In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_theme(color_codes=True)
pd.set_option('display.max_columns', None)
```

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

Out[2]:

g_Q4_Q1	Total_Trans_Amt	Total_Trans_Ct	Total_Ct_Chng_Q4_Q1	Avg_Utilization_Ratio	Naive_Bayes_Classifier_Att
1.335	1144	42	1.625	0.061	
1.541	1291	33	3.714	0.105	
2.594	1887	20	2.333	0.000	
1.405	1171	20	2.333	0.760	
2.175	816	28	2.500	0.000	

In [3]:
```
df.nunique()
```

Out[3]:
```
CLIENTNUM
10127
Attrition_Flag
2
Customer_Age
45
Gender
2
Dependent_count
6
Education_Level
7
Marital_Status
4
Income_Category
6
Card_Category
4
Months_on_book
44
Total_Relationship_Count
6
Months_Inactive_12_mon
7
Contacts_Count_12_mon
7
Credit_Limit
6205
Total_Revolving_Bal
1974
Avg_Open_To_Buy
6813
Total_Amt_Chng_Q4_Q1
1158
Total_Trans_Amt
5033
Total_Trans_Ct
126
Total_Ct_Chng_Q4_Q1
830
Avg_Utilization_Ratio
964
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_
Education_Level_Months_Inactive_12_mon_1     1704
Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mon_Dependent_count_
Education_Level_Months_Inactive_12_mon_2      640
dtype: int64
```

Data Preprocessing Part 1

In [4]:
```
# Drop the last 2 columns
df = df.iloc[:, :-2]
df.head()
```

Out[4]:

	CLIENTNUM	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income
0	768805383	Existing Customer	45	M	3	High School	Married	
1	818770008	Existing Customer	49	F	5	Graduate	Single	Less
2	713982108	Existing Customer	51	M	3	Graduate	Married	
3	769911858	Existing Customer	40	F	4	High School	Unknown	Less
4	709106358	Existing Customer	40	M	3	Uneducated	Married	

In [5]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[5]:
```
Attrition_Flag     2
Gender             2
Education_Level    7
Marital_Status     4
Income_Category    6
Card_Category      4
dtype: int64
```

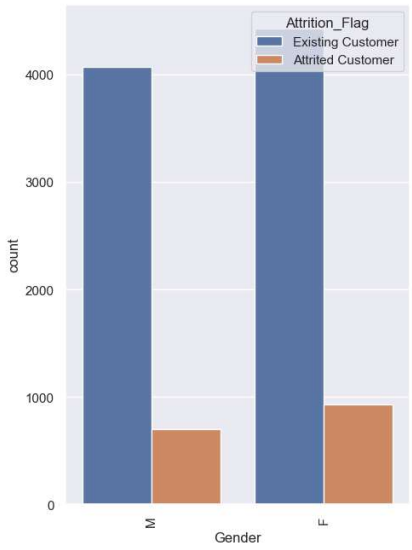
In [6]:
```
# Drop Identifier Column
df.drop(columns = 'CLIENTNUM', inplace = True)
df.head()
```

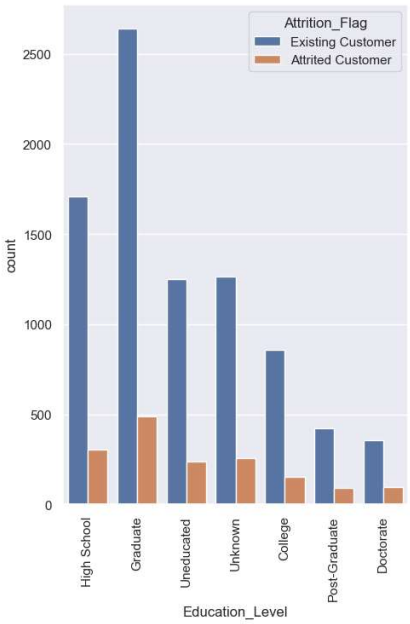
Out[6]:

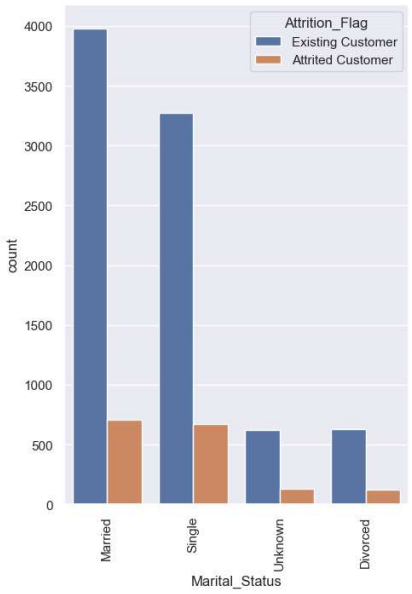
	Attrition_Flag	Customer_Age	Gender	Dependent_count	Education_Level	Marital_Status	Income_Category	C
0	Existing Customer	45	M	3	High School	Married	$60K - 80K	
1	Existing Customer	49	F	5	Graduate	Single	Less than $40K	
2	Existing Customer	51	M	3	Graduate	Married	$80K - 120K	
3	Existing Customer	40	F	4	High School	Unknown	Less than $40K	
4	Existing Customer	40	M	3	Uneducated	Married	$60K - 80K	

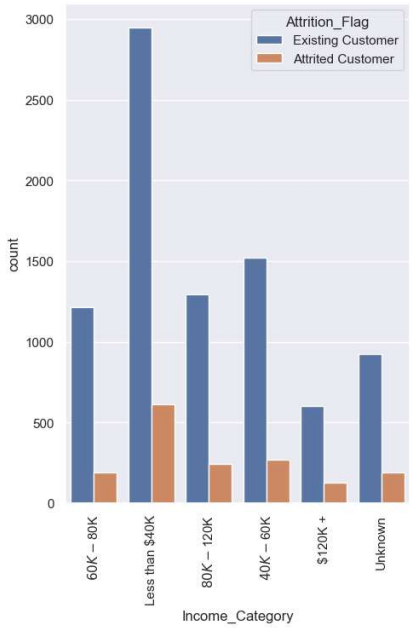
Exploratory Data Analysis

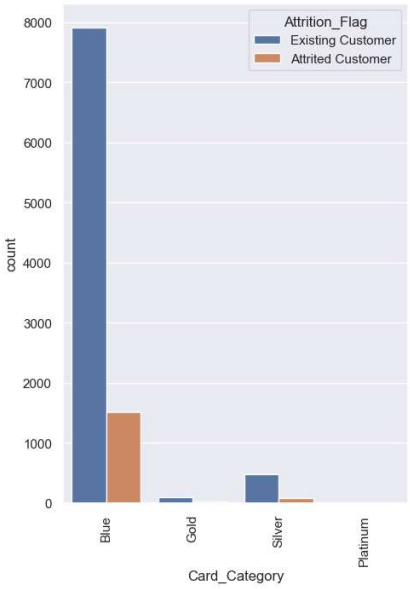
In [7]:
```
# list of categorical variables to plot
cat_vars = ['Gender', 'Education_Level', 'Marital_Status',
            'Income_Category', 'Card_Category']

# 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='Attrition_Flag', 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()
```


Attrition_Flag
Existing Customer
Attrited Customer
count
Gender
M
F

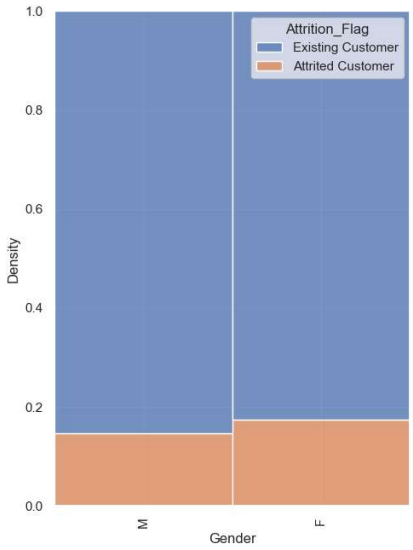

Attrition_Flag
Existing Customer
Attrited Customer
count
Education_Level
High School
Graduate
Uneducated
Unknown
College
Post-Graduate
Doctorate

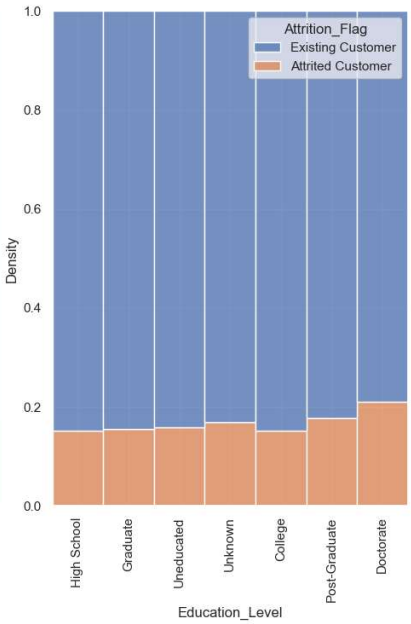

Attrition_Flag
Existing Customer
Attrited Customer
count
Marital_Status
Married
Single
Unknown
Divorced

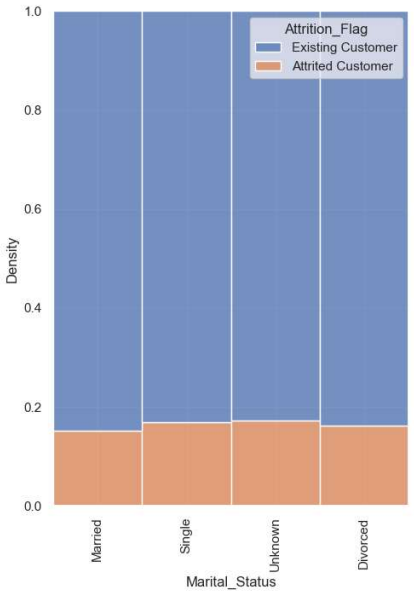

Attrition_Flag
Existing Customer
Attrited Customer
count
Income_Category
60K – 80K
Less than $40K
80K – 120K
40K – 60K
$120K +
Unknown

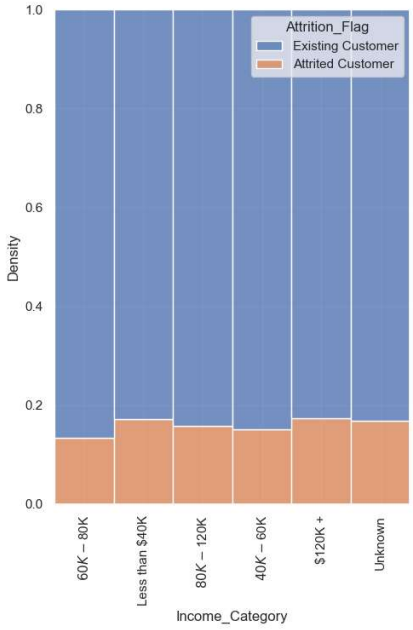

Attrition_Flag
Existing Customer
Attrited Customer
count
Card_Category
Blue
Gold
Silver
Platinum

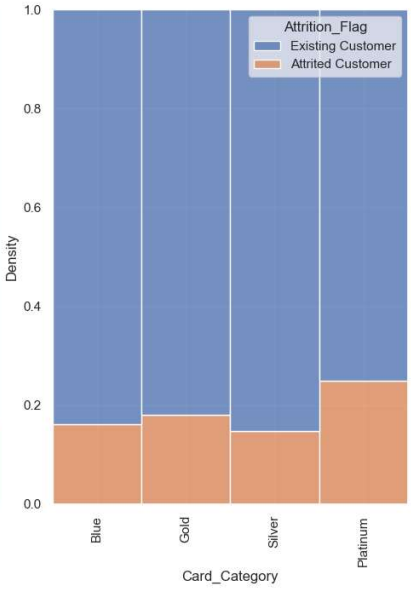
In [9]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Gender', 'Education_Level', 'Marital_Status',
            'Income_Category', 'Card_Category']

# 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='Attrition_Flag', data=df, ax=axs[i], multiple="fill", kde=Fals
    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()
```


Attrition_Flag
Existing Customer
Attrited Customer
Density
1.0
0.8
0.6
0.4
0.2
0.0
M
F
Gender

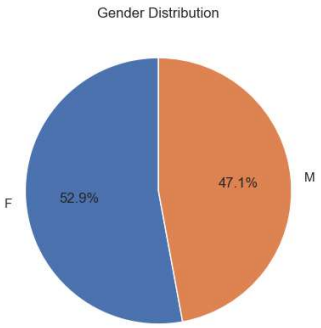

Attrition_Flag
Existing Customer
Attrited Customer
Density
1.0
0.8
0.6
0.4
0.2
0.0
High School
Graduate
Uneducated
Unknown
College
Post-Graduate
Doctorate
Education_Level

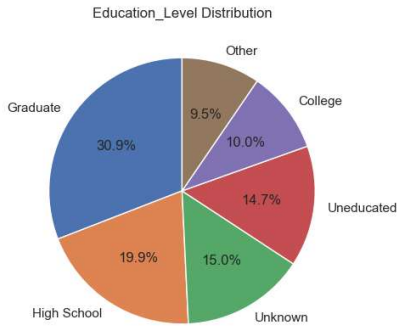

Attrition_Flag
Existing Customer
Attrited Customer
Density
1.0
0.8
0.6
0.4
0.2
0.0
Married
Single
Unknown
Divorced
Marital_Status

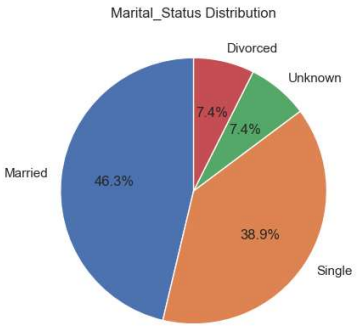

Attrition_Flag
Existing Customer
Attrited Customer
Density
1.0
0.8
0.6
0.4
0.2
0.0
60K – 80K
Less than $40K
80K – 120K
40K – 60K
$120K +
Unknown
Income_Category

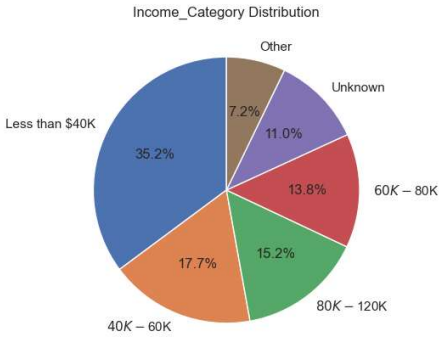

Attrition_Flag
Existing Customer
Attrited Customer
Density
1.0
0.8
0.6
0.4
0.2
0.0
Blue
Gold
Silver
Platinum
Card_Category

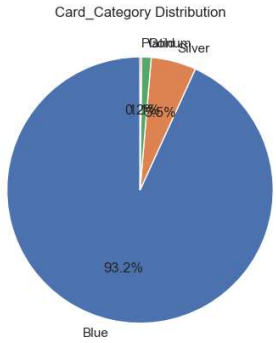
In [10]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

# Filter categorical columns with 'object' data type
cat_cols = [col for col in df.columns if col != 'Attrition_Flag' and df[col].dtype == 'obje

# Create a figure with subplots
num_cols = len(cat_cols)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))

# Flatten the axs array for easier indexing
axs = axs.flatten()

# Create a pie chart for each categorical column
for i, col in enumerate(cat_cols):
    if i < len(axs):  # Ensure we don't exceed the number of subplots
        # Count the number of occurrences for each category
        cat_counts = df[col].value_counts()

        # Group categories beyond the top max_categories as 'Other'
        if len(cat_counts) > max_categories:
            cat_counts_top = cat_counts[:max_categories]
            cat_counts_other = pd.Series(cat_counts[max_categories:].sum(), index=['Other']
            cat_counts = cat_counts_top.append(cat_counts_other)

        # Create a pie chart
        axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)
        axs[i].set_title(f'{col} Distribution')

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Gender Distribution
F
52.9%
47.1%
M

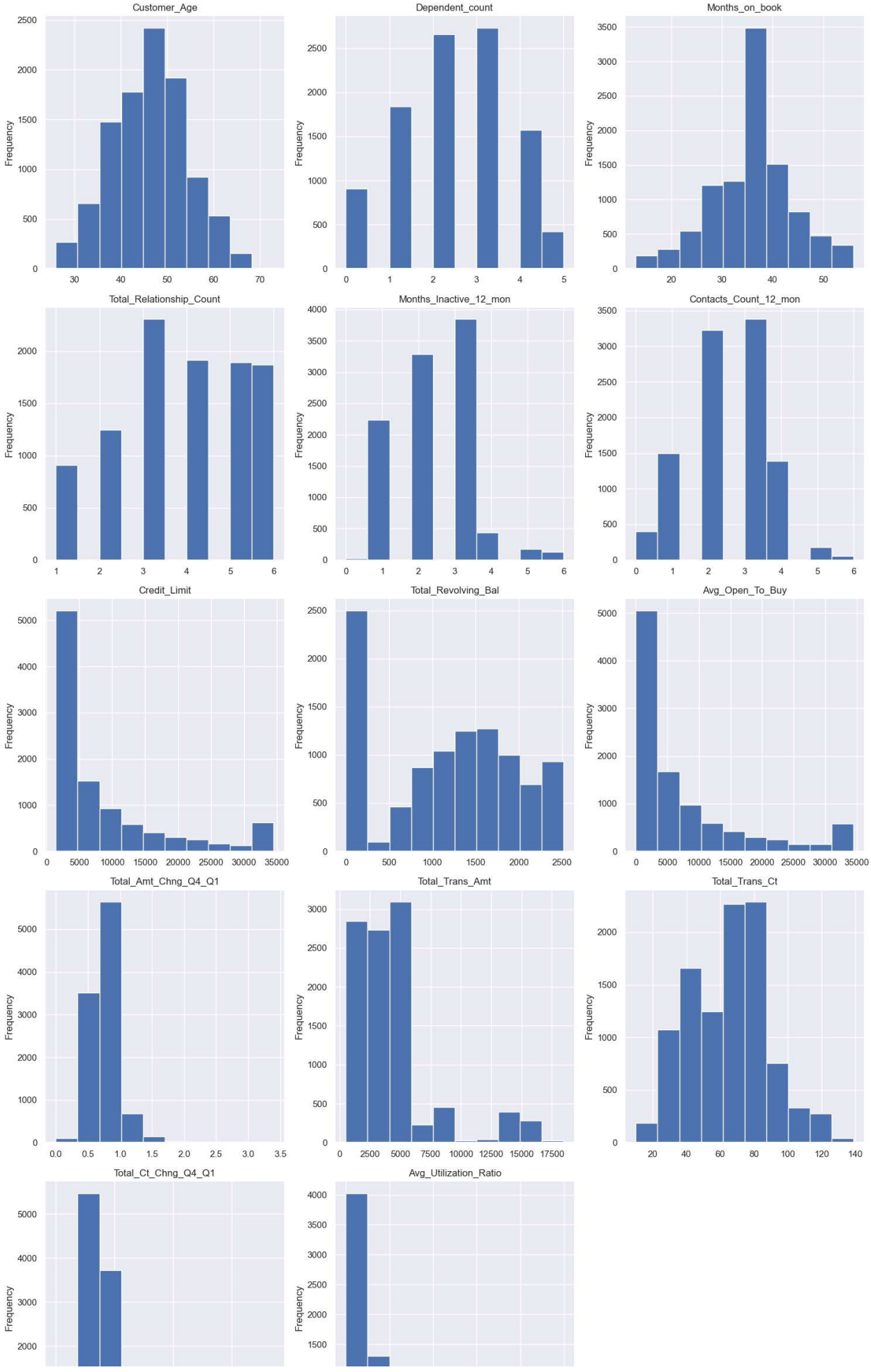

Education_Level Distribution
Other
9.5%
College
10.0%
Graduate
30.9%
Uneducated
14.7%
19.9%
15.0%
High School
Unknown

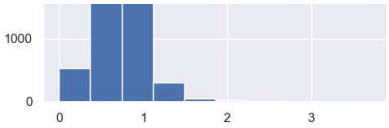

Marital_Status Distribution
Divorced
Unknown
7.4%
7.4%
Married
46.3%
38.9%
Single

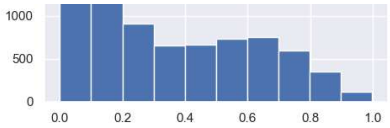

Income_Category Distribution
Other
Unknown
7.2%
11.0%
Less than $40K
35.2%
13.8%
60K – 80K
17.7%
15.2%
80K – 120K
40K – 60K


Card_Category Distribution
Silver
93.2%
Blue

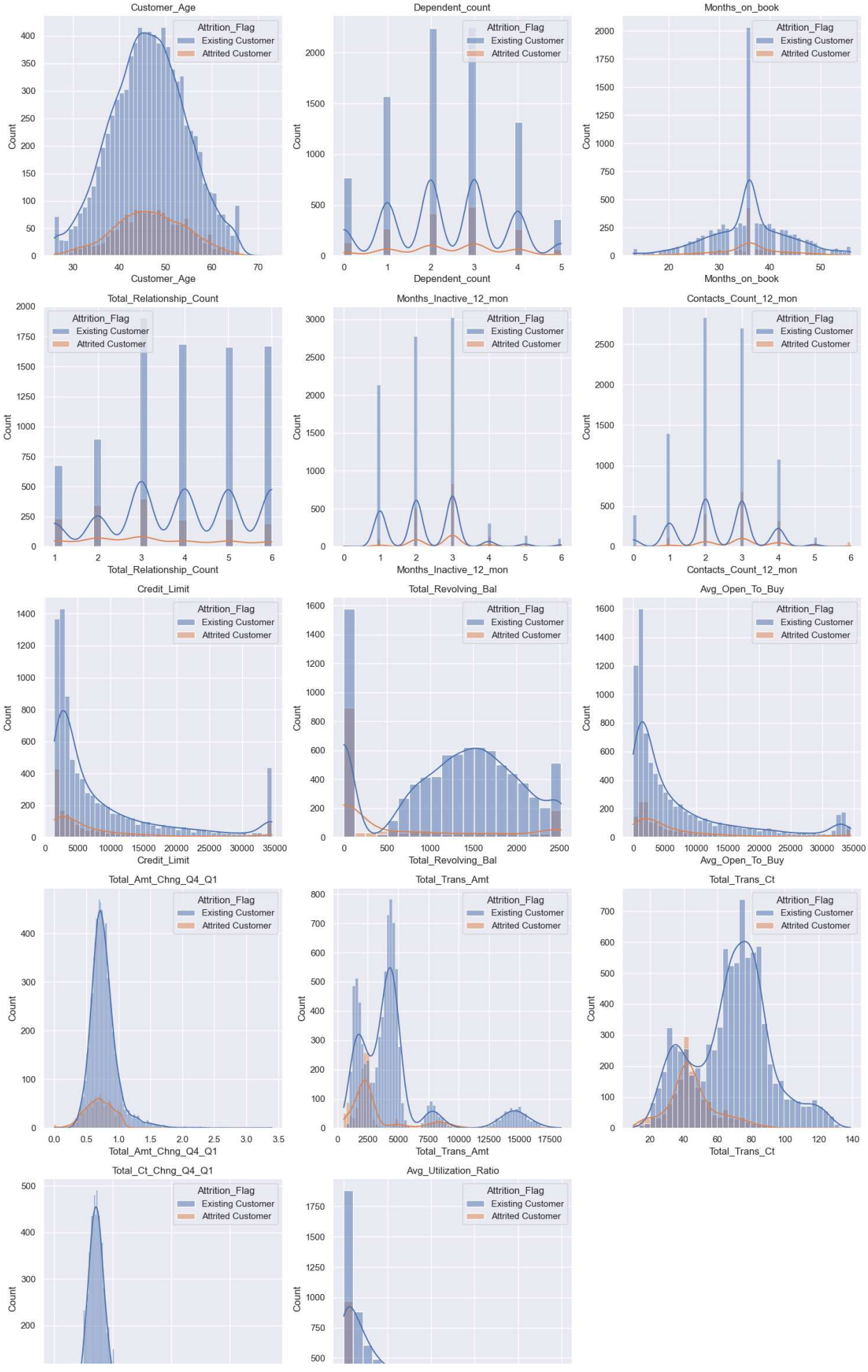
In [11]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable
for i, var in enumerate(int_vars):
    df[var].plot.hist(ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

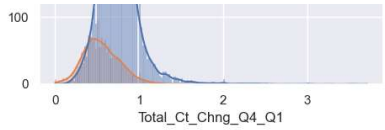

Customer_Age
Dependent_count
Months_on_book
Total_Relationship_Count
Months_Inactive_12_mon
Contacts_Count_12_mon
Credit_Limit
Total_Revolving_Bal
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1
Total_Trans_Amt
Total_Trans_Ct
Total_Ct_Chng_Q4_Q1
Avg_Utilization_Ratio
Frequency

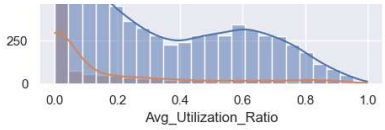

1000
0
0
1
2
3


1000
500
0
0.0
0.2
0.4
0.6
0.8
1.0

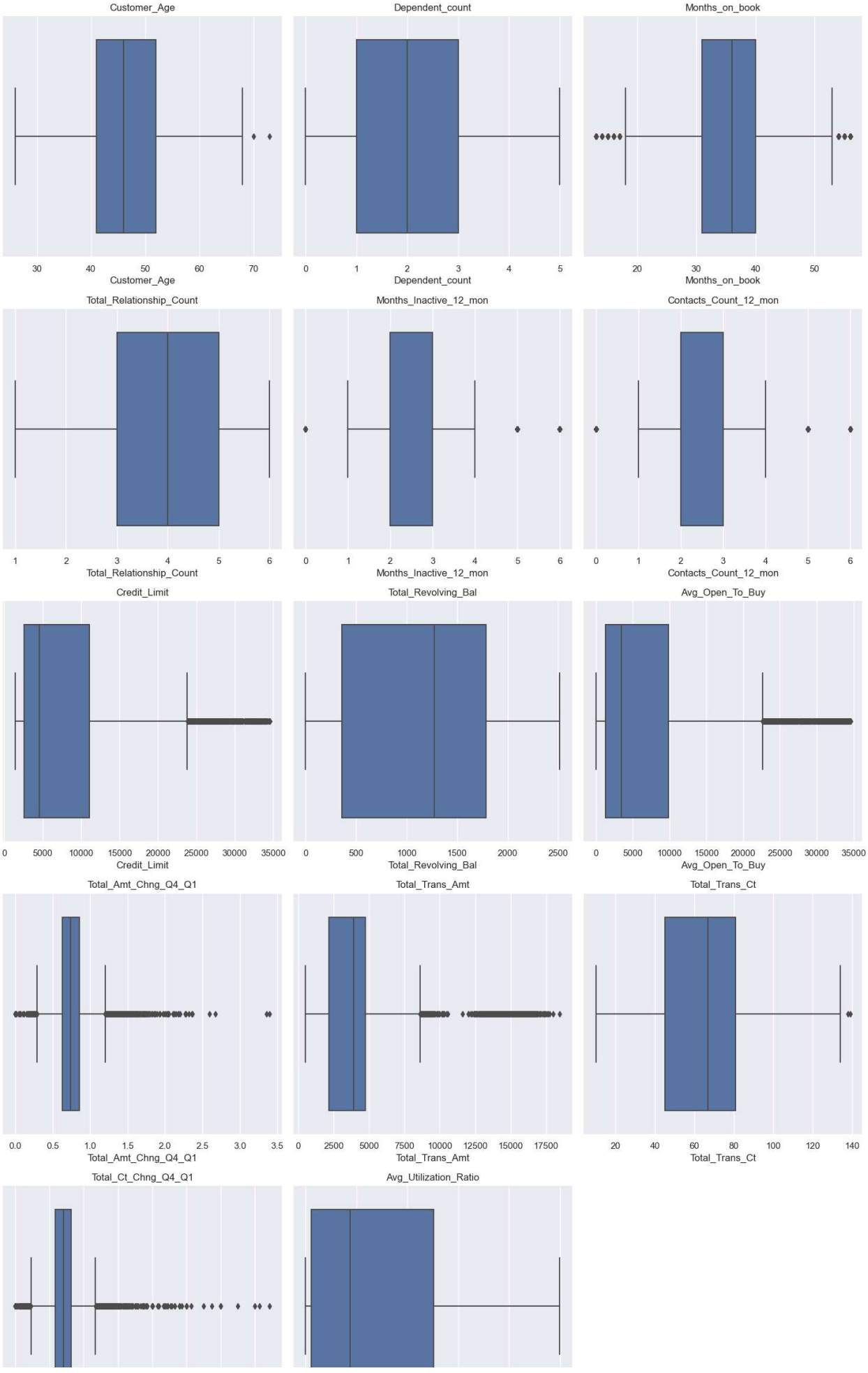
In [12]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    sns.histplot(data=df, x=var, hue='Attrition_Flag', kde=True, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

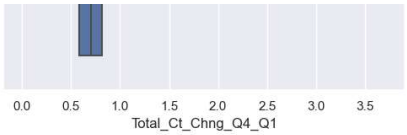

Customer_Age
Attrition_Flag
Existing Customer
Attrited Customer
Count
Customer_Age
Dependent_count
Dependent_count
Months_on_book
Months_on_book
Total_Relationship_Count
Total_Relationship_Count
Months_Inactive_12_mon
Months_Inactive_12_mon
Contacts_Count_12_mon
Contacts_Count_12_mon
Credit_Limit
Credit_Limit
Total_Revolving_Bal
Total_Revolving_Bal
Avg_Open_To_Buy
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1
Total_Amt_Chng_Q4_Q1
Total_Trans_Amt
Total_Trans_Amt
Total_Trans_Ct
Total_Trans_Ct
Total_Ct_Chng_Q4_Q1
Avg_Utilization_Ratio

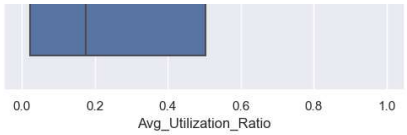

100
0
0
1
2
3
Total_Ct_Chng_Q4_Q1


250
0
0.0
0.2
0.4
0.6
0.8
1.0
Avg_Utilization_Ratio

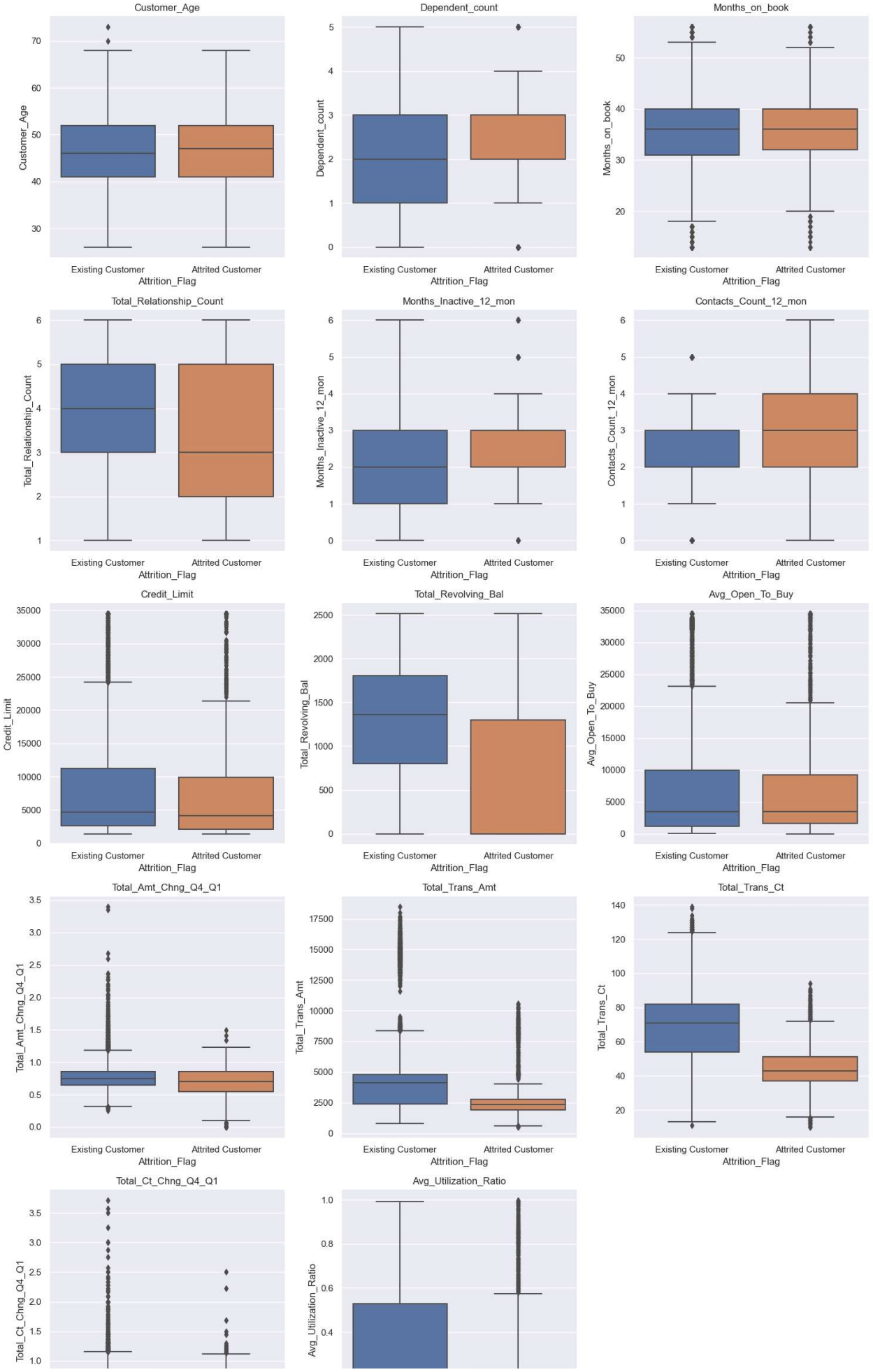
In [13]:
```
# Get the names of all columns with data type 'int' or 'float'
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(num_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each numerical variable using Seaborn
for i, var in enumerate(num_vars):
    sns.boxplot(x=df[var], ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```

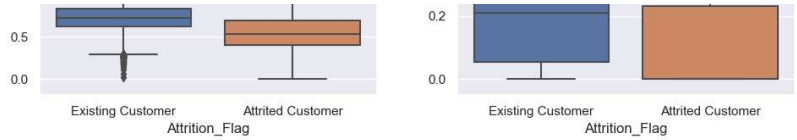

Customer_Age
30 40 50 60 70
Customer_Age
Dependent_count
0 1 2 3 4 5
Dependent_count
Months_on_book
20 30 40 50
Months_on_book
Total_Relationship_Count
1 2 3 4 5 6
Total_Relationship_Count
Months_Inactive_12_mon
0 1 2 3 4 5 6
Months_Inactive_12_mon
Contacts_Count_12_mon
0 1 2 3 4 5 6
Contacts_Count_12_mon
Credit_Limit
0 5000 10000 15000 20000 25000 30000 35000
Credit_Limit
Total_Revolving_Bal
0 500 1000 1500 2000 2500
Total_Revolving_Bal
Avg_Open_To_Buy
0 5000 10000 15000 20000 25000 30000 35000
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1
0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5
Total_Amt_Chng_Q4_Q1
Total_Trans_Amt
0 2500 5000 7500 10000 12500 15000 17500
Total_Trans_Amt
Total_Trans_Ct
20 40 60 80 100 120 140
Total_Trans_Ct
Total_Ct_Chng_Q4_Q1
Avg_Utilization_Ratio


0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
Total_Ct_Chng_Q4_Q1

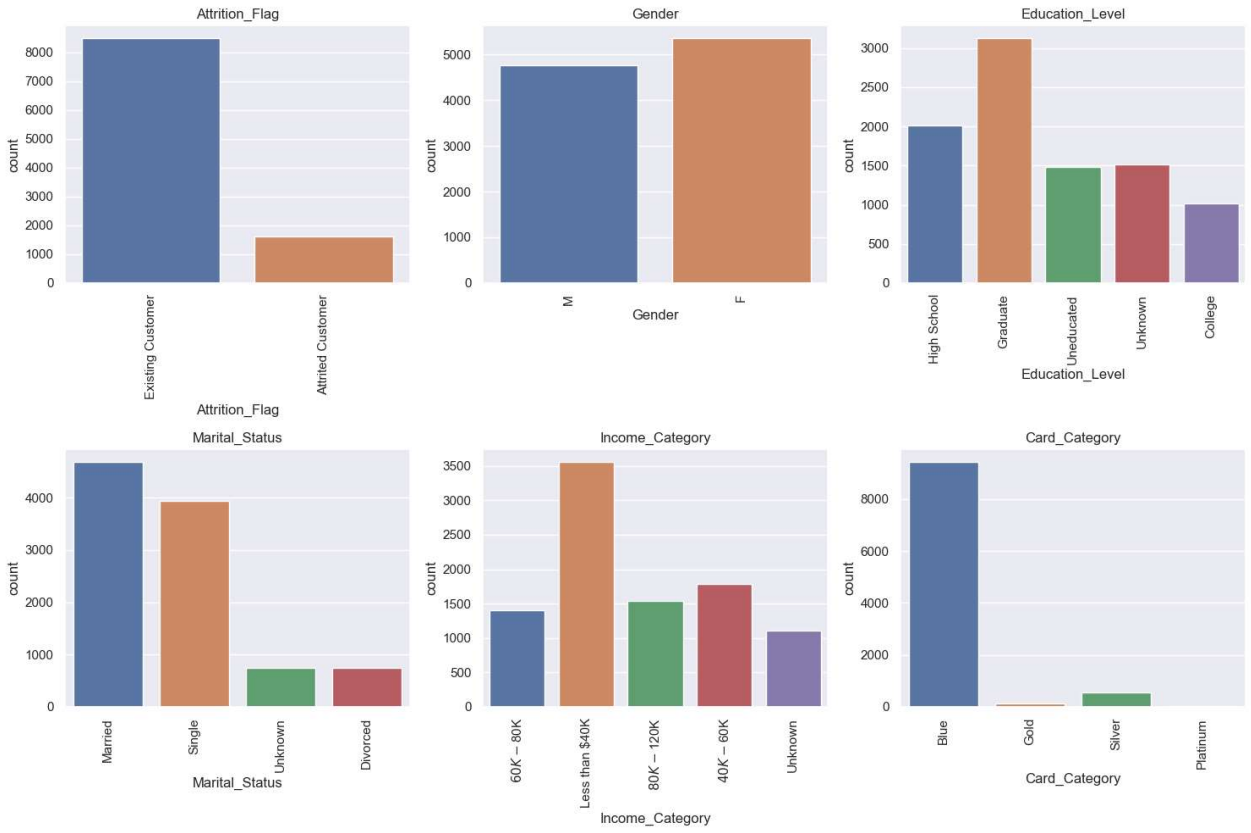

0.0
0.2
0.4
0.6
0.8
1.0
Avg_Utilization_Ratio

In [15]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a box plot for each integer variable using Seaborn with hue='attrition'
for i, var in enumerate(int_vars):
    sns.boxplot(y=var, x='Attrition_Flag', data=df, ax=axs[i])
    axs[i].set_title(var)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```


Customer_Age
Customer_Age
Existing Customer
Attrited Customer
Attrition_Flag
Dependent_count
Dependent_count
Months_on_book
Months_on_book
Total_Relationship_Count
Total_Relationship_Count
Months_Inactive_12_mon
Months_Inactive_12_mon
Contacts_Count_12_mon
Contacts_Count_12_mon
Credit_Limit
Credit_Limit
Total_Revolving_Bal
Total_Revolving_Bal
Avg_Open_To_Buy
Avg_Open_To_Buy
Total_Amt_Chng_Q4_Q1
Total_Amt_Chng_Q4_Q1
Total_Trans_Amt
Total_Trans_Amt
Total_Trans_Ct
Total_Trans_Ct
Total_Ct_Chng_Q4_Q1
Total_Ct_Chng_Q4_Q1
Avg_Utilization_Ratio
Avg_Utilization_Ratio



In [16]:
```
# Get the names of all columns with data type 'object' (categorical columns)
cat_vars = df.select_dtypes(include='object').columns.tolist()

# Create a figure with subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a countplot for the top 5 values of each categorical variable using Seaborn
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(5).index
    filtered_df = df[df[var].isin(top_values)]
    sns.countplot(x=var, data=filtered_df, ax=axs[i])
    axs[i].set_title(var)
    axs[i].tick_params(axis='x', rotation=90)

# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```



Data Preprocessing Part 2

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

Out[17]:
```
Series([], dtype: float64)
```

Label Encoding for Object Datatypes

In [18]:
```
# 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()}")
```

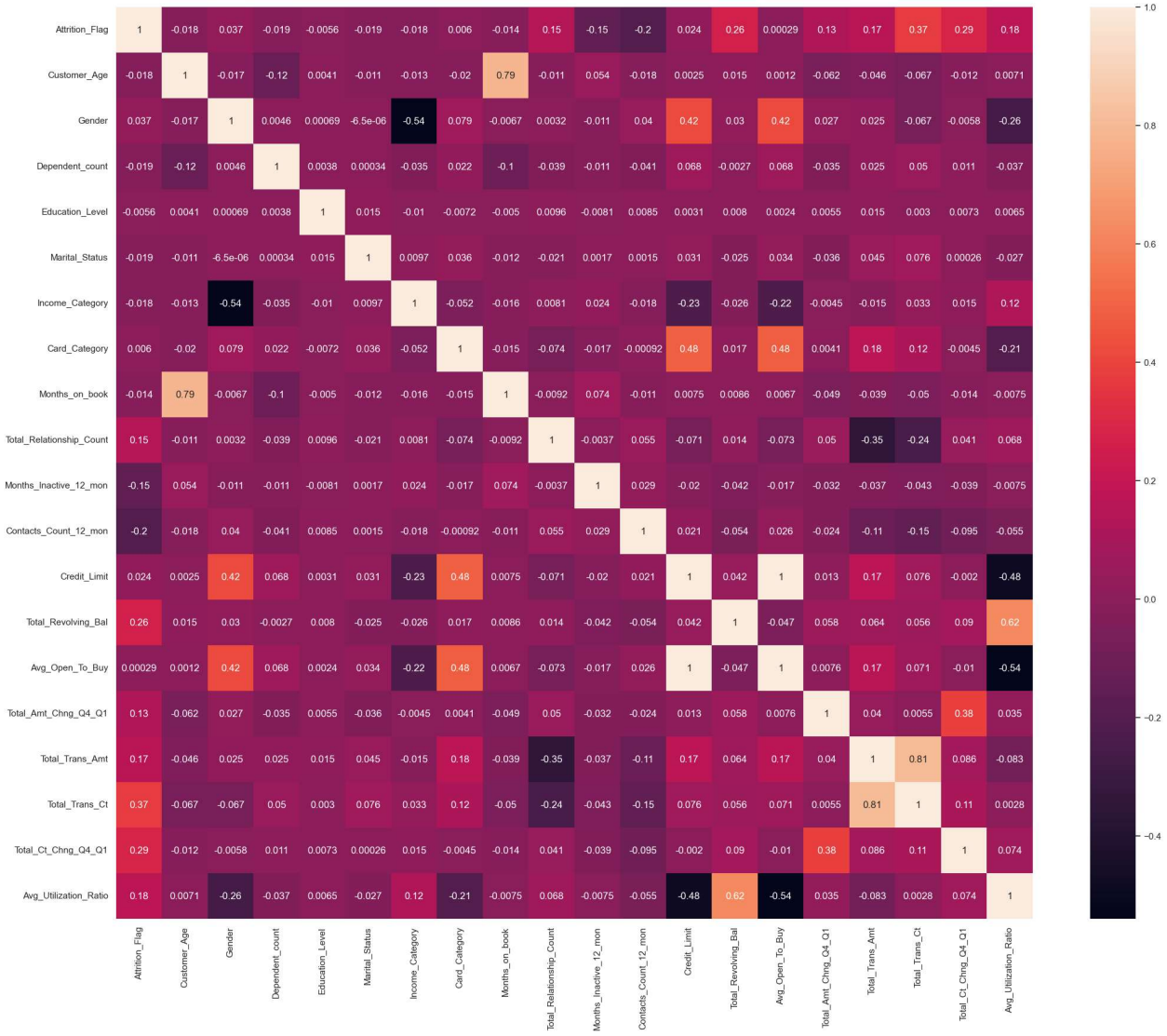
```
Attrition_Flag: ['Existing Customer' 'Attrited Customer']
Gender: ['M' 'F']
Education_Level: ['High School' 'Graduate' 'Uneducated' 'Unknown' 'College' 'Post-Graduat
e'
 'Doctorate']
Marital_Status: ['Married' 'Single' 'Unknown' 'Divorced']
Income_Category: ['$60K - $80K' 'Less than $40K' '$80K - $120K' '$40K - $60K' '$120K +'
 'Unknown']
Card_Category: ['Blue' 'Gold' 'Silver' 'Platinum']
```

In [19]:
```
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()}")
```

```
Attrition_Flag: [1 0]
Gender: [1 0]
Education_Level: [3 2 5 6 0 4 1]
Marital_Status: [1 2 3 0]
Income_Category: [2 4 3 1 0 5]
Card_Category: [0 1 3 2]
```

In [22]:
```
# Correlation Heatmap
plt.figure(figsize=(25, 20))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[22]: <AxesSubplot:>



Train Test Split

In [23]:
```
X = df.drop('Attrition_Flag', axis=1)
y = df['Attrition_Flag']
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)
```

Remove Outlier from Train Data using Z-Score

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

# Define the columns for which you want to remove outliers
selected_columns = ['Credit_Limit', 'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1',
                    'Total_Trans_Amt', 'Total_Ct_Chng_Q4_Q1']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Classifier

In [25]:
```
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 [26]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=42, max_depth=8, min_samples_leaf=1, min_sample
dtree.fit(X_train, y_train)
```

Out[26]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=8, random_state=42)
```

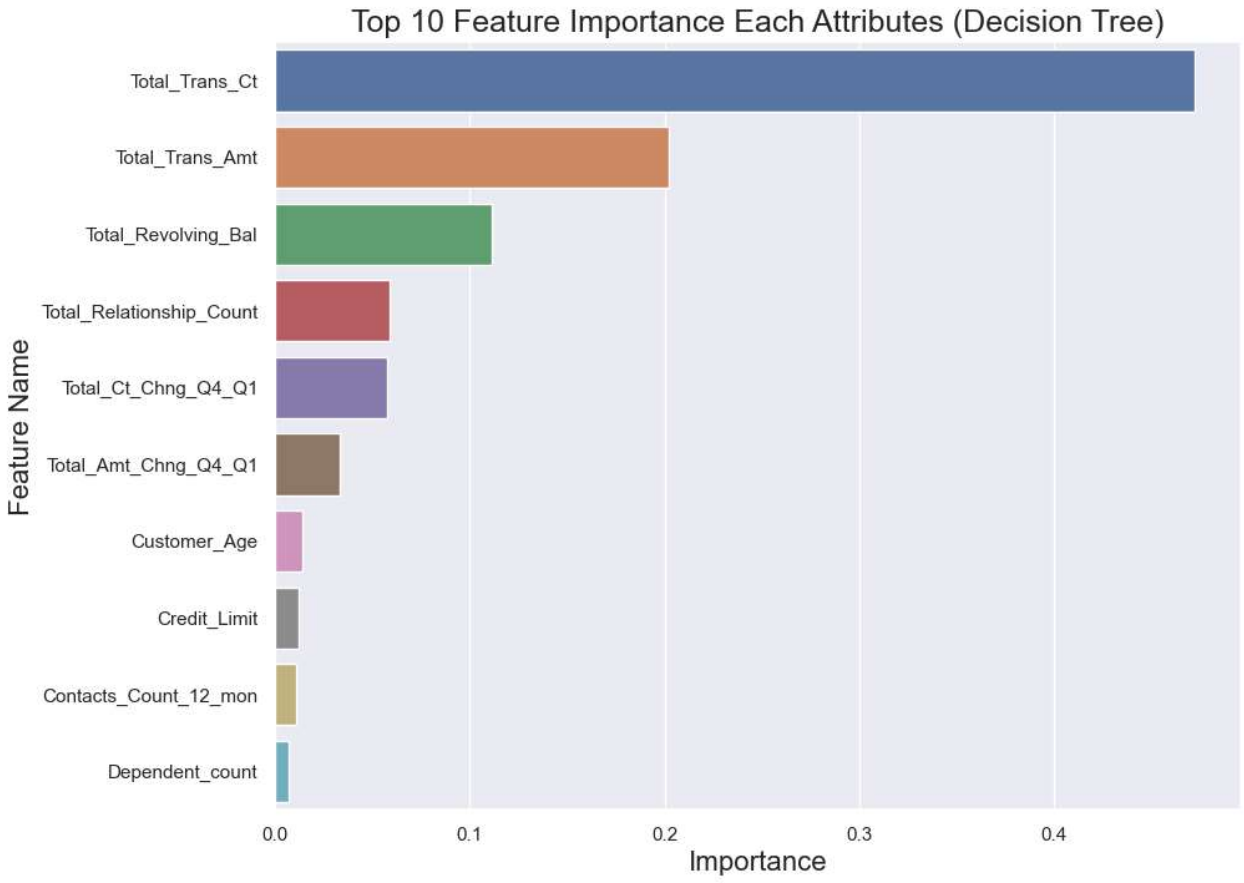
In [27]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 92.6 %
```

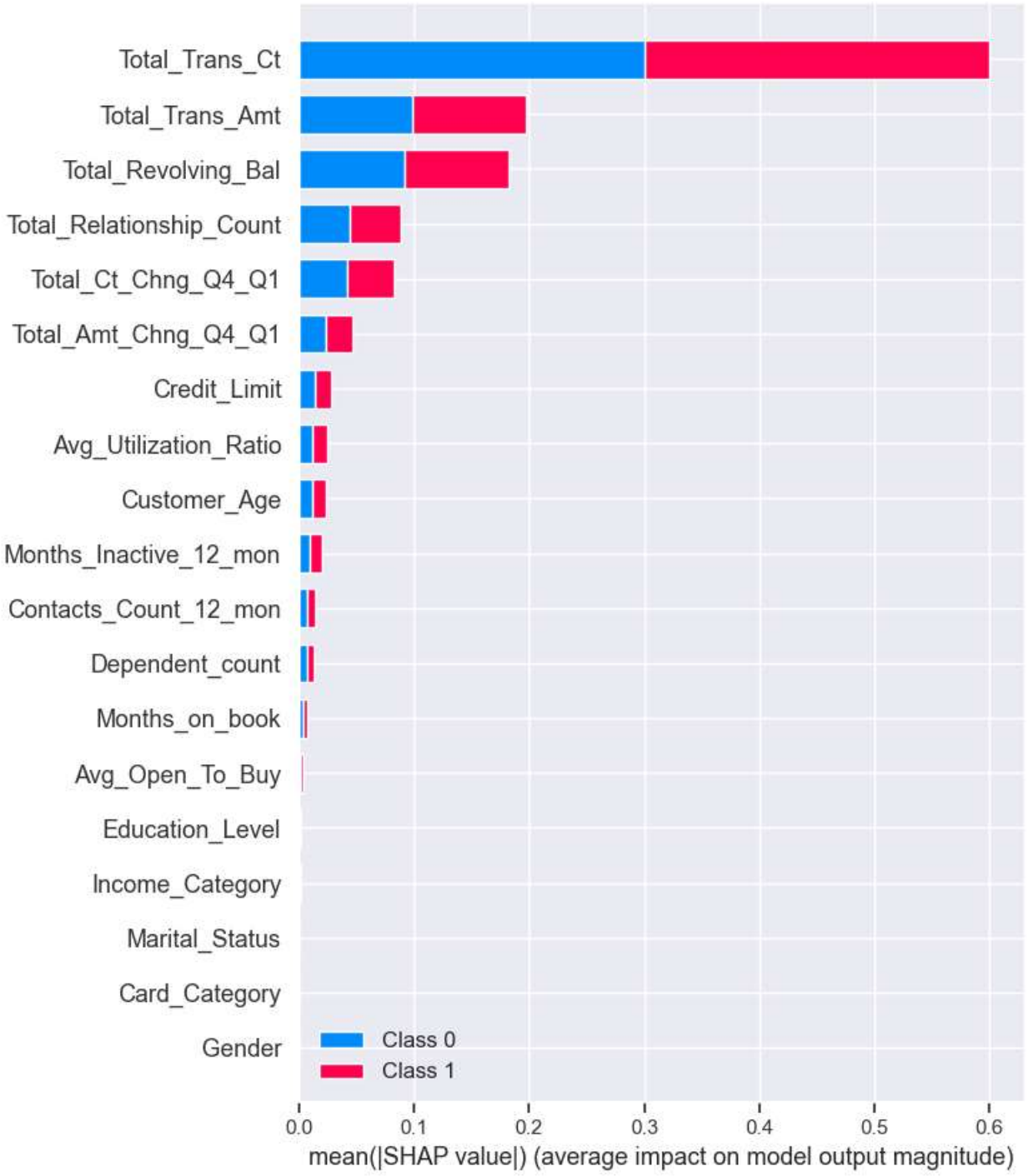
In [28]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccar
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, 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 :  0.9259624876604146
Precision Score :  0.9259624876604146
Recall Score :  0.9259624876604146
Jaccard Score :  0.8621323529411765
Log Loss :  2.557179291581486
```

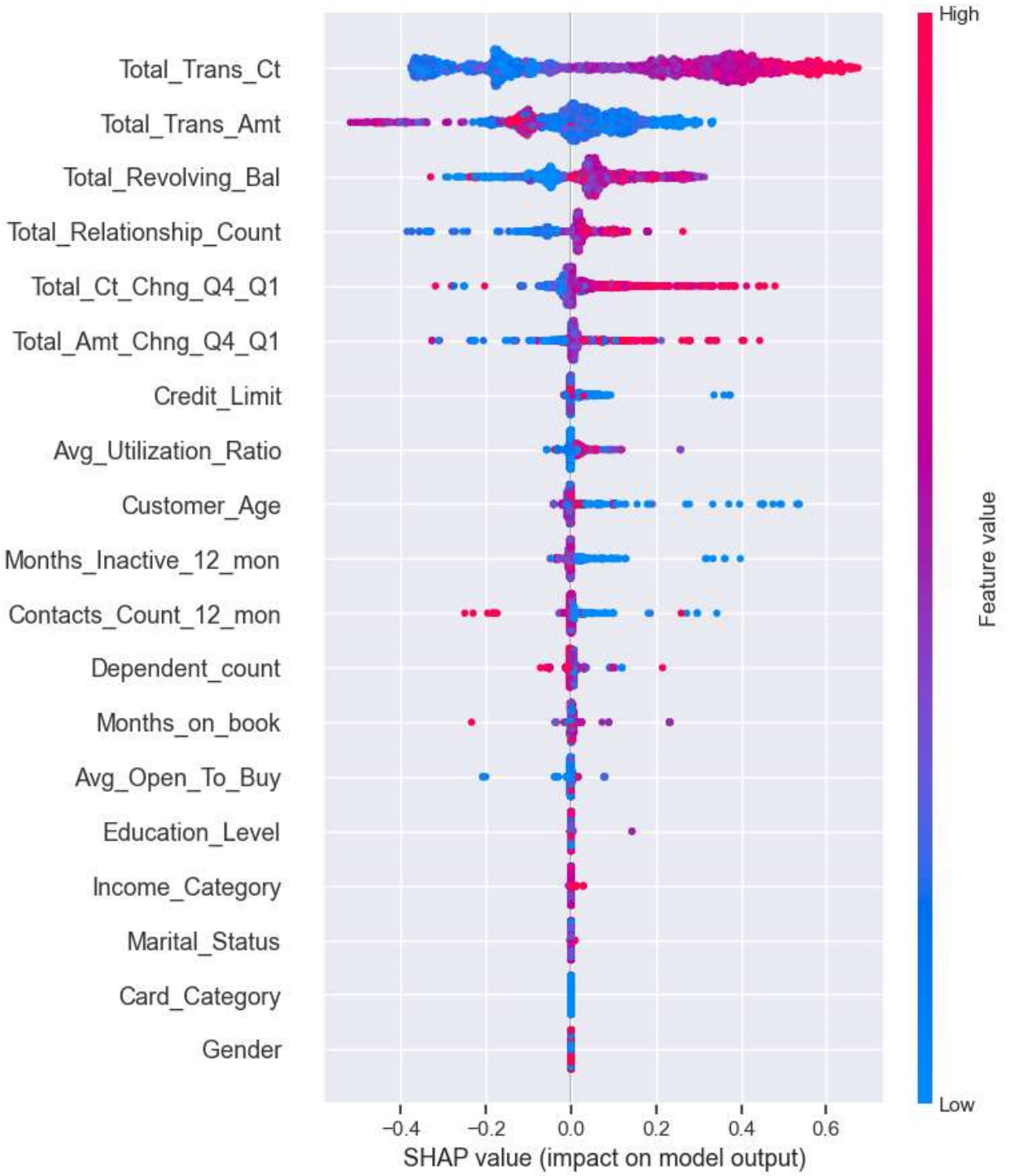
In [29]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [30]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

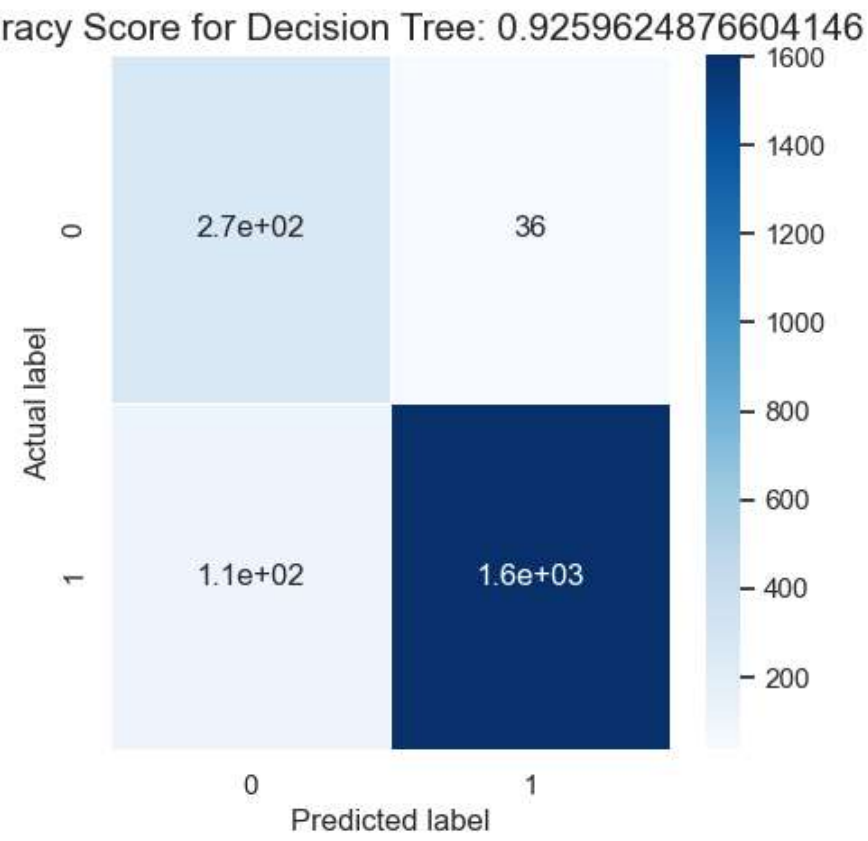


In [31]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



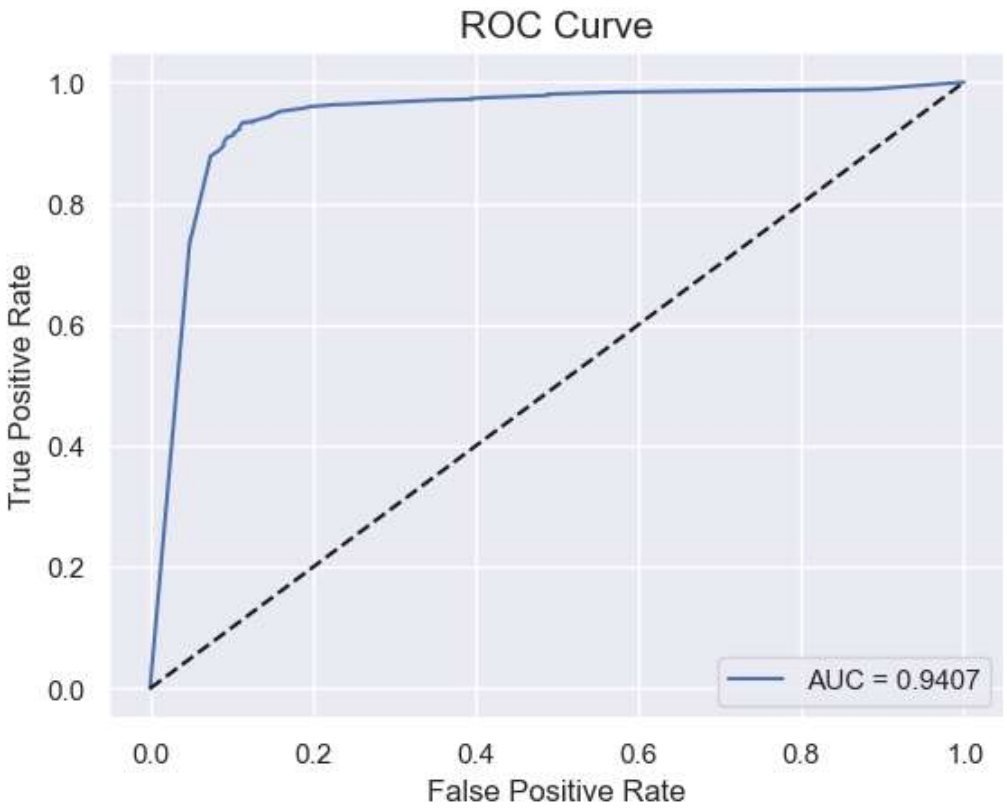
In [32]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_tes
plt.title(all_sample_title, size = 15)
```

Out[32]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9259624876604146')
```



In [33]:
```
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.D
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[33]: <matplotlib.legend.Legend at 0x258a64ab370>



Random Forest Classifier

In [34]:
```
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': None, 'n_estimators': 100, 'random_state': 0}
```

In [35]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_depth=None, max_features=None, n_estimator
rfc.fit(X_train, y_train)
```

Out[35]:
```
RandomForestClassifier(class_weight='balanced', max_features=None,
                       random_state=0)
```

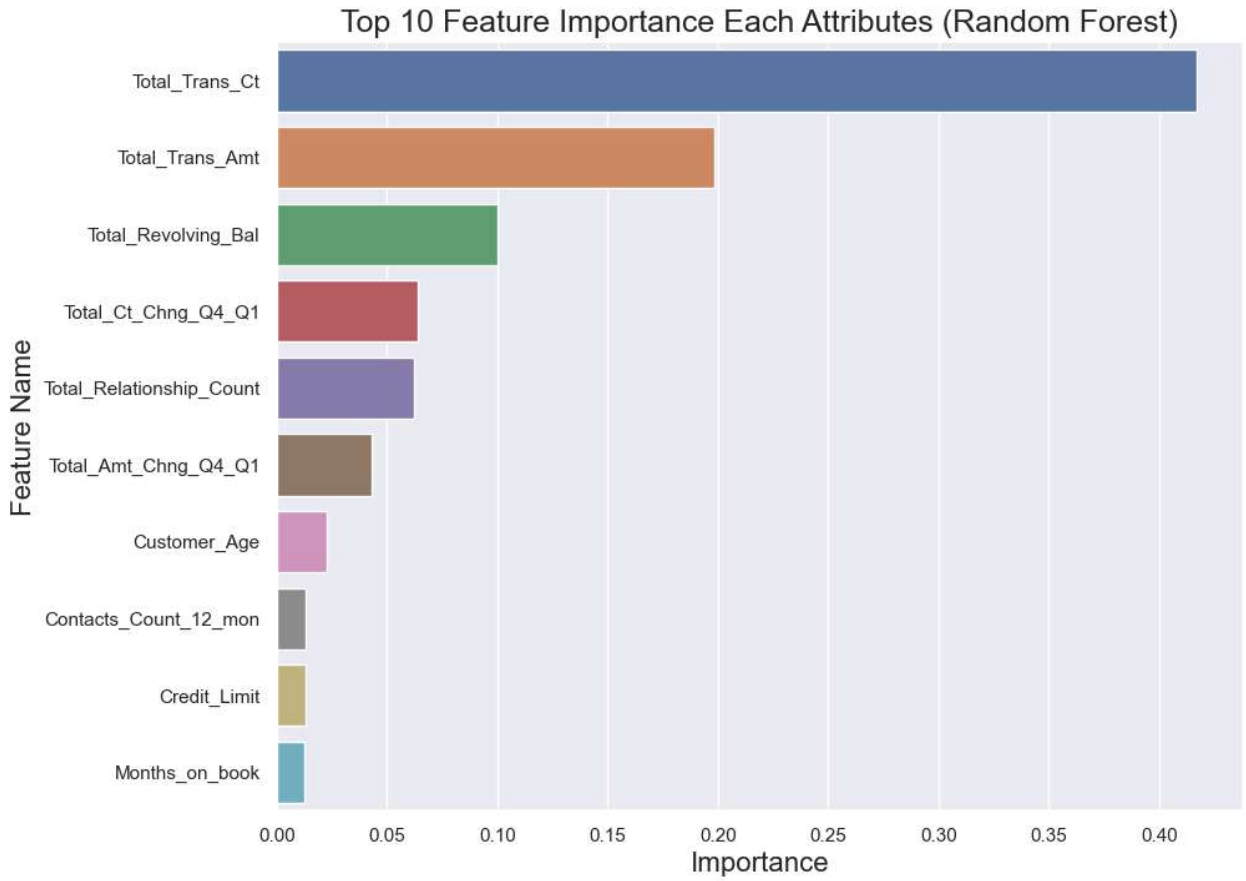
In [36]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 95.85 %
```

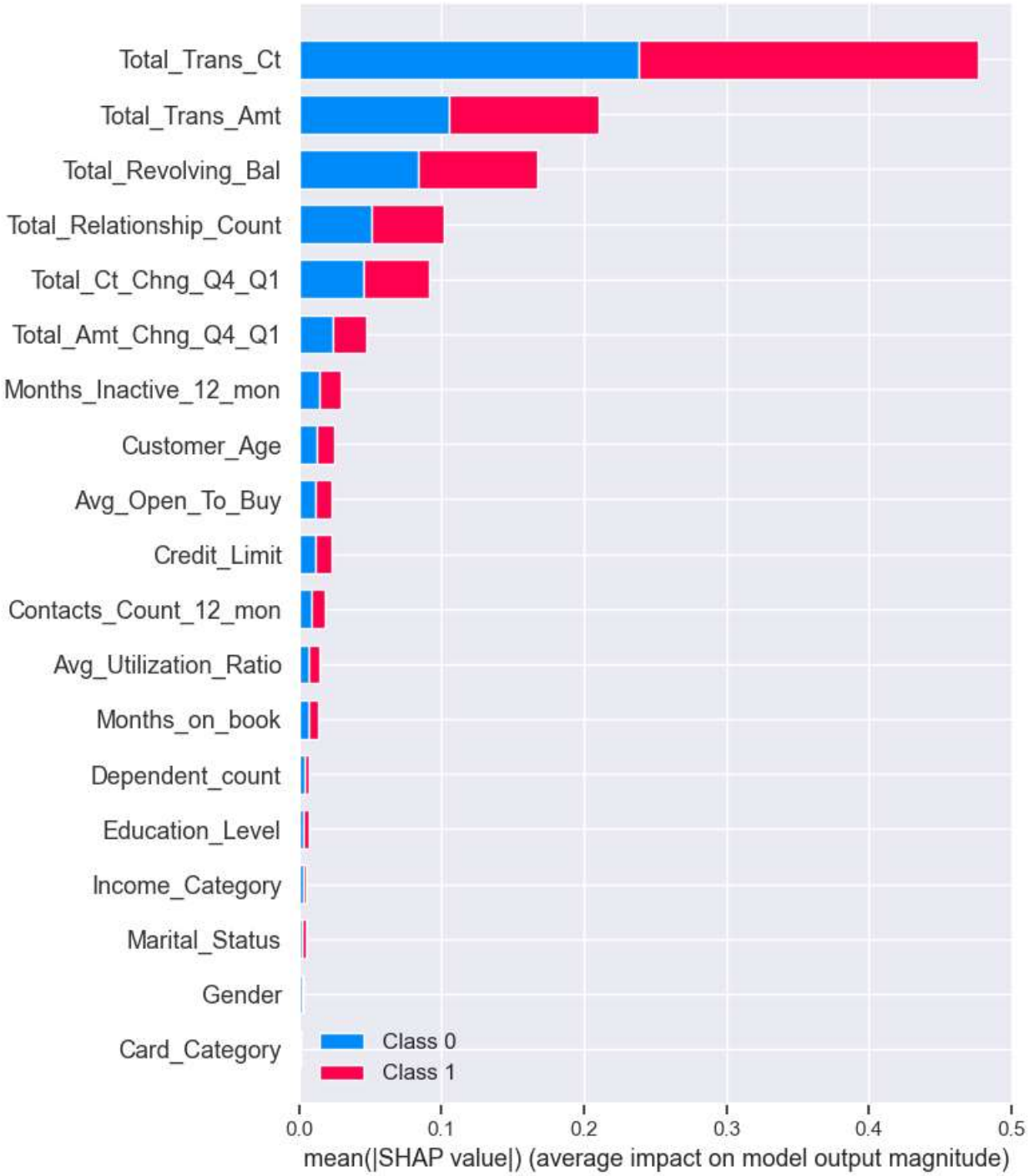
In [37]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccar
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, 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 :  0.9585389930898321
Precision Score :  0.9585389930898321
Recall Score :  0.9585389930898321
Jaccard Score :  0.9203791469194312
Log Loss :  1.4320325748476992
```

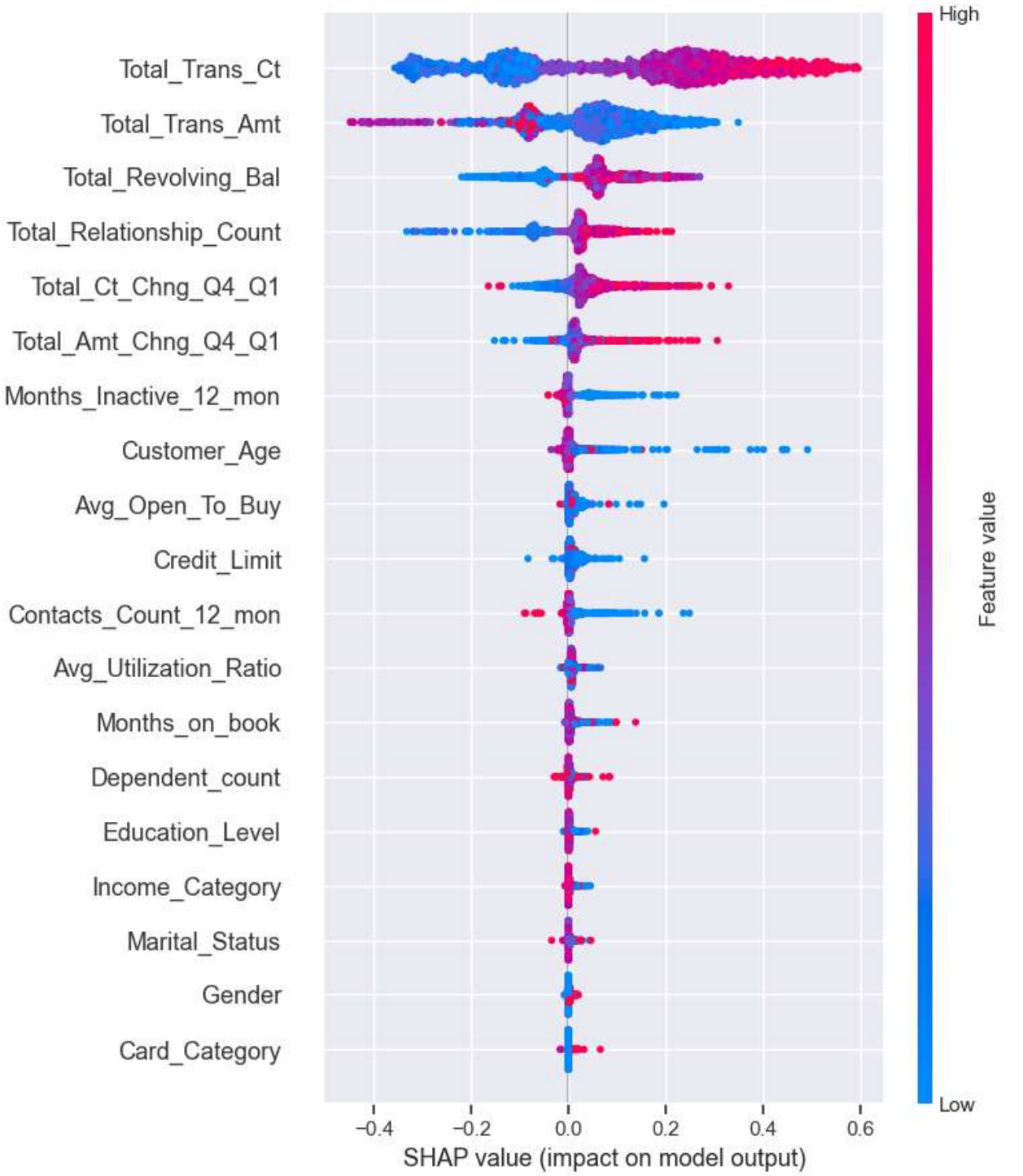
In [38]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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 [39]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

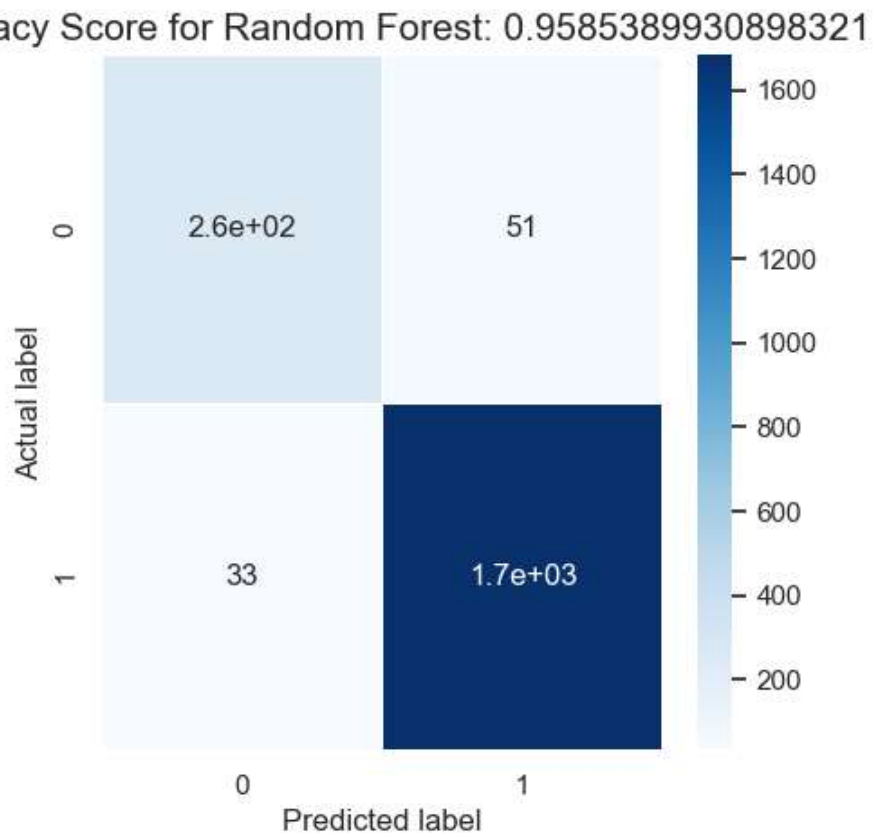


In [40]:
```
# compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



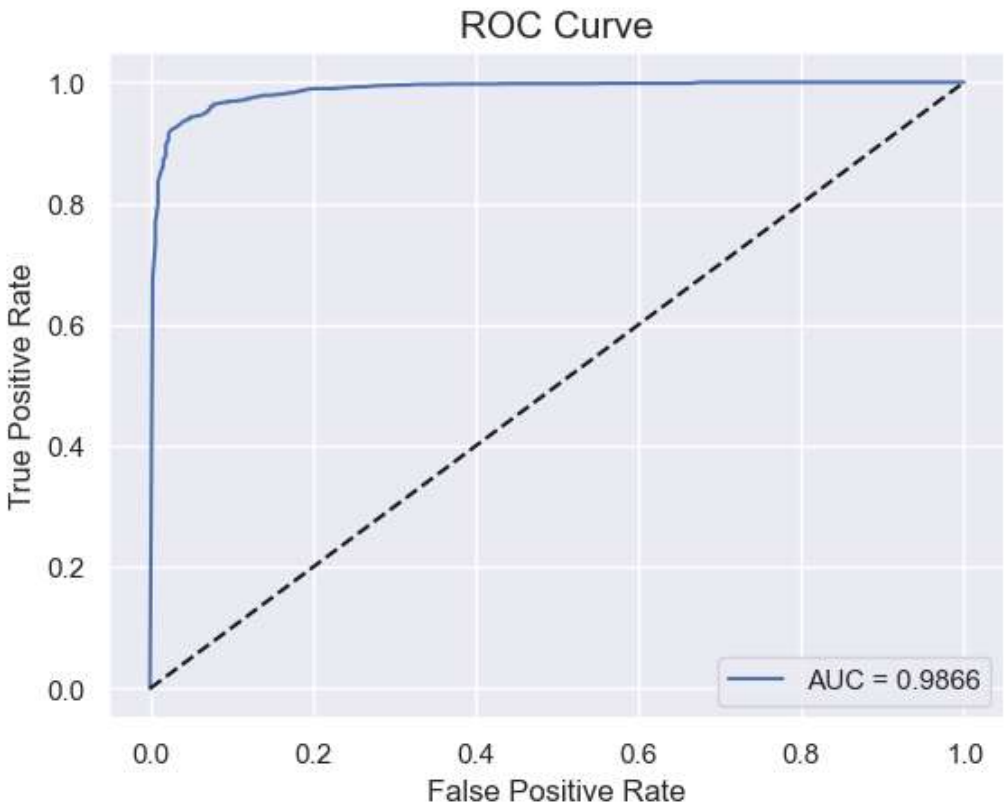
In [41]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[41]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9585389930898321')
```



In [42]:
```
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.D
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[42]: <matplotlib.legend.Legend at 0x258a655d460>



XGBoost Classifier

In [43]:
```
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': 5, 'n_estimators': 200}
```

In [44]:
```
from xgboost import XGBClassifier
xgb = XGBClassifier(gamma=0, learning_rate=0.1, max_depth=5, n_estimators=200)
xgb.fit(X_train, y_train)
```

Out[44]:
```
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=5, 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, ...)
```

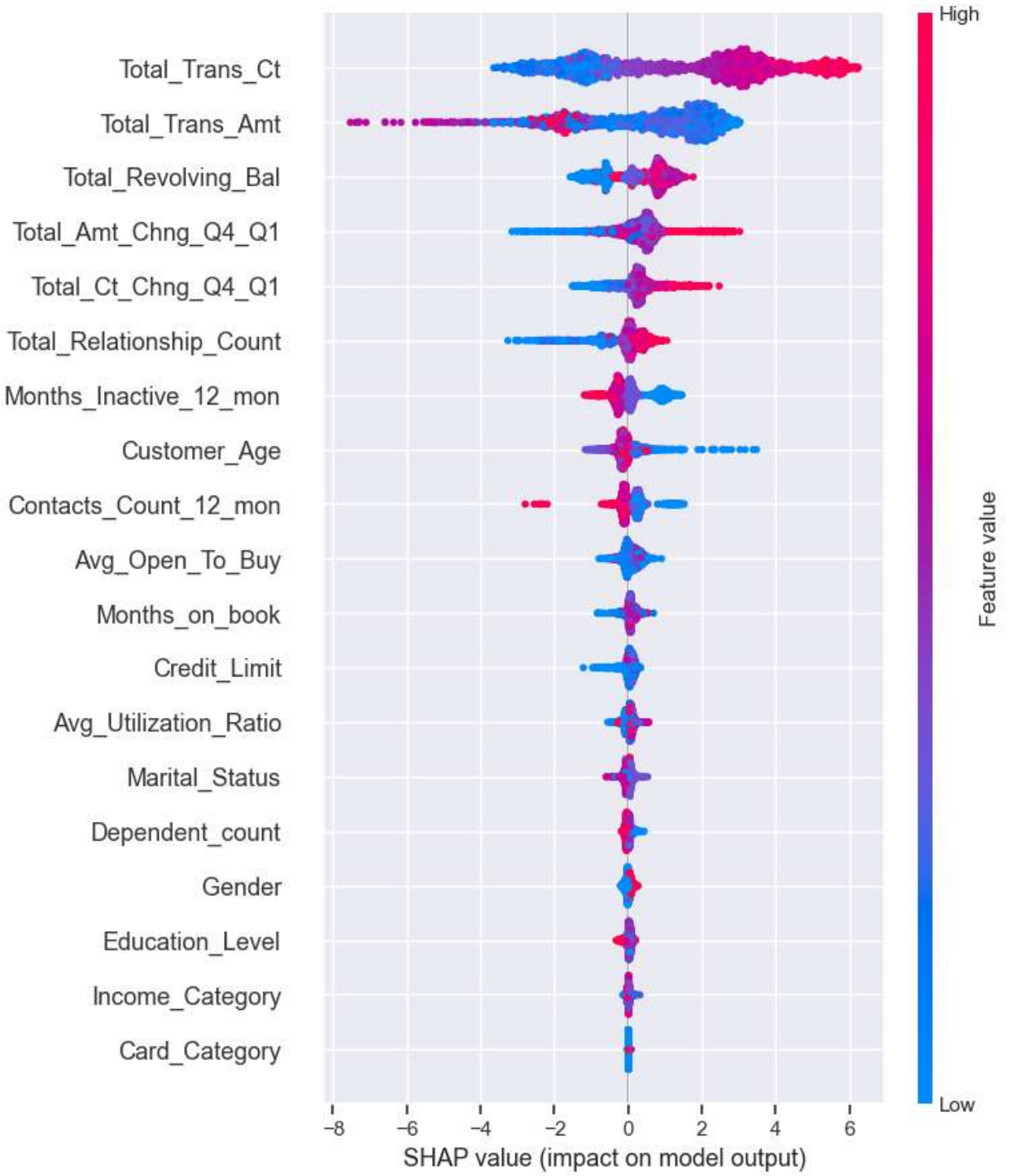
In [45]:
```
from sklearn.metrics import accuracy_score
y_pred = xgb.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 97.33 %
```

In [46]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccar
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)))
```

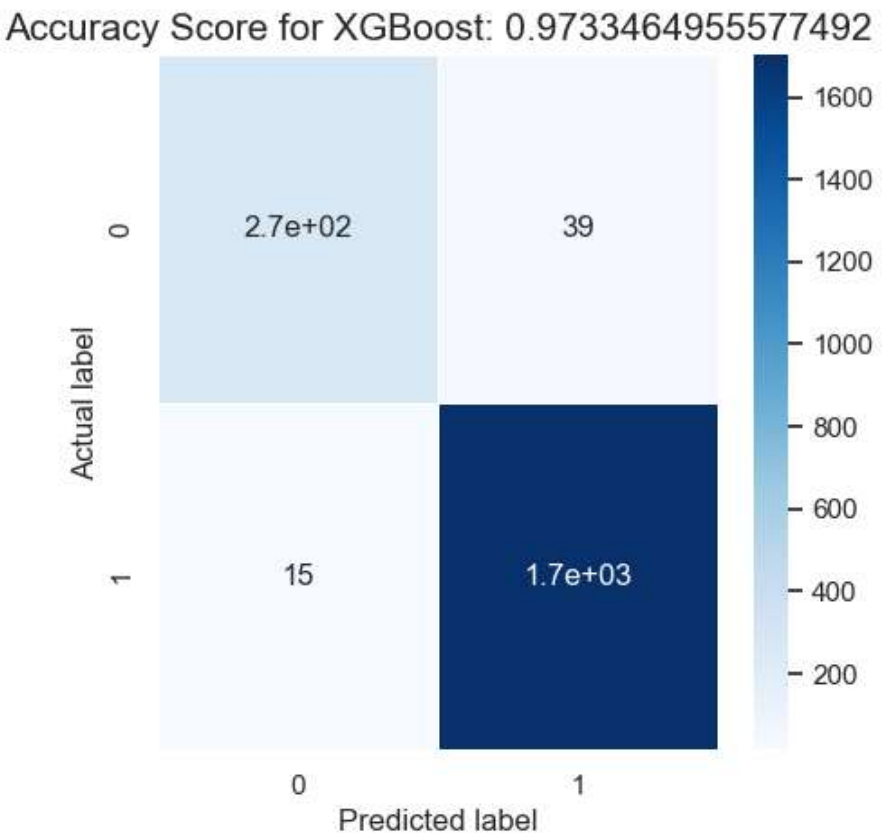
```
F-1 Score :  0.9733464955577492
Precision Score :  0.9733464955577492
Recall Score :  0.9733464955577492
Jaccard Score :  0.948076923076923
Log Loss :  0.920594822124855
```

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



In [48]:
```
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[48]:
```
Text(0.5, 1.0, 'Accuracy Score for XGBoost: 0.9733464955577492')
```



In [49]:
```
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.D
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[49]: <matplotlib.legend.Legend at 0x258a85c1a60>