Dataset Introduction

Dataset Link: https://www.kaggle.com/datasets/blastchar/telco-customer-churn

This project uses the **Telco Customer Churn** dataset from Kaggle. The data was originally by **IBM** and has already been cleaned and labeled.

This project aims to:

- **perform** Exploratory Data Analysis for the dataset.
- **apply** data Preprocessing and Feature Engineering preparing the dataset for model training.
- **build a machine learning model** able to `predict customer churn` using their usage patterns, account information and demographics in order to help businesses take proactive retention measures.

Import Dependencies

We import all necessary libraries for data manipulation, visualization and analysis.

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

Importing the dataset

We import the dataset from Google Drive. We simply connect our colab file to Google Drive by clicking the Drive icon on the navigation bar at the left or by running the code:

from google.colab import drive
drive.mount('/content/drive')

```
dataset = ('/content/drive/MyDrive/Machine Learning Datasets/Customer Churn Prediction/Telco-Customer-Churn.csv')
dataset = pd.read_csv(dataset)
dataset.head(5)
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Dev
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	

5 rows × 21 columns

Dataset Overview

The dataset contains information about **telecommunication company customers** and whether they **churned** (left the company) in the last month.

Target Variable: `Churn` (`Yes` = Customer left, `No` = customer stayed)

Column Categories

1. **Target Variable**

- `Churn` — Indicates if the customer left within the last month.

2. **Services Signed Up**

- **Phone service:** `PhoneService`, `MultipleLines`
- **Internet services:** `InternetService`, `OnlineSecurity`, `OnlineBackup`, `DeviceProtection`, `TechSupport`, `StreamingTV`, `StreamingMovies`

3. **Customer Account Information**

- **Tenure:** `tenure` — how long they've been a customer (months)
- **Contract type:** `Contract`
- **Billing:** `PaperlessBilling`
- **Payment method:** `PaymentMethod`
- **Charges:** `MonthlyCharges`, `TotalCharges`

4. **Demographic Information**

- **Gender:** `gender`
- **Family status:** `Partner`, `Dependents`

```
print(f"The dataset has {dataset.shape[0]:,} rows and {dataset.shape[1]:,} columns.")
dataset.head(5)
```

```
The dataset has 7,043 rows and 21 columns.
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	Dev
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	

5 rows × 21 columns

Data Types and Missing Values

Inspecting column data types and identifying missing values for cleaning and preprocessing.

```
dataset.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
```

```
dataset.isnull().sum()
```

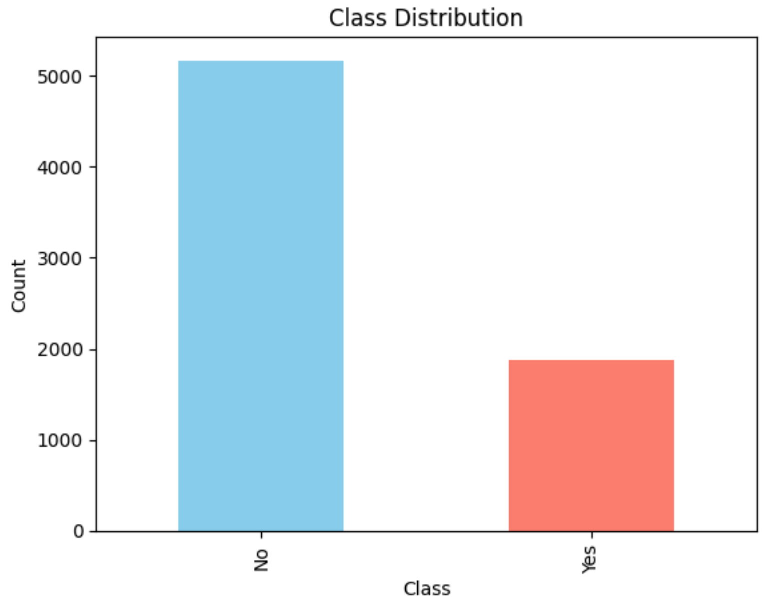
	0
customerID	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	0
Churn	0

dtype: int64

Class Distribution

We check for class imbalance. If class count differ significantly a possible use case scenario for SMOTE might be needed later on to normalize the class distribution.

```
dataset['Churn'].value_counts().plot(kind='bar', color=['skyblue', 'salmon'])
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Count')
plt.show()
```



Statistical Summary

We use the .describe() method to obtain a statistical summary of the numerical features. This includes measures such as mean, standard deviation, and quartiles, which help assess the central tendency and spread of data.

```
dataset.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

Unique Values for Categorical Features

Checking different categorical values in each columns. It helps in planning encoding strategies later on in the process.

```
cat_cols = dataset.select_dtypes(include=['object']).columns
for col in cat_cols:
    print(f"{col}: {dataset[col].unique()}\n")
```

```
customerID: ['7590-VHVEG' '5575-GNVDE' '3668-QPYBK' ... '4801-JZAZL' '8361-LTMKD'
 '3186-AJIEK']

gender: ['Female' 'Male']

Partner: ['Yes' 'No']

Dependents: ['No' 'Yes']

PhoneService: ['No' 'Yes']

MultipleLines: ['No phone service' 'No' 'Yes']

InternetService: ['DSL' 'Fiber optic' 'No']

OnlineSecurity: ['No' 'Yes' 'No internet service']
```

```
OnlineBackup: ['Yes' 'No' 'No internet service']

DeviceProtection: ['No' 'Yes' 'No internet service']

TechSupport: ['No' 'Yes' 'No internet service']

StreamingTV: ['No' 'Yes' 'No internet service']

StreamingMovies: ['No' 'Yes' 'No internet service']

Contract: ['Month-to-month' 'One year' 'Two year']

PaperlessBilling: ['Yes' 'No']

PaymentMethod: ['Electronic check' 'Mailed check' 'Bank transfer (automatic)'
 'Credit card (automatic)']

TotalCharges: ['29.85' '1889.5' '108.15' ... '346.45' '306.6' '6844.5']

Churn: ['No' 'Yes']
```

Observation: We can observe that on the given list of Categorical columns `TotalCharges` was included. This could mean that the values were stored as an object string instead of a numeric type. In order to solve this we simply convert the whole column into numeric type.

Converting `TotalCharges` into numeric type

We simply convert the whole column by using *`pd.to_numeric`*.

```
print("Initial column type:", dataset['TotalCharges'].dtypes)
dataset['TotalCharges'] = pd.to_numeric(dataset['TotalCharges'], errors='coerce')
print("Total NaN values:", dataset['TotalCharges'].isna().sum())
```

```
Initial column type: object
Total NaN values: 11
```

```
dataset['TotalCharges'] = dataset['TotalCharges'].fillna(0)
print("Final column type:", dataset['TotalCharges'].dtypes)
print("Total NaN values:", dataset['TotalCharges'].isna().sum())
```

```
Final column type: float64
Total NaN values: 0
```

Exploratory Data Analysis

Numerical Feature Distribution

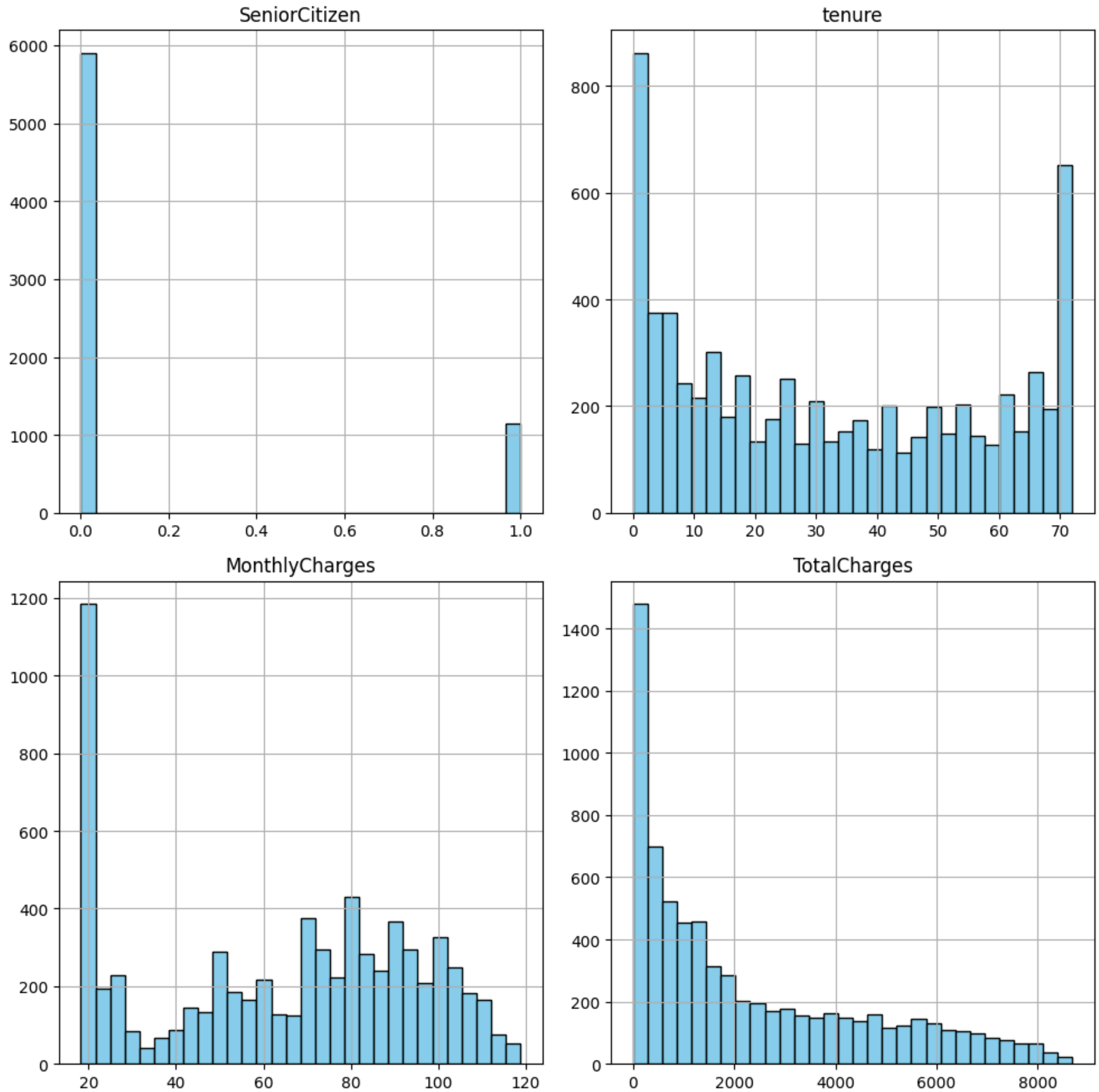
Visualizing numerical features such as tenure, MonthlyCharges, and TotalCharges helps us understand:

- the range and spread of each variable.
- presence of outliers.
- General customer spending and tenure patterns.

This step is useful for spotting potential skewness and deciding whether transformations (e.g., log-scaling) might be needed.

```
num_cols = dataset.select_dtypes(include=['int64', 'float64']).columns

dataset[num_cols].hist(bins=30, figsize=(10, 10), color='skyblue', edgecolor='black')
plt.tight_layout()
plt.show()
```



Categorical Feature Distributions

Examine the distribution of categorical features such as `Contract`, `InternetService`, and `PaymentMethod`. This will show us:

- the most common customer service type
- whether any categories are rate and might need combining or special handling.

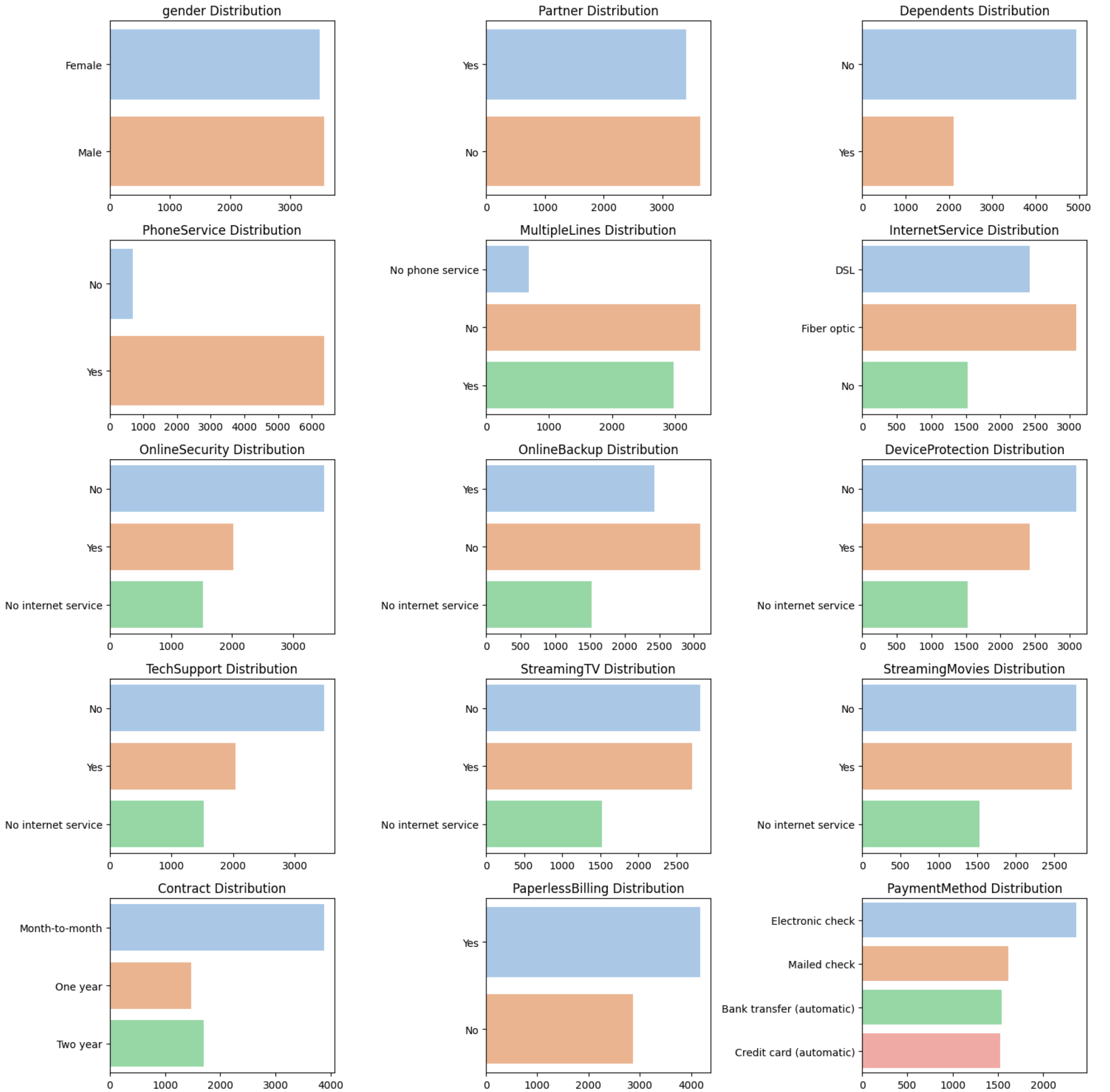
```
cat_cols = dataset.select_dtypes(include=[object]).columns.drop(['Churn', 'customerID'])

fig, axes = plt.subplots(nrows=(len(cat_cols) + 2) // 3, ncols=3, figsize=(15, 15))
axes = axes.flatten()

for i, col in enumerate(cat_cols):
    sns.countplot(y=col, data=dataset, ax=axes[i], palette='pastel', hue=col, legend=False)
    axes[i].set_title(f'{col} Distribution')
    axes[i].set_ylabel('')
    axes[i].set_xlabel('')

for j in range(len(cat_cols), len(axes)):
    fig.delaxes(axes[j])
```

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


gender Distribution
Female
Male
Partner Distribution
Yes
No
Dependents Distribution
No
Yes
PhoneService Distribution
No
Yes
MultipleLines Distribution
No phone service
No
Yes
InternetService Distribution
DSL
Fiber optic
No
OnlineSecurity Distribution
No
Yes
No internet service
OnlineBackup Distribution
Yes
No
No internet service
DeviceProtection Distribution
No
Yes
No internet service
TechSupport Distribution
No
Yes
No internet service
StreamingTV Distribution
No
Yes
No internet service
StreamingMovies Distribution
No
Yes
No internet service
Contract Distribution
Month-to-month
One year
Two year
PaperlessBilling Distribution
Yes
No
PaymentMethod Distribution
Electronic check
Mailed check
Bank transfer (automatic)
Credit card (automatic)

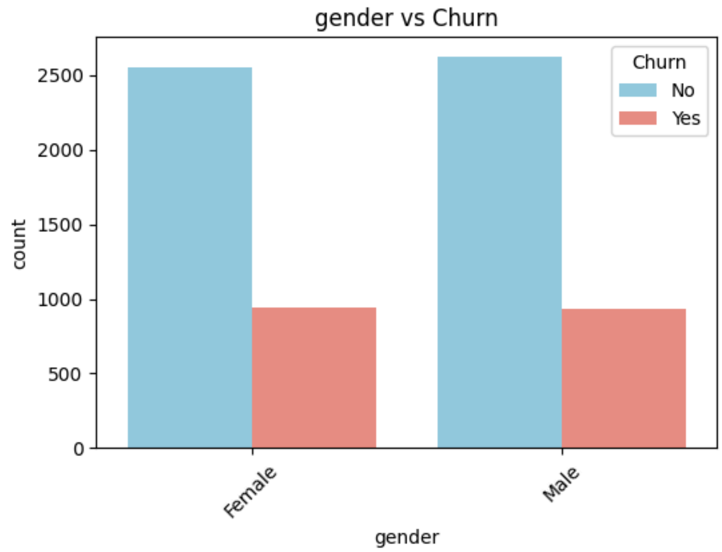
Target Variable vs Feature correlation

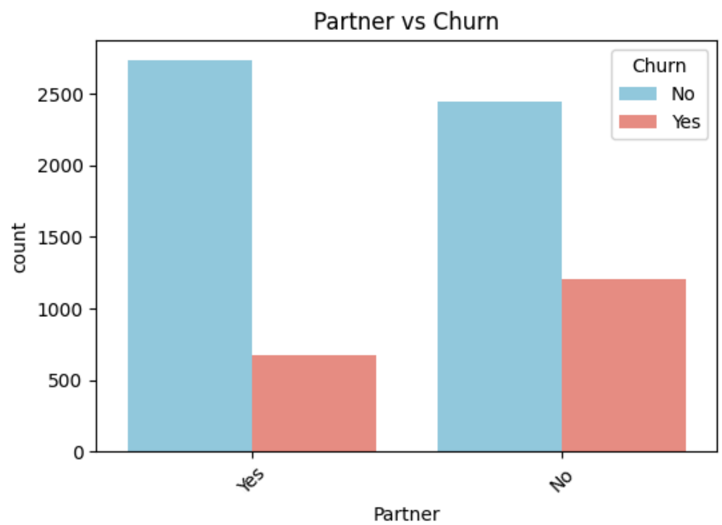
We now explore how each feature relates to churn.

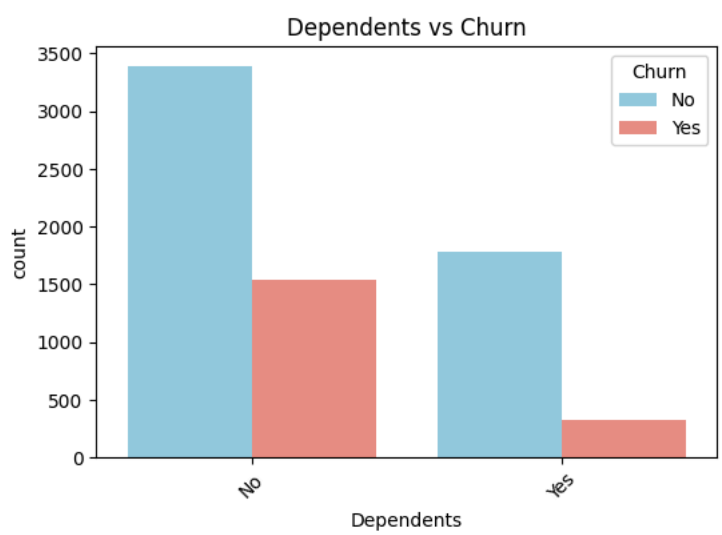
- For categorical variables: Compare category frequencies between churned and non-churned customers.
- For numerical variables: Compare value distributions using boxplots.

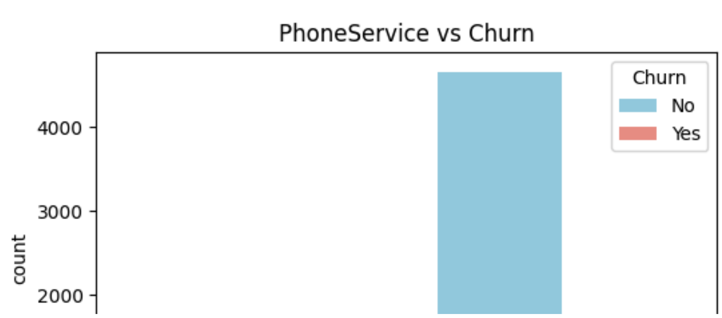
This helps identify key churn drivers, such as whether month-to-month contracts or high monthly charges are linked to higher churn rates.

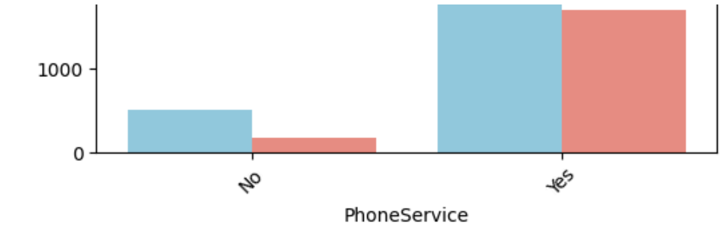
```
for col in cat_cols:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, hue='Churn', data=dataset, palette=['skyblue','salmon'])
    plt.title(f'{col} vs Churn')
    plt.xticks(rotation=45)
    plt.show()
```

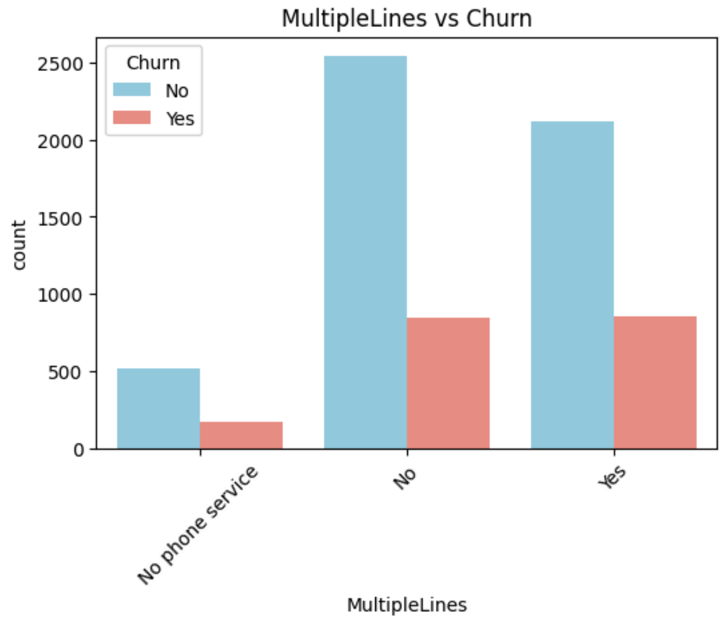

gender vs Churn
Churn
No
Yes
count
0
500
1000
1500
2000
2500
Female
Male
gender

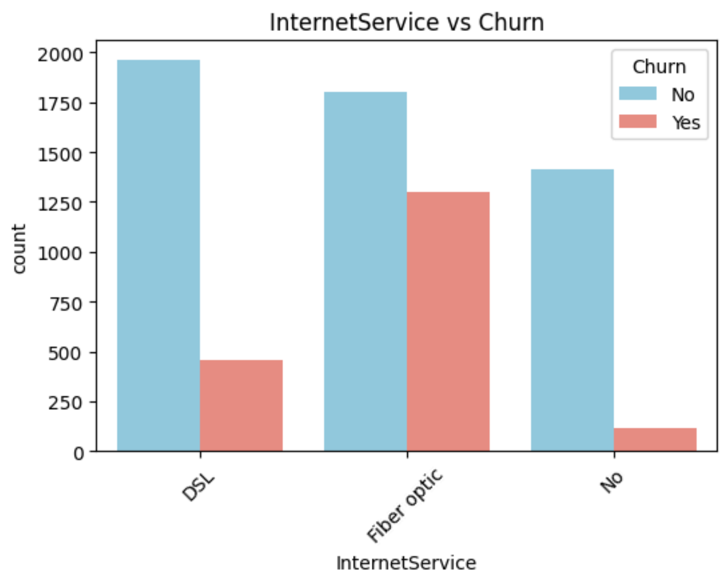

Partner vs Churn
Churn
No
Yes
count
0
500
1000
1500
2000
2500
Yes
No
Partner

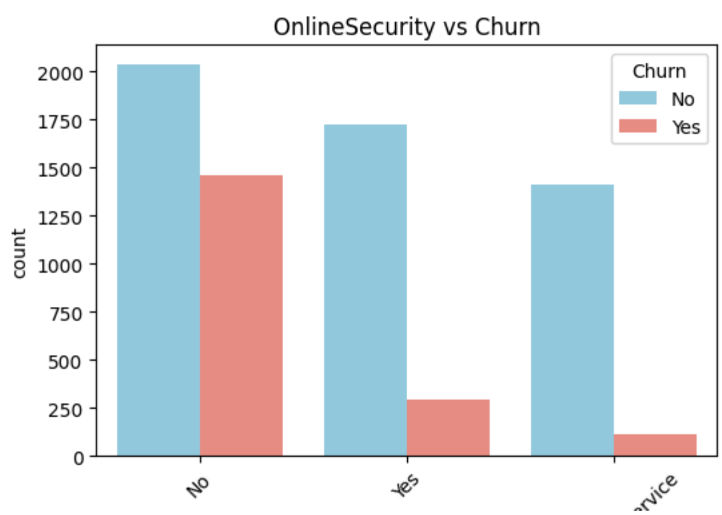

Dependents vs Churn
Churn
No
Yes
count
0
500
1000
1500
2000
2500
3000
3500
No
Yes
Dependents

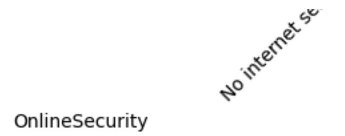

PhoneService vs Churn
Churn
No
Yes
count
2000
3000
4000

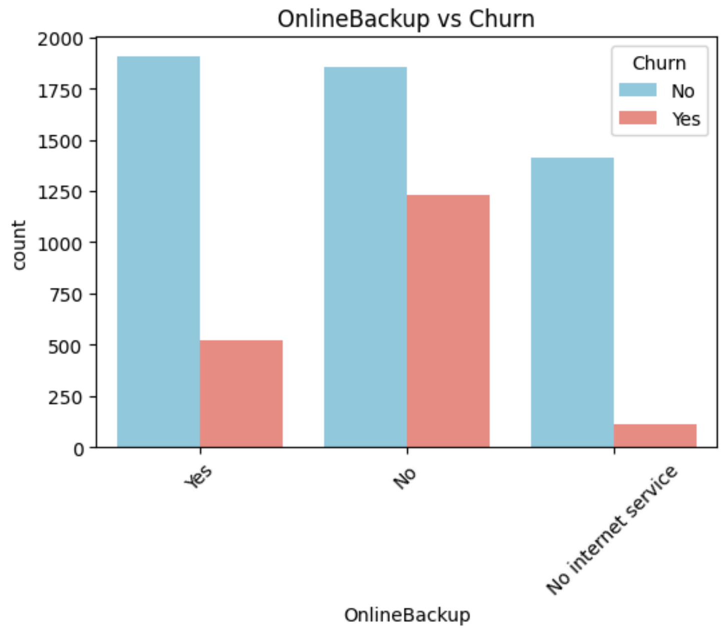

1000
0
No
Yes
PhoneService

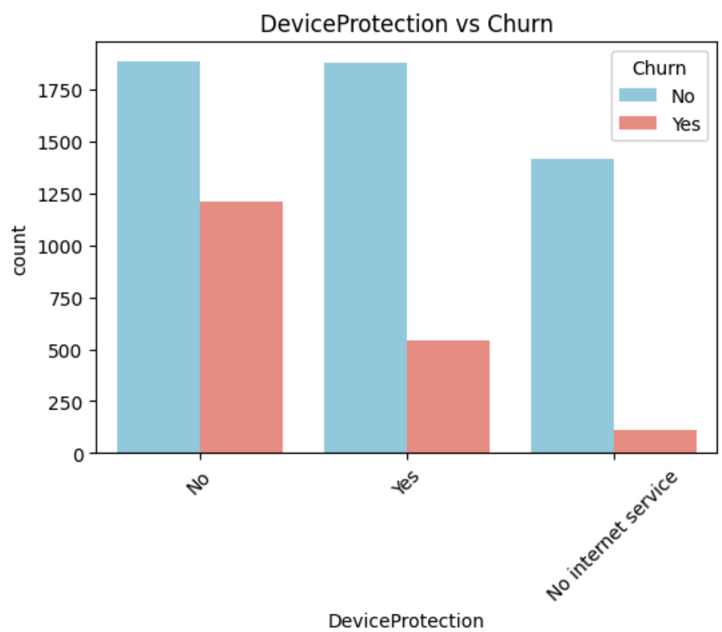

MultipleLines vs Churn
Churn
No
Yes
count
2500
2000
1500
1000
500
0
No phone service
No
Yes
MultipleLines

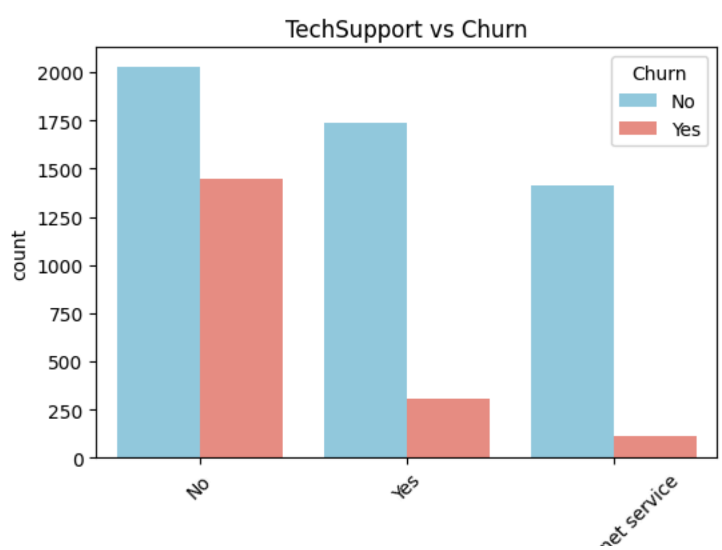

InternetService vs Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
DSL
Fiber optic
No
InternetService

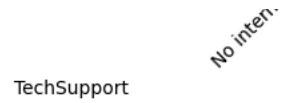

OnlineSecurity vs Churn
Churn
No
Yes
count
2000
1750
1500
1250
1000
750
500
250
0
No
Yes

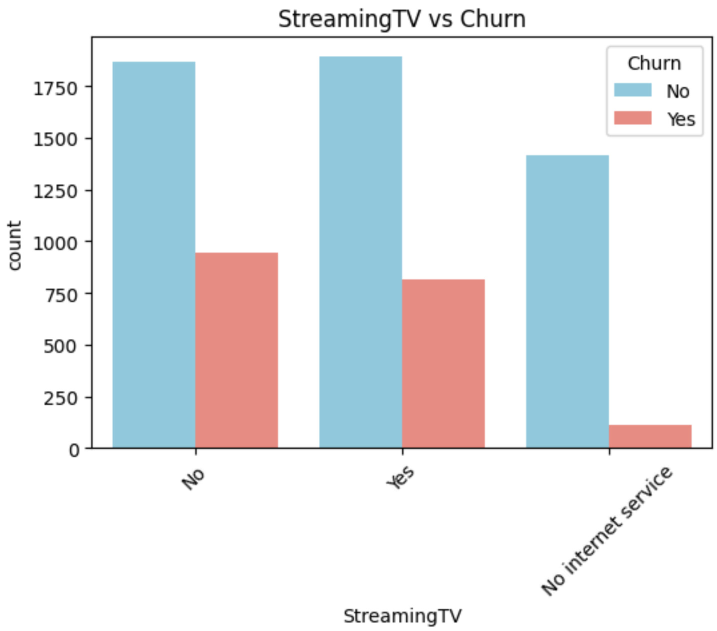

No internet se
OnlineSecurity

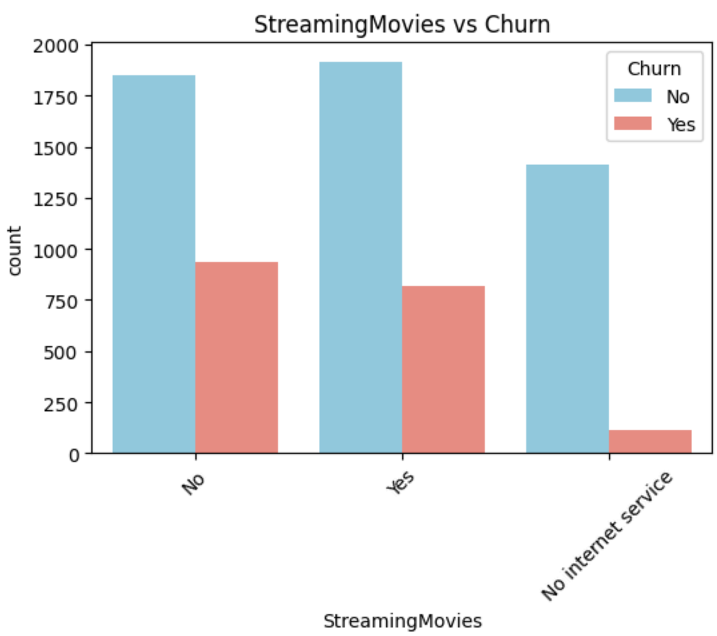

OnlineBackup vs Churn
count
0
250
500
750
1000
1250
1500
1750
2000
Churn
No
Yes
Yes
No
No internet service
OnlineBackup

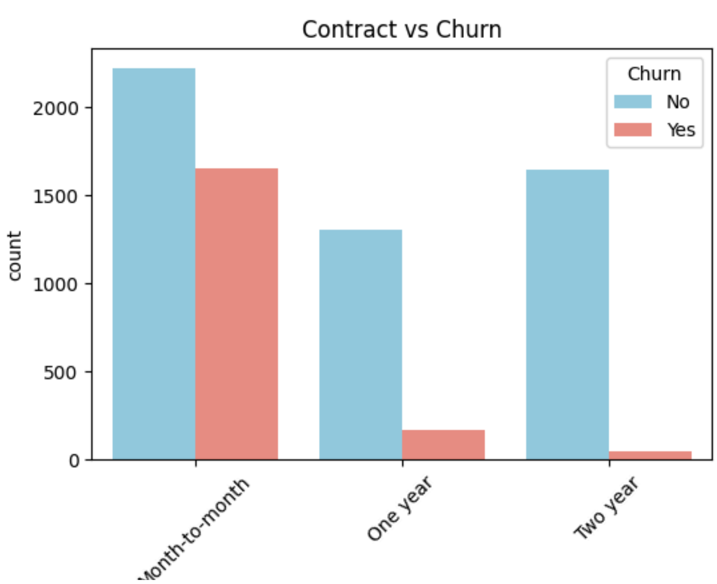

DeviceProtection vs Churn
count
0
250
500
750
1000
1250
1500
1750
Churn
No
Yes
No
Yes
No internet service
DeviceProtection


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

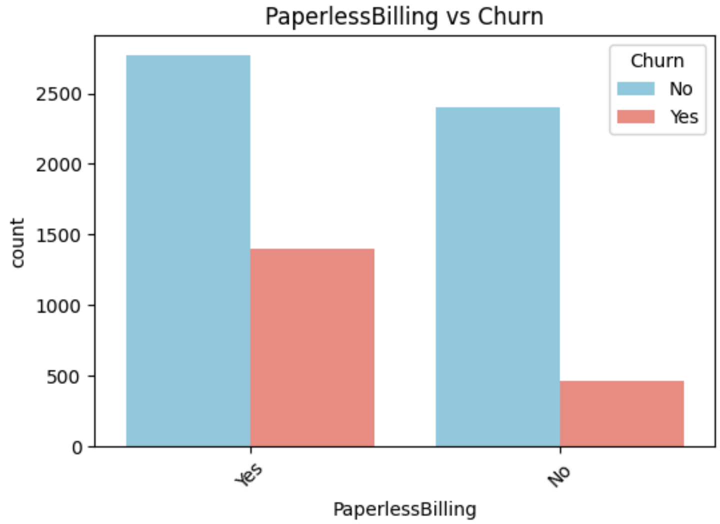

No intern
TechSupport

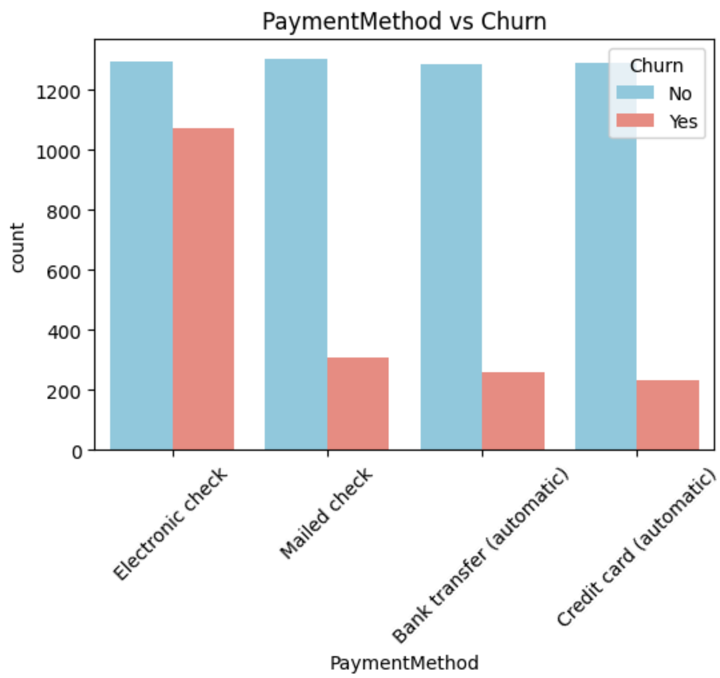

StreamingTV vs Churn
Churn
No
Yes
count
0
250
500
750
1000
1250
1500
1750
No
Yes
No internet service
StreamingTV


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


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

Contract


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


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

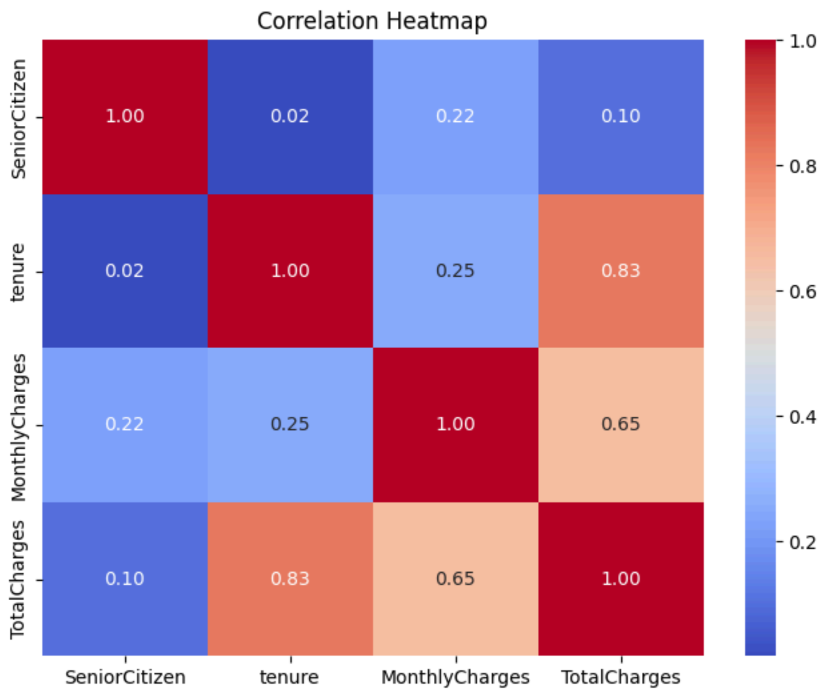
Correlation Analysis

We now check the relationship between numerical variable using correlation heatmap. This helps:

- Identify redundant features (high correlation, rule of thumb: `|correlation| > 0.85` consider dropping one feature).
- Spot variables with potential predictive power for churn.

For example, `MonthlyCharges` and `TotalCharges` might be strongly correlated, meaning we should be cautious about multicollinearity.

```
plt.figure(figsize=(8,6))
corr = dataset[num_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap')
plt.show()
```



Data Processing

We know proceed to **Data Processing** ensuring the data is clean, numeric where necessary and ready for ML models.

```
dataset = dataset.drop('customerID', axis=1)
dataset.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	Device
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	

Encoding Categorical Variables

Machine learning models require numerical input, so categorical features must be transformed into numeric format.

In this dataset, we have three types of categorical features:

1. **Binary (Yes/No)** — Converted directly to `0` and `1`.
2. **Ordinal (ordered)** — Categories have a natural order, encoded using **Ordinal Encoding**.
3. **Nominal (unordered)** — No inherent order, encoded using **One-Hot Encoding** to create separate binary columns for each category.

This ensures the data is in a suitable format for all models, avoids misinterpretation of category order, and preserves model flexibility.

Encoding Binary Yes/No Columns

```
binary_map = {'Yes': 1, 'No': 0}
binary_cols = ['Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']

for col in binary_cols:
    dataset[col] = dataset[col].map(binary_map)


dataset['gender'] = dataset['gender'].map({'Male': 1, 'Female': 0})
```

Encoding Ordinal Columns

```
from sklearn.preprocessing import OrdinalEncoder

ordinal_cols = ['Contract']
ordinal_order = [['Month-to-month', 'One year', 'Two year']]
dataset[ordinal_cols] = OrdinalEncoder(categories=ordinal_order).fit_transform(dataset[ordinal_cols])
```

One-hot encoding Nominal Columns

```
nominal_cols = ['InternetService', 'PaymentMethod', 'MultipleLines',
    'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
    'TechSupport', 'StreamingTV', 'StreamingMovies']

dataset = pd.get_dummies(dataset, columns=nominal_cols, drop_first=True)
```

```
dataset.head()
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Contract	PaperlessBilling	MonthlyCharges	TotalCharges	...	Onlin
0	0	0	1	0	1	0	0.0	1	29.85	29.85	...	
1	1	0	0	0	34	1	1.0	0	56.95	1889.50	...	
2	1	0	0	0	2	1	0.0	1	53.85	108.15	...	
3	1	0	0	0	45	0	1.0	0	42.30	1840.75	...	
4	0	0	0	0	2	1	0.0	1	70.70	151.65	...	

5 rows × 30 columns

Checking Numerical Feature Distributions

Before proceeding and deciding on a scaling method, it's important to check the distribution of numerical features.

This helps us see:

- If features are **roughly normally distributed** → `StandardScaler` is a good choice.
- If features are **skewed** → we may need `MinMaxScaler` or transformations (e.g., log).
- If features have **extreme outliers** → `RobustScaler` may be better.

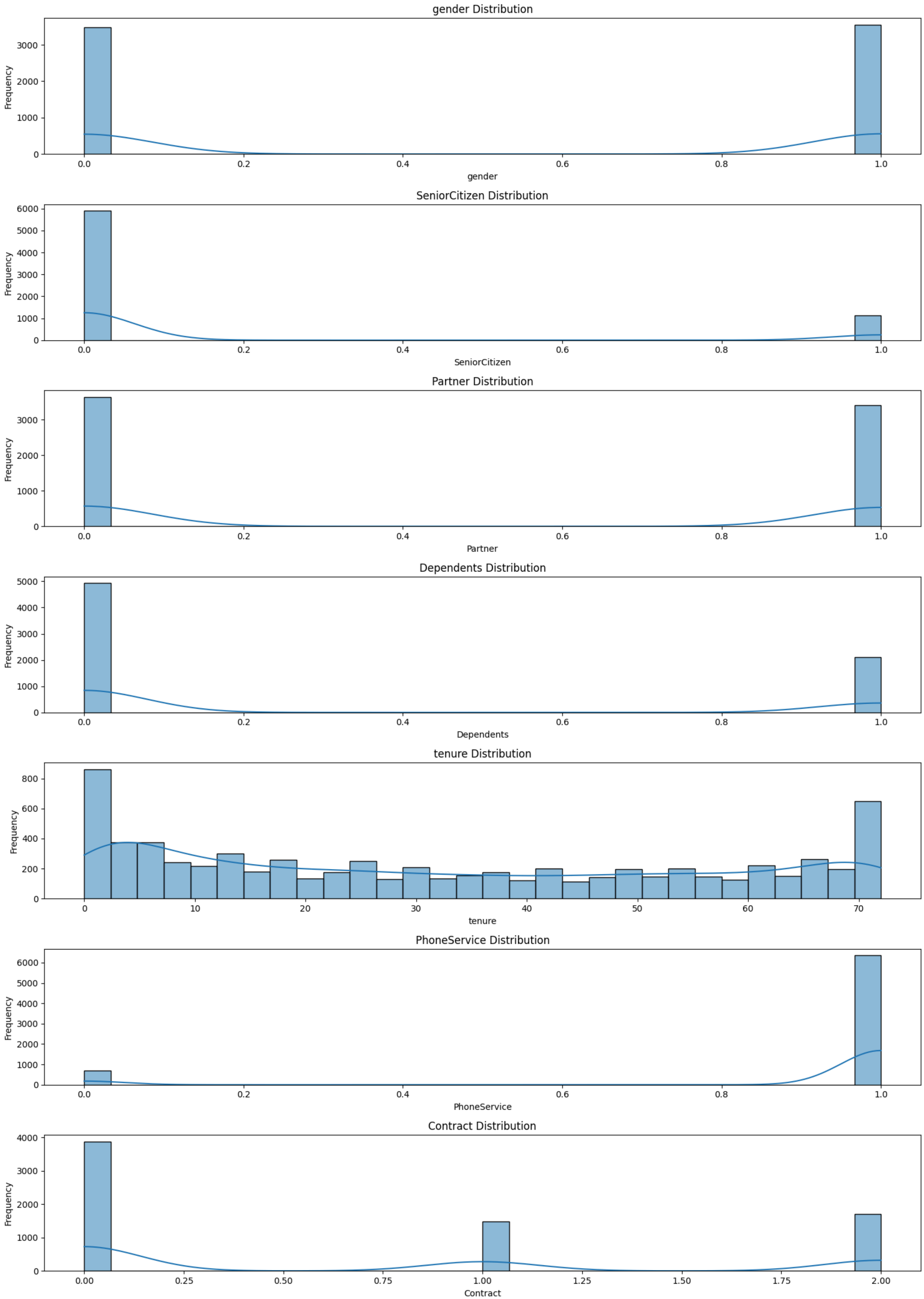
We'll plot histograms for each numeric feature to visualize their distributions.

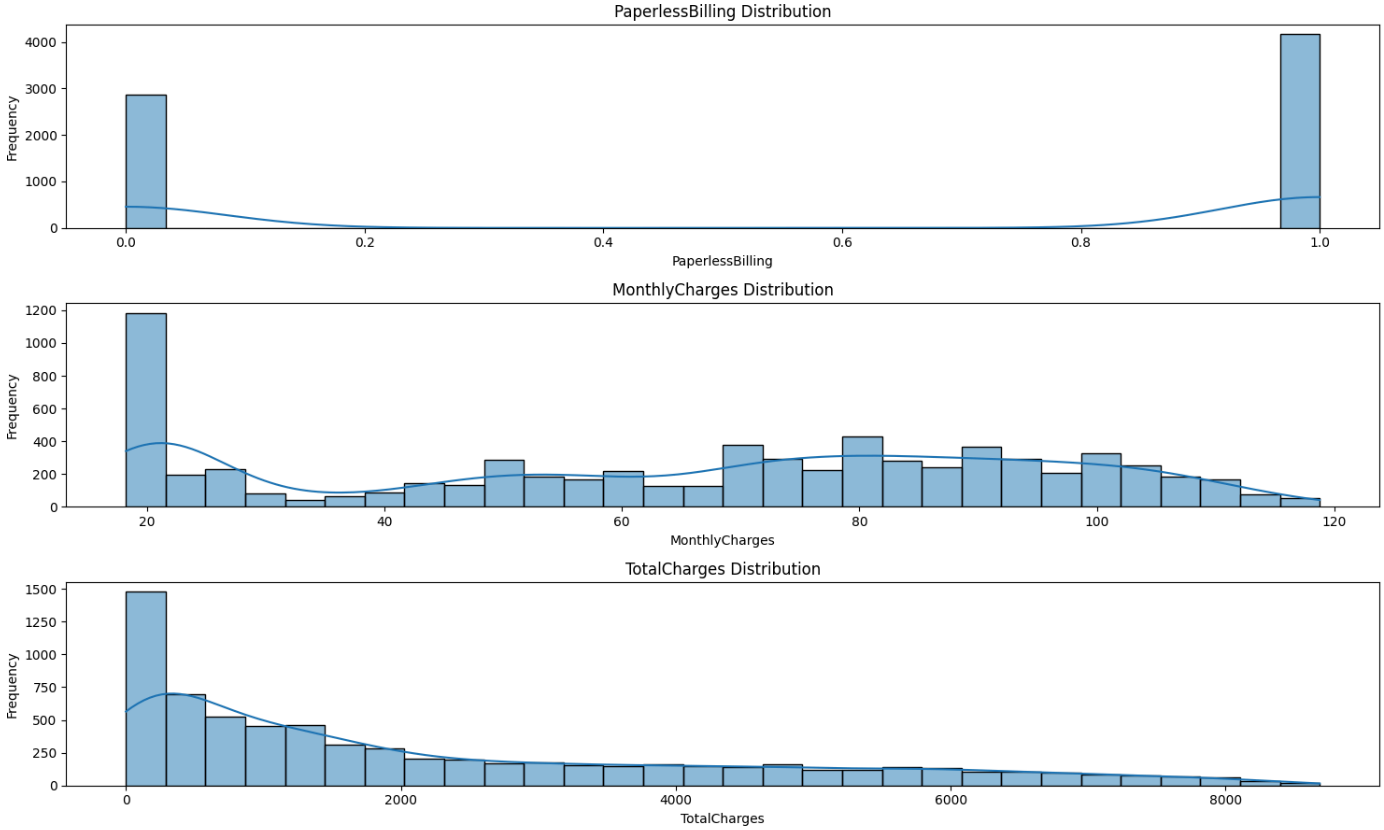
```
numeric_features = dataset.select_dtypes(include=['int64', 'float64']).columns.drop(['Churn'])

plt.figure(figsize=(15, len(numeric_features) * 3))
```

```
for i, col in enumerate(numeric_features, 1):
  plt.subplot(len(numeric_features), 1, i)
  sns.histplot(dataset[col], kde=True, bins=30)
  plt.title(f'{col} Distribution')
  plt.xlabel(col)
  plt.ylabel('Frequency')

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


gender Distribution
Frequency
gender
SeniorCitizen Distribution
Frequency
SeniorCitizen
Partner Distribution
Frequency
Partner
Dependents Distribution
Frequency
Dependents
tenure Distribution
Frequency
tenure
PhoneService Distribution
Frequency
PhoneService
Contract Distribution
Frequency
Contract


PaperlessBilling Distribution
Frequency
PaperlessBilling
MonthlyCharges Distribution
Frequency
MonthlyCharges
TotalCharges Distribution
Frequency
TotalCharges