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)
pd.set_option('display.max_columns', None)
```

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

Out[2]:

t_visit_time	days_since_last_login	avg_time_spent	avg_transaction_value	avg_frequency_login_days
16:08:02	17	300.63	53005.25	17.0
12:38:13	16	306.34	12838.38	10.0
22:53:21	14	516.16	21027.00	22.0
15:57:50	11	53.27	25239.56	6.0
15:46:44	20	113.13	24483.66	16.0

Data Preprocessing Part 1

In [3]:
```
# Drop identifier column
df.drop(columns = ['Unnamed: 0', 'security_no', 'referral_id'], inplace=True)
df.head()
```

Out[3]:

ue	avg_frequency_login_days	points_in_wallet	used_special_discount	offer_application_preference	pa
25	17.0	781.75	Yes	Yes	
38	10.0	NaN	Yes	No	
00	22.0	500.69	No	Yes	
56	6.0	567.66	No	Yes	
36	16.0	663.06	No	Yes	

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

Out[4]:
```
gender                        3
region_category               3
membership_category           6
joining_date               1096
joined_through_referral       3
preferred_offer_types         3
medium_of_operation           4
internet_option               3
```

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

Out[4]:
```
gender                             3
region_category                    3
membership_category                6
joining_date                    1096
joined_through_referral            3
preferred_offer_types              3
medium_of_operation                4
internet_option                    3
last_visit_time                30101
avg_frequency_login_days        1654
used_special_discount              2
offer_application_preference       2
past_complaint                     2
complaint_status                   5
feedback                           9
dtype: int64
```

In [5]:
```
# Only Extract Year on Joining Date
df['joining_date'] = df['joining_date'].str[:4].astype(int)
df['joining_date'].nunique()
```

Out[5]: 3

In [6]:
```
#  Remove Last visit time column
df.drop(columns = 'last_visit_time', inplace=True)
df.head()
```

Out[6]:

login	avg_time_spent	avg_transaction_value	avg_frequency_login_days	points_in_wallet	used_specia
17	300.63	53005.25	17.0	781.75	
16	306.34	12838.38	10.0	NaN	
14	516.16	21027.00	22.0	500.69	
11	53.27	25239.56	6.0	567.66	
20	113.13	24483.66	16.0	663.06	

In [8]:
```
# Replace 'Error' values with 0
df['avg_frequency_login_days'] = df['avg_frequency_login_days'].replace('Error',

# Convert the column to integer data type
df['avg_frequency_login_days'] = df['avg_frequency_login_days'].astype(float)
```

In [9]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[9]:
```
gender                       3
region_category              3
membership_category          6
joined_through_referral      3
preferred_offer_types        3
medium_of_operation          4
internet_option              3
used_special_discount        2
```

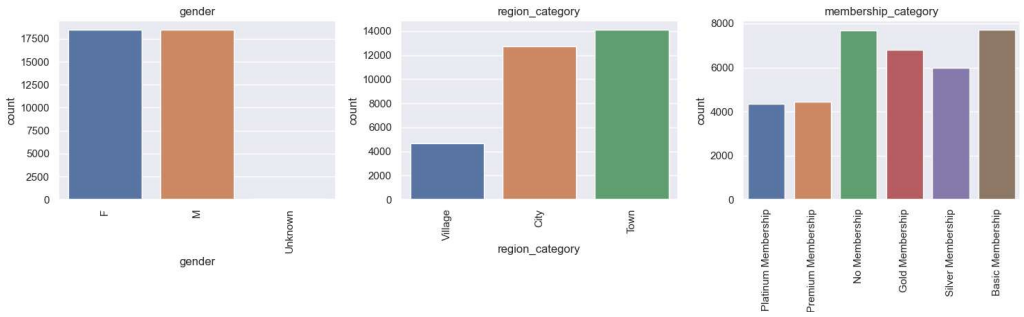
In [9]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

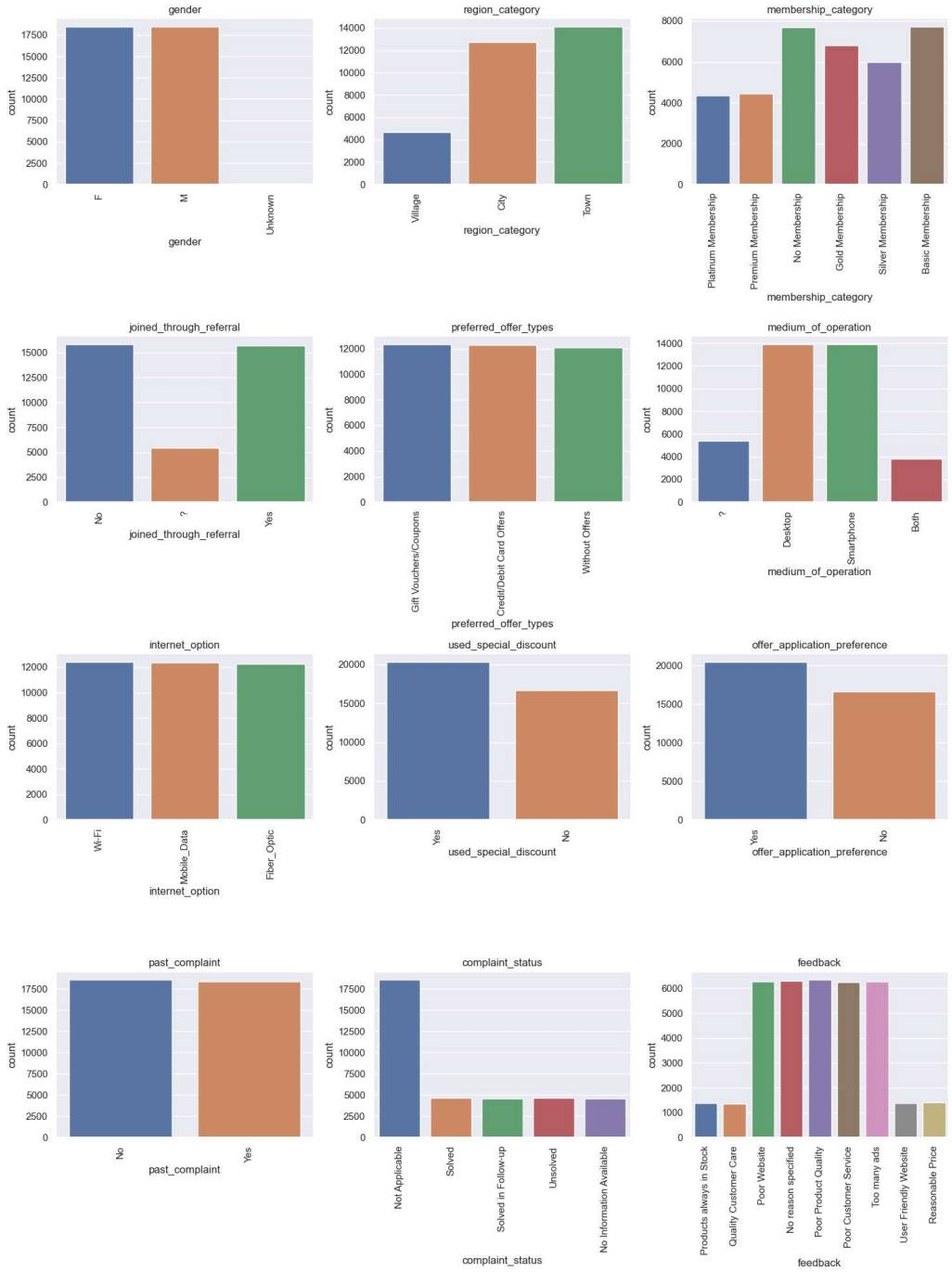
Out[9]:
```
gender                          3
region_category                 3
membership_category             6
joined_through_referral         3
preferred_offer_types           3
medium_of_operation             4
internet_option                 3
used_special_discount           2
offer_application_preference    2
past_complaint                  2
complaint_status                5
feedback                        9
dtype: int64
```

Exploratory Data Analysis

In [10]:
```
# Get the names of all columns with data type 'object' (categorical columns) excl
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 6 values of each categorical variable using Seab
for i, var in enumerate(cat_vars):
```

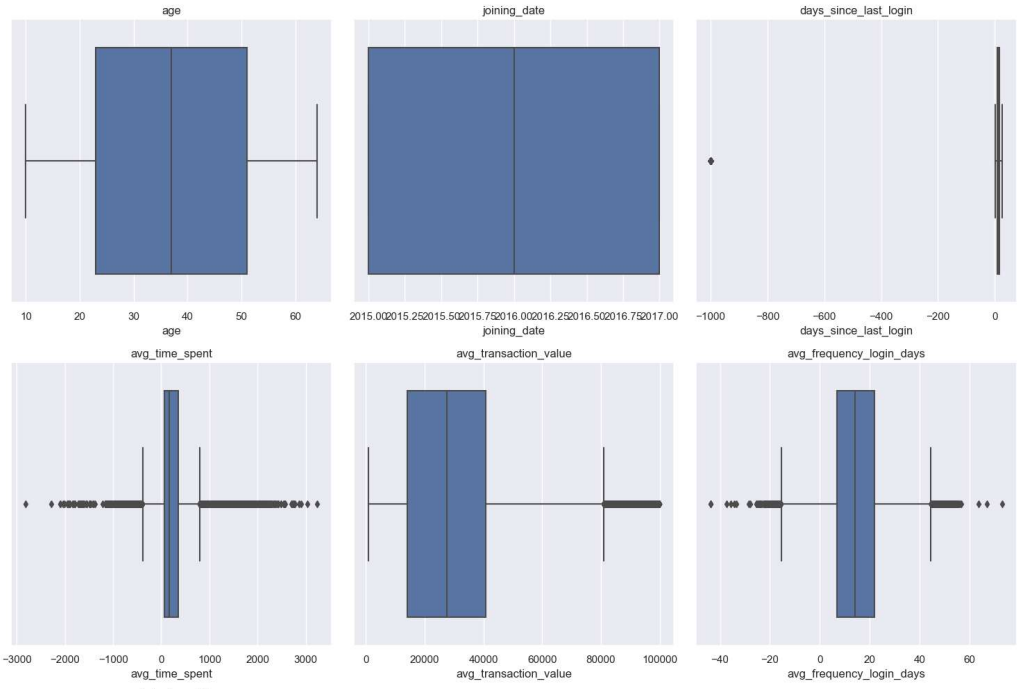
In [10]:
```
# Get the names of all columns with data type 'object' (categorical columns) excl
cat_vars = df.select_dtypes(include='object').columns.tolist()

# 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 6 values of each categorical variable using Seab
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().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()
```





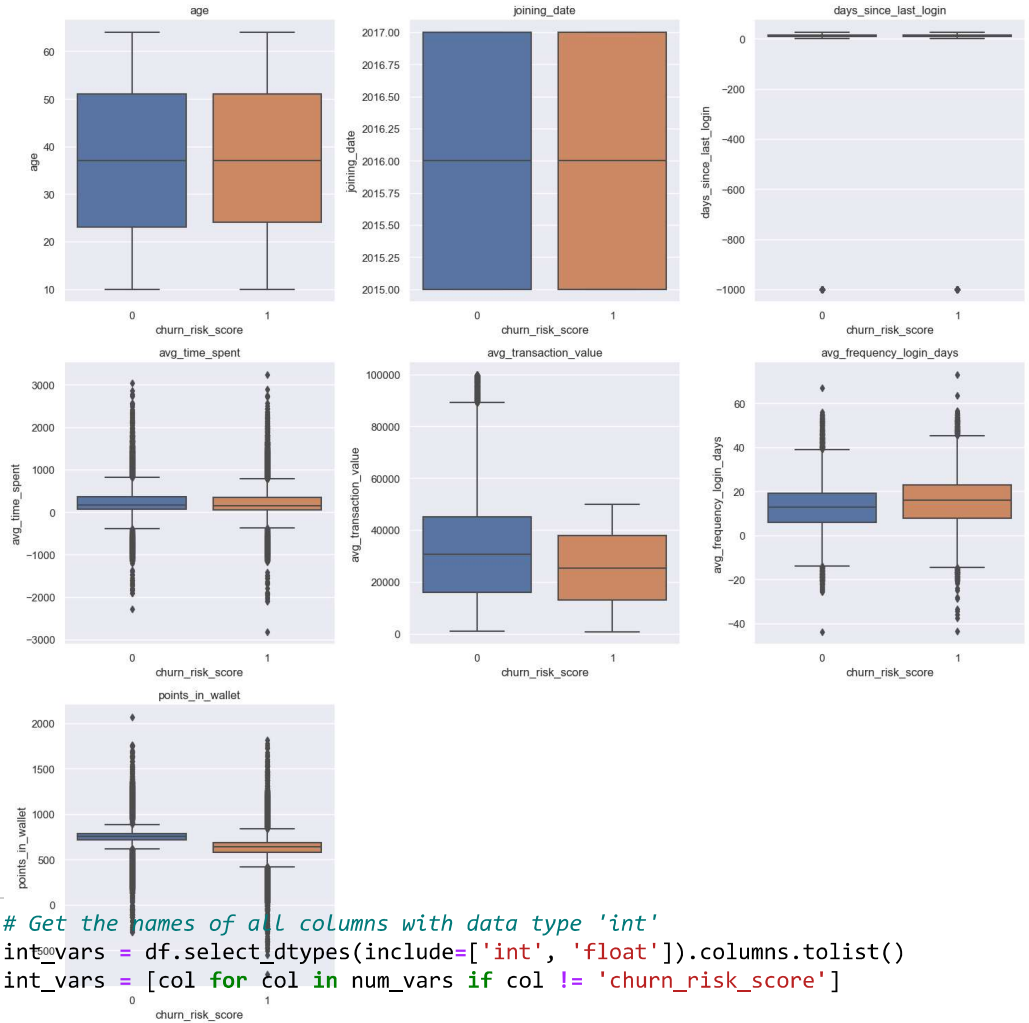
In [12]:
```
# Get the names of all columns with data type 'int' or 'float', excluding 'churn_
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
num_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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
```

In [12]:
```
# Get the names of all columns with data type 'int' or 'float', excluding 'churn_
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
num_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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()
```



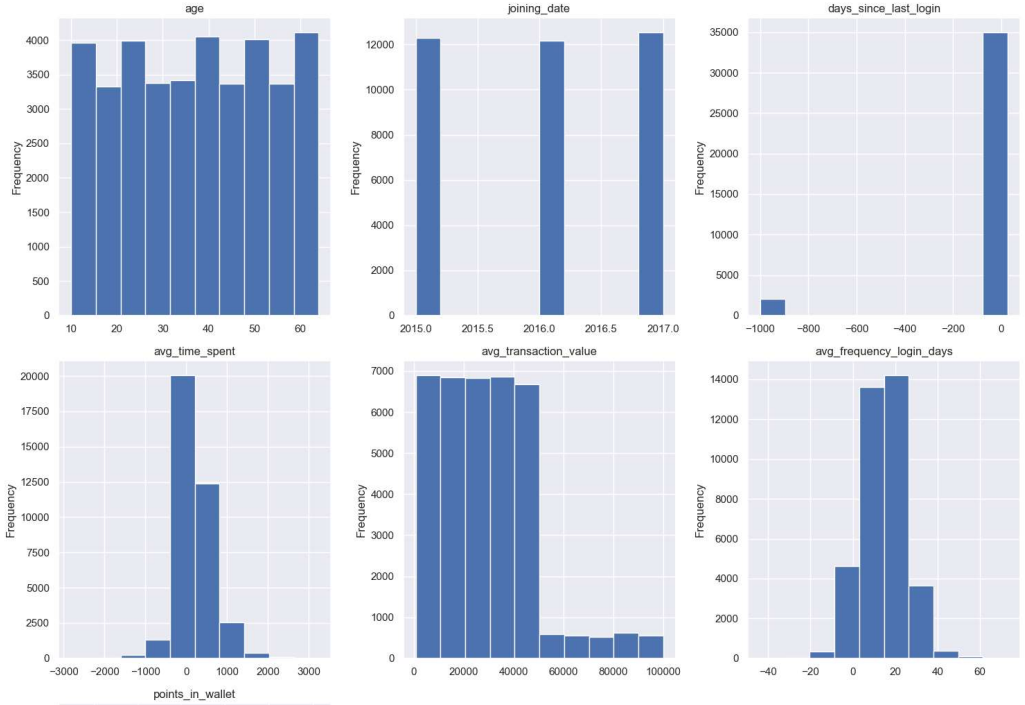
In [14]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
int_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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 subp
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'
```

In [14]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
int_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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 subp
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='churn_risk_score', 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()
```



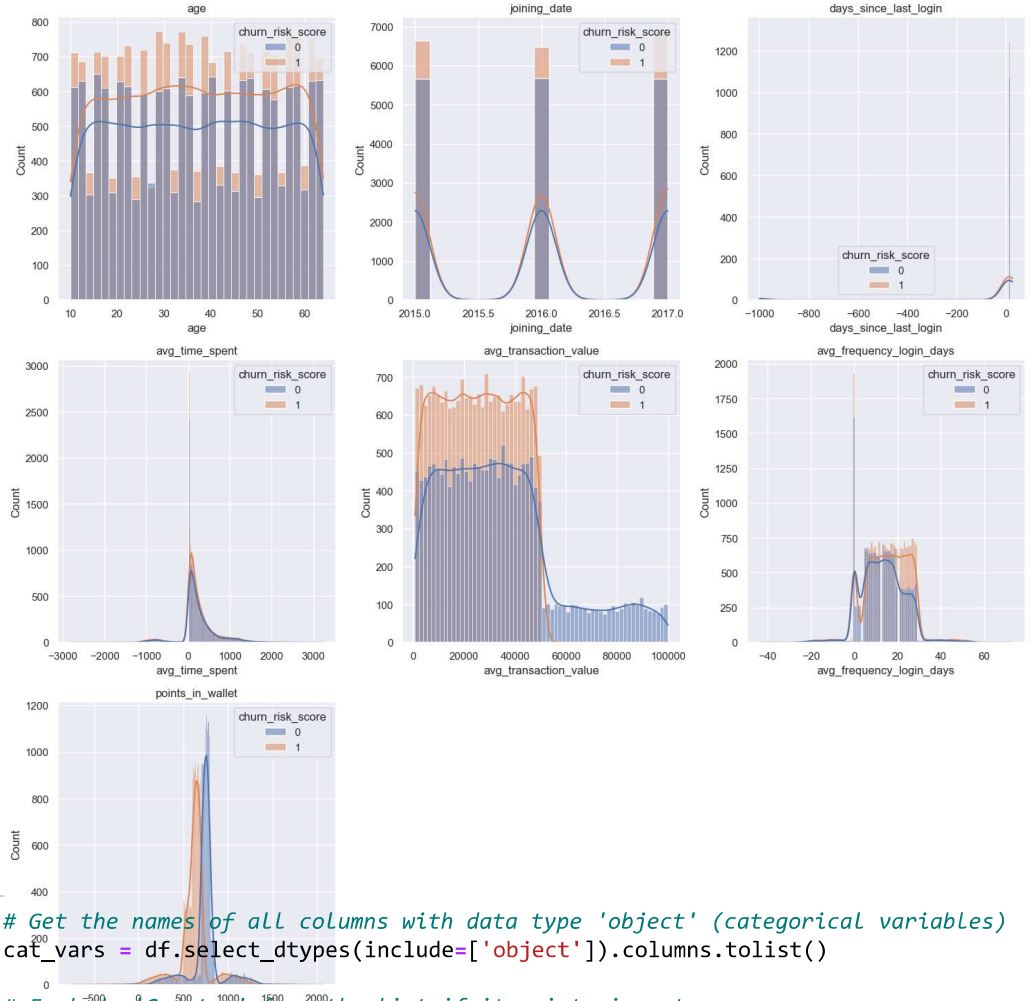
In [15]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
int_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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 subp
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable
```

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

# 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 subp
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()
```



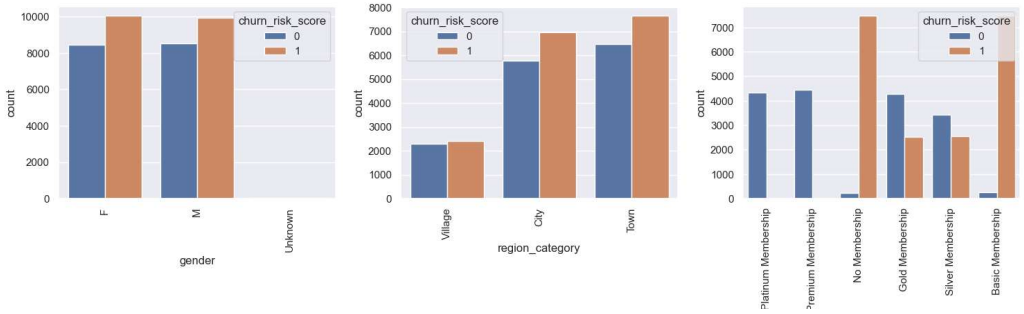
In [16]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
int_vars = [col for col in num_vars if col != 'churn_risk_score']

# 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 subp
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'
```

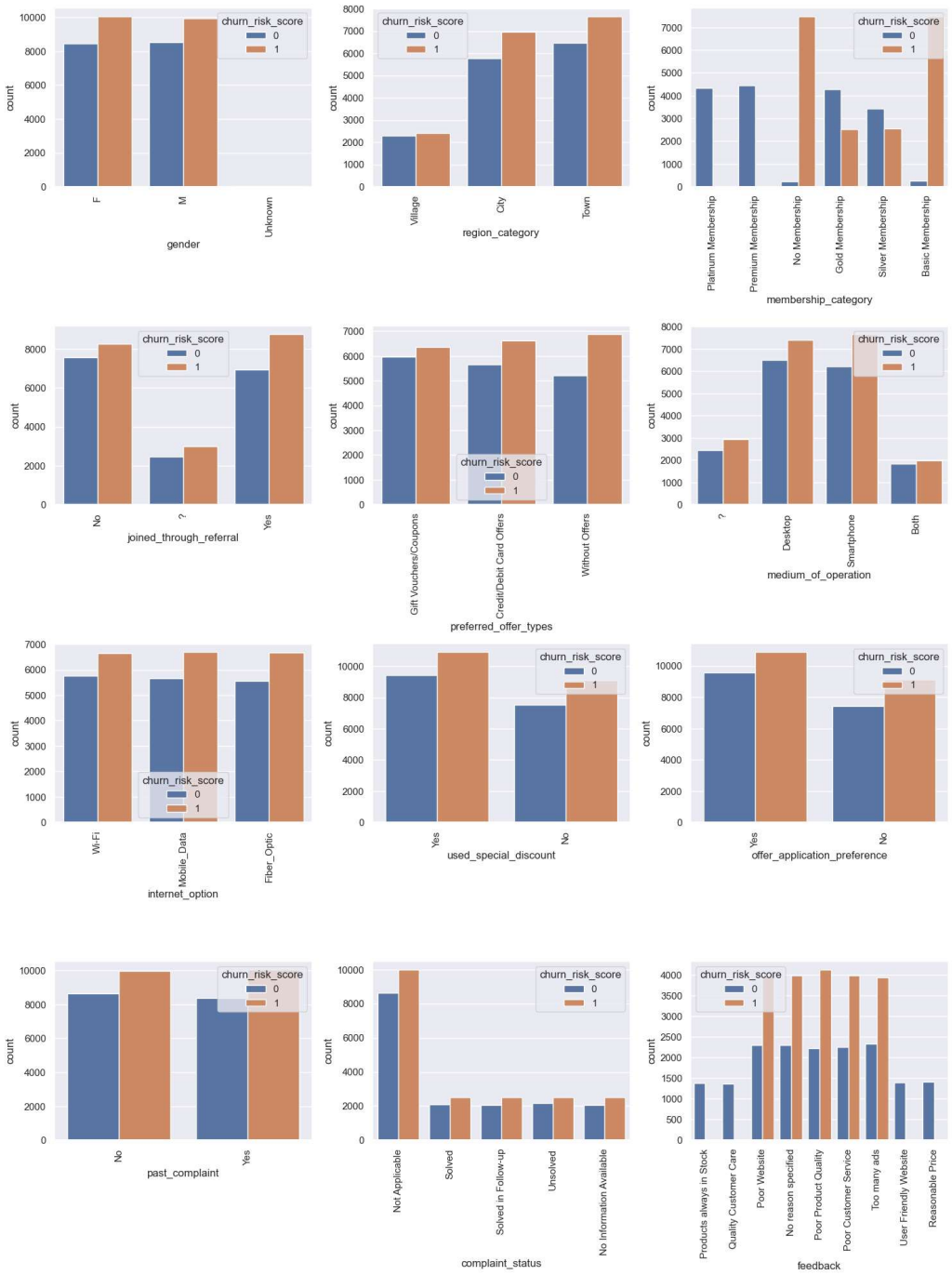
In [16]:
```
# Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()
int_vars = [col for col in num_vars if col != 'churn_risk_score']
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subp
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='churn_risk_score', 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()
```



In [17]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()
# Exclude 'Country' from the list if it exists in cat_vars
if 'churn_risk_score' in cat_vars:
    cat_vars.remove('churn_risk_score')

# Create a figure with subplots, but only include the required number of subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3  # To make sure there are enough rows for the subp
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
```

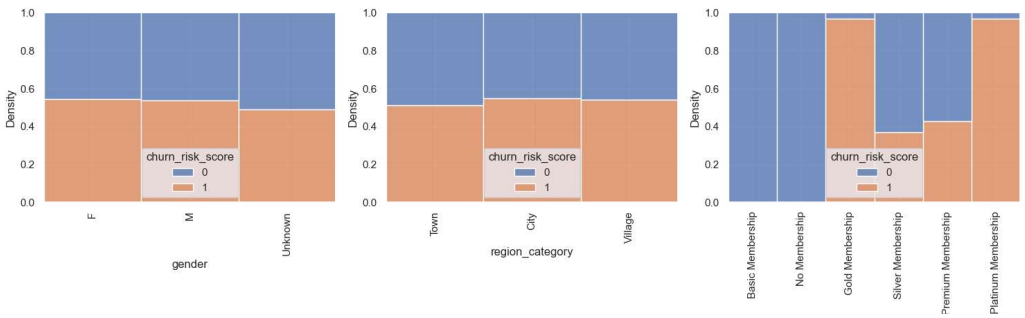
In [17]:
```
# Get the names of all columns with data type 'object' (categorical variables)
cat_vars = df.select_dtypes(include=['object']).columns.tolist()

# Exclude 'Country' from the list if it exists in cat_vars
if 'churn_risk_score' in cat_vars:
    cat_vars.remove('churn_risk_score')

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

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    filtered_df = df[df[var].notnull()]  # Exclude rows with NaN values in the va
    sns.countplot(x=var, hue='churn_risk_score', data=filtered_df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

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

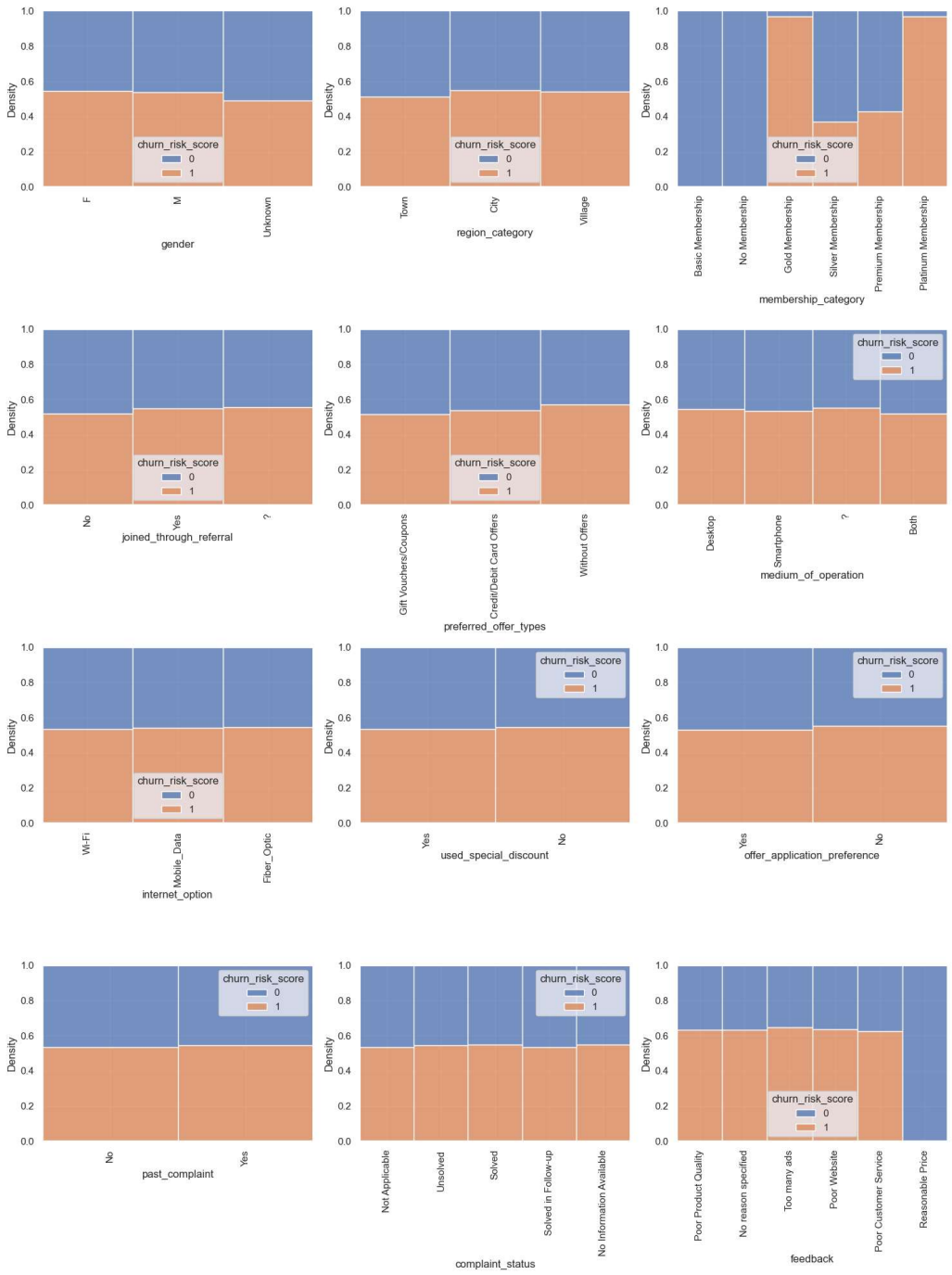




In [18]:
```
import warnings

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

# Exclude 'Attrition' from the list if it exists in cat_vars
if 'churn_risk_score' in cat_vars:
    cat_vars.remove('churn_risk_score')

# Create a figure with subplots, but only include the required number of subplots
num_cols = len(cat_vars)
```

In [18]:
```
import warnings

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

# Exclude 'Attrition' from the list if it exists in cat_vars
if 'churn_risk_score' in cat_vars:
    cat_vars.remove('churn_risk_score')

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

# Create a count plot for the top 6 values of each categorical variable as a dens
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(6).index
    filtered_df = df[df[var].isin(top_values)]

    # Set x-tick positions explicitly
    tick_positions = range(len(top_values))
    axs[i].set_xticks(tick_positions)
    axs[i].set_xticklabels(top_values, rotation=90)  # Set x-tick labels

    sns.histplot(x=var, hue='churn_risk_score', data=filtered_df, ax=axs[i], mult
    axs[i].set_xlabel(var)

# Remove any remaining blank subplots
for i in range(num_cols, len(axs)):
    fig.delaxes(axs[i])

# Adjust spacing between subplots
fig.tight_layout()

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

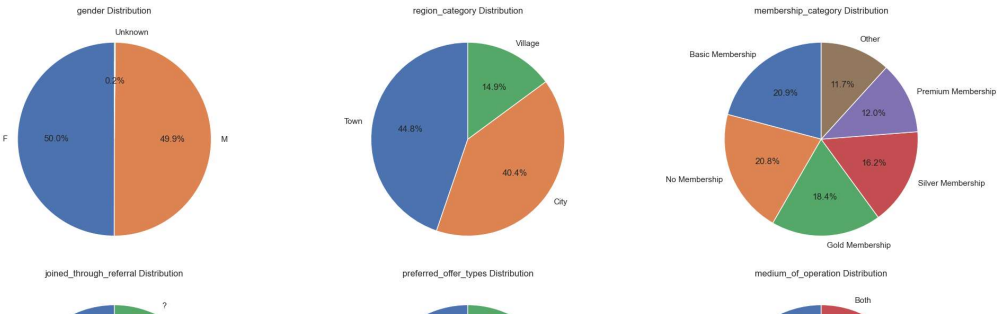


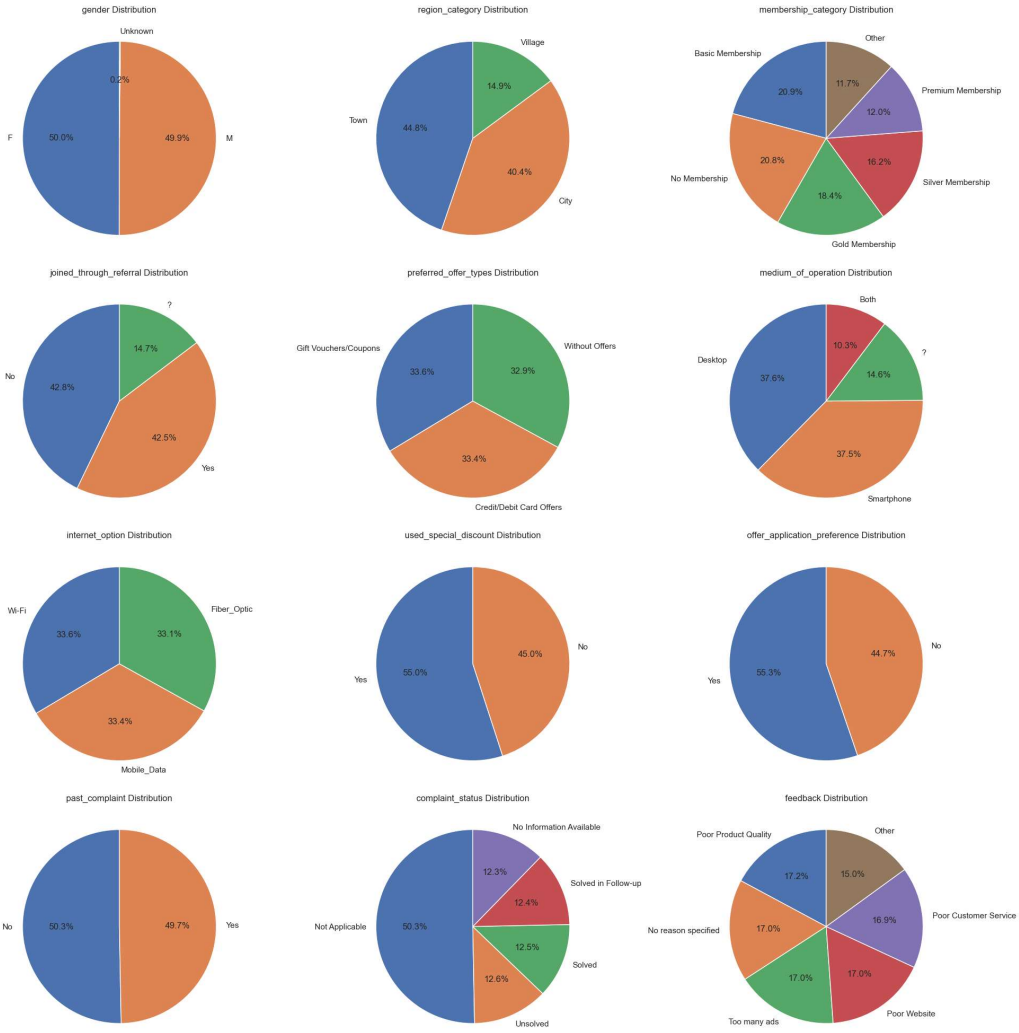


In [19]:
```
# Specify the maximum number of categories to show individually
max_categories = 5

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

# 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=(20, 5*num_rows))
```

In [19]:
```
# 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 != 'y' and df[col].dtype == 'object

# 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=(20, 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
            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%%', starta
        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()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_6924\943223290.py:25: FutureWarni
ng: The series.append method is deprecated and will be removed from pandas in a
future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
C:\Users\Michael\AppData\Local\Temp\ipykernel_6924\943223290.py:25: FutureWarni
ng: The series.append method is deprecated and will be removed from pandas in a
future version. Use pandas.concat instead.
  cat_counts = cat_counts_top.append(cat_counts_other)
```





Data Preprocessing Part 2

In [20]:
```
# 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[20]:
```
region_category          14.673443
points_in_wallet          9.307418
preferred_offer_types     0.778547
dtype: float64
```

In [21]:
```
# Drop null value from region_category and preferred_offer_types columns
df.dropna(subset=['region_category', 'preferred_offer_types'], inplace=True)
# Fill null value from points_in_wallet with median
df['points_in_wallet'].fillna(df['points_in_wallet'].median(), inplace=True)
```

In [22]:
```
df.shape
```

Out[22]:
```
(31325, 20)
```

Label Encoding for Object Datatypes

In [23]:
```
# 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()}")
```

Out[23]:
```
gender: ['F' 'M' 'Unknown']
region_category: ['Village' 'City' 'Town']
membership_category: ['Platinum Membership' 'Premium Membership' 'No Membershi
p'
 'Gold Membership' 'Silver Membership' 'Basic Membership']
```

In [22]:
```
df.shape
```

Out[22]:
```
(31325, 20)
```

In [23]:
```
# 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()}")
```

Label Encoding for Object Datatypes

```
gender: ['F' 'M' 'Unknown']
region_category: ['Village' 'City' 'Town']
membership_category: ['Platinum Membership' 'Premium Membership' 'No Membershi
p'
 'Gold Membership' 'Silver Membership' 'Basic Membership']
joined_through_referral: ['No' '?' 'Yes']
preferred_offer_types: ['Gift Vouchers/Coupons' 'Credit/Debit Card Offers' 'Wit
hout Offers']
medium_of_operation: ['?' 'Desktop' 'Smartphone' 'Both']
internet_option: ['Wi-Fi' 'Mobile_Data' 'Fiber_Optic']
used_special_discount: ['Yes' 'No']
offer_application_preference: ['Yes' 'No']
past_complaint: ['No' 'Yes']
complaint_status: ['Not Applicable' 'Solved' 'Solved in Follow-up' 'Unsolved'
 'No Information Available']
feedback: ['Products always in Stock' 'Quality Customer Care' 'Poor Website'
 'No reason specified' 'Poor Customer Service' 'Poor Product Quality'
 'Too many ads' 'User Friendly Website' 'Reasonable Price']
```

In [24]:
```
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()}")
```

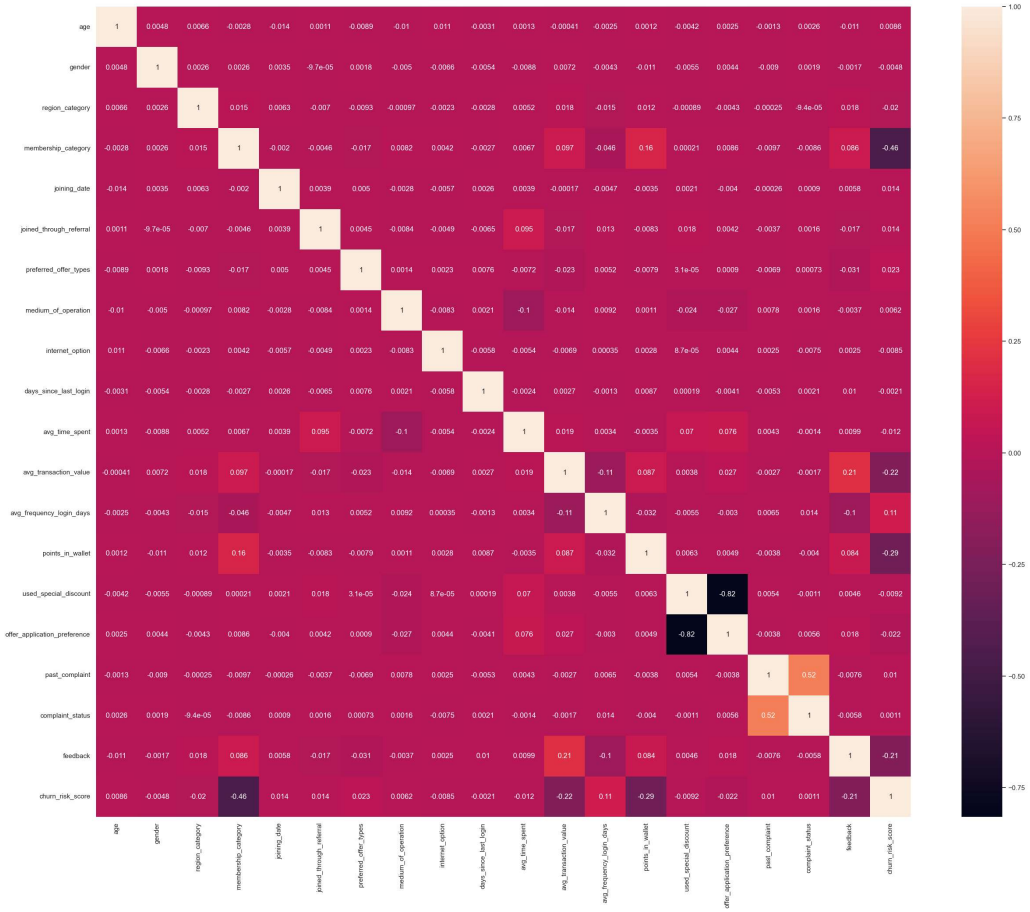
```
gender: [0 1 2]
region_category: [2 0 1]
membership_category: [3 4 2 1 5 0]
joined_through_referral: [1 0 2]
preferred_offer_types: [1 0 2]
medium_of_operation: [0 2 3 1]
internet_option: [2 1 0]
used_special_discount: [1 0]
offer_application_preference: [1 0]
past_complaint: [0 1]
complaint_status: [1 2 3 4 0]
feedback: [4 5 3 0 1 2 7 8 6]
```

In [26]:
```
# Correlation Heatmap
plt.figure(figsize=(30, 24))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

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

In [26]:
```
# Correlation Heatmap
plt.figure(figsize=(30, 24))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[26]: <AxesSubplot:>



Train Test Split

In [27]:
```
X = df.drop('churn_risk_score', axis=1)
y = df['churn_risk_score']
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_sta
```

Remove Outlier from Train Data using Z-Score

In [28]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['days_since_last_login', 'avg_time_spent', 'avg_transaction_v
                    'avg_frequency_login_days', 'points_in_wallet']

# 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
```

In [28]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['days_since_last_login', 'avg_time_spent', 'avg_transaction_v
                    'avg_frequency_login_days', 'points_in_wallet']

# 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 [29]:
```
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': 7, 'min_samples_leaf': 1, 'min_samples_split': 3, 'random_state':
42}
```

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=42, max_depth=7, min_samples_leaf=1,
dtree.fit(X_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=7,
                       min_samples_split=3, random_state=42)
```

In [31]:
```
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 : 93.38 %
```

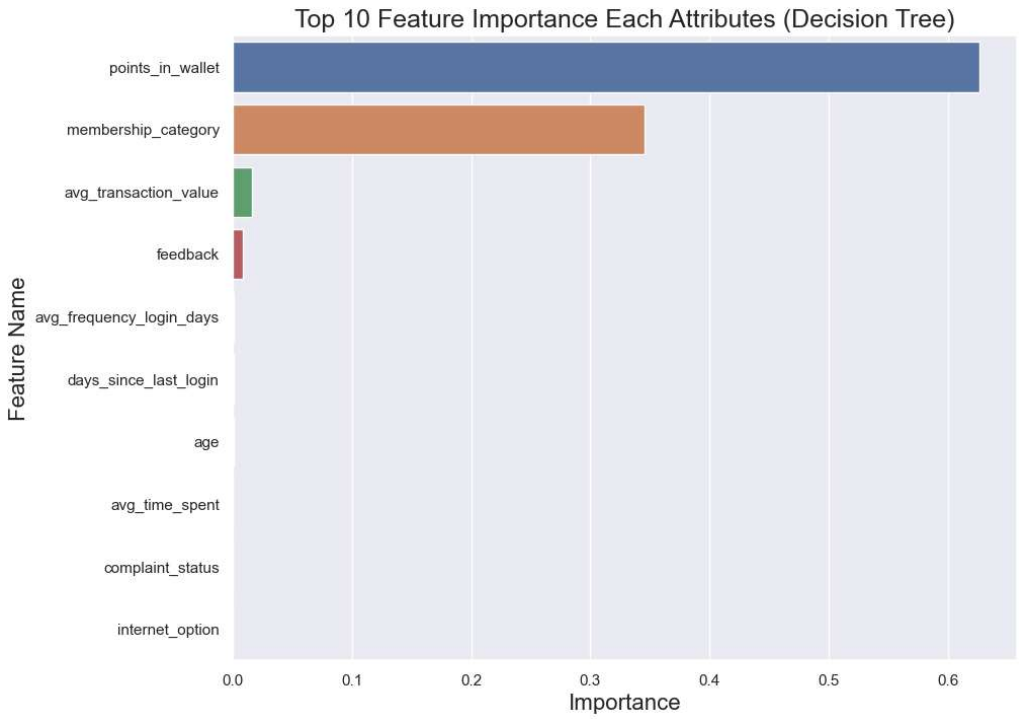
In [32]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_sc
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.9337589784517158
Precision Score :  0.9337589784517158
Recall Score :  0.9337589784517158
Jaccard Score :  0.875748502994012
```

In [32]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_sc
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)))
```

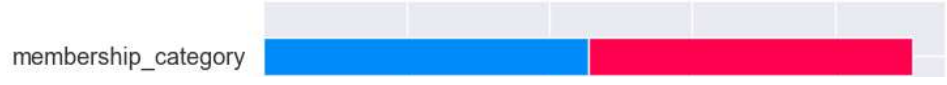
```
F-1 Score :  0.9337589784517158
Precision Score :  0.9337589784517158
Recall Score :  0.9337589784517158
Jaccard Score :  0.875748502994012
Log Loss :  2.287902848189152
```

In [33]:
```
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=1
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



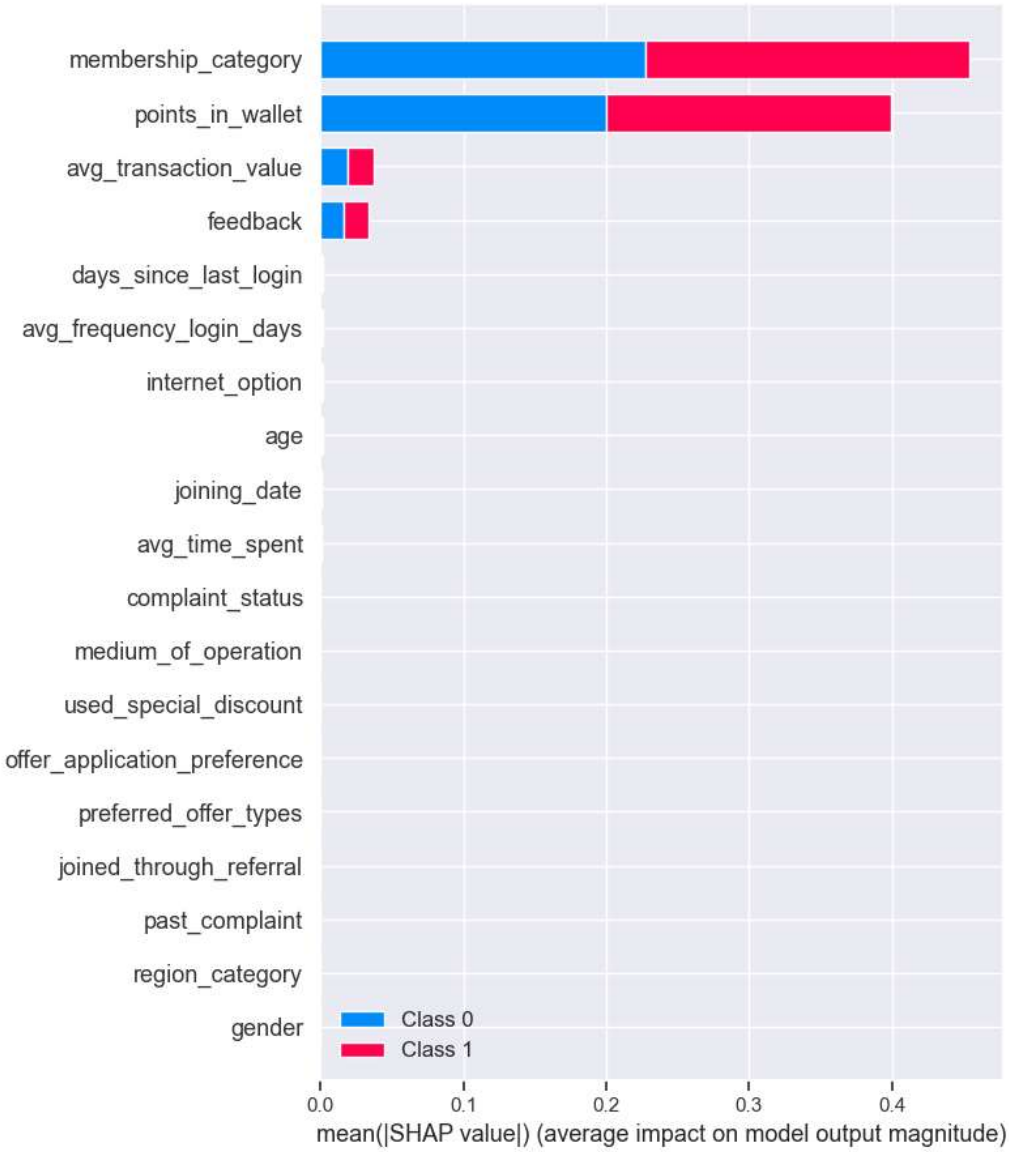
In [34]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to for
ce console mode (e.g. in jupyter console)
```

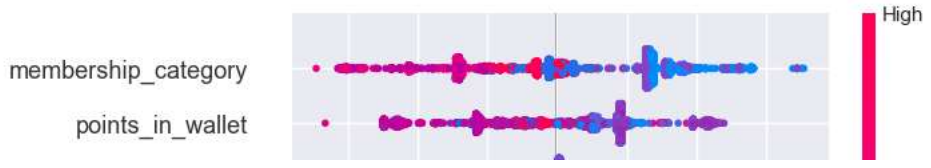


In [34]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

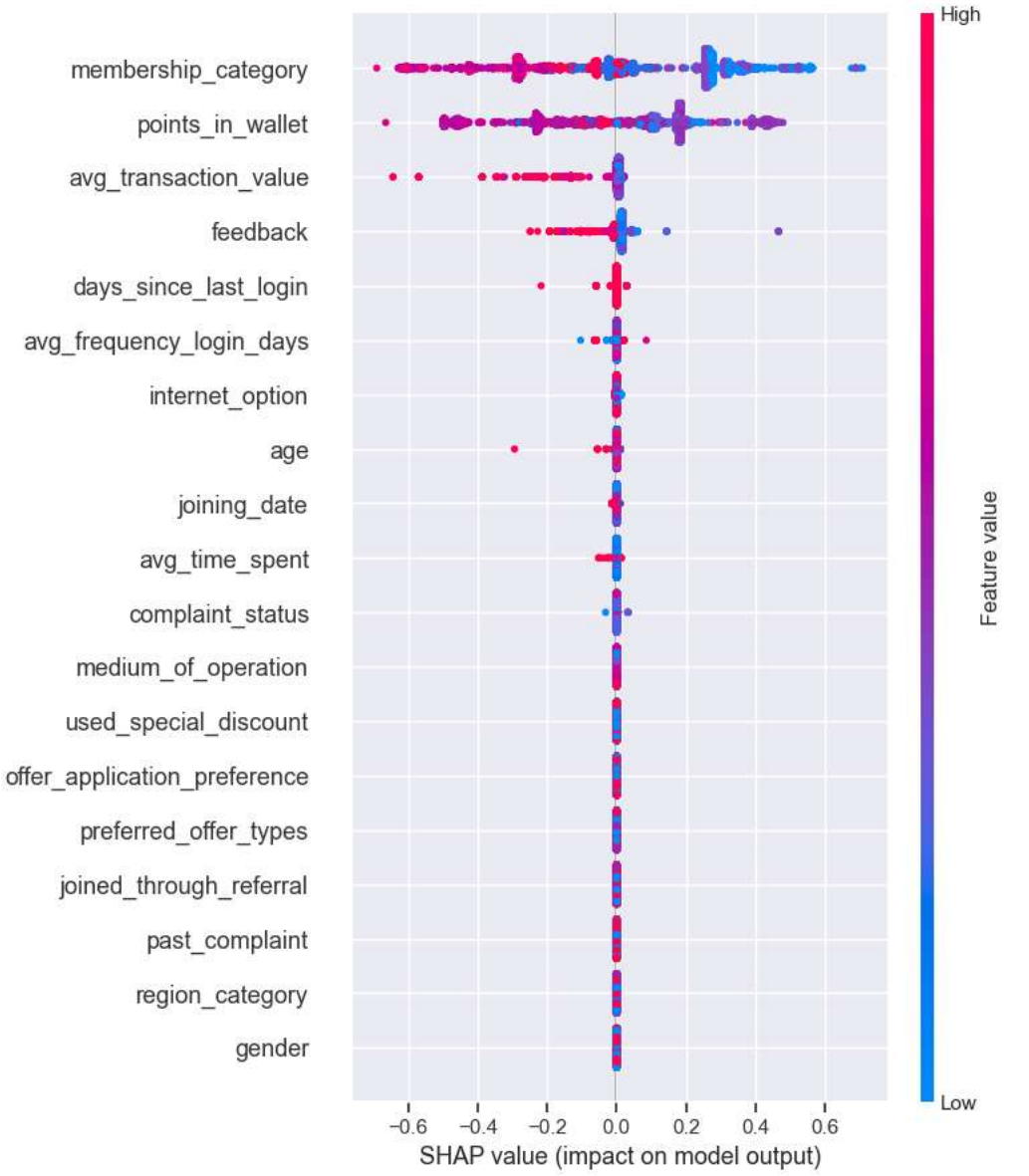
```
Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to for
ce console mode (e.g. in jupyter console)
```



In [35]:
```
# 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 [35]:
```
# 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)
```

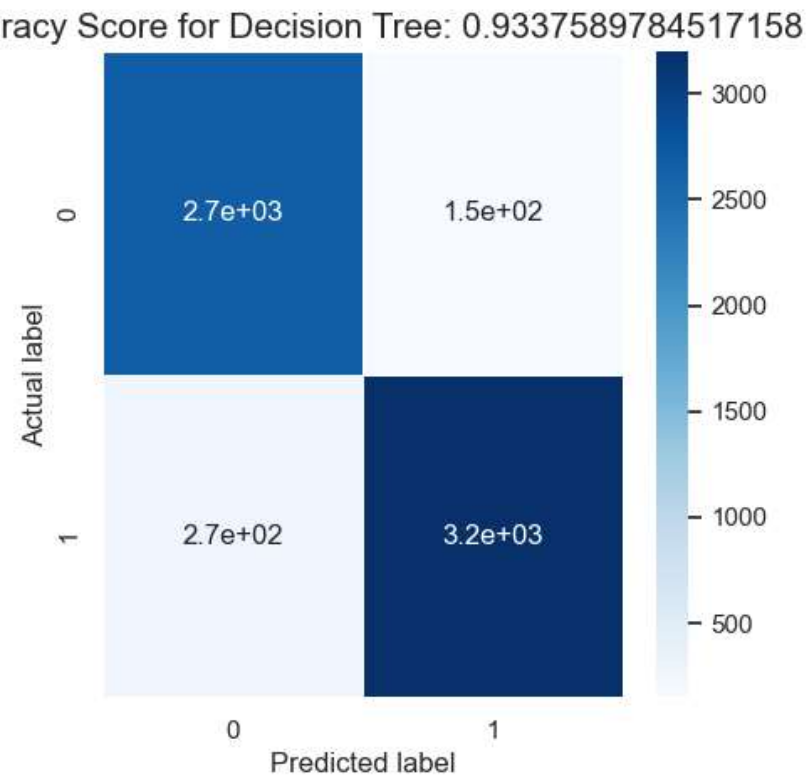


In [36]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_t
plt.title(all_sample_title, size = 15)
```

Out[36]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9337589784517158')

In [36]:
```
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_t
plt.title(all_sample_title, size = 15)
```

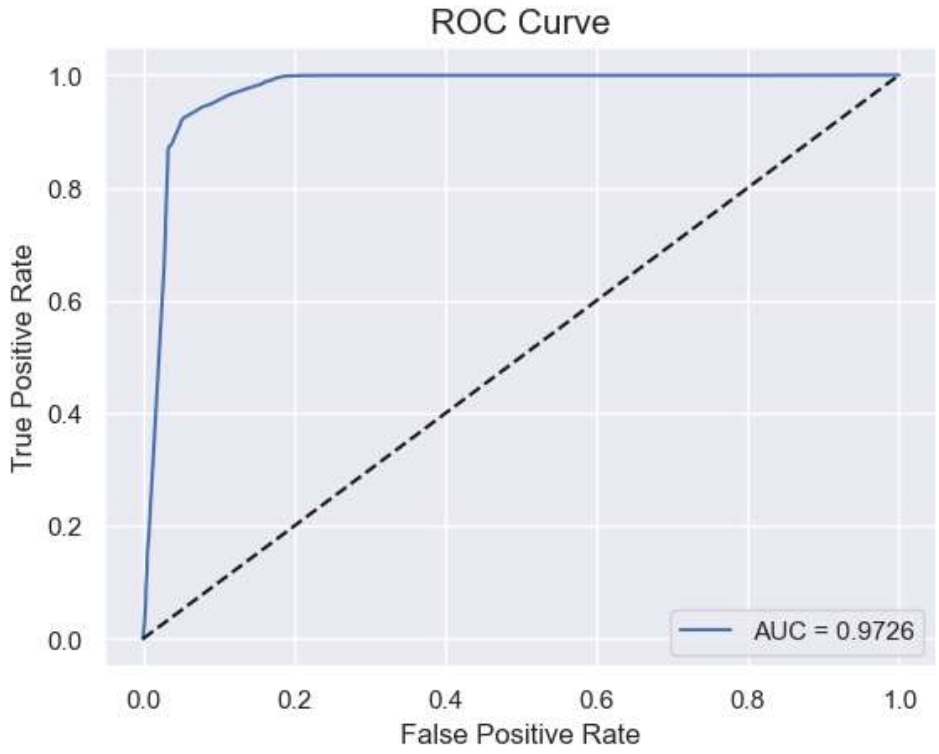
Out[36]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9337589784517158')
```



In [37]:
```
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_actua
df_actual_predicted.index = y_test.index

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

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
```

In [37]:
```
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_actua
df_actual_predicted.index = y_test.index

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

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[37]:
```
<matplotlib.legend.Legend at 0x1dd0aa13520>
```



Random Forest Classifier

In [38]:
```
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
```

In [38]:
```
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': 10, 'max_features': None, 'n_estimators': 100, 'random_state': 4
2}
```

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_depth=10, max_features=None, n_
rfc.fit(X_train, y_train)
```

Out[39]:
```
RandomForestClassifier(class_weight='balanced', max_depth=10, max_features=Non
e,
                       random_state=42)
```

In [40]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

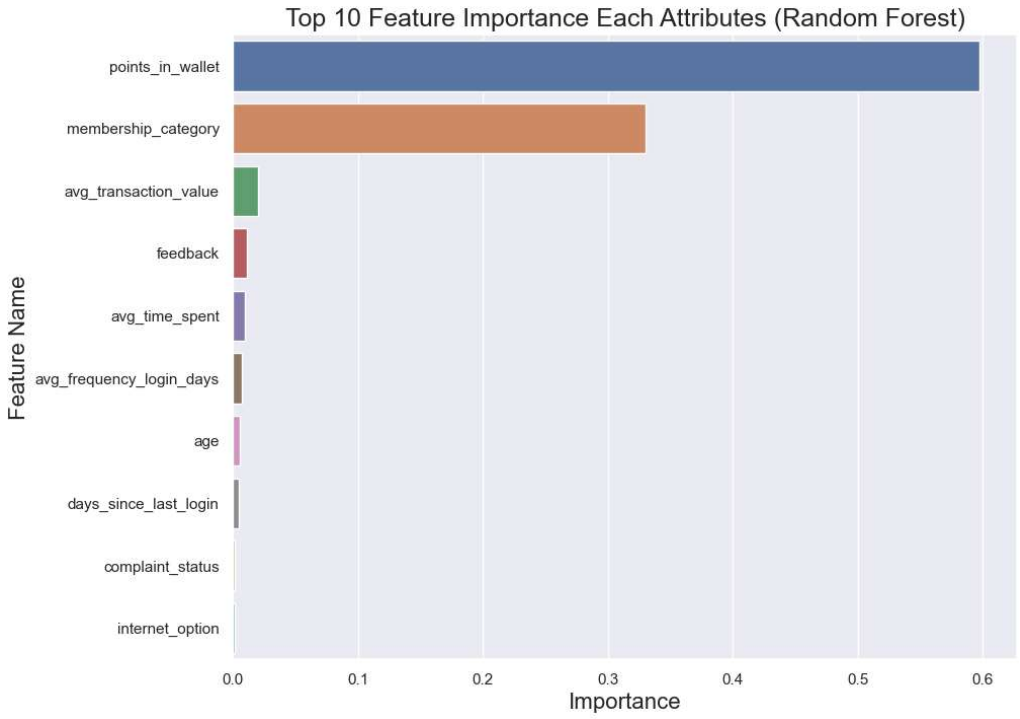
```
Accuracy Score : 93.2 %
```

In [41]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_sc
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 :  0.9320031923383879
Precision Score :  0.9320031923383879
Recall Score :  0.9320031923383879
Jaccard Score :  0.872664773576446
Log Loss :  2.3485491257744773
```

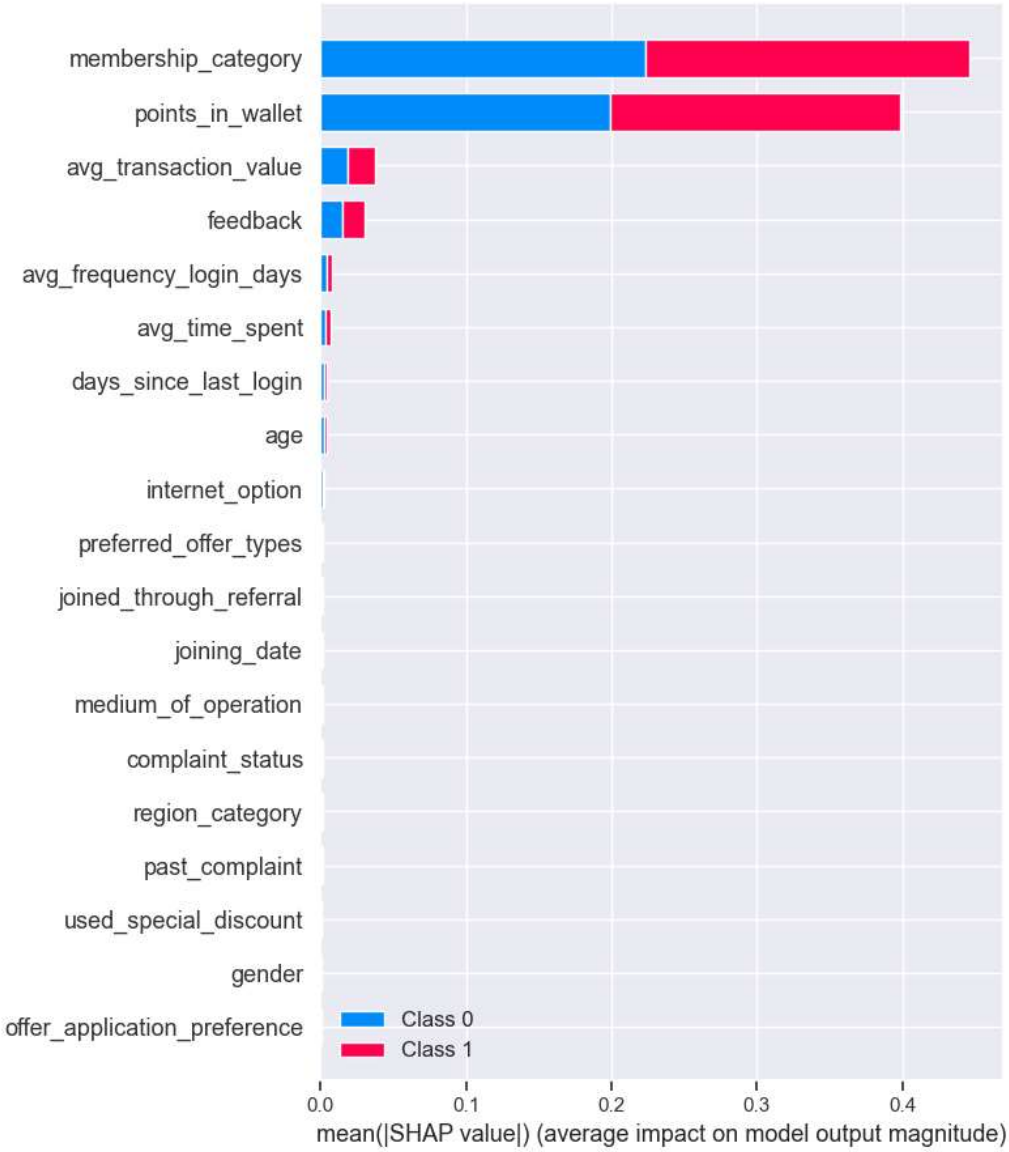
In [42]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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=1
plt.xlabel('Importance', fontsize=16)
```

In [42]:
```
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=1
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```

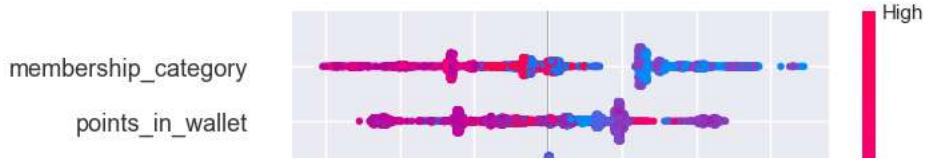


In [43]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

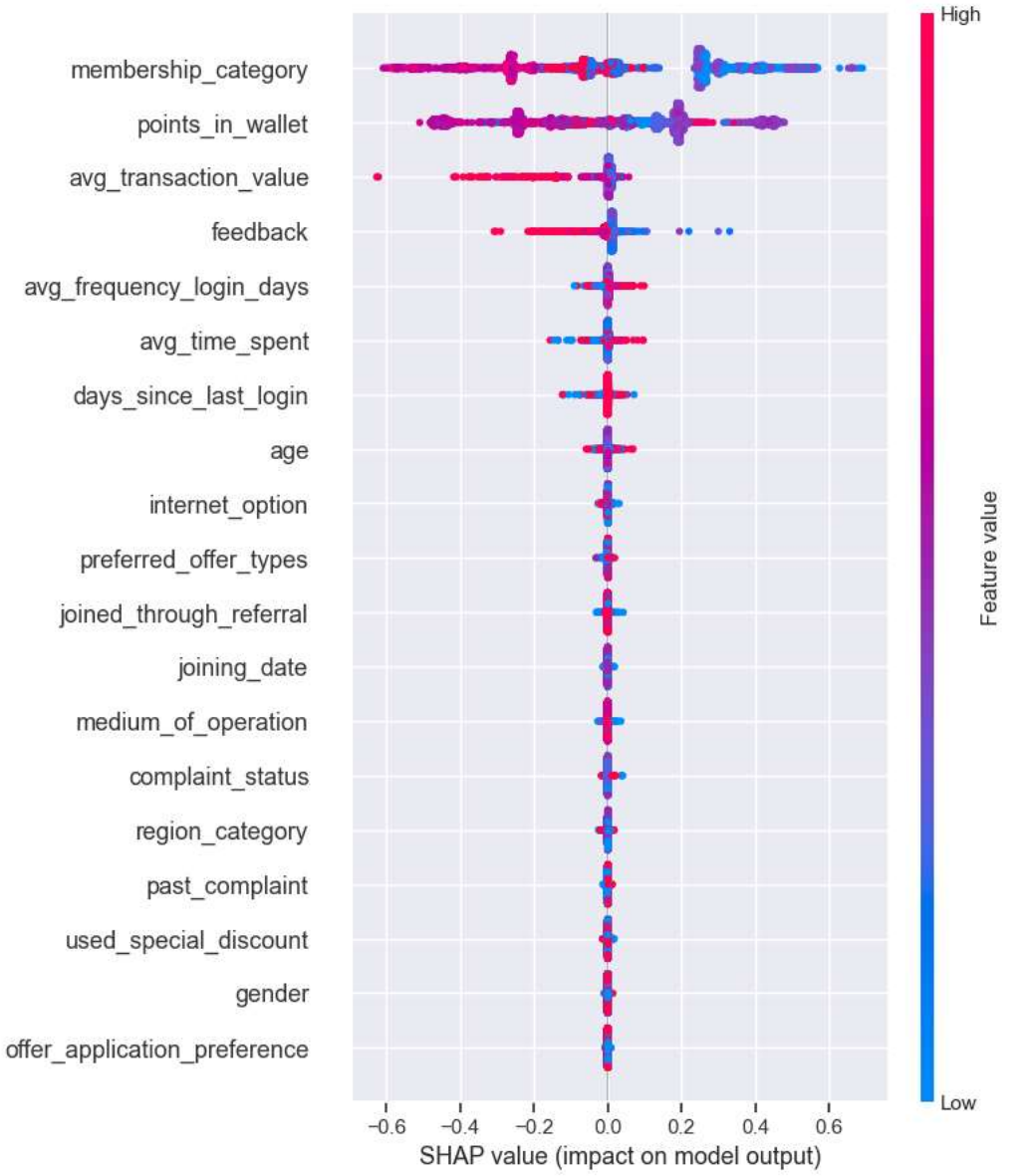
In [43]:
```
import shap
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [44]:
```
# 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 [44]:
```
# 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)
```

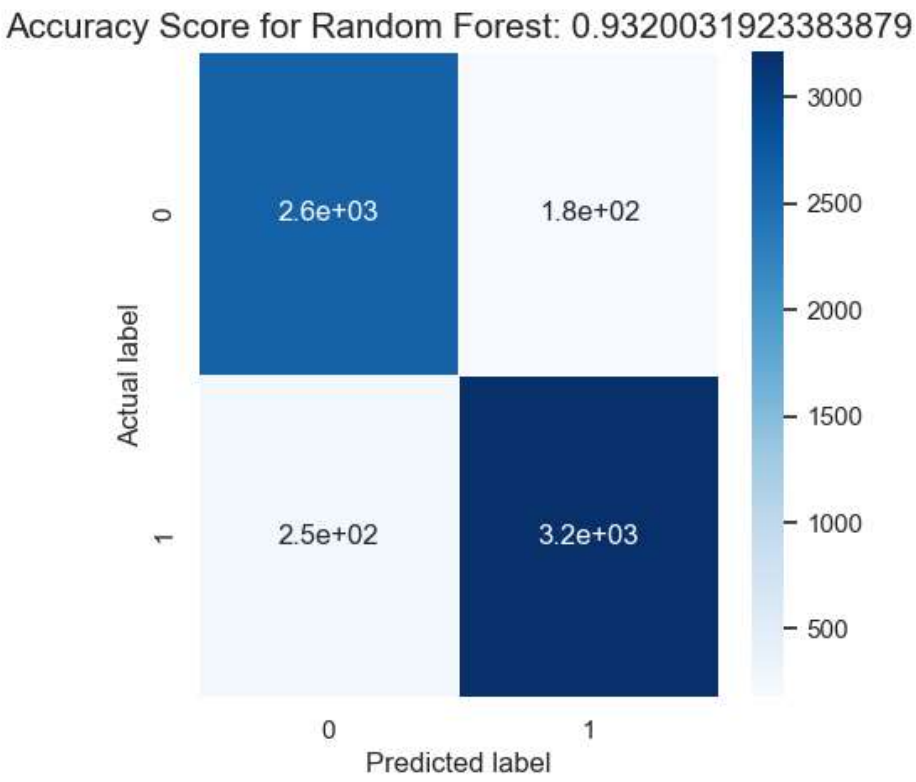


In [45]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Random Forest: {0}'.format(rfc.score(X_tes
plt.title(all_sample_title, size = 15)
```

Out[45]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9320031923383879')
```

In [45]:
```
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_tes
plt.title(all_sample_title, size = 15)
```

Out[45]:
```
Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9320031923383879')
```



In [46]:
```
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_actua
df_actual_predicted.index = y_test.index

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

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
```

In [46]:
```
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_actua
df_actual_predicted.index = y_test.index

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

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[46]: <matplotlib.legend.Legend at 0x1dd0aa135b0>