Enhancing Customer Retention in Telco: A Churn Prediction with Ensemble Learning

About Dataset

Context

"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 described on the column Metadata.

The data set includes information about:*

- Customers who left within the last month – the column is called Churn
- 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
- Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges
- Demographic info about customers – gender, age range, and if they have partners and dependents

About Feature

- **customerID**: Customer ID
- **gender**: Whether the customer is a male or a female
- **Senior Citizen**: 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
- **Phone Service**: Whether the customer has a phone service or not (Yes, No)
- **Multiple Lines**: Whether the customer has multiple lines or not (Yes, No, No phone service)
- **Internet Service**: Customer's internet service provider (DSL, Fiber optic, No)
- **Online Security**: Whether the customer has online security or not (Yes, No, No internet service)
- **Online Backup**: Whether the customer has online backup or not (Yes, No, No internet service)
- **Device Protection**: Whether the customer has device protection or not (Yes, No, No internet service)
- **Tech Support**: 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)
- **Streaming**: 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)
- **Paperless Billing**: Whether the customer has paperless billing or not (Yes, No)
- **Payment Method**: The customer's payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- **Monthly Charges**: The amount charged to the customer monthly
- **Total Charges**: The total amount charged to the customer
- **Churn Label**: Whether the customer churned or not (Yes or No)

Implementations

```
1. Imported Required Libraries
2. Data Extraction
3. Data Preparation
 3.1 Detecting Missing Values
```

1. Imported Required Libraries

In [1]:

```
# Data manipulation and analysis library
import pandas as pd
import numpy as np

# Plotly [data visualizations and plots]
# a high-level interface for creating interactive visualizations with Plotly
import plotly.express as px
# a lower-level interface for creating interactive plots and charts using Plotly
import plotly.graph_objects as go
# create subplots within a single figure using Plotly
from plotly.subplots import make_subplots
# Plotly Figure Factory is a sub-module for creating various specialized plot types, such as heatmaps, tab
import plotly.figure_factory as ff

# Multicollinearity
# the statsmodels library to calculate the Variance Inflation Factor (VIF)
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Split the dataset
from sklearn.model_selection import train_test_split

# Module from scikit-learn contains Machine Learning Models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression

# Evaluating the performance of machine learning models
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix, precision_score, f1_score, accuracy_score, cla
from sklearn.metrics import auc

# Standardization
from sklearn.preprocessing import StandardScaler

# K-fold cross validation
# a technique for splitting data into multiple folds for model training and evaluation
from sklearn.model_selection import KFold

# Score evaluation
from sklearn.model_selection import cross_val_score

# Voting Classifier
# an ensemble method in scikit-learn that combines the predictions of multiple machine learning models to
from sklearn.ensemble import VotingClassifier
```

2. Data Extraction

In [2]:

```
df = pd.read_csv('Telco-Customer-Churn.csv')
df.head()
```

Out[2]:

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

5 rows × 21 columns

In [3]:

```
df.shape
```

Out[3]:

```
(7043, 21)
```

In [4]:

```
list(df.columns.values)
```

Out[4]:

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

The data set includes information about:

1. Customers who left within the last month – the column is called Churn
2. 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
3. Customer account information - how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges

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

In [5]:

```
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
```

Variables are of different types, which are categorized below Categorical:

1. Binary (7): SeniorCitizen, gender, Partner, Dependents, PhoneService, PaperlessBilling, and Churn
2. Multimonial (11): CustomerID, MultipleLines,InternetService, OnlineSecurity, OnlineBackup, DeviceProtection, TechSupport, StreamingTV, StreamingMovies, Contract, PaymentMethod
3. Continuous(3): TotalCharges, MonthlyCharges and Tenure

-> The target the we will use to guide the exploration is Churn. -> The Total Charges is observed to be stored in string datatype, meanwhile it should be stored as numerical data type.

3. Data Preparation

3.1 Detecting Missing Values

An in depth analysis can reveal some indirect missingness in the dataset (which can be in form of blankspaces). We will drop **customer ID** first, then transform the **Total Charges** into numerical data.

In [6]:

```
# Data manipulation

df = df.drop(['customerID'], axis = 1)
df.head()
```

Out[6]:

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

In [7]:

```
# Convert 'Total Charges' into numerical values

# erros = 'coerce' is used if there are any non-numeric values in the 'TotalCharges' column, they will be

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

In [8]:

```
df['TotalCharges'].dtype
```

Out[8]:

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

In [9]:

```
# Check for missing values

df.isnull().sum()
```

Out[9]:

```
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
```

Results return 11 missing values in **Total Charges**. We will calculate how much proportion they take in the dataset to decide the handling missing values strategy.

In [10]:

```
# Calculate the missing values
def missing_values(n):
    df_m=pd.DataFrame()
    df_m["missing_values, %"]=df.isnull().sum()*100/len(df.isnull())
    df_m["missing_values, sum"]=df.isnull().sum()
    return df_m.sort_values(by="missing_values, %", ascending=False)
```

In [11]:

```
missing_values(df)
```

Out[11]:

	missing_values, %	missing_values, sum
TotalCharges	0.156183	11
gender	0.000000	0
SeniorCitizen	0.000000	0
MonthlyCharges	0.000000	0
PaymentMethod	0.000000	0
PaperlessBilling	0.000000	0
Contract	0.000000	0
StreamingMovies	0.000000	0
StreamingTV	0.000000	0
TechSupport	0.000000	0
DeviceProtection	0.000000	0
OnlineBackup	0.000000	0
OnlineSecurity	0.000000	0
InternetService	0.000000	0
MultipleLines	0.000000	0
PhoneService	0.000000	0
tenure	0.000000	0
Dependents	0.000000	0
Partner	0.000000	0
Churn	0.000000	0

Since the missing values of **Total Charges** comprise of the dataset 0.16%, we have two options:

1. Decide to delete them since they only comprise a small percentage of the dataset.
2. Decide to not delete them and impute with mean value.

However, we will assess these values in detail to identify whether there is a reason behind then finalise the decision.

In [12]:

```
# Filter df to find rows that have missing values in 'Total Charges'

df[np.isnan(df['TotalCharges'])]
```

Out[12]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	On
488	Female	0	Yes	Yes	0	No	No phone service	DSL	
753	Male	0	No	Yes	0	Yes	No	No	
936	Female	0	Yes	Yes	0	Yes	No	DSL	
1082	Male	0	Yes	Yes	0	Yes	Yes	No	
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	
3331	Male	0	Yes	Yes	0	Yes	No	No	
3826	Male	0	Yes	Yes	0	Yes	Yes	No	
4380	Female	0	Yes	Yes	0	Yes	No	No	
5218	Male	0	Yes	Yes	0	Yes	No	No	
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	
6754	Male	0	No	Yes	0	Yes	Yes	DSL	

From the entries above, we detect inconsistency in **tenure, monthly charges and total charges values**.

- Tenure refers to the number of months that a customer has been with the company, therefore, the "0" entry seems to not make sense and appears skeptical.
- Considering Monthly Charges column in which charged amounts are still recored. We assume that the "0" in tenure the may present a short amount of time (less than 1 month), and they are new customers who haven't completed a billing cycle. We will double check if there are any other '0' values in the **tenure** column.

In [13]:

```
# Filter df to find the index positions of rows where the 'tenure' column has a value of 0

df[df['tenure'] == 0].index
```

Out[13]:

```
Int64Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

There are no other missing values in the tenure column. We now proceed to handling missing values.

Having said that the missing values in Total Charges could possibly represent new customers who have not finish the billing cycle, we decide to not delete them and replace with mean value.

In [14]:

```
# Impute missing values with corresponding monthly charges

df['TotalCharges'].fillna(df['TotalCharges'].mean(), inplace=True)
df[df['tenure'] == 0]
```

Out[14]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	On
488	Female	0	Yes	Yes	0	No	No phone service	DSL	
753	Male	0	No	Yes	0	Yes	No	No	
936	Female	0	Yes	Yes	0	Yes	No	DSL	
1082	Male	0	Yes	Yes	0	Yes	Yes	No	
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	
3331	Male	0	Yes	Yes	0	Yes	No	No	
3826	Male	0	Yes	Yes	0	Yes	Yes	No	
4380	Female	0	Yes	Yes	0	Yes	No	No	
5218	Male	0	Yes	Yes	0	Yes	No	No	
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	
6754	Male	0	No	Yes	0	Yes	Yes	DSL	

In [15]:

```
# Check for missing values again
df.isnull().sum()
```

Out[15]:

```
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
```

3.2 Detecting Outliers

In [16]:

```
df.describe()
```

Out[16]:

	SeniorCitizen	tenure	MonthlyCharges	TotalCharges
count	7043.000000	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692	2283.300441
std	0.368612	24.559481	30.090047	2265.000258
min	0.000000	0.000000	18.250000	18.800000
25%	0.000000	9.000000	35.500000	402.225000
50%	0.000000	29.000000	70.350000	1400.550000
75%	0.000000	55.000000	89.850000	3786.600000
max	1.000000	72.000000	118.750000	8684.800000

Numerical columns

In [17]:

```
# Define numerical columns

numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
```

In [18]:

```
# Describing descriptive stats of the data
df[numerical_cols].describe().T
```

Out[18]:

	count	mean	std	min	25%	50%	75%	max
tenure	7043.0	32.371149	24.559481	0.00	9.000	29.00	55.00	72.00
MonthlyCharges	7043.0	64.761692	30.090047	18.25	35.500	70.35	89.85	118.75
TotalCharges	7043.0	2283.300441	2265.000258	18.80	402.225	1400.55	3786.60	8684.80

In [19]:

```
# Detecting outliers
def detect_outliers_iqr(data):
    # Calculate the first quartile (Q1) and third quartile (Q3)
    Q1 = np.percentile(data, 25)
    Q3 = np.percentile(data, 75)

    # Calculate the IQR
    IQR = Q3 - Q1

    # Define the lower and upper bounds to identify outliers
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Find the outliers
    outliers = [x for x in data if x < lower_bound or x > upper_bound]

    return outliers
```

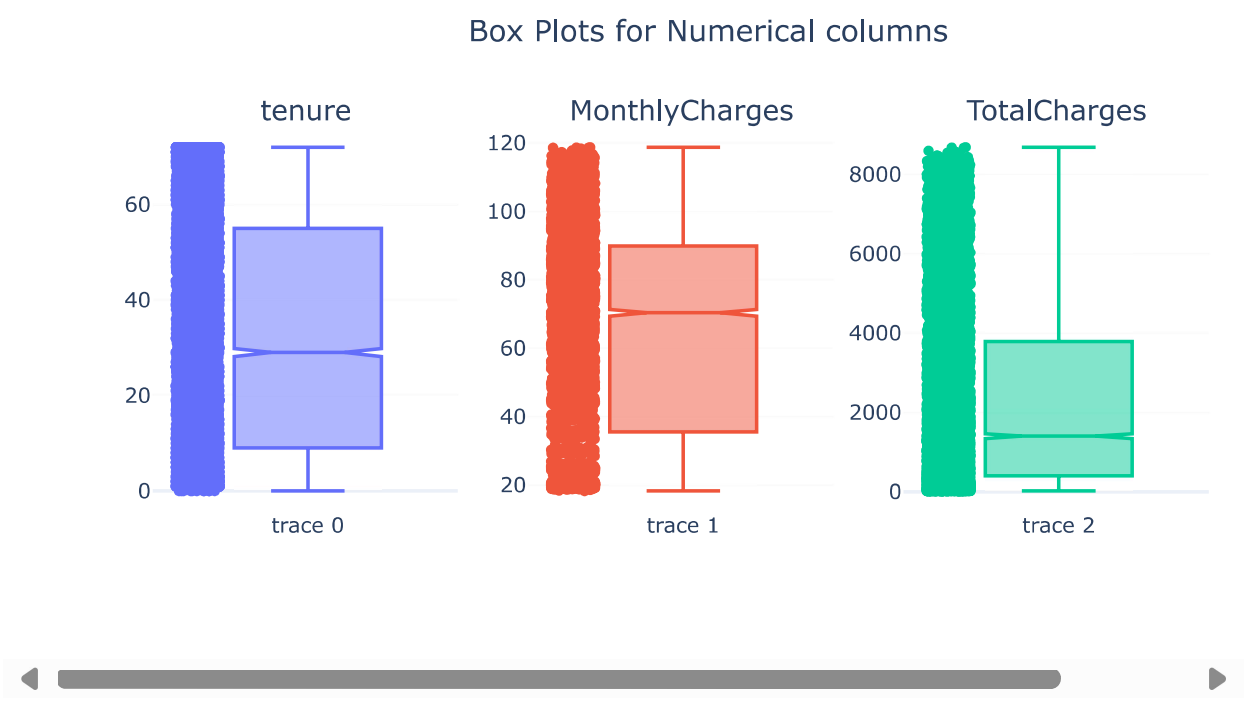
In [20]:

```
# Apply outlier detection to all numerical columns
outliers_by_column = {}
for column in numerical_cols:
    data_column = df[column]
    outliers = detect_outliers_iqr(data_column)
    outliers_by_column[column] = outliers

# Print the outliers for each column
for column, outliers in outliers_by_column.items():
    print(f"Outliers in {column}: {outliers}")
```

```
Outliers in tenure: []
Outliers in MonthlyCharges: []
Outliers in TotalCharges: []
```

In [21]:

```
# Create separate box plots for 'tenure', 'TotalCharges', and 'MonthlyCharges'

x = numerical_cols

# Calculate the number of rows and columns for subplots
total_plots = len(x)
rows, cols = divmod(total_plots, 3)
if cols > 0:
    rows += 1

# Create subplots dynamically
fig = make_subplots(rows=rows, cols=3, subplot_titles=x)

# Add box plots to subplots
row, col = 1, 1
for i, col_name in enumerate(x):
    box=go.Box(y=df[col_name],boxpoints='all', notched=True )
    fig.add_trace(box, row=row, col=col)
    col += 1
    if col > 3:
        col = 1
        row += 1

# Update layout
fig.update_layout(template = 'plotly_white', height=400, width=800, title_text="Box Plots for Numerical c
                  title_x=0.5)  # Center the title

# Show the figure
fig.show()
```



We confirm that there is no outlier in the numerical features.

3.3. Label Encode Binary Data

Categorical Variables

- Independent variables for machine learning models typically must be only numerical values.

- Therefore, label Encoding Binary Data is used for all categorical variables with only two unique values.

In [22]:

```
# Check for unique values to be make informed encoding decision
unique_counts = df.nunique()
print("Unique Value Counts:")
print(unique_counts)
```

```
Unique Value Counts:
gender                 2
SeniorCitizen          2
Partner                2
Dependents             2
tenure                73
PhoneService           2
MultipleLines          3
InternetService        3
OnlineSecurity         3
OnlineBackup           3
DeviceProtection       3
TechSupport            3
StreamingTV            3
StreamingMovies        3
Contract               3
PaperlessBilling       2
PaymentMethod          4
MonthlyCharges      1585
TotalCharges        6531
Churn                  2
dtype: int64
```

In [23]:

```
# Label-Encoding for Categorical Data

cols = ['gender', 'SeniorCitizen', 'Partner', 'Dependents', 'PhoneService', 'PaperlessBilling', 'Churn']
```

In [24]:

```
# Change data type for categorical data variables

df[cols] = df[cols].astype('category')
```

In [25]:

```
# Label encoding for categorical data variables
for column in cols:
    df[column] = df[column].cat.codes # perform label encoding on the values within that column
```

This will replace the original categorical values with integer codes.

In [26]:

```
# Check data types of all columns
print(df.dtypes)
```

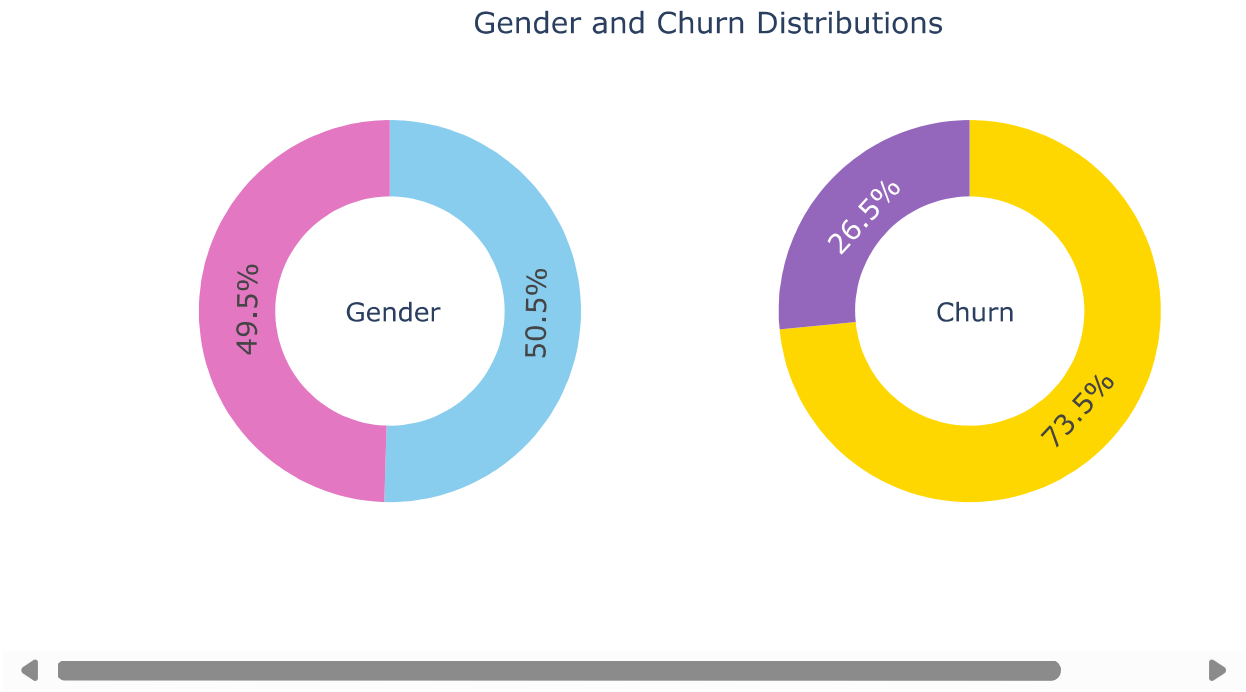
```
gender                int8
SeniorCitizen         int8
Partner               int8
Dependents            int8
tenure               int64
PhoneService          int8
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling      int8
PaymentMethod       object
MonthlyCharges     float64
TotalCharges       float64
Churn                 int8
dtype: object
```

4. Exploring Data Analysis

4.1. Visualisng data

In [27]:

```
g_labels = ['Male', 'Female']
c_labels = ['No', 'Yes']

colors1 = ["rgb(136,204,238)", '#e377c2']  # Define four colors for the categories
colors2 =["gold", '#9467bd']
# 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",marker=dict(color
              1, 1)
fig.add_trace(go.Pie(labels=c_labels, values=df['Churn'].value_counts(), name="Churn",marker=dict(colors=
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.6, hoverinfo="label+percent+name", textfont_size=16)#,marker=dict(colors=colors))

fig.update_layout(
    title_text="Gender and Churn Distributions",
    title_x=0.5,  # Center the title
    # Adjust width and height to resize the chart
    width=800, height=400,
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='Gender', x=0.18, y=0.5, font_size=15, showarrow=False),
                 dict(text='Churn', x=0.82, y=0.5, font_size=15, showarrow=False)])
fig.show()
```



Inference:

- 26.5 % of customers switched to another firm.
- Customers are 49.5 % female and 50.5 % male.

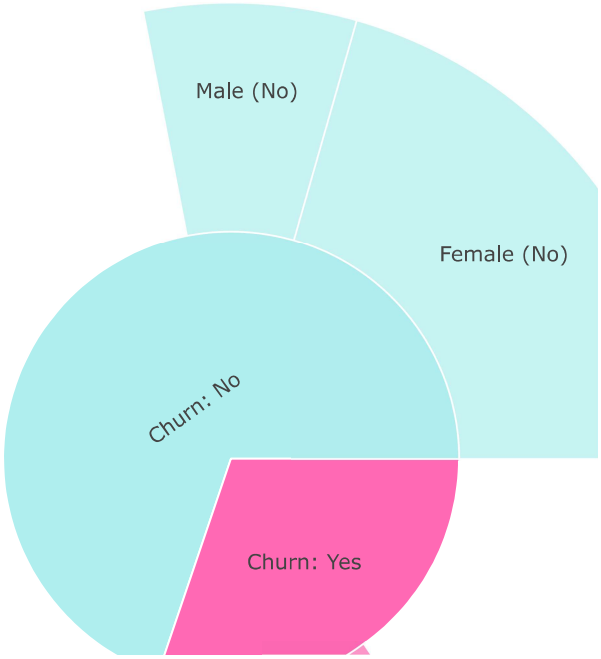
Sunburst chart

- A pie chart with two levels where the outer pie chart represents "Churn: Yes" and "Churn: No," and the inner pie chart represents "Male" and "Female" within each churn category. Unfortunately, Plotly does not support nesting pie charts.
- However, you can achieve a similar visual representation using a sunburst chart.
- Sunburst charts allow for hierarchical data representation, which can visually resemble a pie chart within another pie chart.

For multilevel pie charts representing hierarchical data, you can use the Sunburst chart.

In [28]:

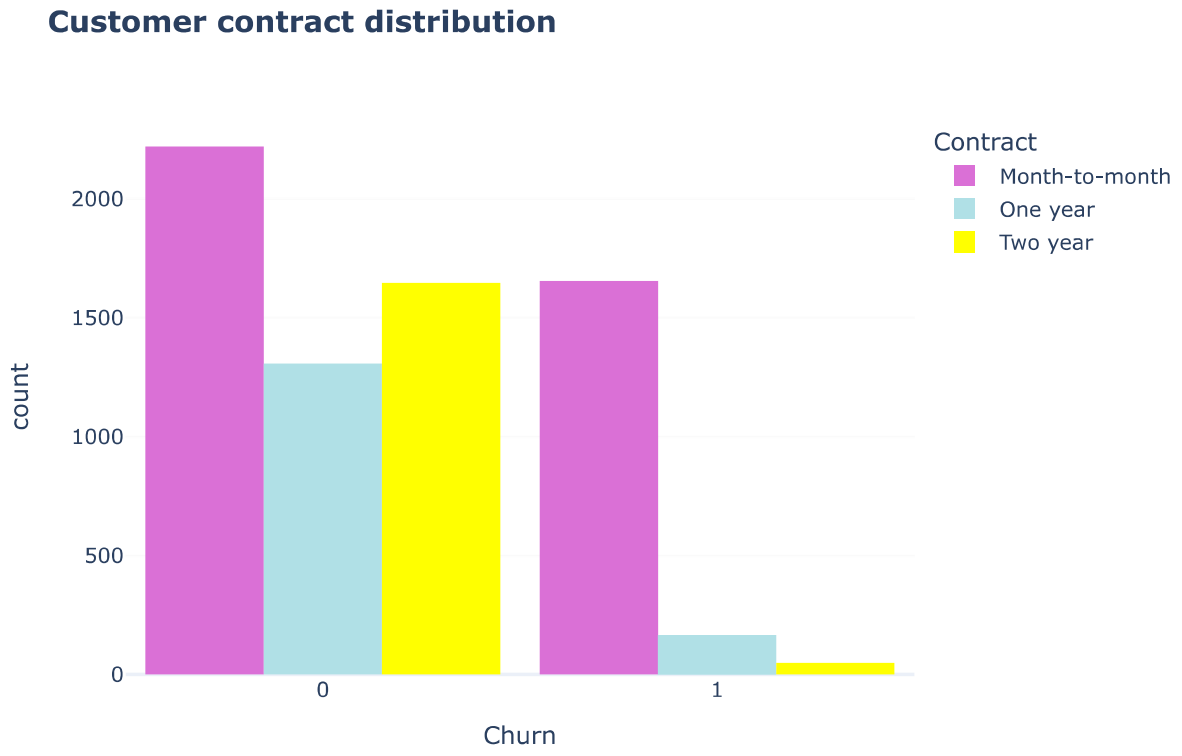
```
# Data
labels = ["Churn: Yes", "Churn: No", "Male (Yes)", "Female (Yes)", "Male (No)", "Female (No)"]
parent = ["", "", "Churn: Yes", "Churn: Yes", "Churn: No", "Churn: No"]
values = [1869, 5163, 939, 930, 930, 2544, 2619]

# Create a sunburst chart
fig = go.Figure(go.Sunburst(
    labels=labels,
    parents=parent,
    values=values,
))

# Customize the layout
fig.update_layout(
    margin=dict(t=0, l=0, r=0, b=0),
    sunburstcolorway=["paleturquoise", 'hotpink', "rgb(136,204,238)", '#e377c2'],
)

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



There is negligible difference in customer percentage/ count who chnaged the service provider. Both genders behaved in similar fashion when it comes to migrating to another service provider/ firm.

In [29]:

```
# Create a custom color map
color_map = {
    "Month-to-month": "orchid",
    "One year": "powderblue ",
    "Two year": "yellow"
}

# Create the histogram with custom colors
fig = px.histogram(df, x="Churn", color="Contract", barmode="group", title="<b>Customer contract distribu
                   color_discrete_map=color_map)

# Customize the layout
fig.update_layout(template="plotly_white", width=700, height=500, bargap=0.1)

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

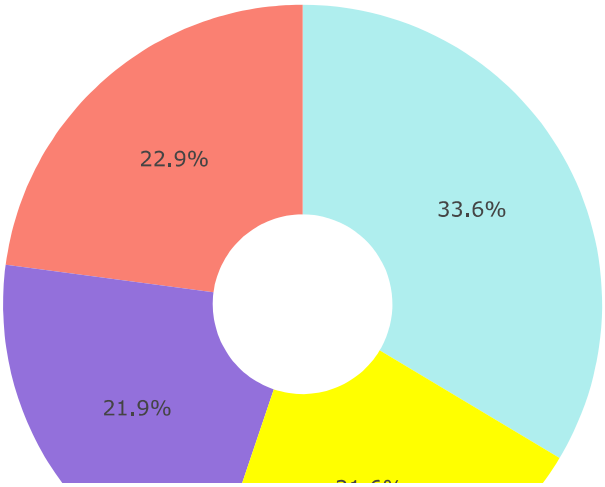


About 75% of customer with Month-to-Month Contract opted to move out as compared to 13% of customrs with One Year Contract and 3% with Two Year Contract

In [30]:

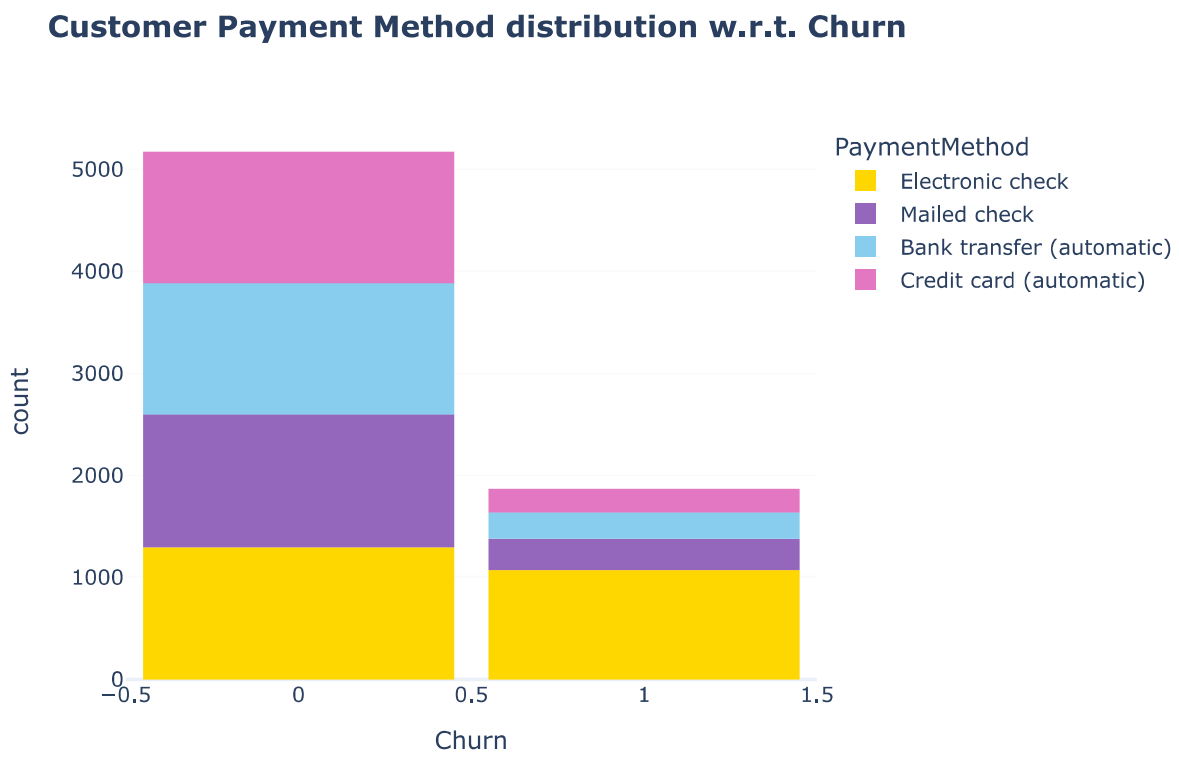
```
labels = df['PaymentMethod'].unique()
values = df['PaymentMethod'].value_counts()

colors = ["paleturquoise", 'salmon', "mediumpurple", 'yellow']
fig = go.Figure(data=[go.Pie(labels=labels, values=values, hole=.3,marker=dict(colors=colors))])
fig.update_layout(title_text="<b>Payment Method Distribution</b>")
fig.show()
```

Payment Method Distribution



In [31]:

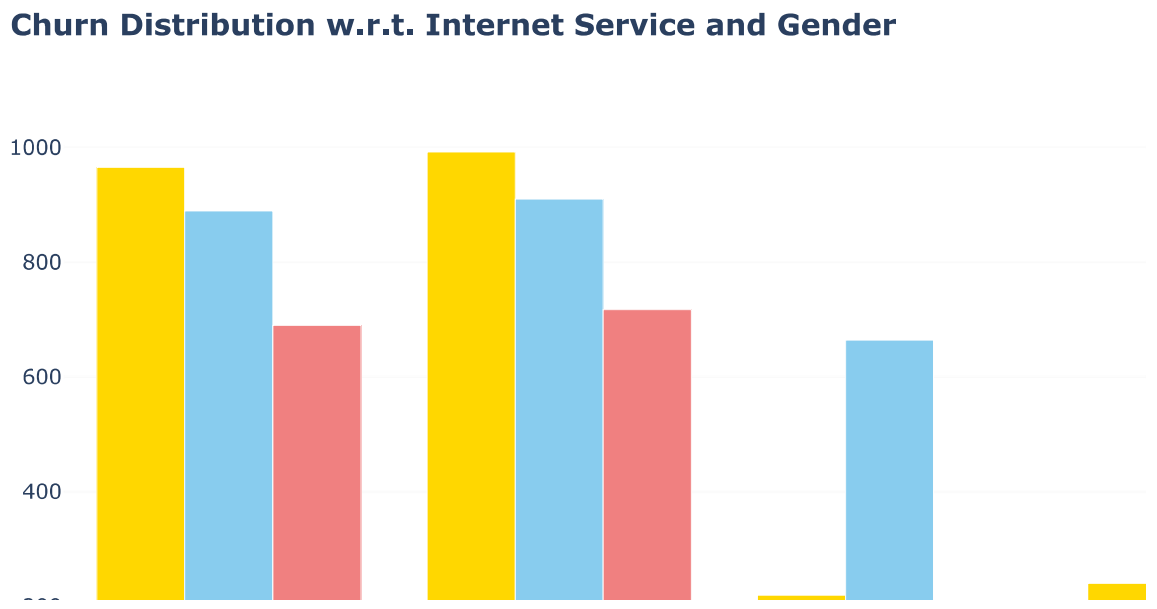
```
# Create a custom color sequence
custom_colors = ["gold", '#9467bd', "rgb(136,204,238)", '#e377c2']

# Create the histogram with custom colors
fig = px.histogram(df, x="Churn", color="PaymentMethod", title="<b>Customer Payment Method distribution w
                   color_discrete_sequence=custom_colors)

# Customize the layout
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)

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



- 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.

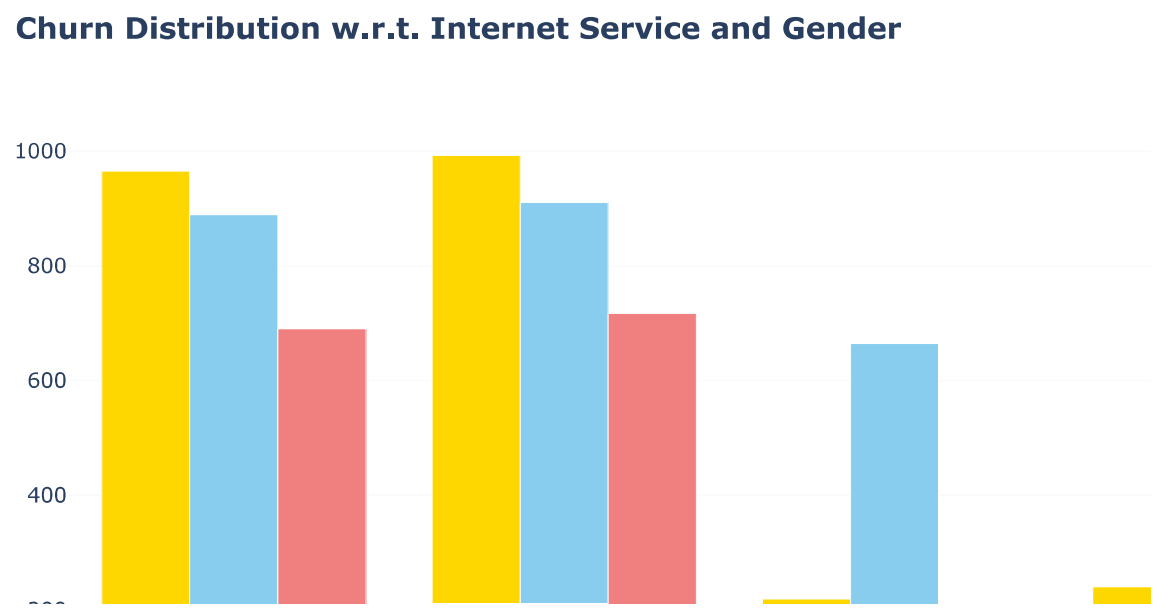
In [32]:

```
# Custom colors for Internet Services
custom_colors = ['gold', 'rgb(136,204,238)', 'lightcoral']

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',marker=dict(color=custom_colors[0])
))

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',marker=dict(color=custom_colors[1])
))

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',marker=dict(color=custom_colors[2])
))

fig.update_layout(template="plotly_white",title_text="<b>Churn Distribution w.r.t. Internet Service and G

fig.show()
```



In [33]:

```
internet_services = ['DSL', 'Fiber optic', 'No Internet']
churn_labels = ['Churn: No', 'Churn: No', 'Churn: Yes', 'Churn: Yes']
gender = ['Female', 'Male', 'Female', 'Male']

values = {
    'DSL': [965, 992, 219, 240],
    'Fiber optic': [889, 910, 664, 633],
    'No Internet': [690, 717, 56, 57],
}

# Custom colors for Internet Services
custom_colors = ['gold', 'rgb(136,204,238)', 'lightcoral']

# Create a Figure
fig = go.Figure()

# Iterate through Internet Service categories
for i, service in enumerate(internet_services):
    trace = go.Bar(
        x=[churn_labels, gender],
        y=values[service],
        name=service,
        marker=dict(color=custom_colors[i])
    )
    fig.add_trace(trace)

# Customize the layout
fig.update_layout(template="plotly_white",title_text="<b>Churn Distribution w.r.t. Internet Service and G

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

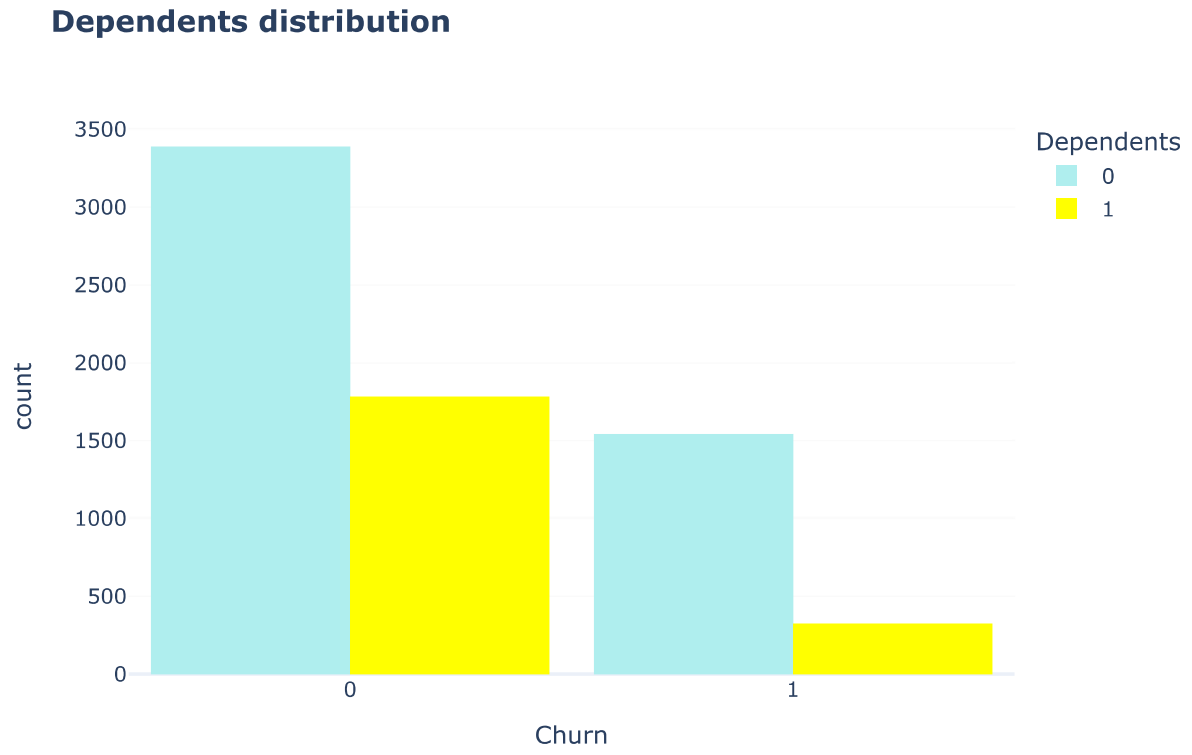


- A lot of customers choose the Fiber optic service and it's also evident that the customers who use Fiber optic have high churn rate, this might suggest a dissatisfaction with this type of internet service.

- Customers having DSL service are majority in number and have less churn rate compared to Fibre optic service.

In [34]:

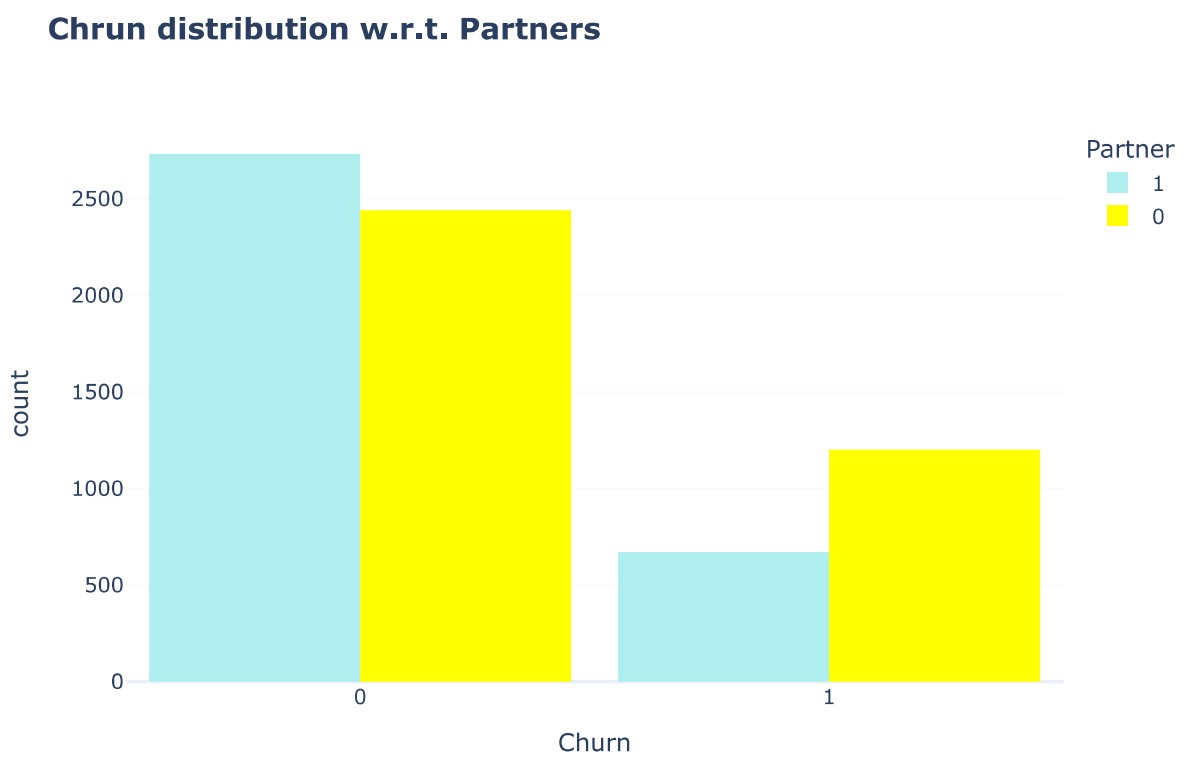
```
color_map = ["paleturquoise", "yellow"]
fig = px.histogram(df, x="Churn", color="Dependents", barmode="group", title="<b>Dependents distribution<
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



Customers without dependents are more likely to churn

In [35]:

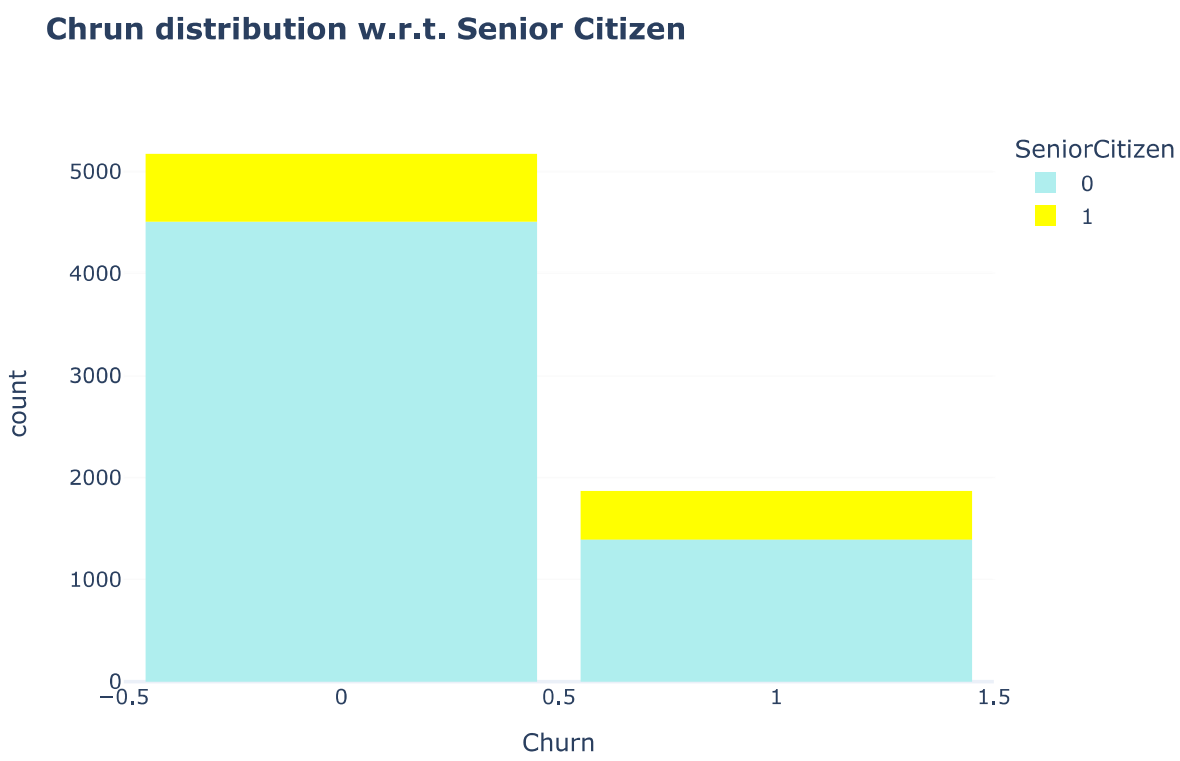
```
color_map = ["paleturquoise", "yellow"]
fig = px.histogram(df, x="Churn", color="Partner", barmode="group", title="<b>Chrun distribution w.r.t. P
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



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

In [36]:

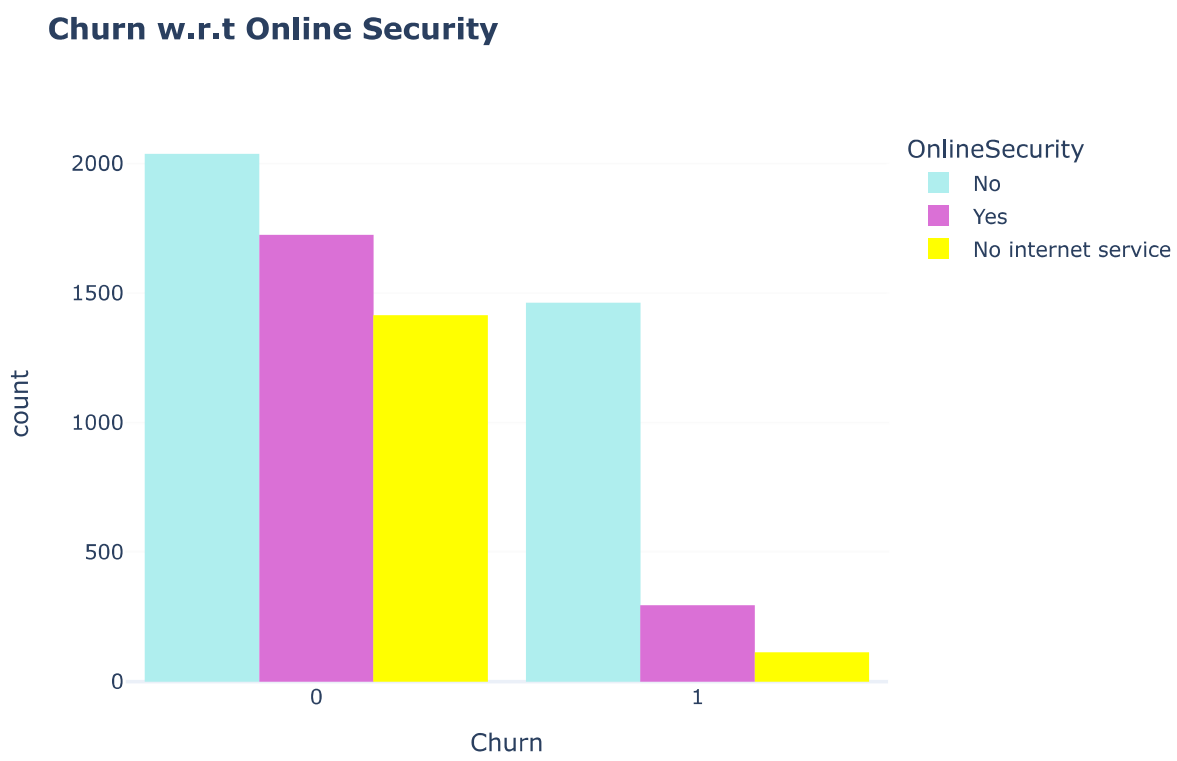
```
color_map = ["paleturquoise", "yellow"]
fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution w.r.t. Senior Citiz
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



- It can be observed that the fraction of senior citizen is very less.
- Most of the senior citizens churn.

In [37]:

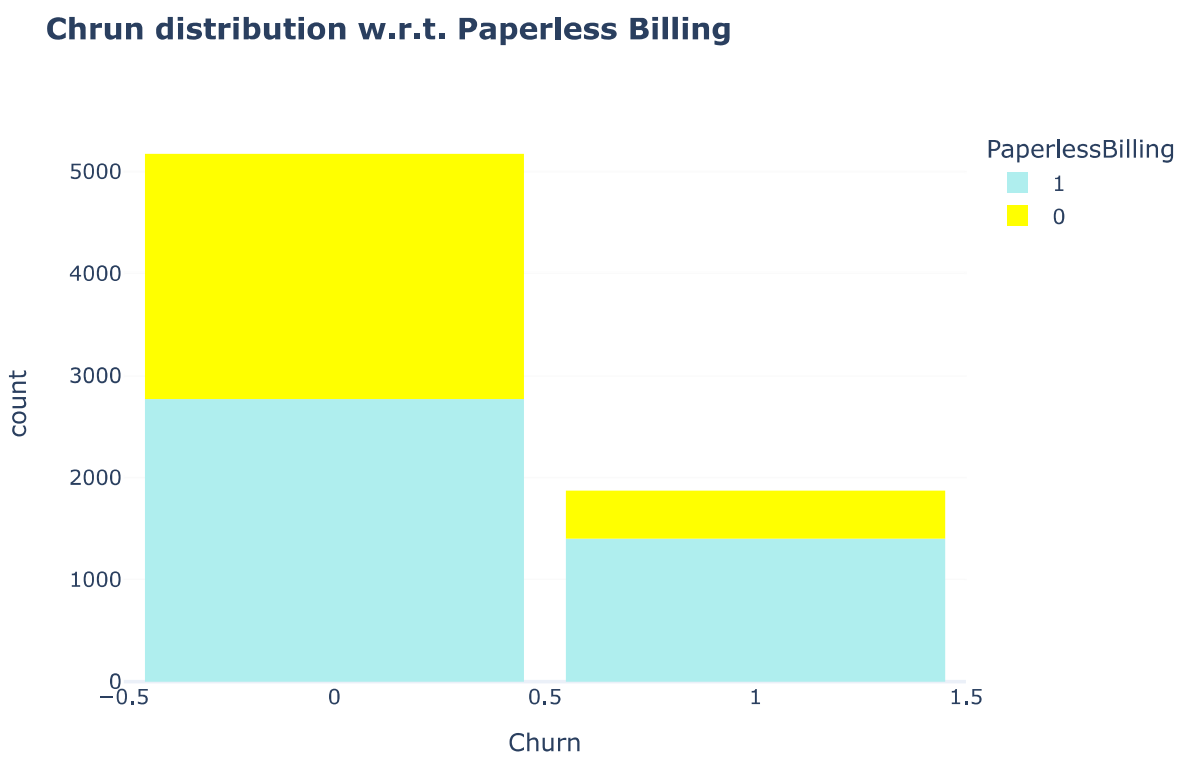
```
color_map = ["paleturquoise",'orchid',"yellow"]
fig = px.histogram(df, x="Churn", color="OnlineSecurity", barmode="group", title="<b>Churn w.r.t Online S
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



Most customers churn in the absence of online security

In [38]:

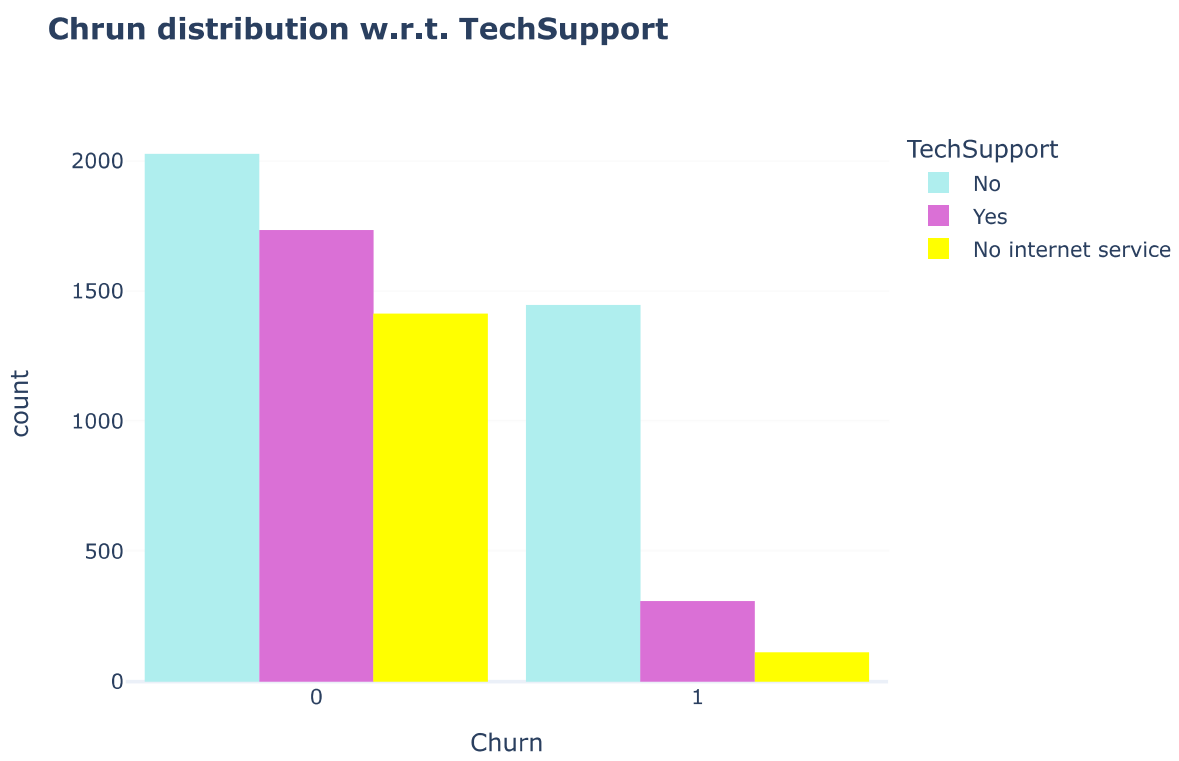
```
color_map = ["paleturquoise", "yellow"]
fig = px.histogram(df, x="Churn", color="PaperlessBilling",  title="<b>Chrun distribution w.r.t. Paperless
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



Customers with Paperless Billing are most likely to churn.

In [39]:

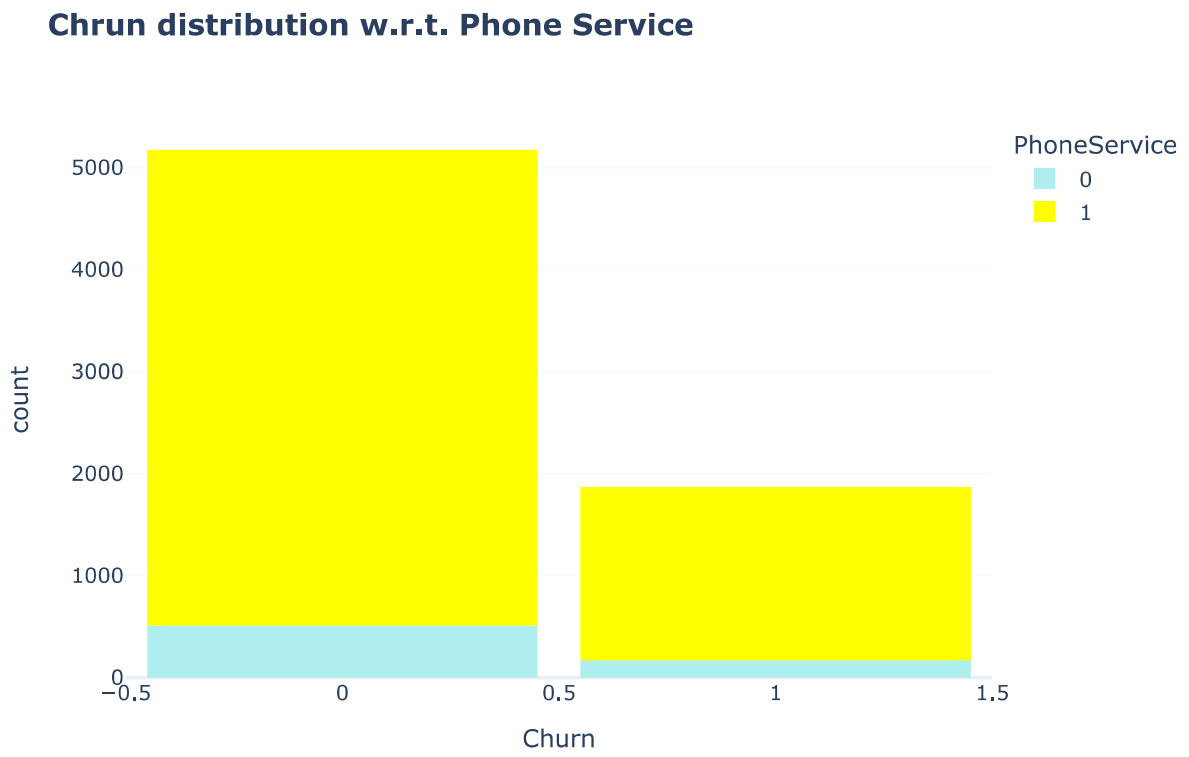
```
color_map = ["paleturquoise",'orchid',"yellow"]
fig = px.histogram(df, x="Churn", color="TechSupport",barmode="group",  title="<b>Chrun distribution w.r.
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



Customers with no TechSupport are most likely to migrate to another service provider.

In [40]:

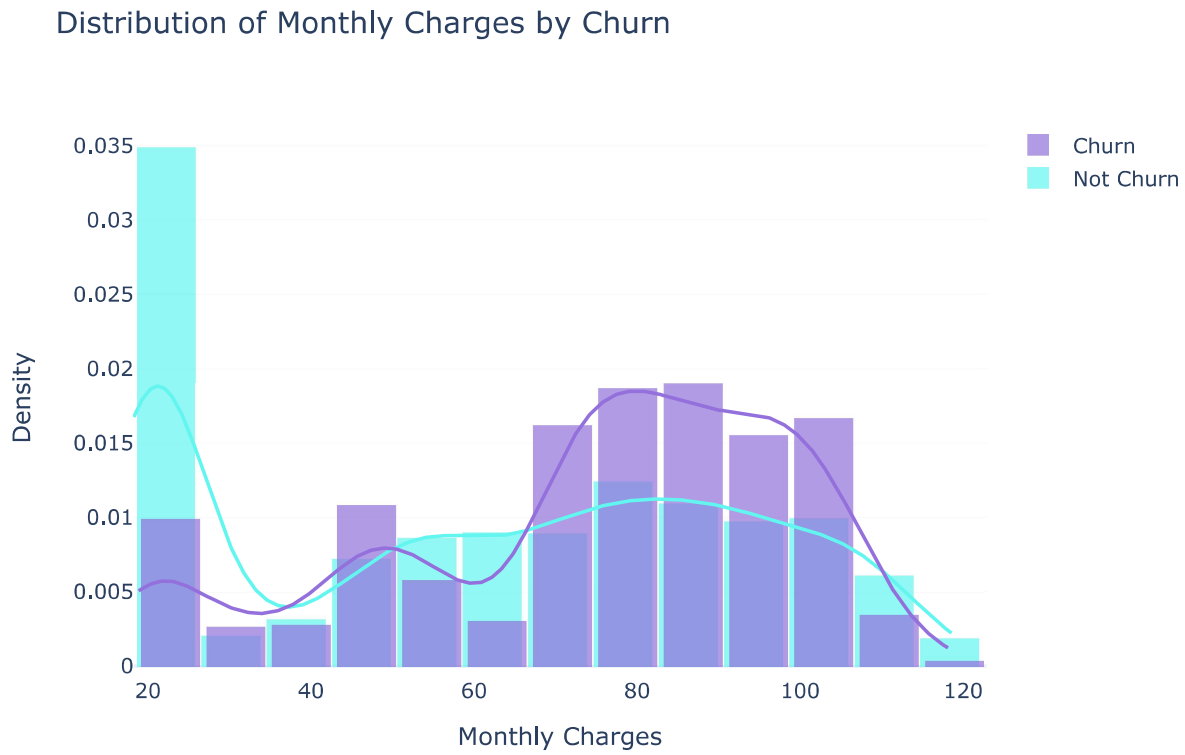
```
color_map = ["paleturquoise", "yellow"]
fig = px.histogram(df, x="Churn", color="PhoneService", title="<b>Chrun distribution w.r.t. Phone Service
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1)
fig.show()
```



Very small fraction of customers don't have a phone service and out of that, 1/3rd Customers are more likely to churn.

In [41]:

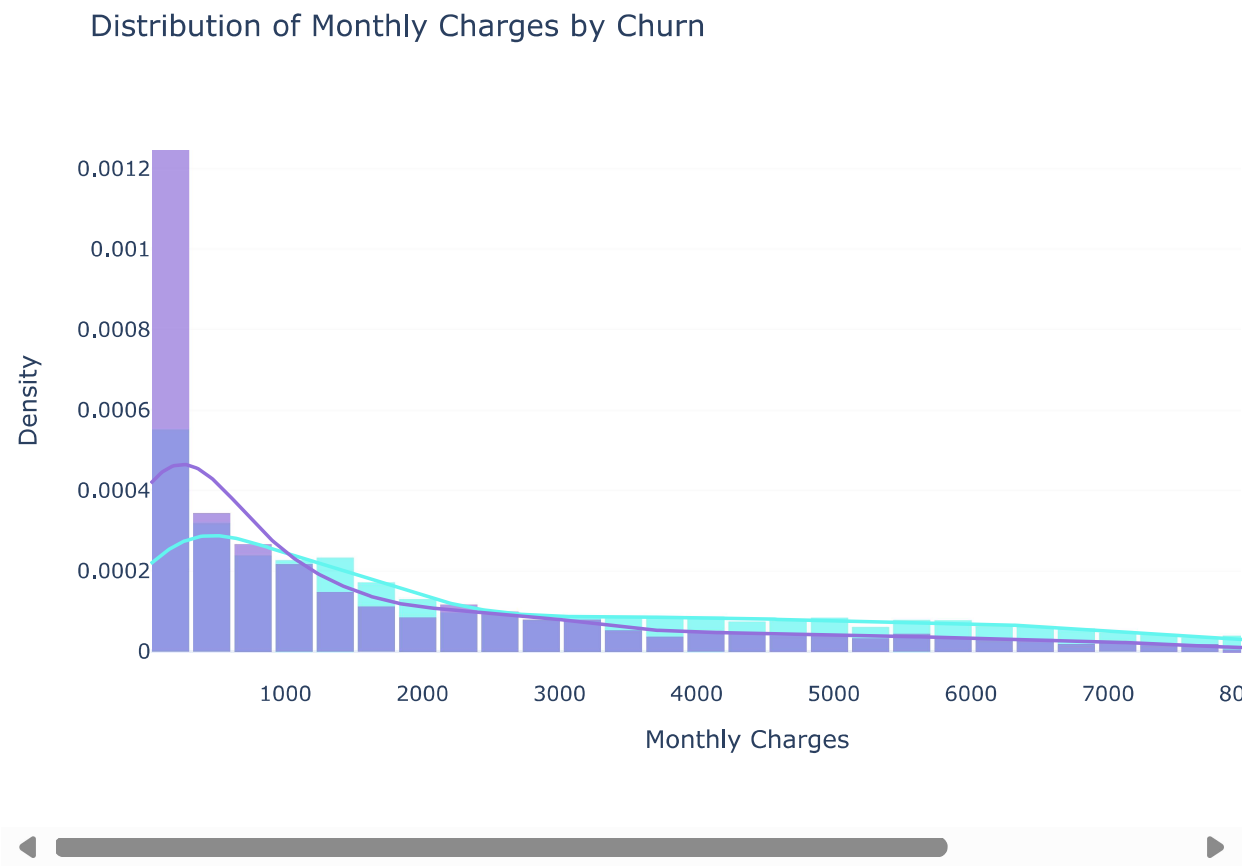
```
# kdeplot as distplot in Plotly

# Assuming you have a DataFrame 'df' containing your data

# Create a figure using ff.create_distplot
fig = ff.create_distplot(
    [df['MonthlyCharges'][df['Churn'] == 0], df['MonthlyCharges'][df['Churn'] == 1]],
    group_labels=['Not Churn', 'Churn'],
    colors=['#63F5EF',"mediumpurple"],bin_size=[8,8],
    show_rug=False, curve_type='kde'
)

# Update the layout to set font scale and paper context
fig.update_layout(template="plotly_white",width=700, height=500, bargap=0.1,
    title="Distribution of Monthly Charges by Churn",
    xaxis_title="Monthly Charges",
    yaxis_title="Density",
)

# Show the figure
fig.show()
```



Customers with higher Monthly Charges are also more likely to churn

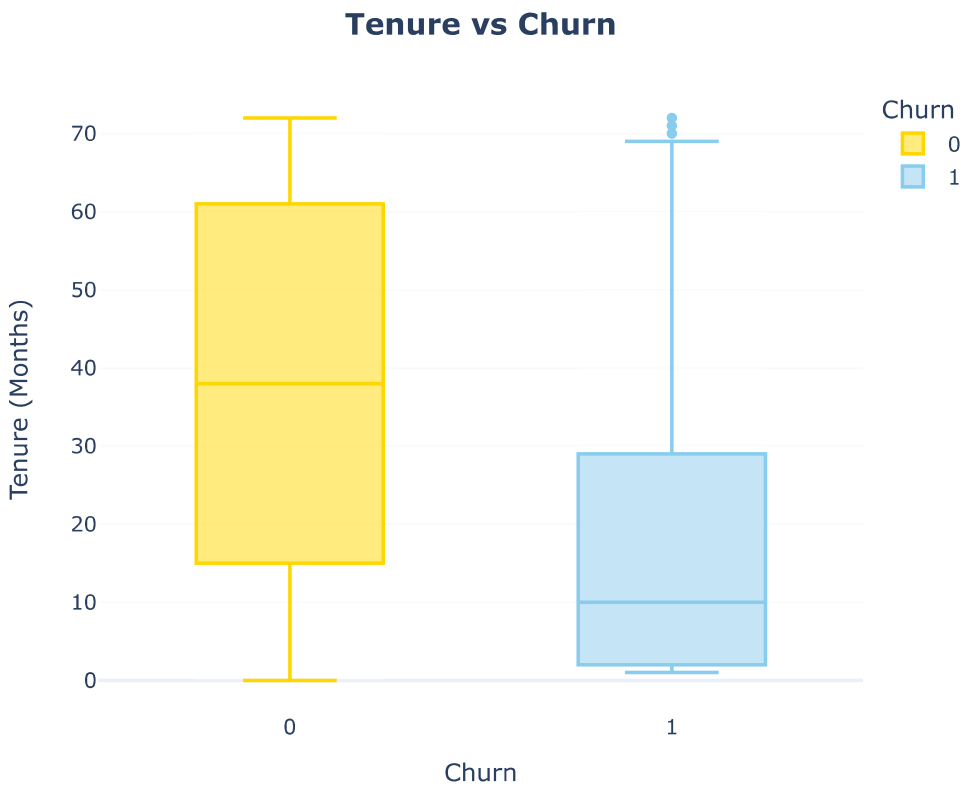
In [42]:

```
# Create a figure using ff.create_distplot
fig = ff.create_distplot(
    [df['TotalCharges'][df['Churn'] == 0], df['TotalCharges'][df['Churn'] == 1]],
    group_labels=['Not Churn', 'Churn'],
    colors=['#63F5EF',"mediumpurple"],bin_size=[300,300],
    show_rug=False, curve_type='kde'
)

# Update the layout to set font scale and paper context
fig.update_layout(template="plotly_white",width=900, height=500, bargap=0.1,
    title="Distribution of Monthly Charges by Churn",
    xaxis_title="Monthly Charges",
    yaxis_title="Density",
)

# Show the figure
fig.show()
```



In [43]:

```
# Define custom colors for 'Churn' categories
colors = ["gold", 'rgb(136,204,238)']

# Create a box plot with custom colors
fig = px.box(
    df,
    x='Churn',
    y='tenure',
    color='Churn',
    color_discrete_sequence=colors
)

# Update y-axis properties
fig.update_yaxes(title_text='Tenure (Months)')

# Update x-axis properties
fig.update_xaxes(title_text='Churn')

# Update size and title
fig.update_layout(
    template="plotly_white",
    autosize=True,
    width=600,
    height=500,
    title_font=dict(size=25, family='Courier'),
    title='<b>Tenure vs Churn</b>',title_x=0.5
)

# Show the figure
fig.show()
```

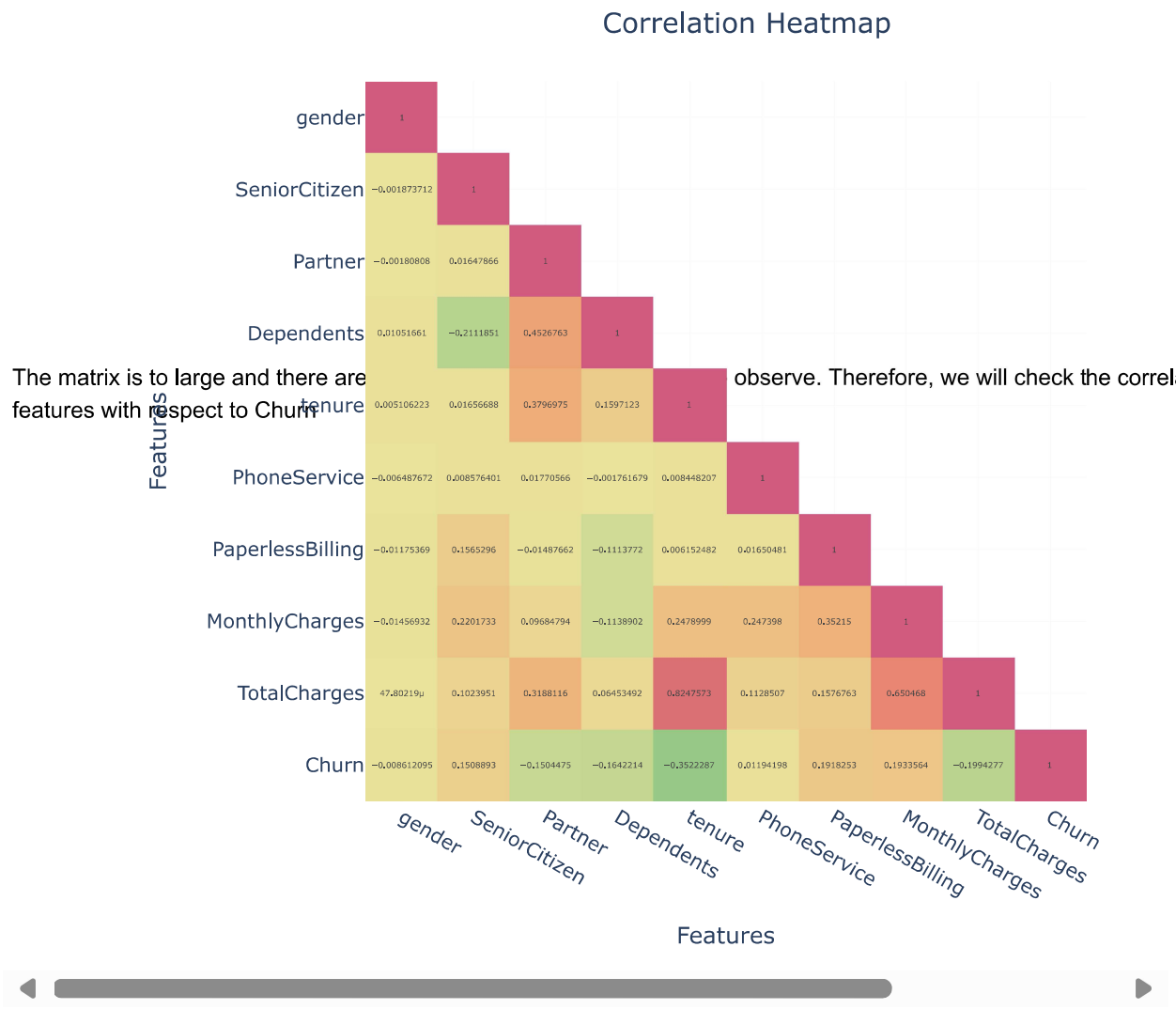


4.2 Correlation between variables

In [44]:

```
# Assuming you have a DataFrame 'df' containing your data

# Calculate the correlation matrix
corr = df.corr()

# Create a mask for the upper triangular part
mask = np.triu(np.ones(corr.shape), k=1)

# Set upper triangular values to NaN
corr[mask == 1] = None

# Create a correlation heatmap using Plotly Express
fig = px.imshow(
    corr,
    color_continuous_scale='temps',
    zmin=-1,
    zmax=1,
    x=corr.columns,
    y=corr.columns,text_auto=True
)

# Update the layout
fig.update_layout(template="plotly_white",
    title='Correlation Heatmap',title_x=0.5,
    xaxis_title='Features',
    yaxis_title='Features',
    width=900,
    height=600,
)

# Show the figure
fig.show()
```

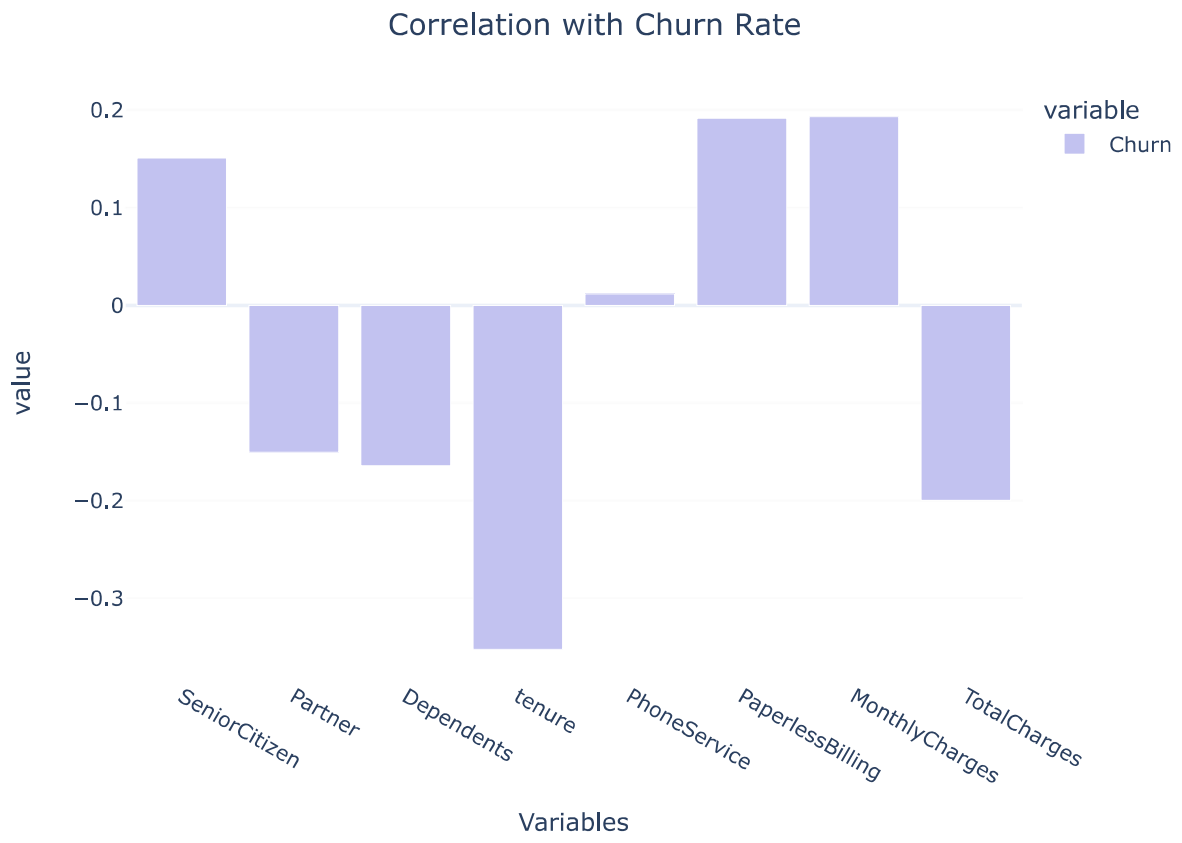
```
C:\Users\Muthu Ishwarya\AppData\Local\Temp\ipykernel_128\1932477654.py:4: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it
will default to False. Select only valid columns or specify the value of numeric_only to si
lence this warning.
```



The matrix is to large and there are observe. Therefore, we will check the correlation of features with respect to Churn

In [45]:

```
# Include 'Churn' in the selected variables
ds_corr = df[['Churn', 'SeniorCitizen', 'Partner', 'Dependents',
              'tenure', 'PhoneService', 'PaperlessBilling',
              'MonthlyCharges', 'TotalCharges']]

# Calculate the correlations
correlations = ds_corr.corr()

# Create a bar plot using Plotly Express
fig = px.bar(correlations['Churn'].drop('Churn'),
             labels={'index': 'Variables', 'Churn': 'Correlation'},
             color_discrete_sequence=['#c2c2f0'],
             width=700,
             height=500)

# Update size and title
fig.update_layout(template="plotly_white",title='Correlation with Churn Rate',title_x=0.5)


# Show the figure
fig.show()
```



In terms of positive correlation with **churn**, there are three variables **senior citizen, paperless billing, and monthly charges**:

- It can be interpreted that senior citizen has tendency to churn. The reasons for that could related to external factors such as competitors.
- A positive relationship between monthly charges and churn is explainable since the more charged amounts per month, the more likely customers churn.
- Interesting insight to notice is that the paperless billing is actually positively correlate with churn.

In terms of negative relationship:

- A longer **tenure** could refer to loyalty, which results in less churn risk. Therefore, the negative relationship is presented.
- In constrast to monthly charges relationship with churn, it is interesting that **total charges** negatively correlated to churn. The reason can be that total charges depend on the time the customer has spent with a company (as tenure also has a negative

relationship). Additionally, this variable remains questionable whether it is tracked by the customer in order to understand customer behavior.

Since the Contract type has 3 unique values, we will use **Hot encoding for categorical data** to transform variables having 3 or more than 3 categorical data.

In [46]:

```
# Copy data to new 'dataset' variable to conserve original values
dataset = df.copy()
```

In [47]:

```
# Hot-Encoding for categorical data
dataset = pd.get_dummies(dataset)
dataset.head()
```

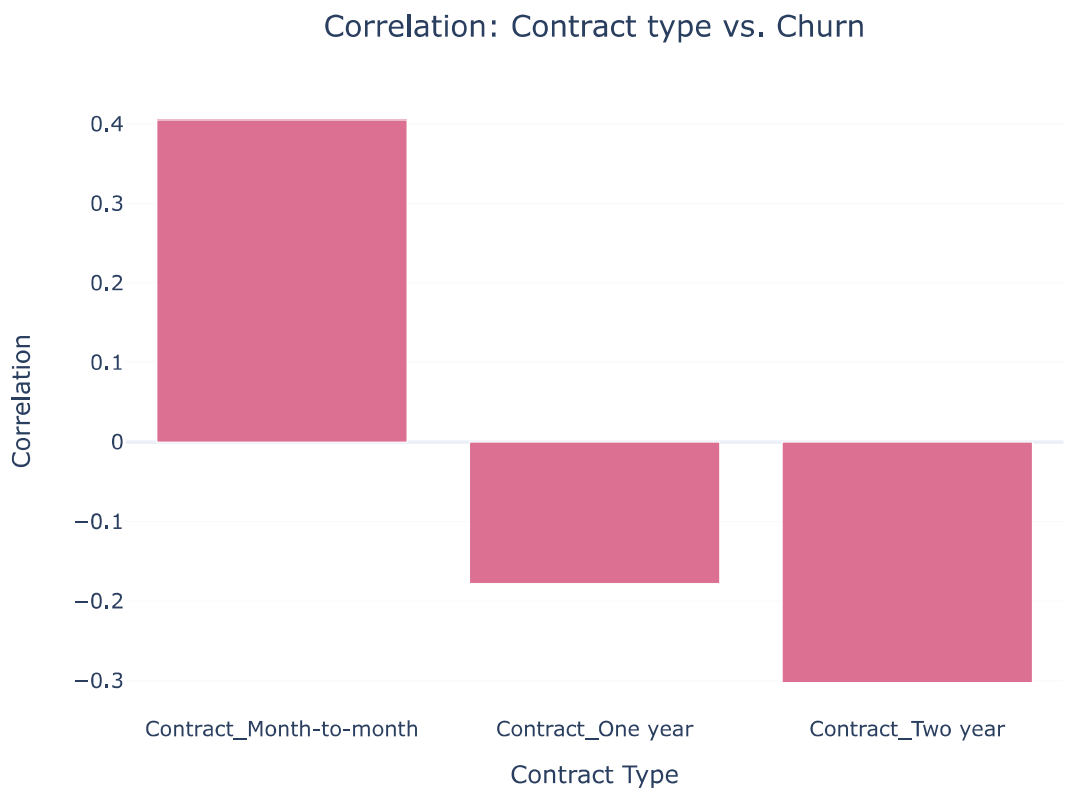
Out[47]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharges	Tc
0	0	0	1	0	1	0	1	29.85	
1	1	0	0	0	34	1	0	56.95	
2	1	0	0	0	2	1	1	53.85	
3	1	0	0	0	45	0	0	42.30	
4	0	0	0	0	2	1	1	70.70	

5 rows × 41 columns

In [48]:

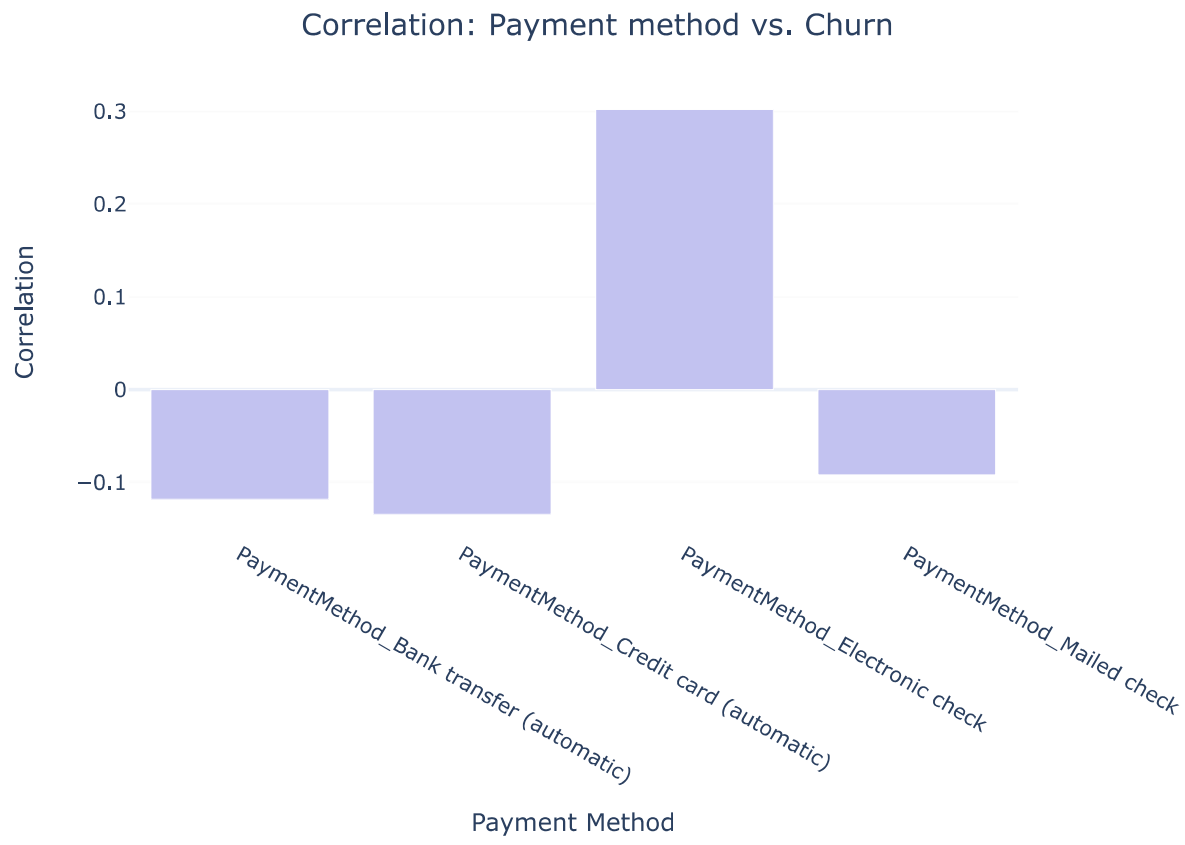
```
import plotly.express as px
import pandas as pd

# Assuming you have a DataFrame 'dataset' containing your data

# Select the 'Contract' type variables
ds_contract_type_corr = dataset[['Churn','Contract_Month-to-month', 'Contract_One year', 'Contract_Two ye

# Calculate the correlations with 'Churn'
correlations = ds_contract_type_corr.corrwith(dataset['Churn']).drop('Churn')

# Create a DataFrame for plotting
corr_df = pd.DataFrame({'Contract Type': correlations.index, 'Correlation': correlations.values})

# Create a bar plot using Plotly Express
fig = px.bar(corr_df, x='Contract Type', y='Correlation', color_discrete_sequence=['palevioletred'])

# Update the layout
fig.update_layout(template="plotly_white",
    title='Correlation: Contract type vs. Churn',title_x=0.5,
    xaxis_title='Contract Type',
    yaxis_title='Correlation',
    width=700,
    height=500,
)

# Show the figure
fig.show()
```



Subscription Month-to-month plan is most exposed to a churn risk. Longer contract duration such as one year and two year is a good strategy to prevent churn.

In [49]:

```
import plotly.express as px
import pandas as pd

# Assuming you have a DataFrame 'dataset' containing your data

# Select the 'PaymentMethod' variables
ds_payment_method_corr = dataset[['Churn','PaymentMethod_Bank transfer (automatic)',
                                  'PaymentMethod_Credit card (automatic)',
                                  'PaymentMethod_Electronic check',
                                  'PaymentMethod_Mailed check']]

# Calculate the correlations with 'Churn'
correlations = ds_payment_method_corr.corrwith(dataset['Churn']).drop('Churn')

# Create a DataFrame for plotting
corr_df = pd.DataFrame({'Payment Method': correlations.index, 'Correlation': correlations.values})

# Create a bar plot using Plotly Express
fig = px.bar(corr_df, x='Payment Method', y='Correlation', color_discrete_sequence=['#c2c2f0'])

# Update the layout
fig.update_layout(template="plotly_white",
    title='Correlation: Payment method vs. Churn',title_x=0.5,
    xaxis_title='Payment Method',
    yaxis_title='Correlation',
    width=700,
    height=500,
)

# Show the figure
fig.show()
```



From the positive correlation between electronic check payment method and churn, marketing and product team should take consideration to investigate the reasons.

4.3 Identify Multicollinearity

High correlation among independent variables gives rise to a phenomenon known as **multicollinearity**, indicating that one independent variable can be predicted from another. Variables exhibiting significant multicollinearity are essentially duplicative and can introduce challenges in model interpretation, potentially leading to overfitting issues.

Overfitting is a statistical modeling error that occurs when a function is excessively tailored to a specific set of data points. Consequently, the model's usefulness is limited to the initial dataset and does not generalize well to other datasets.

The **Variable Inflation Factor (VIF)** serves as a valuable tool for assessing multicollinearity. It gauges the strength of a variable's correlation with a group of other independent variables within a dataset. VIF typically starts at 1, and a value exceeding 10 suggests substantial multicollinearity among independent variables.

In [50]:

```
# Multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [51]:

```
# Calculate Variable Inflation Factors

def calculate_vif(X):
# Calculate Variable Inflation Factors
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["Variable Inflation Factors"] = [variance_inflation_factor(X.values, i)
    for i in range(X.shape[1])]
    return(vif)

ds_vif = dataset[['gender', 'SeniorCitizen', 'Partner', 'Dependents',\
                  'tenure', 'PhoneService', 'PaperlessBilling', \
                  'MonthlyCharges','TotalCharges']]
```

In [52]:

```
vif = calculate_vif(ds_vif)
vif
```

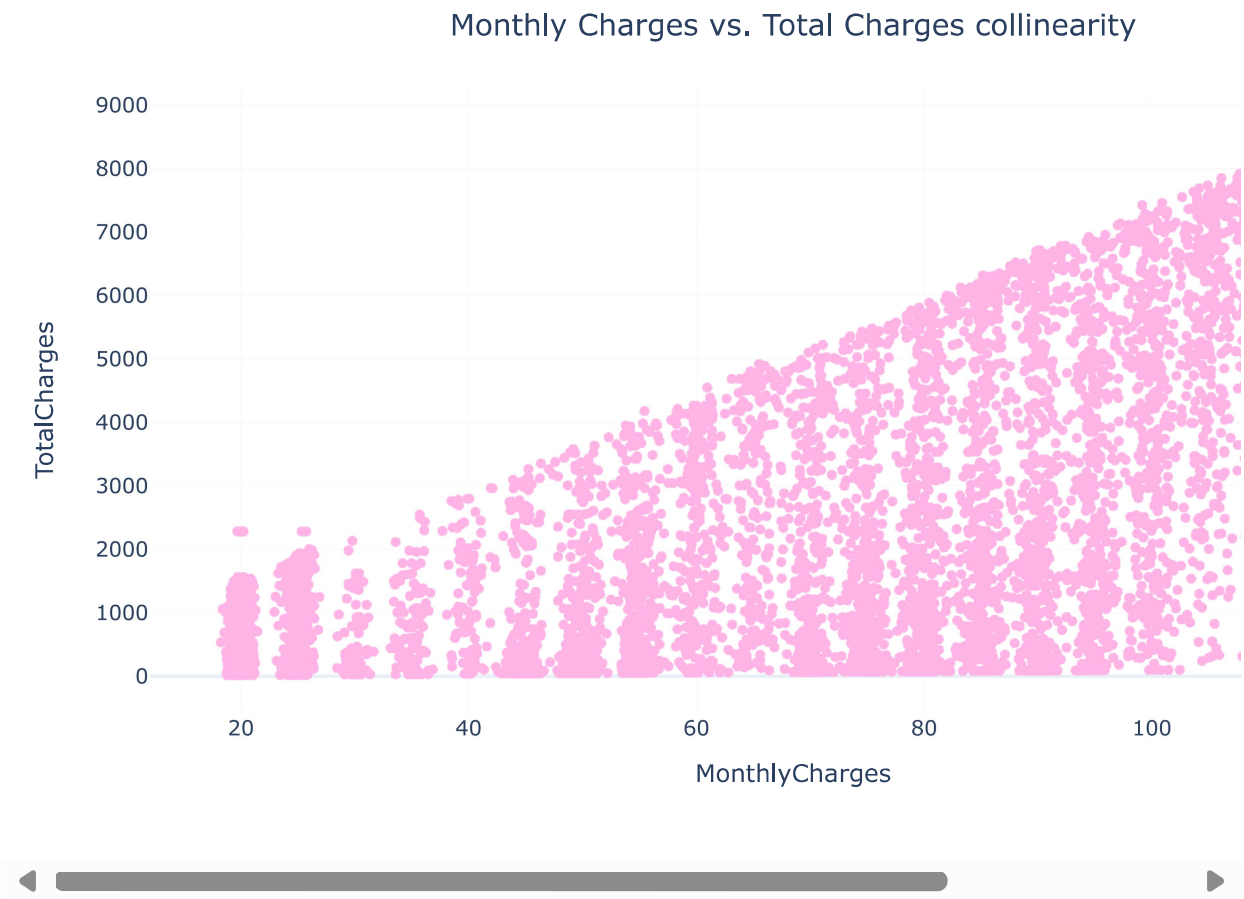
Out[52]:

	variables	Variable Inflation Factors
0	gender	1.921285
1	SeniorCitizen	1.327766
2	Partner	2.815272
3	Dependents	1.921208
4	tenure	10.549726
5	PhoneService	7.976437
6	PaperlessBilling	2.814154
7	MonthlyCharges	13.988695
8	TotalCharges	12.570370

It is noticeable that **Monthly Charges, Total Charges** have a high VIF score. We will visualise to examine how they are correlated to each other by using scatter plot.

In [53]:

```
# Create a scatter plot using Plotly Express
fig = px.scatter(
    ds_vif,
    x='MonthlyCharges',
    y='TotalCharges',
    color_discrete_sequence=['#ffb3e6']
)
# Update the layout
fig.update_layout(template="plotly_white",
    title='Monthly Charges vs. Total Charges collinearity',title_x=0.5,
    width=900,
    height=500,
)

# Show the figure
fig.show()
```



The scatter plot reveals that the **TotalCharges and Monthly Charges** variables exhibit collinearity. To mitigate multicollinearity, it is advisable to eliminate one of these correlated features. The most effective strategy involves discarding the **Total Charges** variable while retaining **Monthly Charges** because of its strong positive correlation with the **Churn** variable.

In [54]:

```
# Drop 'TotalCharges' from VIF test dataset
ds_vif2 = ds_vif.drop(columns = "TotalCharges")

# Check colinearity again
vif2 = calculate_vif(ds_vif2)
vif2
```

Out[54]:

	variables	Variable Inflation Factors
0	gender	1.879536
1	SeniorCitizen	1.323089
2	Partner	2.814574
3	Dependents	1.908533
4	tenure	3.287603
5	PhoneService	5.963240
6	PaperlessBilling	2.745897
7	MonthlyCharges	7.453993

The removal of the **Total Charges** variable effectively decreased multicollinearity among correlated features in the test dataset, which includes **tenure.** As the final step of our analysis for Machine Learning algorithms, it is essential to exclude **Total Charges** from the primary dataset.

In [55]:

```
# Drop the "Total Charges" from main dataset
dataset = dataset.drop(columns = "TotalCharges")
print(dataset.dtypes)
```

```
gender                                     int8
SeniorCitizen                              int8
Partner                                    int8
Dependents                                 int8
tenure                                    int64
PhoneService                               int8
PaperlessBilling                           int8
MonthlyCharges                          float64
Churn                                      int8
MultipleLines_No                          uint8
MultipleLines_No phone service            uint8
MultipleLines_Yes                         uint8
InternetService_DSL                       uint8
InternetService_Fiber optic               uint8
InternetService_No                        uint8
OnlineSecurity_No                         uint8
OnlineSecurity_No internet service        uint8
OnlineSecurity_Yes                        uint8
OnlineBackup_No                           uint8
OnlineBackup_No internet service          uint8
OnlineBackup_Yes                          uint8
DeviceProtection_No                       uint8
DeviceProtection_No internet service      uint8
DeviceProtection_Yes                      uint8
TechSupport_No                            uint8
TechSupport_No internet service           uint8
TechSupport_Yes                           uint8
StreamingTV_No                            uint8
StreamingTV_No internet service           uint8
StreamingTV_Yes                           uint8
StreamingMovies_No                        uint8
StreamingMovies_No internet service       uint8
StreamingMovies_Yes                       uint8
Contract_Month-to-month                   uint8
Contract_One year                         uint8
Contract_Two year                         uint8
PaymentMethod_Bank transfer (automatic)   uint8
PaymentMethod_Credit card (automatic)     uint8
PaymentMethod_Electronic check            uint8
PaymentMethod_Mailed check                uint8
dtype: object
```

5. Build Machine Learning Models

There are two primary categories of machine learning problems:

1. **Supervised Learning**: This type of learning is applied to a dataset of historical data points with the goal of making predictions about future data. Supervised learning can be further categorized into two subtypes:

- **Classification Problems**: These problems involve predicting a discrete set of values or categories. For example, classifying a handwritten character into a known character or determining if an email is spam.
- **Regression**: In regression problems, the objective is to predict a continuous quantity or value for a given set of data. For instance, predicting house prices based on factors such as house size, nearby schools, and other relevant factors, or forecasting sales revenue using historical sales data.

The target variable for **churn** has two states: "yes" or "no," often represented as 1 or 0. This makes it a binary classification problem.

2. Unsupervised Learning: Unsupervised learning is employed to address problems where we have limited or no prior knowledge of the expected outcomes. In this approach, algorithms aim to discover hidden patterns or structures within the

data and group or cluster the data in a way that makes the most sense based on the available information. An example of unsupervised learning includes neural networks, such as those used in chatbots like the LEX session.

In summary, in this dataset, we will **use supervised learning algorithms**.

In [56]:

```
# Split the dataset into features (X) and target variable (y)

X = dataset.drop(columns=['Churn'])  # Features
y = dataset['Churn']  # Target variable
```

In [57]:

```
from sklearn.model_selection import train_test_split
```

In [58]:

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
```

In [59]:

```
# Check for missing values in the training set
missing_train = X_train.isnull().sum()
print("Missing values in training set:")
print(missing_train[missing_train > 0])

# Check for missing values in the test set
missing_test = X_test.isnull().sum()
print("\nMissing values in test set:")
print(missing_test[missing_test > 0])
```

```
Missing values in training set:
Series([], dtype: int64)

Missing values in test set:
Series([], dtype: int64)
```

5.1. kNN

In [60]:

```
knn_model = KNeighborsClassifier(n_neighbors = 10)

knn_model.fit(X_train,y_train)
```

Out[60]:

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=10)
```

In [61]:

```
# Evaluate model
accuracy_knn = knn_model.score(X_test,y_test)
print("Accuracy of K-Nearest Neighbor: ", accuracy_knn)
```

```
Accuracy of K-Nearest Neighbor:  0.7808802650260294
```

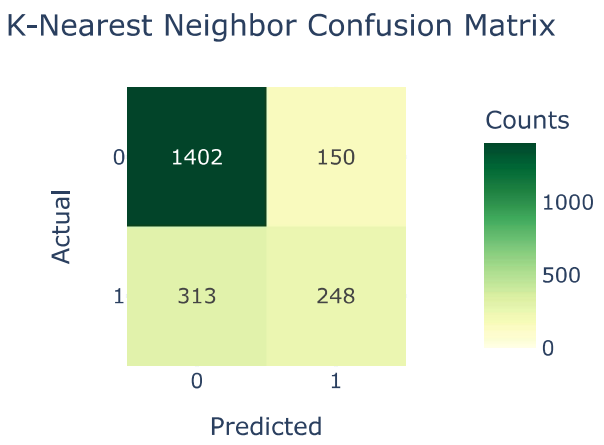
In [62]:

```
# Classification report
knn_prediction = knn_model.predict(X_test)
print(classification_report(y_test, knn_prediction))
```

```
              precision    recall  f1-score   support

           0       0.82      0.90      0.86      1552
           1       0.62      0.44      0.52       561

    accuracy                           0.78      2113
   macro avg       0.72      0.67      0.69      2113
weighted avg       0.77      0.78      0.77      2113
```

In [63]:

```
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, knn_prediction)

# Convert the confusion matrix to a DataFrame for plotting
confusion_df = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])

# Create a heatmap using Plotly Express
fig = px.imshow(
    confusion_df,text_auto=True,
    color_continuous_scale='YlGn',  # You can choose your desired color scale
    zmin=0,  # Minimum value for color scale
    zmax=confusion_df.values.max(),  # Maximum value for color scale
    labels=dict(x='Predicted', y='Actual', color='Counts')
)

# Update the layout
fig.update_layout(
    title='K-Nearest Neighbor Confusion Matrix',
    title_x=0.5,
    width=400,
    height=300,
)

# Show the figure
fig.show()
```



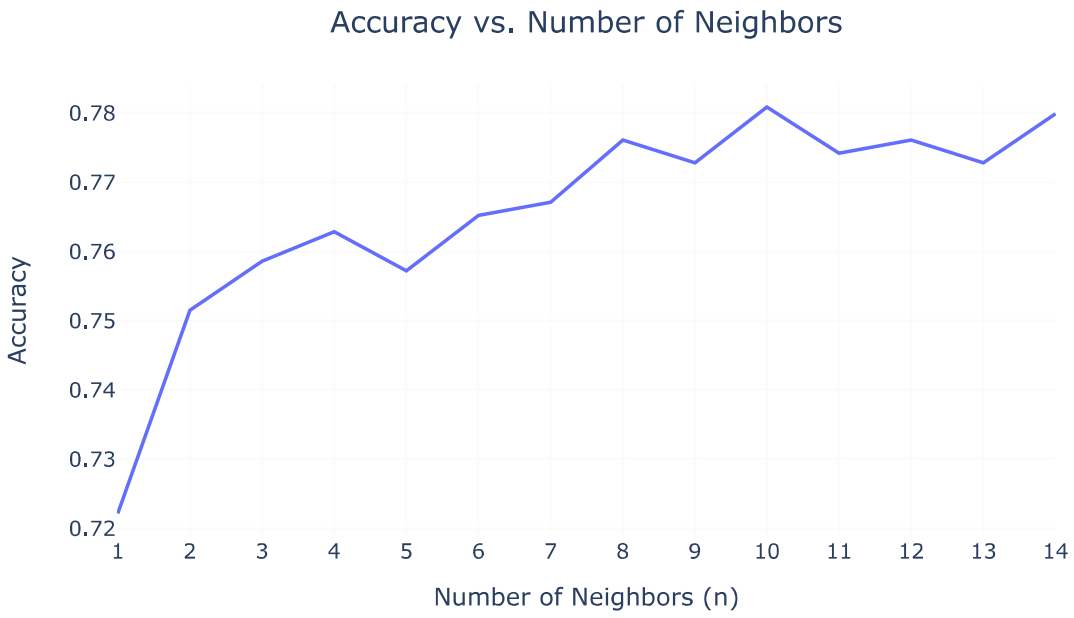
In [64]:

```
a_index = list(range(1, 15))
accuracies = []  # Initialize an empty list to store accuracy values

for i in a_index:
    model = KNeighborsClassifier(n_neighbors=i)
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    accuracy = metrics.accuracy_score(y_test, prediction)
    accuracies.append(accuracy)

print('Accuracies for different values of n are:', accuracies)
```

```
Accuracies for different values of n are: [0.7221959299574066, 0.751538097491718, 0.7586370
089919545, 0.7628963558920966, 0.7572172266919073, 0.7652626597255088, 0.7671557027922385,
0.7761476573592049, 0.7728348319924279, 0.7808802650260294, 0.7742546142924751, 0.776147657
3592049, 0.7728348319924279, 0.7799337434926644]
```

In [65]:

```
# Create a line plot using Plotly Express
fig = px.line(x=a_index, y=accuracies, labels={'x': 'Number of Neighbors (n)', 'y': 'Accuracy'})

# Update the layout
fig.update_layout(template="plotly_white",
    title='Accuracy vs. Number of Neighbors',title_x=0.5,
    xaxis_title='Number of Neighbors (n)',
    yaxis_title='Accuracy',
    xaxis=dict(tickmode='array', tickvals=a_index),
    width=700,
    height=400,
)

# Show the figure
fig.show()
```



5.2. SVM

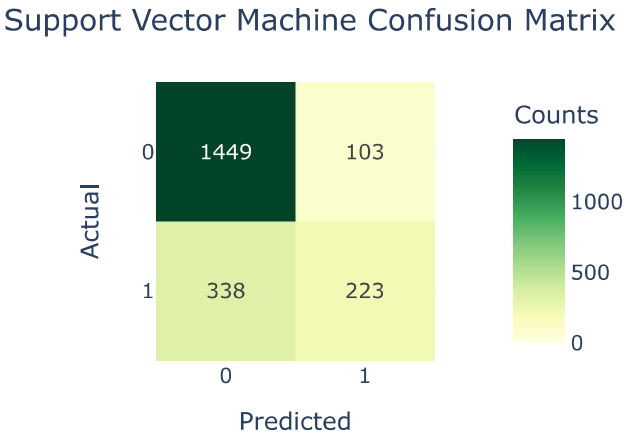
In [66]:

```
svc_model = SVC(random_state = 42)
svc_model.fit(X_train,y_train)

# Evaluate model
accuracy_svc = svc_model.score(X_test,y_test)
print("Accuracy of Support Vector Machine: ", accuracy_svc)
print("")
# Classification repoort
svc_prediction = svc_model.predict(X_test)
print(classification_report(y_test, svc_prediction))
```

```
Accuracy of Support Vector Machine:  0.791292001893043

              precision    recall  f1-score   support

           0       0.81      0.93      0.87      1552
           1       0.68      0.40      0.50       561

    accuracy                           0.79      2113
   macro avg       0.75      0.67      0.69      2113
weighted avg       0.78      0.79      0.77      2113
```

In [67]:

```
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, svc_prediction)

# Convert the confusion matrix to a DataFrame for plotting
confusion_df = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])

# Create a heatmap using Plotly Express
fig = px.imshow(
    confusion_df,text_auto=True,
    color_continuous_scale='YlGn',  # You can choose your desired color scale
    zmin=0,  # Minimum value for color scale
    zmax=confusion_df.values.max(),  # Maximum value for color scale
    labels=dict(x='Predicted', y='Actual', color='Counts')
)

# Update the layout
fig.update_layout(
    title='Support Vector Machine Confusion Matrix',
    title_x=0.5,
    width=400,
    height=300,
)

# Show the figure
fig.show()
```



5.3. Random Forest Classifier

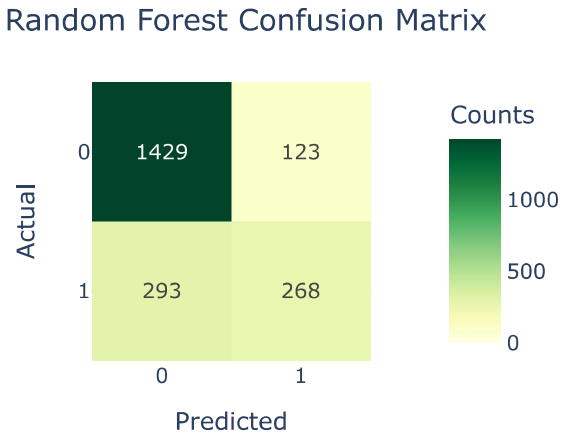
In [68]:

```
from sklearn.ensemble import RandomForestClassifier

random_forest_model = RandomForestClassifier(n_estimators=500,
                                oob_score = True, n_jobs = -1,
                                random_state=42, max_features = "auto",
                                max_leaf_nodes = 30)

random_forest_model.fit(X_train, y_train)

# Evaluate model
accuracy_random_forest = random_forest_model.score(X_test, y_test)
print("Accuracy of Random Forest: ", accuracy_random_forest)
print("")

# Classification report
random_forest_prediction = random_forest_model.predict(X_test)
print(classification_report(y_test, random_forest_prediction))
```

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\ensemble\_forest.py:424: FutureWarning:

`max_features='auto'` has been deprecated in 1.1 and will be removed in 1.3. To keep the pa
st behaviour, explicitly set `max_features='sqrt'` or remove this parameter as it is also t
he default value for RandomForestClassifiers and ExtraTreesClassifiers.


Accuracy of Random Forest:  0.8031235210601041

              precision    recall  f1-score   support

           0       0.83      0.92      0.87      1552
           1       0.69      0.48      0.56       561

    accuracy                           0.80      2113
   macro avg       0.76      0.70      0.72      2113
weighted avg       0.79      0.80      0.79      2113
```

In [69]:

```
# Calculate the confusion matrix
confusion = confusion_matrix(y_test, random_forest_prediction)

# Convert the confusion matrix to a DataFrame for plotting
confusion_df = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])

# Create a heatmap using Plotly Express
fig = px.imshow(
    confusion_df,text_auto=True,
    color_continuous_scale='YlGn',  # You can choose your desired color scale
    zmin=0,  # Minimum value for color scale
    zmax=confusion_df.values.max(),  # Maximum value for color scale
    labels=dict(x='Predicted', y='Actual', color='Counts')
)

# Update the layout
fig.update_layout(
    title='Random Forest Confusion Matrix',
    title_x=0.5,
    width=400,
    height=300,
)

# Show the figure
fig.show()
```

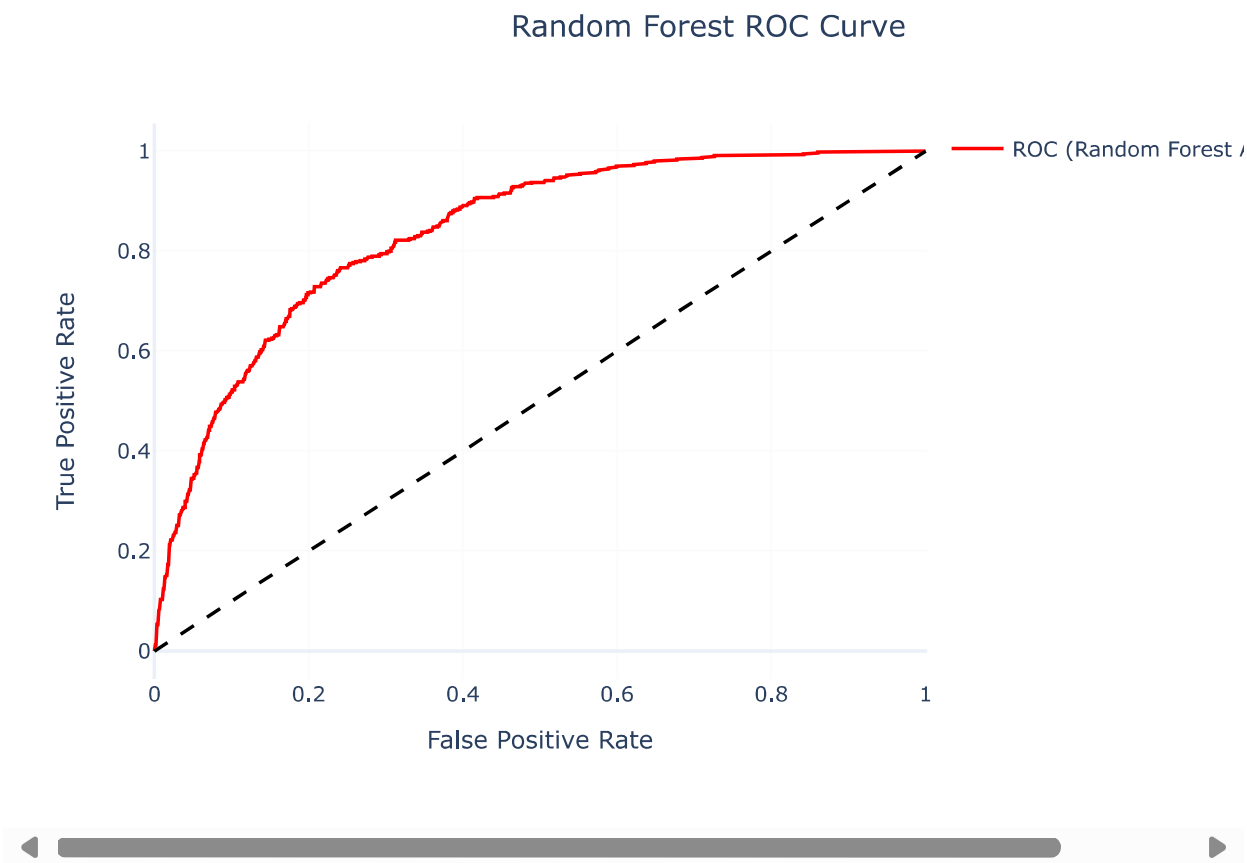


In [70]:

```
y_rfpred_prob = random_forest_model.predict_proba(X_test)[:,1]
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_rfpred_prob)

# Calculate AUC
roc_auc_rf = auc(fpr_rf, tpr_rf)
```

In [71]:

```
# Create a line chart using Plotly
fig = go.Figure()

# Add the random forest ROC curve with a legend title
fig.add_trace(go.Scatter(x=fpr_rf, y=tpr_rf, mode='lines', name='ROC (Random Forest AUC = {:.2f})'.format(

# Add the diagonal line
fig.add_shape(
    type='line',
    x0=0,
    y0=0,
    x1=1,
    y1=1,
    line=dict(dash='dash', color='black')
)

# Update the layout
fig.update_layout(
    template="plotly_white",
    xaxis_title='False Positive Rate',
    yaxis_title='True Positive Rate',
    title='Random Forest ROC Curve',
    title_x=0.5,
    showlegend=True,  # Ensure that the legend is displayed
    width=800,
    height=500,
)

# Show the figure
fig.show()
```



5.4. Decision Tree Classifier

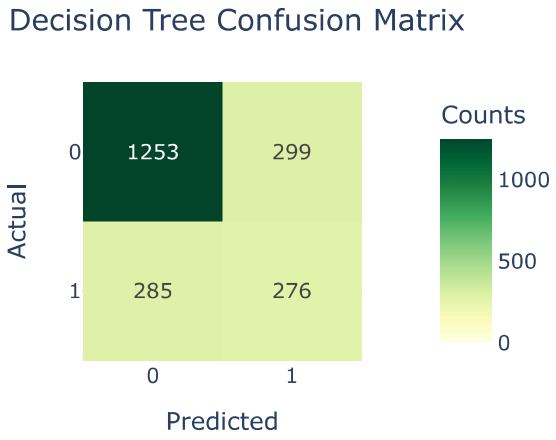
In [72]:

```
decision_tree_model = DecisionTreeClassifier(random_state=42)
decision_tree_model.fit(X_train,y_train)

# Evaluate model
accuracy_decision_tree = decision_tree_model.score(X_test, y_test)
print("Accuracy of Decision Tree: ", accuracy_decision_tree)
print('')
# Decision Tree Classifier gives very low accuracy score.

# Classification report
decision_tree_prediction = decision_tree_model.predict(X_test)
print(classification_report(y_test, decision_tree_prediction))
```

```
Accuracy of Decision Tree:  0.7236157122574538

              precision    recall  f1-score   support

           0       0.81      0.81      0.81      1552
           1       0.48      0.49      0.49       561

    accuracy                           0.72      2113
   macro avg       0.65      0.65      0.65      2113
weighted avg       0.73      0.72      0.72      2113
```

In [73]:

```
# Calculate the confusion matrix
confusion = confusion_matrix(y_test,  decision_tree_prediction)

# Convert the confusion matrix to a DataFrame for plotting
confusion_df = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])

# Create a heatmap using Plotly Express
fig = px.imshow(
    confusion_df,text_auto=True,
    color_continuous_scale='YlGn',  # You can choose your desired color scale
    zmin=0,  # Minimum value for color scale
    zmax=confusion_df.values.max(),  # Maximum value for color scale
    labels=dict(x='Predicted', y='Actual', color='Counts')
)

# Update the layout
fig.update_layout(
    title=' Decision Tree Confusion Matrix',
    title_x=0.5,
    width=400,
    height=300,
)

# Show the figure
fig.show()
```



5.5. Logistic Regression

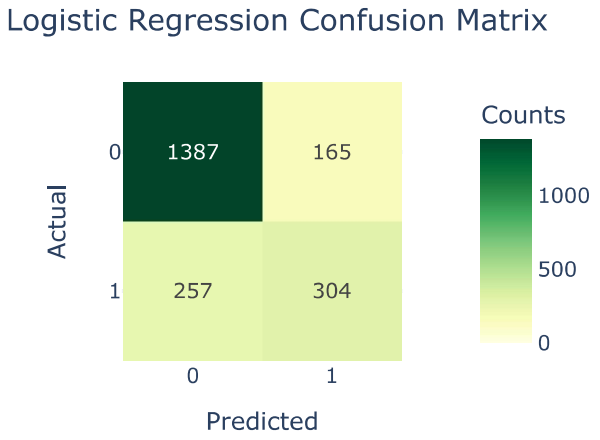
In [74]:

```
logistic_regression_model = LogisticRegression(random_state=42)
logistic_regression_model.fit(X_train, y_train)

# Evaluate model
accuracy_logistic_regression = logistic_regression_model.score(X_test,y_test)
print("Accuracy of Logistic Regression: ", accuracy_logistic_regression)
print("")
# Classification report
logistic_regression_prediction = logistic_regression_model.predict(X_test)
logistic_regression_report = classification_report(y_test, logistic_regression_prediction)

print(logistic_regression_report)
```

```
Accuracy of Logistic Regression:  0.8002839564600095

              precision    recall  f1-score   support

           0       0.84      0.89      0.87      1552
           1       0.65      0.54      0.59       561

    accuracy                           0.80      2113
   macro avg       0.75      0.72      0.73      2113
weighted avg       0.79      0.80      0.79      2113


C:\ProgramData\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: Convergen
ceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/st
able/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://
scikit-learn.org/stable/modules/linear_model.html#logistic-regression)
```

In [75]:

```
# Calculate the confusion matrix
confusion = confusion_matrix(y_test,  logistic_regression_prediction)

# Convert the confusion matrix to a DataFrame for plotting
confusion_df = pd.DataFrame(confusion, columns=['0', '1'], index=['0', '1'])

# Create a heatmap using Plotly Express
fig = px.imshow(
    confusion_df,text_auto=True,
    color_continuous_scale='YlGn',  # You can choose your desired color scale
    zmin=0,  # Minimum value for color scale
    zmax=confusion_df.values.max(),  # Maximum value for color scale
    labels=dict(x='Predicted', y='Actual', color='Counts')
)

# Update the layout
fig.update_layout(
    title=' Logistic Regression Confusion Matrix',
    title_x=0.5,
    width=400,
    height=300,
)

# Show the figure
fig.show()
```

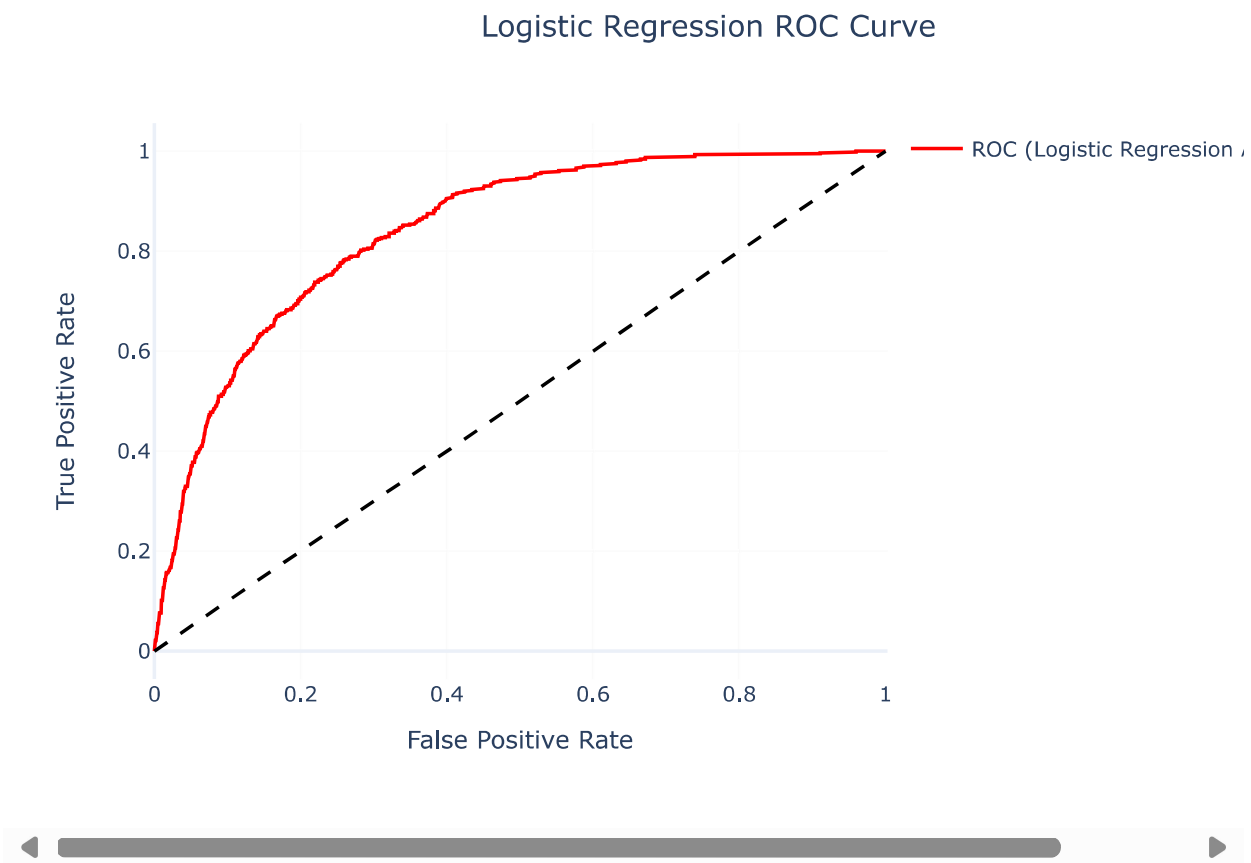


In [76]:

```
y_pred_prob = logistic_regression_model.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Calculate AUC
roc_auc_lr = auc(fpr, tpr)
```

In [77]:

```
# Create a line chart using Plotly
fig = go.Figure()

# Add the random forest ROC curve with a legend title
fig.add_trace(go.Scatter(x=fpr, y=tpr, mode='lines', name='ROC (Logistic Regression AUC = {:.2f})'.format

# Add the diagonal line
fig.add_shape(
    type='line',
    x0=0,
    y0=0,
    x1=1,
    y1=1,
    line=dict(dash='dash', color='black')
)

# Update the layout
fig.update_layout(
    template="plotly_white",
    xaxis_title='False Positive Rate',
    yaxis_title='True Positive Rate',
    title='Logistic Regression ROC Curve',
    title_x=0.5,
    showlegend=True,  # Ensure that the legend is displayed
    width=800,
    height=500,
)

# Show the figure
fig.show()
```



Summary

In [78]:

```
xyz = []

classifiers = ['SVC', 'Random Forest Classifier', 'Logistic Regression', 'kNN', 'Decision Tree']
models = [SVC(random_state=42), RandomForestClassifier(n_estimators=500, random_state=42),
          LogisticRegression(solver='liblinear'), KNeighborsClassifier(n_neighbors=10),
          DecisionTreeClassifier(random_state=42)]

for model in models:
    model.fit(X_train, y_train)
    prediction = model.predict(X_test)
    accuracy = round(metrics.accuracy_score(prediction, y_test),2)
    xyz.append(accuracy)
```

In [79]:

```
models_dataframe = pd.DataFrame(xyz, index=classifiers, columns=['Accuracy'])

models_dataframe=models_dataframe.sort_values(by="Accuracy",ascending=False)

models_dataframe
```

Out[79]:

	Accuracy
Logistic Regression	0.80
SVC	0.79
Random Forest Classifier	0.78
kNN	0.78
Decision Tree	0.72

6. Improve Model Accuracy

6.1. Feature Extraction/ Selection

- Excessive features can have a detrimental impact on algorithm accuracy, especially when these features exhibit correlations or do not align with the fundamental assumptions of the algorithm regarding input data characteristics.
- Feature extraction serves as a strategy for enhancing algorithm accuracy by isolating essential features. This, in turn, not only trims down training time but also mitigates issues related to overfitting.

Two primary methods for feature selection are available:

1. **Correlation Matrix**: This approach focuses on retaining only those features that exhibit minimal correlations with each other.
2. **RandomForestClassifier**: It determines feature importance by assessing their relative significance, aiding in the selection of the most influential features.

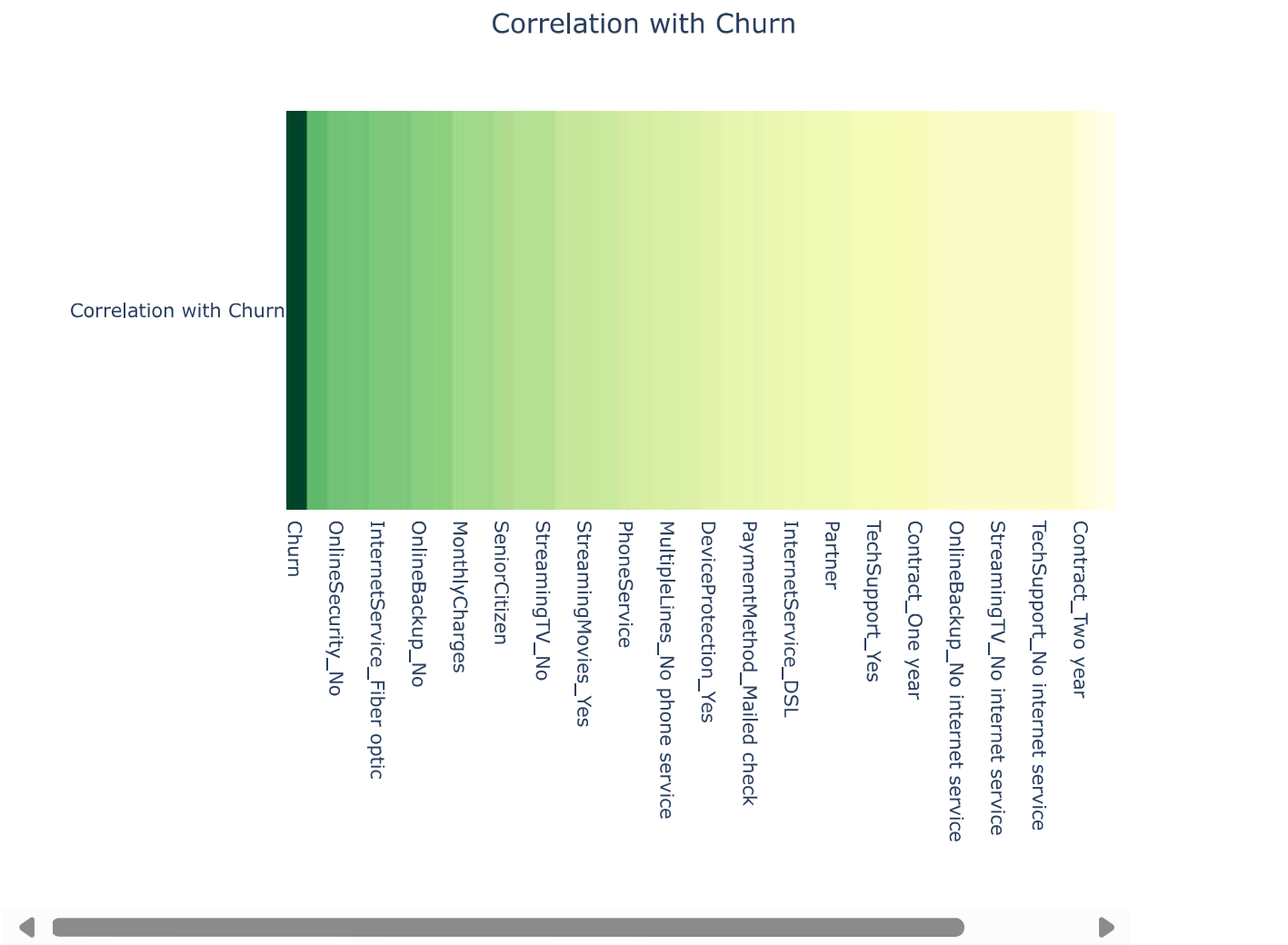
Correlation Matrix

In [80]:

```
corr = dataset.corrwith(df['Churn']).sort_values(ascending = False).to_frame()
corr.columns = ['Correlations']
```

In [81]:

```
# Create a heatmap using Plotly
fig = go.Figure()

# Add the heatmap trace
fig.add_trace(go.Heatmap(
    z=corr['Correlations'].values.reshape(1, -1),
    x=corr.index,
    y=['Correlation with Churn'],
    colorscale='YlGn',  # You can choose your desired color scale
    zmin=corr['Correlations'].min(),
    zmax=corr['Correlations'].max(),
))

# Update the layout
fig.update_layout(
    xaxis_title='',
    yaxis_title='',
    title='Correlation with Churn',
    title_x=0.5,
    width=800,
    height=600,
)

# Show the figure
fig.show()
```



- From the correlation matrix graph above, the features having correlation coefficient ranging from -0.1 to 0.1 will be dropped: StreamingTV_Yes, SteamingMovies_Yes, MultipleLines_Yes, PhoneService, gender, MultipleLines_No, Multiplelines_No phone service, DeviceProtection_Yes, OnlineBackup_Yes, PaymentMethod_Mailed check.

- The remaing features either positively or negatively correlated with Churn.
- Having performed the the vif, Total Charges is also excluded from the set.

6.2. Standardisation

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

In [82]:

```
# List of columns to exclude
columns_to_exclude = ['StreamingTV_Yes', 'StreamingMovies_Yes', 'MultipleLines_Yes', 'PhoneService',
                      'gender','MultipleLines_No', 'MultipleLines_No phone service', 'DeviceProtection_Ye
                      'OnlineBackup_Yes', 'PaymentMethod_Mailed check']
```

In [83]:

```
# Create a new DataFrame with excluded columns
selected_features = dataset.drop(columns=columns_to_exclude)
```

In [84]:

```
# Extract the target variable
telco_target = selected_features['Churn']
```

In [85]:

```
# Extract the feature columns
telco_features = selected_features.drop(columns=['Churn'])
```

In [86]:

```
from sklearn.preprocessing import StandardScaler
```

In [87]:

```
# Standardization
scaler = StandardScaler()
telco_features_standard = scaler.fit_transform(telco_features)
telco_features_standard = pd.DataFrame(telco_features_standard, columns=telco_features.columns)
```

In [88]:

```
# Split the dataset into training and testing sets
X_telco = telco_features_standard
y_telco = telco_target
```

In [89]:

```
X_telco_test, y_telco_train, y_telco_test = train_test_split(X_telco, y_telco, test_size=0.25, random_stat
```

In [90]:

```
abc = []
classifiers = ['SVC','Random Forest Classifier','Logistic Regression','kNN','Decision Tree']
models = [SVC(random_state=42), RandomForestClassifier(n_estimators=500, random_state=42),
          LogisticRegression(solver='liblinear'), KNeighborsClassifier(n_neighbors=10),
          DecisionTreeClassifier(random_state=42)]

for i in models:
    model = i
    model.fit(X_telco_train,y_telco_train)
    prediction=model.predict(X_telco_test)
    abc.append(metrics.accuracy_score(prediction,y_telco_test))

new_models_dataframe=pd.DataFrame(abc,index=classifiers)
new_models_dataframe.columns=['New Accuracy']
```

In [91]:

```
new_models_dataframe=new_models_dataframe.merge(models_dataframe,left_index=True,right_index=True,how='le
new_models_dataframe['Increase']=new_models_dataframe['New Accuracy']-new_models_dataframe['Accuracy']

new_models_dataframe=new_models_dataframe.sort_values(by="New Accuracy",ascending=False)

new_models_dataframe
```

Out[91]:

	New Accuracy	Accuracy	Increase
Logistic Regression	0.796706	0.80	-0.003294
SVC	0.792164	0.79	0.002164
Random Forest Classifier	0.778535	0.78	-0.001465
kNN	0.778535	0.78	-0.001465
Decision Tree	0.746735	0.72	0.026735

The dataframe illustrates the accuracy of the models after applying feature selection. We observe that the accuracy for **SVM** slightly increases by nearly 0.087%, the accuracy rate of **Random Forest Classifier** is improved by 0.097%. For the **kNN** and **Logistic Regression** model, the accuracy rate unfortunately decreases 0.23% and 0.36% respectively. Lastly, **Decision Tree** model generates result more accurate by 2.3%.

6.3. Cross Validation

In [92]:

```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
```

In [93]:

```
# k=5, split the data into 5 equal parts

kfold = KFold(n_splits=5, shuffle=True, random_state=22)
```

In [94]:

```
xyz = []
accuracy = []
classifier_names = [
    ('kNN', KNeighborsClassifier()),
    ('SVM', SVC()),
    ('Random Forest', RandomForestClassifier(n_estimators=500, random_state=42)),
    ('Decision Tree', DecisionTreeClassifier(random_state=42)),
    ('Logistic Regression', LogisticRegression(solver='liblinear'))
]

for name, model in classifier_names:
    cv_result = cross_val_score(model, X_telco, y_telco, cv=kfold, scoring="accuracy")
    xyz.append(round(cv_result.mean(),2))
    accuracy.append(cv_result)
```

In [95]:

```
new_models_df = pd.DataFrame(xyz, index=[name for name, _ in classifier_names], columns=['CV Mean'])

new_models_df=new_models_df.sort_values(by="CV Mean",ascending=False)
new_models_df
```

Out[95]:

	CV Mean
Logistic Regression	0.80
SVM	0.79
Random Forest	0.78
kNN	0.76
Decision Tree	0.73

6.4. Ensembling

In [96]:

```
SVM = SVC(random_state = 42, C = 0.1, probability = True)
LR = LogisticRegression(C = 0.1)
kNN = KNeighborsClassifier(n_neighbors=8)
RF = RandomForestClassifier(random_state=42, n_estimators=500)
DT = DecisionTreeClassifier(random_state=42)
```

In [97]:

```
# Set probability=True for all models to enable probability estimates
kNN.probability = True
RF.probability = True
DT.probability = True
```

In [98]:

```
#for Voting Classifier

from sklearn.ensemble import VotingClassifier
```

SVM with LR

In [99]:

```
ens_results = []

ensemble_SVM_LR = VotingClassifier(estimators=[('SVC', SVM), ('LR', LR)],
                        voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_SVM_LR.score(X_telco_test, y_telco_test))

print('The accuracy for SVM and LR is:',ensemble_SVM_LR.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and LR is: 0.7881885292447472
```

SVM with kNN

In [100]:

```
ensemble_SVM_kNN = VotingClassifier(estimators=[('SVM', SVM), ('kNN', kNN)],
                        voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_SVM_kNN.score(X_telco_test, y_telco_test))

print('The accuracy for SVM and kNN is:', ensemble_SVM_kNN.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and kNN is: 0.7932992617830777
```

SVM with RF

In [101]:

```
ensemble_SVM_RF = VotingClassifier(estimators=[('SVM', SVM), ('RF', RF)],
                        voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_SVM_RF.score(X_telco_test, y_telco_test))

print('The accuracy for SVM and RF is:',ensemble_SVM_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and RF is: 0.7961385576377058
```

SVM with DT

In [102]:

```
ensemble_SVM_DT = VotingClassifier(estimators=[('SVM', SVM), ('DT', DT)],
                        voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_SVM_DT.score(X_telco_test, y_telco_test))

print('The accuracy for SVM and DT is:',ensemble_SVM_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for SVM and DT is: 0.7921635434412265
```

LR with kNN

In [103]:

```
ensemble_LR_kNN = VotingClassifier(estimators=[('LR', LR), ('kNN', kNN)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_LR_kNN.score(X_telco_test, y_telco_test))

print('The accuracy for LR and kNN is:', ensemble_LR_kNN.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and kNN is: 0.7932992617830777
```

LR with RF

In [104]:

```
ensemble_LR_RF = VotingClassifier(estimators=[('LR', LR), ('RF', RF)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)

ens_results.append(ensemble_LR_RF.score(X_telco_test, y_telco_test))

print('The accuracy for LR and RF is:',ensemble_LR_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and RF is: 0.7989778534923339
```

LR with DT

In [105]:

```
ensemble_LR_DT = VotingClassifier(estimators=[('LR', LR), ('RF', DT)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_LR_DT.score(X_telco_test, y_telco_test))
print('The accuracy for LR and DT is:',ensemble_LR_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for LR and DT is: 0.7796706416808632
```

kNN with RF

In [106]:

```
ensemble_kNN_RF = VotingClassifier(estimators=[('kNN', kNN), ('RF', RF)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_kNN_RF.score(X_telco_test, y_telco_test))
print('The accuracy for RF and kNN is:',ensemble_kNN_RF.score(X_telco_test, y_telco_test))
```

```
The accuracy for RF and kNN is: 0.78137421919364
```

kNN with DT

In [107]:

```
ensemble_kNN_DT = VotingClassifier(estimators=[('kNN', kNN), ('DT', DT)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_kNN_DT.score(X_telco_test, y_telco_test))
print('The accuracy for kNN and DT is:',ensemble_kNN_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for kNN and DT is: 0.7853492333901193
```

RF with DT

In [108]:

```
ensemble_RF_DT = VotingClassifier(estimators=[('RF', RF), ('DT', DT)],
                       voting='soft', weights=[2, 1]).fit(X_telco_train, y_telco_train)
ens_results.append(ensemble_RF_DT.score(X_telco_test, y_telco_test))
print('The accuracy for RF and DT is:',ensemble_RF_DT.score(X_telco_test, y_telco_test))
```

```
The accuracy for RF and DT is: 0.7768313458262351
```

All 5 classifiers combined

In [109]:

```
ensembled=VotingClassifier(estimators=[('SVM', SVM), ('LR', LR),('RF', RF), ('DT', DT), ('kNN', kNN) ],
                       voting='soft', weights=[2,1,3, 4, 5]).fit(X_telco_train,y_telco_train)
ens_results.append(round(ensembled.score(X_telco_test,y_telco_test),4))
print('The ensembled model with all the 5 classifiers is:',round(ensembled.score(X_telco_test, y_telco_te
```

```
The ensembled model with all the 5 classifiers is: 0.7865
```

- So the maximum Accuracy which we could get by using ensemble models is 79% .
- Below we can see the results of all our ensemble models together.

In [110]:

```
print(ens_results)
```

```
[0.7881885292447472, 0.7932992617830777, 0.7961385576377058, 0.7921635434412265, 0.79329926
17830777, 0.7989778534923339, 0.7796706416808632, 0.78137421919364, 0.7853492333901193, 0.7
768313458262351, 0.7865]
```

In [111]:

```
classifiers_set = ['SVM with LR', 'SVM with kNN', 'SVM with DT', 'SVM with RF',
                   'LR with DT', 'LR with kNN', 'LR with RF',
                   'DT with RF', 'DT with kNN',
                   'RF with kNN', 'All 5 classifiers combined']
models_combined = pd.DataFrame(ens_results, index = classifiers_set)
models_combined.columns = ['Accuracy']
```

In [112]:

```
# Find the row with the highest accuracy
max_accuracy_row = models_combined['Accuracy'].idxmax()
```

In [113]:

```
# Create a copy of the DataFrame with the highest value highlighted
highlighted_models_combined = models_combined.style.apply(
    lambda x: ['background-color: yellow' if x.name == max_accuracy_row else '' for i in x], axis=1)

# Display the highlighted DataFrame
highlighted_models_combined
```

Out[113]:

	Accuracy
SVM with LR	0.788189
SVM with kNN	0.793299
SVM with DT	0.796139
SVM with RF	0.792164
LR with DT	0.793299
LR with kNN	0.798978
LR with RF	0.779671
DT with RF	0.781374
DT with kNN	0.785349
RF with kNN	0.776831
All 5 classifiers combined	0.786500

Ensemble Model Performance:

Combining all five classifiers resulted in an ensemble accuracy of approximately 0.79. This ensemble approach, which combines the predictions of multiple models, provides a balanced accuracy level.