Banking Customers Churn Prediction using Machine Learning Models

Group 6

Iqra Bismi (016039842)
Janani Ravi Kumar (016044080)
Saniya Lande (016013218)
Shilpa Shivarudraiah (016079973)

Contents

1 Introduction

1.1 Project Background and Problem Definition

Banking sector is a network or group of financial institutions that engages in providing banking services to corporate and individual customers through direct or a third party client service. A bank is a financial institution licensed to receive deposits and make loans. Banks may also provide financial services such as wealth management, currency exchange. The banking sector has become one of the main industries in developed countries. The technical progress and the increasing number of banks raised the level of competition among them. Banks are working hard to survive in this competitive market by implementing multiple strategies.

Churn is the measure of how many customers stop using the product or service. It can be based on actual usage or failure to renew. Churn analysis involves analyzing historical customer data to make churn prediction possible. Some of the bank customers in due course will stop their utilization or end their subscription this could be because they switched to a competitor, no longer need the bank services, they are unhappy with their user experience, or they can no longer afford the cost.

Customer churn has become a big issue in many banks because retaining existing customers costs less when compared the the cost spent to acquire a new customer. Possible churners in a bank can be identified with the use of a customer churn prediction model and as a result the bank can take some action to prevent them from leaving. Churn rate is the key indicator for any business. Improving the customer retention rate for existing customers by just 5% can improve a company’s profitability by 25-95%.

Our case study is a bank that wants to develop a churn model to predict the probability of a customer to churn using machine learning algorithms. The ultimate goal of churn analysis is to reduce occurrence of the churn and hence increasing the profit. As a result more customers staying longer should increase the revenue and thereby increasing the profit.

1.2 Project Objectives

The main objective of this project is to create machine learning models based prediction that will classify if the customer will exit the bank or not. The other objectives of the project are as follows,

- Identify and visualize which factors contribute to customer churn
- Data preprocessing and feature engineering techniques to implement Machine learning models
- Classify bank customers as ‘churned’ and ‘retained’
- Evaluation of machine learning models by calculating F1 score of each model built
- Based on different models’ performance choose a model that will be able to attach a probability to the churn which makes it easier for the customer service team to target those customers to prevent churn
- Comparing different machine learning model performance

1.3 Project Requirements

Data requirements involves required preprocessing steps, format of data and the availability. The dataset used in our project is available on kaggle for free download in the form of csv file. There are similar kind of datasets available online that can help in analysing bank customer churn.

Functional requirements includes processing each bank customer records to find whether the customer is churned or not. The dataset collected from kaggle included the target ‘Churn’. The machine learning model built is trained using our collected dataset that enables it to learn on churned customers. The trained model can be used to predict future customers who will be exiting the bank.

Non-functional requirements are data consistency, maintain secured environment and application that is used for deployment. The google colab logs the code changes and time stamp for each piece of code enabling security and to track local changes.

Machine learning requirements include building classification algorithms that determines the target feature ‘Churn’ which is categorized into class 0 and 1 as whether the customer left the bank or not. The class ‘0’ represents that the customer is not churned and ‘1’ indicates that the customer is churned.

1.4 Project Deliverables

- Data collection methods, data processing techniques and training algorithms for data model
- Data Analytics and visualization methods to explain the dataset and the features.
- List of machine learning models selected for implementation and reasons for choosing those models
- Split train and test data for model training and prediction. Build pipeline to define the workflow of the model.
- Classify the dataset into ‘Churned’ and ‘Retained’ using prediction methods
- ML models encapsulated through Pickle with effective code structure
- Comparison of machine learning models and report the performance
- A web application built using Streamlit as part of deployment

1.5 Technology and Solution Survey

There have been many different studies covering the customer churn analysis of different organizations. Most of these works also mostly use statistical methods such as K-Nearest Neighbors, Support Vector Classifier, Decision trees and Gradient Boosting predictors. But the problem with using these simple statistical methods is that, they are greatly impacted by limitations on data such as data heterogeneity and class imbalance. We have listed the performance and advantages and disadvantages of different methods below.

Existing Solutions	Accuracy	Advantages	Disadvantages
Logistic Regression	Acc: 82.85% (Kaur & Kaur, 2020)	1. Classification after training is very fast. 2. Performs well on linearly separable data	1. Does not perform well on im balanced data 2. Models cannot be trained to understand the complex relationship between variables.
Decision Trees	Acc: 85.25% (Kaur & Kaur, 2020)	1. Computationally efficient.	1. Model suffers from bias.
KNN	Acc: 81.9% (Kaur & Kaur, 2020)	1. No Training required. 2. Increasing the sample size is very easy.	1. Easily distorted by noise. 2. Distance calculation may take a long time.
Random Forest	Acc: 85.21% (Kaur & Kaur, 2020)	1. Allows the incorporation of class weights. 2. Skew in a single tree will not affect overall performance.	1. High Complexity 2. Training time is high.
SVC	AUC: 0.8414 (Shukla, 2021)	1. Scales better to larger datasets. 2. Works well with dense and sparse data	1. Training time is very high. 2. Sensitive to noise.

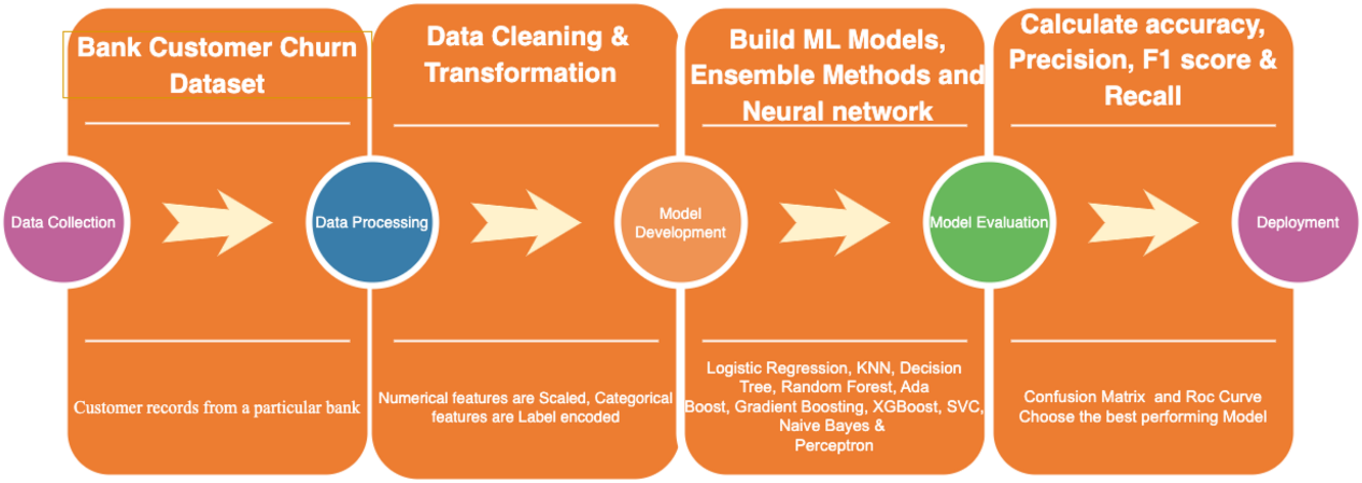
1.6 Literature Survey of existing research

There are several other works in the same area. A couple of works used Synthetic Minority Oversampling Technique (SMOTE) on ensemble methods to achieve higher accuracy. One publication by Linmao Feng(2022), uses a technique combining SMOTE, bagging and Random Forest and achieves 92.1% Area under the ROC curve (AUC) for the taken dataset.

There are also several other works that compare the baseline models on baseline features. These publications state there are no great difference in performance of these models. Most publications had to tweak the models to stack multiple baseline techniques to get better performance on the datasets they use.

1.7 Proposed Project Plan

Below flowchart shows the project plan for churn predictive model.


Bank Customer Churn Dataset
Data Cleaning & Transformation
Build ML Models, Ensemble Methods and Neural network
Calculate accuracy, Precision, F1 score & Recall
Data Collection
Data Processing
Model Development
Model Evaluation
Deployment
Customer records from a particular bank
Numerical features are Scaled, Categorical features are Label encoded
Logistic Regression, KNN, Decision Tree, Random Forest, Ada Boost, Gradient Boosting, XGBoost, SVC, Naive Bayes & Perceptron
Confusion Matrix and Roc Curve Choose the best performing Model

2 Data and Project Management Plan

2.1 Data Management Plan

The data is collected from Kaggle in the CSV format. The dataset includes 10000 instances and 14 features containing information about the card holder in a bank and their possibility of churn. The attributes in the dataset are customer_id, credit_score, surname, geography, gender, age, tenure, balance, number of products , credit_card, active_member, estimated_salary, and churn. The features of the bank customer churn dataset has been processed to predict the probability of a current customer churn based on their overall financial records. The input dataset is read into google colab to implement the executable code in python language.

Permission to access the cloud based web IDE google colab was shared to all the team members through SJSU email ID that enabled the developers to edit and execute the implementation. The data uploaded is read as a data frame through the pandas library imported in the colab environment. The colab has options to manage and retrieve the latest files used and save changes automatically without data loss. The log files can be downloaded with the information on latest edit or code changes with the username and time.

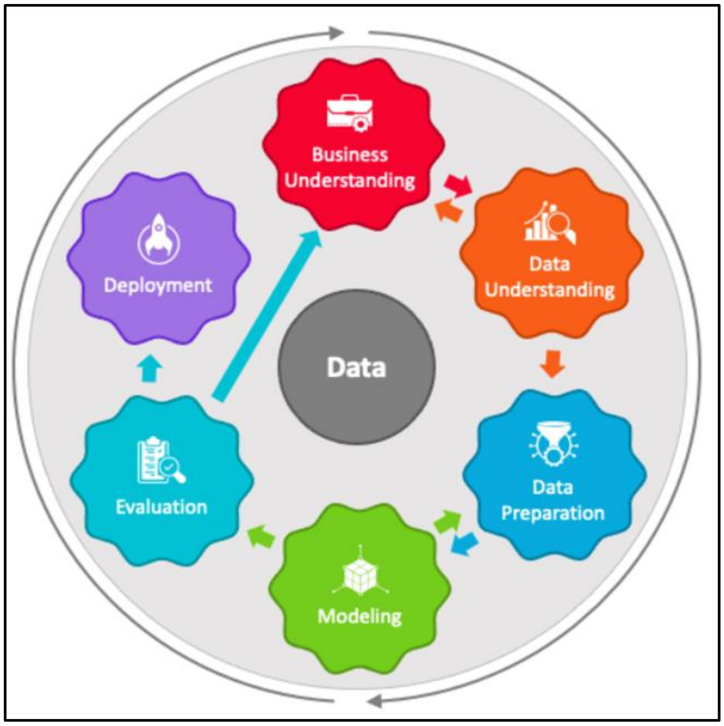
2.2 Project Development Methodology

Our team used Jira as a project and task management tool to record timeline, list of tasks, status of each task and planning the resource allocation. Each member of the team (Iqra, Janani, Saniya and Shilpa) has admin access through their SJSU email address that enabled us to edit, manage and save changes in the project plan. Resource planning and task allocation was performed in a seamless and transparent manner through the tool. The tasks are performed in an order from data collection, preparation to model development and evaluation.

There are six phases following CRISP-DM methodology which is represented as the epic for the project with the title Banking customer churn prediction. Each epic will be owned by a team member and the tasks are split following the organization plan. The below table illustrates the six epics covered as part of our project.

2.2.1 CRISP-DM

Business Understanding	Percentage of people churning and their adverse effect on the sector Factors leading to churn and customer dissatisfaction Effect of Demographics on the churn rate
Data understanding	Data exploration to understand each feature of the dataset and how it is useful and relevant for predicting the target. Understand the feature importance that supports business problem statement
Data Preparation	Clean the data and organize the data into a form that can be used for the data model Normalize the data for consistency and to achieve data efficiency
Model Development	Train the dataset using machine learning classification algorithms Import required python libraries and implement different ML models to choose the best performing algorithm on the dataset
Model Evaluation	Calculate accuracy of prediction, precision, recall and F1 score to choose the best prediction for customer churn Drawing ROC curve to show the performance at all possible classification thresholds
Deployment	Visualization of comparing model performance on tableau dashboard Select the best performing model for churn prediction and deploy a web application using Streamlit.



2.3 Project Organization Plan

The project follows CRISP-DM methodology for the developing the prediction model. JIRA board enabled project plan across the team members given the ownership of each phase in the development methodology. The tasks are divided equally between four members of the team and an expected due date was allocated to each task for time management. The status of the tasks are updated either as ‘To do’, ‘In progress’ or ‘Completed’ in a regular fashion to track the progress. The status of tasks are discussed during the check-in call scheduled on regular basis.

2.4 Project Resource Requirement and Plan

Compute: Google colab

Software: Python, Spyder and Streamlit

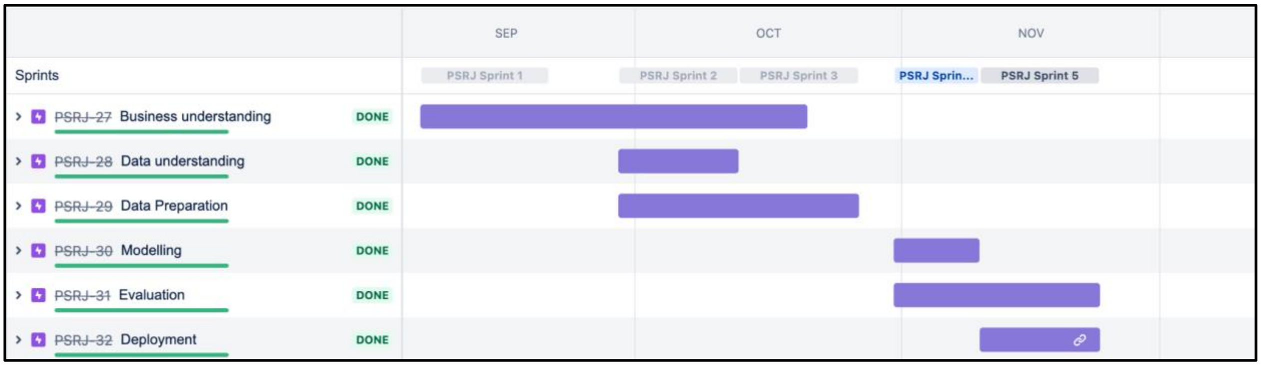
Tools: Google Docs, Google Sheets, Jira and Zoom

2.5 Project Schedule

The following figure 1 describes our project milestones and target deadline for each phase.

Figure 1

Project milestones and deadline



Sprints	
~~PSRJ-27~~ Business understanding	DONE
~~PSRJ-28~~ Data understanding	DONE
~~PSRJ-29~~ Data Preparation	DONE
~~PSRJ-30~~ Modelling	DONE
~~PSRJ-31~~ Evaluation	DONE
~~PSRJ-32~~ Deployment	DONE

Here in the dashboard the list of tasks with their corresponding status can be viewed. The tasks are divided to cover all the epics and to track the progress and flow of tasks defining if there are any blockers or dependencies.

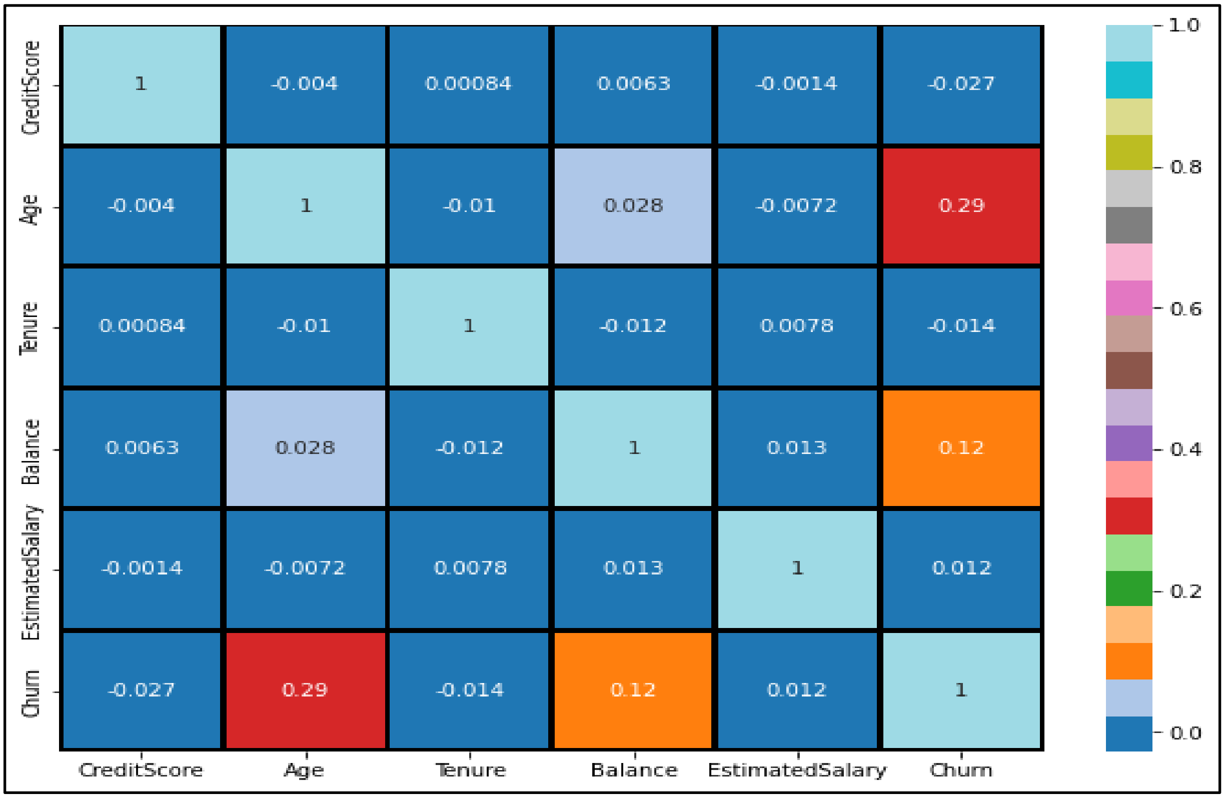
3 Data Engineering

3.1 Data Process

The Data processing phase includes end to end implementation of data collection, transformation, statistical methods to prepare the input data for modelling. It includes the conversion of raw data to machine-readable form to build the models for prediction. Our dataset is in the form a table representing rows and column of values that included categorical and continuous features. The data was plotted on a correlation matrix to figure out which descriptive features had correlation between each other and contributed towards the prediction of churn. From the below correlation matrix represented as a heatmap. The features ‘Age’ and ‘Balance’ has positive correlation with the target column. Analyzing the results of correlation, ‘Churn’ field has a mild positive trend with ‘Balance’ and ‘Age’, whereas it has very weak positive trend with ‘EstimatedSalary’. These features can be used to represent the data analytics and variation with respect to their values when determining whether the customer is churned or retained.

Figure 2

Correlation matrix

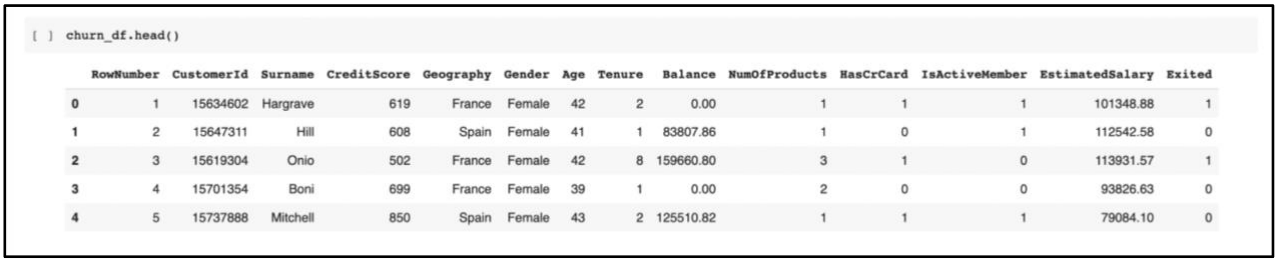


3.2 Data Collection

The dataset used in our project is a bank customer churn data found on Kaggle created by the user Adam. The dataset is formed by fetching the customer records from a particular bank which includes their personal and bank details. Each customer has unique Customer ID to represent them, geography, age and gender describing the customer information. The bank details are mentioned through the features whether the customer is an active member of the bank, owner of a credit card or not, the particular customer's credit score and the current balance in the account. The dataset contains only the authorized bank details shared by the customers and with theirs consent. These details contribute towards the prediction of a customer churn. Considering the 10,000 records of customers our ML models are trained to predict the target feature 'Churn'. The sample dataset is shown below with the descriptive features and the target column.

Figure 3

Sample dataset



```
[ ] churn_df.head()
```

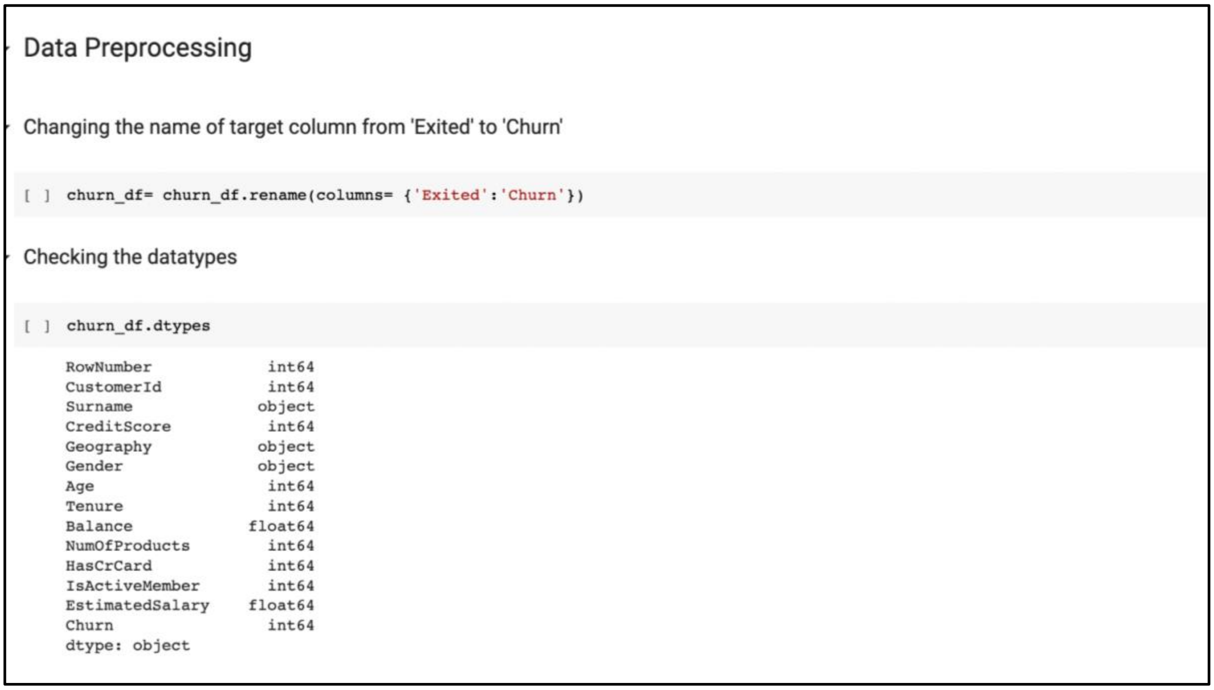
	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

3.3 Data Pre-Processing

The data collected is pre-processed to further use it for modelling and evaluation to get accurate results. First step is to identify the datatype of each feature of the dataset. Our bank customer churn data has 14 features with the categorical and continuous values. The continuous features are with the datatypes int or float and categorical are in the object form.

Figure 4

Data preprocessing steps



Data Preprocessing

Changing the name of target column from 'Exited' to 'Churn'

```
[ ]  churn_df= churn_df.rename(columns= {'Exited':'Churn'})
```

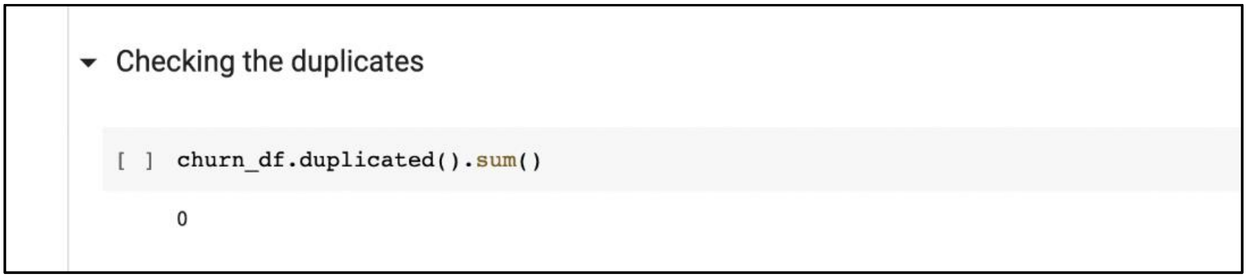
Checking the datatypes

```
[ ]  churn_df.dtypes

     RowNumber            int64
     CustomerId           int64
     Surname             object
     CreditScore          int64
     Geography           object
     Gender              object
     Age                  int64
     Tenure               int64
     Balance            float64
     NumOfProducts        int64
     HasCrCard            int64
     IsActiveMember       int64
     EstimatedSalary    float64
     Churn                int64
     dtype: object
```

The next step in preprocessing step is to find duplicates in the dataset. The dataset is checked for if it contains any duplicate records. df.duplicated() method is used to find duplicate rows in a data frame. It returns a boolean series which identifies whether a row is duplicate or unique. Our dataset had no duplicate values as the Boolean retuned is ‘0’ indicating ‘False’.

Figure 5

Duplicates check



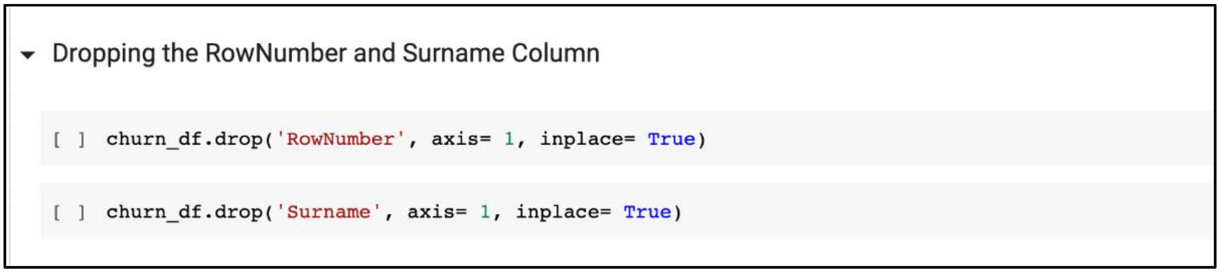
Checking the duplicates

```
[ ] churn_df.duplicated().sum()

    0
```

As a preprocessing step, unwanted columns are removed from the dataset that does not contribute towards prediction and the columns with no feature importance. The features ‘RowNumber’ and ‘Surname’ are dropped from the data frame.

Figure 6

Dropping unwanted columns



Dropping the RowNumber and Surname Column

```
[ ] churn_df.drop('RowNumber', axis= 1, inplace= True)

[ ] churn_df.drop('Surname', axis= 1, inplace= True)
```

The categorical features are defined as the data type ‘category’ for Exploratory Data Analysis to be performed o the dataset. These columns include the target feature ‘Churn’ and other descriptive features ‘'HasCrCard', ‘'IsActiveMember' and 'NumOfProducts'.

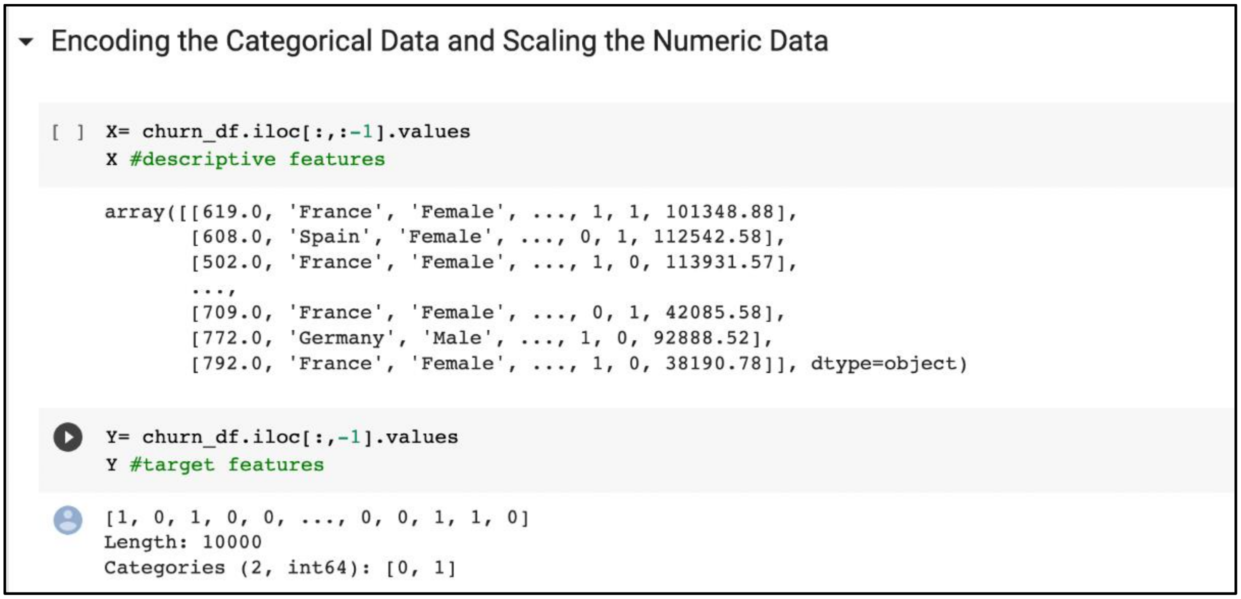
```
churn_df['Churn'] = churn_df['Churn'].astype('category')
churn_df['HasCrCard'] = churn_df['HasCrCard'].astype('category')
churn_df['IsActiveMember'] = churn_df['IsActiveMember'].astype('category')
churn_df['NumOfProducts'] = churn_df['NumOfProducts'].astype('category')
```

3.4 Data Transformation

After the data processing step, the dataset has to be transformed for the machine models to understand and to provide patterns that are easier to understand for the built models. This process involves converting non-numeric features into numeric. The categorical variables are encoded using ‘OneHotEncoder’ that produces group of bits among which the combinations of values are only those with a single high (1) bit and all the others low (0). The column transformer does OneHotEncoder by passing the column numbers of the categorical variables. We have set drop= ‘first’ in onehotencoder to avoid dummy variable trap. The total instances with 10000 are encoded during the transformation step.

Figure 7

Data transformation steps



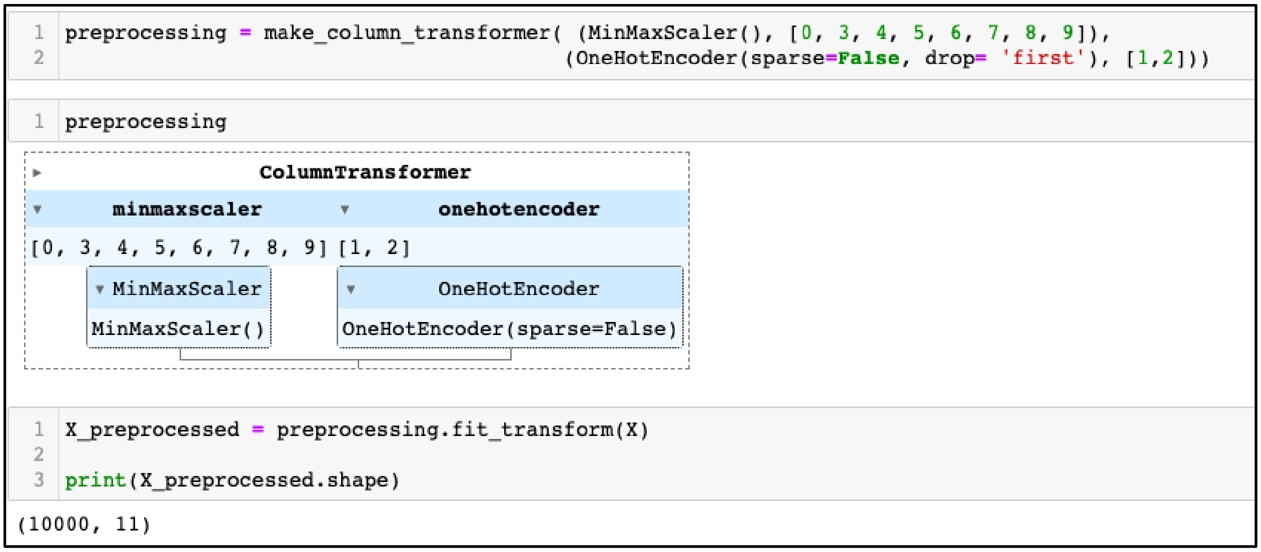
Encoding the Categorical Data and Scaling the Numeric Data

```
[ ] X= churn_df.iloc[:,:-1].values
    X #descriptive features

    array([[619.0, 'France', 'Female', ..., 1, 1, 101348.88],
           [608.0, 'Spain', 'Female', ..., 0, 1, 112542.58],
           [502.0, 'France', 'Female', ..., 1, 0, 113931.57],
           ...,
           [709.0, 'France', 'Female', ..., 0, 1, 42085.58],
           [772.0, 'Germany', 'Male', ..., 1, 0, 92888.52],
           [792.0, 'France', 'Female', ..., 1, 0, 38190.78]], dtype=object)

    Y= churn_df.iloc[:,-1].values
    Y #target features

    [1, 0, 1, 0, 0, ..., 0, 0, 1, 1, 0]
    Length: 10000
    Categories (2, int64): [0, 1]
```

Scaling of numeric data is performed through MinMaxScalar. For every feature, the minimum value of that feature gets transformed into a 0, the maximum value gets transformed into a 1, and every other value gets transformed into a decimal between 0 and 1. The numerical features are passed by their column numbers into the column transformer to scale the dataset. The shape of the transformed dataset is found as (10000, 11).

Figure 8

Preprocessed data



```
preprocessing = make_column_transformer( (MinMaxScaler(), [0, 3, 4, 5, 6, 7, 8, 9]),
                                        (OneHotEncoder(sparse=False, drop= 'first'), [1,2]))
```

```
preprocessing
```

```
▸                  ColumnTransformer
▾      minmaxscaler         ▾        onehotencoder
[0, 3, 4, 5, 6, 7, 8, 9] [1, 2]
     ▾ MinMaxScaler          ▾        OneHotEncoder
     MinMaxScaler()          OneHotEncoder(sparse=False)
```

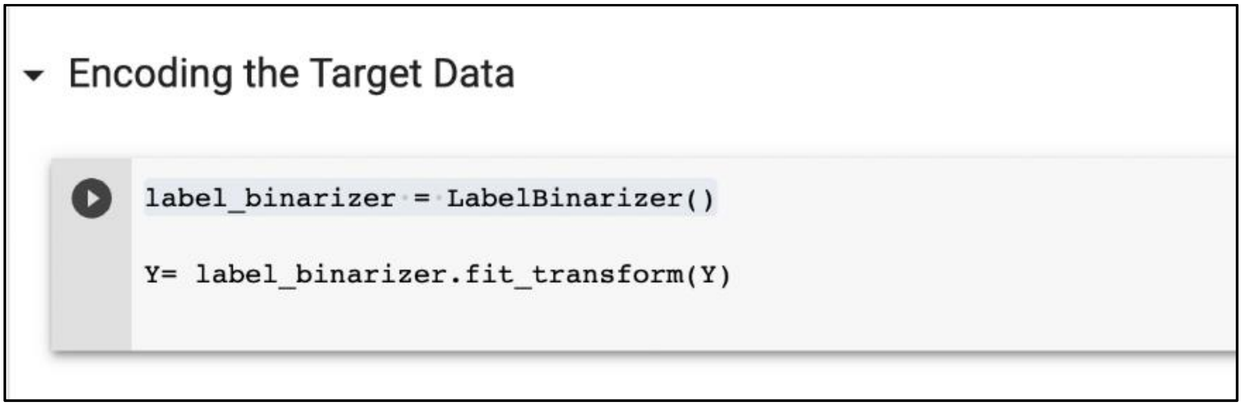
```
X_preprocessed = preprocessing.fit_transform(X)

print(X_preprocessed.shape)
```

```
(10000, 11)
```

The target variable ‘Churn’ is a category that describes whether a customer is ‘churned’ or ‘retained’. OneHotEncoder can be used to normalize labels. It can also be used to transform non-numerical labels. LabelBinarizer returns a numpy array of the target that fits the categorical values as binary encoded numerals.

Figure 9

Preprocessing target data



Encoding the Target Data

```
label_binarizer = LabelBinarizer()

Y= label_binarizer.fit_transform(Y)
```

3.5 Data Preparation

In this phase of data engineering, the cleaned, transformed data is prepared for training the model. The prepared dataset has been classified into X_train, y_train, X_test and y_test within the original dataset. Feature Selection is done to select the best predictors for the target feature which seems to add significant power in the predictability of the target feature. Smote aims to balance class distribution by randomly increasing minority class examples by replicating the dataset values.

During the preprocessing phase, it is important to balance the instances of class records for accurate results. The training data is balanced by the SMOTE algorithm to reflect the effects of the majority class on the machine learning algorithm’s efficiency and performance.

3.6 Data Statistics

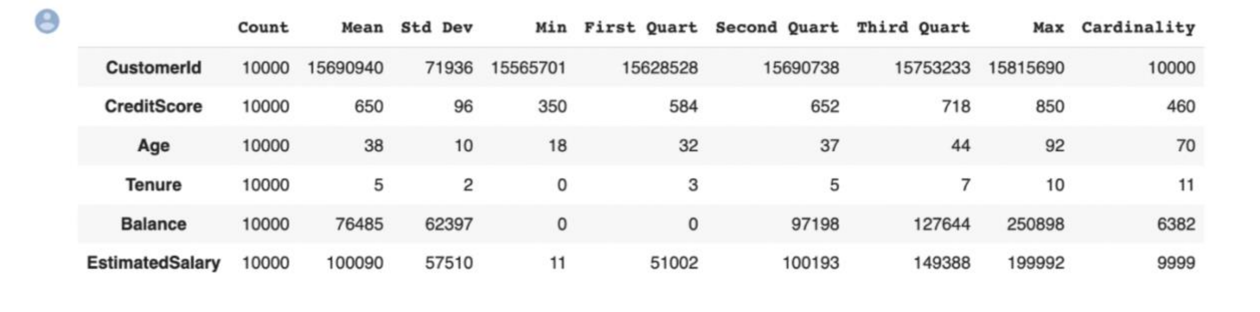
3.6.1 Data Cardinality

As part of Data statistics, the numeric features are shown with the describe() function that returns the statistical summary of the data frame or series. The count of instances of all the features are same indicating that are no missing values. The minimum, maximum and quartile values are calculated through the function.

The minimum values of ‘tenure’ and ‘balance’ are 0 indicating that the bank had customers with no balance and less tenure. The maximum credit score is ‘850’ which shows that the customer in the bank has a maximum credit score. The cardinality of numeric variables describes that the dataset has valid numeric values.

Figure 10

Describing numeric features

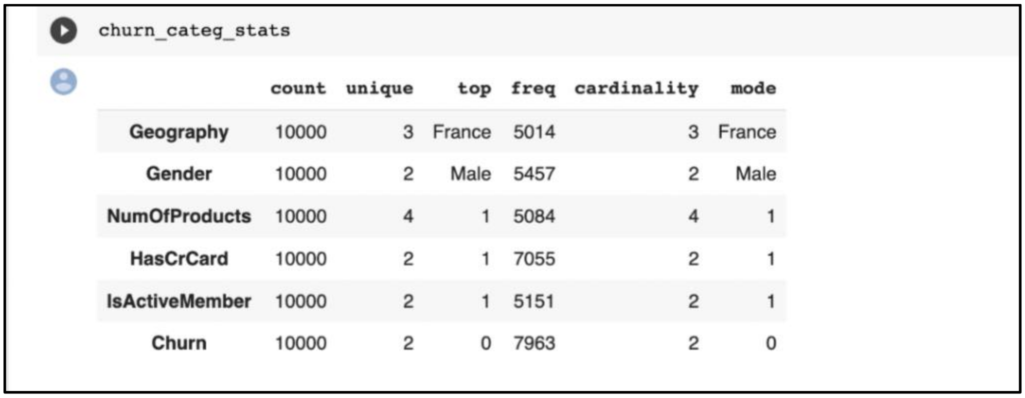


	Count	Mean	Std Dev	Min	First Quart	Second Quart	Third Quart	Max	Cardinality
CustomerId	10000	15690940	71936	15565701	15628528	15690738	15753233	15815690	10000
CreditScore	10000	650	96	350	584	652	718	850	460
Age	10000	38	10	18	32	37	44	92	70
Tenure	10000	5	2	0	3	5	7	10	11
Balance	10000	76485	62397	0	0	97198	127644	250898	6382
EstimatedSalary	10000	100090	57510	11	51002	100193	149388	199992	9999

The cardinality table for categorical features shows a similar amount of count to that in the quantitative table, therefore there are no missing values in the dataset. The categorical features include ‘Geography’, ‘Gender’, ‘NumOfProducts’, ‘HasCrCard’, ‘IsActiveMember’ and the target variable ‘Churn’. The number of unique features are described for each column. ‘Gender’, ‘HasCrCard’, ‘IsActiveMember’ contains binary values ‘0’ and ‘1’. Mode of the features are calculated as the value that is repeatedly occurring in a given dataset. The bak customer churn shows the more number of customers are Male, are active member of the bank, the person is a credit card holder ad located in France.

Figure 11

Describing categorical features



```
churn_categ_stats
```

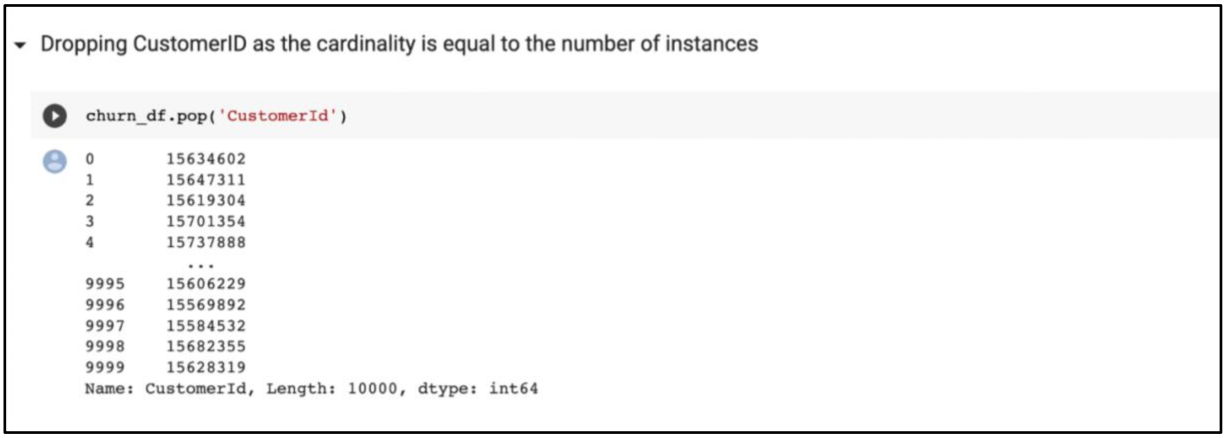
	count	unique	top	freq	cardinality	mode
Geography	10000	3	France	5014	3	France
Gender	10000	2	Male	5457	2	Male
NumOfProducts	10000	4	1	5084	4	1
HasCrCard	10000	2	1	7055	2	1
IsActiveMember	10000	2	1	5151	2	1
Churn	10000	2	0	7963	2	0

3.6.2 Data Quality Report

After checking the cardinality, the column ‘CustomerId’ has same number of rows as the number of instances. It is removed from the data frame before modelling phase. The popitem() method removes the item that was last inserted into the dictionary. Cardinality helps to understand the data and the feature importance.

Figure 12

Dropping column by checking cardinality



Dropping CustomerID as the cardinality is equal to the number of instances

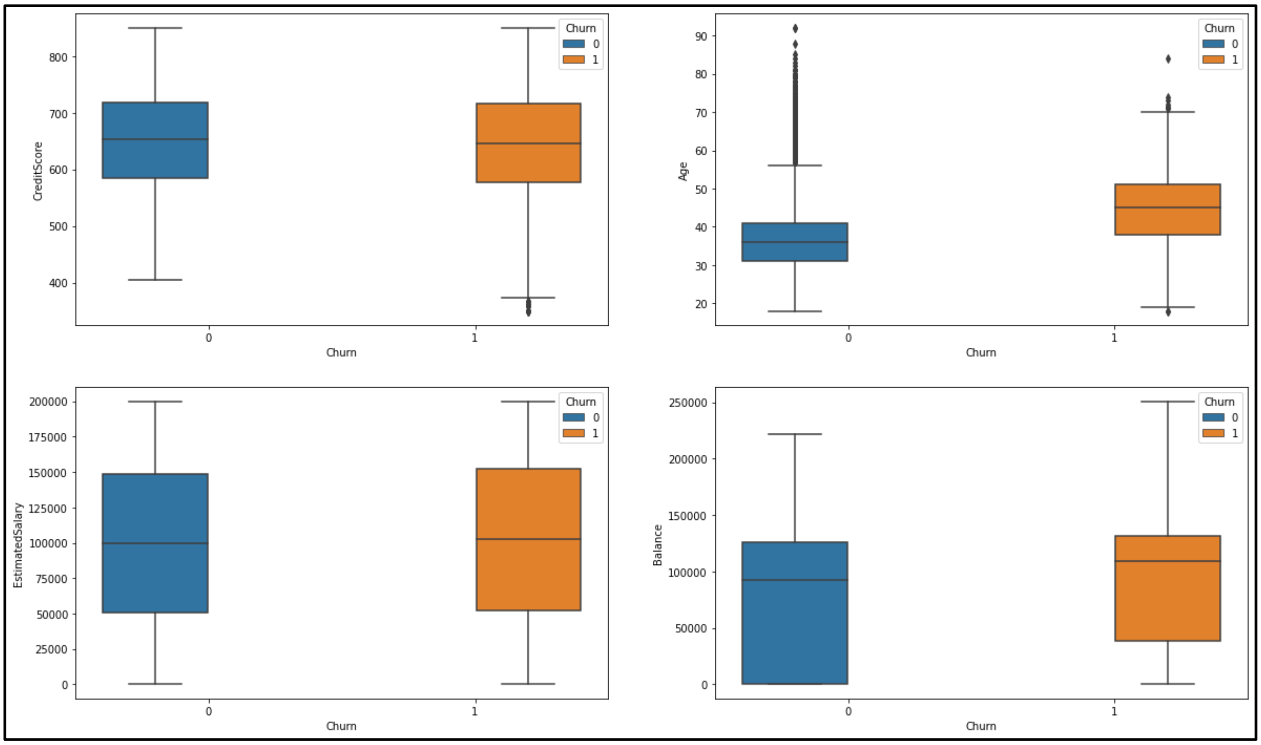
```
churn_df.pop('CustomerId')
```

```
0       15634602
1       15647311
2       15619304
3       15701354
4       15737888
          ...
9995    15606229
9996    15569892
9997    15584532
9998    15682355
9999    15628319
Name: CustomerId, Length: 10000, dtype: int64
```

The box plot representation shows range of values that the dataset lie shoeing the features with respect to Churn. Based on the graph it shows that AGE feature has outliers, data points that are significantly different from the rest of the dataset. They are often abnormal observations that skew the data distribution, and arise due to inconsistent data entry, or erroneous observations.

Figure 13

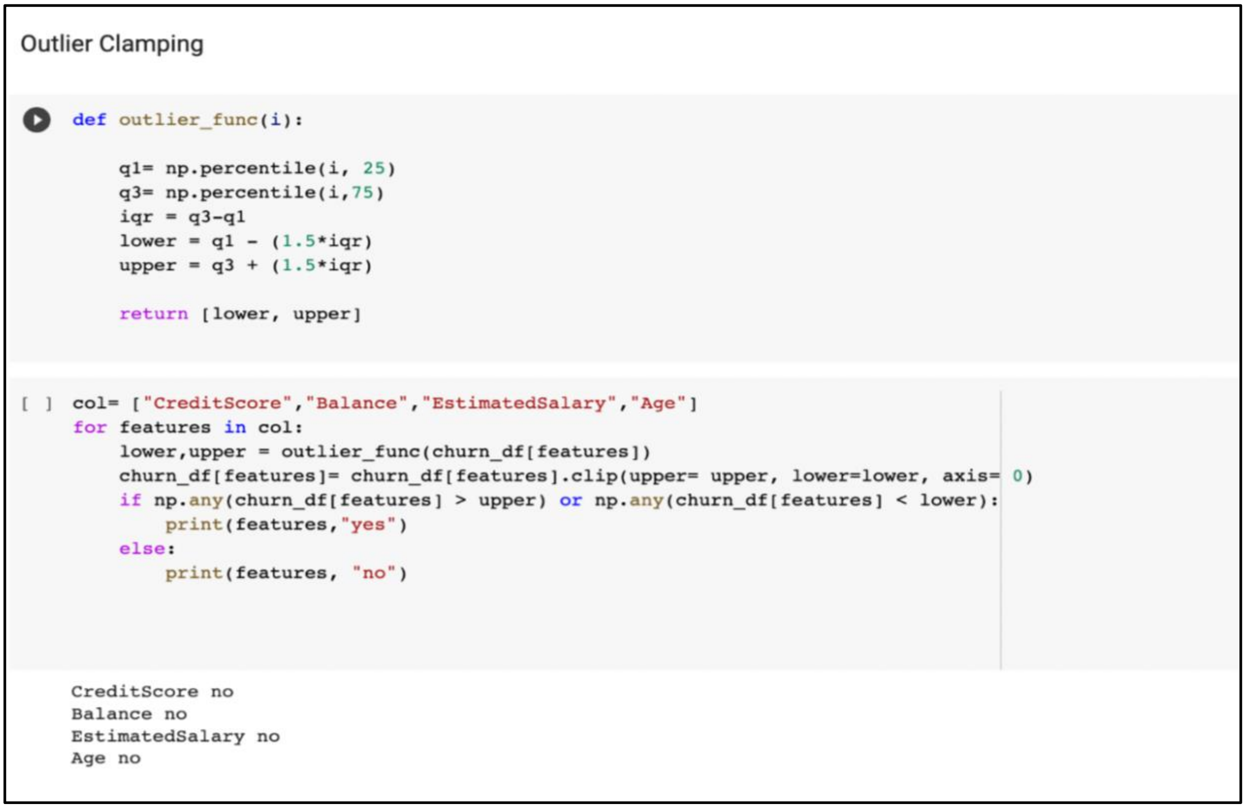
Box Plot to represent Outliers



Clamp transformation as we clamp all values above an upper threshold and below a lower threshold to these threshold values, thus capping the values of outliers. The 25th and 75th percentile to clip the values that eliminates and reduces outliers. The function created to check for outliers that returns ‘yes’ or ‘no’ after clamping of outliers.

Figure 14

Clamping Outliers



Outlier Clamping

```
def outlier_func(i):

    q1= np.percentile(i, 25)
    q3= np.percentile(i,75)
    iqr = q3-q1
    lower = q1 - (1.5*iqr)
    upper = q3 + (1.5*iqr)

    return [lower, upper]
```

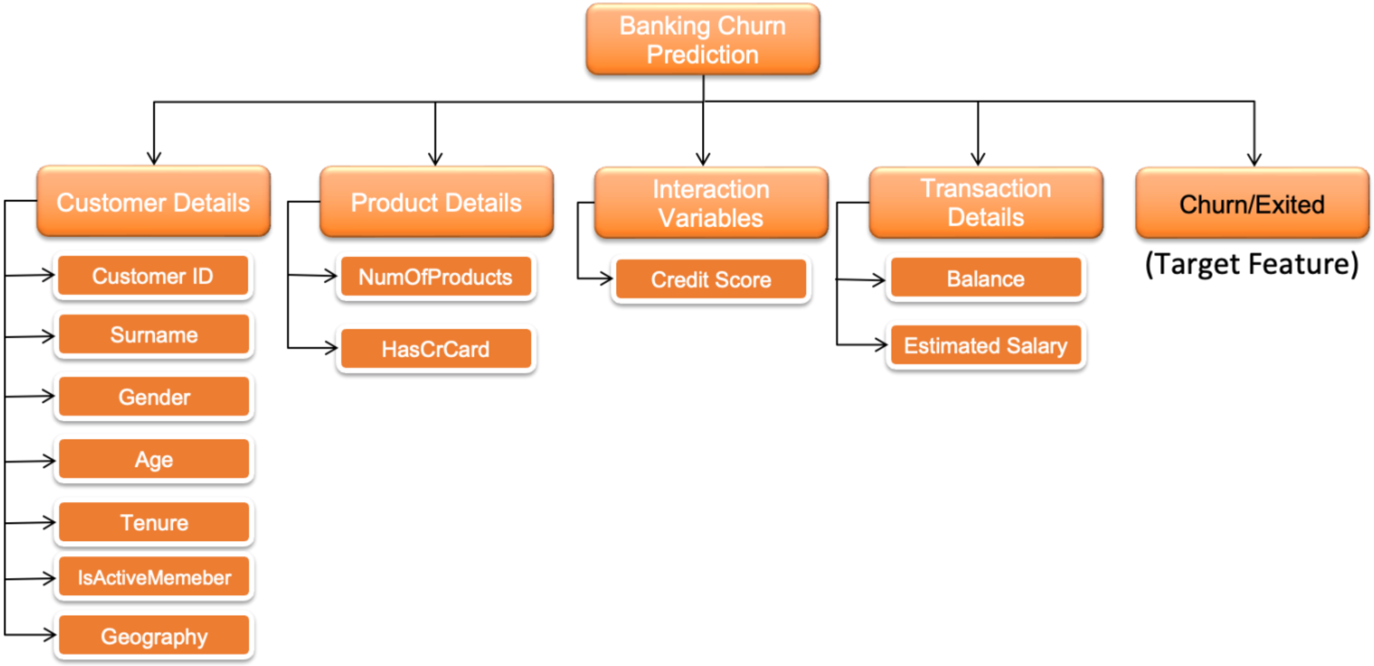
```
[ ] col= ["CreditScore","Balance","EstimatedSalary","Age"]
    for features in col:
        lower,upper = outlier_func(churn_df[features])
        churn_df[features]= churn_df[features].clip(upper= upper, lower=lower, axis= 0)
        if np.any(churn_df[features] > upper) or np.any(churn_df[features] < lower):
            print(features,"yes")
        else:
            print(features, "no")
```

```
CreditScore no
Balance no
EstimatedSalary no
Age no
```

Feature	**Data Quality Issue**	**Potential Handling Strategies**
Age	None	Valid numeric value, age should be greater than 18 for a person to have a bank account
Tenure	None	Non-negative, tenure is the period of time that customer has stayed with the bank. The minimum value is 0 and maximum as 10.
Balance	None	Non-negative, check if all the values are non-negative, if there is a negative value, then drop the row.

Estimated Salary	None	Non-negative, check if all the values are non-negative, if there is a negative value, then drop the row
Churn	None	Binary value '0' and '1' representing 'Churn' or 'Not Churn'. Prediction performed on labelled column using Binary Encoder.

3.6.2 Analytical Base Table

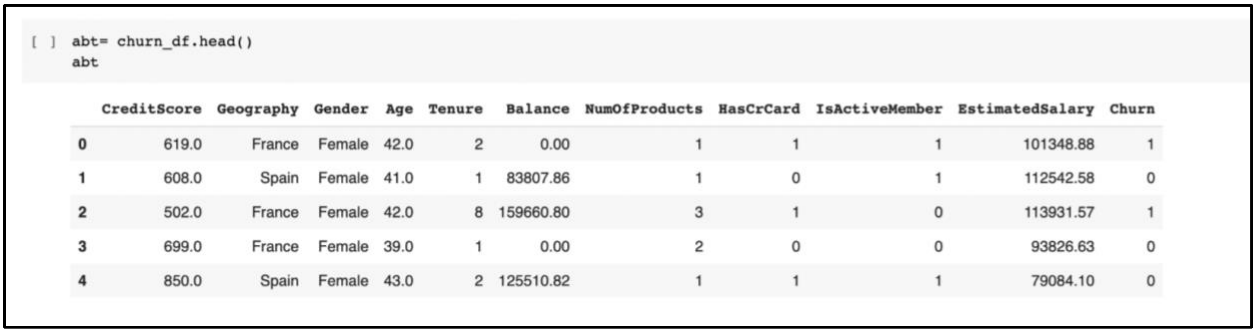


Features	Description
Age	The customer age lies between the range 18 - 75. Age is a numeric feature that influences whether the customer is churned or not. More number of customers are in the age group of 30 - 40. The customers with greater age are more likely to leave the bank.
Tenure	Tenure is the length of time an account has been a customer or active user. It is in the range 0 - 10 years describing as a numeric feature.

IsActiveMember	An active account refers to a bank account with frequent transactions. Active account status might also be associated with checking or charge accounts that have regular transactions. In this dataset, 0 represents that the customer is not an active member and 1 represents that the customer is an active member.
NumofProducts	It refers to the number of products that a customer has purchased through the bank. It is a categorical feature with the values '1', '2', '3' and '4'. More number of customers fall under the category '1' and '2'
HasCrCard	The column denotes whether or not a customer has a credit card. It is divided into two categories represented as '0' (without credit card) and '1'(credit card holder). People with a credit card are less likely to leave the bank.
Balance	Account balance is the net amount available in the bank after all deposits and credits have been balanced with any charges or debits. It includes integers ranging from 0 - 250000.
Estimated Salary	The feature represents the average salary of customers over the year. They are integers and a bank includes customers of all salary range and their account activities vary based on the salary.
Churn	Churn is the target feature of the dataset. This is a binary output given as 0 and 1, where 0 is not-churned and 1 is churned.

Figure 15

ABT Table



```
[ ] abt= churn_df.head()
    abt
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Churn
0	619.0	France	Female	42.0	2	0.00	1	1	1	101348.88	1
1	608.0	Spain	Female	41.0	1	83807.86	1	0	1	112542.58	0
2	502.0	France	Female	42.0	8	159660.80	3	1	0	113931.57	1
3	699.0	France	Female	39.0	1	0.00	2	0	0	93826.63	0
4	850.0	Spain	Female	43.0	2	125510.82	1	1	1	79084.10	0

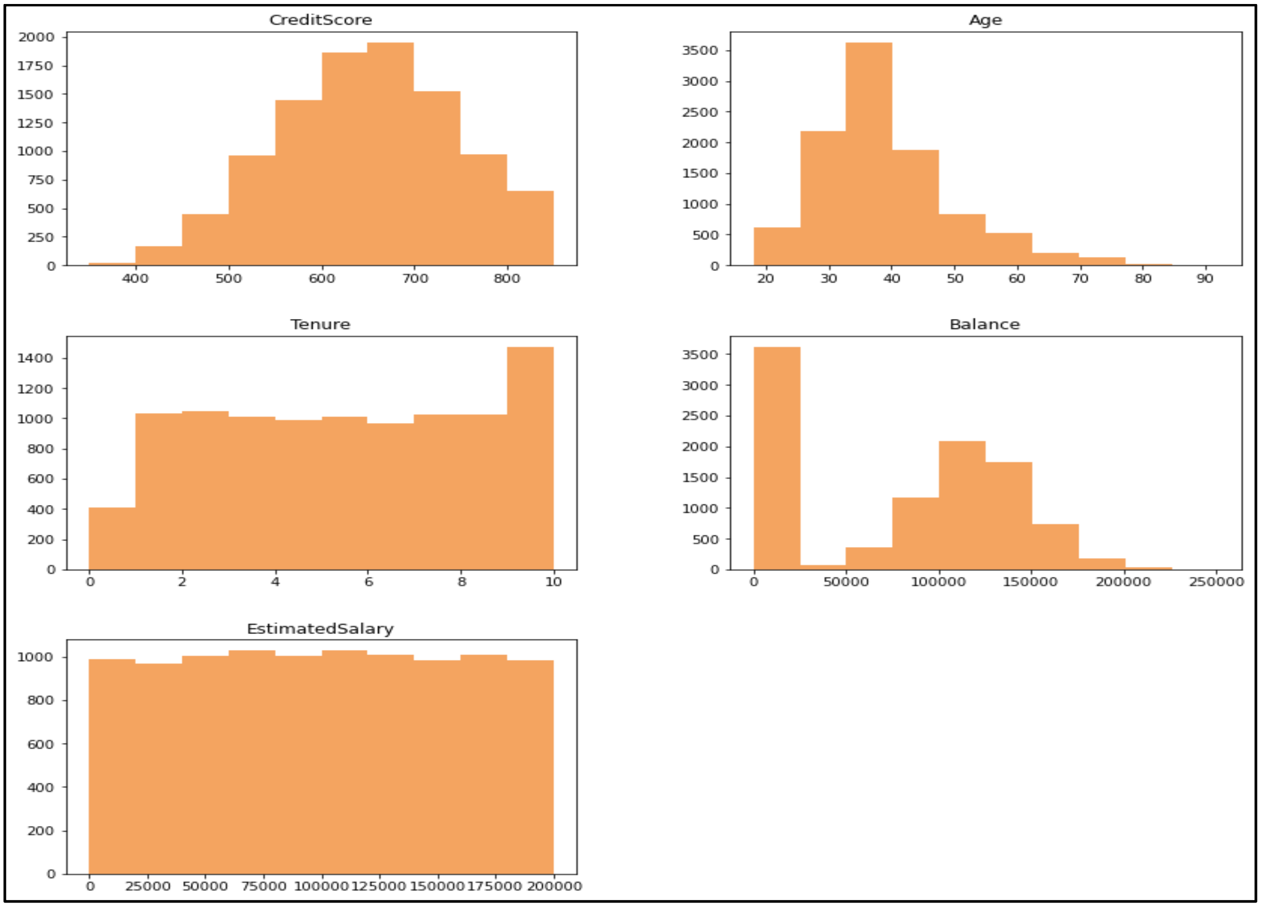
3.7 Data Analytics Results

After data processing, transformation and preparation, Data analytics phase involves the interpretation of data gathered through the use of analytical and logical reasoning to determine patterns, relationships or trends. Our project is described through various visualization graphs using the python libraries seaborn and matplotlib that provides clear idea on the descriptive and target feature relationship.

The below histogram represents the distribution followed by the numerical features of the dataset. The age distribution of the clients is fairly normal with moderate skewness. The estimated salary looks more like a uniform distribution and the credit score is also normally distributed. The distribution of all the numerical variables of the dataset are analyzed.

Figure 16

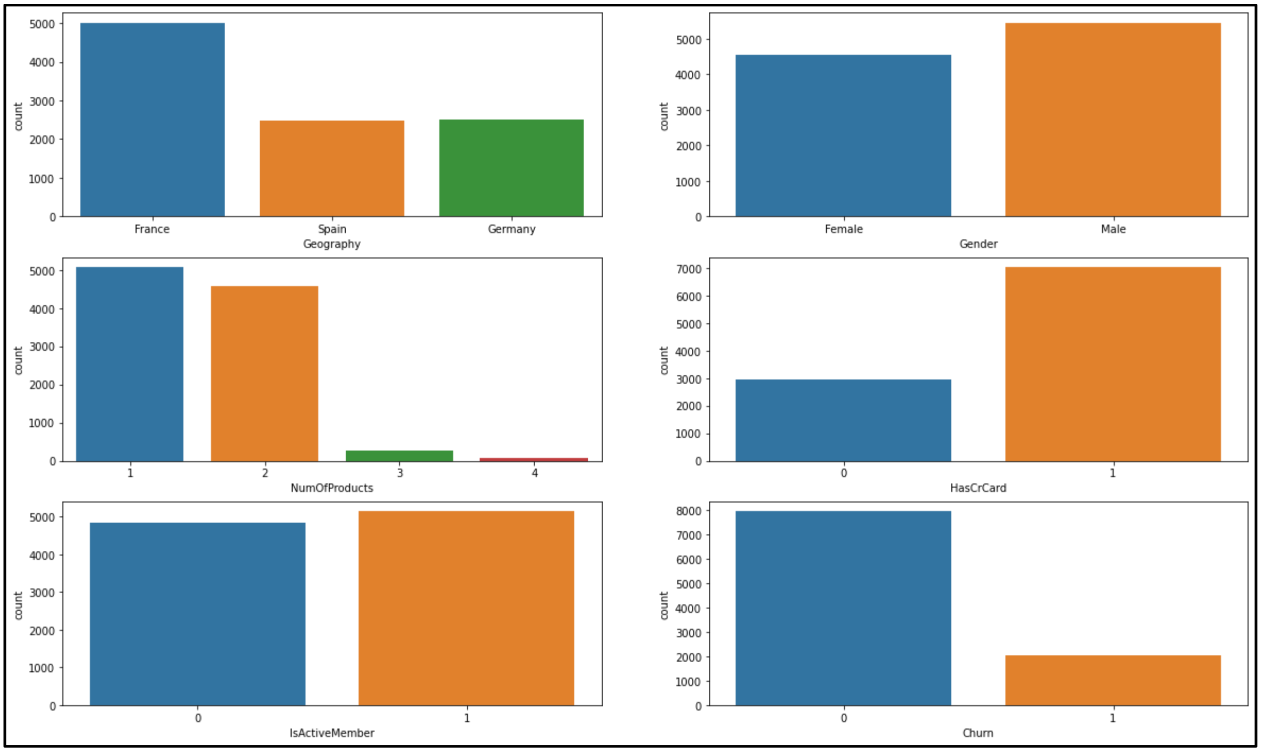
Histogram of Numeric features



The count plot represents the number of instances under each category of the bank customer churn data. Considering the 3 locations given in the dataset more number of customers in the bank are from France. Out 10000 customer records of the bank, more than 5000 customers are Male. The Number of Products are classified as 1 to 4, maximum number of customers had ‘1’ or ‘2’ products. Most of the customers are credit card holders and are active members in the bank. The records also contains not churned customers more than the churned customers.

Figure 17

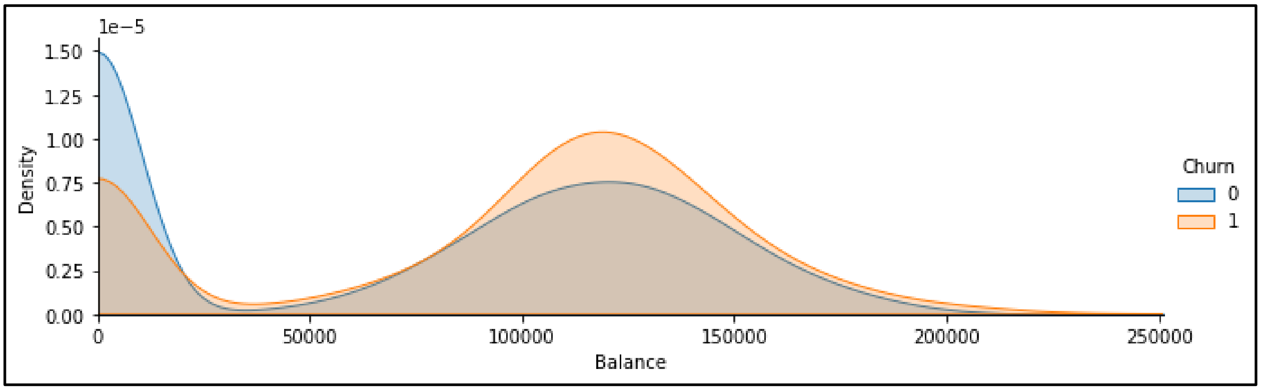
Count plot of Categorical features



The representation of a KDE plot we plotted, which is used for visualizing the Probability Density of a continuous variable, the Balance amount of the customers in the bank. The balance ranges from 0 to 250000 and there are more number of bank customers with balance from 100000 to 150000. The area under the plot represents the churn rate with balance.

Figure 18

KDE plot for Balance



The categorical feature ‘GENDER’ is analyzed to find the churn rate and the fashion in which the customer are churned. For the female customers, 25% of customers exited, while for the male customers, the customer churn rate is only 17%. Female customers are more likely to churn than male members. In our dataset, the customers include 55% of male customers and 45% of female customers.

Figure 19

Bar chart to represent count and percentage of churn

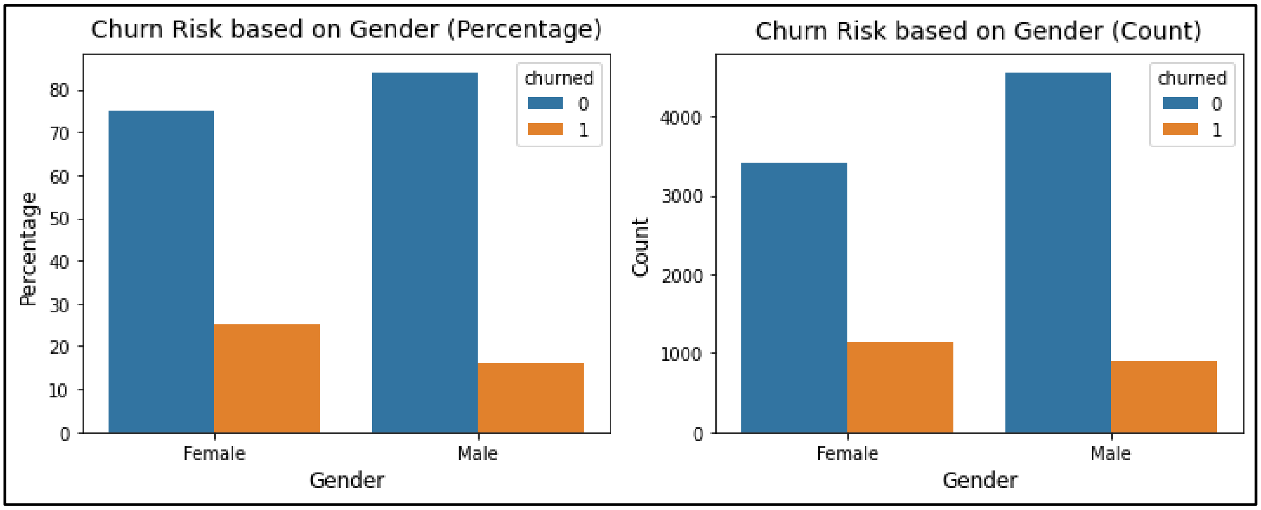
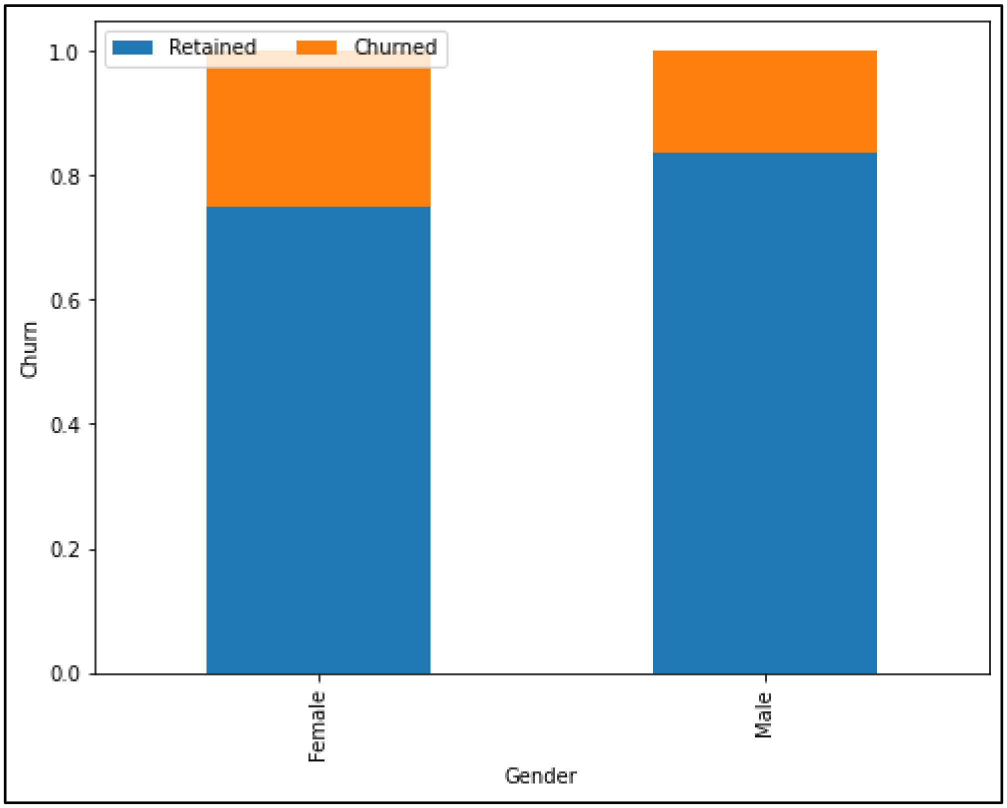


Figure 20

Churn rate w.r.t Gender



The bar graph represents the feature ‘Geography’ with the categories France, Germany and Spain with respect to churn rate. The percentage and count of churn is visualized to view the customers in different locations and if they are churned or not. German customers are two times churned more than French and Spanish customers. 84% customers in France are not churned. 68% of customers from Germany retained and 83% in Spain did not churn.

Figure 21

Bar chart to represent count and percentage of churn

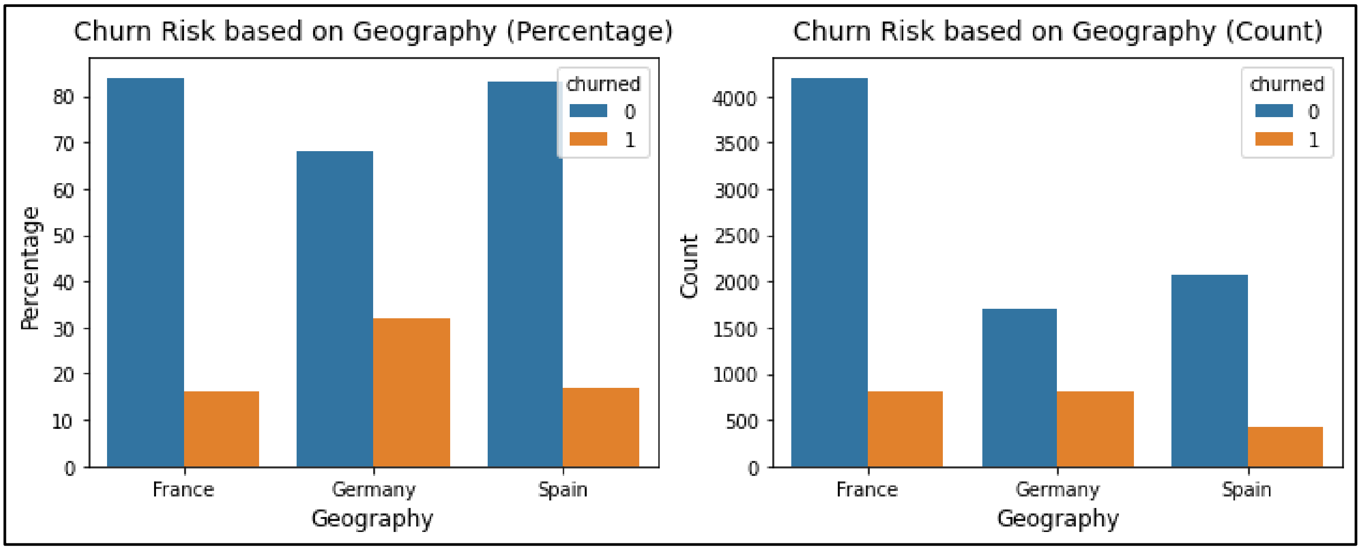
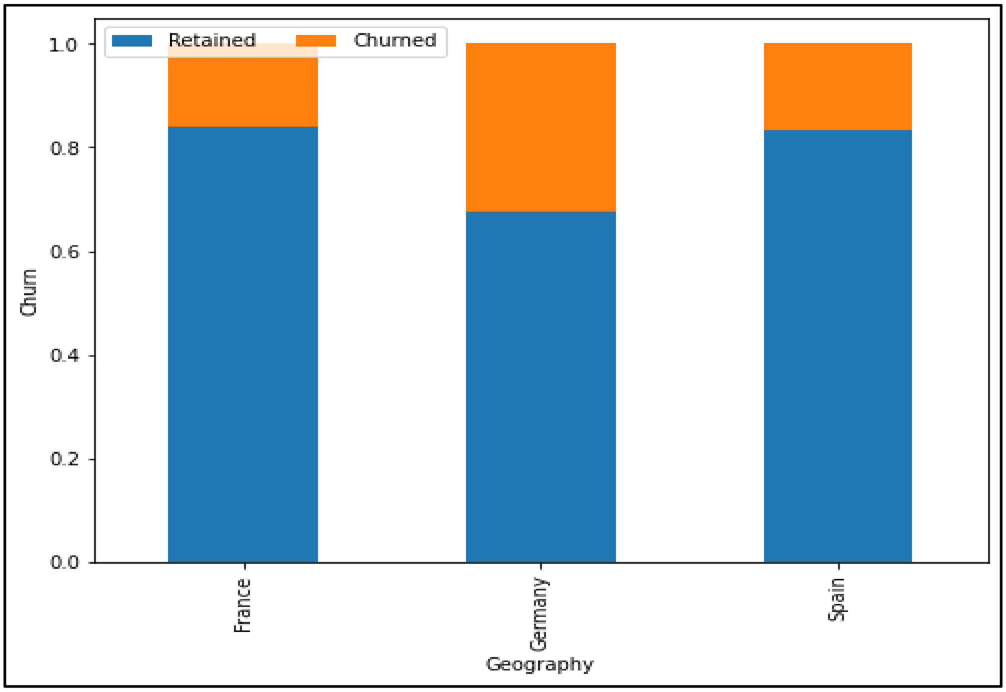


Figure 22

Churn rate w.r.t Geography



Visualizing the ‘IsActiveMember’ feature it is found that the active members, the churn rate is 14.3%, while for the inactive members, the churn rate is 26.9 %. We can find that the inactive members are more likely to churn.

Figure 23

Percentage of churn w.r.t Active member

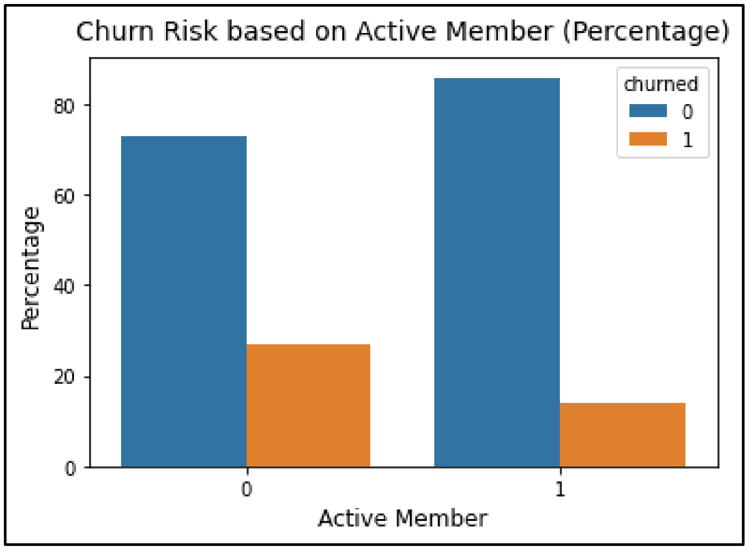
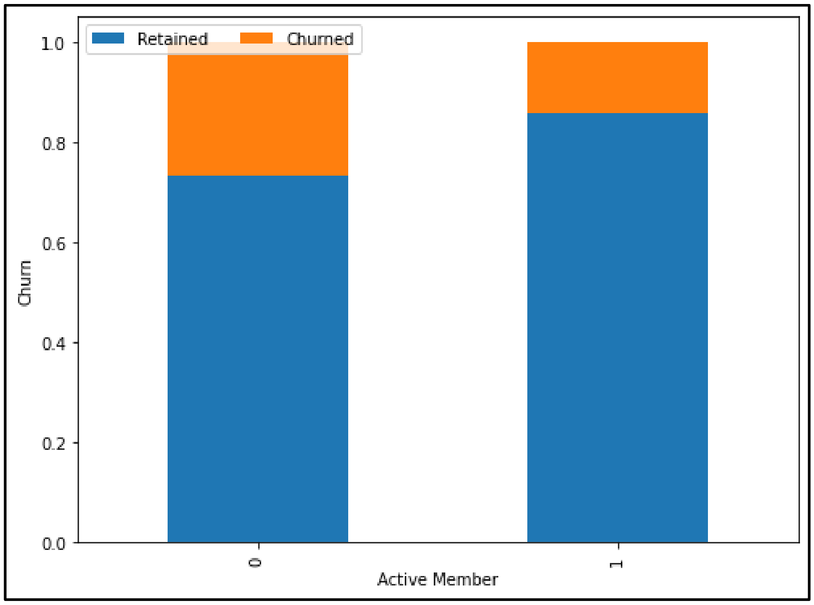


Figure 24

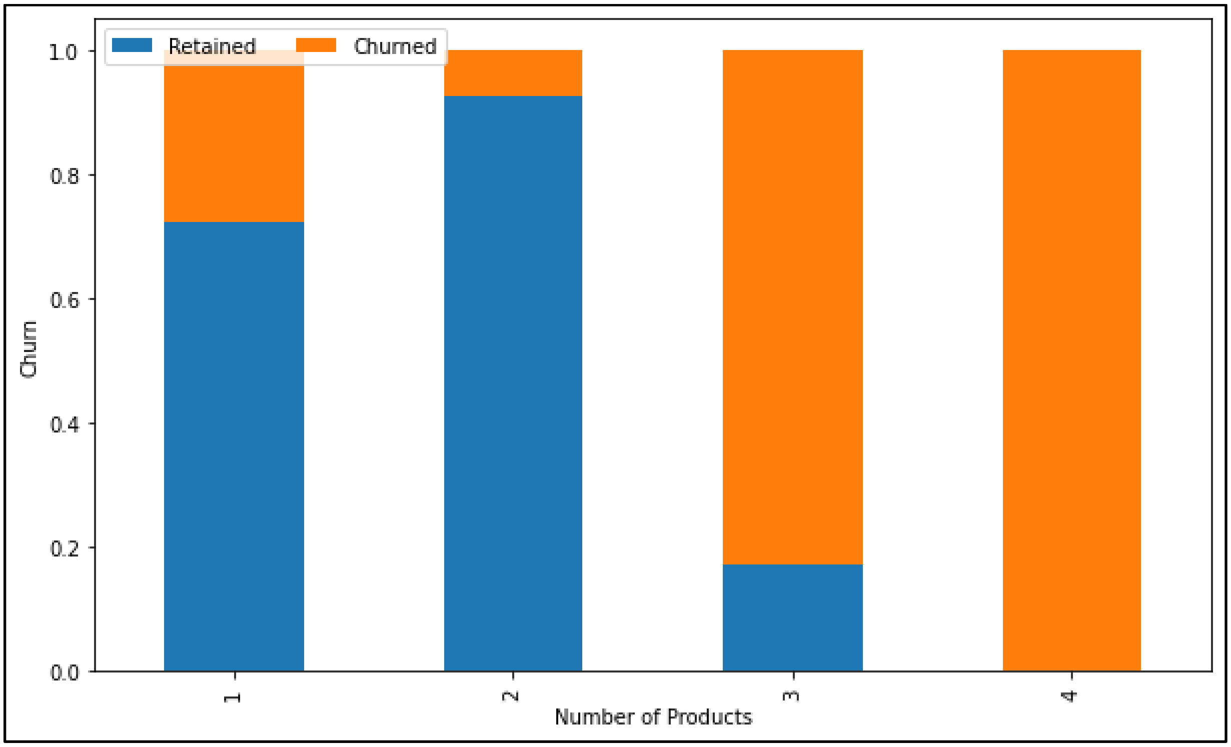
Churn rate w.r.t Active member



From the visualization, it is observed that customers who buy more than 2 products have a high rate of loss. All of the customers with a count of 60 people, who bought 4 products left the bank. Customers having a number of products of 4 and 3 should be closely screened because the churn rate is 100% and 82% for these two indicators of products.

Figure 25

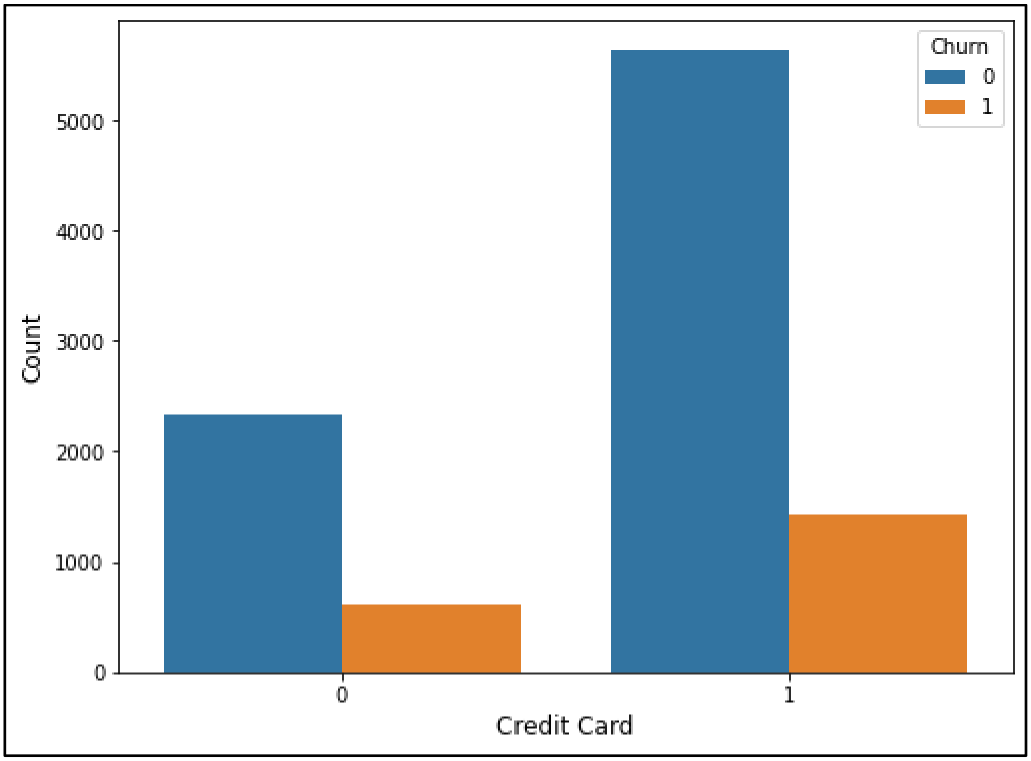
Churn rate w.r.t Number of products



The customers with Credit card has a greater churn rate compared to customers without credit card. The ‘HasCrCard’ feature describes person with credit card are more likely to exit the bank.

Figure 26

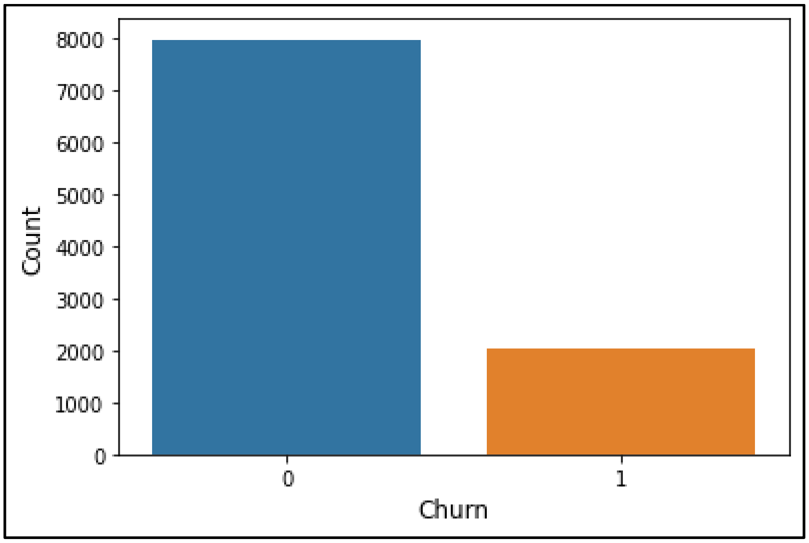
Count of churn w.r.t Credit card



From the graph, the number of customers that exited the bank is lower compared to the number of customers that didn’t leave the bank. Out 10000 records, around 8000 customers are not churned and about 2000 customers are likely to leave the bank. This indicates that there is a class imbalance between the target feature. As this represents the bank customer data, we do not drop any records and make sure our model is trained on an imbalance dataset.

Figure 27

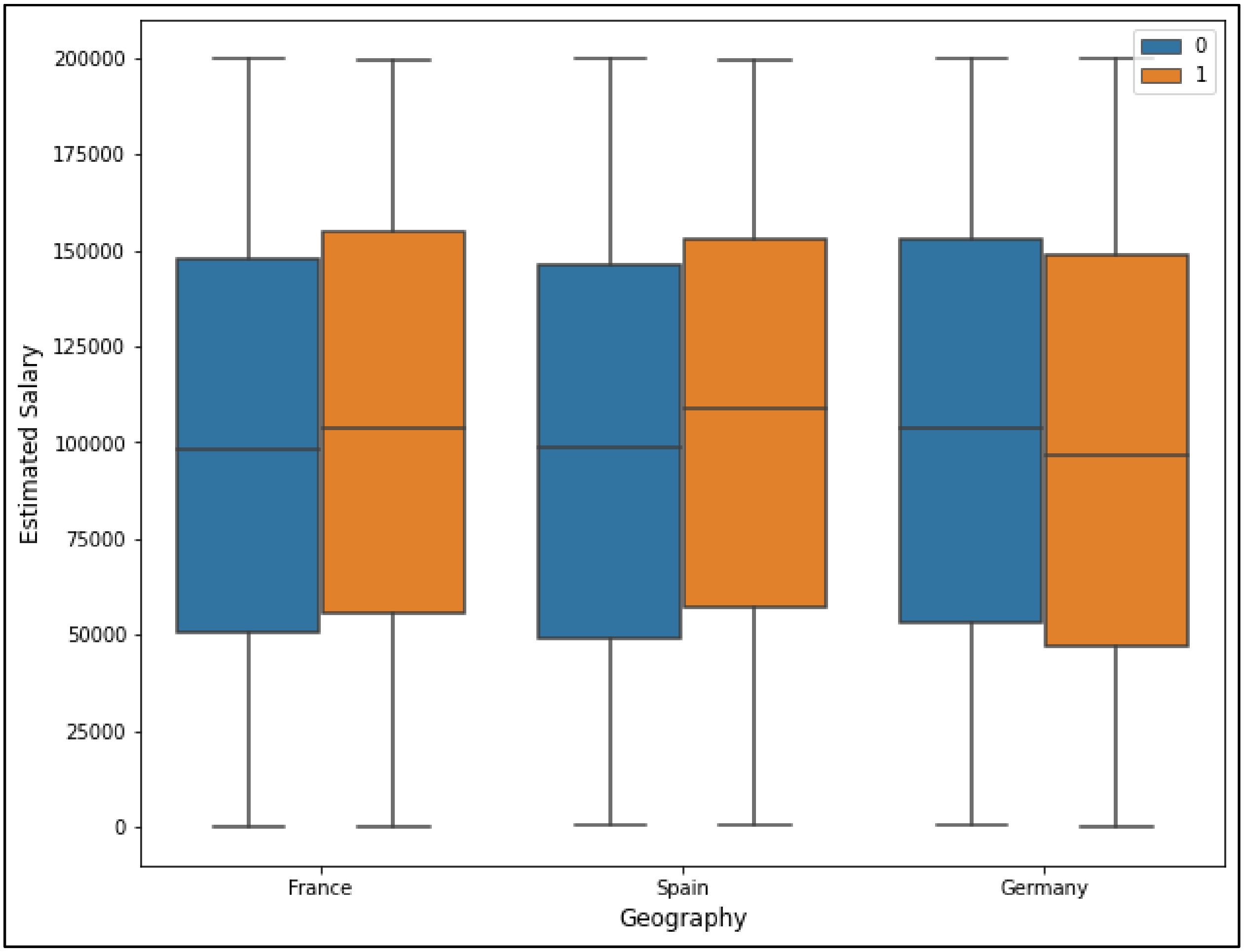
Count of churn and retain



The Estimated Salary is visualized against Geography with respect to the target feature ‘churn’. The minimum and maximum salary is estimated from the range 50000 to 150000. The box plot representation shows the ‘churned’ and ‘retained’ on estimated salary.

Figure 28

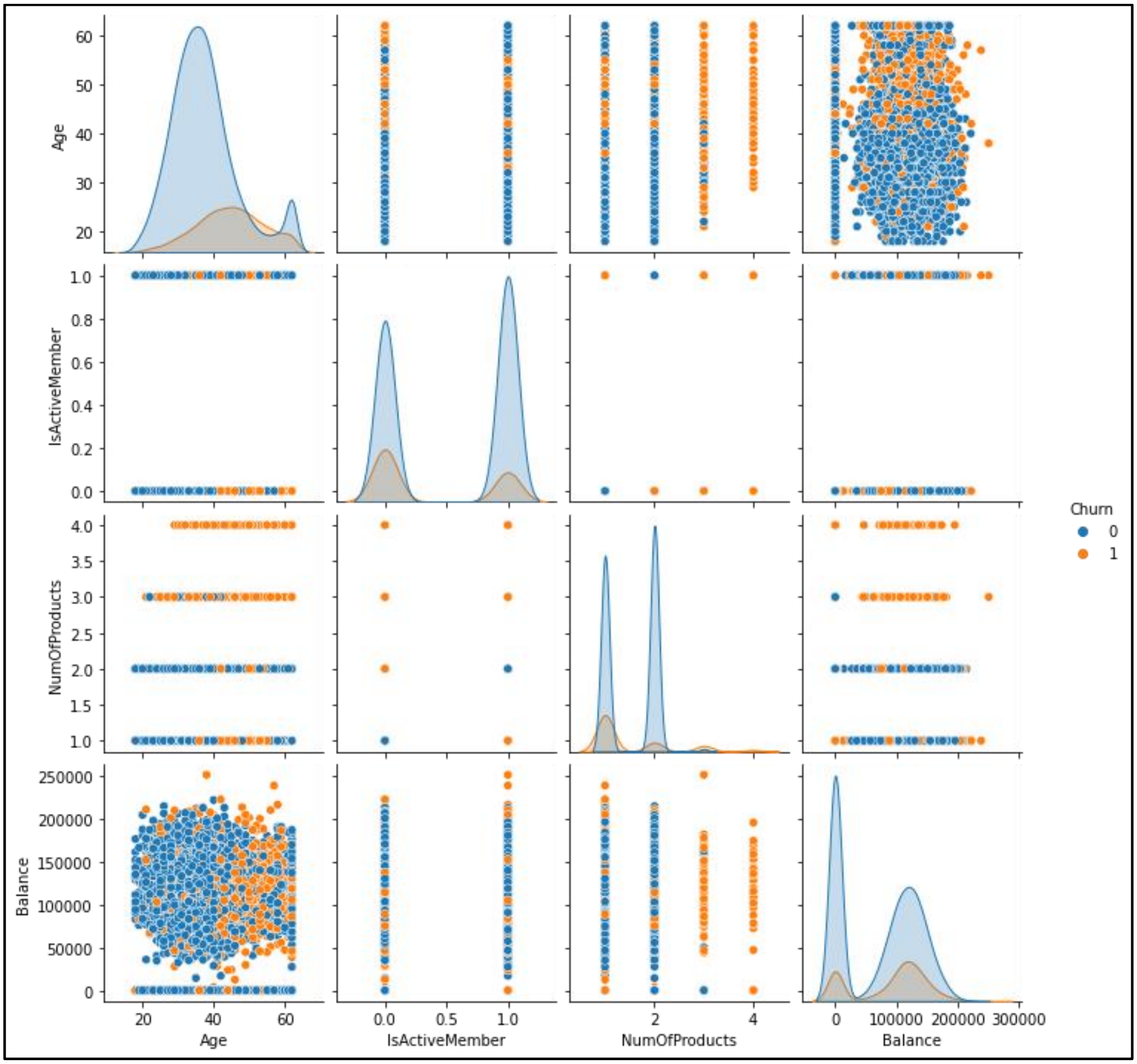
Churn rate w.r.t Geography and Estimated salary



The relationship between the variables ‘Age’, ‘IsActiveMember’, ‘NumOfProducts’ and ‘Balance’. This Pair plot is used to understand the best set of features to explain a relationship between the variables. The hue with respect to the target feature ‘churn’ is differentiated through colors. It is shown that the independent variables are not correlated with each other, which is good for the modeling. However, the strongest correlations are observed for the “age” and “balance” variables with the target.

Figure 29

Reduced Pair plot



Through these different analyses, the bank will be able to predict the savings behaviors of customers and identify which type of customer is most likely to make term deposits on the one hand or to predict future account closures to carry out the necessary actions. Out of the three countries whose data was collected, Germany had the maximum churn rate compared to France and Spain. With respect to Gender, we can see that overall Females have a greater churn capacity than Males. Also people who have a credit card have a greater churn rate than the people who don’t have a credit card. The members who are active stay with the bank and don’t churn as much as compared to those members who are inactive.

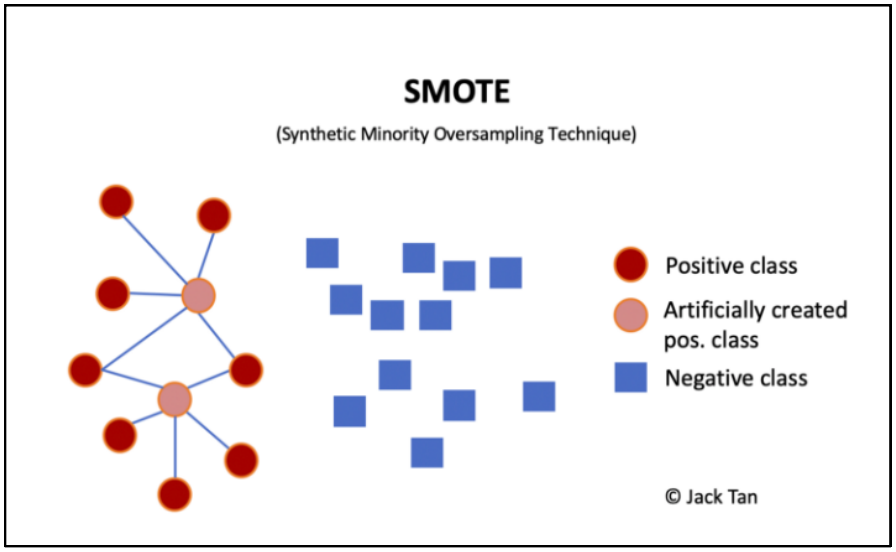
4 Modelling

4.1 Over Sampling

As there is often class imbalance in churn prediction model, we have created a imblearn pipeline to increase the minority class count by replicating them. This method is known as SMOTE (Synthetic Minority Oversampling Technique). Processing the imbalance class is important before training the model so that algorithm is not biased towards any class. Figure 30 below shows oversampling technique.

Figure 30

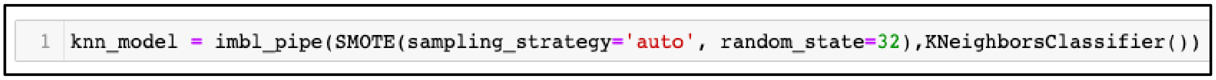
Oversampling


SMOTE
(Synthetic Minority Oversampling Technique)
Positive class
Artificially created pos. class
Negative class
© Jack Tan

To oversample the minority data, we have created a pipeline using imblearn.pipeline as shown in figure. By creating this pipeline, the data will first get oversampled then it will be trained to predict the target class.

Figure 31

Imblearn pipeline to oversample the data



```
knn_model = imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),KNeighborsClassifier())
```

4.2 Model Comparison along with Cross- Validation and Hyper-parameter Tuning

To predict the likelihood that employee will churn, we have implemented multiple supervised learning techniques to compare the performance of each models and select the best model based on accuracy and f1_Score.

We have also performed hyperparamter using GridSearchCV tuning on each model to select the best parameters for the model. GridsearchCV checks for all possible combinations of parameters. In order to get the unbiased results, we also defined cross validation strategy i.e. 5-fold cross validation during hyperparameter tuning. Cross- validation is a method in which model is trained using subset of the data set and then evaluate using the complementary subset of the dataset. By setting cv= 5, the data will split into five parts and one part will be the test data and rest part will be trained data and data will be randomly shuffled in each iteration. Figure 32 below shows the 5 Fold Cross Validation technique.

Figure 32

5- Fold Cross Validation

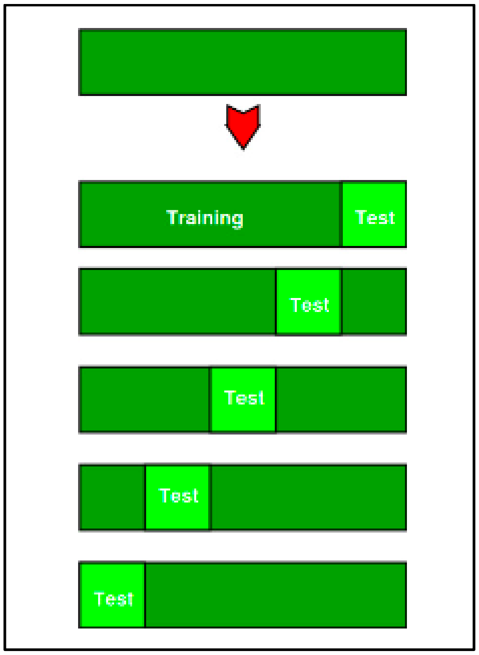
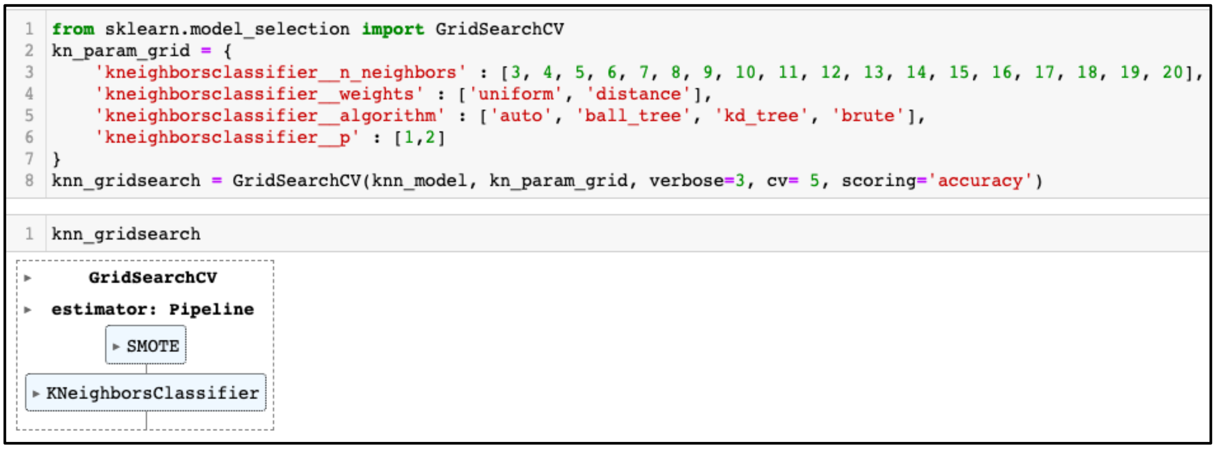


Figure 33 below showing hyper-parameter tuning using GridsearchCV with 5 fold cross validation.

Figure 33

Hyper-Parameter Tuning Using GridSearchCV



```
from sklearn.model_selection import GridSearchCV
kn_param_grid = {
    'kneighborsclassifier__n_neighbors' : [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20],
    'kneighborsclassifier__weights' : ['uniform', 'distance'],
    'kneighborsclassifier__algorithm' : ['auto', 'ball_tree', 'kd_tree', 'brute'],
    'kneighborsclassifier__p' : [1,2]
}
knn_gridsearch = GridSearchCV(knn_model, kn_param_grid, verbose=3, cv= 5, scoring='accuracy')
```

```
knn_gridsearch
```

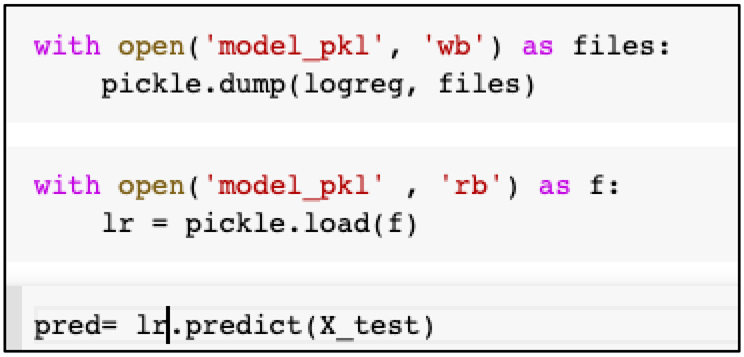
```
▸ GridSearchCV
▸ estimator: Pipeline
    ▸ SMOTE
▸ KNeighborsClassifier
```

4.3 Saving the Models

As machine learning models often take hours to run especially when the dataset is huge with many features or algorithm is lazy learner such as KNN. Also, if machine power goes off, we need to train the model again from scratch and this will take time. Hence, to avoid this we are using python Pickle module to save models. Pickle module is used to serialized and deserialized model when needed. This helps in minimizing the lengthy re-training time and allows to share, commit and re-load machine learning models during evaluation phase. Figure 34 below shows saving and loading the model.

Figure 34

Saving model using python pickle



```
with open('model_pkl', 'wb') as files:
    pickle.dump(logreg, files)
```

```
with open('model_pkl' , 'rb') as f:
    lr = pickle.load(f)
```

```
pred= lr.predict(X_test)
```

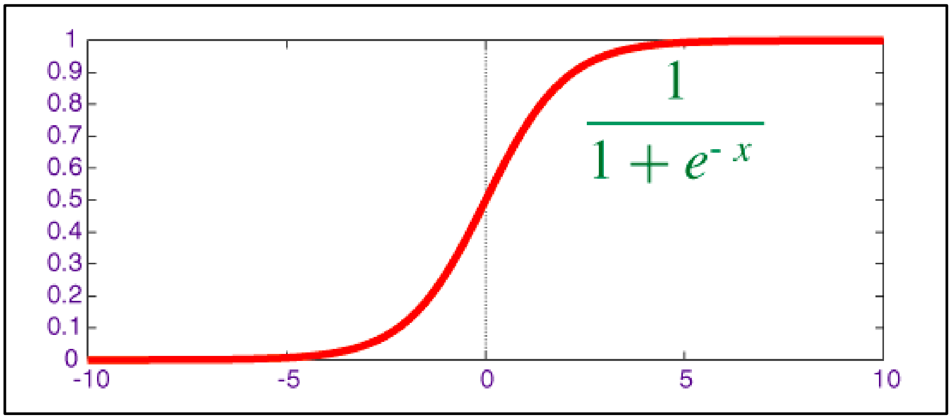
4.4 Logistic Regression

Logistic Regression is one of generalized linear models. It is used in binary classification where the target variable may take class 0 or 1. The algorithm builds the model on the basis of maximum likelihood estimation which is a sigmoid function. The formula to calculate probability for each data point is given below:

$$\boxed{g(z) = \frac{1}{1+e^{-z}}}$$

Figure 35

Sigmoid function for logistic regression



Here z represent the logit function of the target variable. Logistic Regression is based on following assumptions:

1. Linear relationship between logit of target variable and response variable
2. There should not be any multi-collinearity in the data
3. Target variable should be discrete

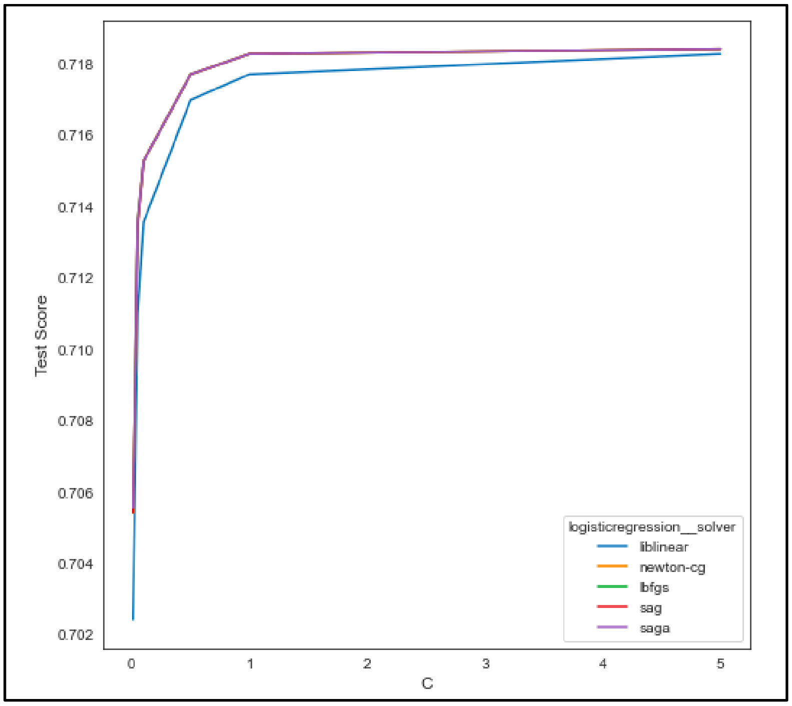
In this model, we have performed hyper-parameter tuning on following parameters:

1. Solver: This is used to optimized the algorithm by L1 or L2 regularization. Some of the solvers are 'liblinear', 'newton-cg', 'lbfgs', 'sag', 'saga'.
2. C: It is used to define regularization parameter. The smaller the value, smaller values specify stronger regularization.

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 36 shows the model performance for different values of C and Solver.

Figure 36

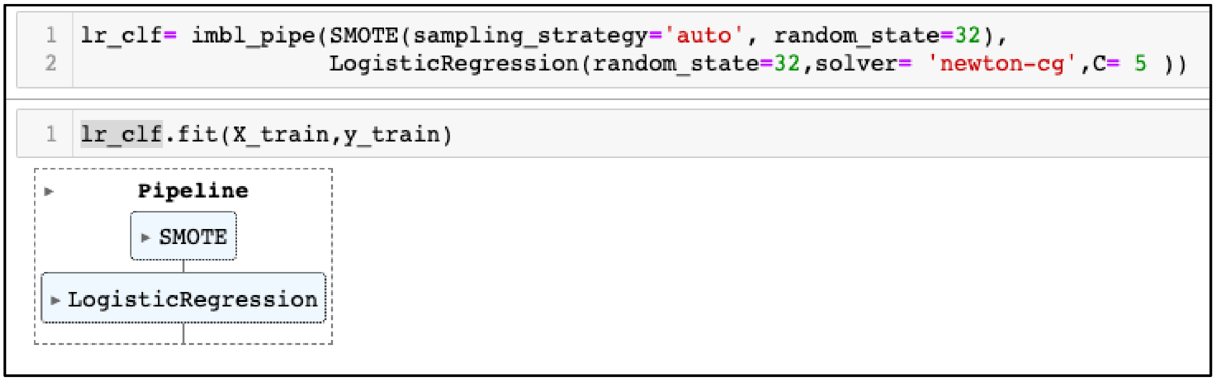
Accuracy score for different values of regularization parameter



After getting the best parameters, the model is trained as shown in figure 37 below.

Figure 37

Training the model with best parameter



```
lr_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),
                  LogisticRegression(random_state=32,solver= 'newton-cg',C= 5 ))
```

```
lr_clf.fit(X_train,y_train)
```

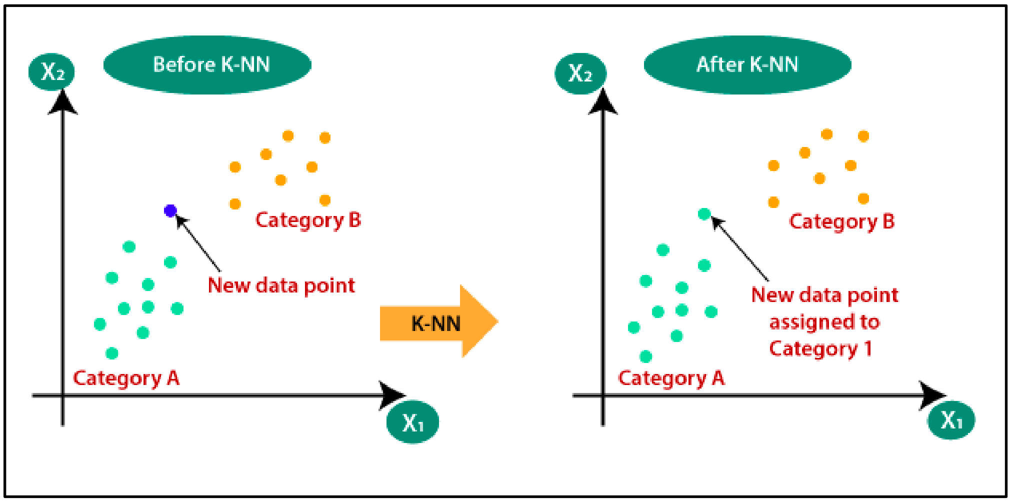
4.5 KNeighbors Classifier

KNeighbors classifier is lazy learner i.e. it doesn't learn discriminative function from training data but memorizes the training dataset and waits for the training data to classify the target variable. It is one of the simple supervised learning and is interpretable. It uses feature similarity to predict the class of query which is based on how closely the query matches the data points in the training dataset. Similarity is determined by calculating the distance between datapoints and query. Euclidean distance is the common distance metric used. However, if the training data set is large then Manhattan distance is preferred as it will less computationally expensive. Results of the KNN depends on following factors:

1. Distance metric
2. Distance rule
3. Value of K i.e. k neighbors considered.

Figure 38

KNN algorithm



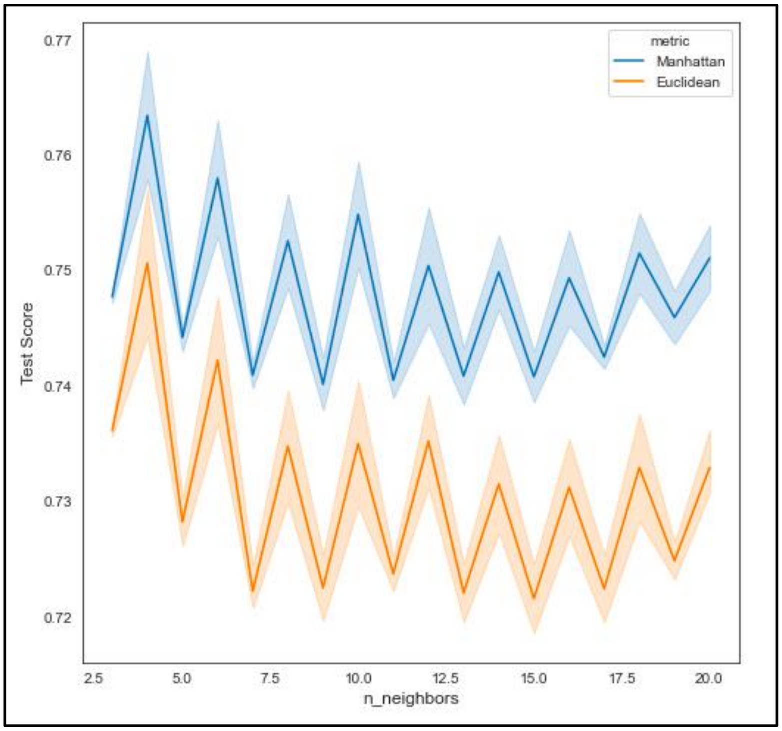
In our model, we have performed hyper-parameter tuning on following parameters:

1. n_neighbors: This value determines the number of neighbors to considered in determining the target variable class. Values are 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20.
2. Weights: If the weight is set to 'uniform' then all points are weighted equally and if the weight is set to 'distance' then weight points by inverse of their distance.
3. Algorithm: 'auto', 'ball_tree', 'kd_tree', 'brute'
4. p: This is the power parameter for the Minkowski metric. If p = 1, then manhattan distance is used and if p=2 then euclidean distance is used.

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 39 shows the model performance for different values of N.

Figure 39

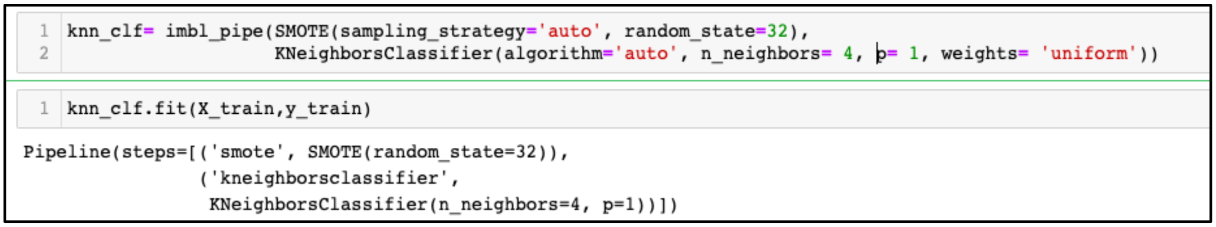
Accuracy score for different values of k



As shown in figure 40 below, we have set n=4, weight= uniform and distance metric= Manhattan. In this model, the algorithm calculated the Manhattan distance between query and each data point. For n=4, 4 nearest neighbours was selected and the majority class in these neigbours predicted the target variable class. If the majority class is churn the then target variable will churn else he/she will stay in the organization.

Figure 40

Training the model with best parameters



```
knn_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),
                    KNeighborsClassifier(algorithm='auto', n_neighbors= 4, p= 1, weights= 'uniform'))
```

```
knn_clf.fit(X_train,y_train)
```

```
Pipeline(steps=[('smote', SMOTE(random_state=32)),
                ('kneighborsclassifier',
                 KNeighborsClassifier(n_neighbors=4, p=1))])
```

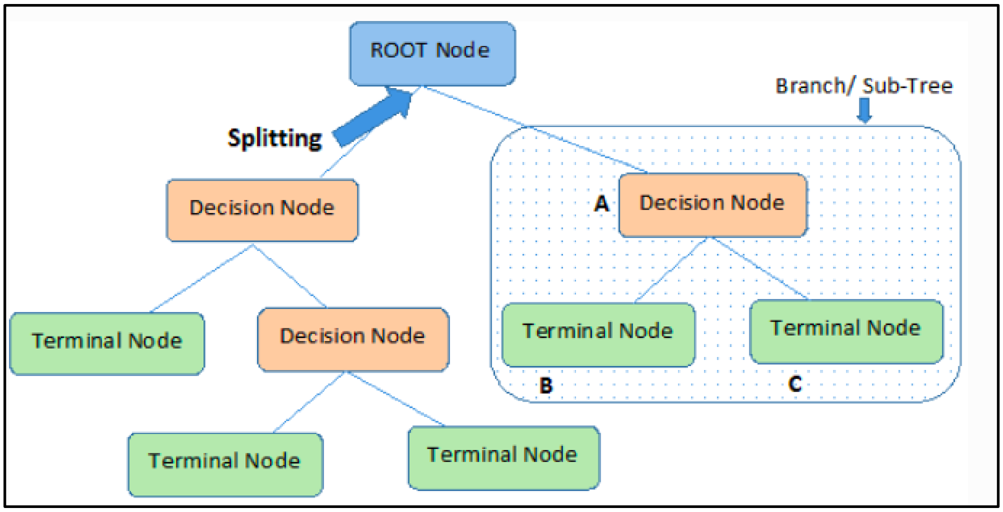
4.6 Decision Tree Classifier

Decision tree classifier is the supervised machine learning technique which solves the problem by transforming the data intro tree representation. Each internal node of the tree denotes an attribute and each leaf node represents the class label. It splits features based on certain thresholds and this process is repeated on each subset in a recursive manner known as recursive partitioning. In our model , we have used ID3 decision tree which is based on top-down recursive partitioning. Following steps are involved in ID3:

1. Initially it begins with original dataset before partitioning the dataset into subsets
2. For selecting the feature for root node, it iterates through each unused attribute of the set and calculates the information gain either through Entropy or by Gini Index.
3. It then selects the feature which has the maximum information gain and the split is made based on the selected attribute to divide the dataset into subsets
4. The algorithm is repeated on each subset, considering only attributes never used before as shown in figure 41 below.

Figure 41

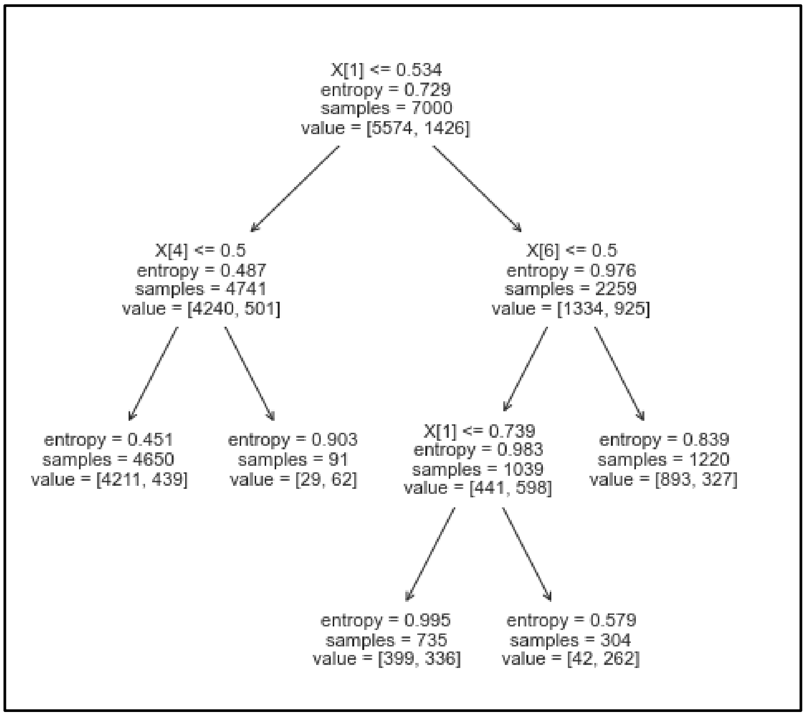
Decision tree algorithm



The tree for our model for max leaf node set to five is shown below in figure 42.

Figure 42

Decision tree for max depth 5



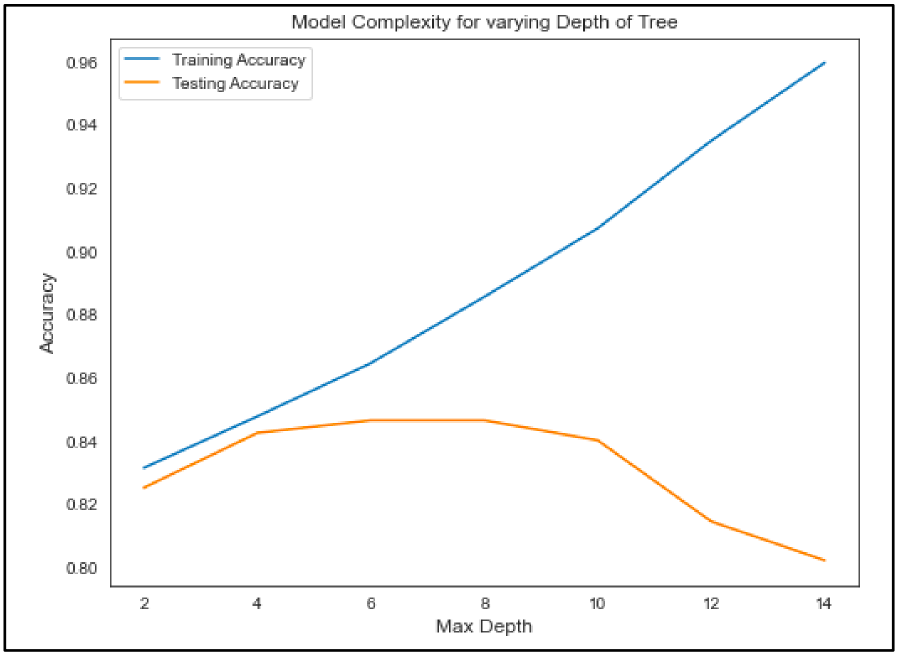
In our model, we have performed hyper-parameter tuning on following parameters:

1. Max_leaf_nodes: This is used to set the depth of the tree. We have defined different values which are 5,10, 20, 30
2. Min_samples_split:This determines the minimum number of samples to be at leaf node. For our model the values are 3, 4
3. Criterion: This is used to defined the method to calculated the information gain. Two methods are defined entropy and gini index.
4. Max_depth: this is used to set the depth of the tree.

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 43 shows the model complexity for at different depth.

Figure 43

Model complexity for different values of max depth

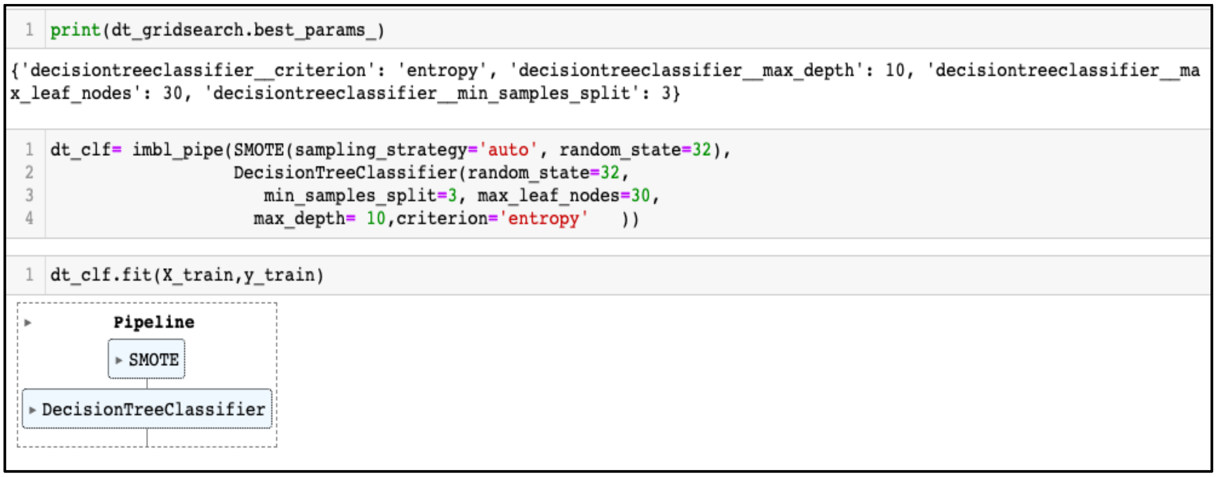


From the figure 43 above we can see that when depth increases the model begins to overfit. Hence, for our model best parameters are:

'decisiontreeclassifier__criterion': 'entropy', 'decisiontreeclassifier__max_depth': 4, 'decisiontreeclassifier__max_leaf_nodes': 30, 'decisiontreeclassifier__min_samples_split': 3. With these best parameters the model in trained and saved as shown below in figure 44.

Figure 44

Training the model with best parameters



```
print(dt_gridsearch.best_params_)
```

```
{'decisiontreeclassifier__criterion': 'entropy', 'decisiontreeclassifier__max_depth': 10, 'decisiontreeclassifier__ma
x_leaf_nodes': 30, 'decisiontreeclassifier__min_samples_split': 3}
```

```
dt_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),
                  DecisionTreeClassifier(random_state=32,
                     min_samples_split=3, max_leaf_nodes=30,
                    max_depth= 10,criterion='entropy'    ))
```

```
dt_clf.fit(X_train,y_train)
```

4.7 Ensemble Techniques

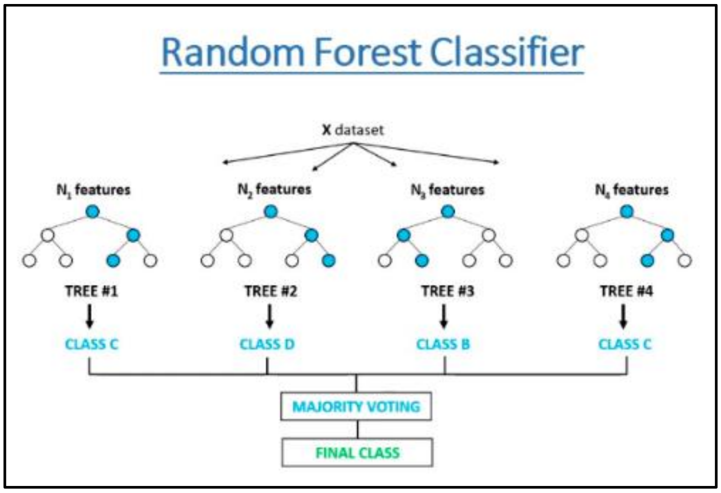
Ensemble methods combines the predictions from different classifiers for improving the overall performance. There are various methods through which ensembling can be done. Some of the common used methods are voting, bagging, boosting etc.we have used bagging and boosting methods in our churn prediction model. These are Random Forest Classifier, Ada Boost, Gradient Boost.

4.7.1 Random Forest Classifier

Random forest model is one of the ensemble model which are based on bagging technique i.e. bootstrap sampling. The basic idea is to combine multiple decision tree to determine the target class instead of depending on single tree. Random forest model has decision trees are base estimators. The dataset is randomly sampled with repetition and the size of the sample is equal to the original dataset size. It further increases the randomness by feature sampling known as sub-space sampling. The target variable class is predicted on the basis of majority votes to the class.

Figure 45

Random forest classifier



In our model, we have performed hyper-parameter tuning on following parameters:

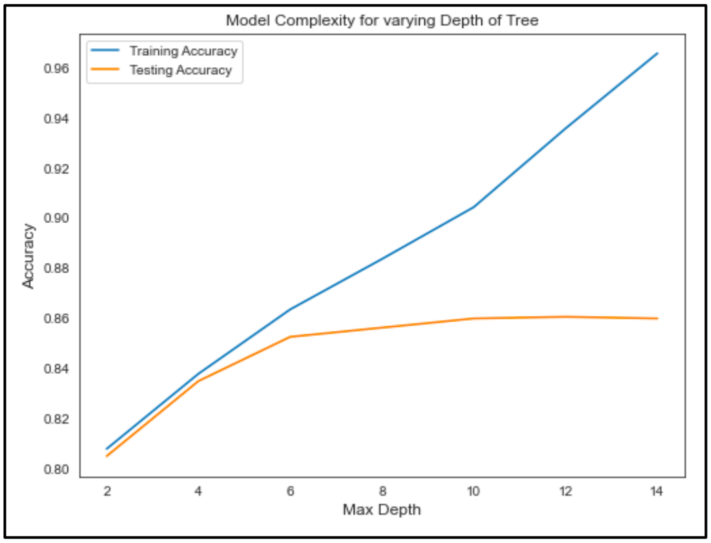
1. n_estimators: This is used to determine the number of estimators in the model. The values are 50, 100, 150,200.
2. Max_features: number of features to consider when looking for the best split. Values are 'sqrt','log2' ,0.33. If “sqrt”, then max_features=sqrt(n_features) and if “log2”, then max_features=log2(n_features).
3. min_samples_leaf: This determines the minimum number of samples to be at leaf node. For our model the values are 1, 5 ,10, 15,20
4. criterion: This is used to defined the method to calculated the information gain. Two methods are defined entropy and gini index.
5. min_samples_split: minimum number of samples required to split an internal node. Values are 3, 4,5.

6. max_depth: to set the maximum depth of the tree.values are 4,6,8,10.

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure shows the model complexity for at different depth.

Figure 46

Model complexity for different values of max depth



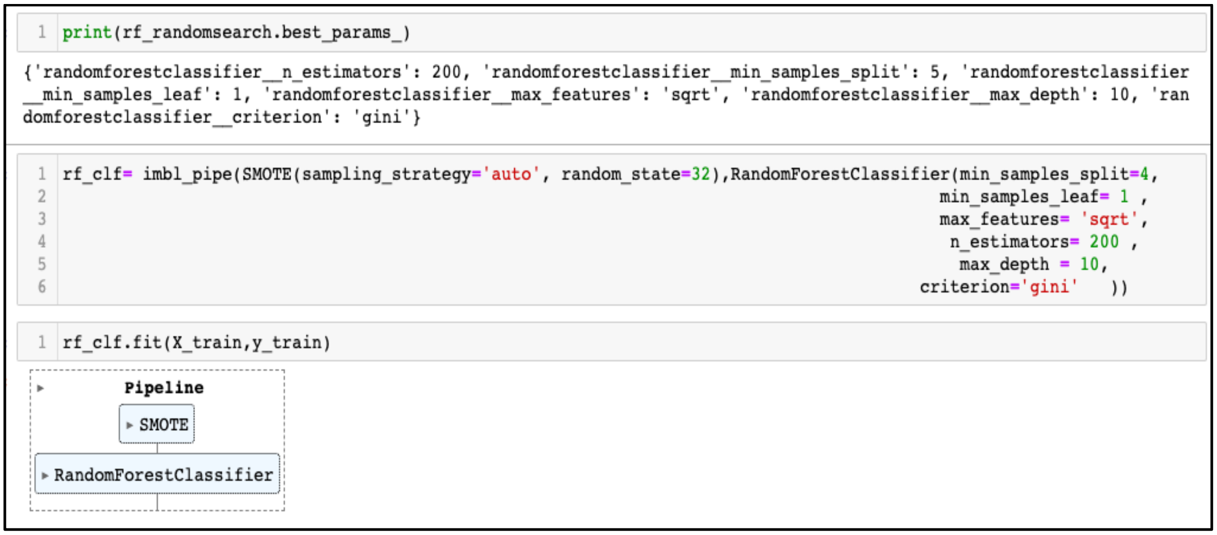
From the figure above we can see that overfitting of the model. Hence, for our model best parameters are:

'randomforestclassifier__n_estimators': 200, 'randomforestclassifier__min_samples_split': 5, 'randomforestclassifier__min_samples_leaf': 1, 'randomforestclassifier__max_features': 'sqrt', 'randomforestclassifier__max_depth': 10, 'randomforestclassifier__criterion': 'gini'

With these best parameters the model is trained and saved as shown below in figure 47.

Figure 47

Training the model with best parameters



```
print(rf_randomsearch.best_params_)
```

```
{'randomforestclassifier__n_estimators': 200, 'randomforestclassifier__min_samples_split': 5, 'randomforestclassifier
__min_samples_leaf': 1, 'randomforestclassifier__max_features': 'sqrt', 'randomforestclassifier__max_depth': 10, 'ran
domforestclassifier__criterion': 'gini'}
```

```
rf_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),RandomForestClassifier(min_samples_split=4,
                                                                                        min_samples_leaf= 1 ,
                                                                                        max_features= 'sqrt',
                                                                                         n_estimators= 200 ,
                                                                                          max_depth = 10,
                                                                                      criterion='gini'   ))
```

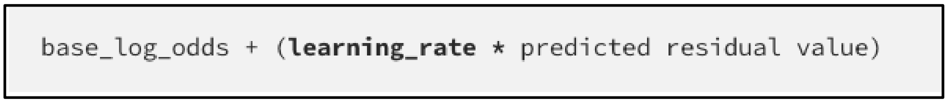
```
rf_clf.fit(X_train,y_train)
```

4.7.2 Gradient Boosting Classifier

Gradient boosting is one of the popular algorithm. In this model, each predictor corrects the wrongly predicted values produced by its predecessor. Each model is trained using residual errors of its predecessor as labels. Following steps are involved in gradient boosting classifier:

1. Algorithm calculates the log of odds of target variable
2. These logits are converted into probabilities in order to make the prediction
3. For each instance, it calculates the residuals i.e. observed values minus the predicted value
4. Once it has done this, it build a new Decision Tree that actually tries to predict the residuals that was previously calculated
5. To make prediction, the model calculate the log of odds of each variable and convert prediction into probability

Formula for making prediction is :



```
base_log_odds + (learning_rate * predicted residual value)
```

The learning_rate is a hyperparameter which is used to scale each trees contribution, sacrificing bias for better variance.

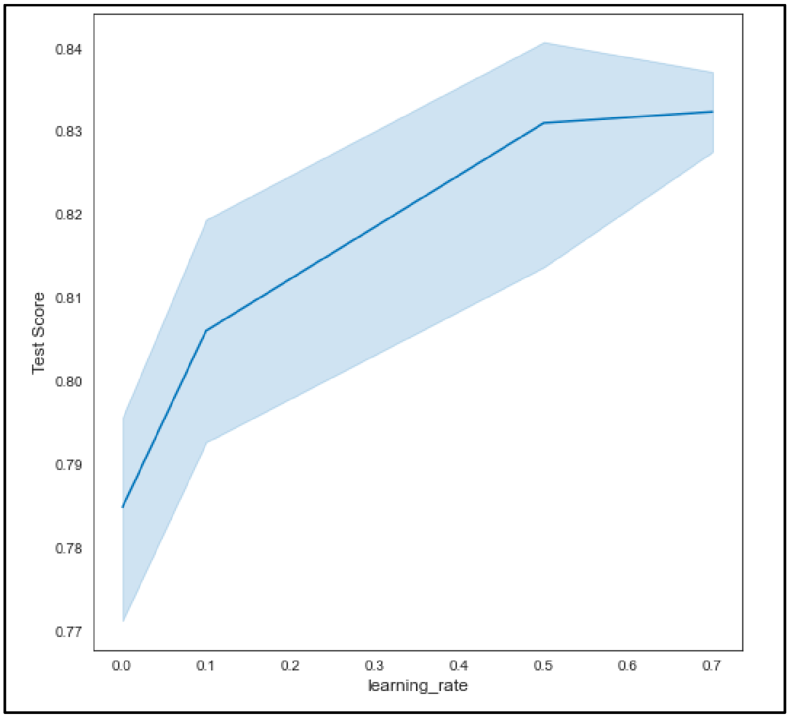
In our model, we have performed hyper-parameter tuning on following parameters:

1. n_estimators: This is used to determine the number of estimators in the model. The values are 50, 100, 150,200.
2. max_features: number of features to consider when looking for the best split. Values are 'sqrt','log2' ,0.33. If “sqrt”, then max_features=sqrt(n_features) and if “log2”, then max_features=log2(n_features).
3. learning_rate: This shrinks the contribution of each tree by learning_rate. Values are 0.001,0.1,0.3,0.5,0.7
4. criterion: This is used to defined the method to calculated the information gain. Two methods are defined entropy and gini index.
5. min_samples_split: minimum number of samples required to split an internal node. Values are 3, 4,5.
6. max_depth: to set the maximum depth of the tree.values are 2,4,5.

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 48 shows the accuracy of the model at different learning rates.

Figure 48

Accuracy score for different values of learning rate



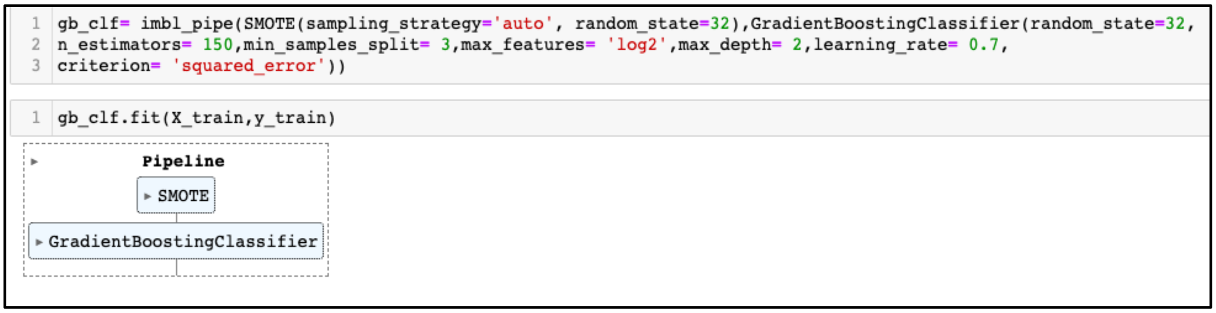
Hence, for our model best parameters are:

'gradientboostingclassifier__n_estimators': 100, 'gradientboostingclassifier__min_samples_split': 4, 'gradientboostingclassifier__max_features': 'sqrt', 'gradientboostingclassifier__max_depth': 4, 'gradientboostingclassifier__learning_rate': 0.5, 'gradientboostingclassifier__criterion': 'squared_error'

With these best parameters the model in trained and saved as shown below in figure 49.

Figure 49

Training the model with best parameters



```
gb_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),GradientBoostingClassifier(random_state=32,
n_estimators= 150,min_samples_split= 3,max_features= 'log2',max_depth= 2,learning_rate= 0.7,
criterion= 'squared_error'))
```

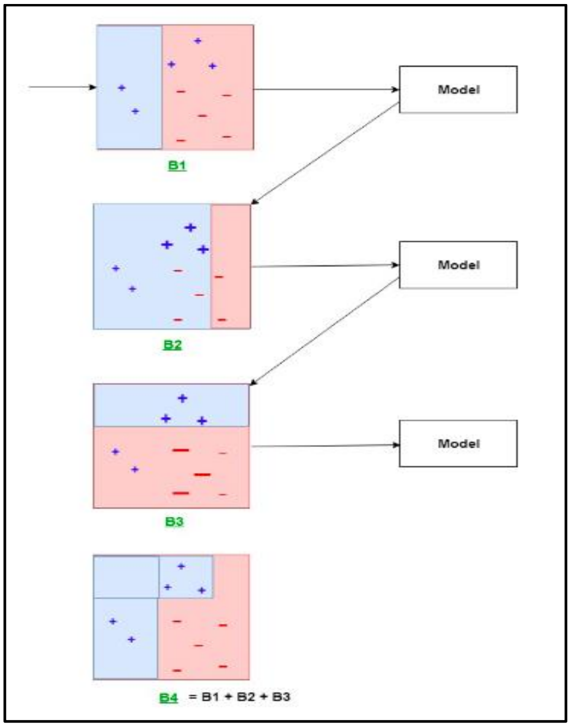
```
gb_clf.fit(X_train,y_train)
```

4.7.3 Ada Boost Classifier

Ada boost(adaptive boosting) is based on concept that weak learners combine to form strong learners. In this model, each estimator is known as stump. Each model is trained sequentially and the model tries to correct the wrongly predicted values by predecessor by tweaking the weight. The weights are increased for errors whereas weights are decreased for correct values. In classifier, the values is predicted based on majority voting and each estimator has weightage ‘alpha’ in final prediction which depend on the error rate of the estimator. Higher error rate leads to less weightage. The figure 50 below shows ada boost model.

Figure 50

Ada boost classifier



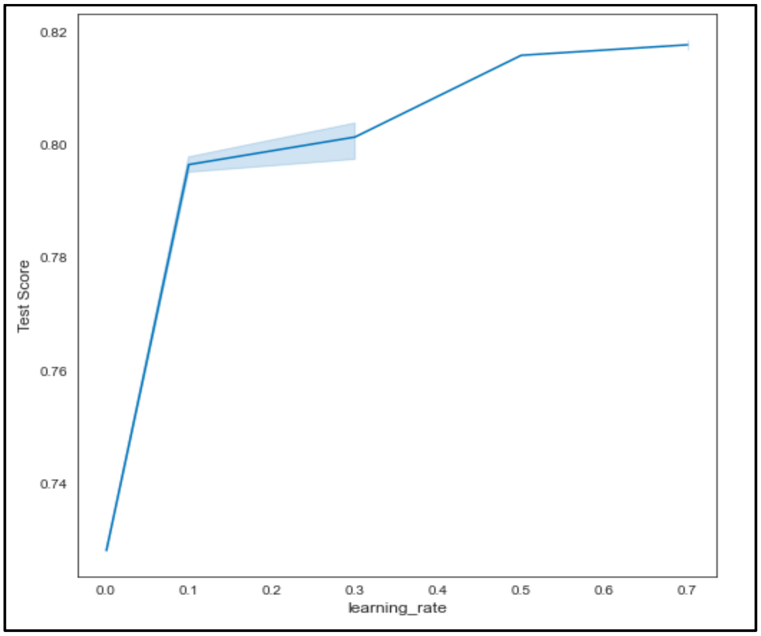
In our model, we have performed hyper-parameter tuning on following parameters:

1. n_estimators: This is used to determine the number of estimators in the model. The values are 50, 100, 150,200.
2. learning_rate: This shrinks the contribution of each tree by learning_rate. Values are 0.001,0.1,0.3,0.5,0.7

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 51 shows the accuracy of the model at different learning rates.

Figure 51

Accuracy score for different values of learning rate



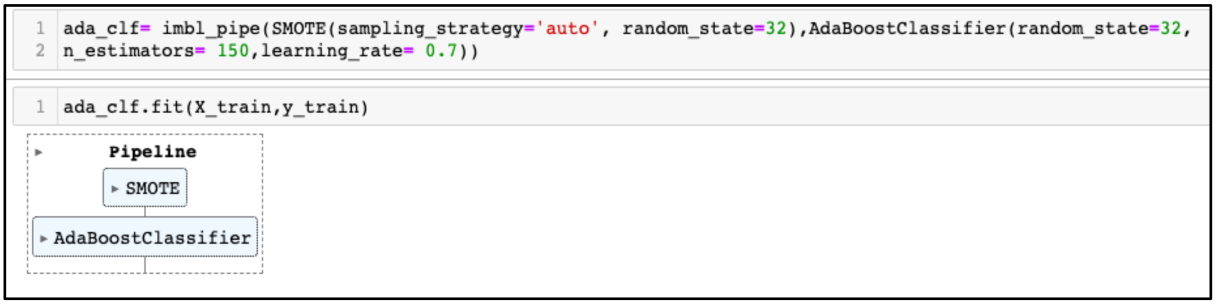
Hence, for our model best parameters are:

'adaboostclassifier__n_estimators': 150, 'adaboostclassifier__learning_rate': 0.7

With these best parameters the model in trained and saved as shown below in figure 52.

Figure 52

Training the model with best parameters



```
ada_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),AdaBoostClassifier(random_state=32,
n_estimators= 150,learning_rate= 0.7))
```

```
ada_clf.fit(X_train,y_train)
```

4.8 XGBoost Classifier

XGboost model used gradient boosting technique along with regularization parameter. It avoid overfitting of the model by pruning the trees based on similarity score.

It considers the “Gain” of a node as the difference between the similarity score of the node and the similarity score of the children. If the gain from a node is found to be minimal then it just stops constructing the tree to a greater depth which can overcome the challenge of overfitting to a great extend. In XGBoost, when the model fails to predict the anomaly for the first time, it gives more preferences and weightage to it in the upcoming iterations thereby increasing its ability to predict the class with low participation. This is good when there is class imbalance in the dataset.

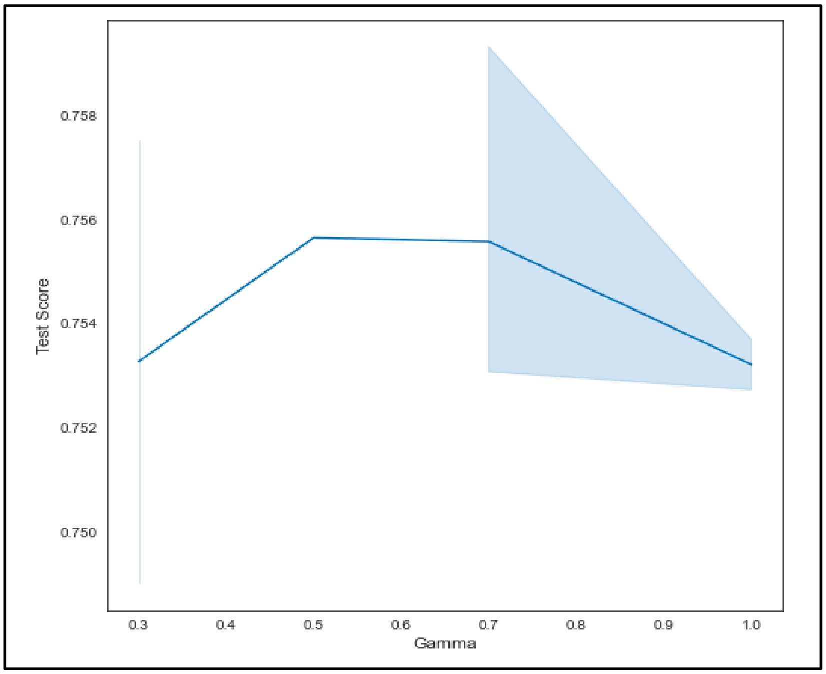
In our model, we have performed hyper-parameter tuning on following parameters:

1. n_estimators: This is used to determine the number of estimators in the model. The values are 50, 100, 150,200.
2. gamma: It is the minimum loss reduction to create new tree-spli**t**. Values are 0.3, 0.5,0.7, 1
3. Max_depth: This is used to define the depth of the tree

Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. Below figure 53 shows the accuracy of the model at different value of gamma function.

Figure 53

Accuracy score for different values of gamma



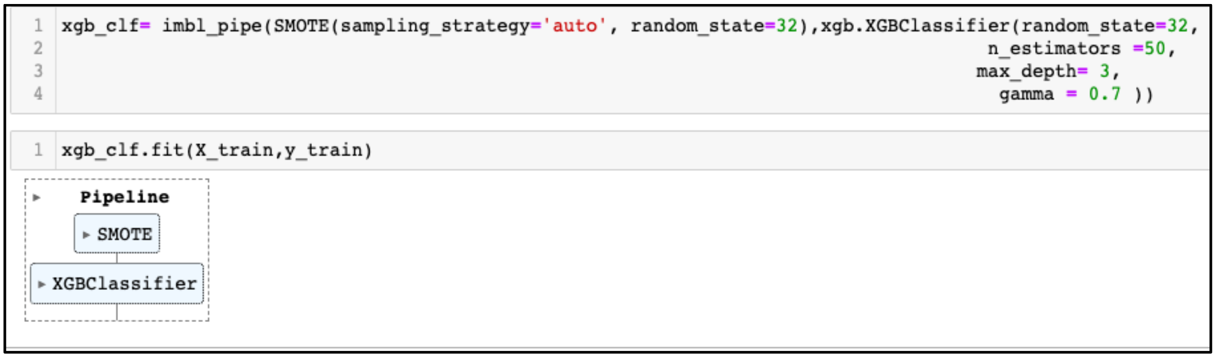
Hence, for our model best parameters are:

'xgbclassifier__n_estimators': 50, 'xgbclassifier__max_depth': 3, 'xgbclassifier__gamma': 0.7

With these best parameters the model in trained and saved as shown below in figure 54.

Figure 54

Training the model with best parameter



```
xgb_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),xgb.XGBClassifier(random_state=32,
                                                                                   n_estimators =50,
                                                                                  max_depth= 3,
                                                                                    gamma = 0.7 ))
```

```
xgb_clf.fit(X_train,y_train)
```

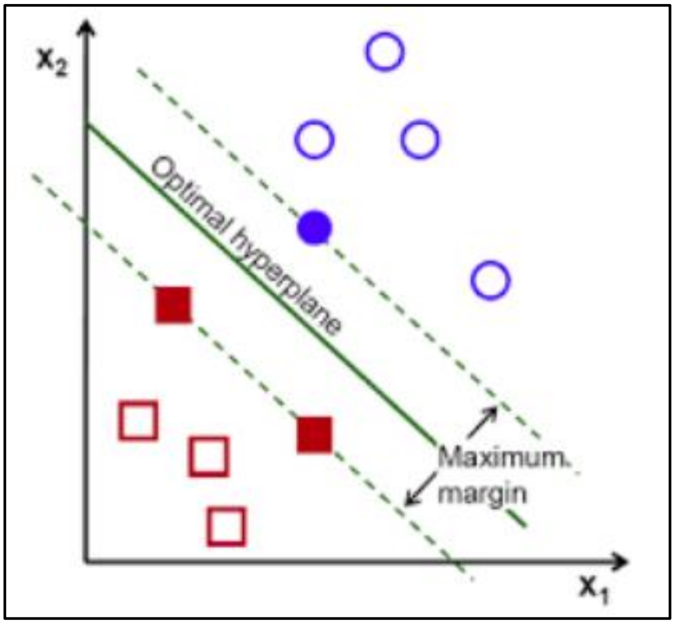
```
▸      Pipeline
    ▸ SMOTE
▸ XGBClassifier
```

4.9 Support Vector Classifier

SVC separates the classes by finding the optimal hyper plane and linearly separating the training data. The algorithm finds the best hyper plane which maximizes the margin i.e. minimum distance between from the decision boundary to any training point and points which are close to hyper plane are called support vectors. If the data is not linearly seperable, it uses kernel trick to linearly separate the data. There are different types of kernel functions available which are polynomial kernel, radial basis kernel etc. The model works well when there dimensionality of the data is high. Figure 55 below shows the optimal hyperplane separating two classes.

Figure 55

Support vector classifier algorithm



In our model, we have performed hyper-parameter tuning on following parameters:

1. kernel: This is used to define the type of transformation to linearly classify the data. Functions are 'linear', 'rbf', 'poly', 'sigmoid'.

2. gamma: This parameter decides how far the influence of a single training example reaches during transformation. Values are 1, 0.1, 0.01
3. C: This parameter controls the amount of regularization applied to the data. Values are 0.0005,0.001, 0.01, 0.1, 0.5

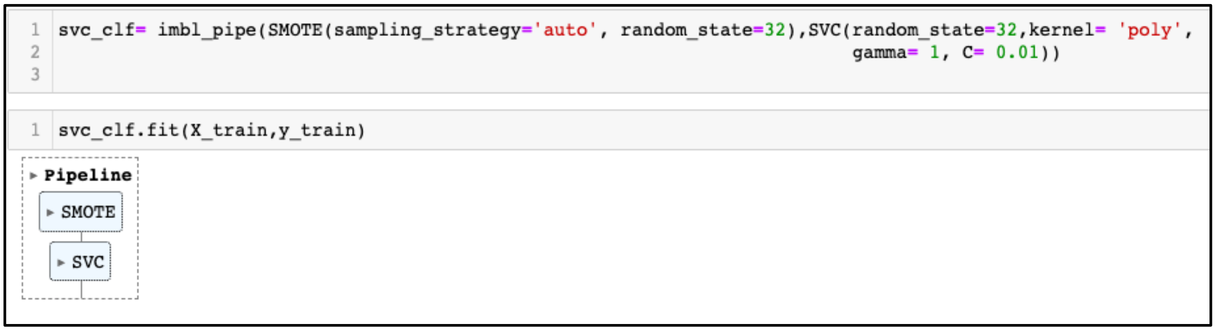
Hence, by using GridSearchCV model performance will check for all possible combinations and the best parameter (by using command lr_gridsearch.best_params_) is selected based on accuracy. For our model best parameters are:

'svc__kernel': 'poly', 'svc__gamma': 1, 'svc__C': 0.01

With these best parameters the model in trained and saved as shown below in figure 56.

Figure 56

Training the model with best parameters



```
svc_clf= imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),SVC(random_state=32,kernel= 'poly',
                                                                       gamma= 1, C= 0.01))
```

```
svc_clf.fit(X_train,y_train)
```

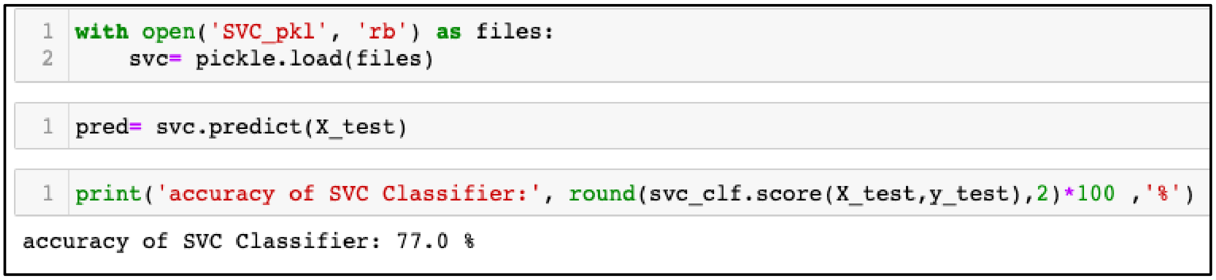
5 Evaluation

5.1 Performance Comparison between Models with and without SMOTE

Saved models are called using pickle.load function as shown below in figure 57 and these models are used to evaluate the performance of these models on test data.

Figure 57

Deserialising the model using pickle.load



```
with open('SVC_pkl', 'rb') as files:
    svc= pickle.load(files)
```

```
pred= svc.predict(X_test)
```

```
print('accuracy of SVC Classifier:', round(svc_clf.score(X_test,y_test),2)*100 ,'%')
```

```
accuracy of SVC Classifier: 77.0 %
```

Results are summarized in table for data with and without SMOTE.

Table 1 showing performance results when the target class is balanced (oversampling using SMOTE)

Table 1

Performance score when data is oversampled

Model	Precision	Recall	F1_score	Auc Roc Score	Accuracy
Logistic Regression	38.0 %	69.0 %	49.0 %	76.58%	71%
KNN	46.0 %	53.0 %	50.0 %	74.78%	78 %
Decision Tree	52.0 %	66.0 %	57.99%	83.94%	81%
Random Forest	**56.1 %**	**61.0 %**	**59.0 %**	**85.56%**	**82%**
Ada Boost	**55.1 %**	**66.0 %**	**60.0 %**	**85.11%**	**82%**

Gradient Boosting	**61.0 %**	**60.0 %**	**61.0 %**	**85.61%**	**84%**
XGBoost	**67.0 %**	**56.1 %**	**61.0 %**	**86.54%**	**85%**
SVC	45.0 %	67.0 %	54.0 %	86.09%	77%
Gaussian Naive Bayes	37.0 %	68.0 %	47.0 %	76.10%	69%
Perceptron	28.96 %	61.0 %	39.0 %	65%	62%

Table 2 showing performance results when the target class is imbalanced.

Table 2

Performance score when target class is imbalanced.

Model	**Precision**	**Recall**	**F1_score**	**Auc Roc Score**	**Accuracy**
Logistic Regression	63.0 %	23.0 %	34.0 %	77.25%	82.0 %
KNN	68.0 %	27.0 %	39.0 %	79.13%	82.0%
Decision Tree	70.0 %	49.0 %	57.99 %	83.14%	85.0 %
Random Forest	77.0 %	46.0 %	56.9 %	86.45%	86.0 %
Ada Boost	76.0 %	44.0 %	56.01 %	86.46%	86.0 %
Gradient Boosting	74.0 %	49.0 %	59.0 %	86.45%	86.0 %
XGBoost	73.0 %	49.0 %	59.0 %	86.45%	86.0 %
SVC	0	0	0	59%	80.0 %

Gaussian Naive Bayes	54.0 %	36.0 %	43.0 %	77.77%	81.0 %
Perceptron	32.0 %	16.0 %	22.0 %	65%	76.0 %

From table 2, we can see that when the target class is imbalanced the f1_score is very low as compared to f1_score in table 1 although the accuracy score is still good. Particularly, for SVC model when the target class is imbalanced the f1 score was 0. This is because f1 score is harmonic mean of precision and recall and if the one the value is less f1 score decreases. For this reason, we have first oversampled the data to make the target class balanced and then trained the model. With reference to results summarized in table 1, we have selected top four models based on f1 score and accuracy. These are random forest, ada boost, gradient boosting, xgboost model.

5.2 Confusion Matrix and Roc Auc Curve for top performing Models

Below figure 58 and figure 59 shows confusion matrix and Roc Auc curve for random forest model.

Figure 58

Confusion matrix for random forest model

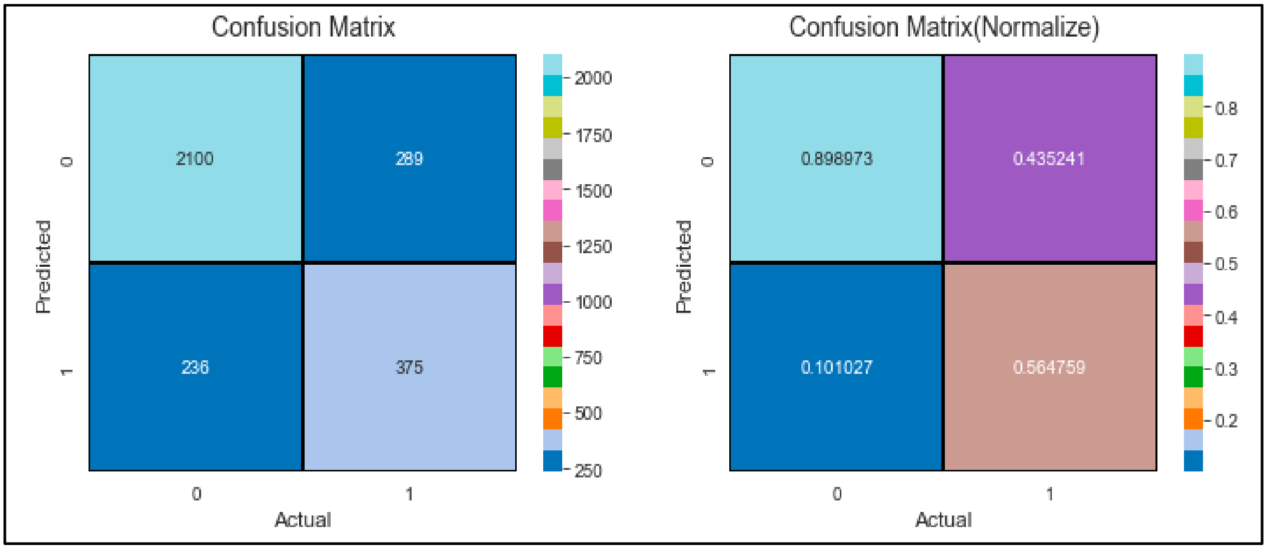
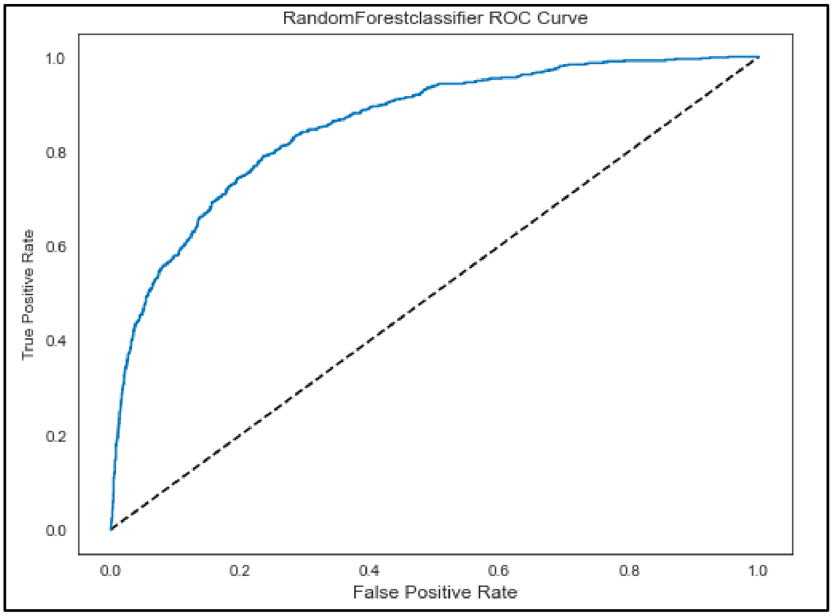


Figure 59

ROC-AUC for random forest model



Below figure 60 and figure 61 shows confusion matrix and Roc Auc curve for gradient boost model.

Figure 60

Confusion matrix for gradient boosting model

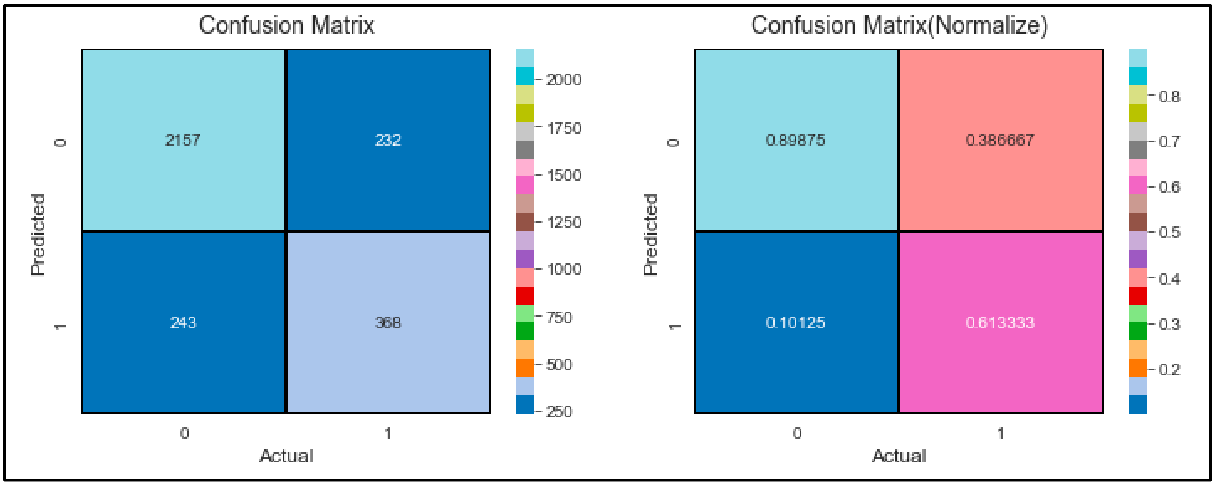
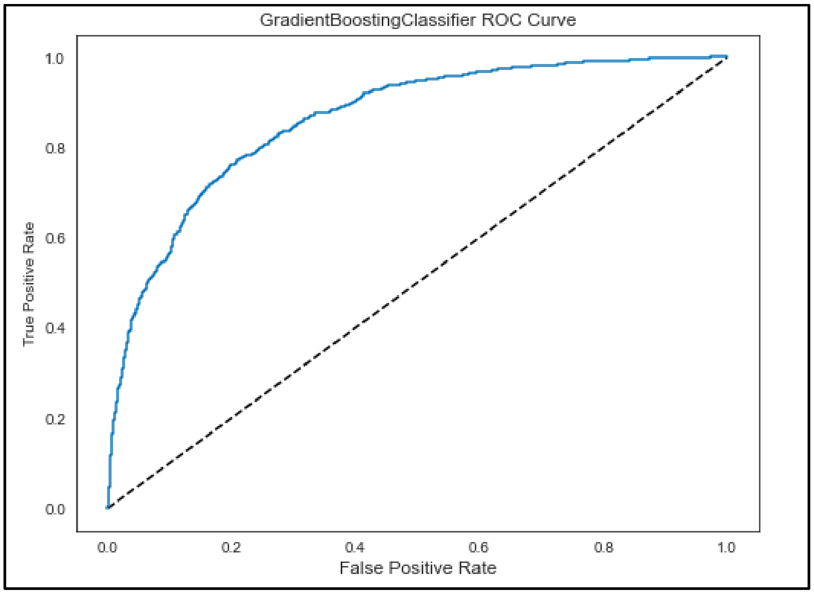


Figure 61

ROC-AUC for gradient boosting model



Below figure 62 and 63 shows confusion matrix and Roc Auc curve for Ada Boost Model.

Figure 62

Confusion matrix for ada boost model

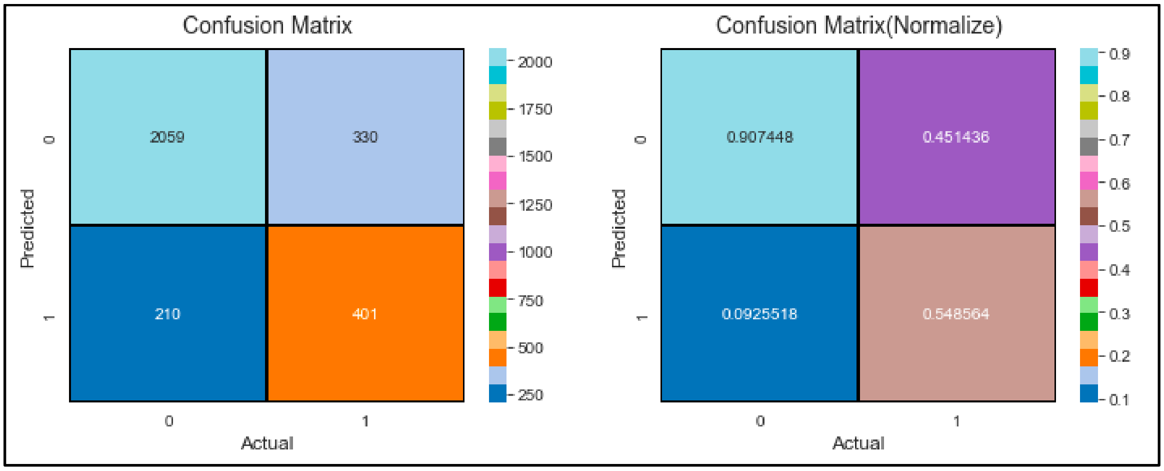
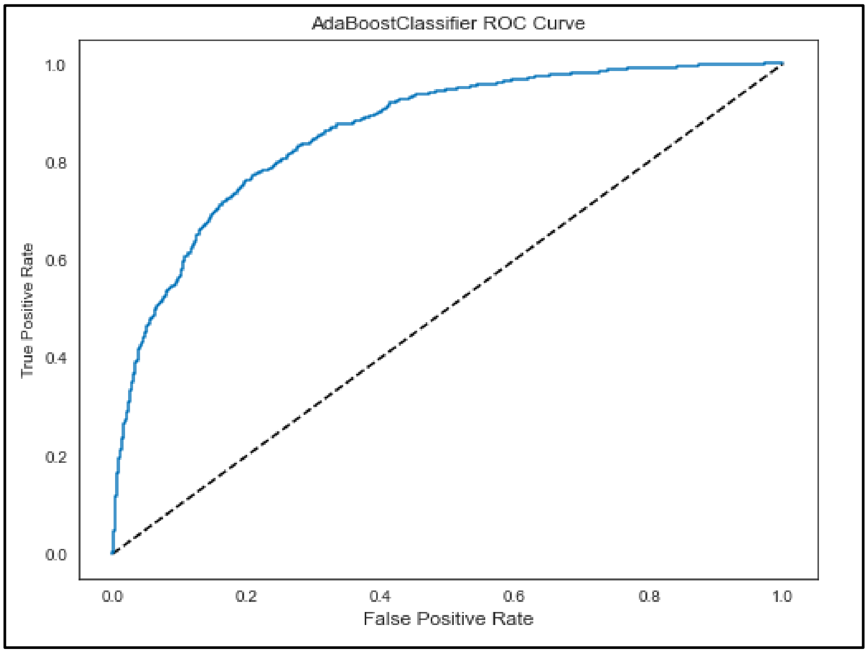


Figure 63

ROC-AUC for ada boost model



Below figure 64 and figure 65 shows confusion matrix and Roc Auc curve for XgBoost Model.

Figure 64

Confusion matric for XGBoost model

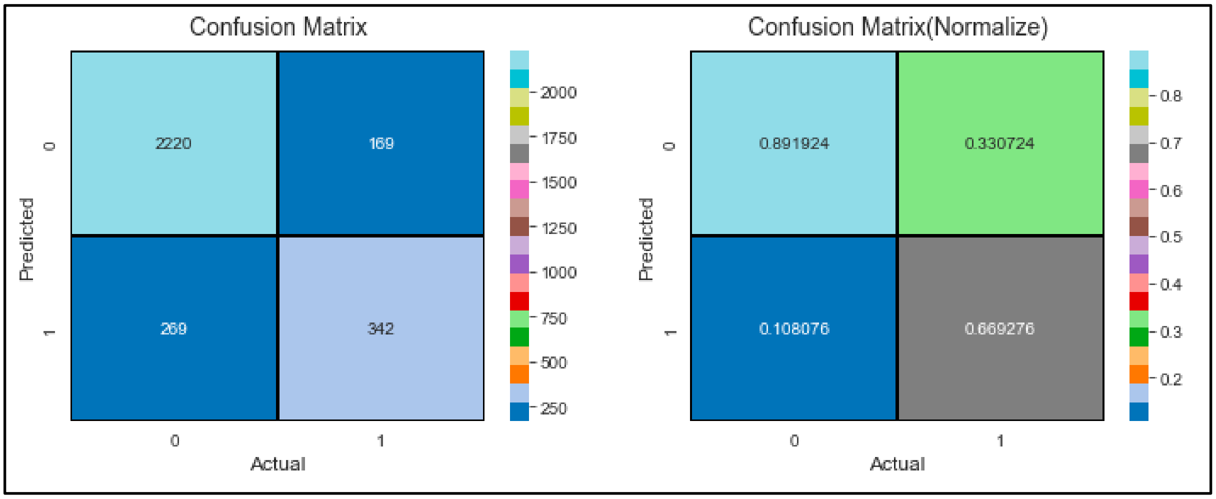
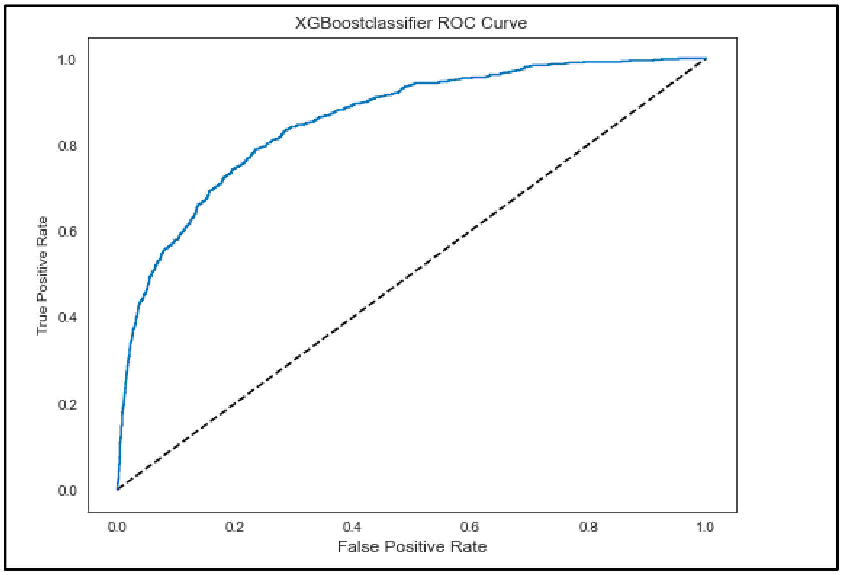


Figure 65

ROC-AUC for XGBoost model



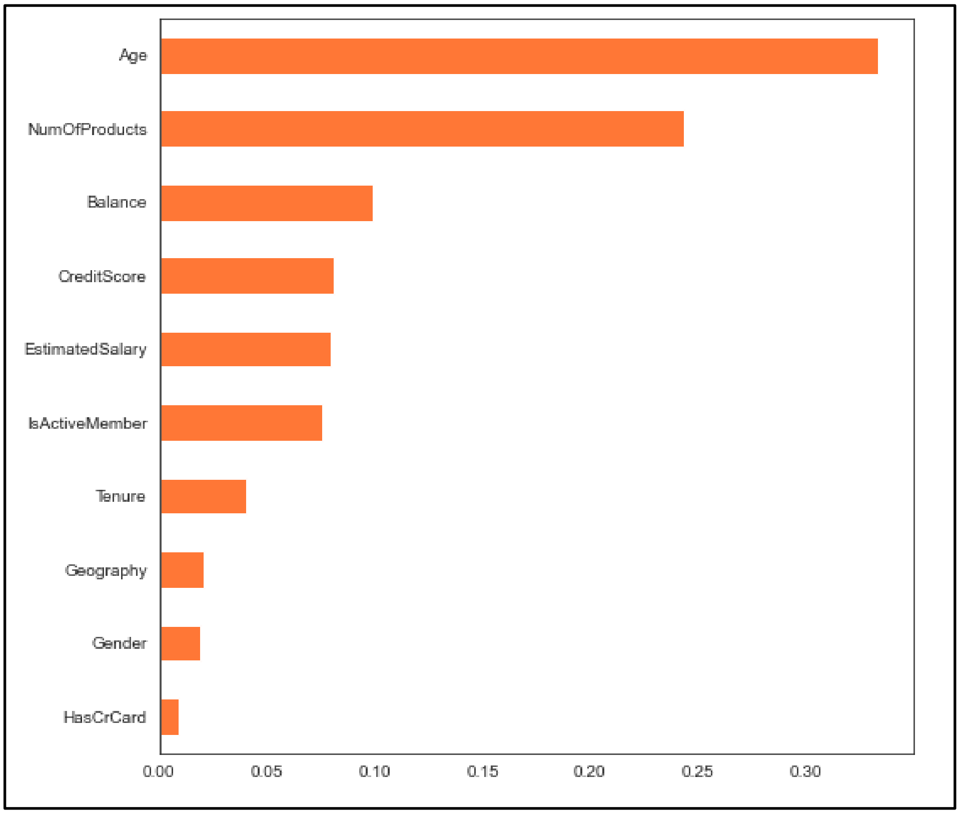
With reference to confusion matrix for top performing model, the true positive rate and true negative almost similar. Also, the ROC curve is quite similar in all these models. Hence, we performed ensemble on these models using voting classifier in which prediction was decided by majority voting.

5.2 Model Explanation

We analyzed which feature is more important by SHAP and random forest feature importance. With reference to figure 66, we can see that Gender and HasCrCard are least relevant features.

Figure 66

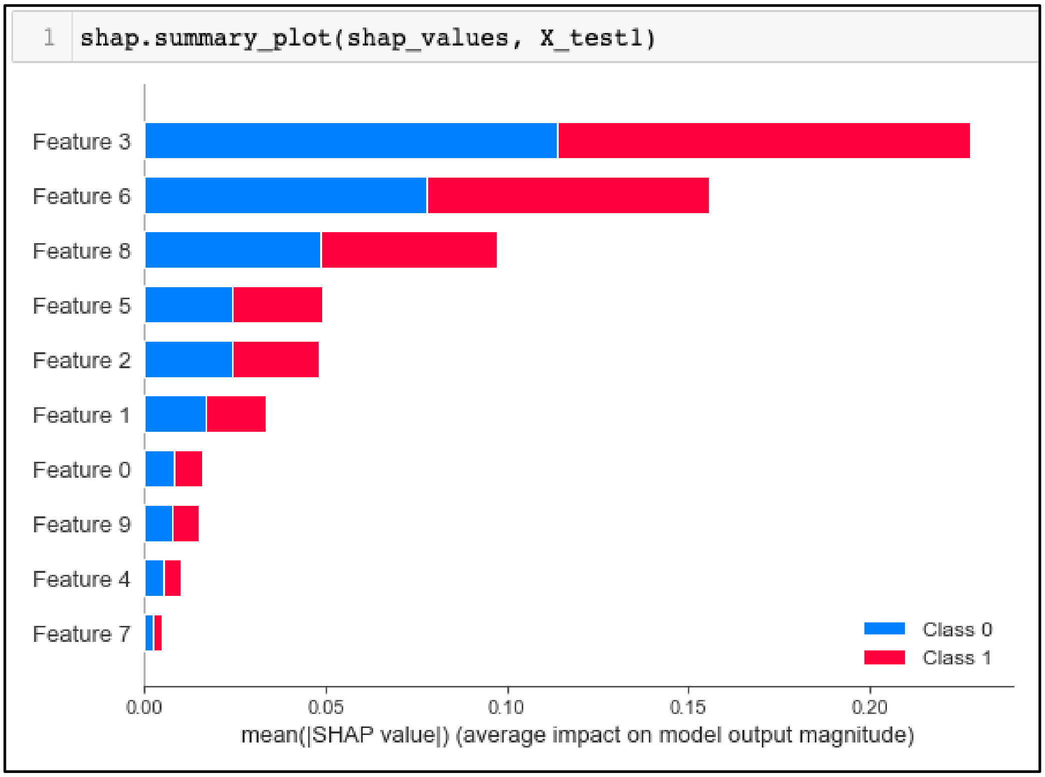
Random forest feature importance



Also, we analyzed the relation of descriptive features with target variable by using SHAP(Shapely additive explanations). By using SHAP, we can see positive and negative relationship of descriptive features with target variable as shown in figure 67 below.

Figure 67

Feature importance using SHAP



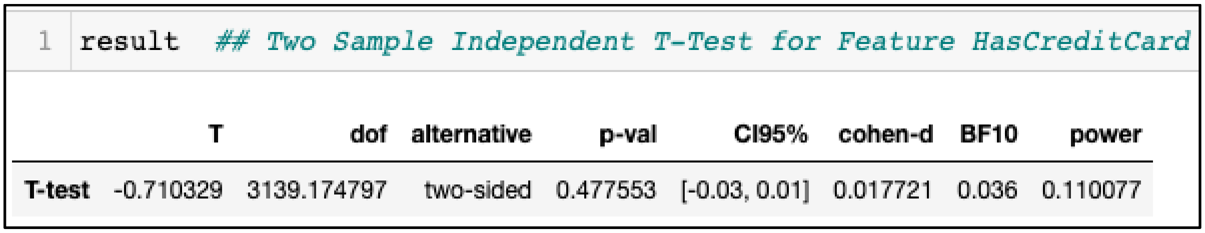
```
shap.summary_plot(shap_values, X_test1)
```

Hence, with reference to above figures, it is evident that HasCrCard is the least important feature whereas Age feature is the most important feature. The ‘Balance’ feature is important feature in both SHAP and random forest feature importance.

To cross verify the result, we also implemented two sample independent t-test using python pinguoin package. Significance level was set at 0.05 to balance the False positive and False negative and from figure 68 we can see that p-value is significantly greater for feature HasCrCard. This shows that there is no difference in variance in both the groups. Hence, ‘HasCrCard’ feature can be dropped.

Figure 68

Two Sample Independent T-test for HasCrCard Feature



```
result   ## Two Sample Independent T-Test for Feature HasCreditCard
```

	T	dof	alternative	p-val	CI95%	cohen-d	BF10	power
T-test	-0.710329	3139.174797	two-sided	0.477553	[-0.03, 0.01]	0.017721	0.036	0.110077

For this reason, we have analyzed the performance of top four models by including only relevant features and by including all features.

Figure 69 below shows the recall score for RF, GB,ADA,XGB model with all features and relevant features.

Figure 69

Recall for top model using all features and relevant features

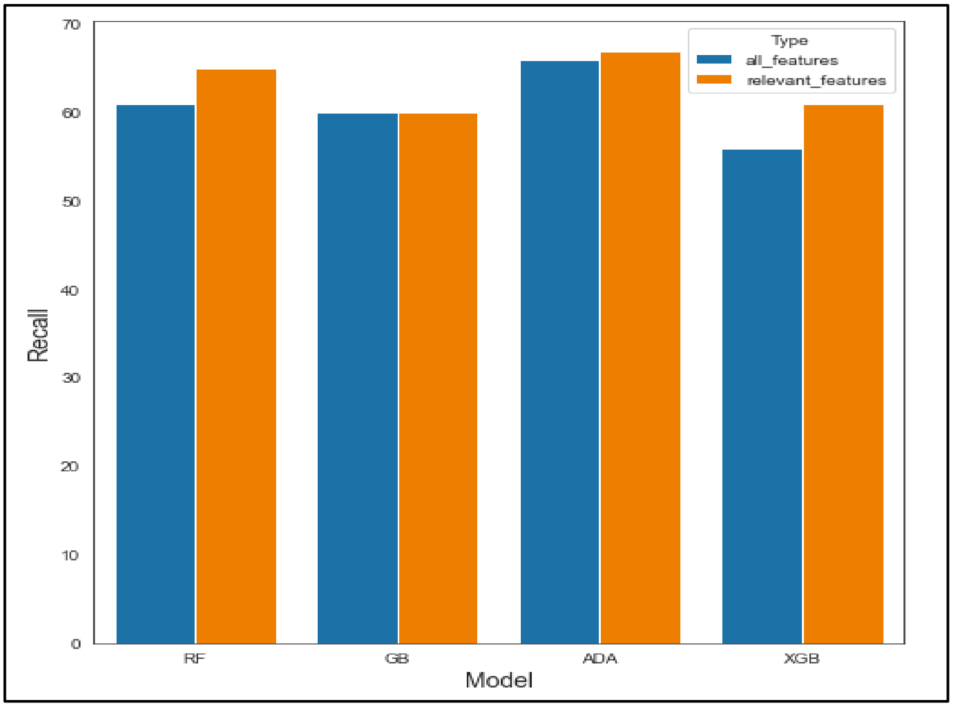
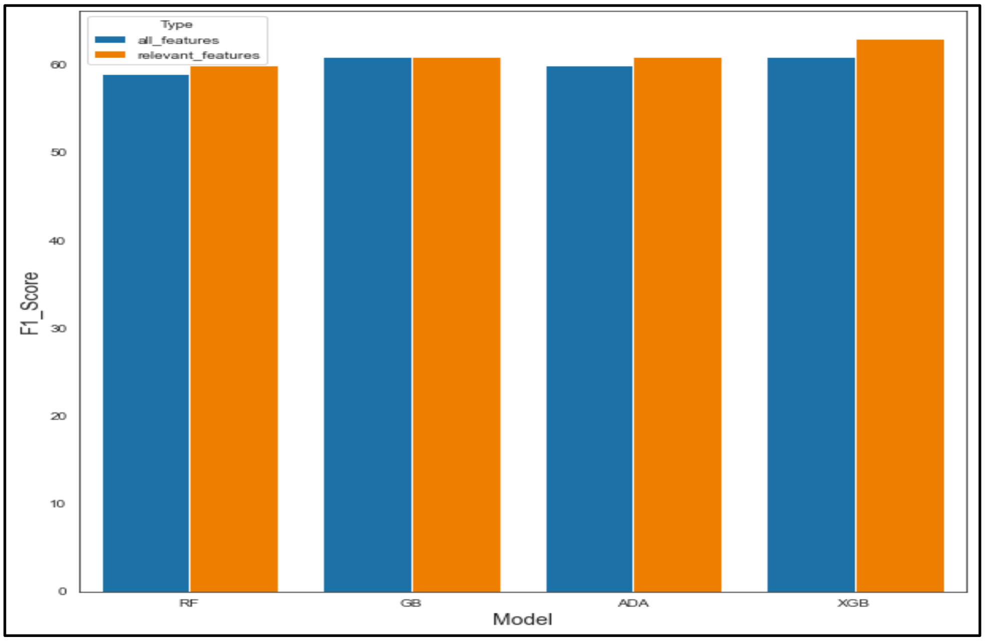


Figure 70 below shows F1 Score for RF, GB,ADA,XGB model with all features and relevant features.

Figure 70

Recall for top model using all features and relevant features



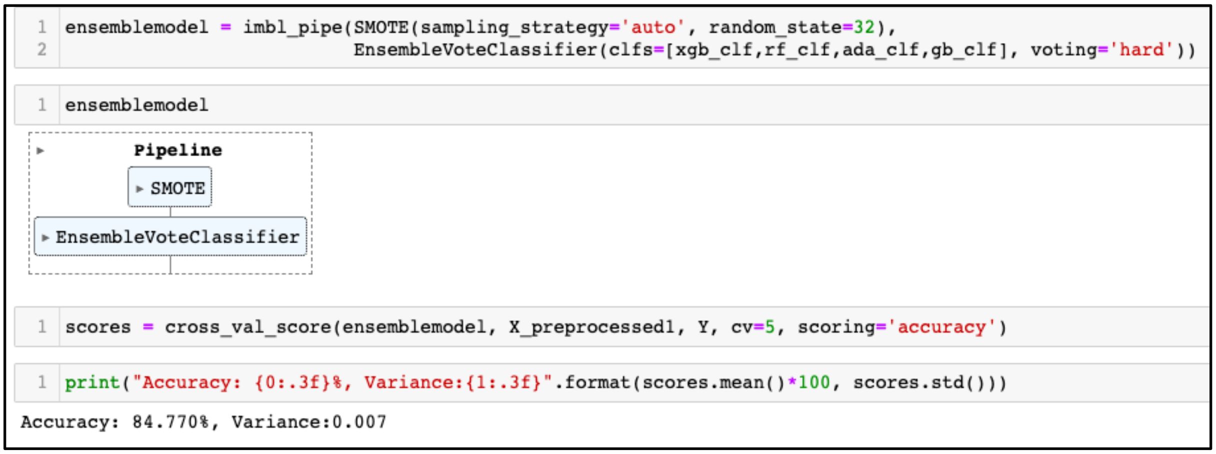
It is observed from the figures that f1 score increases slightly when we trained and test the model with relevant features. Hence, we trained ensemble model using relevant features.

5.3 K-Fold Cross Validation on Ensemble Model

We have used 5 fold cross validation to find the accuracy of the ensemble model. Below figure 71 shows the mean and variance for final ensemble model by performing cross validation on relevant features.

Figure 71

Ensemble model using voting classifier



```
ensemblemodel = imbl_pipe(SMOTE(sampling_strategy='auto', random_state=32),
                          EnsembleVoteClassifier(clfs=[xgb_clf,rf_clf,ada_clf,gb_clf], voting='hard'))
```

```
ensemblemodel
```

```
scores = cross_val_score(ensemblemodel, X_preprocessed1, Y, cv=5, scoring='accuracy')
```

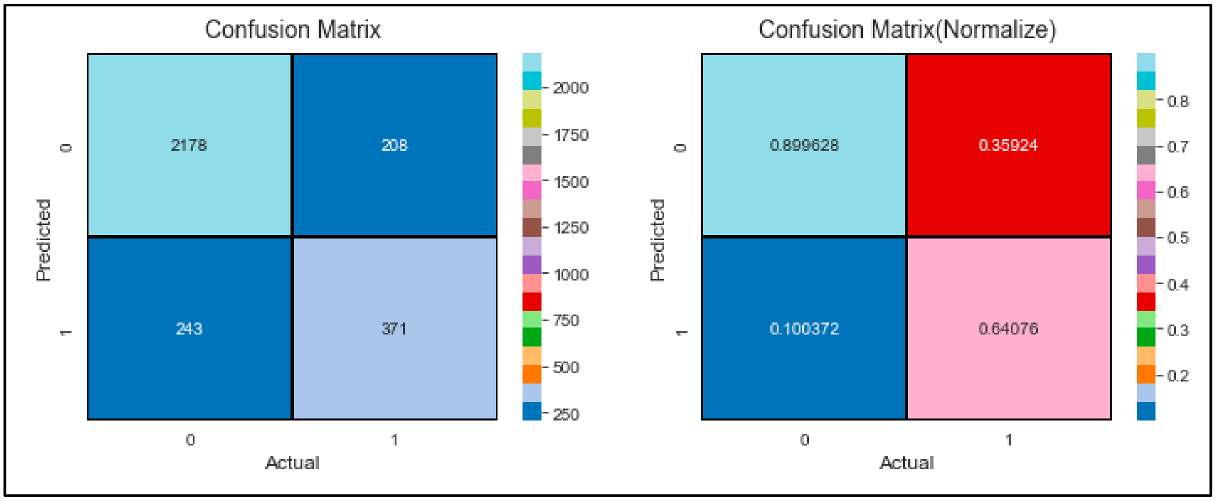
```
print("Accuracy: {0:.3f}%, Variance:{1:.3f}".format(scores.mean()*100, scores.std()))
```

```
Accuracy: 84.770%, Variance:0.007
```

Confusion matrix for ensemble model is shown below in figure 72.

Figure 72

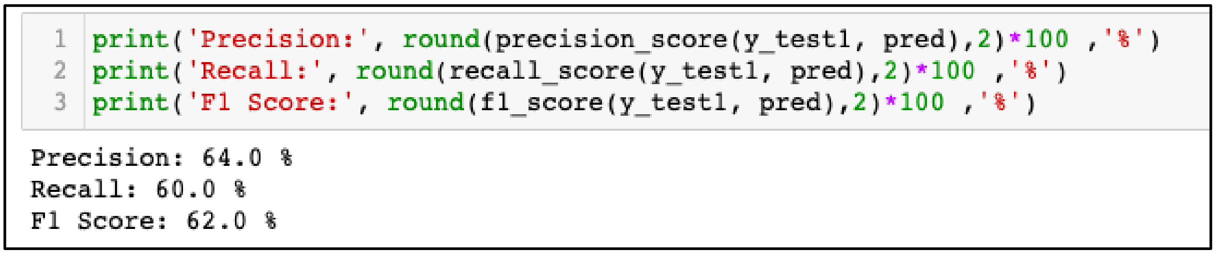
Confusion matrix for ensemble model



F1 score, precision and recall for ensemble model is shown below in figure 73. F1 score for ensemble model is highest as compared to individual models. But recall score is less as compared to individual models as shown below in figure 74.

Figure 73

Precision, recall, and f1 score for ensemble model



```
print('Precision:', round(precision_score(y_test1, pred),2)*100 ,'%')
print('Recall:', round(recall_score(y_test1, pred),2)*100 ,'%')
print('F1 Score:', round(f1_score(y_test1, pred),2)*100 ,'%')
```

```
Precision: 64.0 %
Recall: 60.0 %
F1 Score: 62.0 %
```

Figure 74

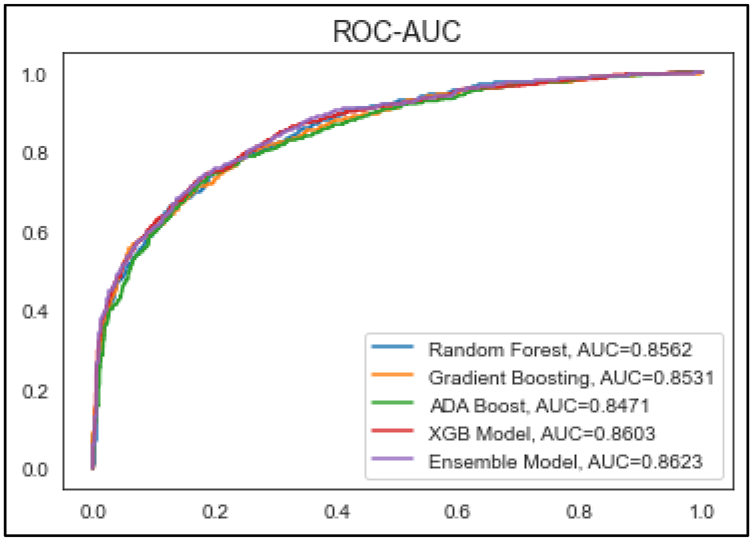
Precision, recall, and f1 score for ensemble model and individual models

Model	Precision	Recall	F1_Score
Ensemble	62.88%	61.81%	62.32%
RF	55.23%	68.48%	61.13%
ADA	54.56%	65.48%	59.52%
GB	60.76%	62.24%	61.47%
XGB	63.14%	61.41%	61.23%

Below figure 75 shows the comparison of ROC-AUC curve for individual models and ensemble model and the auc score for ensemble was highest as compared to individual models.

Figure 75

ROC-AUC   for individual and ensemble model



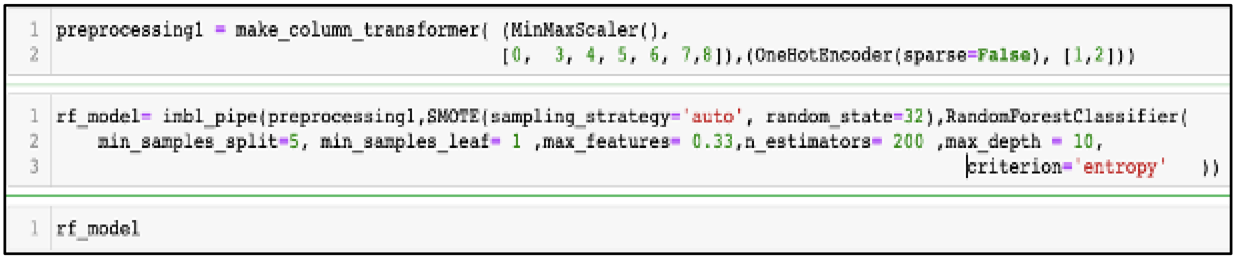
In churn prediction model, recall holds a higher importance as we want to accurately detect churn case. In other words, we want to decrease the false negative so that whenever any customer leaves the bank that effect can be detected accurately. Therefore, for this reason we have selected random forest classifier as final model because its recall score is highest and f1 score is approximately close to ensemble model.

6 Deployment

We have created a web application using the predictive model which can be used by bank to find out which customers are likely to leave the bank. Therefore, using random forest model, we have created a new pipeline in which data will be first processed and then trained with final model as shown in figure 76 below.

Figure 76

Pipeline to preprocess the data and trained the model



```
preprocessing1 = make_column_transformer( (MinMaxScaler(),
                                          [0,  3, 4, 5, 6, 7,8]),(OneHotEncoder(sparse=False), [1,2]))
```

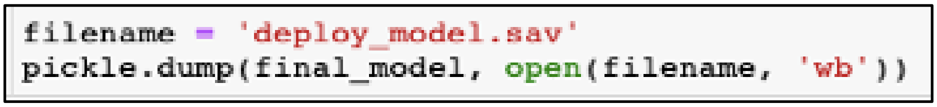
```
rf_model= imbl_pipe(preprocessing1,SMOTE(sampling_strategy='auto', random_state=32),RandomForestClassifier(
    min_samples_split=5, min_samples_leaf= 1 ,max_features= 0.33,n_estimators= 200 ,max_depth = 10,
                                                                                  criterion='entropy'   ))
```

```
rf_model
```

This model was saved on local machine with file named as ‘deploy_model’ using pickle as shown in figure 77 below. This file will be used in Spyder to create an application for Banks.

Figure 77

Using pickle to save the model for deployment

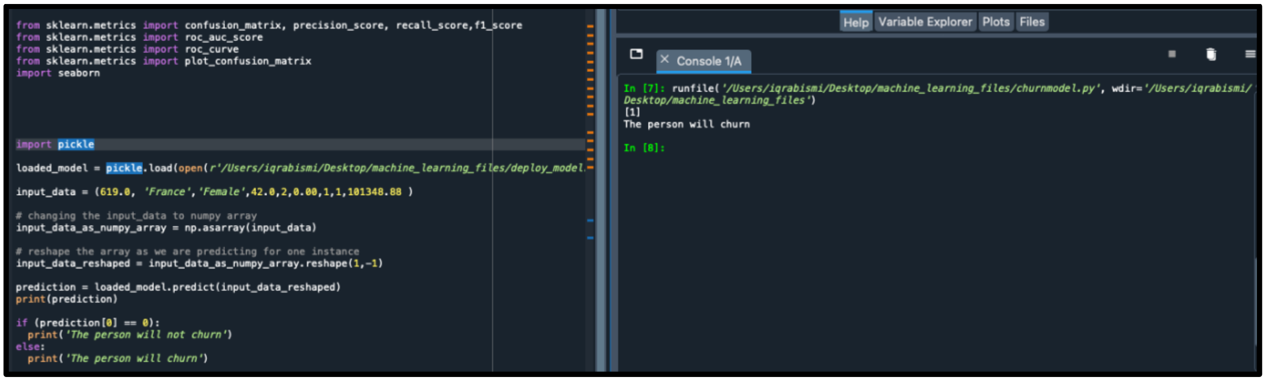


```
filename = 'deploy_model.sav'
pickle.dump(final_model, open(filename, 'wb'))
```

In the next phase, we have installed Spyder, an open source cross platform integrated development environment which is used to deploy projects such as creating an application etc. We also installed Streamlit to create web application for our machine learning project. After downloading the required software, we have uploaded the saved model in spyder environment as shown in figure 78 below.

Figure 78

Using Spyder to run the saved model



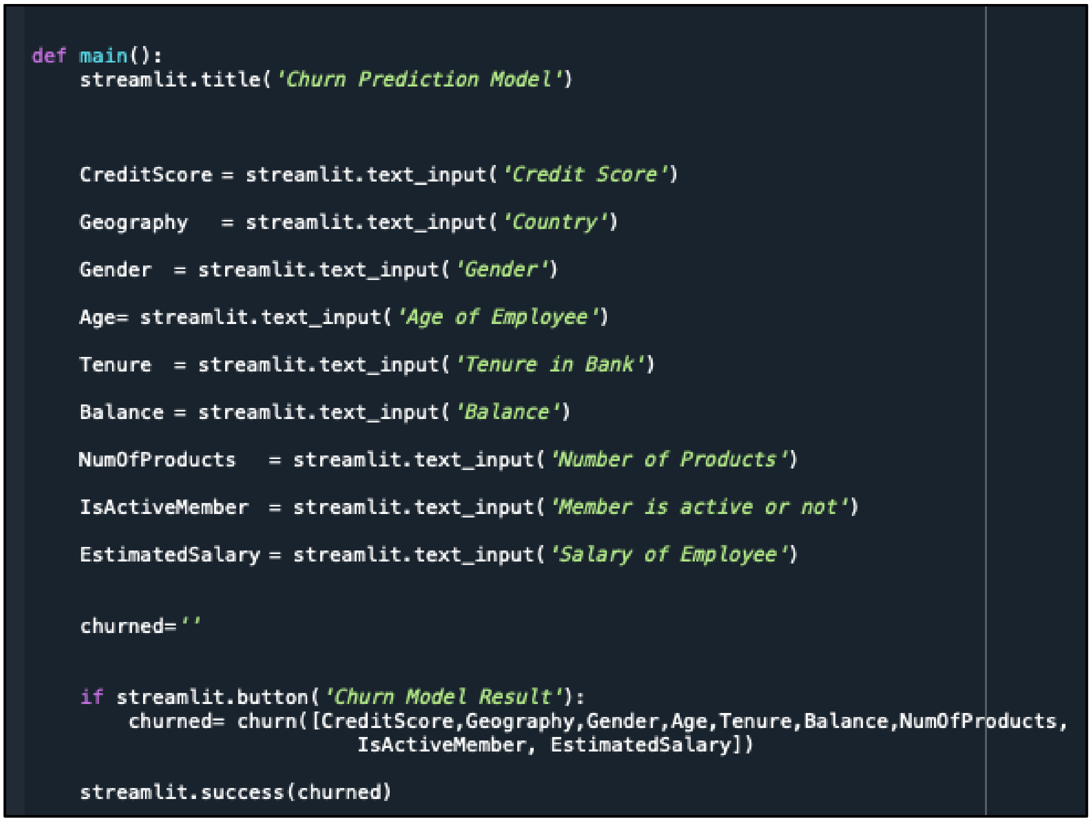
```
from sklearn.metrics import confusion_matrix, precision_score, recall_score,f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import plot_confusion_matrix
import seaborn



import pickle

loaded_model = pickle.load(open(r'/Users/iqrabismi/Desktop/machine_learning_files/deploy_model.

input_data = (619.0,  'France','Female',42.0,2,0.00,1,1,101348.88 )

# changing the input_data to numpy array
input_data_as_numpy_array = np.asarray(input_data)

# reshape the array as we are predicting for one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)

prediction = loaded_model.predict(input_data_reshaped)
print(prediction)

if (prediction[0] == 0):
  print('The person will not churn')
else:
  print('The person will churn')
```

```
In [7]: runfile('/Users/iqrabismi/Desktop/machine_learning_files/churnmodel.py', wdir='/Users/iqrabismi/
Desktop/machine_learning_files')
[1]
The person will churn

In [8]:
```

Figure 78 above shows that model is working in spyder and giving prediction results. Next step involves creating an application using Streamlit as shown in figure below 79.

Figure 79

Using streamlit to define buttons for web application

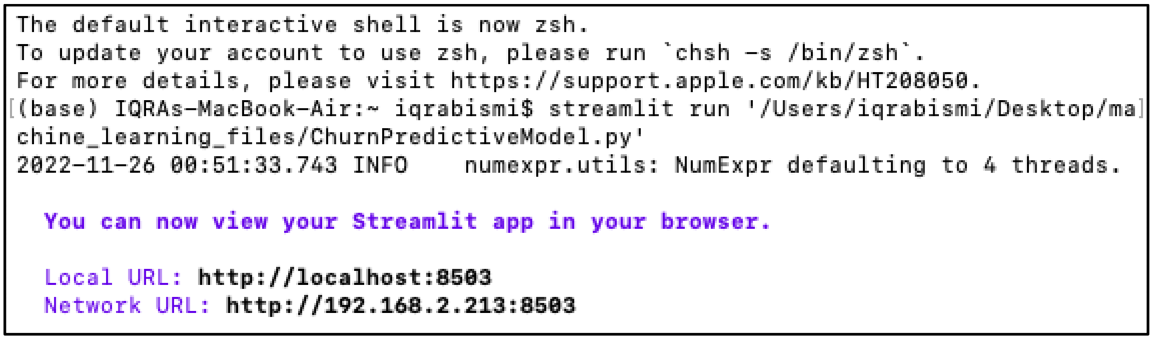


```
def main():
    streamlit.title('Churn Prediction Model')


    CreditScore = streamlit.text_input('Credit Score')

    Geography   = streamlit.text_input('Country')

    Gender  = streamlit.text_input('Gender')

    Age= streamlit.text_input('Age of Employee')

    Tenure  = streamlit.text_input('Tenure in Bank')

    Balance = streamlit.text_input('Balance')

    NumOfProducts   = streamlit.text_input('Number of Products')

    IsActiveMember  = streamlit.text_input('Member is active or not')

    EstimatedSalary = streamlit.text_input('Salary of Employee')


    churned=''


    if streamlit.button('Churn Model Result'):
        churned= churn([CreditScore,Geography,Gender,Age,Tenure,Balance,NumOfProducts,
                            IsActiveMember, EstimatedSalary])

    streamlit.success(churned)
```

Later we saved this file on local machine and ran the command 'streamlit run' in terminal with path of the saved file as shown in figure below 80.

Figure 80

Running streamlit on terminal to get url for web application

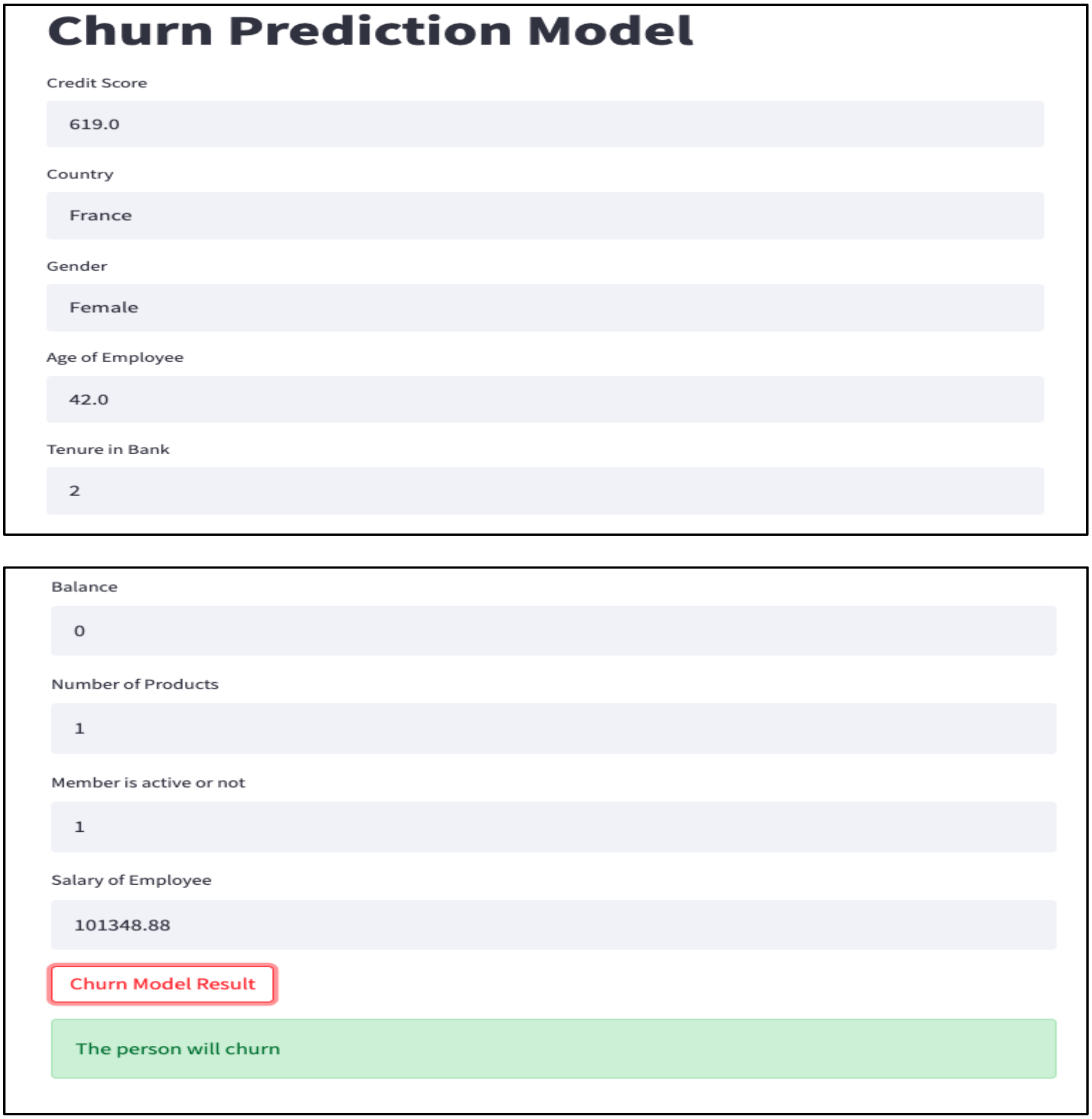


```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
[(base) IQRAs-MacBook-Air:~ iqrabismi$ streamlit run '/Users/iqrabismi/Desktop/ma]
chine_learning_files/ChurnPredictiveModel.py'
2022-11-26 00:51:33.743 INFO    numexpr.utils: NumExpr defaulting to 4 threads.

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8503
  Network URL: http://192.168.2.213:8503
```

This command will give the url as shown in figure above and a new website will be opened as shown below in figure 81.

Figure 81

Web application for churn prediction model



Churn Prediction Model

Credit Score

619.0

Country

France

Gender

Female

Age of Employee

42.0

Tenure in Bank

2

Balance

0

Number of Products

1

Member is active or not

1

Salary of Employee

101348.88

The person will churn

Hence, this application will help bank to know which customers might leave the bank and then accordingly they can provide incentives or offer to those customers which will increase the retention rate.

7 Conclusion and Future work

Through analyzing the F1 score of the built machine learning models, we conclude that the four best performing classifiers are Random Forest, Gradient Boosting, ADA Boost and XG Boost classifier. These models can be used for real time bank customer churn prediction to achieve accurate results of forecasting.

For Future research on Bank customer churn prediction, more number of features can be considered when training the model that can contribute on increasing the accuracy. Considering more features helps the bank to analyze the factors that leads the customer to leave the bank. Business and customer relations can be built through this method when the factors are analyzed. More number of customer records can be collected so that the amount of training data could also increase efficiency in a timely manner. The Bank customer churn prediction could further be expanded to choose different techniques and methods provided to the bank that can restrict the customers leaving them.

Appendix

Links to our Project Code

https://colab.research.google.com/drive/1unsFXj9wuRNpO3atLDAMcQJjRYb7UpMx

https://colab.research.google.com/drive/1imqCoUnIFYcfFDt5LbwJiGrGbeLyMcIi

References

Feng, L. (2022c). Research on Customer Churn Intelligent Prediction Model based on Borderline-SMOTE and Random Forest. *2022 IEEE 4th International Conference on Power, Intelligent Computing and Systems (ICPICS)*. https://doi.org/10.1109/icpics55264.2022.9873702

Sagala, N. T. M., & Permai, S. D. (2021b). Enhanced Churn Prediction Model with Boosted Trees Algorithms in The Banking Sector. *2021 International Conference on Data Science and Its Applications (ICoDSA)*. https://doi.org/10.1109/icodsa53588.2021.9617503

Shukla, A. (2021b). Application of Machine Learning and Statistics in Banking Customer Churn Prediction. *2021 8th International Conference on Smart Computing and Communications (ICSCC)*. https://doi.org/10.1109/icscc51209.2021.9528258

Kaur, I., & Kaur, J. (2020b). Customer Churn Analysis and Prediction in Banking Industry using Machine Learning. *2020 Sixth International Conference on Parallel, Distributed and Grid Computing (PDGC)*. https://doi.org/10.1109/pdgc50313.2020.9315761