Experiment Notebook

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

A. Project

Student Name

Shashikanth Senthil Kumar

Student Id

25218722

Experiment Id

0

business_objective

The primary business objective of this project is to develop a predictive model that accurately identifies existing customers likely to churn within the next month based on their usage patterns, payment history, and other relevant features.

These predictions will be utilized by the organization to:

* Enhance Customer Retention: By identifying at-risk customers, the organization can implement targeted marketing strategies, such as offering a 50% discount for the next three months, to encourage subscription renewals and reduce churn rates.
* Improve Customer Satisfaction: Understanding the factors contributing to churn will allow the organization to make necessary improvements in services and offerings, leading to increased customer satisfaction and loyalty.
* Optimize Revenue Management: By proactively addressing potential churn, the organization can better manage revenue streams, minimize losses, and ultimately improve financial performance.
* Inform Business Strategy: Insights gained from the model will provide valuable data to inform strategic decisions related to pricing, customer support, and service enhancements, ensuring alignment with customer needs and market trends.

By accurately predicting churn, the model aims to empower the organization to make data-driven decisions that enhance customer retention, improve overall service quality, and drive sustainable growth.

B. Experiment Description

experiment_hypothesis

There is no hypothesis to test. This is used for data exploration and analysis and required data preparation for the upcoming experiments.

experiment_expectations

The goal is to assess the baseline performance for this project and fix critical data quality issues impacting models.

C. Data Understanding

`data_understanding_executive_summary`

* Features:
The dataset contains a mix of numerical and categorical variables:
Numerical features: e.g., MonthlyCharges, TotalCharges, AccountAge, ViewingHoursPerWeek, etc.
Categorical features: e.g., SubscriptionType, PaymentMethod, Gender, ContentType, etc.

Considerations:

* Class Imbalance: The imbalance in the churn target will require careful handling during model training to ensure the model doesn't overly favor the majority class.
* Feature Importance: AccountAge shows a clear relationship with churn, suggesting that early customer engagement and retention strategies are critical. SubscriptionType also needs further investigation to assess its predictive power.
* Preprocessing Needs: The presence of missing data and the high incompleteness in certain features like Ethnicity will require preprocessing steps such as imputation or feature exclusion.

Issues Found:

* Data Quality: Missing data in key features such as Postcode, State, and especially Ethnicity could undermine model performance. Imputation strategies or feature exclusion should be carefully considered.
* Multiple Datasets: The original segmentation into multiple datasets creates complexity in data merging and may introduce inconsistencies. However, this issue has been resolved by merging the datasets.
* Class Imbalance: The churn dataset's imbalance (82.25% non-churn, 17.75% churn) could skew model predictions, requiring special techniques to address it during model training.

C.0 Import Packages

```
# Pandas for data handling
import pandas as pd

# Altair for plotting
import altair as alt

# NumPy for numerical computations
import numpy as np

# Matplotlib for basic plotting
import matplotlib.pyplot as plt

# Ensures that Matplotlib plots are displayed inline in the notebook
%matplotlib inline

# Seaborn for statistical data visualization
import seaborn as sns
```

C.1 Load Datasets

```
# Load data
# Do not change this code

df0 = pd.read_csv("subscription_data_0.csv")
df1 = pd.read_csv("subscription_data_1.csv")
df2 = pd.read_csv("subscription_data_2.csv")
df3 = pd.read_csv("subscription_data_3.csv")
df4 = pd.read_csv("subscription_data_4.csv")
df5 = pd.read_csv("subscription_data_5.csv")
df6 = pd.read_csv("subscription_data_6.csv")
df7 = pd.read_csv("subscription_data_7.csv")
df8 = pd.read_csv("subscription_data_8.csv")
df9 = pd.read_csv("subscription_data_9.csv")
```

C.2 Explore Datasets

```
# Display all columns of the DataFrame when printing
pd.set_option('display.max_columns', None)

# Display the first 5 rows of the dataset
df0.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Aurora	Bravo	88588	Wanda Mission	Underpass	Sandyhaven	
1	Ana	María	69	Laura Gateway	Drive	DPO AA 59306	
2	Susan	Lambert	81311	Hunter Fort	Cove	South Sarahshire	
3	David	Martinez	115	Whitney Drive	Valley	Danielside	
4	Miss	Lydia	674	Stuart Ford	Trace	Johnsonland	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df0.shape
```

```
# Display information about the dataset
df0.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12190 entries, 0 to 12189
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                12190 non-null  object
 1   LastName                 12187 non-null  object
 2   BuildingNumber           12190 non-null  int64
 3   StreetName               12190 non-null  object
 4   StreetType               12190 non-null  object
 5   City                     12190 non-null  object
 6   Postcode                 10896 non-null  float64
 7   State                    10896 non-null  object
 8   Ethnicity                1226 non-null   object
 9   AccountAge               12190 non-null  int64
 10  MonthlyCharges           12190 non-null  float64
 11  TotalCharges             12190 non-null  float64
 12  SubscriptionType         12190 non-null  object
 13  PaymentMethod            12190 non-null  object
 14  PaperlessBilling         12190 non-null  object
 15  ContentType              12190 non-null  object
 16  MultiDeviceAccess        12190 non-null  object
 17  DeviceRegistered         12190 non-null  object
 18  ViewingHoursPerWeek      12190 non-null  float64
 19  AverageViewingDuration   12190 non-null  float64
 20  ContentDownloadsPerMonth 12190 non-null  int64
 21  GenrePreference          12190 non-null  object
 22  UserRating               12190 non-null  float64
 23  SupportTicketsPerMonth   12190 non-null  int64
 24  Gender                   12190 non-null  object
```

> Insights:
* This Dataset has 12,190 rows and 31 columns.
* There are some Missing Datas in features like LastName, Postcode, State, and Ethnicity. In that Ethnicity is being highly incomplete (only 10% filled).
* The Prediction of churn is likely based on features like MonthlyCharges, TotalCharges, SubscriptionType, and PaymentMethod.
* Both numerical (e.g., ViewingHoursPerWeek, UserRating) and categorical features (e.g., ContentType, Gender) are present in the dataset, and require preprocessing.

```
# Display the first 5 rows of the dataset
df1.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Catherine	George	41355	Zachary Village	Well	Stephanieburgh	
1	Antonio	Rollins	79884	Thomas Road	Extensions	North Gavin	
2	Brandy	Vargas	884	Hunter Villages	Summit	Monteston	
3	Feargal	Giles	799	Julie Point	Turnpike	APO AE 57284	
4	Jillian	Martin	282	Frederick Fort	Meadow	Lake Kimberly	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df1.shape
```

```
# Display information about the dataset
df1.info()
```

> Insights:

* This Dataset has only 5 rows and 31 columns.
* There are missing values in Postcode and State.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df2.head()
```

```
# Display the shape of the dataset
df2.shape
```

```
# Display information about the dataset
df2.info()
```

> Insights:

* This Dataset has 2769 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df3.head()
```

```
# Display the shape of the dataset
df3.shape
```

```
# Display information about the dataset
df3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2793 entries, 0 to 2792
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                2793 non-null   object
 1   LastName                 2793 non-null   object
 2   BuildingNumber           2793 non-null   int64
 3   StreetName               2793 non-null   object
 4   StreetType               2793 non-null   object
 5   City                     2793 non-null   object
 6   Postcode                 2479 non-null   float64
 7   State                    2479 non-null   object
 8   Ethnicity                2777 non-null   object
 9   AccountAge               2793 non-null   int64
 10  MonthlyCharges           2793 non-null   float64
 11  TotalCharges             2793 non-null   float64
 12  SubscriptionType         2793 non-null   object
 13  PaymentMethod            2793 non-null   object
 14  PaperlessBilling         2793 non-null   object
 15  ContentType              2793 non-null   object
 16  MultiDeviceAccess        2793 non-null   object
 17  DeviceRegistered         2793 non-null   object
 18  ViewingHoursPerWeek      2793 non-null   float64
 19  AverageViewingDuration   2793 non-null   float64
 20  ContentDownloadsPerMonth 2793 non-null   int64
 21  GenrePreference          2793 non-null   object
 22  UserRating               2793 non-null   float64
 23  SupportTicketsPerMonth   2793 non-null   int64
 24  Gender                   2793 non-null   object
```

> Insights:

* This Dataset has 2793 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df4.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Anel	Estrada	4338	Julia Passage	Fords	West Maria	
1	Tiffany	Davis	61183	Huynh Valleys	Meadows	East Kyle	
2	Mark	Wolfe	557	Anderson Streets	Route	Port Veronica	
3	Olivia	Smith-Webb	88361	Beverly Park	Manors	East Matthewside	
4	Kelly	Parker	91997	Davis Oval	Trail	Port George	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df4.shape
```

```
# Display information about the dataset
df4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2794 entries, 0 to 2793
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                2794 non-null   object
 1   LastName                 2794 non-null   object
 2   BuildingNumber           2794 non-null   int64
 3   StreetName               2794 non-null   object
 4   StreetType               2794 non-null   object
 5   City                     2794 non-null   object
 6   Postcode                 2467 non-null   float64
 7   State                    2467 non-null   object
 8   Ethnicity                2784 non-null   object
 9   AccountAge               2794 non-null   int64
 10  MonthlyCharges           2794 non-null   float64
 11  TotalCharges             2794 non-null   float64
 12  SubscriptionType         2794 non-null   object
 13  PaymentMethod            2794 non-null   object
 14  PaperlessBilling         2794 non-null   object
 15  ContentType              2794 non-null   object
 16  MultiDeviceAccess        2794 non-null   object
 17  DeviceRegistered         2794 non-null   object
 18  ViewingHoursPerWeek      2794 non-null   float64
 19  AverageViewingDuration   2794 non-null   float64
 20  ContentDownloadsPerMonth 2794 non-null   int64
 21  GenrePreference          2794 non-null   object
 22  UserRating               2794 non-null   float64
 23  SupportTicketsPerMonth   2794 non-null   int64
 24  Gender                   2794 non-null   object
```

> Insights:

* This Dataset has 2794 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df5.head()
```

```
# Display the shape of the dataset
df5.shape
```

```
# Display information about the dataset
df5.info()
```

> Insights:
* This Dataset has 2816 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df6.head()
```

```
# Display the shape of the dataset
df6.shape
```

```
# Display information about the dataset
df6.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2800 entries, 0 to 2799
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                2800 non-null   object
 1   LastName                 2800 non-null   object
 2   BuildingNumber           2800 non-null   int64
 3   StreetName               2800 non-null   object
 4   StreetType               2800 non-null   object
 5   City                     2800 non-null   object
 6   Postcode                 2496 non-null   float64
 7   State                    2496 non-null   object
 8   Ethnicity                2782 non-null   object
 9   AccountAge               2800 non-null   int64
 10  MonthlyCharges           2800 non-null   float64
 11  TotalCharges             2800 non-null   float64
 12  SubscriptionType         2800 non-null   object
 13  PaymentMethod            2800 non-null   object
 14  PaperlessBilling         2800 non-null   object
 15  ContentType              2800 non-null   object
 16  MultiDeviceAccess        2800 non-null   object
 17  DeviceRegistered         2800 non-null   object
 18  ViewingHoursPerWeek      2800 non-null   float64
 19  AverageViewingDuration   2800 non-null   float64
 20  ContentDownloadsPerMonth 2800 non-null   int64
 21  GenrePreference          2800 non-null   object
 22  UserRating               2800 non-null   float64
 23  SupportTicketsPerMonth   2800 non-null   int64
 24  Gender                   2800 non-null   object
```

> Insights:
* This Dataset has 2800 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df7.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Jasmine	Howell	190	Oneal Port	Landing	Lake Matthewview	
1	Anna	Cook	57	Ryan Island	Drives	Melissaton	
2	Katrina	Chang	4639	Stephenson Mills	Shoals	APO AA 34814	
3	Mr.	Zachary	6838	John Station	Causeway	Port Tyler	
4	Heather	Young	12938	Melissa Tunnel	Circles	North Shaunview	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df7.shape
```

```
# Display information about the dataset
df7.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2785 entries, 0 to 2784
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                2785 non-null   object
 1   LastName                 2785 non-null   object
 2   BuildingNumber           2785 non-null   int64
 3   StreetName               2785 non-null   object
 4   StreetType               2785 non-null   object
 5   City                     2785 non-null   object
 6   Postcode                 2489 non-null   float64
 7   State                    2489 non-null   object
 8   Ethnicity                2779 non-null   object
 9   AccountAge               2785 non-null   int64
 10  MonthlyCharges           2785 non-null   float64
 11  TotalCharges             2785 non-null   float64
 12  SubscriptionType         2785 non-null   object
 13  PaymentMethod            2785 non-null   object
 14  PaperlessBilling         2785 non-null   object
 15  ContentType              2785 non-null   object
 16  MultiDeviceAccess        2785 non-null   object
 17  DeviceRegistered         2785 non-null   object
 18  ViewingHoursPerWeek      2785 non-null   float64
 19  AverageViewingDuration   2785 non-null   float64
 20  ContentDownloadsPerMonth 2785 non-null   int64
 21  GenrePreference          2785 non-null   object
 22  UserRating               2785 non-null   float64
 23  SupportTicketsPerMonth   2785 non-null   int64
 24  Gender                   2785 non-null   object
```

> Insights:
* This Dataset has 2785 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df8.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Jesse	Mcintyre	223	Conrad Summit	Flat	East Darlene	
1	Dr.	Todd	5249	Taylor Harbors	Islands	Lisaville	
2	Pamela	Luna	2240	John Overpass	Oval	Jessicaside	
3	Christopher	Day	966	Carla Isle	Keys	Wilsonborough	
4	Eamonn	Rea	340	Kim Summit	Stravenue	East Alexandriab...	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df8.shape
```

```
# Display information about the dataset
df8.info()
```

> Insights:
* This Dataset has 2850 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

```
# Display the first 5 rows of the dataset
df9.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Carolyn	Barnes	624	Joann Ridge	Hills	South Michelleton	
1	Lisa	Garcia	790	Joshua Unions	Walks	Lake Lauren	
2	Hugh	Harrison	254	Alex Harbor	Fort	Gordonland	
3	Rebecca	Edwards	14851	Rodriguez Dale	Village	East Nicholas	
4	Tricia	Salazar	321	Brown Valleys	Squares	South Sarahmouth	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df9.shape
```

```
# Display information about the dataset
df9.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2784 entries, 0 to 2783
Data columns (total 31 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   FirstName                 2784 non-null   object
 1   LastName                  2784 non-null   object
 2   BuildingNumber            2784 non-null   int64
 3   StreetName                2784 non-null   object
 4   StreetType                2784 non-null   object
 5   City                      2784 non-null   object
 6   Postcode                  2454 non-null   float64
 7   State                     2454 non-null   object
 8   Ethnicity                 2782 non-null   object
 9   AccountAge                2784 non-null   int64
 10  MonthlyCharges            2784 non-null   float64
 11  TotalCharges              2784 non-null   float64
 12  SubscriptionType          2784 non-null   object
 13  PaymentMethod             2784 non-null   object
 14  PaperlessBilling          2784 non-null   object
 15  ContentType               2784 non-null   object
 16  MultiDeviceAccess         2784 non-null   object
 17  DeviceRegistered          2784 non-null   object
 18  ViewingHoursPerWeek       2784 non-null   float64
 19  AverageViewingDuration    2784 non-null   float64
 20  ContentDownloadsPerMonth  2784 non-null   int64
 21  GenrePreference           2784 non-null   object
 22  UserRating                2784 non-null   float64
 23  SupportTicketsPerMonth    2784 non-null   int64
 24  Gender                    2784 non-null   object
```

> Insights:
* This Dataset has 2784 rows and 31 columns.
* There are missing values in Postcode, State and Ethnicity.
* This Dataset has the exact same columns as the df0 Dataset.

> Considerations:
* There are Multiple Datasets having different row counts (ranging from 5 to 12,190) but has the same 31 columns.
* Postcode, State, and Ethnicity columns have missing data, with Ethnicity being the most incomplete.
* Since all datasets have the same features, they can be combined into one for a unified analysis.

> Issues found:
* Having multiple datasets complicates exploratory data analysis (EDA) and may require merging or segmenting strategies.
* Columns like Postcode, State, and Ethnicity contain missing data that needs to be addressed during preprocessing.
* Some features may not be relevant for churn prediction and should be filtered out to improve model performance.

Final Dataset

```
# Combine all the datasets into one using concat method from pandas
df=pd.concat([df0,df1,df2,df3,df4,df5,df6,df7,df8,df9], ignore_index=True)
```

```
# Display all columns of the DataFrame when printing
pd.set_option('display.max_columns', None)

# Display the first 5 rows of the dataset
df.head()
```

	FirstName object	LastName object	BuildingNumber i...	StreetName object	StreetType object	City object	Postcc
0	Aurora	Bravo	88588	Wanda Mission	Underpass	Sandyhaven	
1	Ana	María	69	Laura Gateway	Drive	DPO AA 59306	
2	Susan	Lambert	81311	Hunter Fort	Cove	South Sarahshire	
3	David	Martinez	115	Whitney Drive	Valley	Danielside	
4	Miss	Lydia	674	Stuart Ford	Trace	Johnsonland	

5 rows, 31 cols 10 / page Page 1 of 1

```
# Display the shape of the dataset
df.shape
```

```
# Display information about the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 34586 entries, 0 to 34585
Data columns (total 31 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   FirstName                34586 non-null  object
 1   LastName                 34583 non-null  object
 2   BuildingNumber           34586 non-null  int64
 3   StreetName               34586 non-null  object
 4   StreetType               34586 non-null  object
 5   City                     34586 non-null  object
 6   Postcode                 30802 non-null  float64
 7   State                    30802 non-null  object
 8   Ethnicity                23536 non-null  object
 9   AccountAge               34586 non-null  int64
 10  MonthlyCharges           34586 non-null  float64
 11  TotalCharges             34586 non-null  float64
 12  SubscriptionType         34586 non-null  object
 13  PaymentMethod            34586 non-null  object
 14  PaperlessBilling         34586 non-null  object
 15  ContentType              34586 non-null  object
 16  MultiDeviceAccess        34586 non-null  object
 17  DeviceRegistered         34586 non-null  object
 18  ViewingHoursPerWeek      34586 non-null  float64
 19  AverageViewingDuration   34586 non-null  float64
 20  ContentDownloadsPerMonth 34586 non-null  int64
 21  GenrePreference          34586 non-null  object
 22  UserRating               34586 non-null  float64
 23  SupportTicketsPerMonth   34586 non-null  int64
 24  Gender                   34586 non-null  object
```

```
# Display the summary statistics
df.describe()
```

	BuildingNumber f...	Postcode float64	AccountAge floa...	MonthlyCharges f..	TotalCharges flo...	ViewingHoursPer...	Averag
cou...	34586	30802	34586	34586	34586	34586	
me...	18382.10738	50439.6235	60.03420459	12.4815939	749.1264389	20.575998	9
std	27820.82529	28715.40069	34.35697528	4.303705995	523.0145786	11.23377493	5
min	0	501	1	4.990061547	5.101683897	1.000133293	5
25%	670	25490.25	30	8.744130977	330.7325304	10.8761316	4
50%	4495	50572	60	12.47248126	644.6101834	20.66695996	9
75%	24926.75	75218.5	90	16.17456994	1087.452915	30.30229059	1
max	99999	99947	119	19.98941153	2373.70029	39.9985596	1

8 rows, 12 cols 10 / page Page 1 of 1

```
# Display the summary statistics for object type
df.describe(include='object')
```

> Insights:

* All the datasets have been combined into a single dataset named df.
* The final dataset has 34586 rows and 31 columns.
* LastName, Postcode, State, and Ethnicity columns have missing data, with Ethnicity being the most incomplete.

```
# Set the name of the target column
target_name = 'Churn'
```

```
# Display summary statistics of the target column (Churn) in dataset
df[target_name].describe()
```

```
# Display the count of churns and non-churns in the target
df[target_name].value_counts()
```

```
# Pie chart for Churn proportions
churn_counts = df[target_name].value_counts()
plt.pie(churn_counts, labels=['No Churn', 'Churn'], autopct='%1.2f%%', startangle=90, colors=['lightblue', 'c
plt.title('Proportion of Customers that Churn vs. Stay')
```

```
# Bar Plot for Churn vs. Non-Churn counts
sns.countplot(x='Churn', data=df)
plt.title('Churn vs. Non-Churn Count')
```

> Insights:
* The dataset contains 34,586 records with a binary target variable, Churn.
* Approximately 17.75% of the customers have churned (mean of the target variable), while 82.25% have not churned, indicating a class imbalance.
* The majority class in the dataset is non-churned customers (0), with 28,446 records.
* There are 6,140 churned customers (1), representing the minority class.
* The class imbalance is evident from both the pie chart and bar plot, which show that the majority of customers did not churn.

> Considerations:
* The imbalance in the target variable may require techniques such as resampling (oversampling the minority class or undersampling the majority class), or using class-weighted models to avoid bias during model training.
* For classification evaluation, metrics like Recall, F1-score, or Precision might be more appropriate than accuracy due to the imbalance in churn vs. non-churn classes.
* It's important to assess the impact of the class imbalance on the model performance and predictions, particularly for churn prediction tasks where false negatives (missed churns) could be costly.

> Issues found:
* Class imbalance is notable, with the non-churn class significantly outnumbering the churn class, which may lead to biased model predictions toward the majority class (non-churn).
* The dataset summary doesn't indicate missing values in the target column, but further preprocessing checks should be performed to ensure no missing values are present in the other features.

```
# Display the summary Statistics
df['AccountAge'].describe()
```

> Insights:
* The mean of AccountAge is around 60 months which is approximately 5 years and the Standard Deviation of AccountAge is around 34 months.
* The minimum AccountAge is 1 month and the maximum is 119 months.
* The 25th percentile (25%) is 30 months (approximately 2.5 years), and the 75th percentile (75%) is 90 months (approximately 7.5 years), showing that most customers have been with the service between 2.5 and 7.5 years.

```
# Set the plot size for better visibility
plt.figure(figsize=(10, 6))

# A bar plot created for AccountAge distribution using Seaborn
sns.histplot(df['AccountAge'], bins=30, kde=True)

# Set the title and labels
plt.title('AccountAge Distribution in the Dataset', fontsize=16)
plt.xlabel('AccountAge', fontsize=14)
plt.ylabel('Count', fontsize=14)
```

```
# Set the plot size for better visibility
plt.figure(figsize=(10, 6))

# A bar plot created for AccountAge distribution by Churn status using Seaborn
sns.histplot(data=df, x='AccountAge', hue='Churn', bins=30, kde=False)

# Set the title and labels
plt.title('AccountAge Distribution by Churn Status', fontsize=16)
plt.xlabel('AccountAge', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.legend(['Churned', 'Not Churned'])
```

```
# A box plot created for AccountAge vs Churn using Seaborn
sns.boxplot(x='Churn', y='AccountAge', data=df)

# Set the title and labels
plt.title('AccountAge  vs. Churn',fontsize=16)
plt.xlabel('Churn', fontsize=14)
plt.ylabel('AccountAge', fontsize=14)
```

> Insights:

* The customers who churn tend to have shorter AccountAge and the churn significantly decreases as the AccountAge increases. This suggests that focusing on improving customer experience and engagement in the first few years is crucial to reducing churn.
* Companies could focus retention efforts on customers with an account age below 50 months, offering incentives or targeted marketing to improve their retention rates.
* Customers with longer account ages (over 60 months) are less likely to churn, indicating strong loyalty. However, this group should still be monitored for changes in behavior as churn risk, while lower, still exists.

```
# Display the average value of accountage based on churn status
churned_group = df.groupby('Churn')['AccountAge'].mean()
print(churned_group)
```

> Insights:
* The average AccountAge for non-churned customers is around 63 months, while churned customers have a significantly lower average of around 46 months.
* This supports the earlier observation that newer customers tend to churn more frequently than those who have been with the service longer.

> Considerations:
* Given that customers with shorter account ages (below 50 months) are more likely to churn, it is important to develop targeted retention strategies, such as personalized offers, discounts, or loyalty programs aimed at newer customers.
* Although customers with longer account ages (over 60 months) are less likely to churn, it is still important to monitor these customers. Offering them enhanced services or rewards for loyalty can help maintain high retention rates.
* The analysis of AccountAge suggests the need for implementing tiered retention programs based on how long a customer has been with the service. Understanding when customers are most likely to churn can help refine marketing and customer experience initiatives, leading to more targeted efforts at different stages in the customer lifecycle.
* The company should consider proactive engagement campaigns for customers with an account age of 1 to 50 months. These campaigns should focus on addressing any service issues, improving user experience, and highlighting product features that add value.

> Issues found:
* There is a notable imbalance between the AccountAge distribution for churned and non-churned customers. This might suggest that account age alone is a strong predictor of churn, potentially overshadowing other features that could be important in understanding churn behavior.
* The AccountAge feature alone may not fully explain why customers churn. Other factors, such as customer satisfaction, usage patterns, or external factors (e.g., market trends), might also play a role in churn. Thus, focusing solely on AccountAge could limit the effectiveness of churn prediction models.
* While AccountAge is a useful indicator for churn, it does not provide insights into the reasons behind churn behavior. Combining this analysis with other variables (e.g., customer feedback, usage data) could improve the model's explanatory power.
* It would be helpful to analyze whether customers with very short AccountAge (e.g., 1-12 months) were properly onboarded or whether they had negative experiences that led them to churn early. This could reveal process issues or gaps in customer engagement strategies.

```
# Display the summary statistics
df['SubscriptionType'].describe()
```

```
# Display the unique values and its count
df['SubscriptionType'].value_counts()
```

> Insights:
* There are three types of Subscriptions, they are Basic, Standard, Premium.
* The most frequent subscription type is "Standard," with 11,674 customers (approximately 33.8%) choosing this option.
* This suggests that the "Standard" plan is likely positioned as the most attractive option in terms of price and features, making it a popular choice among customers.

```
# A histogram plot is created for Subscription type
sns.countplot(x='SubscriptionType', data=df)

# Set the title and labels
plt.title('Distribution of Subscription Types')
plt.xlabel('Subscription Type')
plt.ylabel('Count')
```

```
# A piechart is created for the subscription feature
df['SubscriptionType'].value_counts().plot.pie(autopct='%1.1f%%', startangle=90)

# Set the title and labels
plt.title('Proportion of Subscription Types')
plt.ylabel('')
```

> Insights:
* The relatively even distribution between the Standard (33.8%), Premium (32.8%), and Basic (33.5%) subscription types indicates a balanced customer preference across all subscription tiers.
* This suggests that customers' needs vary widely, with some opting for more affordable options while others are willing to pay for premium features.
* The even split also provides an opportunity for the company to focus on tailored marketing strategies for each segment without any one type overwhelmingly dominating the customer base.

```
# A box plot is created for Subscription type vs Account age based on Churn status
sns.boxplot(x='SubscriptionType', y='AccountAge', hue='Churn', data=df, width=0.75)  # Adjust 'width' for gap

# Set the title and labels
plt.title('Subscription Type vs AccountAge by Churn Status', fontsize=16)
plt.xlabel('Subscription Type', fontsize=14)
plt.ylabel('Account Age', fontsize=14)

# Place legend outside the plot
plt.legend(title='Churn Status', bbox_to_anchor=(1.05, 1), loc='upper left')  # Adjust legend position
```

> Insights:
* Across all subscription types (Premium, Basic, Standard), customers with shorter account ages are more likely to churn, while longer-tenured customers show lower churn rates.
* Churned customers typically have a median AccountAge of around 40-50 months, while non-churners have a median around 70-80 months.
* The pattern is consistent across all subscription types, suggesting that AccountAge is a stronger predictor of churn than subscription type.

> Considerations:
* The distribution among subscription types (Premium, Basic, Standard) is fairly even, with each making up roughly one-third of the dataset. This balance allows for more reliable comparisons between categories.
* While AccountAge is the more significant factor in predicting churn, the impact of subscription type could still vary based on service features, pricing, or user experience within each tier. Further analysis could explore specific reasons for churn in each subscription tier.
* Given that churn is more common among newer customers in all subscription tiers, retention efforts should focus on providing early incentives, regardless of the subscription type. Premium customers, in particular, may require tailored loyalty programs due to the higher cost associated with their plan.

> Issues found:
* The three broad subscription categories may oversimplify user experiences. Further segmentation within each subscription type, such as by usage patterns or service satisfaction, might yield more actionable insights.
* The current analysis doesn't factor in additional influencing variables like user engagement metrics, service upgrades/downgrades, or customer support interactions, which could also affect churn behavior.
* Subscription type alone doesn’t appear to be a strong independent predictor of churn when compared to AccountAge. There could be overlapping factors, meaning more sophisticated models may be required to isolate the exact contribution of subscription type to churn behavior.

C.4.c Feature "MonthlyCharges"

```
# Display the summary statistics
df['MonthlyCharges'].describe()
```

>Insights:
* Monthly charges has Mean of $12.48 with a median of $12.47, indicating a symmetrical distribution.
* Its Standard deviation of $4.30, showing moderate variation in charges.
* Charges range from $4.99 to $19.99, reflecting diverse pricing plans.

```
# plot a Box plot for Monthly Charges
plt.figure(figsize=(8, 5))
sns.boxplot(x=df['MonthlyCharges'])
plt.title('Box Plot of Monthly Charges')
plt.xlabel('Monthly Charges')
```

```
# a bar plot for Distribution of Monthly Charges by Churn Status
sns.histplot(data=df,x='MonthlyCharges',kde=True,hue='Churn')
plt.title('Distribution of Monthly Charges by Churn Status')
plt.xlabel('Monthly Charges')
plt.legend(['Churned', 'Not Churned'])
```

```
# Plot Average Monthly Charges by Churn Status
plt.figure(figsize=(10, 6))
sns.barplot(x='Churn', y='MonthlyCharges', data=df, estimator='mean')
plt.title('Average Monthly Charges by Churn Status')
plt.ylabel('Average Monthly Charges')
```

>Insights:
* The box plot shows that the majority of MonthlyCharges are concentrated between 8.74 and 16.17 units, with a median of around 12.47.
* Churn distribution shows that customers with higher MonthlyCharges tend to churn more often, suggesting a correlation between higher charges and the likelihood of churn.
* The bar plot clearly indicates that customers who churn have a higher average monthly charge compared to those who do not churn.

>Considerations:
* MonthlyCharges is symmetrically distributed, with the mean ($12.48) and median ($12.47) closely aligned, suggesting balance in customer charges.
* Moderate variation in charges (standard deviation $4.30) likely arises from different SubscriptionType, PaymentMethod, or usage patterns (ContentType, MultiDeviceAccess).
* Charges range from $4.99 to $19.99, reflecting diverse packages, possibly linked to AccountAge and content consumption (ContentDownloadsPerMonth).
* No outliers in the boxplot, showing consistent billing. Further analysis is needed for features like ViewingHoursPerWeek and DeviceRegistered.
* Higher MonthlyCharges correlate with increased churn, suggesting that customers with higher charges may expect more value or are more sensitive to dissatisfaction.

>Issues Found:
* No outliers are detected in MonthlyCharges, but it is important to explore whether extreme values exist in related variables like TotalCharges or SupportTicketsPerMonth that may contribute to churn.
* Higher charges correlate with increased churn rates, but further exploration into factors like ContentDownloadsPerMonth, GenrePreference, and WatchlistSize is necessary to understand the reasons why higher-paying customers leave.
* While higher-paying customers churn more frequently, analyzing features such as Cohort, PaperlessBilling, and PaymentMethod could provide insights into specific user groups or behaviors that lead to higher churn risks.

feature_selection_executive_summary

The feature selection process for this dataset followed a structured approach to enhance model performance and ensure the relevance of features in predicting churn. The focus was on improving model performance, avoiding redundancy, and protecting customer privacy by removing unnecessary features.
1. Removal of Personal Identifiable Information (PII)
* To safeguard customer privacy, personal details such as names, addresses, and other identifiable information were removed from the dataset.
* This was essential as these features do not contribute to predicting churn and may introduce bias. Features like 'FirstName', 'LastName', 'StreetName', 'Postcode', 'Ethnicity', 'Gender', and 'CustomerID' were dropped.
2. Correlation Analysis
* Correlation analysis was conducted to assess the relationship between numerical features and churn, helping to identify impactful features and detect multicollinearity.
* Modest Positive Correlation: 'MonthlyCharges' (0.0996) and 'SupportTicketsPerMonth' (0.0854).
* Negative Correlation: 'AccountAge' (-0.1927) and 'TotalCharges' (-0.1149).
* Multicollinearity between 'TotalCharges' and 'MonthlyCharges' suggested the need to consider dropping one of these features.
3. Final Selection of Features
After removing irrelevant features and performing correlation analysis, the following features were finalized for modeling:
* Numerical: 'AccountAge', 'MonthlyCharges', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'SupportTicketsPerMonth'
* Categorical: 'SubscriptionType', 'PaymentMethod', 'ContentType', 'MultiDeviceAccess', 'Cohort'
* Other relevant customer behavior-related features like 'ContentDownloadsPerMonth', 'WatchlistSize', and 'ParentalControl' were retained.
This comprehensive selection ensures that the final dataset is optimized for predicting churn, free from unnecessary or redundant features, and complies with privacy standards.

> Rationale:

* To protect customer privacy and reduce irrelevant features, personal details such as names, addresses, and unique identifiers should be removed.
* These features do not contribute to predicting churn and can introduce bias or privacy risks.
* The Cohort feature is also removed due to its improper format, lack of grouping, and no clear order.

```
# Display columns of the dataset
df.columns
```

```
# Drop personal identifiable information (PII) from the dataset
features_to_drop = ['FirstName', 'LastName', 'StreetName', 'StreetType','BuildingNumber', 'City', 'Postcode',
                    'State', 'Ethnicity','Gender','CustomerID', 'Cohort']

# Drop the specified features from all datasets
df_featured = df.drop(columns=features_to_drop)
```

> Results:

* The dataset is successfully cleaned by removing unnecessary personal details and the Cohort feature.
* This ensures that only relevant features contributing to model performance remain, reducing the risk of privacy issues or bias while simplifying the data.

> Rationale:
Correlation analysis is used to assess the relationships between numerical features and the target variable (Churn). Highly correlated features (both positively and negatively) are considered more impactful for prediction. It also helps detect multicollinearity among features.

```
# Select only numeric columns from the DataFrame
numeric_df = df_featured.select_dtypes(include=['float64', 'int64'])

# Compute correlation matrix for numeric data
correlation_matrix = numeric_df.corr()
```

```
# Visualize correlation with heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
```

```
# Sort the features in descending order based on Correlation with Churn
correlation_with_target = correlation_matrix['Churn'].sort_values(ascending=False)
correlation_with_target
```

> Results:
MonthlyCharges (0.0996) and SupportTicketsPerMonth (0.0854) have a modest positive correlation with Churn, while AccountAge (-0.1927) and TotalCharges (-0.1149) show negative correlations. Multicollinearity is identified between TotalCharges and MonthlyCharges, suggesting one may be dropped.

D.3 Final Selection of Features

```
# The final selected features are

features_list = ['AccountAge', 'MonthlyCharges', 'SubscriptionType','PaymentMethod', 'PaperlessBilling', 'Cor
       'DeviceRegistered', 'ViewingHoursPerWeek', 'AverageViewingDuration', 'ContentDownloadsPerMonth', 'Genr
       'SupportTicketsPerMonth', 'WatchlistSize', 'ParentalControl', 'SubtitlesEnabled', 'Churn']
```

E. Data Cleaning

`data_cleaning_executive_summary`

The data cleaning process focused on ensuring the dataset is complete, consistent, and reliable for further analysis and modeling. The steps included checking for missing values, removing duplicate entries, and standardizing categorical text data to lowercase.
1. Copying Datasets:
* Datasets were copied to ensure original data remained unchanged during cleaning.
2. Fixing Missing Values:
* Rationale: Missing values can lead to inaccuracies in data analysis and model predictions.
* Results: No missing values were found in the dataset, confirming data completeness.
3. Fixing Duplicate Values:
* Rationale: Duplicate values can skew the analysis by over-representing certain data points.
* Results: No duplicate entries were found, maintaining data integrity.
4. Fixing Data Standards:
* Rationale: Inconsistent data formats, especially in categorical variables like strings (e.g., 'Yes' vs. 'yes'), can affect the performance of machine learning algorithms and skew analytical results.
* Results: All text-based categorical data were standardized to lowercase, ensuring consistency and improving model performance potential.
This data cleaning process ensures that the dataset is prepared for high-quality analysis and accurate model development.

Copying the Dataset

```
#Copy the datasets
df_cleaned = df[features_list].copy()
```

E.1 Fixing Missing Values

> Rationale:
* Missing values can lead to inaccuracies in data analysis and model predictions.
* To ensure a complete dataset, missing values need to be handled either by imputation (filling with mean, median, or mode) or by removing the rows/columns with significant amounts of missing data.
* This helps improve data quality and ensures that model performance is not biased due to missing information.

```
# Checking for missing values
df_cleaned.isna().sum()
```

> Results:
The dataset has no missing data, ensuring completeness and reliability for further analysis.

E.2 Fixing Duplicate Values

> Rationale:
* Duplicate values can skew the analysis by over-representing certain data points.
* Removing duplicates ensures that each observation is unique, providing an accurate and balanced dataset for analysis or model training.

```
# Checking for duplicate values
df_cleaned.duplicated().sum()
```

> Results:
No duplicate values were found in the dataset, ensuring data integrity and preventing biased results in model development.

E.3 Fixing Data Standard

> Rationale:
* Inconsistent data formats, especially in categorical variables like strings (e.g., 'Yes' vs. 'yes'), can affect the performance of machine learning algorithms and skew analytical results.
* Standardizing text data (such as converting all text to lowercase) ensures consistency, improving the quality of data for analysis and model training.

```
# Standardize text data to lowercase
for i in df_cleaned.columns:
    if df_cleaned[i].dtype == 'object':
        df_cleaned[i] = df_cleaned[i].str.lower()
```

> Results:
All text-based categorical data were standardized to lowercase, ensuring uniformity across the dataset. This improves the model's ability to interpret and process categorical features correctly.

```
```

feature_engineering_executive_summary

* Feature engineering in this section focused on creating new interaction terms and categorizing continuous variables to enhance model performance in predicting customer churn.
* Specifically, interaction terms were introduced to capture the combined effects of key features, such as support engagement, user satisfaction, and viewing behavior, which might jointly influence churn but may not be evident when analyzed independently.
* Additionally, categorical transformations, such as creating tiers for MonthlyCharges, simplify the understanding of price sensitivity and its effect on churn.
* These engineered features, added to the dataset, are designed to improve the model's predictive power by leveraging combined and categorized factors that contribute to customer churn.

Copy Dataset

```
#Copy the datasets
df_eng = df_cleaned.copy()
```

```
```

> Rationale:
* Price sensitivity can significantly impact customer churn.
* By categorizing MonthlyCharges into tiers like "Low", "Medium", and "High", we can analyze the effect of pricing on customer retention.

```
# Create MonthlyChargeTier by binning MonthlyCharges
df_eng['MonthlyChargeTier'] = pd.cut(df_eng['MonthlyCharges'], bins=[0, 4, 10, 15, np.inf], labels=["Very Lo
```

> Results:
* A new categorical feature, MonthlyChargeTier, was created to classify customers based on their monthly charges.
* This will help in understanding if high or low-paying customers churn more frequently.

> Rationale:
* Combining UserRating and SupportTicketsPerMonth can help assess if lower ratings combined with frequent support tickets indicate dissatisfaction and a higher likelihood of churn.
* Customers who frequently raise support tickets and give poor ratings might be at a higher churn risk.

```
# Create interaction term between SupportTicketsPerMonth and UserRating
df_eng['UserRating_SupportTicketsInteraction'] = df_eng['UserRating'] * df_eng['SupportTicketsPerMonth']
```

> Results:
This interaction will identify whether frequent support ticket users with low ratings are more prone to churn, highlighting dissatisfaction patterns.

> Rationale:
* Creating an interaction term between SupportTicketsPerMonth and AverageViewingDuration helps to understand how the combination of customer support engagement and viewing habits affects churn.
* Customers who submit more support tickets and have varying viewing durations may experience differing levels of satisfaction, which can influence their likelihood to churn.

```
# Create interaction term between SupportTicketsPerMonth and AverageViewingDuration
df_eng['SupportTicketsInteraction'] = df_eng['SupportTicketsPerMonth'] * df_eng['AverageViewingDuration']
```

> Results:
* The new feature SupportTicketsInteraction was created by multiplying SupportTicketsPerMonth and AverageViewingDuration, enabling the model to consider how the interplay between support engagement and viewing habits impacts churn predictions.

modeling_preparation_executive_summary

* The dataset was split into three subsets: 80% for training, 10% for validation, and 10% for testing, ensuring a balanced distribution of the target variable for unbiased model evaluation.
* Categorical features were converted to numerical values through label encoding, enhancing the model's ability to interpret these variables effectively.
* Features were standardized to a mean of 0 and a standard deviation of 1, improving convergence and performance for algorithms sensitive to feature scaling.
* Scaled features were converted back to DataFrame format to preserve readability and structure, facilitating easier analysis and integration with tools.
* Overall, these preparations enhance the model's predictive capabilities and provide a solid foundation for further analysis and model tuning.

> Rationale:
* Splitting the dataset ensures that the model is trained on one portion of the data and evaluated on separate subsets for validation and testing.
* The training set is used to fit the model, the validation set helps tune hyperparameters and avoid overfitting, and the test set is kept completely separate to evaluate the final performance of the model on unseen data.
* A typical split of 80% for training, 10% for validation, and 10% for testing helps balance the need for sufficient training data while ensuring enough data is reserved for unbiased model evaluation.

```
# Seperate the features and target
X= df_eng.copy()
y=X.pop(target_name)

from sklearn.model_selection import train_test_split

# Step 1: Split the data into 80% train and 20% (for validation + test)
X_train, X_data, y_train, y_data = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# Step 2: Split the remaining 20% into 10% validation and 10% test sets
X_val, X_test, y_val, y_test = train_test_split(X_data, y_data, test_size=0.5, random_state=42, stratify=y_da
```

```
# Output the sizes of the splits for verification
print("Training set size:", X_train.shape)
print("Validation set size:", X_val.shape)
print("Test set size:", X_test.shape)
```

```
Training set size: (27668, 20)
Validation set size: (3459, 20)
Test set size: (3459, 20)
```

> Results:
The dataset is successfully split into three distinct subsets:
* Training set (80%): It has 27668 instances and Used for learning the model.
* Validation set (10%): It has 3459 instances and Used for model tuning and selection.
* Test set (10%): It has 3459 instances and Held out for final model evaluation to ensure generalization.

G.2 Data Transformation: Encoding Ordinal Categorical Variables

> Rationale:
* The SubscriptionType, PaymentMethod, and MonthlyChargeTier columns have a natural order that can be logically ranked.
* Machine learning models require numerical input, so OrdinalEncoder is used to convert these ordinal categories into numbers while preserving their order.
* This enables the model to interpret the categories correctly, maintaining the hierarchical relationships between them.

```
# Import OrdinalEncoder from sklearn
from sklearn.preprocessing import OrdinalEncoder

# Define OrdinalEncoder for ordinal categorical variables
ordinal_cols = ['SubscriptionType', 'PaymentMethod', 'MonthlyChargeTier']

# Initialize the OrdinalEncoder for transforming categorical features
ordinal_encoder = OrdinalEncoder()

# Apply OrdinalEncoder to the training, validation, and test datasets
X_train[ordinal_cols] = ordinal_encoder.fit_transform(X_train[ordinal_cols])
X_val[ordinal_cols] = ordinal_encoder.transform(X_val[ordinal_cols])
X_test[ordinal_cols] = ordinal_encoder.transform(X_test[ordinal_cols])
```

> Results:
* The categorical columns SubscriptionType, PaymentMethod, and MonthlyChargeTier have been successfully transformed into numerical format using OrdinalEncoder.
* The encoded values preserve the order of the original categories, enabling the model to leverage this ordinal information during training.

G.3 Data Transformation: Label Encoding for Categorical Features

> Rationale:
* Binary categorical features like PaperlessBilling, MultiDeviceAccess, ParentalControl, and SubtitlesEnabled contain two possible values (e.g., "yes" or "no").
* Machine learning models require numerical input, so these binary values must be converted to numeric format.
* LabelEncoder from sklearn is used to encode these binary categorical variables, assigning a 0 to one category and 1 to the other.

```
# Import LabelEncoder from sklearn
from sklearn.preprocessing import LabelEncoder

# Define LabelEncoder for binary categorical variables
label_cols = ['PaperlessBilling', 'MultiDeviceAccess', 'ParentalControl', 'SubtitlesEnabled']

# Initialize the LabelEncoder for transforming categorical features
label_encoder = LabelEncoder()

# Apply LabelEncoder to binary columns in the training, validation, and test datasets
for col in label_cols:
    X_train[col] = label_encoder.fit_transform(X_train[col])
    X_val[col] = label_encoder.transform(X_val[col])
    X_test[col] = label_encoder.transform(X_test[col])
```

> Results:
* The binary columns PaperlessBilling, MultiDeviceAccess, ParentalControl, and SubtitlesEnabled have been successfully transformed into numerical values (0 and 1) across the training, validation, and test datasets.
* This transformation allows the machine learning model to process these features effectively without losing information about their binary nature.

G.4 Data Transformation: Ordinal Encoding for Nominal Categorical Features

> Rationale:
* Nominal categorical features like Cohort, DeviceRegistered, ContentType, and GenrePreference contain categories without any intrinsic ordering.
* However, for certain models, it is beneficial to encode these categories as numerical values to allow the model to understand them better.
* While OrdinalEncoder is typically used for ordinal variables, it can also be applied to nominal variables in this context to ensure they are represented in a numerical format.
* Each unique category in the nominal variables will be assigned a distinct integer value.This transformation facilitates model training by ensuring all features are in a numeric format, which is required by most machine learning algorithms.

```
# Define OrdinalEncoder for nominal categorical variables
nominal_cols = ['DeviceRegistered', 'ContentType', 'GenrePreference']


# Apply OrdinalEncoder to the nominal columns for the training, validation, and test sets
X_train[nominal_cols] = ordinal_encoder.fit_transform(X_train[nominal_cols])
X_val[nominal_cols] = ordinal_encoder.transform(X_val[nominal_cols])
X_test[nominal_cols] = ordinal_encoder.transform(X_test[nominal_cols])
```

> Results:
* The nominal columns Cohort, DeviceRegistered, ContentType, and GenrePreference have been successfully transformed into numerical values.
* Each category in these columns is now represented by a unique integer, allowing machine learning models to effectively interpret and utilize these features during the training process.

```

```

```
# Save training set
# Do not change this code

X_train.to_csv('X_train.csv', index=False)
y_train.to_csv('y_train.csv', index=False)
```

```
# Save validation set
# Do not change this code

X_val.to_csv('X_val.csv', index=False)
y_val.to_csv('y_val.csv', index=False)
```

```
# Save testing set
# Do not change this code

X_test.to_csv('X_test.csv', index=False)
y_test.to_csv('y_test.csv', index=False)
```

I. Assess Baseline Model

`baseline_model_executive_summary`

The baseline model serves as a fundamental benchmark by predicting the most frequent class (non-churn) for all instances in the training, validation, and test datasets. This model provides a reference point to evaluate the performance of more advanced predictive models.
Performance Metrics:
* Training Set Precision: 1.0
* Training Set Recall: 0.0
* Training Set F1-Score: 0.0
* Validation Set Precision: 1.0
* Validation Set Recall: 0.0
* Validation Set F1-Score: 0.0
* Test Set Precision: 1.0
* Test Set Recall: 0.0
* Test Set F1-Score: 0.0
These metrics indicate that while the baseline model achieves perfect precision by classifying all predictions as non-churn, it completely fails to identify any actual churn cases. This results in zero recall and F1-score, highlighting the need for more sophisticated models to improve prediction accuracy and effectively identify churners.

I.1 Simulate Predictions with Baseline Model

> Rationale:
* Using the most frequent class as the baseline helps establish a minimum benchmark for model performance.
* This simple approach allows us to compare the gains achieved by more complex models.
* If a model cannot outperform this baseline, it suggests that it may not be effectively learning from the data.

```
# Define central value for baseline model
y_central = y_train.mode()
```

```
# Generate predictions for training, validation, and test datasets using the baseline model

train_preds = [y_central] * len(y_train)
val_preds = [y_central] * len(y_val)
test_preds = [y_central] * len(y_test)
```

I.2 Selection of Performance Metrics

> Rationale:
* Precision, recall, and F1-score are critical performance metrics in the context of churn prediction.
* Since false positives and false negatives carry different business implications (e.g., incorrectly identifying a customer as likely to churn and offering discounts unnecessarily), these metrics help balance the trade-offs between precision (correctly identifying actual churners) and recall (minimizing missed churners).
* The F1-score provides a harmonic mean, offering a single metric that balances both precision and recall, making it ideal for imbalanced classification problems like churn prediction.

```
# Import the performance metrices
from sklearn.metrics import precision_score,recall_score,f1_score
```

I.2 Baseline Model Performance

```
# Train Metrics
train_precision = precision_score(y_train, train_preds, zero_division=1)
train_recall = recall_score(y_train, train_preds, zero_division=1)
train_f1 = f1_score(y_train, train_preds, zero_division=1)

# Validation Metrics
val_precision = precision_score(y_val, val_preds, zero_division=1)
val_recall = recall_score(y_val, val_preds, zero_division=1)
val_f1 = f1_score(y_val, val_preds, zero_division=1)

# Test Metrics
test_precision = precision_score(y_test, test_preds, zero_division=1)
test_recall = recall_score(y_test, test_preds, zero_division=1)
test_f1 = f1_score(y_test, test_preds, zero_division=1)
```

```
# Print performance metrics for the Training dataset
print("Training Dataset")
print("Precision:", train_precision)
print("Recall:", train_recall)
print("F1-Score:", train_f1)

# Print performance metrics for the Validation dataset
print("\nValidation Dataset")
print("Precision:", val_precision)
print("Recall:", val_recall)
print("F1-Score:", val_f1)

# Print performance metrics for the Test dataset
print("\nTest Dataset")
print("Precision:", test_precision)
print("Recall:", test_recall)
print("F1-Score:", test_f1)
```

```
Training Dataset
Precision: 1.0
Recall: 0.0
F1-Score: 0.0

Validation Dataset
Precision: 1.0
Recall: 0.0
F1-Score: 0.0

Test Dataset
Precision: 1.0
Recall: 0.0
F1-Score: 0.0
```

```
# Define the performance metrics for each dataset
metrics = [ 'Precision', 'Recall', 'F1-Score']
datasets = ['Training', 'Validation', 'Test']

# Values for each dataset (from your precision, recall, f1 scores)
training_metrics = [ train_precision, train_recall, train_f1]
validation_metrics = [ val_precision, val_recall, val_f1]
test_metrics = [ test_precision, test_recall, test_f1]

# Create a DataFrame for easy plotting
data = {
    'Metric': metrics * 3,  # 3 metrics for each dataset
    'Dataset': ['Training']*3 + ['Validation']*3 + ['Test']*3,  # 3 metrics for each dataset
    'Score': training_metrics + validation_metrics + test_metrics  # Concatenating all metrics
}
df_plt = pd.DataFrame(data)

# Set up the plot
plt.figure(figsize=(10, 6))
sns.barplot(x='Metric', y='Score', hue='Dataset', data=df_plt)

# Add plot labels and title
plt.title('Comparison of Precision, Recall, and F1-Score Across Datasets')
plt.ylabel('Score')
plt.ylim(0, 1)  # Since  precision, recall, and F1-Score range between 0 and 1
```

> Results:

* The baseline model, which predicts only the most frequent class (non-churn), demonstrates significant limitations in its ability to handle the churn prediction task.
* While the precision is high (1.0) because the model is correctly identifying all predicted instances as non-churn, the recall is 0.0, indicating that the model fails to identify any actual churners.
* This results in an F1-score of 0.0, showing poor balance between precision and recall.