In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
get_ipython().run_line_magic('matplotlib', 'inline')
```

1. Loading Dataset Into Pandas Dataframe

In [3]:
```
df = pd.read_csv(r'C:\Users\falco\OneDrive\Desktop\PROJECTS\PYTHON PROJECTS\MY PROJECT
```

2. Display Top 5 Rows()

In [4]:
```
df.head()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Nu
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	

3. Last 5 Rows

In [5]:
```
df.tail()
```

Out[5]:

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

4. Shape Of Dataset

In [8]:
```
df.shape
```

```
print('Number of Rows: ', df.shape[0])
print('Number of Columns: ', df.shape[1])
```

```
Number of Rows:  10000
Number of Columns:  14
```

In [ ]:
```
# 5. Overall Info About The Dataset
```

In [9]:
```
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 [ ]:
```
# 6. Null Values
```

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
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
```

7. Overall Statistics About The Dataset

In [11]:
```
df.describe()
```

Out[11]:

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfP
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4

In [12]:
```
df.describe(include = 'all')
```

Out[12]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	
count	10000.00000	1.000000e+04	10000	10000.000000	10000	10000	10000.000000	10000
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	5
std	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.487806	2
min	1.00000	1.556570e+07	NaN	350.000000	NaN	NaN	18.000000	0
25%	2500.75000	1.562853e+07	NaN	584.000000	NaN	NaN	32.000000	3
50%	5000.50000	1.569074e+07	NaN	652.000000	NaN	NaN	37.000000	5
75%	7500.25000	1.575323e+07	NaN	718.000000	NaN	NaN	44.000000	7
max	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000	10

8. Dropping Irrelevant Columns

In [13]:
```
df.columns
```

Out[13]:
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [15]:
```
df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis = 1)
```

In [16]:
```
df.head()
```

Out[16]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
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	

9. Categorical Data Encoding

In [17]:
```
df['Geography'].unique()
```

Out[17]:
```
array(['France', 'Spain', 'Germany'], dtype=object)
```

In [18]:
```
df = pd.get_dummies(df, drop_first = True)
```

In [19]:
```
df.head()
```

Out[19]:

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

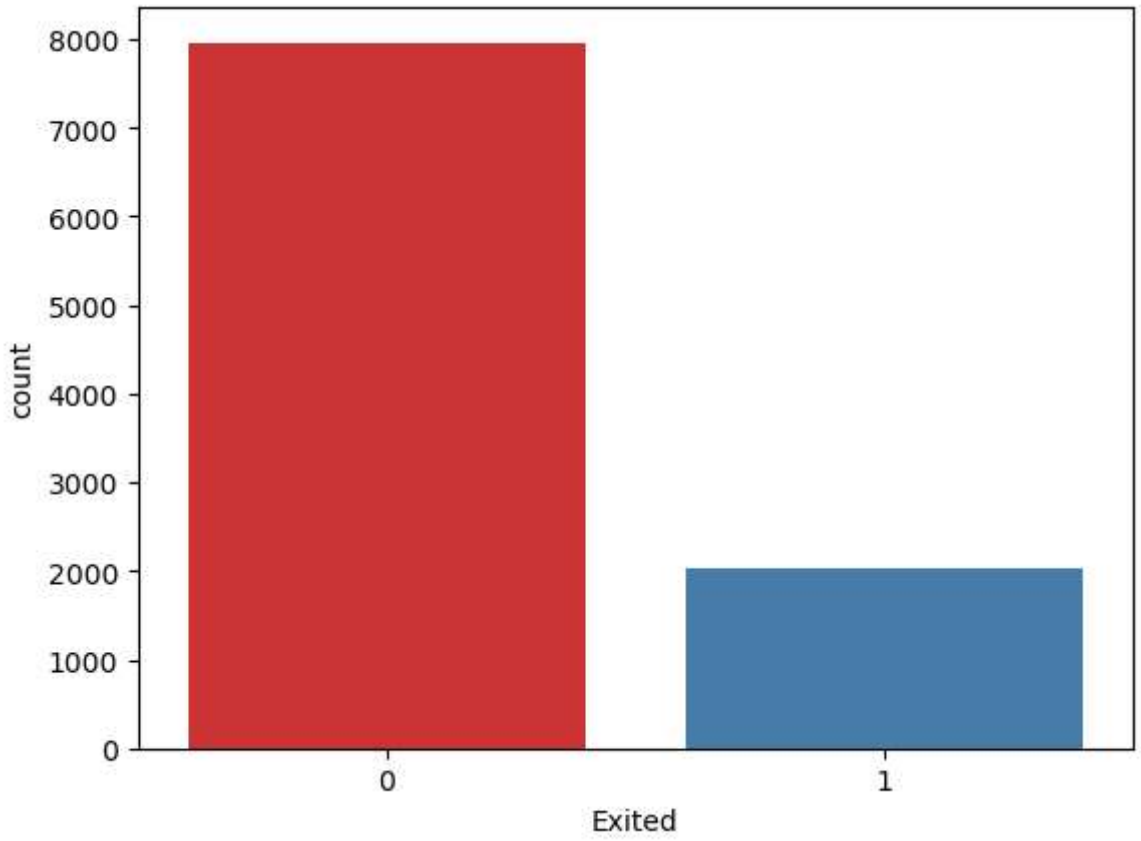
10. Checking Imbalanced Data and Visualising

In [20]:
```
df['Exited'].value_counts()
```

Out[20]:
```
Exited
0    7963
1    2037
Name: count, dtype: int64
```

In [24]:
```
sns.countplot(x = 'Exited', data = df, palette = 'Set1')
```

Out[24]:
```
<Axes: xlabel='Exited', ylabel='count'>
```



11. Assigning X and y Values

In [26]:
```
X = df.drop('Exited', axis = 1)
y = df['Exited']
```

In [27]:
```
X
```

Out[27]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	619	42	2	0.00	1	1	1	10134
1	608	41	1	83807.86	1	0	1	11254
2	502	42	8	159660.80	3	1	0	11393
3	699	39	1	0.00	2	0	0	9382
4	850	43	2	125510.82	1	1	1	7908
...	...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	0	9627
9996	516	35	10	57369.61	1	1	1	10169
9997	709	36	7	0.00	1	0	1	4208
9998	772	42	3	75075.31	2	1	0	9288
9999	792	28	4	130142.79	1	1	0	3819

10000 rows × 11 columns

In [28]:
```
y
```

Out[28]:
```
0       1
1       0
2       1
3       0
4       0
       ..
9995    0
9996    0
9997    1
9998    1
9999    0
Name: Exited, Length: 10000, dtype: int64
```

12. Balancing The Data With SMOTE

In [29]:
```
from imblearn.over_sampling import SMOTE
```

In [30]:
```
Xs, ys = SMOTE().fit_resample(X,y)
```

In [32]:
```
ys.value_counts()
```

Out[32]:
```
Exited
1    7963
0    7963
Name: count, dtype: int64
```

13. Splitting The Data Into Training And Testing

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

X_train, X_test, y_train, y_test = train_test_split(Xs, ys, test_size = 0.2)
```

14. Converting Categorical Values Into Numerical Values Using The StandardScaler

In [34]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
```

In [37]:
```
X_train
```

Out[37]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
2482	567	40	4	118628.800000	1	0	0	919
12074	601	34	1	102978.844759	1	0	0	1796
9705	733	36	1	0.000000	2	0	1	1083
7505	659	27	4	0.000000	2	1	0	993
8510	550	40	8	150490.320000	1	0	0	1664
...	...	...	...	...	...	...	...	
4813	517	39	3	0.000000	2	0	1	124
1433	755	62	1	127706.330000	2	0	1	1423
6117	850	37	2	0.000000	2	1	0	1199
13060	548	53	6	0.000000	1	0	1	735
1619	656	18	10	151762.740000	1	0	1	1270

12740 rows × 11 columns

In [38]:
```
X_test
```

Out[38]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
10333	731	41	6	106110.748773	1	1	0	1948
11542	631	40	2	107841.715476	1	1	0	525
15585	429	51	4	91828.239843	1	1	0	1440
8957	585	37	6	152496.820000	1	1	1	999
334	626	37	6	108269.370000	1	1	0	55
...	...	...	...	...	...	...	...	
12563	748	58	5	146941.760593	1	1	0	418
2450	589	46	10	107238.850000	2	1	0	370
7712	545	30	3	0.000000	2	1	0	1703
15619	624	35	1	83243.639912	2	0	1	727
5153	695	29	9	0.000000	2	1	0	1115

3186 rows × 11 columns

In [39]:
```
y_train
```

Out[39]:
```
2482     0
12074    1
9705     0
7505     0
8510     1
        ..
4813     0
1433     0
6117     0
13060    1
1619     0
Name: Exited, Length: 12740, dtype: int64
```

In [40]:
```
y_test
```

Out[40]:
```
10333    1
11542    1
15585    1
8957     0
334      0
        ..
12563    1
2450     0
7712     0
15619    1
5153     0
Name: Exited, Length: 3186, dtype: int64
```

In [41]:
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [42]:
```
X_train
```

Out[42]:
```
array([[-0.88794153, -0.09917398, -0.31405339, ..., -0.75481043,
         1.58571207,  0.81529544],
       [-0.51755224, -0.70035232, -1.41921135, ...,  1.32483596,
        -0.63063151, -1.22654924],
       [ 0.92042969, -0.49995954, -1.41921135, ..., -0.75481043,
         1.58571207,  0.81529544],
       ...,
       [ 2.19500458, -0.39976315, -1.05082536, ..., -0.75481043,
         1.58571207,  0.81529544],
       [-1.09492378,  1.2033791 ,  0.42271858, ..., -0.75481043,
         1.58571207,  0.81529544],
       [ 0.08160689, -2.30349457,  1.89626253, ..., -0.75481043,
        -0.63063151,  0.81529544]])
```

In [43]:
```
X_test
```

Out[43]:
```
array([[ 8.98642084e-01,  1.02241215e-03,  4.22718582e-01, ...,
         1.32483596e+00, -6.30631513e-01,  8.15295438e-01],
       [-1.90738169e-01, -9.91739783e-02, -1.05082536e+00, ...,
         1.32483596e+00, -6.30631513e-01,  8.15295438e-01],
       [-2.39128628e+00,  1.00298632e+00, -3.14053391e-01, ...,
        -7.54810432e-01,  1.58571207e+00,  8.15295438e-01],
       ...,
       [-1.12760519e+00, -1.10113788e+00, -6.82439377e-01, ...,
        -7.54810432e-01, -6.30631513e-01,  8.15295438e-01],
       [-2.66994787e-01, -6.00155931e-01, -1.41921135e+00, ...,
        -7.54810432e-01,  1.58571207e+00,  8.15295438e-01],
       [ 5.06465193e-01, -1.20133427e+00,  1.52787654e+00, ...,
        -7.54810432e-01, -6.30631513e-01, -1.22654924e+00]])
```

15. Applying Logistic Rgeression And Checking Accuracy, Precision Recall and F1 Scores

In [44]:
```
from sklearn.linear_model import LogisticRegression

lreg = LogisticRegression()
lreg.fit(X_train, y_train)
```

Out[44]:
```
▾ LogisticRegression

LogisticRegression()
```

In [45]:
```
y_pred1 = lreg.predict(X_test)
```

In [48]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [59]:
```
print('Accuracy Score of SVC is:     ', accuracy_score(y_test, y_pred1))
print('Precision Score of SVC is:    ', precision_score(y_test, y_pred1))
print('Recall Score score of SVC is: ', recall_score(y_test, y_pred1))
print('F_1 Score of SVC is:          ', f1_score(y_test, y_pred1))
```

```
Accuracy Score of SVC is:      0.77338355304457
Precision Score of SVC is:     0.7681874229346486
Recall Score score of SVC is:  0.7826633165829145
F_1 Score of SVC is:           0.775357809583074
```

In [ ]:
```
# With acc_score - % corresponding, pc - tp/(tp+fp), rc - tp/(tp+fn)
```

16. Applyting SVC and Checking Score

In [60]:
```
from sklearn import svm

svm = svm.SVC()

svm.fit(X_train, y_train)
```

Out[60]:
```
SVC()
```

In [61]:
```
y_pred2 = svm.predict(X_test)
```

In [62]:
```
print('Accuracy Score of SVC is:      ', accuracy_score(y_test, y_pred2))
print('Precision Score of SVC is:     ', precision_score(y_test, y_pred2))
print('Recall Score score of SVC is:  ', recall_score(y_test, y_pred2))
print('F_1 Score of SVC is:           ', f1_score(y_test, y_pred2))
```

```
Accuracy Score of SVC is:      0.8380414312617702
Precision Score of SVC is:     0.8453145057766367
Recall Score score of SVC is:  0.8272613065326633
F_1 Score of SVC is:           0.8361904761904762
```

17. Applyting KNeighbors Classifier

In [64]:
```
from sklearn.neighbors import KNeighborsClassifier

KnClass = KNeighborsClassifier()
KnClass.fit(X_train, y_train)
```

Out[64]:
```
KNeighborsClassifier()
```

In [65]:
```
y_pred3 = KnClass.predict(X_test)
```

In [66]:
```
print('Accuracy Score of SVC is:      ', accuracy_score(y_test, y_pred3))
print('Precision Score of SVC is:     ', precision_score(y_test, y_pred3))
print('Recall Score score of SVC is:  ', recall_score(y_test, y_pred3))
print('F_1 Score of SVC is:           ', f1_score(y_test, y_pred3))
```

```
Accuracy Score of SVC is:      0.8195229127432517
Precision Score of SVC is:     0.8110091743119267
Recall Score score of SVC is:  0.8329145728643216
F_1 Score of SVC is:           0.8218159281066006
```

18. Applyting Decision Tree Classifier

In [67]:
```
from sklearn.tree import DecisionTreeClassifier

DTClass = DecisionTreeClassifier()
```

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

Out[68]:
```
▾ DecisionTreeClassifier

DecisionTreeClassifier()
```

In [70]:
```
y_pred4 = DTClass.predict(X_test)
```

In [71]:
```
print('Accuracy Score of SVC is:      ', accuracy_score(y_test, y_pred4))
print('Precision Score of SVC is:     ', precision_score(y_test, y_pred4))
print('Recall Score score of SVC is:  ', recall_score(y_test, y_pred4))
print('F_1 Score of SVC is:           ', f1_score(y_test, y_pred4))
```

```
Accuracy Score of SVC is:      0.8016321406151915
Precision Score of SVC is:     0.7955665024630542
Recall Score score of SVC is:  0.8115577889447236
F_1 Score of SVC is:           0.8034825870646767
```

19. Applyting Random Forest

In [72]:
```
from sklearn.ensemble import RandomForestClassifier

RFClass = RandomForestClassifier()
```

In [73]:
```
RFClass.fit(X_train, y_train)
```

Out[73]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [74]:
```
y_pred5 = RFClass.predict(X_test)
```

In [75]:
```
print('Accuracy Score of SVC is:      ', accuracy_score(y_test, y_pred5))
print('Precision Score of SVC is:     ', precision_score(y_test, y_pred5))
print('Recall Score score of SVC is:  ', recall_score(y_test, y_pred5))
print('F_1 Score of SVC is:           ', f1_score(y_test, y_pred5))
```

```
Accuracy Score of SVC is:      0.8706842435655995
Precision Score of SVC is:     0.8701380175658721
Recall Score score of SVC is:  0.8712311557788944
F_1 Score of SVC is:           0.8706842435655996
```

20. Applying Gradient Boost Classifier

In [76]:
```
from sklearn.ensemble import GradientBoostingClassifier

GBClass = GradientBoostingClassifier()
```

In [77]:
```
GBClass.fit(X_train, y_train)
```

Out[77]:
```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

In [78]:
```
y_pred6 = GBClass.predict(X_test)
```

In [79]:
```
print('Accuracy Score of SVC is:     ', accuracy_score(y_test, y_pred6))
print('Precision Score of SVC is:    ', precision_score(y_test, y_pred6))
print('Recall Score score of SVC is: ', recall_score(y_test, y_pred6))
print('F_1 Score of SVC is:          ', f1_score(y_test, y_pred6))
```

```
Accuracy Score of SVC is:      0.8402385436283741
Precision Score of SVC is:     0.8429385687143762
Recall Score score of SVC is:  0.8360552763819096
F_1 Score of SVC is:           0.8394828129927467
```

21. Saving And Visualising Accuracy And Precsion Scores

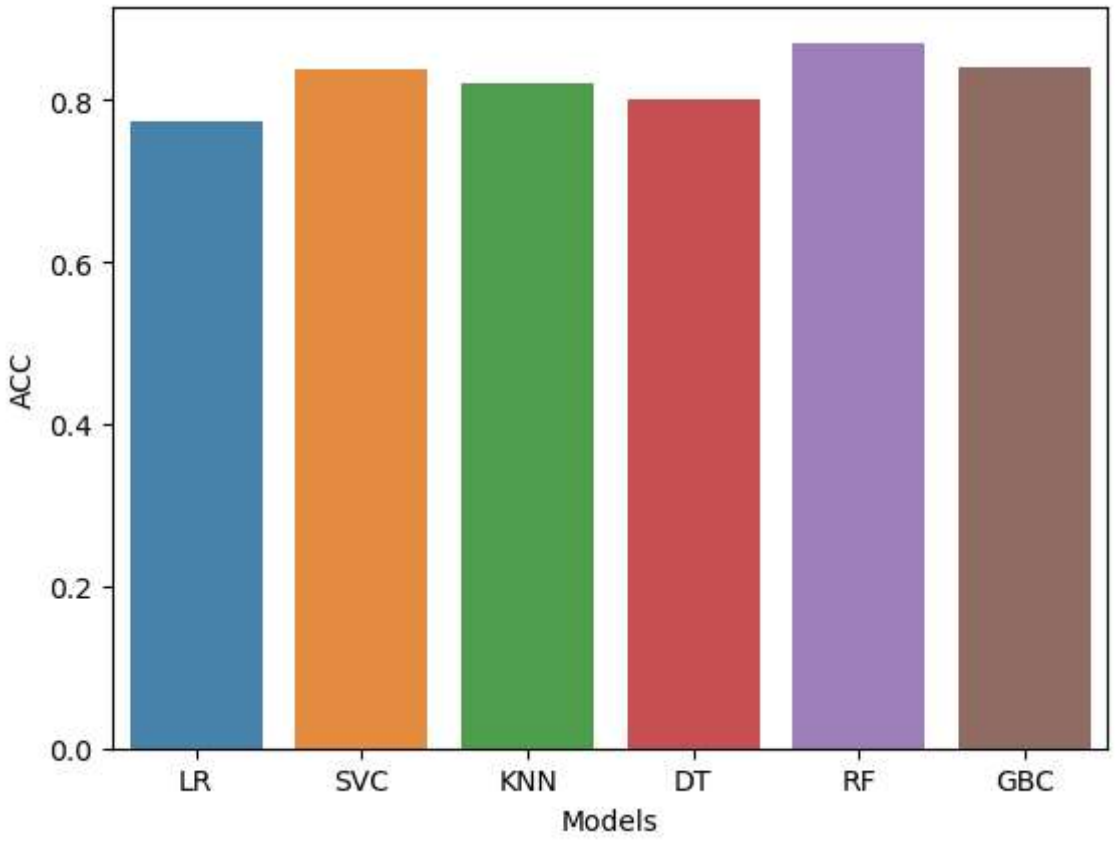
In [118…
```
accuracies = pd.DataFrame({'Models':['LR', 'SVC', 'KNN', 'DT', 'RF', 'GBC'], 'ACC':[ac
                                                                                    a
                                                                                   ac
                                                                                   ac
                                                                                   ac
                                                                                  acc
```

In [115…
```
print(accuracies)

sns.barplot(x=accuracies['Models'], y=accuracies['ACC'], alpha=0.9 )
```

```
  Models       ACC
0     LR  0.773384
1    SVC  0.838041
2    KNN  0.819523
3     DT  0.801632
4     RF  0.870684
5    GBC  0.840239
```

Out[115]:
```
<Axes: xlabel='Models', ylabel='ACC'>
```



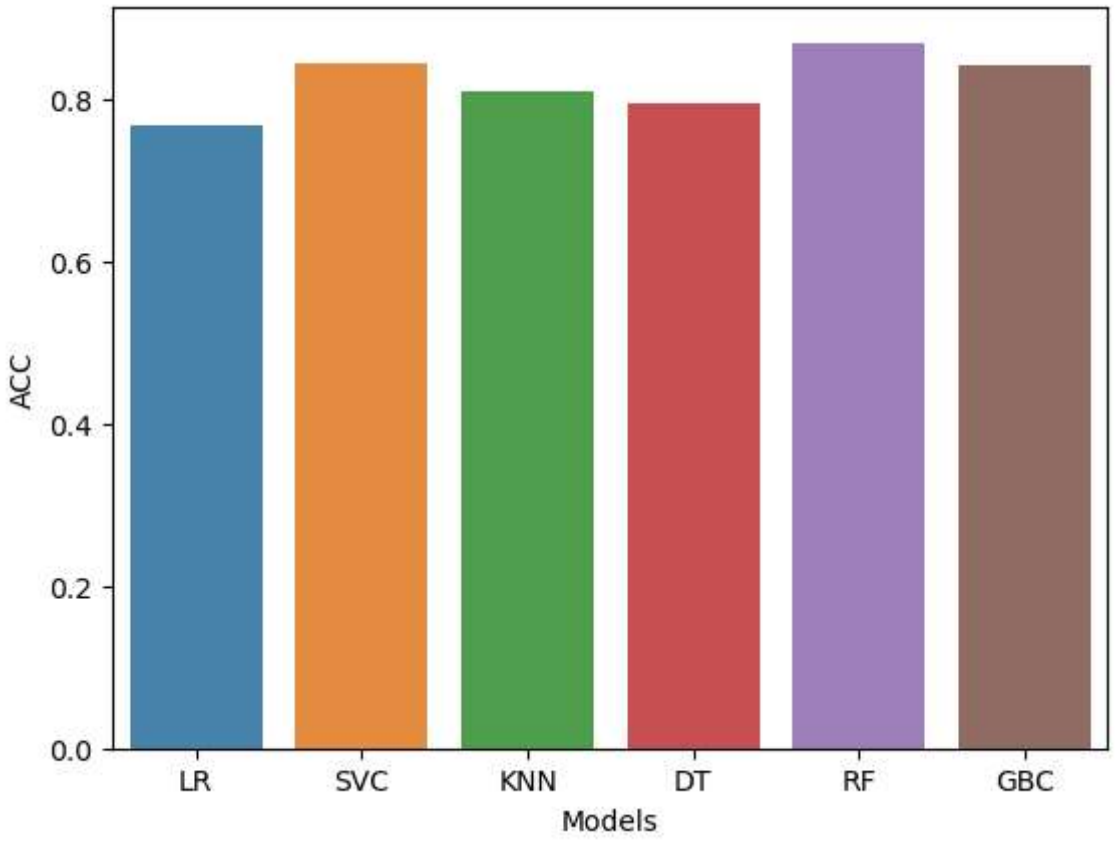
In [119…
```
precisions = pd.DataFrame({'Models':['LR', 'SVC', 'KNN', 'DT', 'RF', 'GBC'], 'ACC':[pr
                                                                                    p
                                                                                   pr
                                                                                   pr
                                                                                   pr
                                                                                  pre
```

In [120…
```
print(precisions)

sns.barplot(x=precisions['Models'], y=precisions['ACC'], alpha=0.9 )
```

```
  Models       ACC
0     LR  0.768187
1    SVC  0.845315
2    KNN  0.811009
3     DT  0.795567
4     RF  0.870138
5    GBC  0.842939
```

Out[120]:
```
<Axes: xlabel='Models', ylabel='ACC'>
```



22. Saving The Model For Later Use

In [123…
```
Xs = sc.fit_transform(Xs)

RFClass.fit(Xs, ys)
```

Out[123]:
```
▾ RandomForestClassifier

RandomForestClassifier()
```

In [125…
```
import joblib

joblib.dump(RFClass,'Customer_Churn_Prediction_Model')
```

Out[125]:
```
['Customer_Churn_Prediction_Model']
```

23. Loading The Model For Prediction

In [126…
```
model = joblib.load('Customer_Churn_Prediction_Model')
```

In [142…
```
model.predict([[608,41,1,100807,1,0,1,200000.58,0,1,0]])
```

Out[142]:
```
array([0], dtype=int64)
```

In [130…
```
X
```

Out[130]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSa
0	619	42	2	0.00	1	1	1	10134
1	608	41	1	83807.86	1	0	1	11254
2	502	42	8	159660.80	3	1	0	11393
3	699	39	1	0.00	2	0	0	9382
4	850	43	2	125510.82	1	1	1	7908
...	...	...	...	...	...	...	...	
9995	771	39	5	0.00	2	1	0	9627
9996	516	35	10	57369.61	1	1	1	10169
9997	709	36	7	0.00	1	0	1	4208
9998	772	42	3	75075.31	2	1	0	9288
9999	792	28	4	130142.79	1	1	0	3819

10000 rows × 11 columns

In [143…

```
ys
```

Out[143]:

```
0        1
1        0
2        1
3        0
4        0
        ..
15921    1
15922    1
15923    1
15924    1
15925    1
Name: Exited, Length: 15926, dtype: int64
```

In [ ]: