Importing necessary libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
%pip install xgboost
import xgboost as xgb

from sklearn import metrics
from sklearn.metrics import accuracy_score, auc, confusion_matrix, roc_auc_score, roc_curve, recall_score
```

```
Requirement already satisfied: xgboost in c:\users\batman\appdata\local\programs\python\python310\lib\site-packages (1.7.4)
Requirement already satisfied: scipy in c:\users\batman\appdata\local\programs\python\python310\lib\site-packages (from xgboost) (1.10.1)
Requirement already satisfied: numpy in c:\users\batman\appdata\local\programs\python\python310\lib\site-packages (from xgboost) (1.24.2)
Note: you may need to restart the kernel to use updated packages.

[notice] A new release of pip available: 22.2.1 -> 23.0.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

```
df=pd.read_csv("C:/Users/Batman/Documents/DF-2209/Machine Learning/project/Churn Modeling.csv")
# Reading dataset
```

```
df
# loading dataset
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfPro
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	

Data Understanding

9996 9997 15569892 Johnstone 516 France Male 35 10 57369.61

```
df.shape
# There are 10,000 rows and 14 columns
```

```
(10000, 14)
```

9999 10000 15628319 Walker 792 France Female 28 4 130142.79

```
df.info()
# Checking the information of the data
```

```
<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
```

```
df.isnull().sum()
# There are no null values in the dataset
```

```
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
```

```
df['Exited'].value_counts()
# As it can be seen this dataset's targer column is imbalanced there are 7963 zero's and 2037 one's
```

```
0    7963
1    2037
Name: Exited, dtype: int64
```

```
df.describe()
# The describe function will display all the decriptive statistics of the data including mean, std, min, max values.
```

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

```
df.nunique()
# The nunique() function is used to count distinct observations over requested axis. Return Series with number of distinct observations.
```

```
RowNumber          10000
CustomerId         10000
Surname             2932
CreditScore          460
Geography              3
Gender                 2
Age                   70
Tenure                11
Balance             6382
NumOfProducts          4
HasCrCard              2
IsActiveMember         2
EstimatedSalary     9999
```

```
Exited                2
dtype: int64
```

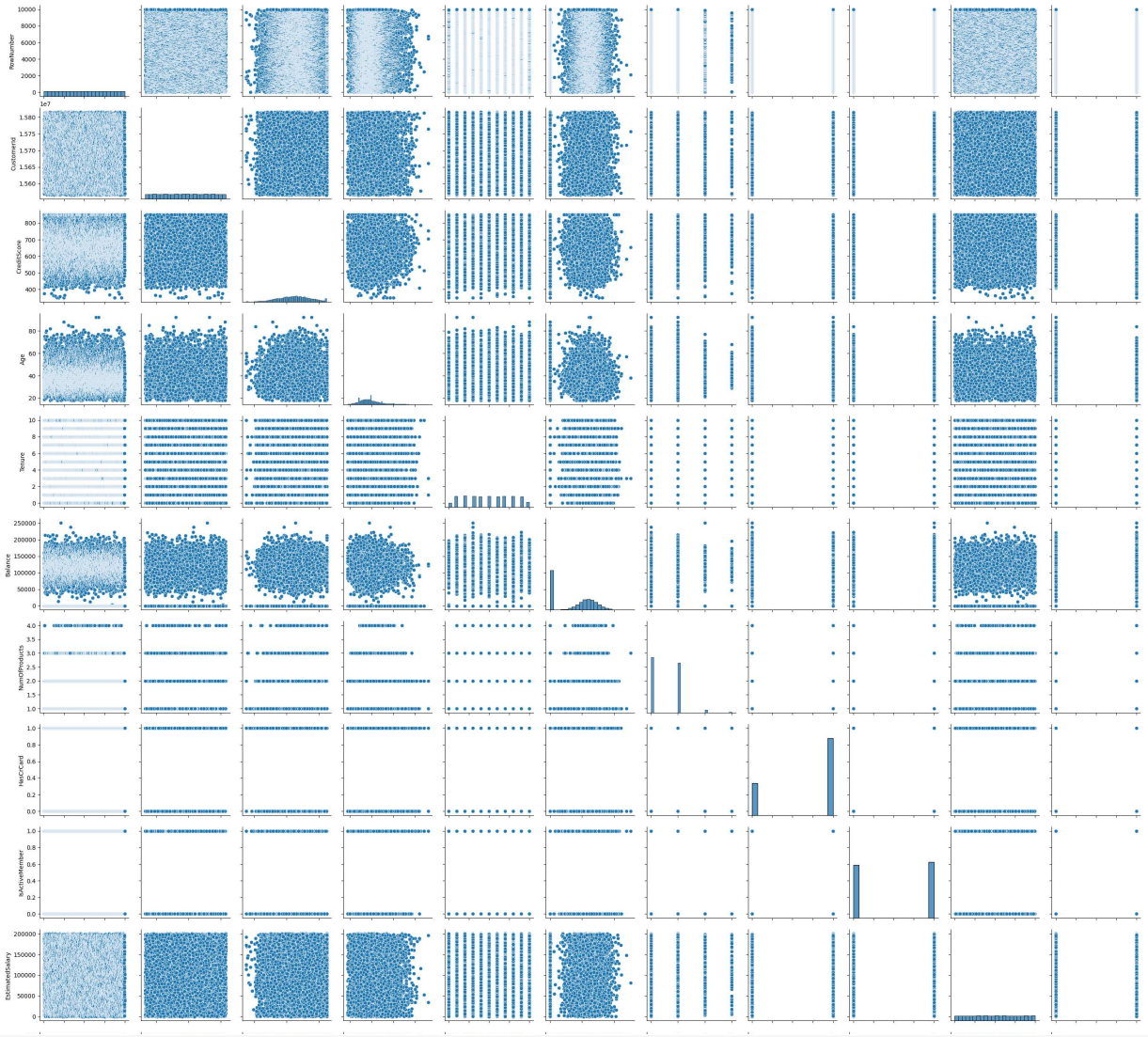
```
df.columns
# Displaying the column names of the dataset.
```

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

Data Visualization

```
sns.pairplot(data=df)
# Displaying scatter plots
```

```
<seaborn.axisgrid.PairGrid at 0x26c7720da20>
```



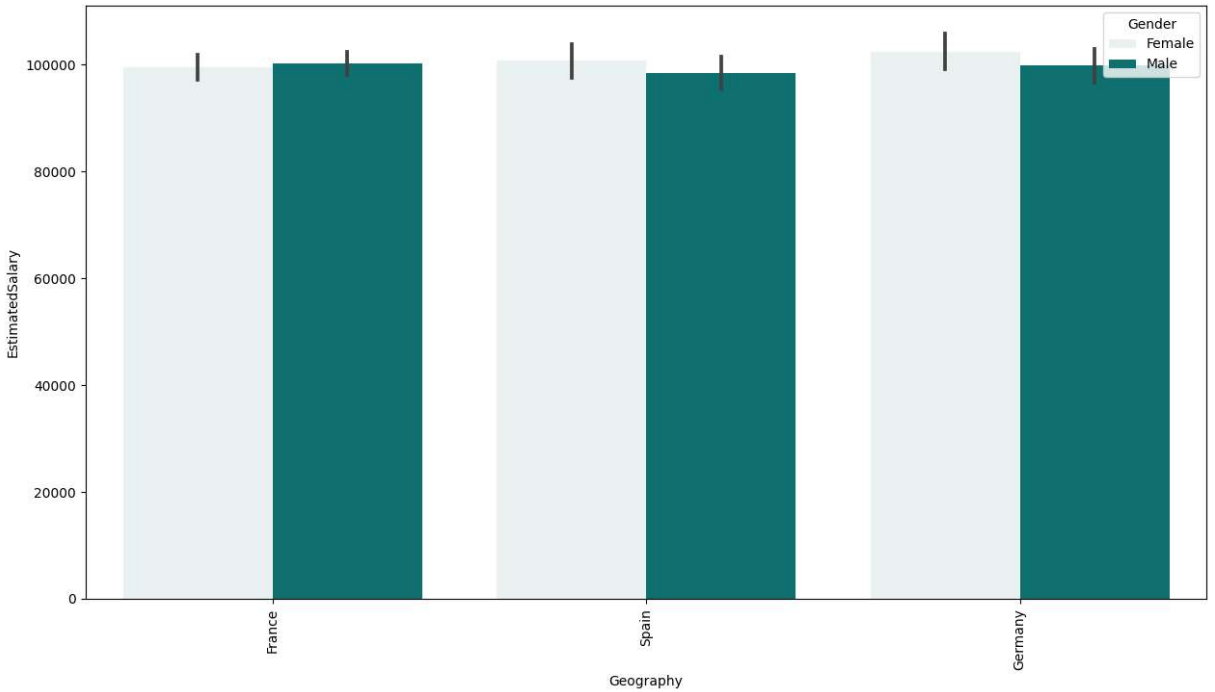

```
df.columns
# Displaying column names
```

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

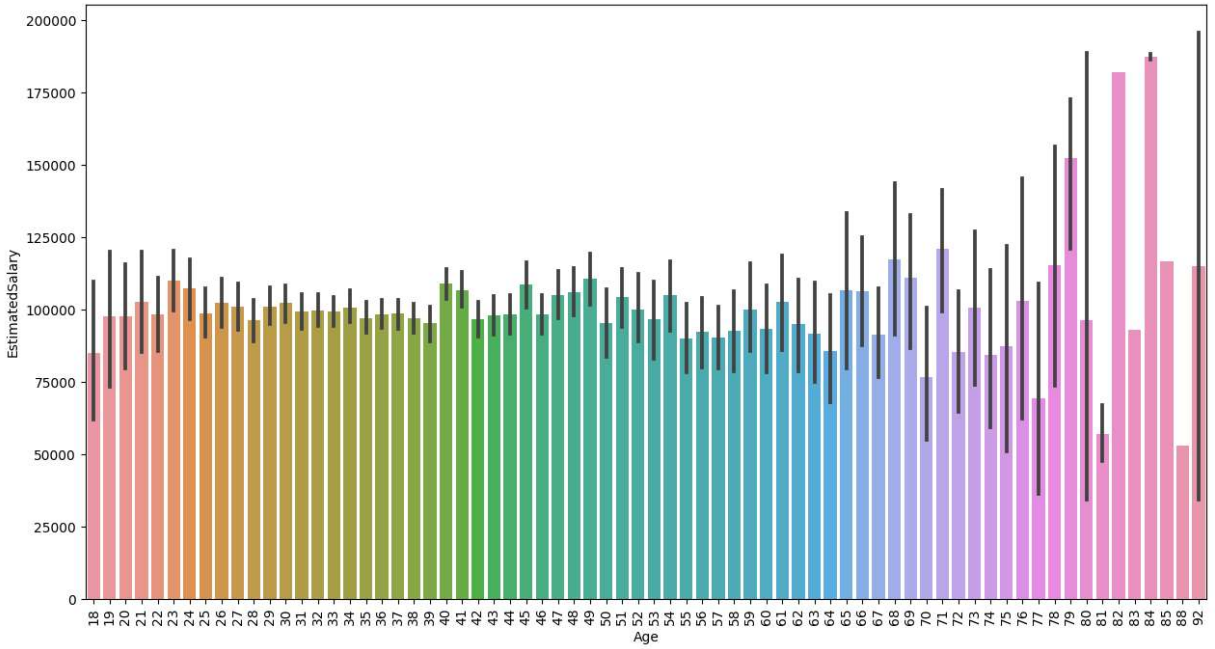
```
df.head(5)
# Displaying the head of the dataset
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProduct
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	

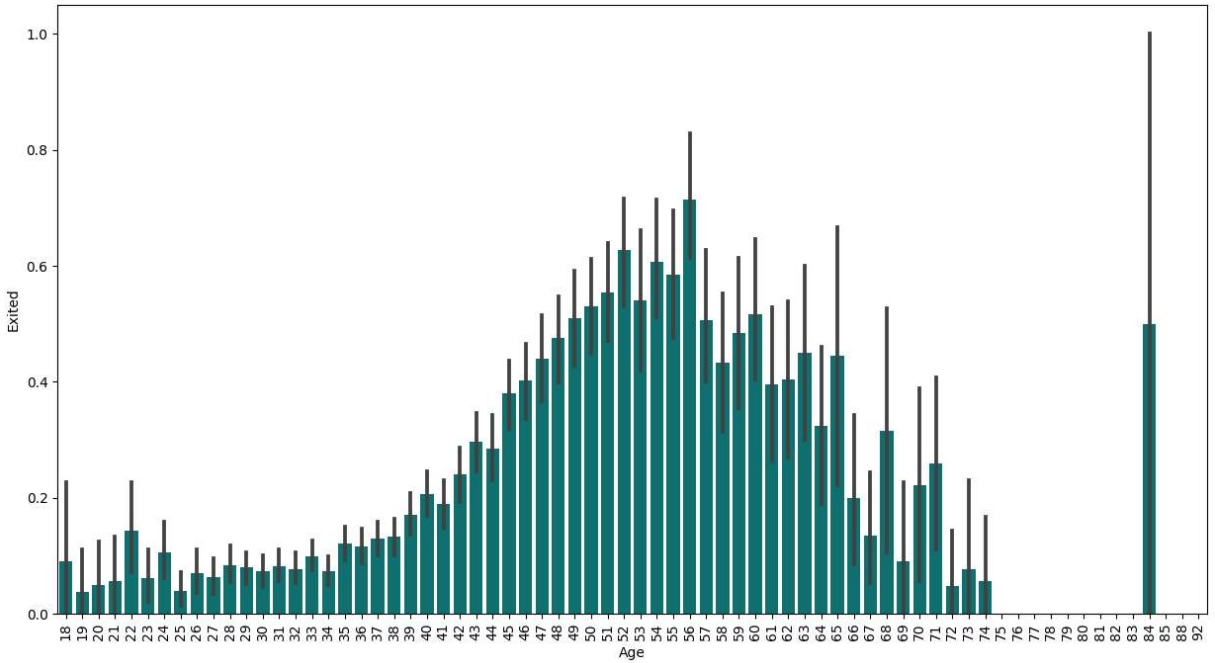
```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Geography',y='EstimatedSalary',hue='Gender',color='teal',data=df);
# Visualizing barplot where Geography is on x-axis and estimated salary is on y axis and key is gender
```



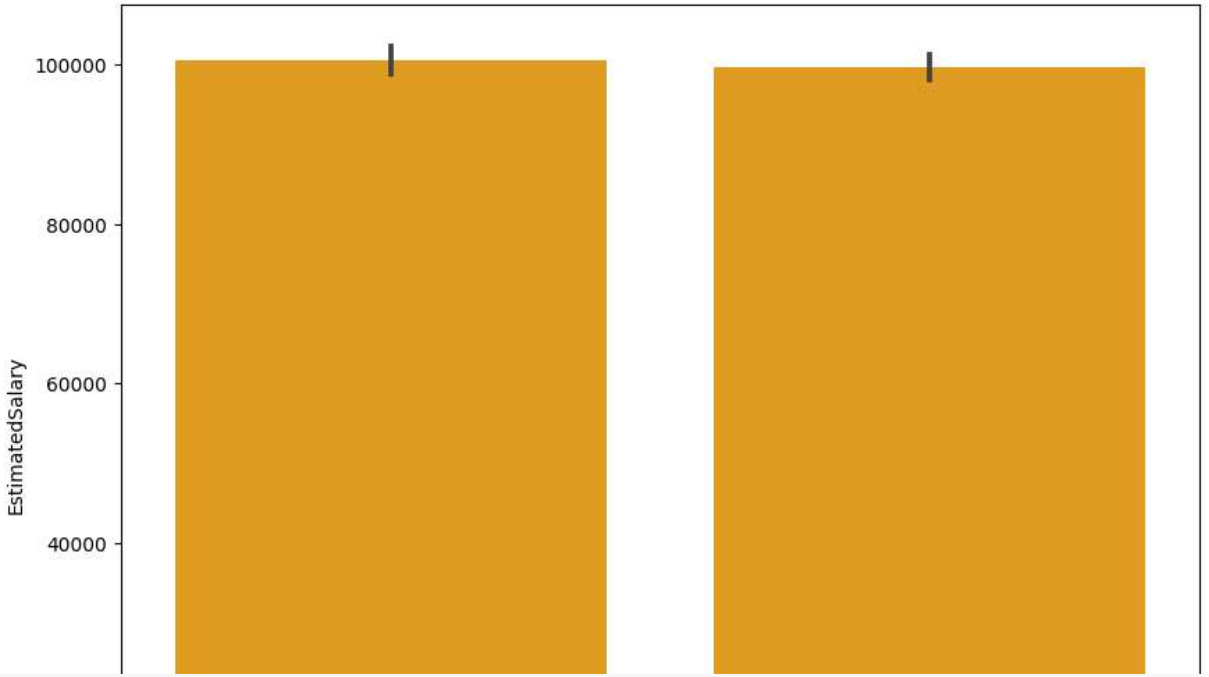
```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Age',y='EstimatedSalary',data=df);
# Visualizing barplot where age is taken on x axis and estimated salary is taken on y axis
```




```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Age',y='Exited',color='teal',data=df);
# Visualizing barplot where age is taken on x axis and exited in on y axis
```



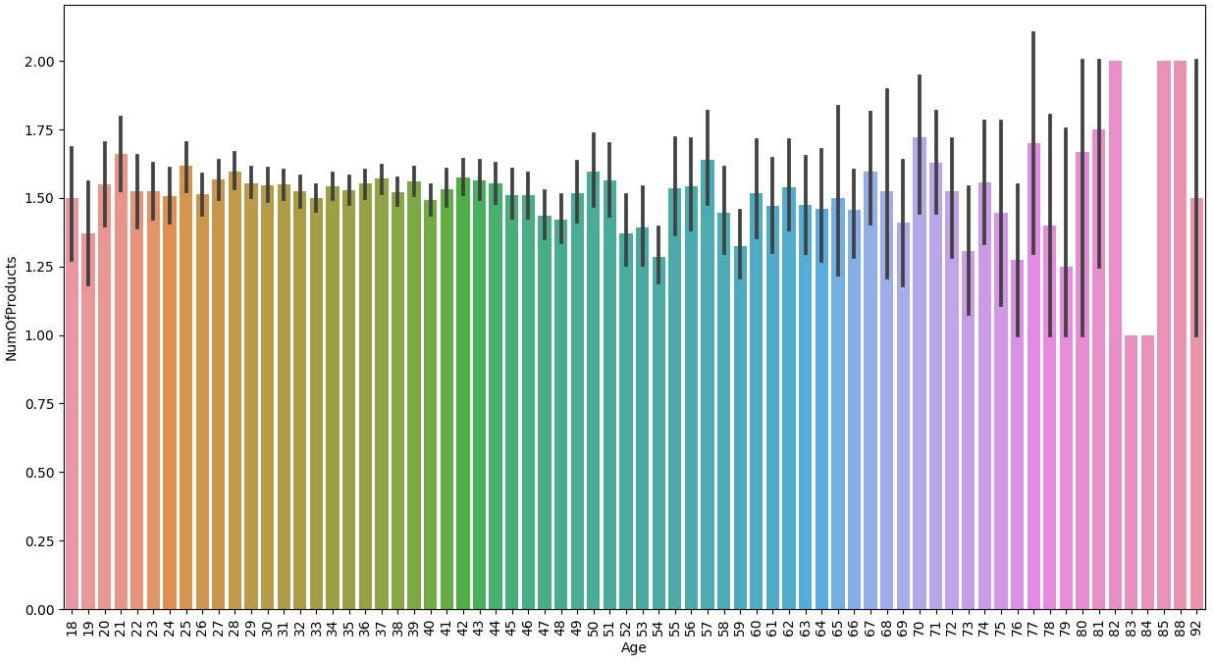

```
plt.figure(figsize=(10, 8))
plt.xticks(rotation=90)
sns.barplot(x='Gender',y='EstimatedSalary',color='orange',data=df);
# Visualizing barplot where gender is taken on x axis and estimated salary is taken on y axis
```



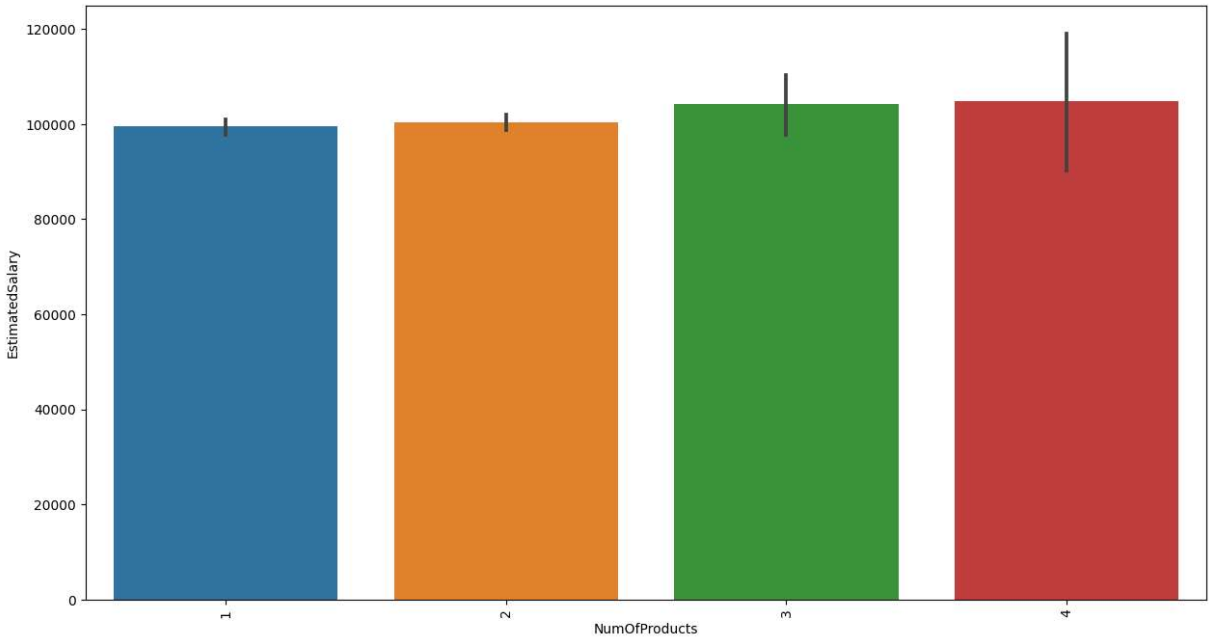

```
plt.figure(figsize=(10, 8))
plt.xticks(rotation=90)
sns.barplot(x='Gender',y='Balance',color='teal',data=df);
# Visualizing barplot where Gender is taken on x axis and Balance is taken on y axis
```


```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Age',y='EstimatedSalary',hue='Gender',data=df);
# Visualizing barplot where Age is taken on x axis and EstimatedSalary is taken on y axis
```

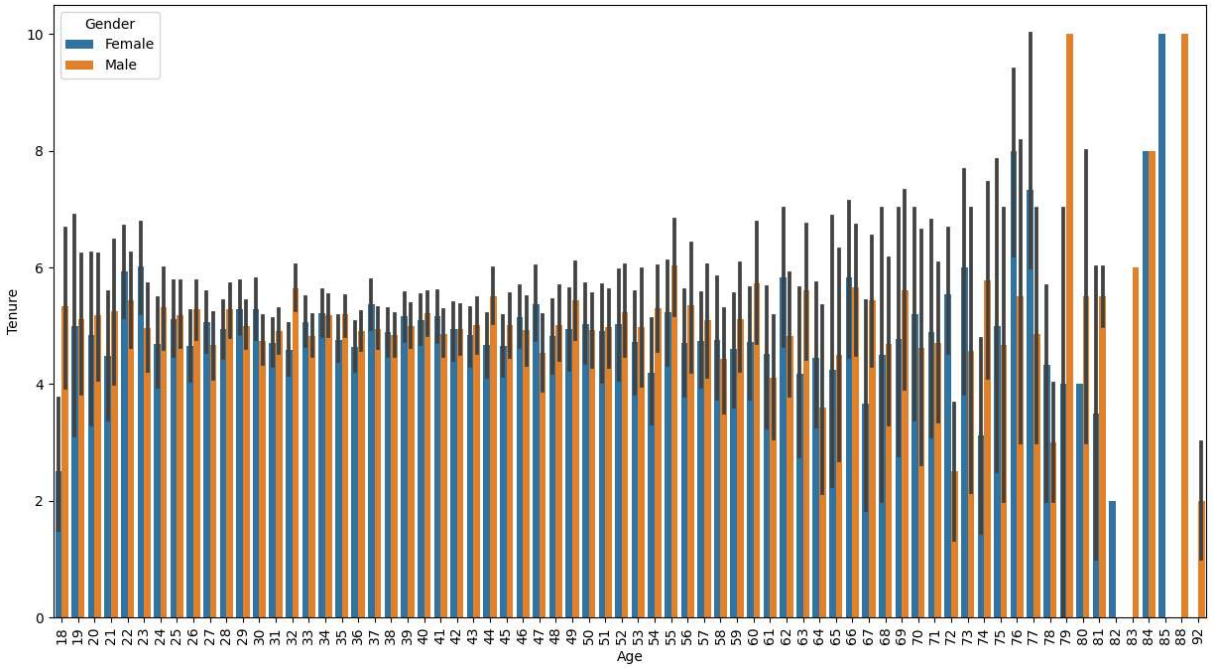

```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Age',y='NumOfProducts',data=df);
# Visualizing barplot where Age is taken on x axis and NumOfProducts is taken on y axis
```




```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='NumOfProducts',y='EstimatedSalary',data=df);
# Visualizing barplot where NumOfProducts is taken on x axis and EstimatedSalary is taken on y axis
```



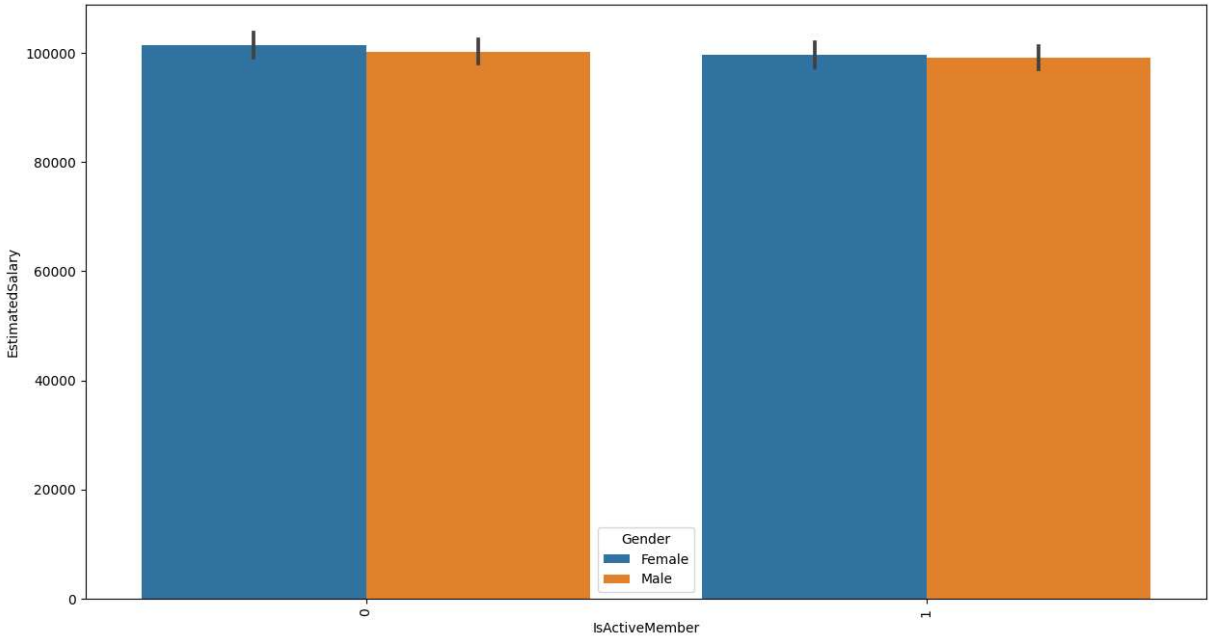

```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='Age',y='Tenure',hue='Gender',data=df);
# Visualizing barplot where Age is taken on x axis and Tenure is taken on y axis and key is gender
```



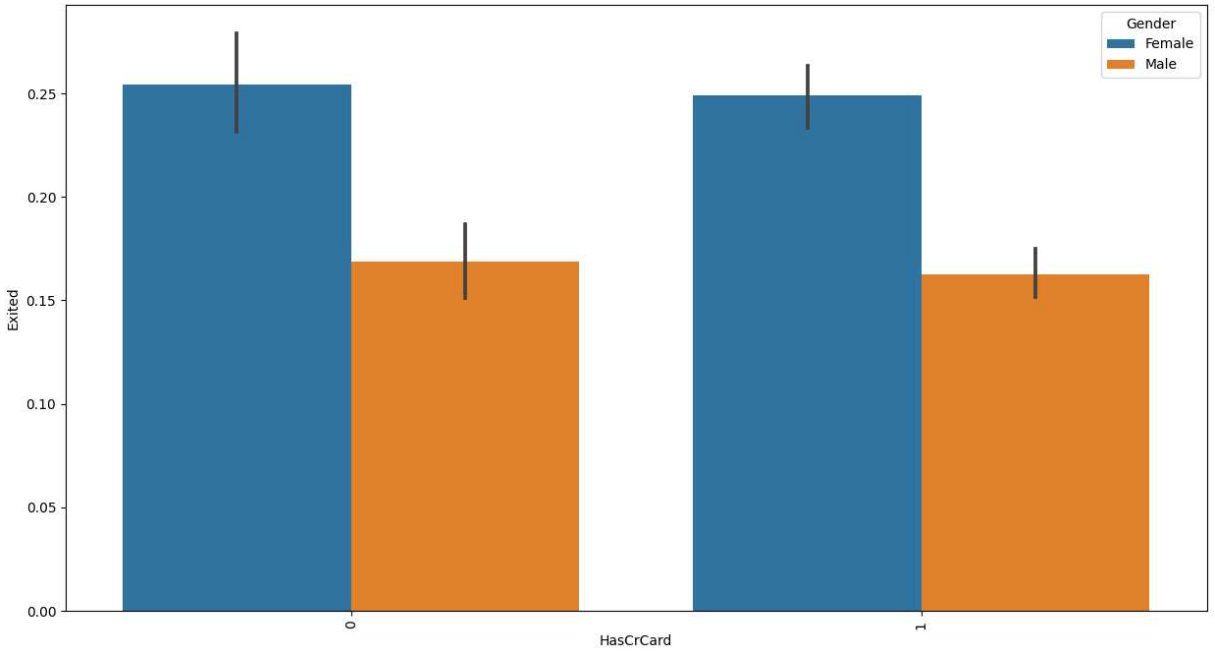

```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='IsActiveMember',y='Geography',hue='Gender',data=df);
# Visualizing barplot where IsActiveMember is taken on x axis and Geography is taken on y axis and key is gender
```


```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='IsActiveMember',y='Exited',hue='Gender',data=df);
# Visualizing barplot where IsActiveMember is taken on x axis and Exited is taken on y axis and key is gender
```

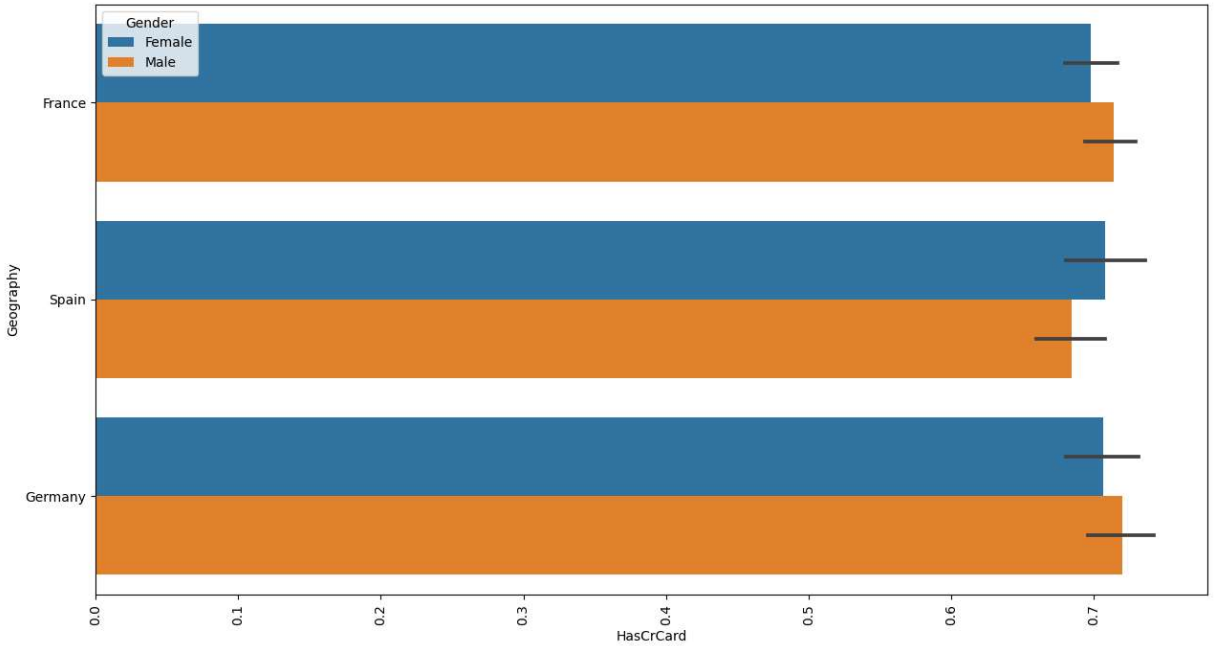

```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='IsActiveMember',y='EstimatedSalary',hue='Gender',data=df);
# Visualizing barplot where IsActiveMember is taken on x axis and EstimatedSalary is taken on y axis and key is gender
```




```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='HasCrCard',y='Exited',hue='Gender',data=df);
# Visualizing barplot where HasCrCard is taken on x axis and Exited is taken on y axis and key is gender
```




```
plt.figure(figsize=(15, 8))
plt.xticks(rotation=90)
sns.barplot(x='HasCrCard',y='Geography',hue='Gender',data=df);
# Visualizing barplot where HasCrCard is taken on x axis and Geography is taken on y axis and key is gender
```



Label Encoding

```
cat_cols=['Geography','Gender']
le=LabelEncoder()
for i in cat_cols:
    df[i]=le.fit_transform(df[i])
df.dtypes
# We convert categorical data into numeric data with the help of label encoding
```

```
RowNumber            int64
CustomerId           int64
Surname             object
CreditScore          int64
Geography            int32
Gender               int32
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
Exited               int64
dtype: object
```

```
df.keys()
# displaying columns
```

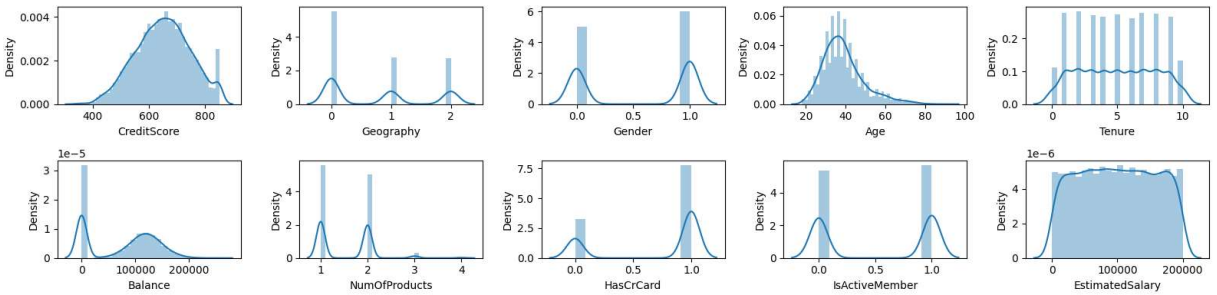
```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',
       'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

```
df.drop(['RowNumber'],axis=1,inplace=True)
df.drop(['CustomerId'],axis=1,inplace=True)
df.drop(['Surname'],axis=1,inplace=True)
# dropping uneccessary columns and removing them from the dataset
```

DistributionPlot

```
rows=2
cols=5
fig, ax=plt.subplots(nrows=rows,ncols=cols,figsize=(16,4))
col=df.columns
index=0
for i in range(rows):
    for j in range(cols):
        sns.distplot(df[col[index]],ax=ax[i][j])
        index=index+1

plt.tight_layout()
# Distribution plot will help us to check if the data is skewed or not
```




```
X=df.drop(labels=['Exited'],axis=1)
Y=df['Exited']
```

```
X.head()
# Splitting data into dependent and independent columns
```

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

```
Y.head()
# This is the target column
```

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

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=40)
print(X_train.shape,X_test.shape,Y_train.shape,Y_test.shape)
# Splitting the data set into training and testing data
```

```
(8000, 10) (2000, 10) (8000,) (2000,)
```

Logistic Regression

```
#fit the model on train data
log_reg = LogisticRegression().fit(X_train, Y_train)

#predict on train
train_preds = log_reg.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds))

#predict on test
test_preds = log_reg.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds))
```

```
print('-'*50)

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds).sum(),'/',((Y_test == test_preds).sum()+(Y_test != test_preds).sum()))
print('-'*50)

# Kappa Score.
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds))
```

```
Model accuracy on train is:  0.788
Model accuracy on test is:  0.8
--------------------------------------------------
ROC score on train is:  0.5196539781240042
ROC score on test is:  0.519866439768977
--------------------------------------------------
confusion_matrix train is:  [[6201  146]
 [1550  103]]
confusion_matrix test is:  [[1575   41]
 [ 359   25]]
Wrong predictions out of total
--------------------------------------------------
400 / 2000
--------------------------------------------------
KappaScore is:  0.05806111299498873
```

Naive Bayes Classifier

```
#fit the model on train data
NB=GaussianNB()
NB.fit(X_train,Y_train)

#predict on train
train_preds2 = NB.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds2))

#predict on test
test_preds2 = NB.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds2))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds2))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds2))
```

```
print('-'*50)

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds2))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds2))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds2).sum(),'/',((Y_test == test_preds2).sum()+(Y_test != test_preds2).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds2))
```

```
Model accuracy on train is:  0.781875
Model accuracy on test is:  0.7915
--------------------------------------------------
ROC score on train is:  0.5236234904696533
ROC score on test is:  0.5225479579207921
--------------------------------------------------
confusion_matrix train is:  [[6117  230]
 [1515  138]]
confusion_matrix test is:  [[1550   66]
 [ 351   33]]
Wrong predictions out of total
--------------------------------------------------
417 / 2000
--------------------------------------------------
KappaScore is:  0.0628876543875736
```

Decision Tree Classifier

```
#fit the model on train data
DT = DecisionTreeClassifier().fit(X,Y)

#predict on train
train_preds3 = DT.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds3))

#predict on test
test_preds3 = DT.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds3))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds3))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds3))
print('-'*50)
```

```
#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds3))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds3))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds3).sum(),'/',((Y_test == test_preds3).sum()+(Y_test != test_preds3).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds3))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  1.0
--------------------------------------------------
ROC score on train is:  1.0
ROC score on test is:  1.0
--------------------------------------------------
confusion_matrix train is:  [[6347    0]
 [   0 1653]]
confusion_matrix test is:  [[1616    0]
 [   0  384]]
Wrong predictions out of total
--------------------------------------------------
0 / 2000
--------------------------------------------------
KappaScore is:  1.0
```

Random Forest Classifier

```
#fit the model on train data
RF=RandomForestClassifier().fit(X_train,Y_train)
#predict on train
train_preds4 = RF.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds4))

#predict on test
test_preds4 = RF.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds4))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds4))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds4))
print('-'*50)

#Confusion matrix
```

```
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds4))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds4))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds4).sum(),'/',((Y_test == test_preds4).sum()+(Y_test != test_preds4).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds4))
```

```
Model accuracy on train is:  1.0
Model accuracy on test is:  0.8695
--------------------------------------------------
ROC score on train is:  1.0
ROC score on test is:  0.7286509900990099
--------------------------------------------------
confusion_matrix train is:  [[6347    0]
 [   0 1653]]
confusion_matrix test is:  [[1547   69]
 [ 192  192]]
Wrong predictions out of total
--------------------------------------------------
261 / 2000
--------------------------------------------------
KappaScore is:  0.5209040045816996
```

K-Nearest Neighbours

```
#fit the model on train data
KNN = KNeighborsClassifier().fit(X_train,Y_train)
#predict on train
train_preds5 = KNN.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds5))

#predict on test
test_preds5 = KNN.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds5))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds5))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds5))
print('-'*50)

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds5))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds5))
```

```
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds5).sum(),'/',((Y_test == test_preds5).sum()+(Y_test != test_preds5).sum()))

print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds5))
```

```
Model accuracy on train is:  0.8105
Model accuracy on test is:  0.7705
--------------------------------------------------
ROC score on train is:  0.5935625969407309
ROC score on test is:  0.5085602310231023
--------------------------------------------------
confusion_matrix train is:  [[6114  233]
 [1283  370]]
confusion_matrix test is:  [[1509  107]
 [ 352   32]]
Wrong predictions out of total
--------------------------------------------------
459 / 2000
--------------------------------------------------
KappaScore is:  0.022622353201710355
```

Support Vector Machine

```
#fit the model on train data
SVM = SVC(kernel='linear')
SVM.fit(X_train, Y_train)

#predict on train
train_preds6 = SVM.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds6))

#predict on test
test_preds6 = SVM.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds6))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds6))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds6))
print('-'*50)

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds6))
```

```
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds6))
print('Wrong predictions out of total')
print('-'*50)

print("recall", metrics.recall_score(Y_test, test_preds6))
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds6).sum(),'/',((Y_test == test_preds6).sum()+(Y_test != test_preds6).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds6))
```

```
Model accuracy on train is:  0.778375
Model accuracy on test is:  0.7885
--------------------------------------------------
ROC score on train is:  0.5073244372564656
ROC score on test is:  0.5058013613861386
--------------------------------------------------
confusion_matrix train is:  [[6152  195]
 [1578   75]]
confusion_matrix test is:  [[1559   57]
 [ 366   18]]
Wrong predictions out of total
--------------------------------------------------
recall 0.046875
--------------------------------------------------
423 / 2000
--------------------------------------------------
KappaScore is:  0.016736401673640322
```

XG-Boost Classifier

```
xgbr =xgb.XGBClassifier().fit(X_train, Y_train)

#predict on train
train_preds7 = xgbr.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds7))

#predict on test
test_preds7 = xgbr.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds7))
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds7))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds7))
print('-'*50)
```

```
#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds7))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds7))
print('Wrong predictions out of total')
print('-'*50)

print("recall", metrics.recall_score(Y_test, test_preds7))
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds7).sum(),'/',((Y_test == test_preds7).sum()+(Y_test != test_preds7).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds7))
```

```
Model accuracy on train is:  0.957375
Model accuracy on test is:  0.867
--------------------------------------------------
ROC score on train is:  0.9046838082041132
ROC score on test is:  0.7469575082508251
--------------------------------------------------
confusion_matrix train is:  [[6312   35]
 [ 306 1347]]
confusion_matrix test is:  [[1522   94]
 [ 172  212]]
Wrong predictions out of total
--------------------------------------------------
recall 0.5520833333333334
--------------------------------------------------
266 / 2000
--------------------------------------------------
KappaScore is:  0.5353679327017131
```

```
# Random Forest Classifier and Xg-boost Regressor model performed well compared to other models
```

Hyper Parameter Tuning

```
#fit the model on train data
RFT=RandomForestClassifier(n_estimators=500,min_samples_split=2,min_samples_leaf=1,max_features='sqrt',random_state=235,verbose=2,max_samples=50).fit(X_train,Y_train)
#predict on train
train_preds8 = RFT.predict(X_train)
#accuracy on train
print("Model accuracy on train is: ", accuracy_score(Y_train, train_preds8))

#predict on test
test_preds8 = RFT.predict(X_test)
#accuracy on test
print("Model accuracy on test is: ", accuracy_score(Y_test, test_preds8))
```

```
print('-'*50)

#We got good accuracy which means our model is performing quite well
#ROC
print("ROC score on train is: ", roc_auc_score(Y_train, train_preds8))
print("ROC score on test is: ", roc_auc_score(Y_test, test_preds8))
print('-'*50)

#Confusion matrix
print("confusion_matrix train is: ", confusion_matrix(Y_train, train_preds8))
print("confusion_matrix test is: ", confusion_matrix(Y_test, test_preds8))
print('Wrong predictions out of total')
print('-'*50)

# Wrong Predictions made.
print((Y_test !=test_preds8).sum(),'/',((Y_test == test_preds8).sum()+(Y_test != test_preds8).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,test_preds8))
```

```
building tree 500 of 500
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    0.6s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
Model accuracy on train is:  0.817375
Model accuracy on test is:  0.833
------------------------------------------------
ROC score on train is:  0.5627739872818145
ROC score on test is:  0.5720529084158417
------------------------------------------------
confusion_matrix train is:  [[6326   21]
 [1440  213]]
confusion_matrix test is:  [[1609    7]
 [ 327   57]]
Wrong predictions out of total
------------------------------------------------
334 / 2000
------------------------------------------------
KappaScore is:  0.21119256348246673
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    0.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:    0.0s
[Parallel(n_jobs=1)]: Done 500 out of 500 | elapsed:    0.0s finished
```

A. RandomSearchCv

```
from sklearn.model_selection import RandomizedSearchCV
```

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 5000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt','log2']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 2000,10)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10,14]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4,6,8]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
              'criterion':['entropy','gini']}
print(random_grid)
```

```
{'n_estimators': [100, 644, 1188, 1733, 2277, 2822, 3366, 3911, 4455, 5000], 'max_features': ['auto', 'sqrt', 'log2'], 'max_depth': [10, 231, 452, 673, 894, 1115, 1336,
```

```
RFT1=RandomForestClassifier()
rf_randomcv=RandomizedSearchCV(estimator=RFT1,param_distributions=random_grid,n_iter=100,cv=5,verbose=2,
```

```
                               random_state=100,n_jobs=-1)
### fit the randomized model
rf_randomcv.fit(X_train,Y_train)
```

```
Fitting 5 folds for each of 100 candidates, totalling 500 fits
```

RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier

```
rf_randomcv.best_params_
# best parameters
```

```
{'n_estimators': 100,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 673,
 'criterion': 'entropy'}
```

```
rf_randomcv
# displaying all parameters
```

RandomizedSearchCV
estimator: RandomForestClassifier
RandomForestClassifier

```
rf_randomcv.best_estimator_
# Displaying best parameters from all parameters mentioned above
```

RandomForestClassifier

```
RandomForestClassifier(criterion='entropy', max_depth=673, max_features='auto',
                       min_samples_leaf=4, min_samples_split=10)
```

```
best_random_grid=rf_randomcv.best_estimator_
# saving all parameters in best_random_grid
```

```
Y_pred=best_random_grid.predict(X_test)

print(confusion_matrix(Y_test,Y_pred))
print("Accuracy Score {}".format(accuracy_score(Y_test,Y_pred)))

# Wrong Predictions made.
print((Y_test !=Y_pred).sum(),'/',((Y_test == Y_pred).sum()+(Y_test != Y_pred).sum()))
print('-'*50)
```

```
# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,Y_pred))
# predicting the randomised search cv models parameters and evaluating the random forest model using evaluation metrics
```

```
[[1565   51]
 [ 193  191]]
Accuracy Score 0.878
244 / 2000
--------------------------------------------------
KappaScore is:  0.5422757151003992
```

B. GridSearchCv

```
from sklearn.model_selection import GridSearchCV
# Using grid search cv model
```

```
rf_randomcv.best_params_
# Visualizing the best parameters of random cv
```

```
{'n_estimators': 100,
 'min_samples_split': 10,
 'min_samples_leaf': 4,
 'max_features': 'auto',
 'max_depth': 673,
 'criterion': 'entropy'}
```

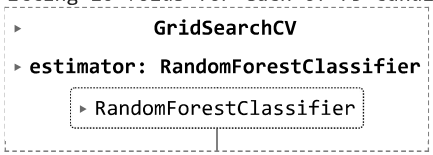
```
param_grid = {
    'criterion': [rf_randomcv.best_params_['criterion']],
    'max_depth': [rf_randomcv.best_params_['max_depth']],
    'max_features': [rf_randomcv.best_params_['max_features']],
    'min_samples_leaf': [rf_randomcv.best_params_['min_samples_leaf'],
                         rf_randomcv.best_params_['min_samples_leaf']+2,
                         rf_randomcv.best_params_['min_samples_leaf'] + 4],
    'min_samples_split': [rf_randomcv.best_params_['min_samples_split'] - 2,
                          rf_randomcv.best_params_['min_samples_split'] - 1,
                          rf_randomcv.best_params_['min_samples_split'],
                          rf_randomcv.best_params_['min_samples_split'] +1,
                          rf_randomcv.best_params_['min_samples_split'] + 2],
    'n_estimators': [rf_randomcv.best_params_['n_estimators'] - 200, rf_randomcv.best_params_['n_estimators'] - 100,
                     rf_randomcv.best_params_['n_estimators'],
                     rf_randomcv.best_params_['n_estimators'] + 100, rf_randomcv.best_params_['n_estimators'] + 200]
}

print(param_grid)
# creating a param_grid
```

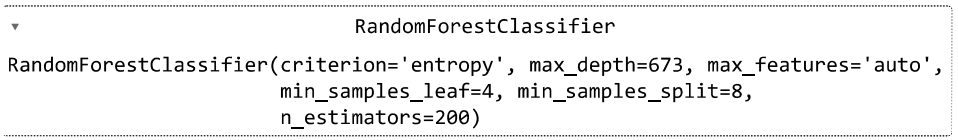
```
{'criterion': ['entropy'], 'max_depth': [673], 'max_features': ['auto'], 'min_samples_leaf': [4, 6, 8], 'min_samples_split': [8, 9, 10, 11, 12], 'n_estimators': [-100, 0
```

```
rf=RandomForestClassifier()
grid_search=GridSearchCV(estimator=rf,param_grid=param_grid,cv=10,n_jobs=-1,verbose=2)
grid_search.fit(X_train,Y_train)
# Fitting the grid_search to the data
```

```
Fitting 10 folds for each of 75 candidates, totalling 750 fits
```

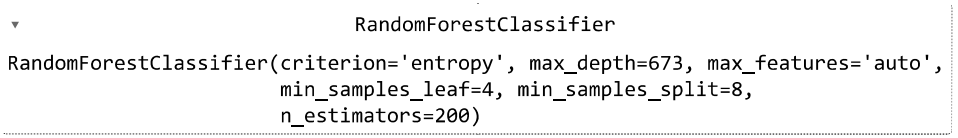



```
grid_search.best_estimator_
# best parameters of grid search
```




```
best_grid=grid_search.best_estimator_
# saving the parameters in best_grid
```

```
best_grid
# Displaying best_grid
```




```
y_pred=best_grid.predict(X_test)
print(confusion_matrix(Y_test,y_pred))
print("Accuracy Score {}".format(accuracy_score(Y_test,y_pred)))

# Wrong Predictions made.
print((Y_test !=y_pred).sum(),'/',((Y_test == y_pred).sum()+(Y_test != y_pred).sum()))
print('-'*50)

# Kappa Score
print('KappaScore is: ', metrics.cohen_kappa_score(Y_test,y_pred))
# Predicting the grid search cv on random forest model
```

```
[[1563   53]
 [ 190  194]]
Accuracy Score 0.8785
243 / 2000
--------------------------------------------------
KappaScore is:  0.5467703188647997
```

The decision tree model showed us overfitting problem.

Hence the randomised search cv on random forest classifier gave us better accuracy which is 87 percent and wrong predictions made by the model are 243/2000 and grid search cv gave us 87 percent accuracy and wrong predictions are 246/2000.

Pickling the model.

```
import pickle
```

```
with open('model.pkl', 'wb') as f:
    pickle.dump(best_grid, f)
```

```
with open('model.pkl', 'rb') as f:
    loaded_model = pickle.load(f)
```

```
import numpy as np
Xnew = np.array([[619,0,0,42,2,0.00,1,1,1,101348]])
loaded_model.predict(Xnew)
```

```
array([1], dtype=int64)
```