In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

In [2]:
```
df = pd.read_csv('Churn_Modelling.csv')
df.head(5)
```

Out[2]:

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

Data Preprocessing Part 1

In [3]:
```
df.select_dtypes(include='object').nunique()
```

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

In [4]:
```
# Remove 'Surname' column because it contains 2932 unique value
#its irrelevant to build machine learning model with this attribute
df.drop(columns='Surname', inplace=True)
df.head()
```

Out[4]:

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

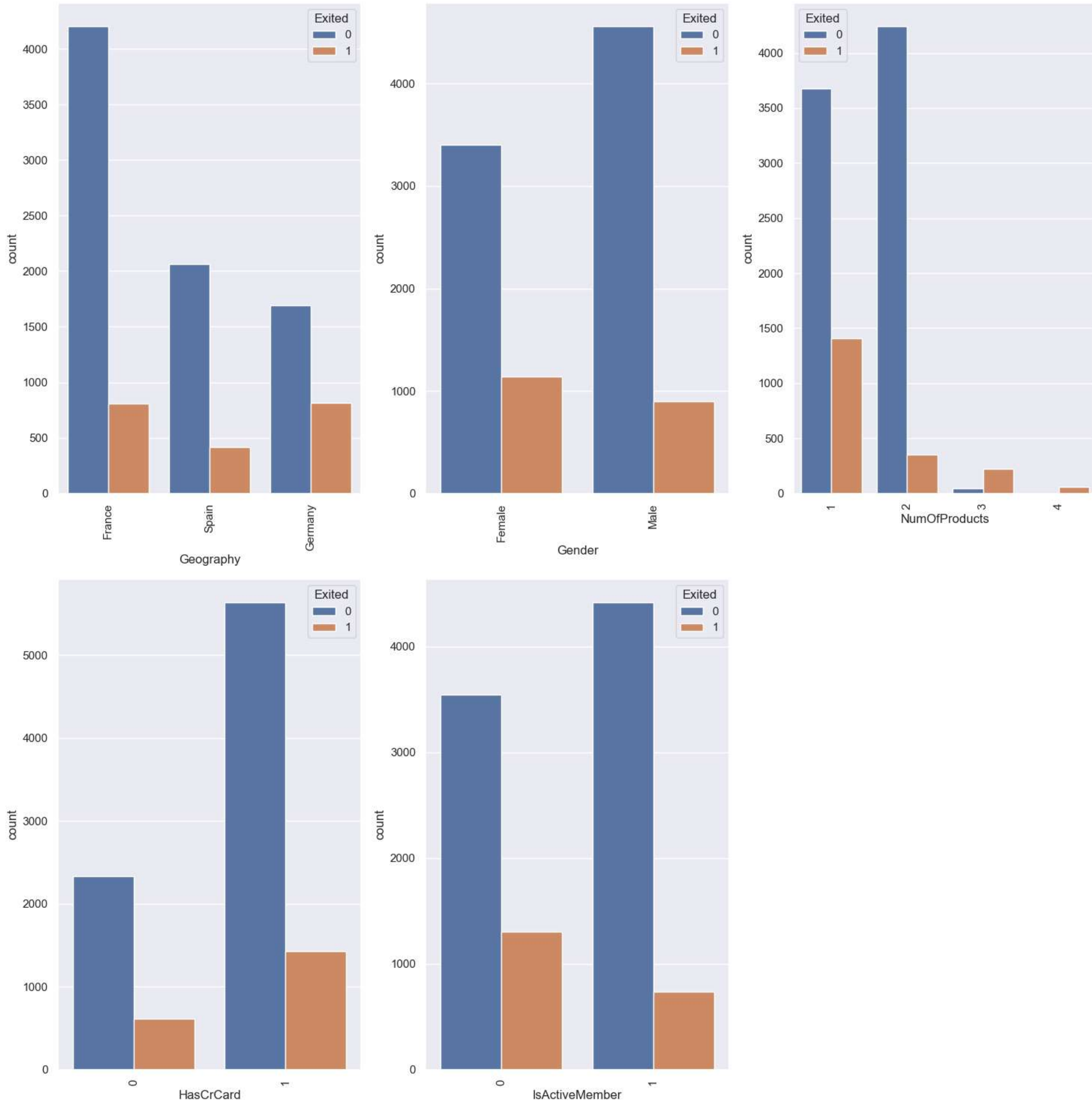
In [5]:
```
# Remove 'RowNumber' because its irrelevant for machine learning modelling
df.drop(columns='RowNumber', inplace=True)
df.head()
```

Out[5]:

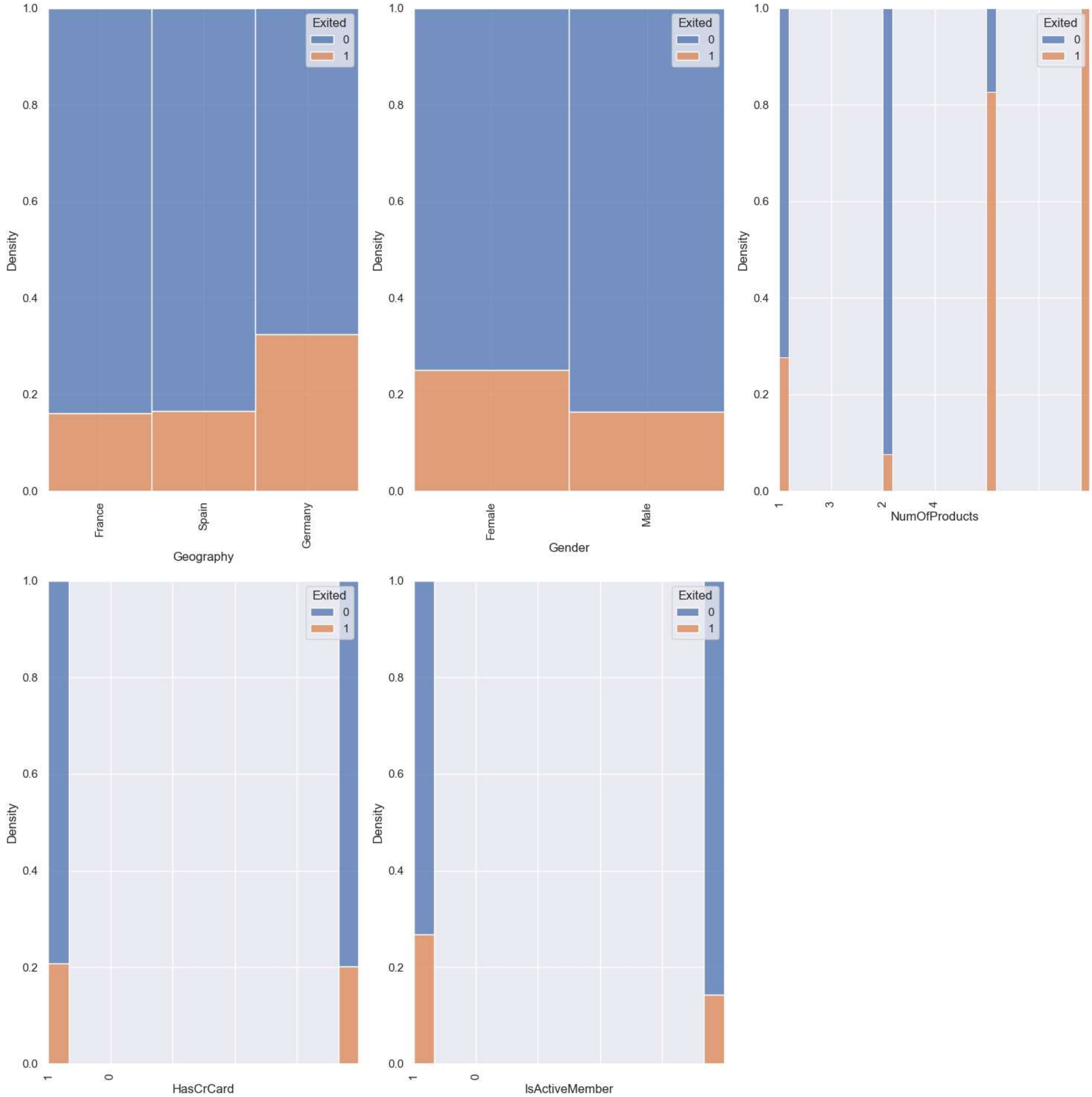
	CustomerId	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	15634602	619	France	Female	42	2	0.00	1	1	
1	15647311	608	Spain	Female	41	1	83807.86	1	0	
2	15619304	502	France	Female	42	8	159660.80	3	1	
3	15701354	699	France	Female	39	1	0.00	2	0	
4	15737888	850	Spain	Female	43	2	125510.82	1	1	

Exploratory Data Analysis

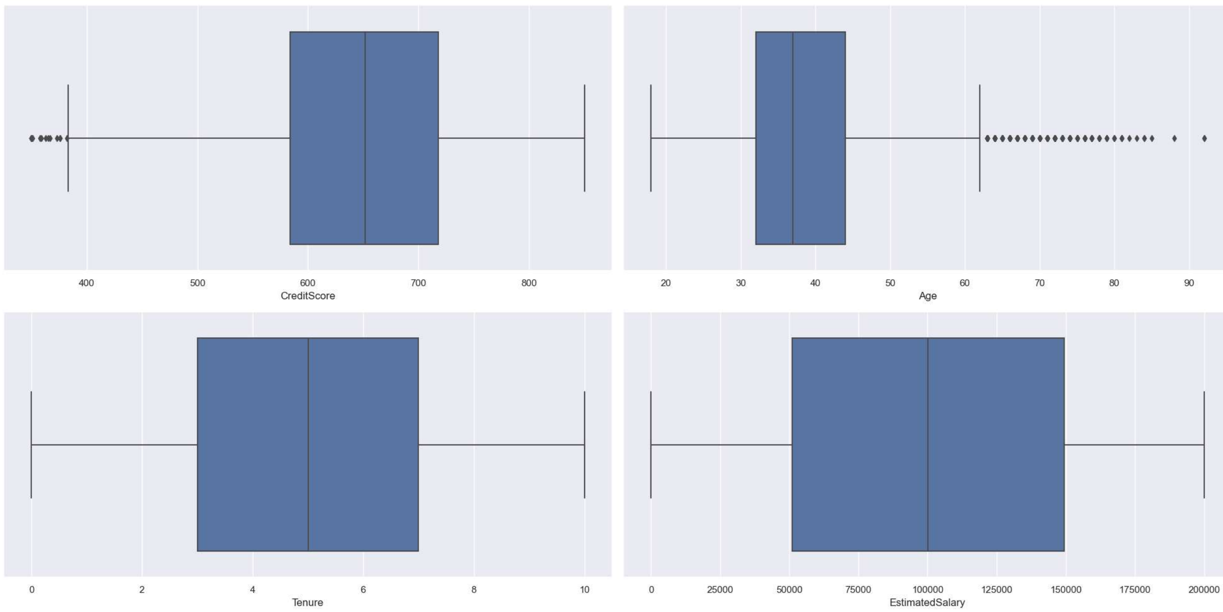
In [6]:
```
# list of categorical variables to plot
cat_vars = ['Geography', 'Gender', 'NumOfProducts', 'HasCrCard', 'IsActiveMember']

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

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='Exited', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# remove the sixth subplot
fig.delaxes(axs[5])

# show plot
plt.show()
```


Exited
0
1
count
Geography
France
Spain
Germany
Gender
Female
Male
NumOfProducts
HasCrCard
IsActiveMember

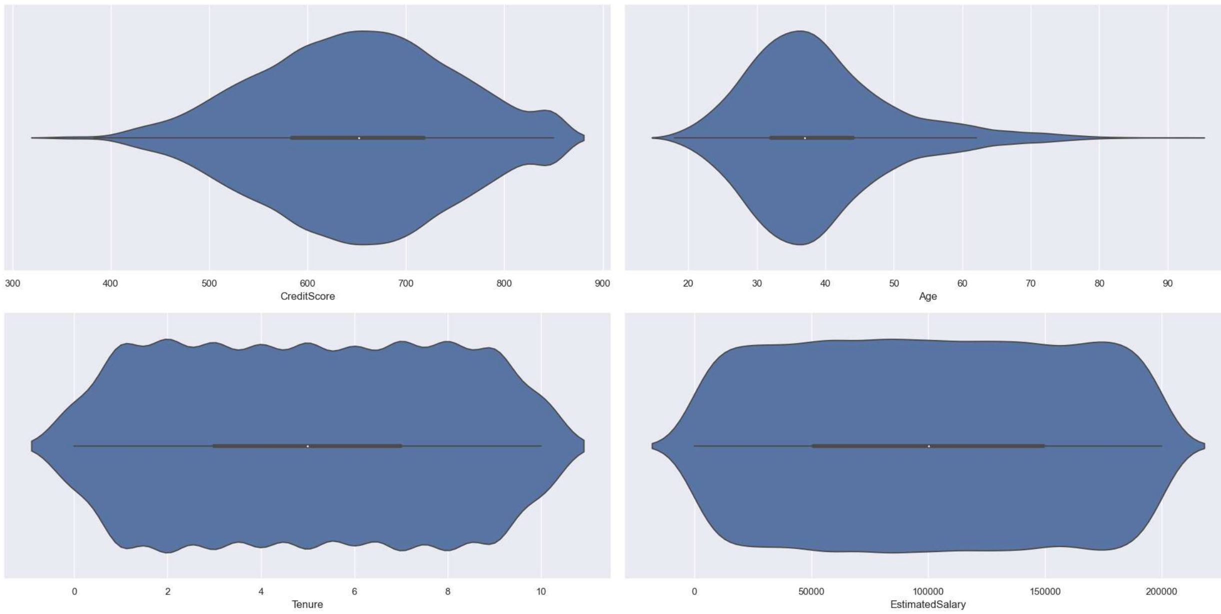
In [7]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Geography', 'Gender', 'NumOfProducts', 'HasCrCard', 'IsActiveMember']

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

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Exited', data=df, ax=axs[i], multiple="fill", kde=False,
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# remove the sixth subplot
fig.delaxes(axs[5])

# show plot
plt.show()
```


Density
Exited
0
1
France
Spain
Germany
Geography
Female
Male
Gender
NumOfProducts
HasCrCard
IsActiveMember

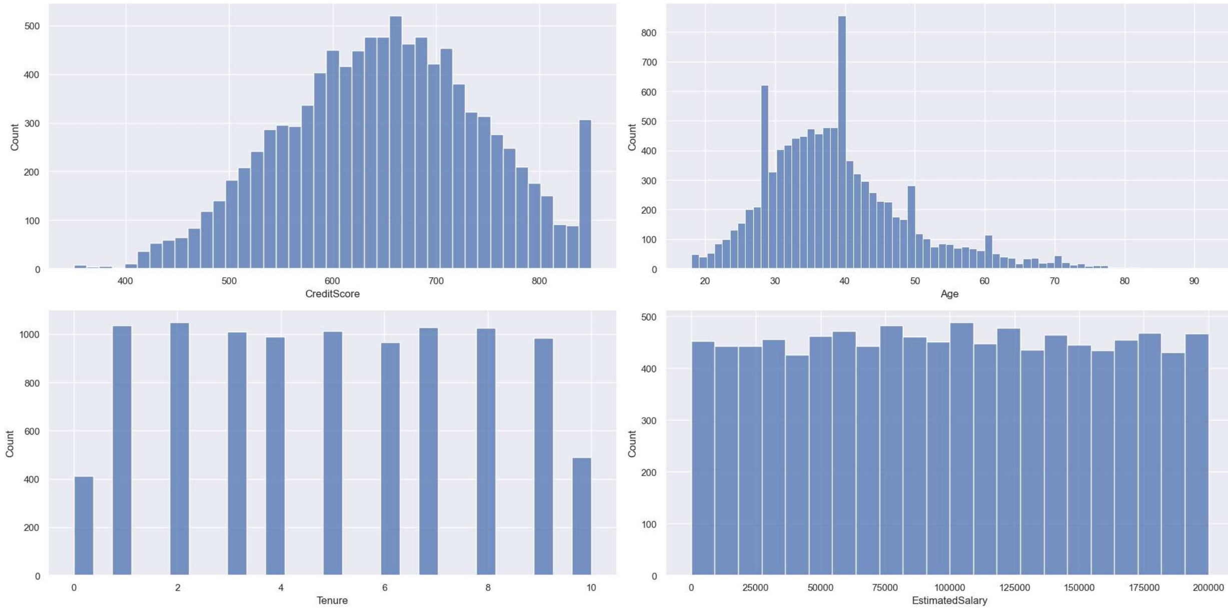
In [8]:
```
num_vars = ['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



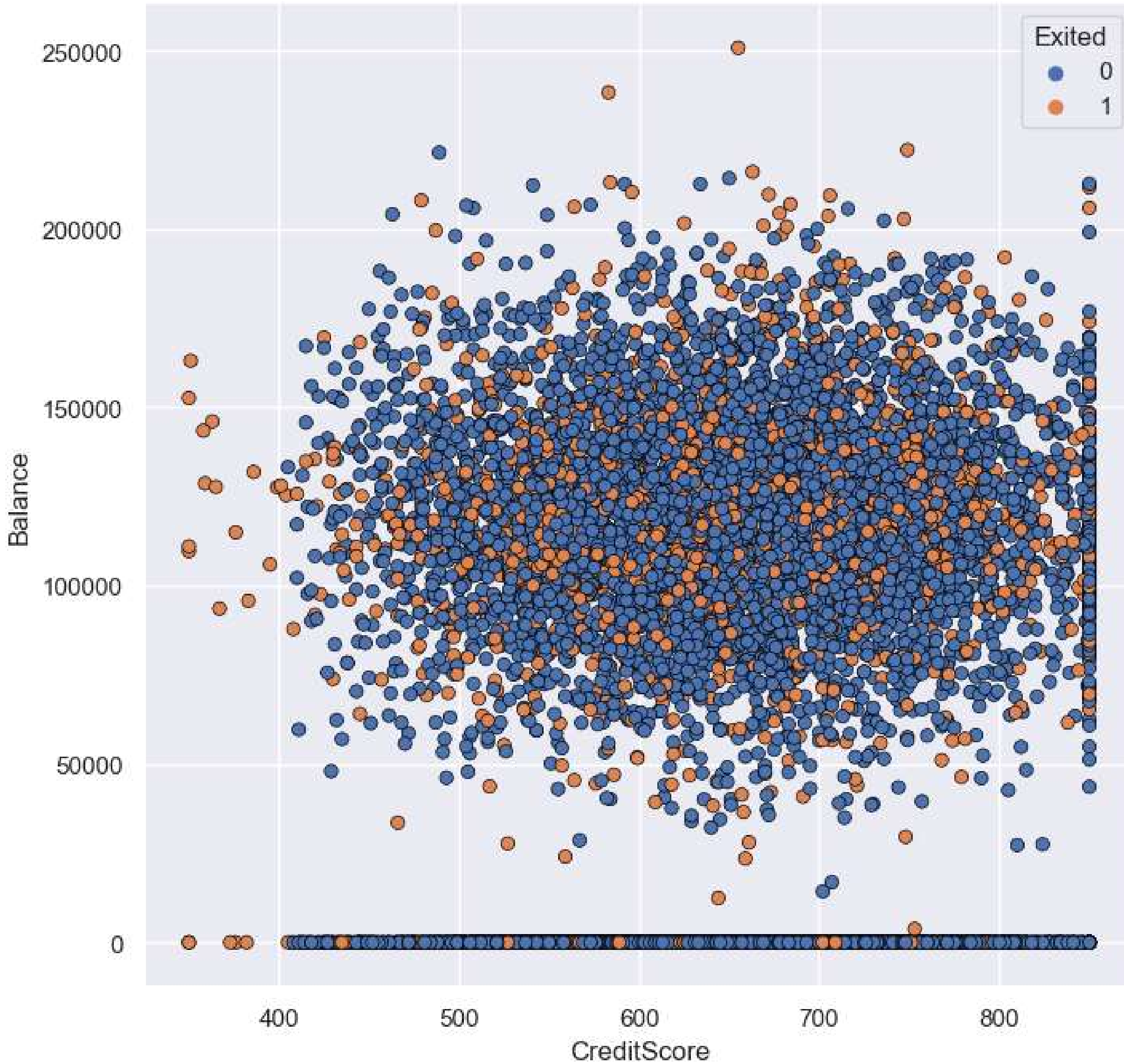
In [11]:
```
num_vars = ['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.violinplot(x=var, data=df, ax=axs[i])

fig.tight_layout()
plt.show()
```



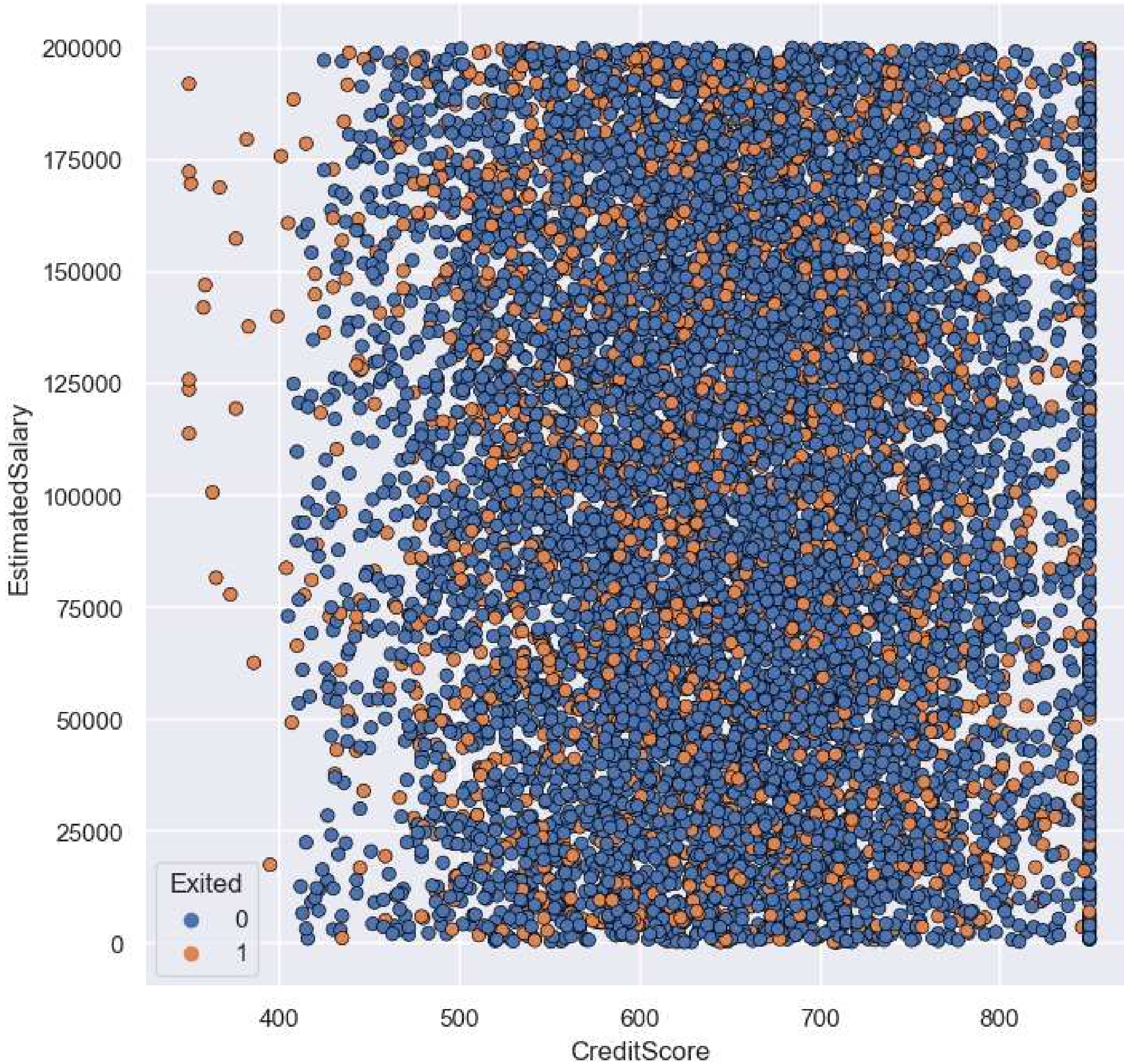
In [15]:
```
num_vars = ['CreditScore', 'Age', 'Tenure', 'EstimatedSalary']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.histplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



In [16]:
```
plt.figure(figsize=(8,8),dpi=100)
sns.scatterplot(x="CreditScore", y="Balance", hue="Exited", data=df, edgecolor="black
plt.show()
```



In [17]:
```
plt.figure(figsize=(8,8),dpi=100)
sns.scatterplot(x="CreditScore", y="EstimatedSalary", hue="Exited", data=df, edgecolor
plt.show()
```



Data Preprocessing Part 2

In [18]:
```
# Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[18]:
```
Series([], dtype: float64)
```

In [19]:
```
df.head()
```

Out[19]:

	CustomerId	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	Is
0	15634602	619	France	Female	42	2	0.00	1	1	
1	15647311	608	Spain	Female	41	1	83807.86	1	0	
2	15619304	502	France	Female	42	8	159660.80	3	1	
3	15701354	699	France	Female	39	1	0.00	2	0	
4	15737888	850	Spain	Female	43	2	125510.82	1	1	

In [20]:
```
# Remove CustomerId because its irrelevant
df.drop(columns='CustomerId', inplace=True)
df.head()
```

Out[20]:

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

Label Encoding for Object datatype

In [21]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Geography: ['France' 'Spain' 'Germany']
Gender: ['Female' 'Male']
```

In [22]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Geography: [0 2 1]
Gender: [0 1]
```

Remove Outlier Using Z-Score

In [23]:
```
df.shape
```

Out[23]: (10000, 11)

In [24]:
```
from scipy import stats

# define a function to remove outliers using z-score for only selected numerical colur
def remove_outliers(df, cols, threshold=3):
    # loop over each selected column
    for col in cols:
        # calculate z-score for each data point in selected column
        z = np.abs(stats.zscore(df[col]))
        # remove rows with z-score greater than threshold in selected column
        df = df[(z < threshold) | (df[col].isnull())]
    return df
```

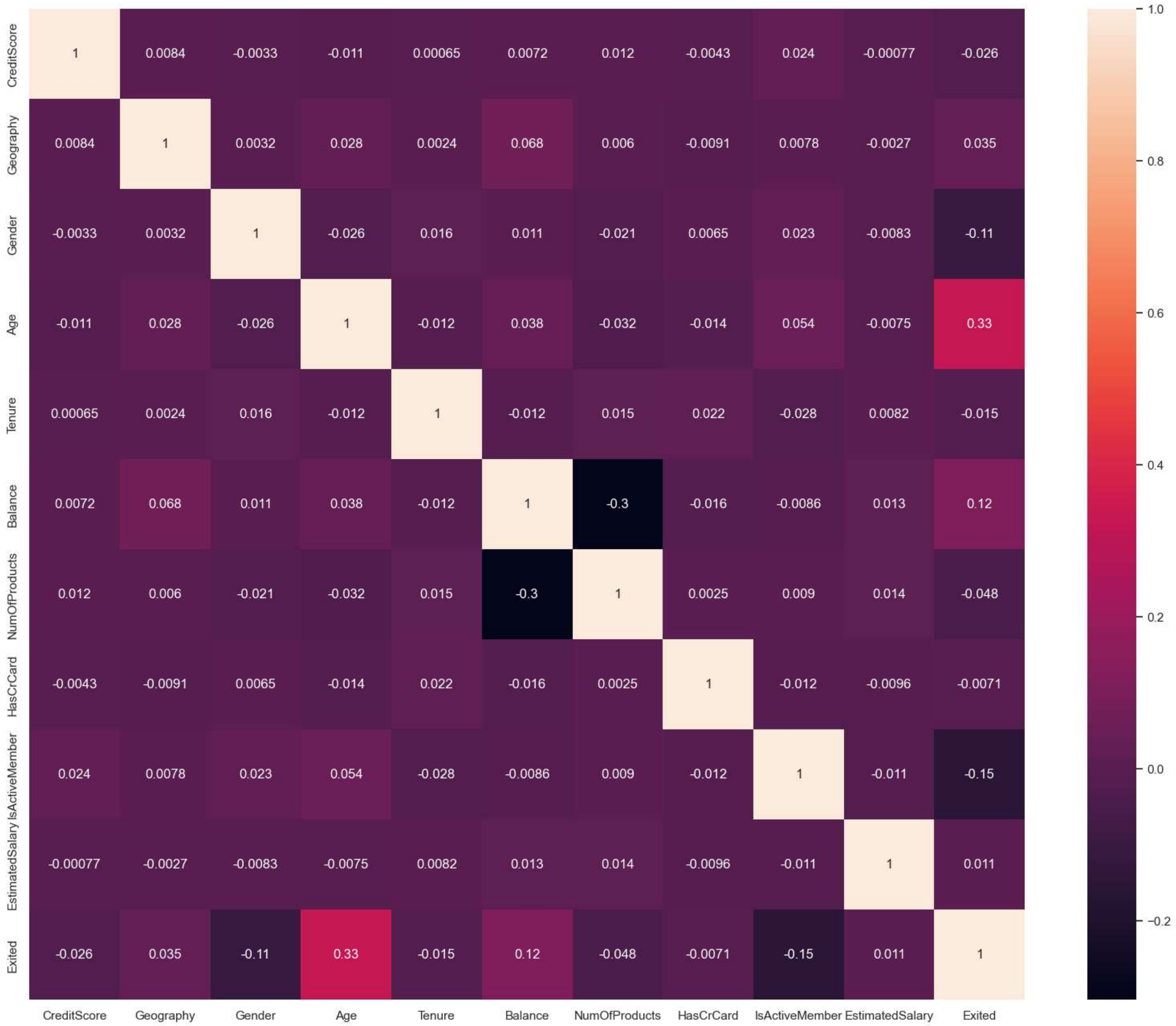
In [25]:
```
selected_cols = ['Age']
df_clean = remove_outliers(df, selected_cols)
df_clean.shape
```

Out[25]: (9867, 11)

Correlation heatmap

In [26]:
```
#Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df_clean.corr(), fmt='.2g', annot=True)
```

Out[26]:
```
<AxesSubplot:>
```



Train Test Split

In [27]:
```
X = df_clean.drop('Exited', axis=1)
y = df_clean['Exited']
```

In [28]:
```
#test size 20% and train size 80%
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2,random_state=0
```

Decision Tree

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = 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]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 5, 'min_samples_leaf': 2, 'min_samples_split': 2, 'random_state': 0}
```

In [31]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=5, min_samples_leaf=2, min_s
dtree.fit(X_train, y_train)
```

Out[31]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=5, min_samples_leaf=2,
                       random_state=0)
```

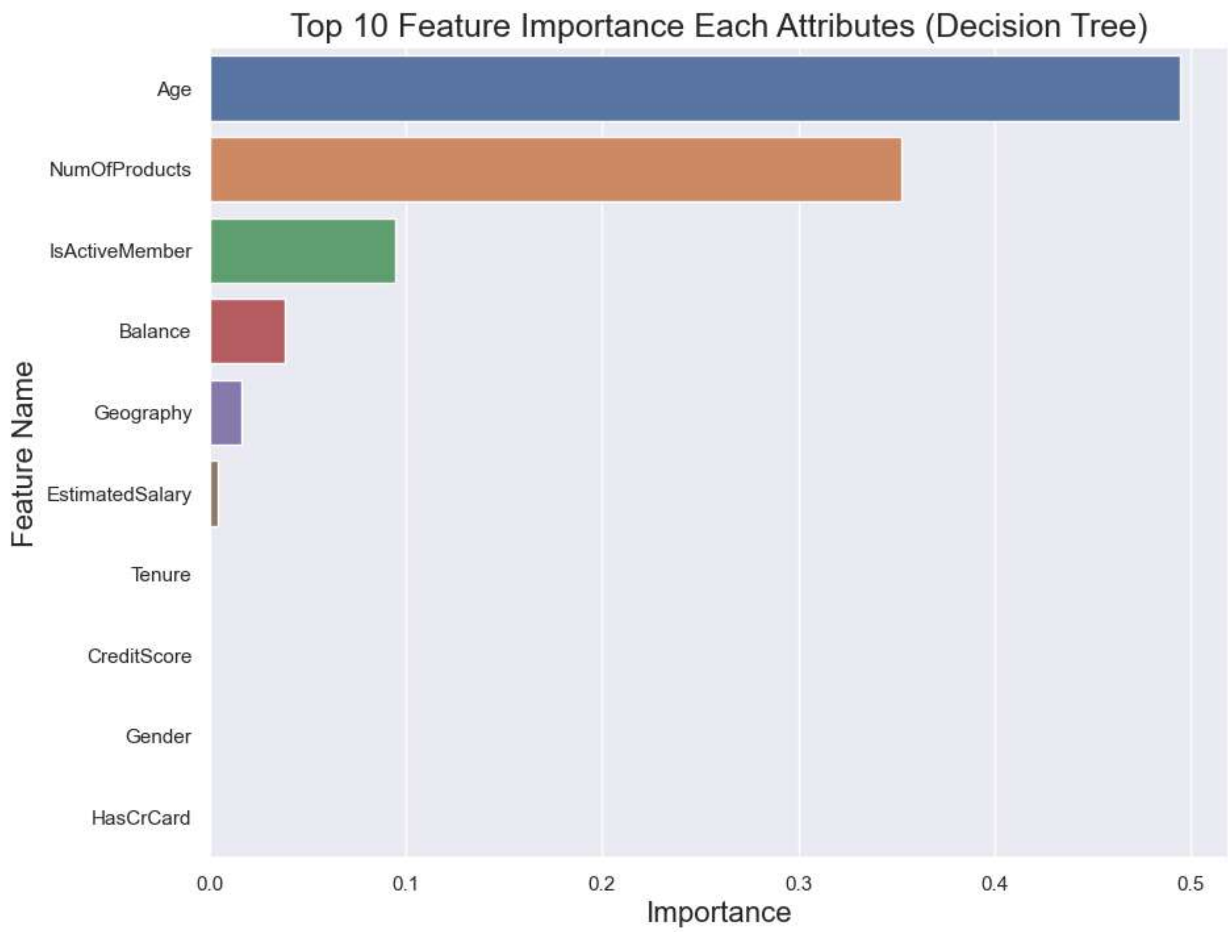
In [32]:
```
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 77.66 %
```

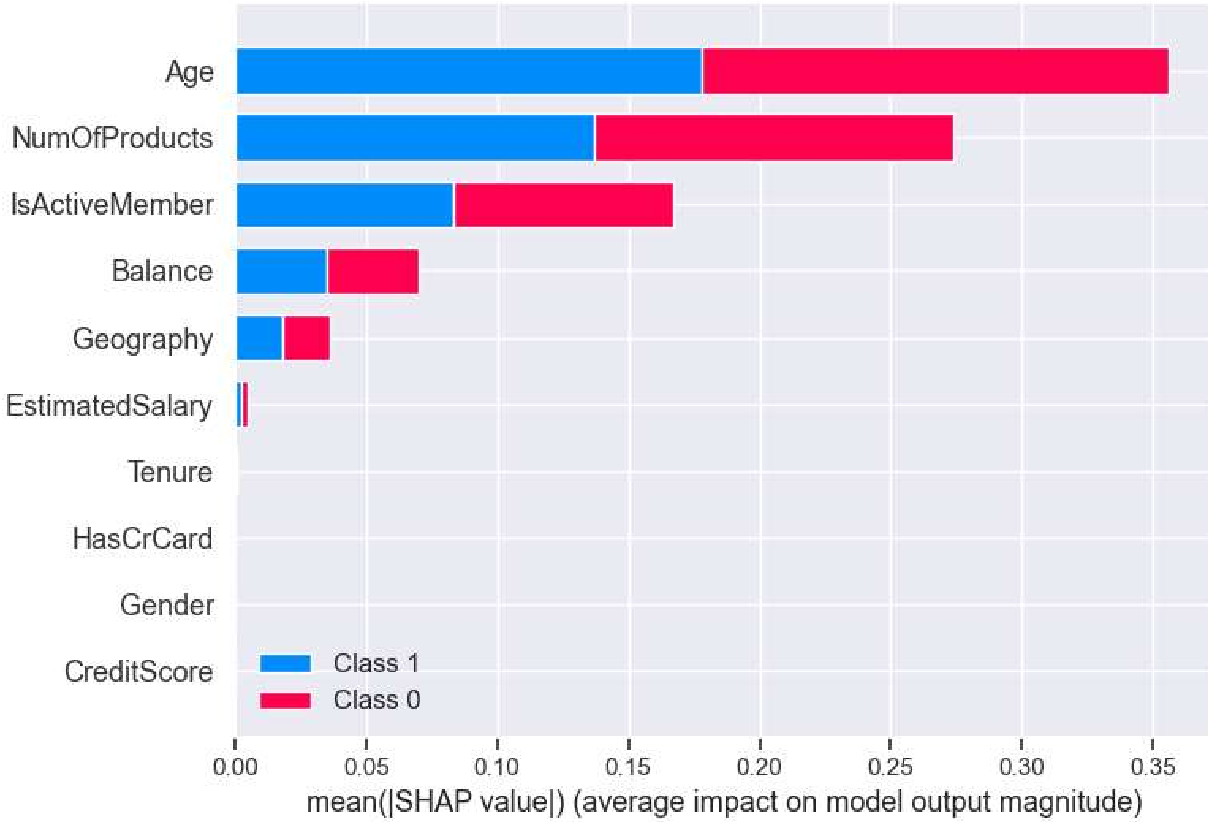
In [33]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, j
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.776595744680851
Precision Score :  0.776595744680851
Recall Score :  0.776595744680851
Jaccard Score :  0.6347826086956522
Log Loss :  7.716230329376803
```

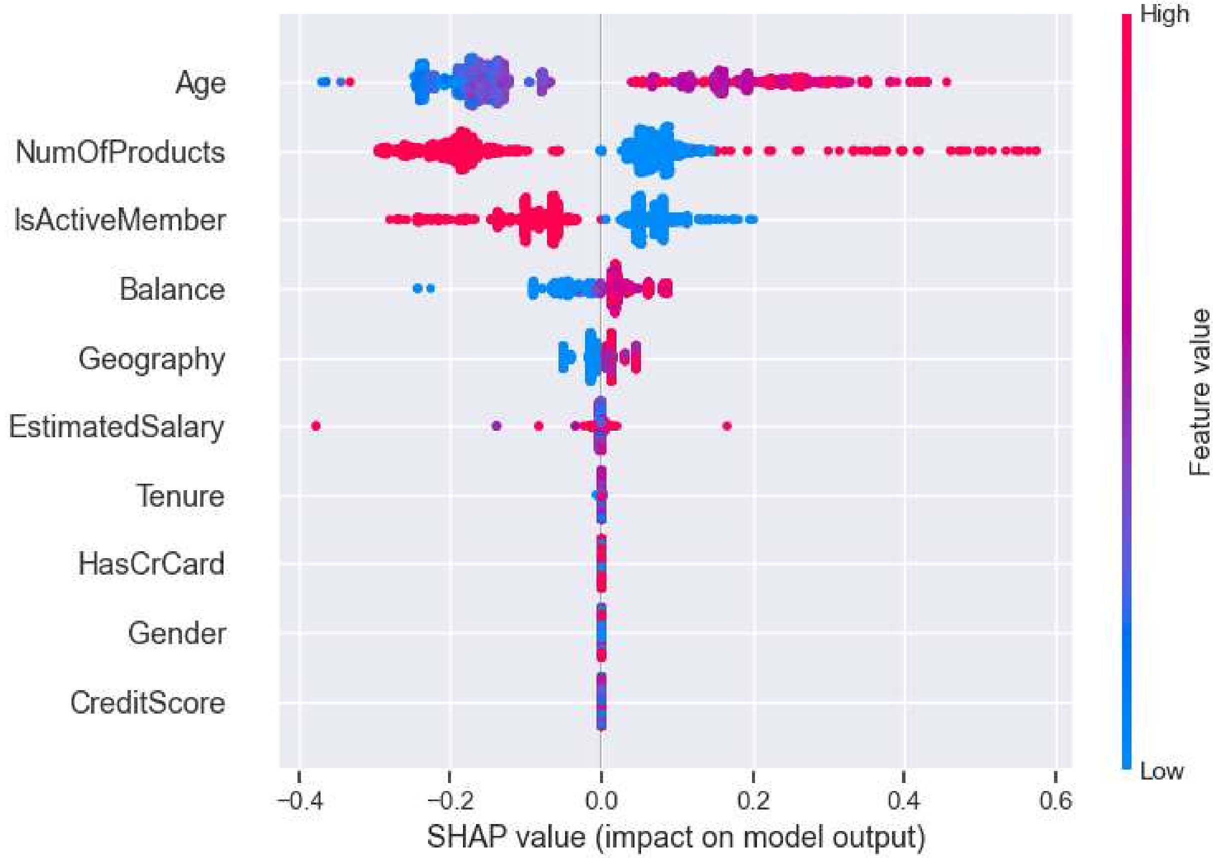
In [34]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [35]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [36]:
```
# compute SHAP values
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)
```

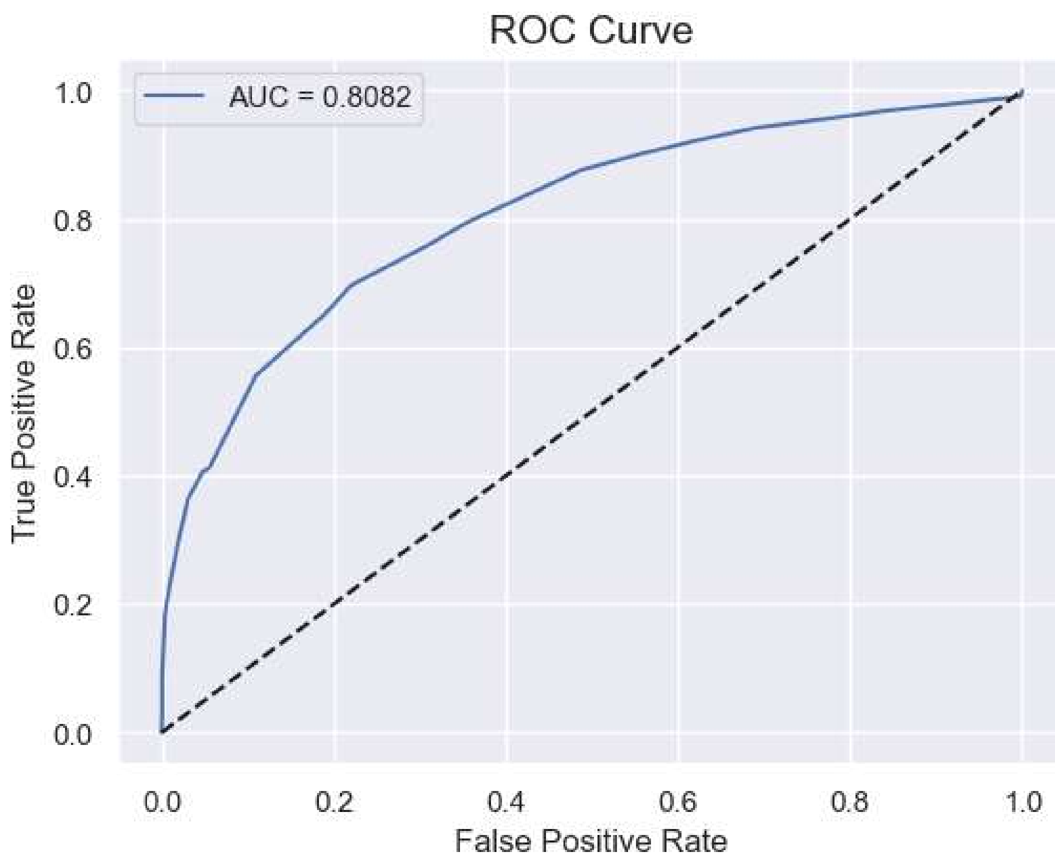


In [37]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test,
plt.title(all_sample_title, size = 15)
```

Out[37]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.776595744680851')
```

In [38]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_prob

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

Out[38]: <matplotlib.legend.Legend at 0x1e3701f9fa0>



Random Forest

In [40]:
```
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', 'log2', None]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': None, 'max_features': 'log2', 'n_estimators': 200}
```

In [43]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_features='log2', n_estimators=200, 
rfc.fit(X_train, y_train)
```

Out[43]:
```
RandomForestClassifier(class_weight='balanced', max_features='log2',
                       n_estimators=200, random_state=42)
```

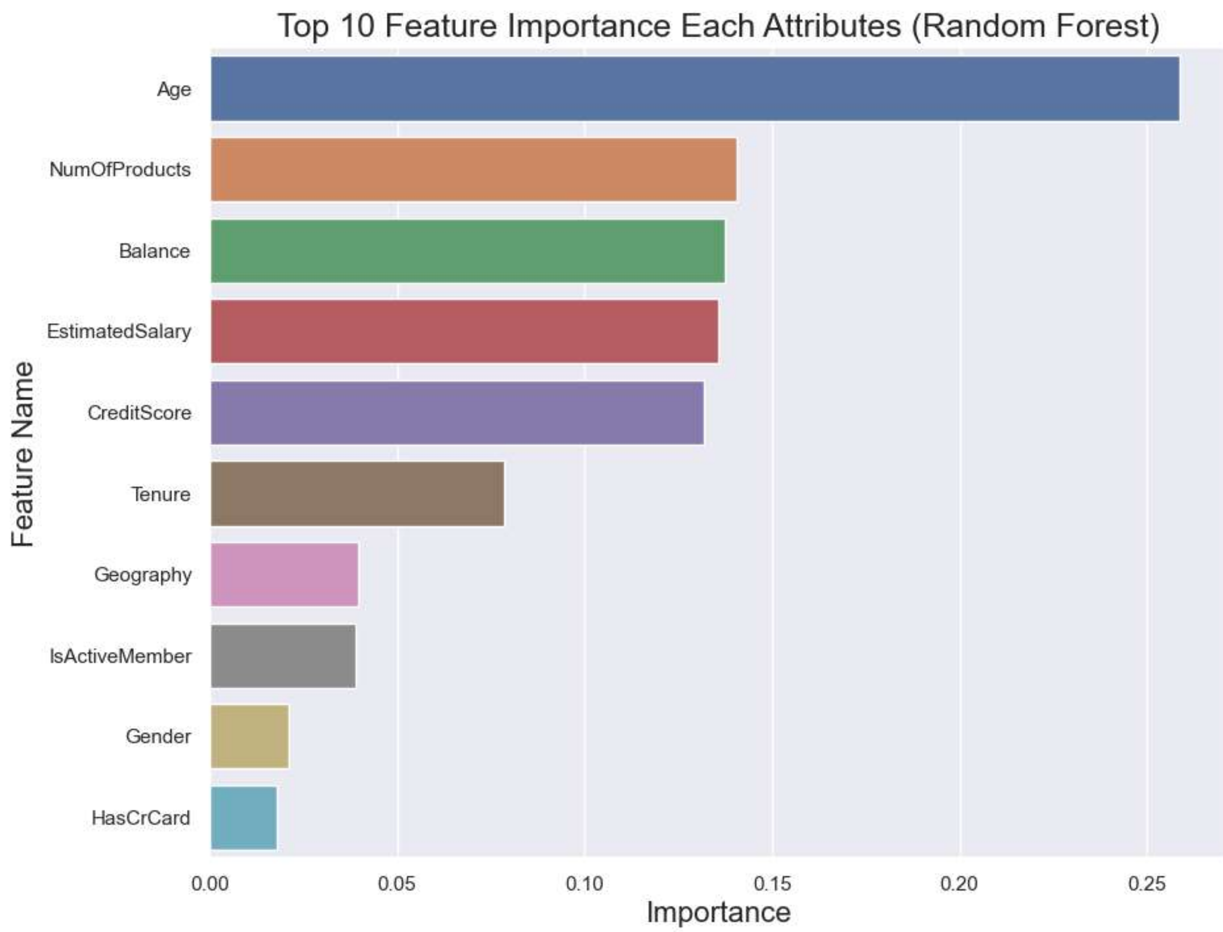
In [44]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 85.21 %
```

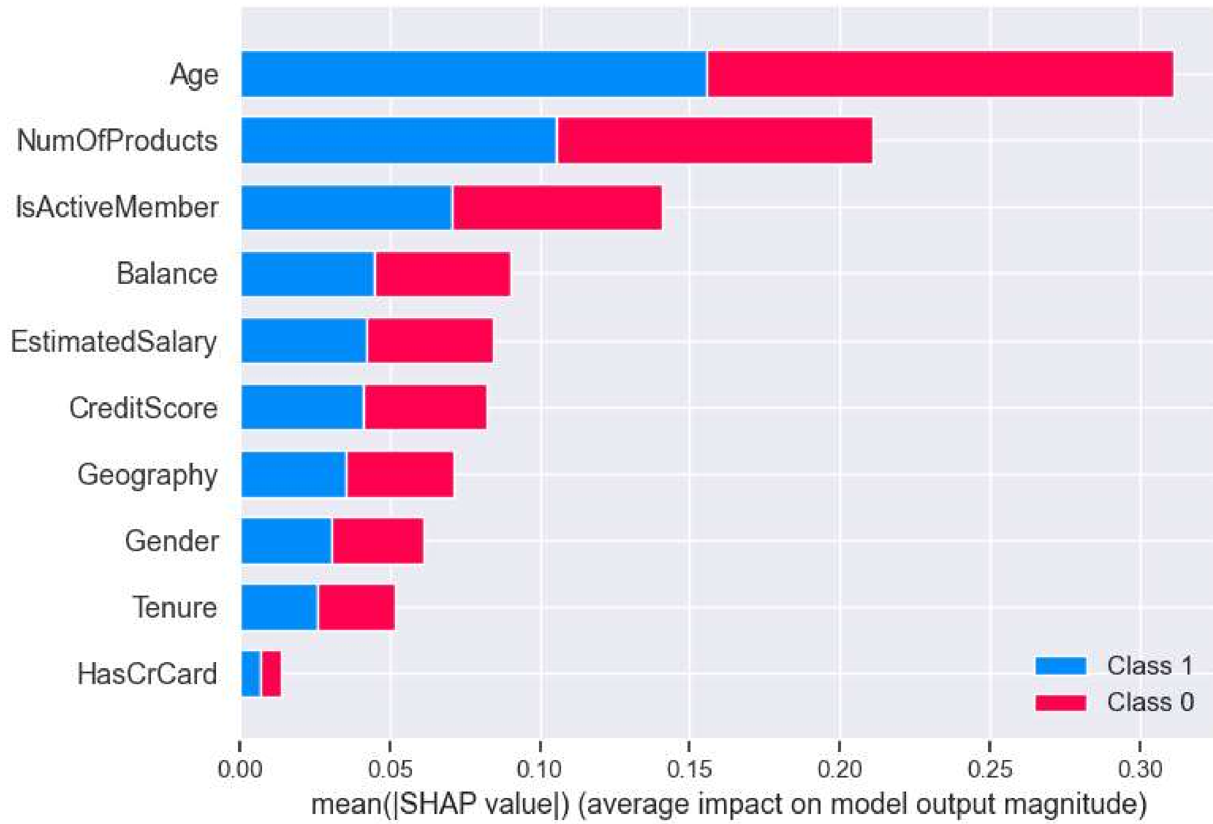
In [45]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, 
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.8520770010131712
Precision Score :  0.8520770010131712
Recall Score :  0.8520770010131712
Jaccard Score :  0.7422771403353927
Log Loss :  5.109098828769286
```

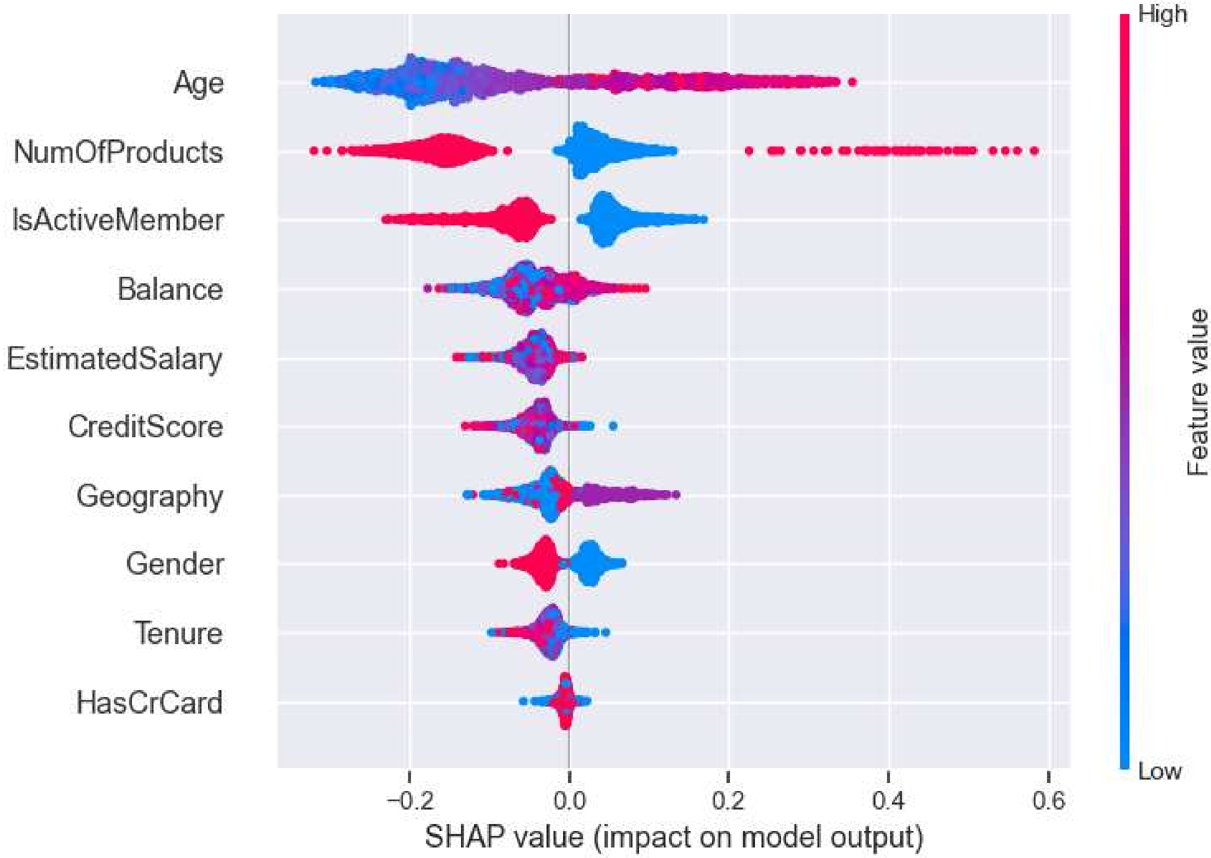
In [46]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



In [47]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

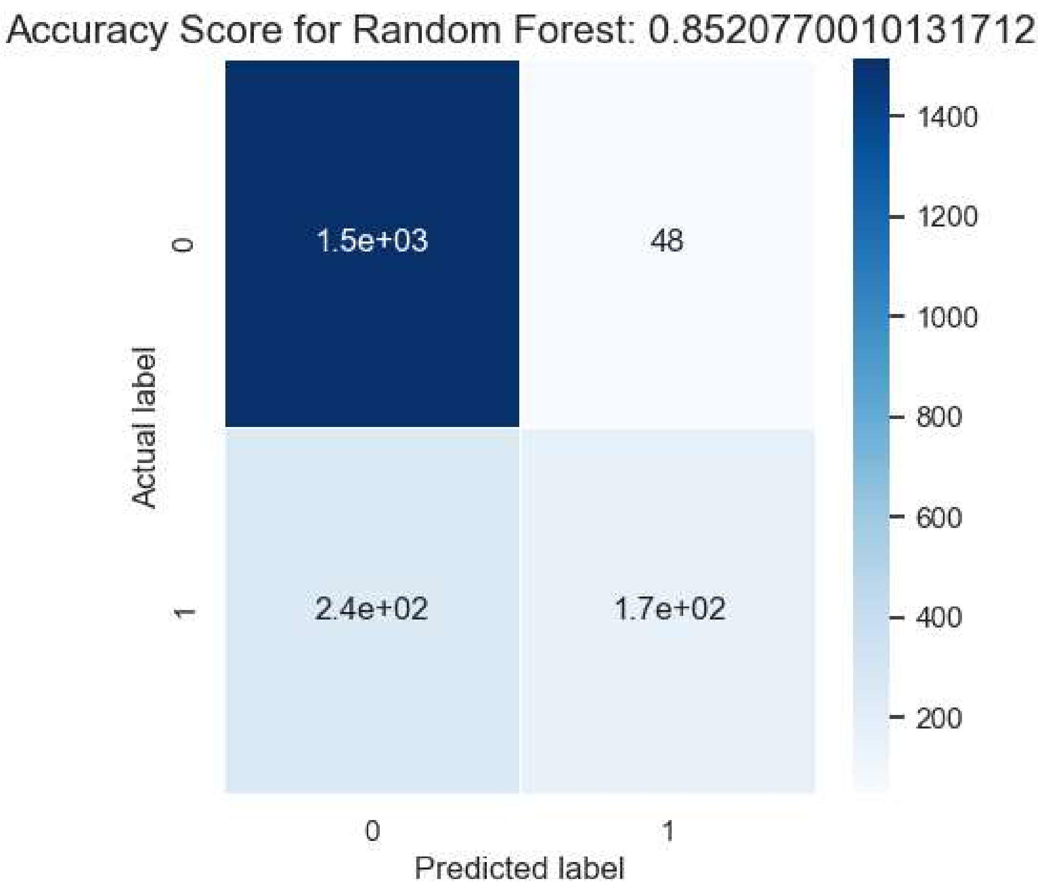


In [48]:
```
import shap
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



In [49]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_
plt.title(all_sample_title, size = 15)
```

Out[49]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.8520770010131712')
```



In [50]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual'])
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_prob

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

Out[50]: <matplotlib.legend.Legend at 0x1e36e0992e0>