1. Introduction

What is Customer Churn

- Customer churn is defined as when customers or subscribers discontinue doing business with a firm or service.
- The telecommunications business has an annual churn rate of 15-25 percent in this highly competitive market.
- Customer churn is a critical metric because it is much less expensive to retain existing customers than it is to acquire new customers.
- To detect early signs of potential churn, one must first develop a holistic view of the customers and their interactions across numerous channels, including store/branch visits, product purchase histories, customer service calls, Web-based transactions, and social media interactions, to mention a few.

Objectives

I will explore the data and try to answer some questions like:

- What's the % of Churn Customers and customers that keep in with the active services?
- Is there any patterns in Churn Customers based on the gender?
- Is there any patterns/preference in Churn Customers based on the type of service provided?
- What's the most profitable service types?
- Which features and services are most profitable?

2. Loading Libraries and Data

In [1]:
```
import numpy as np
import pandas as pd
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 MinMaxScaler
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,
```

In [3]:

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

3. Understanding the Data

In [4]:

```
df.head(10)
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	Inte
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	
4	9237-HQITU	Female	0	No	No	2	Yes	No	
5	9305-CDSKC	Female	0	No	No	8	Yes	Yes	
6	1452-KIOVK	Male	0	No	Yes	22	Yes	Yes	
7	6713-OKOMC	Female	0	No	No	10	No	No phone service	
8	7892-POOKP	Female	0	Yes	No	28	Yes	Yes	
9	6388-TABGU	Male	0	No	Yes	62	Yes	No	

10 rows × 21 columns

- Each row represent a customer and columns are customers' attributes described on the columns Metadata.

In [5]:
```
print('\n'.join(list(df.columns)))
```

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

The data set includes information about:

- **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

In [6]:
```
print(df.shape)
```

```
(7043, 21)
```

In [7]:
```
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
```

- The target will be **Churn**

In [8]:
```
object_types = [i for i in df.columns if df[i].dtype=='object']
```

In [9]:
```
object_types
```

Out[9]:
```
['customerID',
 'gender',
 'Partner',
 'Dependents',
 'PhoneService',
 'MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaperlessBilling',
 'PaymentMethod',
 'TotalCharges',
 'Churn']
```

4. Checking Missing Values

In [10]:
```
df.isnull().sum()
```

Out[10]:
```
customerID          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
```

- There is no Null value in dataset.

5. Data Manipulation

In [11]:
```
df.head()
```

Out[11]:

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

5 rows × 21 columns

In [12]:
```
# Customer ID is not important
df.drop(['customerID'],axis=1, inplace=True)
```

```
df.head()
```

Out[12]:

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

- The type of values in TotalCharges is object but we should convert them to numerical values.

In [13]:
```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
```

- **'coerce'** is an option that tells Pandas to replace any values that cannot be converted to numeric with NaN (Not a Number).

In [14]:
```
df.isnull().sum()
```

Out[14]:
```
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
```

- Now, we have 11 missing values in TotalCharges Column.

In [15]:
```
df[df['TotalCharges'].isnull()]
```

Out[15]:

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

- From above table, we see that tenure columns is 0 even though MonthlyCharges column is not 0 or empty.
- Let's check Tenure column for more 0 values.

In [16]:
```
df[df['tenure']==0].index
```

Out[16]:
```
Int64Index([488, 753, 936, 1082, 1340, 3331, 3826, 4380, 5218, 6670, 6754], dtype='in
t64')
```

In [17]:
```
df[(df['tenure']==0) & pd.isna(df['TotalCharges'])]
```

Out[17]:

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

- Deleting these 11 rows will not affect the data.

In [18]:

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

Out[18]:

```
Int64Index([], dtype='int64')
```

In [19]:

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

Out[19]:

```
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 [20]:

```
print(df['SeniorCitizen'].unique())
print(df['SeniorCitizen'].dtype)
```

```
[0 1]
int64
```

In [21]:

```
df['SeniorCitizen'] = df['SeniorCitizen'].map({0:'No', 1:'Yes'})
print(df['SeniorCitizen'].unique())
print(df['SeniorCitizen'].dtype)
```

```
['No' 'Yes']
object
```

In [22]:

```
df.head()
```

Out[22]:

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

In [23]:

```
df['InternetService'].describe()
```

Out[23]:
```
count              7032
unique                3
top         Fiber optic
freq               3096
Name: InternetService, dtype: object
```

In [24]:
```
df.describe()
```

Out[24]:

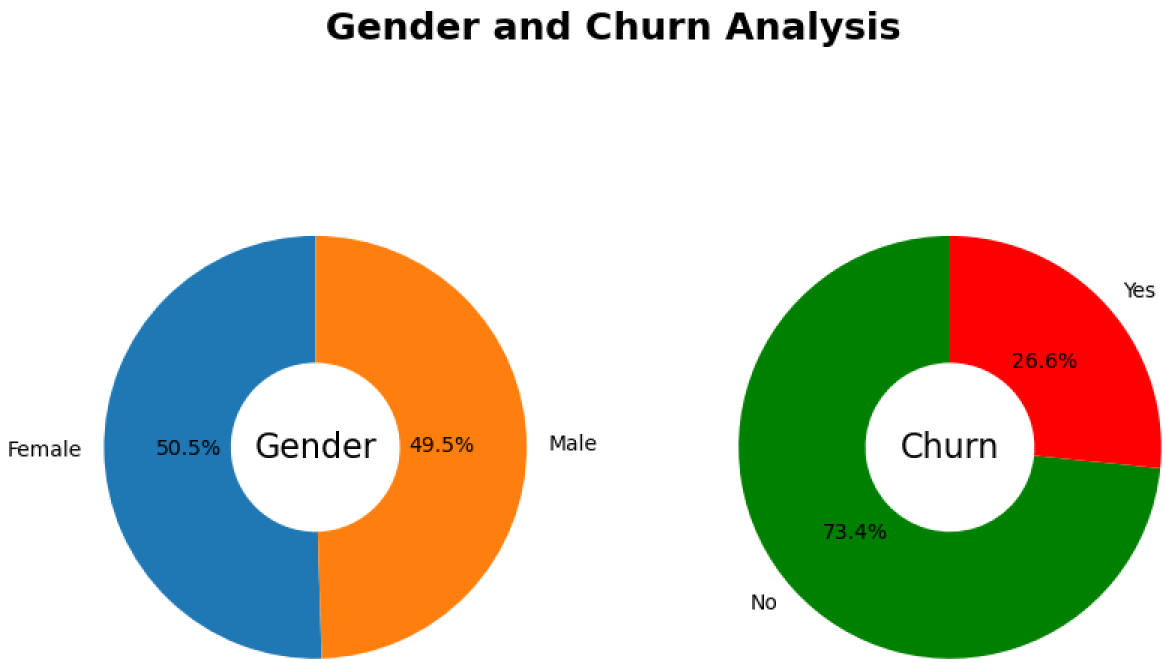
	tenure	MonthlyCharges	TotalCharges
count	7032.000000	7032.000000	7032.000000
mean	32.421786	64.798208	2283.300441
std	24.545260	30.085974	2266.771362
min	1.000000	18.250000	18.800000
25%	9.000000	35.587500	401.450000
50%	29.000000	70.350000	1397.475000
75%	55.000000	89.862500	3794.737500
max	72.000000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

6. Data Visualization

Firstly, checking relationship between gender and Churn.

In [25]:
```
genders = list(df['gender'].unique())
churn_status = list(df['Churn'].unique())

#gender numbers

fig, axs = plt.subplots(1,2, figsize=(10,6))

axs[0].pie(df['gender'].value_counts(), labels=genders, autopct='%1.1f%%', startangle=
axs[0].text(0, 0, "Gender", ha='center', va='center', fontsize=16)  # Add title inside

axs[1].pie(df['Churn'].value_counts(), labels = churn_status, autopct='%1.1f%%', start
           wedgeprops=dict(width=0.6), colors = ['green','red'])
axs[1].text(0, 0, "Churn", ha='center', va='center', fontsize=16)

fig.suptitle("Gender and Churn Analysis", fontsize=18, fontweight='bold')
```

Out[25]:
```
Text(0.5, 0.98, 'Gender and Churn Analysis')
```



- 26.6 % of customers left you firm.
- Customers are 49.5% female and 50.55 male.

In [26]:
```
df['Churn'][df['Churn']=='No'].groupby(by=df['gender']).count()
```

Out[26]:
```
gender
Female    2544
Male      2619
Name: Churn, dtype: int64
```

In [27]:
```
df['Churn'][df['Churn']=='Yes'].groupby(by=df['gender']).count()
```

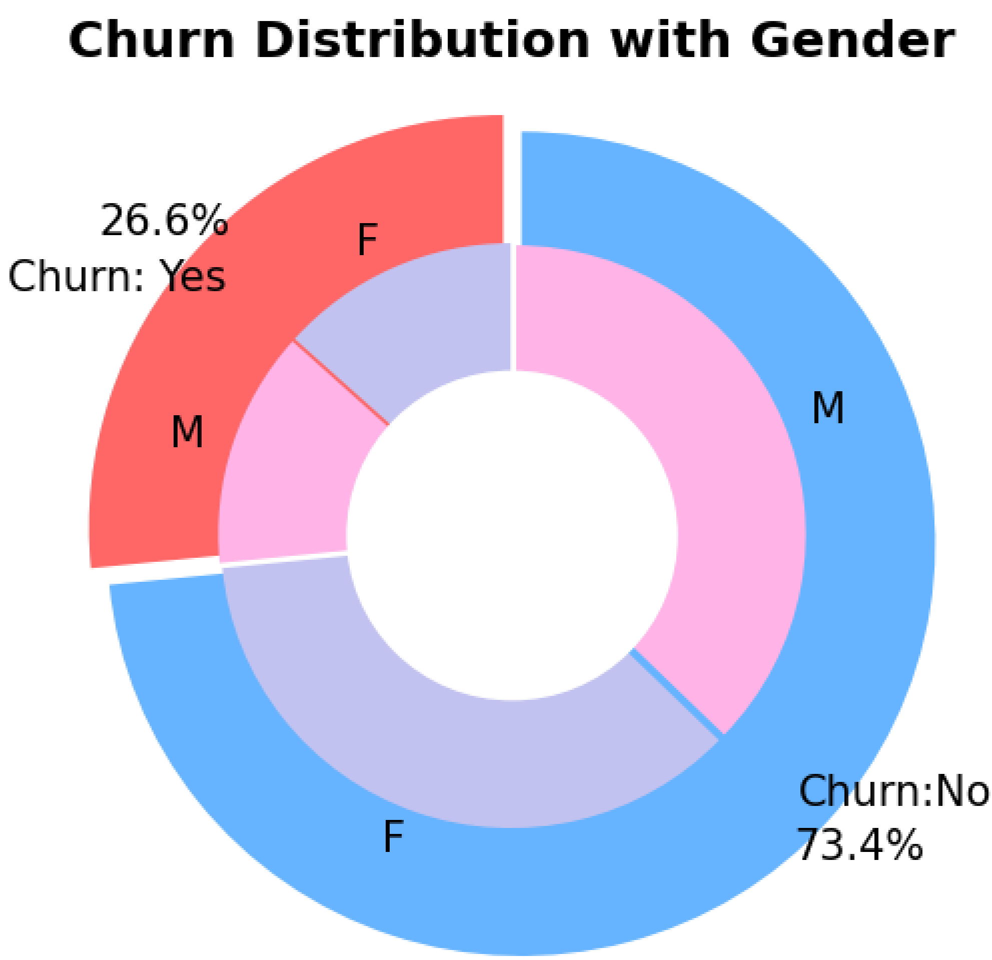
Out[27]:
```
gender
Female    939
Male      930
Name: Churn, dtype: int64
```

In [28]:
```
df['Churn'].value_counts()
```

Out[28]:
```
No     5163
Yes    1869
Name: Churn, dtype: int64
```

In [29]:
```
plt.figure(figsize=(6,6))
labels_churn =["Churn: Yes","Churn:No"]
values = [1869,5163]
labels_gender = ["F","M","F","M"]
sizes_gender = [939,930 , 2544,2619] # Female:Yes, Male:Yes, Female:No, Male:No
colors_churn = ['#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_churn, autopct='%1.1f%%', pctdistance=1.1, labeldistance
```

```
        colors = colors_churn, startangle=90, explode=explode, radius=10, textprops=tex

plt.pie(sizes_gender, labels=labels_gender, colors=colors_gender,startangle=90, exploc
        textprops=textprops)

centre_circle = plt.Circle((0, 0), radius=4, color='white', fc='white', linewidth=0)
fig = plt.gcf()
fig.gca().add_artist(centre_circle)

plt.axis('equal')

plt.title("Churn Distribution with Gender", fontsize=18, fontweight='bold')
plt.show()
```

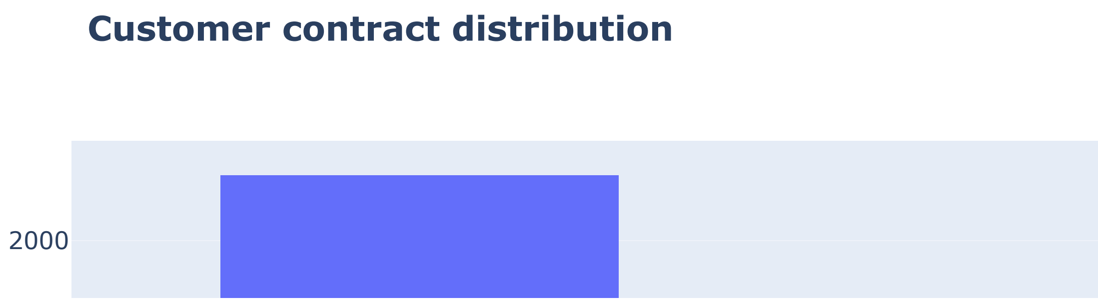


- Both genders behaved in similar fashion when it comes to migrating to another service provider/firm.

Relationship between Churn and Customer Contract

In [30]:
```
fig = px.histogram(df, x='Churn', color='Contract', barmode="group", title="<b>Custome
font_dict = {
    'family': 'Times New Roman',  # Change the font family
    'size': 22,         # Change the font size
    'color': 'black'    # Change the font color
}
```

```
fig.update_layout(
    xaxis_title_font=font_dict,
    yaxis_title_font=font_dict
)
fig.show()
```

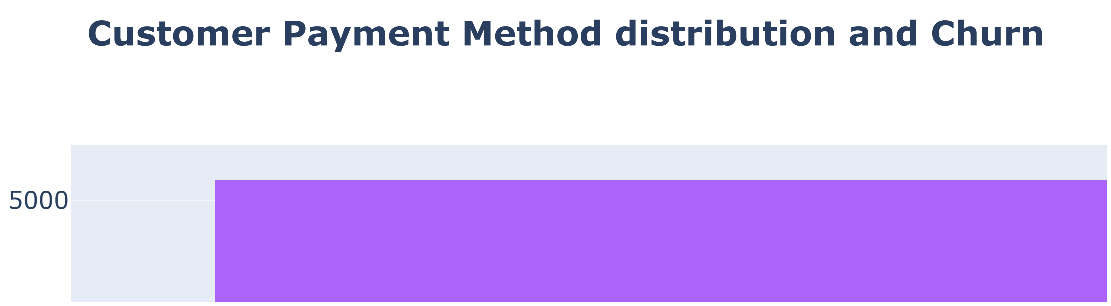


- Customers who have a Month-to-Month Contract are more prone to churning.
- About 43% of customers with Month-to-Month Contract decided to leave the firm whereas this is 13% for customers with One Year Contract and 3% for Two Year Contract

Relationship between Churn and Customer Payment Method

In [31]:
```
fig = px.histogram(df, x = 'Churn', color='PaymentMethod',
                   title="<b>Customer Payment Method distribution and Churn</b>")
font_dict = {
    'family': 'Times New Roman',  # Change the font family
    'size': 22,          # Change the font size
    'color': 'black'     # Change the font color
}
fig.update_layout(
    xaxis_title_font=font_dict,
    yaxis_title_font=font_dict
```

```
)
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.

Relationship between Churn and Internet Service Customers use

In [32]:
```
df['InternetService'].unique()
```

Out[32]:
```
array(['DSL', 'Fiber optic', 'No'], dtype=object)
```

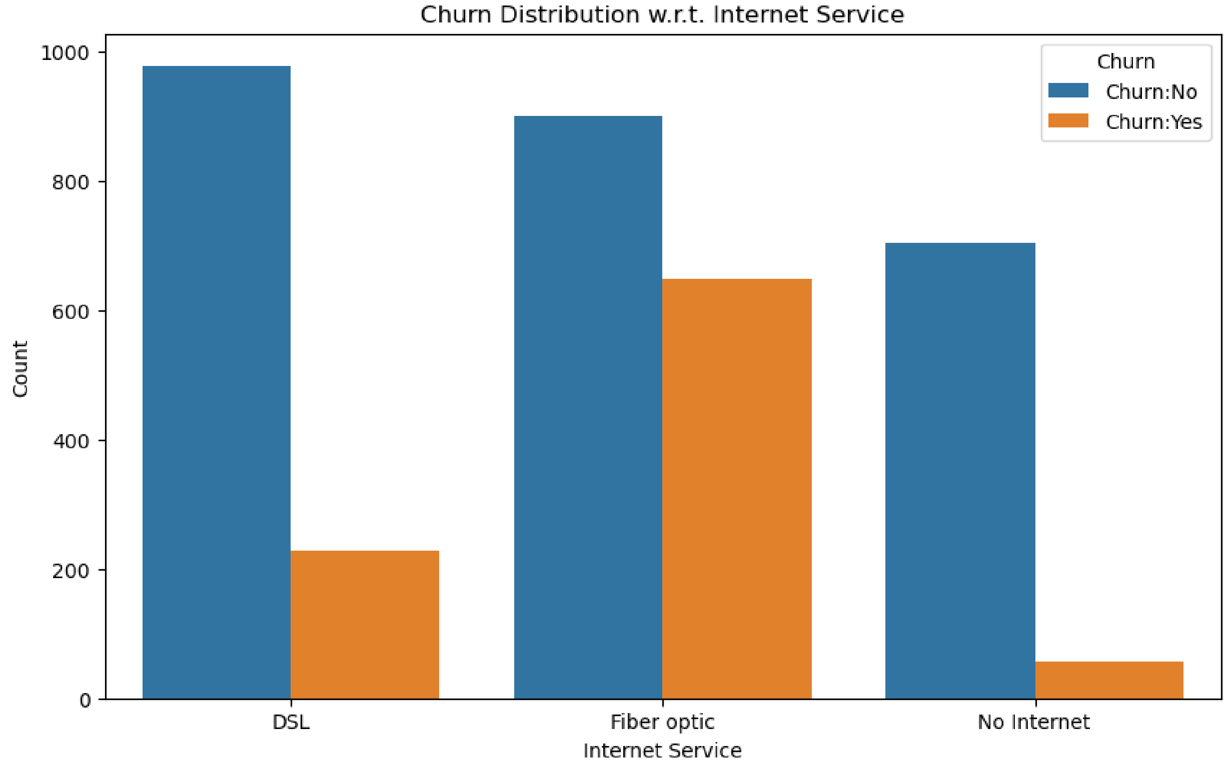
In [33]:
```
df[df['gender']=='Female'][['InternetService','Churn']].value_counts()
```

Out[33]:
```
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 [34]:
```
df[df['gender']=='Male'][['InternetService','Churn']].value_counts()
```

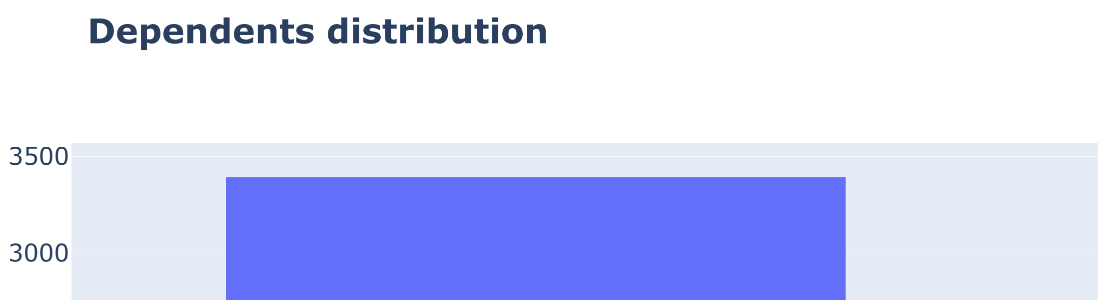
Out[34]:
```
InternetService  Churn
DSL              No       992
Fiber optic      No       910
No               No       717
Fiber optic      Yes      633
DSL              Yes      240
No               Yes       57
dtype: int64
```

In [35]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Sample data
data = {
    'InternetService': ['DSL', 'DSL','DSL', 'DSL', 'Fiber optic', 'Fiber optic', 'Fibe
                        'No Internet', 'No Internet', 'No Internet', 'No Internet'],
    'Gender': ['Female', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Male',
    'Churn': ['Churn:No', 'Churn:Yes', 'Churn:No', 'Churn:Yes', 'Churn:No', 'Churn:Yes
              'Churn:No', 'Churn:Yes', 'Churn:No', 'Churn:Yes'],
    'Count': [965, 219, 992, 240, 889, 664, 910, 633, 690, 56, 717, 57]
}

data = pd.DataFrame(data)

# Create the barplot using Seaborn
plt.figure(figsize=(10, 6))
sns.barplot(data=data, x='InternetService', y='Count', hue='Churn', ci=None)

# Set plot title and labels
plt.title("Churn Distribution w.r.t. Internet Service")
plt.xlabel('Internet Service')
plt.ylabel('Count')

# Show the legend
plt.legend(title='Churn')

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



- The customers who use Fiber optic have high churn rate, this might suggest a dissatisfaction with this type of internet service.
- Customers having DSL service are majority in number and have less churn rate compared to Fibre optic service.

In [36]:
```
fig = px.histogram(df, x='Churn', color='Dependents', barmode='group', title="<b>Deper
fig.show()
```

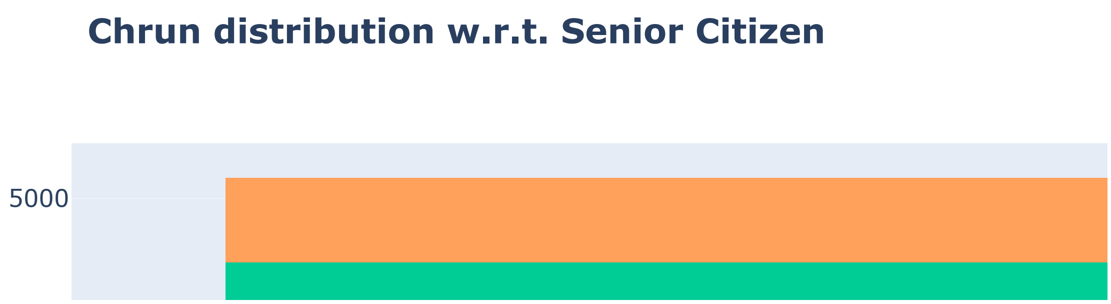


- Customers without dependents are more likely to churn

In [37]:
```
fig = px.histogram(df, x='Churn', color='Partner', barmode='group', title="<b>Chrun di
fig.show()
```

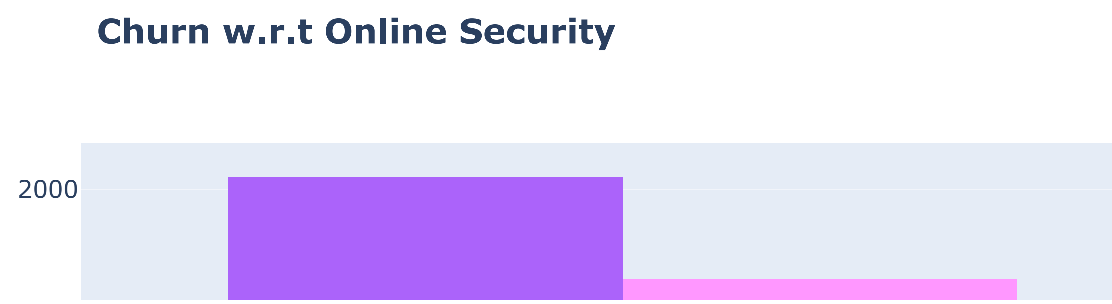
- Customers that doesn't have partners are more likely to churn

In [38]:
```
color_map = {"Yes": '#FFA15A', "No": '#00CC96'}
fig = px.histogram(df, x="Churn", color="SeniorCitizen", title="<b>Chrun distribution
                  color_discrete_map=color_map)
fig.show()
```



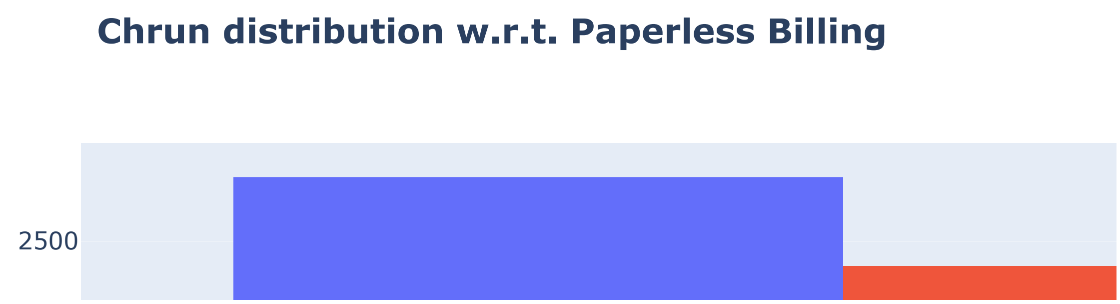
- It can be observed that the fraction of senior citizen is very less.
- Most of the senior citizens churn. However, it is like that because of the high between senior citizen ratio.

In [39]:
```
color_map = {"Yes": "#FF97FF", "No": "#AB63FA"}
fig = px.histogram(df, x='Churn', color='OnlineSecurity', color_discrete_map=color_map
                   title="<b>Churn w.r.t Online Security</b>")
fig.show()
```



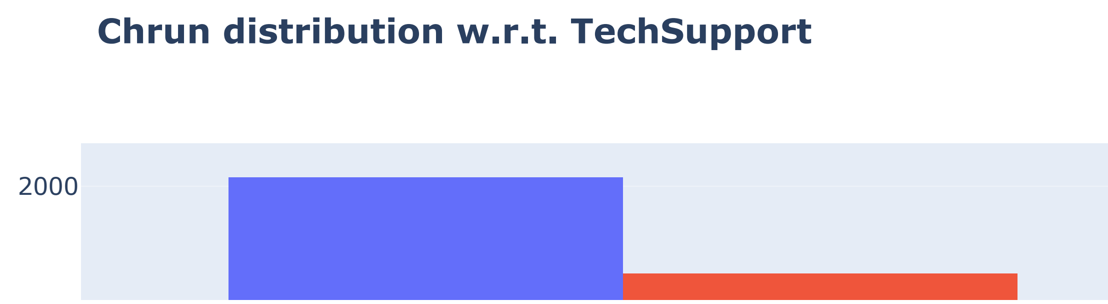
- Most customers churn in the absence of online security

In [40]:
```
fig = px.histogram(df, x='Churn', color='PaperlessBilling', barmode='group',
                   title="<b>Chrun distribution w.r.t. Paperless Billing</b>")
fig.show()
```



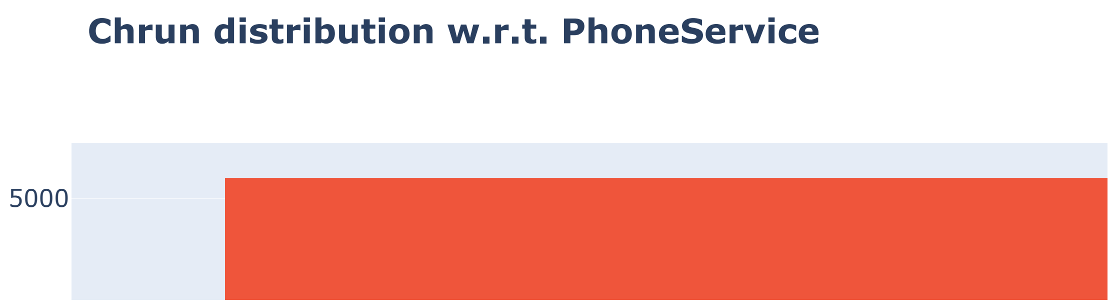
- Customers with Paperless Billing are most likely to churn.

In [41]:
```
fig = px.histogram(df, x='Churn', color='TechSupport', barmode='group',
                   title="<b>Chrun distribution w.r.t. TechSupport</b>")
fig.show()
```



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

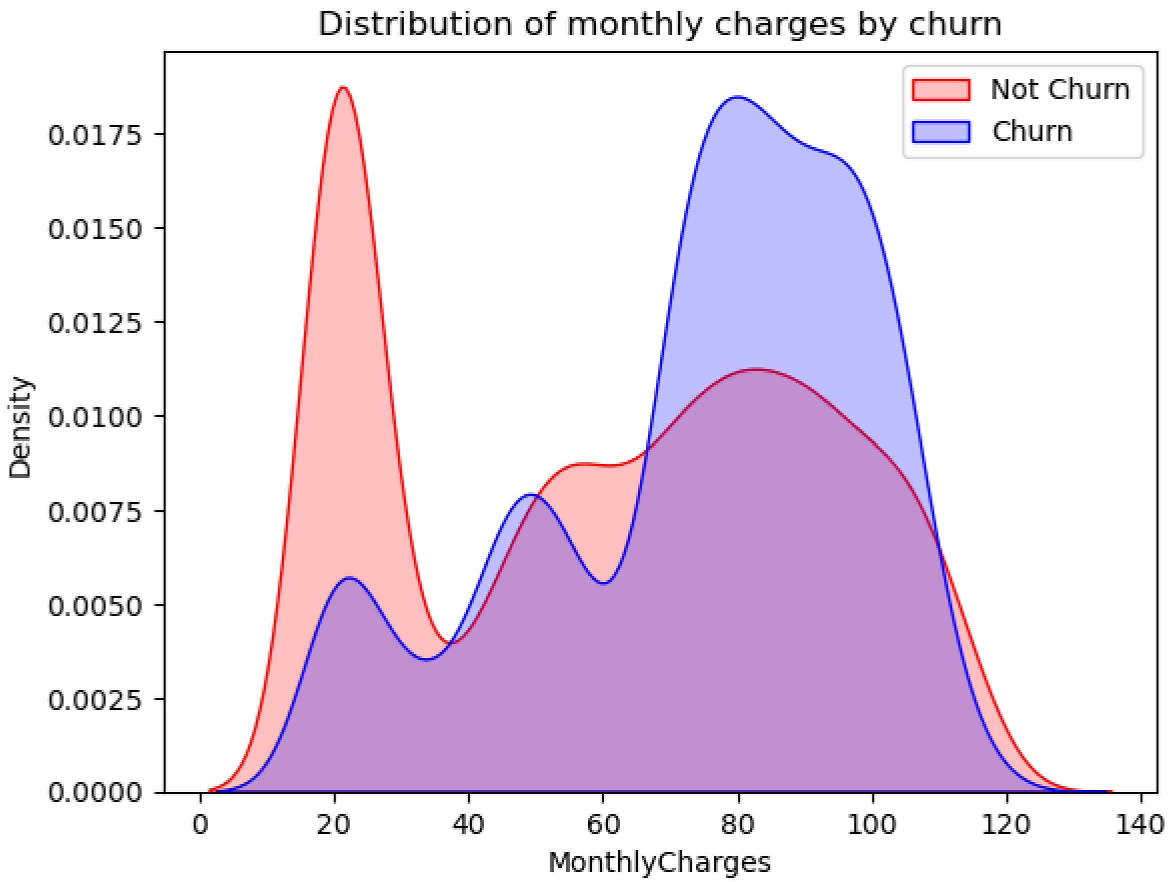
In [42]:
```
fig = px.histogram(df, x='Churn', color='PhoneService',
                   title="<b>Chrun distribution w.r.t. PhoneService</b>")
fig.show()
```



- Only a small percentage of customers do not have phone service, and of those customers, one-third are more likely to churn.

In [43]:
```
ax = sns.kdeplot(df.MonthlyCharges[df['Churn']=='No'], color='red', shade=True)
ax = sns.kdeplot(df.MonthlyCharges[df['Churn']=='Yes'], color='blue', shade=True, ax=a
#ax=ax: This parameter tells Seaborn to plot the new KDE plot on the same axes (ax) th
#This ensures that both KDE plots are overlaid on the same plot.
ax.legend(['Not Churn', 'Churn'])
ax.set_title('Distribution of monthly charges by churn')
```

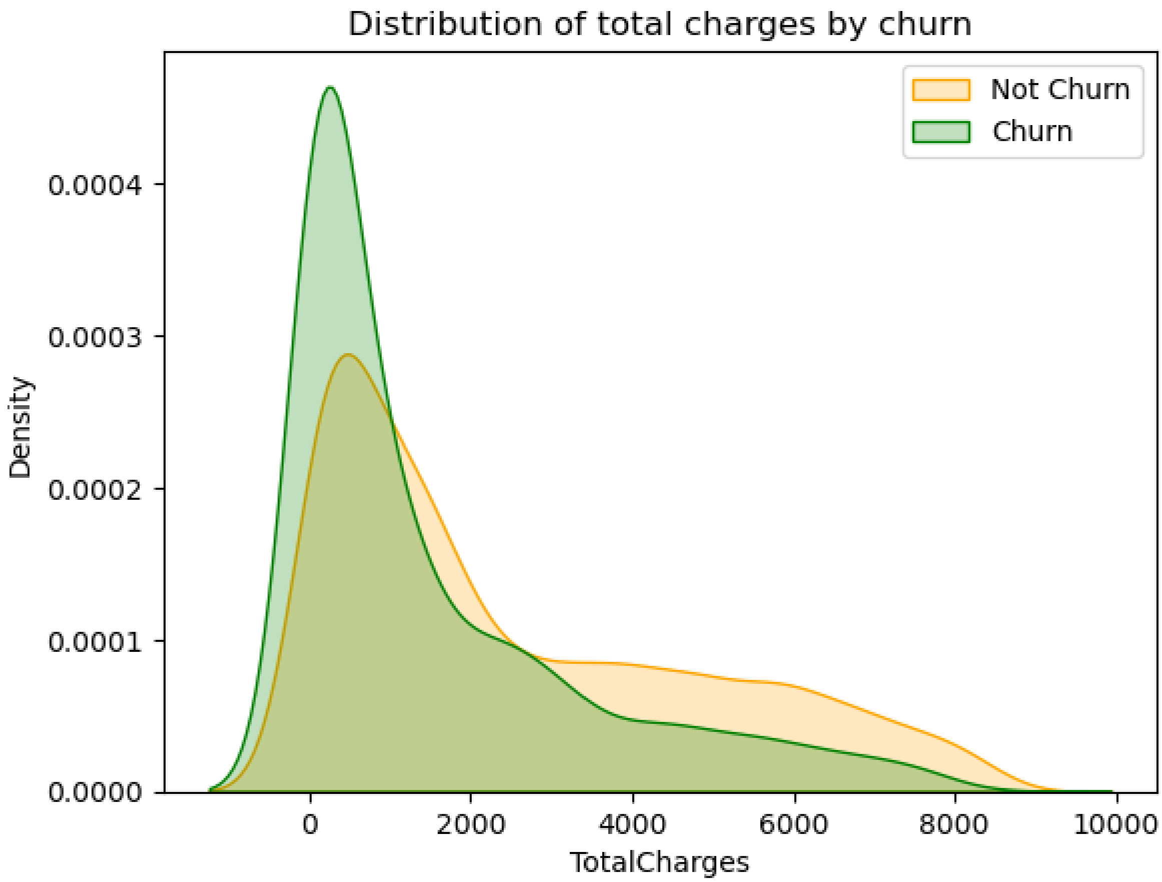
Out[43]:
```
Text(0.5, 1.0, 'Distribution of monthly charges by churn')
```



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

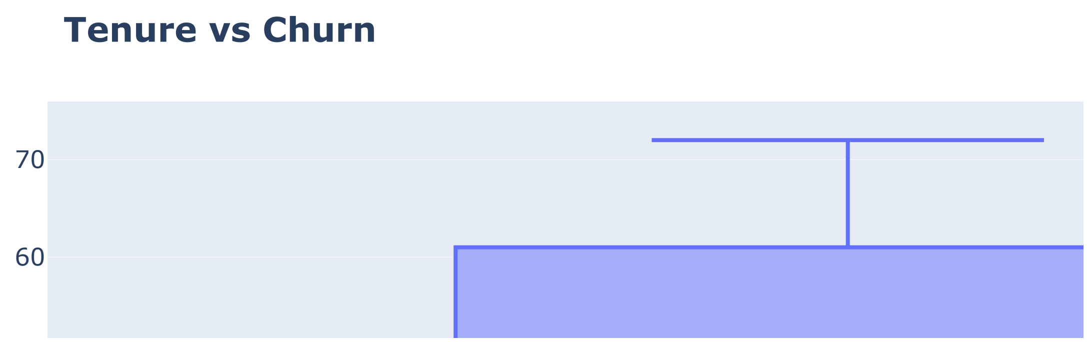
In [44]:
```
ax = sns.kdeplot(df.TotalCharges[df['Churn']=='No'], color='orange', shade=True)
ax = sns.kdeplot(df.TotalCharges[df['Churn']=='Yes'], color='green', shade=True, ax=ax
#ax=ax: This parameter tells Seaborn to plot the new KDE plot on the same axes (ax) th
#This ensures that both KDE plots are overlaid on the same plot.
ax.legend(['Not Churn', 'Churn'])
ax.set_title('Distribution of total charges by churn')
```

Out[44]:
```
Text(0.5, 1.0, 'Distribution of total charges by churn')
```



- The central tendency of the distributions are similar, the peaks of the curves in similar positions.
- The spread of the distributions is similar, it implies that the variability in the values is consistent across the two groups.

In [45]:
```
fig = px.box(df, x='Churn', y='tenure')
fig.update_yaxes(title_text='Tenure (Months)')
fig.update_layout(
title = '<b>Tenure vs Churn</b>'
)
fig.show()
```



- New customers are more likely to churn

Before heatmap, I want to describe a few things to make it more understandable.

- The pd.factorize() function assigns a unique integer to each distinct value in a column, effectively converting categorical data into numerical labels.

In [46]:
```
# For example:
df.apply(lambda x: pd.factorize(x))
```

Out[46]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLi
0	[0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, ...	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	[0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, ...	[0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, ...	[0, 1, 2, 3, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, ...	[0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, ...	[0, 1, 1, 0, 2, 0, 2, 1, 2, 2,
1	Index(['Female', 'Male'], dtype='object')	Index(['No', 'Yes'], dtype='object')	Index(['Yes', 'No'], dtype='object')	Index(['No', 'Yes'], dtype='object')	Int64Index([1, 34, 2, 45, 8, 22, 10, 28, 62...	Index(['No', 'Yes'], dtype='object')	Index(ph service', ' 'Yes'], dty

In [47]:

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

Out[47]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServi
0	0	0	0	0	0	0	0	
1	1	0	1	0	1	1	1	
2	1	0	1	0	2	1	1	
3	1	0	1	0	3	0	0	
4	0	0	1	0	2	1	1	
...	...	...	...	...	...	...	...	
7038	1	0	0	1	65	1	2	
7039	0	0	0	1	21	1	2	
7040	0	0	0	1	26	0	0	
7041	1	1	0	0	54	1	2	
7042	1	0	1	0	33	1	1	

7032 rows × 20 columns

In [48]:

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

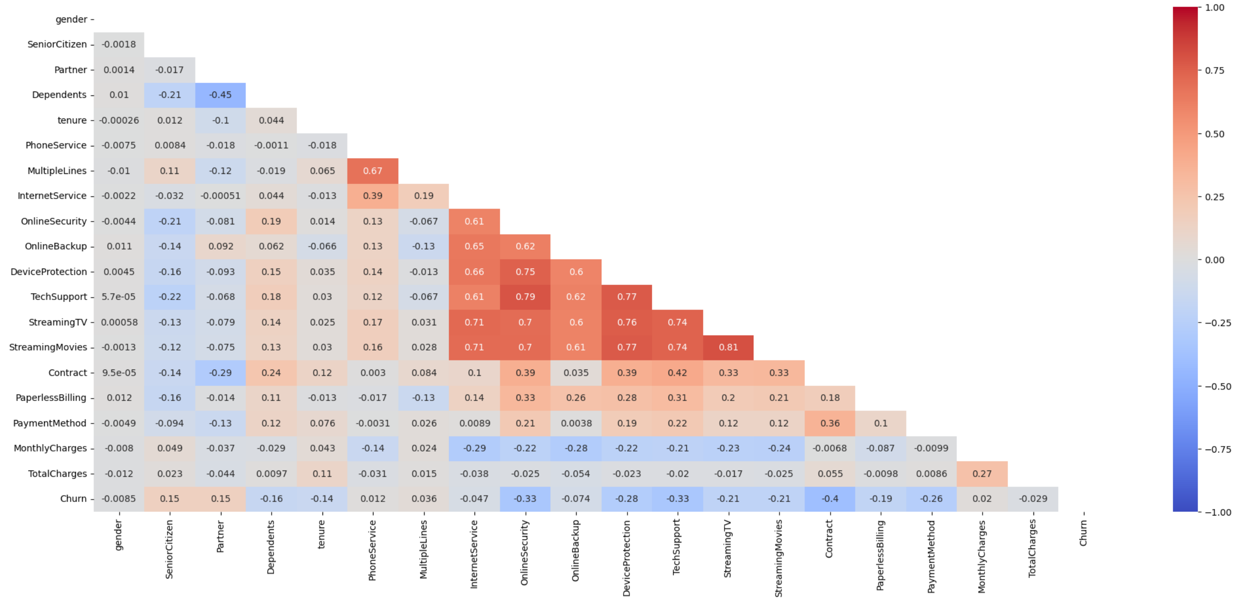
Out[48]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleL
gender	1.000000	-0.001819	0.001379	0.010349	-0.000265	-0.007515	-0.010
SeniorCitizen	-0.001819	1.000000	-0.016957	-0.210550	0.012240	0.008392	0.113
Partner	0.001379	-0.016957	1.000000	-0.452269	-0.100513	-0.018397	-0.118
Dependents	0.010349	-0.210550	-0.452269	1.000000	0.044138	-0.001078	-0.019
tenure	-0.000265	0.012240	-0.100513	0.044138	1.000000	-0.017864	0.064
PhoneService	-0.007515	0.008392	-0.018397	-0.001078	-0.017864	1.000000	0.674
MultipleLines	-0.010284	0.113769	-0.118037	-0.019178	0.064580	0.674824	1.000
InternetService	-0.002236	-0.032160	-0.000513	0.044030	-0.012924	0.387266	0.186
OnlineSecurity	-0.004365	-0.210546	-0.081078	0.188889	0.014436	0.125544	-0.066
OnlineBackup	0.011081	-0.144762	0.091536	0.061970	-0.066232	0.129432	-0.130
DeviceProtection	0.004526	-0.156700	-0.093391	0.154819	0.034744	0.138938	-0.012
TechSupport	0.000057	-0.223438	-0.068277	0.179176	0.030489	0.123533	-0.066
StreamingTV	0.000578	-0.129721	-0.079066	0.138809	0.024719	0.171773	0.030
StreamingMovies	-0.001339	-0.120658	-0.075310	0.125086	0.030252	0.164379	0.027
Contract	0.000095	-0.141820	-0.294094	0.240556	0.118664	0.003019	0.084
PaperlessBilling	0.011902	-0.156258	-0.013957	0.110131	-0.013160	-0.016696	-0.133
PaymentMethod	-0.004928	-0.093712	-0.133280	0.124002	0.075533	-0.003106	0.026
MonthlyCharges	-0.008017	0.049154	-0.036518	-0.028706	0.042605	-0.141696	0.024
TotalCharges	-0.012153	0.022949	-0.044214	0.009710	0.112813	-0.030534	0.014
Churn	-0.008545	0.150541	0.149982	-0.163128	-0.143101	0.011691	0.036

Now, heatmap :

In [49]:

```
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,
                 vmax=1, vmin=-1, cmap='coolwarm')
```



7. Data Preprocessing

In [50]:
```
df_copy = df.copy()
```

In [51]:
```
df.head()
```

Out[51]:

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

In [52]:
```
for _ in df.columns:
    if df[_].dtype =='object':

        print(_ + ':')
        print(df[_].nunique())
```

```
gender:
2
SeniorCitizen:
2
Partner:
2
Dependents:
2
PhoneService:
2
MultipleLines:
3
InternetService:
3
OnlineSecurity:
3
OnlineBackup:
3
DeviceProtection:
3
TechSupport:
3
StreamingTV:
3
StreamingMovies:
3
Contract:
3
PaperlessBilling:
2
PaymentMethod:
4
Churn:
2
```

- For object columns:
  - if there is 2 unique values, LabelEncoding will be applied.
  - if there is more than 2, OneHotEncoding will be applied.

In [53]:
```
le_columns = [i for i in df.columns if df[i].dtype == 'object' and df[i].nunique() ==
ohe_columns = [i for i in df.columns if df[i].dtype == 'object' and df[i].nunique() >
```

In [54]:
```
ohe_columns
```

Out[54]:
```
['MultipleLines',
 'InternetService',
 'OnlineSecurity',
 'OnlineBackup',
 'DeviceProtection',
 'TechSupport',
 'StreamingTV',
 'StreamingMovies',
 'Contract',
 'PaymentMethod']
```

In [55]:
```
le_columns
```

Out[55]:
```
['gender',
 'SeniorCitizen',
 'Partner',
 'Dependents',
 'PhoneService',
 'PaperlessBilling',
 'Churn']
```

In [56]:
```
# LabelEncoding
for _ in le_columns:
    df[_] =LabelEncoder().fit_transform(df[_])
```

In [57]:
```
df.head()
```

Out[57]:

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

In [58]:
```
#OneHotEncoding with Pandas
df = pd.get_dummies(df, columns=ohe_columns)
```

In [59]:
```
df.head()
```

Out[59]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	PaperlessBilling	MonthlyCharg
0	0	0	1	0	1	0	1	29.8
1	1	0	0	0	34	1	0	56.9
2	1	0	0	0	2	1	1	53.8
3	1	0	0	0	45	0	0	42.3
4	0	0	0	0	2	1	1	70.7

5 rows × 41 columns

In [60]:
```
df.shape
```

Out[60]:
```
(7032, 41)
```

- After encoding, we have 20 extra columns

In [61]:
```
# Let's check correlation of Churn with other features

df.corr()['Churn'].sort_values(ascending=False)
```

Out[61]:
```
Churn                                      1.000000
Contract_Month-to-month                    0.404565
OnlineSecurity_No                          0.342235
TechSupport_No                             0.336877
InternetService_Fiber optic                0.307463
PaymentMethod_Electronic check             0.301455
OnlineBackup_No                            0.267595
DeviceProtection_No                        0.252056
MonthlyCharges                             0.192858
PaperlessBilling                           0.191454
SeniorCitizen                              0.150541
StreamingMovies_No                         0.130920
StreamingTV_No                             0.128435
StreamingTV_Yes                            0.063254
StreamingMovies_Yes                        0.060860
MultipleLines_Yes                          0.040033
PhoneService                               0.011691
gender                                    -0.008545
MultipleLines_No phone service            -0.011691
MultipleLines_No                          -0.032654
DeviceProtection_Yes                      -0.066193
OnlineBackup_Yes                          -0.082307
PaymentMethod_Mailed check                -0.090773
PaymentMethod_Bank transfer (automatic)   -0.118136
InternetService_DSL                       -0.124141
PaymentMethod_Credit card (automatic)     -0.134687
Partner                                   -0.149982
Dependents                                -0.163128
TechSupport_Yes                           -0.164716
OnlineSecurity_Yes                        -0.171270
Contract_One year                         -0.178225
TotalCharges                              -0.199484
DeviceProtection_No internet service      -0.227578
StreamingMovies_No internet service       -0.227578
StreamingTV_No internet service           -0.227578
InternetService_No                        -0.227578
TechSupport_No internet service           -0.227578
OnlineSecurity_No internet service        -0.227578
OnlineBackup_No internet service          -0.227578
Contract_Two year                         -0.301552
tenure                                    -0.354049
Name: Churn, dtype: float64
```

In [62]:
```
X = df.drop(columns='Churn')  # X is a DataFrame
y= df['Churn'] # y is a Series
```

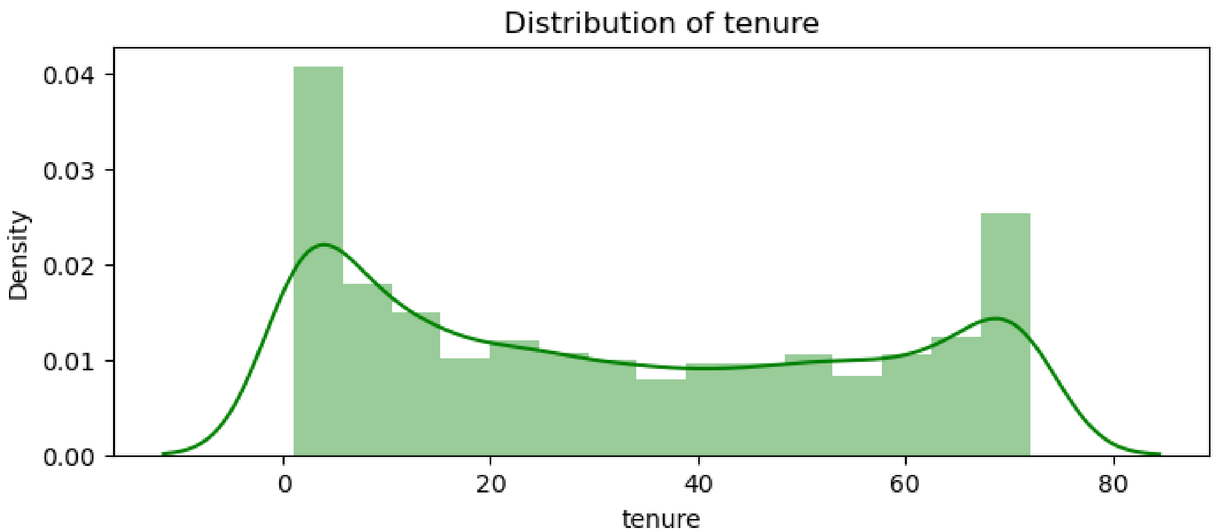
In [63]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state
```

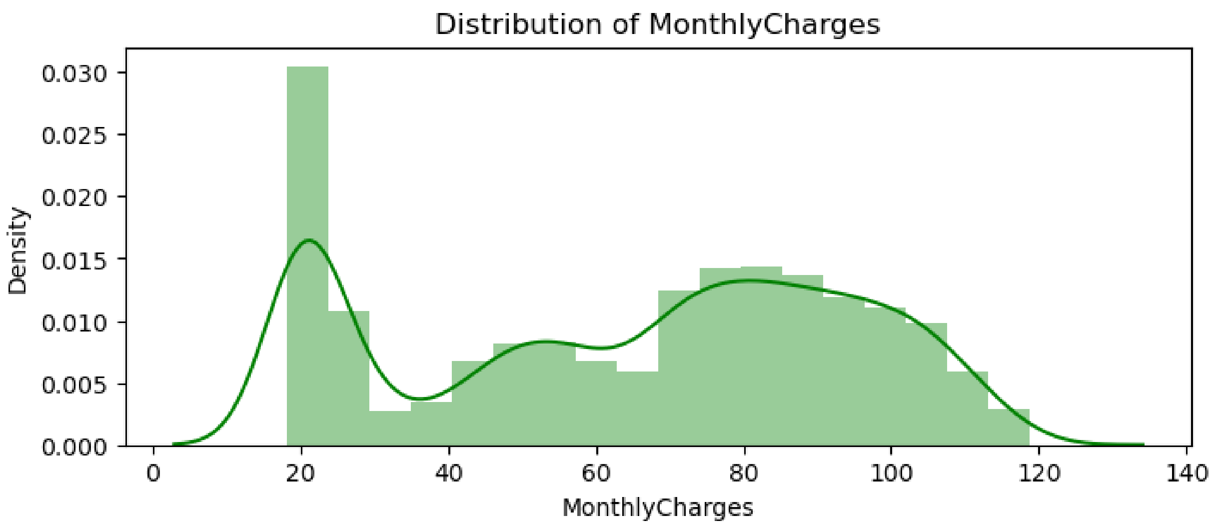
- **stratify** in train_test_split() function is used to ensure that the train and test sets have the same proportions of samples for each class. This is important to do when the target variable is categorical, as it helps to prevent the model from being biased towards one class or another.

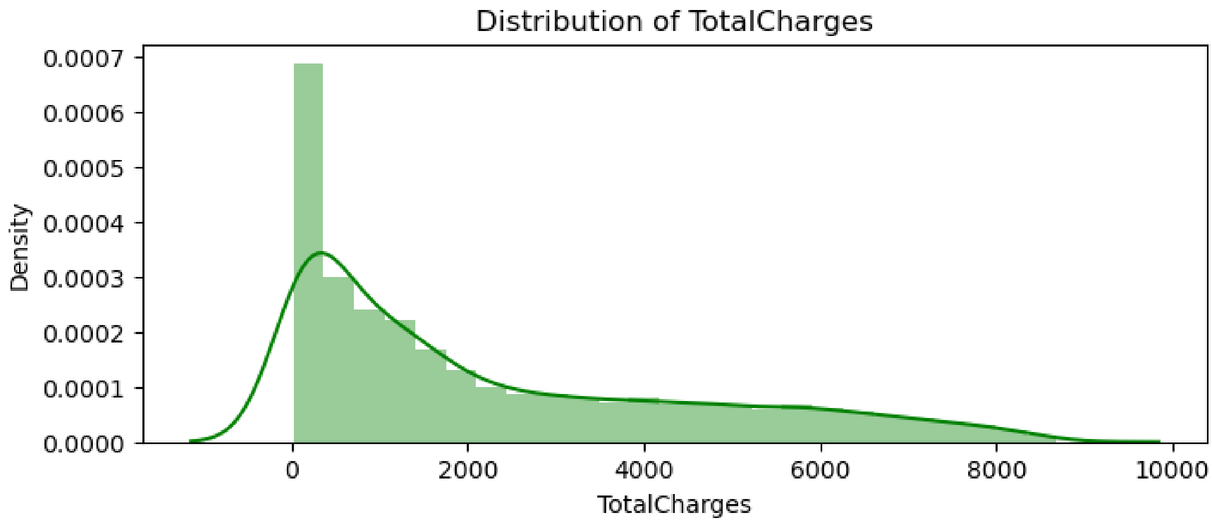
- By setting **stratify=y**, we are telling the train_test_split() function to preserve the same proportions of customers who did churn and did not churn in both the train and test sets

In [64]:
```
def distplot(feature, frame, color='green'):
    plt.figure(figsize=(8,3))
    plt.title('Distribution of {}'.format(feature))
    ax = sns.distplot(frame[feature], color=color)
```

In [65]:
```
numerical_columns = ['tenure', 'MonthlyCharges', 'TotalCharges']

for _ in numerical_columns:
    distplot(_, df)
#Note : distplot  will be removed in seaborn v0.14.0. It has been replaced by histplot
```







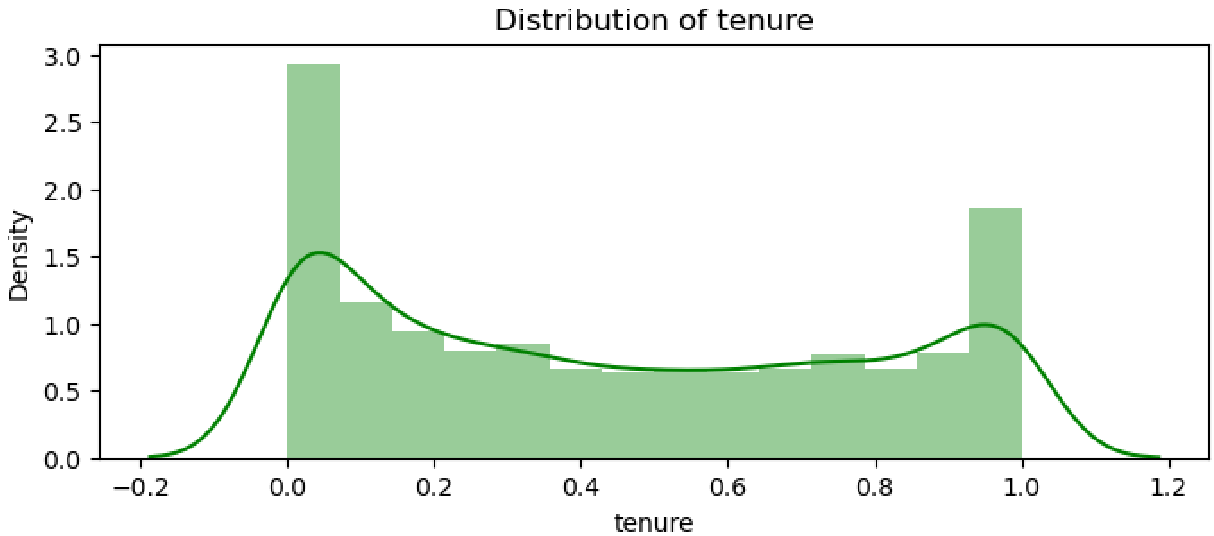
- Since the numerical features are distributed over different value ranges, feature scaling will be applied to scale them down to the same range.
- I will use Min-Max scaling because the distribution of numerical features is far from the normal (Gaussian) distribution.
  `[ (a - min(a)) / (max(a) - min(a)) ]`

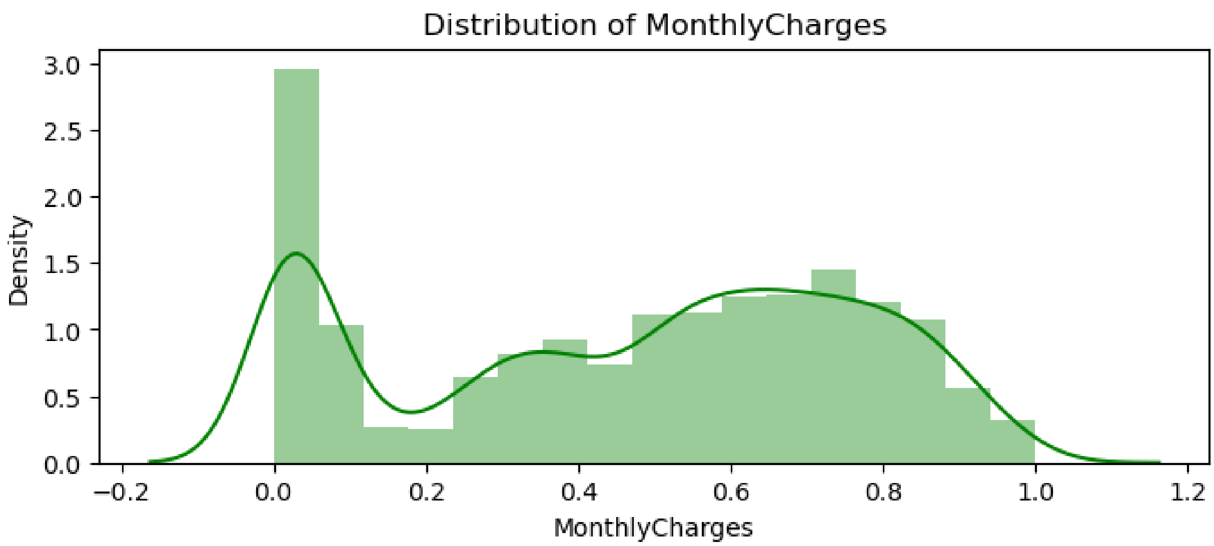
Scaling numeric attributes

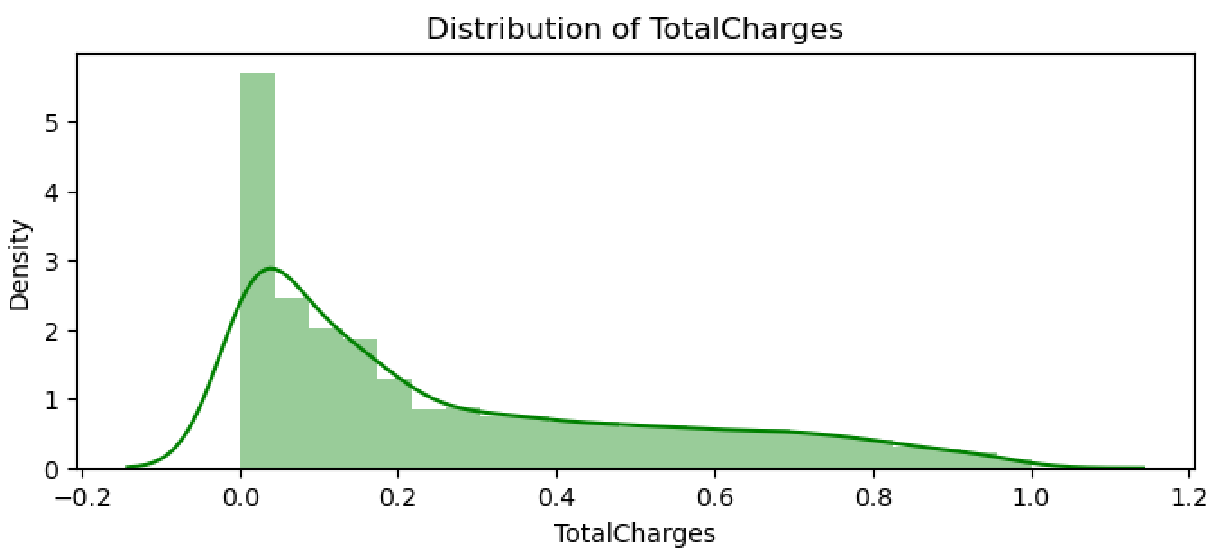
- Feature scaling should be applied to X data, there is no general benefit to applying feature scaling to y data.

In [66]:
```
mms = MinMaxScaler()
mms.fit(X_train[numerical_columns])

X_train[numerical_columns] = mms.transform(X_train[numerical_columns])
X_test[numerical_columns] = mms.transform(X_test[numerical_columns])
```

In [67]:
```
for _ in numerical_columns:
    distplot(_, X_train)
```







8. Machine Learning Model Evaluations and Predictions

KNN

In [68]:
```
knn_model = KNeighborsClassifier(n_neighbors=11)
knn_model.fit(X_train, y_train)
predicted_y = knn_model.predict(X_test)
```

In [69]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.84      0.86      0.85      1291
           1       0.58      0.54      0.56       467

    accuracy                           0.77      1758
   macro avg       0.71      0.70      0.70      1758
weighted avg       0.77      0.77      0.77      1758
```

SVC

In [70]:
```
svc_model = SVC(random_state=2)
svc_model.fit(X_train, y_train)
predicted_y = svc_model.predict(X_test)
```

In [71]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      1291
           1       0.66      0.52      0.58       467

    accuracy                           0.80      1758
   macro avg       0.75      0.71      0.73      1758
weighted avg       0.79      0.80      0.79      1758
```

Random Forest

In [72]:
```
rf_model = RandomForestClassifier(n_estimators=500, oob_score = True, random_state=5,
                                  max_leaf_nodes=30)
rf_model.fit(X_train, y_train)
predicted_y = rf_model.predict(X_test)
```

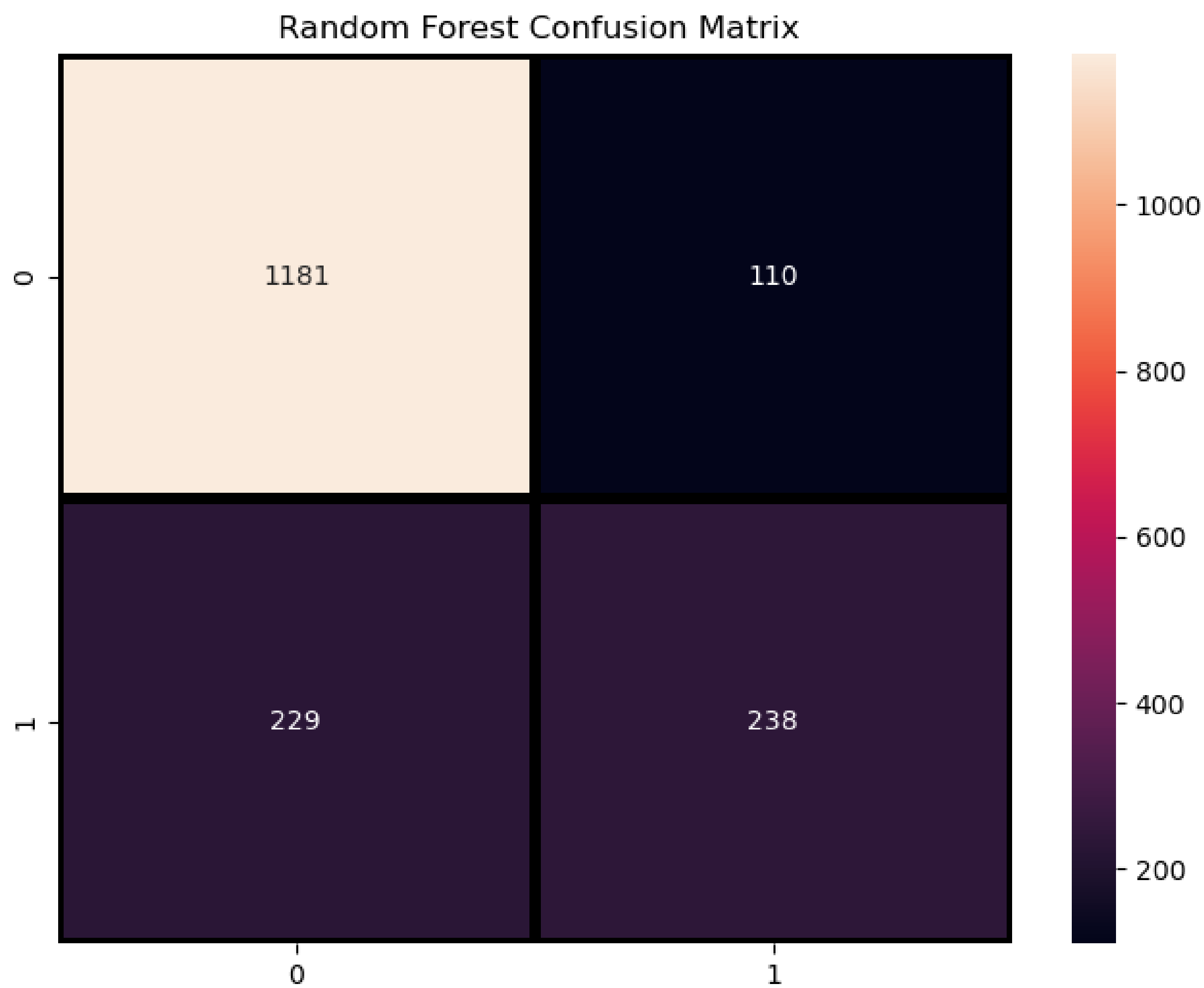
- **n_estimators**: The number of trees in the forest. This is a very important hyperparameter, and the optimal value will depend on the dataset. A good starting point is 500 trees.
- **oob_score**: Whether to use out-of-bag (OOB) samples to estimate the generalization error of the model. This is a useful hyperparameter for evaluating the model, but it can be computationally expensive.
- **random_state**: A seed for the random number generator. This ensures that the results of the model are reproducible.

- **max_features**: The number of features to consider when splitting a node. This can be a number, a fraction, or the string "auto". If "auto" is used, the number of features will be equal to the square root of the number of features.
- **max_leaf_nodes**: The maximum number of leaf nodes in each tree. This can help to prevent overfitting.

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

```
              precision    recall  f1-score   support

           0       0.84      0.91      0.87      1291
           1       0.68      0.51      0.58       467

    accuracy                           0.81      1758
   macro avg       0.76      0.71      0.73      1758
weighted avg       0.80      0.81      0.80      1758
```

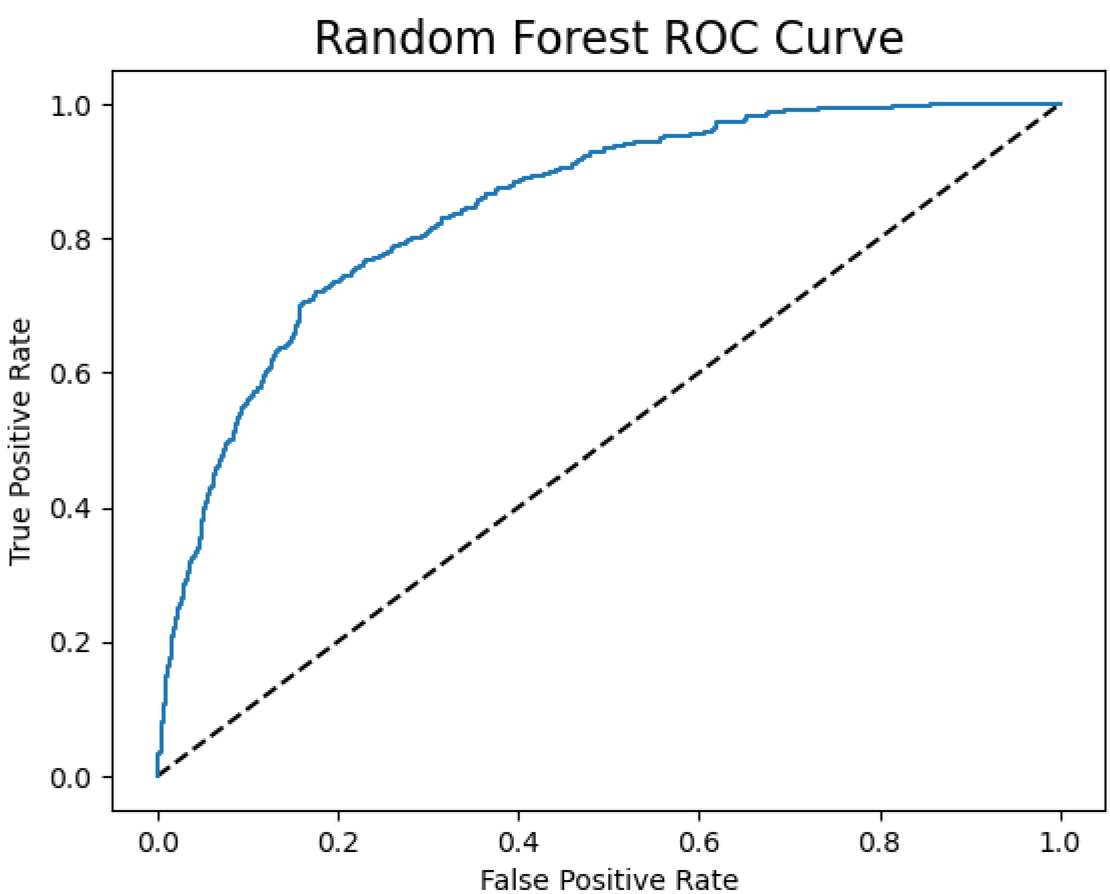
In [74]:
```
plt.figure(figsize=(8,6))
sns.heatmap(confusion_matrix(y_test, predicted_y), annot= True,
           fmt='d', linecolor='k', linewidths=3)

plt.title(' Random Forest Confusion Matrix')
plt.show()
```



Drawing an ROC (Receiver Operating Characteristic) curve for Random Forest Model

- Drawing an ROC curve for a Random Forest model involves evaluating the model's performance across different levels of classification threshold settings. The ROC curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold values.

In [75]:
```
y_rf_prob = rf_model.predict_proba(X_test)[:, 1]
# The [:, 1] indexing is used to select the predicted probabilities for the positive c
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_rf_prob)
plt.plot([0,1], [0,1], 'k--')
plt.plot(fpr_rf, tpr_rf, label='Random Forest')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Random Forest ROC Curve',fontsize=16)
plt.show();
```



- The ROC curve should be a plot that starts at the bottom-left corner (FPR = 0, TPR = 0) and goes towards the top-right corner (FPR = 1, TPR = 1). The closer the curve is to the top-left corner, the better the model's performance. The area under the ROC curve (AUC-ROC) is a common metric used to quantify the model's overall performance; **a higher AUC indicates better discrimination between the classes**.

In [76]:
```
from sklearn.metrics import roc_auc_score

auc_roc = roc_auc_score(y_test, y_rf_prob)
print("AUC-ROC:", auc_roc)
```

```
AUC-ROC: 0.8460101808434939
```

Logistic Regression

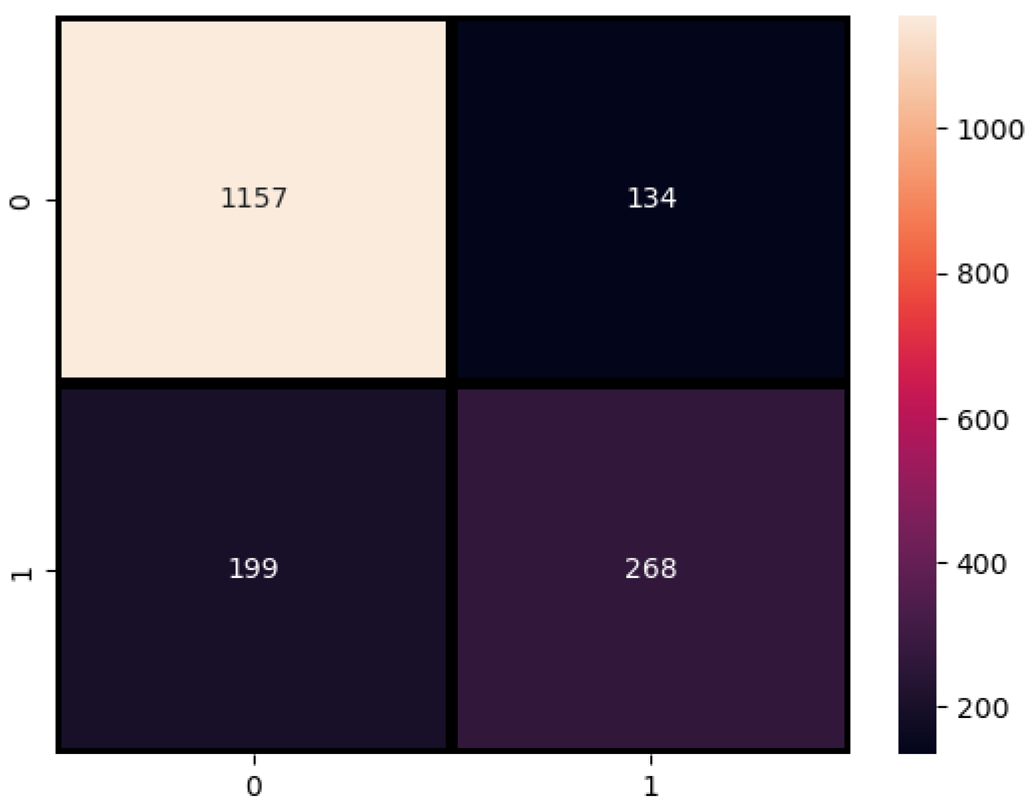
In [77]:
```
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

predicted_y = lr_model.predict(X_test)
```

In [78]:
```
print(classification_report(y_test, predicted_y))
```

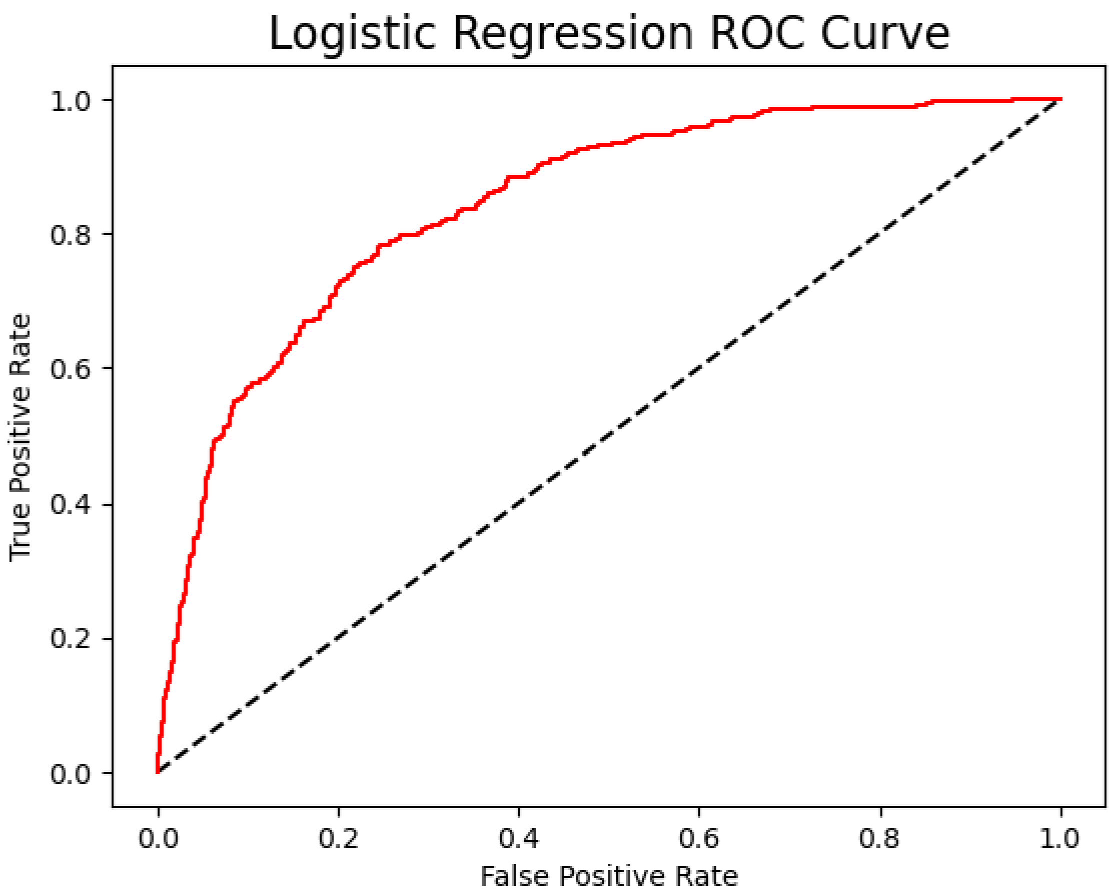
```
              precision    recall  f1-score   support

           0       0.85      0.90      0.87      1291
           1       0.67      0.57      0.62       467

    accuracy                           0.81      1758
   macro avg       0.76      0.74      0.75      1758
weighted avg       0.80      0.81      0.81      1758
```

In [79]:
```
sns.heatmap(confusion_matrix(y_test, predicted_y),
           annot = True, fmt='d', linecolor='k', linewidth=3)
```

Out[79]:
```
<Axes: >
```



In [80]:
```
y_lr_prob = lr_model.predict_proba(X_test)[:, 1]
fpr_lr, tpr_lr, thresholds = roc_curve(y_test, y_lr_prob)
plt.plot([0,1],[0,1], 'k--')
plt.plot(fpr_lr, tpr_lr, 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()
```



In [81]:
```
auc_roc = roc_auc_score(y_test, y_lr_prob)
print("AUC-ROC:", auc_roc)
```

```
AUC-ROC: 0.8428288745838841
```

Decisison Tree Classifier

In [82]:
```
dt_model = DecisionTreeClassifier()
dt_model.fit(X_train, y_train)

predicted_y = dt_model.predict(X_test)
```

In [83]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.83      0.81      0.82      1291
           1       0.50      0.53      0.52       467

    accuracy                           0.74      1758
   macro avg       0.66      0.67      0.67      1758
weighted avg       0.74      0.74      0.74      1758
```

- Decision tree gives very low score.

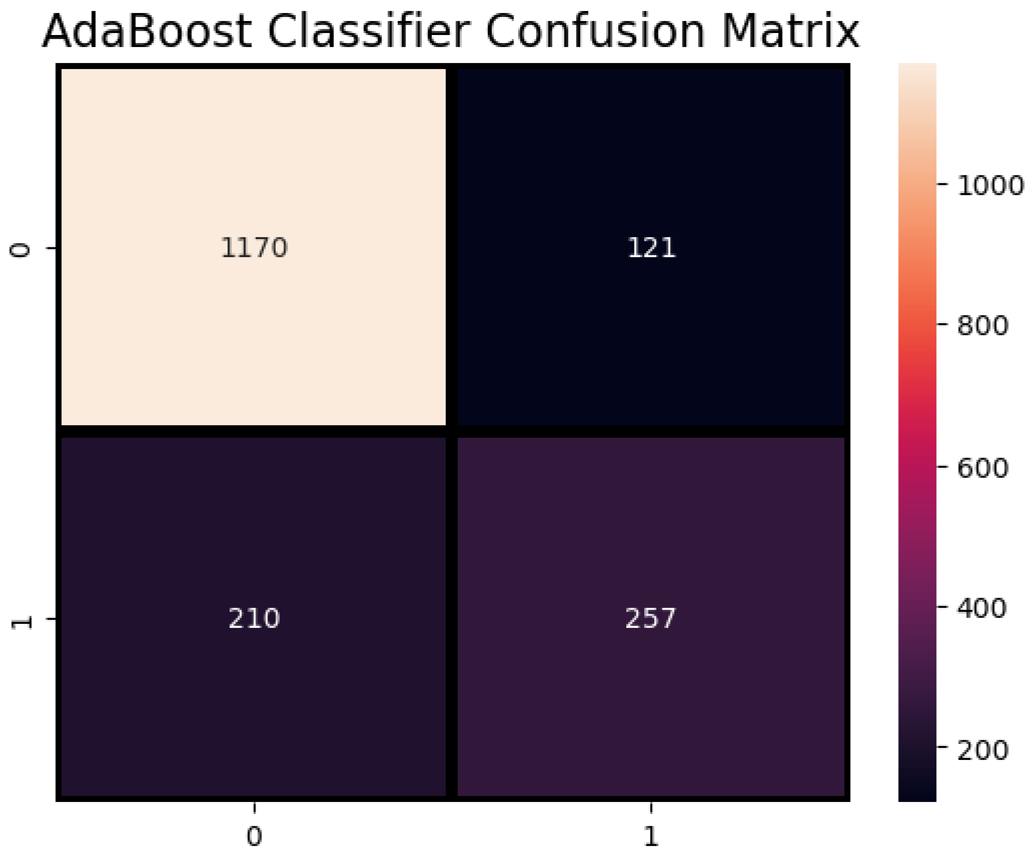
AdaBoost Classifier

In [84]:
```
ab_model = AdaBoostClassifier()
ab_model.fit(X_train, y_train)

predicted_y = ab_model.predict(X_test)
```

In [85]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1291
           1       0.68      0.55      0.61       467

    accuracy                           0.81      1758
   macro avg       0.76      0.73      0.74      1758
weighted avg       0.80      0.81      0.80      1758
```

In [86]:
```
sns.heatmap(confusion_matrix(y_test, predicted_y), annot=True, fmt='d', linecolor='k',
plt.title('AdaBoost Classifier Confusion Matrix', fontsize=16)
plt.show()
```



Gradient Boosting Classifier

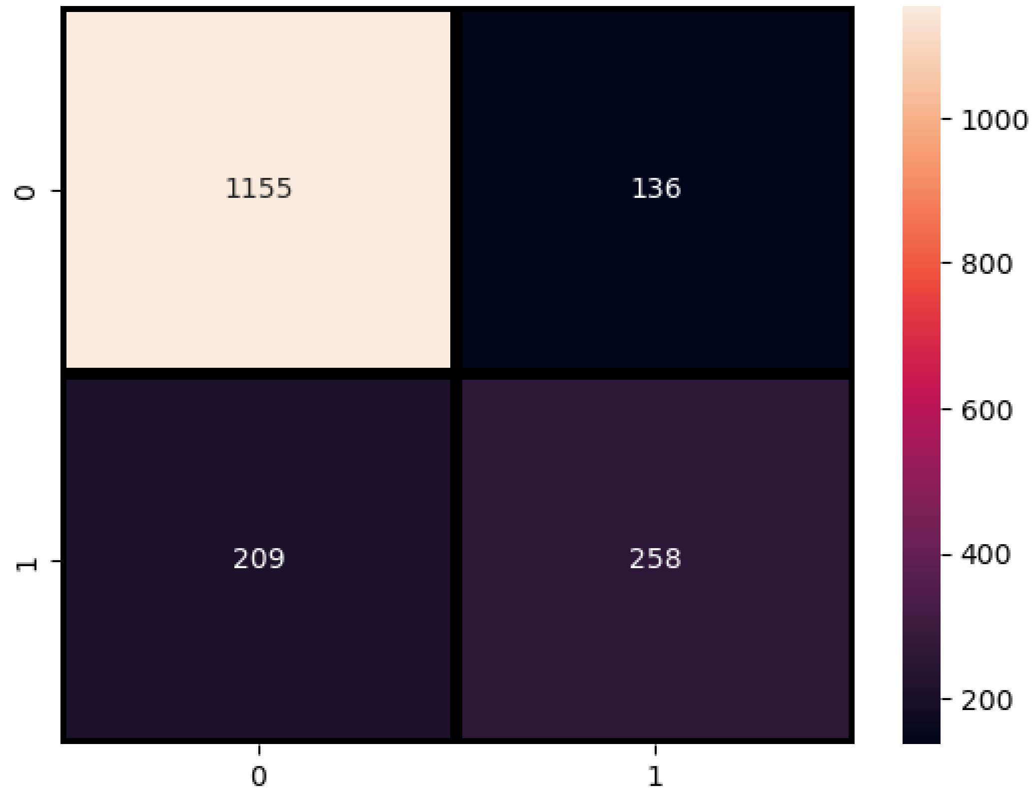
In [87]:
```
gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)

predicted_y = gb_model.predict(X_test)
```

In [88]:
```
print(classification_report(y_test, predicted_y))
```

```
              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1291
           1       0.65      0.55      0.60       467

    accuracy                           0.80      1758
   macro avg       0.75      0.72      0.73      1758
weighted avg       0.80      0.80      0.80      1758
```

In [89]:
```
sns.heatmap(confusion_matrix(y_test, predicted_y), annot=True, fmt='d', linecolor='k',
```

Out[89]:
```
<Axes: >
```



Model Selection

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

In [90]:
```
from sklearn.ensemble import VotingClassifier
clf1 = GradientBoostingClassifier()
clf2 = LogisticRegression()
clf3 = AdaBoostClassifier()
clf4 = RandomForestClassifier()
clf5 = SVC()

eclf = VotingClassifier(estimators=[('gbc', clf1), ('lr', clf2), ('ada', clf3), ('rf',

eclf.fit(X_train, y_train)
predictions = eclf.predict(X_test)
```

In [91]:
```
print(classification_report(y_test, predictions))
```

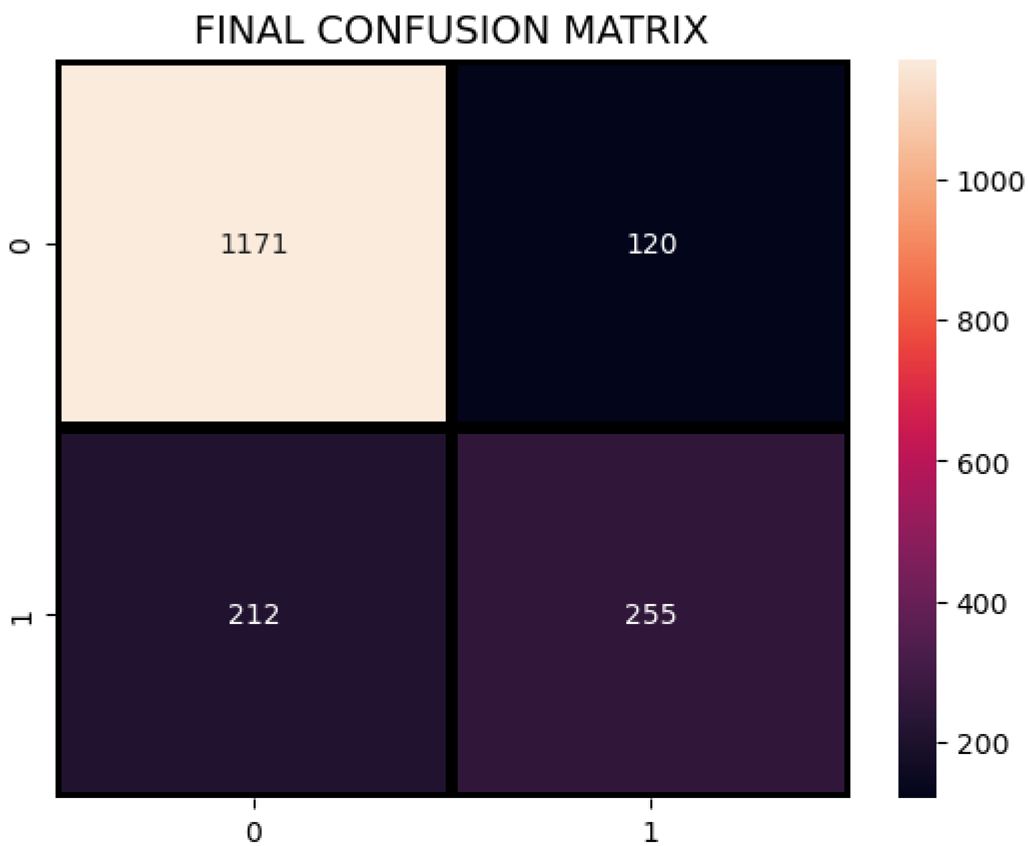
```
              precision    recall  f1-score   support

           0       0.85      0.91      0.88      1291
           1       0.68      0.55      0.61       467

    accuracy                           0.81      1758
   macro avg       0.76      0.73      0.74      1758
weighted avg       0.80      0.81      0.80      1758
```

In [92]:
```
print('Final Accuracy Score')
print(accuracy_score(y_test, predictions))
```

```
Final Accuracy Score
0.8111490329920364
```

This approach of creating an ensemble allows you to harness the strengths of multiple classifiers to potentially improve the overall performance of our model. The choice of individual classifiers and the voting strategy can impact the ensemble's performance.

In [93]:
```
sns.heatmap(confusion_matrix(y_test, predictions), annot=True, fmt='d', linecolor='k',
plt.title("FINAL CONFUSION MATRIX",fontsize=14)
plt.show()
```



From the confusion matrix we can see that: There are total 1171+120=1291 actual non-churn values and the algorithm predicts 1171 of them as non churn and 120 of them as churn. While there are 212+255=467 actual churn values and the algorithm predicts 212 of them as non churn values and 255 of them as churn values.

- Churn is when customers stop doing business with a company, measuring the rate of customer loss over time due to reasons like dissatisfaction or competition.

Ways to Prevent Churn:

1.**Great Customer Service**: Address issues promptly. 2.**Improve Products/Services**: Stay ahead of needs. 3.**Engage Customers**: Use various channels. 4.**Personalize Experiences**: Understand preferences. 5.**Loyalty Programs**: Incentivize repeat business. 6.**Proactive Issue Resolution**: Anticipate and solve problems. 7.**Collect Feedback**: Regularly gather customer insights. 8.**Communicate Value**: Showcase benefits offered. 9.**Flexible Contracts**: Align with customer preferences. 10.**Stay Competitive**: Monitor market trends. 11.**Smooth Onboarding**: Help new customers start strong. 12.**Renewal Reminders**: Notify ahead of renewals. 13.**Data Analytics**: Identify and address indicators of churn.