Telcom Customer Churn Prediction

Content

1. Introduction

What is Customer Churn?

Customer churn is defined as the discontinuation of business by customers or subscribers with a particular firm or service.

In the telecom industry, customers have the option to choose among various service providers and often switch from one to another. The telecommunications business experiences an annual churn rate of 15-25 percent in this highly competitive market.

Individualized customer retention is challenging due to the large number of customers most firms serve, making it unfeasible to dedicate extensive time to each. The associated costs would outweigh the additional revenue. However, if a corporation could forecast which customers are likely to leave in advance, it could focus its customer retention efforts solely on these 'high-risk' clients. The ultimate goal is to expand the coverage area and enhance customer loyalty. The core of success in this market lies within the customer itself.

Customer churn is a critical metric because retaining existing customers is significantly less expensive than acquiring new ones.

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

To detect early signs of potential churn, one must first develop a comprehensive view of customers and their interactions across multiple channels, including store/branch visits, product purchase histories, customer service calls, web-based transactions, and social media interactions, among others.

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

Objectives: I will explore the data and attempt to answer several questions, including:

What percentage of customers churn, and what percentage continue with active services? Are there any patterns among churn customers based on gender? Are there any patterns or preferences among churn customers based on the type of service provided? What are the most profitable service types? Which features and services are the most profitable?

2. Loading libraries and data

In [1]:
```
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')
```

In [2]:
```
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
```

In [3]:
```
#loading data
df = pd.read_csv('Telco-Customer-Churn.csv')
```

3. Undertanding the data

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

In [4]:
```
df.head()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	...	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMet
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	...	No	No	No	No	Month-to-month	Yes	Electronic ch
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	...	Yes	No	No	No	One year	No	Mailed ch
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	...	No	No	No	No	Month-to-month	Yes	Mailed ch
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	...	Yes	Yes	No	No	One year	No	Bank trar (autom
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	...	No	No	No	No	Month-to-month	Yes	Electronic ch

5 rows × 21 columns

The dataset comprises information on:

Customers who left within the last month – represented by the column 'Churn'

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

Customer account details - encompassing their tenure as customers, contract information, payment method, paperless billing status, monthly charges, and total charges

Demographic information about customers – covering gender, age range, and whether they have partners and dependents

In [5]:
```
df.shape
```

Out[5]:
```
(7043, 21)
```

In [6]:
```
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
```

In [7]:
```
df.columns.values
```

Out[7]:
```
array(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies', 'Contract',
       'PaperlessBilling', 'PaymentMethod', 'MonthlyCharges',
       'TotalCharges', 'Churn'], dtype=object)
```

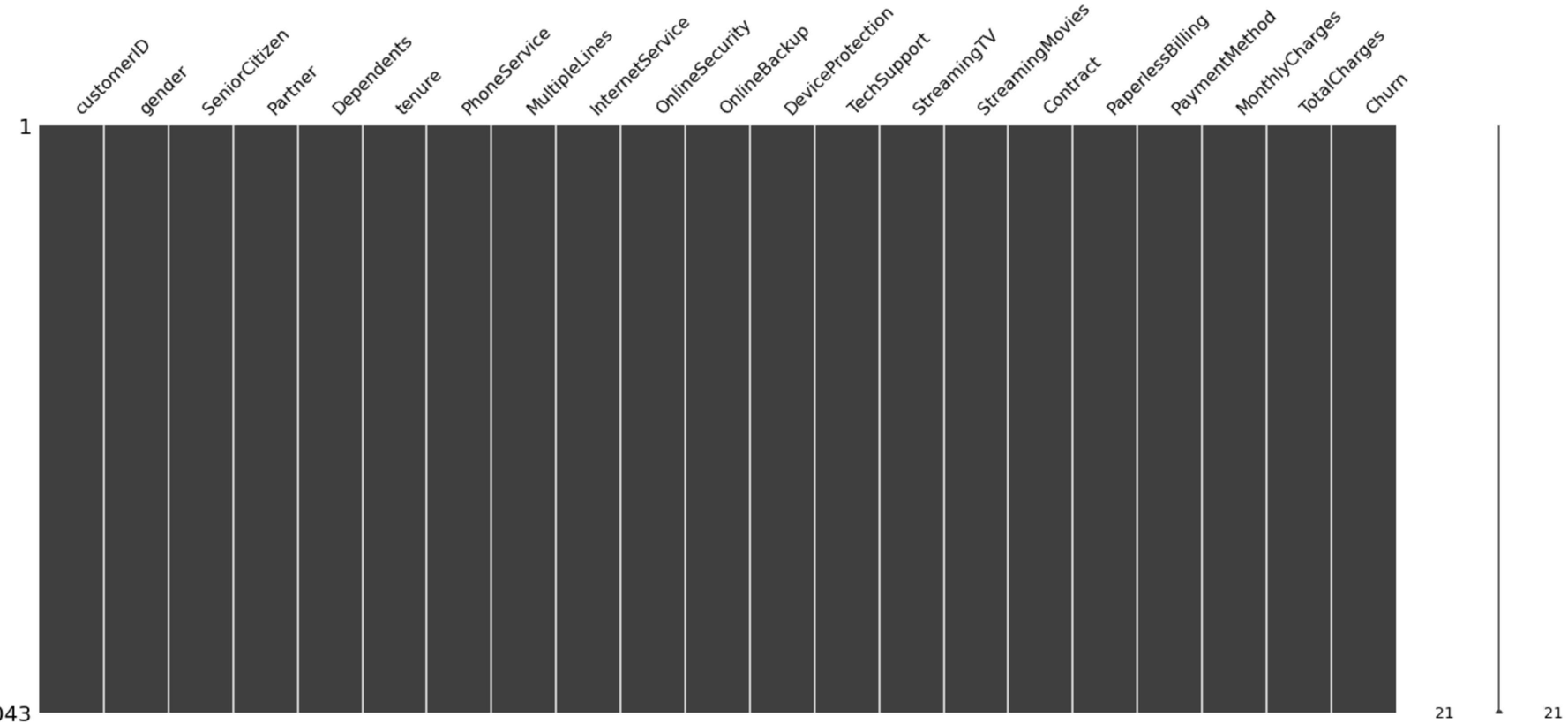
In [8]:
```
df.dtypes
```

Out[8]:
```
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
```

The target for our exploration is Churn.

4. Visualize missing values

In [9]:
```
# Visualize missing values as a matrix
msno.matrix(df);
```



Utilizing this matrix, we quickly identify the dataset's missingness pattern. The visualization above reveals no noticeable or peculiar patterns; in fact, there is no missing data.

5. Data Manipulation

In [10]:
```
df = df.drop(['customerID'], axis = 1)
df.head()
```

Out[10]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMeth
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	No	No	Month-to-month	Yes	Electronic che
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No	One year	No	Mailed che
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No	Month-to-month	Yes	Mailed che
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	No	No	One year	No	Bank trans (automat
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No	Month-to-month	Yes	Electronic che

Upon thorough analysis, we may identify indirect missingness in our data, which could manifest as blank spaces.

In [11]:
```
df['TotalCharges'] = pd.to_numeric(df.TotalCharges, errors='coerce')
df.isnull().sum()
```

Out[11]:
```
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
```

It's evident that the 'TotalCharges' column contains 11 missing values.

In [12]:
```
df[np.isnan(df['TotalCharges'])]
```

Out[12]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentM
488	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	No	Yes	Yes	Yes	No	Two year	Yes	Bank tr (auto
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
936	Female	0	Yes	Yes	0	Yes	No	DSL	Yes	Yes	Yes	No	Yes	Yes	Two year	No	Mailed
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
1340	Female	0	Yes	Yes	0	No	No phone service	DSL	Yes	Yes	Yes	Yes	Yes	No	Two year	No	Cred (auto
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
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
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
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
6670	Female	0	Yes	Yes	0	Yes	Yes	DSL	No	Yes	Yes	Yes	Yes	No	Two year	No	Mailed
6754	Male	0	No	Yes	0	Yes	Yes	DSL	Yes	Yes	No	Yes	No	No	Two year	Yes	Bank tr (auto

It's noteworthy that the 'Tenure' column registers as 0 for these entries, despite the 'MonthlyCharges' column not being empty. Let's further investigate if there are any other instances of 0 values in the 'Tenure' column.

In [13]:
```
df[df['tenure'] == 0].index
```

Out[13]:
```
Int64Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='int64')
```

No further missing values are present in the 'Tenure' column. Let's proceed to delete the 11 rows with missing values in the 'Tenure' column since their removal will not significantly impact the data.

In [14]:
```
df.drop(labels=df[df['tenure'] == 0].index, axis=0, inplace=True)
df[df['tenure'] == 0].index
```

Out[14]:
```
Int64Index([], dtype='int64')
```

To address the issue of missing values in the 'TotalCharges' column, I have chosen to fill them with the mean of the 'TotalCharges' values.

In [15]:
```
df.fillna(df["TotalCharges"].mean())
```

Out[15]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentM
0	Female	0	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	No	No	Month-to-month	Yes	Electronic
1	Male	0	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No	One year	No	Mailed
2	Male	0	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No	Month-to-month	Yes	Mailed
3	Male	0	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	No	No	One year	No	Bank tr (auto
4	Female	0	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No	Month-to-month	Yes	Electronic
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7038	Male	0	Yes	Yes	24	Yes	Yes	DSL	Yes	No	Yes	Yes	Yes	Yes	One year	Yes	Mailed
7039	Female	0	Yes	Yes	72	Yes	Yes	Fiber optic	No	Yes	Yes	No	Yes	Yes	One year	Yes	Cred (auto
7040	Female	0	Yes	Yes	11	No	No phone service	DSL	Yes	No	No	No	No	No	Month-to-month	Yes	Electronic
7041	Male	1	Yes	No	4	Yes	Yes	Fiber optic	No	No	No	No	No	No	Month-to-month	Yes	Mailed
7042	Male	0	No	No	66	Yes	No	Fiber optic	Yes	No	Yes	Yes	Yes	Yes	Two year	Yes	Bank tr (auto

7032 rows × 20 columns

In [16]:
```
df.isnull().sum()
```

Out[16]:
```
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
```

In [17]:
```
df["SeniorCitizen"]= df["SeniorCitizen"].map({0: "No", 1: "Yes"})
df.head()
```

Out[17]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMeth
0	Female	No	Yes	No	1	No	No phone service	DSL	No	Yes	No	No	No	No	Month-to-month	Yes	Electronic che
1	Male	No	No	No	34	Yes	No	DSL	Yes	No	Yes	No	No	No	One year	No	Mailed che
2	Male	No	No	No	2	Yes	No	DSL	Yes	Yes	No	No	No	No	Month-to-month	Yes	Mailed che
3	Male	No	No	No	45	No	No phone service	DSL	Yes	No	Yes	Yes	No	No	One year	No	Bank trans (automat
4	Female	No	No	No	2	Yes	No	Fiber optic	No	No	No	No	No	No	Month-to-month	Yes	Electronic che

In [18]:
```
df["InternetService"].describe(include=['object', 'bool'])
```

Out[18]:
```
count            7032
unique              3
top       Fiber optic
freq             3096
Name: InternetService, dtype: object
```

In [19]:
```
numerical_cols = ['tenure', 'MonthlyCharges', 'TotalCharges']
df[numerical_cols].describe()
```

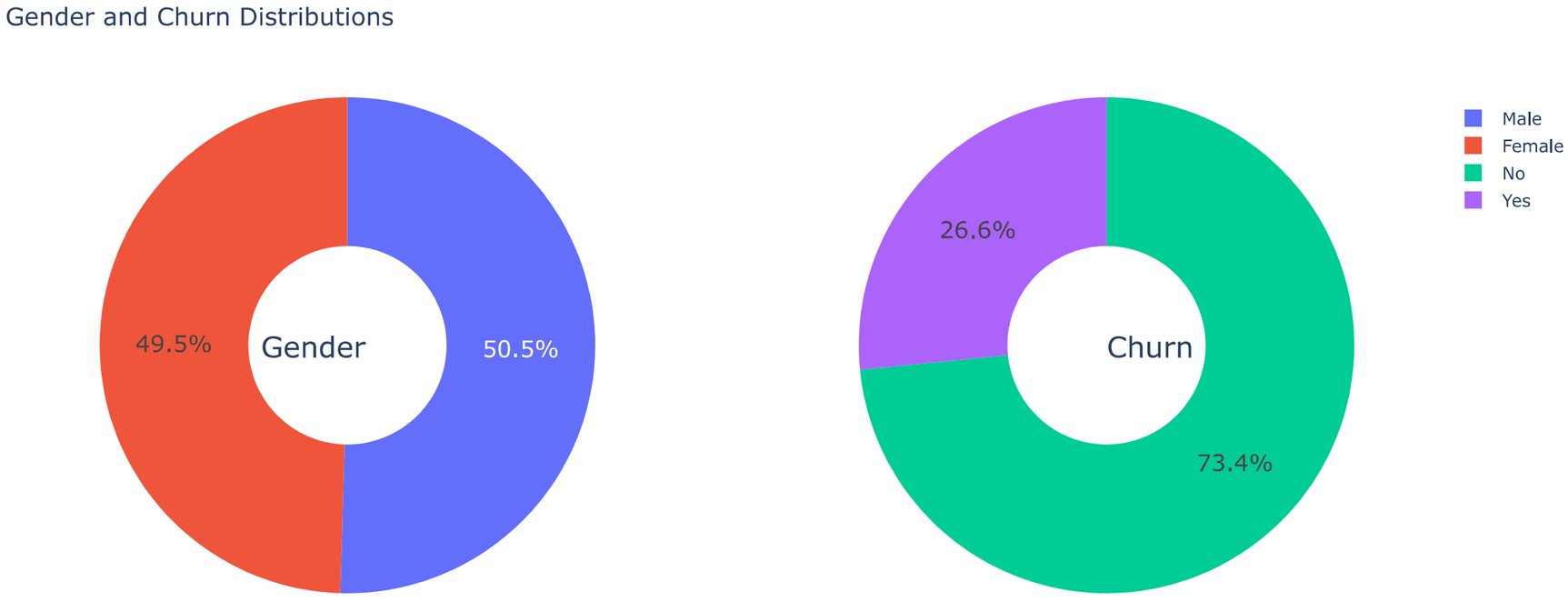
Out[19]:

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

6. Data Visualization

In [20]:

```
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 consist of 49.5% female and 50.5% male."

In [21]:
```
df["Churn"][df["Churn"]=="No"].groupby(by=df["gender"]).count()
```

Out[21]:
```
gender
Female    2544
Male      2619
Name: Churn, dtype: int64
```

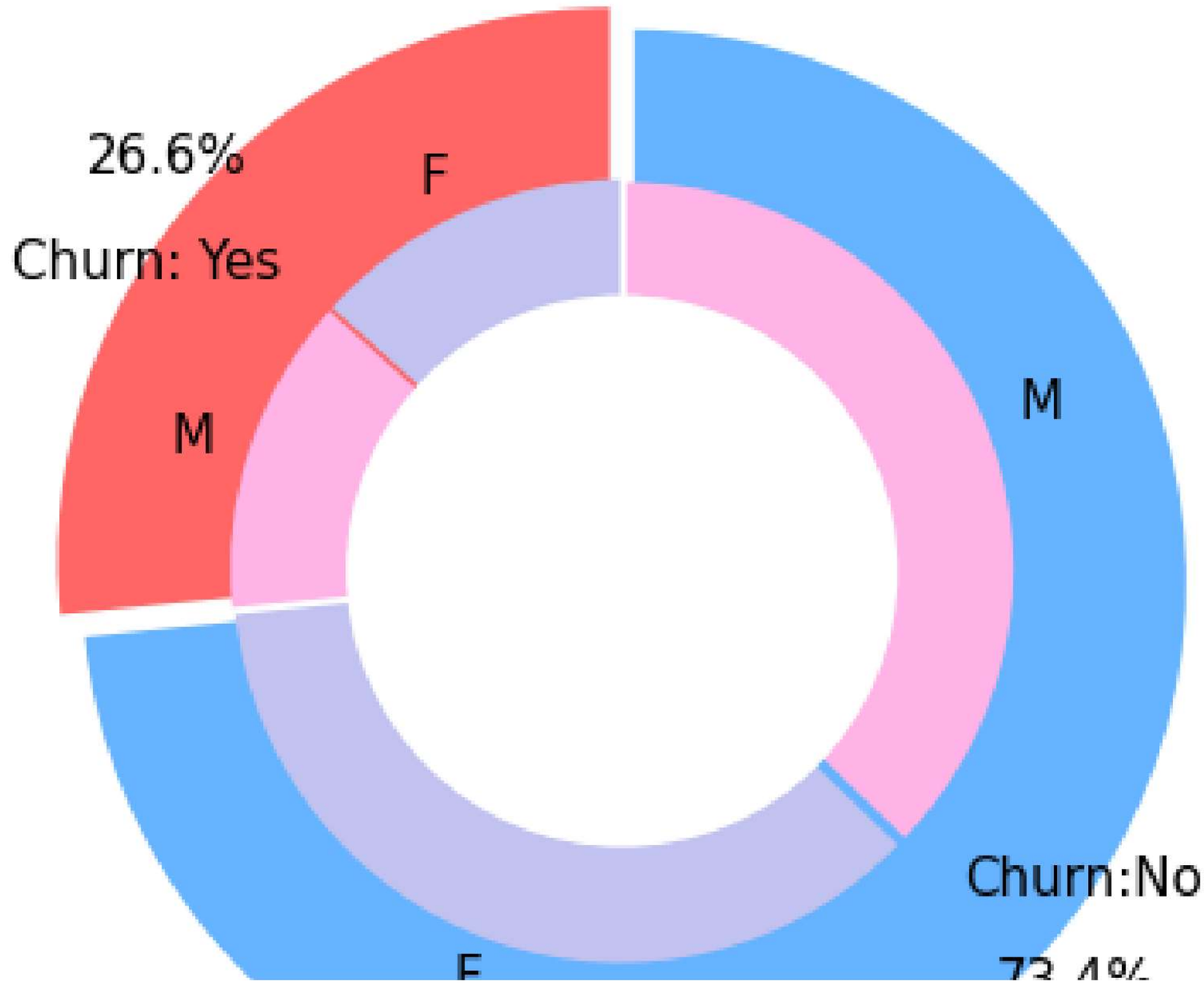
In [22]:
```
df["Churn"][df["Churn"]=="Yes"].groupby(by=df["gender"]).count()
```

Out[22]:
```
gender
Female    939
Male      930
Name: Churn, dtype: int64
```

In [23]:
```
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()
```

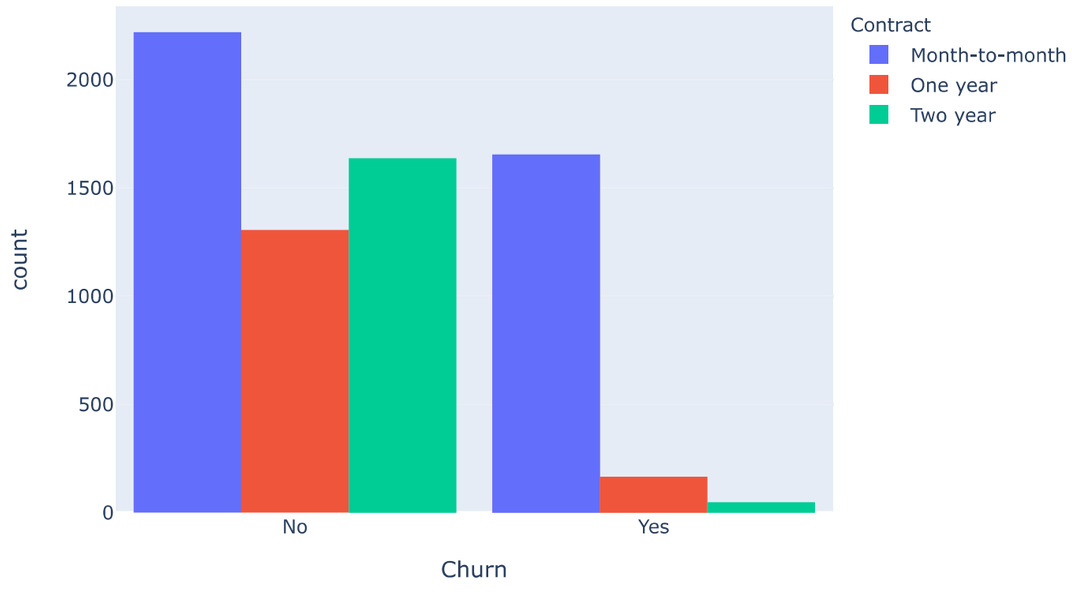
Churn Distribution w.r.t Gender: Male(M), Female(F)



There is a negligible difference in the percentage/count of customers who changed their service provider. Both genders exhibited a similar pattern when it comes to migrating to another service provider or firm.

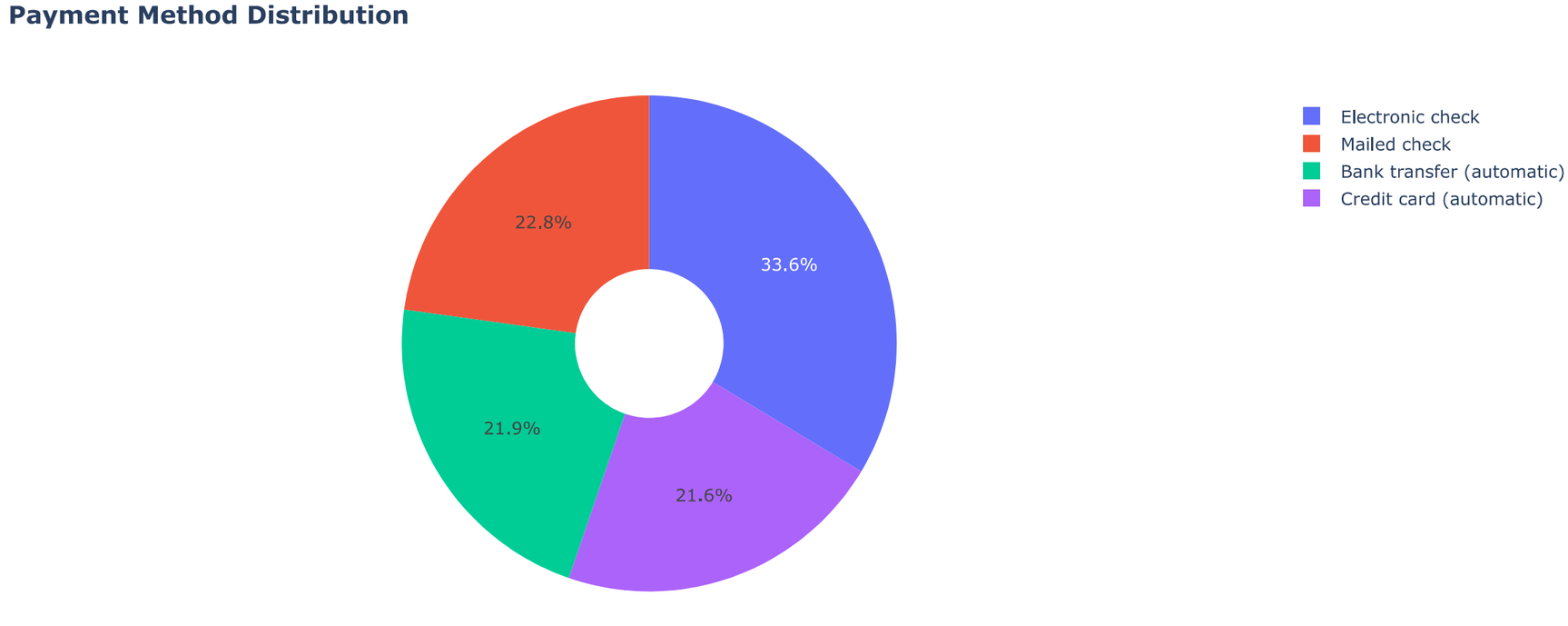
In [24]:
```
fig = px.histogram(df, x="Churn", color="Contract", barmode="group", title="<b>Customer contract distribution<b>")
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

Customer contract distribution

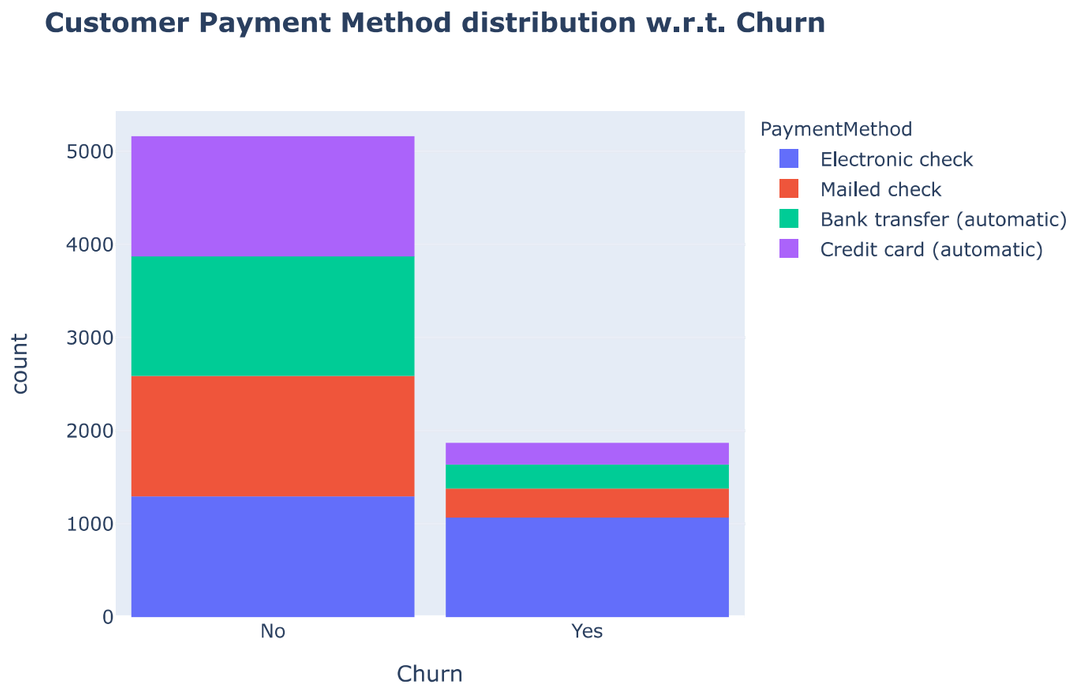


Roughly 75% of customers with a Month-to-Month Contract chose to leave, while only 13% of customers with a One-Year Contract and 3% with a Two-Year Contract opted to switch providers.

In [25]:
```
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()
```



In [26]:
```
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()
```



The majority of customers who moved out had Electronic Check as their payment method. Conversely, customers who opted for Credit Card automatic transfer, Bank Automatic Transfer, or Mailed Check were less likely to move out.

In [27]:
```
df["InternetService"].unique()
```

Out[27]:
```
array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

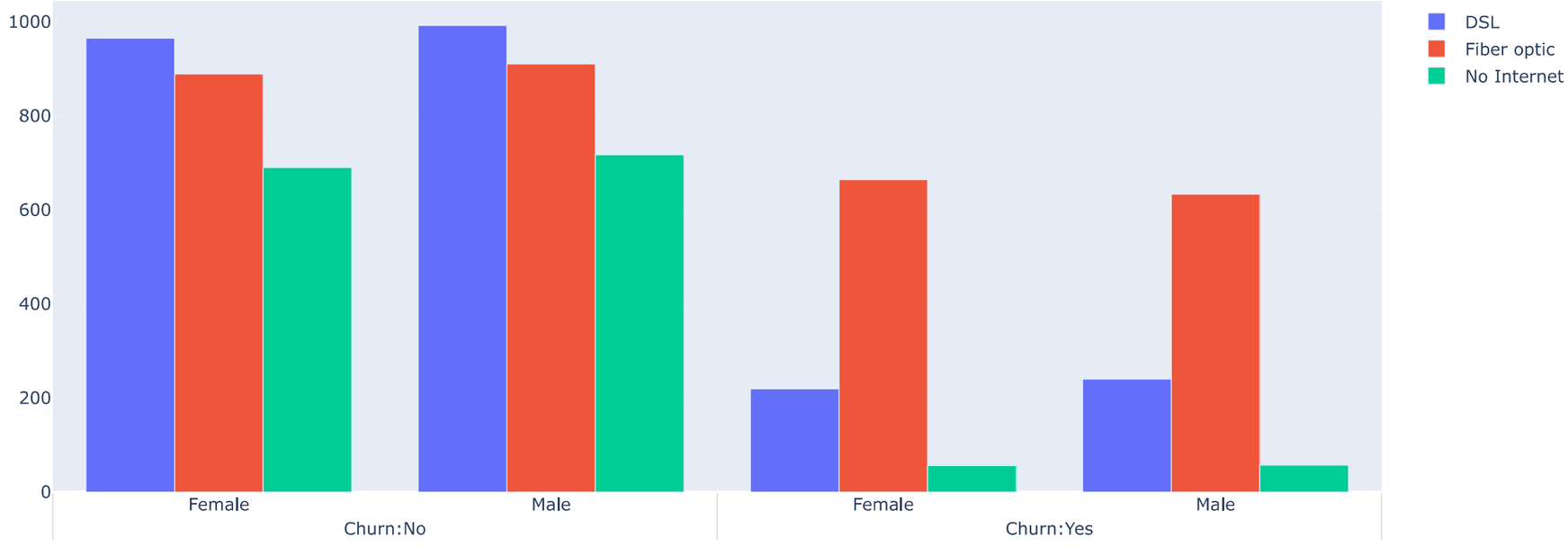
In [28]:
```
 df[df["gender"]=="Male"][["InternetService", "Churn"]].value_counts()
```

Out[28]:
```
InternetService  Churn
DSL              No       992
Fiber optic      No       910
No               No       717
Fiber optic      Yes      633
DSL              Yes      240
No               Yes       57
dtype: int64
```

In [29]:
```
df[df["gender"]=="Female"][["InternetService", "Churn"]].value_counts()
```

Out[29]:
```
InternetService  Churn
DSL              No       965
Fiber optic      No       889
No               No       690
Fiber optic      Yes      664
DSL              Yes      219
No               Yes       56
dtype: int64
```

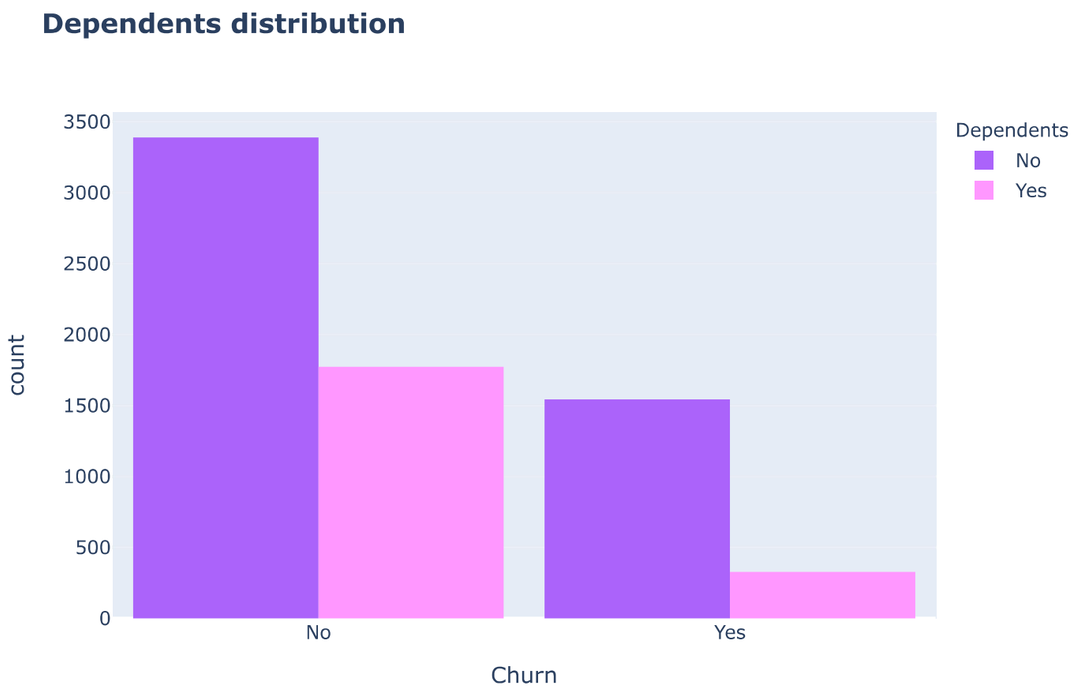
In [30]:
```
fig = go.Figure()

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()
```



Many customers opt for the Fiber Optic service, yet it's noticeable that these customers experience a high churn rate. This might indicate dissatisfaction with this specific internet service.

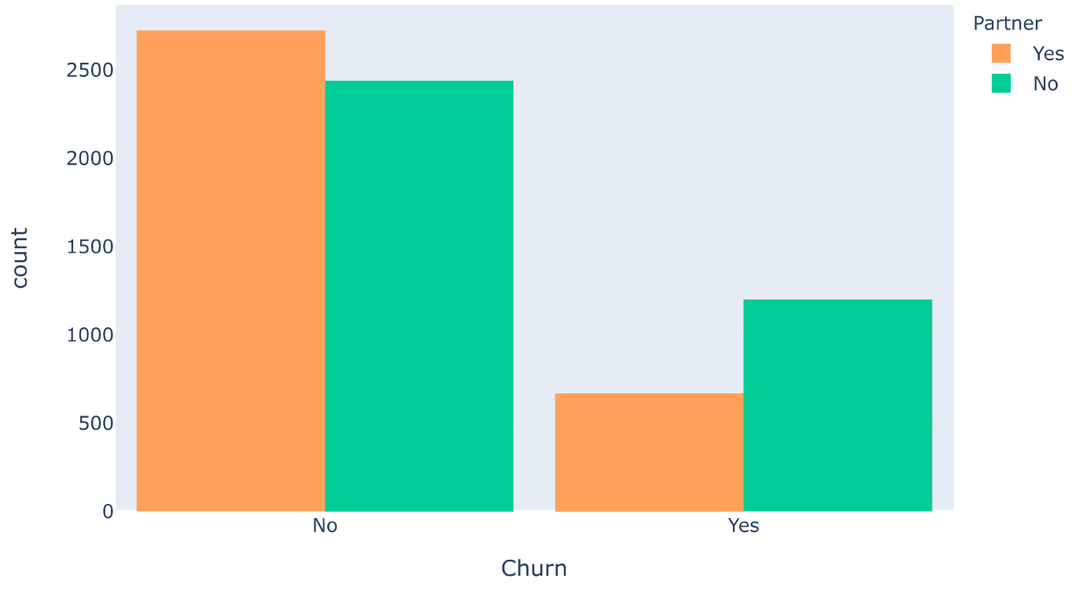
DSL service users constitute the majority in numbers and exhibit a lower churn rate in comparison to Fiber Optic service users.

In [31]:
```
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()
```



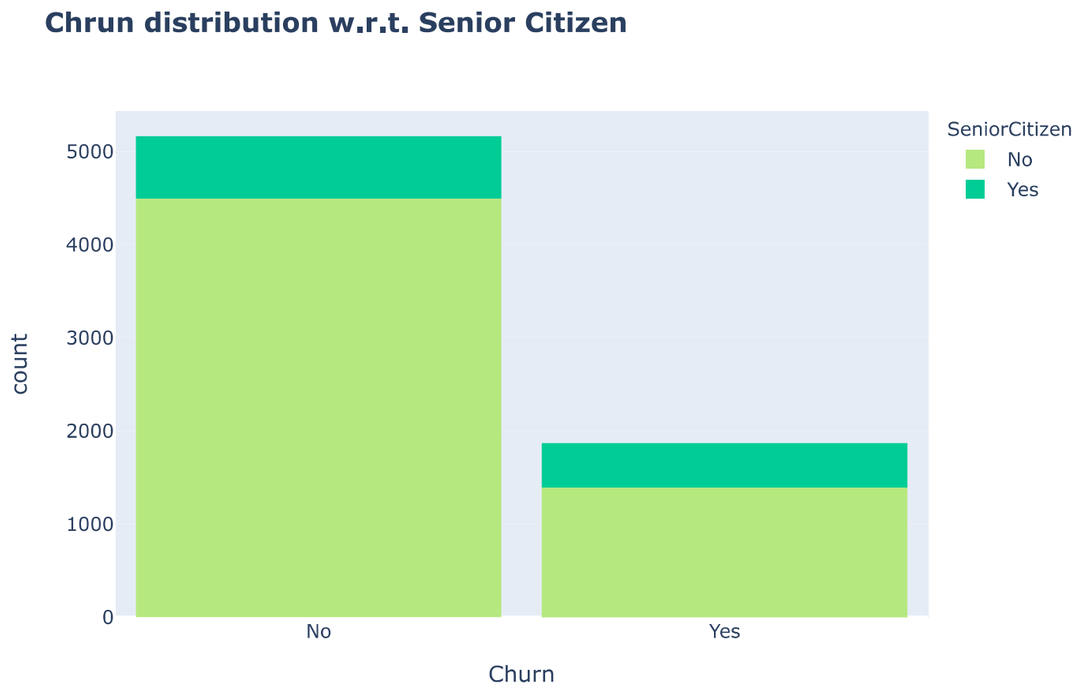
Customers without dependents are more likely to churn

In [32]:
```
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 doesn't have partners are more likely to churn

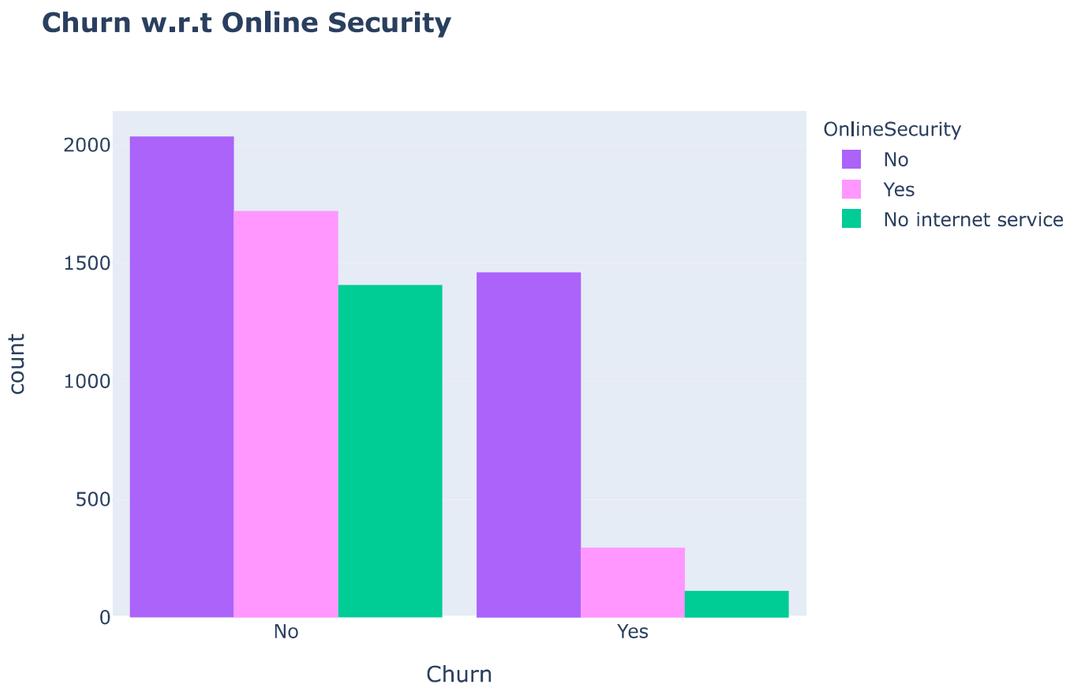
In [33]:
```
color_map = {"Yes": '#00CC96', "No": '#B6E880'}
fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution w.r.t. Senior Citizen</b>", color_discrete_map=color_map)
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```



"The fraction of senior citizens is notably low in the dataset."

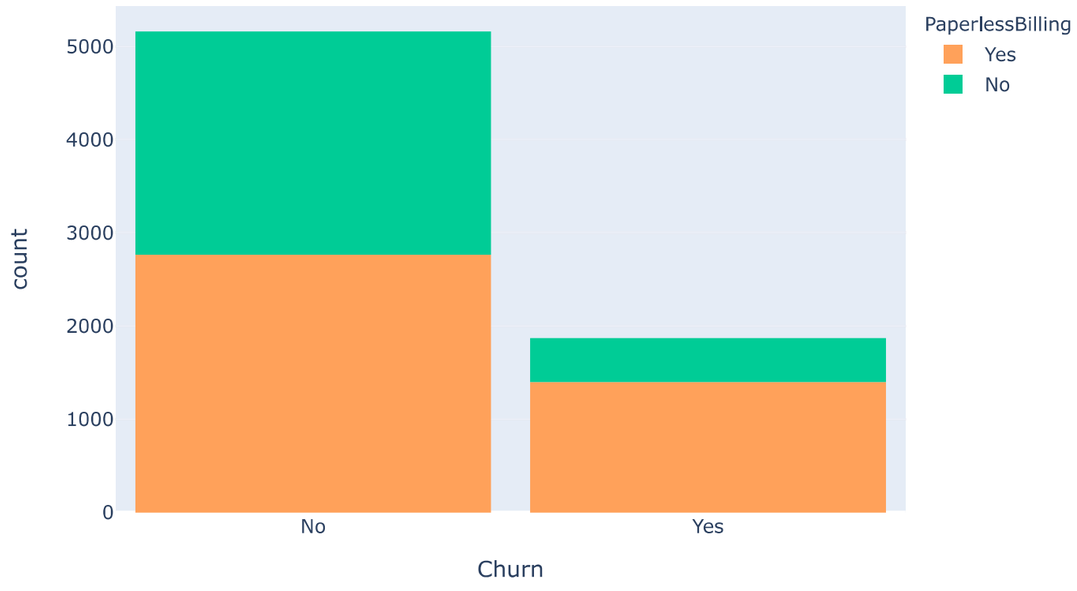
"Additionally, a significant portion of senior citizens are observed to churn."

In [34]:
```
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()
```



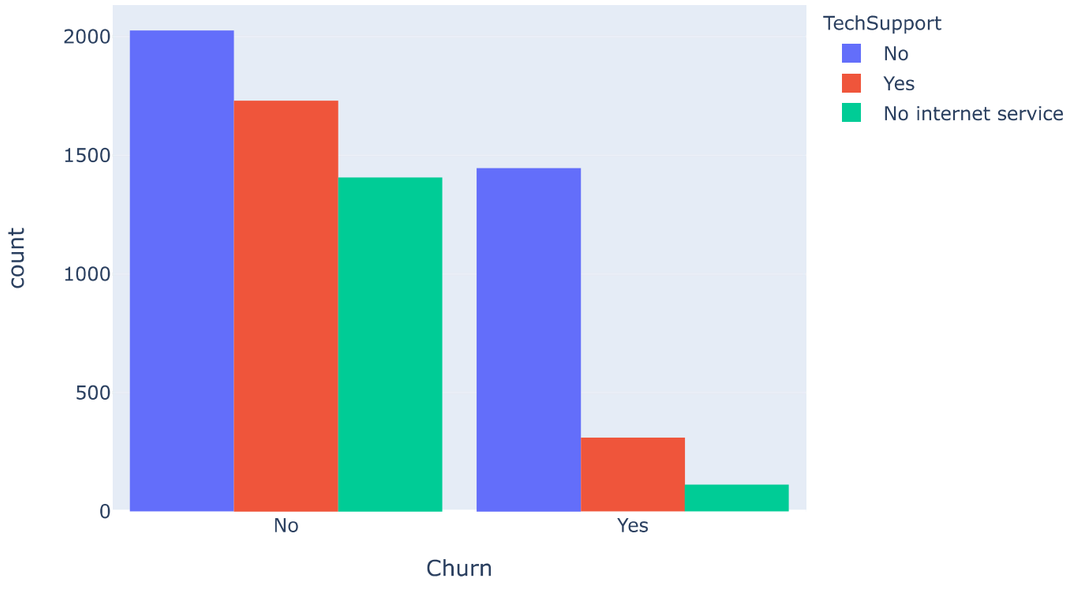
Most customers churn in the absence of online security,

In [35]:
```
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()
```



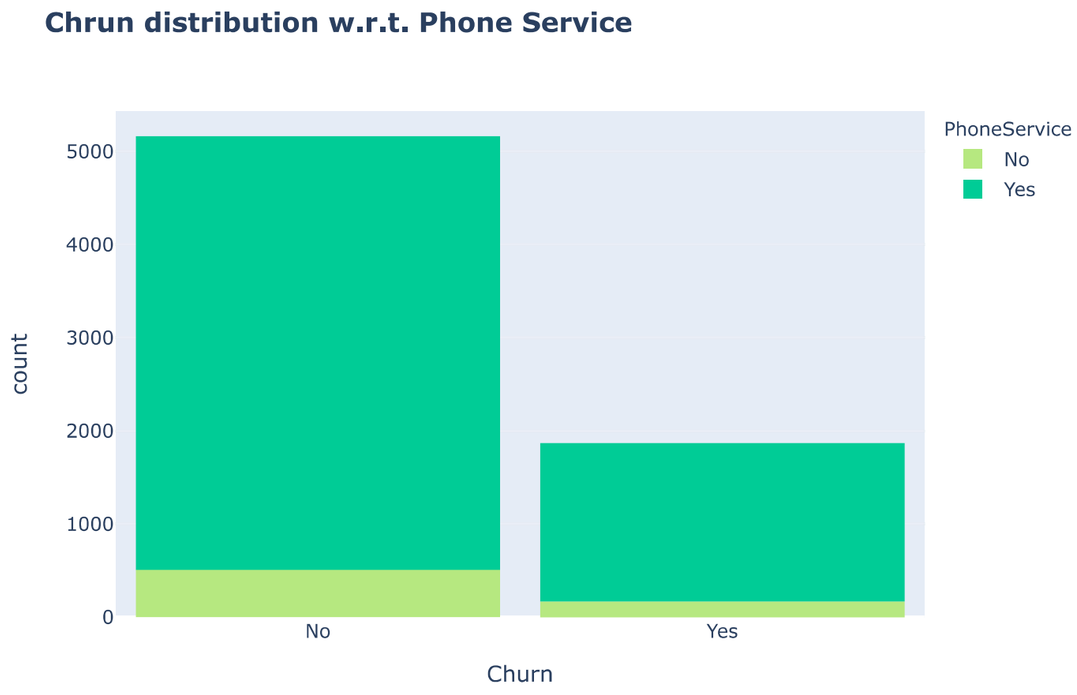
Customers with Paperless Billing are most likely to churn.

In [36]:
```
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()
```



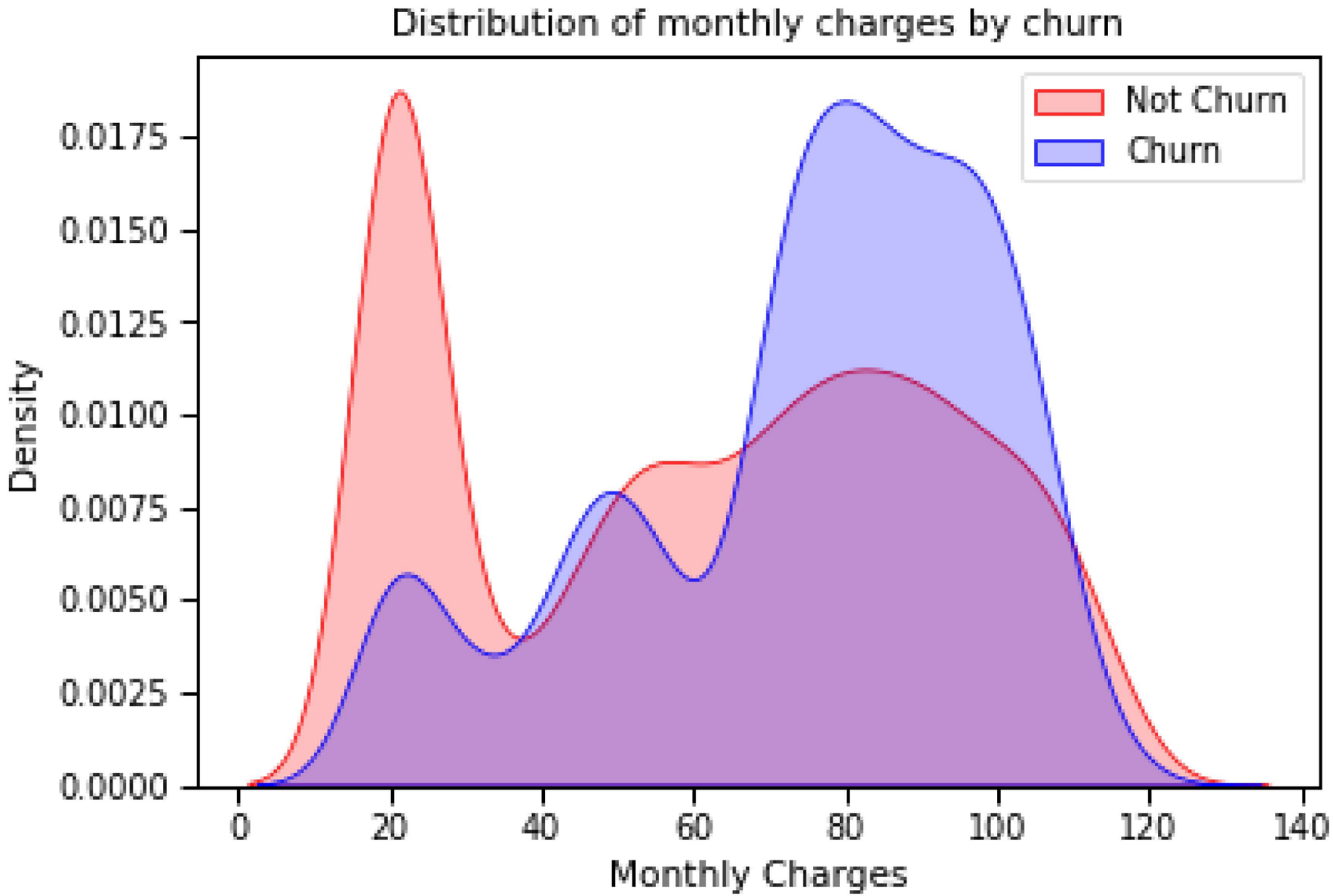
Customers with no TechSupport are most likely to migrate to another service provider.

In [37]:
```
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()
```



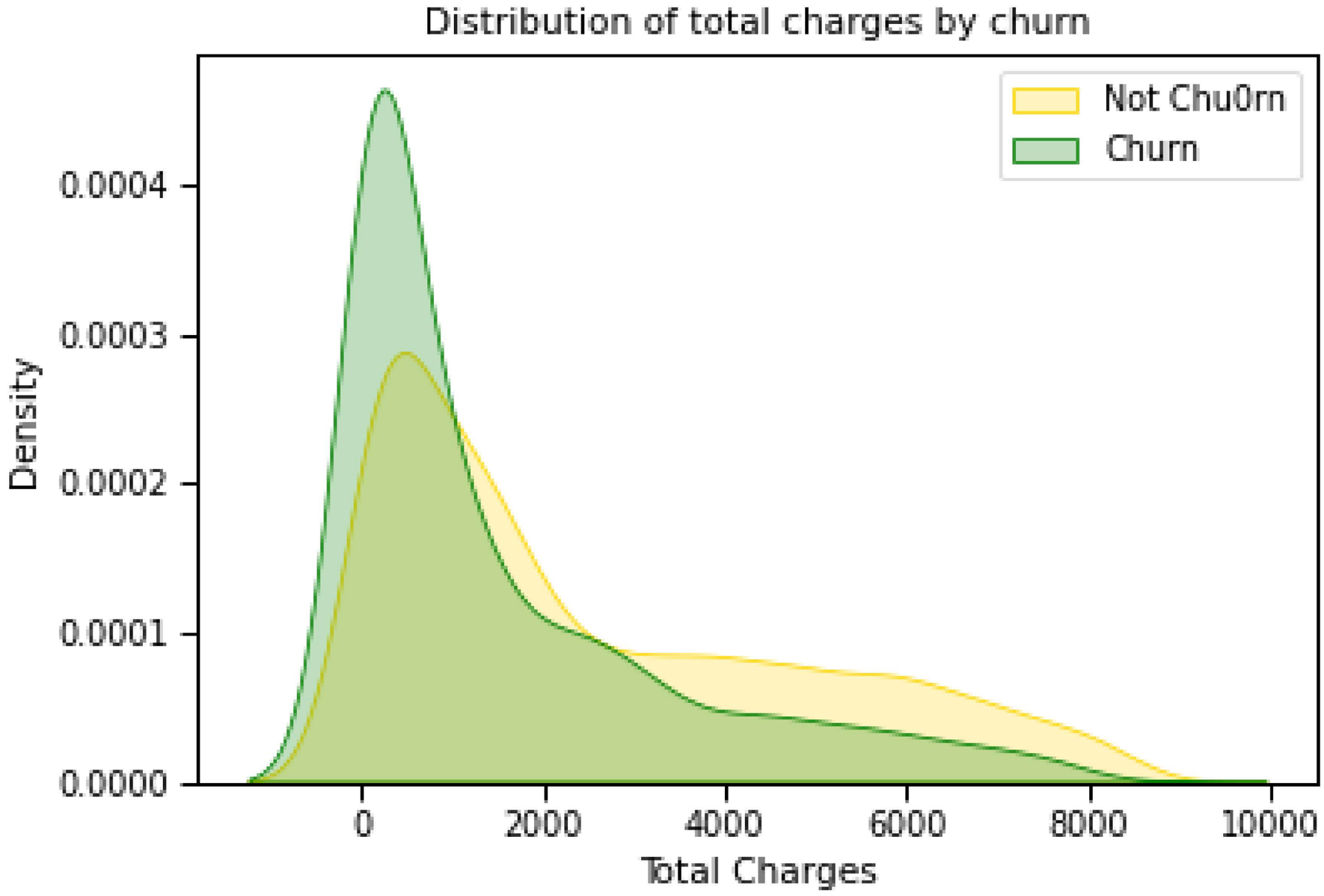
Very small fraction of customers don't have a phone service and out of that, 1/3rd Customers are more likely to churn.

In [38]:
```
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');
```



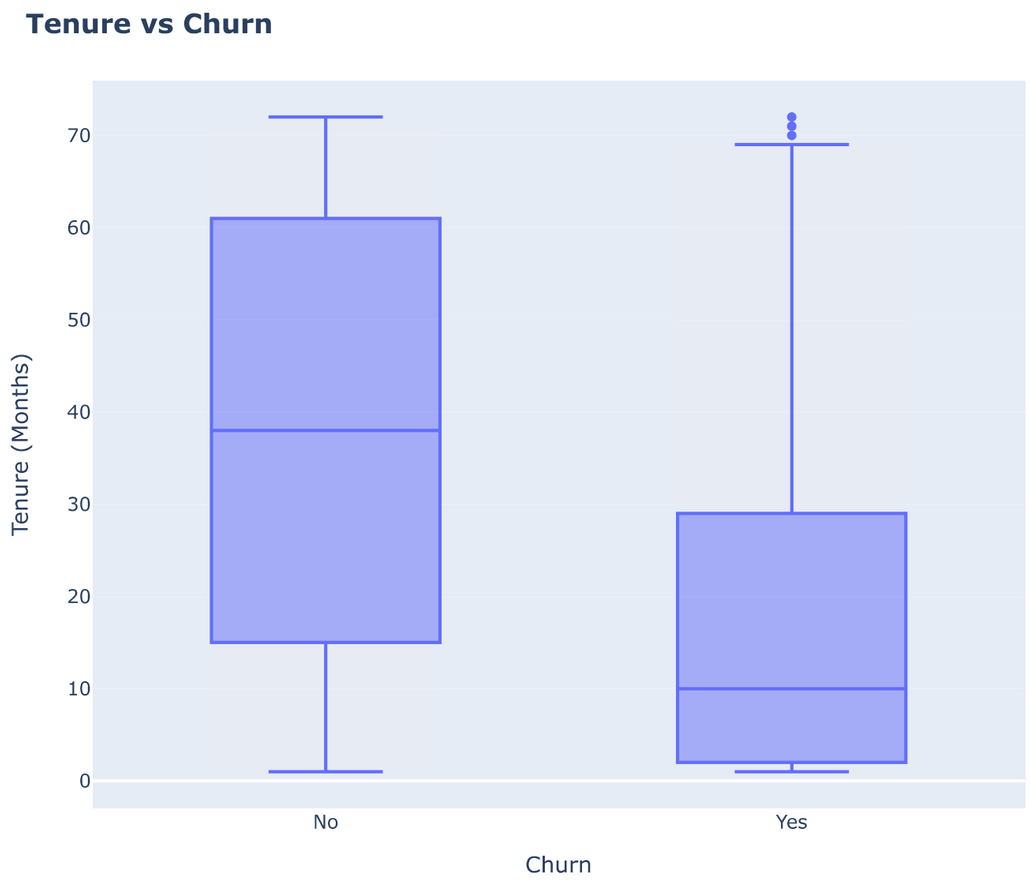
Customers with higher Monthly Charges are also more likely to churn

In [39]:
```
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');
```



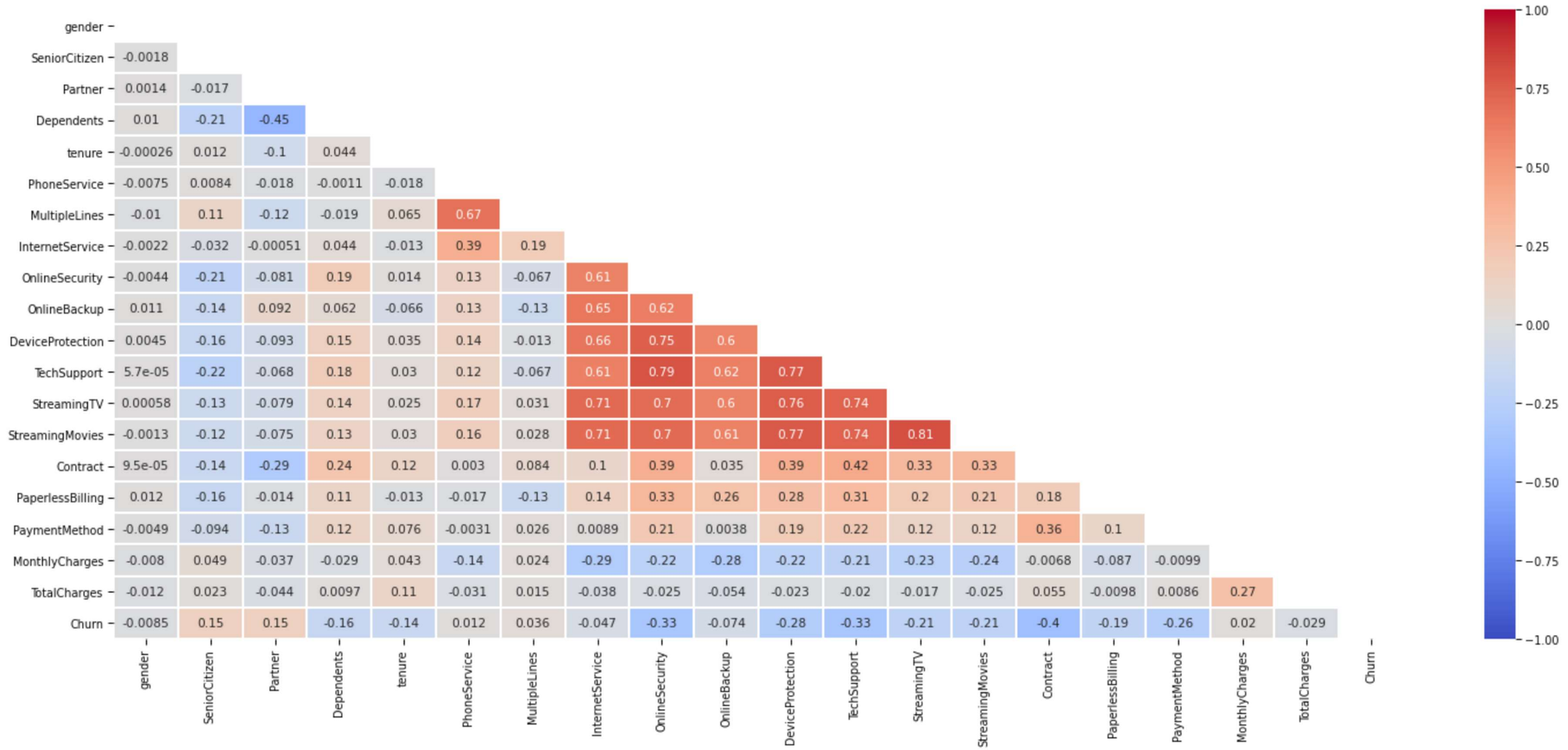
In [40]:
```
fig = px.box(df, x='Churn', y = 'tenure')

# 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

In [41]:
```
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)
```



7. Data Preprocessing

Splitting the data into train and test datasets

In [42]:
```
def object_to_int(dataframe_series):
    if dataframe_series.dtype=='object':
        dataframe_series = LabelEncoder().fit_transform(dataframe_series)
    return dataframe_series
```

In [43]:
```
df = df.apply(lambda x: object_to_int(x))
df.head()
```

Out[43]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies	Contract	PaperlessBilling	PaymentMeth
0	0	0	1	0	1	0	1	0	0	2	0	0	0	0	0	1	
1	1	0	0	0	34	1	0	0	2	0	2	0	0	0	1	0	
2	1	0	0	0	2	1	0	0	2	2	0	0	0	0	0	1	
3	1	0	0	0	45	0	1	0	2	0	2	2	0	0	1	0	
4	0	0	0	0	2	1	0	1	0	0	0	0	0	0	0	1	

In [44]:
```
plt.figure(figsize=(14,7))
df.corr()['Churn'].sort_values(ascending = False)
```

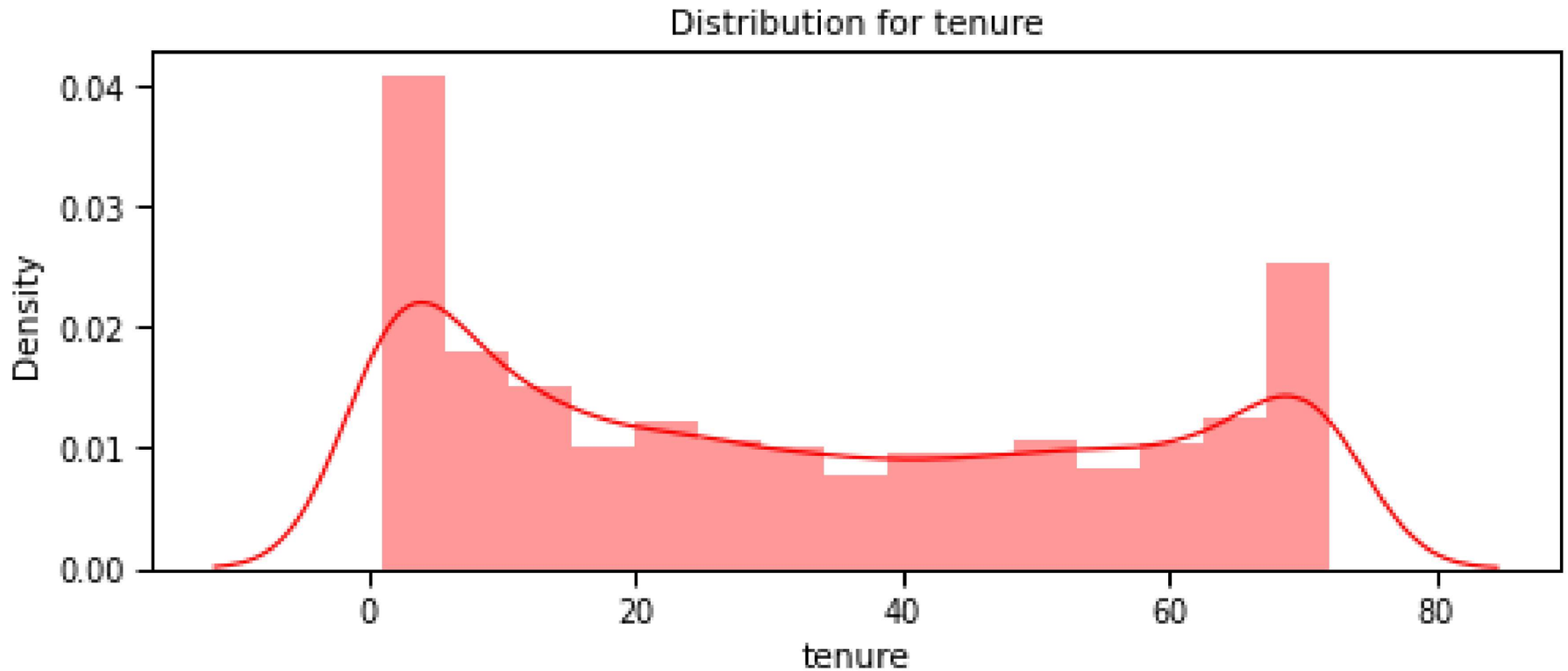
Out[44]:
```
Churn              1.000000
MonthlyCharges     0.192858
PaperlessBilling   0.191454
SeniorCitizen      0.150541
PaymentMethod      0.107852
MultipleLines      0.038043
PhoneService       0.011691
gender            -0.008545
StreamingTV       -0.036303
StreamingMovies   -0.038802
InternetService   -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 1008x504 with 0 Axes>
```

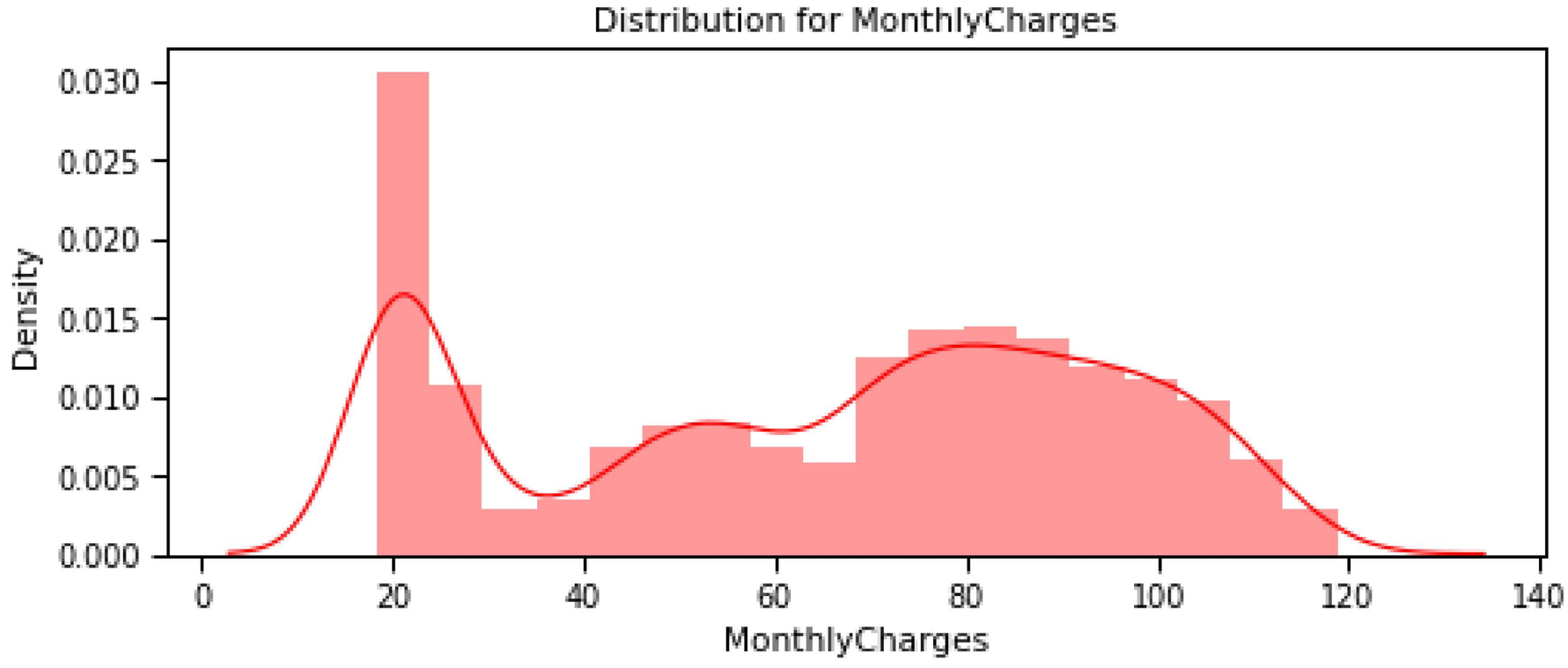
In [45]:
```
X = df.drop(columns = ['Churn'])
y = df['Churn'].values
```

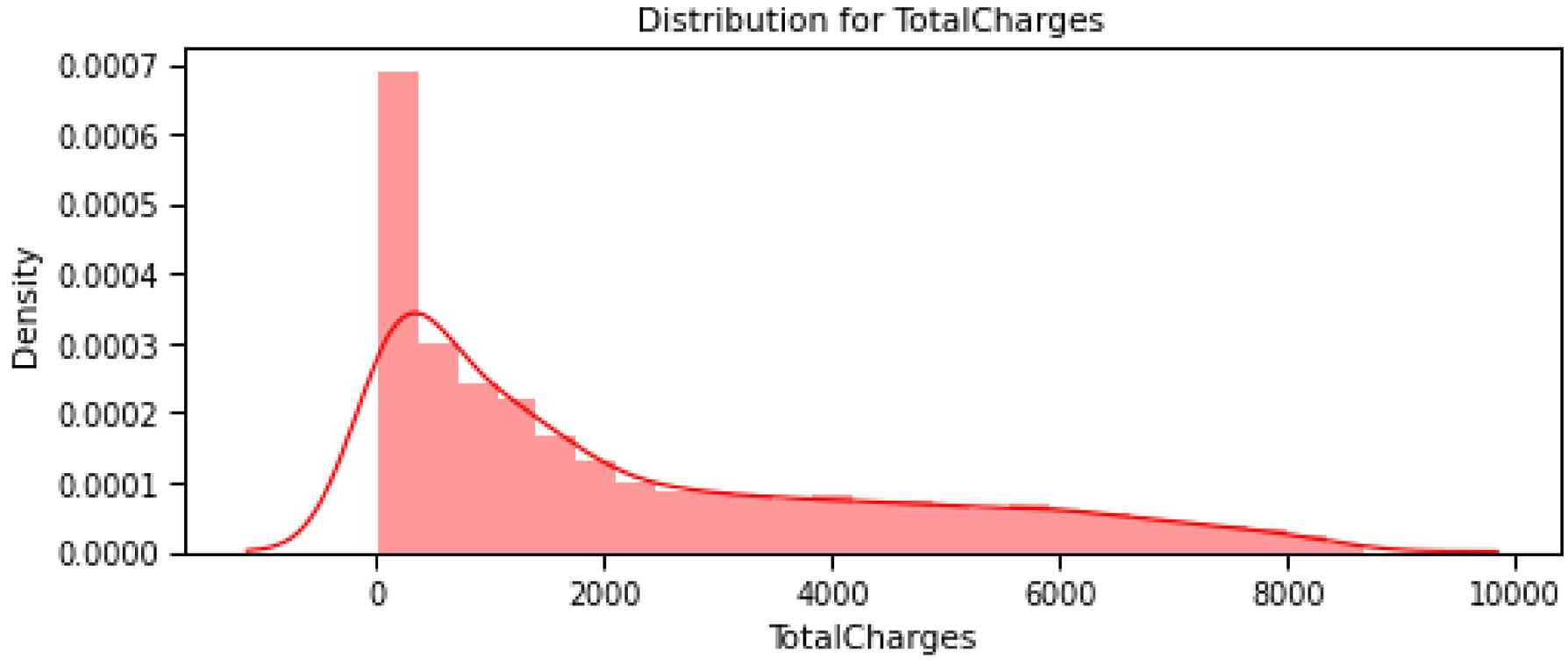
In [46]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.30, random_state = 40, stratify=y)
```

In [47]:
```
def distplot(feature, frame, color='r'):
    plt.figure(figsize=(8,3))
    plt.title("Distribution for {}".format(feature))
    ax = sns.distplot(frame[feature], color= color)
```

In [48]:
```
num_cols = ["tenure", 'MonthlyCharges', 'TotalCharges']
for feat in num_cols: distplot(feat, df)
```


Distribution for tenure
Density
0.04
0.03
0.02
0.01
0.00
0
20
40
60
80
tenure

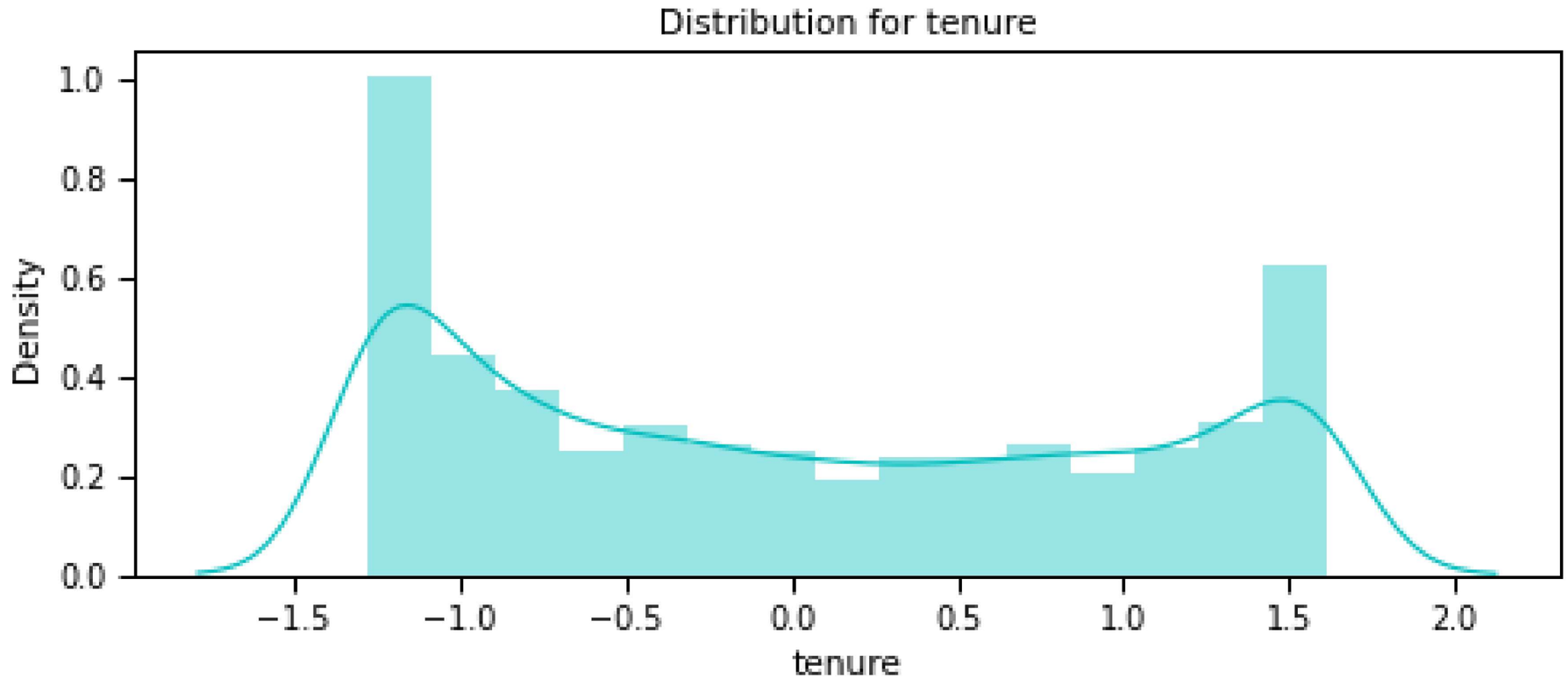

Distribution for MonthlyCharges
Density
0.030
0.025
0.020
0.015
0.010
0.005
0.000
0
20
40
60
80
100
120
140
MonthlyCharges

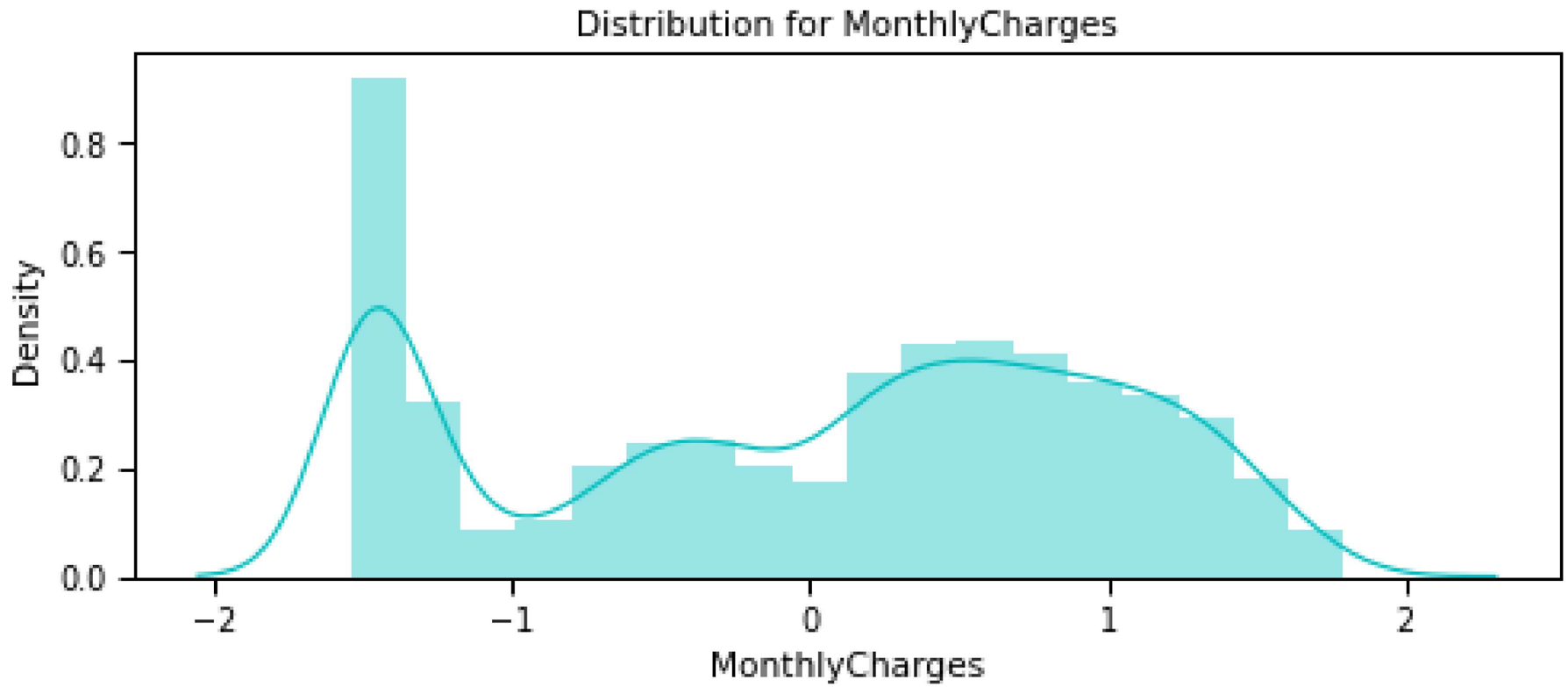


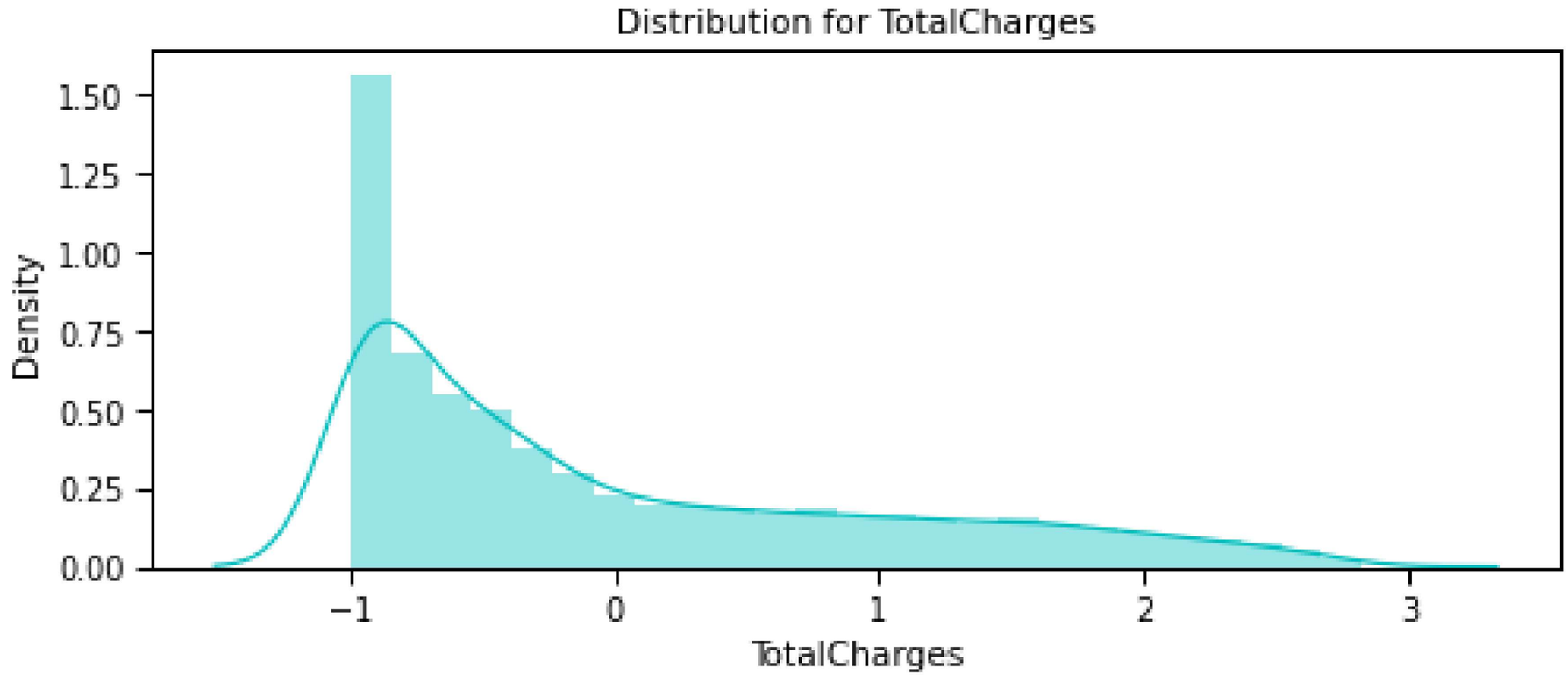
Since the numerical features are distributed over different value ranges, I will use standard scaler to scale them down to the same range.

Standardizing numeric attributes

In [49]:
```
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')
```


Distribution for tenure
Density
1.0
0.8
0.6
0.4
0.2
0.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
tenure


Distribution for MonthlyCharges
Density
0.0
0.2
0.4
0.6
0.8
−2
−1
0
1
2
MonthlyCharges



In [50]:
```
# 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
```

In [51]:
```
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

In [52]:
```
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: 0.7753554502369668
```

In [53]:
```
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
```

SVC

In [54]:
```
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 : 0.8075829383886256
```

In [55]:
```
print(classification_report(y_test, predict_y))
```

```
              precision    recall  f1-score   support

           0       0.84      0.92      0.88      1549
           1       0.69      0.50      0.58       561

    accuracy                           0.81      2110
   macro avg       0.76      0.71      0.73      2110
weighted avg       0.80      0.81      0.80      2110
```

Random Forest

In [56]:
```
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))
```

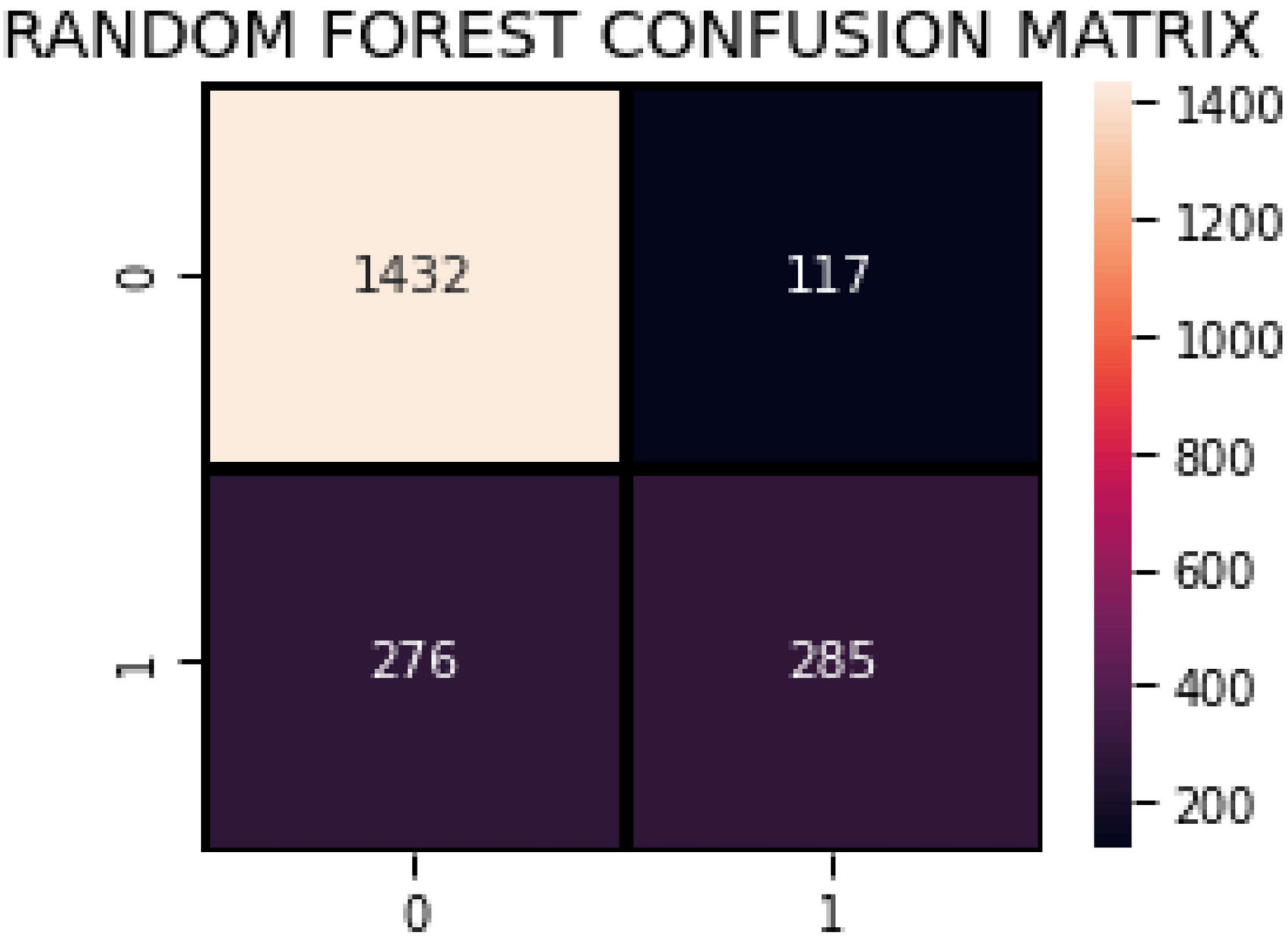
```
0.8137440758293839
```

In [57]:
```
print(classification_report(y_test, prediction_test))
```

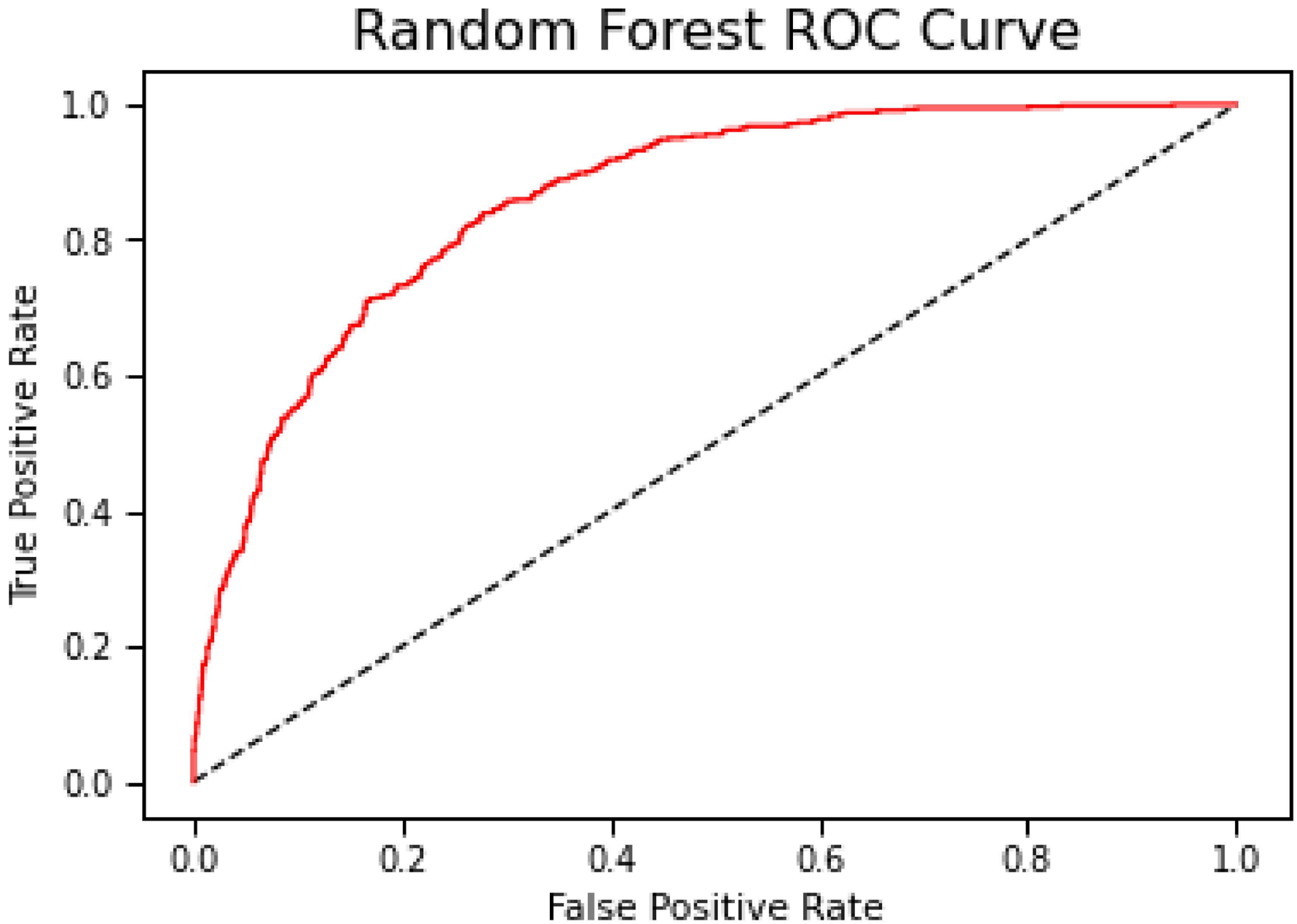
```
              precision    recall  f1-score   support

           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
```

In [58]:

```
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()
```



In [59]:
```
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

In [60]:
```
lr_model = LogisticRegression()
lr_model.fit(X_train,y_train)
```

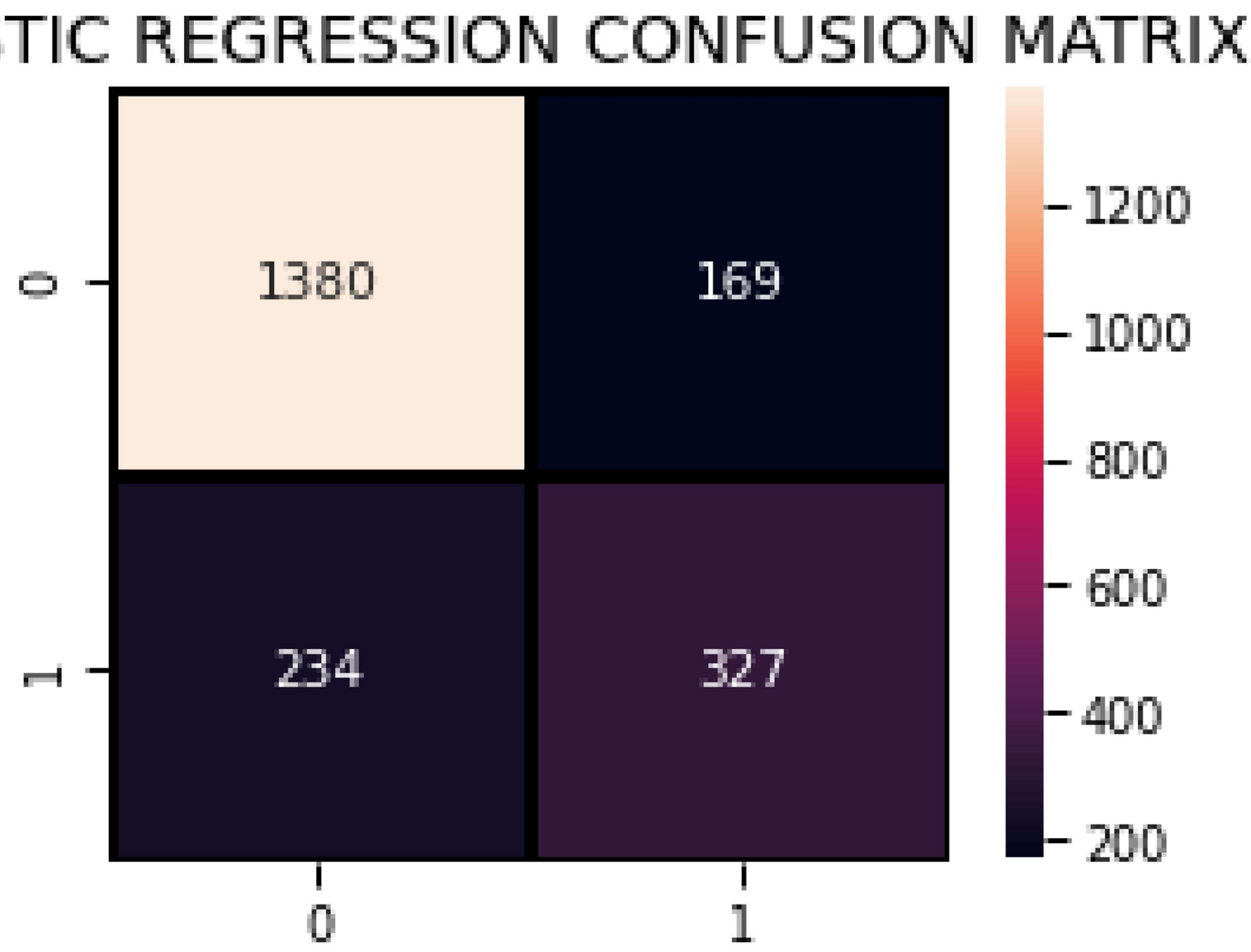
```
accuracy_lr = lr_model.score(X_test,y_test)
print("Logistic Regression accuracy is :",accuracy_lr)
```

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

In [61]:
```
lr_pred= lr_model.predict(X_test)
report = classification_report(y_test,lr_pred)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.86      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      0.81      2110
```

In [62]:
```
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()
```



In [63]:
```
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

In [64]:
```
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 : 0.7260663507109004
```

Decision tree gives very low score.

In [65]:
```
print(classification_report(y_test, predictdt_y))
```

```
              precision    recall  f1-score   support

           0       0.82      0.81      0.81      1549
           1       0.49      0.51      0.50       561

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

AdaBoost Classifier

In [66]:
```
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)
```

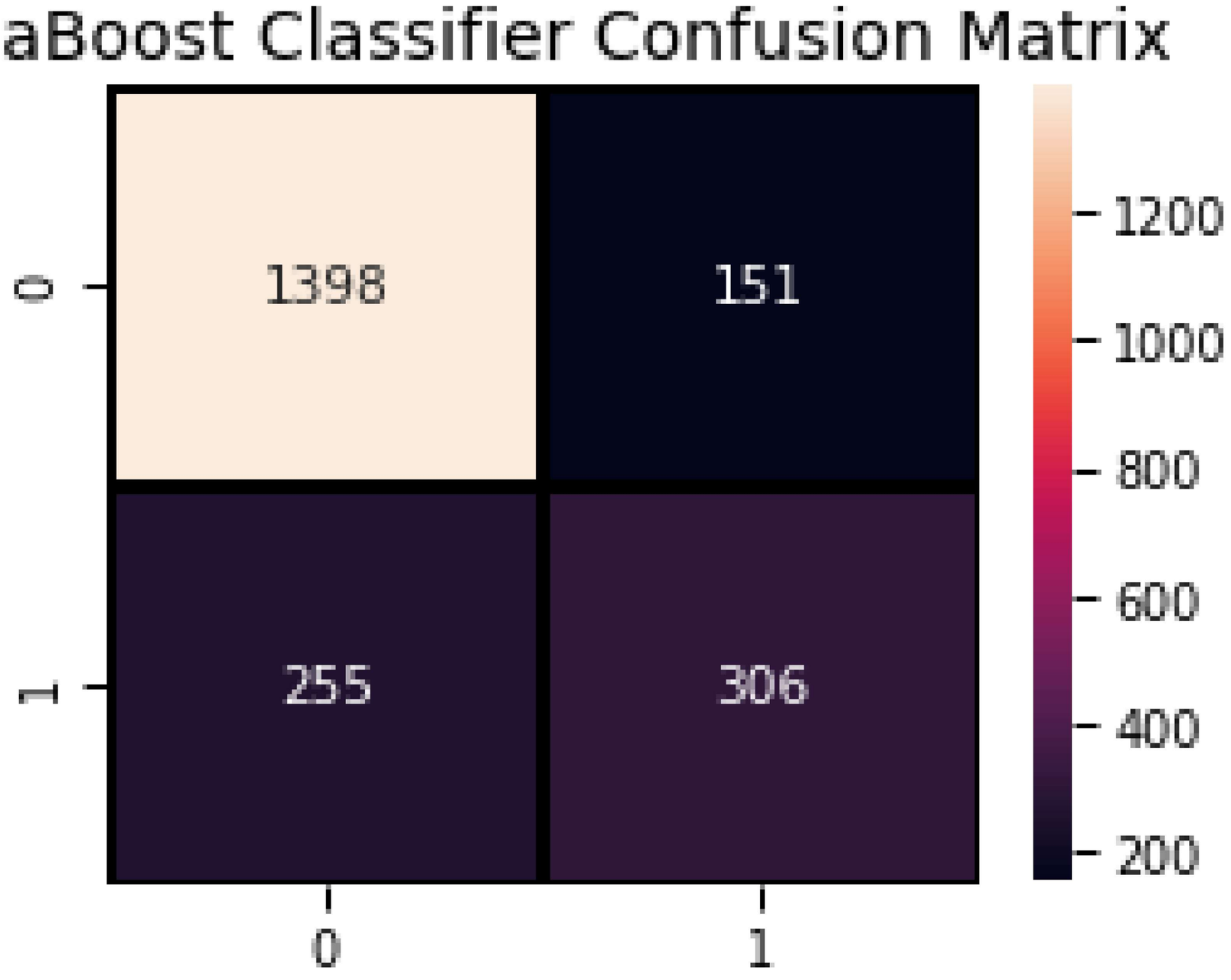
```
AdaBoost Classifier accuracy
```

Out[66]:
```
0.8075829383886256
```

In [67]:
```
print(classification_report(y_test, a_preds))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1549
           1       0.67      0.55      0.60       561

    accuracy                           0.81      2110
   macro avg       0.76      0.72      0.74      2110
weighted avg       0.80      0.81      0.80      2110
```

In [68]:
```
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

In [69]:
```
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
```

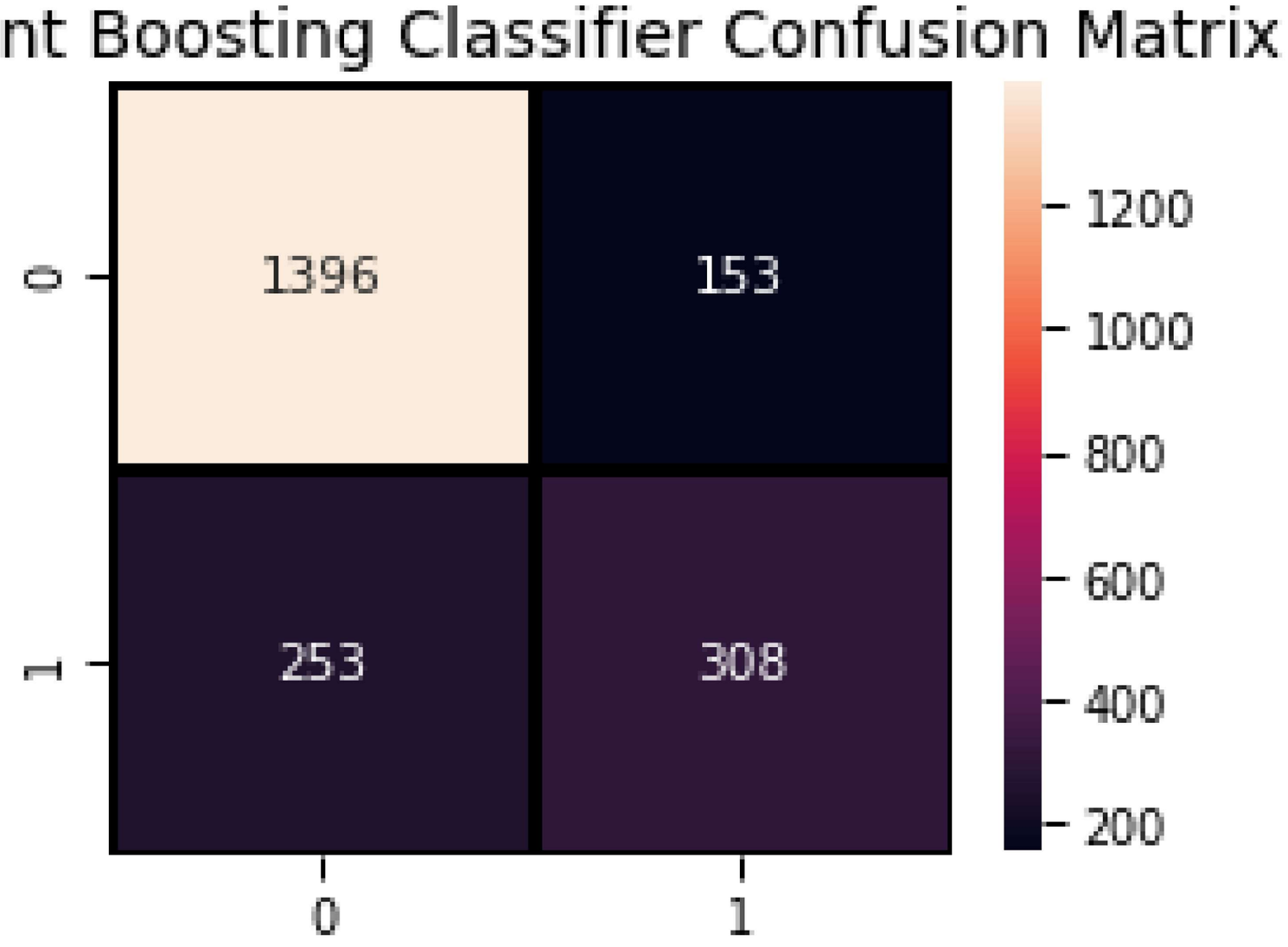
```
gb_pred = gb.predict(X_test)
print("Gradient Boosting Classifier", accuracy_score(y_test, gb_pred))
```

```
Gradient Boosting Classifier 0.8075829383886256
```

In [70]:
```
print(classification_report(y_test, gb_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1549
           1       0.67      0.55      0.60       561

    accuracy                           0.81      2110
   macro avg       0.76      0.73      0.74      2110
weighted avg       0.80      0.81      0.80      2110
```

In [71]:
```
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

Predict the final model based on the highest majority of voting and check it's score.

In [72]:
```
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)
```

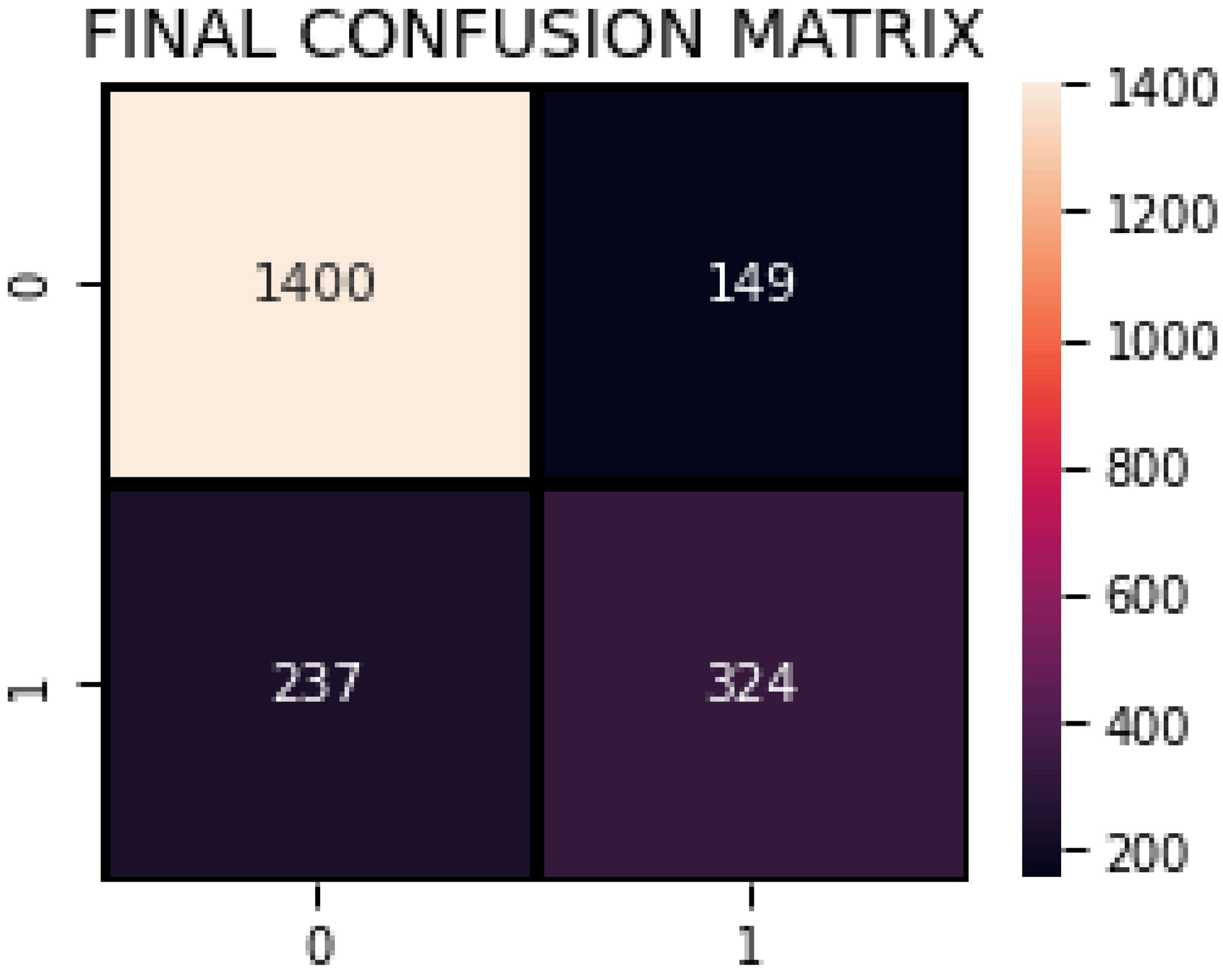
```
print("Final Accuracy Score ")
print(accuracy_score(y_test, predictions))
```

```
Final Accuracy Score
0.8170616113744076
```

In [73]:
```
print(classification_report(y_test, predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1549
           1       0.68      0.58      0.63       561

    accuracy                           0.82      2110
   macro avg       0.77      0.74      0.75      2110
weighted avg       0.81      0.82      0.81      2110
```

In [74]:
```
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()
```



The confusion matrix shows that there are a total of 1549 actual non-churn values, out of which the algorithm predicts 1400 correctly as non-churn and 149 as churn.

For the actual churn values totaling 561, the algorithm predicts 237 as non-churn and 324 as churn.

Customer churn significantly impacts a firm's profitability. To counter this, various strategies can be implemented. Understanding customers is key, identifying those at risk of churning, and working to enhance their satisfaction. Prioritizing improved customer service is fundamental in addressing this issue. Additionally, building customer loyalty through tailored experiences and specialized service aids in reducing churn rates. Some firms conduct surveys among churned customers to comprehend their reasons for leaving, enabling a proactive approach to prevent future churn.