In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```python
df = pd.read_csv('Churn_Modelling.csv')
df.head()
```

Out[2]:

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

In [3]:
```python
df.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 [4]:
```
df.describe(include='all')
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age
count	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.000000
unique	NaN	NaN	2932	NaN	3	2	NaN
top	NaN	NaN	Smith	NaN	France	Male	NaN
freq	NaN	NaN	32	NaN	5014	5457	NaN
mean	5000.50000	1.569094e+07	NaN	650.528800	NaN	NaN	38.921800
std	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.487806
min	1.00000	1.556570e+07	NaN	350.000000	NaN	NaN	18.000000
25%	2500.75000	1.562853e+07	NaN	584.000000	NaN	NaN	32.000000
50%	5000.50000	1.569074e+07	NaN	652.000000	NaN	NaN	37.000000
75%	7500.25000	1.575323e+07	NaN	718.000000	NaN	NaN	44.000000
max	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
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
```

Dropping Irrelevant Features

In [6]:
```
df.columns
```

Out[6]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [7]:
```python
df = df.drop(['RowNumber', 'CustomerId', 'Surname'],axis=1)
```

In [8]:
```python
df.head()
```

Out[8]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActi
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	

Encoding Categorical Data

In [9]:
```python
df = pd.get_dummies(df,drop_first = True)
```

In [10]:
```python
df.head()
```

Out[10]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estimated
0	619	42	2	0.00	1	1	1	1013
1	608	41	1	83807.86	1	0	1	1125
2	502	42	8	159660.80	3	1	0	1139
3	699	39	1	0.00	2	0	0	938
4	850	43	2	125510.82	1	1	1	790

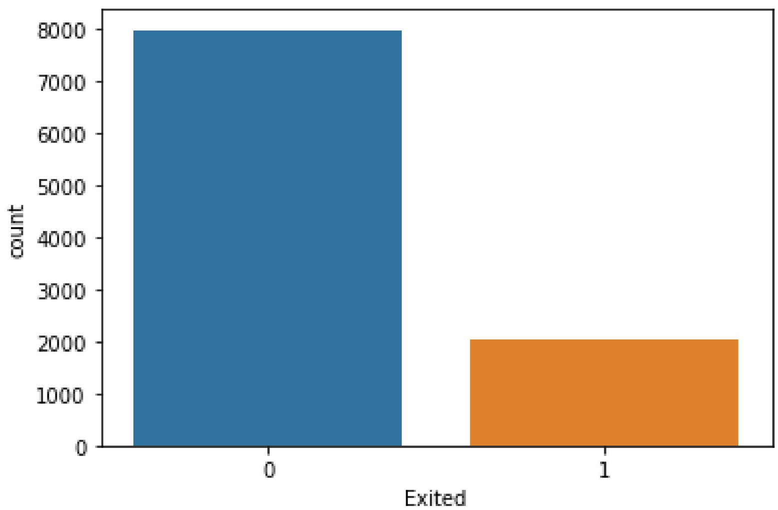
Some insights about the target variable

In [11]:
```python
df['Exited'].value_counts()
```

Out[11]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [12]:
```python
sns.countplot(df['Exited'])
```

Out[12]: <AxesSubplot:xlabel='Exited', ylabel='count'>



In [13]:
```python
X = df.drop('Exited',axis=1)
y = df['Exited']
```

Handling Imbalanced Data with SMOTE

In [14]:
```python
from imblearn.over_sampling import SMOTE
```

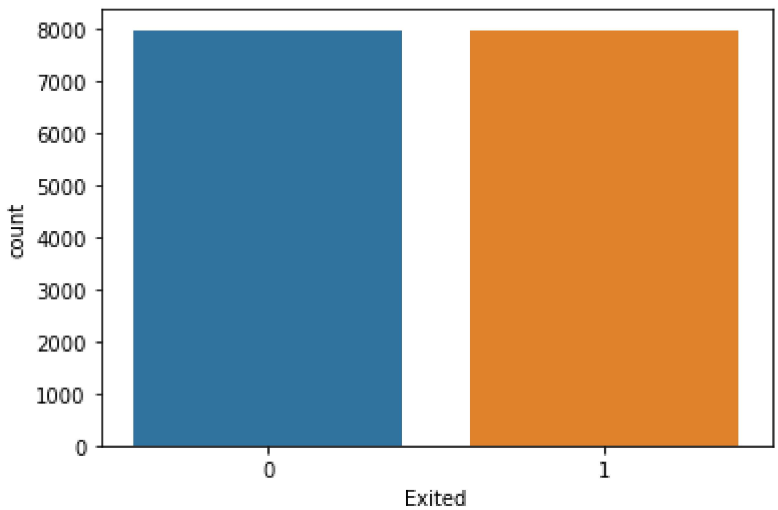
In [15]:
```python
X_res, y_res = SMOTE().fit_resample(X,y)
```

In [16]:
```python
y_res.value_counts()
```

Out[16]:
```
1    7963
0    7963
Name: Exited, dtype: int64
```

In [17]:
```
sns.countplot(y_res)
```

Out[17]: <AxesSubplot:xlabel='Exited', ylabel='count'>



Splitting The Dataset into Training Set and Test Set

In [18]:
```
from sklearn.model_selection import train_test_split
```

In [19]:
```
X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.
```

Feature Scaling

In [20]:
```
from sklearn.preprocessing import StandardScaler
```

In [21]:
```
sc = StandardScaler()
```

In [22]:
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [23]:
```
X_train
```

Out[23]:
```
array([[ 0.75877907, -0.8982394 ,  0.78677282, ..., -0.575693  ,
         2.16445251, -0.84390722],
       [-0.68402381, -0.7976025 ,  1.15294588, ..., -0.575693  ,
        -0.4620106 ,  1.18496439],
       [-1.32891298,  0.2087665 ,  0.78677282, ..., -0.575693  ,
         2.16445251,  1.18496439],
       ...,
       [-0.00634367,  0.0074927 ,  0.42059976, ..., -0.575693  ,
        -0.4620106 , -0.84390722],
       [ 0.47459063, -0.9988763 , -1.41026556, ..., -0.575693  ,
        -0.4620106 ,  1.18496439],
       [ 1.63320506,  2.22150449, -1.41026556, ..., -0.575693  ,
        -0.4620106 , -0.84390722]])
```

Logistic Regression

In [24]:
```
from sklearn.linear_model import LogisticRegression
```

In [25]:
```
log = LogisticRegression()
```

In [26]:
```
log.fit(X_train, y_train)
```

Out[26]:
```
LogisticRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [27]:
```
y_pred1 = log.predict(X_test)
```

In [28]:
```
from sklearn.metrics import accuracy_score
```

In [29]:
```
accuracy_score(y_test,y_pred1)
```

Out[29]:
```
0.7748011720385098
```

In [30]:
```
from sklearn.metrics import precision_score, recall_score, f1_score
```

In [31]:
```
precision_score(y_test, y_pred1)
```

Out[31]:
```
0.7582957804178615
```

In [32]:
```
recall_score(y_test, y_pred1)
```

Out[32]:
```
0.79204107830552
```

In [33]:
```
f1_score(y_test,y_pred1)
```
Out[33]: 0.7748011720385098

SVC

In [34]:
```
from sklearn import svm
```
In [35]:
```
svm = svm.SVC()
```
In [36]:
```
svm.fit(X_train,y_train)
```
Out[36]: SVC()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [37]:
```
y_pred2=svm.predict(X_test)
```
In [38]:
```
accuracy_score(y_test, y_pred2)
```
Out[38]: 0.8287986605274174

In [39]:
```
precision_score(y_test, y_pred2)
```
Out[39]: 0.8214134574693187

In [40]:
```
recall_score(y_test, y_pred2)
```
Out[40]: 0.8305519897304237

In [41]:
```
f1_score(y_test,y_pred2)
```
Out[41]: 0.8259574468085107

KNeighbors Classifier

In [42]:
```
from sklearn.neighbors import KNeighborsClassifier
```
In [43]:
```
knn = KNeighborsClassifier()
```

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

Out[44]:
```
KNeighborsClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [45]:
```
y_pred3=knn.predict(X_test)
```

In [46]:
```
accuracy_score(y_test, y_pred3)
```

Out[46]:
```
0.8080786940142319
```

In [47]:
```
precision_score(y_test, y_pred3)
```

Out[47]:
```
0.785829307568438
```

In [48]:
```
recall_score(y_test, y_pred3)
```

Out[48]:
```
0.8352588789045785
```

In [49]:
```
f1_score(y_test,y_pred3)
```

Out[49]:
```
0.8097904998962872
```

Desicion Tree Classifier

In [50]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [51]:
```
dt = DecisionTreeClassifier()
```

In [52]:
```
dt.fit(X_train, y_train)
```

Out[52]:
```
DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [53]:
```
y_pred4=dt.predict(X_test)
```

In [54]:
```
accuracy_score(y_test, y_pred4)
```

Out[54]:
```
0.7986605274173294
```

In [55]:
```
precision_score(y_test, y_pred4)
```
Out[55]: 0.7804977560179519

In [56]:
```
recall_score(y_test, y_pred4)
```
Out[56]: 0.8185708172871202

In [57]:
```
f1_score(y_test,y_pred4)
```
Out[57]: 0.7990810359231411

Random Forest Classifier

In [58]:
```
from sklearn.ensemble import RandomForestClassifier
```
In [59]:
```
rf = RandomForestClassifier()
```
In [60]:
```
rf.fit(X_train,y_train)
```
Out[60]: RandomForestClassifier()

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [61]:
```
y_pred5=rf.predict(X_test)
```
In [62]:
```
accuracy_score(y_test,y_pred5)
```
Out[62]: 0.8503557974047719

In [63]:
```
precision_score(y_test, y_pred5)
```
Out[63]: 0.8407563025210084

In [64]:
```
recall_score(y_test, y_pred5)
```
Out[64]: 0.8562259306803595

In [65]:
```
f1_score(y_test,y_pred5)
```
Out[65]: 0.8484206063175749

Gradient Boosting Classifier

In [66]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

In [67]:
```
gbc = GradientBoostingClassifier()
```

In [68]:
```
gbc.fit(X_train, y_train)
```

Out[68]:
```
GradientBoostingClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [69]:
```
y_pred6=gbc.predict(X_test)
```

In [70]:
```
accuracy_score(y_test,y_pred6)
```

Out[70]:
```
0.8340309753034743
```

In [71]:
```
precision_score(y_test,y_pred6)
```

Out[71]:
```
0.8307626392459297
```

In [72]:
```
recall_score(y_test,y_pred6)
```

Out[72]:
```
0.8296961916987591
```

In [73]:
```
f1_score(y_test,y_pred6)
```

Out[73]:
```
0.8302290730036395
```

XGBoost

In [86]:
```
import xgboost as xgb

model_xgb = xgb.XGBClassifier(random_state=42, verbosity = 0)
model_xgb.fit(X_train, y_train)
```

Out[86]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=Fals
e,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [87]:
```
y_pred7=model_xgb.predict(X_test)
```

In [88]:
```
accuracy_score(y_test,y_pred7)
```

Out[88]: 0.8522394307241523

In [89]:
```
precision_score(y_test,y_pred7)
```

Out[89]: 0.8382413936126089

In [90]:
```
recall_score(y_test,y_pred6)
```

Out[90]: 0.8296961916987591

In [91]:
```
f1_score(y_test,y_pred6)
```

Out[91]: 0.8302290730036395

Accuracy Summary

In [92]:
```
performance_summary = pd.DataFrame({
    'Model':['LR','SVC','KNN','DT','RF','GBC','XGB'],
    'ACC':[accuracy_score(y_test,y_pred1),
           accuracy_score(y_test,y_pred2),
           accuracy_score(y_test,y_pred3),
           accuracy_score(y_test,y_pred4),
           accuracy_score(y_test,y_pred5),
           accuracy_score(y_test,y_pred6),
           accuracy_score(y_test,y_pred7)
          ]
})
```

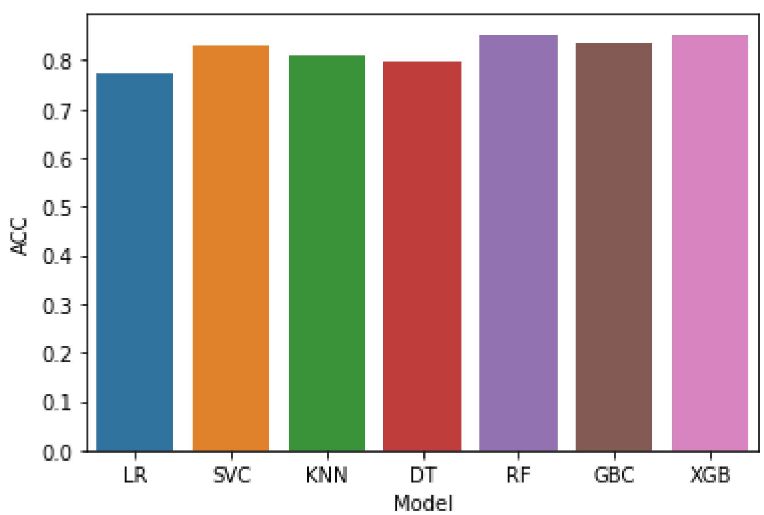
In [93]:
```
performance_summary
```

Out[93]:

	Model	ACC
0	LR	0.774801
1	SVC	0.828799
2	KNN	0.808079
3	DT	0.798661
4	RF	0.850356
5	GBC	0.834031
6	XGB	0.852239

In [94]:
```
sns.barplot(performance_summary['Model'],performance_summary['ACC'])
```

Out[94]: <AxesSubplot:xlabel='Model', ylabel='ACC'>



As we can see, XGBoost Classifier has highest accuracy

In [95]:
```
performance_summary = pd.DataFrame({
    'Model':['LR','SVC','KNN','DT','RF','GBC','XGB'],
    'PRECISION':[precision_score(y_test,y_pred1),
           precision_score(y_test,y_pred2),
           precision_score(y_test,y_pred3),
           precision_score(y_test,y_pred4),
           precision_score(y_test,y_pred5),
           precision_score(y_test,y_pred6),
            precision_score(y_test,y_pred6)
          ]
})
```

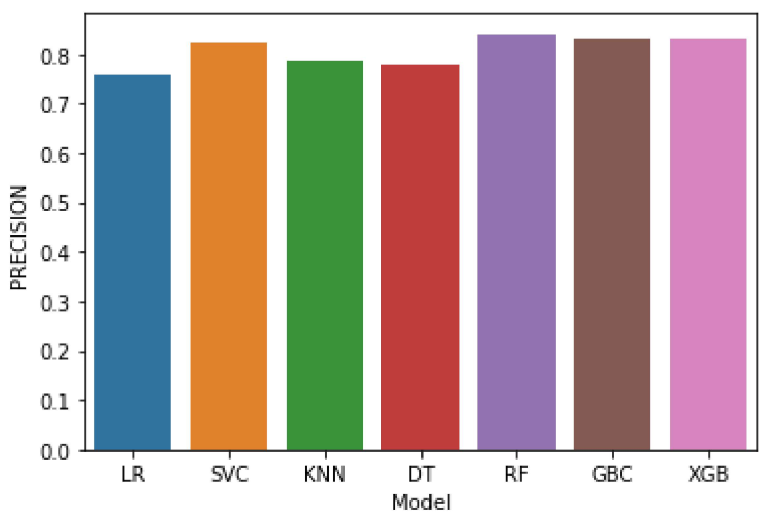
In [96]:
```
performance_summary
```

Out[96]:

	Model	PRECISION
0	LR	0.758296
1	SVC	0.821413
2	KNN	0.785829
3	DT	0.780498
4	RF	0.840756
5	GBC	0.830763
6	XGB	0.830763

In [97]:
```
sns.barplot(performance_summary['Model'],performance_summary['PRECISION'])
```

Out[97]: <AxesSubplot:xlabel='Model', ylabel='PRECISION'>



Saving the best model, XGBoost

In [80]:
```
X_train = sc.fit_transform(X_train)
```

In [98]:
```
model_xgb.fit(X_res,y_res)
```

Out[98]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=Fals
e,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=8,
              num_parallel_tree=1, predictor='auto', random_state=42,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:
```
import joblib
```

In [100]:
```
joblib.dump(model_xgb, 'churn_predict_model')
```

Out[100]:
```
['churn_predict_model']
```

In [101]:
```
model = joblib.load('churn_predict_model')
```

In [85]:
```
df.columns
```

Out[85]:
```
Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCar
d',
       'IsActiveMember', 'EstimatedSalary', 'Exited', 'Geography_Germany',
       'Geography_Spain', 'Gender_Male'],
      dtype='object')
```