

```
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_excel('E_comm.xlsx')
df.head()
```

Out[2]:

	CustomerID	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpentOnApp
0	50001	1	4.0	Mobile Phone	3	6.0	Debit Card	Female	
1	50002	1	NaN	Phone	1	8.0	UPI	Male	
2	50003	1	NaN	Phone	1	30.0	Debit Card	Male	
3	50004	1	0.0	Phone	3	15.0	Debit Card	Male	
4	50005	1	0.0	Phone	1	12.0	CC	Male	

## Data Preprocessing Part 1

```
In [3]: df.dtypes
```

```
Out[3]: CustomerID           int64
Churn                int64
Tenure              float64
PreferredLoginDevice    object
CityTier            int64
WarehouseToHome      float64
PreferredPaymentMode   object
Gender              object
HourSpendOnApp       float64
NumberOfDeviceRegistered int64
PreferedOrderCat     object
SatisfactionScore    int64
MaritalStatus        object
NumberOfAddress      int64
Complain             int64
OrderAmountHikeFromlastYear float64
CouponUsed            float64
OrderCount            float64
DaySinceLastOrder    float64
CashbackAmount        float64
dtype: object
```

```
In [4]: #Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

```
Out[4]: PreferredLoginDevice    3
PreferredPaymentMode    7
Gender                  2
PreferedOrderCat       6
MaritalStatus          3
dtype: int64
```

```
In [5]: # Remove CustomerID column  
df.drop(columns='CustomerID', inplace=True)  
df.head()
```

Out[5]:

	Churn	Tenure	PreferredLoginDevice	CityTier	WarehouseToHome	PreferredPaymentMode	Gender	HourSpendOnApp	N
0	1	4.0	Mobile Phone	3	6.0	Debit Card	Female		3.0
1	1	NaN	Phone	1	8.0	UPI	Male		3.0
2	1	NaN	Phone	1	30.0	Debit Card	Male		2.0
3	1	0.0	Phone	3	15.0	Debit Card	Male		2.0
4	1	0.0	Phone	1	12.0	CC	Male		NaN

## Exploratory Data Analysis

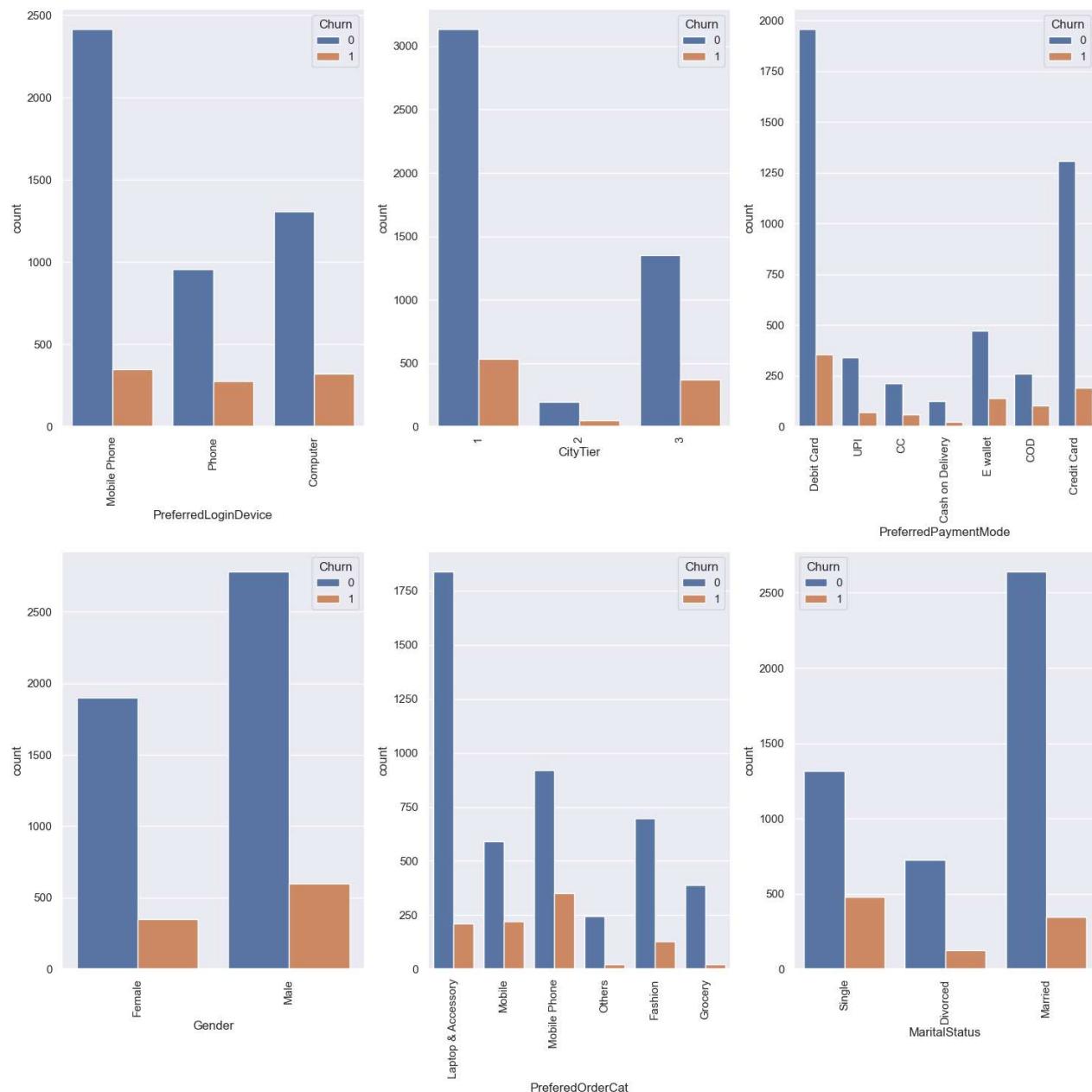
```
In [6]: # List of categorical variables to plot
cat_vars = ['PreferredLoginDevice', 'CityTier', 'PreferredPaymentMode', 'Gender',
            'PreferredOrderCat', 'MaritalStatus']

# 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='Churn', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



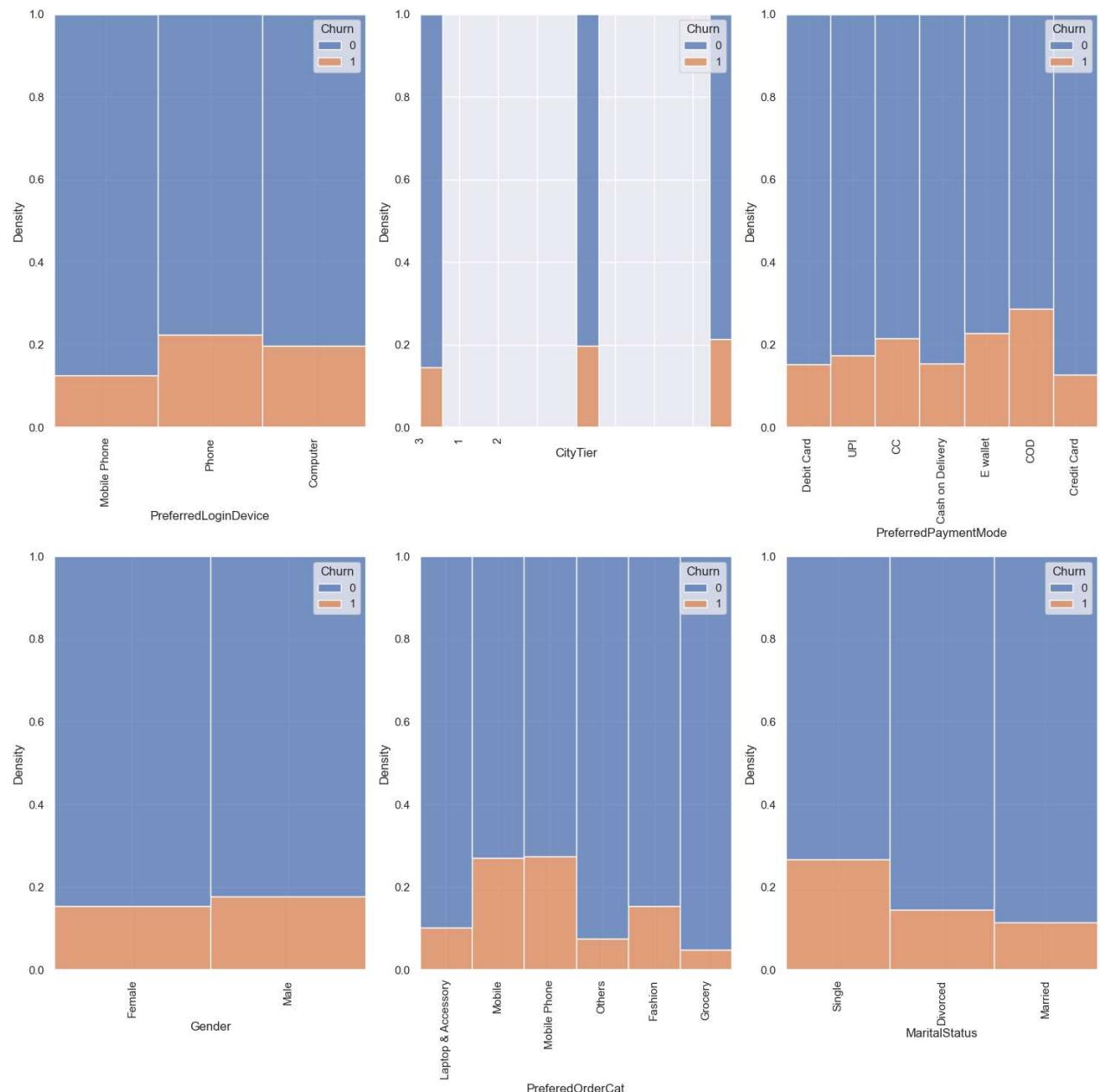
```
In [7]: import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['PreferredLoginDevice', 'CityTier', 'PreferredPaymentMode', 'Gender',
            'PreferredOrderCat', 'MaritalStatus']

# 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='Churn', data=df, ax=axs[i], multiple="fill", kde=False, element="bars")
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```



```
In [8]: cat_vars = ['CityTier', 'HourSpendOnApp', 'NumberOfDeviceRegistered', 'SatisfactionScore']

# create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 20))

# create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%.1f%%', startangle=90)

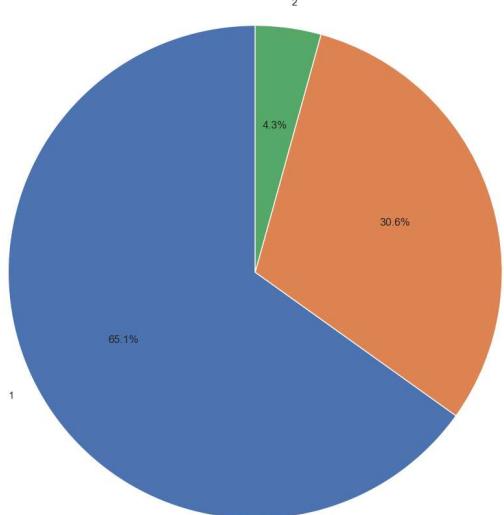
        # set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# adjust spacing between subplots
fig.tight_layout()

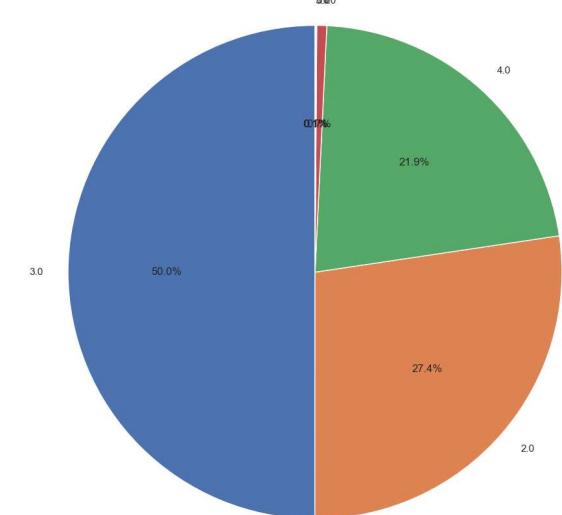
# show the plot
plt.show()
```

## Ecommerce Customer Churn Analysis and Prediction - Jupyter Notebook

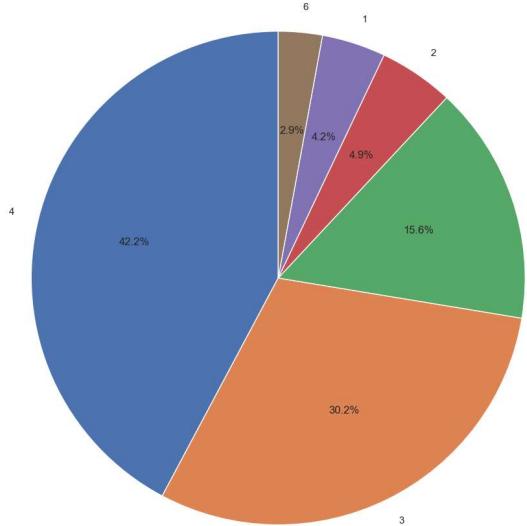
CityTier Distribution



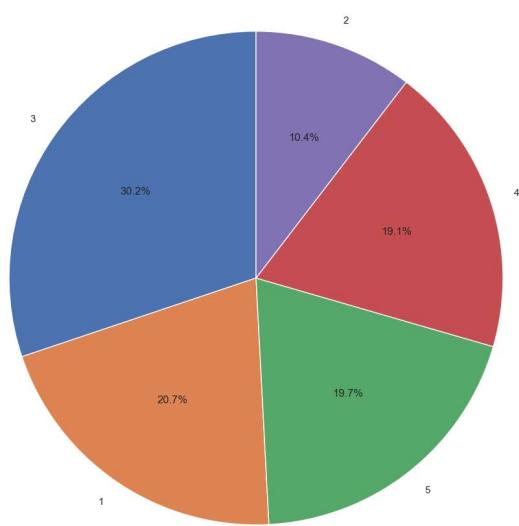
HourSpendOnApp Distribution



NumberOfDeviceRegistered Distribution



SatisfactionScore Distribution



```
In [9]: num_vars = ['Tenure', 'WarehouseToHome', 'NumberOfAddress', 'OrderAmountHikeFromlastYear',
   'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount']

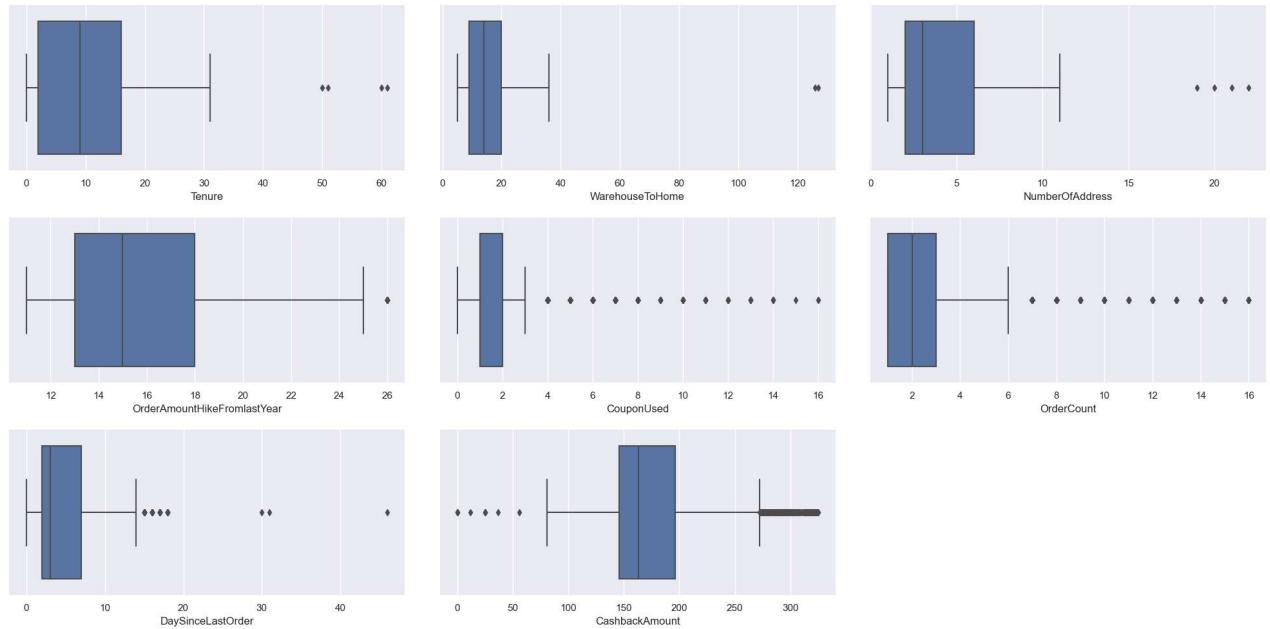
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 10))
axs = axs.flatten()

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

fig.tight_layout()

# remove the ninth subplot
fig.delaxes(axs[8])

plt.show()
```



```
In [10]: num_vars = ['Tenure', 'WarehouseToHome', 'NumberOfAddress', 'OrderAmountHikeFromlastYear',
                 'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount']

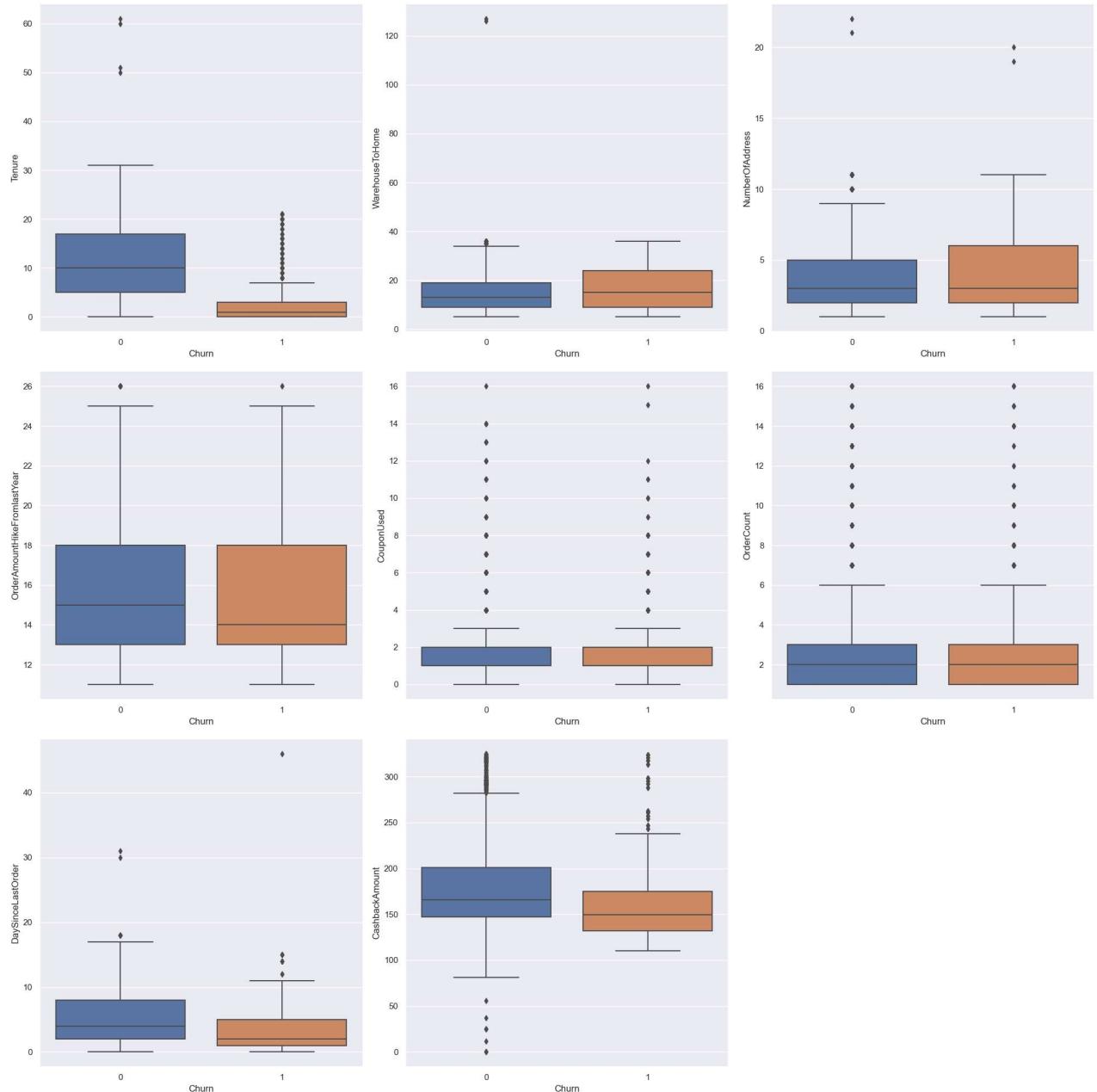
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 20))
axs = axs.flatten()

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

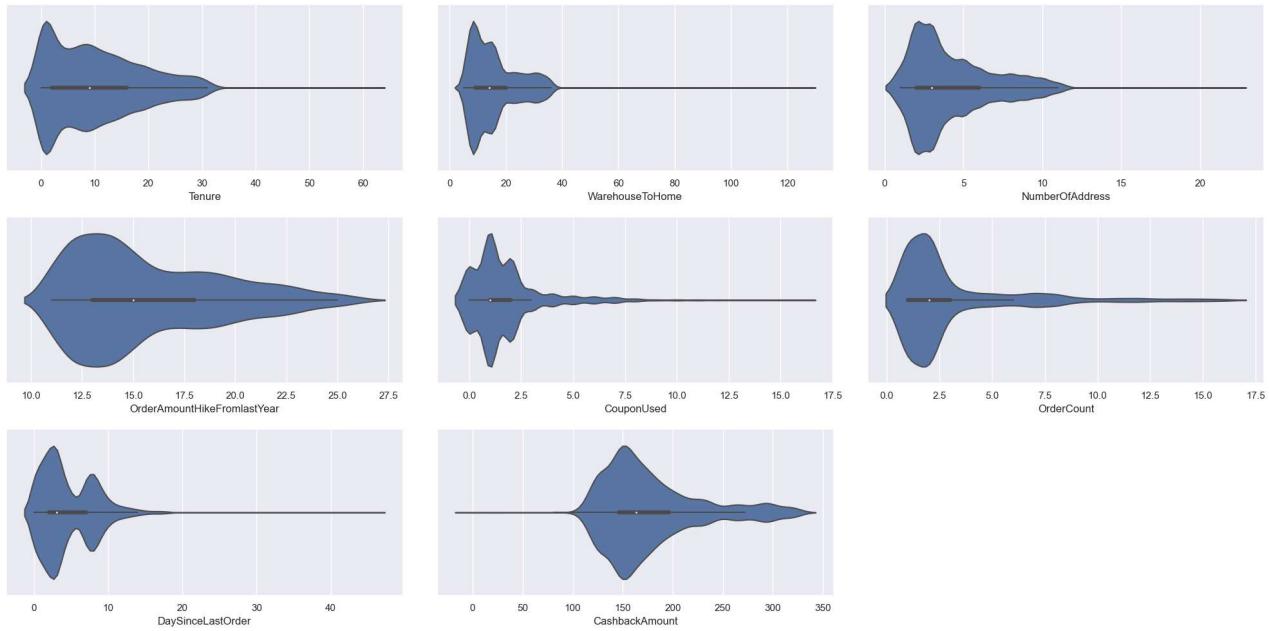
fig.tight_layout()

# remove the ninth subplot
fig.delaxes(axs[8])

plt.show()
```



```
In [11]: num_vars = ['Tenure', 'WarehouseToHome', 'NumberOfAddress', 'OrderAmountHikeFromlastYear',  
                 'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount']  
  
fig, axs = plt.subplots(nrows=3, ncols=3, 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()  
  
# remove the ninth subplot  
fig.delaxes(axs[8])  
  
plt.show()
```



```
In [12]: num_vars = ['Tenure', 'WarehouseToHome', 'NumberOfAddress', 'OrderAmountHikeFromlastYear',
                 'CouponUsed', 'OrderCount', 'DaySinceLastOrder', 'CashbackAmount']

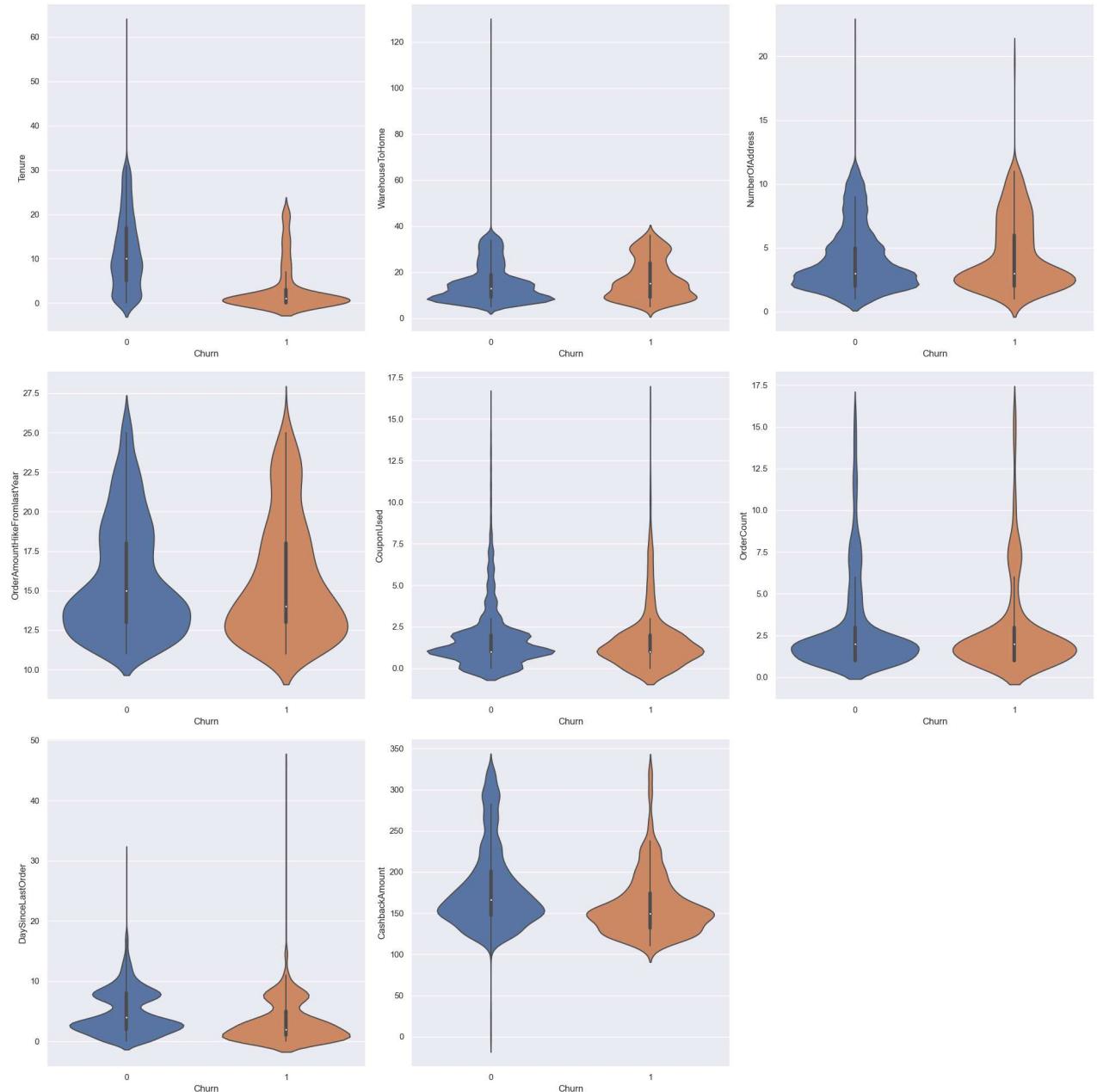
fig, axs = plt.subplots(nrows=3, ncols=3, figsize=(20, 20))
axs = axs.flatten()

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

fig.tight_layout()

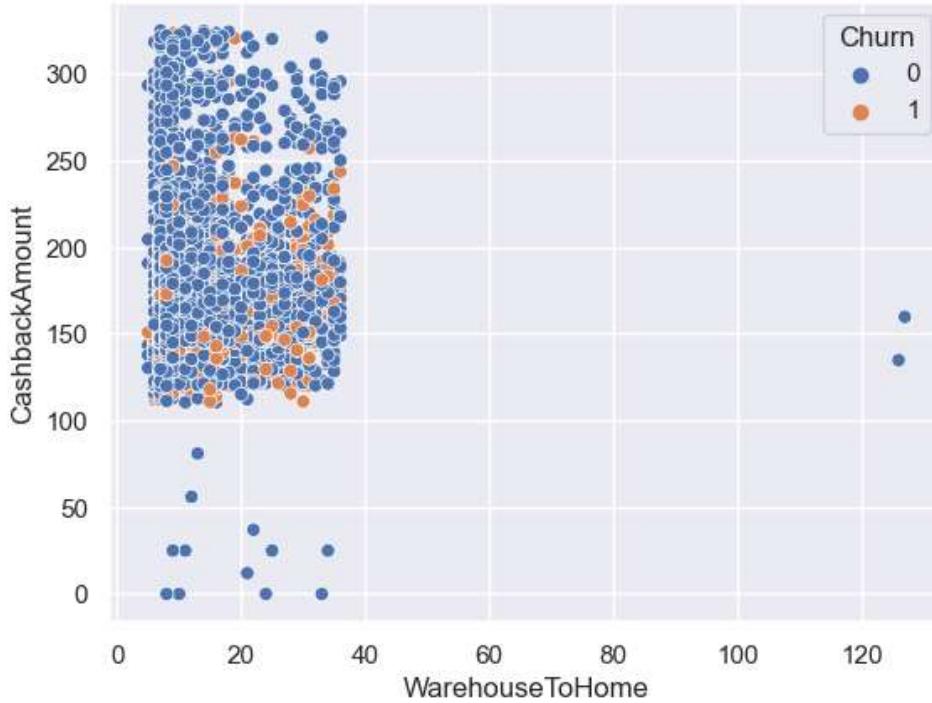
# remove the ninth subplot
fig.delaxes(axs[8])

plt.show()
```



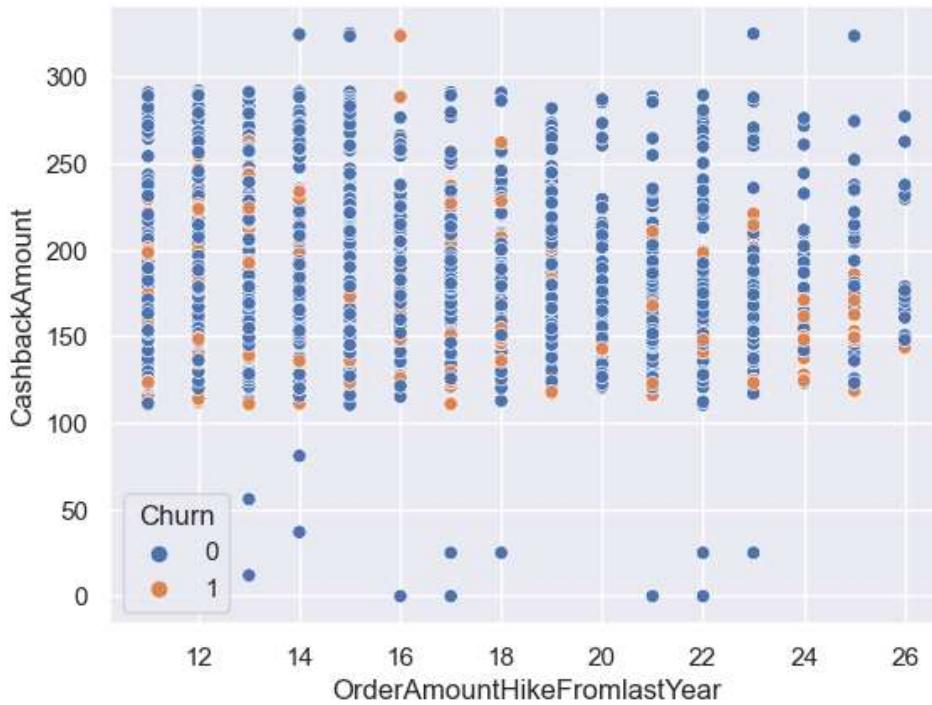
```
In [13]: sns.scatterplot(x='WarehouseToHome', y='CashbackAmount', hue='Churn', data=df)
```

```
Out[13]: <AxesSubplot:xlabel='WarehouseToHome', ylabel='CashbackAmount'>
```



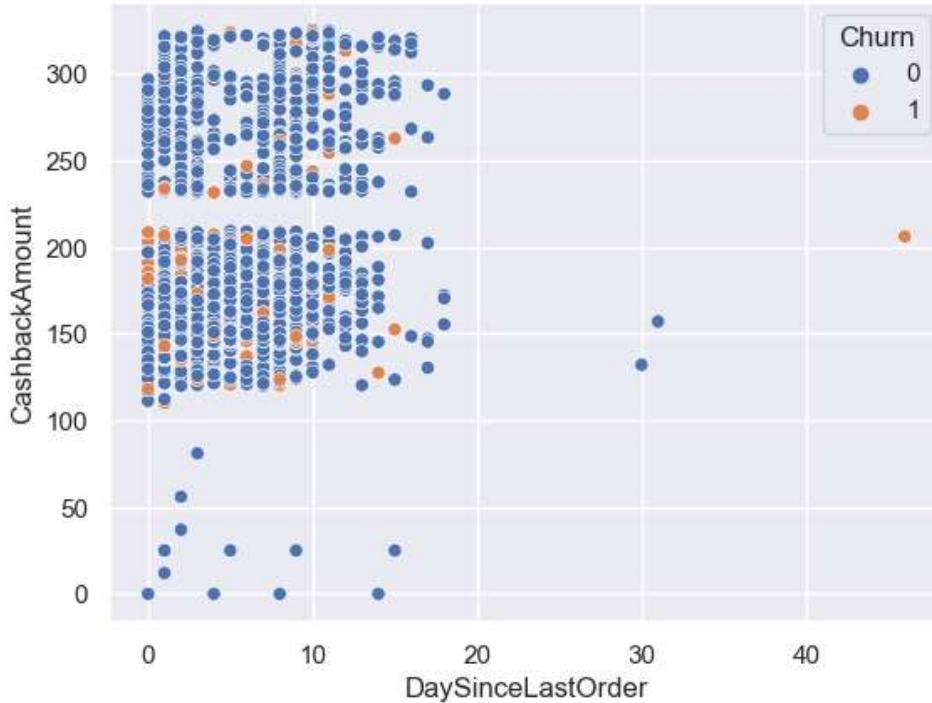
```
In [14]: sns.scatterplot(x='OrderAmountHikeFromlastYear', y='CashbackAmount', hue='Churn', data=df)
```

```
Out[14]: <AxesSubplot:xlabel='OrderAmountHikeFromlastYear', ylabel='CashbackAmount'>
```



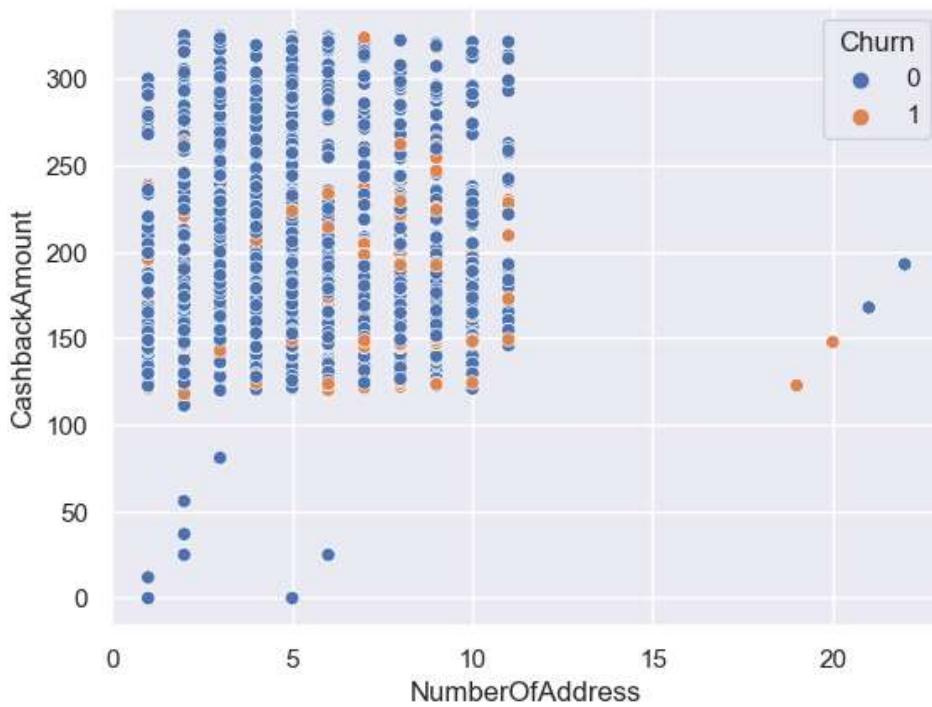
```
In [15]: sns.scatterplot(x='DaySinceLastOrder', y='CashbackAmount', hue='Churn', data=df)
```

```
Out[15]: <AxesSubplot:xlabel='DaySinceLastOrder', ylabel='CashbackAmount'>
```



```
In [16]: sns.scatterplot(x='NumberOfAddress', y='CashbackAmount', hue='Churn', data=df)
```

```
Out[16]: <AxesSubplot:xlabel='NumberOfAddress', ylabel='CashbackAmount'>
```



## Data Preprocessing Part 2

```
In [17]: #Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Out[17]: DaySinceLastOrder      5.452931
OrderAmountHikeFromlastYear   4.706927
Tenure                         4.689165
OrderCount                      4.582593
CouponUsed                      4.547069
HourSpendOnApp                  4.529307
WarehouseToHome                 4.458259
dtype: float64
```

```
In [18]: # fill null value with median
df['HourSpendOnApp'].fillna(df['HourSpendOnApp'].median(), inplace=True)
df['OrderAmountHikeFromlastYear'].fillna(df['OrderAmountHikeFromlastYear'].median(), inplace=True)
df['Tenure'].fillna(df['Tenure'].median(), inplace=True)
df['WarehouseToHome'].fillna(df['WarehouseToHome'].median(), inplace=True)
df['DaySinceLastOrder'].fillna(df['DaySinceLastOrder'].median(), inplace=True)
df['OrderCount'].fillna(df['OrderCount'].median(), inplace=True)
df['CouponUsed'].fillna(df['CouponUsed'].median(), inplace=True)
```

```
In [19]: #Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Out[19]: Series([], dtype: float64)
```

## Label Encoding for Object datatype

```
In [20]: # 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()}")
```

```
PreferredLoginDevice: ['Mobile Phone' 'Phone' 'Computer']
PreferredPaymentMode: ['Debit Card' 'UPI' 'CC' 'Cash on Delivery' 'E wallet' 'COD' 'Credit Card']
Gender: ['Female' 'Male']
PreferredOrderCat: ['Laptop & Accessory' 'Mobile' 'Mobile Phone' 'Others' 'Fashion' 'Grocery']
MaritalStatus: ['Single' 'Divorced' 'Married']
```

```
In [21]: 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])

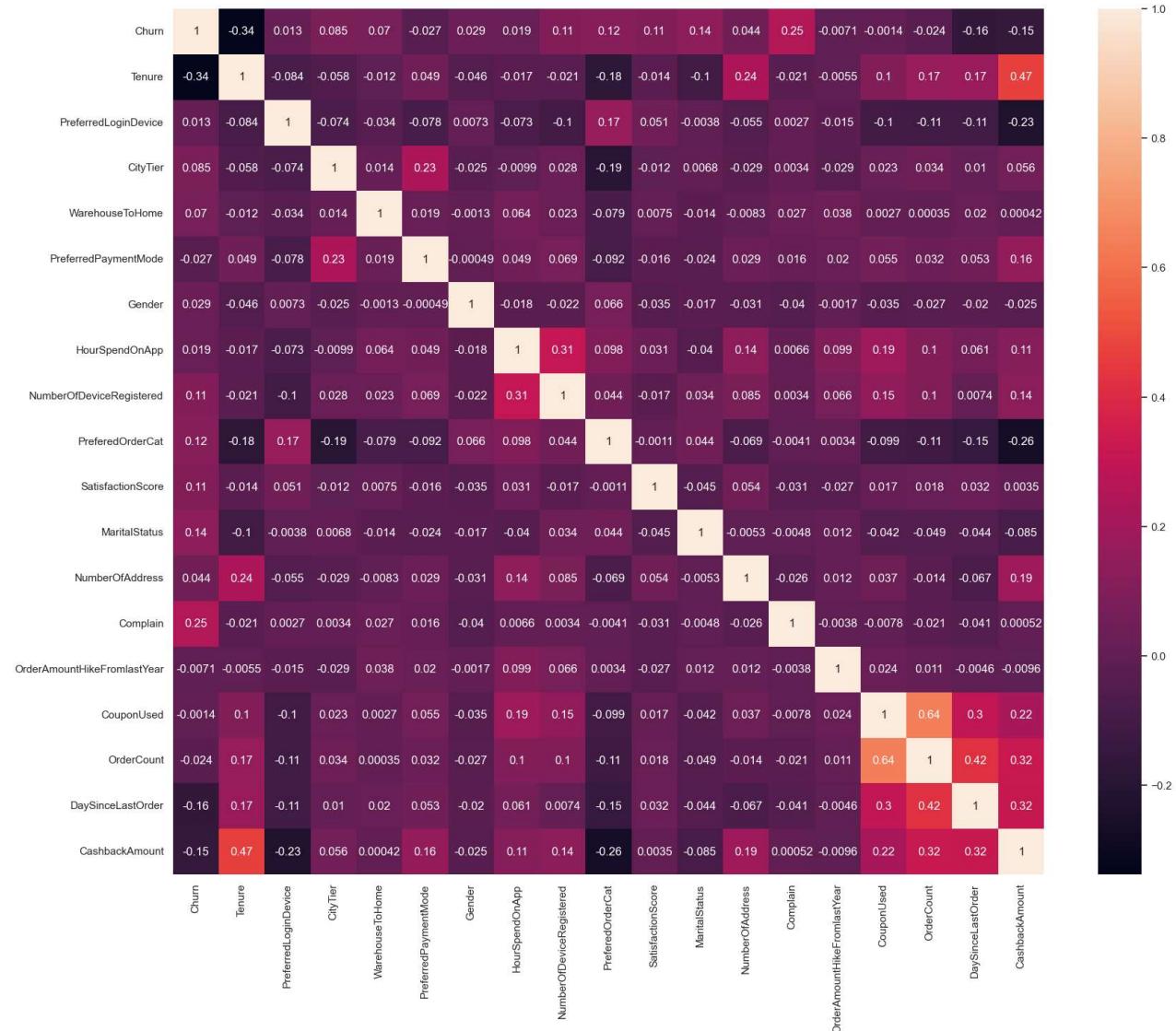
    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
PreferredLoginDevice: [1 2 0]
PreferredPaymentMode: [4 6 0 2 5 1 3]
Gender: [0 1]
PreferredOrderCat: [2 3 4 5 0 1]
MaritalStatus: [2 0 1]
```

## Correlation Heatmap

```
In [22]: #Correlation Heatmap
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[22]: <AxesSubplot:>



## Train Test Split

```
In [24]: from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Churn', axis=1)
y = df['Churn']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

## Remove Outlier in Train Data using Z-Score

```
In [25]: from sklearn.model_selection import train_test_split

# Select the column for outlier removal
selected_column = ['WarehouseToHome', 'DaySinceLastOrder', 'CouponUsed', 'OrderCount']

# Split the data into training and test sets
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

# Calculate the z-scores for the selected column in the training set
train_z_scores = np.abs((train_df[selected_column] - train_df[selected_column].mean()) / train_df[selected_column].std())

# Set the threshold for defining outliers (e.g., z-score > 3)
threshold = 3

# Filter the training DataFrame, removing rows with z-scores above the threshold in the selected column
train_df_no_outliers = train_df[train_z_scores <= threshold]
```

## Decision Tree

```
In [26]: 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': 8, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 42}
```

```
In [27]: from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=42, max_depth=8, min_samples_leaf=1, min_samples_split=2)
dtree.fit(X_train, y_train)
```

```
Out[27]: DecisionTreeClassifier(class_weight='balanced', max_depth=8, random_state=42)
```

```
In [28]: from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score : ", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

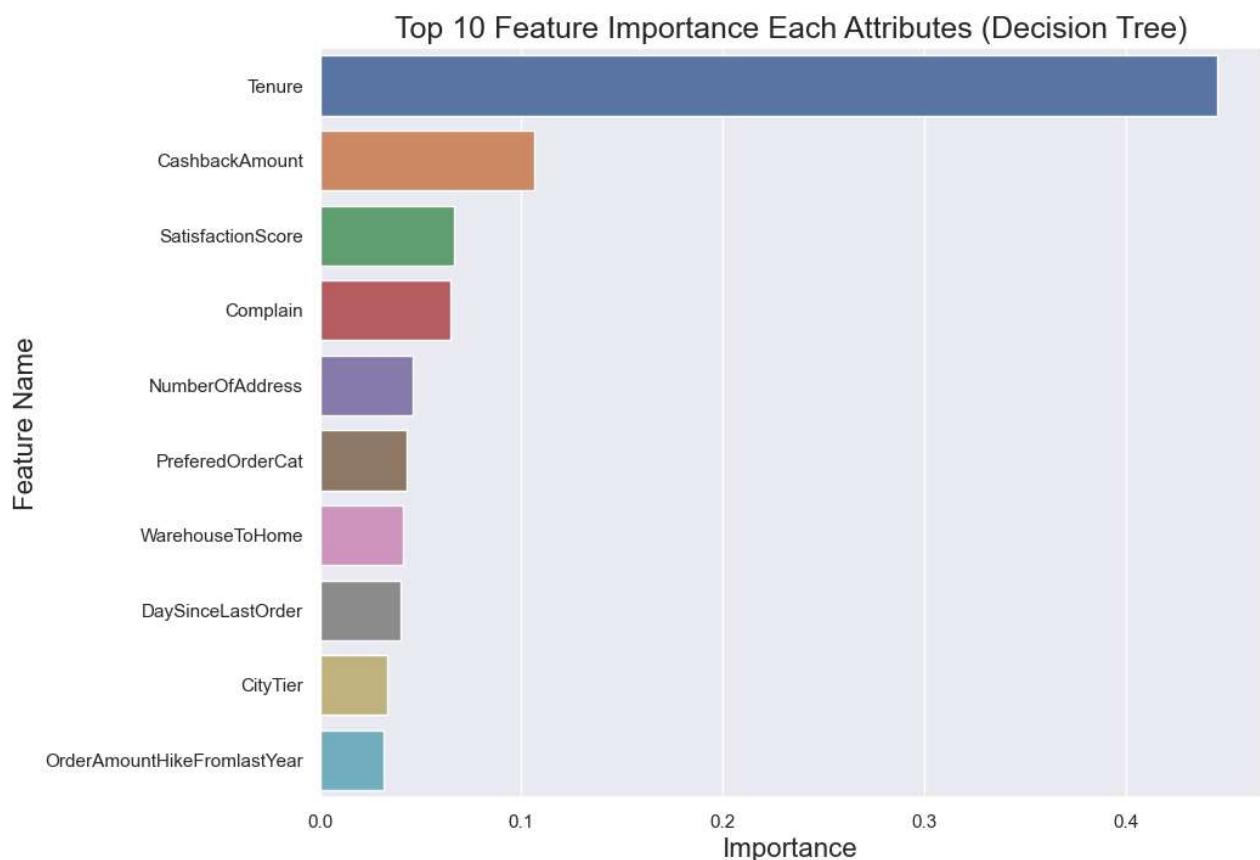
Accuracy Score : 88.01 %

```
In [29]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_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)))
```

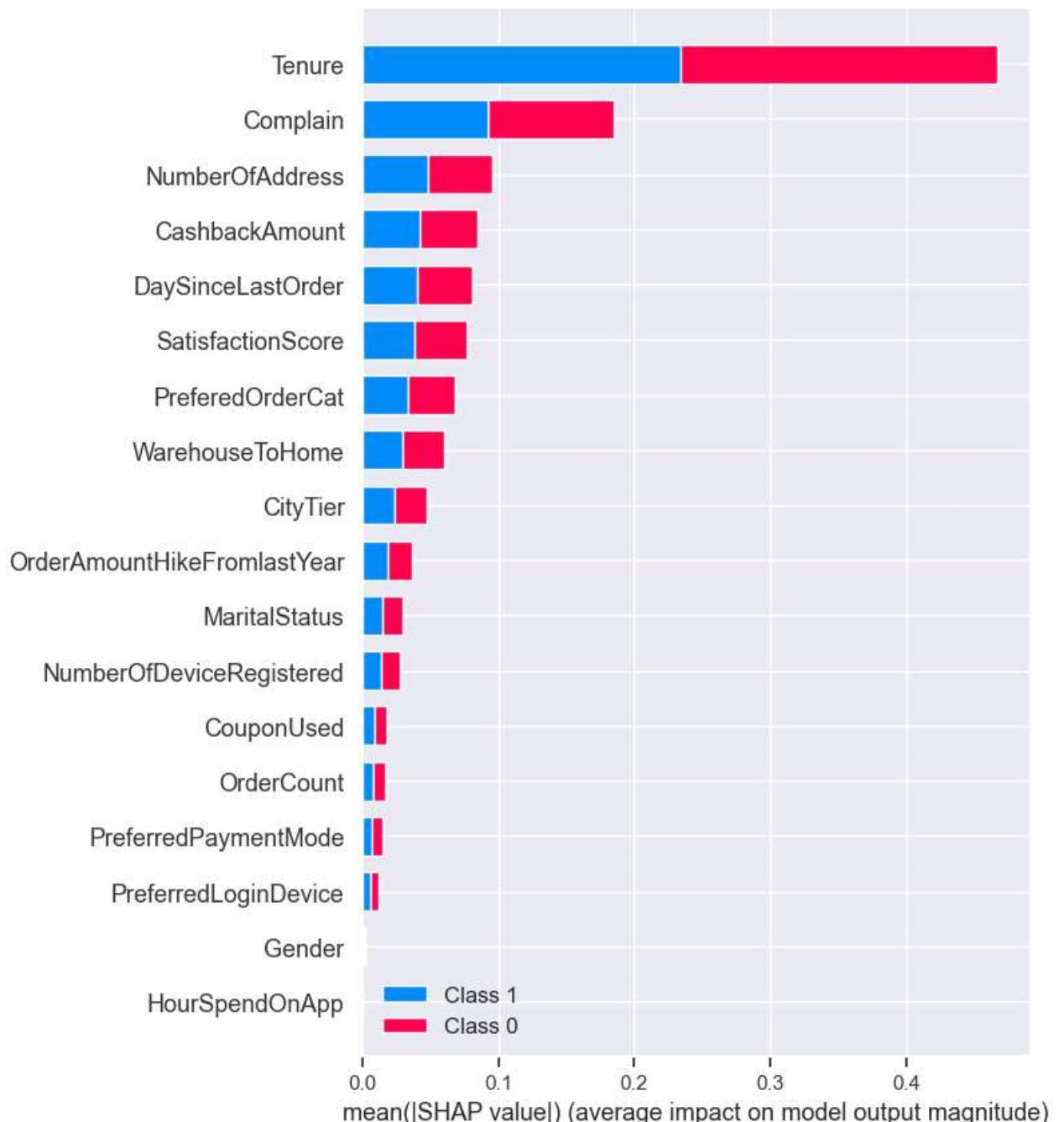
F-1 Score : 0.8801065719360568  
Precision Score : 0.8801065719360568  
Recall Score : 0.8801065719360568  
Jaccard Score : 0.7858842188739096  
Log Loss : 4.141051836789631

```
In [30]: imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

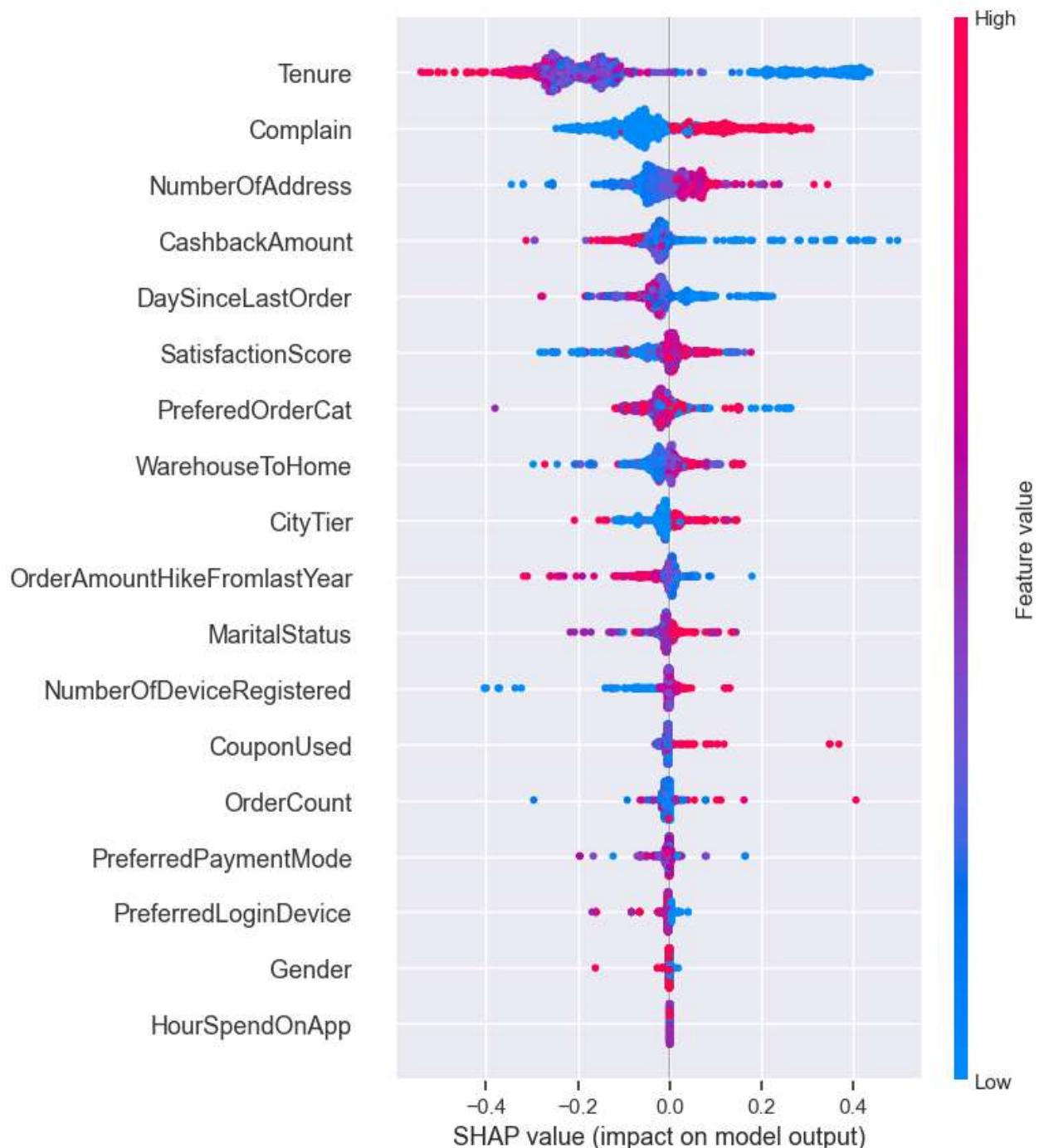
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



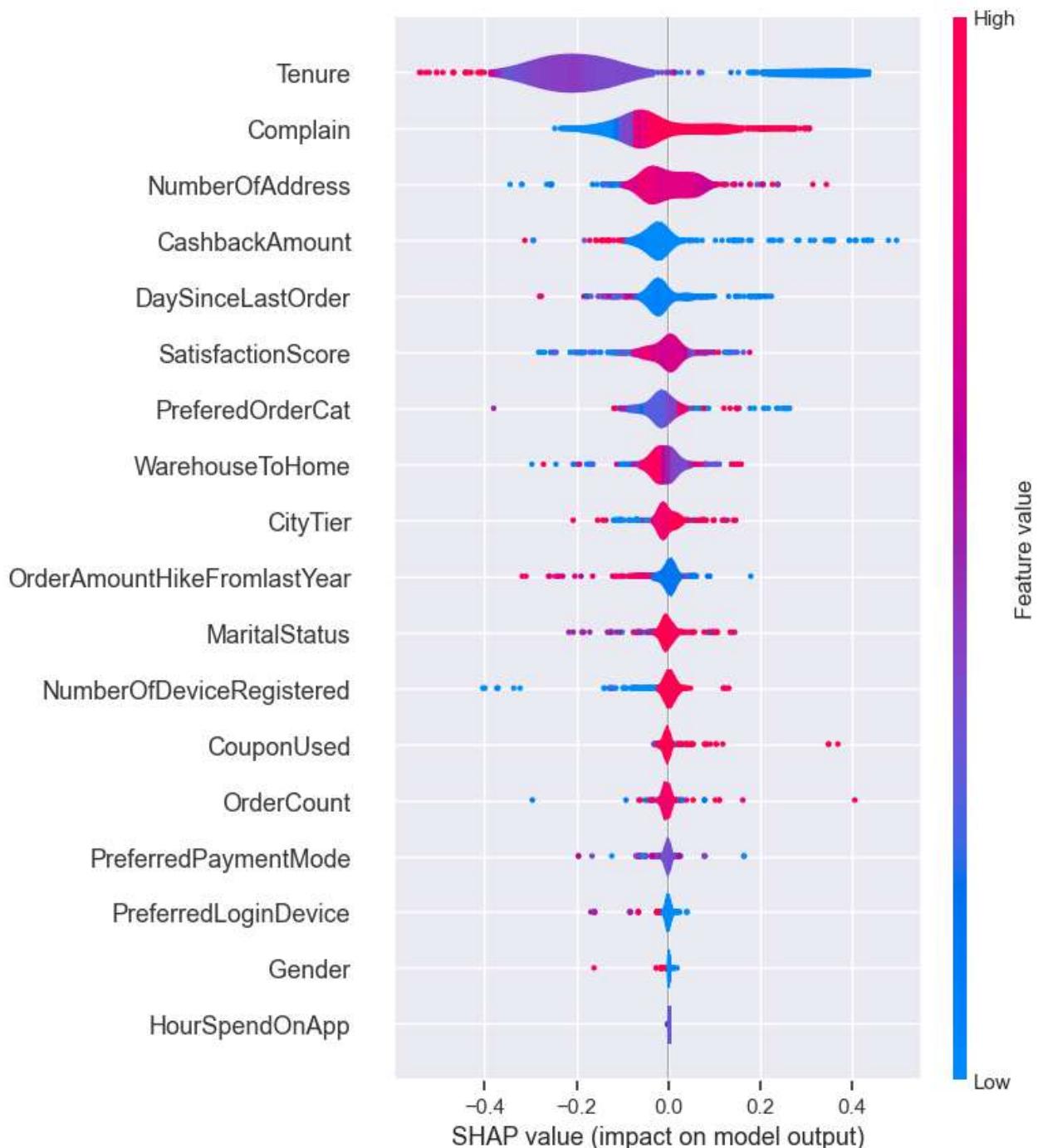
```
In [31]: import shap  
explainer = shap.TreeExplainer(dtree)  
shap_values = explainer.shap_values(X_test)  
shap.summary_plot(shap_values, X_test)
```



```
In [32]: # 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)
```

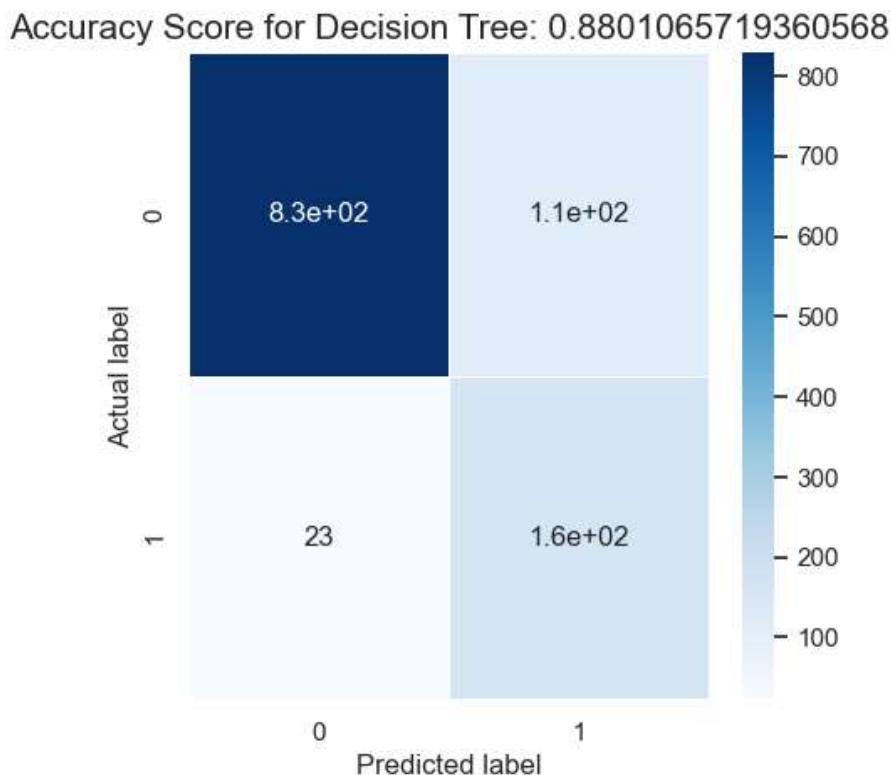


```
In [33]: # compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



```
In [34]: 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, y_test))
plt.title(all_sample_title, size = 15)
```

```
Out[34]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.8801065719360568')
```



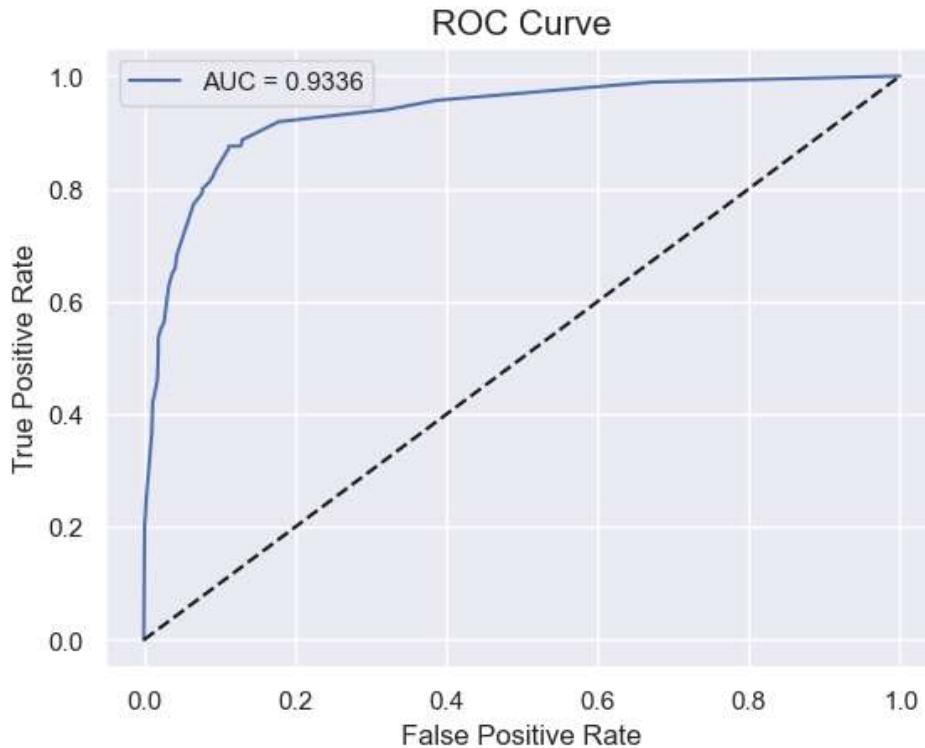
```
In [35]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:, :, 1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(df_actual_predicted)], axis=1)
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

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[35]: <matplotlib.legend.Legend at 0x247eea3db20>
```



## Random Forest

```
In [36]: 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],
    'random_state': [0, 42]
}

# 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': 'sqrt', 'n\_estimators': 100, 'random\_state': 0}

```
In [37]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_features='sqrt', n_estimators=100, class_weight='balanced')
rfc.fit(X_train, y_train)
```

```
Out[37]: RandomForestClassifier(class_weight='balanced', max_features='sqrt',
                                random_state=0)
```

```
In [38]: y_pred = rfc.predict(X_test)
print("Accuracy Score : ", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

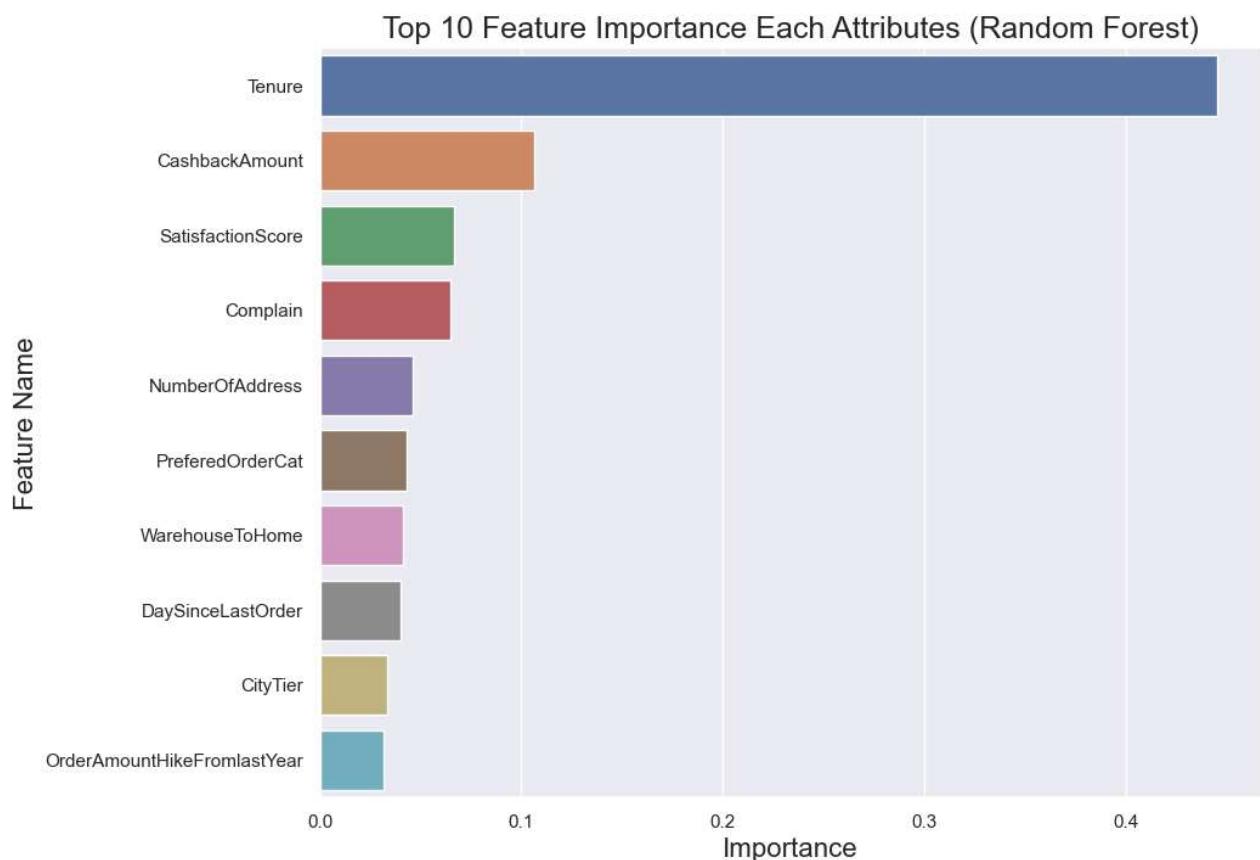
Accuracy Score : 97.6 %

```
In [39]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_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)))
```

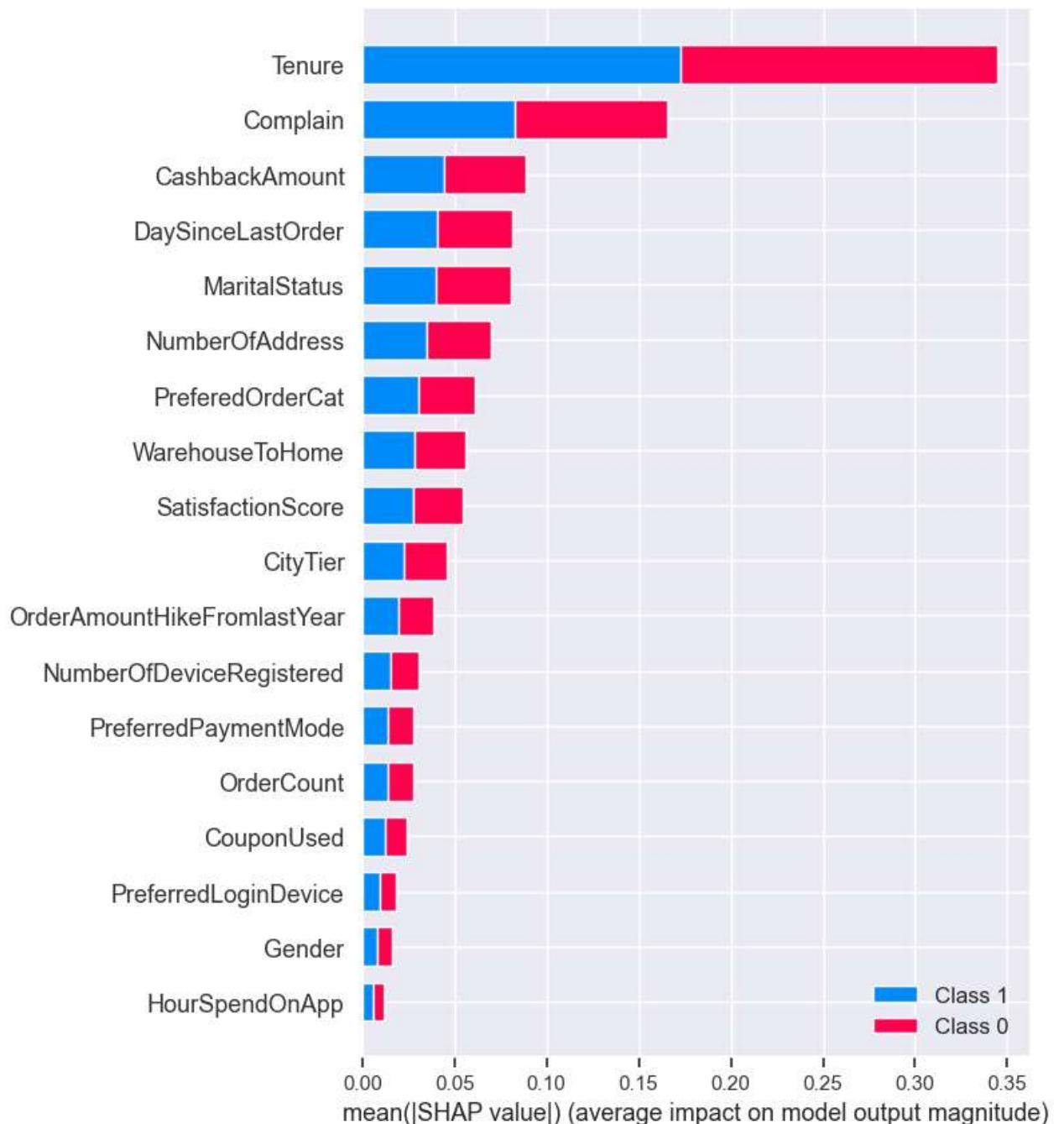
F-1 Score : 0.9760213143872114  
 Precision Score : 0.9760213143872114  
 Recall Score : 0.9760213143872114  
 Jaccard Score : 0.9531656548135299  
 Log Loss : 0.8281958808680727

```
In [40]: imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

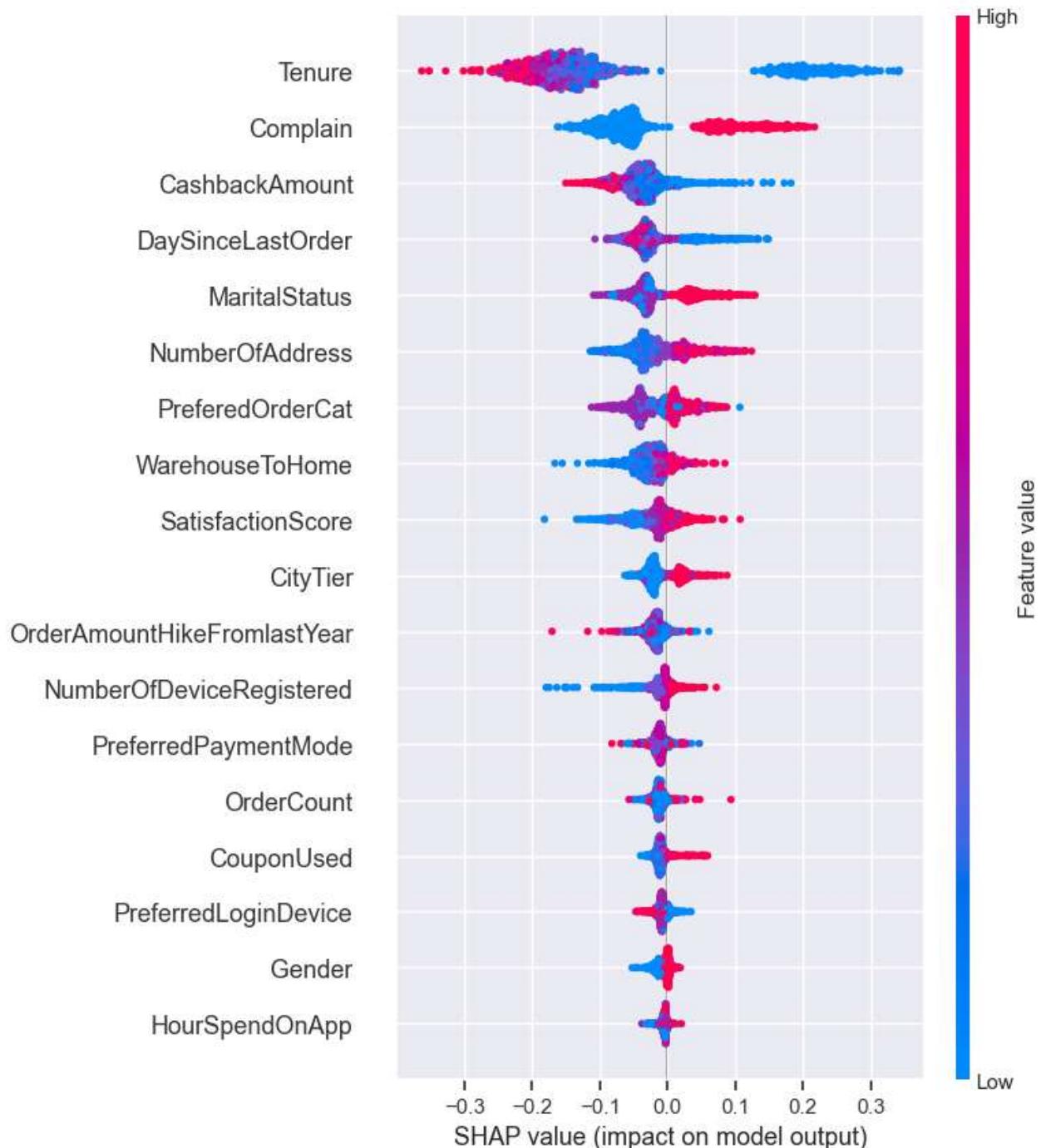
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



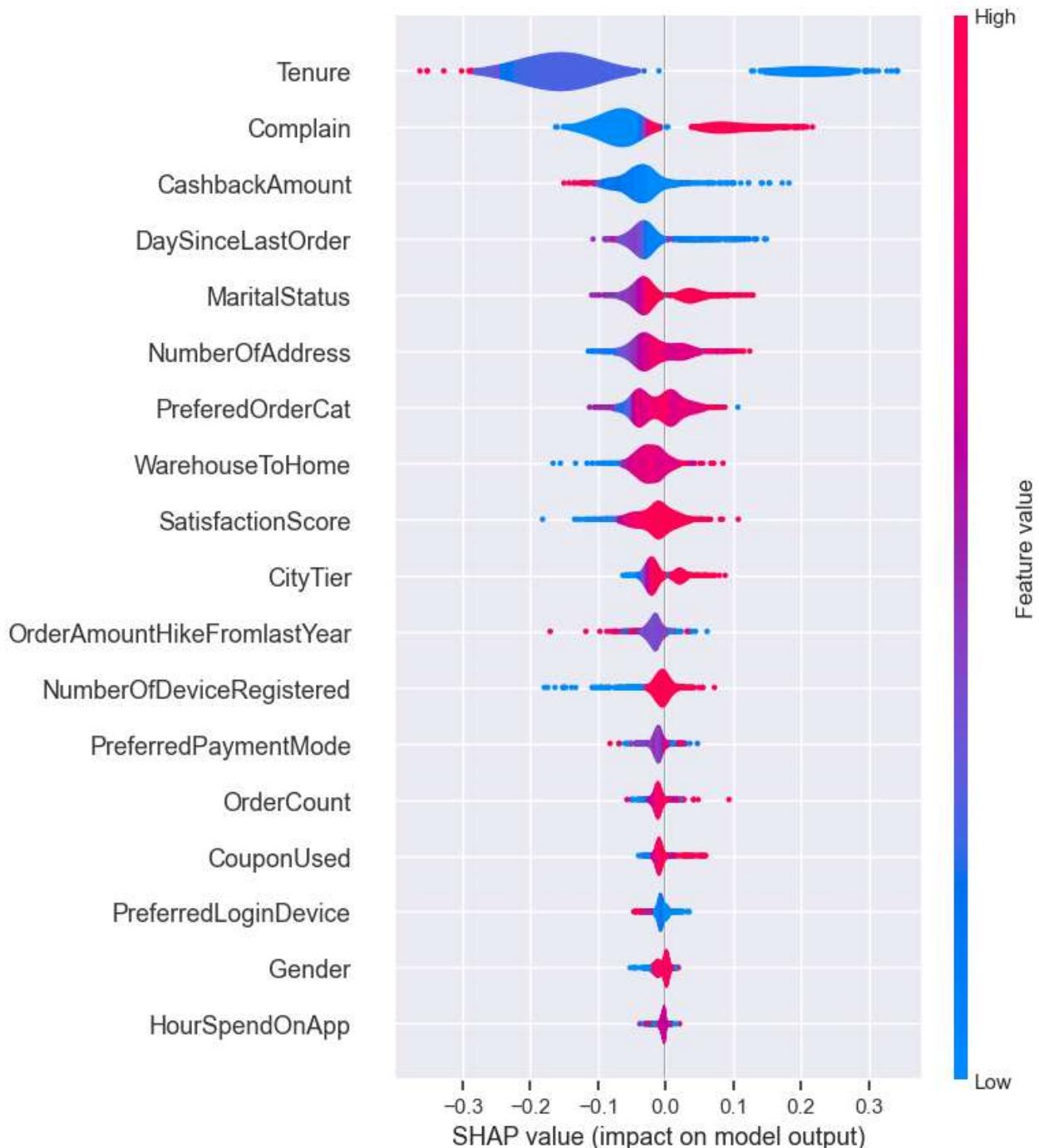
```
In [41]: import shap  
explainer = shap.TreeExplainer(rfc)  
shap_values = explainer.shap_values(X_test)  
shap.summary_plot(shap_values, X_test)
```



```
In [42]: # 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)
```

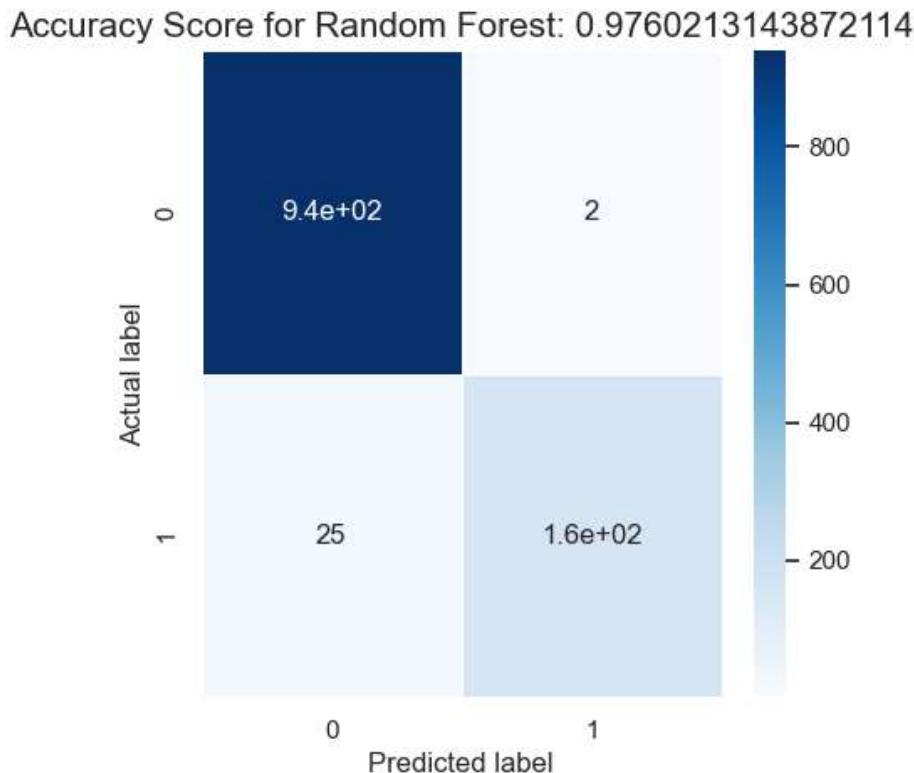


```
In [43]: # compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin")
```



```
In [44]: 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_test))
plt.title(all_sample_title, size = 15)
```

```
Out[44]: Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9760213143872114')
```



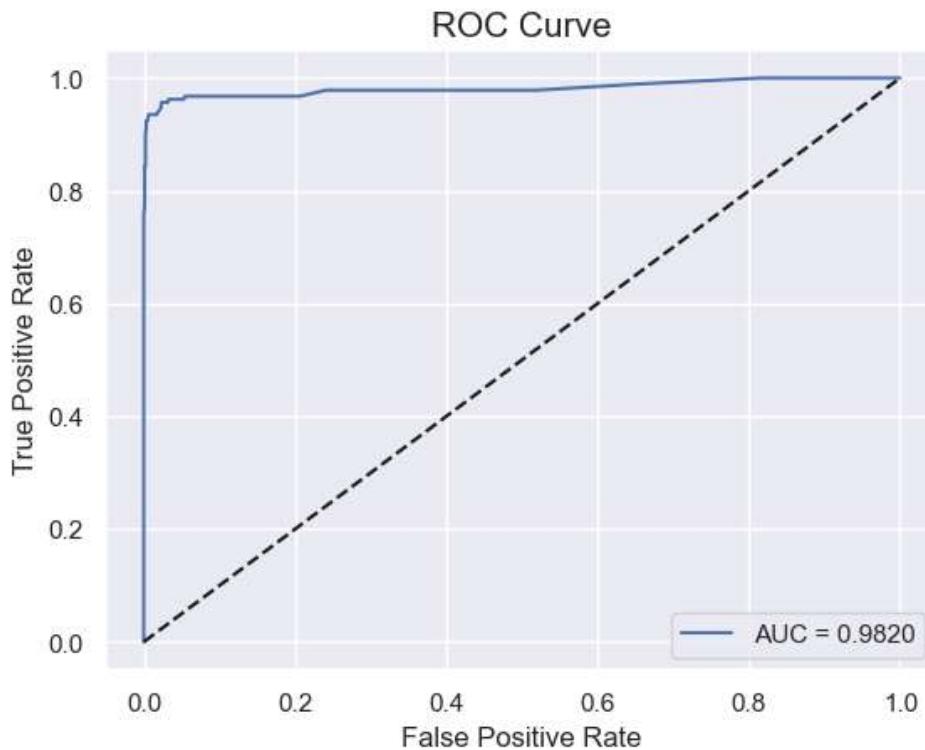
```
In [45]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:,1]

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

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

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[45]: <matplotlib.legend.Legend at 0x247ef5c1bb0>
```



## XGBoost

```
In [46]: from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier

# Create an XGBoost classifier
xgb = XGBClassifier()

# Define the parameter grid for grid search
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.1, 0.01, 0.001],
    'gamma': [0, 0.1, 0.2]
}

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

# Print the best hyperparameters
print(grid_search.best_params_)

{'gamma': 0, 'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 200}
```

```
In [47]: from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0, learning_rate=0.1, max_depth=7, n_estimators=200)
xgb.fit(X_train, y_train)
```

```
Out[47]: XGBClassifier(base_score=None, booster=None, callbacks=None,
                      colsample_bylevel=None, colsample_bynode=None,
                      colsample_bytree=None, early_stopping_rounds=None,
                      enable_categorical=False, eval_metric=None, feature_types=None,
                      gamma=0, gpu_id=None, grow_policy=None, importance_type=None,
                      interaction_constraints=None, learning_rate=0.1, max_bin=None,
                      max_cat_threshold=None, max_cat_to_onehot=None,
                      max_delta_step=None, max_depth=7, max_leaves=None,
                      min_child_weight=None, missing=nan, monotone_constraints=None,
                      n_estimators=200, n_jobs=None, num_parallel_tree=None,
                      predictor=None, random_state=None, ...)
```

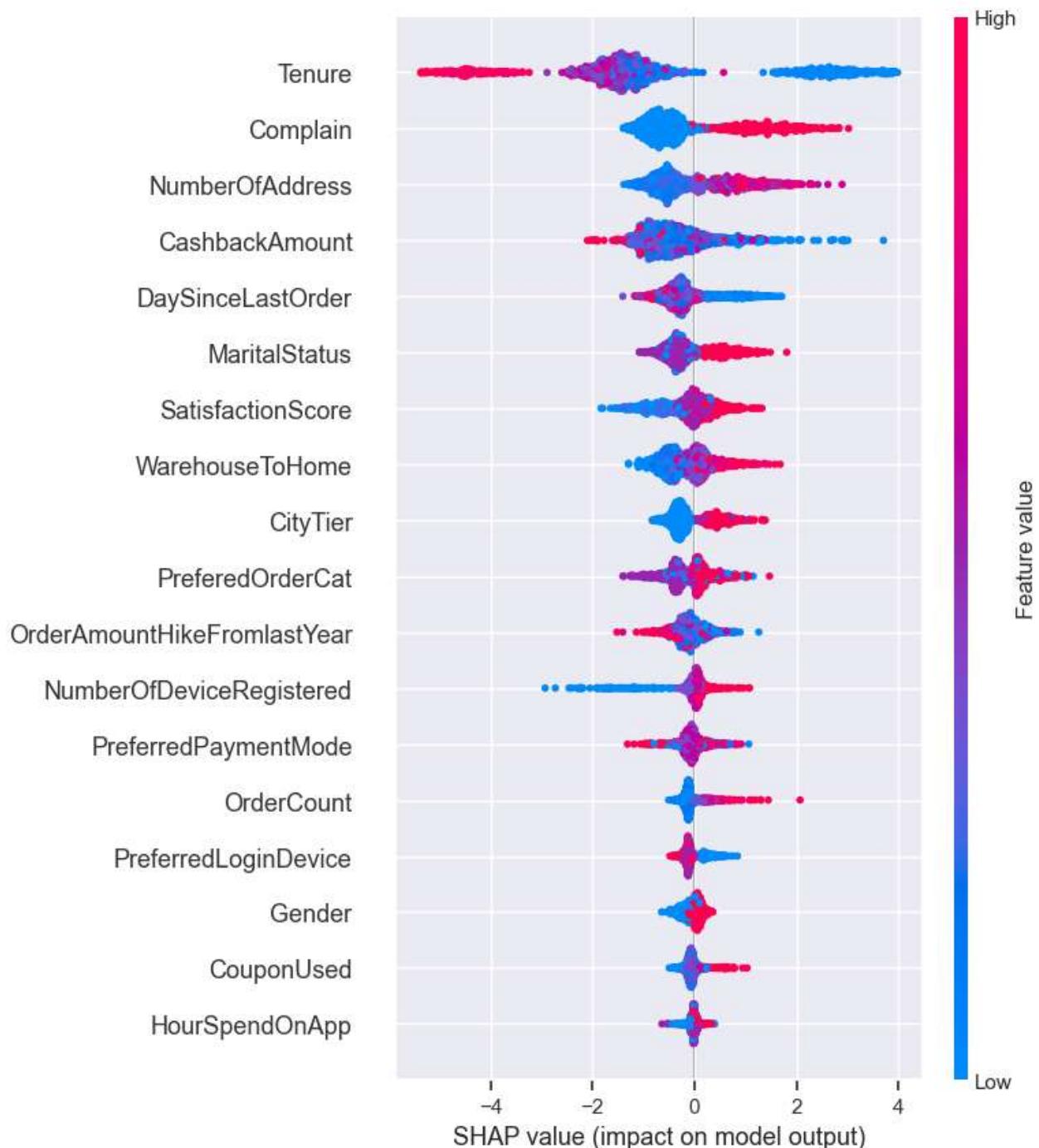
```
In [48]: from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

Accuracy Score : 97.78 %

```
In [49]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_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)))
```

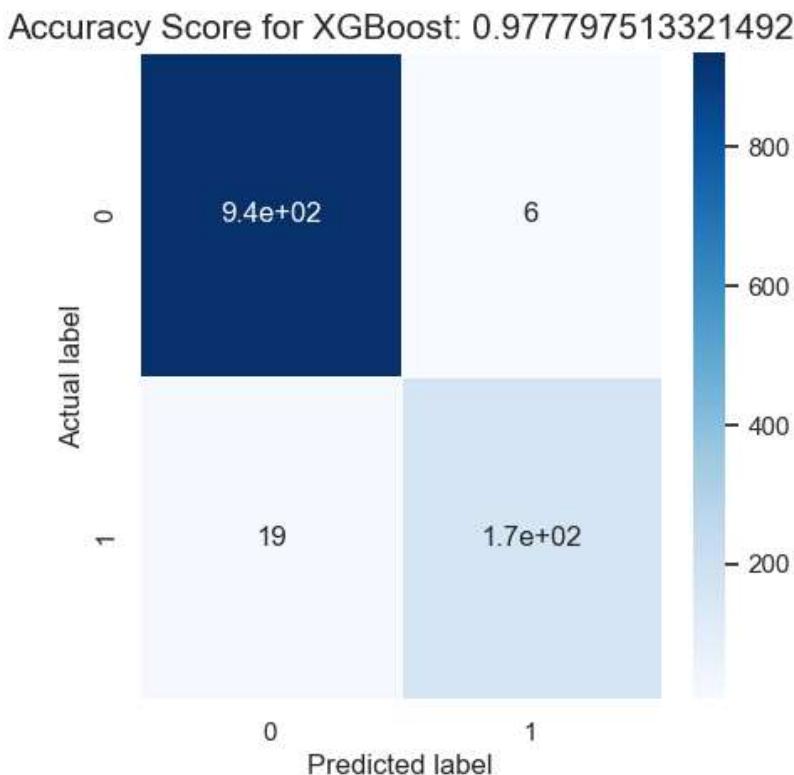
F-1 Score : 0.977797513321492  
 Precision Score : 0.977797513321492  
 Recall Score : 0.977797513321492  
 Jaccard Score : 0.9565595134665508  
 Log Loss : 0.7668509835322816

```
In [50]: import shap  
explainer = shap.TreeExplainer(xgb)  
shap_values = explainer.shap_values(X_test)  
shap.summary_plot(shap_values, X_test)
```



```
In [52]: 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 XGBoost: {0}'.format(xgb.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

```
Out[52]: Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.977797513321492')
```



```
In [53]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = xgb.predict_proba(X_test)[:,1]

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

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

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[53]: <matplotlib.legend.Legend at 0x247eeb33850>
```

