Prediction of Loan Application Status

Group Members:

Aditya Som Choudhury
Techno India Main Saltlake
161300110620

Amartya Paul
BP Poddar Institute of Management and Technology
161150110011

Debaleen Das Spandan
Bengal College of Engineering and Technology
171250110041

Palash Mondal
Techno India Saltlake
181300120082

Shouvit Pradhan
Bengal College of Engineering and Technology
171250110093

Contents

Acknowledgement

I take this opportunity to express my profound gratitude and deep regards to my faculty, **Prof. Arnab Chakraborty** for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time shall carry me a long way in the journey of life on which I am about to embark.

I am obliged to my project team members for the valuable information provided by them in their respective fields. I am grateful for their cooperation during the period of my assignment.

Aditya Som Choudhury

Amartya Paul

Debaleen Das Spandan

Palash Mondal

Shouvit Pradhan

Project Objective

In this project we have a shortened 'Predict Loan Application Status' Dataset from Kaggle. In this dataset, the target attribute is the loan status which is in binary. So, in this project we need to do binary classification based on the attributes present in our dataset and predict whether a loan application should be approved or not.

Our objective in this project is to study the given dataset of 'Predict Loan Application Status'. We might need to pre-process the given dataset if we need to. Then, we would train 4 models viz. 'KNN classifier model', 'Naive Bayes classifier model', 'Decision Tree classifier model' and 'Linear Regression classifier model'. After training the aforementioned models, we will need to find out the score, classification report, plot the Receiver Operating Characteristic graph and find out the Area Under Curve (AUC) for each of the models trained. Our next step would be to use the trained models to predict the outcomes using the given test dataset and compare the outcome of each model. We would then choose the best model based on the accuracy score and classification report.

Our methodology for solving the problems in the given project is described below:

- Load the required dataset.
- Study the dataset.
- Describe the dataset.
- Visualise the dataset.
- Find out if the dataset needs to be pre-processed.
    - It will be determined on the basis of whether the dataset has null values or outliers or any such discrepancy that might affect the output of the models to be trained.
- If the dataset is required to be pre-processed, take the necessary steps to pre-process the data.
- Find out the principal attributes for training.
- Split the given dataset for training and testing purpose.
- Fit the previously split train data in the aforementioned 4 models.
- Calculate the accuracy of the 4 models and find out the classification reports.
- Plot the necessary graphs.
- Use each trained model to predict the outcomes of the given test dataset.
- Choose the best model among the 4 trained models bases on the accuracy and classification reports.

Project Scope

The broad scope of 'Predict Loan Approval Status' project is given below:

- The given dataset has attributes based on which the status of the approval of the loan application will be predicted.
- It is a useful project as the Classifier models can be used to quickly determine the loan approval status of large datasets.
- Various banking institution can use these models and modify them according to their needs to use in their loan approval status. This will reduce the manual labour and time spent on determining whether to approve a loan application.
- Customers who intend to take a loan can use these trained models to check whether their loan application will be approved or not. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- The dataset given to us is a shortened form of the original dataset from Kaggle. So, the results might have some mismatch with the real-world applications. But that can be avoided if the models are trained accordingly.

Data Description

Source of the data: Kaggle. The given dataset is a shortened version of the original dataset in Kaggle.

Data Description: The given train dataset has 100 rows and 13 columns.

Columns	Attribute Name	Type	Description	Target Attribute
Application Id	Application_ID	Non-categorical	An Id number for the specific loan application	No
Gender	Gender	Categorical	Gender of the applicant. (M/F)	No
Married	Married	Categorical	Marital status of the applicant. (Yes/No)	No
Dependent	Dependent	Categorical	Number of dependents on the applicant. (0, 1, 2, 3+)	No
Education	Education	Categorical	Education status of the applicant. (Graduate/Not Graduate)	No
Self-Employed	Self_Employed	Categorical	Whether the applicant is self-employed or not. (Yes/No)	No
Applicant Income	ApplicantIncome	Non-categorical	Income amount of the applicant	No
Co-applicant Income	CoapplicantIncome	Non-categorical	Income amount of the co-applicant	No
Loan Amount	LoanAmount	Non-categorical	Amount of loan applied for.	No
Loan Amount Term	Loan_Amount_Term	Non-categorical	Term period for applied loan amount	No
Credit History	Credit_History	Categorical	Credit history of the applicant. (1/0)	No
Property Area	Property_Area	Categorical	Location of property of the applicant. (Urban/Semiurban/Rural)	No
Loan Status	Loan_Status	Categorical	Loan approval status of the loan application. (Y/N)	Yes

Table 1: Data Description

The following table shows the 5 number statistics of the given dataset:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term
Mean	4122.83000	1700.550000	134.600000	341.130000
Standard deviation	2258.89434	1947.668891	62.103856	61.022211
Minimum	1000.00000	0.000000	17.000000	60.000000
25%	2636.00000	0.000000	100.000000	360.000000
50%	3598.00000	1558.500000	120.000000	360.000000
75%	4710.00000	2394.500000	152.750000	360.000000
Maximum	12841.00000	10968.000000	349.000000	480.000000

Table 2: 5 number statistics of the given dataset

Now we will pre-process the data. The methodology followed is given below:

- Checking for null values.
    - If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- Checking for outliers.
    - If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

Data Pre-processing

As the given dataset had Categorical and Non-categorical data mixed, we converted the categorical data into non-categorical data accordingly. We converted the binary categories into 0 and 1. We converted the other categorical attributes into suitable numerical values. The following table shows the conversion record:

Non-Numeric to Numeric Change table		
Column	Initial Value	Replaced Value
Married	No	0
	Yes	1
Gender	M	0
	F	1
Education	Not Graduate	0
	Graduate	1
Self-Employed	No	0
	Yes	1
Loan Status	N	0
	Y	1
Dependents	'0'	0
	'1'	1
	'2'	2
	'3+'	3
Property Area	Rural	0
	Semiurban	1
	Urban	2

Table 3: Categorical to Numerical change in values

We searched for null values in our dataset and formed the following table:

Column Name	Count of Null Value
Application Id	0
Gender	1
Married	0
Dependents	0
Education	0
Self-Employed	6
Applicant Income	0
Co-applicant Income	0
Loan Amount	5
Loan Amount Term	5
Credit History	8
Property Area	0
Loan Status	0

Table 4: Count of Null values

To visualise the null values, we made a heatmap plot using seaborn library function heatmap. The heatmap plot is given below:

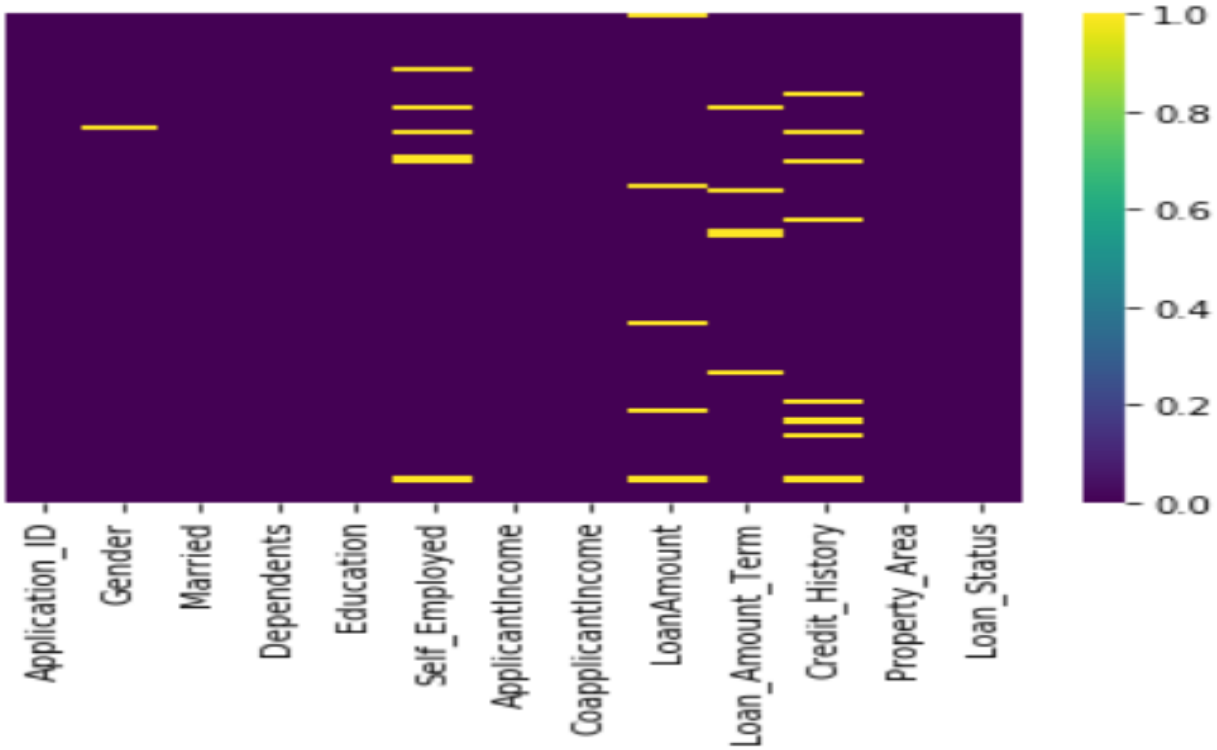


Fig 1: Heatmap of the given Data

The heatmap shows that the dataset has null values

To remove the null values, we had the following methodology:

- filling Null values in 'Credit_History' with 0 (as stated in the dataset description from source)
- filling Null values in binary columns with mode value. This was applied in columns ‘Self_Employed’ and ‘Gender’.
- filling Null values with mean +- std in non-binary columns. This was applied in column ‘LoanAmount’ and “Loan_Amount_Term’.

After removing the null values, the following heatmap was obtained:

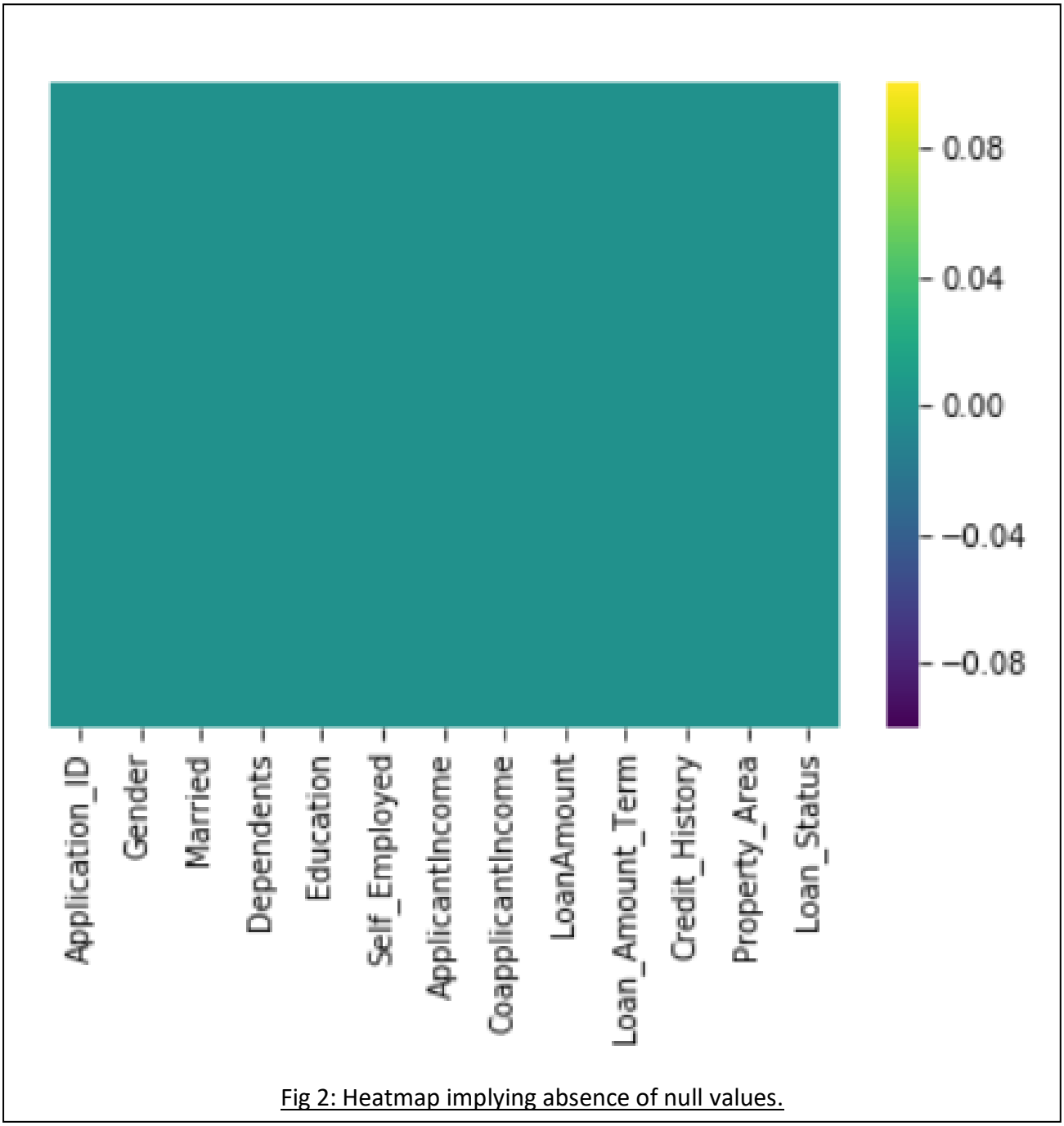


Fig 2: Heatmap implying absence of null values.

Now we have successfully handled Null values and converted non-numeric values to Numeric values. We didn't drop the rows with null values as we have a small dataset (only 100 entries).

So, we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'Loan_Status' column in the dataset.

The following table gives the correlation value of each attribute with our target attribute i.e. 'Loan_Status':

Columns	**Correlation value**
Gender	-0.09335200560186736
Education	0.013861413315217981
Self_Employed	0.04212465653604544
ApplicantIncome	-0.1196673096146476
CoapplicantIncome	-0.03110315854904195
LoanAmount	-0.18017272662780226
Loan_Amount_Term	0.11608628153827855
Credit_History	0.5306941097826289
Property_Area	0.24889474150088223
Married	0.12792979689405484
Dependents	0.0057840889995565475

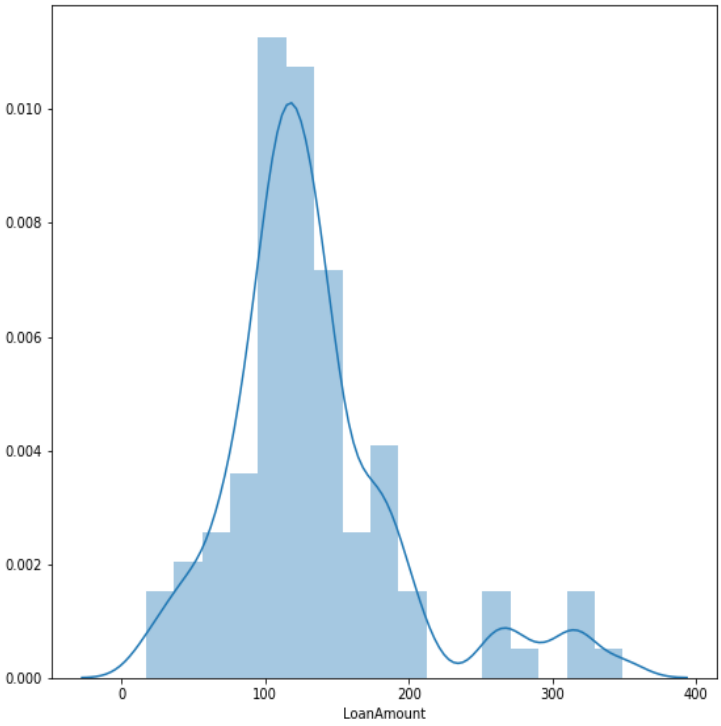
Table 5: Correlation values with Target Attribute

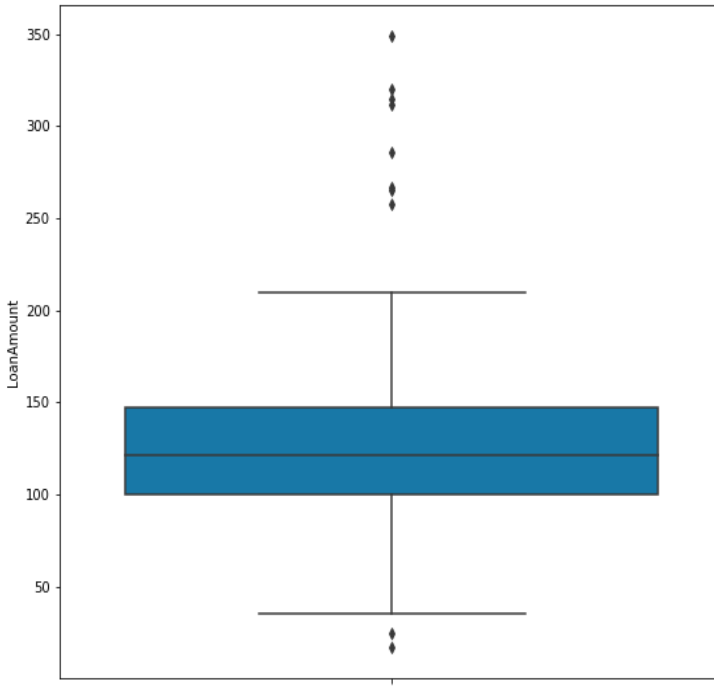
Outliers are extreme values that deviate from other observations on data, they may indicate a variability in a measurement, experimental errors or a novelty. In other words, an outlier is an observation that diverges from an overall pattern on a sample.

Most common causes of outliers on a data set:

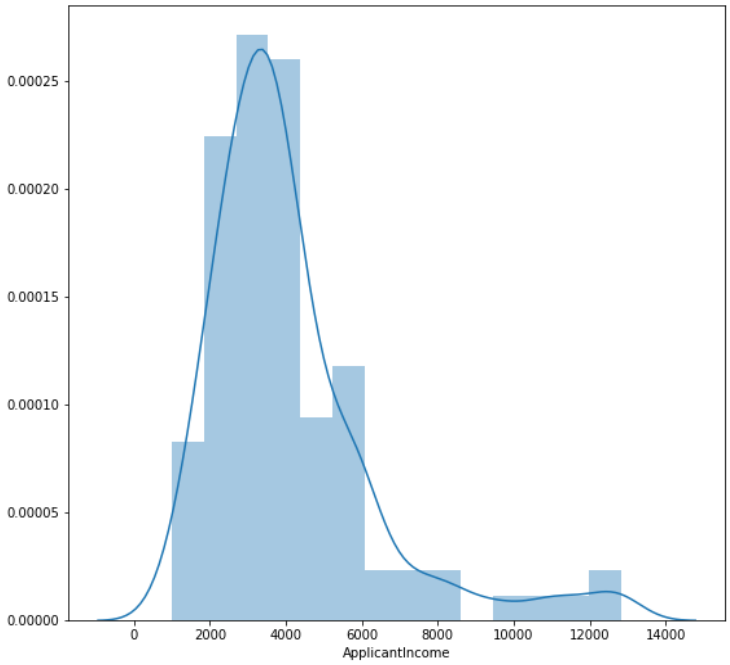
- Data entry errors (human errors)
- Measurement errors (instrument errors)
- Experimental errors (data extraction or experiment planning/executing errors)
- Intentional (dummy outliers made to test detection methods)
- Data processing errors (data manipulation or data set unintended mutations)
- Sampling errors (extracting or mixing data from wrong or various sources)
- Natural (not an error, novelties in data)

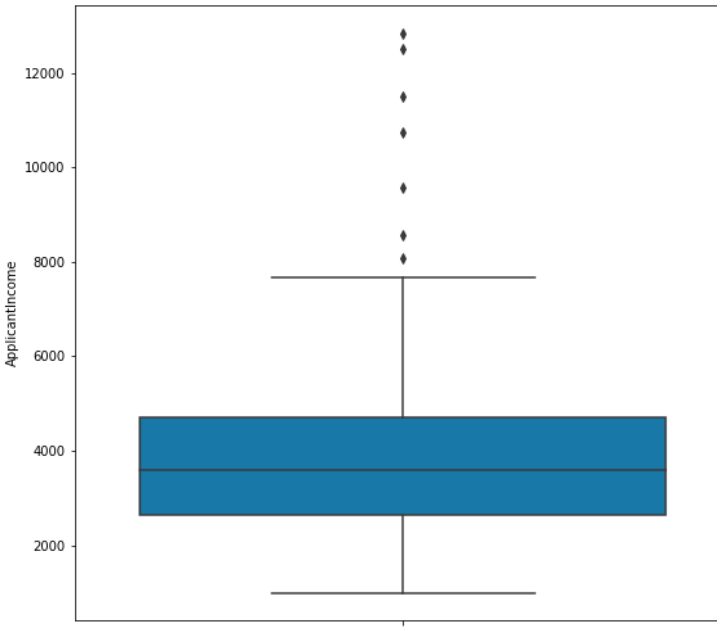
We plot distribution graph and boxplot to visualise the outliers. The plots are given below:



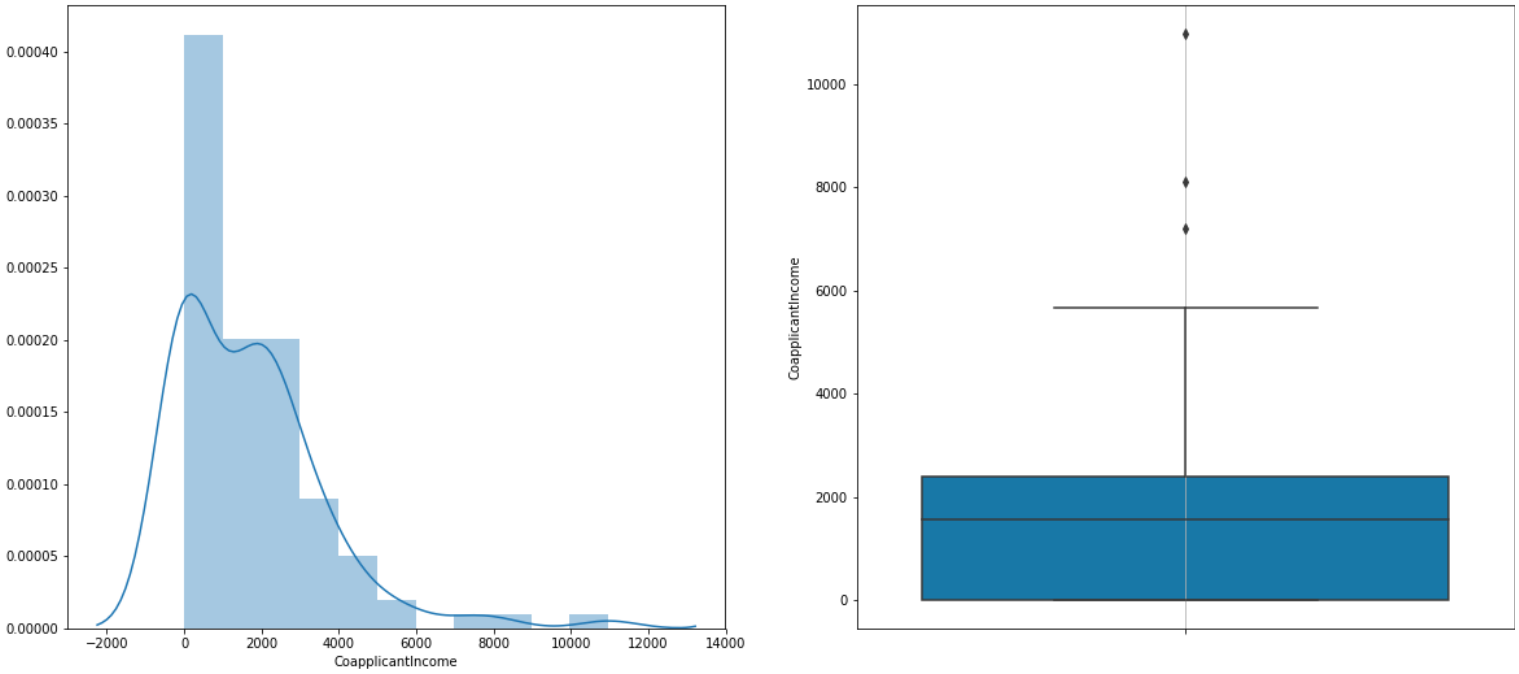


Distribution plot and boxplot of Loan Amount showing possible outliers





Distribution plot and boxplot of Applicant Income showing possible outliers



Distribution plot and boxplot of Co-applicant Income showing possible outliers

Fig 3: Distribution plot and boxplot of 3 attributes viz.

'LoanAmount', 'ApplicantIncome' and 'CoapplicantIncome'

From the distribution plots and boxplots, we can see that there are possible outliers in the columns 'LoanAmount', 'ApplicantIncome' and 'CoapplicantIncome'.

We have not checked for outliers in the columns viz. 'Gender', 'Married', 'Education', 'Self_Employed', and 'Credit_History' as they are columns having values in binaries and also initially, they had non-numeric values. We have not checked for outliers in the columns viz. 'Dependents' and 'Property_Area' as they are columns having values that were initially non-numeric. We have not checked for outliers in the column 'Loan_Amount_Term' as the mean and MAD values of this column are in the range 3*(mean or MAD). So, this method won't have any effect on the data of the column 'Loan_Amount_Term'.

Now, we had to handle the outliers. We handled the outliers using non-parametric statistics method. We used Median Absolute Deviation or MAD to handle the outliers. In this method, data outside the range 3 * mad is excluded and replaced with median of the data.

After handling the data, we replot the same distribution and boxplot but this time using the modified data. We obtain the following plots:

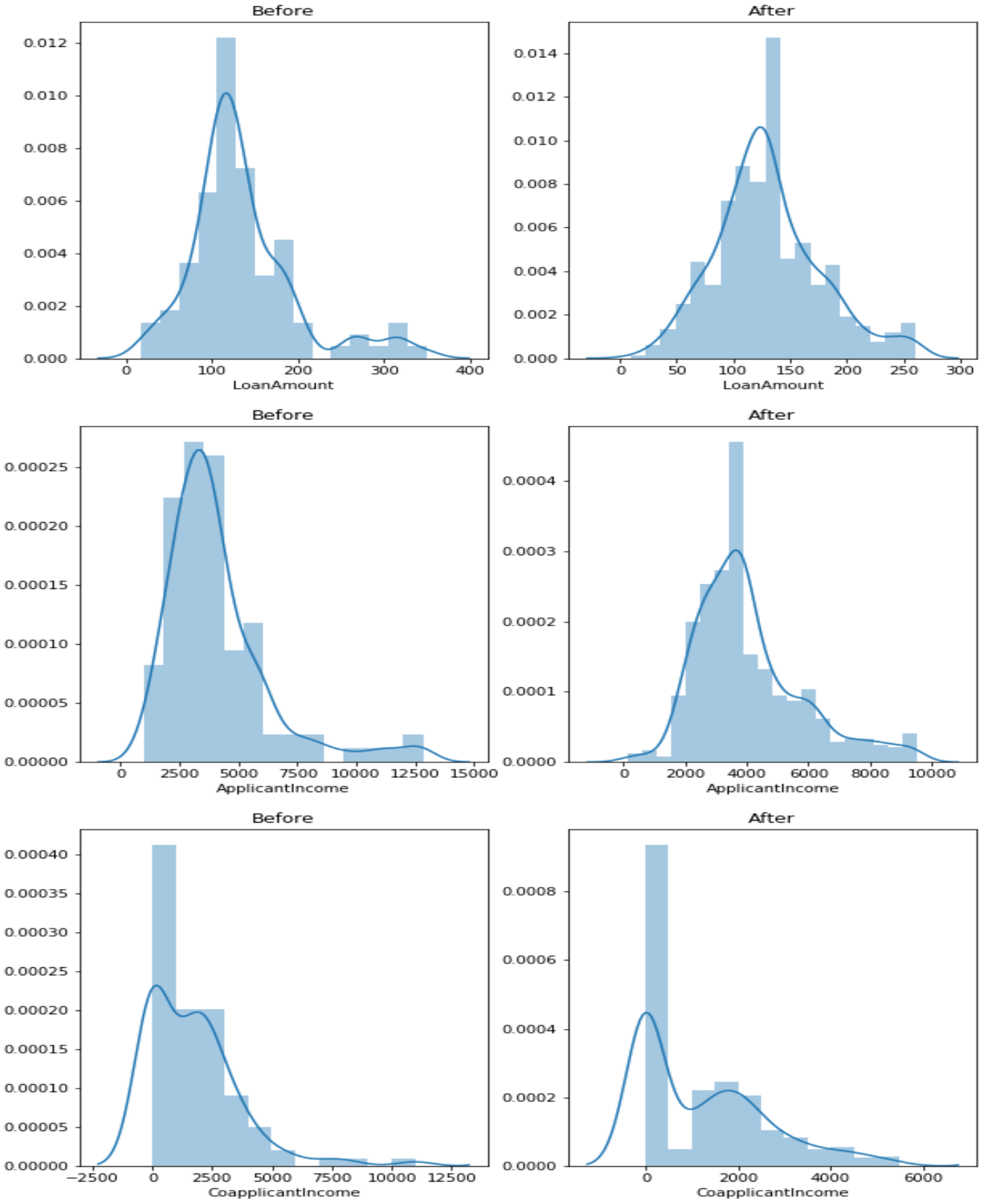


Fig 4: Comparison between distribution plot and boxplot of 3 attributes before and after handling outliers

The following plots show the comparison of the 3 attributes viz. 'LoanAmount', 'ApplicantIncome' and 'CoapplicantIncome' before and after handling outliers:

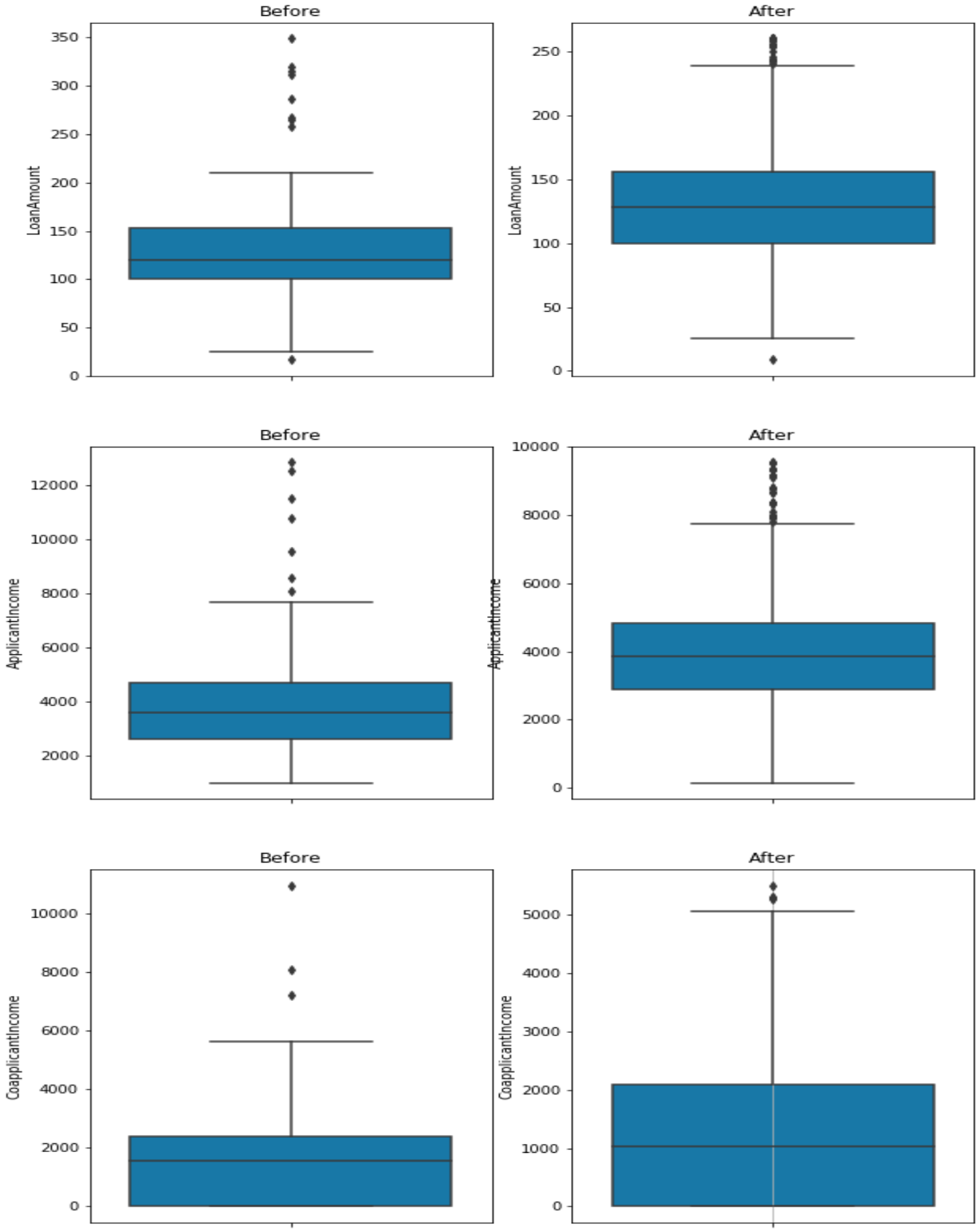
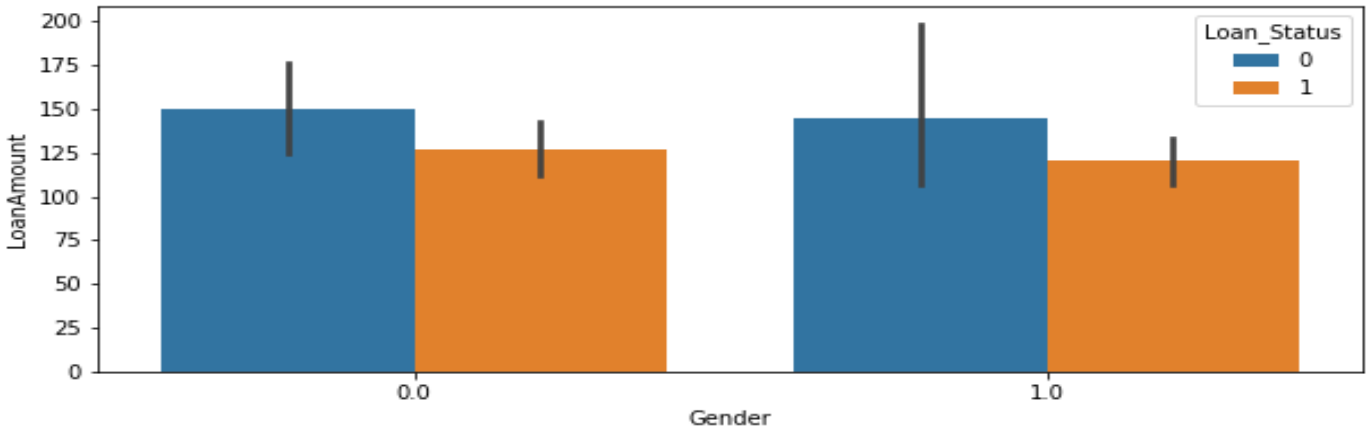
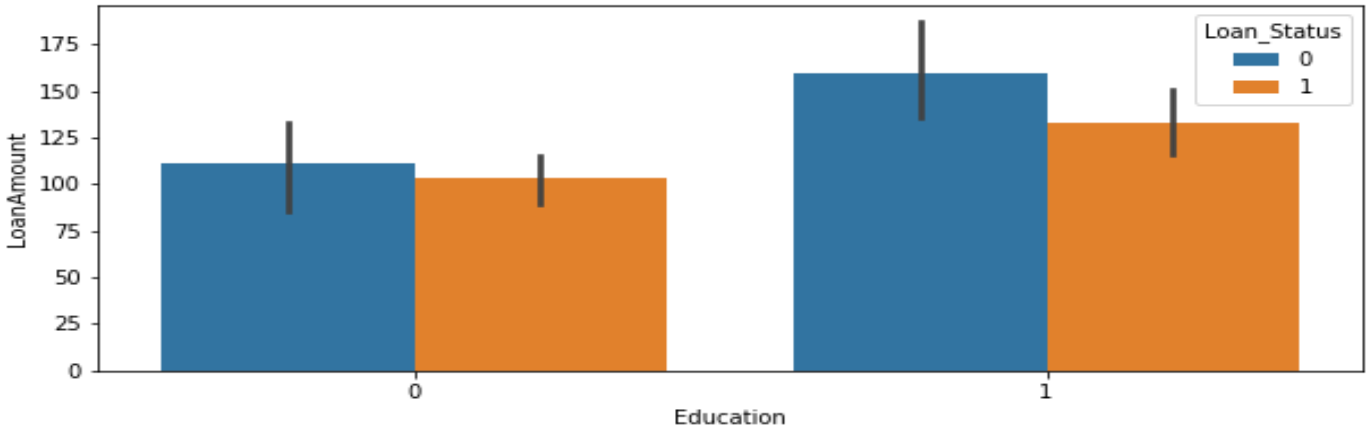


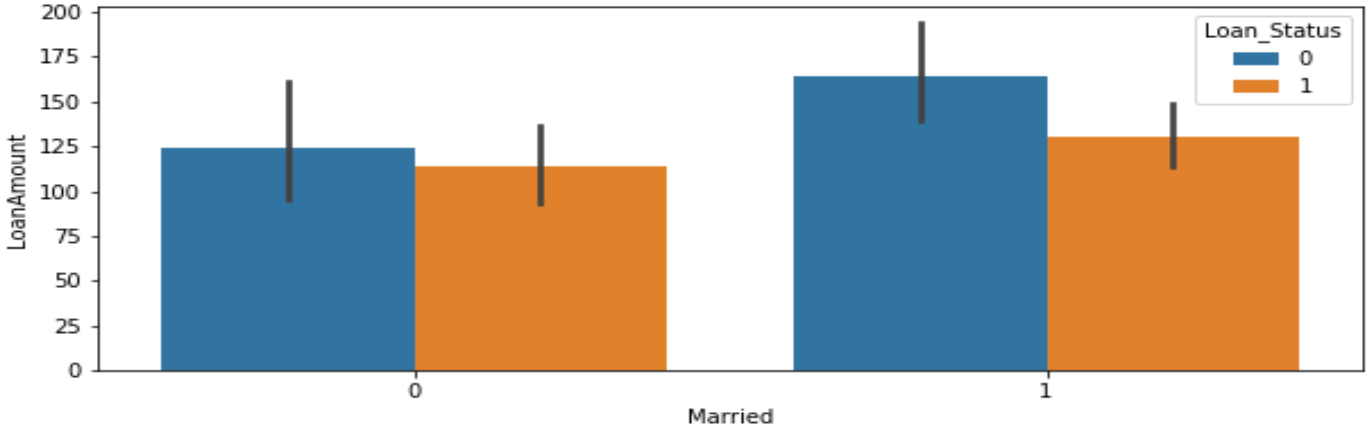
Fig 5: Comparison boxplot of 3 attributes

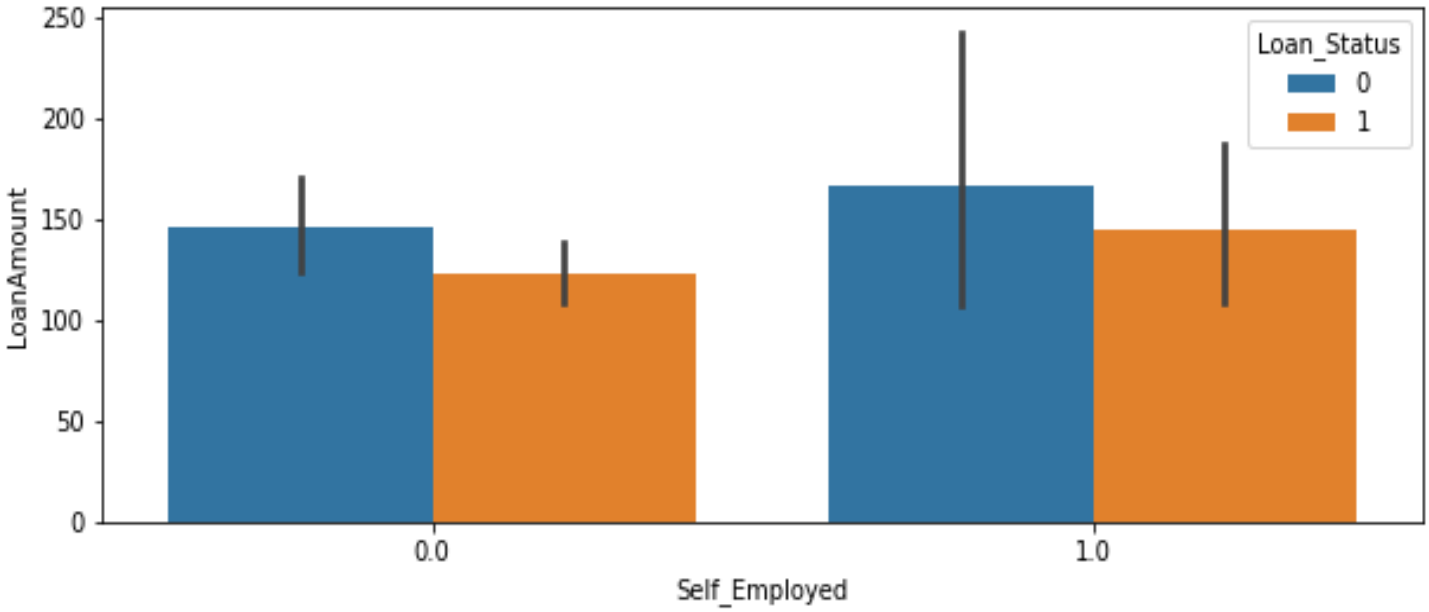
From the distribution plots and boxplots shown above, we can see that a lot of outliers have been handled. Now we will use this data to train a Random Forest Classifier model. We will be training a Random Forest Classifier model to determine the principal attributes in the dataset.

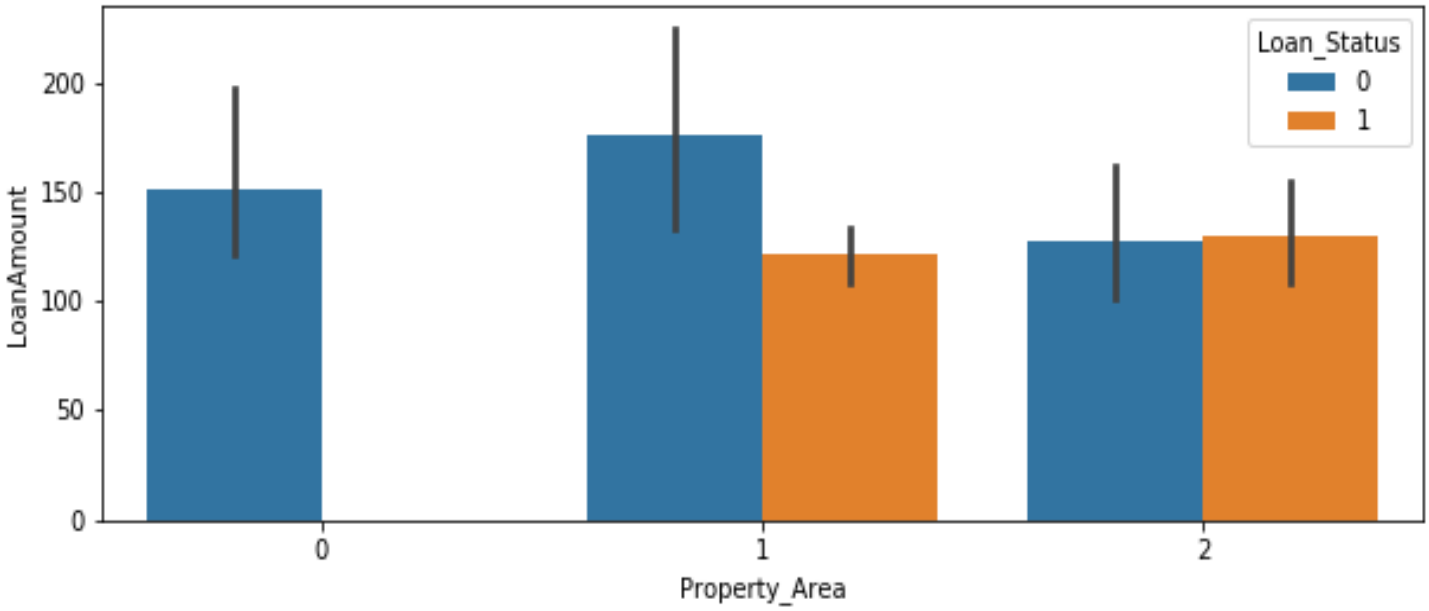
We will now visualise other attributes with respect to our target column i.e. 'Loan_Status'. We have used bar plot to visualise other attributes against the data in 'LoanAmount' column and using our target attribute i.e. 'Loan_Status' as hue.

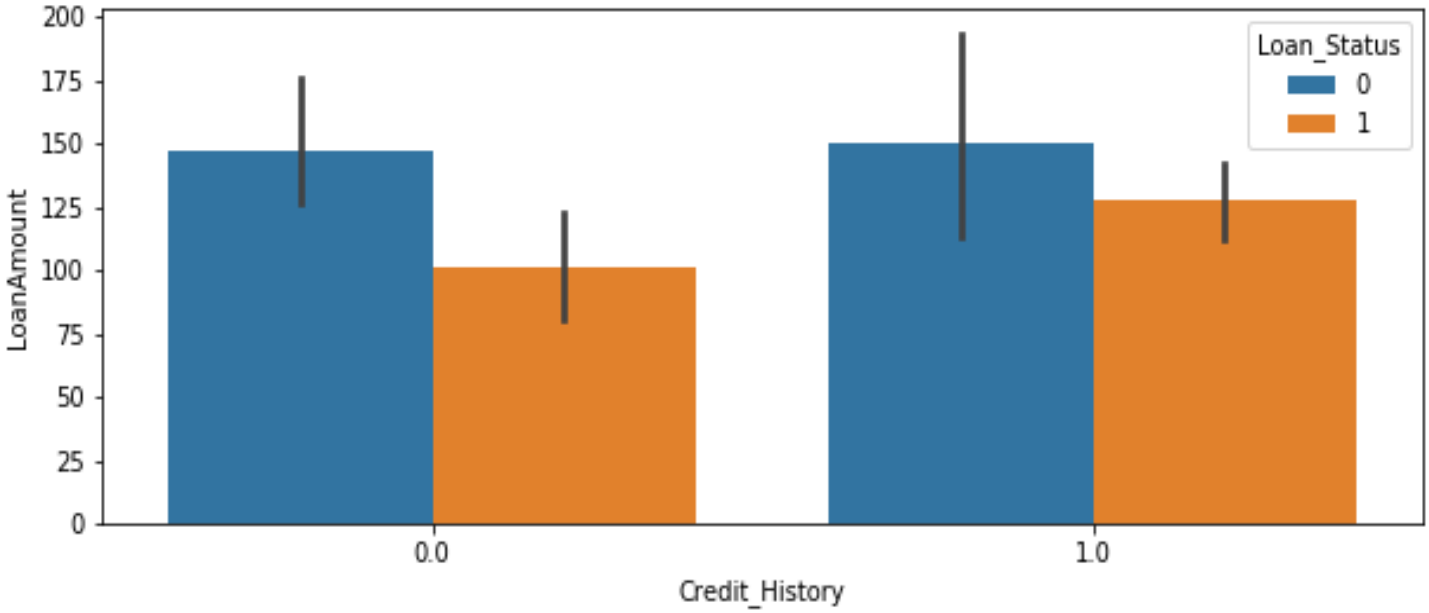







Loan_Status
0
1
LoanAmount
250
200
150
100
50
0
0.0
1.0
Self_Employed


Loan_Status
0
1
LoanAmount
200
150
100
50
0
0
1
2
Property_Area


Loan_Status
0
1
LoanAmount
200
175
150
125
100
75
50
25
0
0.0
1.0
Credit_History

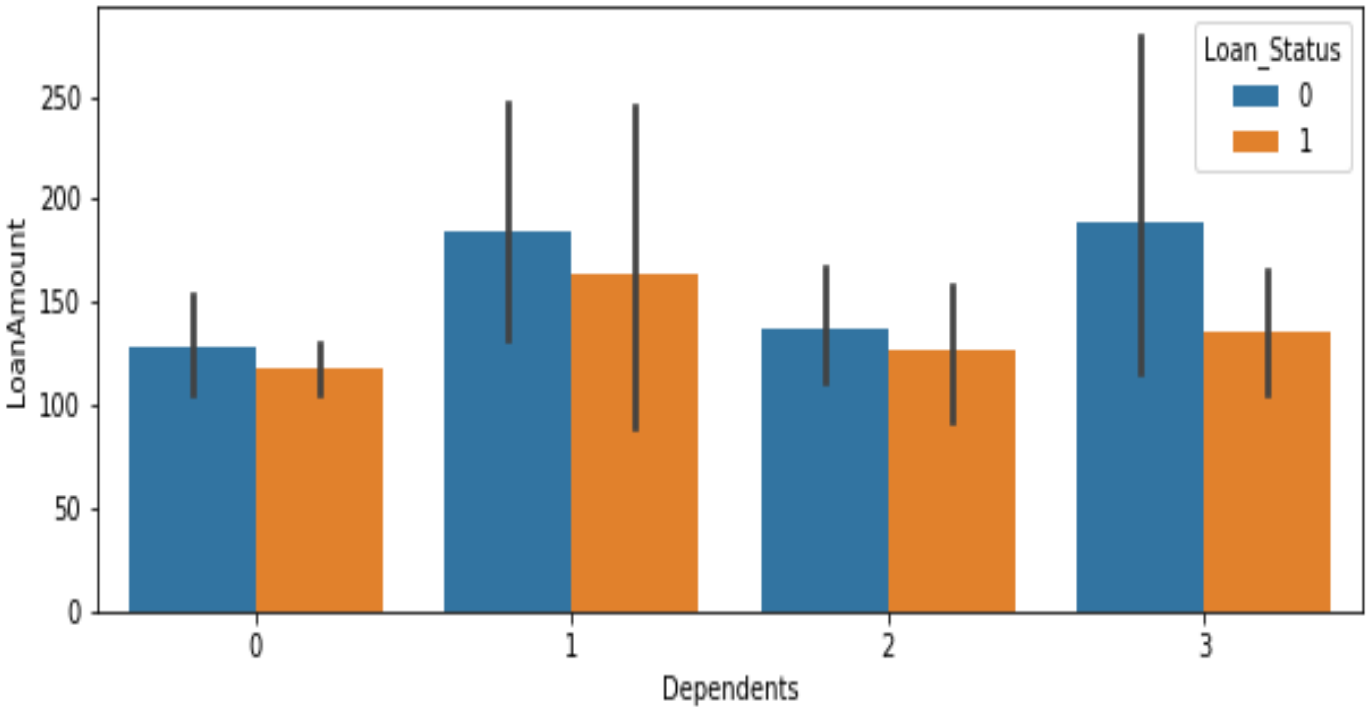


Fig 6: Visualising other attributes

From the above graphs one interesting thing that can be noted is that no loans are approved i.e. 'Loan_Status' = 1 for 'Property_Area' value 'Rural' i.e. 'Property_Area' = 0. All the loan requests for 'Property_Area' value 'Rural' i.e. 'Property_Area' = 0 are denied. We will now train a Random Forest Classifier Model.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement unless specified otherwise.

We have used Random Forest Classifier to determine the principal attributes as this classifier has a built-in feature importance function.

Model Building

Splitting data for training and testing purpose

We split the given train dataset into two parts for training and testing purpose. The split ratio we used is 0.75 which indicates we used 75% data for training purpose and 25% data for testing purpose. We will be using the same split ratio for all the models trained.

Random Forest Classifier Model

The object description of the Random Forest Classifier used is given below:

Object Name	RandomForestClassifier
Parameters	**Value**
bootstrap	True
class_weight	None
criterion	'gini'
max_depth	None
max_features	'auto'
max_leaf_nodes	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
n_estimators	100
n_jobs	2
oob_score	False
random_state	0
verbose	1
warm_start	False

Table 6: Object Parameter Table for Random Forest Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Loan_Status	**0**	**1**
Actual Loan_Status		
0	7	2
1	2	7

Table 7: Confusion matrix of Random Forest Classifier

We trained the Random Forest Classifier Model although it was not our goal, to find out the feature importance of the given attributes and choose the principal attributes for training our goal models. Random Forest Classifier has an inbuilt feature importance function that gives the value of importance of each attribute and helps in determining the target attribute.

The following table consists of the feature importance obtained from our Random Forest Classifier:

Attribute	Feature Importance
'Gender'	0.017472373045798174
'Married'	0.032310146537087026
'Dependents'	0.06274092539838881
'Education'	0.01727880655895561
'Self_Employed'	0.02014039877295202
'ApplicantIncome'	0.1746432652177679
'CoapplicantIncome'	0.11756790122516193
'LoanAmount'	0.12842666389187288
'Loan_Amount_Term'	0.060373745068559366
'Credit_History'	0.22613376236476954
‘Property_Area'	0.14291201191868688

Table 8: Feature importance table obtained from Random Forest Classifier

As we can see from the calculated feature importance values, the attributes viz. 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Credit_History' and 'Property_Area' have higher values compared to other attributes. So, we will select these 5 attributes as our principal attribute to train our required models.

One interesting thing to notice here is that although 'Married' attribute had a high correlation value with our target attribute, 'Loan_Status', the feature importance of 'Married' attribute is lower.

The following table shows the selected principle attributes and their respective feature importance:

Attribute	Feature Importance
'ApplicantIncome'	0.1746432652177679
'CoapplicantIncome'	0.11756790122516193
'LoanAmount'	0.12842666389187288
'Credit_History'	0.22613376236476954
‘Property_Area'	0.14291201191868688

Table 9: Feature importance of selected principal attributes

NOTE: These values will change with each run. But these 5 attributes will always have the highest feature importance if trained as above.

Now we will modify the dataset to include only the selected principal attributes. We made a copy of the initial dataset and then dropped the non-principal attributes viz. ‘Gender’, ‘Married’, 'Dependents', 'Self_Employed', 'Loan_Amount_Term'. Now we obtained a final dataset which we will use for training purpose of our required models.

Given below is the description of the final dataset obtained:

Please Note, we have not included our target attribute 'Loan_Status' in the data description.

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
Type	Non-categorical	Non-categorical	Non-categorical	Categorical	Categorical
unique	NaN	NaN	NaN	2	3
Mean	3636.720000	1606.455000	120.480000	NaN	NaN
Standard deviation	1325.607679	1707.964583	38.476145	NaN	NaN
Minimum	1000.000000	0.000000	17.000000	NaN	NaN
25%	2636.00000	0.000000	100.000000	NaN	NaN
50%	3597.000000	1542.250000	120.000000	NaN	NaN
75%	4197.500000	2341.500000	138.000000	NaN	NaN
Maximum	7660.000000	8106.000000	210.000000	NaN	NaN

Table 10: 5 Description of the final data obtained

Now we will be training our required models. Our project goal requires us to train specific 4 classifier models viz.

1. KNN classifier

2. Naive Bayes Classifier

3. Decision Tree Classifier

and

4. Regression (As the target attribute is binary, we will be using Logistic Regression Classifier)

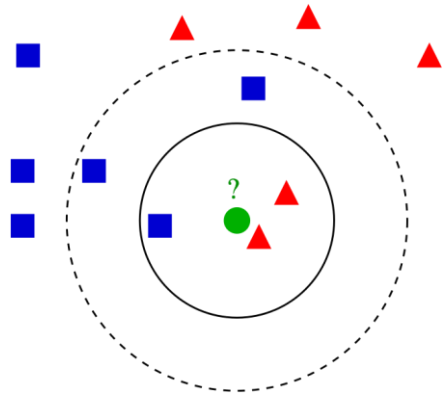
We will be using the final dataset obtained after pre-processing the given train dataset to train our required models.

KNN Classifier

k-NN can be used for both classification and regression predictive problems. However, it is more widely used in classification problems in the industry. In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). For e.g. if k = 1, then the object is simply assigned to the class of that single nearest neighbor. k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms. The neighbors are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

An example of k-NN classification:



Example of k-NN classification. The test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

We used a GridSearchCV object to find the best optimum value of k. Our test results gave the value of k = 5. The object description of the k-NN Classifier used is given below:

Object Name	KNeighborsClassifier
Parameters	**Value**
algorithm	'auto'
leaf_size	30
metric	'minkowski'
metric_params	None
n_neighbors	3
p	2
weights	'uniform'

Table 11: Object Parameter Table for k-NN Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Loan_Status	**0**	**1**
Actual Loan_Status		
0	1	8
1	2	13

Table 12: Confusion matrix of k-NN Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained k-NN model. The ROC curve is given below:

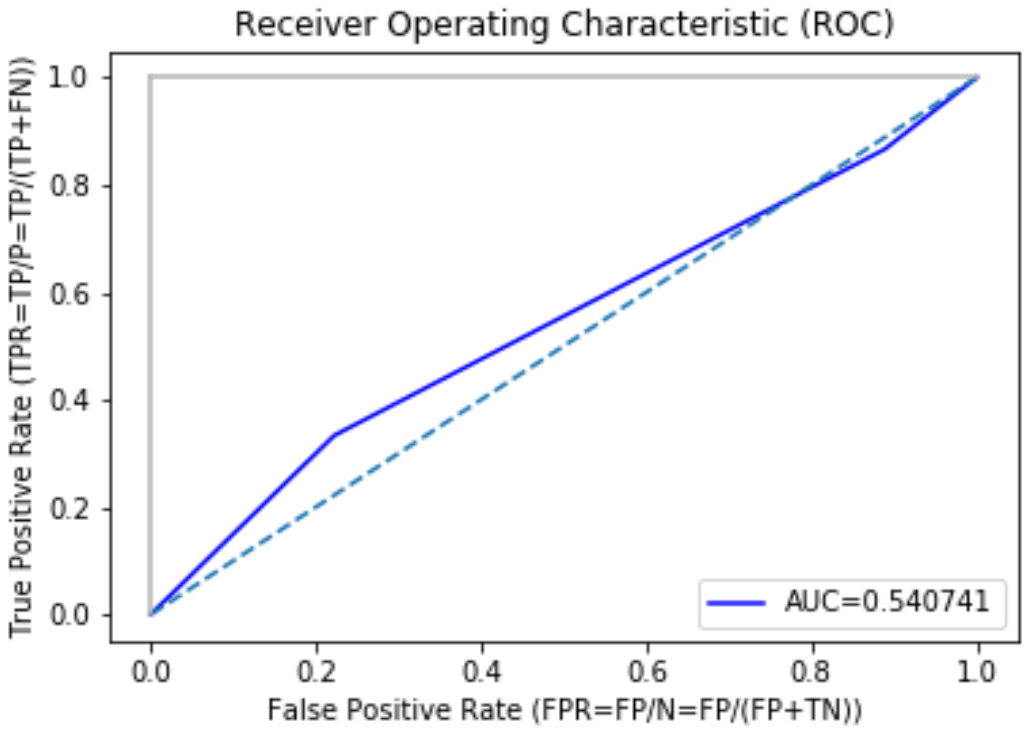


Fig 7: ROC curve of k-NN classifier model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our KNN classifier model is 0.540741

AUC value of KNN = 0.540741

Now we will be preparing the classification report of our k-NN model.

Classification Report of k-NN classifier model				
Accuracy			0.625	
	precision	**recall**	**f1-score**	**support**
0	0.33	0.11	0.17	9
1	0.62	0.87	0.72	15

Table 13: Classification Report table of k-NN classifier

Naive Bayes Classifier

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem. The fundamental Naive Bayes assumption is that each feature makes an:

- independent
- equal

contribution to the outcome. The assumptions made by Naive Bayes are not generally correct in real-world situations. In-fact, the independence assumption is never correct but often works well in practice.

Bayes Theorem:

Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. Bayes' theorem is stated mathematically as the following equation:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

where A and B are events and P(B) is the probability of occurrence of event B.

Basically, we are trying to find probability of event A, given the event B is true. Event B is also termed as evidence.

P(A) is the priori of A (the prior probability, i.e. Probability of event before evidence is seen). The evidence is an attribute value of an unknown instance (here, it is event B).

P(A|B) is a posteriori probability of B, i.e. probability of event after evidence is seen.

The class-data relation from the Bayes Theorem can be obtained as follows:

$$P(Class|Data) = \frac{P(Class)P(Data|Class)}{P(Data)}$$

Where,

- P(Class|Data) = Posterior
- P(Class) = Prior
- P(Data|Class) = Likelihood
- P(Data) = Marginal Probability

In other words, it can be written as:

$$Posterior = \frac{Prior * Likelihood}{Marginal\ Probability}$$

In application, we do not need to calculate the Marginal Probability for classification. We only need to calculate the numerator of the posterior for classification.

Types of Naive Bayes Classifier:

Multinomial Naive Bayes:

This is mostly used for document classification problem, i.e. whether a document belongs to the category of sports, politics, technology etc. The features/predictors used by the classifier are the frequency of the words present in the document.

Bernoulli Naive Bayes:

This is similar to the multinomial Naive Bayes but the predictors are Boolean variables. The parameters that we use to predict the class variable take up only values yes or no, for example if a word occurs in the text or not.

Gaussian Naive Bayes:

When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution.

The object description of the Naive Bayes used is given below:

Object Name	GaussianNB
Parameters	**Value**
priors	None
var_smoothing	1e-09

Table 14: Object Parameter Table for Gaussian Naive Bayes Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Loan_Status	**0**	**1**
Actual Loan_Status		
0	7	2
1	0	15

Table 15: Confusion matrix of Gaussian Naive Bayes Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Gaussian Naive Bayes model. The ROC curve is given below:

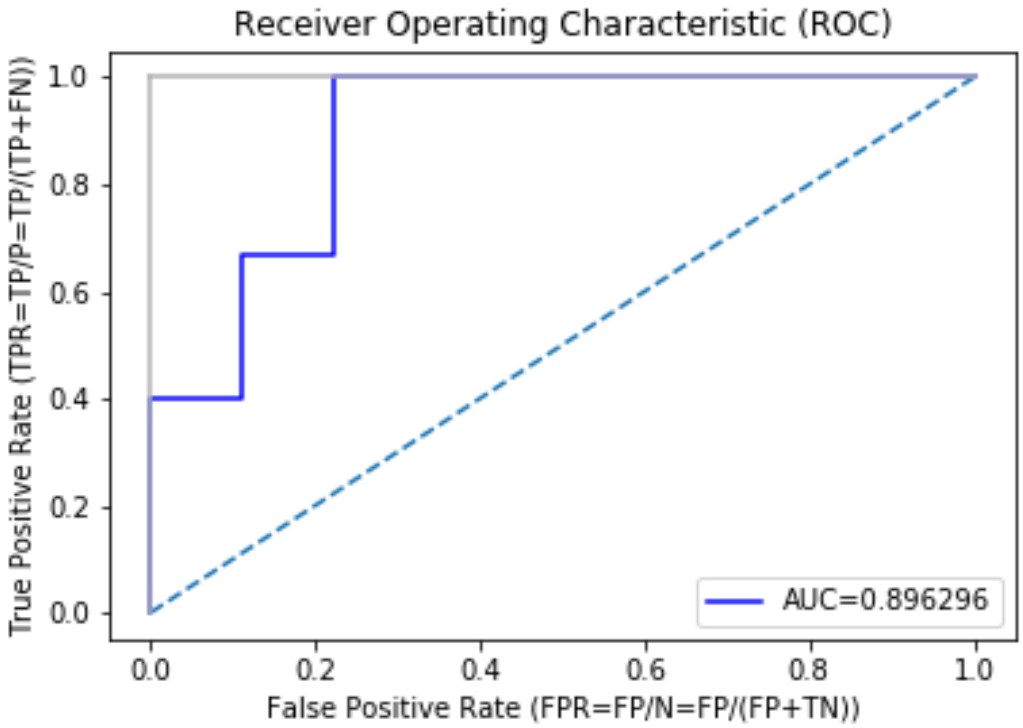


Fig 8: ROC curve of Gaussian Naive Bayes classifier model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Gaussian Naive Bayes model is 0.896296.

AUC value of Gaussian Naive Bayes = 0.896296.

Now we will be preparing the classification report of our Gaussian Naive Bayes model.

Classification Report of Gaussian Naive Bayes model				
Accuracy			0.91667	
	precision	**recall**	**f1-score**	**support**
0	1.0	0.78	0.88	9
1	0.88	1.0	0.94	15

Table 16: Classification Report table of Gaussian Naive Bayes classifier

Decision Tree

Decision tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions. Though a commonly used tool in data mining for deriving a strategy to reach a particular goal, it's also widely used in machine learning.

The object description of the Decision Tree Classifier used is given below:

Object Name	DecisionTreeClassifier
Parameters	**Value**
class_weight	None
criterion	'gini'
max_depth	None
max_features	None
max_leaf_nodes	None
min_impurity_decrease	0.0
min_impurity_split	None
min_samples_leaf	1
min_samples_split	2
min_weight_fraction_leaf	0.0
presort	False
random_state	None
splitter	'best'

Table 17: Object Parameter Table for Decision Tree Classifier

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Loan_Status	**0**	**1**
Actual Loan_Status		
0	6	3
1	2	13

Table 18: Confusion matrix of Decision Tree Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Decision Tree model. The ROC curve is given below:

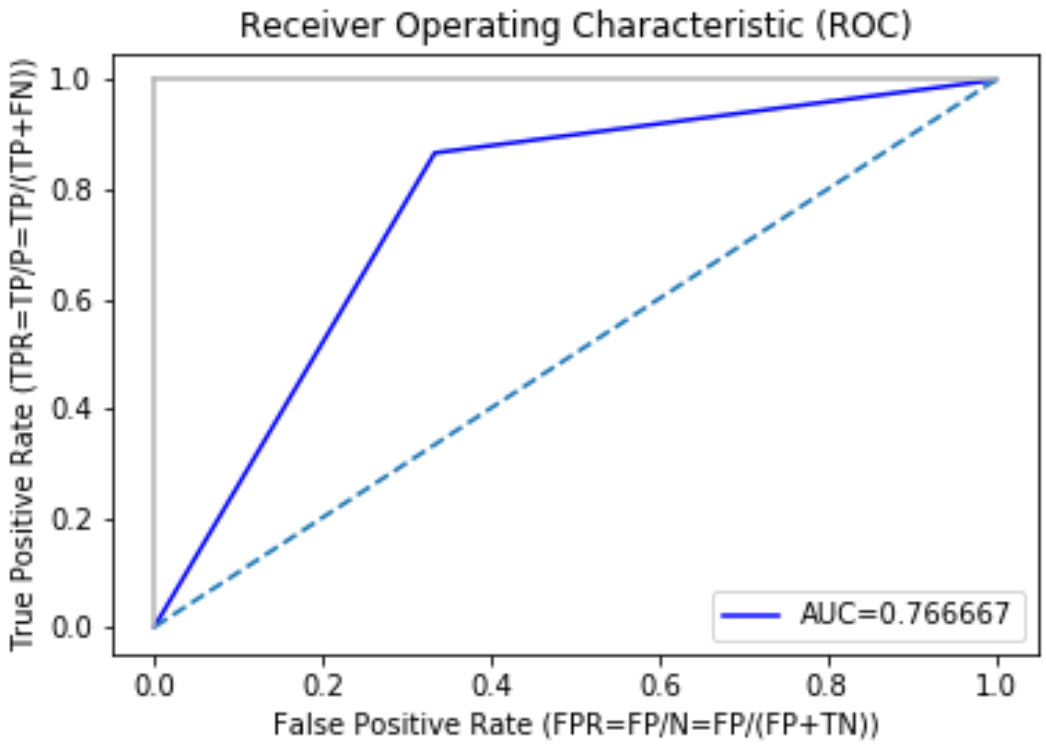


Fig 9: ROC curve of Decision Tree model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Decision Tree classifier model is 0.766667.

AUC value of Decision Tree = 0.766667.

Now we will be preparing the classification report of our Decision Tree

Classification Report of Decision Tree classifier model				
Accuracy			0.7916666666666666	
	precision	**recall**	**f1-score**	**support**
0	0.75	0.67	0.71	9
1	0.81	0.87	0.84	15

Table 19: Classification Report table of Decision Tree classifier

The decision tree prepared by the trained model is given in the next page.

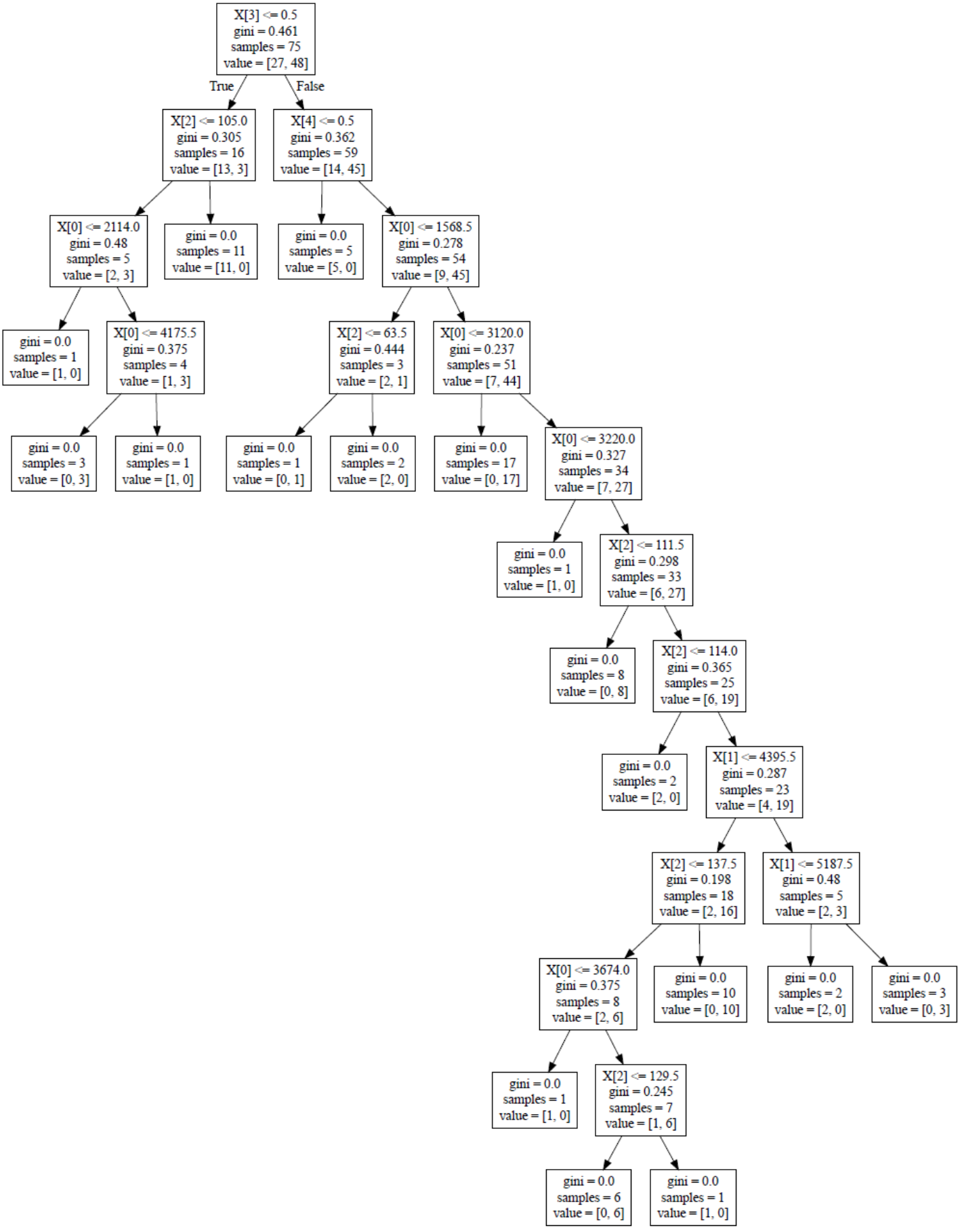


Fig 10: Decision Tree

Logistic Regression

Logistic Regression (also called Logit Regression) is commonly used to estimate the probability that an instance belongs to a particular class (e.g., what is the probability that this email is spam?). If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class (called the positive class, labelled "1"), or else it predicts that it does not (i.e., it belongs to the negative class, labelled "0"). This makes it a binary classifier.

Binary logistic regression major assumptions:

- The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
- There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
- There should be no high correlations (multicollinearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Some authors suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met.

At the centre of the logistic regression analysis is the task estimating the log odds of an event. Mathematically, logistic regression estimates a multiple linear regression function defined as:

$$\text{logit}(p) = \log\left(\frac{p(y=1)}{1-(p=1)}\right) = \beta_0 + \beta_1 x_{i2} + \beta_2 \cdot x_{i2} + \_ + \beta_p \cdot x_{in}$$

for i = 1…….n.

The object description of the Logistic Regression Classifier used is given below:

Object Name	LogisticRegression
Parameters	**Value**
C=1.0	C=1.0
solver	'lbfgs'
class_weight	None
dual	False
fit_intercept	True
intercept_scaling	1
max_iter	100
multi_class	'warn'
n_jobs	None
penalty	'l2'
random_state=None	None
tol	0.0001
verbose	0
warm_start	False

Table 20: Object Parameter Table for Logistic Regression Classifier

The intercept of the trained model is: -0.62505495.

The coefficients of the model are given in the table below:

Attribute	**Coefficient**
ApplicantIncome	-1.90032183e-04
CoapplicantIncome	-6.94385392e-05
LoanAmount	1.41847856e-03
Credit_History	1.50926139e+00
Property_Area	7.27142583e-01

Table 21: Coefficients of attributes if the trained Logistic Regression Model

Now we created a confusion matrix to view the actual and predicted test results. Given below is the confusion matrix:

Predicted Loan_Status	**0**	**1**
Actual Loan_Status		
0	6	3
1	1	14

Table 22: Confusion matrix of Logistic Regression Classifier

Now we plot the Receiver Operating Characteristics (ROC) curve for our trained Logistic Regression model. The ROC curve is given below:

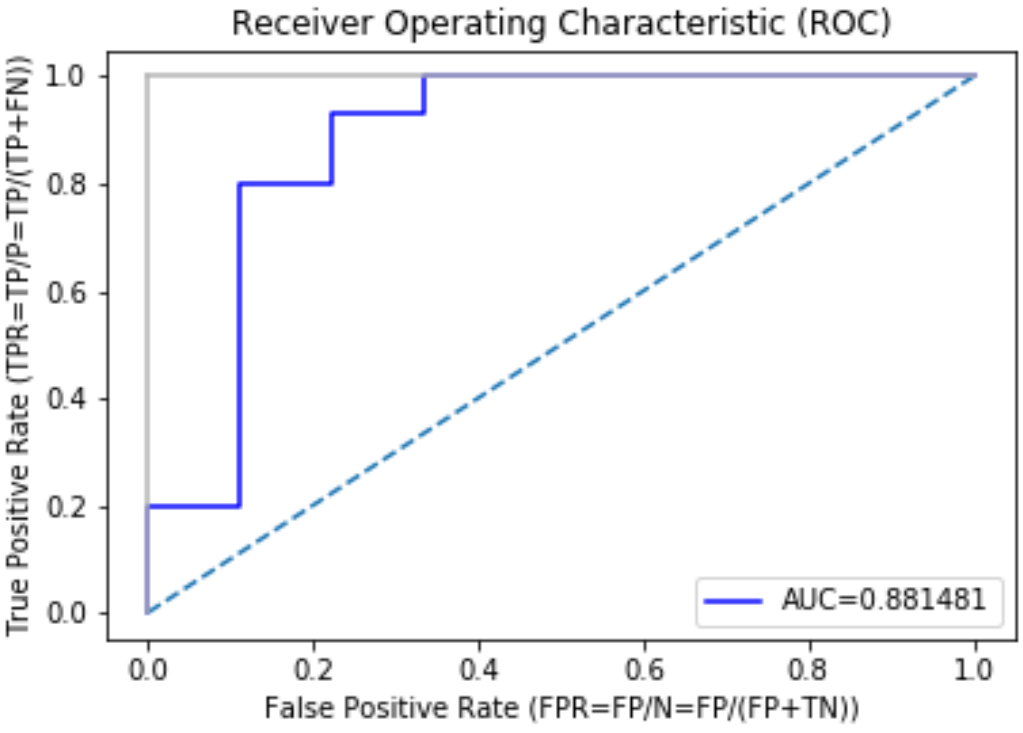


Fig 11: ROC curve of Logistic Regression model

From the Receiver Operating Characteristic (ROC) graph, we find the Area Under Curve (AUC) for our Logistic Regression model is 0.881481.

AUC value of Logistic Regression = 0.881481.

Now we will be preparing the classification report of our Logistic Regression model.

Classification Report of Logistic Regression Classifier model				
Accuracy			0.881481	
	precision	**recall**	**f1-score**	**support**
0	0.86	0.67	0.75	9
1	0.82	0.93	0.87	15
Logistic Regression RMSE			0.4082	
Logistic Regression R-squared			0.8333	

Table 23: Classification Report table of Logistic Regression classifier

Comparison of the Models trained

We trained 4 models using the 4 algorithms viz.

1. k-Nearest Neighbour
2. Gaussian Naive Bayes
3. Decision Tree and
4. Logistic Regression

The 4 models had different accuracy. The comparison of the accuracies of the models are given below:

Model	Accuracy %
k-Nearest Neighbour	58.334
Gaussian Naive Bayes	91.667
Decision Tree	79.167
Logistic Regression	83.334

Table 24: Accuracy Comparison table

The following bar graph shows the accuracy comparison in graphical way:

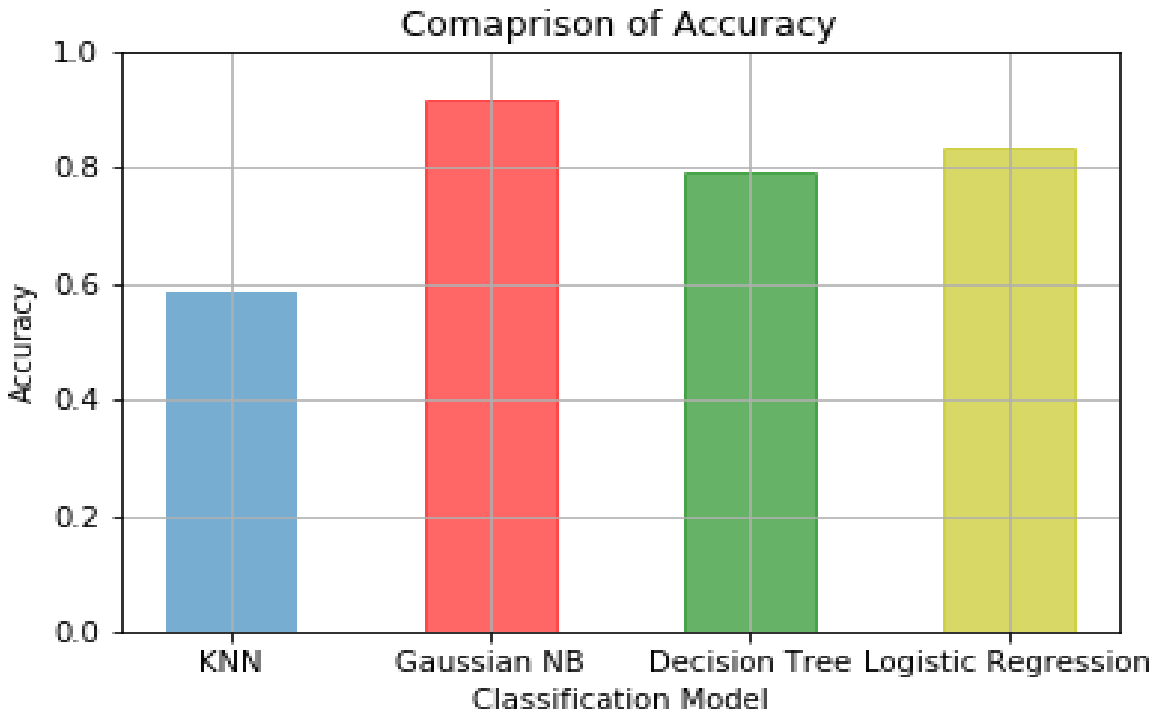


Fig 12: Comparison of accuracy of the 4 different models trained

Thus, from the above comparison we can see that Gaussian Naive Bayes classifier has the highest accuracy. So, our selected model is Gaussian Naive Bayes Classifier Model.

We also trained the above-mentioned classifiers without removing outliers from the dataset. The following table shows the comparison of accuracy of the classifiers trained:

Classifier Model Trained	Data Used	Accuracy %
k-NN	Removing outliers, without standardizing	58.334
k-NN	Removing outliers, with standardizing	56.000
k-NN	With outliers, without standardizing	55.556
K-NN	With outliers, with standardizing	60.714
Gaussian Naive Bayes	With outliers	76.923
Gaussian Naive Bayes	Removing outliers	91.667
Logistic Regression	With outliers	68.000
Logistic Regression	Removing outliers	83.334
Decision Tree	With outliers	80.000
Decision Tree	Removing outliers	79.167

Table 25: Comparison of different models trained with different version of the dataset

We choose the following classifier models as they were most accurate and in accordance with each other:

Classifier Model Trained	Data Used	Accuracy %
k-NN	Removing outliers, without standardizing	58.334
Gaussian Naive Bayes	Removing outliers	91.667
Logistic Regression	Removing outliers	83.334
Decision Tree	Removing outliers	79.167

Table 26: Selected classifier models and dataset used

Test Dataset

Testing the given Test Dataset

We were given a test dataset for this loan approval problem. We pre-process the given test dataset in similar way we pre-processed our train dataset. The methodology followed is given below:

- Change the categorical values to numeric values using same process we used during changing values in our train dataset.
- Checking for null values.
    - If null values are present, we will fill them or drop the row containing the null value based on the dataset.
- Checking for outliers.
    - If outliers are present, they will either be removed or replaced by following a suitable method depending on the dataset.

N.B.: We handled the outliers using non-parametric statistics method. We used Median Absolute Deviation or MAD to handle the outliers. In this method, data outside the range 3 * mad is excluded and replaced with median of the data. This is the same procedure we used to handle outliers in our train dataset.

After pre-processing of the test dataset had been done, we retrained the models with 100% of the train dataset given and used those trained models to predict the outcome for each input in the test dataset. There was a total of 514 inputs in the test dataset.

The outcomes obtained are tabulated as below:

Classifier Models	**Outcomes**	
	0 (No)	**1 (Yes)**
Logistic Regression	165	349
Gaussian Naive Bayes	247	267
Decision Tree	259	255
K-Nearest Neighbour	142	372

Table 27: Outcomes of 4 different classifier models on the test dataset.

As, the accuracy of the Gaussian Naive Bayes Classifier Model is the highest, we accept the outcome provided by the Gaussian Naive Bayes Classifier.

Codes

Loan Approval System

@authors: Shouvit & Debaleen

Importing modules

[1]:
```
import numpy as np # for numpy functions and ndarray
import pandas as pd # for handling data
import matplotlib.pyplot as plt # for plotting graphs i.e. data visualisation
import seaborn as sns # for plotting graphs i.e. data visualisation
```

Loading data and getting data descriptions

[2]:
```
# loading training dataset
Data = pd.read_csv("train.csv")

# making a copy of the main Data to manipulate
X = Data.copy()

# dataset description
X.describe(include="all")
```

[2]:

	Application_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
count	100.000000	99	100	100	100	94	100.00000	100.000000	95.000000	95.000000	92.000000	100	100
unique	NaN	2	2	4	2	2	NaN	NaN	NaN	NaN	NaN	3	2
top	NaN	M	Yes	0	Graduate	No	NaN	NaN	NaN	NaN	NaN	Urban	Y
freq	NaN	84	69	60	77	81	NaN	NaN	NaN	NaN	NaN	50	64
mean	1160.470000	NaN	NaN	NaN	NaN	NaN	4122.83000	1700.550000	134.221053	341.684211	0.836957	NaN	NaN
std	104.622212	NaN	NaN	NaN	NaN	NaN	2258.89434	1947.668891	63.456163	61.309342	0.371429	NaN	NaN
min	1002.000000	NaN	NaN	NaN	NaN	NaN	1000.00000	0.000000	17.000000	60.000000	0.000000	NaN	NaN
25%	1062.500000	NaN	NaN	NaN	NaN	NaN	2636.00000	0.000000	99.500000	360.000000	1.000000	NaN	NaN
50%	1153.000000	NaN	NaN	NaN	NaN	NaN	3598.00000	1558.500000	120.000000	360.000000	1.000000	NaN	NaN
75%	1253.500000	NaN	NaN	NaN	NaN	NaN	4710.00000	2394.500000	154.500000	360.000000	1.000000	NaN	NaN
max	1343.000000	NaN	NaN	NaN	NaN	NaN	12841.00000	10968.000000	349.000000	480.000000	1.000000	NaN	NaN

From the data description, we can see that there are null values in columns 'Gender', 'Self_Employed', 'LoanAmount', 'Loan_Amount_Term', and 'Credit_History'. Also there are non-numeric columns.
So we need to pre-process the data before using it to train any model.

Pre-Processing Data

Changing non-numeric values to numeric values

[3]:
```
X.Married.replace(['No', 'Yes'], [0, 1], inplace=True) # replacing 'No' with 0 and 'Yes' with 1 in 'Married'
                                                       # column

X.Gender.replace(['M', 'F'], [0, 1], inplace = True) # replacing 'M' with 0 and 'F' with 1 in 'Gender' column

X.Dependents.replace(['0', '1', '2', '3+'], [0, 1, 2, 3], inplace=True) # replacing '0' with 0, '1' with 1, '2'
                                                                        # with 2 and '3+' with 3 in
                                                                        # 'Dependents' column

X.Education.replace(['Not Graduate', 'Graduate'], [0, 1], inplace=True) # replacing 'Not Graduate' with 0
                                                                        # and 'Graduate' with 1 in 'Education'
                                                                        # column

X.Self_Employed.replace(['No', 'Yes'], [0, 1], inplace=True) # replacing 'No' with 0 and 'Yes' with 1 in
                                                             # 'Self_Employed' column

X.Property_Area.replace(['Rural','Semiurban', 'Urban'], [0, 1, 2], inplace=True) # replacing 'Rural' with 0,
                                                                                 # 'Semiurban' with 1 and
                                                                                 # 'Urban' with 2 in
                                                                                 # 'Property_Area' column

X.Loan_Status.replace(['N', 'Y'], [0, 1], inplace = True) # replacing 'N' with 0 and 'Y' with 1 in
                                                          # 'Loan_Status' column

print(X.describe(include="all"))
print(X.dtypes)
```

```
       Application_ID     Gender     Married  Dependents   Education  \
count      100.000000  99.000000  100.000000  100.000000  100.000000
mean      1160.470000   0.151515    0.690000    0.730000    0.770000
std        104.622212   0.360375    0.464823    1.013594    0.422953
min       1002.000000   0.000000    0.000000    0.000000    0.000000
25%       1062.500000   0.000000    0.000000    0.000000    1.000000
50%       1153.000000   0.000000    1.000000    0.000000    1.000000
75%       1253.500000   0.000000    1.000000    1.250000    1.000000
max       1343.000000   1.000000    1.000000    3.000000    1.000000

       Self_Employed  ApplicantIncome  CoapplicantIncome  LoanAmount  \
count      94.000000       100.00000         100.000000   95.000000
mean        0.138298      4122.83000        1700.550000  134.221053
std         0.347063      2258.89434        1947.668891   63.456163
min         0.000000      1000.00000           0.000000   17.000000
25%         0.000000      2636.00000           0.000000   99.500000
50%         0.000000      3598.00000        1558.500000  120.000000
75%         0.000000      4710.00000        2394.500000  154.500000
max         1.000000     12841.00000       10968.000000  349.000000

       Loan_Amount_Term  Credit_History  Property_Area  Loan_Status
count         95.000000       92.000000     100.000000   100.000000
mean         341.684211        0.836957       1.410000     0.640000
std           61.309342        0.371429       0.652811     0.482418
min           60.000000        0.000000       0.000000     0.000000
25%          360.000000        1.000000       1.000000     0.000000
50%          360.000000        1.000000       1.500000     1.000000
75%          360.000000        1.000000       2.000000     1.000000
max          480.000000        1.000000       2.000000     1.000000
```

```
Application_ID          int64
Gender                float64
Married                 int64
Dependents              int64
Education               int64
Self_Employed         float64
ApplicantIncome         int64
CoapplicantIncome       int64
LoanAmount            float64
Loan_Amount_Term      float64
Credit_History        float64
Property_Area           int64
Loan_Status             int64
dtype: object
```

Non-Numeric to Numeric Change table

Columns	Initial Value	Replaced Value	
Married	No	0	
	Yes	1	
Gender	M	0	
	F	1	
Dependents	0	0	Note: Changed type from str to int
	1	1	Note: Changed type from str to int
	2	2	Note: Changed type from str to int
	3+	3	
Education	Not Graduate	0	
	Graduate	1	
Self_Employed	No	0	
	Yes	1	
Property_Area	Rural	0	
	Semiurban	1	
	Urban	2	
Loan_Status	N	0	
	Y	1	

Searching for Null values

[4]:
```
print("Null values count:\n", X.isnull().sum())  # sum of Null values in a column
```

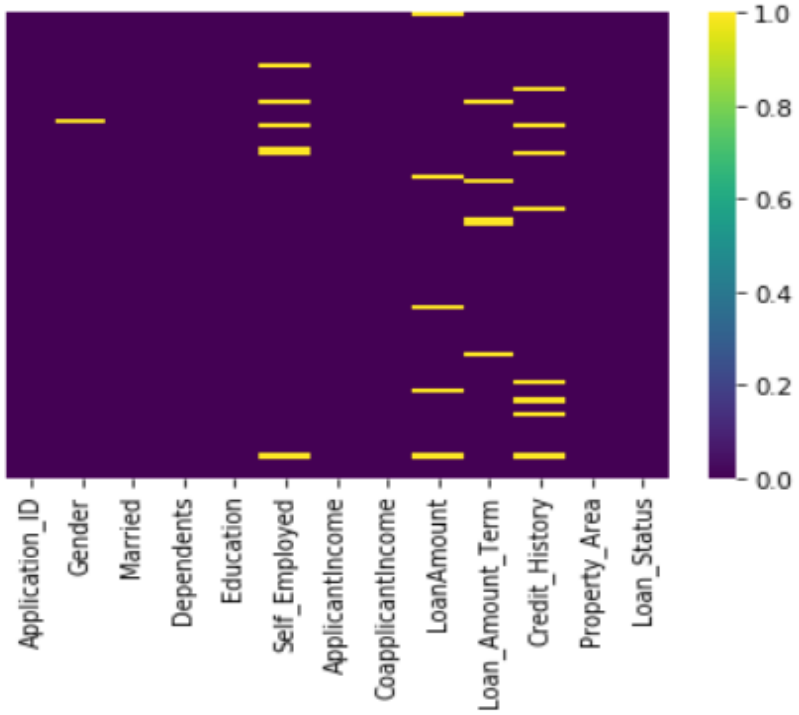
```
Null values count:
 Application_ID       0
Gender                1
Married               0
Dependents            0
Education             0
Self_Employed         6
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount            5
Loan_Amount_Term      5
Credit_History        8
Property_Area         0
Loan_Status           0
dtype: int64
```

We can see that the dataset has null values. Let us visualise the null values using heatmap.

Visualising Null values using heatmap

[5]:
```
sns.heatmap(X.isnull(),yticklabels=False,cbar=True,cmap='viridis')
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2a0db995780>
```

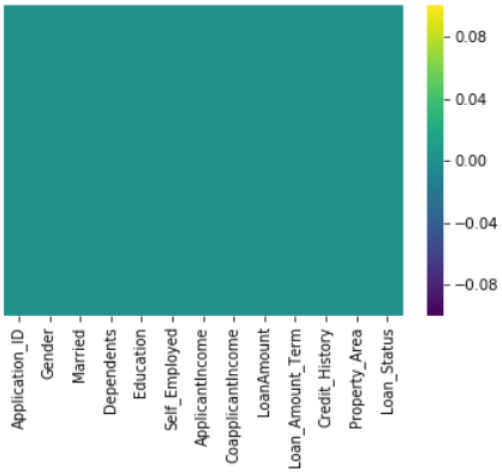


Handling Null values

[6]:
```
# filling Null values in 'Credit_History' with 0 (as stated in the dataset description from source)
X.Credit_History.fillna(0, inplace=True)

#filling Null values in binary columns with mode value
X.Self_Employed.fillna(X.Self_Employed.mode()[0], inplace=True)
X.Gender.fillna(X.Gender.mode()[0], inplace=True)

# filling Null values with mean +- std in non-binary columns
avg = X.LoanAmount.mean()
std = X.LoanAmount.std()
count = X.LoanAmount.isnull().sum()
random = np.random.randint(avg-std,avg+std,size=count)
X['LoanAmount'][np.isnan(X['LoanAmount'])]=random

avg = X.Loan_Amount_Term.mean()
std = X.Loan_Amount_Term.std()
count = X.Loan_Amount_Term.isnull().sum()
random = np.random.randint(avg-std,avg+std,size=count)
X['Loan_Amount_Term'][np.isnan(X['Loan_Amount_Term'])]=random


sns.heatmap(X.isnull(),yticklabels=False,cbar=True,cmap='viridis')
X.describe(include="all")
```

[6]:

	Application_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_
count	100.000000	100.00000	100.000000	100.000000	100.000000	100.000000	100.00000	100.000000	100.000000	100.00
mean	1160.470000	0.15000	0.690000	0.730000	0.770000	0.130000	4122.83000	1700.550000	135.480000	341.83
std	104.622212	0.35887	0.464823	1.013594	0.422953	0.337998	2258.89434	1947.668891	62.427409	60.02
min	1002.000000	0.00000	0.000000	0.000000	0.000000	0.000000	1000.00000	0.000000	17.000000	60.00
25%	1062.500000	0.00000	0.000000	0.000000	1.000000	0.000000	2636.00000	0.000000	100.000000	360.00
50%	1153.000000	0.00000	1.000000	0.000000	1.000000	0.000000	3598.00000	1558.500000	122.000000	360.00
75%	1253.500000	0.00000	1.000000	1.250000	1.000000	0.000000	4710.00000	2394.500000	159.750000	360.00
max	1343.000000	1.00000	1.000000	3.000000	1.000000	1.000000	12841.00000	10968.000000	349.000000	480.00



Now we have successfully handled Null values and converted non-numeric values to Numeric values. We didn't drop the rows with null values as we have a small dataset(only 100 entries).
So we are moving on to find if there are any outliers in our data and find the correlations of different attributes to our target i.e. 'Loan_Status' column in the dataset.

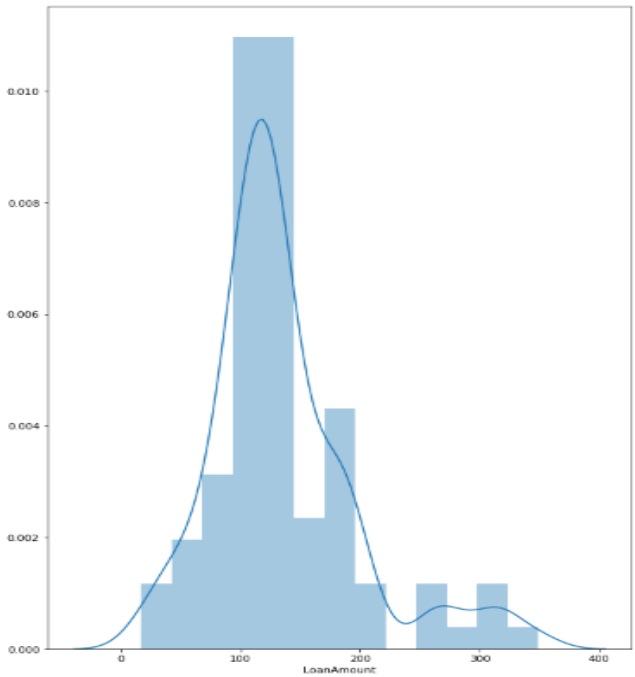
Checking correlation of every attribute with our target attribute i.e. 'Loan_Status'

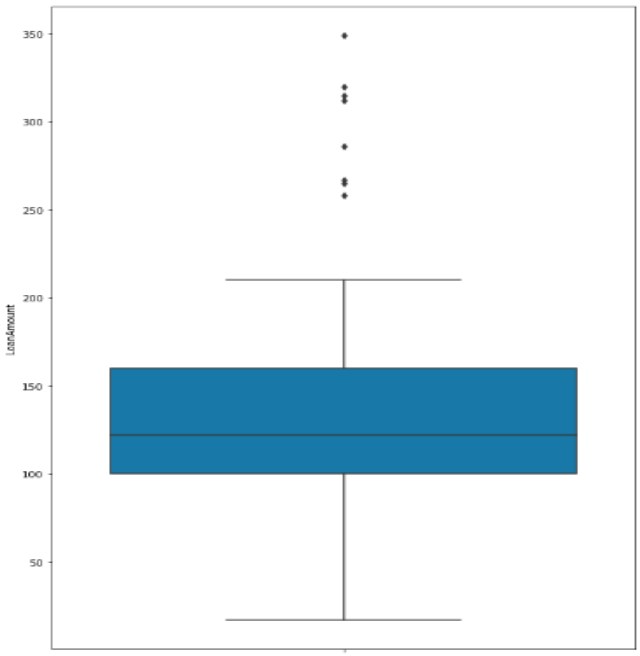
[8]:
```
f = X.columns[1:-1] # selecting all columns except 'Application_ID' since that is not a determining factor
print("\n\nCorrelation of each attribute with Loan Status: \n")
for i in f:
    corr = X[i].corr(X['Loan_Status'])
    print(i, ":", corr)
```

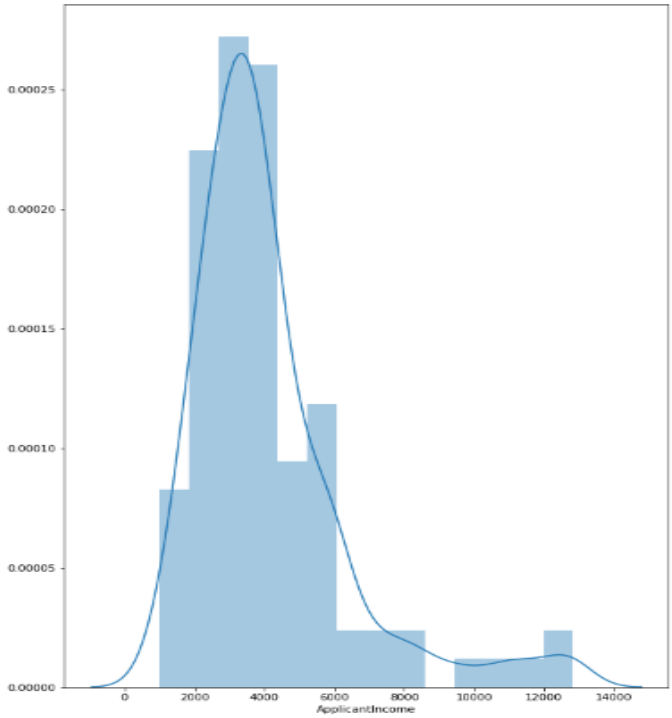
```
Correlation of each attribute with Loan Status:

Gender : -0.09335200560186736
Married : 0.12792979689405484
Dependents : 0.0057840889995565475
Education : -0.013861413315217981
Self_Employed : 0.04212465653604544
ApplicantIncome : -0.1196673096146476
CoapplicantIncome : -0.03110315854904195
LoanAmount : -0.14446441796336063
Loan_Amount_Term : -0.09806487882763729
Credit_History : 0.5306941097826289
Property_Area : 0.24889474150088223
```

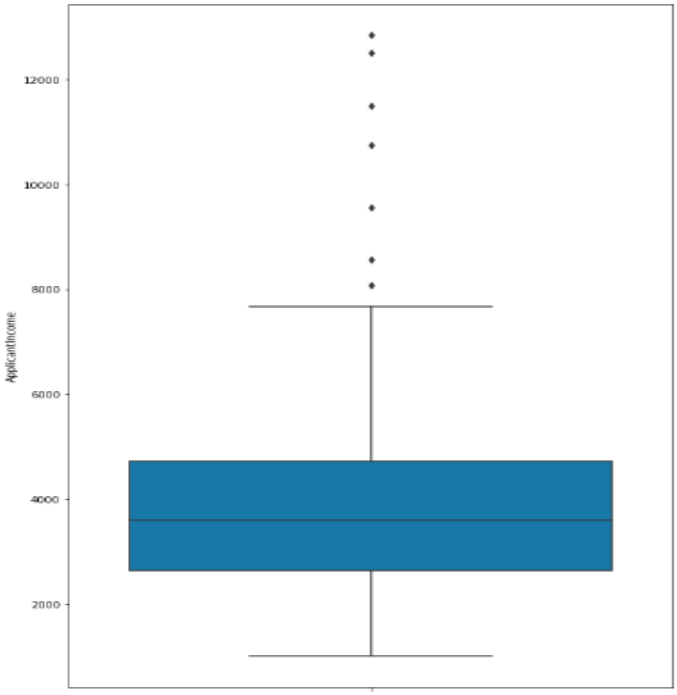
Checking for outliers

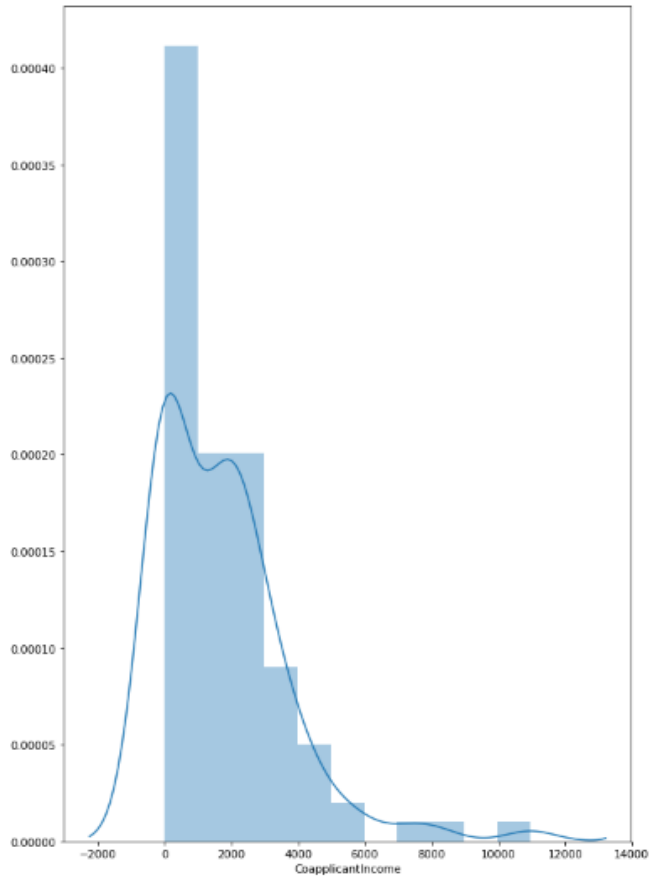
[9]:
```
# plotting distribution plot and boxplot for

f, axes = plt.subplots(3, 2, figsize=(20, 50))

sns.distplot(old["LoanAmount"], ax=axes[0][0])
plt.grid()
sns.boxplot( y="LoanAmount", data=old, palette = "winter", ax = axes[0][1])
plt.grid()

sns.distplot(old["ApplicantIncome"], ax=axes[1][0])
plt.grid()
sns.boxplot( y="ApplicantIncome", data=old, palette = "winter", ax = axes[1][1])
plt.grid()

sns.distplot(old["CoapplicantIncome"], ax=axes[2][0])
plt.grid()
sns.boxplot( y="CoapplicantIncome", data=old, palette = "winter", ax = axes[2][1])
plt.grid()
```

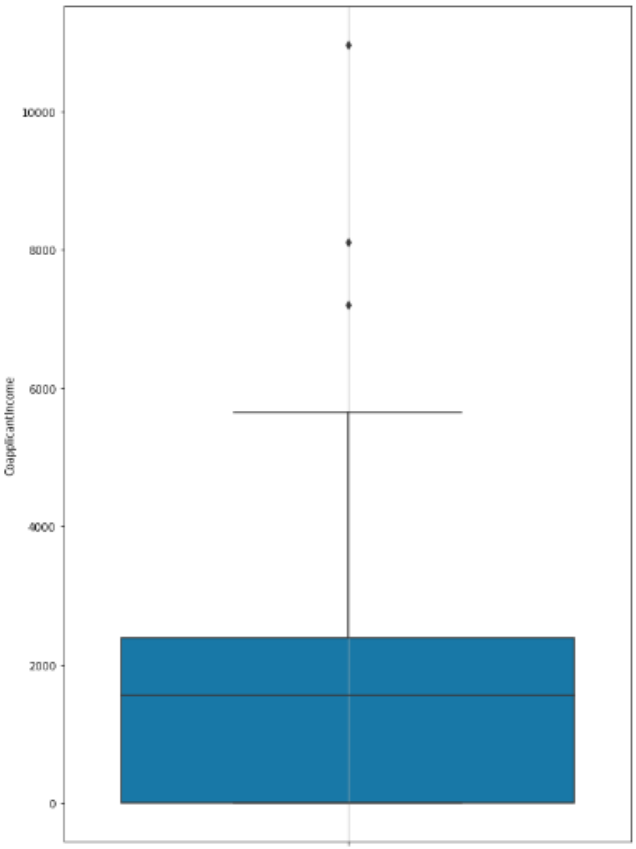

0.010
0.008
0.006
0.004
0.002
0.000
0
100
200
300
400
LoanAmount


350
300
250
200
150
100
50
LoanAmount


0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
2000
4000
6000
8000
10000
12000
14000
ApplicantIncome


12000
10000
8000
6000
4000
2000
ApplicantIncome





Checking for outliers

[9]:
```
# plotting distribution plot and boxplot for

f, axes = plt.subplots(3, 2, figsize=(20, 50))

sns.distplot(old["LoanAmount"], ax=axes[0][0])
plt.grid()
sns.boxplot( y="LoanAmount", data=old, palette = "winter", ax = axes[0][1])
plt.grid()

sns.distplot(old["ApplicantIncome"], ax=axes[1][0])
plt.grid()
sns.boxplot( y="ApplicantIncome", data=old, palette = "winter", ax = axes[1][1])
plt.grid()

sns.distplot(old["CoapplicantIncome"], ax=axes[2][0])
plt.grid()
sns.boxplot( y="CoapplicantIncome", data=old, palette = "winter", ax = axes[2][1])
plt.grid()
```

From the distribution plots and boxplots we can see that there are possible outliers in the columns 'LoanAmount', 'ApplicantIncome' and 'CoapplicantIncome'.

We have not checked for outliers in the columns viz. 'Gender', 'Married', 'Education', 'Self_Employed', and 'Credit_History' as they are columns having values in binaries and also initially they had non-numeric values. We have not checked for outliers in the columns viz. 'Dependents' and 'Property_Area' as they are columns having values that were initially non-numeric. We have not checked for outliers in the column 'Loan_Amount_Term' as the mean and MAD values of this column are in the range 3*mean. So this method won't have any affect on the data of the column 'Loan_Amount_Term'.

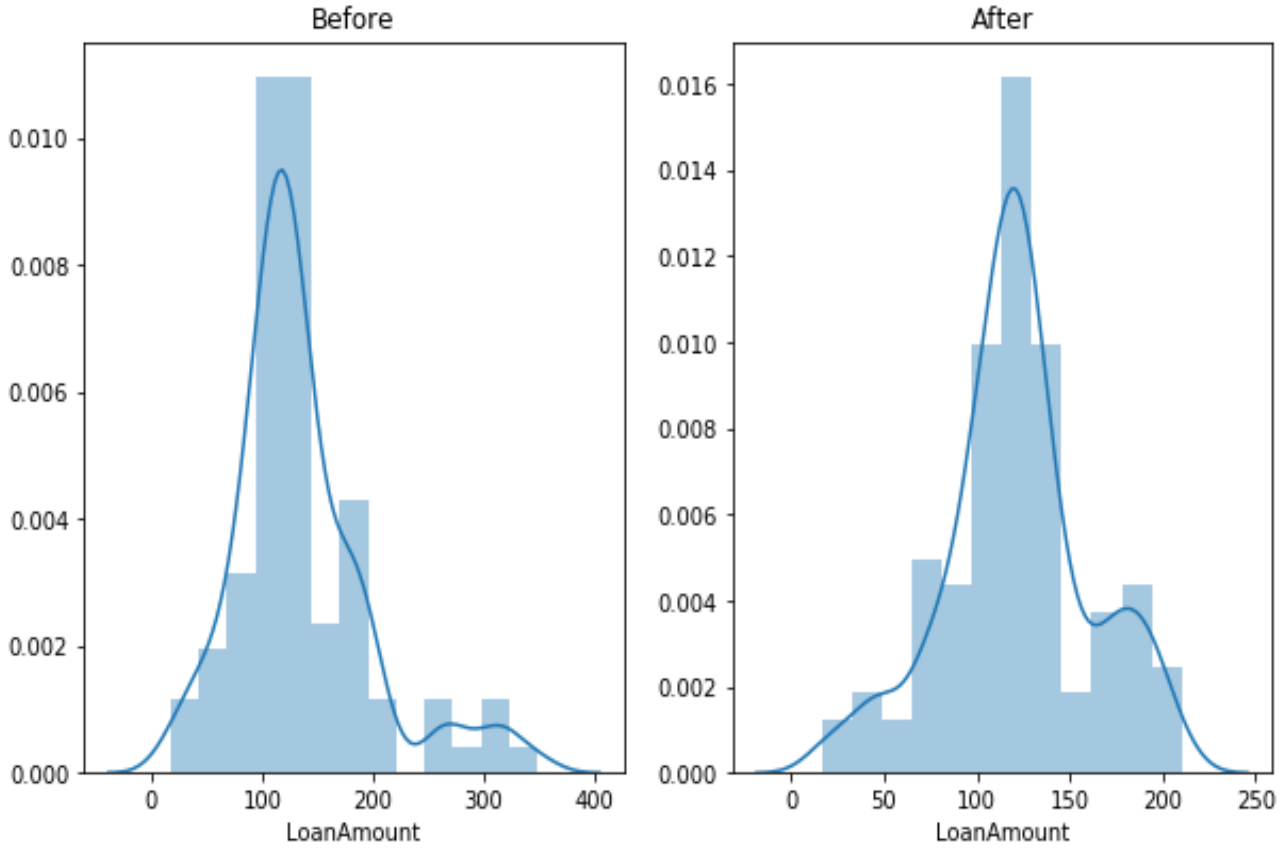
Handling possible outliers using non-parametric statistics method

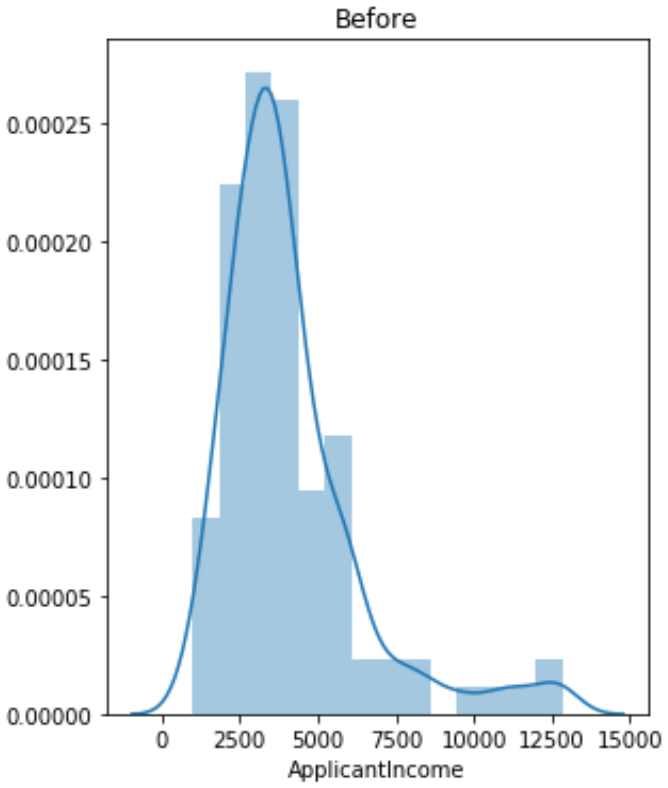
We used Median Absolute Deviation or MAD to handle the outliers

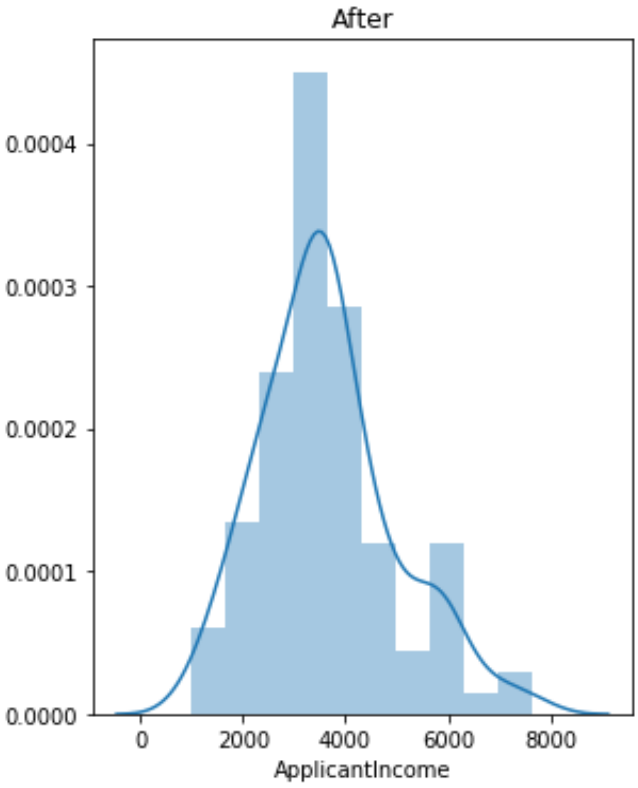
[10]:
```
# Median Absolute Deviation(MAD) , based on the median, is a robust non-parametric statistics
# exclude data outside 3 * mad and replace those values with median of the data.

# handling outliers with MAD

# Handling 'ApplicantIncome' column
mad=1.4826*np.median(np.abs(X.ApplicantIncome-X.ApplicantIncome.median()))
size_outlr_mad=X.ApplicantIncome
size_outlr_mad[((X.ApplicantIncome-X.ApplicantIncome.median()).abs()>3*mad)]=X.ApplicantIncome.median()
print("ApplicantIncome MAD :\n",size_outlr_mad.median())

# Handling 'CoapplicantIncome' column
mad=1.4826*np.median(np.abs(X.CoapplicantIncome-X.CoapplicantIncome.median()))
size_outlr_mad=X.CoapplicantIncome
size_outlr_mad[((X.CoapplicantIncome-X.CoapplicantIncome.median()).abs()>3*mad)]=X.CoapplicantIncome.median()
print("CoapplicantIncome MAD :\n",size_outlr_mad.median())

# Handling 'LoanAmount' column
mad=1.4826*np.median(np.abs(X.LoanAmount-X.LoanAmount.median()))
size_outlr_mad=X.LoanAmount
size_outlr_mad[((X.LoanAmount-X.LoanAmount.median()).abs()>3*mad)]=X.LoanAmount.median()
print("LoanAmount MAD :\n",size_outlr_mad.median())
```

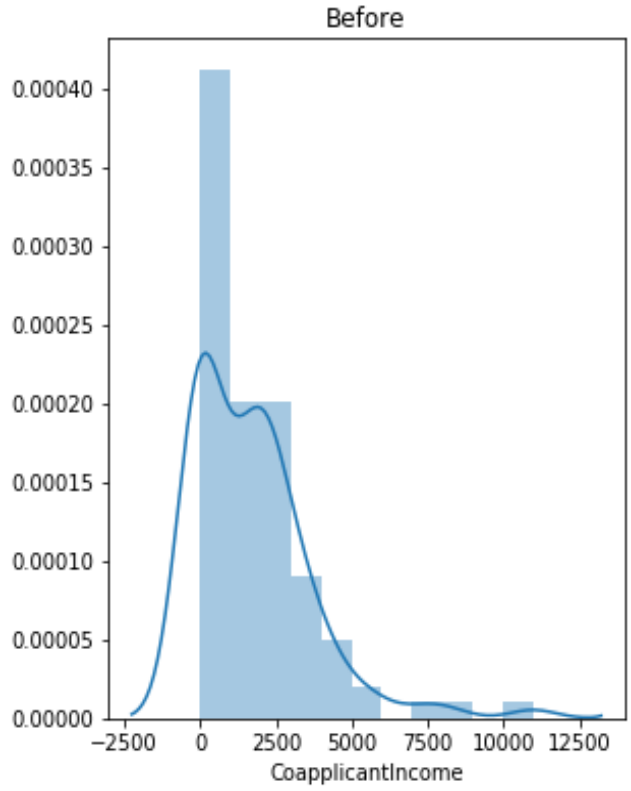
```
ApplicantIncome MAD :
 3597.0
CoapplicantIncome MAD :
 1542.25
LoanAmount MAD :
 122.0
```

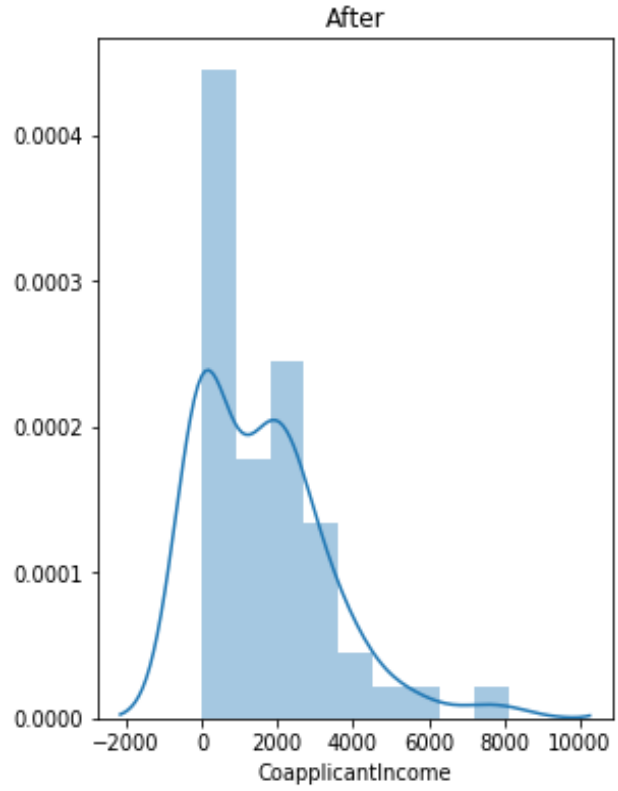
[11]:
```
# comparing distribution plots of dataset before and after handling outliers

f, axes = plt.subplots(3, 2, figsize=(10, 20))

sns.distplot(old["LoanAmount"], ax=axes[0][0]).set_title("Before")
plt.grid()
sns.distplot(X["LoanAmount"], ax=axes[0][1]).set_title("After")
plt.grid()

sns.distplot(old["ApplicantIncome"], ax=axes[1][0]).set_title("Before")
plt.grid()
sns.distplot(X["ApplicantIncome"], ax=axes[1][1]).set_title("After")
plt.grid()

sns.distplot(old["CoapplicantIncome"], ax=axes[2][0]).set_title("Before")
plt.grid()
sns.distplot(X["CoapplicantIncome"], ax=axes[2][1]).set_title("After")
plt.grid()
f.savefig('oldnewcompa.png')
```



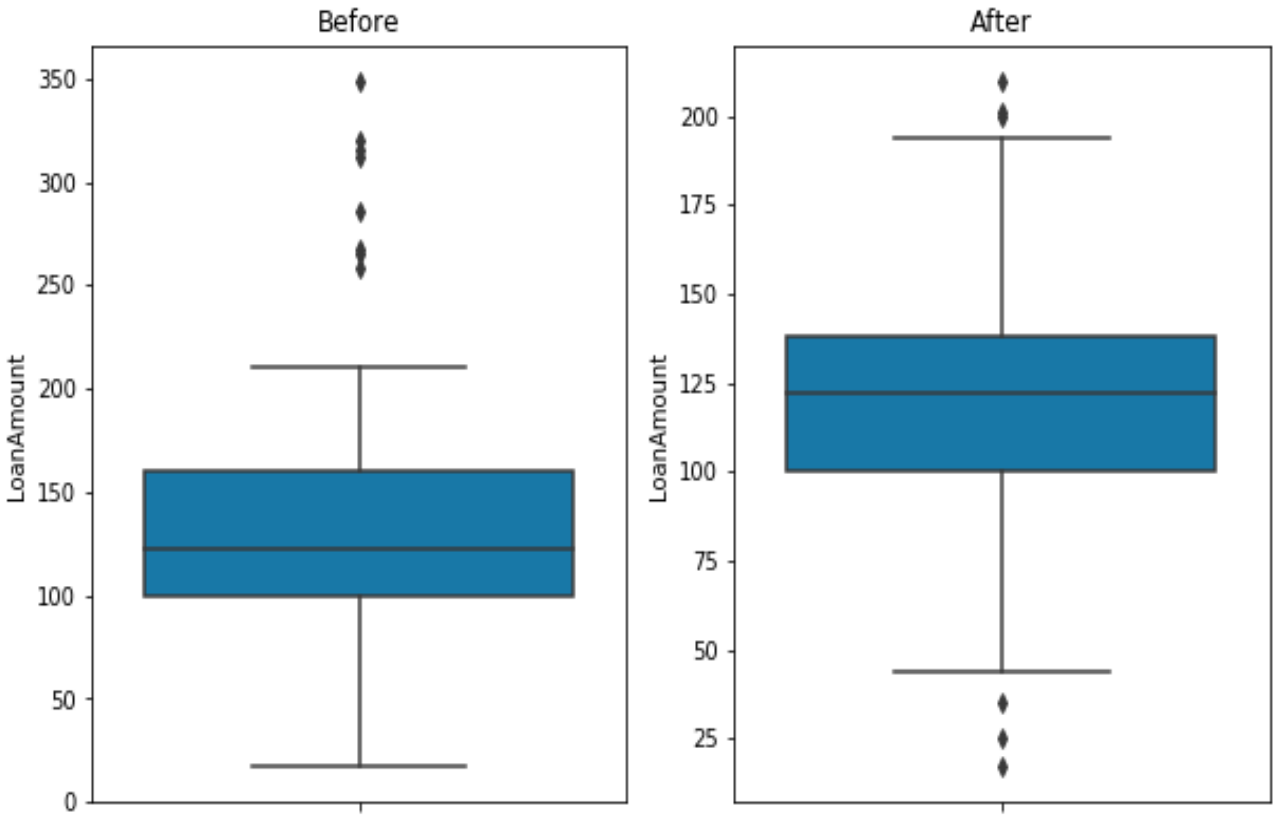

Before
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
0
2500
5000
7500
10000
12500
15000
ApplicantIncome

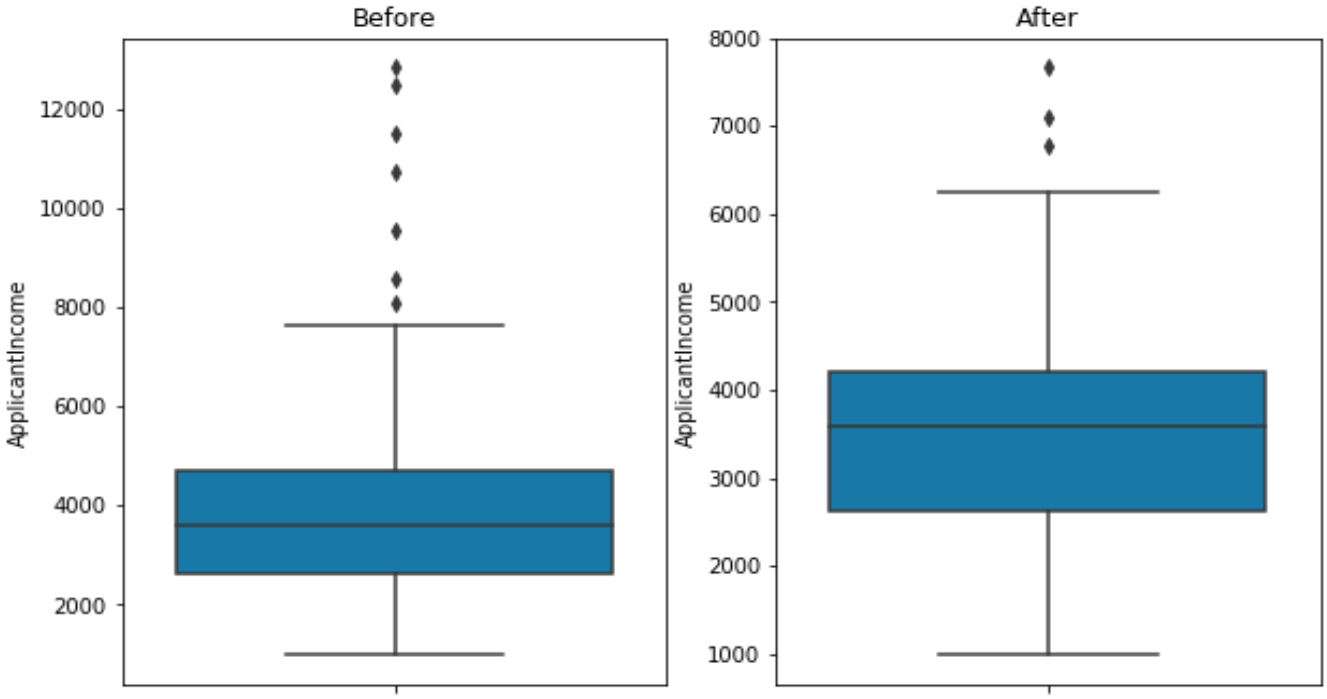

After
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
ApplicantIncome

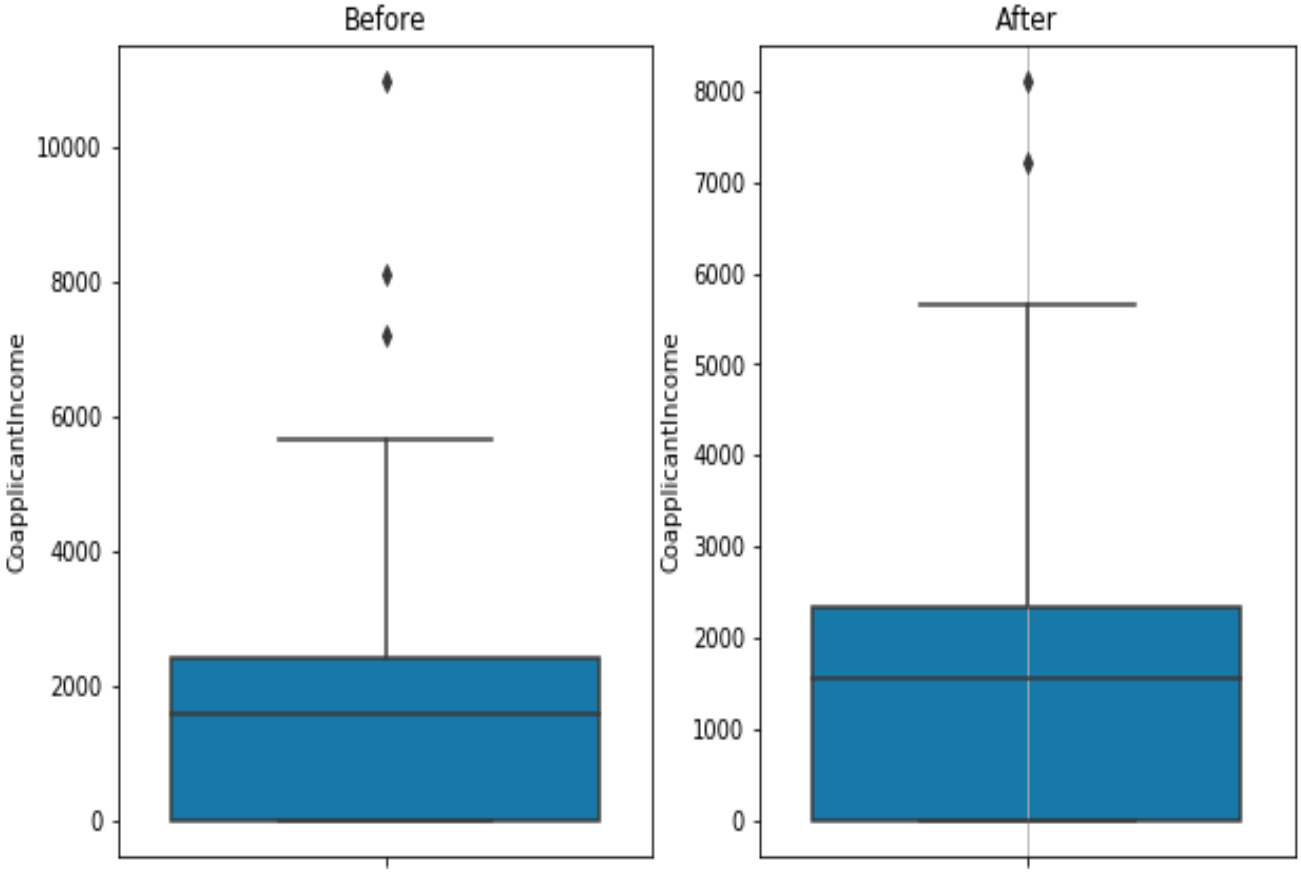

Before
0.00040
0.00035
0.00030
0.00025
0.00020
0.00015
0.00010
0.00005
0.00000
−2500
0
2500
5000
7500
10000
12500
CoapplicantIncome


After
0.0004
0.0003
0.0002
0.0001
0.0000
−2000
0
2000
4000
6000
8000
10000
CoapplicantIncome

[12]:
```
# comparing boxplots plots of dataset before and after handling outliers

f, axes = plt.subplots(3, 2, figsize=(10, 20))

sns.boxplot( y="LoanAmount", data=old, palette = "winter", ax = axes[0][0]).set_title("Before")
plt.grid()
sns.boxplot( y="LoanAmount", data=X, palette = "winter", ax = axes[0][1]).set_title("After")
plt.grid()

sns.boxplot( y="ApplicantIncome", data=old, palette = "winter", ax = axes[1][0]).set_title("Before")
plt.grid()
sns.boxplot( y="ApplicantIncome", data=X, palette = "winter", ax = axes[1][1]).set_title("After")
plt.grid()

sns.boxplot( y="CoapplicantIncome", data=old, palette = "winter", ax = axes[2][0]).set_title("Before")
plt.grid()
sns.boxplot( y="CoapplicantIncome", data=X, palette = "winter", ax = axes[2][1]).set_title("After")
plt.grid()
f.savefig("boxafterbefore.png")
```




Before
After
ApplicantIncome
12000
10000
8000
6000
4000
2000
8000
7000
6000
5000
4000
3000
2000
1000

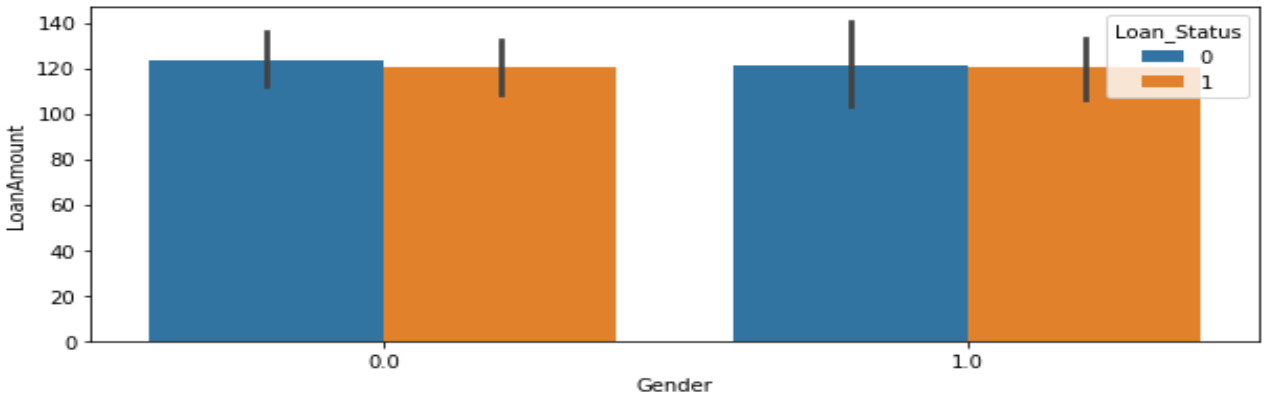

Before
After
CoapplicantIncome
10000
8000
6000
4000
2000
0
8000
7000
6000
5000
4000
3000
2000
1000
0

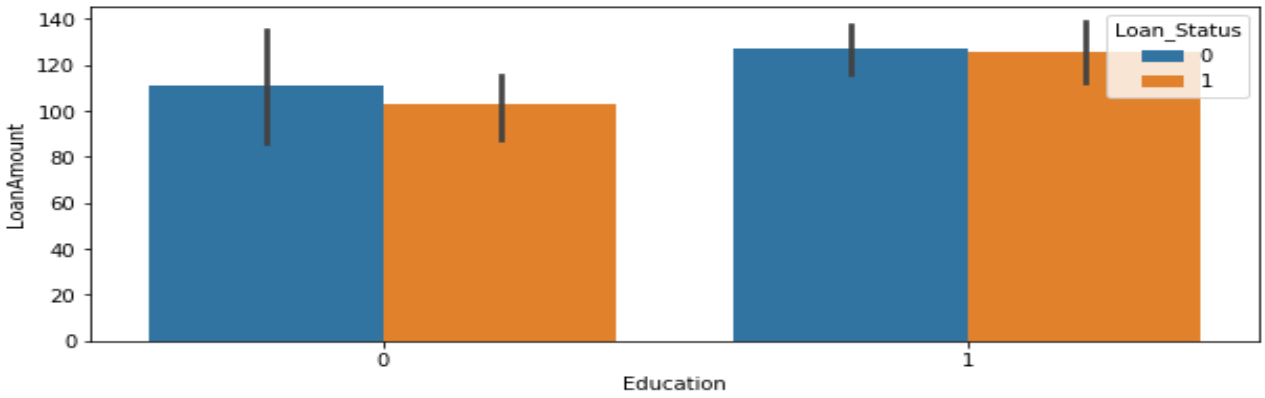
From the distribution plots and boxplots shown above, we can see that a lot of outliers have been handled. Now we will use this data to train a Random Forest Classifier model. We will be training a Random Forest Classifier model to determine the principal attributes in the dataset.

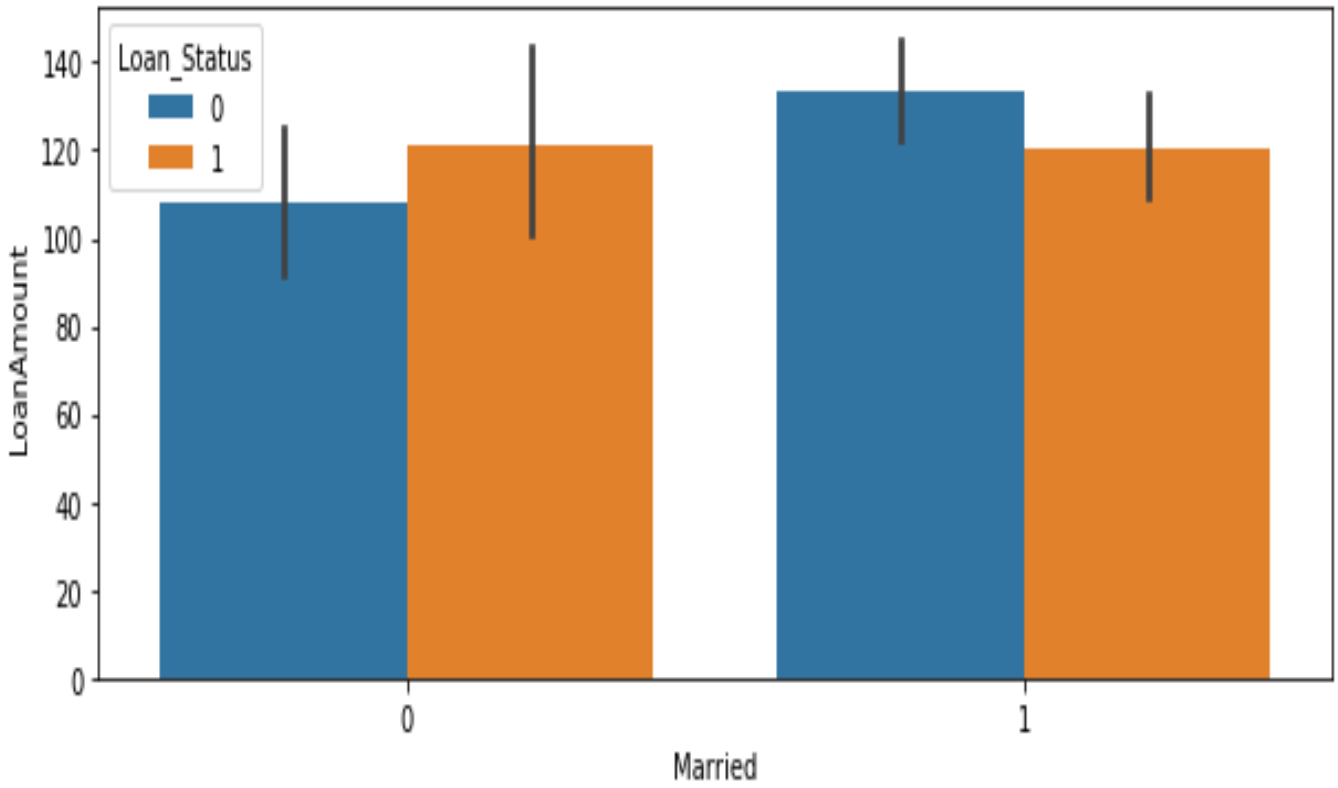
Visualising other attributes with respect to our target column i.e. 'Loan_Status'

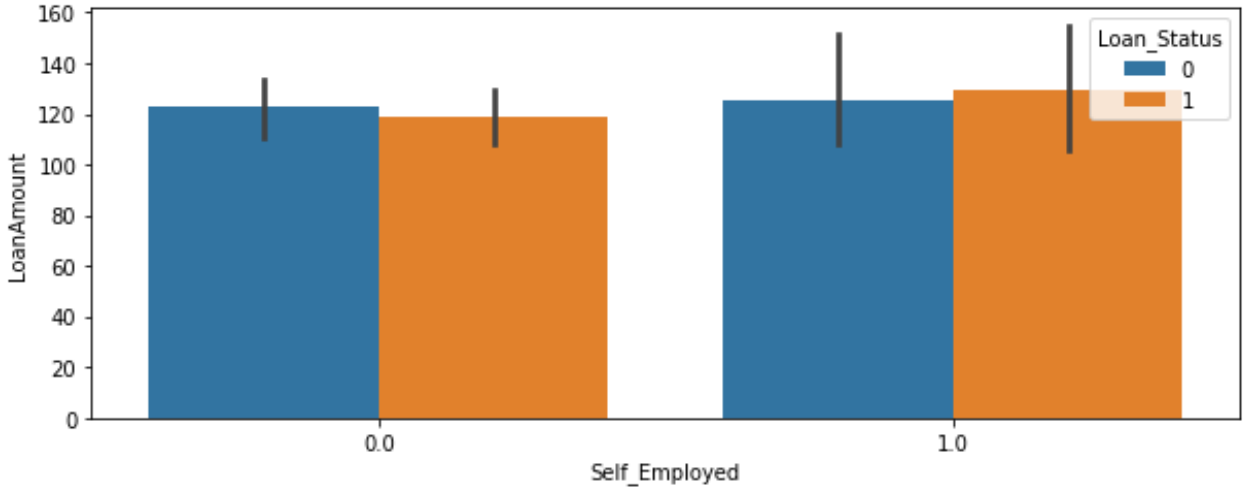
We have used barplot to visualise other attributes against the data in 'LoanAmount' column and using our target attribute i.e. 'Loan_Status' as hue.

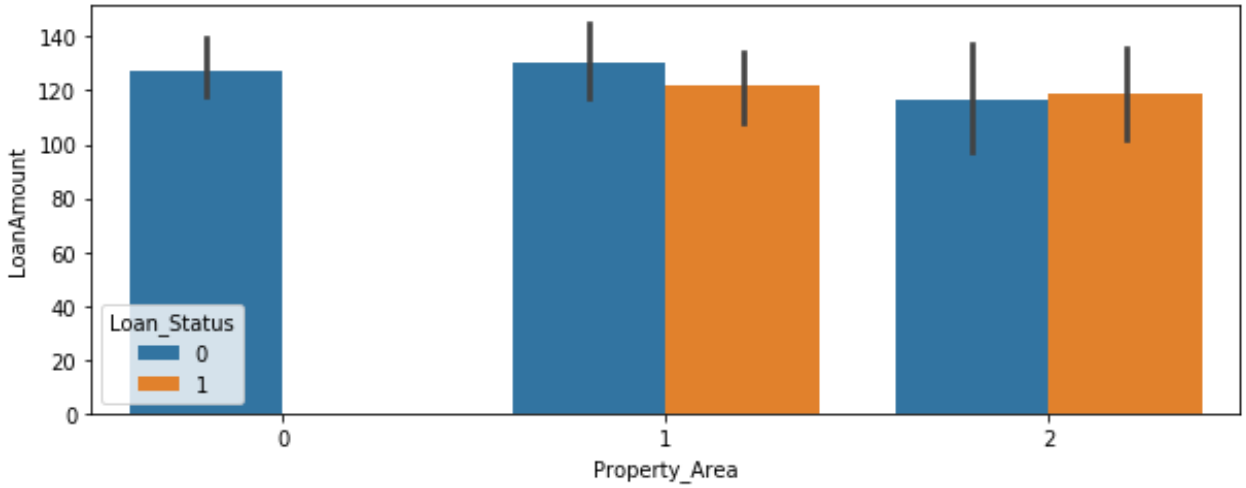
[13]:
```
f, axes = plt.subplots(7, 1, figsize=(10, 30)) # preparing the canvas

sns.barplot(x = X.Gender, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[0])

sns.barplot(x = X.Education, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[1])

sns.barplot(x = X.Married, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[2])

sns.barplot(x = X.Self_Employed, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[3])

sns.barplot(x = X.Property_Area, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[4])

sns.barplot(x = X.Credit_History, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[5])

sns.barplot(x = X.Dependents, y = X.LoanAmount, hue = X.Loan_Status, ax = axes[6])
```

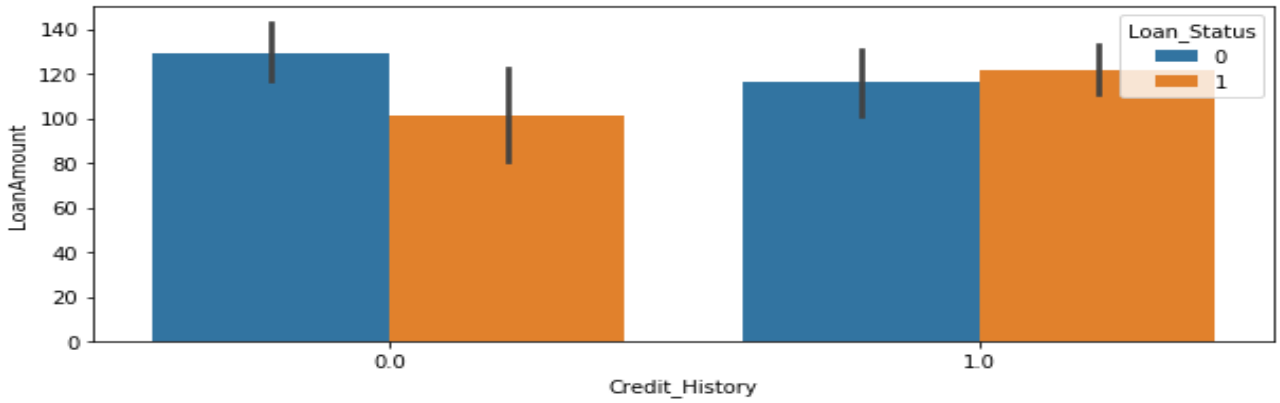

Loan_Status
0
1
LoanAmount
0
20
40
60
80
100
120
140
0.0
1.0
Gender

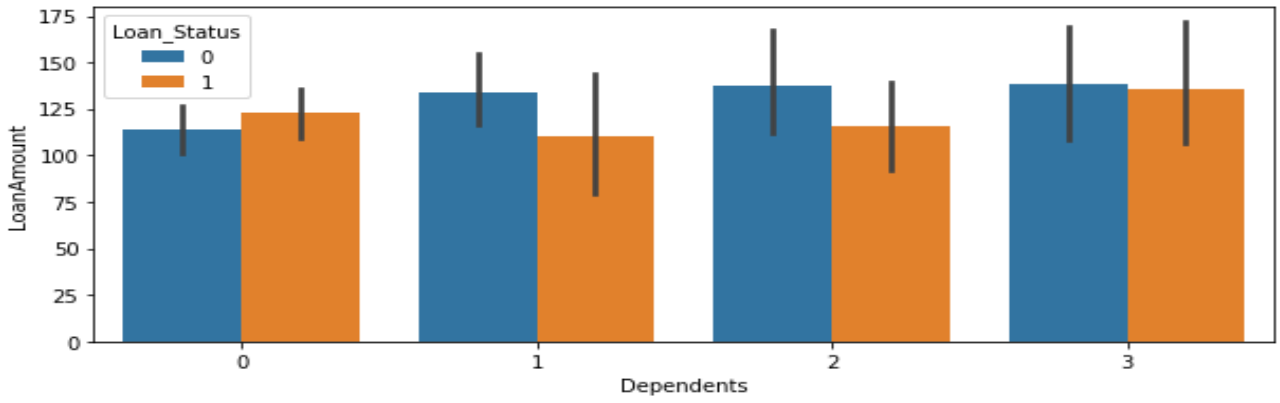

Loan_Status
0
1
LoanAmount
0
20
40
60
80
100
120
140
0
1
Education


Loan_Status
0
1
LoanAmount
0
20
40
60
80
100
120
140
0
1
Married


Loan_Status
0
1
LoanAmount
Self_Employed
0.0
1.0


Loan_Status
0
1
LoanAmount
Property_Area
0
1
2


Loan_Status
0
1
LoanAmount
Credit_History
0.0
1.0


Loan_Status
0
1
LoanAmount
Dependents
0
1
2
3

From the above graphs one interesting thing that can be noted is that no loans are approved i.e. 'Loan_Status' = 1 for 'Property_Area' value 'Rural' i.e. 'Property_Area' = 0. All the loan requests for 'Property_Area' value 'Rural' i.e. 'Property_Area' = 0 are denied.

We will now train a Random Forest Classifier Model.

Random Forest Classifier

Training a Random Forest Classifier to determine the principal attributes.

We have used Random Forest Classifier to determine the principal attributes as this classifier has a built-in feature importance function.

[14]:
```
# importing required module
from sklearn.ensemble import RandomForestClassifier
```

Now we will split the dataset into train and test parts for training the model. We use 75% of the data to train and 25% for testing purpose.

[15]:
```
X["is_train"] = np.random.uniform(0,1,len(X)) <= 0.75  # making a new column with random boolean values
                                                       # indicating which data belongs to train and which
                                                       # which data belongs to test.
train, test = X[X["is_train"]==True], X[X["is_train"]==False]  # splitting train and test dataframe
print("no. of observations for the training dataframes:",len(train))
print("no. of observations for the test dataframes:",len(test))
```

```
no. of observations for the training dataframes: 76
no. of observations for the test dataframes: 24
```

[16]:
```
f = X.columns[1:12]  # selecting feature attributes
print(f)
y = train["Loan_Status"] # selecting target attribute

cif = RandomForestClassifier(n_jobs=2, n_estimators=100, random_state=0, verbose=1) # Random Forest Classifier
                                                                                    # Object
print(cif) # printing details of the Random Forest Classifier Object
cif.fit(train[f], y) # training the model
preds = cif.predict(test[f])  # predicting the target attribute for test dataset
print(preds)
```

```
Index(['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed',
       'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area'],
      dtype='object')
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=2,
            oob_score=False, random_state=0, verbose=1, warm_start=False)
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.1s finished
[1 1 0 1 1 0 0 1 0 0 0 0 1 0 1 1 1 1 0 0 0 1 1 1]
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
```

[17]:
```
# Creating a confusion matrix to view the actual and predicted results
pd.crosstab(test["Loan_Status"], preds, rownames=["Actual Loan_Status"], colnames=["Predicted Loan_Status"])
```

[17]:

Actual Loan_Status \ Predicted Loan_Status	0	1
0	7	1
1	4	12

[18]:
```
z = test["Loan_Status"]
cif.score(test[f], z) # accuracy of the Random Forest Classifier Model
```

```
[Parallel(n_jobs=2)]: Using backend ThreadingBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done  46 tasks      | elapsed:    0.0s
[Parallel(n_jobs=2)]: Done 100 out of 100 | elapsed:    0.0s finished
```

[18]:
```
0.7916666666666666
```

[19]:
```
# view a list of the features & their importance scores
list(zip(train[f], cif.feature_importances_))
```

[19]:
```
[('Gender', 0.022998752001938204),
 ('Married', 0.029356323610496546),
 ('Dependents', 0.04632544519905084),
 ('Education', 0.01632747444018825),
 ('Self_Employed', 0.01291837290989398),
 ('ApplicantIncome', 0.1908564421569762),
 ('CoapplicantIncome', 0.10614248164617252),
 ('LoanAmount', 0.1759079613657668),
 ('Loan_Amount_Term', 0.03874584019674761),
 ('Credit_History', 0.21857887517056812),
 ('Property_Area', 0.1418420313022009)]
```

As we can see from the calculated feature importance values, the attributes viz. 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Credit_History' and 'Property_Area' have higher values compared to other attributes. So we will select these 5 attributes as our principal attribute to train our required models.
One interesting thing to notice here is that although 'Married' attribute had a high correlation value with our target attribute, 'Loan_Status', the feature importance of 'Married' attribute is lower.

NOTE: These values will change with each run. But these 5 attributes will always have the highest feature importance if trained as above.

Sl.No.	Principal Attributes selected	Feature importance
1.	ApplicantIncome	0.1746432652177679
2.	CoapplicantIncome	0.11756790122516193
3.	LoanAmount	0.12842666389187288
4.	Credit_History	0.22613376236476954
5.	Property_Area	0.14291201191868688

Now we will modify the dataset to include only the selected principal attributes.

[20]:
```
final_Dataset = X.copy() # making a new copy of the modified dataset
final_Dataset = final_Dataset.drop(["Application_ID", "Education","Gender", "Married", \
                        "Dependents", "Self_Employed", "Loan_Amount_Term", "is_train"],axis=1) # keeping only
                                                                                               # the principal attributes
                                                                                               # and the target attribute
final_Dataset.describe(include="all") # describing the final dataset
```

[20]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area	Loan_Status
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000
mean	3636.720000	1606.455000	121.520000	0.770000	1.410000	0.640000
std	1325.607679	1707.964583	39.314962	0.422953	0.652811	0.482418
min	1000.000000	0.000000	17.000000	0.000000	0.000000	0.000000
25%	2636.000000	0.000000	100.000000	1.000000	1.000000	0.000000
50%	3597.000000	1542.250000	122.000000	1.000000	1.500000	1.000000
75%	4197.500000	2341.500000	138.000000	1.000000	2.000000	1.000000
max	7660.000000	8106.000000	210.000000	1.000000	2.000000	1.000000

Saving the modified dataset containing only the principal attributes

[21]:
```
final_Dataset.to_csv(r'finalTrain.csv', index = None, header = True)
```

Training our required models

Our project goal requires us to train specific 4 classifier models viz.
1. KNN classifier
2. Naive Bayes Classifier
3. Decision Tree Classifier
and
4. Linear Regression Classifier (As the target attribute is binary, we will be using Logistic Regression Classifier)

We will be using the dataset obtained after pre-processing the given train dataset to train our required models.

[22]:
```
# importing required modules

from sklearn import model_selection
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.metrics import roc_curve, roc_auc_score, auc, mean_squared_error, accuracy_score, classification_report
```

[23]:
```
final_Dataset["is_train"] = np.random.uniform(0,1,len(final_Dataset)) <= 0.75
train, test = final_Dataset[final_Dataset["is_train"]==True], final_Dataset[final_Dataset["is_train"]==False]
print("no. of observations for the training dataframes:",len(train))
print("no. of observations for the test dataframes:",len(test))
f = final_Dataset.columns[:-2]
print(f)
```

```
no. of observations for the training dataframes: 76
no. of observations for the test dataframes: 24
Index(['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount', 'Credit_History',
       'Property Area'],
```

KNN classifier

[24]:
```
# importing required modules
from sklearn.neighbors import KNeighborsClassifier
# from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import neighbors
```

[25]:
```
# First Find the value of K :
            # standardizer = StandardScaler()
            # standardize features
            # X_std = standardizer.fit_transform(final_Dataset[f])
# fit a k-neearest neighbor classifier
# fit a KNN with 5 neighbors
y = final_Dataset["Loan_Status"]
knn = KNeighborsClassifier(n_neighbors=5, metric="euclidean",
                           n_jobs=-1).fit(final_Dataset[f], y)
# create search space of possible values of k
# create a pipeline
            # pipe = Pipeline([("standardizer", standardizer), ("knn", knn)])
pipe = Pipeline([("knn", knn)])
# create space of candidate values
search_space = [{"knn__n_neighbors":[2,3,4,5,6,7,8,9]}]
# search over possible values of k
# create grid search GridSearchCV impliments a "fit" and "score" method.
# the parameters of the estimator used to apply these methods are
# optimized by cross-validated grid-search over a parameter grid
clf = GridSearchCV(pipe, search_space, cv = 2, verbose = 1).fit(final_Dataset[f], y)
# view k for best performing model
# best neighborhood size (k)
k = clf.best_estimator_.get_params()["knn__n_neighbors"]
print("k =", k)
```

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
k = 3
[Parallel(n_jobs=1)]: Done  16 out of  16 | elapsed:   13.7s finished
```

We didn't standardize the data. The reason is that with standardized data, the value of k came 6 and with k = 6, the accuracy of KNN classifier becomes very less (0.37).But without standardizing the data the value of k comes 3. And accuracy of KNN with k = 3, is higher than accuracy of KNN with k = 6

108]:
```
# Training the K-NN with K as 3
# convert the data into np.arrays
# the scikit-learn library requires data to be formatted as numpy array.
X = train[f].as_matrix(columns=["ApplicantIncome", "CoapplicantIncome",
                                "LoanAmount", "Credit_History","Property_Area"])
y = np.array(train["Loan_Status"])
# train the learner
KNNmodel = neighbors.KNeighborsClassifier(k, weights="uniform") # k neighbors with same
                                                                # importance(weightage) to all
KNNmodel.fit(X, y)
# MODEL has been trained
preds = KNNmodel.predict(test[f])
acc_knn = KNNmodel.score(test[f], test["Loan_Status"])
print(acc_knn)
```

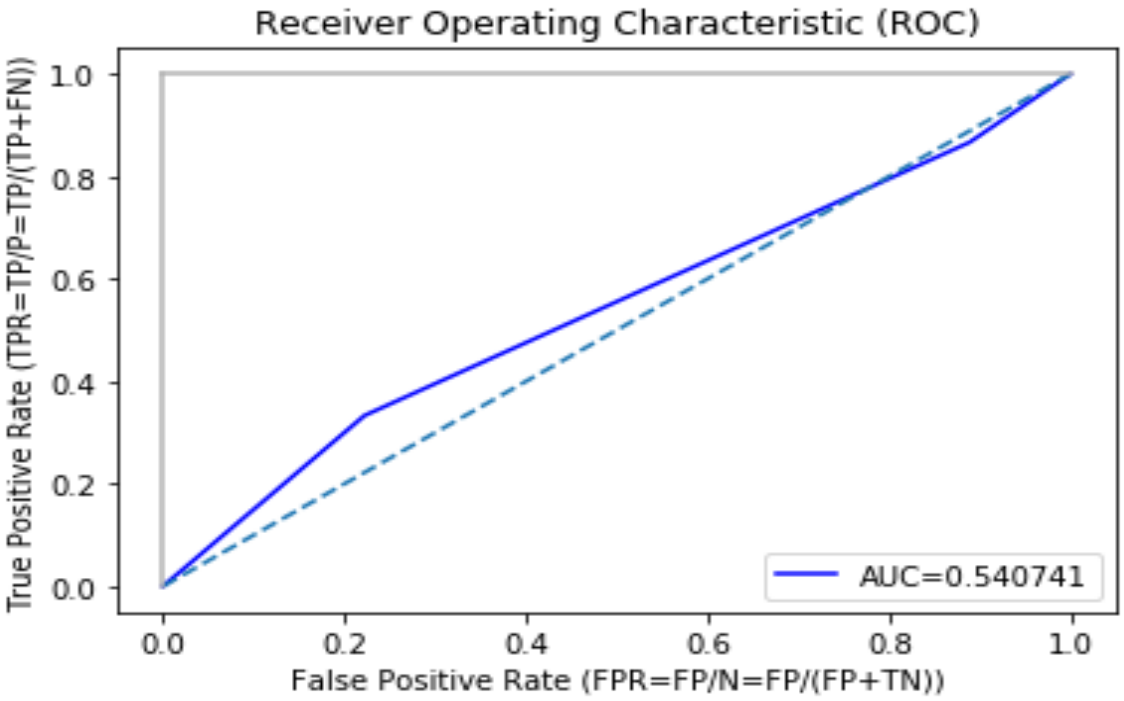
```
0.5833333333333334
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:5: FutureWarning: Method .as_matrix will be removed in a future version. Use .values instead.
  """
```

109]:
```
# Create confusion matrix
pd.crosstab(test["Loan_Status"], preds, rownames=["Actual Loan_Status"],
            colnames=["Predicted Loan_Status"])
```

109]:

Actual Loan_Status \ Predicted Loan_Status	0	1
0	1	8
1	2	13

119]:
```
# ROC graph for K-NN
y_score = KNNmodel.predict_proba(test[f])[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(test["Loan_Status"], y_score)
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc=auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c="0.7")
plt.plot([1, 1] , c="0.7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
#plt.show()
plt.savefig('roc_knn.png')
```



From the Reciever Operating Characterisitc(ROC) graph, we find the Area Under Curve(AUC) for our k-NN classifier model is 0.540741

AUC value of k-NN = 0.540741

106]:
```
# Cross-Validate Model Using Recall
print(cross_val_score(KNNmodel, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="recall"))
# Cross-validate model using precision
print(cross_val_score(KNNmodel, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="precision"))
```

```
[0.76923077 0.84615385 0.76923077 0.53846154 0.83333333]
[0.625      0.64705882 0.66666667 0.58333333 0.58823529]
```

107]:
```
print("Accuracy of k-NN classifier: ", acc_knn)
print(classification_report(test["Loan_Status"],preds))
```

```
Accuracy of k-NN classifier:  0.5833333333333334
              precision    recall  f1-score   support

           0       0.33      0.11      0.17         9
           1       0.62      0.87      0.72        15

   micro avg       0.58      0.58      0.58        24
   macro avg       0.48      0.49      0.44        24
weighted avg       0.51      0.58      0.51        24
```

Naive Bayes

We will now train a Gaussian Naive Bayes classifier model using our dataset

[31]:
```
# importing required modules
from sklearn.naive_bayes import GaussianNB
```

[120]:
```
Naive_Bayes = GaussianNB()
Naive_Bayes.fit(train[f],train["Loan_Status"])
predict = Naive_Bayes.predict(test[f])
acc_gnb = accuracy_score(test["Loan_Status"], predict)
print(acc_gnb)
print(Naive_Bayes)
```

```
0.9166666666666666
GaussianNB(priors=None, var_smoothing=1e-09)
```

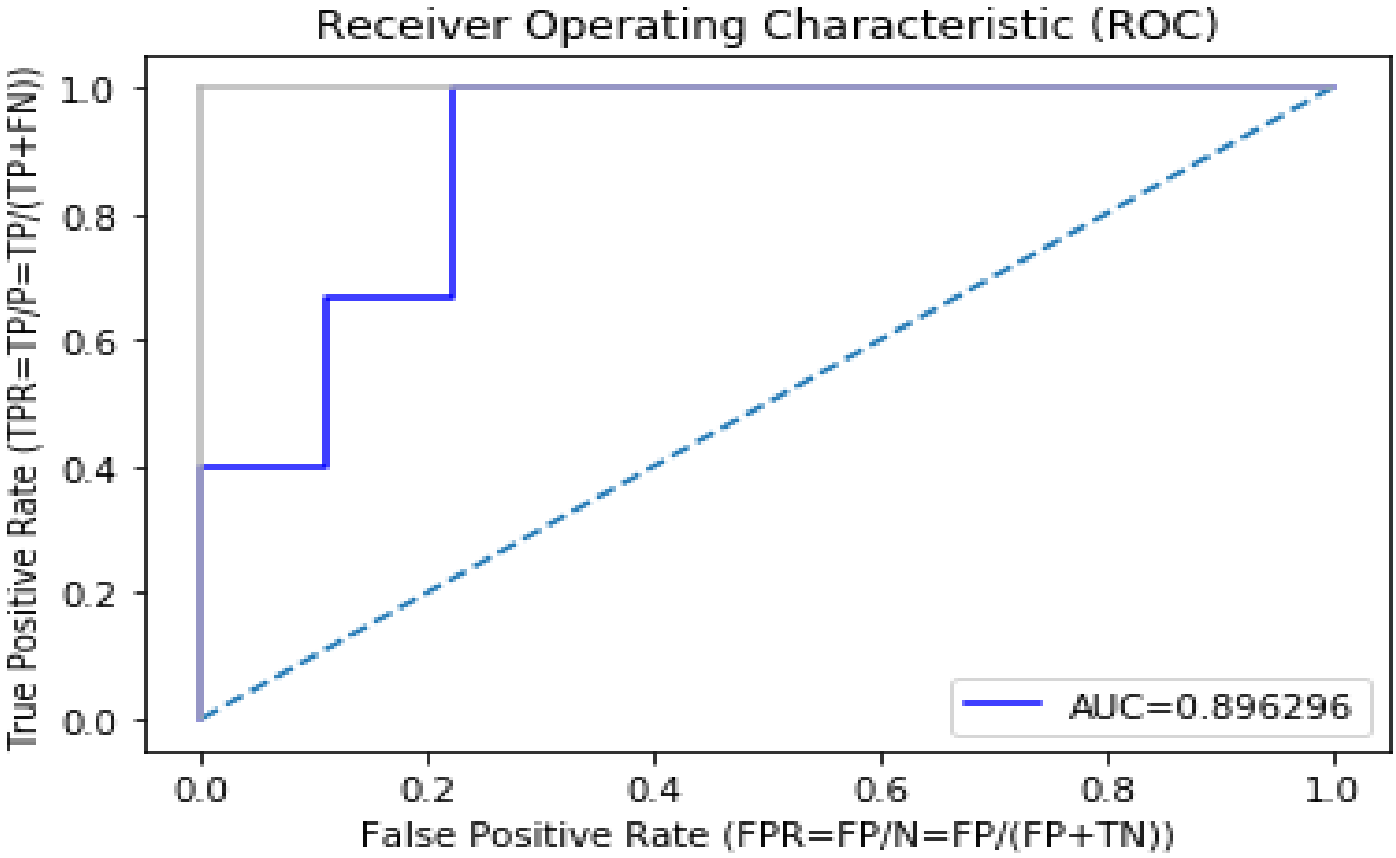
We can see that the accuracy of our Gaussian NB classifier is 91.667%.

[121]:
```
# Create confusion matrix
pd.crosstab(test["Loan_Status"], predict, rownames=["Actual Loan_Status"], colnames=["Predicted Loan_Status"])
```

[121]:

Actual Loan_Status \ Predicted Loan_Status	0	1
0	7	2
1	0	15

124]:
```
# ROC graph for Naive Bayes
y_score = Naive_Bayes.predict_proba(test[f])[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(test["Loan_Status"], y_score)
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc=auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
#plt.show()
plt.savefig('roc_gnb.png')
```



From the Recievер Operating Characterisitc(ROC) graph, we find the Area Under Curve(AUC) for our Gaussian NB classifier model is 0.896296

AUC value of Naive Bayes = 0.896296

[99]:
```
# Cross-Validate Model Using Recall
print(cross_val_score(Naive_Bayes, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="recall"))
# Cross-validate model using precision
print(cross_val_score(Naive_Bayes, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="precision"))
```

```
[0.92307692 0.92307692 1.         0.84615385 1.        ]
[0.92307692 0.85714286 0.8125     0.91666667 0.75      ]
```

[125]:
```
# Classificaton Report for our trained Naive Bayes classifier model
print("Accuracy: ", Naive_Bayes.score(test[f], test["Loan_Status"]))
print(classification_report(test["Loan_Status"],predict))
```

```
Accuracy:  0.9166666666666666
              precision    recall  f1-score   support

           0       1.00      0.78      0.88         9
           1       0.88      1.00      0.94        15

   micro avg       0.92      0.92      0.92        24
   macro avg       0.94      0.89      0.91        24
weighted avg       0.93      0.92      0.91        24
```

Decision Tree

[37]:
```
# importing required modules

from sklearn.tree import DecisionTreeClassifier
```

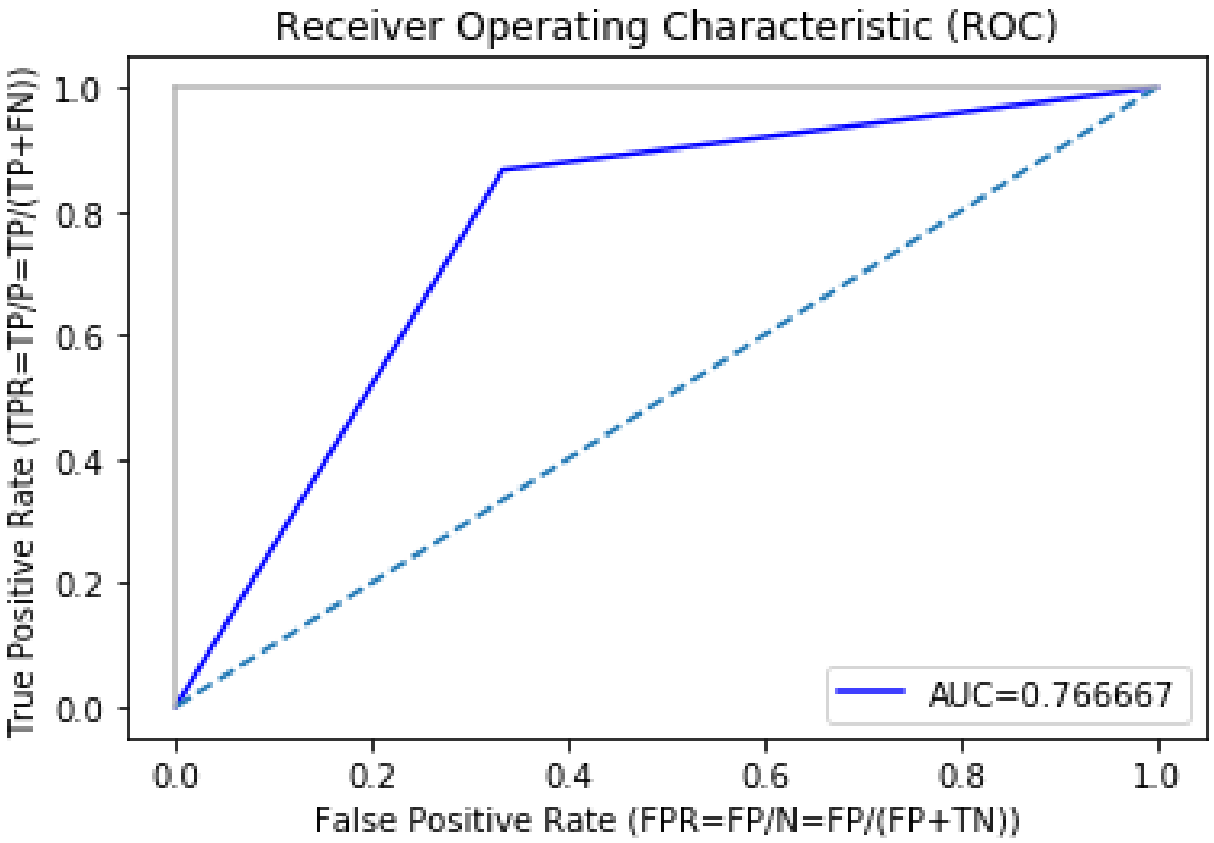
[126]:
```
Decision_Tree = DecisionTreeClassifier()
Decision_Tree.fit(train[f],train["Loan_Status"])
prediction = Decision_Tree.predict(test[f])
acc_dtree = accuracy_score(test["Loan_Status"], prediction)
print(acc_dtree)
Decision_Tree
```

```
0.7916666666666666
```

[126]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

[127]:
```
# to draw the Decision Tree

from sklearn import tree
with open("FinalDTree.txt", "w") as a:
    a = tree.export_graphviz(Decision_Tree, out_file=a)
```

129]:
```
# ROC graph for Decision Tree
y_score = Decision_Tree.predict_proba(test[f])[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(test["Loan_Status"], y_score)
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc=auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
#plt.show()
plt.savefig('roc_dtree.png')
```



From the Reciever Operating Characterisitc(ROC) graph, we find the Area Under Curve(AUC) for our Decision Tree classifier model is 0.766667

AUC value of Decision Tree = 0.76667

[82]:
```
# Cross-Validate Model Using Recall
print(cross_val_score(Decision_Tree, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="recall"))
# Cross-validate model using precision
print(cross_val_score(Decision_Tree, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="precision"))
```

```
[0.53846154 0.46153846 0.92307692 0.69230769 1.        ]
[0.77777778 0.75       0.75       0.81818182 0.71428571]
```

[130]:
```
# Classificaton Report for our trained Decision Tree classifier model
print("Accuracy: ", Decision_Tree.score(test[f], test["Loan_Status"]))
print(classification_report(test["Loan_Status"],prediction))
```

```
Accuracy:  0.7916666666666666
              precision    recall  f1-score   support

           0       0.75      0.67      0.71         9
           1       0.81      0.87      0.84        15

   micro avg       0.79      0.79      0.79        24
   macro avg       0.78      0.77      0.77        24
weighted avg       0.79      0.79      0.79        24
```

Logistic Regression Model

[44]:
```
# importing required modules

from sklearn.linear_model import LogisticRegression
```

[131]:
```
LogisticRegressionModel = LogisticRegression(solver='lbfgs')
LogisticRegressionModel.fit(train[f],train["Loan_Status"])
print("The intercept for the model is:",LogisticRegressionModel.intercept_)
print("The coefficients for the model is:",LogisticRegressionModel.coef_)
predictions = LogisticRegressionModel.predict(test[f])
acc_lr = accuracy_score(test["Loan_Status"], predictions)
print(acc_lr)
LogisticRegressionModel
```

```
The intercept for the model is: [-0.62505495]
The coefficients for the model is: [[-1.90032183e-04 -6.94385392e-05 -1.41847856e-03  1.50926139e+00
   7.27142583e-01]]
0.8333333333333334
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converge. In
crease the number of iterations.
  "of iterations.", ConvergenceWarning)
```

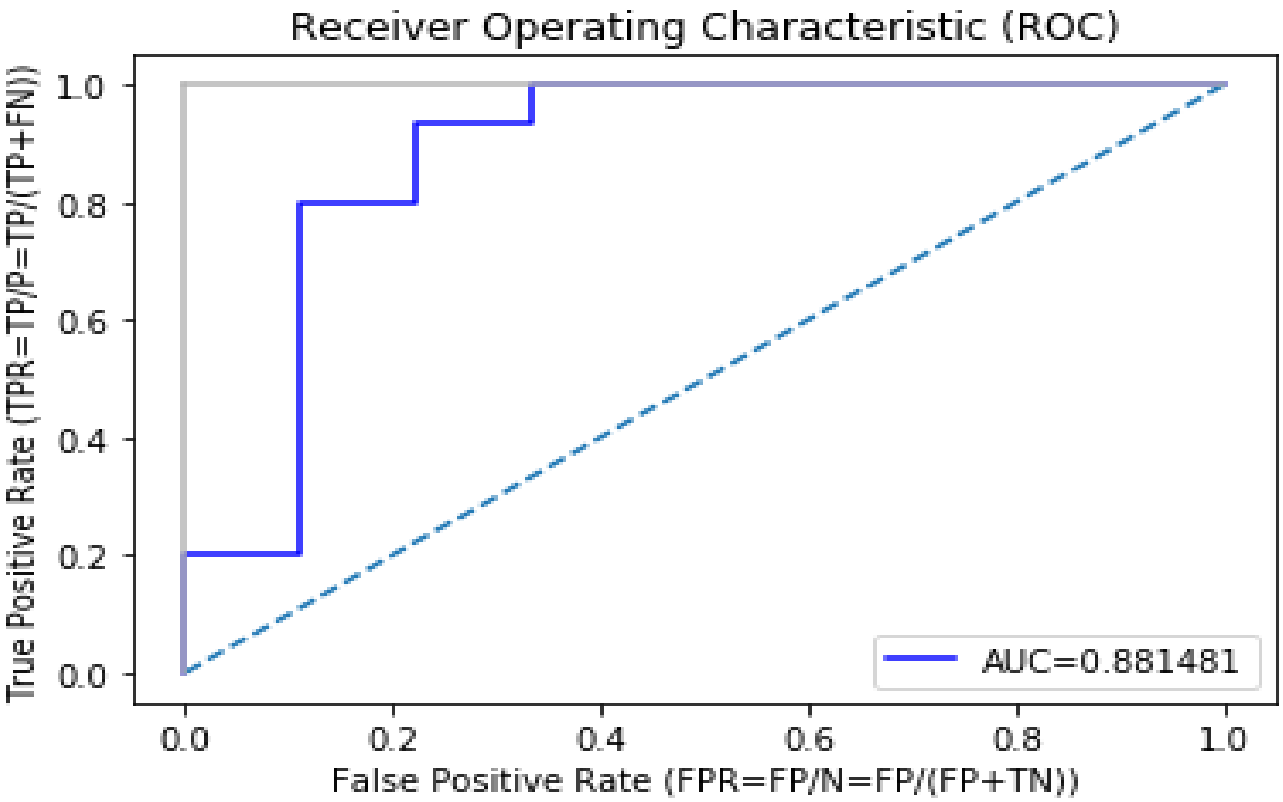
[131]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

[132]:
```
# Create confusion matrix
pd.crosstab(test["Loan_Status"], predictions, rownames=["Actual Loan_Status"], colnames=["Predicted Loan_Status"])
```

[132]:

Actual Loan_Status \ Predicted Loan_Status	0	1
0	6	3
1	1	14

[133]:
```
# ROC graph for Logistic Regression
y_score = LogisticRegressionModel.predict_proba(test[f])[:,1]
false_positive_rate, true_positive_rate, threshold = roc_curve(test["Loan_Status"], y_score)
plt.title('Receiver Operating Characteristic (ROC)')
roc_auc=auc(false_positive_rate,true_positive_rate)
plt.plot(false_positive_rate, true_positive_rate,'b',label="AUC=%f"%roc_auc)
plt.legend(loc='lower right')
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7")
plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate (TPR=TP/P=TP/(TP+FN))')
plt.xlabel('False Positive Rate (FPR=FP/N=FP/(FP+TN))')
#plt.show()
plt.savefig('roc_lr.png')
```



```
From the Reciever Operating Characterisitc(ROC) graph, we find the Area Under Curve(AUC) for our Logistic Regression classifier model is 0.881481

AUC value of Logistic Regression classifier = 0.881481
```

[134]:
```
# Cross-Validate Model Using Recall
print(cross_val_score(LogisticRegressionModel, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="recall"))
# Cross-validate model using precision
print(cross_val_score(LogisticRegressionModel, final_Dataset[f], final_Dataset["Loan_Status"], cv=5, scoring="precision"))
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
[0.92307692 0.92307692 1.         0.84615385 1.        ]
[0.75       0.75       0.8125     0.91666667 0.75      ]
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:758: ConvergenceWarning: lbfgs failed to converge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

[135]:
```
# RMSE for Logistic Regression
lin_mse = mean_squared_error(predictions, test["Loan_Status"])
lin_rmse = np.sqrt(lin_mse)
print('Logistic Regression RMSE: %.4f' % lin_rmse)
```

```
Logistic Regression RMSE: 0.4082
```

[136]:
```
# R-squared for Logistic Regression
print ("Prediction : ",predictions)
print('Logistic Regression R squared: %.4f' % LogisticRegressionModel.score(test[f], test["Loan_Status"]))
```

```
Prediction :  [1 1 0 1 1 1 1 0 1 1 1 1 1 1 0 1 0 0 1 0 1 0 1 1]
Logistic Regression R squared: 0.8333
```

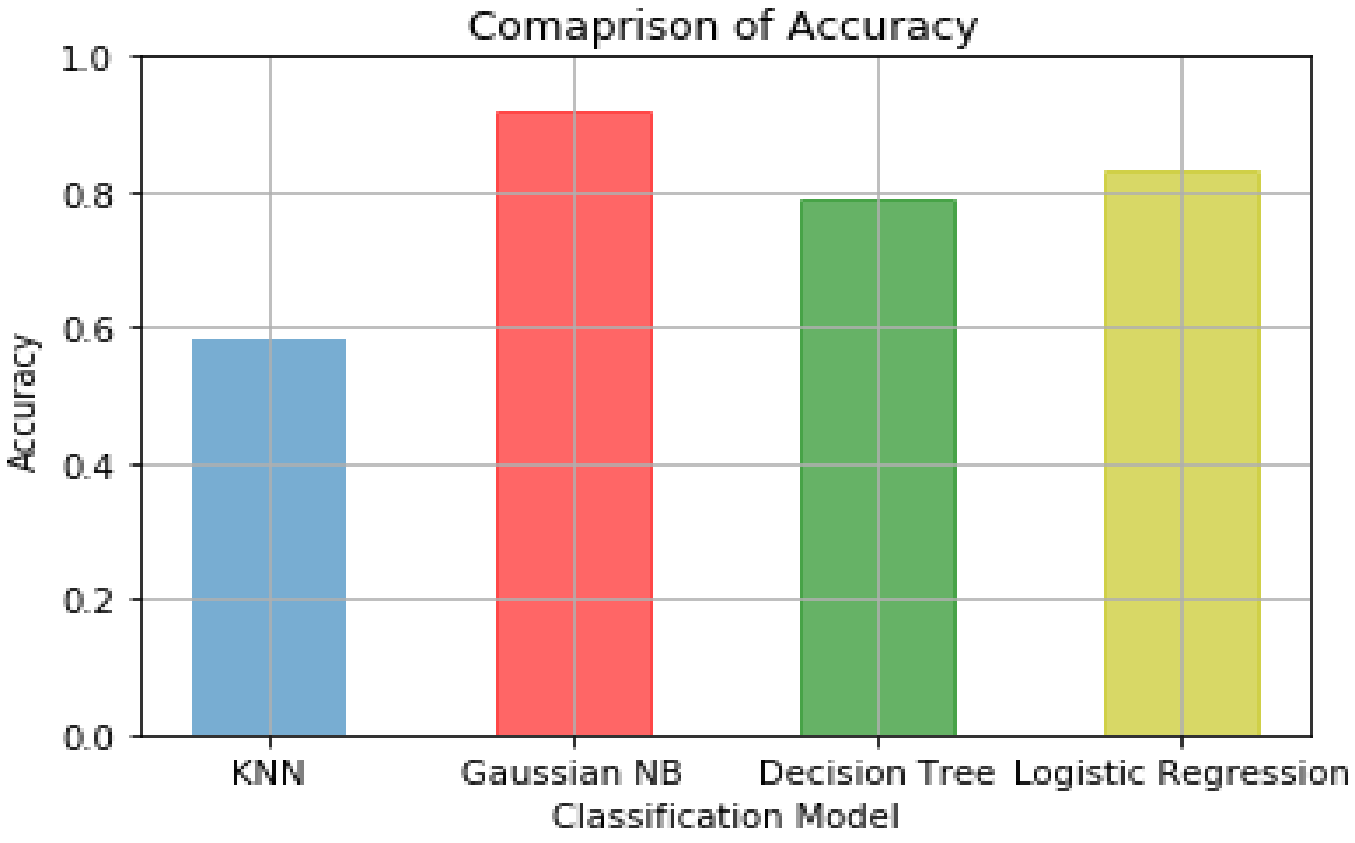
[137]:
```
# Classificaton Report for our trained Logistic Regression classifier model
print("Accuracy: ", LogisticRegressionModel.score(test[f], test["Loan_Status"]))
print(classification_report(test["Loan_Status"],predictions))
```

```
Accuracy:  0.8333333333333334
              precision    recall  f1-score   support

           0       0.86      0.67      0.75         9
           1       0.82      0.93      0.87        15

   micro avg       0.83      0.83      0.83        24
   macro avg       0.84      0.80      0.81        24
weighted avg       0.84      0.83      0.83        24
```

[138]:
```
# printing the accuracy scores of all the 4 models:
print("KNN:", acc_knn)
print("GNB:", acc_gnb)
print("LR:", acc_lr)
print("DTree:", acc_dtree)
```

```
KNN: 0.5833333333333334
GNB: 0.9166666666666666
LR: 0.8333333333333334
DTree: 0.7916666666666666
```

[140]:
```
# comapring accuracies of 4 models using barplot
modelss = ["KNN", "Gaussian NB", "Decision Tree", "Logistic Regression"]
accuracies = [acc_knn, acc_gnb, acc_dtree, acc_lr]
barlist = plt.bar(modelss, accuracies, width=0.5, alpha=0.6)
plt.ylim(0, 1.0)
barlist[1].set_color('r')
barlist[2].set_color('g')
barlist[3].set_color('y')
plt.xlabel('Classification Model')
plt.ylabel('Accuracy')
plt.title('Comaprison of Accuracy')
plt.grid()
#plt.show()
plt.savefig('accu_models.png')
```



Test Dataset:

Now we will re-train the 4 models using 100% of our train dataset and use the models to predict the target attribute for our given test dataset. We have to pre-process the test dataset in the same manner as we pre-processed our train dataset to remove any miscalculations.

[54]:
```
# laoding the test dataset
Data = pd.read_csv("test.csv")
# making a copy of the main Data to manipulate
X = Data.copy()
# viewing the numeric dataset
Data.describe(include="all")
```

[54]:

	Application_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
count	514.000000	502	511	499	514	488	514.000000	514.000000	497.000000	505.000000	472.000000	514
unique	NaN	2	2	4	2	2	NaN	NaN	NaN	NaN	NaN	3
top	NaN	M	Yes	0	Graduate	No	NaN	NaN	NaN	NaN	NaN	Semiurban
freq	NaN	405	329	285	403	419	NaN	NaN	NaN	NaN	NaN	192
mean	2163.075875	NaN	NaN	NaN	NaN	NaN	5652.608949	1605.816965	148.742455	342.059406	0.843220	NaN
std	467.056621	NaN	NaN	NaN	NaN	NaN	6574.855143	3081.978215	89.057125	65.870517	0.363979	NaN
min	1345.000000	NaN	NaN	NaN	NaN	NaN	150.000000	0.000000	9.000000	12.000000	0.000000	NaN
25%	1760.250000	NaN	NaN	NaN	NaN	NaN	2894.250000	0.000000	100.000000	360.000000	1.000000	NaN
50%	2150.000000	NaN	NaN	NaN	NaN	NaN	3862.000000	1031.000000	128.000000	360.000000	1.000000	NaN
75%	2543.750000	NaN	NaN	NaN	NaN	NaN	6000.000000	2229.750000	170.000000	360.000000	1.000000	NaN
max	2990.000000	NaN	NaN	NaN	NaN	NaN	81000.000000	41667.000000	700.000000	480.000000	1.000000	NaN

[55]:
```
# dropping the values which are not the principal attributes
for i in X.columns:
    if i not in f:
        X = X.drop([i], axis=1)
X
```

[55]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
0	4288	3263.0	133.0	1.0	Urban
1	4843	3806.0	151.0	1.0	Semiurban
2	13650	0.0	NaN	1.0	Urban
3	4652	3583.0	NaN	1.0	Semiurban
4	3816	754.0	160.0	1.0	Urban
5	3052	1030.0	100.0	1.0	Urban
6	11417	1126.0	225.0	1.0	Urban
7	7333	0.0	120.0	1.0	Rural
8	3800	3600.0	216.0	0.0	Urban
9	2071	754.0	94.0	1.0	Semiurban
10	5316	0.0	136.0	1.0	Urban
11	2929	2333.0	139.0	1.0	Semiurban
12	3572	4114.0	152.0	0.0	Rural
13	7451	0.0	NaN	1.0	Semiurban
14	5050	0.0	118.0	1.0	Semiurban
15	14583	0.0	185.0	1.0	Rural
16	3167	2283.0	154.0	1.0	Semiurban
17	2214	1398.0	85.0	NaN	Urban

[56]:
```
X.describe(include="all")
```

[56]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
count	514.000000	514.000000	497.000000	472.000000	514
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	Semiurban
freq	NaN	NaN	NaN	NaN	192
mean	5652.608949	1605.816965	148.742455	0.843220	NaN
std	6574.855143	3081.978215	89.057125	0.363979	NaN
min	150.000000	0.000000	9.000000	0.000000	NaN
25%	2894.250000	0.000000	100.000000	1.000000	NaN
50%	3862.000000	1031.000000	128.000000	1.000000	NaN
75%	6000.000000	2229.750000	170.000000	1.000000	NaN
max	81000.000000	41667.000000	700.000000	1.000000	NaN

[57]:
```
# making the non numeric values numeric
X.Property_Area.replace(['Rural','Semiurban', 'Urban'], [0, 1, 2], inplace=True)
X.dtypes
```

[57]:
```
ApplicantIncome       int64
CoapplicantIncome   float64
LoanAmount          float64
Credit_History      float64
Property_Area         int64
dtype: object
```

[58]:
```
# filling na values in credit history with 0 (as stated in the ppt)
X.Credit_History.fillna(0, inplace=True)
# filling NaN values with mean +- std in float type columns
avg = X.LoanAmount.mean()
std = X.LoanAmount.std()
count = X.LoanAmount.isnull().sum()
random = np.random.randint(avg-std,avg+std,size=count)
X['LoanAmount'][np.isnan(X['LoanAmount'])]=random
X.describe(include="all")
```

[58]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
count	514.000000	514.000000	514.00000	514.000000	514.000000
mean	5652.608949	1605.816965	148.66537	0.774319	0.964981
std	6574.855143	3081.978215	87.94022	0.418437	0.791487
min	150.000000	0.000000	9.00000	0.000000	0.000000
25%	2894.250000	0.000000	100.00000	1.000000	0.000000
50%	3862.000000	1031.000000	128.00000	1.000000	1.000000
75%	6000.000000	2229.750000	171.50000	1.000000	2.000000
max	81000.000000	41667.000000	700.00000	1.000000	2.000000

[59]:

```
# handling outliers with MAD

mad=1.4826*np.median(np.abs(X.ApplicantIncome-X.ApplicantIncome.median()))
size_outlr_mad=X.ApplicantIncome
size_outlr_mad[((X.ApplicantIncome-X.ApplicantIncome.median()).abs()>3*mad)]=X.ApplicantIncome.median()
print("ApplicantIncome MAD :\n",size_outlr_mad.median())

mad=1.4826*np.median(np.abs(X.CoapplicantIncome-X.CoapplicantIncome.median()))
size_outlr_mad=X.CoapplicantIncome
size_outlr_mad[((X.CoapplicantIncome-X.CoapplicantIncome.median()).abs()>3*mad)]=X.CoapplicantIncome.median()
print("CoapplicantIncome MAD :\n",size_outlr_mad.median())

mad=1.4826*np.median(np.abs(X.LoanAmount-X.LoanAmount.median()))
size_outlr_mad=X.LoanAmount
size_outlr_mad[((X.LoanAmount-X.LoanAmount.median()).abs()>3*mad)]=X.LoanAmount.median()
print("LoanAmount MAD :\n",size_outlr_mad.median())
X.describe(include="all")
```

```
ApplicantIncome MAD :
 3860.5
CoapplicantIncome MAD :
 1030.5
LoanAmount MAD :
 128.0
```

[59]:

	ApplicantIncome	CoapplicantIncome	LoanAmount	Credit_History	Property_Area
count	514.000000	514.000000	514.000000	514.000000	514.000000
mean	4067.149805	1209.268327	130.367704	0.774319	0.964981
std	1743.417951	1379.440909	46.348698	0.418437	0.791487
min	150.000000	0.000000	9.000000	0.000000	0.000000
25%	2894.250000	0.000000	100.000000	1.000000	0.000000
50%	3860.500000	1030.500000	128.000000	1.000000	1.000000
75%	4836.500000	2082.000000	157.750000	1.000000	2.000000
max	9538.000000	5500.000000	260.000000	1.000000	2.000000

Logistic Regression

[60]:
```
LogisticRegressionModel.fit(final_Dataset[f],final_Dataset["Loan_Status"])
predictions = LogisticRegressionModel.predict(X[f])
#print("1:", predictions.count('1'), "\n0:", predictions.count('0'))
len(predictions)
```

[60]: 514

Naive Bayes

[61]:
```
Naive_Bayes.fit(final_Dataset[f],final_Dataset["Loan_Status"])
predict = Naive_Bayes.predict(X[f])
len(predict)
```

[61]: 514

KNN

[62]:
```
KNNmodel.fit(final_Dataset[f],final_Dataset["Loan_Status"])
preds = KNNmodel.predict(X[f])
len(preds)
```

[62]: 514

Decision Tree

[63]:
```
Decision_Tree.fit(final_Dataset[f],final_Dataset["Loan_Status"])
prediction = Decision_Tree.predict(X[f])
len(prediction)
```

[63]: 514

[64]:
```
print(sum(1 for i,j,k,l in zip(predict,prediction,predictions, preds) if i==j==k==l))
```

213

Variable Name	Model Name
predictions	Logistic Regression
predict	Naive Bayes
prediction	Decision Tree
preds	KNN

[67]:
```
unique, counts = np.unique(predictions, return_counts = True)
dict(zip(unique, counts))
```

[67]: {0: 165, 1: 349}

[68]:
```
unique, counts = np.unique(predict, return_counts = True)
dict(zip(unique, counts))
```

[68]: {0: 247, 1: 267}

[69]:
```
unique, counts = np.unique(prediction, return_counts = True)
dict(zip(unique, counts))
```

[69]: {0: 259, 1: 255}

[70]:
```
unique, counts = np.unique(preds, return_counts = True)
dict(zip(unique, counts))
```

[70]: {0: 142, 1: 372}

Future Scope of Improvement

- Various banking institution can use these models and modify them according to their needs to use in their loan approval status. This will reduce the manual labour and time spent on determining whether to approve a loan application.
- Customers who intend to take a loan can use these trained models to check whether their loan application will be approved or not. The trained models would be required to be implemented in a platform or interface easily accessible as well as with an easy GUI.
- We saw a high value of correlation of "Married" attribute with our target attribute. But the feature importance of "Married" attribute was significantly lower. With more data and further analysis, it might be possible to describe the reason of this mismatch.
- No loan application having value "Rural" in "Property_Area" attribute was approved. With further research and more data for analysis, a more decisive conclusion can be made.

Certificate

This is to certify that Mr. Debaleen Das Spandan of Bengal College of Engineering and Technology, registration number: 171250110041, has successfully completed a project on *Prediction of Loan Approval Status* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Mr. Shouvit Pradhan of Bengal College of Engineering and Technology, registration number: 171250110093, has successfully completed a project on *Prediction of Loan Approval Status* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Mr. Aditya Som Choudhury of Techno India Main Salt lake, registration number: 161300110620, has successfully completed a project on *Prediction of Loan Approval Status* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Mr. Amartya Paul of BP Poddar Institute of Management and Technology, registration number: 161150110011, has successfully completed a project on *Prediction of Loan Approval Status* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School

Certificate

This is to certify that Mr. Palash Mondal of Techno India Saltlake, registration number: 181300120082, has successfully completed a project on *Prediction of Loan Approval Status* using *Machine Learning with Python* under the guidance of Prof. Arnab Chakraborty.

Prof. Arnab Chakraborty

Globsyn Finishing School