Understanding business problem

Why do customers Churn?

Customer churn happens for a variety of reasons, but the most common ones include:

1. **Poor Customer Experience**

   Bad service, long wait times, or unhelpful support can frustrate customers and push them toward competitors.

2. **Lack of Engagement**

   If customers don't see the value in your product or service or forget about it, they may cancel or stop using it.

3. **High Prices or Better Offers Elsewhere**

   If competitors offer a better price, promotions, or superior value, customers may leave for a better deal.

4. **Unmet Expectations**

   If your product/service doesn't deliver what was promised, customers will lose trust and leave.

5. **Poor Onboarding Process**

   If users don't understand how to use your product effectively from the start, they may never fully adopt it.

6. **No Personalization or Customization**

   Customers expect businesses to cater to their needs. A one-size-fits-all approach can make them feel undervalued.

7. **Product or Market Fit Issues**

   If your solution doesn't fully address a customer's needs or if their needs change, they might look elsewhere.

8. **Technical Issues or Reliability Problems**

   Frequent downtime, slow performance, or buggy features can lead to frustration and abandonment.

9. **Lack of Loyalty Programs or Incentives**

If there’s no reason for customers to stay, they’re more likely to switch to a competitor.

10. **Business or Personal Changes**

    Sometimes, churn is out of your control—customers might go out of business, change jobs, or experience financial difficulties.

Telecom Dataset

```
# Importing Necessary libraries

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

NumPy: A library for numerical computing in Python, providing support for arrays, matrices, and mathematical functions.

Pandas: A data manipulation and analysis library that offers data structures like DataFrames and Series.

Matplotlib: A plotting library for creating static, animated, and interactive visualizations in Python.

Seaborn: A statistical data visualization library built on Matplotlib, offering attractive and informative graphics.

Data Preprocessing

```
#Clean missing values, handle outliers, and explore feature
distributions.
df= pd.read_csv("C:/Users/bhara/Desktop/DBS - Msc. in Data
Analytics/Python Programming/Assignments/telecom/Telecom-Churn.csv")
print(df.head())

   customerID  gender  SeniorCitizen Partner Dependents  tenure
PhoneService  \
0  7590-VHVEG  Female              0     Yes         No       1
No
1  5575-GNVDE    Male              0      No         No      34
Yes
2  3668-QPYBK    Male              0      No         No       2
Yes
3  7795-CFOCW    Male              0      No         No      45
No
4  9237-HQITU  Female              0      No         No       2
Yes
```

```
      MultipleLines InternetService OnlineSecurity  ... 
DeviceProtection  \
0  No phone service             DSL             No  ... 
No   
1                No             DSL            Yes  ... 
Yes   
2                No             DSL            Yes  ... 
No   
3  No phone service             DSL            Yes  ... 
Yes   
4                No     Fiber optic             No  ... 
No   

  TechSupport StreamingTV StreamingMovies        Contract 
PaperlessBilling  \
0          No          No              No  Month-to-month 
Yes   
1          No          No              No        One year 
No   
2          No          No              No  Month-to-month 
Yes   
3         Yes          No              No        One year 
No   
4          No          No              No  Month-to-month 
Yes   

               PaymentMethod MonthlyCharges  TotalCharges Churn  
0           Electronic check          29.85         29.85    No  
1               Mailed check          56.95        1889.5    No  
2               Mailed check          53.85        108.15   Yes  
3  Bank transfer (automatic)          42.30       1840.75    No  
4           Electronic check          70.70        151.65   Yes  

[5 rows x 21 columns]
```

Meaning of the Columns

customerID-A unique identifier for each customer.

gender-The customer's gender (e.g., Male or Female).

SeniorCitizen-Indicates if the customer is a senior citizen (0 = No, 1 = Yes).

Partner-Whether the customer has a partner (Yes/No).

Dependents-Whether the customer has dependents (e.g., children or other family) (Yes/No).

tenure-Number of months the customer has stayed with the company.

PhoneService-Whether the customer has phone service (Yes/No).

MultipleLines-If the customer has more than one phone line (Yes, No, No phone service).

InternetService-Type of internet service (e.g., DSL, Fiber optic, or No internet service).

OnlineSecurity-Whether the customer has online security add-on service (Yes, No, No internet service).

OnlineBackup-Whether the customer has online backup service (Yes, No, No internet service).

DeviceProtection-Whether the customer has device protection plan (Yes, No, No internet service).

TechSupport-Whether the customer has technical support service (Yes, No, No internet service).

StreamingTV-Whether the customer has streaming TV service (Yes, No, No internet service).

StreamingMovies-Whether the customer has streaming movies service (Yes, No, No internet service).

Contract-The type of contract (Month-to-month, One year, Two year).

PaperlessBilling-Whether the customer uses paperless billing (Yes/No).

PaymentMethod-The payment method (e.g., Electronic check, Mailed check, Bank transfer, Credit card).

MonthlyCharges-The amount charged to the customer monthly.

TotalCharges-The total amount charged to the customer over their entire tenure.

Churn-Whether the customer has left the service (Yes/No). This is the target variable in churn prediction.

Checking Data type of all columns

```
df.dtypes
```

```
customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
```

```
StreamingMovies      object
Contract             object
PaperlessBilling     object
PaymentMethod        object
MonthlyCharges      float64
TotalCharges         object
Churn                object
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   gender            7043 non-null   object
 2   SeniorCitizen     7043 non-null   int64
 3   Partner           7043 non-null   object
 4   Dependents        7043 non-null   object
 5   tenure            7043 non-null   int64
 6   PhoneService      7043 non-null   object
 7   MultipleLines     7043 non-null   object
 8   InternetService   7043 non-null   object
 9   OnlineSecurity    7043 non-null   object
 10  OnlineBackup      7043 non-null   object
 11  DeviceProtection  7043 non-null   object
 12  TechSupport       7043 non-null   object
 13  StreamingTV       7043 non-null   object
 14  StreamingMovies   7043 non-null   object
 15  Contract          7043 non-null   object
 16  PaperlessBilling  7043 non-null   object
 17  PaymentMethod     7043 non-null   object
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object
 20  Churn             7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

```
df.shape
```

```
(7043, 21)
```

```
# Drop customerID (not useful for analysis)
df.drop('customerID', axis=1, inplace=True)

# There are 7043 rows and 21 columns i.e, features

for col in df.columns:
    print(col)
```

```
gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract
PaperlessBilling
PaymentMethod
MonthlyCharges
TotalCharges
Churn
```

```
#Getting the overview of the dataset
df.describe()
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

```
df.apply(lambda x: len(x.unique()))
```

```
gender               2
SeniorCitizen        2
Partner              2
Dependents           2
tenure              73
PhoneService         2
MultipleLines        3
InternetService      3
OnlineSecurity       3
OnlineBackup         3
DeviceProtection     3
TechSupport          3
StreamingTV          3
StreamingMovies      3
```

```
Contract                3
PaperlessBilling        2
PaymentMethod           4
MonthlyCharges       1585
TotalCharges         6531
Churn                   2
dtype: int64
```

```
df['tenure'].unique()
```

```
array([ 1, 34,  2, 45,  8, 22, 10, 28, 62, 13, 16, 58, 49, 25, 69, 52,
71,
       21, 12, 30, 47, 72, 17, 27,  5, 46, 11, 70, 63, 43, 15, 60, 18,
66,
        9,  3, 31, 50, 64, 56,  7, 42, 35, 48, 29, 65, 38, 68, 32, 55,
37,
       36, 41,  6,  4, 33, 67, 23, 57, 61, 14, 20, 53, 40, 59, 24, 44,
19,
       54, 51, 26,  0, 39], dtype=int64)
```

```
df['PaymentMethod'].unique()
```

```
array(['Electronic check', 'Mailed check', 'Bank transfer
(automatic)',
       'Credit card (automatic)'], dtype=object)
```

Remove the duplicate rows, If any and verify using shape function.

```
df_cleaned = df.drop_duplicates()

df_cleaned.shape # No duplicate values

(7021, 20)
```

Handling missing values

```
# Convert 'TotalCharges' to numeric, coercing errors to NaN
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'],
errors='coerce')

# Check for missing values
missing_values = df.isnull().sum()

# Show rows with missing 'TotalCharges' values
missing_total_charges = df[df['TotalCharges'].isnull()]

missing_values, missing_total_charges.head()
```

```
(gender              0
 SeniorCitizen       0
 Partner             0
```

```
 Dependents           0
 tenure               0
 PhoneService         0
 MultipleLines        0
 InternetService      0
 OnlineSecurity       0
 OnlineBackup         0
 DeviceProtection     0
 TechSupport          0
 StreamingTV          0
 StreamingMovies      0
 Contract             0
 PaperlessBilling     0
 PaymentMethod        0
 MonthlyCharges       0
 TotalCharges        11
 Churn                0
 dtype: int64,
       gender  SeniorCitizen Partner Dependents  tenure
PhoneService  \
 488   Female              0     Yes        Yes       0           No

 753     Male              0      No        Yes       0          Yes

 936   Female              0     Yes        Yes       0          Yes

 1082    Male              0     Yes        Yes       0          Yes

 1340  Female              0     Yes        Yes       0           No


          MultipleLines InternetService       OnlineSecurity  \
 488   No phone service             DSL                  Yes
 753                 No              No  No internet service
 936                 No             DSL                  Yes
 1082               Yes              No  No internet service
 1340  No phone service             DSL                  Yes

              OnlineBackup     DeviceProtection
TechSupport  \
 488                    No                  Yes                  Yes

 753   No internet service  No internet service  No internet service

 936                   Yes                  Yes                   No

 1082  No internet service  No internet service  No internet service

 1340                  Yes                  Yes                  Yes
```

```
                StreamingTV      StreamingMovies  Contract
PaperlessBilling  \
 488                    Yes                   No  Two year
Yes
 753   No internet service  No internet service  Two year
No
 936                    Yes                  Yes  Two year
No
 1082  No internet service  No internet service  Two year
No
 1340                   Yes                   No  Two year
No

                    PaymentMethod  MonthlyCharges  TotalCharges Churn
 488   Bank transfer (automatic)           52.55           NaN    No
 753                Mailed check           20.25           NaN    No
 936                Mailed check           80.85           NaN    No
 1082               Mailed check           25.75           NaN    No
 1340    Credit card (automatic)           56.05           NaN
No  )

# Drop rows with missing TotalCharges
df_cleaned = df.dropna(subset=['TotalCharges'])

# Confirm the shape after dropping
df_cleaned.shape

(7032, 20)

# After cleaning the data we are left with 7032 rows and 21 columns
i.e, features
```

If there were missing values

Numerical Data:

```
Mean: df['col'].fillna(df['col'].mean(), inplace=True)

Median (for skewed data): df['col'].fillna(df['col'].median(),
inplace=True)

Mode (for categorical-like numbers): df['col'].fillna(df['col'].mode()
[0], inplace=True)
```

Categorical Data:

```
Mode: df['col'].fillna(df['col'].mode()[0], inplace=True)
```

Advanced Methods:

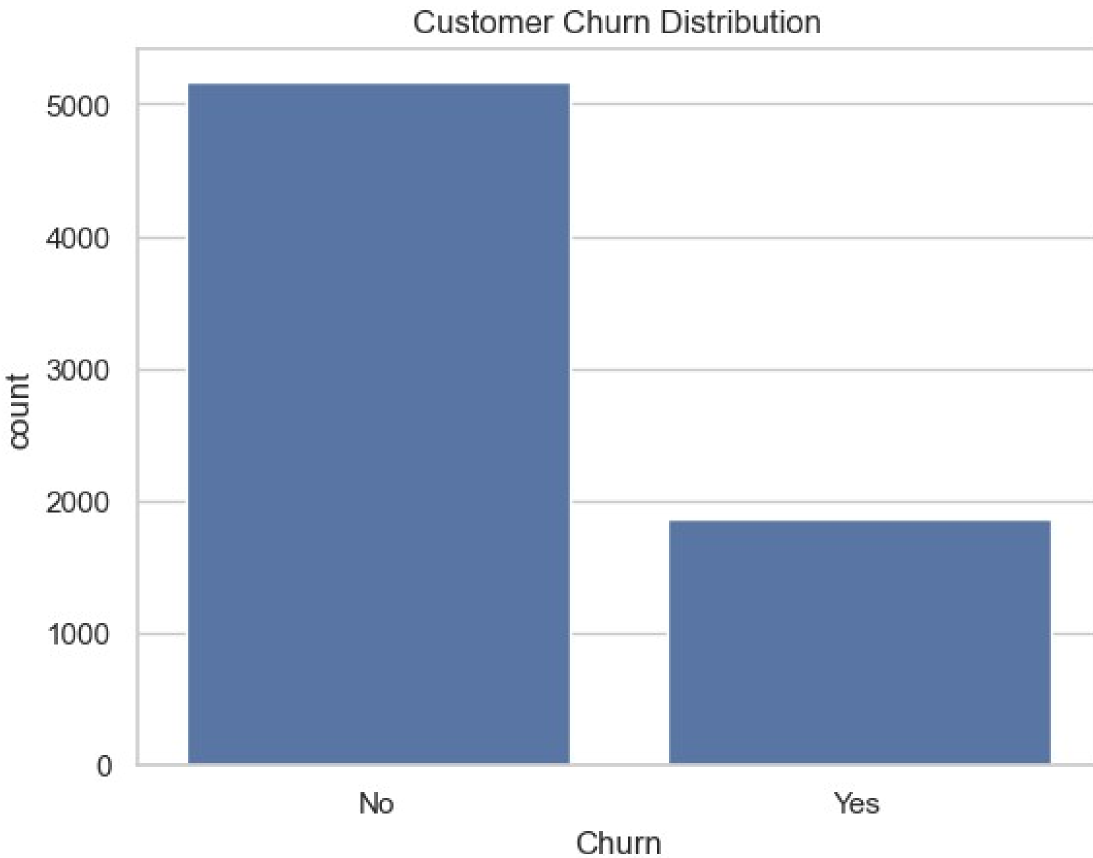
```
Interpolation (for time-series data): df.interpolate(method='linear')

KNN Imputation: Using scikit-learn’s KNNImputer

ML-based Imputation: Train a model (e.g., Random Forest) to predict
missing values.
```

```
#We can handle missing values with these sample codes

# Let A, B, C be columns of a dataframe

# # 1. Fill missing values with Mean
# df['A'].fillna(df['A'].mean(), inplace=True)
# df['B'].fillna(df['B'].mean(), inplace=True)

# # 2. Fill missing values with Median
# df['A'].fillna(df['A'].median(), inplace=True)
# df['B'].fillna(df['B'].median(), inplace=True)

# # 3. Fill missing values with Mode (most frequent value)
# df['C'].fillna(df['C'].mode()[0], inplace=True)

# print("\nDataFrame after handling missing values:")
# print(df)
```

Exploratory Data Analysis

Distribution of Churn

```
sns.countplot(x=df_cleaned["Churn"])
plt.title("Customer Churn Distribution")
plt.show()
```

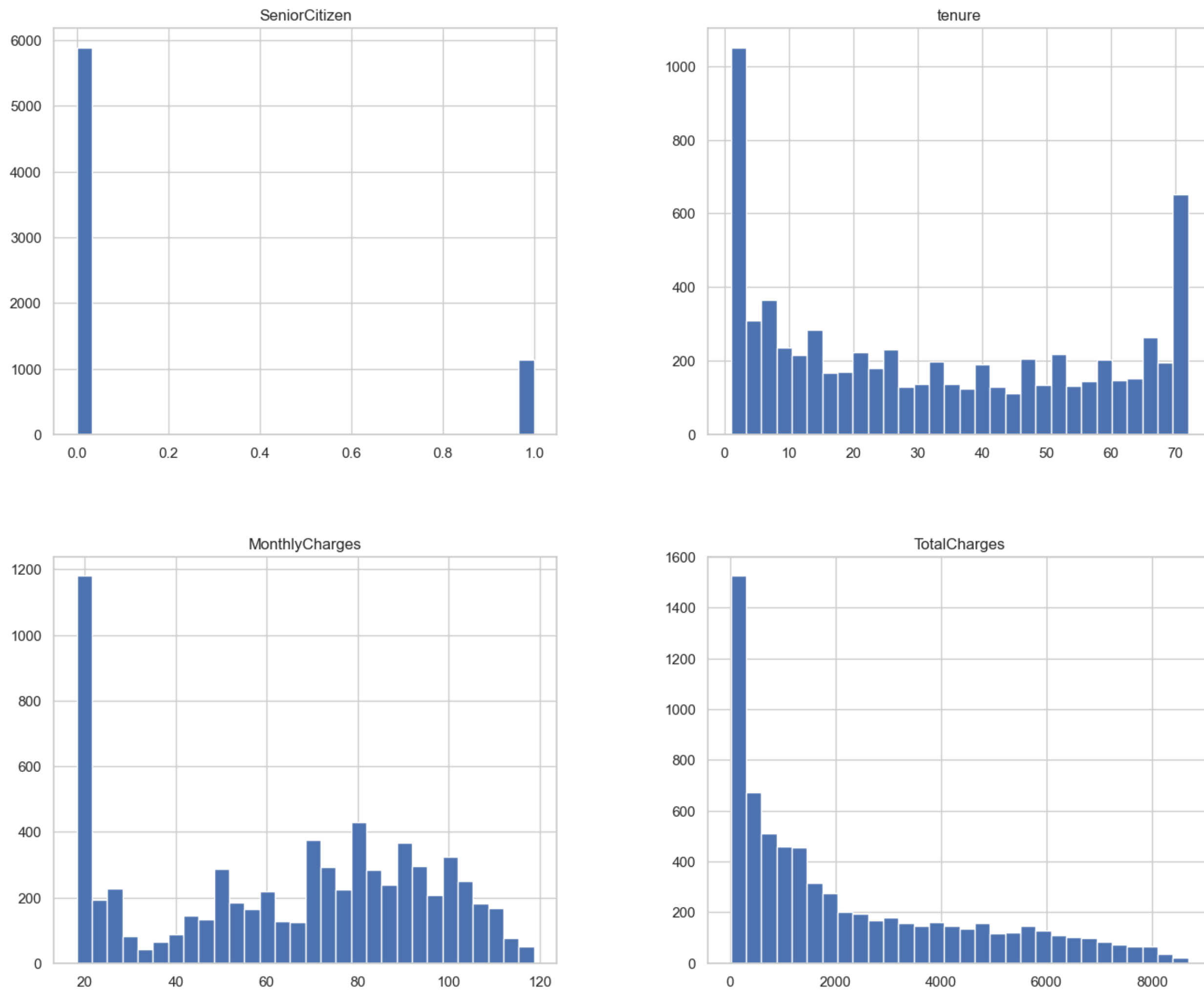


We can infer that 1/3 of our customers churn from our service, let's find out the reason through data analysis

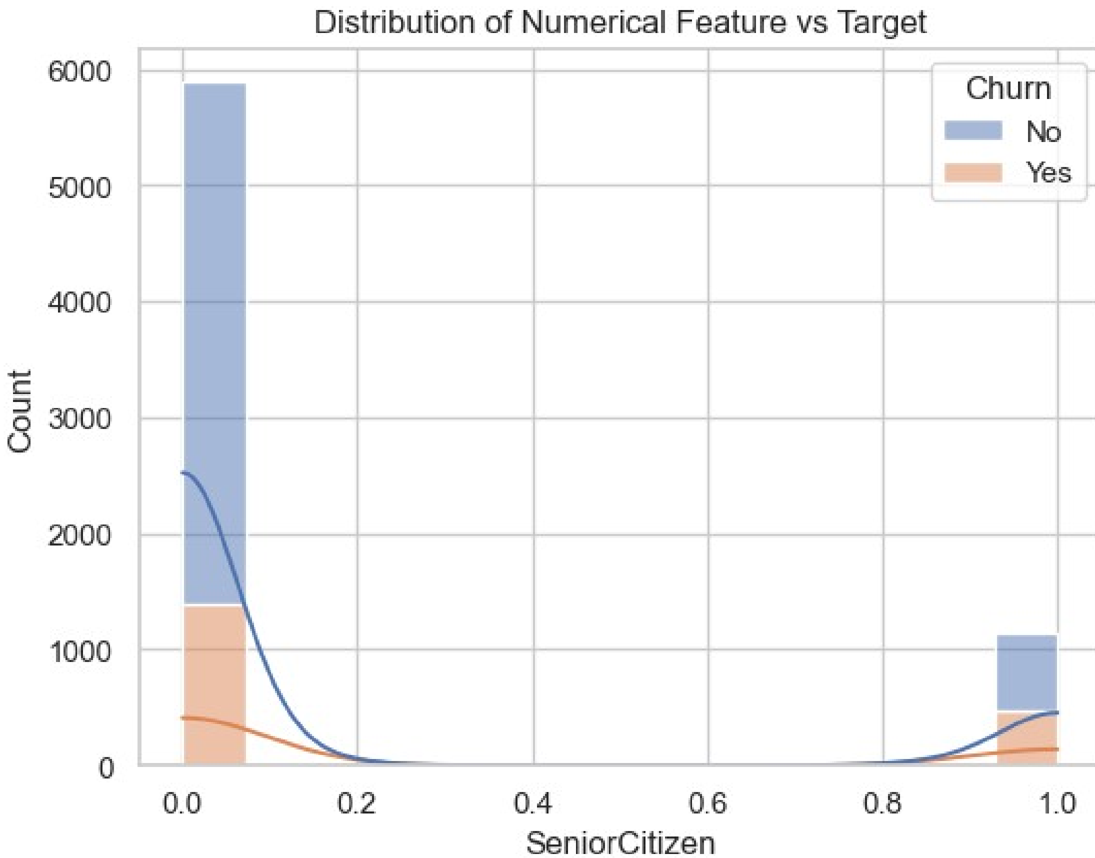
Histogram

Relationship between numerical features and target variable

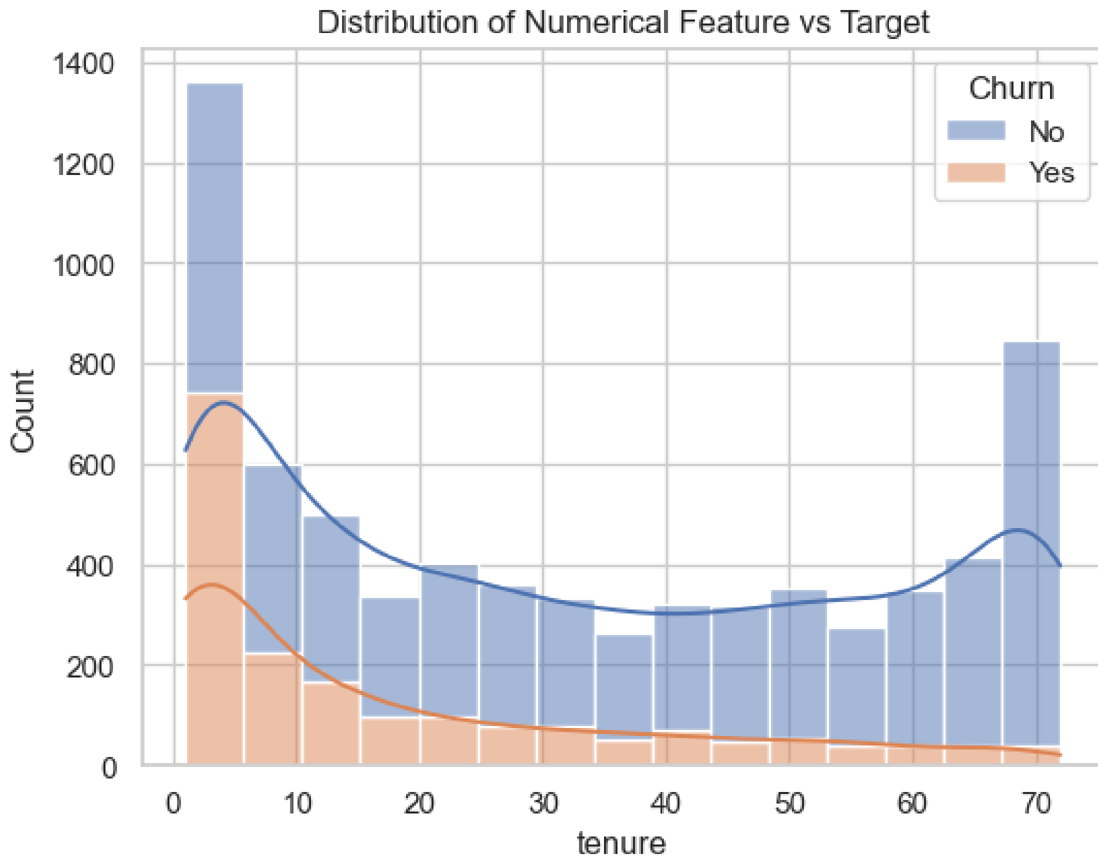
```
df_cleaned.hist(figsize=(16, 13), bins=30)
plt.show()
```

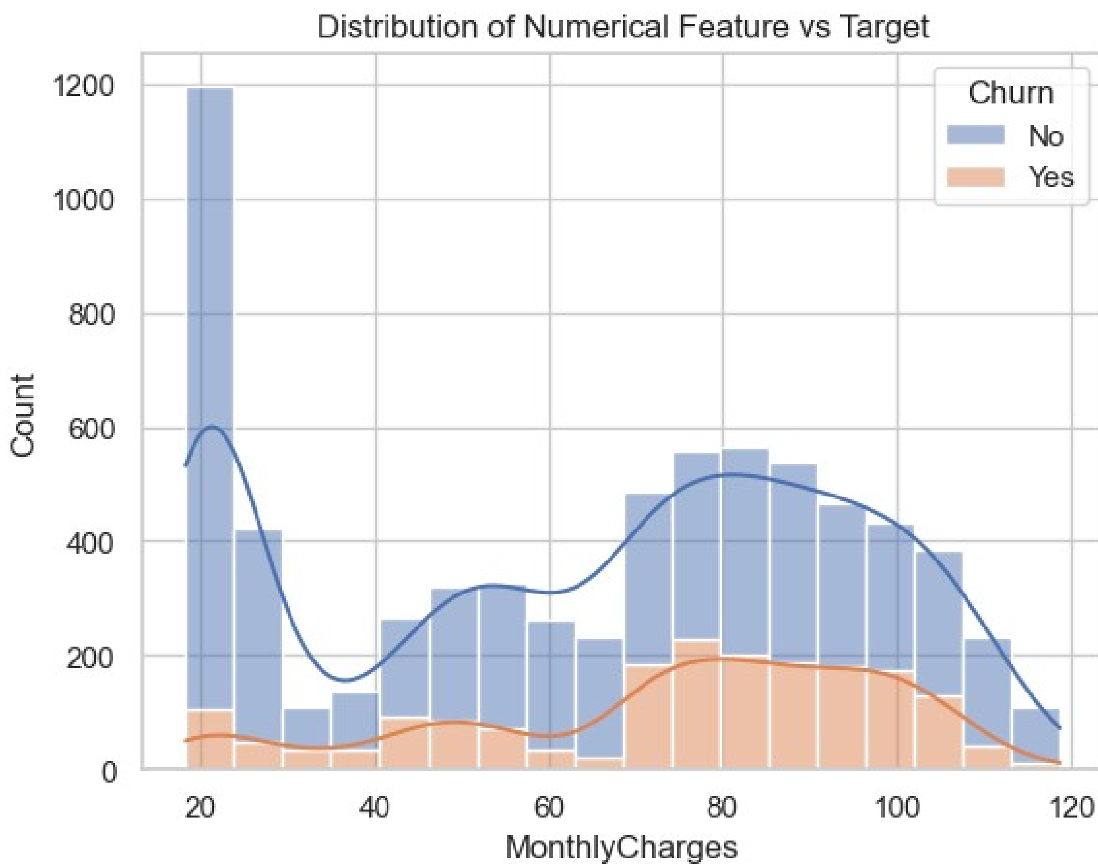


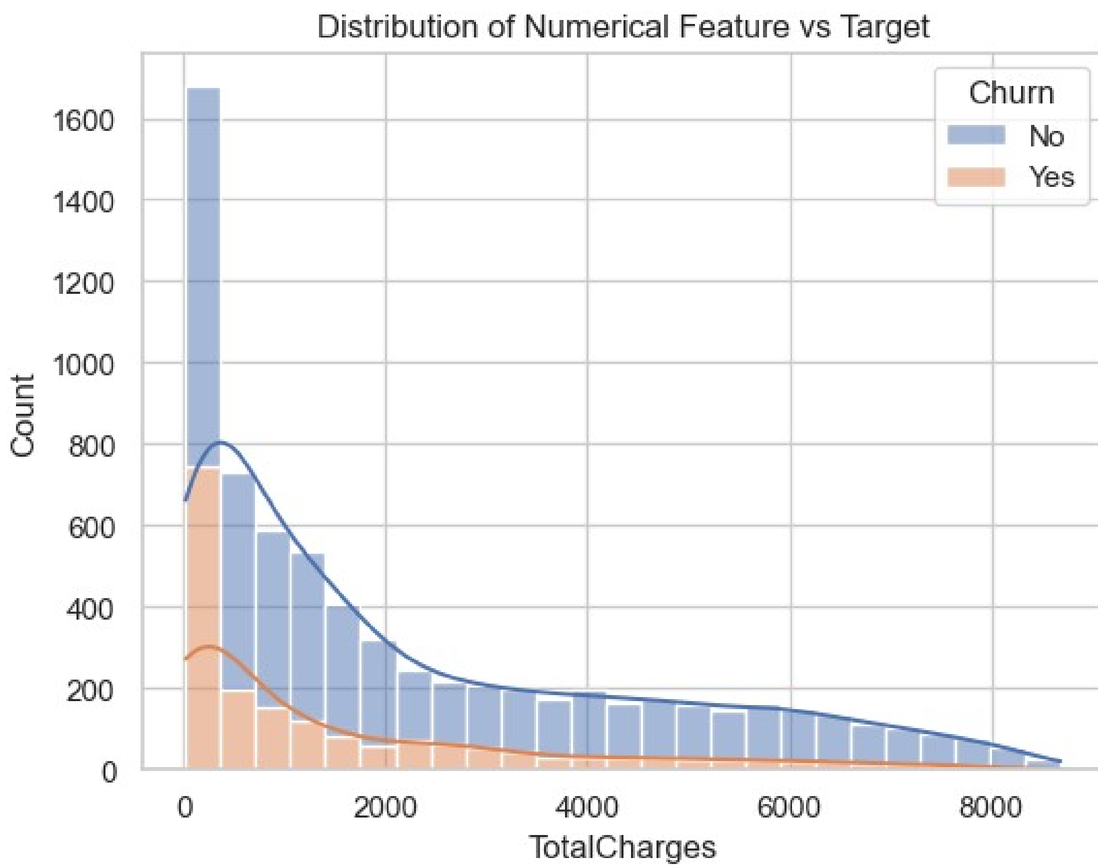

```
df_numeric = df_cleaned.select_dtypes(include=["number"])
for col in df_numeric.columns:
    sns.histplot(data=df_cleaned, x=df_cleaned[col], hue='Churn',
kde=True, multiple="stack")
    plt.title('Distribution of Numerical Feature vs Target')
    plt.show()
```



Distribution of Numerical Feature vs Target
Churn
No
Yes
6000
5000
4000
3000
2000
1000
0
Count
0.0
0.2
0.4
0.6
0.8
1.0
SeniorCitizen


Distribution of Numerical Feature vs Target
Churn
No
Yes
Count
tenure
0
200
400
600
800
1000
1200
1400
0
10
20
30
40
50
60
70

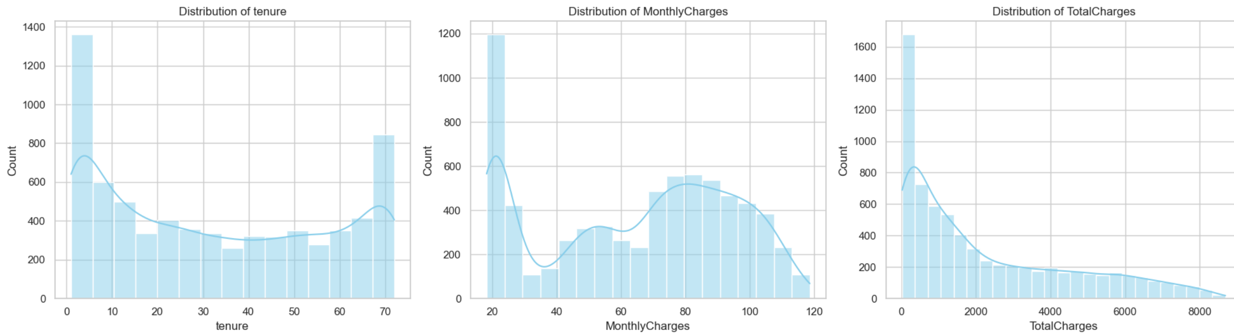

Distribution of Numerical Feature vs Target
Churn
No
Yes
Count
1200
1000
800
600
400
200
0
20
40
60
80
100
120
MonthlyCharges



From the abover histograms we can infer that,

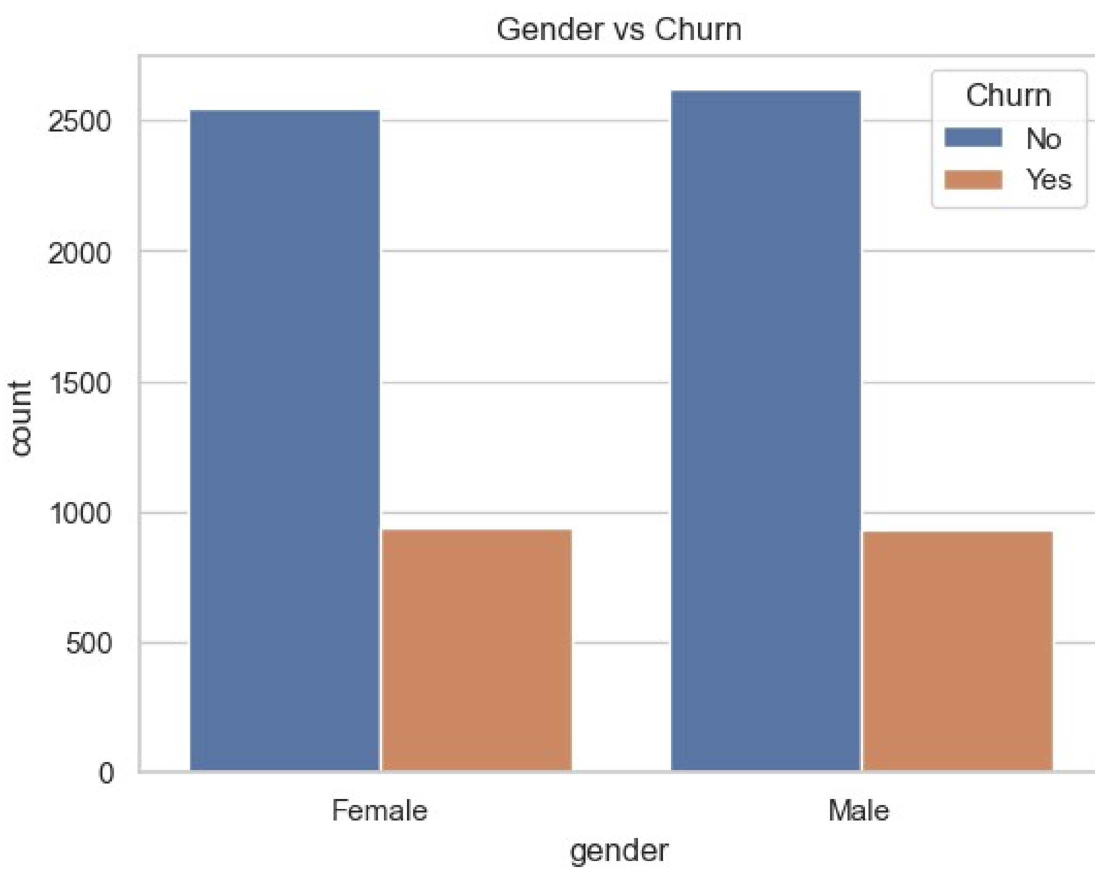
1. 2/3 of customers who churn are not Senior citizens.
2. Most of the customers churn in the first couple of months and within 10 months.
3. The customers with monthly charges greater than 70 tend to Churn.

```
# Set visual style
sns.set(style="whitegrid")

# Plot histograms for numeric columns
fig, axs = plt.subplots(1, 3, figsize=(18, 5))

numeric_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']

for i, col in enumerate(numeric_cols):
    sns.histplot(df_cleaned[col], kde=True, ax=axs[i], 
color='skyblue')
    axs[i].set_title(f'Distribution of {col}')
    axs[i].set_xlabel(col)
    axs[i].set_ylabel('Count')

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

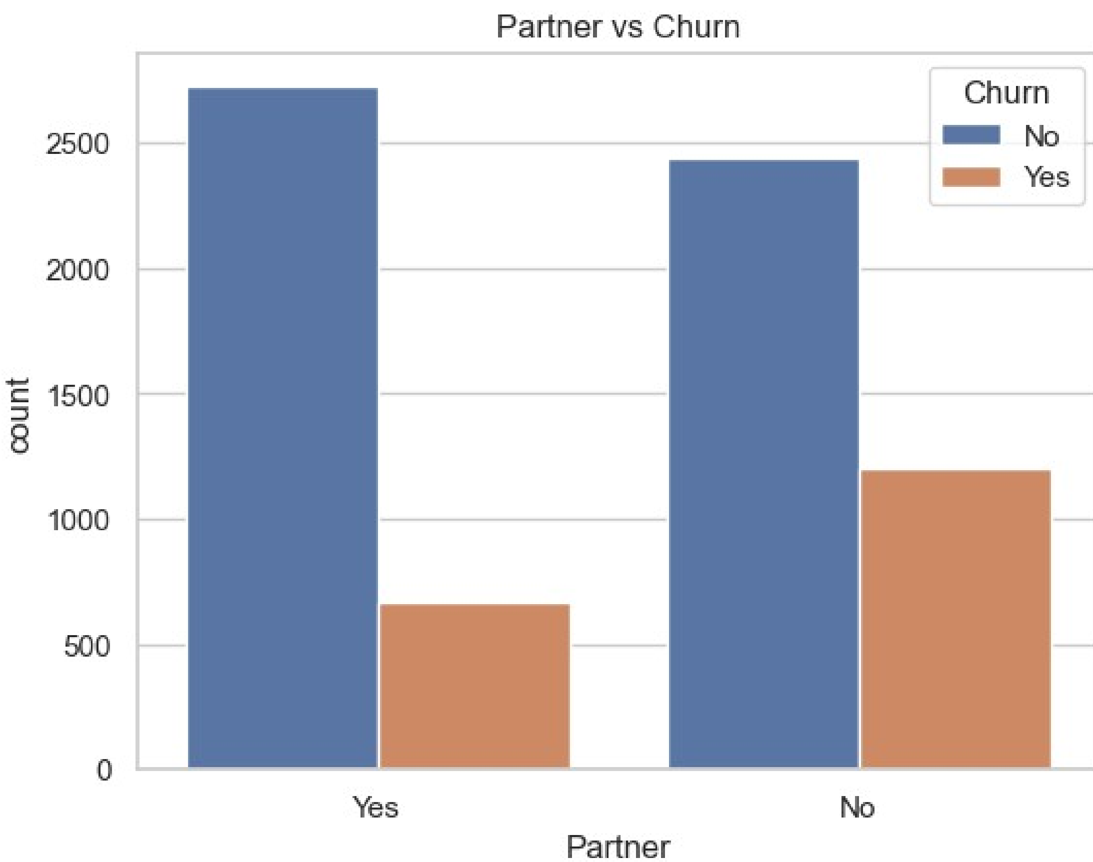


Count Plot

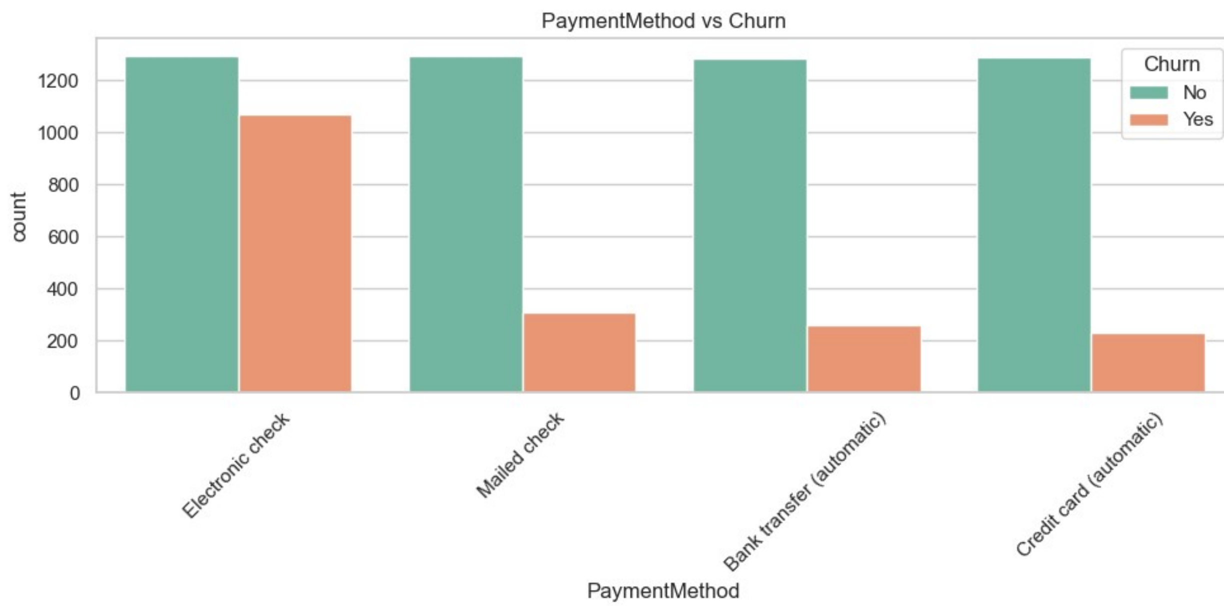
```
sns.countplot(x='gender', hue='Churn', data=df_cleaned)
plt.title('Gender vs Churn')
plt.show()
```

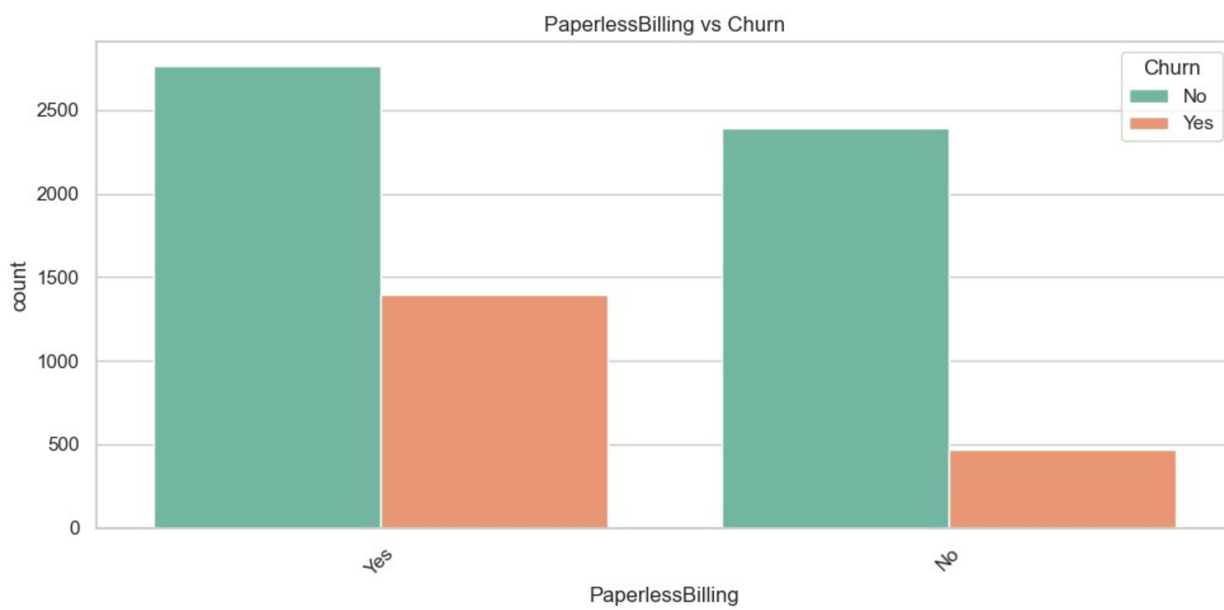


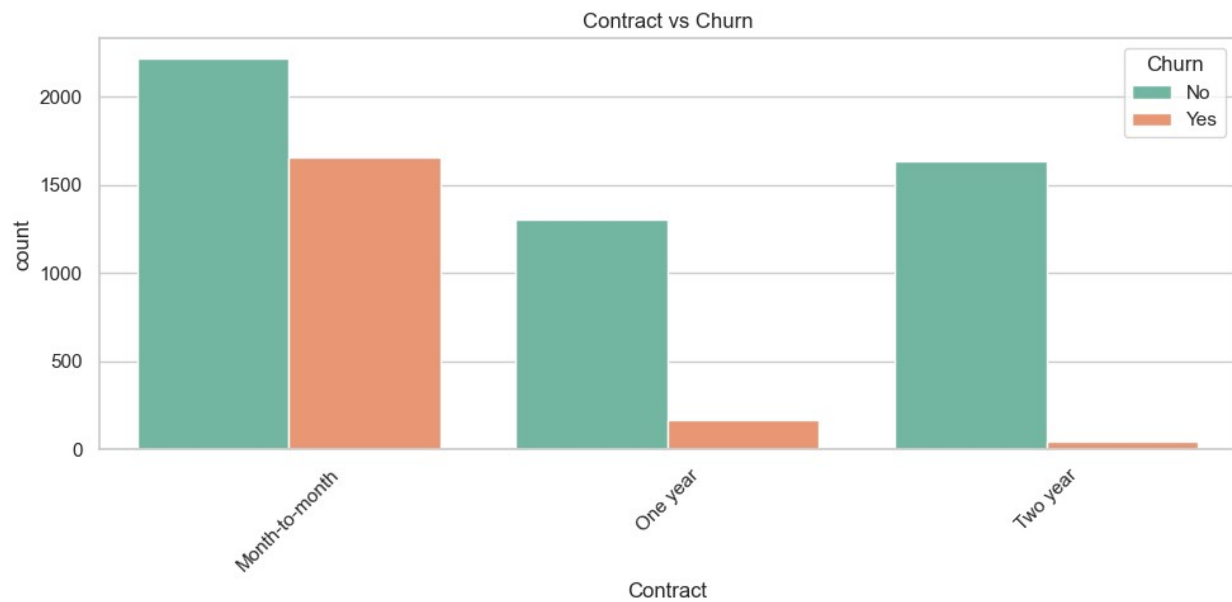

```
sns.countplot(x='Partner', hue='Churn', data=df_cleaned)
plt.title('Partner vs Churn')
plt.show()
```

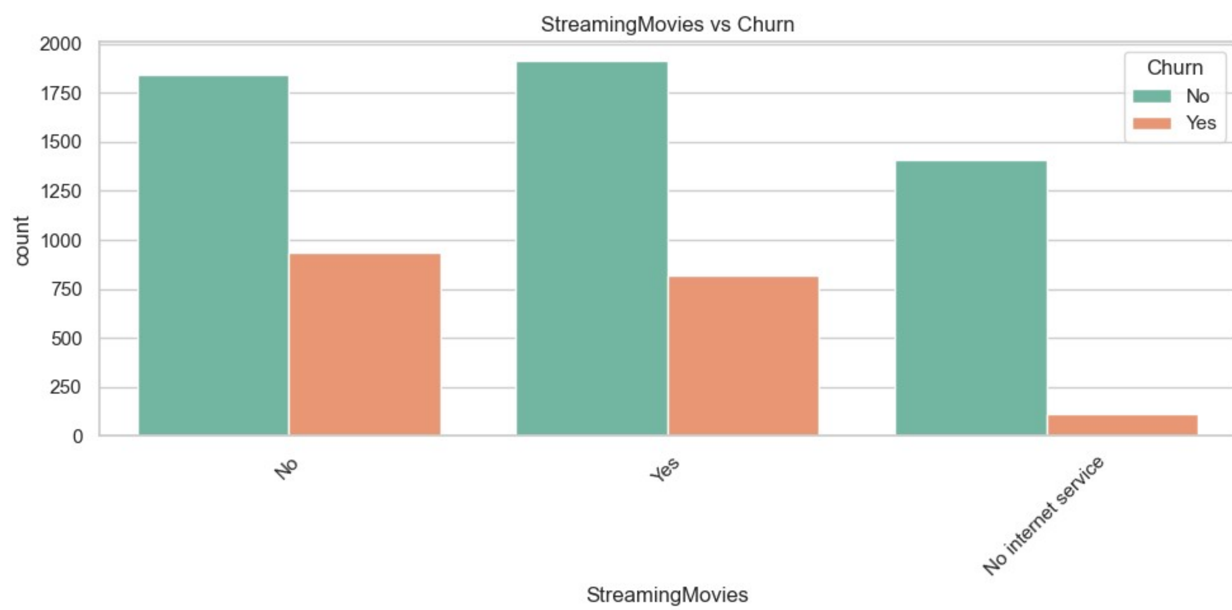


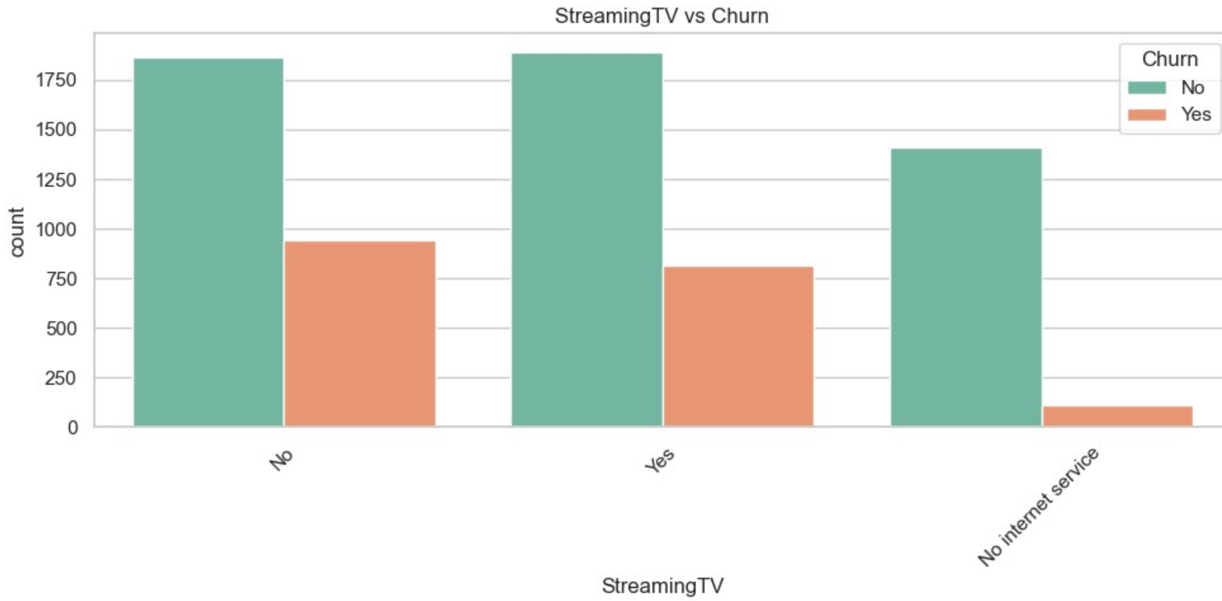

```
# People without the partner tend to churn more.

y = ['PaymentMethod','PaperlessBilling','Contract',
     'StreamingMovies','StreamingTV','TechSupport','DeviceProtection',

'OnlineBackup','OnlineSecurity','InternetService','MultipleLines',
     'PhoneService','Dependents']

for i in y:
    plt.figure(figsize=(10, 5))  # Set figure size for each plot
    sns.countplot(x=i, hue='Churn', data=df_cleaned, palette='Set2')
    plt.title(f'{i} vs Churn')
    plt.xticks(rotation=45)  # Rotate x labels if they overlap
    plt.tight_layout()       # Adjust layout to prevent clipping
    plt.show()
```

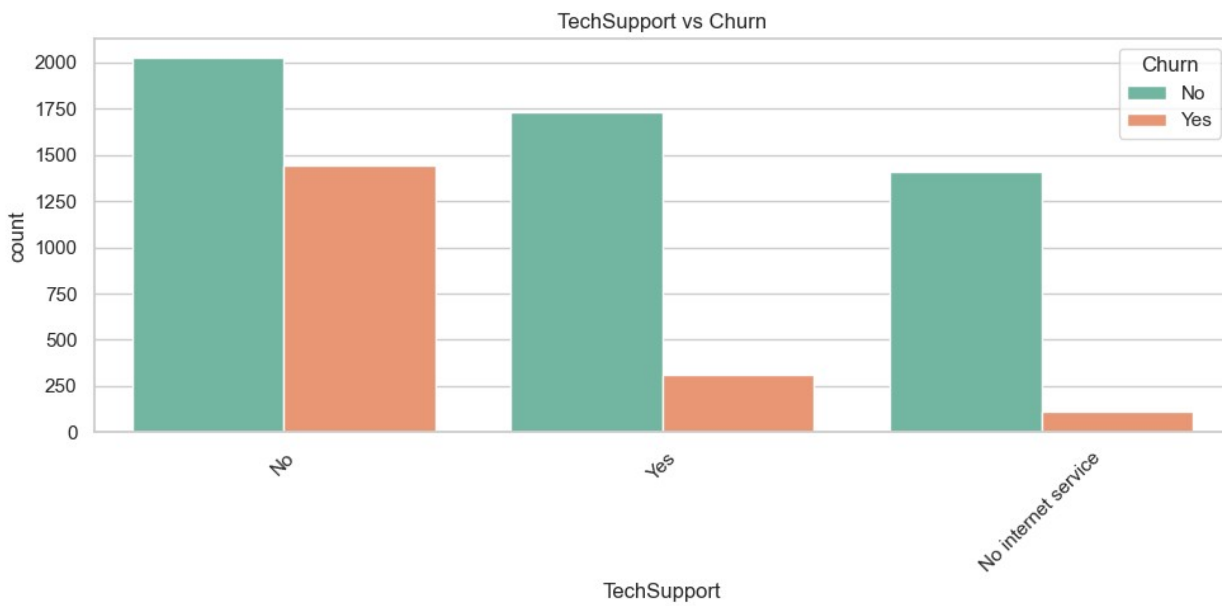

PaymentMethod vs Churn
Churn
No
Yes
count
0
200
400
600
800
1000
1200
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)
PaymentMethod

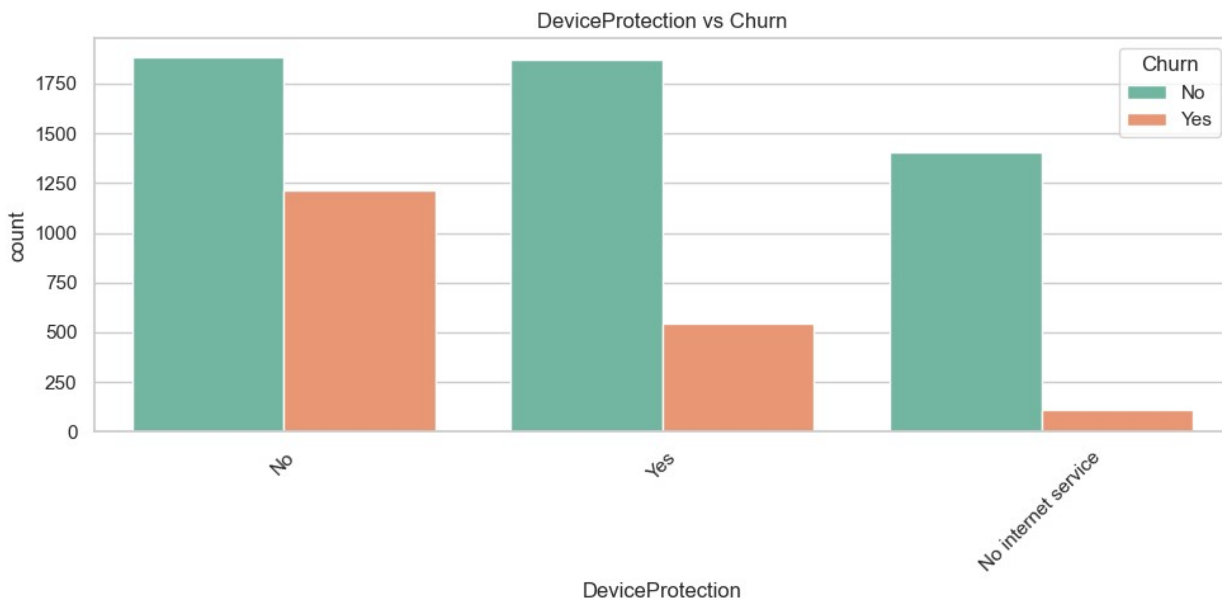

PaperlessBilling vs Churn
Churn
No
Yes
count
0
500
1000
1500
2000
2500
Yes
No
PaperlessBilling

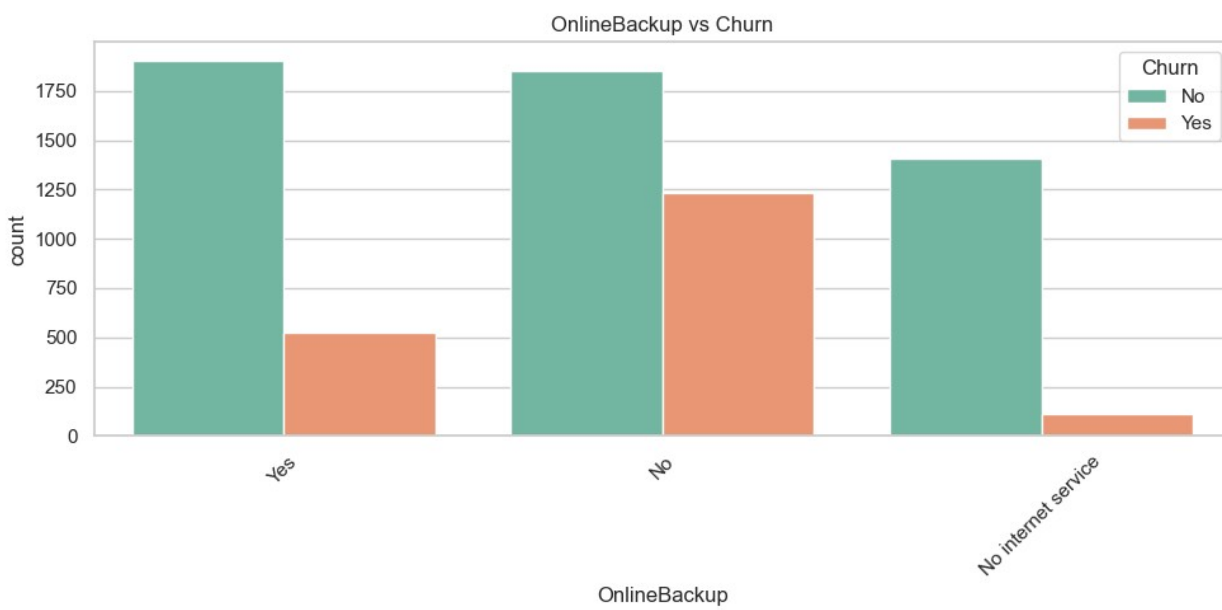

Contract vs Churn
Churn
No
Yes
count
2000
1500
1000
500
0
Month-to-month
One year
Two year
Contract


StreamingMovies vs Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
StreamingMovies

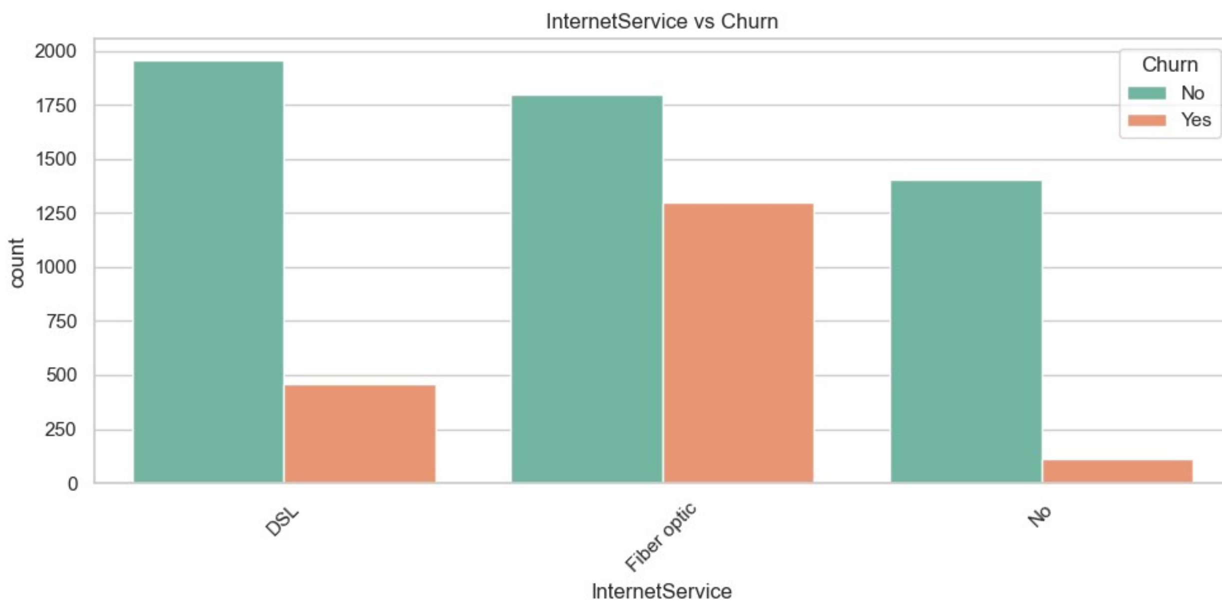

StreamingTV vs Churn
count
1750
1500
1250
1000
750
500
250
0
Churn
No
Yes
No
Yes
No internet service
StreamingTV

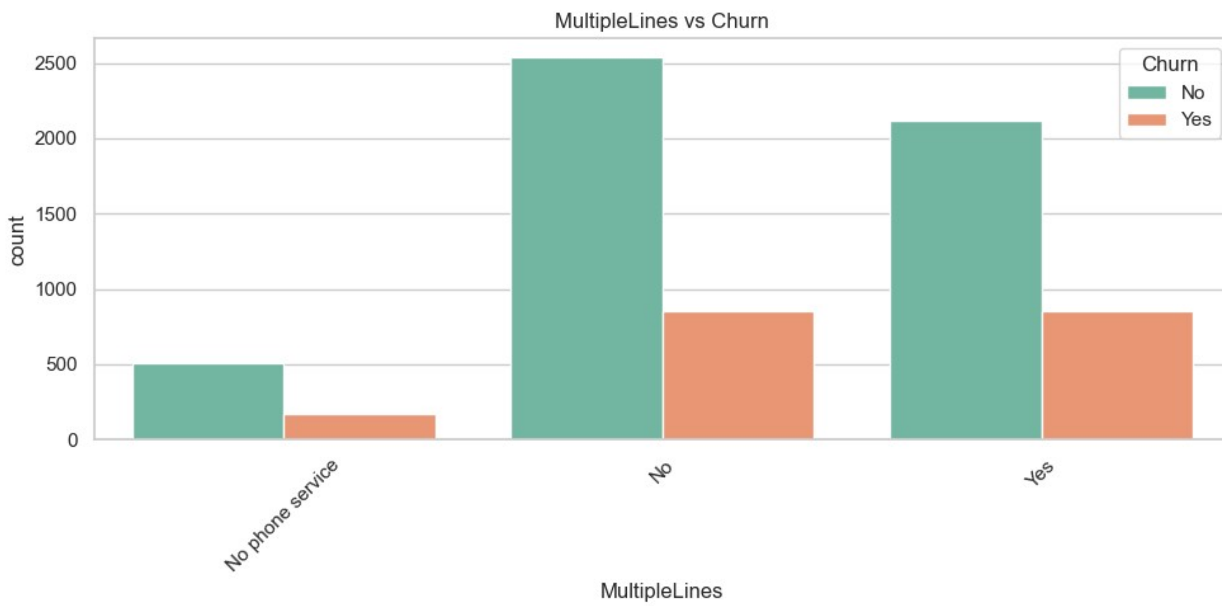

TechSupport vs Churn
count
2000
1750
1500
1250
1000
750
500
250
0
Churn
No
Yes
No
Yes
No internet service
TechSupport


DeviceProtection vs Churn
Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
DeviceProtection

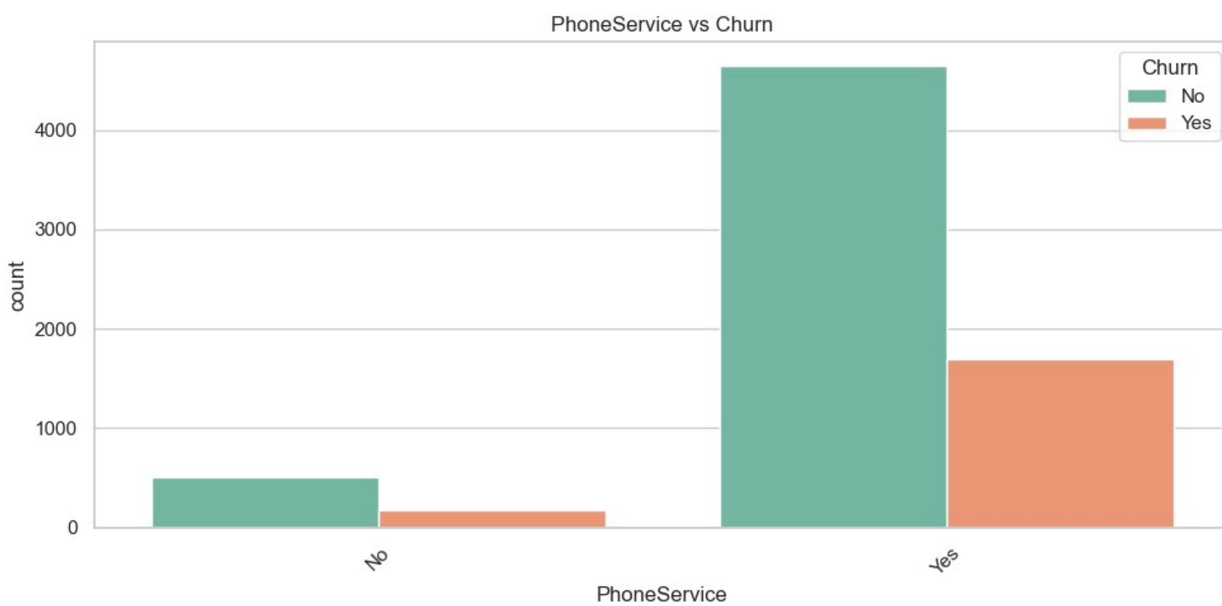

OnlineBackup vs Churn
Churn
No
Yes
count
1750
1500
1250
1000
750
500
250
0
Yes
No
No internet service
OnlineBackup


OnlineSecurity vs Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes
No internet service
OnlineSecurity

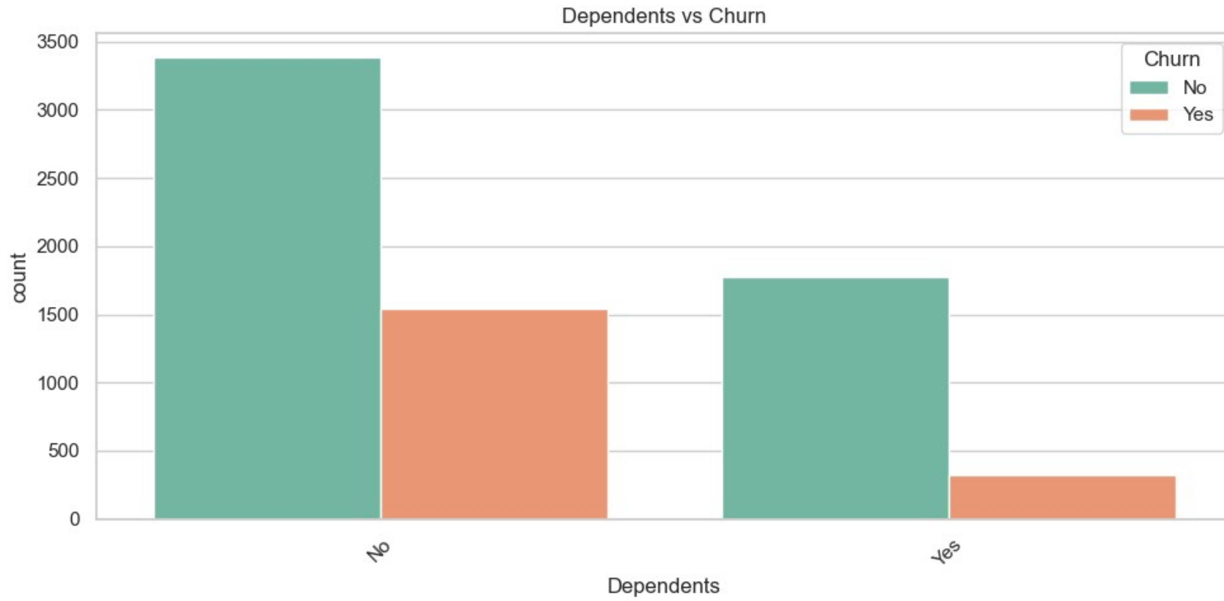

InternetService vs Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService



MultipleLines vs Churn
count
2500
2000
1500
1000
500
0
Churn
No
Yes
No phone service
No
Yes
MultipleLines



PhoneService vs Churn
count
4000
3000
2000
1000
0
Churn
No
Yes
No
Yes
PhoneService

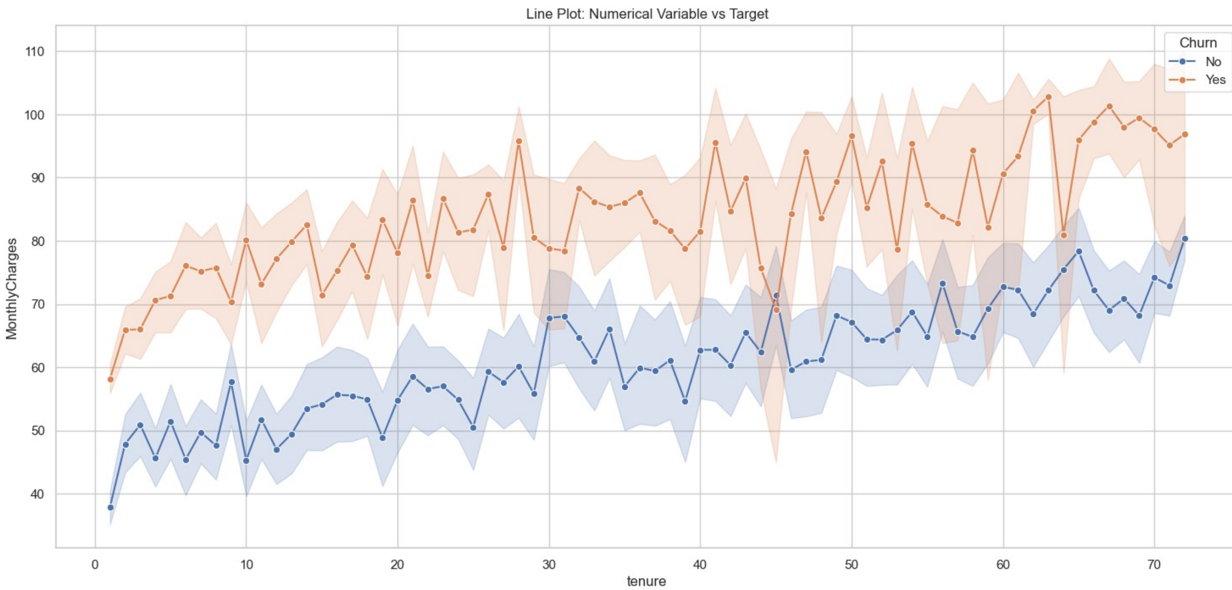


Inference:

1. People who do Electronic check and Paperless Billing tend to churn more.
2. People who have month to month contrat tend to churn the highest.
3. People with the lack of Tech Support tend to churn more.
4. People without Online Backup and Online Security tend to churn more.
5. People with Fiber optic service tend to Churn more.

Line Plot

```
plt.figure(figsize=(18, 8))
sns.lineplot(x='tenure', y='MonthlyCharges', hue='Churn',
data=df_cleaned, marker='o')
plt.title('Line Plot: Numerical Variable vs Target')
plt.show()
```

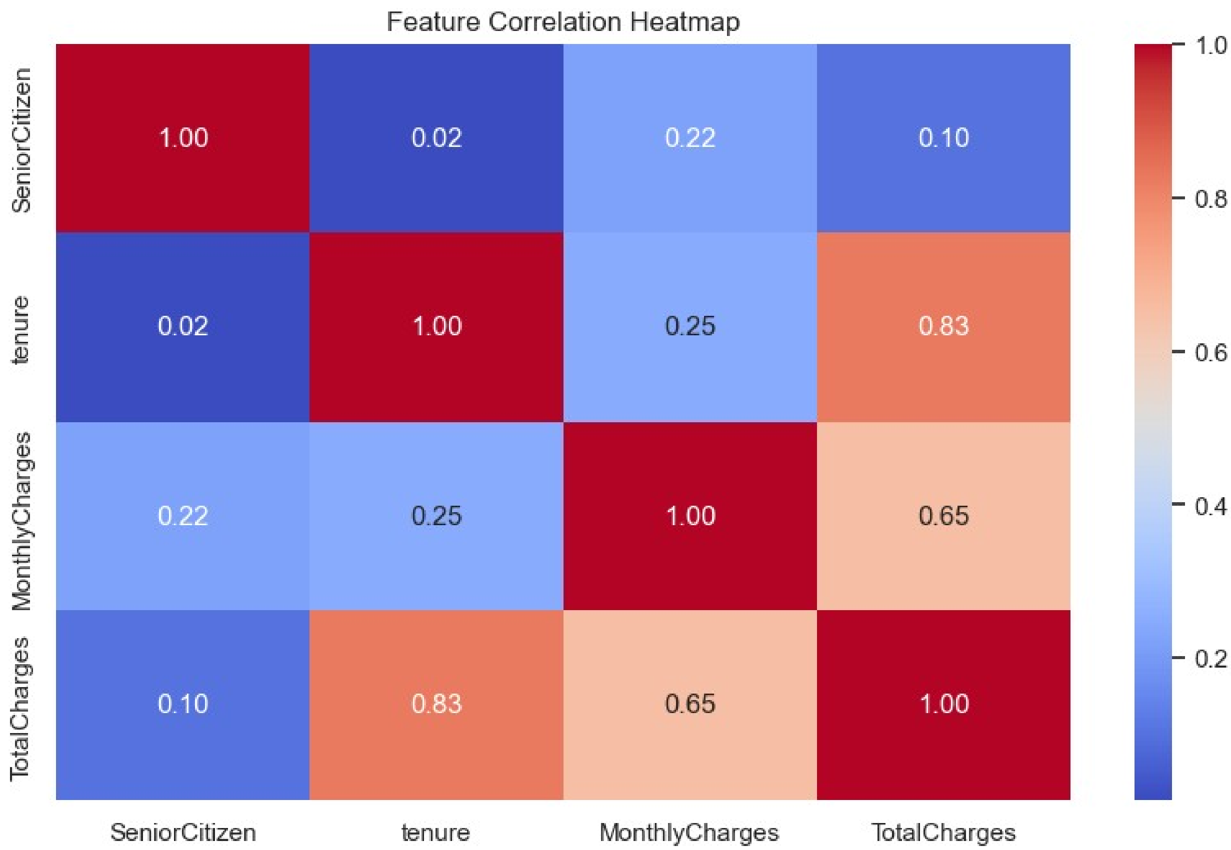


Inference:

Higher monthly charges lead to high churn rate. Also people with longer tenure have high monthly charges which might lead them to cut down the service.

Correlation Matrix

```
plt.figure(figsize=(10, 6))
sns.heatmap(df_numeric.corr(), annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Feature Correlation Heatmap")
plt.show()
```

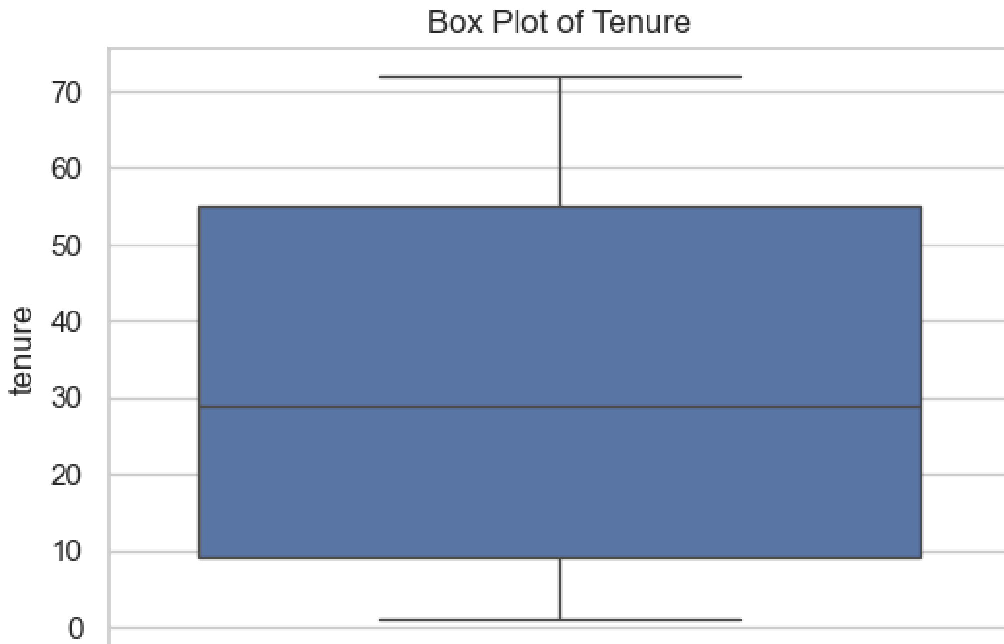


Inference:

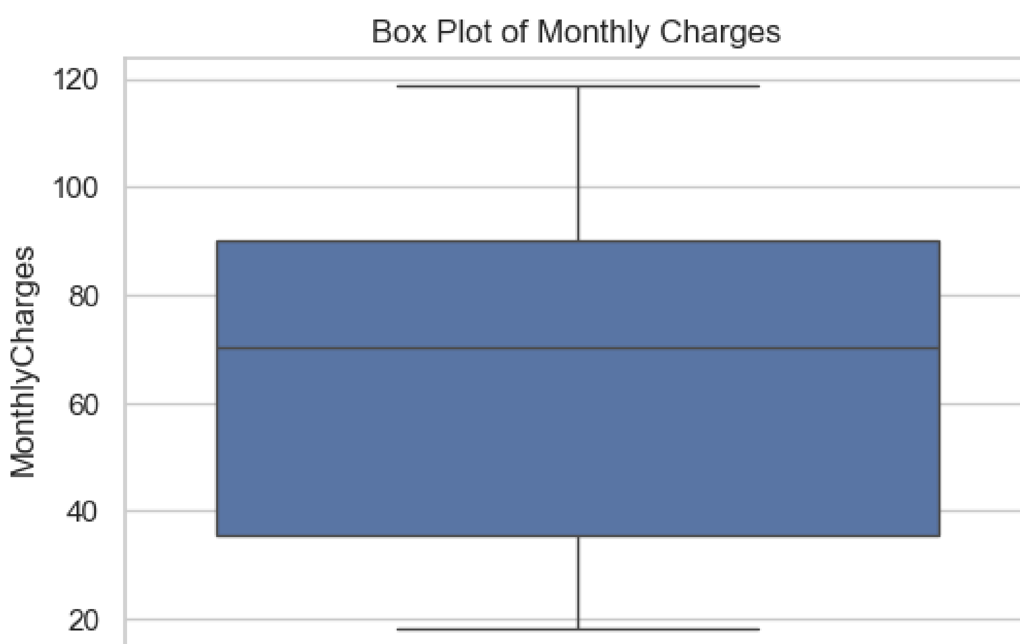
We can clearly see that tenure and Total charges & TotalCharges and Monthly charges are high correlated

Outlier detection using boxplot

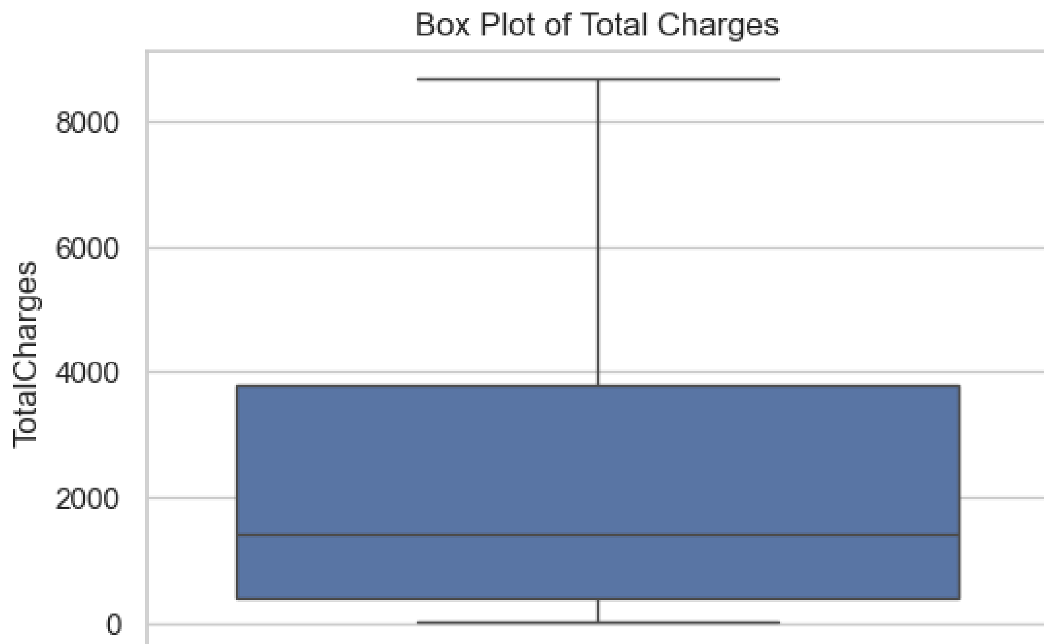
```
# Plots boxplots for all numerical columns in a DataFrame.
plt.figure(figsize=(6, 4))
sns.boxplot(y=df_cleaned['tenure'])  # Boxplot for the specific column
plt.title(f'Box Plot of Tenure')
plt.show()
```




```
# Plots boxplots for all numerical columns in a DataFrame.
plt.figure(figsize=(6, 4))
sns.boxplot(y=df_cleaned['MonthlyCharges'])  # Boxplot for the
specific column
plt.title(f'Box Plot of Monthly Charges')
plt.show()
```

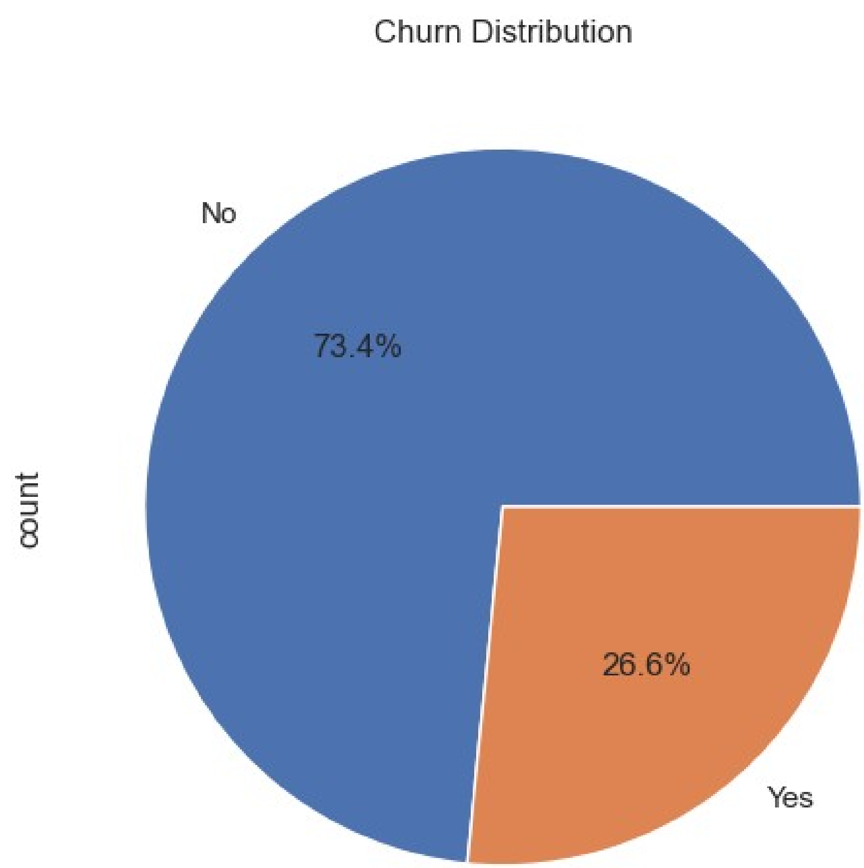


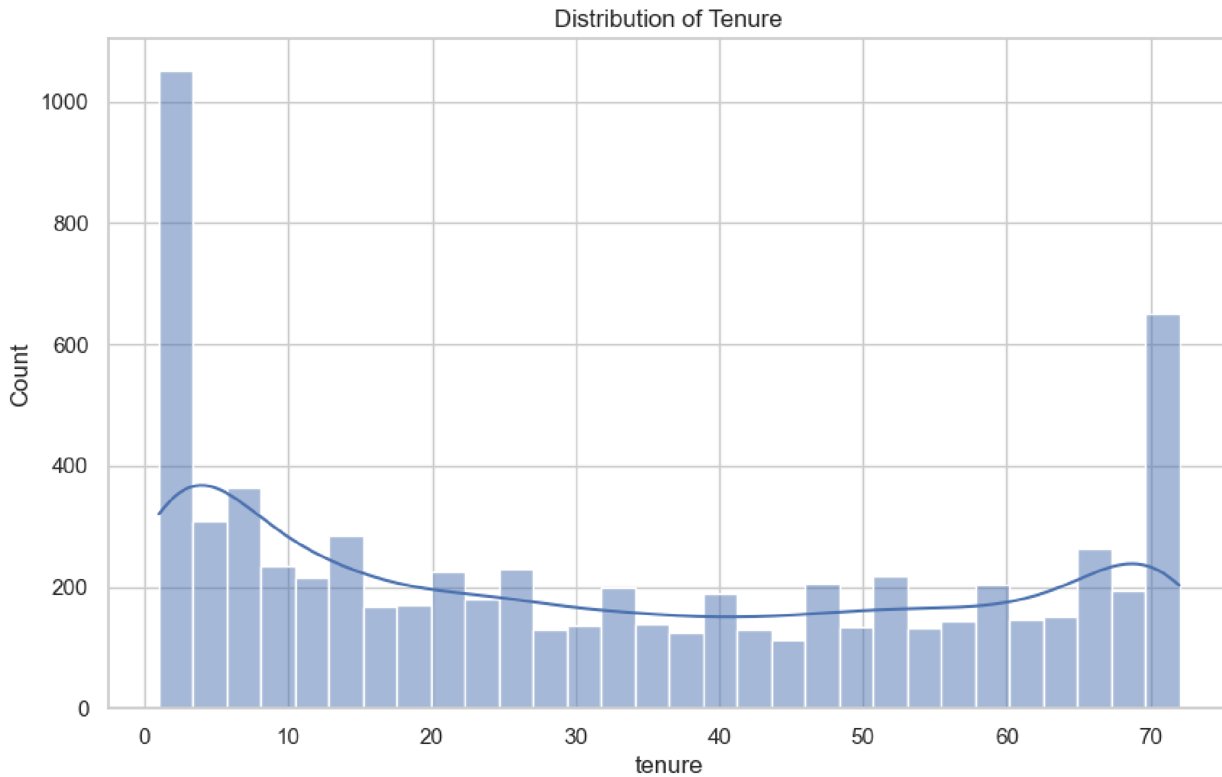
```
# Plots boxplots for all numerical columns in a DataFrame.
plt.figure(figsize=(6, 4))
sns.boxplot(y=df_cleaned['TotalCharges'])  # Boxplot for the specific
column
plt.title(f'Box Plot of Total Charges')
plt.show()
```

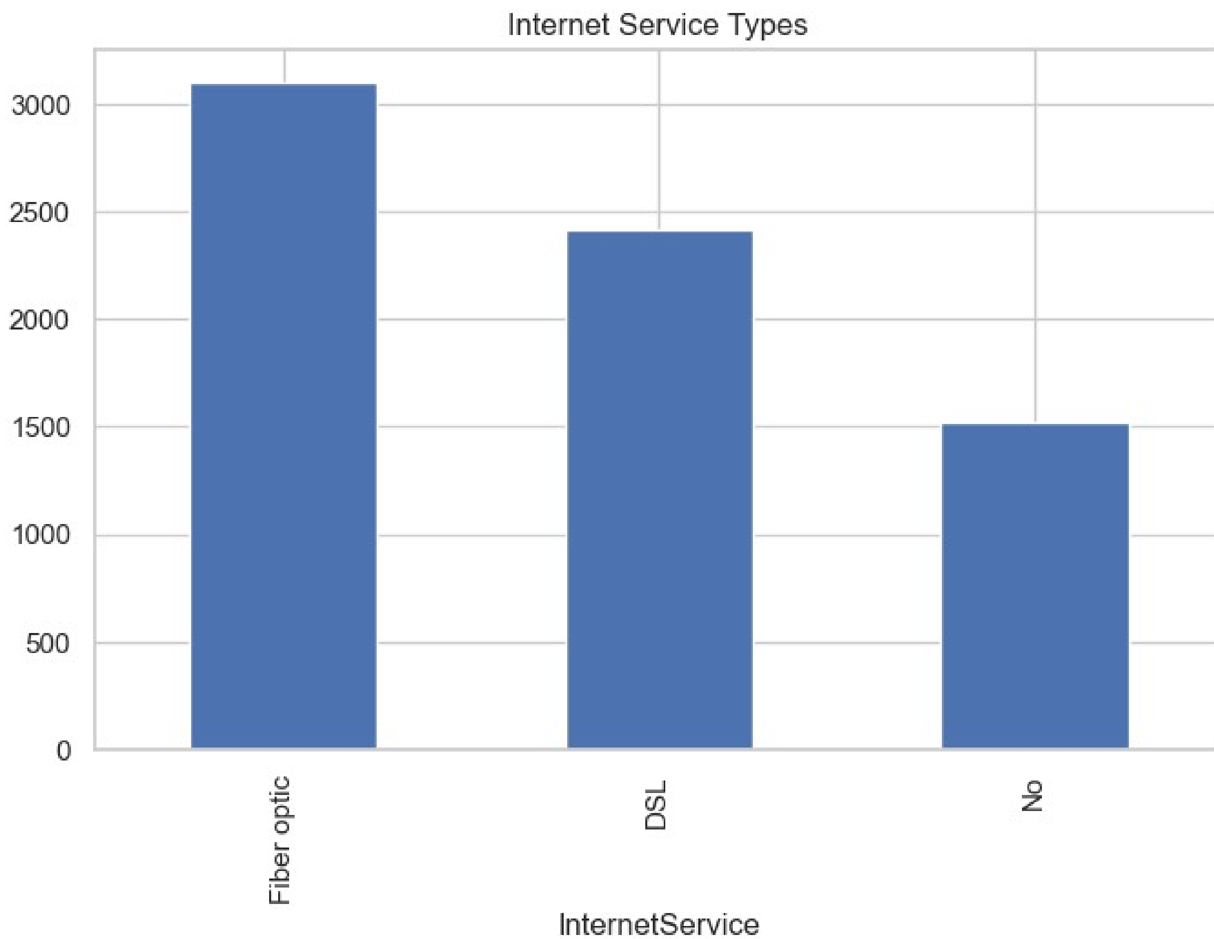


Univariate Analysis

```
# Plot Churn Distribution (Pie Chart)
plt.figure(figsize=(6, 6))
df_cleaned['Churn'].value_counts().plot.pie(autopct='%1.1f%%',
labels=['No', 'Yes'])
plt.title('Churn Distribution')
plt.show()

# Histogram for Tenure
plt.figure(figsize=(10, 6))
sns.histplot(df_cleaned['tenure'], bins=30, kde=True)
plt.title('Distribution of Tenure')
plt.show()

# Bar Chart for Internet Service
plt.figure(figsize=(8, 5))
df_cleaned['InternetService'].value_counts().plot(kind='bar')
plt.title('Internet Service Types')
plt.show()
```

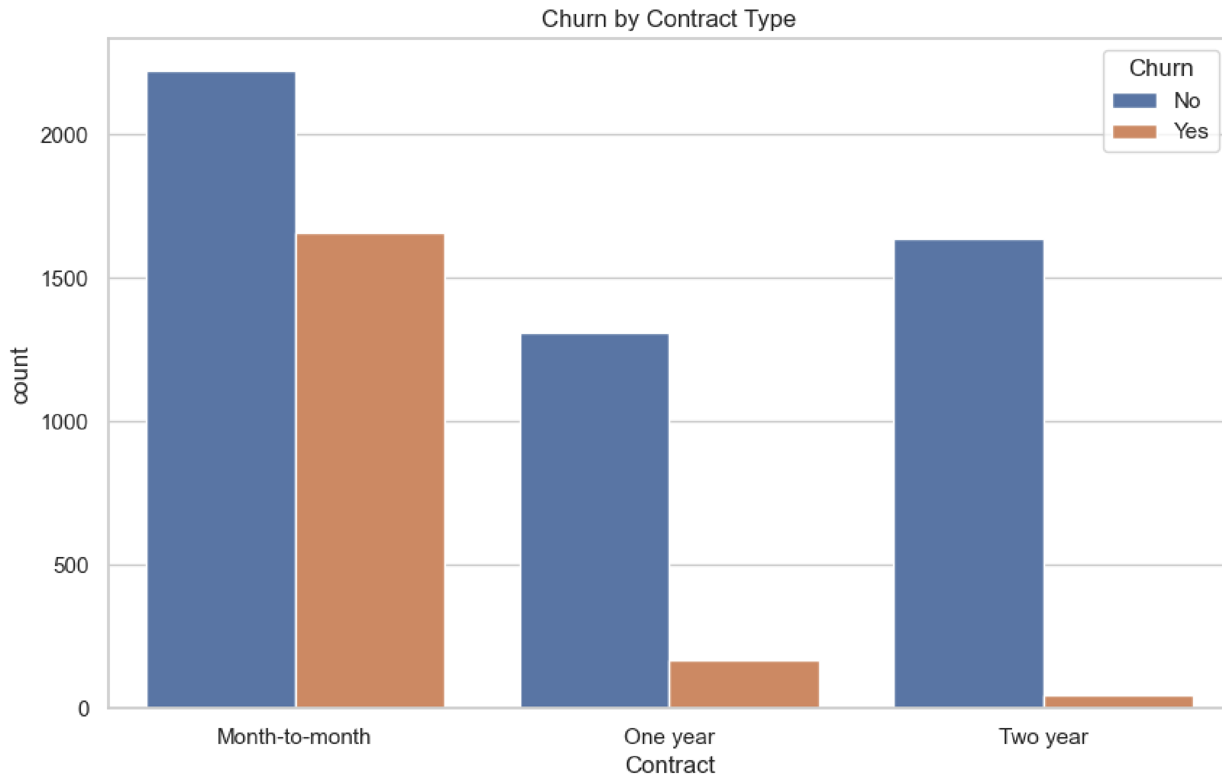

Churn Distribution
No
73.4%
count
26.6%
Yes


Distribution of Tenure
Count
1000
800
600
400
200
0
0
10
20
30
40
50
60
70
tenure

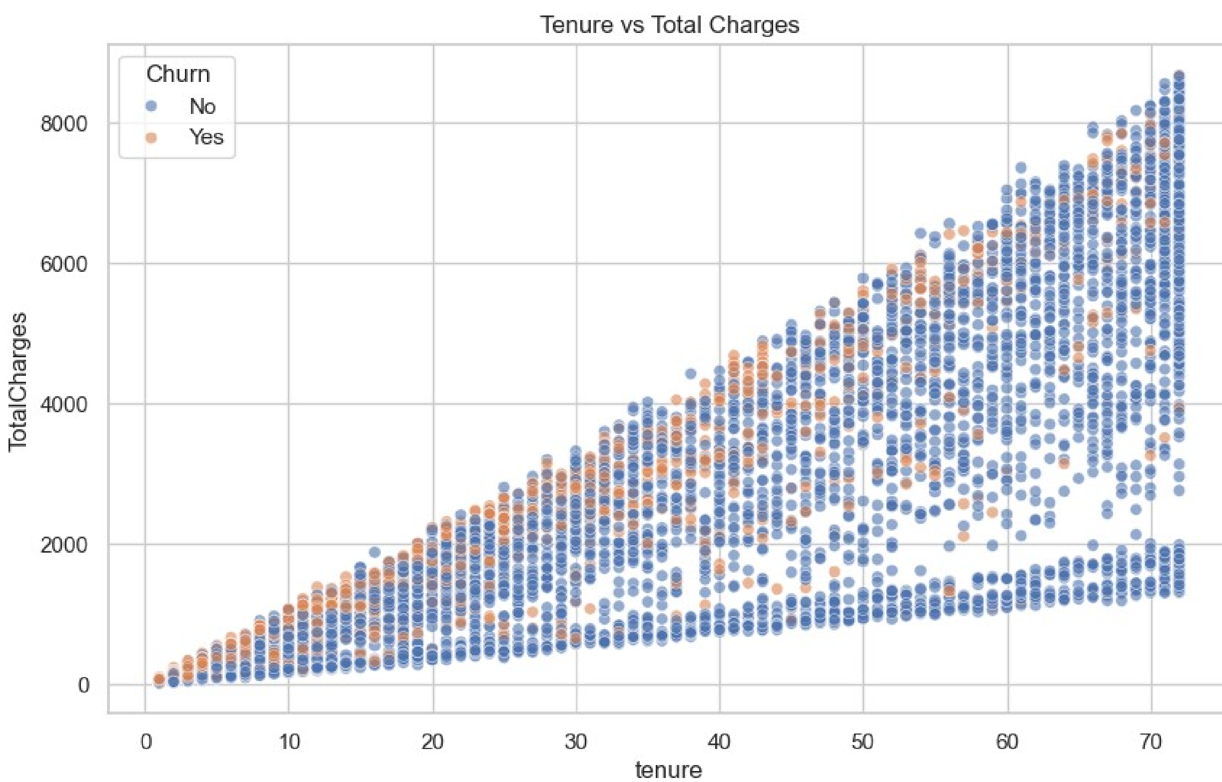


Bivariate Analysis

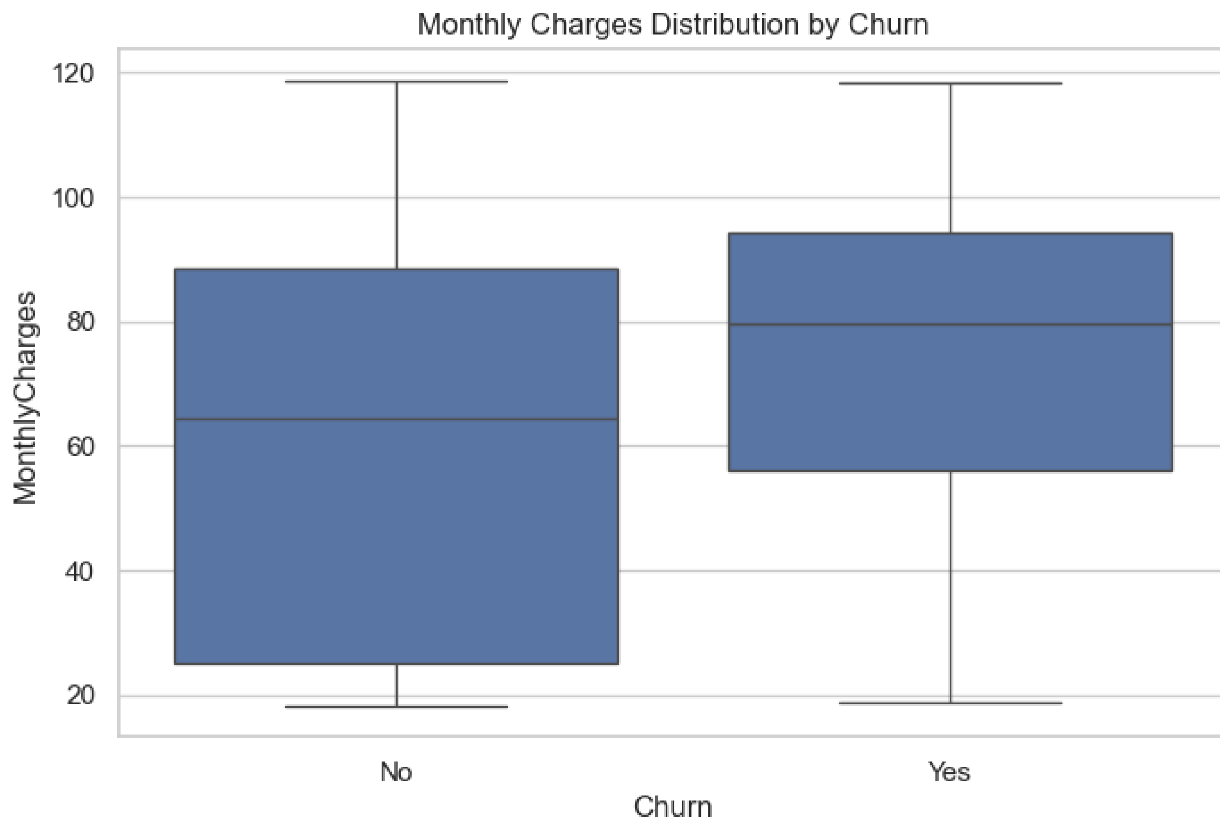
```
# Churn vs Contract Type (Bar Plot)
plt.figure(figsize=(10, 6))
sns.countplot(x='Contract', hue='Churn', data=df_cleaned)
plt.title('Churn by Contract Type')
plt.show()

# Scatter Plot: Tenure vs TotalCharges
plt.figure(figsize=(10, 6))
sns.scatterplot(x='tenure', y='TotalCharges', hue='Churn',
data=df_cleaned, alpha=0.6)
plt.title('Tenure vs Total Charges')
plt.show()

# Box Plot: Monthly Charges by Churn
plt.figure(figsize=(8, 5))
sns.boxplot(x='Churn', y='MonthlyCharges', data=df_cleaned)
plt.title('Monthly Charges Distribution by Churn')
plt.show()
```



Churn by Contract Type
Churn
No
Yes
count
2000
1500
1000
500
0
Month-to-month
One year
Two year
Contract



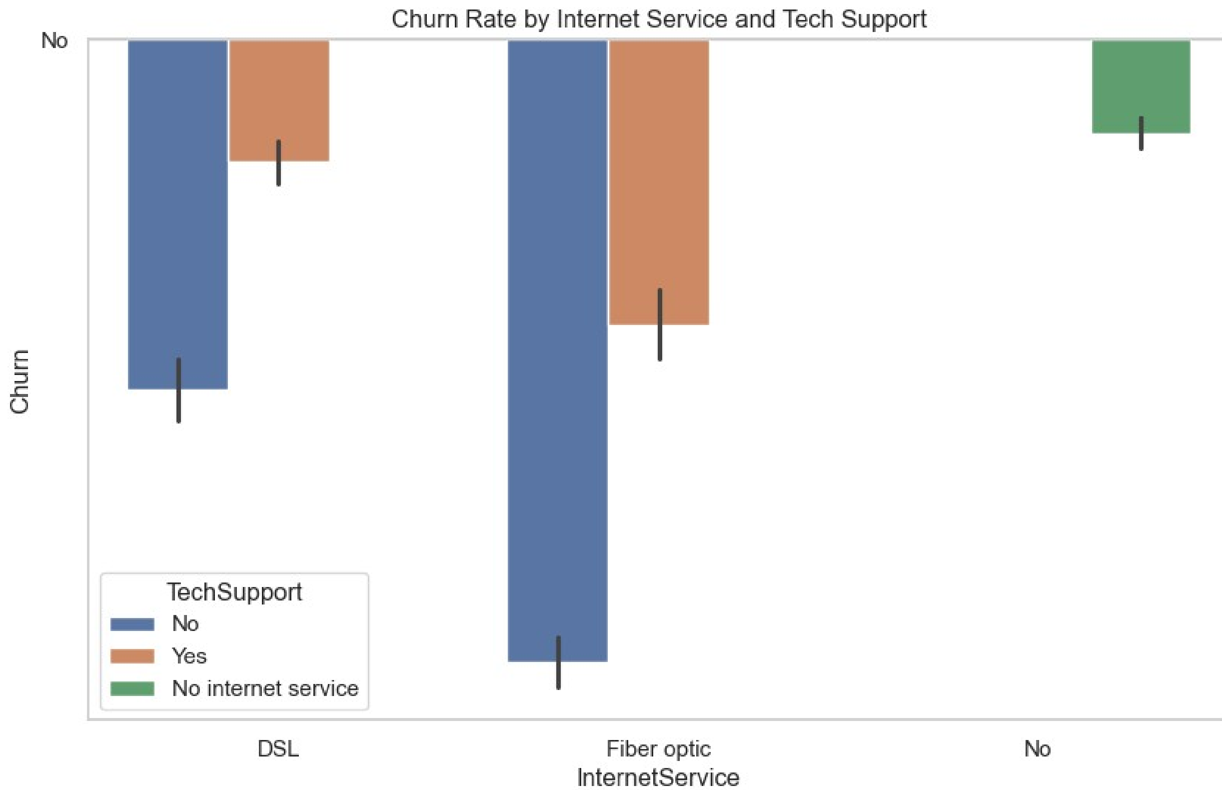
Tenure vs Total Charges
Churn
No
Yes
TotalCharges
8000
6000
4000
2000
0
0
10
20
30
40
50
60
70
tenure

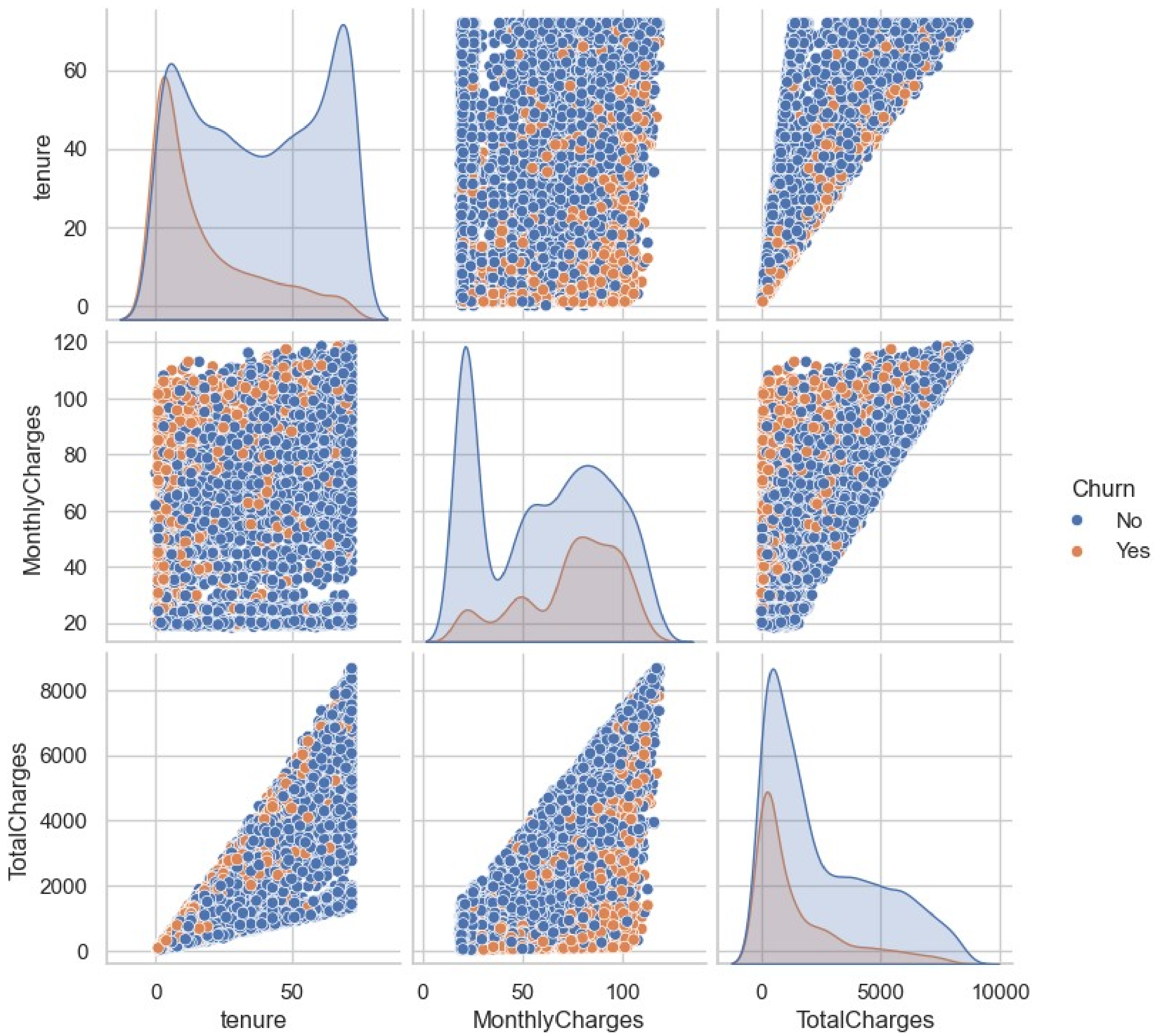


Advanced Analysis

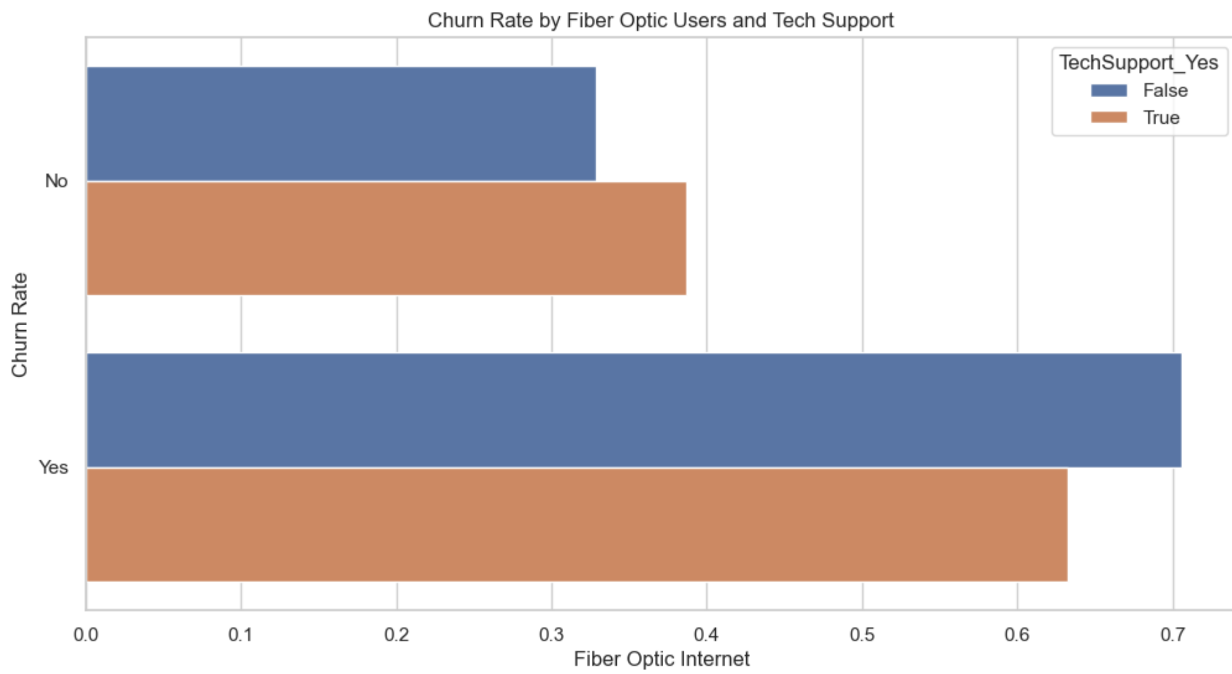
Multivarite analysis

```
# Churn Rate by Internet Service and Tech Support
plt.figure(figsize=(10, 6))
sns.barplot(x='InternetService', y='Churn', hue='TechSupport',
data=df)
plt.title('Churn Rate by Internet Service and Tech Support')
plt.show()

# Pair Plot for Numeric Variables
sns.pairplot(df[['tenure', 'MonthlyCharges', 'TotalCharges',
'Churn']], hue='Churn')
plt.show()
```


Churn Rate by Internet Service and Tech Support
No
Churn
TechSupport
No
Yes
No internet service
DSL
Fiber optic
No
InternetService




```
plt.figure(figsize=(12, 6))
sns.barplot(x='InternetService_Fiber optic', y='Churn',
hue='TechSupport_Yes',
            data=df, errorbar=None)
plt.title('Churn Rate by Fiber Optic Users and Tech Support')
plt.xlabel('Fiber Optic Internet')
plt.ylabel('Churn Rate')
plt.show()
```



Advanced Data Preprocessing

Handling Categorical Variables

```
df = df_cleaned

df.info()

<class 'pandas.core.frame.DataFrame'>
Index: 7032 entries, 0 to 7042
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   gender            7032 non-null   object
 1   SeniorCitizen     7032 non-null   int64
 2   Partner           7032 non-null   object
 3   Dependents        7032 non-null   object
 4   tenure            7032 non-null   int64
 5   PhoneService      7032 non-null   object
 6   MultipleLines     7032 non-null   object
 7   InternetService   7032 non-null   object
 8   OnlineSecurity    7032 non-null   object
 9   OnlineBackup      7032 non-null   object
 10  DeviceProtection  7032 non-null   object
 11  TechSupport       7032 non-null   object
 12  StreamingTV       7032 non-null   object
 13  StreamingMovies   7032 non-null   object
 14  Contract          7032 non-null   object
 15  PaperlessBilling  7032 non-null   object
```

```
 16  PaymentMethod     7032 non-null   object
 17  MonthlyCharges    7032 non-null   float64
 18  TotalCharges      7032 non-null   float64
 19  Churn             7032 non-null   object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

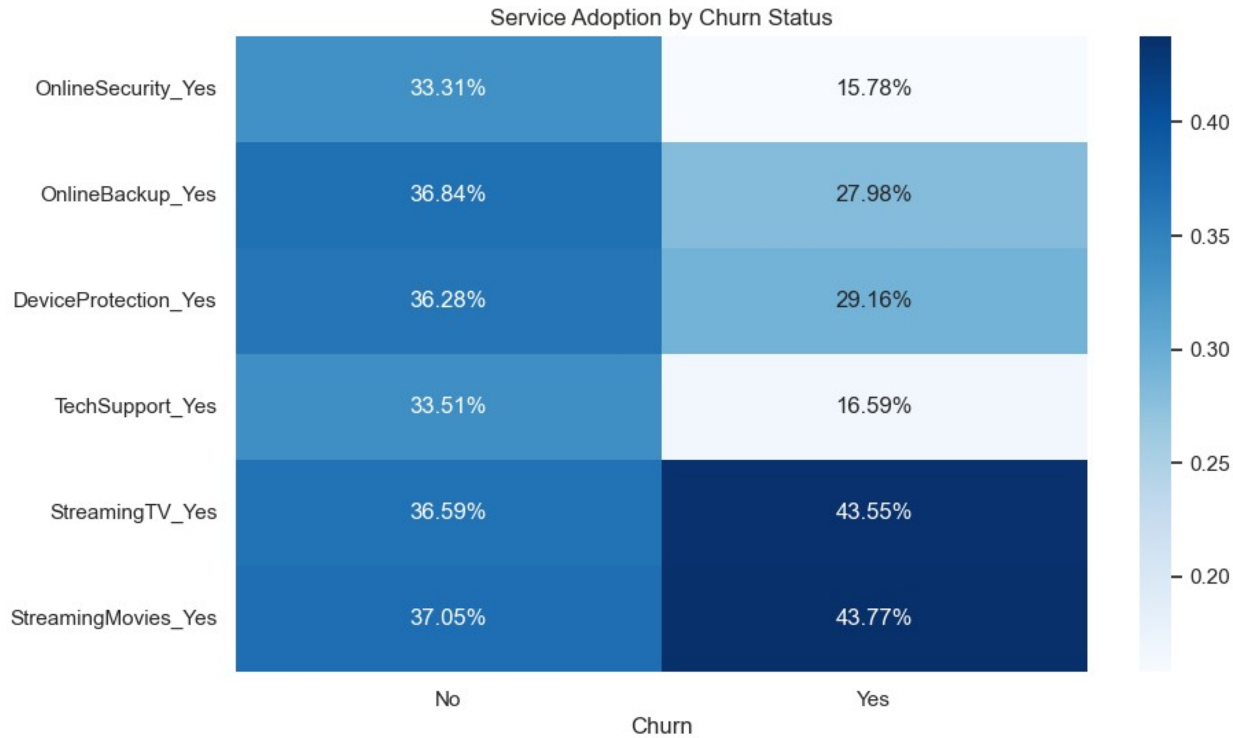
```
# First verify the actual columns
print("Current columns in DataFrame:", df.columns.tolist())

# Then adjust your categorical columns list to match EXACTLY
cat_cols = ['gender', 'MultipleLines', 'InternetService',
            'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
            'TechSupport', 'StreamingTV', 'StreamingMovies',
            'Contract', 'PaymentMethod']
```

```
Current columns in DataFrame: ['gender', 'SeniorCitizen', 'Partner',
'Dependents', 'tenure', 'PhoneService', 'MultipleLines',
'InternetService', 'OnlineSecurity', 'OnlineBackup',
'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies',
'Contract', 'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
'TotalCharges', 'Churn']
```

```
# Filter to only include columns that actually exist in the DataFrame
cat_cols = [col for col in cat_cols if col in df.columns]

# Now perform one-hot encoding
if cat_cols:  # Only run if there are columns to encode
    df = pd.get_dummies(df, columns=cat_cols, drop_first=True)
else:
    print("No matching categorical columns found - check your column
names")
```

Heatmap of Services

```
services = ['OnlineSecurity_Yes', 'OnlineBackup_Yes',
'DeviceProtection_Yes',
            'TechSupport_Yes', 'StreamingTV_Yes',
'StreamingMovies_Yes']
service_churn = df.groupby('Churn')[services].mean()

plt.figure(figsize=(10, 6))
sns.heatmap(service_churn.T, annot=True, cmap='Blues', fmt='.2%')
plt.title('Service Adoption by Churn Status')
plt.show()
```

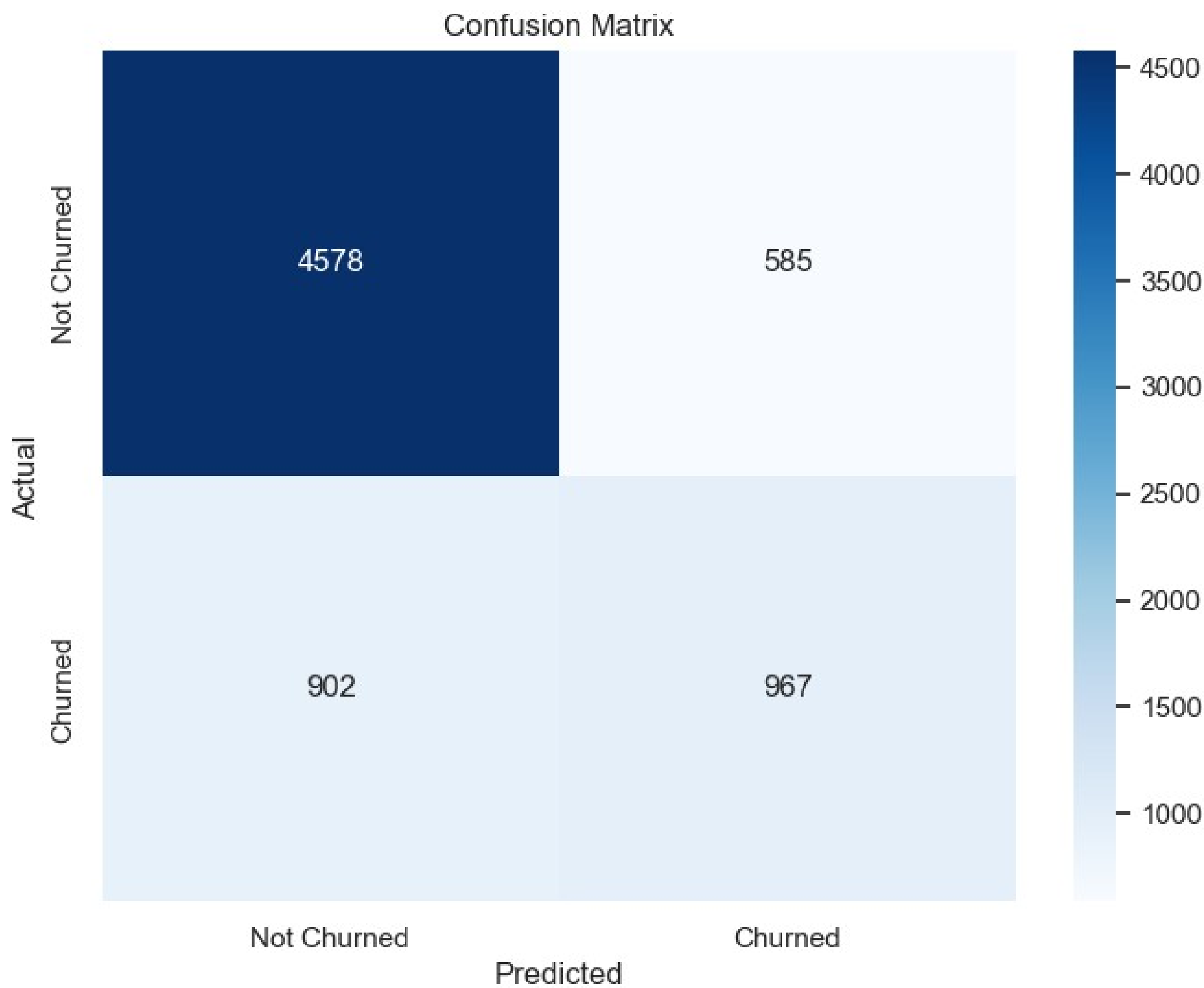


Key Drivers of Churn (Logistic Regression)

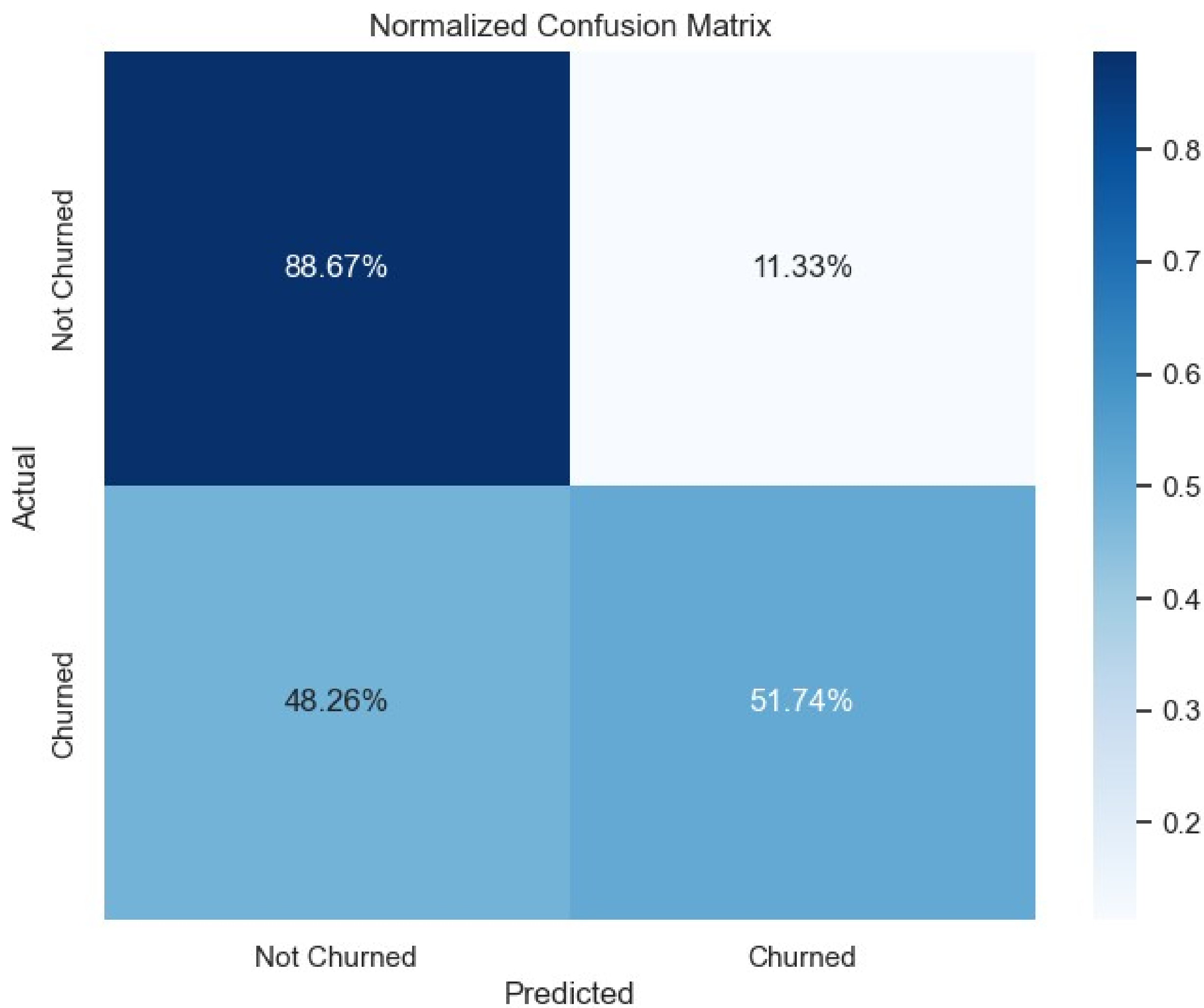
```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import LabelEncoder

# Prepare data
X = df.drop(['Churn', 'tenure', 'TotalCharges', 'TenureGroup'],
axis=1)
y = df['Churn']

# Convert binary categorical variables
X = X.replace({'Yes': 1, 'No': 0, 'Male': 1, 'Female': 0})

# For other categoricals, use one-hot encoding
X = pd.get_dummies(X, drop_first=True)

# Ensure all data is numeric
X = X.apply(pd.to_numeric, errors='coerce').dropna(axis=1)

# Now fit the model
model = LogisticRegression(max_iter=1000)
model.fit(X, y)

# Feature importance
importance = pd.DataFrame({'Feature': X.columns, 'Coefficient':
model.coef_[0]})
```

```
C:\Users\bhara\AppData\Local\Temp\ipykernel_193664\1751603280.py:9:
FutureWarning: Downcasting behavior in `replace` is deprecated and
will be removed in a future version. To retain the old behavior,
explicitly call `result.infer_objects(copy=False)`. To opt-in to the
future behavior, set `pd.set_option('future.no_silent_downcasting',
True)`
  X = X.replace({'Yes': 1, 'No': 0, 'Male': 1, 'Female': 0})
```

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Fit the model (assuming X and y are preprocessed)
model.fit(X, y)

# Predict on the same data (or test data if available)
y_pred = model.predict(X)

# Generate the confusion matrix
cm = confusion_matrix(y, y_pred)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```




```
cm_normalized = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]  # 
Normalize by row

plt.figure(figsize=(8, 6))
sns.heatmap(cm_normalized, annot=True, fmt='.2%', cmap='Blues',
            xticklabels=['Not Churned', 'Churned'],
            yticklabels=['Not Churned', 'Churned'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Normalized Confusion Matrix')
plt.show()
```


Normalized Confusion Matrix
88.67%
11.33%
48.26%
51.74%
Not Churned
Churned
Actual
Not Churned
Churned
Predicted
0.8
0.7
0.6
0.5
0.4
0.3
0.2

--- End ---