sunbase-assignment

September 27, 2023

0.1 Setup: Imports and Installations

[ ]:
```
!pip install catboost
```

```
Collecting catboost
  Downloading catboost-1.2.2-cp310-cp310-manylinux2014_x86_64.whl (98.7 MB)
                           98.7/98.7 MB
5.4 MB/s eta 0:00:00
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-
packages (from catboost) (0.20.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (from catboost) (3.7.1)
Requirement already satisfied: numpy>=1.16.0 in /usr/local/lib/python3.10/dist-
packages (from catboost) (1.23.5)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-
packages (from catboost) (1.5.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages
(from catboost) (1.11.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages
(from catboost) (5.15.0)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages
(from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-
packages (from pandas>=0.24->catboost) (2023.3.post1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.1.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->catboost) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.42.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (23.1)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-
packages (from matplotlib->catboost) (9.4.0)
```

```
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.1.1)
Requirement already satisfied: tenacity>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from plotly->catboost) (8.2.3)
Installing collected packages: catboost
Successfully installed catboost-1.2.2
```

[ ]:
```python
import numpy as np
import pandas as pd
pd.set_option('display.max_columns', None)

## Data visualization
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import missingno as msno # missing values visualization

## Stats
from statsmodels.graphics.gofplots import qqplot
from scipy.stats import shapiro
from scipy.stats import norm
from scipy.stats import zscore

## Preprocessing
from sklearn.model_selection import train_test_split as tts
from sklearn.preprocessing import StandardScaler

## Classes Imbalance
from imblearn.over_sampling import SMOTE

## Models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier, RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import VotingClassifier, StackingClassifier
from sklearn.ensemble import HistGradientBoostingClassifier

## Metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

## Time
import time
```

```
## Warnings
import warnings
warnings.filterwarnings('ignore')

# some more imports
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,classification_report
from sklearn.model_selection import RandomizedSearchCV
from sklearn.impute import SimpleImputer

import os
for dirname, _, filenames in os.walk('/content/drive/MyDrive/internwork'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

1 1 Data Preprocessing:

Load the provided dataset and perform initial data exploration. Handle missing data and outliers. Prepare the data for machine learning by encoding categorical variables and splitting it into training and testing sets.

[ ]:
```
df = pd.read_excel('/content/drive/MyDrive/internwork/
↪customer_churn_large_dataset.xlsx')
df
```

[ ]:
```
       CustomerID             Name  Age  Gender     Location  \
0               1       Customer_1   63    Male  Los Angeles
1               2       Customer_2   62  Female     New York
2               3       Customer_3   24  Female  Los Angeles
3               4       Customer_4   36  Female        Miami
4               5       Customer_5   46  Female        Miami
…             …                …  …     …            …
99995       99996   Customer_99996   33    Male      Houston
99996       99997   Customer_99997   62  Female     New York
99997       99998   Customer_99998   64    Male      Chicago
99998       99999   Customer_99999   51  Female     New York
99999      100000  Customer_100000   27  Female  Los Angeles

       Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn
0                              17         73.36             236      0
1                               1         48.76             172      0
```

```
2                                  5       85.47           460       0
3                                  3       97.94           297       1
4                                 19       58.14           266       0
…                              …        …           …     …
99995                             23       55.13           226       1
99996                             19       61.65           351       0
99997                             17       96.11           251       1
99998                             20       49.25           434       1
99999                             19       76.57           173       1

[100000 rows x 9 columns]
```

[ ]:
```
df.shape
```

[ ]:
```
(100000, 9)
```

[ ]:
```
# df.drop('CustomerID',axis=1,inplace=True)
# df.drop('Name',axis=1,inplace=True)
```

[ ]:
```
df.tail(2)
```

[ ]:
```
       CustomerID             Name  Age  Gender     Location  \
99998       99999   Customer_99999   51  Female     New York
99999      100000  Customer_100000   27  Female  Los Angeles

       Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn
99998                          20         49.25             434      1
99999                          19         76.57             173      1
```

Details of the dataset:

[ ]:
```
print(df.head())
print(df.info())
print(df.describe())
```

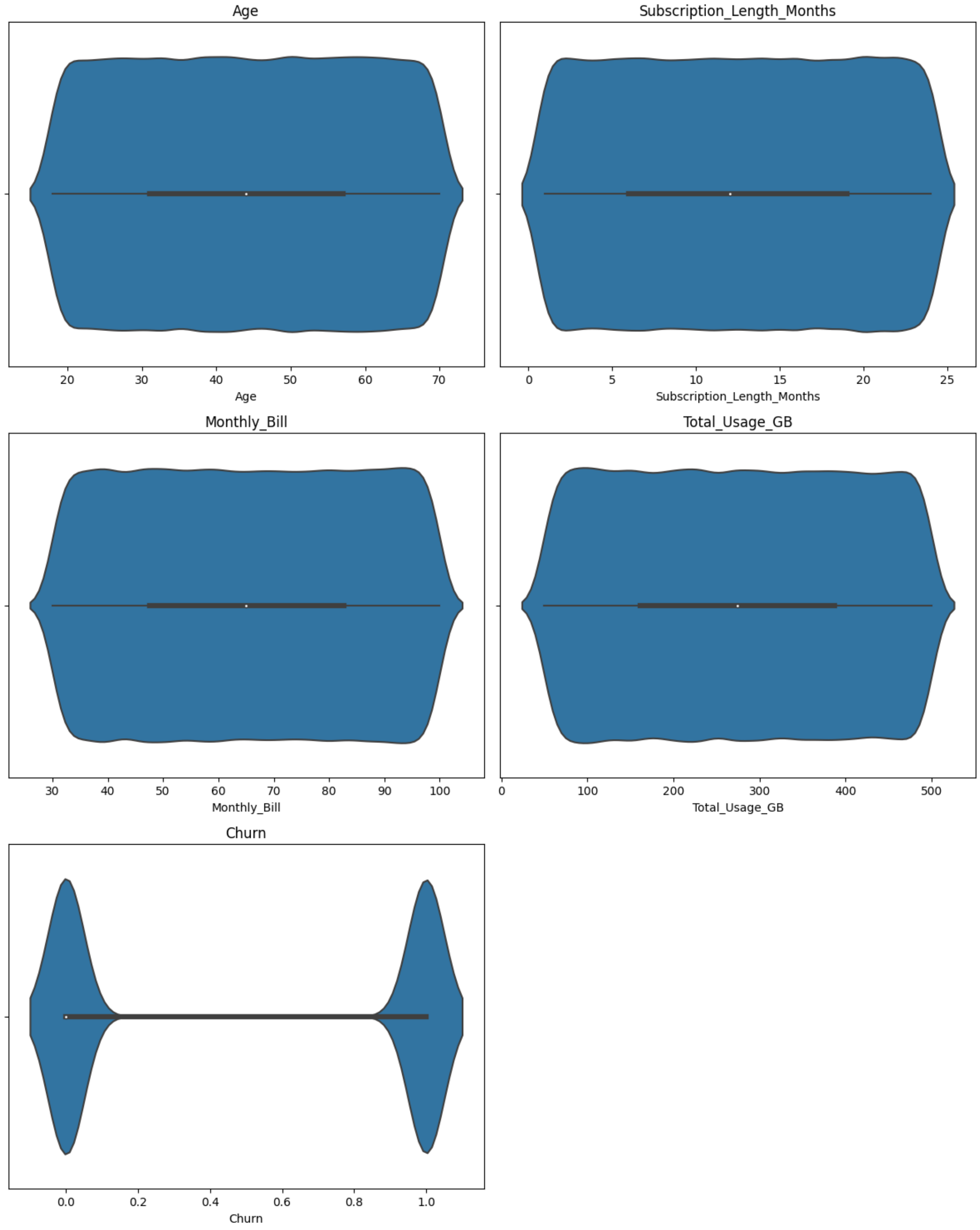
```
   CustomerID        Name  Age  Gender     Location  \
0           1  Customer_1   63    Male  Los Angeles
1           2  Customer_2   62  Female     New York
2           3  Customer_3   24  Female  Los Angeles
3           4  Customer_4   36  Female        Miami
4           5  Customer_5   46  Female        Miami

   Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn
0                          17         73.36             236      0
1                           1         48.76             172      0
2                           5         85.47             460      0
3                           3         97.94             297      1
4                          19         58.14             266      0
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 9 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   CustomerID                  100000 non-null  int64  
 1   Name                        100000 non-null  object 
 2   Age                         100000 non-null  int64  
 3   Gender                      100000 non-null  object 
 4   Location                    100000 non-null  object 
 5   Subscription_Length_Months  100000 non-null  int64  
 6   Monthly_Bill                100000 non-null  float64
 7   Total_Usage_GB              100000 non-null  int64  
 8   Churn                       100000 non-null  int64  
dtypes: float64(1), int64(5), object(3)
memory usage: 6.9+ MB
None
          CustomerID            Age  Subscription_Length_Months  \
count  100000.000000  100000.000000               100000.000000   
mean    50000.500000      44.027020                   12.490100   
std     28867.657797      15.280283                    6.926461   
min         1.000000      18.000000                    1.000000   
25%     25000.750000      31.000000                    6.000000   
50%     50000.500000      44.000000                   12.000000   
75%     75000.250000      57.000000                   19.000000   
max    100000.000000      70.000000                   24.000000   

        Monthly_Bill  Total_Usage_GB          Churn  
count  100000.000000   100000.000000  100000.000000  
mean       65.053197      274.393650       0.497790  
std        20.230696      130.463063       0.499998  
min        30.000000       50.000000       0.000000  
25%        47.540000      161.000000       0.000000  
50%        65.010000      274.000000       0.000000  
75%        82.640000      387.000000       1.000000  
max       100.000000      500.000000       1.000000  
```

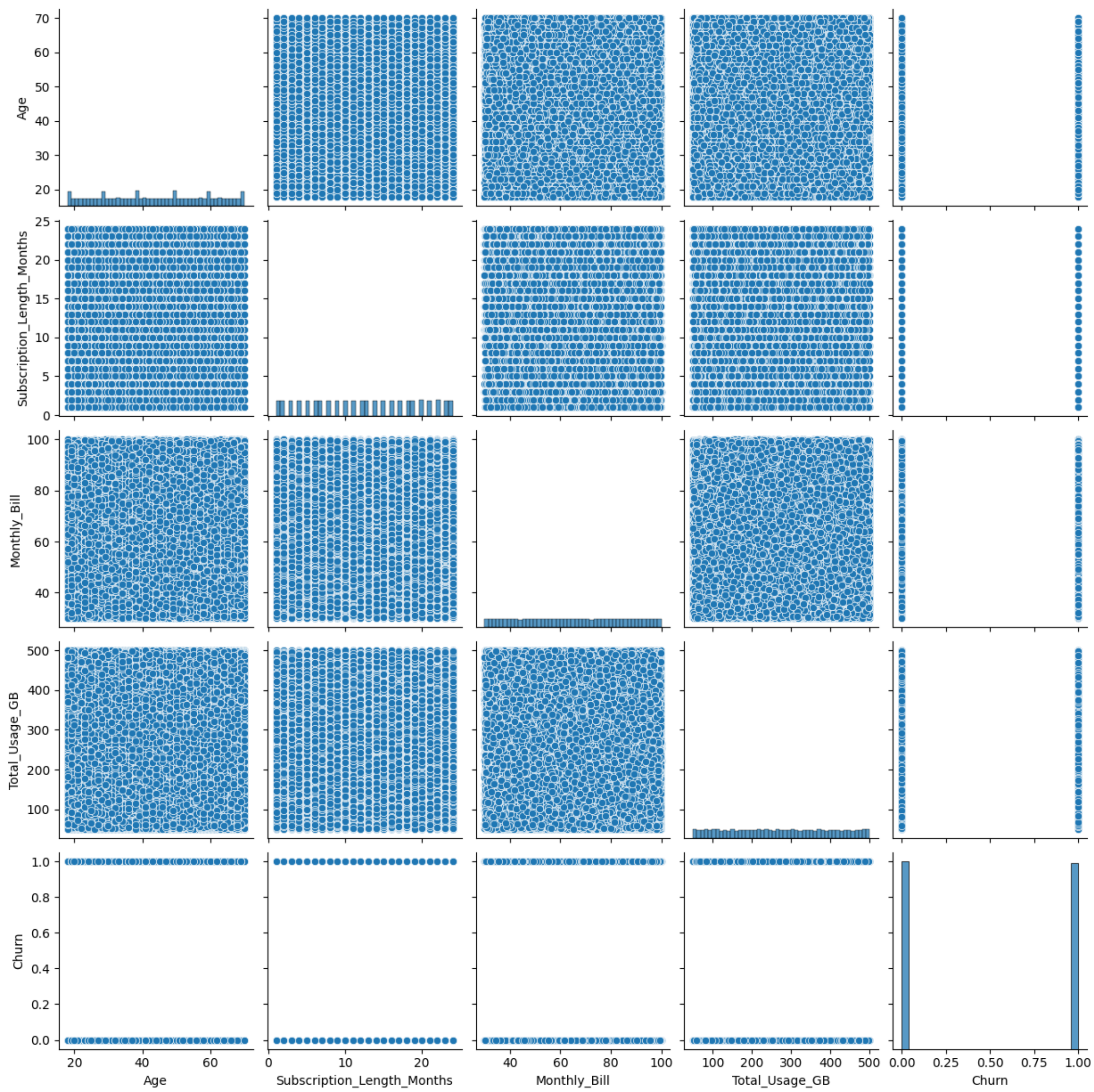
Violin Plots to detect any possible outliers, but on plotting them we are able to see that there are negligible outliers

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

# Assuming you have a DataFrame named 'df' with numerical columns
numerical_columns = df.select_dtypes(include=['number']).columns

# Set the number of columns per row in the subplot grid
num_plots_per_row = 2
```

```python
# Calculate the number of rows needed based on the number of numerical columns
num_rows = len(numerical_columns) // num_plots_per_row
if len(numerical_columns) % num_plots_per_row != 0:
    num_rows += 1

# Create subplots
fig, axes = plt.subplots(num_rows, num_plots_per_row, figsize=(12, 5 *
 ↪num_rows))

# Flatten the axes if there's only one row
if num_rows == 1:
    axes = axes.reshape(1, -1)

# Plot barplots for each numerical column
for i, column in enumerate(numerical_columns):
    row_idx = i // num_plots_per_row
    col_idx = i % num_plots_per_row

    ax = axes[row_idx, col_idx]

    sns.violinplot(data=df, x=column, ax=ax)
    ax.set_title(column)

# Remove empty subplots
for i in range(len(numerical_columns), num_rows * num_plots_per_row):
    fig.delaxes(axes.flatten()[i])

plt.tight_layout()
plt.show()
```




```
[ ]: # Data visualization (e.g., histograms, scatter plots)
sns.pairplot(df)
plt.show()
```



No missing values found

[ ]:
```
df.isnull().sum()
```

[ ]:
```
Age                           0
Gender                        0
Location                      0
Subscription_Length_Months    0
Monthly_Bill                  0
Total_Usage_GB                0
Churn                         0
Senior Citizen                0
Junior Citizen                0
```

```
Median Citizen              0
dtype: int64
```

[ ]:
```
df
```

[ ]:
```
       CustomerID             Name  Age  Gender     Location  \
0               1       Customer_1   63    Male  Los Angeles
1               2       Customer_2   62  Female     New York
2               3       Customer_3   24  Female  Los Angeles
3               4       Customer_4   36  Female        Miami
4               5       Customer_5   46  Female        Miami
…             …                …  …     …          …
99995       99996   Customer_99996   33    Male      Houston
99996       99997   Customer_99997   62  Female     New York
99997       99998   Customer_99998   64    Male      Chicago
99998       99999   Customer_99999   51  Female     New York
99999      100000  Customer_100000   27  Female  Los Angeles

       Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
0                              17         73.36             236      0
1                               1         48.76             172      0
2                               5         85.47             460      0
3                               3         97.94             297      1
4                              19         58.14             266      0
…                             …           …             …    …
99995                          23         55.13             226      1
99996                          19         61.65             351      0
99997                          17         96.11             251      1
99998                          20         49.25             434      1
99999                          19         76.57             173      1

       Senior Citizen  Junior Citizen  Median Citizen SubscriptionCategory
0                   1               0               0            Two Years
1                   1               0               0            One Month
2                   0               0               1             One Year
3                   0               0               1             One Year
4                   0               0               1            Two Years
…                 …             …             …                  …
99995               0               0               1            Two Years
99996               1               0               0            Two Years
99997               1               0               0            Two Years
99998               1               0               0            Two Years
99999               0               0               1            Two Years

[100000 rows x 13 columns]
```

[ ]:
```
df_encoded.shape
```

[ ]: (100000, 15)

[ ]:
```
df_encoded
```

[ ]:

```
       Age  Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
0       63                          17         73.36             236      0
1       62                           1         48.76             172      0
2       24                           5         85.47             460      0
3       36                           3         97.94             297      1
4       46                          19         58.14             266      0
…      …                          …           …             …    …
99995   33                          23         55.13             226      1
99996   62                          19         61.65             351      0
99997   64                          17         96.11             251      1
99998   51                          20         49.25             434      1
99999   27                          19         76.57             173      1

       Senior Citizen  Junior Citizen  Median Citizen  Gender_Female  \
0                   1               0               0              0
1                   1               0               0              1
2                   0               0               1              1
3                   0               0               1              1
4                   0               0               1              1
…                 …               …               …            …
99995               0               0               1              0
99996               1               0               0              1
99997               1               0               0              0
99998               1               0               0              1
99999               0               0               1              1

       Gender_Male  Location_Chicago  Location_Houston  Location_Los Angeles  \
0                1                 0                 0                     1
1                0                 0                 0                     0
2                0                 0                 0                     1
3                0                 0                 0                     0
4                0                 0                 0                     0
…             …                 …                 …                   …
99995            1                 0                 1                     0
99996            0                 0                 0                     0
99997            1                 1                 0                     0
99998            0                 0                 0                     0
99999            0                 0                 0                     1

       Location_Miami  Location_New York
0                   0                  0
1                   0                  1
2                   0                  0
```

```
3                    1                   0
4                    1                   0
…                 …                 …
99995                0                   0
99996                0                   1
99997                0                   0
99998                0                   1
99999                0                   0

[100000 rows x 15 columns]
```

Outlier Removal is not needed yet but can be donw with z score method by uncommenting this:

```
[ ]: # z_scores.describe()
     # # Example: Remove outliers using Z-score
     # z_scores = zscore(df_encoded)
     # df_no_outliers = df_encoded[(abs(z_scores) < 3).all(axis=1)]
     # df_no_outliers
```

We could say looking at the above result that this doesnt have outliers

Normalisation did not have a lot of effect her on the dataset

We will now do encoding of categorical variables in our data

1.0.1 Encoding

```
[ ]: from sklearn.preprocessing import LabelEncoder

     # Create a LabelEncoder instance
     label_encoder = LabelEncoder()
     # Apply label encoding to the 'Age_Group_Indicator' column
     df['Age_Group_Indicator'] = label_encoder.
      ↪fit_transform(df['Age_Group_Indicator'])
     df = pd.get_dummies(df, columns=['Gender','Location'])
```

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

```
[ ]:    CustomerID        Name  Age  Subscription_Length_Months  Monthly_Bill  \
     0           1  Customer_1   63                          17         73.36
     1           2  Customer_2   62                           1         48.76
     2           3  Customer_3   24                           5         85.47
     3           4  Customer_4   36                           3         97.94
     4           5  Customer_5   46                          19         58.14

        Total_Usage_GB  Churn  Average_Monthly_Data_Usage  Billing_Change_Rate  \
     0             236      0                   13.882353                  NaN
     1             172      0                  172.000000               -24.60
```

```
2             460      0                    92.000000                36.71
3             297      1                    99.000000                12.47
4             266      0                    14.000000               -39.80

   Billing_As_Percentage  Customer_Tenure_Months  Churn_History  \
0             112.769247                      17            NaN
1              74.954041                       1            0.0
2             131.384781                       5            0.0
3             150.553708                       3            0.0
4              89.373010                      19            1.0

   Age_Group_Indicator  Remaining_Subscription_Length  Average_Bill_Change  \
0                    1                             17                  NaN
1                    1                              0                  NaN
2                    2                              3                  NaN
3                    0                              0             8.193333
4                    0                             15             3.126667

   Gender_Female  Gender_Male  Location_Chicago  Location_Houston  \
0              0            1                 0                 0
1              1            0                 0                 0
2              1            0                 0                 0
3              1            0                 0                 0
4              1            0                 0                 0

   Location_Los Angeles  Location_Miami  Location_New York
0                     1               0                  0
1                     0               0                  1
2                     1               0                  0
3                     0               1                  0
4                     0               1                  0
```

[ ]:
```
df.describe()
```

[ ]:
```
          CustomerID            Age  Subscription_Length_Months  \
count  100000.000000  100000.000000               100000.000000
mean    50000.500000      44.027020                   12.490100
std     28867.657797      15.280283                    6.926461
min         1.000000      18.000000                    1.000000
25%     25000.750000      31.000000                    6.000000
50%     50000.500000      44.000000                   12.000000
75%     75000.250000      57.000000                   19.000000
max    100000.000000      70.000000                   24.000000

        Monthly_Bill  Total_Usage_GB          Churn  \
count  100000.000000   100000.000000  100000.000000
mean       65.053197      274.393650       0.497790
```

```
std        20.230696      130.463063       0.499998
min        30.000000       50.000000       0.000000
25%        47.540000      161.000000       0.000000
50%        65.010000      274.000000       0.000000
75%        82.640000      387.000000       1.000000
max       100.000000      500.000000       1.000000

       Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
count               100000.000000         99999.000000          100000.000000
mean                    43.349682             0.000032             100.000000
std                     65.786483            28.588777              31.098696
min                      2.083333           -69.910000              46.116104
25%                     12.687500           -20.680000              73.078653
50%                     21.909091            -0.030000              99.933598
75%                     42.545455            20.580000             127.034495
max                    500.000000            69.460000             153.720347

       Customer_Tenure_Months  Churn_History  Age_Group_Indicator  \
count           100000.000000   99999.000000        100000.000000
mean                12.490100       0.497785             0.866270
std                  6.926461       0.499998             0.777375
min                  1.000000       0.000000             0.000000
25%                  6.000000       0.000000             0.000000
50%                 12.000000       0.000000             1.000000
75%                 19.000000       1.000000             1.000000
max                 24.000000       1.000000             2.000000

       Remaining_Subscription_Length  Average_Bill_Change  Gender_Female  \
count                  100000.000000         99997.000000  100000.000000
mean                   -49987.009900             0.000048       0.502160
std                     28867.620918             9.551407       0.499998
min                    -99986.000000           -23.286667       0.000000
25%                    -74986.250000            -6.856667       0.000000
50%                    -49988.000000            -0.020000       1.000000
75%                    -24985.250000             6.876667       1.000000
max                        17.000000            23.166667       1.000000

         Gender_Male  Location_Chicago  Location_Houston  \
count  100000.000000     100000.000000     100000.000000
mean        0.497840          0.199580          0.201570
std         0.499998          0.399687          0.401175
min         0.000000          0.000000          0.000000
25%         0.000000          0.000000          0.000000
50%         0.000000          0.000000          0.000000
75%         1.000000          0.000000          0.000000
max         1.000000          1.000000          1.000000
```

```
       Location_Los Angeles  Location_Miami  Location_New York
count         100000.000000   100000.000000      100000.000000
mean               0.200410        0.200310           0.198130
std                0.400309        0.400234           0.398593
min                0.000000        0.000000           0.000000
25%                0.000000        0.000000           0.000000
50%                0.000000        0.000000           0.000000
75%                0.000000        0.000000           0.000000
max                1.000000        1.000000           1.000000
```

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

# Define your target variable and features
X = df.drop('Churn', axis=1)
y = df['Churn']
```

[ ]:
```
from sklearn.preprocessing import MinMaxScaler

# # # Create a MinMaxScaler object
scaler = MinMaxScaler()

# # # Fit the scaler to your data and transform it
X_normalized = scaler.fit_transform(X)
```

Various methods with/without normalisation and with/without extra features were tried and tested here

[ ]:
```
# Without Normalisation
X_train2, X_test2, y_train2, y_test2 = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[ ]:
```
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
X_train3, X_test3, y_train3, y_test3 = train_test_split(X_imputed, y,␣
↪test_size=0.2, random_state=42)
```

[ ]:
```
# Split the data into training and testing sets (e.g., 80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,␣
↪test_size=0.2, random_state=42)
```

[ ]:
```
X_train[0]
```

[ ]:
```
array([0.69230769, 0.17391304, 0.77857143, 0.34444444, 1.        ,
       0.        , 0.        , 1.        , 0.        , 0.        ,
       0.        , 0.        , 0.        , 1.        ])
```

2 2 Feature Engineering:

- Generate relevant features from the dataset that can help improve the model's prediction accuracy.
- Apply feature scaling or normalization if necessary.

[ ]:
```
import pandas as pd

# Assuming you have a DataFrame 'df' with the original features: Age,␣
↪Subscription_Length_Months, Monthly_Bill, Total_Usage_GB
# Replace 'your_dataset.csv' with the path to your dataset
# df = pd.read_csv('your_dataset.csv')

# Feature Engineering

# 1. Average Monthly Data Usage
df['Average_Monthly_Data_Usage'] = df['Total_Usage_GB'] /␣
↪df['Subscription_Length_Months']

# 2. Billing Change Rate
df['Billing_Change_Rate'] = df['Monthly_Bill'].diff()

# 3. Billing Amount as a Percentage
df['Billing_As_Percentage'] = (df['Monthly_Bill'] / df['Monthly_Bill'].mean())␣
↪* 100

# 4. Customer Tenure in Months
df['Customer_Tenure_Months'] = df['Subscription_Length_Months']

# 5. Churn History (Assuming 'Churn' is a binary column indicating churn␣
↪history)
df['Churn_History'] = df['Churn'].shift(1)  # Lagged version of the churn column

# 6. Age Group Indicator (Assuming age groups are defined)
age_bins = [0, 30, 50, 100]  # Define your age groups as needed
age_labels = ['Young', 'Middle-Aged', 'Senior']
df['Age_Group_Indicator'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels)

# 7. Remaining Subscription Length
df['Remaining_Subscription_Length'] = df['Subscription_Length_Months'] - df.
↪index

# 8. Average Bill Change
df['Average_Bill_Change'] = df['Billing_Change_Rate'].rolling(window=3).mean()

# Display the updated DataFrame with engineered features
print(df.head())
```

```
   CustomerID        Name  Age  Gender     Location  \
0           1  Customer_1   63    Male  Los Angeles
1           2  Customer_2   62  Female     New York
2           3  Customer_3   24  Female  Los Angeles
3           4  Customer_4   36  Female        Miami
4           5  Customer_5   46  Female        Miami

   Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
0                          17         73.36             236      0
1                           1         48.76             172      0
2                           5         85.47             460      0
3                           3         97.94             297      1
4                          19         58.14             266      0

   Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
0                   13.882353                  NaN             112.769247
1                  172.000000               -24.60              74.954041
2                   92.000000                36.71             131.384781
3                   99.000000                12.47             150.553708
4                   14.000000               -39.80              89.373010

   Customer_Tenure_Months  Churn_History Age_Group_Indicator  \
0                      17            NaN              Senior
1                       1            0.0              Senior
2                       5            0.0               Young
3                       3            0.0         Middle-Aged
4                      19            1.0         Middle-Aged

   Remaining_Subscription_Length  Average_Bill_Change
0                             17                  NaN
1                              0                  NaN
2                              3                  NaN
3                              0             8.193333
4                             15             3.126667
```

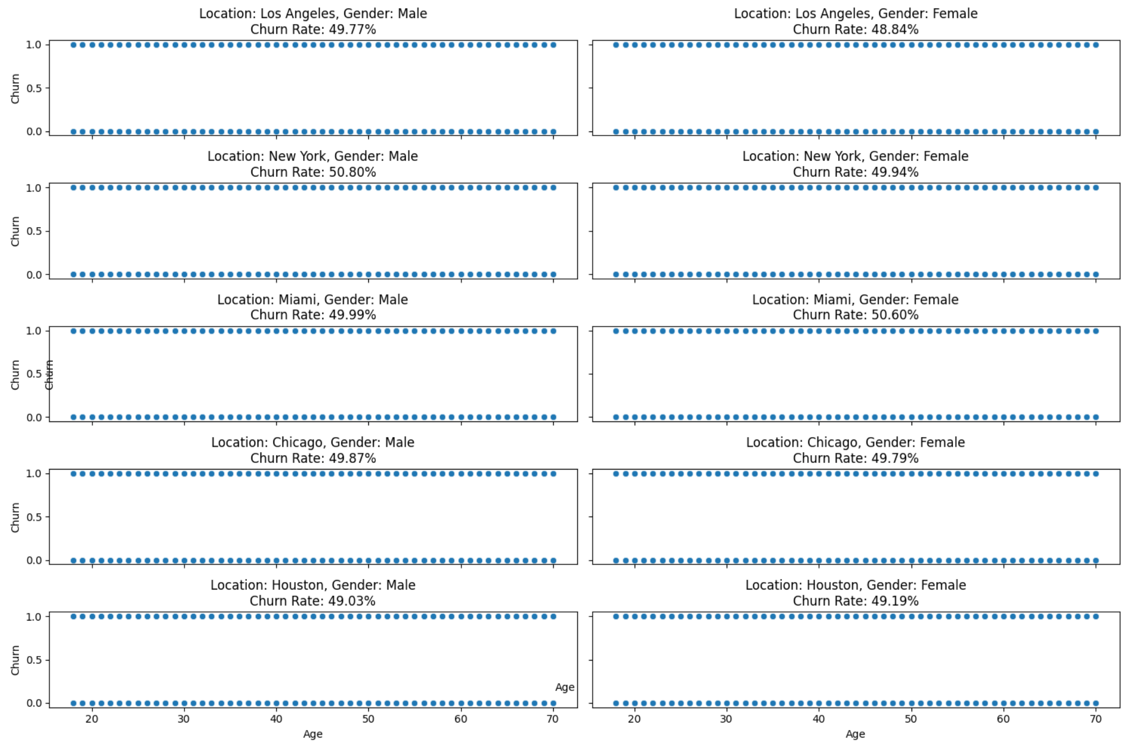
[ ]:
```
df.drop(columns=['Name','CustomerID'],axis=1,inplace=True)
```

[ ]:
```
df.columns
```

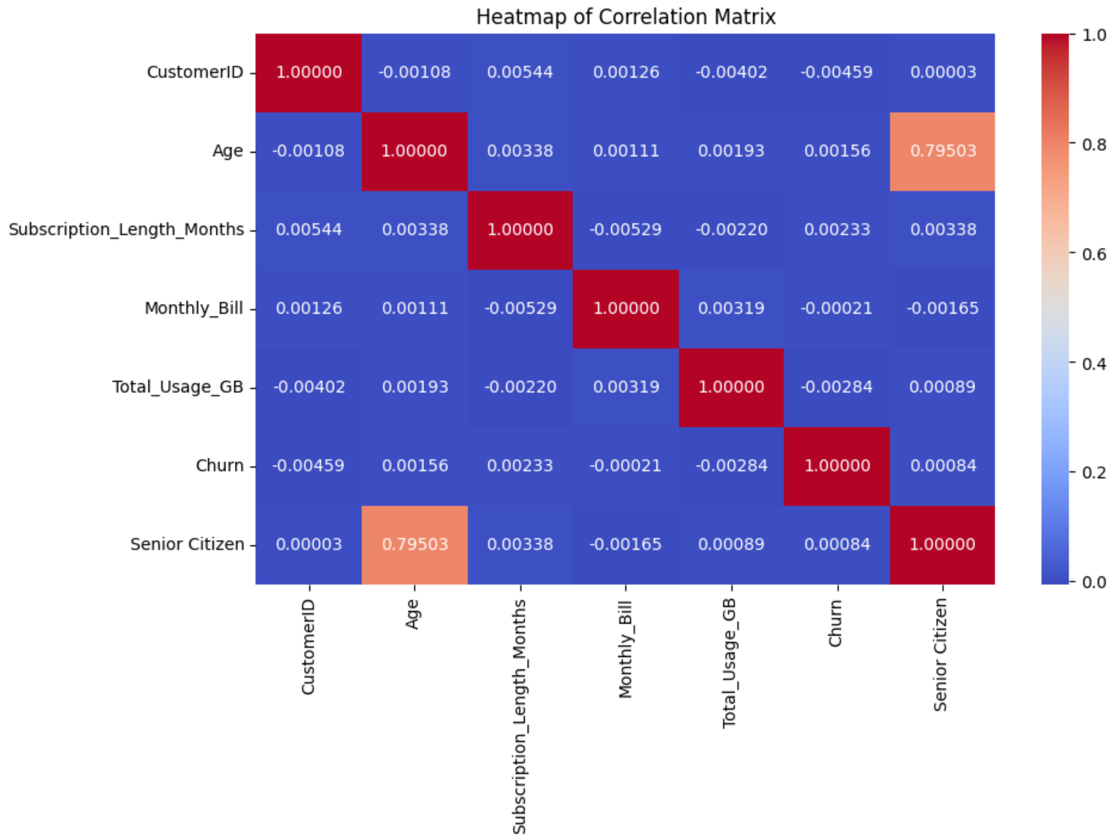
[ ]:
```
Index(['Age', 'Gender', 'Location', 'Subscription_Length_Months',
       'Monthly_Bill', 'Total_Usage_GB', 'Churn', 'Average_Monthly_Data_Usage',
       'Billing_Change_Rate', 'Billing_As_Percentage',
       'Customer_Tenure_Months', 'Churn_History', 'Age_Group_Indicator',
       'Remaining_Subscription_Length', 'Average_Bill_Change'],
      dtype='object')
```

2.0.1 Graphical Analysis

[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Filter the data by city and gender
cities = df['Location'].unique()
genders = df['Gender'].unique()

# Create subplots for different cities and genders
fig, axes = plt.subplots(len(cities), len(genders), figsize=(15, 10),␣
↪sharex=True, sharey=True)

for i, city in enumerate(cities):
    for j, gender in enumerate(genders):
        # Filter data for the current city and gender
        subset = df[(df['Location'] == city) & (df['Gender'] == gender)]

        # Calculate churn rate for the subset
        churn_rate = subset['Churn'].mean()

        # Create a scatter plot
        sns.scatterplot(data=subset, x='Age', y='Churn', ax=axes[i, j])
        axes[i, j].set_title(f'Location: {city}, Gender: {gender}\nChurn Rate:␣
↪{churn_rate:.2%}')

# Add labels and a common y-axis label
fig.text(0.5, 0.08, 'Age', ha='center')
fig.text(0.04, 0.5, 'Churn', va='center', rotation='vertical')

# Adjust subplot spacing
plt.tight_layout()

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

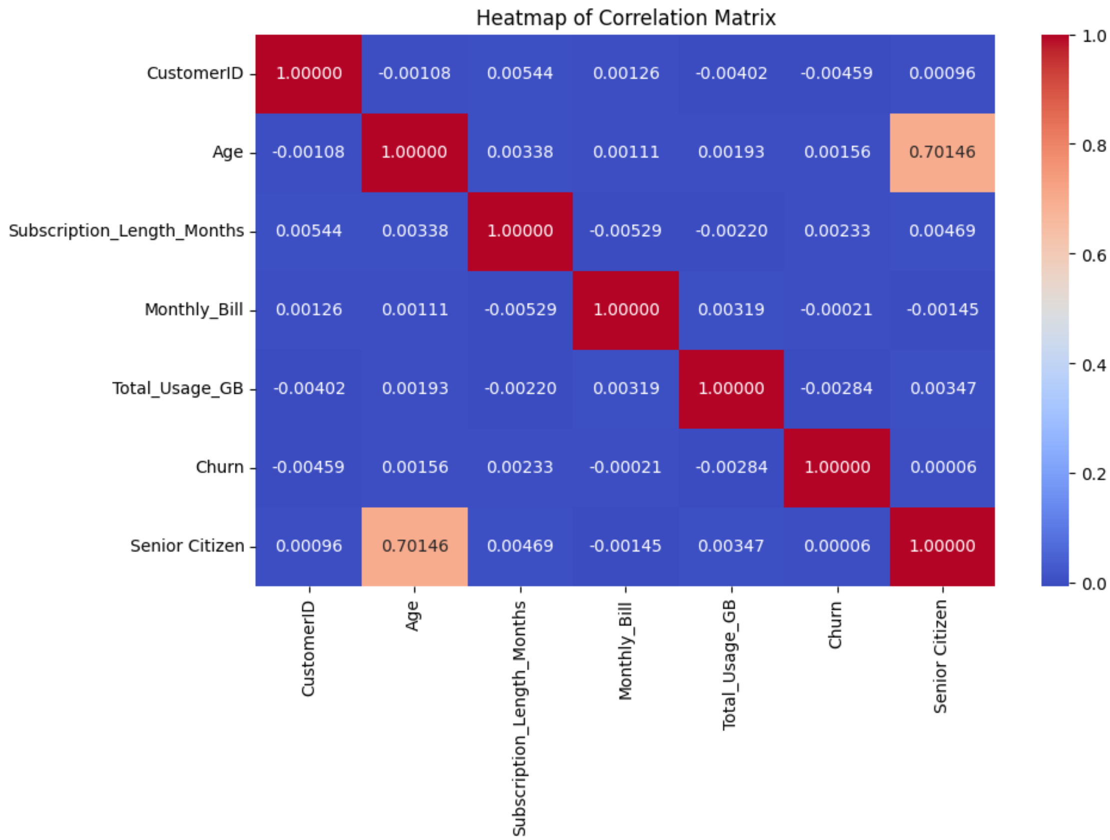


[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
senior_age_threshold = 55

# Create a new column 'Senior Citizen' based on age
df['Senior Citizen'] = df['Age'] >= senior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and␣
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



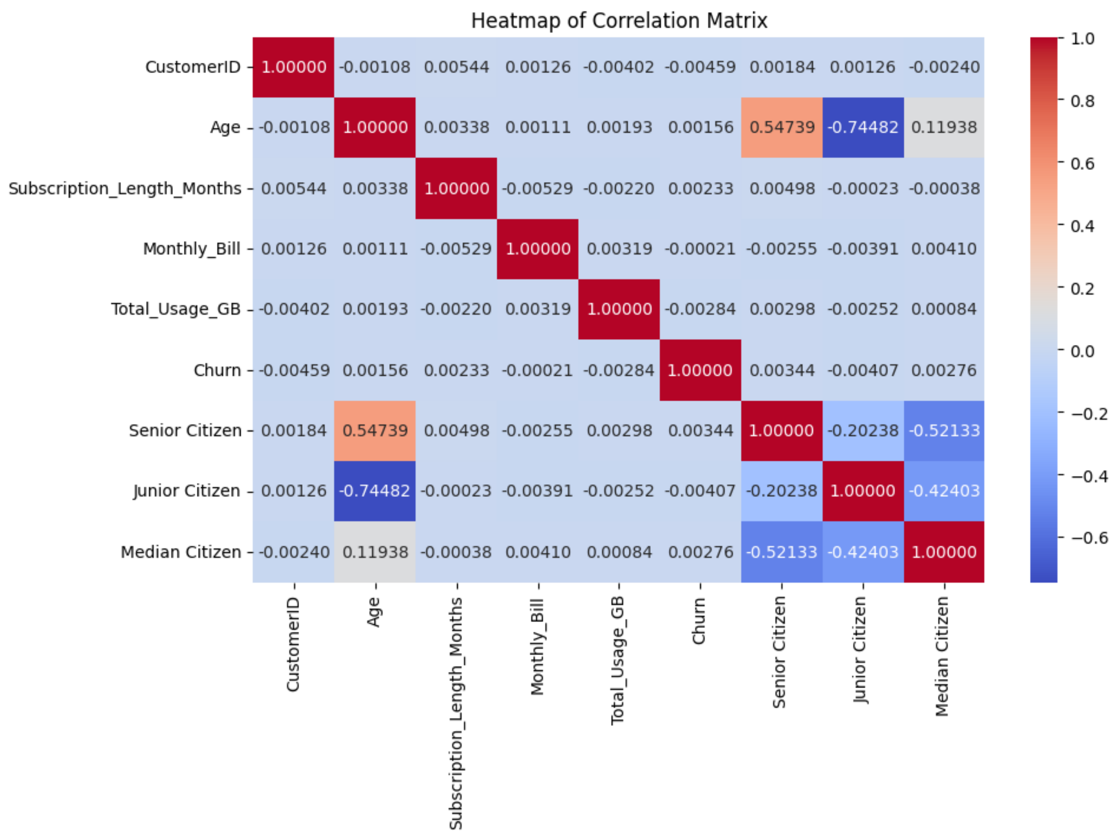
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
senior_age_threshold = 60

# Create a new column 'Senior Citizen' based on age
df['Senior Citizen'] = df['Age'] >= senior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and␣
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
```

```
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



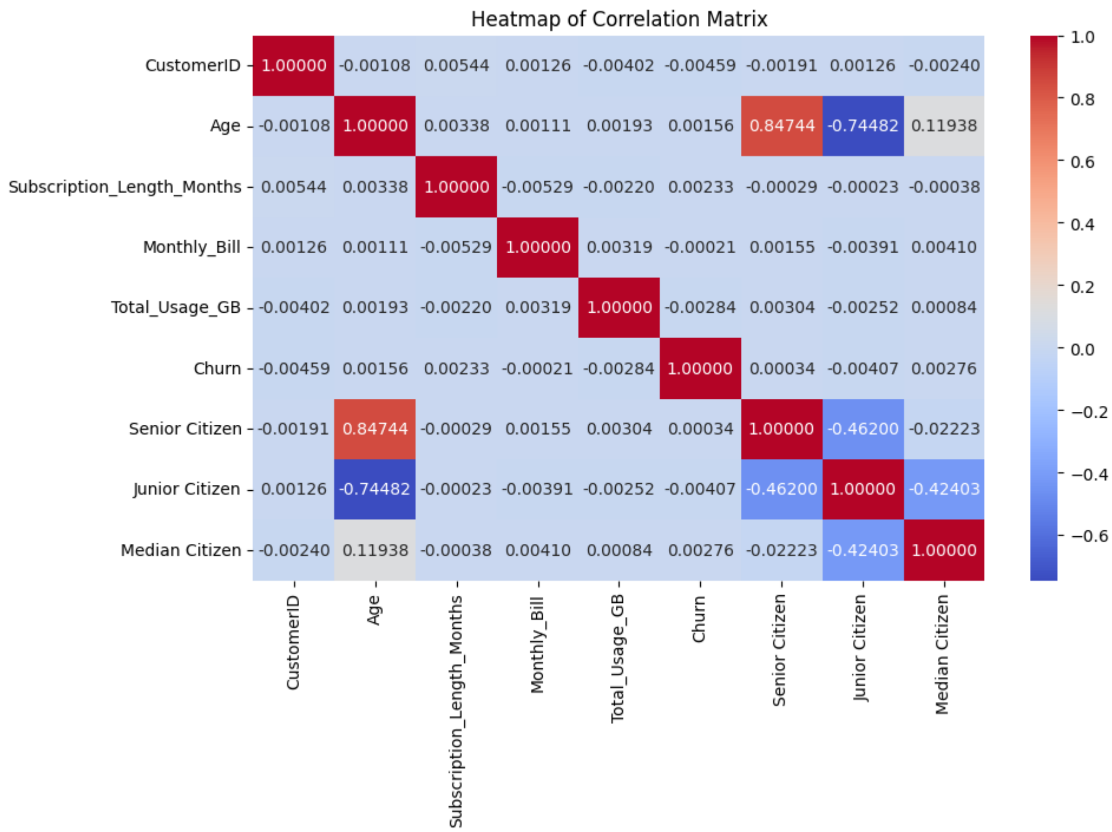
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
senior_age_threshold = 65

# Create a new column 'Senior Citizen' based on age
df['Senior Citizen'] = df['Age'] >= senior_age_threshold
```

```
# Create a heatmap to visualize the relationship between 'Senior Citizen' and
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



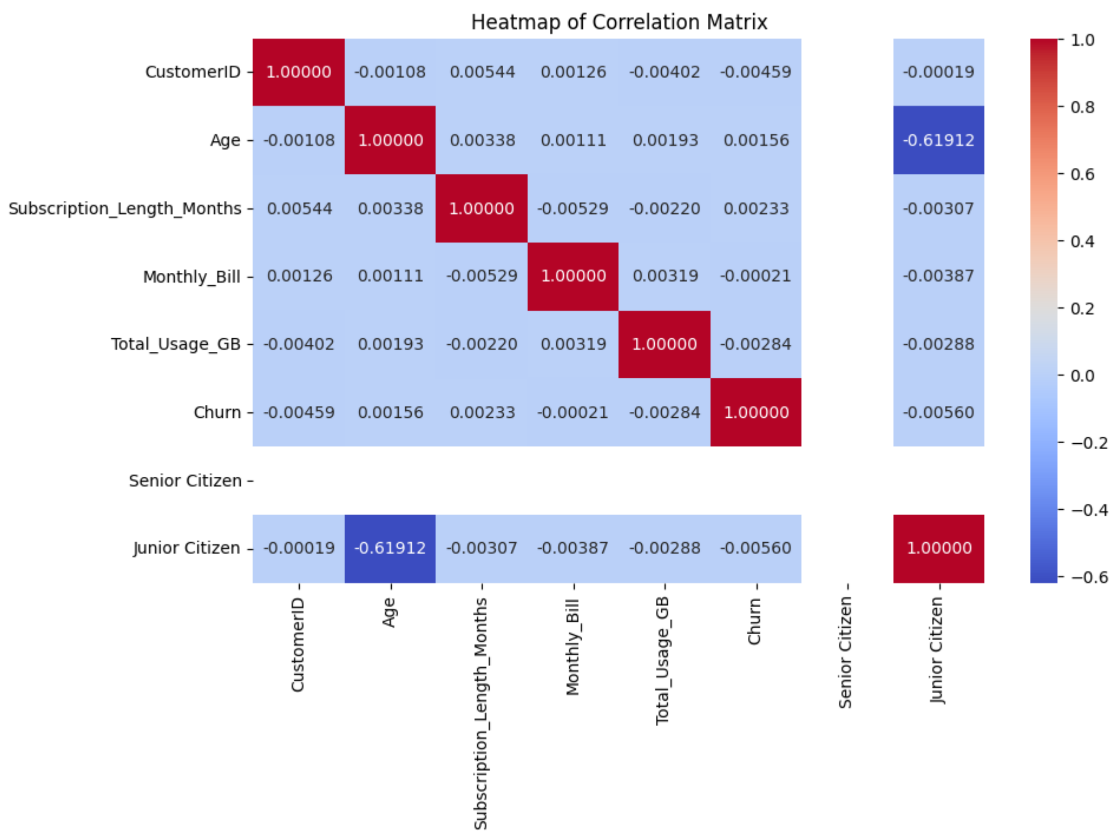

```
[ ]: import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
senior_age_threshold = 50
```

```python
# Create a new column 'Senior Citizen' based on age
df['Senior Citizen'] = df['Age'] >= senior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



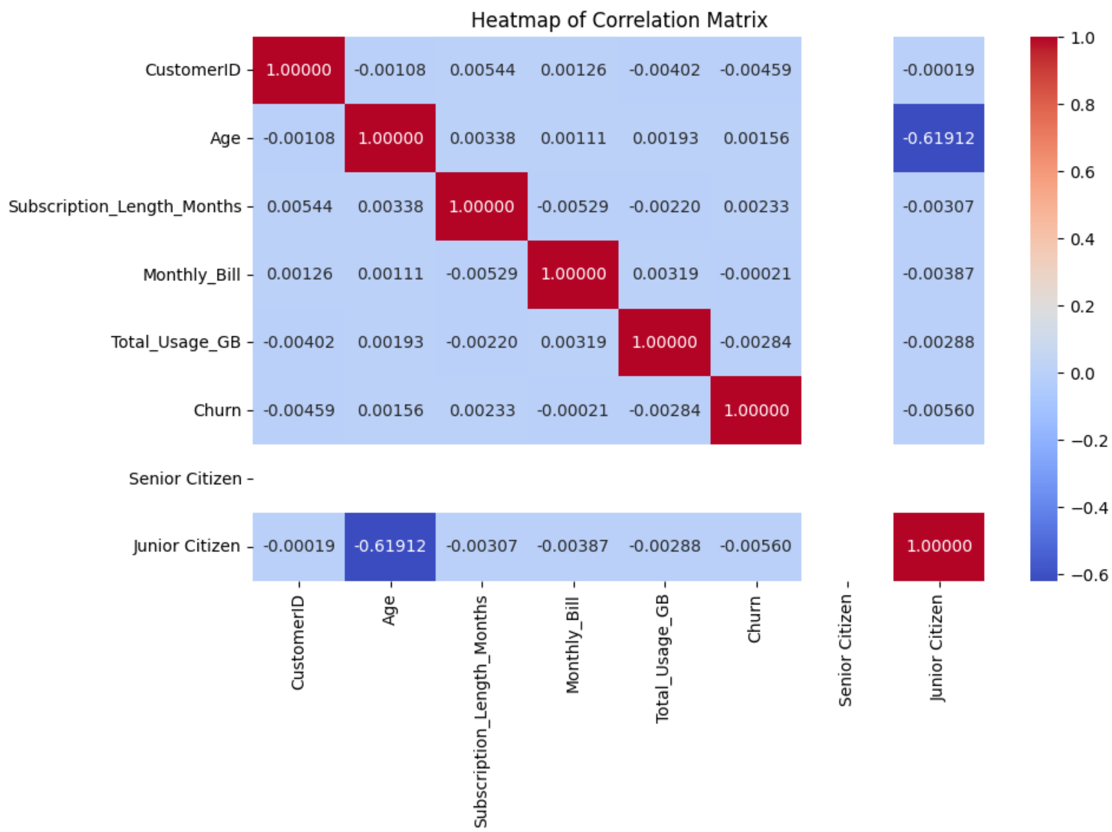
[ ]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'
```

```
# Define the age threshold for senior citizens
junior_age_threshold = 25

# Create a new column 'Senior Citizen' based on age
df['Junior Citizen'] = df['Age'] <= junior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



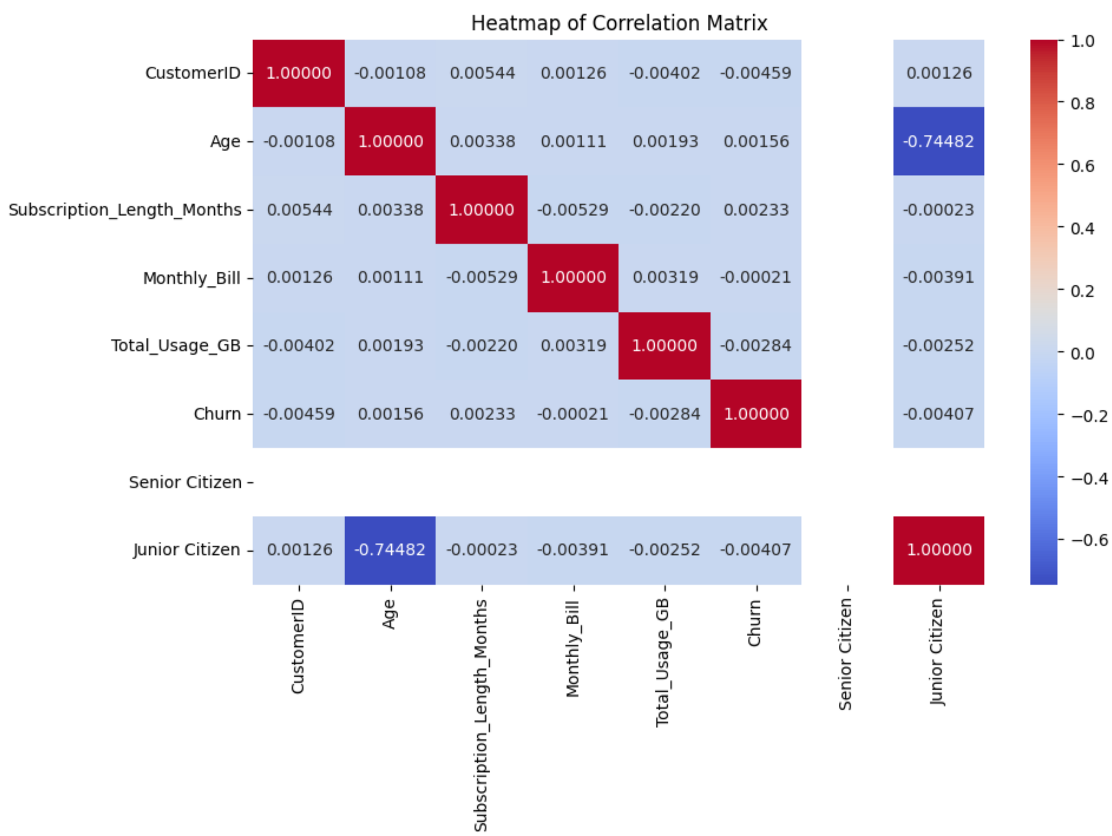
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with your dataset file path
```

```
# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
junior_age_threshold = 25

# Create a new column 'Senior Citizen' based on age
df['Junior Citizen'] = df['Age'] <= junior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and␣
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



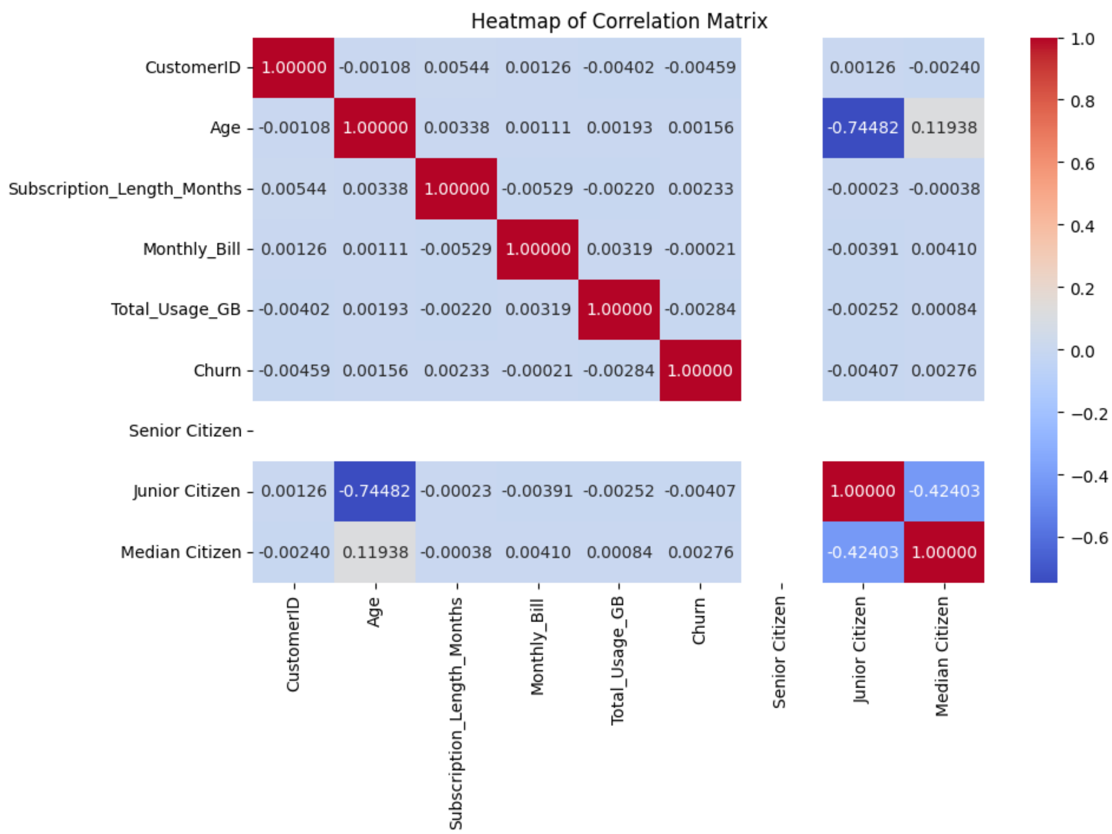
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
```

```
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
junior_age_threshold = 30

# Create a new column 'Senior Citizen' based on age
df['Junior Citizen'] = df['Age'] <= junior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and␣
↪other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```

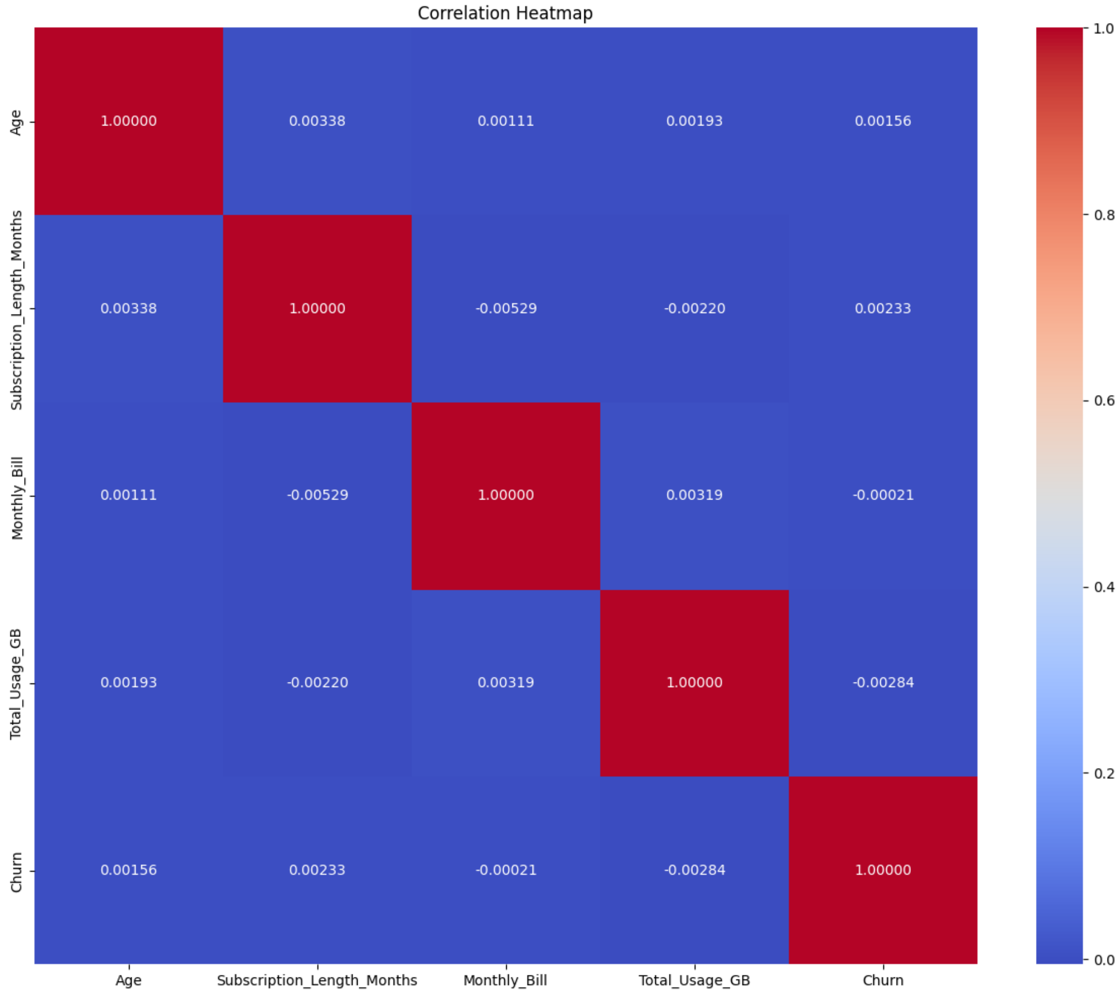


[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
high_age_threshold = 65
low_age_threshold = 25

# Create a new column 'Senior Citizen' based on age
# df['Median Citizen'] = df[low_age_threshold <= df['Age'] <= high_age_threshold]

df['Median Citizen'] = df['Age'].apply(lambda age: high_age_threshold >=age >= low_age_threshold)


# Create a heatmap to visualize the relationship between 'Senior Citizen' and other variables
plt.figure(figsize=(10, 6))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



[ ]:
```python
correlation_matrix = pd.DataFrame(df,columns=df.columns).corr()

# Create a heatmap
plt.figure(figsize=(15, 12))  # Adjust the figure size as needed
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Correlation Heatmap')
plt.show()
```



Surprisingly we see we do not have multicollinearity amongst our features

more usage => more bill

more usage => less churn

more bill => less churn

age => increase in churn

[ ]:
```
# df['Senior Citizen'] = df['Age'] >= 50
# df['Junior Citizen'] = df['Age'] <= 20
# df['Median Citizen'] = df['Age'].apply(lambda age: 50 >= age >= 20)

# # Convert boolean columns to integers (0 or 1)
# df['Senior Citizen'] = df['Senior Citizen'].astype(int)
# df['Junior Citizen'] = df['Junior Citizen'].astype(int)
# df['Median Citizen'] = df['Median Citizen'].astype(int)
```

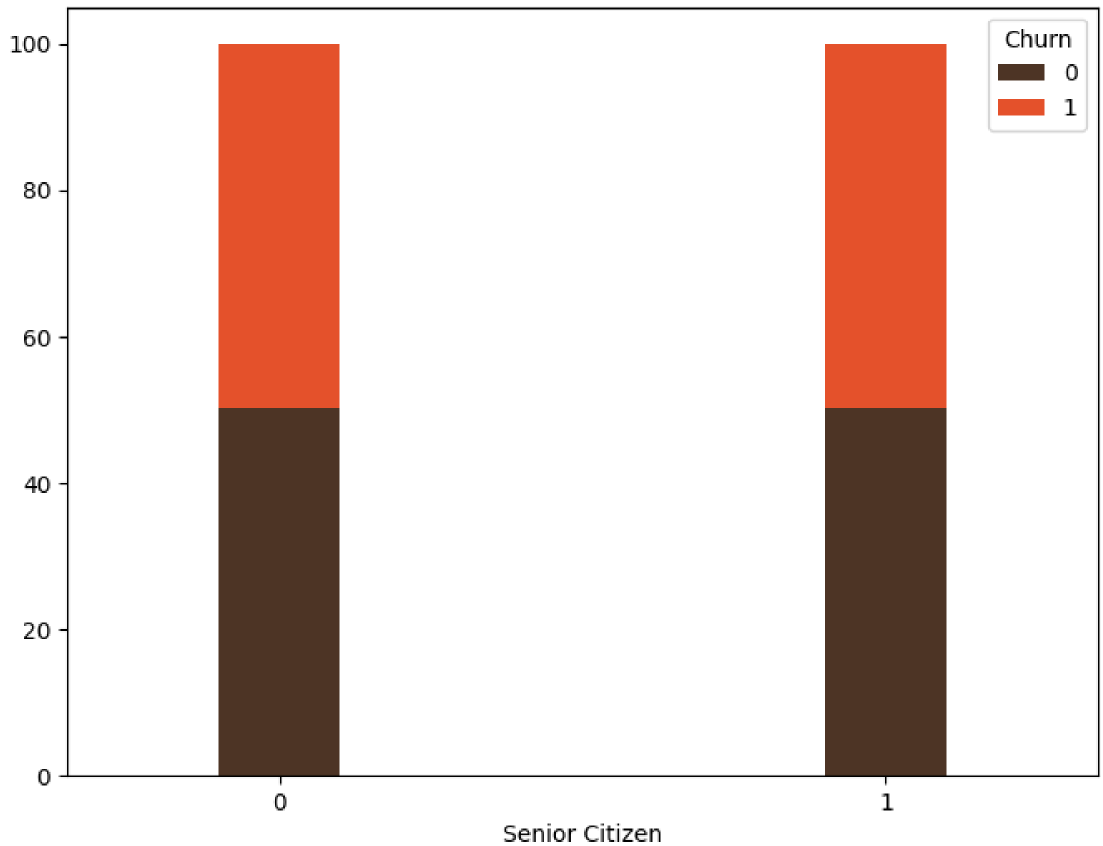
```
# df
```

```
[ ]:        CustomerID             Name  Age  Gender     Location  \
0                1       Customer_1   63    Male  Los Angeles
1                2       Customer_2   62  Female     New York
2                3       Customer_3   24  Female  Los Angeles
3                4       Customer_4   36  Female        Miami
4                5       Customer_5   46  Female        Miami
…              …              …   …      …          …
99995        99996   Customer_99996   33    Male      Houston
99996        99997   Customer_99997   62  Female     New York
99997        99998   Customer_99998   64    Male      Chicago
99998        99999   Customer_99999   51  Female     New York
99999       100000  Customer_100000   27  Female  Los Angeles

       Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
0                              17         73.36             236      0
1                               1         48.76             172      0
2                               5         85.47             460      0
3                               3         97.94             297      1
4                              19         58.14             266      0
…                             …           …             …    …
99995                          23         55.13             226      1
99996                          19         61.65             351      0
99997                          17         96.11             251      1
99998                          20         49.25             434      1
99999                          19         76.57             173      1

       Senior Citizen  Junior Citizen  Median Citizen
0                   1               0               0
1                   1               0               0
2                   0               0               1
3                   0               0               1
4                   0               0               1
…                 …             …             …
99995               0               0               1
99996               1               0               0
99997               1               0               0
99998               1               0               0
99999               0               0               1

[100000 rows x 12 columns]
```

```
[ ]: colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['Senior Citizen','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
```

```
                                                            width = 0.2,
                                                            stacked = True,
                                                            rot = 0,
                                                            figsize = (8,6),
                                                            color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
                color = 'white',
               weight = 'bold',size =14)
```

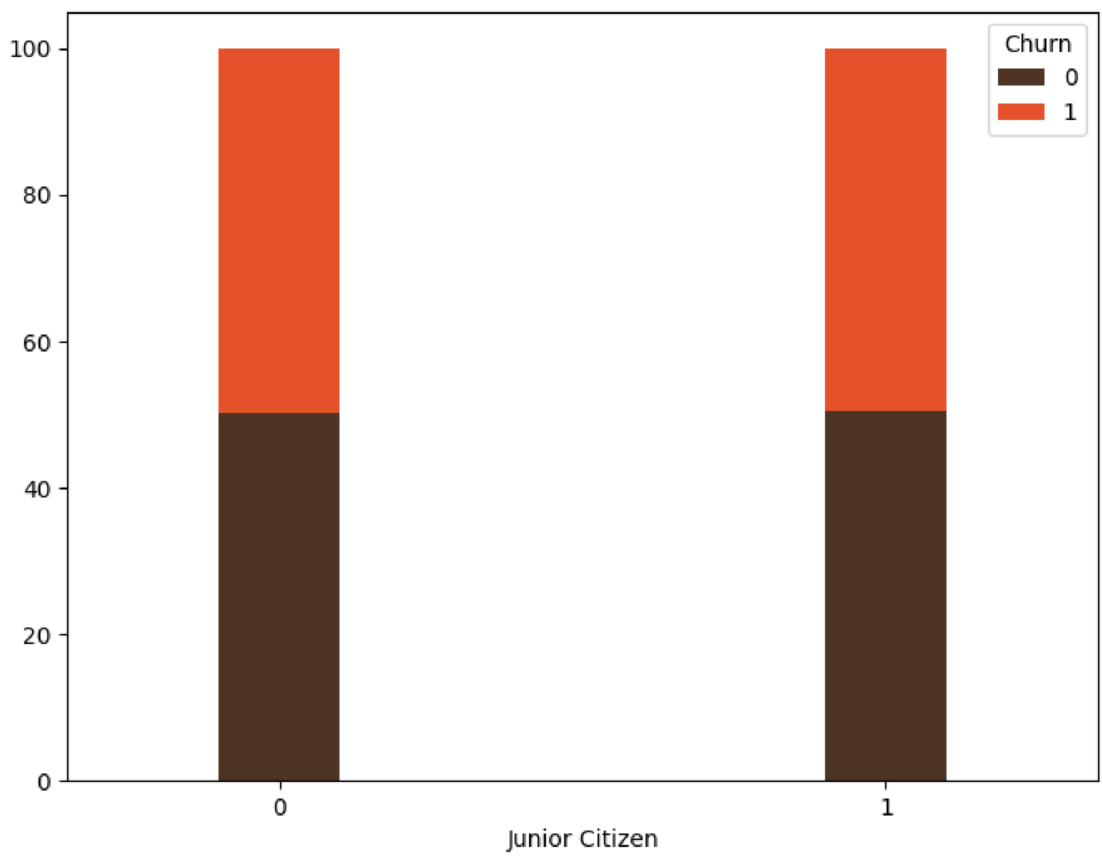
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-72-18b65e94b1fe> in <cell line: 10>()
      8                                                             figsize␣
 ↪= (8,6),
      9                                                             color =␣
 ↪colors)
---> 10 ax.yaxis.set_major_formatter(mtick.PercentFormatter())
     11 ax.legend(loc='center',prop={'size':14},title = 'Churn')
     12 ax.set_ylabel('% Customers')

NameError: name 'mtick' is not defined
```



[ ]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['Junior Citizen','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
```

```
                color = 'white',
                weight = 'bold',size =14)
```

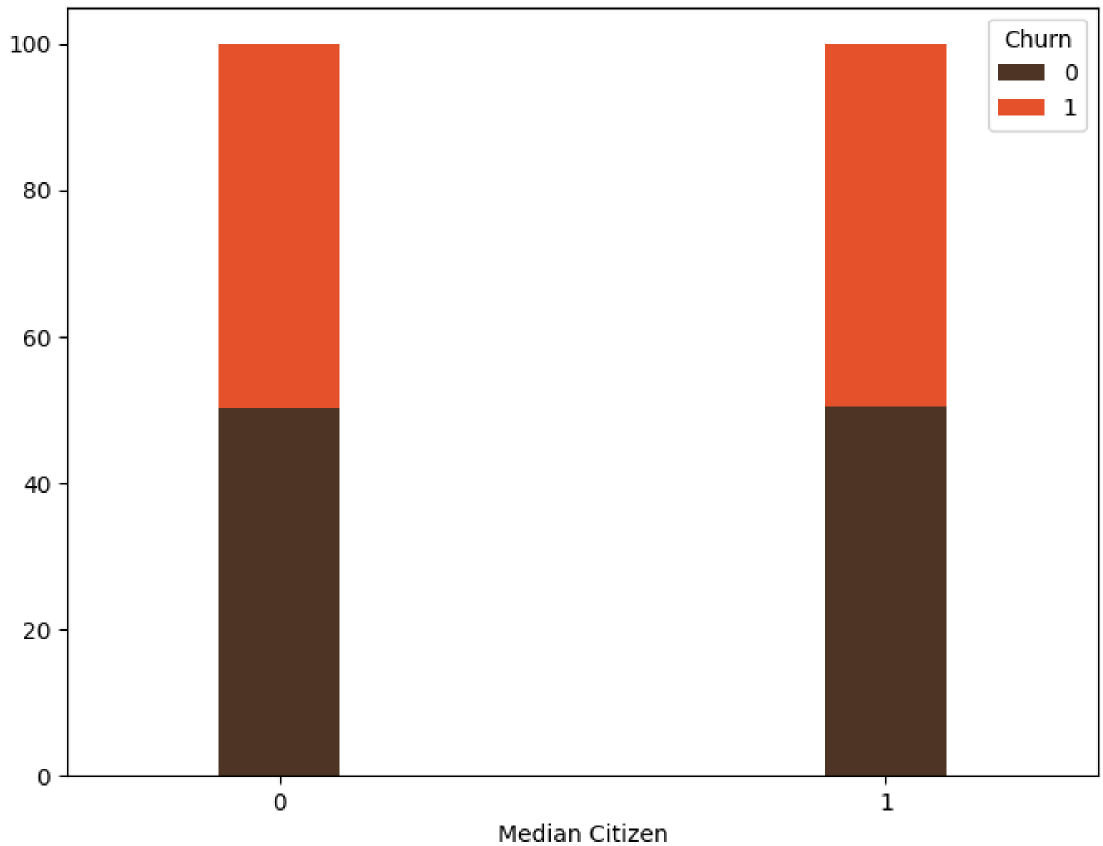
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-73-f80bf588d86d> in <cell line: 10>()
      8                                                             figsize
 ↪= (8,6),
      9                                                             color =
 ↪colors)
---> 10 ax.yaxis.set_major_formatter(mtick.PercentFormatter())
     11 ax.legend(loc='center',prop={'size':14},title = 'Churn')
     12 ax.set_ylabel('% Customers')

NameError: name 'mtick' is not defined
```



[ ]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['Median Citizen','Churn']).size().unstack()
```

```
ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
                color = 'white',
               weight = 'bold',size =14)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-74-dfa4f5ff82e9> in <cell line: 10>()
      8                                                                 figsize␣
 ↪= (8,6),
      9                                                                 color =␣
 ↪colors)
---> 10 ax.yaxis.set_major_formatter(mtick.PercentFormatter())
     11 ax.legend(loc='center',prop={'size':14},title = 'Churn')
     12 ax.set_ylabel('% Customers')

NameError: name 'mtick' is not defined
```

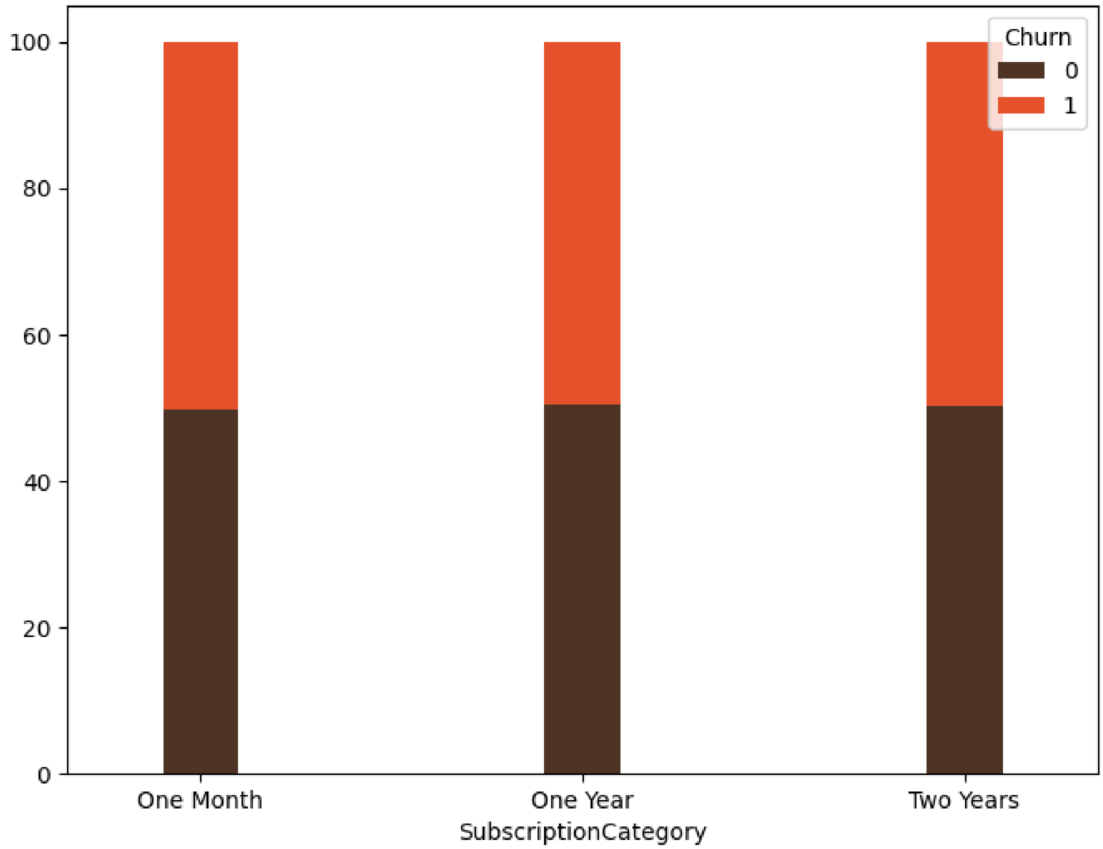


[ ]:
```
# Create a new column based on the duration
def categorize_subscription(duration):
    if duration <= 1:
        return 'One Month'
    elif duration <= 12:
        return 'One Year'
    elif duration <= 24:
        return 'Two Years'
    else:
        return 'Greater than Two Years'

df['SubscriptionCategory'] = df['Subscription_Length_Months'].
 ↪apply(categorize_subscription)
```

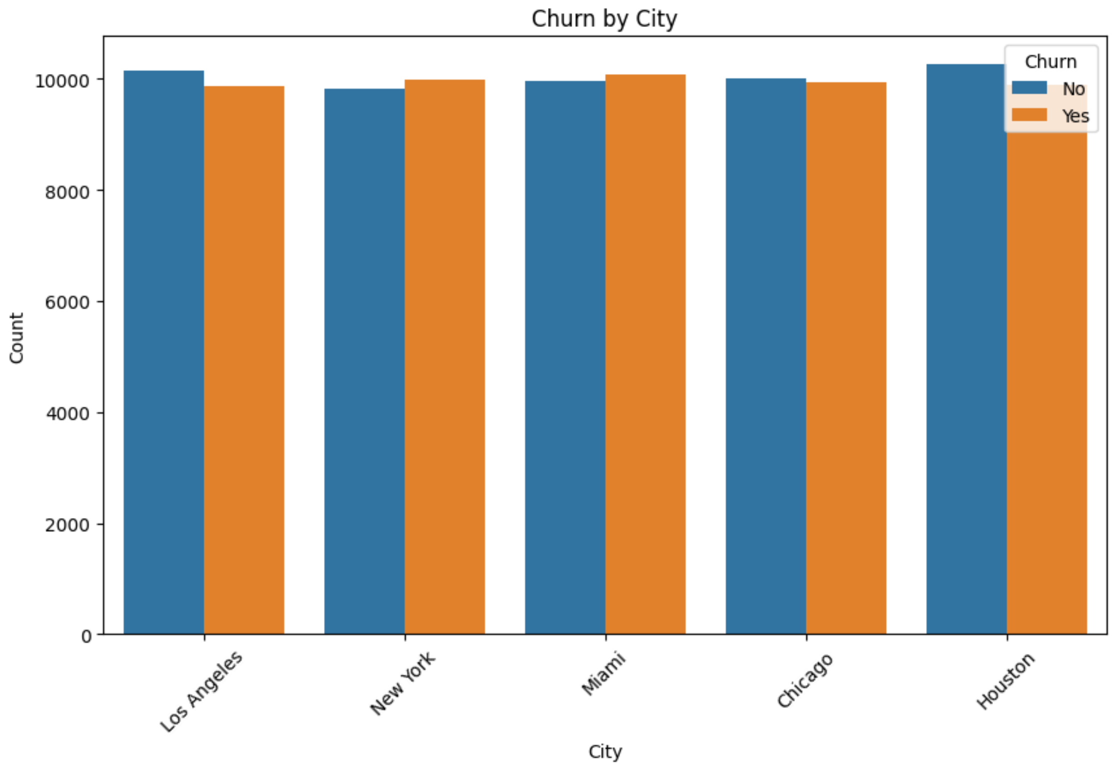
[ ]:
```
colors = ['#4D3425','#E4512B']
seniority_churn = df.groupby(['SubscriptionCategory','Churn']).size().unstack()

ax = (seniority_churn.T*100.0 / seniority_churn.T.sum()).T.plot(kind='bar',
                                                                width = 0.2,
                                                                stacked = True,
```

```
                                                                rot = 0,
                                                                figsize = (8,6),
                                                                color = colors)
ax.yaxis.set_major_formatter(mtick.PercentFormatter())
ax.legend(loc='center',prop={'size':14},title = 'Churn')
ax.set_ylabel('% Customers')
ax.set_title('Churn by Seniority Level',size = 14)

# Code to add the data labels on the stacked bar chart
for p in ax.patches:
    width, height = p.get_width(), p.get_height()
    x, y = p.get_xy()
    ax.annotate('{:.0f}%'.format(height), (p.get_x()+.25*width, p.get_y()+.
 ↪4*height),
                color = 'white',
               weight = 'bold',size =14)
```

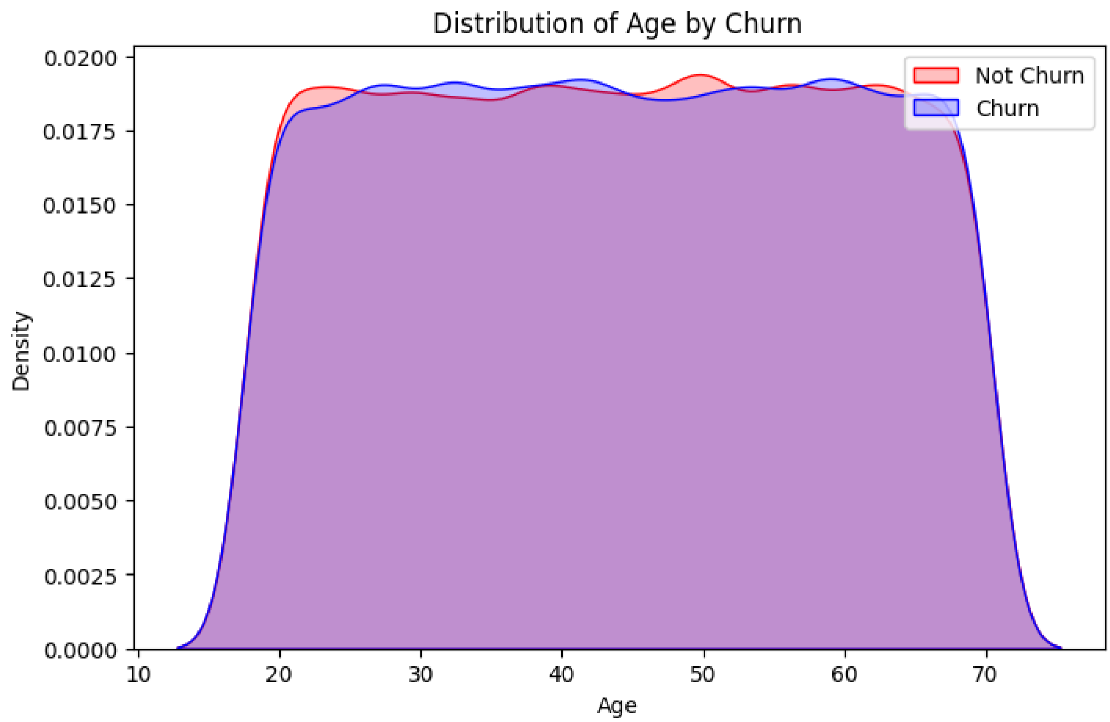
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-78-8aee91c6acb7> in <cell line: 10>()
      8                                                                figsize␣
 ↪= (8,6),
      9                                                                color =␣
 ↪colors)
---> 10 ax.yaxis.set_major_formatter(mtick.PercentFormatter())
     11 ax.legend(loc='center',prop={'size':14},title = 'Churn')
     12 ax.set_ylabel('% Customers')

NameError: name 'mtick' is not defined
```



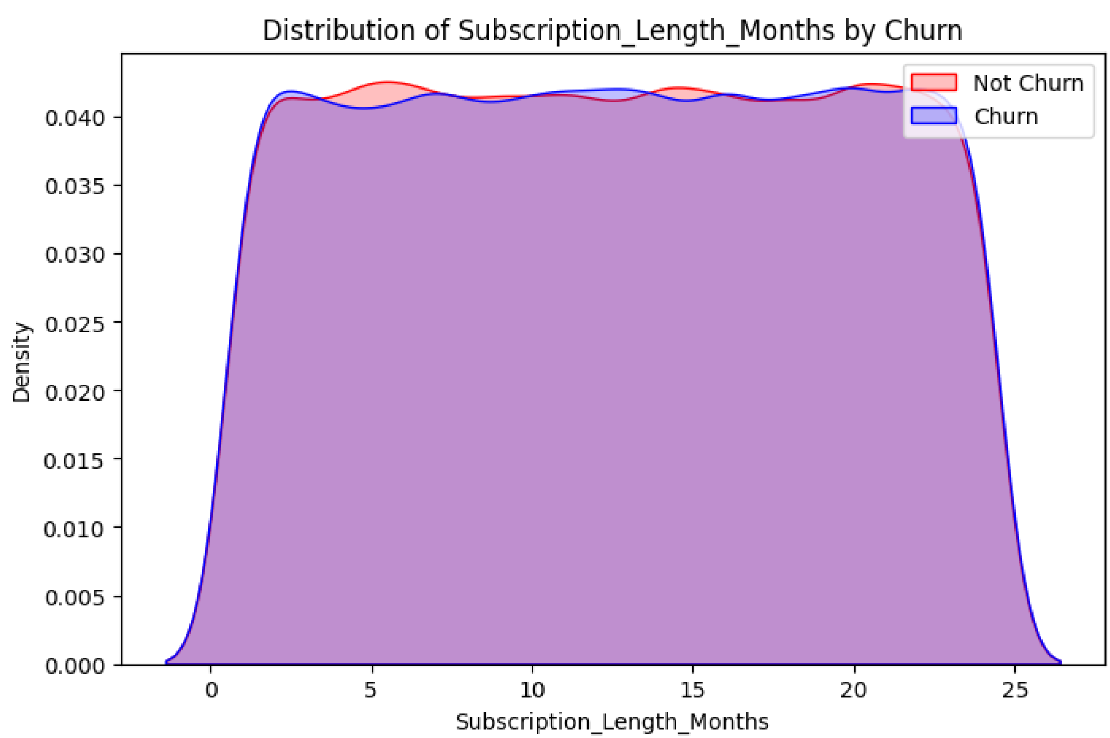
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset includes 'Churn' and 'City' columns

# Create a countplot to visualize the relationship between cities and churn
plt.figure(figsize=(10, 6))
sns.countplot(x='Location', hue='Churn', data=df)
plt.title('Churn by City')
plt.xlabel('City')
plt.ylabel('Count')
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability if needed
plt.legend(title='Churn', labels=['No', 'Yes'])
plt.show()
```



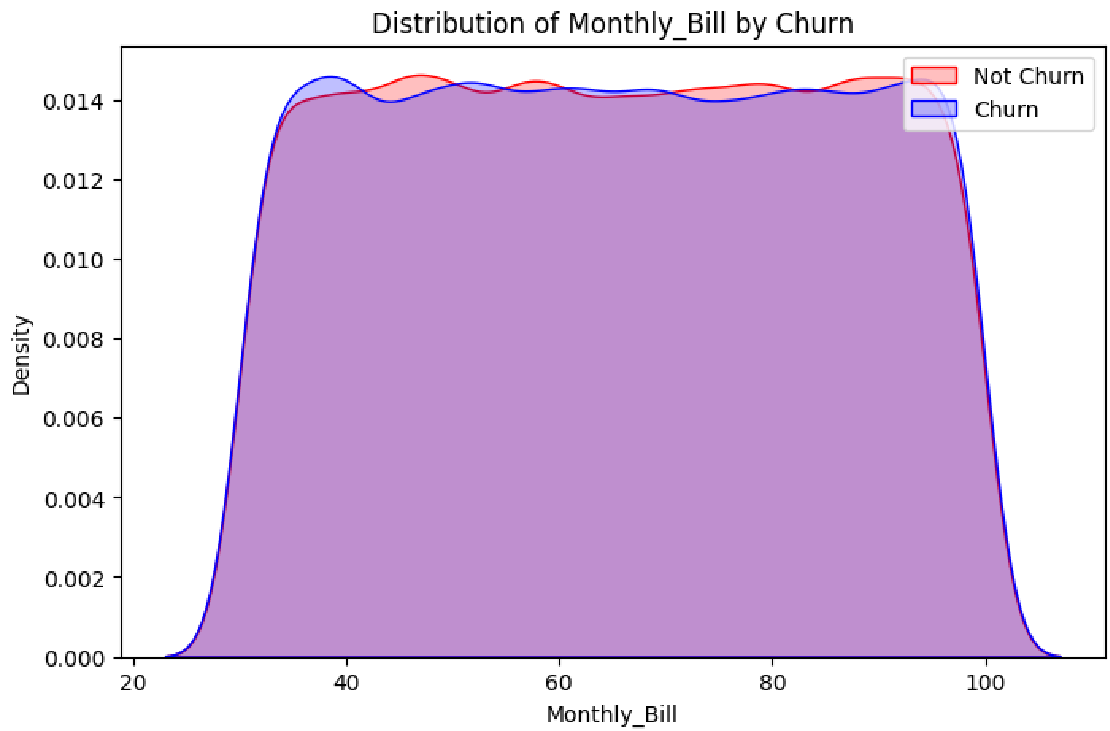
[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

continuous=['Age', 'Subscription_Length_Months','Monthly_Bill',␣
↪'Total_Usage_GB']
categorical=[ 'Gender', 'Location']

for col in continuous:
    plt.figure(figsize=(8, 5))
    ax = sns.kdeplot(df[col][df["Churn"] == 0], color="Red", shade=True)
    ax = sns.kdeplot(df[col][df["Churn"] == 1], ax=ax, color="Blue", shade=True)
    ax.legend(["Not Churn", "Churn"], loc='upper right')
    ax.set_ylabel('Density')
    ax.set_xlabel(col)
    ax.set_title(f'Distribution of {col} by Churn')
    plt.show()
```

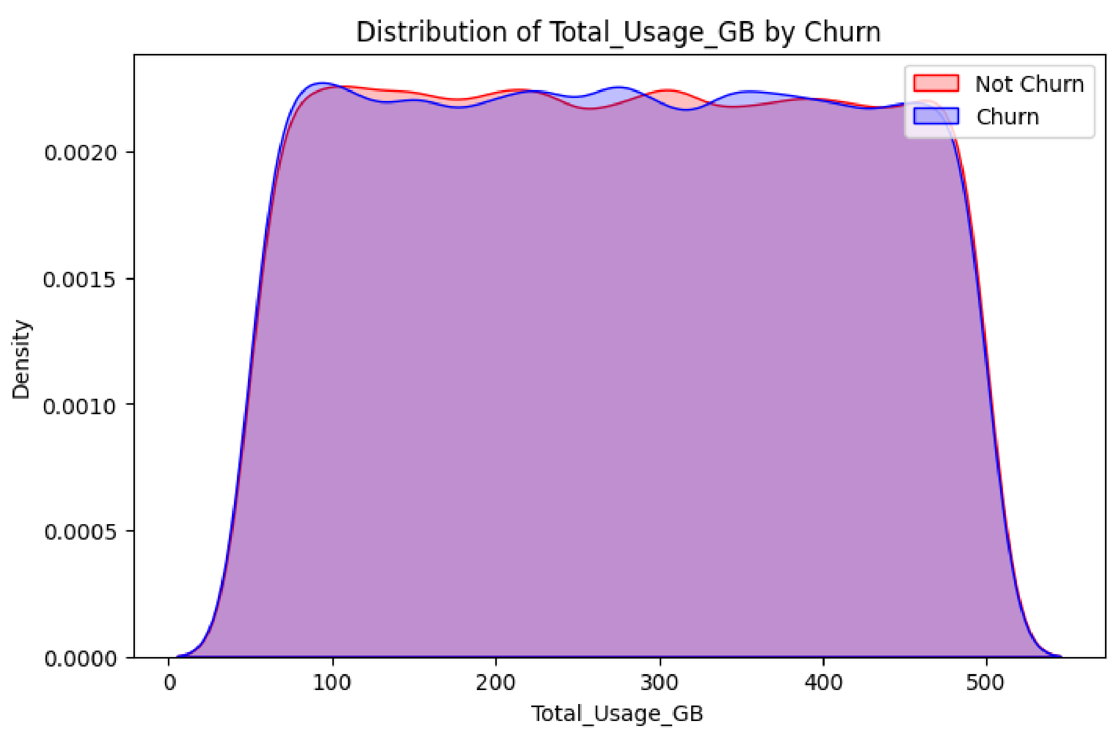


Distribution of Age by Churn
Not Churn
Churn
Density
Age
0.0200
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
10
20
30
40
50
60
70



Distribution of Subscription_Length_Months by Churn
Not Churn
Churn
Density
Subscription_Length_Months
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
5
10
15
20
25


Distribution of Monthly_Bill by Churn
Not Churn
Churn
Density
0.000
0.002
0.004
0.006
0.008
0.010
0.012
0.014
20
40
60
80
100
Monthly_Bill


Distribution of Total_Usage_GB by Churn
Not Churn
Churn
Density
0.0000
0.0005
0.0010
0.0015
0.0020
0
100
200
300
400
500
Total_Usage_GB

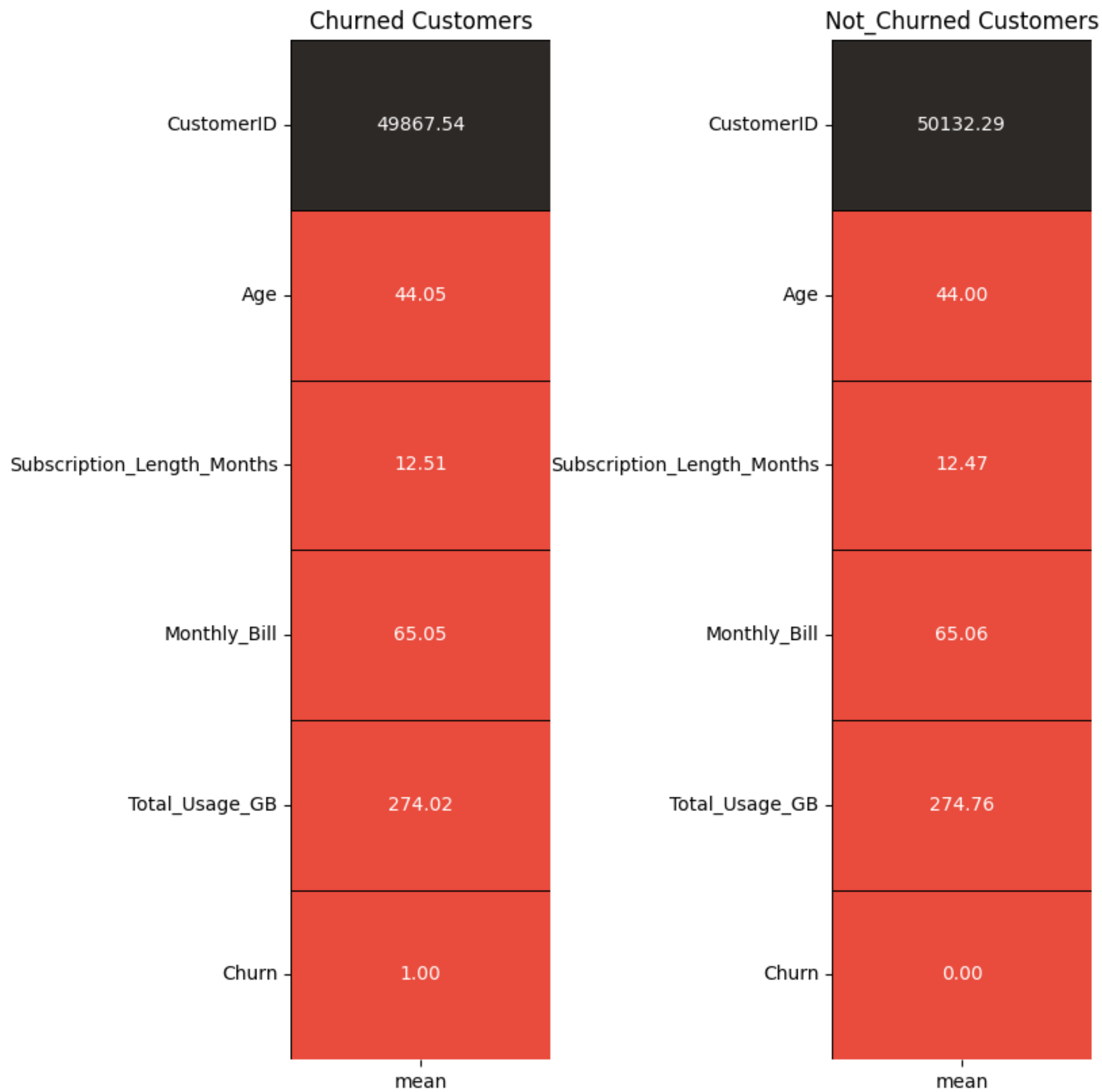
[ ]:
```
# todo
```

[ ]:
```
df
```

[ ]:
```
       Age  Gender     Location  Subscription_Length_Months  Monthly_Bill  \
0       63    Male  Los Angeles                          17         73.36
1       62  Female     New York                           1         48.76
2       24  Female  Los Angeles                           5         85.47
3       36  Female        Miami                           3         97.94
4       46  Female        Miami                          19         58.14
…      …     …           …                          …            …
99995   33    Male      Houston                          23         55.13
99996   62  Female     New York                          19         61.65
99997   64    Male      Chicago                          17         96.11
99998   51  Female     New York                          20         49.25
99999   27  Female  Los Angeles                          19         76.57

       Total_Usage_GB  Churn  Senior Citizen  Junior Citizen  Median Citizen
0                 236      0               1               0               0
1                 172      0               1               0               0
2                 460      0               0               0               1
3                 297      1               0               0               1
4                 266      0               0               0               1
…                …    …              …               …               …
99995             226      1               0               0               1
99996             351      0               1               0               0
99997             251      1               1               0               0
99998             434      1               1               0               0
99999             173      1               0               0               1

[100000 rows x 10 columns]
```

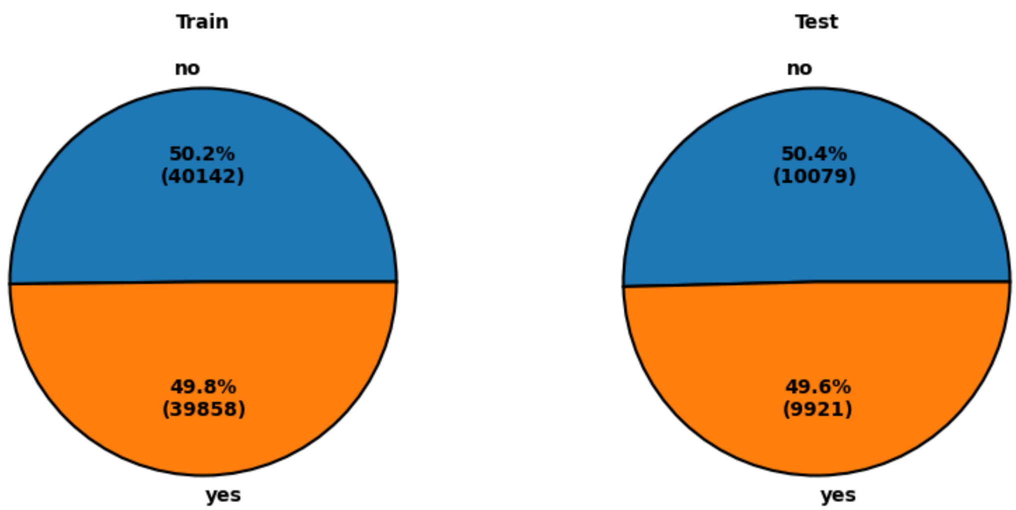
[ ]:
```
colors = ['#E94B3C','#2D2926']

churn = df[df['Churn'] == 1].describe().T
not_churn = df[df['Churn'] == 0].describe().T

fig,ax = plt.subplots(nrows = 1,ncols = 2,figsize = (8,8))
plt.subplot(1,2,1)
sns.heatmap(churn[['mean']],annot = True,cmap = colors,linewidths = 0.
 ↪4,linecolor = 'black',cbar = False,fmt = '.2f')
plt.title('Churned Customers');

plt.subplot(1,2,2)
sns.heatmap(not_churn[['mean']],annot = True,cmap = colors,linewidths = 0.
 ↪4,linecolor = 'black',cbar = False,fmt = '.2f',)
plt.title('Not_Churned Customers');
```

```
fig.tight_layout(pad = 0)
```

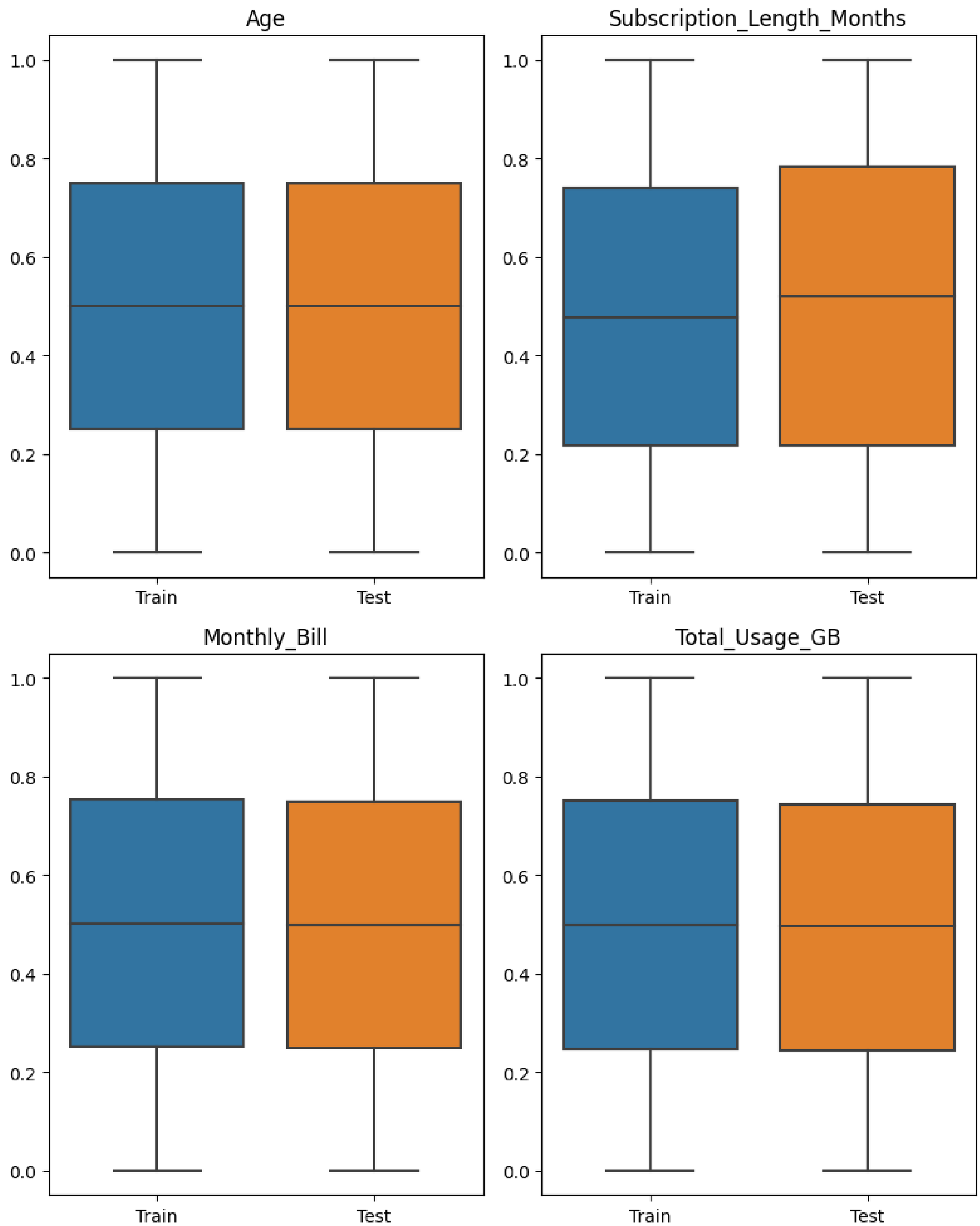


[ ]:
```
# Visualization of the objective variable in the training and test set.
df_pct_train = y_train.value_counts().to_frame().rename(index = {0:'no', 1:
↪'yes'})
df_pct_train = df_pct_train.rename(columns = {'y':'count'})

labels_train = df_pct_train.index.to_list()
values_train = df_pct_train.iloc[:,0]

df_pct_test = y_test.value_counts().to_frame().rename(index = {0:'no', 1:'yes'})
df_pct_test = df_pct_test.rename(columns = {'y':'count'})
```

```python
labels_test = df_pct_test.index.to_list()
values_test = df_pct_test.iloc[:,0]


fig,axes = plt.subplots(1,2,figsize = (9,4))

def autopct_fun(abs_values):
    gen = iter(abs_values)
    return lambda pct: f"{pct:.1f}%\n({next(gen)})"


axes[0].pie(x = values_train, labels = labels_train, autopct =␣
 ↪autopct_fun(values_train),
            wedgeprops = {'linewidth':1.5, 'edgecolor':'black'},
            textprops = {'fontsize':10, 'fontweight':'bold'})
axes[0].set_title('Train', fontsize = 10, fontweight = 'bold', color = 'black')

axes[1].pie(x = values_test, labels = labels_test, autopct =␣
 ↪autopct_fun(values_test),
            wedgeprops = {'linewidth':1.5, 'edgecolor':'black'},
            textprops = {'fontsize':10, 'fontweight':'bold'})
axes[1].set_title('Test', fontsize = 10, fontweight = 'bold', color = 'black')

fig.tight_layout()
fig.subplots_adjust(top = 0.9)
fig.show()
```

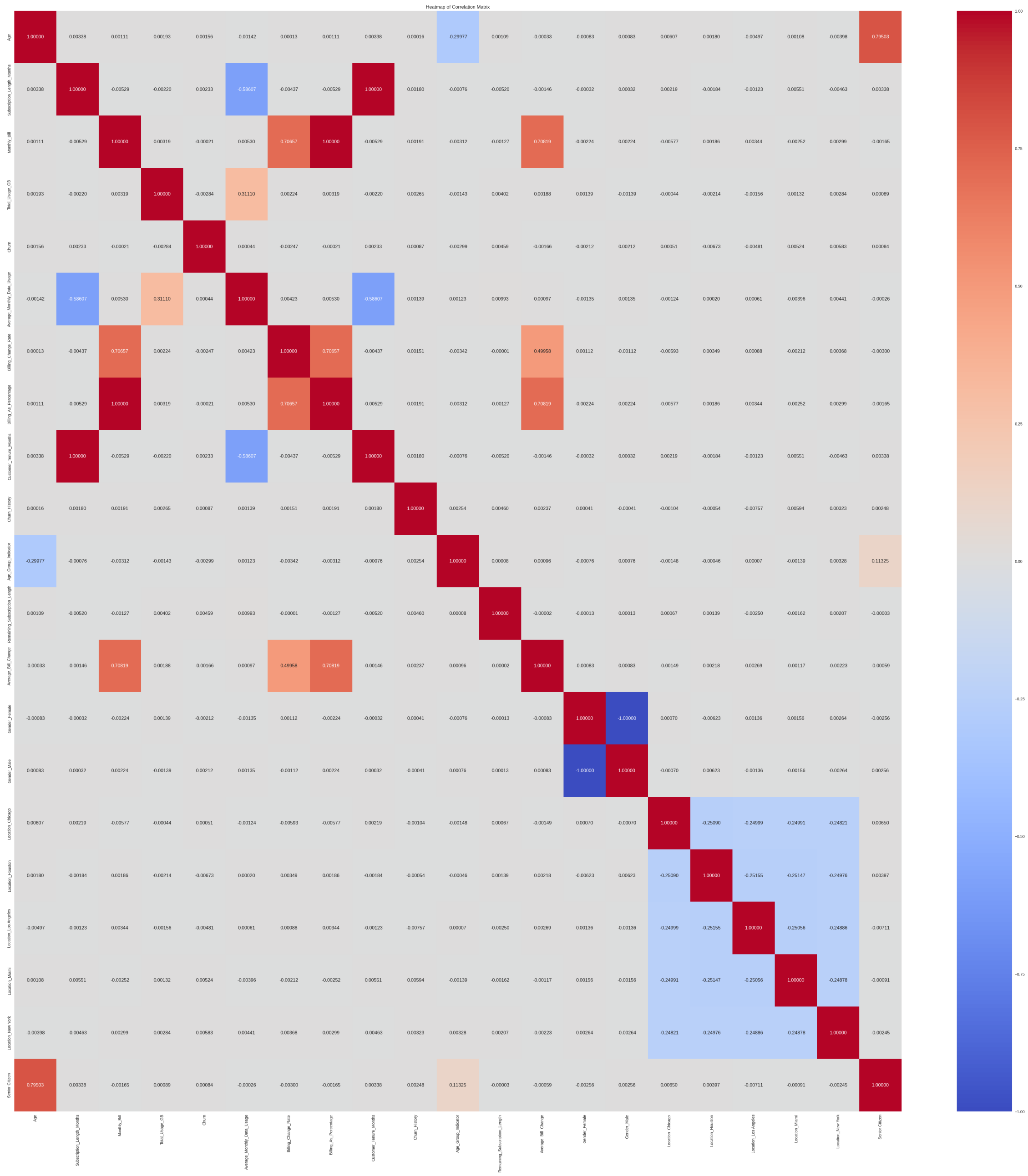


[ ]:
```python
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the number of rows needed based on the number of columns
num_cols = len(continuous)
num_plots_per_row = 2
num_rows = int(np.ceil(num_cols / num_plots_per_row))

# Adjust the figure size based on the number of rows and plots per row
fig, axes = plt.subplots(num_rows, num_plots_per_row, figsize=(8, 5 * num_rows))

# Flatten the axes if there's only one row
if num_rows == 1:
    axes = axes.reshape(1, -1)

for i, column in enumerate(continuous):
    row_idx = i // num_plots_per_row
    col_idx = i % num_plots_per_row

    # Create a single axis for each variable
    ax = axes[row_idx, col_idx]

    # Create boxplots for train and test data side by side
    sns.boxplot(data=[X_train_cont[column], X_test_cont[column]], ax=ax)
    ax.set_title(column)
    ax.set_xticklabels(['Train', 'Test'])

plt.tight_layout()
plt.show()
```



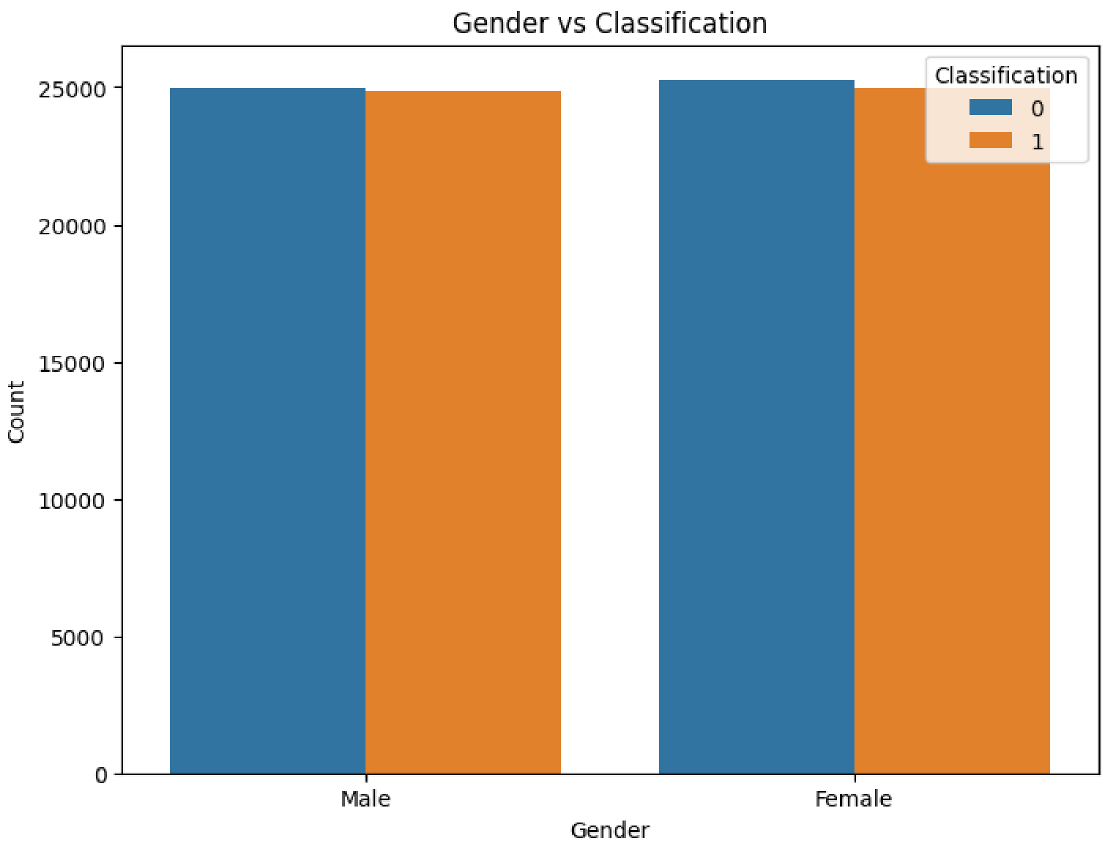
[ ]:
```python
# with new features:
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

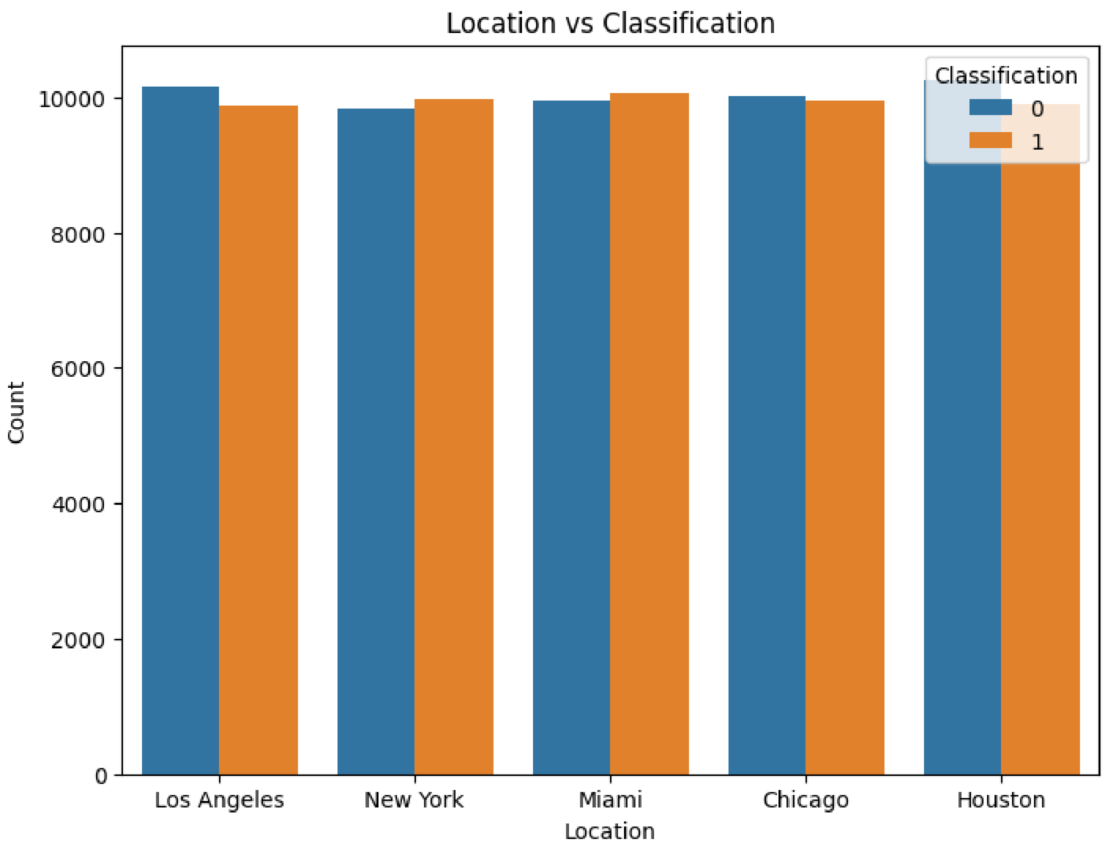
```python
# Load your dataset
# df = pd.read_csv('customer_data.csv')  # Replace 'customer_data.csv' with␣
↪your dataset file path

# Assuming your dataset has columns like 'age', 'churn', 'city', and 'gender'

# Define the age threshold for senior citizens
senior_age_threshold = 55

# Create a new column 'Senior Citizen' based on age
df['Senior Citizen'] = df['Age'] >= senior_age_threshold

# Create a heatmap to visualize the relationship between 'Senior Citizen' and␣
↪other variables
plt.figure(figsize=(50, 50))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".5f")
plt.title('Heatmap of Correlation Matrix')
plt.show()
```



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

# Replace 'categorical_vars' with the list of your categorical variable names
categorical_vars = ['Gender', 'Location']

# Loop through each categorical variable and create bar plots
for cat_var in categorical_vars:
```

```python
plt.figure(figsize=(8, 6))
sns.countplot(x=cat_var, hue='Churn', data=df)
plt.title(f'{cat_var} vs Classification')
plt.xlabel(cat_var)
plt.ylabel('Count')
plt.legend(title='Classification', loc='upper right')
plt.show()
```





3 3 Model Building:

- Choose appropriate machine learning algorithms (e.g., logistic regression, random forest, or neural networks).
- Train and validate the selected model on the training dataset.
- Evaluate the model's performance using appropriate metrics (e.g., accuracy, precision, recall, F1-score).

[ ]:
```
SEED=42
```

[ ]:
```
# Estimators for the VotingClassifier and StackingClassifier.
e1 = XGBClassifier(objective = 'binary:logistic', random_state = SEED,
                   tree_method = "exact", booster = 'gbtree')
e2 = LGBMClassifier(random_state = SEED, verbose = -1)
e3 = CatBoostClassifier(random_state = SEED, verbose = 0)

voting1 = VotingClassifier(estimators = [('lgbm', e2), ('cb', e3)])
```

[ ]:

```python
lr = LogisticRegression(random_state = SEED)

rf = RandomForestClassifier(random_state = SEED)

et = ExtraTreesClassifier(bootstrap = True, random_state = SEED)

svc = SVC(random_state = SEED)

knn = KNeighborsClassifier()

hgb = HistGradientBoostingClassifier(random_state = SEED)

xgb = XGBClassifier(objective = 'binary:logistic', tree_method = "exact",
                    random_state = SEED, booster = 'gbtree')

lgbm = LGBMClassifier(random_state = SEED, verbose = -1)

cb = CatBoostClassifier(random_state = SEED, verbose = 0)

voting = VotingClassifier(estimators = [('lgbm', e2), ('cb', e3)])

stacking = StackingClassifier(estimators = [('lgbm', e2), ('cb', e3)],
                              final_estimator = voting1, stack_method='predict')

# We create a dictionary where we store our models with their respective names.
# base_models = {'Logistic_Regression':lr,
              #  'Random_Forest':rf,
              #  'Extra_Trees':et,
              #  'SVC':svc,
              #  'KNN':knn,
              #  'XGB':xgb,
              #  'LGBM':lgbm,
              #  'CatBoost':cb,
              #  'Voting':voting,
              #  'Stacking':stacking,
              #  'HistGrad':hgb}

base_models = {'Logistic_Regression':lr,
               'Random_Forest':rf,
               'Extra_Trees':et,
               'KNN':knn,
               'XGB':xgb,
               'LGBM':lgbm,
               'CatBoost':cb,
               'Voting':voting,
               'Stacking':stacking,
               'HistGrad':hgb}
```

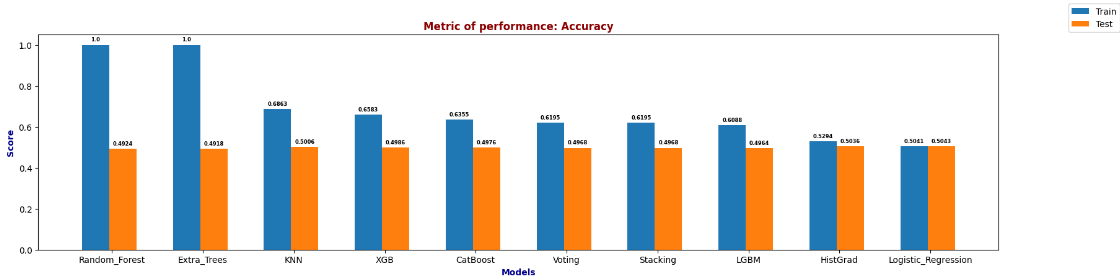
[ ]:
```
# without smote
## Training

# Dictionary where we will store the metrics of each model.
accuracy_train = {}
accuracy_test = {}

confusion_matrix_train = {}
confusion_matrix_test = {}

for model_name, model in base_models.items():
  start = time.time()
  model.fit(X_train, y_train)
  end = time.time()
  print(f'* {model_name}: {end-start} seconds')
  y_pred_train = model.predict(X_train)
  y_pred_test = model.predict(X_test)
  accuracy_train[model_name] = accuracy_score(np.array(y_train), y_pred_train)
  accuracy_test[model_name] = accuracy_score(np.array(y_test), y_pred_test)
  confusion_matrix_train[model_name] = confusion_matrix(np.array(y_train),␣
 ↪y_pred_train)
  confusion_matrix_test[model_name] = confusion_matrix(np.array(y_test),␣
 ↪y_pred_test)

# We create a dataframe showing the accuracy results in training and testing.
df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient = 'index').
 ↪rename(columns = {0:'Train'})
df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient = 'index').
 ↪rename(columns = {0:'Test'})
df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index = True,␣
 ↪right_index = True)
df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending = False)
df_accuracy
```

```
* Logistic_Regression: 0.24571847915649414 seconds
* Random_Forest: 17.444645881652832 seconds
* Extra_Trees: 5.900312185287476 seconds
* KNN: 0.004964351654052734 seconds
* XGB: 5.007220983505249 seconds
* LGBM: 0.46331143379211426 seconds
* CatBoost: 15.28477144241333 seconds
* Voting: 13.338232040405273 seconds
* Stacking: 79.97222471237183 seconds
* HistGrad: 0.22475051879882812 seconds
```

[ ]:
```
                        Train     Test
Random_Forest        1.000000  0.49240
```

```
Extra_Trees          1.000000  0.49180
KNN                  0.686288  0.50055
XGB                  0.658275  0.49855
CatBoost             0.635550  0.49765
Voting               0.619537  0.49675
Stacking             0.619537  0.49675
LGBM                 0.608750  0.49635
HistGrad             0.529438  0.50355
Logistic_Regression  0.504088  0.50430
```

[ ]:
```
# Let's visualize the metric in a bar graph.
fig,ax = plt.subplots(figsize = (20, 4.5))
n = len(df_accuracy.index)
x = np.arange(n)
width = 0.3

rects1 = ax.bar(x = x-width, height = df_accuracy.iloc[:,0], width = width)
rects2 = ax.bar(x = x, height = df_accuracy.iloc[:,1], width = width)
ax.set_xticks(x-0.12, df_accuracy.index.to_list())
ax.set_xlabel('Models', fontsize = 10, fontweight = 'bold', color = 'darkblue')
ax.set_ylabel('Score', fontsize = 10, fontweight = 'bold', color = 'darkblue')
ax.set_title('Metric of performance: Accuracy', fontsize = 12, fontweight =␣
↪'bold', color = 'darkred')

def autolabel(rects):
  for rect in rects:
    height = rect.get_height()
    ax.annotate(text = f'{round(height,4)}',
                xy = (rect.get_x()+rect.get_width()/2, height),
                xytext = (0,3),
                textcoords = 'offset points',
                ha = 'center',
                va = 'bottom',
                size = 6,
                weight = 'bold',
                color = 'black')

autolabel(rects1)
autolabel(rects2)
fig.legend(["Train", "Test"])
fig.show()
```



This time we will also try to find out the feauture importance for all the features as given in the dataset after feature engineering & Normalisation

[ ]:
```
X_train
```

[ ]:
```
array([[0.69230769, 0.17391304, 0.77857143, …, 0.        , 0.        ,
        1.        ],
       [0.19230769, 1.        , 0.74371429, …, 0.        , 0.        ,
        1.        ],
       [0.75      , 0.47826087, 0.31842857, …, 0.        , 0.        ,
        0.        ],
       …,
       [0.98076923, 0.04347826, 0.66057143, …, 0.        , 0.        ,
        0.        ],
       [0.71153846, 0.47826087, 0.84557143, …, 0.        , 0.        ,
        0.        ],
       [0.15384615, 0.69565217, 0.57728571, …, 1.        , 0.        ,
        0.        ]])
```

[ ]:
```
# Performance with 3 age features
## Training

# Dictionary where we will store the metrics of each model.
accuracy_train = {}
accuracy_test = {}

confusion_matrix_train = {}
confusion_matrix_test = {}

feature_importance = {}  # Dictionary to store feature importance for each model

for model_name, model in base_models.items():
    start = time.time()
    model.fit(X_train, y_train)
    end = time.time()
    print(f'* {model_name}: {end-start} seconds')
```

```python
    # Calculate feature importance for the current model
    if hasattr(model, 'feature_importances_'):
        feature_importance[model_name] = model.feature_importances_
    else:
        feature_importance[model_name] = None

    y_pred_train = model.predict(X_train)
    y_pred_test = model.predict(X_test)

    accuracy_train[model_name] = accuracy_score(np.array(y_train), y_pred_train)
    accuracy_test[model_name] = accuracy_score(np.array(y_test), y_pred_test)

    confusion_matrix_train[model_name] = confusion_matrix(np.array(y_train),␣
 ↪y_pred_train)
    confusion_matrix_test[model_name] = confusion_matrix(np.array(y_test),␣
 ↪y_pred_test)

# We create a dataframe showing the accuracy results in training and testing.
df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').
 ↪rename(columns={0: 'Train'})
df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').
 ↪rename(columns={0: 'Test'})
df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True,␣
 ↪right_index=True)
df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)
df_accuracy

# Display feature importance for each model
for model_name, importance_scores in feature_importance.items():
    print(f"Feature Importance for {model_name}:\n{importance_scores}\n")
```

```
* Logistic_Regression: 0.24431657791137695 seconds
* Random_Forest: 14.356007099151611 seconds
* Extra_Trees: 5.80819034576416 seconds
* KNN: 0.14777493476867676 seconds
* XGB: 6.133852005004883 seconds
* LGBM: 0.37482261657714844 seconds
* CatBoost: 12.685113668441772 seconds
* Voting: 12.999969005584717 seconds
* Stacking: 80.21916794776917 seconds
* HistGrad: 0.3250269889831543 seconds
Feature Importance for Logistic_Regression:
None

Feature Importance for Random_Forest:
[0.16646724 0.16392413 0.3267288  0.31016322 0.00227566 0.00193949
```

```
 0.00284723 0.00350085 0.00354699 0.00393858 0.00358155 0.00402646
 0.00359787 0.00346193]

Feature Importance for Extra_Trees:
[0.20060919 0.19896479 0.29138996 0.28682991 0.00209836 0.00186667
 0.00259327 0.00184835 0.00186831 0.00240161 0.00251508 0.00243365
 0.00235886 0.00222201]

Feature Importance for KNN:
None

Feature Importance for XGB:
[0.09142012 0.08974636 0.09829018 0.09034033 0.         0.
 0.1056229  0.08631585 0.         0.08588835 0.08964626 0.09026954
 0.08443113 0.08802893]

Feature Importance for LGBM:
[636 435 775 823   0   1  17  90  15  35  39  33  51  50]

Feature Importance for CatBoost:
[17.96148895 15.61215413 24.49617662 24.86973519  0.42153705  0.59495148
  1.50227705  2.23542685  1.32647558  2.08496323  2.00419713  2.64969369
  1.93182017  2.30910288]

Feature Importance for Voting:
None

Feature Importance for Stacking:
None

Feature Importance for HistGrad:
None
```

[ ]:
```
df_accuracy
```

[ ]:

	Train	Test
Extra_Trees	1.000000	0.49590
Random_Forest	1.000000	0.49355
KNN	0.687137	0.49800
XGB	0.656438	0.50260
CatBoost	0.634200	0.49660
Voting	0.619263	0.49570
Stacking	0.619263	0.49570
LGBM	0.611463	0.49935
HistGrad	0.530438	0.50395
Logistic_Regression	0.504062	0.50425

Features like Senior Citizen, Junior Citizen, Median Citizen, Gender_Male, Location_Chicago, Location_Los Angeles, Location_Miami, and Location_New York have consistently low importance scores across models. KNN Model:

[ ]:
```
#Without normalisation:
# Performance with 3 age features
## Training

# Dictionary where we will store the metrics of each model.
accuracy_train = {}
accuracy_test = {}

confusion_matrix_train = {}
confusion_matrix_test = {}

feature_importance = {}  # Dictionary to store feature importance for each model

for model_name, model in base_models.items():
    start = time.time()
    model.fit(X_train2, y_train2)
    end = time.time()
    print(f'* {model_name}: {end-start} seconds')

    # Calculate feature importance for the current model
    if hasattr(model, 'feature_importances_'):
        feature_importance[model_name] = model.feature_importances_
    else:
        feature_importance[model_name] = None

    y_pred_train2 = model.predict(X_train2)
    y_pred_test2 = model.predict(X_test2)

    accuracy_train[model_name] = accuracy_score(np.array(y_train2),␣
 ↪y_pred_train2)
    accuracy_test[model_name] = accuracy_score(np.array(y_test2), y_pred_test2)

    confusion_matrix_train[model_name] = confusion_matrix(np.array(y_train2),␣
 ↪y_pred_train2)
    confusion_matrix_test[model_name] = confusion_matrix(np.array(y_test2),␣
 ↪y_pred_test2)

# We create a dataframe showing the accuracy results in training and testing.
df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').
 ↪rename(columns={0: 'Train'})
df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').
 ↪rename(columns={0: 'Test'})
```

```
df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True,␣
↪right_index=True)
df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)
df_accuracy

# Display feature importance for each model
for model_name, importance_scores in feature_importance.items():
    print(f"Feature Importance for {model_name}:\n{importance_scores}\n")
```

```
* Logistic_Regression: 0.0907135009765625 seconds
* Random_Forest: 10.954647064208984 seconds
* Extra_Trees: 5.647444009780884 seconds
* KNN: 0.15998244285583496 seconds
* XGB: 4.426905393600464 seconds
* LGBM: 0.3773632049560547 seconds
* CatBoost: 12.584888696670532 seconds
* Voting: 13.656790971755981 seconds
* Stacking: 77.42293548583984 seconds
* HistGrad: 0.17517542839050293 seconds
Feature Importance for Logistic_Regression:
None

Feature Importance for Random_Forest:
[0.16646724 0.16392413 0.3267288  0.31016322 0.00227566 0.00193949
 0.00284723 0.00350085 0.00354699 0.00393858 0.00358155 0.00402646
 0.00359787 0.00346193]

Feature Importance for Extra_Trees:
[0.20070098 0.19890828 0.29132962 0.28690111 0.00209314 0.00187971
 0.00261977 0.00184923 0.00187382 0.00237257 0.00250284 0.00243681
 0.00232927 0.00220285]

Feature Importance for KNN:
None

Feature Importance for XGB:
[0.09142012 0.08974636 0.09829018 0.09034033 0.         0.
 0.1056229  0.08631585 0.         0.08588835 0.08964626 0.09026954
 0.08443113 0.08802893]

Feature Importance for LGBM:
[629 439 827 770   0   0  20  87  16  41  37  40  42  52]

Feature Importance for CatBoost:
[17.96148895 15.61215413 24.49617662 24.86973519  0.42153705  0.59495148
  1.50227705  2.23542685  1.32647558  2.08496323  2.00419713  2.64969369
  1.93182017  2.30910288]
```

```
Feature Importance for Voting:
None

Feature Importance for Stacking:
None

Feature Importance for HistGrad:
None
```

[ ]:
```
df_accuracy
```

[ ]:

	Train	Test
Extra_Trees	1.000000	0.49565
Random_Forest	1.000000	0.49450
KNN	0.687625	0.49650
XGB	0.656438	0.50255
CatBoost	0.634200	0.49660
Voting	0.616762	0.49890
Stacking	0.616762	0.49890
LGBM	0.607775	0.50085
HistGrad	0.530438	0.50395
Logistic_Regression	0.502575	0.49975

[ ]:
```
# Let's visualize the metric in a bar graph.
fig,ax = plt.subplots(figsize = (20, 4.5))
n = len(df_accuracy.index)
x = np.arange(n)
width = 0.3

rects1 = ax.bar(x = x-width, height = df_accuracy.iloc[:,0], width = width)
rects2 = ax.bar(x = x, height = df_accuracy.iloc[:,1], width = width)
ax.set_xticks(x-0.12, df_accuracy.index.to_list())
ax.set_xlabel('Models', fontsize = 10, fontweight = 'bold', color = 'darkblue')
ax.set_ylabel('Score', fontsize = 10, fontweight = 'bold', color = 'darkblue')
ax.set_title('Metric of performance: Accuracy with 3 age features', fontsize =␣
 ↪12, fontweight = 'bold', color = 'darkred')

def autolabel(rects):
  for rect in rects:
    height = rect.get_height()
    ax.annotate(text = f'{round(height,4)}',
                xy = (rect.get_x()+rect.get_width()/2, height),
                xytext = (0,3),
                textcoords = 'offset points',
                ha = 'center',
```

```
                va = 'bottom',
                size = 6,
                weight = 'bold',
                color = 'black')

autolabel(rects1)
autolabel(rects2)
fig.legend(["Train", "Test"])
fig.show()
```

[ ]:
```
#Without normalisation:
# Performance with 3 age features
## Training

# Dictionary where we will store the metrics of each model.
accuracy_train = {}
accuracy_test = {}

confusion_matrix_train = {}
confusion_matrix_test = {}

feature_importance = {}  # Dictionary to store feature importance for each model

for model_name, model in base_models.items():
    start = time.time()
    model.fit(X_train2, y_train2)
    end = time.time()
    print(f'* {model_name}: {end-start} seconds')

    # Calculate feature importance for the current model
    if hasattr(model, 'feature_importances_'):
        feature_importance[model_name] = model.feature_importances_
    else:
        feature_importance[model_name] = None

    y_pred_train2 = model.predict(X_train2)
    y_pred_test2 = model.predict(X_test2)

    accuracy_train[model_name] = accuracy_score(np.array(y_train2),␣
 ↪y_pred_train2)
    accuracy_test[model_name] = accuracy_score(np.array(y_test2), y_pred_test2)

    confusion_matrix_train[model_name] = confusion_matrix(np.array(y_train2),␣
 ↪y_pred_train2)
    confusion_matrix_test[model_name] = confusion_matrix(np.array(y_test2),␣
 ↪y_pred_test2)
```

```python
# We create a dataframe showing the accuracy results in training and testing.
df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').
 ↪rename(columns={0: 'Train'})
df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').
 ↪rename(columns={0: 'Test'})
df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True,␣
 ↪right_index=True)
df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)
df_accuracy

# Display feature importance for each model
for model_name, importance_scores in feature_importance.items():
    print(f"Feature Importance for {model_name}:\n{importance_scores}\n")
```

```
* Logistic_Regression: 0.03347945213317871 seconds
* Random_Forest: 34.56139636039734 seconds
* Extra_Trees: 6.170707702636719 seconds
* KNN: 0.13585519790649414 seconds
* XGB: 7.86917519569397 seconds
* LGBM: 0.5224440097808838 seconds
* CatBoost: 14.47956919670105 seconds
* Voting: 14.985882997512817 seconds
* Stacking: 94.602698802948 seconds
* HistGrad: 0.25035524368286133 seconds
Feature Importance for Logistic_Regression:
None

Feature Importance for Random_Forest:
[0.08572883 0.04907891 0.10429026 0.10701157 0.10959937 0.12000839
 0.10394139 0.0491369  0.01652237 0.01273596 0.12152581 0.12042025]

Feature Importance for Extra_Trees:
[0.0984813  0.06733216 0.10163665 0.10323029 0.1012007  0.10700234
 0.10208763 0.06730873 0.01915719 0.01866966 0.10723561 0.10665776]

Feature Importance for KNN:
None

Feature Importance for XGB:
[0.10259998 0.08856104 0.10673914 0.10528548 0.10892413 0.10314061
 0.         0.         0.08181228 0.08819617 0.10636989 0.10837135]

Feature Importance for LGBM:
[329 156 386 418 357 433   0   0  49  18 451 403]

Feature Importance for CatBoost:
[11.5482856   3.89865863  8.91325583 11.80431674 11.40613182 14.05020688
```

```
  4.97266225  2.29599524  1.86921901  1.32473985 13.94304471 13.97348342]

Feature Importance for Voting:
None

Feature Importance for Stacking:
None

Feature Importance for HistGrad:
None
```

[ ]:
```
X.columns
```

[ ]:
```
Index(['Age', 'Subscription_Length_Months', 'Monthly_Bill', 'Total_Usage_GB',
       'Average_Monthly_Data_Usage', 'Billing_Change_Rate',
       'Billing_As_Percentage', 'Customer_Tenure_Months', 'Churn_History',
       'Age_Group_Indicator', 'Remaining_Subscription_Length',
       'Average_Bill_Change'],
      dtype='object')
```

[ ]:
```
df_accuracy
```

[ ]:

	Train	Test
Extra_Trees	1.000000	0.50405
Random_Forest	1.000000	0.50075
KNN	0.684450	0.50260
CatBoost	0.678113	0.50460
XGB	0.663963	0.50200
Voting	0.646250	0.50395
LGBM	0.624525	0.50200
HistGrad	0.538150	0.51150
Logistic_Regression	0.501763	0.50400
Stacking	0.480050	0.50200

WARNING: Runtime no longer has a reference to this dataframe, please re-run this cell and try again.

[ ]:
```
#Without normalisation:
# Performance with 3 age features
## Training

# Dictionary where we will store the metrics of each model.
accuracy_train = {}
accuracy_test = {}

confusion_matrix_train = {}
confusion_matrix_test = {}
```

```
feature_importance = {}  # Dictionary to store feature importance for each model

for model_name, model in base_models.items():
    start = time.time()
    model.fit(X_train3, y_train3)
    end = time.time()
    print(f'* {model_name}: {end-start} seconds')

    # Calculate feature importance for the current model
    if hasattr(model, 'feature_importances_'):
        feature_importance[model_name] = model.feature_importances_
    else:
        feature_importance[model_name] = None

    y_pred_train3 = model.predict(X_train3)
    y_pred_test3 = model.predict(X_test3)

    accuracy_train[model_name] = accuracy_score(np.array(y_train3),␣
 ↪y_pred_train3)
    accuracy_test[model_name] = accuracy_score(np.array(y_test3), y_pred_test3)

    confusion_matrix_train[model_name] = confusion_matrix(np.array(y_train3),␣
 ↪y_pred_train3)
    confusion_matrix_test[model_name] = confusion_matrix(np.array(y_test3),␣
 ↪y_pred_test3)

# We create a dataframe showing the accuracy results in training and testing.
df_accuracy_train = pd.DataFrame.from_dict(accuracy_train, orient='index').
 ↪rename(columns={0: 'Train'})
df_accuracy_test = pd.DataFrame.from_dict(accuracy_test, orient='index').
 ↪rename(columns={0: 'Test'})
df_accuracy = pd.merge(df_accuracy_train, df_accuracy_test, left_index=True,␣
 ↪right_index=True)
df_accuracy = df_accuracy.sort_values(['Train', 'Test'], ascending=False)
df_accuracy

# Display feature importance for each model
for model_name, importance_scores in feature_importance.items():
    print(f"Feature Importance for {model_name}:\n{importance_scores}\n")
```

[ ]:
```
df_accuracy
```

[ ]:

	Train	Test
Random_Forest	1.000000	0.50830
Extra_Trees	1.000000	0.50020

```
CatBoost             0.685213  0.50165
KNN                  0.684500  0.50255
XGB                  0.663100  0.49615
Voting               0.655525  0.50445
LGBM                 0.634988  0.50530
HistGrad             0.562588  0.50505
Stacking             0.510925  0.50200
Logistic_Regression  0.501763  0.50400
```

4. Model Optimization:

- Fine-tune the model parameters to improve its predictive performance.
- Explore techniques like cross-validation and hyperparameter tuning.

[ ]:
```python
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Define your HistGradientBoostingClassifier with default hyperparameters
hist_grad = HistGradientBoostingClassifier(random_state=42)

# Define the hyperparameter grid to search
param_grid_hist_grad = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_iter': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
    'max_bins': [100, 255, 512],
    'l2_regularization': [0.1, 0.01, 0.001]
}

# Create a GridSearchCV instance with cross-validation
grid_search_hist_grad = GridSearchCV(hist_grad, param_grid_hist_grad, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)

# Fit the grid search to your training data
grid_search_hist_grad.fit(X_train3, y_train3)

# Get the best hyperparameters
best_params_hist_grad = grid_search_hist_grad.best_params_

# Initialize the HistGradientBoostingClassifier with the best hyperparameters
best_hist_grad = HistGradientBoostingClassifier(**best_params_hist_grad,␣
 ↪random_state=42)

# Train the final model on the entire training dataset
best_hist_grad.fit(X_train3, y_train3)
```

```
# Make predictions on the validation set
y_val_pred_hist_grad = best_hist_grad.predict(X_test3)

# Evaluate the model's performance on the validation set
accuracy_hist_grad = accuracy_score(y_test3, y_val_pred_hist_grad)
print(f"HistGrad Validation Accuracy: {accuracy_hist_grad:.2f}")
```

```
HistGrad Validation Accuracy: 0.50
```

[ ]:
```
from sklearn.neighbors import KNeighborsClassifier

# Define your K-Nearest Neighbors (KNN) Classifier with default hyperparameters
knn = KNeighborsClassifier()

# Define the hyperparameter grid to search
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],  # Adjust the number of neighbors to test
    'weights': ['uniform', 'distance'],  # Weighting scheme
    'p': [1, 2]  # Minkowski distance power (1 for Manhattan, 2 for Euclidean)
}

# Create a GridSearchCV instance with cross-validation
grid_search_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='accuracy',␣
↪n_jobs=-1)

# Fit the grid search to your training data
grid_search_knn.fit(X_train3, y_train3)

# Get the best hyperparameters
best_params_knn = grid_search_knn.best_params_

# Initialize the K-Nearest Neighbors (KNN) model with the best hyperparameters
best_knn = KNeighborsClassifier(**best_params_knn)

# Train the final KNN model on the entire training dataset
best_knn.fit(X_train3, y_train3)

# Make predictions on the validation set
y_val_pred_knn = best_knn.predict(X_test3)

# Evaluate the KNN model's performance on the validation set
accuracy_knn = accuracy_score(y_test3, y_val_pred_knn)
print(f"KNN Validation Accuracy: {accuracy_knn:.2f}")
```

```
KNN Validation Accuracy: 0.50
```

[ ]:
```
# lets try my xgb method here:
# X=X.drop('user-definedlabeln',axis=1)
# y=one_hot_encoded_data['user-definedlabeln']
# X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.
↪30,random_state=30)


gbm_param_grid = {
    'n_estimators': range(10, 100,10),
    'max_depth': range(3, 5),
    'learning_rate': [.4, .45, .5, .55, .6],
    'colsample_bytree': [.6, .7, .8, .9, 1]
}
gbm = XGBClassifier(n_estimators=100)
xgb_random = RandomizedSearchCV(param_distributions=gbm_param_grid,
                                    estimator = gbm, scoring = "accuracy",
                                    verbose = 1, n_iter = 50, cv = 5)
xgb_random.fit(X_normalized, y)
rfc3=xgb_random.predict(X_test)
# Print the best parameters and lowest RMSE
print("Best parameters found: ", xgb_random.best_params_,end="\n\n")
print("Best accuracy found: ", xgb_random.best_score_,end="\n\n")

print("accuracy=",accuracy_score(y_test,rfc3),end="\n\n")
print("confusion matrix:\n\n",classification_report(y_test,rfc3,digits=5))
```

[ ]:
```
# import pandas as pd

# Assuming you have a DataFrame 'df' with the original features: Age,␣
↪Subscription_Length_Months, Monthly_Bill, Total_Usage_GB
# Replace 'your_dataset.csv' with the path to your dataset
# df = pd.read_csv('your_dataset.csv')

# Feature Engineering

# 1. Average Monthly Data Usage
df['Average_Monthly_Data_Usage'] = df['Total_Usage_GB'] /␣
↪df['Subscription_Length_Months']

# 2. Billing Change Rate
df['Billing_Change_Rate'] = df['Monthly_Bill'].diff()

# 3. Billing Amount as a Percentage
df['Billing_As_Percentage'] = (df['Monthly_Bill'] / df['Monthly_Bill'].mean())␣
↪* 100

# 4. Customer Tenure in Months
```

```
df['Customer_Tenure_Months'] = df['Subscription_Length_Months']

# 5. Churn History (Assuming 'Churn' is a binary column indicating churn␣
↪history)
df['Churn_History'] = df['Churn'].shift(1)  # Lagged version of the churn column

# 6. Age Group Indicator (Assuming age groups are defined)
age_bins = [0, 30, 50, 100]  # Define your age groups as needed
age_labels = ['Young', 'Middle-Aged', 'Senior']
df['Age_Group_Indicator'] = pd.cut(df['Age'], bins=age_bins, labels=age_labels)

# 7. Remaining Subscription Length
df['Remaining_Subscription_Length'] = df['Subscription_Length_Months'] - df.
↪index

# 8. Average Bill Change
df['Average_Bill_Change'] = df['Billing_Change_Rate'].rolling(window=3).mean()

# Display the updated DataFrame with engineered features
print(df.head())
```

```
   CustomerID        Name  Age  Gender     Location  \
0           1  Customer_1   63    Male  Los Angeles
1           2  Customer_2   62  Female     New York
2           3  Customer_3   24  Female  Los Angeles
3           4  Customer_4   36  Female        Miami
4           5  Customer_5   46  Female        Miami

   Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
0                          17         73.36             236      0
1                           1         48.76             172      0
2                           5         85.47             460      0
3                           3         97.94             297      1
4                          19         58.14             266      0

   Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
0                   13.882353                  NaN             112.769247
1                  172.000000               -24.60              74.954041
2                   92.000000                36.71             131.384781
3                   99.000000                12.47             150.553708
4                   14.000000               -39.80              89.373010

   Customer_Tenure_Months  Churn_History Age_Group_Indicator  \
0                      17            NaN              Senior
1                       1            0.0              Senior
2                       5            0.0               Young
3                       3            0.0         Middle-Aged
```

```
4                      19           1.0         Middle-Aged

   Remaining_Subscription_Length  Average_Bill_Change
0                             17                  NaN
1                              0                  NaN
2                              3                  NaN
3                              0             8.193333
4                             15             3.126667
```

[ ]:
```
df.tail(1)
```

[ ]:
```
       CustomerID             Name  Age  Subscription_Length_Months  \
99999      100000  Customer_100000   27                          19

       Monthly_Bill  Total_Usage_GB  Churn  Average_Monthly_Data_Usage  \
99999         76.57             173      1                    9.105263

       Billing_Change_Rate  Billing_As_Percentage  Customer_Tenure_Months  \
99999                27.32              117.70367                      19

       Churn_History  Age_Group_Indicator  Remaining_Subscription_Length  \
99999            1.0                    2                         -99980

       Average_Bill_Change  Gender_Female  Gender_Male  Location_Chicago  \
99999             4.973333              1            0                 0

       Location_Houston  Location_Los Angeles  Location_Miami  \
99999                 0                     1               0

       Location_New York
99999                  0
```

[ ]:
```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Assuming you have engineered the additional features and added them to your␣
 ↪dataset
# Replace 'your_dataset.csv' with the path to your dataset
# data = pd.read_csv('your_dataset.csv')

# Define your features and target variable
features = ['Age', 'Subscription_Length_Months', 'Monthly_Bill',␣
 ↪'Total_Usage_GB',  # Original features
            'Average_Monthly_Data_Usage', 'Billing_Change_Rate',␣
 ↪'Billing_As_Percentage',
            'Customer_Tenure_Months', 'Churn_History', 'Age_Group_Indicator',
            'Remaining_Subscription_Length', 'Average_Bill_Change']
```

```python
X = df[features]  # Features
y = df['Churn']  # Target variable

# Create an imputer instance (for example, using mean imputation)

# Fit and transform the imputer on your feature data
X_imputed = imputer.fit_transform(X)

# Now, you can use X_imputed in your model

# Initialize and train a Random Forest model
model = RandomForestClassifier(random_state=42)
model.fit(X_imputed, y)

# Get feature importances
feature_importances = model.feature_importances_

# Create a DataFrame to display feature importances
importance_df = pd.DataFrame({'Feature': features, 'Importance':
 ↪feature_importances})

# Sort features by importance (highest to lowest)
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Print the feature importances
print(importance_df)
```

```
                            Feature  Importance
10  Remaining_Subscription_Length    0.121874
5             Billing_Change_Rate    0.120788
11            Average_Bill_Change    0.120777
4       Average_Monthly_Data_Usage   0.109554
3                   Total_Usage_GB   0.106813
6           Billing_As_Percentage    0.104632
2                     Monthly_Bill   0.104546
0                              Age   0.086257
1       Subscription_Length_Months   0.048144
7           Customer_Tenure_Months   0.047611
8                    Churn_History   0.015857
9              Age_Group_Indicator   0.013146
```

4 4 Model Optimization:

- Fine-tune the model parameters to improve its predictive performance.
- Explore techniques like cross-validation and hyperparameter tuning.

Best Performing Model XGBoost Validation Accuracy : 0.5157

Feature Importances:

Remaining_Subscription_Length (0.1219): This feature has the highest importance score. It suggests that the remaining subscription length is a significant predictor of customer churn. Customers with shorter remaining subscription lengths might be more likely to churn.

Billing_Change_Rate (0.1208): Billing change rate is the second most important feature. It indicates that fluctuations in the monthly billing amount play a crucial role in predicting churn. Higher billing change rates might be associated with a higher likelihood of churn.

Average_Bill_Change (0.1208): This feature is very similar in importance to Billing_Change_Rate. It also captures billing fluctuations but might represent a smoothed or averaged version of those changes.

Average_Monthly_Data_Usage (0.1096): The average monthly data usage is the fourth most important feature. It suggests that customers' data usage patterns can impact their likelihood of churning. Higher data usage may correlate with lower churn rates.

Total_Usage_GB (0.1068): This feature represents the total data usage in gigabytes and is also important. It's related to Average_Monthly_Data_Usage but considers the overall data consumption.

Billing_As_Percentage (0.1046): The percentage of the monthly bill relative to some reference value is the sixth most important feature. This might capture information about customers' budget constraints or sensitivity to price changes.

Monthly_Bill (0.1045): The actual monthly billing amount is an important feature. High monthly bills could be a factor leading to churn.

Age (0.0863): Age is also significant but less so than some of the billing-related features. It indicates that customer age plays a role in predicting churn, with younger or older customers potentially having different behaviors.

Subscription_Length_Months (0.0481): Subscription length in months is less important compared to other features. However, it still contributes to the model's predictions, suggesting that longer subscription commitments might reduce churn.

Customer_Tenure_Months (0.0476): Customer tenure, or how long a customer has been with the provider, is also less important but still relevant. Longer tenure might indicate loyalty and reduce churn.

Churn_History (0.0159): Churn history, which captures whether a customer has churned in the past, has some importance. It's less critical than the other features, indicating that current behavior and characteristics are more predictive.

Age_Group_Indicator (0.0131): Age group indicator has the lowest importance in your model. It suggests that the specific age group category might not be as crucial as the overall age feature.

[ ]:
```
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
```

```python
# Define your HistGradientBoostingClassifier with default hyperparameters
hist_grad = HistGradientBoostingClassifier(random_state=42)

# Define the hyperparameter grid to search
param_grid_hist_grad = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_iter': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'min_samples_leaf': [1, 2, 3],
    'max_bins': [100, 255, 512],
    'l2_regularization': [0.1, 0.01, 0.001]
}

# Create a GridSearchCV instance with cross-validation
grid_search_hist_grad = GridSearchCV(hist_grad, param_grid_hist_grad, cv=5,␣
 ↪scoring='accuracy', n_jobs=-1)

# Fit the grid search to your training data
grid_search_hist_grad.fit(X_train3, y_train3)

# Get the best hyperparameters
best_params_hist_grad = grid_search_hist_grad.best_params_

# Initialize the HistGradientBoostingClassifier with the best hyperparameters
best_hist_grad = HistGradientBoostingClassifier(**best_params_hist_grad,␣
 ↪random_state=42)

# Train the final model on the entire training dataset
best_hist_grad.fit(X_train3, y_train3)

# Make predictions on the validation set
y_val_pred_hist_grad = best_hist_grad.predict(X_test3)

# Evaluate the model's performance on the validation set
accuracy_hist_grad = accuracy_score(y_test3, y_val_pred_hist_grad)
print(f"HistGrad Validation Accuracy: {accuracy_hist_grad:.2f}")
```

[ ]:

```python
from sklearn.neighbors import KNeighborsClassifier

# Define your K-Nearest Neighbors (KNN) Classifier with default hyperparameters
knn = KNeighborsClassifier()

# Define the hyperparameter grid to search
param_grid_knn = {
    'n_neighbors': [3, 5, 7, 9],  # Adjust the number of neighbors to test
    'weights': ['uniform', 'distance'],  # Weighting scheme
```

```
    'p': [1, 2]  # Minkowski distance power (1 for Manhattan, 2 for Euclidean)
}

# Create a GridSearchCV instance with cross-validation
grid_search_knn = GridSearchCV(knn, param_grid_knn, cv=5, scoring='accuracy',␣
↪n_jobs=-1)

# Fit the grid search to your training data
grid_search_knn.fit(X_train3, y_train3)

# Get the best hyperparameters
best_params_knn = grid_search_knn.best_params_

# Initialize the K-Nearest Neighbors (KNN) model with the best hyperparameters
best_knn = KNeighborsClassifier(**best_params_knn)

# Train the final KNN model on the entire training dataset
best_knn.fit(X_train3, y_train3)

# Make predictions on the validation set
y_val_pred_knn = best_knn.predict(X_test3)

# Evaluate the KNN model's performance on the validation set
accuracy_knn = accuracy_score(y_test3, y_val_pred_knn)
print(f"KNN Validation Accuracy: {accuracy_knn:.2f}")
```

[ ]:
```
!pip install pycaret -q
```

[ ]:
```
# pycaret trial
import pycaret.anomaly as anm
import pycaret.classification as cl
import pandas as pd
```

[ ]:
```
dfsample = df.sample(frac=1)
dfsample
```

[ ]:

	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn	\
64111	55	22	74.13	135	1	
67350	20	18	45.55	316	1	
77239	36	18	74.98	58	1	
39551	51	19	51.63	114	0	
44014	66	24	91.28	466	0	
…	…	…	…	…	…	
79659	57	17	42.56	166	1	
33394	69	20	60.33	242	1	
44770	54	24	49.38	94	0	
4910	39	21	87.52	142	1	

```
13157   65                           19          72.81            498       0

        Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
64111                     6.136364               -16.10             113.952893
67350                    17.555556               -54.10              70.019618
77239                     3.222222                 3.88             115.259516
39551                     6.000000                16.17              79.365815
44014                    19.416667                20.81             140.315933
…                              …                   …                  …
79659                     9.764706               -10.92              65.423380
33394                    12.100000               -22.78              92.739485
44770                     3.916667                 1.16              75.907107
4910                      6.761905                45.80             134.536048
13157                    26.210526                24.41             111.923785

        Customer_Tenure_Months  Churn_History  Age_Group_Indicator  \
64111                       22            0.0                    1
67350                       18            0.0                    2
77239                       18            1.0                    0
39551                       19            1.0                    1
44014                       24            0.0                    1
…                        …             …                    …
79659                       17            1.0                    1
33394                       20            1.0                    1
44770                       24            0.0                    1
4910                        21            1.0                    0
13157                       19            1.0                    1

        Remaining_Subscription_Length  Average_Bill_Change  Gender_Female  \
64111                          -64089             5.356667              1
67350                          -67332            -0.523333              0
77239                          -77221             8.626667              0
39551                          -39532             4.270000              0
44014                          -43990             4.756667              0
…                                 …                  …             …
79659                          -79642             1.670000              1
33394                          -33374             5.013333              1
44770                          -44746             5.803333              1
4910                            -4889            12.240000              0
13157                          -13138            -8.413333              1

        Gender_Male  Location_Chicago  Location_Houston  Location_Los Angeles  \
64111             0                 0                 0                     0
67350             1                 0                 0                     0
77239             1                 0                 0                     0
39551             1                 0                 0                     1
44014             1                 0                 1                     0
```

```
…              …                …                …                    …
79659          0                1                0                    0
33394          0                1                0                    0
44770          0                0                0                    1
4910           1                0                1                    0
13157          0                0                0                    1

       Location_Miami  Location_New York
64111               0                  1
67350               0                  1
77239               0                  1
39551               0                  0
44014               0                  0
…                 …                  …
79659               0                  0
33394               0                  0
44770               0                  0
4910                0                  0
13157               0                  0

[100000 rows x 20 columns]
```

[ ]:
```
sizetest = int(0.2*(df.shape[0]))
dfTrain = dfsample[:-sizetest]
dfTest = dfsample[-sizetest:]

an = anm.setup(dfTrain)
model = anm.create_model('histogram')
preds = anm.assign_model(model)
preds = preds.drop('Anomaly',axis=1)
preds
```

```
<pandas.io.formats.style.Styler at 0x7e0ca6714cd0>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Processing:   0%|          | 0/3 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[ ]:
```
       Age  Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
64111   55                          22     74.129997             135      1
67350   20                          18     45.549999             316      1
77239   36                          18     74.980003              58      1
39551   51                          19     51.630001             114      0
44014   66                          24     91.279999             466      0
…      …                          …          …                …      …
```

```
93562  68                      11   47.500000         452     1
49046  39                       1   42.970001          52     0
18891  55                      13   39.650002         170     0
20680  44                       7   35.930000         150     1
71895  56                      12   59.110001         172     0

       Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
64111                    6.136364           -16.100000             113.952896
67350                   17.555555           -54.099998              70.019615
77239                    3.222222             3.880000             115.259514
39551                    6.000000            16.170000              79.365814
44014                   19.416666            20.809999             140.315933
…                             …                  …                      …
93562                   41.090908           -11.540000              73.017166
49046                   52.000000           -50.720001              66.053635
18891                   13.076923           -43.849998              60.950119
20680                   21.428572           -58.310001              55.231720
71895                   14.333333           -28.870001              90.864098

       Customer_Tenure_Months  Churn_History  Age_Group_Indicator  \
64111                      22            0.0                    1
67350                      18            0.0                    2
77239                      18            1.0                    0
39551                      19            1.0                    1
44014                      24            0.0                    1
…                         …              …                  …
93562                      11            1.0                    1
49046                       1            0.0                    0
18891                      13            1.0                    1
20680                       7            0.0                    0
71895                      12            1.0                    1

       Remaining_Subscription_Length  Average_Bill_Change  Gender_Female  \
64111                         -64089             5.356667              1
67350                         -67332            -0.523333              0
77239                         -77221             8.626667              0
39551                         -39532             4.270000              0
44014                         -43990             4.756667              0
…                                …                  …              …
93562                         -93551            -0.123333              1
49046                         -49045           -18.113333              1
18891                         -18878             1.946667              1
20680                         -20673           -16.766666              1
71895                         -71883            -9.160000              0

       Gender_Male  Location_Chicago  Location_Houston  Location_Los Angeles  \
64111            0                 0                 0                     0
```

```
67350           1               0               0                 0
77239           1               0               0                 0
39551           1               0               0                 1
44014           1               0               1                 0
…             …             …             …             …
93562           0               1               0                 0
49046           0               0               0                 0
18891           0               0               1                 0
20680           0               0               0                 0
71895           1               0               1                 0

       Location_Miami  Location_New York  Anomaly_Score
64111               0                  1      11.414200
67350               0                  1       8.010435
77239               0                  1       7.468376
39551               0                  0      11.714371
44014               0                  0      11.352058
…                 …                …            …
93562               0                  0      11.615839
49046               0                  1       7.544109
18891               0                  0      11.751043
20680               0                  1       7.774417
71895               0                  0      11.810555

[80000 rows x 21 columns]
```

[ ]:
```
classif = cl.setup(preds,target='Churn',normalize=True,␣
 ↪normalize_method='minmax',fold=9)
top3 = classif.compare_models(n_select=9,␣
 ↪fold=9,include=['ada','gbc','et','lr','rf','mlp','xgboost','lightgbm','catboost'],turbo=Tru
preds
```

```
<pandas.io.formats.style.Styler at 0x7e0c62ce29e0>

<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7e0c62c8f130>

Processing:   0%|          | 0/49 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

[ ]:
```
       Age  Subscription_Length_Months  Monthly_Bill  Total_Usage_GB  Churn  \
64111   55                          22     74.129997             135      1
67350   20                          18     45.549999             316      1
77239   36                          18     74.980003              58      1
39551   51                          19     51.630001             114      0
44014   66                          24     91.279999             466      0
…     …                         …          …                …    …
```

```
93562   68                           11     47.500000            452       1
49046   39                            1     42.970001             52       0
18891   55                           13     39.650002            170       0
20680   44                            7     35.930000            150       1
71895   56                           12     59.110001            172       0

       Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
64111                    6.136364           -16.100000             113.952896
67350                   17.555555           -54.099998              70.019615
77239                    3.222222             3.880000             115.259514
39551                    6.000000            16.170000              79.365814
44014                   19.416666            20.809999             140.315933
…                             …                  …                     …
93562                   41.090908           -11.540000              73.017166
49046                   52.000000           -50.720001              66.053635
18891                   13.076923           -43.849998              60.950119
20680                   21.428572           -58.310001              55.231720
71895                   14.333333           -28.870001              90.864098

       Customer_Tenure_Months  Churn_History  …  \
64111                      22            0.0  …
67350                      18            0.0  …
77239                      18            1.0  …
39551                      19            1.0  …
44014                      24            0.0  …
…                         …              …  …
93562                      11            1.0  …
49046                       1            0.0  …
18891                      13            1.0  …
20680                       7            0.0  …
71895                      12            1.0  …

       Remaining_Subscription_Length  Average_Bill_Change  Gender_Female  \
64111                         -64089             5.356667              1
67350                         -67332            -0.523333              0
77239                         -77221             8.626667              0
39551                         -39532             4.270000              0
44014                         -43990             4.756667              0
…                                …                  …             …
93562                         -93551            -0.123333              1
49046                         -49045           -18.113333              1
18891                         -18878             1.946667              1
20680                         -20673           -16.766666              1
71895                         -71883            -9.160000              0

       Gender_Male  Location_Chicago  Location_Houston  Location_Los Angeles  \
64111            0                 0                 0                     0
```

67350	1	0	0	0
77239	1	0	0	0
39551	1	0	0	1
44014	1	0	1	0
…	…	…	…	…
93562	0	1	0	0
49046	0	0	0	0
18891	0	0	1	0
20680	0	0	0	0
71895	1	0	1	0

	Location_Miami	Location_New York	Anomaly_Score
64111	0	1	11.414200
67350	0	1	8.010435
77239	0	1	7.468376
39551	0	0	11.714371
44014	0	0	11.352058
…	…	…	…
93562	0	0	11.615839
49046	0	1	7.544109
18891	0	0	11.751043
20680	0	1	7.774417
71895	0	0	11.810555

[80000 rows x 21 columns]

[ ]:
```
an = anm.setup(dfTest)
model = anm.create_model('histogram')
dfTestsAnom  = anm.assign_model(model)
dfTestsAnom  = dfTestsAnom.drop('Anomaly',axis=1)
dfTestsAnom
         # Description
```

<pandas.io.formats.style.Styler at 0x7e0c62d35ab0>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Processing: 0%| | 0/3 [00:00<?, ?it/s]

<IPython.core.display.HTML object>

[ ]:

	Age	Subscription_Length_Months	Monthly_Bill	Total_Usage_GB	Churn \
15177	19	2	31.990000	123	0
75601	21	8	47.070000	484	0
60617	65	11	64.629997	65	1
1425	60	17	67.180000	117	1
41658	44	12	53.570000	381	0

```
…     …                          …          …              …     …
79659   57                            17      42.560001              166       1
33394   69                            20      60.330002              242       1
44770   54                            24      49.380001               94       0
4910    39                            21      87.519997              142       1
13157   65                            19      72.809998              498       0

       Average_Monthly_Data_Usage  Billing_Change_Rate  Billing_As_Percentage  \
15177                   61.500000           -16.510000              49.175140
75601                   60.500000            -3.480000              72.356171
60617                    5.909091            30.500000              99.349457
1425                     6.882353            22.910000             103.269333
41658                   31.750000            20.740000              82.347992
…                             …                 …                      …
79659                    9.764706           -10.920000              65.423378
33394                   12.100000           -22.780001              92.739487
44770                    3.916667             1.160000              75.907104
4910                     6.761905            45.799999             134.536041
13157                   26.210526            24.410000             111.923782

       Customer_Tenure_Months  Churn_History  …  \
15177                       2            0.0  …
75601                       8            0.0  …
60617                      11            0.0  …
1425                       17            0.0  …
41658                      12            1.0  …
…                         …            …   …
79659                      17            1.0  …
33394                      20            1.0  …
44770                      24            0.0  …
4910                       21            1.0  …
13157                      19            1.0  …

       Remaining_Subscription_Length  Average_Bill_Change  Gender_Female  \
15177                         -15175            -4.816667              0
75601                         -75593           -15.013333              1
60617                         -60606            -2.283333              1
1425                           -1408             8.823334              0
41658                         -41646             5.063334              0
…                               …                  …            …
79659                         -79642             1.670000              1
33394                         -33374             5.013333              1
44770                         -44746             5.803333              1
4910                           -4889            12.240000              0
13157                         -13138            -8.413333              1

       Gender_Male  Location_Chicago  Location_Houston  Location_Los Angeles  \
```

```
15177            1                 0                 0                     0
75601            0                 0                 0                     0
60617            0                 0                 0                     1
1425             1                 1                 0                     0
41658            1                 0                 0                     1
…            …                …                …                   …
79659            0                 1                 0                     0
33394            0                 1                 0                     0
44770            0                 0                 0                     1
4910             1                 0                 1                     0
13157            0                 0                 0                     1

       Location_Miami  Location_New York  Anomaly_Score
15177               0                  1       7.914317
75601               0                  1       8.042049
60617               0                  0      11.650193
1425                0                  0      11.467817
41658               0                  0       7.469678
…                 …                …            …
79659               0                  0      11.284363
33394               0                  0      11.684547
44770               0                  0      11.334445
4910                0                  0       7.652183
13157               0                  0      11.650511

[20000 rows x 21 columns]
```

[ ]:
```
finalpreds1 = classif.predict_model(top3[0],dfTestsAnom)
finalpreds2 = classif.predict_model(top3[1],dfTestsAnom)
finalpreds3 = classif.predict_model(top3[2],dfTestsAnom)
# finalpreds
```

```
<pandas.io.formats.style.Styler at 0x7e0c62e1da50>

<pandas.io.formats.style.Styler at 0x7e0c62b69bd0>

<pandas.io.formats.style.Styler at 0x7e0c62c4bf70>
```

saving our xgboost model

[ ]:
```
top3[0].save_model('/content/drive/MyDrive/internwork/xgb.bin')
```

[ ]:
```
from pycaret.classification import save_model

save_model(top3[0], '/content/drive/MyDrive/internwork/xgb_caret.bin')
save_model(top3[1], '/content/drive/MyDrive/internwork/catboost_caret.bin')
save_model(top3[1], '/content/drive/MyDrive/internwork/lgbm_caret.bin')
```

```
Transformation Pipeline and Model Successfully Saved
```

```
Transformation Pipeline and Model Successfully Saved
Transformation Pipeline and Model Successfully Saved
```

[ ]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['Age',
                                              'Subscription_Length_Months',
                                              'Monthly_Bill', 'Total_Usage_GB',
                                              'Average_Monthly_Data_Usage',
                                              'Billing_Change_Rate',
                                              'Billing_As_Percentage',
                                              'Customer_Tenure_Months',
                                              'Churn_History',
                                              'Age_Group_Indicator',
                                              'Remaining_Subscription_Length',
                                              'Avera…
verbose='deprecated'))),
                 ('normalize',
                  TransformerWrapper(exclude=None, include=None,
                                     transformer=MinMaxScaler(clip=False,
                                                              copy=True,
                                                              feature_range=(0,
1)))),
                 ('clean_column_names',
                  TransformerWrapper(exclude=None, include=None,
transformer=CleanColumnNames(match='[\\]\\[\\,\\{\\}\\"\\:]+'))),
                 ('trained_model',
                  <catboost.core.CatBoostClassifier object at 0x7e0c62b6a1a0>)],
          verbose=False),
 '/content/drive/MyDrive/internwork/lgbm_caret.bin.pkl')
```

[ ]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
acc = accuracy_score(list(finalpreds['Churn'].
 ↪values),list(finalpreds['prediction_label'].astype('int').values))
cfm = confusion_matrix(list(finalpreds['Churn'].
 ↪values),list(finalpreds['prediction_label'].astype('int').values))
print("Conf. Matrix\n",cfm,"\n")
print("Accuracy: ",acc*100,"%")
```

[ ]:
```
continuous=['Age', 'Subscription_Length_Months','Monthly_Bill',␣
 ↪'Total_Usage_GB']
categorical=[ 'Gender', 'Location']
```

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

```
# Create a new DataFrame for each class of sleepiness
df_churn_0 = df[df['Churn'] == 0]
df_churn_1 = df[df['Churn'] == 1]


import matplotlib.pyplot as plt
import seaborn as sns

# Calculate the number of rows needed based on the number of columns
num_cols = len(continuous)
num_plots_per_row = 2
# num_rows = int(np.ceil(num_cols / num_plots_per_row))
num_rows = 2

# Adjust the figure size based on the number of rows and plots per row
fig, axes = plt.subplots(num_rows, num_plots_per_row, figsize=(10, 10 *␣
 ↪num_rows))

# Flatten the axes if there's only one row
if num_rows == 1:
    axes = axes.reshape(1, -1)

for i, column in enumerate(continuous):
    row_idx = i // num_plots_per_row
    col_idx = i % num_plots_per_row

    # Create a single axis for each variable
    ax = axes[row_idx, col_idx]

    # Create boxplots for train and test data side by side
    sns.boxplot(data=[df_churn_0[column], df_churn_1[column]], ax=ax)
    ax.set_title(column)
    ax.set_xticklabels(['Churn 0', 'Churn 1'])

plt.tight_layout()
plt.show()
```

[ ]:
```
X_train = pd.DataFrame(X_train, columns=X.columns)
X_train_cont = X_train[continuous]  # Replace 'continous' with the actual list␣
 ↪of continuous columns

X_test = pd.DataFrame(X_test, columns=X.columns)
X_test_cont = X_test[continuous]  # Replace 'continous' with the actual list of␣
 ↪continuous columns
```

5 5 Deployment - Done with Flask and Azure

- Once satisfied with the model's performance, deploy it into a production-like environment (you can simulate this in a development environment).
- Ensure the model can take new customer data as input and provide churn predictions.

Deployed API Link:

https://sunbase.azurewebsites.net/predict

POST API

Body:

```
# Sample

{
    "age": 30,
    "subscription_length_months": 40,
    "monthly_bill": 76.57,
    "total_usage_gb": 173,
    "gender":"Female",
    "location":"Houston"
}
```

Python Flask Script:

```
# This is formatted as code
```

Requirements.txt file:

```
# This is formatted as code
```

Instructions to run locally

1. Create a folder with some name
2. Create 2 files app.py and requirements.txt
3. Run

```
pip install -r requirements.txt
python app.py
```

4. Now open a software to make a post request and use the above code as a sample