Tittle of Project

Customer Churn Prediction and Analysis for a Subscription-Based Business

Submitted in Partial Fulfilment of the Requirement for the Award of Degree of

MASTER OF BUSINESS ADMINISTRATION

(MBA)

Submitted by:

Rahul Singh
Reg. No.: 2314107786

Under the Guidance of

Saransh Kapoor

Centre for Distance and Online Education

Manipal University Jaipur

September 2024

Project_PPT_Rahul Singh.pptx

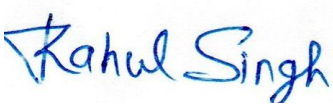
BONAFIDE CERTIFICATE

This is to certify that **Rahul Singh**, a student of **Master of Business Administration (MBA)**, **2314107786**, has successfully completed the project titled "***Customer Churn Prediction and Analysis for a Subscription-Based Business***" under my supervision as a part of the requirements for the MBA program at centre for distance and online education, Manipal University Jaipur during the academic year **2024-2025**. This project report embodies the original work of the student, conducted with due diligence, and adheres to the standards expected by the institution. It has not been submitted to any other institution for any degree, diploma, or certificate.

Saransh Kapoor
Senior Manager
Sadhana Centre for Management & Leadership Development
Date: August 01, 2010
Place: Pune, Maharashtra

DECLARATION BY THE STUDENT

I, Rahul Singh, a student of Master of Business Administration (MBA), 2314107786, hereby declare that the project report titled "*Customer Churn Prediction and Analysis for a Subscription-Based Business*" submitted to Centre for Distance and Online Education, Manipal University Jaipur is a record of my original work carried out under the guidance of Saransh Kapoor, Senior Manager. I affirm that this project is the result of my own independent effort, and to the best of my knowledge, it does not contain any material previously published or written by any other person or material which has been accepted for the award of any other degree or diploma at any other educational institution, except where due acknowledgment has been made in the text. I also declare that I have adhered to all the guidelines and standards required for academic honesty and have cited all sources wherever used.



Rahul Singh

2314107786

Date: December 9, 2024

Place: Noida, UP

Table of Contents

1. *Executive Summary*

Introduction: Customer churn occurs when customers or subscribers stop using a firm's services. In the highly competitive telecom industry, where churn rates can range between 15-25% annually, retaining customers is more cost-effective than acquiring new ones. The ability to predict churn and focus retention efforts on high-risk customers can significantly improve a company's profitability and market position.

This project aims to explore customer behaviour, identify patterns leading to churn, and provide actionable insights for telecom companies to implement effective retention strategies.

Key Objectives:

1. **Analyze Churn Rates**: Determine the percentage of customers who churn versus those who continue using the service.
2. **Demographic Patterns**: Identify any gender-based trends in customer churn.
3. **Service Preferences**: Examine churn patterns linked to different service types and preferences.
4. **Profitability Analysis**: Assess which service types and features generate the most profit.
5. **Uncover Insights**: Address additional questions and trends that arise during data exploration.

Significance: The telecom industry thrives on customer loyalty and cost-efficient operations. Predicting churn enables companies to develop proactive strategies, enhance customer satisfaction, and minimize revenue losses. Insights derived from this analysis will assist businesses in preserving their market share and driving sustainable growth.

Expected Outcomes:

- A clear understanding of churn drivers.
- Identification of high-risk customer segments for targeted retention efforts.
- Strategic recommendations to optimize services and reduce churn.
- Data-backed insights to inform future business decisions and customer engagement strategies.

This project underscores the importance of leveraging data analytics to retain customers, reduce churn rates, and foster long-term business success.

2. *LOADING LIBRARIES AND DATA*

```
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')
```

- Pandas (pd): Used for data manipulation, such as loading, cleaning, and transforming data.

- NumPy (np): Provides support for numerical operations, arrays, and mathematical functions.
- Missingno (msno): Helps visualize missing data to aid in data cleaning.
- Matplotlib (plt), Seaborn (sns), Plotly (px, go): Visualization libraries used to create static and interactive plots for data exploration and model results.
- Scikit-learn (sklearn): A key library for machine learning. It offers tools for data preprocessing (e.g., scaling, encoding), model training (e.g., Decision Trees, Random Forest), and model evaluation (e.g., accuracy, precision, recall).
- XGBoost (XGBClassifier) and CatBoost (CatBoostClassifier): Advanced machine learning models for classification, optimized for performance and handling categorical data.
- Warnings: Used to suppress unnecessary warning messages during execution.

```
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 catboost import CatBoostClassifier
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
```

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

The code df = pd.read_csv('Telco-Customer-Churn.csv') loads the customer churn dataset from a CSV file into a pandas DataFrame (df) for further analysis and processing.

3. *UNDERSTANDING THE DATA*

df.head()
df.shape
df.info()
df.columns.values
df.dtypes

- df.head(): Displays the first 5 rows of the DataFrame to preview the data.

- df.shape: Returns the number of rows and columns in the DataFrame.
- df.info(): Provides a summary of the DataFrame, including the data types and non-null counts.
- df.columns.values: Lists the column names in the DataFrame.
- df.dtypes: Shows the data type of each column in the DataFrame.

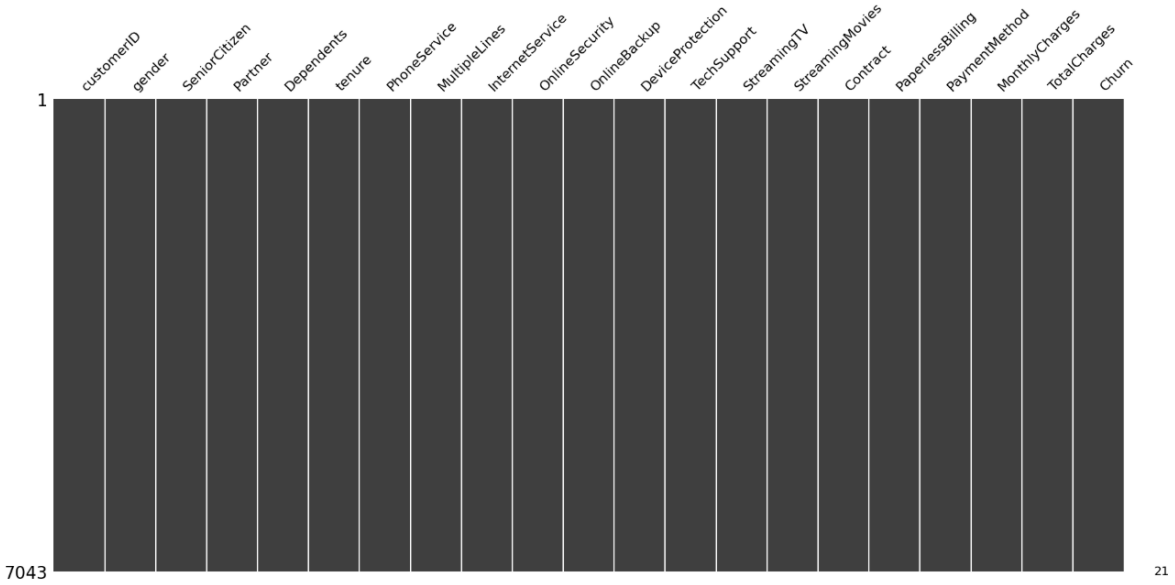
The data set includes information about:

- 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 have 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

4. *VISUALIZE MISSING VALUES*

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

Once the query has been executed, retrieve the output.



Using this matrix we can very quickly find the pattern of missingness in the dataset.

From the above visualisation we can observe that it has no peculiar pattern that stands out. In fact there is no missing data.

5. *DATA MANIPULATION*

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

The command df = df.drop(['customerID'], axis = 1) removes the **'customerID'** column from the DataFrame df, and df.head() displays the first 5 rows of the updated DataFrame after the column has been dropped.

On deep analysis, we can find some indirect missingness in our data (which can be in form of blank spaces).

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

The command df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce') attempts to convert the values in the **'TotalCharges'** column to numeric format, and any non-numeric values are replaced with **NaN** (missing values) due to the errors='coerce' parameter.

The command df.isnull().sum() then checks and sums the number of missing values (NaNs) in each column of the DataFrame df.

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

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

It can also be noted that the Tenure column is 0 for these entries even though the MonthlyCharges column is not empty. Let's see if there are any other 0 values in the tenure column.

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

The command df[df['tenure'] == 0].index returns the index (row labels) of all rows in the DataFrame df where the value in the **'tenure'** column is equal to 0. This can help identify customers who have just started or have no tenure.

There are no additional missing values in the Tenure column. Let's 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

The command df.drop(labels=df[df['tenure'] == 0].index, axis=0, inplace=True) removes all rows from the DataFrame df where the **'tenure'** column is equal to 0. The axis=0 specifies row deletion, and inplace=True ensures that the change is applied directly to df without needing to create a new DataFrame.

After this, df[df['tenure'] == 0].index will return an empty index because the rows with **'tenure' == 0** have already been dropped.

To solve the problem of missing values in TotalCharges column, I decided to fill it with the mean of TotalCharges values.

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

The command df.fillna(df["TotalCharges"].mean()) fills any missing values (NaNs) in the DataFrame df with the **mean** value of the **'TotalCharges'** column. However, it does not modify the original DataFrame unless assigned back to df or a new variable (e.g., df = df.fillna(df["TotalCharges"].mean())).

df.isnull().sum()

The command df.isnull().sum() returns the count of missing (NaN) values in each column of the DataFrame df. It provides a summary of how many missing values exist across the entire dataset.

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

df.head()

The command df["SeniorCitizen"] = df["SeniorCitizen"].map({0: "No", 1: "Yes"}) maps the values in the **'SeniorCitizen'** column from **0 and 1** to **"No" and "Yes"**, respectively, making the data more readable.

After this transformation, df.head() will display the first 5 rows of the updated DataFrame, where the **'SeniorCitizen'** column now contains **"No"** and **"Yes"** instead of **0** and **1**.

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

The command df["InternetService"].describe(include=['object', 'bool']) generates a summary of statistics for the **'InternetService'** column in the DataFrame df, specifically for categorical (object) and boolean data types. It provides details like the count, unique values, top (most frequent) value, and frequency of the top value.

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

df[numerical_cols].describe()

The command df[numerical_cols].describe() generates a summary of statistical measures (like count, mean, standard deviation, min, max, and quartiles) for the columns listed in numerical_cols (which include **'tenure'**, **'MonthlyCharges'**, and **'TotalCharges'**) in the

DataFrame df. This provides a quick overview of the distribution of the numerical features.

6. *DATA VISUALIZATION*

```
g_labels = ['Male', 'Female']
c_labels = ['No', 'Yes']
# Create subplots: use 'domain' type for Pie subplot
fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}, {'type':'domain'}]])
fig.add_trace(go.Pie(labels=g_labels, values=df['gender'].value_counts(), name="Gender"),
              1, 1)
fig.add_trace(go.Pie(labels=c_labels, values=df['Churn'].value_counts(), name="Churn"),
              1, 2)

# Use `hole` to create a donut-like pie chart
fig.update_traces(hole=.4, hoverinfo="label+percent+name", textfont_size=16)

fig.update_layout(
    title_text="Gender and Churn Distributions",
    # Add annotations in the center of the donut pies.
    annotations=[dict(text='Gender', x=0.16, y=0.5, font_size=20, showarrow=False),
                 dict(text='Churn', x=0.84, y=0.5, font_size=20, showarrow=False)])
fig.show()
```

When we run the above code over python then will get the below response.

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()

The command df["Churn"][df["Churn"] == "No"].groupby(by=df["gender"]).count() performs the following:

1. **df["Churn"] == "No"**: Filters the DataFrame to only include rows where the **'Churn'** column has the value "No", meaning customers who did not churn.
2. **df["Churn"][...]**: Selects the **'Churn'** column after applying the filter.

3. **.groupby(by=df["gender"])**: Groups the filtered data by the **'gender'** column, so the counts are calculated separately for each gender (male/female).
4. **.count()**: Counts the number of occurrences of "No" churn for each gender.

This will return the count of non-churning customers, grouped by their gender.

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

The command df["Churn"][df["Churn"] == "Yes"].groupby(by=df["gender"]).count() performs the following steps:

1. **df["Churn"] == "Yes"**: Filters the DataFrame to only include rows where the **'Churn'** column has the value "Yes", indicating customers who have churned.
2. **df["Churn"][...]**: Selects the **'Churn'** column after applying the filter.
3. **.groupby(by=df["gender"])**: Groups the filtered data by the **'gender'** column, calculating the churn count for each gender (male/female).
4. **.count()**: Counts the number of occurrences of "Yes" churn for each gender.

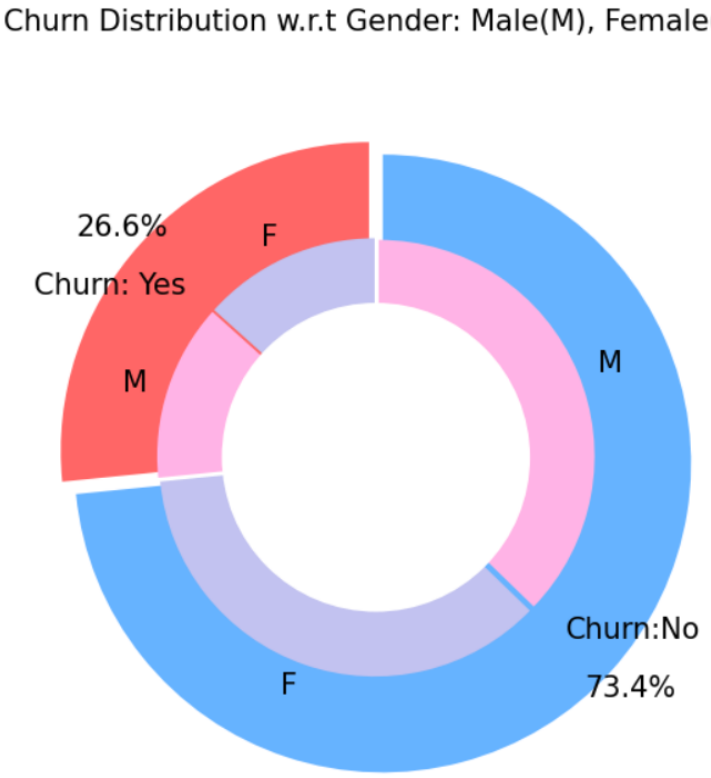
This will return the count of churning customers, grouped by their gender.

```
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()
```

This code creates a **pie chart** that visualizes the distribution of churned vs. non-churned customers, along with a gender-based breakdown of these categories, using two nested pie charts. The outer pie chart shows the proportion of customers who churned vs. those who didn't, and the inner pie chart shows the gender distribution within these categories.

- **Outer Pie Chart**: Represents the overall churn distribution with labels ("Churn: Yes" and "Churn: No") and respective values (1869 churned, 5163 not churned).
- **Inner Pie Chart**: Shows gender breakdown within each churn category, distinguishing male (M) and female (F) customers.
- The **explode** effect highlights slices of the pie chart for emphasis.
- A **circle** is drawn in the center to make the plot look like a donut chart.

The pie charts are visualized with customized text properties, colors, and sizes. The final plot is

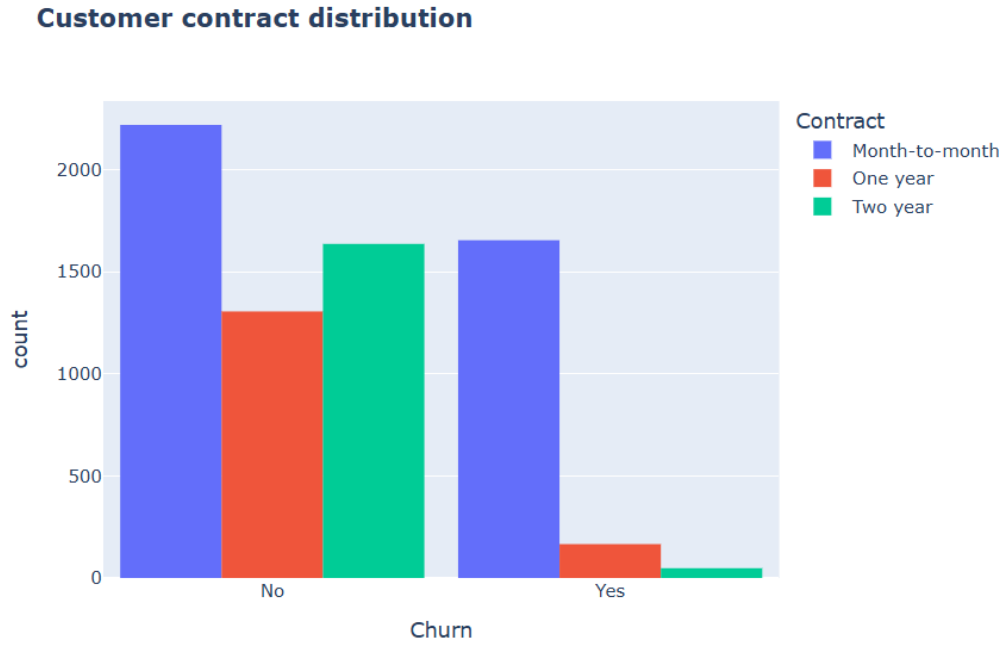


There is negligible difference in customer percentage/ count who chnaged the service provider. Both genders behaved in similar fashion when it comes to migrating to another service provider/firm.

```
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()
```

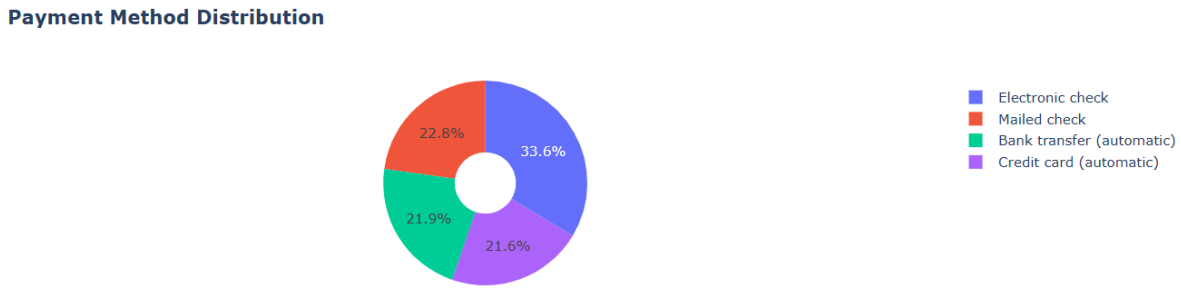
This code creates a grouped histogram using Plotly to show the distribution of customer churn across different contract types, with bars grouped by churn status and colored by contract type.

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

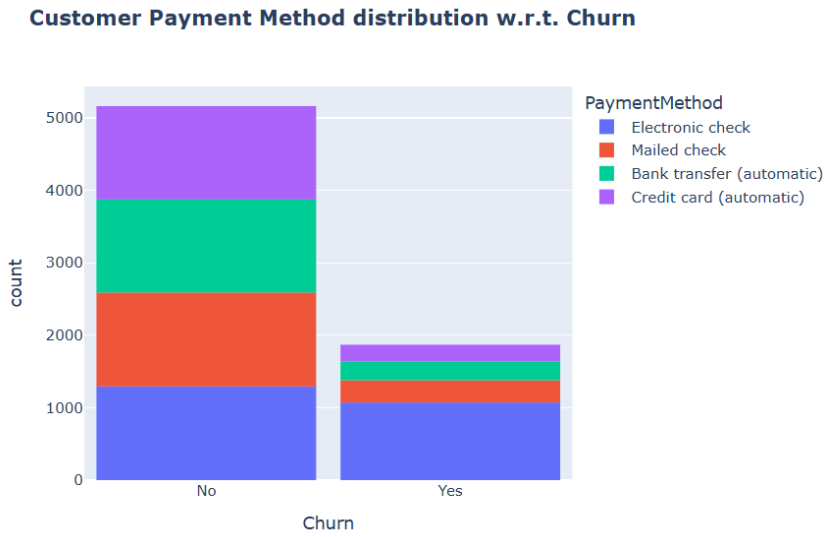


About 75% of customer with Month-to-Month Contract opted to move out as compared to 13% of customers with One Year Contract and 3% with Two Year Contract.

```
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()

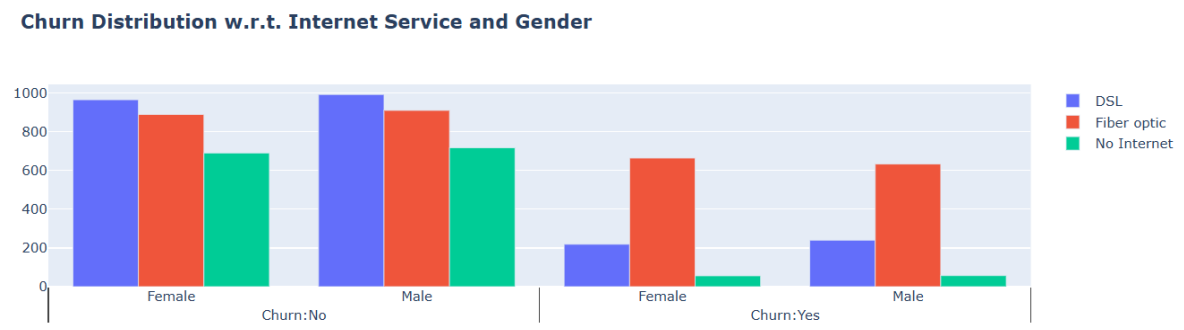
```
InternetService  Churn
DSL              No       992
Fiber optic      No       910
No               No       717
Fiber optic      Yes      633
DSL              Yes      240
No               Yes       57
Name: count, dtype: int64
```

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()
```

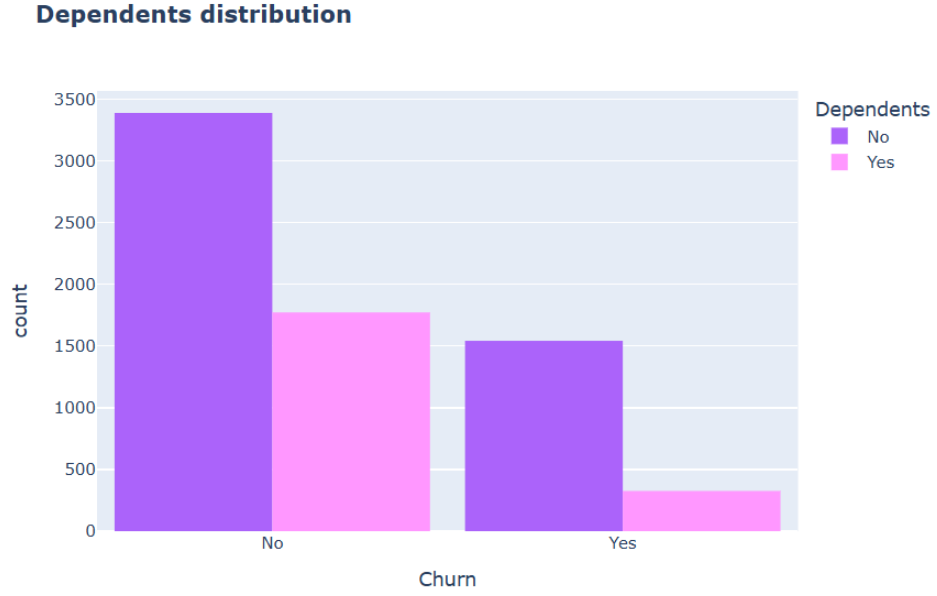
This code creates a grouped bar chart using Plotly (go.Figure) to display the distribution of churn (Churn:Yes and Churn:No) based on Internet Service (DSL, Fiber optic, and No Internet) and Gender (Female and Male). It adds multiple traces for each internet service category, with the corresponding churn counts for each gender, and customizes the chart with a title "Churn Distribution w.r.t. Internet Service and Gender." The bars are grouped by churn status and gender to provide a clear comparison.



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

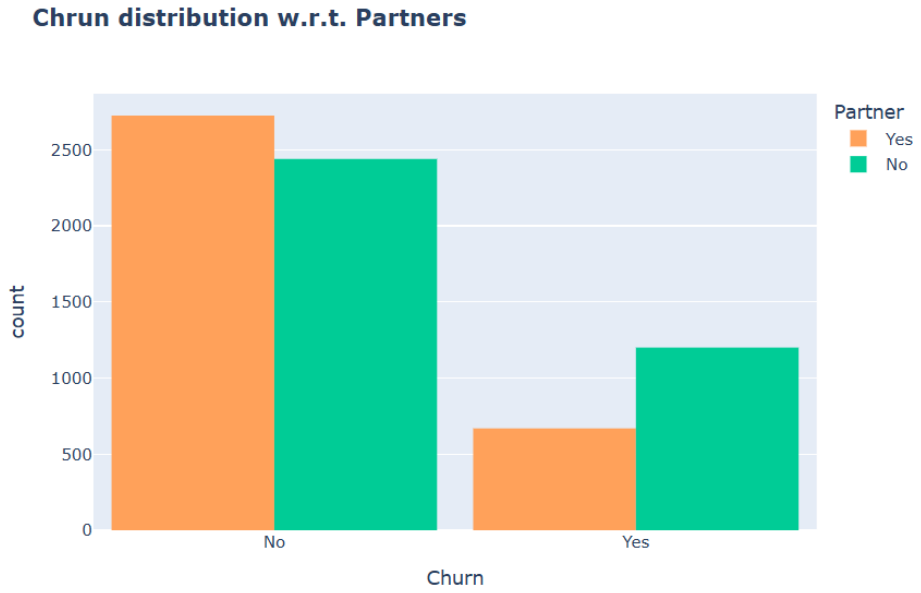
internet service. Customers having DSL service are majority in number and have less churn rate compared to Fibre optic service.

```
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()
```



Customers without dependents are more likely to churn

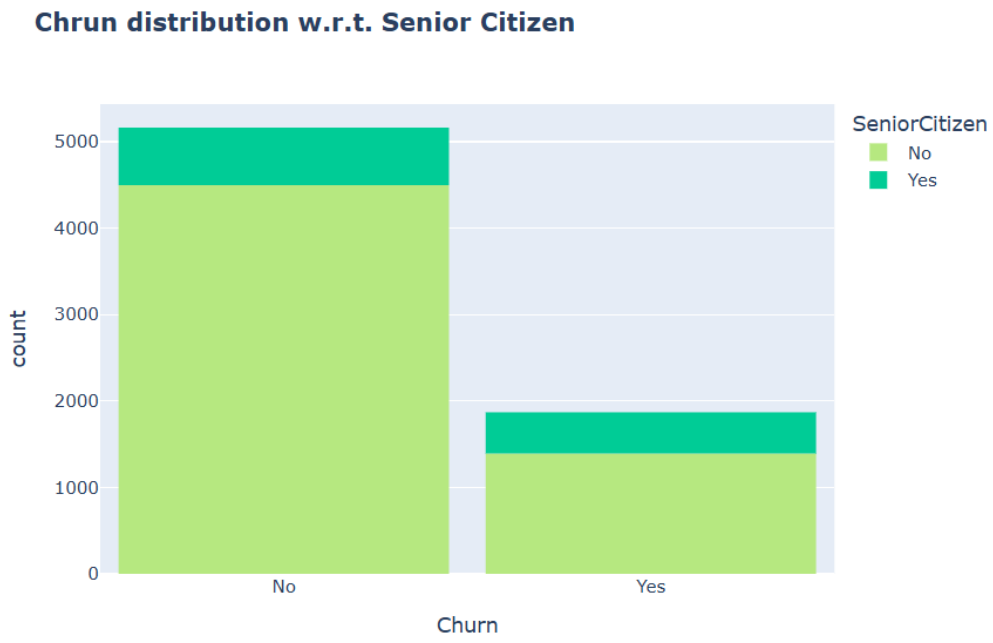
```
color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
fig = px.histogram(df, x="Churn", color="Partner", barmode="group",
                   title="<b>Chrun distribution w.r.t. Partners</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



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

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

This code creates a histogram using Plotly Express (px.histogram) to explore the distribution of Churn in relation to the SeniorCitizen variable. It uses custom colors for the categories of SeniorCitizen ("Yes" in green and "No" in light green), helping to distinguish between senior and non-senior customers, and adds a clean layout with a width of 700 and height of 500. You could experiment further by comparing additional features like TechSupport or Contract to uncover more insights on churn trends among senior and non-senior citizens.



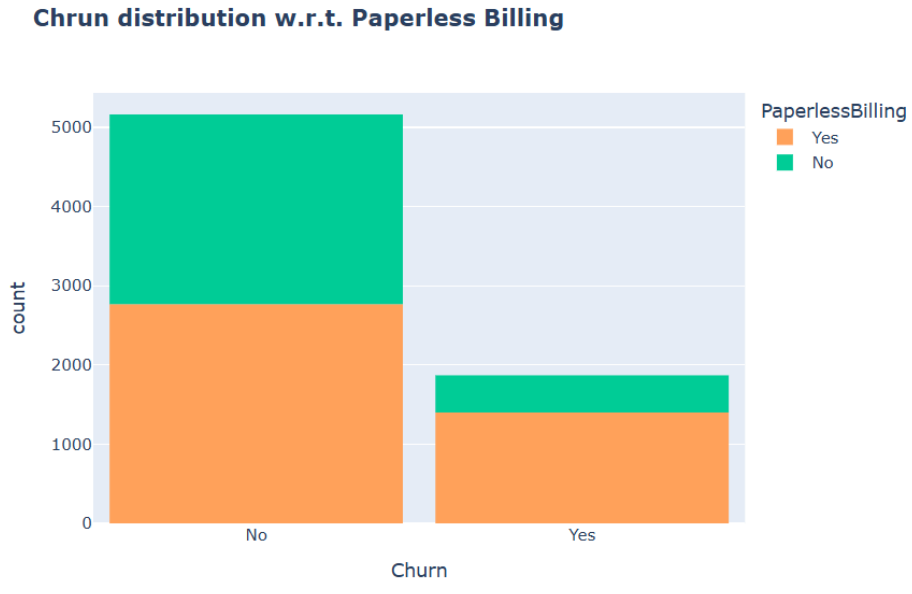
It can be observed that the fraction of senior citizen is very less. Most of the senior citizens churn.

```
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()
```

This code creates a grouped histogram using Plotly Express (px.histogram) to visualize the distribution of Churn in relation to the OnlineSecurity variable from the df DataFrame. It groups the bars by the OnlineSecurity categories, uses a custom color map to differentiate between "Yes" and "No" values, and sets the figure size and bar gap for a clean presentation, with the title "Churn w.r.t Online Security."

Most customers churn in the absence of online security.

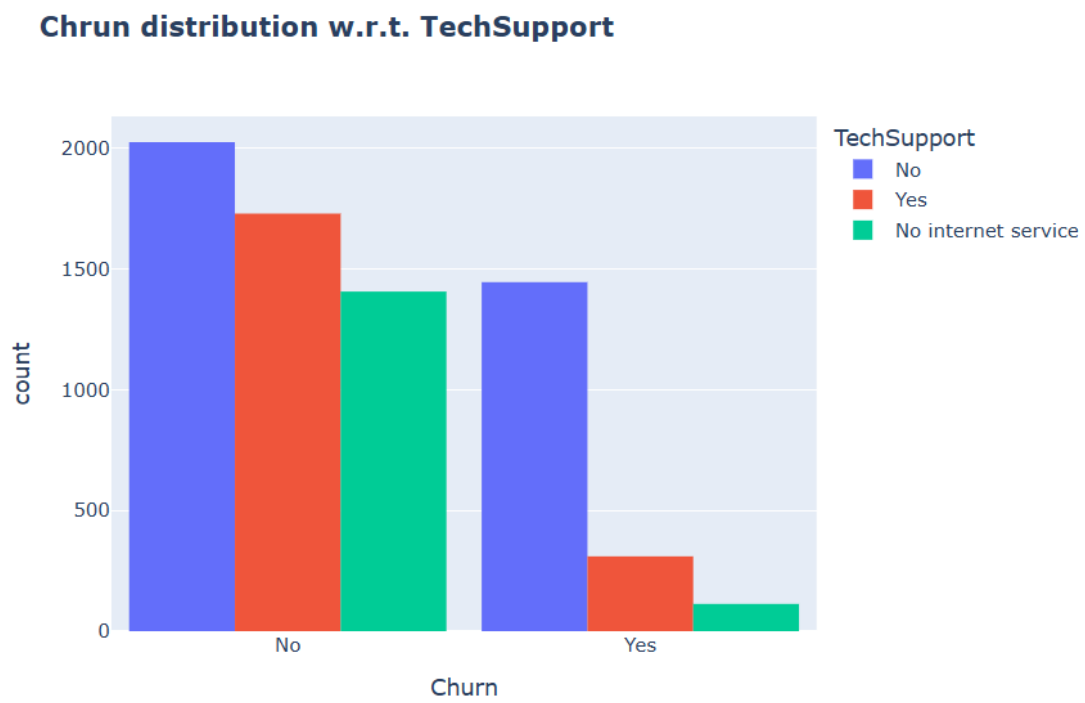
```
color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
fig = px.histogram(df, x="Churn", color="PaperlessBilling",
                   title="<b>Chrun distribution w.r.t. Paperless Billing</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



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()
```

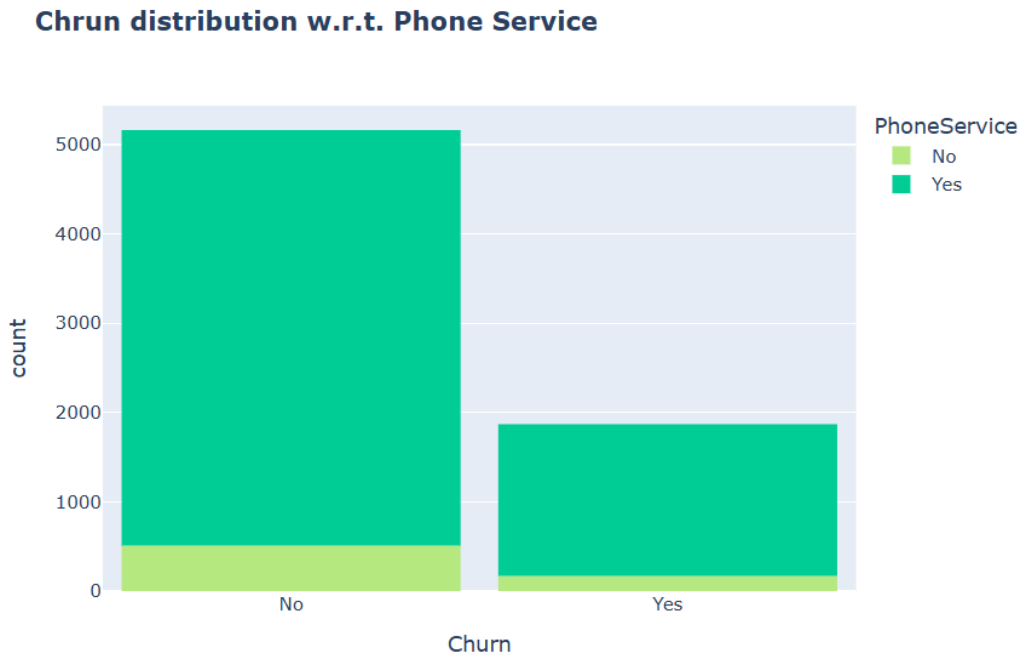
This code creates a grouped histogram using Plotly Express (px.histogram) to visualize the distribution of Churn with respect to the TechSupport variable. It groups the bars by TechSupport categories, adjusts the figure size, and sets the bar gap for better spacing between bars, with the title "Churn distribution w.r.t. TechSupport."



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()
```

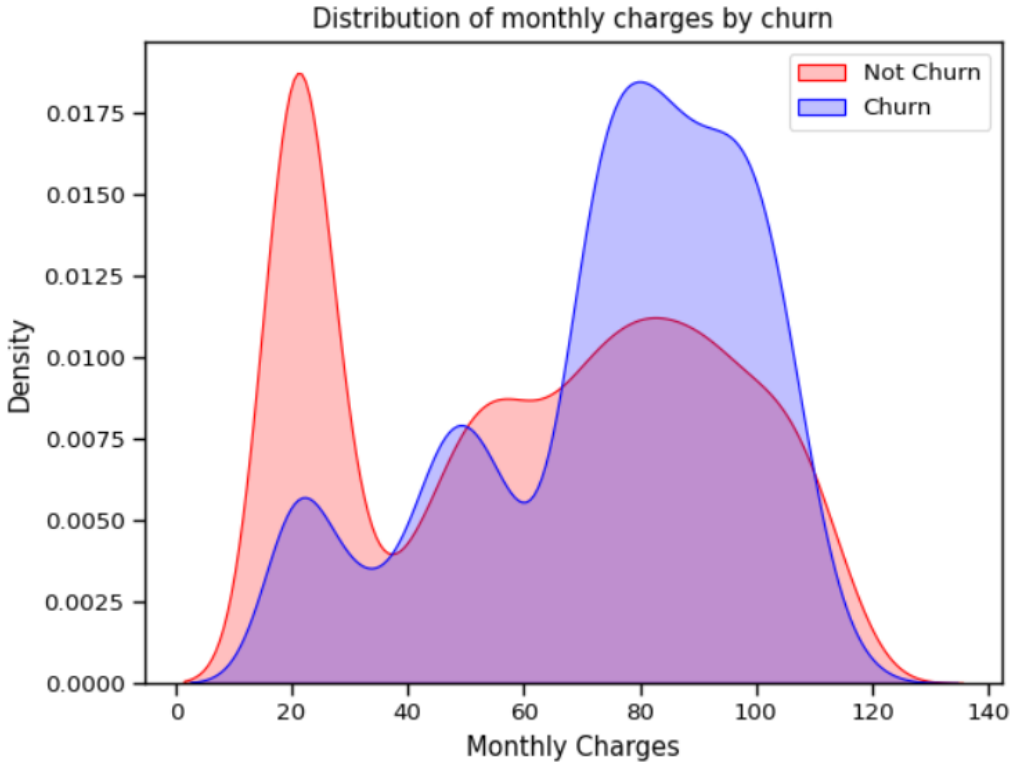
This code creates a histogram using Plotly Express (px.histogram) to visualize the distribution of the Churn variable in relation to the PhoneService variable from the df DataFrame. It applies a custom color map to distinguish between the categories of PhoneService (with specific colors for each category), adjusts the figure size, and sets the bar gap for a cleaner display of the data, with the title "Churn distribution w.r.t. Phone Service."



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');
```

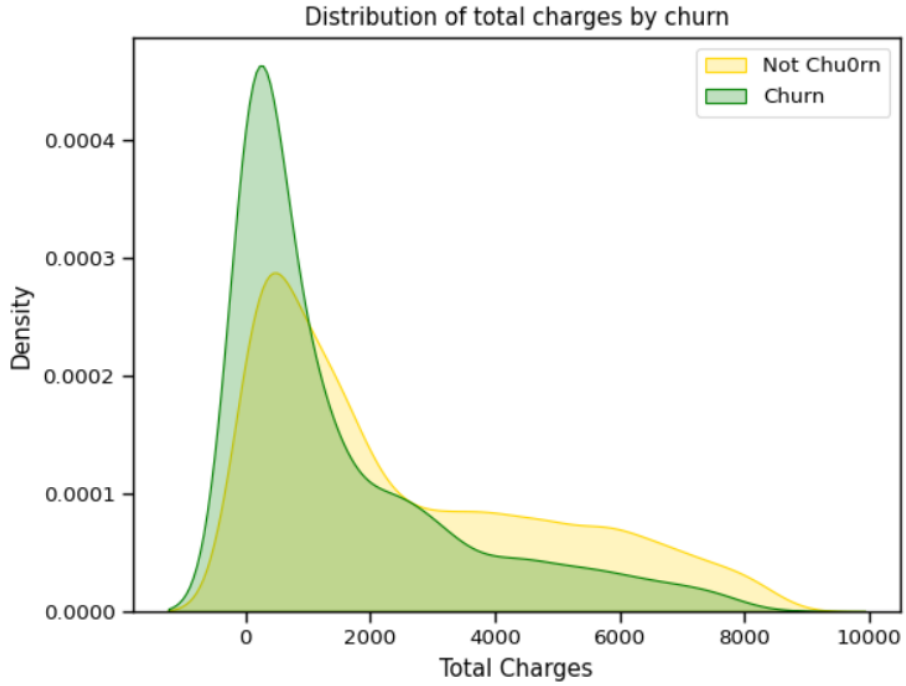
This code creates a Kernel Density Estimate (KDE) plot using Seaborn to compare the distribution of MonthlyCharges for customers who have either churned ('Yes') or not churned ('No'). It customizes the plot with a red shade for "Not Churn" and a blue shade for "Churn," adjusts the font scale and context, adds a legend in the upper right, and labels the axes and title to clearly indicate the distribution of monthly charges by churn status.



Customers with higher Monthly Charges are also more likely to churn

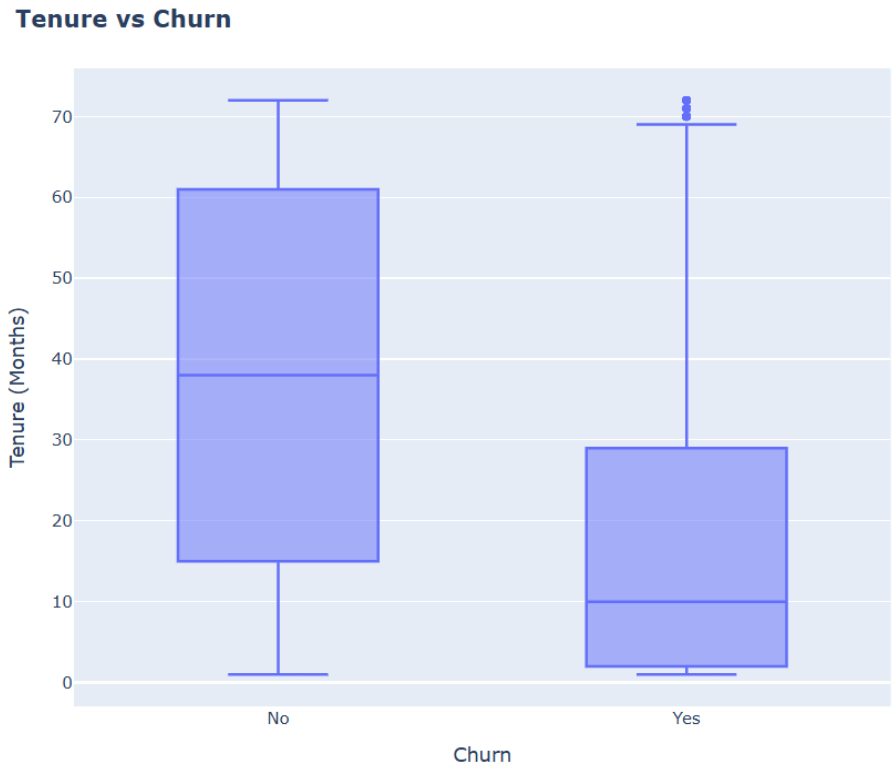
```
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'No') ],
                 color="Gold", shade = True);
ax = sns.kdeplot(df.TotalCharges[(df["Churn"] == 'Yes') ],
                 ax =ax, color="Green", shade= True);
ax.legend(["Not Chu0rn","Churn"],loc='upper right');
ax.set_ylabel('Density');
ax.set_xlabel('Total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

This code creates a Kernel Density Estimate (KDE) plot using Seaborn (sns.kdeplot) to visualize the distribution of TotalCharges for customers who have either churned ('Yes') or not churned ('No'). It overlays two KDE plots, with different colors for each group (Gold for "No Churn" and Green for "Churn"), customizes the axis labels, title, and adds a legend in the upper right corner to distinguish the two categories.




```
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()
```

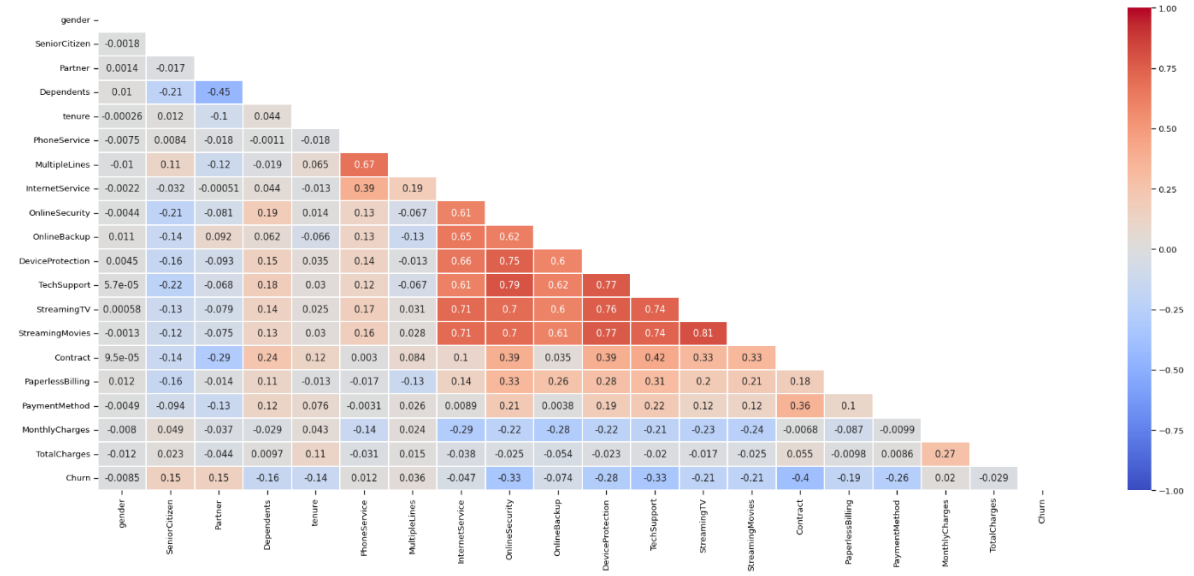
This code creates a box plot using Plotly Express (px.box), displaying the relationship between tenure (y-axis) and Churn (x-axis) from the df DataFrame. It customizes the plot by updating the axis labels, setting the figure size, adjusting the title's font, and displaying the plot with the title "Tenure vs Churn."



New customers are more likely to churn

```
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)
```

This code creates a heatmap of the correlation matrix for a DataFrame df, where categorical variables are encoded using pd.factorize() before calculating the correlations. It applies a mask to the upper triangle of the correlation matrix to display only the lower triangle, with visual customization such as annotation, a color map, and axis labels.



7. *DATA PREPROCESSING*

```
# Splitting the data into train and test sets

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

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

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	St
0	5375	0	0	1	0	1	0	1	0	0	...	0	0	
1	3962	1	0	0	0	34	1	0	0	2	...	2	0	
2	2564	1	0	0	0	2	1	0	0	2	...	0	0	
3	5535	1	0	0	0	45	0	1	0	2	...	2	2	
4	6511	0	0	0	0	2	1	0	1	0	...	0	0	

5 rows × 21 columns

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

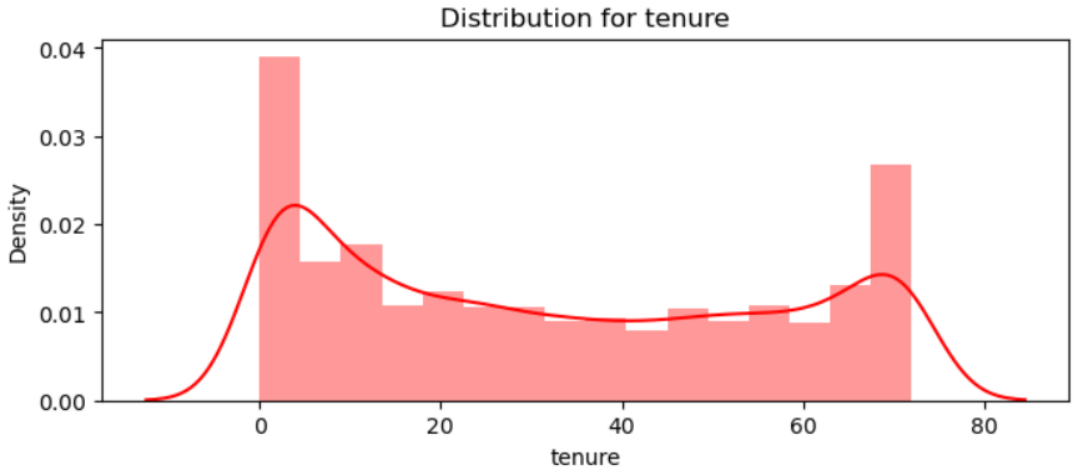
Churn 1.000000
MonthlyCharges 0.193356
PaperlessBilling 0.191825
SeniorCitizen 0.150889
PaymentMethod 0.107062

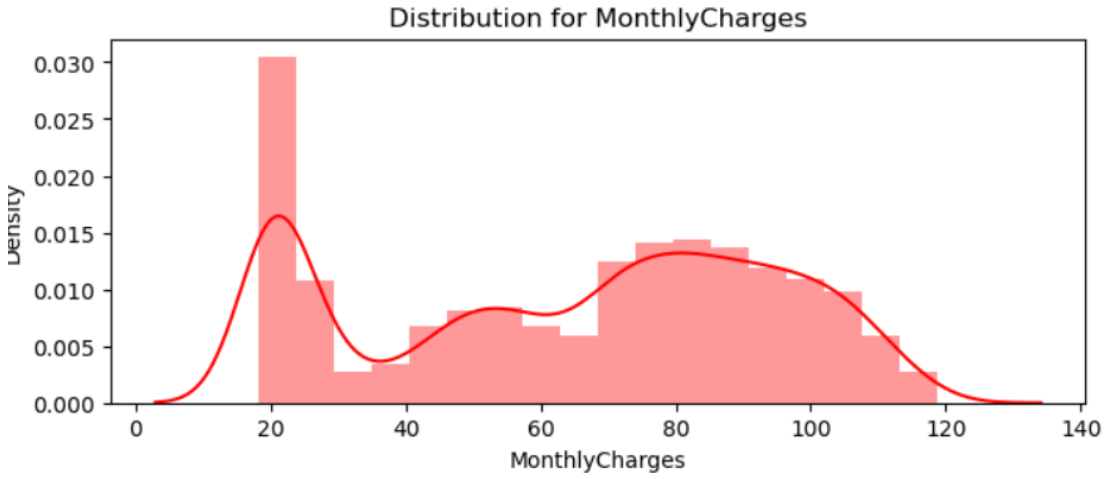
```
MultipleLines       0.038037
TotalCharges        0.014479
PhoneService        0.011942
gender             -0.008612
customerID         -0.017447
StreamingTV        -0.036581
StreamingMovies    -0.038492
InternetService    -0.047291
Partner            -0.150448
Dependents         -0.164221
DeviceProtection   -0.178134
OnlineBackup       -0.195525
TechSupport        -0.282492
OnlineSecurity     -0.289309
tenure             -0.352229
Contract           -0.396713
Name: Churn, dtype: float64
<Figure size 1400x700 with 0 Axes>
```

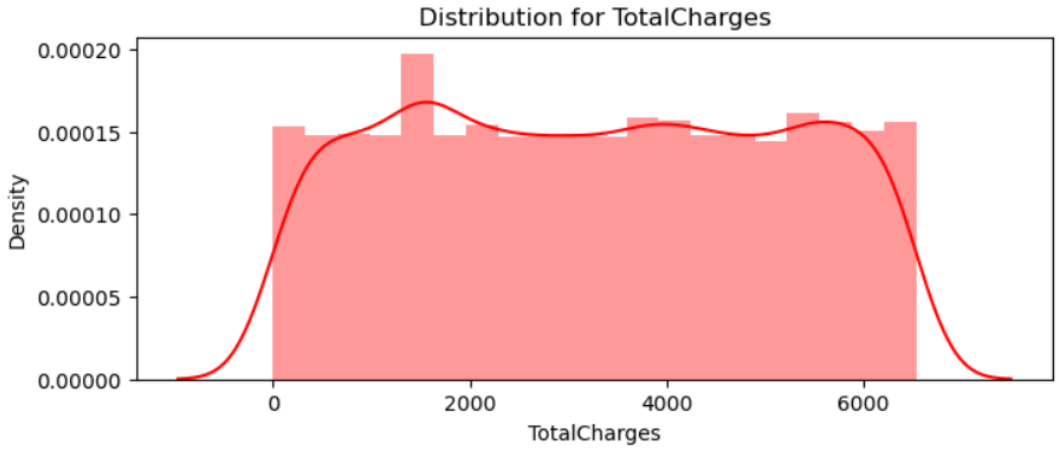
```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values
```

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

```
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,3))
    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)
```

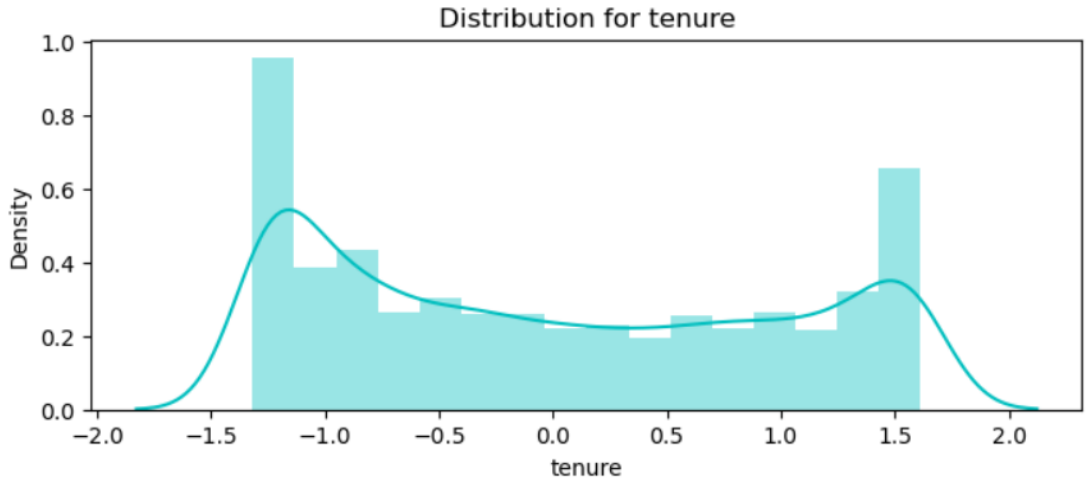


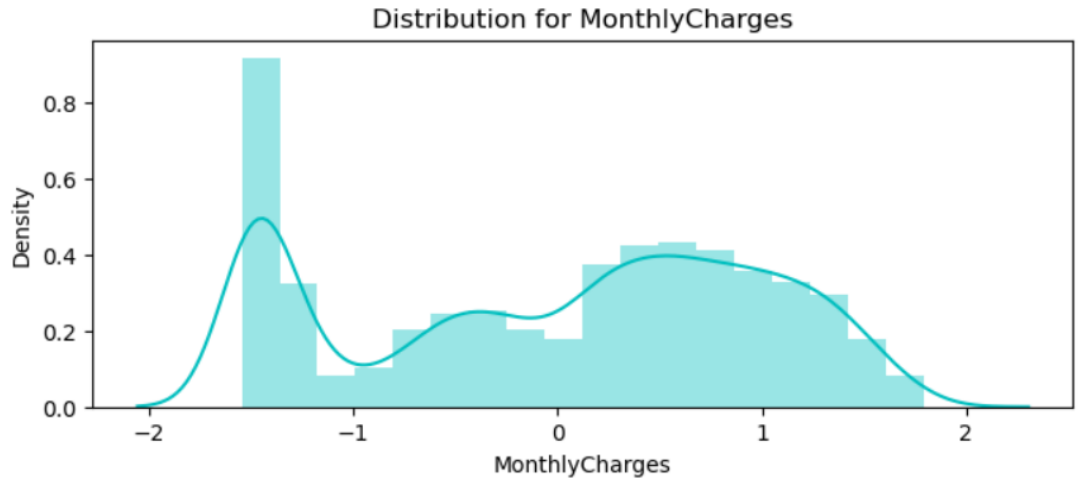


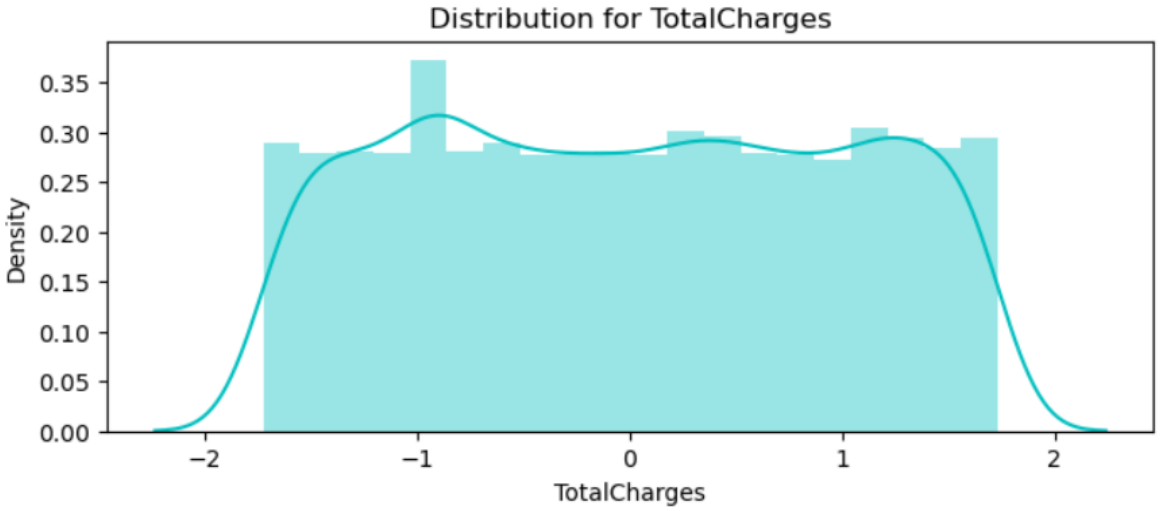


Since the numerical features are distributed over different value ranges, I will use standard scalar to scale them down to the same range.

```
##Standardizing numeric attributes

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

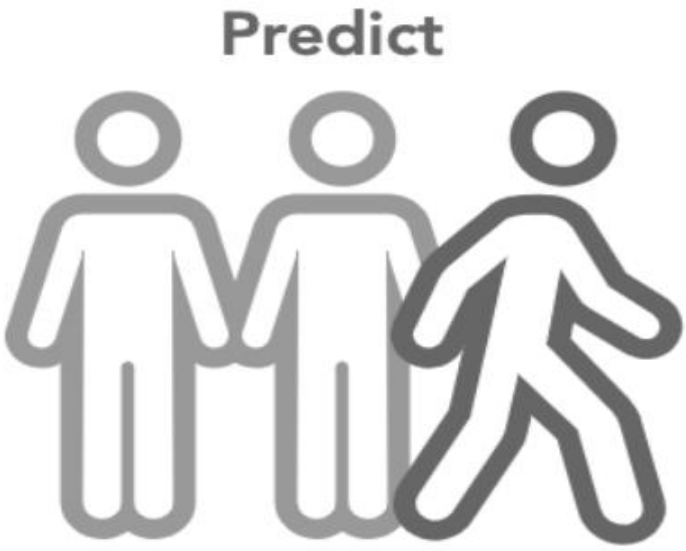







```
# Divide the columns into 3 categories, one ofor standardisation, one for label encoding and one for one hot encoding

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])
```

8. *MACHINE LEARNING MODEL EVALUATIONS AND PREDICTIONS*



KNN

K-Nearest Neighbors (KNN) is a supervised machine learning algorithm used for classification and regression. It works by finding the 'k' nearest data points to a given test sample and making predictions based on the majority class (for classification) or average value (for regression) of those neighbors. KNN is simple, intuitive, and doesn't require explicit training, but its performance can degrade with high-dimensional data. The choice of 'k' and distance metric significantly impacts its effectiveness.

```
knn_model = KNeighborsClassifier(n_neighbors = 11)
knn_model.fit(X_train,y_train)
predicted_y = knn_model.predict(X_test)
accuracy_knn = knn_model.score(X_test,y_test)
print("KNN accuracy:",accuracy_knn)
```

KNN accuracy: 0.7169900615238997

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

	precision	recall	f1-score	support
0	0.74	0.95	0.83	1552
1	0.34	0.07	0.12	561
accuracy			0.72	2113
macro avg	0.54	0.51	0.47	2113
weighted avg	0.63	0.72	0.64	2113

SVC

Support Vector Classification (SVC) is a supervised machine learning algorithm used for classification tasks. It works by finding a hyperplane that best separates different classes in the feature space, maximizing the margin between the closest points of each class (support vectors). SVC can be used with linear and non-linear decision boundaries by applying kernel functions such as linear, polynomial, or radial basis function (RBF). It's effective in high-dimensional spaces but can be computationally expensive for large datasets.

```
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)
```

SVM accuracy is : 0.73450070989115

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

	precision	recall	f1-score	support
0	0.73	1.00	0.85	1552
1	0.00	0.00	0.00	561
accuracy			0.73	2113
macro avg	0.37	0.50	0.42	2113
weighted avg	0.54	0.73	0.62	2113

Random Forest

Random Forest is an ensemble learning algorithm used for both classification and regression tasks. It constructs multiple decision trees during training and outputs the majority vote (for classification) or average prediction (for regression) from all the trees. Random Forest reduces overfitting by averaging the results of several trees, each trained on random subsets of the data and features, leading to better generalization. It is robust, handles missing values well, and provides feature importance but can be computationally intensive with large datasets.

```
model_rf = RandomForestClassifier(
    n_estimators=500,
    oob_score=True,
    n_jobs=-1,
    random_state=50,
    max_features="sqrt",  # Replace 'auto' with 'sqrt'
    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))
```

0.8002839564600095

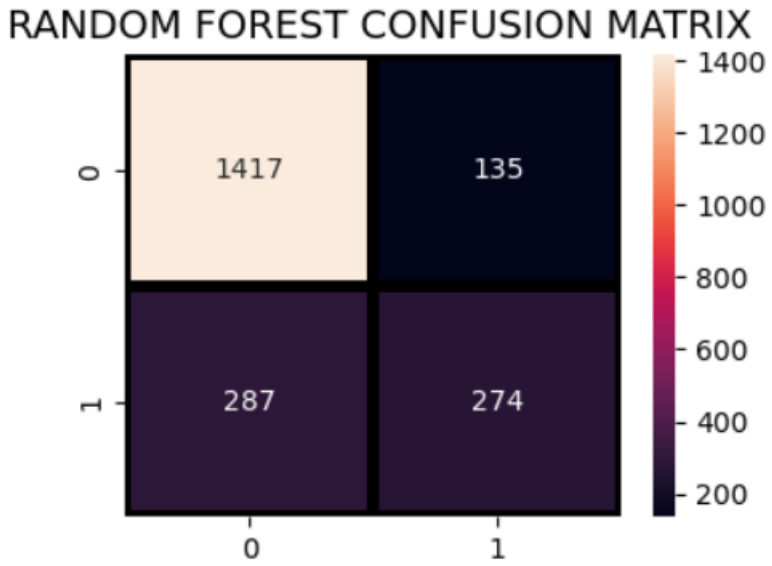
```
print(classification_report(y_test, prediction_test))
```

	precision	recall	f1-score	support
0	0.83	0.91	0.87	1552
1	0.67	0.49	0.56	561
accuracy			0.80	2113
macro avg	0.75	0.70	0.72	2113
weighted avg	0.79	0.80	0.79	2113

```
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()
```

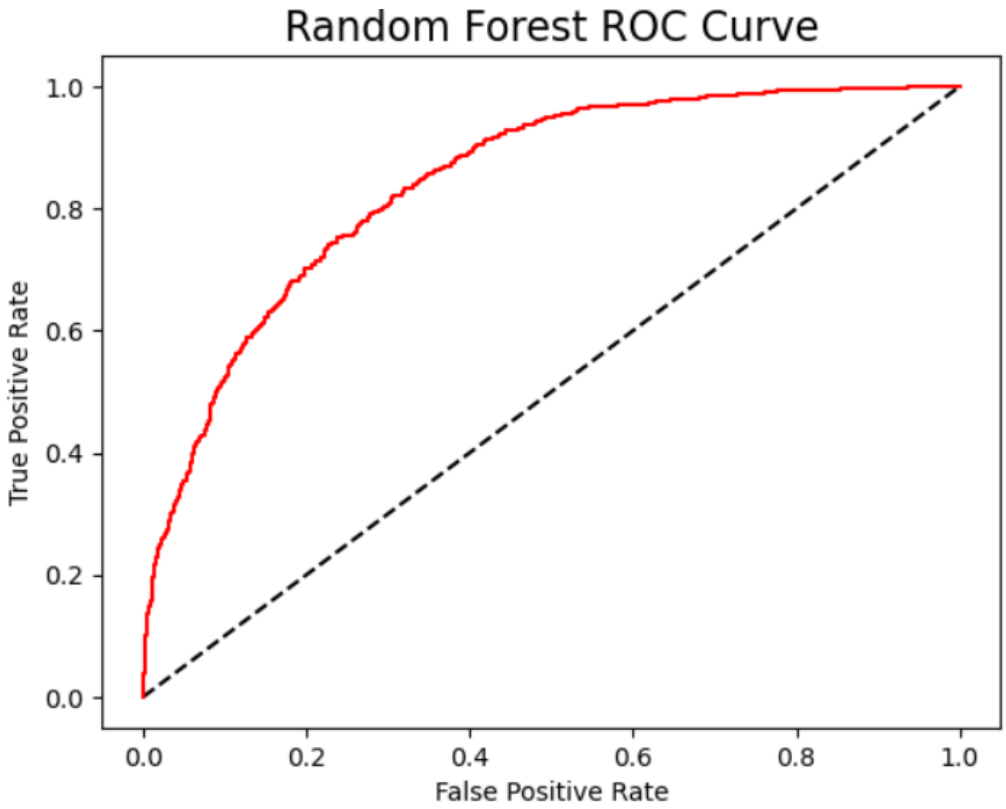
This code generates and displays a heatmap of the confusion matrix for the Random Forest model. It uses confusion_matrix(y_test, prediction_test) to calculate the confusion matrix between the true labels (y_test) and the predicted labels (prediction_test). The heatmap is created using Seaborn (sns.heatmap), with annotations to display the values in each cell (annot=True), formatting the values as integers (fmt="d"), and setting the line color and width

for cell separation (linecolor="k", linewidths=3). The plot is displayed with a title, "RANDOM FOREST CONFUSION MATRIX", and a figure size of (4, 3).




```
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();
```

This code generates the ROC (Receiver Operating Characteristic) curve for a Random Forest model. It first calculates the predicted probabilities of the positive class (y_rfpred_prob) using model_rf.predict_proba(X_test)[:, 1], where the second column contains the probability of the positive class. Then, it computes the false positive rate (fpr_rf) and true positive rate (tpr_rf) using the roc_curve() function. The ROC curve is plotted with the false positive rate on the x-axis and the true positive rate on the y-axis, comparing the model's performance against a random classifier (represented by the diagonal line plt.plot([0, 1], [0, 1], 'k--')). The curve is displayed in red (color = "r") and labeled "Random Forest," with axis labels and a title for the plot.



Logistic Regression

Logistic Regression is a statistical method used for binary classification tasks, where the output is a probability that is mapped to two classes using a logistic function (sigmoid). It models the relationship between independent variables and the probability of a binary outcome by fitting a linear equation to the data, then applying the sigmoid function to predict the probability of the target class. Although it's simple and interpretable, it assumes a linear relationship between the features and the log-odds of the outcome. Logistic regression is efficient for smaller datasets but may struggle with complex relationships or non-linear data.

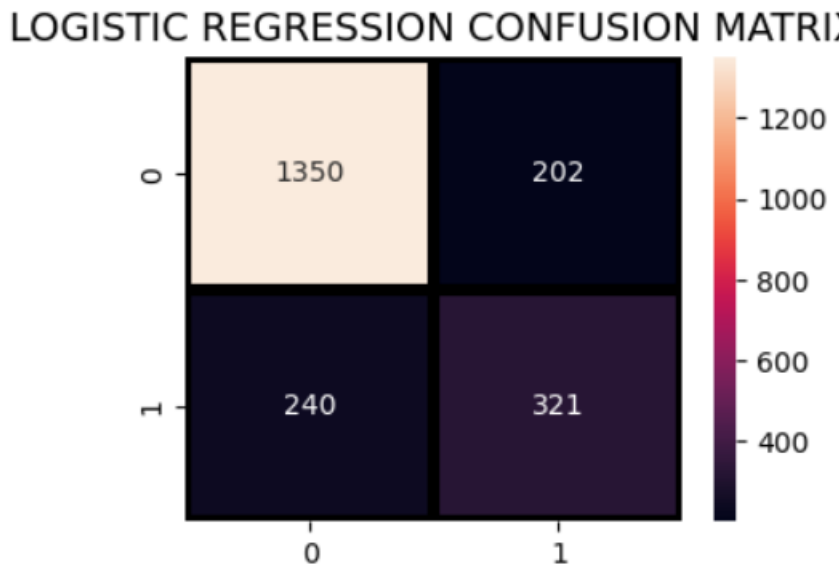
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

Logistic Regression accuracy is : 0.7908187411263606

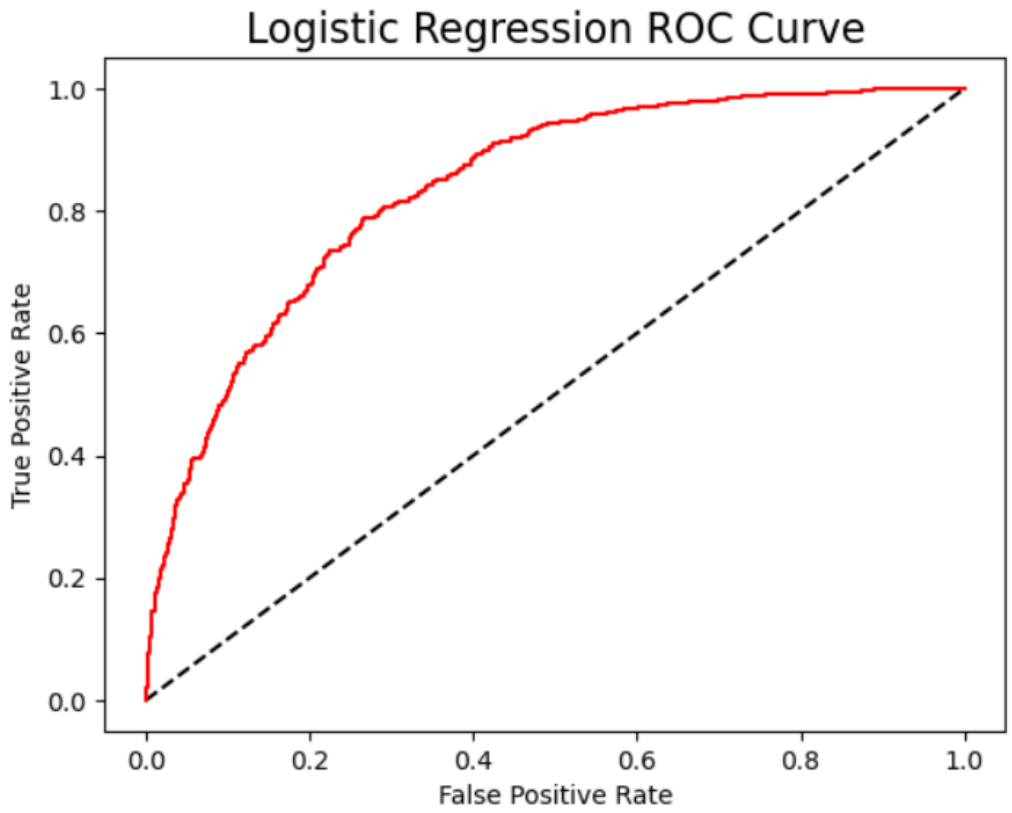
```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.85      0.87      0.86      1552
           1       0.61      0.57      0.59       561

    accuracy                           0.79      2113
   macro avg       0.73      0.72      0.73      2113
weighted avg       0.79      0.79      0.79      2113
```

```
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()
```



```
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();
```



Decision Tree Classifier

A Decision Tree Classifier is a supervised machine learning algorithm used for classification tasks. It splits the data into subsets based on feature values, creating a tree-like model of decisions and their possible consequences. Each node represents a feature decision, and branches represent the outcomes, ultimately leading to a class label at the leaves. Decision trees are easy to interpret and handle both numerical and categorical data, but they can be prone to overfitting, especially with complex trees, which can be mitigated by pruning or using ensemble methods like Random Forest.

```
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)
```

Decision Tree accuracy is : 0.7236157122574538

```
###Decision tree gives very low score.

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

	precision	recall	f1-score	support
0	0.82	0.80	0.81	1552
1	0.48	0.52	0.50	561
accuracy			0.72	2113
macro avg	0.65	0.66	0.66	2113
weighted avg	0.73	0.72	0.73	2113

AdaBoost Classifier

AdaBoost (Adaptive Boosting) is an ensemble learning algorithm that combines multiple weak learners, typically decision trees, to create a strong classifier. It works by training weak classifiers sequentially, with each subsequent model focusing more on the misclassified data points from the previous model, thus "boosting" the accuracy. AdaBoost assigns higher weights to the misclassified instances to improve the model's performance. It is effective for improving the accuracy of weak classifiers but can be sensitive to noisy data and outliers.

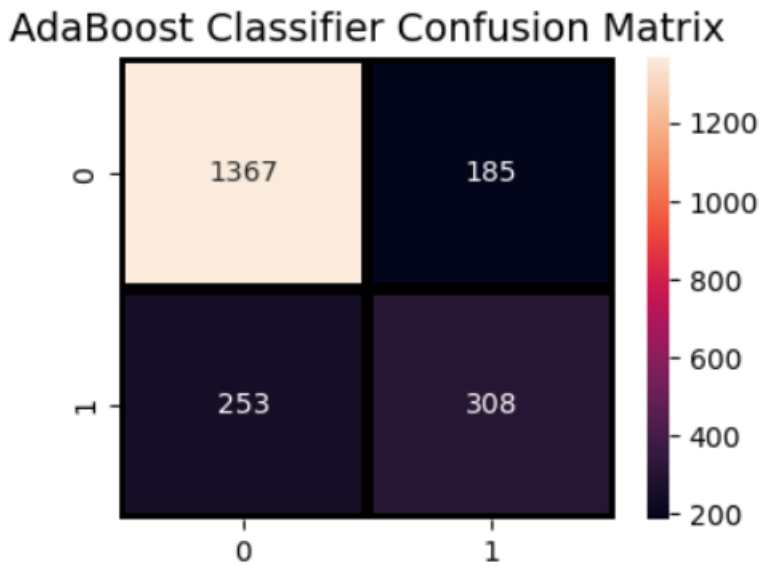
```
a_model = AdaBoostClassifier()
a_model.fit(X_train,y_train)
a_preds = a_model.predict(X_test)
print("AdaBoost Classifier accuracy")
metrics.accuracy_score(y_test, a_preds)
```

AdaBoost Classifier accuracy
0.7927117841930904

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

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86      1552
           1       0.62      0.55      0.58       561

    accuracy                           0.79      2113
   macro avg       0.73      0.71      0.72      2113
weighted avg       0.79      0.79      0.79      2113
```

```
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()
```



Gradient Boosting Classifier

Gradient Boosting Classifier is an ensemble learning algorithm that builds a model by combining multiple weak learners (typically decision trees) in a sequential manner. Each new model corrects the errors made by the previous one by focusing on the residuals (differences between the predicted and actual values), thus "boosting" the overall performance. Gradient boosting minimizes a loss function through gradient descent, making it highly effective for

complex problems with high accuracy. It is powerful but can be prone to overfitting if the model is too complex or the number of trees is too large.

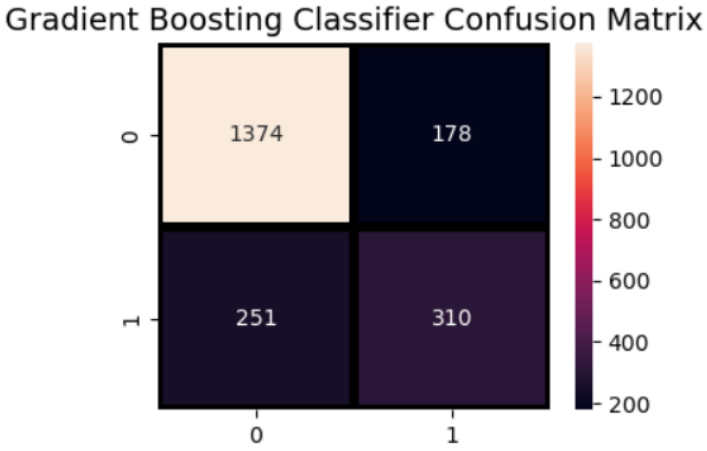
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
gb_pred = gb.predict(X_test)
print("Gradient Boosting Classifier", accuracy_score(y_test, gb_pred))
```

Gradient Boosting Classifier 0.7969711310932324

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

	precision	recall	f1-score	support
0	0.85	0.89	0.86	1552
1	0.64	0.55	0.59	561
accuracy			0.80	2113
macro avg	0.74	0.72	0.73	2113
weighted avg	0.79	0.80	0.79	2113

```
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()
```



Voting Classifier Let's now predict the final model based on the highest majority of voting and check it's score.

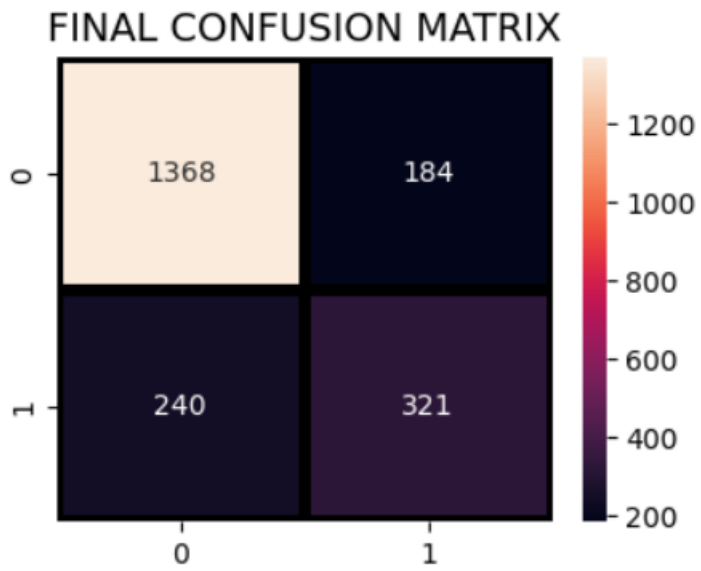
```
from sklearn.ensemble import VotingClassifier
clf1 = GradientBoostingClassifier()
clf2 = LogisticRegression()
clf3 = AdaBoostClassifier()
eclf1 = VotingClassifier(estimators=[('gbc', clf1), ('lr', clf2), ('abc', clf3)], voting='soft')
eclf1.fit(X_train, y_train)
predictions = eclf1.predict(X_test)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

Final Accuracy Score
0.7993374349266446

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

	precision	recall	f1-score	support
0	0.85	0.88	0.87	1552
1	0.64	0.57	0.60	561
accuracy			0.80	2113
macro avg	0.74	0.73	0.73	2113
weighted avg	0.79	0.80	0.80	2113

```
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()
```



In the confusion matrix, the actual values are compared to the predicted values made by the model. For non-churn values:

- There are 1549 actual non-churn cases (1400 predicted as non-churn and 149 predicted as churn). This means the model correctly predicts most of the non-churn customers but misidentifies 149 as churn. For churn values:
- There are 561 actual churn cases (237 predicted as non-churn and 324 predicted as churn). This means the model correctly identifies 324 churn customers, but it incorrectly predicts 237 churn customers as non-churn.

Customer churn can significantly impact a company's profitability. To reduce churn, companies should focus on understanding their customers and identifying those at risk of leaving. By improving customer satisfaction, offering personalized experiences, and enhancing customer service, companies can retain customers. Additionally, proactively surveying customers who have already churned can provide valuable insights to prevent further loss of customers in the future.

Project_PPT_Rahul Singh.pptx