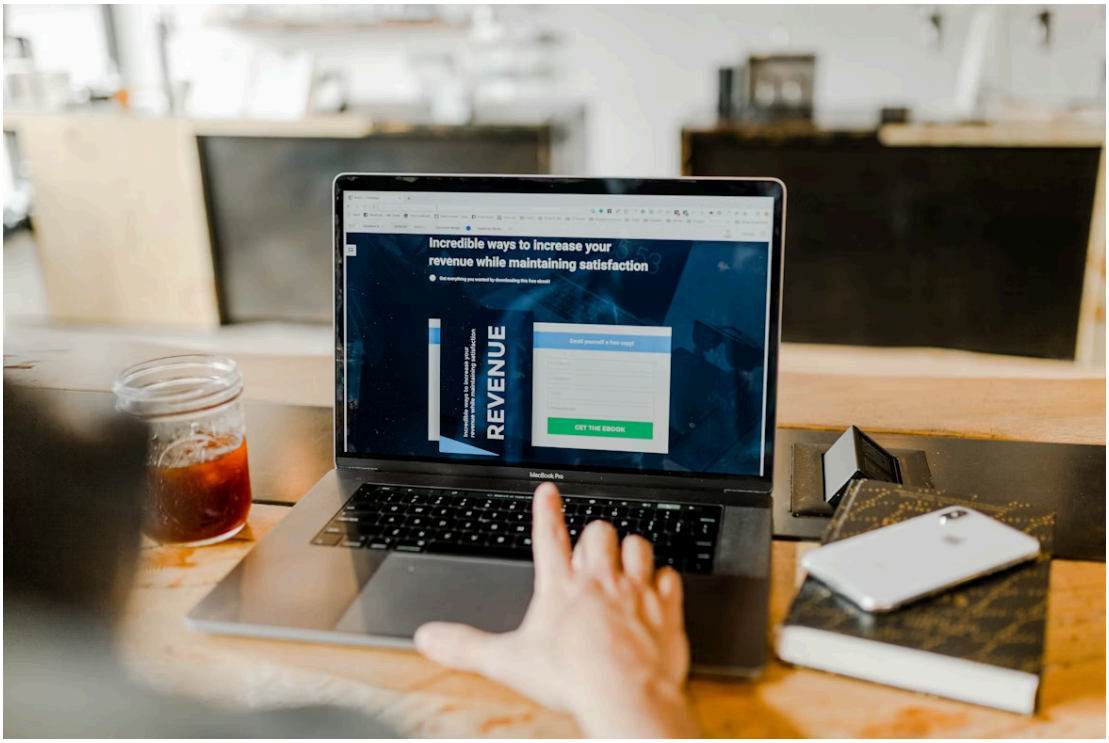
PREDICTIVE MODELING FOR CUSTOMER LIFETIME VALUE IN TELECOMMUNICATIONS INDUSTRY

Business Overview

In the telecom industry, understanding Customer Lifetime Value (CLTV) holds paramount importance as it quantifies the total revenue anticipated from a customer throughout their engagement period. This comprehension of CLTV plays a pivotal role in steering strategic decisions concerning both customer acquisition and retention.

Within our telecom-focused endeavors, this project falls under the `telecom data` cluster. The analysis of CLTV empowers telecom companies to discern high-value customers, tailor retention strategies, and optimize the allocation of marketing resources. Furthermore, it aids in refining sales tactics, pricing strategies, enhancing customer satisfaction, and mitigating churn rates.

The identification of high-value customers facilitates the customization of marketing and sales approaches, nurturing these relationships to curtail churn rates and foster long-term loyalty. Additionally, CLTV analysis serves as a guiding beacon for telecom companies in refining pricing strategies by pinpointing the most valued services, allowing for adjustments that maximize revenue and profitability. Moreover, it offers invaluable insights into enhancing overall customer satisfaction by identifying critical factors and areas for improvement, enabling targeted enhancements to ensure sustained satisfaction and loyalty.



Package Requirements

Avoid installing the libraries in the next cell if you have previously run the requirements.txt file:

```
pip install -r requirements.txt
```

In [1]:
```
# # Installing packages
# !pip install scipy
# !pip install seaborn
# !pip install numpy
# !pip install pandas==2.0.1
# !pip install xgboost==1.7.4
# !pip install matplotlib==3.7.1
# !pip install deepchecks==0.12.0
# !pip install scikit_learn==1.2.2
```

After installing the packages, please restart the runtime if required.

I've added some lines to ensure that this notebook can run both on Google Colab and locally on my machine.

Important Libraries

In [2]:
```
# Importing libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sys
from scipy.stats import ttest_ind


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

In [3]:
```
# Configuring display options for improved data viewing with large datasets
pd.set_option('display.max_columns', 200)  # Set maximum displayed columns to 20
pd.set_option('display.max_rows', 300)     # Set maximum displayed rows to 300
```

In [4]:
```
# Works if I'm working on Google Colab, else it passes on anaconda.
try:
    # Attempting to import the drive module from google.colab
    from google.colab import drive

    # Mounting Google Drive
    drive.mount('/content/drive')
    print("Google Drive mounted successfully!")
except Exception as e:
    # Handling the case when the import fails, likely due to working outside of
    print("Error occurred while mounting Google Drive:", e)
    pass
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dri
ve.mount("/content/drive", force_remount=True).
Google Drive mounted successfully!
```

Data Exploration

In [5]:
```
# Reading data from a CSV file
try:
    # Reading data from a CSV file on local machine
    df = pd.read_csv('../data/processed/telecom_data.csv')
    print("CSV file read successfully from local machine!")
except FileNotFoundError:
    try:
        # Reading data from a CSV file hosted on Google Drive
        df = pd.read_csv('/content/drive/Othercomputers/MyLaptop/git_repos/proje
        print("CSV file read successfully from Google Drive!")
    except Exception as e:
        print("An error occurred while reading the CSV file:", e)
```

```
CSV file read successfully from Google Drive!
```

In [6]:
```
# Displaying the first few rows of the DataFrame
df.head()
```

Out[6]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents
0	hthjctifkiudi0	1	1.0	71638	Female	36.000000	No	No
1	uqdtniwvxqzeu1	6	6.0	72566	Male	36.657198	No	No
2	uqdtniwvxqzeu1	7	6.0	72566	Male	36.607828	No	No
3	uqdtniwvxqzeu1	8	6.0	72566	Male	36.943638	No	No
4	uqdtniwvxqzeu1	9	6.0	72566	Male	36.632494	No	No

In [7]:
```
# Information on nulls and dtypes
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 653753 entries, 0 to 653752
Data columns (total 74 columns):
 #   Column                      Non-Null Count   Dtype  
---  ------                      --------------   -----  
 0   Customer ID                 653753 non-null  object 
 1   Month                       653753 non-null  int64  
 2   Month of Joining            653753 non-null  float64
 3   zip_code                    653753 non-null  int64  
 4   Gender                      653753 non-null  object 
 5   Age                         653753 non-null  float64
 6   Married                     653753 non-null  object 
 7   Dependents                  653753 non-null  object 
 8   Number of Dependents        648808 non-null  float64
 9   Location ID                 653753 non-null  object 
 10  Service ID                  653753 non-null  object 
 11  state                       653753 non-null  object 
 12  county                      653753 non-null  object 
 13  timezone                    653753 non-null  object 
 14  area_codes                  653753 non-null  object 
 15  country                     653753 non-null  object 
 16  latitude                    653753 non-null  float64
 17  longitude                   653753 non-null  float64
 18  arpu                        653753 non-null  float64
 19  roam_ic                     653753 non-null  float64
 20  roam_og                     653753 non-null  float64
 21  loc_og_t2t                  653753 non-null  float64
 22  loc_og_t2m                  653753 non-null  float64
 23  loc_og_t2f                  653753 non-null  float64
 24  loc_og_t2c                  653753 non-null  float64
 25  std_og_t2t                  653753 non-null  float64
 26  std_og_t2m                  653753 non-null  float64
 27  std_og_t2f                  653753 non-null  float64
 28  std_og_t2c                  653753 non-null  float64
 29  isd_og                      653753 non-null  float64
 30  spl_og                      653753 non-null  float64
 31  og_others                   653753 non-null  float64
 32  loc_ic_t2t                  653753 non-null  float64
 33  loc_ic_t2m                  653753 non-null  float64
 34  loc_ic_t2f                  653753 non-null  float64
 35  std_ic_t2t                  653753 non-null  float64
 36  std_ic_t2m                  653753 non-null  float64
 37  std_ic_t2f                  653753 non-null  float64
 38  std_ic_t2o                  653753 non-null  float64
 39  spl_ic                      653753 non-null  float64
 40  isd_ic                      653753 non-null  float64
 41  ic_others                   653753 non-null  float64
 42  total_rech_amt              653753 non-null  float64
 43  total_rech_data             443849 non-null  float64
 44  vol_4g                      653753 non-null  float64
 45  vol_5g                      653753 non-null  float64
 46  arpu_5g                     653753 non-null  object 
 47  arpu_4g                     653753 non-null  object 
 48  night_pck_user              280650 non-null  float64
 49  fb_user                     243359 non-null  float64
 50  aug_vbc_5g                  653753 non-null  float64
 51  Churn Value                 653753 non-null  int64  
 52  Referred a Friend           653753 non-null  object 
 53  Number of Referrals         653366 non-null  float64
 54  Phone Service               653753 non-null  object 
```

```
 55  Multiple Lines              607673 non-null  object
 56  Internet Service            653753 non-null  object
 57  Internet Type               328503 non-null  object
 58  Streaming Data Consumption  653753 non-null  int64
 59  Online Security             653753 non-null  object
 60  Online Backup               653753 non-null  object
 61  Device Protection Plan      653753 non-null  object
 62  Premium Tech Support        653753 non-null  object
 63  Streaming TV                653753 non-null  object
 64  Streaming Movies            653753 non-null  object
 65  Streaming Music             653753 non-null  object
 66  Unlimited Data              642650 non-null  object
 67  Payment Method              653753 non-null  object
 68  Status ID                   653753 non-null  object
 69  Satisfaction Score          653753 non-null  int64
 70  Churn Category              653753 non-null  object
 71  Churn Reason                653753 non-null  object
 72  Customer Status             653753 non-null  object
 73  offer                       653753 non-null  object
dtypes: float64(37), int64(5), object(32)
memory usage: 369.1+ MB
```

In [8]:
```
# Displaying all column names in the DataFrame
df.columns
```

Out[8]:
```
Index(['Customer ID', 'Month', 'Month of Joining', 'zip_code', 'Gender', 'Age',
       'Married', 'Dependents', 'Number of Dependents', 'Location ID',
       'Service ID', 'state', 'county', 'timezone', 'area_codes', 'country',
       'latitude', 'longitude', 'arpu', 'roam_ic', 'roam_og', 'loc_og_t2t',
       'loc_og_t2m', 'loc_og_t2f', 'loc_og_t2c', 'std_og_t2t', 'std_og_t2m',
       'std_og_t2f', 'std_og_t2c', 'isd_og', 'spl_og', 'og_others',
       'loc_ic_t2t', 'loc_ic_t2m', 'loc_ic_t2f', 'std_ic_t2t', 'std_ic_t2m',
       'std_ic_t2f', 'std_ic_t2o', 'spl_ic', 'isd_ic', 'ic_others',
       'total_rech_amt', 'total_rech_data', 'vol_4g', 'vol_5g', 'arpu_5g',
       'arpu_4g', 'night_pck_user', 'fb_user', 'aug_vbc_5g', 'Churn Value',
       'Referred a Friend', 'Number of Referrals', 'Phone Service',
       'Multiple Lines', 'Internet Service', 'Internet Type',
       'Streaming Data Consumption', 'Online Security', 'Online Backup',
       'Device Protection Plan', 'Premium Tech Support', 'Streaming TV',
       'Streaming Movies', 'Streaming Music', 'Unlimited Data',
       'Payment Method', 'Status ID', 'Satisfaction Score', 'Churn Category',
       'Churn Reason', 'Customer Status', 'offer'],
      dtype='object')
```

In [9]:
```
# Display the shape of the dataframe
df.shape
```

Out[9]:
```
(653753, 74)
```

In [10]:
```
# Display the information of the dataframe
df.isna().sum()
```

Out[10]:

Customer ID	0
Month	0
Month of Joining	0
zip_code	0
Gender	0
Age	0
Married	0
Dependents	0
Number of Dependents	4945
Location ID	0
Service ID	0
state	0
county	0
timezone	0
area_codes	0
country	0
latitude	0
longitude	0
arpu	0
roam_ic	0
roam_og	0
loc_og_t2t	0
loc_og_t2m	0
loc_og_t2f	0
loc_og_t2c	0
std_og_t2t	0
std_og_t2m	0
std_og_t2f	0
std_og_t2c	0
isd_og	0
spl_og	0
og_others	0
loc_ic_t2t	0
loc_ic_t2m	0
loc_ic_t2f	0
std_ic_t2t	0
std_ic_t2m	0
std_ic_t2f	0
std_ic_t2o	0
spl_ic	0
isd_ic	0
ic_others	0
total_rech_amt	0
total_rech_data	209904
vol_4g	0
vol_5g	0
arpu_5g	0
arpu_4g	0
night_pck_user	373103
fb_user	410394
aug_vbc_5g	0
Churn Value	0
Referred a Friend	0
Number of Referrals	387
Phone Service	0
Multiple Lines	46080
Internet Service	0
Internet Type	325250
Streaming Data Consumption	0
Online Security	0

```
Online Backup                     0
Device Protection Plan            0
Premium Tech Support              0
Streaming TV                      0
Streaming Movies                  0
Streaming Music                   0
Unlimited Data                11103
Payment Method                    0
Status ID                         0
Satisfaction Score                0
Churn Category                    0
Churn Reason                      0
Customer Status                   0
offer                             0
dtype: int64
```

Data Dictionary

In order to conduct EDA effectively, I require a comprehensive understanding of the variables present in our dataset. A data dictionary serves as a valuable resource in achieving this objective.

Column name	Description
Customer ID	Unique identifier for each customer
Month	Calendar Month- 1:12
Month of Joining	Calender Month -1:14, Month for which the data is captured
zip_code	Zip Code
Gender	Gender
Age	Age(Years)
Married	Marital Status
Dependents	Dependents - Binary
Number of Dependents	Number of Dependents
Location ID	Location ID
Service ID	Service ID
state	State
county	County
timezone	Timezone
area_codes	Area Code
country	Country
latitude	Latitude
longitude	Longitude
arpu	Average revenue per user

Column name	Description
roam_ic	Roaming incoming calls in minutes
roam_og	Roaming outgoing calls in minutes
loc_og_t2t	Local outgoing calls within same network in minutes
loc_og_t2m	Local outgoing calls outside network in minutes(outside same + partner network)
loc_og_t2f	Local outgoing calls with Partner network in minutes
loc_og_t2c	Local outgoing calls with Call Center in minutes
std_og_t2t	STD outgoing calls within same network in minutes
std_og_t2m	STD outgoing calls outside network in minutes(outside same + partner network)
std_og_t2f	STD outgoing calls with Partner network in minutes
std_og_t2c	STD outgoing calls with Call Center in minutes
isd_og	ISD Outgoing calls
spl_og	Special Outgoing calls
og_others	Other Outgoing Calls
loc_ic_t2t	Local incoming calls within same network in minutes
loc_ic_t2m	Local incoming calls outside network in minutes(outside same + partner network)
loc_ic_t2f	Local incoming calls with Partner network in minutes
std_ic_t2t	STD incoming calls within same network in minutes
std_ic_t2m	STD incoming calls outside network in minutes(outside same + partner network)
std_ic_t2f	STD incoming calls with Partner network in minutes
std_ic_t2o	STD incoming calls operators other networks in minutes
spl_ic	Special Incoming calls in minutes
isd_ic	ISD Incoming calls in minutes
ic_others	Other Incoming Calls
total_rech_amt	Total Recharge Amount in Local Currency
total_rech_data	Total Recharge Amount for Data in Local Currency
vol_4g	4G Internet Used in GB
vol_5g	5G Internet used in GB
arpu_5g	Average revenue per user over 5G network
arpu_4g	Average revenue per user over 4G network
night_pck_user	Is Night Pack User(Specific Scheme)

Column name	Description
fb_user	Social Networking scheme
aug_vbc_5g	Volume Based cost for 5G network (outside the scheme paid based on extra usage)
offer	Offer Given to User
Referred a Friend	Referred a Friend : Binary
Number of Referrals	Number of Referrals
Phone Service	Phone Service: Binary
Multiple Lines	Multiple Lines for phone service: Binary
Internet Service	Internet Service: Binary
Internet Type	Internet Type
Streaming Data Consumption	Streaming Data Consumption
Online Security	Online Security
Online Backup	Online Backup
Device Protection Plan	Device Protection Plan
Premium Tech Support	Premium Tech Support
Streaming TV	Streaming TV
Streaming Movies	Streaming Movies
Streaming Music	Streaming Music
Unlimited Data	Unlimited Data
Payment Method	Payment Method
Status ID	Status ID
Satisfaction Score	Satisfaction Score
Churn Category	Churn Category
Churn Reason	Churn Reason
Customer Status	Customer Status
Churn Value	Binary Churn Value

Exploring `total_rech_data` column.

Exploring the nulls in `total_rech_data` and `Internet Type` columns

In [11]:
```
# Null values in total recharge data
df['total_rech_data'].isna().sum()
```

Out[11]: 209904

In [12]:
```
# Calculating the percentage of missing values in the 'total_rech_data' column
print(
    "Percentage of missing values in the total_rech_data: ",
    round(
        (df['total_rech_data'].isna().sum() / df.shape[0]) * 100,
        2),
    "%"
    )
```

```
Percentage of missing values in the total_rech_data:  32.11 %
```

In [13]:
```
# Counting the number of null values in the 'Internet Type' column
df['Internet Type'].isna().sum()
```

Out[13]: 325250

In [14]:
```
# Calculating the percentage of missing values in the 'Internet Type' column
print(
    "Percentage of missing values in the total_rech_data: ",
    round(
        (df['Internet Type'].isna().sum() / df.shape[0]) * 100,
        2),
    "%"
    )
```

```
Percentage of missing values in the total_rech_data:  49.75 %
```

My observation:

- The total_rech_data column has a significant proportion of missing values, accounting for approximately 49.75% of the dataset.
- The absence of values in the total_rech_data column may imply instances where customers either haven't recharged their accounts or have done so, but the data wasn't accurately recorded.
- One plausible scenario is that customers with missing recharge data could be those who received complimentary data services and thus didn't require a recharge. Conversely, these missing values might also stem from technical issues such as data recording errors or system malfunctions.

Exploring the nulls in `total_rech_data` and `Internet Service` columns

In [15]:
```
# Analyzing Internet Service distribution for customers with missing total recha
df[df['total_rech_data'].isna()]['Internet Service'].value_counts(dropna=False)
```

Out[15]:
```
Internet Service
Yes    209904
Name: count, dtype: int64
```

My observation:

- Upon deeper analysis, it is evident that all customers with missing recharge data have subscribed to internet service. This observation prompts the need to investigate whether these customers have actively utilized the internet service or not.

Exploring the nulls in `total_rech_data` and `Unlimited Data` columns

In [16]:
```
# Check the distribution of values in the Unlimited Data column for rows where
# total_rech_data is missing
df[df['total_rech_data'].isna()]['Unlimited Data'].value_counts(normalize=True)
```

Out[16]:
```
Unlimited Data
Yes    0.86249
No     0.13751
Name: proportion, dtype: float64
```

Observation:

Among customers with missing recharge data, a significant majority (86.2%) have opted for unlimited data plans, while 13.7% have chosen plans without unlimited data. This highlights a potential association between the absence of recharge data and the preference for unlimited data plans. Further investigation is recommended to understand this correlation better.

Exploring the nulls in `total_rech_data`, `arpu_4g` and `arpu_5g` columns

In [17]:
```
# Checking the average revenue for 4G and 5G users with missing recharge data.
df[(df['total_rech_data'].isna())][['arpu_4g','arpu_5g']].value_counts()
```

Out[17]:
```
arpu_4g         arpu_5g
Not Applicable  Not Applicable       195182
297.57          8530.983628675347         4
544.17          8536.565905559497         3
395.94          8533.210427447017         3
290.09          8530.814304137255         3
                                        ...
222.42          1468.94                   1
222.56          8529.28562953066          1
222.67          8529.288119597395         1
222.73          8529.289477815615         1
2559.56         8582.18822919918          1
Name: count, Length: 14247, dtype: int64
```

My observation:

- In the data preprocessing step, I suggest filling in the missing values in the total_rech_data column with 0, particularly when the Average Revenue Per User (ARPU) is not applicable.

- This choice is informed by the fact that ARPU reflects the revenue generated per user, and when it's not applicable, it likely indicates that the user isn't contributing

any revenue to the company. Therefore, assuming a total recharge amount of 0 in such cases seems like a reasonable approach.

In [18]:
```
# Checking the value counts of ARPU 4g and 5g
df[['arpu_4g','arpu_5g']].value_counts()
```

Out[18]:
```
arpu_4g         arpu_5g
Not Applicable  Not Applicable    195182
0.0             0.0              184117
                63.0              13024
63.0            0.0               12969
254687.0        0.0               10911
                                  ...
192.88          566.93                1
192.89          274.15                1
                2848.71               1
                648.54                1
2416.82         167.18                1
Name: count, Length: 195845, dtype: int64
```

In [19]:
```
# Replace total recharge data with 0 where ARPU for 4G and 5G are not applicable
df.loc[(df['arpu_4g'] == 'Not Applicable') | (df['arpu_5g'] == 'Not Applicable')
```

In [20]:
```
# Calculate the percentage of missing values in the total_rech_data column
missing_percentage = (df['total_rech_data'].isna().sum() / df.shape[0]) * 100

# Printing the missing values
print(
    "Percentage of missing values in the total_rech_data after exploration: ",
    round(missing_percentage, 2), "%"
    )
```

```
Percentage of missing values in the total_rech_data after exploration:  2.25 %
```

Conclusion:

After exploring the data and considering the distribution of ARPU values for 4G and 5G users, it was observed that a significant number of instances have ARPU labeled as "Not Applicable," suggesting zero revenue contribution. Therefore, it was deemed appropriate to replace missing values in the total_rech_data column with 0 in such cases.

Following this replacement, the percentage of missing values in the total_rech_data column decreased to 2.25% from 32.11%. This approach ensures data integrity and allows for a more comprehensive analysis, accounting for the potential revenue impact of users with missing recharge data.

Considering ARPU

It's important to note that I cannot simply fill other values with 0, especially considering the ARPU (Average Revenue Per User) factor.

In [21]:
```
# Calculate the mean of 'total_rech_data' where either 'arpu_4g' or 'arpu_5g' is
# not equal to 'Not Applicable'
df.loc[
    (df['arpu_4g'] != 'Not Applicable') | (df['arpu_5g'] != 'Not Applicable'),
```

```
    'total_rech_data'
    ].mean()
```

Out[21]: 4.85274721808543

I will use this mean to fill the NaN values.

In [22]:
```
# Fill NaN values in 'total_rech_data' with the mean of 'total_rech_data' where
# either 'arpu_4g' or 'arpu_5g' is not equal to 'Not Applicable',
# using the mean calculated from valid data.
df['total_rech_data'] = df[
    'total_rech_data'
    ].fillna(
        df.loc[
            (df['arpu_4g'] != 'Not Applicable') | (df['arpu_5g'] != 'Not Applica
            'total_rech_data'
            ].mean()
            )
```

In [23]:
```
# Check the value counts for Internet Type
df['Internet Type'].value_counts(dropna=False)
```

Out[23]:
```
Internet Type
NaN            325250
Fiber Optic    134991
Cable          112100
DSL             81412
Name: count, dtype: int64
```

In [24]:
```
# Check value counts for Internet Service where Internet Type is null
df[df['Internet Type'].isna()]['Internet Service'].value_counts(dropna=False)
```

Out[24]:
```
Internet Service
No     236152
Yes     89098
Name: count, dtype: int64
```

Since there is no Internet Service for all null values in Internet Type, I will fill these null values with "Not Applicable".

In [25]:
```
# Fill null values in the 'Internet Type' column with 'Not Applicable'
df['Internet Type'] = df['Internet Type'].fillna('Not Applicable')
```

In [26]:
```
# Insert a new column named 'total_recharge' before the last column in the dataf
# The values of 'total_recharge' are calculated as the sum of 'total_rech_amt' a
df.insert(loc=df.shape[1] - 1, column='total_recharge', value=df['total_rech_amt
```

In [27]:
```
# Calculate the percentage of missing values in columns
df_missing_columns = (round(((df.isnull().sum() / len(df.index)) * 100), 2)
                      .to_frame('null')
                      .sort_values('null', ascending=False))
df_missing_columns
```

Out[27]:

	null
fb_user	62.78
night_pck_user	57.07
Multiple Lines	7.05
Unlimited Data	1.70
Number of Dependents	0.76
Number of Referrals	0.06
total_rech_amt	0.00
total_rech_data	0.00
vol_4g	0.00
vol_5g	0.00
arpu_5g	0.00
Internet Service	0.00
arpu_4g	0.00
ic_others	0.00
isd_ic	0.00
aug_vbc_5g	0.00
Churn Value	0.00
Referred a Friend	0.00
Phone Service	0.00
Customer ID	0.00
Internet Type	0.00
std_ic_t2o	0.00
Payment Method	0.00
total_recharge	0.00
Customer Status	0.00
Churn Reason	0.00
Churn Category	0.00
Satisfaction Score	0.00
Status ID	0.00
Streaming Music	0.00
Streaming Data Consumption	0.00
Streaming Movies	0.00
Streaming TV	0.00

	null
Premium Tech Support	0.00
Device Protection Plan	0.00
Online Backup	0.00
Online Security	0.00
spl_ic	0.00
std_ic_t2f	0.00
Month	0.00
std_ic_t2m	0.00
longitude	0.00
latitude	0.00
country	0.00
area_codes	0.00
timezone	0.00
county	0.00
state	0.00
Service ID	0.00
Location ID	0.00
Dependents	0.00
Married	0.00
Age	0.00
Gender	0.00
zip_code	0.00
Month of Joining	0.00
arpu	0.00
roam_ic	0.00
roam_og	0.00
isd_og	0.00
std_ic_t2t	0.00
loc_ic_t2f	0.00
loc_ic_t2m	0.00
loc_ic_t2t	0.00
og_others	0.00
spl_og	0.00

	null
std_og_t2c	0.00
loc_og_t2t	0.00
std_og_t2f	0.00
std_og_t2m	0.00
std_og_t2t	0.00
loc_og_t2c	0.00
loc_og_t2f	0.00
loc_og_t2m	0.00
offer	0.00

Dropping unnecessary columns!

In [28]:
```
# Remove columns from the DataFrame
df = df.drop(columns=['night_pck_user', 'fb_user', 'Customer Status'])
```

I will replace 'Not Applicable' with 0 in both the 'arpu_4g' and 'arpu_5g' columns.

In [29]:
```
# Replace 'Not Applicable' with 0 in the 'arpu_4g' column
df['arpu_4g'] = df['arpu_4g'].replace('Not Applicable', 0)

# Replace 'Not Applicable' with 0 in the 'arpu_5g' column
df['arpu_5g'] = df['arpu_5g'].replace('Not Applicable', 0)

# Convert the 'arpu_4g' column to float data type
df['arpu_4g'] = df['arpu_4g'].astype(float)

# Convert the 'arpu_5g' column to float data type
df['arpu_5g'] = df['arpu_5g'].astype(float)
```

Detecting Outliers

Outlier detection involves identifying and removing data points significantly different from the rest, crucial for accurate analysis. Parametric methods rely on data distribution assumptions, while non-parametric methods assess data ranks without distribution assumptions, both essential for various data scenarios.

More information about this can be found in the resource directory.

In [30]:
```
# Display the data types of each column in the dataframe
df.dtypes
```

Out[30]:

Customer ID	object
Month	int64
Month of Joining	float64
zip_code	int64
Gender	object
Age	float64
Married	object
Dependents	object
Number of Dependents	float64
Location ID	object
Service ID	object
state	object
county	object
timezone	object
area_codes	object
country	object
latitude	float64
longitude	float64
arpu	float64
roam_ic	float64
roam_og	float64
loc_og_t2t	float64
loc_og_t2m	float64
loc_og_t2f	float64
loc_og_t2c	float64
std_og_t2t	float64
std_og_t2m	float64
std_og_t2f	float64
std_og_t2c	float64
isd_og	float64
spl_og	float64
og_others	float64
loc_ic_t2t	float64
loc_ic_t2m	float64
loc_ic_t2f	float64
std_ic_t2t	float64
std_ic_t2m	float64
std_ic_t2f	float64
std_ic_t2o	float64
spl_ic	float64
isd_ic	float64
ic_others	float64
total_rech_amt	float64
total_rech_data	float64
vol_4g	float64
vol_5g	float64
arpu_5g	float64
arpu_4g	float64
aug_vbc_5g	float64
Churn Value	int64
Referred a Friend	object
Number of Referrals	float64
Phone Service	object
Multiple Lines	object
Internet Service	object
Internet Type	object
Streaming Data Consumption	int64
Online Security	object
Online Backup	object
Device Protection Plan	object

```
Premium Tech Support           object
Streaming TV                   object
Streaming Movies               object
Streaming Music                object
Unlimited Data                 object
Payment Method                 object
Status ID                      object
Satisfaction Score              int64
Churn Category                 object
Churn Reason                   object
total_recharge                float64
offer                          object
dtype: object
```

In [31]:
```
# Define a list of continuous columns in the dataframe
cts_cols = [
    'Age', 'Number of Dependents', 'roam_ic', 'roam_og', 'loc_og_t2t',
    'loc_og_t2m', 'loc_og_t2f', 'loc_og_t2c', 'std_og_t2t', 'std_og_t2m',
    'std_og_t2f', 'std_og_t2c', 'isd_og', 'spl_og', 'og_others',
    'loc_ic_t2t', 'loc_ic_t2m', 'loc_ic_t2f', 'std_ic_t2t', 'std_ic_t2m',
    'std_ic_t2f', 'std_ic_t2o', 'spl_ic', 'isd_ic', 'ic_others',
    'total_rech_amt', 'total_rech_data', 'vol_4g', 'vol_5g', 'arpu_5g',
    'arpu_4g', 'arpu', 'aug_vbc_5g', 'Number of Referrals', 'Satisfaction Score'
    'Streaming Data Consumption'
    ]
```

In [32]:
```
# Create an empty dataframe with columns from cts_cols and quantiles as index
quantile_df = pd.DataFrame(columns=cts_cols, index=[0.1, 0.25, 0.5, 0.75, 0.8, 0

# Calculate quantiles for each column in cts_cols and store them in quantile_df
for col in cts_cols:
    quantile_df[col] = df[col].quantile([0.1, 0.25, 0.5, 0.75, 0.8, 0.9, 0.95, 0
```

Observations:

1. Calculating quantiles for each continuous variable allows us to understand the spread and distribution of the data, primarily focusing on identifying potential outliers.

2. Quantiles, such as the 0.25 and 0.75 quantiles, divide the distribution into equal parts, offering insights into where certain percentages of observations fall within the data.

3. By utilizing these quantiles as thresholds, we can effectively flag potential outliers in the dataset, prompting further investigation to determine their significance and validity.

In [33]:
```
# Display the quantiles dataframe
quantile_df
```

Out[33]:

	Age	Number of Dependents	roam_ic	roam_og	loc_og_t2t	loc_og_t2m	loc_og_t2f	lc
0.10	24.0	0.0	0.000000	0.000	0.000000	0.0000	0.000000	
0.25	28.0	0.0	12.090000	14.710	32.700000	26.2600	1.460000	
0.50	34.0	0.0	50.560000	75.100	171.330000	135.4600	7.800000	
0.75	43.0	1.0	162.030000	135.290	309.090000	618.2300	14.090000	
0.80	47.0	2.0	496.935485	146.820	856.841431	1392.8500	43.876522	
0.90	55.0	4.0	969.048000	689.608	3614.015883	2644.5980	126.595429	
0.95	61.0	7.0	1283.198000	1954.392	5079.830000	3479.4380	183.494000	
0.97	64.0	8.0	1494.043200	2550.390	5806.054400	3756.4444	206.744400	
0.99	74.0	9.0	1646.899600	3041.760	6191.204000	4060.2988	257.650000	

Observation:

Outliers were identified in the features `vol_5g`, `arpu_4g`, and `arpu_5g`.

In [34]:
```
# Calculate specific quantiles for the 'arpu_4g' column
df['arpu_4g'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[34]:
```
0.750       228.220000
0.800       783.290000
0.900      2224.100000
0.950      8675.302558
0.970      8839.721689
0.990    254687.000000
0.999    254687.000000
Name: arpu_4g, dtype: float64
```

In [35]:
```
# Calculate the proportion of rows where 'arpu_4g' equals 254687
df[df['arpu_4g'] == 254687].shape[0] / df.shape[0]
```

Out[35]:
```
0.019651152652454366
```

In [36]:
```
# Display the rows where 'arpu_4g' equals 254687
df[df['arpu_4g'] == 254687]
```

Out[36]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Marrie
9	uqdtniwvxqzeu1	14	6.0	72566	Male	36.463423	N
86	ucpurmfkdlnwi18	13	12.0	71747	Female	20.000000	Ye
103	sirifvlkipkel21	13	11.0	92865	Female	40.000000	Ye
112	dnnrchjlmrylq24	14	9.0	91423	Female	48.000000	Ye
145	pltaycxycbhvo31	11	7.0	95126	Other	35.000000	N
...	...	...	...	...	...	...	.
653317	tphemcbndfpem162885	5	5.0	91604	Female	23.000000	Ye
653369	umbrcxomoexlc162896	8	5.0	94939	Female	55.000000	Ye
653423	dkjfuyorfdngv162907	13	11.0	87553	Male	23.000000	Ye
653536	jqvmittclvgqd162934	11	7.0	98907	Not Specified	40.000000	Ye
653580	lvinoatdykyvc162940	7	6.0	98132	Male	27.000000	No Specifie

12847 rows × 72 columns

I'll check the value of `total_rech_data` for these observations.

In [37]:
```
# Obtain the value counts of 'total_rech_data' for observations where 'arpu_4g'
df[df['arpu_4g'] == 254687]['total_rech_data'].value_counts()
```

Out[37]:
```
total_rech_data
0.0    12847
Name: count, dtype: int64
```

Now that the `total recharge data` amount is 0 and there is no ARPU, I'll replace it with 0.

In [38]:
```
# Replace the outlier value 254687 in 'arpu_4g' column with 0
df['arpu_4g'] = df['arpu_4g'].replace(254687, 0)
```

In [39]:
```
# Further examination of 'arpu_4g' column's quantiles
df['arpu_4g'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[39]:
```
0.750      120.570000
0.800      504.112000
0.900     1893.758000
0.950     2493.880000
0.970     8675.470757
0.990     8839.721689
0.999    87978.000000
Name: arpu_4g, dtype: float64
```

In [40]:
```
# Filtering for 'arpu_4g' value of 87978 and computing unique value counts in 't
df[df['arpu_4g'] == 87978]['total_rech_data'].value_counts()
```

Out[40]:
```
total_rech_data
0.0    5007
Name: count, dtype: int64
```

Observations:

In all rows where the 'arpu_4g' value is 87978, the 'total_rech_data' column shows a value of 0. This suggests potential outliers. Consequently, I've chosen to replace the 'arpu_4g' value with 0 for these specific rows.

In [41]:
```
# Replace the 'arpu_4g' values of 87978 with 0
df['arpu_4g'] = df['arpu_4g'].replace(87978, 0)
```

In [42]:
```
# Re-evaluating the quantiles
df['arpu_4g'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[42]:
```
0.750     107.760000
0.800     432.246000
0.900    1803.560000
0.950    2424.072000
0.970    2735.554400
0.990    8705.097343
0.999    8839.721689
Name: arpu_4g, dtype: float64
```

In [43]:
```
# Evaluate churn status associated with ARPU exceeding 8000
df[df['arpu_4g'] > 8000]['Churn Value'].value_counts()
```

Out[43]:
```
Churn Value
0    16157
1      980
Name: count, dtype: int64
```

Observation:

1. A higher Average Revenue Per User (ARPU) often indicates increased revenue generation per user, potentially contributing to improved profitability. Nonetheless, elevated ARPU values may also hint at higher churn rates within the business.

2. Elevated ARPU could correlate with increased churn for several reasons. Firstly, when a business charges premium prices for its services, it may attract a customer base more sensitive to pricing changes, thus more prone to switching to competitors

offering better deals. Consequently, this dynamic can lead to a heightened churn rate for the business.

In [44]:
```
# Count the occurrences of 'total_rech_data' values for observations with an 'ar
df[df['arpu_5g'] == 254687]['total_rech_data'].value_counts()
```

Out[44]:
```
total_rech_data
0.0    12614
Name: count, dtype: int64
```

In [45]:
```
# Count the occurrences of 'total_rech_data' values for observations with an 'ar
df[df['arpu_5g'] == 87978]['total_rech_data'].value_counts()
```

Out[45]:
```
total_rech_data
0.0    5130
Name: count, dtype: int64
```

Now that the majority of 'total recharge data' values for ARPU_5G around the quantiles are 0, I'll replace them with 0.

In [46]:
```
# Replace values of 87978 and 254687 with 0 in the 'arpu_5g' column where total
df['arpu_5g'] = df['arpu_5g'].replace([87978, 254687], 0)
```

In [47]:
```
# Check the quantiles of ARPU 5G
df['arpu_5g'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[47]:
```
0.750      96.490000
0.800     417.102000
0.900    1797.618000
0.950    2543.904000
0.970    2792.060000
0.990    8587.153966
0.999    8724.440600
Name: arpu_5g, dtype: float64
```

Assumptions for CLTV:

To simplify the calculation process for customer lifetime value (CLTV), I'm making several assumptions:

- I assume that customers are not charged for incoming calls or messages. This assumption is based on the difficulty of tracking incoming usage and its limited revenue impact for telecom companies.
- Additionally, I assume that customers are not billed for calls made to the call center. This assumption aims to ensure that customers feel encouraged to seek assistance when needed without financial barriers.

In [48]:
```
# Considering incoming usage as free,
# I'll focus on outgoing calls, data usage, and premium services columns, assumi

# Creating a new column for total outgoing call usage
df['outgoing_call_usage'] = df['roam_og'] + df['loc_og_t2t'] + df['loc_og_t2m']
```

```
# Creating a new column for total data usage
df['data_usage'] = df['vol_4g'] + df['vol_5g'] + df['Streaming Data Consumption'
```

In [49]:
```
# Calculating the total ARPU for each user in each month
df['total_arpu'] = df[['arpu', 'arpu_4g', 'arpu_5g']].sum(axis=1)
```

In [50]:
```
# Removing values after the 0.90 quantile and replacing them with group medians
df['outgoing_call_usage'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[50]:
```
0.750     4436.480000
0.800     5594.251281
0.900     8960.748141
0.950    12312.060683
0.970    14537.184235
0.990    17997.599267
0.999    22152.084652
Name: outgoing_call_usage, dtype: float64
```

In [51]:
```
# Removing values after the 0.90 quantile and replacing them with group medians
df['data_usage'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[51]:
```
0.750     1125.82000
0.800     2953.23600
0.900    10039.23000
0.950    14873.23200
0.970    17099.29520
0.990    19111.58200
0.999    21748.18856
Name: data_usage, dtype: float64
```

In [52]:
```
# Removing values after the 0.90 quantile and replacing them with group medians
df['total_arpu'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[52]:
```
0.750     2181.600000
0.800     3066.298000
0.900     5783.486000
0.950     9176.836704
0.970    10687.852517
0.990    17325.527488
0.999    19254.865964
Name: total_arpu, dtype: float64
```

In [53]:
```
# Defining the columns to check for outliers
cols_to_check = ['outgoing_call_usage', 'data_usage', 'total_arpu']

# Calculating the upper bound for outlier detection
q = 0.90
upper_bound = df[cols_to_check].quantile(q)

# Replacing values above the upper bound with NaN
df[cols_to_check] = df[cols_to_check].mask(df[cols_to_check] > upper_bound, np.n
```

In [54]:
```
# Verifying the quantiles for 'data_usage'
df['data_usage'].quantile([0.75, 0.8, 0.9, 0.95, 0.97, 0.99, 0.999])
```

Out[54]:
```
0.750     908.27000
0.800    1026.61000
0.900    3552.40800
0.950    6428.95800
0.970    8309.97120
0.990    9530.69160
0.999    9984.09232
Name: data_usage, dtype: float64
```

In [55]:
```
# Filling missing values with median
df['data_usage'] = df['data_usage'].fillna(df['data_usage'].median())
df['outgoing_call_usage'] = df['outgoing_call_usage'].fillna(df['outgoing_call_u
df['total_arpu'] = df['total_arpu'].fillna(df['total_arpu'].median())
```

In [56]:
```
# Verify missing values
df[cols_to_check].isnull().sum()
```

Out[56]:
```
outgoing_call_usage    0
data_usage             0
total_arpu             0
dtype: int64
```

In [57]:
```
# Example of a churned customer
df[df['Customer ID'] == "tqhiqgvbbhley51"]
```

Out[57]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents
239	tqhiqgvbbhley51	1	1.0	87654	Female	50.0	Not Specified	Yes
240	tqhiqgvbbhley51	2	1.0	87654	Female	50.0	Not Specified	Yes
241	tqhiqgvbbhley51	3	1.0	87654	Female	50.0	Not Specified	Yes
242	tqhiqgvbbhley51	4	1.0	87654	Female	50.0	Not Specified	Yes

In [58]:
```
# Creating a new column called Tenure
df['Tenure'] = df['Month'] - df['Month of Joining'] + 1
```

In [59]:
```
# Displaying the first 3 rows of the DataFrame
df.head(3)
```

Out[59]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents
0	hthjctifkiudi0	1	1.0	71638	Female	36.000000	No	No
1	uqdtniwvxqzeu1	6	6.0	72566	Male	36.657198	No	No
2	uqdtniwvxqzeu1	7	6.0	72566	Male	36.607828	No	No

Saving the Data

In [60]:
```
# Save the DataFrame to a CSV file

try:
    # Try to save the DataFrame to a CSV file in the local directory
    df.to_csv('../data/processed/telecom_data.csv_2', index=False)
    print("DataFrame saved locally successfully!")

except Exception as e:
    # If there's an error accessing the local directory, attempt to save it to a
    print("Error occurred while accessing the local directory:", e)

    try:
        # Define an alternative path on Google Drive
        alternative_path = "/content/drive/Othercomputers/MyLaptop/git_repos/pro
        # Try saving the DataFrame to the alternative location
        df.to_csv(alternative_path, index=False)
        print("DataFrame saved to an alternative location on Google Drive:", alt

    except Exception as e:
        # If both attempts fail, print an error message
        print("Unable to save the DataFrame:", e)
```

```
Error occurred while accessing the local directory: Cannot save file into a non-e
xistent directory: '../data/processed'
DataFrame saved to an alternative location on Google Drive: /content/drive/Otherc
omputers/MyLaptop/git_repos/projectpro/Projects/MachineLearning/Telecommunication
_project/data/processed/telecom_data_2.csv
```

Exploring ARPU and CLTV

ARPU Analysis

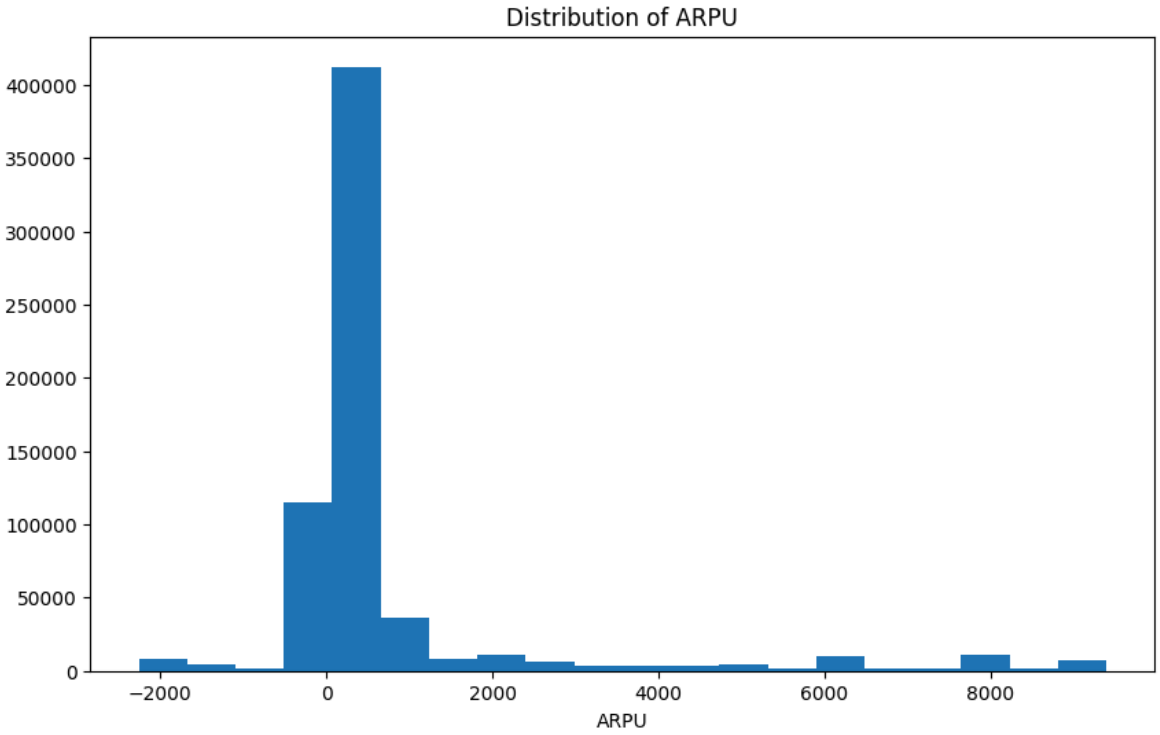
I'm revisiting ARPU analysis because it's crucial for CLTV. It helps me pinpoint the most profitable customer segments, allowing me to forecast future revenue. CLTV considers both customer revenue and acquisition/retention costs. By understanding which segments have higher ARPU, I can allocate resources more effectively, boosting CLTV.

Analyzing ARPU trends over time helps me spot customer spending patterns, informing marketing and retention strategies to maximize CLTV.

Regarding ARPU hypotheses:

- **Churned Customers ARPU:** I'll compare ARPU between churned and non-churned customers. Higher ARPU for churned customers suggests I need to improve retention strategies for high-value customers.
- **ARPU and Service Usage:** I'll examine the correlation between service usage and ARPU. A positive correlation indicates that customers using more services tend to spend more.
- **Impact of Security Services:** I'll compare ARPU between customers using online security/backup services and those who don't. Higher ARPU for service users suggests these services attract high-value customers.
- **Satisfaction and ARPU:** I'll analyze the correlation between customer satisfaction scores and ARPU. A positive correlation suggests satisfied customers spend more, emphasizing the importance of customer satisfaction in revenue growth.

In [61]:
```
# Plotting ARPU distribution
plt.figure(figsize=(10,6))
plt.hist(df['arpu'], bins=20)
plt.title('Distribution of ARPU')
plt.xlabel('ARPU')
plt.show()
```



Observation:

- I notice that the distribution of ARPU values is right-skewed, with most values concentrated below 500. This suggests that there are very high ARPU values for a

few customers, supporting our hypotheses.

- I also observe some negative ARPU values, which may be due to refunds. A negative ARPU indicates that the company is losing money on a per-user basis, rather than making a profit. It's not uncommon for telecom companies to have negative ARPU due to the high costs associated with acquiring and servicing customers.

In [62]:
```
# Investigating negative ARPU values
df['arpu'].describe()
```

Out[62]:
```
count    653753.000000
mean        781.196280
std        1807.269437
min       -2258.680000
25%         118.940000
50%         348.540000
75%         580.650000
max        9394.500000
Name: arpu, dtype: float64
```

In [63]:
```
# Investigating the number of people with negative ARPU
df[df['arpu'] < 0].shape
```

Out[63]:
```
(115942, 76)
```

In [64]:
```
# Calculating the percentage of customers with negative ARPU
percentage_negative_arpu = df[df['arpu'] < 0].shape[0] / df.shape[0]
percentage_negative_arpu
```

Out[64]:
```
0.17734832574382067
```

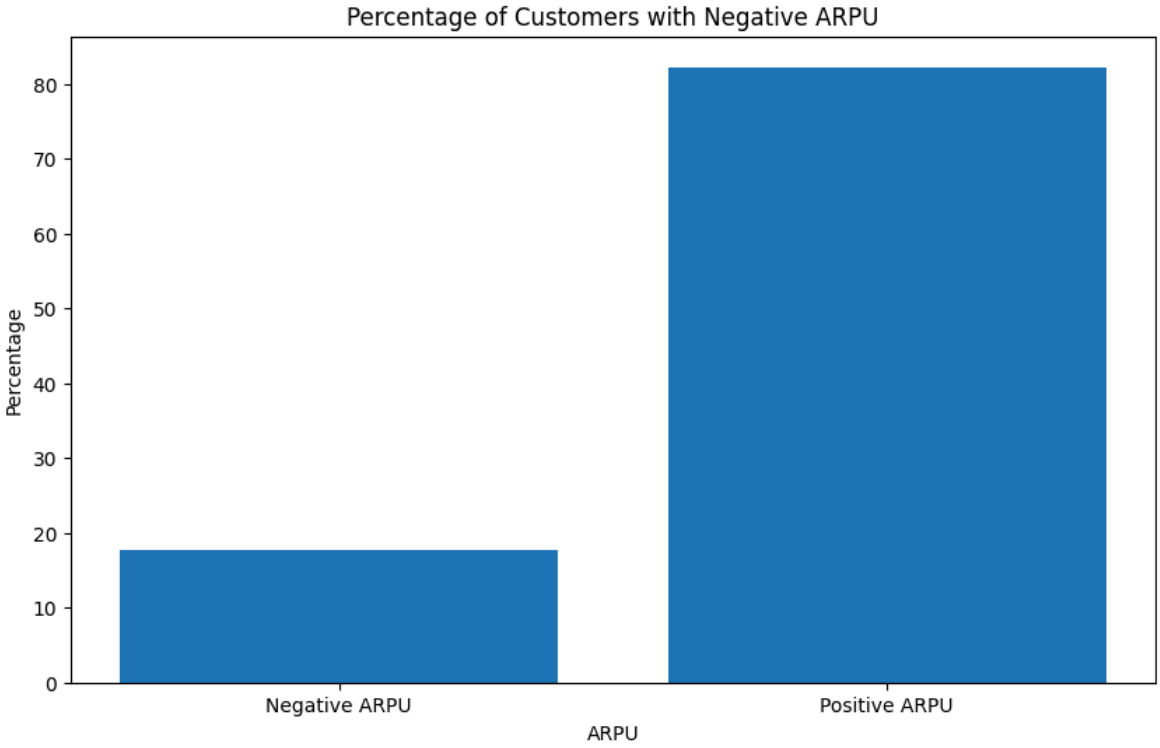
Summary from Univariate Analysis of Negative ARPU: Insights and Implications:

The univariate data analysis revealed that 17.7% of customers exhibit negative ARPU, suggesting a significant portion of the customer base isn't generating revenue for the telecom company.

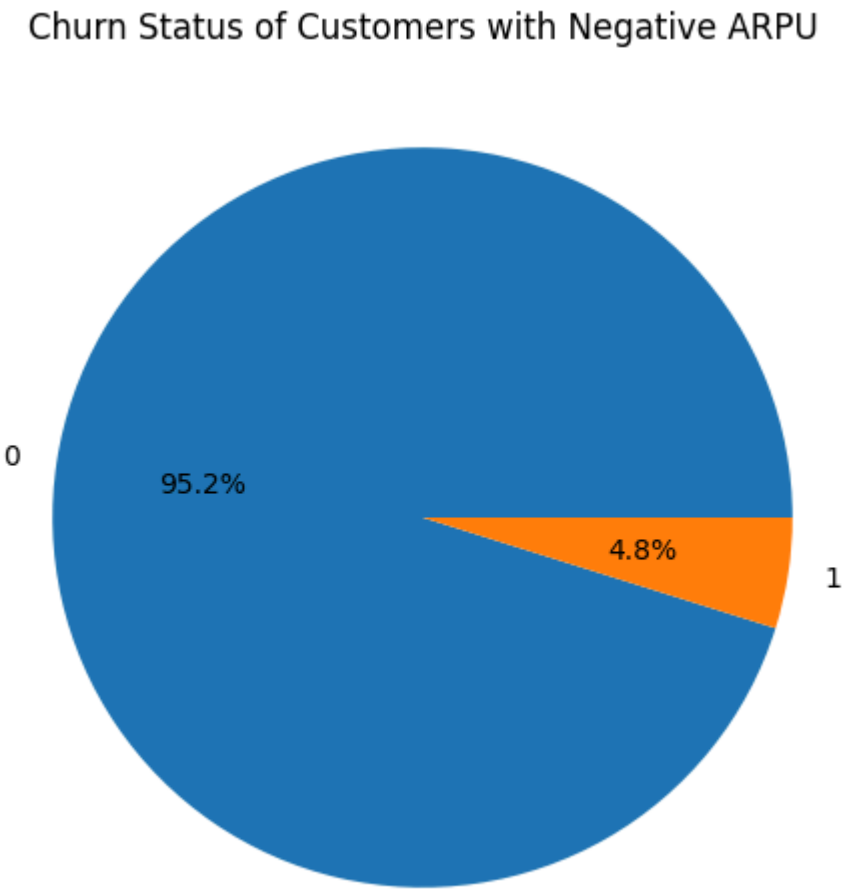
- Possible reasons for negative ARPU include inactive accounts, delinquencies, or outstanding balances.
- To gain deeper insights, further exploration can focus on related variables such as churn status. Additionally, examining the distribution of negative ARPU across demographics like age, gender, and location may uncover underlying patterns or trends.
- Moreover, investigating correlations between negative ARPU and usage patterns, service subscriptions, and customer satisfaction scores is essential for a comprehensive understanding of its implications on the telecom company's operations.

In [65]:
```
# Calculate the percentage of customers with negative ARPU
negative_arpu_percentage = (df[df['arpu'] < 0]['arpu'].count() / df.shape[0]) *

# Create a bar plot to visualize the negative ARPU percentage
```

```
plt.figure(figsize=(10,6))
plt.bar(['Negative ARPU', 'Positive ARPU'], [negative_arpu_percentage, 100 - neg
plt.title('Percentage of Customers with Negative ARPU')
plt.xlabel('ARPU')
plt.ylabel('Percentage')
plt.show()
```



In [66]:
```
# Calculate the relationship between negative ARPU and churn status
churn_negative_arpu = df[df['arpu'] < 0]['Churn Value'].value_counts(normalize=T

# Create a pie chart to visualize the churn status of customers with negative AR
plt.figure(figsize=(10,6))
plt.pie(churn_negative_arpu, labels=churn_negative_arpu.index, autopct='%1.1f%%'
plt.title('Churn Status of Customers with Negative ARPU')
plt.show()
```



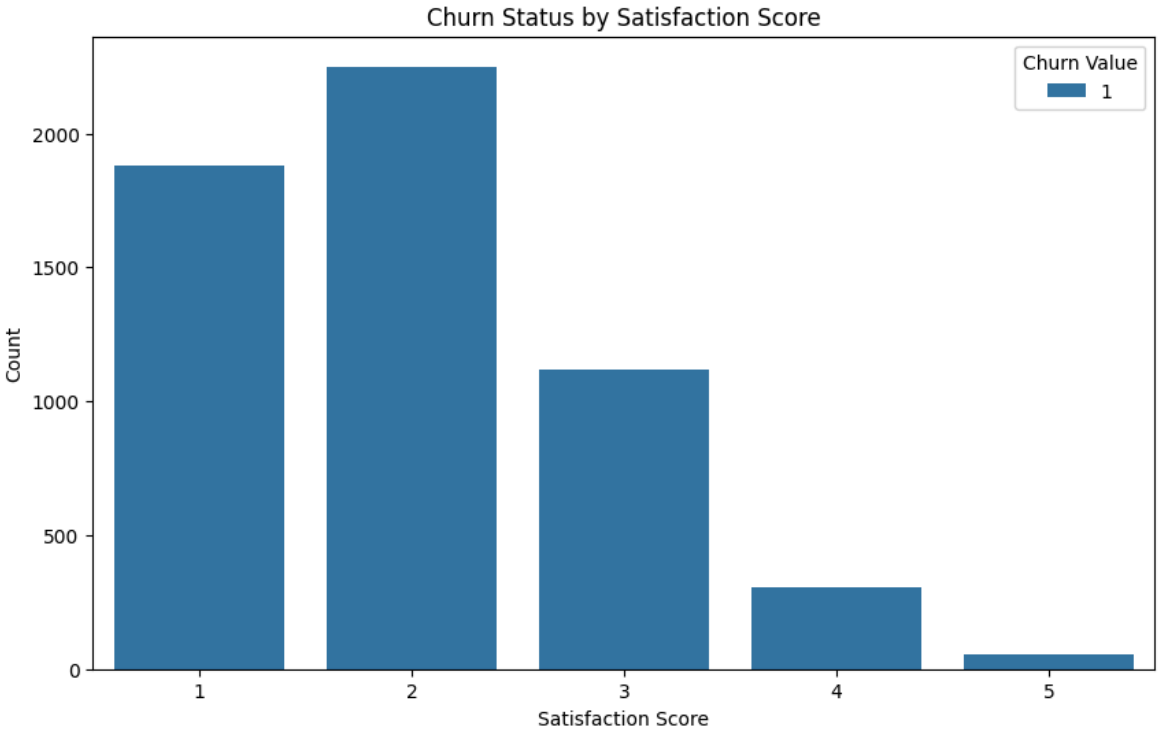
Observation:

- The bar plot reveals that 95.2% of customers with negative ARPU did not churn, while only 4.8% churned.
- This suggests that having negative ARPU is not necessarily a strong predictor of churn. Further analysis is necessary to understand the relationship between negative ARPU and churn, considering factors like usage patterns, service subscriptions, and customer satisfaction.
- Negative ARPU could stem from various factors such as inactive accounts, delinquencies, or outstanding balances. It may also include expenses incurred by the company to retain customers.
- Among the customers with negative ARPU who churned, it's possible they sought better deals elsewhere or experienced dissatisfaction with the services.

In [67]:
```
# Analyzing further
negative_arpu_churn = df[(df['arpu'] < 0) & (df['Churn Value'] == 1)]

# Print the number of customers with negative ARPU who churned
print("Number of customers with negative ARPU who churned:", len(negative_arpu_c
```

```
Number of customers with negative ARPU who churned: 5610
```

In [68]:
```
# Count plot of negative churned
plt.figure(figsize=(10, 6))
sns.countplot(x='Satisfaction Score', hue='Churn Value', data=negative_arpu_chur

# Add labels and title
plt.xlabel('Satisfaction Score')
plt.ylabel('Count')
plt.title('Churn Status by Satisfaction Score')
plt.show()
```



In [69]:
```
# Calculate the average customer satisfaction score for this group
avg_satisfaction = negative_arpu_churn['Satisfaction Score'].mean()
print("Average customer satisfaction score:", round(avg_satisfaction, 2))
```

```
Average customer satisfaction score: 2.0
```

Observation:

- The average customer satisfaction score for churned customers with negative ARPU is 2. This low satisfaction score indicates dissatisfaction among these customers with the services provided by the telecom company.
- The findings suggest a need for the company to enhance its services and support to improve customer satisfaction levels and potentially retain more customers.

Data Preprocessing and Feature Engineering for Service Analysis

In [70]:
```
# List of variables to loop over for analysis
services = ['Internet Service', 'Online Security', 'Online Backup', 'Device Prot
            'Streaming TV', 'Streaming Movies', 'Streaming Music', 'Unlimited Da
```

In [71]:
```
# Encode categorical data
mapping_dict = {'Yes': 1, 'No': 0}
for column in services:
    # replace 'Yes' and 'No' with 1 and 0
    df[column] = df[column].replace(mapping_dict)
```

In [72]:
```
# Checking data types
df[services].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 653753 entries, 0 to 653752
Data columns (total 9 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Internet Service       653753 non-null  int64
 1   Online Security        653753 non-null  int64
 2   Online Backup          653753 non-null  int64
 3   Device Protection Plan 653753 non-null  int64
 4   Premium Tech Support   653753 non-null  int64
 5   Streaming TV           653753 non-null  int64
 6   Streaming Movies       653753 non-null  int64
 7   Streaming Music        653753 non-null  int64
 8   Unlimited Data         642650 non-null  float64
dtypes: float64(1), int64(8)
memory usage: 44.9 MB
```

Unlimited data should be of integer data type.

In [73]:
```
# Let's inspect the frequency of values in the "Unlimited Data" column.
df['Unlimited Data'].value_counts(dropna=False)
```

Out[73]:
```
Unlimited Data
1.0    352341
0.0    290309
NaN     11103
Name: count, dtype: int64
```

In [74]:
```
# Convert the column to numeric type, handling errors to coerce non-finite value
df['Unlimited Data'] = pd.to_numeric(df['Unlimited Data'], errors='coerce')
```

In [75]:
```
# Lets check the value counts
df['Unlimited Data'].fillna(0).value_counts(dropna=False)
```

Out[75]:
```
Unlimited Data
1.0    352341
0.0    301412
Name: count, dtype: int64
```

In [76]:
```
# Replace "None" with 0
df['Unlimited Data'] = df['Unlimited Data'].replace("None", 0)
```

In [77]:
```
# Calculate the number of services each person uses
df['num_services'] = df[services].sum(axis=1)
```

In [78]:
```
# Verify the first row of the DataFrame
df.head(1)
```

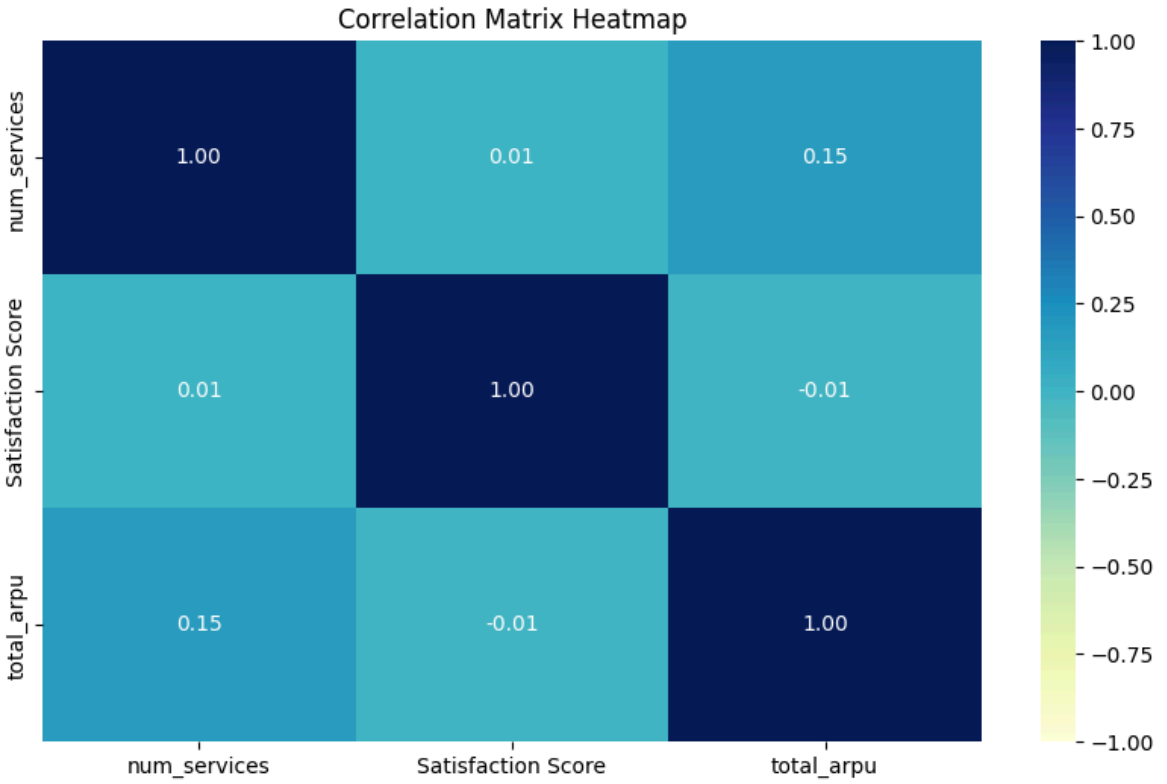
Out[78]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents	Numb Depen
0	hthjctifkiudi0	1	1.0	71638	Female	36.0	No	No	

Exploring Correlation

In measuring the strength of relationships between variables, I often rely on correlation. This metric reveals how one variable changes as another changes, indicating both the direction and magnitude of the relationship. Among the methods available, the Pearson correlation coefficient stands out, especially for assessing linear relationships between two continuous variables.

In [79]:
```
# Test correlation between number of services and satisfaction score to ARPU
correlation = df[['num_services', 'Satisfaction Score', 'total_arpu']].corr()

plt.figure(figsize=(10,6))
sns.heatmap(correlation, cmap="YlGnBu", annot=True, fmt=".2f", vmin=-1, vmax=1,
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Observations and Insights

The correlation coefficients reveal several insights into the relationships between satisfaction score, number of services used, and ARPU.

- **Satisfaction Score:** There is a weak negative correlation (-0.01) between satisfaction score and ARPU. This suggests that higher ARPU customers may not necessarily have higher satisfaction scores. Therefore, assuming that higher-paying customers are more satisfied with services might not always hold true.
- **Number of Services Used:** A weak positive correlation (0.15) exists between the number of services used and ARPU. This implies that customers using more services tend to have higher ARPU. Consequently, the company could potentially increase revenue by encouraging customers to utilize more services. Strategies such as offering bundled services or promotions might incentivize customers to use additional services.

In summary, while the correlation analysis supports the hypothesis that the number of services used is positively correlated with ARPU, it challenges the assumption that high ARPU customers are necessarily more satisfied. Therefore, it is essential for the company to prioritize improving customer satisfaction across all segments, irrespective of their ARPU levels.

Hypothesis Testing (T-Tests) for ARPU Variation Across Service Usage

A t-test is a statistical hypothesis test that compares the means of two groups. It is commonly used to determine if there is a significant difference between the means of two groups, such as a control group and a treatment group. The test is based on the t-distribution, which is a probability distribution that describes how the means of random samples from a population with a normal distribution are distributed.

There are two types of t-tests: the one-sample t-test and the two-sample t-test. The one-sample t-test is used to compare the mean of a sample to a known value, while the two-sample t-test is used to compare the means of two independent samples.

To perform a t-test, the first step is to formulate the null and alternative hypotheses. The null hypothesis is that there is no significant difference between the means of the two groups, while the alternative hypothesis is that there is a significant difference between the means of the two groups.

The next step is to calculate the test statistic, which is the t-value. This is done by taking the difference between the means of the two groups and dividing it by the standard error of the difference between the means. The standard error is a measure of the variability of the sample means and is calculated using the sample size and the standard deviation of each group.

Finally, the t-value is compared to a critical value from the t-distribution based on the degrees of freedom and the significance level chosen. If the calculated t-value is greater than the critical value, the null hypothesis is rejected, and it is concluded that there is a significant difference between the means of the two groups.

In [80]:
```
# Loop over the variables and perform t-tests
for var in services:
    group1 = df[df[var] == 1]['arpu']
    group0 = df[df[var] == 0]['arpu']
    t, p = ttest_ind(group1, group0, equal_var=False)
    if group1.mean() > group0.mean():
        print('Variable:', var)
        print('Group 1 Mean ARPU:', group1.mean())
        print('Group 0 Mean ARPU:', group0.mean())
        print('Higher weightage in Group 1')
        print('T-Statistic:', t)
        print('P-Value:', p, (p < 0.05))
        print('-----------------------')
    else:
        print('Variable:', var)
        print('Group 1 Mean ARPU:', group1.mean())
        print('Group 0 Mean ARPU:', group0.mean())
        print('Higher weightage in Group 0')
        print('T-Statistic:', t)
        print('P-Value:', p, (p < 0.05))
        print('-----------------------')
```

```
Variable: Internet Service
Group 1 Mean ARPU: 780.1507871374382
Group 0 Mean ARPU: 782.8323135470201
Higher weightage in Group 0
T-Statistic: -0.5849065656666134
P-Value: 0.5586107869078516 False
-----------------------
Variable: Online Security
Group 1 Mean ARPU: 784.0445228155231
Group 0 Mean ARPU: 780.7468038615277
Higher weightage in Group 1
T-Statistic: 0.506349585745111
P-Value: 0.612612197139793 False
-----------------------
Variable: Online Backup
Group 1 Mean ARPU: 783.0423254245743
Group 0 Mean ARPU: 780.2821661459787
Higher weightage in Group 1
T-Statistic: 0.5807817078583458
P-Value: 0.5613878886892029 False
-----------------------
Variable: Device Protection Plan
Group 1 Mean ARPU: 785.3021311329582
Group 0 Mean ARPU: 777.1122647582899
Higher weightage in Group 1
T-Statistic: 1.8319811981788083
P-Value: 0.06695467663042813 False
-----------------------
Variable: Premium Tech Support
Group 1 Mean ARPU: 782.08167396498
Group 0 Mean ARPU: 780.8904378213067
Higher weightage in Group 1
T-Statistic: 0.2324505781713282
P-Value: 0.816188232294468 False
-----------------------
Variable: Streaming TV
Group 1 Mean ARPU: 783.1097251914558
Group 0 Mean ARPU: 779.3390193129104
Higher weightage in Group 1
T-Statistic: 0.8433931302197835
P-Value: 0.39900892158643997 False
-----------------------
Variable: Streaming Movies
Group 1 Mean ARPU: 781.6996947832434
Group 0 Mean ARPU: 780.7160179633561
Higher weightage in Group 1
T-Statistic: 0.21997652624846156
P-Value: 0.8258895051277693 False
-----------------------
Variable: Streaming Music
Group 1 Mean ARPU: 777.1339648452011
Group 0 Mean ARPU: 784.5586718983704
Higher weightage in Group 0
T-Statistic: -1.6545240019026117
P-Value: 0.09802158673766194 False
-----------------------
Variable: Unlimited Data
Group 1 Mean ARPU: 780.4920732472237
Group 0 Mean ARPU: 782.2777603863469
Higher weightage in Group 0
```

```
T-Statistic: -0.39411467740575407
P-Value: 0.693496510935397 False
----------------------
```

Observations:

- All t-tests conducted on the services with ARPU yielded false significance, indicating that there is no statistically significant difference in ARPU between customers who use a particular service and those who do not. This suggests that offering or not offering a specific service does not have a significant impact on ARPU.
- However, it's essential to acknowledge that t-tests rely on certain assumptions, such as normality and equal variance.
- Increasing the significance level, for example, to 0.10, allows for a greater probability of rejecting the null hypothesis when it's actually true. While this increases the likelihood of detecting a significant difference between the means of the variables being tested, it also raises the risk of making a Type I error (rejecting the null hypothesis when it's actually true).

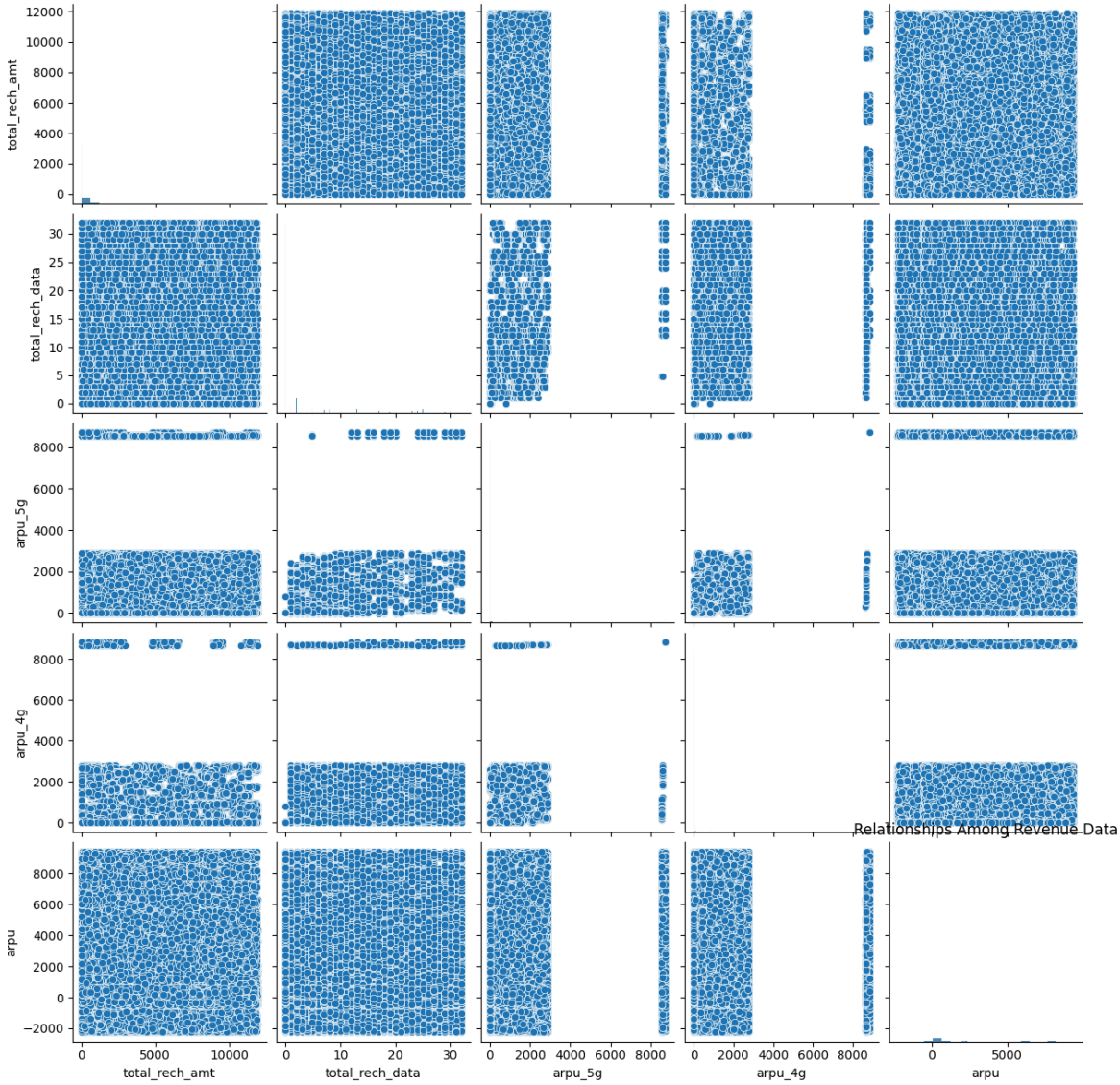
Exploring Relationships with Pair Plots

Pair plots offer a comprehensive view of relationships between variables, facilitating pattern recognition and correlation detection. Each variable is plotted against every other variable, revealing distributions on the diagonal and scatterplots off-diagonally.

Utilizing pair plots aids in understanding correlations, identifying patterns, and spotting outliers or anomalies. It's particularly valuable for determining variables strongly linked to the target, essential in predictive modeling and feature selection.

In [81]:
```
# Visualizing Relationships
plt.figure(figsize=(10,6))
sns.pairplot(df[['total_rech_amt', 'total_rech_data', 'arpu_5g', 'arpu_4g', 'arp
plt.title('Relationships Among Revenue Data')
plt.show()
```

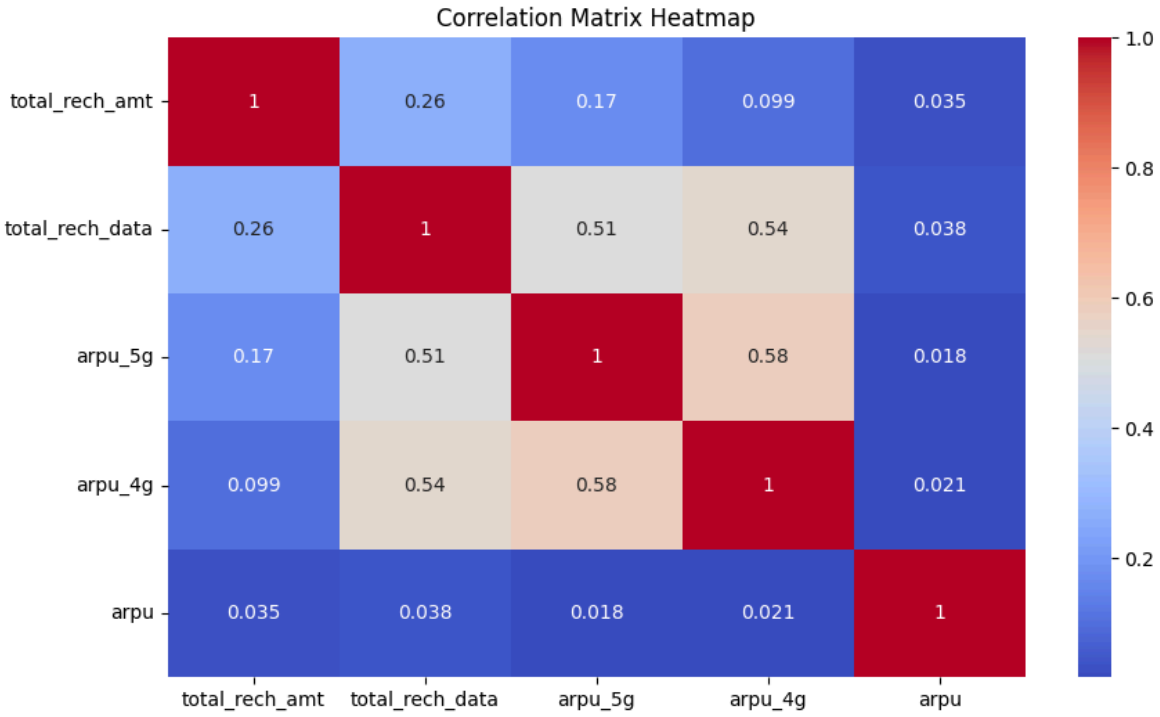
```
<Figure size 1000x600 with 0 Axes>
```



Observations:

From the pair plot analysis, it's evident that there exists a positive correlation between the total recharge amount and ARPU, as well as between the total recharge data and ARPU. This suggests that customers who recharge more frequently and with larger amounts tend to have higher ARPU values.

In [82]:
```
# Create a correlation heatmap
plt.figure(figsize=(10,6))
corr_matrix = df[['total_rech_amt', 'total_rech_data', 'arpu_5g', 'arpu_4g', 'ar
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Observations:

- There is a positive correlation of 0.26 between total recharge amount and total recharge data, implying that customers who recharge more tend to use more data.
- A weak positive correlation of 0.035 exists between total recharge amount and ARPU, indicating that customers who recharge more may have slightly higher ARPU, although the relationship is not very strong.
- Similarly, total recharge data and ARPU exhibit a very weak positive correlation of 0.038, suggesting a minor relationship between data usage and ARPU, but not statistically significant.
- Notably, ARPU 5G and ARPU 4G share a moderate positive correlation of 0.578, suggesting that customers using these services tend to have similar ARPU levels.
- Furthermore, ARPU 5G and total recharge data, as well as ARPU 4G and total recharge data, both exhibit moderate positive correlations of 0.512 and 0.541, respectively, indicating that customers using these services tend to consume more data.

Overall, while some relationships exist between total recharge amount, total recharge data, and ARPU, they are not particularly strong based on the correlation matrix.

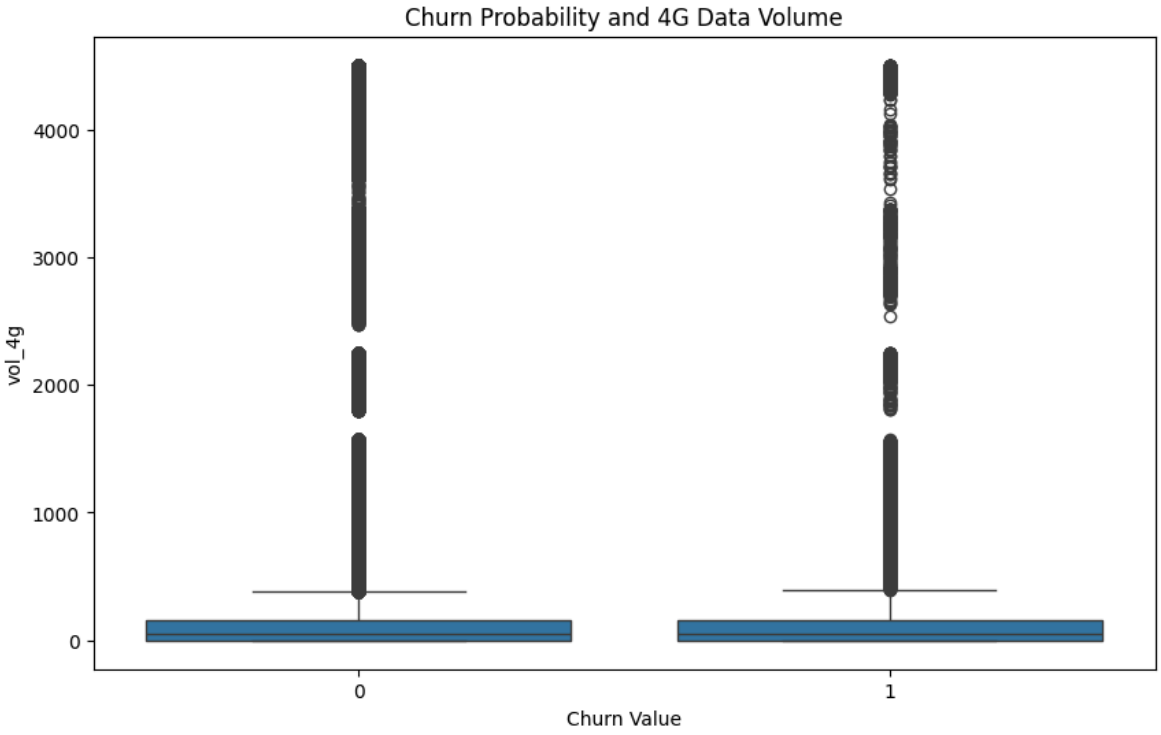
Exploring Customer Lifetime

Customer Lifetime Value (CLTV) serves as a pivotal metric for gauging the total worth of a customer throughout their tenure with the company. By delving into CLTV, I can derive insights crucial for informed decisions on customer acquisition and retention strategies, thereby pinpointing the most lucrative customer segments.

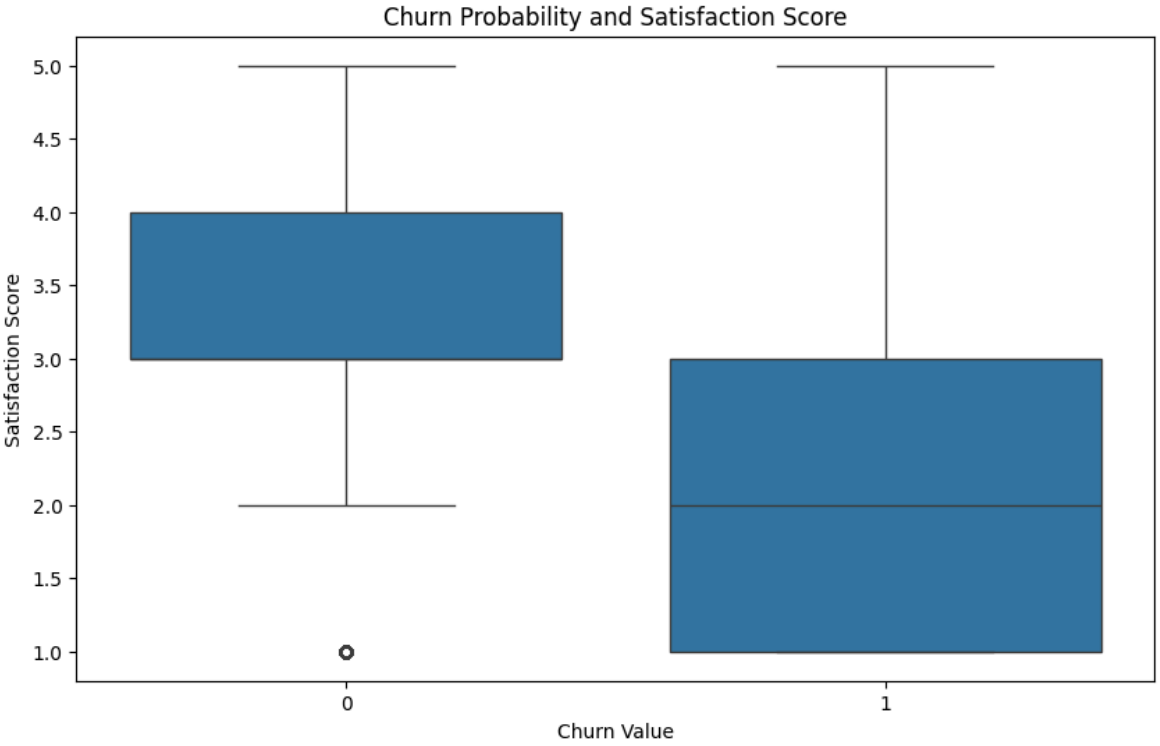
Key Factors and Hypotheses

- **Data Usage Impact:** I hypothesize that customers who extensively use data, particularly 4G and 5G services, would exhibit a higher CLTV. Their heightened reliance on mobile services could translate into prolonged customer tenure and increased willingness to invest in premium services.
- **Satisfaction Score Influence:** Customers with higher satisfaction scores are expected to sustain longer relationships with the company. This hypothesis posits that contented customers are more likely to remain loyal, potentially driving positive word-of-mouth referrals and bolstering revenue.
- **Referral Effect:** Customers who refer friends may exhibit stronger brand loyalty, fostering enduring relationships. This hypothesis underscores the emotional connection fostered through referrals, potentially extending customer lifetimes.
- **Unlimited Data Plans:** Customers availing unlimited data plans may experience enhanced peace of mind, translating into prolonged relationships. This hypothesis suggests that eliminating data usage concerns could foster long-term customer loyalty.
- **Streaming Service Usage:** Increased utilization of streaming services may indicate a higher CLTV, reflecting a greater need for mobile services. This hypothesis suggests that customers heavily invested in streaming services are more inclined to maintain long-term relationships.
- **Average Revenue per User (ARPU):** A higher ARPU might signify a longer customer lifetime, driven by increased service investment. This hypothesis underscores the perceived value customers derive from their expenditures, potentially fostering loyalty and longevity.

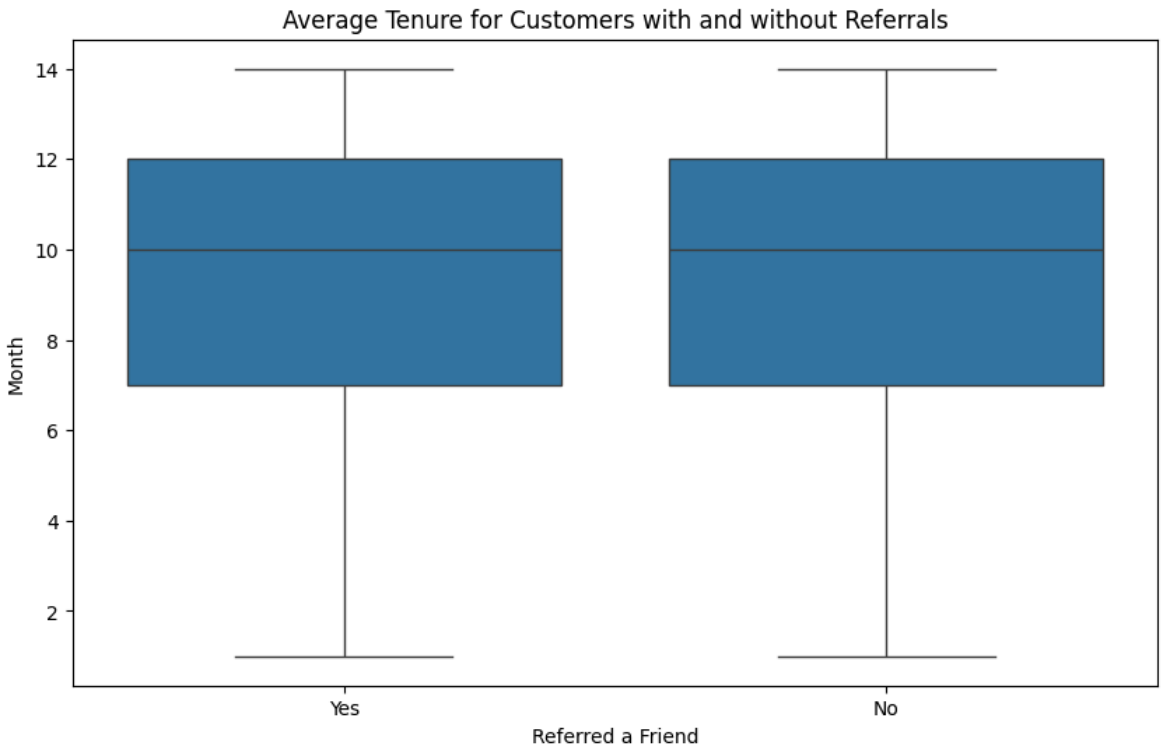
In [83]:
```
# Explore the relationship between data usage and churn likelihood
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn Value', y='vol_4g', data=df)
plt.title("Churn Probability and 4G Data Volume")
plt.show()
```



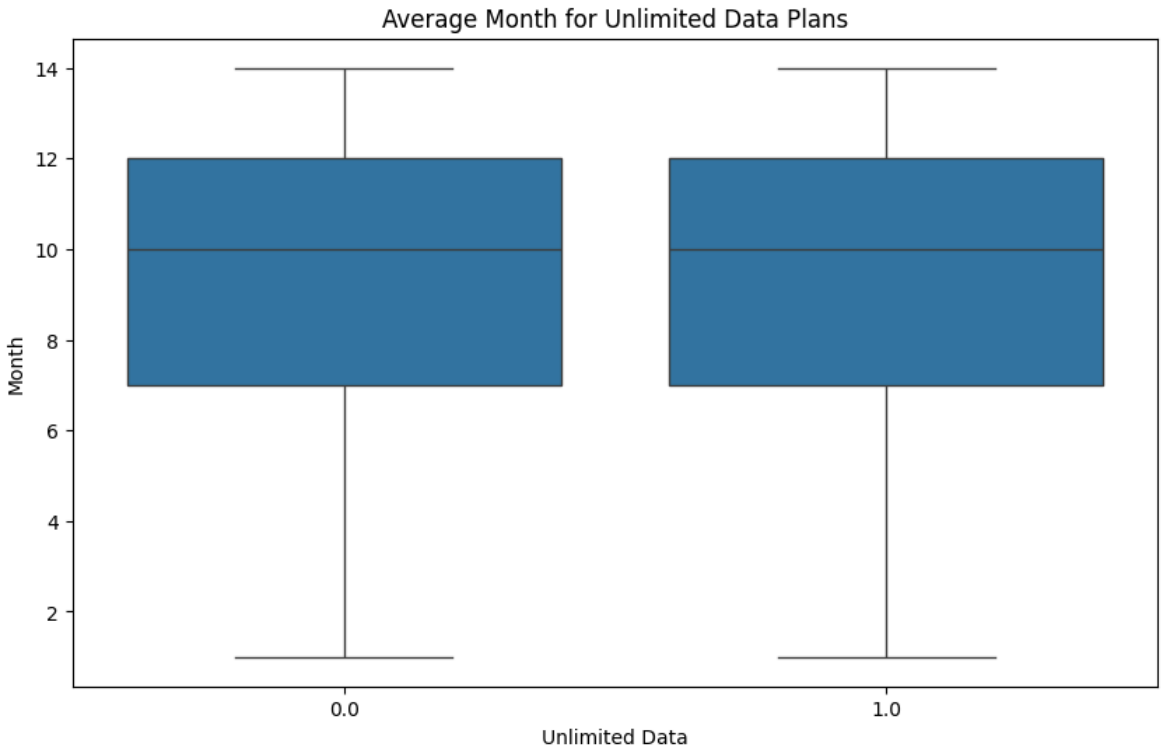
In [84]:
```
# Investigate the relationship between satisfaction score and churn probability
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn Value', y='Satisfaction Score', data=df)
plt.title("Churn Probability and Satisfaction Score")
plt.show()
```



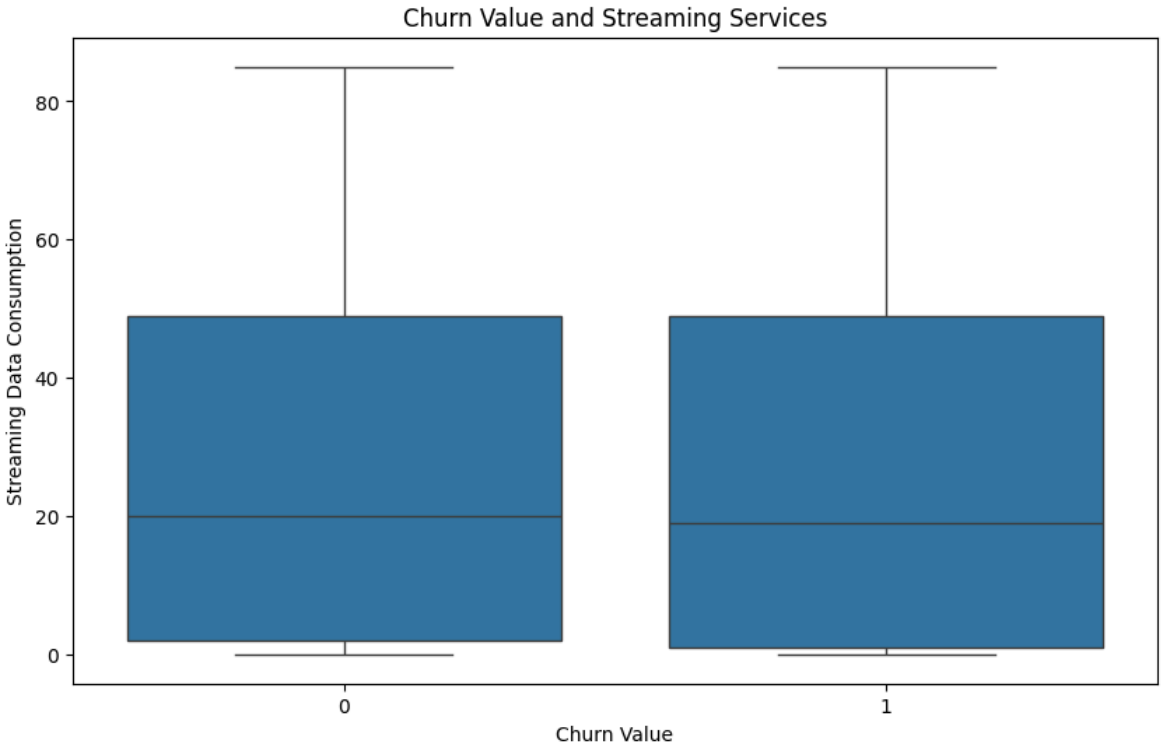
In [85]:
```
# Analyze the relationship between customer referrals and average tenure
plt.figure(figsize=(10,6))
sns.boxplot(x='Referred a Friend', y='Month', data=df)
plt.title("Average Tenure for Customers with and without Referrals")
plt.show()
```



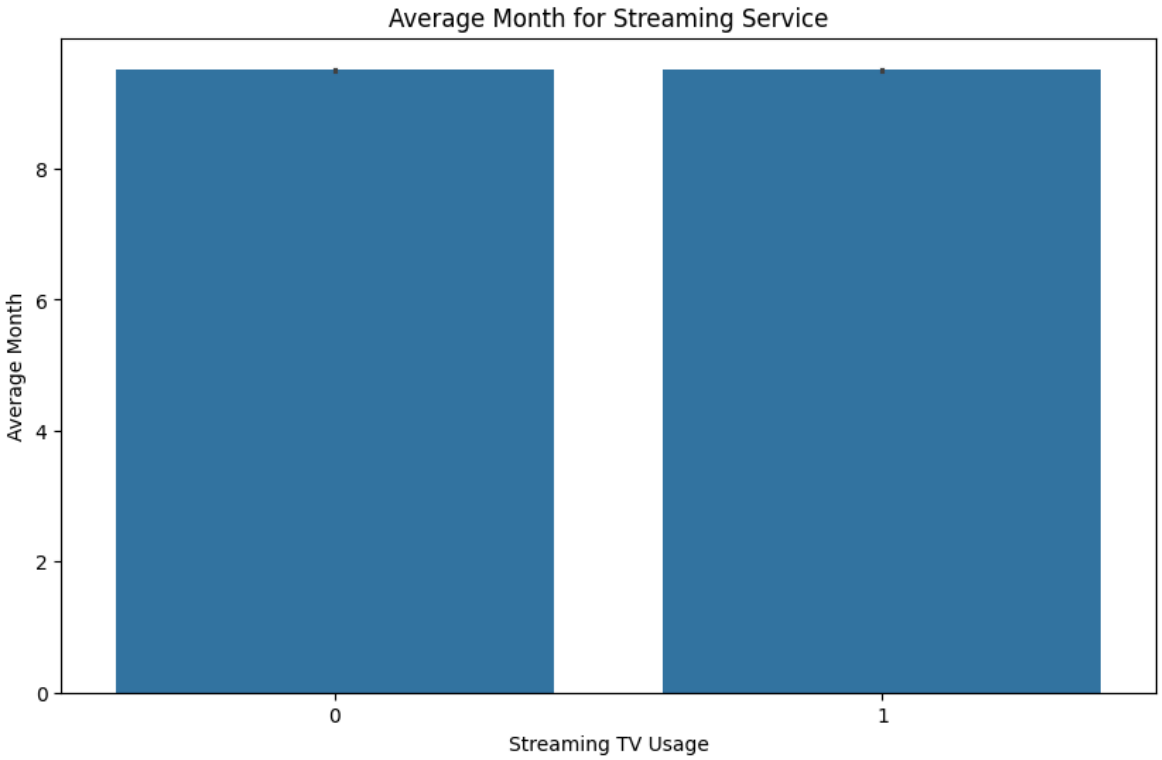
In [86]:
```
# Customers who have unlimited data plans are more likely to have a longer lifet
plt.figure(figsize=(10,6))
sns.boxplot(x='Unlimited Data', y='Month', data=df)
plt.title("Average Month for Unlimited Data Plans")
plt.show()
```



In [87]:
```
# Customers who use more streaming services are less likely to churn.
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn Value', y='Streaming Data Consumption', data=df)
plt.title("Churn Value and Streaming Services")
plt.show()
```

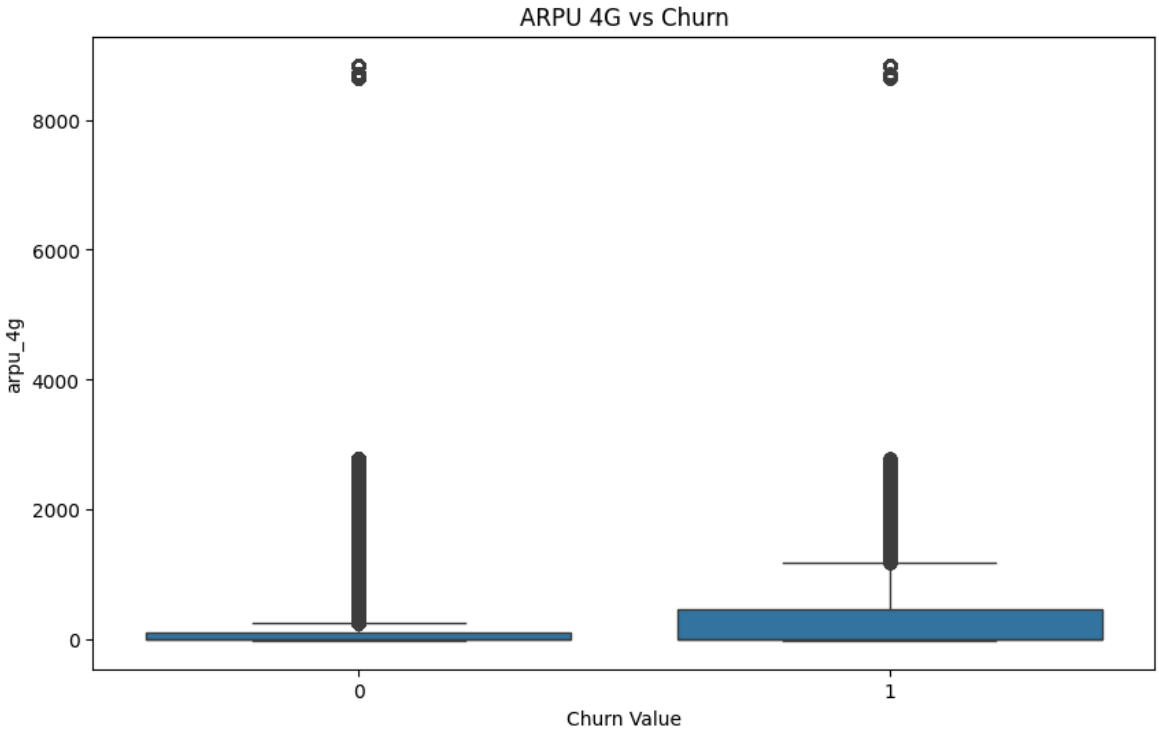


In [88]:
```
# Visualizing the Impact of Streaming Services on Customer Lifetime
plt.figure(figsize=(10,6))
sns.barplot(x='Streaming TV', y='Month', data=df)
plt.title("Average Month for Streaming Service")
plt.xlabel("Streaming TV Usage")
plt.ylabel("Average Month")
plt.show()
```



In [89]:
```
# Create a box plot to visualize the relationship between churn and ARPU 4G
plt.figure(figsize=(10,6))
sns.boxplot(x='Churn Value', y='arpu_4g', data=df)
```

```
plt.title('ARPU 4G vs Churn')
plt.show()
```



Observations:

- Customers who churn tend to have lower satisfaction scores compared to those who don't churn, indicating dissatisfaction as a potential cause for churn.
- Surprisingly, churned customers exhibit higher average revenue per user (ARPU) for 4G services compared to non-churned customers. This suggests that high revenue alone doesn't guarantee customer retention, highlighting the importance of addressing underlying issues beyond financial aspects.
- Exploring additional factors like network coverage, pricing, and customer service quality could provide insights into the reasons for dissatisfaction, aiding in the development of targeted strategies to reduce churn and enhance customer retention.

Understanding Customer Lifetime Value (CLTV)

In this step, I'll delve into the concept of CLTV, a crucial metric for evaluating the overall monetary contribution of customers throughout their relationship with the business. CLTV can be computed through various methodologies, including revenue-based, profit-based, and customer-based approaches. The revenue-based method, which factors in metrics like Average Revenue Per User (ARPU), service costs, and customer longevity, is commonly employed.

Exploring Different CLTV Calculation Methods

- **Historical Method:** This technique relies on past customer behavior, analyzing transactional histories such as average purchase value and relationship duration. It assumes that customer behavior will remain consistent over time.
- **Predictive CLTV:** Utilizing predictive modeling, this method forecasts future customer value by considering demographics, past behavior, and purchase history. However, it necessitates labeled CLTV data and the development of accurate predictive models.
- **Probabilistic CLTV:** By employing probability theory, this approach estimates customer value while accounting for uncertainties in future behavior.
- **Cohort-based CLTV:** Customers are grouped based on when they initiated business with the company (e.g., monthly or quarterly cohorts), facilitating comparisons of cohort behaviors over time.
- **Customer Segmentation CLTV:** This strategy segments customers based on behavior and demographics, enabling tailored CLTV calculations and personalized marketing strategies.

In [90]:
```
# Group by Month of Joining to count new customers for each month
new_customers = df.groupby('Month of Joining')['Customer ID'].nunique().reset_in

# Rename the column to 'New Customers'
new_customers = new_customers.rename(columns={'Customer ID': 'New Customers'})
```

In [91]:
```
# Let's consider only one year of data for analysis
new_customers
```

Out[91]:

	Month of Joining	New Customers
0	1.0	6252
1	2.0	10610
2	3.0	7460
3	4.0	5064
4	5.0	4606
5	6.0	15962
6	7.0	8965
7	8.0	10229
8	9.0	11716
9	10.0	8484
10	11.0	4562
11	12.0	4320

In [92]:
```
# Filter churned customers and calculate monthly loss of customers
churned_customers = df[df['Churn Value'] == 1]

# Group by Month and count churned customers
lost_customers_by_month = churned_customers.groupby('Month')['Churn Value'].coun
lost_customers_by_month
```

Out[92]:
```
Month
1     2278
2     2416
3     2912
4     1522
5     1560
6     2005
7      866
8     1937
9     2452
10    3033
11    3263
12    1675
13    2741
14    1205
Name: Churn Value, dtype: int64
```

In [93]:
```
# Calculating the dynamic churn rates
dynamic_churn_rates = new_customers.merge(
    lost_customers_by_month,
    left_on="Month of Joining",
    right_on="Month"
    ).rename(columns={"Churn Value": "Lost Customers"})
dynamic_churn_rates
```

Out[93]:

	Month of Joining	New Customers	Lost Customers
0	1.0	6252	2278
1	2.0	10610	2416
2	3.0	7460	2912
3	4.0	5064	1522
4	5.0	4606	1560
5	6.0	15962	2005
6	7.0	8965	866
7	8.0	10229	1937
8	9.0	11716	2452
9	10.0	8484	3033
10	11.0	4562	3263
11	12.0	4320	1675

In [94]:
```
# Add a new column for Total Customers
dynamic_churn_rates['Total Customers'] = dynamic_churn_rates['New Customers'].cu
```

```
# Add a new column for Churn Rate
dynamic_churn_rates['Churn Rate'] = dynamic_churn_rates['Lost Customers'] / dyna
```

In [95]:
```
# Rename the column
dynamic_churn_rates = dynamic_churn_rates.rename(columns={"Month of Joining": "M
dynamic_churn_rates
```

Out[95]:

	Month	New Customers	Lost Customers	Total Customers	Churn Rate
0	1.0	6252	2278	6252	0.364363
1	2.0	10610	2416	14584	0.165661
2	3.0	7460	2912	19628	0.148359
3	4.0	5064	1522	21780	0.069881
4	5.0	4606	1560	24864	0.062741
5	6.0	15962	2005	39266	0.051062
6	7.0	8965	866	46226	0.018734
7	8.0	10229	1937	55589	0.034845
8	9.0	11716	2452	65368	0.037511
9	10.0	8484	3033	71400	0.042479
10	11.0	4562	3263	72929	0.044742
11	12.0	4320	1675	73986	0.022639

In [96]:
```
# Merge monthly churn rate to the original data
df = pd.merge(df, dynamic_churn_rates[['Month', 'Churn Rate']], on='Month')
```

In [97]:
```
# Display the first 3 rows of the DataFrame
df.head(3)
```

Out[97]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents
0	hthjctifkiudi0	1	1.0	71638	Female	36.0	No	No
1	tqhiqgvbbhley51	1	1.0	87654	Female	50.0	Not Specified	Yes
2	dvezrgwvxoslg150	1	1.0	88230	Not Specified	59.0	No	No

Now I'll proceed with the calculation of Customer Lifetime Value (CLTV) based on the total Average Revenue Per User (ARPU) as we discussed earlier.

In [98]:
```
# CLTV calculation
df['CLTV_total_arpu'] = (df['total_arpu'] * 0.1) * (1 / df['Churn Rate'])
```

In [99]:
```
# View the first 3 rows of the dataframe
df.head(3)
```

Out[99]:

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependents
0	hthjctifkiudi0	1	1.0	71638	Female	36.0	No	No
1	tqhiqgvbbhley51	1	1.0	87654	Female	50.0	Not Specified	Yes
2	dvezrgwvxoslg150	1	1.0	88230	Not Specified	59.0	No	No

In [100…
```
# Calculate the maximum tenure of each customer
max_tenure = df.groupby('Customer ID')['Tenure'].max().reset_index()
max_tenure
```

Out[100…

	Customer ID	Tenure
0	aaabuvfyemtjj82040	1.0
1	aaamjcrckmkor19758	6.0
2	aaarmkekydoyp67068	7.0
3	aaazpyyafsdxp85262	2.0
4	aabakestdecft46766	1.0
...	...	...
98225	zzzdxcssycihg147958	3.0
98226	zzzhmvfzxlfut88113	7.0
98227	zzzkqzcxfkqgz125167	7.0
98228	zzzsamaczmpqj3670	5.0
98229	zzzusoxnlenoq48949	6.0

98230 rows × 2 columns

Observation:

In the telecom industry, I frequently see customers switching between service providers, so I must identify and categorize them based on their usage behavior and retention potential.**

To properly segment customers, I need to create flag columns such as completed_6_months, churned_within_6_months, and not_completed_6_months.

For example, the completed_6_months flag column will identify customers who have been using the service for the past 6 months without churning. The churned_within_6_months flag column will identify customers who churned within 6 months of service, and the not_completed_6_months flag column will identify customers who are still using the service but have not completed 6 months yet.

By creating these flag columns, I can easily filter and segment customers based on their status, which helps me in targeted marketing, understanding churn behavior, and predicting CLTV.

In [101…
```
# Creating completed 6 months column and encoding them where max tenure is > 6
max_tenure.loc[max_tenure['Tenure'] >= 6, 'completed_6_months'] = 1
```

In [102…
```
def create_flags(df):
    """
    Create flag columns indicating customer status based on their tenure and chu

    Args:
    df (DataFrame): Input DataFrame containing customer data.

    Returns:
    DataFrame: DataFrame with flag columns indicating customer status.
    """
    # Initialize flags to zero
    df['completed_6_months'] = 0
    df['churned_within_6_months'] = 0
    df['not_completed_6_months'] = 0

    # Find customers who have completed at least 6 months
    max_tenure = df.groupby('Customer ID')['Tenure'].max().reset_index()
    completed_ids = max_tenure.loc[max_tenure['Tenure'] >= 6, 'Customer ID']
    df.loc[df['Customer ID'].isin(completed_ids), 'completed_6_months'] = 1

    churned_ids = df.loc[(df['Churn Value'] == 1) & (df['Month'] == df.groupby('
    # group by customer ID and find the maximum tenure
    max_tenure = df.groupby('Customer ID')['Tenure'].max().reset_index()

    # filter again to include only the rows where the maximum tenure is less tha
    filtered_max_tenure = max_tenure[max_tenure['Tenure'] < 6]
    customer_ids_filtered = filtered_max_tenure['Customer ID'].tolist()
    df.loc[df['Customer ID'].isin(customer_ids_filtered), 'churned_within_6_mont

    # Find customers who have not completed 6 months and have not churned
    not_completed_ids = df.groupby('Customer ID')['Tenure'].count().reset_index(
    not_completed_ids = not_completed_ids.loc[(not_completed_ids['Tenure'] < 6)
    df.loc[df['Customer ID'].isin(not_completed_ids), 'not_completed_6_months']

    return df
```

In [103…
```
# Create flags indicating customer status
df = create_flags(df)
```

In [104…
```
# Verify flags for a specific customer
df[df['Customer ID'] == "tqhiqgvbbhley51"]
```

Out[104…

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Dependent
1	tqhiqgvbbhley51	1	1.0	87654	Female	50.0	Not Specified	Ye
431020	tqhiqgvbbhley51	2	1.0	87654	Female	50.0	Not Specified	Ye
445604	tqhiqgvbbhley51	3	1.0	87654	Female	50.0	Not Specified	Ye
465233	tqhiqgvbbhley51	4	1.0	87654	Female	50.0	Not Specified	Ye

In [105…
```
# Verify if the customer has completed 6 months
df[df['Customer ID'] == 'aaamjcrckmkor19758']
```

Out[105…

	Customer ID	Month	Month of Joining	zip_code	Gender	Age	Married	Depen
51008	aaamjcrckmkor19758	7	7.0	94061	Male	33.0	Not Specified	
98430	aaamjcrckmkor19758	8	7.0	94061	Male	33.0	Not Specified	
155181	aaamjcrckmkor19758	9	7.0	94061	Male	33.0	Not Specified	
221325	aaamjcrckmkor19758	10	7.0	94061	Male	33.0	Not Specified	
292894	aaamjcrckmkor19758	11	7.0	94061	Male	33.0	Not Specified	
365937	aaamjcrckmkor19758	12	7.0	94061	Male	33.0	Not Specified	

Usage Patterns and Deviations

As I delve into understanding customer behavior in the telecom industry, I prioritize analyzing outgoing call usage and data usage as they serve as crucial revenue drivers. To

derive insights into monthly deviations, I compute the variance between the current month's usage and the previous month's usage.

First, I streamline the dataset by removing irrelevant columns like incoming usage, which is typically free, and outgoing usage to the call center, also assumed to be free. Then, I employ the .diff() method to calculate the deviations, capturing the differences between consecutive rows.

By scrutinizing these usage patterns and monthly deviations, I aim to uncover valuable insights such as identifying heavy users of outgoing calls or data, and pinpointing customers experiencing significant fluctuations in usage from one month to the next.

In [106…

```
# First, let's streamline the dataset by removing unnecessary columns
# Assuming incoming usage is free, we retain outgoing call usage, data usage, an

cols_to_keep = [
    'Customer ID', 'Month', 'Month of Joining', 'Gender', 'Age', 'Married',
    'Dependents', 'arpu',  'roam_og', 'loc_og_t2t', 'loc_og_t2m', 'loc_og_t2f',
    'std_og_t2t', 'std_og_t2m', 'std_og_t2f', 'isd_og', 'spl_og', 'og_others',
    'total_rech_amt', 'total_rech_data', 'vol_4g', 'vol_5g', 'arpu_5g',
    'arpu_4g', 'Churn Value', 'Referred a Friend', 'Number of Referrals',
    'Phone Service', 'Multiple Lines', 'Internet Service', 'Internet Type',
    'Streaming Data Consumption', 'Online Security', 'Online Backup',
    'Device Protection Plan', 'Premium Tech Support', 'Streaming TV',
    'Streaming Movies', 'Streaming Music', 'Unlimited Data', 'Payment Method',
    'Satisfaction Score', 'Churn Category', 'Churn Reason', 'total_recharge',
    'offer', 'Tenure', 'total_arpu', 'Churn Rate', 'outgoing_call_usage',
    'data_usage', 'CLTV_total_arpu', 'completed_6_months',
    'churned_within_6_months', 'not_completed_6_months']

df = df[cols_to_keep]
```

In [107…

```
def normalized_col(df_col, req_int_min, req_int_max):
    """
    Normalize values in a DataFrame column between a specified range.

    Parameters:
        df_col (pandas.Series): Input column to be normalized.
        req_int_min (float): Minimum value of the desired range.
        req_int_max (float): Maximum value of the desired range.

    Returns:
        pandas.Series: Normalized column within the specified range.
    """
    # Find the minimum and maximum values of the input column
    min_col = df_col.min()
    max_col = df_col.max()

    # normalize the column values between req_int_min and req_int_max
    df_norm_col = ((df_col - min_col) / (max_col - min_col) * (req_int_max - req

    # Return the normalized column
    return df_norm_col
```

In [108…

```
# Define columns for which deviation will be calculated
dev_cols = ['total_recharge', 'outgoing_call_usage', 'data_usage', 'Satisfaction
```

```
# Calculate the deviation for each column
for col in dev_cols:
    # Generate the column name for deviation
    col_name = col + '_dev'
    # Calculate deviation and fill NaN with 0
    df[col_name] = df.groupby('Customer ID')[col].diff().fillna(0)

# Normalize the deviation columns between 1 and 10
for col in dev_cols:
    # Normalize deviation column
    df[col + '_dev'] = normalized_col(df[col + '_dev'], 1, 10)
```

In [109…

```
# Display the first 3 rows of the DataFrame
df.head(3)
```

Out[109…

	Customer ID	Month	Month of Joining	Gender	Age	Married	Dependents	arpu	ro
0	hthjctifkiudi0	1	1.0	Female	36.0	No	No	273.07	
1	tqhiqgvbbhley51	1	1.0	Female	50.0	Not Specified	Yes	401.95	
2	dvezrgwvxoslg150	1	1.0	Not Specified	59.0	No	No	378.91	

In [110…

```
# Check for missing values in the DataFrame and sum them up column-wise
df.isnull().sum()
```

Out[110…

Customer ID	0
Month	0
Month of Joining	0
Gender	0
Age	0
Married	0
Dependents	0
arpu	0
roam_og	0
loc_og_t2t	0
loc_og_t2m	0
loc_og_t2f	0
std_og_t2t	0
std_og_t2m	0
std_og_t2f	0
isd_og	0
spl_og	0
og_others	0
total_rech_amt	0
total_rech_data	0
vol_4g	0
vol_5g	0
arpu_5g	0
arpu_4g	0
Churn Value	0
Referred a Friend	0
Number of Referrals	304
Phone Service	0
Multiple Lines	36084
Internet Service	0
Internet Type	0
Streaming Data Consumption	0
Online Security	0
Online Backup	0
Device Protection Plan	0
Premium Tech Support	0
Streaming TV	0
Streaming Movies	0
Streaming Music	0
Unlimited Data	8664
Payment Method	0
Satisfaction Score	0
Churn Category	0
Churn Reason	0
total_recharge	0
offer	0
Tenure	0
total_arpu	0
Churn Rate	0
outgoing_call_usage	0
data_usage	0
CLTV_total_arpu	0
completed_6_months	0
churned_within_6_months	0
not_completed_6_months	0
total_recharge_dev	0
outgoing_call_usage_dev	0
data_usage_dev	0
Satisfaction Score_dev	0

dtype: int64

I observed that the `Number of Referrals` is not been used significantly!

Next Steps for Customer Segmentation

As I move forward with customer segmentation, I will implement the following steps to categorize customers into four distinct groups based on their behavior and engagement:

Determining Loyalty and Retention Scores

I will calculate the Loyalty Score and Retention Score for each customer using the provided formulas:

- **Loyalty Score** = (Total Recharge Deviation * w1) + (Usage Deviation * w2) + (Satisfaction Deviation * w3)
- **Retention Score** = (Total Recharge Deviation * w4) + (Usage Deviation * w5) + (Satisfaction Deviation * w6)

Adjusting the weights (w1, w2, w3, w4, w5, w6) will allow me to fine-tune the importance of each parameter based on business requirements and its impact on customer loyalty and retention.

Categorizing Customer Segments

Based on the computed scores and customer history, I will categorize customers into the following segments:

- **New Customers:** Customers who have joined the company recently and have not completed six months. For this group, I will emphasize the importance of satisfaction deviation due to their limited history with the company.
- **Churned Customers:** Customers who have left the company within six months. Although their loyalty score is zero, I will calculate retention scores, placing higher importance on customer satisfaction deviation to understand factors contributing to churn.
- **Loyal Customers:** Customers who have completed six months with the company and exhibit strong brand attachment. Total recharge deviation will be the primary focus for this segment, with lesser emphasis on usage deviation and satisfaction deviation.
- **Retained Customers:** Customers who have completed six months with the company and demonstrate high engagement levels. Here, I will prioritize total recharge deviation and satisfaction deviation while considering usage deviation as less significant.

By applying these steps, I will create meaningful customer segments that provide insights into customer behavior, loyalty, and retention, allowing for targeted strategies and personalized approaches to enhance customer satisfaction and retention rates.

In [111…

```
# Define weights for different customer segments:
# - "not_completed_6_months": Weights assigned to parameters for customers who h
#                             not completed six months with the company.
# - "churned_within_6_months": Weights assigned to parameters for customers who
#                             left the company within six months.
# - "completed_6_months": Weights assigned to parameters for customers who have
#                             six months with the company.

customer_weights = { "not_completed_6_months": {"w1": 0.2,"w2": 0.3, "w3": 0.5,
                     "churned_within_6_months": { "w1": 0, "w2": 0, "w3": 0, "w4
                     "completed_6_months": { "w1": 0.5,"w2": 0.3,"w3": 0.2, "w4"
                     }
```

In [112…

```
def calculate_scores(weight_dic, total_recharge_dev, usage_dev, satisfaction_sco
    """
    Calculate loyalty and retention scores based on given weights and deviation

    Parameters:
        weight_dic (dict): Dictionary containing weights for each parameter.
        total_recharge_dev (float): Deviation value for total recharge.
        usage_dev (float): Deviation value for usage.
        satisfaction_score_dev (float): Deviation value for satisfaction score.

    Returns:
        tuple: Loyalty score and retention score.
    """
    wt1 = weight_dic['w1']
    wt2 = weight_dic['w2']
    wt3 = weight_dic['w3']
    wt4 = weight_dic['w4']
    wt5 = weight_dic['w5']
    wt6 = weight_dic['w6']

    loyalty_score = wt1 * total_recharge_dev + wt2 * usage_dev + wt3 * satisfact
    retention_score = wt4 * total_recharge_dev + wt5 * usage_dev + wt6 * satisfa

    return loyalty_score, retention_score
```

In [113…

```
def get_customer_category(row):
    """
    Categorize customers based on their behavior and engagement.

    Parameters:
        row (pandas.Series): Row of DataFrame representing a customer.

    Returns:
        tuple: Customer category, loyalty score, and retention score.
    """
    # Extract feature values
    total_recharge_dev = row['total_recharge_dev']
    outgoing_call_usage_dev = row['outgoing_call_usage_dev']
    data_usage_dev = row['data_usage_dev']
    satisfaction_score_dev = row['Satisfaction Score_dev']
```

```
    usage_dev = (outgoing_call_usage_dev + data_usage_dev)/2
    # Define loyalty and retention weights


    # Categorize new customers
    if row['not_completed_6_months'] == 1:

        loyalty_score, retention_score = calculate_scores(customer_weights['not_
        return 'new_customer', loyalty_score, retention_score

    # Categorize churned customers
    elif row['churned_within_6_months'] == 1:
        loyalty_score, retention_score = calculate_scores(customer_weights['chur

        return 'churned_within_6_months', loyalty_score, retention_score

    # Categorize existing customers
    elif row['completed_6_months'] == 1:

        loyalty_score, retention_score = calculate_scores(customer_weights['comp
        # Categorize based on loyalty and retention scores
        if loyalty_score > retention_score:
            return 'loyal', loyalty_score, retention_score

        elif row['Churn Value'] == 1:
            return 'churned', loyalty_score, retention_score

        elif loyalty_score < retention_score:
            return 'retained', loyalty_score, retention_score
```

In [114…

```
def assign_customer_segments(df):
    """
    Assign customer segments to each customer in the DataFrame.

    Parameters:
        df (pandas.DataFrame): DataFrame containing customer data.

    Returns:
        pandas.DataFrame: DataFrame with assigned customer segments.
    """
    # create columns for loyalty and retention scores
    df['loyalty_score'] = 0
    df['retention_score'] = 0

    # apply the function to each row
    df[['customer_segment', 'loyalty_score', 'retention_score']] = df.apply(get_

    # get the last assigned segment for each customer
    last_segment = df.groupby('Customer ID')['customer_segment'].last()

    # map the last segment to all rows of each customer
    df['customer_segment'] = df['Customer ID'].map(last_segment)

    return df
```

In [115…

```
# Make a copy of the original DataFrame
df_final = df.copy()
```

```
# Assign customer segments to the copied DataFrame
df_final = assign_customer_segments(df_final)
```

Observations:

As I'm dealing with these new customers, I should prioritize retention over loyalty. Thus, I'll assign more weight to retention in the score calculation.

It's essential to understand that calculating the loyalty score involves considering various factors like total recharge, usage, and satisfaction. Therefore, a high loyalty score suggests that despite deviations in behavior, the customer still demonstrates loyalty traits based on their overall interactions with the company.

In [116…

```
# Display the first few rows of the DataFrame
df_final.head()
```

Out[116…

	Customer ID	Month	Month of Joining	Gender	Age	Married	Dependents	arpu	rc
0	hthjctifkiudi0	1	1.0	Female	36.0	No	No	273.07	
1	tqhiqgvbbhley51	1	1.0	Female	50.0	Not Specified	Yes	401.95	
2	dvezrgwvxoslg150	1	1.0	Not Specified	59.0	No	No	378.91	
3	vtgzwaptuamhj154	1	1.0	Female	25.0	No	No	584.89	
4	vsugxqvomxetj156	1	1.0	Female	34.0	Yes	Not Specified	571.93	

In [117…

```
# Retrieve the column names of the DataFrame
df_final.columns
```

Out[117…
```
Index(['Customer ID', 'Month', 'Month of Joining', 'Gender', 'Age', 'Married',
       'Dependents', 'arpu', 'roam_og', 'loc_og_t2t', 'loc_og_t2m',
       'loc_og_t2f', 'std_og_t2t', 'std_og_t2m', 'std_og_t2f', 'isd_og',
       'spl_og', 'og_others', 'total_rech_amt', 'total_rech_data', 'vol_4g',
       'vol_5g', 'arpu_5g', 'arpu_4g', 'Churn Value', 'Referred a Friend',
       'Number of Referrals', 'Phone Service', 'Multiple Lines',
       'Internet Service', 'Internet Type', 'Streaming Data Consumption',
       'Online Security', 'Online Backup', 'Device Protection Plan',
       'Premium Tech Support', 'Streaming TV', 'Streaming Movies',
       'Streaming Music', 'Unlimited Data', 'Payment Method',
       'Satisfaction Score', 'Churn Category', 'Churn Reason',
       'total_recharge', 'offer', 'Tenure', 'total_arpu', 'Churn Rate',
       'outgoing_call_usage', 'data_usage', 'CLTV_total_arpu',
       'completed_6_months', 'churned_within_6_months',
       'not_completed_6_months', 'total_recharge_dev',
       'outgoing_call_usage_dev', 'data_usage_dev', 'Satisfaction Score_dev',
       'loyalty_score', 'retention_score', 'customer_segment'],
      dtype='object')
```

In [118…
```
# Count the occurrences of each value in the 'customer_segment' column
df_final['customer_segment'].value_counts()
```

Out[118…
```
customer_segment
loyal                     171620
retained                  169225
new_customer              118986
churned_within_6_months    50399
churned                     1642
Name: count, dtype: int64
```

In [119…
```
# Sort the DataFrame by 'Customer ID' and 'Month'
sorted_df = df_final.sort_values(by=['Customer ID', 'Month'])
```

In [120…
```
# Taking summary of journey through tenure
customer_df = sorted_df.groupby('Customer ID').agg({
    'Age': 'first',
    'Month of Joining': 'first',
    'Gender': 'first',
    'Married': 'first',
    'Dependents': 'first',
    'arpu': 'mean',
    'roam_og': 'mean',
    'loc_og_t2t': 'mean',
    'loc_og_t2m': 'mean',
    'loc_og_t2f': 'mean',
    'std_og_t2t': 'mean',
    'std_og_t2m': 'mean',
    'std_og_t2f': 'mean',
    'isd_og': 'mean',
    'spl_og': 'mean',
    'og_others': 'mean',
    'total_rech_amt': 'sum',
    'total_rech_data': 'sum',
    'vol_4g': 'mean',
    'vol_5g': 'mean',
    'arpu_5g': 'mean',
    'arpu_4g': 'mean',
    'Churn Value': 'max',
    'Referred a Friend': 'max',
```

```
    'Number of Referrals': 'sum',
    'Phone Service': 'last',
    'Multiple Lines': 'last',
    'Internet Service': 'last',
    'Internet Type': 'last',
    'Streaming Data Consumption': 'last',
    'Online Security': 'last',
    'Online Backup': 'last',
    'Device Protection Plan': 'last',
    'Premium Tech Support': 'last',
    'Streaming TV': 'last',
    'Streaming Movies': 'last',
    'Streaming Music': 'last',
    'Unlimited Data': 'last',
    'Satisfaction Score': 'mean',
    'Churn Category': 'last',
    'Churn Reason': 'last',
    'Tenure': 'max',
    'total_recharge': 'sum',
    'outgoing_call_usage': 'sum',
    'data_usage': 'sum',
    'CLTV_total_arpu': 'sum',
    'total_recharge_dev': 'mean',
    'outgoing_call_usage_dev': 'mean',
    'data_usage_dev': 'mean',
    'Satisfaction Score_dev': 'mean',
    'loyalty_score': 'last',
    'retention_score': 'last',
    'customer_segment': 'last',
    'completed_6_months': 'max',
    'churned_within_6_months': 'max',
    'not_completed_6_months': 'max'
}).reset_index()
```

I will carefully examine the `loyal`, `retained` and `:churned` segments individually.

Loyal Customer Segment

In [121…

```
# Filter for all loyal customers
loyal_customers = customer_df[customer_df['customer_segment'] == 'loyal'].reset_
```

In [122…

```
# Display the first 3 rows of loyal customers
loyal_customers.head(3)
```

Out[122...

	Customer ID	Age	Month of Joining	Gender	Married	Dependents	arpu	r
0	aaarmkekydoyp67068	40.0	6.0	Female	Yes	Yes	1359.390000	60
1	aaczaykaqywmw75824	45.0	2.0	Male	No	Not Specified	167.559091	55
2	aadddwgslgkhi101735	33.0	2.0	Female	Yes	Yes	117.070000	302

In [123...

```
# Analyze one specific loyal customer
loyal_customers[loyal_customers['Customer ID'] == "vtgzwaptuamhj154"]
```

Out[123...

	Customer ID	Age	Month of Joining	Gender	Married	Dependents	arpu	roa
17974	vtgzwaptuamhj154	25.0	1.0	Female	No	No	339.58875	433.

In [124...

```
# Sort loyal customers by CLTV in descending order
top_customers = loyal_customers.sort_values(by='CLTV_total_arpu', ascending=Fals

# Display top 10 customers by CLTV
top_customers.head(10)
```

Out[124...

	Customer ID	Age	Month of Joining	Gender	Married	Dependents	arpu
5399	gmpsofakdnaxa81274	35.0	1.0	Male	Yes	Not Specified	1022.82000(
21275	zycwgwajgizts53811	27.0	4.0	Female	No	No	861.92888(
1848	cgiislaunjcej92126	61.0	6.0	Female	Yes	Yes	1193.47000(
714	axdlwhwsvpfdt160486	68.0	6.0	Male	No	Yes	770.601429
6125	hikvxddqquoyw128515	36.0	2.0	Female	Not Specified	No	1115.571818
18840	wwjwgdvihfvcj113784	31.0	2.0	Male	Yes	Not Specified	426.496364
20066	ykfrhabwtpwal63288	41.0	2.0	Male	No	No	1132.782727
4700	frbhanghojaod2768	33.0	3.0	Male	No	No	1281.761000
19260	xkglastxejkuw24869	36.0	2.0	Female	Not Specified	Yes	2132.490909
5895	hbutygzjnjkgq73818	41.0	7.0	Male	No	No	1012.895000

In [125...
```
# Calculate average order value for loyal customers
avg_order_value = loyal_customers['total_rech_amt'].sum() / loyal_customers.shap
avg_order_value
```

Out[125... 13528.968848081699

In [126...
```
# Calculate customer lifespan for loyal customers
lifespan = loyal_customers['Tenure'].mean()
lifespan
```

Out[126... 8.039537171499509

In [127...
```
# Calculate average recharge amount per month for loyal customers
avg_recharge_per_month = avg_order_value / lifespan
avg_recharge_per_month
```

Out[127... 1682.804440041953

In [128...
```
# Calculate the sum of service subscriptions for loyal customers
service_counts = loyal_customers[services].sum().sort_values(ascending=False)
```

```
# Display the top 5 most popular services among loyal customers
print('Top 5 Most Popular Services Among Loyal Customers:')
service_counts.head(5)
```

```
Top 5 Most Popular Services Among Loyal Customers:
```

Out[128…

```
Internet Service          13172.0
Unlimited Data            11809.0
Device Protection Plan    10655.0
Streaming TV              10376.0
Streaming Movies          10346.0
dtype: float64
```

Observations:

Based on my analysis, I found that the top 5 most popular services among loyal customers are Internet Service, Unlimited Data, Device Protection Plan, Streaming TV, and Streaming Movies. These services have the highest count of usage among loyal customers, indicating a high level of satisfaction and interest in them.

This insight is valuable as it allows me to tailor marketing strategies to encourage new customers to sign up for these popular services and to focus on retaining existing customers by emphasizing the benefits of these services. Additionally, it enables me to allocate resources effectively and prioritize improvements and updates for these popular services to further enhance customer satisfaction and loyalty.

Churned Customer Segment

In [129…

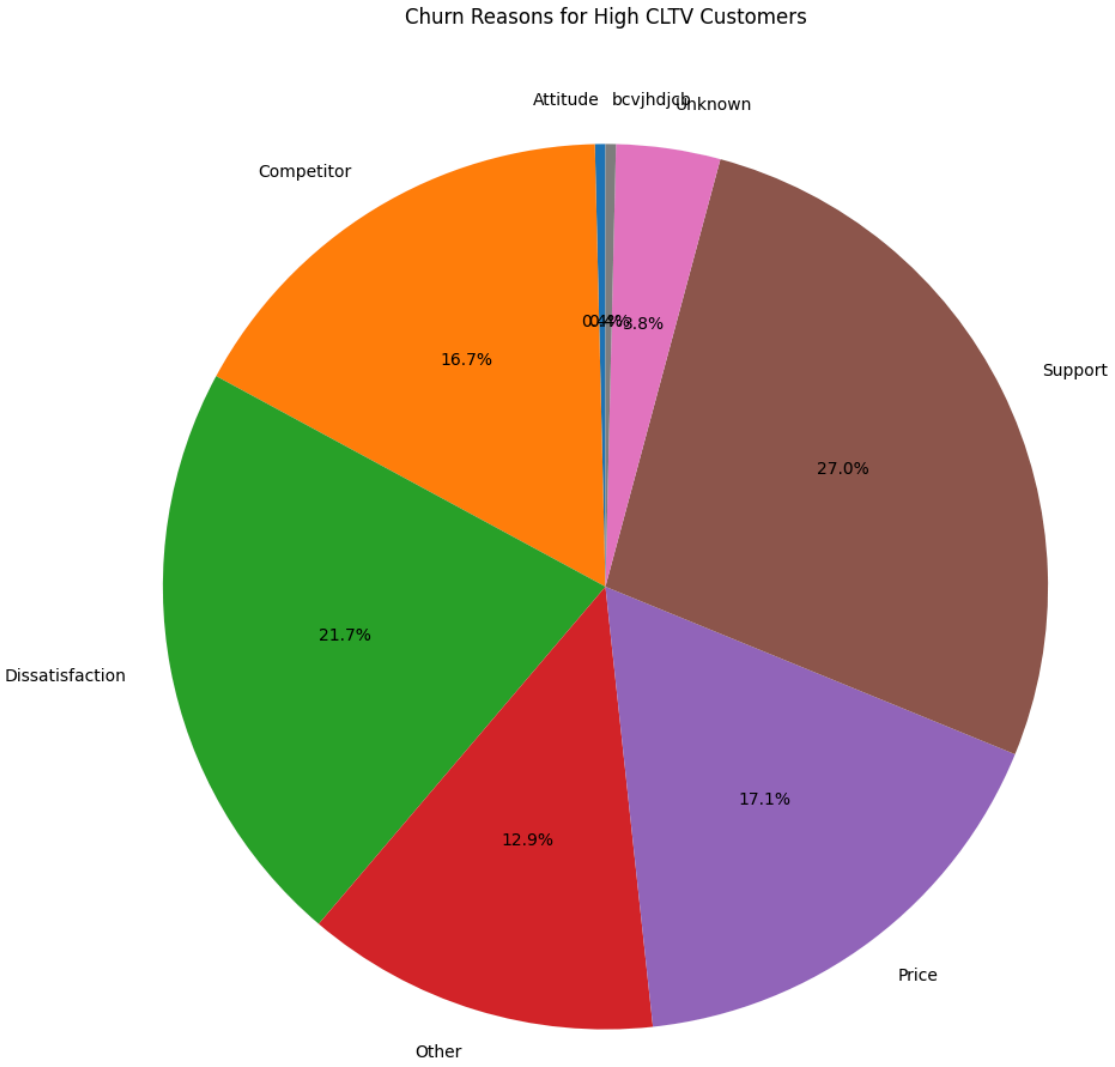
```
# Extract churned customers for analysis
churned_customers = customer_df[customer_df['customer_segment'] == 'churned'].re
```

In [130…

```
# Group by churn reason and calculate count
churn_reason_counts = churned_customers.groupby('Churn Category')['Customer ID']

# Plot pie chart
plt.figure(figsize=(20,12))
churn_reason_counts.plot(kind='pie', autopct='%1.1f%%', startangle=90, ylabel=''
```

Out[130…

```
<Axes: title={'center': 'Churn Reasons for High CLTV Customers'}>
```



Observations:

Upon analyzing the churn reasons for customers who churned after completing 6 months with the company, I observed a variety of factors contributing to churn. Notably, the most common factor appears to be support, followed by dissatisfaction and price concerns.

Recommendations:

- To mitigate churn, I recommend focusing on enhancing support services, particularly online and phone support channels. This can involve investing in staff training and development to improve expertise and service quality, thereby reducing customer frustration and churn rates.
- It's crucial to identify and address specific pain points leading to churn among high CLTV customers. Conducting targeted surveys and gathering feedback from churned customers can provide valuable insights into areas needing improvement, allowing for tailored solutions to enhance customer satisfaction and retention.
- Addressing pricing concerns is vital. Implementing dynamic pricing strategies and offering competitive prices and promotions can help retain high-value customers by

providing them with attractive incentives to stay loyal to the company.

Retained Customer Segment

In [131…
```
# Select the data for retained customers
retained_customers = customer_df[customer_df['customer_segment'] == 'retained'].
```

In [132…
```
# Calculate the average CLTV for retained customers
avg_cltv = retained_customers['CLTV_total_arpu'].mean()

# Display the average CLTV for retained customers
print('Average CLTV for Retained Customers:', round(avg_cltv, 2))
```

```
Average CLTV for Retained Customers: 19309.49
```

Observations:

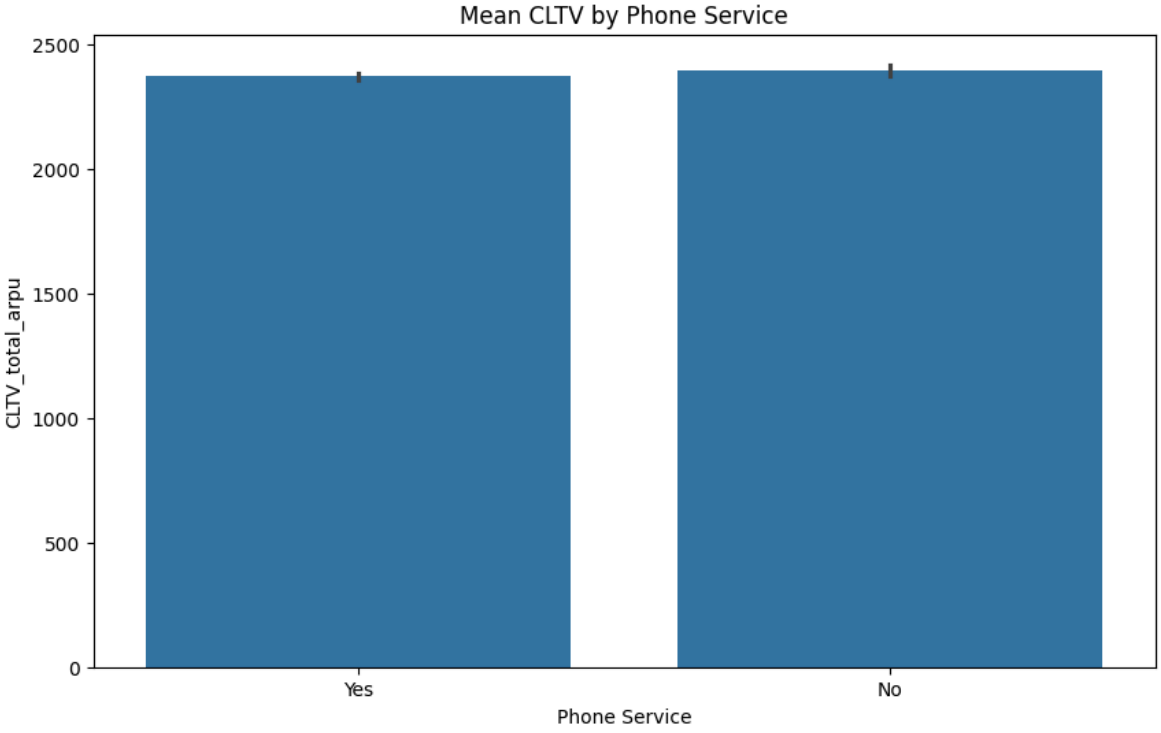
I observe that the average CLTV value for retained customers, $19,309.49, is significantly higher than that of loyal customers$,13,528.97. This indicates a considerable potential for revenue growth and enhanced profitability through customer retention strategies.

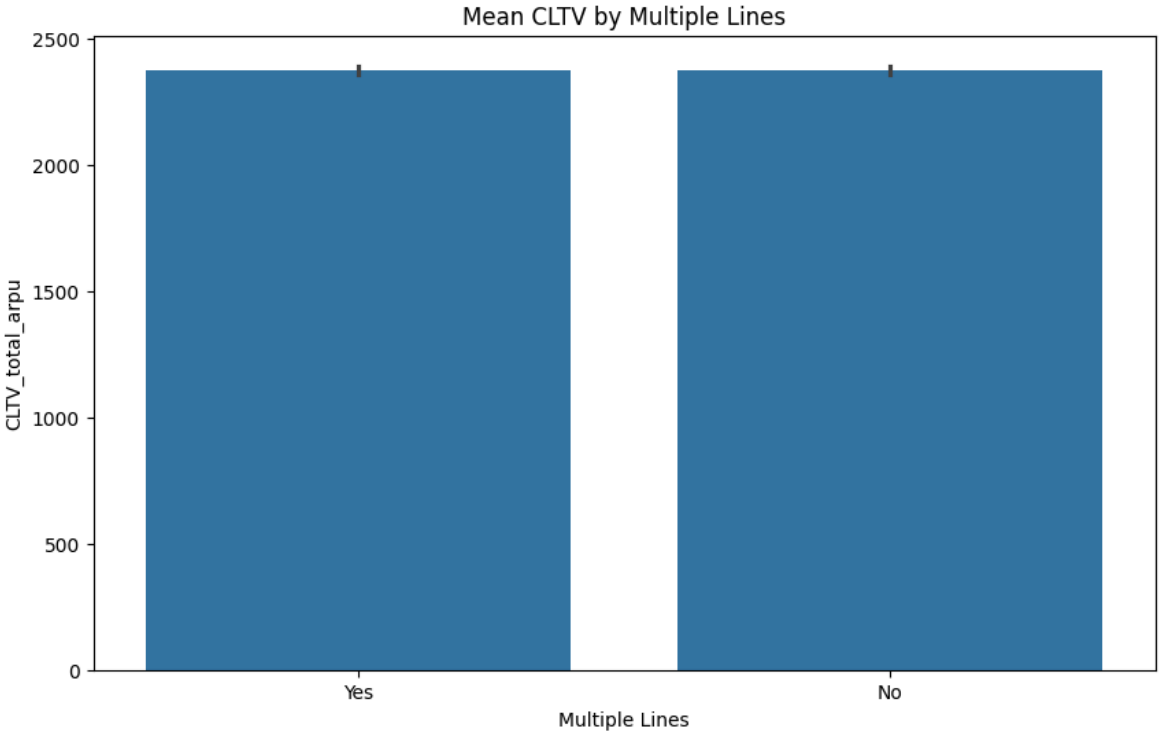
Recommendations:

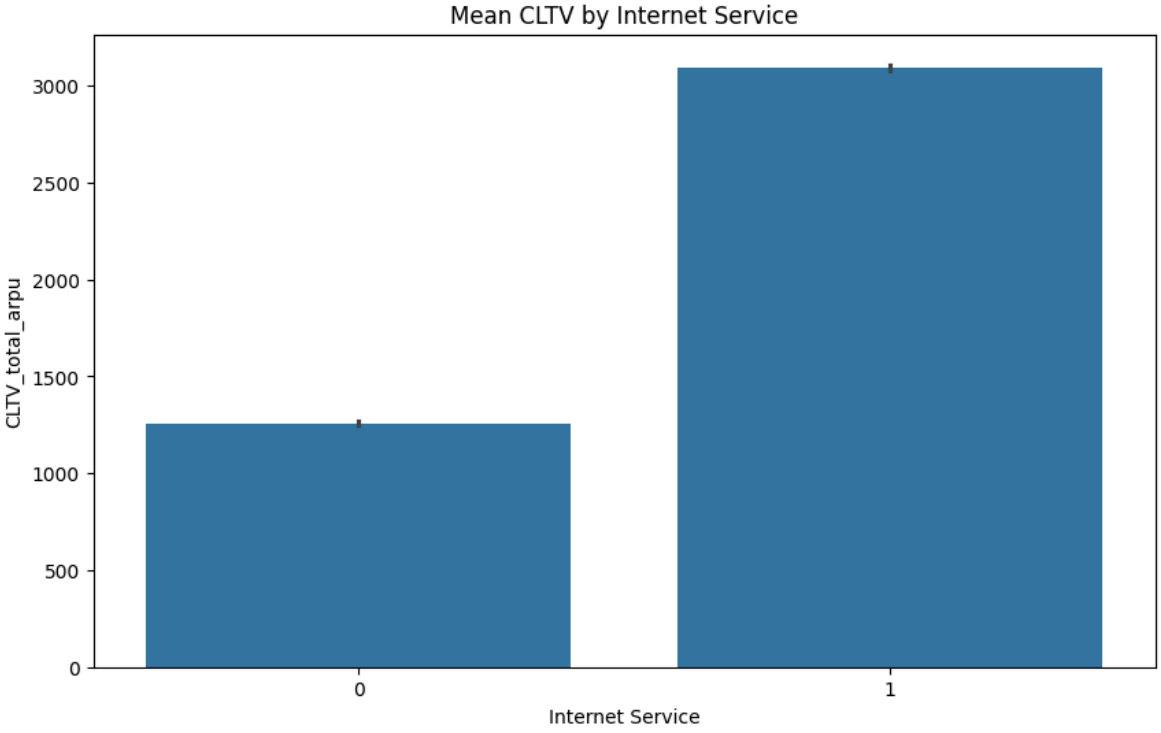
- To mitigate churn risk, I propose implementing personalized incentives tailored to individual customer preferences. This may involve offering targeted discounts or promotions on popular services, thereby fostering greater loyalty and retention among at-risk customers.
- Prioritizing exceptional customer support is essential in retaining customers. By promptly addressing their concerns and providing effective solutions, we can bolster overall satisfaction levels and reinforce long-term commitment to our services.

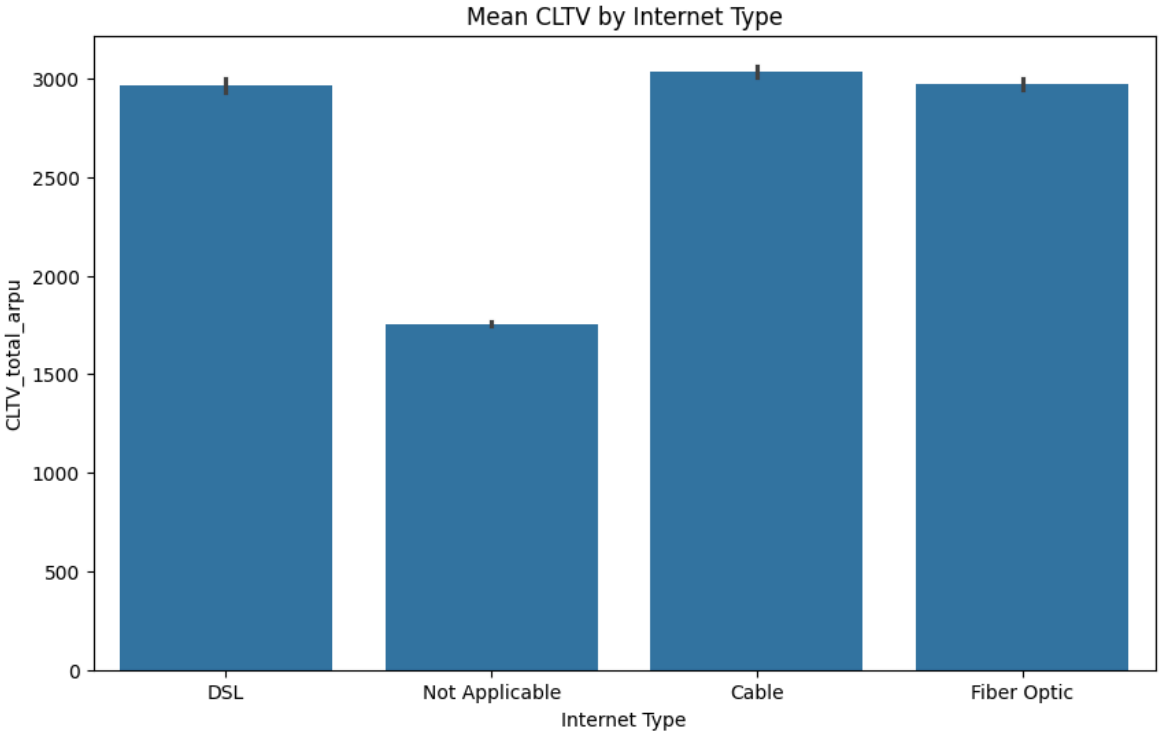
Feature Importance

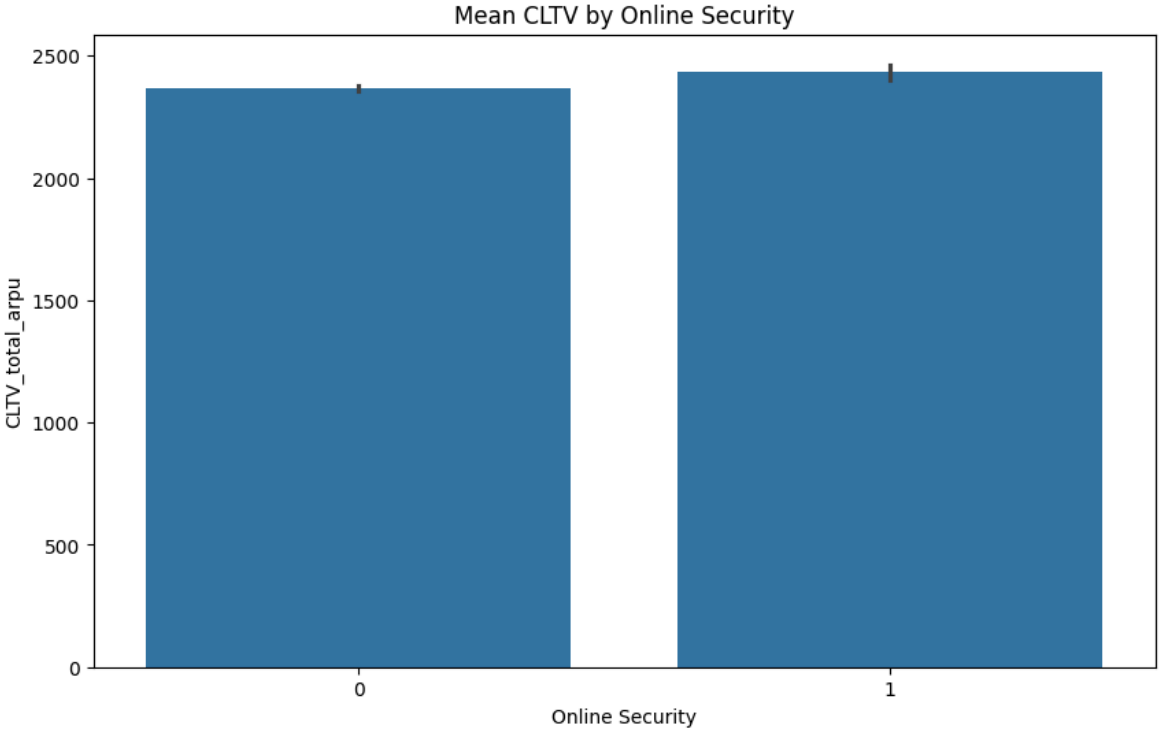
In [133…
```
# List of Categorical Services Columns
cat_cols = ['Phone Service', 'Multiple Lines',
            'Internet Service', 'Internet Type',
            'Online Security', 'Online Backup', 'Device Protection Plan',
            'Premium Tech Support', 'Streaming TV', 'Streaming Movies',
            'Streaming Music', 'Unlimited Data']

# Loop over categorical columns to plot mean CLTV by category
for col in cat_cols:
    # Create a bar plot of mean CLTV by category
    plt.figure(figsize=(10, 6))
    sns.barplot(x=col, y='CLTV_total_arpu', data=df_final)
    plt.title(f"Mean CLTV by {col}")
    plt.show()
```

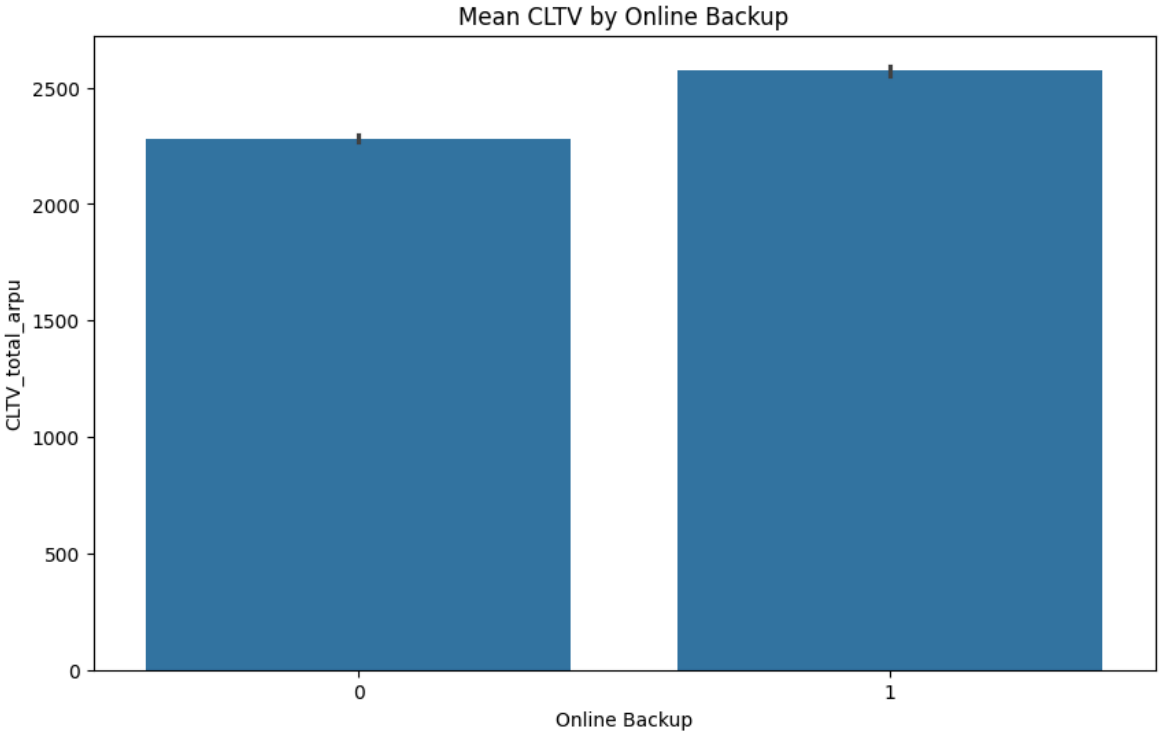

Mean CLTV by Phone Service
CLTV_total_arpu
2500
2000
1500
1000
500
0
Yes
No
Phone Service

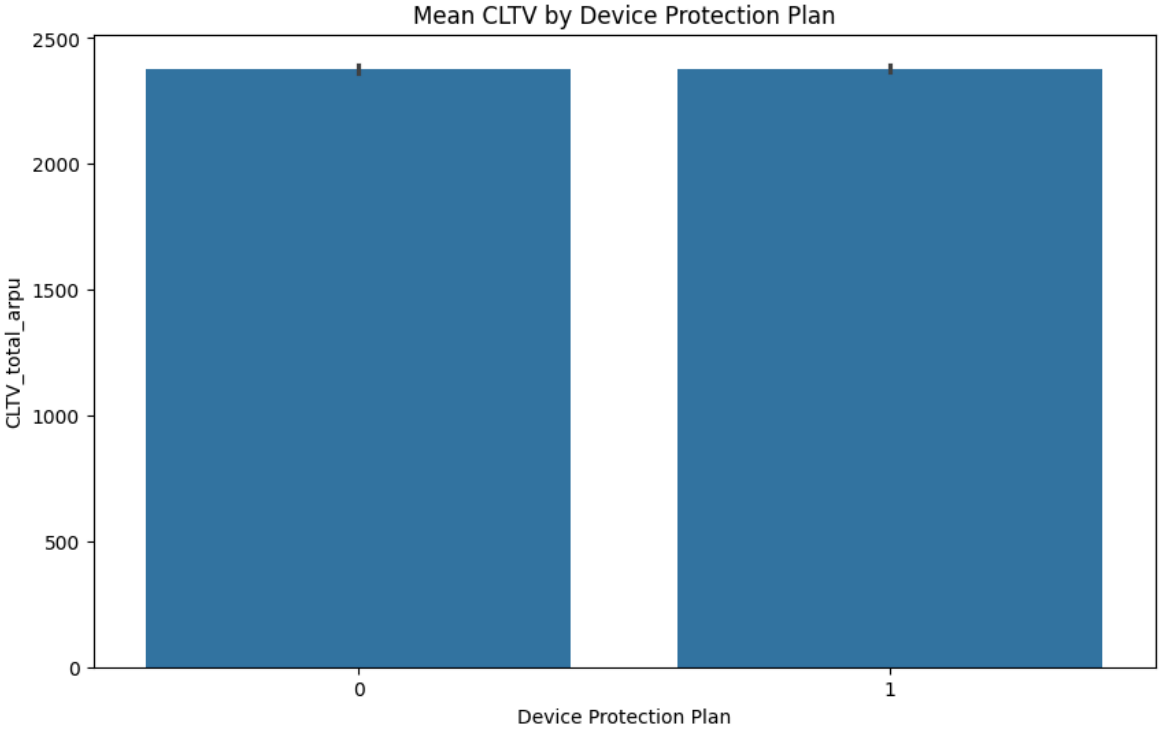

Mean CLTV by Multiple Lines
CLTV_total_arpu
2500
2000
1500
1000
500
0
Yes
No
Multiple Lines

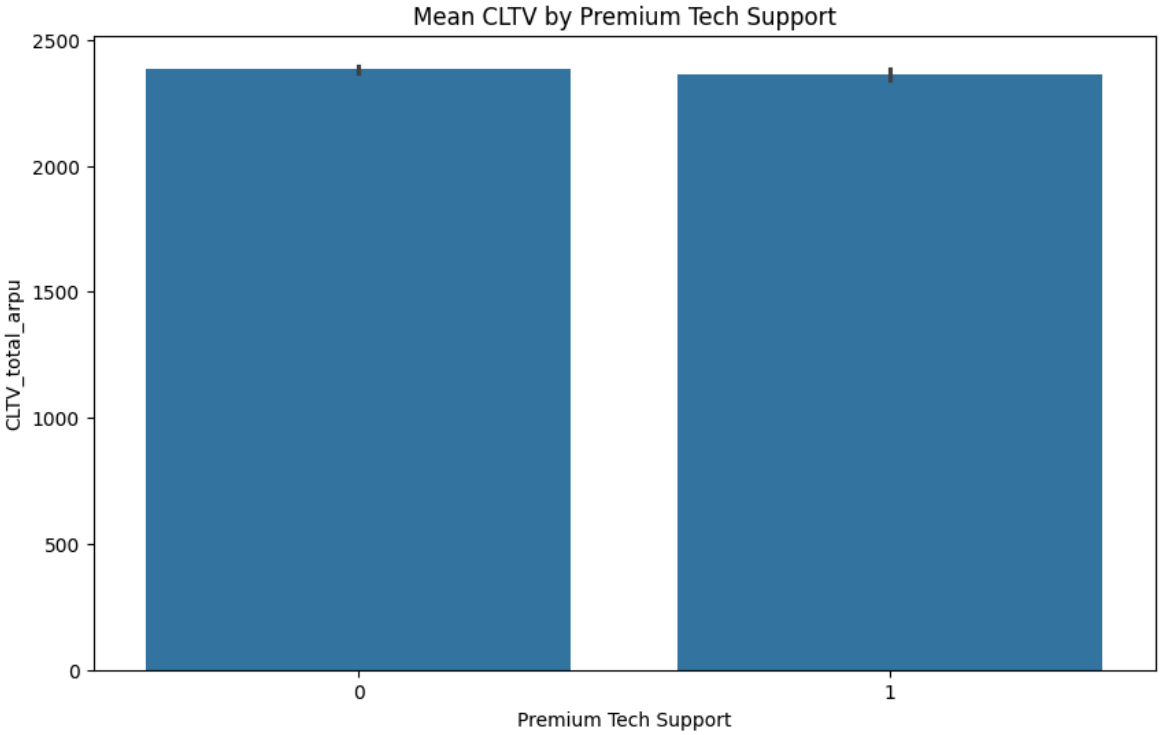

Mean CLTV by Internet Service
CLTV_total_arpu
3000
2500
2000
1500
1000
500
0
0
1
Internet Service

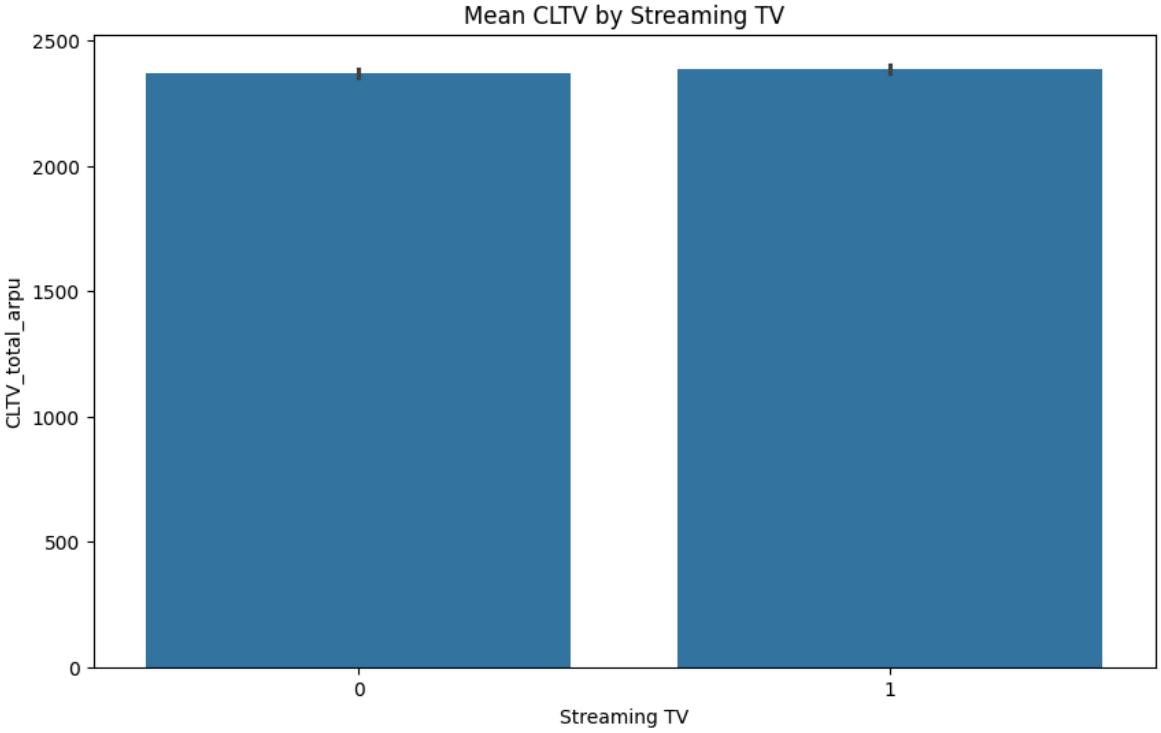

Mean CLTV by Internet Type
CLTV_total_arpu
3000
2500
2000
1500
1000
500
0
DSL
Not Applicable
Cable
Fiber Optic
Internet Type

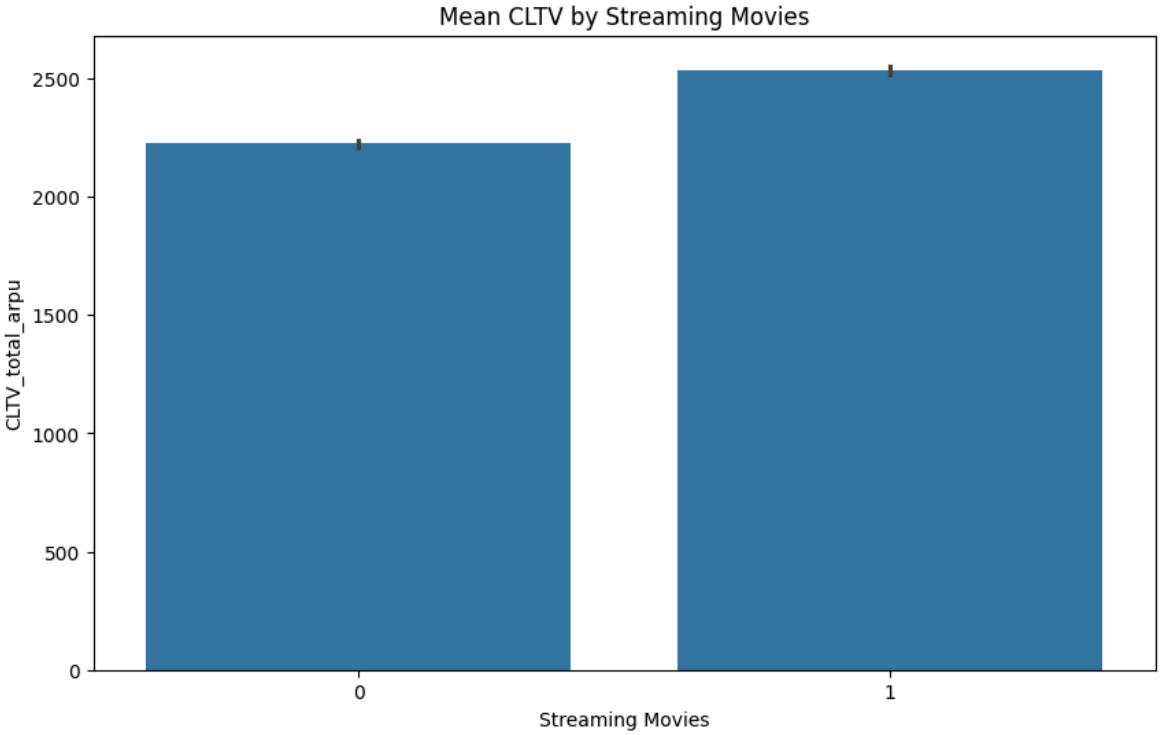

Mean CLTV by Online Security
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Online Security

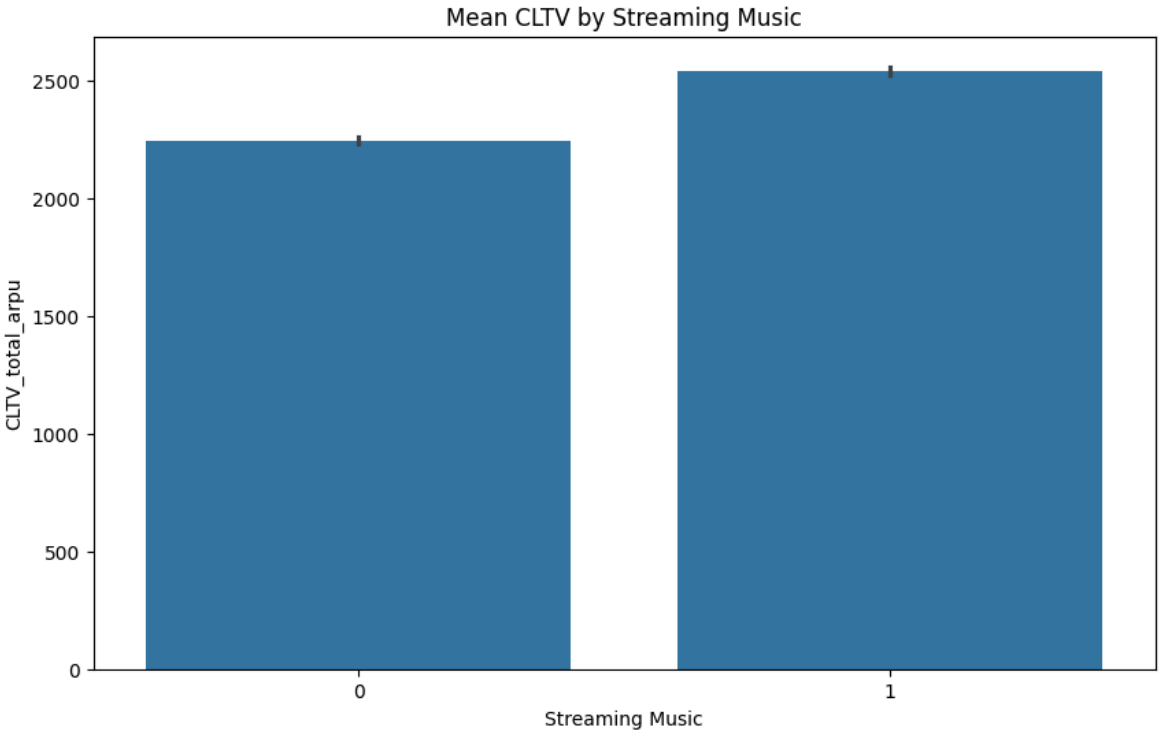

Mean CLTV by Online Backup
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Online Backup

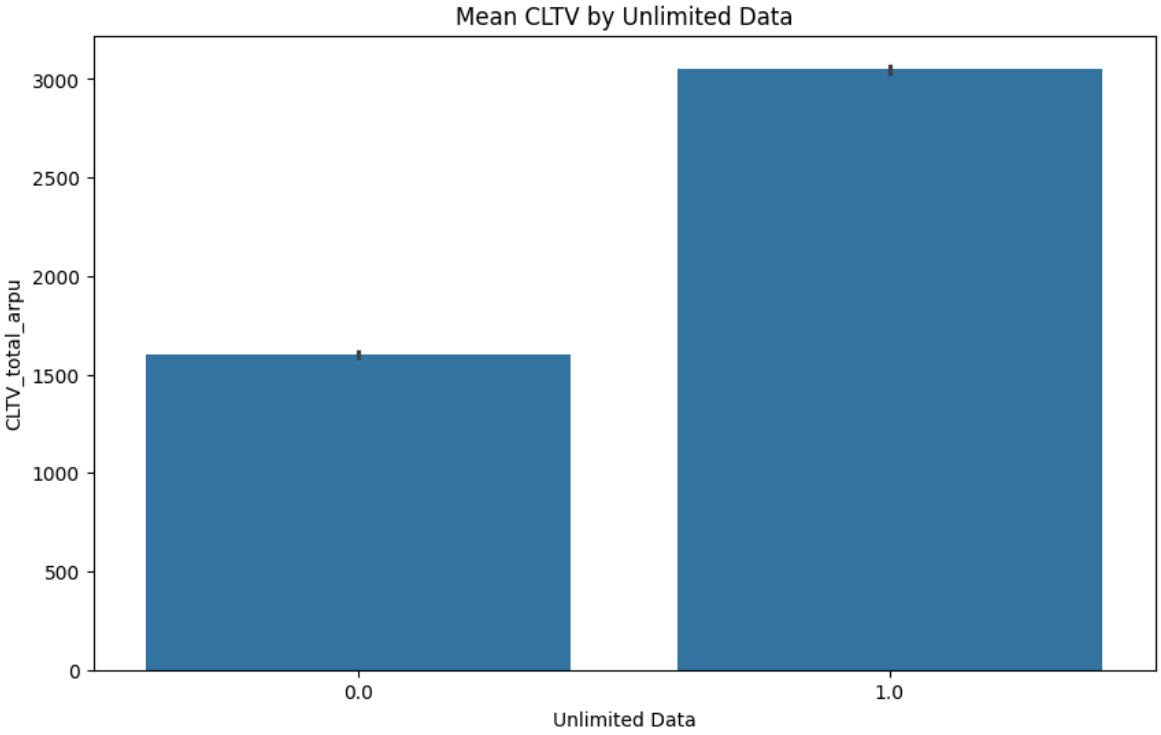

Mean CLTV by Device Protection Plan
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Device Protection Plan


Mean CLTV by Premium Tech Support
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Premium Tech Support


Mean CLTV by Streaming TV
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Streaming TV


Mean CLTV by Streaming Movies
CLTV_total_arpu
2500
2000
1500
1000
500
0
0
1
Streaming Movies





Observations and Insights:

- I've observed that customers who subscribe to online backup services tend to have higher Customer Lifetime Value (CLTV) compared to those who do not.
- Similarly, I've noticed that customers who opt for device protection plans demonstrate higher CLTV than those who do not.
- Premium tech support subscribers exhibit higher CLTV compared to non-subscribers.
- Additionally, I've found that customers with streaming TV services also show higher CLTV than those without.

- Moreover, I've seen that customers who use internet services, online backup, streaming movies and music, and unlimited data access tend to have higher CLTV.
- Furthermore, I've observed that customers who stream TV and music services also exhibit higher CLTV compared to those who do not subscribe to these services.
- The presence of streaming movies, streaming music, and unlimited data options seems to positively influence CLTV, indicating that customers who opt for these services may be more engaged and loyal.

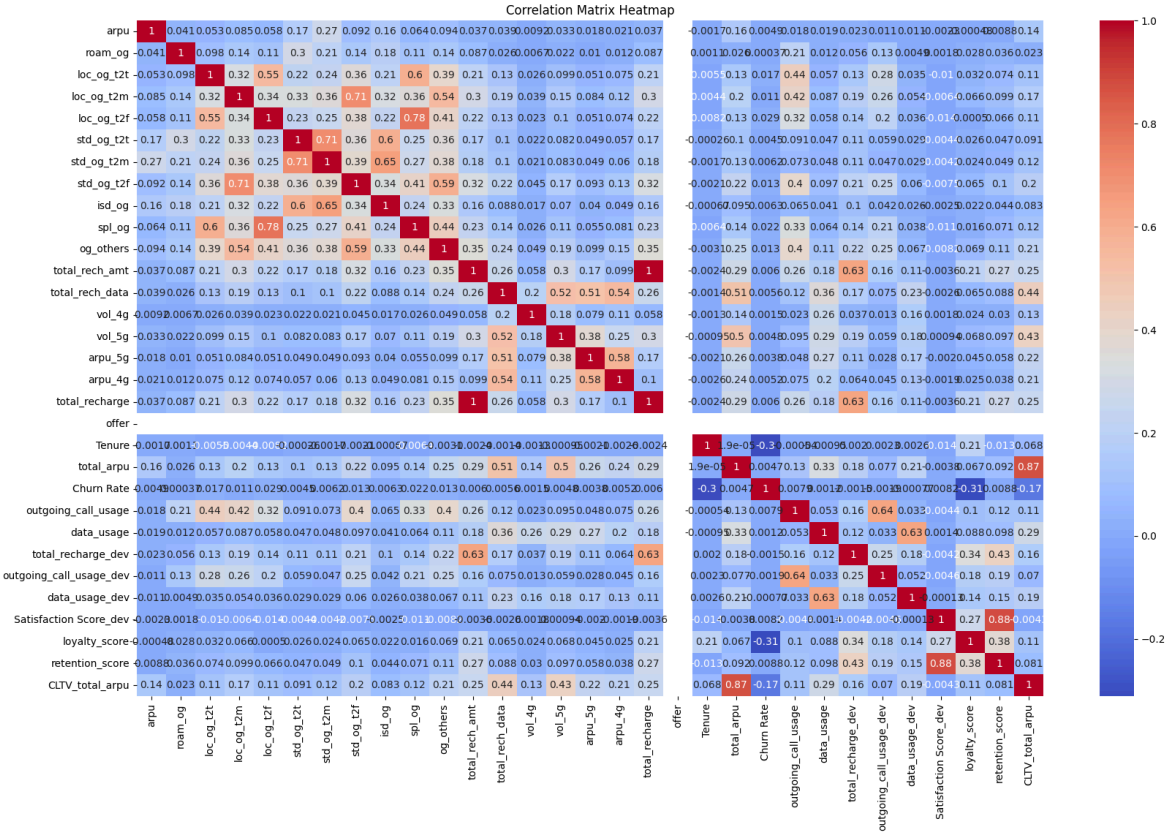
Recommendations:

1. I suggest that the company should focus on providing high-quality internet services and online security features to retain and attract more customers. Enhancing the reliability and security of internet services can contribute to higher customer satisfaction and retention.
2. I recommend considering promoting streaming movies and music services and unlimited data access to customers, as these are services that are associated with higher CLTV. Offering attractive discounts or bundling these services with existing packages can incentivize customers to opt for these features, leading to increased revenue and customer loyalty.
3. Offering personalized bundles and promotions to customers who have subscribed to multiple services is crucial, as these customers tend to have higher CLTV. By customizing offerings based on individual preferences and usage patterns, the company can enhance the value proposition for customers and strengthen their loyalty.
4. I advise analyzing the reasons why customers aren't satisfied and finding ways to improve their satisfaction with these services or encourage them to opt for more valuable services. Conducting targeted surveys, gathering feedback, and analyzing customer interactions can provide insights into areas for improvement. Implementing measures to address dissatisfaction and enhance service quality can help mitigate churn risk and increase CLTV over time.

In [134…

```
# Identify and drop non-numeric columns
numeric_features = ['arpu', 'roam_og', 'loc_og_t2t', 'loc_og_t2m',
       'loc_og_t2f', 'std_og_t2t', 'std_og_t2m', 'std_og_t2f', 'isd_og',
       'spl_og', 'og_others', 'total_rech_amt', 'total_rech_data', 'vol_4g',
       'vol_5g', 'arpu_5g', 'arpu_4g',
       'total_recharge', 'offer', 'Tenure', 'total_arpu', 'Churn Rate',
       'outgoing_call_usage', 'data_usage', 'total_recharge_dev',
       'outgoing_call_usage_dev', 'data_usage_dev', 'Satisfaction Score_dev',
       'loyalty_score', 'retention_score', 'CLTV_total_arpu']

numeric_df = df_final[numeric_features]

# Handle or remove non-numeric values
numeric_df = numeric_df.apply(pd.to_numeric, errors='coerce')

# Compute correlation matrix
```

```
corr = numeric_df.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(20,12))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix Heatmap')
plt.show()
```



Observation:

This correlation matrix heatmap provides insights into the relationship between CLTV and revenue-related columns. As I calculated CLTV from ARPU and other features that influence ARPU, this heatmap helps me understand how these variables correlate with CLTV. It shows the strength and direction of these relationships, offering valuable insights into which factors might have a significant impact on CLTV.

High Value Customer Identification

In [135…

```
# Selecting High-Value Customers
high_value = customer_df[customer_df['CLTV_total_arpu'] > customer_df['CLTV_tota
```

In [136…

```
# Calculating the Average CLTV for High-Value Customers
avg_cltv = high_value['CLTV_total_arpu'].mean()

print("The average CLTV of high-value customers is:", avg_cltv)
```

```
The average CLTV of high-value customers is: 28720.95648955717
```

In [137…

```
# Calculating Average Age for High-Value Customers
avg_age = high_value['Age'].mean()
```

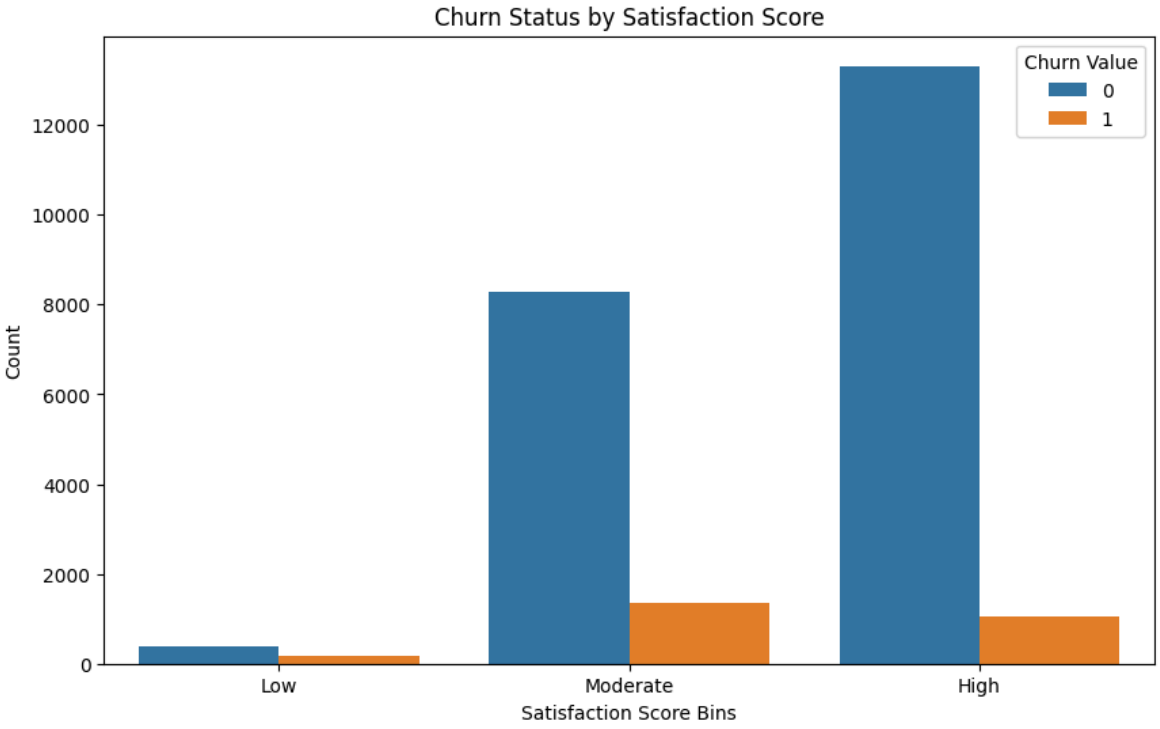
```
print("The average age of high-value customers is:", avg_age)
```

```
The average age of high-value customers is: 36.52670659903135
```

In [138…

```
# Analyzing Satisfaction Score Among High-Value Customers
plt.figure(figsize=(10,6))

# Creating bins for satisfaction score
bins = [0, 2, 3, 5]
labels = ['Low', 'Moderate', 'High']
high_value['Satisfaction Score Bins'] = pd.cut(high_value['Satisfaction Score'],

# Creating a countplot of satisfaction score bins by churn status
plt.figure(figsize=(10,6))
sns.countplot(x='Satisfaction Score Bins', hue='Churn Value', data=high_value)

# Adding labels and title
plt.xlabel('Satisfaction Score Bins')
plt.ylabel('Count')
plt.title('Churn Status by Satisfaction Score')
plt.show()
```

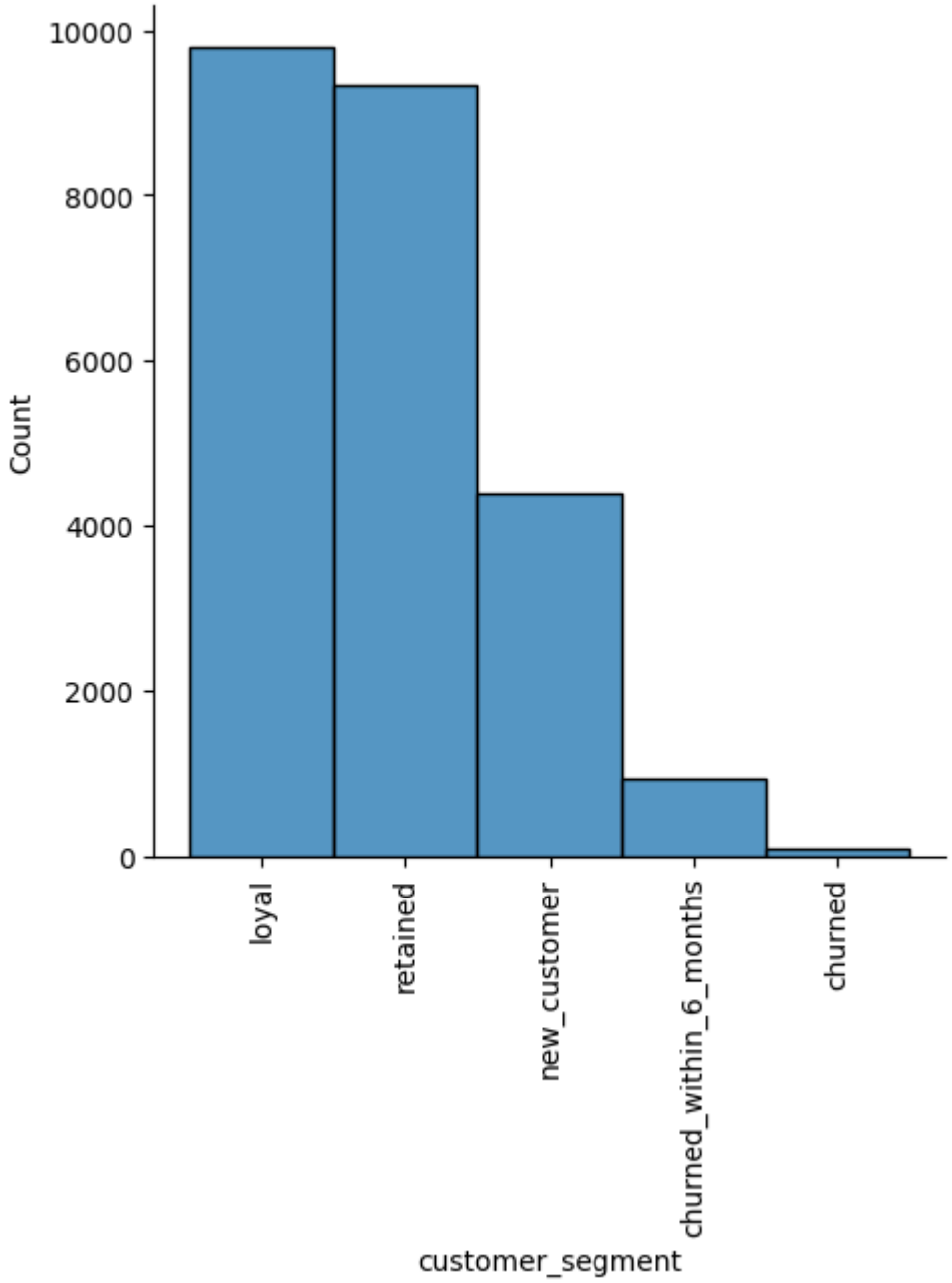
```
<Figure size 1000x600 with 0 Axes>
```



In [139…

```
# Visualizing Distribution of Customer Segments in the High CLTV Group
plt.figure(figsize=(10,6))
sns.displot(data=high_value, x='customer_segment', discrete=True)
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Observation:

In the churn status distribution by satisfaction score bins among high-value customers, it's evident that a considerable number of customers with a moderate satisfaction score have churned compared to those with low or high satisfaction scores. This indicates a potential correlation between satisfaction levels and churn rates among high-value customers.

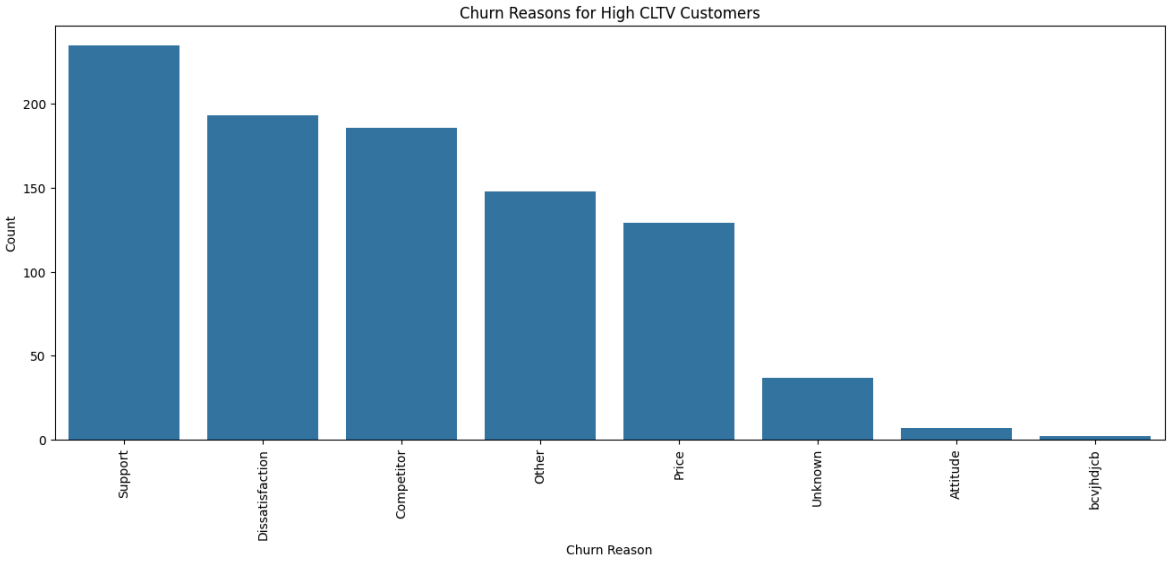
Furthermore, the distribution of customer segments within the high CLTV group shows that the majority of customers are labeled as "retained" or "loyal," indicating their long-term association with the company. However, there is a notable proportion of customers labeled as "churned" or "churned within 6 months," suggesting a churn risk among certain segments of high-value customers.

These observations underscore the importance of identifying and addressing factors contributing to customer churn among high-value segments. By understanding the specific needs and preferences of these customers, the company can implement targeted

strategies to enhance their satisfaction, loyalty, and overall retention, thereby maximizing their lifetime value to the business.

In [140…
```
# Filter high-value customers who churned within six months
high_value_churned = high_value[high_value['customer_segment'] == 'churned_withi
```

In [141…
```
# Count the number of customers for each churn reason
hv_churn_reason_counts = high_value_churned['Churn Category'].value_counts()

# Plot the churn reason counts
plt.figure(figsize=(16,6))
sns.barplot(x=hv_churn_reason_counts.index, y=hv_churn_reason_counts)
plt.title('Churn Reasons for High CLTV Customers')
plt.xlabel('Churn Reason')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```



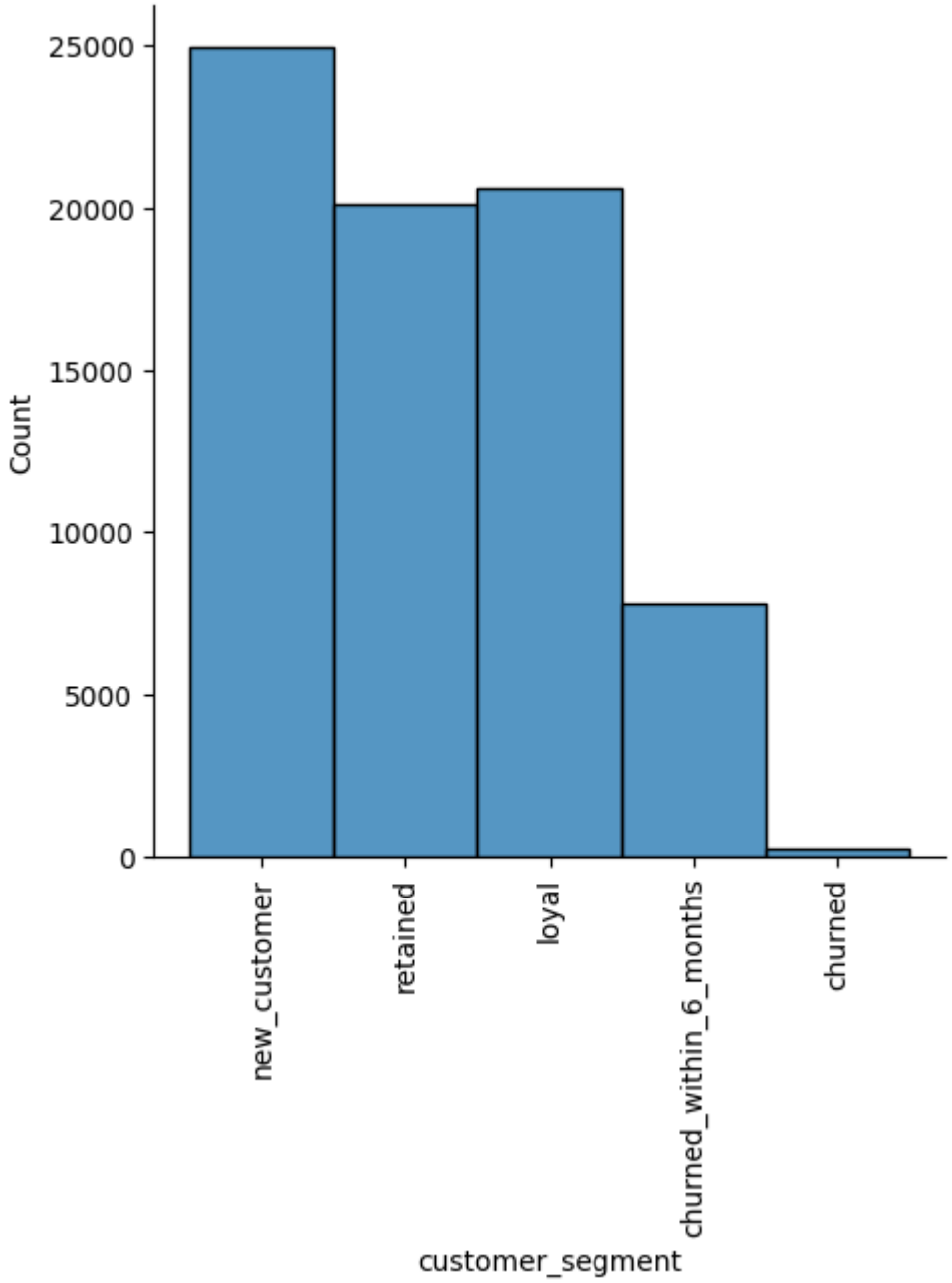
Observation

Upon analyzing the retention strategy for high-value customers who churned within 6 months, it's evident that the most prevalent reason for churn is support-related issues. This highlights potential shortcomings in the support services provided by the company, leading to customer dissatisfaction and subsequent churn. To mitigate this trend, it's imperative for the company to prioritize enhancements in its support infrastructure and service delivery, aiming to bolster customer satisfaction and retention among this valuable segment.

In [142…
```
# Selecting customers with CLTV_total_arpu above the 25th percentile
low_cltv_df = customer_df[customer_df['CLTV_total_arpu'] > customer_df['CLTV_tot
```

In [143…
```
# Plotting the distribution of customer segments in the low CLTV group
plt.figure(figsize=(10,6))
sns.displot(data=low_cltv_df, x='customer_segment', discrete=True)
plt.xticks(rotation=90)
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



Observations:

- In the low CLTV group, I observe a significant presence of loyal and retained customers. This insight suggests that we have an opportunity to capitalize on the loyalty of these customers by offering them targeted promotions and discounts. By incentivizing these loyal customers to explore additional services, we can potentially increase revenue streams and strengthen customer loyalty.
- Furthermore, I believe that conducting surveys among the low CLTV group can provide valuable insights into their specific needs and preferences. Armed with this information, we can tailor our offerings and develop customized plans and services that resonate with this segment, thereby enhancing customer satisfaction and retention.

Project Summary

In my analysis of a telecom company's customer data for calculating CLTV, I began with exploratory data analysis to grasp feature distributions and pinpoint outliers. Subsequently, I delved into ARPU analysis across varied customer segments, revealing insights into usage patterns and subscribed services.

From my findings, customers utilizing 5G networks, consuming more data, and exhibiting longer tenure and higher satisfaction scores tend to yield higher ARPU. Interestingly, churned customers demonstrated higher ARPU, indicating potential revenue benefits from longer customer retention.

Using historical usage data, I computed CLTV for each customer, highlighting top segments with the highest CLTV for targeted marketing and personalized offers. Additionally, I identified popular services among loyal customers, including internet service, unlimited data, and streaming options.

Further analysis unveiled common churn factors among customers leaving after six months, emphasizing support, dissatisfaction, and pricing concerns. Recommendations centered on enhancing support services, addressing dissatisfaction, and ensuring competitive pricing for churn reduction.

Overall, this project underscores the significance of leveraging customer data for revenue growth and retention strategies, empowering businesses with actionable insights for data-driven decision-making.

Future Scope

In future endeavors, I'll explore diverse CLTV calculation approaches tailored to specific business needs and data availability. This entails incorporating additional variables like demographics, transaction data, and customer behavior, while also considering external factors such as market trends and competition.

Expanding the project scope could involve experimenting with various CLTV computation methods and integrating more variables for a comprehensive understanding of customer value. Additionally, developing a dashboard or visualization tool for real-time CLTV tracking can empower management with actionable insights for strategic decision-making, maximizing customer retention and revenue growth.