgb_cl
          n_estimators = [10, 100, 1000]

In [128]: grid = dict(n_estimators=n_estimators)

In [129]: grid_search_xgb = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1,
                                         scoring='r2',error_score=0,verbose=2,cv=2)

In [130]: grid_search_xgb.fit(X_train, Y_train)

          Fitting 2 folds for each of 3 candidates, totalling 6 fits

Out[130]: GridSearchCV(cv=2, error_score=0,
                       estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                               callbacks=None, colsample_bylevel=1,
                                               colsample_bynode=1, colsample_bytree=1,
                                               early_stopping_rounds=None,
                                               enable_categorical=False, eval_metric=None,
                                               gamma=0, gpu_id=-1,
                                               grow_policy='depthwise',
                                               importance_type=None,
                                               interaction_constraints='',
                                               learning_rate=0.300000012, max_bin=256,
                                               max_cat_to_onehot=4, max_delta_step=0
```

```
In [131]: print(f"Best: {grid_search_xgb.best_score_:.3f} using {grid_search_xgb.best_params_}")

          Best: 0.463 using {'n_estimators': 10}

In [132]: means = grid_search_xgb.cv_results_['mean_test_score']
          stds = grid_search_xgb.cv_results_['std_test_score']
          params = grid_search_xgb.cv_results_['params']

In [133]: for mean, stdev, param in zip(means, stds, params):
              print(f"{mean:.3f} ({stdev:.3f}) with: {param}")

          0.463 (0.010) with: {'n_estimators': 10}
          0.458 (0.007) with: {'n_estimators': 100}
          0.436 (0.004) with: {'n_estimators': 1000}
```

```
In [117]: #Identifying Overfitting in RF

In [118]: print("training accuracy in RF=",accuracy_score(Y_train,rf.predict(X_train)))
          print("testing accuracy in RF=",accuracy_score(Y_test,rf.predict(X_test)))

          training accuracy in RF= 1.0
          testing accuracy in RF= 0.9210740439381611
```

In [143]:
```
#Identifying Overfitting in XGBoost
```

In [144]:
```
print("training accuracy in XGBoost=",accuracy_score(Y_train,xgb_cl.predict(X_train)))
print("testing accuracy in XGBoost=",accuracy_score(Y_test,xgb_cl.predict(X_test)))
```

```
training accuracy in XGBoost= 0.9343978844589097
testing accuracy in XGBoost= 0.9247355573637104
```

Again for both the models we get very high accuracy rate that is very low error rate for both training and test data. Thus it says that the models are good fit on the data and we don't have to worry about overfitting or underfitting.

Model Evaluations:

Random Forest Classifier:

```
In [116]: print(accuracy_score(Y_test,Y_pred_rf))
          print(mean_absolute_error(Y_test,Y_pred_rf))
          print(np.sqrt(mean_squared_error(Y_test,Y_pred_rf)))

0.9210740439381611
0.07892595606183889
0.28093763731803345
```

The Random Forest Classifier model fits 92.1% of the data. The MAE and the RMSE values are 0.079 and 0.28 respectively.

The Confusion Matrix is shown below

```
In [146]: # Calculate the confusion matrix
          print(confusion_matrix(Y_test, Y_pred_rf))

[[ 419   95]
 [  99 1845]]
```

The Confusion Matrix tell us that 2264 outcomes can be predicted correctly and 194 outcomes will be predicted incorrectly.

```
In [146]: # Calculate the confusion matrix
          print(confusion_matrix(Y_test, Y_pred_rf))

[[ 419   95]
 [  99 1845]]
```

From the precision value we can infer that out of all the members that the model predicted have been sanctioned good loan,95% actually got. From the recall value we can infer that out of all the members who have been actually sanctioned good loan, the model predicted this outcome correctly for 95% of those member.F1-Score =0.95 which is very close to 1 which tells us that the model does a very good job in predicting if the members have been sanctioned good loans.

```
In [147]: # Calculate the classification report
          print(classification_report(Y_test, Y_pred_rf))

              precision    recall  f1-score   support

           0       0.81      0.82      0.81       514
           1       0.95      0.95      0.95      1944

    accuracy                           0.92      2458
   macro avg       0.88      0.88      0.88      2458
weighted avg       0.92      0.92      0.92      2458
```

Model Evaluations

XG Boost

```
In [124]: Y_pred_xgb = xgb_cl.predict(X_test)

In [125]: print(accuracy_score(Y_test,Y_pred_xgb))
          print(mean_absolute_error(Y_test,Y_pred_xgb))
          print(np.sqrt(mean_squared_error(Y_test,Y_pred_xgb)))

          0.9247355573637104
          0.07526444263628966
          0.2743436579115502
```

The XG Boost model fits 92.47% of the data. The MAE and the RMSE values are 0.075 and 0.274 respectively.

The confusion matrix is shown below

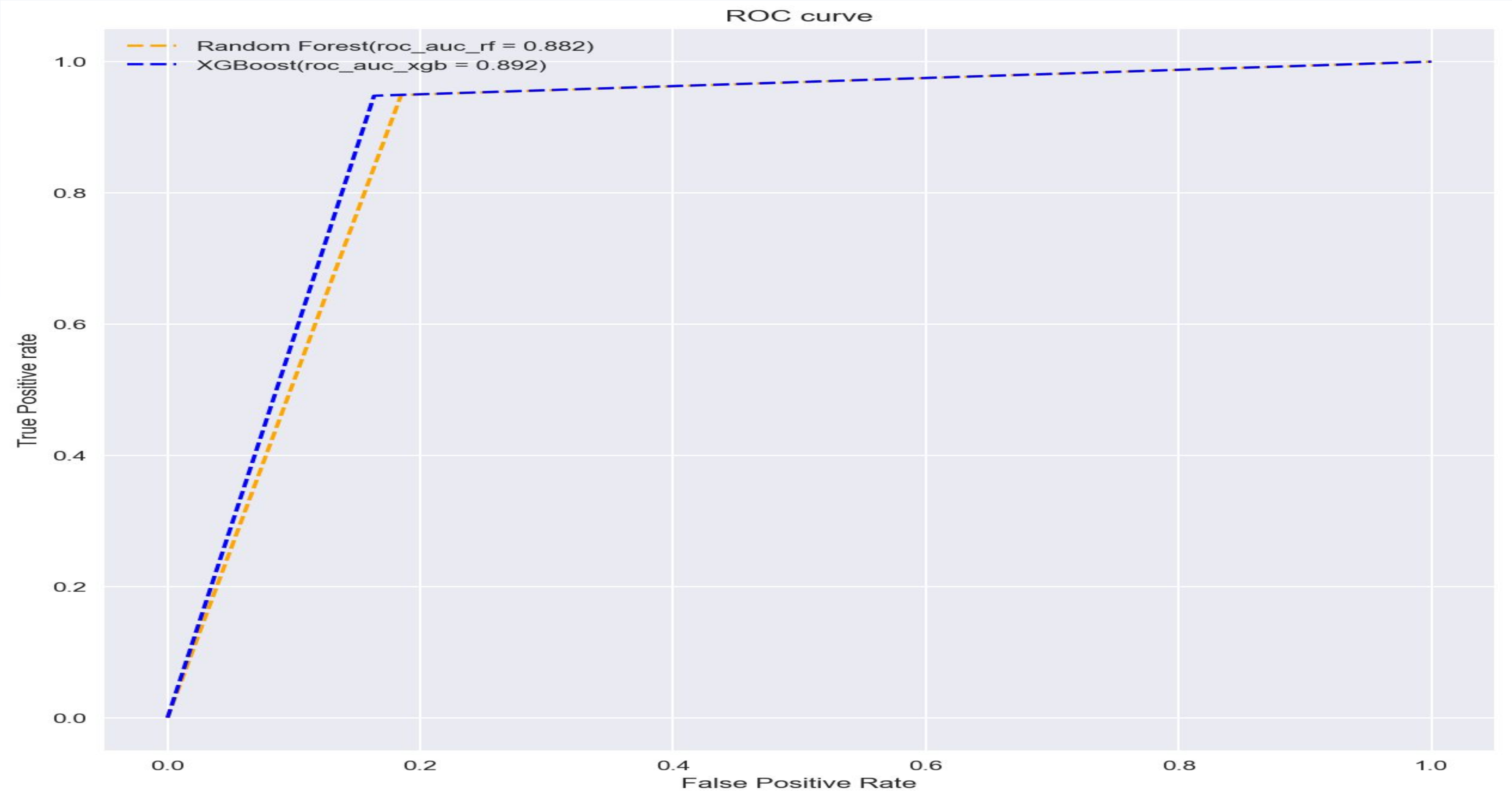
```
In [149]: # Calculate the confusion matrix
          print(confusion_matrix(Y_test,Y_pred_xgb))

          [[ 430   84]
           [ 101 1843]]
```

The Confusion Matrix tell us that 2273 outcomes can be predicted correctly and185 outcomes will be predicted incorrectly.

```
In [150]: # Calculate the classification report
          print(classification_report(Y_test, Y_pred_xgb))

              precision    recall  f1-score   support

           0       0.81      0.84      0.82       514
           1       0.96      0.95      0.95      1944

    accuracy                           0.92      2458
   macro avg       0.88      0.89      0.89      2458
weighted avg       0.93      0.92      0.93      2458
```

From the precision value we can infer that out of all the members that the model predicted have been sanctioned good loan,96% actually got. From the recall value we can infer that out of all the members who have been actually sanctioned good loan, the model predicted this outcome correctly for 95% of those member. F1-score =0.95 which is very close to 1 which tells us that the model does a good job in predicting if the members have been sanctioned good loans.

ROC Curve of the models



RESULTS AND INTERPRETATIONS FOR BOTH THE CASES

- Both Random Forest Classifier and Extreme Gradient Boost doesn't cause overfitting. They fit very well in the data with XG Boost being very marginally higher in terms of percentage of fit.
- The MAE and RMSE values for both the models are very low.
- The Confusion Matrix, precision, recall and F1-scores tell us that the models do a very good job in predicting the outcomes correctly.
- The ROC Curves are hugging the top left corner of both the models which indicate very high true Positive rate and low False Positive Rate. The high ROC-AUC scores of both the models tell us that how good the classifiers are.
- The accuracy score in case 2 is slightly lower as compared to case 1.Also the MAE and RMSE values in case 2 are slightly higher than case 1.
- However all the metrics scores in case 2 says that both the classifiers are very good here as well.

Business Proposal

- For these kind of problems the company should first deal with missing data.
- They should apply certain feature selection techniques to select the optimal number of attributes in order to reduce the complexity of the dataset.
- For this kind of dataset where the records are very high and the data is non-linear it would be a wise choice to go with bagging and boosting algorithms like Random Forest Classifiers and Extreme Gradient Boosting.
- Our results have shown that both the above classifiers are very good for getting maximum accurate predictions.

References

- Introduction to Statistical Learning
- www.analyticsvidhya.com
- www.simplilearn.com
- www.towardsdatascience.com
- www.medium.com

THANK YOU