CUSTOMER CHURN MODELLING

The dataset for modelling was obtained from Kaggle platform.This dataset includes demographic information such as customer age, gender, and geographical location and various banking-related attributes are captured, including customer account balance, the number of products held, credit card usage, and transaction histor.

The target variable for developing predictive models to forecast churn probability is a binary churn label,1 indicating whether a customer has churned/ exited the bank or 0 if not

The following were the steps followed

i) Loading the dataset

ii)Explonatory data analysis

iii)Data preprocessing

iv) Models training and testing

v)Deployment of best model

Amongst 9 models which were trained,Cat boost classifier had the highest accuracy(87.37%).Followed closely bt XGBoost Classifier(87.17%),Gradient boosting Classief(87%),Random Forest(86.93%)

Other models had the accuracy around 80%, these models are Logistic Regression,Support Vector Machine,Artificial Neural Network,K-Nearest Neighbours and Gaussian Naive Bayes.

In [1]:
```
# importing relevant packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()
%matplotlib inline
```

In [2]:
```
#loading the dataset
data=pd.read_csv("Desktop/churn_modelling.csv")
```

In [3]:
```
#first five rows of the data
data.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMembe
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	

In [4]:
```
#columns in data
data.columns
```

Out[4]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

Tenure-number of years customer has been in the bank

Number of products the customer is utilising

HasCrCrad-whether customer held a credit card with the bank or not

Exited-1 represent the customer left the bank(closed account) and 0 if the customer is retained(continues to be a customer)

In [5]:
```
#data types of the features
data.dtypes
```

Out[5]:
```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

In [6]:
```
#information on data
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 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
```

In [7]:

```
#descriptive statistics
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.500250e+03	10000.00
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.575323e+07	15815690.00
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.180000e+02	850.00
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.400000e+01	92.00
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.000000e+00	10.00
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.276442e+05	250898.09
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.000000e+00	4.00
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.000000e+00	1.00
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.000000e+00	1.00
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.493882e+05	199992.48
Exited	10000.0	2.037000e-01	0.402769	0.00	0.00	0.000000e+00	0.000000e+00	1.00

In [8]:

```
#checking for null values
data.isnull().sum().sum()
```

Out[8]:

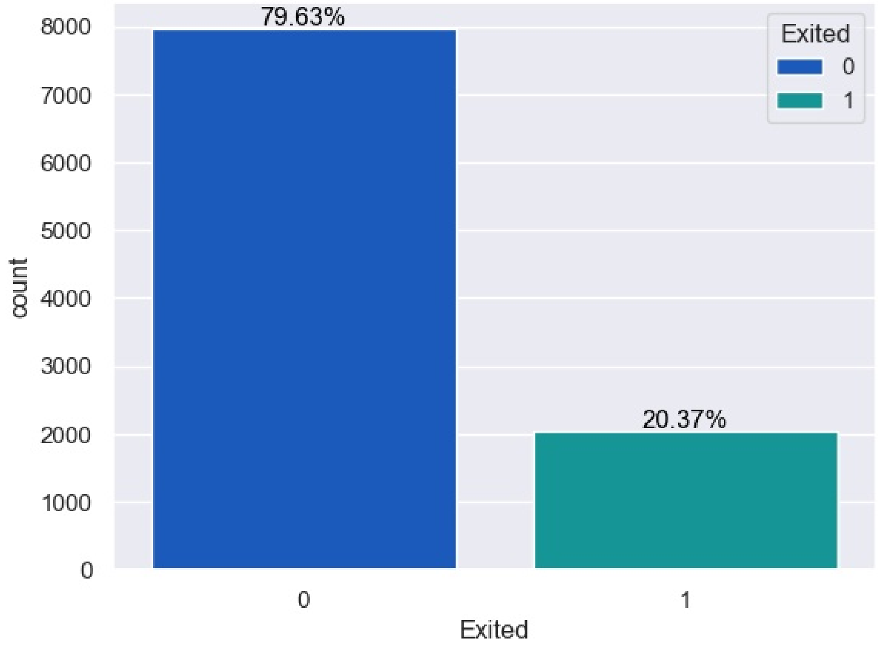
```
0
```

EXPLONATORY DATA ANALYSIS

In [9]:

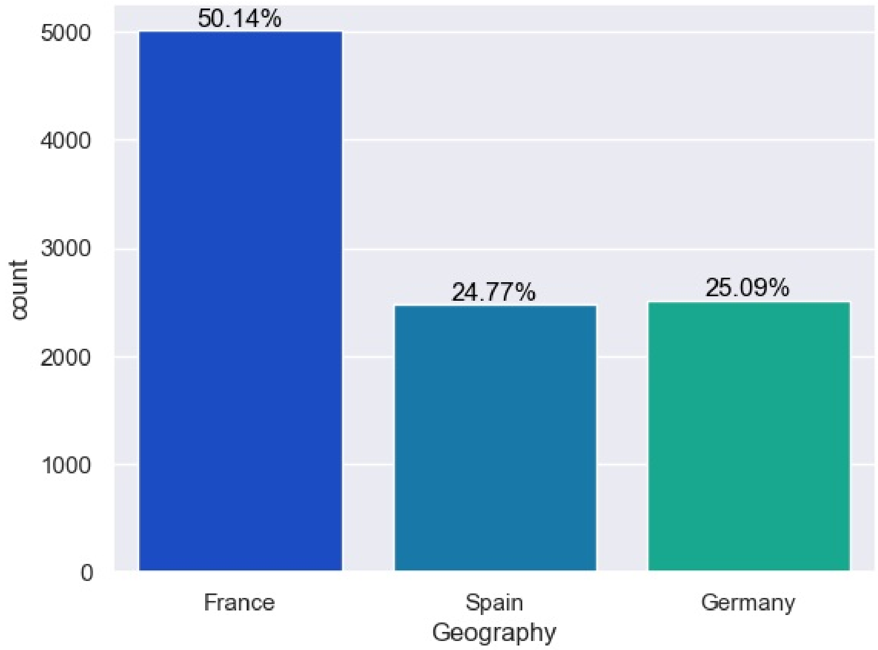
```
#percentage countplot on target variable(those who exited and thoes who did not)
ax=sns.countplot(x='Exited',data=data,hue='Exited',palette='winter')
total = float(len(data['Exited']))
for p in ax.patches:
    height = p.get_height()
    if height!=0:
        ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)

plt.show()
```



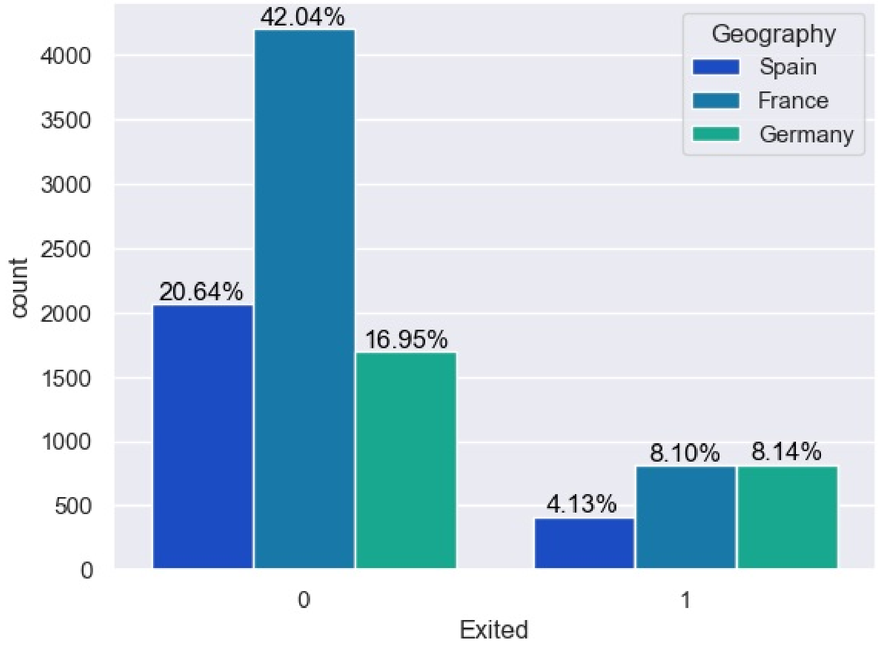
The results shows that 79.63% of the customers were retained whereas 20.37% exited the bank

In [10]:
```
#percentage countplot on Geographical location
ax=sns.countplot(x='Geography',data=data,hue='Geography',palette='winter')
total = float(len(data['Geography']))
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)
plt.show()
```



Majority of the customers(50.14%)registered in this bank came from France,24.77% are from Spain and 25.09% are Germans

In [11]:
```
# percentage countplot on those who exited based on geographical area
ax=sns.countplot(x='Exited',data=data,hue='Geography',palette='winter')
total = float(len(data['Exited']))
for p in ax.patches:
    height = p.get_height()
    if height!=0:
        ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)
plt.show()
```



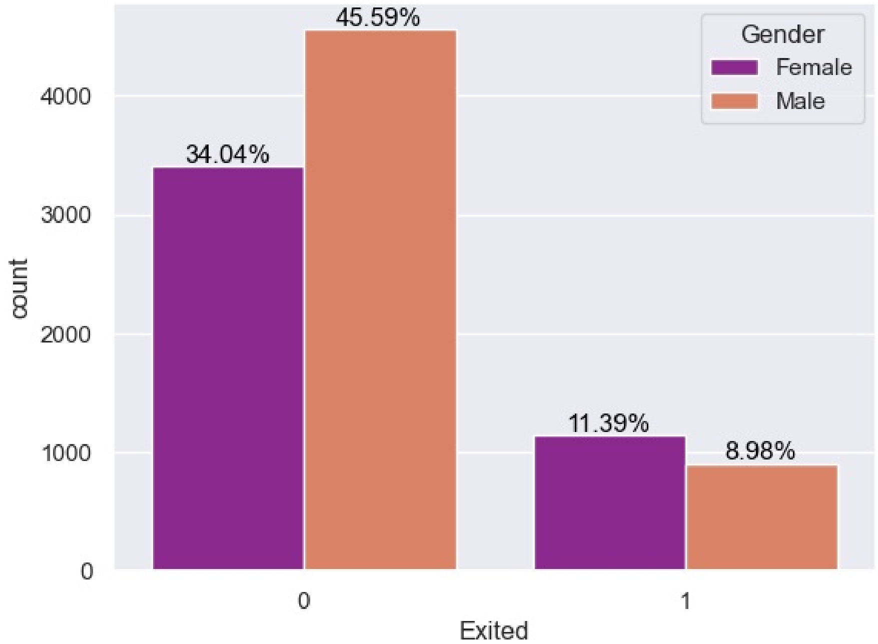
From France, 42.04% did not exit the bank whereas 8.10% exited the bank

From Spain,20.64% didn't exit the bank whereas 4.13% exited the bank

From Germany,16.95%did not exit the bank whereas 8.14% exited the bank

In [12]:

```
#percentage countplot on those who exited based on gender
ax=sns.countplot(x='Exited',data=data,hue='Gender',palette='plasma')
total = float(len(data['Exited']))
for p in ax.patches:
    height = p.get_height()
    if height!=0:
        ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)
plt.show()
```



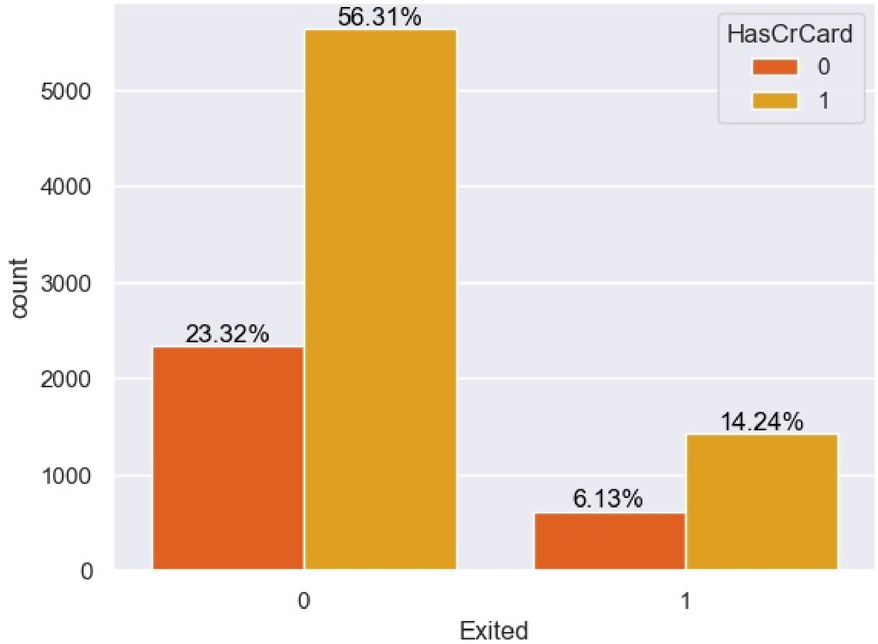
Amongst female 34.04% did not exit the bank whereas 11.39% exited

For male gender,45.59% did not exit the bank,8.98% exited.

This also show that male were more than female by 9.14%

In [13]:

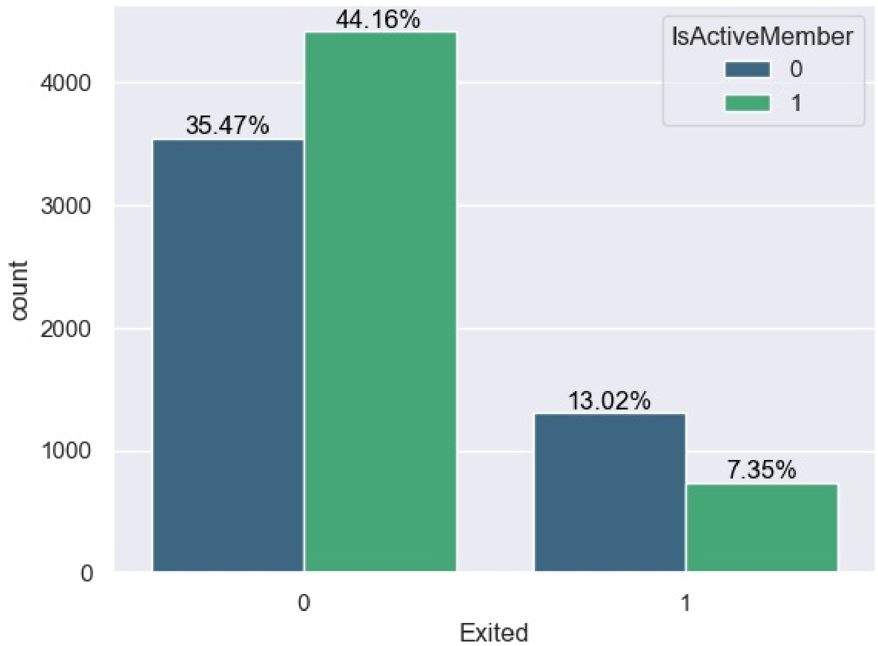
```
# countplot on those who exited based on if a person has a credit card or not
ax=sns.countplot(x='Exited',data=data,hue='HasCrCard',palette='autumn')
total = float(len(data['Exited']))
for p in ax.patches:
    height = p.get_height()
    if height!=0:
        ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)
plt.show()
```



Of those who did not exited,23.32% had credit card whereas 56.13% did not have credit card

Of those who exited,6.13% had credit card whereas 14.24% did not have credit card

In [14]:
```
# countplot on those who exited based on if a member is active
ax=sns.countplot(x='Exited',data=data,hue='IsActiveMember',palette='viridis')
total = float(len(data['Exited']))
for p in ax.patches:
    height = p.get_height()
    if height!=0:
        ax.annotate(f'{height / total:.2%}',(p.get_x() + p.get_width() / 2., height),
                ha='center', va='bottom', color='black', fontsize=12)
plt.show()
```



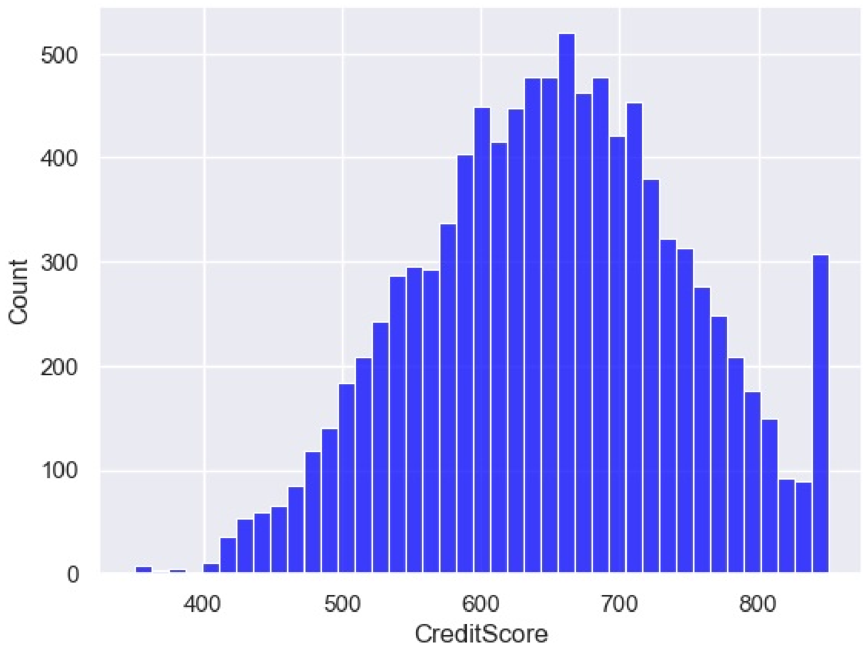
Of those who did not exited,35.47% are active members whereas 44.16% are not active members

Of those who exited,13.02% have been active members whereas 7.35% are not active members

Plots on continous data attributes

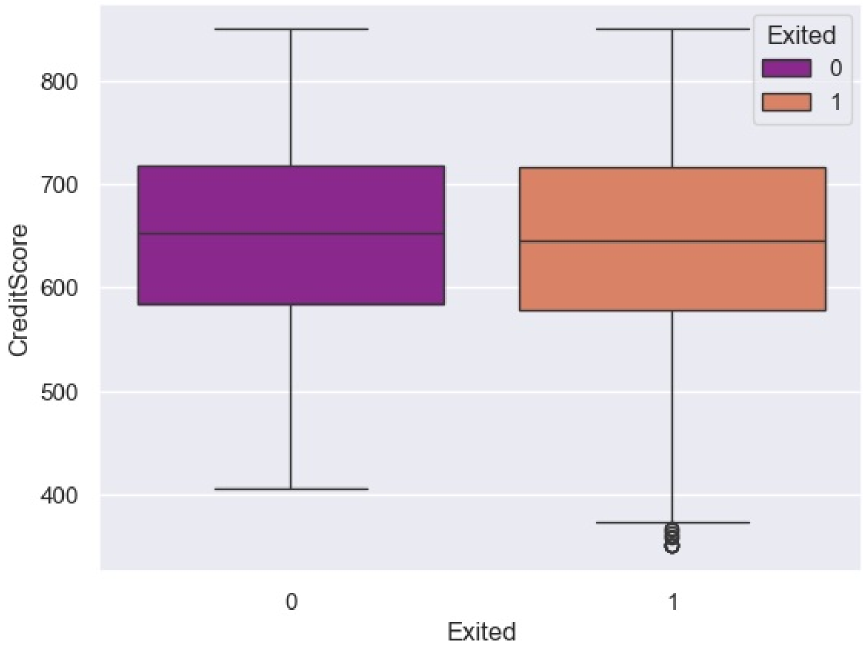
In [15]:
```
#histogram on credit score which is seen to be normal distributed
sns.histplot(data['CreditScore'],color='blue')
```

Out[15]: <Axes: xlabel='CreditScore', ylabel='Count'>



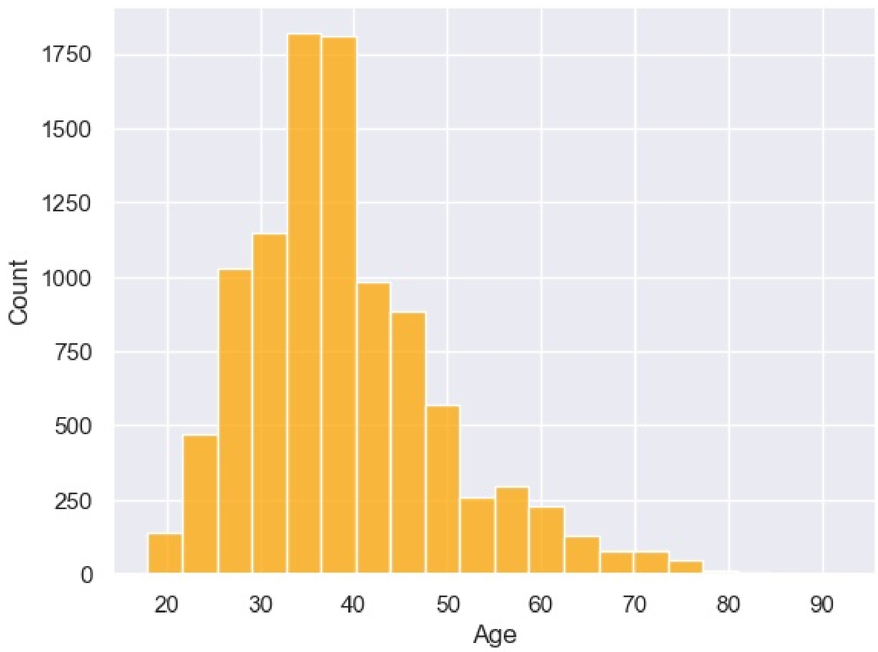
In [16]:
```
#boxplot of those who exited based on credit score
sns.boxplot(x='Exited',y='CreditScore',data=data,hue='Exited',legend=True,palette='plasma')
```

Out[16]: <Axes: xlabel='Exited', ylabel='CreditScore'>

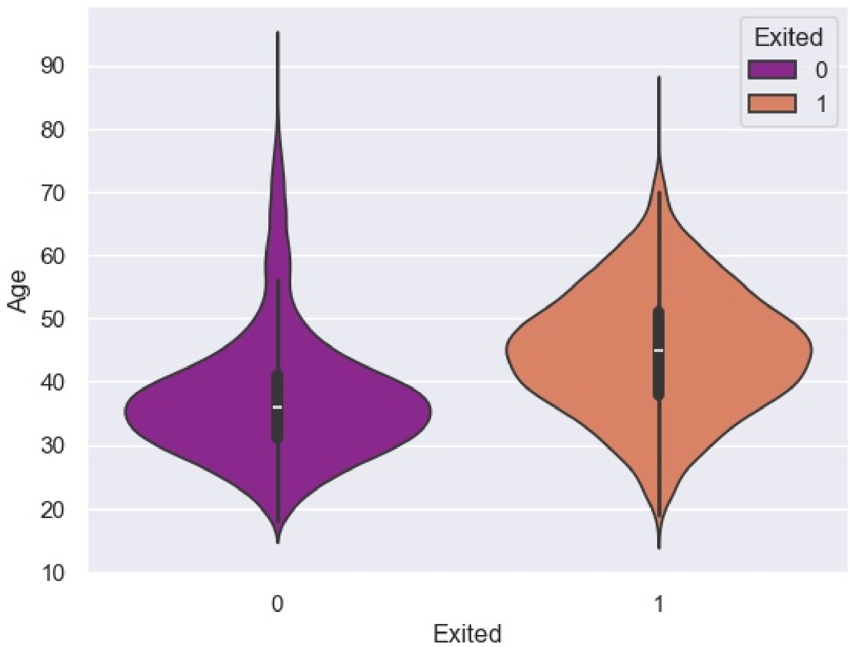


In [17]:
```
# histogram on age
sns.histplot(data['Age'],bins=20,color='orange')
```

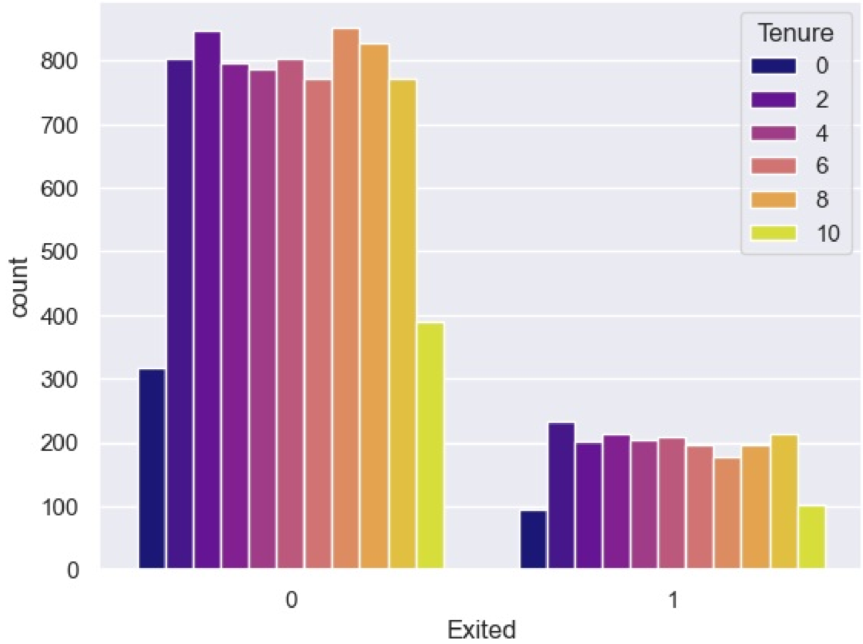
Out[17]: <Axes: xlabel='Age', ylabel='Count'>



In [18]:
```
#violinplot of those who exited based on their age
sns.violinplot(x='Exited',y='Age',data=data,hue='Exited',legend=True,palette='plasma')
```
Out[18]: <Axes: xlabel='Exited', ylabel='Age'>

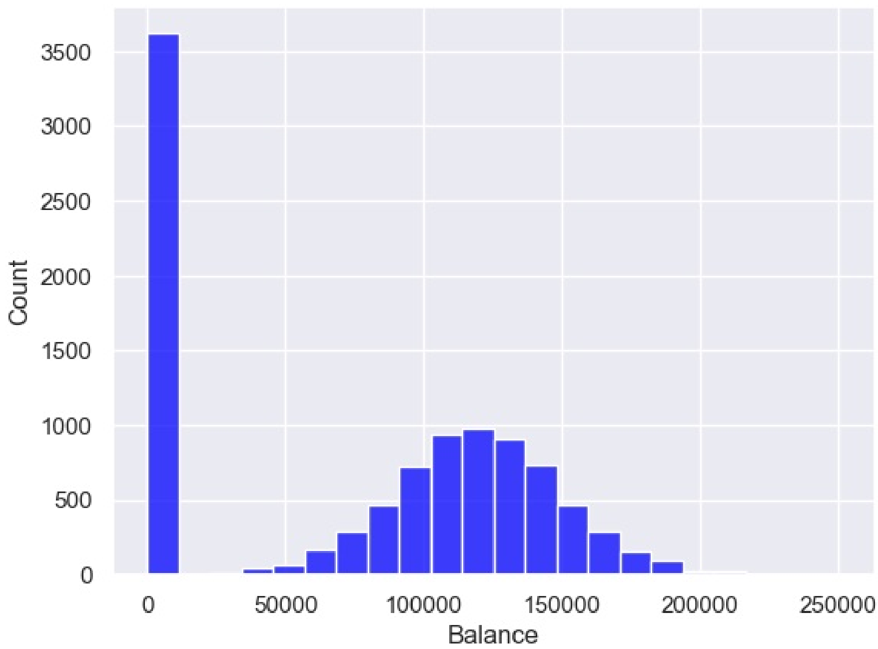


In [19]:
```
## countplot on those who exited based on their tenure
sns.countplot(x='Exited',data=data,hue='Tenure',palette='plasma',legend=True)
```
Out[19]: <Axes: xlabel='Exited', ylabel='count'>



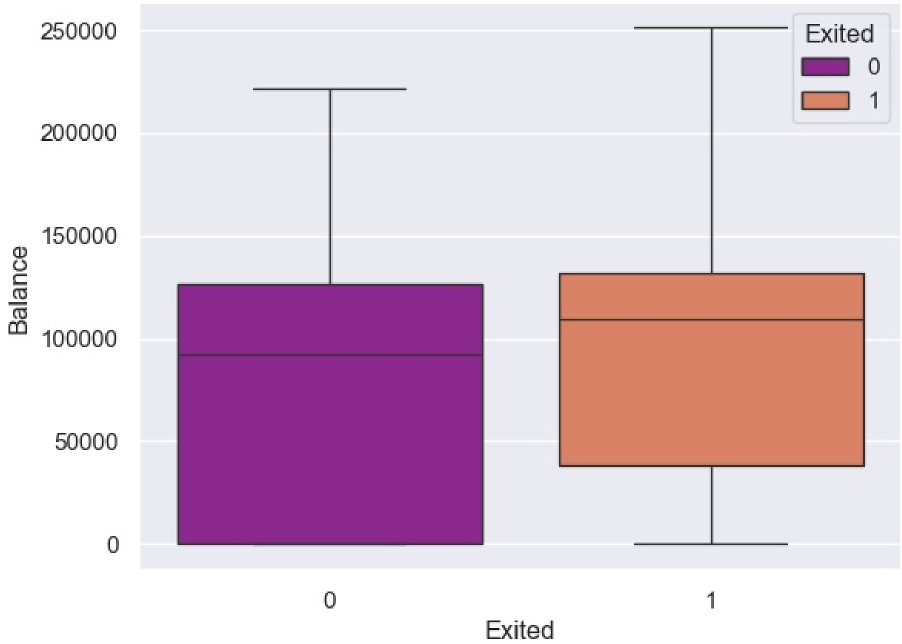

```
In [20]: #histogram on bank balances
sns.histplot(data['Balance'],color='blue')
```

```
Out[20]: <Axes: xlabel='Balance', ylabel='Count'>
```



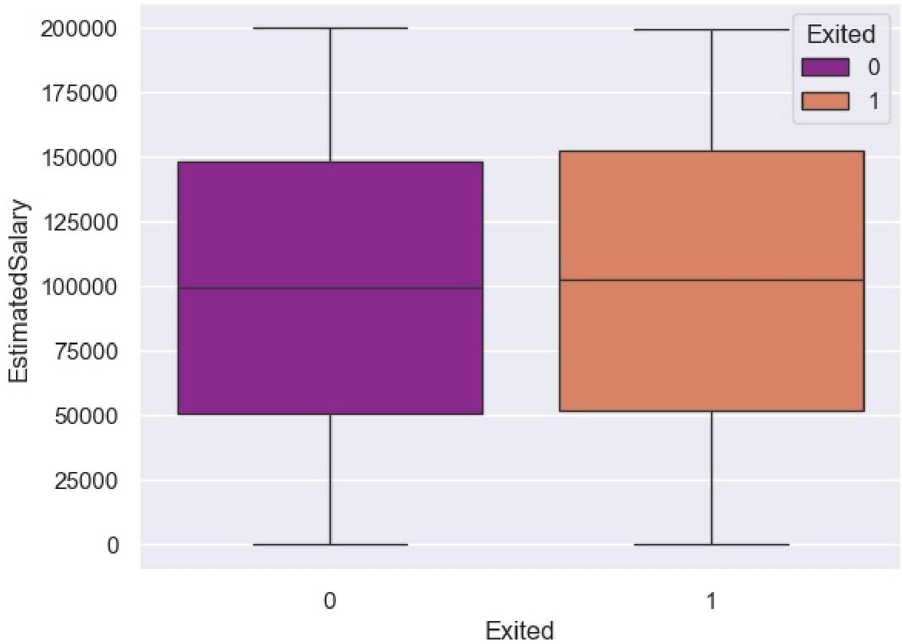

```
In [21]: #compairing those who exited based on bank balances
sns.boxplot(x='Exited',y='Balance',data=data,hue='Exited',legend=True,palette='plasma')
```

```
Out[21]: <Axes: xlabel='Exited', ylabel='Balance'>
```




```
In [22]: #compairing those who exited based on their estimated salary
         sns.boxplot(x='Exited',y='EstimatedSalary',data=data,hue='Exited',legend=True,palette='plasma')
```

```
Out[22]: <Axes: xlabel='Exited', ylabel='EstimatedSalary'>
```



Data preprocessing

```
In [23]: #importing labelencoder for changing object variables to int or object type
         from sklearn.preprocessing import LabelEncoder
```

```
In [24]: #calling an instance of label
         label_encoder = LabelEncoder()
```

```
In [25]: #transforming object variables
         data['Geography'] = label_encoder.fit_transform(data['Geography'])
```

```python
data['Gender'] = label_encoder.fit_transform(data['Gender'])
```

In [26]:
```python
#removing the unneccesary features from the dataset
data.drop(['RowNumber','CustomerId','Surname'],axis=1,inplace=True)
```

In [27]:
```python
#checking for head of remaining dataset
data.head()
```

Out[27]:

	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

In [28]:
```python
#shape of data
data.shape
```

Out[28]:

(10000, 11)

In [29]:
```python
#correlation between features
data.corr()
```

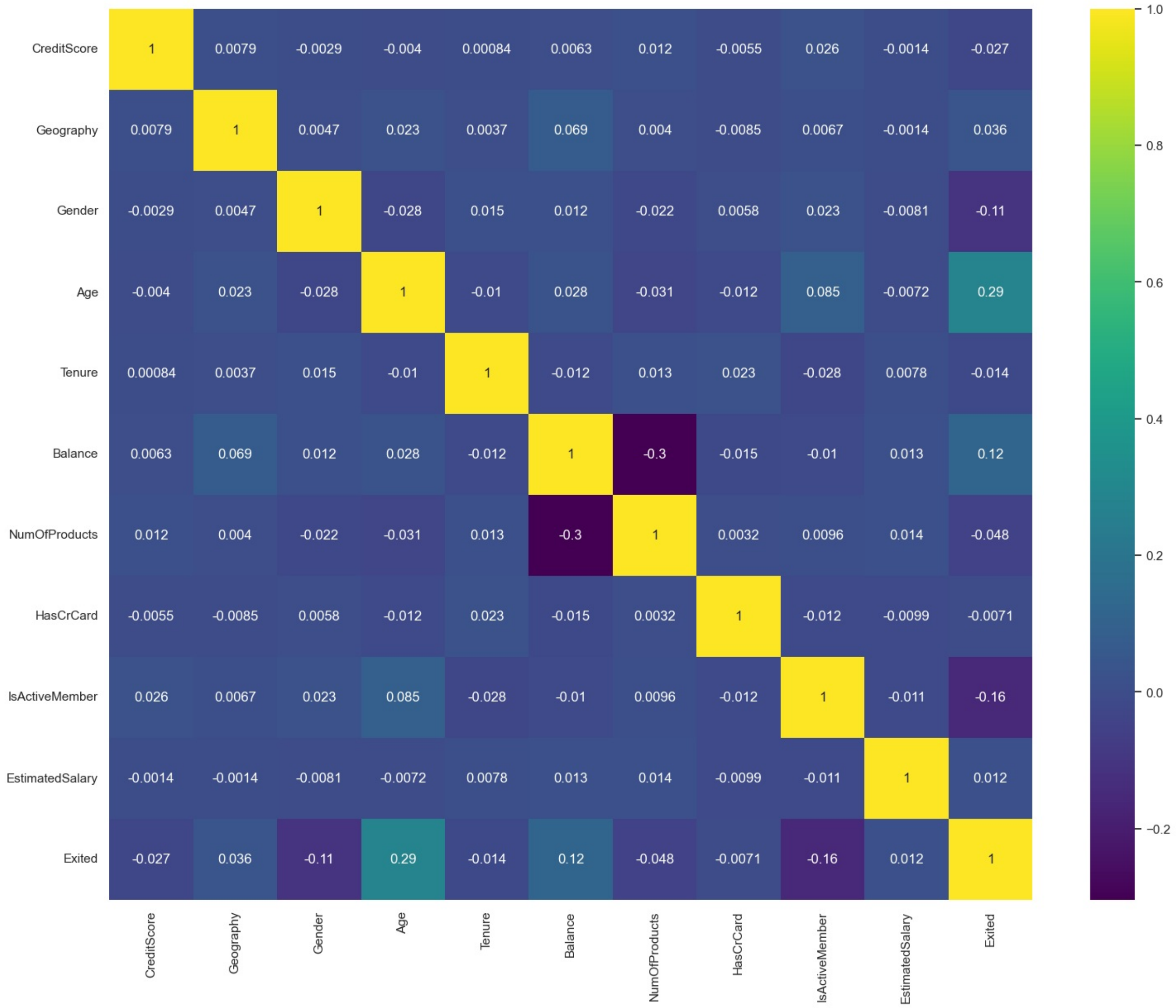
Out[29]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedS
CreditScore	1.000000	0.007888	-0.002857	-0.003965	0.000842	0.006268	0.012238	-0.005458	0.025651	-0.00
Geography	0.007888	1.000000	0.004719	0.022812	0.003739	0.069408	0.003972	-0.008523	0.006724	-0.00
Gender	-0.002857	0.004719	1.000000	-0.027544	0.014733	0.012087	-0.021859	0.005766	0.022544	-0.00
Age	-0.003965	0.022812	-0.027544	1.000000	-0.009997	0.028308	-0.030680	-0.011721	0.085472	-0.00
Tenure	0.000842	0.003739	0.014733	-0.009997	1.000000	-0.012254	0.013444	0.022583	-0.028362	0.00
Balance	0.006268	0.069408	0.012087	0.028308	-0.012254	1.000000	-0.304180	-0.014858	-0.010084	0.0
NumOfProducts	0.012238	0.003972	-0.021859	-0.030680	0.013444	-0.304180	1.000000	0.003183	0.009612	0.0
HasCrCard	-0.005458	-0.008523	0.005766	-0.011721	0.022583	-0.014858	0.003183	1.000000	-0.011866	-0.00
IsActiveMember	0.025651	0.006724	0.022544	0.085472	-0.028362	-0.010084	0.009612	-0.011866	1.000000	-0.0
EstimatedSalary	-0.001384	-0.001369	-0.008112	-0.007201	0.007784	0.012797	0.014204	-0.009933	-0.011421	1.00
Exited	-0.027094	0.035943	-0.106512	0.285323	-0.014001	0.118533	-0.047820	-0.007138	-0.156128	0.0

In [30]:
```python
#heatmaap of features
plt.figure(figsize=(18,14))
sns.heatmap(data.corr(),annot=True,cmap='viridis')
```

Out[30]:

<Axes: >



SPLITTING THE DATA

In [31]:
```
#Splitting data
```

In [32]:
```
#declaring  dependent and independent variables
X=data.drop('Exited',axis=1)
```

```
y=data['Exited']
```

In [33]:
```
from sklearn.model_selection import train_test_split
```

In [34]:
```
# data splitting to train set and test set
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=42)
```

In [35]:
```
#Standardizing the dataset
```

In [36]:
```
from sklearn.preprocessing import StandardScaler
```

In [37]:
```
scaler=StandardScaler()
```

In [38]:
```
scaler.fit(X_train)
```

Out[38]:
```
▾ StandardScaler
StandardScaler()
```

In [39]:
```
pd.DataFrame(scaler.transform(X_train))
```

Out[39]:

	0	1	2	3	4	5	6	7	8	9
0	-0.344595	1.507307	-1.098232	-0.656750	-0.342170	1.583725	0.819663	0.645981	0.970714	1.248214
1	-0.095181	0.302012	-1.098232	-0.466380	0.698162	1.344106	-0.903352	-1.548034	0.970714	1.521225
2	-0.947345	-0.903282	0.910554	-0.561565	0.351385	-1.222055	0.819663	-1.548034	-1.030169	1.263615
3	-0.354987	0.302012	0.910554	0.199916	1.044940	-0.618965	-0.903352	0.645981	0.970714	1.646839
4	0.642668	-0.903282	0.910554	-0.180824	1.391718	1.152808	0.819663	-1.548034	0.970714	0.875112
...	...	...	...	...	...	...	...	...	...	...
6995	1.203850	-0.903282	0.910554	1.437322	1.044940	-0.106936	-0.903352	0.645981	0.970714	-0.545387
6996	0.310116	-0.903282	-1.098232	1.818063	-1.382503	-1.222055	-0.903352	0.645981	0.970714	-1.736501
6997	0.860905	-0.903282	-1.098232	-0.085639	-1.382503	-1.222055	2.542677	-1.548034	-1.030169	-0.149259
6998	0.154233	-0.903282	0.910554	0.390286	1.044940	1.820806	-0.903352	0.645981	-1.030169	-0.057544
6999	0.466000	0.302012	0.910554	1.151767	-1.382503	1.143904	-0.903352	0.645981	0.970714	-0.819426

7000 rows × 10 columns

In [40]:
```
pd.DataFrame(scaler.transform(X_test))
```

Out[40]:

	0	1	2	3	4	5	6	7	8	9
0	-0.583617	0.302012	0.910554	-0.656750	-0.688948	0.324894	0.819663	-1.548034	-1.030169	-1.023964
1	-0.303026	-0.903282	0.910554	0.390286	-1.382503	-1.222055	0.819663	0.645981	0.970714	0.790096
2	-0.531655	1.507307	-1.098232	0.485471	-0.342170	-1.222055	0.819663	0.645981	-1.030169	-0.733048
3	-1.518919	0.302012	0.910554	1.913248	1.044940	0.683891	0.819663	0.645981	0.970714	1.211571
4	-0.957737	1.507307	-1.098232	-1.132675	0.698162	0.777369	-0.903352	0.645981	0.970714	0.240116
...	...	...	...	...	...	...	...	...	...	...
2995	0.819336	1.507307	-1.098232	0.009546	-1.035726	0.806485	-0.903352	0.645981	0.970714	-0.450516
2996	-1.217544	-0.903282	-1.098232	-0.751935	0.698162	0.567168	0.819663	0.645981	0.970714	-1.119522
2997	-0.448517	-0.903282	0.910554	-0.656750	0.698162	-0.072394	-0.903352	0.645981	0.970714	0.886280
2998	-0.749893	-0.903282	0.910554	-0.751935	-1.035726	-1.222055	0.819663	-1.548034	0.970714	-0.638471
2999	-1.238328	0.302012	-1.098232	-1.608601	1.738495	0.629369	-0.903352	-1.548034	-1.030169	-0.988171

3000 rows × 10 columns

TRAINING THE MODELS

MODEL ONE: LOGISTIC REGRESSION MODEL(LG)

In [41]:
```
#importing,fitting and testing the mode
```

In [42]:
```
#importing Log model
from sklearn.linear_model import LogisticRegression
```

In [43]:
```
logmodel=LogisticRegression()
```

In [44]:
```
logmodel.fit(X_train,y_train)
```

Out[44]:

```
▾ LogisticRegression
LogisticRegression()
```

In [45]:
```
predict1=logmodel.predict(X_test)
```

In [46]:
```
logmodel.coef_
```

Out[46]:
```
array([[-4.97852492e-03,  3.25986910e-04, -1.06140216e-03,
         4.40881365e-02, -1.86520775e-03,  3.74394175e-06,
        -4.48691633e-04, -2.49477736e-04, -1.39479800e-03,
        -1.59209564e-06]])
```

In [47]:
```
logmodel.intercept_
```

Out[47]:
```
array([-0.00017916])
```

In [48]:
```
logmodel.score(X_test,y_test)
```

Out[48]:
```
0.8006666666666666
```

In [49]:
```
from sklearn.metrics import classification_report,confusion_matrix
```

In [50]:
```
print(confusion_matrix(y_test,predict1))
print('\n')
print(classification_report(y_test,predict1))
```

```
[[2354   62]
 [ 536   48]]


              precision    recall  f1-score   support

           0       0.81      0.97      0.89      2416
           1       0.44      0.08      0.14       584

    accuracy                           0.80      3000
   macro avg       0.63      0.53      0.51      3000
weighted avg       0.74      0.80      0.74      3000
```

MODELTWO: SUPPORT VECTOR MACHINE MODEL (SVC)

In [51]:
```
#importing,fitting and testing the model
```

In [52]:
```
from sklearn.svm import SVC
```

In [53]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [54]:
```
model=SVC()
```

In [55]:
```
model.fit(X_train,y_train)
```

Out[55]:

```
▾ SVC
SVC()
```

In [56]:
```
predict2=model.predict(X_test)
```

In [57]:
```
print(confusion_matrix(y_test,predict2))
print('\n')
print(classification_report(y_test,predict2))
```

```
[[2416    0]
 [ 584    0]]


              precision    recall  f1-score   support

           0       0.81      1.00      0.89      2416
           1       0.00      0.00      0.00       584

    accuracy                           0.81      3000
   macro avg       0.40      0.50      0.45      3000
weighted avg       0.65      0.81      0.72      3000
```

In [58]:
```
## GridSearchCV ON SVC
```

In [59]:
```
from sklearn.model_selection import GridSearchCV
```

In [60]:
```
param_grid={'C':[10,100,1000],'gamma':[1,0.1,0.01]}
```

In [61]:
```
grid=GridSearchCV(SVC(),param_grid,verbose=0)
```

In [62]:
```
grid.fit(X_train,y_train)
```

Out[62]:
```
▸ GridSearchCV
▸ estimator: SVC
  ▸ SVC
```

In [63]:
```
grid.pred=grid.predict(X_test)
```

In [64]:
```
print(confusion_matrix(y_test,grid.pred))
print('\n')
print(classification_report(y_test,grid.pred))
```

```
[[2416    0]
 [ 584    0]]


              precision    recall  f1-score   support

           0       0.81      1.00      0.89      2416
           1       0.00      0.00      0.00       584

    accuracy                           0.81      3000
   macro avg       0.40      0.50      0.45      3000
weighted avg       0.65      0.81      0.72      3000
```

MODEL THREE: RANDOM FOREST CLASSIFIER MODEL(RF)

In [65]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [66]:
```
rfc=RandomForestClassifier()
```

In [67]:
```
rfc.fit(X_train,y_train)
```

Out[67]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

In [68]:
```
predict3=rfc.predict(X_test)
```

In [69]:
```
print(confusion_matrix(y_test,predict3))
print('\n')
print(classification_report(y_test,predict3))
```

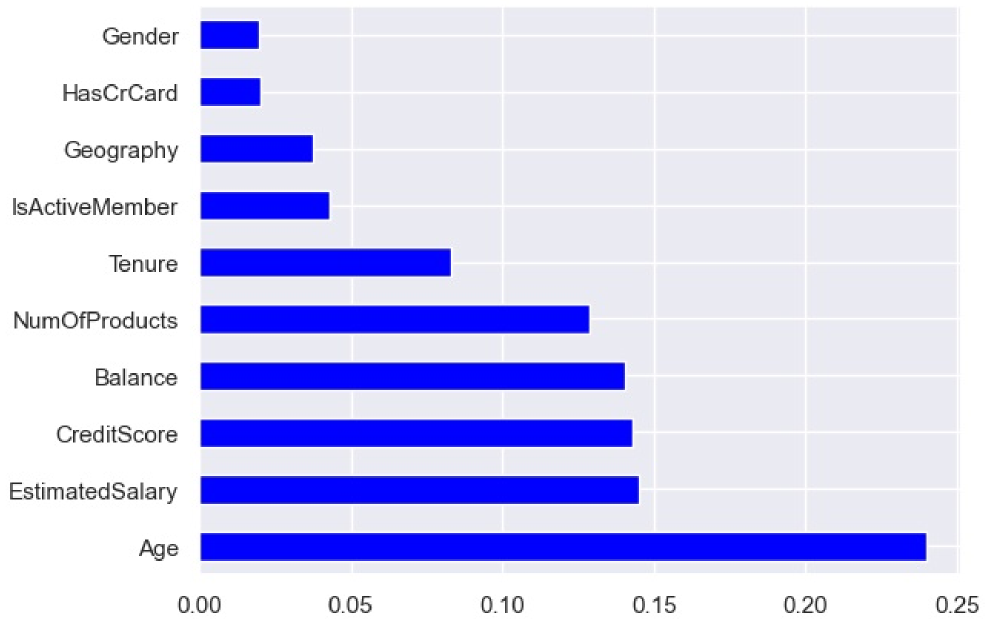
```
[[2338   78]
 [ 317  267]]


              precision    recall  f1-score   support

           0       0.88      0.97      0.92      2416
           1       0.77      0.46      0.57       584

    accuracy                           0.87      3000
   macro avg       0.83      0.71      0.75      3000
weighted avg       0.86      0.87      0.85      3000
```

In [70]:
```
#checking importance of variables using barplot in RFC
feat_importances = pd.Series(rfc.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh',color='blue')
plt.show()
```



MODEL FOUR: CAT BOOST CLASSIFIER MODEL(CBC)

In [71]:
```
#importing,fitting and testing the model
```

In [72]:
```
 from catboost import CatBoostClassifier
```

In [73]:
```
cbc=CatBoostClassifier(verbose=False)
```

In [74]:
```
cbc.fit(X_train,y_train)
```

Out[74]:
```
<catboost.core.CatBoostClassifier at 0x22e1140c9d0>
```

In [75]:
```
predict4=cbc.predict(X_test)
```

In [76]:
```
print(confusion_matrix(y_test,predict4))
print('\n')
print(classification_report(y_test,predict4))
```

```
[[2333   83]
 [ 296  288]]


              precision    recall  f1-score   support

           0       0.89      0.97      0.92      2416
           1       0.78      0.49      0.60       584

    accuracy                           0.87      3000
   macro avg       0.83      0.73      0.76      3000
weighted avg       0.87      0.87      0.86      3000
```

MODEL FIVE: KNEIGHBORSCLASSIFIER MODEL(KNN)

In [77]:
```
#importing,fitting and testing the model
```

In [78]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [79]:
```
knn=KNeighborsClassifier(n_neighbors=10)
```

In [80]:
```
knn.fit(X_train,y_train)
```

Out[80]:
```
▾         KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

In [81]:
```
predict5=knn.predict(X_test)
```

In [82]:
```
print(confusion_matrix(y_test,predict5))
print('\n')
print(classification_report(y_test,predict5))
```

```
[[2392   24]
 [ 573   11]]


              precision    recall  f1-score   support

           0       0.81      0.99      0.89      2416
           1       0.31      0.02      0.04       584

    accuracy                           0.80      3000
   macro avg       0.56      0.50      0.46      3000
weighted avg       0.71      0.80      0.72      3000
```

In [83]:
```
# experimenting with different k_values
```

In [84]:
```
from sklearn.metrics import accuracy_score
```

In [85]:
```
k_range = list(range(1,26))
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
print(y_pred)
print(accuracy_score(y_test, y_pred))
```

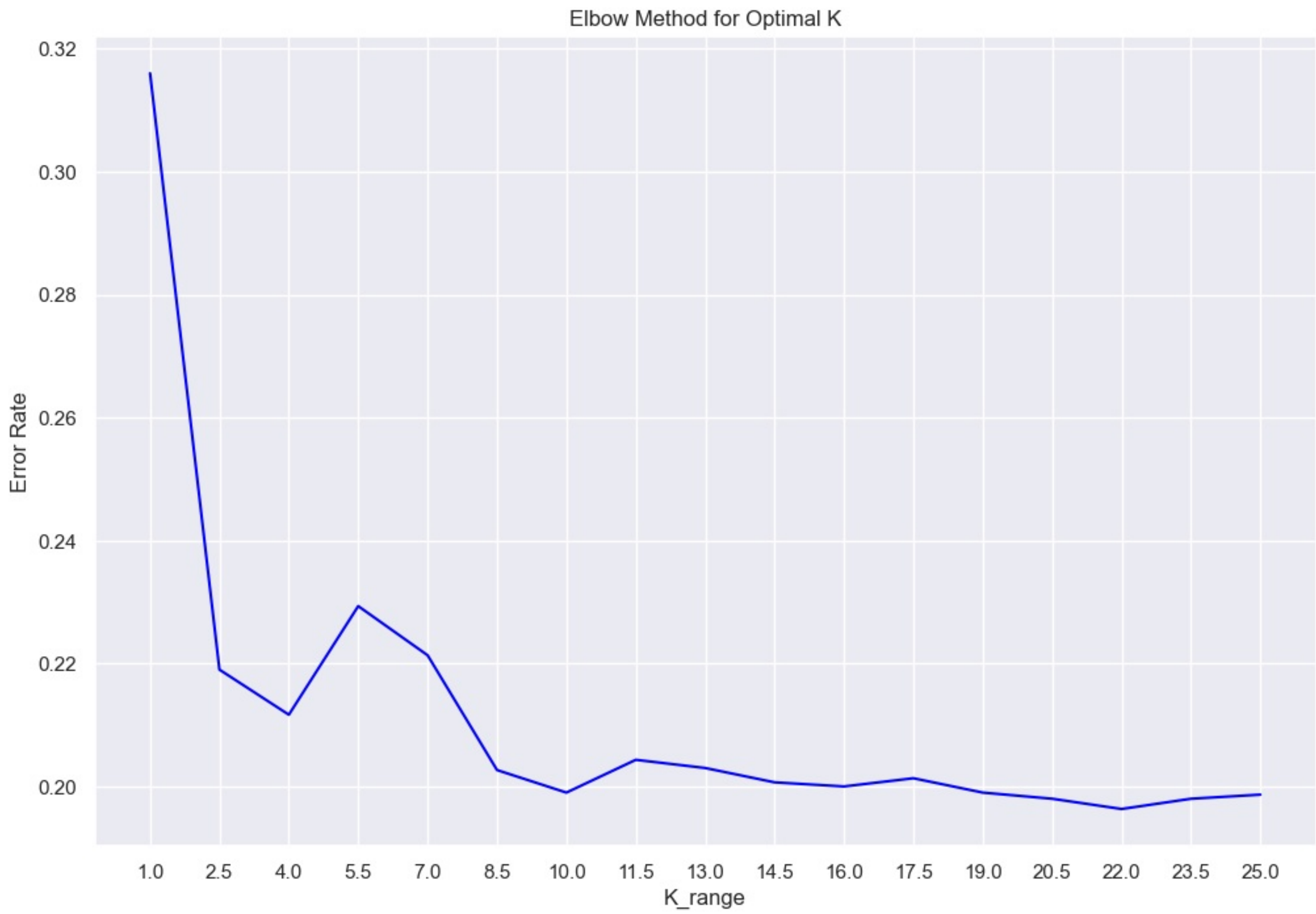
```
[0 0 0 ... 0 0 0]
0.8013333333333333
```

In [86]:
```
k_range = list(np.arange(1,26,1.5))
scores = []
error_rates = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=int(k))
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)
    scores.append(accuracy_score(y_test, y_pred))
    error = 1 - knn.score(X_test, y_test)
    error_rates.append(error)
```

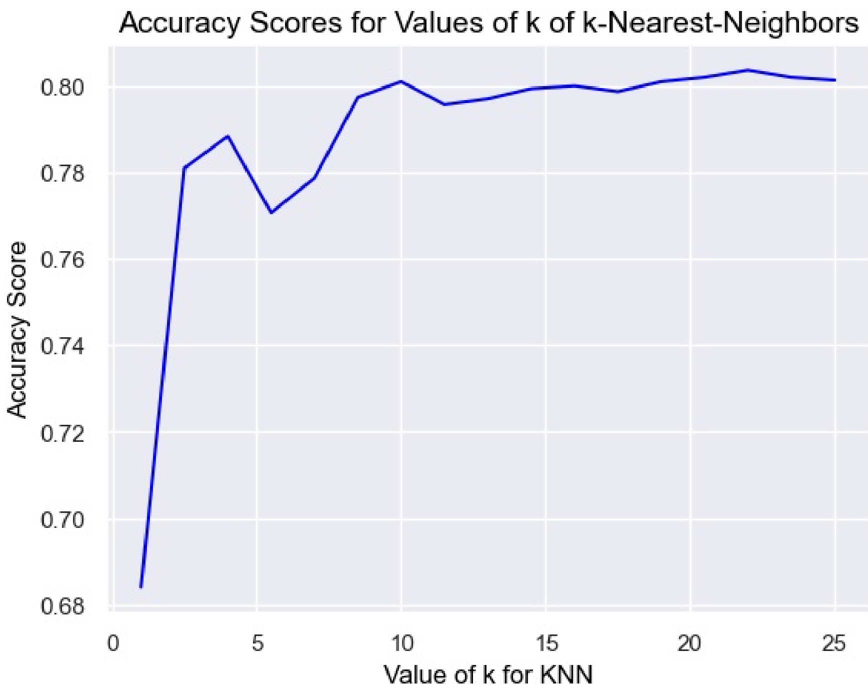
In [87]:
```
print(y_pred)
print(accuracy_score(y_test, y_pred))
```

```
[0 0 0 ... 0 0 0]
0.8013333333333333
```

In [88]:
```
#A graph of Elbow Method for Optimal K
plt.figure(figsize=(12,8))
plt.plot(k_range, error_rates, linestyle='-',color='blue')
plt.title('Elbow Method for Optimal K')
plt.xlabel('K_range')
plt.ylabel('Error Rate')
plt.xticks(k_range)
plt.grid(True)
plt.show()
```



In [89]:
```
#plot on Accuracy Scores for Values of k of k-Nearest-Neighbors
plt.plot(k_range,scores,color='blue')
plt.xlabel('Value of k for KNN',color='black')
plt.ylabel('Accuracy Score',color='black')
plt.title('Accuracy Scores for Values of k of k-Nearest-Neighbors',color='black',fontsize='14')
plt.show()
```



MODEL SIX :GRADIENT BOOSTING CLASSIFIER MODEL(GBC)

In [90]:
```
#importing,fitting and testing the model
```

In [91]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [92]:
```
Gbc=GradientBoostingClassifier()
```

In [93]:
```
Gbc.fit(X_train,y_train)
```

Out[93]:

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [94]:
```
predict6=Gbc.predict(X_test)
```

In [95]:
```
print(confusion_matrix(y_test,predict6))
print('\n')
print(classification_report(y_test,predict6))
```

```
[[2342   74]
 [ 316  268]]


              precision    recall  f1-score   support

           0       0.88      0.97      0.92      2416
           1       0.78      0.46      0.58       584

    accuracy                           0.87      3000
   macro avg       0.83      0.71      0.75      3000
weighted avg       0.86      0.87      0.86      3000
```

MODEL SEVEN :GAUSSIAN NAIVE BAYES MODEL(GNB)

In [96]:
```
#importing,fitting and testing the model
```

In [97]:
```
from sklearn.naive_bayes import GaussianNB
```

In [98]:
```
gnb= GaussianNB()
```

In [99]:
```
gnb.fit(X_train,y_train)
```

Out[99]:

```
▾ GaussianNB
GaussianNB()
```

In [100…
```
predict7=gnb.predict(X_test)
```

In [101…
```
print(confusion_matrix(y_test,predict7))
print('\n')
print(classification_report(y_test,predict7))
```

```
[[2332   84]
 [ 540   44]]


              precision    recall  f1-score   support

           0       0.81      0.97      0.88      2416
           1       0.34      0.08      0.12       584

    accuracy                           0.79      3000
   macro avg       0.58      0.52      0.50      3000
weighted avg       0.72      0.79      0.73      3000
```

MODEL EIGHT : XGBOOSTCLASSIFIER MODEL(XGB)

In [102…
```
#importing,fitting and testing the model
```

In [103…
```
import xgboost as xgb
```

In [104…
```
xgb_classifier = xgb.XGBClassifier(n_estimators=100,objective='binary:logistic', tree_method='hist', eta=0.1, m
```

In [105…
```
xgb_classifier.fit(X_train,y_train)
```

Out[105]:

```
XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=True, eta=0.1, eval_metric=None,
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=3,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=10
0,
```

In [106…

```
predict8=xgb_classifier.predict(X_test)
```

In [107…

```
print(confusion_matrix(y_test,predict8))
print('\n')
print(classification_report(y_test,predict8))
```

```
[[2347   69]
 [ 316  268]]


              precision    recall  f1-score   support

           0       0.88      0.97      0.92      2416
           1       0.80      0.46      0.58       584

    accuracy                           0.87      3000
   macro avg       0.84      0.72      0.75      3000
weighted avg       0.86      0.87      0.86      3000
```

MODEL NINE: ARTIFICIAL NEURAL NETWORK MODEL(ANN)

In [108…

```
#importing,fitting and testing the model
```

In [109…

```
from tensorflow.keras.models import Sequential
```

In [110…

```
from tensorflow.keras.layers import Dense,Dropout
```

In [111…

```
model=Sequential()
model.add(Dense(11,activation ='relu'))
model.add(Dense(10,activation ='relu'))
model.add(Dense(10,activation ='relu'))
model.add(Dense(6,activation ='relu'))
model.add(Dense(6,activation ='relu'))
model.add(Dense(units=1,activation ='sigmoid'))
```

In [112…

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

In [113…

```
from tensorflow.keras.callbacks import EarlyStopping
```

In [114…

```
early_stop=EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=25)
```

In [115…

```
model.fit(x=X_train,y=y_train,epochs=20,batch_size=50,validation_data=(X_test,y_test),callbacks=[early_stop])
```

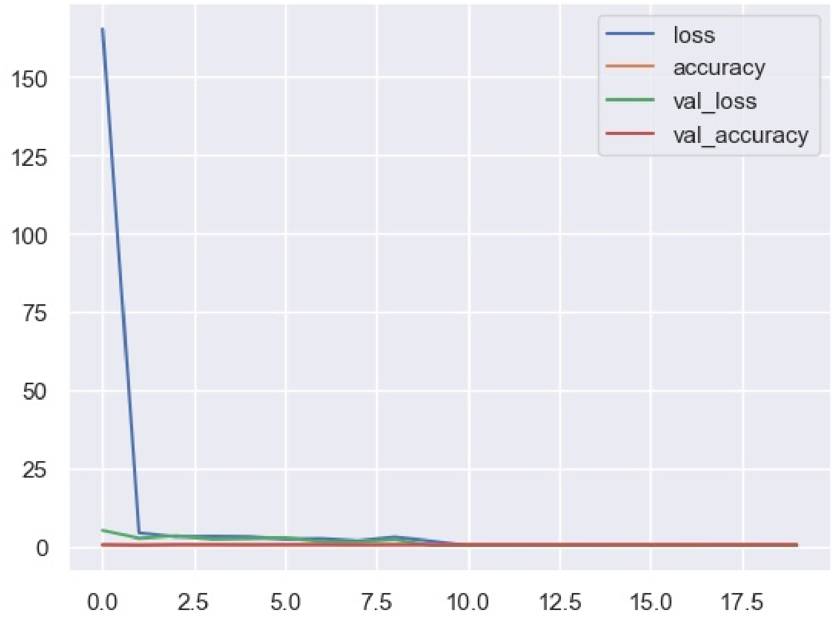
```
Epoch 1/20
140/140 [==============================] - 2s 5ms/step - loss: 165.3694 - accuracy: 0.7157 - val_loss: 5.3013 -
val_accuracy: 0.7960
Epoch 2/20
140/140 [==============================] - 0s 2ms/step - loss: 4.5088 - accuracy: 0.7193 - val_loss: 2.7940 - v
al_accuracy: 0.6470
Epoch 3/20
140/140 [==============================] - 0s 2ms/step - loss: 3.2714 - accuracy: 0.7370 - val_loss: 3.5763 - v
al_accuracy: 0.8027
Epoch 4/20
140/140 [==============================] - 0s 2ms/step - loss: 3.4144 - accuracy: 0.7384 - val_loss: 2.5334 - v
al_accuracy: 0.8053
Epoch 5/20
140/140 [==============================] - 0s 2ms/step - loss: 3.2955 - accuracy: 0.7477 - val_loss: 2.7008 - v
al_accuracy: 0.7820
Epoch 6/20
140/140 [==============================] - 1s 4ms/step - loss: 2.5536 - accuracy: 0.7481 - val_loss: 2.9682 - v
al_accuracy: 0.8053
Epoch 7/20
140/140 [==============================] - 0s 3ms/step - loss: 2.6487 - accuracy: 0.7669 - val_loss: 1.7465 - v
al_accuracy: 0.8053
Epoch 8/20
140/140 [==============================] - 0s 2ms/step - loss: 2.0185 - accuracy: 0.7333 - val_loss: 1.6572 - v
al_accuracy: 0.7657
Epoch 9/20
140/140 [==============================] - 0s 2ms/step - loss: 3.1581 - accuracy: 0.7487 - val_loss: 2.4355 - v
al_accuracy: 0.8053
Epoch 10/20
140/140 [==============================] - 0s 2ms/step - loss: 1.9456 - accuracy: 0.7677 - val_loss: 0.5273 - v
al_accuracy: 0.8053
Epoch 11/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5646 - accuracy: 0.7877 - val_loss: 0.5384 - v
al_accuracy: 0.8053
Epoch 12/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5752 - accuracy: 0.7863 - val_loss: 0.5303 - v
al_accuracy: 0.8053
Epoch 13/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5645 - accuracy: 0.7810 - val_loss: 0.5185 - v
al_accuracy: 0.8053
Epoch 14/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5726 - accuracy: 0.7746 - val_loss: 0.5302 - v
al_accuracy: 0.8053
Epoch 15/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5661 - accuracy: 0.7789 - val_loss: 0.5118 - v
al_accuracy: 0.8053
Epoch 16/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5436 - accuracy: 0.7901 - val_loss: 0.5087 - v
al_accuracy: 0.8053
Epoch 17/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5352 - accuracy: 0.7923 - val_loss: 0.5075 - v
al_accuracy: 0.8053
Epoch 18/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5288 - accuracy: 0.7924 - val_loss: 0.5276 - v
al_accuracy: 0.8053
Epoch 19/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5427 - accuracy: 0.7901 - val_loss: 0.5531 - v
al_accuracy: 0.8053
Epoch 20/20
140/140 [==============================] - 0s 2ms/step - loss: 0.5307 - accuracy: 0.7899 - val_loss: 0.5069 - v
al_accuracy: 0.8053
```

Out[115]:
```
<keras.src.callbacks.History at 0x22e2818b350>
```

In [116…
```
losses=pd.DataFrame(model.history.history)
```

In [117…
```
#graph on model.history
losses.plot()
```

Out[117]:
```
<Axes: >
```



In [118…
```
predict9=(model.predict(X_test)>0.5).astype('int32')
```

```
94/94 [==============================] - 0s 1ms/step
```

In [119…
```
predict9
```

Out[119]:
```
array([[0],
       [0],
       [0],
       ...,
       [0],
       [0],
       [0]])
```

In [120…
```
print(confusion_matrix(y_test,predict9))
print('\n')
print(classification_report(y_test,predict9))
```

```
[[2416    0]
 [ 584    0]]


              precision    recall  f1-score   support

           0       0.81      1.00      0.89      2416
           1       0.00      0.00      0.00       584

    accuracy                           0.81      3000
   macro avg       0.40      0.50      0.45      3000
weighted avg       0.65      0.81      0.72      3000
```

MODEL EVALUATION

In [121…
```
#comparing accuracy score for all the models
```

In [122…
```
from sklearn.metrics import accuracy_score
```

In [123…
```
Accuracy_scores=pd.DataFrame({'Models':['LR','SVC','RF','CBC','KNN','GBC','GNB','XGB','ANN'],
                        'ACCURACY':[accuracy_score(y_test,predict1),
                        accuracy_score(y_test,predict2),
                        accuracy_score(y_test,predict3),
```

```
                                accuracy_score(y_test,predict4),
                                accuracy_score(y_test,predict5),
                                accuracy_score(y_test,predict6),
                                accuracy_score(y_test,predict7),
                                accuracy_score(y_test,predict8),
                                accuracy_score(y_test,predict9)]})
```

In [124…

```
Accuracy_scores
```

Out[124]:

	Models	ACCURACY
0	LR	0.800667
1	SVC	0.805333
2	RF	0.868333
3	CBC	0.873667
4	KNN	0.801000
5	GBC	0.870000
6	GNB	0.792000
7	XGB	0.871667
8	ANN	0.805333

CONCLUSION

Amongst 9 models which were trained,Cat boost classifier had the highest accuracy(87.37%).Followed closely bt XGBoost Classifier(87.17%),Gradient boosting Classief(87%),Random Forest(86.93%)
Other models had the accuracy around 80%, these models are Logistic Regression,Support Vector Machine,Artificial Neural Network,K-Nearest Neighbours and Gaussian Naive Bayes.
Hence Cat Boost classifier can be used for model deployment