Customer_Segmentation_and_Profiling

[ ]:
```
# Customer Segmentation and Profiling
### Author: Scott Eugley
### Date: 5/3/2024
```

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Load the CSV file into a DataFrame
df = pd.read_csv('customer_data.csv')
```

[2]:
```
# Value Based Segmentation
```

[3]:
```
# Min, Max, Range, and Median of PhoneCoTenure

# Find the minimum value of PhoneCoTenure
min_phone_co_tenure = df['PhoneCoTenure'].min()

# Find the maximum value of PhoneCoTenure
max_phone_co_tenure = df['PhoneCoTenure'].max()

# Calculate the range of PhoneCoTenure
range_phone_co_tenure = max_phone_co_tenure - min_phone_co_tenure

# Calculate the median of PhoneCoTenure
median_phone_co_tenure = df['PhoneCoTenure'].median()

# Calculate the mean of PhoneCoTenure
mean_phone_co_tenure = df['PhoneCoTenure'].mean()

# Print the mean
print("Mean of PhoneCoTenure:", mean_phone_co_tenure)

# Print the median
print("Median value of PhoneCoTenure:", median_phone_co_tenure)



# Print the results
print("Minimum value of PhoneCoTenure:", min_phone_co_tenure)
```

```python
print("Maximum value of PhoneCoTenure:", max_phone_co_tenure)
print("Range of PhoneCoTenure:", range_phone_co_tenure)
```

```
Mean of PhoneCoTenure: 38.26060233558697
Median value of PhoneCoTenure: 38.0
Minimum value of PhoneCoTenure: 1
Maximum value of PhoneCoTenure: 72
Range of PhoneCoTenure: 71
```

[4]:
```python
# Create a new column 'phone_co_tenure_hi_lo' based on the median. >38 is high␣
↪and <=38 is low.
df['phone_co_tenure_hi_lo'] = df['PhoneCoTenure'].apply(lambda x: 'high' if x >␣
↪median_phone_co_tenure else 'low')

# Print the first few rows to verify the new column
print(df[['PhoneCoTenure', 'phone_co_tenure_hi_lo']].head())
```

```
   PhoneCoTenure phone_co_tenure_hi_lo
0              5                   low
1             39                  high
2             65                  high
3             36                   low
4             21                   low
```

[5]:
```python
# Get counts of 'high' and 'low' in the 'phone_co_tenure_hi_lo' column
counts = df['phone_co_tenure_hi_lo'].value_counts()

# Print the counts
print("Counts of 'high' and 'low' in the 'phone_co_tenure_hi_lo' column:")
print(counts)
```

```
Counts of 'high' and 'low' in the 'phone_co_tenure_hi_lo' column:
low     2454
high    2427
Name: phone_co_tenure_hi_lo, dtype: int64
```

[6]:
```python
# Calculate the minimum of data_equip_voice_sum
min_data_equip_voice_sum = df['data_equip_voice_sum'].min()

# Calculate the maximum of data_equip_voice_sum
max_data_equip_voice_sum = df['data_equip_voice_sum'].max()

# Calculate the range of data_equip_voice_sum
range_data_equip_voice_sum = max_data_equip_voice_sum - min_data_equip_voice_sum

# Calculate the median of data_equip_voice_sum
median_data_equip_voice_sum = df['data_equip_voice_sum'].median()
```

```
# Calculate the mean of data_equip_voice_sum
mean_data_equip_voice_sum = df['data_equip_voice_sum'].mean()

# Print the mean
print("Mean of data_equip_voice_sum:", mean_data_equip_voice_sum)


# Print the results
print("Minimum of data_equip_voice_sum:", min_data_equip_voice_sum)
print("Maximum of data_equip_voice_sum:", max_data_equip_voice_sum)
print("Range of data_equip_voice_sum:", range_data_equip_voice_sum)
print("Median of data_equip_voice_sum:", median_data_equip_voice_sum)
```

```
Mean of data_equip_voice_sum: 64.16096086867445
Minimum of data_equip_voice_sum: 2.85
Maximum of data_equip_voice_sum: 590.4
Range of data_equip_voice_sum: 587.55
Median of data_equip_voice_sum: 49.85
```

[7]:
```
# Create new column 'total_monthly_spend_high_low'. high is 50 or greater and␣
↪low is less than 50.

# Define the threshold value for segmentation
threshold = 50

# Create a new column 'total_monthly_spend_high_low' based on the segmentation␣
↪around the threshold
df['total_monthly_spend_high_low'] = ['high' if x >= threshold else 'low' for x␣
↪in df['data_equip_voice_sum']]

print(df[['data_equip_voice_sum', 'total_monthly_spend_high_low']].head())
```

```
   data_equip_voice_sum total_monthly_spend_high_low
0                  49.0                          low
1                 127.2                         high
2                  85.2                         high
3                  18.0                          low
4                  28.2                          low
```

[8]:
```
# Count the occurrences of high and low values in the␣
↪'total_monthly_spend_high_low' column
high_low_counts = df['total_monthly_spend_high_low'].value_counts()

# Print the counts
print("Counts of high and low values in the 'total_monthly_spend_high_low'␣
↪column:")
print(high_low_counts)
```

```
Counts of high and low values in the 'total_monthly_spend_high_low' column:
low     2448
high    2433
Name: total_monthly_spend_high_low, dtype: int64
```

[9]:
```python
# Sum up number of premium services used by each customer, then classify as␣
↪either high (4 or higher) or low (less than 4)

# Create the 'num_prem_serv' column by summing the values of premium service␣
↪columns
premium_services = ['Multiline', 'VM', 'Pager', 'Internet', 'CallerID',␣
↪'CallWait', 'CallForward', 'ThreeWayCalling']
df['num_prem_serv'] = df[premium_services].apply(lambda row: row[row == 'Yes'].
↪count(), axis=1)

# Create the 'num_prem_serv_high_low' column based on the 'num_prem_serv' column
df['num_prem_serv_high_low'] = df['num_prem_serv'].apply(lambda x: 'high' if x␣
↪>= 4 else 'low')

# Display the first few rows to verify
print(df[['num_prem_serv', 'num_prem_serv_high_low']].head(10))
```

```
   num_prem_serv num_prem_serv_high_low
0              6                   high
1              5                   high
2              1                    low
3              1                    low
4              5                   high
5              5                   high
6              2                    low
7              5                   high
8              1                    low
9              0                    low
```

[10]:
```python
# Count of high and low values in the 'num_prem_serv_high_low' column
high_low_counts = df['num_prem_serv_high_low'].value_counts()

# Display the count of high and low values
print("Count of 'high' and 'low' values in 'num_prem_serv_high_low' column:")
print(high_low_counts)
```

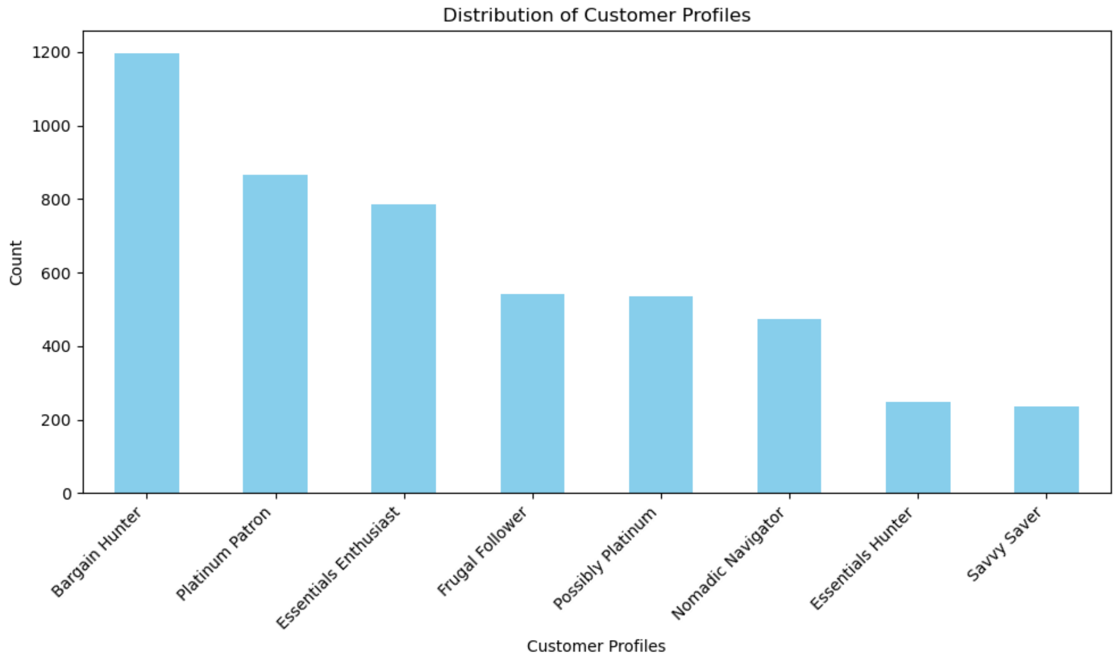
```
Count of 'high' and 'low' values in 'num_prem_serv_high_low' column:
low     2770
high    2111
Name: num_prem_serv_high_low, dtype: int64
```

[11]:
```python
# Define a function to assign profile labels based on the provided criteria
def assign_profile_label(row):
```

```
    if row['phone_co_tenure_hi_lo'] == 'high' and␣
↪row['total_monthly_spend_high_low'] == 'high' and␣
↪row['num_prem_serv_high_low'] == 'high':
        return 'Platinum Patron'
    elif row['phone_co_tenure_hi_lo'] == 'high' and␣
↪row['total_monthly_spend_high_low'] == 'high' and␣
↪row['num_prem_serv_high_low'] == 'low':
        return 'Essentials Enthusiast'
    elif row['phone_co_tenure_hi_lo'] == 'high' and␣
↪row['total_monthly_spend_high_low'] == 'low' and row['num_prem_serv_high_low']␣
↪== 'high':
        return 'Savvy Saver'
    elif row['phone_co_tenure_hi_lo'] == 'low' and␣
↪row['total_monthly_spend_high_low'] == 'high' and␣
↪row['num_prem_serv_high_low'] == 'high':
        return 'Possibly Platinum'
    elif row['phone_co_tenure_hi_lo'] == 'high' and␣
↪row['total_monthly_spend_high_low'] == 'low' and row['num_prem_serv_high_low']␣
↪== 'low':
        return 'Frugal Follower'
    elif row['phone_co_tenure_hi_lo'] == 'low' and␣
↪row['total_monthly_spend_high_low'] == 'low' and row['num_prem_serv_high_low']␣
↪== 'high':
        return 'Nomadic Navigator'
    elif row['phone_co_tenure_hi_lo'] == 'low' and␣
↪row['total_monthly_spend_high_low'] == 'high' and␣
↪row['num_prem_serv_high_low'] == 'low':
        return 'Essentials Hunter'
    elif row['phone_co_tenure_hi_lo'] == 'low' and␣
↪row['total_monthly_spend_high_low'] == 'low' and row['num_prem_serv_high_low']␣
↪== 'low':
        return 'Bargain Hunter'
    else:
        return 'Unknown'

# Apply the function to create the customer_profiles column
df['customer_profiles'] = df.apply(assign_profile_label, axis=1)
```

[12]:
```
# Count the occurrences of each customer profile
profile_counts = df['customer_profiles'].value_counts()

# Plot the histogram
plt.figure(figsize=(10, 6))
profile_counts.plot(kind='bar', color='skyblue')
plt.title('Distribution of Customer Profiles')
plt.xlabel('Customer Profiles')
```

```
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[13]:
```
profile_counts = df['customer_profiles'].value_counts()
print(profile_counts)
```

```
Bargain Hunter          1197
Platinum Patron          866
Essentials Enthusiast    785
Frugal Follower          541
Possibly Platinum        535
Nomadic Navigator        475
Essentials Hunter        247
Savvy Saver              235
Name: customer_profiles, dtype: int64
```

[14]:
```
# Group the DataFrame by 'customer_profiles' and calculate the desired statistics
profile_statistics = df.groupby('customer_profiles').agg({
    'Age': 'mean',
    'DebtToIncomeRatio': 'mean',
    'CardSpendMonth': 'mean',
    'EducationYears': 'mean',
    'HHIncome': 'mean',
    'Gender': lambda x: x.value_counts().to_dict(),  # Count of each gender
```

```
    'total_debt': 'mean'
}).reset_index()

# Create new columns for male and female counts
profile_statistics['Male_Count'] = profile_statistics['Gender'].apply(lambda x:␣
↪x.get('Male', 0))
profile_statistics['Female_Count'] = profile_statistics['Gender'].apply(lambda x:
↪ x.get('Female', 0))

# Drop the original 'Gender' column
profile_statistics.drop(columns=['Gender'], inplace=True)

# Rename the columns for clarity
profile_statistics.rename(columns={
    'Age': 'Avg_Age',
    'DebtToIncomeRatio': 'Avg_DTI',
    'CardSpendMonth': 'Avg_CardSpendMonth',
    'EducationYears': 'Avg_EducationYears',
    'HHIncome': 'Avg_HHIncome',
    'total_debt': 'Avg_total_debt'
}, inplace=True)

# Display the table
print(profile_statistics)
```

```
       customer_profiles    Avg_Age    Avg_DTI  Avg_CardSpendMonth  \
0         Bargain Hunter  37.878028   9.771763         3176.464745
1  Essentials Enthusiast  59.329936  10.116051         3384.895541
2      Essentials Hunter  37.805668   9.497571         3356.752632
3        Frugal Follower  54.334566   9.836969         3168.833087
4      Nomadic Navigator  37.711579  10.183158         3140.541263
5        Platinum Patron  55.788684  10.282679         3864.466397
6      Possibly Platinum  38.046729   9.564112         3401.353084
7            Savvy Saver  53.859574   9.760426         3713.005106

   Avg_EducationYears  Avg_HHIncome  Avg_total_debt  Male_Count  Female_Count
0           14.083542  38135.338346   369280.200501         578           619
1           14.239490  55943.949045   562165.987261         378           407
2           16.319838  50121.457490   470289.878543         118           129
3           13.192237  48243.992606   473614.787431         267           274
4           13.932632  43442.105263   451271.789474         245           230
5           15.256351  85459.584296   910848.383372         440           426
6           16.530841  58112.149533   554582.056075         272           263
7           13.468085  63582.978723   604208.510638         124           111
```

[27]:

```
# Decision Tree Segmentation
```

```
import numpy as np

from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.model_selection import GridSearchCV
```

[16]:
```
# Define the features (independent variables) and the target variable
features = ['Gender', 'Age', 'EducationYears', 'HHIncome', 'total_debt',
            'CardSpendMonth', 'VoiceLastMonth', 'EquipmentLastMonth',␣
↪'DataLastMonth']
target_variable = 'PhoneCoTenure'

# Encode binary variable ('Gender') using LabelEncoder
binary_features = ['Gender']
for feature in binary_features:
    le = LabelEncoder()
    df[feature] = le.fit_transform(df[feature])

# Split the data into features (X) and target variable (y)
X = df[features]
y = df[target_variable]

# Train the Decision Tree model
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X, y)

# Predict tenure categories for all customers
predicted_categories = decision_tree.predict(X)

# Map predicted categories to 'low', 'medium', 'high' based on the specified␣
↪thresholds
def map_to_tenure_category(prediction):
    if prediction <= 24:
        return 'low'
    elif 25 <= prediction <= 48:
        return 'medium'
    else:
        return 'high'

# Apply the mapping function to the predicted categories
predicted_categories_mapped = np.array([map_to_tenure_category(pred) for pred in␣
↪predicted_categories])


# Add a new column 'tenure_high_med_low' to the dataframe
```

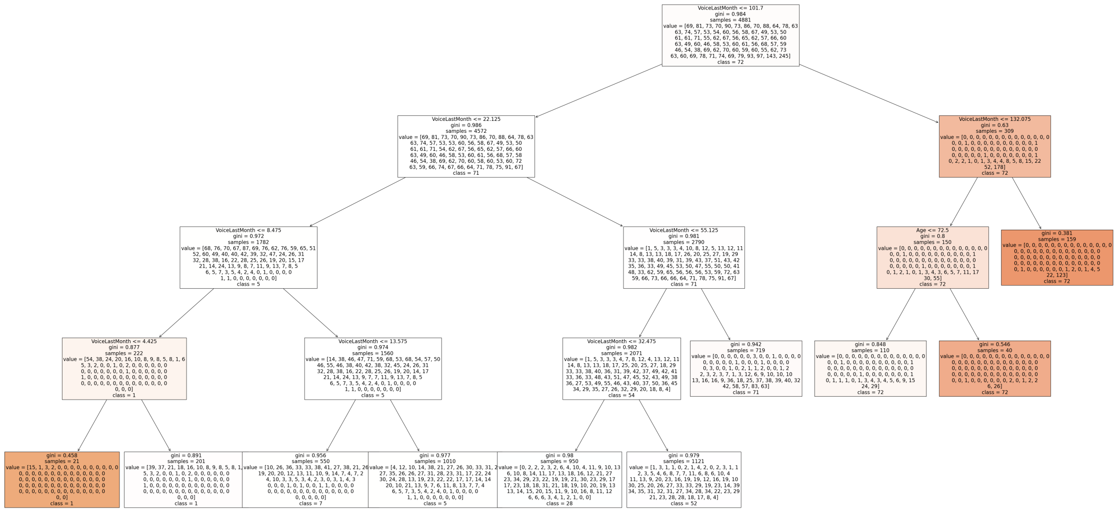
```
df['tenure_high_med_low'] = predicted_categories_mapped
```

[17]:
```
# List feature importances
feature_importances = decision_tree.feature_importances_
for i, feature in enumerate(features):
    print(f"{feature}: {feature_importances[i]}")
```

```
Gender: 0.042767331679083455
Age: 0.12433594506689107
EducationYears: 0.11166728011825425
HHIncome: 0.13480155626951532
total_debt: 0.15498377941188332
CardSpendMonth: 0.1549492037585316
VoiceLastMonth: 0.18109802092214677
EquipmentLastMonth: 0.055122028807606534
DataLastMonth: 0.04027485396608779
```

[18]:
```
# Check unique values in the target variable
unique_classes = np.unique(y)

# Ensure the class names provided to plot_tree match the unique classes
class_names = [str(cls) for cls in unique_classes]
```

[19]:
```
# Define the parameter grid for tuning
param_grid = {'ccp_alpha': np.linspace(0, 0.1, 100)}  # Vary the complexity␣
↪parameter alpha

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(),␣
↪param_grid=param_grid, cv=5)

# Fit the grid search to the data
grid_search.fit(X, y)

# Get the best estimator (pruned decision tree)
pruned_decision_tree = grid_search.best_estimator_

# Visualize the pruned decision tree
plt.figure(figsize=(64, 32))
plot_tree(pruned_decision_tree, feature_names=features, class_names=class_names,␣
↪filled=True, fontsize=16)
plt.show()
```



[20]:
```python
# Define the file path for the CSV file
csv_file_path = 'profile_statistics.csv'

# Export the DataFrame to a CSV file
df.to_csv(csv_file_path, index=False)

print("DataFrame exported to CSV file successfully.")
```

```
DataFrame exported to CSV file successfully.
```

[21]:
```python
# Print the first 25 rows of the DataFrame with the 'tenure_high_med_low' column
print(df[['Gender', 'Age', 'EducationYears', 'HHIncome', 'total_debt',
          'CardSpendMonth', 'VoiceLastMonth', 'EquipmentLastMonth',␣
 ↪'DataLastMonth', 'tenure_high_med_low']].head(25))
```

```
    Gender  Age  EducationYears  HHIncome  total_debt  CardSpendMonth  \
0        0   20              15     31000      344100           816.6
1        1   22              17     15000      279000           426.0
2        0   67              14     35000      346500          1842.2
3        1   23              16     20000      114000          3409.9
4        1   26              16     23000       39100          2551.0
5        1   64              17    107000      599200          2282.7
6        0   52              14     77000      146300          8223.2
7        0   44              16     97000     1396800          5927.0
8        0   66              12     16000       41600          3265.9
9        1   47              11     84000      344400          1996.4
10       0   59              19     47000      404200          4889.7
11       0   33               8     19000       17100          3382.6
12       1   44              10     73000      204400          5343.6
```

```
13        1   58            18     63000    661500          5935.0
14        0   72            20     17000    166600          2331.7
15        0   66            13     23000    213900          2974.7
16        0   57            17    171000   1624500          3059.4
17        1   63            14    424000   4536800          4957.5
18        0   28            11     23000    110400          4420.9
19        0   78            16     22000    334400            81.1
20        1   61            16     35000    353500          2719.8
21        1   70            17     28000    260400          2677.1
22        1   61            14     12000    258000          2450.3
23        1   37            11     29000    455300          5566.1
24        1   39            12    130000   1469000          4537.4

    VoiceLastMonth  EquipmentLastMonth  DataLastMonth tenure_high_med_low
0            19.50               29.50           0.00                 low
1            26.70               54.85          45.65              medium
2            85.20                0.00           0.00                high
3            18.00                0.00           0.00              medium
4             9.15                0.00          19.05                 low
5            24.30               35.50           0.00              medium
6            11.40                0.00           0.00                 low
7            44.55                0.00           0.00              medium
8            63.15                0.00           0.00                high
9            10.95                0.00           0.00                 low
10           25.65               31.20           0.00                high
11           10.80                0.00           0.00                 low
12           20.25                0.00           0.00                 low
13           36.30                0.00           0.00                high
14           41.70               38.55           0.00                high
15          116.10                0.00          43.25                high
16           12.45               43.15          40.20              medium
17          104.40                0.00           0.00                high
18           10.35                0.00           0.00              medium
19           21.90               24.85           0.00                 low
20           33.60                0.00           0.00                high
21           42.75                0.00           0.00                high
22           25.05               29.10           0.00                high
23           39.30               43.50          31.50              medium
24            4.50                0.00           0.00                 low
```

[22]:
```python
# Count the occurrences of 'low', 'medium', and 'high' in the␣
↪'tenure_high_med_low' column
tenure_counts = df['tenure_high_med_low'].value_counts()

# Print the counts
print("Counts of 'low', 'medium', and 'high' customers:")
print(tenure_counts)
```

```
Counts of 'low', 'medium', and 'high' customers:
high      1849
low       1599
medium    1433
Name: tenure_high_med_low, dtype: int64
```

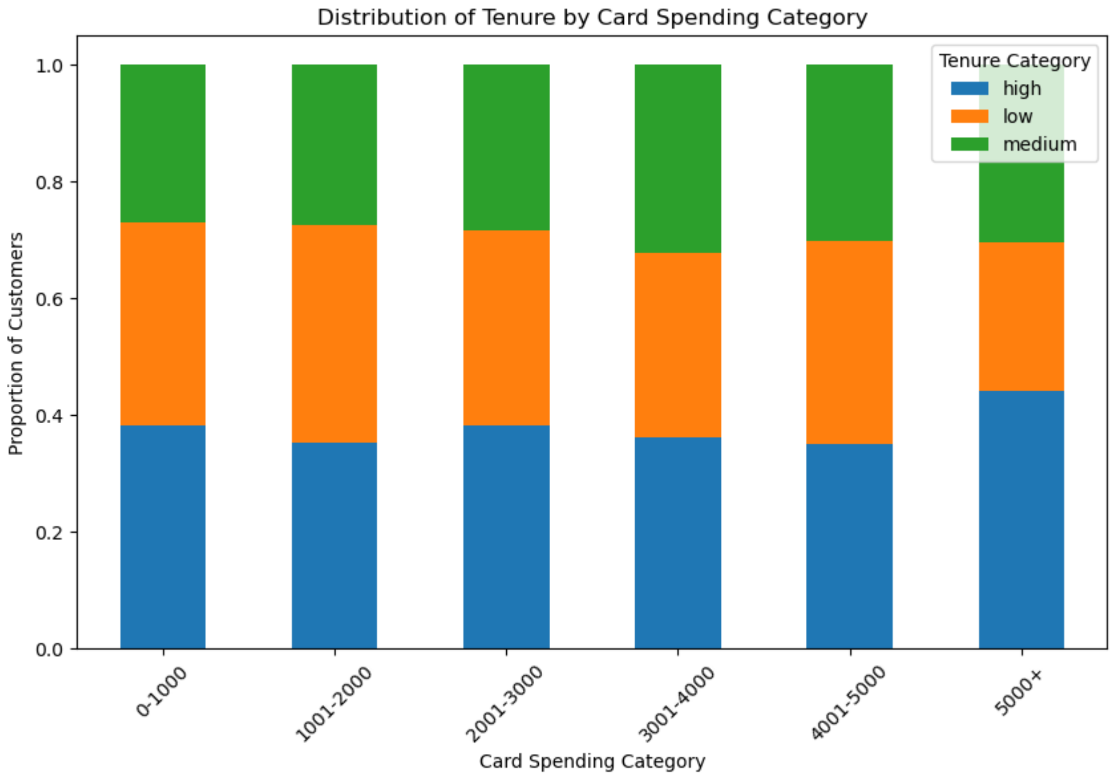
[23]:

```
# Group the DataFrame by 'tenure_high_med_low' and calculate the mean age for␣
↪each category
age_tenure_relation = df.groupby('tenure_high_med_low')['Age'].mean()

# Print the result
print("Average Age by Tenure Category:")
print(age_tenure_relation)
```

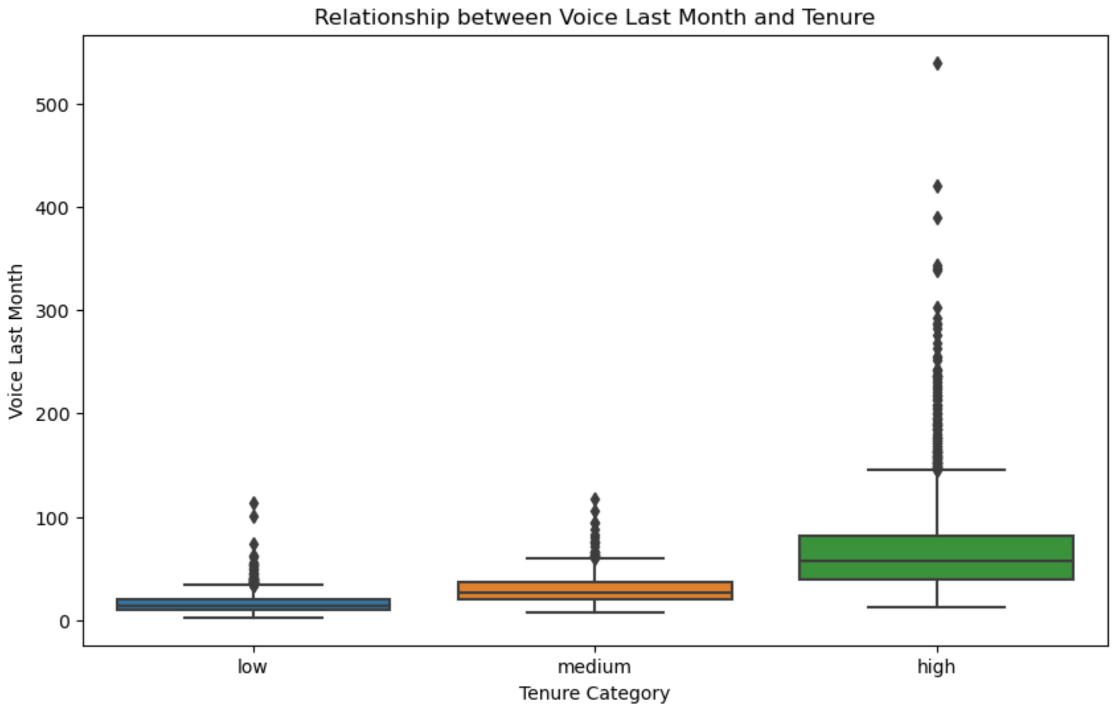
```
Average Age by Tenure Category:
tenure_high_med_low
high      59.072472
low       34.642902
medium    45.545010
Name: Age, dtype: float64
```

[24]:

```
# Define bins for CardSpendMonth
bins = [0, 1000, 2000, 3000, 4000, 5000, np.inf]
labels = ['0-1000', '1001-2000', '2001-3000', '3001-4000', '4001-5000', '5000+']

# Bin the CardSpendMonth data
df['CardSpendCategory'] = pd.cut(df['CardSpendMonth'], bins=bins, labels=labels,␣
↪right=False)

# Group the DataFrame by 'CardSpendCategory' and 'tenure_high_med_low' and␣
↪calculate the count of customers in each category
spend_tenure_relation = df.groupby(['CardSpendCategory', 'tenure_high_med_low']).
↪size().unstack(fill_value=0)

# Normalize the counts to get proportions
spend_tenure_relation = spend_tenure_relation.div(spend_tenure_relation.
↪sum(axis=1), axis=0)

# Plot the proportions
spend_tenure_relation.plot(kind='bar', stacked=True, figsize=(10, 6))
plt.title('Distribution of Tenure by Card Spending Category')
plt.xlabel('Card Spending Category')
plt.ylabel('Proportion of Customers')
plt.xticks(rotation=45)
plt.legend(title='Tenure Category')
plt.show()
```



[25]:
```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x='tenure_high_med_low', y='VoiceLastMonth', data=df)
plt.xlabel('Tenure Category')
plt.ylabel('Voice Last Month')
plt.title('Relationship between Voice Last Month and Tenure')
plt.show()
```


Relationship between Voice Last Month and Tenure
500
400
300
200
100
0
Voice Last Month
low
medium
high
Tenure Category