Introduction to Machine Learning

Report: Loan Default Prediction Project

University of Arizona, Tucson

Contents:

Viren Sasalu
virensasalu@gmail.com

Introduction:

For a country to have a flourishing economy, banking finance sector of the company must be strong. The roll out of credit and repayment makes up most of the profit of the bank. Credits/loans and repayment gives a fantastic insights about the intricate systems of the economy.

Defaults mean that there is an issue with the flow of money and would disrupt the balance of the bank. If left unaddressed, it could collapse the country's economy by contributing to affects like increase in **interest rates**, **tax reforms** and **inflation**.

To put a stop to this, banks need to device a system that predicts defaulters. That is what we are going to discuss in this report by analysing the dataset of **German Bank** which has valuable information about the domain and the critical challenges faced by the bank in identifying potential defaulters. In this report, we will look into some of the approaches taken to understand the dataset by exploring the data, understanding the relation and use some **Machine Learning Models** with the help of graphical representation to predict defaulters.

Data Description:

The bank has historical information on relevant features for each customer such as employment duration, existing loans count, saving balance, percentage of income, age, default status.

The data set has 17 columns and 1000 rows. Columns are described below and each row is a customer. This data set contains historical data of the customers who have taken loans from a German bank and the bank is facing issues with loan defaulters.

- checking_balance - Amount of money available in account of customers
- months_loan_duration - Duration since loan taken
- credit_history - credit history of each customers
- purpose - Purpose why loan has been taken
- amount - Amount of loan taken
- savings_balance - Balance in account
- employment_duration - Duration of employment
- percent_of_income - Percentage of monthly income
- years_at_residence - Duration of current residence
- age - Age of customer
- other_credit - Any other credits taken
- housing- Type of housing, rent or own
- existing_loans_count - Existing count of loans
- job - Job type
- dependents - Any dependents on customer
- phone - Having phone or not
- default - Default status (Target column)

Some of the Questions I have answered throughout the program:

- What are some of the best ML models right for this dataset?
- How will I determine the right model amongst all the models?
- How does some features effect the 'default'?
- what are the feature importance, using Feature importance?
- Perform complete EDA.
- How will you address the dataset to fit the model. Will you use one-hot-encoding? or Map each attributes?
- How can you determine if the models are under fitting/overfitting?
- How are you accessing the accuracy and laying all the models on heat map?
- What steps will you take to compare the models
- Create a dummy dataset and predict the defaults(Deployment proof)?
- Are there any features that are considered important but the data doesn't agree? How do you prove it?

Methods and Materials

Exploratory Data Analysis(EDA):

After I imported all the necessary libraries pre-emptively, I loaded the dataset from the dataset 'German loan data'. Later on, I performed a well through scan of the dataset to identify any obvious issue before I begin with Data Understanding and Cleaning.

Data Understanding, Cleaning and Pre-Paring/Pre-Processing:

I performed some rudimentary steps to understand the dataset, good amount of information was already provided in the problem statement, discussing each features. I conducted a comprehensive exploration on the dataset to identify only a few typographic errors. Further, I retrieved data information [*print(data.info())*] to understand what kind of dataset I was dealing with. Followed by that, I looked into any missing values [*print(data.isnull().sum())*] so that I can arrive at how I wanted to handle it depending on the datatype. It appeared there were no missing values.

Next step was to check the unique values in each feature. So I ran a for loop to understand the unique values. This is where I found out that '**car0'** was a typographic error and I replaced it [*data['purpose'] = data['purpose'].replace('car0', 'car')*].

```
print(data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64 
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64 
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64 
 8   years_at_residence    1000 non-null   int64 
 9   age                   1000 non-null   int64 
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64 
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64 
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
None
```

```
print(data.isnull().sum())
```

```
checking_balance        0
months_loan_duration    0
credit_history          0
purpose                 0
amount                  0
savings_balance         0
employment_duration     0
percent_of_income       0
years_at_residence      0
age                     0
other_credit            0
housing                 0
existing_loans_count    0
job                     0
dependents              0
phone                   0
default                 0
dtype: int64
```

```
for column in data.columns:
    unique_values = (data[column].unique().sum())
    print(f"Unique values in {column}:{unique_values}")
```

```
Unique values in checking_balance:< 0 DM1 - 200 DMunknown> 200 DM
Unique values in months_loan_duration:862
Unique values in credit_history:criticalgoodpoorperfectvery good
Unique values in purpose:furniture/applianceseducationcarbusinessrenovations
Unique values in amount:3118140
Unique values in savings_balance:unknown< 100 DM500 - 1000 DM> 1000 DM100 - 500 DM
Unique values in employment_duration:> 7 years1 - 4 years4 - 7 yearsunemployed< 1 year
Unique values in percent_of_income:10
Unique values in years_at_residence:10
Unique values in age:2394
Unique values in other_credit:nonebankstore
Unique values in housing:ownotherrent
Unique values in existing_loans_count:10
Unique values in job:skilledunskilledmanagementunemployed
Unique values in dependents:3
Unique values in phone:yesno
Unique values in default:noyes
```

There was another pressing issue, when I found out that '**unknown'** was present between the dataset. At first I decided to drop it but found out that 'unknown' had a frequency (number of occurance) of 394 [*data.describe(include=['O']).T*].

```
data.describe(include=['O']).T
```

	count	unique	top	freq
checking_balance	1000	4	unknown	394
credit_history	1000	5	good	530
purpose	1000	6	furniture/appliances	473
savings_balance	1000	5	< 100 DM	603
employment_duration	1000	5	1 - 4 years	339
other_credit	1000	3	none	814
housing	1000	3	own	713
job	1000	4	skilled	630
phone	1000	2	no	596
default	1000	2	no	700

Dropping this would mean a major chunk of data set would be lost. There was no other way of handling this as it might affect the models behaviour. I made a decision to keep 'unknown' as an attribute.

Furthermore, I performed a numerical qualitative analysis to understand the data more precisely. [*data.describe().T*]

```
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
months_loan_duration	1000.0	20.903	12.058814	4.0	12.0	18.0	24.00	72.0
amount	1000.0	3271.258	2822.736876	250.0	1365.5	2319.5	3972.25	18424.0
percent_of_income	1000.0	2.973	1.118715	1.0	2.0	3.0	4.00	4.0
years_at_residence	1000.0	2.845	1.103718	1.0	2.0	3.0	4.00	4.0
age	1000.0	35.546	11.375469	19.0	27.0	33.0	42.00	75.0
existing_loans_count	1000.0	1.407	0.577654	1.0	1.0	1.0	2.00	4.0
dependents	1000.0	1.155	0.362086	1.0	1.0	1.0	1.00	2.0

Data Visualization: 'Datavis'

Most important part of **EDA** is visualizing the data. This provides us the opportunity to learn the patterns of the dataset. I performed various types of graphical analysis to draw inferences and to understand the behaviours of each feature. Certain features proved to provide insights with the predictor 'default' also intrigued me at some points. This behaviours raised a lot of questions prompting me to understand the Banking Industry and the dataset itself. After my own, research to understand how the banking sectors work and attain some basic knowledge about the domain, I was set.

Pair plots: A type of data visualization that display pairwise relationships among a set of variables. They are particularly useful when dealing with multivariate data, allowing you to quickly identify patterns, correlations, and potential outliers.

```
Features = ['months_loan_duration', 'amount', 'percent_of_income', 'age', 'existing_loans_co
sns.pairplot(ProcessedBankData, hue='default', vars=Features)
plt.show()
print('No: 0')
print('Yes: 1')
```

Here, I decided to address only the numerical features to check the relationship with 'default'.

Features=['months_loan_duration', 'amount', 'percent_of_income', 'age', 'existing_loans_count' 'dependents']

Interesting observations between existing loans and amount. Some obvious outlies were found with age- feature. I decided to keep it. This wild observations will help the ML models perform better at predicting.

It was obvious that a **Corelation Matrix** and a **Heat Map** would help a lot in understanding the data.

```
corr_matrix = ProcessedBankData[['months_loan_duration', 'amount', 'percent_of_income', 'age
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Heatmap')
plt.show()
```

In [82]:
```
corr_matrix
```

Out[82]:

months_loan_duration	amount	percent_of_income	age	existing_loans_count	dependents	purpose	credit_history
1.000000	0.624984	0.074749	-0.036136	-0.011284	-0.023834	0.148723	0.077186
0.624984	1.000000	-0.271316	0.032716	0.020795	0.017142	0.198534	0.059905
0.074749	-0.271316	1.000000	0.058266	0.021669	-0.071207	-0.077530	-0.044375
-0.036136	0.032716	0.058266	1.000000	0.149254	0.118201	0.101045	-0.147086
-0.011284	0.020795	0.021669	0.149254	1.000000	0.109667	0.121013	-0.437066
-0.023834	0.017142	-0.071207	0.118201	0.109667	1.000000	0.109593	-0.011550
0.148723	0.198534	-0.077530	0.101045	0.121013	0.109593	1.000000	0.037127
0.077186	0.059905	-0.044375	-0.147086	-0.437066	-0.011550	0.037127	1.000000

My Observations:

months_loan_duration and amount (0.624984): There is a relatively strong positive correlation (0.62) between the duration of the loan in months and the loan amount. This suggests that, on average, as the loan duration increases, the loan amount tends to increase. as well.

months_loan_duration and percent_of_income (0.074749)**:** There is a weak positive correlation (0.07) between the loan duration. and the percentage of income. The correlation is not strong, suggesting a relatively weak relationship.

amount and percent_of_income (-0.271316)**:** There is a moderate negative correlation (-0.27) between the loan amount and the percentage of income. This suggests that, on average, as the loan amount increases, the percentage of income used to pay the loan. tends to decrease.

age and existing_loans_count (0.149254): There is a weak positive correlation (0.15) between age and the count of existing loans. This suggests that, on average, as age increases, the number of existing loans tends to also increase slightly.

existing_loans_count and dependents (0.109667): There is a weak positive correlation (0.11) between the count of existing loans. and the number of dependents. This suggests that, on average, as the count of existing loans increases, the number of dependents. also increases slightly.

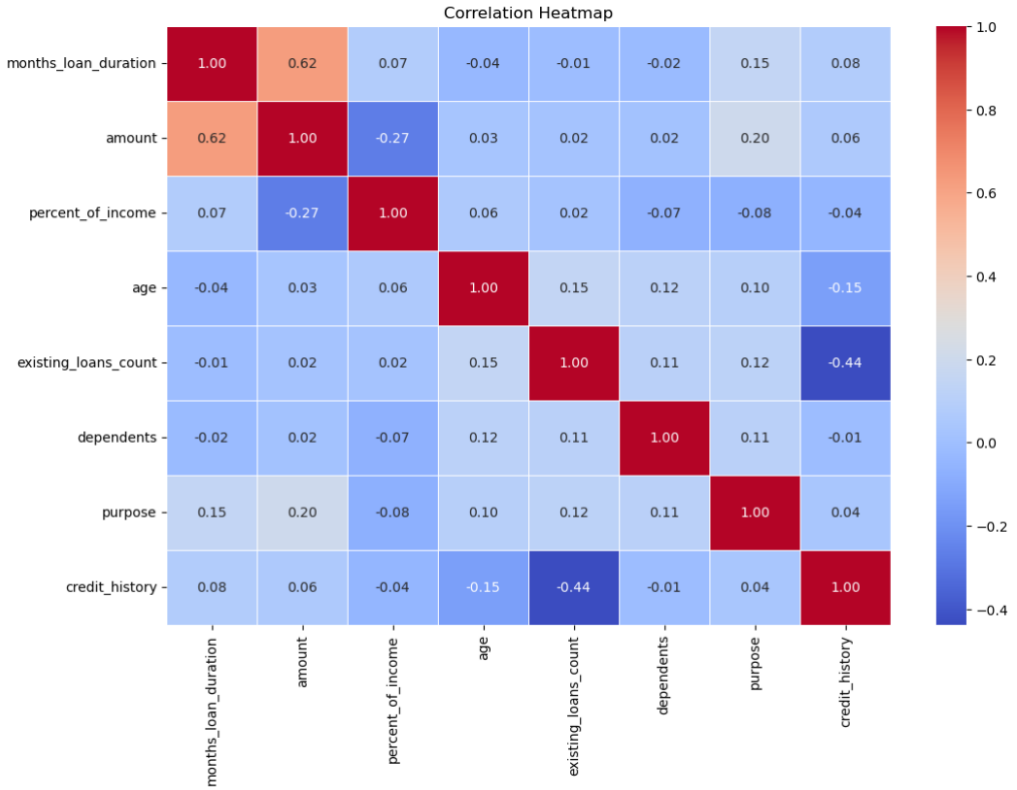
existing_loans_count and purpose (0.121013): There is a weak positive correlation (0.12) between the count of existing loans and the purpose. This suggests that, on average, as the count of existing loans increases, the purpose variable also tends to increase. slightly.

existing_loans_count and credit_history (-0.437066): There is a moderate negative correlation (-0.44) between the count of existing. loans and credit history. This suggests that, on average, as the count of existing loans increases, the credit history tends to decrease.

dependents and purpose (0.109593): There is a weak positive correlation (0.11) between the number of dependents and the purpose. This suggests that, on average, as the number of dependents increases, the purpose variable also tends to increase slightly.

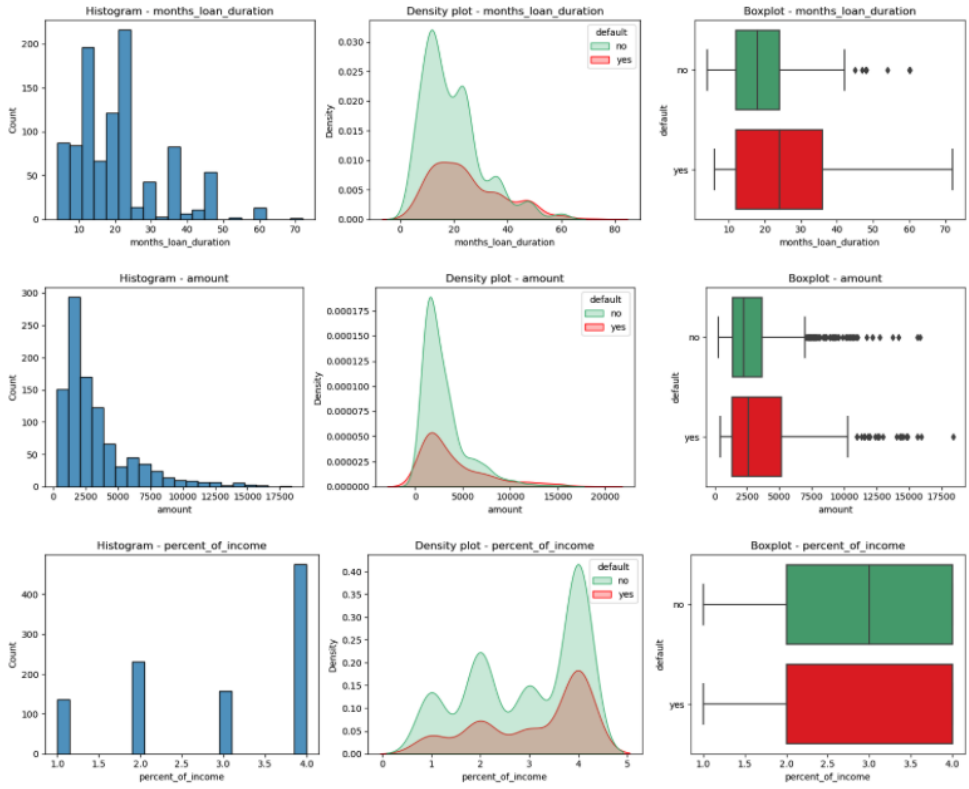
dependents and credit_history (-0.011550): There is a very weak negative correlation (-0.01) between the number of dependents. and credit history. The correlation is very close to zero, indicating little to no linear relationship.

purpose and credit_history (0.037127): There is a very weak positive correlation (0.04) between purpose and credit history. The correlation is very close to zero, indicating little to no linear relationship.

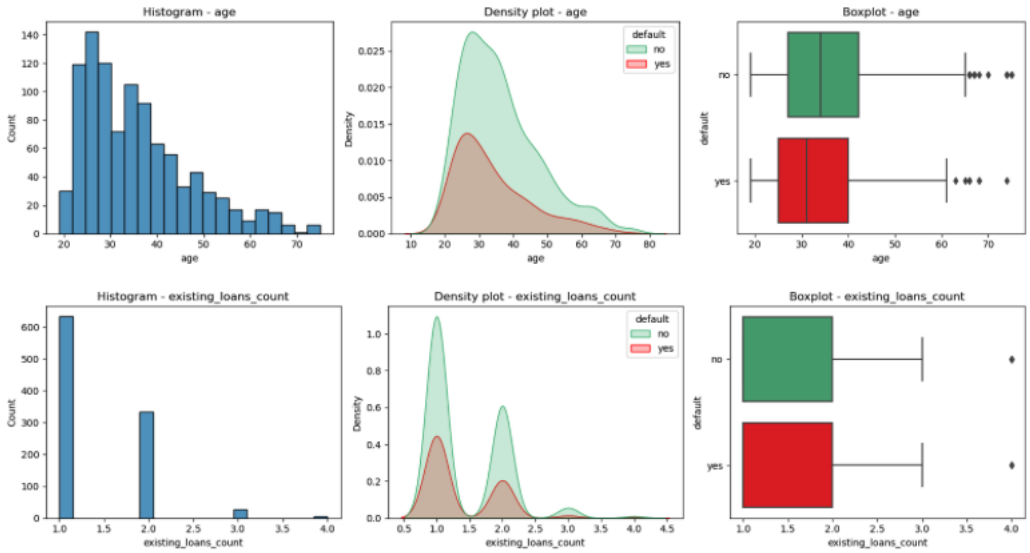


Correlation- Heat Map

Histogram, Density plot (hue='default') & Boxplots (hue='default') on some numerical columns

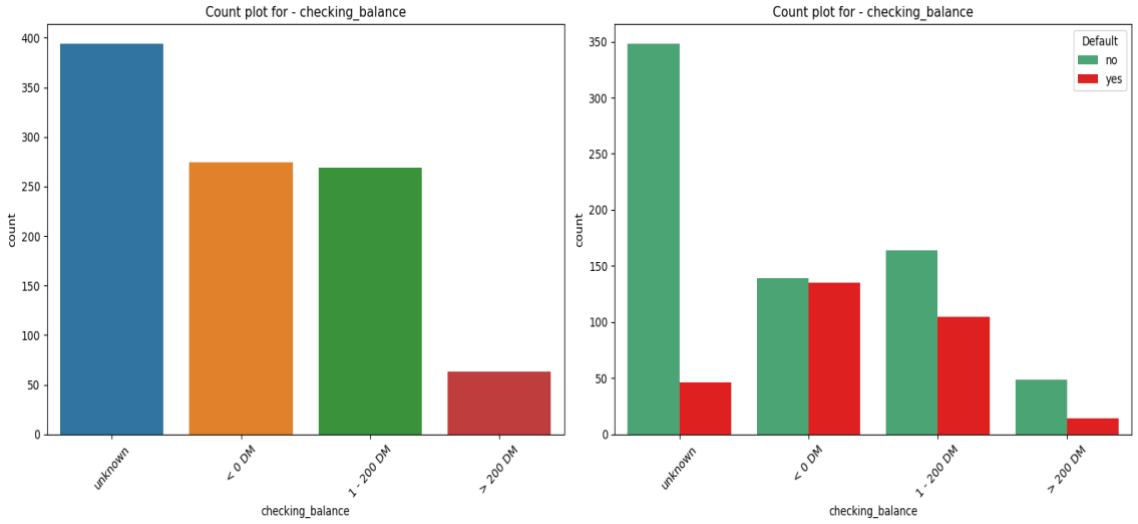


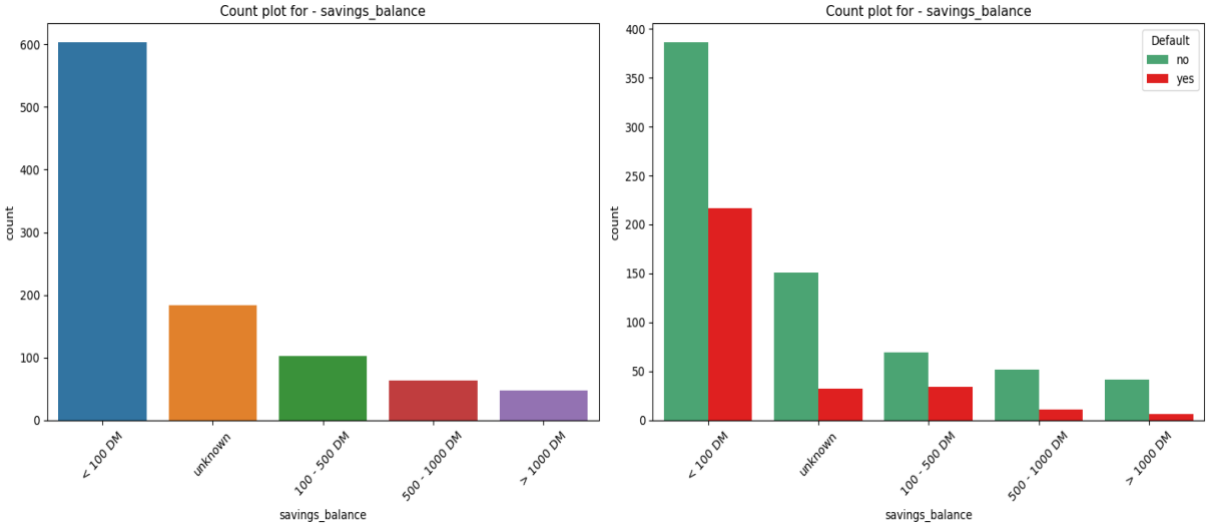
Features like Loan duration, Loan Amount and Income are vital and must be analyses well. I initially added credit history here but, I found no patterns as the count was **skewed** and I observed the '**good'** credit history were the most defaulters. On the other hand, looking at the counts (which I have addressed later on) I found most number of loan applicants where under 'good' credit history. Hindsight, I could have created a new feature where I took ratio of the count with each attributes. This method would have helped me explore **'Data Engineering'**.



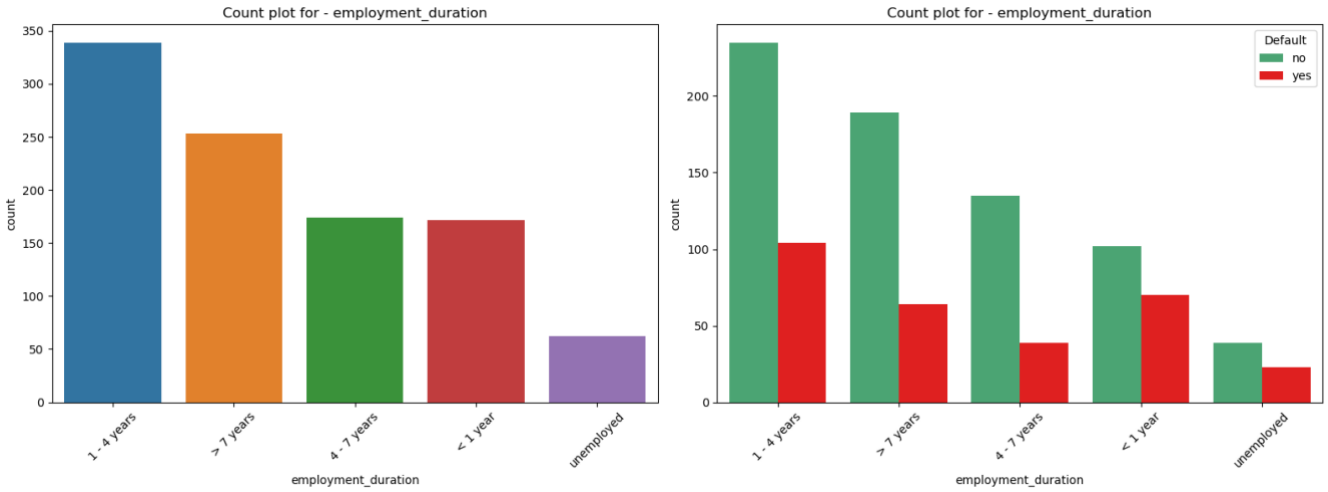
Age and Existing loan are some important feature too. It directly gives us the idea about how the certain participants would repay based on their age and the number of loan they have already taken. This directly shows the credibility.

For **categorical features**, I have performed the below **count plots** to analyse the distribution pattern of customers across different categories. On comparison without default allowed me to capture some ideas and patterns.

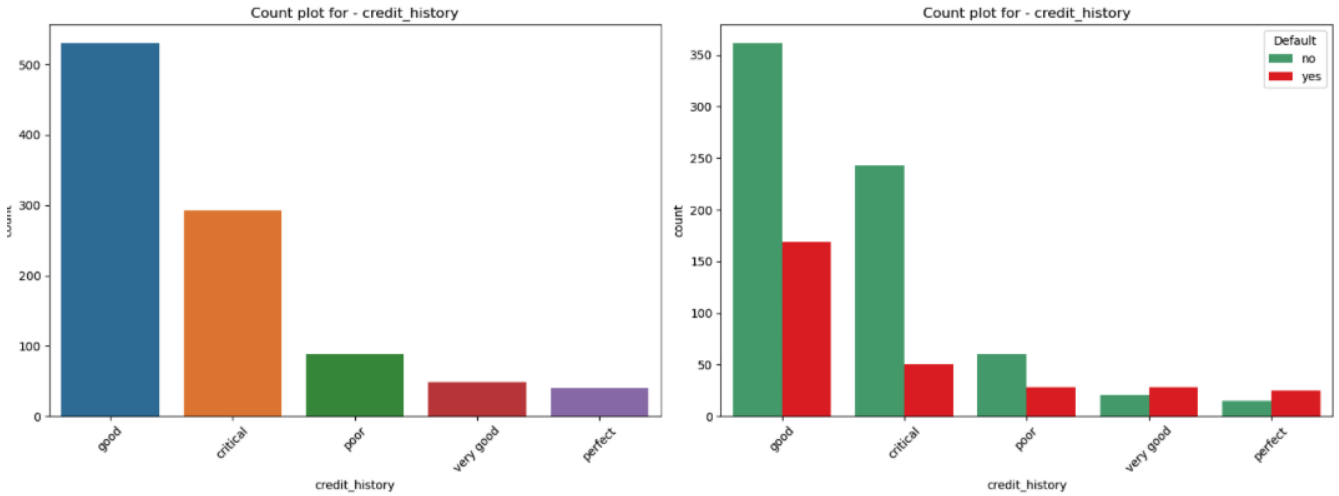




For categorical features, I used count plot. Savings and checking are the indicators of **financial health**. More the amount in saving or checking, its better as the participant is unlikely to default. Even here, the count plot is right skewed which means most participants fall in <100DM which is why a ratio/average between total count and attribute count would have given more comprehensive insight. Nonetheless, It is clear that, more the amount in savings and checking account unlikely the chances of default.

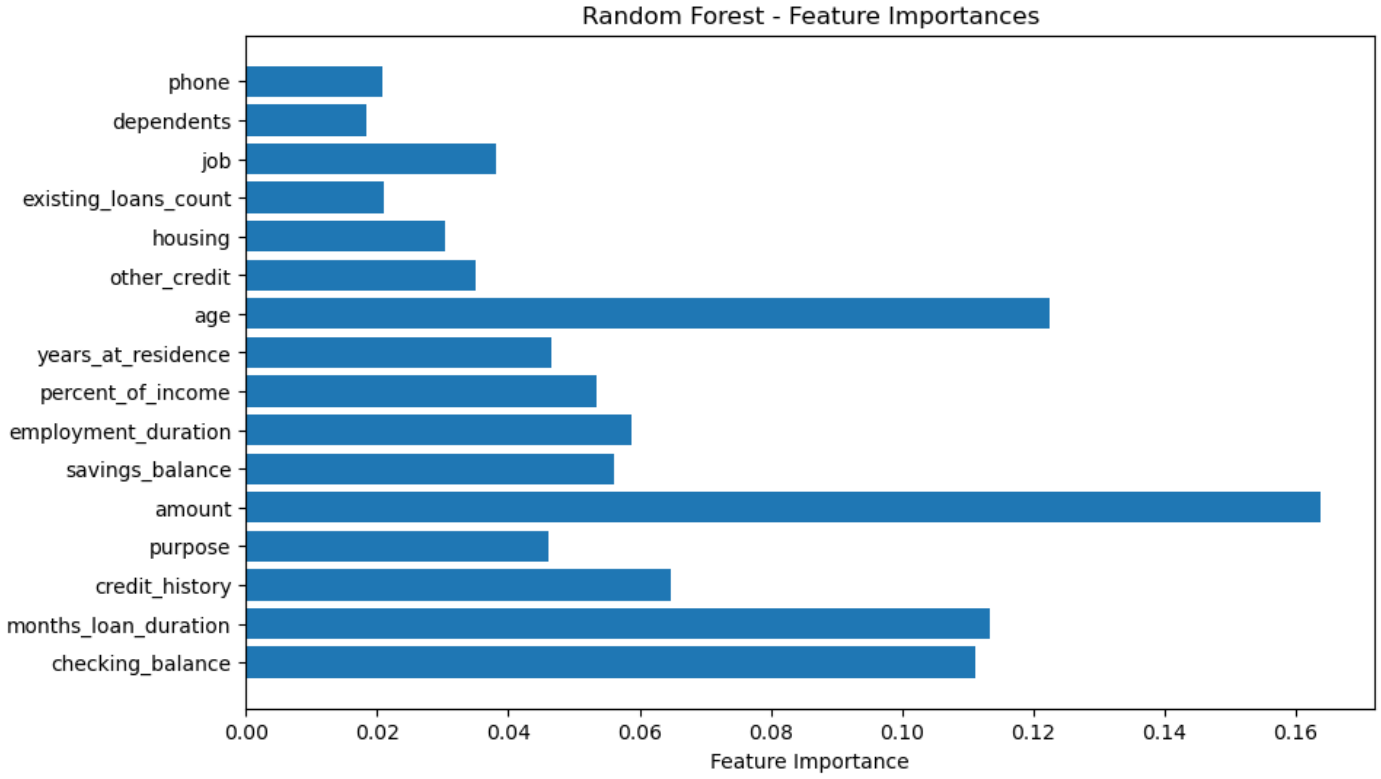


Point to note here is that, longer a participant has worked, it is unlikely that they might default. It represent a sense of responsibility and accountability. This shows the ableness of the participant. Although, the graphically representation here is arbitrary, more analysis needs to done in effect with 'duration of month' and 'saving balance'



According to my research and personal understanding, Credit history is the most important factor which decides the likelihood of 'default'. Here, regardless of the imbalance in count, point to worth noting is that answering the question "*Are there any features that are considered important but the data doesn't agree*?" Yes, from the above plots, it is evident that '**Good'** and '**Very Good'** credit history attributes have the most defaulters. That means, we need to focus more on other features rather than credit history. "*How can I prove it?* "

Answer: The features have importance against the predictor. I ran a feature importance function after I fit a **Random forest**. [*feature_importances = RF.feature_importances_*] , clearly credit history has not fared well.



Data Pre-Processing:

Initially, I began with **one hot-encoding** but the dataset got too complicated for me to handle. I resorted to manually mapping each features. Then I spilt the the data into **Test-Train sets at 80:20** with **stratification**. [*X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=random_seed, stratify=y)*] To keep the results consistent I set the **Random Seed to 42.** Followed by that, to ensure consistent scaling, I standardised the data using [*scaler = StandardScaler()*].

Models Training: 'ProcessedBankData'

I used all the relevant models to see which fit better to our needs. My primary goal was to not to **underfit/overfit** the data and have a high performing models. I applied ' **Logistic Regression', 'K-Nearest Neighbours', 'Quadratic Discriminant Analysis', 'Random Forest', 'Gradient Boosting' , 'AdaBoost', 'Linear Discriminant Analysis'.**

These models gave me all the necessary information and credentials to make a decision. I printed **confusion matrix** of all the models and **classification report** which contains details such as **accuracy**, **f1** score, **precision** and **recall**. With all these details, I wanted to make sure there is no issue of fitting.

Model fitting and Evaluations:

I wanted to use '**Hyperparameter tuning'** which I learnt about. I made a decision to keep it simple. Instead, I used cross validation. **Cross-validation** is a statistical technique used in machine learning to assess how well a predictive model will generalize to an independent dataset. The primary goal of cross-validation is to provide a more robust and unbiased estimate of a model's performance compared to a single train-test split

Throughout my program, I performed CV for all the models. Here is an example of my model (Random Forest)

Model 1: RandomForestClassifier

```
RF = RandomForestClassifier(random_state=random_seed)
RF.fit(X_train_scaled, y_train)
```

```
RandomForestClassifier(random_state=42)
```

```
RFPredYtest = RF.predict(X_test_scaled)
print("Classification Report of RandomForest (test) :\n", classification_report(y_test, RFPr
print("Confusion Matrix  of RandomForest (test) :\n", confusion_matrix(y_test, RFPredYtest))
```

```
Classification Report of RandomForest (test) :
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       140
           1       0.67      0.52      0.58        60

    accuracy                           0.78       200
   macro avg       0.74      0.70      0.72       200
weighted avg       0.77      0.78      0.77       200

Confusion Matrix  of RandomForest (test) :
 [[125  15]
 [ 29  31]]
```

```
RFPredYtrain = RF.predict(X_train_scaled)
print("Classification Report of RandomForest (train) :\n", classification_report(y_train, RFI
print("Confusion Matrix  of RandomForest (train) :\n", confusion_matrix(y_train, RFPredYtrai
```

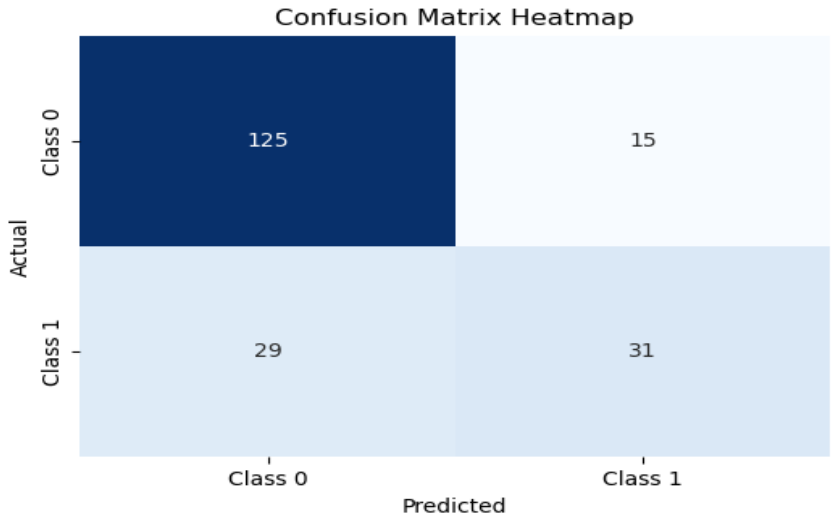
```
Classification Report of RandomForest (train) :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       560
           1       1.00      1.00      1.00       240

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800

Confusion Matrix  of RandomForest (train) :
 [[560   0]
 [  0 240]]
```

Cross-validation results: [0.77 0.76 0.715 0.76 0.79]
Mean accuracy: 0.759
Standard deviation: 0.02457641145488904

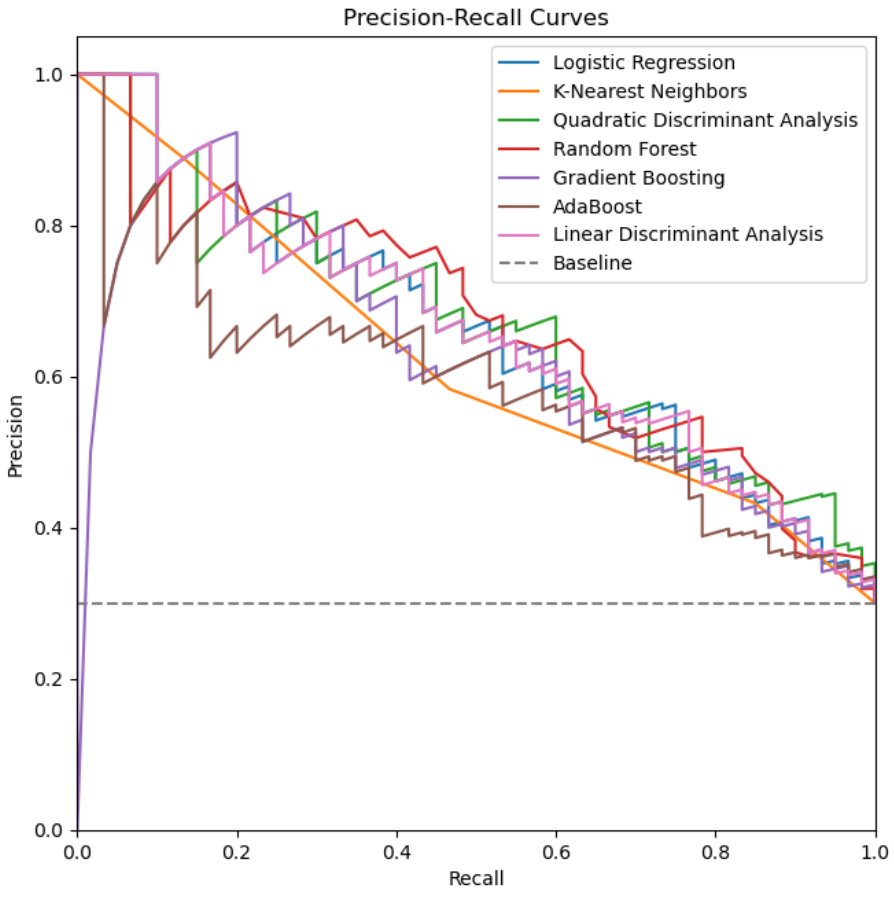


Random Forest Classifier - Confusion Matrix (Test set)

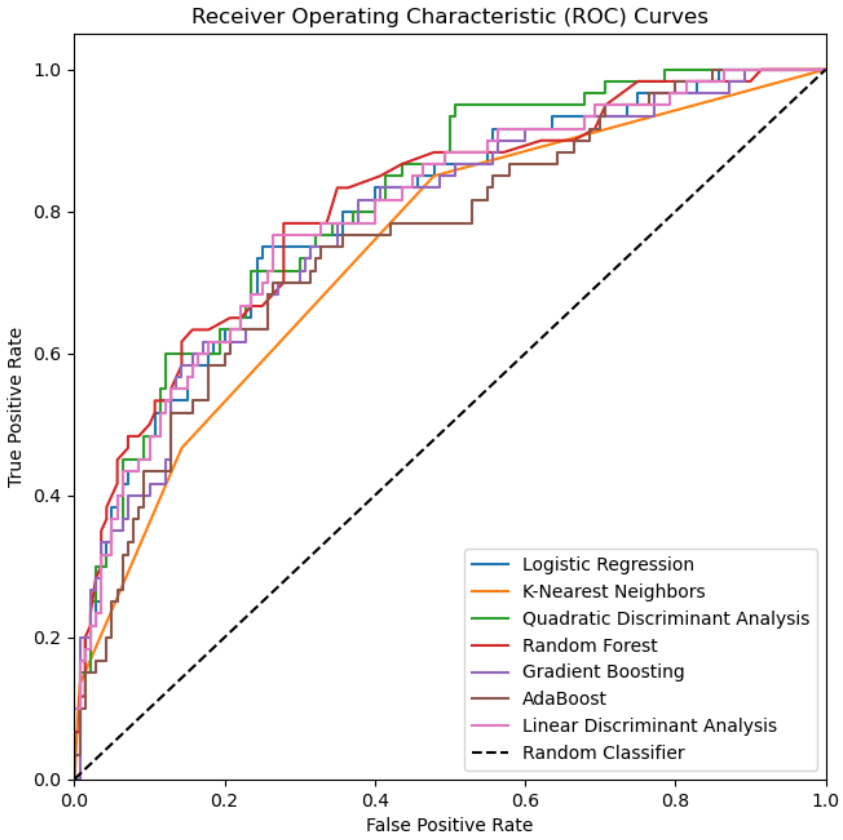
After deriving at all the models and I noticed that there is a pressing need to **collate** all the data and bring in form where I can compare them with their **Precision-Recall Area Under the Curve and ROC curve**.

Model Comparison and Final Selection:

Previously we have established that the data set is **imbalanced**, PR AUC is particularly useful when dealing with imbalanced datasets, where one class significantly outnumbers the other. It provides insights into a model's ability to correctly identify positive instances (high precision) while accounting for the imbalanced nature of the data. **Precision-Recall curves and PR AUC** focus on the performance of a model conce therning positive class. This is important in scenarios where the **positive class** is of greater interest, such as identifying cases of default. The sensitivity to false positives and false negatives, providing a more comprehensive evaluation of a model's performance.



Precision-Recall for all the fitted model.



Generally, the model with the best AUC-PR is considered as good model.

Results

Upon fitting about 7 models I arrived at the point of discussing what is next move.

1. Logistic Regression
2. K-Nearest Neighbours
3. Quadratic Discriminant Analysis
4. Random Forest
5. Gradient Boosting
6. AdaBoost
7. Linear Discriminant Analysis

```
AUC-PR Values (Area Under the Curve in a Precision-Recall plot):
                                Model   AUC-PR
                        Random Forest 0.671070
Quadratic Discriminant Analysis 0.661009
                  Logistic Regression 0.660227
     Linear Discriminant Analysis 0.658843
                  K-Nearest Neighbors 0.620856
                    Gradient Boosting 0.616956
                             AdaBoost 0.584090
```

Model 1: Logistic Regression

A statistical model that predicts the probability of binary outcomes, commonly used for classification tasks.

Classification Report of LogisticRegression (train) :

	precision	recall	f1-score	support
0	0.79	0.89	0.84	560
1	0.64	0.46	0.53	240
accuracy			0.76	800
macro avg	0.71	0.67	0.69	800
weighted avg	0.75	0.76	0.75	800

Confusion Matrix of LogisticRegression (train) :

```
[[497  63]
 [130 110]]
```

Classification Report of LogisticRegression (test) :

	precision	recall	f1-score	support
0	0.81	0.89	0.85	140
1	0.67	0.52	0.58	60
accuracy			0.78	200
macro avg	0.74	0.70	0.72	200
weighted avg	0.77	0.78	0.77	200

Confusion Matrix of LogisticRegression (test) :

```
[[125  15]
 [ 29  31]]
```

```
Cross-validation results: [0.77  0.76  0.705 0.755 0.795]
Mean accuracy: 0.7569999999999999
Standard deviation: 0.02942787793912435
```

Model 2: K-Nearest Neighbours

An algorithm that classifies data points based on the majority class of their k-nearest neighbors in the feature space

Classification Report of KNeighborsClassifier (test):

	precision	recall	f1-score	support
0	0.79	0.86	0.82	140
1	0.58	0.47	0.52	60
accuracy			0.74	200
macro avg	0.69	0.66	0.67	200
weighted avg	0.73	0.74	0.73	200

Confusion Matrix of KNeighborsClassifier (test):

```
[[120  20]
 [ 32  28]]
```

Classification Report of KNeighborsClassifier (train) :

	precision	recall	f1-score	support
0	0.86	0.94	0.90	560
1	0.83	0.64	0.72	240
accuracy			0.85	800
macro avg	0.84	0.79	0.81	800
weighted avg	0.85	0.85	0.85	800

Confusion Matrix of KNeighborsClassifier (train) :

```
[[528  32]
 [ 86 154]]
```

```
Cross-validation results: [0.625 0.625 0.615 0.605 0.625]
Mean accuracy: 0.619
Standard deviation: 0.008000000000000007
```

Model 3: Quadratic Discriminant Analysis

A classification algorithm that models the distribution of each class with a quadratic decision boundary.

```
Classification Report of QuadraticDiscriminantAnalysis (train):
              precision    recall  f1-score   support

           0       0.82      0.88      0.85       560
           1       0.65      0.55      0.59       240

    accuracy                           0.78       800
   macro avg       0.73      0.71      0.72       800
weighted avg       0.77      0.78      0.77       800

Confusion Matrix  of QuadraticDiscriminantAnalysis (train):
 [[490  70]
 [109 131]]
```

```
Classification Report of QuadraticDiscriminantAnalysis(test) :
              precision    recall  f1-score   support

           0       0.83      0.86      0.85       140
           1       0.65      0.60      0.63        60

    accuracy                           0.79       200
   macro avg       0.74      0.73      0.74       200
weighted avg       0.78      0.79      0.78       200

Confusion Matrix  of QuadraticDiscriminantAnalysis (test):
 [[121  19]
 [ 24  36]]
```

```
Cross-validation results: [0.78  0.745 0.69  0.755 0.76 ]
Mean accuracy: 0.7459999999999999
Standard deviation: 0.030232432915661973
```

Model 4: Random Forest

An ensemble learning method that constructs a multitude of decision trees and merges their predictions to improve accuracy and control overfitting

```
Classification Report of RandomForest (test) :
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       140
           1       0.67      0.52      0.58        60

    accuracy                           0.78       200
   macro avg       0.74      0.70      0.72       200
weighted avg       0.77      0.78      0.77       200

Confusion Matrix  of RandomForest (test) :
 [[125  15]
 [ 29  31]]
```

```
Classification Report of RandomForest (train) :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       560
           1       1.00      1.00      1.00       240

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800

Confusion Matrix  of RandomForest (train) :
 [[560   0]
 [  0 240]]
```

```
Cross-validation results: [0.77  0.76  0.715 0.76  0.79 ]
Mean accuracy: 0.759
Standard deviation: 0.02457641145488904
```

Model 5: Gradient Boosting

A machine learning technique that builds a predictive model by combining the outputs of multiple weak learners, typically decision trees, to create a strong predictive model.

Classification Report of GradientBoostingClassifier (test):

	precision	recall	f1-score	support
0	0.81	0.87	0.84	140
1	0.64	0.53	0.58	60
accuracy			0.77	200
macro avg	0.73	0.70	0.71	200
weighted avg	0.76	0.77	0.76	200

Confusion Matrix of GradientBoostingClassifier (test) :

```
[[122  18]
 [ 28  32]]
```

Classification Report of GradientBoostingClassifier (train):

	precision	recall	f1-score	support
0	0.89	0.96	0.93	560
1	0.90	0.72	0.80	240
accuracy			0.89	800
macro avg	0.89	0.84	0.86	800
weighted avg	0.89	0.89	0.89	800

Confusion Matrix of GradientBoostingClassifier (train):

```
[[540  20]
 [ 67 173]]
```

```
Cross-validation results: [0.77678571 0.72072072 0.74774775 0.79279279 0.73873874 0.7117117
1
 0.81081081 0.74774775 0.79279279]
Mean accuracy: 0.7599831974831974
Standard deviation: 0.03272974835112599
```

Model 6: AdaBoost

An ensemble learning algorithm that iteratively combines weak classifiers to build a strong model, giving more weight to misclassified instances in each iteration.

Classification Report of AdaBoostClassifier (test) :

	precision	recall	f1-score	support
0	0.81	0.84	0.82	140
1	0.58	0.53	0.56	60
accuracy			0.74	200
macro avg	0.69	0.68	0.69	200
weighted avg	0.74	0.74	0.74	200

Confusion Matrix of AdaBoostClassifier (test) :

```
[[117  23]
 [ 28  32]]
```

Classification Report of AdaBoostClassifier (train) :

	precision	recall	f1-score	support
0	0.83	0.89	0.86	560
1	0.70	0.56	0.62	240
accuracy			0.80	800
macro avg	0.76	0.73	0.74	800
weighted avg	0.79	0.80	0.79	800

Confusion Matrix of AdaBoostClassifier (train) :

```
[[501  59]
 [105 135]]
```

```
Cross-validation results: [0.79464286 0.7027027  0.72072072 0.79279279 0.67567568 0.6756756
8
 0.78378378 0.72072072 0.71171171]
Mean accuracy: 0.7309362934362934
Standard deviation: 0.04492032180372101
```

Model 7: Linear Discriminant Analysis

A statistical technique for classification and dimensionality reduction, seeking to find the linear combinations of features that best differentiate between classes.

```
Classification Report of LinearDiscriminantAnalysis (test) :
              precision    recall  f1-score   support

           0       0.81      0.88      0.84       140
           1       0.65      0.52      0.57        60

    accuracy                           0.77       200
   macro avg       0.73      0.70      0.71       200
weighted avg       0.76      0.77      0.76       200

Confusion Matrix  of LinearDiscriminantAnalysis (test) :
 [[123  17]
 [ 29  31]]
```

```
Classification Report of LinearDiscriminantAnalysis (train) :
              precision    recall  f1-score   support

           0       0.80      0.89      0.84       560
           1       0.65      0.47      0.54       240

    accuracy                           0.76       800
   macro avg       0.72      0.68      0.69       800
weighted avg       0.75      0.76      0.75       800

Confusion Matrix  of LinearDiscriminantAnalysis (train) :
 [[498  62]
 [127 113]]
```

```
Cross-validation results: [0.77678571 0.72972973 0.72072072 0.8018018  0.72072072 0.6576576
6
 0.8018018  0.78378378 0.71171171]
Mean accuracy: 0.7449681824681825
Standard deviation: 0.046087280293524976
```

Discussion

General Thoughts (Future Directions):

Something I could have done differently is data pre-processing. I could have used **one-hot-encoding** to sort out my data. It is a popular approach to processing data before fitting ML models. I could have used some technique like **Data Engineering** to attend the skewed issue. Most importantly, the occurrence of 'unknown' in the dataset, this uncertainty if solved would have increased the efficiency on the models. Hindsight, I could have implemented **Hyper-Tuning** and established a different **Threshold** to access the Recall.

Least performing models and its observation:

AdaBoost , **Gradient boosting** and **k-nearest neighbour** proved to be unfit for my approach, also thought they some merits with low false positives and high recall, as whole unit they did not perform as well as the rest of the models.

With clear imbalance in the dataset it was right for me to choose AUC-PR instead of AUC-ROC. The classification reports and confusion matrices for the AdaBoostClassifier on the test and training datasets provide insights into its performance. In terms of precision, recall, and F1-score, it is evident that AdaBoostClassifier faces challenges, especially with the positive class (class 1, indicating defaults)

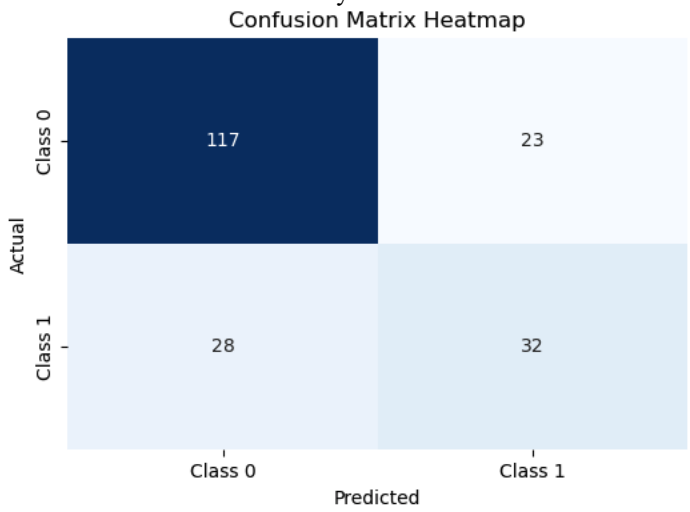
```
Classification Report of AdaBoostClassifier (test) :
              precision    recall  f1-score   support

           0       0.81      0.84      0.82       140
           1       0.58      0.53      0.56        60

    accuracy                           0.74       200
   macro avg       0.69      0.68      0.69       200
weighted avg       0.74      0.74      0.74       200

Confusion Matrix  of AdaBoostClassifier (test) :
 [[117  23]
 [ 28  32]]
```

On the test set, the precision for class 1 is relatively low (0.58), suggesting that when the model predicts a default, it is correct around 58% of the time. The recall for class 1 is also moderate (0.53), indicating that the model captures approximately 53% of the
actual defaults. These metrics collectively contribute to a lower F1-score for class 1 (0.56), reflecting the difficulty the model faces in achieving both high precision and recall simultaneously.



The confusion matrix on the test set further highlights the issue, with a noticeable number of false positives (28) and false negatives (23). This implies that AdaBoostClassifier tends to misclassify non-default cases as defaults (Type I errors) and fails to identify a significant portion of actual defaults (Type II errors).

```
Classification Report of AdaBoostClassifier (train) :
              precision    recall  f1-score   support

           0       0.83      0.89      0.86       560
           1       0.70      0.56      0.62       240

    accuracy                           0.80       800
   macro avg       0.76      0.73      0.74       800
weighted avg       0.79      0.80      0.79       800

Confusion Matrix  of AdaBoostClassifier (train) :
 [[501  59]
 [105 135]]
```

Examining the training set metrics, similar challenges are observed. While the precision for class 1 has improved compared to the test set, the recall is still relatively low. The confusion matrix for the training set indicates a substantial number of false negatives (105), reinforcing the model's struggle to identify instances of defaults during the training phase.

Best performing result and its observation:

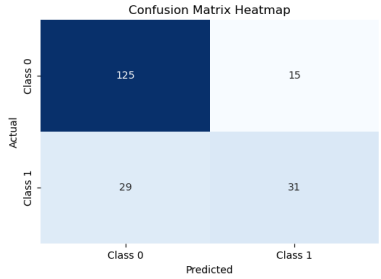
On the test set, the **RandomForest** model demonstrates a commendable balance between precision and recall for both classes. The precision for class 1 (default) is 0.67, indicating that when the model predicts a default, it is correct around 67% of the time. The recall for class 1 is 0.52, suggesting that the model captures approximately 52% of the actual defaults. These metrics contribute to an overall F1-score for class 1 of 0.58. While there is room for improvement, the RandomForest model achieves a reasonable trade-off between precision and recall, considering the inherent tension between these two metrics.

```
Classification Report of RandomForest (test) :
              precision    recall  f1-score   support

           0       0.81      0.89      0.85       140
           1       0.67      0.52      0.58        60

    accuracy                           0.78       200
   macro avg       0.74      0.70      0.72       200
weighted avg       0.77      0.78      0.77       200

Confusion Matrix  of RandomForest (test) :
 [[125  15]
 [ 29  31]]
```



The confusion matrix on the test set reveals that the model has 29 false positives and 15 false negatives. This implies that it misclassifies some non-default cases as defaults (Type I errors) and fails to identify a portion of actual defaults (Type II errors). However, these errors are relatively balanced, indicating a robust performance.

```
Classification Report of RandomForest (train) :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       560
           1       1.00      1.00      1.00       240

    accuracy                           1.00       800
   macro avg       1.00      1.00      1.00       800
weighted avg       1.00      1.00      1.00       800

Confusion Matrix  of RandomForest (train) :
 [[560   0]
 [  0 240]]
```

Looking at the training set, the RandomForest model achieves perfect precision, recall, and F1-score for both classes. The confusion matrix for the training set shows no misclassifications, highlighting the model's ability to perfectly fit the training data. While this may raise concerns about overfitting, the cross-validation results provide assurance.

```
Cross-validation results: [0.77  0.76  0.715 0.76  0.79 ]
Mean accuracy: 0.759
Standard deviation: 0.02457641145488904
```

The cross-validation results exhibit consistent and high accuracy across folds, with a mean accuracy of 0.759 and a low standard deviation (0.0246). This suggests that the model generalizes well to unseen data, reinforcing its reliability beyond the training set.

Finally, the Random Forest model emerges as a **robust performer**, demonstrating a strong ability to classify instances of both classes. It strikes a good balance between precision and recall, with a reasonable F1-score, and exhibits generalization capability, as evidenced by the cross-validation results.

Limitations:

It is essential to acknowledge the limitations inherent in this study, including the relatively small dataset, class imbalance, and the identified need for more advanced feature engineering. Despite these constraints, the results underscore the significance of employing sophisticated machine learning techniques in the domain of financial risk assessment.

Citation:

1. **Libraries**: Scikit-Learn, sklearn- metrics, seaborn, pandas, numpy, statsmodels, matplotlib, sklearn – neighbors, models
2. **Websites (Domain):** https://www.hdfcbank.com/personal/resources/learning-centre/borrow/7-factors-that-determine-whether-your-loan-gets-sanctioned
   : https://www.icicibank.com/blogs/personal-loan/eligibility-for-a-personal-loan
3. **Website (Models):** https://www.databricks.com/glossary/machine-learning-models
   : https://www.statsmodels.org/stable/index.html
   : https://www.youtube.com/channel/UCfzlCWGWYyIQ0aLC5w48gBQ (sentdex)

Conclusion

In this undertaking, I set out to create a predictive model aimed at forecasting loan defaults, utilizing historical data sourced from a German bank. The process involved a methodical approach that spanned data exploration, visualization, and the implementation of diverse machine learning algorithms. The primary objective was to construct a dependable and precise model capable of aiding the bank in pinpointing potential loan defaulters and managing financial risks effectively.

A thorough examination of the dataset revealed insightful patterns through rigorous analysis. **Exploratory Data Analysis (EDA)** illuminated the interrelationships among various features and their influence on the likelihood of loan defaults. Multiple machine learning models were explored, with each one meticulously fine-tuned and assessed, considering essential performance metrics and the area under the **Precision-Recall curve (AUC-PR).** This approach prioritized models that adeptly balanced precision and recall.

Among the various models, **Random Forest**, recognized as a potent ensemble technique, emerged as the most promising for predicting loan defaults. It consistently exhibited robust performance in terms of AUC-PR and recall, showcasing its proficiency in identifying potential default cases while minimizing false negatives, thereby enhancing the bank's capacity to accurately predict loan defaults.

Something I learnt while I was researching and fetching Domain knowledge was the term '**Mortgage**' and '**Collateral'**. This seemed to be bullet proof as in the event of default, the bank is liable to recover the money pending by selling the asset/collateral which would totally solve the issue.