Final Project Submission

Please fill out:

- Student name:Maureen Awino Oketch
- Student pace: part time
- Instructor name:Faith Rotich

SyriaTel Customer Churn Prediction

Project Overview

1. Business Problem

Maintaining customer retention holds immense value for businesses, particularly within the competitive telecommunications industry. Research indicates that the costs associated with acquiring new customers can significantly outweigh those of retaining current ones.

This project aims to create a predictive model for SyriaTel, enabling the identification of customers prone to churning. By analyzing diverse customer data encompassing demographics, usage patterns, complaints, and billing records, we seek to unveil correlations and factors influencing churn.

Through comprehensive exploratory data analysis and meticulous feature selection, we aim to pinpoint the primary drivers of churn and construct a robust machine learning model. The model's performance will undergo evaluation using pertinent metrics before its deployment into operational use.

This initiative will empower SyriaTel to actively retain customers at risk of leaving, thereby mitigating financial losses attributed to customer attrition. Continuous monitoring and refinement of the model will be undertaken to ensure its efficacy in predicting churn and facilitating targeted retention strategies.

1.1 Problem Statement

SyriaTel has faced significant losses due to a notable churn rate. The goal of this project is to construct a precise predictive model that pinpoints customers at risk of churning within SyriaTel, a telecommunications company.

Through proactive identification of potential service discontinuations, the aim is to diminish customer attrition and uphold a larger customer base. Ultimately, this endeavor strives to assist SyriaTel in mitigating financial losses attributed to churn, elevating overall customer retention rates, and refining business strategies to bolster profitability.

1.2 Business Objectives

The primary business objectives of this project for SyriaTel are;

- 1.To understand the factors that contribute to customer churn.'
- 2.To understand the attributes of the customer who churn
- 3.To enhance overall customer retention.

DATA UNDERSTANDING

Summary of the Features in the Dataset

- State: This represents the geographic location of the customer.
- Account Length: This column indicates the length of time a customer has held their account.
- Area Code: The area code of the customer's phone number.
- Phone Number: This is usually a unique identifier for customers
- International Plan: A binary variable that indicates whether the customer has an international calling plan.
- Voice Mail Plan: A binary variable indicating whether the customer has a voicemail plan.
- Number Vmail Messages: This represents the number of voicemail messages received by the customer.
- Total Day Minutes, Total Day Calls, Total Day Charge: These columns contain information about the customer's usage during the daytime.
- Total Eve Minutes, Total Eve Calls, Total Eve Charge: These columns contain information about the customer's usage in the evening.
- Total Night Minutes, Total Night Calls, Total Night Charge: These columns contain information about nighttime usage.
- Total Intl Minutes, Total Intl Calls, Total Intl Charge: These columns relate to international usage.
- Customer Service Calls: The number of customer service calls made by the customer.
- Churn: True if the customer terminated their contract, otherwise false.(This is the target variable and all the other columns are potential features for modeling.)

Data Loading

In [43]:
```
# importing relevant libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, co
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.feature_selection import SelectFromModel
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoosting
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
```

In [44]:
```
# Importing the dataset
df = pd.read_csv('Syria customerchurn.csv')
df.head()
```

Out[44]:

	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 calls	to cha
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12

5 rows × 21 columns

In [45]:
```
df.shape
```

Out[45]:
```
(3333, 21)
```

In [46]:
```
df.columns
```

Out[46]:
```
Index(['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'],
      dtype='object')
```

In [47]:
```
df.describe()
```

Out[47]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	to
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348	100.
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.

In [48]:
```
# Understand the information about our dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [49]:
```
df.isna().sum()
```

Out[49]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

we have noted ther are no missing values in our data set.

In [50]:
```
# checking for duplicates
df.duplicated().sum()
```

Out[50]:
```
0
```

There is no duplicates as well in our dataset

In [51]:
```
df.select_dtypes('object')
```

Out[51]:

	state	phone number	international plan	voice mail plan
0	KS	382-4657	no	yes
1	OH	371-7191	no	yes
2	NJ	358-1921	no	no
3	OH	375-9999	yes	no
4	OK	330-6626	yes	no
...	...	...	...	...
3328	AZ	414-4276	no	yes
3329	WV	370-3271	no	no
3330	RI	328-8230	no	no
3331	CT	364-6381	yes	no
3332	TN	400-4344	no	yes

3333 rows × 4 columns

In [52]:
```
df.select_dtypes('int64')
```

Out[52]:

	account length	area code	number vmail messages	total day calls	total eve calls	total night calls	total intl calls	customer service calls
0	128	415	25	110	99	91	3	1
1	107	415	26	123	103	103	3	1
2	137	415	0	114	110	104	5	0
3	84	408	0	71	88	89	7	2
4	75	415	0	113	122	121	3	3
...	...	...	...	...	...	...	...	...
3328	192	415	36	77	126	83	6	2
3329	68	415	0	57	55	123	4	3
3330	28	510	0	109	58	91	6	2
3331	184	510	0	105	84	137	10	2
3332	74	415	25	113	82	77	4	0

3333 rows × 8 columns

In [53]:
```
df.select_dtypes('float64')
```

Out[53]:

	total day minutes	total day charge	total eve minutes	total eve charge	total night minutes	total night charge	total intl minutes	total intl charge
0	265.1	45.07	197.4	16.78	244.7	11.01	10.0	2.70
1	161.6	27.47	195.5	16.62	254.4	11.45	13.7	3.70
2	243.4	41.38	121.2	10.30	162.6	7.32	12.2	3.29
3	299.4	50.90	61.9	5.26	196.9	8.86	6.6	1.78
4	166.7	28.34	148.3	12.61	186.9	8.41	10.1	2.73
...	...	...	...	...	...	...	...	...
3328	156.2	26.55	215.5	18.32	279.1	12.56	9.9	2.67
3329	231.1	39.29	153.4	13.04	191.3	8.61	9.6	2.59
3330	180.8	30.74	288.8	24.55	191.9	8.64	14.1	3.81
3331	213.8	36.35	159.6	13.57	139.2	6.26	5.0	1.35
3332	234.4	39.85	265.9	22.60	241.4	10.86	13.7	3.70

3333 rows × 8 columns

In [54]:
```
df.select_dtypes('bool')
```

Out[54]:

	churn
0	False

	churn
1	False
2	False
3	False
4	False
...	...
3328	False
3329	False
3330	False
3331	False
3332	False

3333 rows × 1 columns

Finding 1: Converting data type

After reviewing the data, we discovered an issue with the 'area code' column. Despite being listed as integers, the values within it are more like placeholders or labels rather than meaningful numerical data. To safeguard our predictive modeling process from potential interference, we've decided to convert this column into a string data type. This change ensures that the 'area code' is seen as a categorical variable without numerical implications. This adjustment is crucial for preserving the accuracy of our predictive model, particularly when it depends on numerical inputs, preventing any confusion about the 'area code' being a quantitative factor

In [55]:
```
# Convert the 'area code' column to an object (string) column
df['area code'] = df['area code'].astype(str)
df.select_dtypes('object')
```

Out[55]:

	state	area code	phone number	international plan	voice mail plan
0	KS	415	382-4657	no	yes
1	OH	415	371-7191	no	yes
2	NJ	415	358-1921	no	no
3	OH	408	375-9999	yes	no
4	OK	415	330-6626	yes	no
...	...	...	...	...	...
3328	AZ	415	414-4276	no	yes
3329	WV	415	370-3271	no	no
3330	RI	510	328-8230	no	no
3331	CT	510	364-6381	yes	no
3332	TN	415	400-4344	no	yes

3333 rows × 5 columns

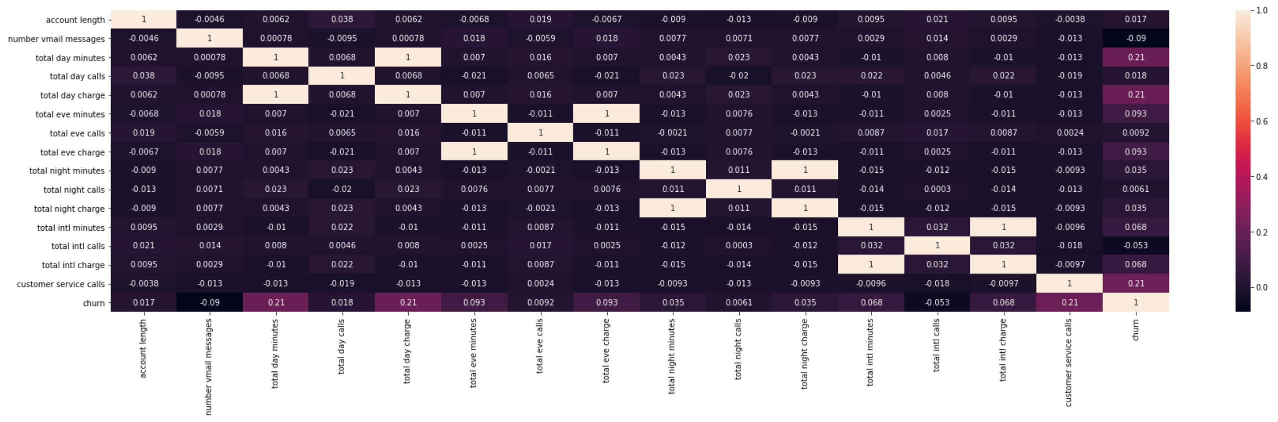
In [56]:
```python
# Check for the unique values
df.nunique()
```

Out[56]:
```
state                        51
account length              212
area code                     3
phone number               3333
international plan            2
voice mail plan               2
number vmail messages        46
total day minutes          1667
total day calls             119
total day charge           1667
total eve minutes          1611
total eve calls             123
total eve charge           1440
total night minutes        1591
total night calls           120
total night charge          933
total intl minutes          162
total intl calls             21
total intl charge           162
customer service calls       10
churn                         2
dtype: int64
```

Exploratory Data Analysis

Multivariate Analysis

In [57]:
```python
# Plot a heatmap to check how the columns are correlated
plt.rcParams['figure.figsize'] = (30, 7)
sns.heatmap(df.corr(), annot=True)
plt.show()
```

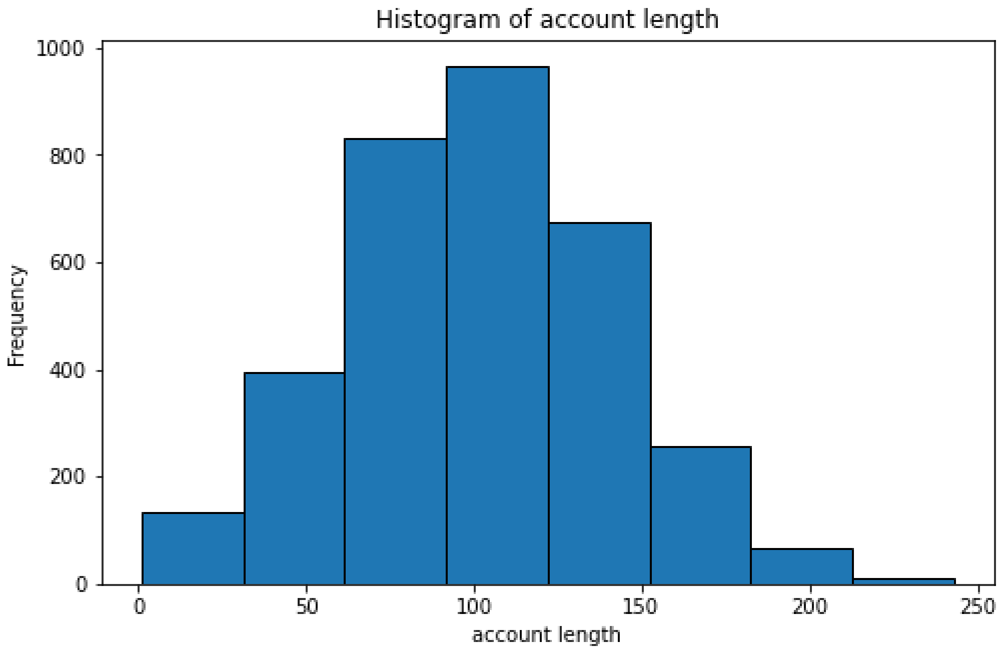


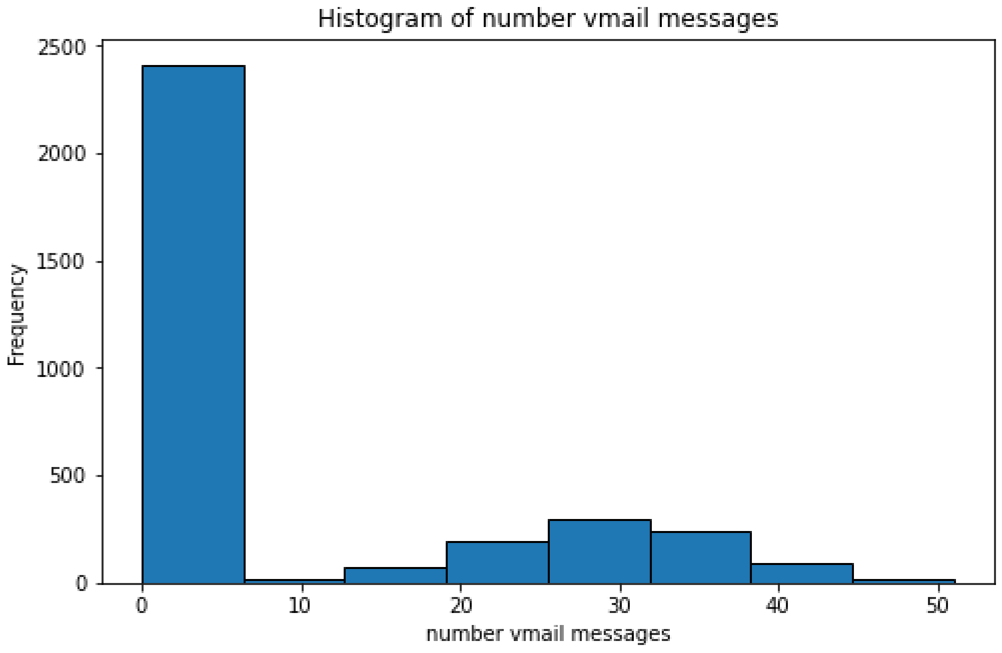
High correlation - Multicollinearity

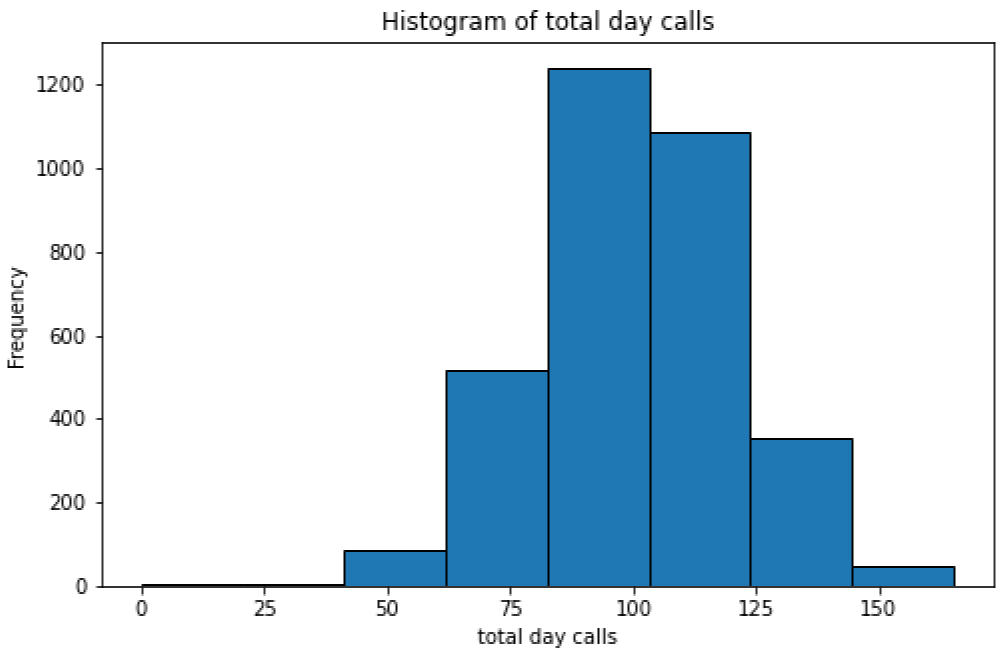
From the heatmap representation of the data revealed that several columns exhibit high levels of correlation with each other which indicates the presence of multicollinearity, a condition where independent variables in our dataset are highly interrelated. Multicollinearity can make it challenging to discern the unique impact of each independent variable on the dependent variable in our modeling and may result to overfitting problems, particularly when employing Logistic

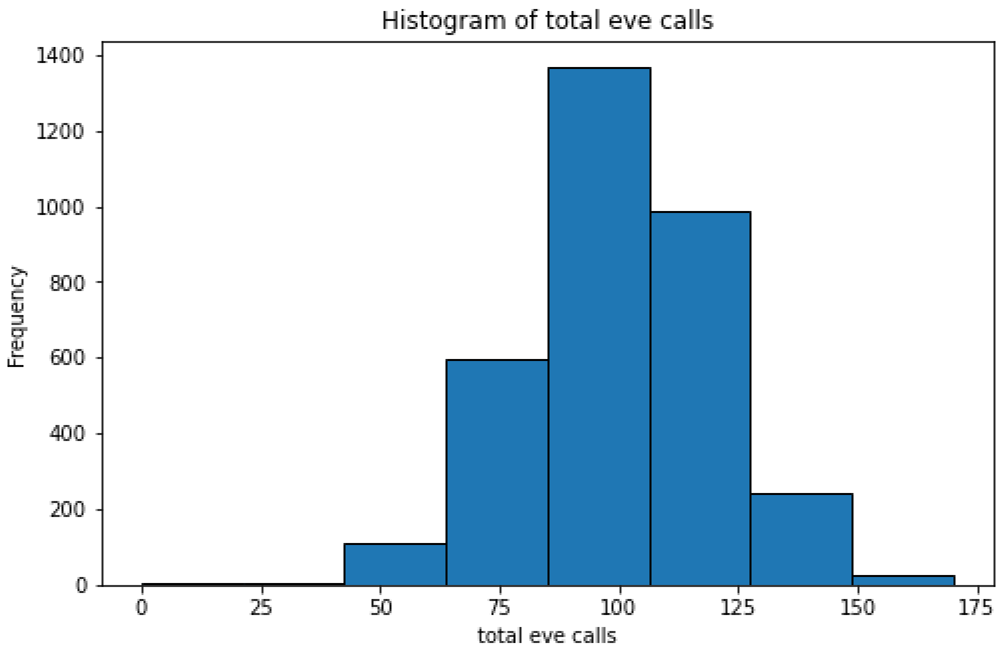
Regression, as this method is sensitive to multicollinearity.this will e dealt with during our feature selection.

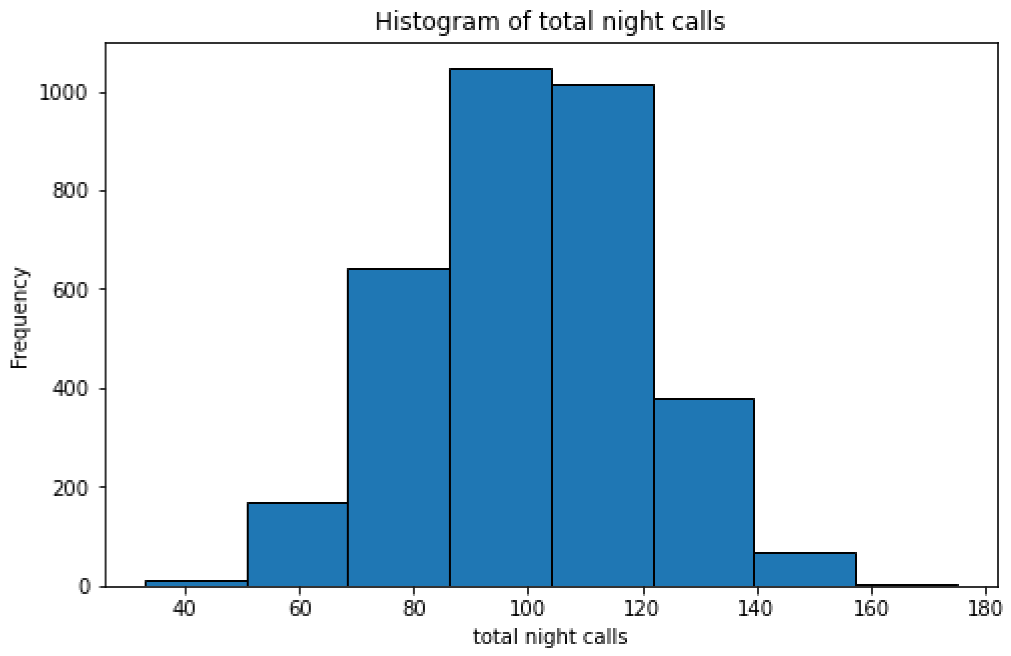
In [69]:
```
# Plot Histograms for numerical features to visualize their distribution

numerical_features = df.select_dtypes('int64', 'float64')
for feature in numerical_features:
    plt.figure(figsize=(8, 5))  # Set the figure size
    plt.hist(df[feature], bins=8, edgecolor='k', )  # Customize the number of bins
    plt.title(f'Histogram of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()
```

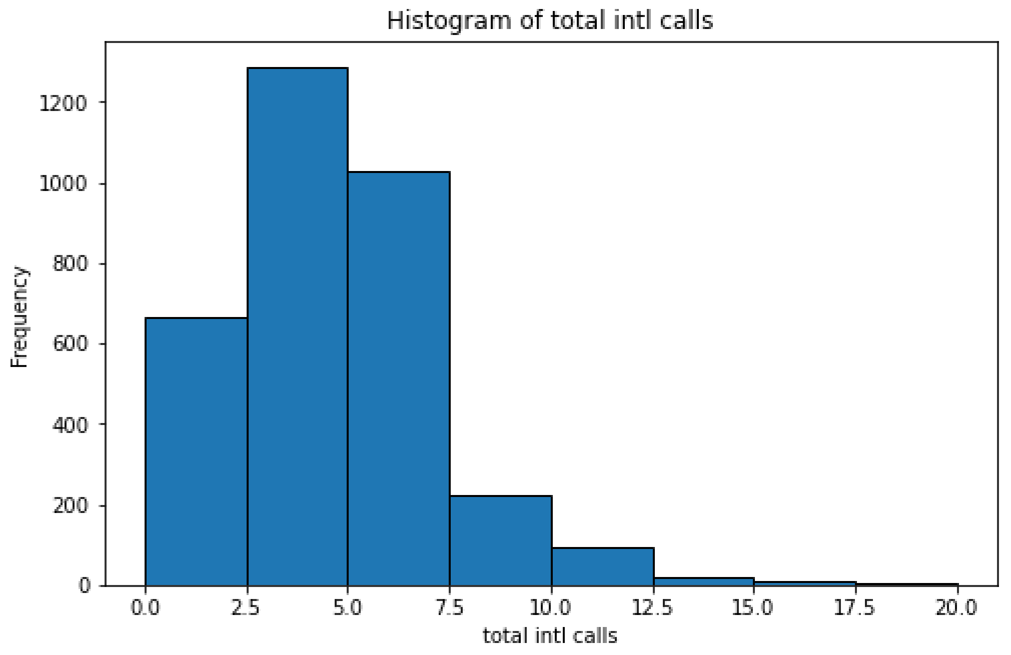


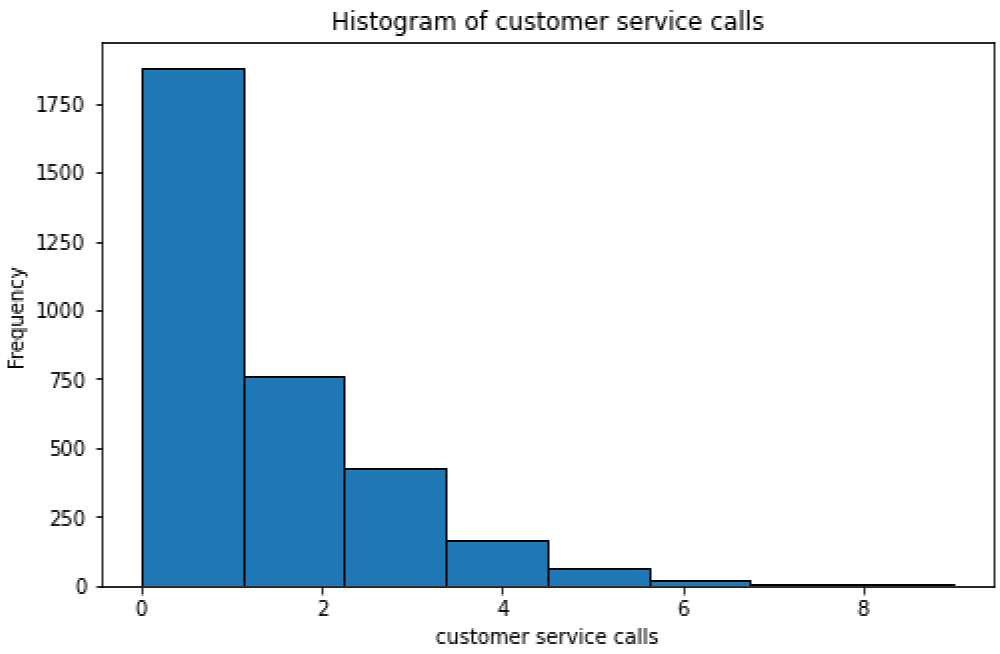



Histogram of total day calls
Frequency
0
200
400
600
800
1000
1200
0
25
50
75
100
125
150
total day calls

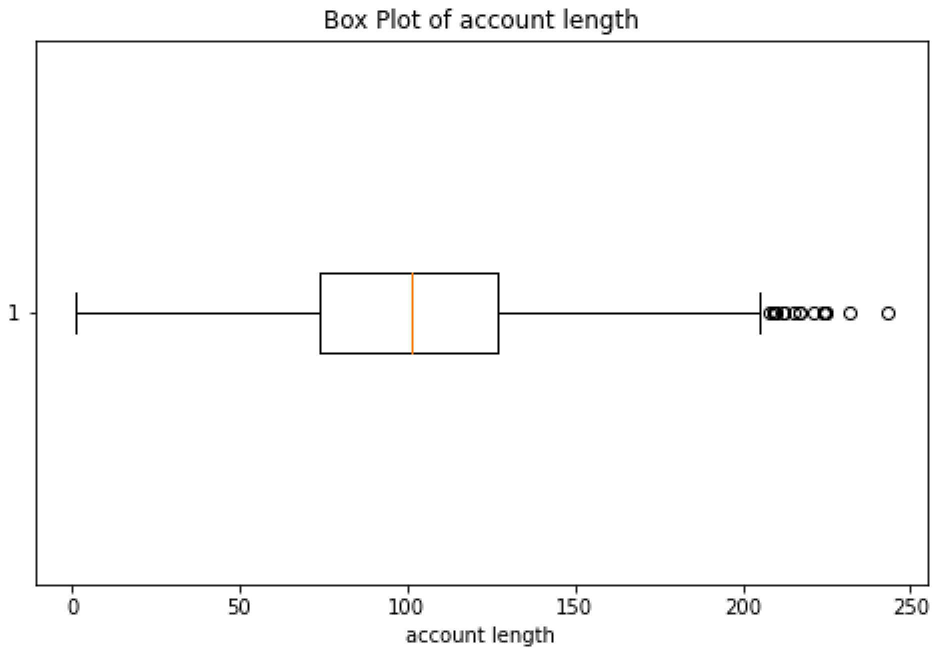

Histogram of total eve calls
Frequency
0
200
400
600
800
1000
1200
1400
0
25
50
75
100
125
150
175
total eve calls

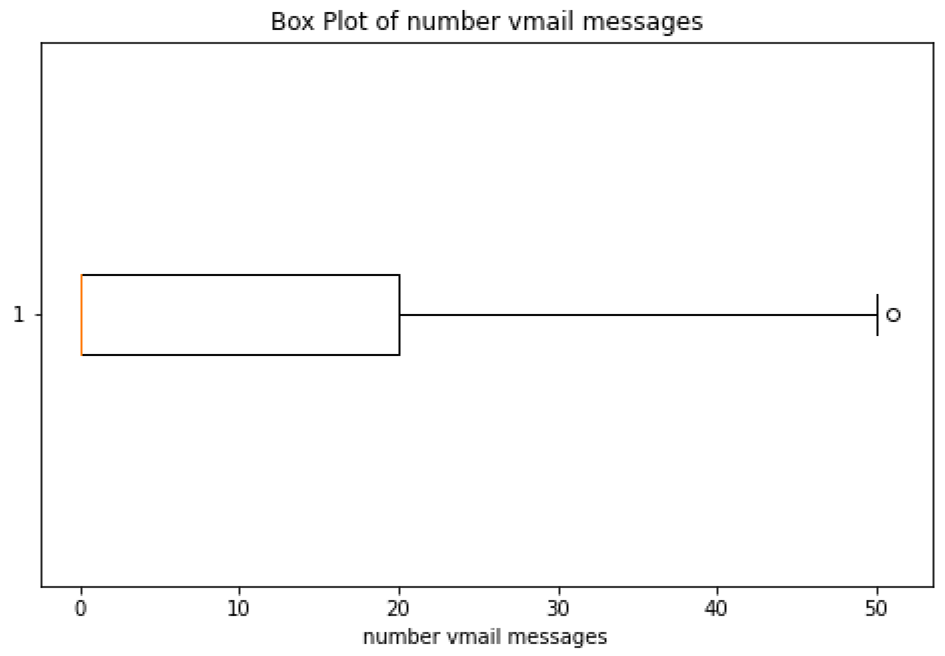

Histogram of total night calls
Frequency
0
200
400
600
800
1000
40
60
80
100
120
140
160
180
total night calls

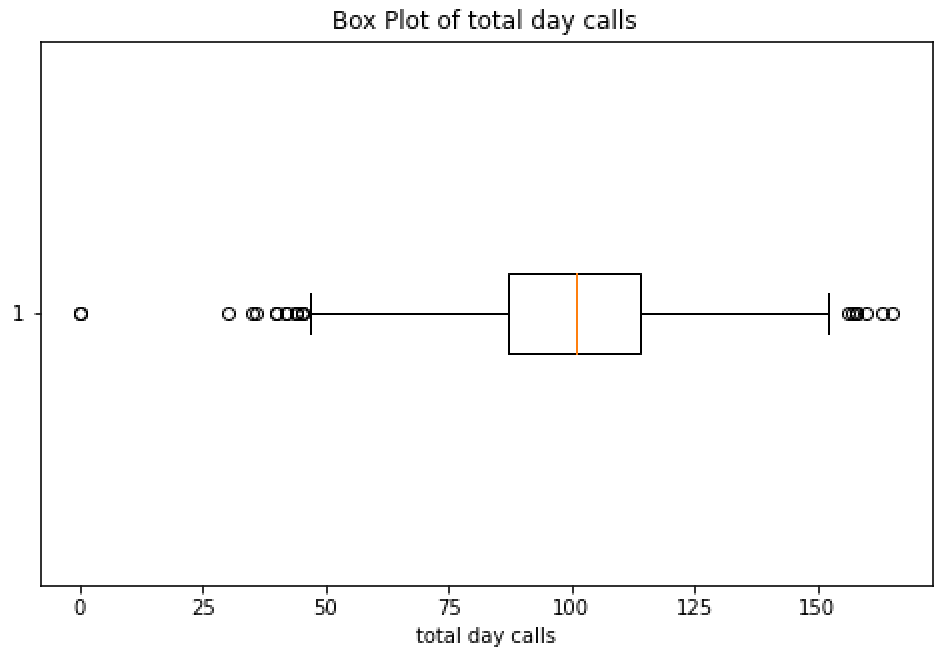

Histogram of total intl calls
Frequency
0
200
400
600
800
1000
1200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
total intl calls

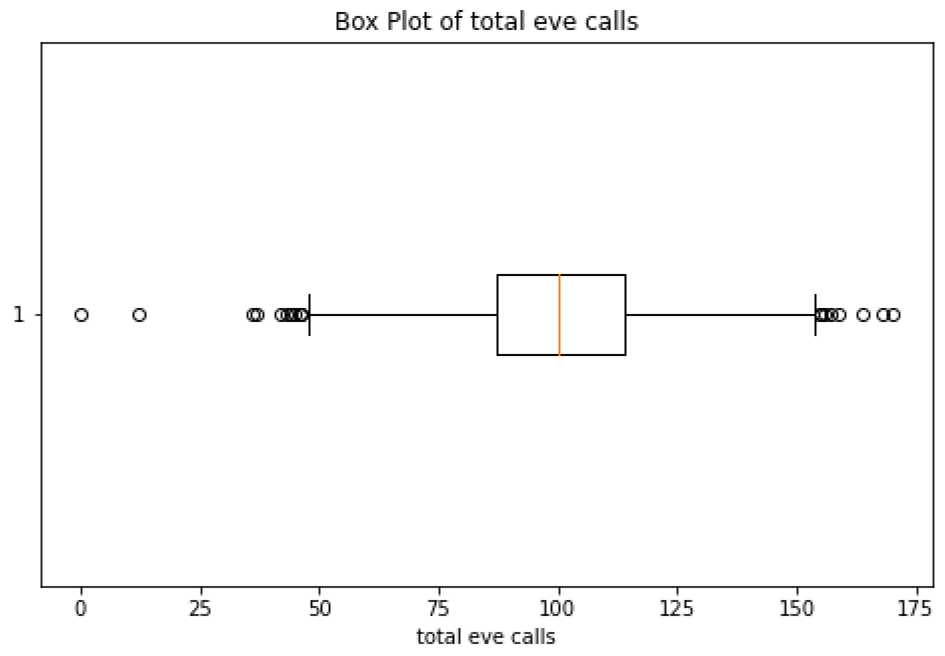


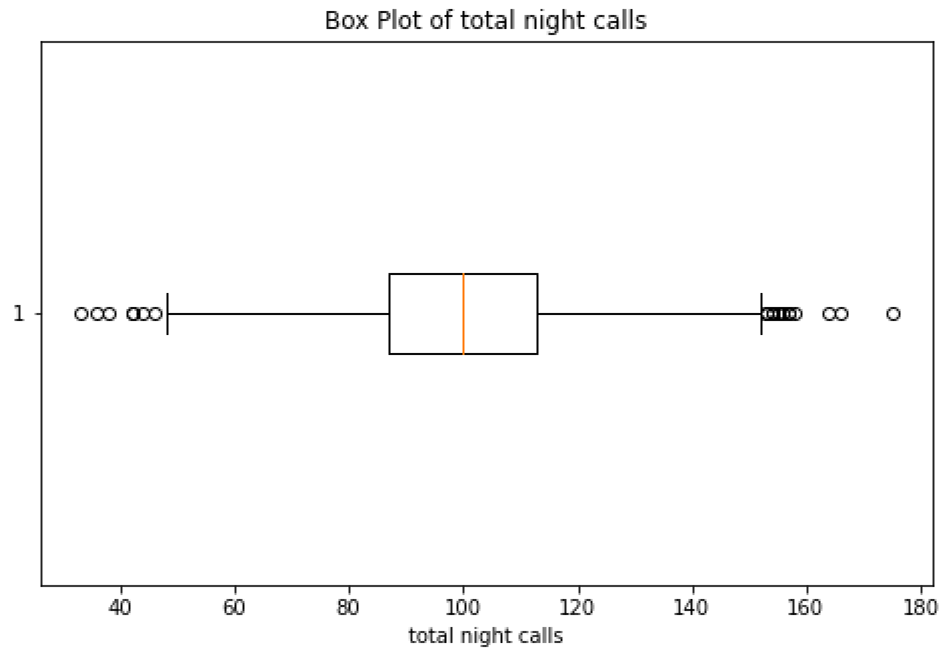
In [70]:
```
# Box plots to identify outliers and visualize the spread of data.
numerical_features = df.select_dtypes('int64', 'float64')

for feature in numerical_features:
    plt.figure(figsize=(8, 5))  # Set the figure size
    plt.boxplot(df[feature], vert=False)  # Create a horizontal box plot
    plt.title(f'Box Plot of {feature}')
    plt.xlabel(feature)
    plt.show()
```

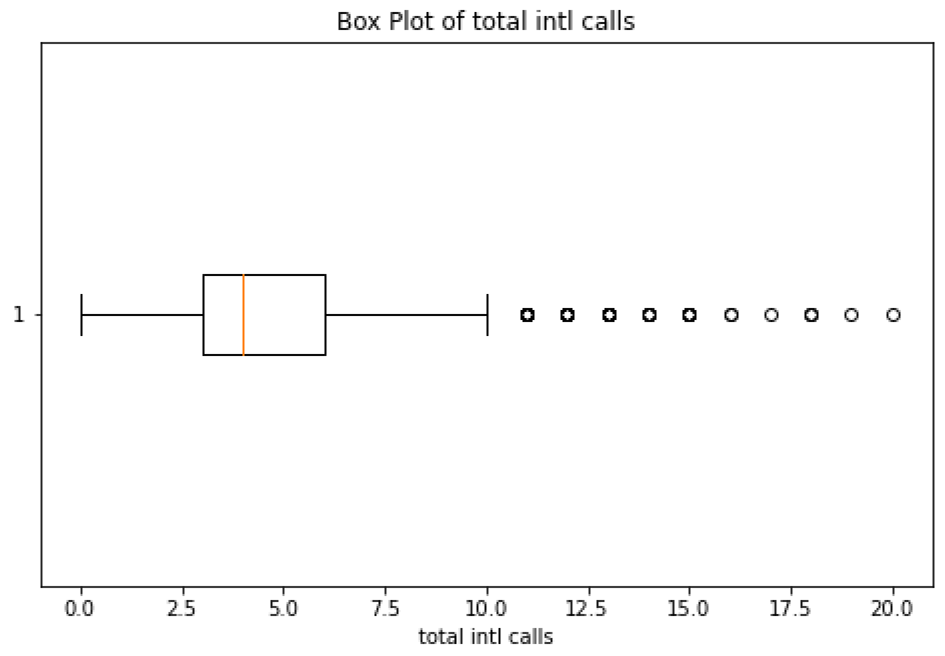


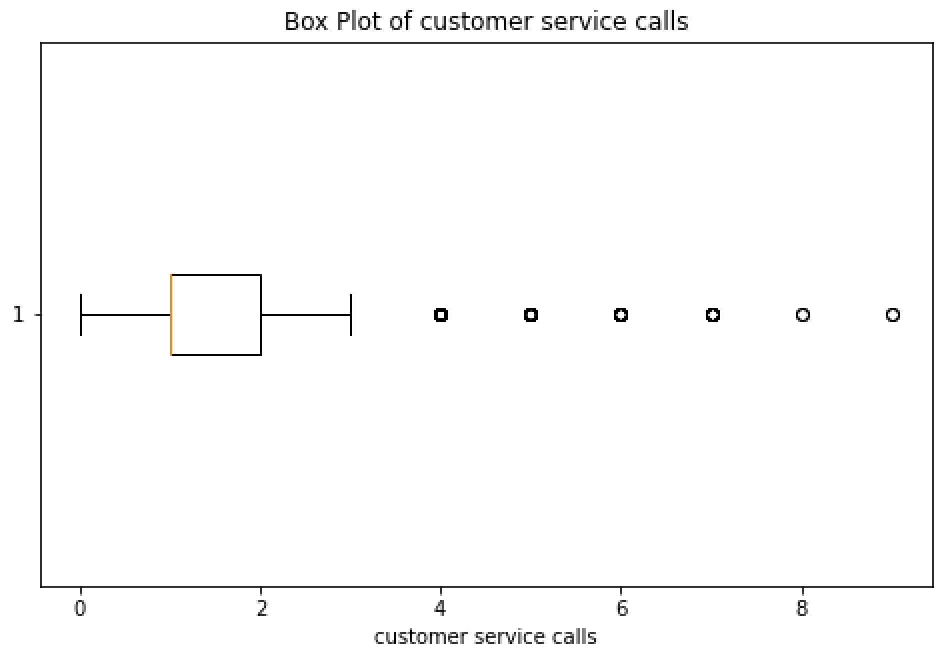

Box Plot of number vmail messages
1
0
10
20
30
40
50
number vmail messages


Box Plot of total day calls
1
0
25
50
75
100
125
150
total day calls


Box Plot of total eve calls
1
0
25
50
75
100
125
150
175
total eve calls


Box Plot of total night calls
1
40
60
80
100
120
140
160
180
total night calls



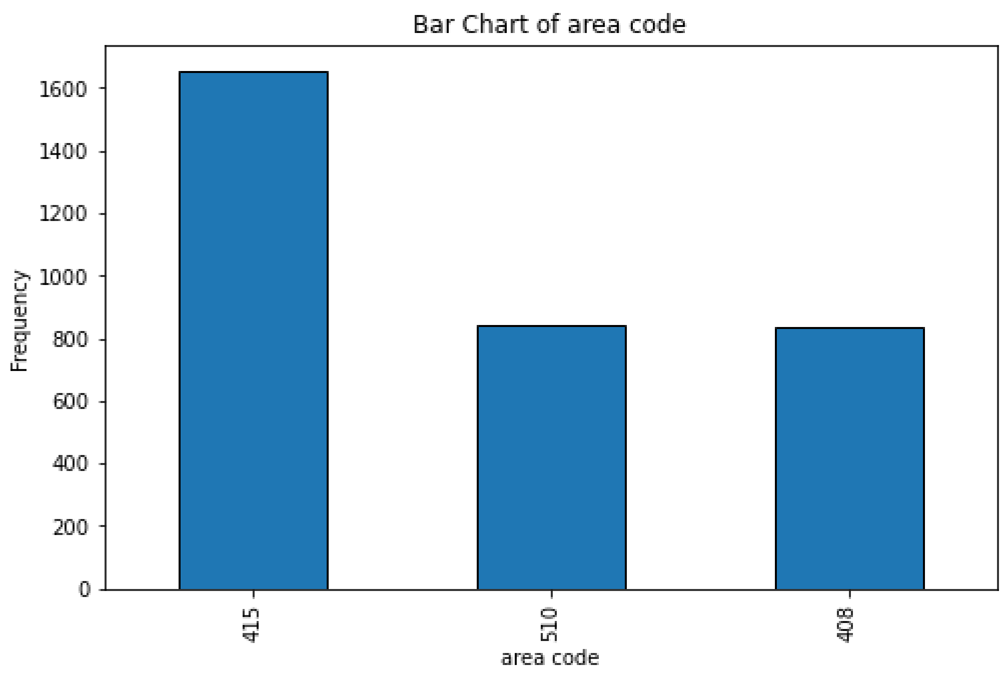


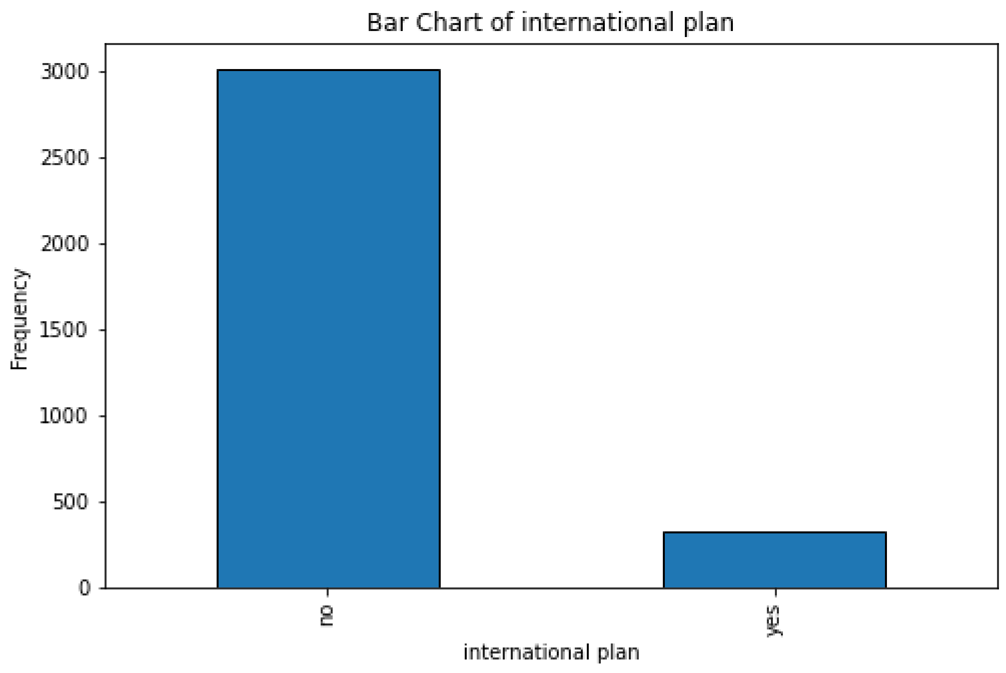
Outliers

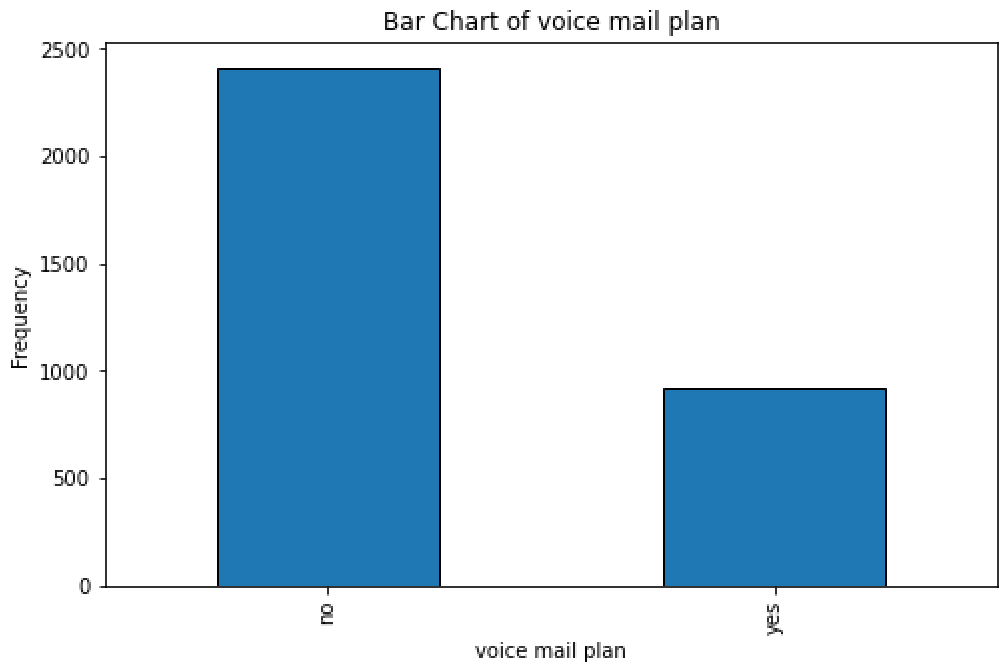
We have observed the presence of a significant number of outliers in our dataset, as indicated by the boxplots which have the potential to impact our modeling process. However, it is important to note that, in this case, these outliers are not anomalies that should be removed. Instead, they are a noteworthy aspect of our dataset that we should be aware of during our modeling process. These outliers may carry valuable information or insights that could be relevant to our analysis, and as such, it is essential to consider and account for them when developing our models and interpreting the results. Understanding the nature and impact of these outliers is a critical part of ensuring the robustness and accuracy of our data analysis

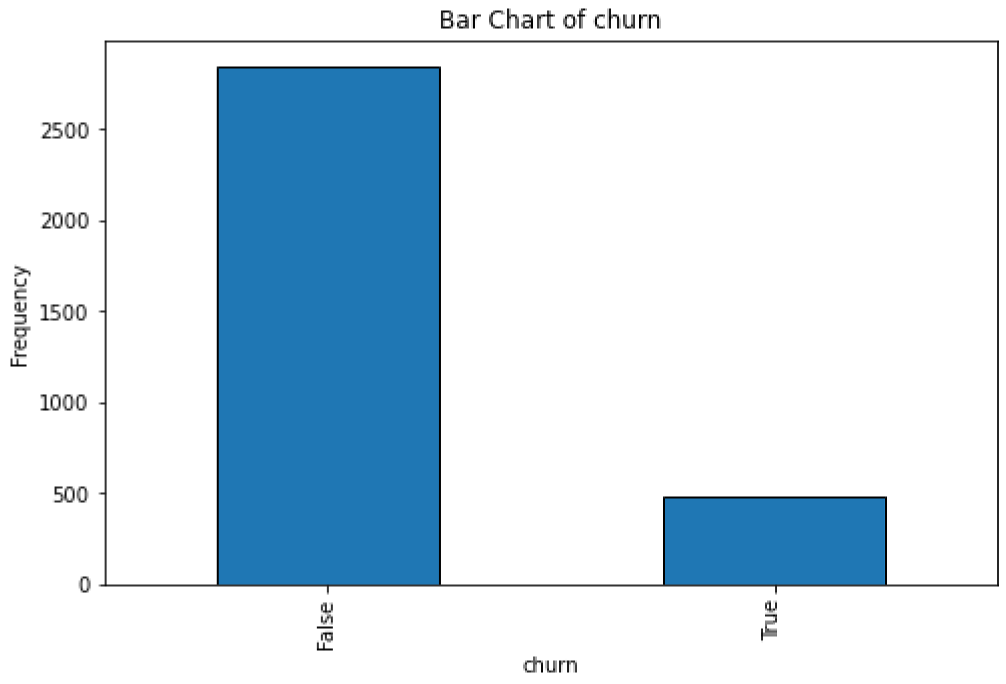
In [71]:
```
# Bar charts for categorical features to show the frequency of each category.
categorical_features = ['area code','international plan', 'voice mail plan','churn']
```

```
for feature in categorical_features:
    plt.figure(figsize=(8, 5))  # Set the figure size
    df[feature].value_counts().plot(kind='bar', edgecolor='k')  # Create the bar chart
    plt.title(f'Bar Chart of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()
```









3. Data Preparation

step 1: Removing irrelevant columns

as noted above our dataset does not contain any missing values and duplicates, thus we will only proceesd and remove the columns that are not essential for our modeling. this step helps in ;

- *Dimensionality Reduction*: By eliminating unnecessary columns, we can reduce the dimensionality of our data. This simplification can lead to faster model training and improved model interpretability.

- *Efficiency*: Removing non-essential columns enhances the efficiency of data processing and model training. It reduces computational demands, making our workflow more efficient.
- *Model Performance*: Unnecessary columns can introduce noise into our analysis and modeling, potentially leading to less accurate results. Eliminating such columns can result in a cleaner and more focused dataset.

From our heatmap above we noted that Total day minutes & total day charge,Total eve minutes & total eve charge,Total night minutes & total night charge,Total intl charge & tolat int charge all have correlation 1 hence we can drop one in each of these

The following columns were dropped because of high Multicolinearity Between each other.

- total day minutes
- total eve minutes
- total night minutes
- total intl minutes

- the phone number was also dropped as it serves as a unique identifier for customers but is unlikely to be a predictive feature for churn

In [72]:
```
# Create a new dataframe with the updated columns
df_subset = df.drop(columns= ['total day minutes', 'total eve minutes', 'total night ca
df_subset
```

Out[72]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night minutes	tot nigl charg
0	KS	128	415	no	yes	25	110	45.07	99	16.78	244.7	11.(
1	OH	107	415	no	yes	26	123	27.47	103	16.62	254.4	11.4
2	NJ	137	415	no	no	0	114	41.38	110	10.30	162.6	7.3
3	OH	84	408	yes	no	0	71	50.90	88	5.26	196.9	8.8
4	OK	75	415	yes	no	0	113	28.34	122	12.61	186.9	8.4
...	...	...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	415	no	yes	36	77	26.55	126	18.32	279.1	12.5
3329	WV	68	415	no	no	0	57	39.29	55	13.04	191.3	8.6
3330	RI	28	510	no	no	0	109	30.74	58	24.55	191.9	8.6
3331	CT	184	510	yes	no	0	105	36.35	84	13.57	139.2	6.2
3332	TN	74	415	no	yes	25	113	39.85	82	22.60	241.4	10.8

3333 rows × 16 columns

DATA PREPROCESSING

In [73]:
```
#one hot encoding categorical values
df_encoded = pd.get_dummies(df_subset)
df_encoded
```

Out[73]:

	account length	number vmail messages	total day calls	total day charge	total eve calls	total eve charge	total night minutes	total night charge	total intl calls	total intl charge	...	state_WI	sta
0	128	25	110	45.07	99	16.78	244.7	11.01	3	2.70	...	0	
1	107	26	123	27.47	103	16.62	254.4	11.45	3	3.70	...	0	
2	137	0	114	41.38	110	10.30	162.6	7.32	5	3.29	...	0	
3	84	0	71	50.90	88	5.26	196.9	8.86	7	1.78	...	0	
4	75	0	113	28.34	122	12.61	186.9	8.41	3	2.73	...	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
3328	192	36	77	26.55	126	18.32	279.1	12.56	6	2.67	...	0	
3329	68	0	57	39.29	55	13.04	191.3	8.61	4	2.59	...	0	
3330	28	0	109	30.74	58	24.55	191.9	8.64	6	3.81	...	0	
3331	184	0	105	36.35	84	13.57	139.2	6.26	10	1.35	...	0	
3332	74	25	113	39.85	82	22.60	241.4	10.86	4	3.70	...	0	

3333 rows × 70 columns

In [74]:
```
# Spliting data into Predictor and Target Variables
y = df_encoded.churn   # Target Variable
X = df_encoded.drop(['churn'], axis=1)   # Predictor Variables
```

Scale the Data

This is to ensure that the features are on a common scale.

In [75]:
```
#scale the data
scaler = StandardScaler()

# fit the scaler to the data and transform the data
X_scaled = pd.DataFrame(scaler.fit_transform(X))

X_scaled.head()
```

Out[75]:

	0	1	2	3	4	5	6	7	8	
0	0.676489	1.234883	0.476643	1.567036	-0.055940	-0.070427	0.866743	0.866029	-0.601195	-0.08
1	0.149065	1.307948	1.124503	-0.334013	0.144867	-0.107549	1.058571	1.059390	-0.601195	1.24
2	0.902529	-0.591760	0.675985	1.168464	0.496279	-1.573900	-0.756869	-0.755571	0.211534	0.69
3	-0.428590	-0.591760	-1.466936	2.196759	-0.608159	-2.743268	-0.078551	-0.078806	1.024263	-1.30
4	-0.654629	-0.591760	0.626149	-0.240041	1.098699	-1.037939	-0.276311	-0.276562	-0.601195	-0.04

5 rows × 69 columns

In [76]:
```
# creating training and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=4
```

In [77]:
```
#Using SMOTE to deal with class imbalance
# Previous original class distribution
print(y_train.value_counts())
smote = SMOTE(sampling_strategy='auto', random_state=42)
# Fit SMOTE to training data
X_train_resampled, y_train_resampled = SMOTE().fit_resample(X_train, y_train)

# Preview synthetic sample class distribution
print('\n')
print(pd.Series(y_train_resampled).value_counts())

# X_train_resampled and y_train_resampled contain the oversampled data

# You can use X_train_resampled and y_train_resampled to train your model
# Then, test it on the original testing data (X_test, y_test) for evaluation
```

```
False    2284
True      382
Name: churn, dtype: int64


True     2284
False    2284
Name: churn, dtype: int64
```

MODELING

Model 1: Logistic Regression (basic model)

In [78]:
```
#create an instance
lr = LogisticRegression()

# fit the model
lr.fit(X_train_resampled, y_train_resampled)

#predict the model
y_hat_train = lr.predict(X_train_resampled)
y_hat_test=lr.predict(X_test)
#testing the perfomance of the model using different metrics
print('Training Precision: ', precision_score(y_train_resampled, y_hat_train))
print('Testing Precision: ', precision_score(y_test, y_hat_test))
print('\n\n')

print('Training Recall: ', recall_score(y_train_resampled, y_hat_train))
print('Testing Recall: ', recall_score(y_test, y_hat_test))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train_resampled, y_hat_train))
print('Testing Accuracy: ', accuracy_score(y_test, y_hat_test))
print('\n\n')
print('Training F1-Score: ',  f1_score(y_train_resampled, y_hat_train))
print('Testing F1-Score: ',  f1_score(y_test, y_hat_test))
```

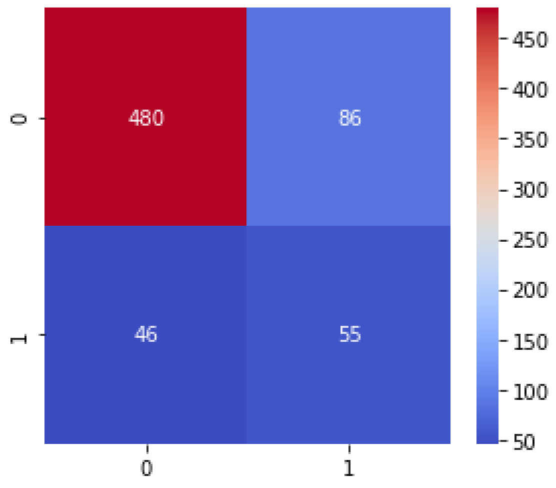
```
Training Precision:  0.849013308857274
Testing Precision:  0.3900709219858156



Training Recall:  0.8099824868651488
Testing Recall:  0.5445544554455446



Training Accuracy:  0.832968476357268
Testing Accuracy:  0.8020989505247377



Training F1-Score:  0.8290387631637911
Testing F1-Score:  0.45454545454545453
```

In [79]:

```
# Create a heatmap of the confusion matrix
plt.figure(figsize=(10,4))
plt.subplot(1,2,2)
cm = confusion_matrix(y_test, y_hat_test)
sns.heatmap(cm, annot=True, cmap='coolwarm', fmt='d')
```

Out[79]:

```
<AxesSubplot:>
```



The confusion matrix provides a comprehensive evaluation of the model's performance by categorizing its predictions into four types:

True Negatives (TN): There are 480 instances where the model correctly predicted "not churn" (0), and the actual value was also "not churn" (0).

False Positives (FP): In 86 instances, the model predicted "churn" (1), but the actual value was "not churn" (0).

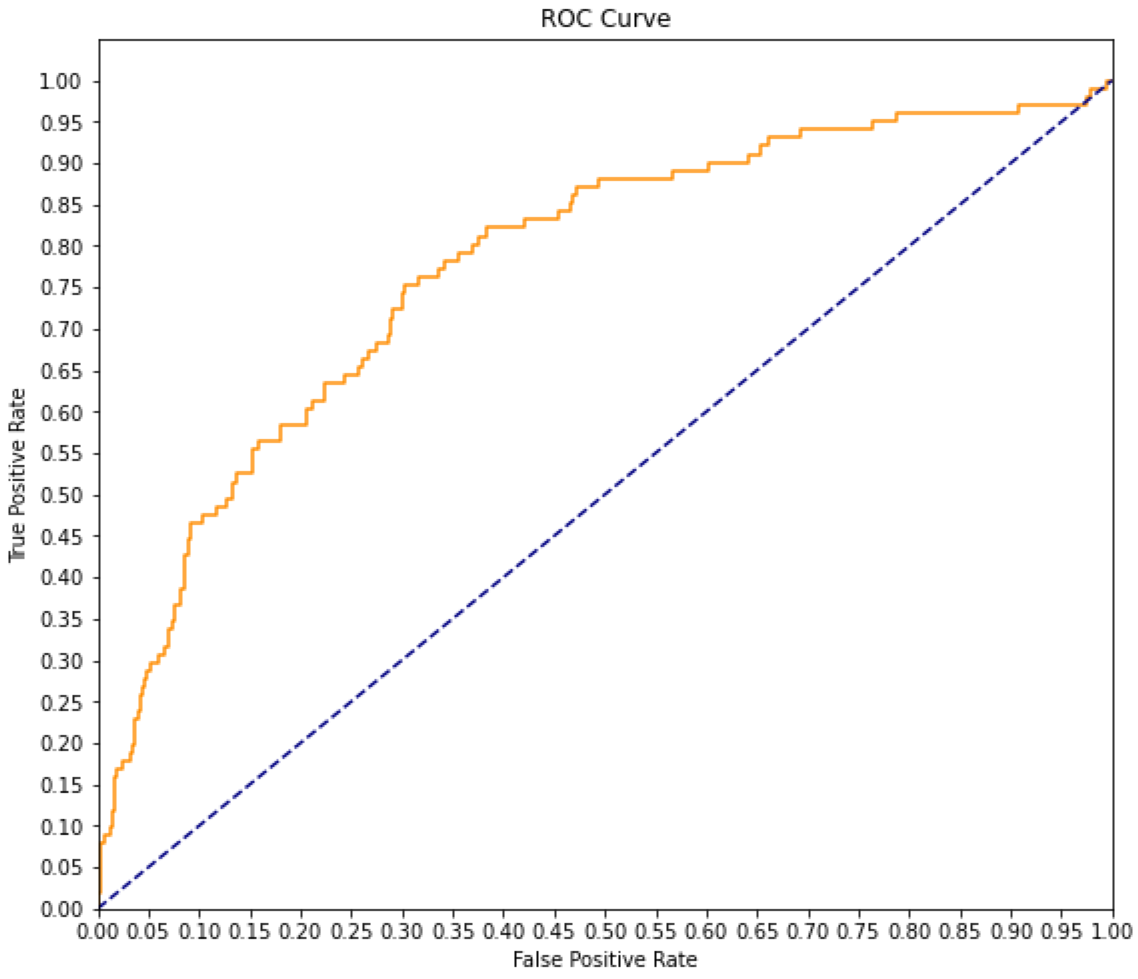
False Negatives (FN): The model incorrectly predicted "not churn" (0) for 46 instances, while the actual value was "churn" (1).

True Positives (TP): The model correctly predicted "churn" (1) for 55 instances, and the actual value was also "churn" (1).

The confusion matrix allows us to assess the model's performance in terms of different types of errors it makes. In this case, there is a notable number of false negatives (86), indicating that the model struggles to accurately identify instances that are actually churned.

In [80]:
```
# Calculate the ROC curve
y_pred_prob = lr.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)
# Calculate the area under the ROC curve
auc = roc_auc_score(y_test, y_pred_prob)

# Plot the ROC curve
plt.figure(figsize=(20,8))
plt.subplot(1,2,1)
plt.plot(fpr, tpr,color='darkorange')
plt.plot([0, 1], [0, 1], color='navy', 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")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")

#print AUC
print('The AUC score is:', auc )
```

```
The AUC score is: 0.775565895812196
```



Interpratation;

- .Precision : precision is the ratio of correctly predicted positive observations to the total predicted positive observations.
    - Training Precision: 84% - Out of all instances predicted as positive in the training set, 84% were actually positive.
    - Testing Precision: 38% - Out of all instances predicted as positive in the testing set, only 38% were actually positive.
- .Recall: Recall is the ratio of correctly predicted positive observations to all actual positives.
    - .Training Recall: 80% - The model identified 80.30% of all actual positives in the training set.
    - .Testing Recall: 54% - The model identified 54% of all actual positives in the testing set.
- Accuracy: Accuracy measures the overall correctness of predictions.
    - .Training Accuracy: 83% - The model correctly predicted 83% of instances in the training set.
    - .Testing Accuracy: 80% - The model correctly predicted 80% of instances in the testing set.
- .F1-Score: The F1-Score is the harmonic mean of precision and recall, providing a balanced assessment of a classifier's performance.
    - .Training F1-Score: 82% - The harmonic mean of precision and recall in the training set.

  - .Testing F1-Score: 45% - The harmonic mean of precision and recall in the testing set.
- .The AUC score of 0.78 signifies the model's ability to discriminate between churn and non-churn cases.

These metrics help gauge the model's performance. While the training performance appears better across most metrics (precision, recall, accuracy, and F1-score), there seems to be a notable drop in performance when applying the model to the testing set. This could indicate potential overfitting, where the model performs well on the training data but struggles to generalize to new, unseen data.

To further enhance the model's performance, additional refinements could be made to find a better balance between precision and recall, ultimately leading to improved customer retention and churn prediction capabilities.

Model 2: Decision Trees

In [88]:
```
# make an instance
Dt = DecisionTreeClassifier(criterion="entropy", random_state = 42)
#fit the model
Dt.fit(X_train_resampled, y_train_resampled)
#predict
Dt_y_pred_train =  Dt.predict(X_train_resampled)
Dt_y_pred_test = Dt.predict(X_test)
```

In [89]:
```
#testing the perfomance of the model using different metrics
print('Training Precision: ', precision_score(y_train_resampled , Dt_y_pred_train ))
print('Testing Precision: ', precision_score(y_test, Dt_y_pred_test ))
print('\n\n')

print('Training Recall: ', recall_score(y_train_resampled, Dt_y_pred_train))
print('Testing Recall: ', recall_score(y_test, Dt_y_pred_test ))
print('\n\n')

print('Training Accuracy: ', accuracy_score(y_train_resampled, Dt_y_pred_train))
print('Testing Accuracy: ', accuracy_score(y_test,Dt_y_pred_test ))
print('\n\n')

print('Training F1-Score: ',  f1_score(y_train_resampled, Dt_y_pred_train))
print('Testing F1-Score: ',  f1_score(y_test,Dt_y_pred_test ))
print('\n\n')

print('Testing Area Under the Curve: ',  roc_auc_score(y_test, Dt_y_pred_test))
```

```
Training Precision:  1.0
Testing Precision:  0.5658914728682171



Training Recall:  1.0
Testing Recall:  0.7227722772277227



Training Accuracy:  1.0
Testing Accuracy:  0.8740629685157422
```

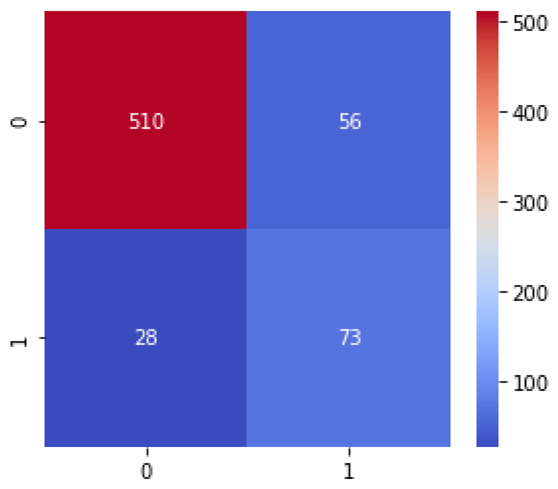
```
Training F1-Score:  1.0
Testing F1-Score:  0.6347826086956522



Testing Area Under the Curve:  0.8119161739495504
```

From the above model we obtained:

- Testing Precision: 56
- Testing Recall: 0.72
- Testing Accuracy: 0.87
- Testing F1-Score: 0.63
- Testing Area Under the Curve: 0.81

In [90]:
```
#confusion matrix
plt.figure(figsize=(10,4))
plt.subplot(1,2,2)
cm = confusion_matrix(y_test, Dt_y_pred_test)
sns.heatmap(cm, annot=True, cmap='coolwarm', fmt='d')
```

Out[90]:
```
<AxesSubplot:>
```



The confusion matrix reveals that the model correctly predicts 510 non-churned customers (true negatives) and 73 churned customers (true positives), but it incorrectly predicts 56 non-churned customers as churned (false positive) and misses 28 churned customers (false negatives)

Improve Results With Tuning

We want to optimize the above model hyperparameters to improve performance.

In [100…
```
from sklearn.model_selection import GridSearchCV

#Specify the hyperparameters and their respective values to search over
# Define the model
dt3 = DecisionTreeClassifier()
# Define the parameter grid to search through
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 5, 10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# scores
scores = ['f1', 'recall', 'precision','accuracy']
# Initialize the Grid Search using 5-fold cross-validation
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',  # Use appropriate scoring metric
                           cv=5,  # Choose cross-validation folds
                           n_jobs=-1)  # Use all available CPU cores
#. Fit the Grid Search to your data
grid_search.fit(X_train_resampled, y_train_resampled)

best_dt_model = grid_search.best_estimator_
best_dt_model.fit(X_train_resampled, y_train_resampled)

# Predict using the best model
Dt_y_pred_train = best_dt_model.predict(X_train_resampled)
Dt_y_pred_test = best_dt_model.predict(X_test)


# Print the best parameters
print("best_dt_model:", best_dt_model)
```

```
best_dt_model: DecisionTreeClassifier(criterion='entropy', max_depth=10, min_samples_lea
f=2,
                       min_samples_split=5, random_state=42)
```

We created a model using the best parameters which were:

- criterion='entropy',
- max_depth=15,
- min_samples_leaf=4,
- min_samples_split=10
- random_state =42

In [101…

```
#model with the optimal parameters
dt3_tuned = DecisionTreeClassifier(criterion='entropy',
                                   max_depth=15,
                                   min_samples_leaf=4,
                                   min_samples_split=10)

# fitting the model
dt3_tuned.fit(X_train_resampled, y_train_resampled)

# making predictions of the test data
dt3_y_pred = dt3_tuned.predict(X_test)


# evaluating of the model
dt3_f1_score = f1_score(y_test, dt3_y_pred)
dt3_acc_score = accuracy_score(y_test, dt3_y_pred)
dt3_prec_score = precision_score(y_test, dt3_y_pred)
dt3_rec_score = recall_score(y_test, dt3_y_pred)
dt3_auc_score= roc_auc_score(y_test, dt3_y_pred)

print(f' The F1 Score of the Test Data is {dt3_f1_score}')
print(f' The Accuracy Score of the Test Data is {dt3_acc_score}')



print(f' The Precision Score of the Test Data is {dt3_prec_score}')



print(f' The Recall Score of the Test Data is {dt3_rec_score}')

print(f' The AUC of the Test Data is {dt3_auc_score}')
```

```
The F1 Score of the Test Data is 0.7203791469194313
The Accuracy Score of the Test Data is 0.9115442278860569
The Precision Score of the Test Data is 0.6909090909090909
The Recall Score of the Test Data is 0.7524752475247525
The AUC of the Test Data is 0.8462022880733303
```

after Tuning the Decision Tree Classifier with the best hyperparameters we can see significant improvements in various evaluation metrics as below;

- The F1 Score increased from 0.63 to 73, indicating a better balance between precision and recall. The model is now more adept at correctly classifying positive instances (churn cases) while minimizing both false positives and false negatives.
- The Precision Score increased from 0.56 to 0.70, indicating a higher proportion of correct positive predictions. This means that the model is more accurate in identifying customers who are likely to churn.
- The Recall Score has slightly increased from 0.72 to 0.76, indicating a slight improvement in capturing actual churn cases. The model is now better at identifying customers at risk of churning, reducing missed opportunities for retention.
- The acuracy score has increased from 0.87 to 0.91 indicating an improvement of accuracy in overall prediction
- The AUC slightly increased from 0.83 to 0.85, showing enhanced discrimination between churn and non-churn cases.

In conclusion, the tuning of the hyperparameters has successfully improved the model's performance,

MODEL 3 - KNN

In [107…

```
 # K-Nearest Neighbors
knn = KNeighborsClassifier(n_neighbors=5)
#fit the model
knn.fit(X_train_resampled, y_train_resampled)
#predict
knn_y_pred = knn.predict(X_test)
#metrics
accuracy = accuracy_score(y_test, knn_y_pred)
f1_score = f1_score(y_test, knn_y_pred)
precision = precision_score(y_test, knn_y_pred)
```

```
recall = recall_score(y_test, knn_y_pred)
AUC = roc_auc_score(y_test, knn_y_pred)

print(f' The Recall Score is {recall}')

print(f' The Precision Score is {precision}')

print(f' The Accuracy Score is {accuracy}')

print(f' The F1 Score is {f1_score}')

print(f' The AUC Score is {AUC}')
```

```
 The Recall Score is 0.5148514851485149
 The Precision Score is 0.20392156862745098
 The Accuracy Score is 0.6221889055472264
 The F1 Score is 0.29213483146067415
 The AUC Score is 0.5780971206661303
```

Hyperparameter tuning for the KNN Model

In [108…

```
#hyperparameter tuning
# Define the parameter grid for GridSearchCV
param_grid = {
        'n_neighbors': [3, 5, 7, 9],  # List of different values for the number of neig
        'weights': ['uniform', 'distance'],  # Different weight options
        'metric': ['euclidean', 'manhattan']  # Different distance metrics
    }


# scores
scores = ['f1', 'recall', 'precision']

# Create a grid search object using 10-fold cross-validation
grid_search = GridSearchCV(knn, param_grid, cv=10, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train_resampled, y_train_resampled)

# Get the best parameters from the grid search
best_params = grid_search.best_params_

# Print the best parameters and the best score
print("Best Parameters:", best_params)
print("Best Scores:", grid_search.best_score_)
```

```
Best Parameters: {'metric': 'euclidean', 'n_neighbors': 3, 'weights': 'distance'}
Best Scores: 0.8412822949057543
```

We obtained the following as the best parameters:

- 'metric': 'euclidean'
- 'n_neighbors': 3
- 'weights': 'distance'

In [109…

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, co
# Train the random forest classifier with best parameters
knn_1= KNeighborsClassifier(metric = 'euclidean',
                            n_neighbors = 3,
                            weights = 'distance')
```

```
knn_1.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test data
knn_1_y_pred = knn_1.predict(X_test)

# Evaluate the model's accuracy

knn_1_prec_score = precision_score(y_test, knn_1_y_pred)

knn_1_rec_score = recall_score(y_test, knn_1_y_pred)

knn_1_acc_score = accuracy_score(y_test,knn_1_y_pred)

knn_1_f1_score = f1_score(y_test, knn_1_y_pred)

knn_1_auc_score = roc_auc_score(y_test, knn_1_y_pred)



print(f' The Recall Score is {knn_1_rec_score}')

print(f' The Precision Score is {knn_1_prec_score}')

print(f' The Accuracy Score is {knn_1_acc_score}')

print(f' The F1 Score is {knn_1_f1_score}')

print(f' The AUC Score is {knn_1_auc_score}')
```

```
 The Recall Score is 0.40594059405940597
 The Precision Score is 0.2
 The Accuracy Score is 0.664167916041979
 The F1 Score is 0.2679738562091503
 The AUC Score is 0.5580939719413637
```

interpirataion of the KNN Model

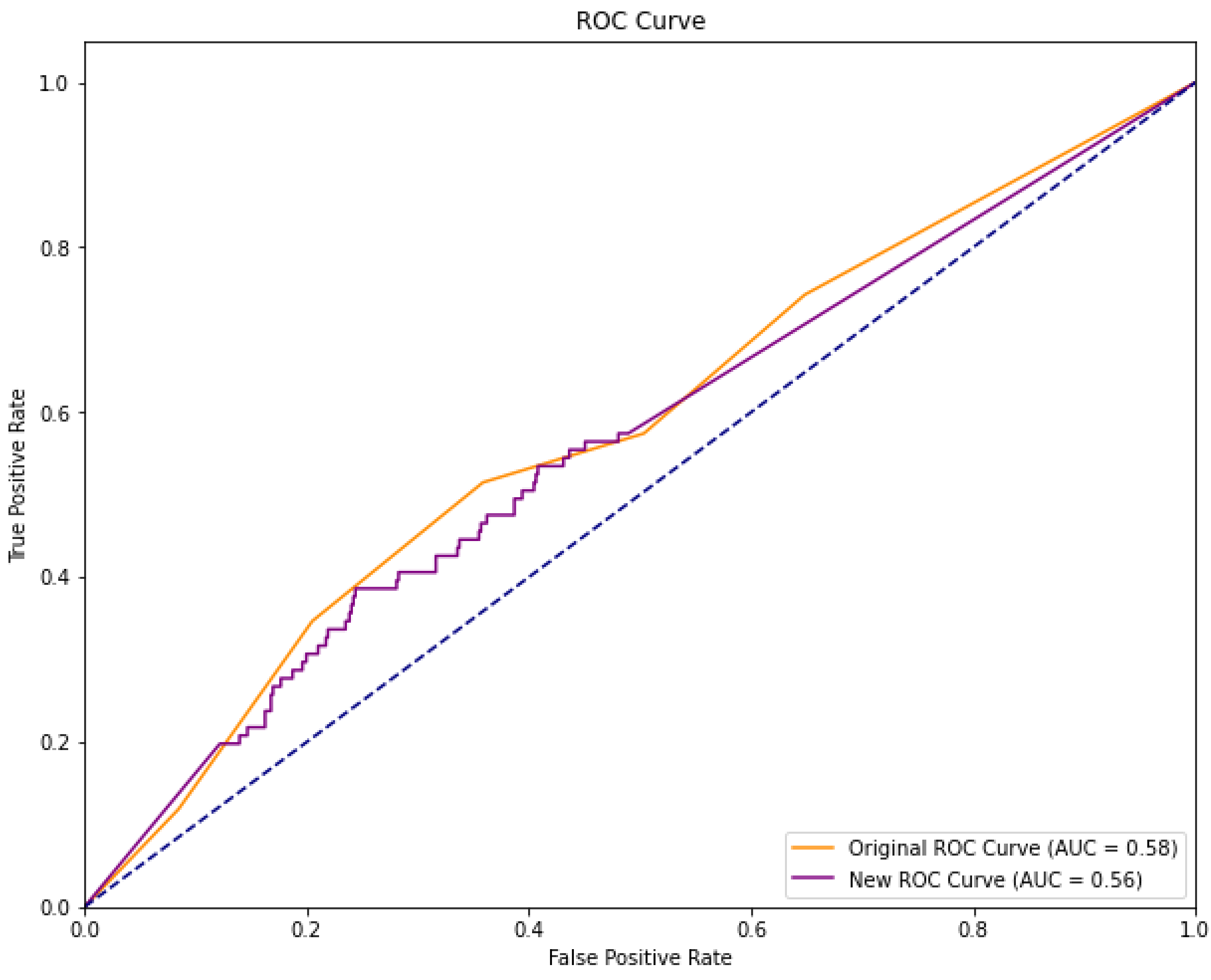
- Both models have relatively low recall scores, but the tuned model shows a slight decrease in recall compared to the baseline.
- The precision scores for both models are low.
- The accuracy of the tuned model is slightly better than the baseline.
- The F1 scores are amost similar for both models, with a slight decrease in the tuned model.
- The AUC scores are similar, with a slightly higher value for the tuned model.

The main concern is the low recall, indicating that both models struggle to capture a significant portion of actual churn cases.Prioritizing recall is essential to effectively identify churned customers and optimize retention strategies for better customer retention.

In [110…

```
# Calculate the ROC curve
y_pred_knn = knn.predict_proba(X_test)[:, 1]
y_pred_knn1 =knn_1.predict_proba(X_test)[:, 1]

# Compute the ROC curve
```

```
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_knn)
fpr_knn1, tpr_knn1, thresholds_knn1 = roc_curve(y_test, y_pred_knn1)

# Calculate the area under the ROC curve
auc_knn = roc_auc_score(y_test, y_pred_knn)
auc_knn1 = roc_auc_score(y_test, y_pred_knn1)


# Plot the ROC curves
plt.figure(figsize=(10, 8))
plt.plot(fpr_knn, tpr_knn, color='darkorange', label='Original ROC Curve (AUC = {:.2f})
plt.plot(fpr_knn1, tpr_knn1, color='purple', label='New ROC Curve (AUC = {:.2f})'.forma

plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc='lower right');
```

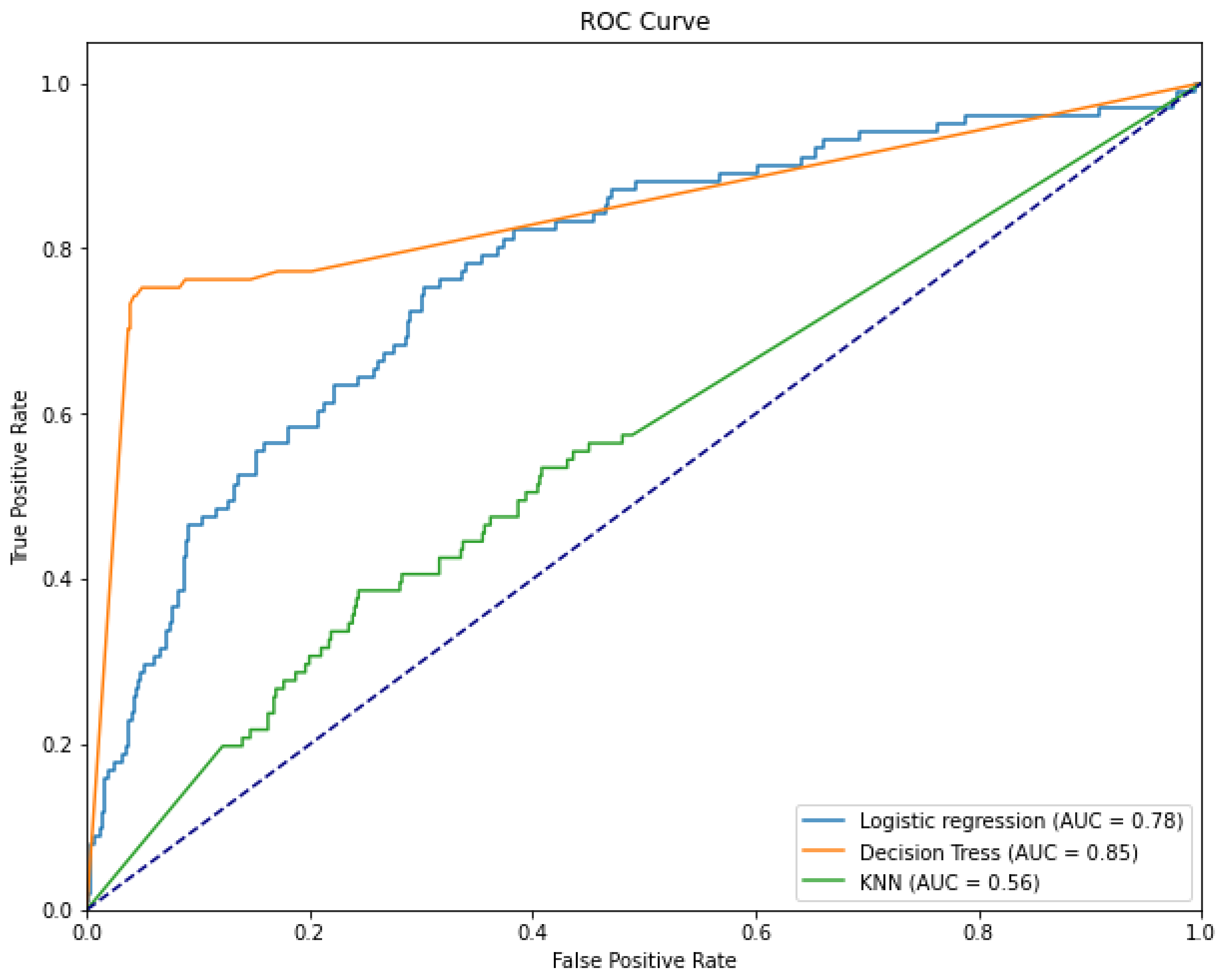


Model Evaluation and Selection

We will determine the best model using the ROC curve to compare the three computed models above

In [112…

```
# Calculate the ROC curve
y_pred_lr = lr.predict_proba(X_test)[:, 1]
y_pred_Dt = dt3_tuned.predict_proba(X_test)[:, 1]
y_pred_knn =knn_1.predict_proba(X_test)[:, 1]

# Compute the ROC curve
fpr_lr, tpr_lr, thresholds_lr = roc_curve(y_test, y_pred_lr)
fpr_Dt, tpr_Dt, thresholds_Dt = roc_curve(y_test, y_pred_Dt )
fpr_knn, tpr_knn, thresholds_knn = roc_curve(y_test, y_pred_knn)

# Calculate the area under the ROC curve
auc_lr = roc_auc_score(y_test, y_pred_lr)
auc_Dt = roc_auc_score(y_test, y_pred_Dt )
auc_knn = roc_auc_score(y_test, y_pred_knn)

# Plot the ROC curves
plt.figure(figsize=(10, 8))
plt.plot(fpr_lr, tpr_lr,  label='Logistic regression (AUC = {:.2f})'.format(auc_lr))
plt.plot(fpr_Dt, tpr_Dt,  label='Decision Tress (AUC = {:.2f})'.format(auc_Dt))
plt.plot(fpr_knn, tpr_knn,  label='KNN (AUC = {:.2f})'.format(auc_knn))


plt.plot([0, 1], [0, 1], color='navy', linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend(loc='lower right');
plt.show()
```



From above ROC curve we note that the decision model perfom way more better than the logistic regression and the KNN.

- The logistic regression model shows a balanced performance with reasonably good accuracy, precision, recall, and F1 score. It captures positive cases effectively while maintaining precision. The ROC AUC score is also decent.

- The decision tree model had best prefomance in most of the parameter with arecall of 75%, accuracy score of 91% precion score of 69% and F1 score of 72%.
- The KNN classifier model achieves decent accuracy and performance for the majority class but struggles with the minority class, similar to the decision tree model thus it may not be good for unseen data.

Model of Choice: Decision Trees

The decision tree was our best model because:

- the model achieved a high recall score of 0.75. Recall, also known as sensitivity or true positive rate, measures the proportion of actual churn cases correctly identified by the model. In our case, a recall score of 0.75 implies that the decision tree model can capture a significant number of churn cases, correctly identifying 75% of the actual instances of customers who are likely to churn. This high recall score is crucial because it aligns with our objective of identifying and mitigating churn effectively.

- the model demonstrated an accuracy score of 0.91. Accuracy represents the overall correctness of the model's predictions. This indicates that the model performs well in accurately predicting churn, which is essential for making informed business decisions.
- The model's substantial ability to balance precision and recall is reflected in its F1 score of 0.72. The F1 score combines precision and recall into a single metric, providing a balanced measure of a model's performance. A higher F1 score indicates that the model effectively identifies true positives while minimizing false positives and false negatives. In our case, the Decision tree model achieves an F1 score of 0.72, suggesting that it strikes a good balance between precision and recall, resulting in accurate identification of churn cases.

. Recommendations and Future Investigations

Customer Service Calls Investigation:

Dig deeper to understand why some customers need to contact customer service frequently and if there is a relationship between customers who call and their churn rate. This will help in finding ways to better assist them.

International Plan Churn Investigation:

Since some of the customers with international plans are leaving, it's essential to explore bundled plan for them in oreder to retain these customers.

High Churn States Analysis:

Look into the states where many customers are leaving to identify any patterns or reasons for the high churn rates.

Incentives for High Bill Customers:

Find ways to reward customers with high daily charges (over $55) for them to stay with SyriaTel. This might involve offering extra benefits and perks. Currently, all of these high bill customers are leaving, which is a concern.

Incentives for Customers who stay more than 6 months:

Find ways to encouragte customers to stay with the company even longer, eg giving them loyalty points offers etc, as this will help in creating a form of loyalty

Conclusion

It is important to monitor the effectiveness of these measures through ongoing analysis of churn rates, customer feedback, and market trends. Regularly evaluating and adapting these strategies based on customer needs and preferences will help optimize retention efforts and reduce churn effectively. Additionally, providing personalized offers, loyalty programs, and targeted marketing campaigns can further contribute to improving customer satisfaction and loyalty.