```
import numpy as np
import pandas as pd

# For visualization
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
pd.options.display.max_rows = None
pd.options.display.max_columns = None
```

```
df = pd.read_csv('Churn_Modelling.csv')
df.shape
```

```
(10000, 14)
```

```
# Check columns list and missing values
df.isnull().sum()
```

```
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.nunique()
```

```
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 = df.drop(["RowNumber", "CustomerId", "Surname"], axis = 1)
# Review the top rows of what is left of the data frame
df.head()
```

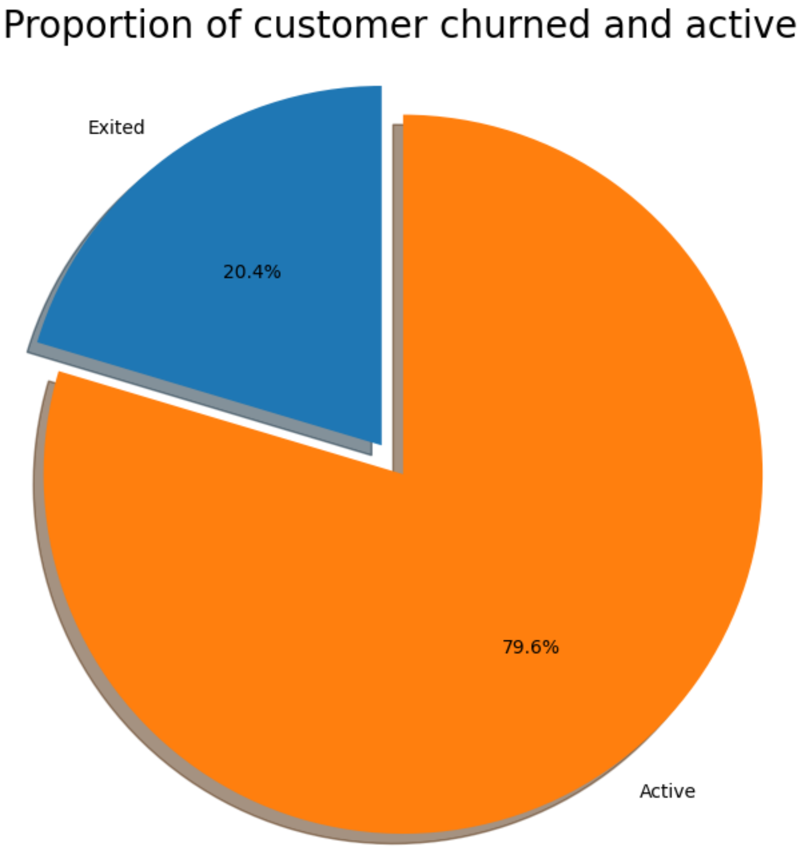
	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	France	Female	42	2	0.00	1	1	1	101348.88	1
1	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
2	502	France	Female	42	8	159660.80	3	1	0	113931.57	1
3	699	France	Female	39	1	0.00	2	0	0	93826.63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

```
# Check variable data types
df.dtypes
```

```
CreditScore          int64
Geography           object
Gender              object
Age                  int64
Tenure               int64
Balance            float64
NumOfProducts        int64
HasCrCard            int64
IsActiveMember       int64
EstimatedSalary    float64
```

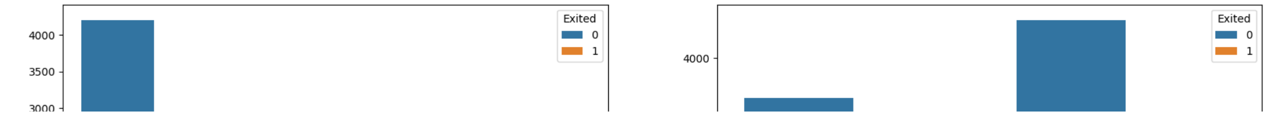
```
Exited              int64
dtype: object
```

```
labels = 'Exited', 'Active'
sizes = [df.Exited[df['Exited']==1].count(), df.Exited[df['Exited']==0].count()]
explode = (0, 0.1)
fig1, ax1 = plt.subplots(figsize=(10, 8))
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')
plt.title("Proportion of customer churned and active", size = 20)
plt.show()
```



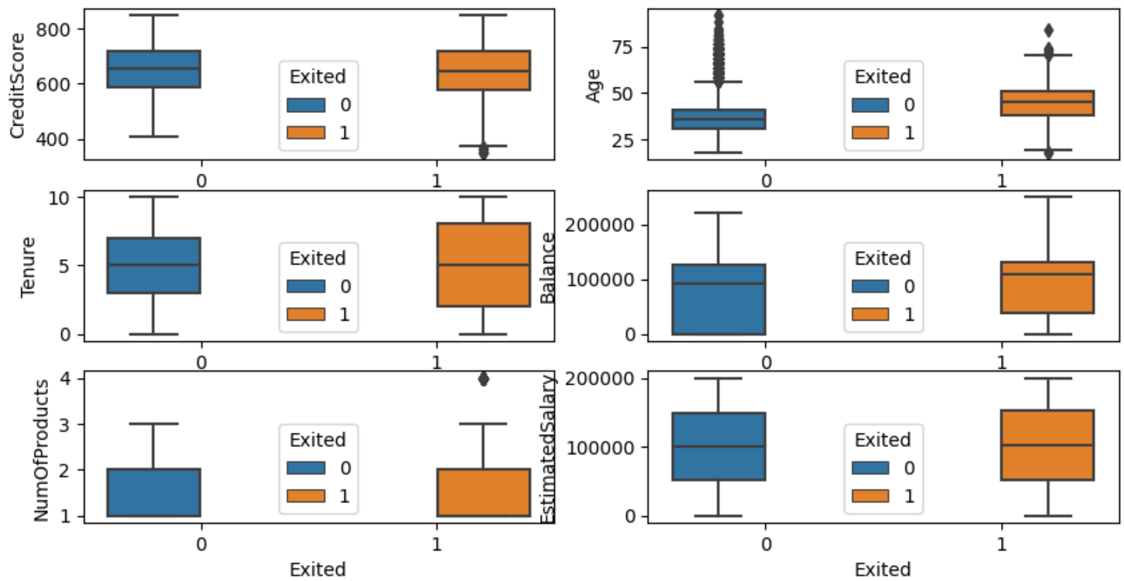

```
fig, axarr = plt.subplots(2, 2, figsize=(20, 12))
sns.countplot(x='Geography', hue = 'Exited',data = df, ax=axarr[0][0])
sns.countplot(x='Gender', hue = 'Exited',data = df, ax=axarr[0][1])
sns.countplot(x='HasCrCard', hue = 'Exited',data = df, ax=axarr[1][0])
sns.countplot(x='IsActiveMember', hue = 'Exited',data = df, ax=axarr[1][1])
```

<Axes: xlabel='IsActiveMember', ylabel='count'>




```
fig, axarr = plt.subplots(3, 2, figsize=(10, 5))
sns.boxplot(y='CreditScore',x = 'Exited', hue = 'Exited',data = df, ax=axarr[0][0])
sns.boxplot(y='Age',x = 'Exited', hue = 'Exited',data = df , ax=axarr[0][1])
sns.boxplot(y='Tenure',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][0])
sns.boxplot(y='Balance',x = 'Exited', hue = 'Exited',data = df, ax=axarr[1][1])
sns.boxplot(y='NumOfProducts',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][0])
sns.boxplot(y='EstimatedSalary',x = 'Exited', hue = 'Exited',data = df, ax=axarr[2][1])
```

<Axes: xlabel='Exited', ylabel='EstimatedSalary'>

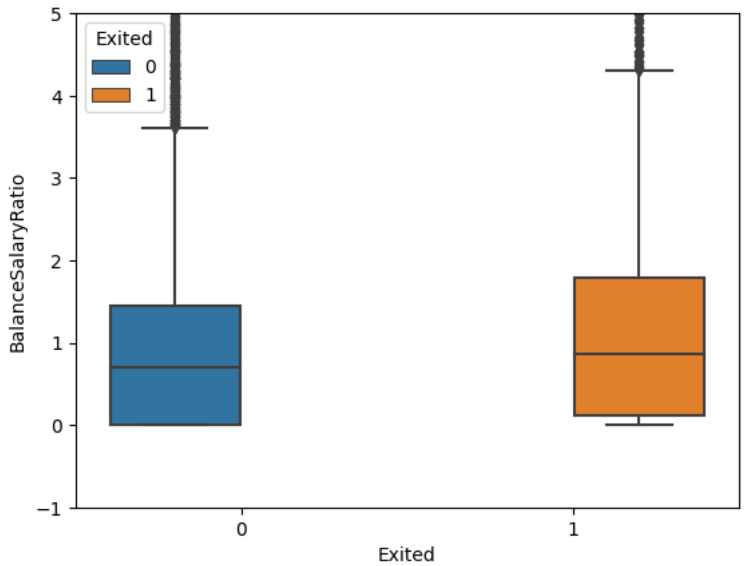



```
# Split Train, test data
df_train = df.sample(frac=0.8,random_state=200)
df_test = df.drop(df_train.index)
print(len(df_train))
print(len(df_test))
```

```
8000
2000
```

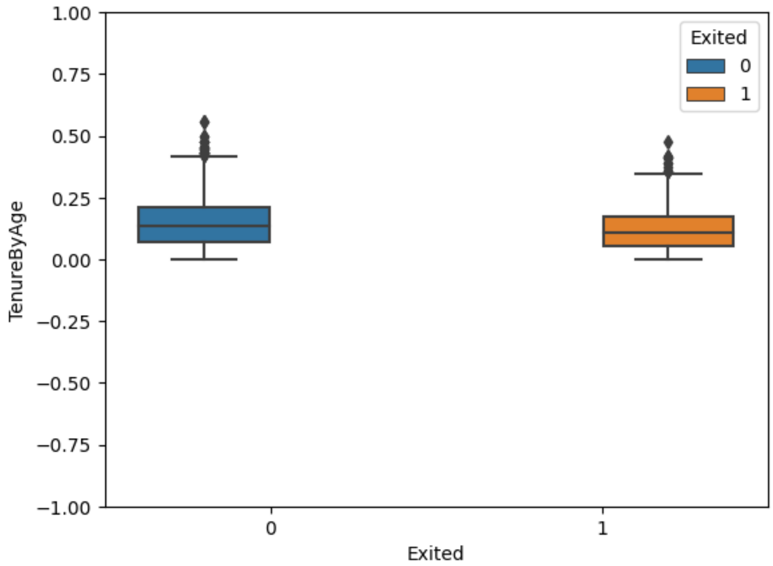
```
df_train['BalanceSalaryRatio'] = df_train.Balance/df_train.EstimatedSalary
sns.boxplot(y='BalanceSalaryRatio',x = 'Exited', hue = 'Exited',data = df_train)
plt.ylim(-1, 5)
```

(-1.0, 5.0)




```
# Given that tenure is a 'function' of age, we introduce a variable aiming to standardize tenure over age:
df_train['TenureByAge'] = df_train.Tenure/(df_train.Age)
```

```
sns.boxplot(y='TenureByAge',x = 'Exited', hue = 'Exited',data = df_train)
plt.ylim(-1, 1)
plt.show()
```




```
df_train['CreditScoreGivenAge'] = df_train.CreditScore/(df_train.Age)
```

```
# Resulting Data Frame
df_train.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exi
8159	461	Spain	Female	25	6	0.00	2	1	1	15306.29	
6332	619	France	Female	35	4	90413.12	1	1	1	20555.21	
8895	699	France	Female	40	8	122038.34	1	1	0	102085.35	
5351	558	Germany	Male	41	2	124227.14	1	1	1	111184.67	
4314	638	France	Male	34	5	133501.36	1	0	1	155643.04	

```
continuous_vars = ['CreditScore',  'Age', 'Tenure', 'Balance','NumOfProducts', 'EstimatedSalary',
                   'TenureByAge','CreditScoreGivenAge']
```

```
cat_vars = ['HasCrCard', 'IsActiveMember','Geography', 'Gender']
```

```
df_train = df_train[['Exited'] + continuous_vars  ]
```

```
df_train.head()
```

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	TenureByAge	CreditScoreGivenAge
8159	0	461	25	6	0.00	2	15306.29	0.240000	18.440000
6332	0	619	35	4	90413.12	1	20555.21	0.114286	17.685714
8895	0	699	40	8	122038.34	1	102085.35	0.200000	17.475000
5351	0	558	41	2	124227.14	1	111184.67	0.048780	13.609756
4314	0	638	34	5	133501.36	1	155643.04	0.147059	18.764706

```
# minMax scaling the continuous variables
minVec = df_train[continuous_vars].min().copy()
maxVec = df_train[continuous_vars].max().copy()
```

```
df_train[continuous_vars] = (df_train[continuous_vars]-minVec)/(maxVec-minVec)
df_train.head()
```

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	TenureByAge	CreditScoreGivenAge
8159	0	0.222	0.094595	0.6	0.000000	0.333333	0.076118	0.432000	0.323157
6332	0	0.538	0.229730	0.4	0.360358	0.000000	0.102376	0.205714	0.305211
8895	0	0.698	0.297297	0.8	0.486406	0.000000	0.510225	0.360000	0.300198
5351	0	0.416	0.310811	0.2	0.495130	0.000000	0.555744	0.087805	0.208238
4314	0	0.576	0.216216	0.5	0.532094	0.000000	0.778145	0.264706	0.330882

```
# data prep pipeline for test data
def DfPrepPipeline(df_predict,df_train_Cols,minVec,maxVec):
    # Add new features
    df_predict['BalanceSalaryRatio'] = df_predict.Balance/df_predict.EstimatedSalary
    df_predict['TenureByAge'] = df_predict.Tenure/(df_predict.Age - 18)
    df_predict['CreditScoreGivenAge'] = df_predict.CreditScore/(df_predict.Age - 18)
    # Reorder the columns
    continuous_vars = ['CreditScore','Age','Tenure','Balance','NumOfProducts','EstimatedSalary','BalanceSalaryRatio',
                   'TenureByAge','CreditScoreGivenAge']
    cat_vars = ['HasCrCard','IsActiveMember',"Geography", "Gender"]
    df_predict = df_predict[['Exited'] + continuous_vars + cat_vars]
    # Change the 0 in categorical variables to -1
    df_predict.loc[df_predict.HasCrCard == 0, 'HasCrCard'] = -1
    df_predict.loc[df_predict.IsActiveMember == 0, 'IsActiveMember'] = -1
    # One hot encode the categorical variables
    lst = ["Geography", "Gender"]
    remove = list()
    for i in lst:
        for j in df_predict[i].unique():
            df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
        remove.append(i)
    df_predict = df_predict.drop(remove, axis=1)
    # Ensure that all one hot encoded variables that appear in the train data appear in the subsequent data
    L = list(set(df_train_Cols) - set(df_predict.columns))
    for l in L:
        df_predict[str(l)] = -1
    # MinMax scaling coontinuous variables based on min and max from the train data
    df_predict[continuous_vars] = (df_predict[continuous_vars]-minVec)/(maxVec-minVec)
    # Ensure that The variables are ordered in the same way as was ordered in the train set
    df_predict = df_predict[df_train_Cols]
    return df_predict
```

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from scipy.stats import uniform

# Fit models
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# Scoring functions
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.impute import SimpleImputer
# Function to give best model score and parameters
def best_model(model):
    print(model.best_score_)
    print(model.best_params_)
    print(model.best_estimator_)
def get_auc_scores(y_actual, method,method2):
    auc_score = roc_auc_score(y_actual, method);
    fpr_df, tpr_df, _ = roc_curve(y_actual, method2);
    return (auc_score, fpr_df, tpr_df)
```

```
# Function to give best model score and parameters
def best_model(model):
    print(model.best_score_)
    print(model.best_params_)
    print(model.best_estimator_)
```

```
def get_auc_scores(y_actual, method,method2):
    auc_score = roc_auc_score(y_actual, method);
    fpr_df, tpr_df, _ = roc_curve(y_actual, method2);
    return (auc_score, fpr_df, tpr_df)
```

```
# Fit primal logistic regression
log_primal = LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=250, multi_class='
                                penalty='l2', random_state=None, solver='lbfgs',tol=1e-05, verbose=0, warm_start=False)
log_primal.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

```
▾                              LogisticRegression
LogisticRegression(C=100, max_iter=250, multi_class='multinomial', tol=1e-05)
```

```
# Fit logistic regression with pol 2 kernel
poly2 = PolynomialFeatures(degree=2)
df_train_pol2 = poly2.fit_transform(df_train.loc[:, df_train.columns != 'Exited'])
log_pol2 = LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,intercept_scaling=1, max_iter=300, multi_class='mul
                              penalty='l2', random_state=None, solver='lbfgs',tol=0.0001, verbose=0, warm_start=False)
log_pol2.fit(df_train_pol2,df_train.Exited)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾                        LogisticRegression
LogisticRegression(C=10, max_iter=300, multi_class='multinomial')
```

```
# Fit SVM with RBF Kernel
SVM_RBF = SVC(C=100, cache_size=200, class_weight=None, coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.1, kernel='rbf', max_
              random_state=None, shrinking=True,tol=0.001, verbose=False)
SVM_RBF.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

```
▾                   SVC
SVC(C=100, gamma=0.1, probability=True)
```

```
# Fit SVM with Pol Kernel
SVM_POL = SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,  decision_function_shape='ovr', degree=2, gamma=0.1, kernel='poly',  m
              probability=True, random_state=None, shrinking=True, tol=0.001, verbose=False)
SVM_POL.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

```
▾                              SVC
SVC(C=100, degree=2, gamma=0.1, kernel='poly', probability=True)
```

```
# Fit Extreme Gradient Boost Classifier
XGB = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,colsample_bytree=1, gamma=0.01, learning_rate=0.1, max_delta_st
                    min_child_weight=5, missing=None, n_estimators=20,n_jobs=1, nthread=None, objective='binary:logistic', random_state=(
                    reg_lambda=1, scale_pos_weight=1, seed=None, silent=True, subsample=1)
XGB.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

```
[15:46:45] WARNING: ../src/learner.cc:767:
Parameters: { "silent" } are not used.
```

```
▾                                 XGBClassifier
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=None, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0.01, gpu_id=None,
              grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.1, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None, max_delta_step=0,
              max_depth=7, max_leaves=None, min_child_weight=5, missing=None,
              monotone_constraints=None, n_estimators=20, n_jobs=1,
              nthread=None, num_parallel_tree=None, predictor=None, ...)
```

```
print(classification_report(df_train.Exited, log_primal.predict(df_train.loc[:, df_train.columns != 'Exited'])))
```

```
              precision    recall  f1-score   support

           0       0.80      0.98      0.88      6353
           1       0.40      0.04      0.08      1647
```

```
    accuracy                           0.79      8000
   macro avg       0.60      0.51      0.48      8000
weighted avg       0.72      0.79      0.72      8000
```

```
print(classification_report(df_train.Exited,  log_pol2.predict(df_train_pol2)))
```

```
              precision    recall  f1-score   support

           0       0.85      0.96      0.90      6353
           1       0.72      0.34      0.46      1647

    accuracy                           0.84      8000
   macro avg       0.78      0.65      0.68      8000
weighted avg       0.82      0.84      0.81      8000
```

```
print(classification_report(df_train.Exited,  SVM_RBF.predict(df_train.loc[:, df_train.columns != 'Exited'])))
```

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.90      6353
           1       0.85      0.13      0.22      1647

    accuracy                           0.82      8000
   macro avg       0.83      0.56      0.56      8000
weighted avg       0.82      0.82      0.76      8000
```

```
print(classification_report(df_train.Exited,  SVM_POL.predict(df_train.loc[:, df_train.columns != 'Exited'])))
```

```
              precision    recall  f1-score   support

           0       0.81      0.99      0.90      6353
           1       0.86      0.13      0.22      1647

    accuracy                           0.82      8000
   macro avg       0.83      0.56      0.56      8000
weighted avg       0.82      0.82      0.76      8000
```

```
y = df_train.Exited
X = df_train.loc[:, df_train.columns != 'Exited']

# Check for missing values in X
print(X.isnull().sum())  # Check if there are any missing values in X

# Handle missing values using imputation (replace missing values with the mean)
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)

# Convert the imputed array back to a DataFrame
X_imputed = pd.DataFrame(X_imputed, columns=X.columns)

# Now make predictions using the XGBoost model
xgb_model = XGBClassifier()
xgb_model.fit(X_imputed, y)

# Make predictions and calculate AUC scores
predictions = xgb_model.predict(X_imputed)
proba_predictions = xgb_model.predict_proba(X_imputed)[:, 1]

# Calculate AUC scores and classification report
auc_score, fpr, tpr = get_auc_scores(y, predictions, proba_predictions)
print("AUC Score:", auc_score)
print("Classification Report:")
print(classification_report(y, predictions))
```

```
CreditScore            0
Age                    0
Tenure                 0
Balance                0
NumOfProducts          0
EstimatedSalary        0
TenureByAge            0
CreditScoreGivenAge    0
dtype: int64
AUC Score: 0.876442206928901
Classification Report:
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      6353
           1       0.96      0.76      0.85      1647
```

```
    accuracy                          0.94      8000
   macro avg       0.95      0.88      0.91      8000
weighted avg       0.94      0.94      0.94      8000
```

```
# Make the data transformation for test data
df_test = DfPrepPipeline(df_test,df_train.columns,minVec,maxVec)
df_test = df_test.mask(np.isinf(df_test))
df_test = df_test.dropna()
df_test.shape
```

```
<ipython-input-20-941607351b30>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
<ipython-input-20-941607351b30>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
<ipython-input-20-941607351b30>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
<ipython-input-20-941607351b30>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
<ipython-input-20-941607351b30>:20: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
(0, 9)
```

```
plt.plot([0,1], [0,1], 'k--', label = 'Random: 0.5')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC Curve')
plt.legend(loc='best')
#plt.savefig('roc_results_ratios.png')
plt.show()
```