

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

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

Out[2]:

	customer_id	Name	age	gender	security_no	region_category	membership_category	joining_
0	fffe4300490044003600300030003800	Pattie Morrisey	18	F	XW0DQ7H	Village	Platinum Membership	2017-0
1	fffe43004900440032003100300035003700	Traci Peery	32	F	5K0N3X1	City	Premium Membership	2017-0
2	fffe4300490044003100390032003600	Merideth Mcmeen	44	F	1F2TCL3	Town	No Membership	2016-0
3	fffe43004900440036003000330031003600	Eufemia Cardwell	37	M	VJGJ33N	City	No Membership	2016-1
4	fffe43004900440031003900350030003600	Meghan Kosak	31	F	SVZXCWB	City	No Membership	2017-0

◀ ▶

## Data Preprocessing Part 1

```
In [3]: # Drop identifier columns
df.drop(columns = ['customer_id', 'Name', 'security_no', 'referral_id'], inplace=True)
```

```
In [4]: df.head()
```

Out[4]:

	age	gender	region_category	membership_category	joining_date	joined_through_referral	preferred_offer_types	medium_o
0	18	F	Village	Platinum Membership	2017-08-17		No	Gift Vouchers/Coupons
1	32	F	City	Premium Membership	2017-08-28		?	Gift Vouchers/Coupons
2	44	F	Town	No Membership	2016-11-11		Yes	Gift Vouchers/Coupons
3	37	M	City	No Membership	2016-10-29		Yes	Gift Vouchers/Coupons
4	31	F	City	No Membership	2017-09-12		No	Credit/Debit Card Offers

◀ ▶

```
In [5]: df.shape
```

Out[5]: (36992, 21)

```
In [6]: #Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

```
Out[6]: 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 [7]: # Drop last_visit_time
df.drop(columns = 'last_visit_time', inplace=True)

# Only extract year from joining_date
df['joining_date'] = df['joining_date'].str[:4].astype(int)

# Change error from avg_frequency_login_days into null value
df['avg_frequency_login_days'] = df['avg_frequency_login_days'].replace('Error', np.nan).astype(float)
```

```
In [8]: #Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

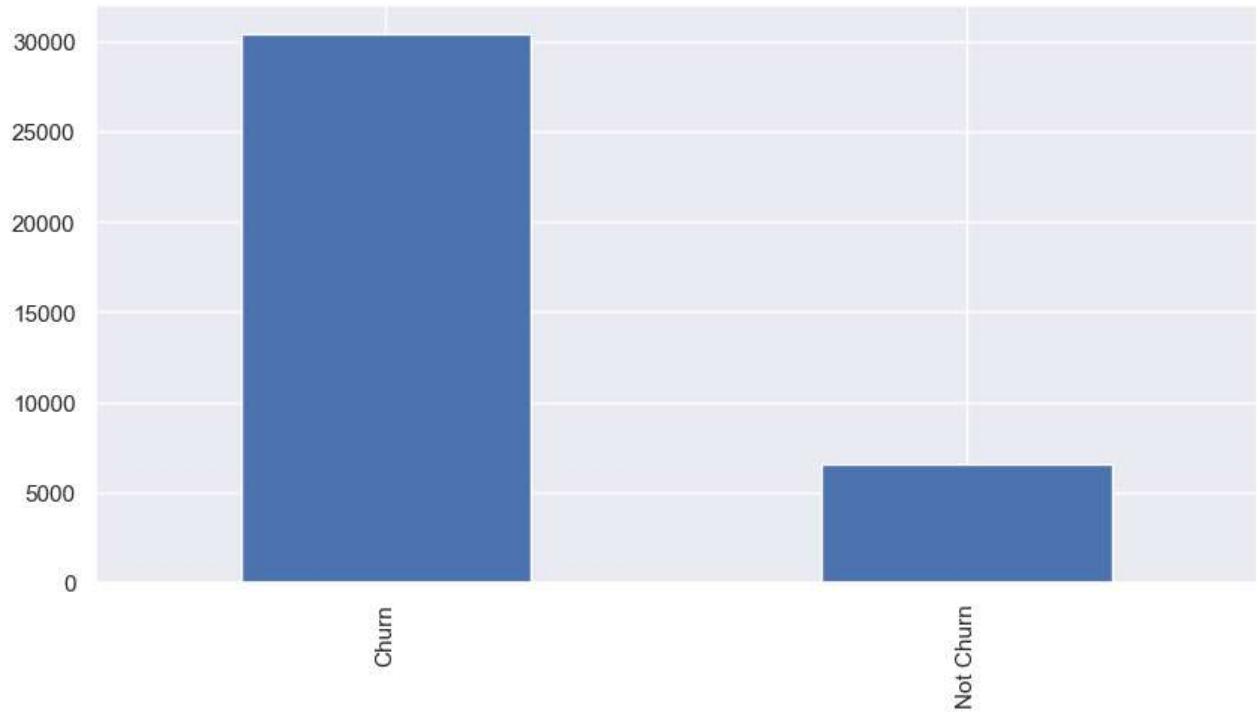
```
Out[8]: gender                      3
region_category                  3
membership_category                6
joined_through_referral            3
preferred_offer_types              3
medium_of_operation                 4
internet_option                   3
used_special_discount                2
offer_application_preference        2
past_complaint                     2
complaint_status                   5
feedback                           9
dtype: int64
```

```
In [9]: # Categorize churn_risk_score into 2 types
def categorize_churn(score):
    if score in (-1, 1, 2):
        return 'Not Churn'
    elif score in (3, 4, 5):
        return 'Churn'
    else:
        return 'Unknown'

df['churn_category'] = df['churn_risk_score'].apply(categorize_churn)
df['churn_category'] = df['churn_category'].astype(str)
```

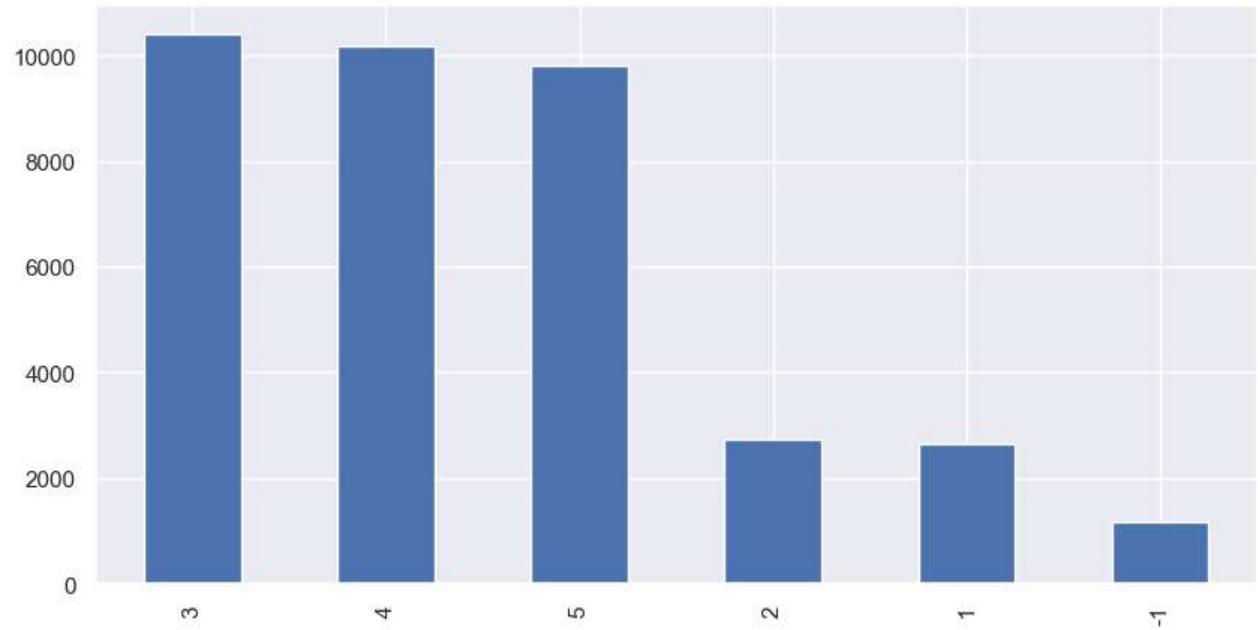
```
In [10]: plt.figure(figsize=(10,5))
df['churn_category'].value_counts().plot(kind='bar')
```

Out[10]: <AxesSubplot:>



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

Out[11]: <AxesSubplot:>



```
In [12]: df.drop(columns = 'churn_risk_score', inplace=True)
```

## Exploratory Data Analysis

```
In [13]: # 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 6 values of each categorical variable using Seaborn
for i, var in enumerate(cat_vars):
    top_values = df[var].value_counts().nlargest(6).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)

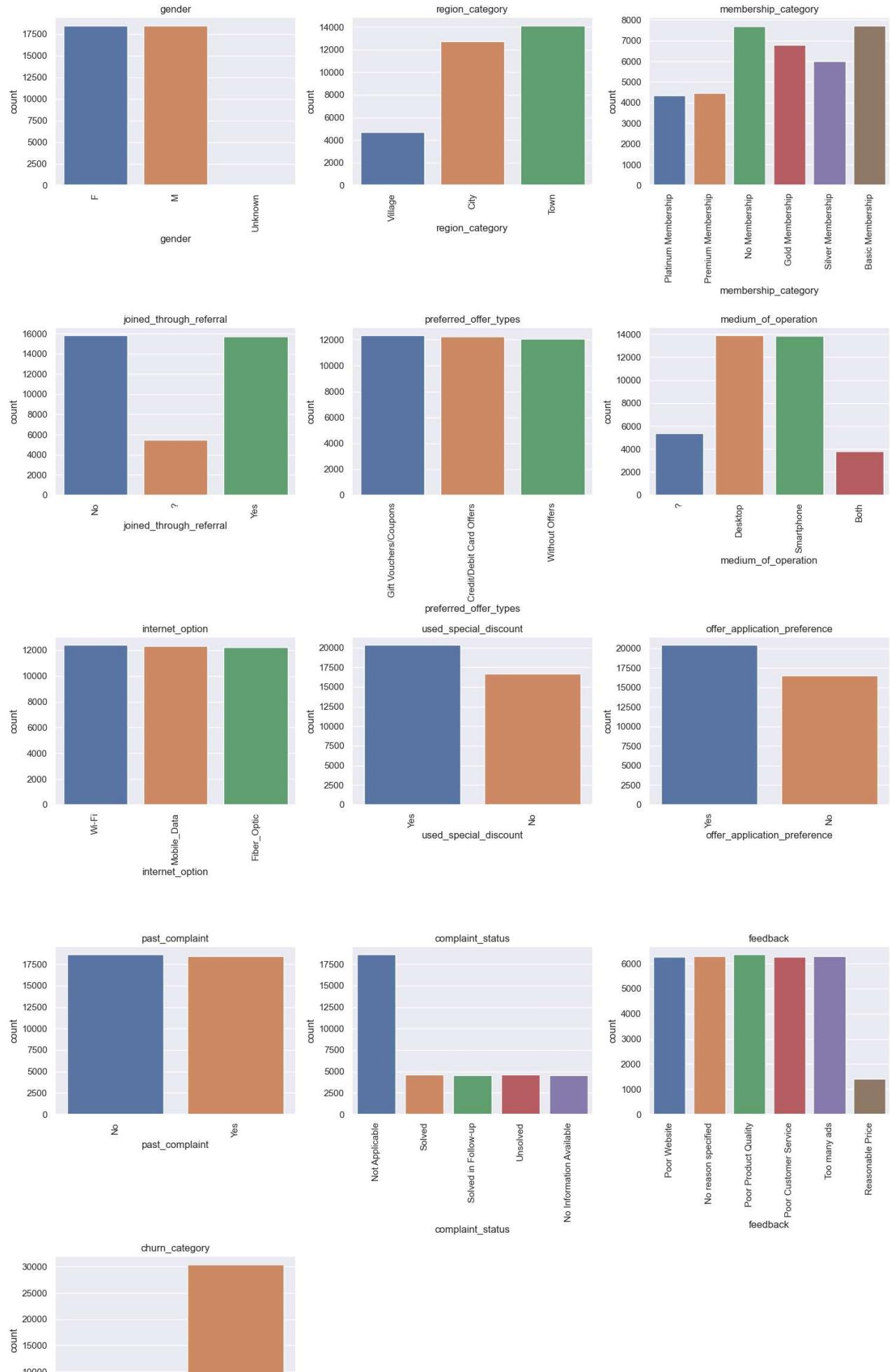
# Remove any extra empty subplots if needed
if num_cols < len(axs):
    for i in range(num_cols, len(axs)):
        fig.delaxes(axs[i])

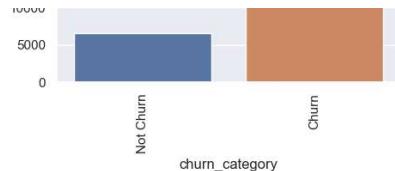
# Adjust spacing between subplots
fig.tight_layout()

# Show plot
plt.show()
```



## Churn Risk Rate - HackerEarth ML - Jupyter Notebook





```
In [14]: # Get the names of all columns with data type 'int' or 'float'
num_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

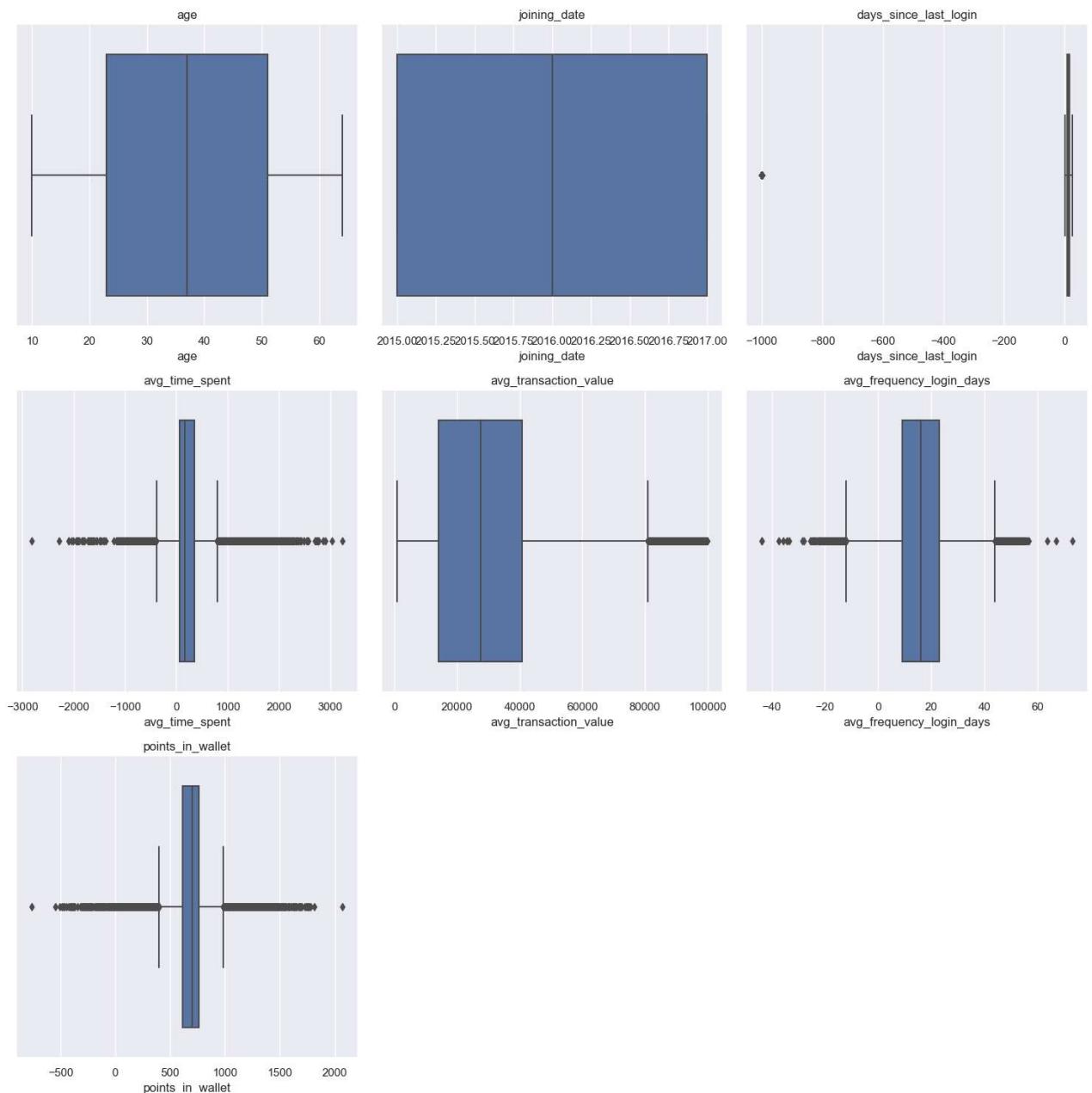
# Create a figure with subplots
num_cols = len(num_vars)
num_rows = (num_cols + 2) // 3
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# 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()
```



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

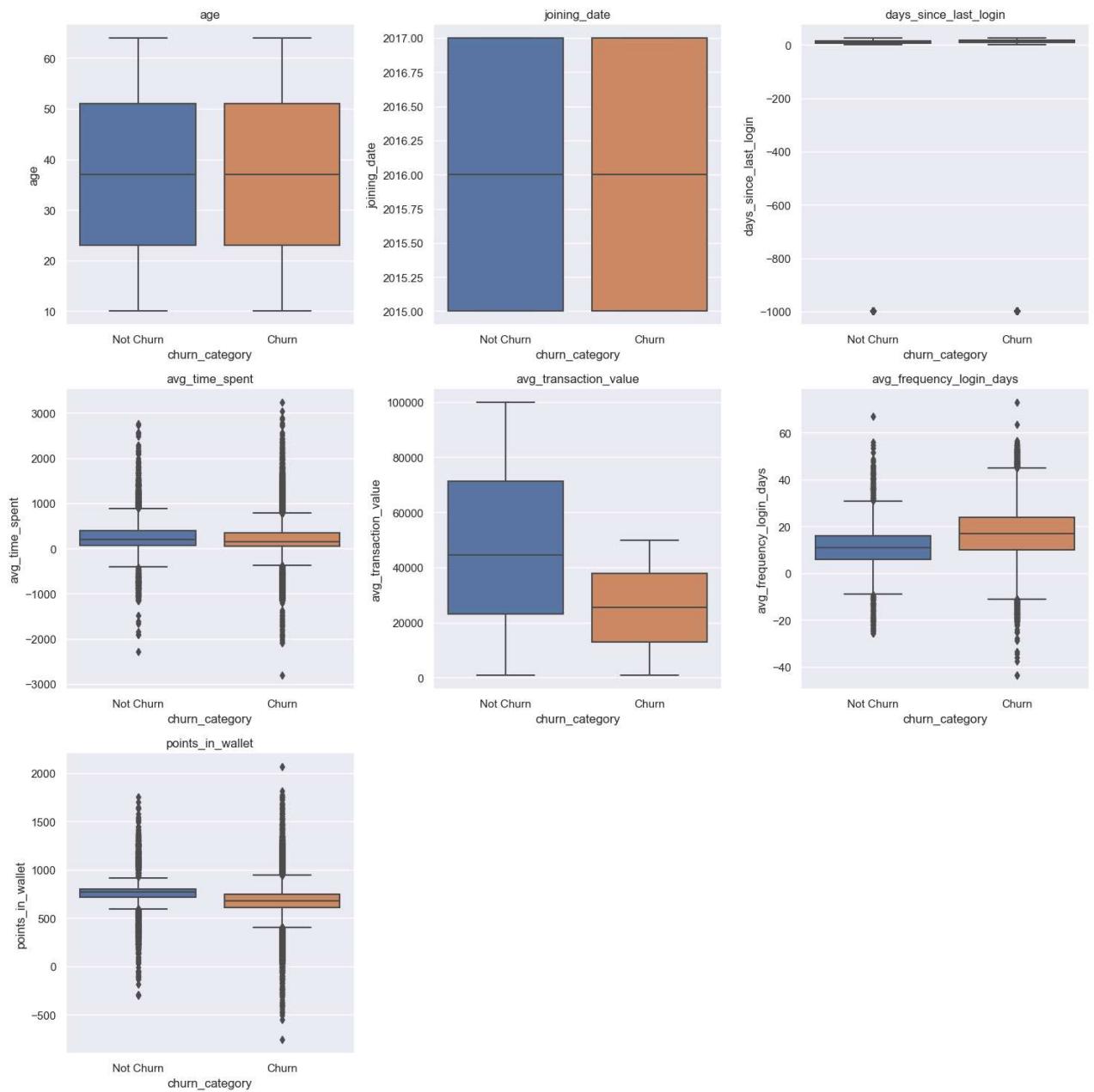
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3 # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# 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='churn_category', 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 [16]: # Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

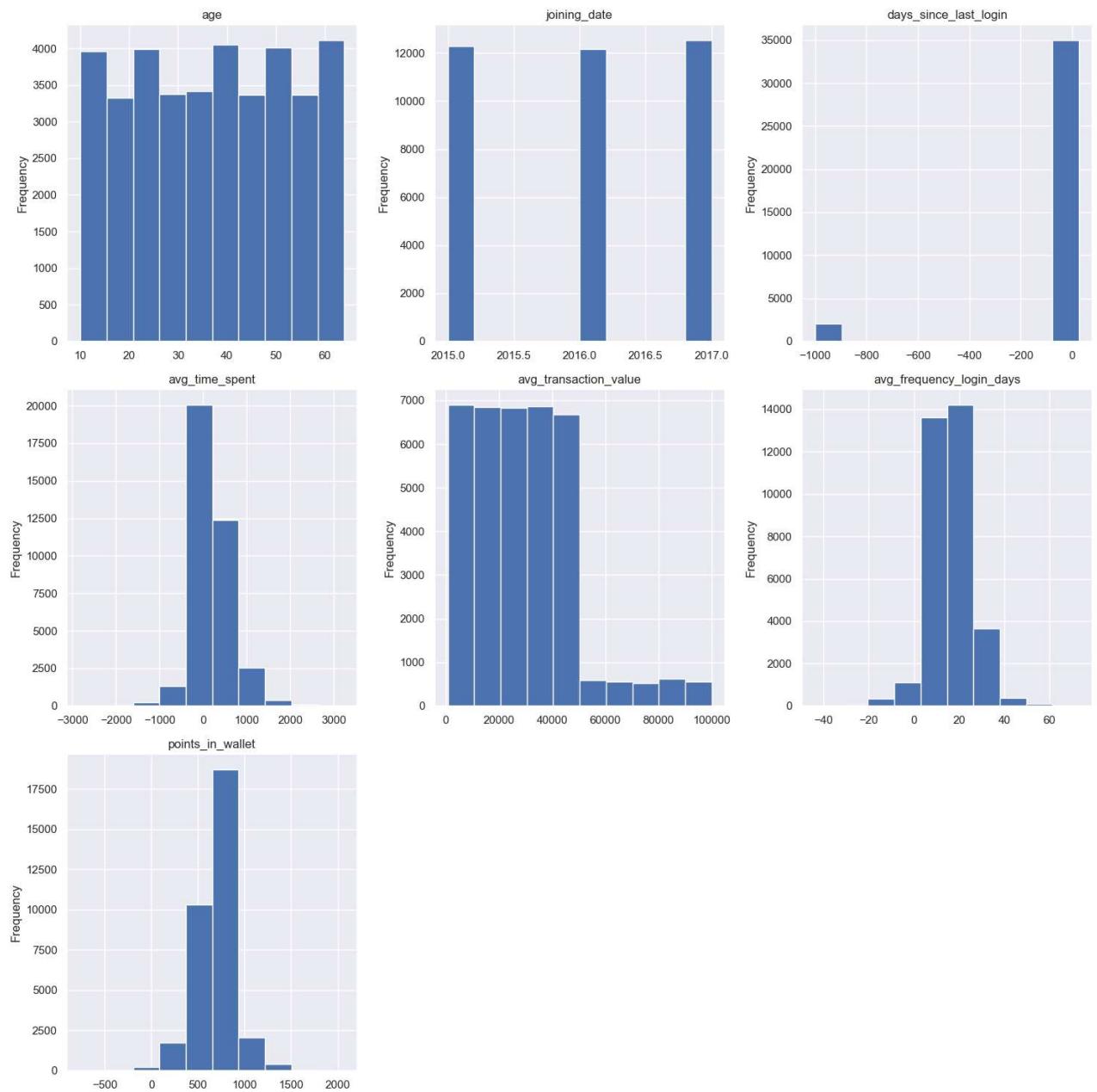
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3 # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# 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()
```



```
In [17]: # Get the names of all columns with data type 'int'
int_vars = df.select_dtypes(include=['int', 'float']).columns.tolist()

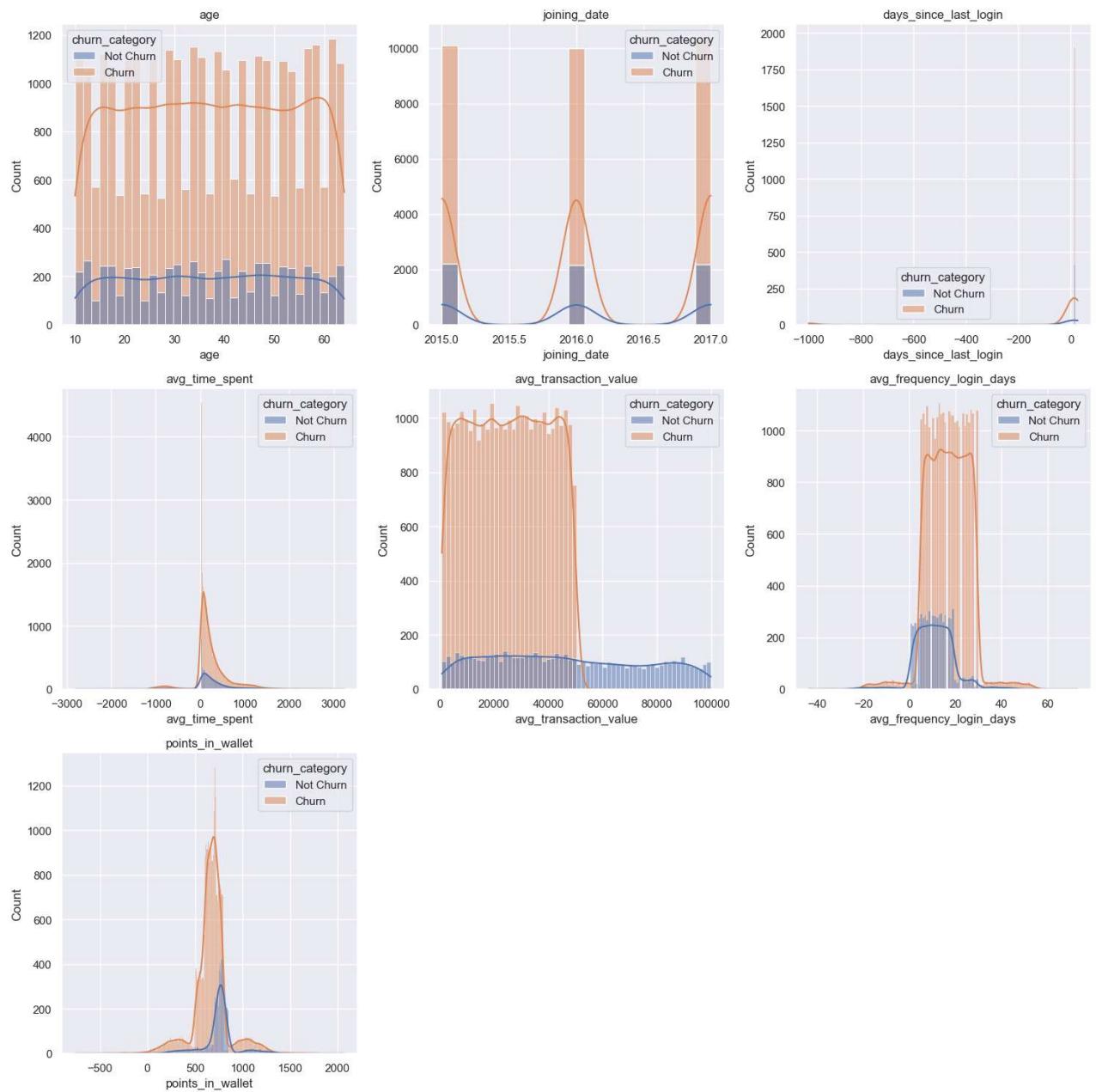
# Create a figure with subplots
num_cols = len(int_vars)
num_rows = (num_cols + 2) // 3 # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a histogram for each integer variable with hue='Attrition'
for i, var in enumerate(int_vars):
    sns.histplot(data=df, x=var, hue='churn_category', 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 [18]: # 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_category' in cat_vars:
    cat_vars.remove('churn_category')

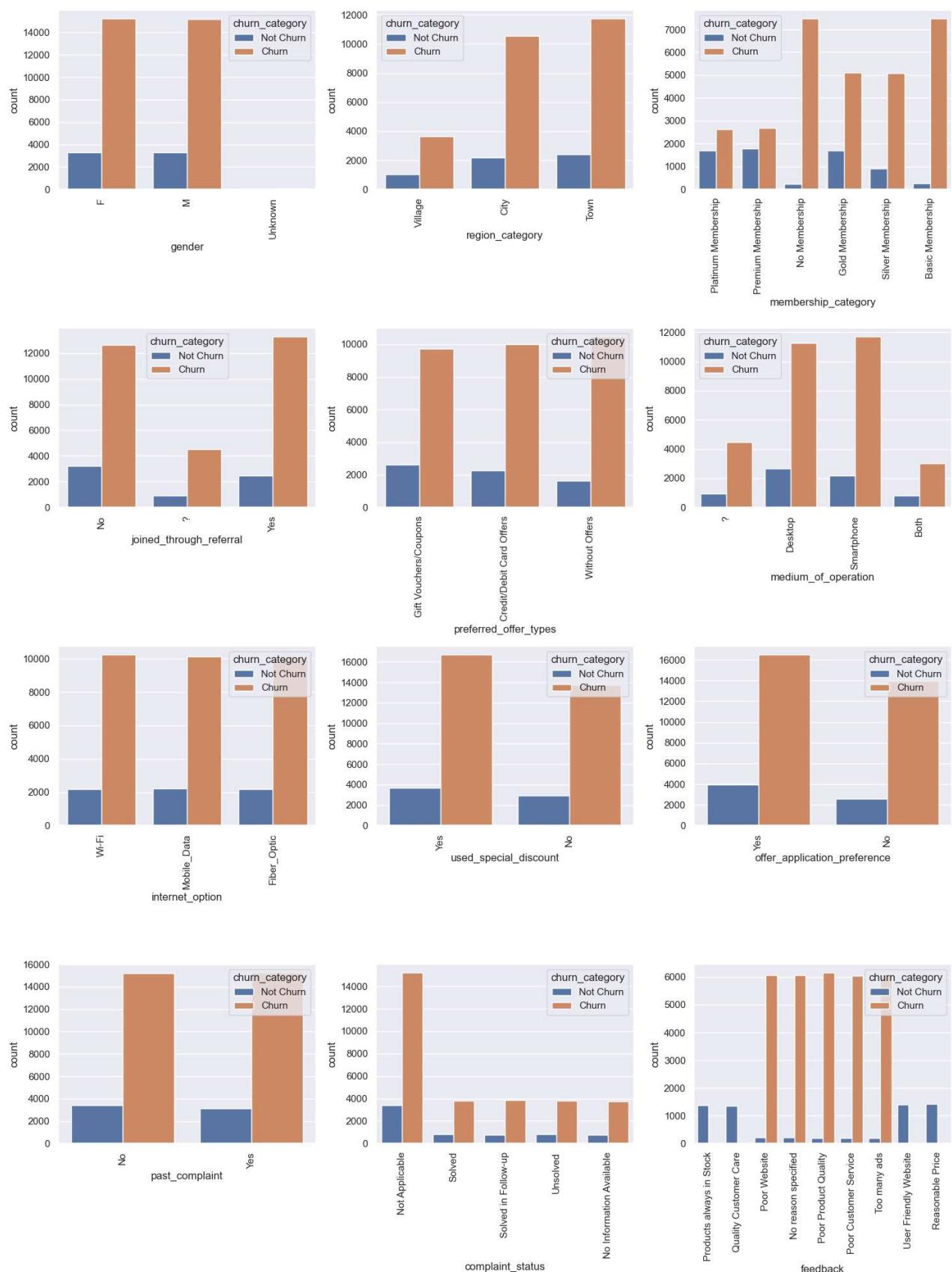
# Create a figure with subplots, but only include the required number of subplots
num_cols = len(cat_vars)
num_rows = (num_cols + 2) // 3 # To make sure there are enough rows for the subplots
fig, axs = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 5*num_rows))
axs = axs.flatten()

# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='churn_category', data=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 [19]: # 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_category' in cat_vars:
    cat_vars.remove('churn_category')

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

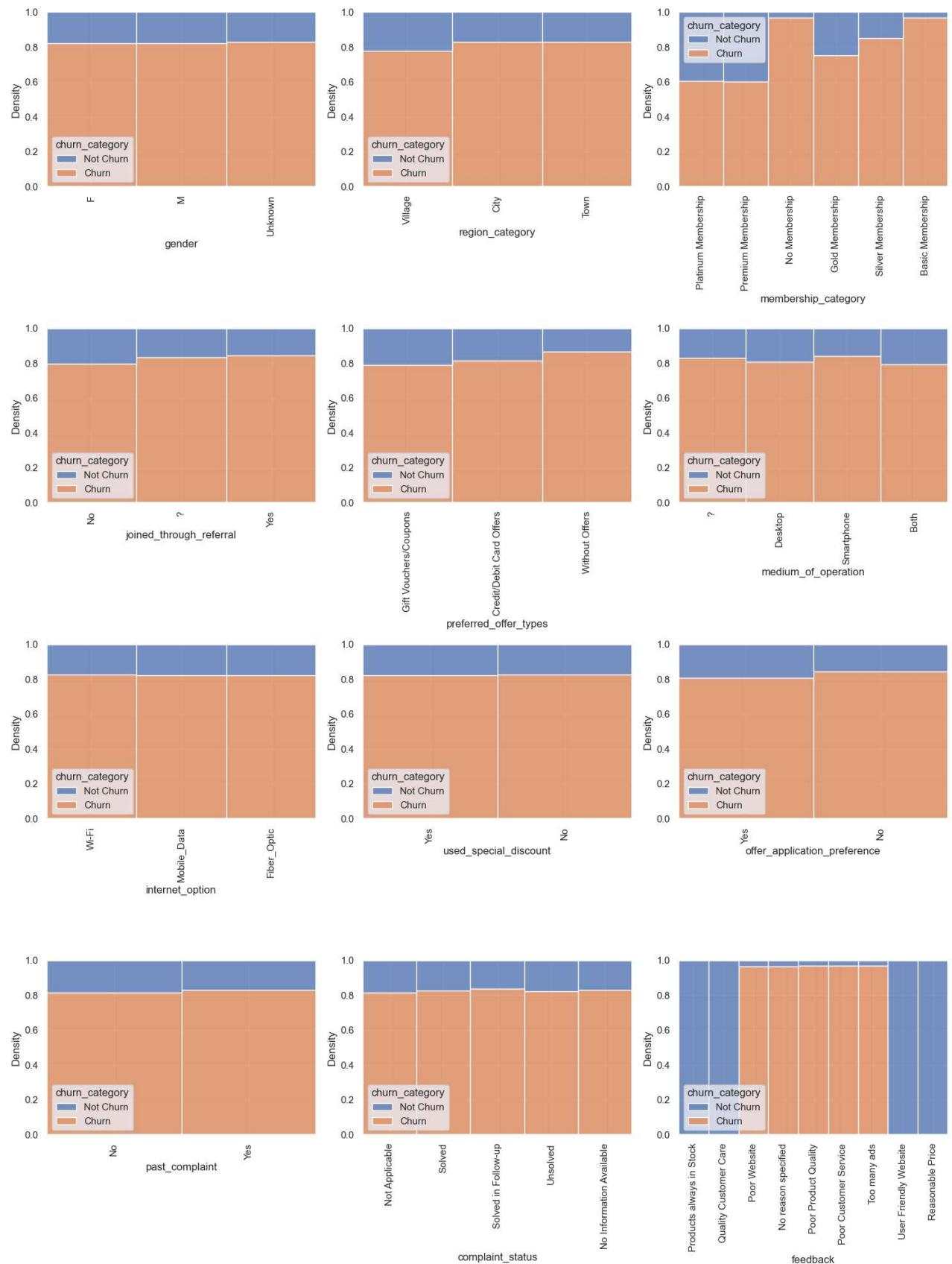
# Create a count plot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='churn_category', data=df, ax=axs[i], multiple="fill", kde=False, element="count")
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# 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()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\2500184484.py:17: UserWarning: FixedFormatter should only be used together with FixedLocator
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
```



```
In [20]: # 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=['Other'])
            cat_counts = cat_counts_top.append(cat_counts_other)

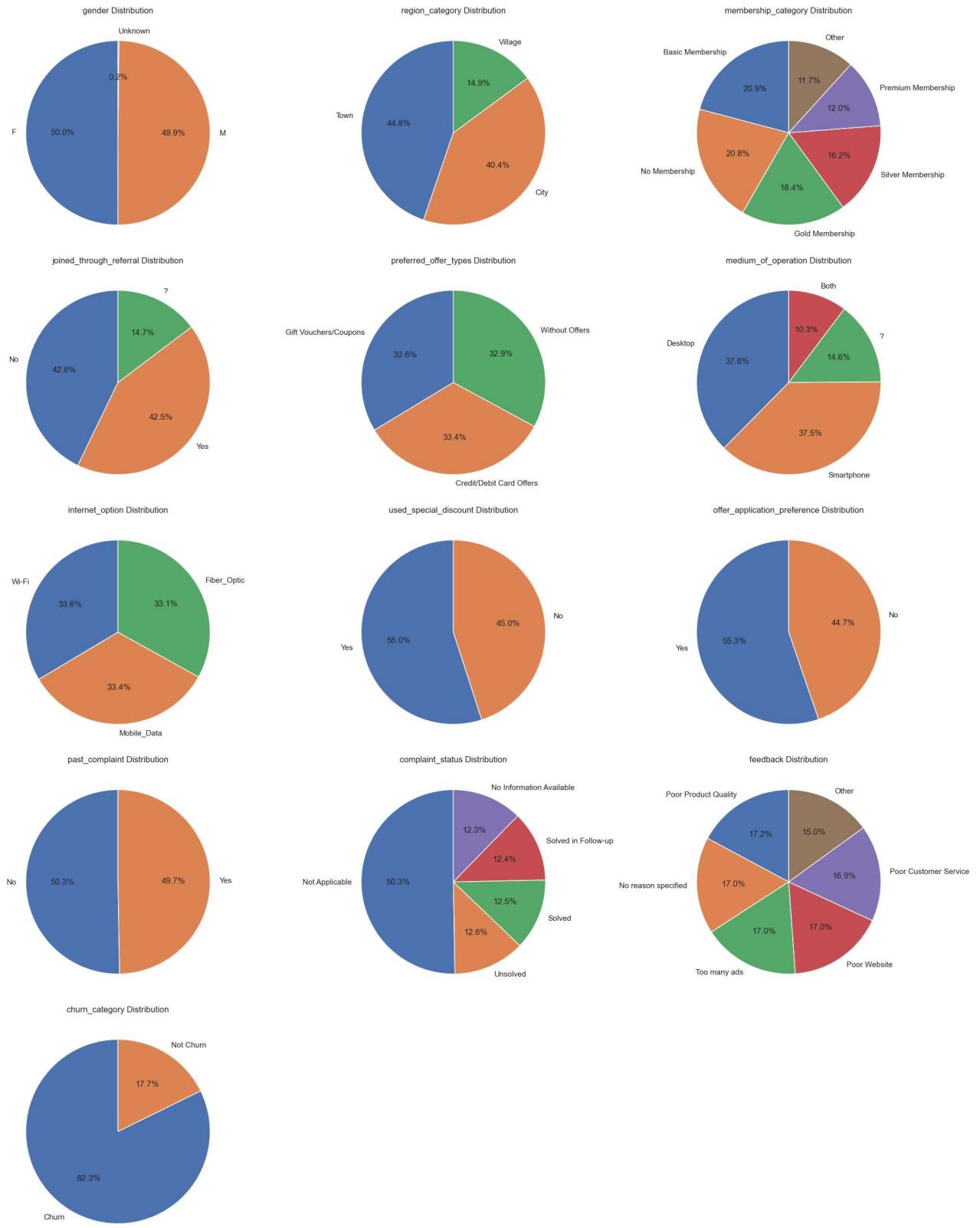
        # Create a pie chart
        axs[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)
        axs[i].set_title(f'{col} Distribution')

# 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()
```

```
C:\Users\Michael\AppData\Local\Temp\ipykernel_3508\943223290.py:25: FutureWarning: 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_3508\943223290.py:25: FutureWarning: 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 [21]: # Check the amountnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Out[21]: region_category      14.673443
avg_frequency_login_days    9.520978
points_in_wallet            9.307418
preferred_offer_types       0.778547
dtype: float64
```

```
In [22]: # Drop rows with null values in categorical columns
df = df.dropna(subset=['region_category', 'preferred_offer_types'])

# Fill null values with median for numerical columns
df['avg_frequency_login_days'].fillna(df['avg_frequency_login_days'].median(), inplace=True)
df['points_in_wallet'].fillna(df['points_in_wallet'].median(), inplace=True)
```

```
In [23]: # Check the amountnt of missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

```
Out[23]: Series([], dtype: float64)
```

## Label Encoding for Object Datatypes

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

gender: ['F' 'M' 'Unknown']
region_category: ['Village' 'City' 'Town']
membership_category: ['Platinum Membership' 'Premium Membership' 'No Membership'
    'Gold Membership' 'Silver Membership' 'Basic Membership']
joined_through_referral: ['No' '?' 'Yes']
preferred_offer_types: ['Gift Vouchers/Coupons' 'Credit/Debit Card Offers' 'Without 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']
churn_category: ['Not Churn' 'Churn']
```

```
In [25]: 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])

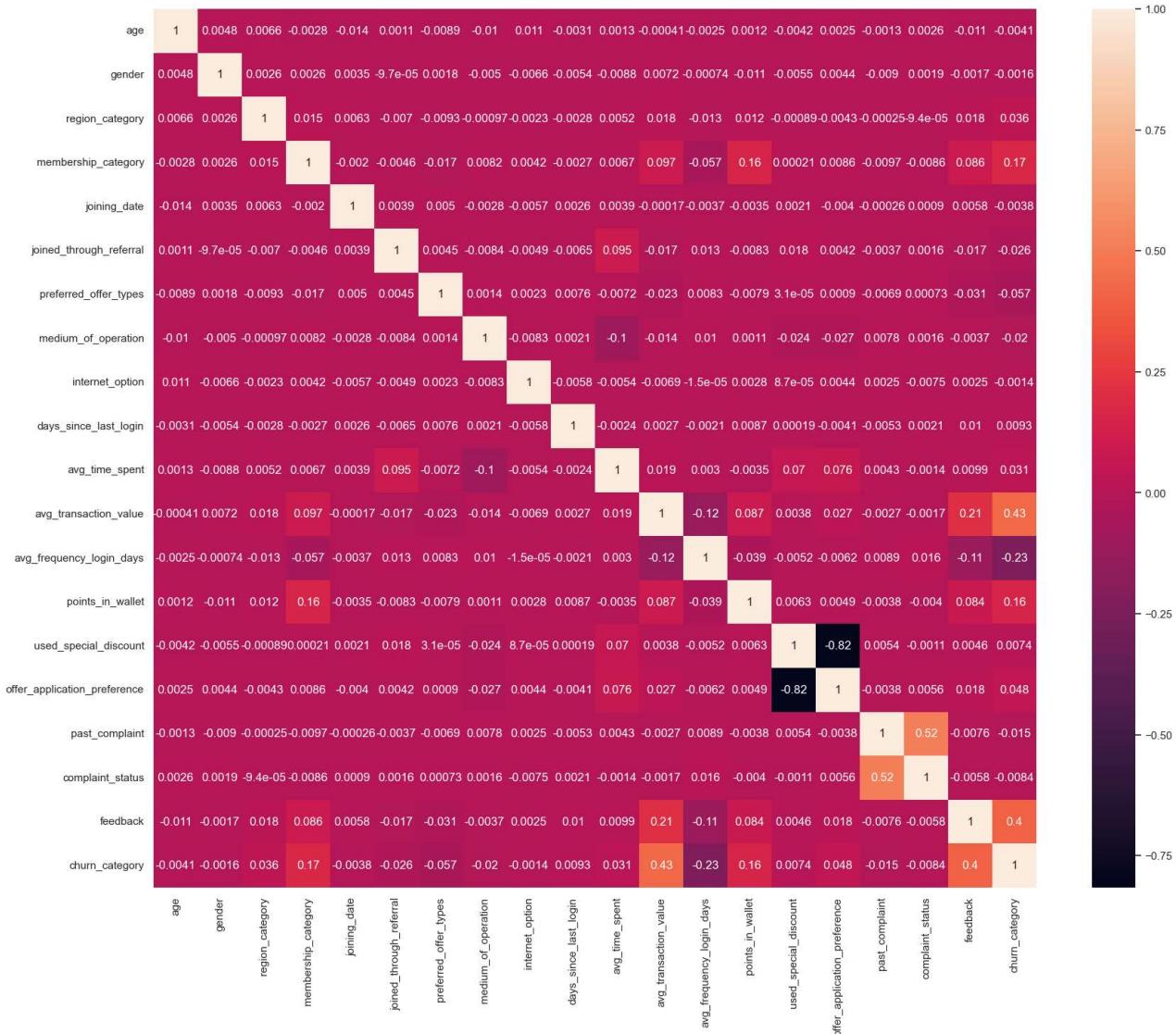
    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
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]
churn_category: [1 0]
```

In [26]: # Correlation Heatmap

```
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[26]: <AxesSubplot:>



## Train Test Split

```
In [27]: X = df.drop('churn_category', axis=1)
y = df['churn_category']
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 [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_value',
                    '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': 4, 'min_samples_leaf': 3, 'min_samples_split': 2, 'random_state': 0}
```

```
In [30]: from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=4, min_samples_leaf=3, min_samples_split=2,
                               dtree.fit(X_train, y_train)

Out[30]: DecisionTreeClassifier(class_weight='balanced', max_depth=4, min_samples_leaf=3,
                                 random_state=0)
```

```
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), "%")

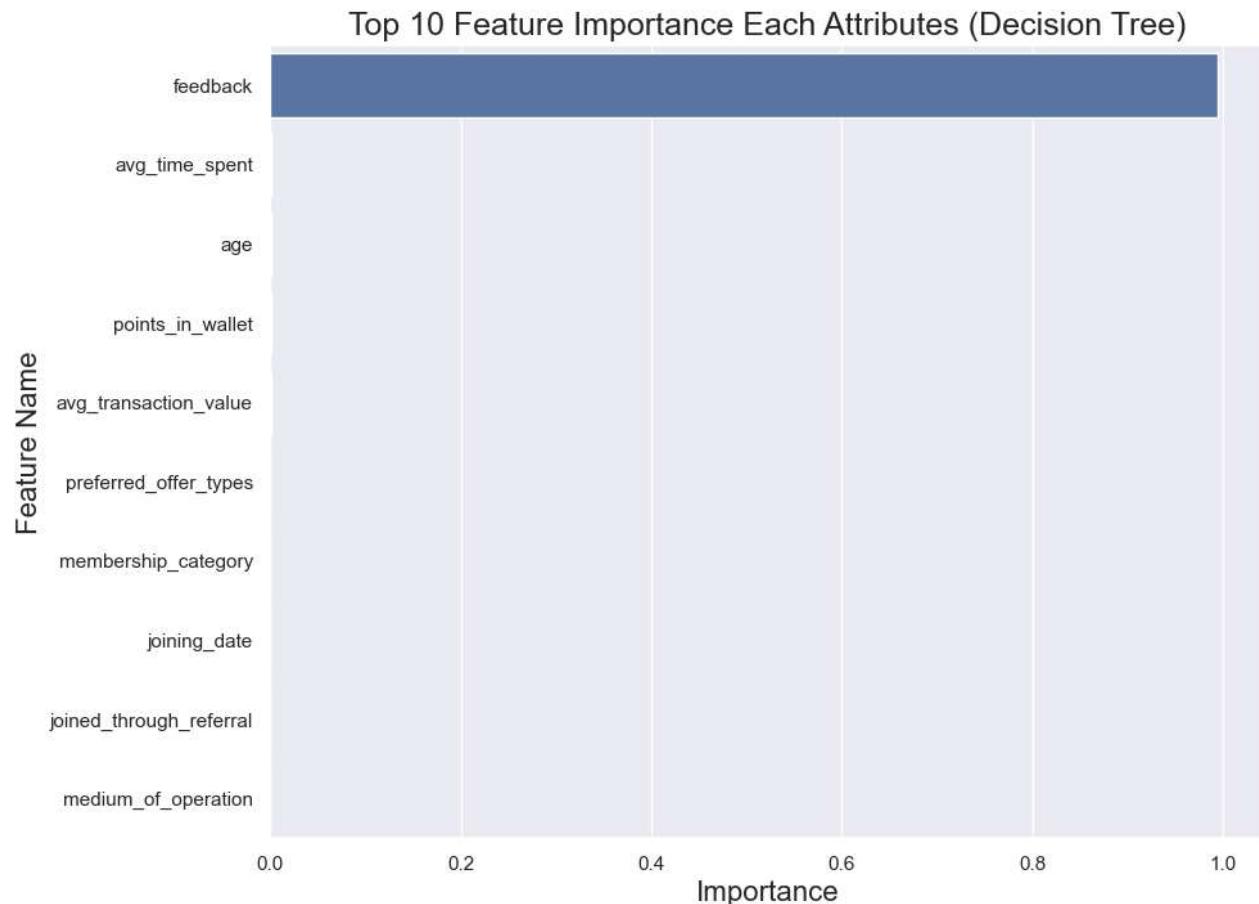
Accuracy Score : 97.02 %
```

```
In [32]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
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 : 0.9701516360734238  
 Precision Score : 0.9701516360734238  
 Recall Score : 0.9701516360734238  
 Jaccard Score : 0.9420334779913205  
 Log Loss : 1.0309292857751775

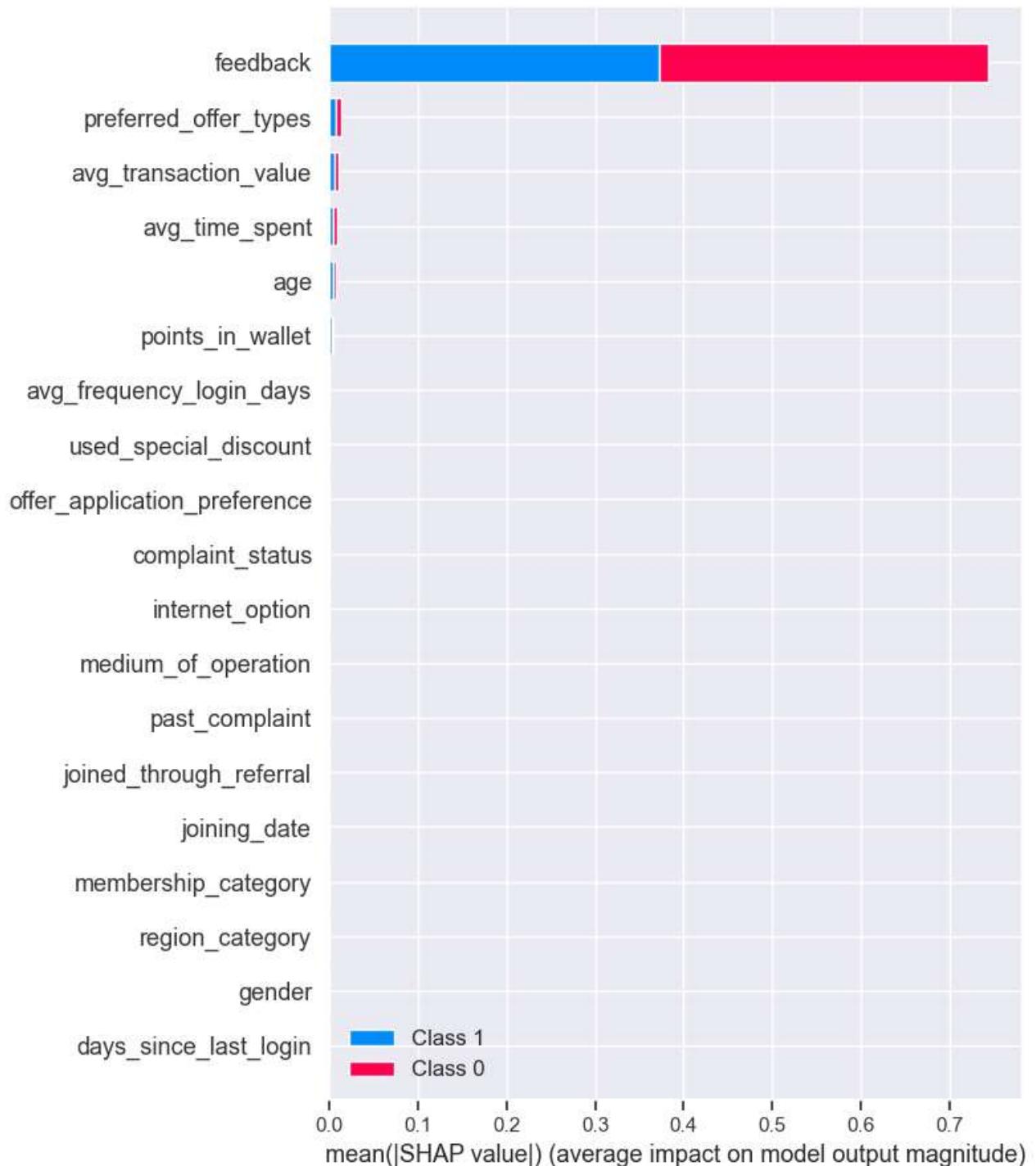
```
In [33]: imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

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=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```

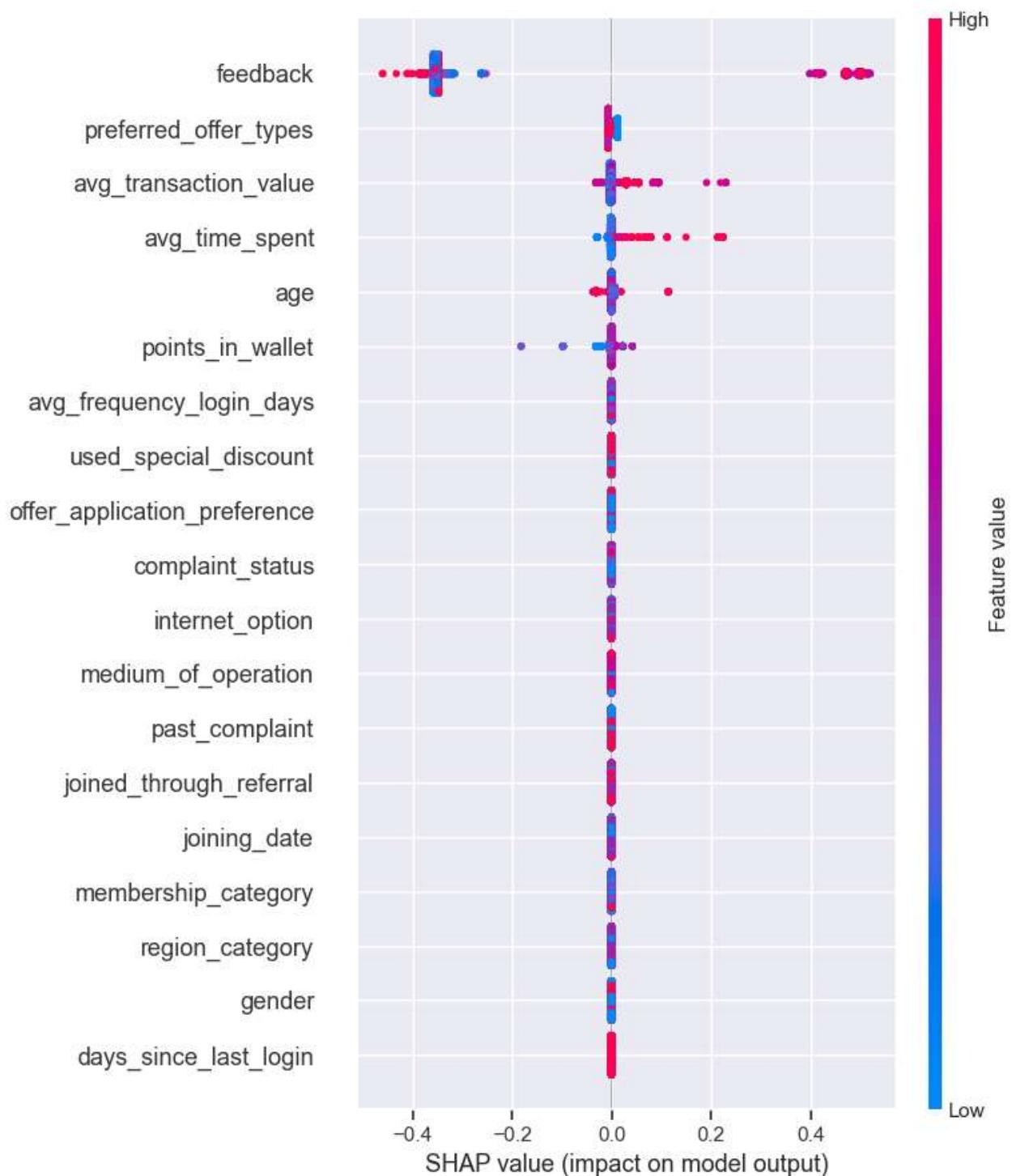


```
In [34]: import shap  
explainer = shap.TreeExplainer(dtree)  
shap_values = explainer.shap_values(X_test)  
shap.summary_plot(shap_values, X_test)
```

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

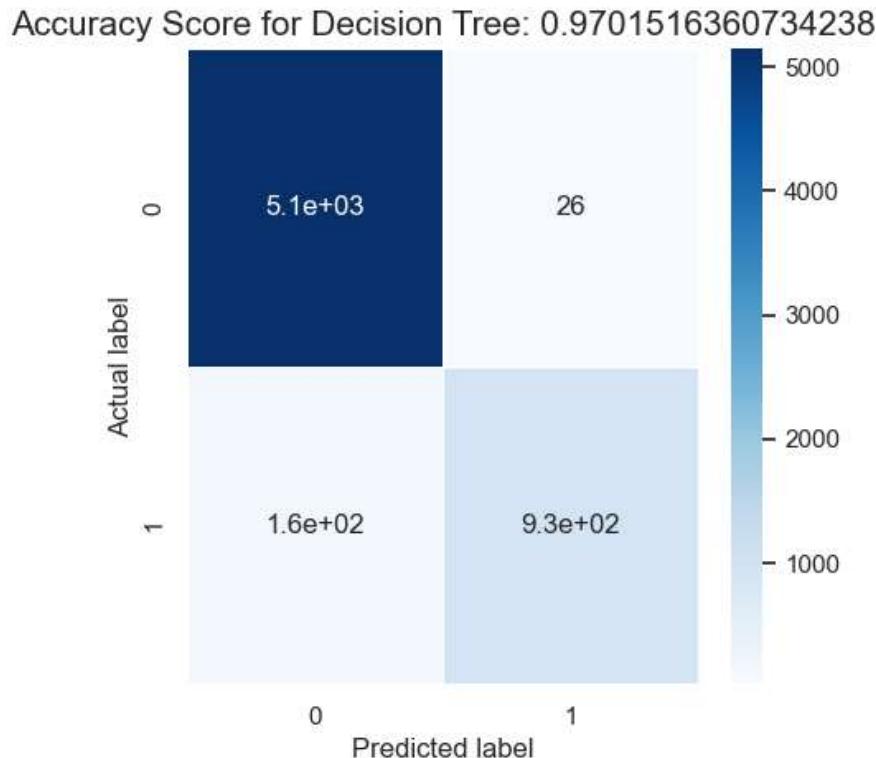


```
In [35]: # compute SHAP values
explainer = shap.TreeExplainer(dtrees)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



```
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_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[36]: Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.9701516360734238')



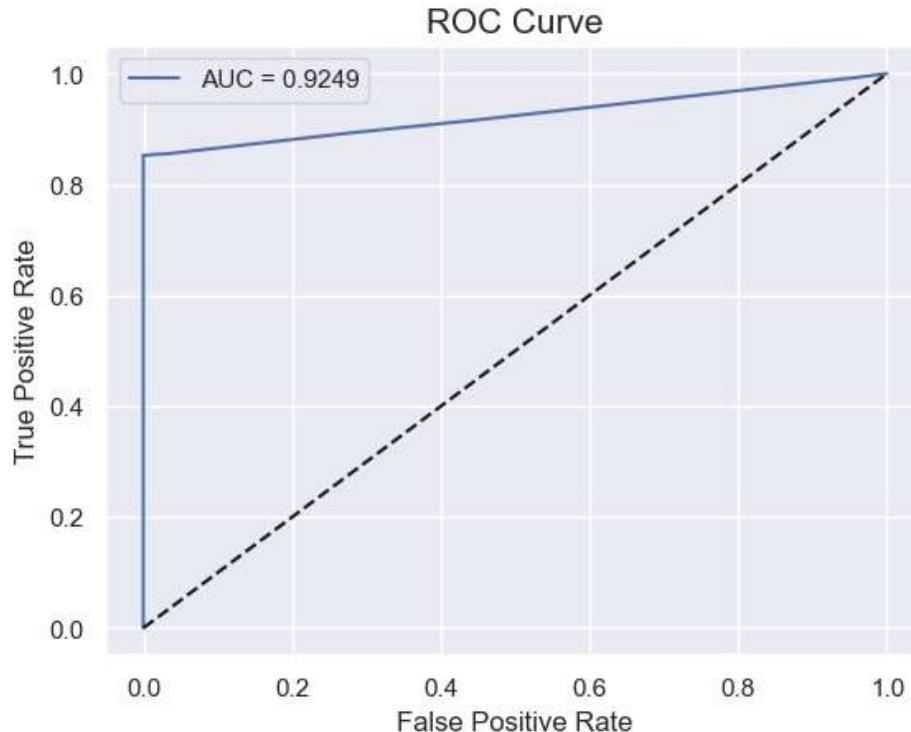
```
In [37]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba)], 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[37]: <matplotlib.legend.Legend at 0x18ee52f7880>



## 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
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 [39]: from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=0, max_depth=None, max_features=None, n_estimators=100, class_weight='balanced')
rfc.fit(X_train, y_train)
```

```
Out[39]: RandomForestClassifier(class_weight='balanced', max_features=None,
                                 random_state=0)
```

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

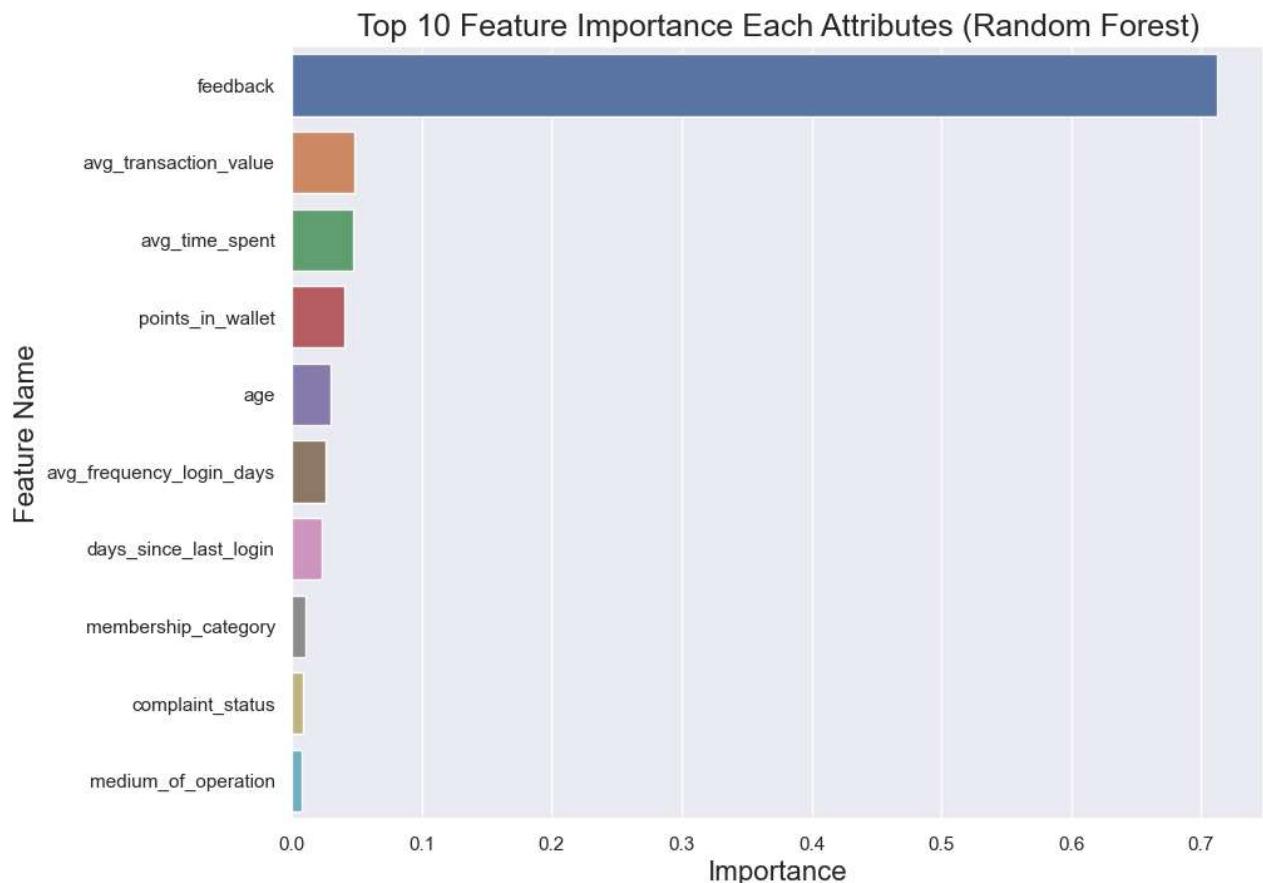
Accuracy Score : 97.41 %

```
In [41]: from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score, log_loss
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))
```

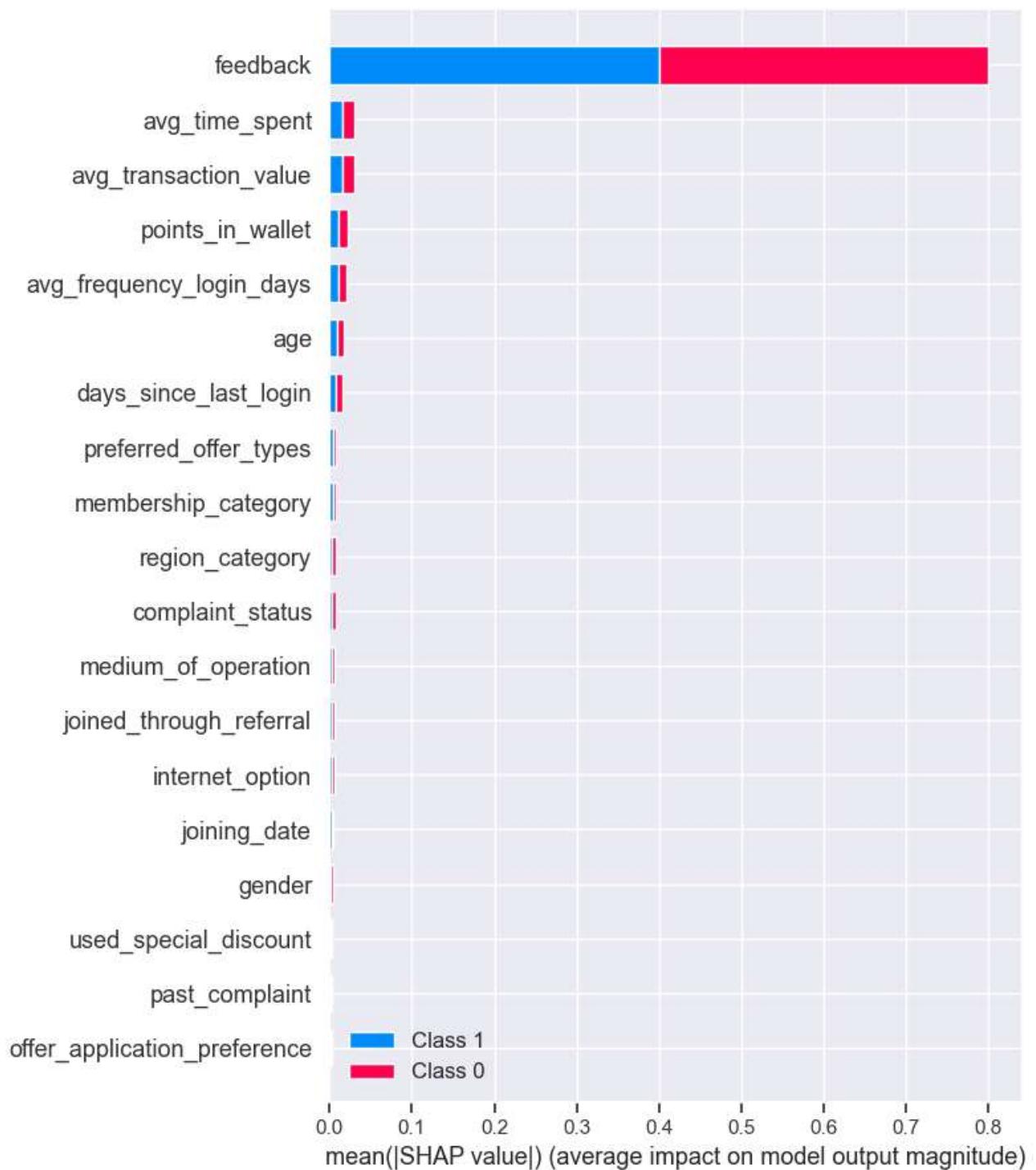
F-1 Score : 0.9741420590582601  
 Precision Score : 0.9741420590582601  
 Recall Score : 0.9741420590582601  
 Jaccard Score : 0.9495876769877081  
 Log Loss : 0.8931017678488374

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

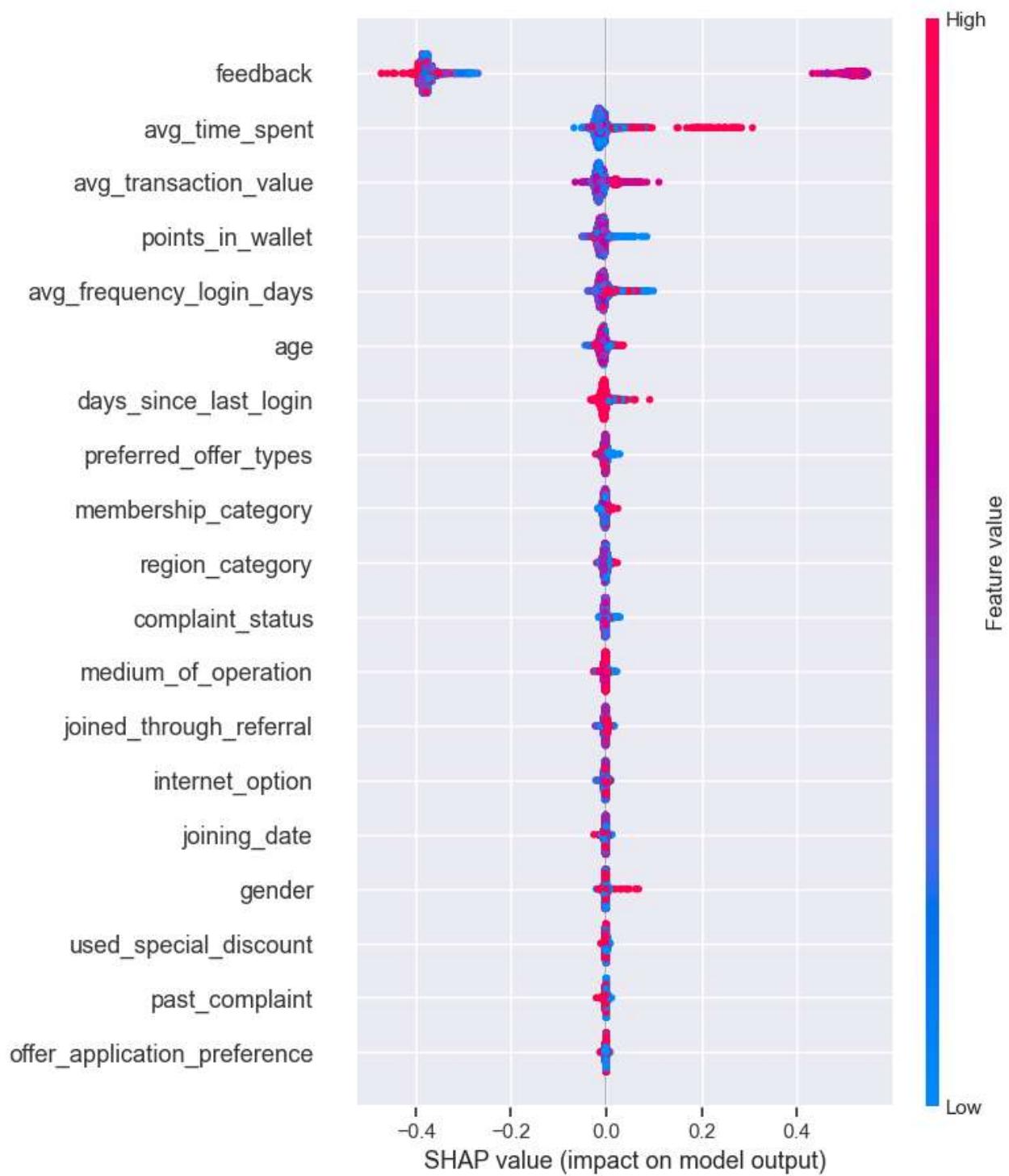
fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



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

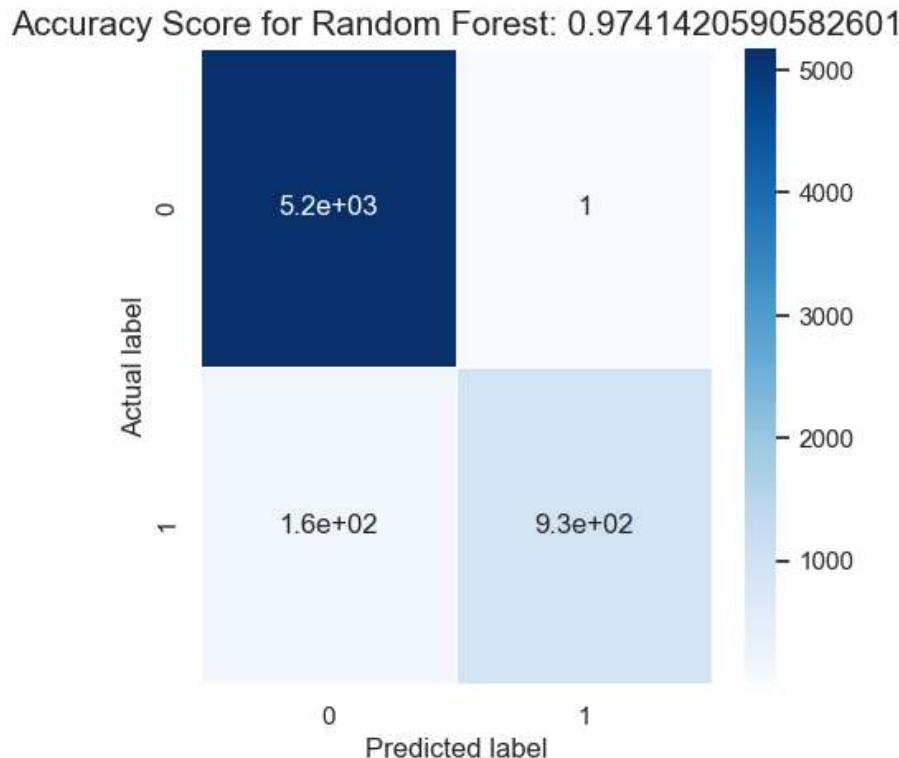


```
In [44]: # compute SHAP values
explainer = shap.TreeExplainer(rfc)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```



```
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_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[45]: Text(0.5, 1.0, 'Accuracy Score for Random Forest: 0.9741420590582601')



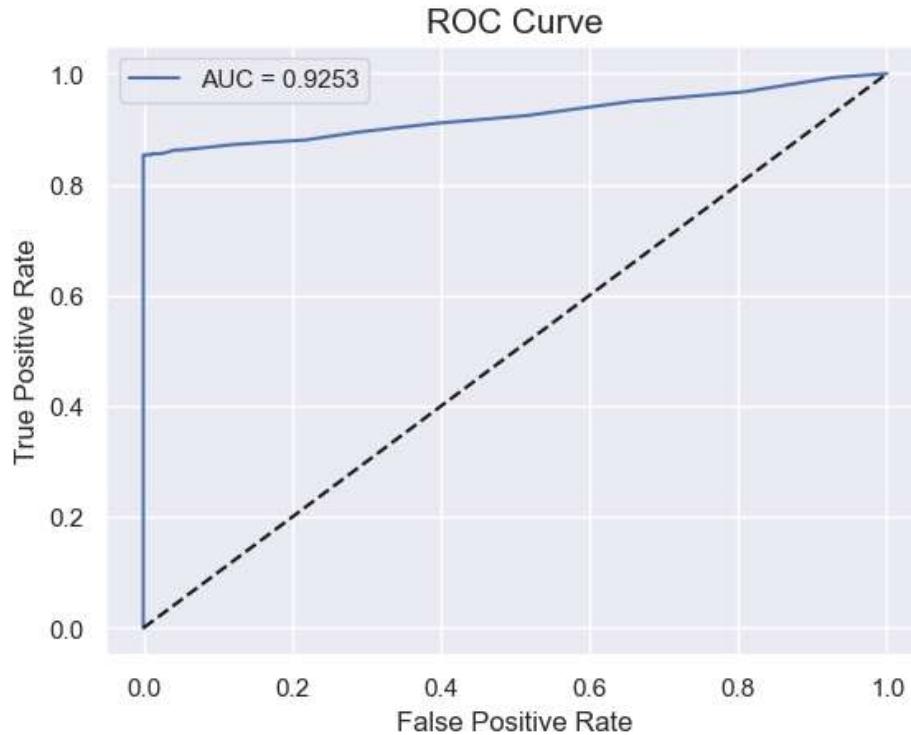
```
In [46]: from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = rfc.predict_proba(X_test)[:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame(y_pred_proba)], 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[46]: <matplotlib.legend.Legend at 0x18ee52f7820>
```



```
In [ ]:
```