BANK CUSTOMER CHURN PREDICTION

Objective of the study:

- The aim of this project is to develop a predictive model for customer churn in a subscription-based service or business.
- Customer churn, or customer attrition, refers to the rate at which customers stop using a service.
- By analyzing historical customer data and utilizing machine learning algorithms such as Logistic Regression, Random Forests, or Gradient Boosting,aim to predict which customers are likely to churn in the future.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

Importing Libraries

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

Read the dataset

```
df=pd.read_csv('/content/drive/MyDrive/datascience/PROJECT/Machine Learning project/bank customer churn prediction dataset (ML).csv')
```

```
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	
2	3	15619304	Onio	502	France	Female	42	8	1
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	1
...	...	...	...	...	...	...	...	...	
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	
9999	10000	15628319	Walker	792	France	Female	28	4	1

10000 rows × 14 columns

```
#To print the first 5 rows using head function
df.head()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Bal
0	1	15634602	Hargrave	619	France	Female	42	2	
1	2	15647311	Hill	608	Spain	Female	41	1	8380
2	3	15619304	Onio	502	France	Female	42	8	15966
3	4	15701354	Boni	699	France	Female	39	1	
4	5	15737888	Mitchell	850	Spain	Female	43	2	12551

```
#To print the last 5 rows using tail function
df.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

```
#view the shape of dataset
df.shape
```

```
(10000, 14)
```

Observation:

The Df has 1000 rows with 14 attributes. We review this further to identify what attributes will be necessary and what data manipulation needs to be carried out before Exploratory analysis and prediction modelling

```
#view the dataset dimension
df.ndim
```

```
2
```

```
#view the all columns in the dataset
df.columns
```

```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
#checking for missing values
df.isna().any()
```

```
RowNumber          False
CustomerId         False
Surname            False
CreditScore        False
Geography          False
Gender             False
Age                False
Tenure             False
Balance            False
NumOfProducts      False
HasCrCard          False
IsActiveMember     False
EstimatedSalary    False
Exited             False
dtype: bool
```

```
#To check for the duplicate values
df.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

```
#Removing duplicate rows
df=df.drop_duplicates()
df
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMem
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

10000 rows × 14 columns

```
#After removing duplicate values checking the number of rows and columns
df.shape
```

```
(10000, 14)
```

```
#check the descriptive statistics of numeric variables
df.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000

Observation

- The average age of the customers is 39.
- Tenure is maximum 10 years which assume age of the bank.
- Minimum value and first quantiles of balance are equal 0 which means the distribution may not be normal.

```
#view the dataset information
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 0 to 9999
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   RowNumber        10000 non-null  int64  
 1   CustomerId       10000 non-null  int64  
 2   Surname          10000 non-null  object 
 3   CreditScore      10000 non-null  int64  
 4   Geography        10000 non-null  object 
 5   Gender           10000 non-null  object 
 6   Age              10000 non-null  int64  
 7   Tenure           10000 non-null  int64  
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64  
 10  HasCrCard        10000 non-null  int64  
 11  IsActiveMember   10000 non-null  int64  
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64  
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

Observation:

Here,

- surname,geography and gender are in object type.
- balance,estimated salary are in float data type.
- and the remainders are in int data type.

```
#checking unique value to object type columns
features=['Surname','Geography','Gender']
for i in features:
  print(df[i].unique(),i)
  print('----------------')
```

```
['Hargrave' 'Hill' 'Onio' ... 'Kashiwagi' 'Aldridge' 'Burbidge'] Surname
----------------
['France' 'Spain' 'Germany'] Geography
----------------
['Female' 'Male'] Gender
----------------
```

```
# checking value counts of each column
features=['RowNumber','CustomerId','Surname','CreditScore','Geography','Gender','Age','Tenure','Balance','NumOfProducts','HasCrCard','Is

for i in features:

  print(df[i].value_counts(),i)
  print('--------------------')
```

```
1        1
6671     1
6664     1
6665     1
6666     1
        ..
3334     1
3335     1
3336     1
3337     1
10000    1
Name: RowNumber, Length: 10000, dtype: int64 RowNumber
--------------------
15634602    1
15667932    1
15766185    1
15667632    1
15599024    1
           ..
15599078    1
15702300    1
15660735    1
15671390    1
15628319    1
Name: CustomerId, Length: 10000, dtype: int64 CustomerId
--------------------
Smith       32
Scott       29
Martin      29
Walker      28
Brown       26
            ..
Izmailov     1
Bold         1
Bonham       1
Poninski     1
Burbidge     1
Name: Surname, Length: 2932, dtype: int64 Surname
--------------------
850    233
678     63
655     54
705     53
667     53
      ...
404      1
351      1
365      1
417      1
419      1
Name: CreditScore, Length: 460, dtype: int64 CreditScore
--------------------
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64 Geography
--------------------
```

```
Male      5457
```

Observation:

From the above, we will not require the first 2 attributes as the are specific to a customer. It is borderline with the surname as this would result to profiling so we exclude this as well.

```
#Drop the columns which are not necessary
df=df.drop(['RowNumber','CustomerId','Surname'],axis=1)
```

```
df.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63	0
4	850	Spain	Female	43	2	125510.82	1	1	1	79084.10	0

```
df.dtypes
```

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

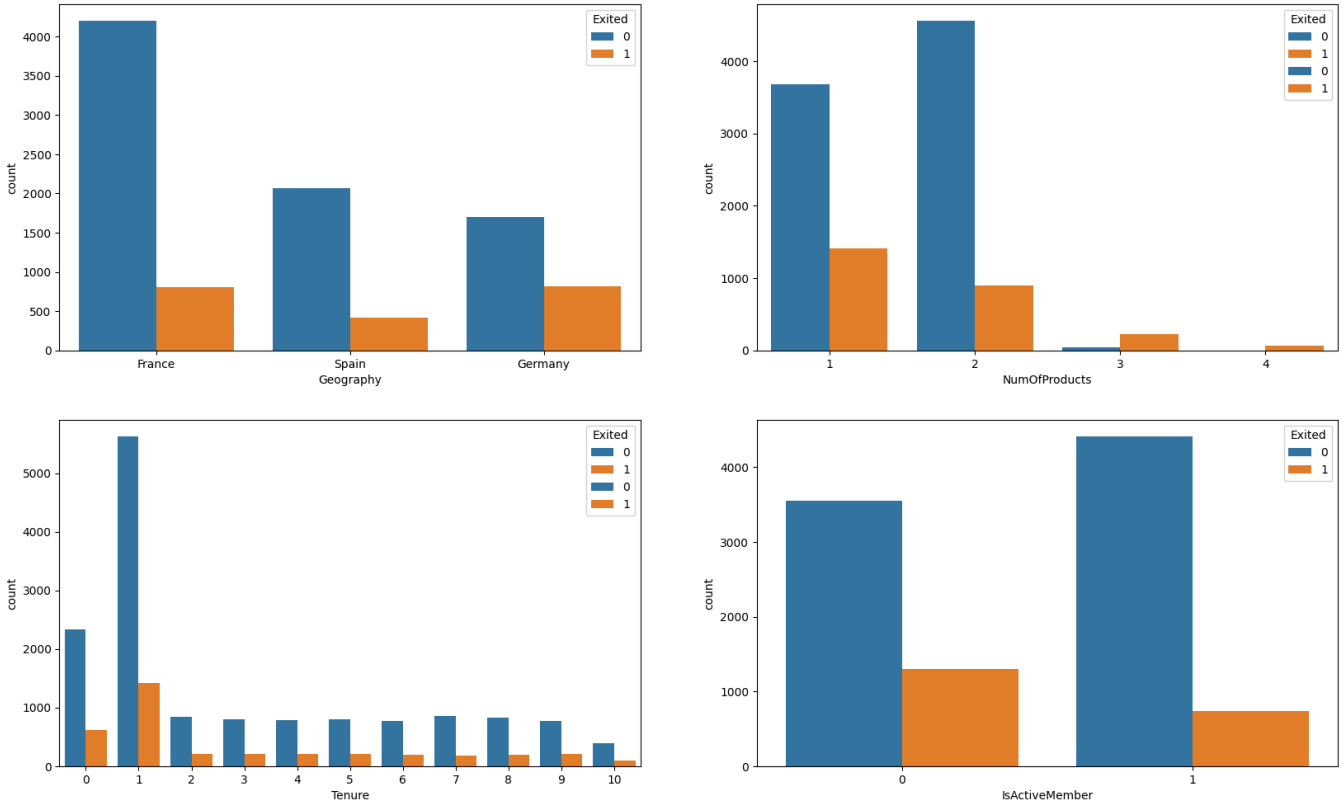
Observation:

So we moslty have categorical variables and 5 continuous variables.

Visualizing categorical variables

```
import matplotlib.pyplot as plt
import seaborn as sns
fig,axarr = plt.subplots(2, 2, figsize=(20, 12))
sns.countplot(x='Geography', hue = 'Exited',data = df, ax=axarr[0][0])
sns.countplot(x='Gender', hue = 'Exited',data = df, ax=axarr[0][1])
sns.countplot(x='HasCrCard', hue = 'Exited',data = df, ax=axarr[1][0])
sns.countplot(x='IsActiveMember', hue = 'Exited',data = df, ax=axarr[1][1])
sns.countplot(x='Tenure', hue = 'Exited',data = df, ax=axarr[1][0])
sns.countplot(x='NumOfProducts', hue = 'Exited',data = df, ax=axarr[0][1])
```

```
<Axes: xlabel='NumOfProducts', ylabel='count'>
```

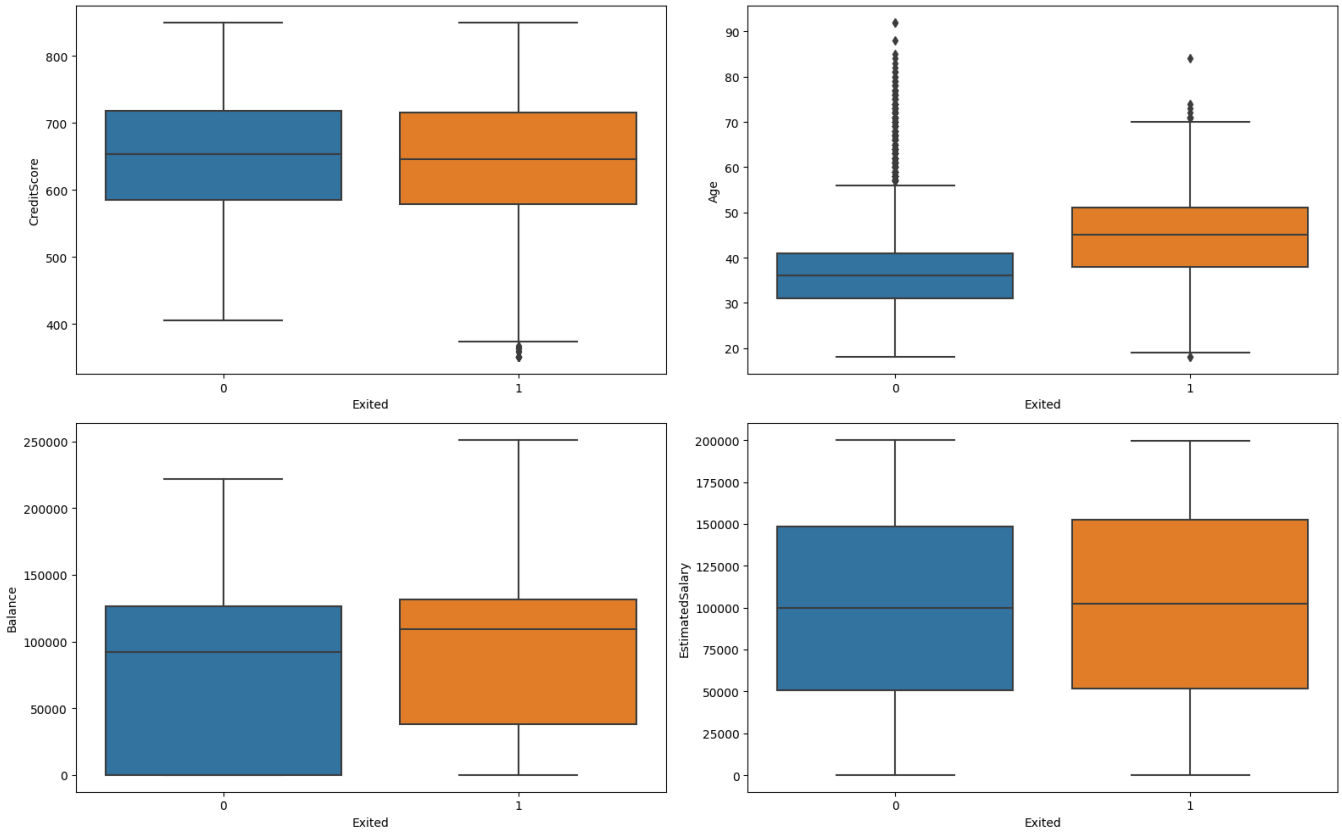


Observation:

- Majority of the customers are from france. However, the proportion of churned customers is with inversely related to the population of customers alluding to the bank possibly having a problem (maybe not enough customer service resources allocated) in the areas where it has fewer clients.
- The proportion of female customers churning is also greater than that of male customers.
- Interestingly, majority of the customers that churned are those with credit cards. Given that majority of the customers have credit cards could prove this to be just a coincidence.
- Most of the customers have 1 or 2 products and most customers which churned are having 1 products maybe they are not satisfied so they are churning.
- .Majority of customers have tenure between 1 to 9 and churing rate is also high between these tenures.
- Unsurprisingly the inactive members have a greater churn. Worryingly is that the overall proportion of inactive mebers is quite high suggesting that the bank may need a program implemented to turn this group to active customers as this will definately have a positive impact on the customer churn.

Visualizing continuous variables

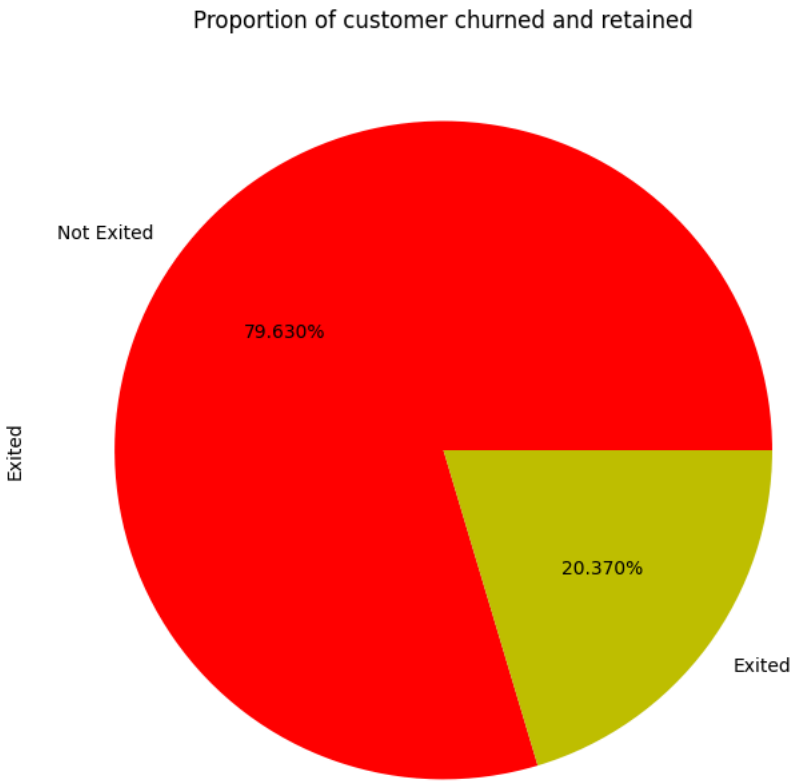
```
fig, ax = plt.subplots(2, 2, figsize = (16, 10))
sns.boxplot(x = 'Exited', y = 'CreditScore', data = df, ax = ax[0][0])
sns.boxplot(x = 'Exited', y = 'Age', data = df, ax = ax[0][1])
sns.boxplot(x = 'Exited', y = 'Balance', data = df, ax = ax[1][0])
sns.boxplot(x = 'Exited', y = 'EstimatedSalary', data = df, ax = ax[1][1])
plt.tight_layout()
plt.show()
```



Observation:

- There is no significant difference in credit score distribution etween custers which are churned or not.
- The older customers are churning at more than the younger ones alluding to a difference in service preference in the age categories.
- The bank may need to review their target market or review the strategy for retention between the different age groups With regard to the tenure, the clients on either extreme end (spent little time with the bank or a lot of time with the bank) are more likely to churn compared to those that are of average tenure.
- Worryingly, the bank is losing customers with significant bank balances which is likely to hit their available capital for lending. Neither the product nor the salary has a significant effect on the likelihood to churn.

```
plt.figure(figsize=(8,8))
labels = ['Not Exited', 'Exited']
df["Exited"].value_counts().plot(kind='pie',labels = labels,autopct='%1.3f%%',colors=['r','y'])
labels = ['Not Exited', 'Exited']
plt.title("Proportion of customer churned and retained")
plt.show()
```



Observation:

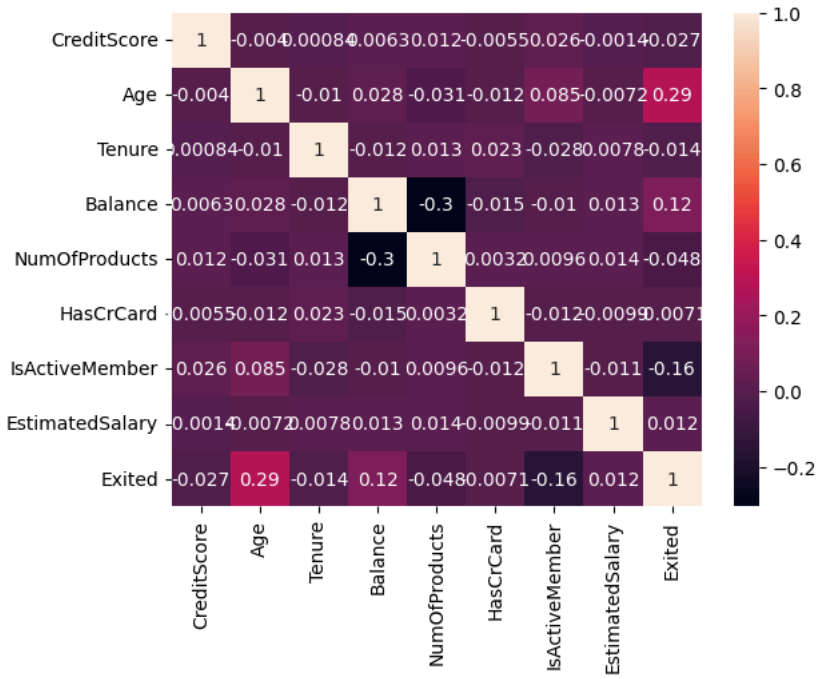
So about 20% of the customers have churned. So the baseline model could be to predict that 20% of the customers will churn. Given 20% is a small number, we need to ensure that the chosen model does predict with great accuracy this 20% as it is of interest to the bank to identify and keep this bunch as opposed to accurately predicting the customers that are retained.

```
cor_mtx=df.corr()['Exited']
cor_mtx
```

```
<ipython-input-1020-eb56047a69c6>:1: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  cor_mtx=df.corr()['Exited']
CreditScore       -0.027094
Age                0.285323
Tenure            -0.014001
Balance            0.118533
NumOfProducts     -0.047820
HasCrCard         -0.007138
IsActiveMember    -0.156128
EstimatedSalary    0.012097
Exited             1.000000
Name: Exited, dtype: float64
```

```
import seaborn as sns
sns.heatmap(df.corr(),annot=True)
```

```
<ipython-input-1021-084798591dac>:2: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future v
  sns.heatmap(df.corr(),annot=True)
<Axes: >
```



Observation:

Overall,the correaltion is very weak

- weak positive correlation with age
- very weak positive correlation with balance
- very weak negative correlation with number of products and membership

```
#check varaible data types of all columns
df.dtypes
```

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

convert categorical variables into numerical format

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
lst=['Geography','Gender']
for i in lst:
  df[i]=le.fit_transform(df[i])
  df
```

```
df
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	0	0	42	2	0.00	1	1	1	101348.88	1
1	608	2	0	41	1	83807.86	1	0	1	112542.58	0
2	502	0	0	42	8	159660.80	3	1	0	113931.57	1
3	699	0	0	39	1	0.00	2	0	0	93826.63	0
4	850	2	0	43	2	125510.82	1	1	1	79084.10	0
...	...	...	...	...	...	...	...	...	...	...	...
9995	771	0	1	39	5	0.00	2	1	0	96270.64	0
9996	516	0	1	35	10	57369.61	1	1	1	101699.77	0
9997	709	0	0	36	7	0.00	1	0	1	42085.58	1
9998	772	1	1	42	3	75075.31	2	1	0	92888.52	1
9999	792	0	0	28	4	130142.79	1	1	0	38190.78	0

10000 rows × 11 columns

Here the value indicates:

female=0, male=1

france=2,spain=0,germany=1

MODEL CREATION

Classification Algorithms

1. **K-NEAREST NEIGHBOR (KNN)**
2. **GAUSSIAN NAIVE BAYES**
3. **BERNOULLI NAIVE BAYES**
4. **SUPPORT VECTOR MACHINE**
5. **DECISION TREE**
6. **RANDOM FOREST**
7. **ADA BOOST**
8. **LOGISTIC REGRESSION**

PERFORMANCE EVALUATION

Accuracy performance metrics can be decisive when dealing with imbalanced data. In this blog, we will learn about the Confusion matrix and its associated terms, which looks confusing but are trivial. The confusion matrix, precision, recall, and F1 score gives better intuition of prediction results as compared to accuracy.

Confusion Matrix: It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.

Accuracy Score: Accuracy is the measure of correct predictions made by our model. It is equal to the number of correct predictions made upon total number of predictions made by the model.

Precision Score: It is defined as the ratio of true positives to the sum of true and false positives. It is also known as Positive Predictive Value (PPV).

Recall Score: It is defined as the ratio of true positives to the sum of true positives and false negatives. It is also called True Positive Rate (TPR) or sensitivity.

F1 score: It is the weighted harmonic mean of precision and recall. The closer the value of the F1 score is to 1.0 , the better the expected performance of the model is.

split the data frame into x and y

```
x=df.drop('Exited',axis=1)
y=df['Exited']


#predictors
x.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary
0	619	0	0	42	2	0.00	1	1	1	101348.88
1	608	2	0	41	1	83807.86	1	0	1	112542.58
2	502	0	0	42	8	159660.80	3	1	0	113931.57
3	699	0	0	39	1	0.00	2	0	0	93826.63
4	850	2	0	43	2	125510.82	1	1	1	79084.10

```
#target
y.head()
```

```
0    1
1    0
2    1
3    0
4    0
Name: Exited, dtype: int64
```

```
x.shape
```

```
(10000, 10)
```

Train Test Split

- The train-test split is a technique for evaluating the performance of a machine Learning algoritham.
- Train Dataset Used to fit the machine learning model.
- choosing split percentage:Train:80%,Test 20%

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=42)


#train base size
x_train.shape,y_train.shape
```

```
((8000, 10), (8000,))
```

```
#test base size
x_test.shape,y_test.shape
```

```
((2000, 10), (2000,))
```

SCALING

- It helps to balance the impact of all variables on the euclidean distance calculation and can help to improve the performance of the algorithm.
- If not scaled,the feature with a higher value range starts dominating when calculating distances.

Standardization

.it helps to balance the impact of all variables on the euclidean distance calculation and can help to improve the performance of the algorithm.

.if not scaled,the feature with a higher value range starts dominating when calculating distances.

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()


x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

CLASSIFICATION ALGORITHMS

1) KNearest Neighbors

It is used for classification and regression.in both cases,the input consist of the k closest training examples in data set.The output depends on whether k-nn is used for classification or regression.

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(x_train,y_train)

y_pred1=knn_model.predict(x_test)
y_pred1
```

```
array([0, 0, 0, ..., 1, 0, 0])
```

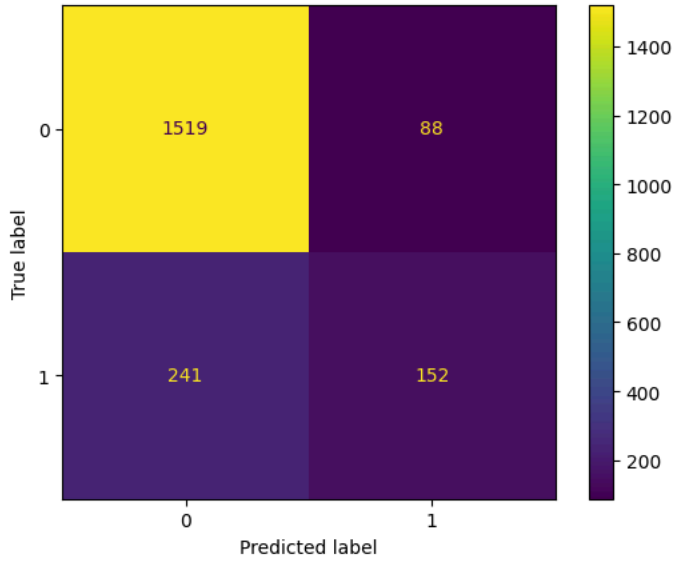
```
from sklearn.linear_model import LogisticRegression

model=LogisticRegression()
model.fit(xtest,ytest)
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred1)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1519   88]
 [ 241  152]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806974d7b0>
```



From KNearest Neighbors ConfusionMatrix we observe that:

- 1519 customers are correctly classified under not “churned” category & 88 are misclassified
- 152 customers are correctly classified under “churned” category & 241 are misclassified

```
# Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

a=accuracy_score(y_test,y_pred1)
print('Accuracy: ',a*100,'\n')
print(classification_report(y_test,y_pred1))
```

```
Accuracy:  83.55

              precision    recall  f1-score   support
```

```
              0       0.86      0.95      0.90      1607
              1       0.63      0.39      0.48       393

       accuracy                           0.84      2000
      macro avg       0.75      0.67      0.69      2000
   weighted avg       0.82      0.84      0.82      2000
```

```
training_score = knn_model.score(x_train,y_train)
training_score
```

```
0.877875
```

```
testing_score =knn_model.score(x_train,y_train)
testing_score
```

```
0.877875
```

2) NAIVE BAYES : GaussianNB

Naive Bayes is a probabilistic classification algorithm based on Bayes theorem.it assume that the features are independent of each other,making the calculation computationally efficient. This type of Naive Bayes is used when variables are continuous in nature.

```
from sklearn.naive_bayes import GaussianNB
bayes_model = GaussianNB()
bayes_model.fit(x_train,y_train)
y_pred2 = bayes_model.predict(x_test)
y_pred2
```

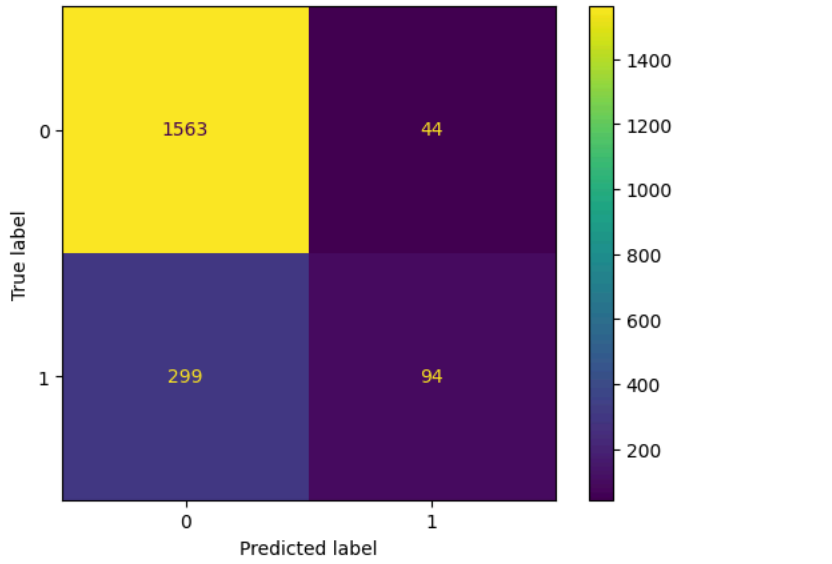
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred2)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1563   44]
 [ 299   94]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806c14e980>
```



From Naive Bayes ConfusionMatrix we observe that:

- 1563 customers are correctly classified under not “churned” category & 44 are misclassified
- 94 customers are correctly classified under “churned” category & 299 are misclassified.

```
#Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

b=accuracy_score(y_test,y_pred2)
print('Accuracy: ',b*100,'\n')
print(classification_report(y_test,y_pred2))
```

```
Accuracy:  82.85

              precision    recall  f1-score   support

           0       0.84      0.97      0.90      1607
           1       0.68      0.24      0.35       393

    accuracy                           0.83      2000
   macro avg       0.76      0.61      0.63      2000
weighted avg       0.81      0.83      0.79      2000
```

```
training_score = bayes_model.score(x_train,y_train)
training_score
```

```
0.829875
```

```
testing_score =bayes_model.score(x_train,y_train)
testing_score
```

```
0.829875
```

3) NAIVE BAYES : BernoulliNB

- Bernoulli Naive Bayes algorithm is a supervised learning algorithm, which is based on Bayes theorem and used for solving classification problems.
- This is used when features are binary. So instead of using the frequency of the word, if you have discrete features in 1s and 0s that represent the presence or absence of a feature. In that case, the features will be binary and will use Bernoulli Naive Bayes.

```
from sklearn.naive_bayes import BernoulliNB
bayes_model1 = BernoulliNB()
bayes_model1.fit(x_train,y_train)
y_pred3 = bayes_model1.predict(x_test)
y_pred3
```

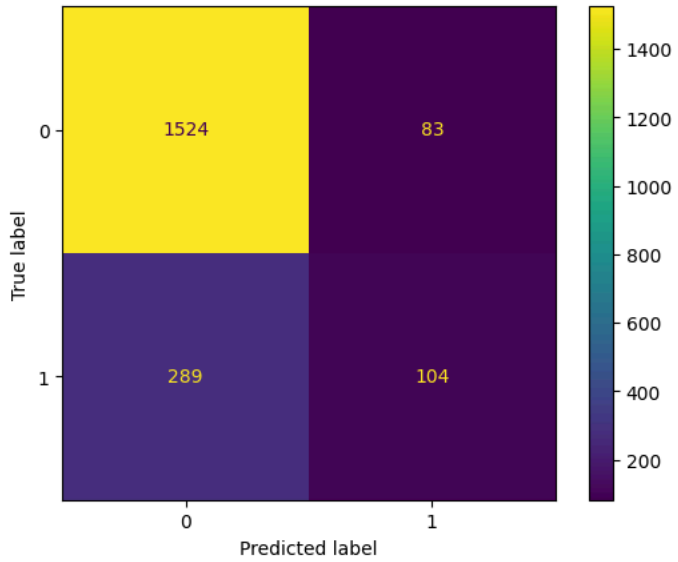
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred3)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1524   83]
 [ 289  104]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806c01ccd0>
```



From Naive Bayes ConfusionMatrix we observe that:

- 1524 customers are correctly classified under not “churned” category & 83 are misclassified
- 104 customers are correctly classified under “churned” category & 289 are misclassified

```
#Accuracy score and clasification report
from sklearn.metrics import accuracy_score,classification_report

c=accuracy_score(y_test,y_pred3)
print('Accuracy: ',c*100,'\n')
print(classification_report(y_test,y_pred3))
```

```
Accuracy:  81.39999999999999

              precision    recall  f1-score   support

           0       0.84      0.95      0.89      1607
           1       0.56      0.26      0.36       393

    accuracy                           0.81      2000
   macro avg       0.70      0.61      0.62      2000
weighted avg       0.78      0.81      0.79      2000
```

```
training_score = bayes_model1.score(x_train,y_train)
training_score
```

```
0.807125
```

```
testing_score = bayes_model1.score(x_train,y_train)
testing_score
```

```
0.807125
```

4) Support Vector Machine

Here the Machine Learning models learn from the past input data and predict the output.support vector machines are basically supervised learning models used for classification and regression analysis.

```
from sklearn.svm import SVC
sv_model=SVC()
sv_model.fit(x_train,y_train)
y_pred4=sv_model.predict(x_test)
y_pred4
```

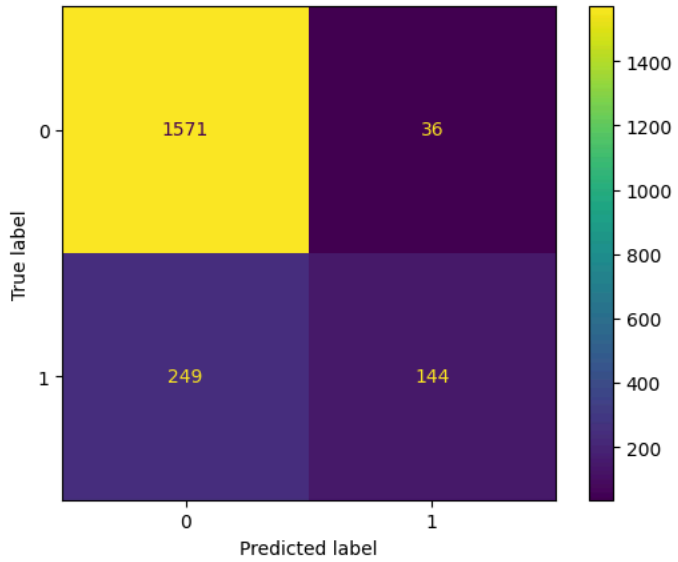
```
array([0, 0, 0, ..., 1, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred4)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1571   36]
 [ 249  144]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806c0f0190>
```



From Support Vector Machine ConfusionMatrix we observe that:

- 1571 customers are correctly classified under not “churned” category & 36 are misclassified
- 144 customers are correctly classified under “churned” category & 249 are misclassified

```
#Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

d=accuracy_score(y_test,y_pred4)
print('Accuracy: ',d*100,'\n')
print(classification_report(y_test,y_pred4))
```

```
Accuracy:  85.75

              precision    recall  f1-score   support

           0       0.86      0.98      0.92      1607
           1       0.80      0.37      0.50       393

    accuracy                           0.86      2000
   macro avg       0.83      0.67      0.71      2000
weighted avg       0.85      0.86      0.84      2000
```

```
training_score = sv_model.score(x_train,y_train)
training_score
```

```
0.862625
```

```
testing_score = sv_model.score(x_train,y_train)
testing_score
```

```
0.862625
```

5) DecisionTreeClassifier

Decision Trees(DTs) are a non-parametric supervised learning method used for classification and regression.the goal is to create the model that predict the value of a target variable by learning simple decision rules inferred from the data features.

```
from sklearn.tree import DecisionTreeClassifier
tree_model=DecisionTreeClassifier(max_depth = 5,criterion = 'entropy')
tree_model.fit(x_train,y_train)
y_pred5 = tree_model.predict(x_test)
y_pred5
```

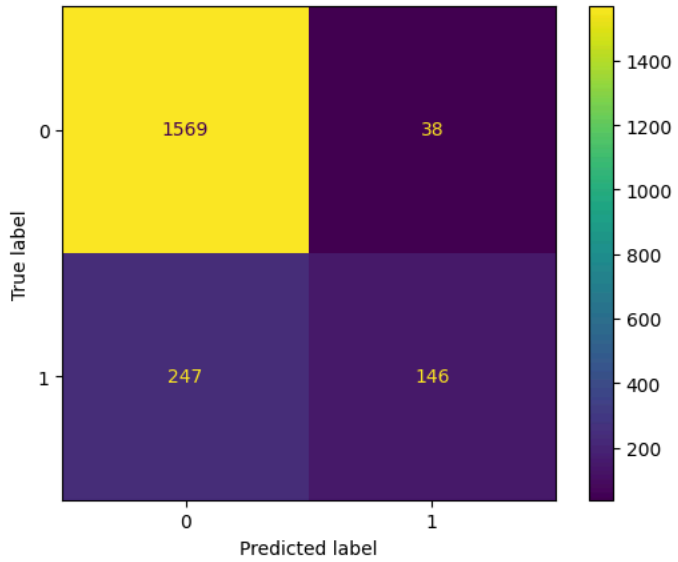
```
array([0, 0, 0, ..., 1, 0, 0])
```

```
#confusion matrixfrom sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred5)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1569   38]
 [ 247  146]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806af2fe80>
```



From DecisionTree ConfusionMatrix we observe that:

- 1569 customers are correctly classified under not “churned” category & 38 are misclassified
- 146 customers are correctly classified under “churned” category & 247 are misclassified

```
#Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

e=accuracy_score(y_test,y_pred5)
print('Accuracy: ',e*100,'\n')
print(classification_report(y_test,y_pred5))
```

```
Accuracy:  85.75

              precision    recall  f1-score   support

           0       0.86      0.98      0.92      1607
           1       0.79      0.37      0.51       393

    accuracy                           0.86      2000
   macro avg       0.83      0.67      0.71      2000
weighted avg       0.85      0.86      0.84      2000
```

```
training_score = tree_model.score(x_train,y_train)
training_score
```

```
0.857375
```

```
testing_score = tree_model.score(x_train,y_train)
testing_score
```

```
0.857375
```

6) RandomForest Classifier

A Random forest is a machine learning technique that is used to solve regression and classification problems.it utilizes ensemble learning,which is a technique that combines many classifiers to provide solution to complex problems.easy to use machine learning algorithm that produce,even without hyper-parameter tuning,a great result most of the time.

```
from sklearn.ensemble import RandomForestClassifier
forest_model = RandomForestClassifier(n_estimators=100,criterion='gini',max_depth=5,max_features=5)
forest_model.fit(x_train,y_train)
y_pred6 = forest_model.predict(x_test)
y_pred6
```

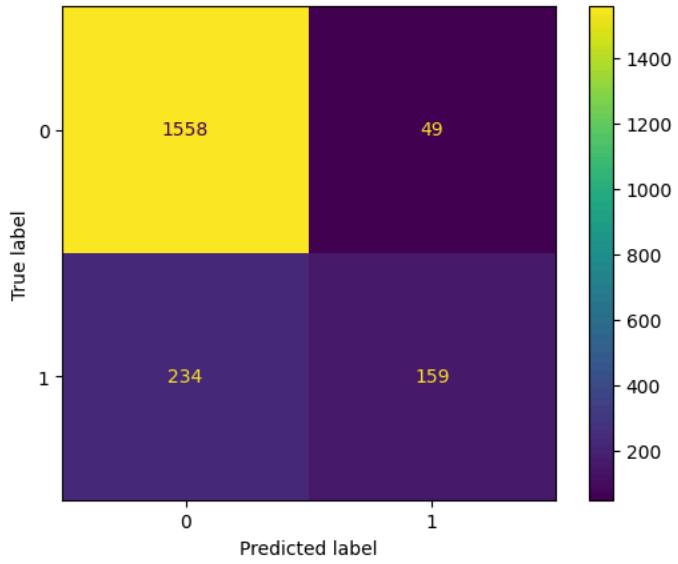
```
array([0, 0, 0, ..., 1, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result = confusion_matrix(y_test,y_pred6)
print(result)
labels=[0,1]
cmd = ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1558   49]
 [ 234  159]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806949a440>
```



From RandomForest ConfusionMatrix we observe that:

- 1558 customers are correctly classified under not “churned” category & 49 are misclassified
- 159 customers are correctly classified under “churned” category & 234 are misclassified

```
#Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

f=accuracy_score(y_test,y_pred6)
print('Accuracy: ',f*100,'\n')
print(classification_report(y_test,y_pred6))
```

```
Accuracy:  85.85000000000001

              precision    recall  f1-score   support

           0       0.87      0.97      0.92      1607
           1       0.76      0.40      0.53       393

    accuracy                           0.86      2000
   macro avg       0.82      0.69      0.72      2000
weighted avg       0.85      0.86      0.84      2000
```

```
training_score =forest_model.score(x_train,y_train)
training_score
```

```
0.858625
```

```
testing_score = forest_model.score(x_train,y_train)
testing_score
```

```
0.858625
```

7) AdaBoostClassifier

An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.

```
from sklearn.ensemble import AdaBoostClassifier
adb_model = AdaBoostClassifier(n_estimators=10,learning_rate=1.0)
adb_model.fit(x_train,y_train)
y_pred7=adb_model.predict(x_test)
y_pred7
```

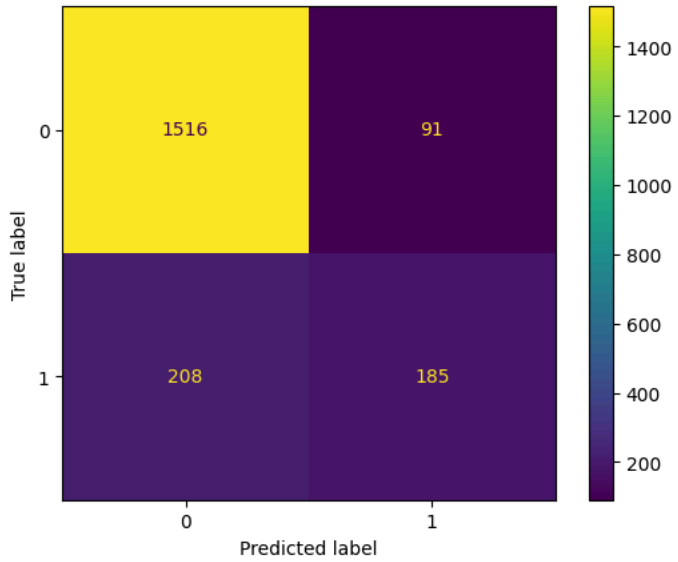
```
array([0, 0, 0, ..., 1, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result=confusion_matrix(y_test,y_pred7)
print(result)
labels=[0,1]
cmd=ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1516   91]
 [ 208  185]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806ae85930>
```



From AdaBoost ConfusionMatrix we observe that:

- 1516 customers are correctly classified under not “churned” category & 91 are misclassified
- 185 customers are correctly classified under “churned” category & 208 are misclassified

```
#accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

g=accuracy_score(y_test,y_pred7)
print('Accuracy: ',g*100,'\n')
print(classification_report(y_test,y_pred7))
```

```
Accuracy:  85.05

              precision    recall  f1-score   support
```

```
              0       0.88      0.94      0.91      1607
              1       0.67      0.47      0.55       393

       accuracy                           0.85      2000
      macro avg       0.77      0.71      0.73      2000
   weighted avg       0.84      0.85      0.84      2000
```

```
training_score = adb_model.score(x_train,y_train)
training_score
```

```
0.850375
```

```
training_score = adb_model.score(x_train,y_train)
training_score
```

```
0.850375
```

8) XgBoost

XgBoost is a gradient boosting algorithm for supervised learning. It's a highly efficient and scalable implementation of the boosting algorithm, with performance comparable to that of other state-of-the-art machine learning algorithms in most cases.

```
from xgboost import XGBClassifier
xg_model = XGBClassifier()
xg_model.fit(x_train,y_train)
y_pred8 = xg_model.predict(x_test)
y_pred8
```

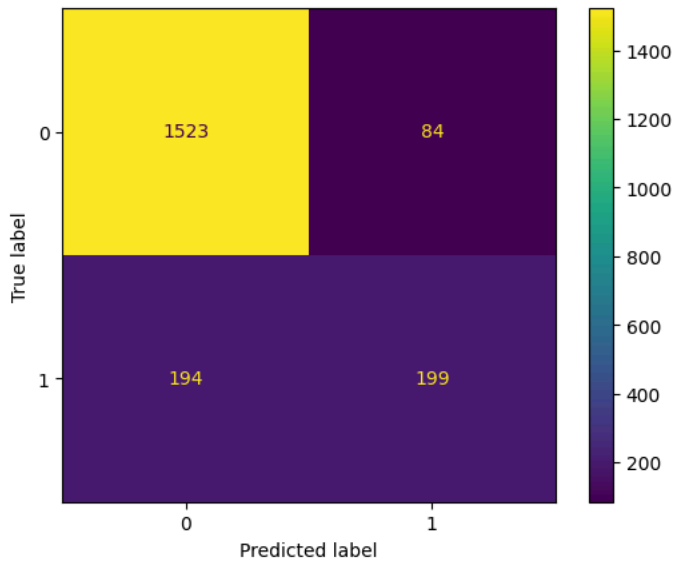
```
array([0, 0, 0, ..., 1, 0, 1])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result=confusion_matrix(y_test,y_pred8)
print(result)
labels=[0,1]
cmd=ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1523   84]
 [ 194  199]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806ae85a80>
```



From XgBoost ConfusionMatrix we observe that:

- 1523 customers are correctly classified under not "churned" category & 84 are misclassified
- 199 customers are correctly classified under "churned" category & 194 are misclassified

```
#Accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

h=accuracy_score(y_test,y_pred8)
print('Accuracy: ',h*100,'\n')
print(classification_report(y_test,y_pred8))
```

```
Accuracy:  86.1

              precision    recall  f1-score   support

           0       0.89      0.95      0.92      1607
           1       0.70      0.51      0.59       393

    accuracy                           0.86      2000
   macro avg       0.80      0.73      0.75      2000
weighted avg       0.85      0.86      0.85      2000
```

```
training_score = xg_model.score(x_train,y_train)
training_score
```

```
0.955375
```

```
training_score = xg_model.score(x_train,y_train)
training_score
```

```
0.955375
```

9) LogisticRegression

Logistic regression is one of the most popular machine learning algorithms,and its used for predict the categorical dependent variable using a given set of independent variable

```
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(x_train,y_train)
y_pred9 = lr_model.predict(x_test)
y_pred9
```

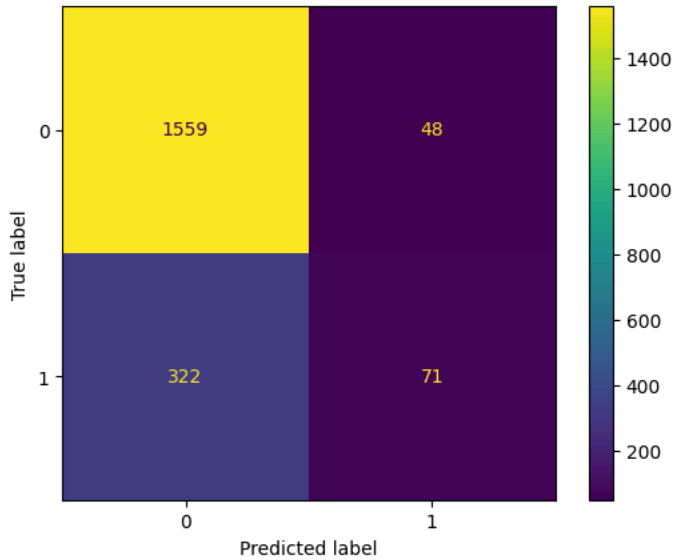
```
array([0, 0, 0, ..., 0, 0, 0])
```

```
#confusion matrix
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result=confusion_matrix(y_test,y_pred9)
print(result)
labels=[0,1]
cmd=ConfusionMatrixDisplay(result,display_labels=labels)
```

```
[[1559   48]
 [ 322   71]]
```

```
cmd.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b806ad49b40>
```



From Logistic Regression ConfusionMatrix we observe that:

- 1559 customers are correctly classified under not “churned” category & 48 are misclassified.
- 71 customers are correctly classified under “churned” category & 322 are misclassified

```
#accuracy score and classification report
from sklearn.metrics import accuracy_score,classification_report

i=accuracy_score(y_test,y_pred9)
print('Accuracy: ',i*100,'\n')
print(classification_report(y_test,y_pred9))
```

```
     Accuracy:  81.5

                   precision    recall  f1-score   support

                0       0.83      0.97      0.89      1607
                1       0.60      0.18      0.28       393

             accuracy                           0.81      2000
            macro avg       0.71      0.58      0.59      2000
         weighted avg       0.78      0.81      0.77      2000
```

```
training_score = lr_model.score(x_train,y_train)
training_score
```

```
     0.807125
```

```
testing_score = lr_model.score(x_train,y_train)
testing_score
```

```
     0.807125
```

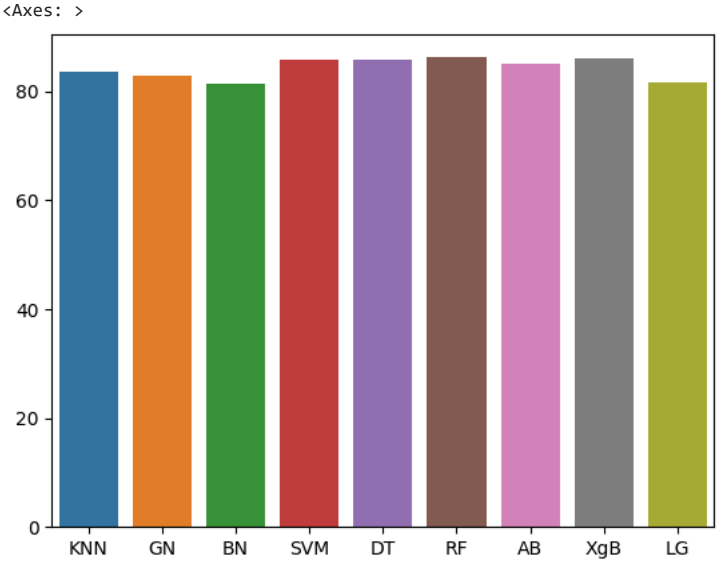
Accuracy Score Comparison

```
dic={'model':['KNN','GN','BN','SVM','DT','RF','AB','XgB','LG'],'accuracy score':[83.55,82.85,81.39,85.75,85.75,86.2,85.05,86.1,81.5]}
result=pd.DataFrame(dic)
result
```

	model	accuracy score
0	KNN	83.55
1	GN	82.85
2	BN	81.39
3	SVM	85.75
4	DT	85.75
5	RF	86.20
6	AB	85.05
7	XgB	86.10
8	LG	81.50

```
sns.barplot(x='model',y='accuracy score',data=dic)
```

```
<Axes: >
```



conclusion

From the perfomance evaluation RandomForest have highest accuracy.and also using other perfomance evaluation precision value is high for RandomForest .so we choose RandomForest is the best model

To Check The Performance Of The Balanced Dataset

Balancing an imbalanced data set

Balancing an imbalanced dataset is an important preprocessing step when working with machine learning models, especially in classification tasks. There are several techniques you can use to balance your dataset. Here, I'll use the method: **oversampling.**

Oversampling

Oversampling is a technique used in machine learning to address class imbalance by increasing the number of instances in the minority class (the less frequent class). This helps to balance the class distribution, which can lead to better model performance.

Model with implementation of SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) - For data balancing

```
from imblearn.over_sampling import SMOTE

sm=SMOTE()

x_res,y_res = sm.fit_resample(x,y)


y.value_counts()

     0    7963
     1    2037
     Name: Exited, dtype: int64


y_res.value_counts()

     1    7963
     0    7963
     Name: Exited, dtype: int64


from sklearn.model_selection import train_test_split

x_train1,x_test1,y_train1,y_test1 = train_test_split(x_res,y_res,test_size=0.20,random_state=0)
```

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

```
forest_model2 = RandomForestClassifier(n_estimators=100,criterion='gini',max_depth=5,max_features=5)
forest_model2.fit(x_train1,y_train1)

y_pred = forest_model2.predict(x_test1)
y_pred
```

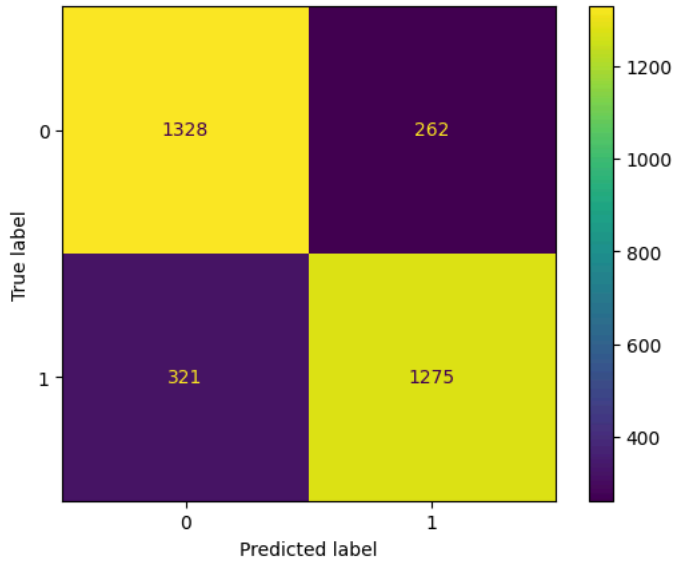
```
array([1, 0, 1, ..., 0, 1, 0])
```

```
from sklearn.metrics import confusion_matrix,ConfusionMatrixDisplay
result_os = confusion_matrix(y_test1,y_pred)
print(result_os)
labels=[0,1]
cmd_os = ConfusionMatrixDisplay(result_os,display_labels=labels)
```

```
[[1328  262]
 [ 321 1275]]
```

```
cmd_os.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7b8068e889d0>
```




```
training_score = forest_model2.score(x_train1,y_train1)
training_score
```

```
0.8208006279434851
```

```
testing_score = forest_model2.score(x_train1,y_train1)
testing_score
```

```
0.8208006279434851
```