In [5]:

```
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
print(os.getcwd())
os.chdir("C:\\Leina\\Data_sets\\TD_assignment")

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, VotingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import  confusion_matrix , plot_roc_curve, classification_report
```

```
C:\Users\dmin
```

In [6]:

```
# Read Data

train = pd.read_csv("hackathon_train_main.csv")

train.head()
```

Out[6]:

	CustomerId	CreditScore	City	Gender	Age	BranchId	Tenure	Balance	Currency(
0	15630310	719.0	Montreal	Male	30	4476	6	137072.660	
1	15605332	703.5	Ottawa	Male	34	5229	6	139722.510	
2	15691340	847.0	Toronto	Male	38	5444	6	112996.740	
3	15646408	714.5	Ottawa	Female	44	5972	7	145579.285	
4	15570486	641.0	Ottawa	Male	30	8117	8	139596.430	

In [7]:

```
train.shape
```

Out[7]:

```
(14646, 18)
```

In [8]:

```
train.columns
```

Out[8]:

```
Index(['CustomerId', 'CreditScore', 'City', 'Gender', 'Age', 'BranchId',
       'Tenure', 'Balance', 'CurrencyCode', 'PrefLanguage', 'NumOfProduct
s',
       'PrimaryAcHolder', 'HasOnlineService', 'HasCrCard', 'PrefContact',
       'IsActiveMember', 'EstimatedSalary', 'Exited'],
      dtype='object')
```

In [9]:

```
#Exploratory Data Analysis

#Below are the steps involved to understand, clean and prepare your data for building y
our predictive model:

# Variable Identification
# Univariate Analysis
# Bi-variate Analysis
# Missing values treatment
# Outlier treatment
# Variable transformation
# Variable creation
```

In [10]:

```
# Missing Data Analysis

train.isnull().sum()
```

Out[10]:

```
CustomerId          0
CreditScore         0
City                0
Gender              0
Age                 0
BranchId            0
Tenure              0
Balance             0
CurrencyCode        0
PrefLanguage        0
NumOfProducts       0
PrimaryAcHolder     0
HasOnlineService    0
HasCrCard           0
PrefContact         0
IsActiveMember      0
EstimatedSalary     0
Exited              0
dtype: int64
```

In [43]:

```
train.describe
```

Out[43]:

```
<bound method NDFrame.describe of        CustomerId  CreditScore      City  
Gender  Age  BranchId  Tenure  \
0        15630310   719.000000  Montreal    Male   30      4476       6   
1        15605332   703.500000    Ottawa    Male   34      5229       6   
2        15691340   847.000000   Toronto    Male   38      5444       6   
3        15646408   714.500000    Ottawa  Female   44      5972       7   
4        15570486   641.000000    Ottawa    Male   30      8117       8   
...           ...          ...       ...     ...  ...       ...     ...   
14641    15619190   688.437694  Montreal    Male   38      8257       6   
14642    15612892   655.830612  Montreal  Female   56      6593       5   
14643    15627870   574.784021  Montreal  Female   50      2050       4   
14644    15612446   734.837753  Montreal  Female   54      2229       4   
14645    15696527   556.430055  Montreal  Female   31      5500       6   

            Balance CurrencyCode PrefLanguage  NumOfProducts  PrimaryAcHol
der  \
0      137072.66000          USD       French              1              
0   
1      139722.51000          CAD       French              2              
0   
2      112996.74000          CAD      English              1              
0   
3      145579.28500          USD       French              2              
1   
4      139596.43000          CAD      English              1              
1   
...             ...          ...          ...            ...              
...   
14641   73489.22922          CAD       French              1              
1   
14642  118536.28140          USD       French              1              
0   
14643  121068.41540          CAD       French              1              
0   
14644  119225.82630          CAD       French              1              
0   
14645  135046.43700          CAD       French              1              
1   

       HasOnlineService  HasCrCard PrefContact  IsActiveMember  \
0                     0          1      Mobile               1   
1                     0          1       Email               1   
2                     0          1      Mobile               1   
3                     0          0      Mobile               1   
4                     0          1      Mobile               0   
...                 ...        ...         ...             ...   
14641                 1          1       Email               1   
14642                 1          1       Email               1   
14643                 1          1  Home_Phone               1   
14644                 0          1      Mobile               1   
14645                 0          1       Email               1   

       EstimatedSalary  Exited  
```

```
0        130569.83000       0
1         91335.61000       0
2        109511.43000       0
3        121477.81000       1
4        114166.59000       0
...              ...      ...
14641     24938.60806       1
14642     68263.62747       1
14643    100072.46180       1
14644     44498.79960       1
14645    178553.31250       1

[14646 rows x 18 columns]>
```

In [11]:

```
# Data Type Analysis

train.dtypes
```

Out[11]:

```
CustomerId            int64
CreditScore         float64
City                 object
Gender               object
Age                   int64
BranchId              int64
Tenure                int64
Balance             float64
CurrencyCode         object
PrefLanguage         object
NumOfProducts         int64
PrimaryAcHolder       int64
HasOnlineService      int64
HasCrCard             int64
PrefContact          object
IsActiveMember        int64
EstimatedSalary     float64
Exited                int64
dtype: object
```

In [26]:

```
# Univariate Analysis
"""At this stage, we explore variables one by one. Method to perform uni-variate analys
is will depend on whether the variable
type is categorical or continuous.
Let’s look at these methods and statistical measures for categorical and continuous var
iables individually:

Continuous Variables:- In case of continuous variables, we need to understand the centr
al tendency and spread of the variable.
These are measured using various statistical metrics such as Histogram and Bar plot
s:"""
```

Out[26]:

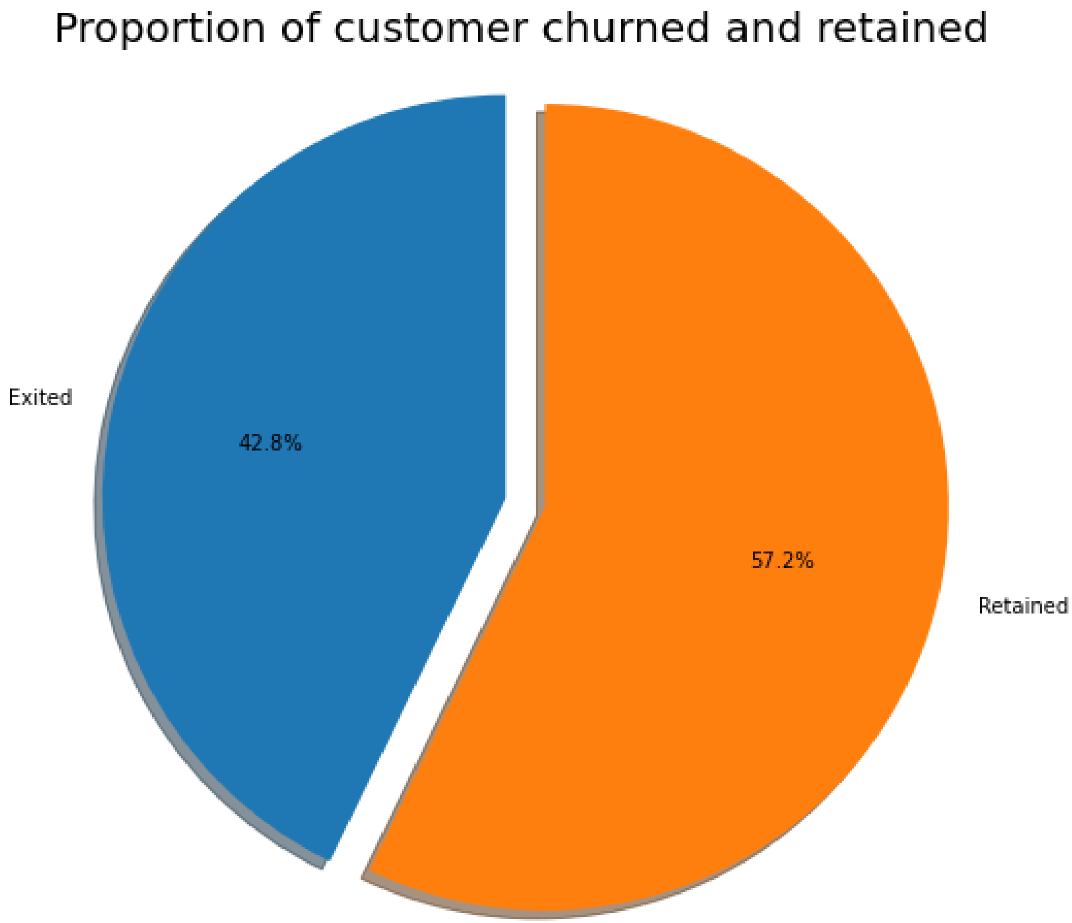
```
'At this stage, we explore variables one by one. Method to perform uni-var
iate analysis will depend on whether the variable \ntype is categorical or
continuous. \nLet’s look at these methods and statistical measures for cat
egorical and continuous variables individually:\n\nContinuous Variables:-
In case of continuous variables, we need to understand the central tendenc
y and spread of the variable.\nThese are measured using various statistica
l metrics such as Histogram and Bar plots:'
```

In [12]:

```
# remove columns which will not help in predicting churn
train.drop(["CustomerId","PrefLanguage","BranchId","CurrencyCode"], axis = 1, inplace =
True)
```

In [48]:

```
labels = 'Exited', 'Retained'
sizes = [train.Exited[train['Exited']==1].count(), train.Exited[train['Exited']==0].cou
nt()]
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 retained", size = 20)
plt.show()
```



In [ ]:

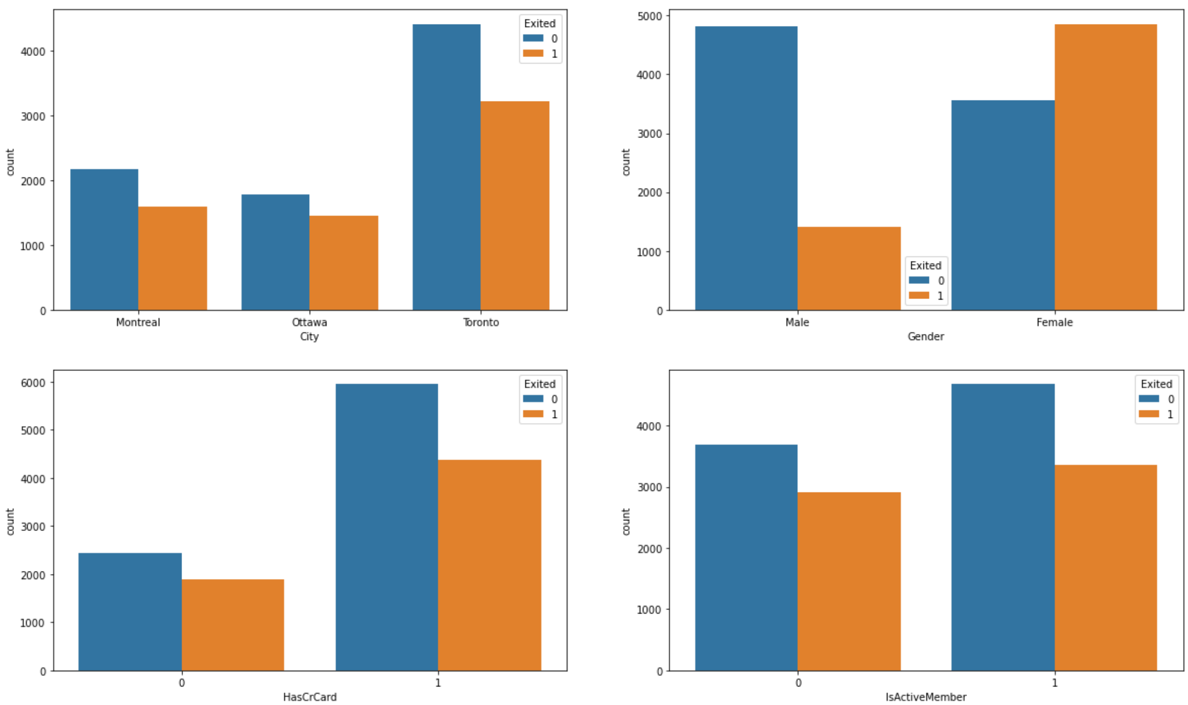
```
So about 42.8% of the customers have churned. So the baseline model could be to predict
that 42.8% of the customers will churn.
Given 42.8% is a high number.
```

In [13]:

```
# We first review the 'Status' relation with categorical variables
fig, axarr = plt.subplots(2, 2, figsize=(20, 12))
sns.countplot(x='City', hue = 'Exited',data = train, ax=axarr[0][0])
sns.countplot(x='Gender', hue = 'Exited',data = train, ax=axarr[0][1])
sns.countplot(x='HasCrCard', hue = 'Exited',data = train, ax=axarr[1][0])
sns.countplot(x='IsActiveMember', hue = 'Exited',data = train, ax=axarr[1][1])
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2e1e66c2040>
```



In [ ]:

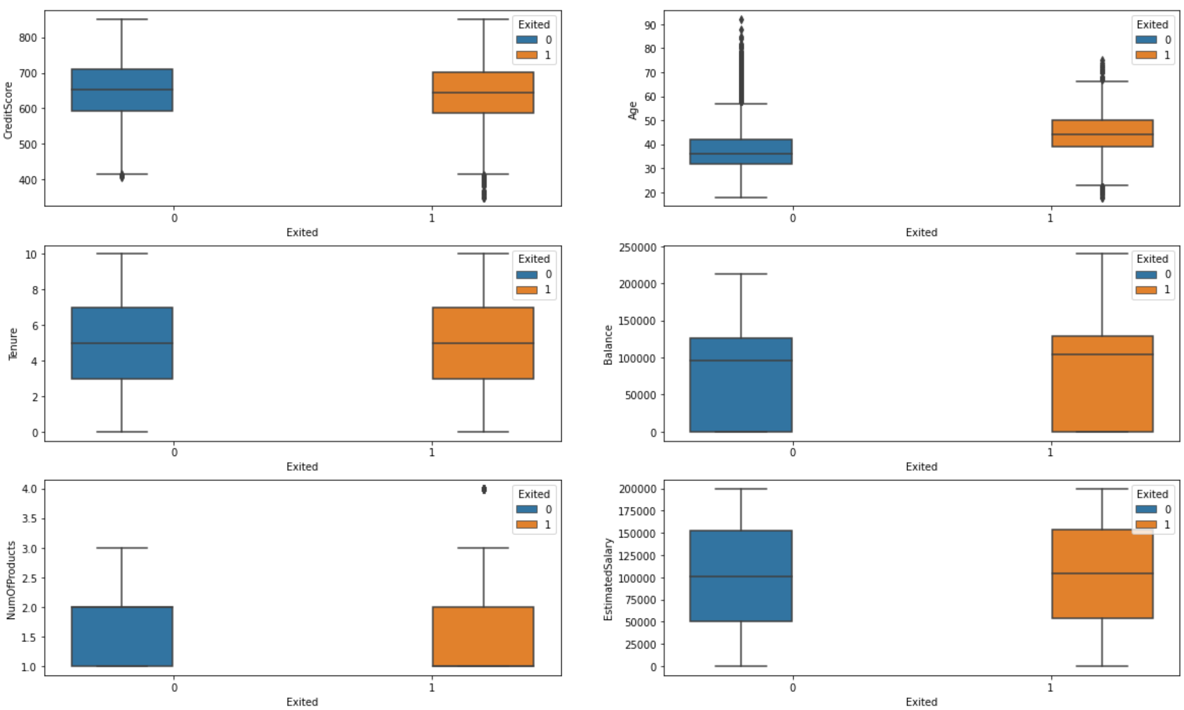
```
We note the following:

Majority of the data is from Toronto.
The proportion of male customers churning is also greater than that of female customers
Interestingly, majority of the customers that churned are those with credit cards. Give
n that majority of the customers
have credit cards could prove this to be just a coincidence.
Unsurprisingly the active members have a greater churn.
Worryingly is that the overall proportion of inactive members is quite high suggesting
that the bank may need a program
implemented to turn this group to active customers as this will definately have a posit
ive impact on the customer churn.
```

In [55]:

```
# Relations based on the continuous data attributes
fig, axarr = plt.subplots(3, 2, figsize=(20, 12))
sns.boxplot(y='CreditScore',x = 'Exited', hue = 'Exited',data = train, ax=axarr[0][0])
sns.boxplot(y='Age',x = 'Exited', hue = 'Exited',data = train , ax=axarr[0][1])
sns.boxplot(y='Tenure',x = 'Exited', hue = 'Exited',data = train, ax=axarr[1][0])
sns.boxplot(y='Balance',x = 'Exited', hue = 'Exited',data = train, ax=axarr[1][1])
sns.boxplot(y='NumOfProducts',x = 'Exited', hue = 'Exited',data = train, ax=axarr[2][0
])
sns.boxplot(y='EstimatedSalary',x = 'Exited', hue = 'Exited',data = train, ax=axarr[2][
1])
```

Out[55]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x144ac23eee0>
```



In [ ]:

```
We note the following:

There is no significant difference in the credit score distribution between retained an
d churned customers.

The older customers are churning at more than the younger ones alluding to a difference
in service preference
in the age categories. The bank may need to review their target market or review the st
rategy for retention between
the different age groups

With regard to the tenure, the clients on either extreme end (spent little time with th
e bank or a lot of time with the bank) are more likely to churn compared to those that
are of average tenure.
Worryingly, the bank is losing customers with significant bank balances which is likely
to hit their available capital for
lending.

Neither the product nor the salary has a significant effect on the likelihood to churn.
```

In [57]:

```
#Feature engineering
#We seek to add features that are likely to have an impact on the probability of churni
ng. We first split the train and test sets

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

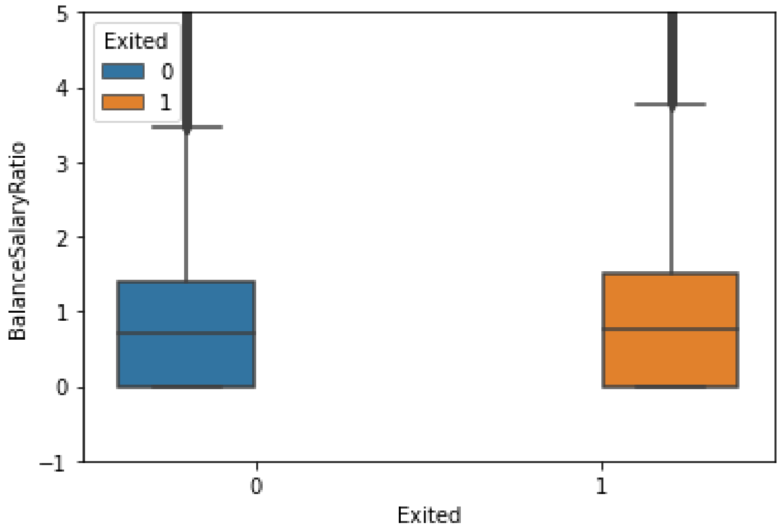
```
11717
2929
```

In [58]:

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

Out[58]:

```
(-1.0, 5.0)
```

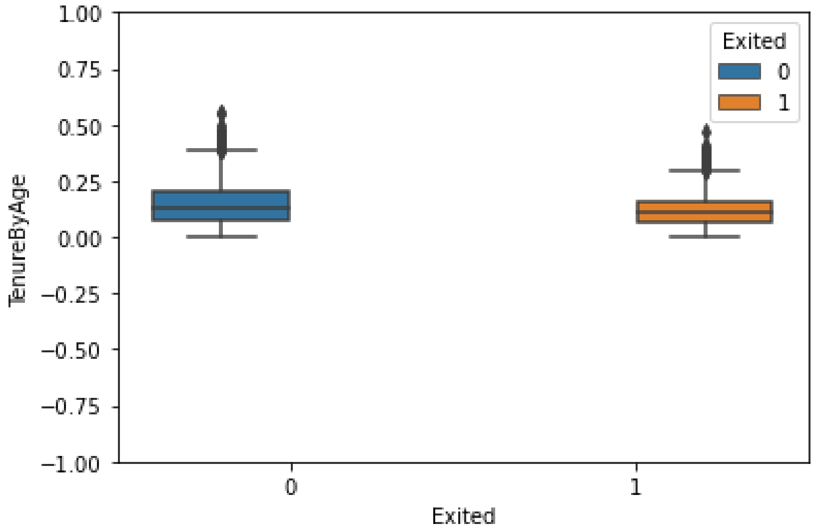


In [ ]:

```
#we have seen that the salary has little effect on the chance of a customer churning.
#However as seen above, the ratio of the bank balance and the estimated salary indicate
s that customers
#with a higher balance salary ratio churn more which would be worrying to the bank as t
his impacts their
#source of loan capital.
```

In [59]:

```
# Given that tenure is a 'function' of age, we introduce a variable aiming to standardi
ze 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()
```



In [60]:

```
'''Lastly we introduce a variable to capture credit score given age to take into accoun
t credit behaviour visavis adult life
:-)'''
df_train['CreditScoreGivenAge'] = df_train.CreditScore/(df_train.Age)
```

In [61]:

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

Out[61]:

	CreditScore	City	Gender	Age	BranchId	Tenure	Balance	CurrencyCode
6159	754.000000	Montreal	Female	19	9765	9	0.00000	CAD
7550	613.000000	Ottawa	Male	38	7908	9	67111.65000	USD
14541	726.634002	Montreal	Male	43	3076	2	97172.41241	CAD
6101	630.000000	Montreal	Female	39	1927	7	135483.17000	CAD
8456	592.000000	Toronto	Female	38	7073	8	119278.01000	USD

In [62]:

```
#Data prep for model fitting
# Arrange columns by data type for easier manipulation

continuous_vars = ['CreditScore',  'Age', 'Tenure', 'Balance','NumOfProducts', 'Estimat
edSalary', 'BalanceSalaryRatio',
                   'TenureByAge','CreditScoreGivenAge']
cat_vars = ['HasCrCard', 'IsActiveMember','City', 'Gender']
df_train = df_train[['Exited'] + continuous_vars + cat_vars]
df_train.head()
```

Out[62]:

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Ba
6159	0	754.000000	19	9	0.00000	1	189641.1100	
7550	1	613.000000	38	9	67111.65000	1	78566.6400	
14541	1	726.634002	43	2	97172.41241	1	107483.5708	
6101	1	630.000000	39	7	135483.17000	1	140881.2000	
8456	0	592.000000	38	8	119278.01000	2	19370.7300	

In [63]:

```
'''For the one hot variables, we change 0 to -1 so that the models can capture a negati
ve relation
where the attribute in inapplicable instead of 0'''
df_train.loc[df_train.HasCrCard == 0, 'HasCrCard'] = -1
df_train.loc[df_train.IsActiveMember == 0, 'IsActiveMember'] = -1
df_train.head()
```

Out[63]:

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Ba
6159	0	754.000000	19	9	0.00000	1	189641.1100	
7550	1	613.000000	38	9	67111.65000	1	78566.6400	
14541	1	726.634002	43	2	97172.41241	1	107483.5708	
6101	1	630.000000	39	7	135483.17000	1	140881.2000	
8456	0	592.000000	38	8	119278.01000	2	19370.7300	

In [64]:

```
# One hot encode the categorical variables
lst = ['City', 'Gender']
remove = list()
for i in lst:
    if (df_train[i].dtype == np.str or df_train[i].dtype == np.object):
        for j in df_train[i].unique():
            df_train[i+'_'+j] = np.where(df_train[i] == j,1,-1)
        remove.append(i)
df_train = df_train.drop(remove, axis=1)
df_train.head()
```

Out[64]:

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Ba
6159	0	754.000000	19	9	0.00000	1	189641.1100	
7550	1	613.000000	38	9	67111.65000	1	78566.6400	
14541	1	726.634002	43	2	97172.41241	1	107483.5708	
6101	1	630.000000	39	7	135483.17000	1	140881.2000	
8456	0	592.000000	38	8	119278.01000	2	19370.7300	

In [65]:

```
# 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()
```

Out[65]:

	Exited	CreditScore	Age	Tenure	Balance	NumOfProducts	EstimatedSalary	Ba
6159	0	0.808000	0.013514	0.9	0.000000	0.000000	0.948341	
7550	1	0.526000	0.270270	0.9	0.280118	0.000000	0.392856	
14541	1	0.753268	0.337838	0.2	0.405589	0.000000	0.537470	
6101	1	0.560000	0.283784	0.7	0.565495	0.000000	0.704492	
8456	0	0.484000	0.270270	0.8	0.497856	0.333333	0.096816	

In [68]:

```
# 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','Estimate
dSalary','BalanceSalaryRatio',
                    'TenureByAge','CreditScoreGivenAge']
    cat_vars = ['HasCrCard','IsActiveMember',"City", "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 = ["City", "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
```

In [86]:

```
'''Model fitting and selection
For the model fitting, I will try out the following

Logistic regression in the primal space and with different kernels
SVM in the primal and with different Kernels
Ensemble models'''

# Support functions
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 sklearn.ensemble import GradientBoostingClassifier

# 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
```

In [71]:

```
# 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)
```

In [73]:

```
# Fit primal logistic regression
log_primal = LogisticRegression(C=100, class_weight=None, dual=False,
                                fit_intercept=True,intercept_scaling=1, max_iter=250, m
ulti_class='auto',n_jobs=None,
                                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)
```

Out[73]:

```
LogisticRegression(C=100, max_iter=250, tol=1e-05)
```

In [74]:

```
# 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,i
ntercept_scaling=1, max_iter=300,
                              multi_class='auto', n_jobs=None,
                              penalty='l2', random_state=None, solver='liblinear',tol=
0.0001, verbose=0, warm_start=False)
log_pol2.fit(df_train_pol2,df_train.Exited)
```

Out[74]:

```
LogisticRegression(C=10, max_iter=300, solver='liblinear')
```

In [76]:

```
#Fit SVM with RBF Kernel
SVM_RBF = SVC(C=100, cache_size=200, class_weight=None, coef0=0.0, decision_function_sh
ape='ovr', degree=3, gamma=0.1,
              kernel='rbf', max_iter=-1, probability=True,
              random_state=None, shrinking=True,tol=0.001, verbose=False)
SVM_RBF.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

Out[76]:

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

In [78]:

```
# Fit SVM with Pol Kernel
SVM_POL = SVC(C=100, cache_size=200, class_weight=None, coef0=0.0,  decision_function_s
hape='ovr', degree=2, gamma=0.1,
              kernel='poly',  max_iter=-1,
              probability=True, random_state=None, shrinking=True, tol=0.001, verbose=F
alse)
SVM_POL.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

Out[78]:

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

In [79]:

```
# Fit Random Forest classifier
RF = RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',max_dep
th=8, max_features=6,
                            max_leaf_nodes=None,min_impurity_decrease=0.0,
                            min_impurity_split=None,min_samples_leaf=1,
                            min_samples_split=3,min_weight_fraction_leaf=0.0, n_estimat
ors=50, n_jobs=None,
                            oob_score=False, random_state=None, verbose=0,warm_start=Fa
lse)
RF.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
```

Out[79]:

```
RandomForestClassifier(max_depth=8, max_features=6, min_samples_split=3,
                       n_estimators=50)
```

In [88]:

```
# Fit Extreme Gradient Boost Classifier
XGB = GradientBoostingClassifier(n_estimators=20, learning_rate=0.5, max_features=2, ma
x_depth=2, random_state=0)
XGB.fit(df_train.loc[:, df_train.columns != 'Exited'],df_train.Exited)
#n_estimators=20, learning_rate=0.5, max_features=2, max_depth=2, random_state=0
```

Out[88]:

```
GradientBoostingClassifier(learning_rate=0.5, max_depth=2, max_features=2,
                           n_estimators=20, random_state=0)
```

In [ ]:

```
#Review best model fit accuracy : Keen interest is on the performance in predicting 1's
(Customers who churn)
```

In [100]:

```
print(confusion_matrix(df_train.Exited, log_primal.predict(df_train.loc[:, df_train.col
umns != 'Exited'])))
print(classification_report(df_train.Exited, log_primal.predict(df_train.loc[:, df_trai
n.columns != 'Exited'])))
```

```
[[5200 1545]
 [1694 3278]]
              precision    recall  f1-score   support

           0       0.75      0.77      0.76      6745
           1       0.68      0.66      0.67      4972

    accuracy                           0.72     11717
   macro avg       0.72      0.72      0.72     11717
weighted avg       0.72      0.72      0.72     11717
```

In [101]:

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

```
[[5615 1130]
 [1405 3567]]
              precision    recall  f1-score   support

           0       0.80      0.83      0.82      6745
           1       0.76      0.72      0.74      4972

    accuracy                           0.78     11717
   macro avg       0.78      0.77      0.78     11717
weighted avg       0.78      0.78      0.78     11717
```

In [102]:

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

```
[[5670 1075]
 [1297 3675]]
              precision    recall  f1-score   support

           0       0.81      0.84      0.83      6745
           1       0.77      0.74      0.76      4972

    accuracy                           0.80     11717
   macro avg       0.79      0.79      0.79     11717
weighted avg       0.80      0.80      0.80     11717
```

In [103]:

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

```
[[5608 1137]
 [1494 3478]]
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      6745
           1       0.75      0.70      0.73      4972

    accuracy                           0.78     11717
   macro avg       0.77      0.77      0.77     11717
weighted avg       0.77      0.78      0.77     11717
```

In [104]:

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

```
[[5849  896]
 [1242 3730]]
              precision    recall  f1-score   support

           0       0.82      0.87      0.85      6745
           1       0.81      0.75      0.78      4972

    accuracy                           0.82     11717
   macro avg       0.82      0.81      0.81     11717
weighted avg       0.82      0.82      0.82     11717
```

In [97]:

```
print("Confusion Matrix:")
print(confusion_matrix(df_train.Exited,  XGB.predict(df_train.loc[:, df_train.columns != 'Exited'])))

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

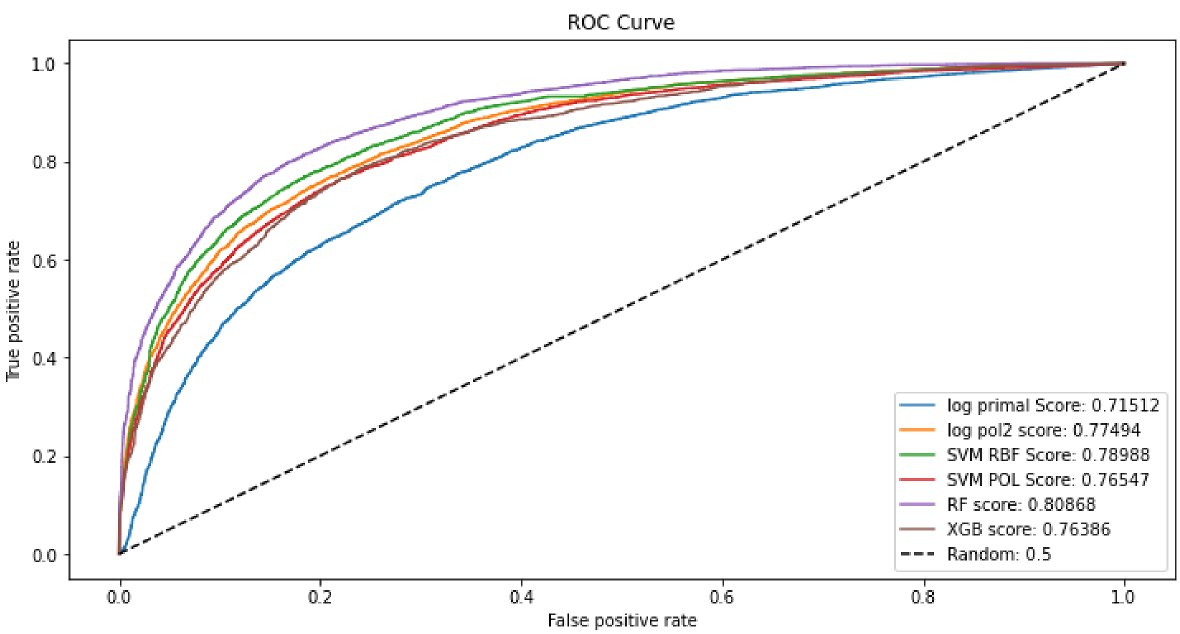
```
Confusion Matrix:
[[5589 1156]
 [1496 3476]]
              precision    recall  f1-score   support

           0       0.79      0.83      0.81      6745
           1       0.75      0.70      0.72      4972

    accuracy                           0.77     11717
   macro avg       0.77      0.76      0.77     11717
weighted avg       0.77      0.77      0.77     11717
```

In [95]:

```
y = df_train.Exited
X = df_train.loc[:, df_train.columns != 'Exited']
X_pol2 = df_train_pol2
auc_log_primal, fpr_log_primal, tpr_log_primal = get_auc_scores(y, log_primal.predict(X
),log_primal.predict_proba(X)[:,1])
auc_log_pol2, fpr_log_pol2, tpr_log_pol2 = get_auc_scores(y, log_pol2.predict(X_pol2),l
og_pol2.predict_proba(X_pol2)[:,1])
auc_SVM_RBF, fpr_SVM_RBF, tpr_SVM_RBF = get_auc_scores(y, SVM_RBF.predict(X),SVM_RBF.pr
edict_proba(X)[:,1])
auc_SVM_POL, fpr_SVM_POL, tpr_SVM_POL = get_auc_scores(y, SVM_POL.predict(X),SVM_POL.pr
edict_proba(X)[:,1])
auc_RF, fpr_RF, tpr_RF = get_auc_scores(y, RF.predict(X),RF.predict_proba(X)[:,1])
auc_XGB, fpr_XGB, tpr_XGB = get_auc_scores(y, XGB.predict(X),XGB.predict_proba(X)[:,1])
```

In [99]:

```
plt.figure(figsize = (12,6), linewidth= 1)
plt.plot(fpr_log_primal, tpr_log_primal, label = 'log primal Score: ' + str(round(auc_l
og_primal, 5)))
plt.plot(fpr_log_pol2, tpr_log_pol2, label = 'log pol2 score: ' + str(round(auc_log_pol
2, 5)))
plt.plot(fpr_SVM_RBF, tpr_SVM_RBF, label = 'SVM RBF Score: ' + str(round(auc_SVM_RBF, 5
)))
plt.plot(fpr_SVM_POL, tpr_SVM_POL, label = 'SVM POL Score: ' + str(round(auc_SVM_POL, 5
)))
plt.plot(fpr_RF, tpr_RF, label = 'RF score: ' + str(round(auc_RF, 5)))
plt.plot(fpr_XGB, tpr_XGB, label = 'XGB score: ' + str(round(auc_XGB, 5)))
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()
```



In [ ]:

```
'''From the above results, my main aim is to predict the customers that will possibly c
hurn so they can be put in some sort
of scheme/promotion of $200 to prevent churn hence the recall measures on the
1's is of more importance to me than the overall accuracy score of the model.

Given that in the data we had 48% of churn, a recall greater than this baseline will al
ready be an improvement but we
want to get as high as possible while trying to maintain a high precision so that the b
ank can train its resources effectively
towards clients highlighted by the model without wasting too much resources on the fals
e positives.

From the review of the fitted models above, the best model that gives a decent balance
 of the recall and precision is the
random forest where according to the fit on the training set, with a precision score on
1's of 0.81,
out of all customers that the model thinks will churn, 81% do actually churn and with t
he
recall score of 0.75 on the 1's, the model is able to highlight 74% of all those who ch
urned.
```

In [105]:

```
#Test model prediction accuracy on test data
# 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
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:966: Se
ttingWithCopyWarning:
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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self.obj[item] = s
<ipython-input-68-90ce4ee68310>:25: 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-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df_predict[i+'_'+j] = np.where(df_predict[i] == j,1,-1)
```

Out[105]:

```
(2929, 17)
```

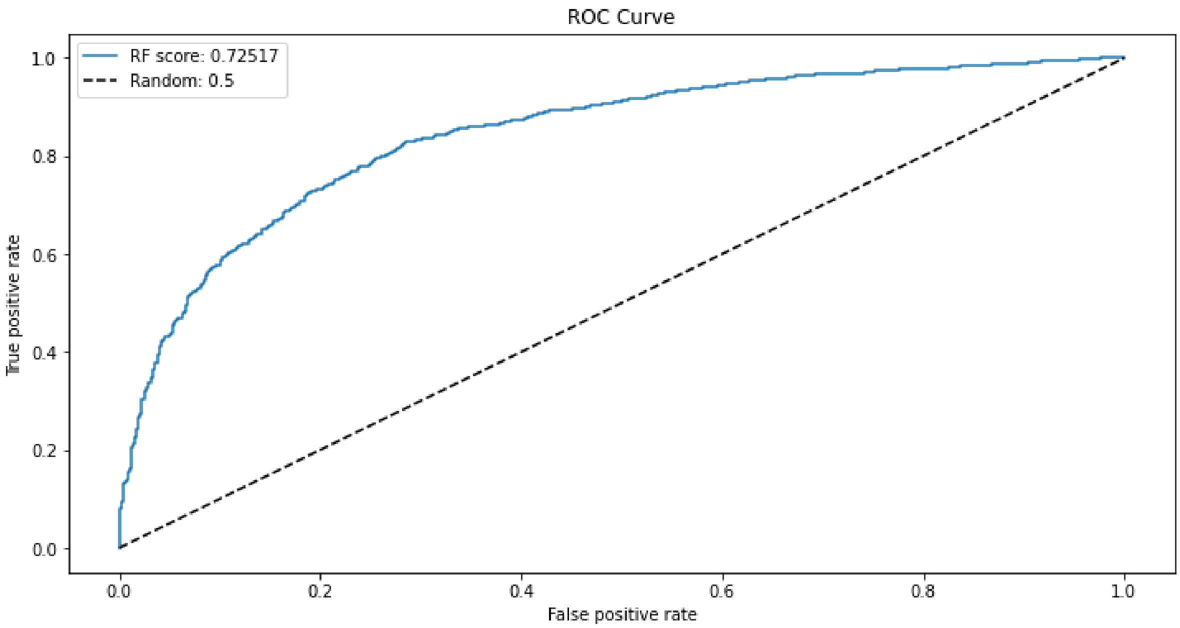
In [106]:

```
print(confusion_matrix(df_test.Exited,  RF.predict(df_test.loc[:, df_test.columns != 'E
xited'])))

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

```
[[1502  128]
 [ 612  687]]
              precision    recall  f1-score   support

           0       0.71      0.92      0.80      1630
           1       0.84      0.53      0.65      1299

    accuracy                           0.75      2929
   macro avg       0.78      0.73      0.73      2929
weighted avg       0.77      0.75      0.73      2929
```

In [107]:

```
auc_RF_test, fpr_RF_test, tpr_RF_test = get_auc_scores(df_test.Exited, RF.predict
                                                        (df_test.loc[:, df_test.columns
!= 'Exited']),
                                                        RF.predict_proba(df_test.loc[:,
df_test.columns != 'Exited'])[:,1])
plt.figure(figsize = (12,6), linewidth= 1)
plt.plot(fpr_RF_test, tpr_RF_test, label = 'RF score: ' + str(round(auc_RF_test, 5)))
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()
```



In [15]:

```
'''The precision of the model on previousy unseen test data is slightly higher with reg
ard to predicting
1's i.e. those customers that churn. However, in as much as the model has a high accura
cy,
it still some half of those who end up churning.
This could be imporved by providing retraining the model with more data over time.'''
```

Out[15]:

```
"The precision of the model on previousy unseen test data is slightly high
er with regard to predicting \n1's i.e. those customers that churn. Howeve
r, in as much as the model has a high accuracy, \nit still some half of th
ose who end up churning.\nThis could be imporved by providing retraining t
he model with more data over time."
```

In [ ]:

```

```