PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100 Feet Ring Road, BSK III Stage, Bengaluru-560 085

**Department of Electronics and Communication Engineering**

**Course Title: Machine Learning and Applications**
**Course Code: UE21EC352B**

**Teacher: Dr. Shruthi M. L. J.**

# Project Title:
# Predicting Customer Churn using Unsupervised Model to Pre-Cluster Supervised Model Inputs

**Done By: Team 13**
**Semester: 06 Section: A**


**Anna Singh - PES1UG21EC050**
**Anushka Anjali- PES1UG21EC053**
**Archanaa A Chandaragi - PES1UG21EC056**

**Table of Contents:**

**Abstract:**

The primary key for running any successful business is its ability to maintain customer loyalty and prevent revenue loss due to churning of customers. Being able to predict which customers are at risk of being churned is an excellent strategy that helps businesses stay afloat by retaining customers and offering them incentives to do so. Using Machine Learning models to predict this data and preprocessing it to compute the values of churn as closely as possible makes the computations easier. In this paper, we shall see a novel approach to predict customer churn using unsupervised models to pre-cluster supervised model inputs.

**Introduction:**

A strong customer base is an important requirement of running a successful business as it shows a lot about the quality of business and its nature of fostering relationships with its customers.
It is easier to maintain relations and retain existing customers, rather than drawing in newer customers. This is why understanding the rate of customer churn and its impacts is very important in the functioning of any business.

**Problem Statement:**

In this project we will attempt to perform pre-clustering of the given dataset by an unsupervised model. Then we will supply that as an input to the supervised model which predicts customer churn.

**Relevance:**

- Improving the accuracy of customer churn forecasts by a hundredth of a percent can save a company a significant amount of money.
- A prior study found that the cost of obtaining new consumers is often five times that of retaining existing ones.
- By using an unsupervised model cluster prediction can be implemented at a much earlier state which is a huge advantage in terms of time taken to train the model.

## Implementation: Code and output screenshots

This code has been written on Google Collab with the segmentation and results as follows:

```
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.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from xgboost import XGBClassifier
from sklearn import metrics
from sklearn.metrics import roc_curve
from sklearn.metrics import recall_score, confusion_matrix,
precision_score, f1_score, accuracy_score, classification_report
```

We start the code by importing the following library functions:

- *Matplotlib.pyplot* is a collection of command style functions that make matplotlib work like MATLAB.
- *NumPy* is a python library that is used when working with and creating, manipulating and analyzing arrays. It also has functions for working in the domain of linear algebra, fourier transform, and matrices.
- The *sklearn* library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
- The *sklearn.preprocessing* package provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators.
- *LabelEncoder* can be used to normalize labels. It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels. We shall see this in the later stages of the code.
- *One-hot encoding* is a technique in machine learning that turns categorical data, like colors (red, green, blue), into numerical data for machines to understand. It creates new binary columns for each category, with a 1 marking the presence of that category and 0 elsewhere.
- *Pandas* allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant.

```
#loading data
df = pd.read_csv('/content/WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

```
df.shape
```

```
(7043, 21)
```

*pd.read* allows us to read a data (csv) file as a pandas DataFrame. *df.shape* provides the number of rows and columns in the data frame.

**df.head()**

| | customerID | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | ... | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBilling | PaymentMethod | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | ... | No | No | No | No | Month-to-month | Yes | Electronic check | 29.85 | 29.85 | No |
| 1 | 5575-GNVDE | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | ... | Yes | No | No | No | One year | No | Mailed check | 56.95 | 1889.5 | No |
| 2 | 3668-QPYBK | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | ... | No | No | No | No | Month-to-month | Yes | Mailed check | 53.85 | 108.15 | Yes |
| 3 | 7795-CFOCW | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | ... | Yes | Yes | No | No | One year | No | Bank transfer (automatic) | 42.30 | 1840.75 | No |
| 4 | 9237-HQITU | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | ... | No | No | No | No | Month-to-month | Yes | Electronic check | 70.70 | 151.65 | Yes |

5 rows × 21 columns

*df.head* returns the first n (in this case, 5 rows) rows based on position.

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 information about customers** – gender, age range, and if they have partners and dependents

```
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.columns* provides access to the column labels of the data frame. It returns an index object representing the names of the columns in the dataframe.
The *.values()* method returns all of its values of a Python dictionary in a view object that will reflect any changes to the dictionary values. It takes no arguments.

```
df.describe()
```

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

*df.describe* is used for calculating some statistical data like percentile, mean, std, min and max of the numerical values of the Series or DataFrame. It analyzes both numeric and object series and also the DataFrame column sets of mixed data types.

We use the **Churn** target to guide the exploration.

```
df.dtypes

df.columns.to_series().groupby(df.dtypes).groups
```

```
{int64: ['SeniorCitizen', 'tenure'], float64: ['MonthlyCharges'], object: ['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod', 'TotalCharges', 'Churn']}
```

*df.dtypes* returns a series with the datatypes of each column, depending on the type of data it has.

*df.columns.to_series.groupby(df.dtypes).groups* is used in the following manner - convert the DataFrame columns into a Series using *df. columns. to_series()* and then use the *groupby()* function along with *df. dtypes* to group columns by their data types. Therefore in the output, we see the names of columns, i.e., labels in the bracket beside each of the data types.

```
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.info()* prints information about the dataframe, and contains information like the number of columns, column labels, range index, data types, etc.

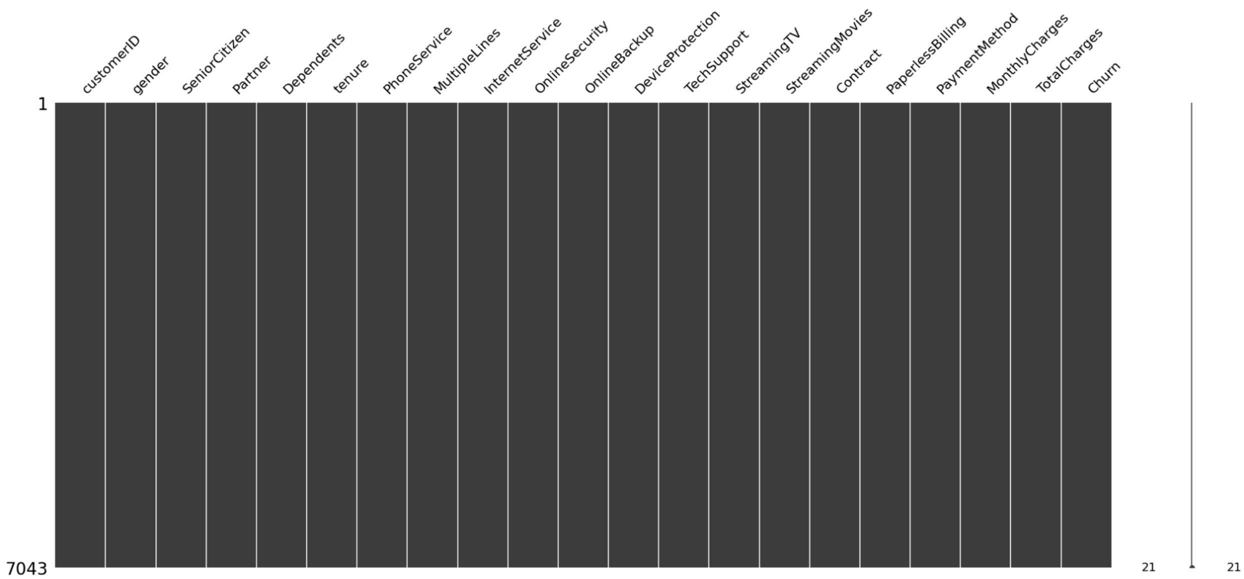
```
df.isna().any()
```

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

*df.isna().any()* method checks whether the objects of a Dataframe or a Series contain missing or null values (NA, NaN) and returns a new object with the same shape as the original but with boolean values True or False as the elements. The *any()* method returns one value for each column, True if ANY value in that column is True, otherwise False.

```
# Visualize missing values as a matrix
msno.matrix(df);
```

The *msno.matrix()* nullity matrix is a data-dense display which lets you quickly visually pick out patterns in data completion. The sparkline on the right summarizes the general shape of the data completeness and points out the rows with the maximum and minimum nullity in the dataset. There are no missing values as we can see in the image below.




After this, we now move to Data Manipulation in the dataset and drop the 'Customer ID' Field.

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

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBilling | PaymentMethod | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | Yes | No | No | No | No | Month-to-month | Yes | Electronic check | 29.85 | 29.85 | No |
| 1 | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | No | Yes | No | No | No | One year | No | Mailed check | 56.95 | 1889.5 | No |
| 2 | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | Yes | No | No | No | No | Month-to-month | Yes | Mailed check | 53.85 | 108.15 | Yes |
| 3 | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | No | Yes | Yes | No | No | One year | No | Bank transfer (automatic) | 42.30 | 1840.75 | No |
| 4 | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | No | No | No | No | No | Month-to-month | Yes | Electronic check | 70.70 | 151.65 | Yes |

```
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
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        11
Churn                0
dtype: int64
```

*pd.to_numeric()* is used to convert given arguments into numeric types. If the error argument is passed as raise , then invalid parsing will raise an exception. If the error argument is passed as *coerce* , then invalid parsing will be set as NaN . If the error argument is passed as 'ignore' , then invalid parsing will return the input.

The function df. isnull(). sum() returns the number of missing values in the dataset, which in this case we see that TotalCharges has 11 missing values.

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

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBilling | PaymentMethod | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 488 | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | No | Yes | Yes | Yes | No | Two year | Yes | Bank transfer (automatic) | 52.55 | NaN | No |
| 753 | Male | 0 | No | Yes | 0 | Yes | No | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | Two year | No | Mailed check | 20.25 | NaN | No |
| 936 | Female | 0 | Yes | Yes | 0 | Yes | No | DSL | Yes | Yes | Yes | No | Yes | Yes | Two year | No | Mailed check | 80.85 | NaN | No |
| 1082 | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | Two year | No | Mailed check | 25.75 | NaN | No |
| 1340 | Female | 0 | Yes | Yes | 0 | No | No phone service | DSL | Yes | Yes | Yes | Yes | Yes | No | Two year | No | Credit card (automatic) | 56.05 | NaN | No |
| 3331 | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | Two year | No | Mailed check | 19.85 | NaN | No |
| 3826 | Male | 0 | Yes | Yes | 0 | Yes | Yes | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | Two year | No | Mailed check | 25.35 | NaN | No |
| 4380 | Female | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | Two year | No | Mailed check | 20.00 | NaN | No |
| 5218 | Male | 0 | Yes | Yes | 0 | Yes | No | No | No internet service | No internet service | No internet service | No internet service | No internet service | No internet service | One year | Yes | Mailed check | 19.70 | NaN | No |
| 6670 | Female | 0 | Yes | Yes | 0 | Yes | Yes | DSL | No | Yes | Yes | Yes | Yes | No | Two year | No | Mailed check | 73.35 | NaN | No |
| 6754 | Male | 0 | No | Yes | 0 | Yes | Yes | DSL | Yes | Yes | No | Yes | No | No | Two year | Yes | Bank transfer (automatic) | 61.90 | NaN | No |

The Tenure column is 0 for these entries even though the MonthlyCharges column is not empty. Checking if there are any other 0 values in the tenure column.

In NumPy, you can use the *isnan()* function to check for NaN values in an array. This function returns a Boolean array indicating which values in the input array are NaN. You can also use the *nan_to_num()* function to replace NaN values with a specified value, such as zero.

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

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

There are no additional missing values in the Tenure column. Delete the rows with missing values in Tenure columns since there are only 11 rows and deleting them will not affect the data.

```
df.drop(labels=df[df['tenure'] == 0].index, axis=0, inplace=True)
df[df['tenure'] == 0].index
```

```
Index([], dtype='int64')
```

To solve the problem of missing values in the TotalCharges column, imputation was done.

```
df.fillna(df["TotalCharges"].mean())
```

The *fillna()* method replaces the NULL values with a specified value. The fillna() method returns a new DataFrame object unless the in place parameter is set to True , in that case the fillna() method does the replacing in the original DataFrame instead. Here we are imputing the mean value of the TotalCharges column, as demonstrated by *df.fillna(df.mean())*.

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBilling | PaymentMethod | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | Female | 0 | Yes | No | 1 | No | No phone service | DSL | No | Yes | No | No | No | No | Month-to-month | Yes | Electronic check | 29.85 | 29.85 | No |
| 1 | Male | 0 | No | No | 34 | Yes | No | DSL | Yes | No | Yes | No | No | No | One year | No | Mailed check | 56.95 | 1889.50 | No |
| 2 | Male | 0 | No | No | 2 | Yes | No | DSL | Yes | Yes | No | No | No | No | Month-to-month | Yes | Mailed check | 53.85 | 108.15 | Yes |
| 3 | Male | 0 | No | No | 45 | No | No phone service | DSL | Yes | No | Yes | Yes | No | No | One year | No | Bank transfer (automatic) | 42.30 | 1840.75 | No |
| 4 | Female | 0 | No | No | 2 | Yes | No | Fiber optic | No | No | No | No | No | No | Month-to-month | Yes | Electronic check | 70.70 | 151.65 | Yes |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 7038 | Male | 0 | Yes | Yes | 24 | Yes | Yes | DSL | Yes | No | Yes | Yes | Yes | Yes | One year | Yes | Mailed check | 84.80 | 1990.50 | No |
| 7039 | Female | 0 | Yes | Yes | 72 | Yes | Yes | Fiber optic | No | Yes | Yes | No | Yes | Yes | One year | Yes | Credit card (automatic) | 103.20 | 7362.90 | No |
| 7040 | Female | 0 | Yes | Yes | 11 | No | No phone service | DSL | Yes | No | No | No | No | No | Month-to-month | Yes | Electronic check | 29.60 | 346.45 | No |
| 7041 | Male | 1 | Yes | No | 4 | Yes | Yes | Fiber optic | No | No | No | No | No | No | Month-to-month | Yes | Mailed check | 74.40 | 306.60 | Yes |
| 7042 | Male | 0 | No | No | 66 | Yes | No | Fiber optic | Yes | No | Yes | Yes | Yes | Yes | Two year | Yes | Bank transfer (automatic) | 105.65 | 6844.50 | No |

7032 rows × 20 columns

```
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
```

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

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

*df.describe()* describes the feature InternetService and gives values such as count, unique, top, freq, name, dtype, etc.

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

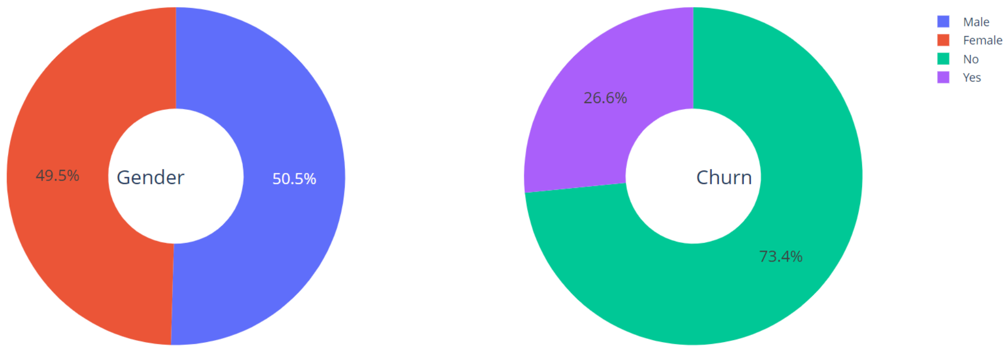
| | tenure | MonthlyCharges | TotalCharges |
|---|---|---|---|
| **count** | 7032.000000 | 7032.000000 | 7032.000000 |
| **mean** | 32.421786 | 64.798208 | 2283.300441 |
| **std** | 24.545260 | 30.085974 | 2266.771362 |
| **min** | 1.000000 | 18.250000 | 18.800000 |
| **25%** | 9.000000 | 35.587500 |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

Here we have described the count, min, max, std values of the given columns.

After the data manipulation step, we perform 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()
```

Gender and Churn Distributions




- 26.6 % of customers switched to another firm.
- Customers are 49.5 % female and 50.5 % male

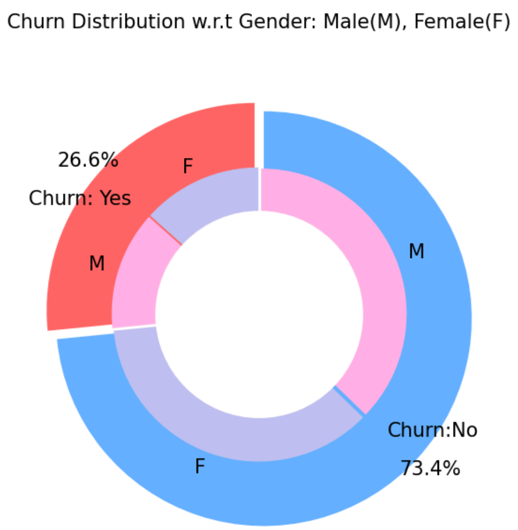
```
df["Churn"][df["Churn"]=="No"].groupby(by=df["gender"]).count()
```

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

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

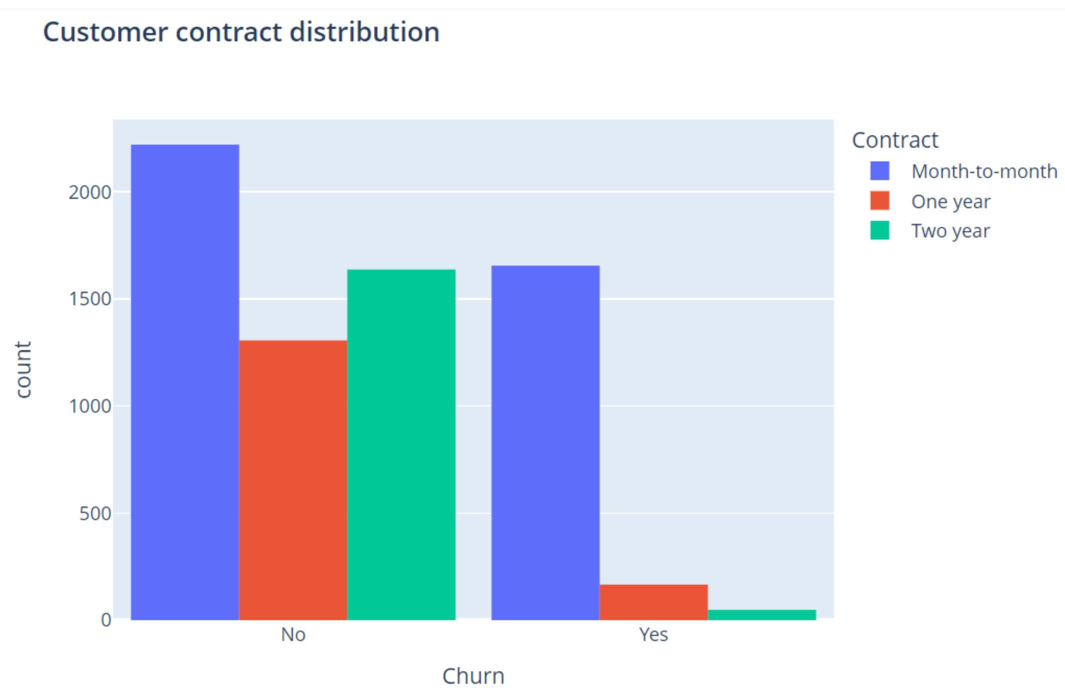
```
gender
Female    939
Male      930
Name: Churn, dtype: int64
```

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



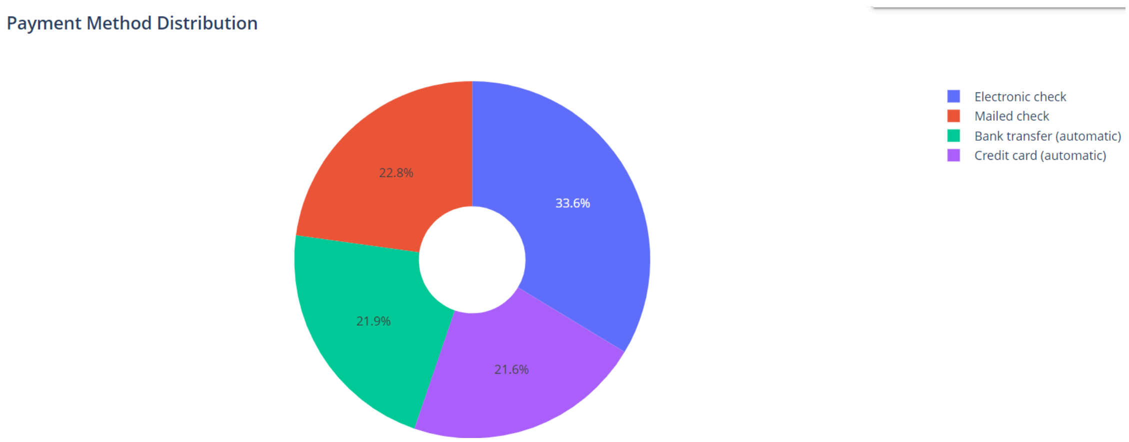

- There is negligible difference in customer percentage/ count who changed the service provider. Both genders behaved in similar fashion when it came to migrating to another service provider/firm.

```
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()
```

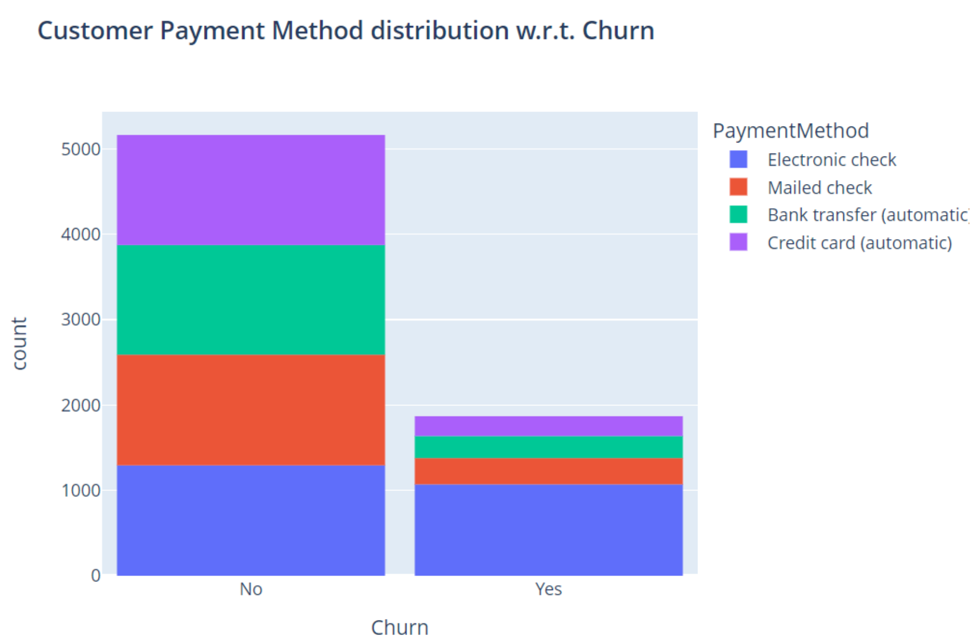


About 75% of customer with Month-to-Month Contract opted to move out as compared to 13% of customers with One Year Contract and 3% with Two Year Contract

```
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()
```




```
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()
```



- 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)
```

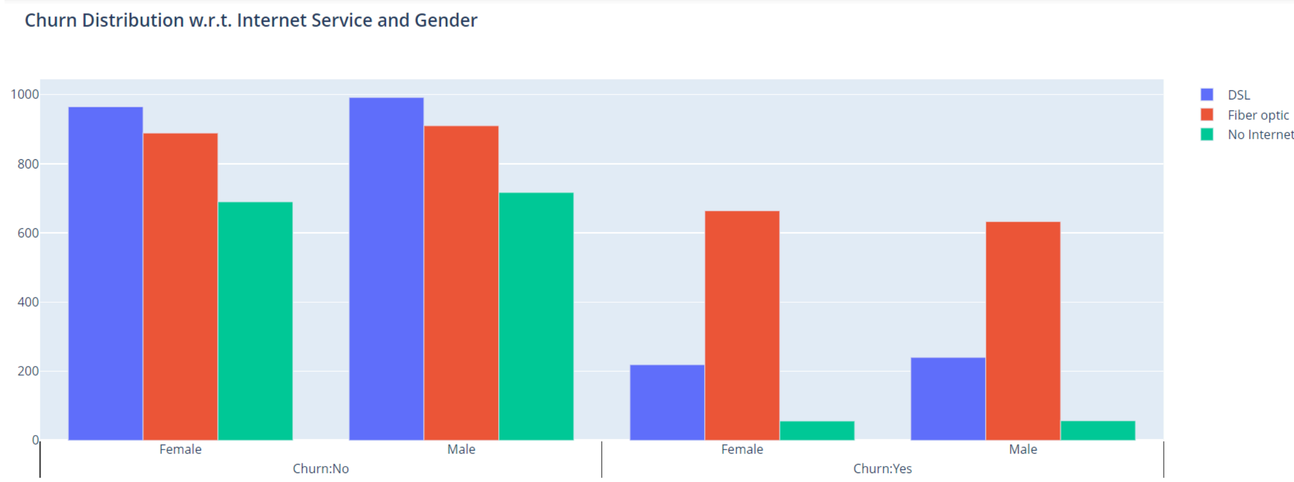
```
df[df["gender"]=="Male"][["InternetService", "Churn"]].value_counts()
```

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

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

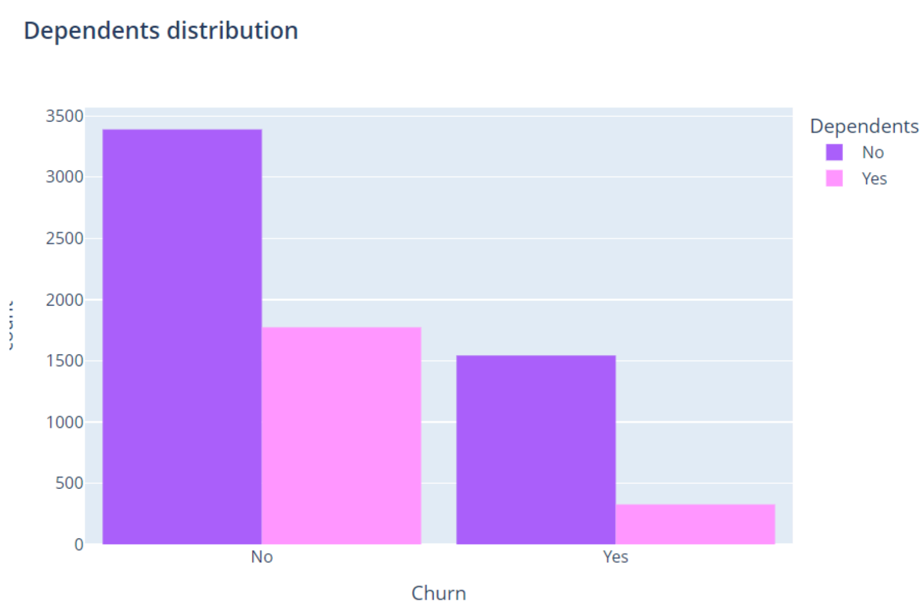
```
InternetService  Churn
DSL              No       965
Fiber optic      No       889
No               No       690
Fiber optic      Yes      664
DSL              Yes      219
No               Yes       56
Name: count, dtype: int64
```

```
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()
```



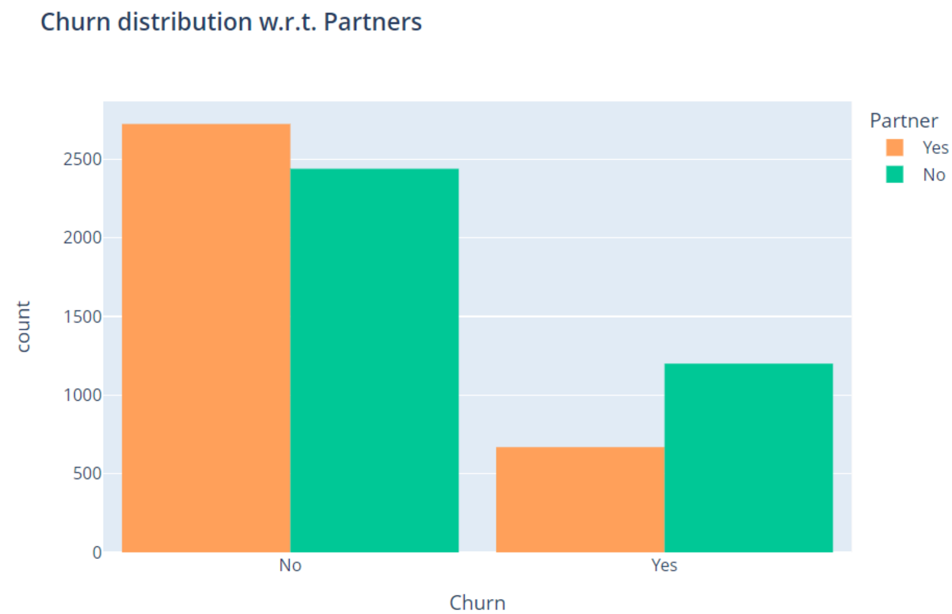

- A lot of customers choose the Fiber optic service and it's also evident that the customers who use Fiber optic have high churn rate, this might suggest a dissatisfaction with this type of internet service.
- Customers having DSL service are majority in number and have less churn rate compared to Fibre optic service

```
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()
```



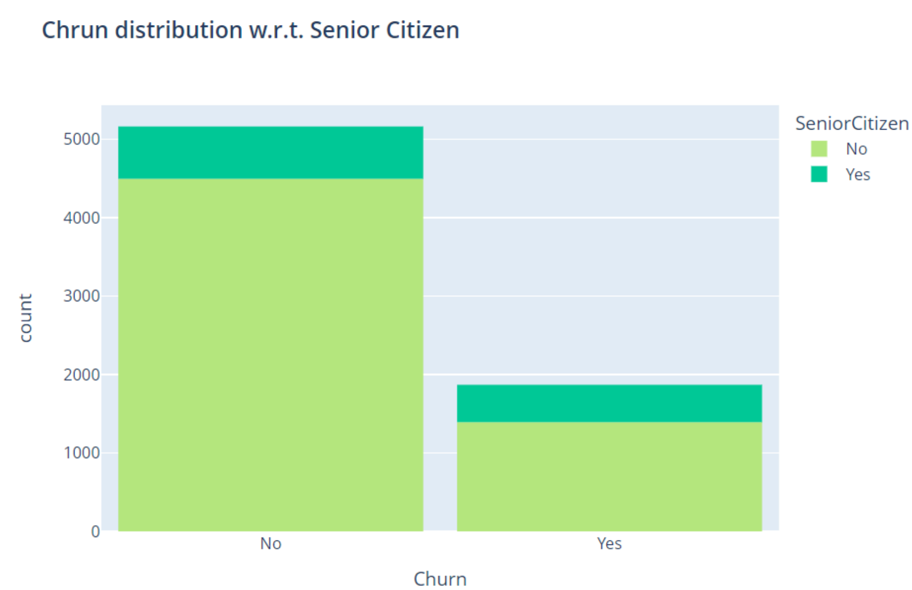

This shows that the customers without dependents are more likely to churn.

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



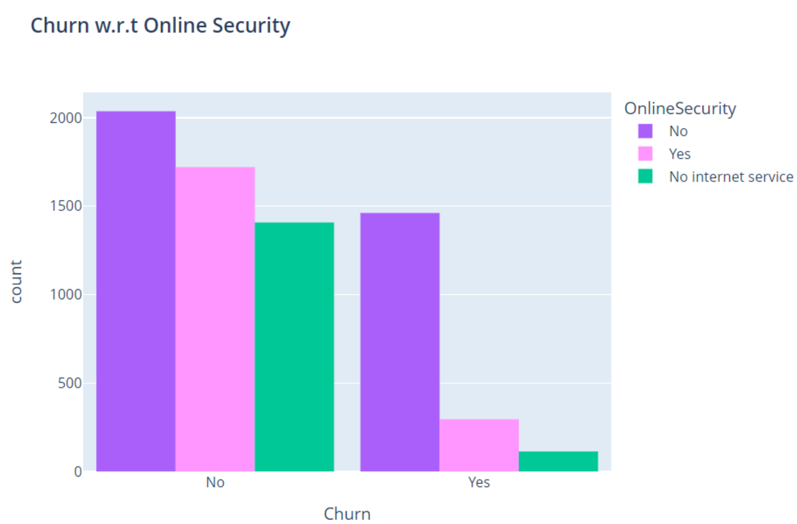

This shows that customers who don’t have partners are more likely to churn.

```python
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()
```



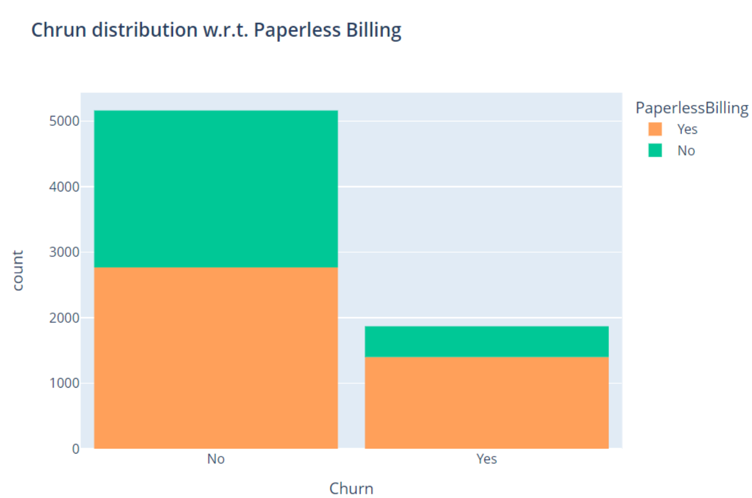
- It can be observed that the fraction of senior citizens retention is very less.
- Most of the senior citizens churn.

```
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()
```



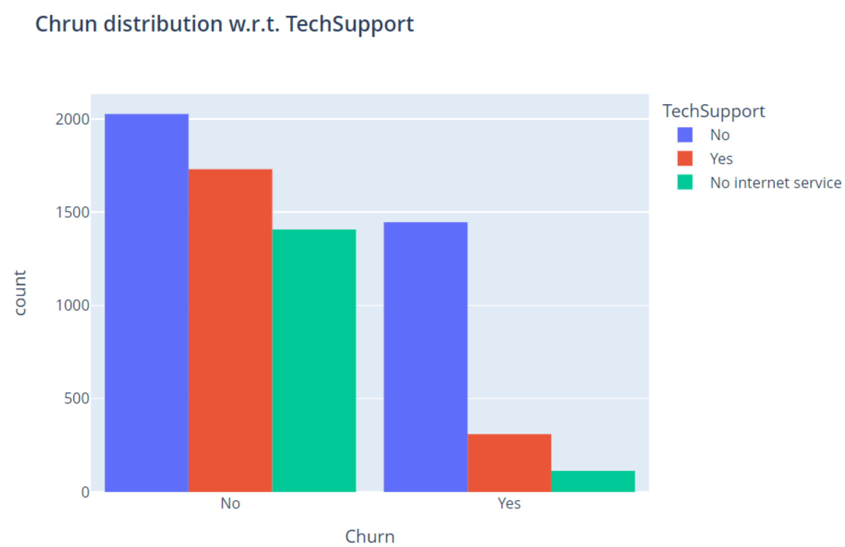

Most customers churn in the absence of online security.

```
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()
```



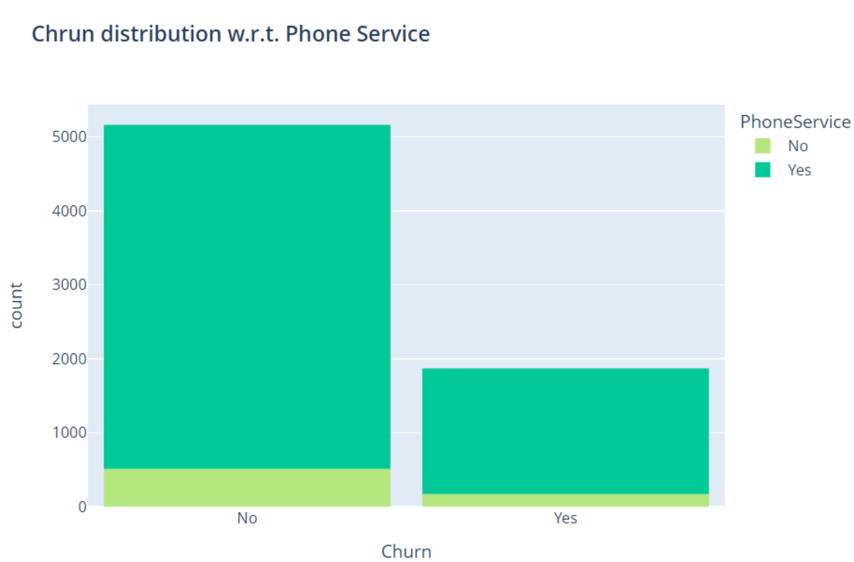
Customers with Paperless Billing are most likely to churn.

```
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()
```



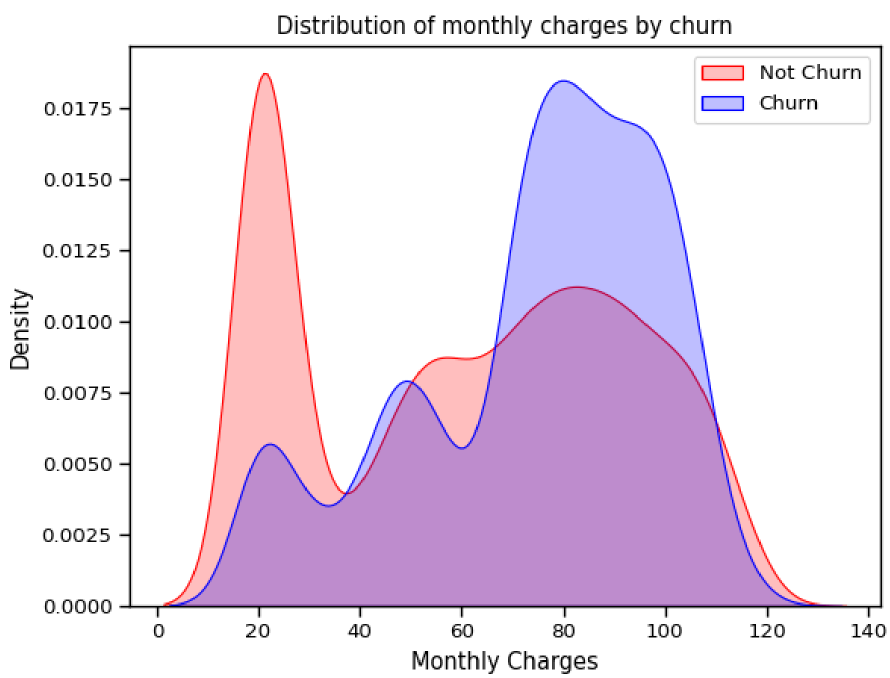

Customers with no TechSupport are most likely to migrate to another service provider.

```
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()
```



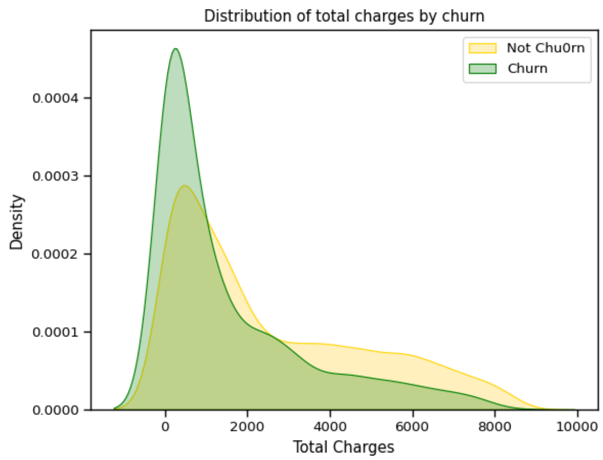
Very small fraction of customers don't have a phone service and out of that, 1/3rd Customers are more likely to churn.

```
sns.set_context("paper",font_scale=1.1)
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'No') ],
                color="Red", shade = True);
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'Yes') ],
                ax =ax, color="Blue", shade= True);
ax.legend(["Not Churn","Churn"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Monthly Charges');
ax.set_title('Distribution of monthly charges by churn');
```



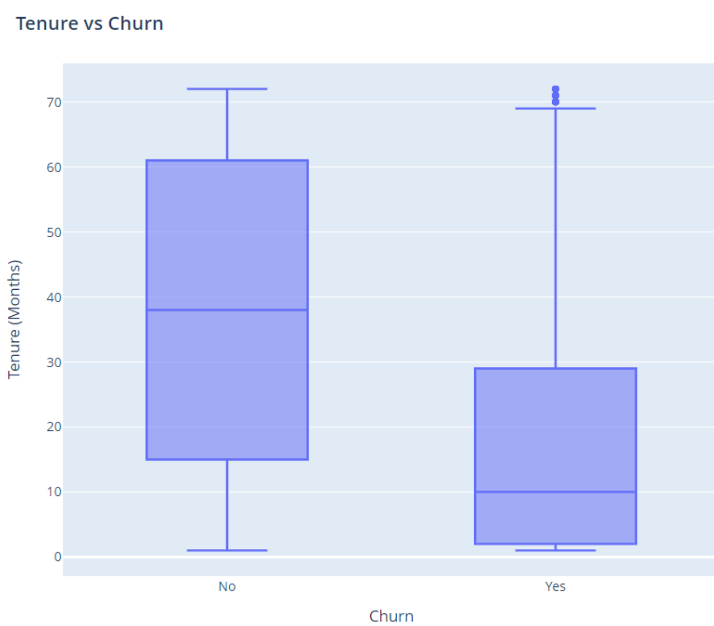

Customers with higher Monthly Charges are also more likely to churn.

```
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'No') ],
                color="Gold", shade = True);
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'Yes') ],
                ax =ax, color="Green", shade= True);
ax.legend(["Not Chu0rn","Churn"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Total Charges');
ax.set_title('Distribution of total charges by churn');
```

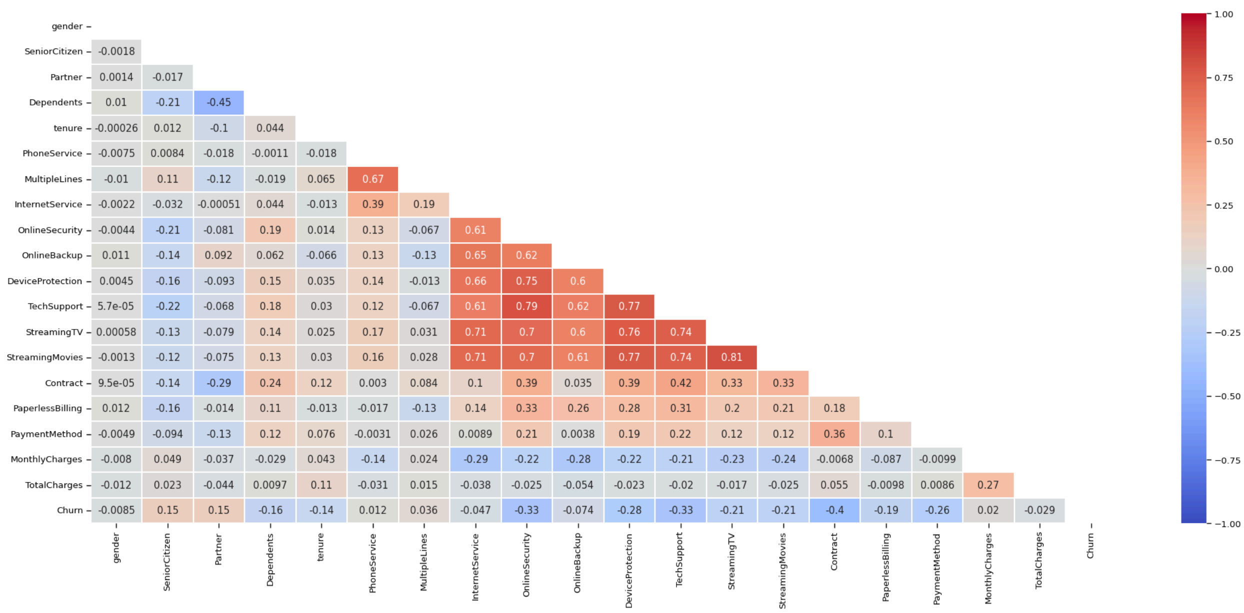



Customers who have stayed with the business for a longer time tend to stay with the business as opposed to newly joined customers. New customers are more likely to get churned.

```
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()
```



```python
plt.figure(figsize=(25, 10))

corr = df.apply(lambda x: pd.factorize(x)[0]).corr()

mask = np.triu(np.ones_like(corr, dtype=bool))

ax = sns.heatmap(corr, mask=mask, xticklabels=corr.columns, yticklabels=corr.columns,
                 annot=True, linewidths=.2, cmap='coolwarm', vmin=-1, vmax=1)
```




After the data visualization process, we now move on to the data preprocessing part. We first perform Label Encoding for normalizing the labels. Since the numerical features are distributed over different value ranges,standard scalar is used to scale them down to the same range (Normalization).

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

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

| | gender | SeniorCitizen | Partner | Dependents | tenure | PhoneService | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | DeviceProtection | TechSupport | StreamingTV | StreamingMovies | Contract | PaperlessBilling | PaymentMethod | MonthlyCharges | TotalCharges | Churn |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 29.85 | 29.85 | 0 |
| 1 | 1 | 0 | 0 | 0 | 34 | 1 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 3 | 56.95 | 1889.50 | 0 |
| 2 | 1 | 0 | 0 | 0 | 2 | 1 | 0 | 0 | 2 | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 3 | 53.85 | 108.15 | 1 |
| 3 | 1 | 0 | 0 | 0 | 45 | 0 | 1 | 0 | 2 | 0 | 2 | 2 | 0 | 0 | 1 | 0 | 0 | 42.30 | 1840.75 | 0 |
| 4 | 0 | 0 | 0 | 0 | 2 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 2 | 70.70 | 151.65 | 1 |

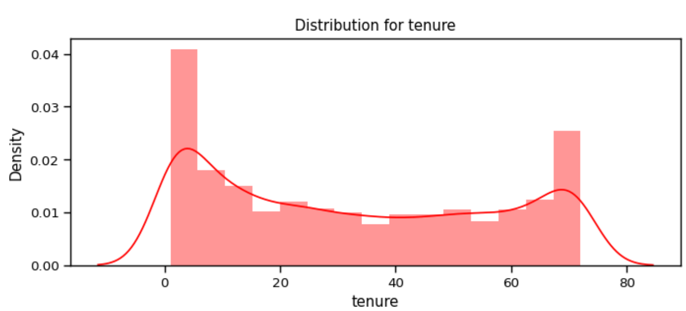
```
plt.figure(figsize=(14,7))
df.corr()['Churn'].sort_values(ascending = False)
```

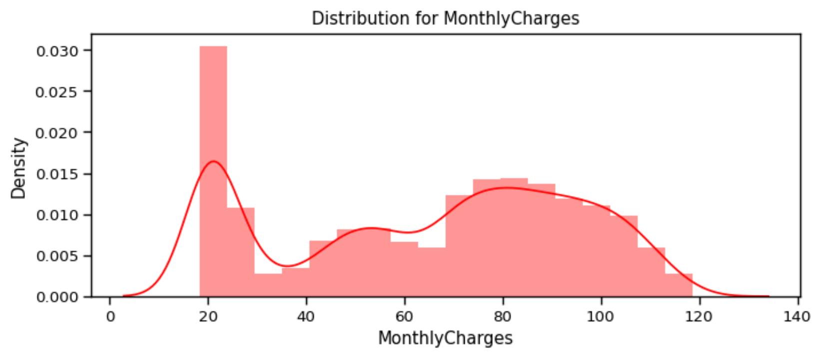
```
Churn               1.000000
MonthlyCharges      0.192858
PaperlessBilling    0.191454
SeniorCitizen       0.150541
PaymentMethod       0.107852
MultipleLines       0.038043
PhoneService        0.011691
gender             -0.008545
StreamingTV        -0.036303
StreamingMovies    -0.038802
InternetService    -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
<Figure size 1400x700 with 0 Axes>
```

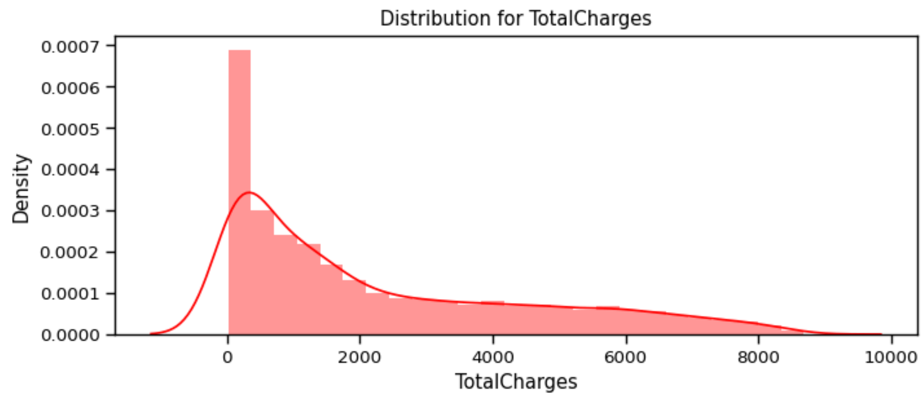
```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values
```

```
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,3))
    plt.title("Distribution for {}".format(feature))
    ax = sns.distplot(frame[feature], color= color)
```

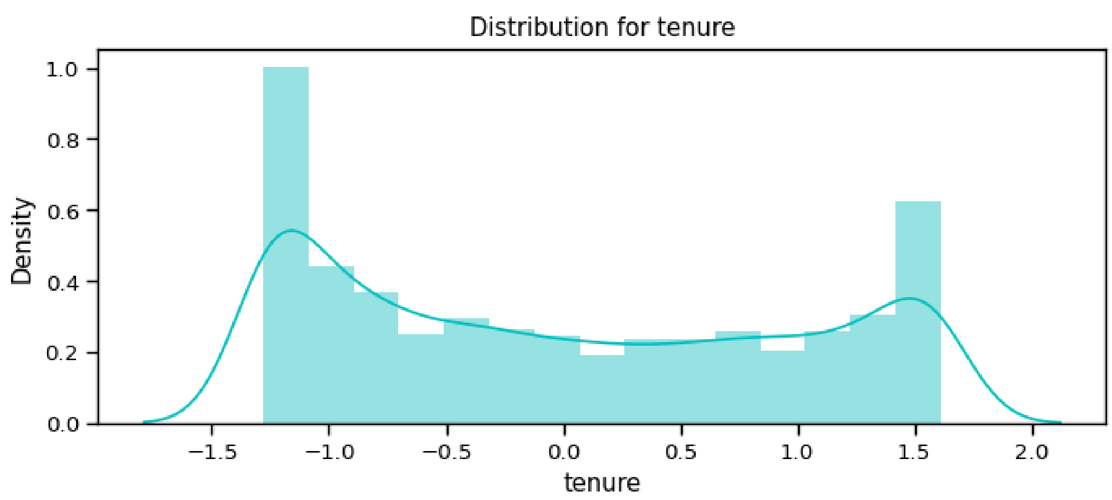
```
num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
for feat in num_cols: distplot(feat, df)
```

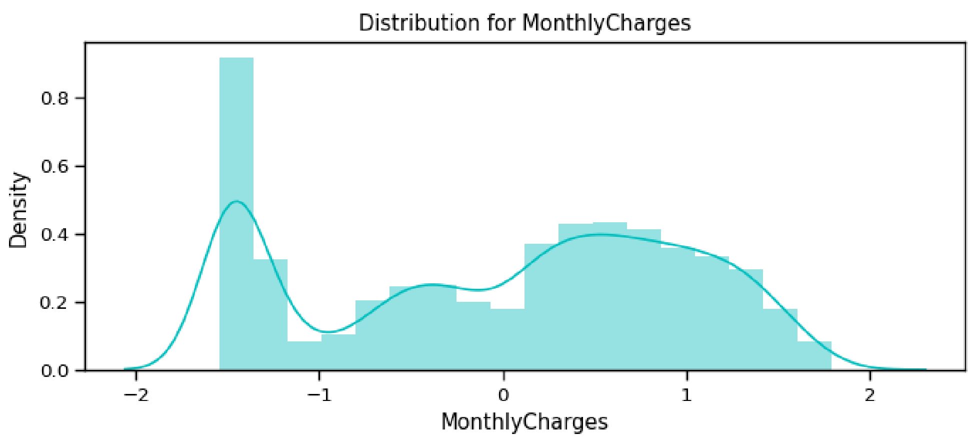


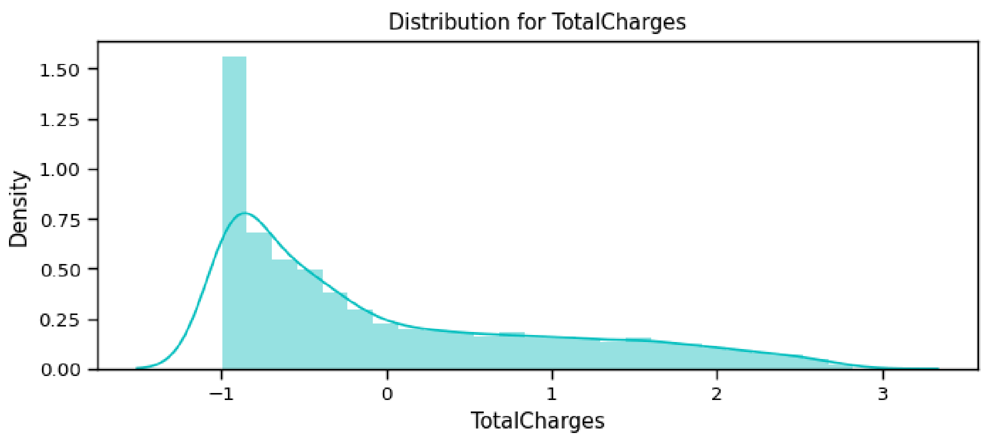



```
df_std = pd.DataFrame(StandardScaler().fit_transform(df[num_cols].astype('float64')),
                       columns=num_cols)
for feat in numerical_cols: distplot(feat, df_std, color='c')
```






After this stage, we now perform K Means Clustering for clustering the dataset.

```
from sklearn.cluster import KMeans

# Assuming 'df' is your DataFrame and it's already preprocessed
X = df[['tenure', 'MonthlyCharges', 'TotalCharges']] # Select the features to use

# Create a KMeans instance with 4 clusters
kmeans = KMeans(n_clusters=4, random_state=0)

# Fit the model to the data
kmeans.fit(X)

# Get the cluster assignments for each data point
labels = kmeans.labels_

# Add the cluster labels to the original DataFrame
df['cluster'] = labels

# Display the DataFrame with the added cluster labels
print(df)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=40, stratify=y)
```

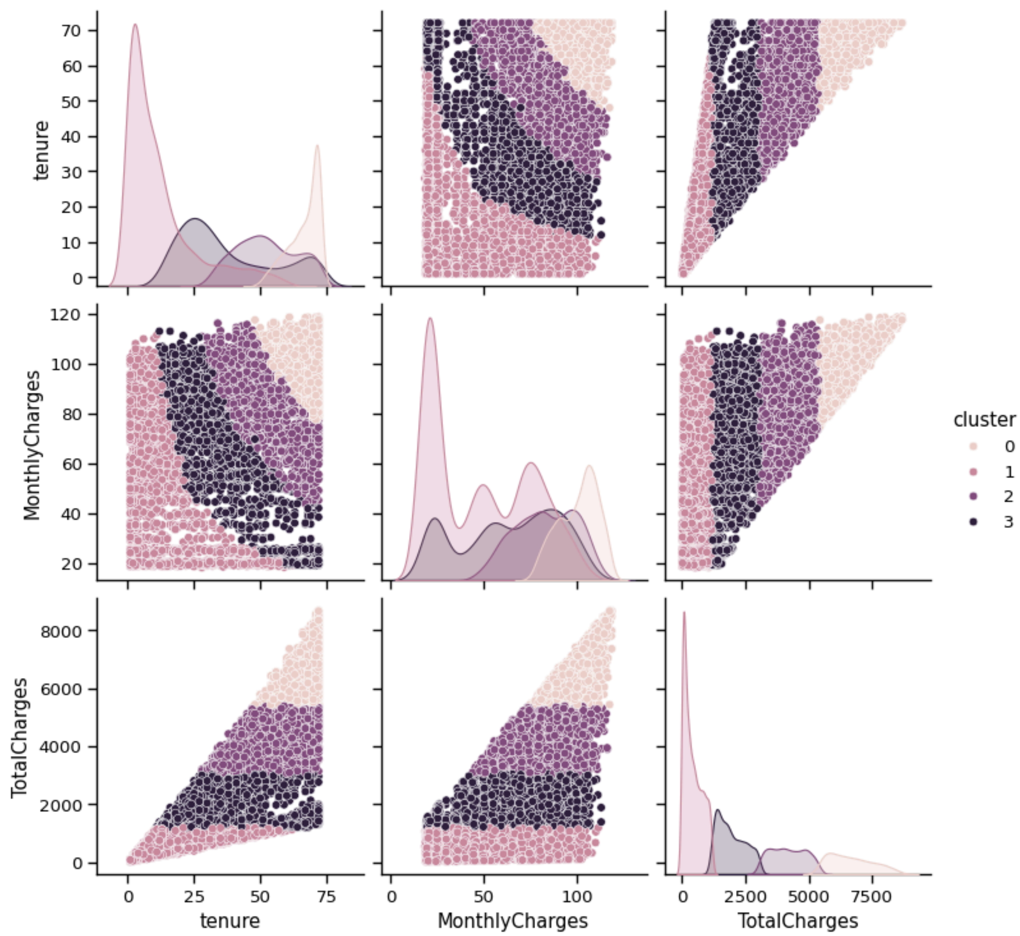
```
      gender  SeniorCitizen  Partner  Dependents  tenure  PhoneService  \
0          0              0        1           0       1             0
1          1              0        0           0      34             1
2          1              0        0           0       2             1
3          1              0        0           0      45             0
4          0              0        0           0       2             1
...      ...            ...      ...         ...     ...           ...
7038       1              0        1           1      24             1
7039       0              0        1           1      72             1
7040       0              0        1           1      11             0
7041       1              1        1           0       4             1
7042       1              0        0           0      66             1

      MultipleLines  InternetService  OnlineSecurity  OnlineBackup  ...  \
0                 1                0               0             2  ...
1                 0                0               2             0  ...
2                 0                0               2             2  ...
3                 1                0               2             0  ...
4                 0                1               0             0  ...
...             ...              ...             ...           ...  ...
7038              2                0               2             0  ...
7039              2                1               0             2  ...
7040              1                0               2             0  ...
7041              2                1               0             0  ...
7042              0                1               2             0  ...

      TechSupport  StreamingTV  StreamingMovies  Contract  PaperlessBilling  \
0               0            0                0         0                 1
1               0            0                0         1                 0
2               0            0                0         0                 1
3               2            0                0         1                 0
4               0            0                0         0                 1
...           ...          ...              ...       ...               ...
7038            2            2                2         1                 1
7039            0            2                2         1                 1
7040            0            0                0         0                 1
7041            0            0                0         0                 1
7042            2            2                2         2                 1

      PaymentMethod  MonthlyCharges  TotalCharges  Churn  cluster
0                 2           29.85         29.85      0        1
1                 3           56.95       1889.50      0        3
2                 3           53.85        108.15      1        1
3                 0           42.30       1840.75      0        3
4                 2           70.70        151.65      1        1
...             ...             ...           ...    ...      ...
7038              3           84.80       1990.50      0        3
7039              1          103.20       7362.90      0        0
7040              2           29.60        346.45      0        1
7041              3           74.40        306.60      1        1
7042              0          105.65       6844.50      0        0

[7032 rows x 21 columns]
```

```python
# Divide the columns into 3 categories, one for standardisation, one for label
# encoding and one for one hot encoding

cat_cols_ohe =['PaymentMethod', 'Contract', 'InternetService']
# those that need one-hot encoding
cat_cols_le = list(set(X_train.columns)- set(num_cols) - set(cat_cols_ohe))
#those that need label encoding

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

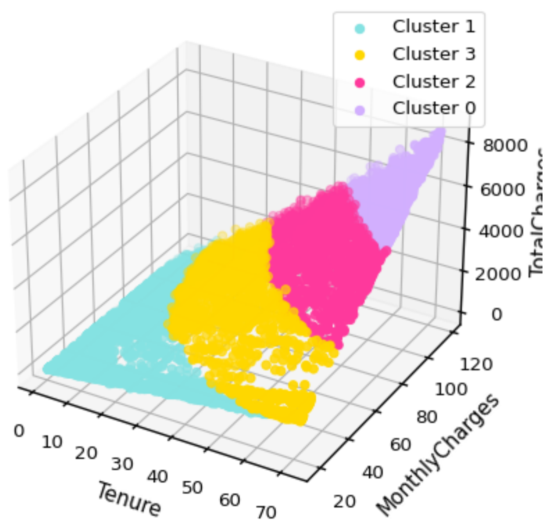
```python
# Group by cluster and calculate the mean tenure, churn rate, and total charges
cluster_summary = df.groupby('cluster')[['tenure', 'MonthlyCharges', 'TotalCharges']].mean()
print(cluster_summary)
```

```
             tenure  MonthlyCharges  TotalCharges
cluster
0         66.285714      100.409918   6654.743422
1         12.100345       48.288664    426.630699
2         52.107354       82.734996   4175.672570
3         37.500296       62.889212   1944.728749
```

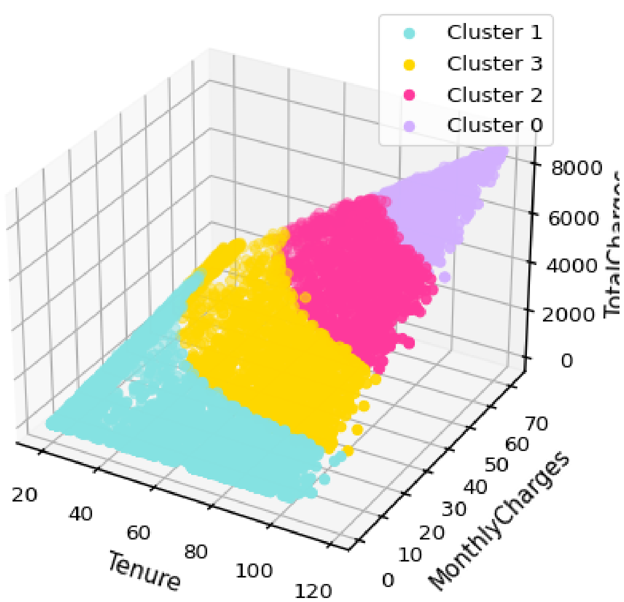
```python
import seaborn as sns
# Create a pair plot
sns.pairplot(df, hue='cluster', vars=['tenure', 'MonthlyCharges', 'TotalCharges'])
# Show the plot
plt.show()
```



```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D subplot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
colors=['#d2afff','#85e2e2','#ff399c','#ffd700']
# Plot each cluster with a different color
for cluster in df['cluster'].unique():
    cluster_data = df[df['cluster'] == cluster]
    ax.scatter(cluster_data['tenure'], cluster_data['MonthlyCharges'],
               cluster_data['TotalCharges'], color=colors[cluster],
               label= f'Cluster {cluster}')

# Set labels for the axes
ax.set_xlabel('Tenure')
ax.set_ylabel('MonthlyCharges')
ax.set_zlabel('TotalCharges')

# Add a legend
ax.legend()

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



```python
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Create a 3D subplot
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
colors=['#d2afff','#85e2e2','#ff399c','#ffd700']
# Plot each cluster with a different color
for cluster in df['cluster'].unique():
    cluster_data = df[df['cluster'] == cluster]
    ax.scatter(cluster_data['MonthlyCharges'], cluster_data['tenure'],
               cluster_data['TotalCharges'], color=colors[cluster],
               label= f'Cluster {cluster}')

# Set labels for the axes
ax.set_xlabel('Tenure')
ax.set_ylabel('MonthlyCharges')
ax.set_zlabel('TotalCharges')

# Add a legend
ax.legend()

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



We now evaluate the efficiency of models using different machine learning models and predict the probability of a customer being churned.

1. **KNN**:

```
num_folds=5
knn_model = KNeighborsClassifier(n_neighbors = 11)
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
knn_model.fit(X_train,y_train)
cv_scores = cross_val_score(knn_model, X_train, y_train, cv=kf)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
print("KNN accuracy:",accuracy_knn)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
KNN accuracy: 0.7739336492890996
Cross Validations scores:  [0.7715736  0.7857868  0.78556911 0.77947154 0.76626016]
Mean CV score: 0.7777322438199
Standard deviation of CV scores: 0.007731170295883199
```

```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.85      1549
           1       0.60      0.44      0.51       561

    accuracy                           0.77      2110
   macro avg       0.71      0.67      0.68      2110
weighted avg       0.76      0.77      0.76      2110
```

2. **SVC**:

```
svc_model = SVC(random_state = 1)
svc_model.fit(X_train,y_train)
predict_y = svc_model.predict(X_test)
accuracy_svc = svc_model.score(X_test,y_test)
print("SVM accuracy is :",accuracy_svc)
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(svc_model, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
SVM accuracy is : 0.79478672985782
Cross Validations scores:  [0.79086294 0.77664975 0.78760163 0.78963415 0.77845528]
Mean CV score: 0.7846407494531797
Standard deviation of CV scores: 0.005908174314368137
```

```
print(classification_report(y_test, predict_y))
```

```
              precision    recall  f1-score   support

           0       0.81      0.95      0.87      1549
           1       0.73      0.37      0.49       561

    accuracy                           0.79      2110
   macro avg       0.77      0.66      0.68      2110
weighted avg       0.78      0.79      0.77      2110
```

3. **Random Forest**:

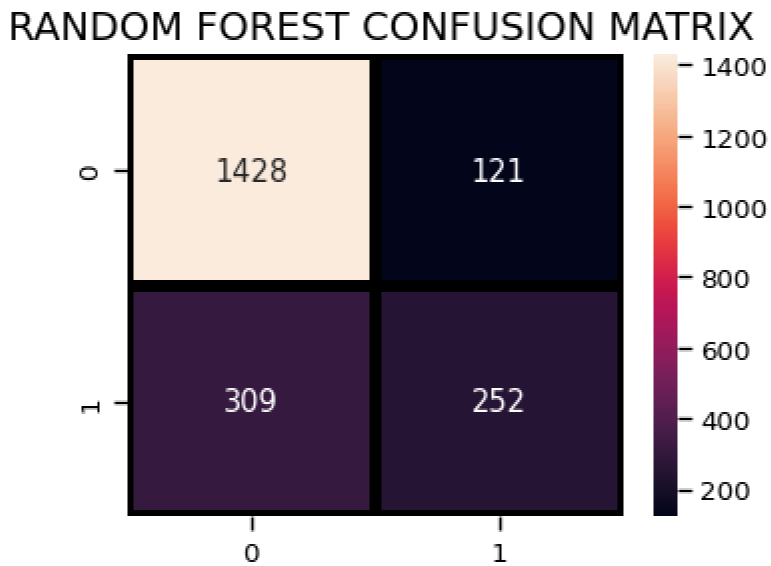
```
model_rf = RandomForestClassifier(n_estimators=500 , oob_score = True, n_jobs = -1,
                                  random_state =50, max_features = "auto",
                                  max_leaf_nodes = 30)
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(model_rf, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
0.7962085308056872
Cross Validations scores:  [0.7857868  0.7857868  0.78658537 0.79369919 0.7804878 ]
Mean CV score: 0.7864691923568982
Standard deviation of CV scores: 0.004218877291186537
```

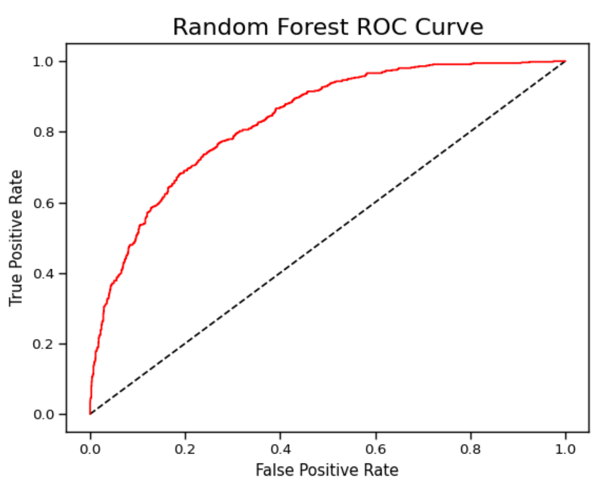
```
print(classification_report(y_test, prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      1549
           1       0.68      0.45      0.54       561

    accuracy                           0.80      2110
   macro avg       0.75      0.69      0.70      2110
weighted avg       0.78      0.80      0.78      2110
```

```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, prediction_test),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title(" RANDOM FOREST CONFUSION MATRIX",fontsize=14)
plt.show()
```




```
y_rfpred_prob = model_rf.predict_proba(X_test)[:,1]
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_rfpred_prob)
plt.plot([0, 1], [0, 1], 'k--' )
plt.plot(fpr_rf, tpr_rf, label='Random Forest',color = "r")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve',fontsize=16)
plt.show();
```



4. **Logistic Regression:**

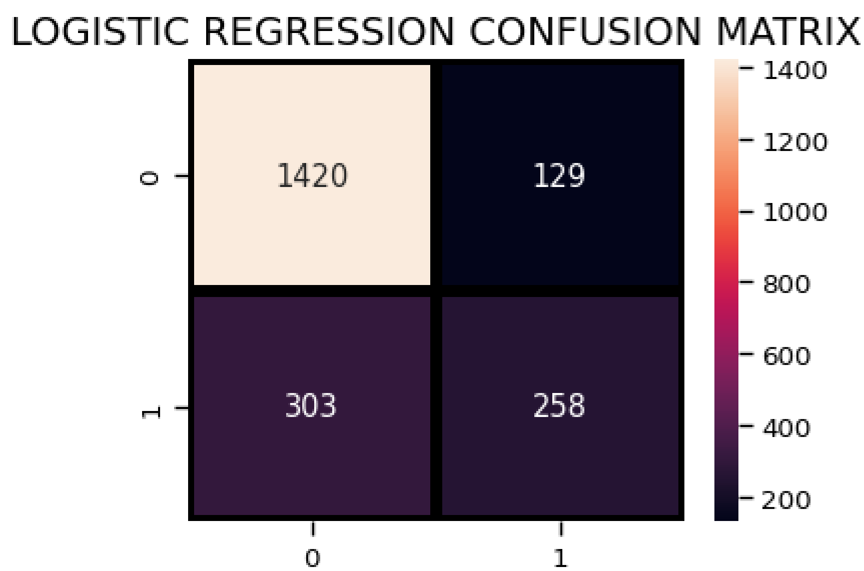
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(lr_model, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
Logistic Regression accuracy is : 0.795260663507109
Cross Validations scores:  [0.78883249 0.7715736  0.79878049 0.78455285 0.76930894]
Mean CV score: 0.7826096735586645
Standard deviation of CV scores: 0.010979060275301336
```

```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      1549
           1       0.67      0.46      0.54       561

    accuracy                           0.80      2110
   macro avg       0.75      0.69      0.71      2110
weighted avg       0.78      0.80      0.78      2110
```

```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, lr_pred),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title("LOGISTIC REGRESSION CONFUSION MATRIX",fontsize=14)
plt.show()
```




```python
y_pred_prob = lr_model.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
plt.plot([0, 1], [0, 1], 'k--' )
plt.plot(fpr, tpr, label='Logistic Regression',color = "r")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Logistic Regression ROC Curve',fontsize=16)
plt.show();
```

5. **Decision Tree Classifier:**

```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train,y_train)
predictdt_y = dt_model.predict(X_test)
accuracy_dt = dt_model.score(X_test,y_test)
print("Decision Tree accuracy is :",accuracy_dt)
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(svc_model, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
Decision Tree accuracy is : 0.7241706161137441
Cross Validations scores:  [0.79086294 0.77664975 0.78760163 0.78963415 0.77845528]
Mean CV score: 0.7846407494531797
Standard deviation of CV scores: 0.005908174314368137
```

```
print(classification_report(y_test, predictdt_y))
```

```
              precision    recall  f1-score   support

           0       0.82      0.80      0.81      1549
           1       0.49      0.52      0.50       561

    accuracy                           0.73      2110
   macro avg       0.65      0.66      0.66      2110
weighted avg       0.73      0.73      0.73      2110
```

Through these computations, we see that the Decision Tree Classifier gives the lowest values for the chosen metric of accuracy.

6. **AdaBoost Classifier:**

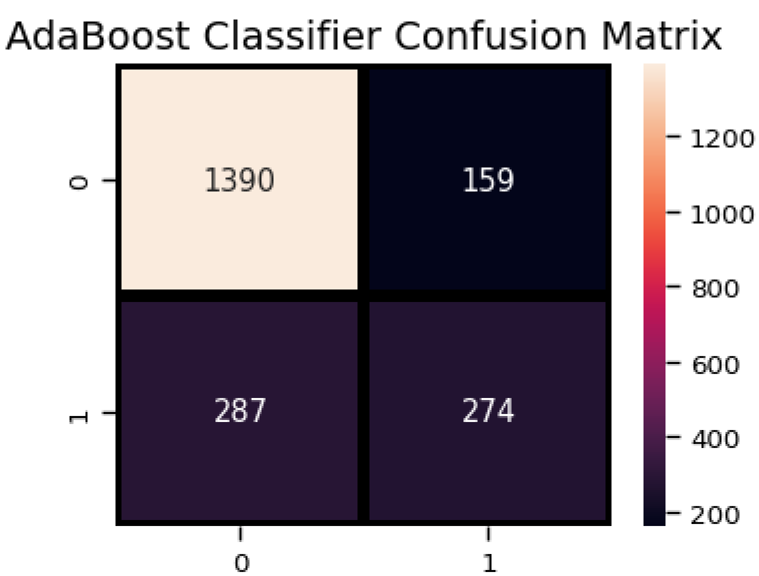
```
a_model = AdaBoostClassifier()
a_model.fit(X_train,y_train)
a_preds = a_model.predict(X_test)
print("AdaBoost Classifier accuracy:")
print(metrics.accuracy_score(y_test, a_preds))
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(a_model, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
AdaBoost Classifier accuracy:
0.7886255924170616
Cross Validations scores:  [0.78883249 0.78680203 0.78963415 0.79268293 0.7804878 ]
Mean CV score: 0.7876878791630556
Standard deviation of CV scores: 0.004066309938177863
```

```
print(classification_report(y_test, a_preds))
```

```
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1549
           1       0.63      0.49      0.55       561

    accuracy                           0.79      2110
   macro avg       0.73      0.69      0.71      2110
weighted avg       0.78      0.79      0.78      2110
```

```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, a_preds),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title("AdaBoost Classifier Confusion Matrix",fontsize=14)
plt.show()
```




7. **Gradient Boosting Classifier:**

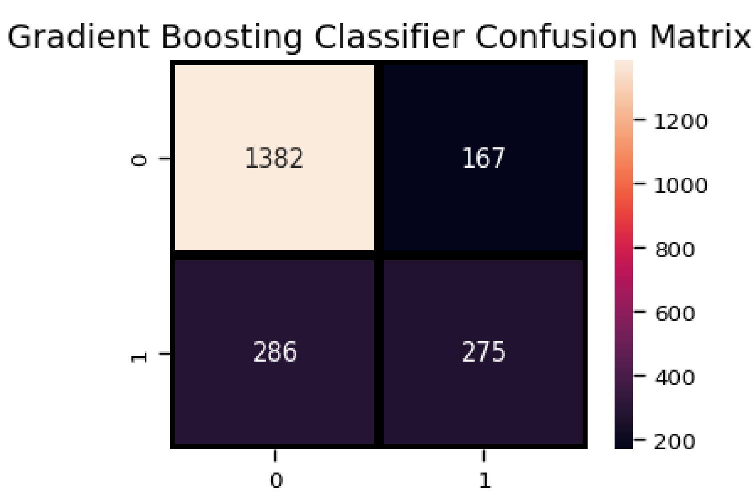
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print("Gradient Boosting Classifier", accuracy_score(y_test, gb_pred))
num_folds=5
kf = KFold(n_splits=num_folds, shuffle=True, random_state=42)
cv_scores = cross_val_score(gb, X_train, y_train, cv=kf)
print("Cross Validations scores: ",cv_scores)
print("Mean CV score:", cv_scores.mean())
print("Standard deviation of CV scores:", cv_scores.std())
```

```
Gradient Boosting Classifier 0.7853080568720379
Cross Validations scores:  [0.79086294 0.79390863 0.78963415 0.78963415 0.76930894]
Mean CV score: 0.7866697618752838
Standard deviation of CV scores: 0.008819945133368874
```

```
print(classification_report(y_test, gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1549
           1       0.62      0.49      0.55       561

    accuracy                           0.79      2110
   macro avg       0.73      0.69      0.70      2110
weighted avg       0.77      0.79      0.78      2110
```

```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, gb_pred),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title("Gradient Boosting Classifier Confusion Matrix",fontsize=14)
plt.show()
```



8. **(Soft) Voting Classifier:** We go with the Soft Voting Classifier to compute the values by predicting the final model based on the highest majority of voting and check its score. We don't use *Hard Voting Classifier* as there is an imbalance in the dataset.

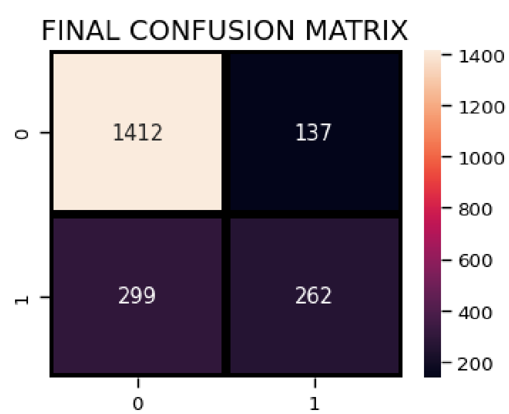
```
from sklearn.ensemble import VotingClassifier
clf1 = GradientBoostingClassifier()
clf2 = LogisticRegression()
clf3 = AdaBoostClassifier()
eclf1 = VotingClassifier(estimators=[('gbc', clf1), ('lr', clf2), ('abc', clf3)], voting='soft')
eclf1.fit(X_train, y_train)
predictions = eclf1.predict(X_test)
print("Final Accuracy Score ")
print(accuracy_score(y_test, predictions))
```

```
Final Accuracy Score
0.7933649289099526
```

```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.83      0.91      0.87      1549
           1       0.66      0.47      0.55       561

    accuracy                           0.79      2110
   macro avg       0.74      0.69      0.71      2110
weighted avg       0.78      0.79      0.78      2110
```

```
plt.figure(figsize=(4,3))
sns.heatmap(confusion_matrix(y_test, predictions),
                annot=True,fmt = "d",linecolor="k",linewidths=3)

plt.title("FINAL CONFUSION MATRIX",fontsize=14)
plt.show()
```



## Analysis: Observations and Inferences

We see that in the base paper [2], the models compared were the RFM model and the cluster based model. This showed that the **RFM based model performed better than the cluster based model with respect to all of the performance metrics**. This paper also established that clustered dataset acts as a reliable input for further predictive modeling analysis.

Table 1: RFM based model performance comparison

| | Logistic Regression | CART | Random Forest | Support Vector Machine | Gradient Boosting Machine |
|---|---|---|---|---|---|
| **Accuracy** | 0.9718 | 0.9851 | 0.9875 | 0.9757 | 0.9703 |
| **Sensitivity** | 0.9472 | 0.9759 | 0.9662 | 0.9526 | 0.9332 |
| **Specificity** | 0.9822 | 0.9889 | 0.9966 | 0.9855 | 0.9865 |
| **Balanced Accuracy** | 0.9647 | 0.9824 | 0.9814 | 0.9691 | 0.9598 |

Table 2: Cluster based model performance comparison

| | Logistic Regression | CART | Random Forest | Support Vector Machine | Gradient Boosting Machine |
|---|---|---|---|---|---|
| **Accuracy** | 0.8929 | 0.9061 | 0.8991 | 0.9077 | 0.9030 |
| **Sensitivity** | 0.8420 | 0.8783 | 0.8750 | 0.9509 | 0.8638 |
| **Specificity** | 0.9118 | 0.9160 | 0.9074 | 0.8953 | 0.9175 |
| **Balanced Accuracy** | 0.8769 | 0.8972 | 0.8912 | 0.9231 | 0.8906 |

The base paper we used[2] compared different models such as logistic regression, CART, Random Forest, Support Vector Machine, and Gradient Boosting Machine. We see that, on the Retail Dataset used in this paper, the accuracy values obtained lie between the range **0.8420 to 0.9822**. The dataset has around 53,000 data points and is a relatively balanced set. According to the paper, its scope can be extended to design a prediction on prediction model, where the future churn behavior of a customer is established by taking their future clusters into consideration.

We have taken into consideration the future scope of this paper, and after having implemented the given steps in the paper, we introduced our novelty by computing the values on the Telecom Dataset, visualized and preprocessed the data, and then performed **K-means clustering**, as suggested in [2]. After clustering the data points, we ran different models on the cleaned dataset and obtained the values as mentioned below:

| Name of classifier | Accuracy | Precession | Recall | K Fold Cross Validation Score |
|---|---|---|---|---|
| KNN | 0.773 | 0.82 | 0.89 | 0.777 |
| SVM | 0.794 | 0.81 | 0.95 | 0.784 |
| Random Forest | 0.796 | 0.82 | 0.92 | 0.786 |
| Logistic Regression | 0.795 | 0.82 | 0.92 | 0.782 |
| Decision Tree Classifier | 0.724 | 0.82 | 0.80 | 0.784 |
| AdaBoost Classifier | 0.789 | 0.83 | 0.90 | 0.787 |
| Gradient Boosting Classifier | 0.785 | 0.83 | 0.89 | 0.786 |
| Voting Classifier | 0.793 | 0.83 | 0.91 | - |

K Fold Cross Validation Score is not present in soft voting classifier as it is an algorithm that can be used to combine the predictions of multiple classifiers using the probability of predictions. Its input values are the outputs of other classification models and not the dataset as a whole.

The Voting Classifier averages out the values and presents the most accurate model to use with respect to the dataset. We have obtained the highest value of accuracy to be present in the case of the **Random Forest** model (0.796).

Customer churn is definitely bad to a firm 's profitability. Various strategies can be implemented to eliminate customer churn. The best way to avoid customer churn is for a company to truly know its customers. This includes identifying customers who are at risk of churning and working to improve their satisfaction.

**Improving customer service** is, of course, at the top of the priority for tackling this issue. **Building customer loyalty** through relevant experiences and specialized service is another strategy to reduce customer churn.

Some firms survey customers who have already churned to understand their reasons for leaving in order to adopt a **proactive approach** to avoiding future customer churn.

**Future Scope:**
There exists a class imbalance in the dataset. Fixing the imbalance will help improve the performance of the model.

**References:**


- Code
- [2] H. A. S and M. C, "Evaluative study of cluster based customer churn prediction against conventional RFM based churn model," 2023 Second International Conference on Electrical, Electronics, Information and Communication Technologies (ICEEICT), Trichirappalli, India, 2023
- Dataset Implemented in base Paper mentioned at [2]
- F. Chen, X. Wei, S. Yu, P. Ma and S. He, "Customer Churn Prediction based on Stacking Model," 2023 4th International Conference on Computer Vision, Image and Deep Learning (CVIDL), Zhuhai, China, 2023
- Telecom Dataset
- Analysis of Dataset



**Acknowledgements:**

We would like to extend our heartfelt gratitude to our professor Dr. Shruthi M.L.J ma'am for giving us the opportunity to present our idea. We are grateful for the guidance and knowledge we received in the process of completing this project.