In [1]:
```
#Lets import the necessary library
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from scipy import stats
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_curve,roc_auc_score
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#import the data
data=pd.read_csv(r"D:\My projects\Bankcustomer_churn\Churn Modeling.csv")
```

In [3]:
```
#check the data
data.head()
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Nu
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

In [4]:
```
data.tail()
```

Out[4]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61
9997	9998	15584532	Liu	709	France	Female	36	7	0.00
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79

In [5]:
```
print(f"The data contains: \nTotla number of  rows: {data.shape[0]} \nTotal number of
```

```
The data contains:
Totla number of  rows: 10000
Total number of Columns: 14
```

In [6]:
```
data.isnull().sum()
```

Out[6]:
```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

In [7]:
```
data.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	
RowNumber	10000.0	5.000500e+03	2886.895680	1.00	2500.75	5.000500e+03	7.5002
CustomerId	10000.0	1.569094e+07	71936.186123	15565701.00	15628528.25	1.569074e+07	1.5753
CreditScore	10000.0	6.505288e+02	96.653299	350.00	584.00	6.520000e+02	7.1800
Age	10000.0	3.892180e+01	10.487806	18.00	32.00	3.700000e+01	4.4000
Tenure	10000.0	5.012800e+00	2.892174	0.00	3.00	5.000000e+00	7.0000
Balance	10000.0	7.648589e+04	62397.405202	0.00	0.00	9.719854e+04	1.2764
NumOfProducts	10000.0	1.530200e+00	0.581654	1.00	1.00	1.000000e+00	2.0000
HasCrCard	10000.0	7.055000e-01	0.455840	0.00	0.00	1.000000e+00	1.0000
IsActiveMember	10000.0	5.151000e-01	0.499797	0.00	0.00	1.000000e+00	1.0000
EstimatedSalary	10000.0	1.000902e+05	57510.492818	11.58	51002.11	1.001939e+05	1.4938
Exited	10000.0	2.037000e-01	0.402769	0.00	0.00	0.000000e+00	0.0000

In [8]:
```
data1=data.drop(['RowNumber','CustomerId','Surname'],axis=1)
```

In [9]:
```
data1
```

Out[9]:

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActive
0	619	France	Female	42	2	0.00	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	
2	502	France	Female	42	8	159660.80	3	1	
3	699	France	Female	39	1	0.00	2	0	
4	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	
9995	771	France	Male	39	5	0.00	2	1	
9996	516	France	Male	35	10	57369.61	1	1	
9997	709	France	Female	36	7	0.00	1	0	
9998	772	Germany	Male	42	3	75075.31	2	1	
9999	792	France	Female	28	4	130142.79	1	1	

10000 rows × 11 columns

In [10]:
```
print(f"The data contains: \nTotla number of  rows: {data1.shape[0]} \nTotal number of
```

```
The data contains:
Totla number of  rows: 10000
Total number of Columns: 11
```

In [11]:
```
data1.describe().T
```

Out[11]:

	count	mean	std	min	25%	50%	75%	
CreditScore	10000.0	650.528800	96.653299	350.00	584.00	652.000	718.0000	
Age	10000.0	38.921800	10.487806	18.00	32.00	37.000	44.0000	
Tenure	10000.0	5.012800	2.892174	0.00	3.00	5.000	7.0000	
Balance	10000.0	76485.889288	62397.405202	0.00	0.00	97198.540	127644.2400	250
NumOfProducts	10000.0	1.530200	0.581654	1.00	1.00	1.000	2.0000	
HasCrCard	10000.0	0.705500	0.455840	0.00	0.00	1.000	1.0000	
IsActiveMember	10000.0	0.515100	0.499797	0.00	0.00	1.000	1.0000	
EstimatedSalary	10000.0	100090.239881	57510.492818	11.58	51002.11	100193.915	149388.2475	199
Exited	10000.0	0.203700	0.402769	0.00	0.00	0.000	0.0000	

In [12]:
```
cat_val=data1.columns.tolist()
```

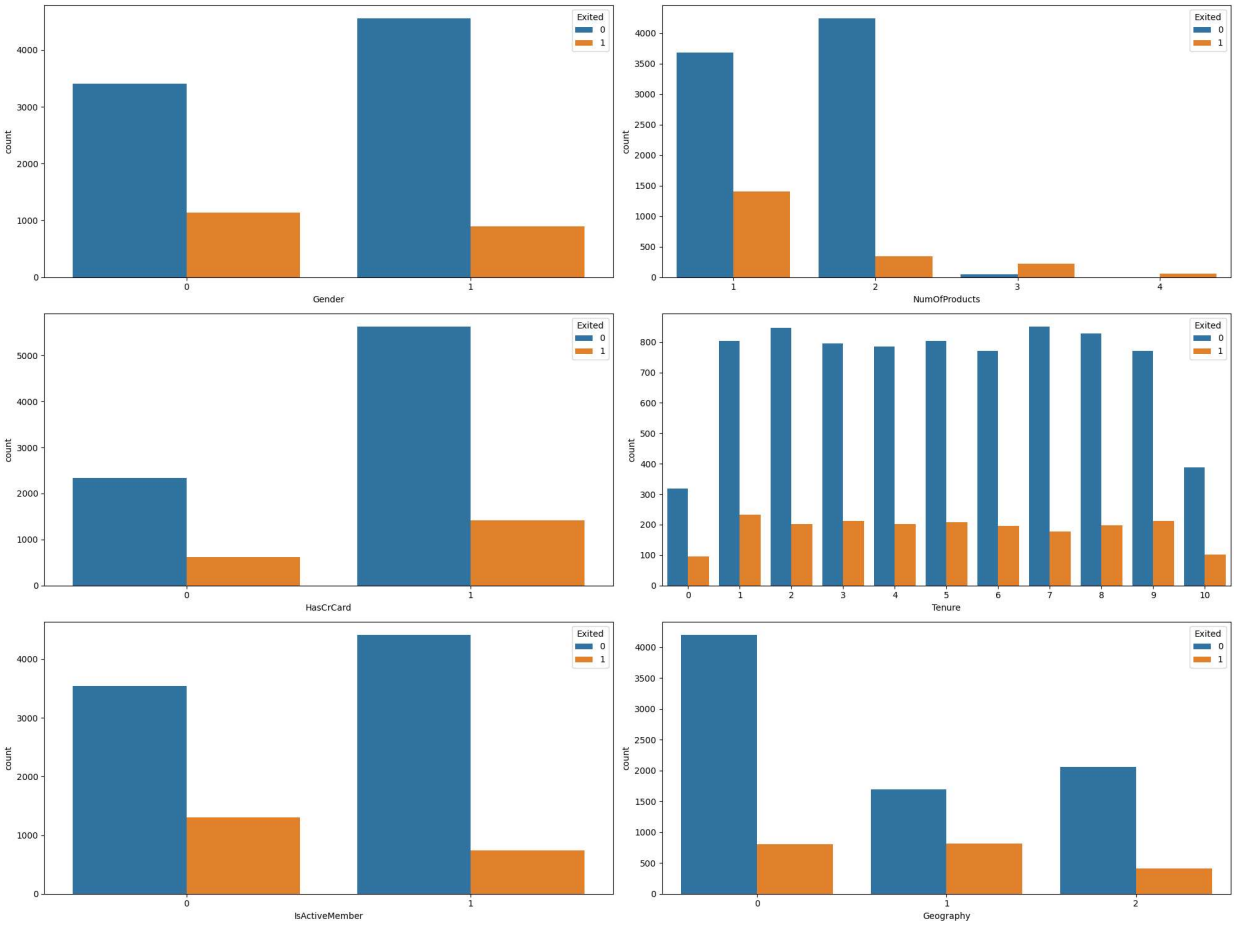
In [13]:
```
#Check the features in the data
for column in cat_val:
    print(data1[column].value_counts())
    print("#"*40)
```

```
CreditScore
850    233
678     63
655     54
705     53
667     53
      ...
404      1
351      1
365      1
417      1
419      1
Name: count, Length: 460, dtype: int64
########################################
Geography
France     5014
Germany    2509
Spain      2477
Name: count, dtype: int64
########################################
Gender
Male      5457
Female    4543
Name: count, dtype: int64
########################################
Age
37    478
38    477
35    474
36    456
34    447
     ...
92      2
82      1
88      1
85      1
83      1
Name: count, Length: 70, dtype: int64
########################################
Tenure
2     1048
1     1035
7     1028
8     1025
5     1012
3     1009
4      989
9      984
6      967
10     490
0      413
Name: count, dtype: int64
########################################
Balance
0.00         3617
130170.82       2
105473.74       2
85304.27        1
159397.75       1
             ...
```

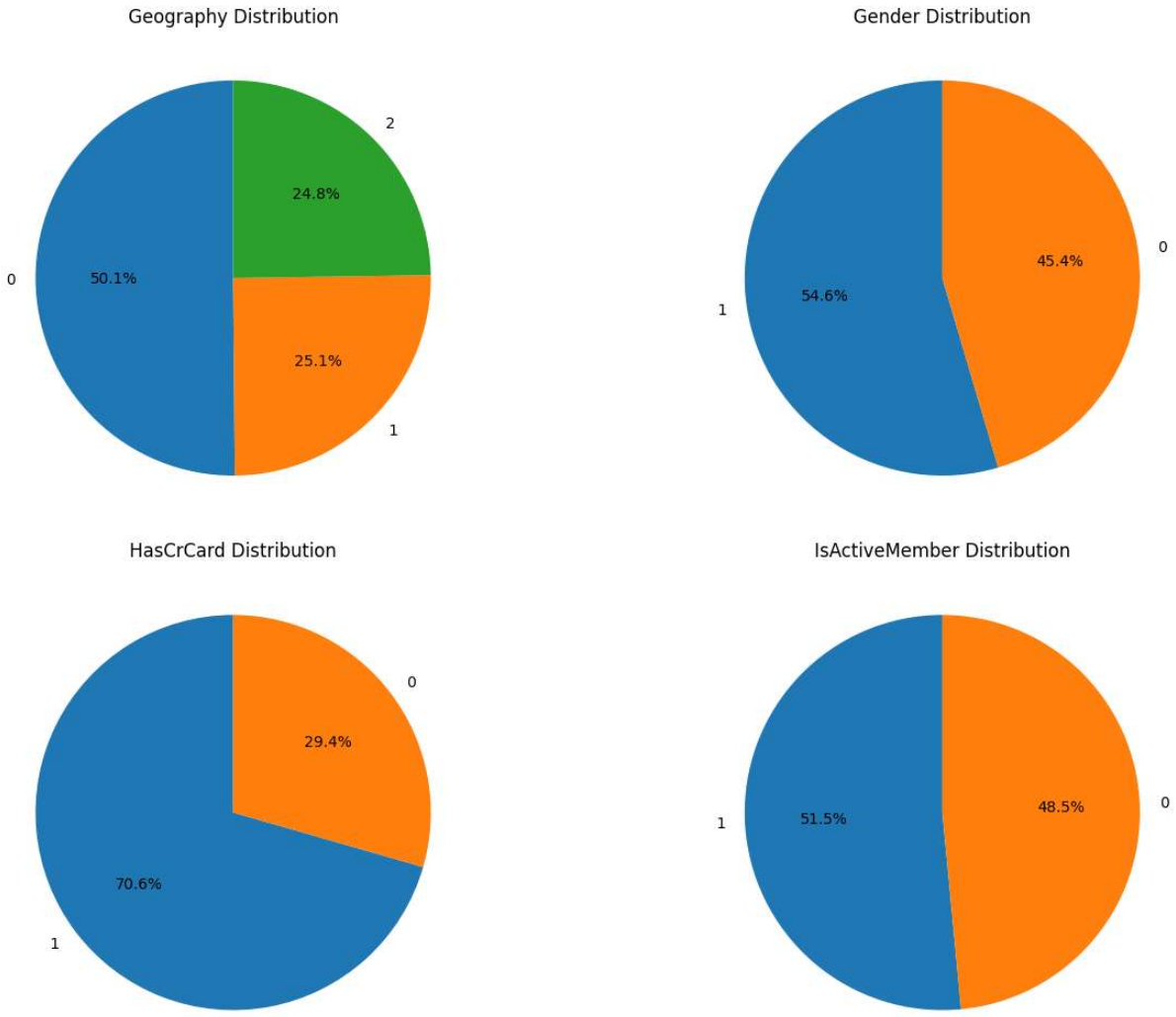
```
81556.89        1
112687.69       1
108698.96       1
238387.56       1
130142.79       1
Name: count, Length: 6382, dtype: int64
########################################
NumOfProducts
1    5084
2    4590
3     266
4      60
Name: count, dtype: int64
########################################
HasCrCard
1    7055
0    2945
Name: count, dtype: int64
########################################
IsActiveMember
1    5151
0    4849
Name: count, dtype: int64
########################################
EstimatedSalary
24924.92     2
101348.88    1
55313.44     1
72500.68     1
182692.80    1
            ..
120893.07    1
188377.21    1
55902.93     1
4523.74      1
38190.78     1
Name: count, Length: 9999, dtype: int64
########################################
Exited
0    7963
1    2037
Name: count, dtype: int64
########################################
```

Exploratory Data Analysis (EDA)

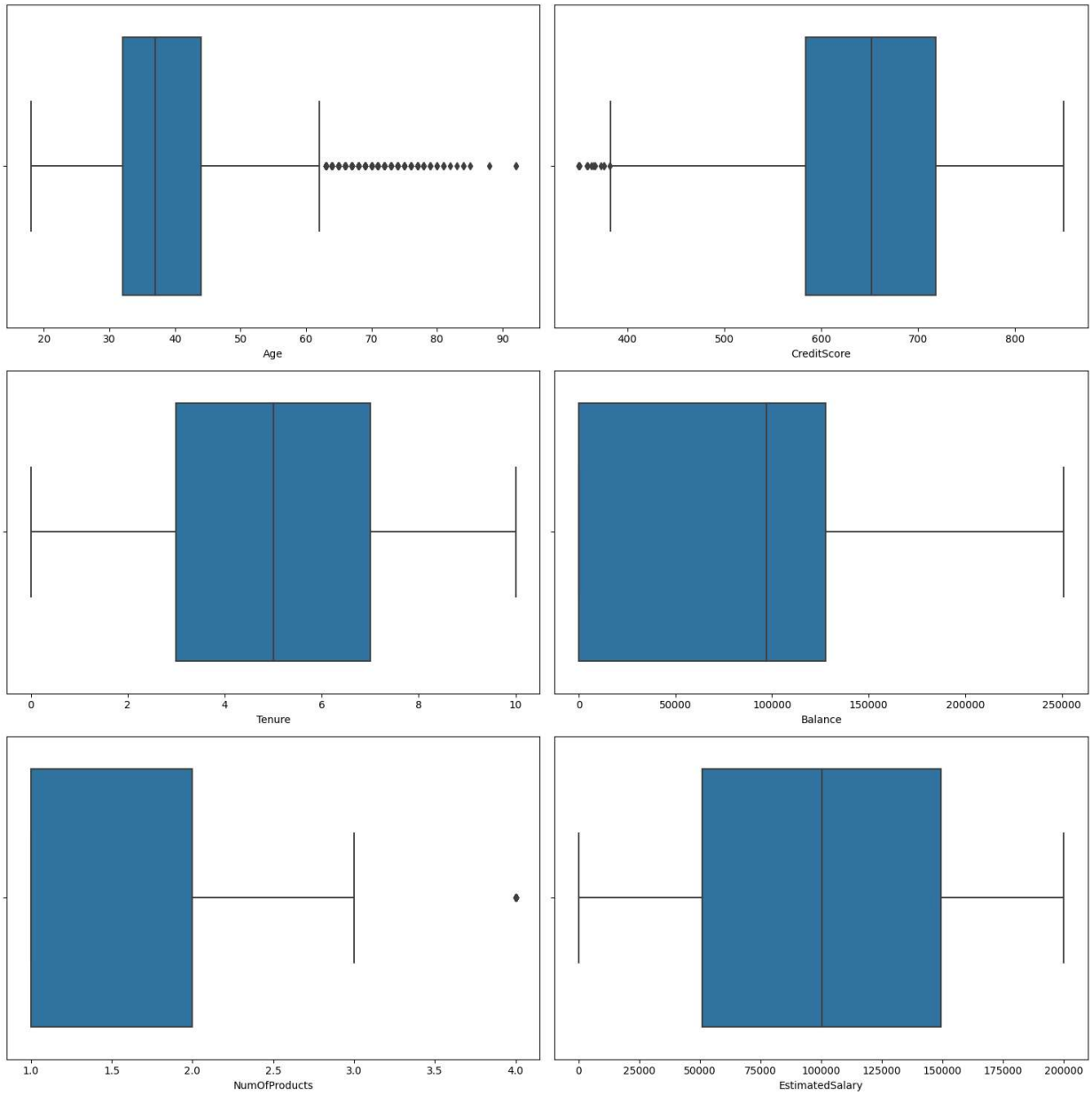
In [96]:
```
col_m=["Gender","NumOfProducts","HasCrCard","Tenure",'IsActiveMember','Geography']

fig,axs=plt.subplots(nrows=3,ncols=2,figsize=(20,15))
axs=axs.flatten()
for i,var in enumerate (col_m):
    sns.countplot(data=data1,x=var,hue='Exited',ax=axs[i])
plt.tight_layout()
plt.show()
```



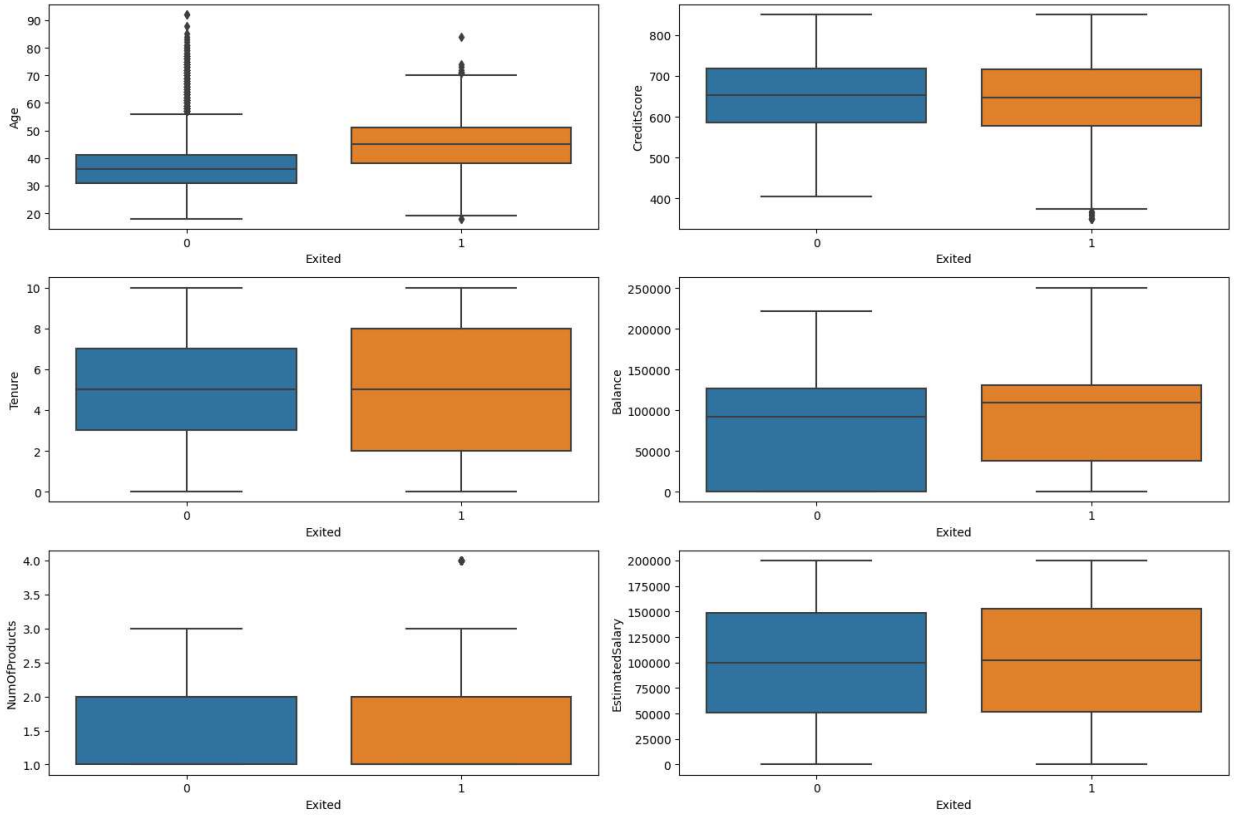
In [42]:
```
cat_obj=['Geography','Gender','HasCrCard','IsActiveMember']
fig,axs=plt.subplots(nrows=2,ncols=2,figsize=(15,10))

for i, var in enumerate(cat_obj):
    if i < len(axs.flat):
        obj_cont=data1[var].value_counts()
        axs.flat[i].pie(obj_cont,labels=obj_cont.index,autopct="%1.1f%%",startangle=90
        axs.flat[i].set_title(f'{var} Distribution')
fig.tight_layout()
plt.show()
```



In [43]:
```
num=['Age','CreditScore','Tenure','Balance','NumOfProducts','EstimatedSalary']

fig,axs=plt.subplots(nrows=3,ncols=2,figsize=(15,15))
axs=axs.flatten()

for i, var in enumerate(num):
    sns.boxplot(x=var,data=data1,ax=axs[i])
fig.tight_layout()
plt.show()
```



In [44]:
```
num_1=['Age','CreditScore','Tenure','Balance','NumOfProducts','EstimatedSalary']
fig,axs=plt.subplots(nrows=3,ncols=2,figsize=(15,10))
axs=axs.flatten()
for i,var in enumerate (num_1):
    sns.boxplot(y=var,x='Exited',data=data1,ax=axs[i])
fig.tight_layout()
plt.show()
```



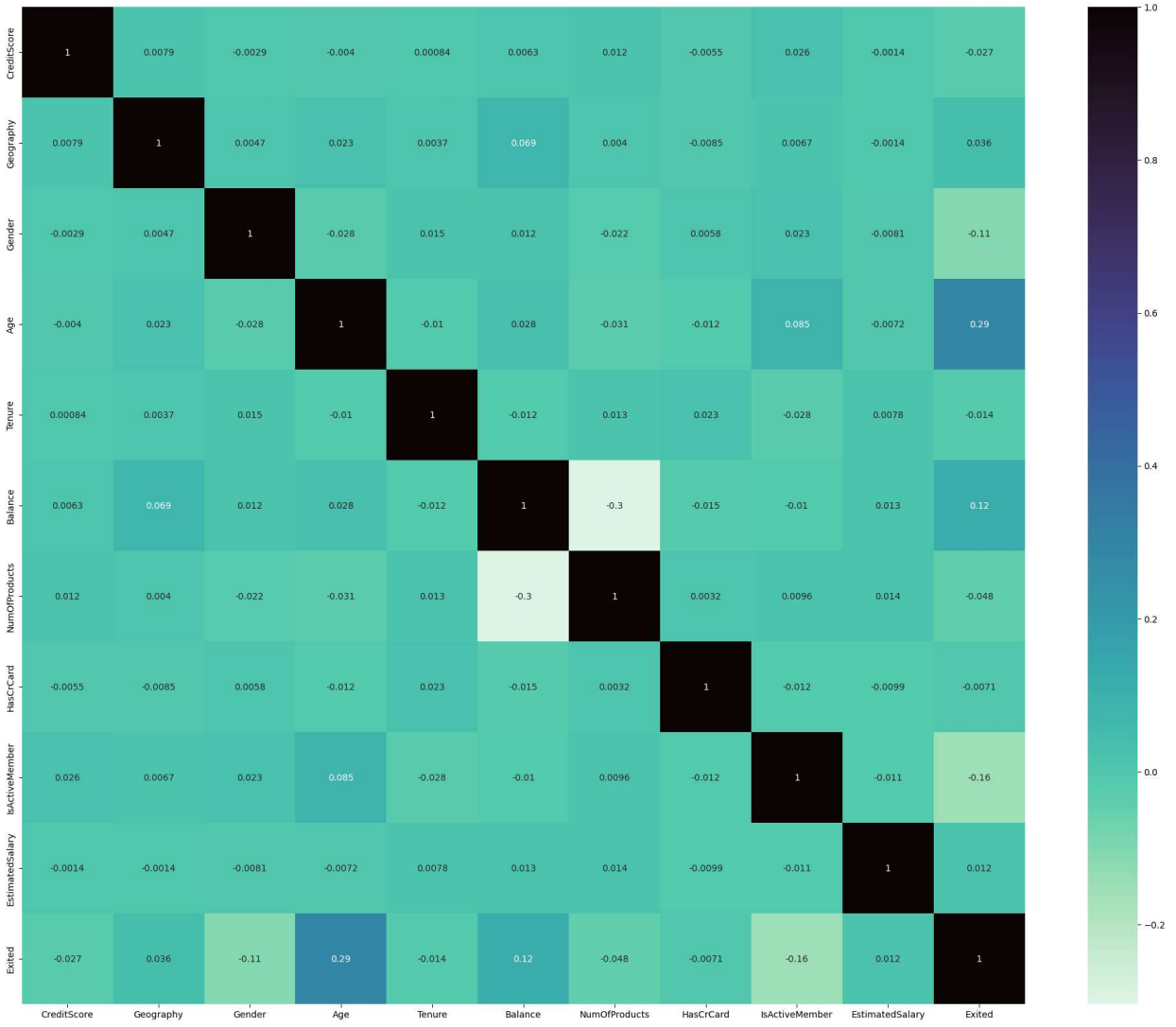
In [45]:
```
data1.isnull().sum()*100/data1.shape[0]
```

Out[45]:
```
CreditScore        0.0
Geography          0.0
Gender             0.0
Age                0.0
Tenure             0.0
Balance            0.0
NumOfProducts      0.0
HasCrCard          0.0
IsActiveMember     0.0
EstimatedSalary    0.0
Exited             0.0
dtype: float64
```

In [46]:
```
#label encoding
for col in data1.select_dtypes(include=['object']).columns:
    print(f'{col}:\n{data1[col].unique()}')
```

In [47]:
```
for col in data1.select_dtypes(include=['object']).columns:

    lable_encod=preprocessing.LabelEncoder()
    lable_encod.fit(data1[col].unique())
    data1[col]=lable_encod.transform(data1[col])
    print(f'{col}:\n{data1[col].unique()}')
```

In [48]:
```
plt.figure(figsize=(25,20))
sns.heatmap(data1.corr(),cmap='mako_r',annot=True);
```



In [49]:
```
x=data1.drop(['Exited'],axis=1)
y=data1['Exited']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Outlier detection and remove

In [50]:
```
sele_colum=['Age','CreditScore','NumOfProducts']
z_score=np.abs(stats.zscore(x_train[sele_colum]))
threshold=3
outlier_inde=np.where(z_score > threshold)[0]
X_train=x_train.drop(x_train.index[outlier_inde])
Y_train=y_train.drop(y_train.index[outlier_inde])
```

Train The model with Decision Tree

In [51]:
```
decision_tree=DecisionTreeClassifier(class_weight='balanced')
param_grid={
    "max_depth":[3,4,5,6,7,8],
    "min_samples_split":[2,3,4],
    "min_samples_leaf":[1,2,3,4],
```

```
    "random_state":[0,42]
}
grid_search=GridSearchCV(decision_tree,param_grid,cv=5)
grid_search.fit(X_train,Y_train)
print(grid_search.best_params_)
```

```
{'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 0}
```

In [52]:
```
dtree=DecisionTreeClassifier(random_state=0,max_depth=3,min_samples_leaf=1,min_samples
dtree.fit(X_train,Y_train)
```

Out[52]:
```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [53]:
```
from sklearn.metrics import accuracy_score
y_pred=dtree.predict(x_test)
print(f'Accuracy_score: {round(accuracy_score(y_test,y_pred)*100,2)}%')
```

```
Accuracy_score: 84.25%
```

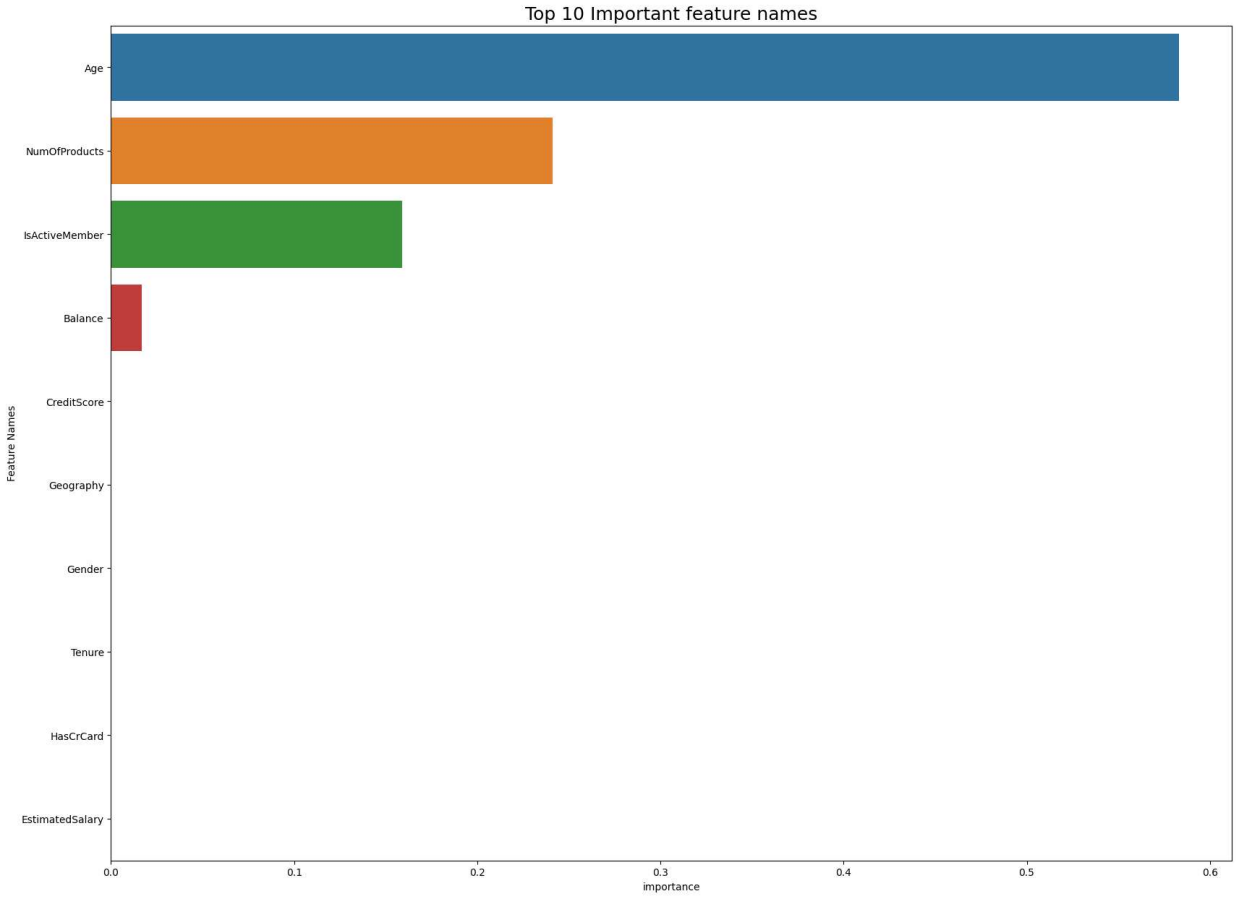
Checking the Accuracy and validation

In [54]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,log_l
print("F1 Score :",(f1_score(y_test,y_pred,average='micro')))
print("Precision Score: ",(precision_score(y_test,y_pred,average='micro')))
print("Recall_score: ",(recall_score(y_test,y_pred,average='micro')))
print("Log Los :",(log_loss(y_test,y_pred)))
```

```
F1 Score : 0.8425
Precision Score:  0.8425
Recall_score:  0.8425
Log Los : 5.439864878374019
```

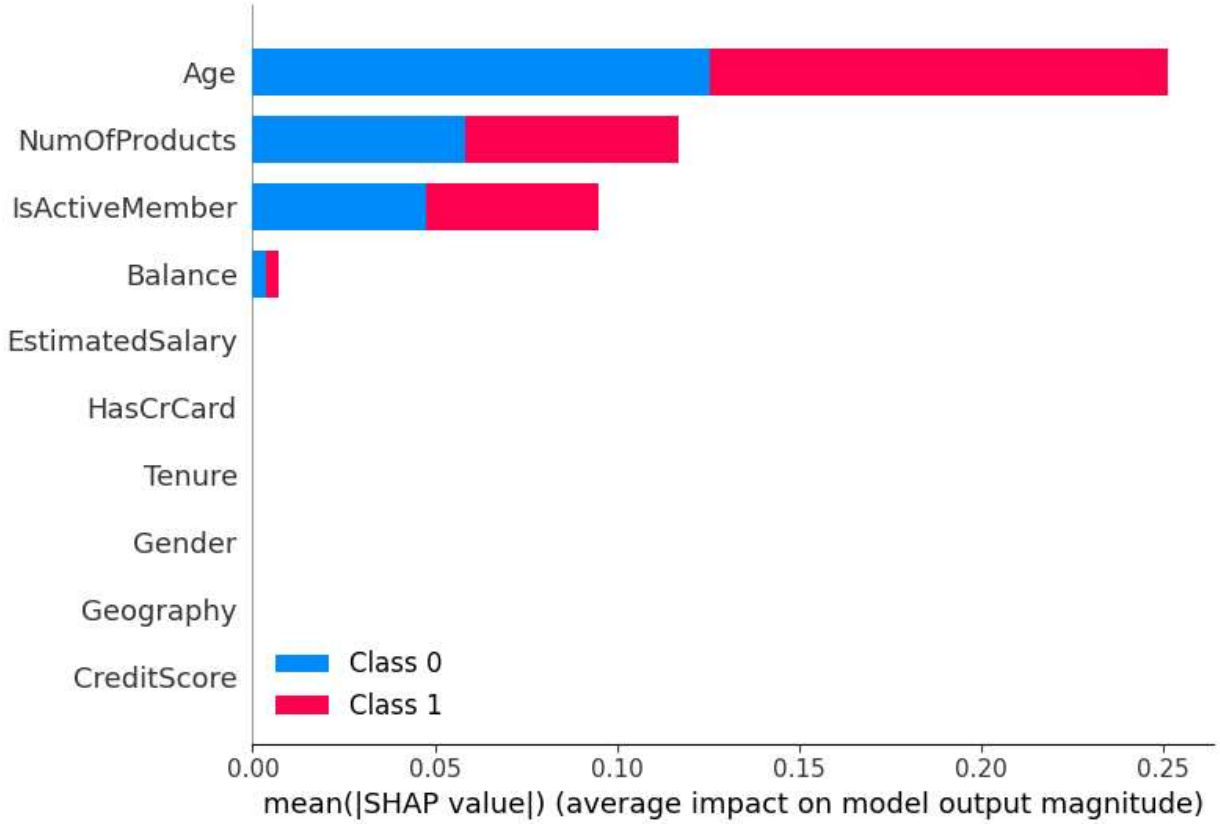
In [55]:
```
im_df=pd.DataFrame({"Feature Names":X_train.columns,"importance":dtree.feature_importa
fe=im_df.sort_values(by="importance",ascending=False)
```

In [56]:
```
fe1=fe.head(10)
plt.figure(figsize=(20,15))
sns.barplot(data=fe1,x="importance",y="Feature Names")
plt.title("Top 10 Important feature names",fontsize=18)
plt.show()
```

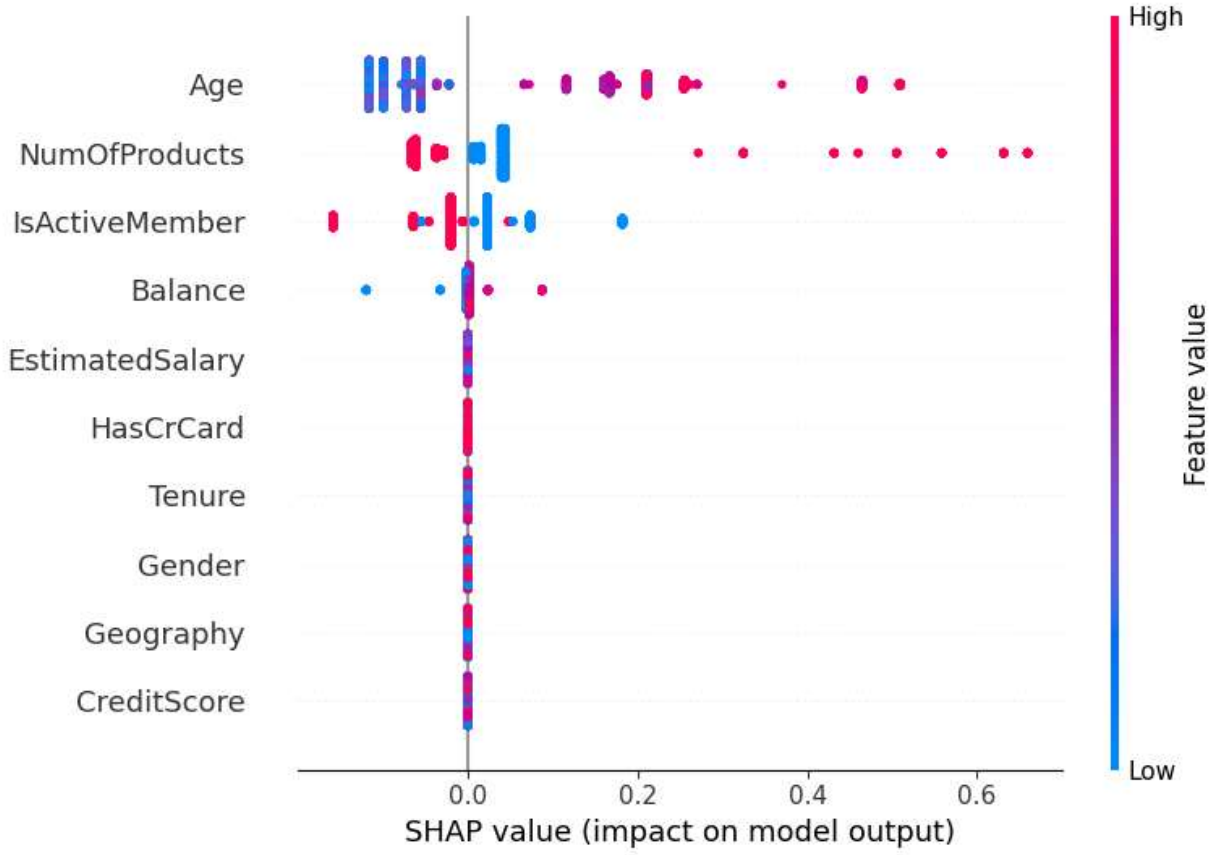


Feature importance explainer with SHAP

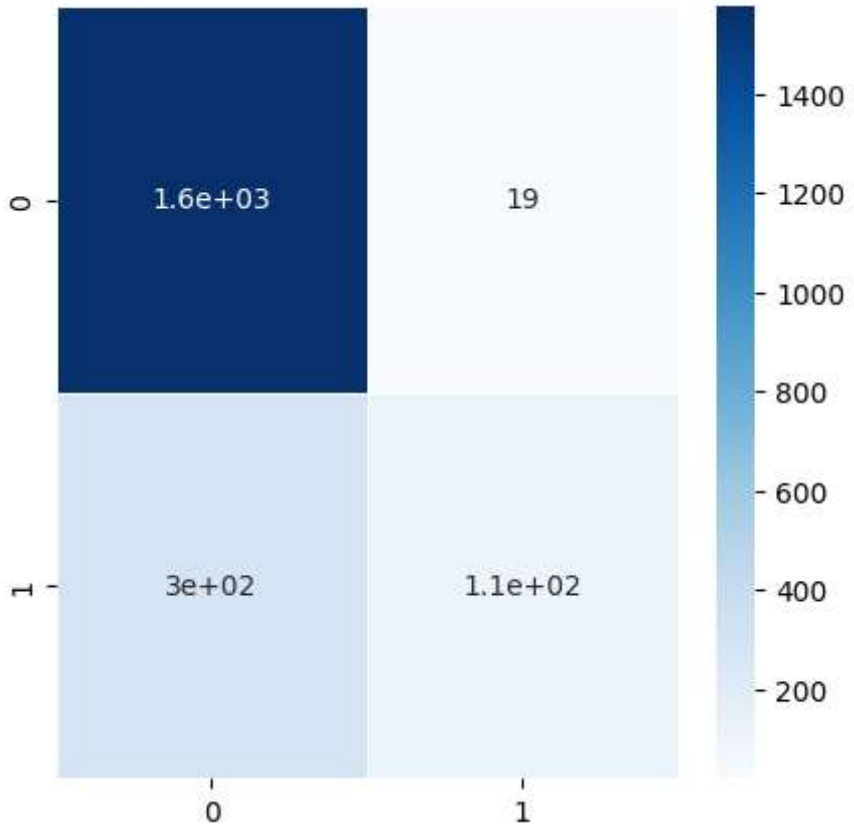
In [57]:
```
import shap
explainer=shap.TreeExplainer(dtree)
shap_values=explainer.shap_values(x_test)
shap.summary_plot(shap_values,x_test)
```



In [61]:
```
explainer=shap.TreeExplainer(dtree)
shap_values=explainer.shap_values(x_test)
shap.summary_plot(shap_values[1],x_test.values,feature_names=x_test.columns)
```

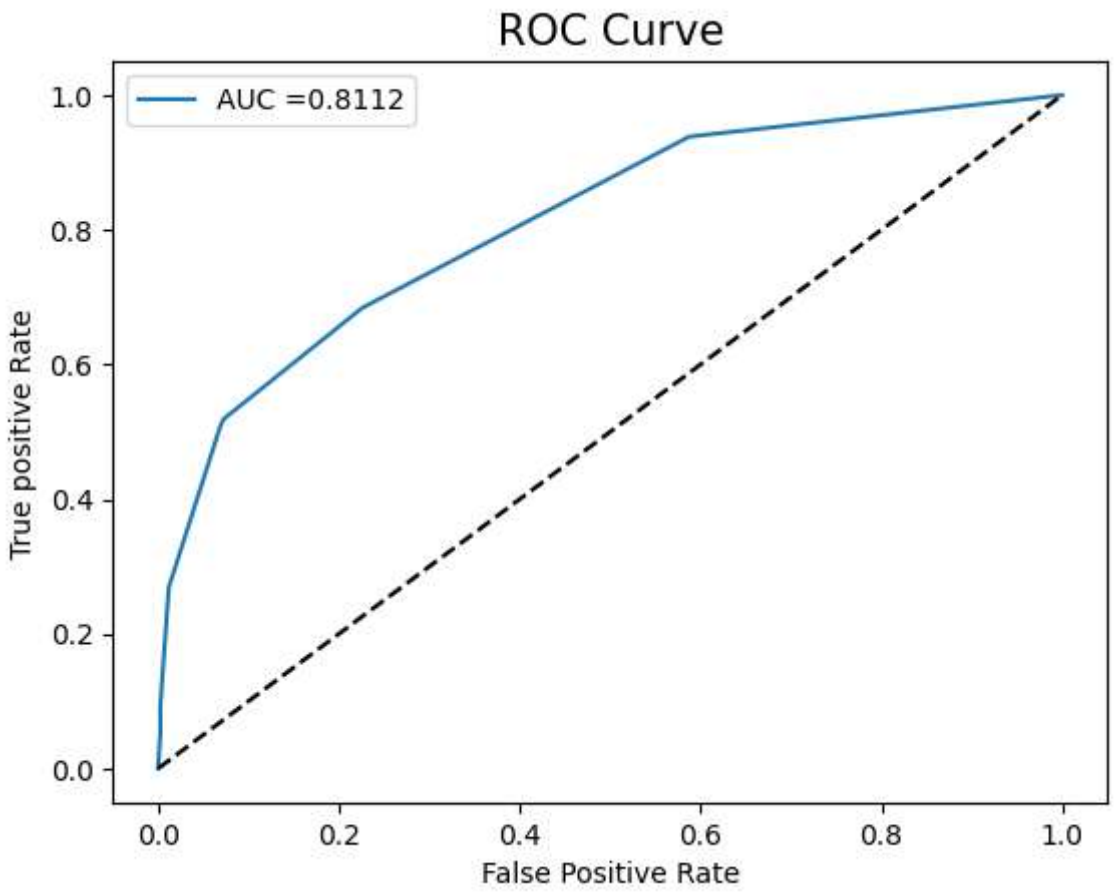


In [62]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,annot=True,cmap="Blues",linewidths=0.5)
plt.show()
```



In [63]:
```
y_pred_prob=dtree.predict_proba(x_test)[:][:,1]
df_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test),columns=['y_actual']),
                               pd.DataFrame(y_pred_prob,columns=['y_pred_prob'])],axis
df_actual_predicted.index=y_test.index
fqr,tqr,tr=roc_curve(df_actual_predicted['y_actual'],df_actual_predicted['y_pred_prob'
auc=roc_auc_score(df_actual_predicted['y_actual'],df_actual_predicted['y_pred_prob'])

plt.plot(fqr,tqr,label='AUC =%0.4f' %auc)
plt.plot(fqr,fqr,linestyle="--",color="k")
plt.xlabel('False Positive Rate')
plt.ylabel('True positive Rate')
plt.title('ROC Curve',size=15)
plt.legend()
```

Out[63]:
```
<matplotlib.legend.Legend at 0x2e452e73d30>
```



Random Forest Calssifier

In [64]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt'],
    'random_state': [0, 42]
}

grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, Y_train)
print(grid_search.best_params_)
```

```
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 100, 'random_state': 0}
```

In [65]:
```
rfc=RandomForestClassifier(random_state=0,max_features='sqrt',n_estimators=100,class_w
rfc.fit(X_train,Y_train)
```

Out[65]:
```
RandomForestClassifier(class_weight='balanced', max_features='sqrt',
                       random_state=0)
```

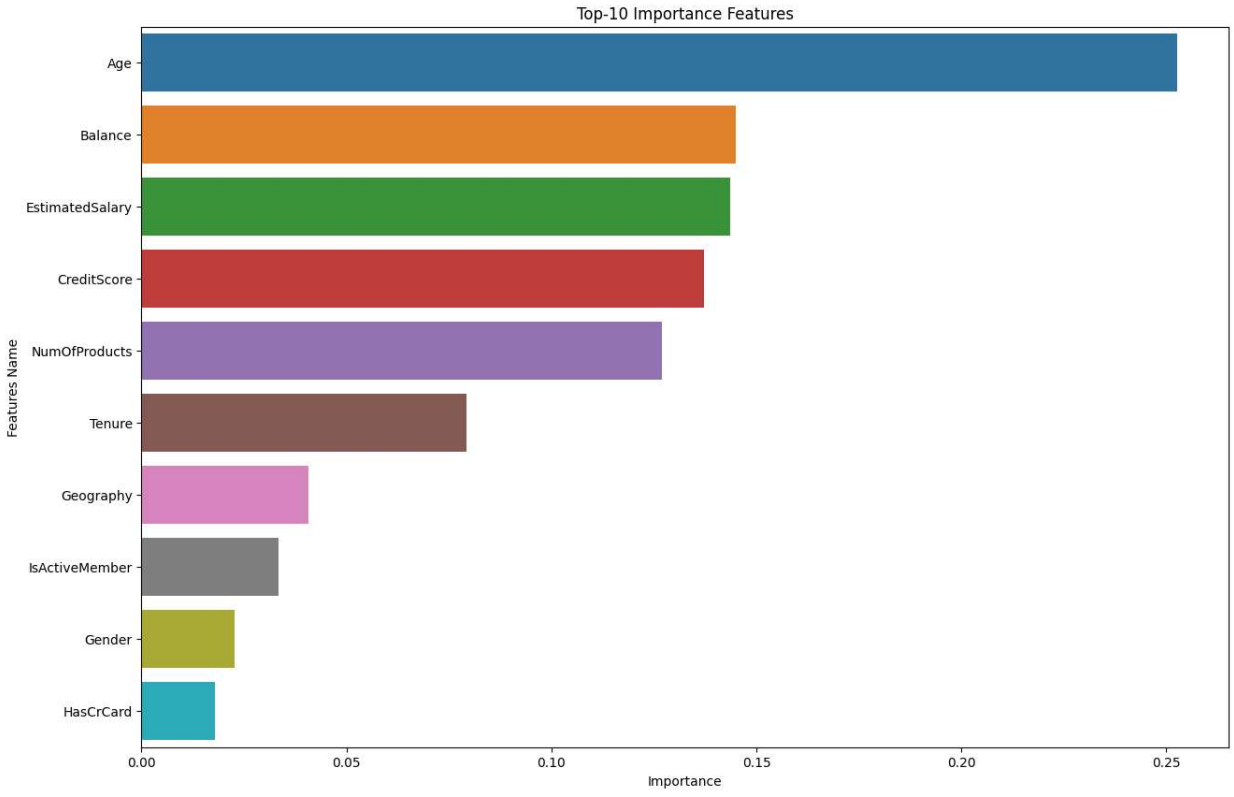
In [66]:
```
y_pred_1=rfc.predict(x_test)
print("Accuracy_score :", round(accuracy_score(y_test,y_pred_1)*100,2),"%")
```

```
Accuracy_score : 86.4 %
```

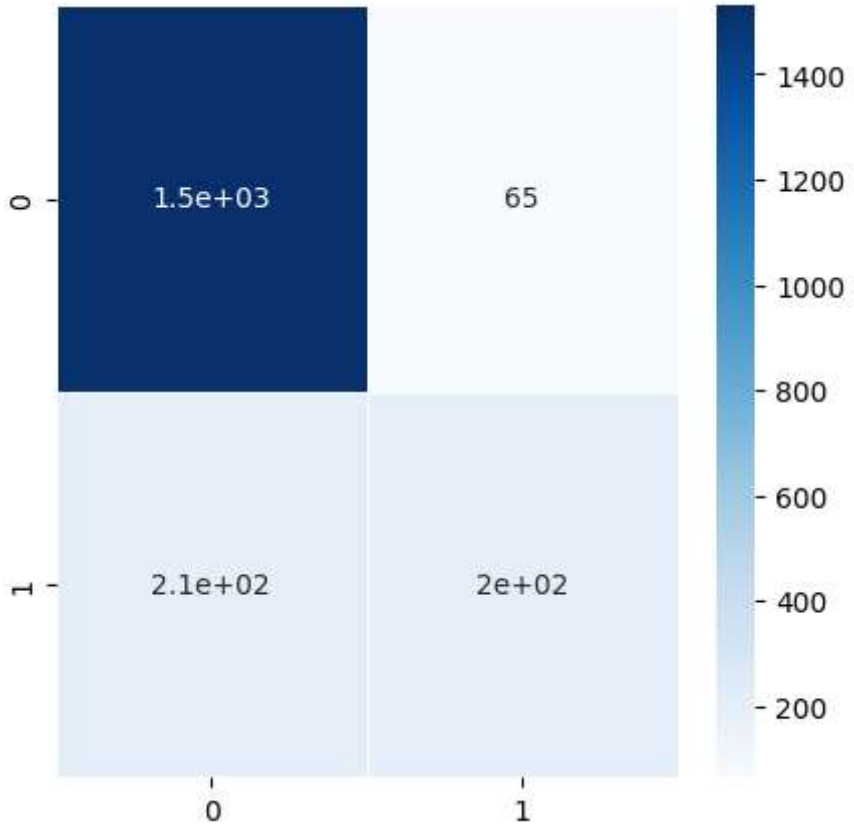
In [67]:
```
print("F1 Score :",(f1_score(y_test,y_pred_1,average='micro')))
print("Precision Score: ",(precision_score(y_test,y_pred_1,average='micro')))
print("Recall_score: ",(recall_score(y_test,y_pred_1,average='micro')))
print("Log Los :",(log_loss(y_test,y_pred_1)))
```

```
F1 Score : 0.864
Precision Score:  0.864
Recall_score:  0.864
Log Los : 4.697299576624335
```

In [68]:
```
im_fea=pd.DataFrame({
    'Features Name':X_train.columns,
    'Importance':rfc.feature_importances_
})
fim=im_fea.sort_values(by='Importance',ascending=False)

fim2=fim.head(10)
plt.figure(figsize=(15,10))
sns.barplot(data=fim2,x='Importance',y='Features Name')
plt.title('Top-10 Importance Features')
plt.xlabel('Importance')
plt.ylabel('Features Name')
plt.show()
```

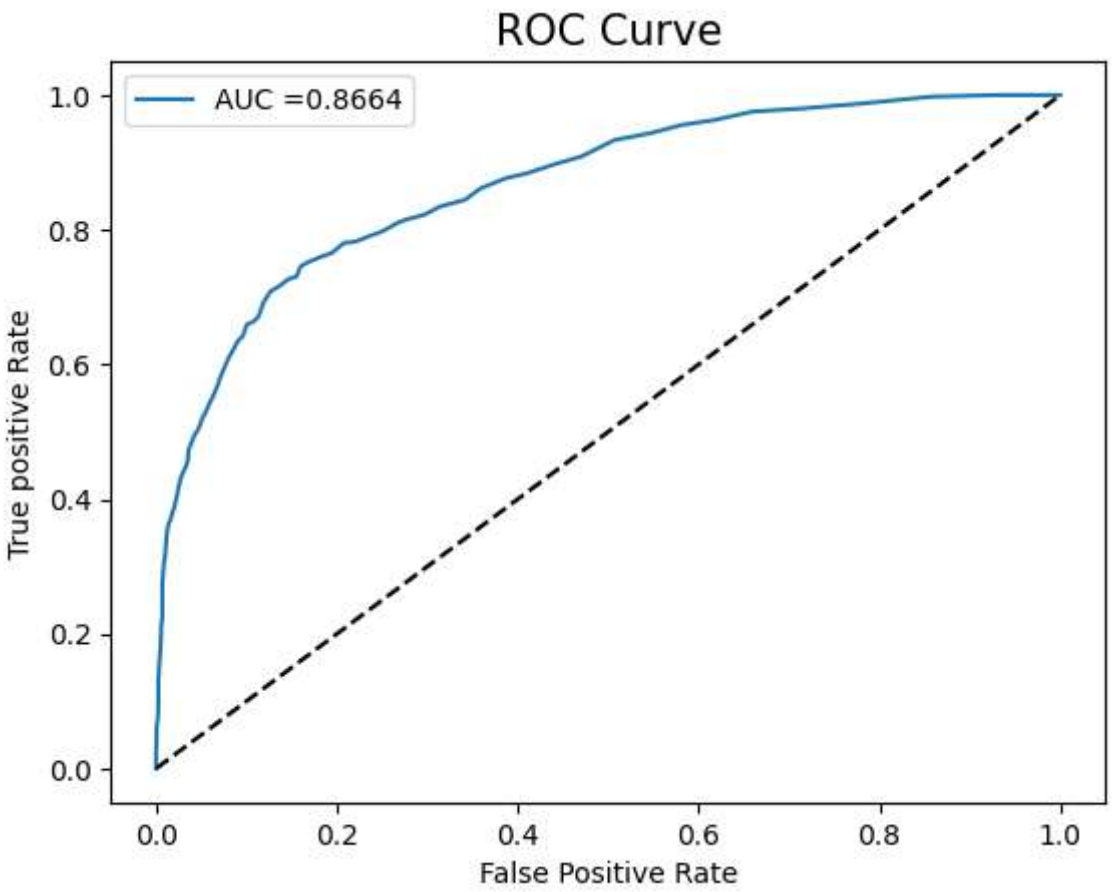


In [70]:
```
cm=confusion_matrix(y_test,y_pred_1)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,annot=True,cmap="Blues",linewidths=0.5)
plt.show()
```



In [71]:
```
y_pred_prob=rfc.predict_proba(x_test)[:][:,1]
df_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test),columns=['y_actual']),
                               pd.DataFrame(y_pred_prob,columns=['y_pred_prob'])],axis
df_actual_predicted.index=y_test.index
fqr,tqr,tr=roc_curve(df_actual_predicted['y_actual'],df_actual_predicted['y_pred_prob'
auc=roc_auc_score(df_actual_predicted['y_actual'],df_actual_predicted['y_pred_prob'])

plt.plot(fqr,tqr,label='AUC =%0.4f' %auc)
plt.plot(fqr,fqr,linestyle="--",color="k")
plt.xlabel('False Positive Rate')
plt.ylabel('True positive Rate')
plt.title('ROC Curve',size=15)
plt.legend()
```

Out[71]:
```
<matplotlib.legend.Legend at 0x2e452788100>
```



Lets do some statical analysis on the parameters

Chi_square test for categorical features

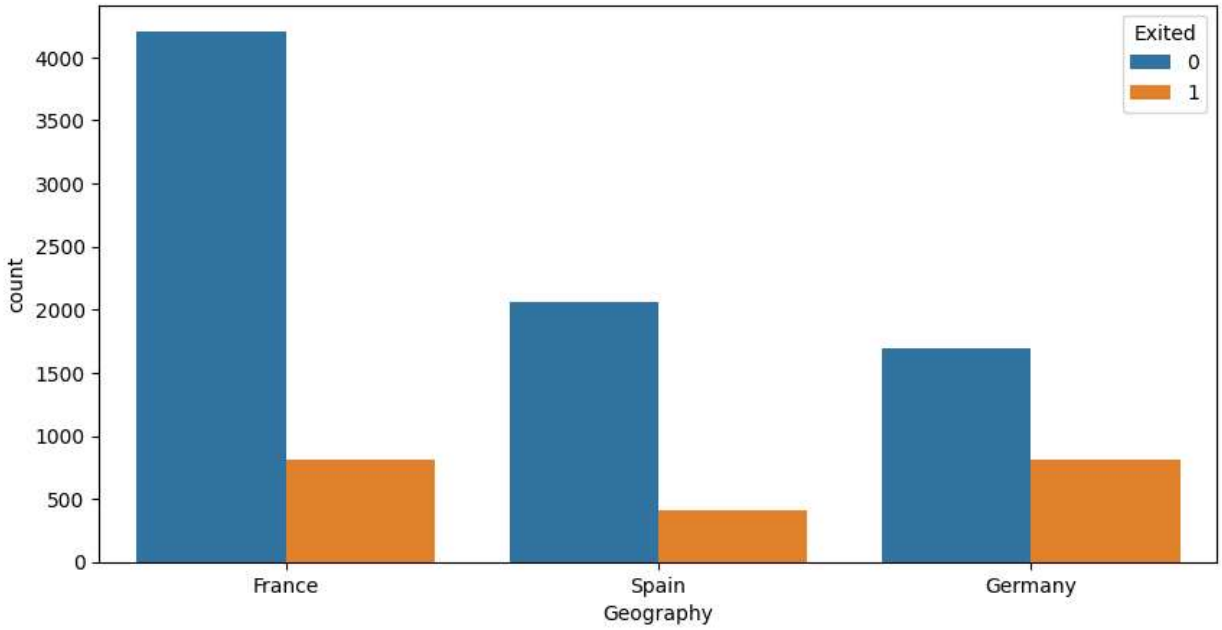
In [83]:
```
#First will do the chi_square test on the categorical values
#Select the categorical columns from the data
cat_col=data.select_dtypes(include='object').columns
cat_col1=['Geography', 'Gender']
```

In [80]:
```
#lets import the library for coluring the text and stats
import colorama
from colorama import Fore
from scipy.stats import chi2_contingency
import stat
```

In [88]:
```
for i in cat_col1:
    print(i + ":")
    plt.figure(figsize=(10, 5))
    sns.countplot(data=data, hue="Exited", x=i)
    plt.show()
    a = np.array(pd.crosstab(data.Exited, data[i]))
    (stats, p, dof,_) = chi2_contingency(a, correction=False)
    if p > 0.05:
        print(Fore.RED + "'{}' is a 'bad Predictor'".format(i))
```

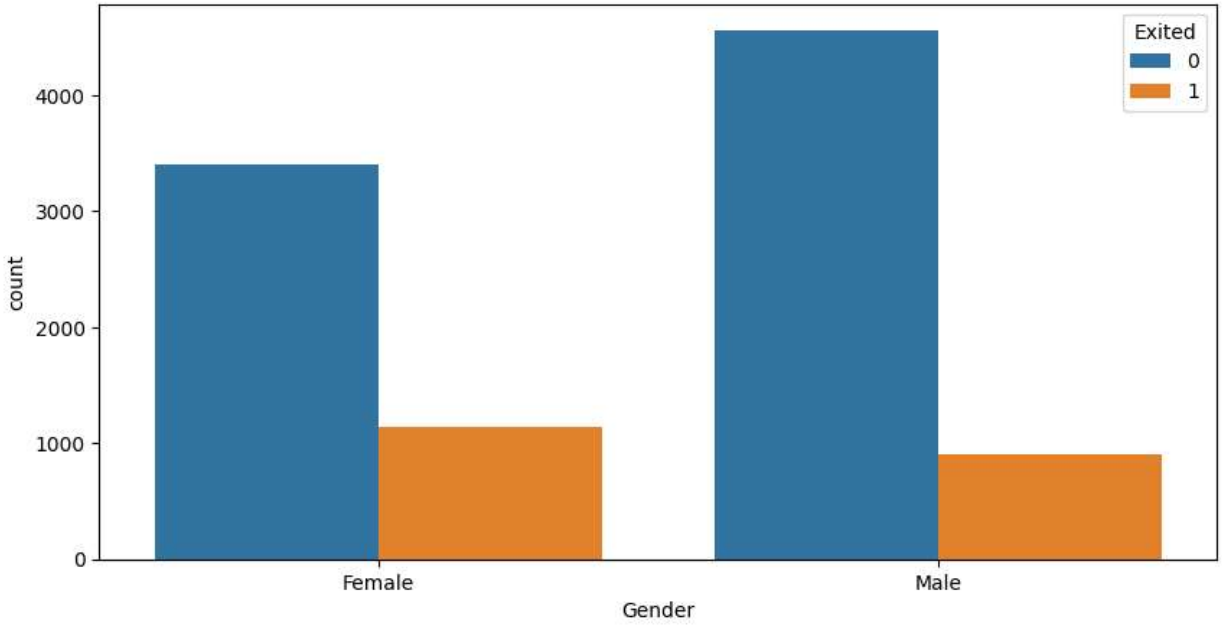
```
        print('p_val={}\n'.format(p))
    else:
        print(Fore.GREEN + "'{}' is a Good Predictor".format(i))
        print('p_val={}\n'.format(p))
```

Geography:




```
'Geography' is a Good Predictor
p_val=3.830317605354266e-66
```

Gender:




```
'Gender' is a Good Predictor
p_val=1.7204149874840846e-26
```

From the above analysis both the categorical features are good predictor, same we have used both features for prediction.

Anova Test for continous features

In [89]:
```
#Lets import the necessary stats library
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from statsmodels.formula.api import ols
import statsmodels.api as smf
```

In [91]:
```
#select the continous features
cont_f=data.select_dtypes(exclude='object').columns
cont_f
```

Out[91]:
```
Index(['RowNumber', 'CustomerId', 'CreditScore', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

In [93]:
```
#Lets take only necessary columns
cont_f1=[ 'CreditScore', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']
```

In [95]:
```
for i in cont_f1:
    print("-------------------------------------------------------------------------------
    print(i+":\n")
    print('ANOVA:\n')
    mod=ols(i+'~Exited',data=data).fit()
    aov_table=smf.stats.anova_lm(mod,type = 2)
    print(aov_table,'\n')
    print('Pvalue={}\n'.format(aov_table['PR(>F)'][0]))
    p=aov_table['PR(>F)'][0]

    if p>0.05:
        print(Fore.RED +"'{}' is a 'bad Predictor'\n".format(i))
        print('Avg of this feature is same for both card approved group and not approv
        print("p_val = {}\n".format(p))
    else:
        print('TUKEY:\n')
        print(Fore.RED +"'{}' is a 'good Predictor'\n".format(i))
        print('Avg of this feature is not same for both card approved group and not ap
        print('we need to perform Tuckey as atleast one category is different\n')
        print(Fore.GREEN +"'{}' is a 'good Predictor'\n".format(i))
        tukey=pairwise_tukeyhsd(data[i],data.Exited,alpha=0.05)
        print(tukey,'\n')
```

```
--------------------------------------------------------------------------------
------------------------------
CreditScore:

ANOVA:

              df        sum_sq       mean_sq         F    PR(>F)
Exited       1.0  6.856799e+04  68567.991506  7.344522  0.006738
Residual  9998.0  9.334069e+07   9335.936359       NaN       NaN

Pvalue=0.006738213892205324

TUKEY:

'CreditScore' is a 'good Predictor'

Avg of this feature is not same for both card approved group and not approved group

we need to perform Tuckey as atleast one category is different

'CreditScore' is a 'good Predictor'

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj   lower   upper  reject
-----------------------------------------------------
     0      1  -6.5017 0.0067 -11.2044 -1.799   True
-----------------------------------------------------

--------------------------------------------------------------------------------
------------------------------
Age:

ANOVA:

              df        sum_sq       mean_sq           F         PR(>F)
Exited       1.0  8.953639e+04  89536.388944  886.063275  1.239931e-186
Residual  9998.0  1.010294e+06    101.049656         NaN            NaN

Pvalue=1.2399313093422792e-186

TUKEY:

'Age' is a 'good Predictor'

Avg of this feature is not same for both card approved group and not approved group

we need to perform Tuckey as atleast one category is different

'Age' is a 'good Predictor'

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=================================================
group1 group2 meandiff p-adj lower  upper  reject
-------------------------------------------------
     0      1   7.4296   0.0 6.9404 7.9189   True
-------------------------------------------------

--------------------------------------------------------------------------------
------------------------------
```

```
Tenure:

ANOVA:

              df        sum_sq    mean_sq         F    PR(>F)
Exited       1.0     16.394553  16.394553  1.960164  0.161527
Residual  9998.0  83621.967047   8.363869       NaN       NaN

Pvalue=0.1615268494946745

'Tenure' is a 'bad Predictor'

Avg of this feature is same for both card approved group and not approved group

p_val = 0.1615268494946745

--------------------------------------------------------------------------------
------------------------------
Balance:

ANOVA:

              df        sum_sq       mean_sq           F        PR(>F)
Exited       1.0  5.469738e+11  5.469738e+11  142.473832  1.275563e-32
Residual  9998.0  3.838349e+13  3.839117e+09         NaN           NaN

Pvalue=1.275563319153163e-32

TUKEY:

'Balance' is a 'good Predictor'

Avg of this feature is not same for both card approved group and not approved group

we need to perform Tuckey as atleast one category is different

'Balance' is a 'good Predictor'

    Multiple Comparison of Means - Tukey HSD, FWER=0.05
============================================================
group1 group2  meandiff  p-adj   lower      upper    reject
------------------------------------------------------------
     0      1 18363.2426   0.0 15347.5818 21378.9033   True
------------------------------------------------------------

--------------------------------------------------------------------------------
------------------------------
NumOfProducts:

ANOVA:

              df       sum_sq   mean_sq          F    PR(>F)
Exited       1.0     7.735764  7.735764  22.915223  0.000002
Residual  9998.0  3375.143836  0.337582        NaN       NaN

Pvalue=1.7173330048040421e-06

TUKEY:

'NumOfProducts' is a 'good Predictor'
```

```
Avg of this feature is not same for both card approved group and not approved group

we need to perform Tuckey as atleast one category is different

'NumOfProducts' is a 'good Predictor'

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
     0      1  -0.0691   0.0 -0.0973 -0.0408   True
-----------------------------------------------------

--------------------------------------------------------------------------------
------------------------------
HasCrCard:

ANOVA:

              df       sum_sq   mean_sq         F    PR(>F)
Exited       1.0     0.105854  0.105854  0.509401  0.475415
Residual  9998.0  2077.591646  0.207801       NaN       NaN

Pvalue=0.4754149183755565

'HasCrCard' is a 'bad Predictor'

Avg of this feature is same for both card approved group and not approved group

p_val = 0.4754149183755565

--------------------------------------------------------------------------------
------------------------------
IsActiveMember:

ANOVA:

              df       sum_sq    mean_sq           F        PR(>F)
Exited       1.0    60.884518  60.884518  249.800794  1.348269e-55
Residual  9998.0  2436.835382   0.243732         NaN           NaN

Pvalue=1.3482685165052417e-55

TUKEY:

'IsActiveMember' is a 'good Predictor'

Avg of this feature is not same for both card approved group and not approved group

we need to perform Tuckey as atleast one category is different

'IsActiveMember' is a 'good Predictor'

Multiple Comparison of Means - Tukey HSD, FWER=0.05
=====================================================
group1 group2 meandiff p-adj  lower   upper  reject
-----------------------------------------------------
     0      1  -0.1937   0.0 -0.2178 -0.1697   True
-----------------------------------------------------
```

```
------------------------------------------------------------------------------
------------------------------
EstimatedSalary:

ANOVA:

              df        sum_sq       mean_sq         F   PR(>F)
Exited       1.0  4.839451e+09  4.839451e+09  1.463262  0.22644
Residual  9998.0  3.306642e+13  3.307304e+09       NaN      NaN

Pvalue=0.22644042802263928

'EstimatedSalary' is a 'bad Predictor'

Avg of this feature is same for both card approved group and not approved group

p_val = 0.22644042802263928
```

From the above anova test there were three bad predictor identified 1)Tenure 2)Has card 3)Estimated Salary