CUSTOMER CHURN PREDICTION

In [106…]
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn import svm
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [2]:
```
data = pd.read_csv('C:\Churn_Modelling.csv')
```

In [3]:
```
data.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	N
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 [4]:
```
data.tail()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.0(
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.6
9997	9998	15584532	Liu	709	France	Female	36	7	0.0(
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.3
9999	10000	15628319	Walker	792	France	Female	28	4	130142.7

In [5]:
```
data.shape
```

Out[5]:
```
(10000, 14)
```

In [6]:
```
print("Number of Rows", data.shape[0])
print("Number of Columns", data.shape[1])
```

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

In [7]:
```
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 [10]:
```
data.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
```

In [12]:
```
data.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	1000
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	

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	
max	10000.00000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000	1

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

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

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

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

Out[16]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveM
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	

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

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

In [19]:
```
data = pd.get_dummies(data,drop_first = True)
```

In [20]:
```
data.head()
```

Out[20]:

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSala
0	619	42	2	0.00	1	1	1	101348.8
1	608	41	1	83807.86	1	0	1	112542.5
2	502	42	8	159660.80	3	1	0	113931.5
3	699	39	1	0.00	2	0	0	93826.6
4	850	43	2	125510.82	1	1	1	79084.1

In [80]:
```
(data[data['Exited'] == 1].shape[0] / data.shape[0])*100
```

Out[80]:
```
20.369999999999997
```

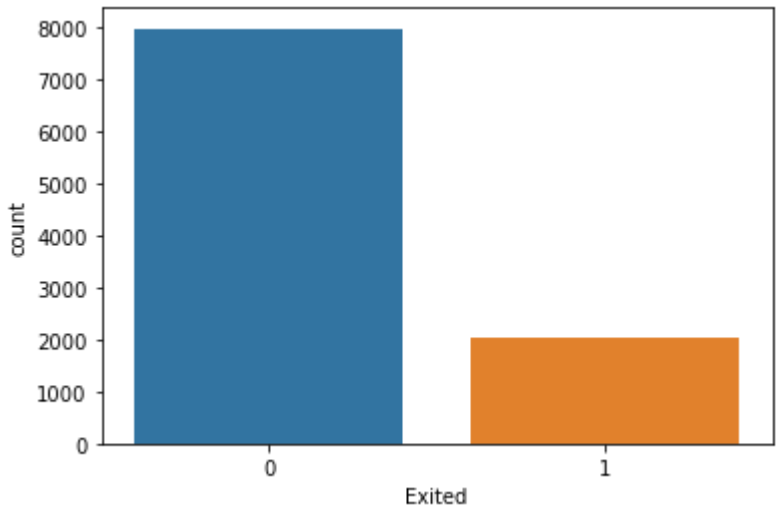
In [22]:
```
data['Exited'].value_counts()
```

Out[22]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```

In [31]:
```
sns.countplot(data['Exited'])
```

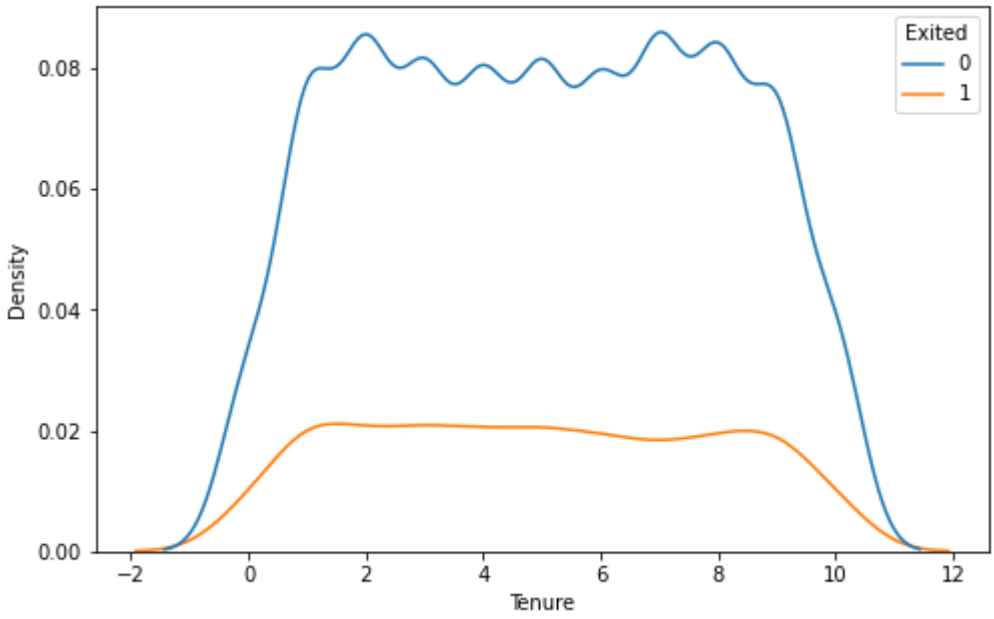
```
C:\Users\SYS\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variable as a keyword arg: x. From version 0.12, the only valid po
sitional argument will be `data`, and passing other arguments without an explicit ke
yword will result in an error or misinterpretation.
  warnings.warn(
```

Out[31]:
```
<AxesSubplot:xlabel='Exited', ylabel='count'>
```



In [82]:
```
plt.figure(figsize=(8,5))
sns.kdeplot(x = data['Tenure'], hue = data['Exited'])
```

Out[82]:
```
<AxesSubplot:xlabel='Tenure', ylabel='Density'>
```



In [40]:
```
x = data.drop('Exited',axis = 1)
y = data['Exited']
```

In [33]:
```
y
```

Out[33]:
```
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
```

In [44]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size = 0.20,random_stat
```

In [47]:
```
 sc = StandardScaler()
```

In [48]:
```
x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
```

In [49]:
```
x_train
```

Out[49]:
```
array([[ 1.058568  ,  1.71508648,  0.68472287, ..., -0.57831252,
        -0.57773517,  0.90750738],
       [ 0.91362605, -0.65993547, -0.6962018 , ...,  1.72916886,
        -0.57773517,  0.90750738],
       [ 1.07927399, -0.18493108, -1.73189531, ...,  1.72916886,
        -0.57773517, -1.10191942],
       ...,
       [ 0.16821031, -0.18493108,  1.3751852 , ..., -0.57831252,
        -0.57773517, -1.10191942],
       [ 0.37527024, -0.37493284,  1.02995403, ..., -0.57831252,
         1.73089688,  0.90750738],
       [ 1.56586482,  1.14508121,  0.68472287, ..., -0.57831252,
         1.73089688,  0.90750738]])
```

LogisticRegression

In [51]:
```
log = LogisticRegression()
```

In [52]:
```
log.fit(x_train, y_train)
```

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

In [89]:
```
y_pred1 = log.predict(x_test)
```

In [90]:
```
accuracy_score(y_test,y_pred1)
```

Out[90]:
```
0.808
```

In [91]:
```
precision_score(y_test, y_pred1)
```

Out[91]: 0.5891472868217055

In [92]:
```
recall_score(y_test, y_pred1)
```

Out[92]: 0.18673218673218672

In [61]:
```
f1_score(y_test, y_pred1)
```

Out[61]: 0.2835820895522388

In [ ]:
```
pc = TP / (FP + TP)
```

rc = TP / (TP + FN)

SVM

In [66]:
```
svm = svm.SVC()
```

In [67]:
```
svm.fit(x_train,y_train)
```

Out[67]: SVC()

In [68]:
```
y_pred2 = svm.predict(x_test)
```

In [93]:
```
accuracy_score(y_test,y_pred2)
```

Out[93]: 0.861

In [94]:
```
precision_score(y_test, y_pred2)
```

Out[94]: 0.8341968911917098

KNeighbors Classifier

In [73]:
```
knn = KNeighborsClassifier()
```

In [95]:
```
knn.fit(x_train,y_train)
```

Out[95]: KNeighborsClassifier()

In [75]:
```
y_pred3 = knn.predict(x_test)
```

In [96]:
```
accuracy_score(y_test,y_pred3)
```

Out[96]: 0.824

In [97]:
```
precision_score(y_test, y_pred3)
```

Out[97]: 0.6222222222222222

Decision Tree Classifier

In [84]:
```
dt =  DecisionTreeClassifier()
```

In [85]:
```
dt.fit(x_train,y_train)
```

Out[85]: DecisionTreeClassifier()

In [86]:
```
y_pred4 = dt.predict(x_test)
```

In [98]:
```
accuracy_score(y_test,y_pred4)
```

Out[98]: 0.7865

In [99]:
```
precision_score(y_test, y_pred4)
```

Out[99]: 0.4772727272727273

RandomForestClassifier

In [101…
```
rf = RandomForestClassifier()
```

In [102…
```
rf.fit(x_train,y_train)
```

Out[102… RandomForestClassifier()

In [103…
```
y_pred5 = dt.predict(x_test)
```

In [104…
```
accuracy_score(y_test,y_pred5)
```

Out[104… 0.7865

In [105…
```
precision_score(y_test, y_pred5)
```

Out[105… 0.4772727272727273

GradientBoostingClassifier

In [107…
```
gbc = GradientBoostingClassifier()
```

In [108…
```
gbc.fit(x_train,y_train)
```

Out[108…
```
GradientBoostingClassifier()
```

In [109…
```
y_pred6 = dt.predict(x_test)
```

In [110…
```
accuracy_score(y_test,y_pred6)
```

Out[110…
```
0.7865
```

In [111…
```
precision_score(y_test, y_pred6)
```

Out[111…
```
0.4772727272727273
```

In [112…
```
final_data = pd.DataFrame({'Models': ['LR','SVC','KNN','DT','RF','GBC'],'ACC':[accur
```

In [113…
```
final_data
```

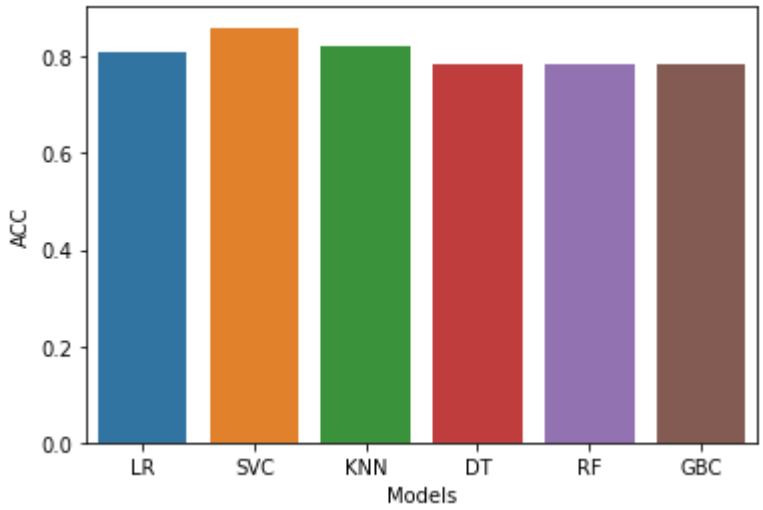
Out[113…

	Models	ACC
0	LR	0.8080
1	SVC	0.8610
2	KNN	0.8240
3	DT	0.7865
4	RF	0.7865
5	GBC	0.7865

In [114…
```
sns.barplot(final_data['Models'],final_data['ACC'])
```

```
C:\Users\SYS\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variables as keyword args: x, y. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[114…
```
<AxesSubplot:xlabel='Models', ylabel='ACC'>
```



In [116...
```
final_data = pd.DataFrame({'Models': ['LR','SVC','KNN','DT','RF','GBC'],'PRE':[preci
```

In [118...
```
final_data
```

Out[118...

	Models	PRE
0	LR	0.589147
1	SVC	0.834197
2	KNN	0.622222
3	DT	0.477273
4	RF	0.477273
5	GBC	0.477273

In [119...
```
sns.barplot(final_data['Models'],final_data['PRE'])
```

```
C:\Users\SYS\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: P
ass the following variables as keyword args: x, y. From version 0.12, the only valid
positional argument will be `data`, and passing other arguments without an explicit
keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[119...
```
<AxesSubplot:xlabel='Models', ylabel='PRE'>
```