In [1]:
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.patches as patches
from matplotlib.path import Path
import pandas as pd
import seaborn as sns

import sklearn
from sklearn.linear_model import LinearRegression

import warnings
warnings.filterwarnings("ignore")


import seaborn as sns

%matplotlib inline


matplotlib.interactive(True)
plt.ion()
matplotlib.is_interactive()
```

Out[1]:
```
True
```

In [2]:
```
import plotly.graph_objs as go
import plotly.express as px
```

In [3]:
```
churn=pd.read_csv('C:/Users/WANJIKU/OneDrive/Documents/churn2.csv')
churn
```

Out[3]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	1	1	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	3	1	
3	4	15701354	Boni	699	France	Female	39	1	0.00	2	0	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	
...	...	...	...	...	...	...	...	...	...	...	...	...
9995	9996	15606229	Obijiaku	771	France	Male	39	5	0.00	2	1	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57369.61	1	1	
9997	9998	15584532	Liu	709	France	Female	36	7	0.00	1	0	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75075.31	2	1	
9999	10000	15628319	Walker	792	France	Female	28	4	130142.79	1	1	

10000 rows × 14 columns

In [4]:
```
churn.describe()
```

Out[4]:

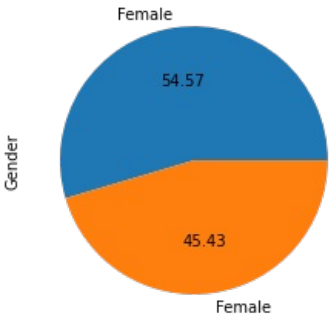
	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	10000.000000	10000.00000	10000.000000	
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288	1.530200	0.70550	0.515100	
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202	0.581654	0.45584	0.499797	
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000	1.000000	0.00000	0.000000	
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000	1.000000	0.00000	0.000000	
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000	1.000000	1.00000	1.000000	
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000	2.000000	1.00000	1.000000	
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000	4.000000	1.00000	1.000000	

In [5]:
```
churn.isnull().sum()

#our dataset has no missing data
```

Out[5]:
```
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 [6]:
```
churn['Gender'].value_counts().plot(kind='pie', autopct='%.2f', labels=churn['Gender'])
```

Out[6]:
```
<AxesSubplot:ylabel='Gender'>
```



In [7]:
```
#how does gender compare to churn?

dd=churn.groupby('Gender').sum()['Exited']
df=pd.DataFrame(dd)
df
```

Out[7]:

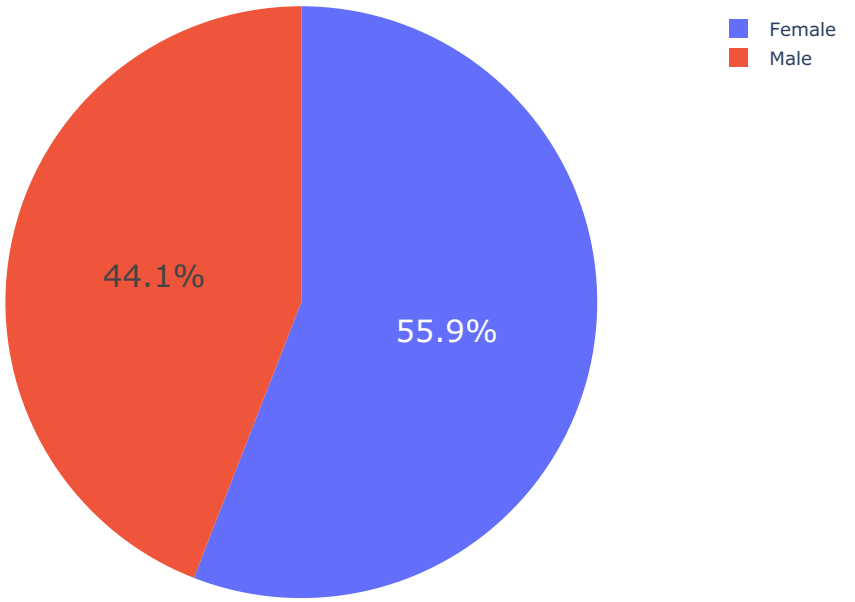
	Exited
Gender	
Female	1139
Male	898

In [8]:
```
df2=df.reset_index()
df2
```

Out[8]:

	Gender	Exited
0	Female	1139
1	Male	898

In [9]:
```
fig=px.pie(df2,values='Exited',names='Gender')
fig.update_traces(title_font=dict(size=25, family='Verdana', color='darkred'), hoverinfo='label+percent',textfo
fig.show()

#more females exited the bank compared to males despite the males being the majority of the population
```

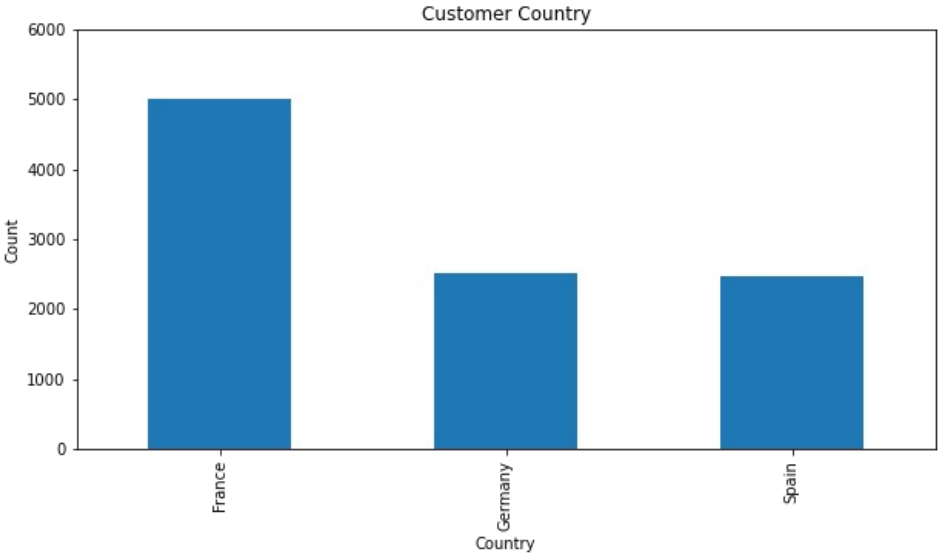


In [10]:
```
churn['Geography'].value_counts()
```

Out[10]:
```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

In [11]:
```
plt.figure(figsize=(10,5))
churn['Geography'].value_counts().plot(kind='bar')

plt.title('Customer Country')
plt.xlabel('Country')
plt.ylabel('Count')

plt.ylim(0, 6000)

#most of the customers reside in France
```

Out[11]:
```
(0.0, 6000.0)
```



In [12]:
```
cc=churn.groupby('Geography').sum()['Exited']
df3=pd.DataFrame(cc)
df3
```

Out[12]:

Geography	Exited
France	810
Germany	814
Spain	413

In [13]:
```
df4=df3.reset_index()
df4
```

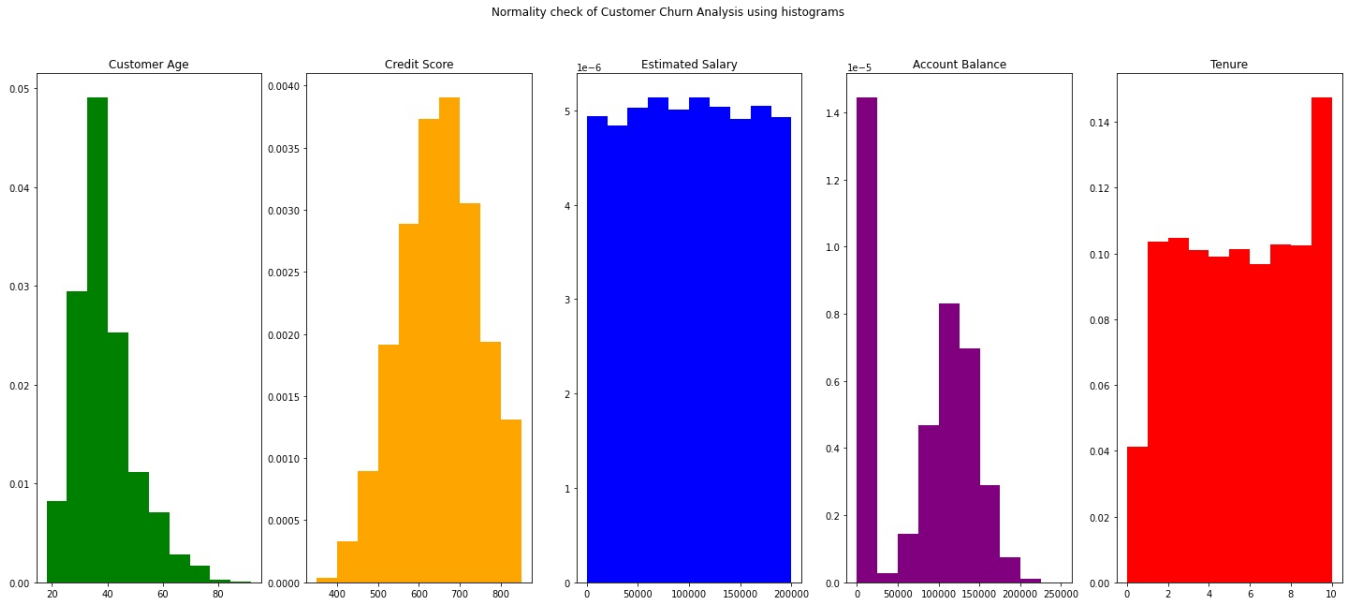
Out[13]:

	Geography	Exited
0	France	810
1	Germany	814
2	Spain	413

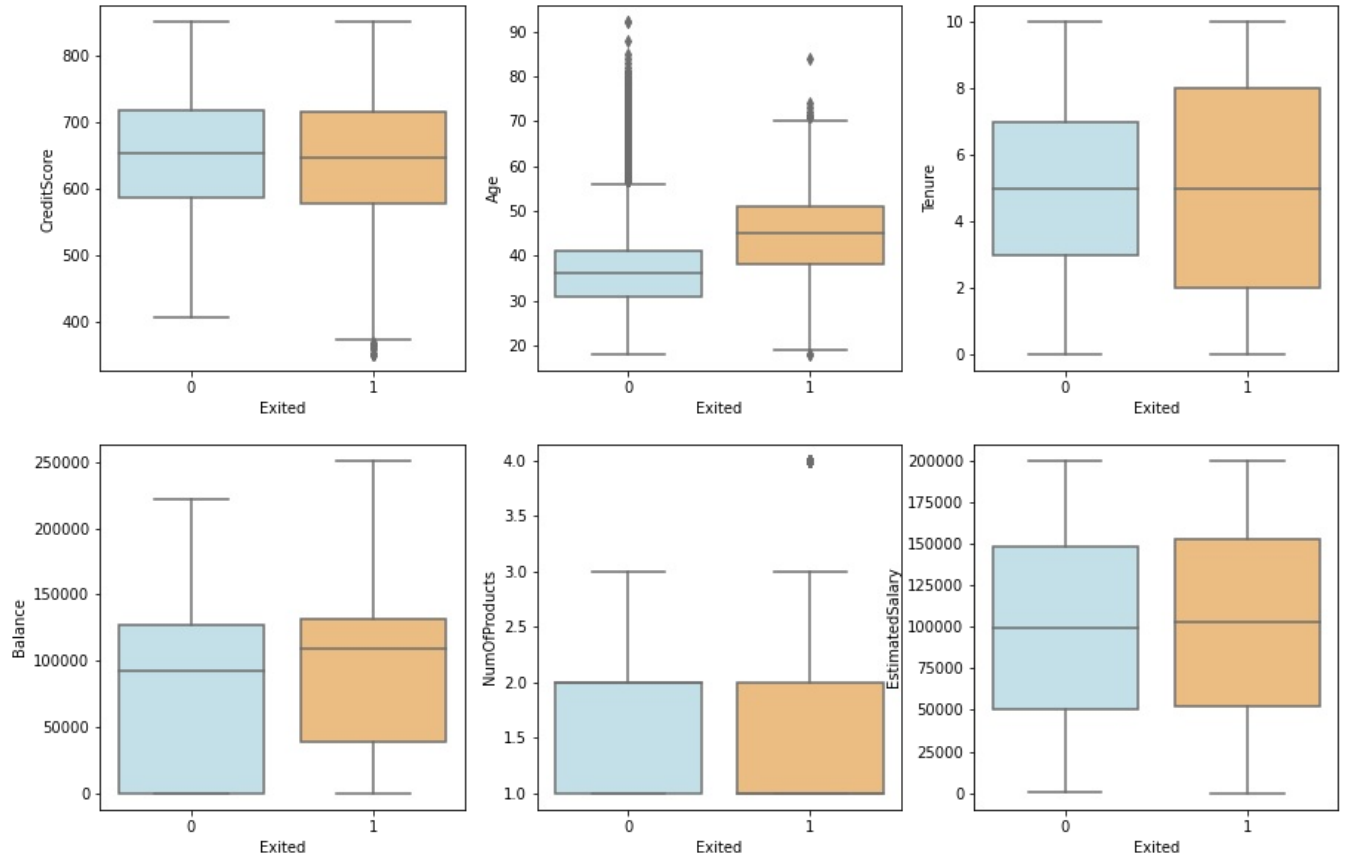
In [14]:
```
fig=px.pie(df4,values='Exited',names='Geography')
fig.update_traces(title_font=dict(size=25, family='Verdana', color='darkred'), hoverinfo='label+percent',textfo
fig.show()

#Germany has the highest churn rate among the 3 Geographies
```

In [15]:
```
#to check the normality of the distribution

plt.rcParams["figure.figsize"] = (25,10)
plt.xlim([0,100])


plt.subplot(1, 5, 1)
Age = plt.hist(churn['Age'], density = True, color = "green")
plt.title("Customer Age")

plt.subplot(1, 5, 2)
CreditScore = plt.hist(churn['CreditScore'], density = True, color = "orange")
plt.title("Credit Score")

plt.subplot(1, 5, 3)
Salary = plt.hist(churn['EstimatedSalary'], density = True, color = "blue")
plt.title("Estimated Salary")

plt.subplot(1, 5, 4)
AccountBalance = plt.hist(churn['Balance'], density = True, color = "purple")
plt.title("Account Balance")

plt.subplot(1, 5, 5)
Tenure = plt.hist(churn['Tenure'], density = True, color = "red")
plt.title("Tenure")


plt.suptitle("Normality check of Customer Churn Analysis using histograms")
```

```
plt.show()
```

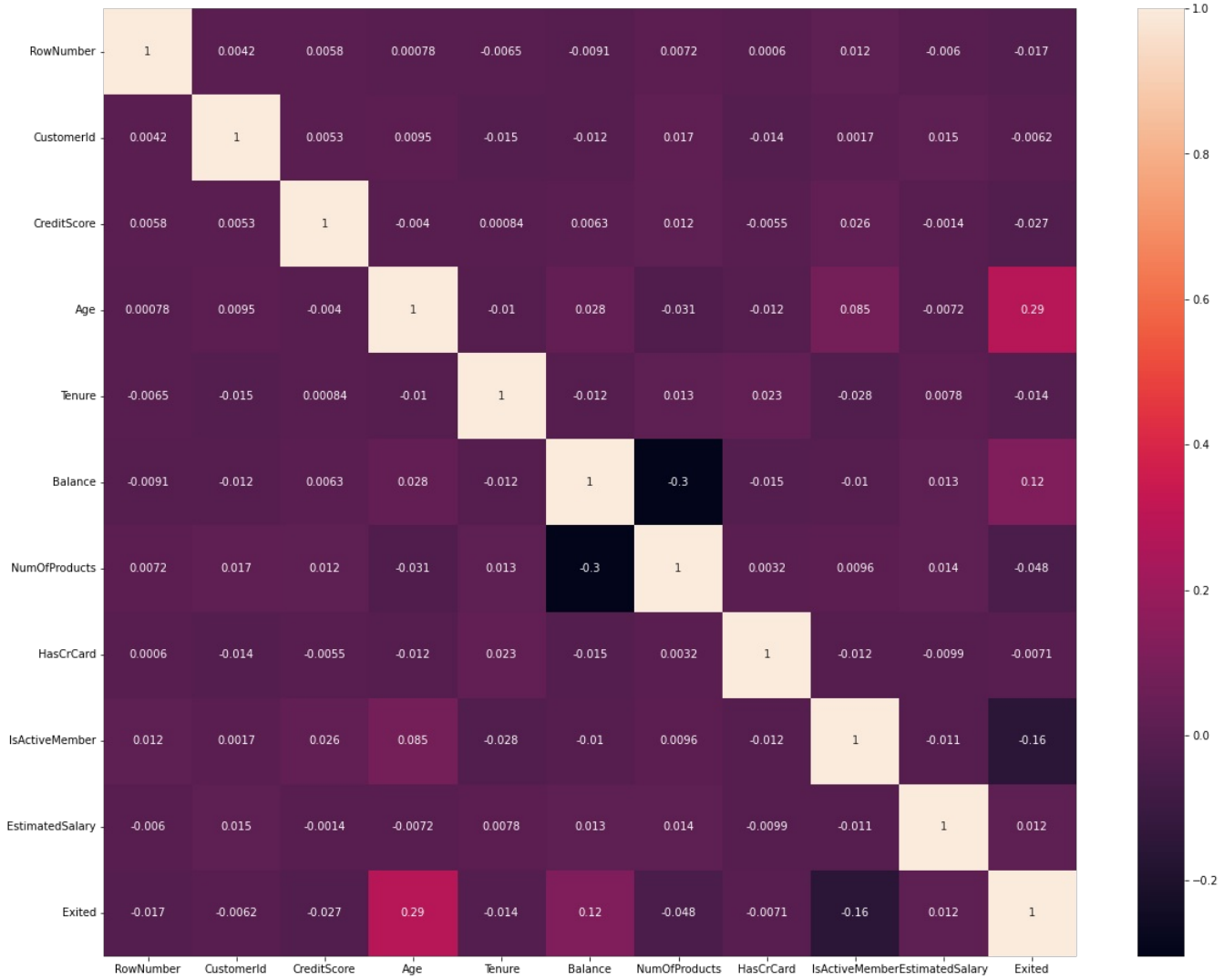


In [16]:
```
num_list = ['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']

fig = plt.figure(figsize=(15,10))

for i in range(len(num_list)):
    column=num_list[i]
    sub=fig.add_subplot(2,3, i+1)
    sns.boxplot(x='Exited', y=column, data=churn, palette='RdYlBu_r')

 #the boxplots show how the value of numerical features vary across the target group.
# for example age, tenure and account balance have distint difference when target is 0
# and when target is 1 suggesting that they are important predictors
#credit score, salary and number of products held appear to be less outstanding as the boxplot distribution is
#similar between target groups
```



In [17]:
```
plt.figure(figsize=(20,16))
sns.heatmap(churn.corr(), fmt='.2g', annot=True)
```

Out[17]:
```
<AxesSubplot:>
```



In [18]:
```
#check the number of unique values on object datatypes

churn.select_dtypes(include='object').nunique()
```

Out[18]:
```
Surname      2932
Geography       3
Gender          2
dtype: int64
```

In [19]:
```
churn[['Geography', 'Gender']] = churn[['Geography', 'Gender']].astype(str)
```

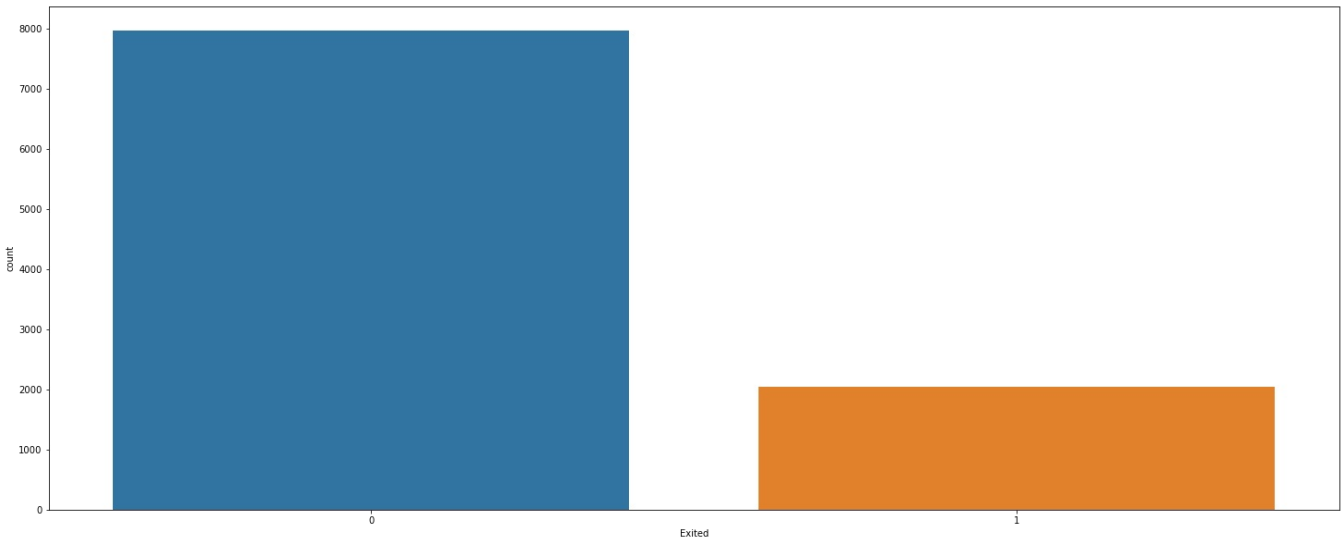
In [20]:
```
from sklearn import preprocessing
```

In [21]:
```
for col in churn.select_dtypes(include=['object']).columns:
    label_encoder=preprocessing.LabelEncoder()
    label_encoder.fit(churn[col].unique())
    churn[col]=label_encoder.transform(churn[col])
    print(f"{col}:{churn[col].unique()}")
```

```
Surname:[1115 1177 2040 ... 1366   44  363]
Geography:[0 2 1]
Gender:[0 1]
```

In [22]:
```
sns.countplot(churn['Exited'])

churn['Exited'].value_counts()

#our label is not balanced
```

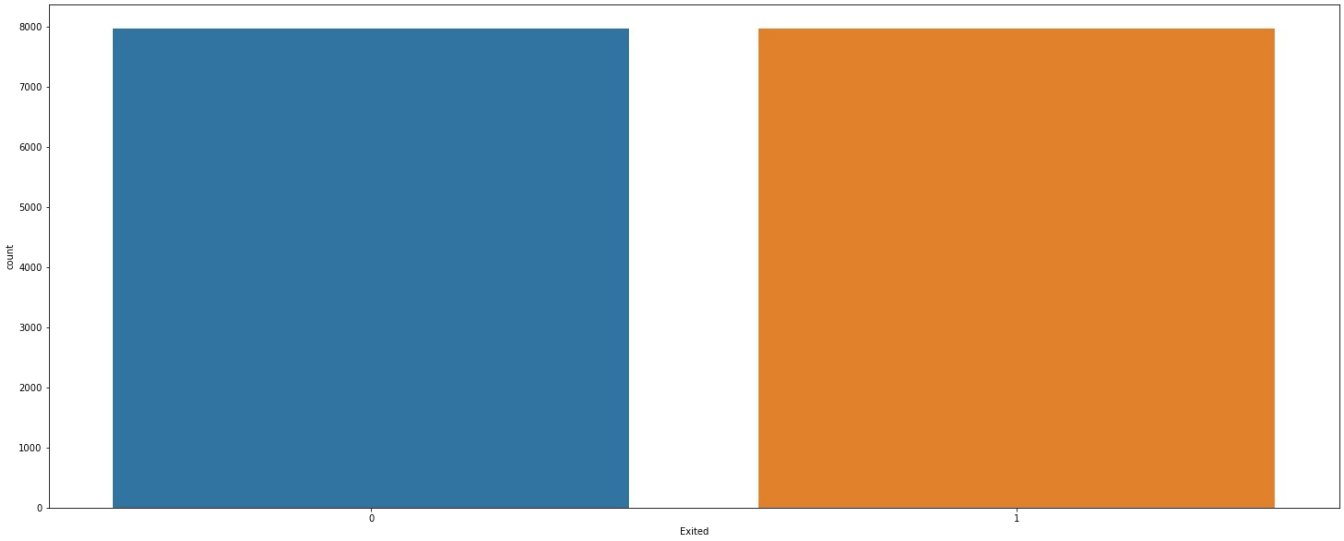
Out[22]:
```
0    7963
1    2037
Name: Exited, dtype: int64
```



In [23]:
```
#we oversample the minority class to balance the label

from sklearn.utils import resample

churn_majority=churn[(churn['Exited']==0)]
churn_minority=churn[(churn['Exited']==1)]

churn_minority_upsampled=resample(churn_minority,
                                  replace=True,
                                  n_samples=7963,
                                  random_state=0)

churn_upsampled=pd.concat([churn_minority_upsampled, churn_majority])
```

In [24]:
```
sns.countplot(churn_upsampled['Exited'])

churn_upsampled['Exited'].value_counts()

#the label is now balanced
```

Out[24]:
```
1    7963
0    7963
Name: Exited, dtype: int64
```



In [25]:
```
churn_upsampled.head()
```

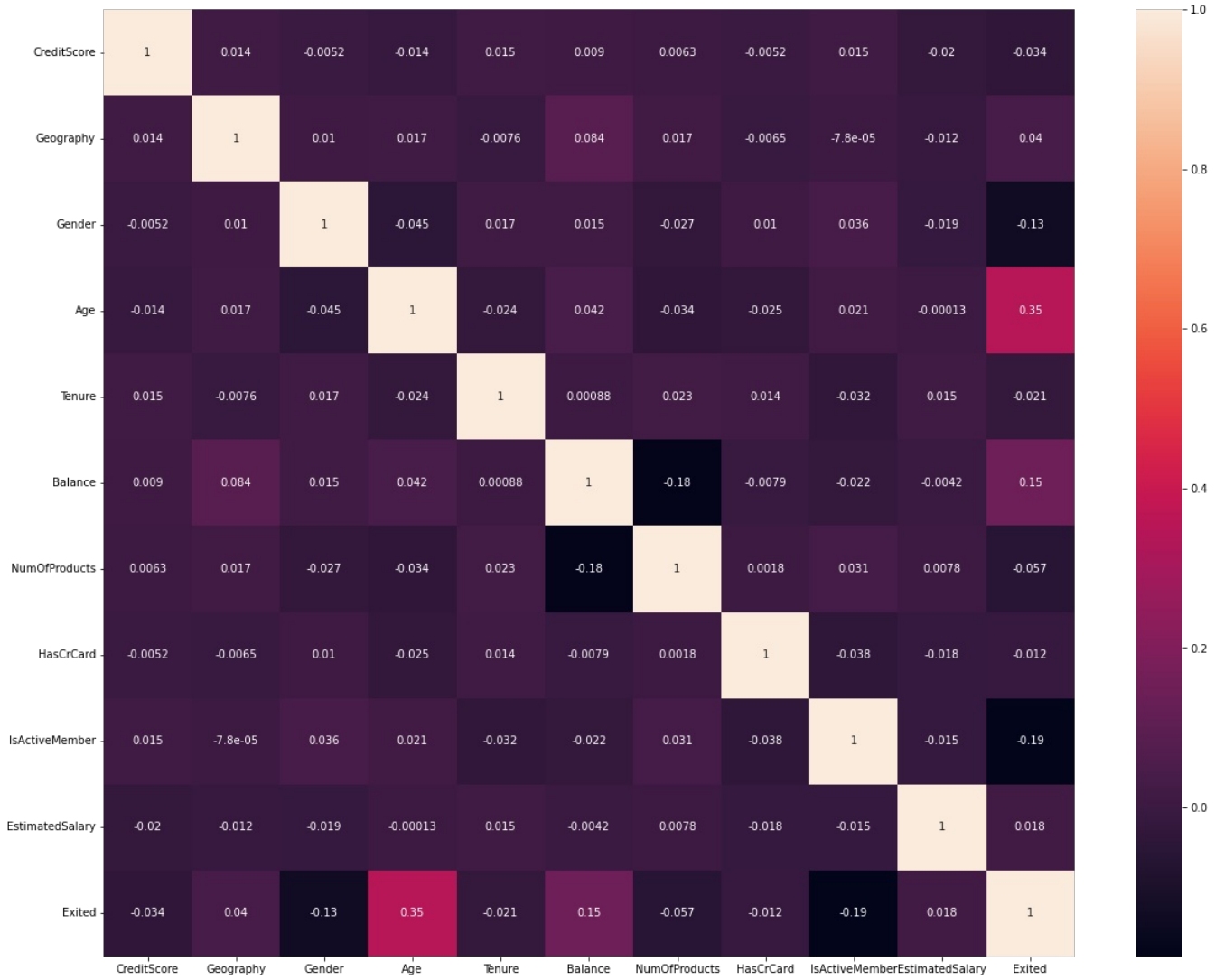
Out[25]:

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMe
3240	3241	15717560	1689	580	0	1	50	0	125647.36	1	1	
2629	2630	15711789	671	768	2	0	42	3	0.00	1	0	
8030	8031	15578141	493	592	2	1	38	3	0.00	1	1	
5853	5854	15765300	1470	596	1	1	40	5	62389.03	3	1	
3957	3958	15756610	409	657	1	0	38	5	123770.46	1	0	

In [26]:
```
churn_upsampled.drop(columns=(['RowNumber','CustomerId','Surname']), inplace=True)

#dropped columns that are not affecting our target variable
```

In [27]:
```
plt.figure(figsize=(20,16))
sns.heatmap(churn_upsampled.corr(), fmt='.2g', annot=True)

#checked correlation in our upsampled data
```

Out[27]:
```
<AxesSubplot:>
```



In [28]:
```
#train test split

X = churn_upsampled.drop('Exited', axis=1)
y = churn_upsampled['Exited']
```

Decision Tree

In [29]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
```

In [30]:
```
X_train, X_test, y_train, y_test = train_test_split (X,y, test_size=0.2, random_state=0)
```

In [31]:
```
#perform param grid to get the best hyper parameters for the model

dtree = DecisionTreeClassifier()

param_grid = { 'max_depth': [ 2,4,6,8],
               'min_samples_split': [2,4,6,8],
               'min_samples_leaf': [1,2,3,4],
               'max_features': ['auto', 'sqrt', 'log2'],
               'random_state': [0,7,42]}

grid_search = GridSearchCV (dtree, param_grid, cv=5)

grid_search.fit(X_train, y_train)
```

```
print (grid_search.best_params_)
```

```
{'max_depth': 8, 'max_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 8, 'random_state': 7}
```

In [32]:
```
#fit the model

dtree = DecisionTreeClassifier (max_depth=8, min_samples_leaf=1, min_samples_split=8, max_features='auto', rand

dtree.fit(X_train, y_train)
```

Out[32]:
```
DecisionTreeClassifier(max_depth=8, max_features='auto', min_samples_split=8,
                       random_state=7)
```

In [33]:
```
y_pred = dtree.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

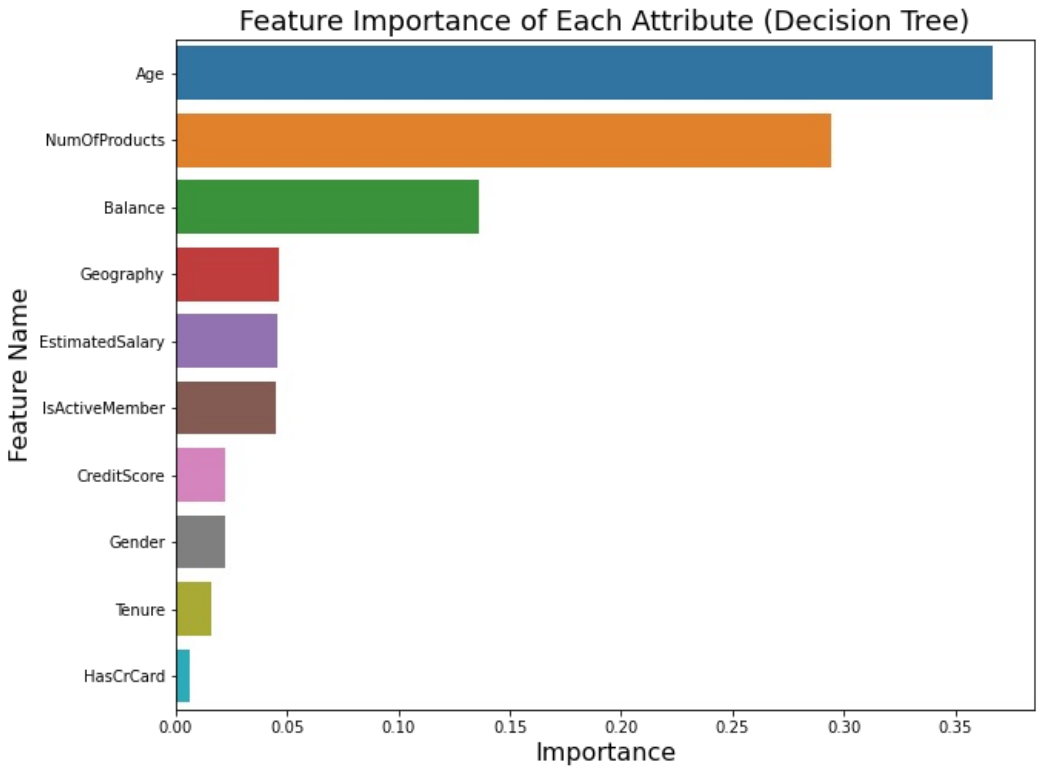
```
Accuracy Score: 78.0 %
```

In [34]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
```

In [35]:
```
print("F-1 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("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

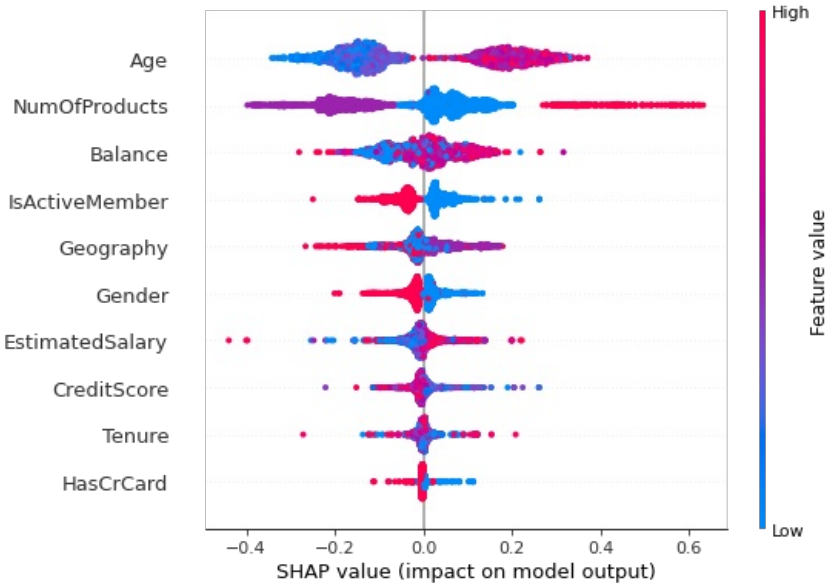
```
F-1 Score: 0.7799748901443817
Precision Score: 0.7799748901443817
Recall Score: 0.7799748901443817
Jaccard Score: 0.639310522253666
Log Loss: 7.5995049847261935
```

In [36]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": dtree.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Decision Tree)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```

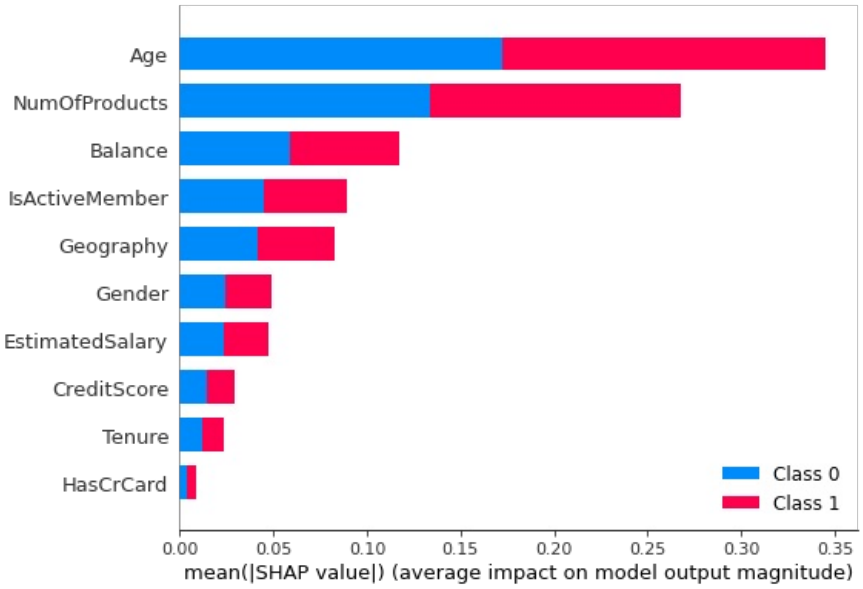


In [37]:
```
import shap
```

In [38]:
```
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)
```

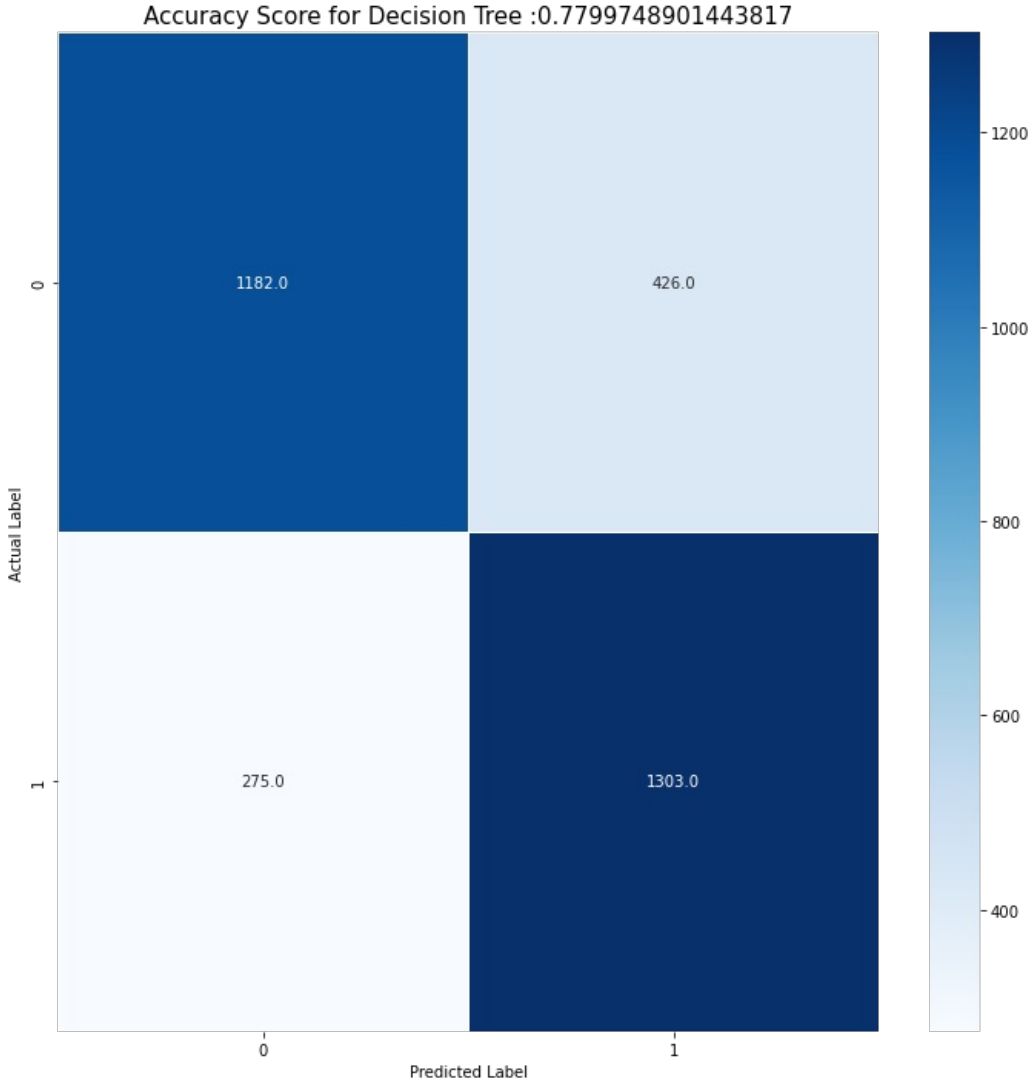


In [39]:
```
shap.summary_plot(shap_values, X_test)
```



In [40]:
```
from sklearn.metrics import confusion_matrix
```

In [41]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Blues')




plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Decision Tree :{0}'.format (dtree.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```

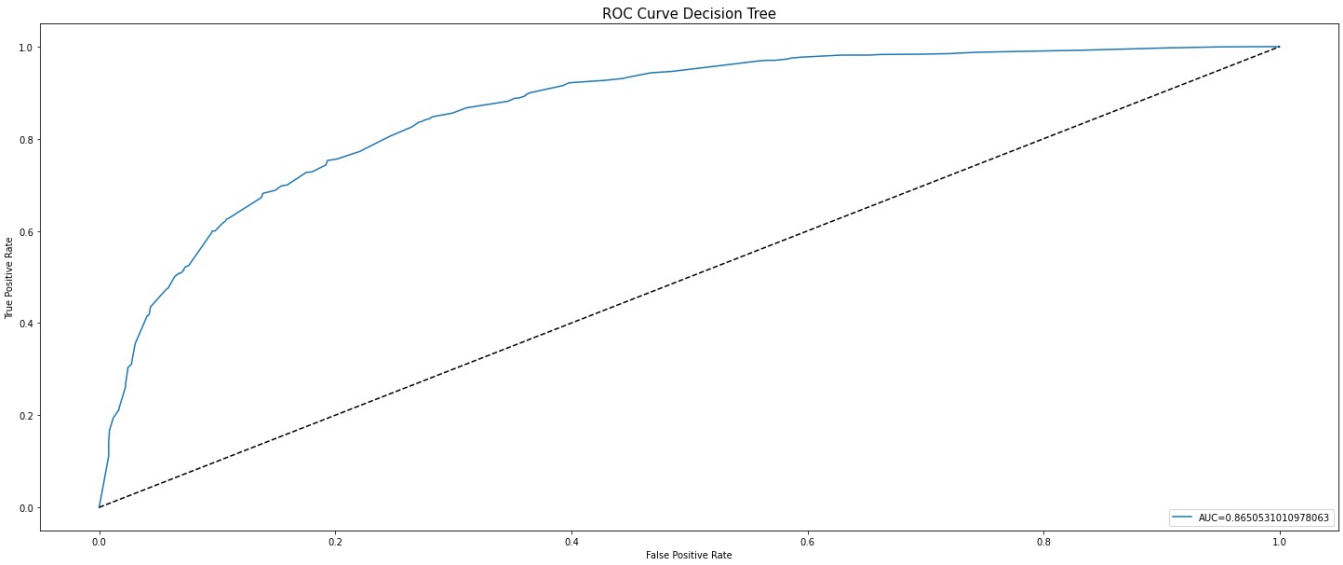


In [42]:
```
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn import metrics
```

In [43]:
```
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]
churn_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
churn_actual_predicted.index=y_test.index
```

In [44]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Decision Tree', size=15)
plt.legend(loc=4)
```

Out[44]:
```
<matplotlib.legend.Legend at 0x2a7e322a160>
```



Random Forest

In [45]:
```
from sklearn.ensemble import RandomForestClassifier

rfc=RandomForestClassifier()
```

In [46]:
```
rfc = RandomForestClassifier (max_depth=9, min_samples_leaf=10, min_samples_split=2, max_features='sqrt', rando

rfc.fit(X_train, y_train)
```

Out[46]:
```
RandomForestClassifier(max_depth=9, max_features='sqrt', min_samples_leaf=10,
                       random_state=0)
```

In [47]:
```
y_pred = rfc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

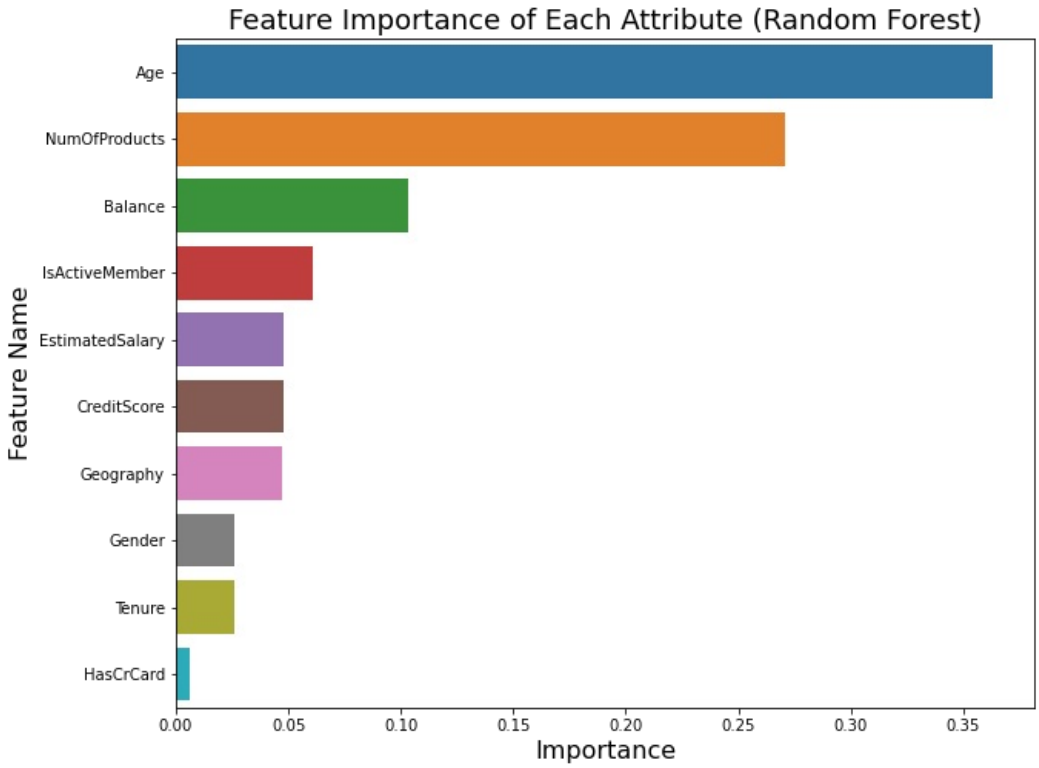
```
Accuracy Score: 82.11 %
```

In [48]:
```
print("F-1 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("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

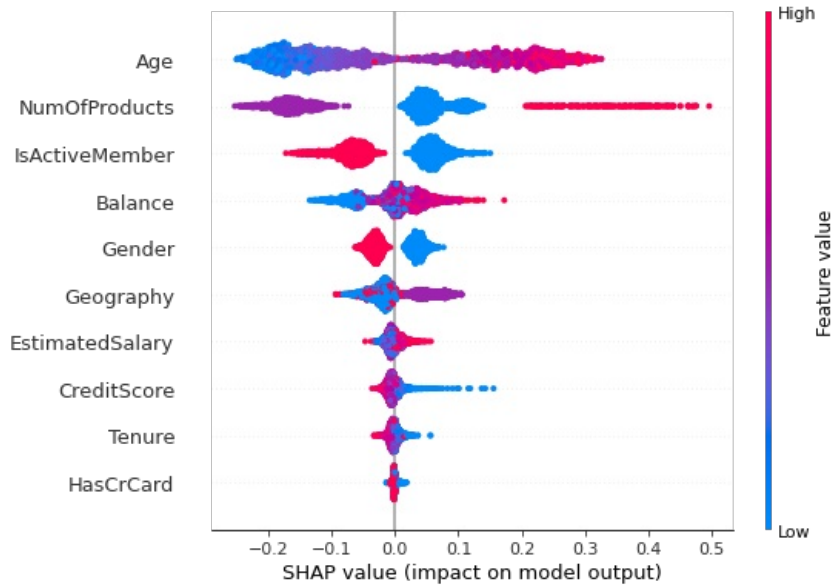
```
F-1 Score: 0.8210922787193974
Precision Score: 0.8210922787193974
Recall Score: 0.8210922787193974
Jaccard Score: 0.6964856230031949
Log Loss: 6.1793180295170185
```

In [49]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": rfc.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (Random Forest)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
```

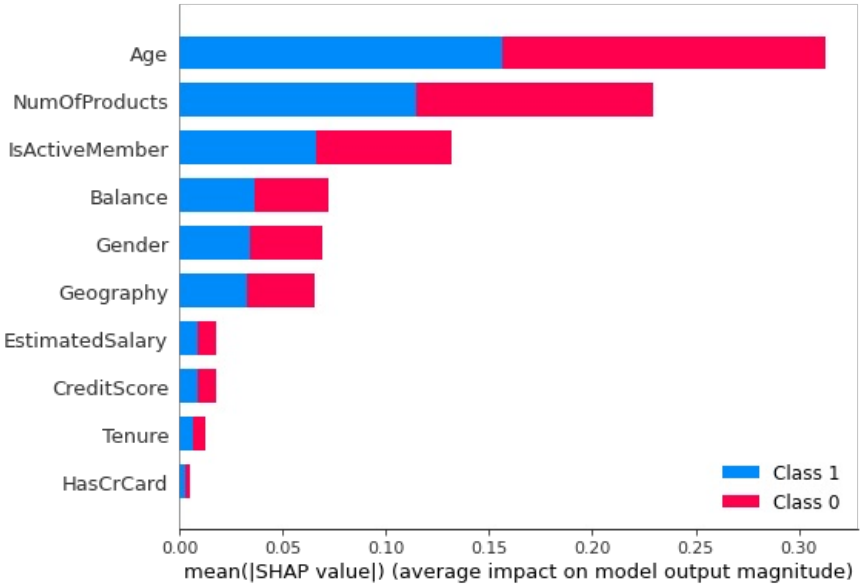
```
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```



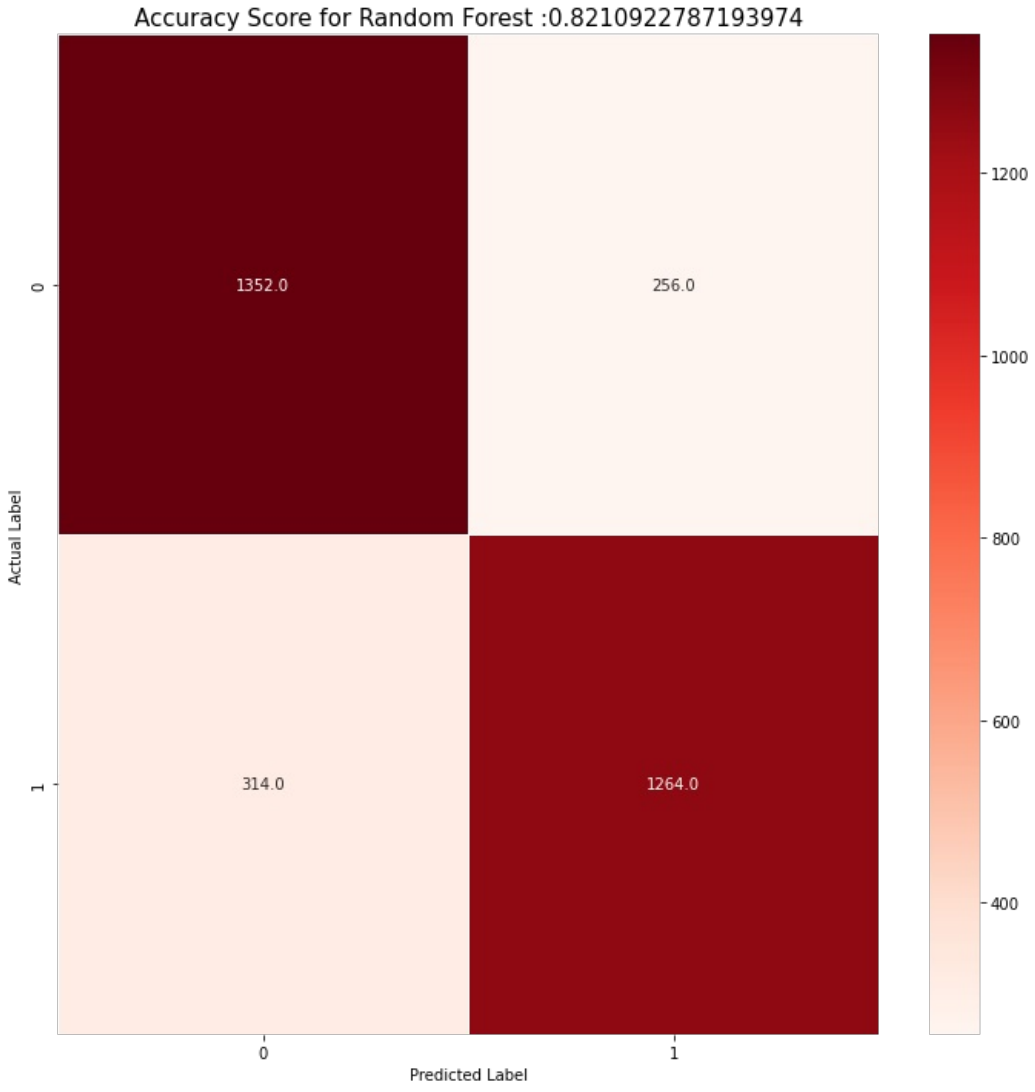
In [50]:
```
explainer=shap.TreeExplainer(rfc)

shap_values=explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names=X_test.columns)
```



In [51]:
```
shap.summary_plot(shap_values, X_test)
```



In [52]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Reds')



plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for Random Forest :{0}'.format (rfc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [53]:
```
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]
churn_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
churn_actual_predicted.index=y_test.index
churn_actual_predicted
```

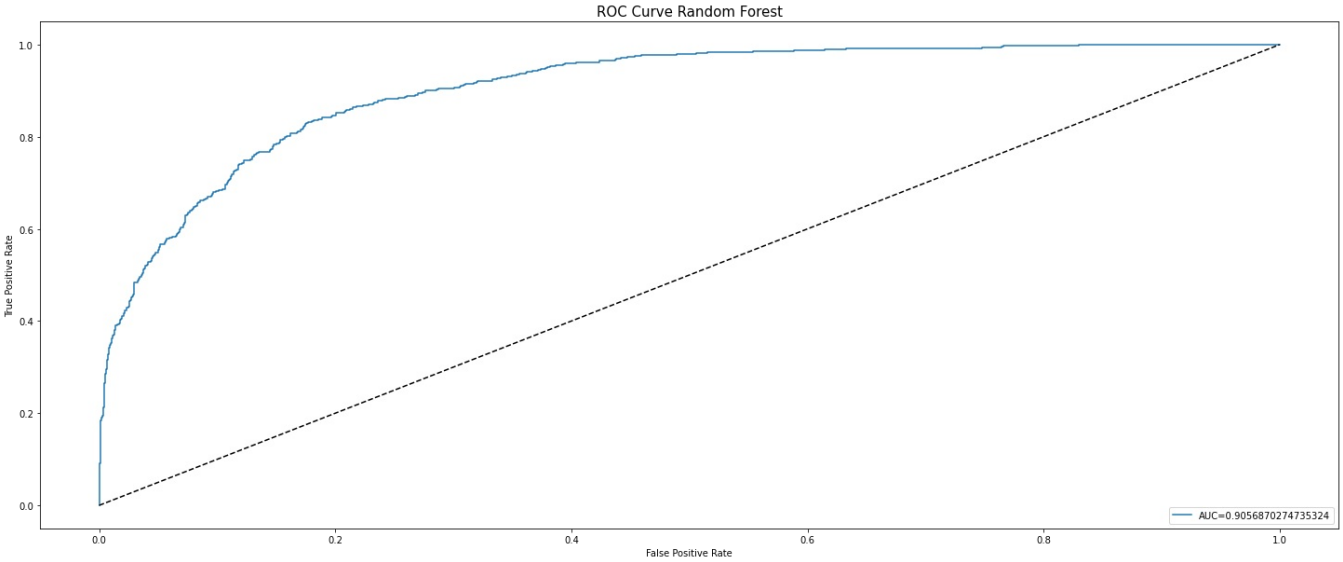
Out[53]:

	y_actual
3699	0
7056	0
43	1
5609	0
7797	0
...	...
6022	1
6913	1
4104	1
8519	0
6803	1

3186 rows × 1 columns

In [54]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)

plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Random Forest', size=15)
plt.legend(loc=4)
```

Out[54]:
```
<matplotlib.legend.Legend at 0x2a7e33a1220>
```



XGBoost

In [55]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\wanjiku\anaconda3\lib\site-packages (2.0.0)
Requirement already satisfied: numpy in c:\users\wanjiku\anaconda3\lib\site-packages (from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\wanjiku\anaconda3\lib\site-packages (from xgboost) (1.7.3)
```

In [56]:
```
from xgboost import XGBClassifier
```

In [57]:
```
xgbc=XGBClassifier()
```

In [58]:
```
xgbc = XGBClassifier (max_depth=9, min_samples_leaf=10, min_samples_split=2, max_features='sqrt', random_state=

xgbc.fit(X_train, y_train)
```

Out[58]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=9, max_features='sqrt',
              max_leaves=None, min_child_weight=None, min_samples_leaf=10,
              min_samples_split=2, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, ...)
```

In [59]:
```
y_pred = xgbc.predict(X_test)

print("Accuracy Score:", round(accuracy_score(y_test, y_pred)*100,2), "%")
```

```
Accuracy Score: 93.5 %
```

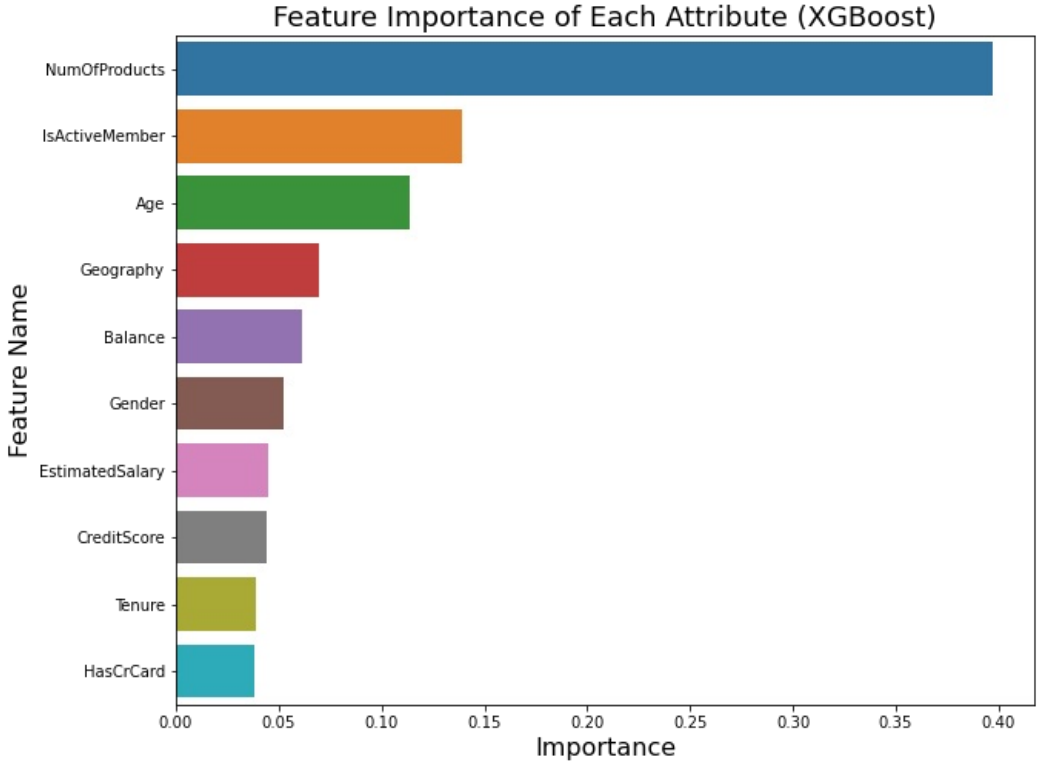
In [60]:
```
print("F-1 Score:", (f1_score(y_test, y_pred, average='micro')))
print("Precision Score:", (precision_score(y_test, y_pred, average='micro')))
print("Recall Score:", (recall_score(y_test, y_pred, average='micro')))
print("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

```
F-1 Score: 0.9350282485875706
Precision Score: 0.9350282485875706
Recall Score: 0.9350282485875706
Jaccard Score: 0.8779840848806366
Log Loss: 2.2440889651268763
```

In [61]:
```
print("F-1 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("Jaccard Score:", (jaccard_score(y_test, y_pred, average='micro')))
print("Log Loss:", (log_loss(y_test, y_pred)))
```

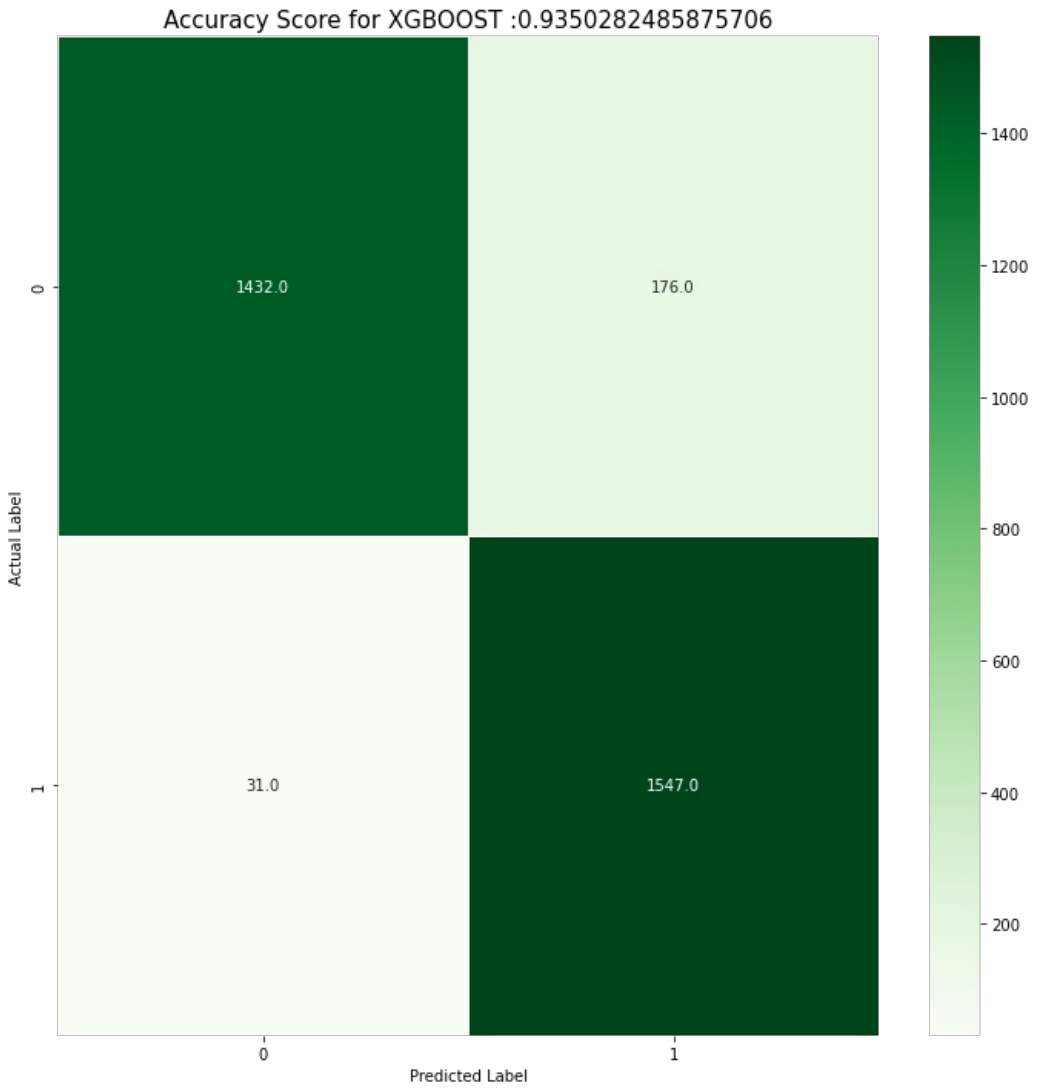
```
F-1 Score: 0.9350282485875706
Precision Score: 0.9350282485875706
Recall Score: 0.9350282485875706
Jaccard Score: 0.8779840848806366
Log Loss: 2.2440889651268763
```

In [62]:
```
imp_df = pd.DataFrame({ "Feature Name": X_train.columns,
                        "Importance": xgbc.feature_importances_})

fi= imp_df.sort_values(by = "Importance", ascending=False)

plt.figure(figsize=(10,8))
sns.barplot(data=fi, x= 'Importance', y= 'Feature Name')
plt.title('Feature Importance of Each Attribute (XGBoost)', fontsize=18)
plt.xlabel( 'Importance', fontsize=16)
plt.ylabel( 'Feature Name', fontsize=16)
plt.show()
```



In [63]:
```
cm = confusion_matrix (y_test, y_pred)

plt.figure(figsize=(10,10))
```

```
sns.heatmap ( data=cm,linewidths=.5, fmt='.1f', annot=True, cmap='Greens')



plt.ylabel('Actual Label')
plt.xlabel('Predicted Label')
all_sample_title = 'Accuracy Score for XGBOOST :{0}'.format (xgbc.score(X_test, y_test))
plt.title(all_sample_title, size=15)
plt.tight_layout()
```



In [64]:
```
y_pred_proba = xgbc.predict_proba(X_test)[:][:,1]
churn_actual_predicted=pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])])
churn_actual_predicted.index=y_test.index
churn_actual_predicted
```

Out[64]:

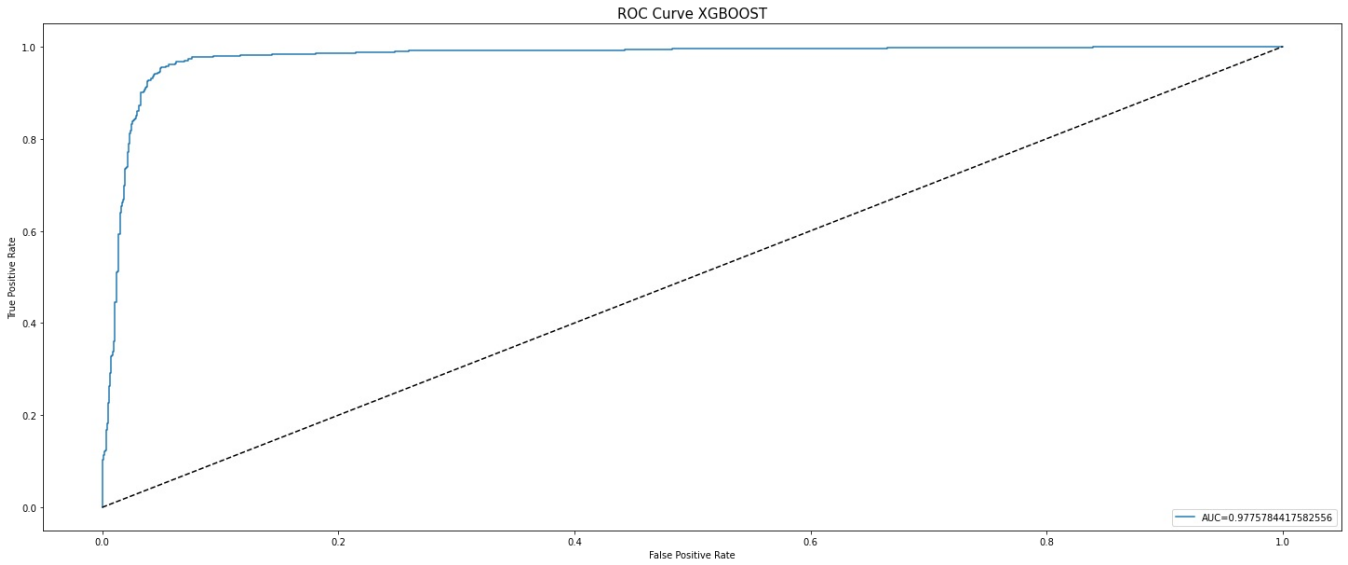
	y_actual
3699	0
7056	0
43	1
5609	0
7797	0
...	...
6022	1
6913	1
4104	1
8519	0
6803	1

3186 rows × 1 columns

In [65]:
```
fpr,tpr,_= metrics.roc_curve(y_test, y_pred_proba)

auc = metrics.roc_auc_score(y_test, y_pred_proba)
```

```
plt.plot(fpr, tpr, label='AUC='+str(auc))
plt.plot(fpr, fpr, linestyle='--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve XGBOOST', size=15)
plt.legend(loc=4)
```

Out[65]: <matplotlib.legend.Legend at 0x2a7e26509d0>



In [ ]:

In [ ]:

In [ ]:

In [ ]: