BUDT704 0506 Data Processing and Analysis in Python

Team Members:

Srivats Narain

Gunakshi Sharma

Aishwarya Sadagopan

Madathil Geetanjali Menon

Sai Sravanthi Varanasi

Varsha Sharma

Introduction

Project Name: Bank Loan Default Prediction

Introduction:

This project aims to tackle the day-to-day challenge faced by financial institutions, which is loan payment default. We have implemented 3 machine learning models that can be used as a reference tool to predict if a person will default on a loan payment based on credit score and other information provided by them. These models will help financial institutions to make the decision on whether to issue the loan or not so that the risk can be minimized and at the same time profit can be maximized.

Project Method:

- The first step has to be to obtain the required information or data to train or test Bank Loan Default Prediction. The data has been obtained from Kaggle
- The second step is to perform Exploratory Data Analysis on the obtained data. This ensured that the data was in the required format for prediction
- The third step was to clean the data after EDA, in this manner, all the outliers and redundant information were removed from the dataset
- The final step was to build the model in order to predict bank loan status. Several models have been used to complete this step, which helped in identifying the best model suited for the application

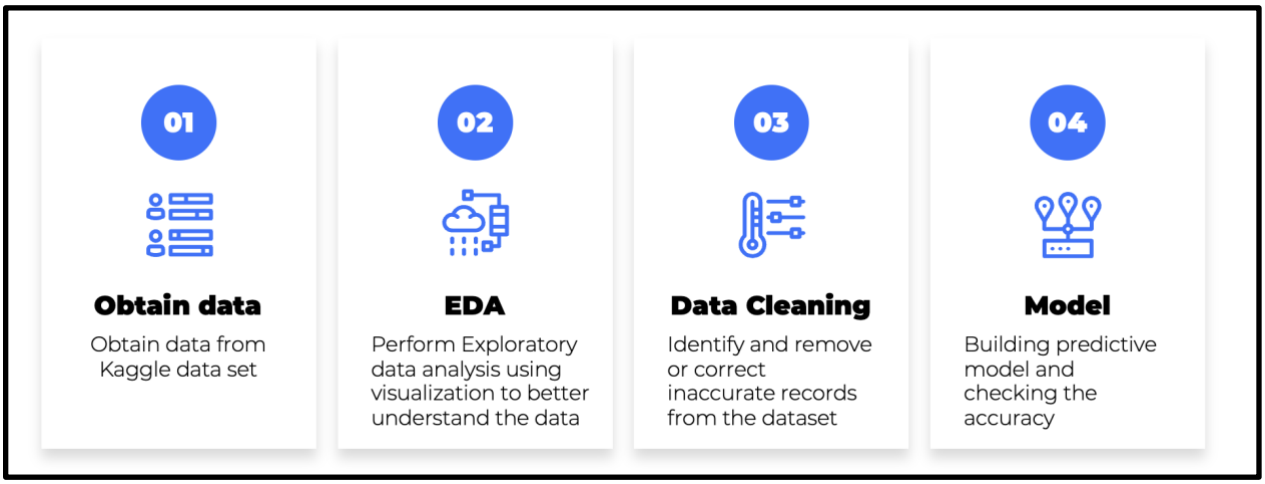

Figure 1: Project Steps

Data Cleaning and Exploratory Analysis:

The dataset obtained from the website consists of 100,000 records with 19 columns including Loan ID, Customer ID, Loan Status, Current Loan Amount, Term, Credit Score, and so on. Even though the dataset is available as an excel and in a nice tabular format, the dataset does have a number of issues, thus considerable data cleaning would still be necessary before any analysis could be performed. These examples of various cleaning techniques are implemented.

1. Removing redundant columns: Some columns such as Loan ID , Customer ID, and Months since the last delinquent, as they have little to no impact on our prediction and hence have been removed.

```
##Removing redundant columns
df.drop(columns=['Loan ID','Customer ID'],inplace=True)
df.drop(columns='Months since last delinquent',inplace=True)
```

2. Removing outliers: An observation that differs abnormally from other values in a population-based random sample is referred to as an outlier. Outliers make data more variable, which reduces statistical power. Therefore, eliminating outliers can make your findings statistically significant. We have removed outliers from column Years of Credit History.

```
##Filtering out outlier data for Years of Credit History using Boolean Mask

df = df[df['Years of Credit History'] <=50]
```

3. Removing null values: We have dropped Null Values in columns such as Bankruptcies, by filling null values with a value and then dropping that value, in order to achieve clean data.

```
##Adding null values with zero
df['Years in current job'].fillna(0,inplace=True)
```

```
##Filtering out values of Years in current job that is not equal to 0
df = df[df['Years in current job'] != 0]
```

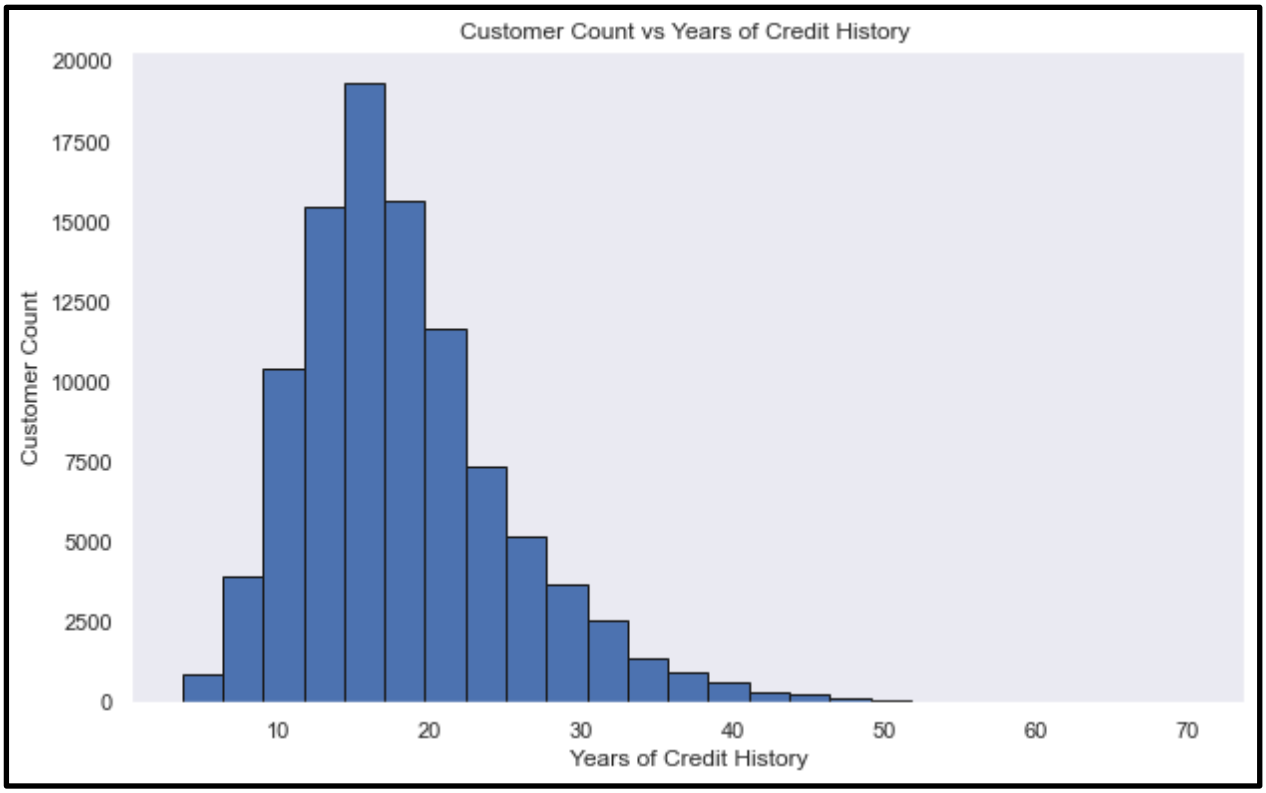

Figure 2: Customer Count vs Years of Credit History

4. Encoding Categorical Variables: All input and output variables must be numbers for machine learning models. This means that before you can fit and evaluate a model, you must encode categorical variables in the data into numbers. We have encoded categorical variables for columns such as Terms, Purpose, Home Ownership, and Years in current job.

```
##Encoding values of Term, Years in current job, Purpose, Home Ownership
df['Term'].replace(to_replace=['Short Term', 'Long Term'], value=[1,0],inplace=True)
df['Years in current job'].replace(to_replace=['8 years', '3 years', '< 1 year', '2 years', '10+ years',
'4 years', '5 years', '1 year', '7 years', '6 years', '9 years'], value=[1,2,3,4,5,6,7,8,9,10,11],inplace=True)
df['Home Ownership'].replace(to_replace=['Home Mortgage', 'Own Home', 'Rent', 'HaveMortgage'], value=[1,2,3,4]
df['Purpose'].replace(to_replace=['Home Improvements', 'Debt Consolidation', 'Buy House', 'other',
'Take a Trip', 'Other', 'Business Loan', 'Buy a Car',
'small_business', 'Medical Bills', 'vacation',
'Educational Expenses', 'wedding', 'major_purchase', 'moving',
'renewable_energy'], value=[1,2,3,4,5,4,6,7,8,9,10,11,12,13,14,15],inplace=True)
```

5. Imbalance Data: Imbalanced data are those datasets that have an uneven distribution of observations across the target class, i.e., one class label has a very high number of observations while the other has a very low number. We have used RandomOverSampler with 0.7 as the sampling_strategy value, which means, the converted dataset would have 700 examples of the minority class if the majority class had 1,000 examples and the minority class had 100.

```
## Using RandomOverSampler we are trying to resolve over sampling by adding duplicate rows from minority class in the data.
over_sampling = im.over_sampling.RandomOverSampler(sampling_strategy=0.7)
X_over_sampling, y_over_sampling = over_sampling.fit_resample(X,y)
```

```
#Decision variable Loan Status count before data resampling
y.value_counts()
```

```
1.0    49997
0.0    11914
Name: Loan_Status, dtype: int64
```

```
#Decision variable Loan Status count after data resampling
y_over_sampling.value_counts()
```

```
1.0    49997
0.0    34997
Name: Loan_Status, dtype: int64
```

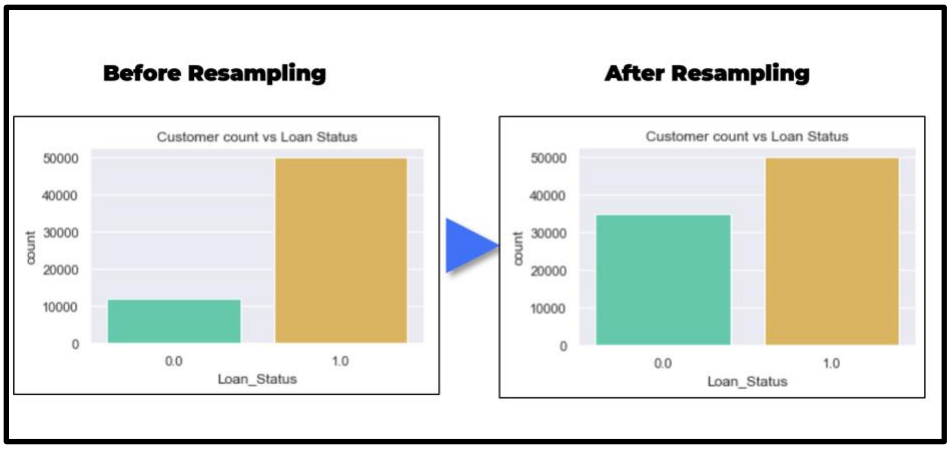

Figure 3: Before and After Data Resampling

Following data cleaning, numerous plots are created to evaluate each attribute and explore their relationships. Before modeling, the objective is to become familiar with the dataset and find any evident trends. To express the interdependencies between two quantitative, continuous variables, correlation is a tool for examining their relationship. Typically, a correlation plot includes a number of numerical variables, each of which is represented by a column. The relationships between each pair of variables are shown by the rows. Positive values indicate a strong relationship, while negative values indicate a weak relationship. The values in the cells represent the strength of the relationship. Every pair of the dataset's correlation coefficients is calculated and presented as a heatmap.

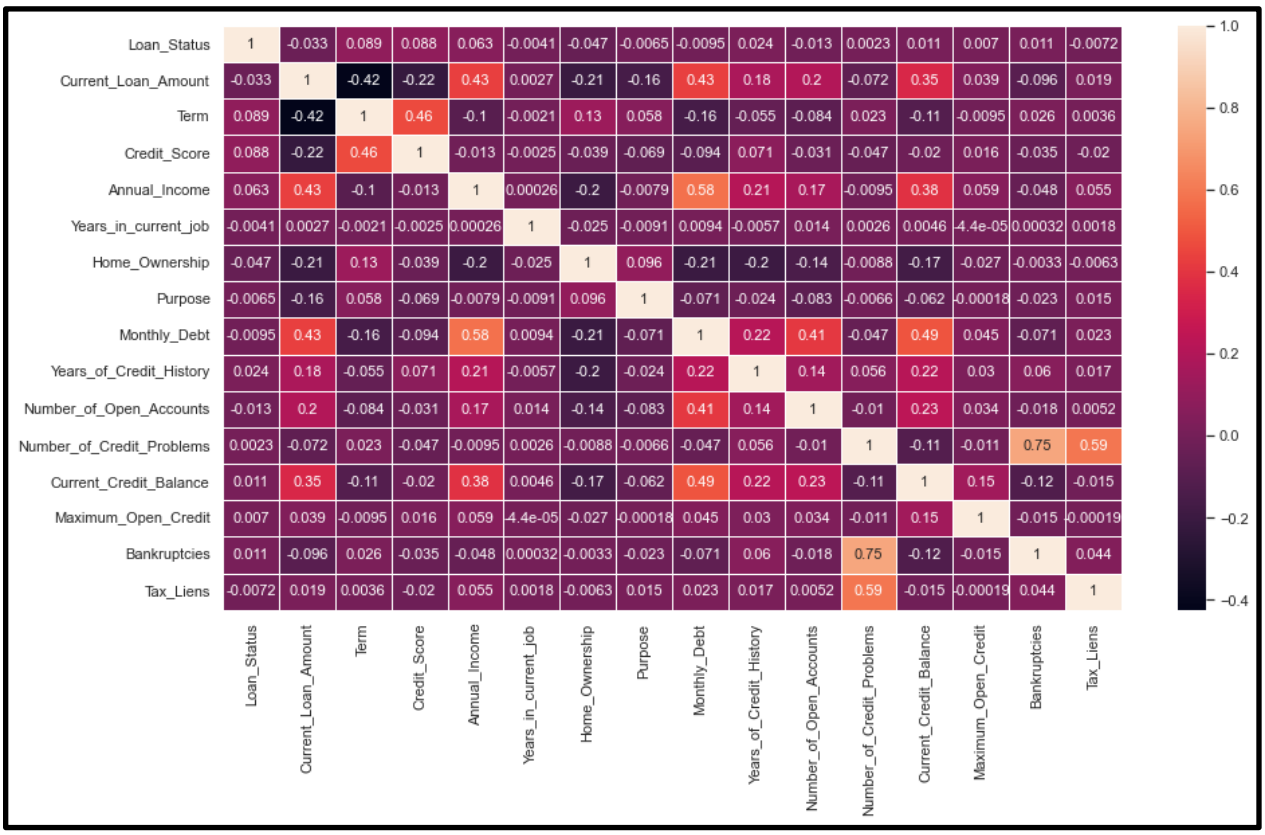

Figure 3: Correlation Matrix

Model:

Logistic Regression:

A classification procedure called logistic regression is used to group observations into discrete classes. In our project is used to answer the question of whether there will be a default on a loan payment or not, we have implemented logistic regression wherein we have created two variables i.e. Independent and dependent variables. Variable X, which is an independent variable, has columns such as Current_Loan_Amount, Term, Credit_Score, Annual_Income, Years_in_current_job, Home_Ownership, Purpose, and so on. Similarly, variable Y, which is our dependent variable, has a Loan_Status column.

Before resampling the data, the accuracy was around 81.03%. Upon looking at the predicted values, we noticed that all of the customers were classified as “Fully Paid” customers - a typical data imbalancing case.

```
prob_accuracy_score = skl.metrics.accuracy_score(y_test,y_pred)
prob_accuracy_score
```

```
0.8103801012167546
```

```
np.unique(y_pred, return_counts=True)
```

```
(array([1.]), array([18574], dtype=int64))
```

```
#Creating object for LogisticRegression()
logreg = LogisticRegression()

#Using train_test_split to partition the data into train and test.

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=42)

print(X_train.shape)
print(y_train.shape)
```

```
(43337, 15)
(43337,)
```

Here, we faced the issue of data imbalance and which we handled using RandomOverSampler, as explained in the data cleaning process and achieved an accuracy of 77.74%. Upon looking at the predicted values, around 1.6k customers were now classified as “Charged off” while ~17k were classified as “Fully paid”

```
## Using RandomOverSampler we are trying to resolve over sampling by adding duplicate rows from minority class in the data.
over_sampling = im.over_sampling.RandomOverSampler(sampling_strategy=0.7)
X_over_sampling, y_over_sampling = over_sampling.fit_resample(X,y)
```

```
y_over_sampling.value_counts()
```

```
1.0    49997
0.0    34997
Name: Loan_Status, dtype: int64
```

```
print(X_over_sampling.shape)
print(y_over_sampling.shape)
```

```
(84994, 15)
(84994,)
```

```
logreg.fit(X_over_sampling,y_over_sampling)
y_pred_over = logreg.predict(X_test)
```

```
prob_accuracy_score_over = skl.metrics.accuracy_score(y_test,y_pred_over)
prob_accuracy_score_over
```

```
0.7742543340152902
```

```
np.unique(y_pred_over, return_counts=True)
```

```
(array([0., 1.]), array([ 1618, 16956], dtype=int64))
```

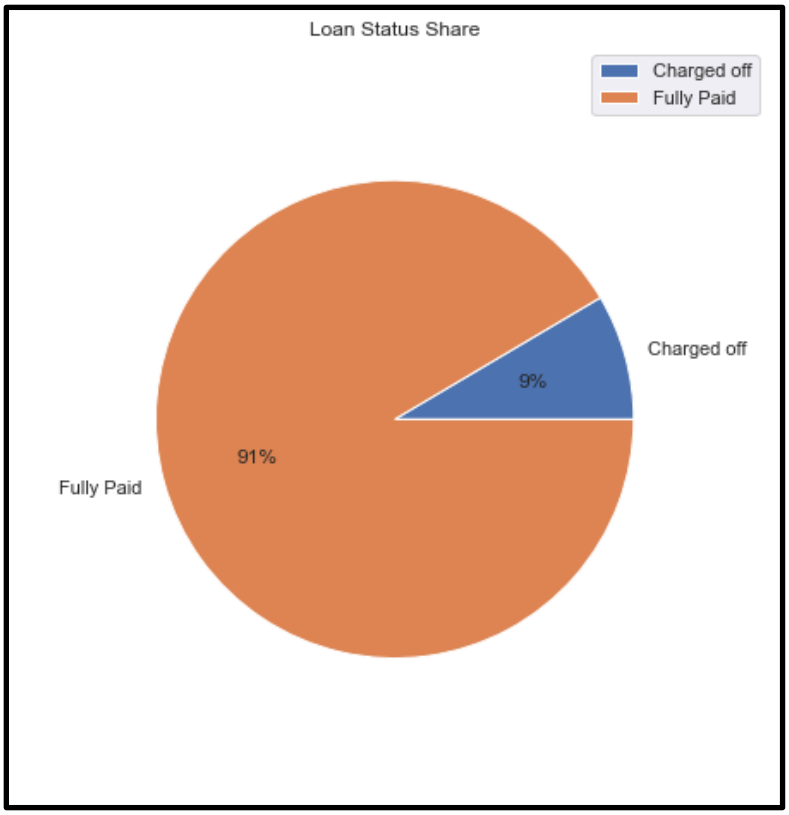

Figure 4: Logistic Regression Output

Decision Tree Algorithm:

A decision tree is a model made up of a number of "questions" arranged in a tree-like hierarchy. Common names for the questions include condition, split, and test. Each leaf node and each non-leaf node carry a condition and a prediction. The decision tree's prediction is the value of the leaf that was reached. The inference path is the collection of nodes that were visited.

```
# Create Decision Tree classifer object
dt = DecisionTreeClassifier(criterion="entropy", max_depth=4)

# Train Decision Tree Classifer
dt.fit(X_over_sampling, y_over_sampling)

# making predictions

prediction = dt.predict(X_over_sampling)
print('Prediction {}'.format(prediction))
```

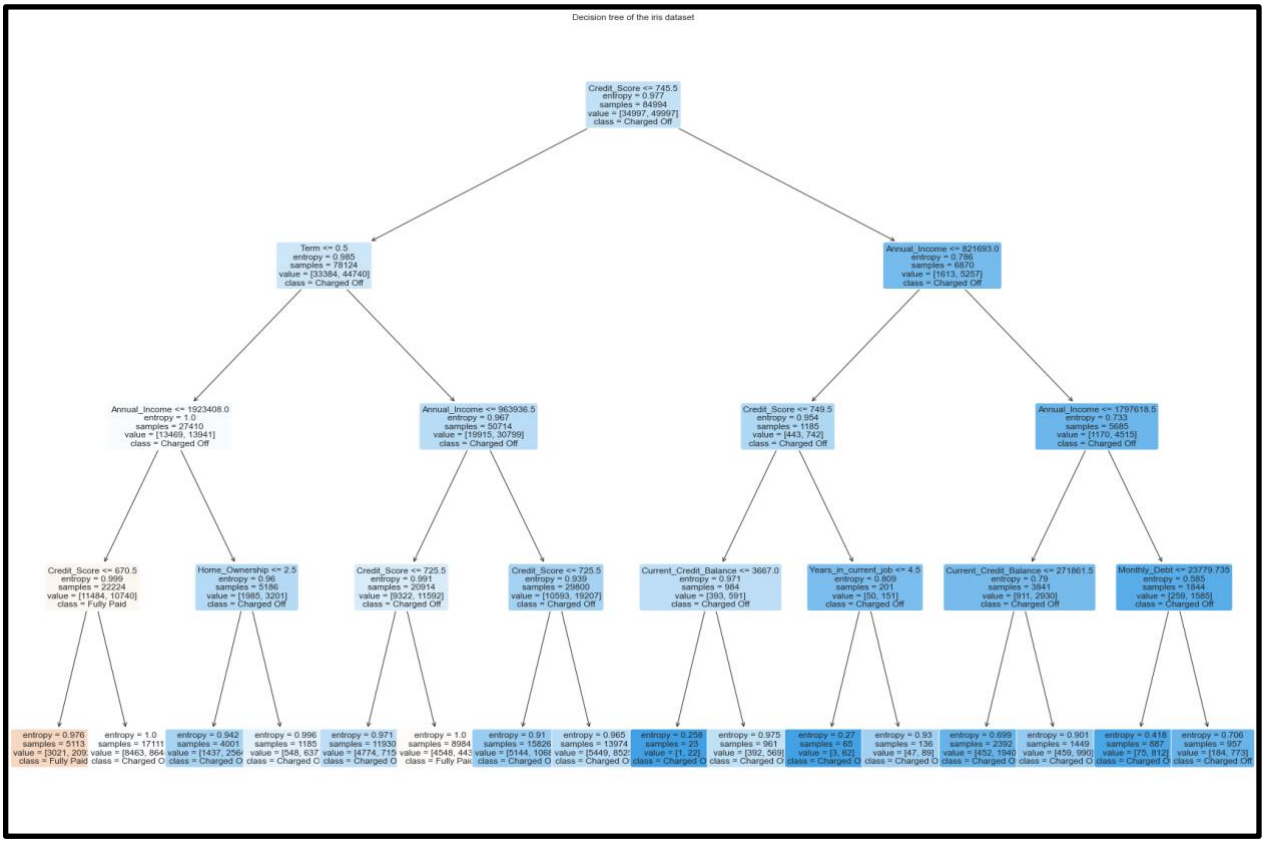

Figure 5: Decision Tree Output

Later, we plotted a graph to see the accuracy change with increase in max_depth parameter. The model's accuracy tends to increase as the max_depth is increased.

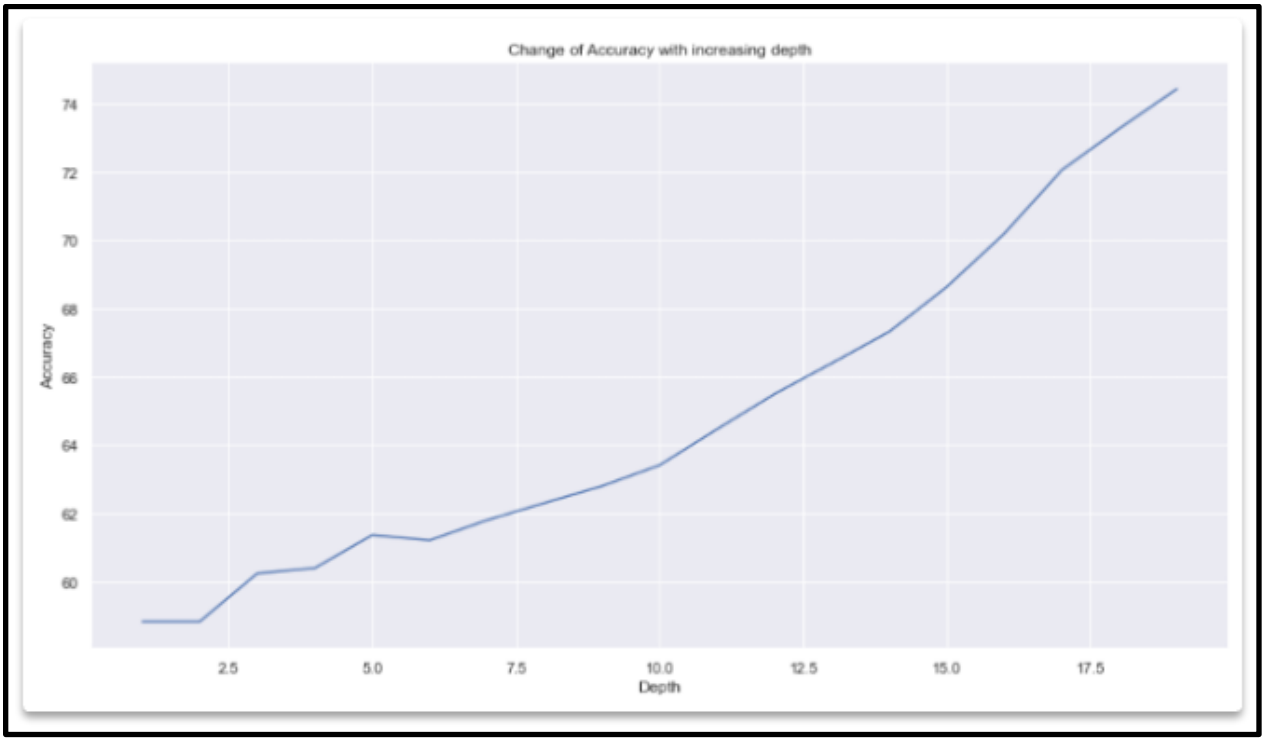

Figure 6: Change of accuracy with increasing depth

KNN Algorithm:

KNN is a supervised learning algorithm. In its simplest form, supervised learning refers to the process of training a computer system utilizing labelled data, which indicates that the right answer has already been assigned to certain data. In order for the supervised learning algorithm to analyze the training data (set of training examples) and provide an accurate result from labelled data, the machine is then given a fresh set of examples (data).
The KNN algorithm believes that related things are located nearby. In other words, related things are located close to one another.

```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=6)
knn.fit(X_train_dec, y_train_dec)
```

Later, we plotted a graph to see the accuracy change with an increase in the number of neighbors.
We observed that the model's accuracy tends to decrease with an increase in the model parameter – neighbors

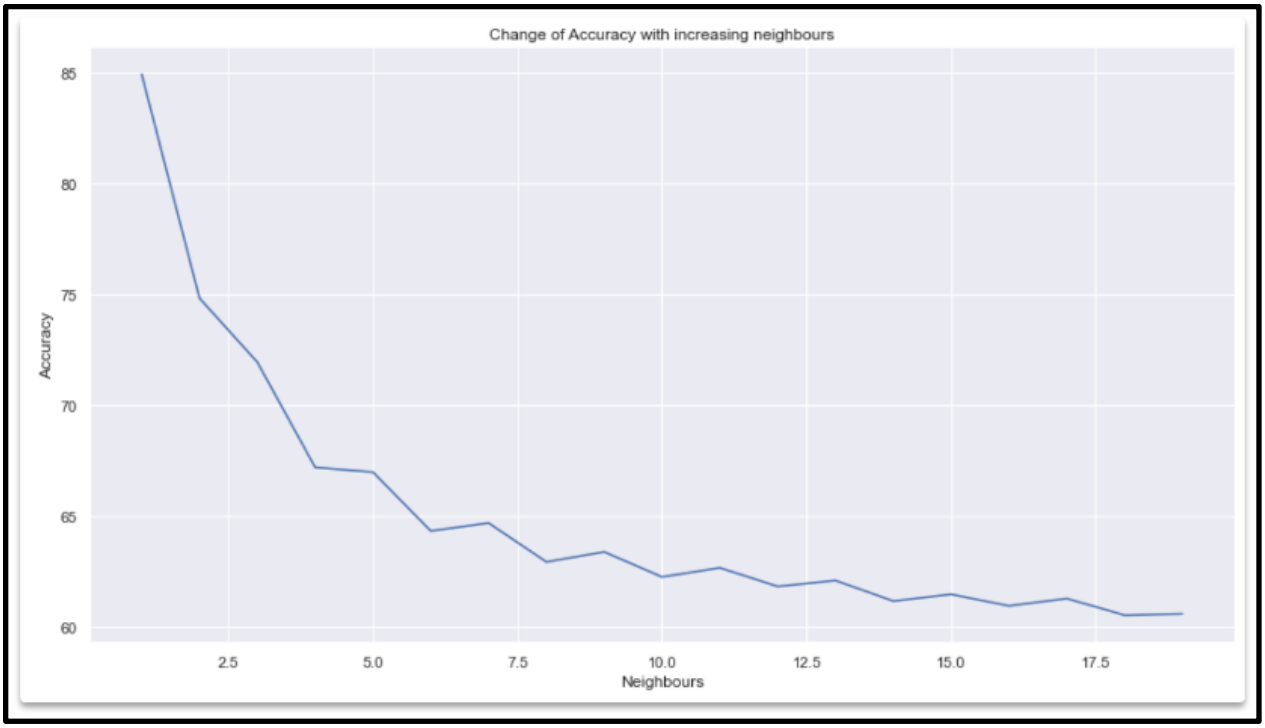

Figure 7: Change of accuracy with increasing neighbor

POST MODEL ANALYSIS

Logistic Regression

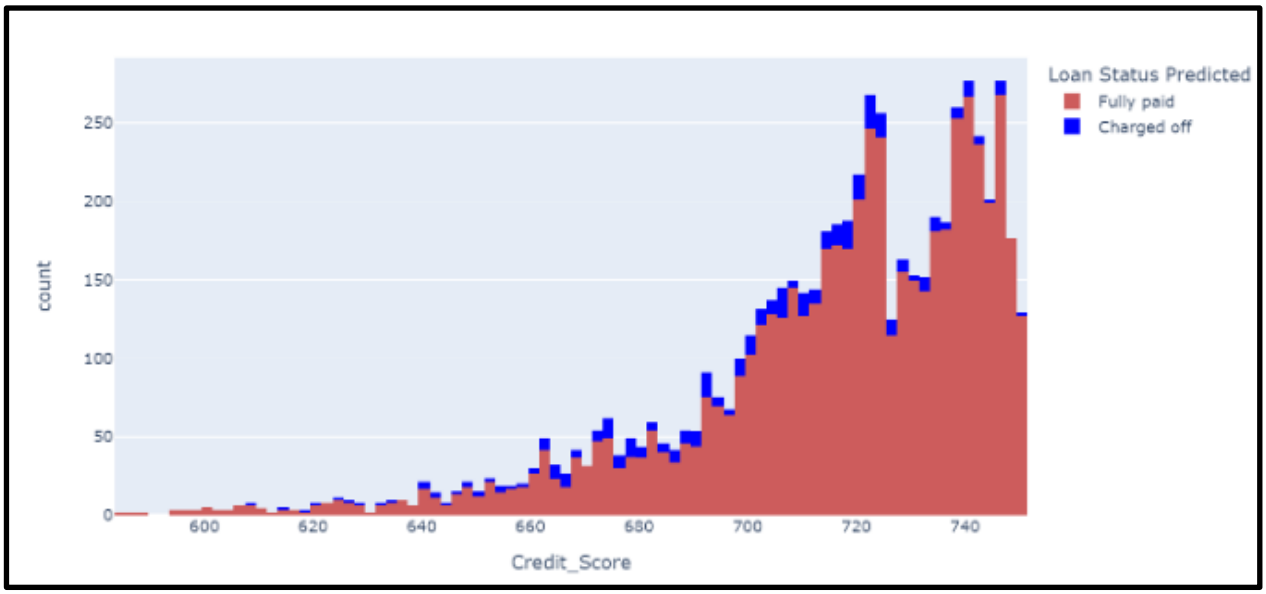

Figure 8: Count Vs Credit Score

Due to improved model performance compared to the other two models, logistic regression exhibits a higher accuracy and therefore a better fit for the dataset chosen.
From Figure 8, we see that people categorized as “Fully Paid” and “Charged Off” are in the credit score range of [640,740].
To avoid any further financial risks for the company we need to scrutinize the loan sanctions from this credit range

Decision Tree

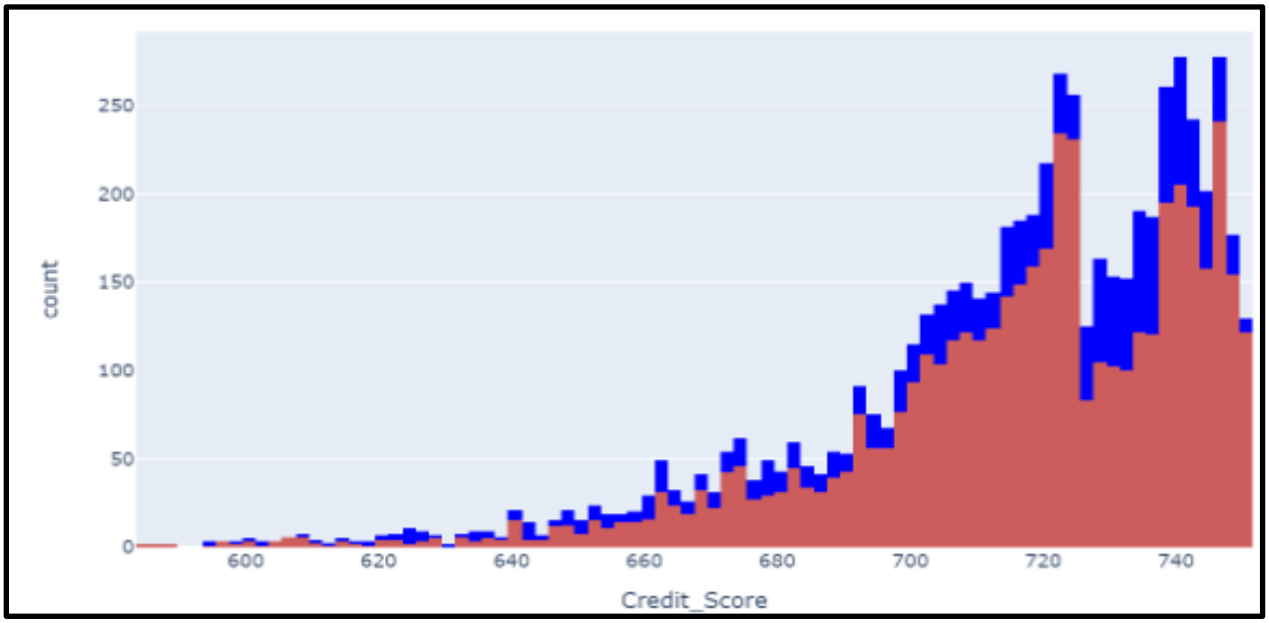

Figure 9: Count Vs Credit Score

Having the next best model accuracy, the decision tree is the second choice for performing prediction on the dataset.