



Loan Approval Prediction

Isaac, Fiona, Alex, Kim



Table of content

- Objective & Business Value
- Data Source (EDA, Data Processing, data cleaning)
- Model Selection, GridSearchCV
- Evaluation
- Recommendation & Conclusion



Objective

Data from bank loan borrowers dataset

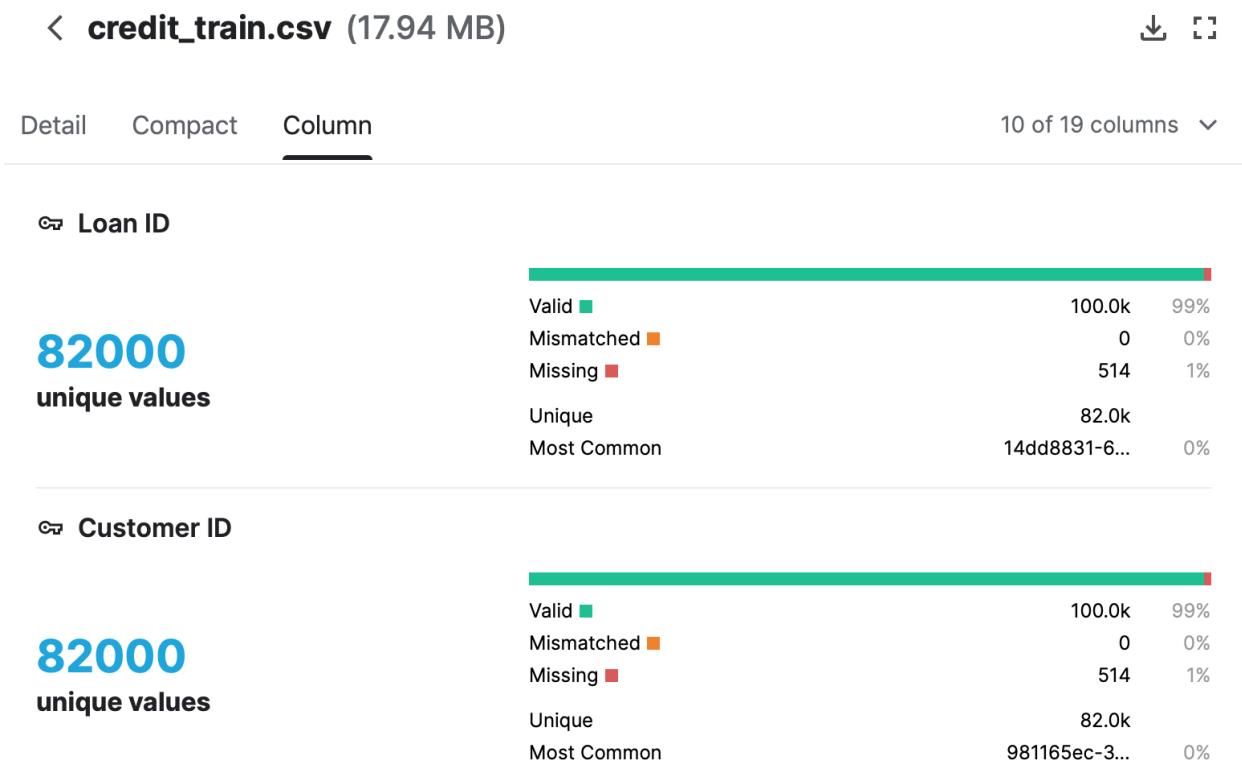
-> Develop a model to predict loan status

Business Value

- >> use the model to generate new insights and determine whether a loan should be granted or not
- > make recommendations and help reducing the risk of bad debt from bank and other companies

Dataset Source

https://www.kaggle.com/zaurbegiev/my-dataset?select=credit_train.csv



Data preprocessing-Duplicated Value

1.Drop duplicated 'Loan ID', as 'Loan ID' is unique

```
: df = df[~df["Loan ID"].duplicated()]
df = df[~df["Loan ID"].isnull()]
df.head()
```



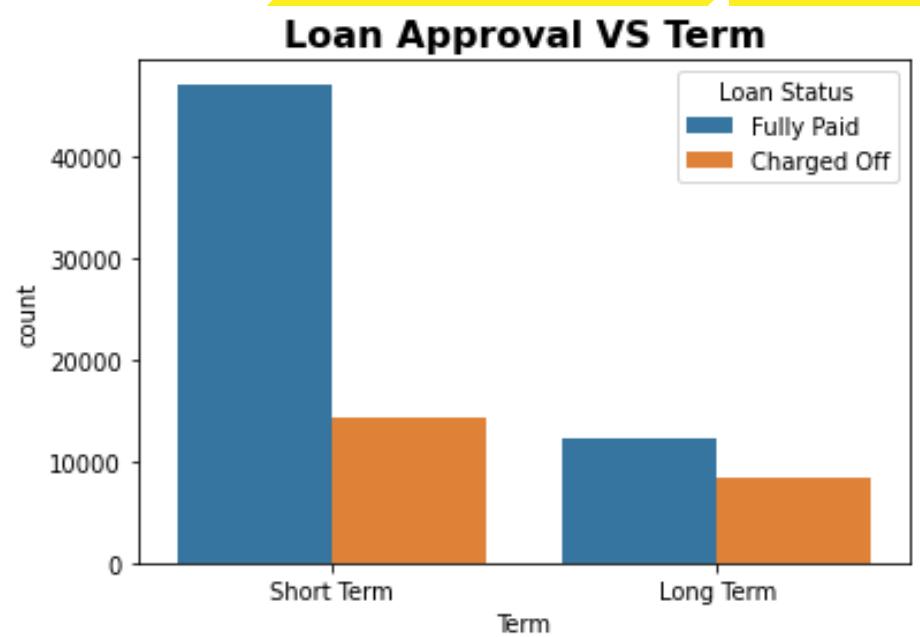
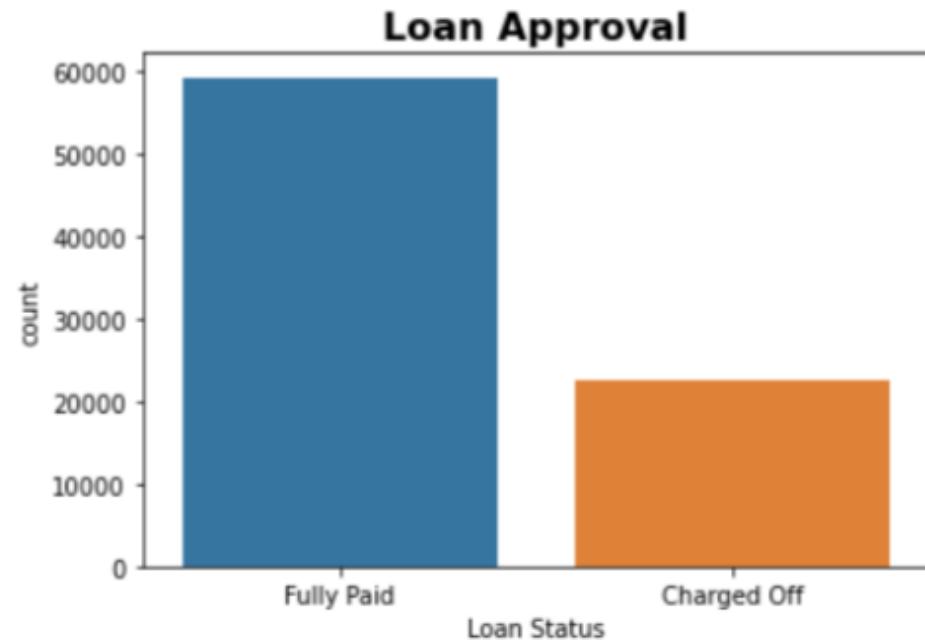
```
df.shape
```

```
(100514, 19)
```

```
df.shape
```

```
(81999, 19)
```

EDA



Data preprocessing-Missing Value

```
df.isnull().sum()  
  
Loan ID           0  
Customer ID      0  
Loan Status       0  
Current Loan Amount 0  
Term              0  
Credit Score     17031  
Annual Income     17031  
Years in current job 3508  
Home Ownership    0  
Purpose            0  
Monthly Debt      0  
Years of Credit History 0  
Months since last delinquent 44621  
Number of Open Accounts 0  
Number of Credit Problems 0  
Current Credit Balance 0  
Maximum Open Credit 2  
Bankruptcies      175  
Tax Liens          8  
dtype: int64
```

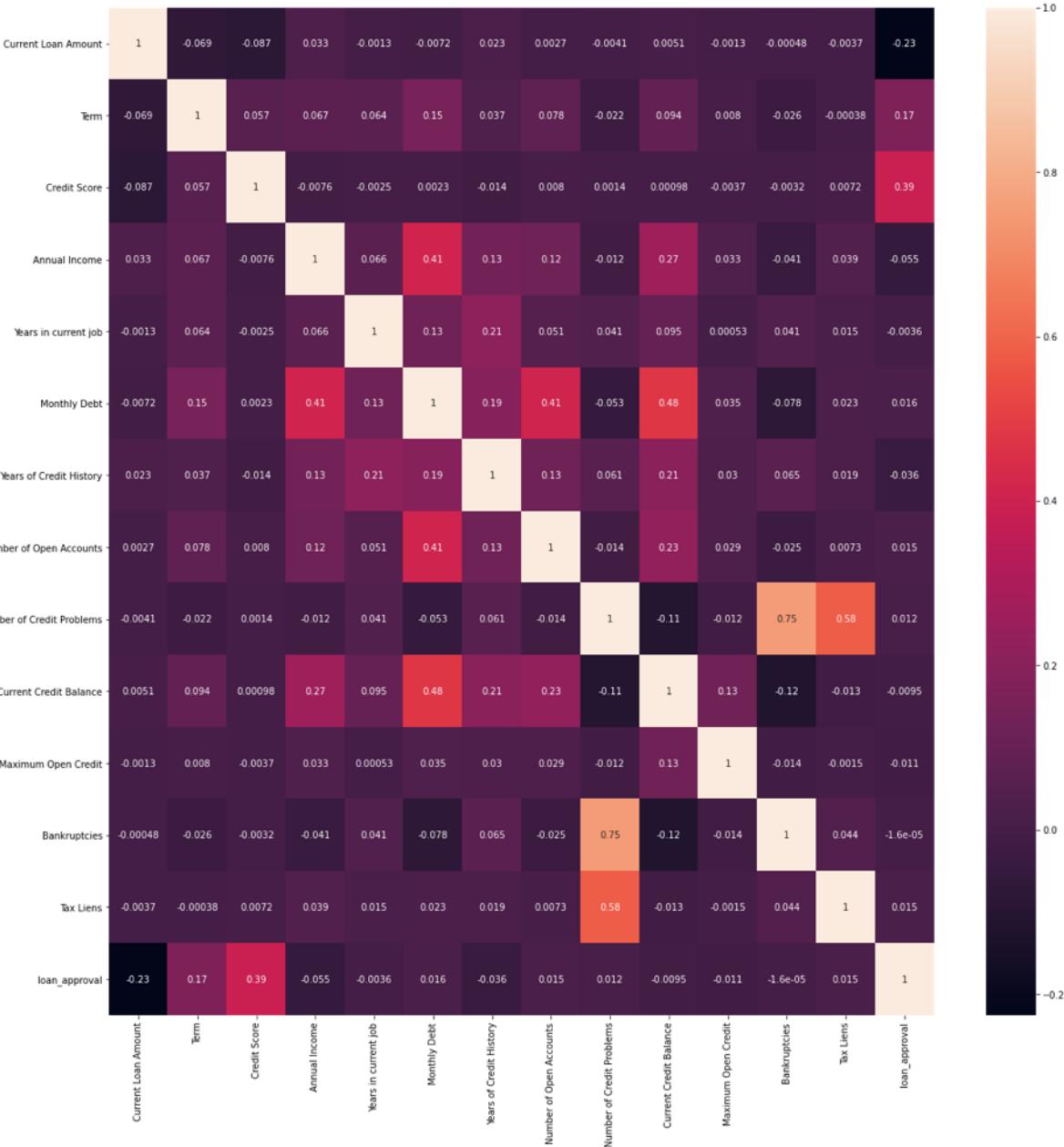
→

```
: #impute median  
df["Credit Score"] = df["Credit Score"].fillna(df["Credit Score"].median())  
df["Annual Income"] = df["Annual Income"].fillna(df["Annual Income"].median())  
df["Maximum Open Credit"] = df["Maximum Open Credit"].fillna(df["Maximum Open Credit"].median())  
df["Years in current job"] = df["Years in current job"].fillna(df["Years in current job"].median())  
  
: #impute mode  
df["Tax Liens"] = df["Tax Liens"].fillna(float(df["Tax Liens"].mode()))  
df["Bankruptcies"] = df["Bankruptcies"].fillna(float(df["Bankruptcies"].mode()))
```

→

```
df.isnull().sum()  
  
Loan ID           0  
Customer ID      0  
Loan Status       0  
Current Loan Amount 0  
Term              0  
Credit Score      0  
Annual Income      0  
Years in current job 0  
Home Ownership     0  
Purpose            0  
Monthly Debt       0  
Years of Credit History 0  
Number of Open Accounts 0  
Number of Credit Problems 0  
Current Credit Balance 0  
Maximum Open Credit 0  
Bankruptcies       0  
Tax Liens          0  
dtype: int64
```

Correlation Matrix between numeric variable and target variable



Data preprocessing-Encoding Method

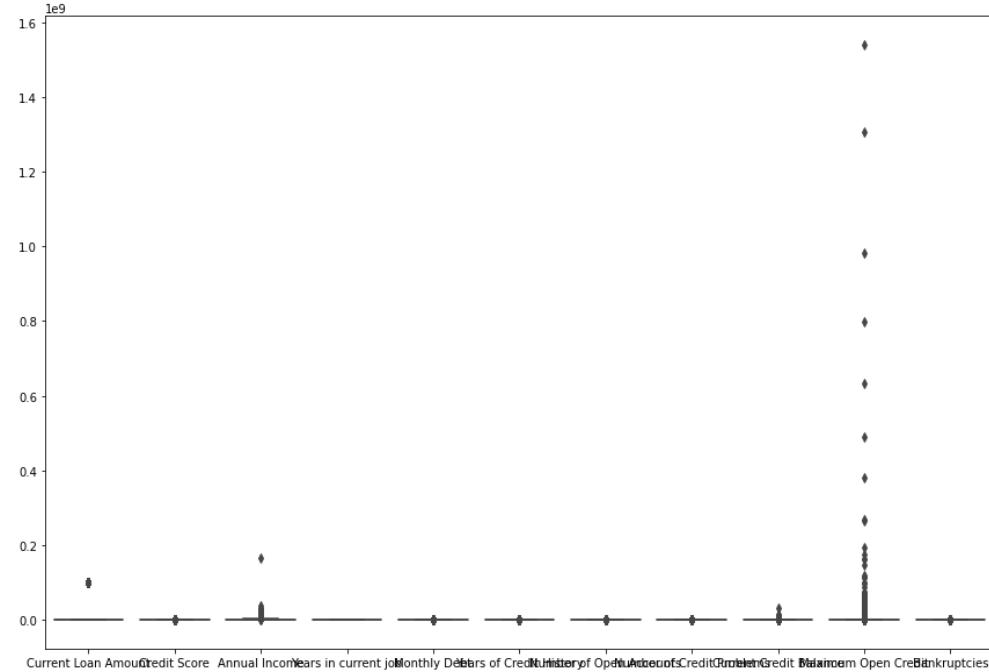
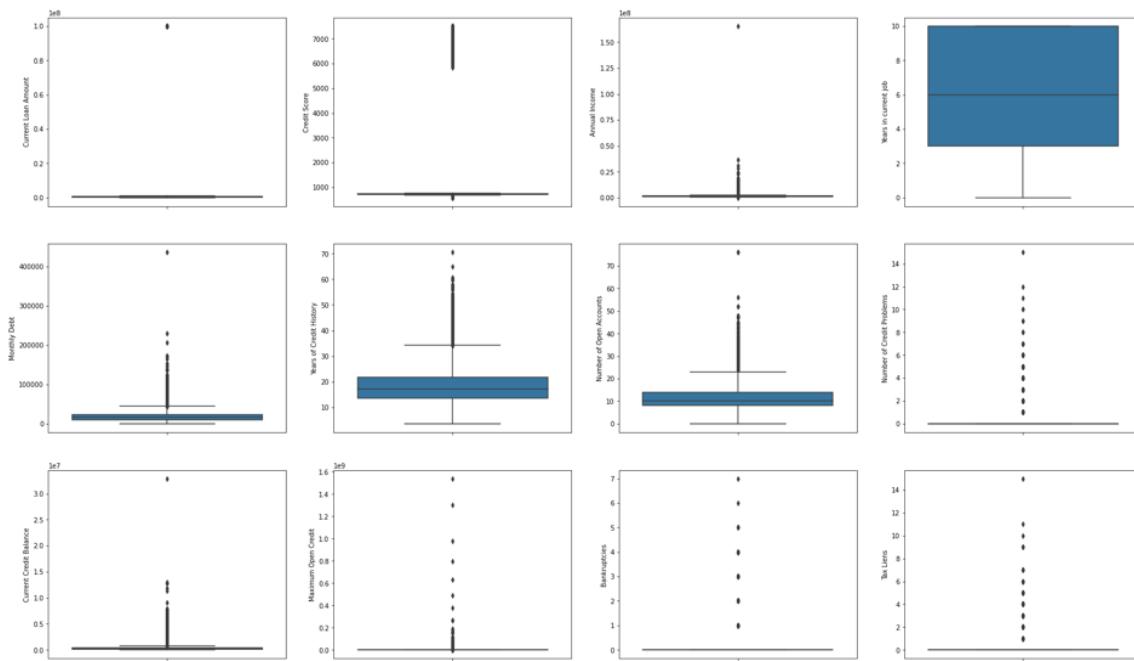
OneHotEncoder: Home Ownership,Purpose

Ordinal Encoder: Term

Label Encoder: Loan Status

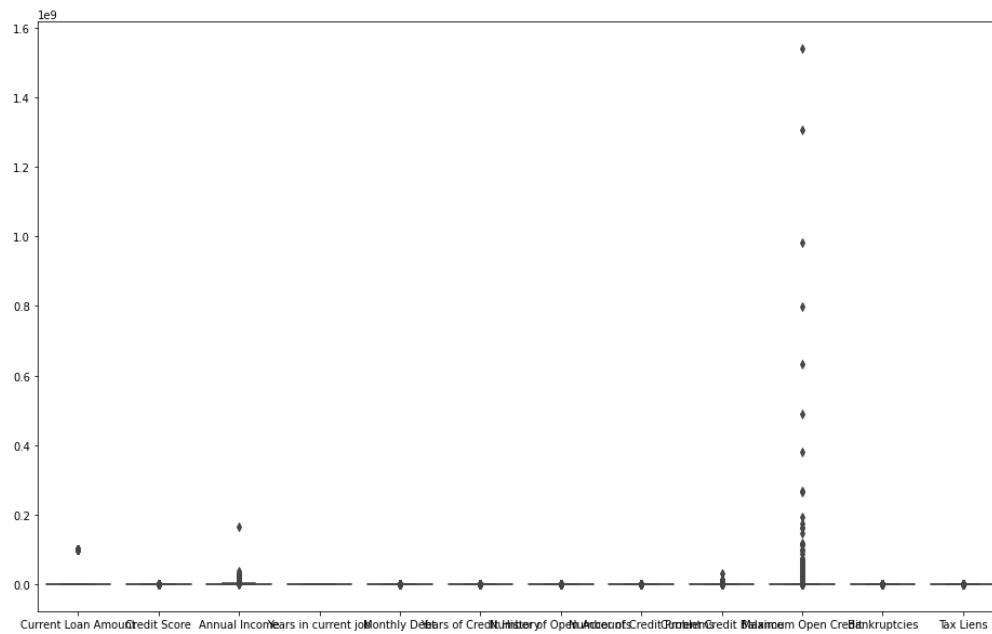
Data preprocessing

-MinMaxScaler

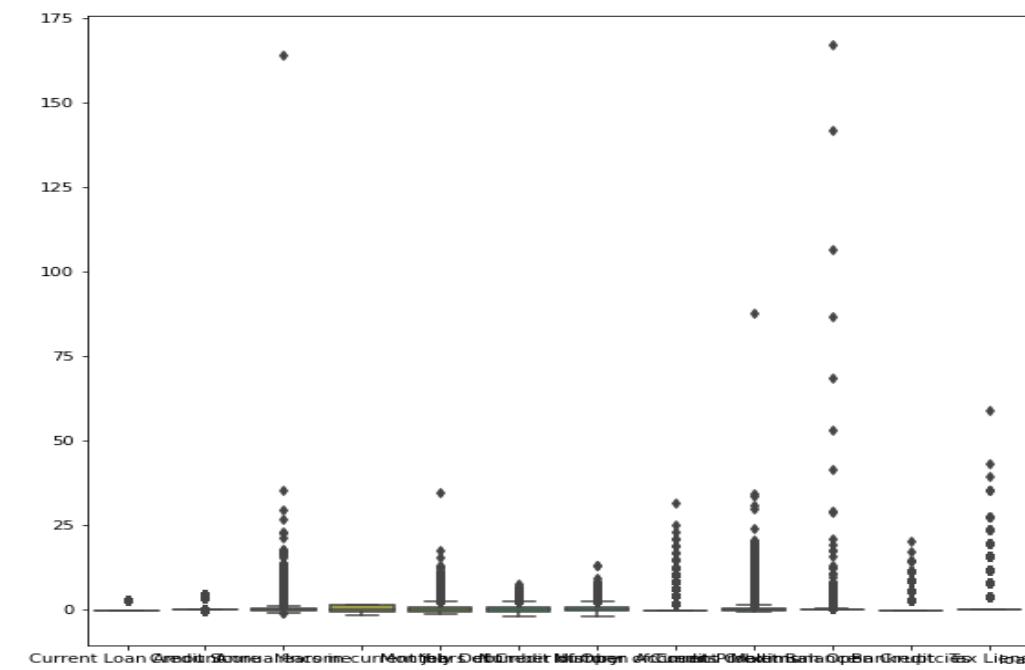


Data preprocessing-MinMaxScaler

Before Scaling



After Scaling



Model Selection

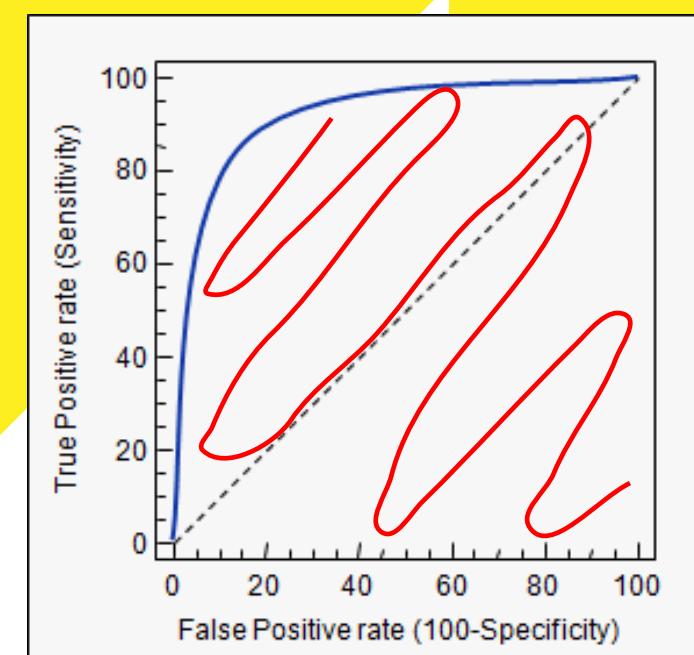
- X (features) = All features (Annual income, no. of credit card, current credit balance etc)
- y (target) = Default (1), Passed (0)

1. Logistic Regression
2. KNN Classifier
3. Decision Tree Classifier
4. Random Forest Classifier
5. AdaBoost Classifier
6. XGBoost Classifier

Model Selection

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=df["Loan Status"])
```

- GridSearchCV to find the best parameters for each classifier model for prediction
- Comparison of 'ROC_AUC' score
 - The higher the AUC score, the higher predictability
 - Different thresholds score



GridSearchCV

```
from sklearn.linear_model import LogisticRegression
parameters = {'C' : np.linspace(0.1,100,10), "penalty":["l1","l2"]}

from sklearn.neighbors import KNeighborsClassifier
parameters = {'n_neighbors': np.arange(1, 12, 2),
              'weights': ['uniform', 'distance']}

from sklearn.tree import DecisionTreeClassifier

parameters={"criterion": ["gini", "entropy"],
            "min_samples_split": [10, 20, 40],
            "max_depth": [2, 6, 8],
            "min_samples_leaf": [20, 40, 100],
            "max_leaf_nodes": [5, 20, 100],
            }

from sklearn.ensemble import RandomForestClassifier
parameters={

    "n_estimators"      : [30,40,50],
    "max_features"      : ["sqrt", "log2"],
    "min_samples_split" : [6,8,10,12,14]
    }

from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier()
parameters = {'n_estimators': [300, 400],
              'learning_rate': [0.3, 0.4]
              }

xgmodel = XGBClassifier()

params = {'learning_rate': [0.001, 0.01, 0.1],
          'max_depth': [3,4]
          }
```

GridSearchCV

Tuned parameters

'ROC_AUC' score

```
LogisticRegression 0.5944481309444813
KNeighborsClassifier 0.6104537478559476
DecisionTreeClassifier 0.6028633780355367
RandomForestClassifier 0.6122866884601849
AdaBoostClassifier 0.598215846324359
XGBClassifier 0.5970091886672886
```

Default parameters

```
{'logreg': 0.600191054403627,
 'knn': 0.6190772837236768,
 'tree': 0.6251851313897117,
 'rf': 0.6298774084938996,
 'ada': 0.6164253212118903,
 'xgb': 0.6344905136532911}
```

Random Forest Classifier

XGBoost Classifier

Model Evaluation

XGBoost Classifier

Confusion Matrix:
(0: Pass, 1: Default)

Threshold = 0.5

Test dataset(Total): 16,400 samples:

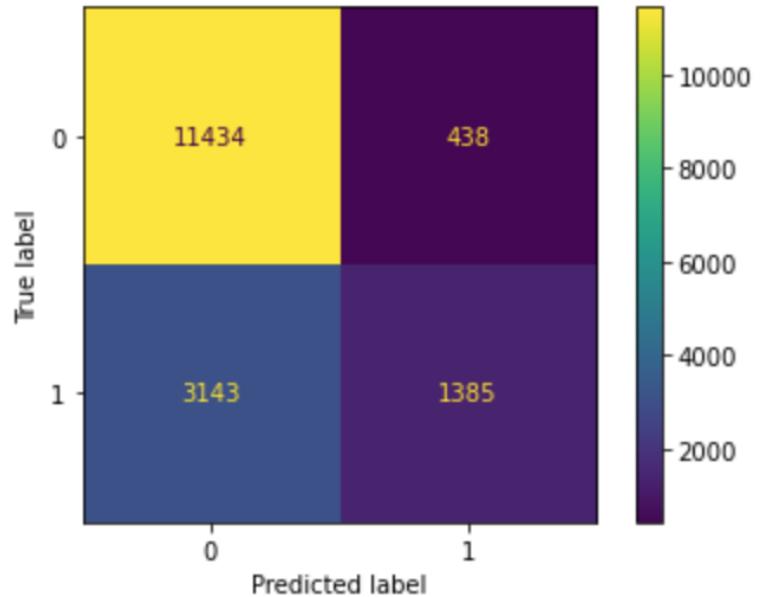
True Positive(TP) = 1,385

11,434

False Negative(FN) = 3,143

True Negative(TN) =

False Positive(FP) = 438



| Accuracy | Precision | Recall / True Positive Rate | False Positive Rate |
|---|--|--|--|
| $(TP+TN) / (\text{Total})$ $= (1,385+11,434) / (16,400)$ $= 0.7816$ | $(TP) / (TN+FP)$ $= (1,385) / (11,434+438)$ $= 0.7597$ | $(TP) / (TP+FN)$ $= (1,385) / (1,385+3,143)$ $= 0.3058$ → Adjust Threshold | $(FP) / (FP+TN)$ $= (438) / (11,434+438)$ $= 0.0369$ |

[1385, 438]
[3143, 11434]
The accuracy_score: 0.7816
precision: 0.75973669775096
recall: 0.30587455830388693

ROC and Precision Recall AUC

| Threshold | Recall | Precision | Successful find-out (e.g. 10 Real Default Cases) | | |
|-----------|--------|-----------|---|--------------------|--------|
| | | | False Positive Rate | True Positive Rate | |
| 0.1 | 0.9759 | 0.3347 | ~ 9 - 10 | 0.74 | 0.9759 |
| 0.2 | 0.845 | 0.3911 | ~ 8 - 9 | 0.5018 | 0.845 |
| 0.3 | 0.6151 | 0.4959 | ~ 6 | 0.2385 | 0.6151 |
| 0.4 | 0.4218 | 0.6122 | ~ 4 | 0.1019 | 0.4218 |
| 0.5 | 0.3059 | 0.7597 | ~ 3 | 0.0369 | 0.3059 |
| 0.6 | 0.2405 | 0.8993 | ~ 2 - 3 | 0.0103 | 0.2405 |
| 0.7 | 0.2102 | 0.9714 | ~ 2 | 0.0024 | 0.2102 |
| 0.8 | 0.2008 | 0.9945 | ~ 2 | 0.0004 | 0.2008 |
| 0.9 | 0.1999 | 1 | ~ 0 | 0 | 0.1999 |

Extremely strict

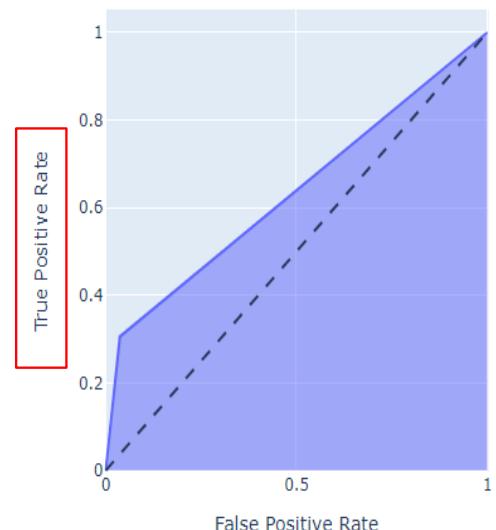
Strict

moderate

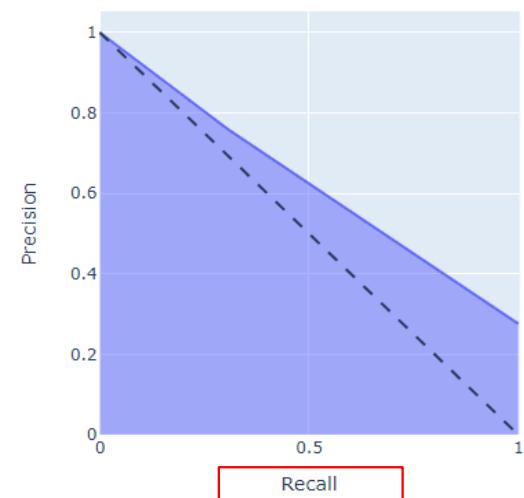
Loose

Extremely loose

ROC Curve (AUC=0.6345)



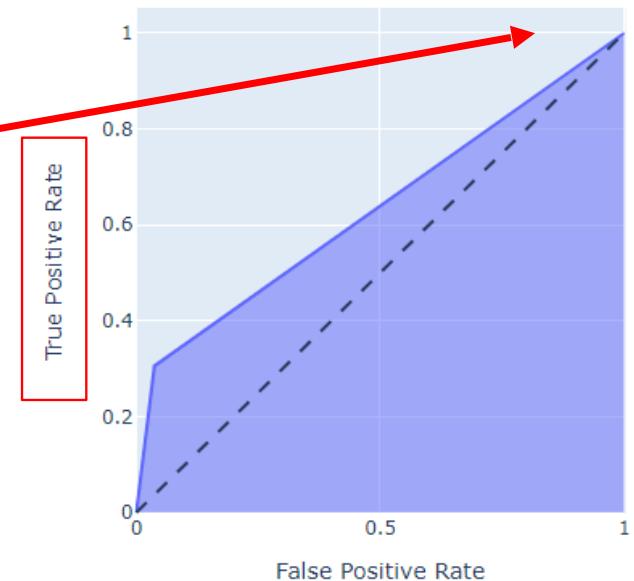
Precision-Recall Curve (AUC=0.6345)



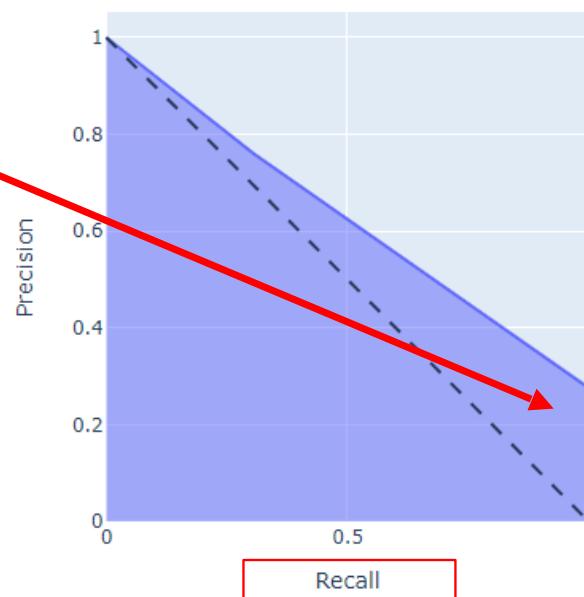
Tradeoff & Threshold

- Make own preference / risk tolerance on the balance between Precision and Recall Rate
- A bank aim to minimize default risk strictly
 - > lower the model threshold
 - = higher Recall Rate
 - = more cases rejected

ROC Curve (AUC=0.6345)



Precision-Recall Curve (AUC=0.6345)



Tradeoff:

- Less chance to approve the loan application
- Revenue may affect

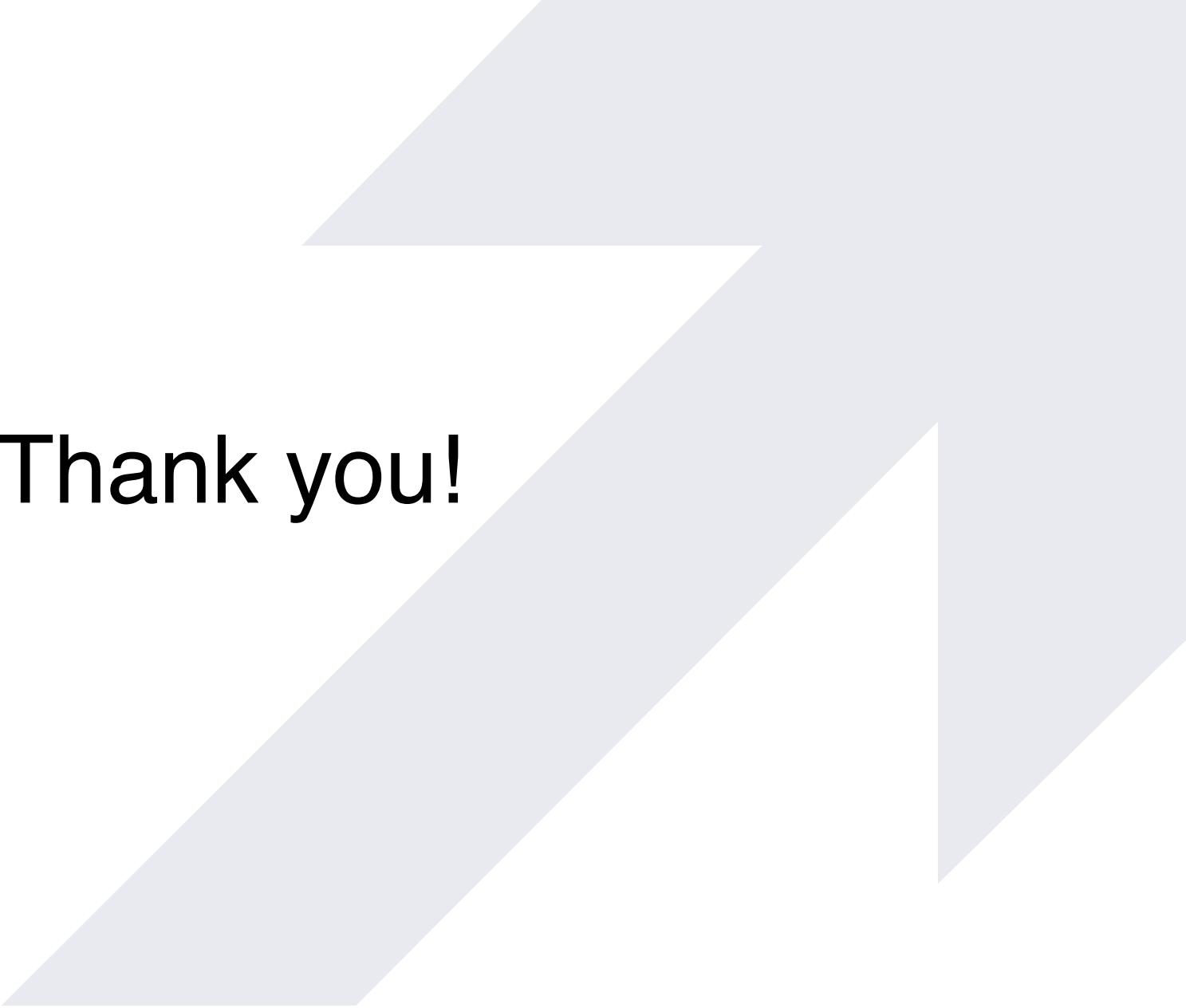
Limitation & Reflection

- Many other factors would affect the default rate, not only the factors/ features in individual level, perhaps the factors in other aspects,

E.g. Economy situation

-> Business worse -> Unemployment -> Default rate increase.

- However, our dataset is limited in the personal aspect only, We ignored the influence of other factors



Thank you!