```
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, classification_report
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
```

```
pip install catboost
```

```
Requirement already satisfied: catboost in /usr/local/lib/python3.11/dist-packages (1.2.8)
Requirement already satisfied: graphviz in /usr/local/lib/python3.11/dist-packages (from catboost) (0.21)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from catboost) (3.10.0)
Requirement already satisfied: numpy<3.0,>=1.16.0 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.0.2)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.11/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from catboost) (1.15.3)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from catboost) (1.17.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboos
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025.2
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas>=0.24->catboost) (2025
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.3.
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (4.5
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (1.4
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->catboost) (3.2.
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->catboost) (8.5.0)
```

loading data

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

3. Undertanding the data

Each row represents a customer, each column contains customer’s attributes described on the column Metadata.

```
df.head()
```

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

5 rows × 21 columns

The data set includes information about:

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

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

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

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

```
df.shape
```

```
(7043, 21)
```

```
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.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)
```

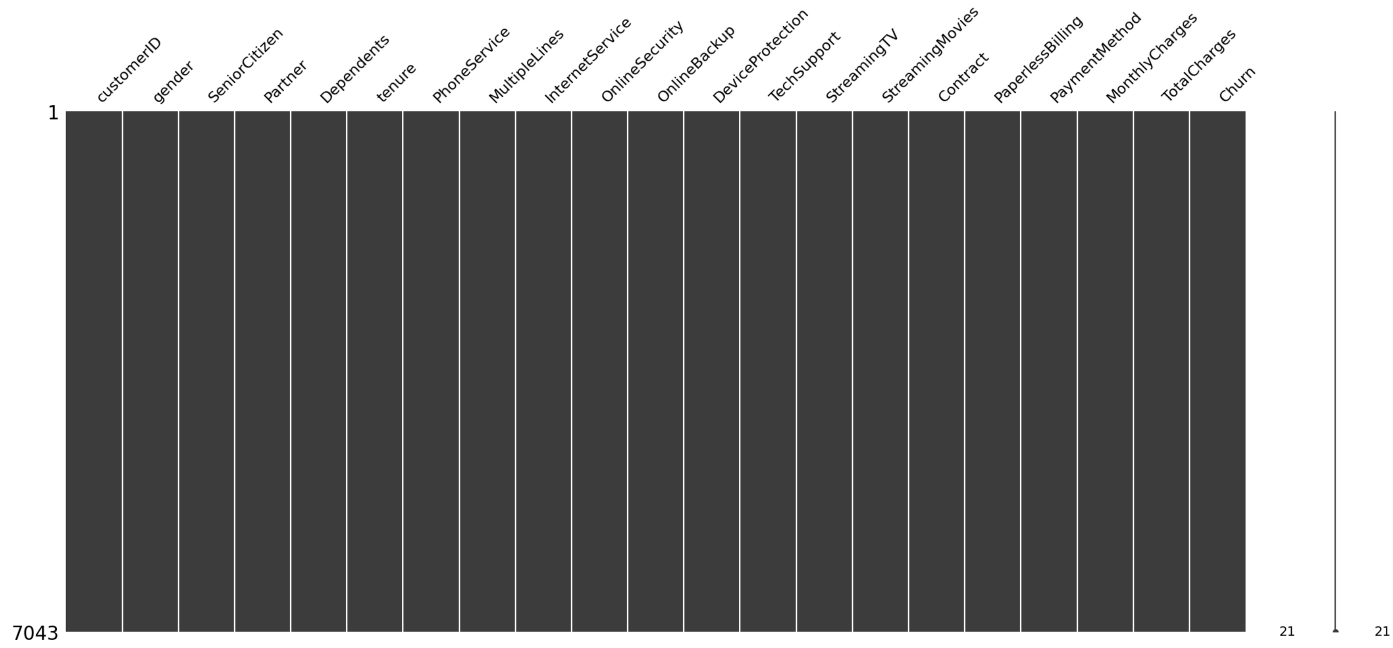
```
df.dtypes
```

	0
customerID	object
gender	object
SeniorCitizen	int64
Partner	object
Dependents	object
tenure	int64
PhoneService	object
MultipleLines	object
InternetService	object
OnlineSecurity	object
OnlineBackup	object
DeviceProtection	object
TechSupport	object
StreamingTV	object
StreamingMovies	object
Contract	object
PaperlessBilling	object
PaymentMethod	object
MonthlyCharges	float64
TotalCharges	object
Churn	object

dtype: object

4. Visualize missing values

```
msno.matrix(df);
```



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()
```

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

On deep analysis, we can find some indirect missingness in our data (which can be in form of blankspaces). Let's see that!

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

	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	11
Churn	0

dtype: int64

Here we see that the TotalCharges has 11 missing values. Let's check this data.

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

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

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

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
```

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

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())
```

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineB
0	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	Male	0	No	No	34	Yes	No	DSL	Yes	
2	Male	0	No	No	2	Yes	No	DSL	Yes	
3	Male	0	No	No	45	No	No phone service	DSL	Yes	
4	Female	0	No	No	2	Yes	No	Fiber optic	No	
...	...	...	...	...	...	...	...	...	...	
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	
7041	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	
7042	Male	0	No	No	66	Yes	No	Fiber optic	Yes	

7032 rows × 20 columns

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

	0
gender	0
SeniorCitizen	0
Partner	0
Dependents	0
tenure	0
PhoneService	0
MultipleLines	0
InternetService	0
OnlineSecurity	0
OnlineBackup	0
DeviceProtection	0
TechSupport	0
StreamingTV	0
StreamingMovies	0
Contract	0
PaperlessBilling	0
PaymentMethod	0
MonthlyCharges	0
TotalCharges	0
Churn	0

dtype: int64

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

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

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

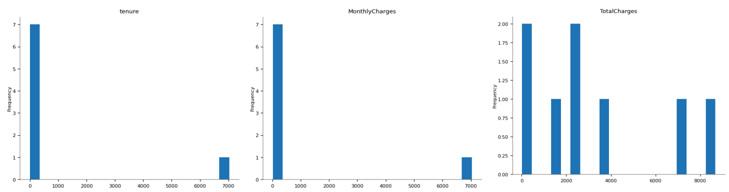
	InternetService
count	7032
unique	3
top	Fiber optic
freq	3096

dtype: object

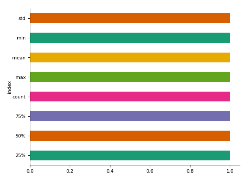
```
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_cols].describe()
```

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

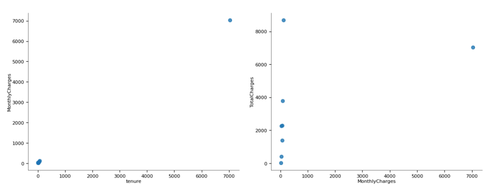
Distributions



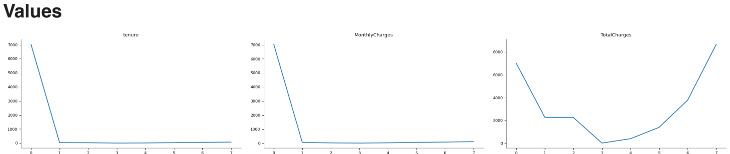
Categorical distributions



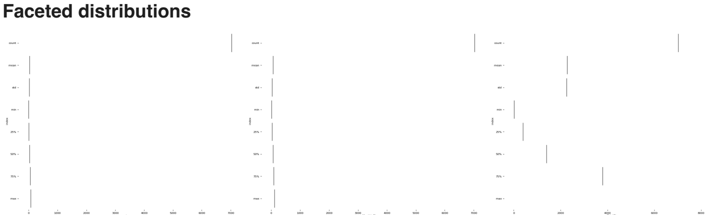
2-d distributions



Values



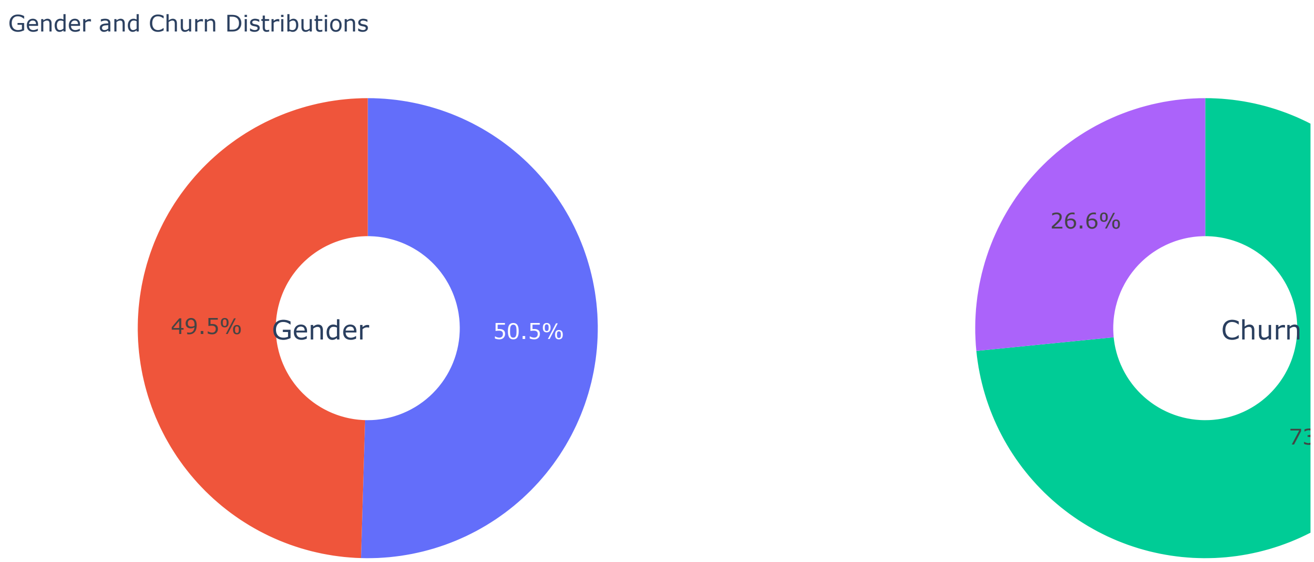
Faceted distributions



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()
```



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	Churn
Female	2544
Male	2619

dtype: int64

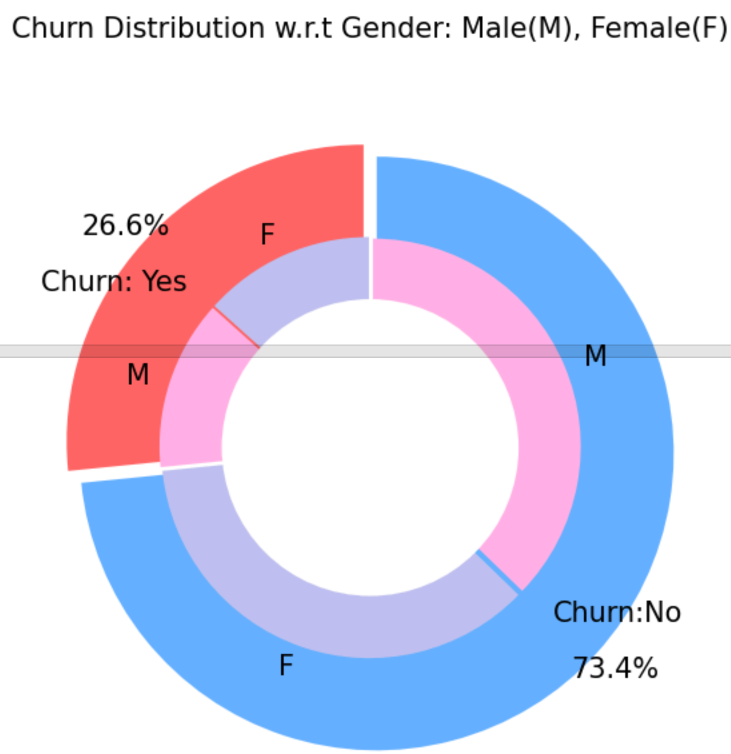
```
df["Churn"][df["Churn"]=="Yes"].groupby(by=df["gender"]).count()
```

gender	Churn
Female	939
Male	930

dtype: int64

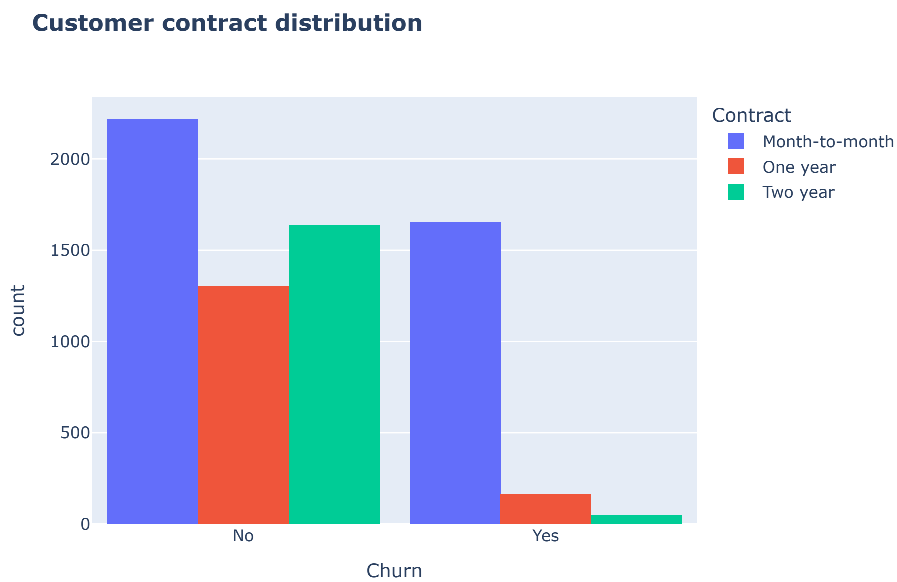
```
plt.figure(figsize=(6, 6))
labels =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619]
colors = ['#ff6666', '#66b3ff']
colors_gender = ['#c2c2f0','#ffb3e6', '#c2c2f0','#ffb3e6']
explode = (0.3,0.3)
explode_gender = (0.1,0.1,0.1,0.1)
textprops = {"fontsize":15}
#Plot
plt.pie(values, labels=labels,autopct='%1.1f%%',pctdistance=1.08, labeldistance=0.8,colors=colors, startangle=90,frame=True, exp
plt.pie(sizes_gender,labels=labels_gender,colors=colors_gender,startangle=90, explode=explode_gender,radius=7, textprops =textpr
#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()
```



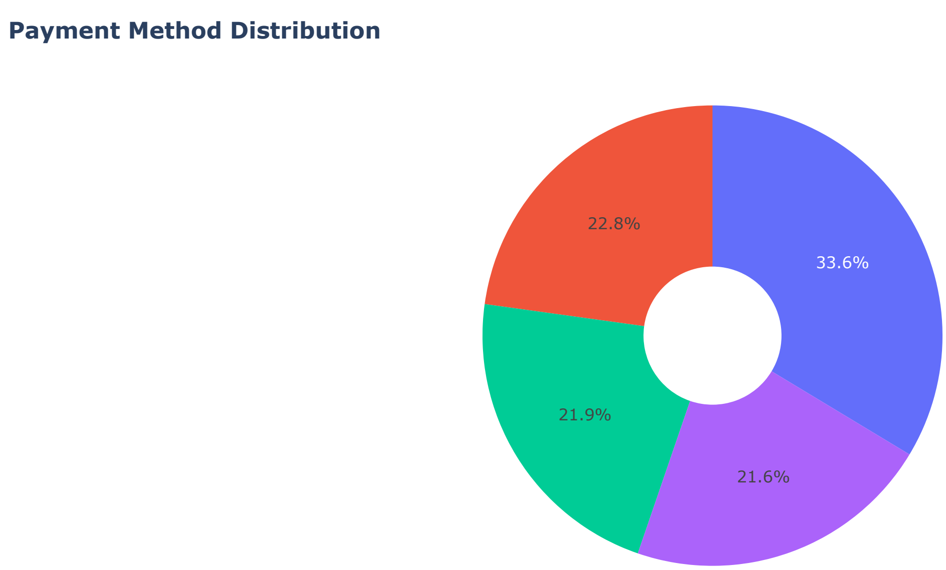
There is negligible difference in customer percentage/ count who chnaged the service provider. Both genders behaved in similar fashion when it comes to migrating to another service provider/firm.

```
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()
```

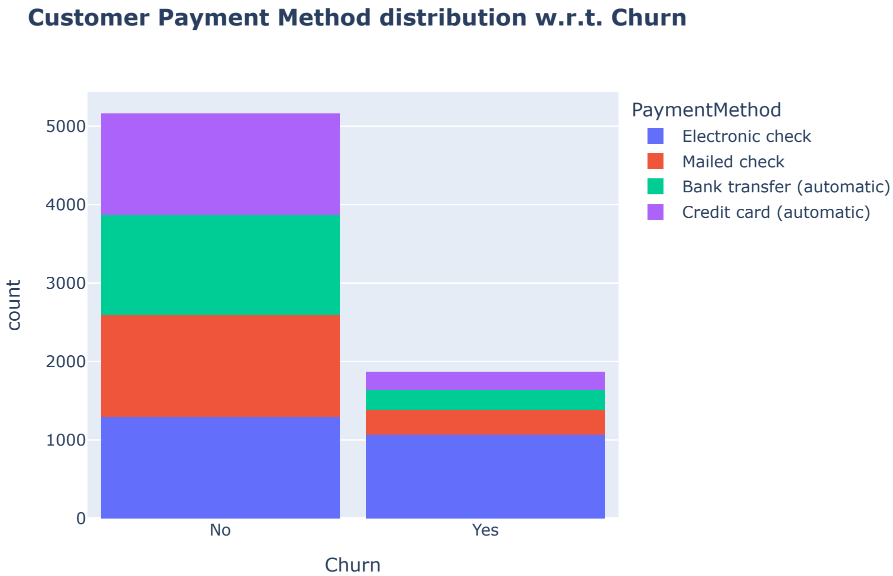


About 75% of customer with Month-to-Month Contract opted to move out as compared to 13% of customrs with One Year Contract and 3% with Two Year Contract

```
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	count
DSL	No	992
Fiber optic	No	910
No	No	717
Fiber optic	Yes	633
DSL	Yes	240
No	Yes	57

dtype: int64

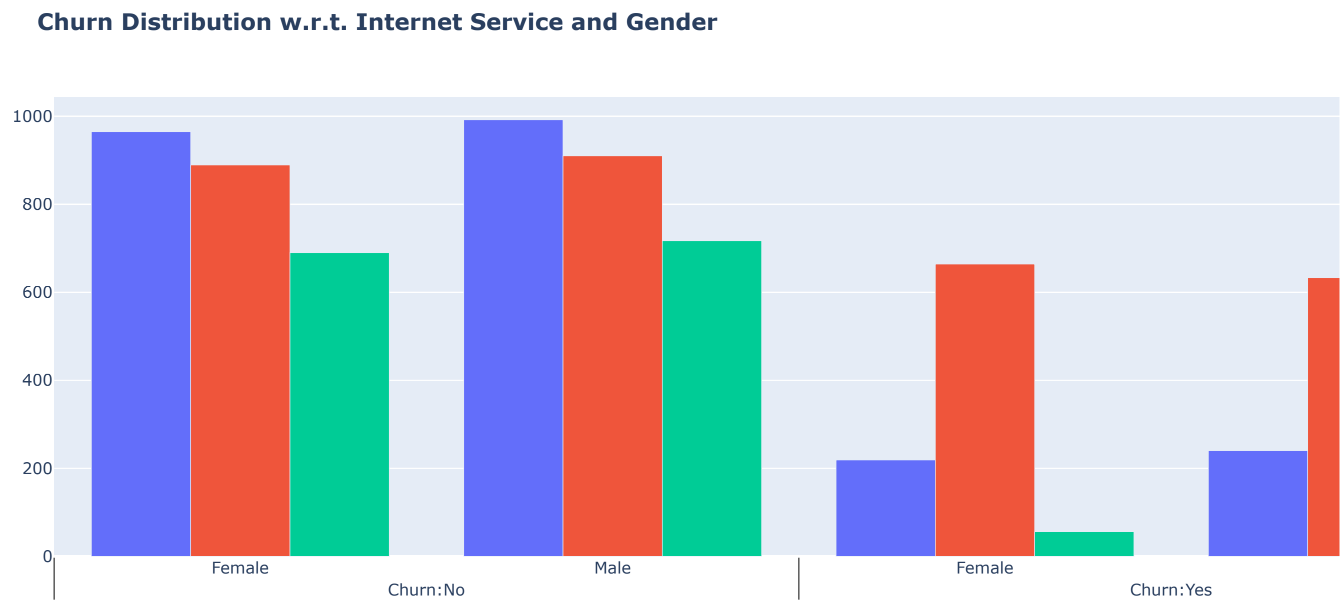
```
df[df["gender"]=="Female"][["InternetService", "Churn"]].value_counts()
```

InternetService	Churn	count
DSL	No	965
Fiber optic	No	889
No	No	690
Fiber optic	Yes	664
DSL	Yes	219
No	Yes	56

dtype: int64

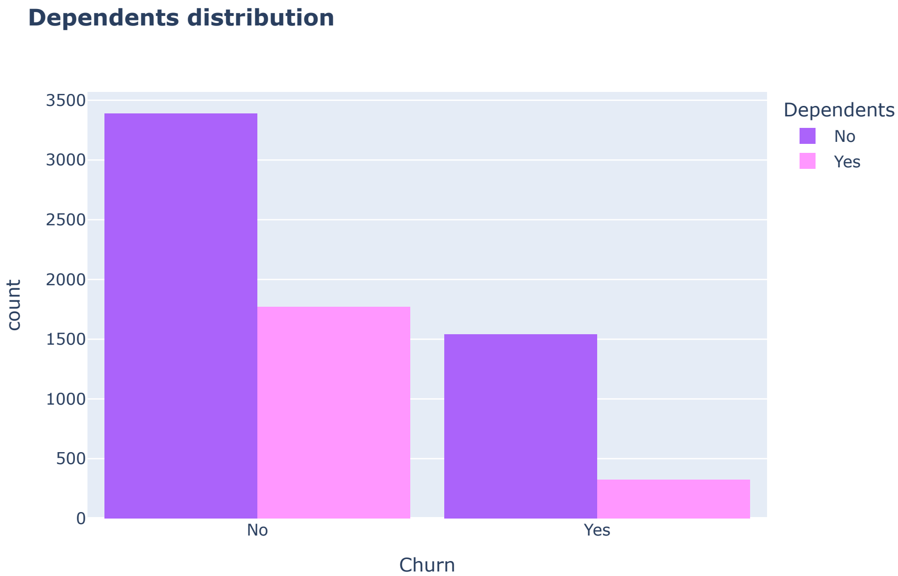
```
fig = go.Figure()
```

```
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()
```



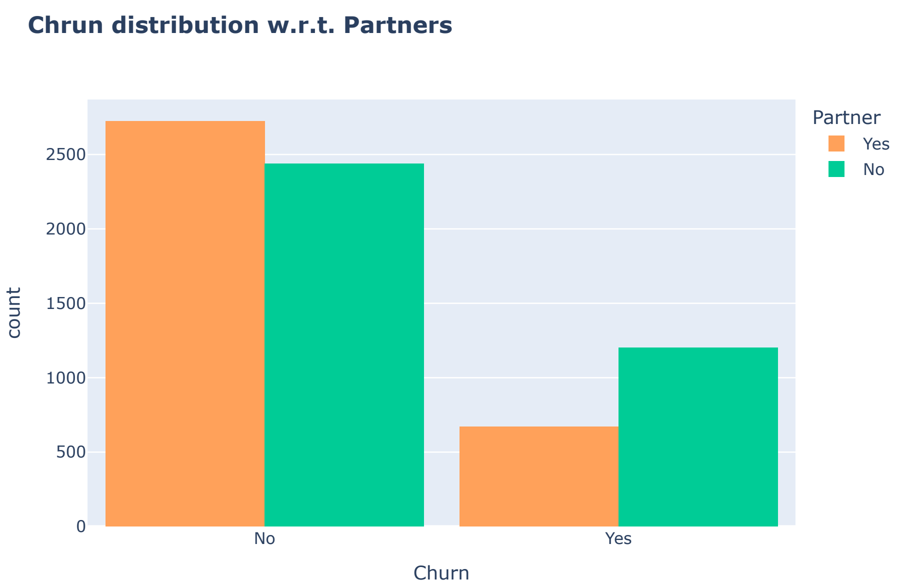
A lot of customers choose the Fiber optic service and it's also evident that the customers who use Fiber optic have high churn rate, this might suggest a dissatisfaction with this type of internet service. Customers having DSL service are majority in number and have less churn rate compared to Fibre 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_ma
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



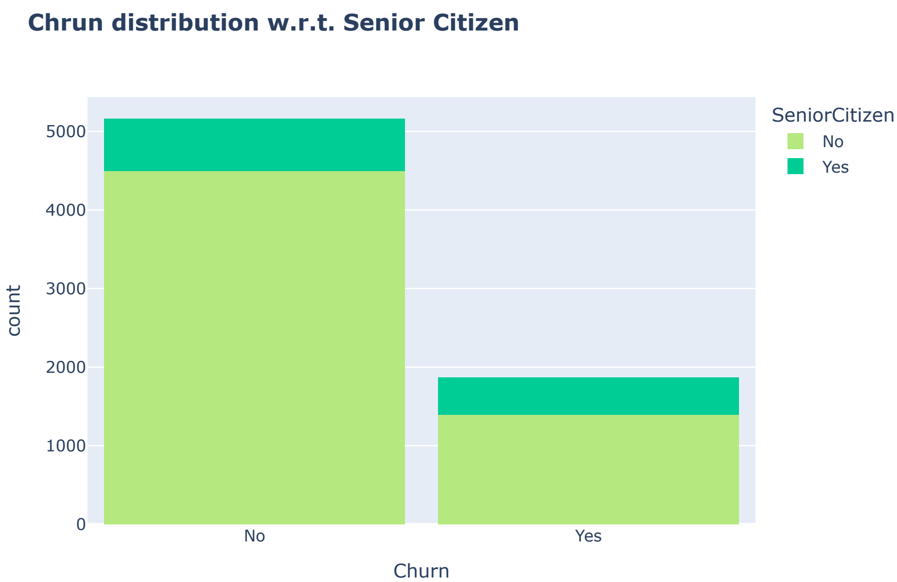
Customers without dependents are more likely to churn

```
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_dis
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



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

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

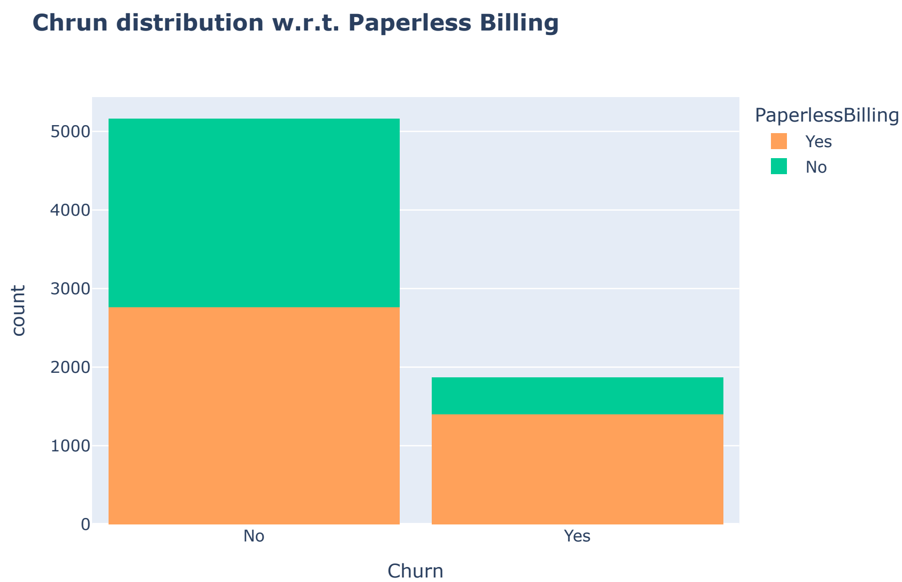


It can be observed that the fraction of senior citizen is very less. Most of the senior citizens churn.

```
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_dis
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

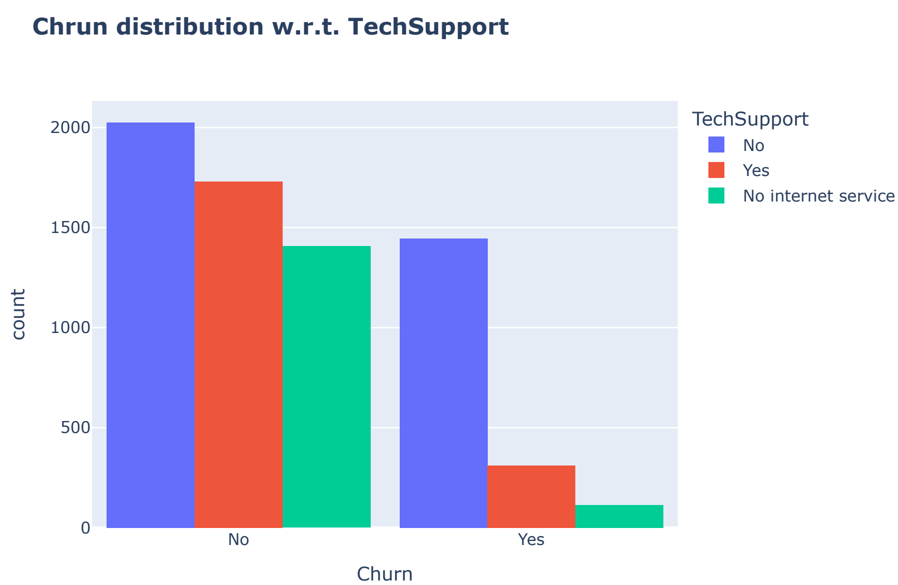
Most customers churn in the absence of online security,

```
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_c
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



Customers with Paperless Billing are most likely to churn.

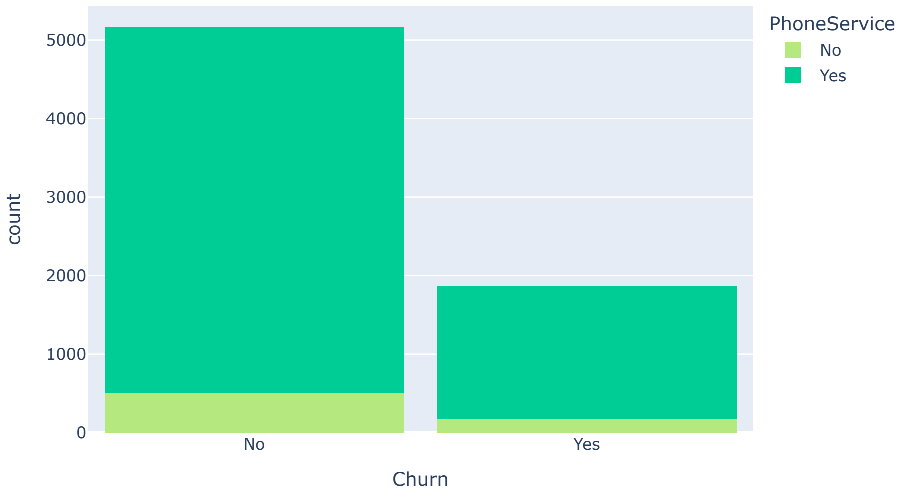
```
fig = px.histogram(df, x="Churn", color="TechSupport",barmode="group",  title="<b>Chrun distribution w.r.t. TechSupport</b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



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

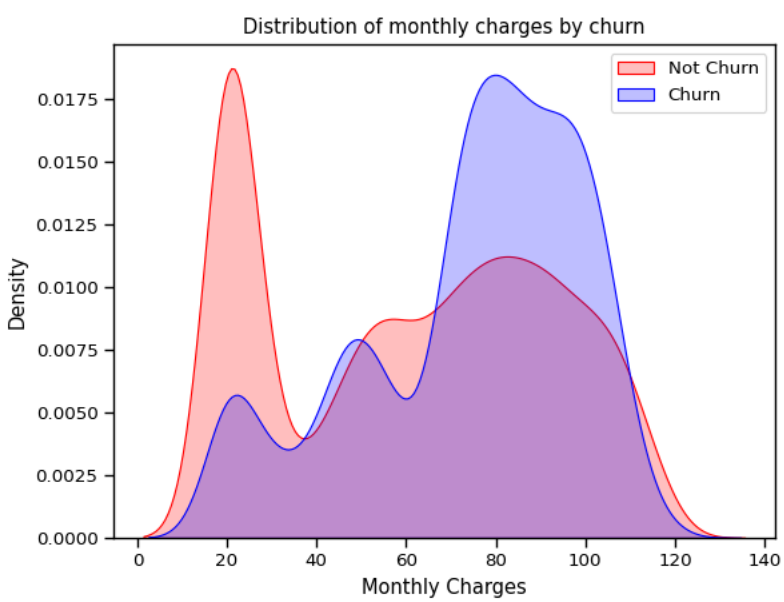
```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="PhoneService", title="<b>Chrun distribution w.r.t. Phone Service</b>", color_discrete_m
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

Chrun 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.

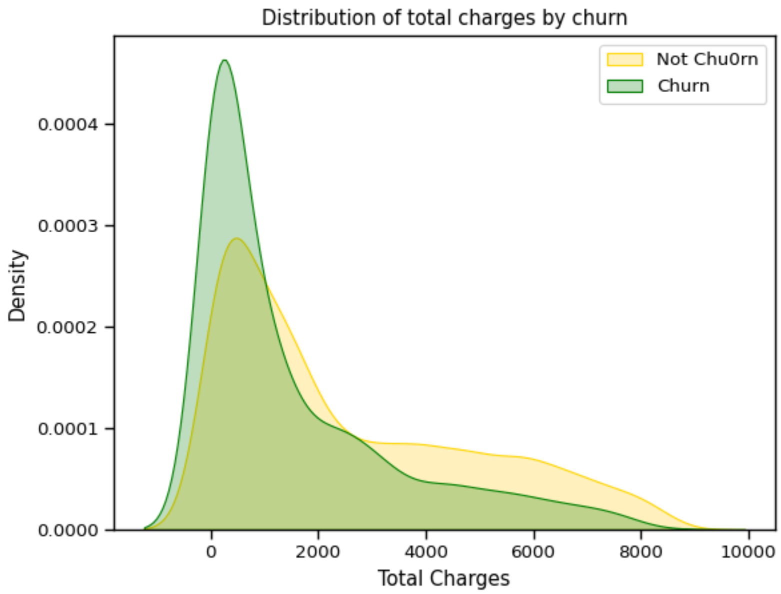
```
sns.set_context("paper",font_scale=1.1)
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'No') ],
                color="Red", shade = True);
ax = sns.kdeplot(df.MonthlyCharges[(df["Churn"] == 'Yes') ],
                ax =ax, color="Blue", shade= True);
ax.legend(["Not Churn","Churn"],loc='upper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



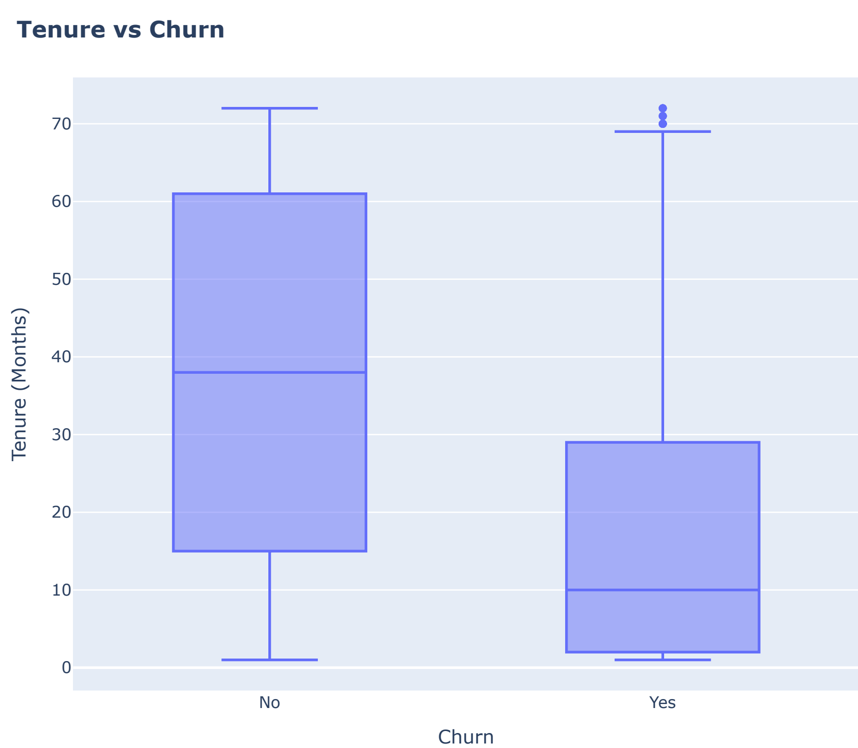
Customers with higher Monthly Charges are also more likely to churn

```
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');
```

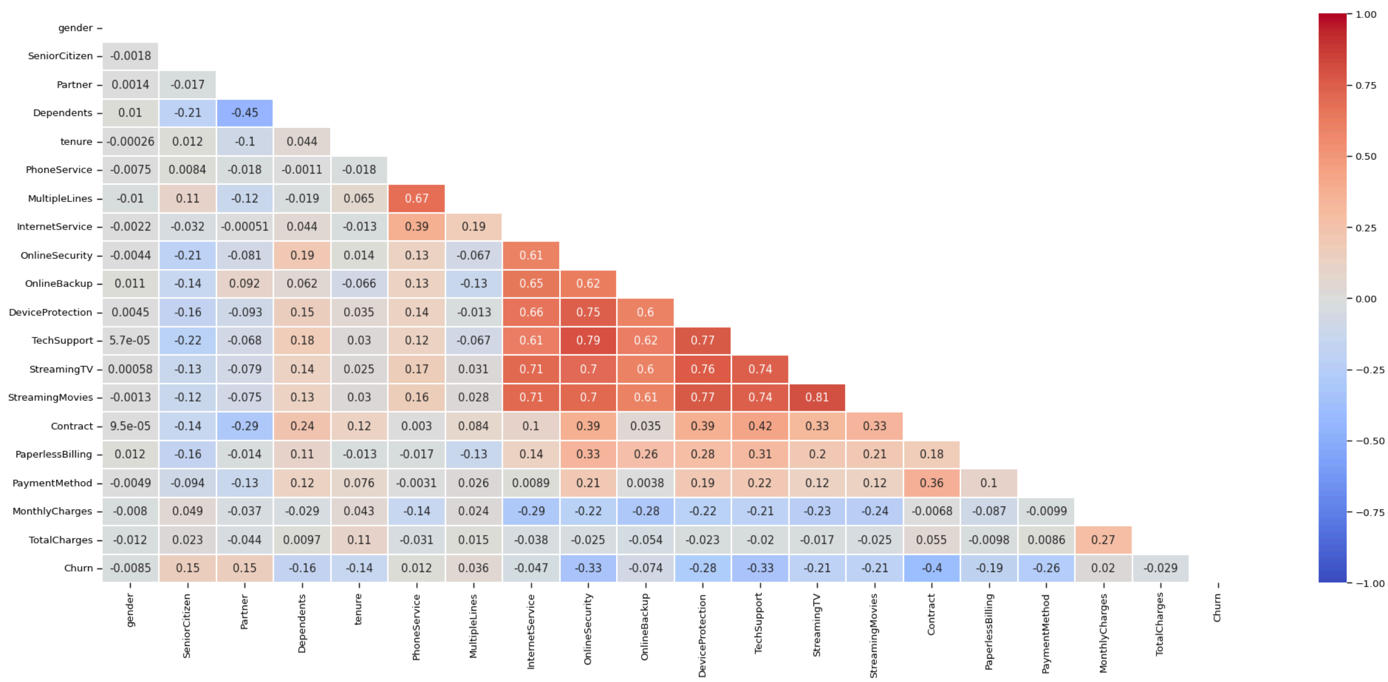



```
fig = px.box(df, x='Churn', y = 'tenure')

# 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()
```



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'
```



7. Data Preprocessing

Splitting the data into train and test sets

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

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

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBack
0	0	0	1	0	1	0	1	0	0	
1	1	0	0	0	34	1	0	0	2	
2	1	0	0	0	2	1	0	0	2	
3	1	0	0	0	45	0	1	0	2	
4	0	0	0	0	2	1	0	1	0	

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

	Churn
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

dtype: float64

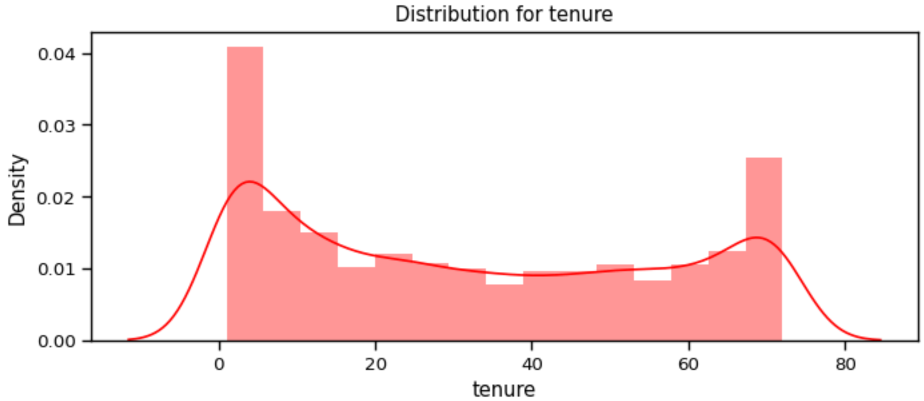
```
<Figure size 1400x700 with 0 Axes>
```

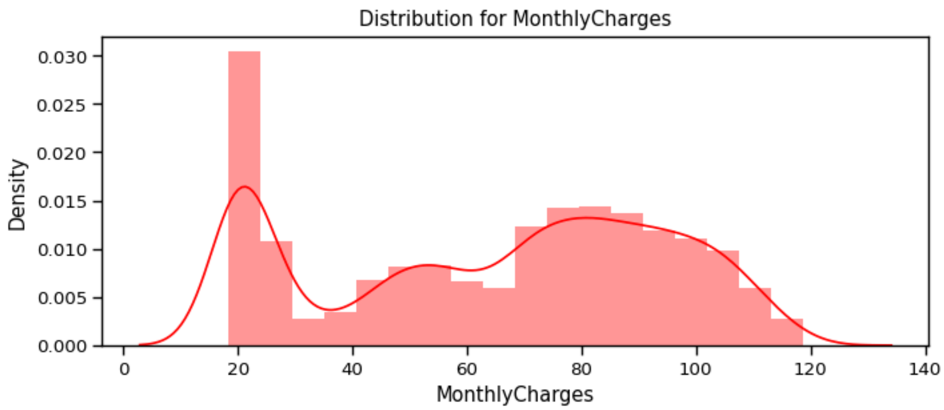
```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values
```

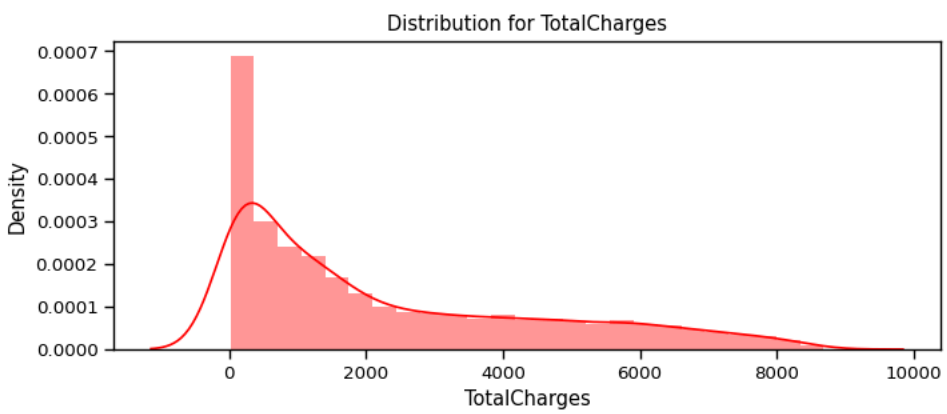
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.30, random_state = 40, stratify=y)
```

```
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)
```



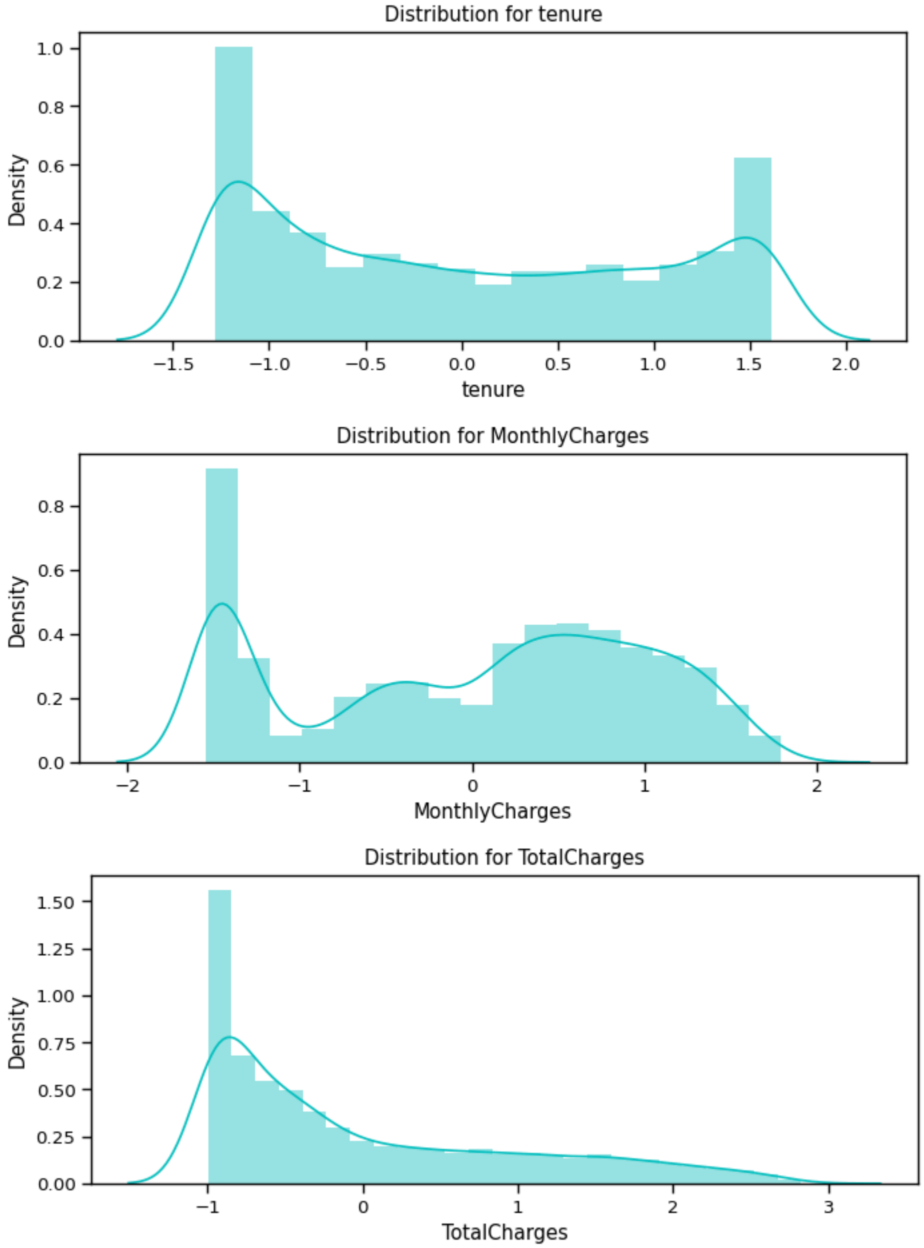




Since the numerical features are distributed over different value ranges, I will use standard scalar to scale them down to the same range.

Standardizing numeric attributes

```
df_std = pd.DataFrame(StandardScaler().fit_transform(df[num_cols].astype('float64')),
                       columns=num_cols)
for feat in numerical_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_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: 0.7758293838862559
```

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

```
              precision    recall  f1-score   support

           0       0.83      0.87      0.85      1549
           1       0.59      0.52      0.55       561

    accuracy                           0.78      2110
   macro avg       0.71      0.69      0.70      2110
weighted avg       0.77      0.78      0.77      2110
```

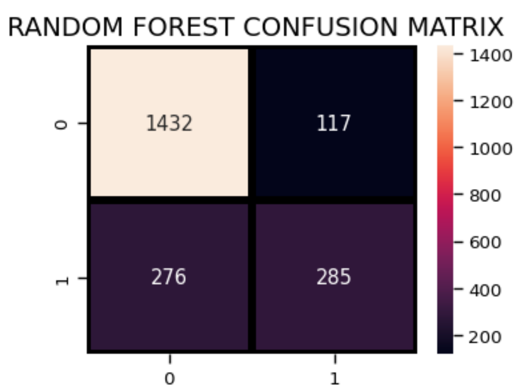
```
model_rf = RandomForestClassifier(n_estimators=500 , oob_score = True, n_jobs = -1,
                                  random_state =50, max_features = "sqrt",
                                  max_leaf_nodes = 30)
model_rf.fit(X_train, y_train)

# Make predictions
prediction_test = model_rf.predict(X_test)
print (metrics.accuracy_score(y_test, prediction_test))
```

```
0.8137440758293839
```

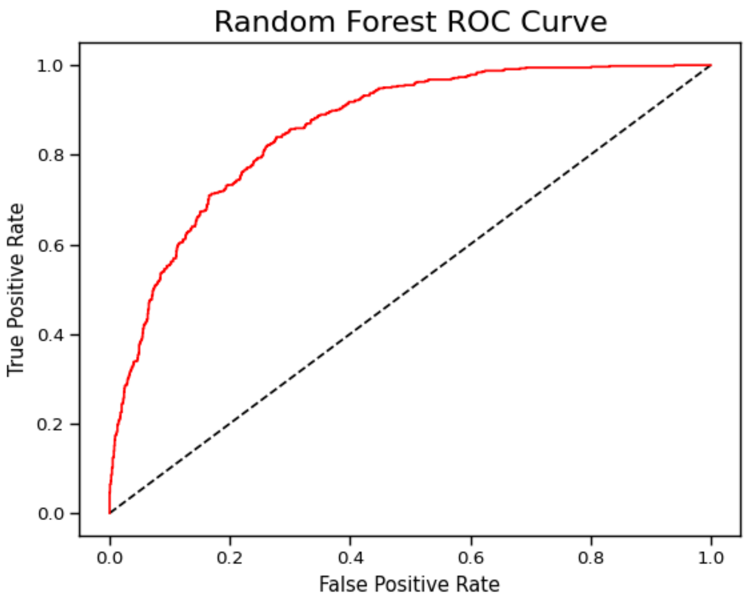
```
print(classification_report(y_test, prediction_test))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1549
           1       0.71      0.51      0.59       561

    accuracy                           0.81      2110
   macro avg       0.77      0.72      0.74      2110
weighted avg       0.80      0.81      0.80      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();
```



Logistic Regression

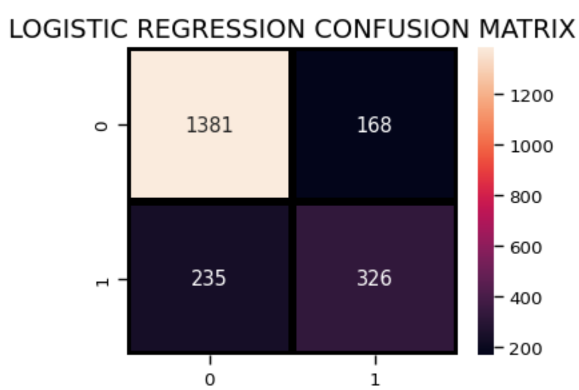
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

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

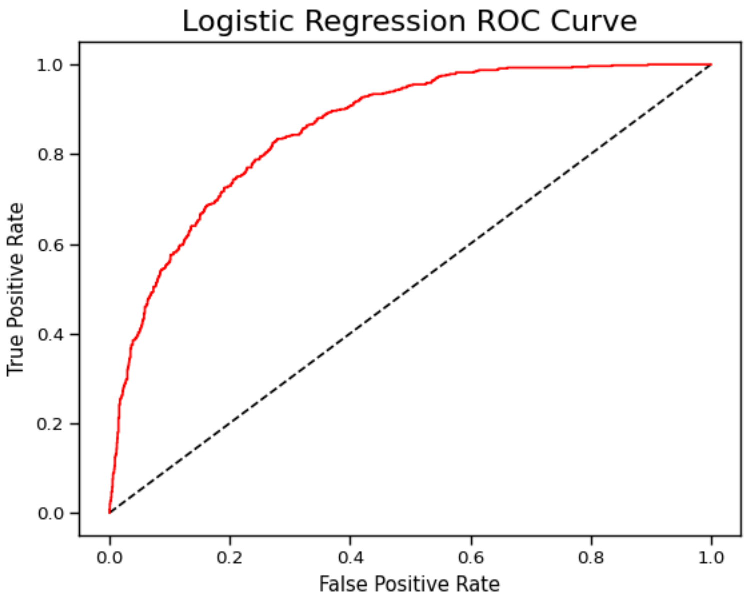
```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1549
           1       0.66      0.58      0.62       561

    accuracy                           0.81      2110
   macro avg       0.76      0.74      0.75      2110
weighted avg       0.80      0.81      0.80      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()
```



```
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

```
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 : 0.7336492890995261
```

Decision tree gives very low score.

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

```
              precision    recall  f1-score   support
```