Telco Customer Churn Analysis

Name: Ronald Wanjohi Gachoka **Student Number:** S2274825 **Course:** Software Development for Data Science **Assignment:** Coursework 2 **Program:** MSc Financial Technology

---

1. Business Understanding

Customer churn is the rate at which customers stop doing business with a specific organization (Frankenfield, 2022). This is is especially important for company's as it essentially measures a company's ability to retain customers. Its helpful to not only look at the churn rate as what will be identified in this analysis but also compare it to the company's growth rate (which will not be covered in this project). This is because if the churn rate is higher than the growth rate then the company is losing customers faster than it is gaining them.

In the competitive space of telecommunications (telco) companies, customer churn is a key metric that companies use to evaluate their performance as it impacts their profitability and brand recognition (Wagh, et al., 2024). According to Wagh, et al., (2024), he states that in some cases, the cost of acquiring a new customer is 5x more expensive than retaining an existing customer. This is why it is important for companies to identify the key factors that lead to customers churning. Several factors can lead to a business churning customers such as high prices, poor customer service, poor product quality and much more.

Despite this project not being able to identify the exact reason why customers churn such as an increase in prices or a competitor offering new services, it will try identify the key factors that lead to customers churning. This will be done by building a model that will predict whether a customer will churn or not.

1.1. Objectives

The main objective of this coursework project is to build a model that can identify customers who are at risk of churning. This will be done by answering the following questions:

- What is the churn rate of the company?
- What is the average tenure of customers?
- What is the average monthly charges of customers?
- What are the key factors that dictate whether a customer will churn or not?
- How do churn rates vary across different customer demographics?

1.2. Success Criteria

1. Successfully predict whether a customer will churn or not with an accuracy of 85% or higher on unseen data.
2. Identify the key factors that dictate whether a customer will churn or not.
3. Aim for a precision score of 0.80 or higher to reduce the number of false positives.
4. Aim for a recall score of 0.80 or higher to reduce the number of false negatives.

Methodology To achieve these goals, the coursework will involve the following steps which is in line with the CRISP-DM methodology (Smart Vision, 2020):

- Business Understanding
- Data Understanding
- Data Preparation
- Modelling
- Evaluation

For this coursework we will not deploy the model, but we will evaluate the model and make recommendations based on the results.

2. Data Understanding

The dataset we've been provided with is a Telco customer churn data from California, USA. The data is available at Telco Customer Churn - Kaggle and has 7,043 customers with multiple features from `OnlineSecurity`, `Contract` type, `Charges`, `Dependents` amongst many more. The target variable is the `Churn` column which is a binary variable with Yes or No values.

This section will include:

- Data Collection
- Data Description
- Data Exploration
- Data Quality

Load Libraries

```
# Data Analysis
import pandas as pd
import numpy as np

# Data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as msno

# Data Preprocessing
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.utils.class_weight import compute_class_weight

# Data Modelling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# Data Evaluation
from sklearn.metrics import RocCurveDisplay, confusion_matrix,
classification_report
```

```
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score

# Data Sampling
from imblearn.over_sampling import SMOTE

# Styling & Settings
sns.set_style("whitegrid")
# pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
%matplotlib inline
```

Load Data

```
df = pd.read_csv("Telco-Customer-Churn.csv")
print(df.shape)
df.head()
```

```
(7043, 21)
```

```
   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 OnlineBackup  \
0  No phone service             DSL             No          Yes
1                No             DSL            Yes           No
2                No             DSL            Yes          Yes
3  No phone service             DSL            Yes           No
4                No     Fiber optic             No           No

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

```
4                No          No          No                No  Month-to-
month

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

  Churn
0    No
1    No
2   Yes
3    No
4   Yes
```

We can see a high-level overview of our data and what each of the columns may contain. We can also see that there are 7,043 rows and 21 columns.

Summary of the data

```
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
```

We can see that all columns have the same number of non-null values which is 7,043. This means that there are no missing values in the dataset.

We can also see `TotalCharges` is an object type and not a numerical type. We will need to convert this to a numerical type in the next section.

```
# Summary statistics (numerical)
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
```

Based of the above summary statistics we can see the we do not have any inacurrate outliers in our dataset as the minimum and maximum values are within the expected range

```
# Summary statistics (categorical)
df.describe(include=['O'])
```

```
        customerID gender Partner Dependents PhoneService
MultipleLines  \
count         7043   7043    7043       7043         7043
7043
unique        7043      2       2          2            2
3
top     7590-VHVEG   Male      No         No          Yes
No
freq             1   3555    3641       4933         6361
3390

       InternetService OnlineSecurity OnlineBackup DeviceProtection  \
count             7043           7043         7043             7043
unique               3              3            3                3
top        Fiber optic             No           No               No
```

```
freq                3096           3498         3088              3095

       TechSupport StreamingTV StreamingMovies        Contract  \
count         7043        7043            7043            7043
unique           3           3               3               3
top             No          No              No  Month-to-month
freq          3473        2810            2785            3875

       PaperlessBilling     PaymentMethod TotalCharges Churn
count              7043              7043         7043  7043
unique                2                 4         6531     2
top                 Yes  Electronic check                 No
freq               4171              2365           11  5174
```

Data Description

Below is a description of each column in the dataset. This will help us understand what each column represents and what type of data it contains.

Column Name	Description	Type	Subtype
Senior Citizen	Whether the customer is a senior citizen or not	Numerical	Binary
Tenure	Number of months the customer has stayed with the company	Numerical	Discrete
Monthly Charges	The amount charged to the customer monthly	Numerical	Continuous
CustomerID	Customer ID	Categorical	Nominal
Gender	Is the customer male or female	Categorical	Nominal
Partner	Whether the customer has a partner or not	Categorical	Nominal
Dependents	Whether the customer has dependents or not	Categorical	Nominal
Phone Service	Whether the customer has a phone service or not	Categorical	Nominal
Multiple Lines	Whether the customer has multiple lines or not	Categorical	Nominal
Internet Service	Customer's internet	Categorical	Nominal

Column Name	Description	Type	Subtype
	service provider (DSL, Fiber optic, No)		
Online Security	Whether the customer has online security or not	Categorical	Nominal
Online Backup	Whether the customer has online backup or not	Categorical	Nominal
Device Protection	Whether the customer has device protection or not	Categorical	Nominal
Tech Support	Whether the customer has tech support or not	Categorical	Nominal
Streaming TV	Whether the customer has streaming TV or not	Categorical	Nominal
Streaming Movies	Whether the customer has streaming movies or not	Categorical	Nominal
Contract	The contract term of the customer (Month-to-month, One year, Two year)	Categorical	Nominal
Paperless Billing	Whether the customer has paperless billing or not	Categorical	Nominal
Payment Method	The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))	Categorical	Nominal
Total Charges	The total amount charged to the customer	Numerical	Continuous
Churn	Whether the customer churned or not (Yes or No)	Categorical	Nominal

We can see that we have 2 columns that have incorrect data types:

- `SeniorCitizen` should be a categorical variable
- `TotalCharges` should be a numerical variable

Data Exploration

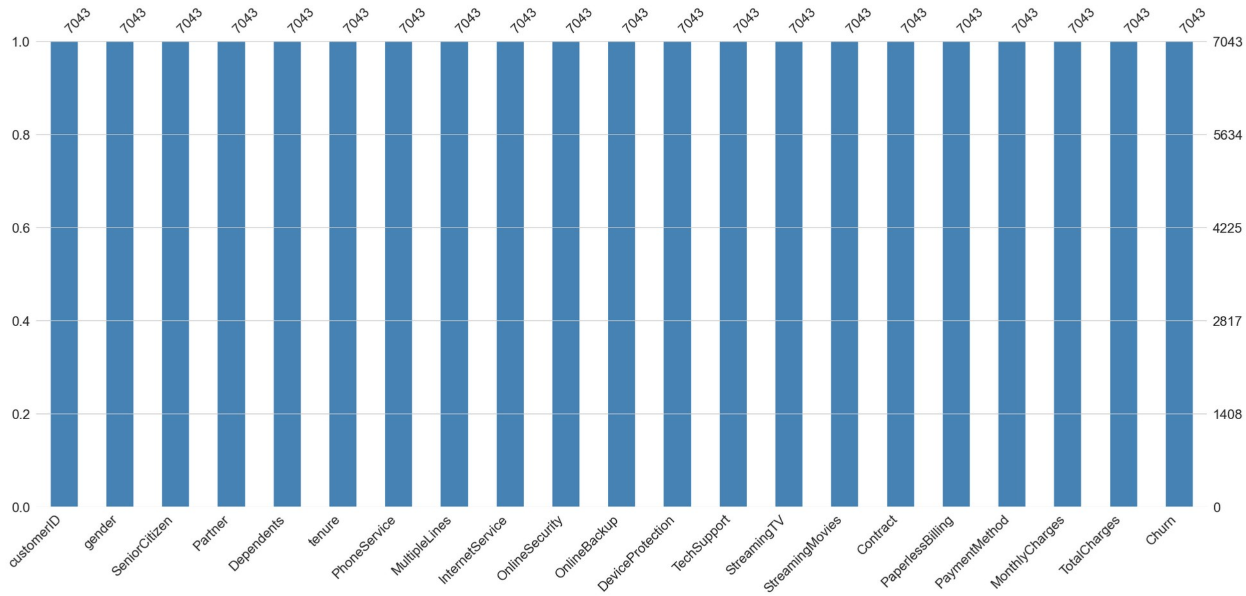
In this section we will explore the data to get a better understanding of the data and identify any issues with the data. We will also look at the distribution of the data and identify any outliers.

Missing Values

```
df.isna().sum()
```

```
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
```

```
msno.bar(df, color="steelblue");
```



From the above chart we can see that there are no missing values in the dataset. However, we can see that there are 11 missing values in the `TotalCharges` column. This is most likely because the column is an object type and not a numerical type. We will convert this column to a numerical type in the next section.

Duplicates

```
df.duplicated().any().sum()
```

```
0
```

```
df['customerID'].duplicated().sum()
```

```
0
```

Based of the 2 code cells above we can see that no row in our data has duplicated values. In addition we did a further check to make sure there are no duplicated customerIDs which is the unique identifier for each customer. This is good as we don't have to worry about removing any duplicated rows.

Data Type Consistency

```
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["TotalCharges"] = pd.to_numeric(df["TotalCharges"],
errors='coerce')
df["TotalCharges"].dtype
```

```
dtype('float64')
```

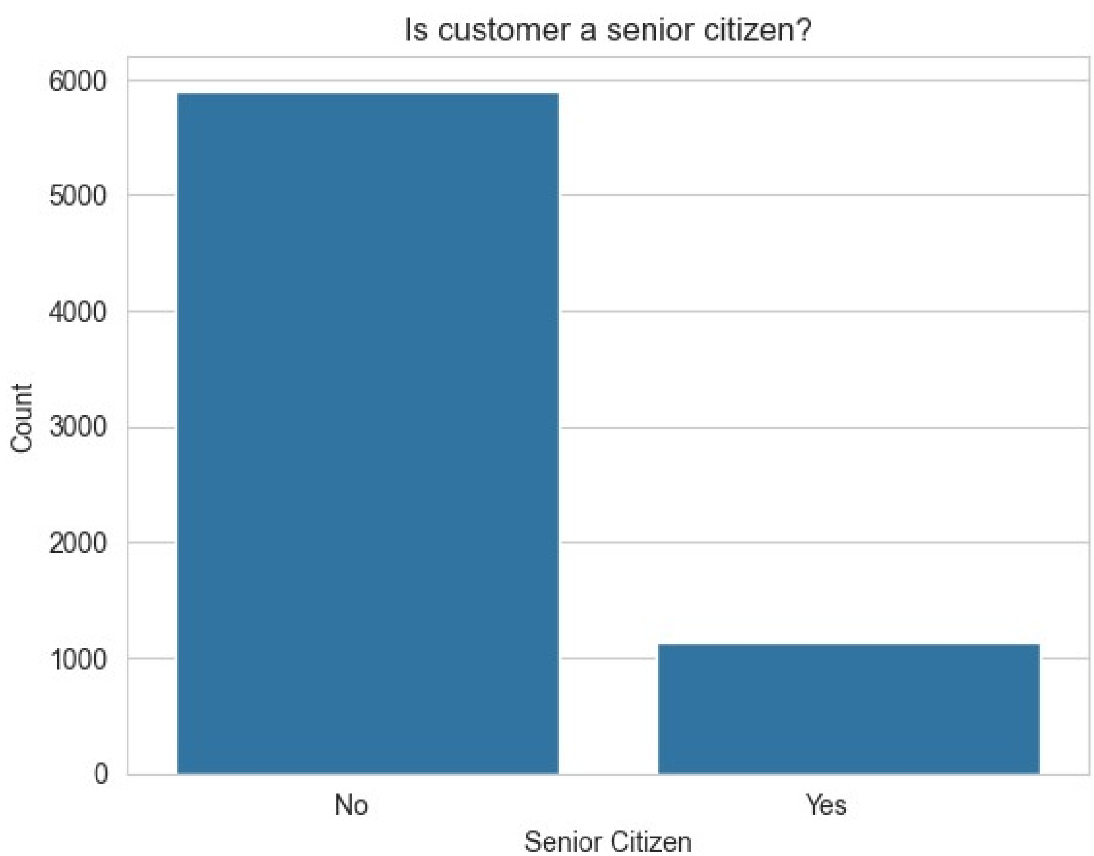
Univariate Analysis

```
def plot_categorical(df:pd.DataFrame, col:str, title:str, xlabel:str,
ylabel:str, txt_rotation:int=0):
    """
    Function to plot categorical data

    :param df: Name of the dataframe
    :param col: Column of intereset
    :param title:  Title of the plot
    :param xlabel: X label of the plot
    :param ylabel: Y label of the plot
    """

    # Map binary variables to Yes or No
    mapping = {0: 'No', 1: 'Yes'}
    if df[col].dtype == 'int64':
        df[col] = df[col].map(mapping)

    # Print summary stats
    print("Data Statistics:")
    print(df[col].value_counts())

    # sns.countplot(x=df[col], data=df)
    sns.barplot(x=df[col].value_counts().index,
y=df[col].value_counts())
    plt.title(title)
    plt.xlabel(xlabel)
    plt.ylabel(ylabel)
    plt.xticks(rotation=txt_rotation, ha="right")
```

```
    plt.show();
```

Senior Citizen

```
plot_categorical(df=df, col="SeniorCitizen", title="Is customer a senior citizen?",
                 xlabel="Senior Citizen", ylabel="Count")

Data Statistics:
SeniorCitizen
No     5901
Yes    1142
Name: count, dtype: int64
```

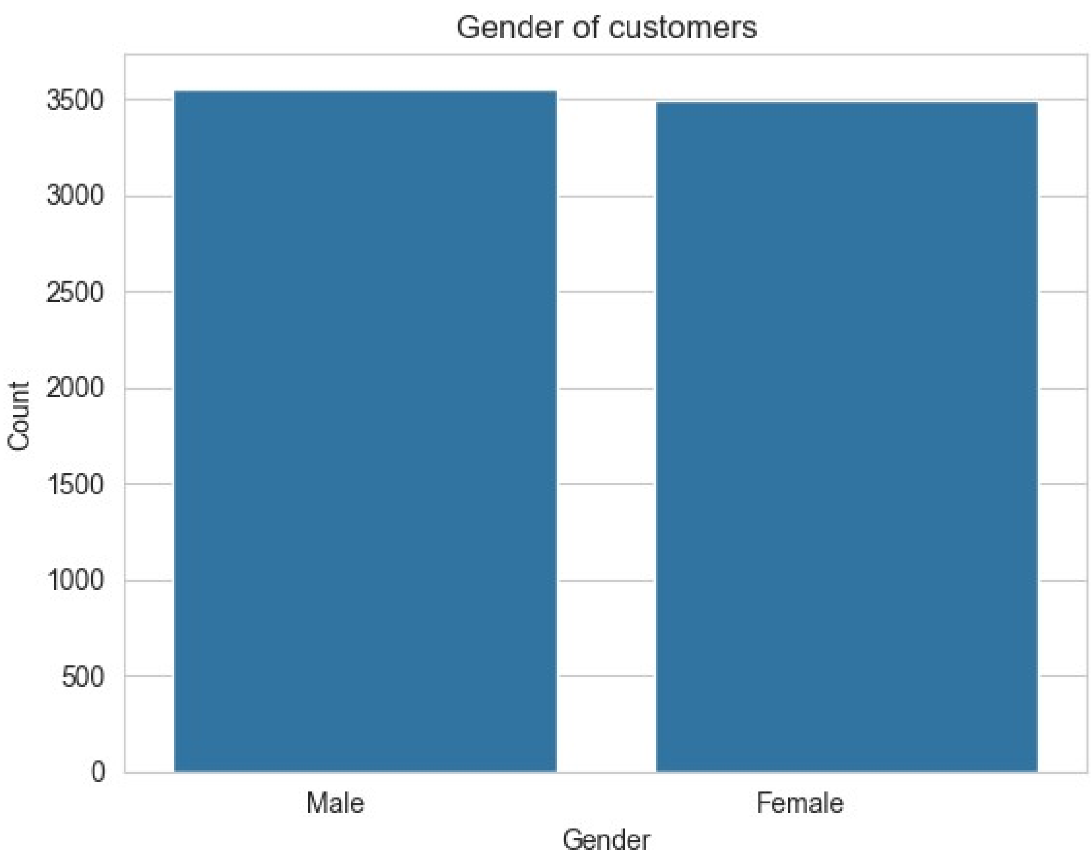


Within our dataset, we have slightly over 1,000 senior citizens and just about 6,000 non-senior citizens. This shows that the majority of the telco's customers are not senior citizens which is good for their business as they can effectively capture the younger market.

Gender

```
plot_categorical(df=df, col="gender", title="Gender of customers",
xlabel="Gender", ylabel="Count")
```

```
Data Statistics:
gender
Male      3555
Female    3488
Name: count, dtype: int64
```

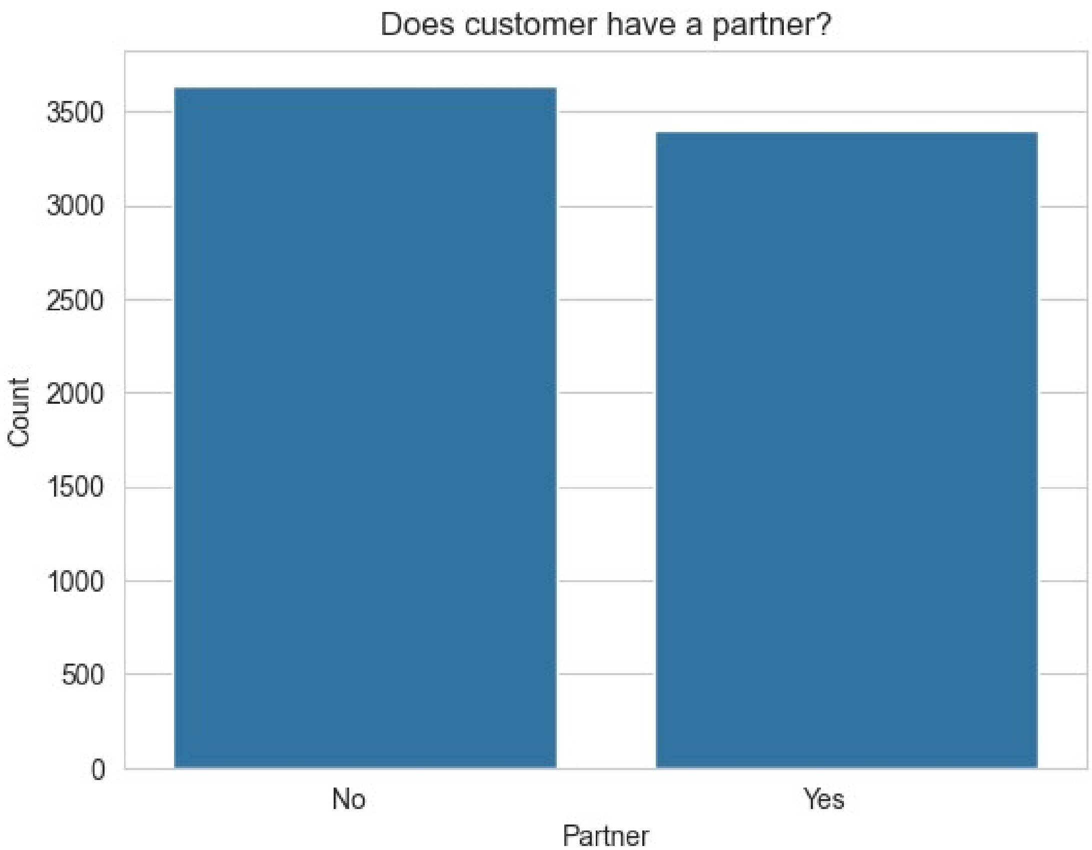


The dataset is very close to being evenly split between male and female customers at 3,500.

Partner

```
plot_categorical(df=df, col="Partner", title="Does customer have a partner?", xlabel="Partner", ylabel="Count")
```

```
Data Statistics:
Partner
No     3641
Yes    3402
Name: count, dtype: int64
```

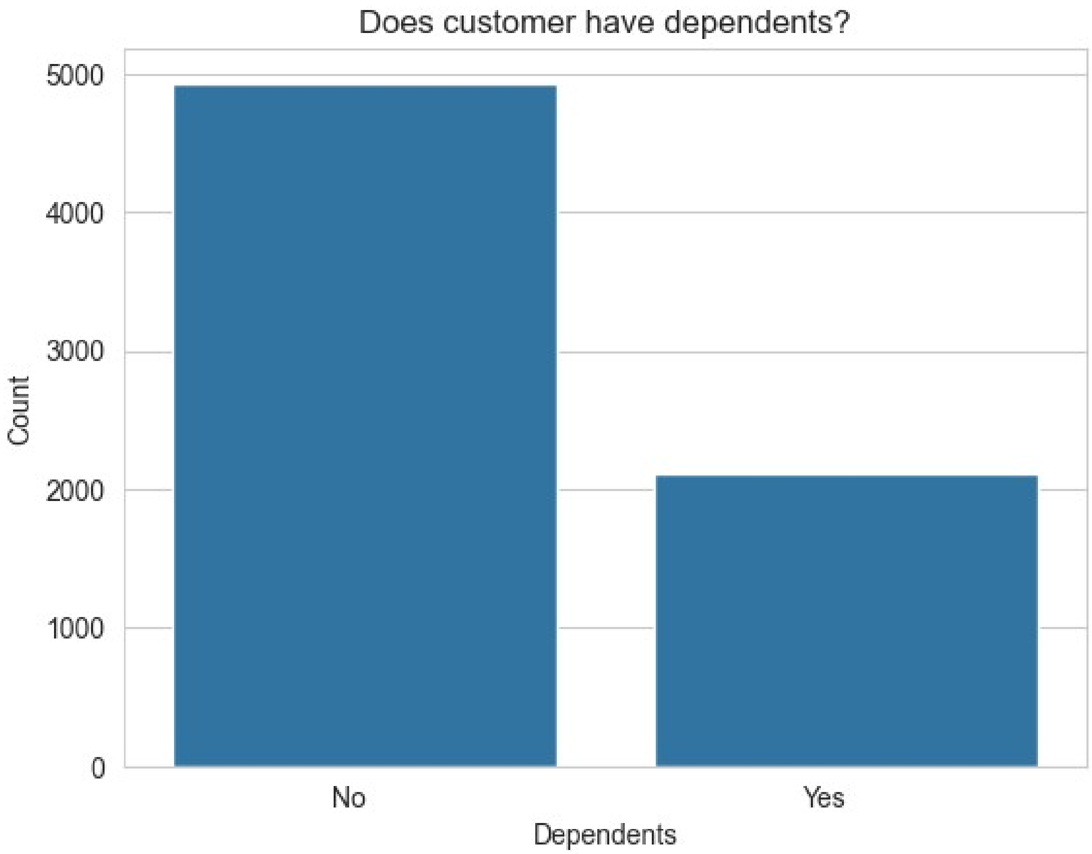


The dataset is also very close to being evenly split between customers who have a partner and those who don't.

Dependents

```
plot_categorical(df=df, col="Dependents", title="Does customer have dependents?",
                 xlabel="Dependents", ylabel="Count")
```

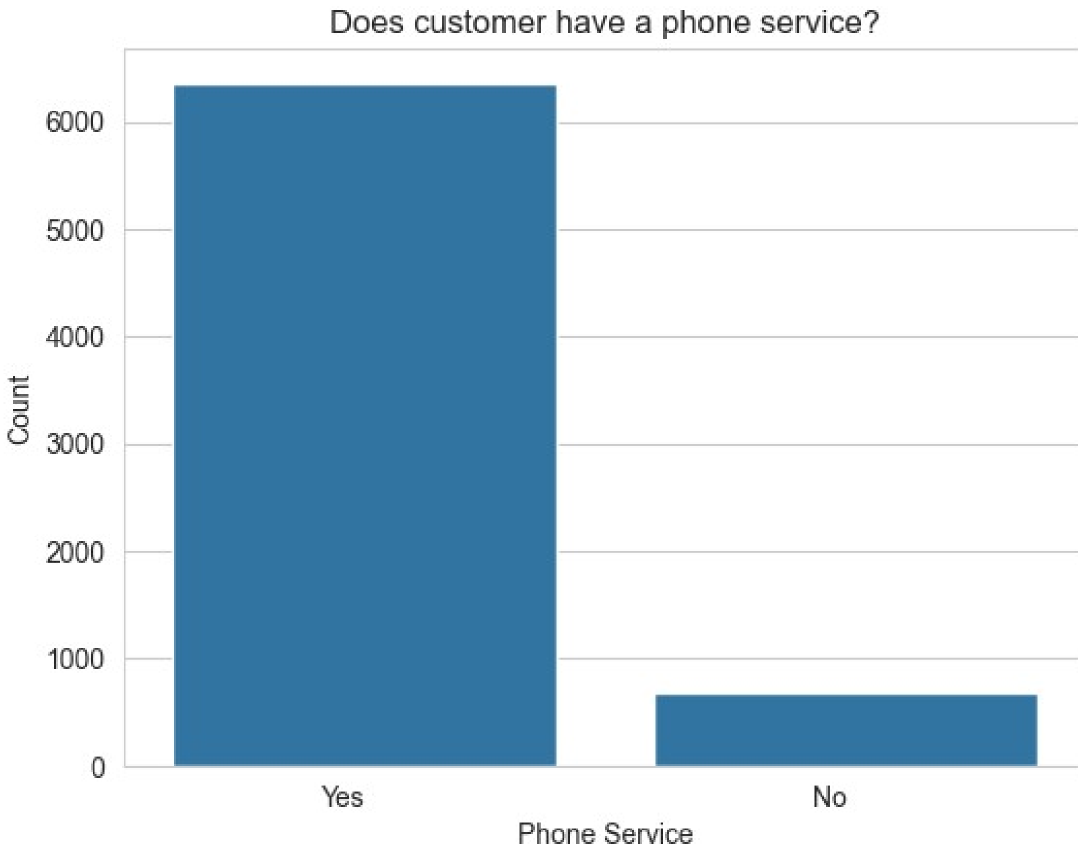
```
Data Statistics:
Dependents
No     4933
Yes    2110
Name: count, dtype: int64
```



Most of the customers do not have dependents which could be closely linked to the low number of senior citizens in the dataset.

Phone Service

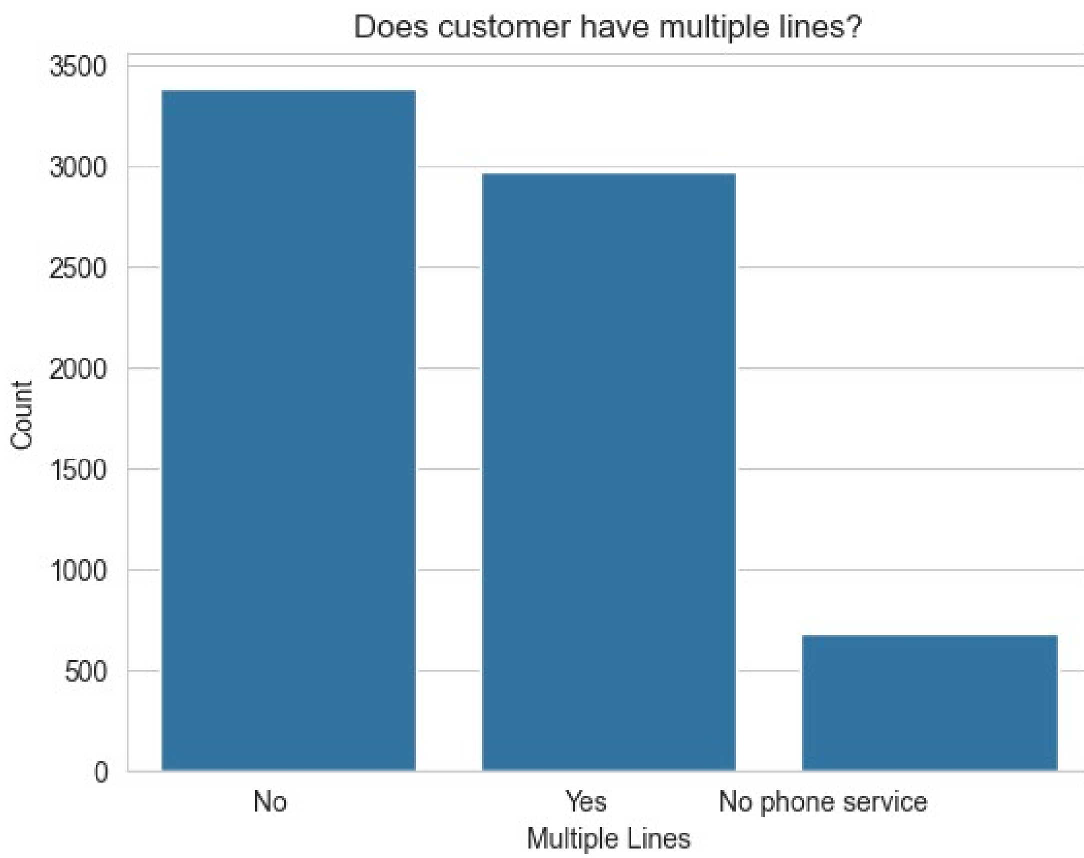
```
plot_categorical(df=df, col="PhoneService", title="Does customer have a phone service?",
                 xlabel="Phone Service", ylabel="Count")

Data Statistics:
PhoneService
Yes    6361
No      682
Name: count, dtype: int64
```



The majority of the customers have a phone service which is good for the telco as they can offer them other services such as Internet service, device protection or other services. The disparity between the customers who have a phone service and those who don't is quite large which was to be expected.

Multiple Lines

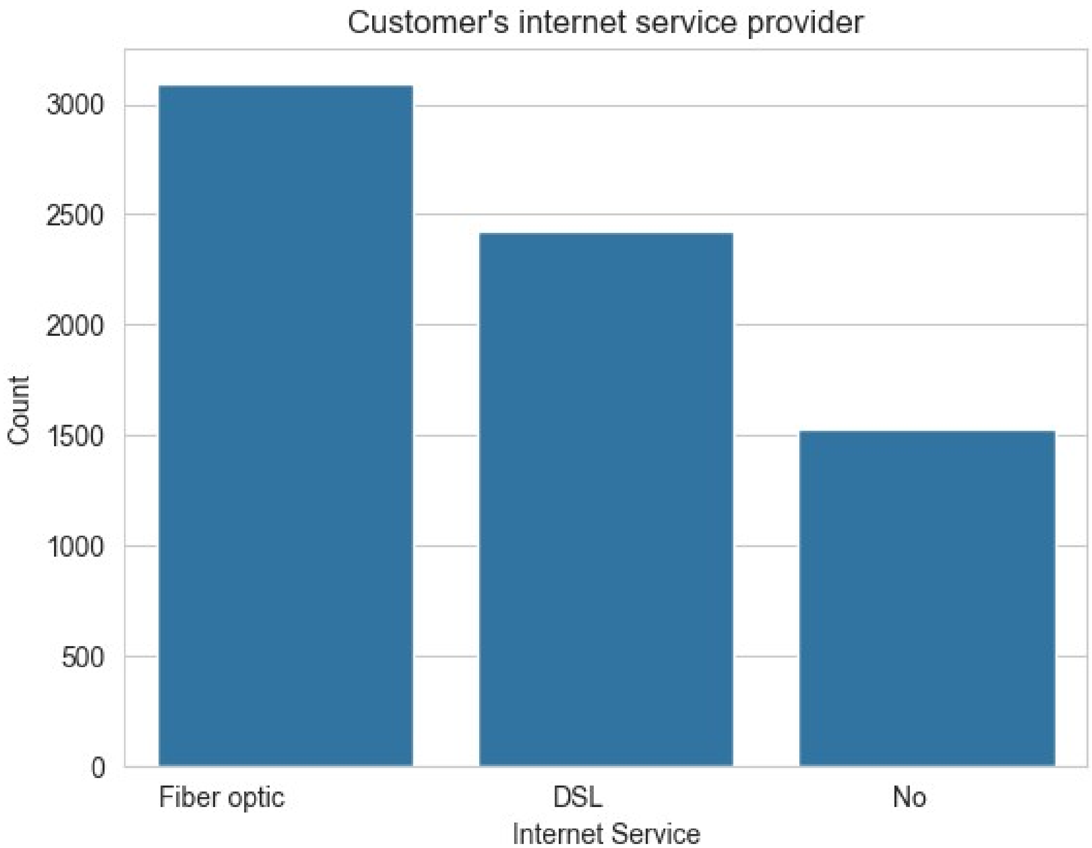
```
plot_categorical(df=df, col="MultipleLines", title="Does customer have multiple lines?",
                 xlabel="Multiple Lines", ylabel="Count")

Data Statistics:
MultipleLines
No                  3390
Yes                 2971
No phone service     682
Name: count, dtype: int64
```



The majority of the customers do not have multiple lines which is to be expected as most customers will usually have only one line. However, what's interesting is that nearly 3,000 customers have multiple lines which is a large number. In addition, over 500 customers have no phone service which was unexpected.

Internet Service

```
plot_categorical(df=df, col="InternetService", title="Customer's internet service provider",
                 xlabel="Internet Service", ylabel="Count")

Data Statistics:
InternetService
Fiber optic    3096
DSL            2421
No             1526
Name: count, dtype: int64
```



The majority of the customers have `Fiber Optic` internet service, this could be due to higher internet speeds or lower prices. The number of customers who have `DSL` is also quite high at just about 2,500 which is good for competition and maintaining a healthy market share.

Online Security

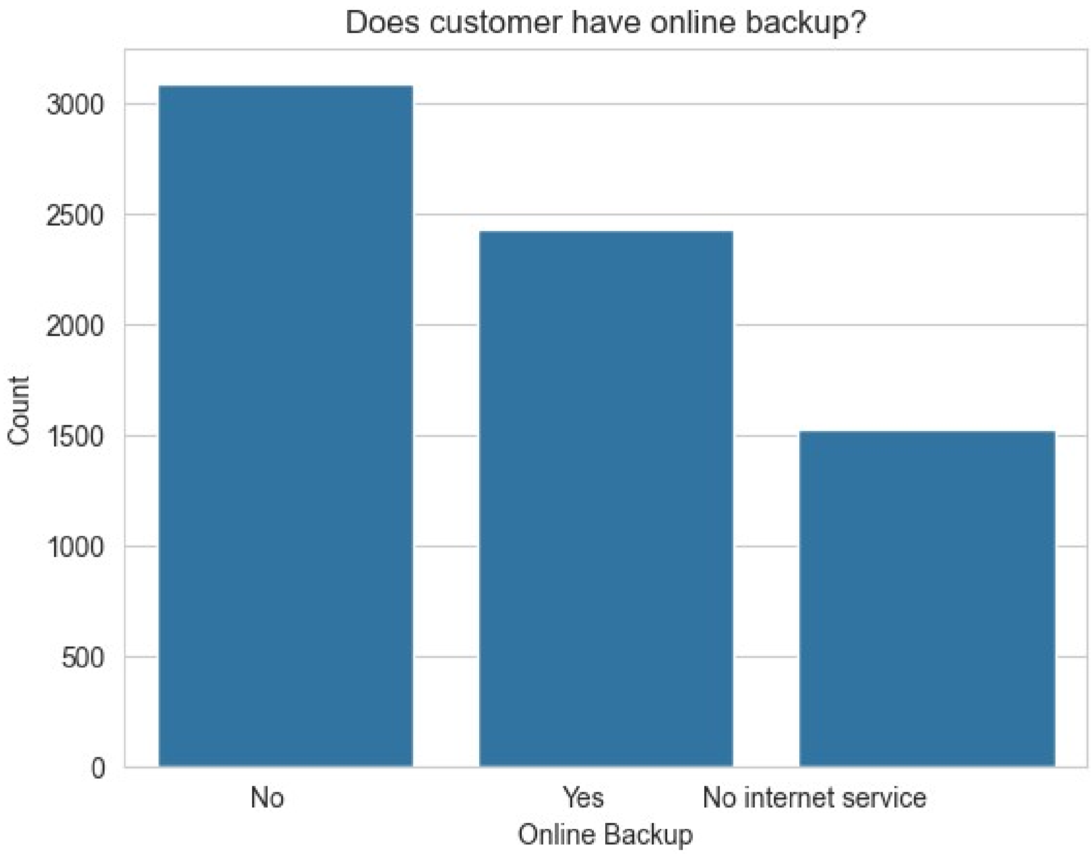
```
plot_categorical(df=df, col="OnlineSecurity", title="Does customer
have online security?",
                 xlabel="Online Security", ylabel="Count")

Data Statistics:
OnlineSecurity
No                     3498
Yes                    2019
No internet service    1526
Name: count, dtype: int64
```

This column indicates whether a customer has subscribed to additional online security service provided by the company. Approximately 3,500 customers do not have online security compared to 2,000 who have. What's interesting is that about 1,500 have been categorizes as not having an internet service.

This could be either miscategorized as if they didn't have an internet service, one would have expected them to have been categorized as `No`

Online Backup

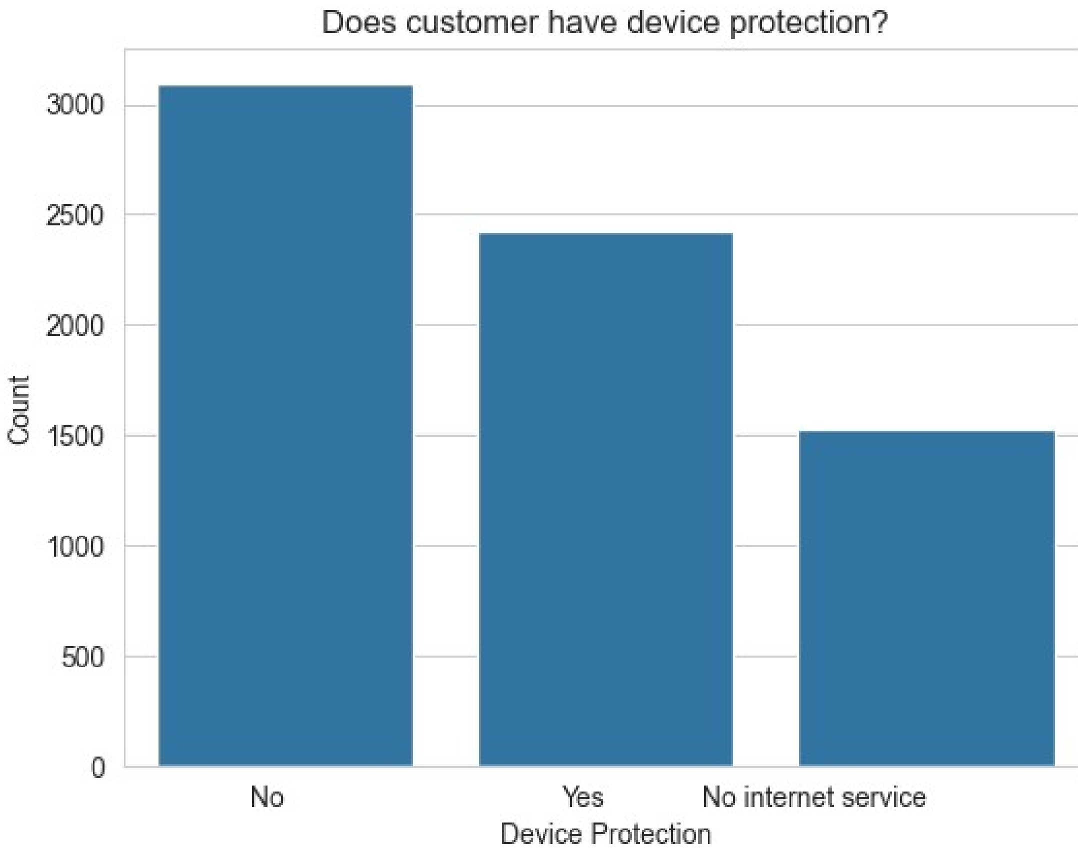
```
plot_categorical(df=df, col="OnlineBackup", title="Does customer have online backup?",
                 xlabel="Online Backup", ylabel="Count")

Data Statistics:
OnlineBackup
No                     3088
Yes                    2429
No internet service    1526
Name: count, dtype: int64
```



Slightly over 3,000 customers have not subscribed to an additional online backup service from the telco company compared to 2,000 who have. As this is a paid for service, the telco company may consider this service when looking at their churn rate.

Device Protection

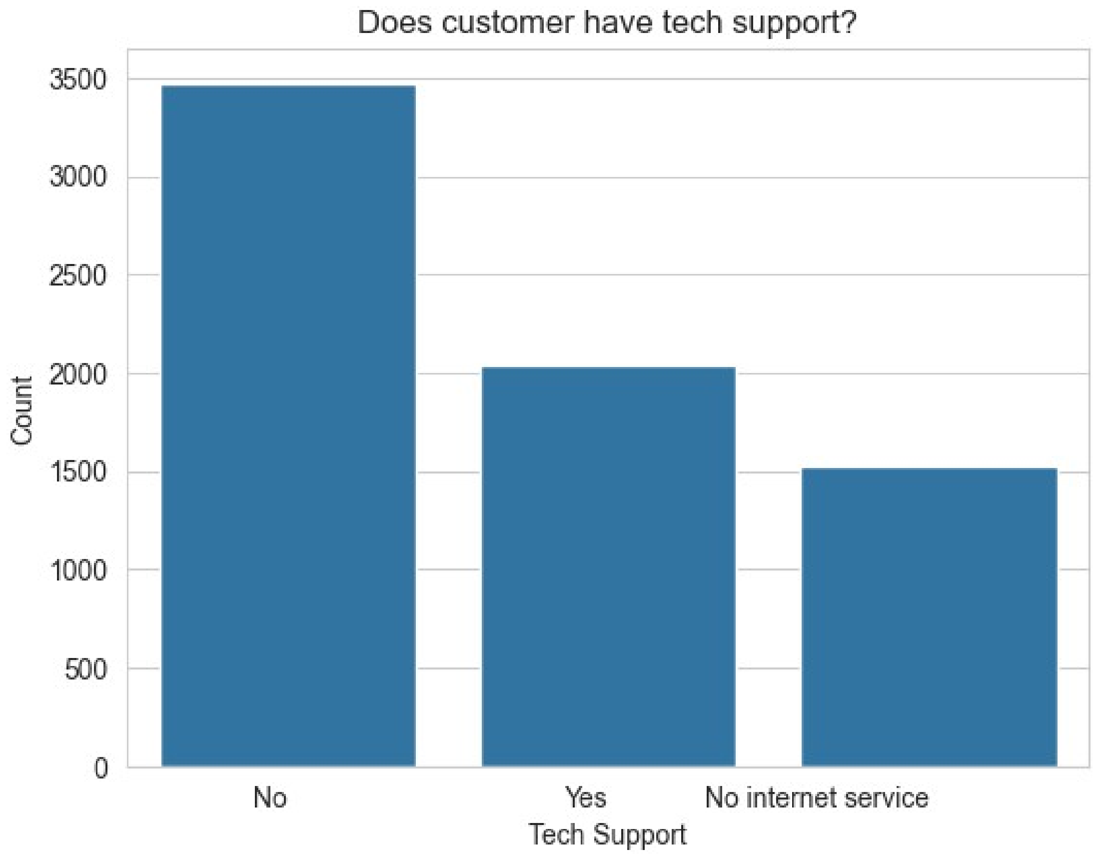
```
plot_categorical(df=df, col="DeviceProtection", title="Does customer have device protection?",
                 xlabel="Device Protection", ylabel="Count")

Data Statistics:
DeviceProtection
No                     3095
Yes                    2422
No internet service    1526
Name: count, dtype: int64
```



The above chart indicates the number of customers who have subscribed to an additional device protection service from the telco company. The number of customers who have not subscribed to this service is slightly over 3,000 compared to about 2,500 who have. This can be a good indicator for the telco company to see if they can improve their device protection service to attract more customers.

Tech Support

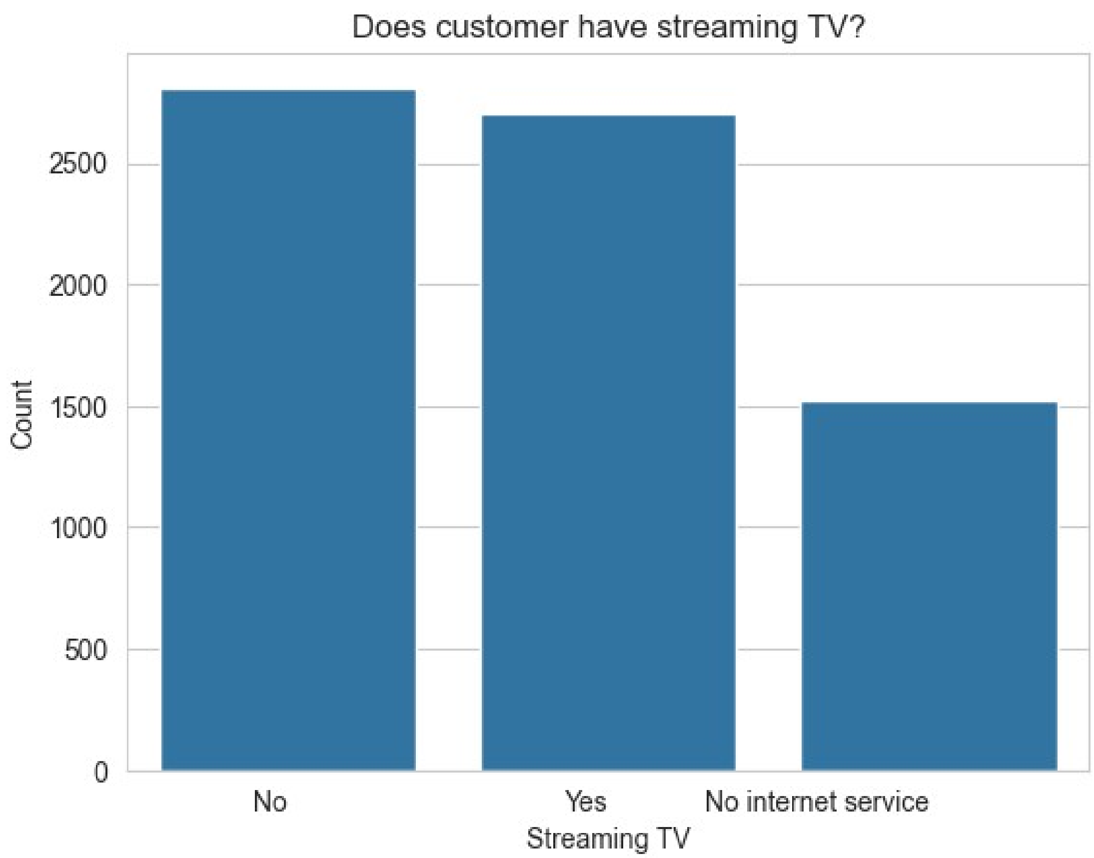
```
plot_categorical(df=df, col="TechSupport", title="Does customer have tech support?",
                 xlabel="Tech Support", ylabel="Count")

Data Statistics:
TechSupport
No                     3473
Yes                    2044
No internet service    1526
Name: count, dtype: int64
```



This chart shows the proportion of customer who have subscribed to an additional tech support service from the telco company. The number of customers who have not subscribed to this service is slightly below 3,500 compared to about 2,000 who have. This can be a good indicator for the telco company to see if they can improve their tech support service to attract more customers either by improving the service or reducing the price.

Streaming TV

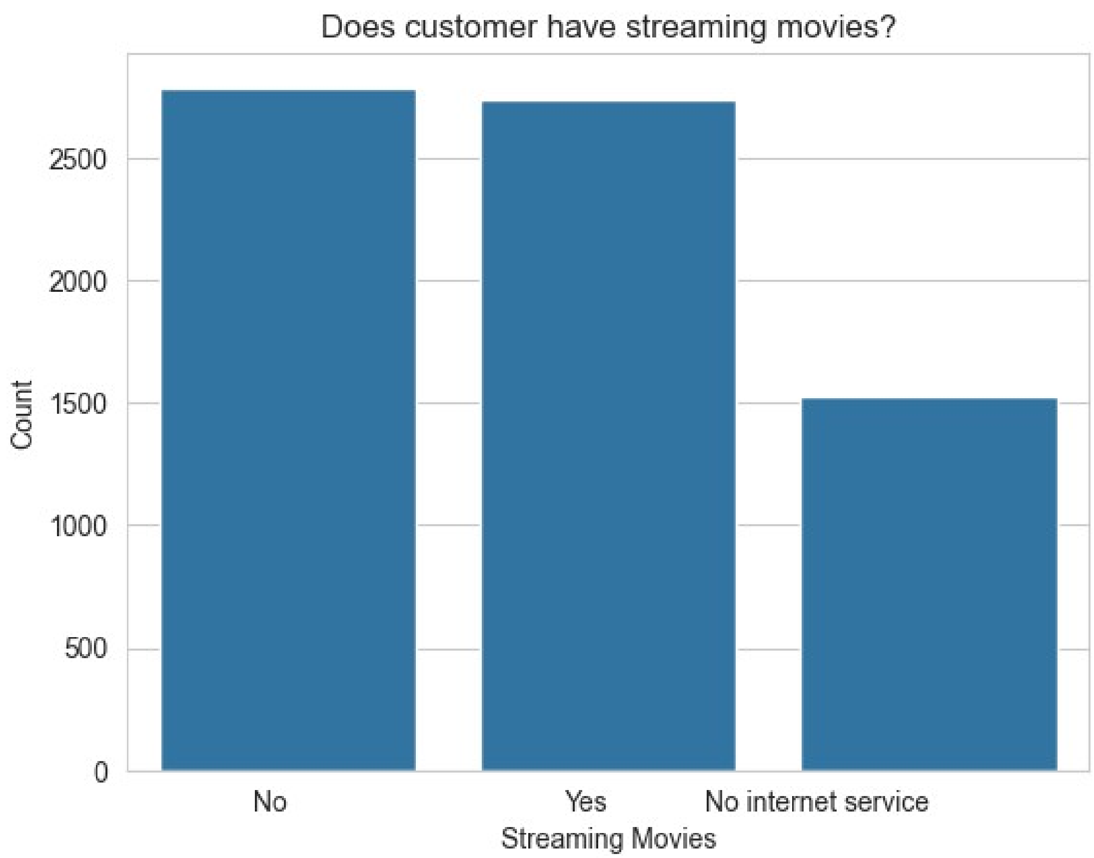
```
plot_categorical(df=df, col="StreamingTV", title="Does customer have streaming TV?",
                 xlabel="Streaming TV", ylabel="Count")

Data Statistics:
StreamingTV
No                     2810
Yes                    2707
No internet service    1526
Name: count, dtype: int64
```



The amount of telco customers who didn't use the telco's services to stream TV from a 3rd party provider is approximately 2,800 customers which is evenly split with the ones who did. However, as this is not a paid for service, the telco company may not be too concerned about this as they are not losing out on revenue.

Streaming Movies

```
plot_categorical(df=df, col="StreamingMovies", title="Does customer have streaming movies?",
                 xlabel="Streaming Movies", ylabel="Count")

Data Statistics:
StreamingMovies
No                     2785
Yes                    2732
No internet service    1526
Name: count, dtype: int64
```

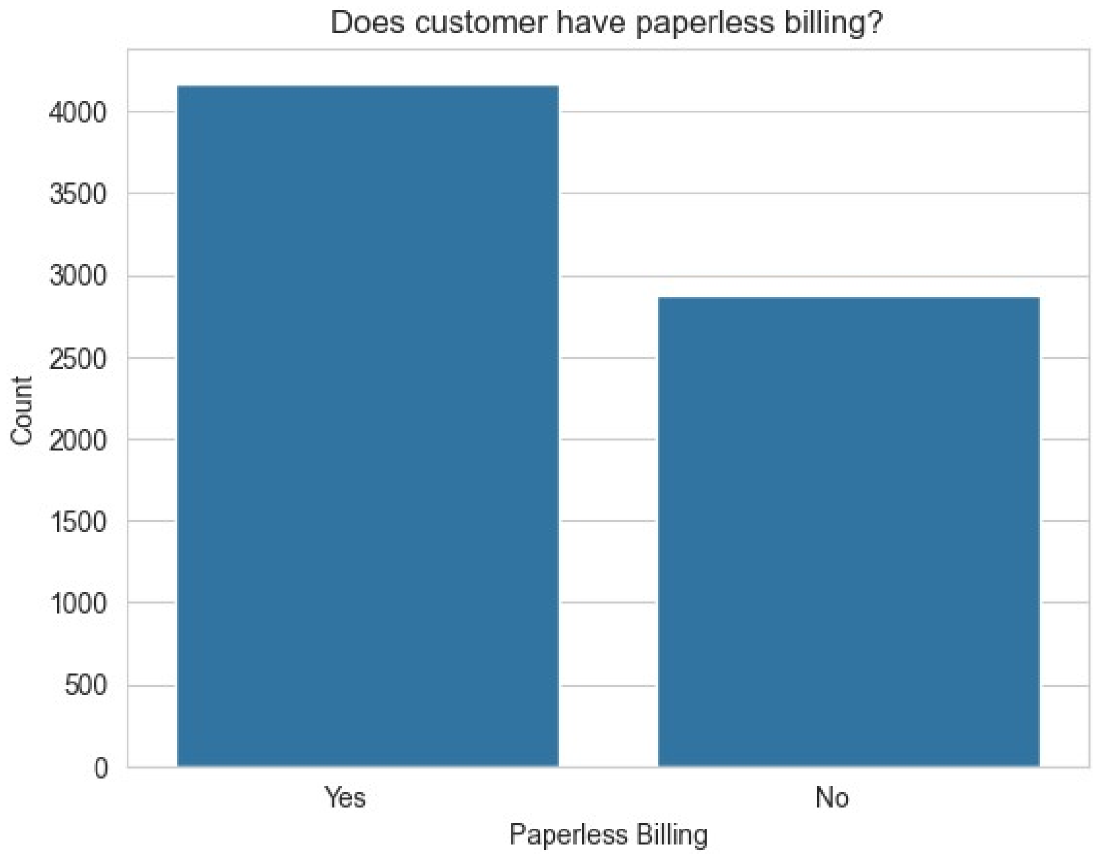


There's an even split between the customers who use the telco's services for streaming movies and those who don't at around 2,700 customers. However, about 1,500 customers have been categorized as not having an internet service and can't use the telco's service for streaming movies. As this is also not a paid for service, the telco company may not be too concerned about this as they are not losing out any revenue from these customers.

Billing Type

```
plot_categorical(df=df, col="PaperlessBilling", title="Does customer have paperless billing?",
                 xlabel="Paperless Billing", ylabel="Count")
```

```
Data Statistics:
PaperlessBilling
Yes    4171
No     2872
Name: count, dtype: int64
```

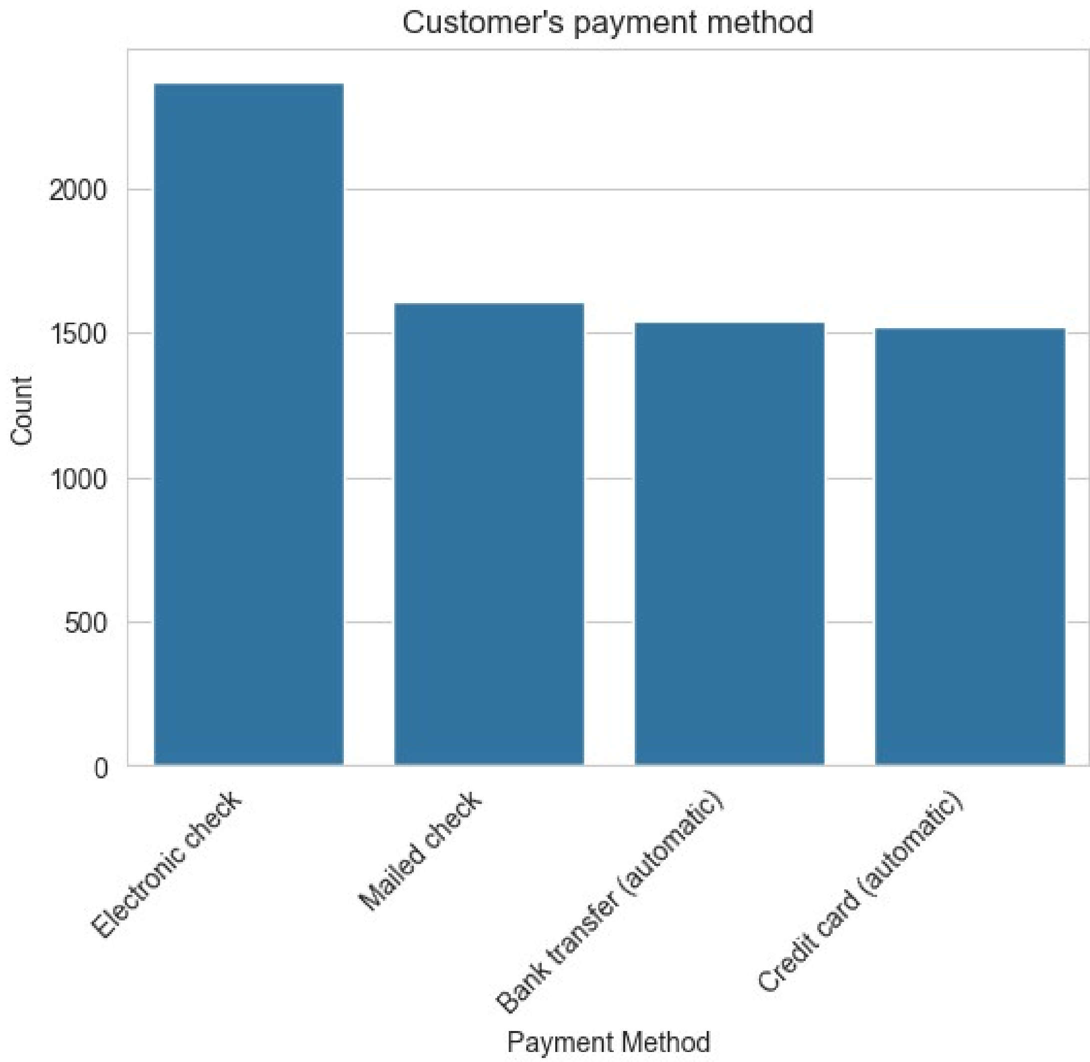


The majority of the customers have paperless billing which is to be expected as it is more convenient for customers and the company. However, the number of customers who do not have paperless billing is quite high at 2,872 customers.

Payment Method

```
plot_categorical(df=df, col="PaymentMethod", title="Customer's payment method",
                 xlabel="Payment Method", ylabel="Count", txt_rotation=45)
```

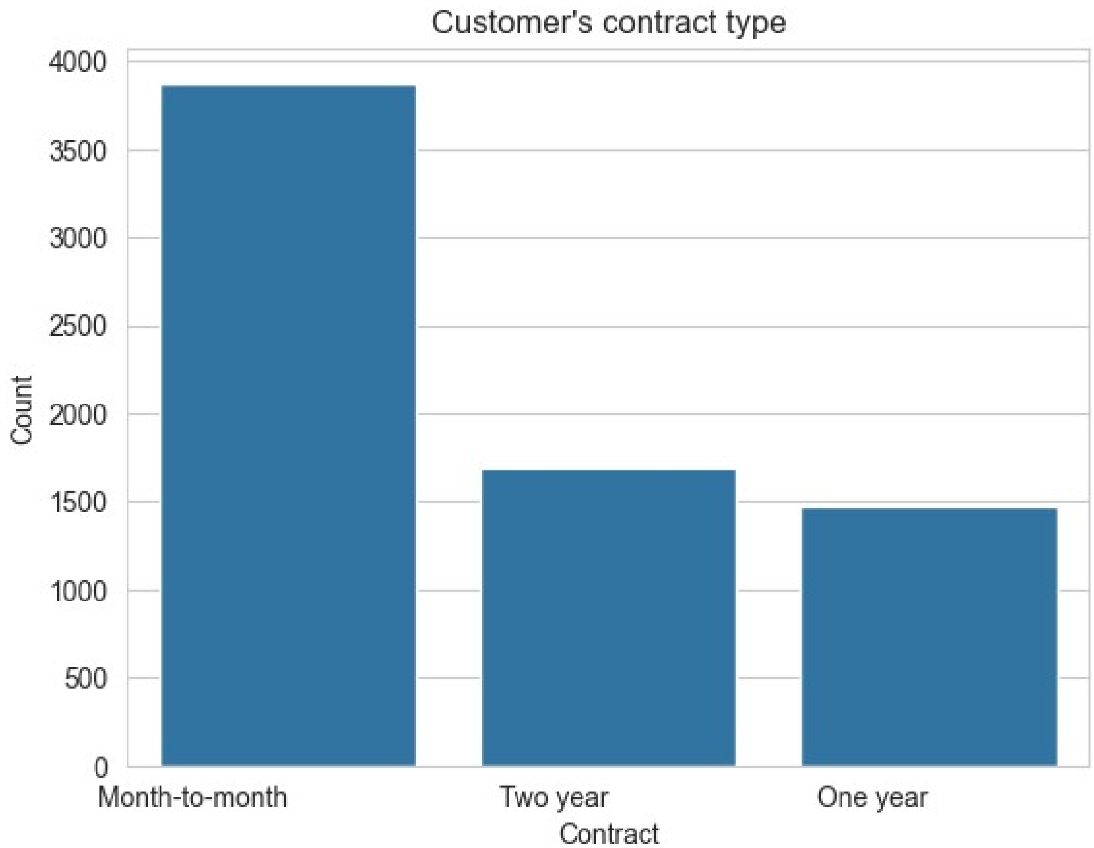
```
Data Statistics:
PaymentMethod
Electronic check             2365
Mailed check                 1612
Bank transfer (automatic)    1544
Credit card (automatic)      1522
Name: count, dtype: int64
```



The most common payment method is electronic check with over 2,000 customers. This wasn't expected as one would expect for a telco company in California, USA to have more customers paying via credit card or bank transfer.

Contract Type

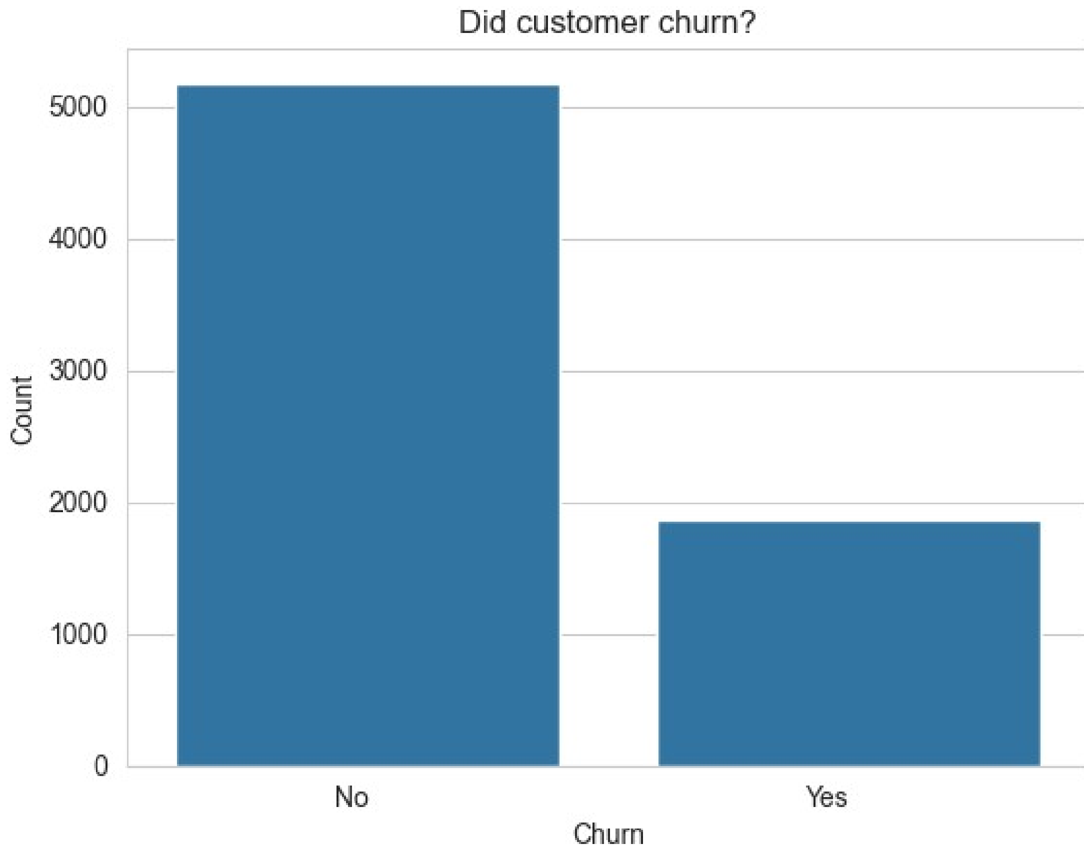
```
plot_categorical(df=df, col="Contract", title="Customer's contract type",
                 xlabel="Contract", ylabel="Count")

Data Statistics:
Contract
Month-to-month    3875
Two year          1695
One year          1473
Name: count, dtype: int64
```



Most of the telco customers are on a month-to-month contract which is to expected as not most people would want to be tied down to long-term contracts. What is interesting is that there are nearly 2,000 customers on a two-year contract which is more than the one-year contract customers which is not what one would expect.

Churn

```
plot_categorical(df=df, col="Churn", title="Did customer churn?",
                 xlabel="Churn", ylabel="Count")

Data Statistics:
Churn
No     5174
Yes    1869
Name: count, dtype: int64
```



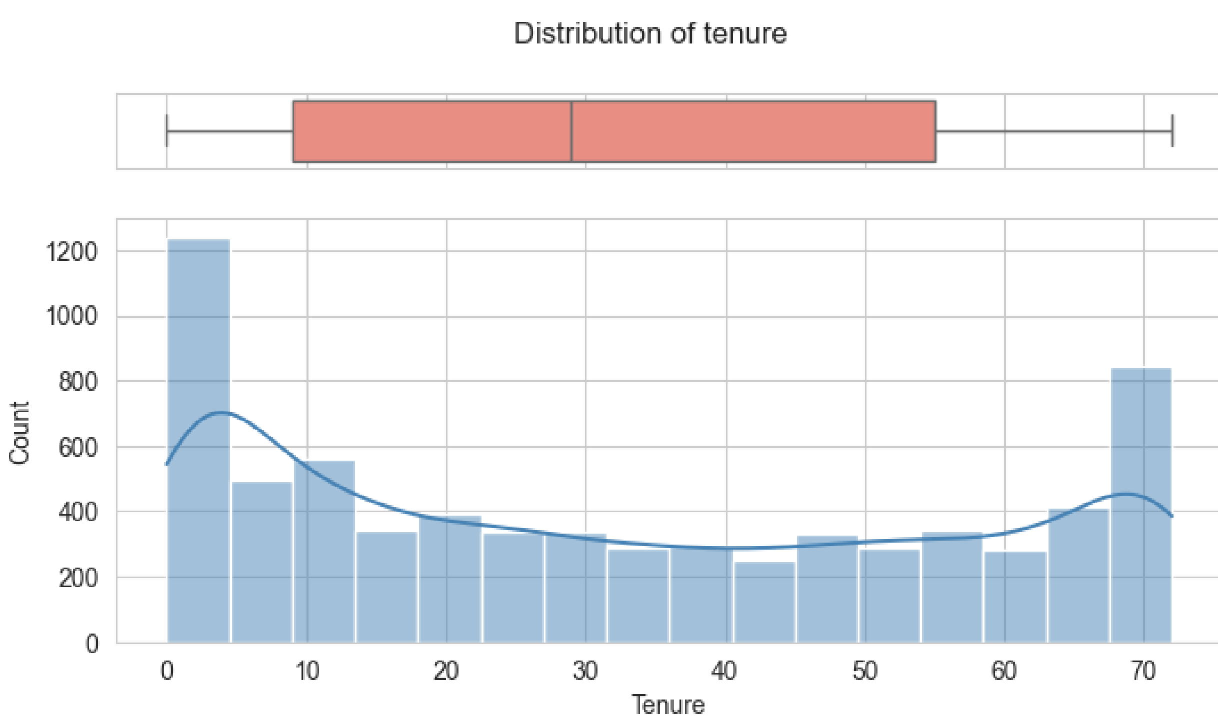
From the graph above we can see that nearly 5,000 customers did not churn compared to about 2,000 who did. This is a good sign for the telco company as they have a high retention rate. However, the challenge this may bring during modelling is the class imbalance between the churned and non-churned customers.

```
def plot_numerical(df:pd.DataFrame, columns:str, title:str,
xlabel:str, ylabel:str, kde:bool = True):
    """
    This function plot the distribution of the stated columns in a
dataframe.

    :param df: Dataframe to be used
    :param columns: Columns for visualizing
    :param title: Title for distribution
    :param xlabel: X label
    :param ylabel: Y label
    """
    print("Summary statistics:")
    print(df[columns].describe())

    fig, (ax_box, ax_hist) = plt.subplots(nrows=2, ncols=1,
                                          sharex=True, figsize=(8,4),
```

```
gridspec_kw={"height_ratios": (.15, .85)})
    sns.boxplot(df[columns], ax=ax_box, color="salmon", orient="h")
    sns.histplot(df[columns], ax=ax_hist, color="steelblue", kde=kde)

    plt.xlabel(xlabel)
    plt.ylabel(ylabel)

    plt.suptitle(title)
```

Tenure

```
plot_numerical(df=df, columns="tenure", title="Distribution of
tenure",
               xlabel="Tenure", ylabel="Count")

Summary statistics:
count    7043.000000
mean       32.371149
std        24.559481
min         0.000000
25%         9.000000
50%        29.000000
75%        55.000000
max        72.000000
Name: tenure, dtype: float64
```

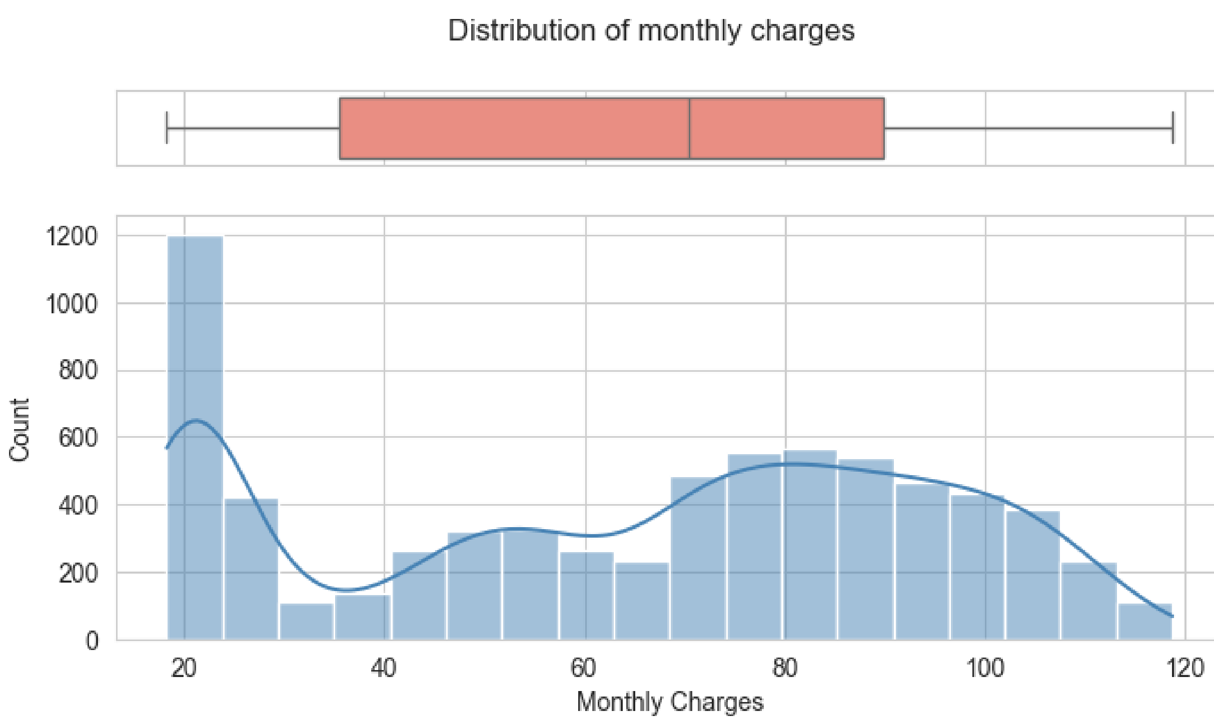


The above distribution shows the total months a customer has been with the telco company. We can see that the average tenure is 32 months with the median being approximately 29 months.

The distribution also has twin peaks at 0 months and 70 months.

Monthly Charges

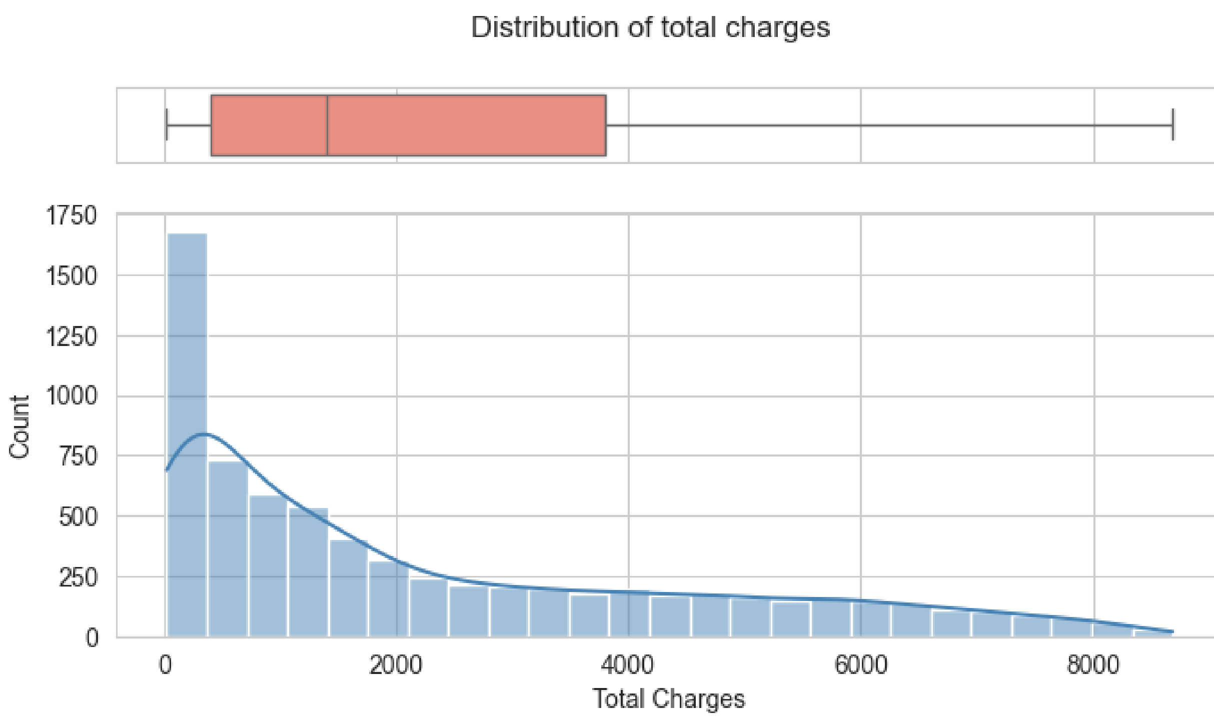
```
plot_numerical(df=df, columns="MonthlyCharges", title="Distribution of monthly charges",
               xlabel="Monthly Charges", ylabel="Count")

Summary statistics:
count    7043.000000
mean       64.761692
std        30.090047
min        18.250000
25%        35.500000
50%        70.350000
75%        89.850000
max       118.750000
Name: MonthlyCharges, dtype: float64
```



The above distribution shows the customers current total monthly charge for all their services from the company. The distribution is interestingly shaped as there is a peak at 20 dollars then a dip before another peak at 80 dollars. The average monthly charge is 64.76 dollars and a maximum of 118.75 dollars which is quite high.

Total Charges

```
plot_numerical(df=df, columns="TotalCharges", title="Distribution of total charges",
               xlabel="Total Charges", ylabel="Count")

Summary statistics:
count    7032.000000
mean     2283.300441
std      2266.771362
min        18.800000
25%       401.450000
50%      1397.475000
75%      3794.737500
max      8684.800000
Name: TotalCharges, dtype: float64
```



The above distribution shows the total amount charged to customers. The distribution is skewed to the right with the 75% of the customers being charged between USD 401 - USD3794.7 (Q1 - Q3). The averagge total charge is USD 2,283 however approximately 25% of the customers are charged less than USD 401.

Bivariate Analysis

In this section we will look at the relationship between the target variable and the other variables in the dataset. This will help us identify which variables have a strong relationship with the target variable and which ones don't.

```
totalcount = df["SeniorCitizen"].value_counts()
print(totalcount)
groupcount = df.groupby(["SeniorCitizen", "Churn"])
["SeniorCitizen"].count().unstack()
groupcount
```

```
SeniorCitizen
No     5901
Yes    1142
Name: count, dtype: int64
```

```
Churn            No   Yes
SeniorCitizen
No             4508  1393
Yes             666   476
```

```
group_proportions = groupcount.div(len(df), axis=0)
group_proportions
```

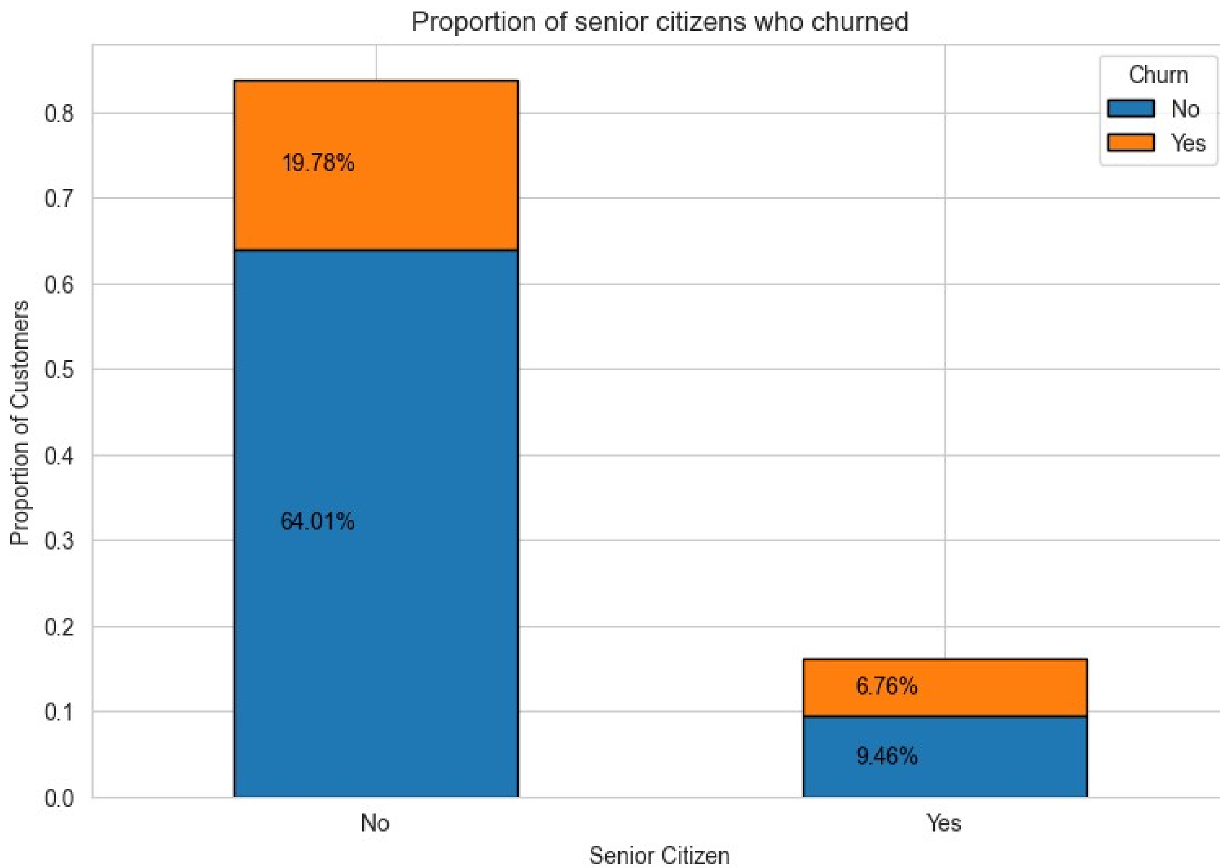
```
Churn                No       Yes
SeniorCitizen
No             0.640068  0.197785
Yes            0.094562  0.067585
```

```
def plot_bivariate(df: pd.DataFrame, x: str, title: str, xlabel: str,
ylabel: str,
                   hue: str = None, rotation: int = 0):
    """
    This function plots the relationship between the target variable
and another variable.

    :param df: Dataframe to be used
    :param x: X variable
    :param title: Title for distribution
    :param xlabel: X label
    :param ylabel: Y label
    :param hue: Hue variable
    """
    # Calculate proportions relative to the entire dataset
    total_counts = df[x].value_counts().sort_values(ascending=False)
    group_counts = df.groupby([x, hue])[x].count().unstack()
    proportions = group_counts.div(len(df), axis=0)

    # Create subplots
    fig, ax = plt.subplots(figsize=(9, 6))

    # Plotting the grouped bar chart
    ax = proportions.plot(kind='bar', stacked=True, edgecolor='black',
ax=ax)

    # Adding annotation
```

```
    for i, (index, row) in enumerate(proportions.iterrows()):
        for j, value in enumerate(row):
            x_pos = i + j * 0 - 0.1 * (len(row) - 1)  # Adjust the
position for hue
            y_pos = row.head(j + 1).sum() - value / 2
            proportion_text = f"{value:.2%}"
            ax.text(x_pos, y_pos, proportion_text, ha='center',
va='center', color='black')

    # Customize the plot
    ax.set_title(title)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_xticks(range(len(total_counts)))
    ax.set_xticklabels(total_counts.index, rotation=rotation,
ha="center")
    ax.legend(title=hue)

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

Senior Citizens

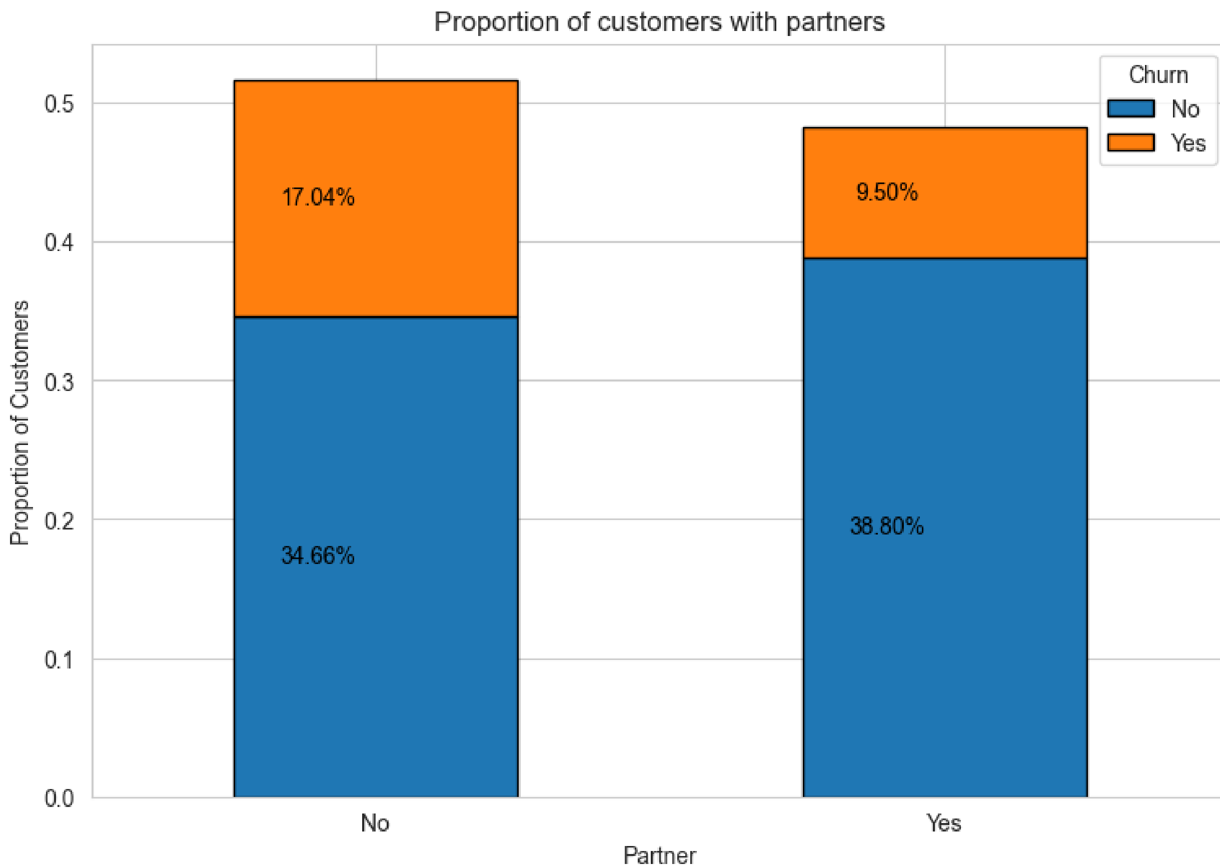
```
plot_bivariate(df=df, x="SeniorCitizen", title="Proportion of senior
citizens who churned",
               xlabel="Senior Citizen", ylabel="Proportion of
Customers", hue="Churn")
```



From the above chart we can see out of the customers who were registered as non-senior citizens 64.01% did not churn while 19.78% churned. However, the proportion of senior customers who churned vs didn't churn is quite similar at 9.46% vs 6.76%

Partner

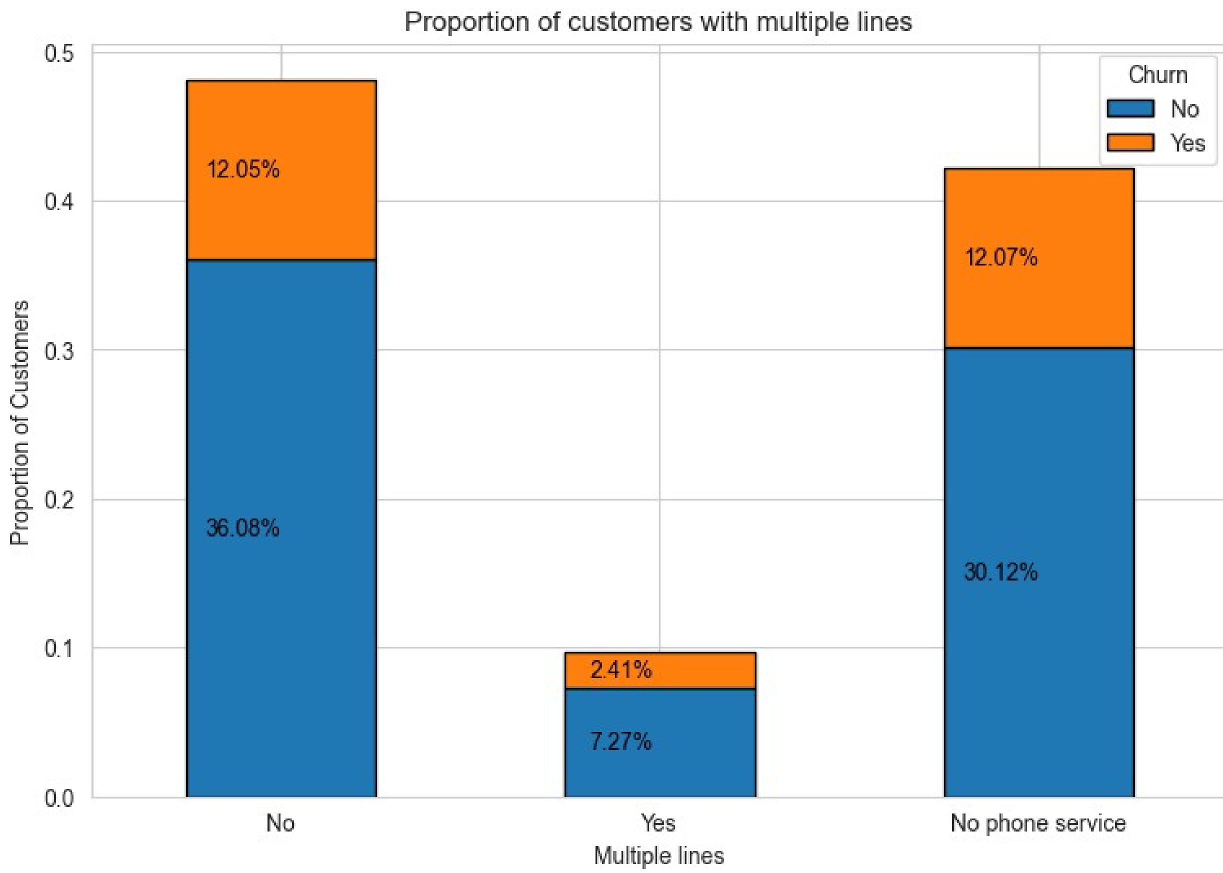
```
plot_bivariate(df=df, x="Partner", title="Proportion of customers with partners",
               xlabel="Partner", ylabel="Proportion of Customers", hue="Churn")
```



From the above chart we can see that the proportion of customers who didn't have a partner and churned is 17.04% of the total customers compared ot 9.50% who had a partner and churned. This would indicate that customers who have a partner are less likely to churn compared to those who don't, however, looking at the proportion of customers who didn't churn, the difference is not that significant.

Multiple Lines

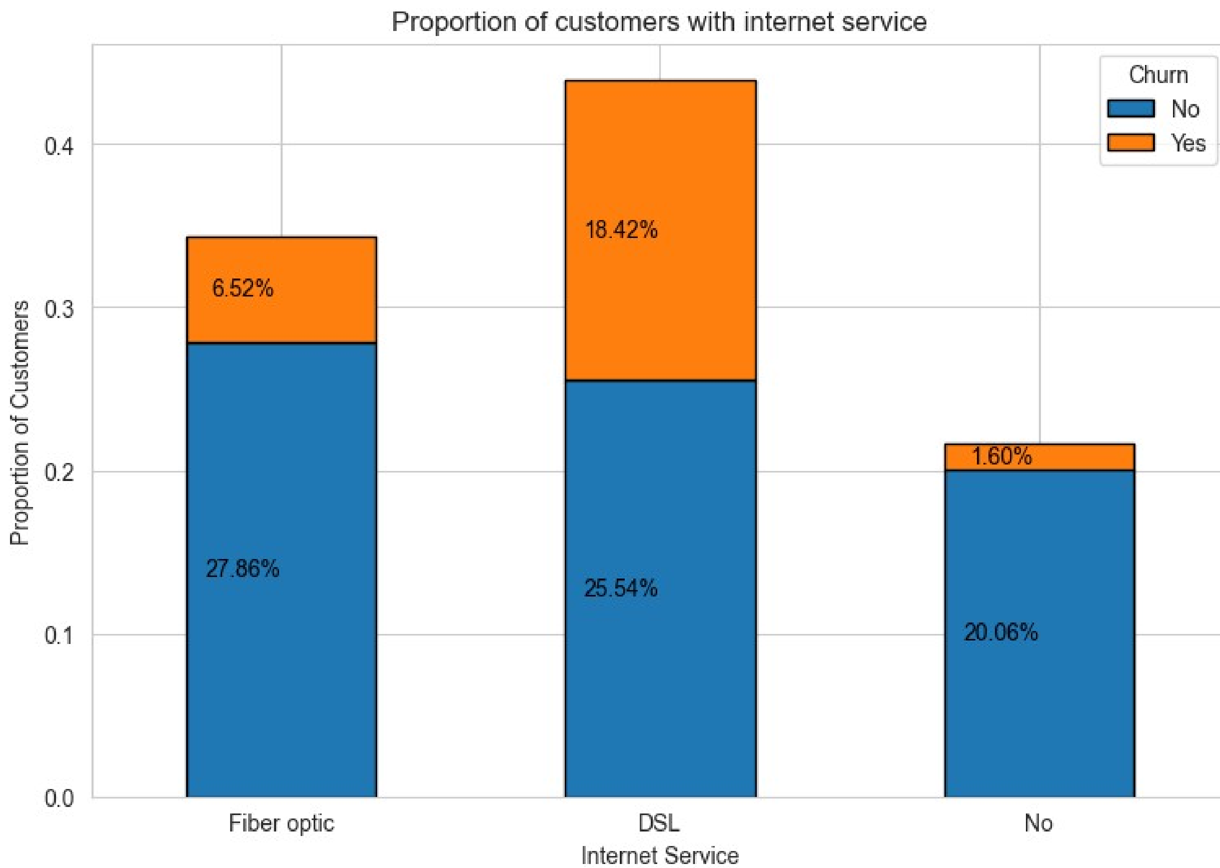
```
plot_bivariate(df=df, x="MultipleLines", title="Proportion of
customers with multiple lines",
               xlabel="Multiple lines", ylabel="Proportion of
Customers", hue="Churn")
```



From the above chart we can see that the customers who are most likely to churn either have not subscribed to a phone service of haven't subscribed to multiple lines from the telco company.

Internet Service

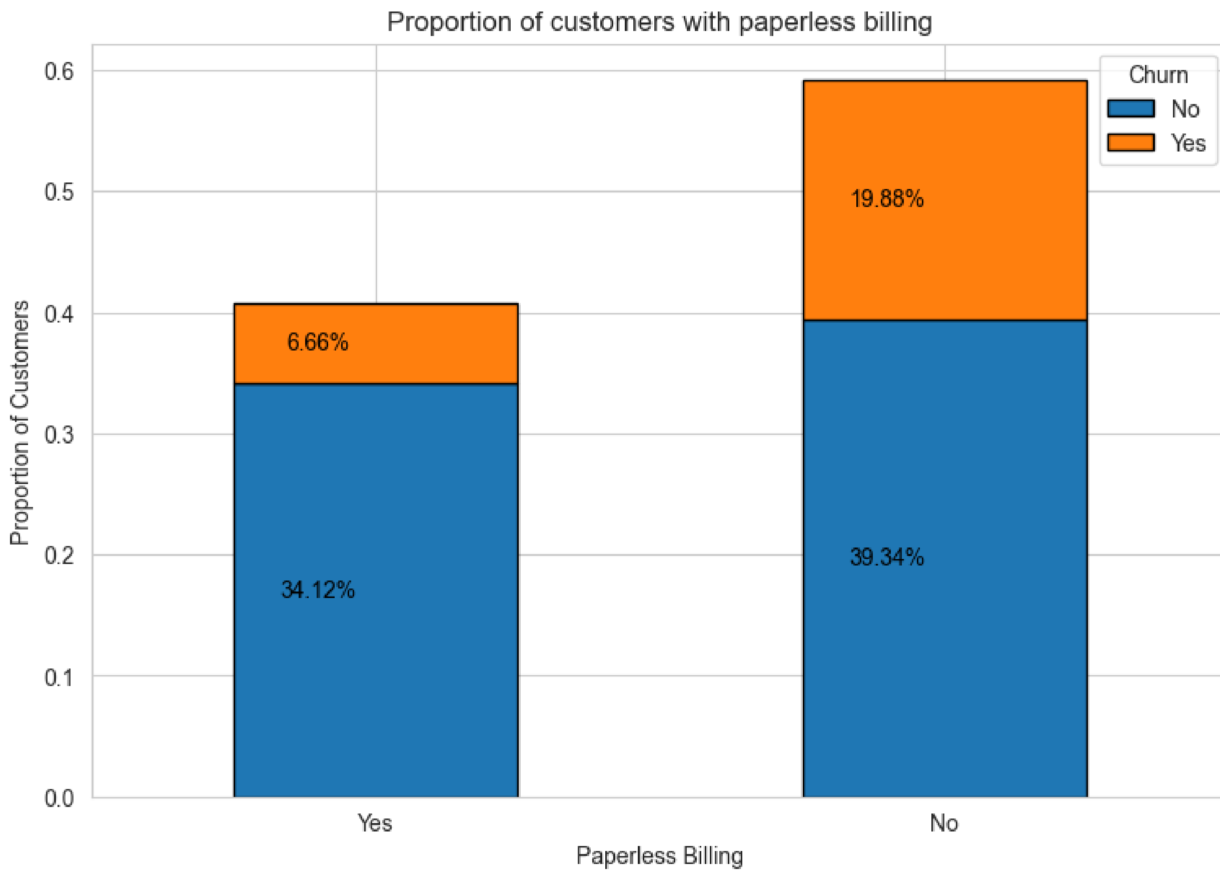
```
plot_bivariate(df=df, x="InternetService", title="Proportion of customers with internet service",
               xlabel="Internet Service", ylabel="Proportion of Customers", hue="Churn")
```



Looking at the above chart we can see that there's a higher chance for a customer to churn if they have subscribed to a DSL internet service. What's interesting is that there are more customers using DSL internet service over Fiber optic service which is up to 100x faster (Socket, n.d.). This could be due to the price of the DSL service being lower than the Fiber optic service.

Billing Type

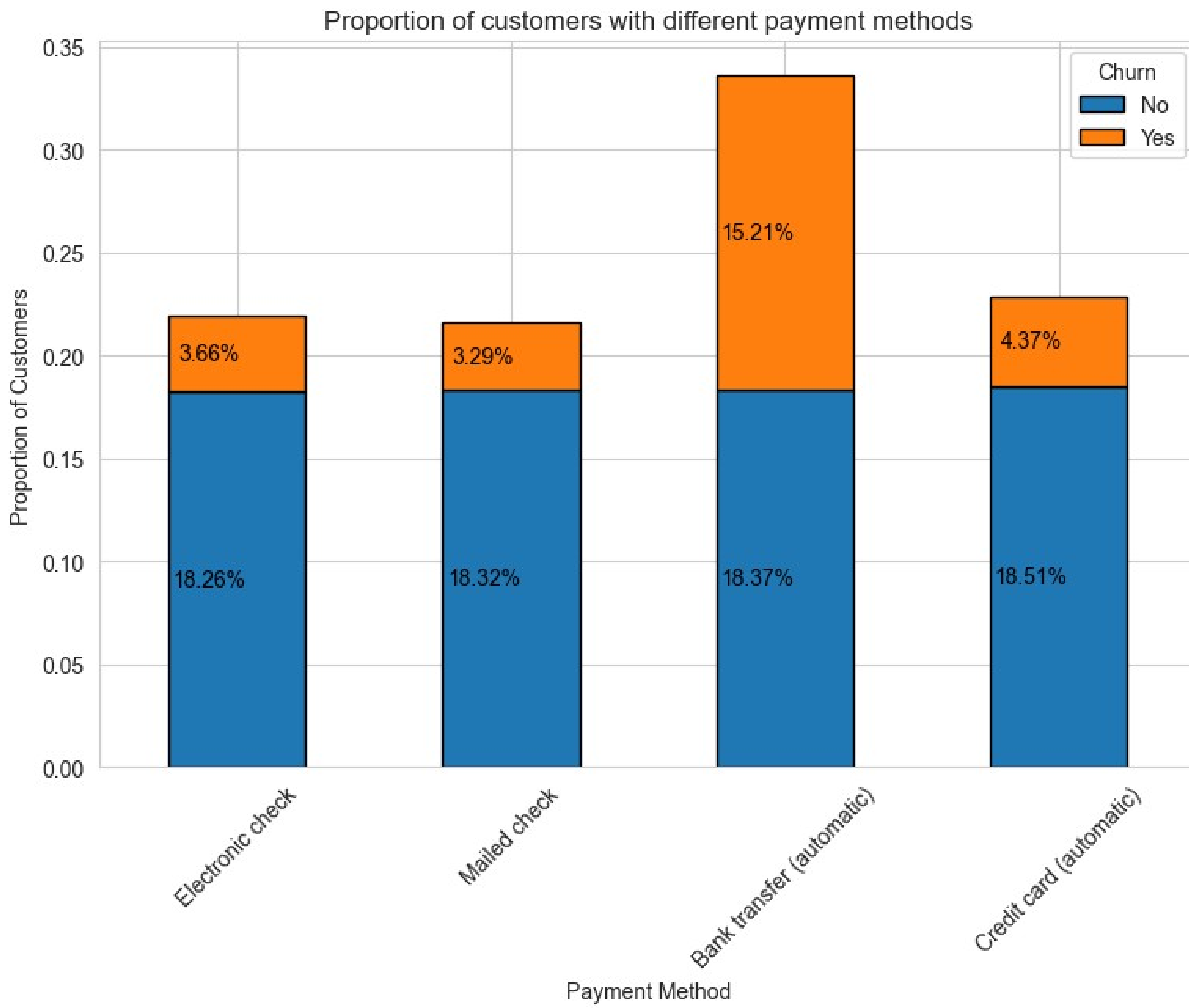
```
plot_bivariate(df=df, x="PaperlessBilling", title="Proportion of customers with paperless billing",
               xlabel="Paperless Billing", ylabel="Proportion of Customers", hue="Churn")
```



From the above chart we can see taht the customers who have not subscribed to paperless billing are more likely to churn (19.88%) compared to those who have paperless billing (6.66%). This could be due to the fact that customers who have paperless billing are more likely to be on a contract and have subscribed to other services from the telco company.

Payment Method

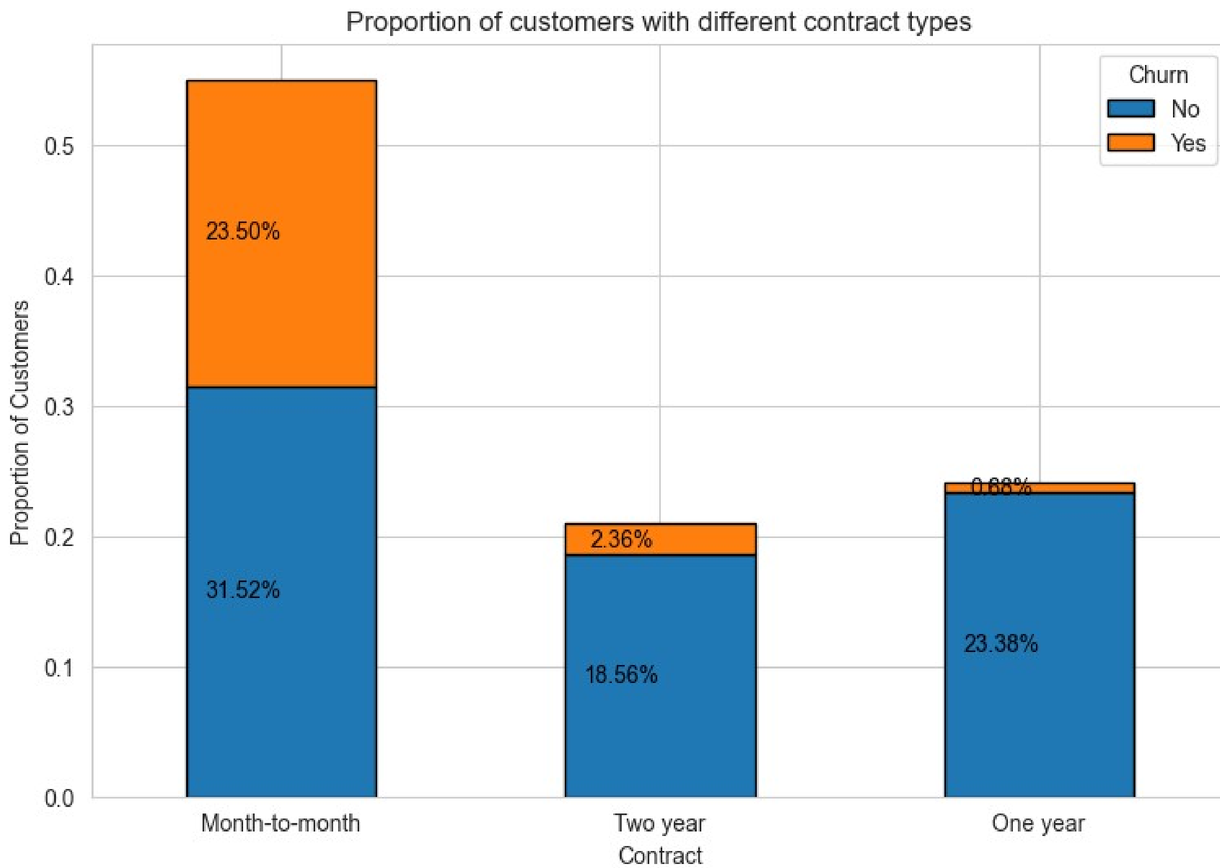
```
plot_bivariate(df=df, x="PaymentMethod", title="Proportion of customers with different payment methods",
               xlabel="Payment Method", ylabel="Proportion of Customers", hue="Churn",
               rotation=45)
```



From the above chart we can see that customers who paid via an automated bank transfer were at a higher risk of churning (15.21%) compared to the other payment methods.

Contract Type

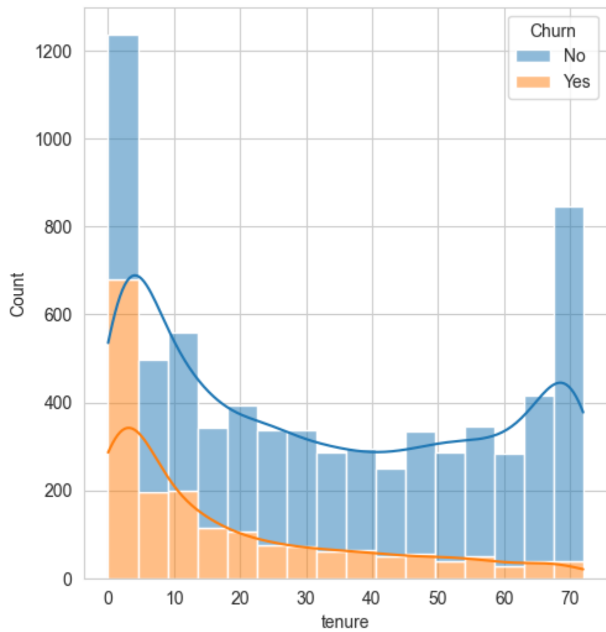
```
plot_bivariate(df=df, x="Contract", title="Proportion of customers with different contract types",
               xlabel="Contract", ylabel="Proportion of Customers", hue="Churn")
```

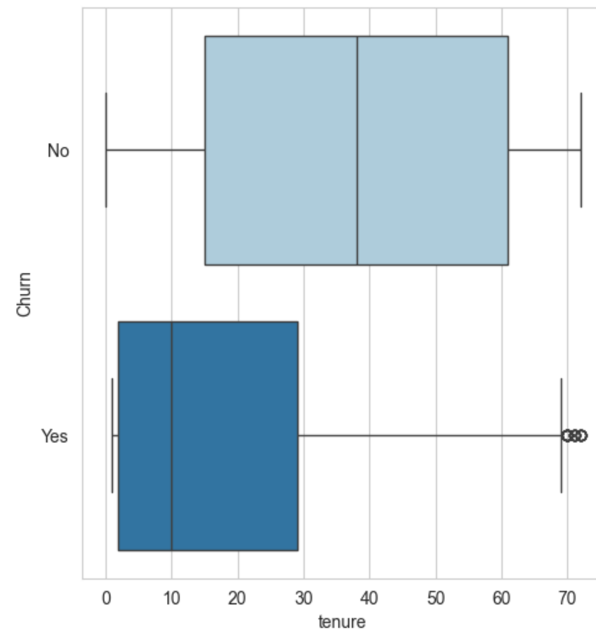


From the above chart we can see that customers who were on a month-to-month contract were at a higher risk of churning (23.50%) compared to those on a one-year contract (0.88%) and two-year contract (2.36%). This is to be expected as customers on a month-to-month contract are not tied down to a long-term contract and can easily switch to another provider which is not a good sign for the telco company.

Tenure

```
fig, (ax1, ax2) = plt.subplots(nrows=1,
                               ncols=2,
                               figsize=(12, 6))

sns.histplot(x="tenure", hue="Churn", data=df, multiple="stack",
             kde=True, ax=ax1)
sns.boxplot(x="tenure", y="Churn", data=df, hue="Churn",
            palette="Paired", ax=ax2);
```





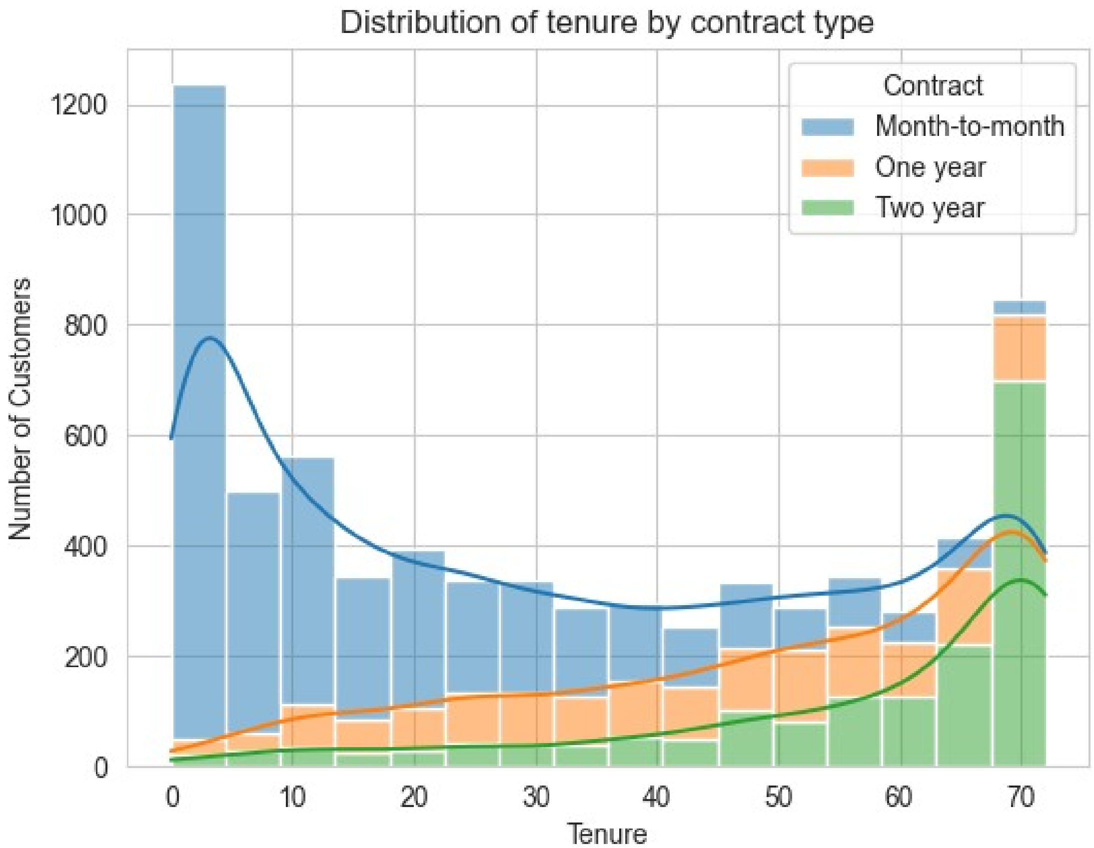
From the box plot we can see on average customers who churned had a lower overall tenure with the telco (2-30 months) compared to those who didn't churn (15-60months). Despite the overall trend there are a few customers who churned after being with the telco for approximately 70+ months.

Looking at the histogram above, we can see that the longer a customer stayed with the telco company, they were less likely to churn. This is to be expected as the longer a customer stays with a company, the more likely they won't churn.

Understanding the factors that made the customers churn after being with the telco for so long could be very beneficial for the company as they can try to improve their services to reduce the number of customers who churn after being with them for so long.

Contract Type & Tenure

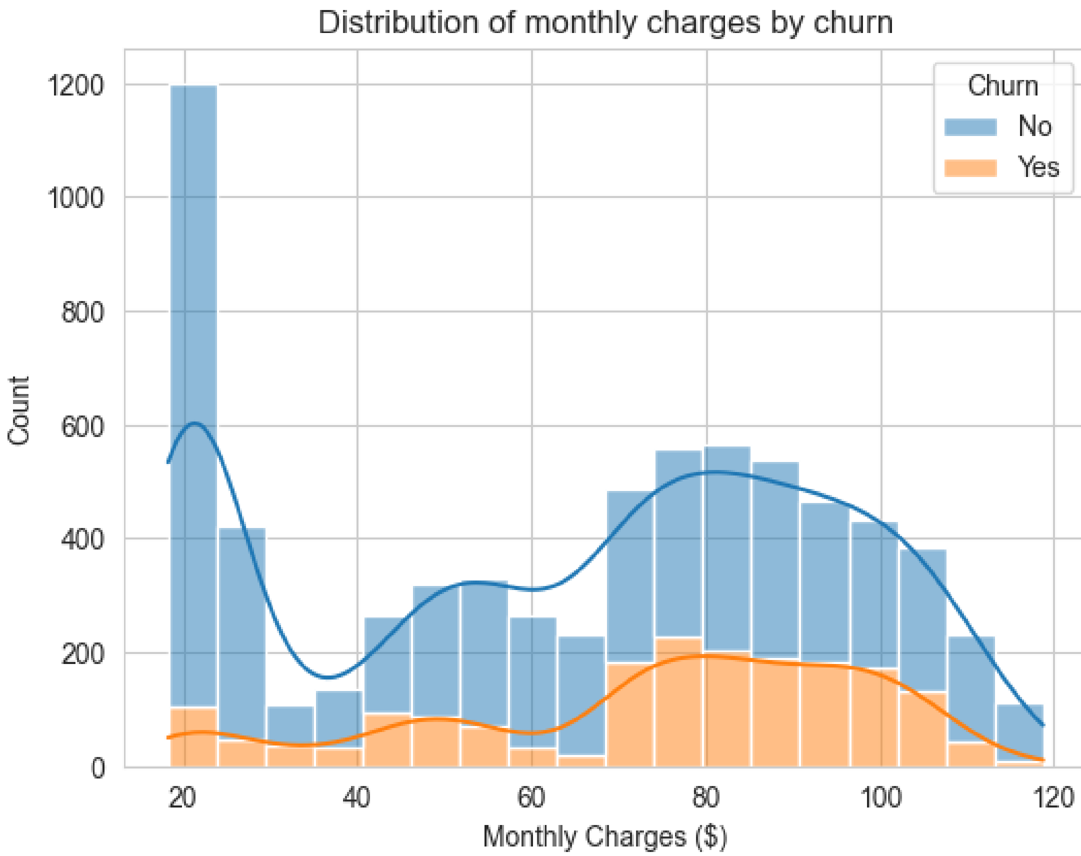
```
sns.histplot(x="tenure", hue="Contract", data=df, multiple="stack",
             kde=True)
plt.title("Distribution of tenure by contract type")
plt.xlabel("Tenure")
plt.ylabel("Number of Customers");
```



Looking at the distribution above we can see that the longer the customers tenure with the telco, they are more likely to be on a two-year contract compared to a one-year or month-to-month contract. This is to be expected as customers who have been with the telco for a long time are more likely to be loyal and pick a long-term contract that may offer them a discount or other benefits.

Monthly Charges

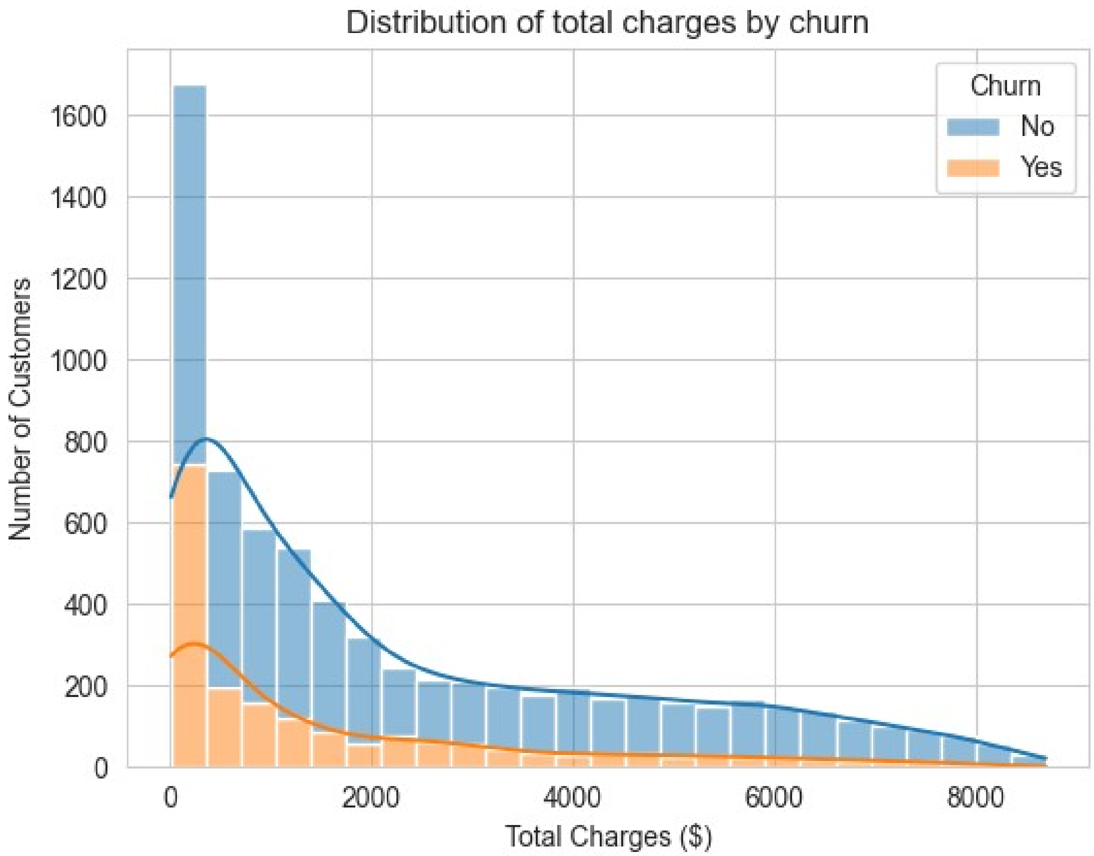
```
sns.histplot(x="MonthlyCharges", hue="Churn", data=df,
multiple="stack",
             kde=True)
plt.title("Distribution of monthly charges by churn")
plt.xlabel("Monthly Charges ($)");
```



From the above distribution we can see the number of customers who churned increased between 70-100 dollars. This could be due to the fact that customers who are paying more are more likely to churn as they may be able to get a better deal from another provider or get cash strapped and not be able to afford the service anymore.

Total Charges

```
sns.histplot(x="TotalCharges", hue="Churn", data=df, multiple="stack",

             kde=True)
plt.title("Distribution of total charges by churn")
plt.xlabel("Total Charges ($)")
plt.ylabel("Number of Customers");
```



From the above distribution we can see that the number of customers who churned decreased as the total charges increased. This is to be expected as the higher the total charges, fewer customers will be able to afford as well as the ones who may churn may not want to pay more and more for the service.

3. Data Preparation

In this section we will prepare the data for modelling. This will include:

- Selecting features
- Encoding categorical variables
- Dropping unnecessary columns
- Correlation analysis

```
df1 = df.copy()
df1.head()

   customerID  gender SeniorCitizen Partner Dependents  tenure
PhoneService  \
0  7590-VHVEG  Female             No     Yes         No       1
No
1  5575-GNVDE    Male             No      No         No      34
Yes
```

```
2  3668-QPYBK    Male             No      No          No       2
Yes
3  7795-CFOCW    Male             No      No          No      45
No
4  9237-HQITU  Female             No      No          No       2
Yes

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

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

  PaperlessBilling              PaymentMethod  MonthlyCharges
TotalCharges  \
0              Yes           Electronic check           29.85
29.85
1               No               Mailed check           56.95
1889.50
2              Yes               Mailed check           53.85
108.15
3               No  Bank transfer (automatic)           42.30
1840.75
4              Yes           Electronic check           70.70
151.65

  Churn
0    No
1    No
2   Yes
3    No
4   Yes
```

Selecting/ Dropping Features

We will drop `customerID` as it is a unique identifier for each customer, in addition we will drop `gender` as it is not a useful feature for modelling.

```
# Dropping customerID column
df1.drop(["customerID","gender", "PaymentMethod"], axis=1,
inplace=True)
df1.head()
```

```
  SeniorCitizen Partner Dependents  tenure PhoneService
MultipleLines  \
0            No     Yes         No       1           No   No phone
service
1            No      No         No      34          Yes
No
2            No      No         No       2          Yes
No
3            No      No         No      45           No   No phone
service
4            No      No         No       2          Yes
No

  InternetService OnlineSecurity OnlineBackup DeviceProtection
TechSupport  \
0             DSL             No          Yes               No
No
1             DSL            Yes           No              Yes
No
2             DSL            Yes          Yes               No
No
3             DSL            Yes           No              Yes
Yes
4     Fiber optic             No           No               No
No

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

   MonthlyCharges  TotalCharges Churn
0           29.85         29.85    No
1           56.95       1889.50    No
2           53.85        108.15   Yes
3           42.30       1840.75    No
4           70.70        151.65   Yes
```

Based of IBM Telco Data (additional) we will drop columns that the telco doesn't charge its customers it is not a useful feature for modelling.

```
free_to_use = ["StreamingTV", "StreamingMovies"]
df1.drop(free_to_use, axis=1, inplace=True)
df1.head()
```

```
  SeniorCitizen Partner Dependents  tenure PhoneService 
MultipleLines  \
0            No     Yes         No       1           No   No phone 
service   
1            No      No         No      34          Yes 
No   
2            No      No         No       2          Yes 
No   
3            No      No         No      45           No   No phone 
service   
4            No      No         No       2          Yes 
No   

  InternetService OnlineSecurity OnlineBackup DeviceProtection 
TechSupport  \
0             DSL             No          Yes               No 
No   
1             DSL            Yes           No              Yes 
No   
2             DSL            Yes          Yes               No 
No   
3             DSL            Yes           No              Yes 
Yes   
4     Fiber optic             No           No               No 
No   

         Contract PaperlessBilling  MonthlyCharges  TotalCharges Churn  

0  Month-to-month              Yes           29.85         29.85    No  

1        One year               No           56.95       1889.50    No  

2  Month-to-month              Yes           53.85        108.15   Yes  

3        One year               No           42.30       1840.75    No  

4  Month-to-month              Yes           70.70        151.65   Yes  
```

Dropping missing rows

```
df1.isna().sum()
```

```
SeniorCitizen        0
Partner              0
Dependents           0
tenure               0
PhoneService         0
MultipleLines        0
InternetService      0
OnlineSecurity       0
OnlineBackup         0
DeviceProtection     0
TechSupport          0
Contract             0
PaperlessBilling     0
MonthlyCharges       0
TotalCharges        11
Churn                0
dtype: int64
```

```
df1.dropna(inplace=True)
df1.isna().sum()
```

```
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
Contract            0
PaperlessBilling    0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64
```

We decided to drop the missing rows as there were only 11 rows with missing values which is a very small number compared to the total number of rows in the dataset. In addition, we can see that the missing values are in the `TotalCharges` column which is a numerical column. As the number of missing values is very small, we can drop these rows without affecting the distribution of the data.

Encoding Categorical Variables

```
# Selecting categorical columns
cat_cols = df1.select_dtypes(include="object").columns.tolist()
cat_cols
```

```
['SeniorCitizen',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'Contract',
 'PaperlessBilling',
 'Churn']

# Encoding categorical columns
label_encoder = LabelEncoder()

for col in cat_cols:
    df1[col] = label_encoder.fit_transform(df1[col])

df1.head()

   SeniorCitizen  Partner  Dependents  tenure  PhoneService
MultipleLines  \
0              0        1           0       1             0
1
1              0        0           0      34             1
0
2              0        0           0       2             1
0
3              0        0           0      45             0
1
4              0        0           0       2             1
0

   InternetService  OnlineSecurity  OnlineBackup  DeviceProtection  \
0                0               0             2                 0
1                0               2             0                 2
2                0               2             2                 0
3                0               2             0                 2
4                1               0             0                 0

   TechSupport  Contract  PaperlessBilling  MonthlyCharges
TotalCharges  \
0            0         0                 1           29.85
29.85
1            0         1                 0           56.95
1889.50
2            0         0                 1           53.85
108.15
```

```
3             2         1                 0           42.30
1840.75
4             0         0                 1           70.70
151.65

   Churn
0      0
1      0
2      1
3      0
4      1
```

```
df1["Contract"].value_counts()
```

```
Contract
0    3875
2    1685
1    1472
Name: count, dtype: int64
```

```
df["Contract"].value_counts()
```

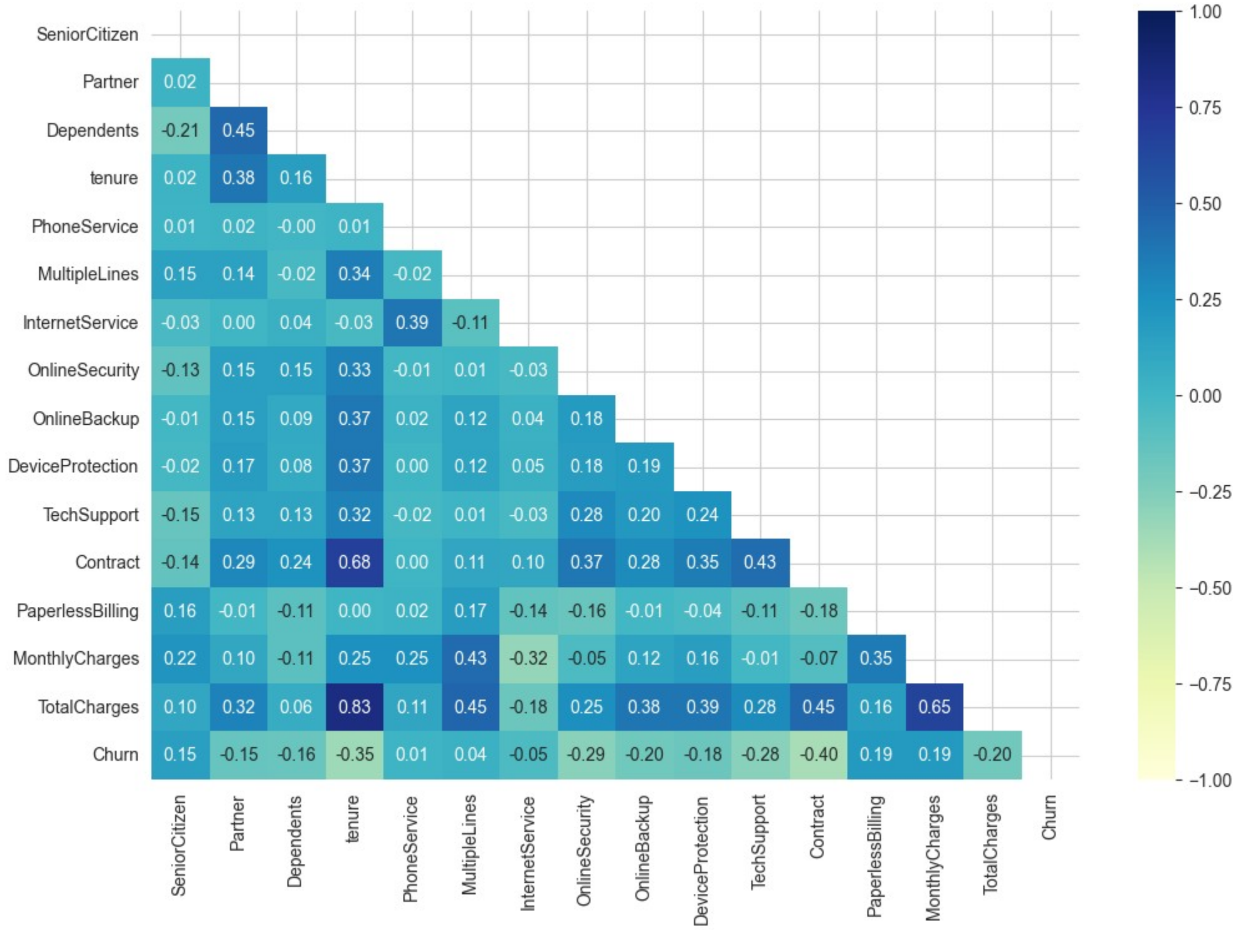
```
Contract
Month-to-month    3875
Two year          1695
One year          1473
Name: count, dtype: int64
```

From the above 3 code cells we can see that the features within our dataframe have been encoded correctly. We can see that the `Contract` column has been encoded as follows:

- Month-to-month = 0
- One year = 1
- Two year = 2

Correlation Analysis

```
# Correlation matrix
corr = df1.corr()

# Plotting heatmap
plt.figure(figsize=(12,8))
mask = np.triu(np.ones_like(corr))
sns.heatmap(corr, annot=True,
            cmap="YlGnBu",
            mask=mask,
            fmt=".2f",
            vmin=-1,
            vmax=1);
```



From the above correlation matrix we can see that there are no strong correlations between the features and the target variable. Based of the heatmap we will remove `PhoneService`, `MultipleLines` and `InternetService` as they have a correlation of 0.01, 0.04 and -0.05 respectively which is very low and may not add any useful insights for modelling.

```
# Dropping addditional features
df1.drop(["PhoneService", "MultipleLines", "InternetService"], axis=1,
inplace=True)
df1.head()

   SeniorCitizen  Partner  Dependents  tenure  OnlineSecurity
OnlineBackup  \
0              0        1           0       1               0
2
1              0        0           0      34               2
0
2              0        0           0       2               2
2
3              0        0           0      45               2
0
4              0        0           0       2               0
```

```
0

   DeviceProtection  TechSupport  Contract  PaperlessBilling
MonthlyCharges  \
0                 0            0         0                 1
29.85
1                 2            0         1                 0
56.95
2                 0            0         0                 1
53.85
3                 2            2         1                 0
42.30
4                 0            0         0                 1
70.70

   TotalCharges  Churn
0         29.85      0
1       1889.50      0
2        108.15      1
3       1840.75      0
4        151.65      1
```

4. Modelling

The goal for this coursework project is to build a model that can identify customers who are at risk of churning based on the features we have in our dataset. The `churn` column is our target variable and is a binary variable of "Yes" or "No" values. As what we are trying to predict is a binary model and not continuous values e.g. a number, we will use a classification model.

In this section we will build a model that will predict whether a customer will churn or not. We will use the following models and compare their performance:

- KNN Classifier
- Logistic Regression
- Random Forest Classifier

```
df2 = df1.copy()

# Splitting data into features and target
X = df2.drop("Churn", axis=1)
y = df2["Churn"]

X.shape, y.shape

((7032, 12), (7032,))

# Splitting data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.1,
                                                    stratify=y,
```

```
                                                    random_state=42)

X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
((6328, 12), (704, 12), (6328,), (704,))
```

```
# Split train data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
                                                  test_size=0.2,
                                                  stratify=y_train,
                                                  random_state=42)

X_train.shape, X_val.shape, y_train.shape, y_val.shape
```

```
((5062, 12), (1266, 12), (5062,), (1266,))
```

We decided to specify stratify as the target variable `Churn` is imbalanced. This will ensure that the train, validation and test sets have the same proportion of churned and non-churned customers. This will help us avoid any class imbalance issues when modelling.

We split the data into 3 sets as follows:

- Train set: 70% of the data
- Validation set: 20% of the data
- Test set: 10% of the data

This is because we want the train data for training the model, the validation data for tuning the model and the test data for evaluating the model. We will use the validation data to tune the hyperparameters of the model and the test data to evaluate the model.

```
# Weight to deal with imbalanced dataset
class_counts = [5174, 1869]

class_weights = compute_class_weight('balanced', classes=[0, 1],
y=y_train)

class_weight_dict = {0: class_weights[0], 1: class_weights[1]}
class_weight_dict
```

```
{0: 0.680925477535647, 1: 1.8817843866171005}
```

KNN Classifier

We chose KNN classifier as it is a simple model that can be used for classification problems. In addition, it is a non-parametric model which means it doesn't make any assumptions about the data.

```
# Build KNN Model
knn = KNeighborsClassifier()
```

```python
# Fit model to train data
knn.fit(X_train, y_train)

# Make predictions on validation data
knn_base = knn.score(X_val, y_val)
knn_base
```

```
0.7614533965244866
```

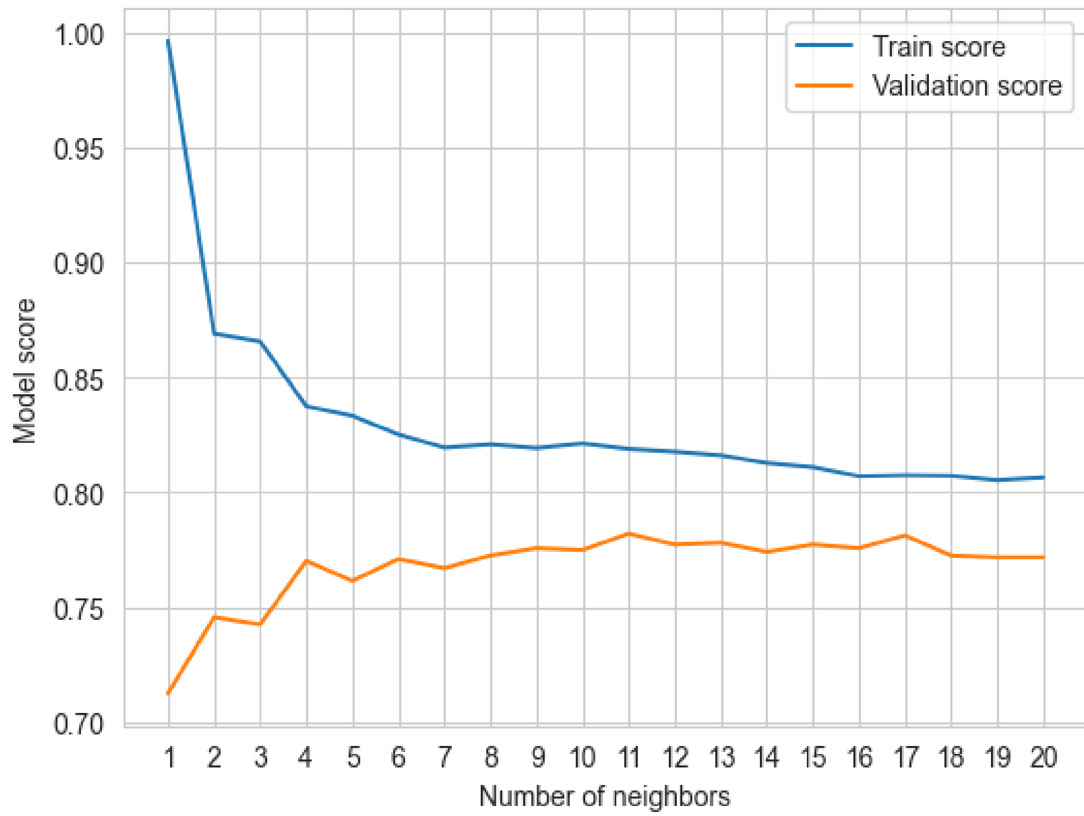
The KNN model has an accuracy of 0.75 on the validation data. This is a good start but we will need to tune the hyperparameters to improve the model.

KNN Hyperparameter Tuning

```python
np.random.seed(1234)
knn_train_scores = []
knn_val_scores = []

# Create a list of different values for n_neighbors
neighbors = range(1, 21) # 1 to 20

# Loop through different neighbors values
for i in neighbors:
    knn.set_params(n_neighbors = i) # set neighbors value

    # Fit the algorithm
    knn.fit(X_train, y_train)

    # Update the  scores
    knn_train_scores.append(knn.score(X_train, y_train))
    knn_val_scores.append(knn.score(X_val, y_val))


knn_train_scores, knn_val_scores
```

```
([0.9964440932437771,
  0.8690241011457922,
  0.8656657447649151,
  0.8374160410904781,
  0.8332674832082181,
  0.8251679178190439,
  0.8194389569340181,
  0.8208218095614381,
  0.8192414065586725,
  0.8212169103121296,
  0.818846305807981,
  0.8176610035559068,
  0.816080600553141,
  0.8127222441722639,
  0.8109442907941525,
  0.8069932832872383,
```

```
  0.8073883840379297,
  0.8071908336625839,
  0.8052153299091268,
  0.8064006321612011],
 [0.7124802527646129,
  0.7456556082148499,
  0.7424960505529226,
  0.7701421800947867,
  0.7614533965244866,
  0.7709320695102686,
  0.7669826224328594,
  0.7725118483412322,
  0.7756714060031595,
  0.7748815165876777,
  0.7819905213270142,
  0.7772511848341233,
  0.778041074249605,
  0.7740916271721959,
  0.7772511848341233,
  0.7756714060031595,
  0.7812006319115324,
  0.7725118483412322,
  0.7717219589257504,
  0.7717219589257504])

# Plotting the train and validation scores
plt.plot(neighbors, knn_train_scores, label="Train score")
plt.plot(neighbors, knn_val_scores, label="Validation score")
plt.xticks(np.arange(1, 21, 1))
plt.xlabel("Number of neighbors")
plt.ylabel("Model score")
plt.legend()

print(f"Maximum KNN score on the test data:
{max(knn_val_scores)*100:.2f}%")
knn_tuned = max(knn_val_scores)

Maximum KNN score on the test data: 78.20%
```




```
# Find the best number of neighbors
np.argmax(knn_val_scores) + 1
```

```
11
```

```
# Evaluate the model on the test data
knn.set_params(n_neighbors = 11)

knn.fit(X_train, y_train)

knn_test_score = knn.score(X_test, y_test)
knn_test_score
```

```
0.7599431818181818
```

Based of the above chart we can see the effect adding more neighbors in the KNN model has on the train and validation scores. We can see that the model performs best with 11 neighbors and the validation score is 78.20%. Upon testing the tuned model on the test data we got a slight decrease in the scores from 78.20% to 75.99%. This is not a significant decrease and we can conclude that the model is not overfitting. However, its performance is not as good as we would like it to be therefore we will try other models.

```
# Dict to keep track of model scores
model_comparison = pd.DataFrame({"model": ["KNN Val", "KNN Test
```

```
Tuned"],
                                                                  "score": [knn_base, knn_test_score]})
model_comparison
```

```
             model     score
0          KNN Val  0.761453
1  KNN Test Tuned  0.759943
```

Logistic Regression

We chose logistic regression as another classification model as it is a simple model to build and it is easy to interpret. In addition, logistic regression are used mainly when dealing with binary data (Raj, 2020) which is what we have in our dataset (Churned or not churned).

```
np.random.seed(1234)

# Build Logistic Regression Model
log_reg = LogisticRegression(max_iter=1000,
                             class_weight=class_weight_dict)

# Fit model to train data
log_reg.fit(X_train, y_train)

# Make predictions on validation data
log_reg_base = log_reg.score(X_val, y_val)
log_reg_base
```

```
0.7496050552922591
```

Our initial logistic regression model has an accuracy score of 74.96% on the validation data which is lower than the KNN model. We'll tune its hyperparameters to see if we can improve the prediction scores.

Logistic Regression Hyperparameter Tuning

```
# Hyperparameter grid for logistic regression
np.random.seed(1234)

log_reg_grid = {"C": np.logspace(-4, 4, 10),
                "solver": ["liblinear", "lbfgs"]}

# Setup gridsearch hyperparameter for LogisticRegression
gs_log_reg = GridSearchCV(LogisticRegression(max_iter=1000),
                          param_grid=log_reg_grid,
                          cv=5,
                          verbose=True)

# Fit hyperparameter search model
gs_log_reg.fit(X_train, y_train)
```

```python
# Find the best hyperparameters
gs_log_reg.best_params_
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
```

```
{'C': 1291.5496650148827, 'solver': 'lbfgs'}
```

```python
# Make predictions on validation data
gs_log_reg.score(X_val, y_val)
```

```
0.7954186413902053
```

```python
# Evaluate the model on the test data
log_reg_tuned = gs_log_reg.score(X_test, y_test)
log_reg_tuned
```

```
0.7670454545454546
```

Based of the above code cells we can see that the tuned logistic regression model has an accuracy score of 79.54% on the validation data. Our model's accuracy score on the validation data improved from 0.7496 to 0.7954 which is a good improvement but not the best.

Upon testing the tuned model on the unseen test data we got an approximate 3% decrease in the accuracy score from 79.54% to 76.70%. This is not a significant decrease and we can conclude that the model is not overfitting. However, its performance is not as good as we would like it to be therefore we will a randomforest classifier before we evaluate our model.

```python
logreg_predictions = gs_rf.predict(X_test)

# Evaluate the model performance on the original test set
lr_accuracy = accuracy_score(y_test, logreg_predictions)
lr_precision = precision_score(y_test, logreg_predictions)
lr_recall = recall_score(y_test, logreg_predictions)
lr_f1 = f1_score(y_test, logreg_predictions)

# Print or use the evaluation metrics as needed
print(f"Accuracy: {lr_accuracy:.4f}")
print(f"Precision: {lr_precision:.4f}")
print(f"Recall: {lr_recall:.4f}")
print(f"F1 Score: {lr_f1:.4f}")
```

```
Accuracy: 0.7486
Precision: 0.5210
Recall: 0.6631
F1 Score: 0.5835
```

We can see that our tuned logistic regression has an accuracy score of 74.86% and a recall scor of 0.66. This means that our model correctly predicted 66% of the customers who churned. This is not ideal as we want to correctly predict as many customers who churned as possible.

```
# Add base and tuned scores to model comparison dataframe
model_comparison = pd.concat([model_comparison,
                              pd.DataFrame({"model": "Logistic Reg
Val",
                                            "score": log_reg_base},
index=[0])], ignore_index=True)

model_comparison = pd.concat([model_comparison,
                              pd.DataFrame({"model": "Logistic Reg
Test Tuned",
                                            "score": log_reg_tuned},
index=[0])], ignore_index=True)
model_comparison
```

	model	score
0	KNN Val	0.761453
1	KNN Test Tuned	0.759943
2	Logistic Reg Val	0.749605
3	Logistic Reg Test Tuned	0.767045

Upon comparing our tests results between the KNN and Logistic Regression, we can see that the KNN performed better by approximately 2% which is unexpected as logistic regression is a more powerful model. However, we will try another model to see if we can improve the prediction scores.

Random Forest Classifier

We chose random forest classifier as another classification model as it is a powerful model that can be used for classification problems. In addition, it is an ensemble model which means it combines multiple models to improve the performance of the model.

```
# Build Random Forest Classifier Model
np.random.seed(1234)
rf = RandomForestClassifier(class_weight=class_weight_dict)

# Fit model to train data
rf.fit(X_train, y_train)

# Make predictions on validation data
rf_base = rf.score(X_val, y_val)
rf_base
```

```
0.7748815165876777
```

Our initial random forest classifier has a validation accuracy of 77.48% which is good for a base model. We'll tune its hyperparameters to see if we can improve the prediction scores.

Random Forest Classifier Hyperparameter Tuning

```
# Hyperparameter grid for random forest classifier
np.random.seed(1234)

rf_grid = {"n_estimators": np.arange(10, 101, 10),
           "max_depth": [None, 3, 5, 10],
           "min_samples_split": np.arange(2, 11, 2),
           "min_samples_leaf": np.arange(1, 11, 2)}

# Setup gridsearch hyperparameter for RandomForestClassifier
gs_rf = GridSearchCV(RandomForestClassifier(),
                     param_grid=rf_grid,
                     cv=3,
                     verbose=True)

# Fit hyperparameter search model
gs_rf.fit(X_train, y_train)

# Find the best hyperparameters
gs_rf.best_params_
```

```
Fitting 3 folds for each of 1000 candidates, totalling 3000 fits
```

```
{'max_depth': 10,
 'min_samples_leaf': 9,
 'min_samples_split': 10,
 'n_estimators': 60}
```

```
# Make predictions on validation data
gs_rf.score(X_val, y_val)
```

```
0.7962085308056872
```

```
# Evaluate the model on the test data
rf_tuned = gs_rf.score(X_test, y_test)
rf_tuned
```

```
0.7826704545454546
```

Based of the above code cells, we can see that the base model had an accuracy score of 77.48% on the validation data. After hyperparameter tuning, the model's accuracy score increased slightly to 79.62% which is better than the previous models but below expectations. However, the test accuracy score was 78.26% which is good as there's only a 1% decrease in the models predictions with unseen data.

```
predictions = gs_rf.predict(X_test)

# Evaluate the model performance
accuracy = accuracy_score(y_test, predictions)
precision = precision_score(y_test, predictions)
```

```
recall = recall_score(y_test, predictions)
f1 = f1_score(y_test, predictions)

# Print or use the evaluation metrics as needed
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.7827
Precision: 0.6197
Recall: 0.4706
F1 Score: 0.5350
```

Based of the above scores, we can see that the random forest does worse than the logistic regression in predicting the class 1 labels as it has a recall score of 0.4706. This could be due to the model not being sensitive enough to the class 1 labels. However, the model has a better accuracy and precision than the logistic regression model.

```
## Add base and tuned scores to model comparison dataframe
model_comparison = pd.concat([model_comparison,
                              pd.DataFrame({"model": "Random Forest 
Val",
                                            "score": rf_base},
index=[0])], ignore_index=True)

model_comparison = pd.concat([model_comparison,
                                pd.DataFrame({"model": "Random Forest 
Test Tuned",
                                                "score": rf_tuned},
index=[0])], ignore_index=True)

model_comparison
```

	model	score
0	KNN Val	0.761453
1	KNN Test Tuned	0.759943
2	Logistic Reg Val	0.749605
3	Logistic Reg Test Tuned	0.767045
4	Random Forest Val	0.774882
5	Random Forest Test Tuned	0.782670

Looking at the above dataframe we can see taht the Random Forest performed best with unseen data with an accuracy score of 78.26% which is good but not ideal with respects to our success criteria. However, we will use the tuned random forest classifier model to evaluate our model.

5. Evaluation

In this section we will evaluate the models we built in the previous section. We will evaluate the models based on the following metrics:

- ROC AUC
- Confusion Matrix
- Classification Report

As the tuned random forest classifier model had the best performance on the validation data, we will use it to evaluate our model.

ROC AUC

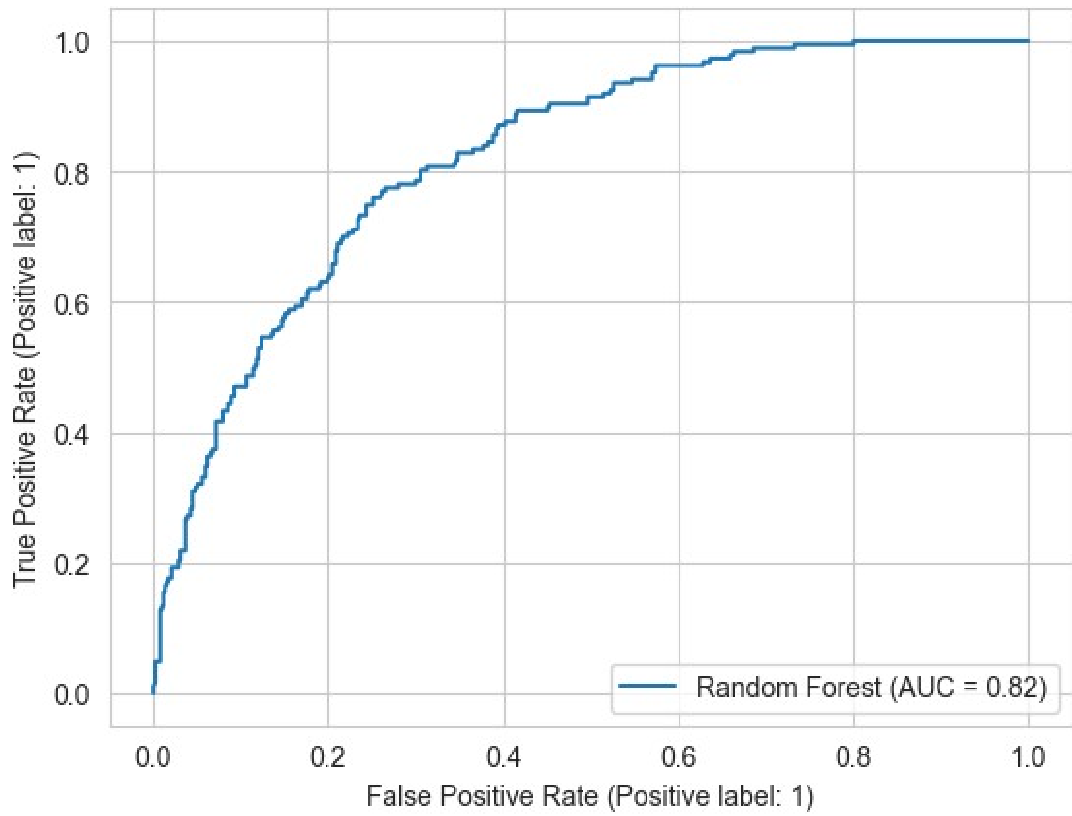
We will use the Roc curve to evaluate the performance of our model by comparing the true positive rate to the false positive rate.

True positive rate (TPR) is the proportion of positive data points that are correctly predicted as positive. Fasle positive rate (FPR) is the proportion of negative data points that are incorrectly predicted as positive.

```
gs_preds = gs_rf.predict_proba(X_test)[:, 1]
gs_preds[:10]
```

```
array([0.41991635, 0.04582353, 0.12964794, 0.35720959, 0.07005916,
       0.6402615 , 0.16409445, 0.55633729, 0.26177402, 0.36797015])
```

```
roc_display = RocCurveDisplay.from_estimator(gs_rf, X_test, y_test,
name="Random Forest")
```

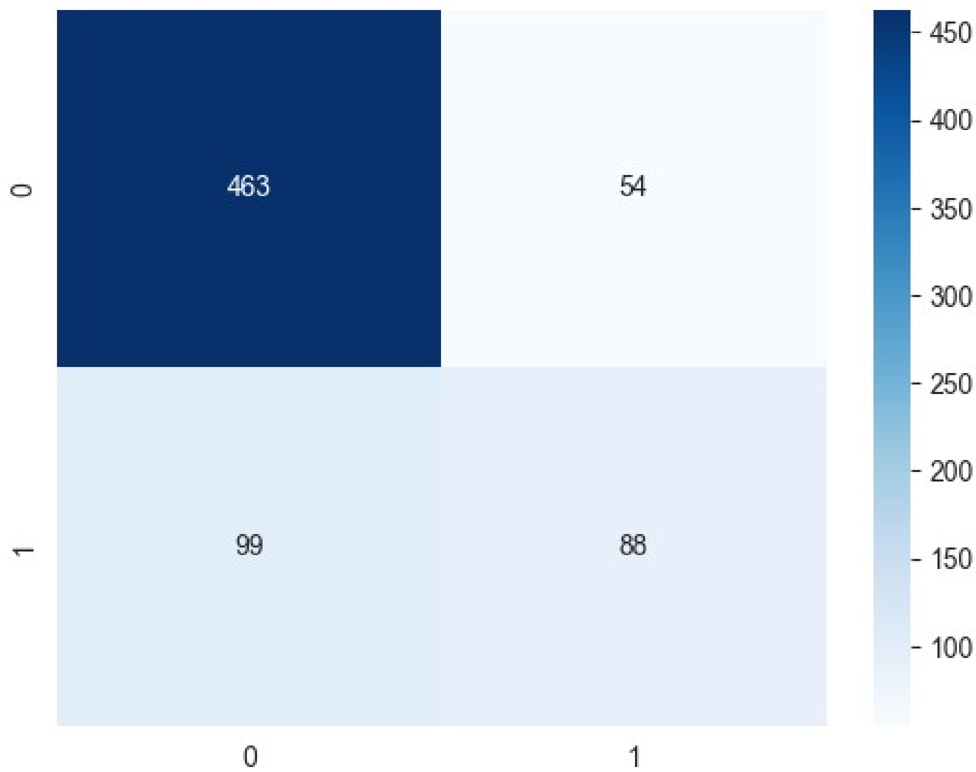


From the above ROC curve we can see that our model has an AUC score of 0.82 which is good however the closer the curve would have been to the y-axis the better the model would have been. We can make the following conclusions from the ROC curve:

- The ideal point lies approximately at 0.82 TPR, 0.3 FPR
- The model achieves a relatively high TPR which is good, however, the FPR is not too high but it shows that there are weaknesses in the models predictions of some classes. This could be due to the class imbalance in the dataset, which will be analyzed further in the confusion matrix and classification report.

Confusion Matrix

```
# Plotting confusion matrix

conf_matrix = confusion_matrix(y_test, gs_preds.round())

sns.heatmap(conf_matrix, annot=True, fmt="g", cmap="Blues")

<Axes: >
```



Based of the confusion matrix above, we can see that the model perfroms exceedingly well in predicting the true negatives with 463 correct predictions. However when analyzing the class 1 predictions, we can see the model misclassified 99 labels and got 88 accurate. This could be due to our models weightage not being able to deal with the class imbalance in the dataset properly.

Classification Report

```
# Classification report
print(classification_report(y_test, gs_preds.round()))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86       517
           1       0.62      0.47      0.53       187

    accuracy                           0.78       704
   macro avg       0.72      0.68      0.70       704
weighted avg       0.77      0.78      0.77       704
```

The classification report adds more insightful information on top of the ROC curve and confusion matrix. We can see that the model has a precision score of 0.82 in predicting the class 0 which is quite good. The model also has a recall score of 0.90 which is really good as it captures 90% of the positive cases.

However, the class 1 underperforms and has a precision score of 0.62 and a recall score of 0.47. This is not good as the model only captures 47% of all positive cases. This could be due to the class imbalance and the model not being sensitive enough to the class 1 labels.

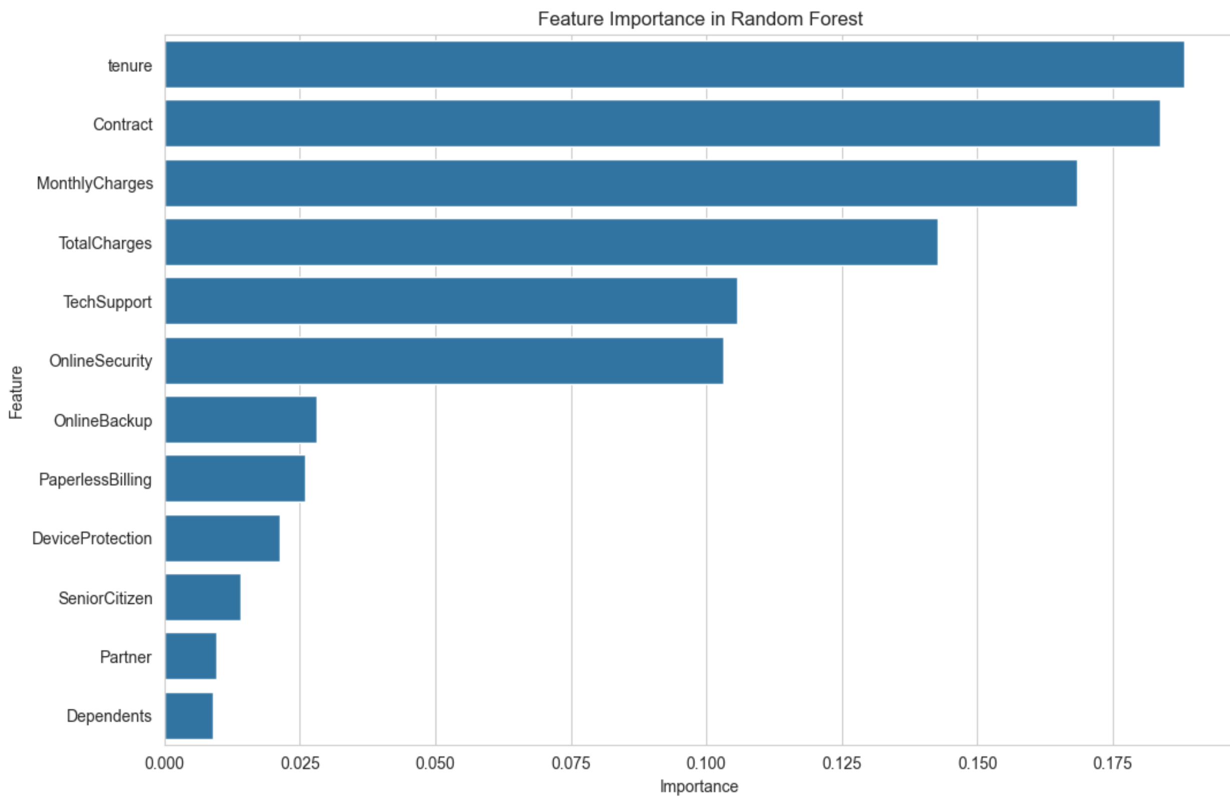
Feature Importance

In this section we will look at the most important features in our model. This will help us understand which features are most important in predicting whether a customer will churn or not.

```
rf_classifier = gs_rf.best_estimator_

rf_classifier.fit(X_train, y_train)

feature_importances = rf_classifier.feature_importances_

#DataFrame to display feature importances
feature_importance_df = pd.DataFrame({'Feature': X.columns,
'Importance': feature_importances})
feature_importance_df =
feature_importance_df.sort_values(by='Importance', ascending=False)

print(feature_importance_df)
```

```
             Feature  Importance
3             tenure    0.188252
8           Contract    0.183782
10    MonthlyCharges    0.168315
11      TotalCharges    0.142606
7        TechSupport    0.105687
4     OnlineSecurity    0.103073
5       OnlineBackup    0.028233
9   PaperlessBilling    0.025995
6   DeviceProtection    0.021367
0      SeniorCitizen    0.014023
1            Partner    0.009583
2         Dependents    0.009085
```

```
plt.figure(figsize=(12, 8))

# bar plot of feature importances
sns.barplot(x='Importance', y='Feature', data=feature_importance_df)

plt.title('Feature Importance in Random Forest')
plt.xlabel('Importance')
plt.ylabel('Feature');
```



From the above feature importance chart we can see that the most important feature in our model is `tenure` which is the total months a customer has been with the telco company. This is to be expected as the longer a customer stays with a company, the more likely they won't churn. This is closely followed by `Contract` and `MonthlyCharges`. However, the least importance features in our model which when removed may have improved its predictive power are `Partner`, `Dependents` and `SeniorCitizen`.

6. Conclusion

Our model has an accuracy score of 78% which is good but not ideal with respects to our success criteria. However, the model has a high recall score of 0.90 in capturing the class 0 labels which is good. The model, unfortunately, underperforms in predicting the class 1 labels with a recall score of 0.47. This could be due to the class imbalance in the dataset and the model not being sensitive enough to the class 1 labels.

For future work, we could try to improve the model by:

- Understanding the most important features when it comes to measuring churn from the telco company by contacting a domain expert.
- Using a different weightage to deal with the class imbalance in the dataset.
- Using a different model that is more sensitive to the class 1 labels.

Collecting additional information from the customers such as location data, such as state and city could help us understand which areas have a higher churn rate and why. In addition, collecting information on the time period e.g. year and quarter could help us understand if there are any seasonal trends in the data.

With regards to the project objectives, our model prediction accuracy slightly fell short of the 80% accuracy target. However, the class 0 recall score of 0.90 is above the 0.80 target which is good. The class 1 recall score of 0.47 is below the 0.80 target which is not good. Therefore, we can conclude that our model, based of the objectives set earlier, is not good enough to predict whether a customer will churn or not.

Limitations

The dataset we used for this project was quite small with only 7,043 rows and 21 columns. This could have affected the performance of our model as it may not have been able to learn enough from the data.

In addition, the dataset was imbalanced with 5,100 of the customers not churning and 1,800 churning. This could have affected the performance of our model as it may not have been able to learn enough from the data. However, real world data is also imbalanced and we need to find ways to deal with this issue.

7. References

Frankenfield, J., 2022. Churn Rate: What It Means, Examples, and Calculations. [Online] Available at: https://www.investopedia.com/terms/c/churnrate.asp [Accessed 11 January 2024].

Raj, A., 2020. Perfect Recipe for Classification Using Logistic Regression. [Online] Available at: https://towardsdatascience.com/the-perfect-recipe-for-classification-using-logistic-regression-f8648e267592#:~:text=Logistic%20regression%20is%20easier%20to,as%20indicators%20of%20feature%20importance. [Accessed 14 January 2024].

Smart Vision, 2020. What is the CRISP-DM methodology?. [Online] Available at: https://www.sv-europe.com/crisp-dm-methodology/#:~:text=CRISP%2DDM%20stands%20for%20cross,We%20did%20not%20invent%20it. [Accessed 11 January 2024].

Socket, n.d. Fiber vs DSL: What's the Difference?. [Online] Available at: https://www.socket.net/blog/posts/fiber-tech/fiber-vs-dsl-whats-difference#:~:text=It's%20simple%20%E2%80%94%20fiber%20is%20fast,does%20that%20mean%20for%20you%3F [Accessed 13 January 2024].

Wagh, S. K. et al., 2024. Customer churn prediction in telecom sector using machine learning techniques. Results in Control and Optimization, Volume 14.