SYRIATEL CHURN RATE REPORT

1. BUSINESS UNDERSTANDING

BUSINESS OVERVIEW

The telecommunication sector is made up of companies that make communication possible on a global scale, whether through the phone, the internet, over airwaves, or cables. These companies create the infrastructure that allows data such as text, voice, audio, or video to be sent anywhere in the world.

The largest companies in the sector are telephone operators (both wired and wireless), satellite companies, cable companies, and internet service providers.

Here we will focus on SyriaTel as the telecommunication company under a telescope. **Syriatel** (سيريتل) is a mobile network provider in Syria· It is one of the only two providers in Syria, the other being MTN Syria. Syriatel was founded in January 2000, with its headquarters in Damascus, Syria. The company is owned by Rami Makhlouf, cousin of Syrian president Bashar al-Assad.The company had approximately 3,500 employees and 8 million customers as of 2016.It is headquartered on Sehnaya Road in

Damascus.Syriatel operate a network of GSM 900 / 1800 & 3G 2100 & 4G 1800 cellular networks.

Problem statement

SyriaTel wants to reduce the money they are losing when customers decide to exit and go for another Telecom company. We are going to identify the factors that contribute to customer churn in the telecom industry.

We will develop a machine learning model that can predict the likelihood of customer churn based on various customer characteristics and usage patterns.
Our goal is to use this model to identify customers who are at high risk of churning and take proactive measures to retain them. By reducing customer churn, the company can improve customer satisfaction, increase revenue, and gain a competitive advantage in the market.

Business Objectives

Based on the given dataset, the following objectives will be defined:

1. ***Main Objective***

Develop a machine learning model to predict the churn score based on usage pattern.

2. ***Specific Objectives***

1. Identify the factors that contribute to customer churn.
2. Evaluate the effectiveness of retention strategies.
3. Give appropriate recommendations to assist in minimizing the churn rate.

3. Research questions

1. What are the most common reasons for customer churn?
2. How likely are the customers likely to churn, based on their usage patterns and customer characteristics?
3. How can the company prevent its customers from churning?
4. What retention strategies have been most effective in reducing customer churn?

Performance Metrics:

The most appropriate performance metric for this project is **'recall'** or **'sensitivity'.** This is because reducing false negatives (i.e. customers who actually churn but are predicted as not churning) is a high priority for the telecom company. By correctly identifying customers who are likely to churn, the company can take proactive measures to retain them and reduce the overall churn rate.

However, it is also important to balance this with other performance metrics such as 'precision' and 'accuracy' to ensure that the model is not overly aggressive in predicting churn and causing false alarms or negatively impacting customer experience. Therefore, a combination of metrics such as precision, recall, F1-score, and AUC-ROC can be used to evaluate the performance of the model and ensure that it meets the business requirements and goals of the telecom company.

2. DATA UNDERSTANDING

Our data was obtained from Kaggle. This is the link: Dataset Link: link text

This phase had the following parts:

1) Collection of data and loading
2) Describing the data
3) Data exploration
4) Verifying quality of our data

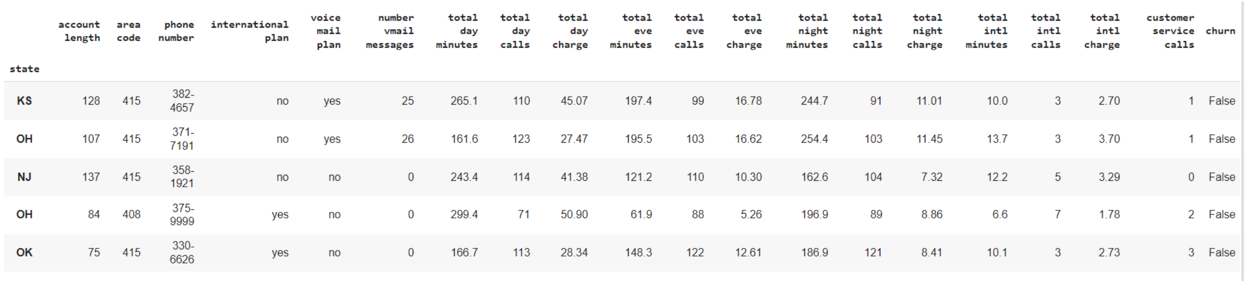
Loading and importing libraries

```
#importing necessary packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.gridspec as gs
import seaborn as sns

%matplotlib inline
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder


from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.tree import plot_tree

from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
import itertools

from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import precision_score, recall_score, accuracy_score,
f1_score, roc_curve, auc, confusion_matrix
```

```
loading the dataset
data = pd.read_csv('/content/SyriaTel_Customer_Churn.csv', index_col = 0)
# A copy of our original dataset to work with
df = data.copy()
```

```
#Display the first five rows of the dataset
df.head()
```



state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls	churn
KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False
OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False
NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False
OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False
OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False

Describing our data

```
#A summary of all columns

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>

Index: 3333 entries, KS to TN

Data columns (total 20 columns):

 #   Column                  Non-Null Count  Dtype

---  ------                  --------------  -----

 0   account length          3333 non-null   int64

 1   area code               3333 non-null   int64

 2   phone number            3333 non-null   object

 3   international plan      3333 non-null   object

 4   voice mail plan         3333 non-null   object

 5   number vmail messages   3333 non-null   int64

 6   total day minutes       3333 non-null   float64

 7   total day calls         3333 non-null   int64

 8   total day charge        3333 non-null   float64

 9   total eve minutes       3333 non-null   float64

 10  total eve calls         3333 non-null   int64

 11  total eve charge        3333 non-null   float64

 12  total night minutes     3333 non-null   float64

 13  total night calls       3333 non-null   int64

 14  total night charge      3333 non-null   float64

 15  total intl minutes      3333 non-null   float64

 16  total intl calls        3333 non-null   int64
 17   total intl charge       3333 non-null   float64
 18   customer service calls  3333 non-null   int64
 19   churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(3)
memory usage: 524.0+ KB
```

Non- categorical features()

- account length
- area code
- number vmail messages
- total day minutes
- total day calls
- total day charge
- total eve minutes
- total eve calls
- total eve charge
- total night minutes
- total night calls
- total night charge
- total intl minutes
- total intl calls
- total intl charge
- customer service calls

Categorical features()

- international plan
- voicemail plan

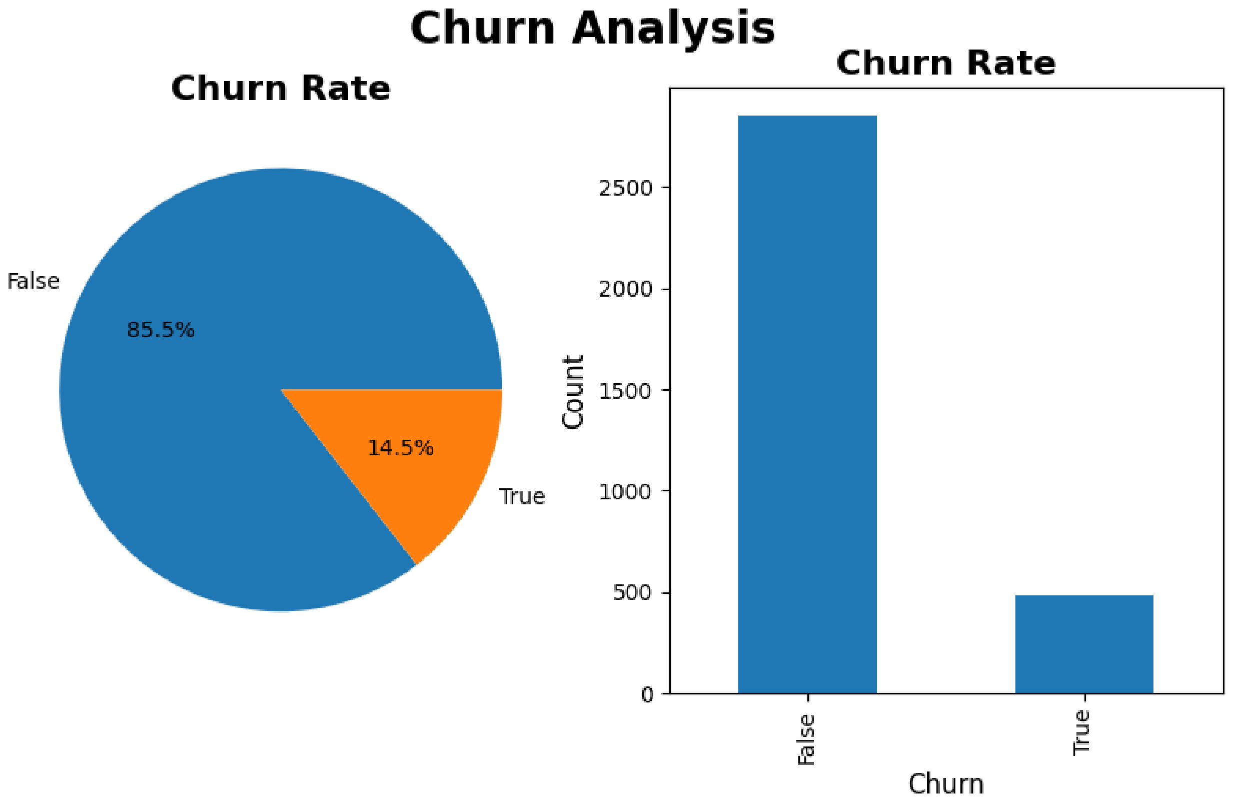
We have a total of 20 columns and 3333 entries.

2.1 Univariate Analysis

In this section we shall be exploring a number of factors that impact the churn rate of customers at SyriaTel.

1) We will first work on the Churn rate and plot using a pie chart and a bar graph.

```
#Plotting a pie chart to display the false and true parameters
#Plotting a bar graph to display the counts of each
fig, ax = plt.subplots(1, 2, figsize=(10,5))


df['churn'].value_counts().plot(kind='pie', autopct='%1.1f%%', ax=ax[0])
ax[0].set_title('Churn Rate', fontsize=16, fontweight='bold')
ax[0].set_ylabel('') # remove the y-axis label


df['churn'].value_counts().plot(kind='bar', ax=ax[1])
ax[1].set_title('Churn Rate', fontsize=16, fontweight='bold')
ax[1].set_xlabel('Churn', fontsize=12)
ax[1].set_ylabel('Count', fontsize=12)


plt.suptitle('Churn Analysis', fontsize=20, fontweight='bold')
plt.show()
```

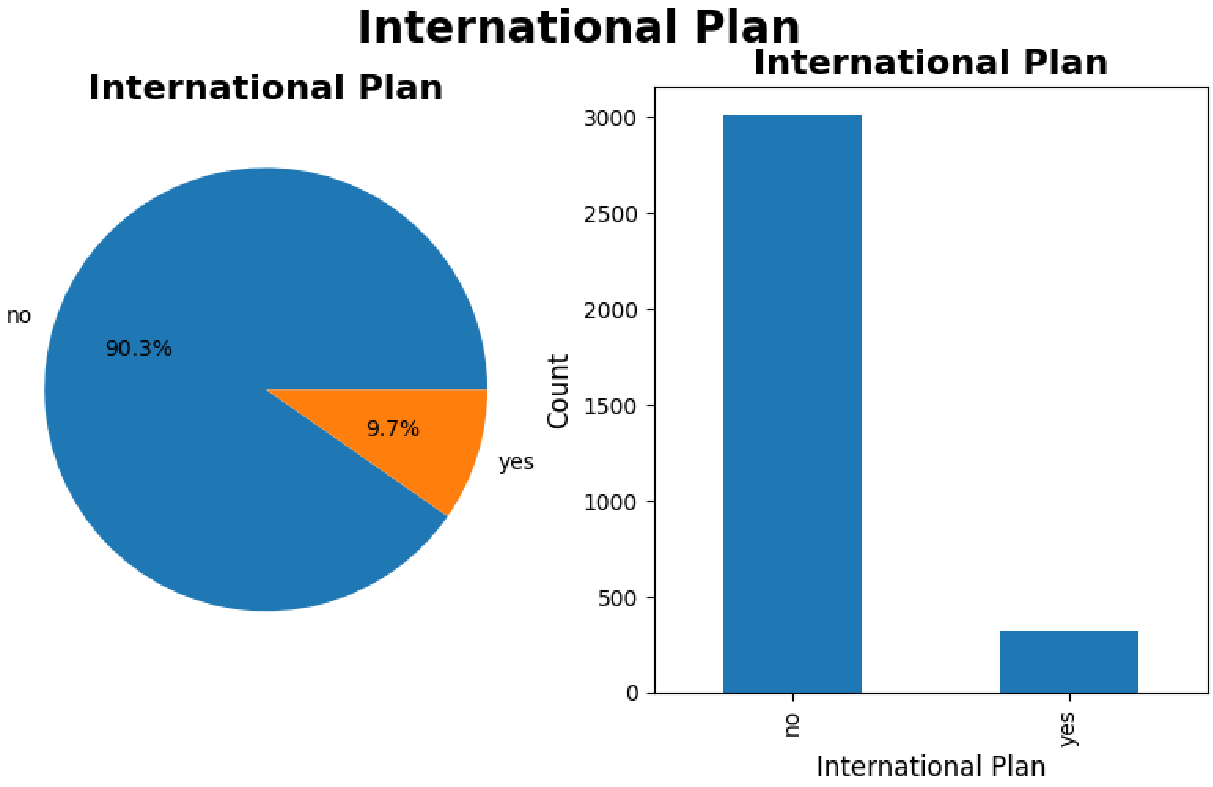


From the visualizations displayed above, we can see that the churn rate of The SyriaTel customers is at 14.5% .

2) Here we shall see the percentage and number of customers who are on the International plan.

```
# Displaying the customers who are on the International Plan

fig, ax = plt.subplots(1, 2, figsize=(10,5))



df['international plan'].value_counts().plot(kind='pie',
autopct='%1.1f%%', ax=ax[0])

ax[0].set_title('International Plan', fontsize=16, fontweight='bold')

ax[0].set_ylabel('') # remove the y-axis label
```

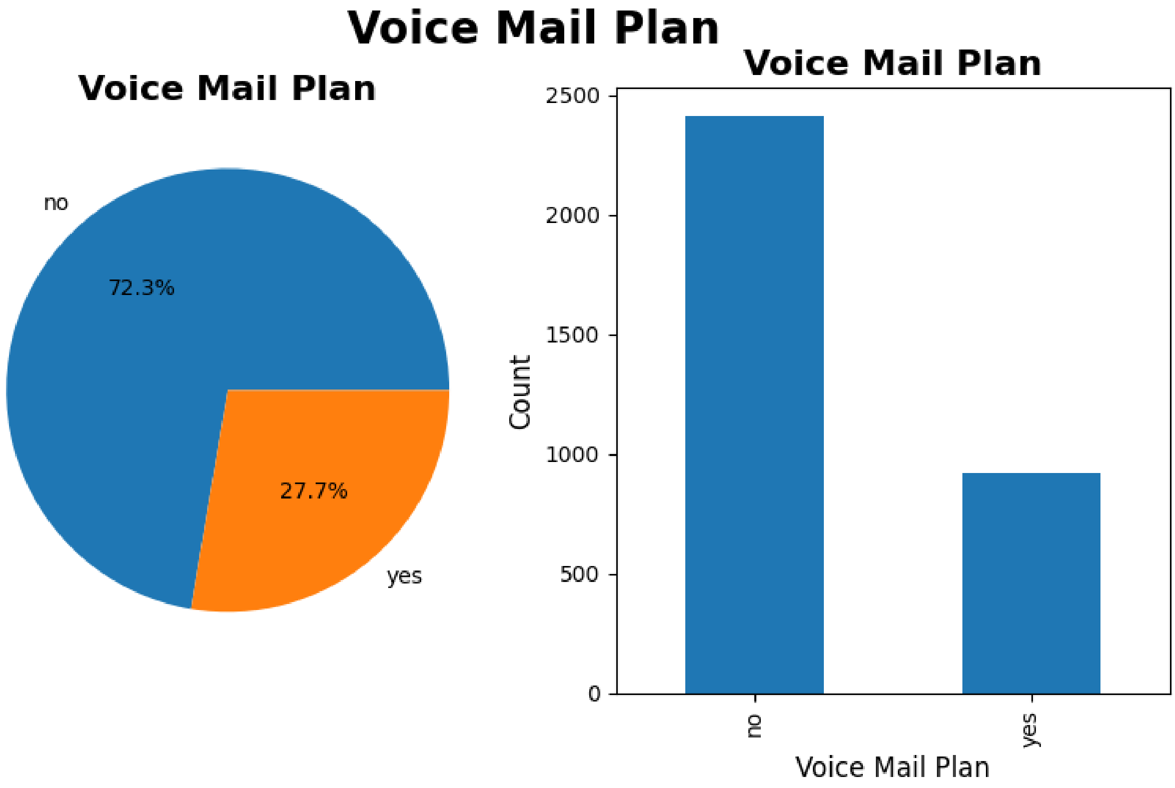
```
df['international plan'].value_counts().plot(kind='bar', ax=ax[1])

ax[1].set_title('International Plan', fontsize=16, fontweight='bold')

ax[1].set_xlabel('International Plan', fontsize=12)

ax[1].set_ylabel('Count', fontsize=12)



plt.suptitle('International Plan', fontsize=20, fontweight='bold')

plt.show()
```



From the visual above we can reach a suitable conclusion that most customers on the SyriaTel telecom are not on the international plan with a high percentage of 90.5%.

3) What is the percentage/number of customers on SyriaTel's voicemail plan.

```
# Displaying the churn rate in accordance to the voice mail plan

fig, ax = plt.subplots(1, 2, figsize=(10,5))



df['voice mail plan'].value_counts().plot(kind='pie', autopct='%1.1f%%', ax=ax[0])

ax[0].set_title('Voice Mail Plan', fontsize=16, fontweight='bold')

ax[0].set_ylabel('') # remove the y-axis label



df['voice mail plan'].value_counts().plot(kind='bar', ax=ax[1])

ax[1].set_title('Voice Mail Plan', fontsize=16, fontweight='bold')

ax[1].set_xlabel('Voice Mail Plan', fontsize=12)

ax[1].set_ylabel('Count', fontsize=12)



plt.suptitle('Voice Mail Plan ', fontsize=20, fontweight='bold')

plt.show()
```



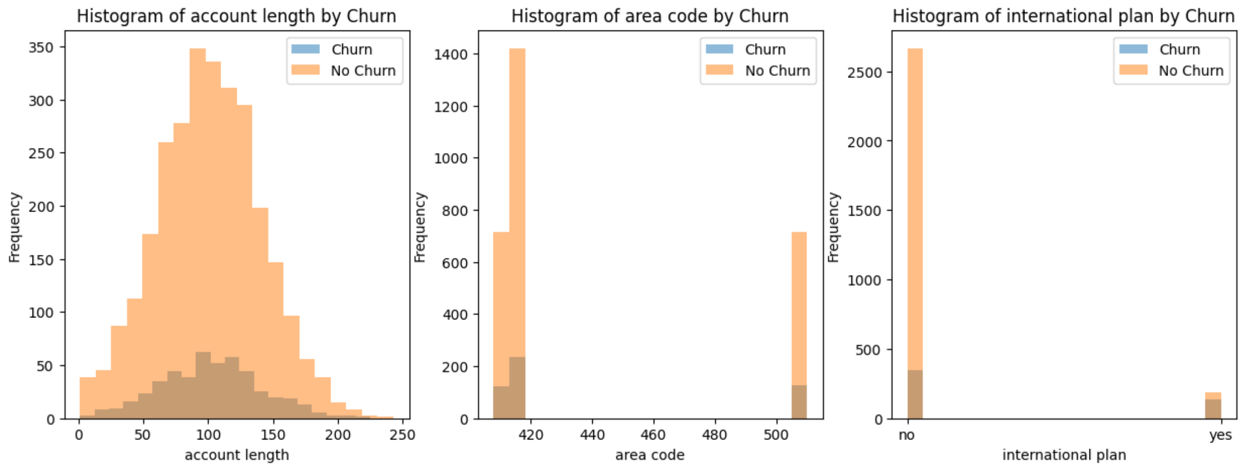
It's evident that 27.8 % of our customers were using the 'voice mail plan'.

2.2 BIVARIATE ANALYSIS

1) We shall plot the churn rate (frequency) of SyriaTel's customers against three variables, that is 'account length', 'area code' and 'international plan'.

```
# Plotting churn rate against 'account length', 'area code',
'international plan'

cols = ['account length', 'area code', 'international plan']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))

for i, col in enumerate(cols):

    axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5,
label='Churn')
```

```
    axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No Churn')

    axs[i].set_xlabel(col)

    axs[i].set_ylabel('Frequency')

    axs[i].set_title(f'Histogram of {col} by Churn')

    axs[i].legend(loc='upper right')

plt.show()
```

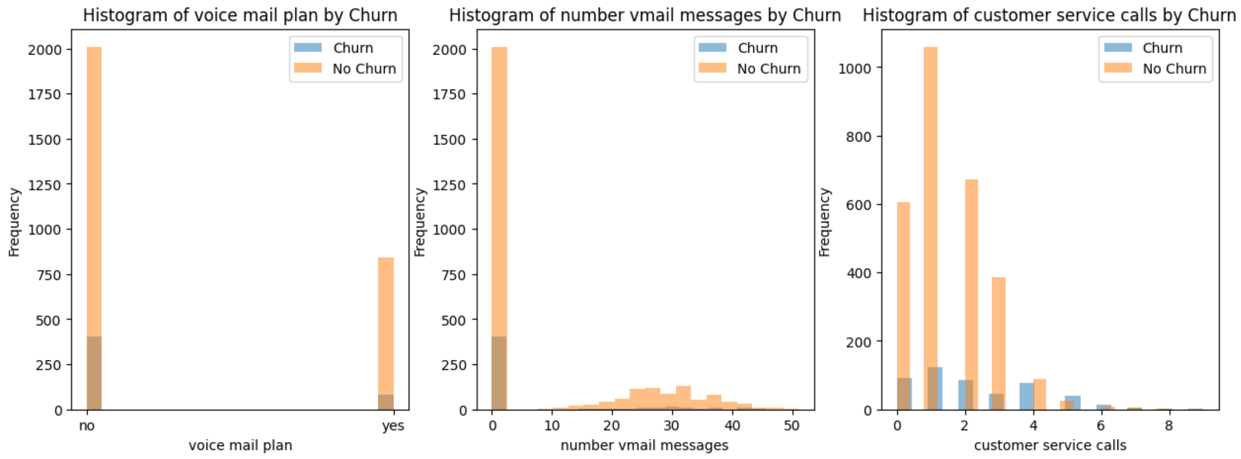


From the above visuals we were able to conclude that:

1. Most customers left the company with an account length of approximately 90 to 120 days of being members.
2. Most customers are in the 415 area code. The highest churn rate appeared to be on the 415 area code.
3. The customers who do not have an international plan are more likely to churn as compared to those that have an international plan.

2) Plotting the churn rate against ‘voice mail plan’, ‘number of vmail messages’ and ‘customer service calls’ from the dataset.

```
#Plotting the churn rate against 'voice mail plan','number of vmail messages' and 'customer services calls' from the dataset

cols = ['voice mail plan', 'number vmail messages', 'customer service calls']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))

for i, col in enumerate(cols):

    axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5, label='Churn')

    axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No Churn')

    axs[i].set_xlabel(col)

    axs[i].set_ylabel('Frequency')

    axs[i].set_title(f'Histogram of {col} by Churn')

    axs[i].legend(loc='upper right')

plt.show()
```

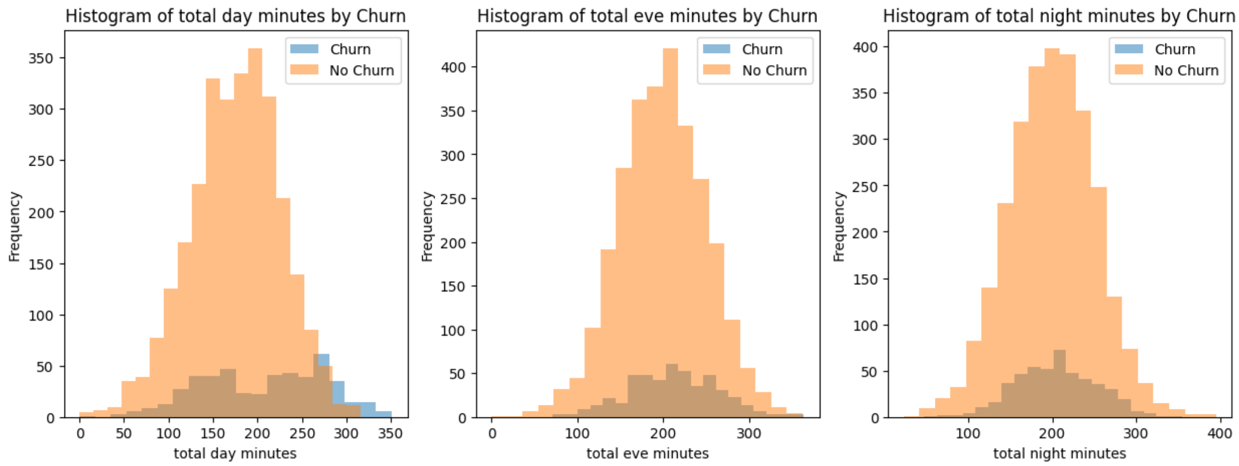


The above analysis explains that:

1. The customers who had no voicemail plan ended up having a higher churn rate as compared to those who had subscribed to having a voicemail plan.
2. The chances of the customers churning are higher when you have less vmail messages as compared to having more vmail messages.
3. The less number of customer service calls made to the telecom company the higher the churn rate of customers.

3) Plotting the customer's churn rate against the following variables: 'total day minutes', 'total evening minutes' and 'total night minutes'.

```
#Plotting churn rate against 'total day minutes', 'total evening minutes' and 'total night minutes'

cols = ['total day minutes', 'total eve minutes', 'total night minutes']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))

for i, col in enumerate(cols):

   axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5, label='Churn')

   axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No Churn')

   axs[i].set_xlabel(col)

   axs[i].set_ylabel('Frequency')

   axs[i].set_title(f'Histogram of {col} by Churn')

   axs[i].legend(loc='upper right')

plt.show()
```



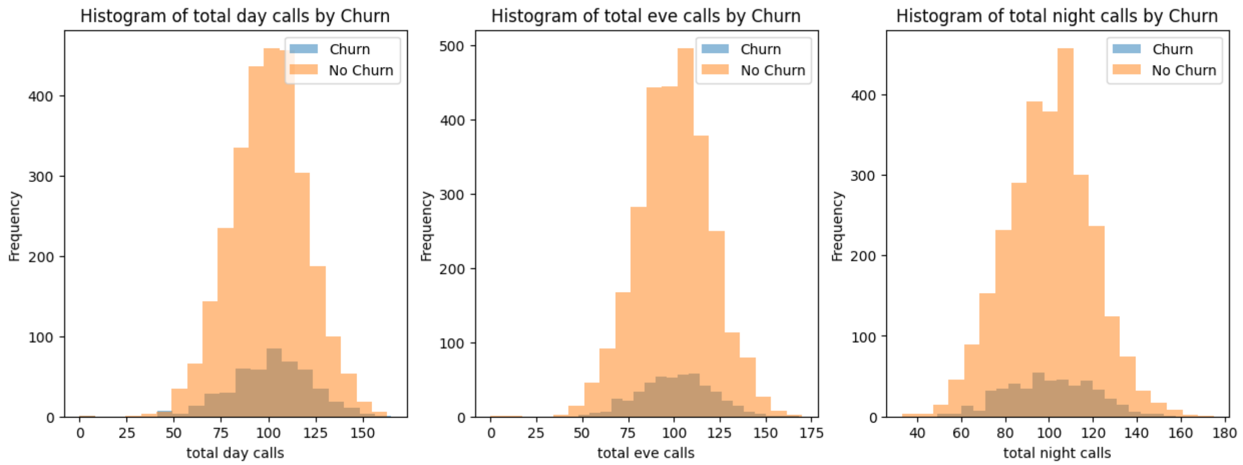
According to the visualizations portrayed above. We were able to deduce that:

1. The churn rate across the board was very minimal against the said variables as the minutes increased during the day towards the evening and in the nighttime.

4) Plotting customer churn rate against ‘total day calls’,’total evening calls’ and ‘total nights calls’.

```
#Plotting the churn rate against 'total day calls', 'total evening calls' and 'total night calls'

cols = ['total day calls', 'total eve calls', 'total night calls']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))

for i, col in enumerate(cols):

   axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5, label='Churn')

   axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No Churn')

   axs[i].set_xlabel(col)

   axs[i].set_ylabel('Frequency')
```

```
    axs[i].set_title(f'Histogram of {col} by Churn')

    axs[i].legend(loc='upper right')

plt.show()
```



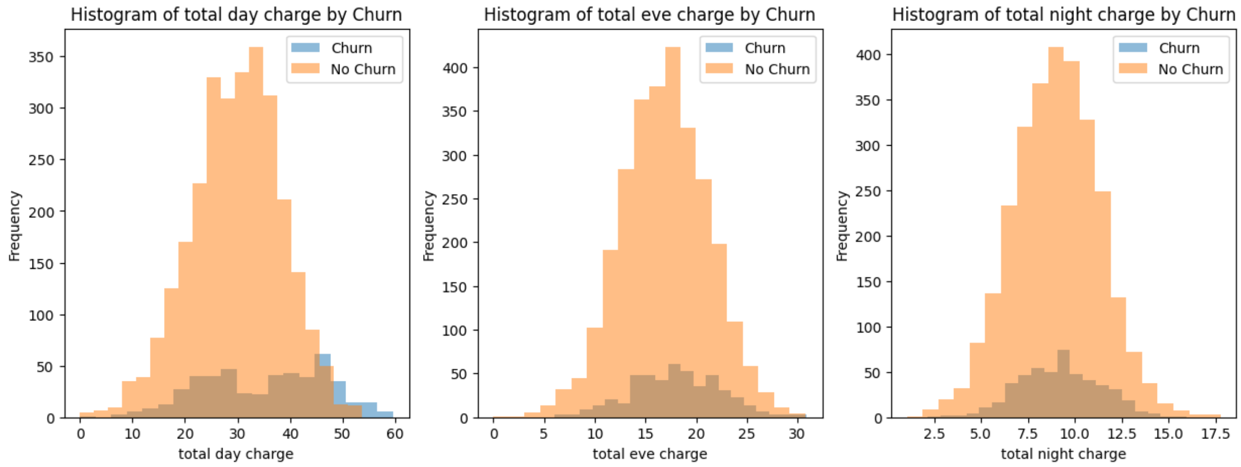
According to the visualizations portrayed above. We were able to deduce that:

1. The churn rate across the board was very minimal against the said variables as the minutes increased during the day towards the evening and in the nighttime.

5) Plotting the churn rate against the following variables: ‘total day charge’, ‘total evening charge’ and ‘total night charge’.

```
# Plotting the churn rate against 'total day charge','total evening
charge' and 'total night charge'

cols = ['total day charge', 'total eve charge', 'total night charge']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))

for i, col in enumerate(cols):

    axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5,
label='Churn')
```

```
    axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No Churn')

    axs[i].set_xlabel(col)

    axs[i].set_ylabel('Frequency')

    axs[i].set_title(f'Histogram of {col} by Churn')

    axs[i].legend(loc='upper right')

plt.show()
```



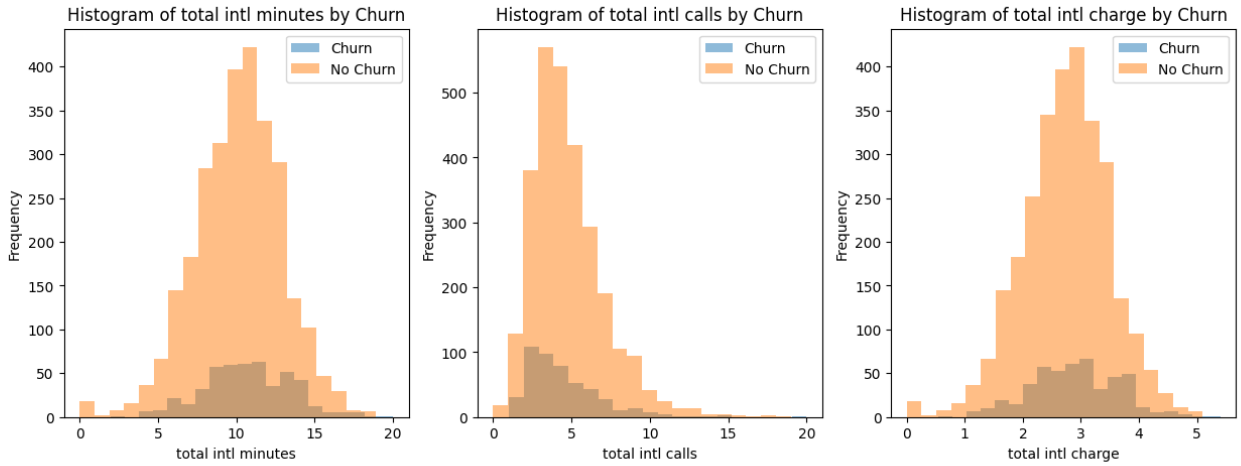
The above analysis explains:

How the charges increase throughout different times of the day from daytime, evening hours to nighttime.

6) Plotting ‘total intl minutes’, ‘total intl calls’ and ‘total intl charges’ against the churn rate of customers.

```
# Plotting the churn rate against 'total intl minutes','total intl calls' and 'total intl charge'

cols = ['total intl minutes', 'total intl calls', 'total intl charge']

fig, axs = plt.subplots(1, len(cols), figsize=(15, 5))
```

```
for i, col in enumerate(cols):
    axs[i].hist(df[df['churn'] == 1][col], bins=20, alpha=0.5,
label='Churn')
    axs[i].hist(df[df['churn'] == 0][col], bins=20, alpha=0.5, label='No
Churn')
    axs[i].set_xlabel(col)
    axs[i].set_ylabel('Frequency')
    axs[i].set_title(f'Histogram of {col} by Churn')
    axs[i].legend(loc='upper right')
plt.show()
```



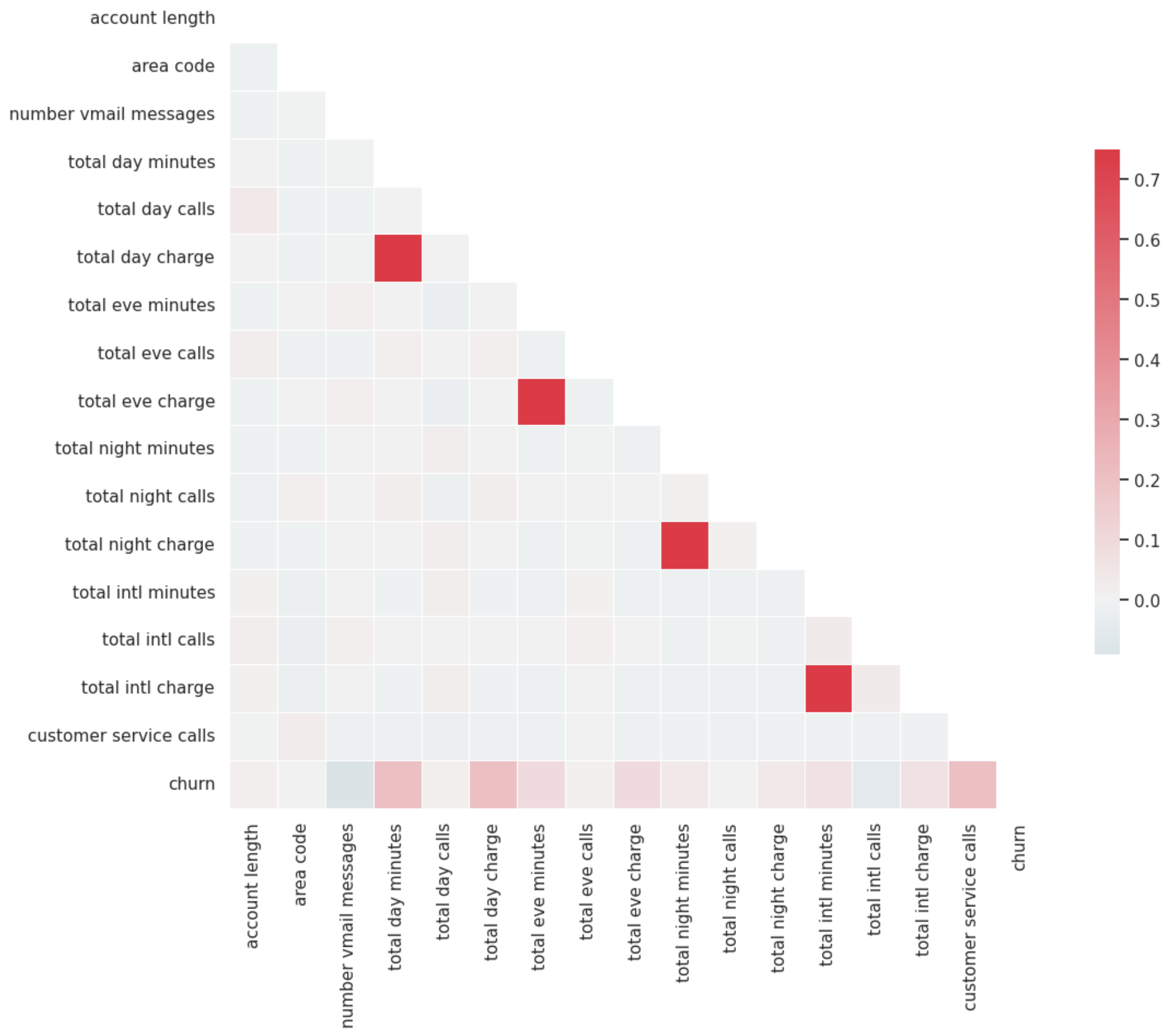
2.3 MULTIVARIATE ANALYSIS

With this analysis we shall be plotting all variables in a heat map.

```
#Creating a heatmap that show the correlation between the different
variables in the dataset.
```

```python
sns.set(style = 'white')

corr = df.corr()

mask = np.zeros_like(corr, dtype = bool)
mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize= (12,12))

cmap = sns.diverging_palette(220, 10, as_cmap=True)

sns.heatmap(corr, mask = mask, cmap = cmap, vmax = .75, center = 0, square = True, linewidth = .5, cbar_kws = {'shrink':.5})
```

```
<ipython-input-18-2a3843f0ea05>:4: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.

  corr = df.corr()

<Axes: >
```

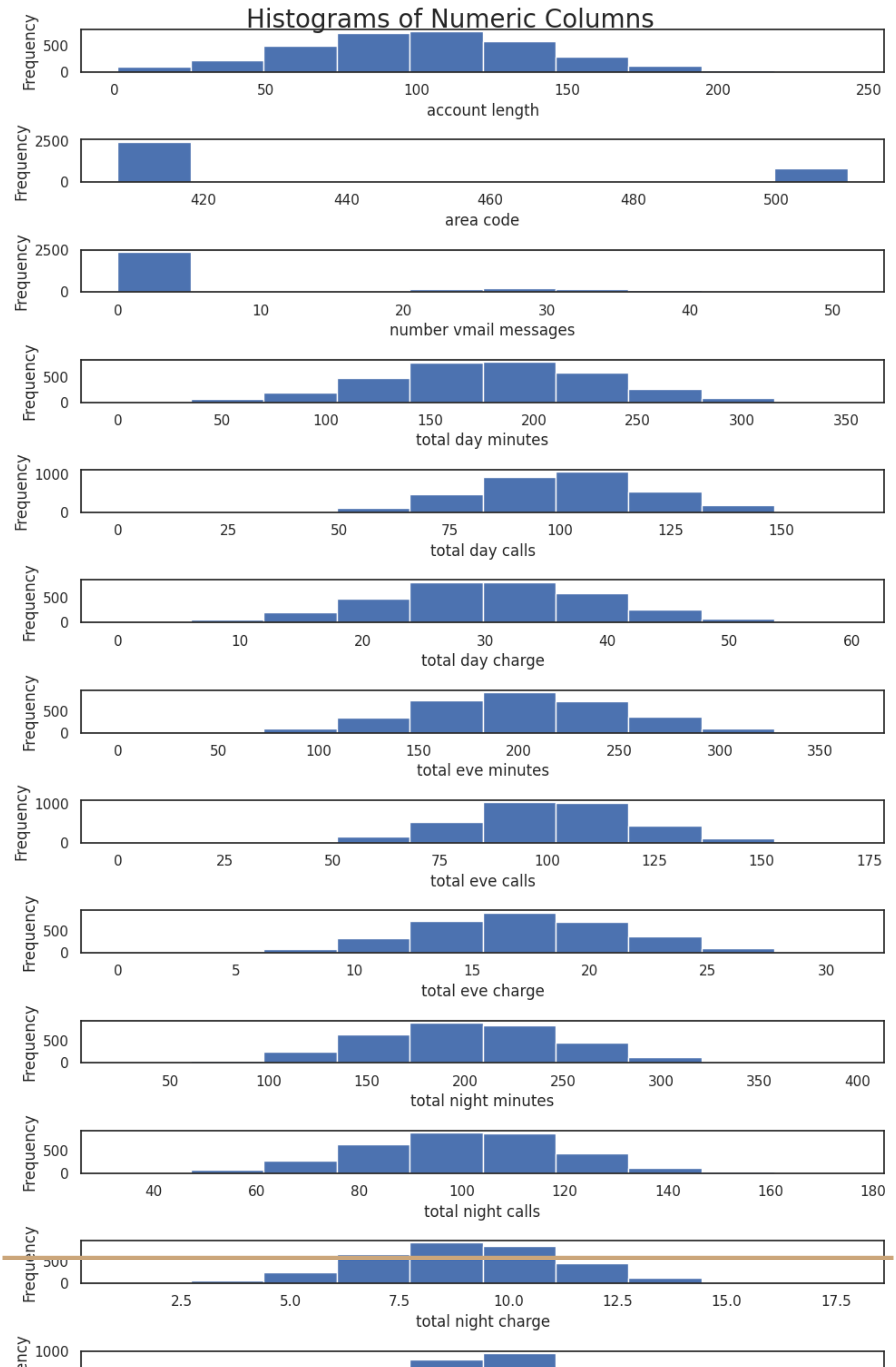


From the heatmap shown, we are able to identify some of factors that highly impacted customer churn at SyriaTel:

- Total day charges
- Total evening charges
- Total night charges
- Total international charge

DISTRIBUTION ANALYSIS

```
#Select only numeric columns
numeric_cols = df.select_dtypes(include=['float64', 'int64'])

# Plotting histograms to get an overview distribution of the columns
fig, axes = plt.subplots(nrows=numeric_cols.shape[1], ncols=1,
figsize=(10, 20))
fig.suptitle('Histograms of Numeric Columns', fontsize=20)

for i, column in enumerate(numeric_cols.columns):
   axes[i].hist(numeric_cols[column])
   axes[i].set_xlabel(column, fontsize=12)
   axes[i].set_ylabel('Frequency', fontsize=12)

plt.tight_layout()

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

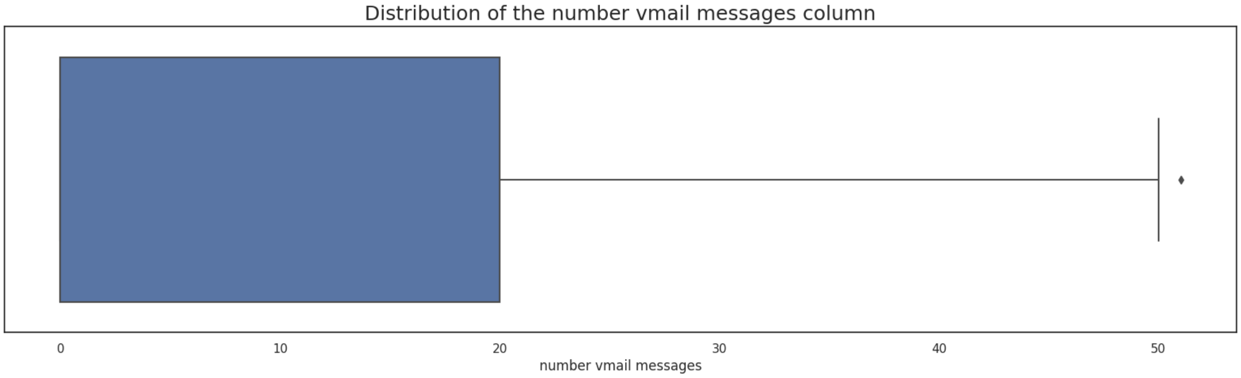

Histograms of Numeric Columns
Frequency
500
0
0
50
100
150
200
250
account length
Frequency
2500
0
420
440
460
480
500
area code
Frequency
2500
0
0
10
20
30
40
50
number vmail messages
Frequency
500
0
0
50
100
150
200
250
300
350
total day minutes
Frequency
1000
0
0
25
50
75
100
125
150
total day calls
Frequency
500
0
0
10
20
30
40
50
60
total day charge
Frequency
500
0
0
50
100
150
200
250
300
350
total eve minutes
Frequency
1000
0
0
25
50
75
100
125
150
175
total eve calls
Frequency
500
0
0
5
10
15
20
25
30
total eve charge
Frequency
500
0
50
100
150
200
250
300
350
400
total night minutes
Frequency
500
0
40
60
80
100
120
140
160
180
total night calls
Frequency
500
0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
total night charge
1000

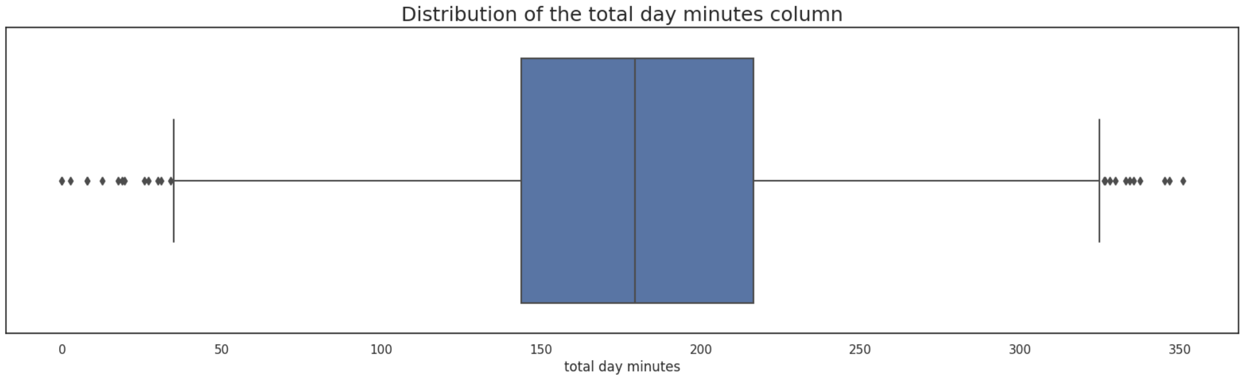
From the graph grid above most columns appear not to have a normal distribution.This will later be corrected through normalization.

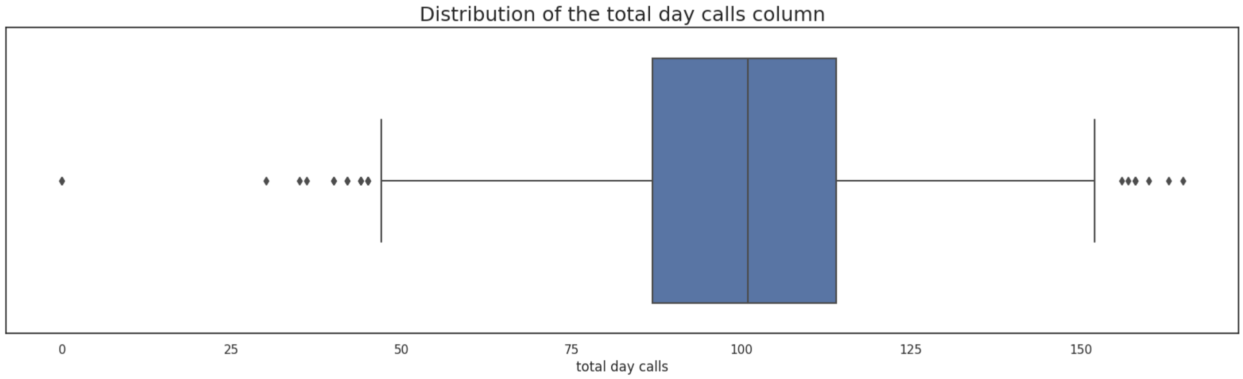
OUTLIERS

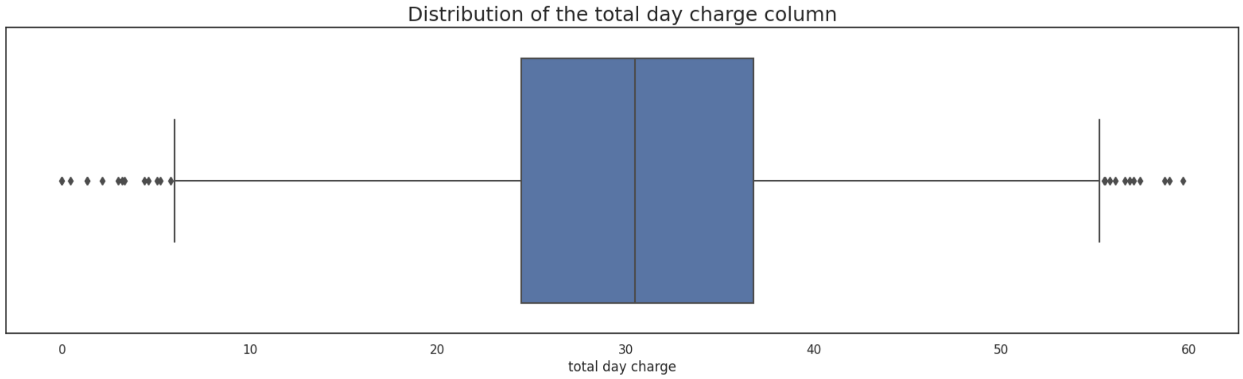
```
cols = ['account length','area code','number vmail messages','total day
minutes','total day calls','total day charge',
'total eve minutes','total eve calls','total eve charge','total night
minutes','total night calls',
'total night charge','total intl minutes','total intl calls','total intl
charge','customer service calls']
```

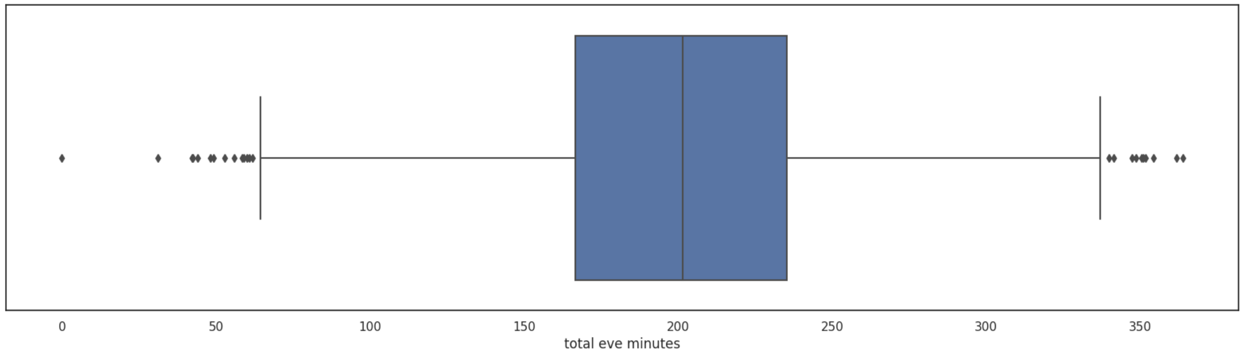
```
# We will use subplots to plot individual boxplots
df1 = df[['number vmail messages','total day minutes','total day
calls','total day charge',
'total eve minutes','total eve calls','total eve charge','total night
minutes','total night calls','total night charge',
'total intl minutes','total intl calls','total intl charge','customer
service calls']]

# Define subplot grid
for column in df1:
        plt.figure(figsize=(20,5))
        plt.title('Distribution of the ' + column+ ' column',fontsize=(18))
        sns.boxplot(data=df1, x=column)
```


Distribution of the number vmail messages column

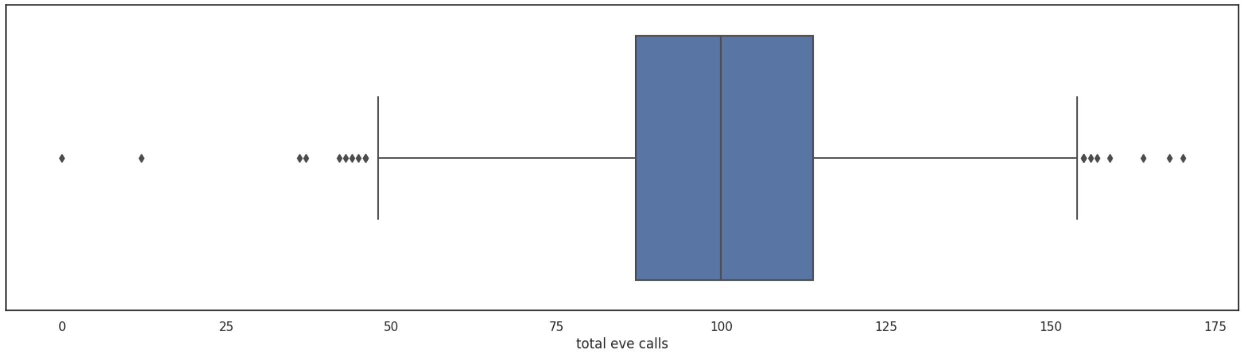

Distribution of the total day minutes column
0
50
100
150
200
250
300
350
total day minutes


Distribution of the total day calls column
0
25
50
75
100
125
150
total day calls

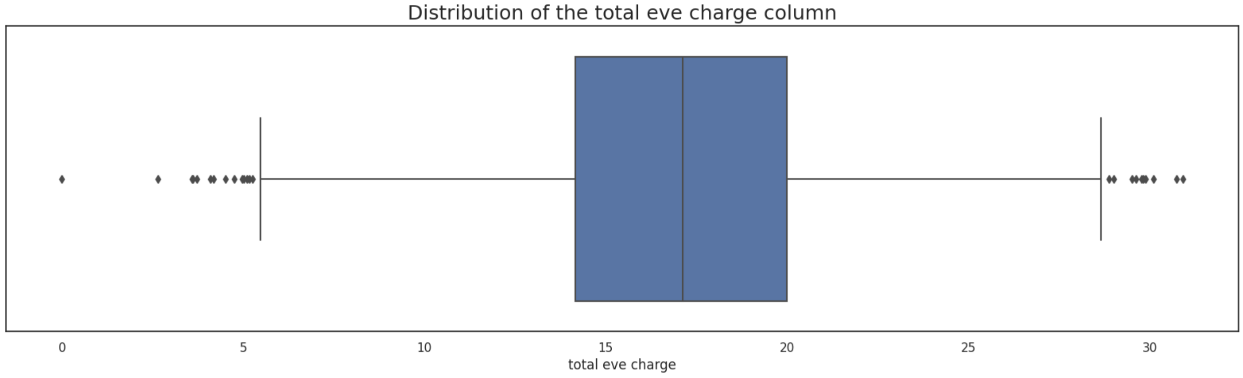

Distribution of the total day charge column
0
10
20
30
40
50
60
total day charge



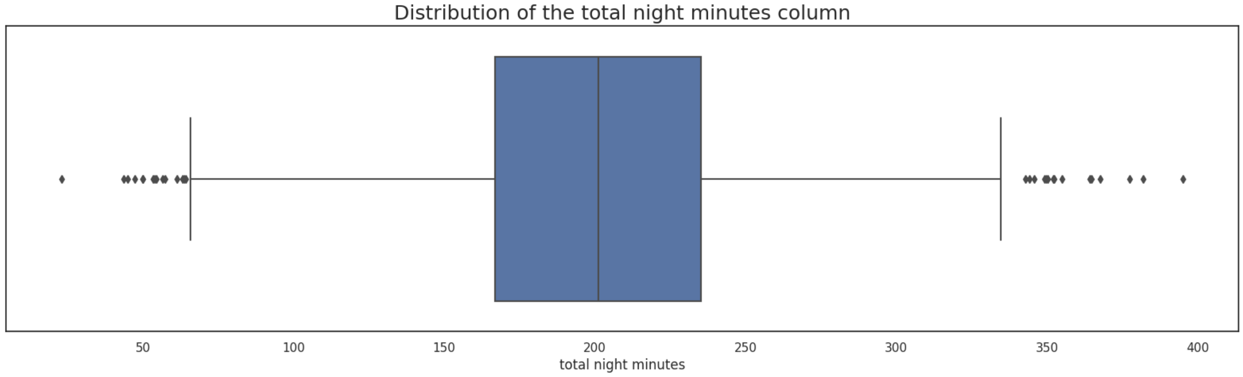
Distribution of the total eve minutes column



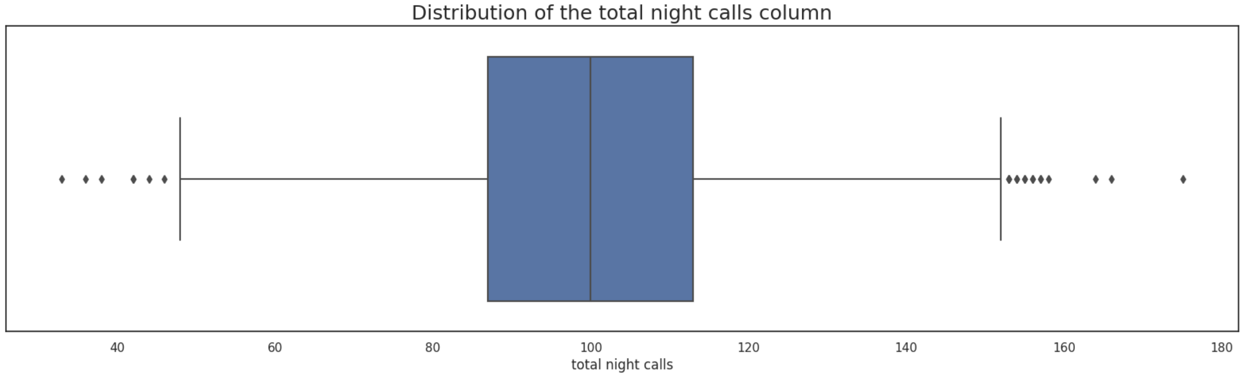
Distribution of the total eve calls column



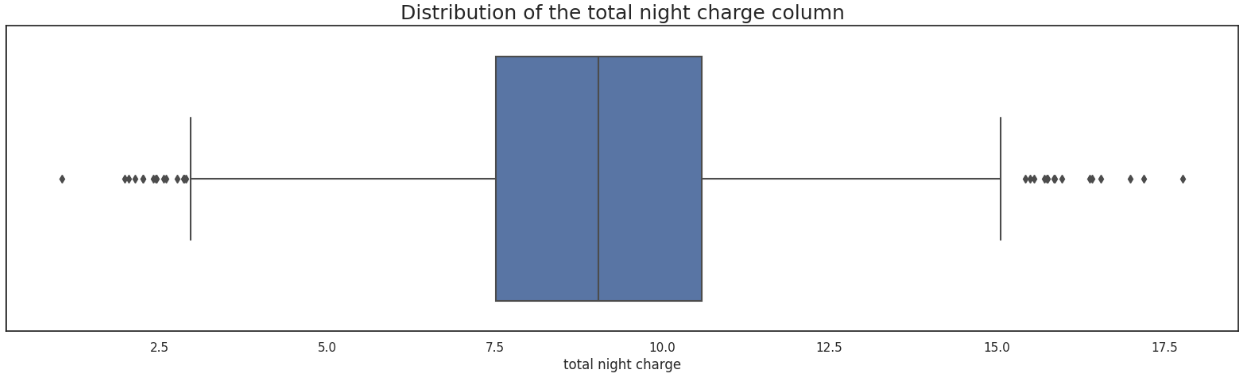
Distribution of the total eve charge column



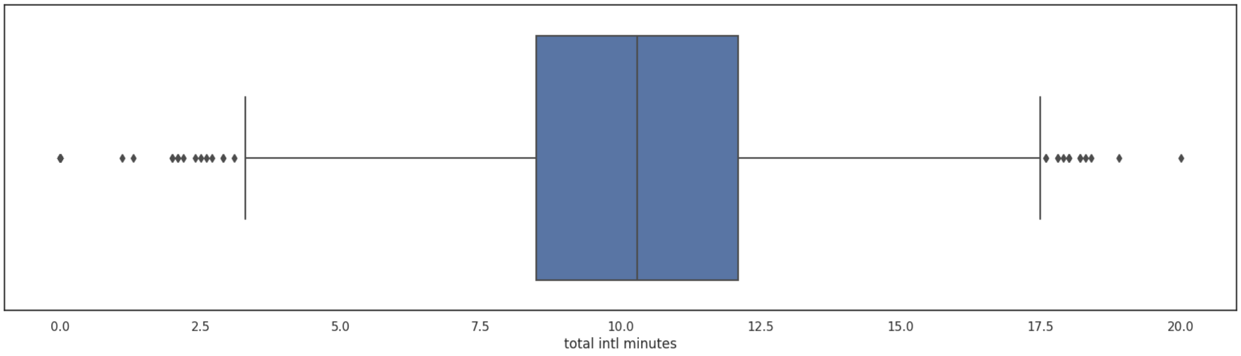
Distribution of the total night minutes column



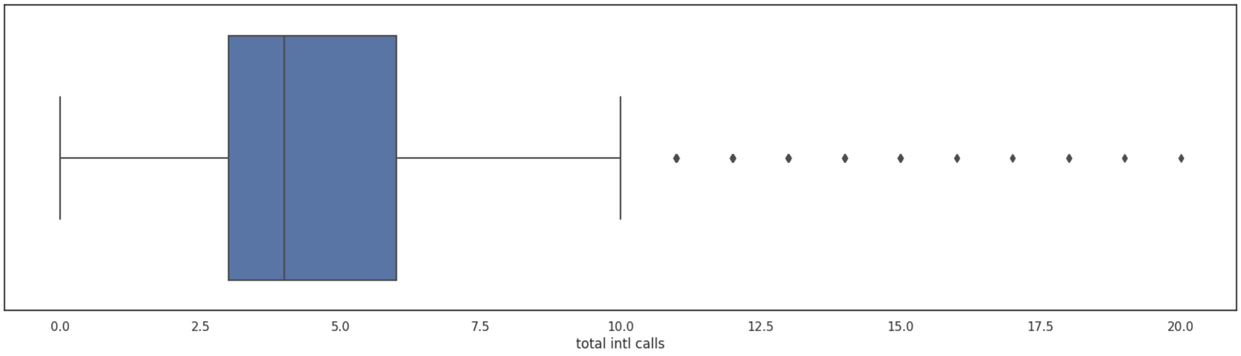
Distribution of the total night calls column



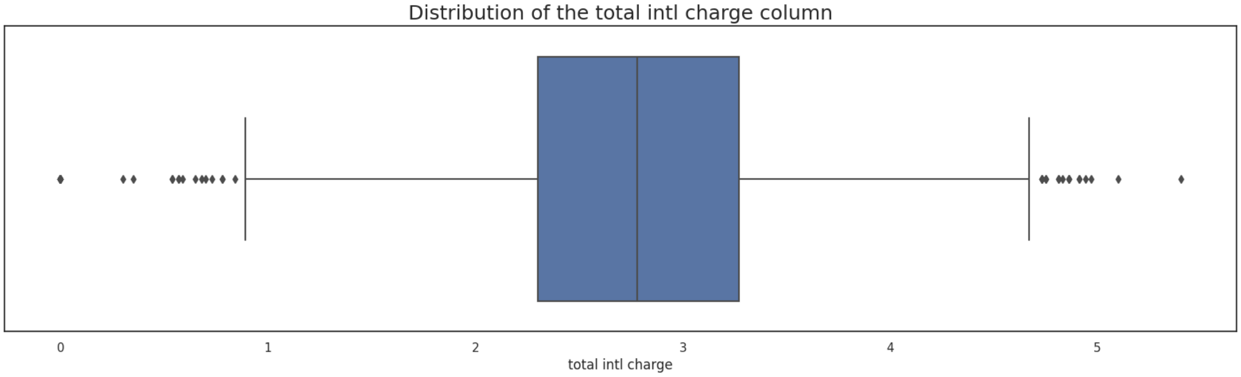
Distribution of the total night charge column



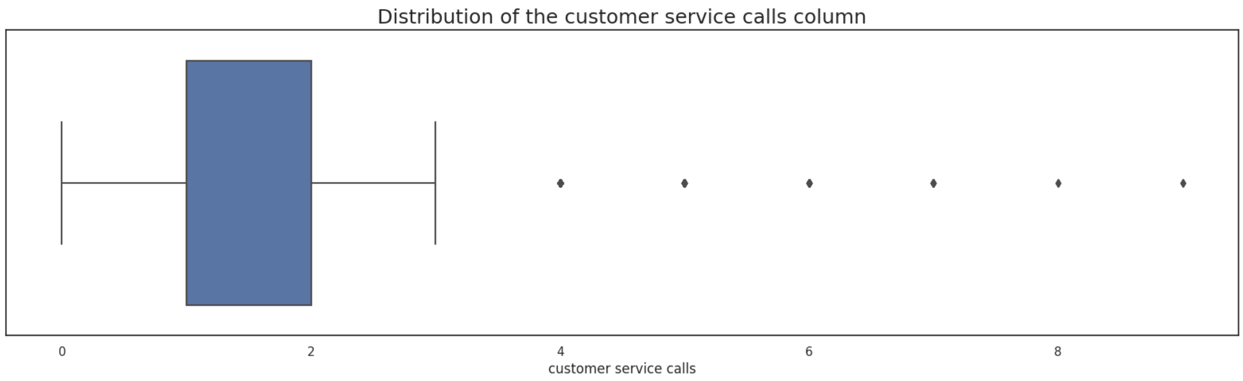
Distribution of the total intl minutes column



Distribution of the total intl calls column



Distribution of the total intl charge column



DATA PREPARATION

We will carry out our data preparation such as :

- Data Type conversion.
- Removing outliers.
- Normalizing our dataset.
- Dealing with multicolinearity.
- Spliting the data.

Data Type conversion

```
# Converting the stings in International plan and Voicemail plan into integers

df['international plan'] = df['international plan'].map({'yes': 1, 'no': 0})

df['voice mail plan'] = df['voice mail plan'].map({'yes': 1, 'no': 0})
```

```
# Converting phone number to integer and replacing the -
```

```
df['phone number'] = df['phone number'].str.replace('-', '').astype(int)

# Confirming the convertions

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>

Index: 3333 entries, KS to TN

Data columns (total 20 columns):

 #   Column               Non-Null Count  Dtype

---  ------               --------------  -----

 0   account length        3333 non-null   int64

 1   area code             3333 non-null   int64

 2   phone number          3333 non-null   int64

 3   international plan    3333 non-null   int64

 4   voice mail plan       3333 non-null   int64

 5   number vmail messages  3333 non-null   int64

 6   total day minutes     3333 non-null   float64

 7   total day calls       3333 non-null   int64

 8   total day charge      3333 non-null   float64

 9   total eve minutes     3333 non-null   float64

 10  total eve calls       3333 non-null   int64

 11  total eve charge      3333 non-null   float64

 12  total night minutes   3333 non-null   float64

 13  total night calls     3333 non-null   int64

 14  total night charge    3333 non-null   float64
 15  total intl minutes     3333 non-null   float64
 16  total intl calls       3333 non-null   int64
 17  total intl charge      3333 non-null   float64
 18  customer service calls  3333 non-null   int64
 19  churn                  3333 non-null   bool
dtypes: bool(1), float64(8), int64(11)
memory usage: 524.0+ KB
```

Removing Outliers

We remove outliers from our dataset to ensure once the data is manipulated it can yield correct results and insights.

```
# Define a function to identify outliers using IQR method
def detect_outliers_IQR(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (data < lower_bound) | (data > upper_bound)
    return outliers


# Loop over each column in the Dataframe and detect outliers
for col in df.columns:
    if np.issubdtype(df[col].dtype, np.number):

        outliers = detect_outliers_IQR(df[col])

        if outliers.any():

            print(f'Column "{col}" has {outliers.sum()}
outlier(s):\n{df[col][outliers]}')

            # drop the outliers

            df = df[~outliers]

            print(f'Outliers in column "{col}" have been dropped.')

            # plot the boxplot of the column without outliers

            fig, ax = plt.subplots()

            ax.boxplot(df[col])

            ax.set_title(f'Boxplot of column "{col}" without outliers')

            plt.show()

        else:

            print(f'Column "{col}" has no outliers.')
```

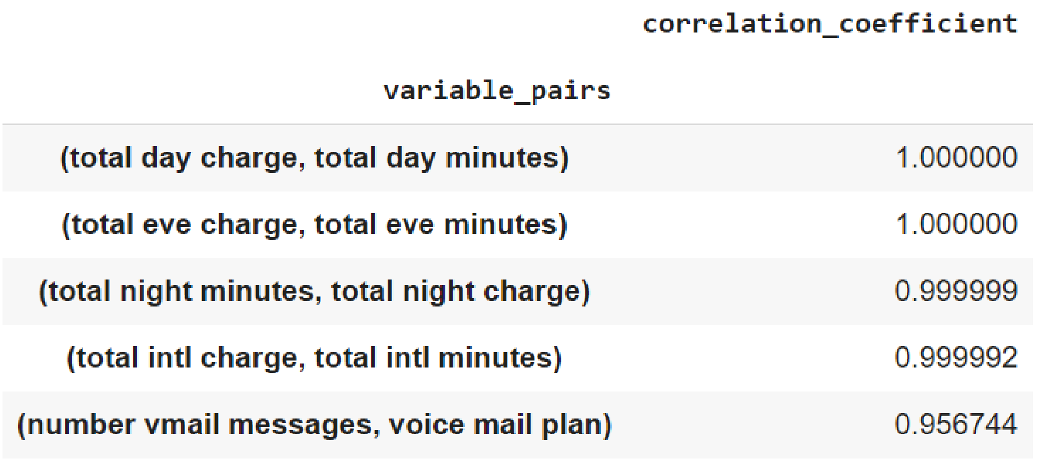
```
Column "account length" has 18 outlier(s):

state

TX    208

WY    215

SD    209

DE    224

UT    243
```

```
TX    217

VA    210

CT    212

NM    232

MI    225

WY    225

ID    224

SC    212

NC    210

DC    217

SC    209

SD    221

NY    209

Name: account length, dtype: int64

Outliers in column "account length" have been dropped.
```

Dealing with multicollinearity

```
#Identifying the variables that are highly correlated

highly_correlated_variables_df = (

   df

   .corr()

   .abs()

   .stack()
```

```
    .reset_index()

    .sort_values(0, ascending=False)

)



highly_correlated_variables_df['variable_pairs'] =
list(zip(highly_correlated_variables_df.level_0,
highly_correlated_variables_df.level_1))

highly_correlated_variables_df =
highly_correlated_variables_df.set_index('variable_pairs').drop(columns=['
level_1', 'level_0'])

highly_correlated_variables_df.columns = ['correlation_coefficient']

highly_correlated_variables_df =
highly_correlated_variables_df.drop_duplicates()



filter_condition = (highly_correlated_variables_df.correlation_coefficient
> 0.75) & (highly_correlated_variables_df.correlation_coefficient < 1)

highly_correlated_variables_df[filter_condition]
```



	correlation_coefficient
variable_pairs	
(total day charge, total day minutes)	1.000000
(total eve charge, total eve minutes)	1.000000
(total night minutes, total night charge)	0.999999
(total intl charge, total intl minutes)	0.999992
(number vmail messages, voice mail plan)	0.956744

Looking at this there is high collinearity among the following columns 'total day minutes', 'total day charge', 'total eve minutes','total eve charge', 'total night minutes', 'total night charge', 'total intl charge', 'total intl minutes'. Multicollinearity can lead to unstable or biased estimates of the model coefficients, and reduce the model's ability to generalize to new data.

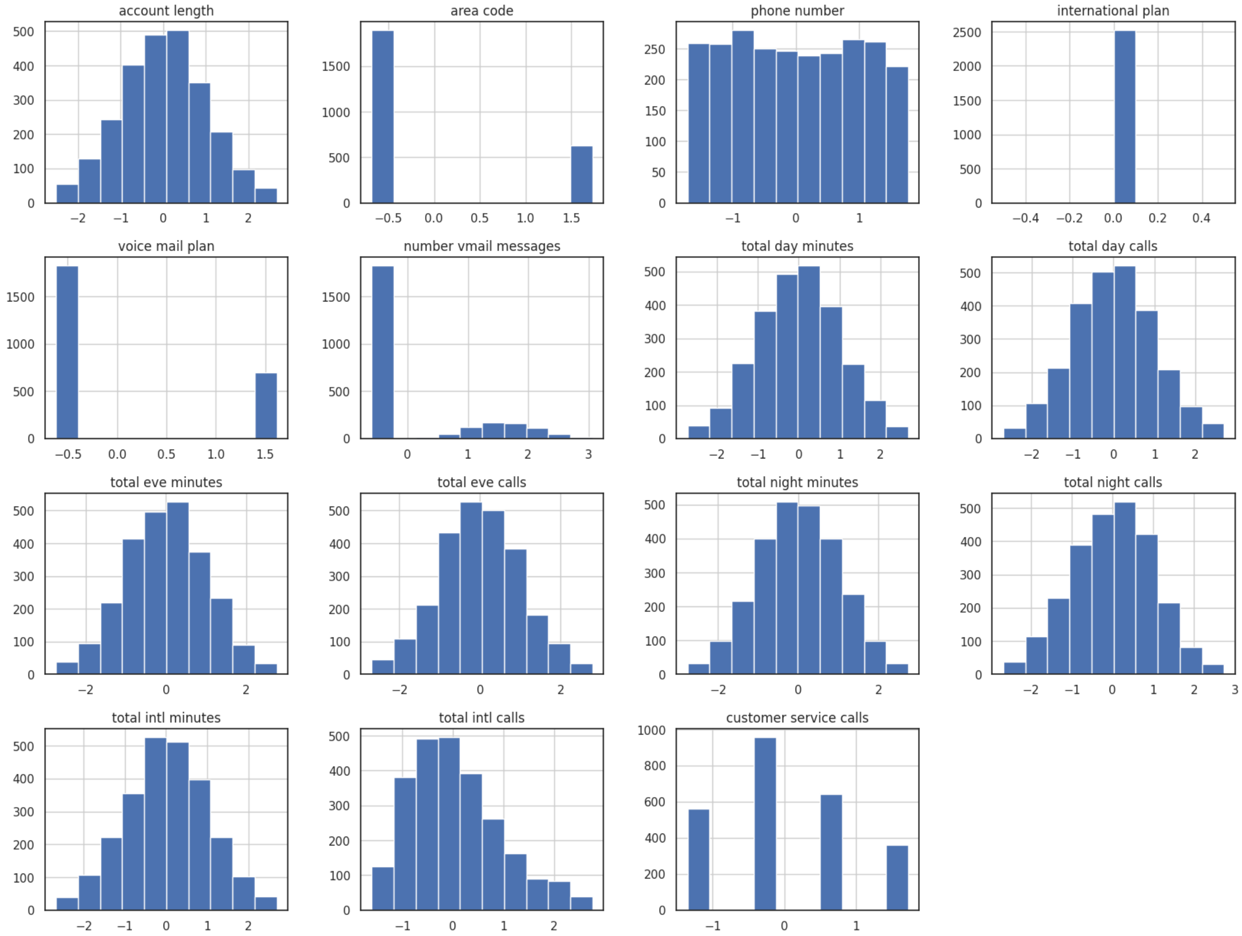
```
#Verifying the drop of columns

#Get the column names

column_names = df.columns.tolist()



#Print the column names

print(column_names)
```

```
'account length', 'area code', 'phone number', 'international plan',
'voice mail plan', 'number vmail messages', 'total day minutes', 'total
day calls', 'total eve minutes', 'total eve calls', 'total night minutes',
'total night calls', 'total intl minutes', 'total intl calls', 'customer
service calls', 'churn']
```

Normalization

```
#Select only numeric columns

numeric_cols = df.select_dtypes(include=['float64', 'int64'])



#Create an instance of the StandardScaler

scaler = StandardScaler()



#Fit and transform the data

numeric_cols_normalized = scaler.fit_transform(numeric_cols)



#Create a DataFrame from the normalized data
```

```
numeric_cols_normalized_df = pd.DataFrame(numeric_cols_normalized, 
columns=numeric_cols.columns)


#Plot histograms of the normalized data

numeric_cols_normalized_df.hist(figsize=(20,15));
```



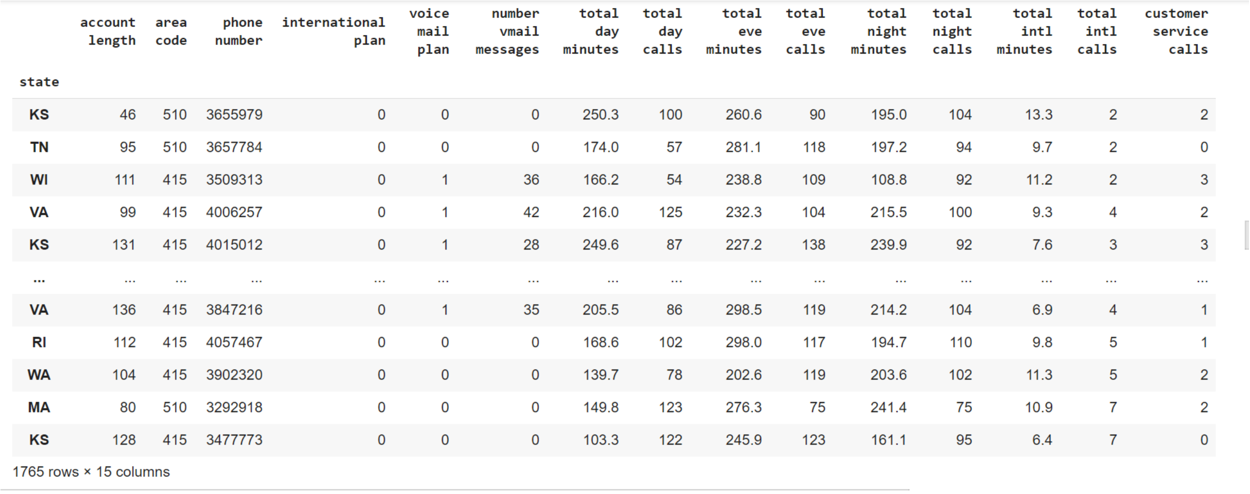
Splitting the data

```
# Split the data into X (independent variables) and y (dependent 
variable)
```

```
from sklearn.preprocessing import LabelEncoder
l=LabelEncoder()
for col in df.columns:
   if df[col].dtype=='object':
       df[col]=l.fit_transform(df[col])
X = df.drop(['churn'], axis=1)
y = df['churn']

# Split the data into training and testing sets with a 70:30 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=42)
```

```
print(len(X_train), len(X_test), len(y_train), len(y_test))
```

```
1765 757 1765 757
```



	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls
state															
KS	46	510	3655979	0	0	0	250.3	100	260.6	90	195.0	104	13.3	2	2
TN	95	510	3657784	0	0	0	174.0	57	281.1	118	197.2	94	9.7	2	0
WI	111	415	3509313	0	1	36	166.2	54	238.8	109	108.8	92	11.2	2	3
VA	99	415	4006257	0	1	42	216.0	125	232.3	104	215.5	100	9.3	4	2
KS	131	415	4015012	0	1	28	249.6	87	227.2	138	239.9	92	7.6	3	3
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
VA	136	415	3847216	0	1	35	205.5	86	298.5	119	214.2	104	6.9	4	1
RI	112	415	4057467	0	0	0	168.6	102	298.0	117	194.7	110	9.8	5	1
WA	104	415	3902320	0	0	0	139.7	78	202.6	119	203.6	102	11.3	5	2
MA	80	510	3292918	0	0	0	149.8	123	276.3	75	241.4	75	10.9	7	2
KS	128	415	3477773	0	0	0	103.3	122	245.9	123	161.1	95	6.4	7	0

1765 rows × 15 columns

Modelling

The target variable for this project is the binary variable 'Churn', indicating whether a customer has churned or not. Since the target variable is binary, a classification model is appropriate for this project. Some appropriate classification models that could be used for this project include logistic regression, decision tree, random forest, support vector machines (SVM), and neural networks. The specific model(s) chosen will depend on the size and complexity of the dataset, as well as the performance metrics and interpretability requirements of the stakeholders.

- Select modeling technique: Choose the appropriate modeling technique based on the business problem and data characteristics. In the case of customer churn, you might use a logistic regression model or a decision tree model.
- Build model: Build the model using the selected technique and the prepared data. This might involve feature selection, parameter tuning, and model evaluation.
- Assess model: Evaluate the performance of the model using various metrics such as accuracy, precision, recall, and F1-score. This will help you determine whether the model is suitable for deployment.

Decision Tree Model

It is not necessary to one-hot encode boolean columns (binary variables) like the "churn" column in your dataset. Decision trees can handle binary variables directly without the need for one-hot encoding.

In fact, one-hot encoding a binary column may introduce unnecessary complexity and redundancy in the decision tree.

Building the model

```
# Create an instance of the decision tree classifier and fit the model on the training data
```

```
clf = DecisionTreeClassifier(random_state=42)

# Fit the model on the training data

clf.fit(X_train, y_train)
```

```
DecisionTreeClassifier

DecisionTreeClassifier(random_state=42)
```

```
# Make predictions on the testing data

y_pred = clf.predict(X_test)
```

Evaluating the model Before tuning

Evaluate the model's performance using appropriate metrics such as accuracy, precision, recall, and F1-score

```
# Evaluate the model

# Calculate accuracy

accuracy = accuracy_score(y_test, y_pred)

print("Accuracy:", accuracy)


# Calculate precision

precision = precision_score(y_test, y_pred)

print("Precision:", precision)


# Calculate recall

recall = recall_score(y_test, y_pred)

print("Recall:", recall)
```

```
# Calculate F1-score
f1 = f1_score(y_test, y_pred)
print("F1-score:", f1)
```

```
Accuracy: 0.9286657859973579
Precision: 0.6037735849056604
Recall: 0.49230769230769234
F1-score: 0.5423728813559323
```

Tuning the model

Determine the optimal hyperparameters for the decision tree model using techniques such as grid search.

```
# Define the parameter grid to search
param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
}

# Create an instance of the decision tree classifier
clf = DecisionTreeClassifier(random_state=42)

# Perform grid search
grid_search = GridSearchCV(clf, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best parameters found
```

```
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'max_depth': 5, 'min_samples_leaf': 1,
'min_samples_split': 2}
```

```
# Use the best model found for predictions
best_clf = grid_search.best_estimator_
y_predd = best_clf.predict(X_test)
```

Evaluating the model After tuning

Evaluate the model's performance using appropriate metrics such as accuracy, precision, recall, and F1-score.

```
# Calculate accuracy
accuracy = accuracy_score(y_test, y_predd)
print("Accuracy:", accuracy)

# Calculate precision
precision = precision_score(y_test, y_predd)
print("Precision:", precision)

# Calculate recall
recall = recall_score(y_test, y_predd)
print("Recall:", recall)

# Calculate F1-score
f1 = f1_score(y_test, y_predd)
print("F1-score:", f1)
```

```
Accuracy: 0.9418758256274768

Precision: 0.7837837837837838

Recall: 0.4461538461538462

F1-score: 0.5686274509803922
```

The performance metrics before and after tuning the model show changes in the accuracy, precision, recall, and F1-score. Here's an explanation of each metric and the differences observed:

1. Accuracy: Accuracy measures the overall correctness of the model's predictions. It calculates the ratio of correctly predicted instances to the total number of instances.
    - Before tuning: Accuracy was 0.9287, indicating that the model correctly predicted 92.87% of the instances.
    - After tuning: Accuracy improved to 0.9419, indicating that the tuned model achieved a higher accuracy of 94.19%.
2. The increase in accuracy after tuning suggests that the model's overall prediction performance improved.
3. Precision: Precision is the ratio of true positives to the sum of true positives and false positives. It measures the proportion of correctly predicted positive instances out of all instances predicted as positive.
    - Before tuning: Precision was 0.6038, indicating that the model correctly predicted 60.38% of the positive instances.
    - After tuning: Precision improved to 0.7838, indicating that the tuned model achieved a higher precision of 78.38%.
4. The increase in precision after tuning suggests that the tuned model reduced the rate of false positives, resulting in more accurate positive predictions.

5. Recall: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances out of all actual positive instances.
   - Before tuning: Recall was 0.4923, indicating that the model correctly predicted 49.23% of the actual positive instances.
   - After tuning: Recall decreased to 0.4462, indicating that the tuned model achieved a lower recall of 44.62%.
6. The decrease in recall after tuning suggests that the tuned model may have missed some positive instances compared to the original model.
7. F1-score: F1-score is the harmonic mean of precision and recall. It provides a single metric that combines both precision and recall into a balanced measure of performance.
   - Before tuning: F1-score was 0.5424.
   - After tuning: F1-score improved to 0.5686.
8. The increase in F1-score after tuning indicates an overall improvement in the model's performance in terms of balancing precision and recall.

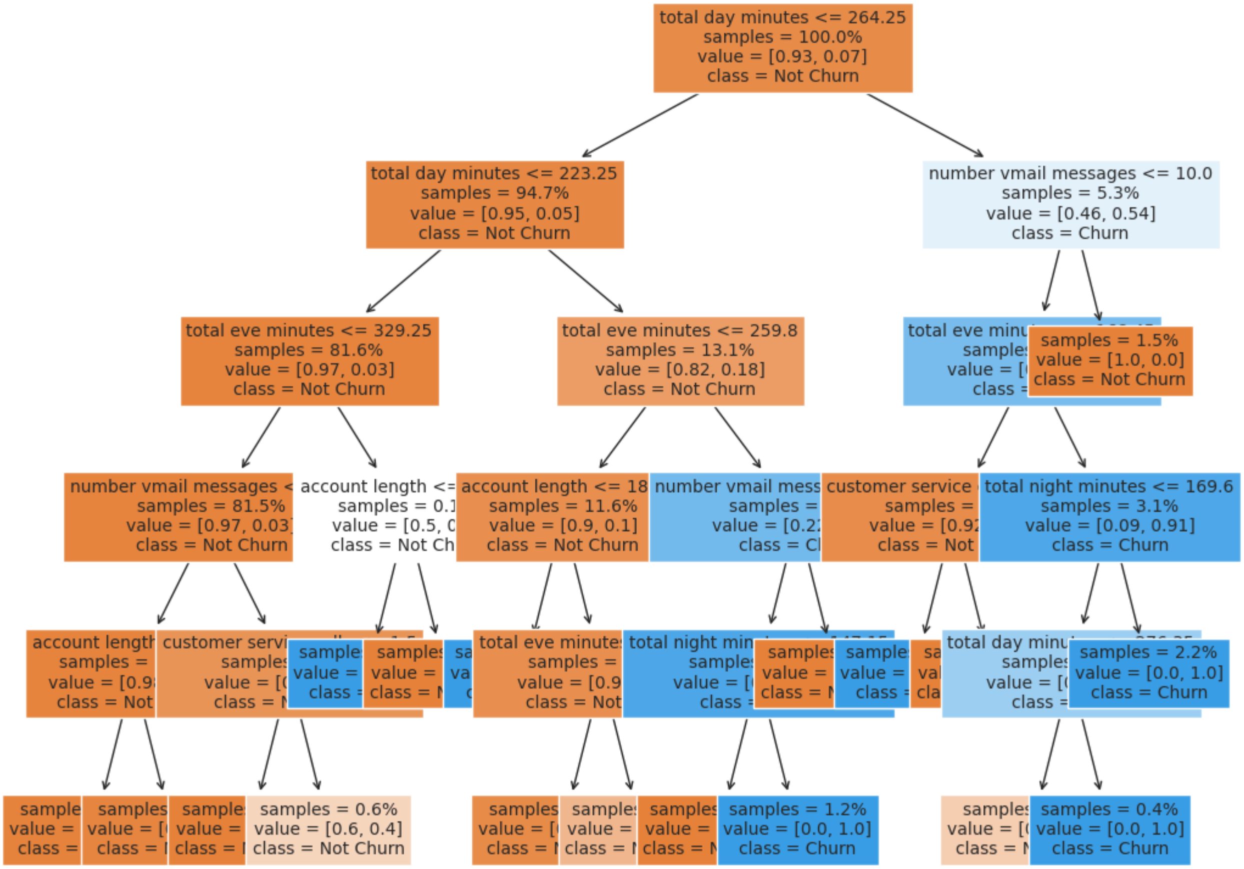
In summary, the performance metrics show that after tuning the model, there was an improvement in accuracy and precision. However, there was a slight decrease in recall, indicating that the tuned model may have sacrificed some recall to achieve higher precision. The overall F1-score increased, suggesting a better balance between precision and recall.

In this instance there is a trade off between the precision and recall rate. The objective of the project is to correctly predict a customer likely to churn hence Precision(True predicted positives) in this case is a priority rather than having a high true positive rate.

Visualizing the model

Visualize the decision tree model to gain insights into how the model is making predictions and the variables that are most important for predicting the target variable.

```
# Visualize the decision tree

plt.figure(figsize=(12, 10))

plot_tree(best_clf, filled=True, feature_names=X.columns,
class_names=['Not Churn', 'Churn'], fontsize=10, impurity=False,
precision=2, proportion=True)

plt.show()
```



Logistic Regression model

```
X = df.drop(['churn'], axis=1)
```

```
y = df['churn']

# Save dataframe column titles to list for reassigning after min max scale

cols = X.columns

# Split the data into training and testing sets with a 70:30 ratio

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,
random_state=42)
```

Scaling our dataset.

```
## Scaling our data

# Instantiate min-max scaling object

mm = MinMaxScaler()

# Fit and transform our feature dataframe

X = pd.DataFrame(mm.fit_transform(X))

# Reassign column names so new dataframe has corresponding names

X.columns = cols

##Building the model

# Instantiate a Logistic Regression model without an intercept. C is set
to an arbitrarily large number. Use 'liblinear'

#solver method.

logreg = LogisticRegression(fit_intercept = False, C = 1e12, solver =
'liblinear')

# Fit the model to our X and y training sets

logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(C=1000000000000.0, fit_intercept=False,
solver='liblinear')
```

Here we shall train and perform testing for our data.

```
# Generate model prediction data for train and test sets

y_hat_train = logreg.predict(X_train)

y_hat_test = logreg.predict(X_test)
```

```
# Pass actual test and predicted target test outcomes to function

cnf_matrix = confusion_matrix(y_test, y_hat_test)
```

```
## Evaluating Model Performance

# Find residual differences between train data and predicted train data

residuals = np.abs(y_train ^ y_hat_train)

# Print value counts of our predicted values

print(pd.Series(residuals).value_counts())

print('----------------------------------')

# Print normalized value counts of our predicted values

print(pd.Series(residuals).value_counts(normalize = True))
```

```
False    1759

True      132

Name: churn, dtype: int64

----------------------------------
```

```
False    0.930196
True     0.069804
Name: churn, dtype: float64
```

Train Set Results - 1759 False (132) True 93% Accuracy

How many times was the classifier correct on the test set? Using the code below we can find that information.

```
# Repeat previous step with test data

# Find residual differences between test data and predicted test data

residuals = np.abs(y_test ^ y_hat_test)

print(pd.Series(residuals).value_counts())

print('---------------------------------')

print(pd.Series(residuals).value_counts(normalize = True))
```

```
False    578
True      53
Name: churn, dtype: int64
---------------------------------
False    0.916006
True     0.083994
Name: churn, dtype: float64
```

Test Set Results: 578 False (53) True 91.6% Accuracy

Confusion Matrix

```
# Call confusion_matrix function from sklearn.metrics using actual y_test
and predicted y_test data sets

cnf_matrix = confusion_matrix(y_test, y_hat_test)

print('Confusion Matrix: \n', cnf_matrix)
```

```
Confusion Matrix:

 [[569   6]

 [ 47   9]]
```

```
def print_metrics(y_train, y_hat_train, y_test, y_hat_test):

    print(f'Training Precision: ', round(precision_score(y_train,
y_hat_train), 2))

    print(f'Testing Precision: ', round(precision_score(y_test,
y_hat_test),2))

    print('\n')

    print(f'Training Recall: ', round(recall_score(y_train, y_hat_train),
2))

    print(f'Testing Recall: ', round(recall_score(y_test, y_hat_test),2))

    print('\n')

    print(f'Training Accuracy: ', round(accuracy_score(y_train,
y_hat_train), 2))

    print(f'Testing Accuracy: ', round(accuracy_score(y_test,
y_hat_test),2))

    print('\n')
    print(f'Training F1-Score: ', round(f1_score(y_train, y_hat_train), 2))

    print(f'Testing F1-Score: ', round(f1_score(y_test, y_hat_test),2))

    print('\n')
```

Our performance metrics before tuning

```
# Print 4 main logistic model metrics for training and test sets
(Precision, Recall, Accuracy, F1)

print_metrics(y_train, y_hat_train, y_test, y_hat_test)
```

```
Training Precision:  0.64
Testing Precision:  0.6


Training Recall:  0.1
Testing Recall:  0.16


Training Accuracy:  0.93
Testing Accuracy:  0.92


Training F1-Score:  0.18
Testing F1-Score:  0.25
```

Tuning our model

```python
# Create range of candidate penalty hyperparameter values
penalty = ['l2']

# Create range of candidate regularization hyperparameter values
C = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

# Create dictionary hyperparameter candidates
hyperparameters = dict(C=C, penalty=penalty)

# Create grid search using 5-fold cross validation
clf = GridSearchCV(logreg, hyperparameters, cv=5)

# Fit grid search to training data
best_model = clf.fit(X_train, y_train)

# Use best hyperparameters to fit logistic regression model
logreg_tuned = LogisticRegression(penalty=best_model.best_estimator_.get_params()['penalty'], C=best_model.best_estimator_.get_params()['C'], fit_intercept=False)
logreg_tuned.fit(X_train, y_train)

# Generate model prediction data for train and test sets using tuned model
y_hat_train_tuned = logreg_tuned.predict(X_train)
```

```
y_hat_test_tuned = logreg_tuned.predict(X_test)

# Compute confusion matrix for test set using tuned model
cnf_matrix_tuned = confusion_matrix(y_test, y_hat_test_tuned)
cnf_matrix_tuned
```

```
array([[573,   2],
       [ 54,   2]])
```

We then check our metrics to see the changes after tuning our model.

```
print_metrics(y_train, y_hat_train_tuned, y_test, y_hat_test_tuned)


Training Precision:  0.71
Testing Precision:  0.5



Training Recall:  0.04
Testing Recall:  0.04



Training Accuracy:  0.93
Testing Accuracy:  0.91



Training F1-Score:  0.07
Testing F1-Score:  0.07
```

From our confusion metrics summary on our logistic model we can tell that

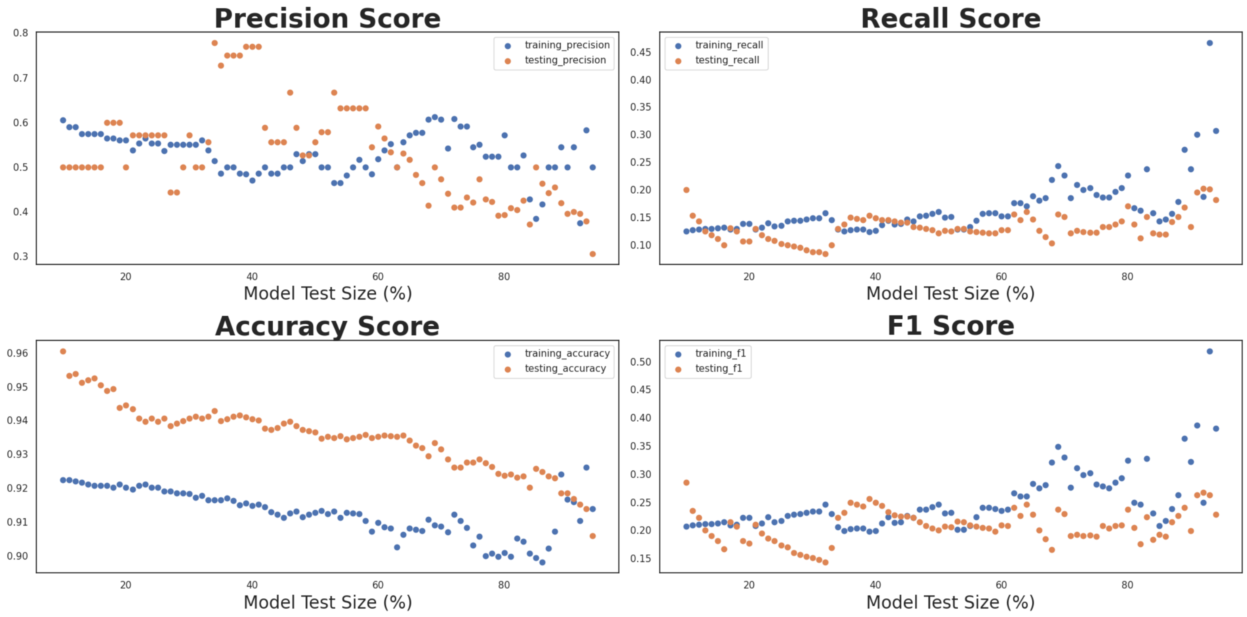
- Precision (number of relevant items selected) improved on our training data from 0.64 to 0.71 while on the testing data it dropped from 0.6 to 0.5
- Recall(number of selected items that are relevant) dropped on both our training and test data from 0.1 to 0.04 and from 0.16 to 0.04.
- Accuracy which tells us the total number of predictions a model gets right was maintained at 0.93 on our training data and dropped slightly with our testing data with a difference of 0.01.

- F1 dropped halfway on our training data and our testing data from 0.18 to 0.07 and from 0.25 to 0.07 respectively.

```
def print_metric_comparisons(X, y):

   # Create an empty list for each of the 4 classification metrics
(Precision/Recall/Accuracy/F1-Score)
   training_precision = []
   testing_precision = []
   training_recall = []
   testing_recall = []
   training_accuracy = []
   testing_accuracy = []
   training_f1 = []
   testing_f1 = []

   # Iterate through a range of test_sizes to use for our logistic
regression, using same parameters as our first logistic regression in our
notebook. Append each respective result metric to its respective list.
   for i in range(10, 95):
       X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=i/100.0, random_state = 33)
       logreg = LogisticRegression(fit_intercept=False, C=1e25,
solver='liblinear')
       model_log = logreg.fit(X_train, y_train)
       y_hat_test = logreg.predict(X_test)
       y_hat_train = logreg.predict(X_train)

       training_precision.append(precision_score(y_train, y_hat_train))
       testing_precision.append(precision_score(y_test, y_hat_test))
       training_recall.append(recall_score(y_train, y_hat_train))
       testing_recall.append(recall_score(y_test, y_hat_test))
       training_accuracy.append(accuracy_score(y_train, y_hat_train))
       testing_accuracy.append(accuracy_score(y_test, y_hat_test))
       training_f1.append(f1_score(y_train, y_hat_train))
       testing_f1.append(f1_score(y_test, y_hat_test))
```

```
 # Use subplots to create a scatter plot of each of the 4 metrics.
    plt.figure(figsize = (20, 10))
    plt.subplot(221)
    plt.title('Precision Score', fontweight = 'bold', fontsize = 30)
    # Scatter plot training precision list
    plt.scatter(list(range(10, 95)), training_precision,
label='training_precision')
    # Scatter plot test precision list
    plt.scatter(list(range(10, 95)), testing_precision,
label='testing_precision')
    plt.xlabel('Model Test Size (%)', fontsize = 20)
    plt.legend(loc = 'best')

    plt.subplot(222)
    plt.title('Recall Score', fontweight = 'bold', fontsize = 30)
    # Scatter plot training recall list
    plt.scatter(list(range(10, 95)), training_recall,
label='training_recall')
    # Scatter plot test recall list
    plt.scatter(list(range(10, 95)), testing_recall,
label='testing_recall')
    plt.xlabel('Model Test Size (%)', fontsize = 20)
    plt.legend(loc = 'best')

    plt.subplot(223)
    plt.title('Accuracy Score', fontweight = 'bold', fontsize = 30)
    # Scatter plot training accuracy list
    plt.scatter(list(range(10, 95)), training_accuracy,
label='training_accuracy')
    # Scatter plot test accuracy list
    plt.scatter(list(range(10, 95)), testing_accuracy,
label='testing_accuracy')
    plt.xlabel('Model Test Size (%)', fontsize = 20)
    plt.legend(loc = 'best')

    plt.subplot(224)
    plt.title('F1 Score', fontweight = 'bold', fontsize = 30)
    # Scatter plot training f1-score list
    plt.scatter(list(range(10, 95)), training_f1, label='training_f1')
    # Scatter plot testing f1-score list
    plt.scatter(list(range(10, 95)), testing_f1, label='testing_f1')
    plt.xlabel('Model Test Size (%)', fontsize = 20)
    plt.legend(loc = 'best')
```

```
# Print residual scatter plot for 4 main logistic model metrics, iterating
through the model and passing multiple
# test-size objects to visualize effects of train/test size on model
performance
print_metric_comparisons(X, y)
```

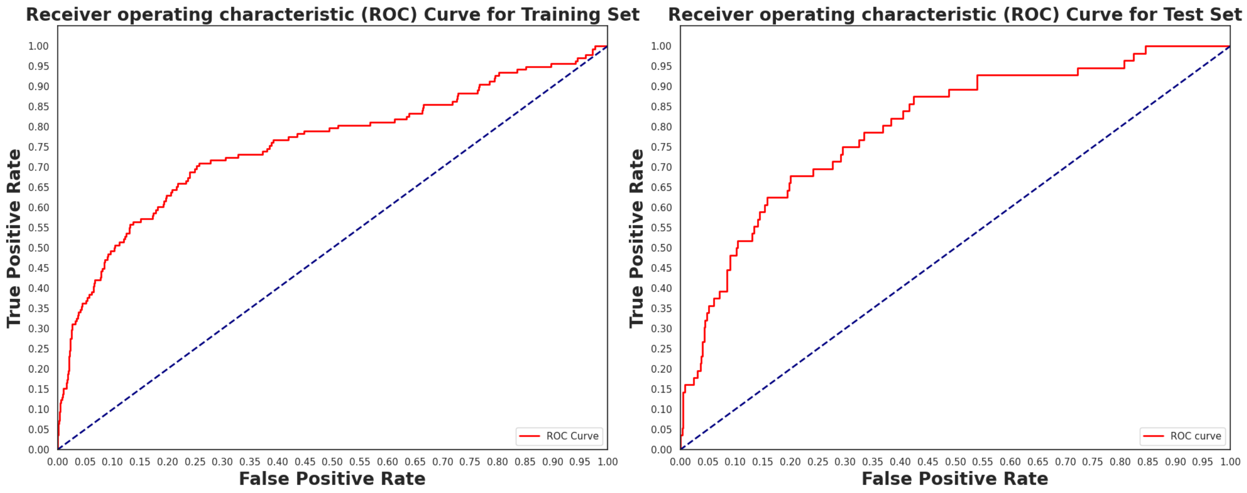


```
def plot_auc(model, X_train, X_test, y_train, y_test):


    # Calculate probability score of each point in training set

    y_train_score = model.decision_function(X_train)
    # Calculate false positive rate, true positive rate, and thresholds for training set
    train_fpr, train_tpr, train_thresholds = roc_curve(y_train, y_train_score)
    # Calculate probability score of each point in test set
    y_test_score = model.decision_function(X_test)
    # Calculate false positive rate, true positive rate, and thresholds for test set
    test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_score)

    # Print Area-Under-Curve scores
    print('Training AUC: {}'.format(auc(train_fpr, train_tpr)))
    print('Test AUC: {}'.format(auc(test_fpr, test_tpr)))

    plt.figure(figsize = (20, 8))
    lw = 2

    # Use Train False/True Positive ratios to plot receiver operating characteristic curve for training set
    plt.subplot(121)
    plt.plot(train_fpr, train_tpr, color = 'red', lw = lw, label = 'ROC Curve')
    # Plot positive line w/ slope = 1 for ROC-curve reference
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
```

```python
    plt.ylim([0.0, 1.05])
    plt.yticks([i/20.0 for i in range(21)])
    plt.xticks([i/20.0 for i in range(21)])
    plt.xlabel('False Positive Rate', fontsize = 20, fontweight = 'bold')
    plt.ylabel('True Positive Rate', fontsize = 20, fontweight = 'bold')
    plt.title('Receiver operating characteristic (ROC) Curve for Training Set', fontweight = 'bold', fontsize = 20)
    plt.legend(loc='lower right')

    # Use Test False/True positive ratios to plot receiver operating characteristic curve for test set
    plt.subplot(122)
    plt.plot(test_fpr, test_tpr, color='red',
        lw=lw, label='ROC curve')
    # Plot positive line w/ slope = 1 for ROC-curve reference
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.yticks([i/20.0 for i in range(21)])
    plt.xticks([i/20.0 for i in range(21)])
    plt.xlabel('False Positive Rate', fontweight = 'bold', fontsize = 20)
    plt.ylabel('True Positive Rate', fontweight = 'bold', fontsize = 20)
    plt.title('Receiver operating characteristic (ROC) Curve for Test Set', fontweight = 'bold', fontsize = 20)
    plt.legend(loc='lower right')
    plt.tight_layout()
```

```
plot_auc(logreg, X_train, X_test, y_train, y_test)
```

```
Training AUC: 0.7543548533776465

Test AUC: 0.7993167701863355
```



From this we can conclude that the Area Under Curve is above the 0.05 score making it a good AUC score This is both the case on our training data and testing data.

Gaussian Naive Bayes

This model is used when working with continuous data .

An assumption made is that our variables are continuous which are also distributed according to a normal distribution (or Gaussian) distribution.

Since we normalized our variables, we shall model using Gaussian Naive Bayes.

Building the model

```
# Define the model
GNB = GaussianNB()

# Train model
GNB.fit(X_train, y_train)

# We predict target values
Y_predict = GNB.predict(X_test)
```

Evaluating the model before tuning

```
# Print the Classification report of test set
print(classification_report(y_test, Y_predict))
# Calculate accuracy
accuracy = accuracy_score(y_test, Y_predict)
print("Accuracy:", accuracy)

# Calculate precision
precision = precision_score(y_test, Y_predict)
print("Precision:", precision)

# Calculate recall
recall = recall_score(y_test, Y_predict)
```

```
print("Recall:", recall)

# Calculate F1-score
f1 = f1_score(y_test, Y_predict)
print("F1-score:", f1)
```

```
              precision    recall  f1-score   support

       False       0.93      1.00      0.96       575
        True       0.87      0.23      0.37        56

    accuracy                           0.93       631
   macro avg       0.90      0.61      0.66       631
weighted avg       0.92      0.93      0.91       631

Accuracy: 0.9286846275752774
Precision: 0.8666666666666667
Recall: 0.23214285714285715
F1-score: 0.3661971830985915
```

Tuning the model

Determine the optimal hyperparameters for the naive bayes model using StandardScaler.

```
# Build a pipeline with StandardScaler and GaussianNB
from sklearn.pipeline import Pipeline
scaled_pipeline_2 = Pipeline([('ss', StandardScaler()),
                             ('GB', GaussianNB())])
scaled_pipeline_2
```

Pipeline

StandardScaler

GaussianNB

```
# Fit the training data to pipeline
scaled_pipeline_2.fit(X_train, y_train)
# predict using our tuned model
y_predict2 = scaled_pipeline_2.predict(X_test)
```

Evaluating the model

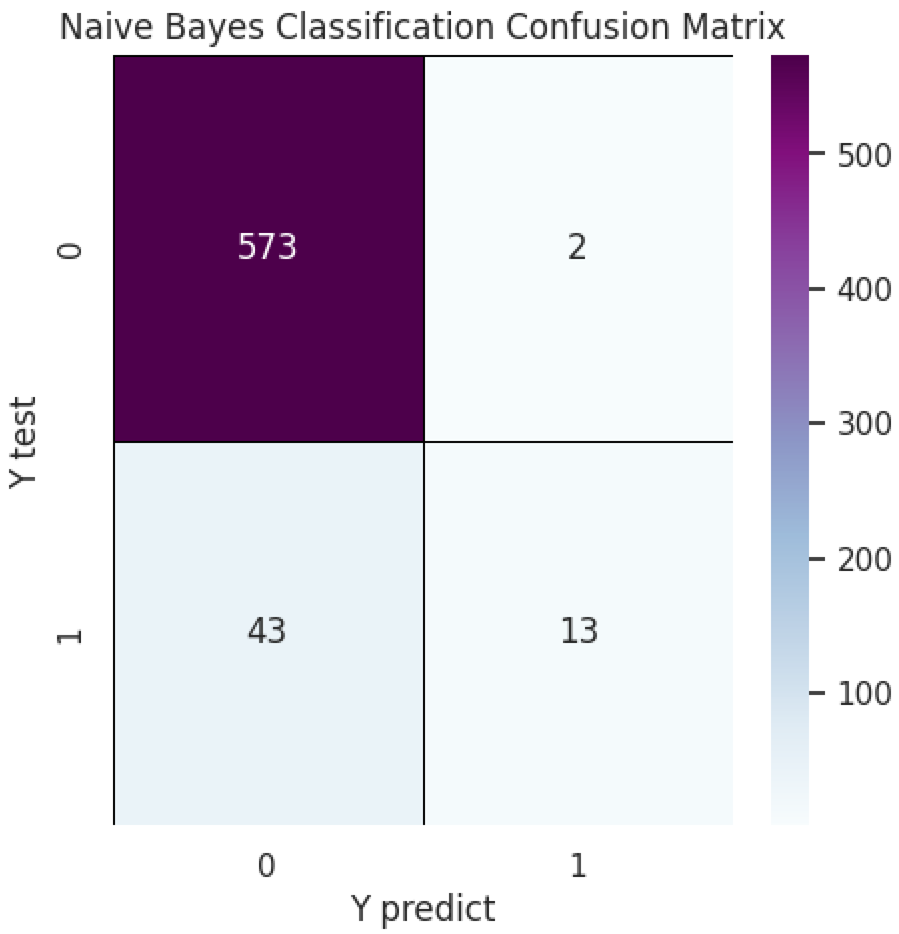
Evaluate the model's performance using appropriate metrics such as accuracy, precision, recall, and F1-score.

```
# Print the Classification report of test set
print(classification_report(y_test, y_predict2))
# Calculate accuracy
accuracy = accuracy_score(y_test, y_predict2)
print("Accuracy:", accuracy)

# Calculate precision
precision = precision_score(y_test, y_predict2)
print("Precision:", precision)

# Calculate recall
recall = recall_score(y_test, y_predict2)
print("Recall:", recall)

# Calculate F1-score
f1 = f1_score(y_test, y_predict2)
```

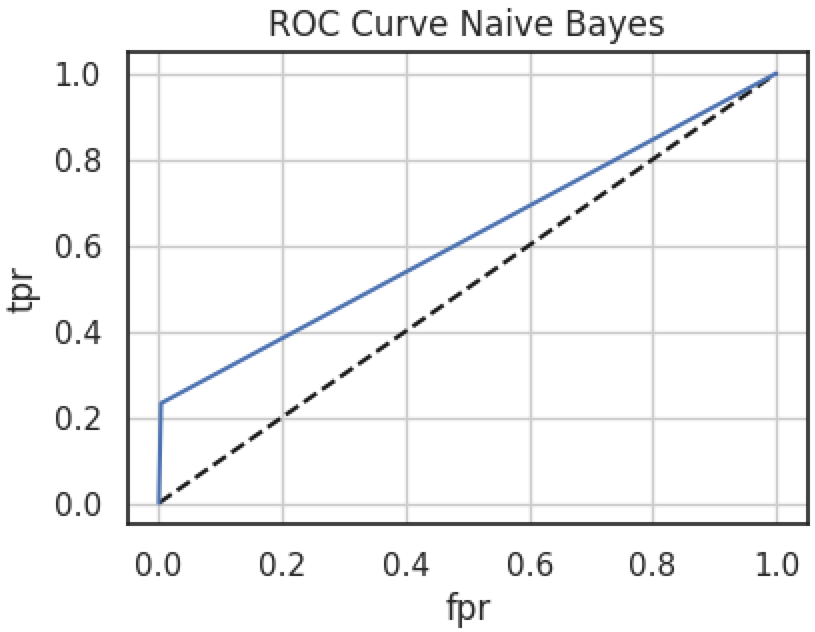
```
print("F1-score:", f1)
```

```
                  precision   recall    f1-score   support

       False       0.94      1.00      0.97       575
        True       1.00      0.36      0.53        56

    accuracy                           0.94       631
   macro avg       0.97      0.68      0.75       631
weighted avg       0.95      0.94      0.93       631

Accuracy: 0.9429477020602218
Precision: 1.0
Recall: 0.35714285714285715
F1-score: 0.5263157894736842
```

```
# plot a confusion matrix to view a summary of prediction results
nbcla_cm = confusion_matrix(y_test, Y_predict)
f, ax = plt.subplots(figsize=(5,5))
sns.heatmap(nbcla_cm, annot=True, linewidth=0.7, linecolor='black',
fmt='g', ax=ax, cmap="BuPu")
plt.title('Naive Bayes Classification Confusion Matrix')
plt.xlabel('Y predict')
plt.ylabel('Y test')
plt.show()
```



```
# plot the Receiver Operating Characteristic curve which shows the
performance of our model
fpr, tpr, thresholds = roc_curve(y_test, Y_predict)
plt.subplot(322)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='ANN')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('ROC Curve Naive Bayes')
plt.grid(True)
plt.subplots_adjust(top=2, bottom=0.08, left=0.10, right=1.4, hspace=0.45,
wspace=0.45)
plt.show()
```



From the above curve, the ROC is approaching 1 which is good. It means that our model is working well.

Observations

We see that there is an increase in all of our performance metrics after tuning indicating that our model is working well.

The accuracy of 0.942 indicates that our model is very good in prediction and results can be relied upon.

What is our best model and why ?

The Decision Tree Model is the Best model out of the three models done since i has the highest precision of 0.85 and F1 Score of 0.64 .

Conclusions

We can conclude a number of things from the previous visuals shown:

- The account length w of 90 days and above has more likelyhood of churning.
- The area code 415 records.
- An increase of charges leads to customers churning.

Recommendations

What can the company do to retain customers and what retention strategies can be adopted:

- Customers who use SyriaTel should be highly encouraged to get onto the international plan.
- A value add on should be given to customers when they start approaching 90 days and above of being members.
- Customer-centric Approach: provide personalized and tailored services that enhance the customer experience.
- Quality Assurance: maintain high-quality customer service across all touchpoints.
- By conducting a thorough analysis, evaluating costs, considering customer value, assessing revenue impacts and exploring value-added services the company can strategically lower call charges while main.

Retention Strategies :

A couple of retention strategies SyriaTel can adopt to maintain its customers:

- Personalized Offers: Tailored offers and discounts to individual customers based on their usage patterns, preferences, and loyalty.
- Proactive Customer Support: to include regular check-ins, timely responses to queries, and proactive troubleshooting.
- Value-added Services that enhance the overall customer experience.
- Provide special retention offers to customers who express their intention to switch to another provider.
- Regularly review and adjust pricing strategies to remain competitive in the market.