LOAN DEFAULTER PREDICTION

JULY 18

DATA SCIENCE PROJECT 02

Shriram HANGARGEKAR

Table of Contents

1 Introduction

1.1 Problem Statement

Problem statement for this project is as below:

> ***"The loan default dataset has 8 variables and 850 records, each record being loan default status for each customer. Each Applicant was rated as "Defaulted" or "Not-Defaulted". New applicants for loan application can also be evaluated on these 8 predictor variables and classified as a default or non-default based on predictor variables."***

The loan default dataset mentioned in the problem statement has following attributes:

Table 1-i: Description of loan default dataset

Sr.	Attribute	Type	Description
1	age	Numerical	Age of the customer
2	ed	Categorical	Education level (converted to numerical)
3	employ	Numerical	Years of service
4	address	Numerical	Years of residence
5	income	Numerical	Income of the customer
6	debtinc	Numerical	Ratio of debt to income
7	creddebt	Numerical	Credit as %age of debts
8	othdebt	Numerical	Any other debts

1.2 Objective

The key objective of this project is to build a model that can predict whether a customer will default his/her loan. Since the objective is to predict Yes/No answer, a classification model is more appropriate to solve our problem. To achieve robust and fair prediction system, multiple models need to be created before finalizing one. Each model is tested and then is evaluated on the basis of different error metric. The report discusses aspects of project from initial data analysis through different data cleaning processes and feature engineering up to developing model for prediction, in addition to these performance parameters selected for evaluation of different model and finalizing on single model is also discussed.

1.3Work Flow

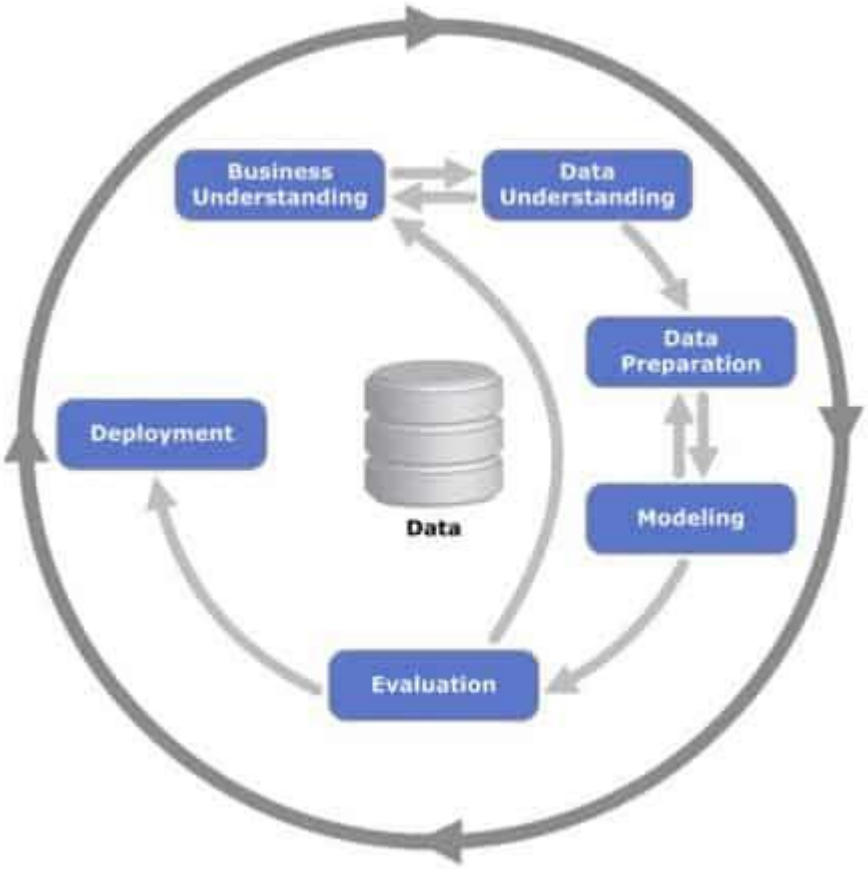


Figure 1-A: CRISP-DM method

Workflow during the project was arranged and prioritized by putting CRISP-DM framework at projects backbone. As the first step in this framework is to understand business, first chapter is aimed at understanding problem statement and its scope. In subsequent chapters, further steps viz. Data Understanding, Data Preparation, Modelling, and Evaluation are discussed. Data understanding covers data overview and basic exploratory analysis. Data preparation focuses on feature selection and data cleaning. After data has been prepared then project proceeds to develop the model, and this step comprises of selection of different algorithm and training those algorithm on train data. In the subsequent chapter, evaluation basis and criteria to select best performing model are discussed. After which final remarks give different observations from developing model.

2 Data Understanding

2.1 Data Overview

Data is record of 850 customers with 9 attributes. Some of those customers have value for their defaulter state, and some of them do not. Thus records with default state available are used as train data and rest as test data.

- Age (age)
  This attribute indicates age of the customer. For an employed person probability of being defaulter may reduce over age whereas the probability may increase for customer who has no fixed source of income.

- Education (ed)
  This attribute indicates level of education of the customer. Higher education level generally implies higher pay-grade. Thus higher pay may also indicate lower probability of defaulting the loan.

- Employment (employ)
  This, as discussed earlier, indicates that customer has fixed source of income and that usually can indicate higher chances of loan repayment. And thus lower probability of being defaulted.

- Address (address)
  This attribute indicates years of residence time of customer in years. Longer residence time can also give sufficient reasons to believe the customer will not default the loan.

- Income (income)
  This is self-reported gross income of customer. This is a strong indicator, because in general condition it can be said that higher income will have less chances of defaulting loan by customer.

- Debt-to-income Ratio (debtinc)
  Another attribute in the data represents ration of customer's debit to his/her income. This is another important attribute since it establishes connection between customer's income and total debt.

- Credit-to-debit ratio (creddebt)
  This attribute describes total debt customer owes to creditors as a percentage of total debt. Lesser the amount higher the confidence that customer will pay up his/her loan.

- Other debts (othdebt)
  This final attribute sums up all other debts on the customer. If liabilities other than loan are lower, then the risk that customer will default loan is also significantly lower.

- Default status (default)
  This is target variable, which is present only in historic data. This indicate whether the customer will default or not as binary values – 1 for default case and 0 for paid up loan case.

Thus, data consists of eight variables to train and develop predictive model. Feature selection is done in next chapter. After developing this basic understanding of data, project can be progressed to next step that is exploratory analysis.

2.2Initial Data Analysis

Target of initial data analysis is to understand underlying structure of given data. Additionally this analysis also helps in detecting outlier points in data and anomalies in values from data. This understanding helps to build the assumption required for model building and exploratory analysis.
In the initial data exploration, data is analyzed to get feel of data. In this process different statistical parameters are observed and inconsistencies such as potential outliers, invalid values present in the data are checked.

2.2.1 Data type of attributes

In preliminary analysis, data types of attributes were found to be appropriate for further procedure.

#	Column	Non-Null Count	Dtype
0	age	850 non-null	int64
1	ed	850 non-null	int64
2	employ	850 non-null	int64
3	address	850 non-null	int64
4	income	850 non-null	int64
5	debtinc	850 non-null	float64
6	creddebt	850 non-null	float64
7	othdebt	850 non-null	float64
8	default	700 non-null	float64

Figure 2-A: Data types of given data

2.2.2 Statistical parameters

Basic statistical parameters helps in understanding spread, maxima, minima, and central tendencies of the data.

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
count	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	700.000000
mean	35.029412	1.710588	8.565882	8.371765	46.675294	10.171647	1.576805	3.078789	0.261429
std	8.041432	0.927784	6.777884	6.895016	38.543054	6.719441	2.125840	3.398803	0.439727
min	20.000000	1.000000	0.000000	0.000000	13.000000	0.100000	0.011696	0.045584	0.000000
25%	29.000000	1.000000	3.000000	3.000000	24.000000	5.100000	0.382176	1.045942	0.000000
50%	34.000000	1.000000	7.000000	7.000000	35.000000	8.700000	0.885091	2.003243	0.000000
75%	41.000000	2.000000	13.000000	12.000000	55.750000	13.800000	1.898440	3.903001	1.000000
max	56.000000	5.000000	33.000000	34.000000	446.000000	41.300000	20.561310	35.197500	1.000000

Figure 2-B: Description of train data

This chart gives simple picture of data. Couple of observations can be made from the above data:

- The number of total observations in train data is 850. However, 'default' variable has 700 count, which are due to 150 test data. This means there are no missing values in data.
- The minimum values for 'employ' and 'address' are zero. This may mean that there are few newly joined employees and newly relocated residents. This is consistent with logic.

Thus, no illogical values are detected in preliminary overview.

2.2.3 Class distribution

As observed from below graph, training data is unbalanced and is biased towards paid up loans. For training this may affect model performance, because performance is highly dependent on data quality

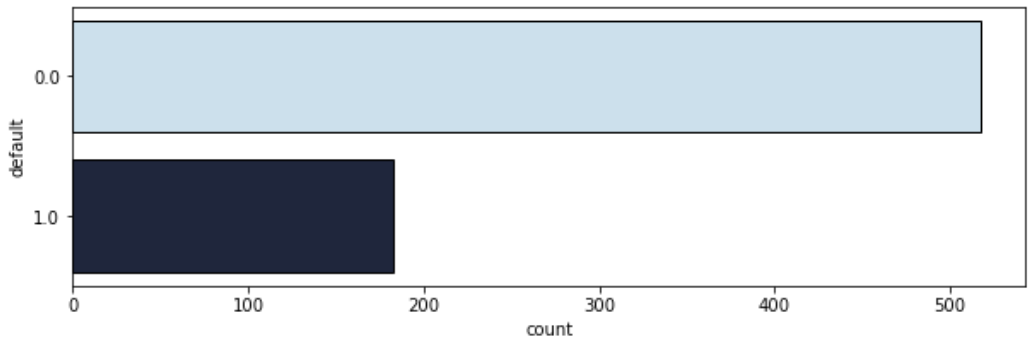


Figure 2-C: Class distribution in train data

2.2.4 Graphical Analysis

Graphical representation of data gives simplistic visual and intuitive picture of data. Two types of graphs are used viz. Univariate and Multivariate graphs. Univariate graphs such as box-plot, violin graph, and histograms give overall distribution of an individual variable. Whereas multivariate graphs such as scatter plot, grouped box-plot, and heat-map give overall picture of relationship between two attributes.

- **Univariate Graphs**

With help of the univariate graphs, probability for each attributes default status is checked. Lower age group, lower employment time, lower resident time, and lower education level have more probability of defaulting the loan.

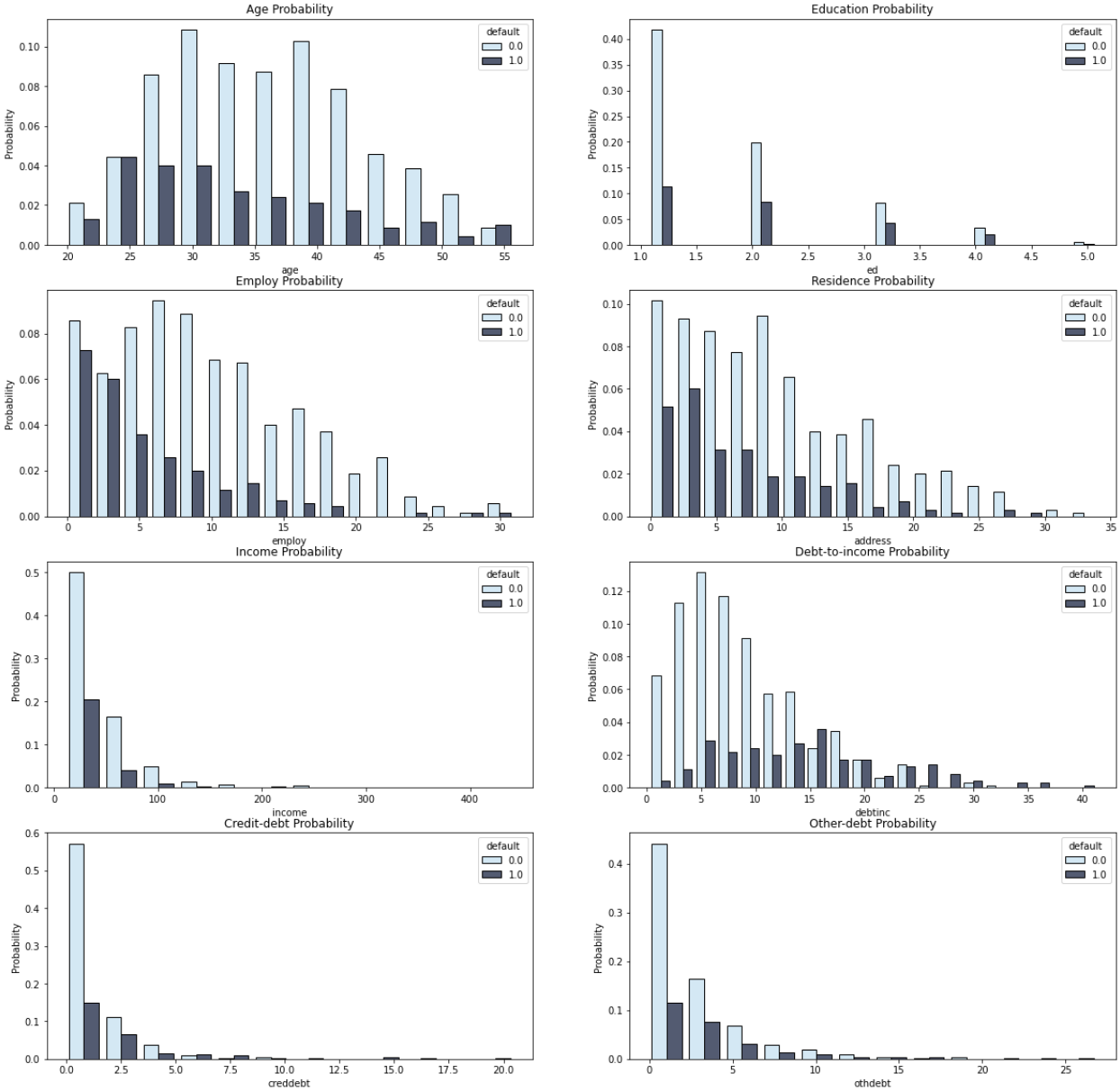


Figure 2-D: Univariate analysis of attributes

❖ Bivariate Graphs

Pairwise graphs can give better picture of relation between a pair of attributes. As observed from scatter plots, line of best fit for all pairs is almost horizontal. This indicates that attributes do not have linear correlation among them. This is good for regression model as it requires input variables to be independent of each other.

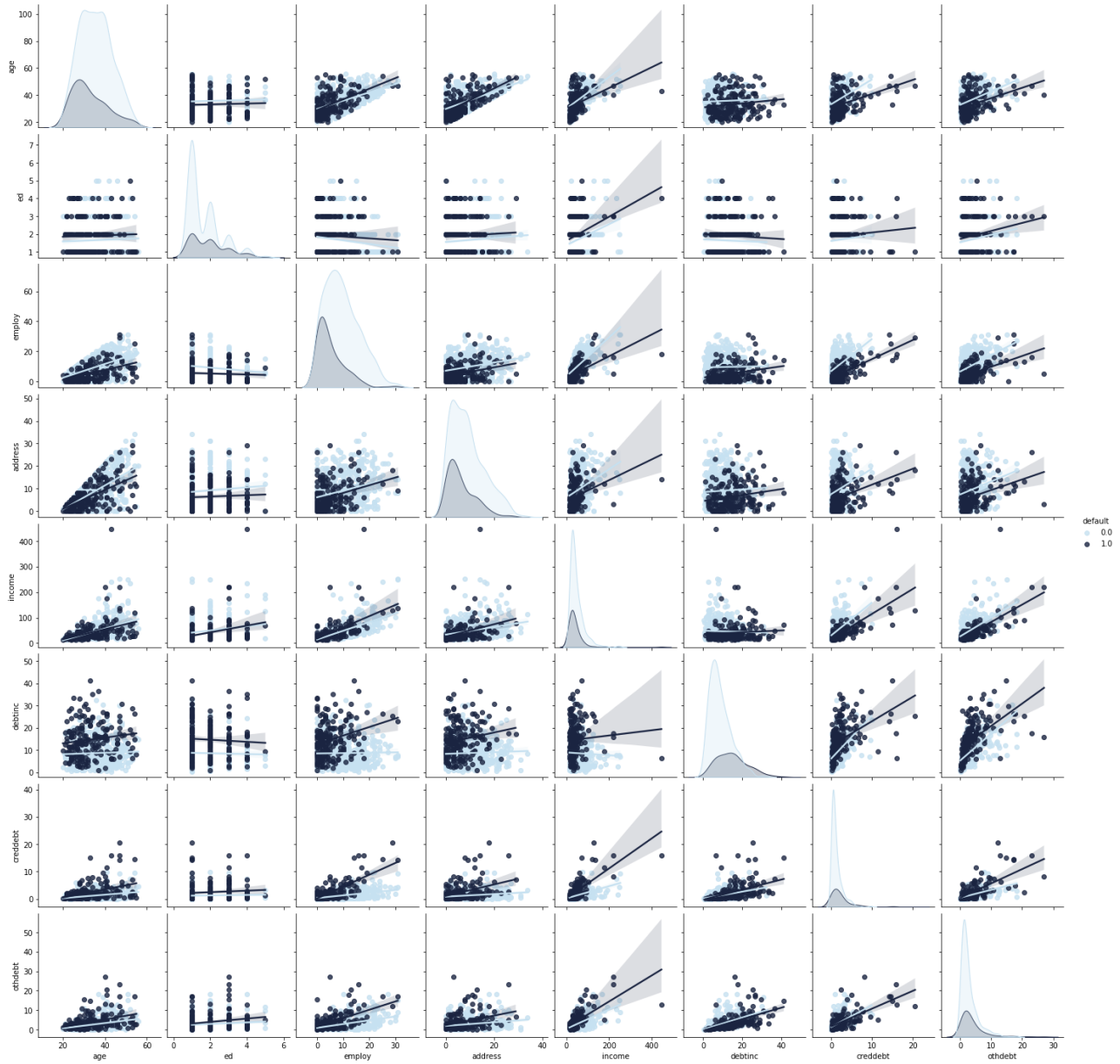


Figure 2-E: Bivariate analysis of attributes

3 Data Preparation

3.1 Feature Engineering

Before proceeding to cleaning the data it is important to understand attributes required for model building. Understanding these requirements helps in effective data preparation. In this process features are transformed on new features are created based on business domain knowledge.
For problem at hand, it is necessary to look at factors contributing to customer defaulting loan. Ideally, customer defaults loan because he/she is doesn't have money to pay up the loan. Such situation may occur if expenditure or debt on the customer exceeds his/her income. Another way available balance of the customer may exhaust depends on unexpected expenditure.

$$\boldsymbol{Probability\ of\ Defaulting\ loan} \propto [\boldsymbol{Available\ Funds}] + [\boldsymbol{Unexpected\ Expense}]$$

- Available Funds

Available funds can be estimated by attributes such as gross income (income) and debt (debtinc, cred debt, and othdebt). Since the factors can be processed directly in model. New feature is not created for the purpose of model and these five attributes are fed to model as they are.

$$\boldsymbol{Funds} \leftarrow \langle \boldsymbol{income}|\boldsymbol{debtinc}|\boldsymbol{creddebt}|\boldsymbol{othdebt} \rangle$$

- Unexpected Expenses

This is a subjective factor which cannot be estimated mathematically. However it would be sound hypothesis that years of residence (address), education level (ed) , customer's age (age), and years of service (employ) can be good indicators of financial strain put by unexpected expenses.

$$\boldsymbol{Unexpected\ Expense} \leftarrow \langle \boldsymbol{age}|\boldsymbol{employ}|\boldsymbol{ed}|\boldsymbol{address} \rangle$$

With hypothesis stated above, features present in dataset can be used as they are for developing model. No new features are derived from existing features. Type of dependence (direct or inverse) is estimated by model through coefficient.
Thus schematic of model can be represented as below:

$$\boldsymbol{Default} \leftarrow \langle \boldsymbol{income}|\boldsymbol{debtinc}|\boldsymbol{creddebt}|\boldsymbol{othdebt}|\boldsymbol{age}|\boldsymbol{employ}|\boldsymbol{ed}|\boldsymbol{address} \rangle$$

3.2 Data Cleaning

First, step in data preparation for modelling is to clean data. In this process data is analyzed for missing values and outliers. Missing values harm data analysis as erroneous data might be considered for analysis by program. Outlier values tend to deviate model outcome by either inflating or deflating coefficient during model training.

3.2.1 Missing Value Analysis

Upon analyzing, it was observed that data has no missing values. Thus no additional treatment is necessary.

	Missing Count	Missing Pcent
age	0	0.0
ed	0	0.0
employ	0	0.0
address	0	0.0
income	0	0.0
debtinc	0	0.0
creddebt	0	0.0
othdebt	0	0.0
default	0	0.0

Figure 3-A: Missing Value Analysis

3.2.2 Outlier Analysis

As observed from distribution graphs data is distributed unevenly but within the finite range. The outliers in the data are treated during feature transformation of the data

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
count	700.000000	657.000000	690.000000	686.000000	660.000000	686.000000	645.000000	652.000000	700.000000
mean	34.860000	1.566210	8.089855	7.881924	38.904545	9.813120	1.078174	2.366007	0.261429
std	7.997342	0.715007	6.218614	6.287217	20.172210	6.106888	0.948970	1.836622	0.439727
min	20.000000	1.000000	0.000000	0.000000	14.000000	0.400000	0.011696	0.045584	0.000000
25%	29.000000	1.000000	3.000000	3.000000	24.000000	5.000000	0.338575	0.994035	0.000000
50%	34.000000	1.000000	7.000000	7.000000	33.000000	8.500000	0.775656	1.819176	0.000000
75%	40.000000	2.000000	12.000000	12.000000	50.000000	13.600000	1.583604	3.230684	1.000000
max	56.000000	3.000000	25.000000	25.000000	101.000000	27.700000	4.160000	8.217600	1.000000

Figure 3-B: Data after removing statistical outliers

Thus data is sufficiently clean for modelling.

3.3 Feature Transformation

Every attribute has different scale. This affects our analysis. To overcome this problem data was normalized to unit scale. This also reduces computational cost of model as transformed features are smaller and quickly processed by computer.

- Min-Max Scaling

This method maps values based on min and max of the data. General formula to calculate transformed value can be given as below:

$$X_{transformed} = \frac{X_{actual} - X_{min}}{X_{max} - X_{min}}$$

Categorical feature 'ed' and continuous feature 'age', which has no outlier were transformed with normal scaling method using minimum and maximum of the data.

- Quintile Scaling

This is also known as percent-point function. It denotes percentile position of value in 0-1 scale.

Rest of the features were scaled by quintile transformation, which uses quintile percentage to assign value. This method is robust and can handle outliers. Thus this method was used for rest of the features.

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
0	0.583333	0.50	0.883838	0.747475	0.986600	0.535354	0.991314	0.830382	1.0
1	0.194444	0.00	0.651515	0.459596	0.434343	0.848485	0.641240	0.758655	0.0
2	0.555556	0.00	0.823232	0.797980	0.752525	0.275421	0.498053	0.535441	0.0
3	0.583333	0.00	0.823232	0.797980	0.961890	0.101010	0.837953	0.171692	0.0
4	0.111111	0.25	0.186869	0.000000	0.368687	0.848485	0.728979	0.680814	1.0

Figure 3-C: Transformed Data

3.4 Feature Selection

In this stage, all features are analyzed to check their importance and inter-correlation with other features. Correlation, VIF are some of the indicators of this.

3.4.1 Correlation Analysis

If the two or more features are highly correlated then they carry same information in data. That is redundant for model building and potentially inflating for result. Thus one of the highly correlated features should be kept in model. Heatmap is plotted to analyze correlation. As a thumb rule, features with correlation more than 0.65 are considered to be correlated. And those will be removed.

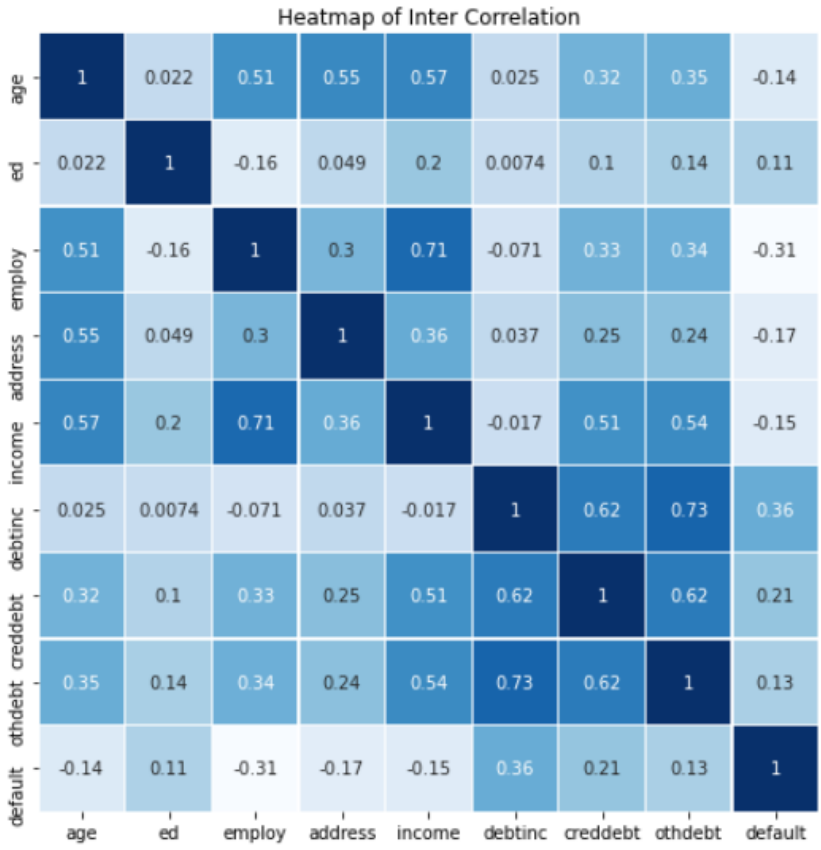


Figure 3-D: Heatmap of correlation of variables

As observed from heatmap, there is no correlation greater than threshold value of 0.66. Thus there is high correlation between some of the features.

3.4.2 Multi co-linearity Analysis

Multi co-linearity is checked as there is possibility that combination of multiple variables may have strong co-linearity which is as harmful as the correlation between two variables. This multi co-linearity is checked with help of variance inflation factor (VIF).

- If the VIF is equal to 1 then it means there is no co-linearity in variable. This is ideal result.
- If the VIF is between 1 and 5 then it indicates moderate co-linearity. This is acceptable condition.
- If the VIF is more than 5 then it indicates high co-linearity. This is not desirable outcome.
- However if more than one variable have VIF greater than 5 then only one with highest VIF is dropped.

	VIF	INPUTS
0	13.984946	Intercept
1	1.921603	age
2	1.280756	ed
3	2.531528	employ
4	1.451421	address
5	8.116490	income
6	10.777322	debtinc
7	4.172091	creddebt
8	9.049529	othdebt

Figure 3-E: VIF for dependent variables

	VIF	INPUTS
0	7.738393	Intercept
1	1.920827	age
2	1.274269	ed
3	2.525024	employ
4	1.446364	address
5	3.370105	income
6	1.764820	creddebt
7	1.840389	othdebt

Figure 3-F: VIF analysis after dropping 'debtinc'

As observed from table, feature with highest VIF is ‘debtinc’. This feature is dropped from our analysis.

4 Modeling

4.1 Pre-modeling Steps

Before building model, some steps are taken that help in building model. These steps include splitting data into train and test, understanding influence of hyper-parameter on model and defining evaluation metrics for data. These preparatory steps ease process of modeling and achieve better results.

4.1.1 Train and Validation data split

Before building model, train data is split into train set for training purpose and test set for validation purpose. This helps in evaluating performance of model. Since it is impossible to know true outcome of test_data, pseudo test set is created to validate performance of model.

- Whole data is divided in dependent variables (X) and target variable (Y)
- 30% of data is in test set and it has 210 observations
- 70% of data in in train data set and it has 490 observations
- X_test, Y_test are validation data set
- X_train, Y_train are training data set

4.1.2 Hyper-parameter optimization

Performance of model depends on the value of different parameters attributed to it. These parameters controls behavior of model. To get optimum performance out of model tuning of these parameters is necessary. Two methods used for hyper parameter optimization:

- GridSearchCV:
  In GridSearchCV, the algorithm creates a grid of all combinations of values of hyper-parameters within provided range. And every set of hyper-parameters in grid are used to train model and to score the test data. Since every possible combination is tried, this method at times can be inefficient, but always provides the best results.
- RandomizedSearchCV:
  In RandomizedSearchCV, the algorithm as name suggests tests random combinations of hyper-parameters within provided range. And several iterations are performed to get good results. This means that not every combination is tested. Thus algorithm compromises on the accuracy of the results but provides better results in less time compared to grid search algorithm.

4.1.3 Evaluation Metrics

There are multiple metrics such as accuracy, precision, recall, f1, fallout, miss rate and selectivity. These metrics are calculated with help of confusion matrix. Confusion matrix maps predicted labels against true labels. This can be represented as below:

Table 4-i: Confusion Matrix

		PREDICTED	
	LABELS	TRUE	FALSE
ACTUAL	TRUE	TP	FN
	FALSE	FP	TN

TP*(True Positives): Labels that are actually true and correctly predicted as true*
FP*(False Positives): Labels that are actually false but wrongly predicted as true*
FN*(False Negatives): Labels that are actually true but wrongly predicted as false*
TN*(True Negatives): Labels that are actually false and correctly predicted as false*

For purpose of the project following metrics are used:

- Accuracy : Accuracy is ratio of total correctly identified labels to total available labels. This is basic metric that gives overview of accuracy of prediction:

$$\boldsymbol{Accuracy} = \frac{\boldsymbol{TP} + \boldsymbol{TN}}{\boldsymbol{Total}}$$

- Recall : Recall is ratio of correctly predicted true positive to total actual positive. This metric specifically identifies performance related to finding positive values:

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FN}}$$

- Precision : Recall is ratio of correctly predicted true positive to total predicted positive. This metric specifically identifies accuracy of finding positive values:

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP} + \boldsymbol{FP}}$$

- F1 : F1 is harmonic mean of precision and recall. This metric allows tradeoff between precision and recall while evaluating performance:

$$\boldsymbol{F1} = \frac{\boldsymbol{2} \times \boldsymbol{Precision} \times \boldsymbol{Recall}}{\boldsymbol{Precision} + \boldsymbol{Recall}}$$

Apart from above metrics it is also important to evaluate model stability. Model stability can be estimated by different graphs: precision-recall curve, ROC-AUC curve and Gain-and-Lift Charts.

- Precision-Recall curve describes precision-recall tradeoff for different score threshold of model.
- ROC-AUC curve is another performance metric which shows probability and separability of class prediction.

These curve helps understand overall performance of model for different threshold of probability. A score threshold is limit of probability after which models assigns an instants one of the class. Following is sample curves:

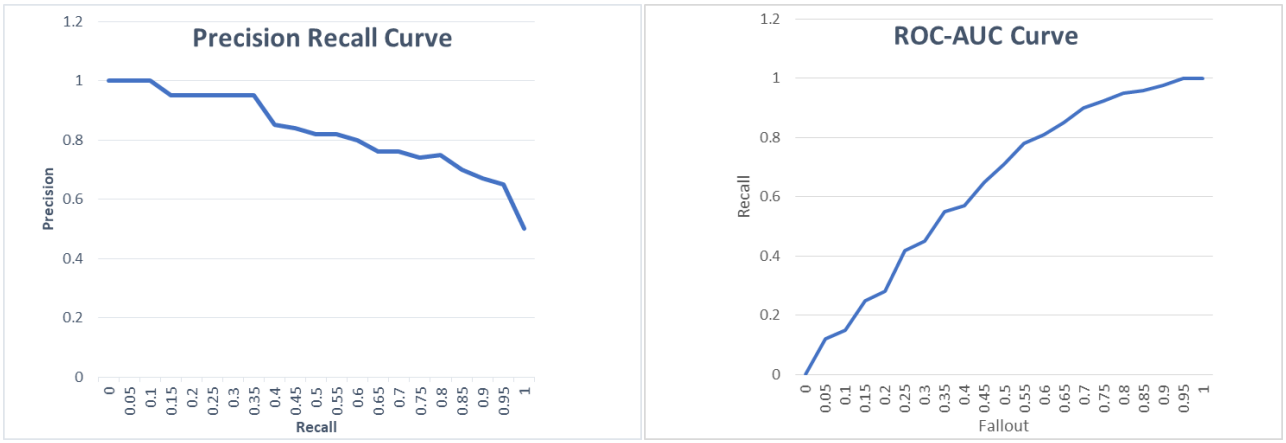


Figure 4-A: Examples of ROC and Precision-Recall curve

4.2Model Building

Since objective is to predict loan default cases, classification models need to be used to achieve objective. As there are several classification algorithms available, before finalizing one, different algorithms are tried:

- Logistic Regression
- Decision Tree Classifier
- KNN Classifier
- Random Forest Classifier
- SVM Classifier
- Naïve Bayes Classifier
- XGBoost Classifier

4.2.1 Logistic Regression

Logistic regression builds a model between target variable and features by assuming linear relationship between log of features and target variable.

- **Hyper-parameter Optimization**

With GridSearchCV following parameters are obtained:

```
Optimum Parameters: {'class_weight': 'balanced', 'fit_intercept': False}
Optimum score     : 0.737987987987988
```

Figure 4-B: Logistic regression optimum hyper-parameter

- **Model Performance**

Following are scores of train and test data sets:

Table 4-ii : Performance of Linear Regression

```
-------------------------Report on Train Data--------------------
                    precision    recall  f1-score   support

               0.0       0.88      0.73      0.80       362
               1.0       0.48      0.73      0.58       128

          accuracy                           0.73       490
         macro avg       0.68      0.73      0.69       490
      weighted avg       0.78      0.73      0.74       490

-------------------------Report on Test Data--------------------
                    precision    recall  f1-score   support

               0.0       0.91      0.71      0.80       155
               1.0       0.49      0.80      0.61        55

          accuracy                           0.73       210
         macro avg       0.70      0.75      0.70       210
      weighted avg       0.80      0.73      0.75       210

----------------------------------------------------------------------
AUC Score : 0.5782457854882305
----------------------------------------------------------------------
```

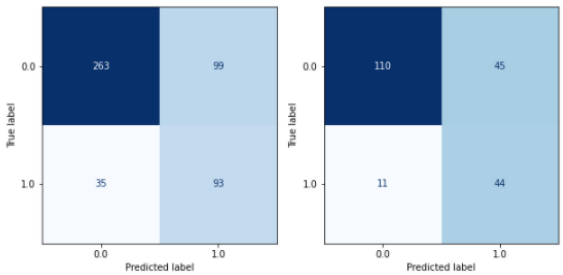


Figure 4-C: Confusion Matrix for Logistic Regression

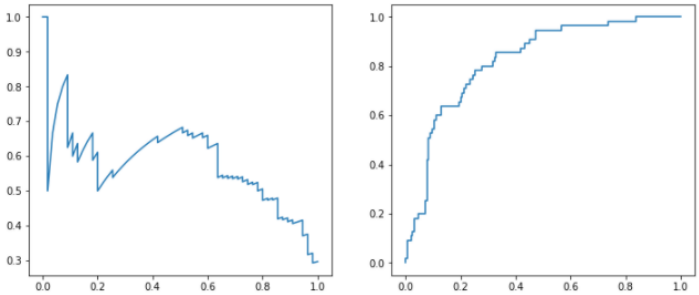


Figure 4-D: Precision-Recall & ROC Curves (Log Reg)

4.2.2 Support Vector Classifier

Support Mechanism plots data in n-dimensional feature and creates hyperplane to classify each data.

❖ Hyper-parameter Optimization

With GridSearchCV following parameters are obtained:

```
Optimum Parameters: {'C': 1, 'class_weight': 'balanced', 'kernel': 'poly'}
Optimum score     : 0.7707207207207207
```

Figure 4-E: SV classifier optimum hyper-parameter

❖ Model Performance

Following are scores of train and test data sets:

Table 4-iii : Performance of SV Classifier

```
-----------------------Report on Train Data----------------------
              precision    recall  f1-score   support

         0.0       0.95      0.68      0.79       362
         1.0       0.50      0.89      0.64       128

    accuracy                           0.73       490
   macro avg       0.72      0.79      0.71       490
weighted avg       0.83      0.73      0.75       490

------------------------Report on Test Data----------------------
              precision    recall  f1-score   support

         0.0       0.89      0.63      0.73       155
         1.0       0.43      0.78      0.55        55

    accuracy                           0.67       210
   macro avg       0.66      0.70      0.64       210
weighted avg       0.77      0.67      0.69       210

----------------------------------------------------------------------
AUC Score : 0.5655284977724268
----------------------------------------------------------------------
```

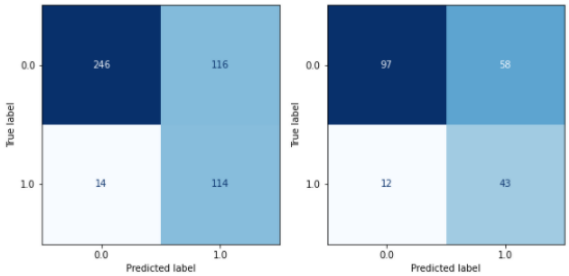


Figure 4-F: Confusion Matrix for SV Classifier

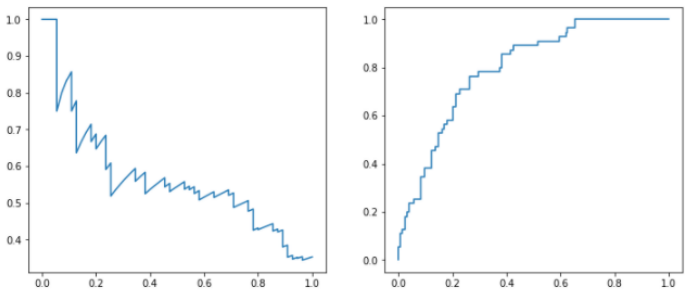


Figure 4-G: Precision-Recall & ROC Curves (SVC)

4.2.3 Decision Tree Classifier

This method creates decision tree based on features to obtain prediction about target variable.

❖ Hyper-parameter Optimization

With GridSearchCV following parameters are obtained:

```
Optimum Parameters: {'class_weight': 'balanced', 'criterion': 'entropy',
'max_depth': 4, 'min_samples_split': 4, 'splitter': 'random'}
Optimum score      : 0.81996996996997
```

Figure 4-H: Decision Tree classifier optimum hyper-parameter

❖ Model Performance

Following are scores of train and test data sets:

Table 4-iv : Performance of Decision Tree Classifier

```
------------------------Report on Train Data----------------------
              precision    recall  f1-score   support

         0.0       0.90      0.65      0.76       362
         1.0       0.45      0.80      0.58       128

    accuracy                           0.69       490
   macro avg       0.68      0.73      0.67       490
weighted avg       0.79      0.69      0.71       490

------------------------Report on Test Data-----------------------
              precision    recall  f1-score   support

         0.0       0.89      0.62      0.73       155
         1.0       0.42      0.78      0.55        55

    accuracy                           0.66       210
   macro avg       0.66      0.70      0.64       210
weighted avg       0.77      0.66      0.68       210

------------------------------------------------------------------
AUC Score : 0.481202956955337
------------------------------------------------------------------
```

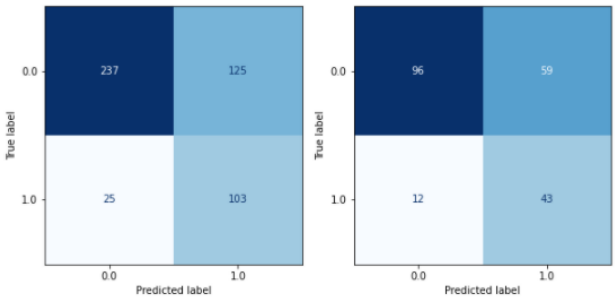


Figure 4-I: Confusion Matrix for Decision Tree Classifier

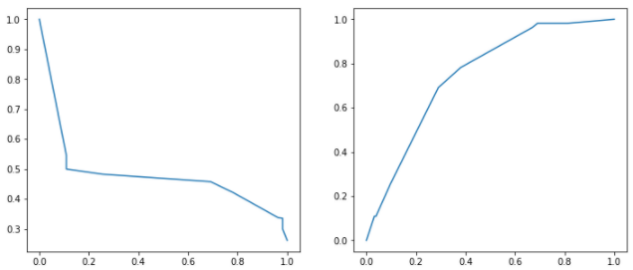


Figure 4-J: Precision-Recall & ROC Curves (Decision Tree)

4.2.4 Random Forest Classifier

This method creates multiple decision trees based on features to obtain prediction about target variable.

❖ Hyper-parameter Optimization

With RandomizedSearchCV following parameters are obtained:

```
Optimum Parameters: {'min samples split': 8, 'min samples leaf': 7, 'max features': 'auto',
'max_depth': 4, 'criterion': 'gini', 'class_weight': 'balanced', 'bootstrap': False}
Optimum score     : 0.7216216216216216
```

Figure 4-K: Random forest classifier optimum hyper-parameter

❖ Model Performance

Following are scores of train and test data sets:

Table 4-v : Performance of Random Forest Classifier

```
------------------------Report on Train Data----------------------
              precision    recall  f1-score   support

         0.0       0.90      0.65      0.76       362
         1.0       0.45      0.80      0.58       128

    accuracy                           0.69       490
   macro avg       0.68      0.73      0.67       490
weighted avg       0.79      0.69      0.71       490

------------------------Report on Test Data-----------------------
              precision    recall  f1-score   support

         0.0       0.89      0.62      0.73       155
         1.0       0.42      0.78      0.55        55

    accuracy                           0.66       210
   macro avg       0.66      0.70      0.64       210
weighted avg       0.77      0.66      0.68       210

------------------------------------------------------------------
AUC Score : 0.481202956955337
------------------------------------------------------------------
```

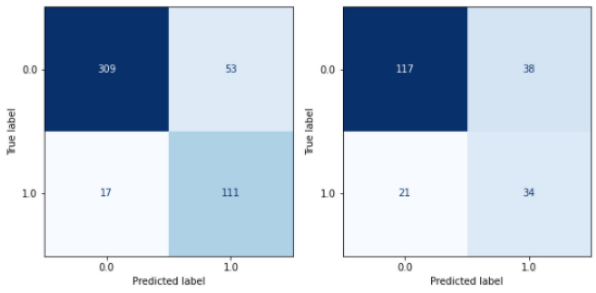


Figure 4-L: Confusion Matrix for Random Forest Classifier

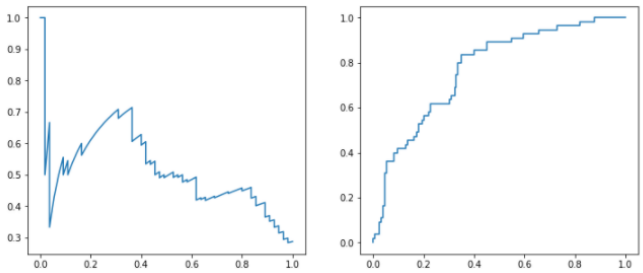


Figure 4-M: Precision-Recall & ROC Curves (Random Forest)

4.2.5 Naïve Bayes Classifier

This method is based on conditional probability of event, probability is estimated based on each feature and then probability of target variable is estimated.

- **Hyper-parameter Optimization**

With GridSearchCV following parameters are obtained:

```
Optimum Parameters: {'alpha': 6, 'norm': False}
Optimum score     : 0.737987987987988
```

Figure 4-N: Naïve Bayes classifier optimum hyper-parameter

- **Model Performance**

Following are scores of train and test data sets:

Table 4-vi : Performance of Naïve Bayes Classifier

```
--------------------------Report on Train Data--------------------
              precision    recall  f1-score   support

         0.0       0.88      0.72      0.80       362
         1.0       0.48      0.73      0.58       128

    accuracy                           0.72       490
   macro avg       0.68      0.73      0.69       490
weighted avg       0.78      0.72      0.74       490

--------------------------Report on Test Data---------------------
              precision    recall  f1-score   support

         0.0       0.92      0.71      0.80       155
         1.0       0.50      0.82      0.62        55

    accuracy                           0.74       210
   macro avg       0.71      0.76      0.71       210
weighted avg       0.81      0.74      0.75       210

------------------------------------------------------------------
AUC Score : 0.5282597060280302
------------------------------------------------------------------
```

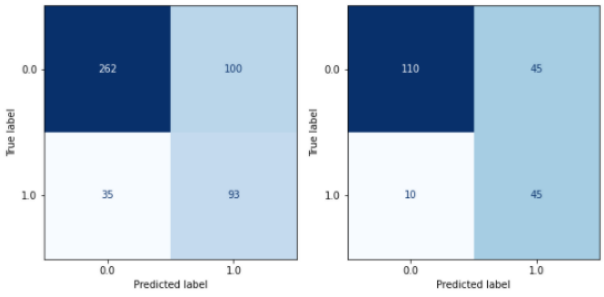


Figure 4-O: Confusion Matrix for Naïve Bayes Classifier

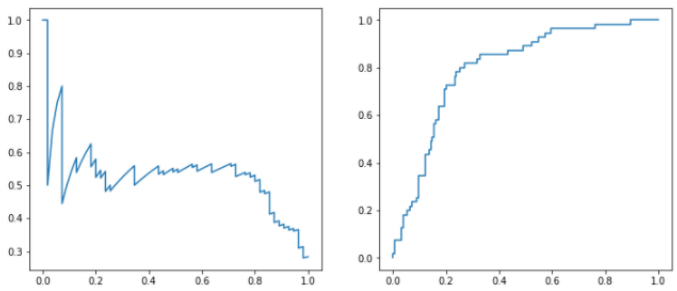


Figure 4-P: Precision-Recall & ROC Curves (Naïve Bayes)

4.2.6 KNN Neighbors Classifier

This method predicts output class based on distance between observed points and assigns class of nearest group to target variable.

❖ Hyper-parameter Optimization

With GridSearchCV following parameters are obtained:

```
Optimum Parameters: {'n_neighbors': 4, 'p': 2, 'weights': 'distance'}
Optimum score     : 0.421021021021021
```

Figure 4-Q: KNN classifier optimum hyper-parameter

❖ Model Performance

Following are scores of train and test data sets:

Table 4-vii : Performance of KNN Classifier

```
-----------------------Report on Train Data-----------------------
                    precision    recall  f1-score   support

               0.0       1.00      1.00      1.00       362
               1.0       1.00      1.00      1.00       128

          accuracy                           1.00       490
         macro avg       1.00      1.00      1.00       490
      weighted avg       1.00      1.00      1.00       490

-----------------------Report on Test Data-----------------------
                    precision    recall  f1-score   support

               0.0       0.82      0.84      0.83       155
               1.0       0.51      0.47      0.49        55

          accuracy                           0.74       210
         macro avg       0.66      0.66      0.66       210
      weighted avg       0.74      0.74      0.74       210

------------------------------------------------------------------------
AUC Score : 0.46163520241606987
```

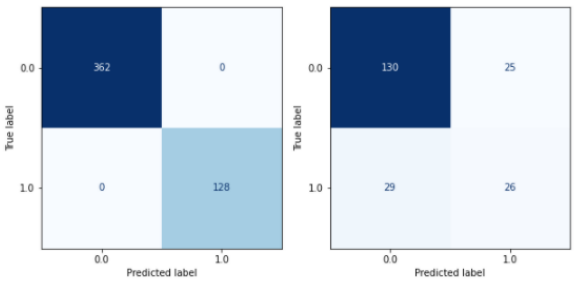


Figure 4-R: Confusion Matrix for KNN Classifier

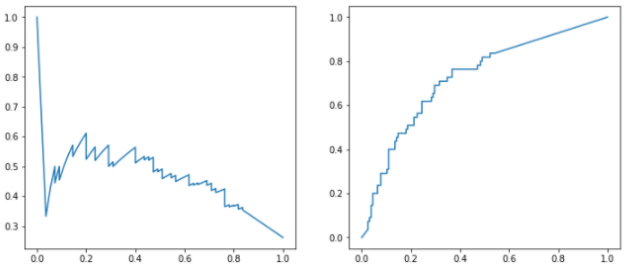


Figure 4-S: Precision-Recall & ROC Curves (KNN)

4.2.7 XGB Classifier

This method predicts output class based on distance between observed points and assigns class of nearest group to target variable.

- **Hyper-parameter Optimization**

With RandomizedSearchCV following parameters are obtained:

```
Optimum Parameters: {'reg_alpha': 0.18873918221350977, 'n_estimators': 300,
'learning_rate': 0.9, 'colsample_bytree': 0.30000000000000004}
Optimum score     : 0.8014285714285714
```

Figure 4-T: XGB classifier optimum hyper-parameter

- **Model Performance**

Following are scores of train and test data sets:

Table 4-viii : Performance of XGB Classifier

```
-------------------------Report on Train Data----------------------
                    precision    recall  f1-score   support

               0.0       1.00      1.00      1.00       362
               1.0       1.00      1.00      1.00       128

          accuracy                           1.00       490
         macro avg       1.00      1.00      1.00       490
      weighted avg       1.00      1.00      1.00       490

-------------------------Report on Test Data-----------------------
                    precision    recall  f1-score   support

               0.0       0.80      0.87      0.84       155
               1.0       0.52      0.40      0.45        55

          accuracy                           0.75       210
         macro avg       0.66      0.64      0.64       210
      weighted avg       0.73      0.75      0.74       210

-----------------------------------------------------------------------
AUC Score : 0.5151148857684276
```

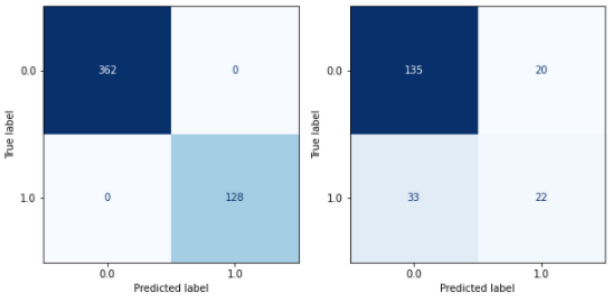


Figure 4-U: Confusion Matrix for XGB Classifier

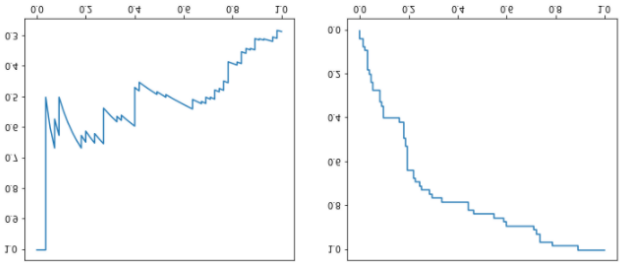


Figure 4-V: Precision-Recall & ROC Curves (XGB)

4.3 Model Selection

Based on evaluation metrics, XGB classifier had better result hence it was finalized for prediction data.
Recall for default case in all model is comparatively low. This was accepted because we did not want to lose probable customer for bank because of misclassification by model.
ROC and Precision-recall curves were used to test stability of model. On this parameter XGB model performed pretty well compared to other tested models.
Therefore considering all factors XGB classifier was selected.

5 Conclusion

Though the selected model performed ok, a lot more improvements are needed to be done to achieve better results:

- The data needs to be much more to improve performance of model. Current data feels lot shorter to train a machine learning model.
- Different scaling techniques were tried to normalize data, yet results did not improve significantly.
- Consultation with banking expert may help to build a better model for prediction. Expert's opinion may help in gaining significant insights for prediction of default cases.

And obviously analyzing data as it is generated will also help for betterment of model.

6 Appendix A

PYTHON CODE

Importing libraries and setting up directory

Libraries used:

1. os: to handle directory structure
2. panda: to handle dataframes
3. numpy and scipy: for numeric and scintific calculation methods
4. matplotlib and seaborn: to plot and graphic visualization of data
5. sklearn: for various machine learning models and hyper parameter tuning algorithms
6. patsy: to define structure of model
7. xgboost: to improve accuracy through ensemble technique
8. joblib: to save model

In [1]:
```
import os
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
from patsy import dmatrices
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import QuantileTransformer, MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn import metrics as mtr
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import ComplementNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
import xgboost as xgb
import joblib
```

Loading and getting an overview of train and test data

csv file loaded in dataframe[dfLoaner] info() and describe() methods used on data

In [2]:
```
os.chdir('K:\\Data_Science\\Project\\Project_02')
```

In [3]:
```
dfLoaner = pd.read_csv('01_Data\\bank-loan.csv')
```

In [4]:
```
dfLoaner.describe()
```

Out[4]:

	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
count	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	850.000000	700.000000
mean	35.029412	1.710588	8.565882	8.371765	46.675294	10.171647	1.576805	3.078789	0.261429
std	8.041432	0.927784	6.777884	6.895016	38.543054	6.719441	2.125840	3.398803	0.439727
min	20.000000	1.000000	0.000000	0.000000	13.000000	0.100000	0.011696	0.045584	0.000000
25%	29.000000	1.000000	3.000000	3.000000	24.000000	5.100000	0.382176	1.045942	0.000000
50%	34.000000	1.000000	7.000000	7.000000	35.000000	8.700000	0.885091	2.003243	0.000000
75%	41.000000	2.000000	13.000000	12.000000	55.750000	13.800000	1.898440	3.903001	1.000000
max	56.000000	5.000000	33.000000	34.000000	446.000000	41.300000	20.561310	35.197500	1.000000

In [5]:
```
dfLoaner.info()
```

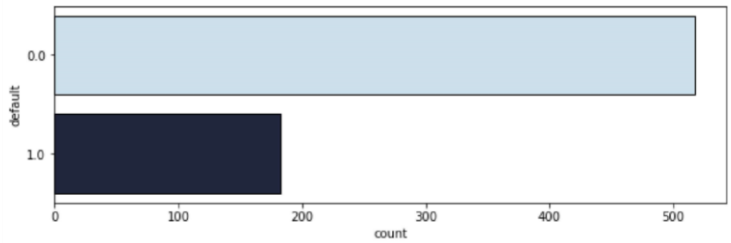
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 850 entries, 0 to 849
Data columns (total 9 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       850 non-null    int64
 1   ed        850 non-null    int64
 2   employ    850 non-null    int64
 3   address   850 non-null    int64
 4   income    850 non-null    int64
 5   debtinc   850 non-null    float64
 6   creddebt  850 non-null    float64
 7   othdebt   850 non-null    float64
 8   default   700 non-null    float64
dtypes: float64(4), int64(5)
memory usage: 59.9 KB
```

Visualizing distribution of the data in train dataset

- horizontal bar graph used for class distribution
- probability distribution plot used for analysis of individual variables
- pairwise plot used for correlation of each pair of variables

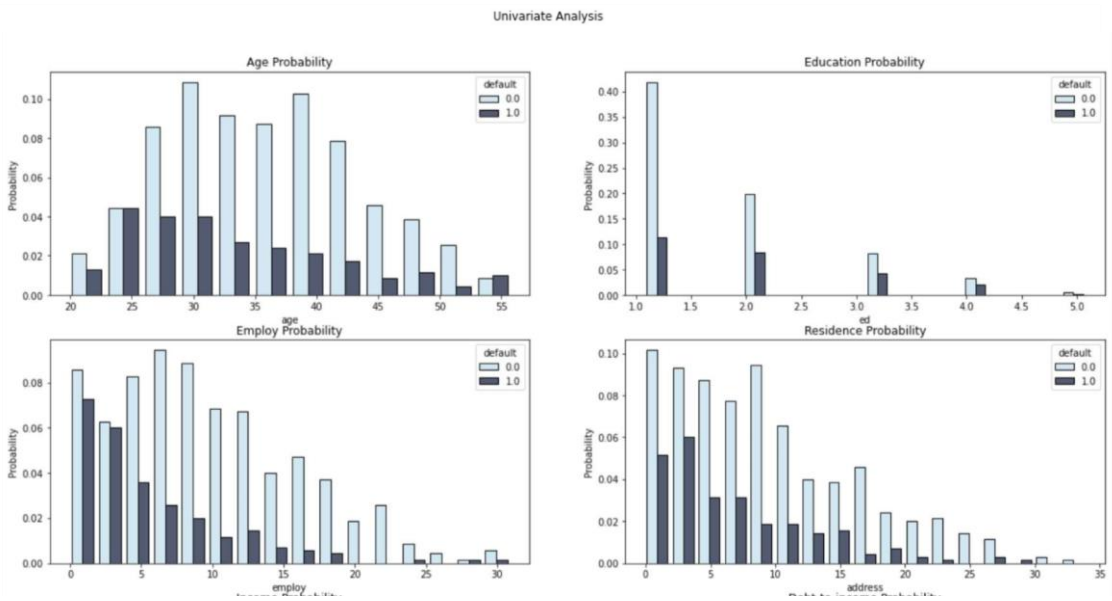
In [6]:
```
clr_set = {0 : sns.color_palette("cubehelix")[-1], 1 : sns.color_palette("cubehelix")[0]}
fig, axes = plt.subplots(figsize=(10,3))
sns.countplot(y = 'default', data = dfLoaner, palette = clr_set, ax = axes, edgecolor = "black")
```

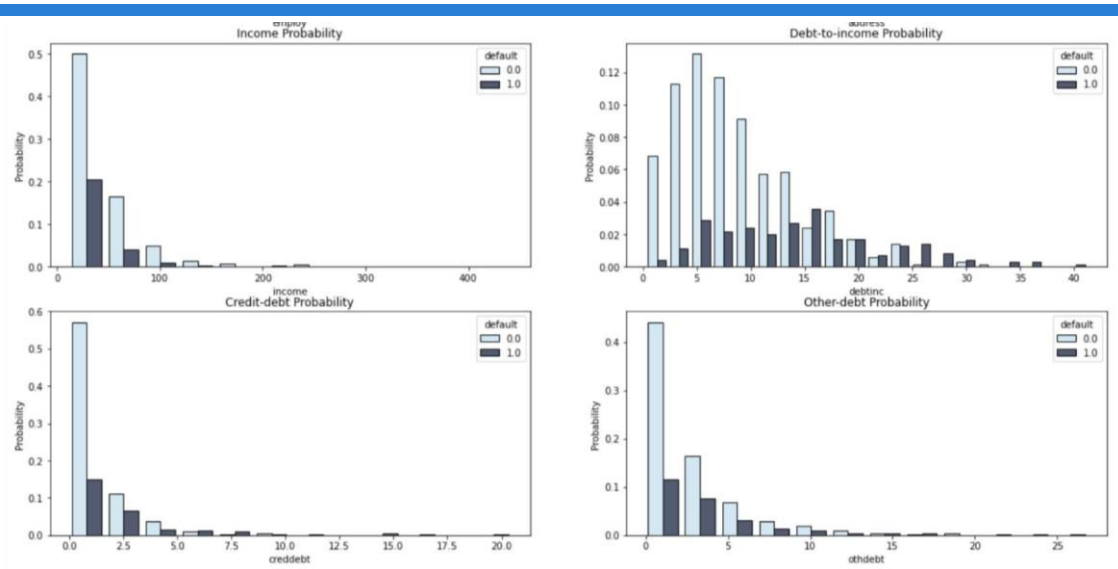
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1c5d3ab0df0>



In [7]:
```
fig, axes = plt.subplots(4, 2, figsize=(20,20) )
fig.suptitle('Univariate Analysis')

axes[0, 0].set_title('Age Probability')
sns.histplot(x='age', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8, bi
ns=12, ax=axes[0, 0])

axes[0, 1].set_title('Education Probability')
sns.histplot(x='ed', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8, ax=
axes[0, 1])

axes[1, 0].set_title('Employ Probability')
sns.histplot(x='employ', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8,
ax=axes[1, 0])

axes[1, 1].set_title('Residence Probability')
sns.histplot(x='address', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8
, ax=axes[1, 1])

axes[2, 0].set_title('Income Probability')
sns.histplot(x='income', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8,
bins=12, ax=axes[2, 0])

axes[2, 1].set_title('Debt-to-income Probability')
sns.histplot(x='debtinc', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8
, ax=axes[2, 1])

axes[3, 0].set_title('Credit-debt Probability')
sns.histplot(x='creddebt', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.
8, bins=12, ax=axes[3, 0])

axes[3, 1].set_title('Other-debt Probability')
sns.histplot(x='othdebt', data=dfLoaner, hue='default', multiple='dodge', palette=clr_set, stat='probability', shrink=.8
, bins=12, ax=axes[3, 1])
```

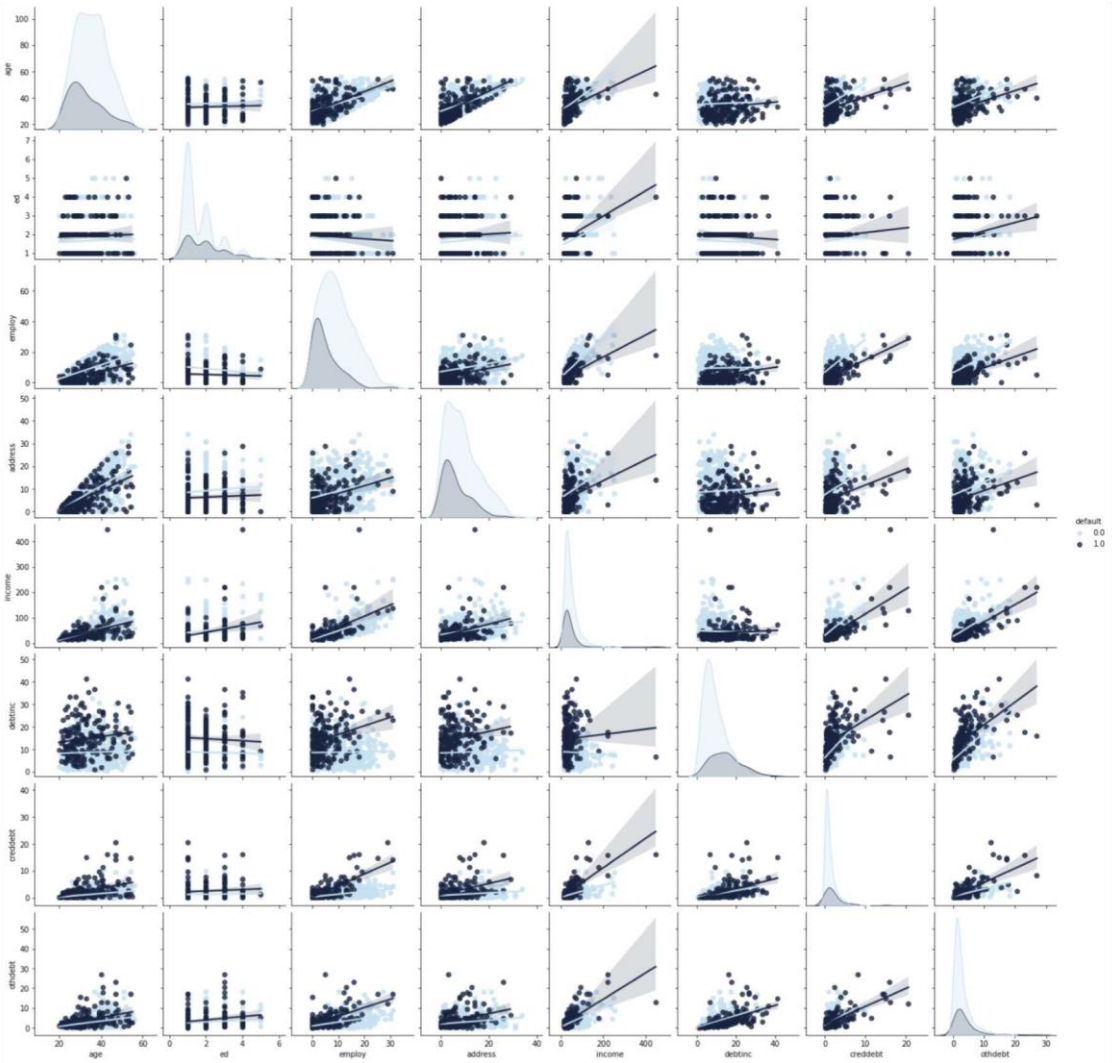
Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1c5d4318be0>





In [8]:
```
sns.pairplot(data = dfLoaner, kind = 'reg', dropna = True, hue = 'default', palette = clr_set)
```

Out[8]: <seaborn.axisgrid.PairGrid at 0x1c5d4fc27f0>



Cleaning data train data

Missing Value Analysis

Missing values in data checked

In [9]:
```
miss_val = pd.DataFrame(dfLoaner.drop(['default'], axis = 1).isnull().sum())
miss_val = miss_val.rename(columns = {'index': 'Variables', 0: 'Missing Count'})
miss_val['Missing Pcent'] = (miss_val['Missing Count']/len(dfLoaner))*100
miss_val
```

Out[9]:

	Missing Count	Missing Pcent
age	0	0.0
ed	0	0.0
employ	0	0.0
address	0	0.0
income	0	0.0
debtinc	0	0.0
creddebt	0	0.0
othdebt	0	0.0

Outlier Analysis

Outlier values in data checked

In [10]:
```
def countOutlier(col):
    Q1 = np.percentile(dfLoaner[col], 25)
    Q3 = np.percentile(dfLoaner[col], 75)
    Max_Bound = Q3 + 1.5*(Q3-Q1)
    Min_Bound = Q1 - 1.5*(Q3-Q1)

    no_of_outliers = len(dfLoaner.loc[((dfLoaner[col]<Min_Bound)|(dfLoaner[col]>Max_Bound)), col])
    return no_of_outliers

outlier_len = [{col : countOutlier(col)} for col in dfLoaner.columns if col not in ['ed', 'default']]
print(outlier_len)
```

```
[{'age': 0}, {'employ': 10}, {'address': 19}, {'income': 53}, {'debtinc': 21}, {'creddebt': 69}, {'othdebt': 61}]
```

Feature Transformation

Fetures sclaed to be on same scale. Feature with outliers treated with quantile transformation And the rest were treated with min-max transformation

In [11]:
```
to_quantile = ['employ', 'address', 'income', 'debtinc', 'creddebt', 'othdebt']
q_trans = QuantileTransformer(n_quantiles=100)
qnt_scaled = q_trans.fit_transform(X = dfLoaner[to_quantile])

to_minmax = ['age', 'ed']
m_trans = MinMaxScaler()
mnx_scaled = m_trans.fit_transform(X = dfLoaner[to_minmax])
```

In [12]:
```
for i, col in enumerate(to_quantile):
    dfLoaner[col] = qnt_scaled[:,i]
for i, col in enumerate(to_minmax):
    dfLoaner[col] = mnx_scaled[:,i]

dfLoaner.head()
```

Out[12]:

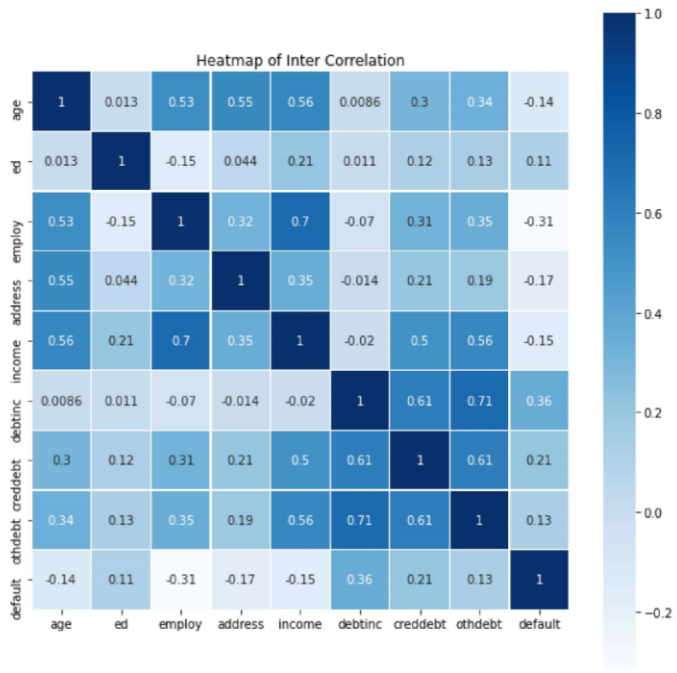
	age	ed	employ	address	income	debtinc	creddebt	othdebt	default
0	0.583333	0.50	0.873737	0.737374	0.982940	0.532051	0.991014	0.827993	1.0
1	0.194444	0.00	0.641414	0.449495	0.429293	0.858586	0.642328	0.758798	0.0
2	0.555556	0.00	0.813131	0.797980	0.747475	0.272727	0.486328	0.534158	0.0
3	0.583333	0.00	0.813131	0.797980	0.958333	0.101010	0.838626	0.173070	0.0
4	0.111111	0.25	0.181818	0.000000	0.363636	0.858586	0.728728	0.677087	1.0

Feature Selection

distribution and correlation after missing value and oulier correction checked If any variables are correleated is checked with help of heatmap of correlation coeficient

In [13]:
```
fig, axes = plt.subplots(figsize=(10,10))
axes.set_title('Heatmap of Inter Correlation')
sns.heatmap(dfLoaner.corr(), square=True, linewidths=0.2,linecolor='w', annot=True, cmap = "Blues", ax = axes)
```

Out[13]: <matplotlib.axes._subplots.AxesSubplot at 0x1c5d791eca0>



Checking multicolinearity in data

with help of VIF, multicolinearity in data is checked.

- if VIF = 1 -> No colinearity in any variables.
- if 1 <= VIF <= 5 -> Moderate corinearity
- if VIF > 5 -> High colinearity if multiple variables have more than 5 VIF, one with highest VIF will be dropped

In [14]:
```
features = " + ".join(dfLoaner.columns)
features = features.replace(" + default", "")

target, attributes = dmatrices("default ~ " + features, dfLoaner, return_type='dataframe')

vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(attributes.values, i) for i in range(attributes.shape[1])]
vif['INPUTS'] = attributes.columns
vif
```

Out[14]:

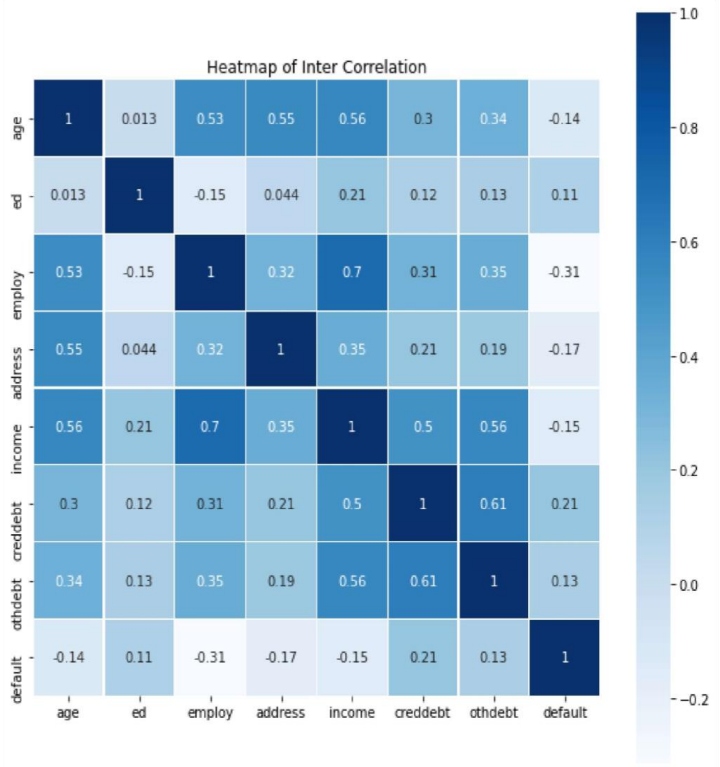
	VIF	INPUTS
0	13.595078	Intercept
1	1.924522	age
2	1.281836	ed
3	2.544158	employ
4	1.453116	address
5	8.105853	income
6	10.594380	debtinc
7	4.123676	creddebt
8	8.960878	othdebt

As observed from these two analysis 'debtinc' feature has introduced multicolinearity in data. We will drop this feature from our data.

In [15]:
```
dfLoaner = dfLoaner.drop(['debtinc'], axis=1)
fig, axes = plt.subplots(figsize=(10,10))
axes.set_title('Heatmap of Inter Correlation')
sns.heatmap(dfLoaner.corr(), square=True, linewidths=0.2,linecolor='w', annot=True, cmap = "Blues", ax = axes)

features = " + ".join(dfLoaner.columns)
features = features.replace(" + default", "")

target, attributes = dmatrices("default ~ " + features, dfLoaner, return_type='dataframe')

vif = pd.DataFrame()
vif['VIF'] = [variance_inflation_factor(attributes.values, i) for i in range(attributes.shape[1])]
vif['INPUTS'] = attributes.columns
vif
```

Out[15]:

	VIF	INPUTS
0	7.678074	Intercept
1	1.924079	age
2	1.275941	ed
3	2.537927	employ
4	1.448019	address
5	3.391935	income
6	1.767355	creddebt
7	1.848654	othdebt



Building models and checking performance

- train data split to validate performance.
- sevral functions defined to visualize and calculate model statistics

In [16]:
```
dfTest = dfLoaner[dfLoaner['default'].isna()].copy().drop(['default'], axis = 1)
dfTrain = dfLoaner[dfLoaner['default'].notna()].copy()
```

In [17]:
```
X = dfTrain.drop(['default'], axis = 1).values
Y = dfTrain['default'].values

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3, random_state=35, stratify=Y)
```

In [18]:
```
def eval_model(model):
    y_trPred = model.predict(X_train)
    y_tsPred = model.predict(X_test)

    fig, axes = plt.subplots(1, 2, figsize = (10,5))
    mtr.plot_confusion_matrix(model, X_train, Y_train, cmap = 'Blues', ax = axes[0], colorbar = False)
    mtr.plot_confusion_matrix(model, X_test, Y_test, cmap = 'Blues', ax = axes[1], colorbar = False)
    print('-----------------------------------Report on Train Data-----------------------------')
    print(mtr.classification_report(Y_train, y_trPred))
    print('-----------------------------------Report on Test Data-----------------------------')
    print(mtr.classification_report(Y_test, y_tsPred))
    print('-----------------------------------------------------------------------------------')
    return None

def plot_model(model):
    Y_prob = model.predict_proba(X_test)[:,1]
    precision, recallp, thresholds = mtr.precision_recall_curve(Y_test, Y_prob)
    fallout, recallr, thresholds = mtr.roc_curve(Y_test, Y_prob)

    auc_pr = mtr.auc(recallp, precision)
    print('AUC Score :',auc_pr)
    print('-----------------------------------------------------------------------------------')
    fig1, ax1 = plt.subplots(1, 2, figsize = (12,5))
    ax1[0].plot(recallp, precision)
    ax1[1].plot(fallout, recallr)

    return None
```

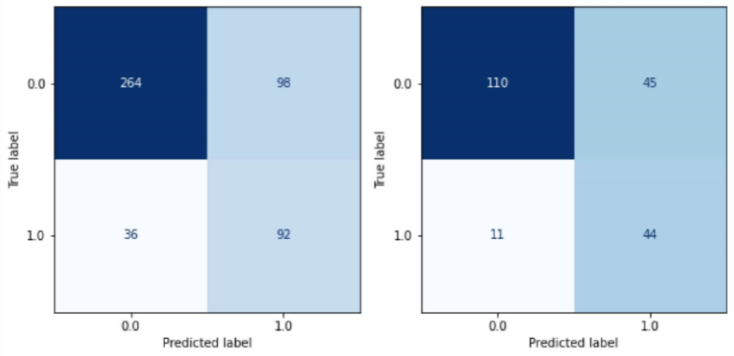
Logistic Regression

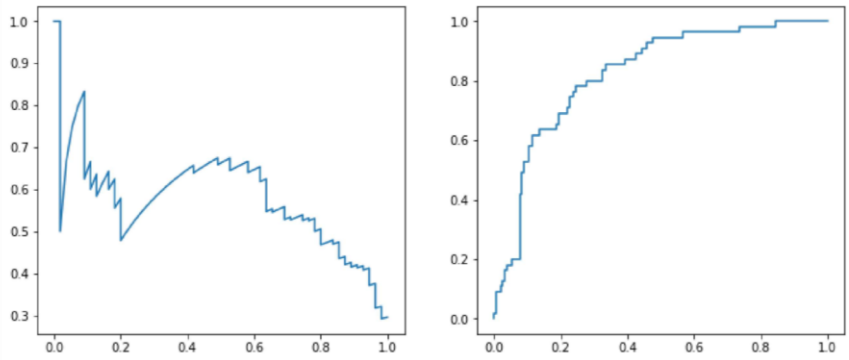
In [19]:
```
lgreg_check = LogisticRegression(penalty = 'l2')
param_grid = {'fit_intercept': [True, False], 'class_weight': [None, 'balanced']}

lgreg_cv = GridSearchCV(lgreg_check, param_grid, cv=5, scoring='recall')
lgreg_cv.fit(X, Y)

print("Optimum Parameters: {}".format(lgreg_cv.best_params_))
print("Optimum score     : {}".format(lgreg_cv.best_score_))
```

```
Optimum Parameters: {'class_weight': 'balanced', 'fit_intercept': False}
Optimum score     : 0.7325825825825826
```

In [20]:
```
lgreg_clf = LogisticRegression(penalty = 'l2', fit_intercept = False, class_weight = 'balanced')
lgreg_clf.fit(X_train, Y_train)

eval_model(lgreg_clf)
plot_model(lgreg_clf)
```

```
----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.88      0.73      0.80       362
         1.0       0.48      0.72      0.58       128

    accuracy                           0.73       490
   macro avg       0.68      0.72      0.69       490
weighted avg       0.78      0.73      0.74       490

----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.91      0.71      0.80       155
         1.0       0.49      0.80      0.61        55

    accuracy                           0.73       210
   macro avg       0.70      0.75      0.70       210
weighted avg       0.80      0.73      0.75       210

------------------------------------------------------------------------------------
AUC Score : 0.5760996444611247
------------------------------------------------------------------------------------
```





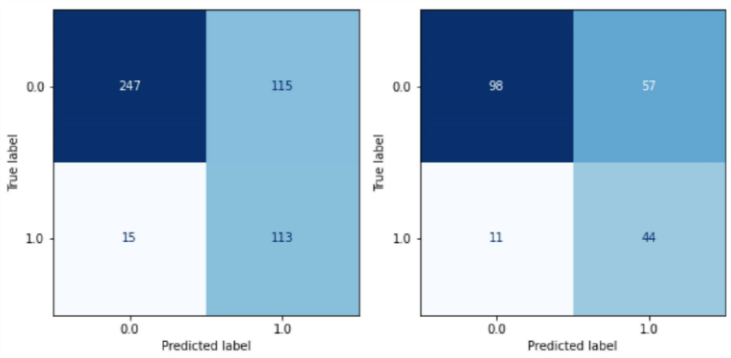
Support Vector Mechanics

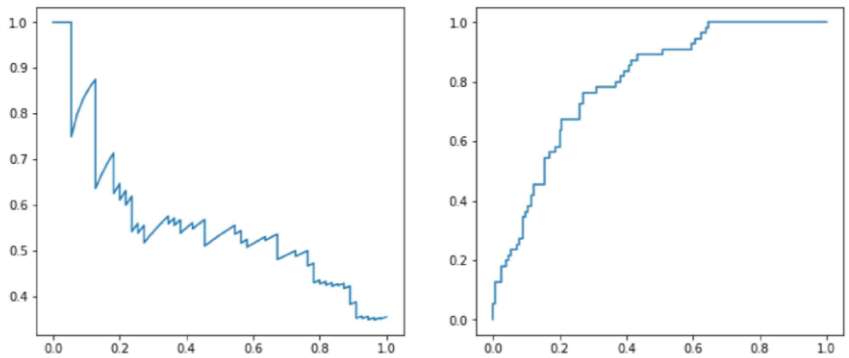
In [21]:
```
svc_check = SVC()
param_grid = {'C': list(range(1,5)), 'class_weight': [None, 'balanced'], 'kernel': ['linear', 'poly', 'rbf']}

svc_cv = GridSearchCV(svc_check, param_grid, cv=5, scoring='recall')
svc_cv.fit(X, Y)

print("Optimum Parameters: {}".format(svc_cv.best_params_))
print("Optimum score     : {}".format(svc_cv.best_score_))
```

```
Optimum Parameters: {'C': 1, 'class_weight': 'balanced', 'kernel': 'rbf'}
Optimum score     : 0.7707207207207208
```

In [22]:
```
svc_clf = SVC(C=1, class_weight='balanced', kernel='poly', probability = True)
svc_clf.fit(X_train, Y_train)

eval_model(svc_clf)
plot_model(svc_clf)
```

```
-----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.94      0.68      0.79       362
         1.0       0.50      0.88      0.63       128

    accuracy                           0.73       490
   macro avg       0.72      0.78      0.71       490
weighted avg       0.83      0.73      0.75       490

-----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.90      0.63      0.74       155
         1.0       0.44      0.80      0.56        55

    accuracy                           0.68       210
   macro avg       0.67      0.72      0.65       210
weighted avg       0.78      0.68      0.70       210

-------------------------------------------------------------------------------------
AUC Score : 0.560316705500102
-------------------------------------------------------------------------------------
```





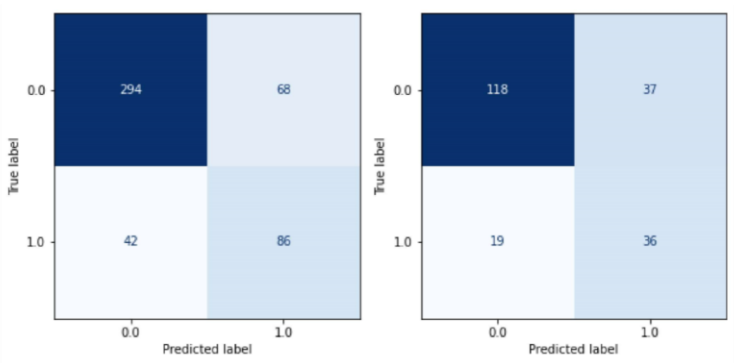
Decision Tree Classifier

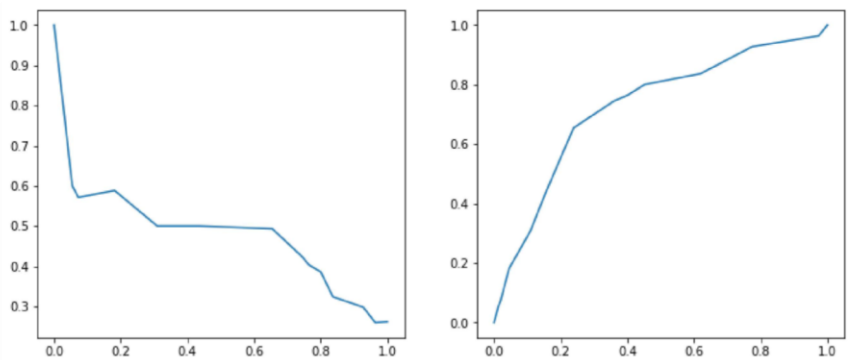
In [23]:
```
dtree_check = DecisionTreeClassifier()
param_grid = {'criterion': ['gini', 'entropy'], 'class_weight': [None, 'balanced'], 'splitter': ['best', 'random'],
              'max_depth': range(2,10,2), 'min_samples_split': range(2,10,2)}

dtree_cv = GridSearchCV(dtree_check, param_grid, cv=5, scoring='recall')
dtree_cv.fit(X, Y)

print("Optimum Parameters: {}".format(dtree_cv.best_params_))
print("Optimum score     : {}".format(dtree_cv.best_score_))
```

```
Optimum Parameters: {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 4, 'min_samples_split': 4, 'splitt
er': 'random'}
Optimum score     : 0.8525525525525527
```

In [24]:
```
dtree_clf = DecisionTreeClassifier(criterion ='entropy', class_weight='balanced', max_depth=4, min_samples_split=4, spli
tter='random')
dtree_clf.fit(X_train, Y_train)

eval_model(dtree_clf)
plot_model(dtree_clf)
```

```
----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.88      0.81      0.84       362
         1.0       0.56      0.67      0.61       128

    accuracy                           0.78       490
   macro avg       0.72      0.74      0.73       490
weighted avg       0.79      0.78      0.78       490

----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.86      0.76      0.81       155
         1.0       0.49      0.65      0.56        55

    accuracy                           0.73       210
   macro avg       0.68      0.71      0.69       210
weighted avg       0.76      0.73      0.74       210

-------------------------------------------------------------------------------------
AUC Score : 0.4830950108580506
-------------------------------------------------------------------------------------
```





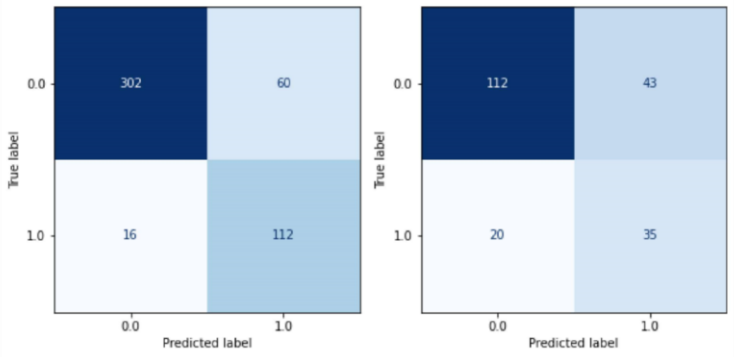
Random Forest Classifier

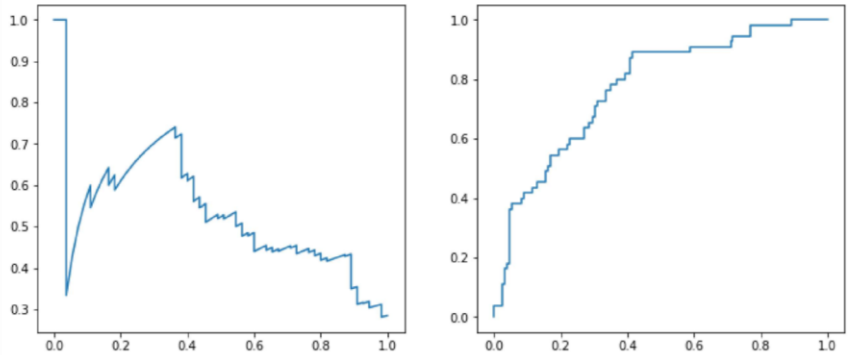
In [25]:
```
rndfor_check = RandomForestClassifier()
param_grid = {'criterion': ['gini', 'entropy'], 'class_weight': [None, 'balanced', 'balanced_subsample'],
              'max_depth': range(2,10,2), 'min_samples_split': range(2,10,2), 'bootstrap': [True, False],
              'max_features': ['auto', 'log2', None], 'min_samples_leaf': range(1,11,2)}

rndfor_cv = RandomizedSearchCV(rndfor_check, param_grid, cv=5, scoring='recall')
rndfor_cv.fit(X, Y)

print("Optimum Parameters: {}".format(rndfor_cv.best_params_))
print("Optimum score     : {}".format(rndfor_cv.best_score_))
```

```
Optimum Parameters: {'min_samples_split': 2, 'min_samples_leaf': 9, 'max_features': 'auto', 'max_depth': 4, 'criterion':
'gini', 'class_weight': 'balanced', 'bootstrap': False}
Optimum score     : 0.6885885885885887
```

In [26]:
```
rndfor_clf = RandomForestClassifier(criterion='gini', class_weight='balanced', max_depth=4, min_samples_split=8, min_sam
ples_leaf=7,
                                    bootstrap=False, max_features='auto')
rndfor_clf.fit(X_train, Y_train)

eval_model(rndfor_clf)
plot_model(rndfor_clf)
```

```
----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.95      0.83      0.89       362
         1.0       0.65      0.88      0.75       128

    accuracy                           0.84       490
   macro avg       0.80      0.85      0.82       490
weighted avg       0.87      0.84      0.85       490

----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.85      0.72      0.78       155
         1.0       0.45      0.64      0.53        55

    accuracy                           0.70       210
   macro avg       0.65      0.68      0.65       210
weighted avg       0.74      0.70      0.71       210

-------------------------------------------------------------------------------------
AUC Score : 0.5303677416173689
-------------------------------------------------------------------------------------
```





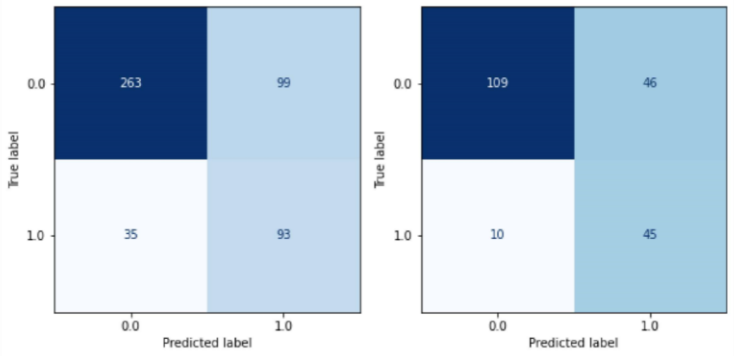
Naive Bayes Classifier

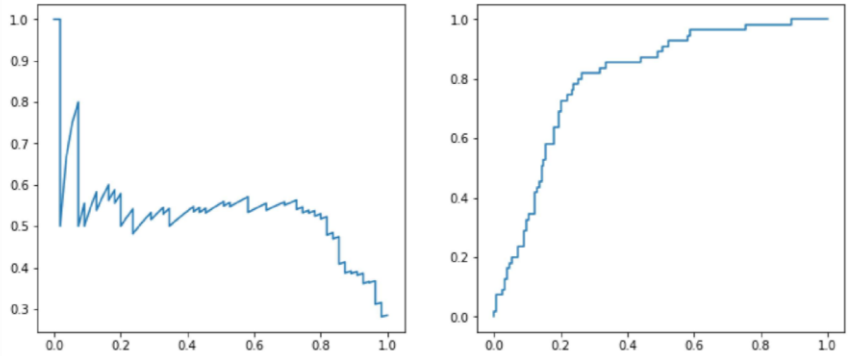
In [27]:
```
cnb_check = ComplementNB()
param_grid = {'alpha' : range(1,10,1), 'norm' : [True, False]}

cnb_CV = GridSearchCV(cnb_check, param_grid, cv=5, scoring='recall')
cnb_CV.fit(X, Y)

print("Optimum Parameters: {}".format(cnb_CV.best_params_))
print("Optimum score     : {}".format(cnb_CV.best_score_))
```

```
Optimum Parameters: {'alpha': 6, 'norm': False}
Optimum score     : 0.737987987987988
```

In [28]:
```
cnb_clf = ComplementNB(alpha=6, norm=False)
cnb_clf.fit(X_train, Y_train)

eval_model(cnb_clf)
plot_model(cnb_clf)
```

```
---------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.88      0.73      0.80       362
         1.0       0.48      0.73      0.58       128

    accuracy                           0.73       490
   macro avg       0.68      0.73      0.69       490
weighted avg       0.78      0.73      0.74       490

---------------------------------Report on Test Data------------------------------
              precision    recall  f1-score   support

         0.0       0.92      0.70      0.80       155
         1.0       0.49      0.82      0.62        55

    accuracy                           0.73       210
   macro avg       0.71      0.76      0.71       210
weighted avg       0.81      0.73      0.75       210

-------------------------------------------------------------------------------------
AUC Score : 0.5288450380085002
-------------------------------------------------------------------------------------
```





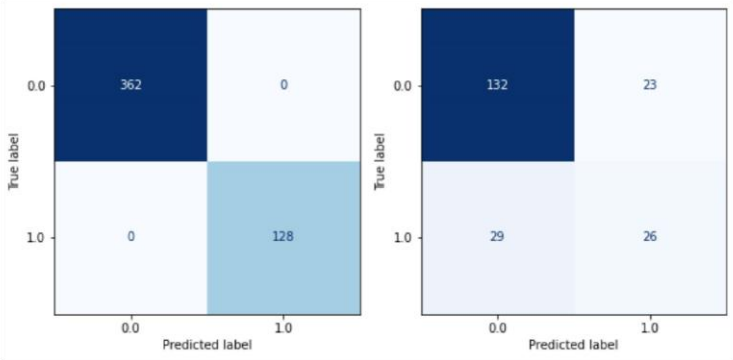
K-Nearest Neighbors Classifier

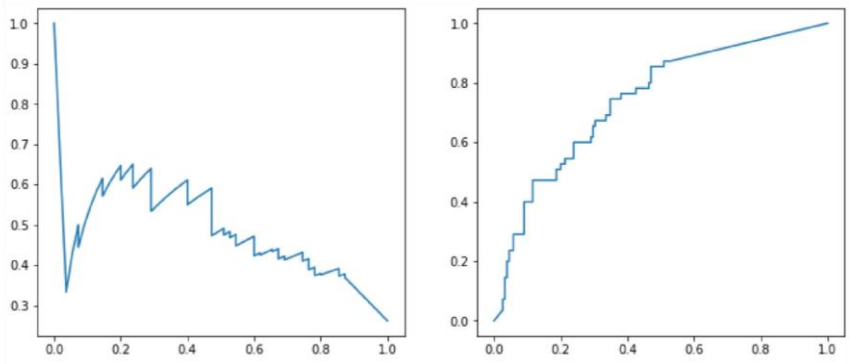
In [29]:
```
knn_check = KNeighborsClassifier()
param_grid = {'n_neighbors' : range(3,7,1), 'weights' : ['uniform', 'distance'], 'p':[1, 2, 3]}

knn_CV = GridSearchCV(knn_check, param_grid, cv=5, scoring='recall')
knn_CV.fit(X, Y)

print("Optimum Parameters: {}".format(knn_CV.best_params_))
print("Optimum score     : {}".format(knn_CV.best_score_))
```

```
Optimum Parameters: {'n_neighbors': 4, 'p': 2, 'weights': 'distance'}
Optimum score     : 0.4321321321321321
```

In [30]:
```
knn_clf = KNeighborsClassifier(n_neighbors=4, p=2, weights='distance')
knn_clf.fit(X_train, Y_train)

eval_model(knn_clf)
plot_model(knn_clf)
```

```
-----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       362
         1.0       1.00      1.00      1.00       128

    accuracy                           1.00       490
   macro avg       1.00      1.00      1.00       490
weighted avg       1.00      1.00      1.00       490

-----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.82      0.85      0.84       155
         1.0       0.53      0.47      0.50        55

    accuracy                           0.75       210
   macro avg       0.68      0.66      0.67       210
weighted avg       0.74      0.75      0.75       210

-------------------------------------------------------------------------------------
AUC Score : 0.48436410692011883
-------------------------------------------------------------------------------------
```





Cross Validation for XGBClassifier

In [31]:
```
dData = xgb.DMatrix(data=X,label=Y)
dTrain = xgb.DMatrix(X_train, label=Y_train)
dTest = xgb.DMatrix(X_test)

params = {"objective":"binary:logistic",'colsample_bytree': 0.5,'learning_rate': 0.1, 'max_depth': 5}
cv_results = xgb.cv(dtrain=dData, params=params, nfold=5, num_boost_round=75, early_stopping_rounds=10, metrics="error",
as_pandas=True, seed=123)

print((cv_results.tail(1)))
```

```
   train-error-mean  train-error-std  test-error-mean  test-error-std
5          0.155714         0.014169         0.231429        0.036027
```

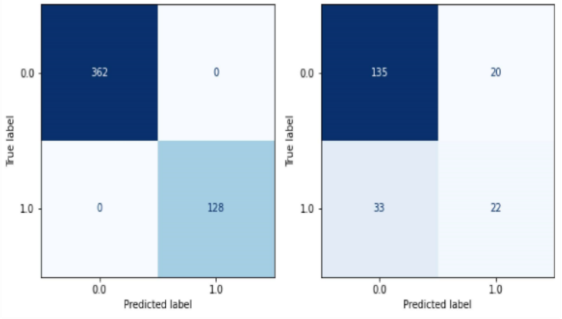
XGBooster Classifier

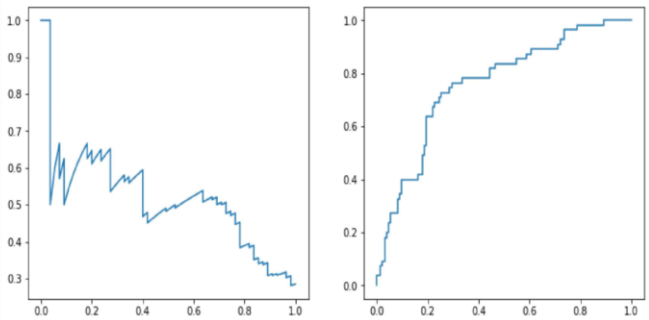
In [32]:
```
xgb_check = xgb.XGBClassifier(use_label_encoder=False, eval_metric = 'error')

param_grid = {'n_estimators': range(100,500,100),  'reg_alpha':np.logspace(-3, 0, 30), 'colsample_bytree': np.arange(0.1
,1,0.2), 'learning_rate': np.arange(0.1, 1, 0.1)}
xgb_cv = RandomizedSearchCV(xgb_check, param_grid, cv=5)
xgb_cv.fit(X,Y)

print("Optimum Parameters: {}".format(xgb_cv.best_params_))
print("Optimum score     : {}".format(xgb_cv.best_score_))
```

```
Optimum Parameters: {'reg_alpha': 0.18873918221350977, 'n_estimators': 300, 'learning_rate': 0.9, 'colsample_bytree': 0.
3000000000000004}
Optimum score     : 0.8014285714285714
```

In [33]:
```
xgb_clf = xgb.XGBClassifier(use_label_encoder=False, eval_metric = 'error', reg_alpha=0.18873918221350977,
                            n_estimators=300, learning_rate=0.9, colsample_bytree=0.3000000000000004)
xgb_clf.fit(X_train, Y_train)

eval_model(xgb_clf)
plot_model(xgb_clf)
```

```
-----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       362
         1.0       1.00      1.00      1.00       128

    accuracy                           1.00       490
   macro avg       1.00      1.00      1.00       490
weighted avg       1.00      1.00      1.00       490

-----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       0.80      0.87      0.84       155
         1.0       0.52      0.40      0.45        55

    accuracy                           0.75       210
   macro avg       0.66      0.64      0.64       210
weighted avg       0.73      0.75      0.74       210

-------------------------------------------------------------------------------------
AUC Score : 0.5150711544704396
-------------------------------------------------------------------------------------
```

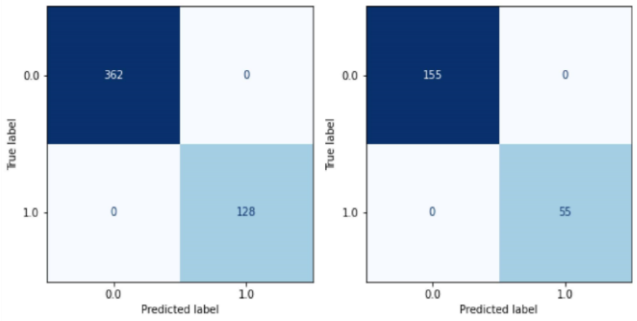


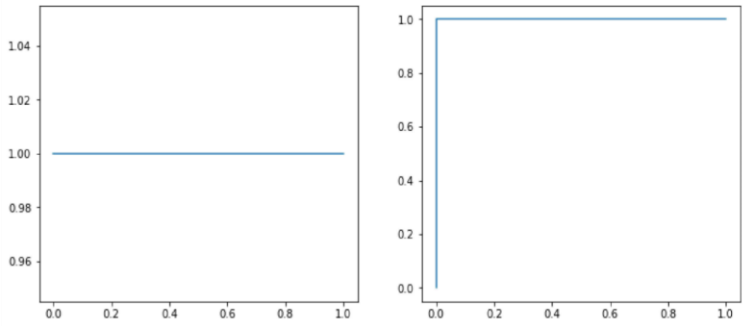


Training entire train data set and predicting values for test dataset

In [34]:
```
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric = 'error', reg_alpha=0.18873918221350977,
                              n_estimators=300, learning_rate=0.9, colsample_bytree=0.30000000000000004)
xgb_model.fit(X, Y)

eval_model(xgb_model)
plot_model(xgb_model)
```

```
-----------------------------------Report on Train Data-----------------------------
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       362
         1.0       1.00      1.00      1.00       128

    accuracy                           1.00       490
   macro avg       1.00      1.00      1.00       490
weighted avg       1.00      1.00      1.00       490

-----------------------------------Report on Test Data-----------------------------
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00       155
         1.0       1.00      1.00      1.00        55

    accuracy                           1.00       210
   macro avg       1.00      1.00      1.00       210
weighted avg       1.00      1.00      1.00       210

-------------------------------------------------------------------------------------
AUC Score : 1.0
-------------------------------------------------------------------------------------
```





Predicing fare amount for test dataset and saving output and final model.

In [35]:
```
Y_Out = xgb_model.predict(dfTest.values)

pd.DataFrame({'default' : Y_Out}).to_csv("loandflt_xgb_output_py.csv")

joblib.dump(xgb_model, 'loandflt_xgbmodel_py.pkl')
```

```
C:\Users\ASUS\anaconda3\lib\site-packages\xgboost\data.py:112: UserWarning: Use subset (sliced data) of np.ndarray is no
t recommended because it will generate extra copies and increase memory consumption
  warnings.warn(
```

Out[35]:
```
['loandflt_xgbmodel_py.pkl']
```

In [ ]:

7 Appendix B

R CODE

```
###--------------clearing environment and setting up directory-------------------###

rm(list = ls())
setwd("K:/Data_Science/Project/Project_02")
getwd()

###------------------- importing libraries ---------------------------------------###

libs = c("ggplot2", "geosphere", "corrgram", "DMwR", "caret", "rpart", "randomForest",
"xgboost","stats", "sp", "pROC", "e1071")

#load Packages
lapply(libs, require, character.only = TRUE)
rm(libs)


###-------------------- loading data ------------------------------------------###

dfLoaner <- read.csv("01_Data/bank-loan.csv", header=TRUE, na.strings = c(" ", "", "NA"))

###----------------------overview of data--------------------------------------###

str(dfLoaner)
#'data.frame':      850 obs. of  9 variables:
#$ age     : int  41 27 40 41 24 41 39 43 24 36 ...
#$ ed      : int  3 1 1 1 2 2 1 1 1 1 ...
#$ employ  : int  17 10 15 15 2 5 20 12 3 0 ...
#$ address : int  12 6 14 14 0 5 9 11 4 13 ...
#$ income  : int  176 31 55 120 28 25 67 38 19 25 ...
#$ debtinc : num  9.3 17.3 5.5 2.9 17.3 10.2 30.6 3.6 24.4 19.7 ...
#$ creddebt: num  11.359 1.362 0.856 2.659 1.787 ...
#$ othdebt : num  5.009 4.001 2.169 0.821 3.057 ...
#$ default : int  1 0 0 0 1 0 0 0 1 0 ...


summary(dfLoaner)

dfLoaner$default <-as.factor(dfLoaner$default)

###-----------------------distribution of individual variables-----------------###
ggplot(dfLoaner, aes(x = age)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = ed)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = employ)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = address)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = income)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = debtinc)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = creddebt)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = othdebt)) + geom_area( stat = 'count')
ggplot(dfLoaner, aes(x = default)) + geom_area( stat = 'count')
```

```
###-----------------------feature tranformation--------------------------------###

#Normalizing Values

preproc_x = preProcess(dfLoaner[,c(1:8)], method = 'range')
dfLoaner[,c(1:8)] <- predict(preproc_x, dfLoaner[,c(1:8)])


###---------------------------------feature selection--------------------------###
#correlation analysis
corrgram(dfLoaner, order = F,
         upper.panel=panel.pie, text.panel=panel.txt, main = "Correlation Plot")


#vif analysis for multicolinearity
vif(dfLoaner)
#Variables      VIF
#1       age 2.022854
#2        ed 1.263713
#3    employ 2.401604
#4   address 1.592315
#5    income 4.215396
#6   debtinc 3.370526
#7  creddebt 2.615818
#8   othdebt 3.861054
#9   default 1.405265
# no vif more than 5

###----------------------------model building-----------------------------------###

#train-test split
dfTrain <- dfLoaner[which(!is.na(dfLoaner$default)),]
dfTest <- dfLoaner[which(is.na(dfLoaner$default)),]
set.seed(1000)
tr.idx = createDataPartition(dfTrain$default,p=0.7,list = FALSE)
to_train <- dfTrain[tr.idx, ]
to_test <- dfTrain[-tr.idx, ]
formula_y <- (default ~ age + ed + employ + address + income + debtinc + creddebt +
othdebt)

#Error Metric defination
err_metric=function(CM)
{
  TN =CM[1,1]
  TP =CM[2,2]
  FP =CM[1,2]
  FN =CM[2,1]
  precision =(TP)/(TP+FP)
  recall_score =(FP)/(FP+TN)
  f1_score=2*((precision*recall_score)/(precision+recall_score))
```

```
  print(paste("Precision value of the model: ",round(precision,2)))
  print(paste("Recall value of the model: ",round(recall_score,2)))
  print(paste("f1 score of the model: ",round(f1_score,2)))
}

#logistic regression
logModel <- glm(formula = formula_y, data=to_train, family=binomial)
summary(logModel)
logPred = predict(logModel,to_test[,-9], type="response")

cnf_mtr <- table(to_test[,9],logPred>0.5)
print(cnf_mtr)
#  FALSE  TRUE
#0   144    11
#1    28    26

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.7"
#[1] "Recall value of the model:  0.07"
#[1] "f1 score of the model:  0.13"

roc_score <- roc(to_test[,9], logPred)
plot(roc_score, main="ROC Curve")

#decision tree
dtreeModel    <-    rpart(formula    =    formula_y,    data=to_train,    method='class',
control=rpart.control(cp=0.01))
summary(dtreeModel)
dtreePred = predict(dtreeModel, to_test[,-9])

cnf_mtr <- table(to_test[,9],dtreePred[,2]>0.5)
print(cnf_mtr)
#  FALSE  TRUE
#0   142    13
#1    30    24

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.65"
#[1] "Recall value of the model:  0.08"
#[1] "f1 score of the model:  0.15"

roc_score <- roc(to_test[,9], dtreePred[,2])
plot(roc_score, main="ROC Curve")

#support vector
svmModel <- svm(formula = formula_y, data=to_train, probability=TRUE)
summary(svmModel)
svmPred = predict(svmModel, to_test[,-9], probability=TRUE)

cnf_mtr <- table(to_test[,9],svmPred)
print(cnf_mtr)
```

```
#  FALSE  TRUE
#0   144    11
#1    32    22

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.67"
#[1] "Recall value of the model:  0.07"
#[1] "f1 score of the model:  0.13"

roc_score <- roc(to_test[,9], attr(svmPred, "probabilities")[,2])
plot(roc_score, main="ROC Curve")

#naive bayes
nbModel <- naiveBayes(formula = formula_y, data=to_train, probability=TRUE)
summary(nbModel)
nbPred = predict(nbModel, to_test[,-9], type="raw")

cnf_mtr <- table(to_test[,9],nbPred[,2]>0.5)
print(cnf_mtr)
#  FALSE  TRUE
#0   147     8
#1    41    13

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.62"
#[1] "Recall value of the model:  0.05"
#[1] "f1 score of the model:  0.1"

roc_score <- roc(to_test[,9], nbPred[,2])
plot(roc_score, main="ROC Curve")

#random forest
rfrstModel <- randomForest(formula = formula_y, data=to_train, importance=TRUE,
ntree=500, nodesize=4)
summary(rfrstModel)
rfrstPred = predict(rfrstModel, to_test[,-9], type="prob")

cnf_mtr <- table(to_test[,9],rfrstPred[,2]>0.5)
print(cnf_mtr)
#  FALSE  TRUE
#0   143    12
#1    26    18

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.7"
#[1] "Recall value of the model:  0.08"
#[1] "f1 score of the model:  0.14"

roc_score <- roc(to_test[,9], rfrstPred[,2])
plot(roc_score, main="ROC Curve")
```

```
#improving accuracy through XGBoost regression
to_train_matrix <- as.matrix(sapply(to_train[-9],as.numeric))
to_test_matrix <- as.matrix(sapply(to_test[-9],as.numeric))


XGBModel <- xgboost(data = to_train_matrix, label = as.matrix(to_train[9]),
objective="binary:logistic", nrounds = 10, colsample_bytree=0.3)
summary(XGBModel)
xgbPred <- predict(XGBModel, to_test_matrix)

cnf_mtr <- table(to_test[,9],xgbPred>0.5)
print(cnf_mtr)
#  FALSE  TRUE
#0   137    18
#1    27    27

err_metric(cnf_mtr)
#[1] "Precision value of the model:  0.6"
#[1] "Recall value of the model:  0.12"
#[1] "f1 score of the model:  0.19"

roc_score <- roc(to_test[,9], xgbPred)
plot(roc_score, main="ROC Curve")

###-----------------------------final model-------------------------------------###
xgb_train <- as.matrix(sapply(dfTrain[-9],as.numeric))
xgb_test <- as.matrix(sapply(dfTest[-9],as.numeric))

finalModel <- xgboost(data = xgb_train, label = as.matrix(dfTrain[9]),
objective="binary:logistic", nrounds = 10, colsample_bytree=0.3)

finalPred <- predict(finalModel, xgb_test)

final_Pred = data.frame("fare_amount" = finalPred>0.5)

#saving predictions in csv file
write.csv(final_Pred,"loandflt_xgb_output_r.csv",row.names = TRUE)

#saving model in dump format
saveRDS(finalModel, "./loandflt_xgbmodel_r")

##############################################################################################
```