ECommerce Purchase Prediction Using Machine Learning

Group 16

Ajay Gaur (1910110037)

Vanaparthy Vishnu Guptha (1910110431)

Sushanth Reddy Sandadi (1910110346)

Prof. Madan Gopal

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 set up. 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.

Project Proposal:

The project intends to train machine learning models to predict whether a customer will buy a certain type of product listed on an eCommerce website or not. The models will help websites designer to create better sales strategies i.e. offering discounts and offers to customers while they are browsing for products in real-time. To do this, we will rely on session and visitor information and use the various machine learning techniques to reach the target audience, offering them the most relevant products for purchase. In this project, we have used 6 different models: Naive Bayes, Logistic Regression, Random Forest, Decision Tree, Support Vector Machine (SVM), and k-nearest neighbors. We have also divided our data into various testing experiments to observe the changes upon standardizing, undersampling for majority class oversampling for minority class, and computed their results 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 have been done in Python and executed using a Jupyter notebook.

Domain Knowledge:

Technology has enabled people to connect more quickly, effectively, and reliably. With each mass implementation of technology to people, each implementation has made greater strides into making life more comfortable and convenient for them. Ecommerce has been one such congregation of technologies and processes which allow commerce to happen in an online form. The online form has allowed for faster transactions, greater selection, and better customer support. All these conveniences have been beneficial for the customers at large. With the COVID-19 pandemic, it has become clear that such a form of commerce has become the norm rather than the second counterpart. The COVID-19 pandemic resulted in an increased demand for doing business via the Internet. This has resulted in a search for tools and techniques for better prediction of sales and new marketing strategies by managers of retail firms, both online and brick-mortar going online. Now, they want to orient their marketing actions to the right target. Now a newer trend has emerged of identifying in real-time the purchasing intention of potential customers as soon as they have logged onto the website. This is in contrast to the older models where the risk of losing customers without having bought anything existed. Most want insights from customer behavioral data of online customers which help them formulate these strategies. For acquiring these insights they are turning to experts in data analysis, data science, and machine learning fields.

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.

Data Description:

The project uses the Online Shopper's Purchasing Intentions Dataset compiled by C Okan Sakar and Yomi Kastro. C Okan Sakar is a faculty of Engineering and Natural Sciences at Bahcesehir University, Turkey.
Link:
https://archive.ics.uci.edu/ml/datasets/Online+Shoppers+Purchasing+Intention+Dataset
The dataset consists of 10 numerical and categorical attributes. There are 12330 sessions that have been complied with. Each session's data has been added after the closing of the session. Dataset has been created such that each session belongs to a different user in a 1-year period. Of the 12330 sessions in the dataset, 84.5% (10422) were negative class samples that did not end with shopping and the rest (1908) were positive class samples ending with shopping. The dataset consists of 10 numerical and 8 categorical attributes. They are listed below with their data:

- The "Bounce Rate", "Exit Rate" and "Page Value" features represent the metrics measured by "Google Analytics" for each page in the e-commerce site.
- Administrative- No. of administrative pages visited in a session.

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

Feature	Min	Max

Administrative	0	27
Administrative Duration	0	3398
Informational	0	24
Informational Duration	0	2549
Product Related	0	705
Product Related Duration	0	63973
Bounce Rate	0	0.2
Exit Rate	0	0.2
Special Day	0	361
Page Value	0	1.0

- Table- II: Categorical features if the data set

Feature	Number of values
Operating System	8
Browser	13
Region	9
Traffic Type	20
Visitor Type	3

Weekend	2
Month	12
Revenue	2

Data Analysis

Data Preprocessing:

```
df = pd.read_csv("online_shoppers_intention.csv")
df.head()
```

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0

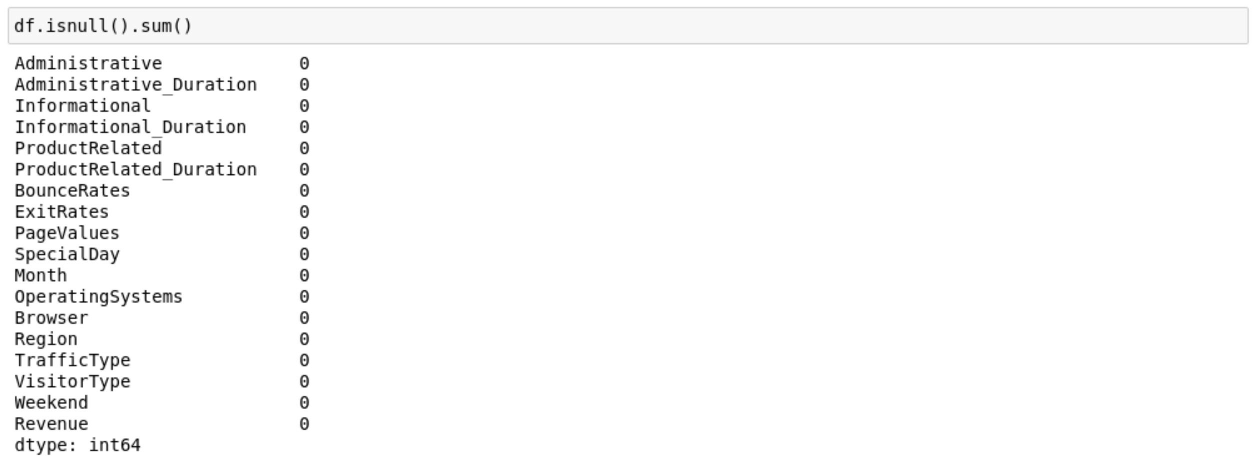
SpecialDay	Month	OperatingSystems	Browser	Region	TrafficType	VisitorType	Weekend	Revenue
0.0	Feb	1	1	1	1	Returning_Visitor	False	False
0.0	Feb	2	2	1	2	Returning_Visitor	False	False
0.0	Feb	4	1	9	3	Returning_Visitor	False	False
0.0	Feb	3	2	2	4	Returning_Visitor	False	False
0.0	Feb	3	3	1	4	Returning_Visitor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

We are first reading the data from the dataset by using Juptyer Notebook and printing the head of the dataset.

```
df.shape
```

```
(12330, 18)
```

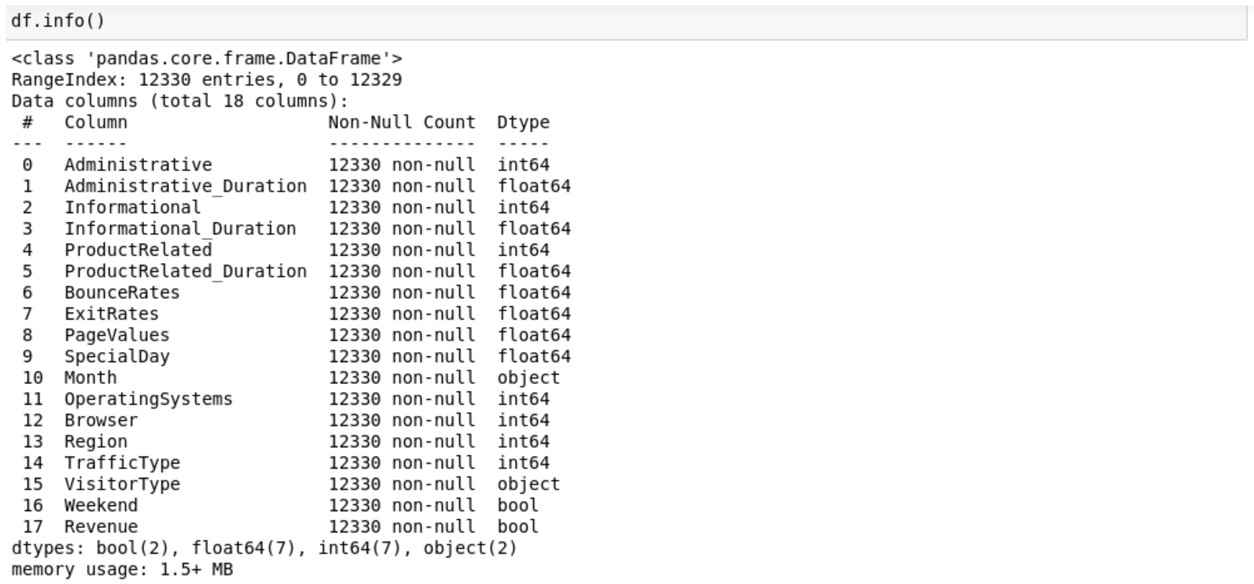
By observing the given dataset, we found that it contains 12330 observations and 18 variables.



```
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
```

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



```
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)
memory usage: 1.5+ MB
```

The above diagram indicates data types of different features in our dataset and they are mostly floating and ‘int64’ data types. 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 respectively.

```
df.describe().T
```

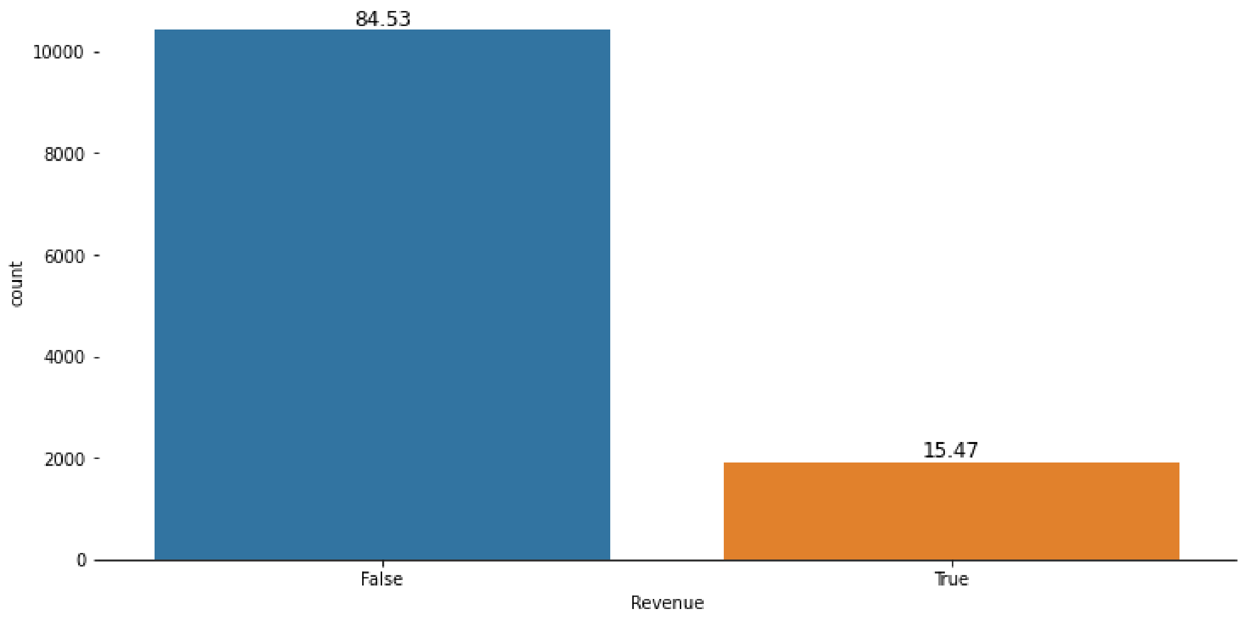
	count	mean	std	min	25%	50%	75%	max
Administrative	12330.0	2.315166	3.321784	0.0	0.000000	1.000000	4.000000	27.000000
Administrative_Duration	12330.0	80.818611	176.779107	0.0	0.000000	7.500000	93.256250	3398.750000
Informational	12330.0	0.503569	1.270156	0.0	0.000000	0.000000	0.000000	24.000000
Informational_Duration	12330.0	34.472398	140.749294	0.0	0.000000	0.000000	0.000000	2549.375000
ProductRelated	12330.0	31.731468	44.475503	0.0	7.000000	18.000000	38.000000	705.000000
ProductRelated_Duration	12330.0	1194.746220	1913.669288	0.0	184.137500	598.936905	1464.157214	63973.522230
BounceRates	12330.0	0.022191	0.048488	0.0	0.000000	0.003112	0.016813	0.200000
ExitRates	12330.0	0.043073	0.048597	0.0	0.014286	0.025156	0.050000	0.200000
PageValues	12330.0	5.889258	18.568437	0.0	0.000000	0.000000	0.000000	361.763742
SpecialDay	12330.0	0.061427	0.198917	0.0	0.000000	0.000000	0.000000	1.000000
OperatingSystems	12330.0	2.124006	0.911325	1.0	2.000000	2.000000	3.000000	8.000000
Browser	12330.0	2.357097	1.717277	1.0	2.000000	2.000000	2.000000	13.000000
Region	12330.0	3.147364	2.401591	1.0	1.000000	3.000000	4.000000	9.000000
TrafficType	12330.0	4.069586	4.025169	1.0	2.000000	2.000000	4.000000	20.000000

Exploratory Data Analysis:

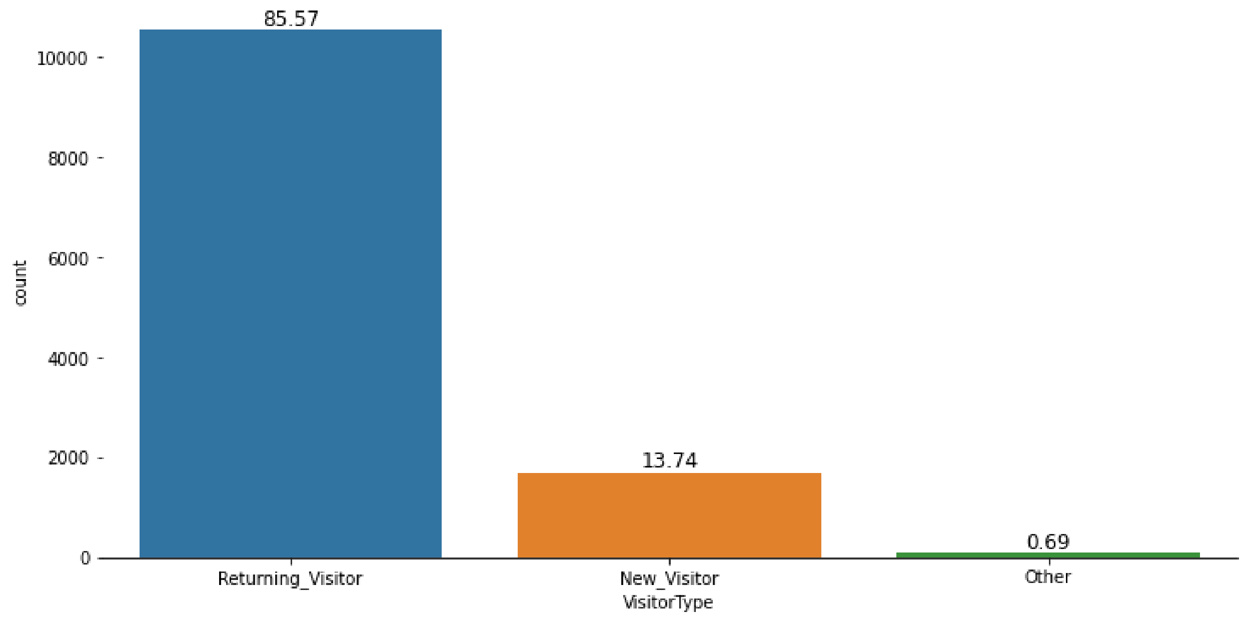
In this portion of our report, we would be focusing on investigating our data further with the help of graphical approaches to find certain patterns and verify our hypothesis. It is a good practice as we will be able to understand our data further before applying various algorithms to it.

Countplot: Univariate Analysis

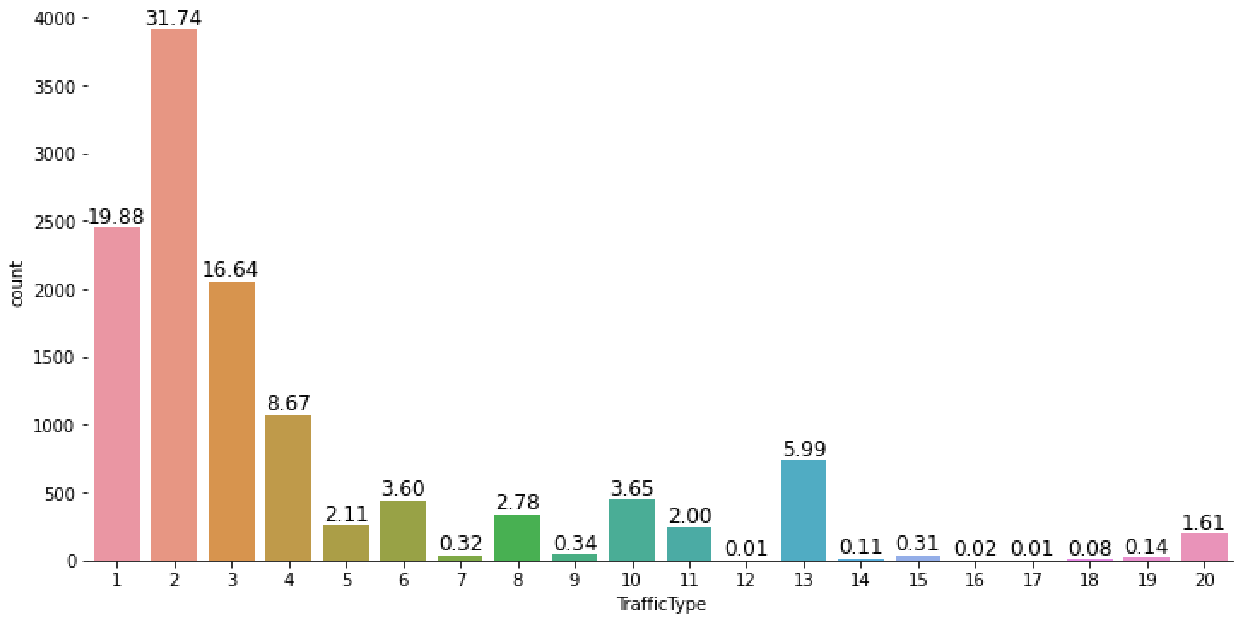
We will first make use of Univariate analysis to find the relation between a single variable and the positive result i.e. whether the customer made a purchase or not.



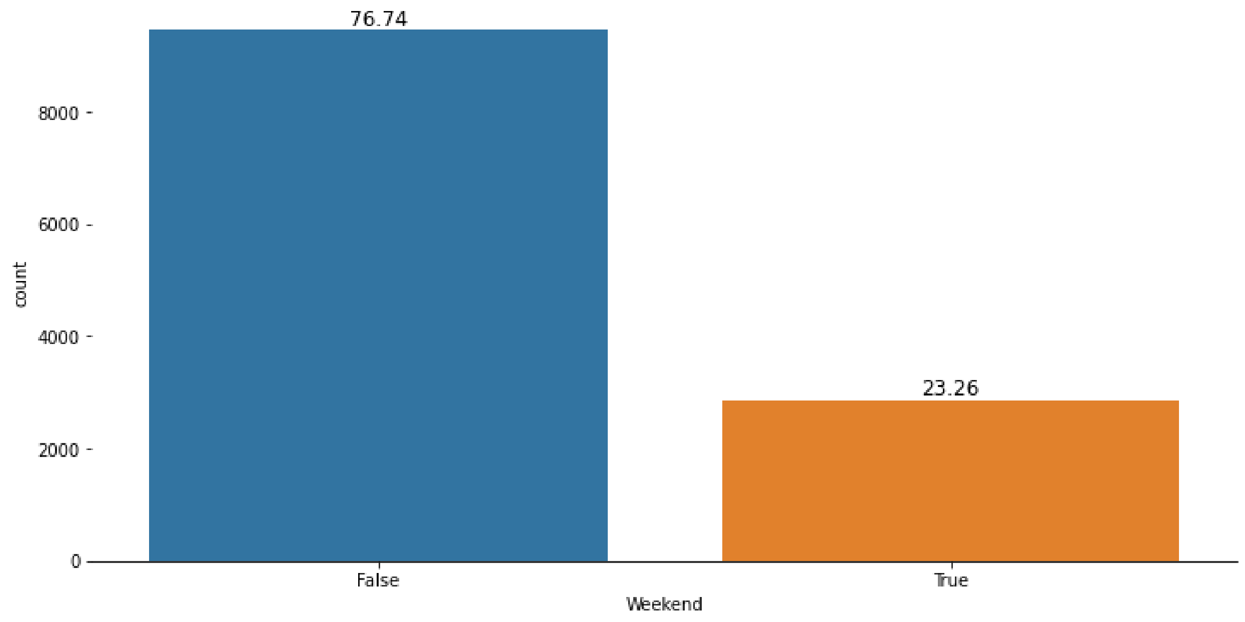
- Only 15.47% percent of customers ended up making the purchase a or more. So, the conversion rate is 15.47%



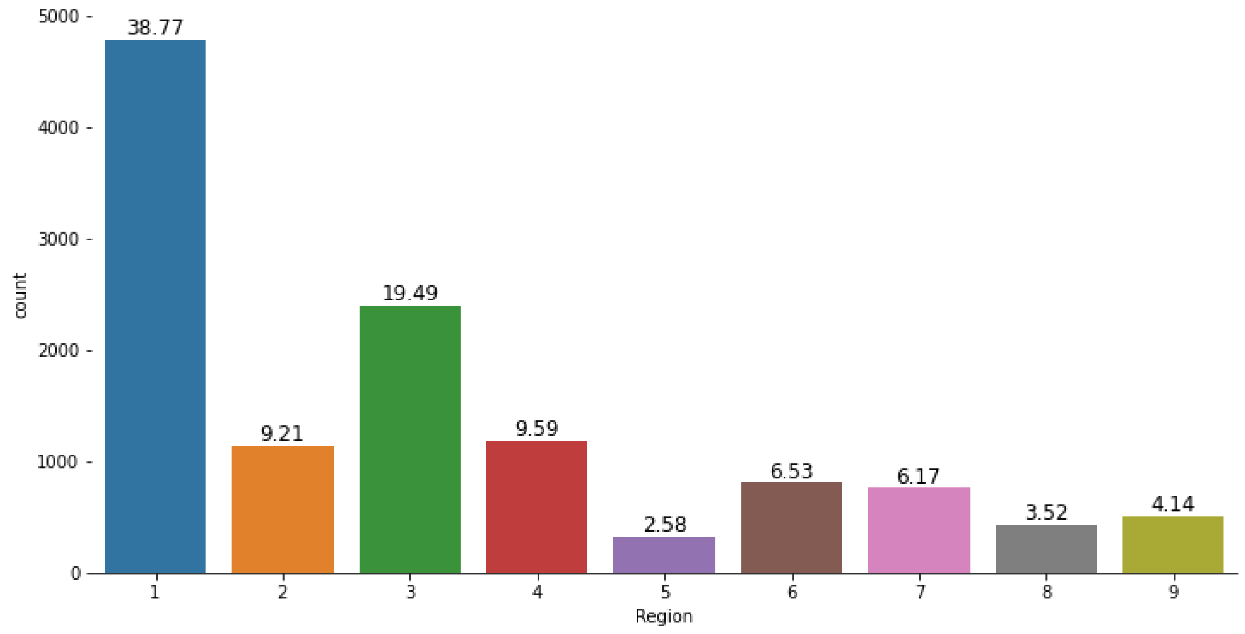
- 85% of customers returns back to the site, which is a good indication. Meaning that our customers are satisfied with what they are getting.
- 13.74% of customers are new customers.



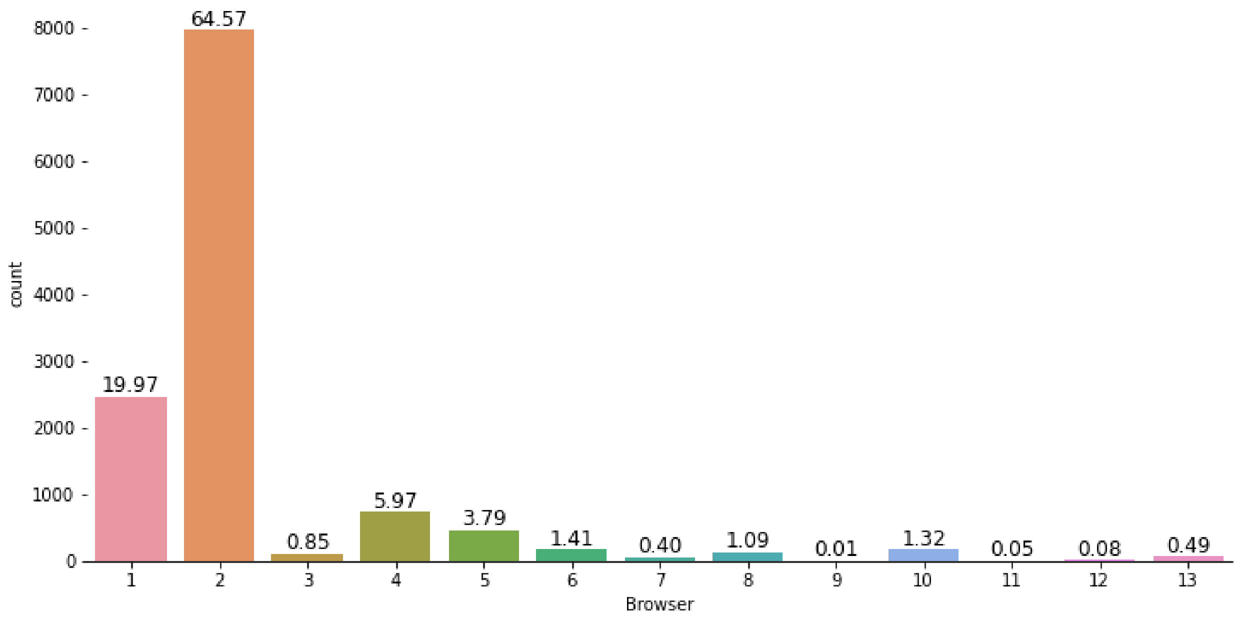
- Most of our visitors are coming from traffic type 2 - around 32%
- Total 83% of visitors coming from 5 traffic type:1,2,3,4, and 13



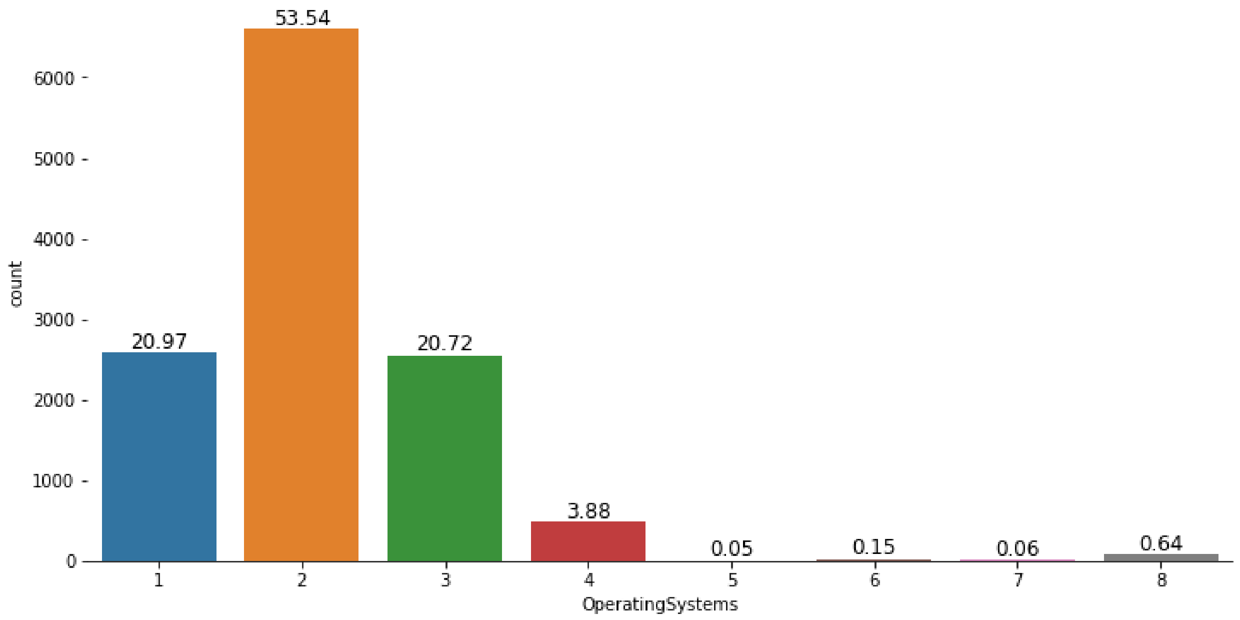
- More visitors visit our site during weekdays than weekends



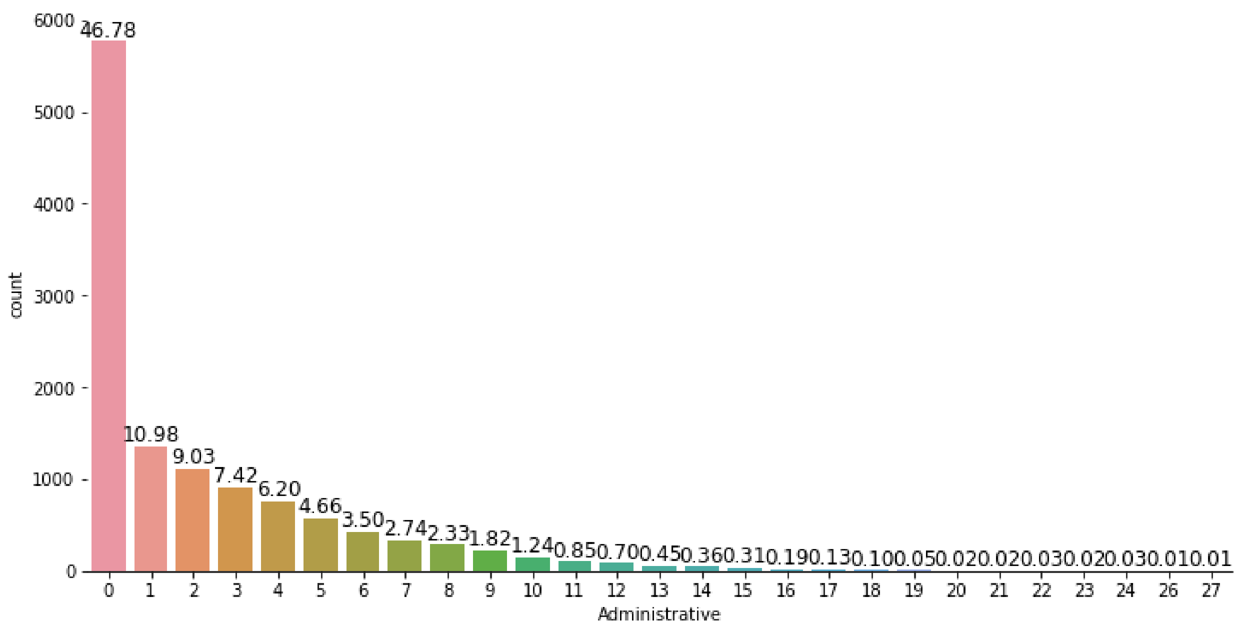
- Around 39% of visitors come from region one followed by 19.5% from region 3



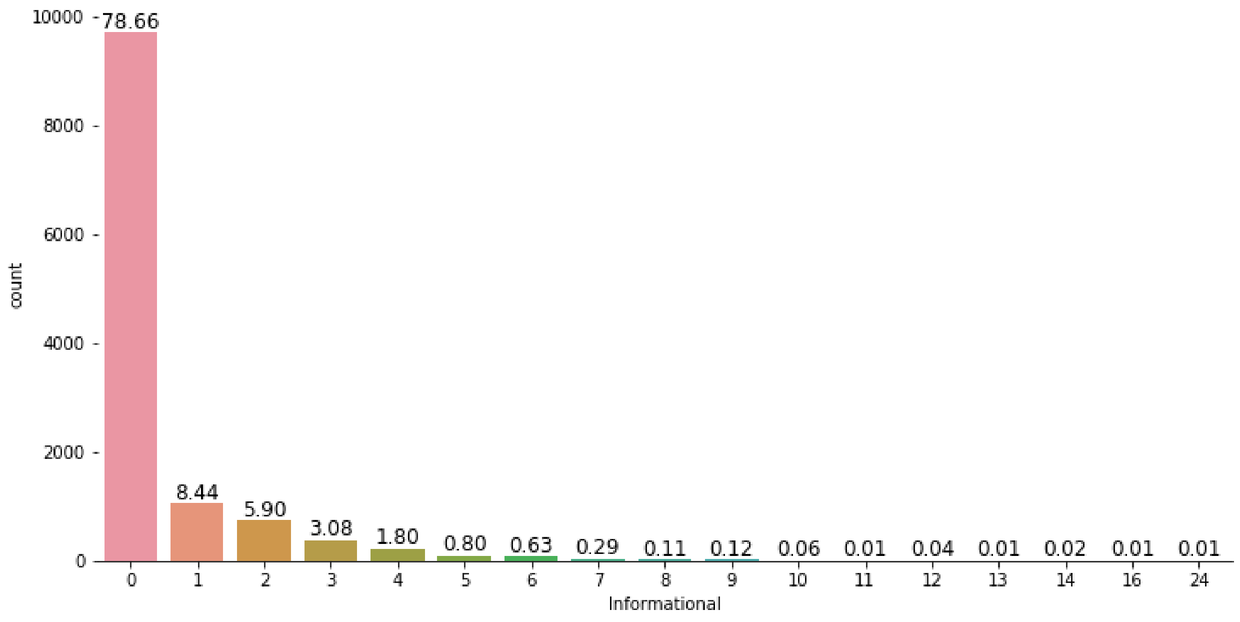
- 64.5% percent of visitors are coming from browser 2 followed by 20% from browser 1.



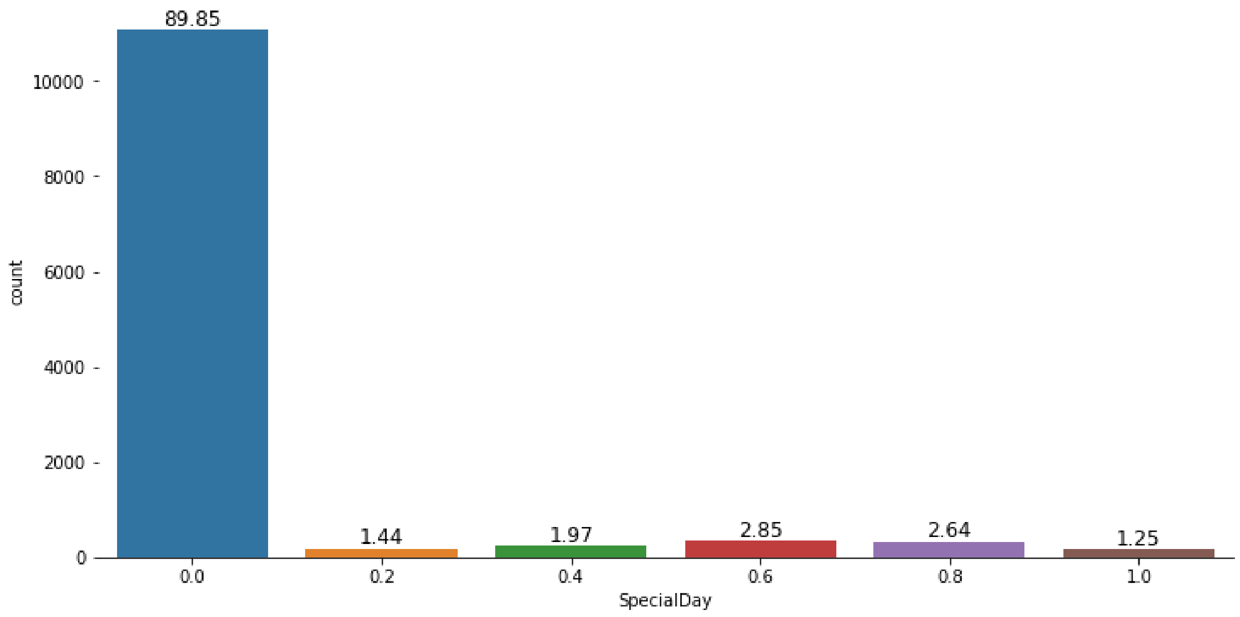
- More than 50% of visitors are using operating system 2.
- Around 95% of visitors are coming from the major three operating systems - os 2 (53.5%), os 1 (21%), os 3 (21%)



- Users tend to visit page 0 the most often.



- We can see that Information page 0 has the highest number of visitors.



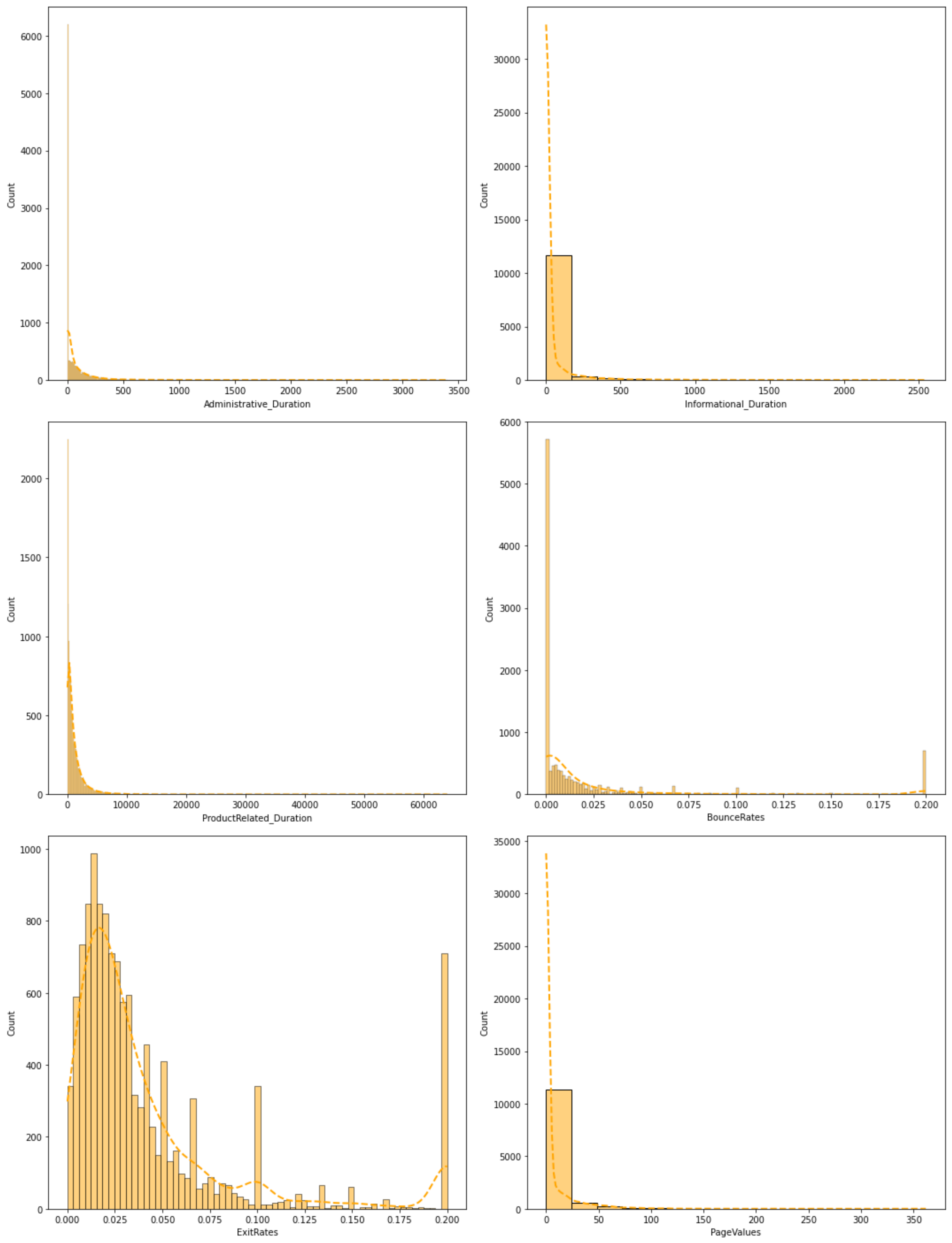
- We can see that special days have no impact on the number of visitors to our website.

Distribution Plots:

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 the probability of individual observations in our dataset. We will be using a distribution plot which can be later helpful for calculating the likelihood of observations.

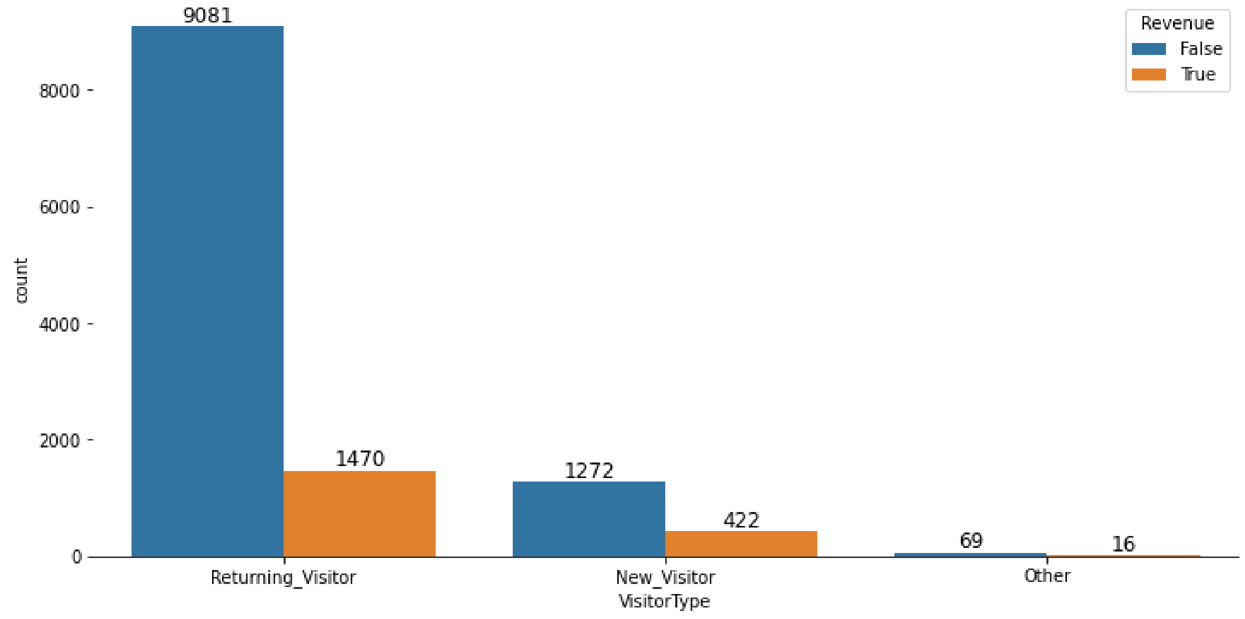
We have not shown the distribution plots for other features such as the category of pages i.e PageValues because they have a high-density function at 0 and very minimal density at other points.

Distribution Plot of Continuous Features

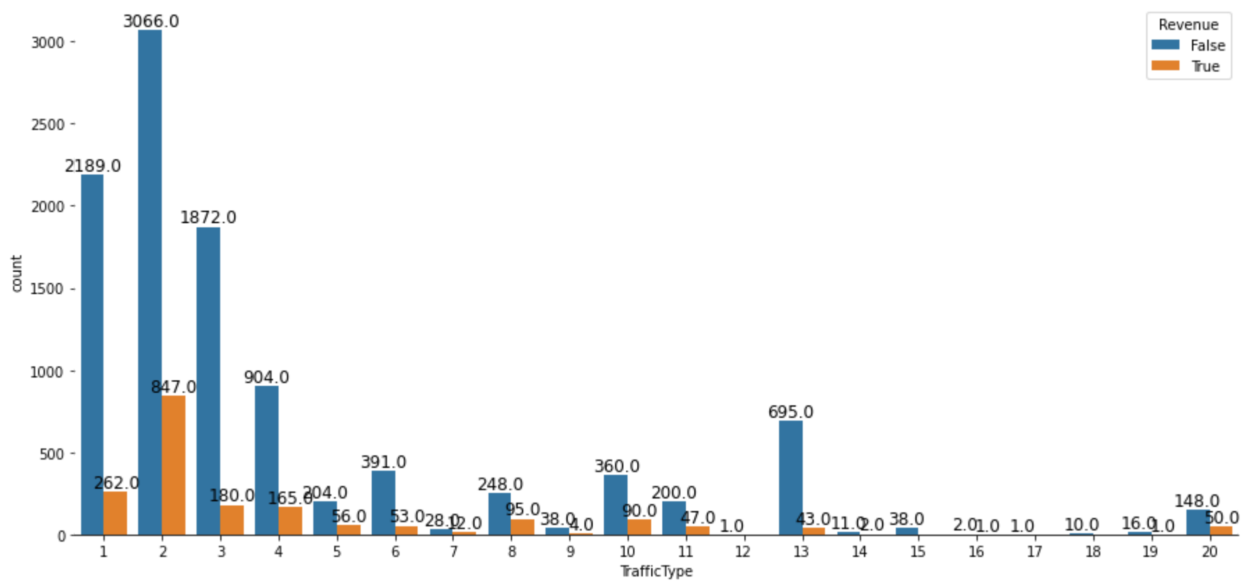


Countplot: Bivariate Analysis

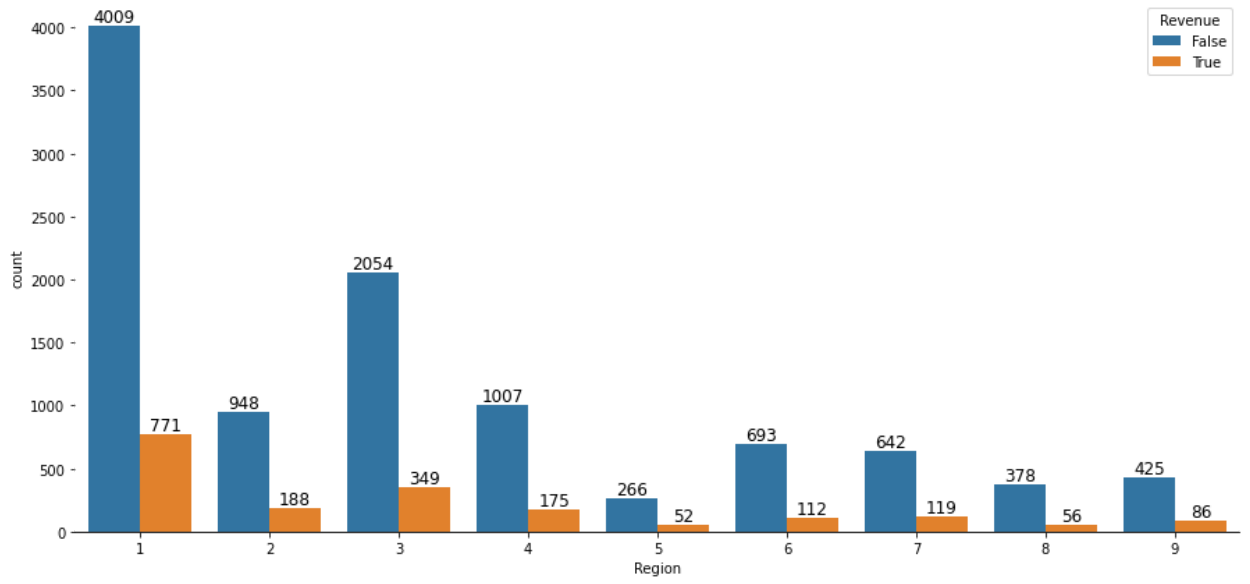
There is a need to evaluate the strength of the relationship between two quantitative variables. This will be done by correlation analysis and bivariate analysis which uses a correlation matrix to give a plot. In this plot, the points of high correlation indicate that the variables have a strong relationship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



- More revenue conversion happens for returning customers than new customers.
- We need to give incentives to new customers to make purchases with us.

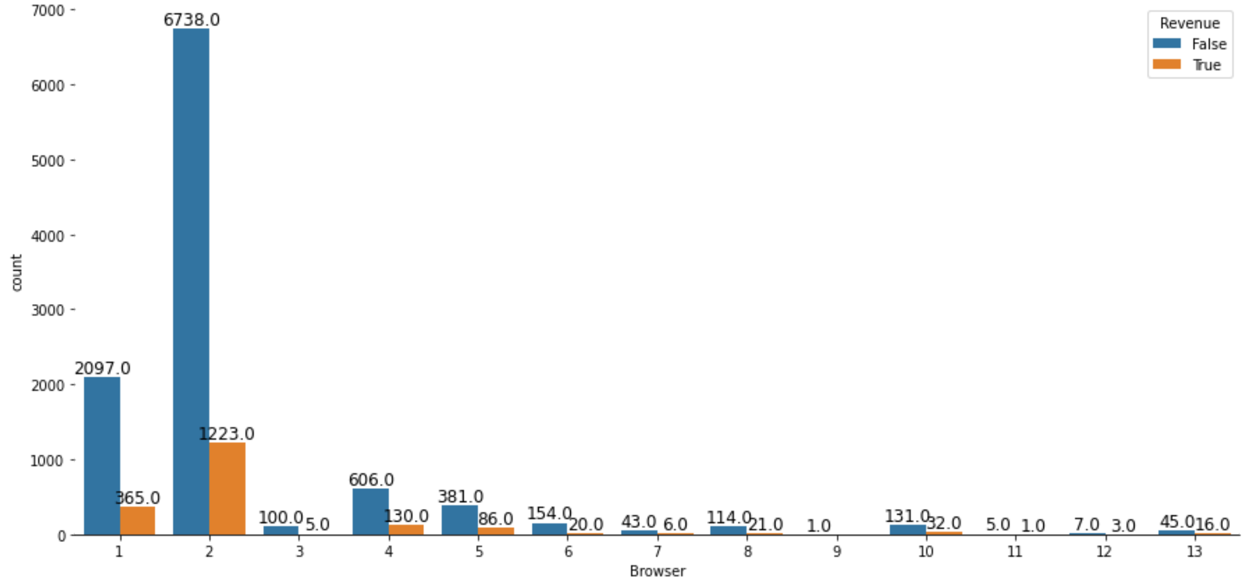


- We can see that more revenue conversion happens for web traffic generated from source 2.
- Even though source 13 generated a considerable amount of web traffic, conversion is very low compared to others.

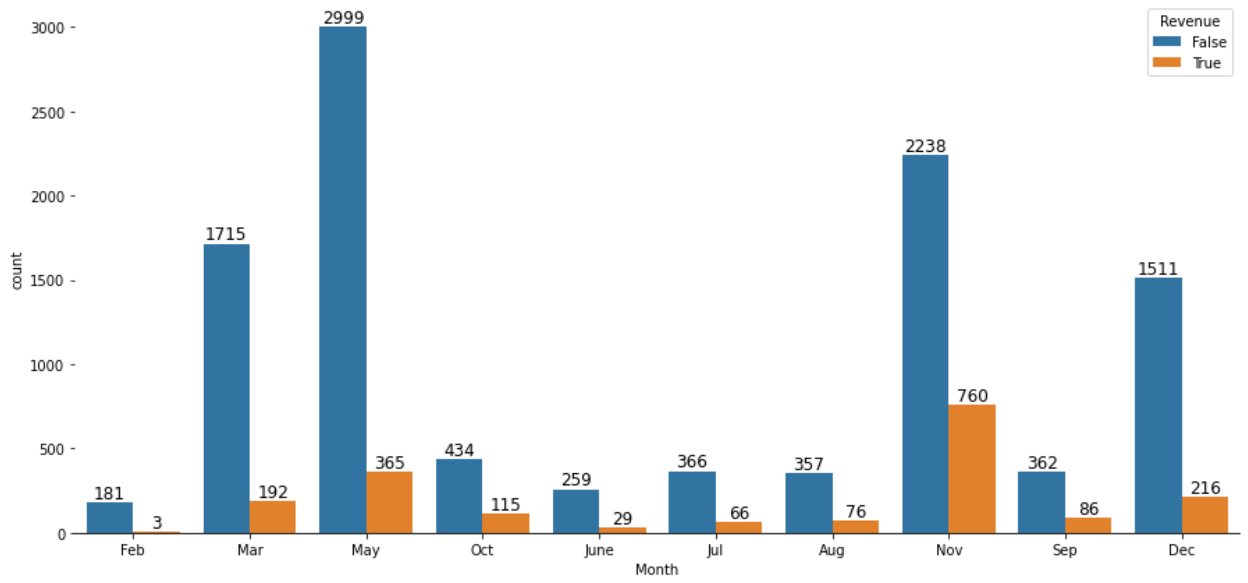


- Region 1 accounts for most sales, and region 3 the second most. With this information, we can plan our marketing and supply chain activities in a better way.
- For example, we might propose building a warehouse specifically catering to the needs of region 1 to increase delivery rates and

ensure that products in the highest demand are always well-stocked.

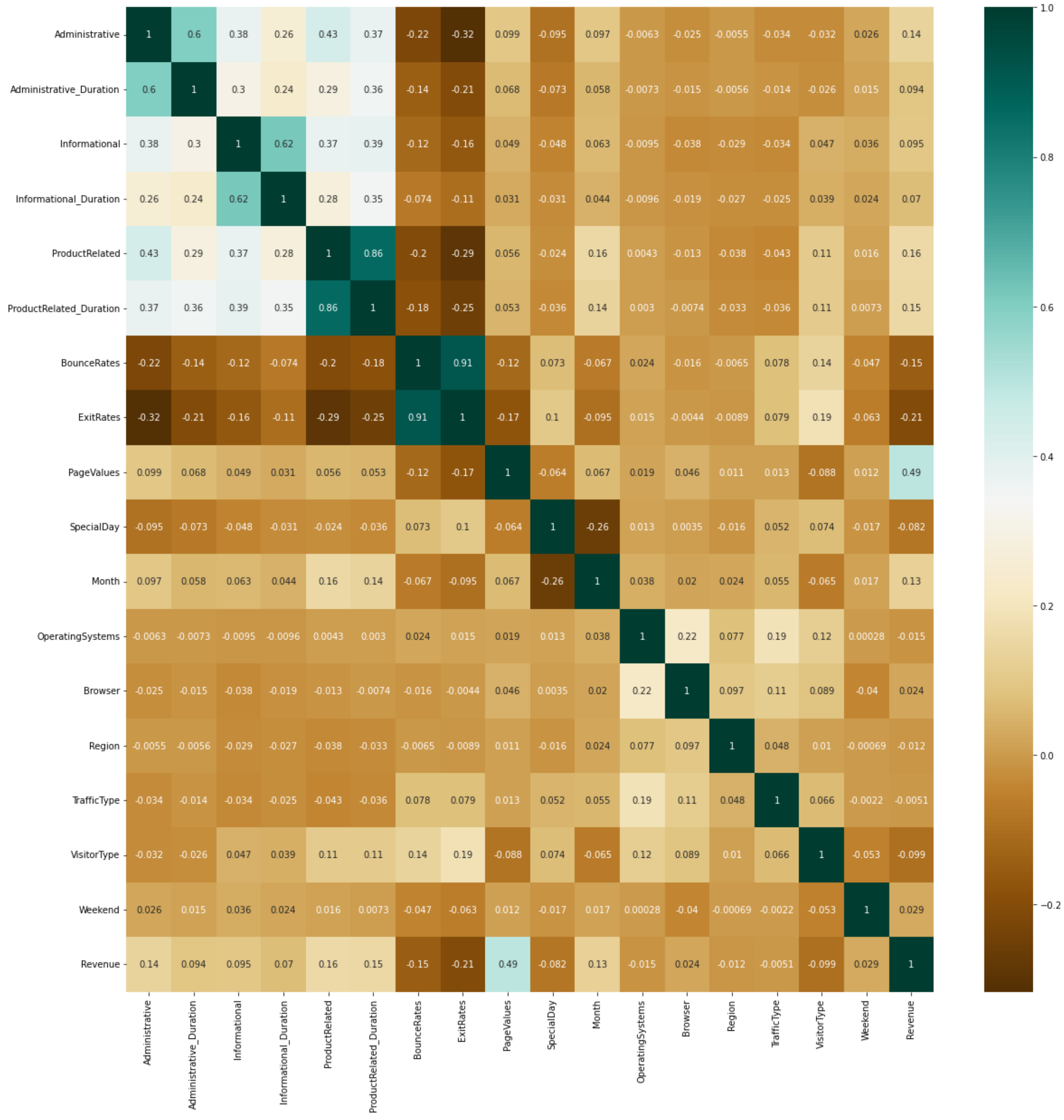


- More revenue-generating transactions have been performed from Browser 2.
- Even though Browser 1 creates a considerable number of sessions, the conversion rate is low.



- Website visitors may be high in May, but we can observe from the preceding bar plot that a greater number of purchases were made in the month of November.

Correlation Matrix:

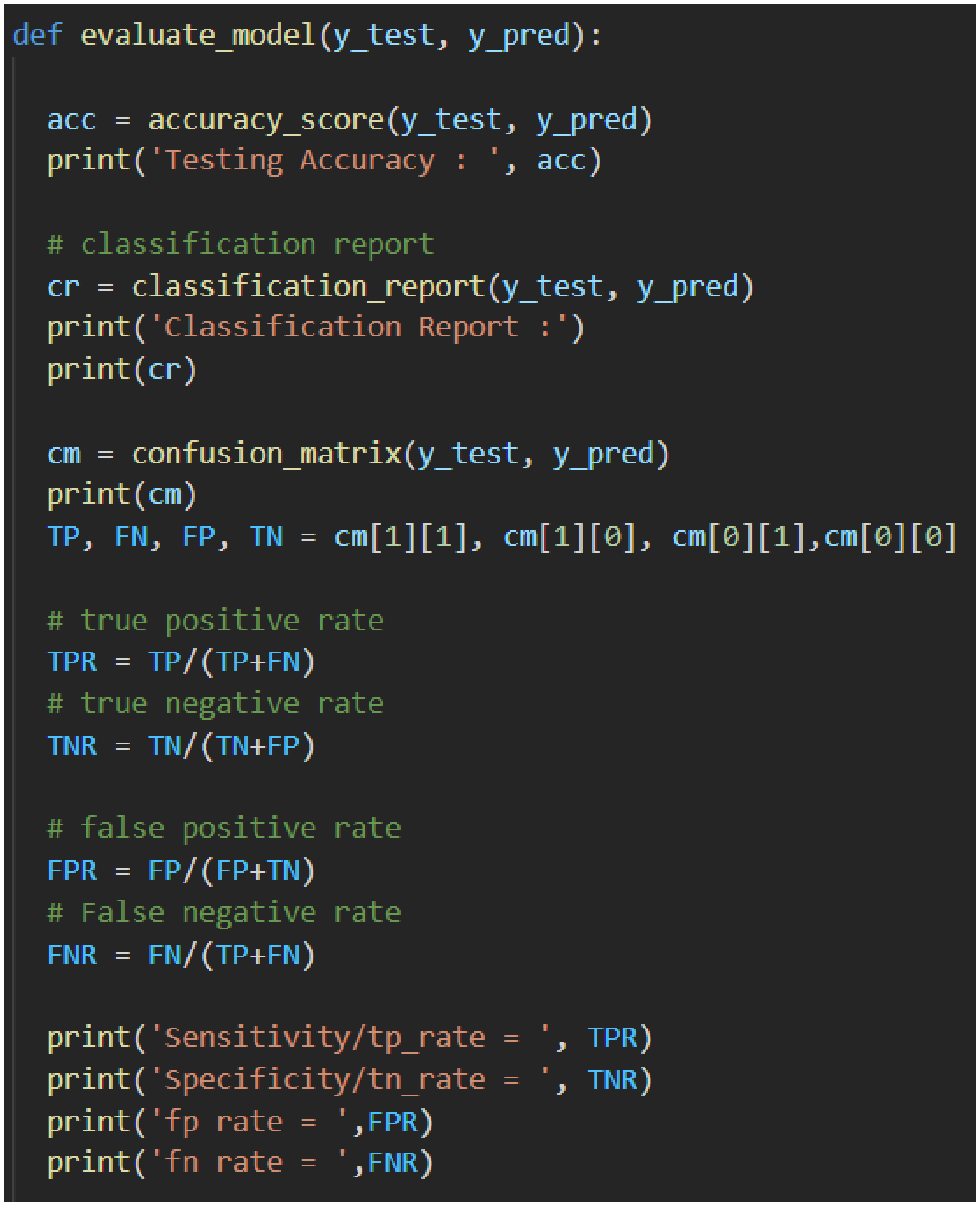


From this correlation plot it can be concluded that even though most of the variables have a low correlation with revenue, Page value has the highest correlation with revenue. We can conclude from this that most of the features in our data set are independent of each other which can be indicated by the low correlation value they have.

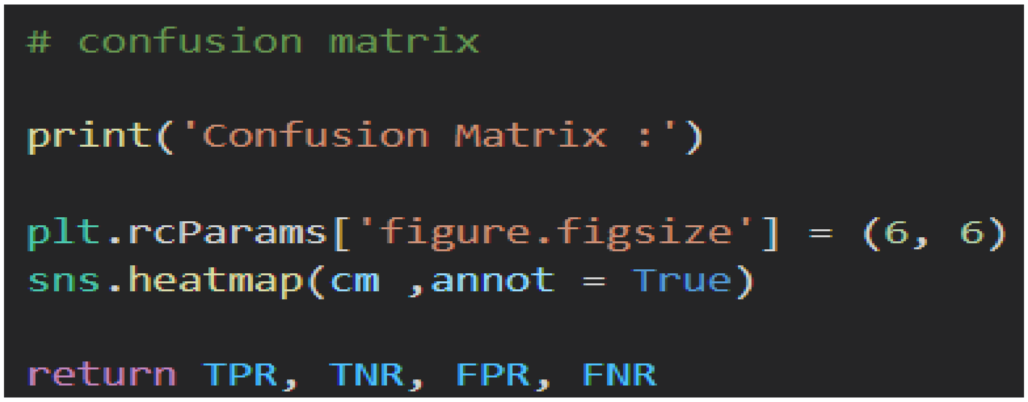
Modeling

Training and Test Data

In machine learning, data need to be separated into 2 sections- training data and testing data (in the case of the 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 the 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%-80% training and 30%-20% 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 a 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 the dataset in a 75%-25% ratio for training and testing respectively



```python
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)
```



```python
# confusion matrix

print('Confusion Matrix :')

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

return TPR, TNR, FPR, FNR
```

Naive Bayes classifier

- Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- It is mainly used in text classification that includes a high-dimensional training dataset.
- Naive Bayes Classifier is one of the simple and most effective Classification algorithms which helps in building the fast machine learning models that can make quick predictions
- we assume that the dataset is independent of every alternative variable such that all options can contribute towards the target data in the model.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.

Bayes theorem:

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

Where

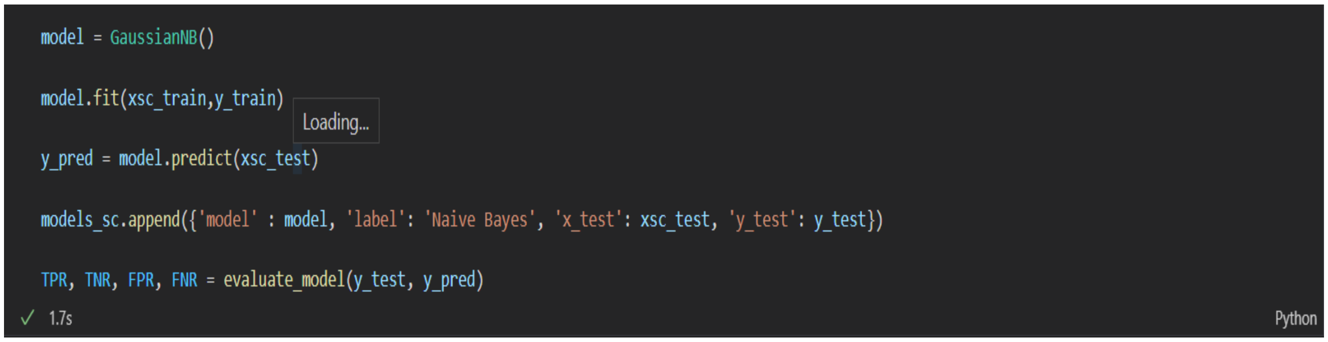
P(A|B) is Posterior probability: Probability of hypothesis A on the observed event B. P(B|A) is Likelihood probability: Probability of the evidence given that the probability of a hypothesis is true.

P(A) ,P(B) are individual probabilities of occurrence of the event

Gaussian: The Gaussian model assumes that features follow a normal distribution. This means if predictors take continuous values instead of discrete, then the model assumes that these values are sampled from the Gaussian distribution.

Steps involved:

- Data Pre-processing step
- Fitting Naive Bayes to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.



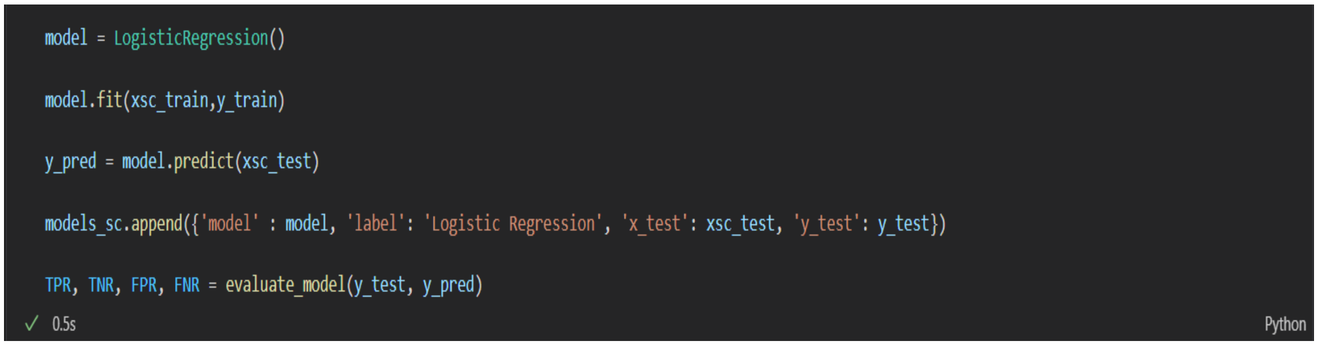
```
model = GaussianNB()

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'Naive Bayes', 'x_test': xsc_test, 'y_test': y_test})

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

Logistic Regression Model

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- itpredicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values which lie between 0 and 1.
- it 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.
- It is a significant machine learning algorithm because it has the ability to provide probabilities and classify new data using continuous and discrete datasets.

Steps involved

- Data Pre-processing step
- Fitting Logistic Regression to the Training set
- Predicting the test result
- Test accuracy of the result(Creation of Confusion matrix)
- Visualizing the test set result.



```
model = LogisticRegression()

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'Logistic Regression', 'x_test': xsc_test, 'y_test': y_test})

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

✓ 0.5s Python

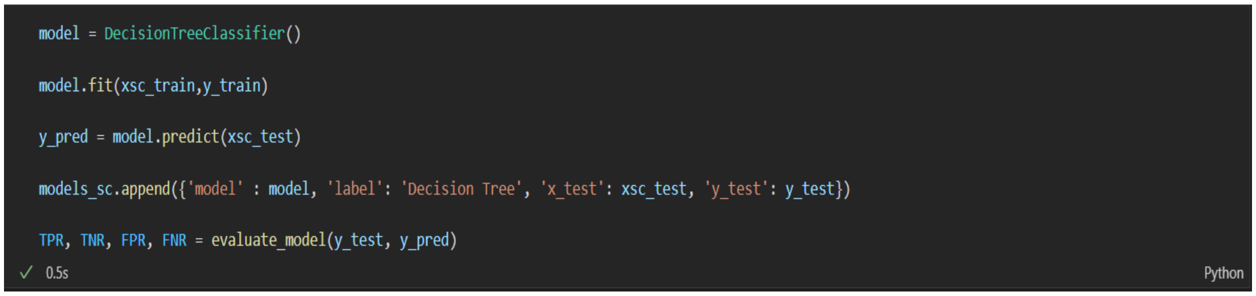
Decision Tree Model

- 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.
- In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- The decisions or the test are performed on the basis of features of the given dataset.
- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.

- It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure.

Steps involved :

- Step-1: Begin the tree with the root node, says S, which contains the complete dataset.
- Step-2: Find the best attribute in the dataset using Attribute Selection Measure (ASM).
- Step-3: Divide the S into subsets that contains possible values for the best attributes.
- Step-4: Generate the decision tree node, which contains the best attribute.
- Step-5: Recursively make new decision trees using the subsets of the dataset created in step -3. Continue this process until a stage is reached where you cannot further classify the nodes and called the final node as a leaf node.



```python
model = DecisionTreeClassifier()

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'Decision Tree', 'x_test': xsc_test, 'y_test': y_test})

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

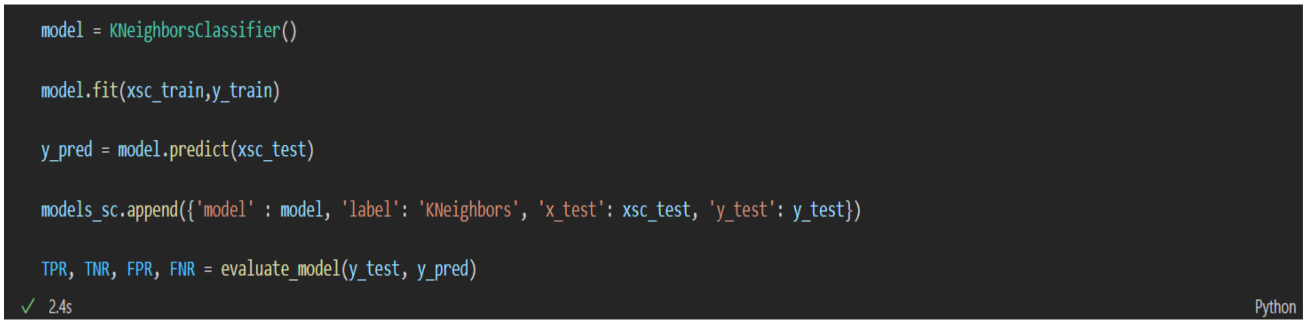
K-Nearest Neighbor(KNN)

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

Steps involved :

- Step-1: Select the number K of the neighbors
- Step-2: Calculate the Euclidean distance of K number of neighbors
- Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.
- Step-4: Among these k neighbors, count the number of the data points in each category.
- Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.
- Step-6: Our model is ready.



```
model = KNeighborsClassifier()

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'KNeighbors', 'x_test': xsc_test, 'y_test': y_test})

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

Random Forest Model

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.
- It is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting
- It takes less training time as compared to other algorithms
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing

Steps involved:

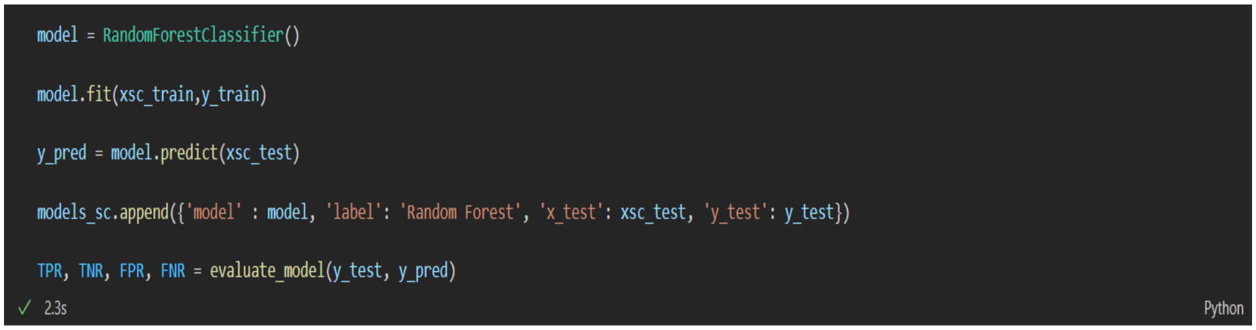
Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes.



```python
model = RandomForestClassifier()

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'Random Forest', 'x_test': xsc_test, 'y_test': y_test})

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

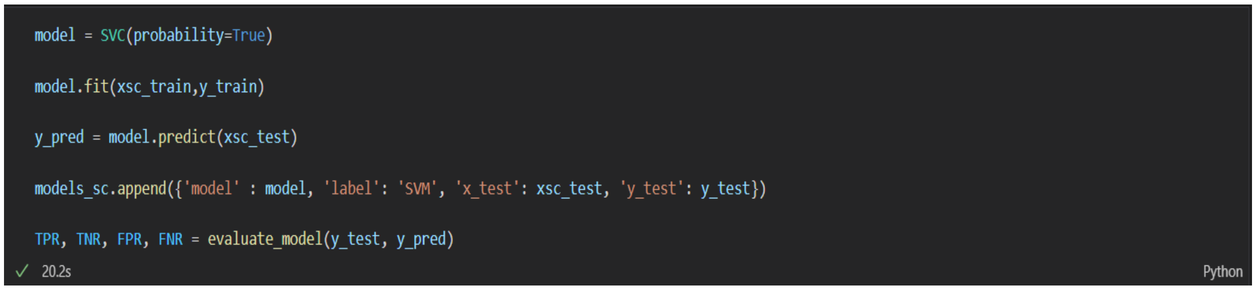
✓ 2.3s Python

Support Vector Machine

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

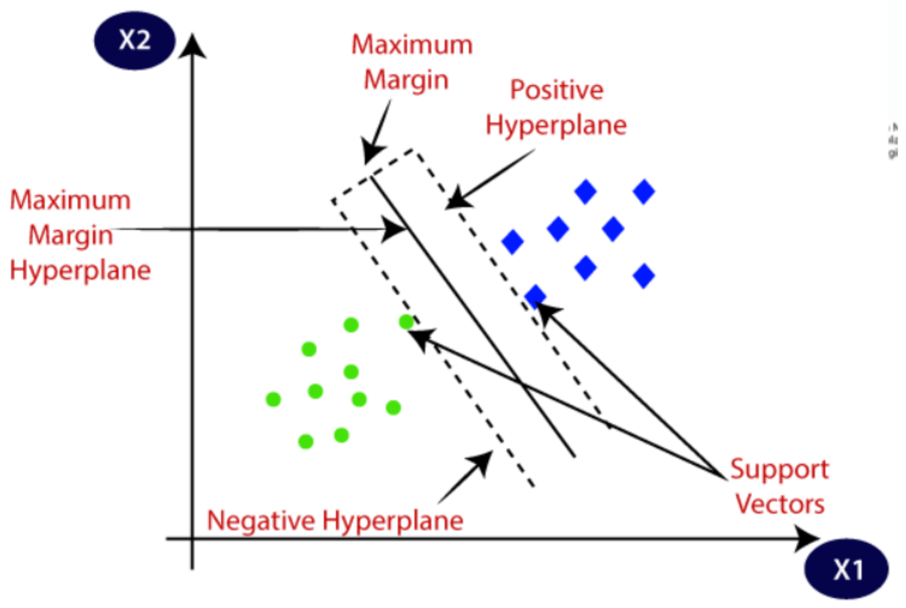
The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.



```python
model = SVC(probability=True)

model.fit(xsc_train,y_train)

y_pred = model.predict(xsc_test)

models_sc.append({'model' : model, 'label': 'SVM', 'x_test': xsc_test, 'y_test': y_test})

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

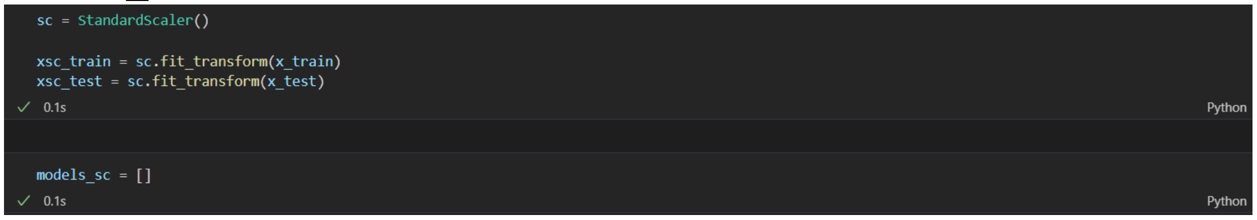
✓ 20.2s Python



The above mention 6 classifiers are the ones which we will be using on our data for the results but first, we will be running these thrice for different types of data to handle normalized, oversampled and undersampled datasets.

Normalized dataset:

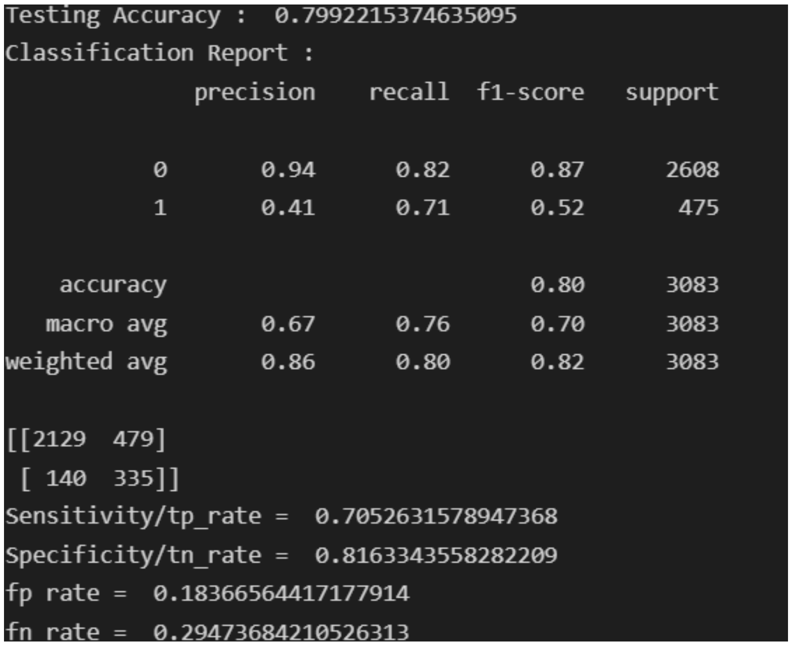
Also known as standardization is a technique often applied as part of data preparation for machine learning. The goal of is to change the values of numeric columns in the dataset to use a common scale. Basically we are scaling by a factor without disturbing the ranges of values or losing information. We first defined a Standardscalar and then used fit_transform to normalize our data.



```
sc = StandardScaler()

xsc_train = sc.fit_transform(x_train)
xsc_test = sc.fit_transform(x_test)
```
✓ 0.1s Python

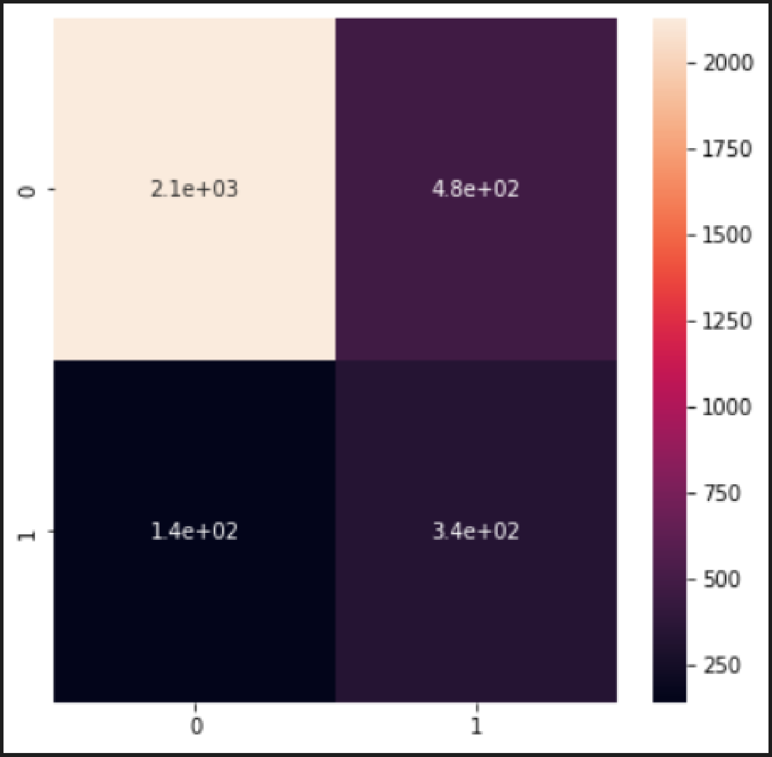
```
models_sc = []
```
✓ 0.1s Python

(a)Naive Bayes

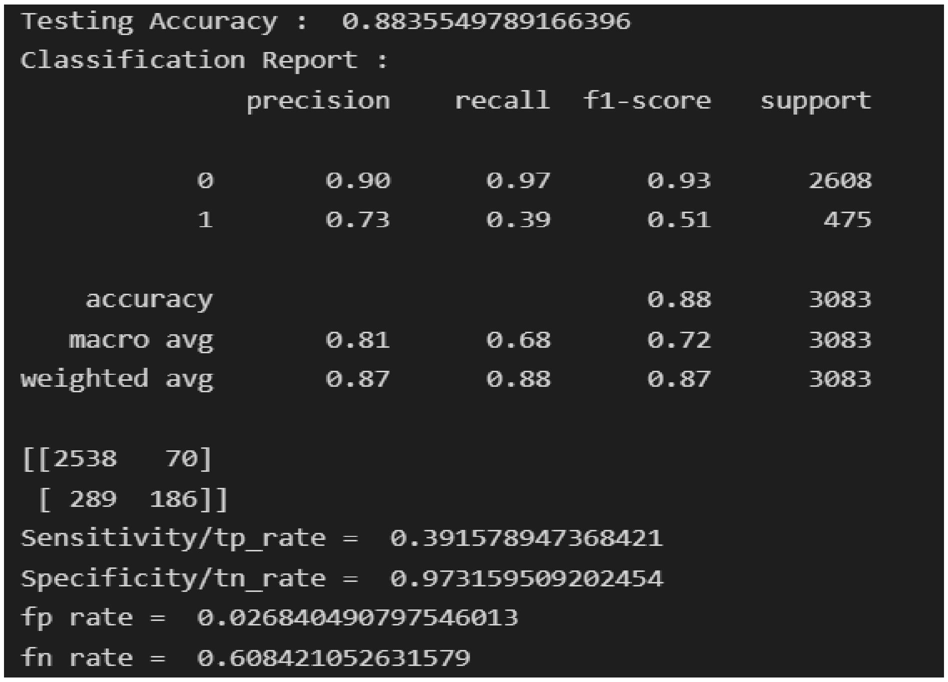


```
Testing Accuracy :  0.7992215374635095
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.82      0.87      2608
           1       0.41      0.71      0.52       475

    accuracy                           0.80      3083
   macro avg       0.67      0.76      0.70      3083
weighted avg       0.86      0.80      0.82      3083

[[2129  479]
 [ 140  335]]
Sensitivity/tp_rate =  0.7052631578947368
Specificity/tn_rate =  0.8163343558282209
fp rate =  0.18366564417177914
fn rate =  0.29473684210526313
```

Confusion Matrix :

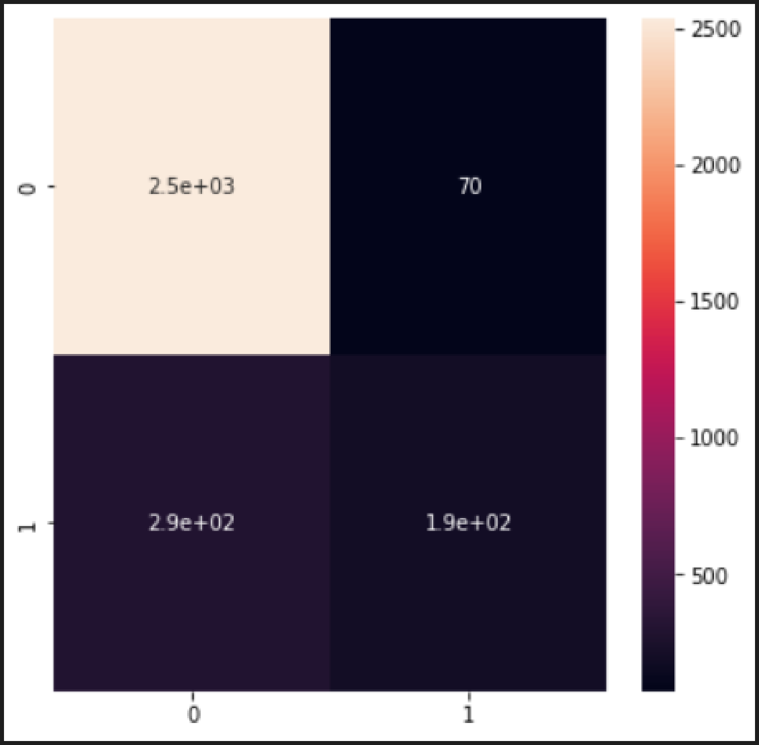


(b)Logistic Regression

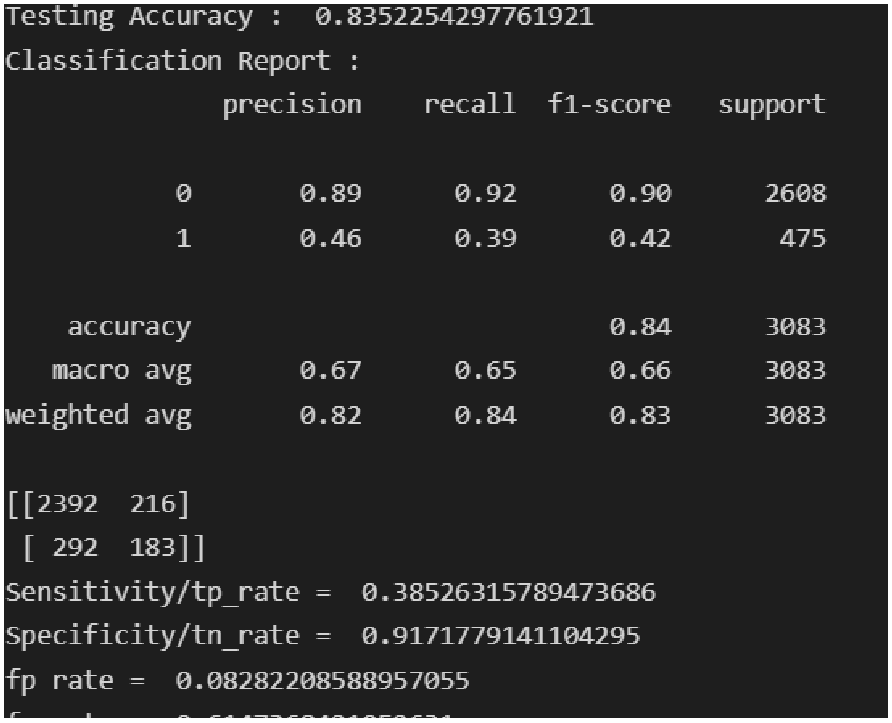


```
Testing Accuracy :  0.8835549789166396
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.97      0.93      2608
           1       0.73      0.39      0.51       475

    accuracy                           0.88      3083
   macro avg       0.81      0.68      0.72      3083
weighted avg       0.87      0.88      0.87      3083

[[2538   70]
 [ 289  186]]
Sensitivity/tp_rate =  0.391578947368421
Specificity/tn_rate =  0.973159509202454
fp rate =  0.026840490797546013
fn rate =  0.608421052631579
```

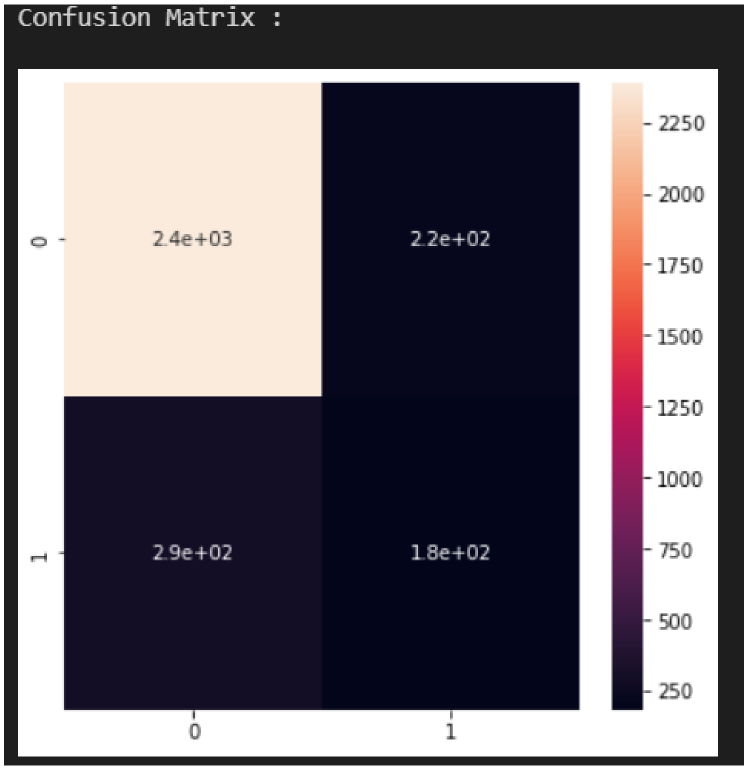
Confusion Matrix :



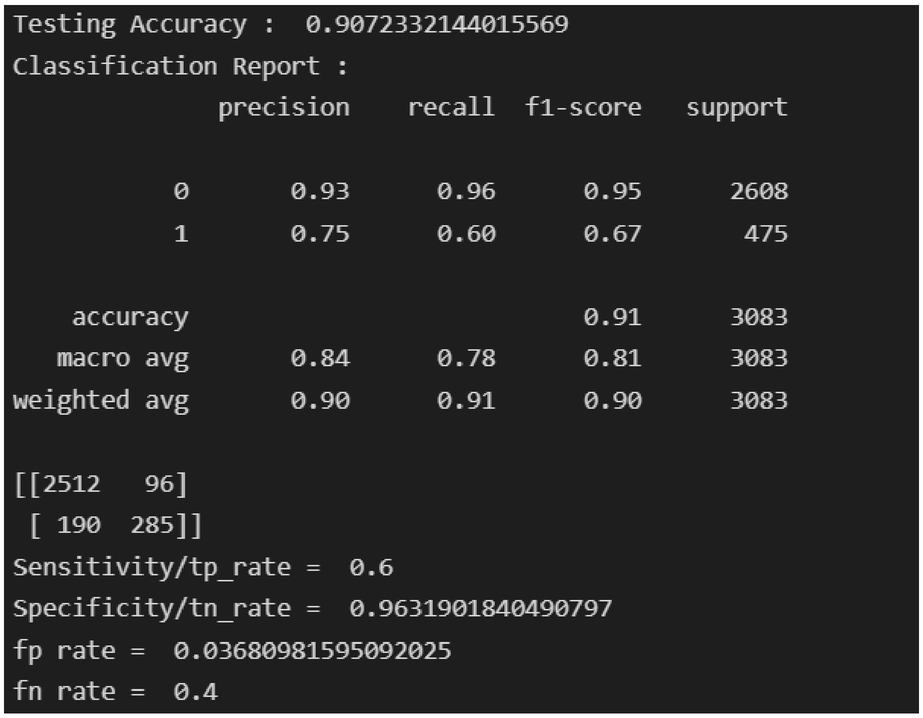
(c)Decision Tree



```
Testing Accuracy :  0.8352254297761921
Classification Report :
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      2608
           1       0.46      0.39      0.42       475

    accuracy                           0.84      3083
   macro avg       0.67      0.65      0.66      3083
weighted avg       0.82      0.84      0.83      3083

[[2392  216]
 [ 292  183]]
Sensitivity/tp_rate =  0.38526315789473686
Specificity/tn_rate =  0.9171779141104295
fp rate =  0.08282208588957055
```

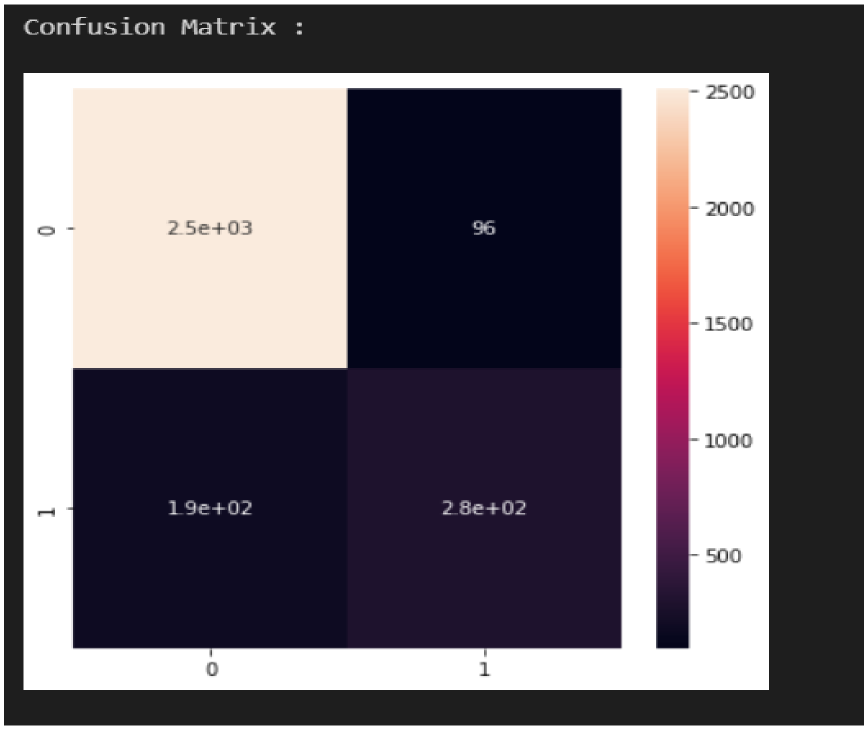


(d)Random Forest



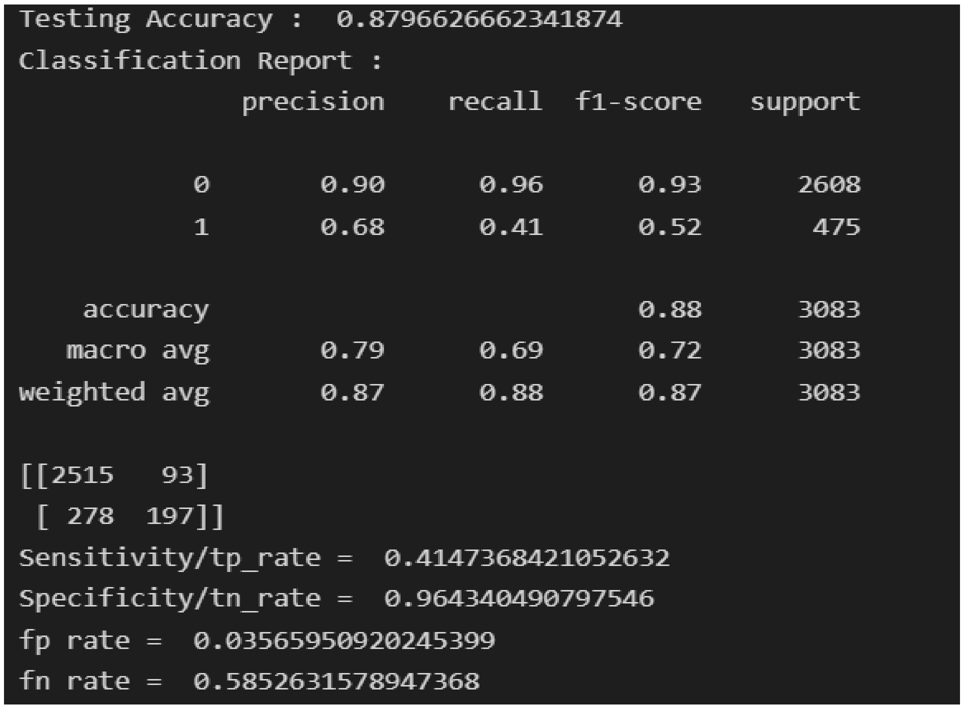
```
Testing Accuracy :  0.9072332144015569
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.96      0.95      2608
           1       0.75      0.60      0.67       475

    accuracy                           0.91      3083
   macro avg       0.84      0.78      0.81      3083
weighted avg       0.90      0.91      0.90      3083

[[2512   96]
 [ 190  285]]
Sensitivity/tp_rate =  0.6
Specificity/tn_rate =  0.9631901840490797
fp rate =  0.03680981595092025
fn rate =  0.4
```

,



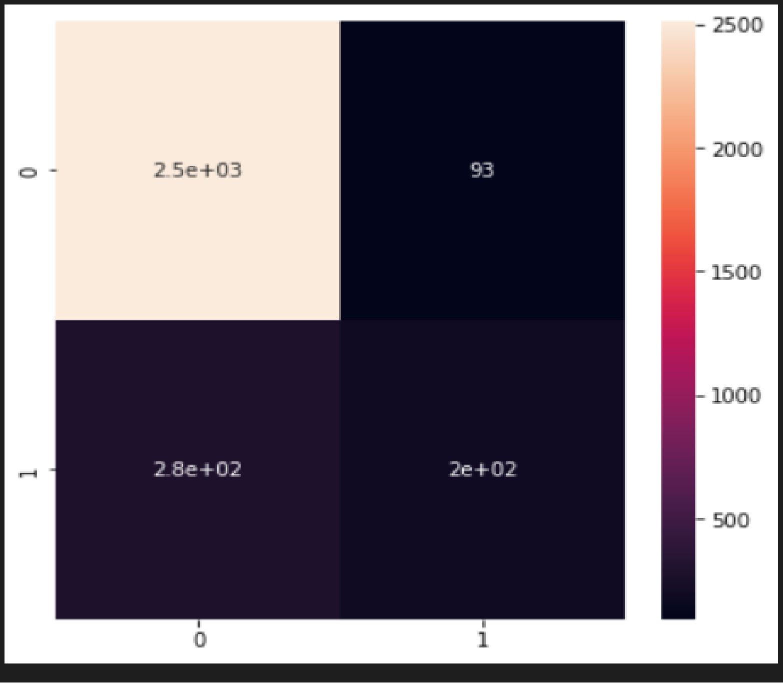
Confusion Matrix :

(e)K-Nearest Neighbor

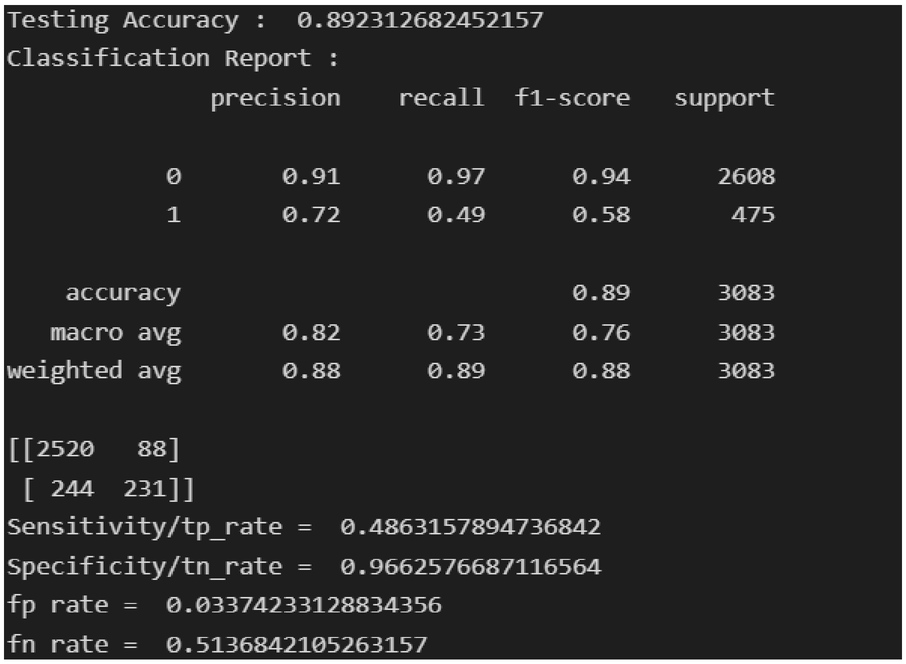


```
Testing Accuracy :  0.8796626662341874
Classification Report :
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      2608
           1       0.68      0.41      0.52       475

    accuracy                           0.88      3083
   macro avg       0.79      0.69      0.72      3083
weighted avg       0.87      0.88      0.87      3083

[[2515   93]
 [ 278  197]]
Sensitivity/tp_rate =  0.4147368421052632
Specificity/tn_rate =  0.964340490797546
fp rate =  0.03565950920245399
fn rate =  0.5852631578947368
```

Confusion Matrix :

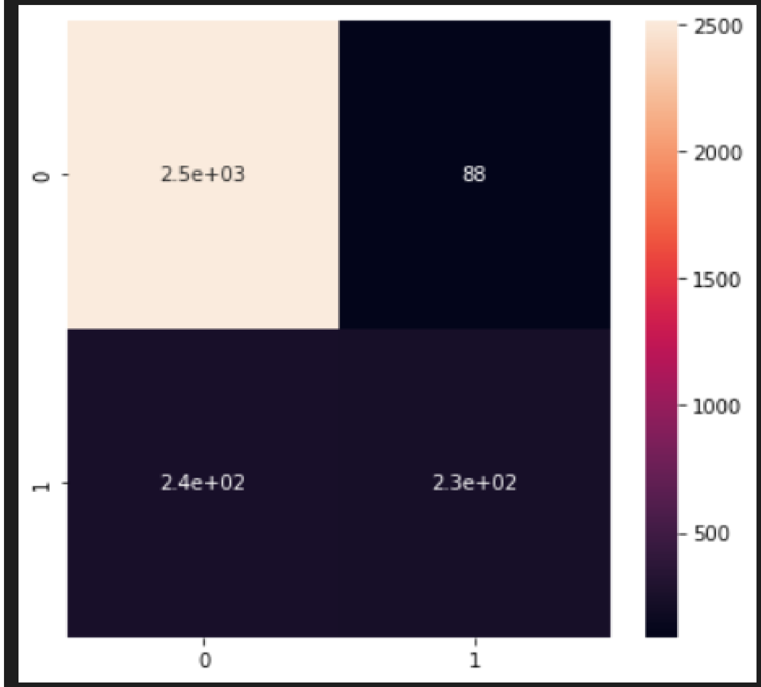


(f)Support Vector Machine



```
Testing Accuracy :  0.892312682452157
Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.97      0.94      2608
           1       0.72      0.49      0.58       475

    accuracy                           0.89      3083
   macro avg       0.82      0.73      0.76      3083
weighted avg       0.88      0.89      0.88      3083

[[2520   88]
 [ 244  231]]
Sensitivity/tp_rate =  0.4863157894736842
Specificity/tn_rate =  0.9662576687116564
fp rate =  0.03374233128834356
fn rate =  0.5136842105263157
```

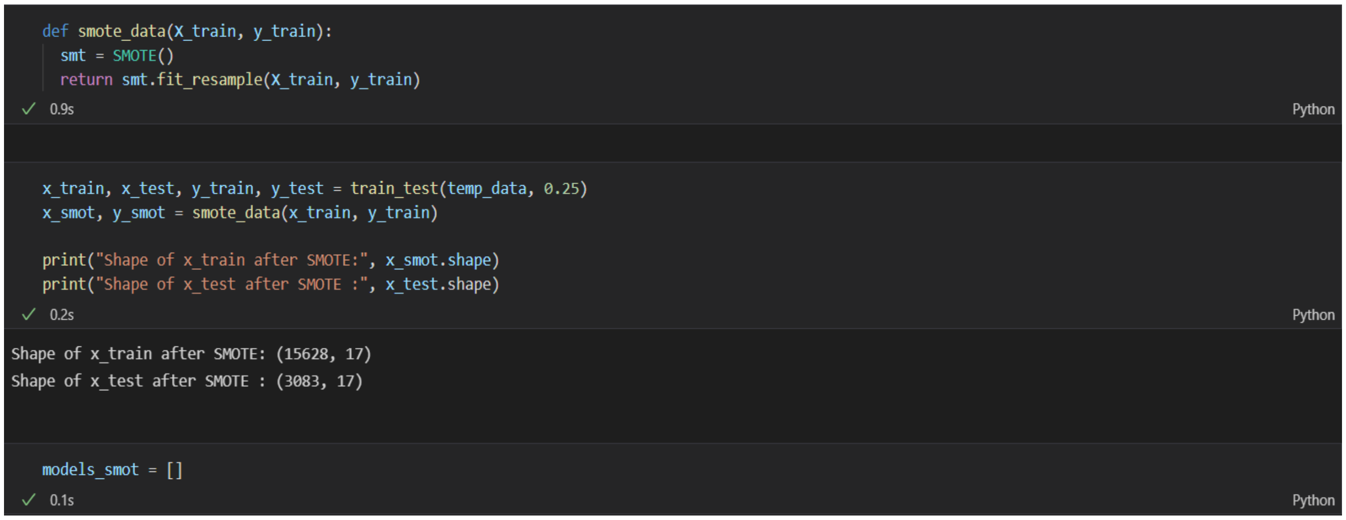
Confusion Matrix :



Oversampled dataset:

For Oversampled dataset, we use SMOTE algorithm.

SMOTE stands for Synthetic Minority Oversampling Technique.This is a widely used technique to balance the training set before the learning phase to deal with the imbalance problem 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 in duplicating examples from the minority class is to synthesize new examples from the minority class. It can be grouped under oversampling algorithm and can be very effective way to balance the data.



```
def smote_data(X_train, y_train):
    smt = SMOTE()
    return smt.fit_resample(X_train, y_train)
```

✓ 0.9s Python

```
x_train, x_test, y_train, y_test = train_test(temp_data, 0.25)
x_smot, y_smot = smote_data(x_train, y_train)

print("Shape of x_train after SMOTE:", x_smot.shape)
print("Shape of x_test after SMOTE :", x_test.shape)
```

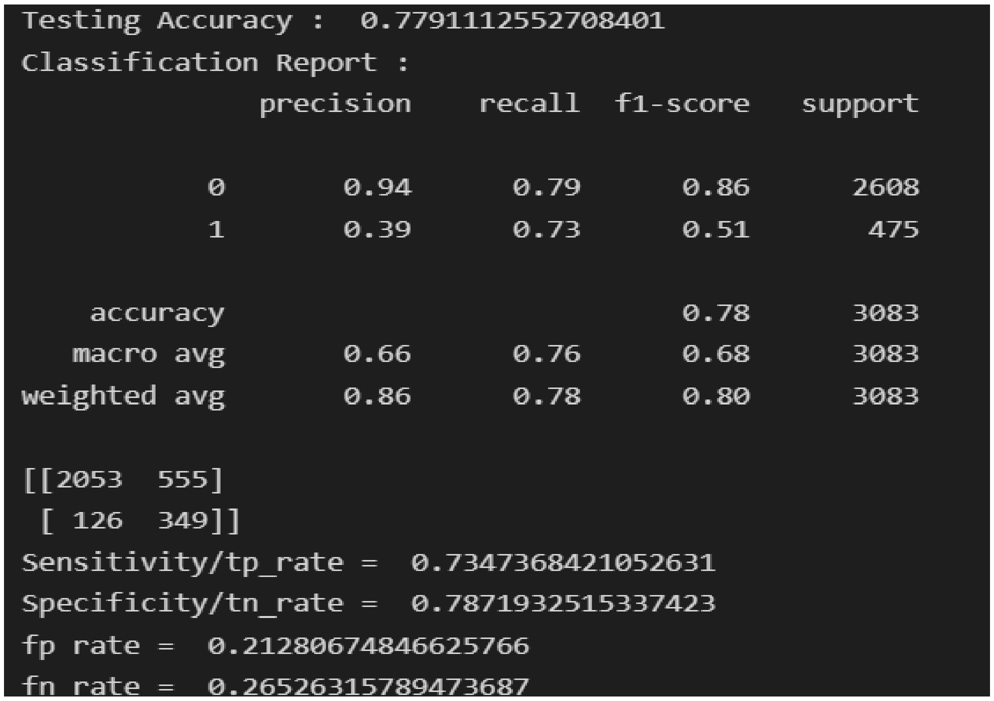
✓ 0.2s Python

```
Shape of x_train after SMOTE: (15628, 17)
Shape of x_test after SMOTE : (3083, 17)
```

```
models_smot = []
```

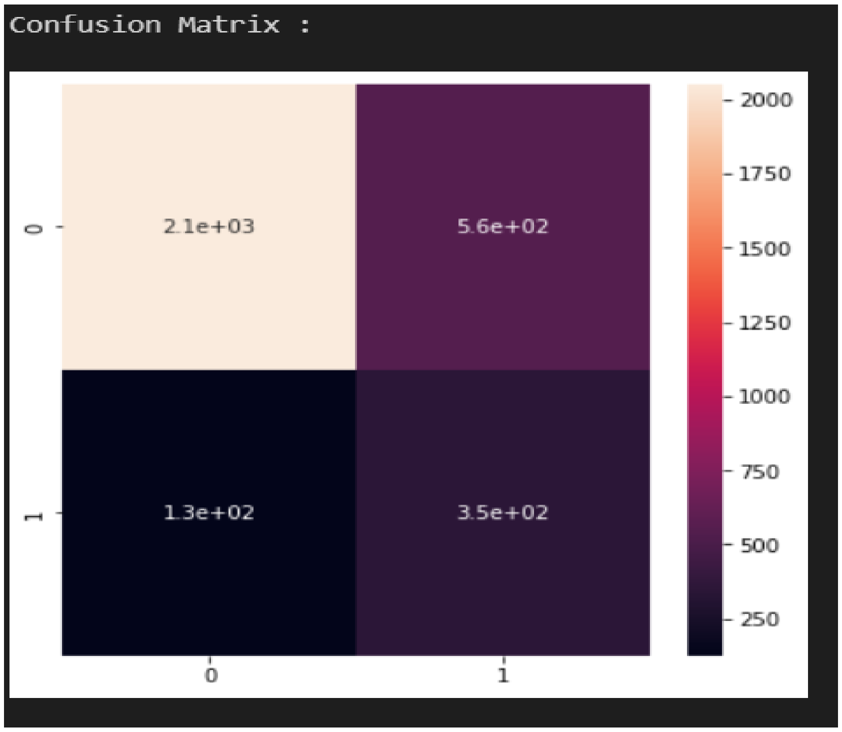
✓ 0.1s Python

(a)Naive Bayes

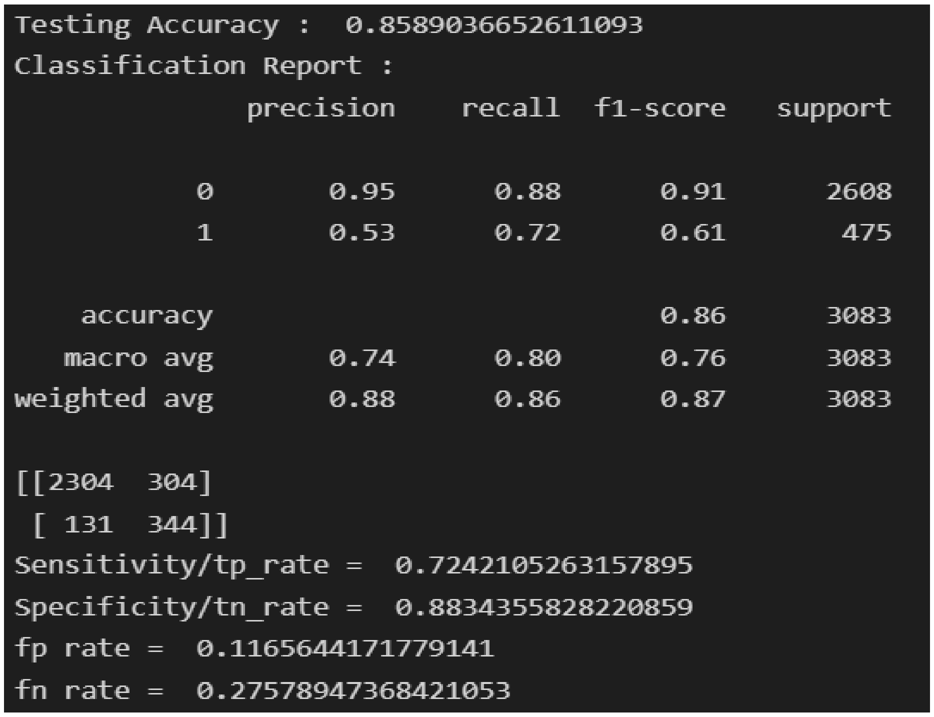


```
Testing Accuracy :  0.7791112552708401
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.79      0.86      2608
           1       0.39      0.73      0.51       475

    accuracy                           0.78      3083
   macro avg       0.66      0.76      0.68      3083
weighted avg       0.86      0.78      0.80      3083

[[2053  555]
 [ 126  349]]
Sensitivity/tp_rate =  0.7347368421052631
Specificity/tn_rate =  0.7871932515337423
fp rate =  0.21280674846625766
fn rate =  0.26526315789473687
```

Confusion Matrix :

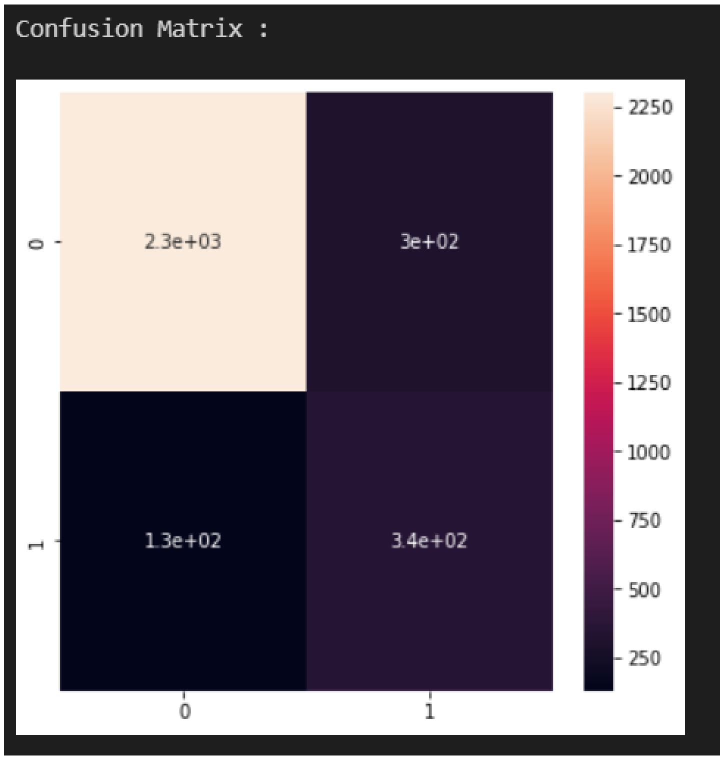


(b)Logistic Regression

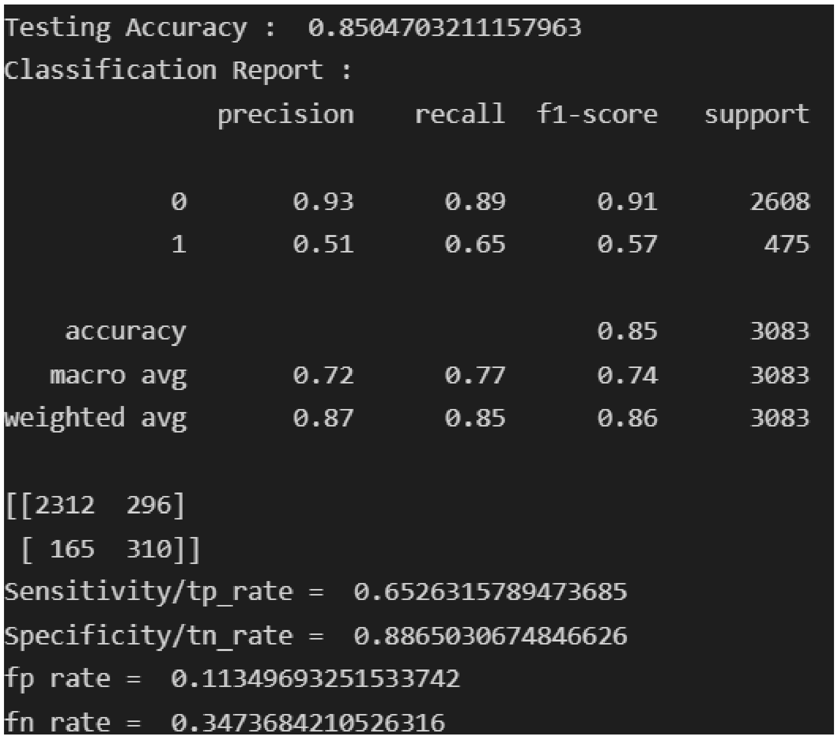


```
Testing Accuracy :  0.8589036652611093
Classification Report :
              precision    recall  f1-score   support

           0       0.95      0.88      0.91      2608
           1       0.53      0.72      0.61       475

    accuracy                           0.86      3083
   macro avg       0.74      0.80      0.76      3083
weighted avg       0.88      0.86      0.87      3083

[[2304  304]
 [ 131  344]]
Sensitivity/tp_rate =  0.7242105263157895
Specificity/tn_rate =  0.8834355828220859
fp rate =  0.1165644171779141
fn rate =  0.27578947368421053
```

Confusion Matrix :

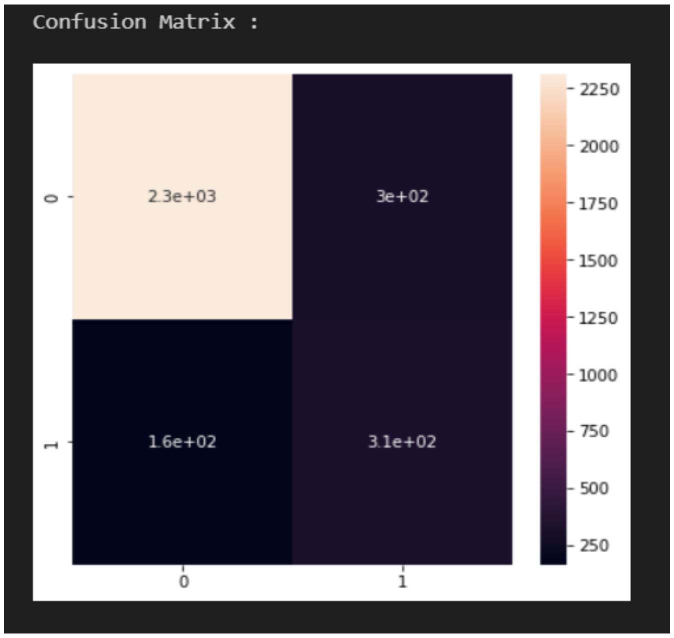


(c)Decision Tree

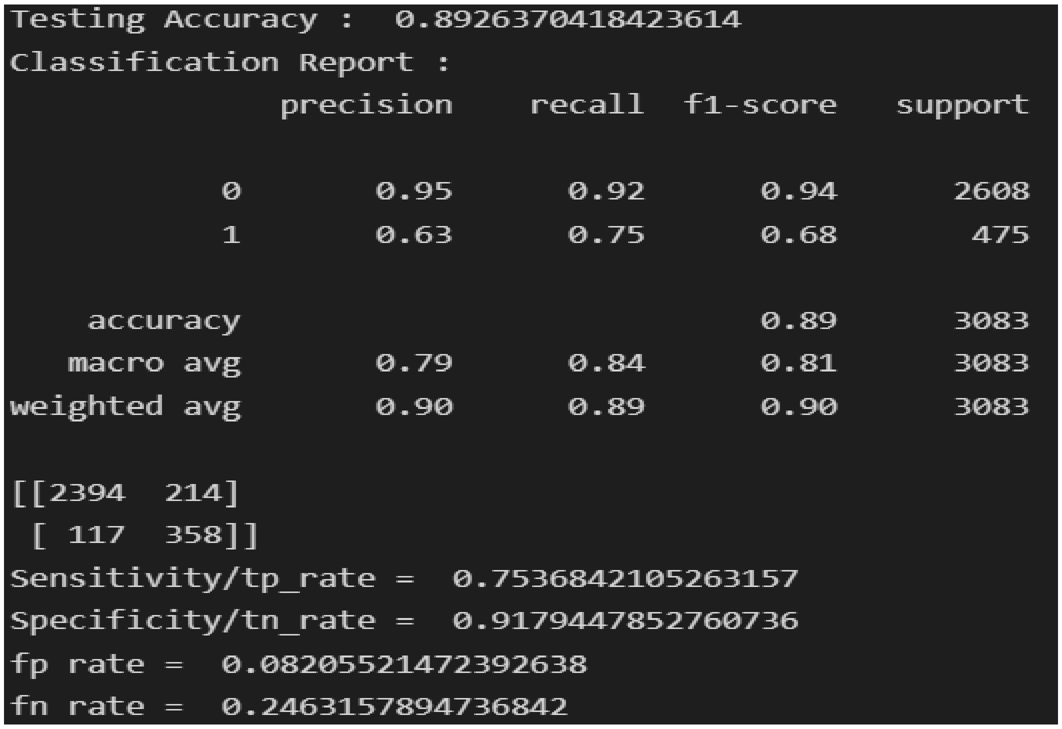


```
Testing Accuracy :  0.8504703211157963
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.89      0.91      2608
           1       0.51      0.65      0.57       475

    accuracy                           0.85      3083
   macro avg       0.72      0.77      0.74      3083
weighted avg       0.87      0.85      0.86      3083

[[2312  296]
 [ 165  310]]
Sensitivity/tp_rate =  0.6526315789473685
Specificity/tn_rate =  0.8865030674846626
fp rate =  0.11349693251533742
fn rate =  0.3473684210526316
```

Confusion Matrix :

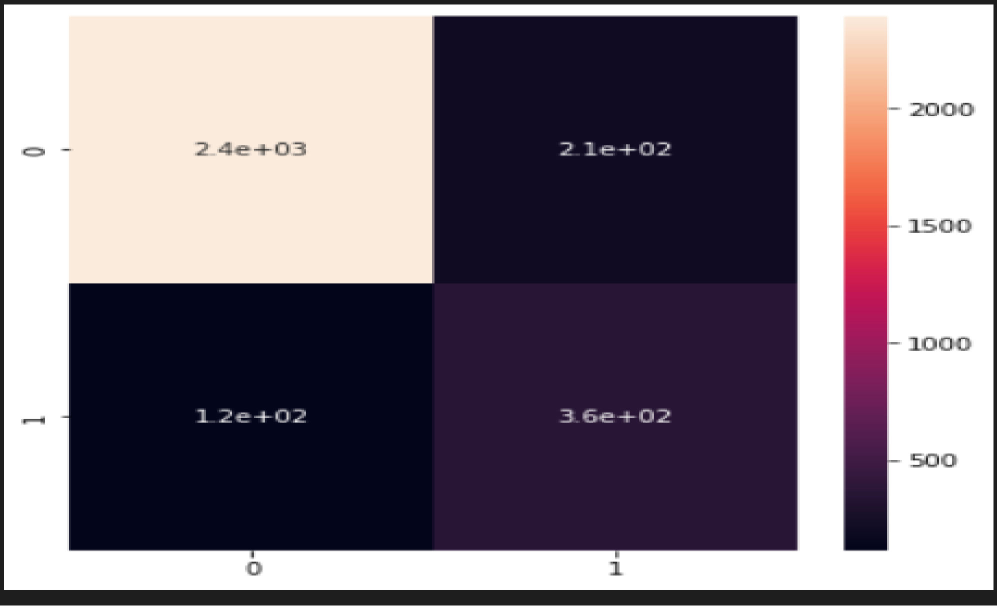


(d)Random Forest

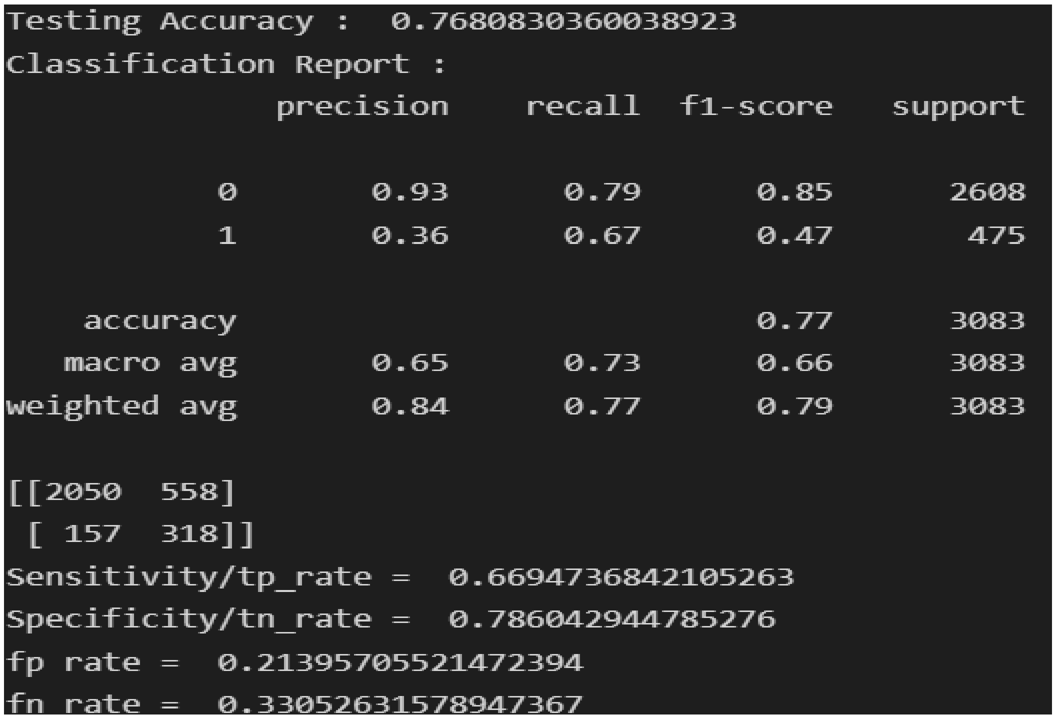


```
Testing Accuracy :  0.8926370418423614
Classification Report :
              precision    recall  f1-score   support

           0       0.95      0.92      0.94      2608
           1       0.63      0.75      0.68       475

    accuracy                           0.89      3083
   macro avg       0.79      0.84      0.81      3083
weighted avg       0.90      0.89      0.90      3083

[[2394  214]
 [ 117  358]]
Sensitivity/tp_rate =  0.7536842105263157
Specificity/tn_rate =  0.9179447852760736
fp rate =  0.08205521472392638
fn rate =  0.2463157894736842
```

```
Confusion Matrix :
```

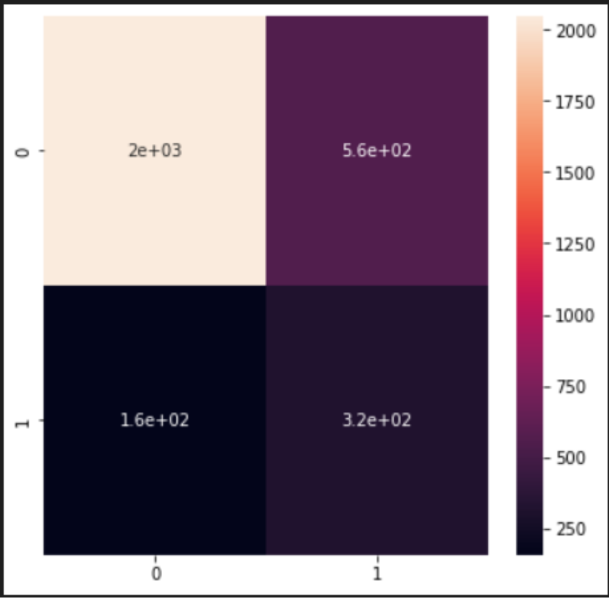


(e)K-Nearest Neighbor

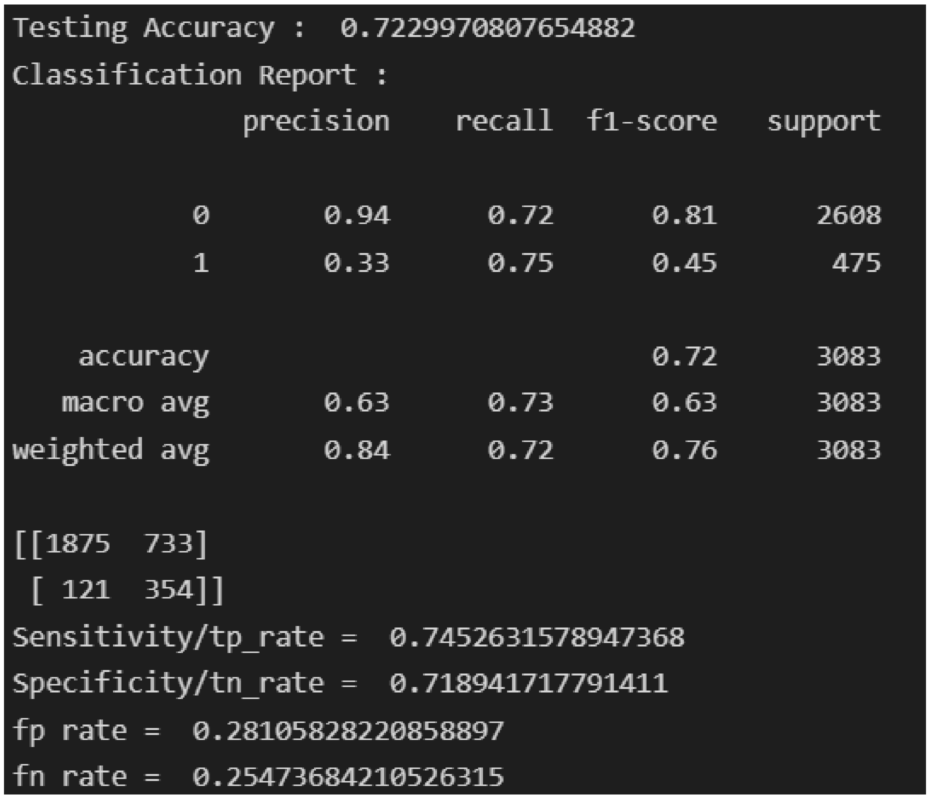


```
Testing Accuracy :  0.7680830360038923
Classification Report :
              precision    recall  f1-score   support

           0       0.93      0.79      0.85      2608
           1       0.36      0.67      0.47       475

    accuracy                           0.77      3083
   macro avg       0.65      0.73      0.66      3083
weighted avg       0.84      0.77      0.79      3083

[[2050  558]
 [ 157  318]]
Sensitivity/tp_rate =  0.6694736842105263
Specificity/tn_rate =  0.786042944785276
fp rate =  0.21395705521472394
fn rate =  0.33052631578947367
```

Confusion Matrix :

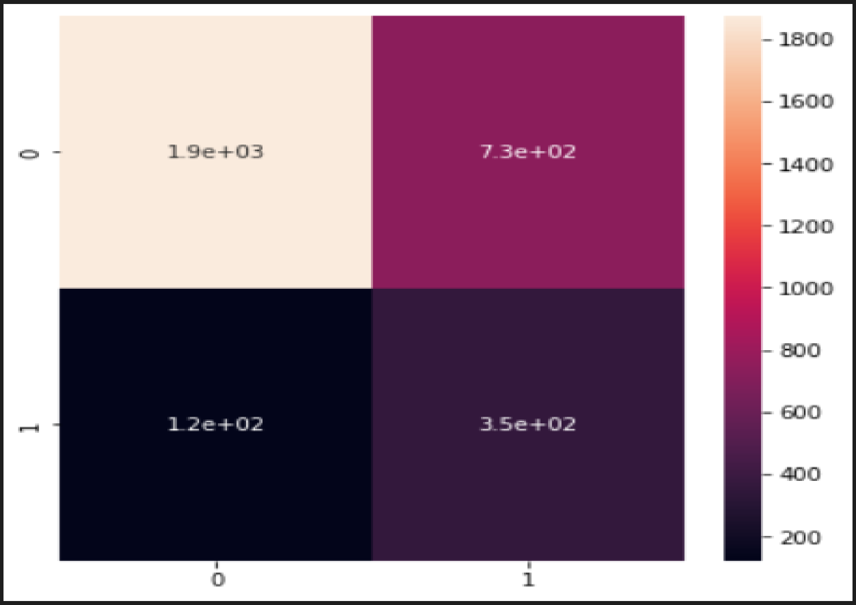


(f)Support Vector Machine



```
Testing Accuracy :  0.7229970807654882
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.72      0.81      2608
           1       0.33      0.75      0.45       475

    accuracy                           0.72      3083
   macro avg       0.63      0.73      0.63      3083
weighted avg       0.84      0.72      0.76      3083

[[1875  733]
 [ 121  354]]
Sensitivity/tp_rate =  0.7452631578947368
Specificity/tn_rate =  0.718941717791411
fp rate =  0.28105828220858897
fn rate =  0.25473684210526315
```

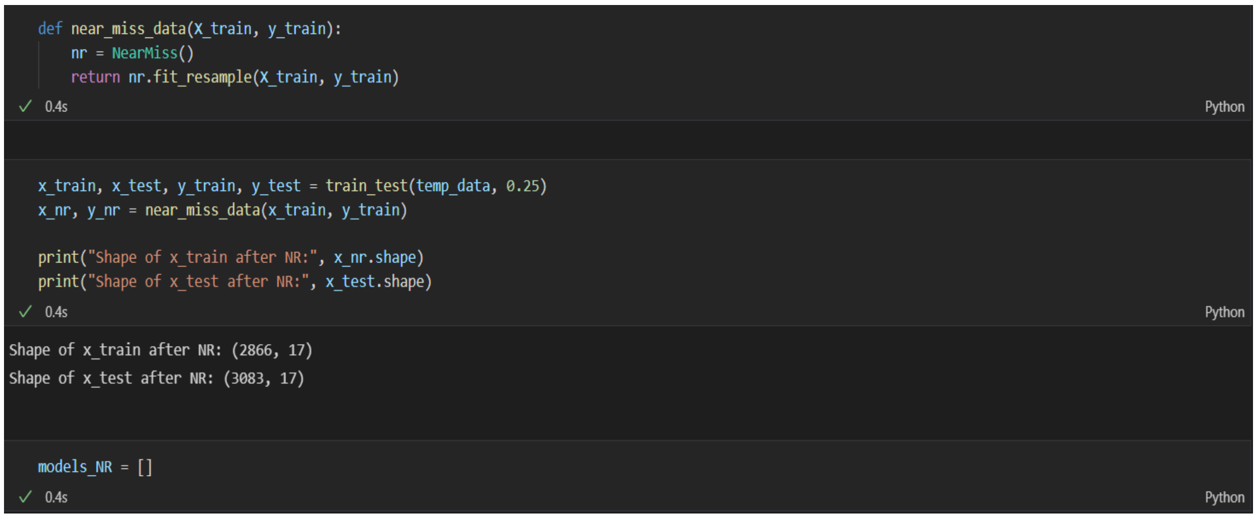
Confusion Matrix :



Undersampled dataset:

For Undersampled dataset, we use Near-Miss algorithm.

It is an algorithm that can help in balancing an imbalanced dataset. It can be grouped under undersampling algorithms and is an efficient way to balance the data. The algorithm does this by looking at the class distribution and randomly eliminating samples from the larger class. When two points belonging to different classes are very close to each other in the distribution, this algorithm eliminates the datapoint of the larger class thereby trying to balance the distribution.



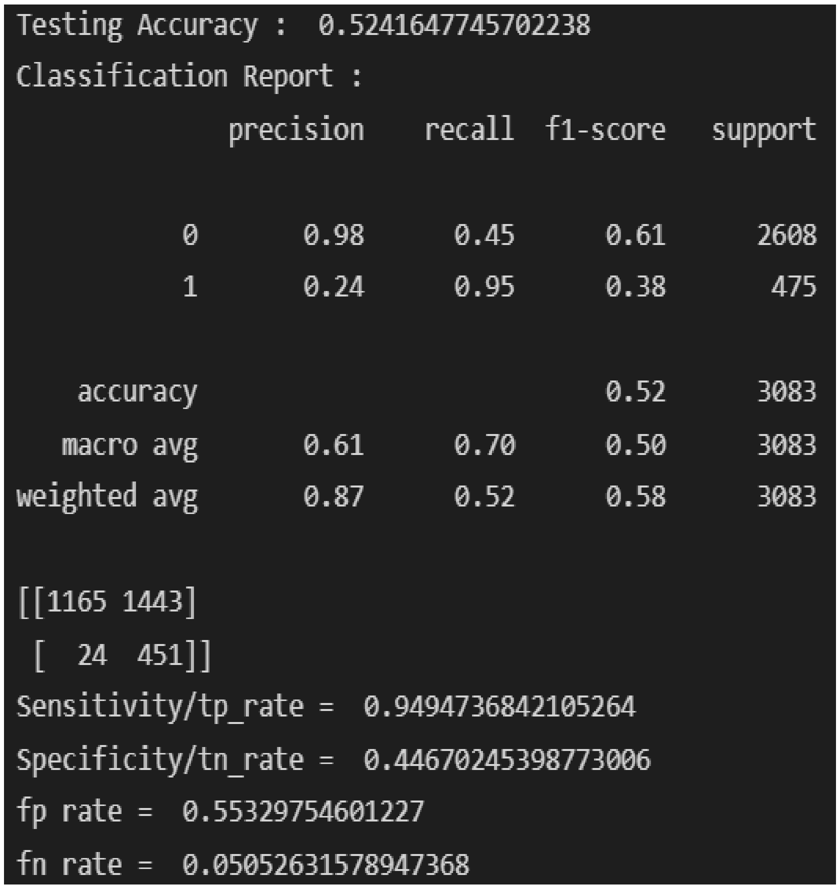
```python
def near_miss_data(X_train, y_train):
    nr = NearMiss()
    return nr.fit_resample(X_train, y_train)
```
✓ 0.4s Python

```python
x_train, x_test, y_train, y_test = train_test(temp_data, 0.25)
x_nr, y_nr = near_miss_data(x_train, y_train)

print("Shape of x_train after NR:", x_nr.shape)
print("Shape of x_test after NR:", x_test.shape)
```
✓ 0.4s Python

```
Shape of x_train after NR: (2866, 17)
Shape of x_test after NR: (3083, 17)
```

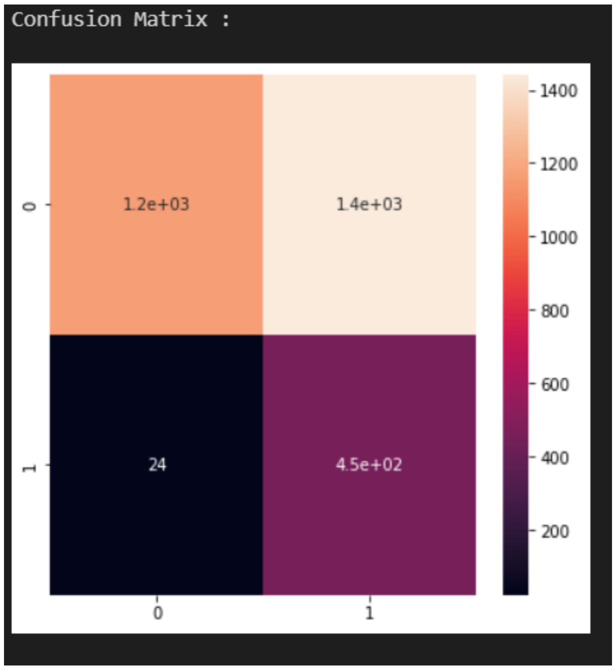
```python
models_NR = []
```
✓ 0.4s Python

(a)Naive Bayes

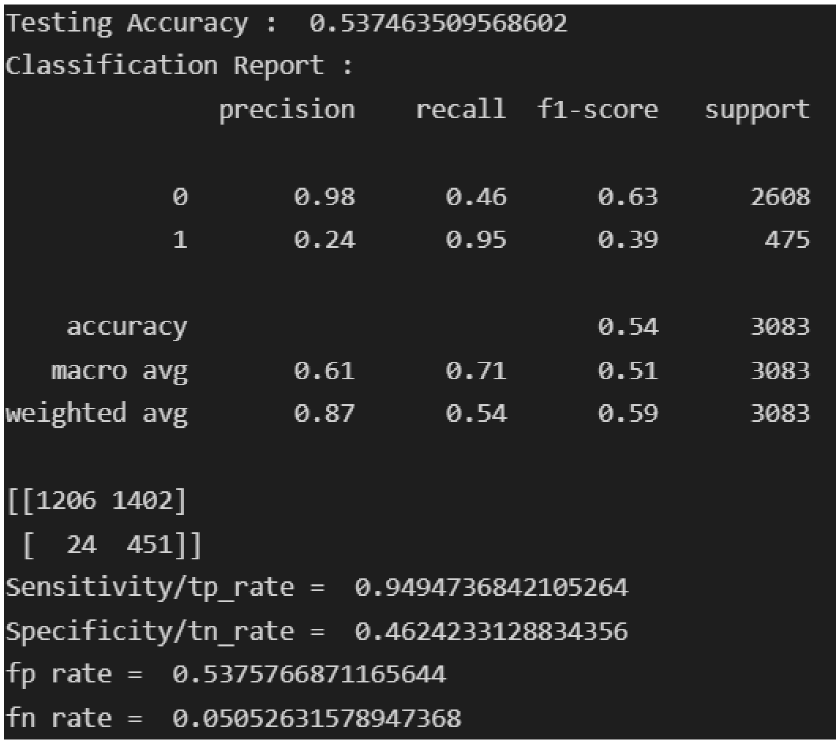


```
Testing Accuracy :  0.5241647745702238
Classification Report :
              precision    recall  f1-score   support

           0       0.98      0.45      0.61      2608
           1       0.24      0.95      0.38       475

    accuracy                           0.52      3083
   macro avg       0.61      0.70      0.50      3083
weighted avg       0.87      0.52      0.58      3083

[[1165 1443]
 [  24  451]]
Sensitivity/tp_rate =  0.9494736842105264
Specificity/tn_rate =  0.44670245398773006
fp rate =  0.55329754601227
fn rate =  0.050526315789473680
```

Confusion Matrix :

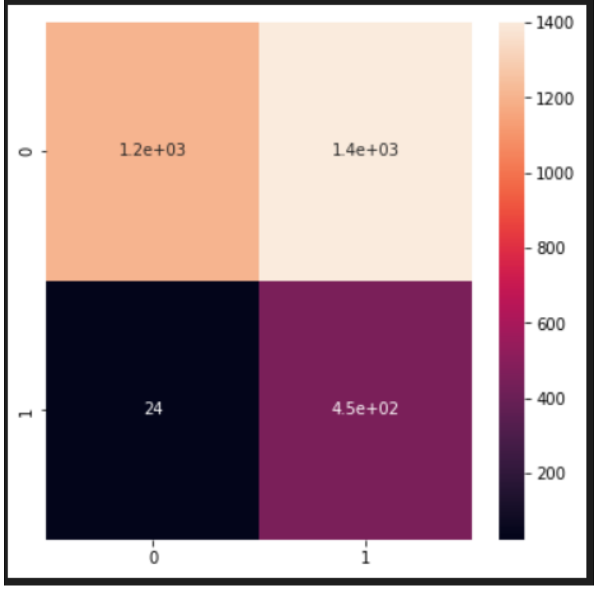


(b)Logistic Regression

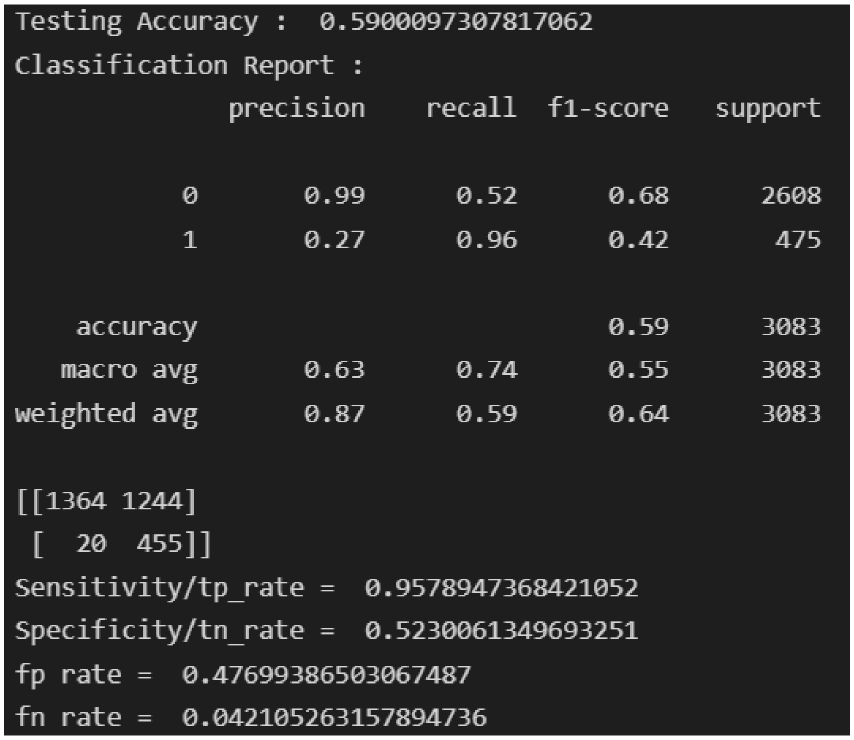


```
Testing Accuracy :  0.537463509568602
Classification Report :
              precision    recall  f1-score   support

           0       0.98      0.46      0.63      2608
           1       0.24      0.95      0.39       475

    accuracy                           0.54      3083
   macro avg       0.61      0.71      0.51      3083
weighted avg       0.87      0.54      0.59      3083

[[1206 1402]
 [  24  451]]
Sensitivity/tp_rate =  0.9494736842105264
Specificity/tn_rate =  0.4624233128834356
fp rate =  0.5375766871165644
fn rate =  0.05052631578947368
```

Confusion Matrix :

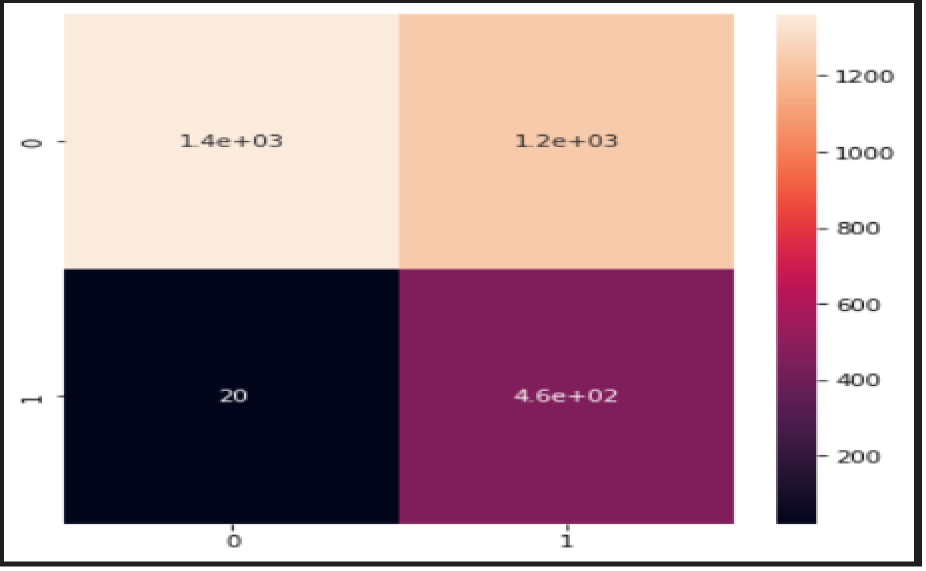


(c)Decision Tree

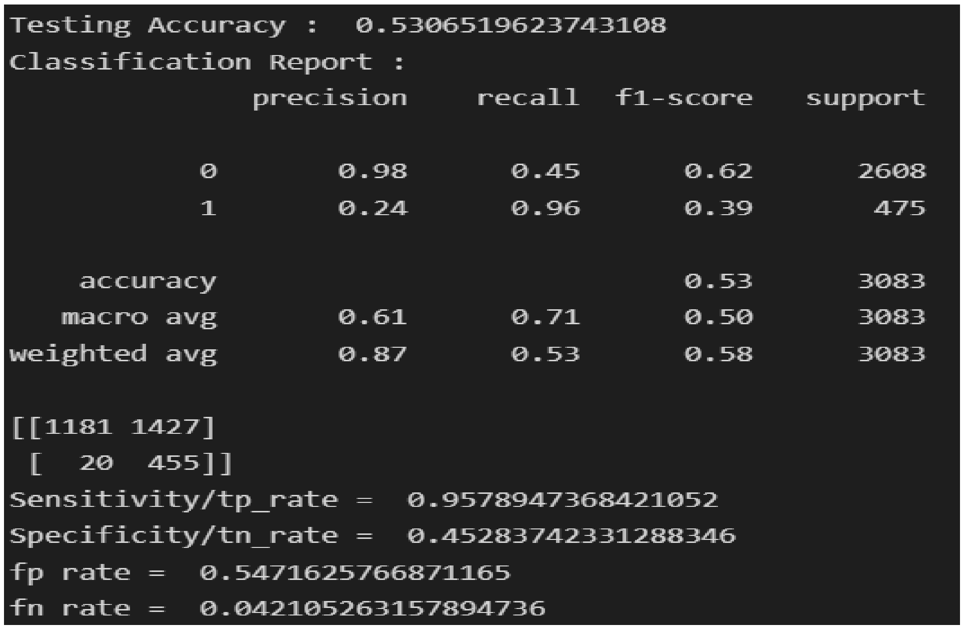


```
Testing Accuracy :  0.5900097307817062
Classification Report :
              precision    recall  f1-score   support

           0       0.99      0.52      0.68      2608
           1       0.27      0.96      0.42       475

    accuracy                           0.59      3083
   macro avg       0.63      0.74      0.55      3083
weighted avg       0.87      0.59      0.64      3083

[[1364 1244]
 [  20  455]]
Sensitivity/tp_rate =  0.9578947368421052
Specificity/tn_rate =  0.5230061349693251
fp rate =  0.47699386503067487
fn rate =  0.042105263157894736
```

Confusion Matrix :

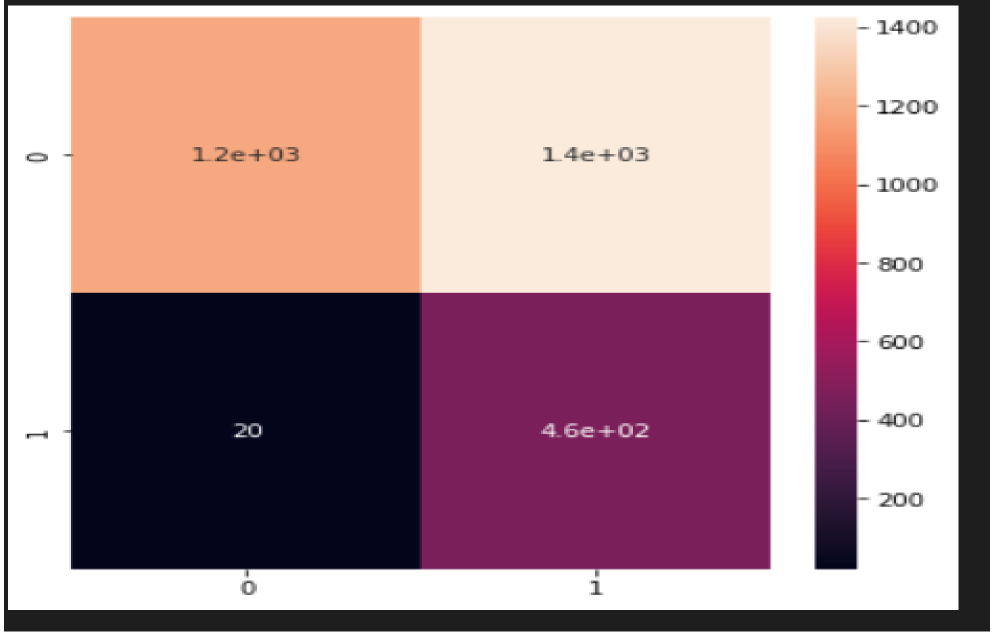


(d)Random Forest

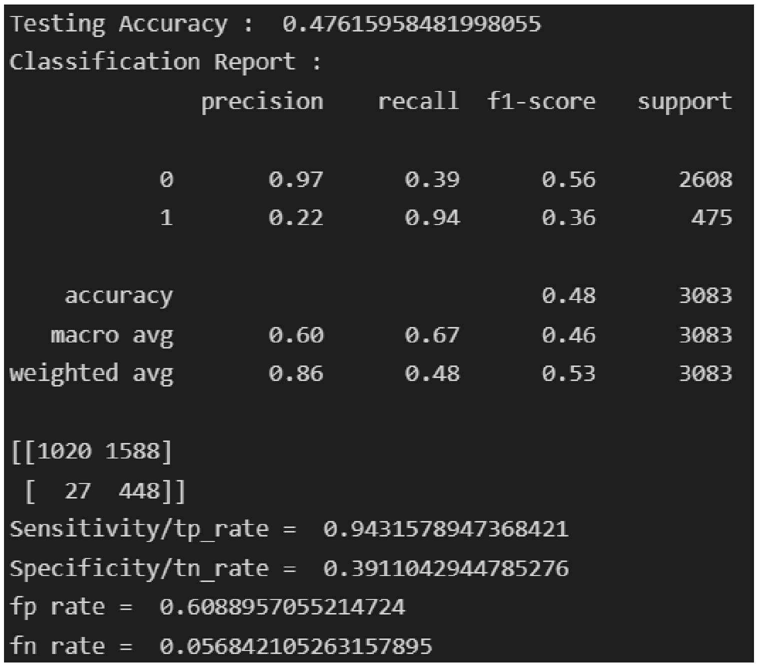


```
Testing Accuracy :  0.5306519623743108
Classification Report :
              precision    recall  f1-score   support

           0       0.98      0.45      0.62      2608
           1       0.24      0.96      0.39       475

    accuracy                           0.53      3083
   macro avg       0.61      0.71      0.50      3083
weighted avg       0.87      0.53      0.58      3083

[[1181 1427]
 [  20  455]]
Sensitivity/tp_rate =  0.9578947368421052
Specificity/tn_rate =  0.45283742331288346
fp rate =  0.5471625766871165
fn rate =  0.042105263157894736
```

Confusion Matrix :

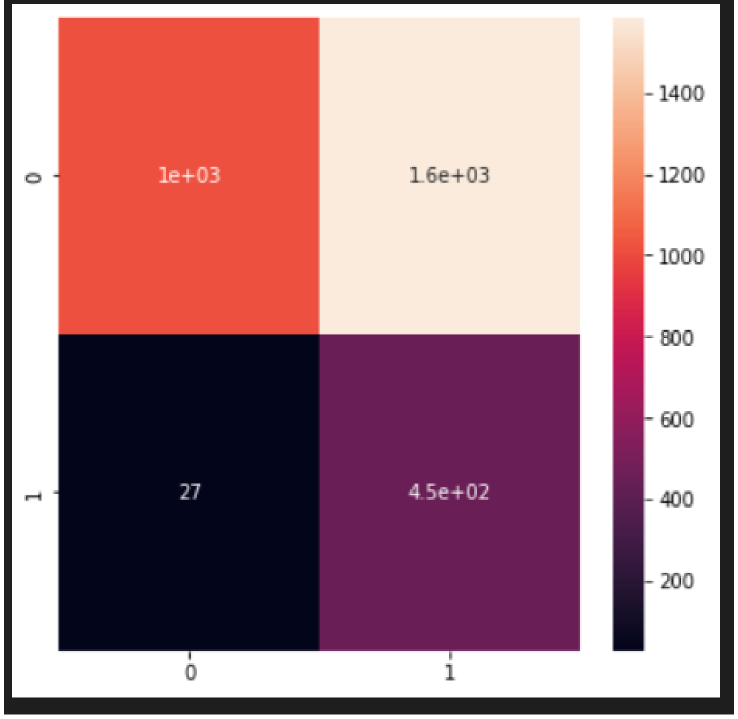


(e)K-Nearest Neighbor

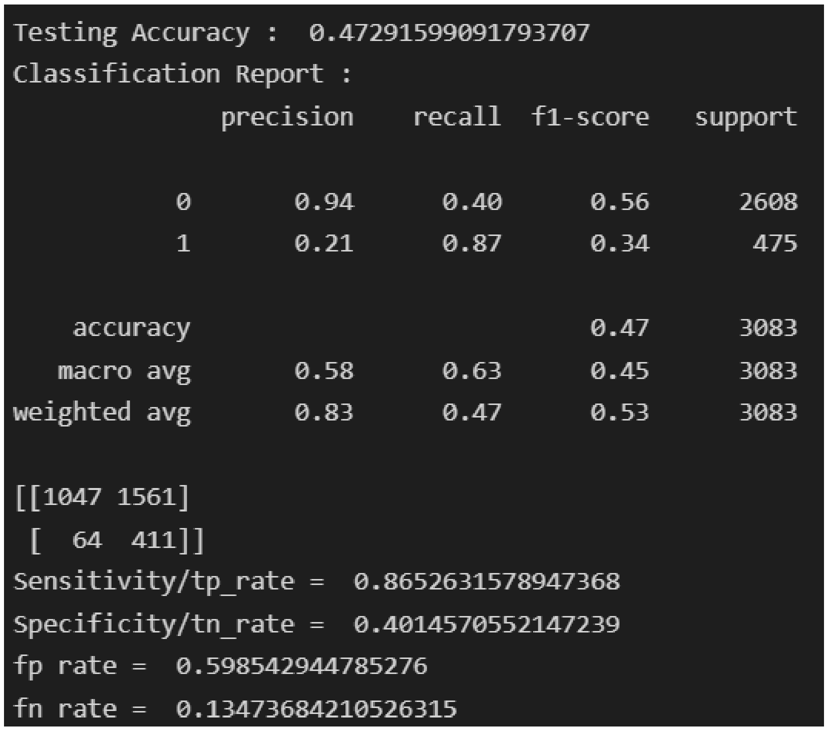


```
Testing Accuracy :  0.47615958481998055
Classification Report :
              precision    recall  f1-score   support

           0       0.97      0.39      0.56      2608
           1       0.22      0.94      0.36       475

    accuracy                           0.48      3083
   macro avg       0.60      0.67      0.46      3083
weighted avg       0.86      0.48      0.53      3083

[[1020 1588]
 [  27  448]]
Sensitivity/tp_rate =  0.9431578947368421
Specificity/tn_rate =  0.3911042944785276
fp rate =  0.6088957055214724
fn rate =  0.056842105263157895
```

```
Confusion Matrix :
```

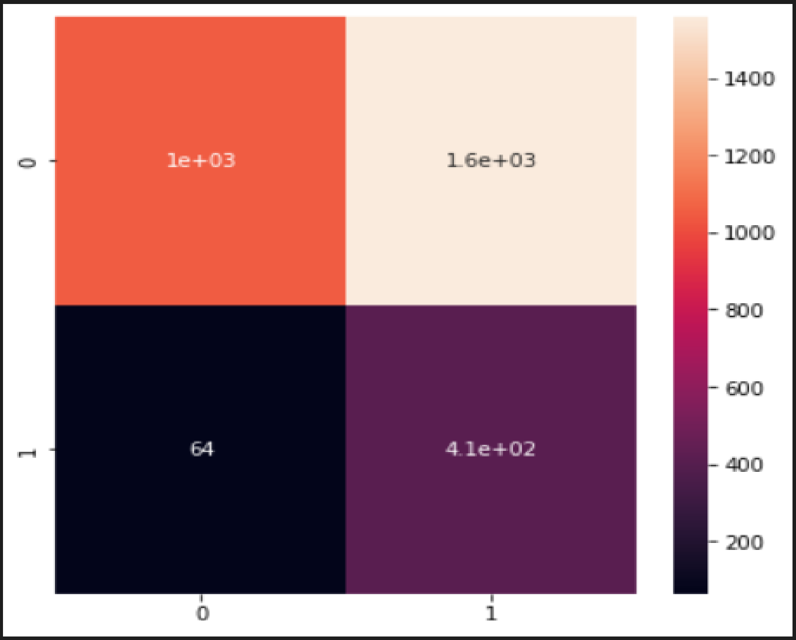


(f)Support Vector Machine



```
Testing Accuracy :  0.472915990917937O7
Classification Report :
              precision    recall  f1-score   support

           0       0.94      0.40      0.56      2608
           1       0.21      0.87      0.34       475

    accuracy                           0.47      3083
   macro avg       0.58      0.63      0.45      3083
weighted avg       0.83      0.47      0.53      3083

[[1047 1561]
 [  64  411]]
Sensitivity/tp_rate =  0.8652631578947368
Specificity/tn_rate =  0.4014570552147239
fp rate =  0.598542944785276
fn rate =  0.13473684210526315
```

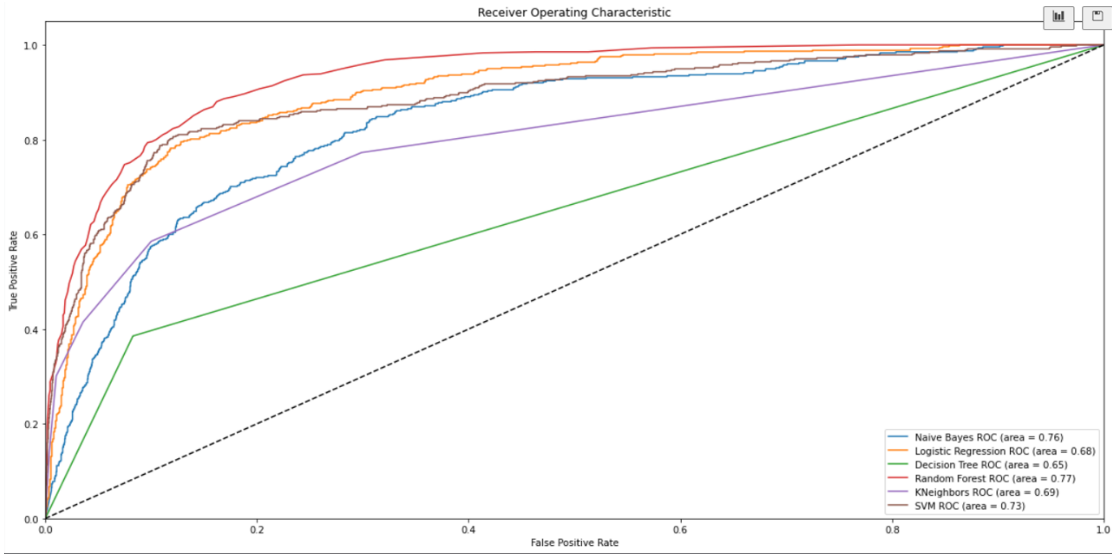
Confusion Matrix :



COMPARISON :

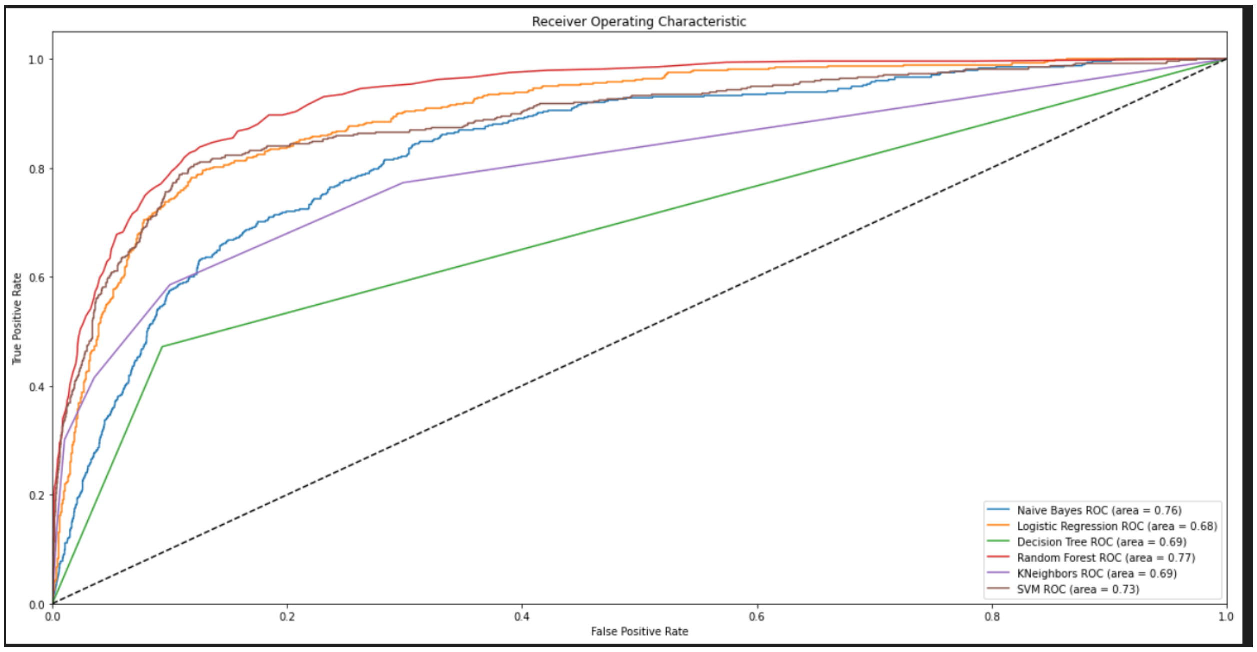
A. Normalized data

Model used	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.799	0.705	0.816	0.183	0.294
Logistic Regression	0.88	0.39	0.970	0.026	0.608
Decision Trees	0.83	0.38	0.906	0.093	0.528
Random forest	0.907	0.6	0.963	0.036	0.4
KNN	0.879	0.414	0.964	0.035	0.58
SVM	0.89	0.48	0.966	0.033	0.51



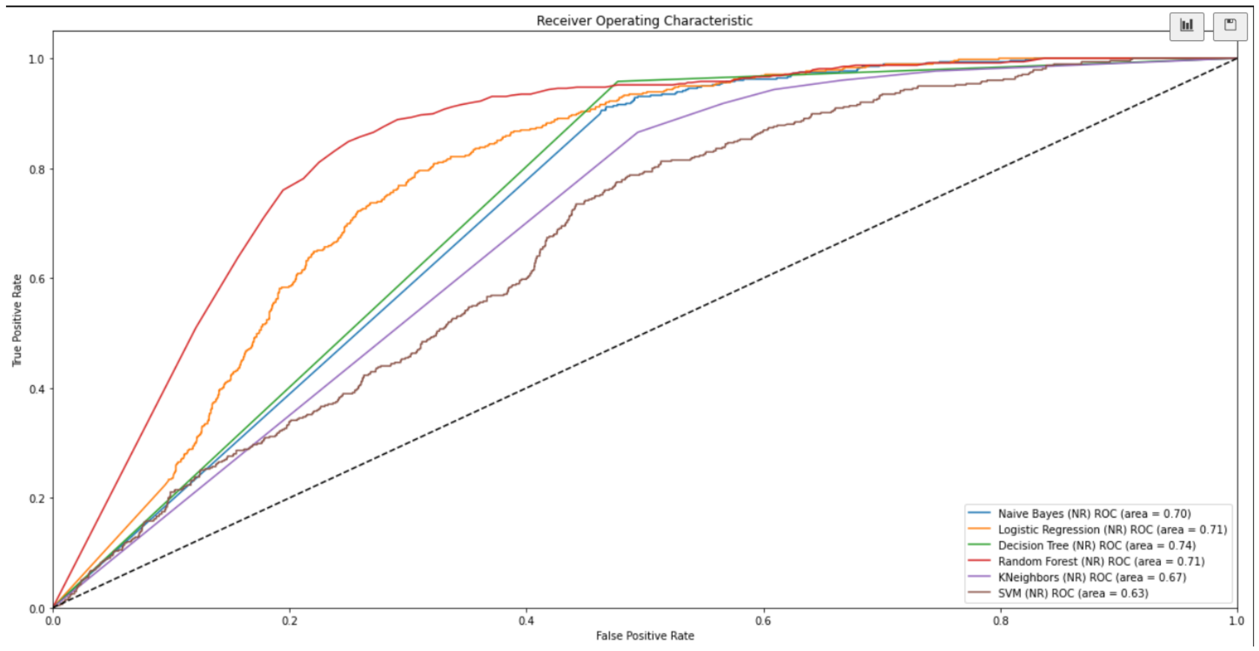
B. Oversampled dataset:

Model used	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.779	0.734	0.787	0.212	0.51
Logistic Regression	0.858	0.724	0.883	0.116	0.61
Decision Trees	0.850	0.652	0.886	0.113	0.57
Random forest	0.892	0.735	0.917	0.082	0.68
KNN	0.768	0.669	0.786	0.213	0.47
SVM	0.722	0.745	0.718	0.281	0.45



C. Undersampled dataset:

Model used	Success Rate	Tp rate (Sensitivity)	Tn rate (Specificity)	Fp rate	F1 score
Naïve Bayes	0.524	0.949	0.446	0.553	0.38
Logistic Regression	0.537	0.949	0.462	0.537	0.39
Decision Trees	0.590	0.957	0.52	0.47	0.42
Random forest	0.530	0.957	0.452	0.547	0.39
KNN	0.476	0.943	0.391	0.608	0.36
SVM	0.472	0.865	0.401	0.598	0.34



Conclusion:

For normalized dataset, it is observed that ROC of Random Forest model covers most area even though it has low F1 score.

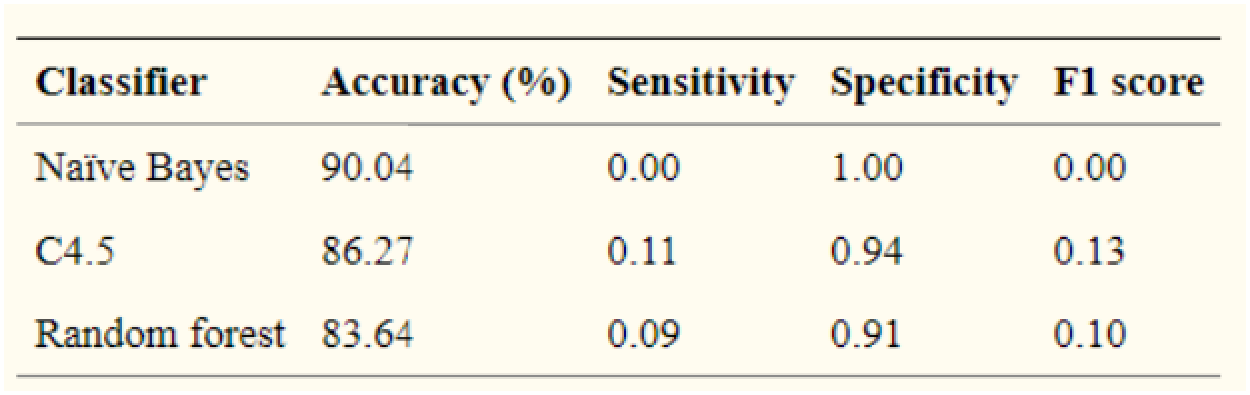
For undersampled dataset, it is observed that ROC of Decision Tree model covers most area while having low F1 score.

For oversampled dataset, it is observed that ROC of Random Forest model covers most area while having medium F1 score.

Comparative Experiment

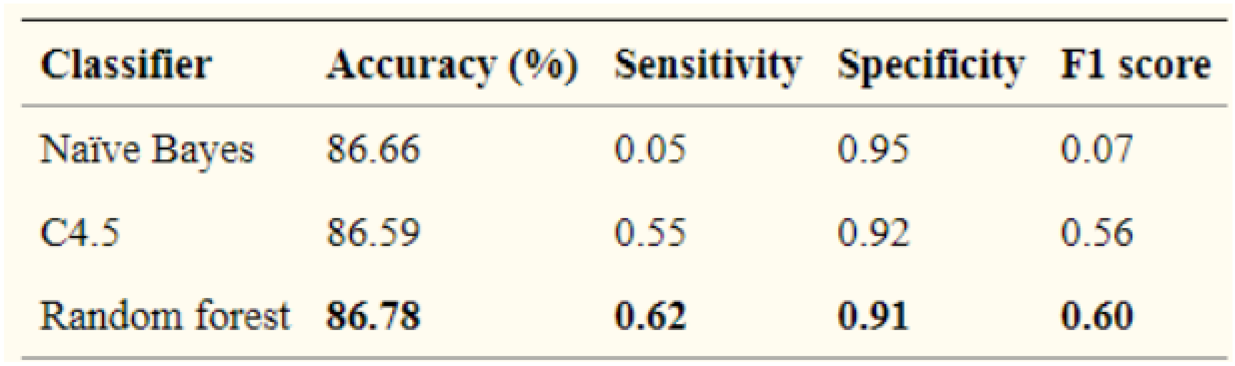
This is the official published paper where they have taken Naïve Bayes, Decision tree and random forest models and applied it on normal data and data after Synthetic Minority Oversampling Technique (smot). Train test size taken by them is 70-30 as opposed to our train test size of75-25.

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**

Bibliography

1. https://www.mbaknol.com/marketing-management/the-engel-kollat-blackwell-model-of-consumer-behavior/
2. https://www.emerald.com/insight/content/doi/10.1108/03090560910976410/full/pdf?title=shopping-orientation-and-online-clothing-purchases-the-role-of-gender-and-purchase-situation
3. https://www.ukessays.com/essays/marketing/literature-review-of-online-purchase-intention-marketing-essay.php
4. Naive Bayes Classifier : https://jakevdp.github.io/PythonDataScienceHandbook/05.05-naive-bayes.html
5. Logisitc Regression Classifier: https://www.datacamp.com/community/tutorials/understanding-logistic-regression-python
6. Decision Tree Classifier: https://www.datacamp.com/community/tutorials/decision-tree-classification-python
7. Random Forest Classifier: https://www.datacamp.com/community/tutorials/random-forests-classifier-python
8. KNeighbors Classifier: https://www.analyticsvidhya.com/blog/2021/01/a-quick-introduction-to-k-nearest-neighbor-knn-classification-using-python/
9. SVM : https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python
10. SMOTE (Oversampling): https://towardsdatascience.com/applying-smote-for-class-imbalance-with-just-a-few-lines-of-code-python-cdf603e58688
11. Near-Miss (Undersampling): https://analyticsindiamag.com/using-near-miss-algorithm-for-imbalanced-datasets/