Churn Prediction

A Machine Learning Model That Can Predict Customers Who Will Leave The Company

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.model_selection import cross_val_predict
from sklearn.metrics import confusion_matrix, classification_report, f1_score, precision_score, recall_score, roc_auc_score, roc_curve
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC


import io
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.metrics import accuracy_score,recall_score
from xgboost import XGBClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score, GridSearchCV
from google.colab import files
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=UserWarning)
%config InlineBackend.figure_format = 'retina'
pd.set_option('display.max_columns', None); pd.set_option('display.max_rows', None);
```

```
uploaded = files.upload()
```

Choose Files churn.csv

- **churn.csv**(text/csv) - 684858 bytes, last modified: 7/30/2023 - 100% done

```
Saving churn.csv to churn (2).csv
```

```
df = pd.read_csv("churn.csv", index_col=0)
```

Part 1/3 --> EDA

```
df.head()
```

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

```
df.shape
```

```
(10000, 13)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 1 to 10000
Data columns (total 13 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   CustomerId       10000 non-null  int64  
 1   Surname          10000 non-null  object 
 2   CreditScore      10000 non-null  int64  
 3   Geography        10000 non-null  object 
 4   Gender           10000 non-null  object 
 5   Age              10000 non-null  int64  
 6   Tenure           10000 non-null  int64  
 7   Balance          10000 non-null  float64
 8   NumOfProducts    10000 non-null  int64  
 9   HasCrCard        10000 non-null  int64  
 10  IsActiveMember   10000 non-null  int64  
 11  EstimatedSalary  10000 non-null  float64
 12  Exited           10000 non-null  int64  
dtypes: float64(2), int64(8), object(3)
memory usage: 1.1+ MB
```

```
df.describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard
count	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000
mean	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550
std	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584
min	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000
5%	1.557882e+07	489.000000	25.000000	1.000000	0.000000	1.000000	0.00000
25%	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000
50%	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000
75%	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000
90%	1.579083e+07	778.000000	53.000000	9.000000	149244.792000	2.000000	1.00000
95%	1.580303e+07	812.000000	60.000000	9.000000	162711.669000	2.000000	1.00000
99%	1.581311e+07	850.000000	72.000000	10.000000	185967.985400	3.000000	1.00000
max	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000

```
categorical_variables = [col for col in df.columns if col in "O"
                        or df[col].nunique() <=11
                        and col not in "Exited"]

categorical_variables
```

```
['Geography',
 'Gender',
 'Tenure',
 'NumOfProducts',
 'HasCrCard',
 'IsActiveMember']
```

```
numeric_variables = [col for col in df.columns if df[col].dtype != "object"
                        and df[col].nunique() >11
                        and col not in "CustomerId"]
numeric_variables
```

```
['CreditScore', 'Age', 'Balance', 'EstimatedSalary']
```

```
df["Exited"].value_counts()
```

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

```
churn = df.loc[df["Exited"]==1]
```

```
not_churn = df.loc[df["Exited"]==0]
```

```
not_churn["Tenure"].value_counts().sort_values()
```

```
0     318
10    389
6     771
9     771
4     786
3     796
1     803
5     803
8     828
2     847
7     851
Name: Tenure, dtype: int64
```

```
churn["Tenure"].value_counts().sort_values()
```

```
0      95
10    101
7     177
6     196
8     197
2     201
4     203
5     209
3     213
9     213
1     232
Name: Tenure, dtype: int64
```

```
not_churn["NumOfProducts"].value_counts().sort_values()
```

```
3      46
1    3675
2    4242
Name: NumOfProducts, dtype: int64
```

```
churn["NumOfProducts"].value_counts().sort_values()
```

```
4      60
3     220
2     348
1    1409
Name: NumOfProducts, dtype: int64
```

HasCrCard

```
# examining the HasCrCard of the not_churn group
not_churn["HasCrCard"].value_counts()
```

```
1    5631
0    2332
Name: HasCrCard, dtype: int64
```

```
# examining the HasCrCard of the churn group
churn["HasCrCard"].value_counts()
```

```
1    1424
0     613
Name: HasCrCard, dtype: int64
```

```
# examining the IsActiveMember of the not_churn group
not_churn["IsActiveMember"].value_counts()
```

```
1    4416
0    3547
Name: IsActiveMember, dtype: int64
```

```
# examining the IsActiveMember of the churn group
churn["IsActiveMember"].value_counts()
```

```
0    1302
1     735
Name: IsActiveMember, dtype: int64
```

```
# Frequency of not_churn group according to Geography
not_churn.Geography.value_counts().sort_values()
```

```
Germany    1695
Spain      2064
France     4204
Name: Geography, dtype: int64
```

```
# Frequency of churn group according to Geography
churn.Geography.value_counts().sort_values()
```

```
Spain      413
France     810
Germany    814
Name: Geography, dtype: int64
```

```
# Frequency of not_churn group according to Gender
not_churn.Gender.value_counts()
```

```
Male      4559
Female    3404
Name: Gender, dtype: int64
```

```
# Frequency of churn group according to Gender
churn.Gender.value_counts()
```

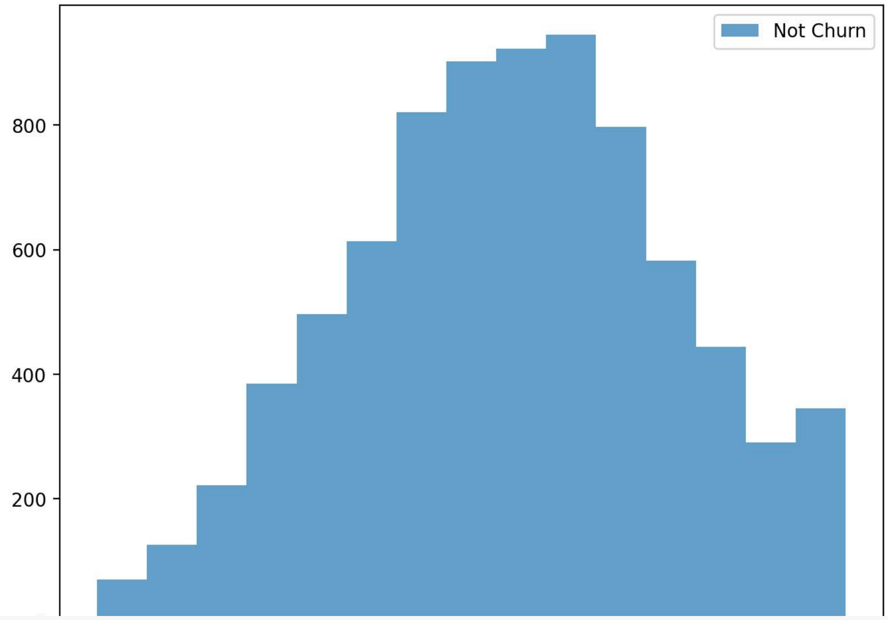
```
Female    1139
Male       898
Name: Gender, dtype: int64
```

CreditScore

```
# Let's examine the credit score of the not_churn group
not_churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count    7963.000000
mean      651.853196
std        95.653837
min       405.000000
5%        492.000000
25%       585.000000
50%       653.000000
75%       718.000000
90%       778.000000
95%       812.000000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

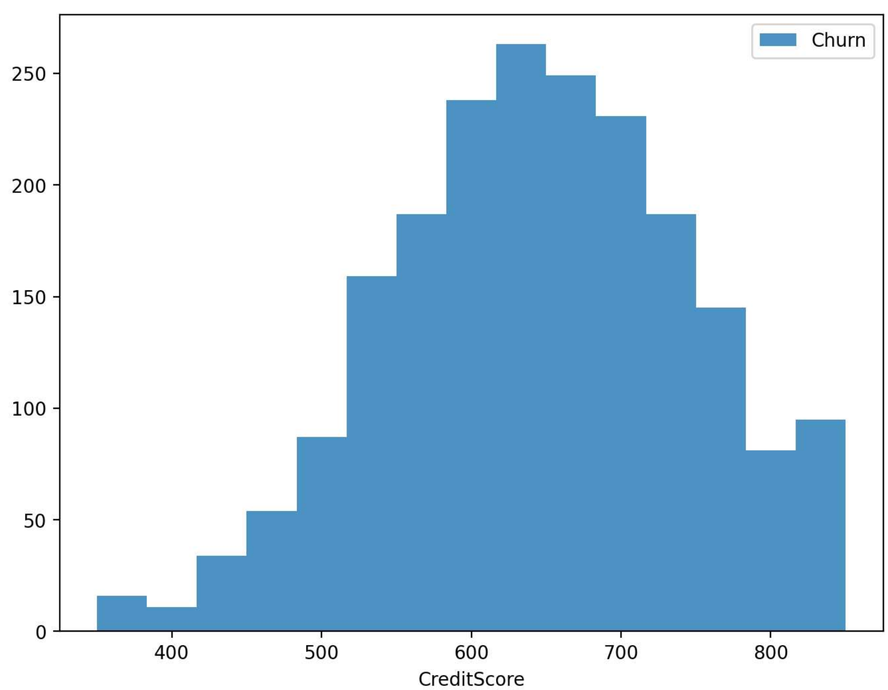
```
# distribution of the Credit Score for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(not_churn["CreditScore"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# Let's examine the credit score of the churn group
churn["CreditScore"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count    2037.000000
mean      645.351497
std       100.321503
min       350.000000
5%        479.000000
25%       578.000000
50%       646.000000
75%       716.000000
90%       776.400000
95%       812.200000
99%       850.000000
max       850.000000
Name: CreditScore, dtype: float64
```

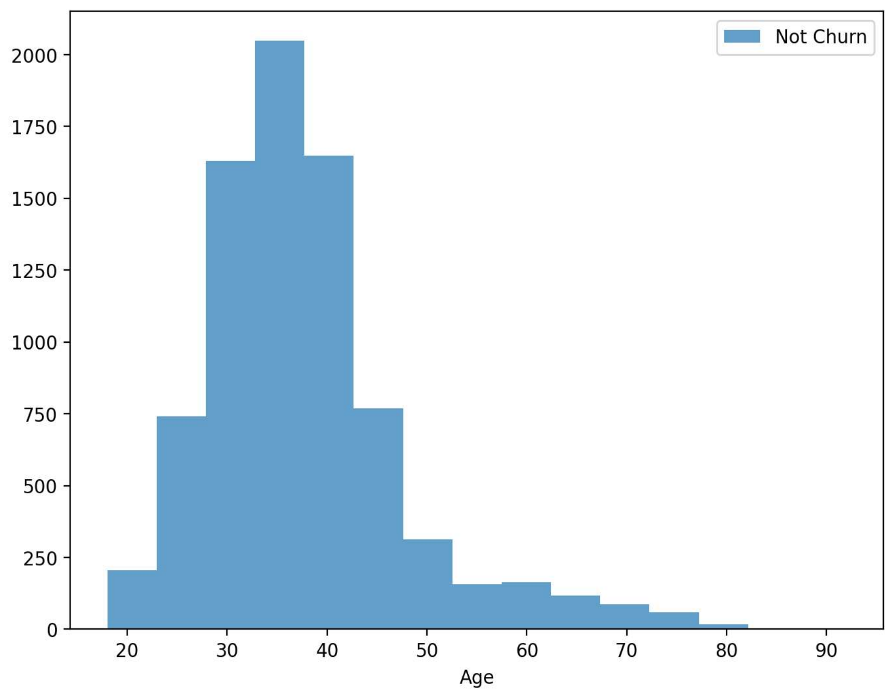
```
# distribution of the Credit Score for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('CreditScore')
pyplot.hist(churn["CreditScore"],bins=15, alpha=0.8, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```



```
# examining the age of the not_churn group
not_churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count    7963.000000
mean       37.408389
std        10.125363
min        18.000000
5%         24.000000
25%        31.000000
50%        36.000000
75%        41.000000
90%        49.000000
95%        59.000000
99%        73.000000
max        92.000000
Name: Age, dtype: float64
```

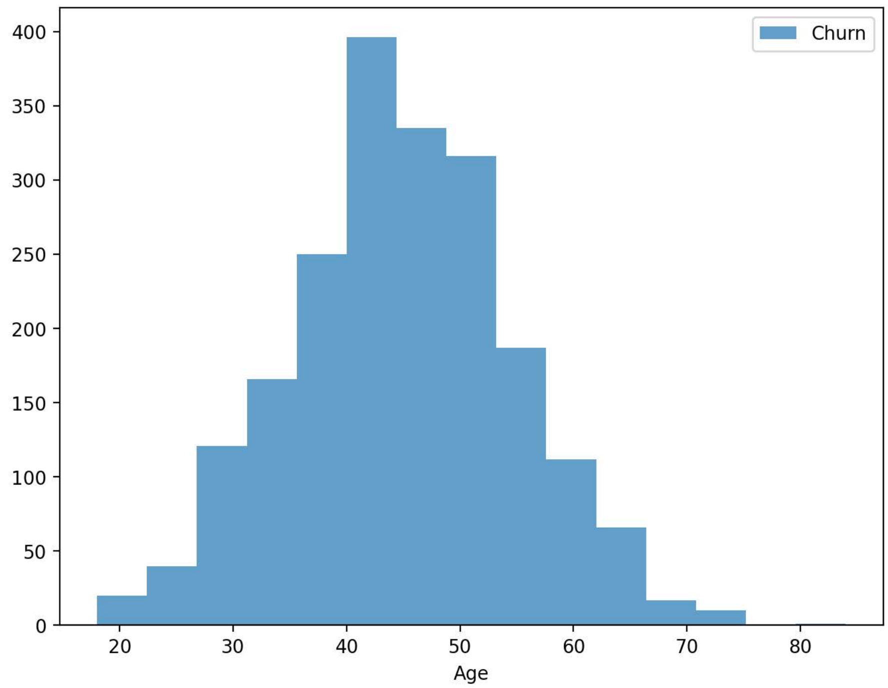
```
# distribution of the Age for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(not_churn["Age"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# examine the age of the churn group
churn["Age"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count    2037.000000
mean       44.837997
std         9.761562
min        18.000000
5%         29.000000
25%        38.000000
50%        45.000000
75%        51.000000
90%        58.000000
95%        61.000000
99%        68.000000
max        84.000000
Name: Age, dtype: float64
```

```
# distribution of the Age for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Age')
pyplot.hist(churn["Age"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# examining the Balance of the not_churn group
not_churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

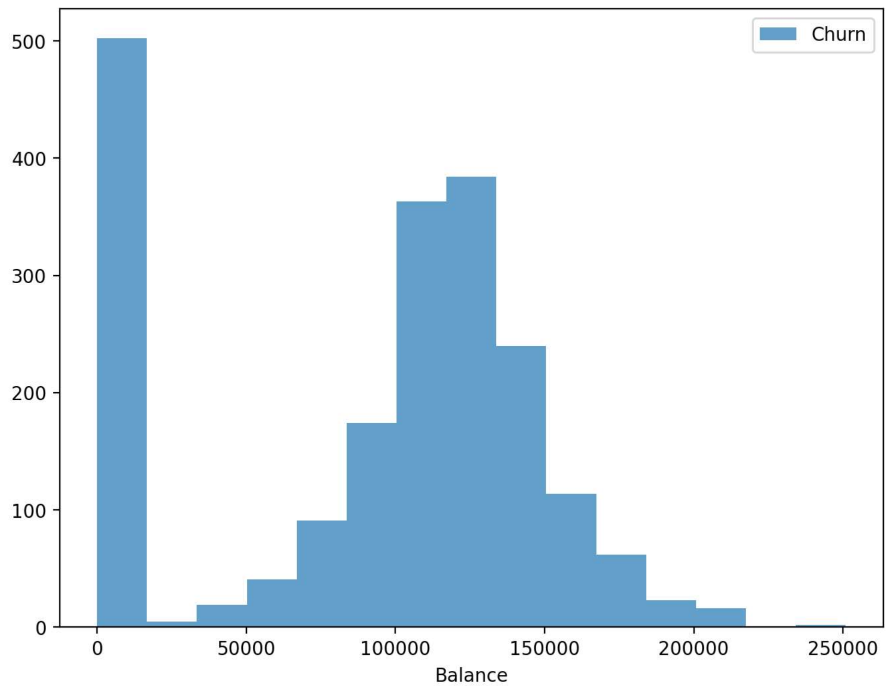
```
count      7963.000000
mean      72745.296779
std       62848.040701
min           0.000000
5%            0.000000
25%           0.000000
50%       92072.680000
75%      126410.280000
90%      148730.298000
95%      161592.595000
99%      183753.906200
max      221532.800000
Name: Balance, dtype: float64
```

```
# distribution of the Balance for not_churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(not_churn["Balance"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```


```
# examining the Balance of the churn group
churn["Balance"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count      2037.000000
mean      91108.539337
std       58360.794816
min           0.000000
5%            0.000000
25%       38340.020000
50%      109349.290000
75%      131433.330000
90%      152080.618000
95%      167698.240000
99%      197355.288400
max      250898.090000
Name: Balance, dtype: float64
```

```
# distribution of the Balance for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('Balance')
pyplot.hist(churn["Balance"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```

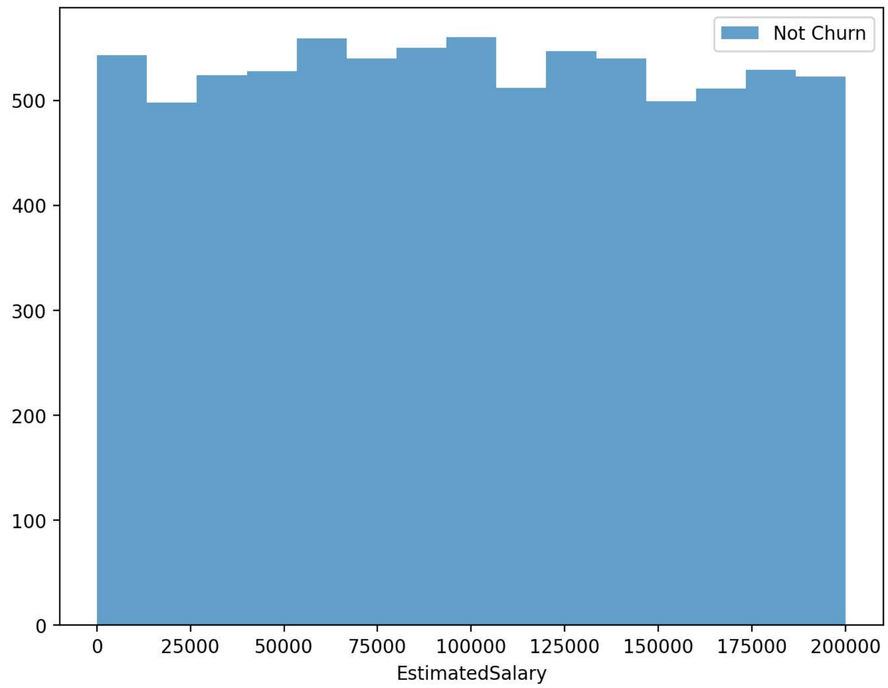



```
# examining the EstimatedSalary of the not_churn group
not_churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

```
count      7963.000000
mean      99738.391772
std       57405.586966
min          90.070000
5%         9773.542000
25%       50783.490000
50%       99645.040000
75%      148609.955000
90%      179453.212000
95%      190107.557000
99%      198131.465200
max      199992.480000
Name: EstimatedSalary, dtype: float64
```

```
# distribution of the Balance for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
```

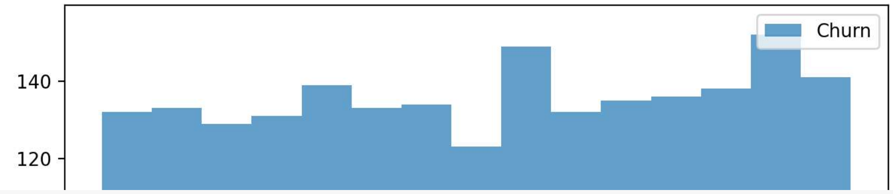
```
pyplot.hist(not_churn["EstimatedSalary"],bins=15, alpha=0.7, label='Not Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# examining the EstimatedSalary of the churn group
churn["EstimatedSalary"].describe([0.05,0.25,0.50,0.75,0.90,0.95,0.99])
```

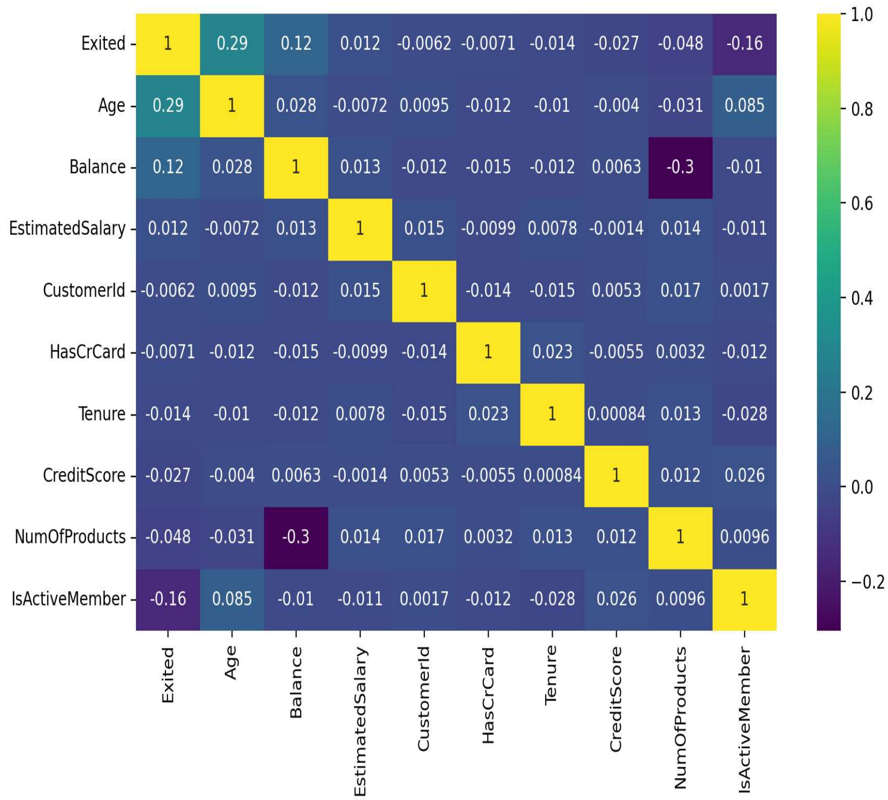
```
count      2037.000000
mean     101465.677531
std       57912.418071
min          11.580000
5%        10030.760000
25%       51907.720000
50%      102460.840000
75%      152422.910000
90%      180169.390000
95%      190328.982000
99%      197717.297600
max      199808.100000
Name: EstimatedSalary, dtype: float64
```

```
# distribution of the EstimatedSalary for churn
pyplot.figure(figsize=(8,6))
pyplot.xlabel('EstimatedSalary')
pyplot.hist(churn["EstimatedSalary"],bins=15, alpha=0.7, label='Churn')
pyplot.legend(loc='upper right')
pyplot.show()
```




```
# Exited correlation matrix
k = 10 #number of variables for heatmap
cols = df.corr().nlargest(k, 'Exited')['Exited'].index
cm = df[cols].corr()
plt.figure(figsize=(10,6))
sns.heatmap(cm, annot=True, cmap = 'viridis')
```

```
<Axes: >
```



Part 2/3-> Data Preprocessing

```
# # Missing Observation Analysis
df.isnull().sum()
```

```
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
```

```
# To determine the threshold value for outliers
def outlier_thresholds(dataframe, variable, low_quantile=0.05, up_quantile=0.95):
    quantile_one = dataframe[variable].quantile(low_quantile)
    quantile_three = dataframe[variable].quantile(up_quantile)
    interquantile_range = quantile_three - quantile_one
```

```
    up_limit = quantile_three + 1.5 * interquantile_range
    low_limit = quantile_one - 1.5 * interquantile_range
    return low_limit, up_limit
```

```
# Are there any outliers in the variables
def has_outliers(dataframe, numeric_columns, plot=False):
   # variable_names = []
    for col in numeric_columns:
        low_limit, up_limit = outlier_thresholds(dataframe, col)
        if dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].any(axis=None):
            number_of_outliers = dataframe[(dataframe[col] > up_limit) | (dataframe[col] < low_limit)].shape[0]
            print(col, " : ", number_of_outliers, "outliers")
            #variable_names.append(col)
            if plot:
                sns.boxplot(x=dataframe[col])
                plt.show()
    #return variable_names
```

```
# There is no outlier
for var in numeric_variables:
    print(var, "has " , has_outliers(df, [var]),  "Outliers")
```

```
CreditScore has  None Outliers
Age has  None Outliers
Balance has  None Outliers
EstimatedSalary has  None Outliers
```

```
# we standardize tenure with age
df["NewTenure"] = df["Tenure"]/df["Age"]
df["NewCreditsScore"] = pd.qcut(df['CreditScore'], 6, labels = [1, 2, 3, 4, 5, 6])
df["NewAgeScore"] = pd.qcut(df['Age'], 8, labels = [1, 2, 3, 4, 5, 6, 7, 8])
df["NewBalanceScore"] = pd.qcut(df['Balance'].rank(method="first"), 5, labels = [1, 2, 3, 4, 5])
df["NewEstSalaryScore"] = pd.qcut(df['EstimatedSalary'], 10, labels = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10])
```

```
df.head()
```

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

```
# Variables to apply one hot encoding
list = ["Gender", "Geography"]
df = pd.get_dummies(df, columns =list, drop_first = True)
```

```
df.head()
```

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

```
# Removing variables that will not affect the dependent variable
df = df.drop(["CustomerId","Surname"], axis = 1)
```

```
# Scale features using statistics that are robust to outliers.
def robust_scaler(variable):
    var_median = variable.median()
    quartile1 = variable.quantile(0.25)
    quartile3 = variable.quantile(0.75)
    interquantile_range = quartile3 - quartile1
    if int(interquantile_range) == 0:
        quartile1 = variable.quantile(0.05)
        quartile3 = variable.quantile(0.95)
        interquantile_range = quartile3 - quartile1
        if int(interquantile_range) == 0:
            quartile1 = variable.quantile(0.01)
            quartile3 = variable.quantile(0.99)
            interquantile_range = quartile3 - quartile1
            z = (variable - var_median) / interquantile_range
            return round(z, 3)

        z = (variable - var_median) / interquantile_range
        return round(z, 3)
    else:
        z = (variable - var_median) / interquantile_range
    return round(z, 3)
```

```
new_cols_ohe = ["Gender_Male","Geography_Germany","Geography_Spain"]
like_num = [col for col in df.columns if df[col].dtypes != 'O' and len(df[col].value_counts()) <= 10]
cols_need_scale = [col for col in df.columns if col not in new_cols_ohe
                   and col not in "Exited"
                   and col not in like_num]

for col in cols_need_scale:
    df[col] = robust_scaler(df[col])
```

```
df.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember
RowNumber							
1	-0.246	0.417	-0.75	-0.761	1	1	1
2	-0.328	0.333	-1.00	-0.105	1	0	1
3	-1.119	0.417	0.75	0.489	3	1	0
4	0.351	0.167	-1.00	-0.761	2	0	0
5	1.478	0.500	-0.75	0.222	1	1	1

Part 3/3 Model

```
# GB Confusion Matrix
model_GB = GradientBoostingClassifier(random_state=12345)
model_GB.fit(X_train, y_train)
y_pred = model_GB.predict(X_test)
conf_mat = confusion_matrix(y_pred,y_test)
conf_mat
```

```
array([[1520,  230],
       [  53,  197]])
```

```
print("True Positive : ", conf_mat[1, 1])
print("True Negative : ", conf_mat[0, 0])
print("False Positive: ", conf_mat[0, 1])
print("False Negative: ", conf_mat[1, 0])
```

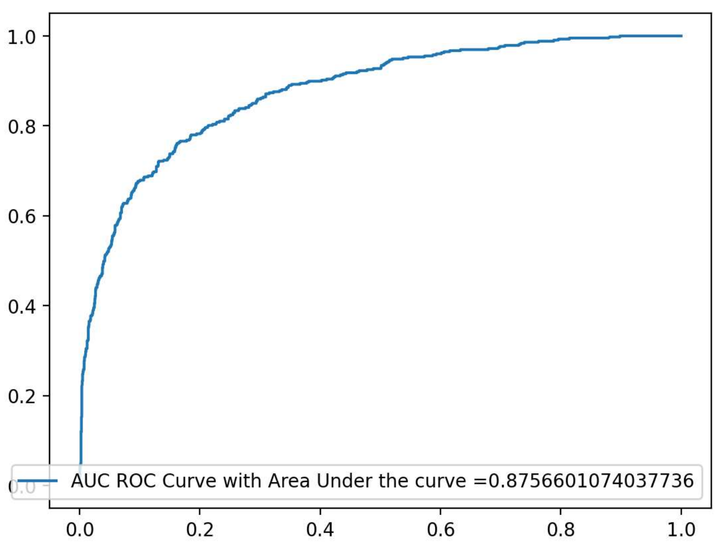
```
True Positive :  197
True Negative :  1520
False Positive:  230
False Negative:  53
```

```
# Classification Report for XGB Model
print(classification_report(model_GB.predict(X_test),y_test))
```

```
              precision    recall  f1-score   support

           0       0.97      0.87      0.91      1750
           1       0.46      0.79      0.58       250

    accuracy                           0.86      2000
   macro avg       0.71      0.83      0.75      2000
weighted avg       0.90      0.86      0.87      2000
```

```
# Auc Roc Curve
def generate_auc_roc_curve(clf, X_test):
    y_pred_proba = clf.predict_proba(X_test)[:, 1]
    fpr, tpr, thresholds = roc_curve(y_test,  y_pred_proba)
    auc = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr,tpr,label="AUC ROC Curve with Area Under the curve ="+str(auc))
    plt.legend(loc=4)
    plt.show()
    pass
```

```
generate_auc_roc_curve(model_GB, X_test)
```

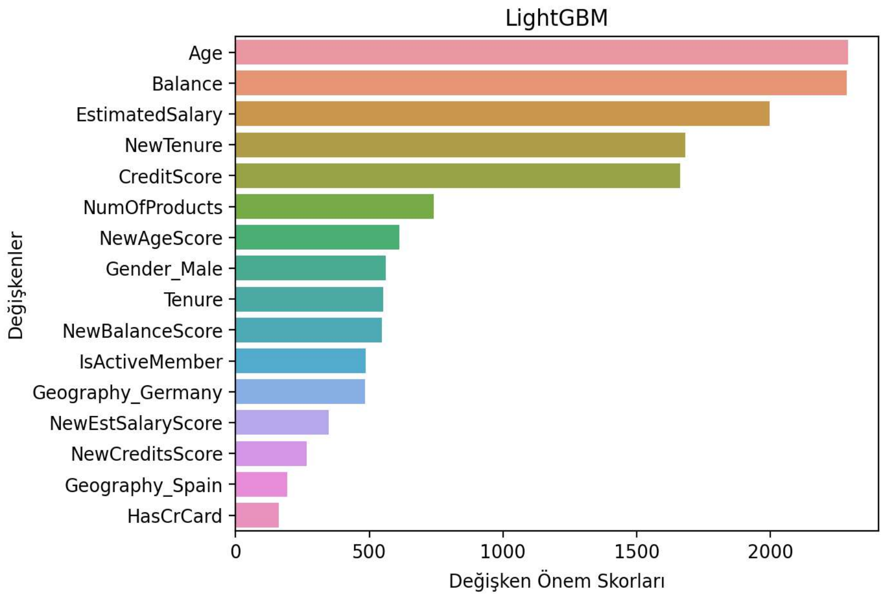


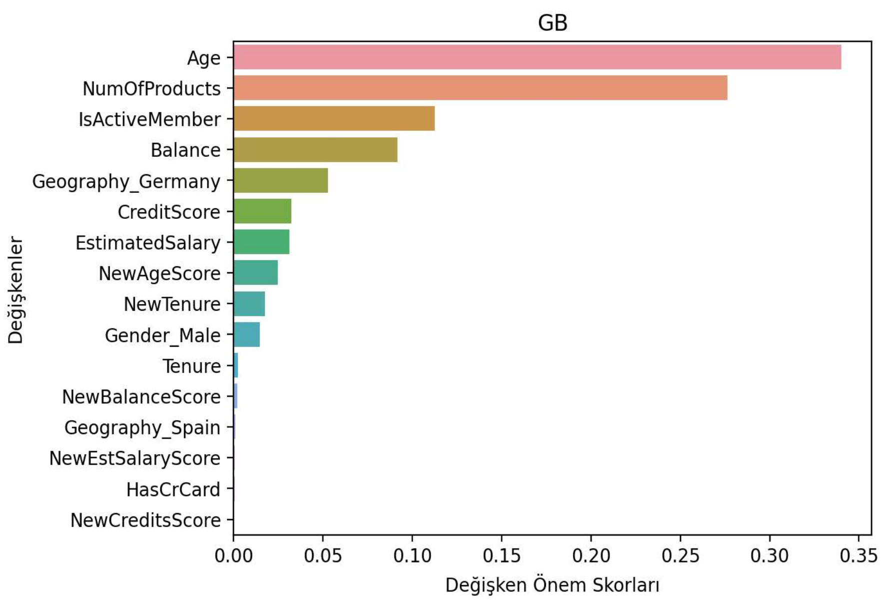
Part 4 EXtra- Model Tuning

```
# LightGBM:
lgb_model = LGBMClassifier()
# Model Tuning
lgbm_params = {'colsample_bytree': 0.5,
 'learning_rate': 0.01,
 'max_depth': 6,
 'n_estimators': 500}

lgbm_tuned = LGBMClassifier(**lgbm_params).fit(X, y)
```

```
#Let's choose the highest 4 models
# GBM
gbm_model = GradientBoostingClassifier()
# Model Tuning
gbm_params = {'learning_rate': 0.1, 'max_depth': 3, 'n_estimators': 200, 'subsample': 1}
gbm_tuned = GradientBoostingClassifier(**gbm_params).fit(X,y)
```

```
for name, model in models:
        base = model.fit(X_train,y_train)
        y_pred = base.predict(X_test)
        acc_score = accuracy_score(y_test, y_pred)
        feature_imp = pd.Series(base.feature_importances_,
                        index=X.columns).sort_values(ascending=False)

        sns.barplot(x=feature_imp, y=feature_imp.index)
        plt.xlabel('Değişken Önem Skorları')
        plt.ylabel('Değişkenler')
        plt.title(name)
        plt.show()
```





5s completed at 3:41 AM