Customer Intent Prediction Using Machine Learning

Group 20

Anirudh Gaur (1810110029)

Pradeep Kumar G (1810110166)

Prof. Madan Gopal

Abstract

In this project, we plan to predict real-time online shopper behavior analysis system i.e. whether the user will be purchasing something from an e-commerce site. There is a constant need for e-commerce sites to improve their model based on how much time user is spending while surfing the net and the period at which he is doing it. So, a need for such model arises which indicates when it is likely that user will make a transaction on the site. We have used six models for this, Naive Bayes, Logistic Regression, decision trees, random forest, XgBoost and neural network algorithm. We have also divided our data into various testing experiments to observe the changes upon standardizing, removing overfitting, and oversampling for minority class and computed there result later in a confusion matrix to obtain various quantities such as accuracy, success rate, sensitivity and other such parameters of the model finding out the best model for each experiment . The techniques and coding part has been done in Python and executed using Jupyter notebook. After performing the two models we have learnt so far, we have computed the result and suggested what changes are still needed in the dataset or our model to get better results in the final project evaluation.

Contents

Overview

This project is related to EED-363 Applied Machine Learning course. The present report starts with a general idea of the project and by representing its objectives.

Then the given dataset will be prepared and setup. An exploratory data analysis is carried out in order to develop a machine learning algorithm that could predict whether an e-commerce transaction has taken place or not until a final model. Results will be explained. Finally, the report will end with some concluding remarks.

Introduction

Purchase prediction has an important role for decision making in e-commerce to improve consumer experience, provide personalized recommendations and increase revenue. Many works investigated purchase prediction for session logs by analysis of users' behavior to predict purchase intention after a session has ended. In most cases, e-shoppers prefer to be anonymous while browsing the websites and after a session has ended, identifying users and offering discounts can be challenging. Therefore, after a session ends, predicting purchase intention may not be useful for the e-commerce strategists. In this work, we propose and develop an early purchase prediction framework using advanced machine learning models to investigate how early purchase intention in an ongoing session can be predicted. Many features such as bounce rate, exit rate and page values can be used to predict results. Other factors which also come into consideration are operating system, visitor type, traffic type etc. along with the time of week (weekend or weekdays) in which the visit to the site is being made and month of the year which could help of to further narrow down and give us best results. In this report, using three classification models; Naïve Bayes, K- Nearest Neighbors and Logistic Regression have been run on the dataset and the results obtained are then measured using various performance metrics to compare among the algorithms in order to find out the best suited model.

Aim of the Project

The objective of this report is to train machine learning models to predict whether customer will buy certain type of item for various e-commerce websites or not. Data will be converted from text to numerical values to create a more robust analysis. Optimal model will be selected following the resulting accuracy, sensitivity, specificity and other factors. We will use machine learning to extract various features which would be helpful to make a classifier with a high accuracy rate and low false negative and low false positive rate.

Dataset

This project covers Online shoppers purchasing intentions dataset (https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset) created by C. Okan Sakar, faculty of Engineering and natural sciences, Bahcesehir University and Yomi Kastro who works at Inveon Information Technologies Consultancy and Trade. The dataset consists of feature vectors belonging to 12,330 sessions and was formed in such a way that each session would belong to a different user in a 1-year period to avoid any tendency to a specific campaign, special day, user profile or period.

The dataset contains 10 numerical and 8 categorical attributes. They are specified below-

1) Administrative- No. of administrative pages visited in a session.
2) Administrative duration- Total time spent on administrative pages.
3) Informational- No. of informational pages visited in a session.
4) Informational duration- Total time spent on Informational pages.
5) Product related- No. of product related pages visited in a session.
6) Product related duration- Total time spent on product related pages.
7) Bounce rate- Percentage of visitors who enter the site and leave without triggering any request.
8) Exit rate- The percentage that a page of certain category was last in that session.
9) Page value- Average value of page that user visited before completing e-commerce transaction.
10) Special day- These are specific days (example Mother's Day) in which the sessions are likely to be finalized with transactions. Takes value between 0 to 1 if order date which is 10 days before the special day and 0 otherwise.
11) Month-Text to Numerical values from 1 for January to 12 for December.
12) Operating systems- Numerical values depending upon different operating systems.
13) Browser- Numerical values depending upon different types of browsers.
14) Region- Different numeric values depending upon region.
15) Traffic type- Numeric values for different online traffic.
16) Visitor type- Returning and new visitor.
17) Weekend- 0 if false and 1 if true.

The final feature is for revenue which is true if transaction has taken place and false otherwise.

The dataset had 12,330 instances out of which 84.5% samples are those ones where transactions did not take place and rest where transaction took place so we needed to select training and testing data accordingly.

Literature Review

Literature encompasses many studies which are turned towards categorization of online visits in e-commerce websites.

A first study of Mobasher et al. [2] assessed two different clustering techniques based on user transactions and pageviews in order to find out useful aggregate profiles that can be used by recommendation systems to achieve effective personalization at early stages of user's visits in an online store.

Later, the study of Moe [3] prepared the ground for a system which can take customized actions according to the category of a visit. For this reason, the author proposed a system which makes use of page-to-page clickstream data from a given online store in order to categorize visits as a buying, browsing, searching, or knowledge-building visit. The proposed system rests on observed in-store navigational patterns (including the general content of the pages viewed) and a k-means algorithm for clustering.

In [4], Poggi et al. proposed a system which handles the loss of throughput in Web servers due to overloading, by assigning priorities to sessions on an e-commerce website according to the revenue that will generate. Data were formed of clickstream and session information and Markov chains, logistic linear regression, decision trees and naïve Bayes were investigated in order to measure the probability of users' purchasing intention [4].

In [5] and [6], authors designed the prediction of purchasing intention problem as a supervised learning problem and historical data collected from an online bookstore were used to categorize the user sessions as browsing and buyer sessions. In this scope, Support vector machines (SVMs) with different kernel types and k-Nearest Neighbor (k-NN) were respectively investigated in [5] and [6] to carry out classification.

In a more recent study [7], Suchacka and Chodak constructed a new approach to analyze historical data obtained from a real online bookstore. The proposed approach is based on association rule discovery in customer sessions and aims to evaluate the purchase probability in an online session.

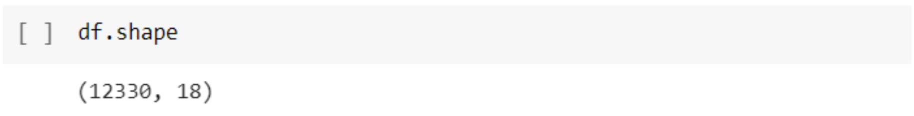
In [8], the author proposed a system that identifies the website component that has the highest business impact on visitors. To build such a system, a data set based on the Google Analytics tracking code [9] has been created. Moreover, naïve Bayes and multilayer perceptron classifiers have been explored for classification.

In [1], authors set up a real-time user behavior analysis system for virtual shopping environment which is made up of two modules. In the first module, the purchasing intention of the visitor is predicted using aggregated pageview data kept track during the visit along with some session and user information. Further, oversampling and feature selection preprocessing algorithms were applied to improve the effectiveness and the scalability of a set of supervised machine learning techniques. The highest accuracy of 87.24% and F1 Score of 0.86 were obtained with a Multilayer Perceptron Network (MLP). In the second module, authors used a Long Short-Term Memory-based Recurrent Neural Network (LSTM-RNN) based on sequential clickstream data to produce the probability estimate of visitor's intention to leave the site without completing the transaction. Within the scope of the entire system proposed in [1], the first module is triggered only if the second module generates a greater value than a predetermined threshold and the final objective consists in deciding whether to offer a content to the online visitor.

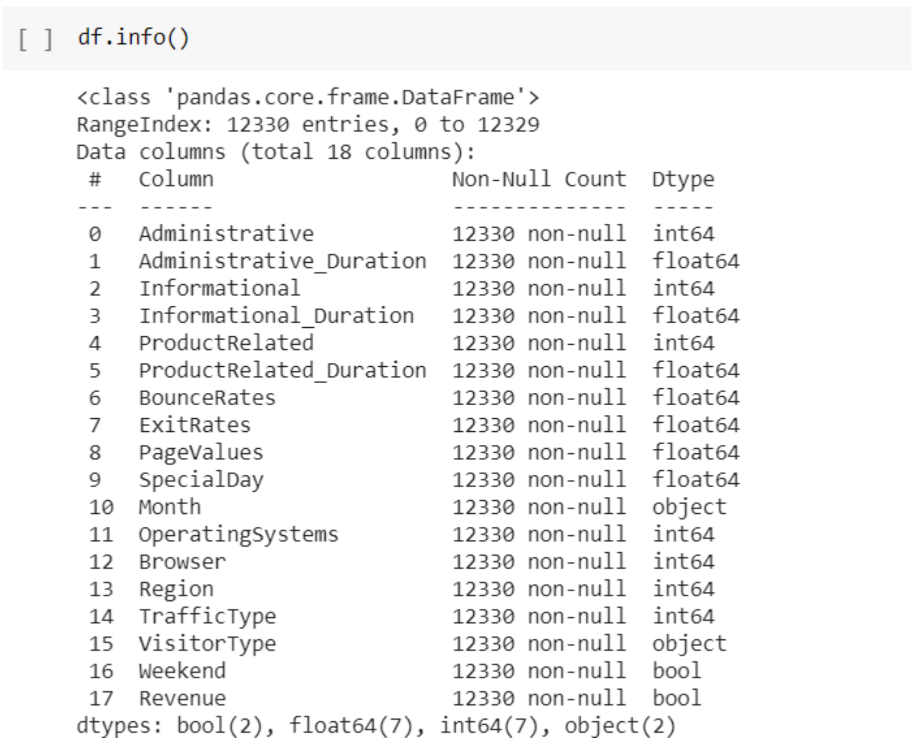
Data Analysis

Data Preprocessing

By observing the given dataset, we found that it contains 12330 observations and 18 variables.

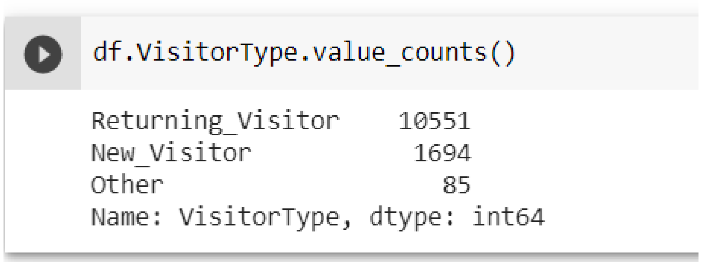


```
[ ] df.shape

    (12330, 18)
```



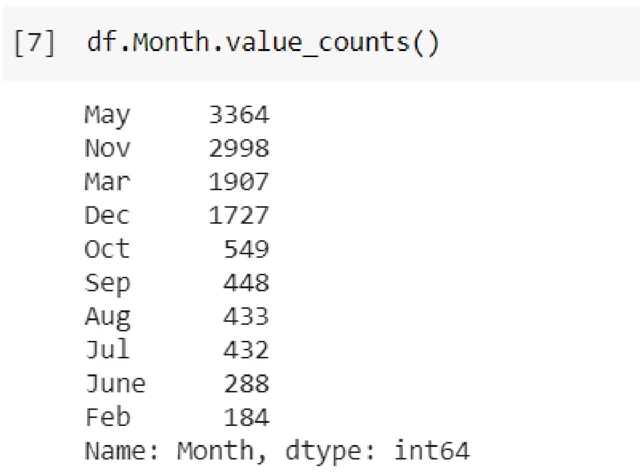
```
[ ] df.info()

    <class 'pandas.core.frame.DataFrame'>
    RangeIndex: 12330 entries, 0 to 12329
    Data columns (total 18 columns):
     #   Column                   Non-Null Count  Dtype  
    ---  ------                   --------------  -----  
     0   Administrative           12330 non-null  int64  
     1   Administrative_Duration  12330 non-null  float64
     2   Informational            12330 non-null  int64  
     3   Informational_Duration   12330 non-null  float64
     4   ProductRelated           12330 non-null  int64  
     5   ProductRelated_Duration  12330 non-null  float64
     6   BounceRates              12330 non-null  float64
     7   ExitRates                12330 non-null  float64
     8   PageValues               12330 non-null  float64
     9   SpecialDay               12330 non-null  float64
     10  Month                    12330 non-null  object 
     11  OperatingSystems         12330 non-null  int64  
     12  Browser                  12330 non-null  int64  
     13  Region                   12330 non-null  int64  
     14  TrafficType              12330 non-null  int64  
     15  VisitorType              12330 non-null  object 
     16  Weekend                  12330 non-null  bool   
     17  Revenue                  12330 non-null  bool   
    dtypes: bool(2), float64(7), int64(7), object(2)
```

The above diagram indicates data types of different features in our dataset and they are mostly float and 'int64' datatypes. Only datatype for 'Month' and 'Visitor Type' are 'objects' which represent the month of the session and whether the customer is new or already existing one respectively.



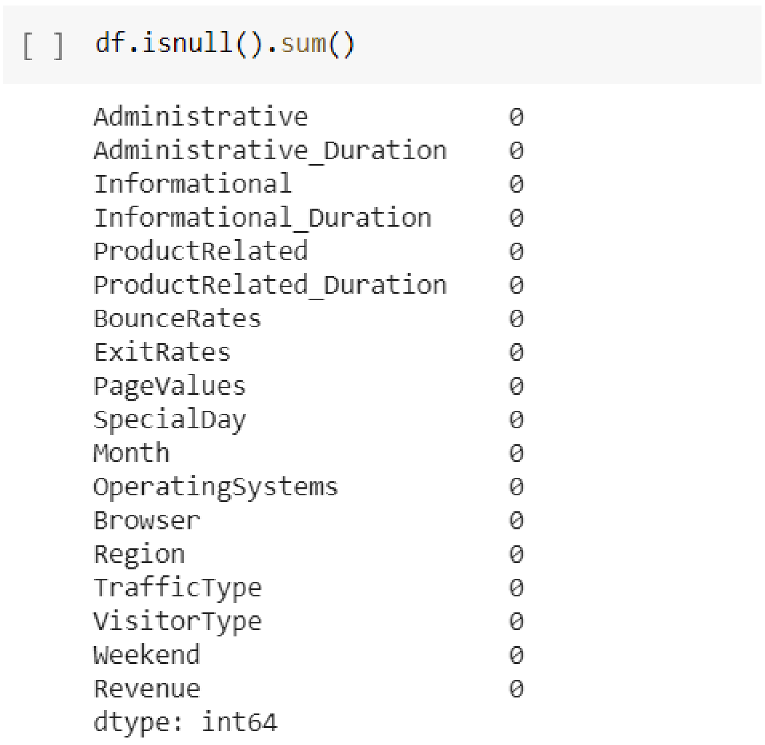
```
df.VisitorType.value_counts()

Returning_Visitor    10551
New_Visitor           1694
Other                   85
Name: VisitorType, dtype: int64
```

We find that there is also a third visitor type given by 'Other' whose number of instances in data is 85. Since the number of instances for 'Other' is very less, we have decided to drop those instacnes.



```
[7]  df.Month.value_counts()

     May     3364
     Nov     2998
     Mar     1907
     Dec     1727
     Oct      549
     Sep      448
     Aug      433
     Jul      432
     June     288
     Feb      184
     Name: Month, dtype: int64
```

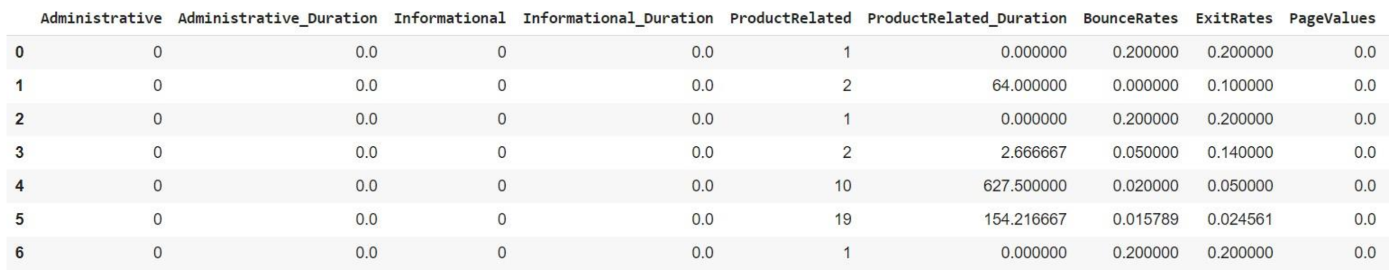
Also in our data there are no null values found after running a can through a function across it.



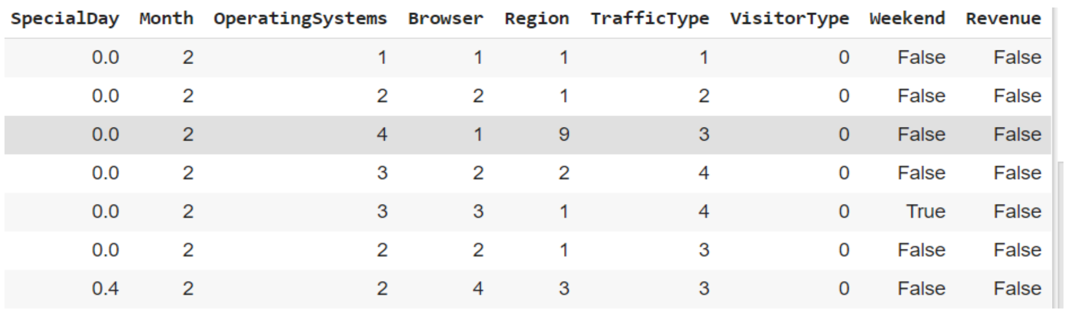
```
[ ]  df.isnull().sum()

     Administrative             0
     Administrative_Duration    0
     Informational              0
     Informational_Duration     0
     ProductRelated             0
     ProductRelated_Duration    0
     BounceRates                0
     ExitRates                  0
     PageValues                 0
     SpecialDay                 0
     Month                      0
     OperatingSystems           0
     Browser                    0
     Region                     0
     TrafficType                0
     VisitorType                0
     Weekend                    0
     Revenue                    0
     dtype: int64
```

NO Null Values

The column 'Month' of the dataset consists of values ranging from 1-12 according to respective months and 'VisitorType' has been classified into 0 for 'Returning_Visitor', 1 for 'New_Visitor', and 2 for 'Other'.



	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0
1	0	0.0	0	0.0	2	64.000000	0.000000	0.100000	0.0
2	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0
3	0	0.0	0	0.0	2	2.666667	0.050000	0.140000	0.0
4	0	0.0	0	0.0	10	627.500000	0.020000	0.050000	0.0
5	0	0.0	0	0.0	19	154.216667	0.015789	0.024561	0.0
6	0	0.0	0	0.0	1	0.000000	0.200000	0.200000	0.0

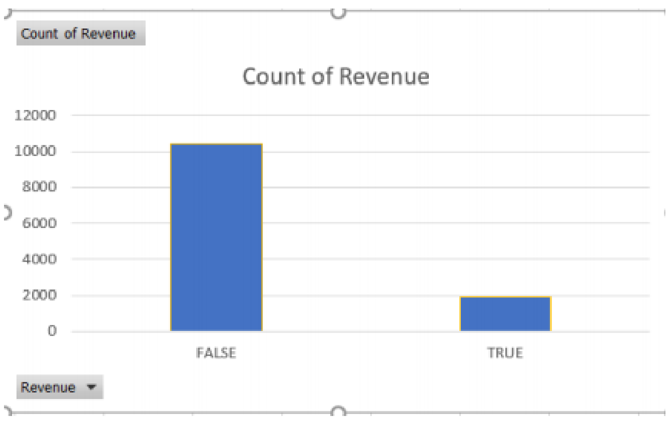


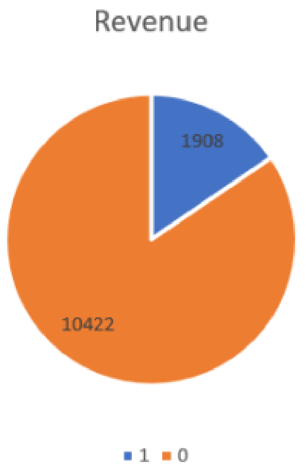
SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0.0	2	1	1	1	1	0	False	False
0.0	2	2	2	1	2	0	False	False
0.0	2	4	1	9	3	0	False	False
0.0	2	3	2	2	4	0	False	False
0.0	2	3	3	1	4	0	True	False
0.0	2	2	2	1	3	0	False	False
0.4	2	2	4	3	3	0	False	False

Exploratory Data analysis

In this portion of our report, we would be focusing on investigating our data further with the help of statistical and graphical approach to find certain patterns and verify our hypothesis. It is a good practice as we will be able to understand our data further before applying various algorithms on it.

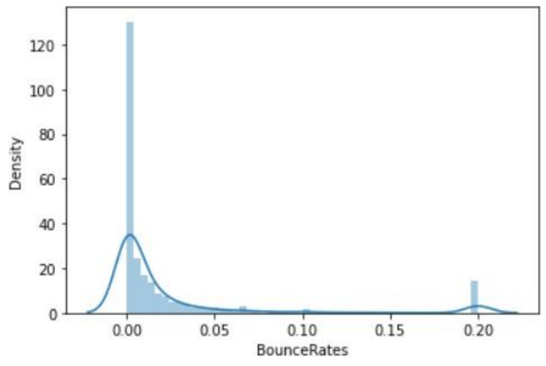
We start with a pie chart and a plot depicting ‘Revenue’ feature which is responsible for showing if the transaction has taken place or not which is described by Booleans true and false respectively.

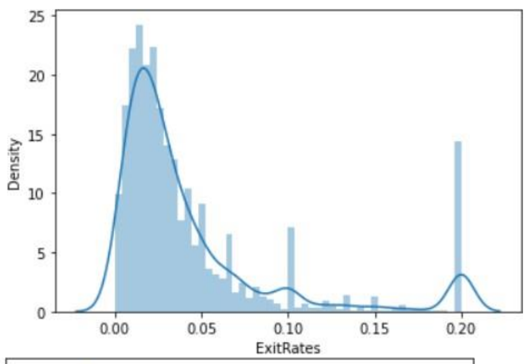


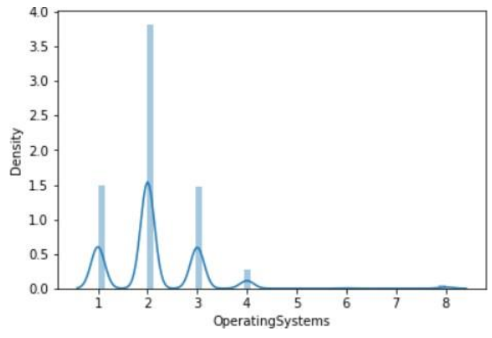


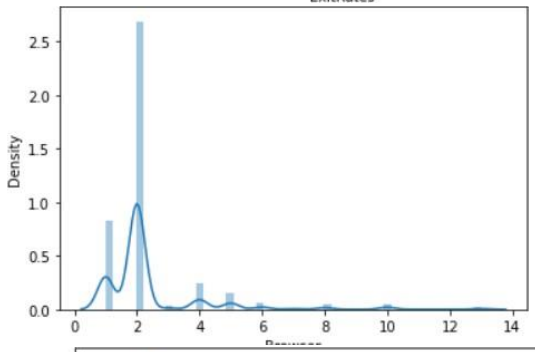
This dataset has 12,330 instances and 18 features. It can be clearly observed from this that 10422(84.52%) out of total transactions failed to generate any revenue and only 1908(15.47%) were successful.

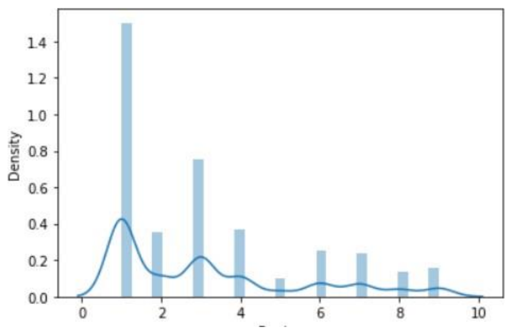
It is also necessary to see how our data is distributed and what is its overall density for a particular feature. This can be further used to calculate probability for individual observations in our dataset. We will be using a distribution plot which can be later helpful for calculating likelihood of observations.

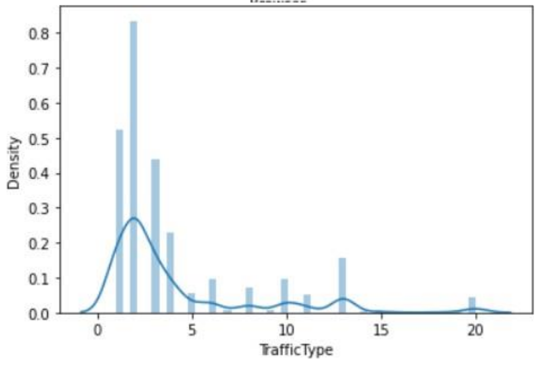










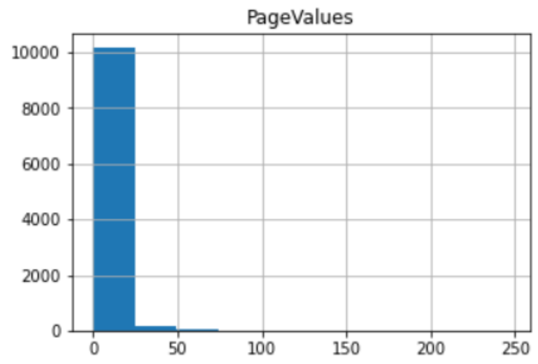


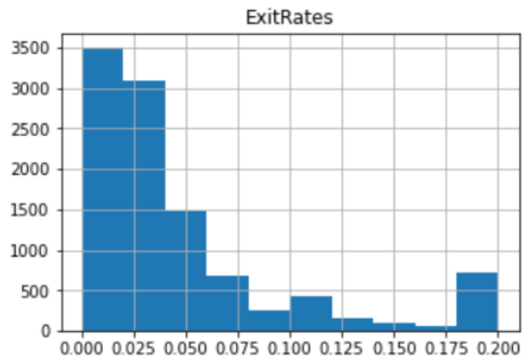
We have not shown the distribution plots for other features such as category of pages because they have high density function at 0 and very minimal density at other points.

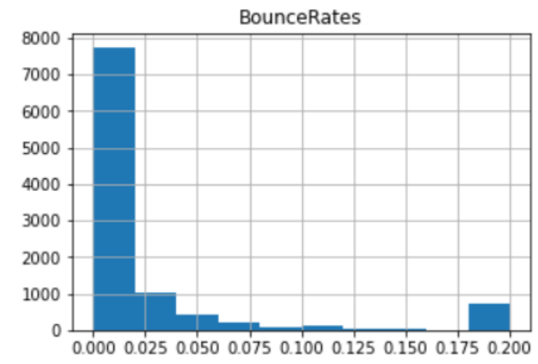
There is a need to evaluate the strength of relationship between two quantitative variables. This will be done be correlation analysis which uses a correlation matrix to give is a plot. In this plot, the points of high correlation indicate that the variables have strong relationship between them.

Distribution of PageValues, Exitrates, Bouncerates with respect to revenue-

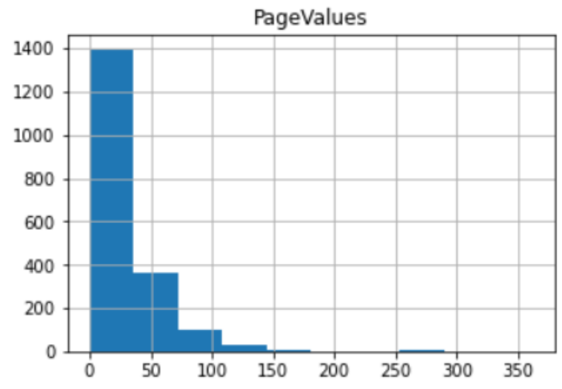
For “Revenue”=0

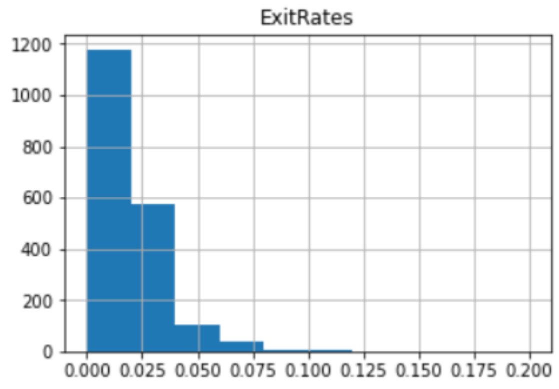


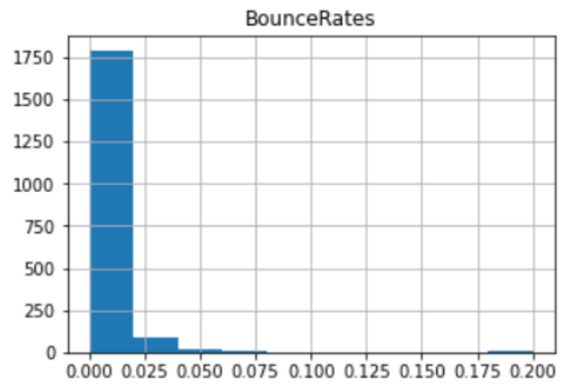




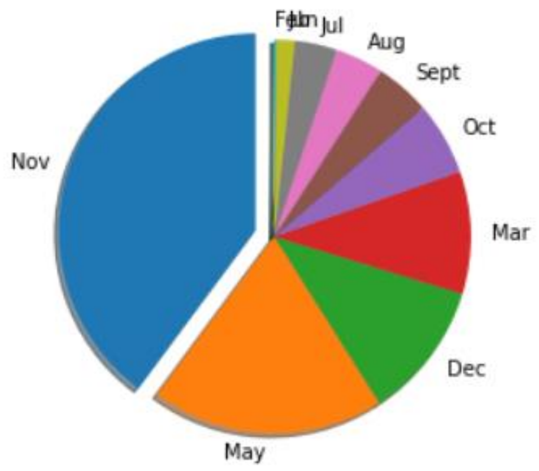
For “Revenue”=1



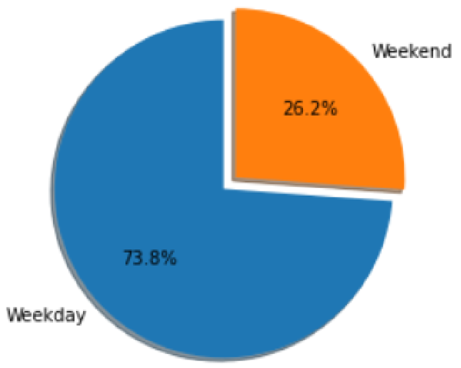




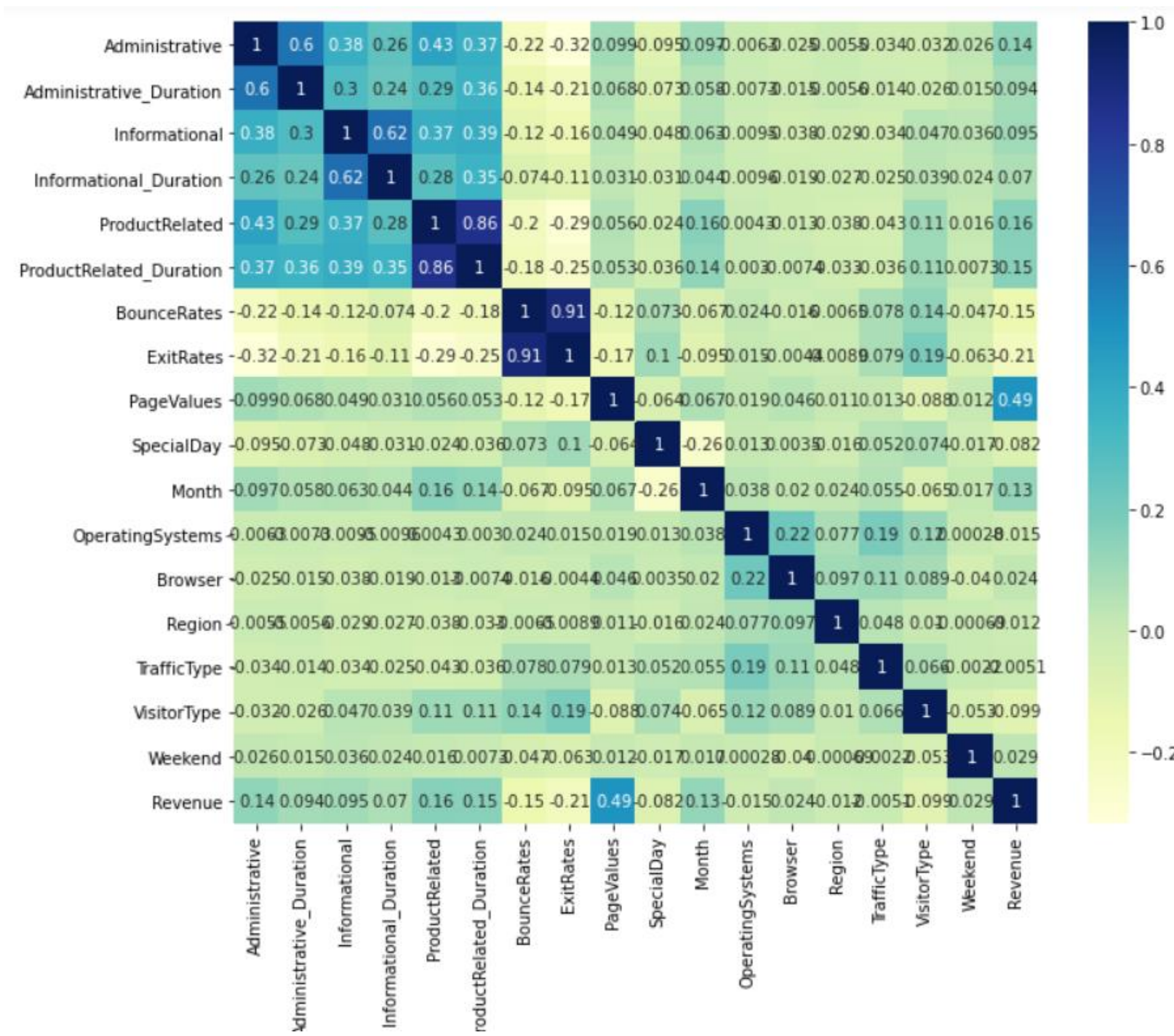
Below is a pie chart to see in which month majority of transactions that took place generated revenue-



Below is a pie chart showing if the successful transaction happened during weekend or weekdays-



From these 2 pie charts, we can clearly observe that majority of the successful transactions have taken place during the month of November. Also majority of the transfers have taken place during weekdays instead of weekends which was unexpected. But these are the information only for those instances where “Revenue” has been generated and not for all instances.



From this correlation plot it can be concluded that even though most of the variables have low correlation with revenue, Page value has the highest correlation with revenue.

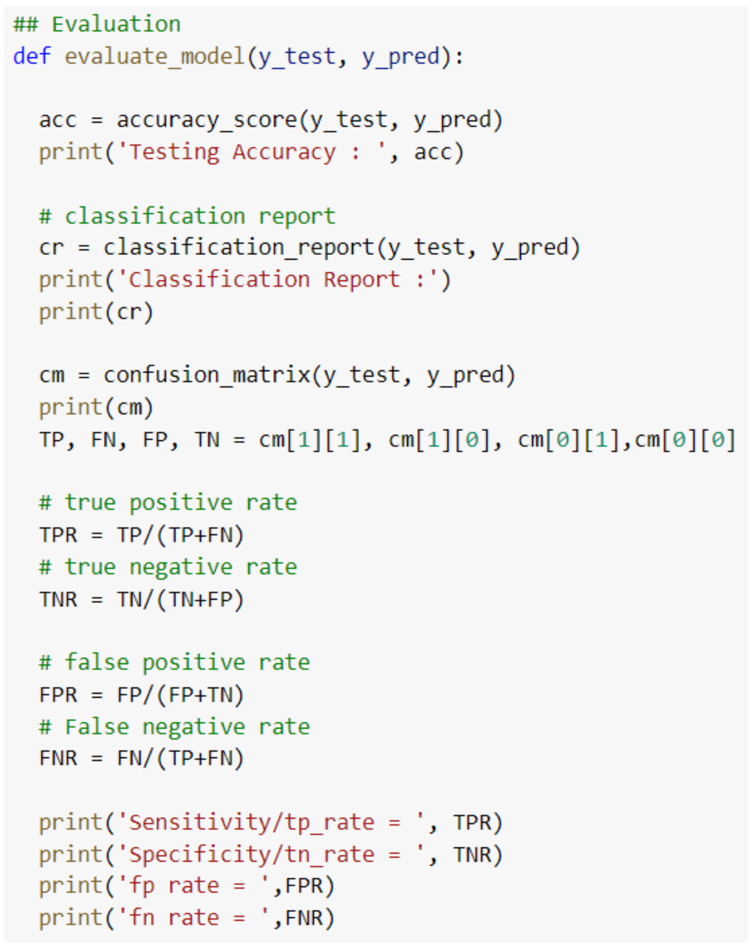
We can conclude from this that most of the features in our data set are independent of each other which can be indicated by the low correlation value they have.

Modeling

Training and Test Data

In machine learning, data needs to be separated into 2 sections- training data and testing data (in case of holdout method and sometimes in 3 sections including validation). This is done in order to fit our model according to training data so that we can test its results on testing data. Test data provides us with ideal standards and is used once the training of model is complete. The splitting of data should be done according to certain guidelines and techniques such that training data is more than testing data (thumb rule is 70% training and 30% testing). Also, some models need large data for training so that their optimization is better. Models with very few features may also require lesser data set but that with high number of features requires a larger one so that a validation section splitting is also possible. In our project, we have decided to split dataset in 80%-20% ratio for training and testing respectively (first 9796 instances for training and next 2449 instances for testing).

Below is the code which we will used to evaluate the performance of our model. We would be calculating various criteria such as accuracy, misclassification error, f1 score, etc. for different models in order to analyze which model is best. These criteria would be explicitly defined by us in the later part of the report for better understanding.



```
## Evaluation
def evaluate_model(y_test, y_pred):

  acc = accuracy_score(y_test, y_pred)
  print('Testing Accuracy : ', acc)

  # classification report
  cr = classification_report(y_test, y_pred)
  print('Classification Report :')
  print(cr)

  cm = confusion_matrix(y_test, y_pred)
  print(cm)
  TP, FN, FP, TN = cm[1][1], cm[1][0], cm[0][1],cm[0][0]

  # true positive rate
  TPR = TP/(TP+FN)
  # true negative rate
  TNR = TN/(TN+FP)

  # false positive rate
  FPR = FP/(FP+TN)
  # False negative rate
  FNR = FN/(TP+FN)

  print('Sensitivity/tp_rate = ', TPR)
  print('Specificity/tn_rate = ', TNR)
  print('fp rate = ',FPR)
  print('fn rate = ',FNR)
```

```
# confusion matrix

print('Confusion Matrix :')

plt.rcParams['figure.figsize'] = (6, 6)
sns.heatmap(cm ,annot = True)

return TPR, TNR, FPR, FNR
```

Naive Bayes classifier

In this, we assume that the dataset is independent of every alternative variable such that all options can contribute towards the target data in the model. Knowledge of Bayes theorem is important which uses contingent probability that successively uses previous information to calculate probability of future events. Even though we assume data to be iid (independently identical data), in a real dataset the features are dependent on one another leading to inaccurate outcomes from naive bayes.

The formula for Naive Bayes theorem is-

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

In this formula, P(A|B) is the posterior probability which means probability that C is true given A. P(A) is prior probability, P(B) is the probability of evidence and P(B|A) is probability of the evidence when hypothesis is true. In our project, we are using this for binary classification whether revenue has been generated.

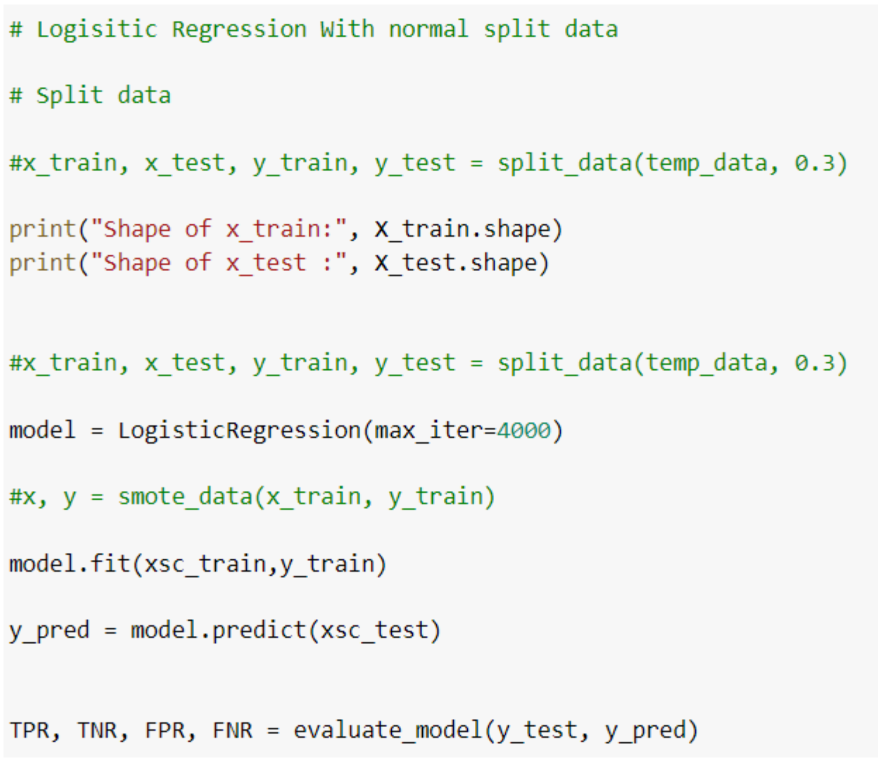
```
gnb = GaussianNB()

#x, y = smote_data(X_train, y_train)

y_pred = gnb.fit(xsc_train, y_train).predict(xsc_test)



TPR, TNR, FPR, FNR = evaluate_model(y_test, y_pred)
```

Logistic Regression Model

Logistic Regression is a supervised machine learning technique, employed in classification jobs (for predictions based on training data). Logistic Regression uses an equation similar to Linear Regression but the outcome of logistic regression is a categorical variable whereas it is a value for other regression models. Binary outcomes can be predicted from the independent variables. The outcome of dependent variable is discrete. Logistic Regression uses a simple equation which shows the linear relation between the independent variables. These independent variables along with their coefficients are united linearly to form a linear equation that is used to predict the output. This algorithm is entitled as logistic regression as the key method behind it is logistic function. The output can be predicted from the independent variables, which form a linear equation.

The output predicted has no restrictions, it can be any value from negative infinity to positive infinity. But the output required is a class variable (i.e., yes or no, 1 or 0). So, the outcome of the linear equation should be flattened into a small range (i.e [0,1]). Logistic function is used here to suppress the outcome value between 0 and 1. Logistic function can also be called cost function. Logistic regression measures the link between the variable quantity, the output, and therefore the freelance variables, the input.

One of the observations we studied from logistic regression is that it gives better results when data is normalized and the features are independent from one another. In our dataset, we earlier showed that most of the features have low correlation values whether in negative or positive depicting low dependency on each other. That is why we have used this model in our report.



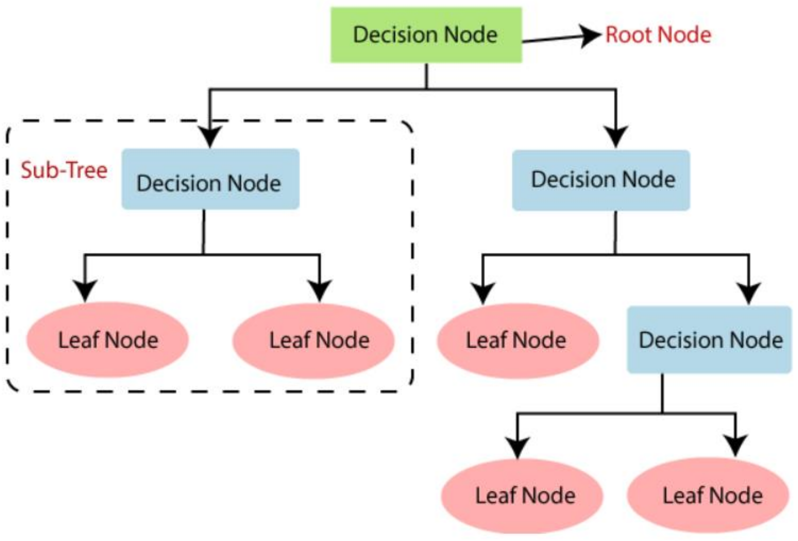
```
# Logisitic Regression With normal split data

# Split data

#x_train, x_test, y_train, y_test = split_data(temp_data, 0.3)

print("Shape of x_train:", X_train.shape)
print("Shape of x_test :", X_test.shape)


#x_train, x_test, y_train, y_test = split_data(temp_data, 0.3)

model = LogisticRegression(max_iter=4000)

#x, y = smote_data(x_train, y_train)

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)


TPR, TNR, FPR, FNR = evaluate_model(y_test, y_pred)
```

Decision Tree Model

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

We will be now explaining how this algorithm works in a brief-

In a decision tree, for predicting the class of the given dataset, the algorithm starts from the root node of the tree. This algorithm compares the values of root attribute with the record (real dataset) attribute and, based on the comparison, follows the branch and jumps to the next node. For the next node, the algorithm again compares the attribute value with the other sub-nodes and move further. It continues the process until it reaches the leaf node of the tree.



While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

1) Information Gain- The measurement of changes in entropy (Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data.) after the segmentation of a dataset based on an attribute. It calculates how much information a feature provides us about a class. According to the value of information gain, we split the node and build the decision tree. A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

```
Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)
```

2) Gini Index- Gini index is a measure of impurity or purity used while creating a decision tree in the CART(Classification and Regression Tree) algorithm. An attribute with the low Gini index should be preferred as compared to the high Gini index. It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits. Gini index can be calculated using the below formula:

$$\text{Gini Index}= 1- \sum_j P_j^2$$

```
## Decision Tree With Normal Data

from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
model.fit(xsc_train, y_train)

y_pred = model.predict(xsc_test)

print('Training Accuracy : ', model.score(xsc_train, y_train))

evaluate_model(y_test, y_pred)
```

Random Forest Model

Random Forest (RF) is a well-known decision tree ensemble that is commonly used in classification. In many cases, RF showed a good performance that outstrips that of many other classification algorithms.
As a decision tree ensemble, RF needs to construct several different decision trees. In order to accomplish that, each tree is set up considering a bootstrap sample set of the original training data. That consists in creating a new set sampling with replacement instances from the original set until getting the size of that original training data. Using one of the bootstrap sample sets of the training data, a random tree is obtained. In order to favor the diversity of the ensemble, each random tree follows a traditional top-down induction procedure with several modifications. At each step, when the best attribute is chosen, only a small subset of attributes from the dataset is considered. Considering only the subset of attributes chosen, we then compute the best attribute as in Classification And Regression Trees (CART). Each tree is built to its maximum depth and no pruning procedure is applied after the tree has been fully built. Lastly, to compute the predicted class for a sample, the predictions of the ensemble of decision trees are aggregated through majority voting.

Difference between random forest and decision trees-

While random forest is a collection of decision trees, there are some differences.
If you input a training dataset with features and labels into a decision tree, it will formulate some set of rules, which will be used to make the predictions.
For example, to predict whether a person will click on an online advertisement, you might collect the ads the person clicked on in the past and some features that describe his/her decision. If you put the features and labels into a decision tree, it will generate some rules that help predict whether the advertisement will be clicked or not. In comparison, the random forest algorithm randomly selects observations and features to build several decision trees and then averages the results.
Another difference is "deep" decision trees might suffer from overfitting. Most of the time, random forest prevents this by creating random subsets of the features and building smaller trees using those subsets. Afterwards, it combines the subtrees. It's important to note this doesn’t work every time and it also makes the computation slower, depending on how many trees the random forest builds.

```
# RandomForest with normal train test data


#X_train, X_test, y_train, y_test = train_test(temp_data, 0.3)

print("Shape of x_train:", X_train.shape)
print("Shape of x_test :", X_test.shape)


model = RandomForestClassifier()

model.fit(xsc_train, y_train)

y_pred = model.predict(xsc_test)

print('Testing Accuracy : ', model.score(xsc_train, y_train))

evaluate_model(y_test, y_pred)
```

XgBoost Model

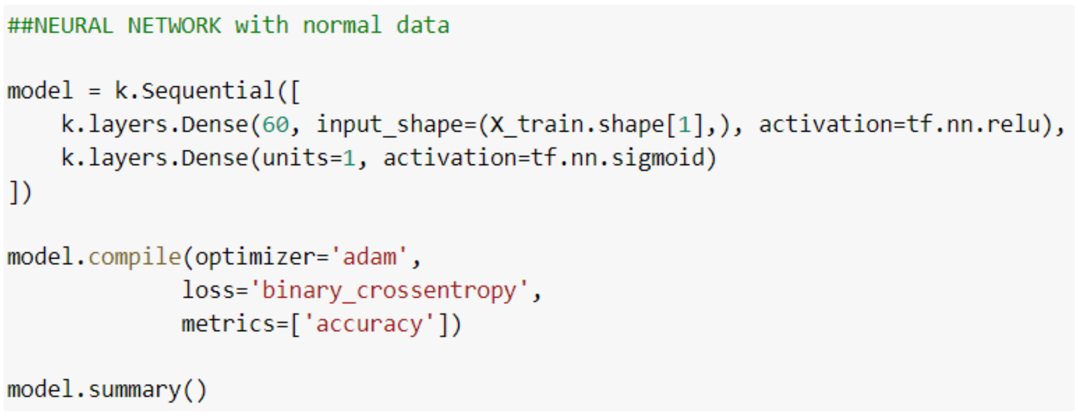
XgBoost is a tree based model which applies a boosting technique which works by successively adding a regularization term to the loss function in a step-wise manner to previous ensembles of weak estimators and then optimizing this loss function, in the hope that this would improve the performance of the model.

This is a self learning model implemented by us based in the research papers we have read and is generally used for binary classification problems.

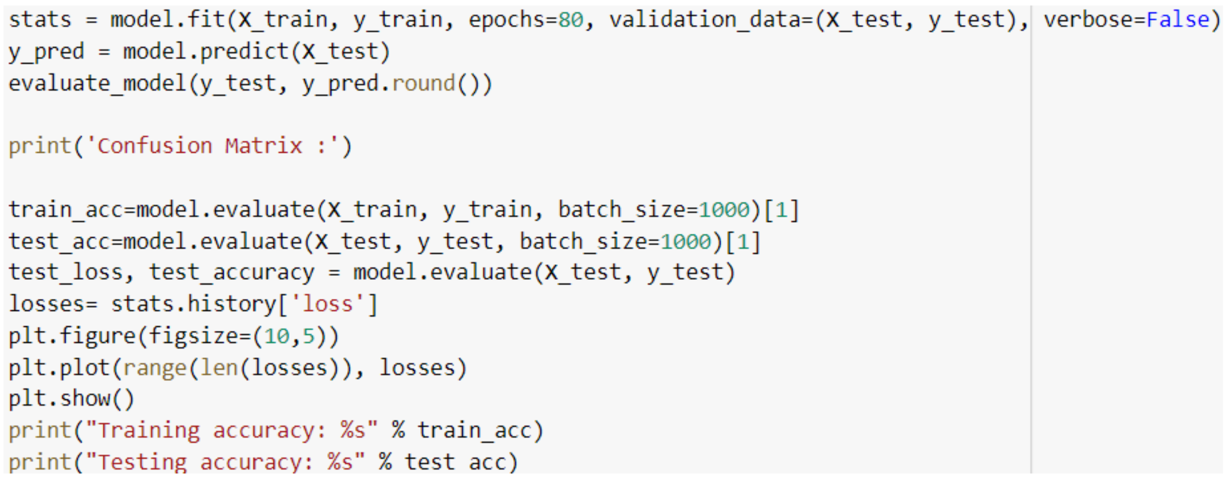
```
## XGBoost with Normal Data
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

model = XGBClassifier()

model.fit(xsc_train, y_train)

print('Training Accuracy : ', model.score(xsc_train, y_train))

y_pred = model.predict(xsc_test)
evaluate_model(y_test, y_pred)
```

Neural Network Model

Artificial Neural Networks (NN) are a type of mathematical algorithms originating in the simulation of networks of biological neurons. An artificial Neural Network consists of nodes (called neurons) and edges (called synapses). Input data is transmitted through the weighted synapses to the neurons where calculations are processed and then either sent to further neurons or represent the output. Neural Networks take in the weights of connections between neurons. The weights are balanced, learning data point in the wake of learning data point. When all weights are trained, the neural network can be utilized to predict the class or a quantity, if there should arise an occurrence of regression of a new input data point. With Neural networks, extremely complex models can be trained and they can be utilized as a kind of black box, without playing out an unpredictable complex feature engineering before training the model. Joined with the “deep approach” even more unpredictable models can be picked up to realize new possibilities. Neural networks normally have two at least two layer of neurons, with first layer neurons having nonlinear and differentiable activation functions. Such networks can approximate any non- linear function. In real life, we are faced with nonlinear problems, and multilayer neural network structures have the capability of providing solutions to these problems. Neural Networks possessing several layers are known as Multi-Layer Perceptron (MLP); their computing power is meaningfully improved over the one layer NN. A typical neural network consist of an input layer, an output layer and hidden layers. A Multi-Layer Perceptron (MLP) can also be regarded as a feedforward neural network with one or more hidden layers. Each hidden layer has its own specific function. Input terminals accept input signal from outside world and redistribute these signals to all the neurons in a hidden layer. The output layer accepts a stimulus pattern from a hidden layer and establishes the output pattern on the entire network. Neurons in the hidden layers perform transformations of input attributes; the weights of the neurons represent the features in the transformed domain. These features are then used by the output layer in determining the output pattern.



```
##NEURAL NETWORK with normal data

model = k.Sequential([
    k.layers.Dense(60, input_shape=(X_train.shape[1],), activation=tf.nn.relu),
    k.layers.Dense(units=1, activation=tf.nn.sigmoid)
])

model.compile(optimizer='adam',
             loss='binary_crossentropy',
             metrics=['accuracy'])

model.summary()
```



```
stats = model.fit(X_train, y_train, epochs=80, validation_data=(X_test, y_test), verbose=False)
y_pred = model.predict(X_test)
evaluate_model(y_test, y_pred.round())

print('Confusion Matrix :')

train_acc=model.evaluate(X_train, y_train, batch_size=1000)[1]
test_acc=model.evaluate(X_test, y_test, batch_size=1000)[1]
test_loss, test_accuracy = model.evaluate(X_test, y_test)
losses= stats.history['loss']
plt.figure(figsize=(10,5))
plt.plot(range(len(losses)), losses)
plt.show()
print("Training accuracy: %s" % train_acc)
print("Testing accuracy: %s" % test_acc)
```

Here, Epochs/Maximum Iterations is defined as the number of times the algorithm sees the entire data set. One forward pass and one backward pass of all the training examples is defined as an epoch.

Splitting into various experiments-

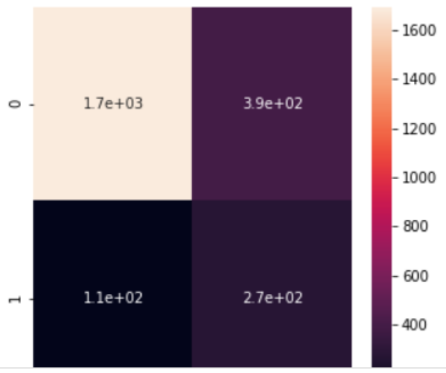
Considering the vast difference between the ranges and values of data for different instances, we have decided to make certain modifications into the testing data in order to get better results from our model. These modifications would be divided into various experiments and performance of all the models which we have defined above will be tested in each experiment. At the ends certain inferences would be drawn out.

Experiment 1- Normalized Data

Normalization or standardization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. We first defined a Standardscalar and then used fit_transform to normalize our data.

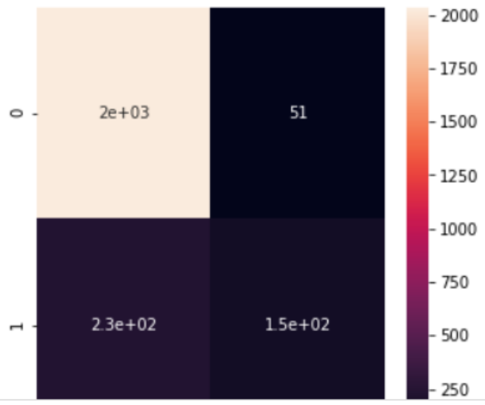
(i) Naïve Bayes Model

```
Testing Accuracy :  0.7964314679643146
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.81      0.87      2084
           1       0.41      0.71      0.52       382

    accuracy                           0.80      2466
   macro avg       0.67      0.76      0.70      2466
weighted avg       0.86      0.80      0.82      2466

[[1692  392]
 [ 110  272]]
Sensitivity/tp_rate =  0.7120418848167539
Specificity/tn_rate =  0.8119001919385797
fp rate =  0.18809980806142035
fn rate =  0.2879581151832461
Success rate = 0.7964314679643146
Misclassification rate = 0.2035685320356853
Confusion Matrix :
```



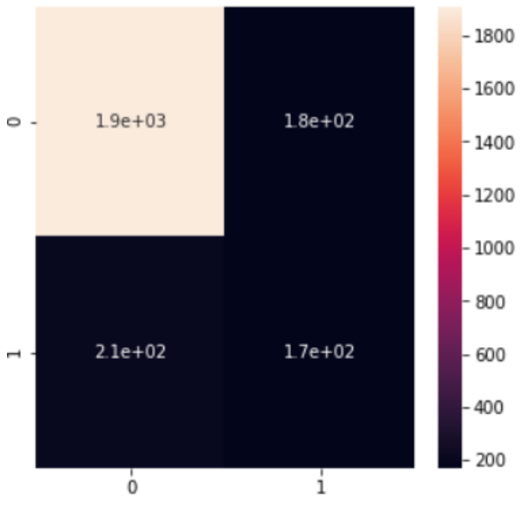
(ii) Logistic Regression Model

```
Testing Accuracy :  0.8856447688564477
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      2084
           1       0.75      0.40      0.52       382

    accuracy                           0.89      2466
   macro avg       0.82      0.69      0.73      2466
weighted avg       0.87      0.89      0.87      2466

[[2033   51]
 [ 231  151]]
Sensitivity/tp_rate =  0.39528795811518325
Specificity/tn_rate =  0.97552783109405
fp rate =  0.024472168905950095
fn rate =  0.6047120418848168
Success rate = 0.8856447688564477
Misclassification rate = 0.11435523114355231
Confusion Matrix :
```



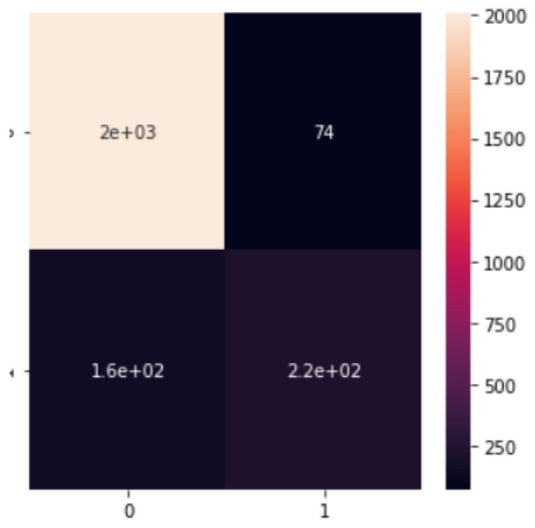
(iii) Decision Tree Model

```
Training Accuracy :  1.0
Testing Accuracy :  0.8434712084347121
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.92      0.91      2084
           1       0.49      0.45      0.47       382

    accuracy                           0.84      2466
   macro avg       0.70      0.68      0.69      2466
weighted avg       0.84      0.84      0.84      2466

[[1910  174]
 [ 212  170]]
Sensitivity/tp_rate =  0.44502617801047123
Specificity/tn_rate =  0.9165067178502879
fp rate =  0.08349328214971209
fn rate =  0.5549738219895288
Success rate = 0.8434712084347121
Misclassification rate = 0.15652879156528793
Confusion Matrix :
(0.44502617801047123,
 0.9165067178502879,
 0.08349328214971209,
 0.5549738219895288)
```



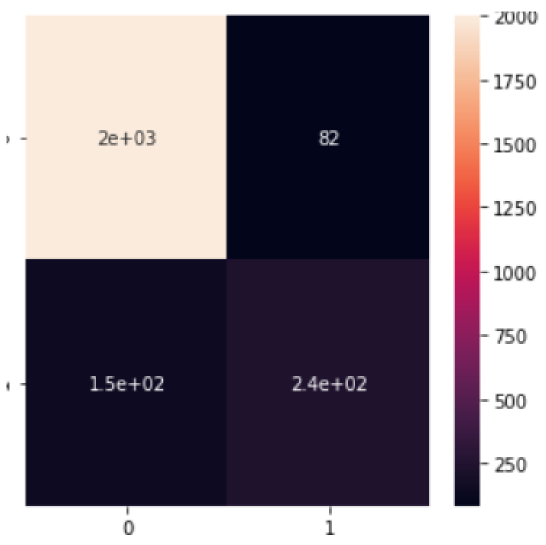
(iv) Random Forest Model

```
Testing Accuracy :  0.9042984590429846
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      2084
           1       0.75      0.58      0.65       382

    accuracy                           0.90      2466
   macro avg       0.84      0.77      0.80      2466
weighted avg       0.90      0.90      0.90      2466

[[2010   74]
 [ 162  220]]
Sensitivity/tp_rate =  0.5759162303664922
Specificity/tn_rate =  0.9644913627639156
fp rate =  0.03550863723608445
fn rate =  0.42408376963350786
Success rate = 0.9042984590429846
Misclassification rate = 0.09570154095701541
Confusion Matrix :
(0.5759162303664922,
 0.9644913627639156,
 0.03550863723608445,
 0.42408376963350786)
```



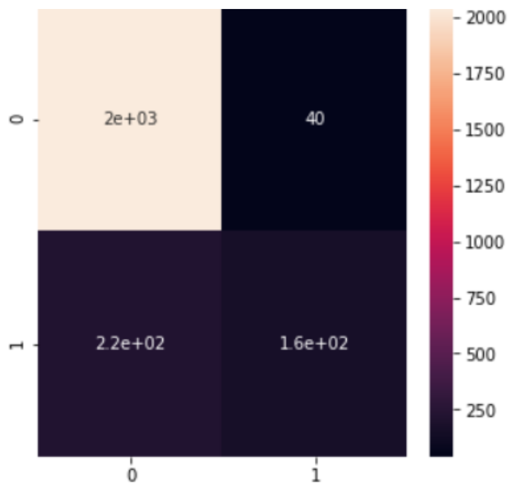
(v) XgBoost Model

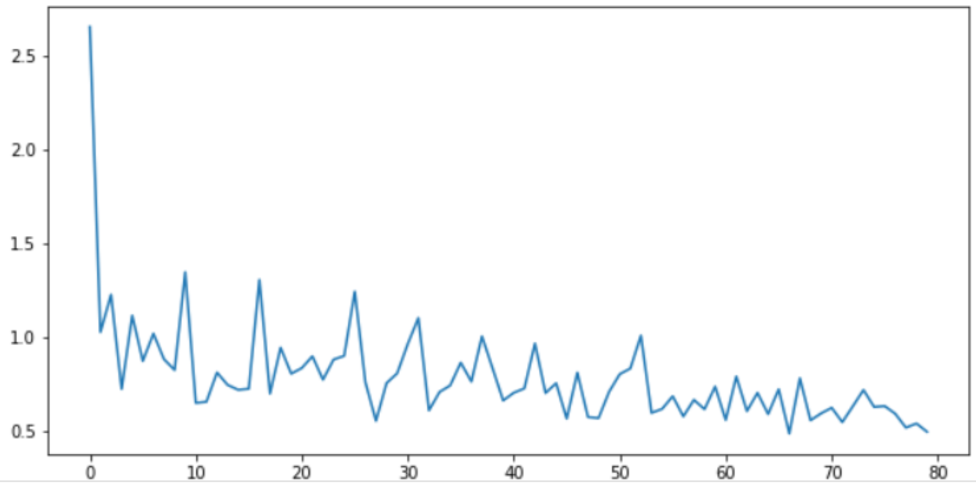
```
Testing Accuracy :  0.9075425790754258
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      2084
           1       0.74      0.62      0.67       382

    accuracy                           0.91      2466
   macro avg       0.84      0.79      0.81      2466
weighted avg       0.90      0.91      0.90      2466

[[2002   82]
 [ 146  236]]
Sensitivity/tp_rate =  0.6178010471204188
Specificity/tn_rate =  0.9606525911708254
fp rate =  0.03934740882917467
fn rate =  0.38219895287958117
Success rate = 0.9075425790754258
Misclassification rate = 0.09245742092457421
Confusion Matrix :
(0.6178010471204188,
 0.9606525911708254,
 0.03934740882917467,
 0.38219895287958117)
```



(vi) Neural Network Model

```
Testing Accuracy :  0.884022708840227
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      2084
           1       0.66      0.51      0.58       382

    accuracy                           0.88      2466
   macro avg       0.79      0.73      0.75      2466
weighted avg       0.87      0.88      0.88      2466

[[1986   98]
 [ 188  194]]
Sensitivity/tp_rate =  0.5078534031413613
Specificity/tn_rate =  0.9529750479846449
fp rate =  0.04702495201535509
fn rate =  0.49214659685863876
Success rate = 0.884022708840227
Misclassification rate = 0.11597729115977291
```





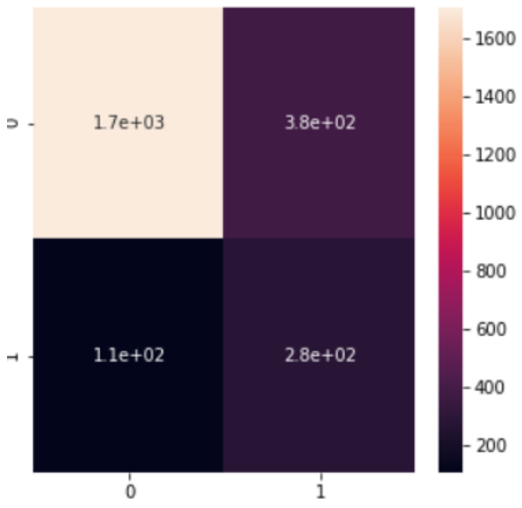
Experiment 2- Imbalance Treatment

To deal with class imbalance problem, we considered the use of the widely used technique to balance the training set before the learning phase, namely the “Synthetic Minority Oversampling Technique” (SMOTE) methodology. In this technique, we simply duplicate examples from the minority class in the training dataset prior to fitting a model. This can balance the class distribution but does not provide any additional information to the model.
An improvement on duplicating examples from the minority class is to synthesize new examples from the minority class. This is a type of data augmentation for tabular data and can be very effective.

```
#X_train, X_test, y_train, y_test = train_test(temp_data, 0.3)
X, y = smote_data(X_train, y_train)

print("Shape of X_train after SMOTE:", X.shape)
print("Shape of X_test after SMOTE :", X_test.shape)
```

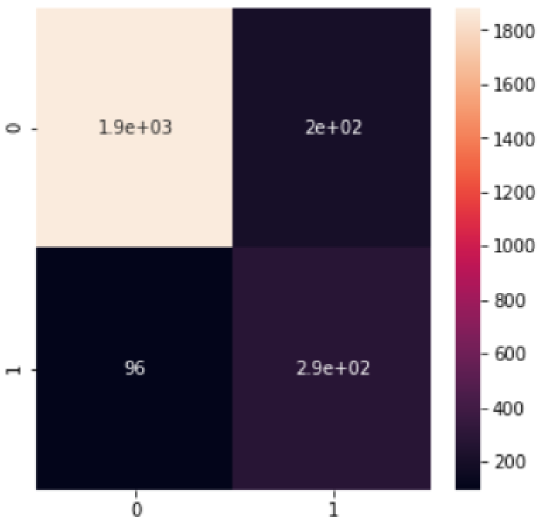
(i) Naïve Bayes Model

```
Testing Accuracy :  0.8041362530413625
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.82      0.88      2084
           1       0.42      0.72      0.53       382

    accuracy                           0.80      2466
   macro avg       0.68      0.77      0.70      2466
weighted avg       0.86      0.80      0.82      2466

[[1707  377]
 [ 106  276]]
Sensitivity/tp_rate =  0.7225130890052356
Specificity/tn_rate =  0.8190978886756238
fp rate =  0.1809021113243762
fn rate =  0.2774869109947644
Success rate = 0.8041362530413625
Misclassification rate = 0.19586374695863748
Confusion Matrix :
(0.7225130890052356,
 0.8190978886756238,
 0.1809021113243762,
 0.2774869109947644)
```



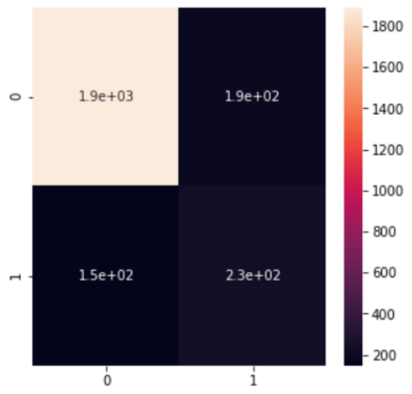
(ii) Logistic Regression Model

```
Testing Accuracy :  0.8791565287915653
Classification Report :
              precision    recall  f1-score   support

           0       0.95      0.90      0.93      2084
           1       0.59      0.75      0.66       382

    accuracy                           0.88      2466
   macro avg       0.77      0.83      0.79      2466
weighted avg       0.89      0.88      0.88      2466

[[1882  202]
 [  96  286]]
Sensitivity/tp_rate =  0.7486910994764397
Specificity/tn_rate =  0.9030710172744721
fp rate =  0.09692898272552783
fn rate =  0.2513089005235602
Success rate = 0.8791565287915653
Misclassification rate = 0.12084347120843471
Confusion Matrix :
(0.7486910994764397,
 0.9030710172744721,
 0.09692898272552783,
 0.2513089005235602)
```



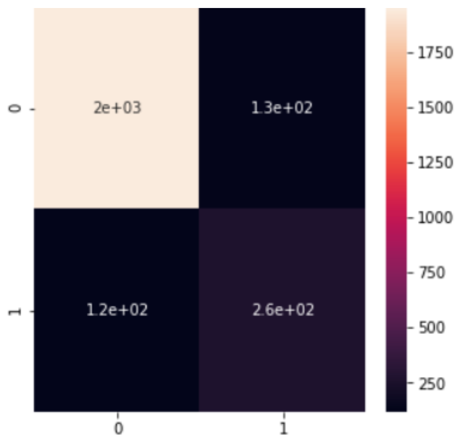
(iii) Decision Tree Model

```
Testing Accuracy :  0.8613138686131386
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.91      0.92      2084
           1       0.55      0.61      0.58       382

    accuracy                           0.86      2466
   macro avg       0.74      0.76      0.75      2466
weighted avg       0.87      0.86      0.86      2466

[[1890  194]
 [ 148  234]]
Sensitivity/tp_rate =  0.612565445026178
Specificity/tn_rate =  0.9069097888675623
fp rate =  0.09309021113243762
fn rate =  0.387434554973822
Success rate = 0.8613138686131386
Misclassification rate = 0.1386861313868613
Confusion Matrix :
(0.612565445026178, 0.9069097888675623, 0.09309021113243762, 0.387434554973822)
```



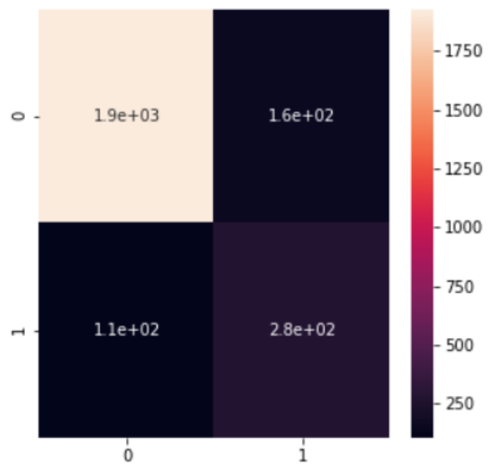
(iv) Random Forest Model

```
Testing Accuracy :  0.8978102189781022
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      2084
           1       0.66      0.69      0.68       382

    accuracy                           0.90      2466
   macro avg       0.80      0.81      0.81      2466
weighted avg       0.90      0.90      0.90      2466

[[1951  133]
 [ 119  263]]
Sensitivity/tp_rate =  0.6884816753926701
Specificity/tn_rate =  0.9361804222648752
fp rate =  0.06381957773512476
fn rate =  0.31151832460732987
Success rate = 0.8978102189781022
Misclassification rate = 0.10218978102189781
Confusion Matrix :
(0.6884816753926701,
 0.9361804222648752,
 0.06381957773512476,
 0.31151832460732987)
```



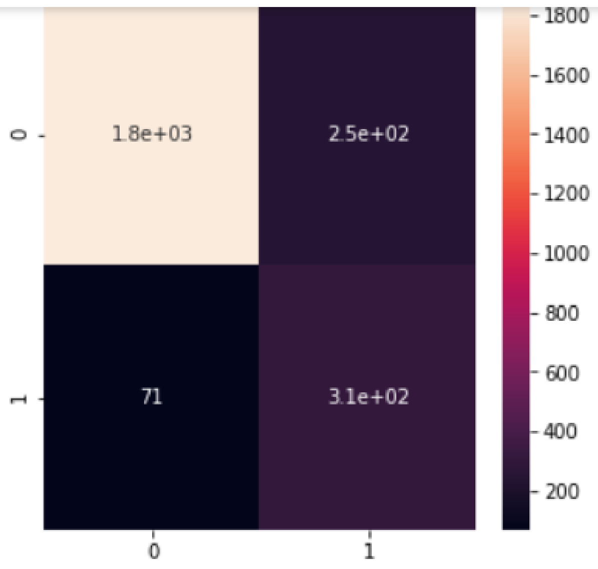
(v) XgBoost Model

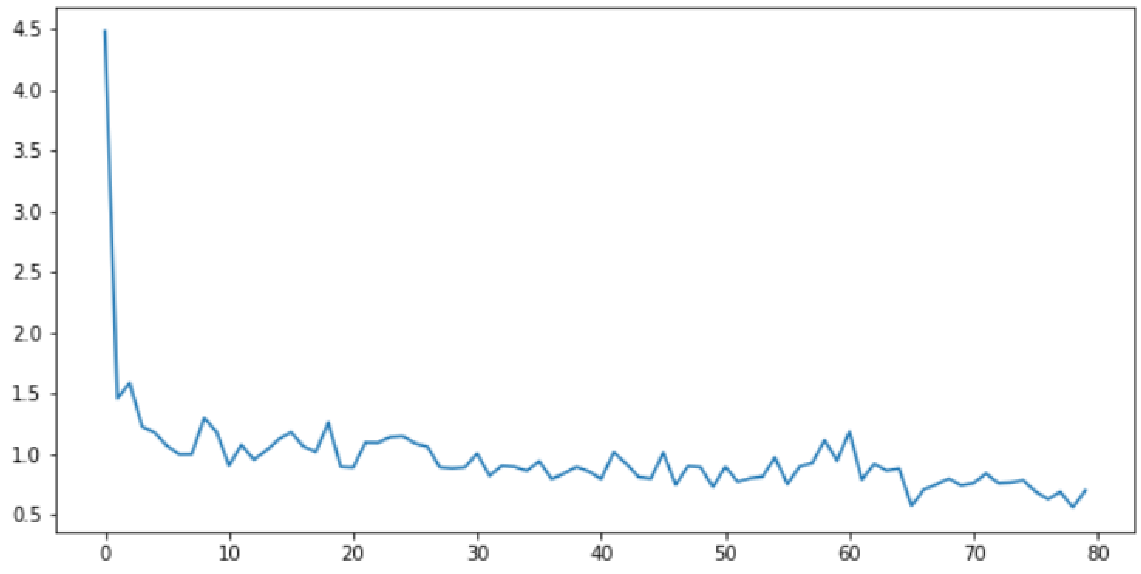
```
Testing Accuracy :  0.8929440389294404
Classification Report :
              precision    recall  f1-score   support

           0       0.95      0.92      0.94      2084
           1       0.64      0.72      0.68       382

    accuracy                           0.89      2466
   macro avg       0.79      0.82      0.81      2466
weighted avg       0.90      0.89      0.90      2466

[[1926  158]
 [ 106  276]]
Sensitivity/tp_rate =  0.7225130890052356
Specificity/tn_rate =  0.9241842610364683
fp rate =  0.07581573896353166
fn rate =  0.2774869109947644
Success rate = 0.8929440389294404
Misclassification rate = 0.1070559610705596
Confusion Matrix :
(0.7225130890052356,
 0.9241842610364683,
 0.07581573896353166,
 0.2774869109947644)
```



(vi) Neural Network Model

```
Testing Accuracy :  0.8714517437145174
Classification Report :
              precision    recall  f1-score   support

           0       0.96      0.88      0.92      2084
           1       0.56      0.81      0.66       382

    accuracy                           0.87      2466
   macro avg       0.76      0.85      0.79      2466
weighted avg       0.90      0.87      0.88      2466

[[1838  246]
 [  71  311]]
Sensitivity/tp_rate =  0.8141361256544503
Specificity/tn_rate =  0.8819577735124761
fp rate =  0.118042226487524
fn rate =  0.18586387434554974
Success rate = 0.8714517437145174
Misclassification rate = 0.12854825628548255
```





Experiment 3- Feature Engineering

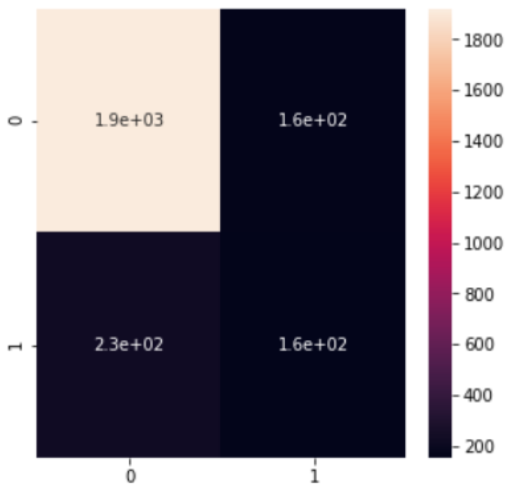
We will be using 2 feature extraction/engineering methods- PCA and RFE.

PCA

Principle Component Analysis (PCA) is a common feature extraction method in data science. Technically, PCA finds the eigenvectors of a covariance matrix with the highest eigenvalues and then uses those to project the data into a new subspace of equal or less dimensions. Practically, PCA converts a matrix of n features into a new dataset of (hopefully) less than n features. That is, it reduces the number of features by constructing a new, smaller number variables which capture a significant portion of the information found in the original features.

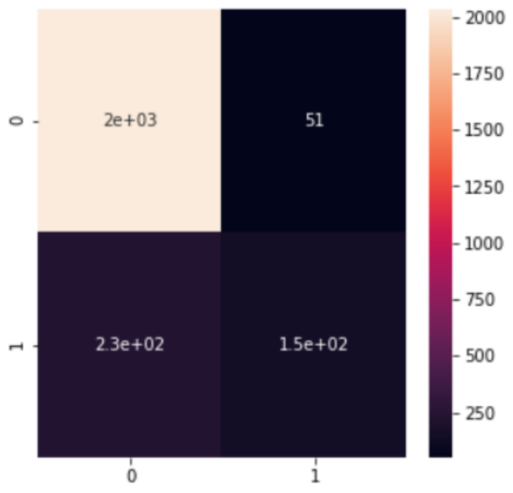
(i) Naïve Bayes Model

```
Testing Accuracy :  0.8410381184103812
Classification Report :
              precision    recall  f1-score   support

           0       0.89      0.92      0.91      2084
           1       0.48      0.41      0.44       382

    accuracy                           0.84      2466
   macro avg       0.69      0.66      0.67      2466
weighted avg       0.83      0.84      0.84      2466

[[1919  165]
 [ 227  155]]
Sensitivity/tp_rate =  0.40575916230366493
Specificity/tn_rate =  0.9208253358925144
fp rate =  0.07917466410748561
fn rate =  0.5942408376963351
Success rate = 0.8410381184103812
Misclassification rate = 0.15896188158961883
Confusion Matrix :
(0.40575916230366493,
 0.9208253358925144,
 0.07917466410748561,
 0.5942408376963351)
```



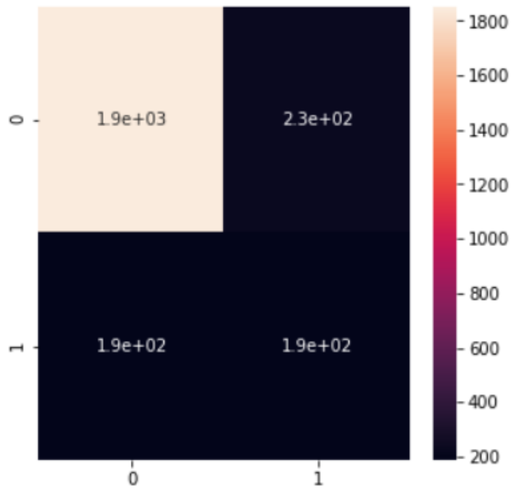
(ii) Logistic Regression Model

```
Testing Accuracy :  0.8856447688564477
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      2084
           1       0.75      0.40      0.52       382

    accuracy                           0.89      2466
   macro avg       0.82      0.69      0.73      2466
weighted avg       0.87      0.89      0.87      2466

[[2033   51]
 [ 231  151]]
Sensitivity/tp_rate =  0.39528795811518325
Specificity/tn_rate =  0.97552783109405
fp rate =  0.024472168905950095
fn rate =  0.6047120418848168
Success rate = 0.8856447688564477
Misclassification rate = 0.11435523114355231
Confusion Matrix :
(0.39528795811518325,
 0.97552783109405,
 0.024472168905950095,
 0.6047120418848168)
```



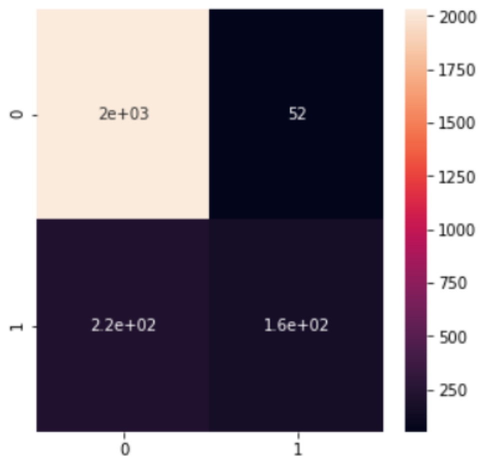
(iii) Decision Tree Model

```
Testing Accuracy :  0.8272506082725061
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      2084
           1       0.45      0.49      0.47       382

    accuracy                           0.83      2466
   macro avg       0.68      0.69      0.68      2466
weighted avg       0.83      0.83      0.83      2466

[[1852  232]
 [ 194  188]]
Sensitivity/tp_rate =  0.49214659685863876
Specificity/tn_rate =  0.8886756238003839
fp rate =  0.11132437619961612
fn rate =  0.5078534031413613
Success rate = 0.8272506082725061
Misclassification rate = 0.17274939172749393
Confusion Matrix :
(0.49214659685863876,
 0.8886756238003839,
 0.11132437619961612,
 0.5078534031413613)
```



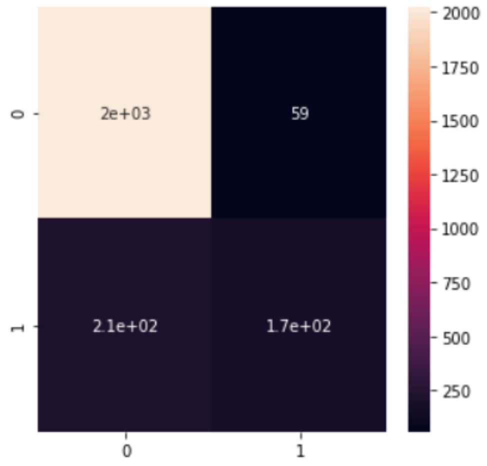
(iv) Random Forest Model

```
Testing Accuracy :  0.889294403892944
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      2084
           1       0.76      0.42      0.54       382

    accuracy                           0.89      2466
   macro avg       0.83      0.70      0.74      2466
weighted avg       0.88      0.89      0.88      2466

[[2032   52]
 [ 221  161]]
Sensitivity/tp_rate =  0.4214659685863874
Specificity/tn_rate =  0.9750479846449136
fp rate =  0.02495201535508637
fn rate =  0.5785340314136126
Success rate = 0.889294403892944
Misclassification rate = 0.11070559610705596
Confusion Matrix :
(0.4214659685863874,
 0.9750479846449136,
 0.02495201535508637,
 0.5785340314136126)
```



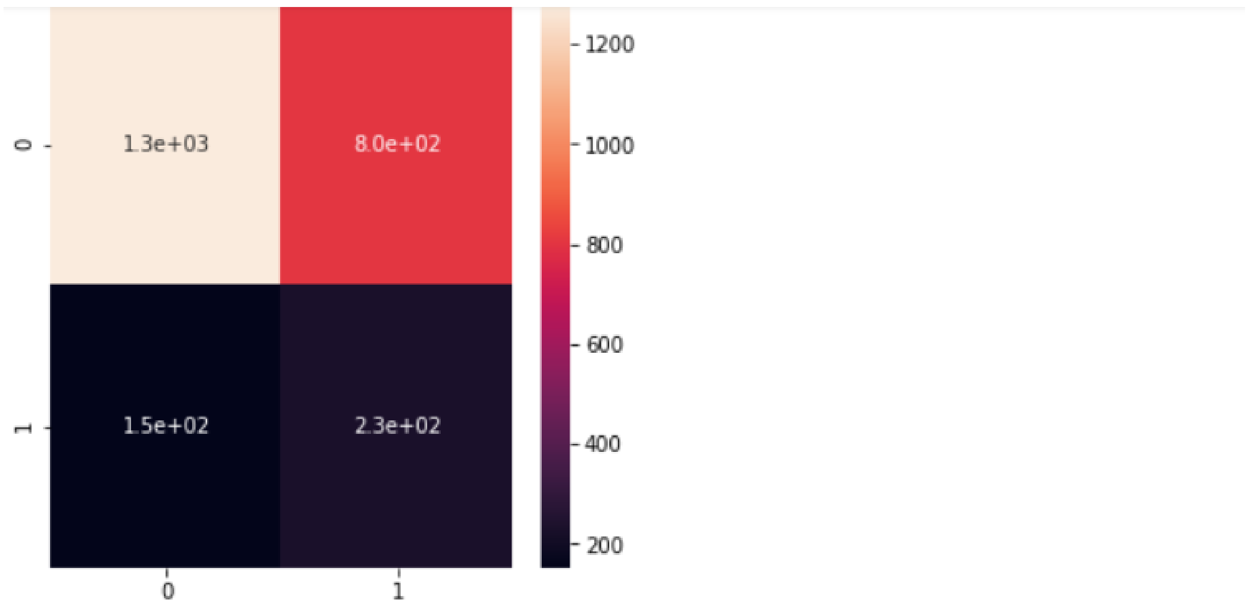
(v) XgBoost Model

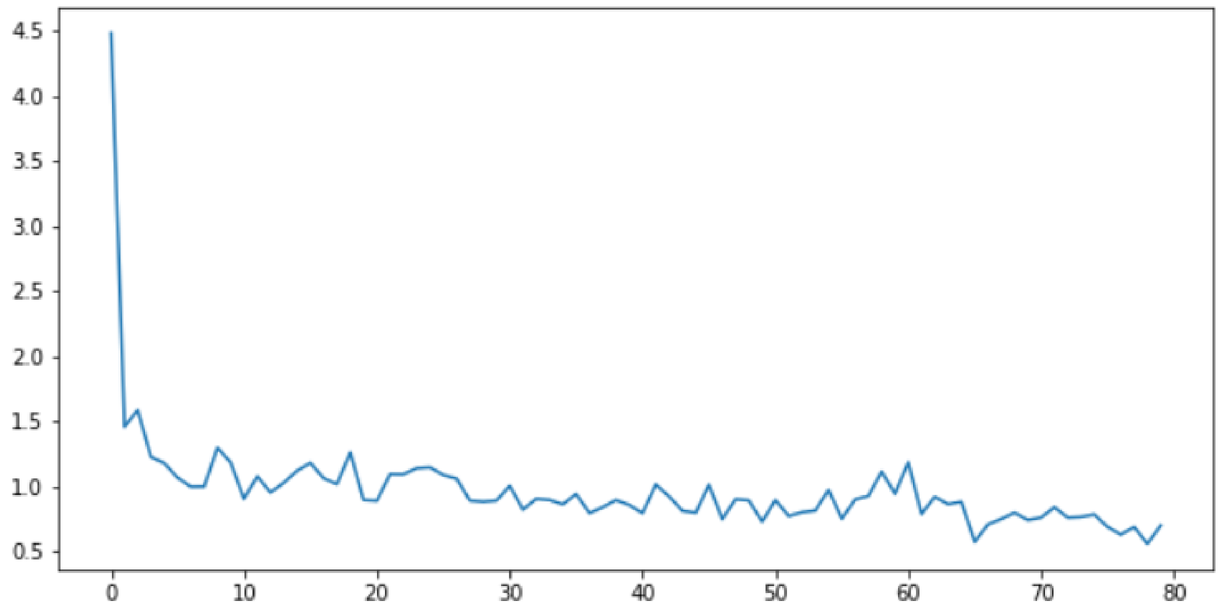
```
Testing Accuracy :  0.8901054339010543
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      2084
           1       0.74      0.45      0.56       382

    accuracy                           0.89      2466
   macro avg       0.82      0.71      0.75      2466
weighted avg       0.88      0.89      0.88      2466

[[2025   59]
 [ 212  170]]
Sensitivity/tp_rate =  0.44502617801047123
Specificity/tn_rate =  0.9716890595009597
fp rate =  0.02831094049904031
fn rate =  0.5549738219895288
Success rate = 0.8901054339010543
Misclassification rate = 0.10989456609894566
Confusion Matrix :
(0.44502617801047123,
 0.9716890595009597,
 0.02831094049904031,
 0.5549738219895288)
```



(vi) Neural Network Model

```
Testing Accuracy :  0.6115166261151662
Classification Report :
              precision    recall  f1-score   support

           0       0.89      0.61      0.73      2084
           1       0.22      0.60      0.32       382

    accuracy                           0.61      2466
   macro avg       0.56      0.61      0.53      2466
weighted avg       0.79      0.61      0.66      2466

[[1279  805]
 [ 153  229]]
Sensitivity/tp_rate =  0.599476439790576
Specificity/tn_rate =  0.6137236084452975
fp rate =  0.3862763915547025
fn rate =  0.4005235602094241
Success rate = 0.6115166261151662
Misclassification rate = 0.3884833738848337
```



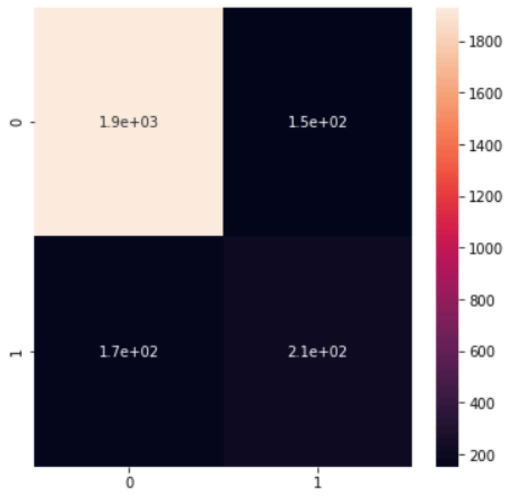


RFE

Recursive Feature Elimination It is a greedy optimization algorithm which aims to find the best performing feature subset. It repeatedly creates models and keeps aside the best or the worst performing feature at each iteration. It constructs the next model with the left features until all the features are exhausted. It then ranks the features based on the order of their elimination.

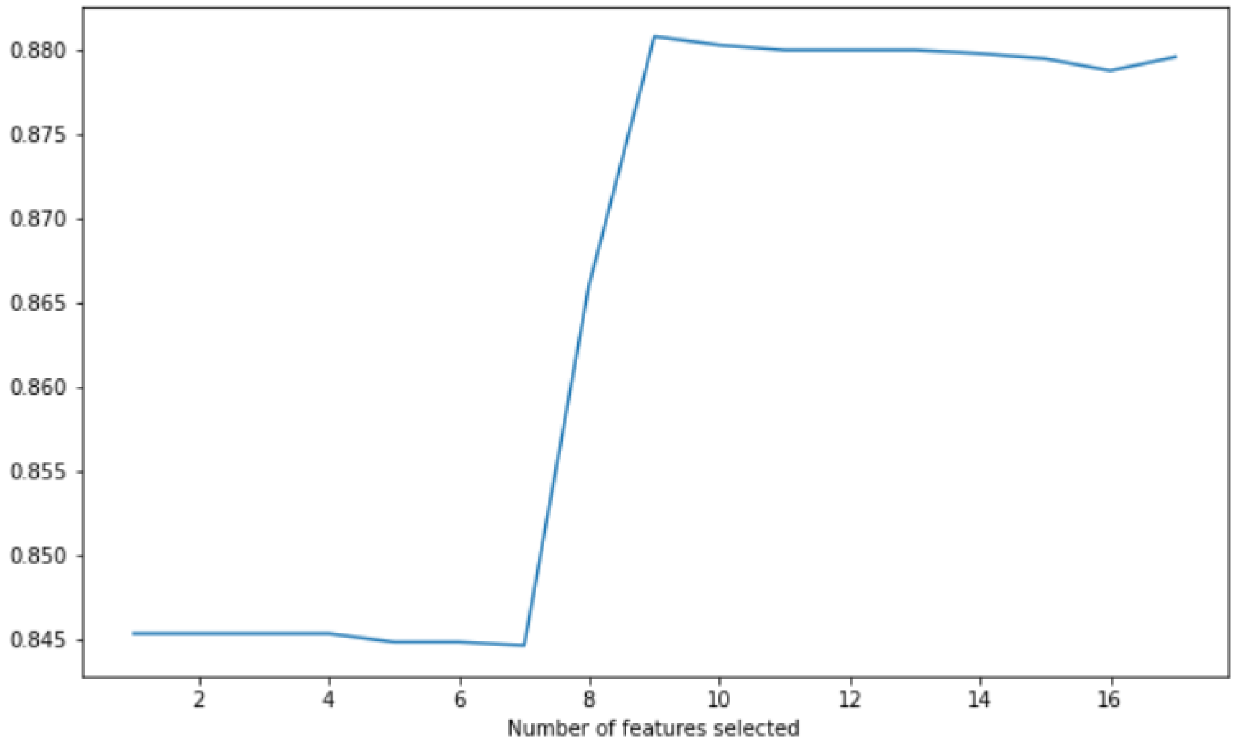
(i) Naïve Bayes Model

```
Testing Accuracy :  0.8678021086780211
Classification Report :
              precision    recall  f1-score   support

           0       0.92      0.93      0.92      2084
           1       0.58      0.55      0.56       382

    accuracy                           0.87      2466
   macro avg       0.75      0.74      0.74      2466
weighted avg       0.87      0.87      0.87      2466

[[1931  153]
 [ 173  209]]
Sensitivity/tp_rate =  0.5471204188481675
Specificity/tn_rate =  0.9265834932821497
fp rate =  0.07341650671785029
fn rate =  0.45287958115183247
Confusion Matrix :
(0.5471204188481675,
 0.9265834932821497,
 0.07341650671785029,
 0.45287958115183247)
```

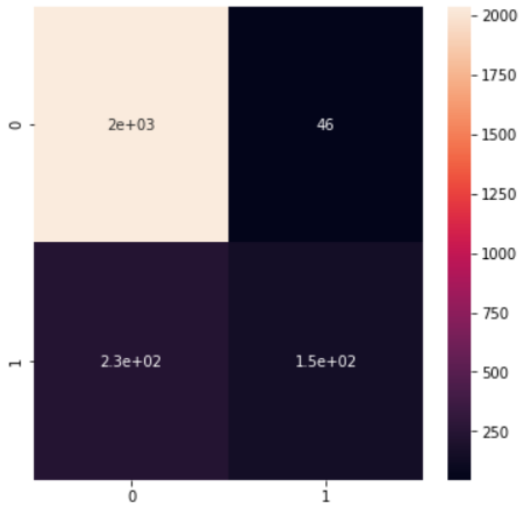


(ii) Logistic Regression Model

```
Optimal number of features: 9
 Selected features: ['Informational', 'BounceRates', 'ExitRates', 'PageValues',
 'SpecialDay', 'Month', 'OperatingSystems', 'VisitorType', 'Weekend']
```

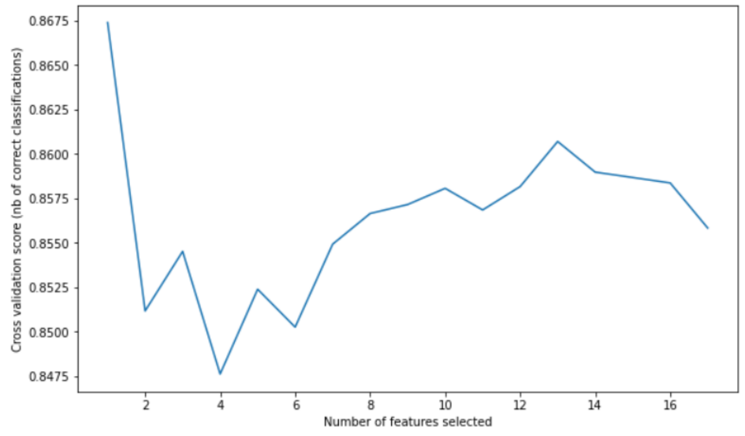


```
Testing Accuracy :  0.8868613138686131
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.98      0.94      2084
           1       0.76      0.39      0.52       382

    accuracy                           0.89      2466
   macro avg       0.83      0.68      0.73      2466
weighted avg       0.88      0.89      0.87      2466

[[2038   46]
 [ 233  149]]
Sensitivity/tp_rate =  0.3900523560209424
Specificity/tn_rate =  0.9779270633397313
fp rate =  0.022072936660268713
fn rate =  0.6099476439790575
Confusion Matrix :
(0.3900523560209424,
 0.9779270633397313,
 0.022072936660268713,
 0.6099476439790575)
```

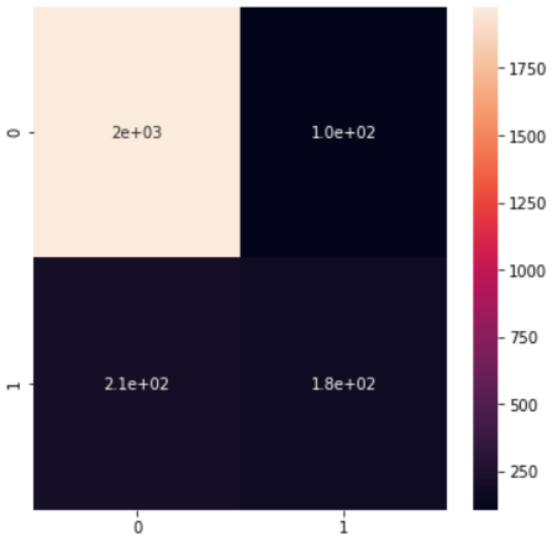


(iii) Decision Tree Model

```
Optimal number of features: 1
Selected features: ['PageValues']
```

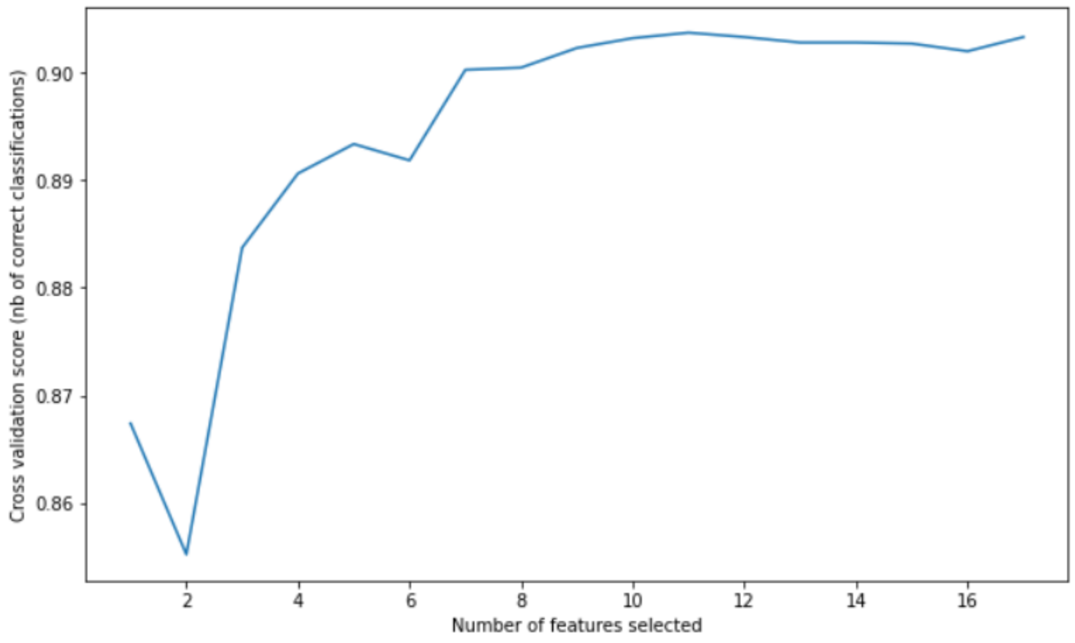


```
Testing Accuracy :  0.8734793187347932
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      2084
           1       0.62      0.46      0.53       382

    accuracy                           0.87      2466
   macro avg       0.77      0.70      0.73      2466
weighted avg       0.86      0.87      0.87      2466

[[1979  105]
 [ 207  175]]
Sensitivity/tp_rate =  0.4581151832460733
Specificity/tn_rate =  0.949616122840691
fp rate =  0.05038387715930902
fn rate =  0.5418848167539267
Confusion Matrix :
(0.4581151832460733,
 0.949616122840691,
 0.05038387715930902,
 0.5418848167539267)
```

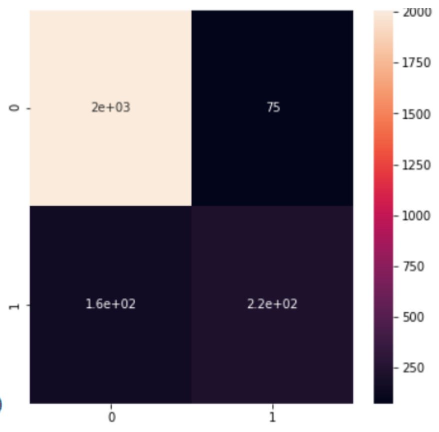


(iv) Random Forest Model

```
Optimal number of features: 11
 Selected features: ['Administrative', 'Administrative_Duration',
 'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
 'BounceRates', 'ExitRates', 'PageValues', 'Month', 'Region', 'TrafficType']
```

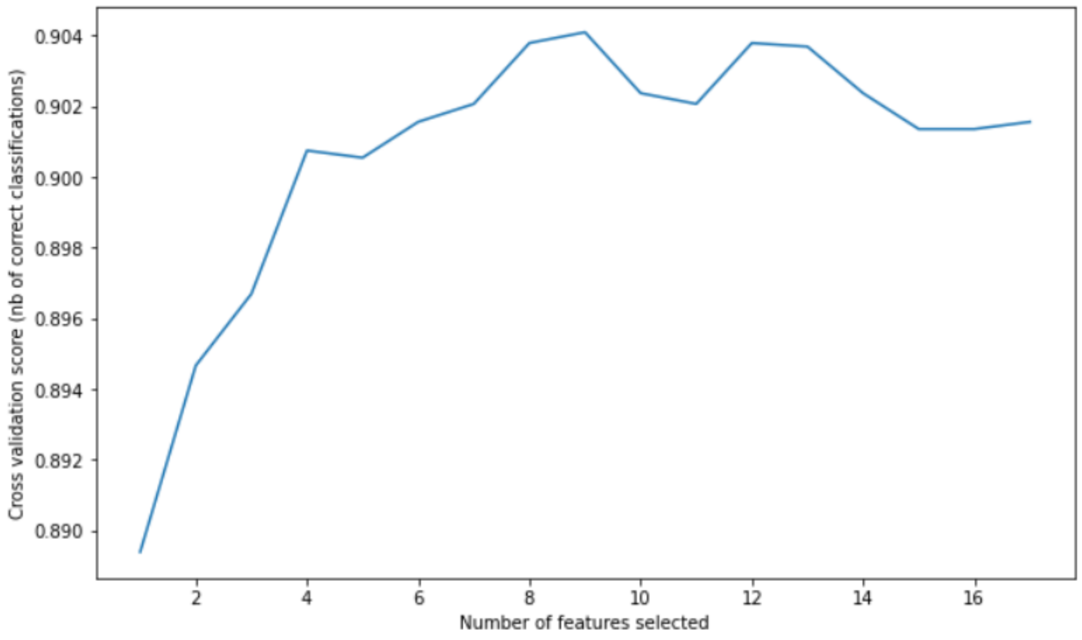


```
Testing Accuracy :  0.9047039740470397
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.96      0.94      2084
           1       0.75      0.58      0.65       382

    accuracy                           0.90      2466
   macro avg       0.84      0.77      0.80      2466
weighted avg       0.90      0.90      0.90      2466

[[2009   75]
 [ 160  222]]
Sensitivity/tp_rate =  0.581151832460733
Specificity/tn_rate =  0.9640115163147792
fp rate =  0.03598848368522073
fn rate =  0.418848167539267
Confusion Matrix :
(0.581151832460733, 0.9640115163147792, 0.03598848368522073, 0.418848167539267)
```

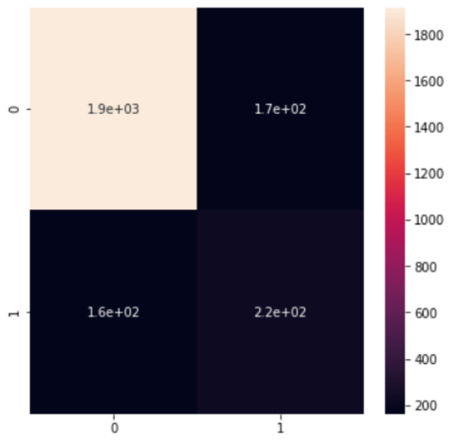


(v) XgBoost Model

```
Optimal number of features: 9
 Selected features: ['Administrative', 'Informational', 'ProductRelated',
 'ProductRelated_Duration', 'BounceRates', 'ExitRates', 'PageValues', 'Month',
 'VisitorType']
```



```
Testing Accuracy :  0.8657745336577454
Classification Report :
              precision    recall  f1-score   support

           0       0.92      0.92      0.92      2084
           1       0.57      0.58      0.57       382

    accuracy                           0.87      2466
   macro avg       0.74      0.75      0.75      2466
weighted avg       0.87      0.87      0.87      2466

[[1915  169]
 [ 162  220]]
Sensitivity/tp_rate =  0.5759162303664922
Specificity/tn_rate =  0.9189059500959693
fp rate =  0.08109404990403071
fn rate =  0.42408376963350786
Confusion Matrix :
(0.5759162303664922,
 0.9189059500959693,
 0.08109404990403071,
 0.42408376963350786)
```

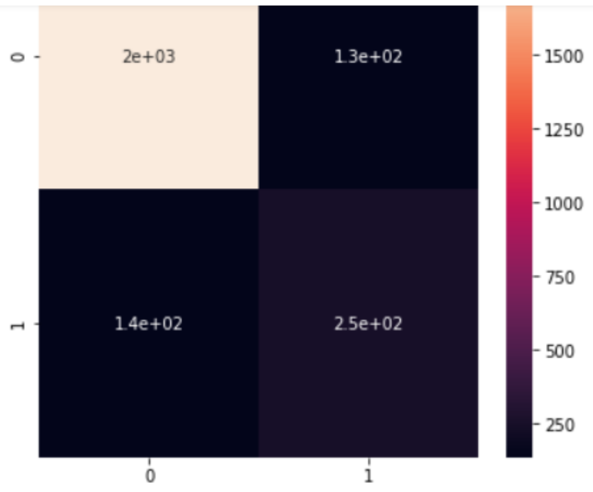


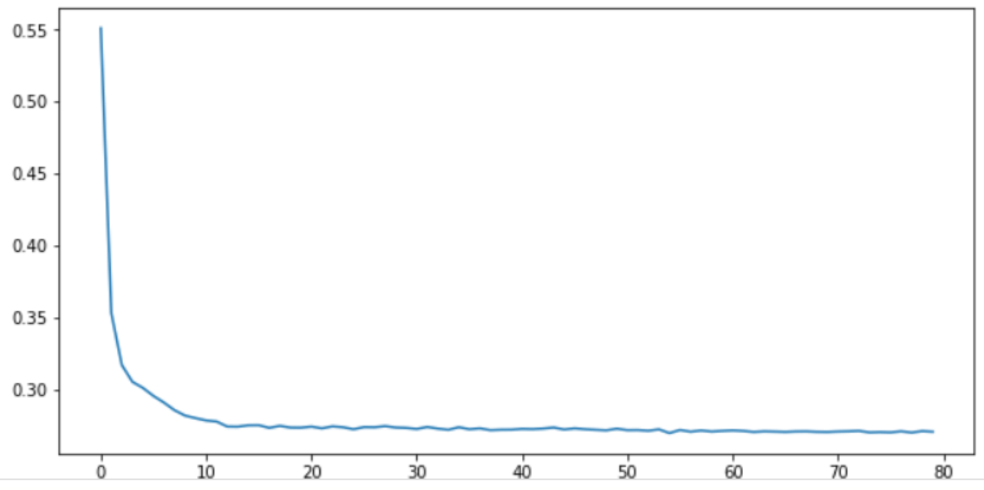
(vi) Neural Network Model

Optimal features-

```
['Administrative', 'Administrative_Duration', 'Informational_Duration', 'ProductRel
ated', 'ProductRelated_Duration', 'BounceRates', 'ExitRates', 'PageValues', 'Month'
, 'Region', 'TrafficType']
```

```
Testing Accuracy :  0.8909164639091647
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.94      0.94      2084
           1       0.65      0.65      0.65       382

    accuracy                           0.89      2466
   macro avg       0.79      0.79      0.79      2466
weighted avg       0.89      0.89      0.89      2466

[[1950  134]
 [ 135  247]]
Sensitivity/tp_rate =  0.6465968586387435
Specificity/tn_rate =  0.935700575815739
fp rate =  0.06429942418426103
fn rate =  0.35340314136125656
```





Experiment 4- Feature Engineering and other techniques combined

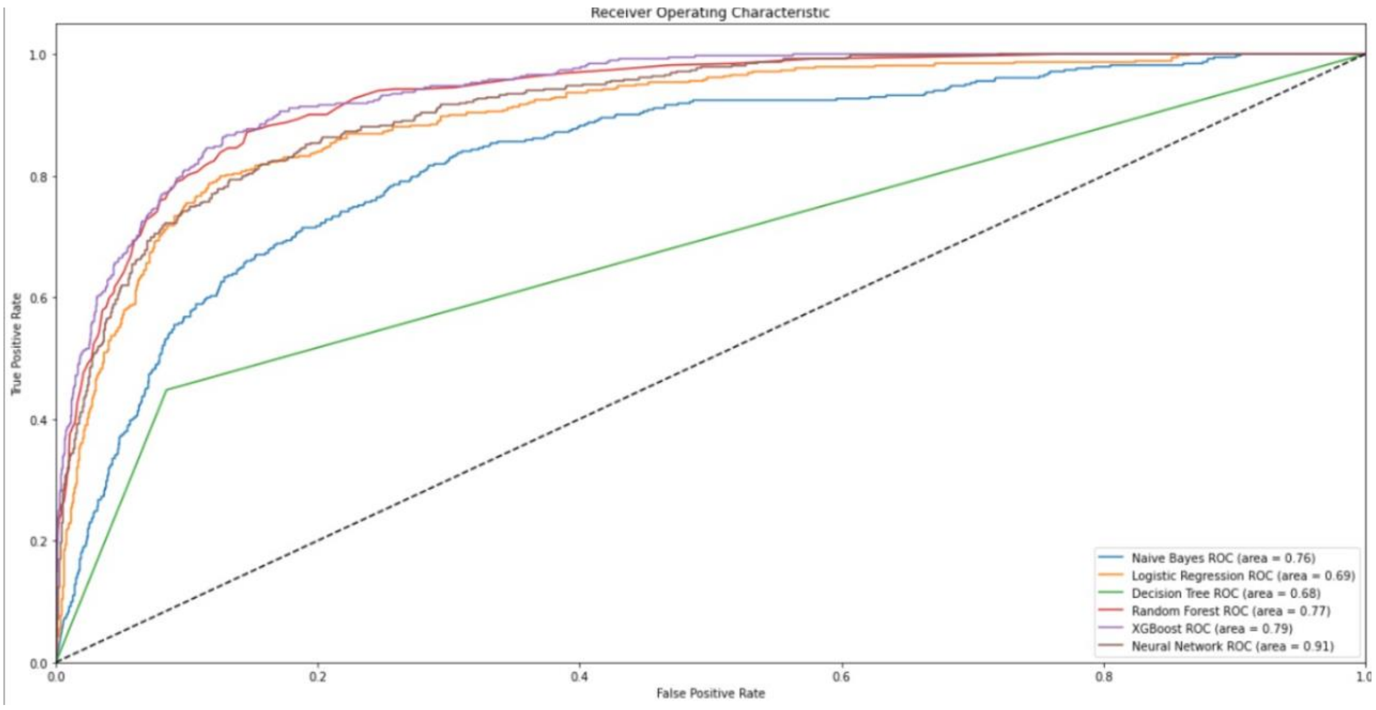
In this experiment, we have further divided it into two categories – PCA+ SMOT and RFE + SMOT thus combining feature extraction with minority overfitting technique.

Comparison

Now, comparing the 4 experiments performed on the basis of success rate/testing accuracy, tp rate(sensitivity), tn rate (specificity), fp rate and f1 score/

Experiment: 1

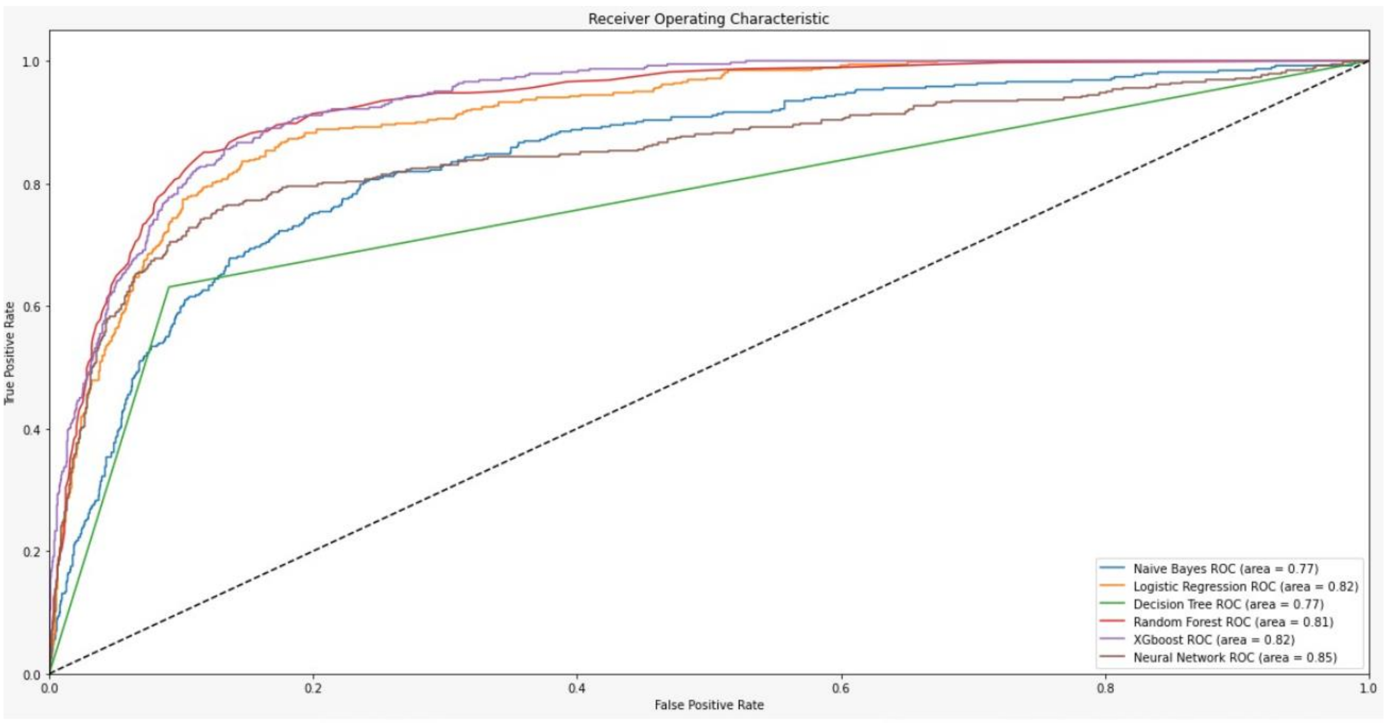
Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.796	0.712	0.811	0.188	0.52
Logistic Regression	0.885	0.395	0.975	0.024	0.52
Decision Trees	0.843	0.445	0.916	0.083	0.47
Random Forest	0.904	0.575	0.964	0.035	0.65
XgBoost	0.907	0.617	0.960	0.039	0.67
Neural Network	0.884	0.507	0.952	0.047	0.58



It can be observed from the ROC curve that Neural Network cover most area even though it has a low F1 score.

Experiment 2:

Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.804	0.722	0.819	0.180	0.53
Logistic Regression	0.879	0.748	0.903	0.096	0.66
Decision Trees	0.861	0.612	0.906	0.093	0.58
Random Forest	0.897	0.688	0.936	0.063	0.68
XgBoost	0.892	0.722	0.924	0.075	0.68
Neural Network	0.871	0.814	0.881	0.118	0.66



Apart from Neural Network, Xgboost has slightly better results than random forest along with highest f1score among all other models leading it to be the optimum model.

Experiment 3:

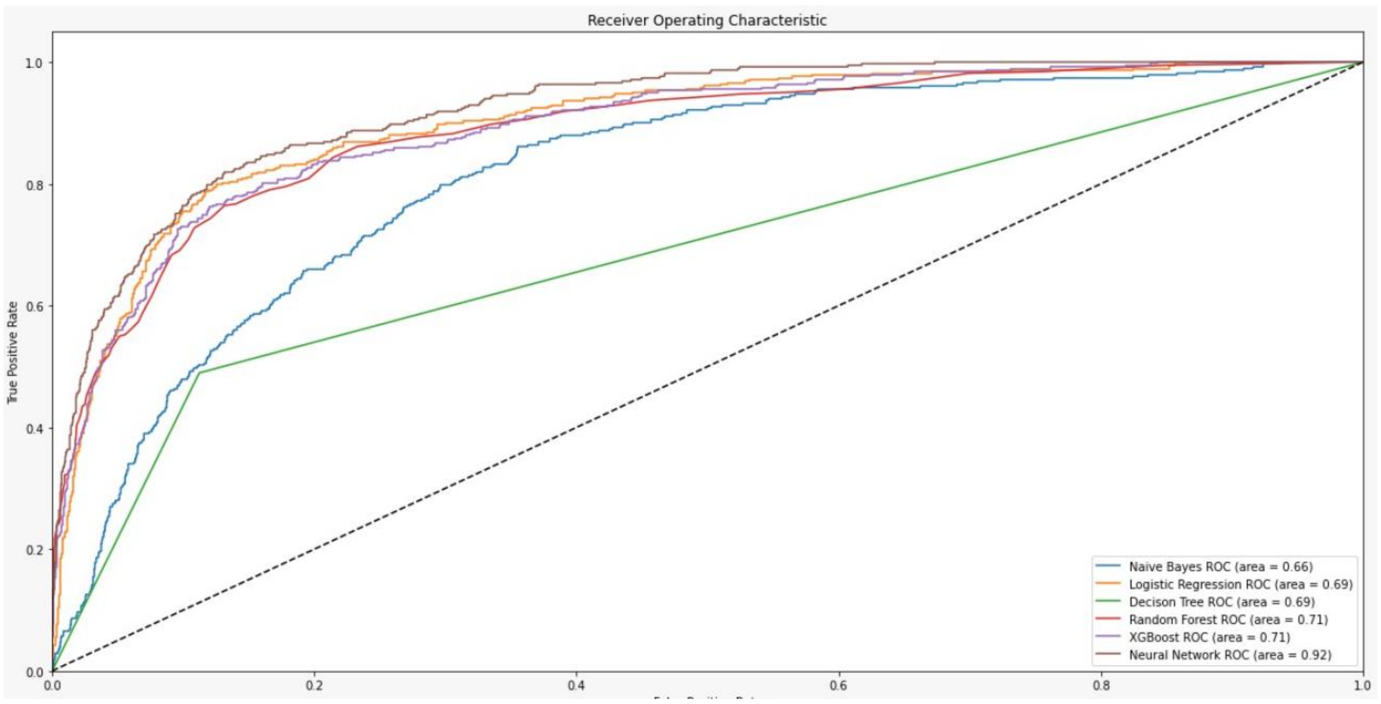
PCA

Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.841	0.405	0.920	0.079	0.44
Logistic Regression	0.885	0.395	0.975	0.024	0.52
Decision Trees	0.827	0.492	0.888	0.111	0.47
Random Forest	0.889	0.421	0.975	0.024	0.54
XgBoost	0.890	0.445	0.971	0.028	0.56
Neural Network	0.611	0.599	0.613	0.386	0.32

RFE

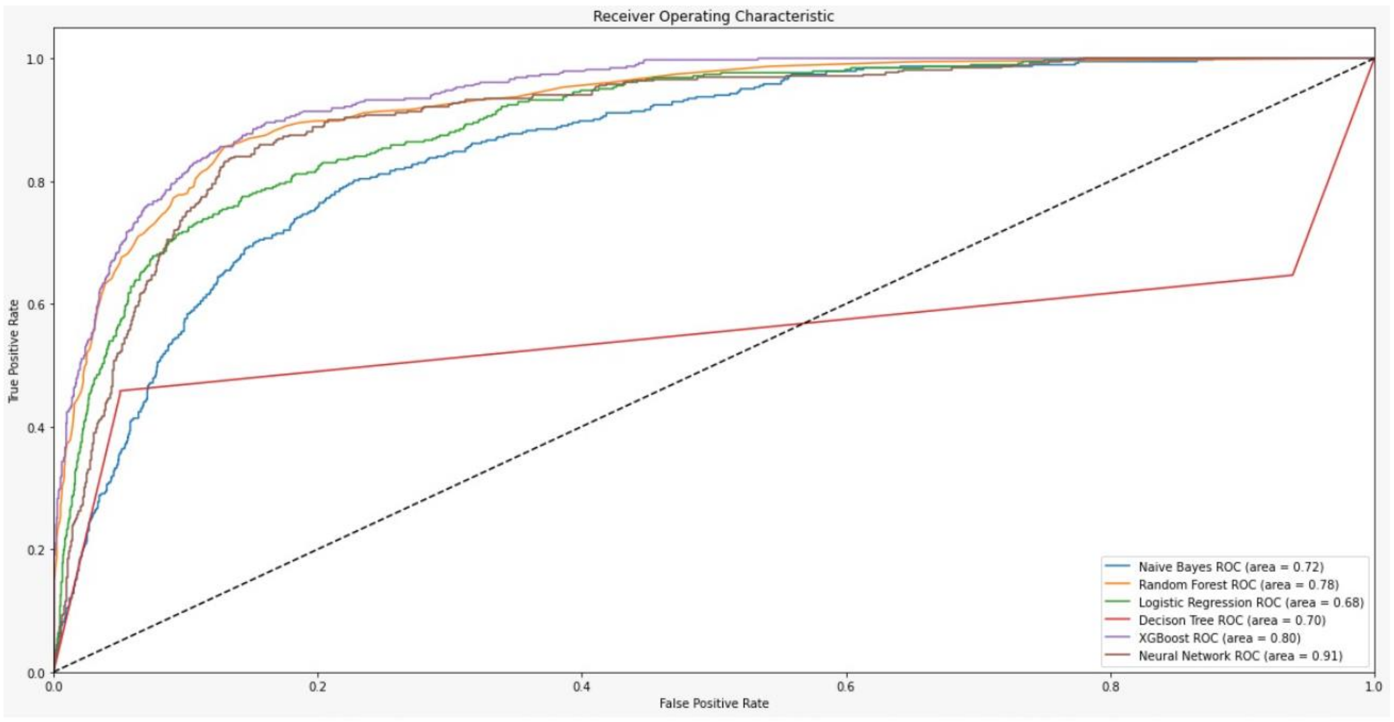
Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.867	0.547	0.926	0.073	0.56
Logistic Regression	0.886	0.390	0.977	0.022	0.52
Decision Trees	0.873	0.458	0.949	0.050	0.53
Random Forest	0.904	0.581	0.964	0.035	0.65
XgBoost	0.865	0.575	0.918	0.081	0.57
Neural Network	0.890	0.646	0.935	0.064	0.65

For PCA:



It is observable that neural network has the best outcome even though its f1 score is low.

For RFE:



In this model also, Neural Network and Xgboost are giving comparable results. So both can be considered as optimum model but random forest has slightly higher f1score so it would be better to consider it.

Experiment 4:

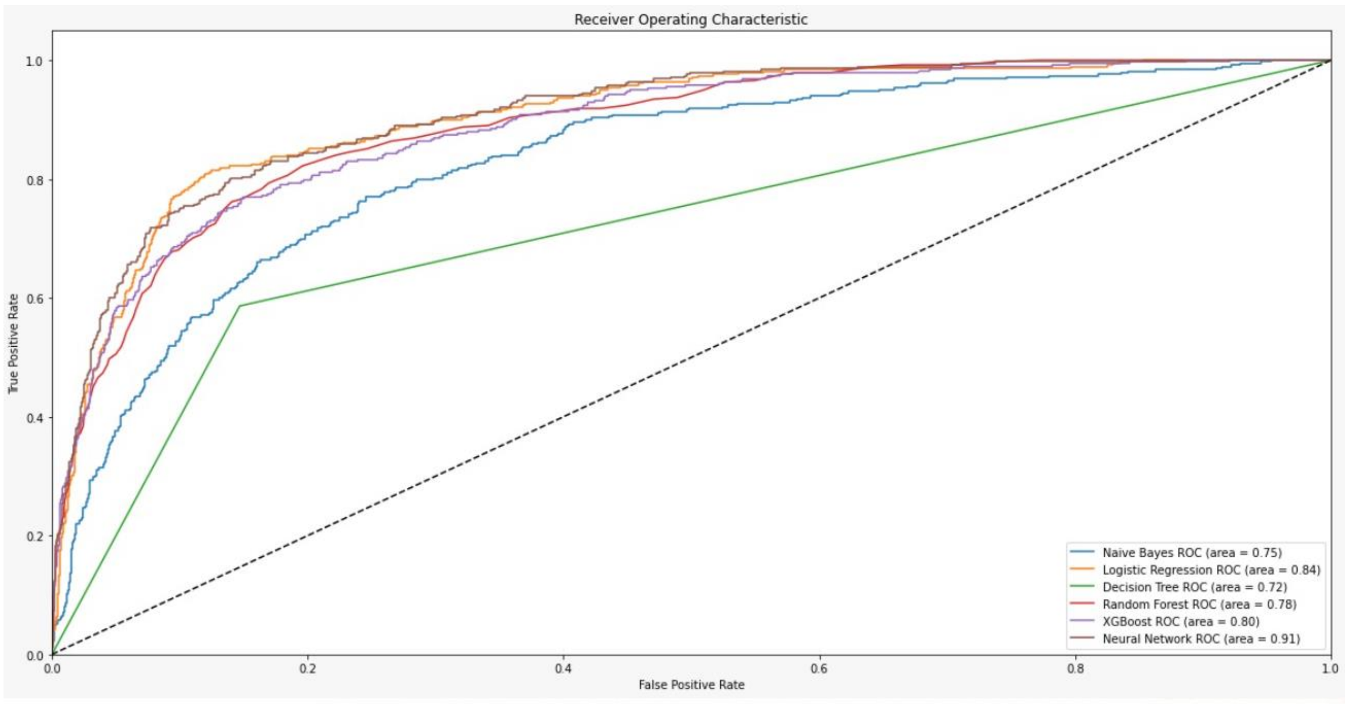
PCA+SMOT

Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.775	0.727	0.784	0.215	0.50
Logistic Regression	0.874	0.780	0.892	0.107	0.66
Decision Trees	0.802	0.589	0.842	0.157	0.48
Random Forest	0.874	0.628	0.919	0.080	0.61
XgBoost	0.832	0.756	0.846	0.153	0.58
Neural Network	0.874	0.746	0.898	0.101	0.65

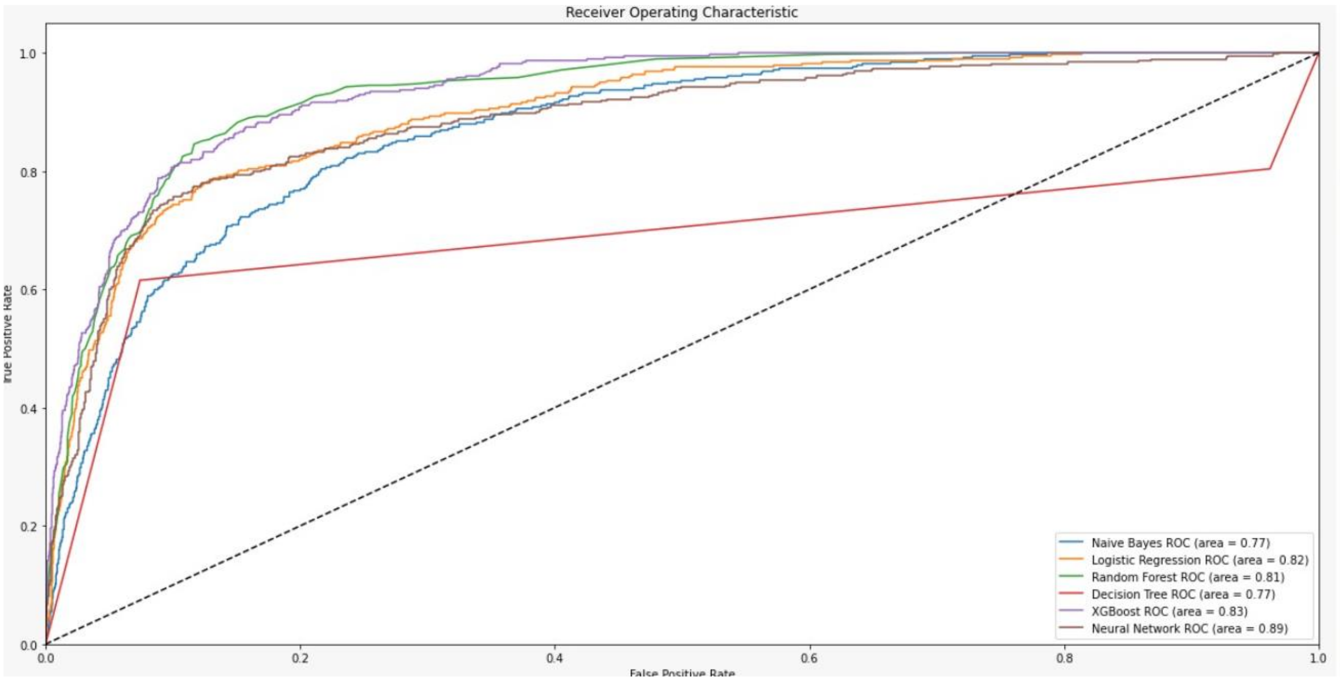
RFE+SMOT

Algorithm	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.840	0.672	0.871	0.128	0.57
Logistic Regression	0.873	0.738	0.897	0.102	0.64
Decision Trees	0.872	0.612	0.920	0.079	0.60
Random Forest	0.899	0.732	0.929	0.070	0.69
XgBoost	0.911	0.641	0.960	0.039	0.69
Neural Network	0.879	0.795	0.894	0.105	0.67

For PCA and SMOT-



For RFE and SMOT-

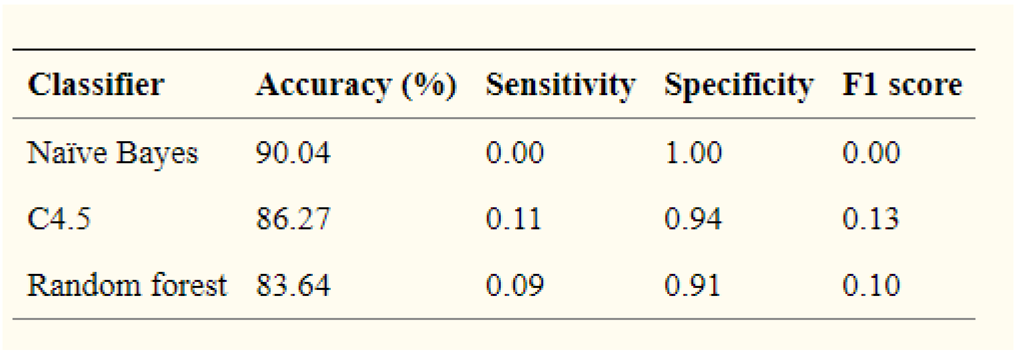


From the above plot of ROC curves for the explored models, we can observe that Neural Network has the largest area under the curve.

Comparative Experiment

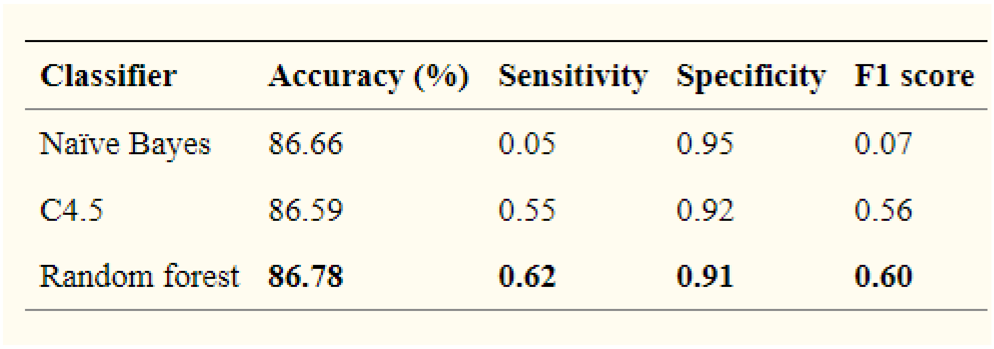
This is the official published paper where they have taken Naïve Bayes, Decision tree and random forest models and applied it on normal data and data after Synthetic Minority Oversampling Technique (smot). Train test size taken by them is 70-30 as opposed to our train test size of 80-20.

On normal data-



Classifier	Accuracy (%)	Sensitivity	Specificity	F1 score
Naïve Bayes	90.04	0.00	1.00	0.00
C4.5	86.27	0.11	0.94	0.13
Random forest	83.64	0.09	0.91	0.10

After oversampling data-



Classifier	Accuracy (%)	Sensitivity	Specificity	F1 score
Naïve Bayes	86.66	0.05	0.95	0.07
C4.5	86.59	0.55	0.92	0.56
Random forest	**86.78**	**0.62**	**0.91**	**0.60**

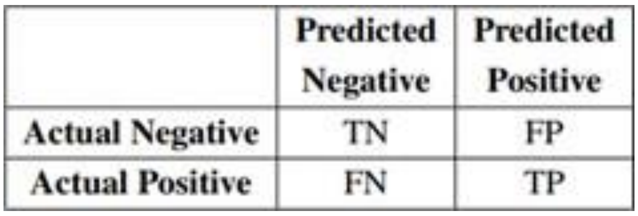
Model Performance

Performance Matrix

There are several matrices available to help us analyze the performance of our model and if it is returning reasonably satisfactory results upon applying different algorithms. In this problem, we are dealing with binary classification of whether or not revenue i.e. transaction on e-commerce site has taken place or not. For this, if target value is 1 (True) means revenue has been generated and if the target value is 0 (False), it means that no e-commerce revenue has been generated in that session.

Confusion Matrix

Confusion Matrix is one of the ways to find how your testing data performs after going into the model and summarizes how good your model is. It basically compares actual values of the testing data which you have kept away from the machine to the predicted machine output. In this matrix, the rows represent the actual values while the columns represent the predicted values.



	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

True Positive (TP)- These are the occurrences where both the predictive and actual class are true i.e. when the revenue has been generated and the model also shows revenue to be generated.

True Negative (TN)- True negative are those data points where the actual outcome is also false i.e. revenue has not been generated for the session and the machine prediction also shows the same.

False Negative (FN)- These are instances from training dataset where revenue has been generated in actual but the model predicts otherwise.

False Positives (FP)- These are those occurrences where the model predicts revenue to be generated but in actual no purchase has been made from e-commerce.

Accuracy- Evaluation of classification models is done by one of the metrics called accuracy. Accuracy is the fraction of prediction. It determines the number of correct predictions over the total number of predictions made by the model.
The formula of accuracy is- Accuracy = (TP+TN)/(TP+TN +FP+FN).

Sensitivity/TP Rate: Classifier's performance to spot positive results is related by Sensitivity. It is a measure of the number of patients who are classified as having complications among those who actually have the complications. Specificity is calculated as follows Precision = TP/(TP+FN).

Specificity/TN Rate: Classifier's performance to spot negative results is related by Specificity. It is a measure of the number of patients who are classified as not having complications among those who actually did not have the complications. Specificity is calculated as follows: TN/(TN+FP).

Conclusion

This online shopper customer intention problem we have taken is a binary classification problem. We have applied 6 models and performed 4 different techniques (referred to as experiments) in this report to find out which model gives better accuracy, sensitivity and other such results in each experiment and then drawn out final conclusion on which model is better overall.

Oversampling and feature selection pre-processing techniques are applied to improve the success rates and scalability of the algorithms. Selection of the optimal model has been done by combining high accuracy with minimal fp rate. The best results are achieved with XGBoost algorithm in terms of tp rate, fp rate and f1 score. Neural Network is also ideal and it covers the largest area amongst other models, though compromises in fields of fp rate.

Bibliography

1. Carmona CJ, Ramírez-Gallego S, Torres F, Bernal E, del Jesús MJ, García S (2012) Web usage mining to improve the design of an e-commerce website: OrOliveSur.com.

2. Rajamma RK, Paswan AK, Hossain MM (2009) Why do shoppers abandon shopping cart? Perceived waiting time, risk, and transaction inconvenience.

3. Ding AW, Li S, Chatterjee P (2015) Learning user real-time intent for optimal dynamic web page transformation.

4. https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning.

5. ROC curves for machine learning https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

6. Naive Bayes classifier https://www.geeksforgeeks.org/naive-bayes-classifiers/

7. Recursive feature elimination https://machinelearningmastery.com/rfe-feature-selection

8. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7256375/

9. Random forest https://builtin.com/data-science/random-forest-algorithm

10. Decision Tree https://www.javatpoint.com/machine-learning-decision-tree-