Major Project Report On

"B-AMIGO (The Prediction Baba)"

Submitted in the fulfillment of the requirements For
the Degree of
Bachelor of Technology Semester VIII in
Electronics & Telecommunication Engineering By

Animesh Shukla	**1814110504**
Kanupriya Saxena	**1814110503**

Under the guidance of
Prof. Dr RB Ghongade

Department of Electronics & Telecommunication Engineering
Bharati Vidyapeeth (Deemed to Be University)
College of Engineering, Pune – 4110043

Academic Year 2021-22

BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)
COLLEGE OF ENGINEERING, PUNE – 4110043

DEPARTMENT OF ELECTRONICS & TELECOMMUNICATION ENGINEERING

CERTIFICATE

*This is to certify that **ANIMESH SHUKLA** and **KANUPRIYA SAXENA** of final year Electronics and Telecommunication have submitted the Major Project report entitled, "**B-AMIGO (The Prediction Baba)**" in fulfillment of the requirement for the award of Bachelor of Technology from Bharati Vidyapeeth (Deemed to be) University, Pune, Session 2018-22. It has been found to be satisfactory and hereby approved for the submission.*

Date:

Prof. RB Ghongade
(Project Guide)

Prof. S.K. Oza
(HOD)

Examiner 1:

Examiner 2:

Acknowledgement

11th July '22

We would like to covet our gratitude to our project guide Prof. Dr. RB Ghongade for his tremendous assistance and support throughout. The innovative ideas, the expansion of our thought process, and the motivation was all possible under his guidance.

We would like to extend our sincere gratitude to the Principal Dr. Vidula Sohoni, Head of Department Electronics & Telecommunication Prof. S.K. Oza for nurturing a congenial yet competitive environment, which motivates all the students to pursue their goals.

Our team would also like to thank the project coordinator Prof. Deepak Ray and all the other professors to give us this opportunity to work on the project B-Amigo (The Prediction Baba). At the end, we thank our parents and friends for their moral support and insights.
Animesh Shukla (PRN: 1814110504)

Kanupriya Saxena (PRN: 1814110503)

Table Of Contents

LIST OF FIGURES

Objective

In the world of Digitalization, the banking industry also needs to be evolved in terms of growing technology. Our project titled as “B-Amigo” () is a Banking Software, designed for the management of banks and other financial institutions involved in loans and payments. The tool is an amalgamation of two broad concepts i.e., Machine Learning and App Development.

The software consists of four prediction models that fulfil the objectives of:

- customer retention
- loan approval
- loan repayment
- detection and prevention of fraudulent transactions

B-Amigo’s design parameters make it ideal for various predictions like whether a customer would leave the services of the bank or not and so on.

The main focus of this idea is to automate tedious operations, going nearly paperless, reducing chances of error that ultimately results in better ties of banks and their customers.

Rationale

All over the world, banks are taking a huge step towards digitalization in order to cope up with the competition and deliver the maximum to its customers. Digitalization has transformed the manual process into digital service by reducing human error and thus, saving time and building customer loyalty. For Example, the customers have been facilitated by the technology of cashless transactions, online banking and what not.

On realising that the technology keeps on evolving, our project “B-Amigo” (The Prediction Baba) is built on one of the booming technologies, Artificial intelligence that helps in faster and efficient decision making by mimicking the human brains.

It helps in analysing the customers for a bank’s better performance by making predictions beforehand in certain aspects. The model ultimately saves time, unnecessary workload and lengthy duties.

Problem Statement

While coming across several banks, we learnt about the lengthy duties and unnecessary workload they come across in their daily tasks.

And since, the digitalization has surged in almost all sectors, why not we make the bank officials’ work easy by developing a tool that helps them in some or the other ways and redefine the working style of the sector? Banking is all about the relationship between a bank employee and a customer. Hence, answering basic questions about the customers, for example, Will the customer continue with the bank services? Or will he/she be able to repay the loan? etc. on the basis of certain data, help a manager works more efficiently and faster than the manual approach.

Introduction

Our project "B-Amigo" (The Prediction Baba) is a software application built for the banking sector. The desired prediction can be made by the user (bank manager) using the customers' data. The technologies used in the successful completion of "B-Amigo" are as follows:

- Data Science
- Machine Learning
- Android Application Development
- Cloud Computing
- Data base Integration

It consists of four features:

1. ***Loan Repayment:*** In this feature, the problem statement "**Will the customer be able to repay the granted loan or not?**" is answered. For example, suppose a customer has been sanctioned a home loan of 25 lakhs, and while he is repaying the same, he requests for a personal loan (say 30 Lakhs) too.

- In such situations, the bank manager needs to verify if the customer will be able to bear the repayment of 55lakhs,
- And hence to reconfirm this, the feature "loan repayment" comes into the picture, which when given data of the customer, predicts whether the person can repay the loan or not based on certain factors.
- Let's say the feature predicts a "Yes" i.e., the customer can repay the loan to the bank, the output "**REPAYMENT will be successful**" will be displayed with the probability percentage of repayment of the loan.

---

2. ***Loan Approval:*** Sometimes it's difficult for the employees to come to a consensus, on whether the customer who has requested the bank for a loan should be approved or not. Hence, to answer the question **"Should the bank approves the loan request of a customer or not?"** the feature "Loan Approval" is added to the software application.

- Let's take the example of Mr. Ram who is a maths professor and has requested his bank for a home loan of 60 lakhs.
- In this case, to make sure whether the customer can be sanctioned with the loan or not, the bank manager uses the feature "Loan Approval" of our software application 'B-Amigo'
- The bank will use the data that has been saved in their systems to determine if the candidate is suitable for the loan or not.
- Let's say, the feature predicts a "NO" i.e., the bank cannot approve the loan for the customer based on the factors that are considered by the app, the output "**Loan should NOT be approved**" is displayed on the screen with a probability percentage of the same.

3. ***Customer Retention***: The retention of the customers is very crucial for a bank as they only rate the success of a bank. So, to maintain their reputation, the banks pay a lot of attention to the retention of their customers. Hence, the analysis of "**Whether the present customer of the bank will continue with the services or not?**" becomes necessary.

- For example, a customer Mr. Gopal has been using the services of a bank for (say) 5 years.
- In this case, to check if the customer will be retained or not, the bank uses the "Customer Retention" module of B-Amigo.
- The bank uses the customer's data like Credit Score, Gender, Bank Balance, etc. to determine the most accurate results for the same problem statement.
- Let's say, the application predicts a "YES" i.e., the probability of the current customer staying back and continuing using the bank's services is high, the output "**customer would STAY**" is displayed.
- Since the predictions are also followed by a probability percentage, the analysis of the customers' retentions becomes crisper.

4. ***Fraudulent Transactions***: In the payment industry, fraud is the illegal usage of credit card details without the real cardholder's knowledge. Hence, the module "Fraudulent Transaction" attempts for online transaction **fraud detection and prevention**.

- For example, A person informs a bank about a transaction that has not been made by him but shows in his online transaction history.
- The bank recovers the customer's data and uses the feature "Fraudulent Transaction" of the B-Amigo application to detect whether the transaction made was fraud or not.
- Let's say, the application predicts a "YES" i.e., the probability of the investigated transaction being a fraud is high, the output "NORMAL TRANSACTION" is displayed.
- Since, the application also has the property of mentioning the probability percentage, the detection of whether the transaction was fraud or not becomes clearer.

Chapter-1: Dealing with Missing Values:

The very first process to proceed with any Data Science Project is to start dealing with the missing values. After obtaining the raw data, we inspect if the data has some missing values in it. In real world, there would hardly be any situation when you don't have to deal with the missing data. Hence, it is quite an important step for your Project's life cycle since the way you deal with the missing data defines how well your Machine Learning Models would perform, since it directly affects the quality of our data.

In some cases, when you have no missing values in your data, then it is well and good: you can directly proceed with the other Data Preprocessing steps which would be talked about in the next chapter [link to Chapter-2]. Apparently, there exists very few cases when you would be getting the whole data and in most of the scenarios, you would have to deal with the missing data.

Dealing with Missing Data depends upon the Measurement of Data:

Let us understand how we dealt with the missing values in one of our problem statements "Loan Approval" where there were a lot of missing values and would further demonstrate each and every detail of it.

The tabular data which we generally work with has two types of Features (i.e. column names of the data we have):

- Categorical Features.
- Numerical Features.

Figure 1.0 Data of Loan Deafaulter

Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area
Male	No	0	Graduate	No	5849	0.0	NaN	360.0	1.0	Urban
Male	Yes	1	Graduate	No	4583	1508.0	128.0	360.0	1.0	Rural
Male	Yes	0	Graduate	Yes	3000	0.0	66.0	360.0	1.0	Urban
Male	Yes	0	Not Graduate	No	2583	2358.0	120.0	360.0	1.0	Urban
Male	No	0	Graduate	No	6000	0.0	141.0	360.0	1.0	Urban

The Categorical Features have level of measurement of Nominal or Ordinal scale. In simple terms, categorical features are the ones which generally have categories less than 10. However, the numerical Features have level of measurement of interval or ratio scale.

Figure 1.0 depicts data of customers for a particular bank and there exists features such as "Gender", "Married", "Dependents", "Education", "ApplicantIncome", etc. Now, it can be observed that feature "Gender" has only 2 categories i.e. "Male" and "Female". Thus, this feature is categorical in nature where as if we consider features such as "ApplicantIncome", it has continuous numerical values, thus it is a Numerical Feature. **Figure 1.1**

Figure 1.1 Categorical and Numerical Features of Data along with the code

```
# columns name of orig_data:
feats_orig = list(orig_data.columns)
cat_feats_orig = []
num_feats_orig = []

# distincting features:
for feature in feats_orig:
    if orig_data[feature].nunique() > 10:
        num_feats_orig.append(feature)
    else:
        cat_feats_orig.append(feature)

print('Categorical Feature: ', cat_feats_orig, end='\n\n')
print('Numerical Features: ', num_feats_orig)
```

```
Categorical Feature:  ['Gender', 'Married', 'Dependents', 'Education', 'Self_Employed', 'Loan_Amount_Term', 'Credit_Histo
'Property_Area', 'Loan_Status']

Numerical Features:  ['ApplicantIncome', 'CoapplicantIncome', 'LoanAmount']
```

shows the numerical and categorical features of data shown in **Figure 1.0** along with the code in Python.

After we have understood what Numerical and Categorical features are, now we should understand how to deal with missing values of each of them.

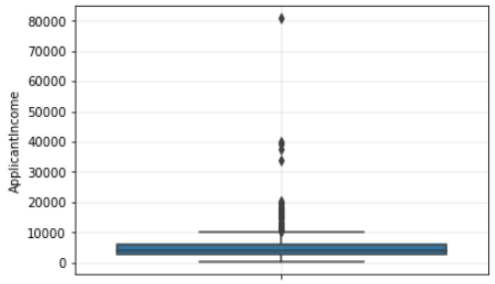
Missing Values of Numerical Features:

In order to start dealing with missing values, we must inspect the proportion of missing values in our data for each of the features. We must decide the technique for dealing with the missing values based upon the proportion of missing values. The techniques are listed as follows:

- ***If Missing Values are less than 5% of whole data:***

For example, if we consider data which has 200 rows and 2 columns: one of the columns has 5 missing values which accounts for 2.5% (hence less than 5% of the whole data) of the whole data. Hence, in such a case we might inspect the probability density function of that numerical feature and if there exist no outliers for that feature, then we could compute the mean for that numerical feature and replace all the missing values with the mean: but if outliers seem to be present, then instead of computing mean, we must compute median (since it is not affected by the outliers) and replace all the missing values with the median.

Figure 1.2 Boxplot of numerical feature, "ApplicantIncome"



We could simply observe the Boxplot of the numerical feature whose missing values are to be dealt with and inspect if the outliers seem to be there. From **Figure 1.2**, we can inspect the boxplot of numerical feature, "ApplicantIncome". There seems to occur a lot of data points lying above the 100-percentile whisker which are nothing but outliers. Thus, if the feature has missing values less than 5% of the whole data, then we must compute median (since outliers seem to be present) of this feature and replace all the missing values with the median computed.

Figure 1.3, Data Points in Original Data and Reference Data

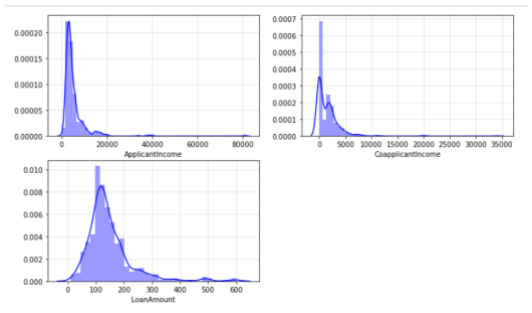
```
: print('total number of rows in Original Data -->', len(orig_data))
  total number of rows in Original Data --> 614

: print('total number of rows in reference Data -->', len(no_null_data))
  total number of rows in reference Data --> 480
```

- ***If missing values are greater than 5% of the whole data:***

Now, it seems to be a bit trivial task of how we can deal with the missing values of a numerical feature if they are large in number. In such a case, we can use the power of Regression to deal with such large proportion of missing values. Since, we would be using Linear Regression for this task, hence we must also verify the assumptions of Linear Regression.

Figure 1.4, Distribution of Numerical Features



First of all, we drop all of the missing values from our original data and consider this data where no missing values are present as the **reference data**. We can clearly observe from the **Figure 1.3**, there exists missing value in the original data where number of rows are 614 but after dropping all of the missing values from our data i.e. in reference data(exists no missing values) number of rows are now reduced to 480 (since rows of missing values are dropped). Now, there are only 3 numerical features in "Loan Approval" problem statement, hence we must observe the Probability Density Function of each of the numerical features. **Figure 1.4** represents the Probability Density function of the 3 numerical features which seem more or less like Exponentially Distributed. Generally, it's a good practice if the distribution of numerical features is normal/gaussian and in order to bring data from Exponential Distribution to Gaussian Distribution, we can perform Logarithmic Transformation to original data. **Figure 1.5** represents the log transformation applied to each of the numerical features and now the distribution seems to be more like normal. Numerical feature "ApplicantIncome" and "LoanAmount" now apparently seem to have Gaussian Distribution.

Figure 1.5, Distribution of Logarithmic Transformation of Numerical Features

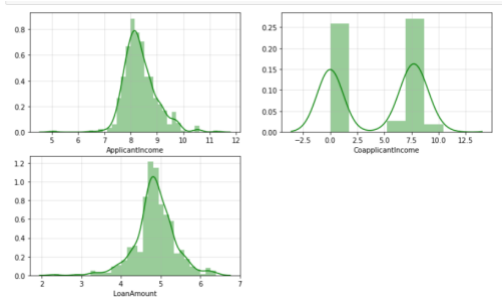
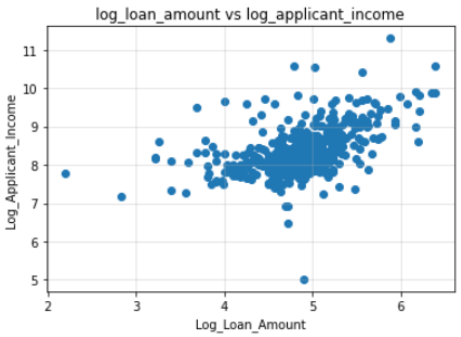


Figure 1.6 shows a scatter plot plotted between "log_loan_amount" and "log_applicant_income" and there seems to be a linear relationship existing between these 2 features. Now, suppose if "log_loan_amount" has a large proportion of missing values and no missing values exist for "log_applicant_income", then we can treat "log_loan_amount" as Dependent or Target Variable and "log_applicant_income" as Independent Variable and train a Regression Model. Further, we can use this Regression model

Figure 1.6, Scatter Plot between 'log_loan_amount' and 'log_applicant_income'



to make predictions of those values for which "log_loan_amount" had missing values in the original data. Thus, we would be training the model on reference data and would be making predictions in the original data for which "log_loan_amount" had missing values. In this way, we would be able to deal with the missing of "log_loan_amount".

Analysis of Regression Model:

Analyzing Regression models are quite the key to make successful predictions. We also need to verify the Assumptions of Linear Regression before making predictions on the missing values of Dependent Variable, "log_loan_amount". **Figure 1.7** depicts the report of the trained regression model.

It could be observed that:

Figure 1.7, Report of Regression Model

OLS Regression Results

Dep. Variable:	log_loan_amount	R-squared (uncentered):	0.991
Model:	OLS	Adj. R-squared (uncentered):	0.991
Method:	Least Squares	F-statistic:	5.023e+04
Date:	Tue, 05 Oct 2021	Prob (F-statistic):	0.00
Time:	15:07:26	Log-Likelihood:	-287.81
No. Observations:	450	AIC:	577.6
Df Residuals:	449	BIC:	581.7
Df Model:	1		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
log_applicant_income	0.5791	0.003	224.125	0.000	0.574	0.584

Omnibus:	82.096	Durbin-Watson:	2.037
Prob(Omnibus):	0.000	Jarque-Bera (JB):	273.754
Skew:	-0.811	Prob(JB):	3.59e-60
Kurtosis:	6.460	Cond. No.	1.00

- R-Squared is 0.991 which means "log_applicant_income" explain 99% variance of "log_loan_amount".
- F-Statistic is 5.023*e+04 (ideally for a good model, it should be as large as possible) and P-Value is 0.0000 (ideally for a good model, it should be less than 0.05) which means the model is reliable and it would make significant predictions.
- P-Value with respect to "log_applicant_income" is also less than 0.05, hence the target feature is significantly dependent on "log_applicant_income".
- Values of Durbin-Watson test is 2.037 (ideally it should be between 1.6 to 2.3), thus it means there exists no correlation between the residuals.
- Hence, over all the model seems to be working really good and we can rely on the predictions of this model.

Now, there comes a very important task of verifying the assumptions of Regression model. If the assumptions are verified, then the model could be relied upon for making predictions of the missing values of the target feature, "log_loan_amount".

Figure 1.8, Verifying Assumptions of Linear Regression Model

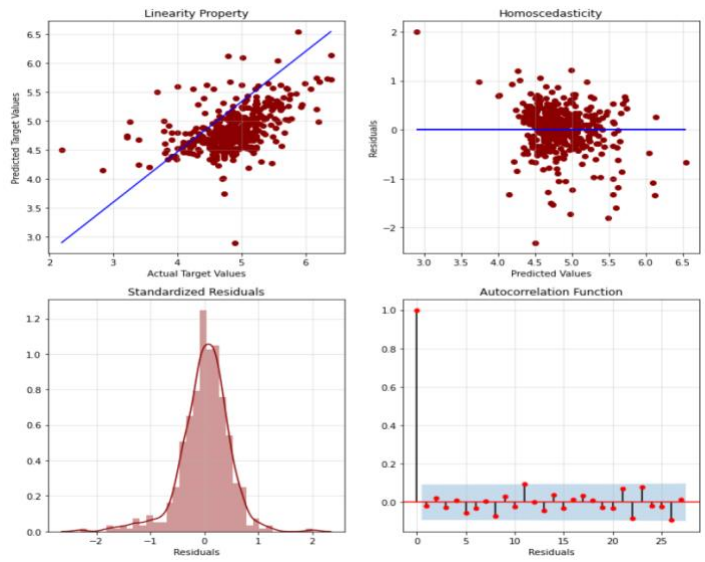


Figure 1.8 explains the assumptions of the trained regression model. The assumptions are as follows:

- **Linearity** – This property defines that the Actual Target Values and the Predicted Target Values when plotted on a

Scatterplot, should have **a linear relationship** between each other. From the very first figure (on top-left) of **Figure 1.8**, it is clearly visible that a linear relation exists between actual target values and predicted values. Hence, this assumption is verified.

- **Homoscedasticity of Residuals** – This property defines that the residuals should be constant for all the range of predicted values i.e. constant variance should exist. This simply means that for all the range of input values sent to the model to make predictions for, the residuals for all the range should be constant. **It symbolizes that the model should behave constant for all the range of input values**. From 2nd figure of **Figure (top-right) 1.8**, it can be observed that the residuals remain somewhat constant for all the values of input except 1 or 2 data points. Thus, this assumption is verified.

- **Residuals must be normally distributed** – This property defines that the probability density function of residuals should be Normal or Gaussian Distribution. From 3rd figure (bottom-left) of **Figure 1.8**, it can be observed that the residuals are normally distributed. Hence, this assumption is verified.

- **No Autocorrelation must exist among the residuals** – This property defines that no linear correlation should exist between the residuals i.e. if we consider a particular residual value, and when its correlation is computed with its next residual value or with its previous residual value, the correlation must be less than 0.05 which means that no correlation should exists. In 4th figure (bottom-right) **of Figure 1.8,** correlation among residuals is plotted along y-axis and residuals are plotted along x-axis. It can be observed for all the residuals, the correlation lies less than 0.05 (in the blue region), hence it can be stated that **no autocorrelation** exists between the residuals. Further, we also have a Durbin Watson test which states whether correlation exists among then residuals or not. If no autocorrelation exists, then the value of Durbin-Watson Test should be between 1.6 to 2.3 and in our regression model, Durbin-Watson value is 2.037 i.e. no autocorrelation exists among residuals. Thus, this assumption is also verified.

Thus, all of the assumptions are verified and we can now rely upon the predictions made by the model.

Figure 1.9 shows the predicted values of "LoanAmount" and along with the Training and Validation Error (Root Mean Square Error i.e. RMSE). The training and validation error is computed to observe if the model Is not overfitting the training data i.e. the model should behave similarly on the validation data as it behaved on training data. Hence, the evaluation metric, RMSE should approximately be same for both training and validation data. From **Figure**

Figure 1.9, Predicted Values of "LoanAmount" along with Training and Validation RMSE

```
Training RMSE --> 0.45870283669656503
Validation RMSE --> 0.4008141601403398

Data after dealing with Missing Values:
 0      151.929559
35       87.930511
63      137.852481
81       90.587492
103     133.061421
113     174.794700
127     119.519178
202     121.778110
284     315.588166
305      81.608666
322     114.721002
338      77.516463
387     103.407865
437      86.534643
479     102.148841
524     133.524654
550     163.410285
551      92.694497
605      90.696968
102     248.194910
435     207.831995
95      165.526199
```

1.9, it can be observed that the Training RMSE and Validation RMSE is approximately the same.

Missing Values of Categorical Features:

Till now we have seen techniques to deal with the missing values of numerical feature. In this section, we would see how we can deal with the missing values of Categorical Features. Here again, we might choose the appropriate technique based upon the count of data points present for each category of the categorical feature.

After plotting the count plot of a categorical feature whose missing values are to be dealt with:

- We inspect the proportion of each category of the categorical feature.
- If the proportion of one of the categories is too large as compared to the proportion of rest of the categories, then we can simply replace all the missing values with this category which seems to have the highest proportion.
- If the proportion of all the categories of the categorical feature remains the same, we must train a Machine Learning Classification Model such as Logistic Regression, Support Vector Classifier, etc. with the categorical feature whose missing values are to be dealt with as the Target feature. Further, we can make predictions for which the Categorical feature has missing values.

Figure 1.10 Count Plot of Categorical Feature, "Credit History"

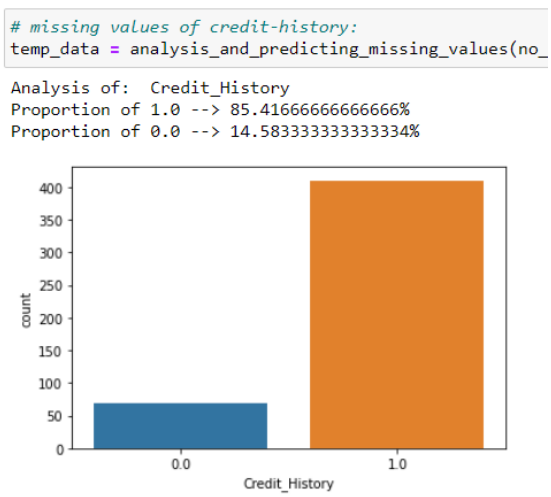


Figure 1.11 Count Plot of Categorical Feature, "Married"

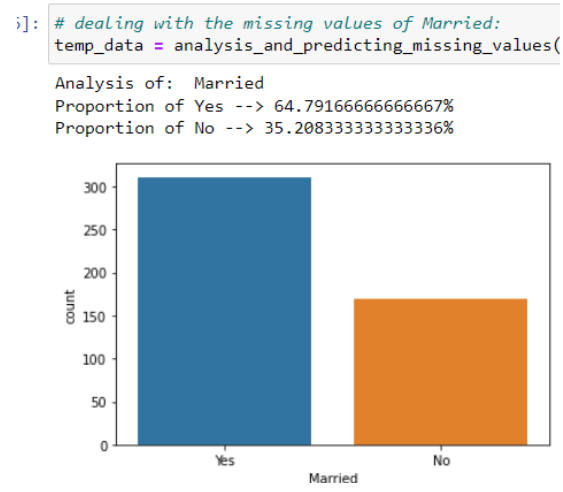


Figure 1.12 displays the count plot of the Categorical Feature, "Credit History", where "Credit History" = 1 means the customer has had good credibility with the bank where as "Credit Score" = 0 means the customer has bad credibility with the bank. Further, the proportion of 0 and 1 can easily be read from the figure. Since, 86% of the customers have good credibility, hence we can replace all of the missing values of "Credit History" with 1.

Figure 1.13 displays the count plot of the Categorical Feature, "Married". It can be observed that 65% of the customers are married where as 35% of them are not married. Thus, in such a case, we might resolve to train a classifier with "Married" as the target feature and the data points for which "Married" is not present should constitute the test data and using the trained model, we can make predictions on test data.

Chapter-2: Data Preprocessing yet to be done:

The data which we collect from multiple sources is raw in nature, hence in order to use that data for modelling purposes or for extracting important business outcomes, we must refine it and then use it for the above purposes. After having learnt how to deal with the missing values, we must proceed with the Data Preprocessing part where we perform various steps to clean the data in order to increase the quality of data. I have mentioned the topics below which we would be covering in this chapter:

- Intuitive understanding of Distribution of Numerical Features.
- When to apply mathematical transformations to transform the numerical features.
- Why we want our numerical features to have Gaussian Distribution.
- Univariate Outliers Investigation Techniques.
- Feature Engineering.

In the previous chapter, we have already looked at how numerical and categorical features could be differentiated using simple logic of Python as could be looked at in Figure 1.1. After obtaining the numerical and categorical features, we then should start exploring the numerical features first and after performing all the steps required to preprocess the Numerical Features, then we should inspect the Categorical Features.

Intuitive Understanding of Numerical Features:

In order to intuitively understand the Numerical Features, first of all let us consider a numerical feature “CreditScore” i.e. credit score of the customer regarding to that particular bank. We use histograms to inspect the distribution of data over different values. Boxplots are used to determine outliers in data as well as for understanding the distribution of data i.e. whether the data is skewed or not, or is it normally distributed? The asterisk in Figure-1.0 of boxplot towards lower whisker represents outliers. Ideally, we want a numerical feature to have Gaussian or Normal Distribution. If data is not normally distributed, we try to apply various transformation techniques such as logarithmic, square root transformation, etc. to the data so that it can be distributed approximately to Normal Distribution.

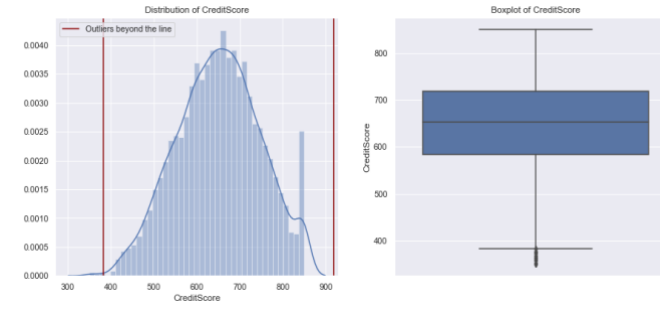


Figure 2.1 Distribution and Boxplot of Numerical Feature ‘CreditScore’

Figure-2.1 displays the Distribution of ‘Credit Score’, a numerical feature in our data. Range of ‘CreditScore’ is 350 units to 850 units and from the distribution, most of the customers seem to have a Credit Score between 600 units to 700 units, which implies that the percentage of customers having ‘CreditScore’ towards the tail of Gaussian i.e. for values less than 350 (left tail) and larger than 850 (right tail) are quite less. Mean of ‘CreditScore’ = 650.55 units and standard deviation = 96.562 units.

The ‘dark red’ line of Figure-1.0 (in histogram) is used to denote the boundary i.e. the upper threshold and lower threshold within which the data should be distributed for a numerical feature in

order to avoid the outliers. Hence, the data lying beyond the 'red line' is considered as the outliers (based upon IQR). However, based upon the values of upper threshold, lower threshold and range for a particular numerical feature, we can further inspect in which region the outliers might occur. For example, in numerical feature, 'CreditScore', the upper threshold is computed to be 919 units and lower threshold is computed to be 383 units [2]. Since the maximum value of 'CreditScore' is 850, thus there are no outliers in the upper end, which can also be observed from the Figure-2.1 where there appears no asterisk towards the upper whisker of the boxplot whereas minimum value of 'CreditScore' is 350 and thus the values lying between 350 to 383 are considered as outliers and if the percentage of outliers lying within 350 to 383 is less than 5% then we can use **'winsorizing' for handling outliers [3],** where the outliers are replaced by or equated to value of the upper or lower threshold.

Hence in the case of 'CreditScore', the percentage of Outliers is computed to be 0.15% on the lower end, which is less than 5% and we can equate the values lying between 350 to 383 to the lower threshold i.e. 383. Further, boxplots are used to derive the percentage of people lying within a particular range of values. For example, from Figure 1, we can infer that 25% of all the customers have 'CreditScores' less than 600, 75% of all the customers have 'CreditScore' less than around 710 and so on. This could help us to understand the data in a better fashion and take certain decisions based on it.

Need to Transform numerical features to Gaussian Distribution:

As discussed in the previous chapter, if the distribution of a particular numerical feature does not seem to be Gaussian in nature, then we might resort to apply some mathematical transformation which brings the distribution approximately to Normal Distribution. The process is more like a hit-and-trial method where you need to apply various transformation on non-gaussian data and see which of the transformation gets you the Gaussian Distribution. The transformations which could be applied to make the data Gaussian are:

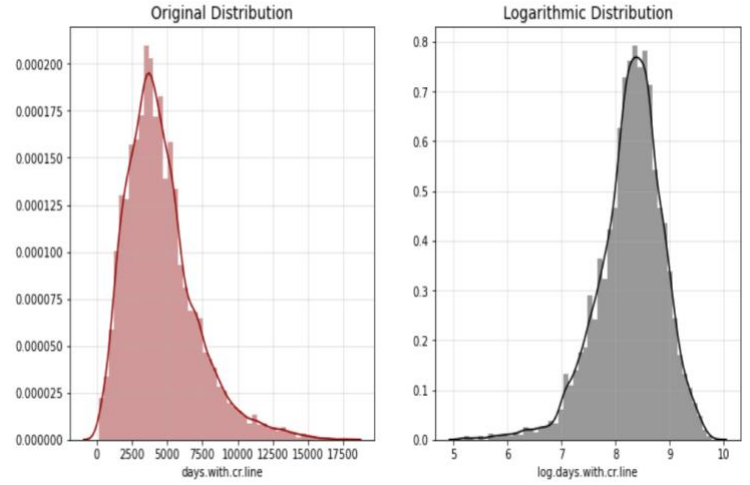


Figure 2.2 (a)Original Distribution and (b)Logarithmic Distribution of same feature

- Logarithmic Transformation if the original distribution of the numerical feature is highly skewed or more like Exponential Distribution.
- Square Root Transformation where we compute the square root of the original numerical feature's values.
- Even after applying these transformation techniques, then also the distribution does not become Gaussian so now, we would use Non-Parametric Statistical Tests for deriving inferences from the data.

- Remember, Parametric Statistical Tests could only be applied to data which is Normally Distributed.

Figure-2.2 displays the original distribution of a numerical features, which seems to be heavily skewed and hence, in such a case, we might apply Logarithmic Transformation. After applying log transformation, the distribution seems like as could be observed from Figure-1.2. Now, the data seems to have Gaussian or Normal Distribution.

Why do we want our Numerical Features to have Gaussian Distribution?

The reason why we want the Numerical features to have Gaussian or Normal Distribution because:

- All the Parametric Statistical Tests apply only to Normally Distributed data.
- These Parametric Statistical Tests such as ANOVA, T-test, etc. help us gain a lot of intuition from the data and we can make some excellent business strategies based upon the outcomes of these tests. The assumptions in all the Parametric Statistical Test has one common feature i.e. the data should be normally distributed.
- These parametric tests are also used for selecting the most important features for our target feature (Feature Selection).
- Further, a feature having Normal Distribution makes the process of finding univariate outliers easy and hence, it is also quite easy to treat them.

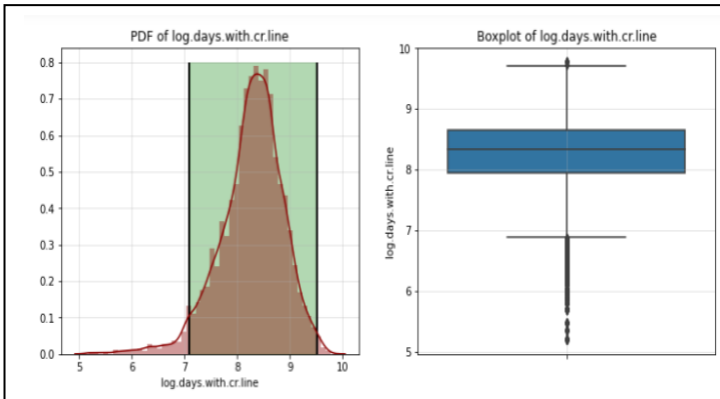


Figure 2.3 Outliers computed using IQR

Outlier Investigation:

Investigating Outliers is one of the most important aspects since it allows to remove the data points which seem to be erroneous and hence, might result in bad predictions if the model is trained on the raw data i.e. without dealing with the outliers. Thus, there are multiple techniques used for detecting outliers which are described as follows:

1) Inter-Quartile Range (IQR):

Inter Quartile Range refers to the difference between the first quartile i.e. 25^{th} Percentile and third quartile i.e. 75^{th} Percentile for a particular numerical feature. Here, is a screenshot of code demonstrating the computation of inter quartile range.

Once IQR is computed, then we set the upper threshold to be (75^{th} Percentile + 1.5*IQR) and the lower threshold to be (25^{th} Percentile - 1.5*IQR). Thus, in order to be an inlier, our data must lie between these two thresholds and if the data point lies between upper and lower threshold, then it is an inlier else it is an outlier. In, this way, outliers are detected using IQR. **In the following figure**, the green region corresponds to inliers and the white region corresponds to the outliers.

2) Z-Score:

Previously, we have discussed many advantages of a numerical feature having Normal Distribution and among those, one of them is inspecting outliers easily. Z-Score corresponds to the standard deviation at which the data is located from the mean. Generally, we use Standardized Normal Distribution and for each data point, z-score can be computed as "**place the formula of z-score i.e. (x-u/std_dev**)".

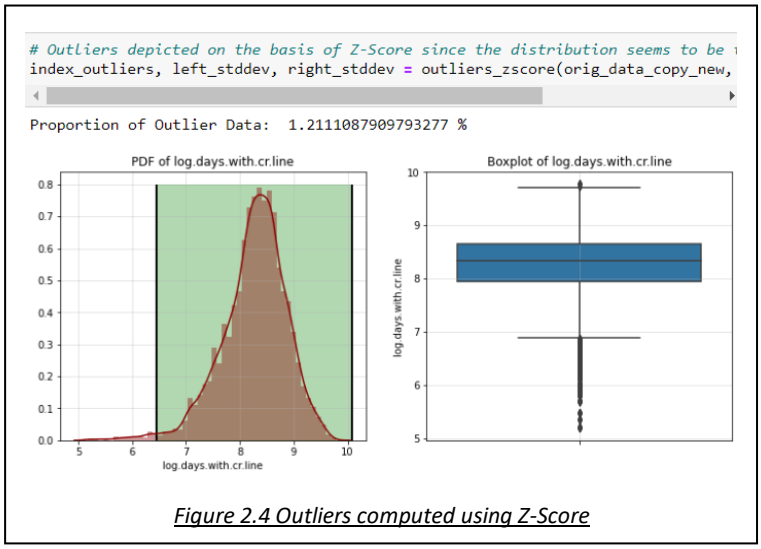
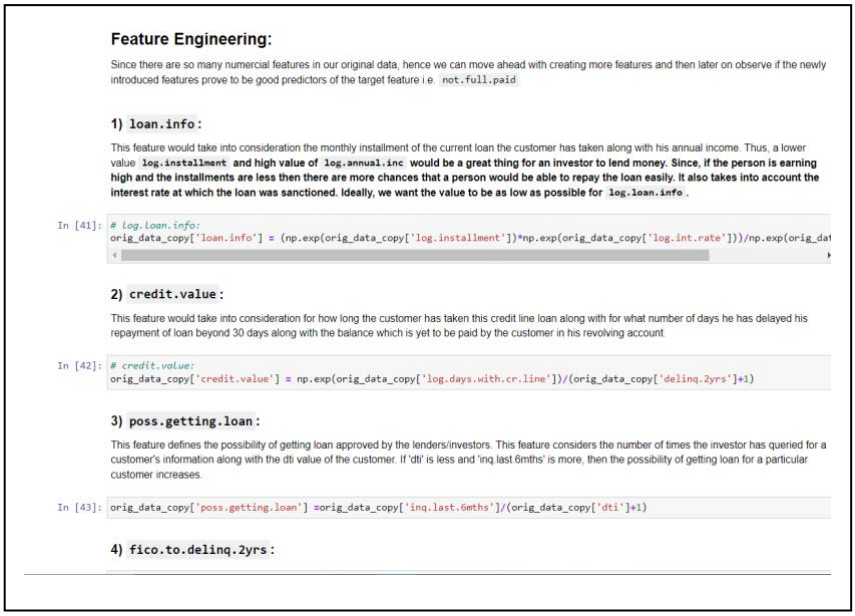


Figure 2.4 Outliers computed using Z-Score

It is the property of Normal Distribution that 68% of the data lies within 1st Standard Deviation, 95% of the data lies within 2nd Standard Deviation and 99.9% of the data lies between 3rd Standard Deviation of the Distribution. As we know, z-score is nothing but the standard deviation of a particular point from its mean. Hence, an outlier is a point which lies beyond 3rd standard deviation i.e. a data point having z-score equal to 3 or more than 3.

There is a **figure attached which corresponds to the outliers based** upon the z-score i.e. the green region corresponds to the 99% of the data or data lying within 3rd standard deviation. This green region corresponds to inliers and the white region beyond the green region corresponds to the outliers or the data points which have their standard deviation or z-score greater than 3.

Feature Engineering:



For a particular problem statement, we have a large number of independent variables or features. We can easily understand the relationship between the independent features and target variable. However, at times we might create more independent features using the existing independent features which might seem to have a better relationship with the target feature (increased R-Square). Thus, Feature Engineering corresponds to the creation of new features using the existing features which might result in creating feature that might have better predictability power for the target feature.

In Artificial Neural Network, the network itself tries all the combination with the input features and updates weights for each of the existing and engineered features in each iteration of Gradient Descent and once the cost function is minimized, then we can set the importance of each

engineered and existing feature based upon their final weights. Make sure that all of the features are scaled to the same range. Thus, ANN automatically does feature engineering and creates more features using existing ones in the form of a Network whereas with traditional Machine learning approaches, we need to engineer features based upon our understanding and insights.

In one of our problem statements i.e. “Loan Repayment (whether a person would be able to repay the loan or not.)”, I have excessively used my instincts and engineered a lot of new features using the existing features. **Let us have a look at a few of them in the figure**.

Chapter 3: Statistics comes to the rescue:

In this chapter, we would learn about some statistical techniques that help us find the most important features which might add some predictable power to our trained machine learning models, derive some really important insights from these statistical tests, using statistics to get rid of multicollinearity and remove multivariate outliers.

Feature Selection corresponds to the selection of those important features which seem to explain the variance of the target feature and in this section, we would refer to some of the Feature Selection techniques for both numerical and categorical features.

Feature Selection for Numerical Feature:

In all the four problem statements or features of our application, the target feature is categorical in nature hence, the predictions done by the application in real-time would be categorical only. Hence ANOVA (advanced version of T-Test) and Kruskal-Wallis Test are used to compute the numerical features which seem to explain the variance of the categorical target feature.

1) One-Way ANOVA: (Parametric Test)

One-Way ANOVA is used to test whether the categories of a categorical feature with respect to a particular numerical feature have the same mean or not. If they have same mean, then it means they are not explaining any variance of that particular numerical feature, hence we can conclude to drop that numerical feature since it is not adding or explaining any of the variance of the categorical feature. Remember, T-test is used for categorical features with number of categories =2 and One-Way ANOVA is used for categorical features with number of categories greater than 2.

ANOVA Results Table:

	CreditScore	Age	Balance	EstimatedSalary
F-Statistic	0.403801	13.667180	958.425446	0.582761
P-Value	0.667788	0.000001	0.000000	0.558374

Figure-3.1 ANOVA Result of 'Geography', a categorical feature.

Let us understand how One-Way ANOVA could be used for feature selection and interpret the results from this test. In one of our problem statements, a categorical feature is "Geography" and the numerical features are "CreditScore", "Age", "Balance" and "EstimatedSalary". Thus, our aim is to compare the mean of the categories of "Geography" with respect to each of the numerical features so as to get an idea of those numerical features which explain the variance of the categorical feature, "Geography".

Figure-3.1 shows the ANOVA Result of the categorical feature 'Geography' with respect to each of the numerical features. P-values and F-values are given for each of the numerical features stating whether the customers belonging to each of the categories i.e. France, Germany and Spain have the same mean with respect to each of the numerical feature or not.

Statistically, If the p-value is less than 0.05 for a particular numerical feature, then it means the null has been rejected and the population mean is different for at least one of the categories whereas if p-value is greater than 0.05, then it means the null is accepted and all the categories have same mean**.** In this case, for a numerical feature to be considered as a good predictor of the target categorical feature**,** it should have p-value less than 0.05 at least for a single Numerical Feature.

From **Figure-3.1**, we can say that numerical features '**Age' and "Balance" have significantly explained the variance of categorical feature, "Geography". Since their p-values are less than 0.05.**

2) Kruskal-Wallis Test: (Non-Parametric Test)

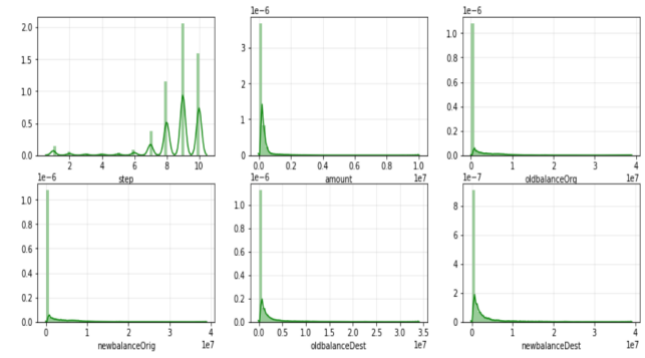


Figure 3.2 Distribution of Numerical Features

Many a times, the numerical feature which explains the variance of the categorical feature does not have a Gaussian Distribution, then at those times we use Kruskal Wallis Test which tests whether the categories of categorical feature have their data belonging to the same population or belonging to different populations. Thus, helps in explaining variance of the categorical feature with respect to the numerical feature. Hence, the **null hypothesis states that data of categories of the categorical feature belongs to the same population whereas the alternate hypothesis states data of categories of the categorical feature comes from different populations**. Ideally, we want to reject the null hypothesis and accept the alternate hypothesis. Hence, the p-value must be smaller than 0.05 in order to reject the null hypothesis.

Let us illustrate the effectiveness of this test with our experimentation in one of the problem statements, “Fraudulent Transaction” where the target categorical feature is “isFraud” and a lot of numerical features are there so we simply wanted to test if the categories of “isFraud” with respect to each numerical feature seems to come from the same population or from different populations. If they come from different population, then the null hypothesis gets rejected and simultaneously the alternate hypothesis gets accepted which means that particular numerical feature explains the variance of “isFraud”.

```
# kruskal wallis test to derive whether the
# following num feats for fraud and
# normal transactions come
# from the same distribution or not.
kruskal_table = kruskal_wallis_num_feats(orig_data_copy, "isFraud")
imp_feats_kruskal = list(kruskal_table["Numerical Feature"])
kruskal_table
```

	Numerical Feature	P-Value
0	step	5.703136e-32
1	newbalanceOrig	1.485443e-20
2	oldbalanceDest	2.801679e-09
3	newbalanceDest	7.516507e-03
4	oldbalanceOrig	1.266187e-04
5	diff_orig	3.018956e-41

Figure 3.3 Result of Kruskal-Wallis Test on Fraudulent Transaction

Let us also understand why we have used Kruskal-Wallis Test and not One-Way ANOVA. **Figure 3.2** demonstrates the distribution of the numerical features and none of them has a gaussian distribution. Had there been any numerical feature with gaussian distribution, we would have used One-Way ANOVA (since parametric tests require this assumption of data being normally distributed) instead of Kruskal-Wallis Test.
Figure 3.3 shows a table which has p-values for each of the numerical feature and since all of them have p-values less than 0.05, thus all of them are explaining variance of the target feature, “isFraud”, hence all of them must be considered for predicting the target feature.

Feature Selection for Categorical Features:

We also have a lot of independent categorical features and the selection of the best categorical features could not be done in the same way as it was done in selecting the best numerical features. Thus, in order to choose the best categorical features for our target feature, there is a test in

Statistics called as Chi-square Test of Independence which tests whether the two categorical features (whose relationship needs to be tested) are dependent or not.

Chi-Square Test of Independence:

Chi square test of independence is used to test if the two categorical features are dependent or not. Thus, the null hypothesis states there is no dependence between the two categorical features and the alternate hypothesis states there exists dependence between the two categorical features. Ideally, we want to reject the null hypothesis and accept the alternate hypothesis. Hence, the p-value must be smaller than 0.05 in order to reject the null hypothesis.

```
# applying chi2 test of independence on data:
chi2_table = chi2_test_of_indep(orig_data_copy, cat_feats_orig)
chi2_table
```

```
Chi-square Test of Independence is computed for -->  fully.paid
```

	Significant Features	P-Values
0	credit.policy	8.875731e-54
1	purpose	1.067245e-18

Figure 3.4 Results of Chi-Square Test of Independence on "Loan Repayment"

In our problem statements, the target feature is categorical in nature hence, we would be using this test to check if there is any other categorical input feature which depends upon the target feature and if the p-value of a categorical input feature with respect to its target feature comes out to be less than 0.05, then it means that input categorical feature is dependent on the target feature, hence it is a good predictor of the target feature.

Let us illustrate the effectiveness of this test with our experimentation in one of the problem statements, "Loan Repayment". Here, there are only two categorical features i.e. "credit.policy" and "purpose" and the target categorical feature is "fully.paid". Thus, we wanted to test the dependence of these categorical features on the target feature. From **Figure 3.4,** we can observe that p-values with respect to both the features are less than 0.05, hence both of them are dependent on the target feature and hence, both are good predictors.

Investigating and Getting Rid of Multicollinearity in Data:

Multicollinearity exists when one independent numerical feature is highly dependent on any other independent numerical feature. This means that either of these two numerical features are redundant i.e. they explain the same amount of variance for the target feature and hence are not increasing the explain ability power of the target feature. Hence, we investigate the presence of multicollinearity within our final selected independent features. Thus, multicollinearity should always be tested before the data is used for model training. There are 2 ways multicollinearity could be inspected:

Pearson's Correlation:

```
# highly correlated independent features with thresholded correlation>0.6:
correlation_table = orig_data_copy[num_feats_anova].corr()
highly_correlated_feats = []
all_feats = list(correlation_table.columns)
for idx_row in range(len(correlation_table)):
    for idx_column in range(len(correlation_table)):
        if (idx_row != idx_column):
            corr_value = correlation_table.iloc[idx_row, idx_column]
            if (corr_value > 0.6):
                info_to_add = (all_feats[idx_row], all_feats[idx_column], corr_value)
                highly_correlated_feats.append(info_to_add)
            else:
                continue
        else:
            continue

# Creating a table for features whose correlation is relly high:
corr_feats_table = pd.DataFrame(highly_correlated_feats, columns=['Feature-1', 'Feature-2', 'Correlation'])
corr_feats_table
```

	Feature-1	Feature-2	Correlation
0	log.annual.inc	fico.to.delinq.2yrs	0.836255
1	credit.value	log.days.with.cr.line	0.841143
2	fico.to.delinq.2yrs	log.annual.inc	0.836255
3	log.days.with.cr.line	credit.value	0.841143

Figure 3.5 Code of computing Correlation in Python along with results

1) Correlation Matrix:

We must compute Pearson's Correlation (which describes the linear relationship between any 2 numerical features) between all the possible pairs of the final selected numerical independent features in order to test if any of the independent numerical feature is highly dependent on any other independent numerical feature so that one of the could be removed in order **to remove redundancy in data**.

Figure 3.5 lists features which seem to have correlation greater than 80% with each other. Thus, higher correlation means both of the features are highly dependent on each other and would explain the same variance of the target feature. Thus, one of them should be removed.

Figure 3.5 shows the code of computing Pearson Correlation for each pair of numerical features in Python and the features which had high correlation i.e. which are linearly dependent on each other, are listed in the table. Thus, "fico.delinq.2.yrs" and "credit.value" are dropped since "log.annual.inc" and "log.days.with.cr.line" explains the same information as they do. In this way, we remove redundant features and get rid of multicollinearity.

2) Variance Inflation Factor (VIF):

$$VIF_i = \frac{1}{1 - R_i^2}$$

This technique trains a Regression Model with one of the finally selected numerical features as the target feature and the other numerical feature as the independent input feature and hence computing the R-Squared of this trained regression model. In this way, the regression model is trained for all possible pairs of numerical features and R-Squared is computed for all the possible pairs. If the R-Squared is greater than 70% for a particular pair of numerical features, then it means that those two numerical features are highly dependent on each other and one of them has to be removed in order to get rid of multicollinearity. The VIF should be less than 5, if it is greater than 5, then multicollinearity does exist In the model and we must further investigate the reason and try to reduce it. The formula of VIF is mentioned alongside.

Inspect Multivariate Outliers:

There are multiple techniques that are used to inspect the multivariate outliers. Earlier in the previous chapter, we discussed about inspecting the Univariate outliers using IQR and z-score. However, once the univariate outliers are treated and the data is cleaned then we finally investigate for multivariate outliers using 3 different algorithms in this case and the data points flagged as outlier by any of the techniques are removed from the actual data. Thus, we simply remove them from our dataset since they generally seem to occur in small number hence, they won't be having much impact on actual data after they get removed.

Now, before understanding the techniques we have used for detecting multivariate outliers, first we should be able to understand how the figure is plotted. Since, the dimensionality of data was too large to be plotted on a 2D plane thus we have used Principal Component Analysis (PCA) to reduce the dimensionality of data up to only 2 features so that the outliers flagged by any outlier detection technique could be easily spotted. Hence, a scatter plot is plotted between the 2 principal component vectors and the outliers flagged by a particular anomaly detection technique are marked in purple color whereas the inliers are represented as red color. The 3 algorithms or techniques used for raising multivariate outliers are described as follows:

- **Mahalanobis Distance**: This technique standardizes the data (unlike Euclidean distance which gets affected if scaling Is not done properly beforehand) and also computes the covariance matrix where the correlation between every two independent features is computed and while computing the mahalanobis distance, it takes into consideration the bivariate relationship between any two features i.e. if 2 features have high correlation, then mahalanabis distance between those 2 data points in space will be less and hence less likely to be flagged as an outlier. **The higher the mahalanobis distance, the more the chances of that record being flagged as an outlier**. Further, we create a chi-square distribution of mahalanobis distances computed for each record and we basically focus on the tail region for a particular mahalanobis distance. Simply, we first construct a chi-square distribution of mahalanobis distance and from this chi-square distribution with degrees of freedom = number of independent features, we compute the probability of a particular mahalanobis distance. However, we compute the Cumulative Distribution Value for each of the record and subtract it by 1 i.e. we are basically computing the area of region towards the tail for each mahalanobis distance using the Chi-Square Distribution. **If the area of region towards the tail for a particular mahalanobis distance is greater than 0.05, it is considered as a normal data point** where as If the area of **region towards the tails for a particular mahalanobis distance is less than 0.05, it means that point is situated far away from the concentrated region** and closer towards the tail and hence, considered as an outlier. 105 data points were flagged as outliers using this technique of multivariate outliers. Figure 3.6 demonstrates the outliers flagged by Mahalanobis Distance.

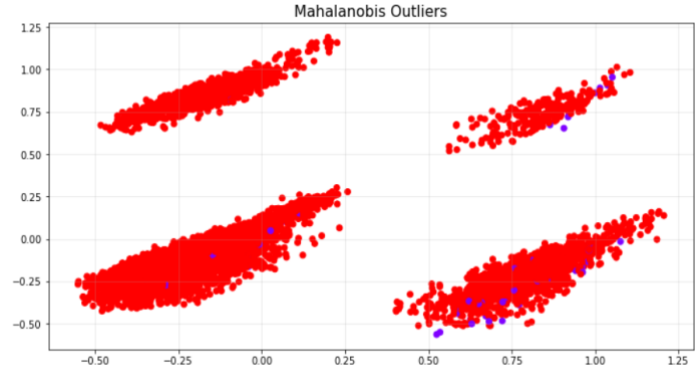


Figure 3.6 Multivariate Outliers flagged by Mahalanobis Distance

- **Local Outlier Factor**: With the help of univariate outlier detection techniques such as IQR, boxplots, we get rid of the Global Outliers but if we want to get rid of the Local Outliers, then we can use this technique. LOF (Local Outlier Factor) is an **unsupervised anomaly detection** task which flags a data point as an outlier by computing **its density with respect to its neighbors** and assigns each data point an LOF-Score. **A data point If assigned higher LOF score, means that the density of that particular data point with respect to its closest neighbors is quite less, hence would**

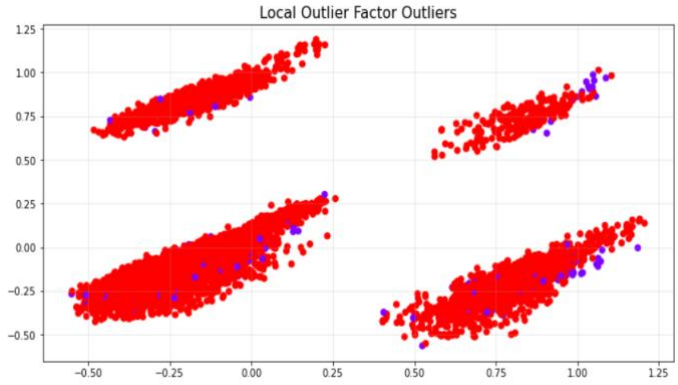


Figure 3.7 Multivariate Outliers flagged by Local Outlier Factor

be flagged as an outlier and vice versa. There are in total 67 data points flagged as Outliers which can be observed from the figure where outliers are represented using blue dots. Figure 3.7 demonstrates the outliers flagged by Local Outlier Factor.

- **Isolation Forest**: It is again an unsupervised anomaly detection algorithm which consists of multiple trees i.e. Forest (like Random Forest) where the aim of the tree is to isolate each and every data point. Here, the features are chosen randomly (unlike decision trees) which are used to isolate each data point based upon some density metrics. **The outliers tend to get separated within just a few iterations where as it takes a lot of splits to separate the inliers**. The outliers are represented in Blue Dots in the figure where green dots represent the inliers. Figure 3.8 demonstrates the outliers flagged by Isolation Forest.

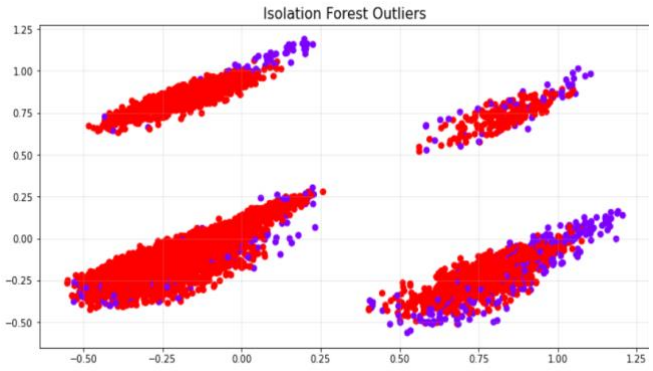


Figure 3.8 Multivariate Outliers Flagged by Isolation Forest

Chapter 4: Generating More Data Synthetically and dividing data into Training and Validation datasets:

Till now, we have been cleaning and preparing the data for model training. We have done a lot of preprocessing in the data and also applied multiple statistical techniques to either deal with outliers or select the most important features which explain the variance of the target feature, etc. Hence, the data seems to be ready for model training.

Is the data actually ready for model training? I don't think so. Although we have successfully cleaned our data, we have not inspected the proportion of data for each category of the target categorical feature. It is one of the most important aspects in Data Science to analyze the proportion of data for each category for a categorical feature since all of our problem statements i.e. loan repayment, loan approval, fraudulent transaction and customer churn have its target feature of Nominal level of measurement i.e. categorical in nature.

In Figure-4.1, we can easily observe that 80% of the total data corresponds to those customers who seem to have successfully repaid the loan (target=1) and we only have 20% of the data for those customers who have not repaid the loan (target=0). Thus, even if we predict "1" randomly for a normal use case, then 80% of the times we would be correct and randomly predicting "1" all the time for any input values gives us an **accuracy of 80%**. Thus, in such a case we might think of increasing the data for the minority class i.e. the customers who have not paid the loan (target=0).

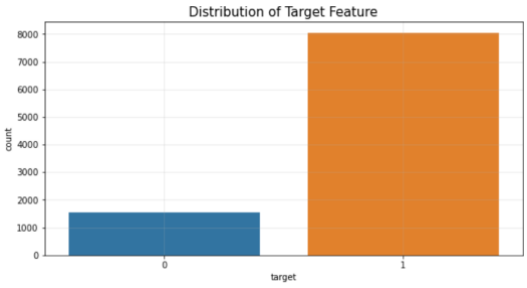


Figure 4.1 Distribution of Categories of Target Feature

Experimentation:

Now, before just going ahead with increasing the data for the minority class, we must experiment with training a Machine Learning Model on this data where asymmetric proportion exists for the categories of the Target Feature and then tune the hyperparameters of the very same model unless there seems to be no further improvement in the validation metrics. Now, note down the validation metrics of this trained model on asymmetric proportion of categories of the Target Feature. Then, we must perform oversampling i.e. increasing the data for the minority class of the target feature and then train another Machine Learning Model, tune the hyperparameters of this model where oversampling has been done and then note down its validation metrics. It would be really interesting to compare the validation metrics of the model trained without Oversampling and the model trained with Oversampling. Ideally, the validation metrics must be better for that model which has been trained with Oversampling and if this does not happen, then it means Oversampling is not able to improve the performance of the machine learning model and it was useless to perform oversampling.

Hence, now we have an understanding that we should not randomly go ahead and perform oversampling rather we must compare both the models i.e. the one trained on oversampled data and the one trained without oversampling. If the performance metrics of the model with Oversampling are better then only, we must use that model else we can proceed with the other model.

Experimenting Practically:

In one of our problem statements i.e. "Loan Repayment", I have performed Oversampling. We start by simply understanding the proportion of data for each category of the Target Categorical Feature as in the **Figure 4.1**. Once we realize that asymmetric distribution of data exists between the categories of the target feature, then we must plan to oversample i.e. increase the data for minority class up to such an extent so that both the categories have equal proportion of data. Now, we start with experimentation process. First of all, we would train the model without oversampling and then the model with oversampling would be trained.

Model Without Oversampling:

We simply take the cleaned data and do not perform oversampling on it. First of all, we divide the whole data on training and validation datasets where training data is used to train the model and validation data is used to verify the performance of the trained model. The machine learning model which we have chosen to train is Support Vector Machine Classifier (SVC). SVC has been trained on the non-oversampled data set and then further, we continue to tune the hyperparameters of this model until the performance metrics do not seem to improve. Thus, we have our best SVC model trained on non-oversampled data and now we note down it's performance metrics on validation data.

```
# evaluating model of SVC without oversampling:
evaluating_model(best_svc_clf, 'SVC')
```

```
Evaluation Table for SVC
           Training Metrics  Validation Metrics
Accuracy                1.0            0.840292
Precision               1.0            0.840292
Recall                  1.0            1.000000
F1-Score                1.0            0.913216
AUC Score               1.0            0.500000
```

Figure 4.2 Validation Metrics of Model without oversampling

Figure 4.2 corresponds to the performance metrics of the trained SVC model and the validation metric seem to be fine. However, it seems that the trained model is overfitting the training data since it has better performance metrics on training data as compared to validation data. Thus, important details regarding this model is as follows:

- Accuracy = 84%
- Precision = 84%
- AUC Score = 50%
- Model seems to overfit the training dataset.

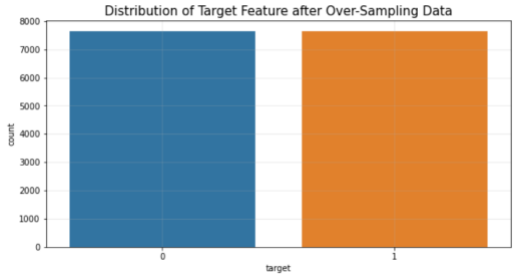


Figure 4.3 Distribution of categories of Target Feature after Oversampling

Model with Oversampling:

The very first step is to perform oversampling on the minority class. We use an algorithm SMOTE (Synthetic Minority Oversampling Technique) which is used to increase the data point of the minority class. SMOTE works in such a manner that it creates a line between two data points of minority class and randomly generates new data points lying within that line which was created earlier. In this SMOTE helps in generating synthetic data for the minority class.

After applying SMOTE on the data, we have equal proportion of data for both the categories of the Target Categorical Feature as can be viewed from **Figure-4.3**. Now equal proportion of data exists between both categories of the target categorical feature. Let us understand with an example. Suppose if we have in total 10,000 data points, out of which 8,000 belongs to class-1 and 2000

belongs to class-2. Hence, class-2 is our minority class and after applying SMOTE, synthetic data points are generated for Class-2 and the data points distribution would be as follows; in total there would now be 16,000 data points where both the classes have the same proportion of data i.e. 8000 each after performing oversampling.

Now suppose, if out of 16000, we want 90% of the data to training data and 10% to be our validation data.

Training data -> 90% of 16000 = 14400 (7200 of class-1 and 7200 of class-2)

Validation Data -> 10% of 16000 = 1600 (800 of class-1 and 800 of class-2)

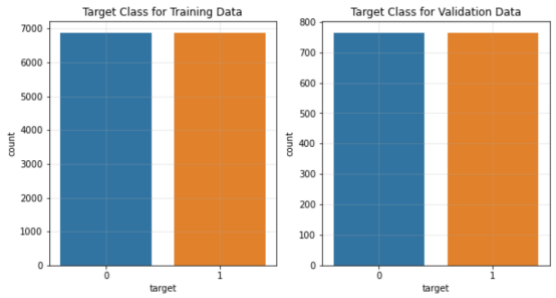


Figure 4.4 Distribution of categories of Target Feature after Oversampling for Training and Validation

Thus, 14400 should be our total training data points and it should not be like (9000 of class-1 + 5400 of class-2) or (8000 of class-1 + 6400 of class-2) rather it should have equal proportion from both the classes i.e. (7200 of class-1 + 7200 of class-2). Similarly, it is for Validation Data. Here is a **Figure-4.4** which represents the data for each category of the Target Feature in Validation and Training Datasets. It is clearly indicated that the both the categories have equal proportion of data.

Now, we train the SVC model on this oversampled training data and verify the model's performance using the validation data. We keep on performing the hyperparameter tuning until there seems to be no improvement in the performance metrics of validation data. **Figure-4.5** represents the performance metrics of both training and validation dataset. The validation metrics seems to be improvised by a large margin and thus is means that after performing oversampling, the model's performance improved a lot and the AUC Score of this model turned up to be 95% which is 45% more than the previous model (without oversampling) trained.

```
# evaluating Best Support Vector Classifier:
evaluating_model(svc_model, 'SVC', save=True, show_plots=True)
```

```
Evaluation Table for SVC
           Training Metrics  Validation Metrics
Accuracy                1.0            0.948298
Precision               1.0            0.925466
Recall                  1.0            0.975131
F1-Score                1.0            0.949649
AUC Score               1.0            0.948298
```

Figure 4.5 Validation Metrics of SVC trained On Data with Oversampling

An additional point to observe is that there does not seem to be any overfitting of the model on training data unlikely as that of the previous model which was overfitted on the training data (without oversampling).

Chapter-5: Machine Learning Algorithms Used:

Different Machine Learning Algorithms are meant for different problem statements. In our product, we have trained supervised machine learning classification algorithms for 3 of the problem statements i.e.:

- Loan Repayment
- Loan Approval
- Customer Retention

However, for the 4th problem statement i.e. "Fraudulent Transaction", we would be using several Anomaly Detection Algorithms to raise the anomalous transactions done by the user. Hence, we would understand the underlying mathematics and logic involved in developing a particular machine learning algorithms for both supervised and anomaly detection algorithms.

Supervised Machine Learning Algorithms:

When we know beforehand the labels of the data that is when we use Supervised Machine Learning Algorithms. Following supervised algorithms have been used for training models for the features or problem statements listed above.

1) Logistic Regression Algorithm:

Logistic Regression is majorly used for separating classes which are linearly separable. Here, prediction is made by the function mentioned in the **Figure 5.1**. We can observe the shape of the sigmoid function i.e. 1/(1+exp(-z)) and the probability for a particular data point is computed using this formula. Now, an algorithm called as "Gradient Descent" is used to update the values of "w" and "b". Thus, with each iteration of the Gradient Descent, the values of "w" and "b" must change in such a manner that the cost function is reduced at each iteration.

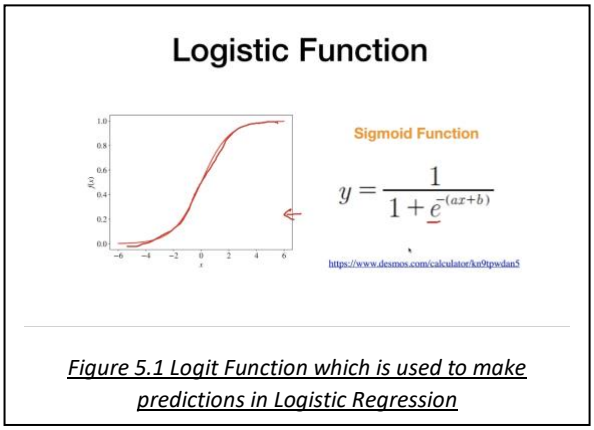


Figure 5.1 Logit Function which is used to make predictions in Logistic Regression

The cost-function for Logistic Regression is mentioned in **Figure 5.2** where meaning of each the alphabet is as follows:

- J(theta) -> symbol to represent cost function
- $y^{(i)}$ -> the actual label for the current data point
- $h_{theta}(x^{(i)})$ -> predicted value for a particular data point using the sigmoid function which in turn uses the values of "w" and "b"
- a summation sign is used which is used to sum the result for all of the data points in the training data.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\log(h_\theta(x^{(i)})) + (1-y^{(i)})\log(1-h_\theta(x^{(i)}))]$$

Figure 5.2 Cost Function of Logistic Regression

Thus, with the help of gradient descent, we are able to train the logistic regression in such a way that at each iteration, the values of "weight" and "biases" change in order to further minimize the cost

function. Gradient Descent keeps on continuing unless no further decrement in cost function is observed or in technical terms, we can say that the gradient of the loss function is zero i.e. it has reached the minimum point of cost-function.

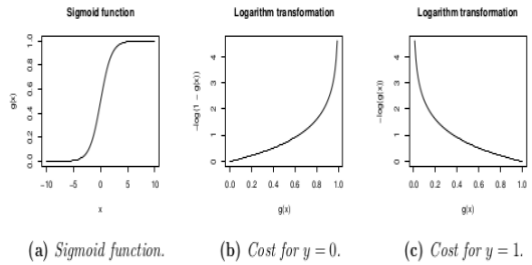


Figure 5.3 Logistic Regression's (a)Logit Function, (b)Cost Function for label=0 and Cost Function for label=1

Figure 5.3 demonstrates the cost function in a more intuitive manner. In the "B" part of the Figure, if we substitute $y^{(i)}=0$ in the cost-function, then the function that gets formed is represented in this figure. This means that we if make predictions for the same data point having actual label $y^{(i)}=0$, then the prediction should also be 0. Thus, this figure demonstrates that near probabilities around 0 (along x-axis), we have less error and we have larger error for predictions around 1.0 (along x-axis). Thus, for misclassification, the model gets penalized and large value of cost-function is computed. Similar is the case defined in (c) part in the same Figure where $y^{(i)}=1$ and less or almost null error is caused when the predicted value is 1 whereas more error is caused when misclassification is done i.e. the predicted probability is 0.

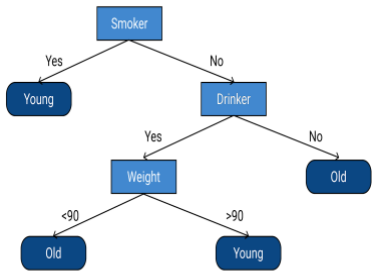


Figure 5.4 Example demonstrating Decision Tree

2) Decision Trees Algorithm:

It is most commonly used algorithm for classification problem statements. In this, we provide the dataset and a tree is created with some conditions at each level based upon each attribute's value and the aim is to create a node which contains all homogeneous data points i.e. the data points should correspond to the same class of the target feature. The very first node in the Decision Trees is called as the root node and the last or terminal nodes are called as "Leaf Nodes".

Figure 5.4 corresponds to the Decision Tree created on a dataset which had its attributes as ["Smoker", "Drinker", "Weight"] and the decision tree based upon the data fields of the attributes predicts whether the person is "Young" or "Old".

```
# psuedocode of Decision Tree:

if Smoker == "Yes":
    print("Young")
else:
    if Drinker == "No":
        print("Old")
    else:
        if Weight > 90:
            print("Old")
        else:
            print("Young")
```

Figure 5.5 Pseudocode of above Decision Tree in Python

The above decision could be interpreted in the following manner. Once, the data is sent to the model in real time i.e. we have the values of "Smoker", "Drinker" and "Weight" and the model checks if "Smoker" == "Yes", then the model predicts "Young" else it would check if "Drinker" == "No", then the prediction would be "Old" else it would check if "Weight" > 90, then the prediction is "Old" else "Young". Hence, in this way prediction is done. The dark blue nodes in the figure are called as "Leaf Nodes" or "Terminal Nodes" and the sky-blue nodes are called as "Decision Nodes" since one or the other

decision is made at that particular node. The pseudocode of the decision tree represented in Figure 5.4 is represented in **Figure 5.5.**

It is recommended to have smaller trees in depth since larger or deeper trees tend to overfit the training data and give very poor results for validation or test data. Although it seems quite easy to create a Decision Tree, it is not. Once the training data is sent to the model, it considers each feature to split the training data and further at each level, a subset of attributes or features are tested so as to which results in the least entropy after the data is splitted based on that attribute and that attribute is chosen which results in splitting the data in such a manner that the data points belong to the same class of the target feature.

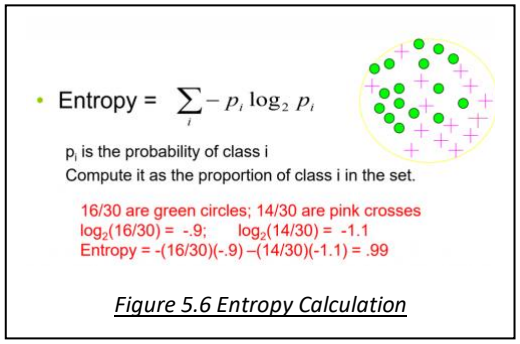


Figure 5.6 Entropy Calculation

Entropy corresponds to the randomness in data at a particular node. High entropy means equal proportion of data points from all the categories of the target feature are present and lowest entropy means all the data points belong to the same class of the target feature and the aim of creating Decision Tree is to group homogeneous data points (data from a particular class of the target feature) in the leaf node. Attach the formula of Entropy.

Information Gain (IG) is the metric which defines how well a given attribute separates the training examples based on the target classification. Thus, as previously mentioned, out of the subset of features to choose from, we compute information gain for each of the attribute and the attribute having **maximum** value of the Information gain is chosen as the attribute to split data at that level.

Thus, Information Gain along with Entropy make help us choosing the right attribute at the right level of the Decision Tree and it should also be noted that a feature if used before in the decision tree could not be used again in later stages of the tree.

If possible, **attach a Figure demonstrating the computation of Entropy and Information Gain.**

3) Random Forest Algorithm:

It is also called as **Bootstrapped Aggregation** Algorithm where the whole training data is sampled into multiple datasets and a decision tree is trained on each of the sampled data set. Thus, while sampling the datasets from the training data, it should be noted that both feature sampling and row sampling is done i.e. a particular record in the training data could be duplicated in the sampled data and the shape of each of the sampled data is the same as that of training data. Thus, it implies that duplication of a particular record is done.

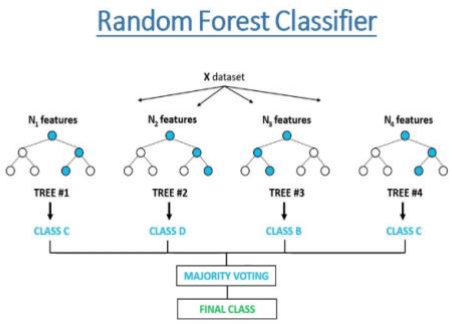


Figure 5.7 Random Forest Algorithm (Ensemble of Decision Trees)

Bootstrapping means if a particular data record has been sampled in data-1 (first sampled dataset), then that same record of data could be sampled in data-2 (second sampled dataset) and so on and so forth. Thus, the **data sampling with replacement is called as "Bootstrapping**" and the **data**

sampling without replacement is called as "Pasting". Thus, as soon as the training data is fed into model, multiple sampled bootstrapped datasets are created with the same dimension as that of the training data and each of the bootstrapped data gets trained by a Decision Tree where each of the Decision Trees are independent of each other.

As can be observed from the **Figure 5.7,** we can easily observe that a particular Decision Tree is trained on each bootstrapped dataset and the predictions of each of the Decision Trees are aggregated together and the class that's been predicted by most of the decision trees (Aggregation or Majority Voting) are predicted as the final class. This is why Random Forest is also called as "Bootstrapped Aggregation". Further, Decision Trees are trained in the similar manner as we discussed before.

Since, now the prediction by the Random Forest is the maximum function of prediction of multiple underlying Decision Trees, thus the Random Forest Model has the characteristic of having Low Bias and Low Variance. Low Variance means the model would not be overfitted on training data and low-bias means that the algorithm was able to understand the underlying patterns of the data and does not underfit the training data. Hence, In most of the cases, Random Forest provides better performance metrics as compared to Decision Trees.

4) Support Vector Machines (SVM):

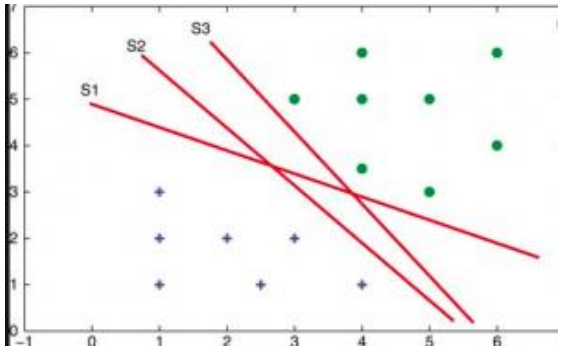


Figure 5.8 Multiple Possible Decision Boundaries

SVM is also called as the "**maximal margin classifier**". In order to understand why is it called so, we must consider some data points of 2 different classes as can be viewed from the **Figure 5.8**. There are 2 different classes where the data points belonging to the first class are represented using "Green Circles" and the data points belonging to the second class are represented using "Purple Stars". It can easily be understood that data of the two classes is linearly separable and hence, we can have multiple decision boundaries which separate the data of these 2 classes. In **Figure-jaks, "S1", "S2" and "S3"** are three different decision boundaries that could be used to classify data. However, is there any way to choose the best decision boundary? Yes, "**Maximum Margin Classification**" is used to choose the best decision boundary based upon the distance of the margin on both sides of the hyperplane. Hence, "**S2**" seems to have the maximum margin and thus it would give the best generalization results. It should be noted that if a classifier with a smaller margin is chosen, then it would overfit the training data and thus would not generalize well on the unseen data points and also increasing the complexity of the model. Thus, we must never choose such a decision boundary.

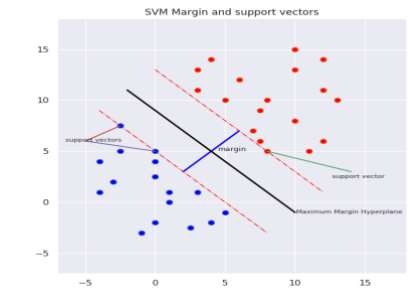


Figure 5.9 Margin and Hyperplanes in SVC

It could be observed from the **Figure 5.9,** there are 2 classes of data i.e. one in red and one in blue and there is a decision boundary drawn in black and the two red-dotted lines are the hyperplanes which determine the support vector and the margin of the classifier. Margin is the distance between the two hyperplanes. The larger the

margin, the better our model would generalize on unseen data. "**Support Vectors**" are those data points which lie on any of the hyperplanes and can be easily pointed out in the **Figure 5.9**.

a) Linear SVM:

In Linear SVM, our aim is to compute the values of "w" i.e. weights and "b" i.e. biases such that our cost function is minimized and some constraints are also required to be met. Since the objective function is quadratic and the constraints are linear, hence it is called as "Convex Optimization Objective" and we use "Lagrangian Multipliers" to compute the values for "weights" and "biases".

b) Linear SVM: Non-Separable Cases:

At times, we see cases where the classes cannot be linearly separated and there are very less misclassifications within the data as can be viewed from the **Figure 5.10**. Thus, most of the data is linearly separable and a very less amount of misclassifications are there. Thus, we introduce what is called as "**slack variable**" represented by " **£**" and slack variables help in computing the amount of misclassification done by each of the classes and for each misclassification, some penalty is added to the cost function, rest the computation of "weights" and "biases" remains the same.

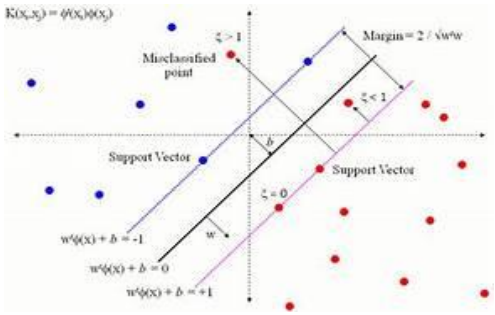


Figure 5.10 Understanding Slack Variables

c) Non-Linear SVM:

Non-Linear SVM is used when the data of different classes cannot be separated by a linear function as can be viewed from the **Figure 5.11**. Here, in Figure, the data points of first graph is linearly separable where as the data points of second graph is non-linearly separable. Hence, now in such as case, we use "Non-Linear SVM" where the data points are mapped to a higher space using a mapping function or transformation function in such a manner that in that higher space, the data points tend to become linearly separable and once they become linearly separable, then we can compute the values of "weights" and "biases" using Linear SVM.

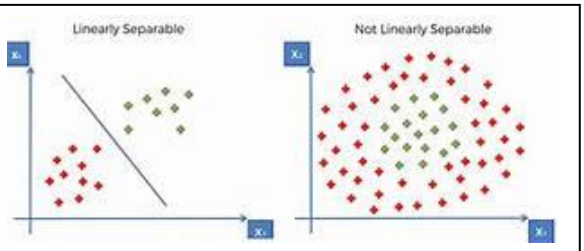


Figure 5.11 Linear and Non-linear Data

Figure 5.12 shows the transformation of the actual data points to a higher order space where the mapping function is "**phie**" and the mapping function is chosen in such a manner that the data points in the higher order space are linearly separable as can be viewed in the figure where the green-plane separates the data of two classes linearly.

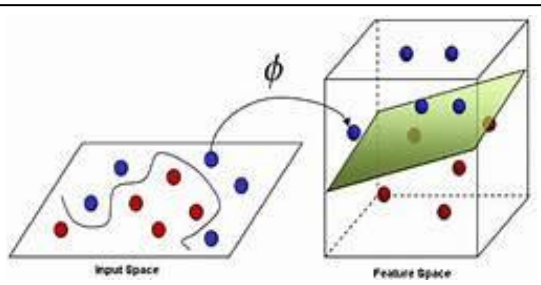


Figure 5.12 Mapping function for transforming data to higher space

The mapping function is chosen based upon the Kernel function which is the dot product of the two vectors in the transformed space. Attach a figure of Kernel Function. Kernel Function helps in speeding up the computation.

The most commonly used Kernel Functions are listed alongside in Figure 5.13.

No	Kernel function	Formula	Optimization parameter
1	Dot -product	$K(x_n,x_i) = (x_n,x_i)$	C
2	RBF	$K(x_n,x_i) = \exp(-\gamma \parallel x_n - x_i \parallel^2 + C$	C and γ
3	Sigmoid	$K(x_n,x_i) = \tanh(\gamma(x_n,x_i)+r)$	C, γ, and r
4	Poly-nomial	$K(x_n,x_i) = (\gamma(x_n,x_i)+r)^d$	C, γ, r, d

Figure 5.13 Kernel Functions

4) Voting Classifier:

It is an ensemble of multiple Machine Learning Algorithms where we train multiple models such as Support Vector Machine (Base Learner 1), Decision Tree (Base Learner 2), Random Forest (Base Learner 3), KNN (Base Learner 4) and many more on the same training data and then we aggregate the results of each of the model and perform voting on the corresponding predictions. The class with the majority is voted by the Meta Learner (as can be viewed from the attached **Figure 5.14**).

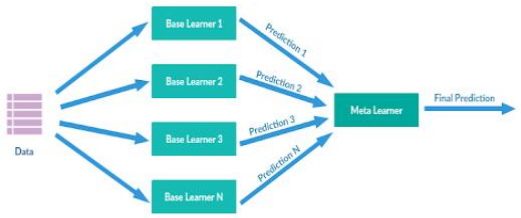


Figure 5.14 Voting Classifier

Chapter-6: Evaluating Machine Learning Algorithms:

While training a machine learning model, you do not know beforehand which algorithm would work best for your problem statement. Hence, with any problem statement, we must train at least 4 to 6 different algorithms such as Logistic regression, support vector machines, decision trees, random forest, etc. and then try to tune the hyperparameters for each of the algorithm and then validate the performance of each of the algorithm on the validation dataset using multiple validation metrics.

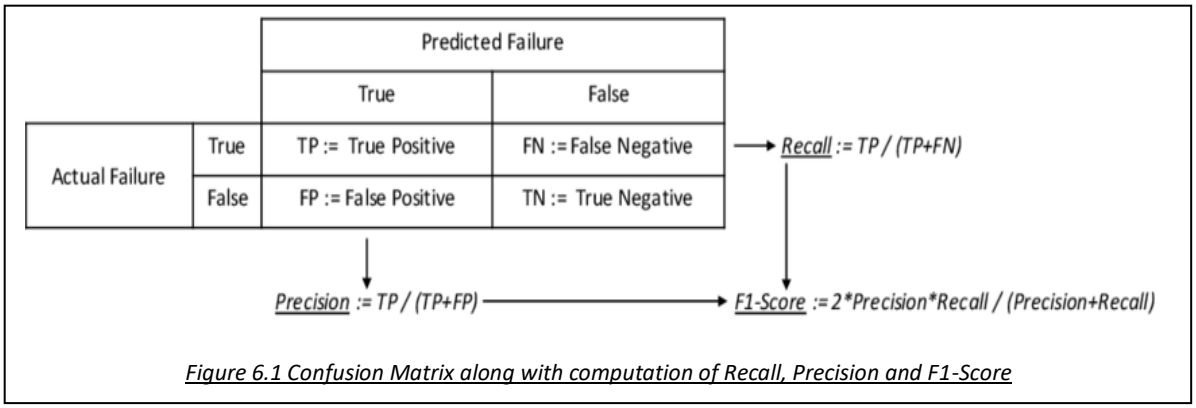


Figure 6.1 Confusion Matrix along with computation of Recall, Precision and F1-Score

1) True Positives and True Negatives: True Positives corresponds to those instances which are predicted positives and their actual labels are also positives, thus correctly classifying instances of positive class. **True Negatives** corresponds to those instances which are predicted as negatives and their actual labels are also negatives, thus correctly classifying the instances of negative class**. False Positives** corresponds to those instances which are predicted positives but their actual labels are negatives, thus misclassification is done**. False Negatives** corresponds to those instances which are predicted as negatives but their actual labels are positives, thus misclassification is done. A Confusion Matrix represents True Positives, True Negatives, False Positives and False Negatives. **Figure 6.1** represents a confusion matrix along with true positives, true negatives, false positives and false negatives.

2) Precision: Precision can be computed using the ratio of True Positives to sum of True Positives and False Positives. In other words, it basically computes the proportion of instances which were correctly classified as positives out all the predictions (in classification, algorithm predicts the discrete numerical value) done by the classification algorithm. Thus, it helps us to understand how correctly the predictions would be made especially in case if you are focusing on more correctly classifying the positive instances.

3) Recall: Recall can be computed using the ratio of True Positives to the sum of True Positives and False Negatives. In other words, it computes the proportion of instances which are correctly classified as positives out of all the instances whose actual label is positive. Hence, this represents how correctly the prediction has been done.

4) F1_Score: **F1 score** is the harmonic mean of Precision and Recall, it will only be large if both i.e. precision and recall both have large values. Thus, in our case, we would like to focus on f1_score we want both of them to be as large as possible based upon our data.

5) Receiver Operating Characteristic (ROC Curve) and Area Under Curve (AUC):

The Receiver Operating Curve is a curve which is plotted with respect to the Sensitivity or True Positive Rate along the x-axis and (1-Specificity) i.e. False Positive Rate along the y-axis. True Positive Rate corresponds to the instances which have been correctly classified as positives and we want True Positive Rate (TPR) rate to be as high as possible. False Positive Rate corresponds to the instances which have been incorrectly classified as positives even though in actual, they belong the negative class. Thus, we want the False Positive Rate (FPR) to be as low as possible. ROC Curve lets us choose the threshold in a way such that the chosen threshold should have maximum TPR and minimum FPR.

Area Under the Curve or AUC is another evaluation metric which is used to compare multiple classification algorithms and choose the one which has the highest Area Under the Curve. The area corresponds to the area lying under to ROC Curve. Figure 1.6 represents a graph Of ROC and Precision Recall Curve of Logistic Regression model evaluated on the test data and we can observe that False Positive Rate is quite high, which means that a lot of the instances which belongs to the negative class are incorrectly classified as those of the positive class. Hence, the model does not seem to be working really good on the test data and is doing a lot of misclassifications. However, the Area under the ROC Curve is 0.7681 or 76.81%, which is not very good and further we would try to select the classification algorithm which has the highest AUC.

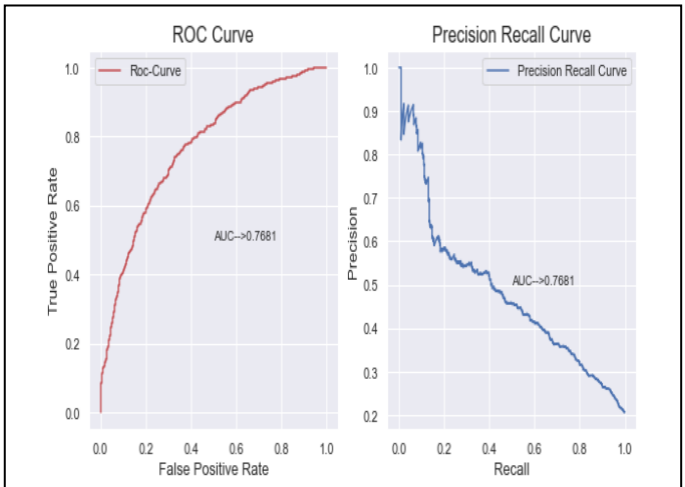


Figure 6.2 ROC Curve and Precision Recall Curve for Logistic Regression evaluated on the test data

6) Precision Recall Curve: Since in our case, we are concerned with the positive class of target feature, hence we want Precision and Recall both to be as high as possible. It should be noted that there exists a trade-off between precision and recall i.e. if Precision is increased, Recall reduces and if Recall is increased, Precision reduces. This characteristic trade-off can be understood by the **Figure** alongside. Thus, we need to choose the values of Precision and Recall in such a manner that both of them remain as high as possible. It can be observed at Precision = 0.46, Recall = 0.46 and this is the maximum trade-off that we can consider, else beyond this, if we try to increase Precision, recall reduces and if we try to increase Recall, precision reduces. Thus, the best values of precision and recall = 0.46 for Logistic Regression model evaluated on test data.

7) Accuracy: Accuracy can be defined as the ratio of sum of true positives and true negatives to the sum of total data points. Error can be defined as the sum of false positives and false negatives to the sum of total data points.

Understanding Learning Curves and Interpreting Overfitting: (Refer Figure 6.3)

- During the initial stages, when the number of training records were pretty less, then it seems that validation error is quite large whereas training error seems to be less: it might be **because we are making predictions on training data and since the model has learnt from training data only, thus it makes better predictions on it (since it is making predictions on the data which it has already seen during training)**. As the number of training instances increases, the validation error also reduces since now, the model is being

trained on more variety of data and thus the model generalizes well. However**, when the model has been trained fully on all dataset**, then the training error and validation error seems to be quite similar, which means **the model now generalizes well on making predictions for unseen data.**

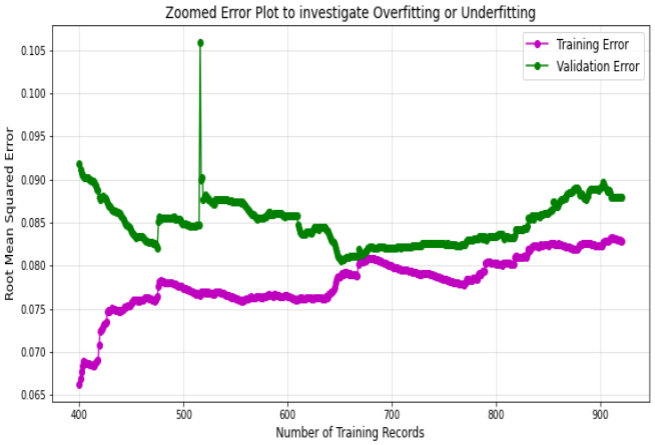


Figure 6.3 Understanding Learning Curves and Analysing Overfitting

- **The model does not seem to get Overfitted on training data since the training error is quite close to validation error** and this is the trademark for a model which generalizes well on test data.
- The model does not seem to be underfitted since the error is pretty low for both training and validation error. Had the **errors of Training and Validation Data been quite high, we could have interpreted that the model is underfitting.**
- It can also be observed that during the initial time when model was trained on less number of records, validation error was pretty high and training error was pretty low which means initially the model was not generalizing well and thus was Overfitted but as the model kept on training on more and more data, it started generalizing well and by the time it gets trained on the whole training data, the generalization error has been reduced to a much lower value as can be observed from the above plots.

Chapter 7: Training Multiple Machine Learning Models for Our Problem Statement, tuning their Hyperparameters and Validating each of them:

We have 4 different features in our application and the features are as follows:

- Loan Repayment
- Loan Approval
- Customer Retention
- Fraudulent Transaction

Thus, for each feature, multiple machine learning algorithms are trained (along with tuning their hyperparameters) and their performance is validated using several performance metrics. The one classifier which seems to outperform the other classifiers is chosen as the final classifier for that particular feature.

1) Loan Repayment:

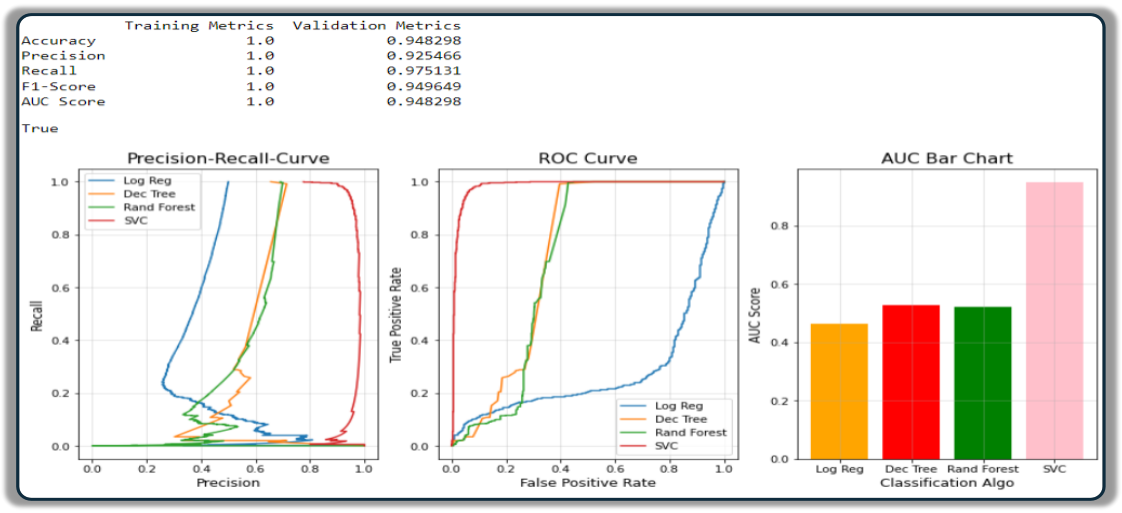


Figure 7.1 Validation Metrics of Loan Repayment

Observations are as follows based upon Figure 7.1:

- For Loan Repayment feature, below 4 algorithms have been trained and their hyperparameters are also tuned:
  - Logistic Regression
  - Decision Trees
  - Random Forest
  - Support Vector Classifier
- The above Figure represents the AUC Bar Chart (comparing the AUC score of the 4 trained algorithms), ROC Curve (comparing the ROC Curve of the 4 trained algorithms) and Precision-Recall Curve (comparing the Precision-Recall Curve of the 4 trained algorithms).
- In the Precision-Recall Curve, it can be observed that the best trade-off for precision and recall is of the Support Vector Classifier (SVC).

- In ROC Curve, the maximum true-positive-rate and the minimum false-positive-rate is for SVC.
- From the AUC Bar Chart, we can observe that SVC has the highest AUC Score i.e. 98%.
- Thus, **SVC seems to outperform the rest of the algorithms** and is chosen as the final classifier for **Loan Repayment**.

2) Loan Approval:

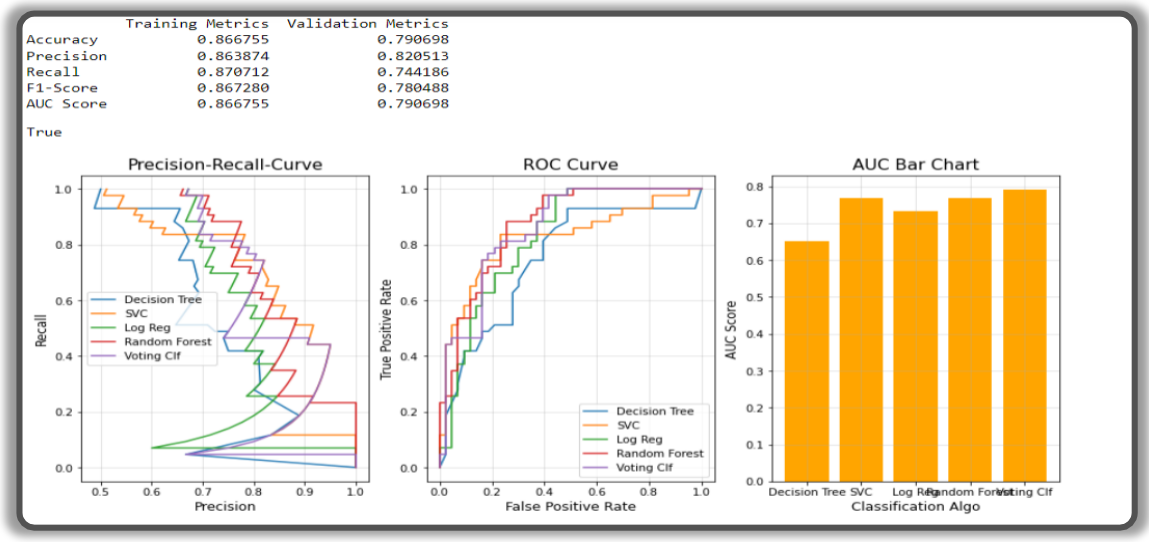


Figure 7.2 Validation Metrics of Loan Approval

Observations are as follows based upon Figure 7.2:

- For Loan Approval feature, below 5 algorithms have been trained and their hyperparameters are also tuned:
  - Logistic Regression
  - Decision Trees
  - Random Forest
  - Support Vector Classifier
  - Voting Classifier
- The above Figure represents the AUC Bar Chart (comparing the AUC score of the 4 trained algorithms), ROC Curve (comparing the ROC Curve of the 4 trained algorithms) and Precision-Recall Curve (comparing the Precision-Recall Curve of the 4 trained algorithms).
- The Precision-Recall Curve and ROC-Curve seem to have haphazard discrete curves because the data points on which the model has been trained were quite less in number. Had there been more data points, the curves would have been slightly continuous as in the case of Loan Repayment feature.
- In the Precision-Recall Curve, it can be observed that the best trade-off for precision and recall is of the Voting Classifier.
- In ROC Curve, the maximum true-positive-rate and the minimum false-positive-rate is for Voting Classifier.

- From the AUC Bar Chart, we can observe that Voting Classifier has the highest AUC Score i.e. 82%.
- It should be noted that the validation metrics seems to be less in number (AUC=82% and not in 90s) since the data points on which the model has been trained are 617 (very small dataset).
- Thus, **Voting Classifier seems to outperform the rest of the classifiers** and is chosen for making final predictions in real time for **Loan Approval**.

3) Fraudulent Transaction:

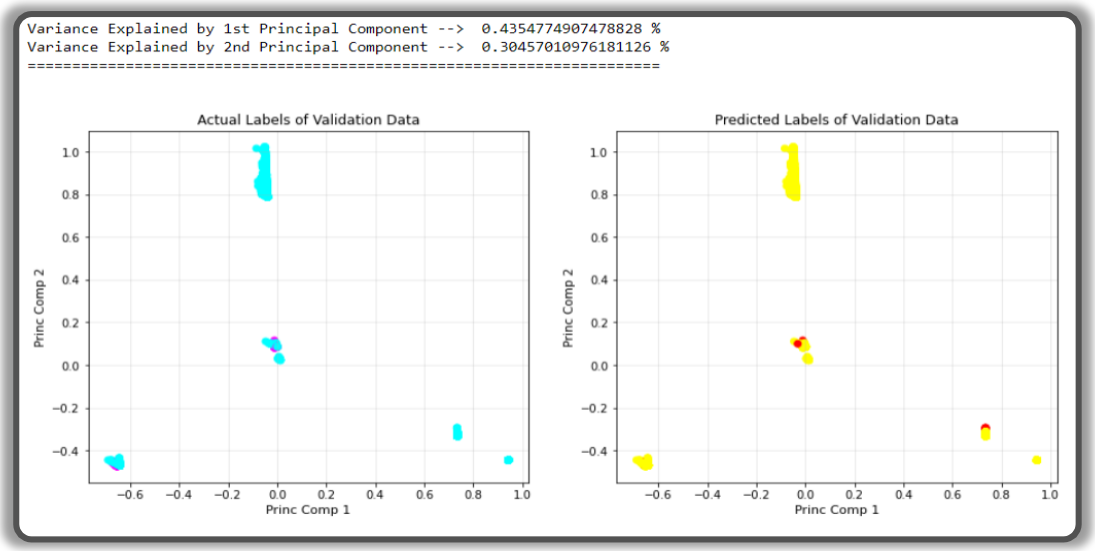


Figure 7.3 Validation Metrics of Fraudulent Transaction

Observations are as follows based upon Figure 7.3:

- Since "**Fraudulent Transaction**" is the problem statement as that of anomaly detection because the proportion of fraudulent data points was around 0.05%. Hence, in such a case, we cannot use the usual Machine Learning Classification Algorithms and we must go ahead with Anomaly Detection Algorithms i.e. Local Outlier Factor and Isolation Forest.
- I have trained these two models and found the predictions of Local Outlier Factor more promising as compared to Isolation Forest.
- Since, there are no predefined metrics for Anomaly detection problem statement, it is best to visualize the fraudulent data points for actual and predicted cases. Since, the data is multi-dimensional, hence we used Principal Component Analysis (PCA) to reduce the dimension of the data set to just 2 dimensions so that the data can be plotted on a scatter plot.
- The first principal component explains 45% of total variance and the Second Principal Component explains 30% of the variance as can be viewed from the figure attached above.
- The first graph **Figure 7.3** is a scatter plot with both the principal vectors along x and y axis respectively and the color of the data points define whether that transaction was **actually fraudulent (pink) or actually normal (blue**).

- The second graph in **Figure 7.3** is a scatter plot with both the principal vectors along x and y axis respectively and the color of the data points define whether that transaction was **predicted as fraudulent (red) or was predicted as normal (yellow).**
- It can be observed that some of the predictions seem to be correct (in the central region) where as the model is misclassifying some of the data points (along lower right side of the predicted graph).
- Thus, Local Outlier Factor seems to work the best and is chosen for making final predictions in real time for Fraudulent Transaction.

4) Customer Retention:

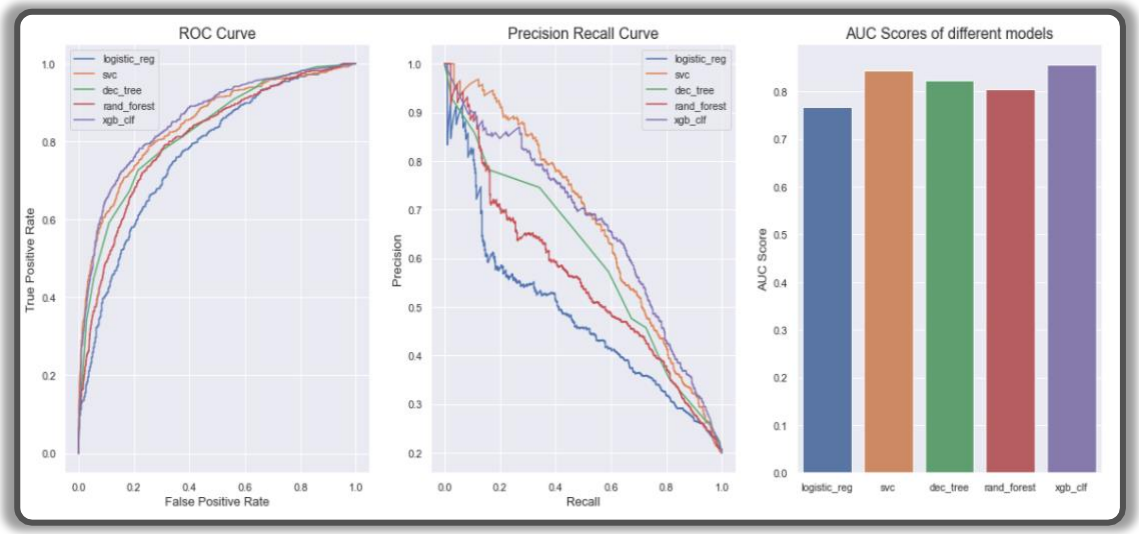


Figure 7.4 Validation Metrics of Customer Retention

Observations are as follows based upon Figure 7.4:

- For Customer Retention feature, below 4 algorithms have been trained and their hyperparameters are also tuned:
  - Logistic Regression
  - Decision Trees
  - Random Forest
  - Support Vector Classifier
  - Xtreme Gradient Boosting Classifier (XG BOOST)
- The above Figure represents the AUC Bar Chart (comparing the AUC score of the 4 trained algorithms), ROC Curve (comparing the ROC Curve of the 4 trained algorithms) and Precision-Recall Curve (comparing the Precision-Recall Curve of the 4 trained algorithms).
- In the Precision-Recall Curve, it can be observed that the best trade-off for precision and recall is of the Support Vector Classifier (SVC) and XB Boost Classifier both.
- In ROC Curve, the maximum true-positive-rate and the minimum false-positive-rate is for SVC and XG Boost Classifier both.
- It seems XG Boost and SVC has quite the same validation metrics. However, we have chosen SVC as the final classifier since it has lesser number of hyperparameters to tune and XG Boost has a lot of hyperparameters to tune. Thus, a lot of computational power is required

to choose the best hyperparameters for XG Boost Classifier since the higher the number of hyperparameters, the more would be the combinations we would be testing with.

- From the AUC Bar Chart, we can observe that SVC and XG Boost have almost the same AUC Score i.e. 84%.
- Thus, **SVC seems to outperform the rest of the classifiers** and is chosen for making final predictions in real time for **Customer Retention**.

Chapter 8: Creation of API using Flask, Deployment of API using Heroku and Testing the API using Postman:

After training and choosing the right models for our problem statements, the next step is to code down classes for each of the feature where the structure of the class remains the same and it has the same methods but the code within these methods changes as per the feature (method overriding is used).

API or Application Programming Interface is the interface which is responsible for secure communication between the client and server where client is our mobile application (in this case) and server is Heroku (Cloud Platform which is Platform As A Service). We create API's in Python using a micro-framework called as Flask where we integrate the above classes and assign each of the class a different URL. Thus, Flask helps us assigning endpoints to each feature (loan repayment, loan approval, etc.). Hence, if we send a request to the endpoint which is attached to Loan Approval, then the "LoanApprovalProcessing" Class's instance would be created and the predictions are make.

We want to host this service for 24 (hours) X 7(days) so that the user can send the request from any part of the world and the cloud makes sure that the response is sent provided no error in deployment is made. Thus, in order to make the api's available all the time, we deploy this Flask App to some cloud service such as Heroku. Heroku is a Platform As A Service(PAAS) where we do not need to provision servers manually, rather we simply need to create an environment in which all of the code is kept along with the dependencies. Rest everything i.e. application runtime, OS of virtual machine, etc. are taken care of by the cloud provider only.

Inheriting Properties of Parent Class using Abstraction:

We have developed 4 different classes i.e. one for each problem statement which are described as follows:

- "FraudulentTransactionProcessing" for "Fraudulent Transaction".
- "LoanApprovalProcessing" for "Loan Approval".
- "LoanRepaymentProcessing" for "Loan Repayment" .
- "CustomerRetentionProcessing" for "Customer Retention".

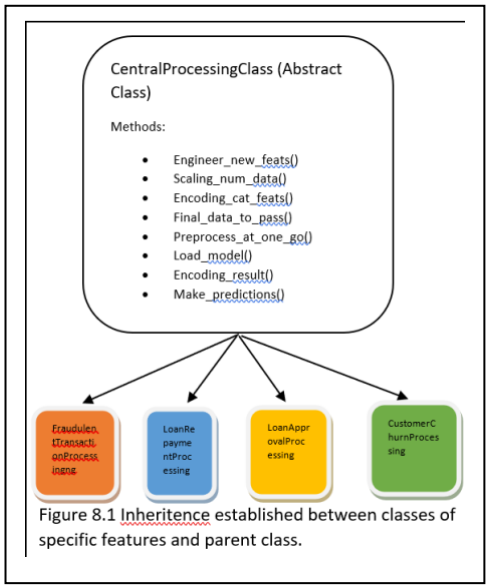


Figure 8.1 Inheritence established between classes of specific features and parent class.

These are all the children classes inherited from the parent class "CentralProcessingClass". Figure 5.8 represents the inheritance architecture between the parent class and the feature specific class. The methods of the parent class "CentralProcessingClass" are as follows:

- "engineer_new_feats ()" -> used for producing new features using data of existence features.
- "scaling_num_data () "-> used for either min-max scaling or standard scaling to scale the data to the same scale.
- "encoding_cat_feats ()" -> used for performing one-hot encoding for the categorical features in this method.

- “final_data_to_pass () “-> used for bringing data in such a format in which it was passed while the model was trained i.e. aligning the features in the correct order as it was while the model was trained.
- “preprocess_at_one_go()” -> we perform all the preprocessing steps over here by calling each of the above methods one by one and converting the data to a data frame.
- “load_model()” -> the model is being loaded over here and then it is being returned.
- “encoding_result()” -> We encode the result in this method i.e. we return the final message through this method only.
- “make_predictions()” -> we make predictions in this method by taking the preprocessed data and loading the final trained model.

Figure-8.1 shows the inheritance structure of how the Children classes inherit the methods defined in the parent class. The classes are coded in exactly the same manner as demonstrated by this figure. There is a parent class “CentralProcessingClass” which is an abstract class and hence all the methods defined within this class, as mentioned in the Figure are abstract methods. Further, we inherit all of our classes for each feature such as “FraudulentTransactionProcessing” for detecting the fraudulent transaction, “LoanApprovalProcessing” for making predictions for loan approval, “LoanRepaymentProcessing” for making predictions for loan repayment and “CustomerChurnProcessing” for making predictions if the customer would leave the services of the bank. Thus, all of these children classes override the methods defined In the parent class.

The above classes are assigned different routes or URL and we need to send a POST request at this URL with the data entered by the Bank Manager for a specific feature.

Testing the API using Postman:

After successfully deploying the code on Heroku, we test the API’s using a tool called as Postman. There are the screenshots of Postman attached where we “Send” a POST request to the deployed URL along with the data that would have been entered by the Bank Manager from the front end and a response would be returned.

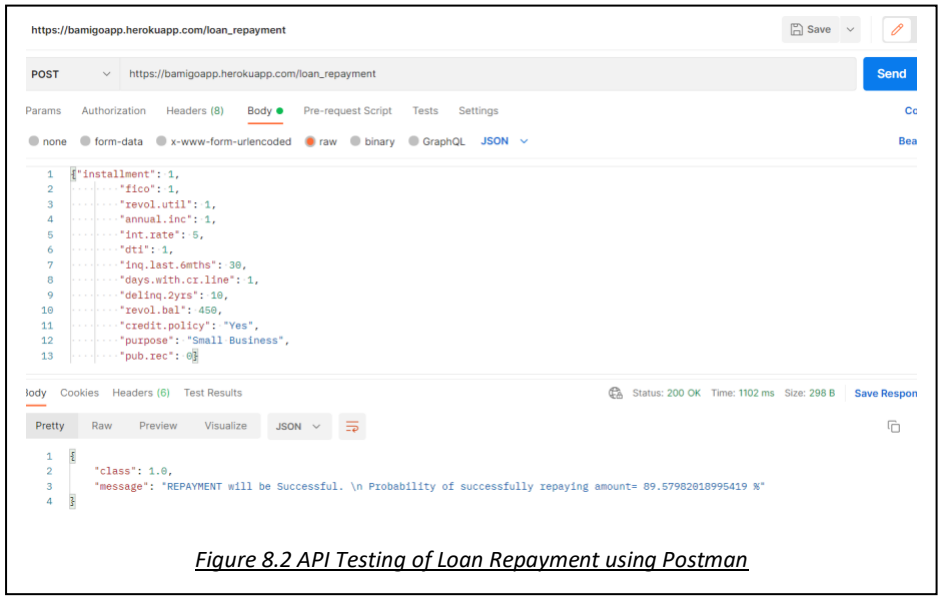


Figure 8.2 API Testing of Loan Repayment using Postman

Figure-8.2 is a screenshot of Postman where we send a POST request to “https://bamigoapp.herokuapp.com/loan_repayment“ along with the whole data which we would be receiving from the Bank Manager. The data over here is in JSON format. We “Send” the request and wait for the response from the server. After few seconds, we receive a

message in json as follows: {“class”: 1.0, “message”: “REPAYMENT will be Successful. The probability of repaying amount = 89.57%.”} where we display the message on the application.

In the same way, we test API’s for each of the feature of our application.

CHAPTER 9: B-AMIGO, ANDROID APPLICATION

The Project B-Amigo is a Banking Software Application developed using flutter with beautiful designs, smooth animations, and great performance.

Flutter is a framework that has provided us with an environment to build a perfect application with all the desired functionalities.

The layout of the application is as follows:

The introductory pages consist of the logo and name of our application i.e., B-Amigo "The Prediction Baba" as shown in the illustration followed by the page consisting of the four modules of our project.

Each icon is hyperlinked with its corresponding feature where the customer details are entered by the bank manager and predictions can be made.

Fig 9.1 Features of "B-Amigo"

The introductory page consists of the four feature as shown in **Figure 9.1**.

Fraudulent Transactions

This module helps the bank employees to detect whether the transaction made was fraud or not on the basis of the provided data. It further helps the bank in preventing fraudulent transactions.

S no.	Fields	Meanings
1.	Step	
2.	Old Balance of Origination	It is the amount; the customer has in his/her account originally or before the transaction.
3.	New Balance of Origination	The amount present in an account after the transaction is made.
4.	Old Balance of Destination	The amount of money present in the account where the money has to be transferred.
5.	New Balance of Destination	The amount shown in the account of the receiver after the transaction has been made.
6.	Type of Transaction	The mode to payment used by the sender to transfer money to the receiver i.e., cash, transfer, debit etc.

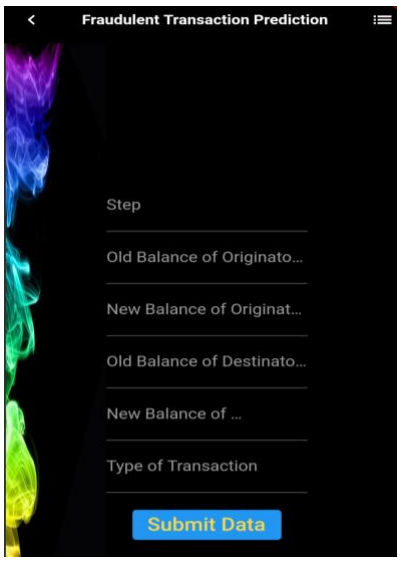


Fig. 9.2 UI for "Fraudulent Transaction" Prediction

After entering the data of the customer, the possible labels that can be predicted from this model are "YES" or "NO" as shown in the **Figure 9.3** and **Figure 9.4**. The percentage displayed below the decision is the probability percentage.

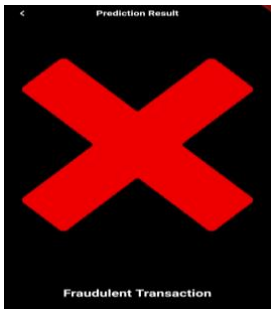


Fig 9.3 UI for "Fraud Transaction"

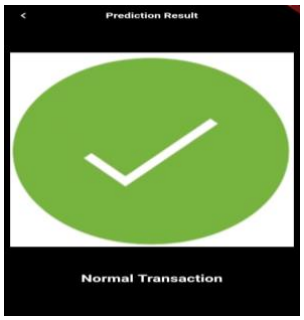


Fig 9.4 UI for "Normal Transaction"

Loan Approval:

This module as explained in the introduction of the Application answers the problem statement "Should the bank manager approve the loan of a customer or not?" based on the binary and categorical features data gathered about the customer. The fields required in the feature are:

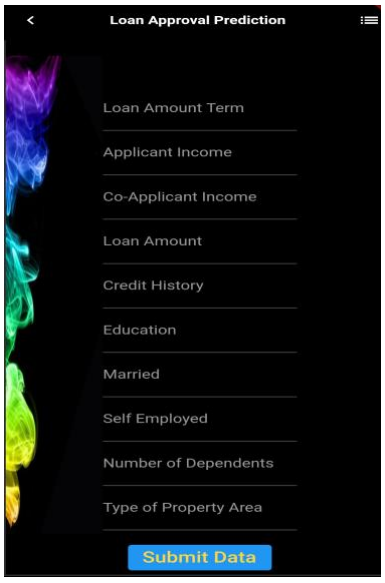


Fig.9.5 UI for "Loan Approval" Prediction

S No.	Fields	Meanings
1.	Loan Amount Term	The amount of time a customer gets to repay the loan by the lender.
2.	Applicant Income	Income of a person at whose request or for whose account a letter of credit is issued.
3.	Co-Applicant Income	Income of an additional person who is considered in the underwriting and approval of a loan.
4.	Loan Amount	The amount of money a customer owes the bank at any given time.
5.	Credit History	The record of how a person has managed his or her credit in the past, including total debt load, number of credit lines, and timeliness of payment
6.	Education	The educational background or the qualification of the customer.
7.	Married	One's situation with regard to whether one is single, married, divorced or widowed.
8.	Self Employed	One's situation with regard to whether he/she is working for oneself or not.
9.	Number of Dependents	Number of those family members who rely on the loan applicant for financial support.
10.	Type of Property Area	The type of property i.e., Urban, Semi-Urban or Rural that the applicant wishes to buy with the loan amount.

After entering the data of the customer, the possible labels that can be predicted from this model are "YES" or "NO" as shown in the **Figure 9.6** and **Figure 9.7** The percentage displayed below the decision is the probability percentage.

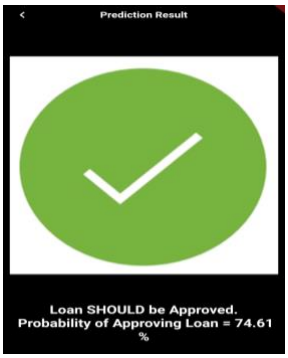


Fig.9.6 UI for Loan Approval "YES"

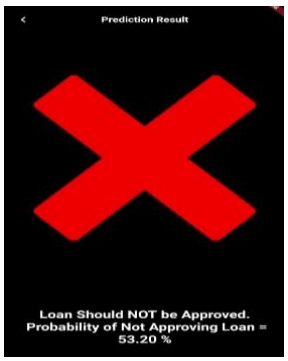


Fig 9.7 UI for Loan Approval "NO"

Loan Repayment

This module predicts whether a bank's customer will be able to repay the loan or not, tht has been approved by the bank. The feature requires the following details of a customer:

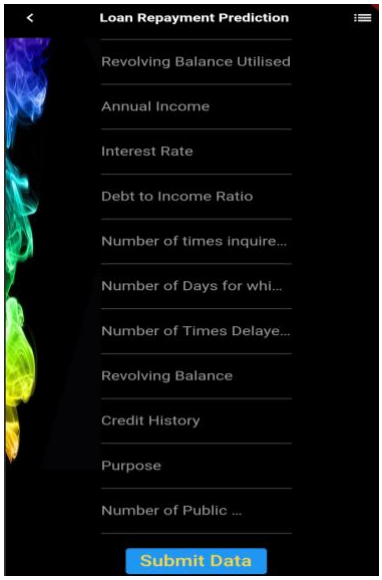


Fig.9.8 UI for "Loan Repayment" Prediction

S no.	Fields	Meanings
1.	Instalments per month	a fixed payment made by a borrower to the bank on a specified date of each month.
2.	FICO Score	is a three-digit number, typically on a 300-850 range, that tells the bank how likely a consumer is to repay borrowed money based on their credit history.
3.	Revolving Balance Utilised	The balance that carries over from one month to the next is the revolving balance on that loan.
4.	Annual Income	The amount of money the customer earns in a year
5.	Interest Rates	The sum charged on the instalments of the loan which are to be paid every month.
6.	Debt to Income Ratio	The ration that compares how much you owe each month to how much you earn.
7.	Number of times inquired in last 2 years	The number of inquiries made by the manager in the last two years.
8.	Number of days for which the loan is approved	The number of days for which the loan has been approved
9.	Number of times delayed in 2 years	Number of times, the customer delayed in repaying the instalments of loan in last 2 years.
10.	Revolving Balance	The balance that carries over from one month to the next is the revolving balance on that loan.
11.	Credit History	The record of how a person has managed his or her credit in the past, including

		total debt load, number of credit lines, and timeliness of payment
12.	Purpose	Purpose is the motive for which the loan is requested e.g., home loan, car loan etc.
13.	Number of Public Documents	The number of public documents of the customer available in the bank e.g., Adhaar, PAN etc.

There are two possible outputs this feature too. It can be either a “YES” or “NO”, and the probability percentage also displays below the result as shown in the fig 9.9.

Fig 9.9 UI for Loan Repayment = “YES”

Customer Retention

This module of our application B-Amigo helps answer questions like, “Will the customer continue with the services of the bank or not?” on the basis of the customers’ data gathered. The module consists of the following fields:

S no.	Fields	Meanings
1.	Credit Score	The score assigned on the basis of the records of how a person has managed the credit including total debt load, number of credit lines, and timeliness of payment.
2.	Gender	A binary categorical feature asking if the customer is either male or female.
3.	Age	Numerical feature asking for the age of the customer.
4.	Bank Balance	The amount of money present n the customer’s bank at that point of time.
5.	Number of Services or Products	The number of services the customer has enrolled for.
6.	Active Members	Whether the customer is an active member or not.
7.	Geographical Region	The region in which the customer has a bank account.

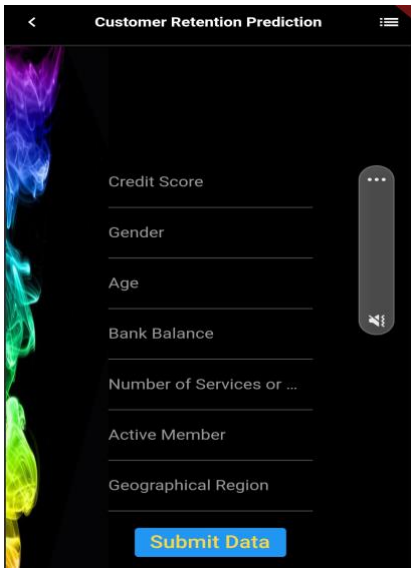


Fig9.10 UI for “Customer Retention”

After entering the data of the customer, the possible labels that can be predicted from this model are “YES” or “NO” as shown in the **Figure 9.11** and **Figure 9.12**.

The percentage displayed below the decision is the probability percentage.

Fig 9.11 UI for Customer Retention “NO”

Fig 9.12 UI for Customer Retention “YES”

Conclusion

Hence, our project "B-Amigo" satisfies all the objectives, mentioned below.

- *Quick decisions can be made by the bank employees (Users).*
- *Date base integration in B-Amigo also helps in storing the customers' data securely along with the predictions made by the bank managers. This data can also be retrieved whenever needed.*
- *The simple interface of B-Amigo application helps the managers to easily navigate between the four features of the application, i.e.,*
    1. *customer retention*
    2. *loan approval*
    3. *loan repayment*
    4. *detection and prevention of fraudulent transactions*
- *The objective of automating the tedious operations of the banking sector, going nearly paperless, reducing chances of error that ultimately results in better ties of banks and their customers is also fulfilled.*

Future Scope

All over the world, banks are taking a huge step towards digitalization in order to cope up with the competition and deliver the maximum to its customers.

The future scope of our project lies in the digitalization that has transformed the manual process into digital service by reducing human error and thus, saving time and building customer loyalty. For Example, the customers have been facilitated by the technology of cashless transactions, online banking and what not.

On realising that the technology keeps on evolving, our project "B-Amigo" (The Prediction Baba) is built on one of the booming technologies, Artificial intelligence that helps in faster and efficient decision making by mimicking the human brains.

The futuristic technologies like machine learning, data science, cloud computing and so on, helps in analysing the customers for a bank's better performance by making predictions beforehand in certain aspects. The model ultimately saves time, unnecessary workload and lengthy duties.