TELECOM CUSTOMER CHURN PREDICTION

Yipeng CHEN

Figure 1: Customer Churn

Did you know that attracting a new customer costs five times as much as keeping an existing one?

Initially,
I would like to express my gratitude to the following Kaggle notebooks that have served as a source of inspiration for the creation of this report:

CUSTOMER CHURN PREDICTION - bhartiprasad17

Telecom Churn Prediction - bandiatindra

Contents

1. Introduction

What is Customer Churn?

Customer churn or customer attrition is defined as when customers discontinue using a company's product or service.

Individualized customer retention is tough because most firms have a large number of customers and can't afford to devote much time to each of them. The costs would be too great, outweighing the additional revenue. However, if a corporation could forecast which customers are likely to leave ahead of time, it could focus customer retention efforts only on these "high risk" clients. The ultimate goal is to expand its coverage area and retrieve more customers loyalty.

Customer churn is a critical metric because it is much less expensive to retain existing customers than it is to acquire new customers. Moreover, customer churn is a giant business killer. Even small increases in churn can cause a significant cut in revenues.

To reduce customer churn, companies need to predict which customers are at high risk of churn.

To detect early signs of potential churn, one must first develop a holistic view of the customers and their interactions across numerous channels, including store/branch visits, product purchase histories, customer service calls, web-based transactions, and social media interactions, to mention a few.

As a result, by addressing churn, these businesses may not only preserve their market position, but also grow and thrive. More customers they have in their network, the lower the cost of initiation and the larger the profit. As a result, the company's key focus for success is reducing client attrition and implementing effective retention strategy.

1.1 About Dataset

Context

IBM Sample Data Sets

"Predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs."

Content

Each row represents a customer, each column contains customer's attributes.

The data set includes information about:

a. Customers who left within the last month – the column is called Churn

b. Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies

c. Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges

d. Demographic info about customers – gender, age range, and if they have partners and dependents

The raw data contains 7043 rows (customers) and 21 columns (features).

The "Churn" column is our target.

Columns description

Column_Name	Column_Description
customerID	Customer ID
gender	Whether the customer is a male or a female
SeniorCitizen	Whether the customer is a senior citizen or not (1, 0)
Partner	Whether the customer has a partner or not (Yes, No)
Dependents	Whether the customer has dependents or not (Yes, No)
tenure	Number of months the customer has stayed with the company
PhoneService	Whether the customer has a phone service or not (Yes, No)
MultipleLines	Whether the customer has multiple lines or not (Yes, No, No phone service)
InternetService	Customer's internet service provider (DSL, Fiber optic, No)
OnlineSecurity	Whether the customer has online security or not (Yes, No, No internet service)
OnlineBackup	Whether the customer has online backup or not (Yes, No, No internet service)
DeviceProtection	Whether the customer has device protection or not (Yes, No, No internet service)
TechSupport	Whether the customer has tech support or not (Yes, No, No internet service)
StreamingTV	Whether the customer has streaming TV or not (Yes, No, No internet service)
StreamingMovies	Whether the customer has streaming movies or not (Yes, No, No internet service)
Contract	The contract term of the customer (Month-to-month, One year, Two year)
PaperlessBilling	Whether the customer has paperless billing or not (Yes, No)
PaymentMethod	The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
MonthlyCharges	The amount charged to the customer monthly
TotalCharges	The total amount charged to the customer
Churn	Whether the customer churned or not (Yes or No)

2. Loading libraries and data

```python
import pandas as pd
import numpy as np
import missingno as msno
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

```python
# Loading data
df = pd.read_csv('Telco-Customer-Churn.csv')
```

3. Undertanding the data

```
df
```

```
##       customerID  gender  SeniorCitizen  ... MonthlyCharges TotalCharges  Churn
## 0     7590-VHVEG  Female              0  ...          29.85        29.85     No
## 1     5575-GNVDE    Male              0  ...          56.95       1889.5     No
## 2     3668-QPYBK    Male              0  ...          53.85       108.15    Yes
## 3     7795-CFOCW    Male              0  ...          42.30      1840.75     No
## 4     9237-HQITU  Female              0  ...          70.70       151.65    Yes
## ...          ...     ...            ...  ...            ...          ...    ...
## 7038  6840-RESVB    Male              0  ...          84.80       1990.5     No
## 7039  2234-XADUH  Female              0  ...         103.20       7362.9     No
## 7040  4801-JZAZL  Female              0  ...          29.60       346.45     No
## 7041  8361-LTMKD    Male              1  ...          74.40        306.6    Yes
## 7042  3186-AJIEK    Male              0  ...         105.65       6844.5     No
##
## [7043 rows x 21 columns]
```

```
df.shape
```

```
## (7043, 21)
```

```
df.columns.values
```

```
## array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
##        'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
##        'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
##        'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
##        'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
##        'TotalCharges', 'Churn'], dtype=object)
```

```
df.info()
```

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

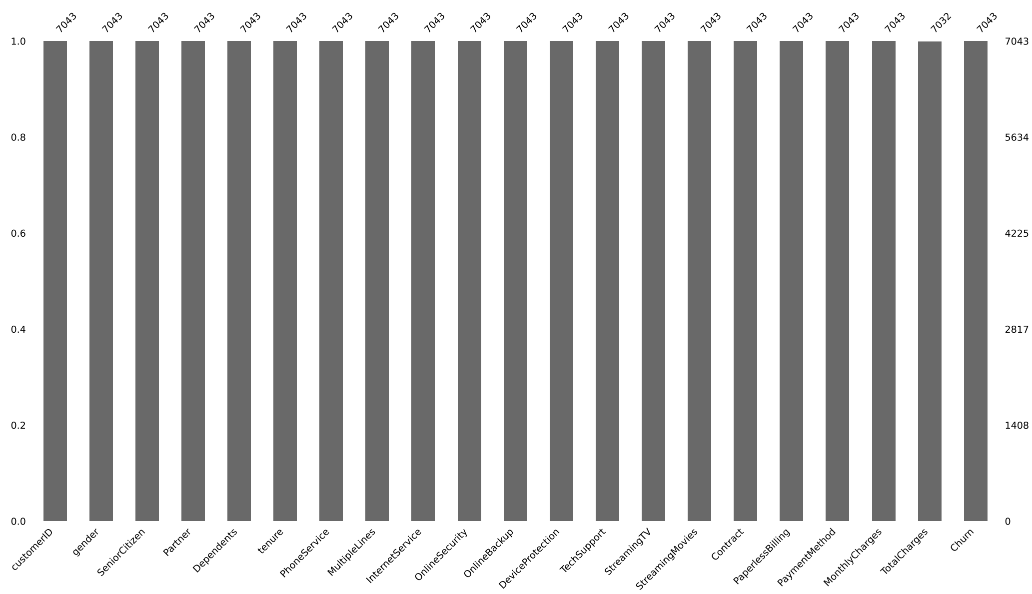
```
df.dtypes
```

```
## 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
```

```
# Converting "TotalCharges" to a numerical data type
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
```

4. Visualize missing values

```
# Visualize missing values as a bar plot
msno.bar(df,fontsize=11,figsize=(20,10))
```



From the above visualization,
we can observe that there are 11 missing values for “TotalCharges”.

```python
# Calculate the number of missing values in each column of the DataFrame
df.isnull().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        11
## Churn                0
## dtype: int64
```

5. Data Manipulation

```
# Remove customer IDs from the data set
df = df.drop(['customerID'], axis = 1)
df
```

```
##       gender  SeniorCitizen Partner  ... MonthlyCharges  TotalCharges Churn
## 0     Female              0     Yes  ...          29.85         29.85    No
## 1       Male              0      No  ...          56.95       1889.50    No
## 2       Male              0      No  ...          53.85        108.15   Yes
## 3       Male              0      No  ...          42.30       1840.75    No
## 4     Female              0      No  ...          70.70        151.65   Yes
## ...      ...            ...     ...  ...            ...           ...   ...
## 7038    Male              0     Yes  ...          84.80       1990.50    No
## 7039  Female              0     Yes  ...         103.20       7362.90    No
## 7040  Female              0     Yes  ...          29.60        346.45    No
## 7041    Male              1     Yes  ...          74.40        306.60   Yes
## 7042    Male              0      No  ...         105.65       6844.50    No
##
## [7043 rows x 20 columns]
```

```
# We know that the "TotalCharges" has 11 missing values. Let's check this :
df[np.isnan(df['TotalCharges'])]
```

```
##       gender  SeniorCitizen Partner  ... MonthlyCharges  TotalCharges Churn
## 488   Female              0     Yes  ...          52.55           NaN    No
## 753     Male              0      No  ...          20.25           NaN    No
## 936   Female              0     Yes  ...          80.85           NaN    No
## 1082    Male              0     Yes  ...          25.75           NaN    No
## 1340  Female              0     Yes  ...          56.05           NaN    No
## 3331    Male              0     Yes  ...          19.85           NaN    No
## 3826    Male              0     Yes  ...          25.35           NaN    No
## 4380  Female              0     Yes  ...          20.00           NaN    No
## 5218    Male              0     Yes  ...          19.70           NaN    No
## 6670  Female              0     Yes  ...          73.35           NaN    No
## 6754    Male              0      No  ...          61.90           NaN    No
##
## [11 rows x 20 columns]
```

```python
# Now, Let us remove these 11 rows from our data set :
df.dropna(inplace = True)
df.isnull().sum()
```

```
## 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
```

```python
df.shape
```

```
## (7032, 20)
```

```
df["SeniorCitizen"]= df["SeniorCitizen"].map({0: "No", 1: "Yes"})
df.head()
```

```
##    gender SeniorCitizen Partner  ... MonthlyCharges  TotalCharges Churn
## 0  Female            No     Yes  ...          29.85         29.85    No
## 1    Male            No      No  ...          56.95       1889.50    No
## 2    Male            No      No  ...          53.85        108.15   Yes
## 3    Male            No      No  ...          42.30       1840.75    No
## 4  Female            No      No  ...          70.70        151.65   Yes
##
## [5 rows x 20 columns]
```

```
df["InternetService"].describe(include=['object', 'bool'])
```

```
## count            7032
## unique              3
## top       Fiber optic
## freq             3096
## Name: InternetService, dtype: object
```

```
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_cols].describe()
```

```
##             tenure  MonthlyCharges  TotalCharges
## count  7032.000000     7032.000000   7032.000000
## mean     32.421786       64.798208   2283.300441
## std      24.545260       30.085974   2266.771362
## min       1.000000       18.250000     18.800000
## 25%       9.000000       35.587500    401.450000
## 50%      29.000000       70.350000   1397.475000
## 75%      55.000000       89.862500   3794.737500
## max      72.000000      118.750000   8684.800000
```

6. Data Visualization

```
# g_labels = ['Male', 'Female']
# c_labels = ['No', 'Yes']
#
# # Create subplots: use 'domain' type for Pie subplot
# fig = make_subplots(rows=1, cols=2,
#                      specs=[[{'type':'domain'}, {'type':'domain'}]])
# fig.add_trace(go.Pie(labels=g_labels,
#                       values=df['gender'].value_counts(), name="Gender"),
#                       1, 1)
# fig.add_trace(go.Pie(labels=c_labels,
#                       values=df['Churn'].value_counts(), name="Churn"),
#                       1, 2)
#
# # Use `hole` to create a donut-like pie chart
# fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)
#
# fig.update_layout(
#     title_text="Gender and Churn Distributions",
#     # Add annotations in the center of the donut pies.
#     annotations=[dict(text='Gender', x=0.16, y=0.5,
#                  font_size=20, showarrow=False),
#                  dict(text='Churn', x=0.84, y=0.5,
#                  font_size=20, showarrow=False)])
# fig.show()
```

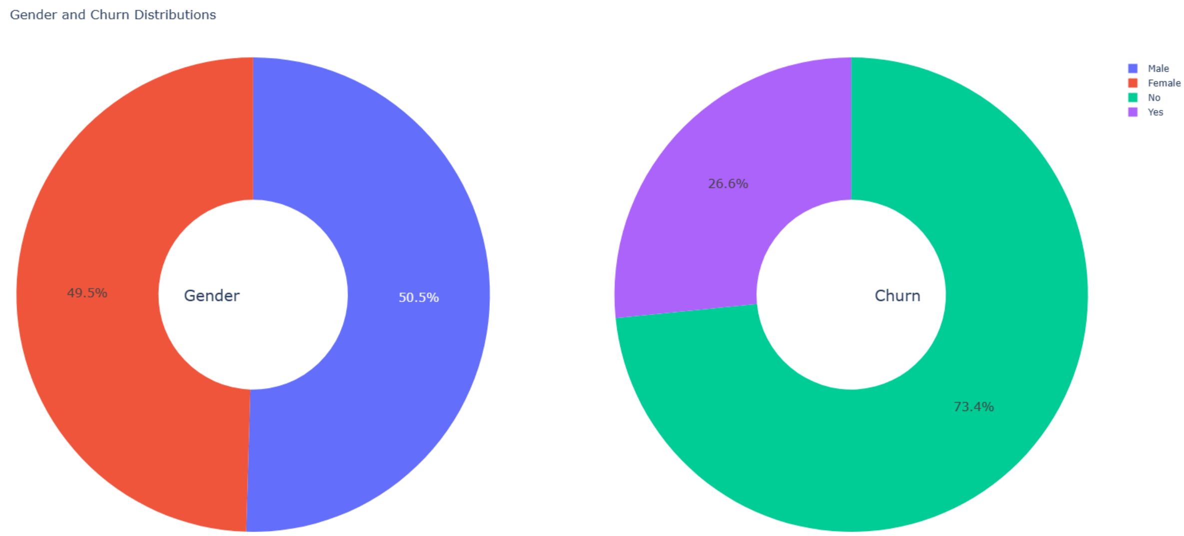


Figure 2: Gender and Churn Distributions

26.6 % of customers switched to another firm.

Customers are 49.5 % female and 50.5 % male.

```
df["Churn"][df["Churn"]=="No"].groupby(by=df["gender"]).count()
```

```
## gender
## Female    2544
## Male      2619
## Name: Churn, dtype: int64
```

```
df["Churn"][df["Churn"]=="Yes"].groupby(by=df["gender"]).count()
```

```
## gender
## Female    939
## Male      930
## Name: Churn, dtype: int64
```

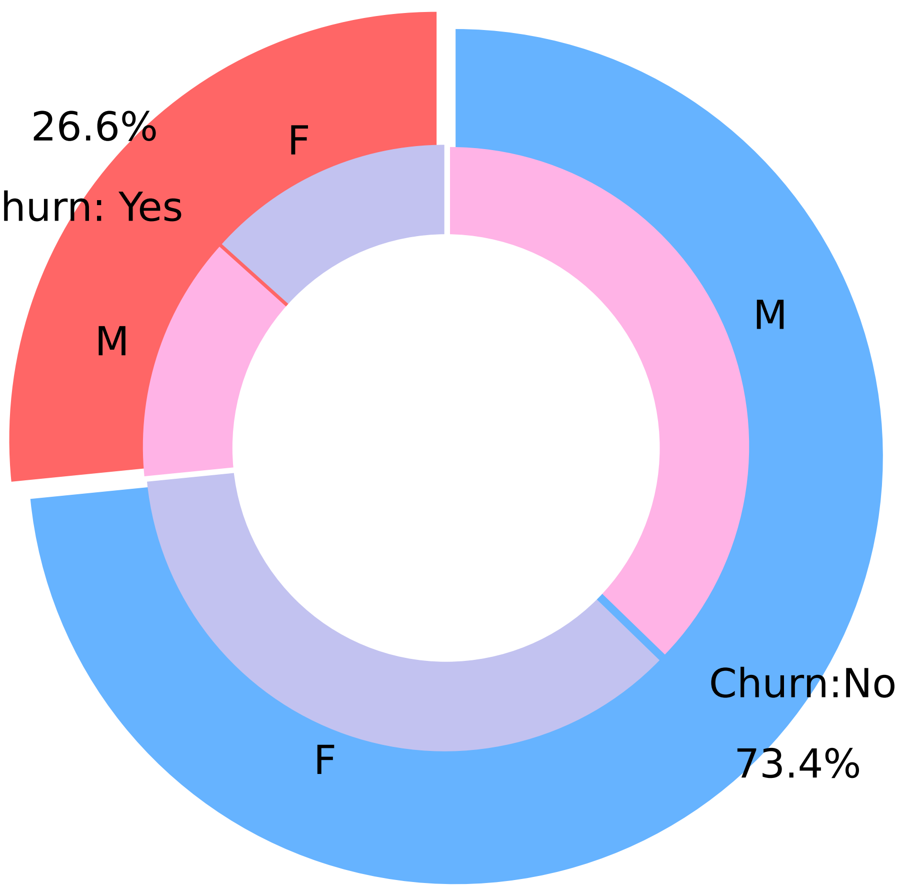
```python
plt.figure(figsize=(6, 6))
labels =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',
pctdistance=1.08, labeldistance=0.8,colors=colors, startangle=90,frame=True,
explode=explode,radius=10, textprops =textprops, counterclock = True, )
```

```python
plt.pie(sizes_gender,labels=labels_gender,
colors=colors_gender,startangle=90, explode=explode_gender,radius=7,
textprops =textprops, counterclock = True, )
#Draw circle
```

```python
centre_circle = plt.Circle((0,0),5,color='black', fc='white',linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.title('Churn Distribution w.r.t Gender: Male(M), Female(F)',
fontsize=15, y=1.1)

# show plot
plt.axis('equal')
```

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

Churn Distribution w.r.t Gender: Male(M), Female(F)



There is negligible difference in customer percentage.
Both genders behaved in similar fashion when it comes to migrating to another service provider.

```
# fig = px.histogram(df, x="Churn", color="Contract", barmode="group",
#                    title="<b>Customer contract distribution<b>")
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

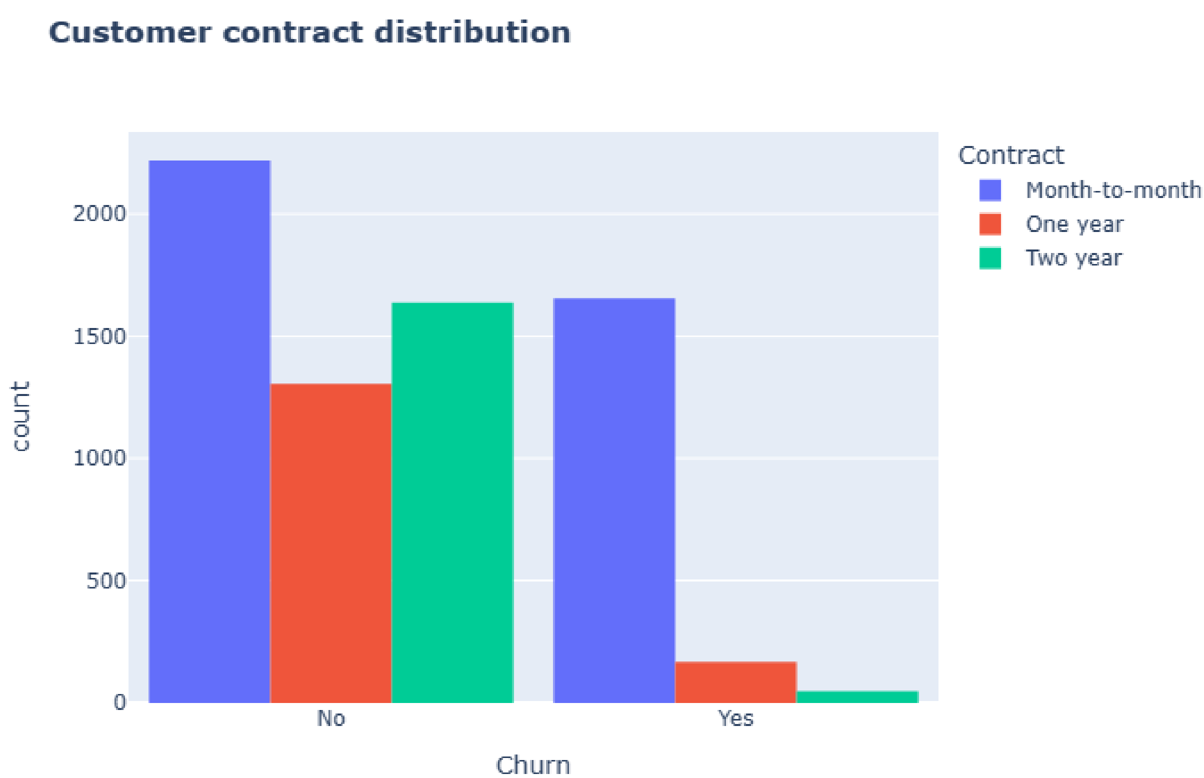


Figure 3: Customer contract distribution

About 88% of customers with Month-to-Month Contract opted to move out compared to 9% of customers with One Year Contract and 3% with Two Year Contract.

```
# labels = df['PaymentMethod'].unique()
# values = df['PaymentMethod'].value_counts()
#
# fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3)])
# fig.update_layout(title_text="<b>Payment Method Distribution</b>")
# fig.show()
```

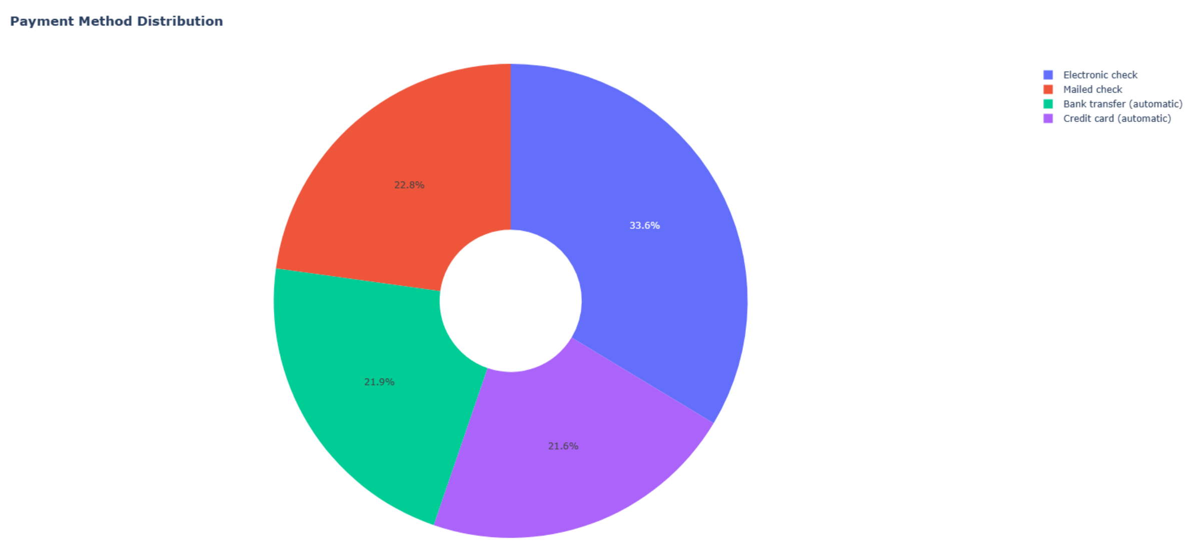


Figure 4: Payment Method Distribution

```python
# fig = px.histogram(df, x="Churn", color="PaymentMethod",
#       title="<b>Customer Payment Method distribution w.r.t. Churn</b>")
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

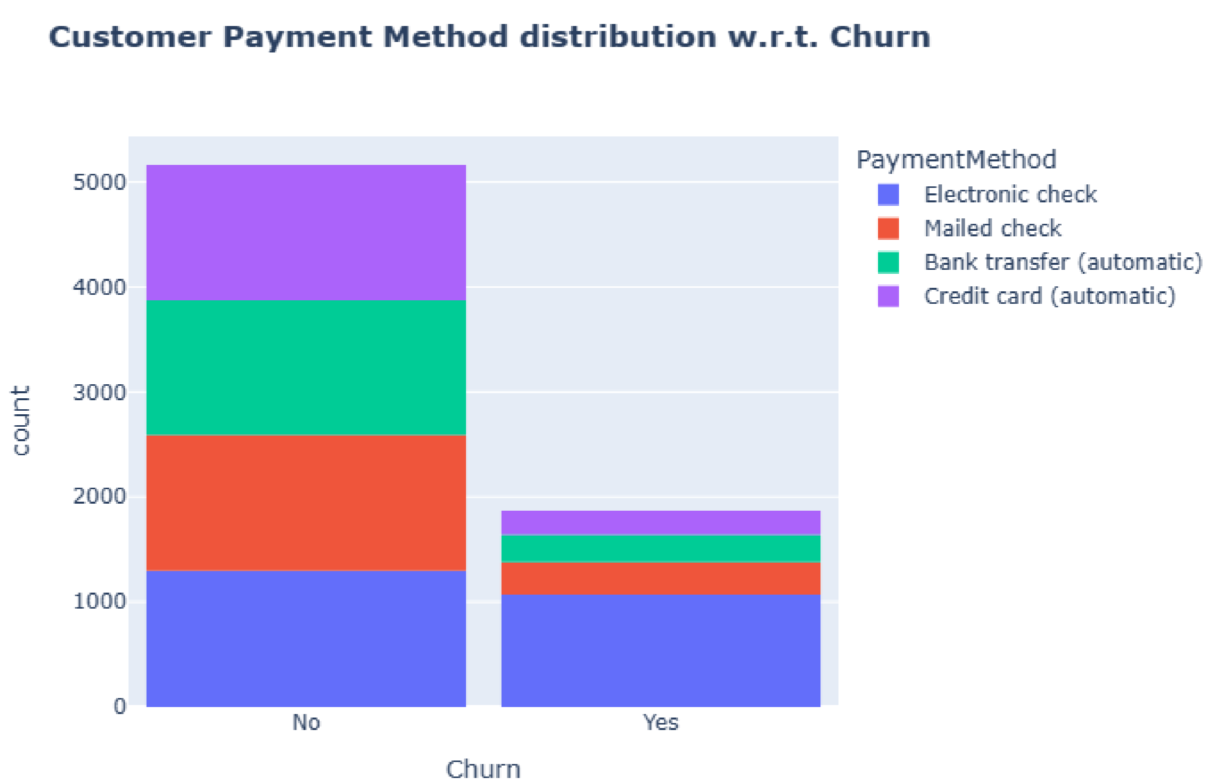


Figure 5: Customer Payment Method distribution w.r.t. Churn

Major customers who moved out were having Electronic Check as Payment Method.

Customers who opted for Credit-Card automatic transfer or Bank Automatic Transfer and Mailed Check as Payment Method were less likely to move out.

```
df["InternetService"].unique()
```

```
## array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

```
df[df["gender"]=="Male"][["InternetService", "Churn"]].value_counts()
```

```
## InternetService  Churn
## DSL              No       992
## Fiber optic      No       910
## No               No       717
## Fiber optic      Yes      633
## DSL              Yes      240
## No               Yes       57
## dtype: int64
```

```
df[df["gender"]=="Female"][["InternetService", "Churn"]].value_counts()
```

```
## InternetService  Churn
## DSL              No       965
## Fiber optic      No       889
## No               No       690
## Fiber optic      Yes      664
## DSL              Yes      219
## No               Yes       56
## dtype: int64
```

A lot of customers choose the Fiber optic service and it's also evident that the customers who use Fiber optic have high churn rate, this might suggest a dissatisfaction with this type of internet service.

```
# fig = go.Figure()
# fig.add_trace(go.Bar(
#   x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
#        ["Female", "Male", "Female", "Male"]],
#   y = [965, 992, 219, 240],
#   name = 'DSL',
# ))
# fig.add_trace(go.Bar(
#   x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
#        ["Female", "Male", "Female", "Male"]],
#   y = [889, 910, 664, 633],
#   name = 'Fiber optic',
# ))
# fig.add_trace(go.Bar(
#   x = [['Churn:No', 'Churn:No', 'Churn:Yes', 'Churn:Yes'],
#        ["Female", "Male", "Female", "Male"]],
#   y = [690, 717, 56, 57],
#   name = 'No Internet',
# ))
# fig.update_layout(
#   title_text="<b>Churn Distribution w.r.t. Internet Service and Gender</b>")
# fig.show()
```

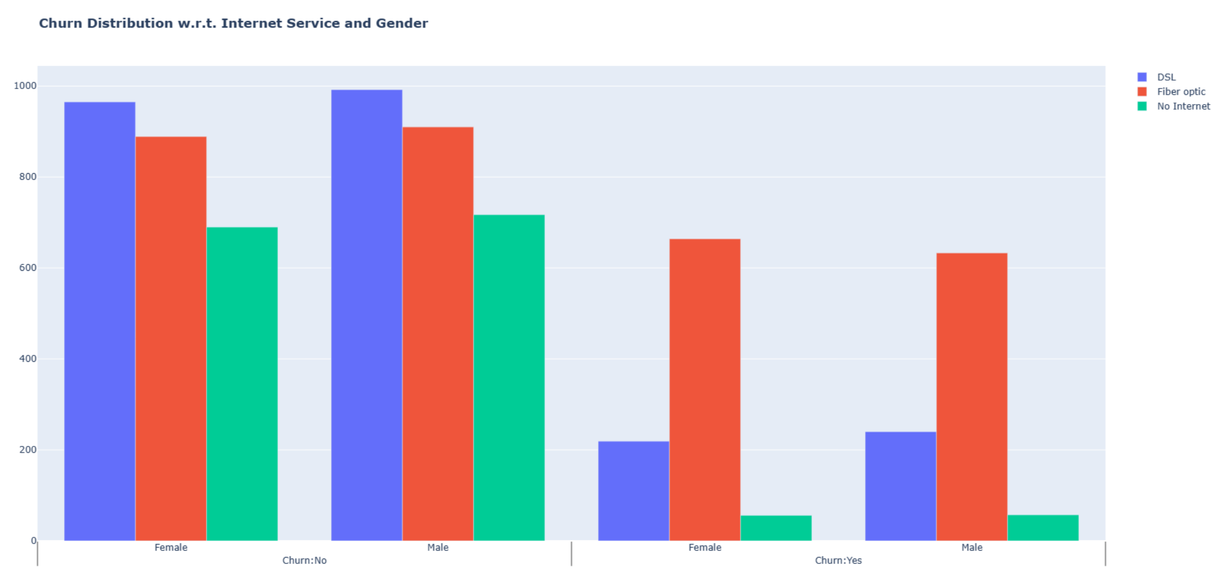


Figure 6: Churn Distribution w.r.t. Internet Service and Gender

```
# color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
# fig = px.histogram(df, x="Churn", color="Dependents",
#       barmode="group", title="<b>Dependents distribution</b>",
#       color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

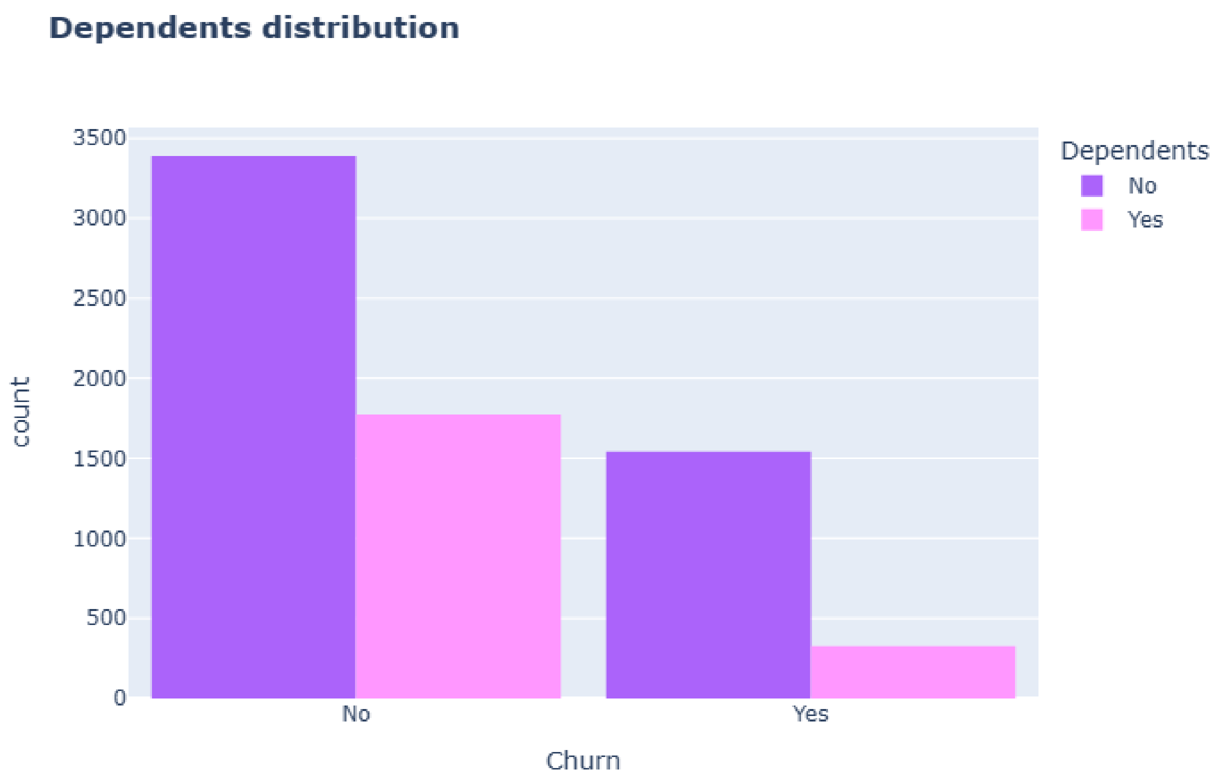


Figure 7: Dependents distribution

Customers without dependents are more likely to churn.

```
# color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
# fig = px.histogram(df, x="Churn", color="Partner",
#        barmode="group", title="<b>Chrun distribution w.r.t. Partners</b>",
#        color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

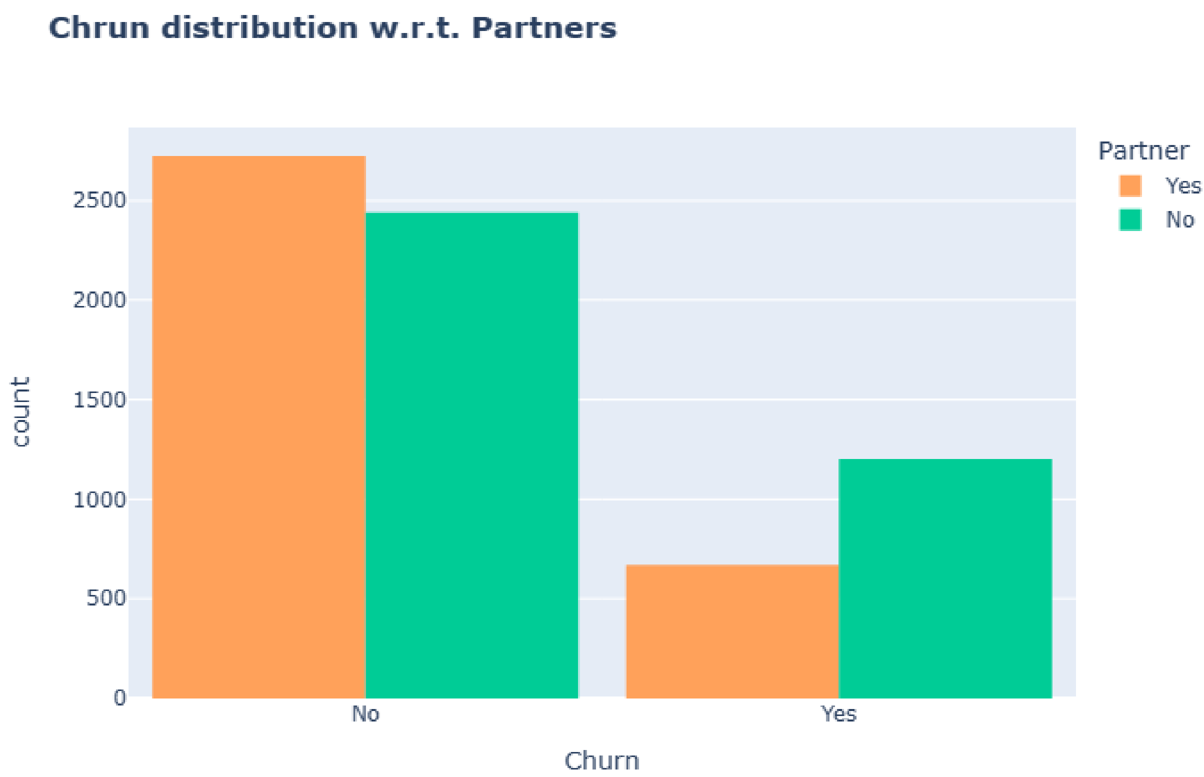


Figure 8: Chrun distribution w.r.t. Partners

Customers that doesn't have partners are more likely to churn.

```
# color_map = {"Yes": '#00CC96', "No": '#B6E880'}
# fig = px.histogram(df, x="Churn", color="SeniorCitizen",
#       title="<b>Chrun distribution w.r.t. Senior Citizen</b>",
#       color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

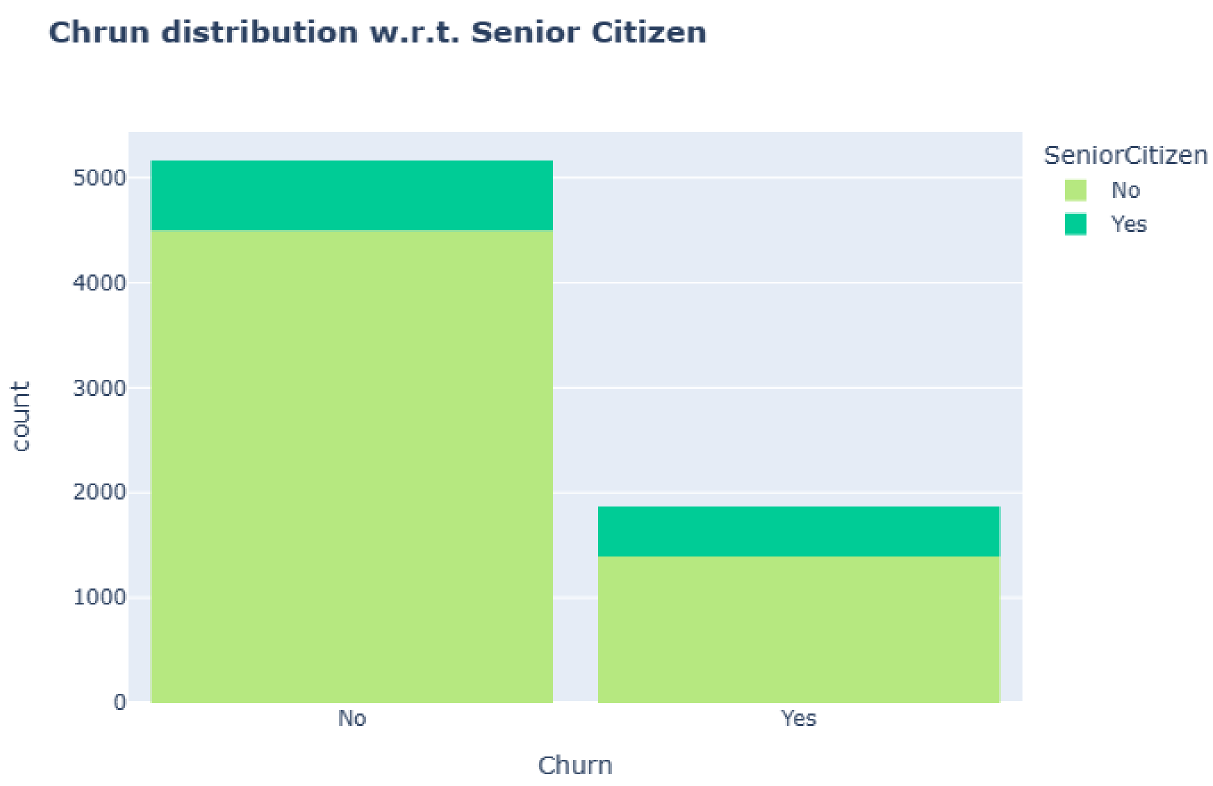


Figure 9: Chrun distribution w.r.t. Senior Citizen

It can be observed that the fraction of senior citizen is very less.

Most of the senior citizens do not churn.

```
# color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
# fig = px.histogram(df, x="Churn", color="OnlineSecurity",
#       barmode="group", title="<b>Churn w.r.t Online Security</b>",
#       color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

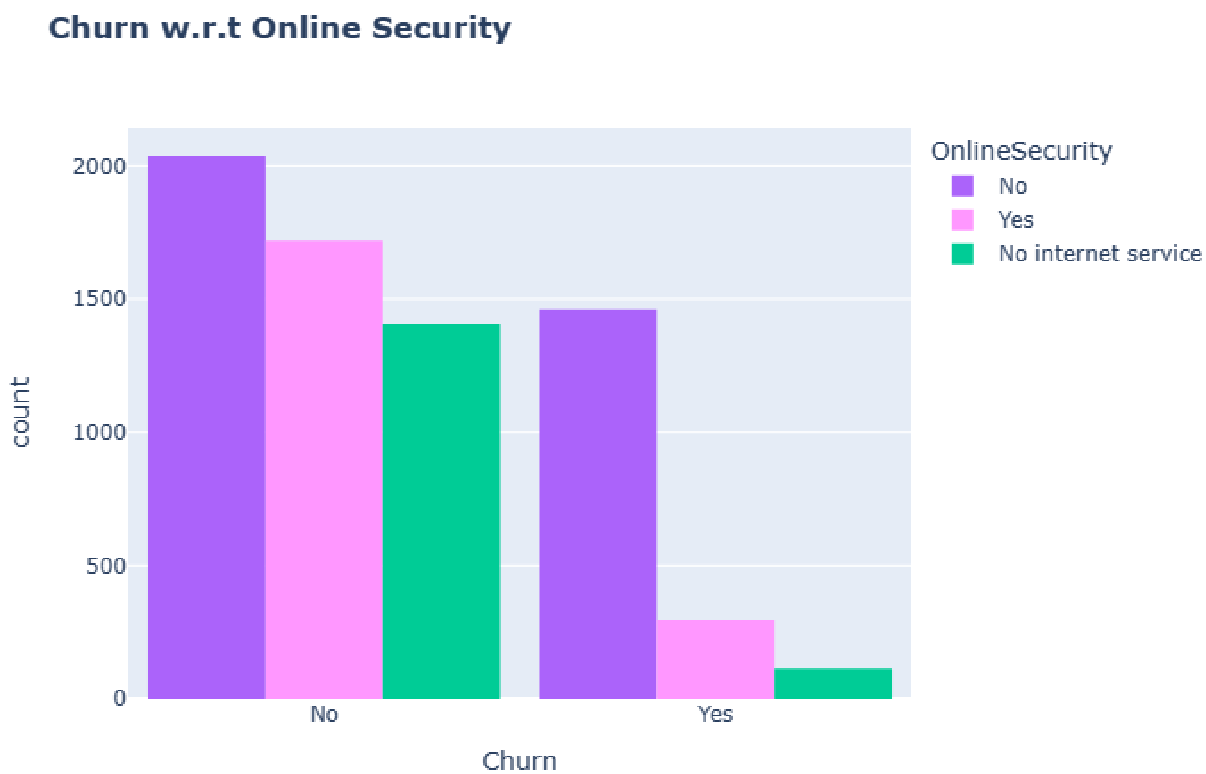


Figure 10: Churn w.r.t Online Security

Most customers churn in the absence of online security.

```
# color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
# fig = px.histogram(df, x="Churn", color="PaperlessBilling",
#       title="<b>Chrun distribution w.r.t. Paperless Billing</b>",
#       color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

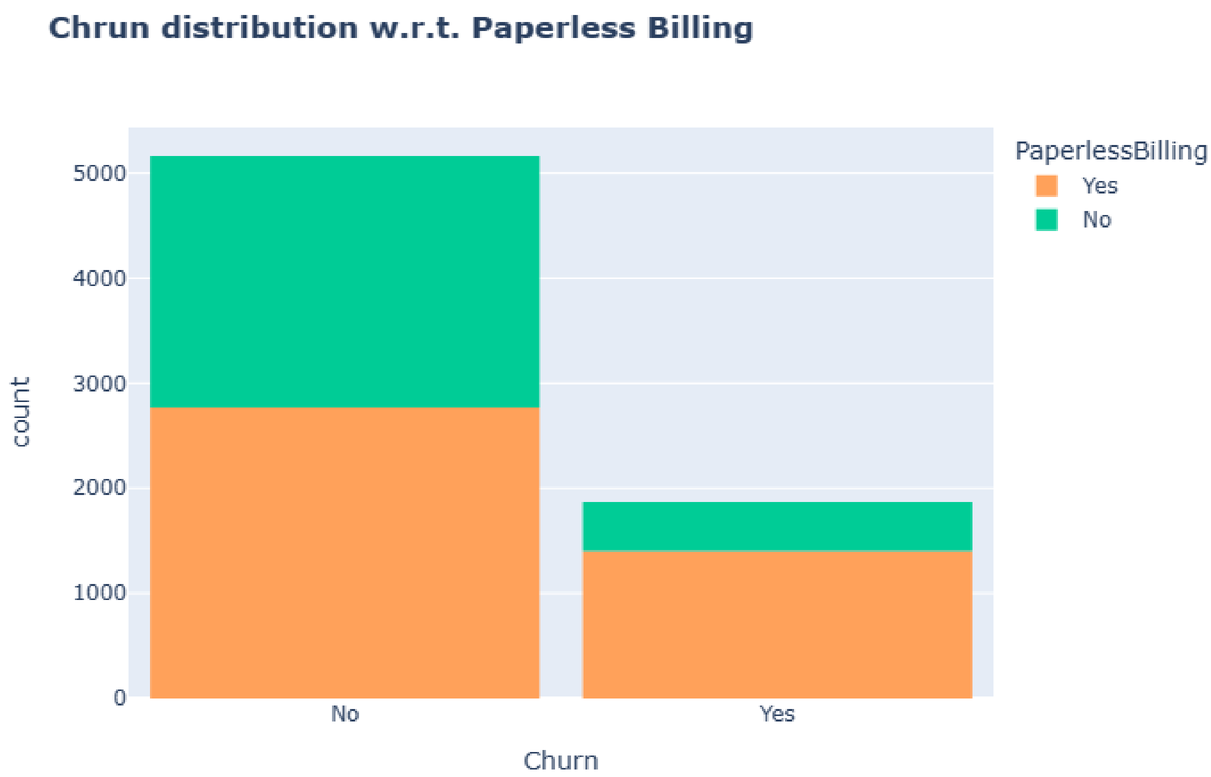


Figure 11: Chrun distribution w.r.t. Paperless Billing

Customers with Paperless Billing are most likely to churn.

```
# fig = px.histogram(df, x="Churn", color="TechSupport",
# barmode="group",  title="<b>Chrun distribution w.r.t. TechSupport</b>")
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

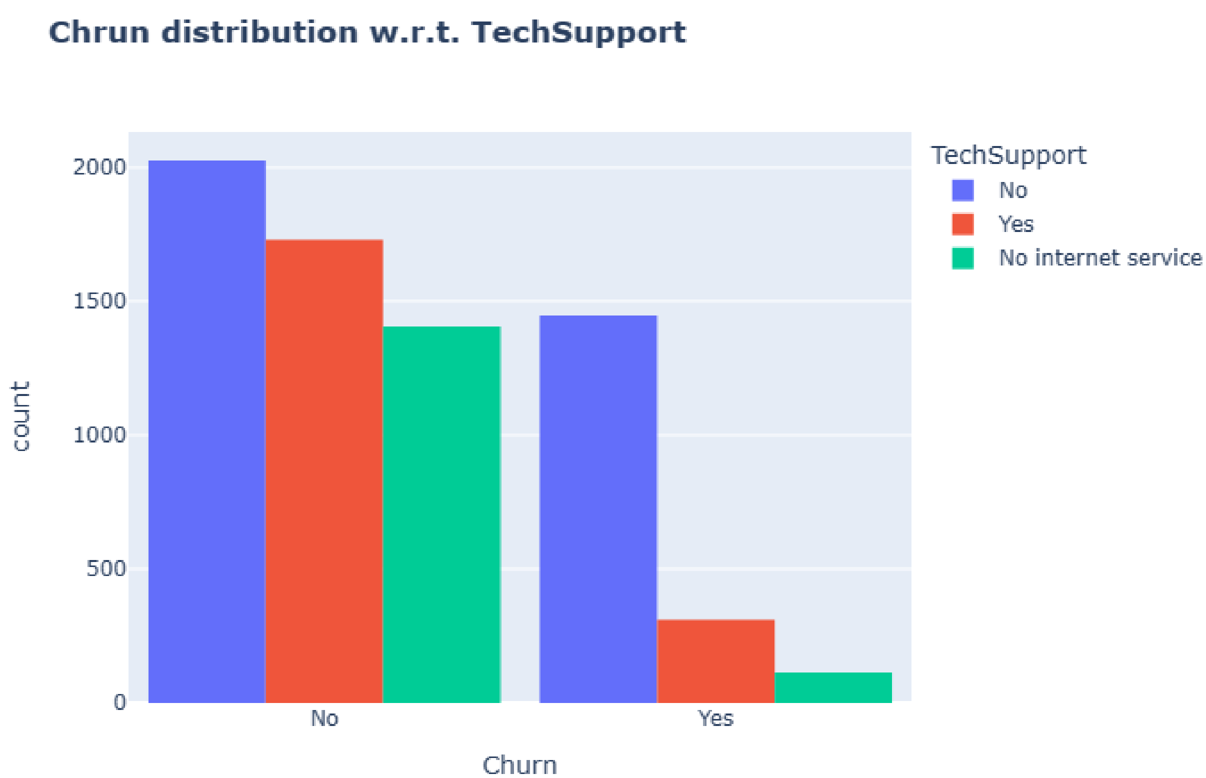


Figure 12: Chrun distribution w.r.t. TechSupport

Customers without Tech Support are most likely to migrate to another service provider.

```
# color_map = {"Yes": '#00CC96', "No": '#B6E880'}
# fig = px.histogram(df, x="Churn", color="PhoneService",
#       title="<b>Chrun distribution w.r.t. Phone Service</b>",
#       color_discrete_map=color_map)
# fig.update_layout(width=700, height=500, bargap=0.1)
# fig.show()
```

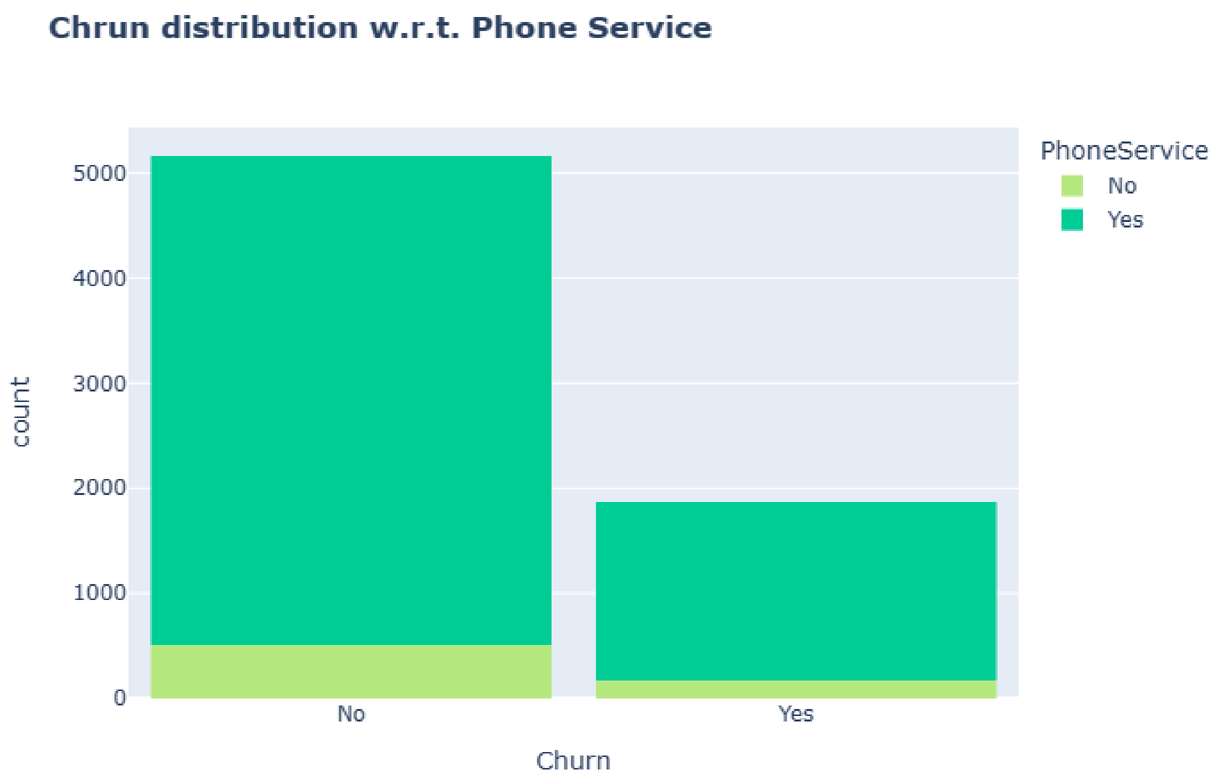
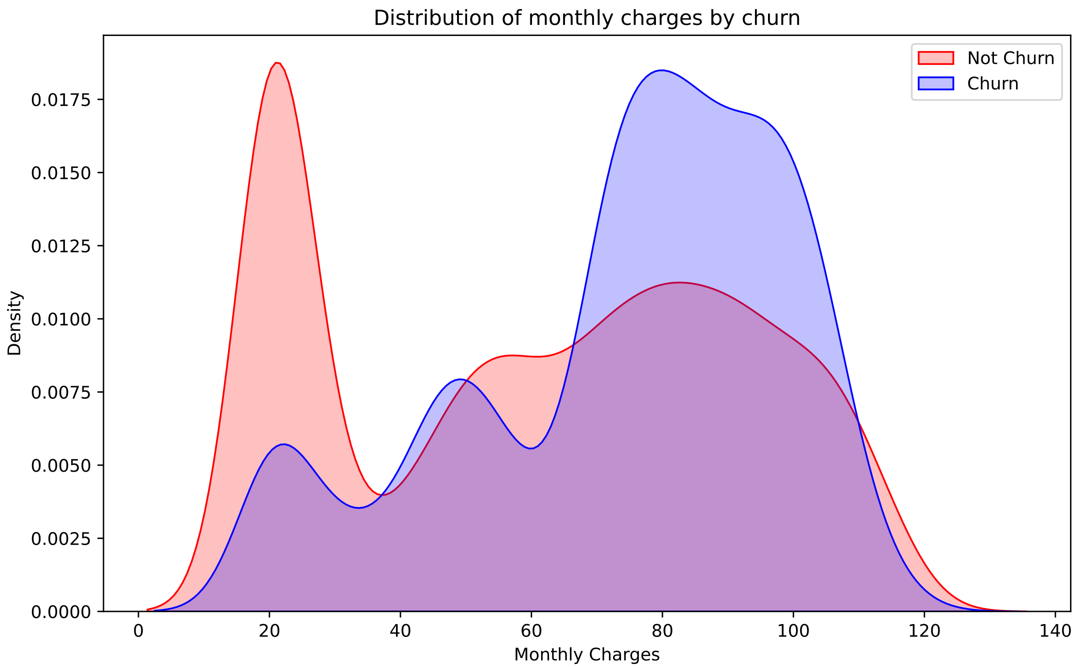


Figure 13: Chrun distribution w.r.t. Phone Service

Very small fraction of customers don't have a phone service.

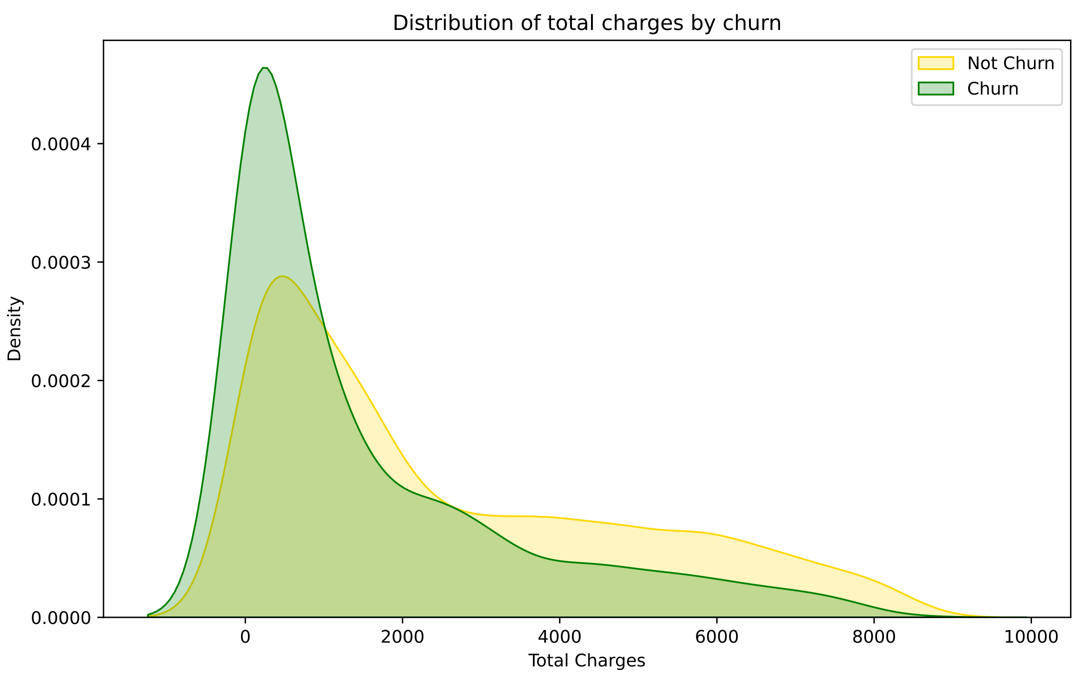
Customers who have a phone service are more likely to churn.

```python
no_churn = df[df["Churn"] == 'No']["MonthlyCharges"]
yes_churn = df[df["Churn"] == 'Yes']["MonthlyCharges"]

plt.figure(figsize=(10, 6))
sns.kdeplot(no_churn, color="red", shade=True, label="Not Churn")
sns.kdeplot(yes_churn, color="blue", shade=True, label="Churn")
plt.xlabel("Monthly Charges")
plt.ylabel("Density")
plt.title("Distribution of monthly charges by churn")
plt.legend(loc="upper right")
plt.show()
```



Customers with higher Monthly Charges are more likely to churn.

```
no_churn = df[df["Churn"] == 'No']["TotalCharges"]
yes_churn = df[df["Churn"] == 'Yes']["TotalCharges"]

plt.figure(figsize=(10, 6))
sns.kdeplot(no_churn, color="gold", shade=True, label="Not Churn")
sns.kdeplot(yes_churn, color="green", shade=True, label="Churn")
plt.xlabel("Total Charges")
plt.ylabel("Density")
plt.title("Distribution of total charges by churn")
plt.legend(loc="upper right")
plt.show()
```



Customers with lower Total Charges are more likely to churn.

```
# fig = px.box(df, x='Churn', y = 'tenure')
# # Update yaxis properties
# fig.update_yaxes(title_text='Tenure (Months)', row=1, col=1)
# # Update xaxis properties
# fig.update_xaxes(title_text='Churn', row=1, col=1)
# # Update size and title
# fig.update_layout(autosize=True, width=750, height=600,
#     title_font=dict(size=25, family='Courier'),
#     title='<b>Tenure vs Churn</b>')
# fig.show()
```

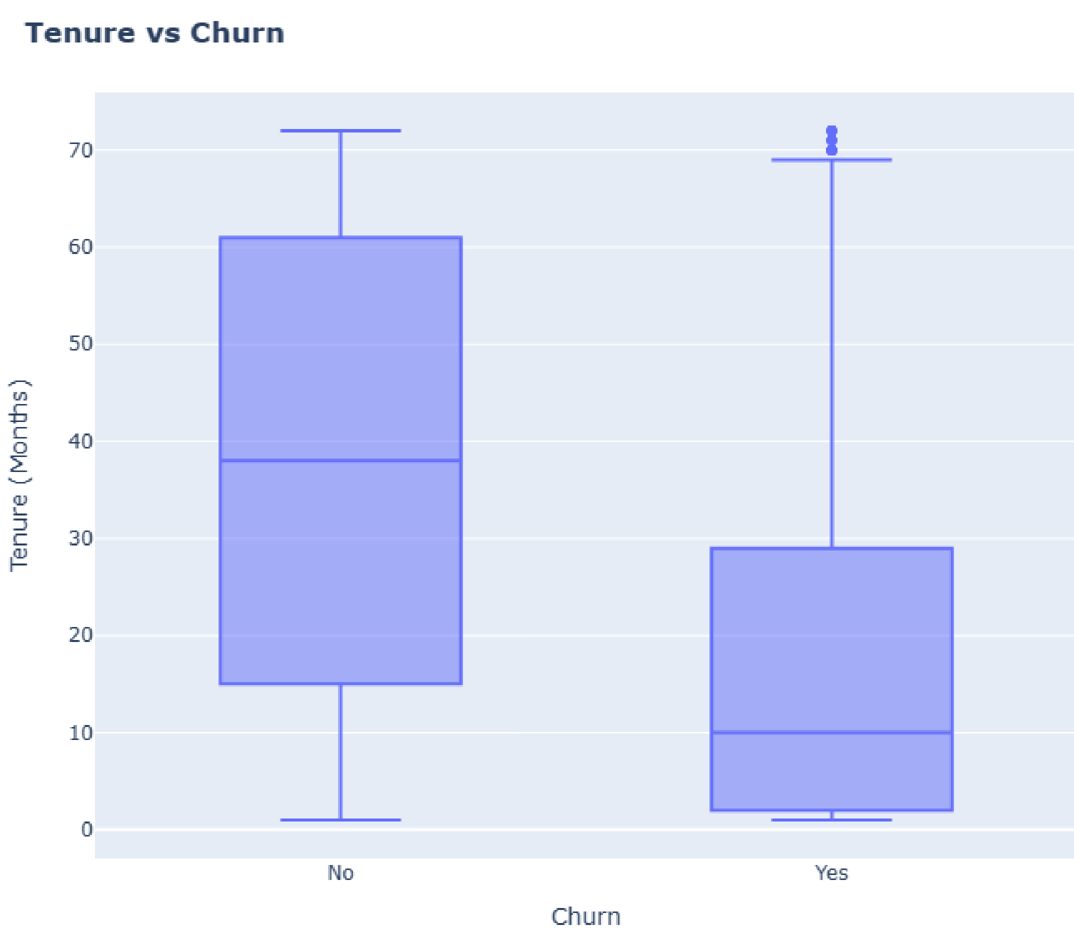


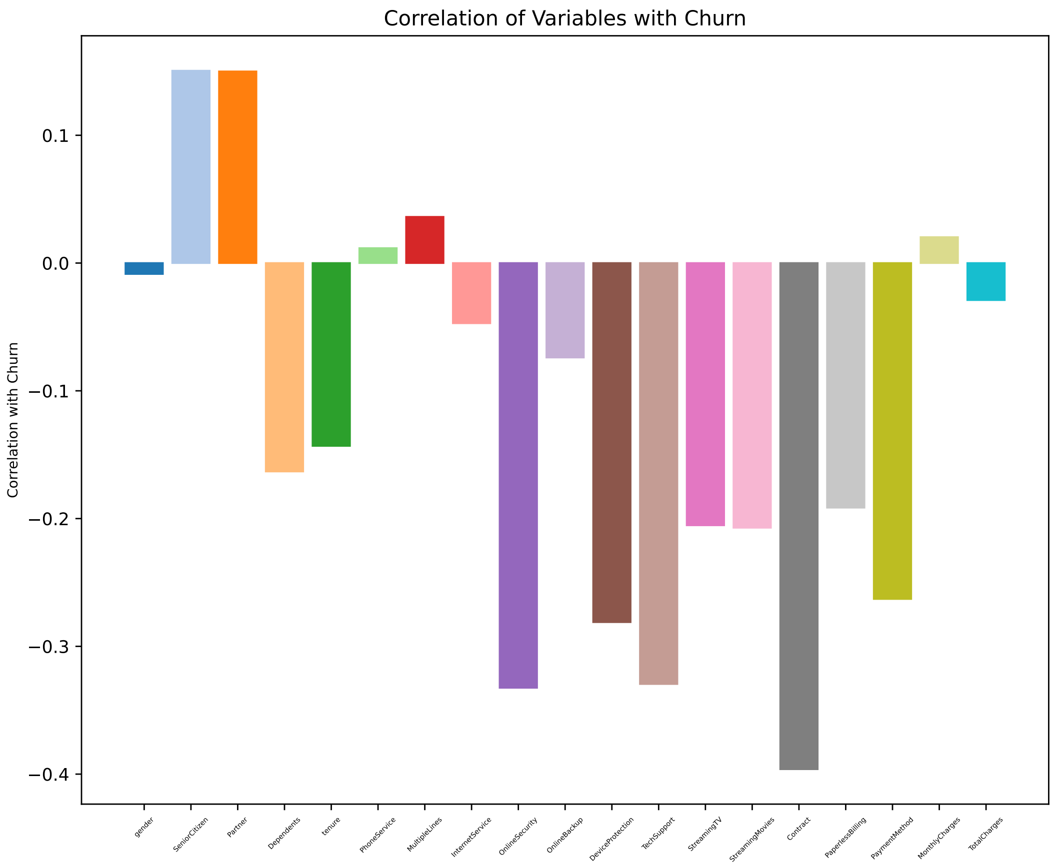
Figure 14: Tenure vs Churn

New customers are more likely to churn.

```python
# Calculate the correlation matrix
corr = df.apply(lambda x: pd.factorize(x)[0]).corr()

# Filter the correlation matrix
# to include only correlations with the 'Churn' variable
churn_corr = pd.DataFrame(corr['Churn'])
churn_corr = churn_corr.reset_index()
churn_corr = churn_corr.drop(churn_corr.index[-1])
```

```python
import matplotlib.pyplot as plt

# Generate a color map with unique colors
num_xticks = len(churn_corr)
color_map = plt.get_cmap('tab20')

# Plotting the correlation matrix
plt.figure(figsize=(10, 8))
bars = plt.bar(churn_corr['index'], churn_corr['Churn'])

# Assigning unique colors to xtick labels
for i, bar in enumerate(bars):
    bar.set_color(color_map(i % num_xticks))

# Set xtick label color to black
plt.xticks(rotation=45, color='black', fontsize=4)
```

```python
plt.ylabel('Correlation with Churn', fontsize=8)
plt.title('Correlation of Variables with Churn')
plt.show()
```


Correlation of Variables with Churn
Correlation with Churn
0.1
0.0
−0.1
−0.2
−0.3
−0.4
gender
SeniorCitizen
Partner
Dependents
tenure
PhoneService
MultipleLines
InternetService
OnlineSecurity
OnlineBackup
DeviceProtection
TechSupport
StreamingTV
StreamingMovies
Contract
PaperlessBilling
PaymentMethod
MonthlyCharges
TotalCharges

7. Data Preprocessing

Splitting the data into train and test sets

```
def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series

df = df.apply(lambda x: object_to_int(x))
df.head()
```

```
##    gender  SeniorCitizen  Partner  ...  MonthlyCharges  TotalCharges  Churn
## 0       0              0        1  ...           29.85         29.85      0
## 1       1              0        0  ...           56.95       1889.50      0
## 2       1              0        0  ...           53.85        108.15      1
## 3       1              0        0  ...           42.30       1840.75      0
## 4       0              0        0  ...           70.70        151.65      1
##
## [5 rows x 20 columns]
```

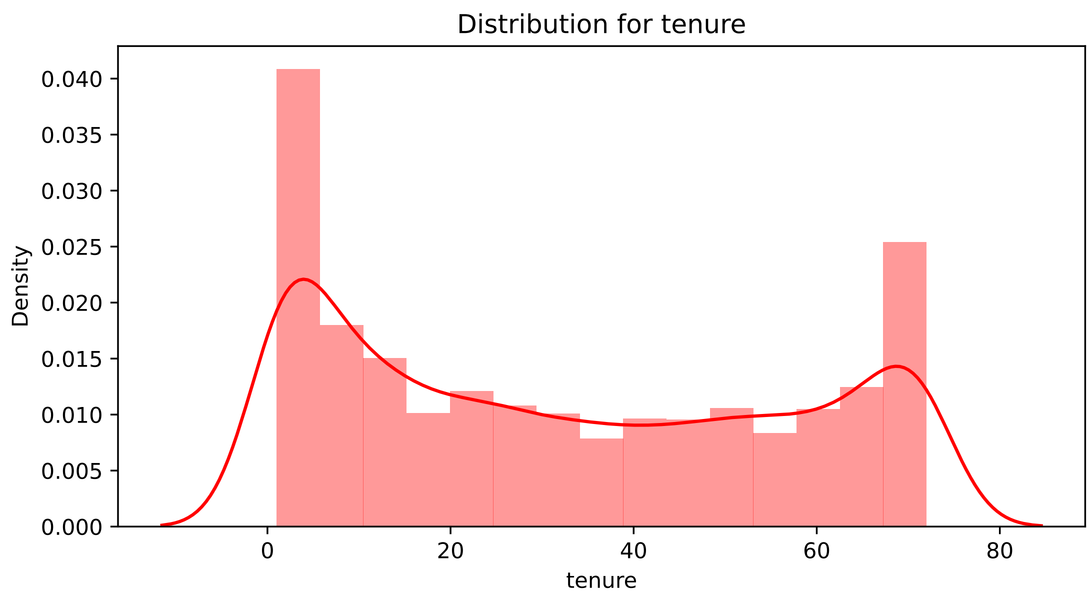
```python
plt.figure(figsize=(14,7))
df.corr()['Churn'].sort_values(ascending = False)
```

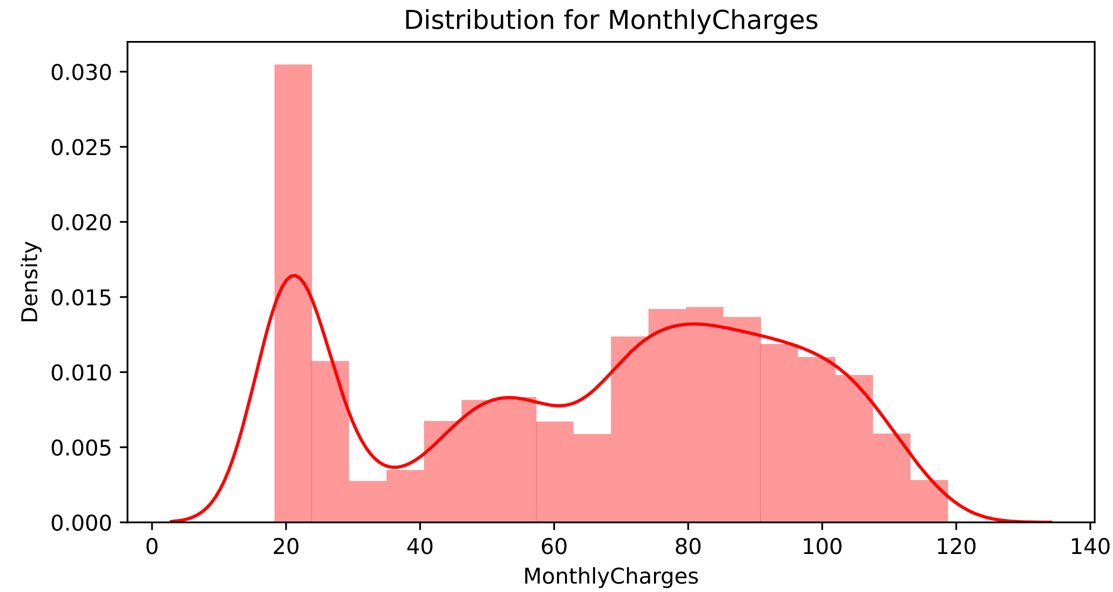
```
## Churn               1.000000
## MonthlyCharges      0.192858
## PaperlessBilling    0.191454
## SeniorCitizen       0.150541
## PaymentMethod       0.107852
## MultipleLines       0.038043
## PhoneService        0.011691
## gender             -0.008545
## StreamingTV        -0.036303
## StreamingMovies    -0.038802
## InternetService    -0.047097
## Partner            -0.149982
## Dependents         -0.163128
## DeviceProtection   -0.177883
## OnlineBackup       -0.195290
## TotalCharges       -0.199484
## TechSupport        -0.282232
## OnlineSecurity     -0.289050
## tenure             -0.354049
## Contract           -0.396150
## Name: Churn, dtype: float64
```

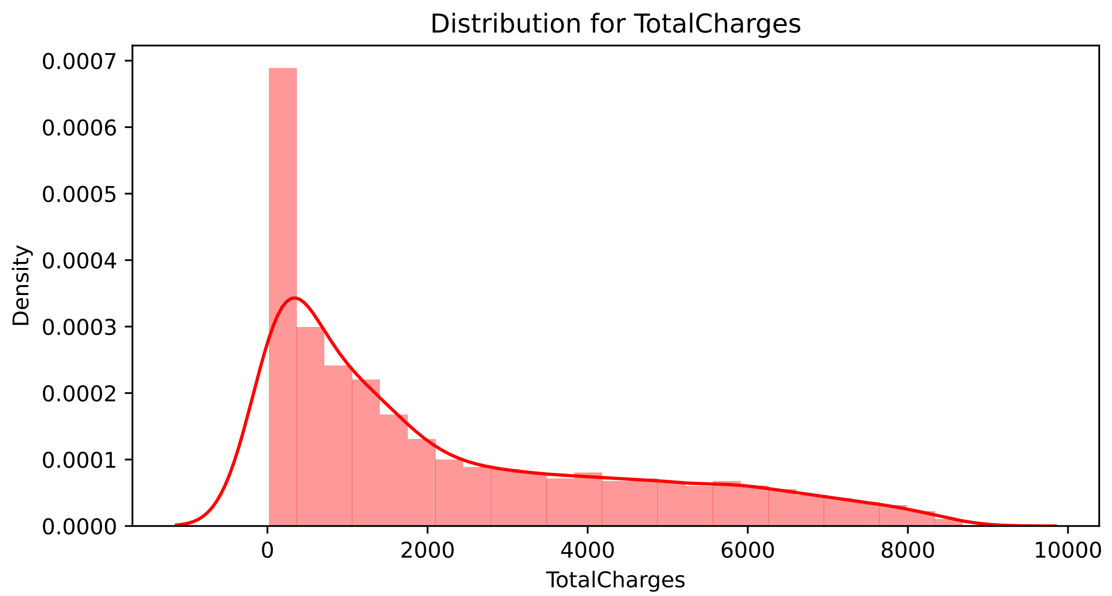
```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values

X_train, X_test, y_train, y_test = train_test_split(X,y,
test_size = 0.30, random_state = 40, stratify=y)
```

```
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,4))
    plt.title("Distribution for {}".format(feature))
    sns.distplot(frame[feature], color= color)
    plt.show()
```

```
num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
for feat in num_cols: distplot(feat, df)
```



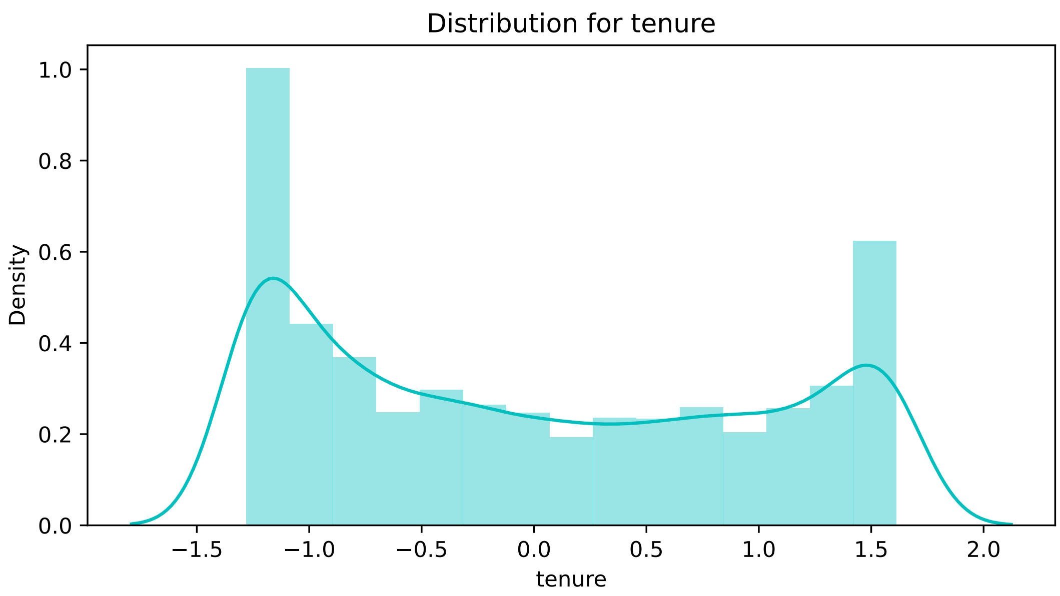

Distribution for MonthlyCharges
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
MonthlyCharges

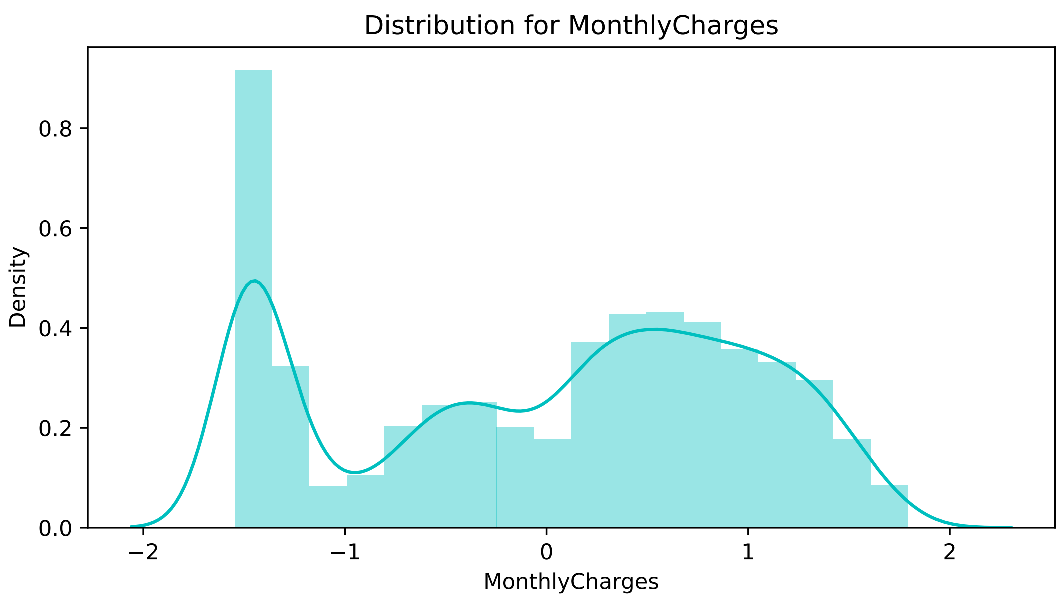

Distribution for TotalCharges
Density
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
0
2000
4000
6000
8000
10000
TotalCharges

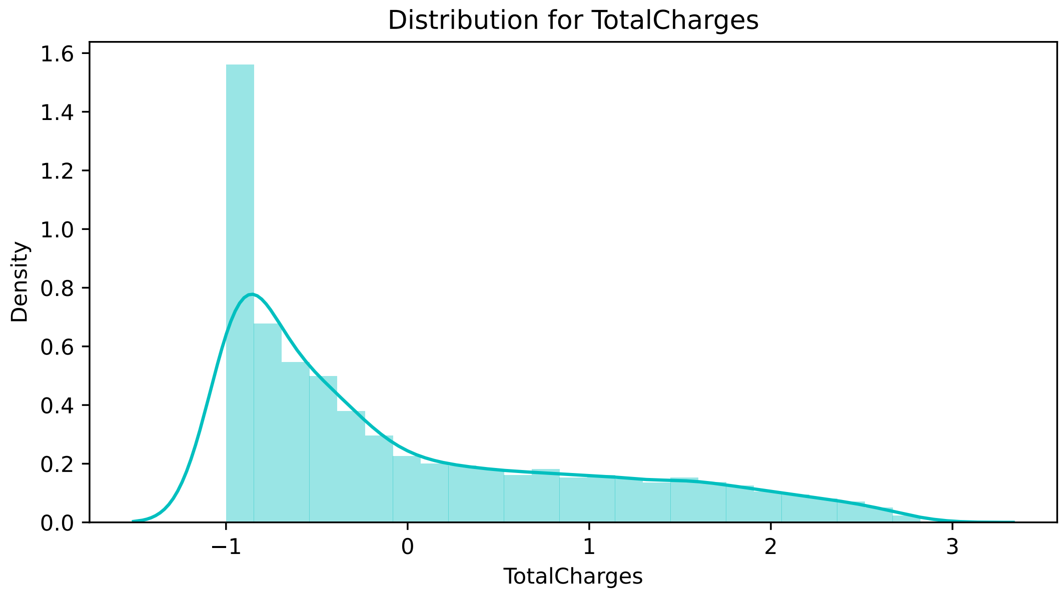
Since the numerical features are distributed over different value ranges, we will use standard scalar to scale them down to the same range.

Standardizing numeric attributes

```
df_std = pd.DataFrame(
  StandardScaler().fit_transform(df[num_cols].astype('float64')),
  columns=num_cols)

for feat in numerical_cols: distplot(feat, df_std, color='c')
```








```
scaler = StandardScaler()

X_train[num_cols] = scaler.fit_transform(X_train[num_cols])
X_test[num_cols] = scaler.transform(X_test[num_cols])
```

8. Machine Learning Model Evaluations and Predictions

8.1 Logistic Regression

```
lr_model = LogisticRegression(random_state=3)
lr_model.fit(X_train,y_train)
```

```
## LogisticRegression(random_state=3)
```

```
prediction_lr = lr_model.predict(X_test)
accuracy_lr = accuracy_score(y_test, prediction_lr)
print("Logistic Regression accuracy is :", accuracy_lr)
```

```
## Logistic Regression accuracy is : 0.8090047393364929
```

8.2 AdaBoost

```
a_model = AdaBoostClassifier(random_state=3)
a_model.fit(X_train,y_train)
```

```
## AdaBoostClassifier(random_state=3)
```

```
prediction_a = a_model.predict(X_test)
accuracy_a = accuracy_score(y_test, prediction_a)
print("AdaBoost Classifier accuracy is :", accuracy_a)
```

```
## AdaBoost Classifier accuracy is : 0.8075829383886256
```

8.3 Gradient Boosting

```
gb_model = GradientBoostingClassifier(random_state=3)
gb_model.fit(X_train, y_train)
```

```
## GradientBoostingClassifier(random_state=3)
```

```
prediction_gb = gb_model.predict(X_test)
accuracy_gb = accuracy_score(y_test, prediction_gb)
print("Gradient Boosting Classifier accuracy is", accuracy_gb)
```

```
## Gradient Boosting Classifier accuracy is 0.8080568720379147
```

8.4 Voting Classifier

```
from sklearn.ensemble import VotingClassifier
lr = LogisticRegression(random_state=3)
abc = AdaBoostClassifier(random_state=3)
gbc = GradientBoostingClassifier(random_state=3)

eclf = VotingClassifier(estimators=[('lr', lr), ('abc', abc), ('gbc', gbc)],
                                    voting='soft', weights=[1,1,1])

eclf.fit(X_train, y_train)
```

```
## VotingClassifier(estimators=[('lr', LogisticRegression(random_state=3)),
##                              ('abc', AdaBoostClassifier(random_state=3)),
##                              ('gbc',
##                               GradientBoostingClassifier(random_state=3))],
##                  voting='soft', weights=[1, 1, 1])
```

```
predictions = eclf.predict(X_test)
accuracy_vot = accuracy_score(y_test, predictions)
print("Voting Classifier Accuracy Score :", accuracy_vot)
```

```
## Voting Classifier Accuracy Score : 0.8170616113744076
```

8.5 Feature importance based on Voting Classifier Model

```
import eli5
from eli5.sklearn import PermutationImportance

perm = PermutationImportance(eclf, random_state=1).fit(X_test, y_test)
weights_df = eli5.formatters.as_dataframe.explain_weights_df(perm,
             feature_names=X_test.columns.tolist())

print(weights_df)
```

```
##              feature    weight       std
## 0             tenure  0.075829  0.010283
## 1     MonthlyCharges  0.044645  0.006638
## 2           Contract  0.037820  0.003357
## 3       TotalCharges  0.012417  0.002671
## 4        TechSupport  0.011469  0.002654
## 5       PhoneService  0.007014  0.003357
## 6     OnlineSecurity  0.006730  0.002550
## 7   PaperlessBilling  0.005687  0.001038
## 8      SeniorCitizen  0.003223  0.001320
## 9    InternetService  0.002370  0.002952
## 10     PaymentMethod  0.002180  0.001546
## 11     MultipleLines  0.001991  0.001569
## 12      OnlineBackup  0.001706  0.001711
## 13  DeviceProtection  0.001422  0.001236
## 14   StreamingMovies  0.000853  0.000355
## 15       StreamingTV  0.000474  0.000599
## 16            gender  0.000095  0.000355
## 17        Dependents -0.000569  0.001511
## 18           Partner -0.000664  0.000643
```

According to this table, it’s better to remove some features for the predictive modeling.

```python
# Create new subsets of data with only the important features
columns_to_drop = ['StreamingMovies', 'StreamingTV', 'gender']

X_train_selected = X_train.drop(columns=columns_to_drop)
X_test_selected = X_test.drop(columns=columns_to_drop)

# Train the individual classifiers on the new data
lr_selected = LogisticRegression(random_state=3)
abc_selected = AdaBoostClassifier(random_state=3)
gbc_selected = GradientBoostingClassifier(random_state=3)

# Update the Voting Classifier with the new classifiers
eclf_selected = VotingClassifier(estimators=[('lr', lr_selected),
                                ('abc', abc_selected), ('gbc', gbc_selected)],
                                 voting='soft', weights=[1, 1, 1])

eclf_selected.fit(X_train_selected, y_train)
```

```
## VotingClassifier(estimators=[('lr', LogisticRegression(random_state=3)),
##                              ('abc', AdaBoostClassifier(random_state=3)),
##                              ('gbc',
##                               GradientBoostingClassifier(random_state=3))],
##                  voting='soft', weights=[1, 1, 1])
```

```python
predictions_selected = eclf_selected.predict(X_test_selected)
accuracy_vot_selected = accuracy_score(y_test, predictions_selected)

print("Final Voting Classifier Accuracy Score:", accuracy_vot_selected)
```

```
## Final Voting Classifier Accuracy Score: 0.8175355450236966
```

9. Advice

After conducting feature selection by removing the attributes “Streaming Movies”, “Streaming TV” and “gender”, our refined machine learning model demonstrates an enhanced accuracy score of 81.75%. This noteworthy improvement surpasses the previous accuracy score of the Voting Classifier, which stood at 81.70%.

Based on the observed enhancement in our predictive model’s performance, it is advisable not to implement differential pricing based on gender. Additionally, it is recommended to exclude the features related to Streaming Movies and Streaming TV Services. By adhering to these recommendations, the company can optimize its customer retention strategies, thereby strengthening customer satisfaction and loyalty.