SyriaTel Customer Churn Prediction

Student Name - Leo Rono

Student Pace - DSF-FT12-Remote

Phase 3 Project

Introduction: Project Overview

In a highly competitive telecommunications market, understanding customer behavior is essential for business sustainability.This project focuses on performing customer churn analysis for SyriaTel, one of Syria's leading mobile network operators.

This project aims to develop a predictive model capable of identifying patterns associated with customer churn. The insights derived from this model will guide the development of effective customer retention strategies. The project follows a structured, end-to-end data science workflow, outlined as follows:

1. Business Understanding – Defining project goals in alignment with business objectives.
2. Data Understanding – Exploring and familiarizing with the dataset to uncover initial insights.
3. Data Preparation – Cleaning and transforming the data to ensure quality and suitability for modeling.
4. Exploratory Data Analysis (EDA) – Investigating key trends, correlations, and anomalies within the data.
5. Modeling – Building and training predictive models to classify churn behavior.
6. Model Evaluation – Assessing model performance using appropriate metrics to ensure reliability.
7. Recommendations and Conclusions – Translating analytical findings into actionable business strategies.

By identifying the key drivers behind customer attrition, the business can proactively design targeted interventions to enhance customer satisfaction and reduce churn rates.

Business understanding

Company Background

SyriaTel is a leading telecommunications provider in Syria, delivering essential services to millions of customers, including voice call services, SMS and messaging, GSM technology-based services, Internet and data connectivity as well as News and media service. Since its establishment in 2000, SyriaTel has played a significant role in the digital transformation of Syria's telecommunications sector; having about 3,500 employees and 8 million subscribers. In light of increasing market competition majorly from a new entrant, Wafa Telecom,and shifting customer preferences, SyriaTel's focus and evolution on customer retention has become more critical than ever.

This evolution is crucial to maintain its competitive edge and continue providing exceptional customer experiences in a rapidly changing market.

Key Stakeholders

1. Business and Marketing Executives - Responsible for customer engagement, promotional strategies, and market segmentation.
2. Senior Management Team - Understand the broader implications of churn on company revenue, customer lifetime value, and growth projections.
3. Customer Service Team - Directly involved with at-risk customers hence can benefit by personalizing their outreach and retention tactics.
4. Potential Investors and Partners - Investors and business partners have a vested interest in Syriatel's market performance and strategic direction. They are essential as they fund and drive SyriaTel's technological advancements

Problem statement

SyriaTel currently lacks the capability to identify customers who are on the verge of churning. The primary objective of this project is to develop a predictive model that can accurately forecast customer churn. Gaining this predictive insight will enable the organization to monitor churn risk on a continuous basis and respond swiftly with targeted incentives and retention strategies, ultimately improving customer loyalty and reducing revenue loss.

Objectives

1. Which customer features and behaviors are most strongly associated with churn?

2. Which machine learning model is best suited for accurately predicting customer churn?

3. What strategies can Syriatel implement to effectively retain customers and minimize churn?

Data Understanding

In [1]:
```
# Import modules & packages

import pandas as pd
import numpy as np

import seaborn as sns
import matplotlib.pyplot as plt
import plotly.graph_objs as go
import plotly.express as px

from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from imblearn.over_sampling import SMOTE,SMOTENC
from sklearn.metrics import f1_score,recall_score,precision_score,confusion_matrix,roc_curve,roc_auc_score,classificatior
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
import xgboost as xgb
from xgboost import XGBClassifier

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Load and Preview the Dataset
df = pd.read_csv("data/bigml_59c28831336c6604c800002a.csv")

df.head()
```

Out[2]:

	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	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	t
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	

5 rows × 21 columns

In [3]:
```
#Shape of the dataframe
df.shape
```

Out[3]: (3333, 21)

The dataset contains 3333 entries(rows) and 21 columns

In [4]:
```
# General statistics of numeric columns
df.describe()
```

Out[4]:

	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
count	3333.000000	3333.000000	3333.000000	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	100.114311	17.083540	200.872037	100.1
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847	19.5
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000	33.(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000	87.(

	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
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000	100.(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000	113.(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000	175.(

This is done to check for any anomalies in the data. We will consider it during the data cleaning

Lets get a general overview of our data

In [5]:
```
#General overview of the DataFrame
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
```

We have `3333 data` records and `21 columns`, with zero null values. We will still need to review each column further to identify anomalies. `Four` of our columns are of the object type, while `eight` are of integer type, `eight` as floats, and `one` column as bolean. Our target variable column is churn, which means we will treat the rest of the columns as features.

Lets see which columns are numerical and which are categorical

In [6]:
```
# Numerical Columns
print(f"Numerical Columns: {df.select_dtypes(include='number').columns}\n")

# Categorical Columns
print(f"Categorical Columns: {df.select_dtypes(include='object').columns}")
```

```
Numerical Columns: Index(['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'],
      dtype='object')

Categorical Columns: Index(['state', 'phone number', 'international plan', 'voice mail plan'], dtype='object')
```

Categorical Features:

`state` : The state where the customer resides.

`phone number` : The phone number of the customer.

`international plan` : Whether the customer has an international plan (Yes or No).

`voice mail plan` : Whether the customer has a voice mail plan (Yes or No).

Numeric Features:

`area code` : The area code associated with the customer's phone number.

`account length` : The number of days the customer has been an account holder.

`number vmail messages` : The number of voice mail messages received by the customer.

`total day minutes` : The total number of minutes the customer used during the day.

`total day calls` : The total number of calls made by the customer during the day.

`total day charge` : The total charges incurred by the customer for daytime usage.

`total eve minutes` : The total number of minutes the customer used during the evening.

`total eve calls` : The total number of calls made by the customer during the evening.

`total eve charge` : The total charges incurred by the customer for evening usage.

`total night minutes` : The total number of minutes the customer used during the night.

`total night calls` : The total number of calls made by the customer during the night.

`total night charge` : The total charges incurred by the customer for nighttime usage.

`total intl minutes` : The total number of international minutes used by the customer.

`total intl calls` : The total number of international calls made by the customer.

`total intl charge` : The total charges incurred by the customer for international usage.

`customer service calls` : The number of customer service calls made by the customer.

Let's preview the top 10 and top bottom 10 data records to get a small understanding of what we are dealing with.

In [7]:
```
# Previewing First 10 rows
df.head(10)
```

Out[7]:

	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	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	t
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10.0	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13.7	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12.2	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6.6	

	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	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	t
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10.1	
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.98	...	101	18.75	203.9	118	9.18	6.3	
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.09	...	108	29.62	212.6	118	9.57	7.5	
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.69	...	94	8.76	211.8	96	9.53	7.1	
8	LA	117	408	335-4719	no	no	0	184.5	97	31.37	...	80	29.89	215.8	90	9.71	8.7	
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.96	...	111	18.87	326.4	97	14.69	11.2	

10 rows × 21 columns

In [8]:
```
# Previewing the Last 10 rows
df.tail(10)
```

Out[8]:

	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	total eve charge	total night minutes	total night calls	total night charge	tota int minute
3323	IN	117	415	362-5899	no	no	0	118.4	126	20.13	...	97	21.19	227.0	56	10.22	13.(
3324	WV	159	415	377-1164	no	no	0	169.8	114	28.87	...	105	16.80	193.7	82	8.72	11.(
3325	OH	78	408	368-8555	no	no	0	193.4	99	32.88	...	88	9.94	243.3	109	10.95	9.:
3326	OH	96	415	347-6812	no	no	0	106.6	128	18.12	...	87	24.21	178.9	92	8.05	14.(
3327	SC	79	415	348-3830	no	no	0	134.7	98	22.90	...	68	16.12	221.4	128	9.96	11.(

	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	total eve charge	total night minutes	total night calls	total night charge	tota int minute
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83	12.56	9.9
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123	8.61	9.(
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91	8.64	14.1
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137	6.26	5.(
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77	10.86	13.7

10 rows × 21 columns

Most of the columns have 2 or more words as the columns names. We need to remove the whitespaces so as to make the column names easily addressible by replacing them with underscore '_'

In [9]:
```
# Removing whitespaces in the column name and replacing with '_'
df.columns = df.columns.str.replace(' ', '_')
```

In [10]:
```
# Confirmation of replacement of whitespaces with '_'
df.head(10)
```

Out[10]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_ca
0	KS	128	415	382-4657	no	yes	25	265.1	1
1	OH	107	415	371-7191	no	yes	26	161.6	1
2	NJ	137	415	358-1921	no	no	0	243.4	1
3	OH	84	408	375-9999	yes	no	0	299.4	
4	OK	75	415	330-6626	yes	no	0	166.7	1
5	AL	118	510	391-8027	yes	no	0	223.4	
6	MA	121	510	355-9993	no	yes	24	218.2	

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_ca
7	MO	147	415	329-9001	yes	no	0	157.0	
8	LA	117	408	335-4719	no	no	0	184.5	
9	WV	141	415	330-8173	yes	yes	37	258.6	

10 rows × 21 columns

Data Preparation

Here we will perform data cleaning, exploratory data analysis (EDA), and data preprocessing.

Data Cleaning

In this section, we will be looking at the missing values in the dataset as well as the duplicate records and removing irrelevant features

Check missing values

In [11]:
```
df.isna().sum()
```

Out[11]:
```
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
```

There are no null values across all the columns

Check duplicates

In [12]:
```
df.duplicated().sum()
```

Out[12]:
```
0
```

There are no duplicated values

As the 'area_code' represent geographic regions and not mathematical quantities, let's convert them to categorical features

In [13]:
```
#Converting 'area_code' to a categorical feature
df["area_code"] = df["area_code"].astype(object)
```

Let's also drop the 'phone_number' column as it would cause overfitting later on and is not relevant

In [14]:
```
#Dropping 'phone_number' column
df = df.drop("phone_number", axis=1)
```

Exploratory Data Analysis

In this section, we will explore and analyze the dataset to uncover patterns, understand relationships between variables(univatiate and bivariate analysis), detect outliers, and gain meaningful insights before applying any modeling or statistical methods.

Univariate Analysis

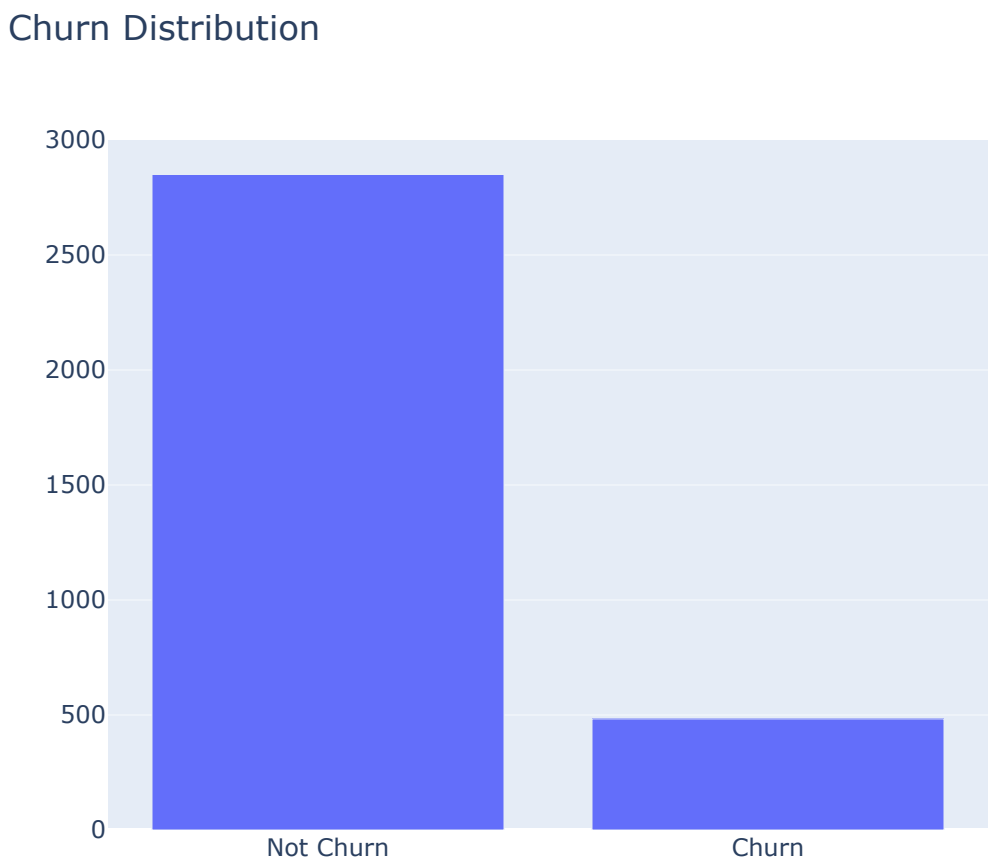
In this section, we'll examine each feature in the dataset to understand its distribution, central tendency, and variability, while also identifying potential outliers and revealing any underlying patterns.

Let's begin by examining the "churn" feature.

Distribution of "Churn" Feature

In [15]:
```
#Plot target variable distribution
class_counts = df.groupby("churn").size()
```

```
# Create bar chart of the value counts using Plotly
fig = go.Figure(
    data=[go.Bar(x=class_counts.index, y=class_counts.values)],
    layout=go.Layout(title="Churn Distribution", xaxis=dict(tickvals=[0, 1], ticktext=["Not Churn", "Churn"]),
          hovermode = 'closest',width=600)
)

fig.show()
```

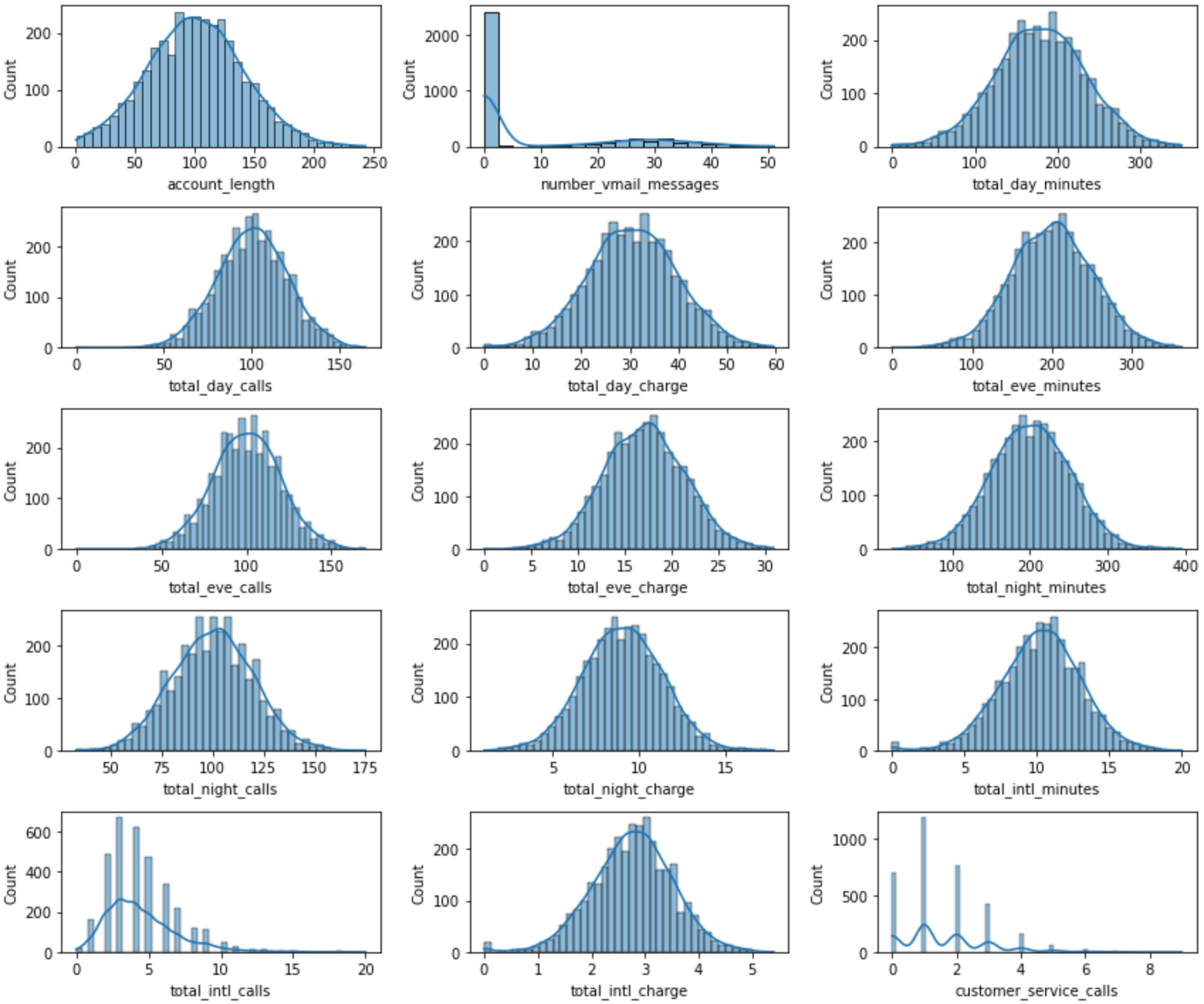


Out of the 3,333 customers in the dataset, 483 have ended their contracts. This represents a churn rate of about 15%; indicating an imbalance between the churn and non-churn classes.

This class imbalance must be addressed before modeling, as it can lead to the model making false predictions

Distribution of the numerical features

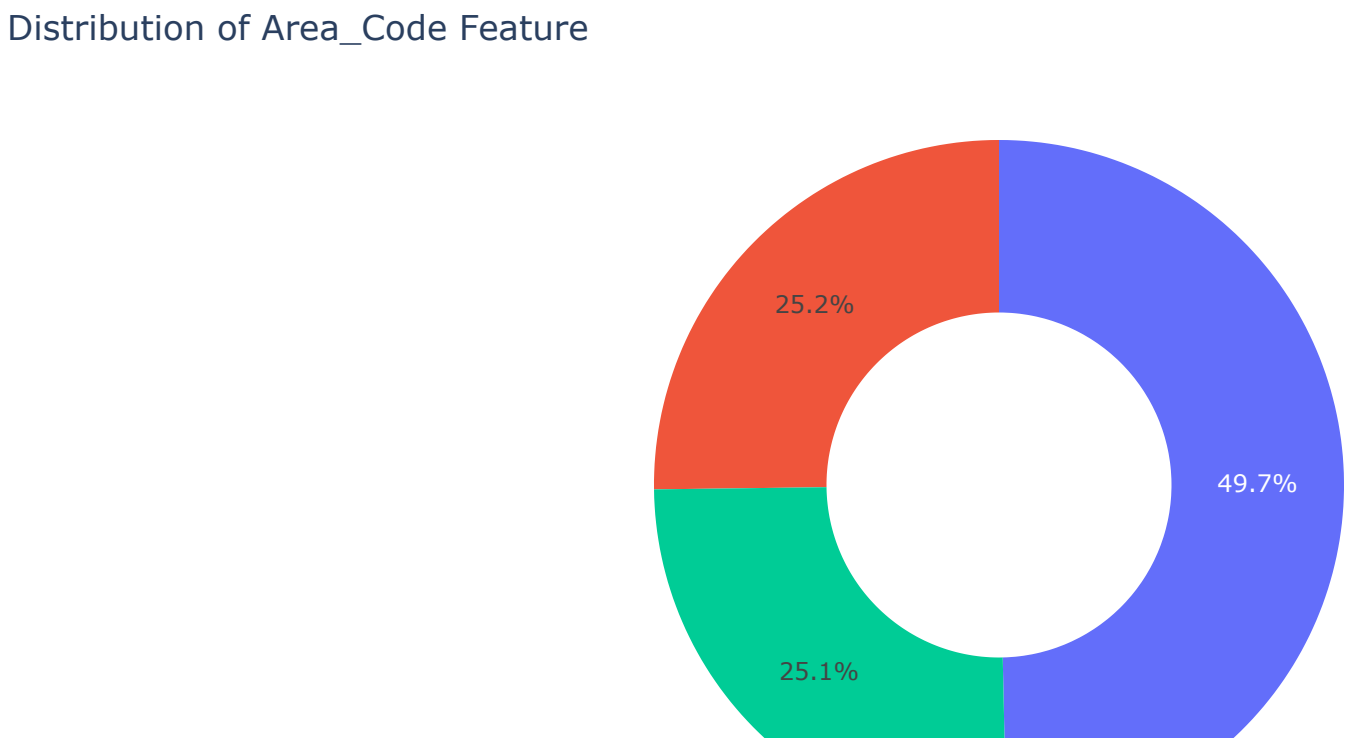
In [16]:
```
#Check distribution of numeric features
numeric_features = ['account_length', 'number_vmail_messages', 'total_day_minutes', 'total_day_calls', 'total_day_charge'
'total_eve_minutes', 'total_eve_calls', 'total_eve_charge', 'total_night_minutes', 'total_night_calls',
'total_night_charge', 'total_intl_minutes', 'total_intl_calls', 'total_intl_charge', 'customer_service_calls']

# Calculate the number of rows and columns for subplots
nrows = (len(numeric_features) - 1) // 3 + 1
ncols = min(3, len(numeric_features))

# Create subplots
fig, axes = plt.subplots(nrows=nrows, ncols=ncols, figsize=(12, 10))

# Flatten axes
axes = axes.flatten() if nrows > 1 else [axes]

# Plot numeric features
for i, feature in enumerate(numeric_features):
    ax = axes[i]
    sns.histplot(df[feature], kde=True, ax=ax)
    ax.set_xlabel(feature)
    ax.set_ylabel("Count")

# Remove empty subplots
if len(numeric_features) < nrows * ncols:
    for i in range(len(numeric_features), nrows * ncols):
        fig.delaxes(axes[i])

# Adjust subplot spacing
fig.tight_layout()

plt.show()
```


Count
account_length
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

Based on the distribution plots of the features above, most variables exhibit a roughly normal distribution, except for customer_service_calls and number_vmail_messages. While total_intl_calls appears slightly right-skewed, it still maintains a generally normal pattern. In contrast, customer_service_calls displays multiple peaks, suggesting the presence of several modes within the population.

Distribution of the "area_code" feature

In [17]:
```
# Area_code feature pie chart
area = df['area_code'].value_counts()
transanction = area.index
quantity = area.values

# Create pie circle
figure = px.pie(df,
                values = quantity,
                names = transanction,
                hole = .5,
                title = 'Distribution of Area_Code Feature')
figure.show()
```



- Around half of the customers are in `area code 415`.
- A quarter of the customers are in `area code 510` and another quarter are in `area code 408`.

Distribution of the categorical features

Here, we will be analysisng the 3 categorical features in our dataset which include: `state`, `international_plan` and `voice_mail_plan`

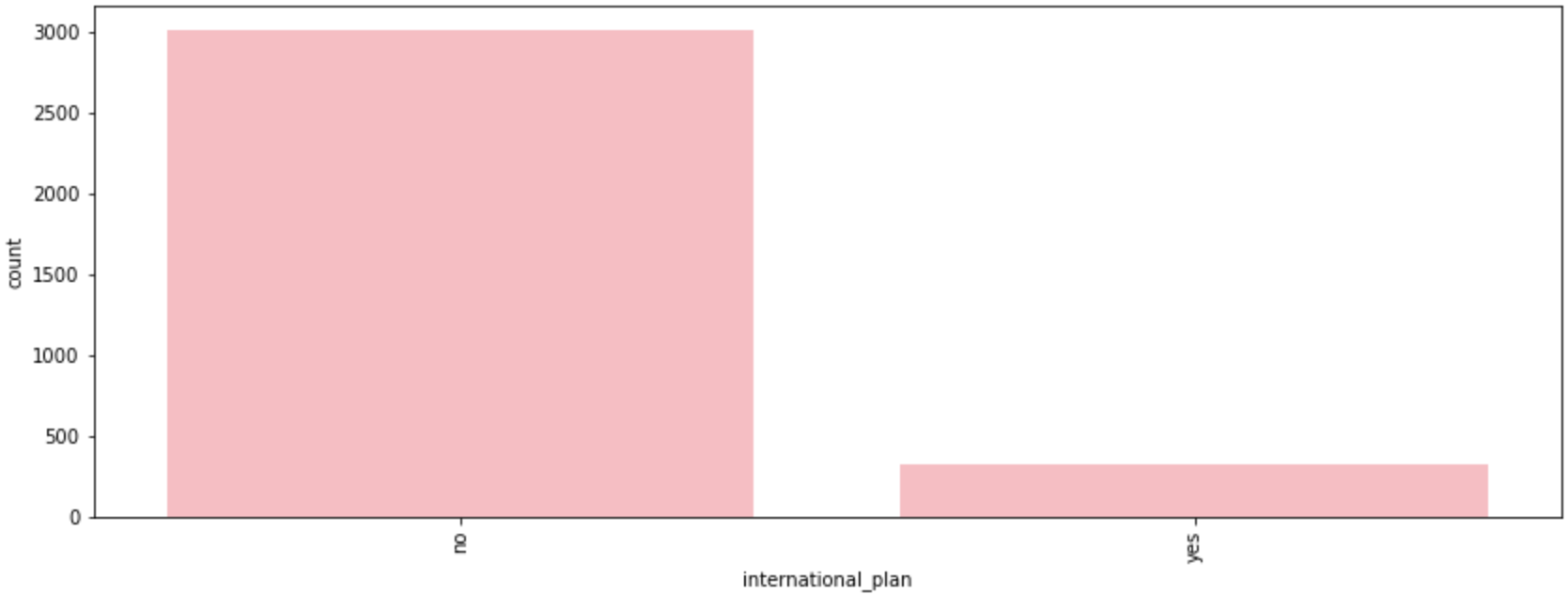
In [18]:
```
#Check distribution of categorical features
def plot_categorical_distribution(data, feature):
    """
    Plots the distribution of a categorical feature in the given data.
    """
    plt.figure(figsize=(14, 5))
    sns.countplot(x=feature, data=data,color='lightpink', order=data[feature].value_counts().index)
    plt.xticks(rotation=90)
    plt.show()
```

International plan

In [19]:
```
df['international_plan'].value_counts()
```

Out[19]:
```
no     3010
yes     323
Name: international_plan, dtype: int64
```

In [20]:
```
plot_categorical_distribution(df,'international_plan')
```



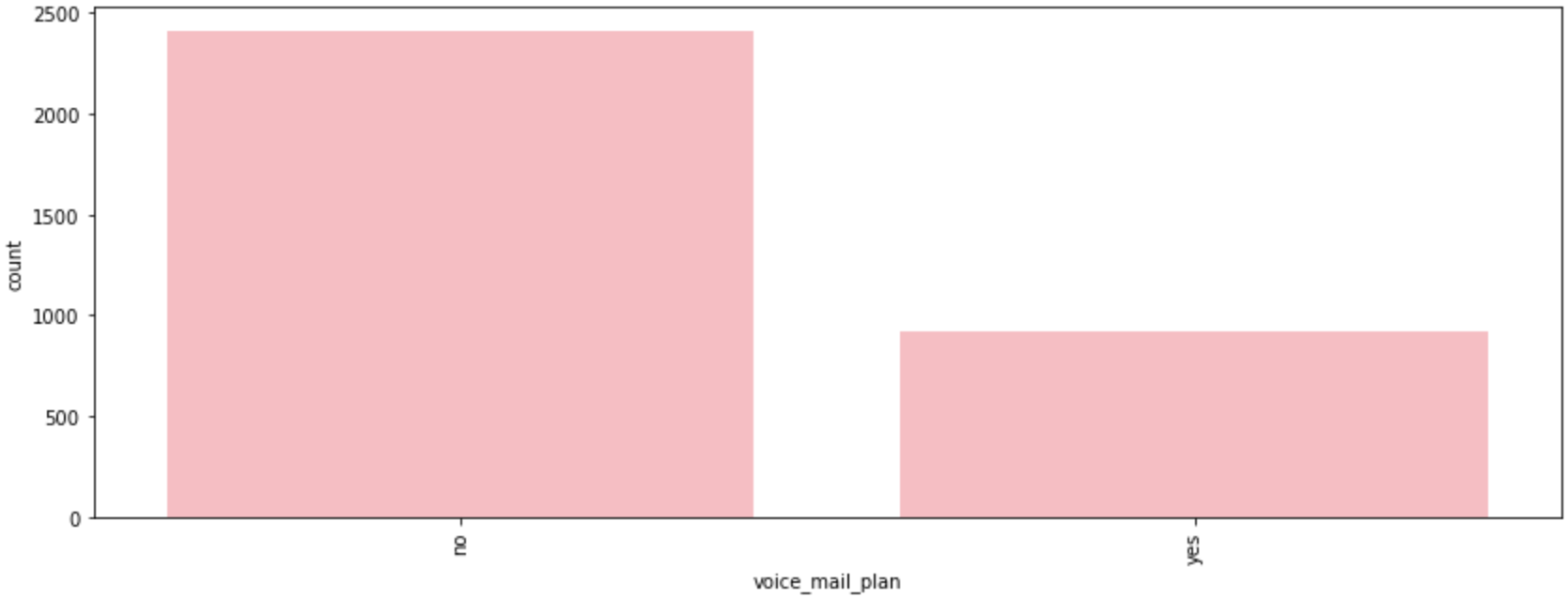
Out of the 3,333 customers, only 323 have subscribed to an international plan, which accounts for approximately 9.7% of the total customer base.

Voicemail plan

In [21]:
```
df['voice_mail_plan'].value_counts()
```

Out[21]:
```
no     2411
yes     922
Name: voice_mail_plan, dtype: int64
```

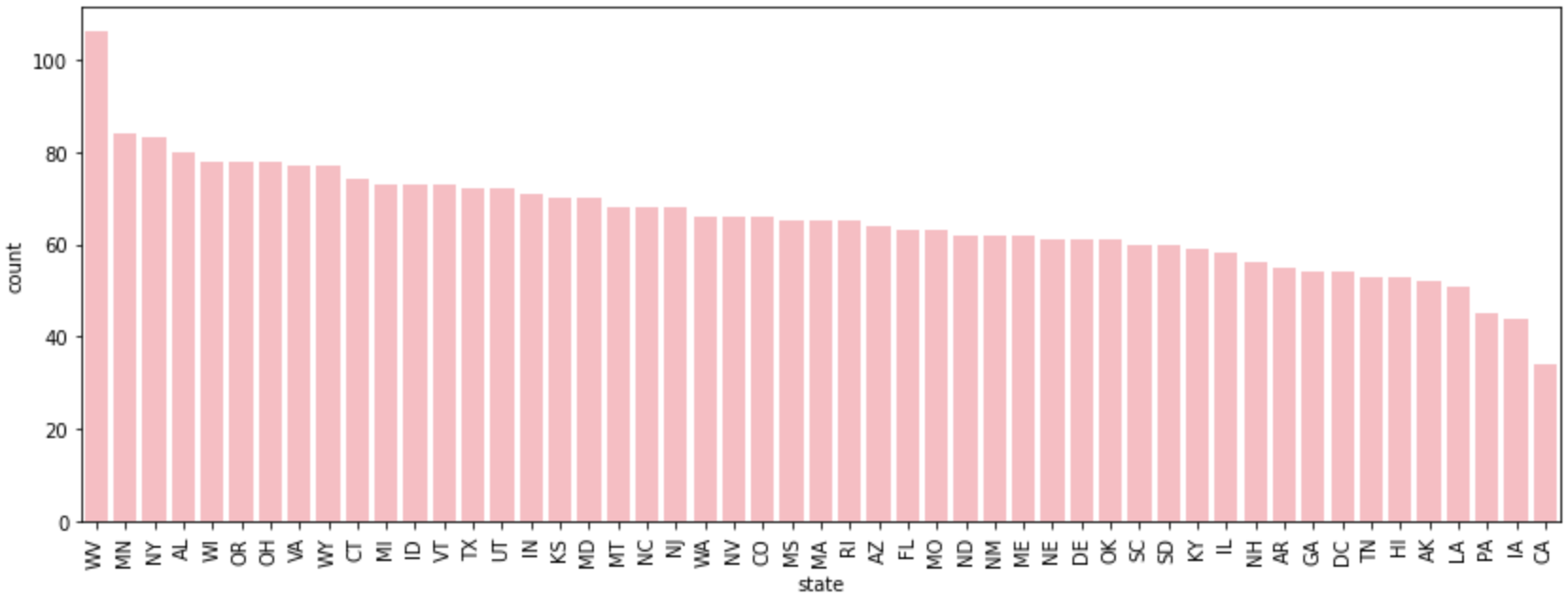
In [22]:
```
plot_categorical_distribution(df, 'voice_mail_plan')
```



Out of the 3,333 customers, 922 have a voicemail plan, which represents approximately 27.7% of the total customer base.

State

In [23]:
```
plot_categorical_distribution(df, 'state')
```

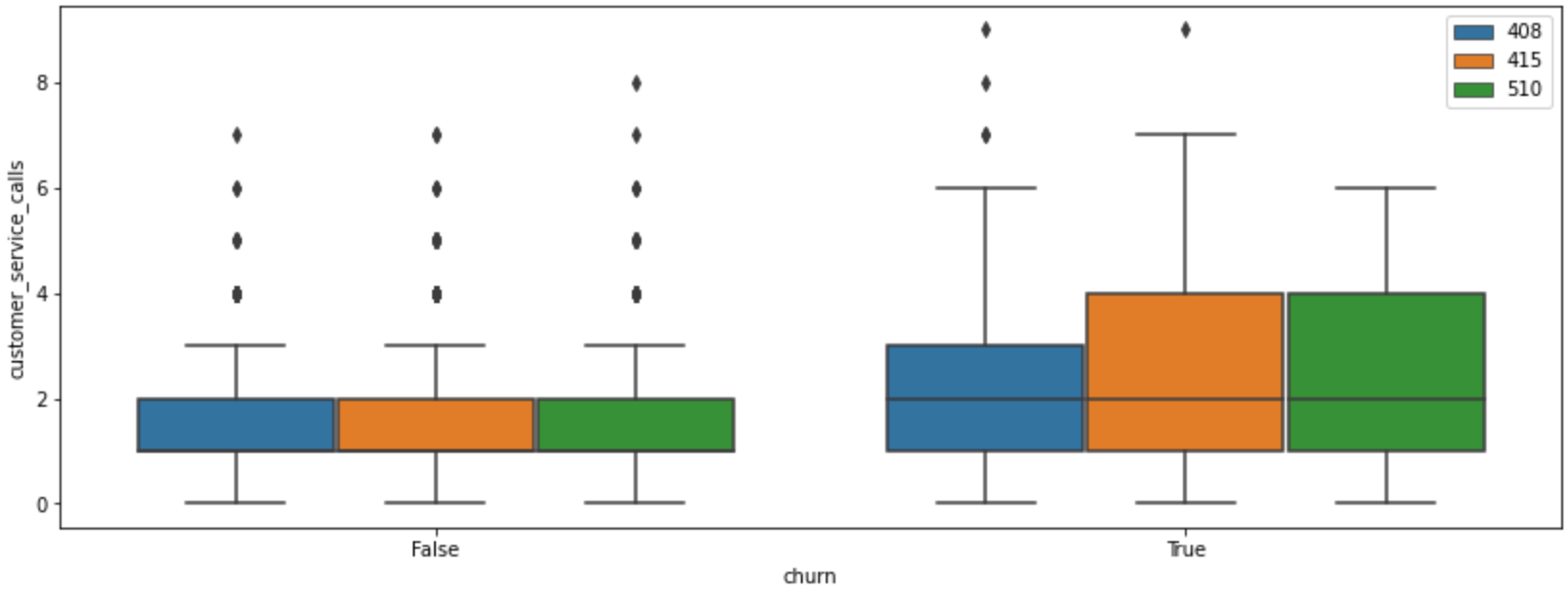


Most customers are from `West Virginia` , `Minnesota` , `NewYork` , `Alabama` and `Wisconsin` .

Bivariate Analysis

In this section, we analyze the relationships between pairs of variables in the dataset to understand how changes in one may be associated with changes in another.

In [24]:
```
# Boxplot showing which area code has highest churn
plt.figure(figsize=(14,5))
sns.boxplot(data=df,x='churn',y='customer_service_calls',hue='area_code');
plt.legend(loc='upper right');
```

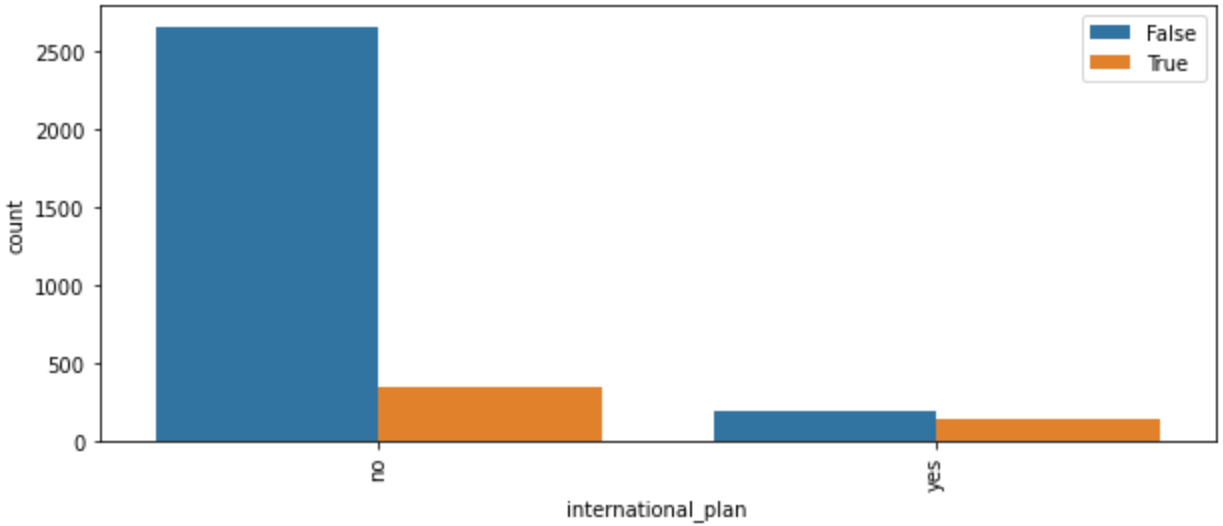


Among the customers who have churned, the majority are from area codes 415 and 510. Additionally, the data shows a noticeable presence of outliers.

In [25]:
```
#Check distribution of categorical features based on churn rate
def plot_categorical_distribution(data, feature):
    """
    Plots the distribution of a categorical feature in the given data.
    """
    plt.figure(figsize=(10, 4))
    churn_counts = data.groupby(feature)["churn"].sum().sort_values(ascending=False)
    top_10_categories = churn_counts.head(10).index.tolist()
    sns.countplot(x=feature, hue="churn", data=data, order=top_10_categories)
    plt.xticks(rotation=90)
    plt.legend(loc="upper right")
    plt.show()
```

International plan

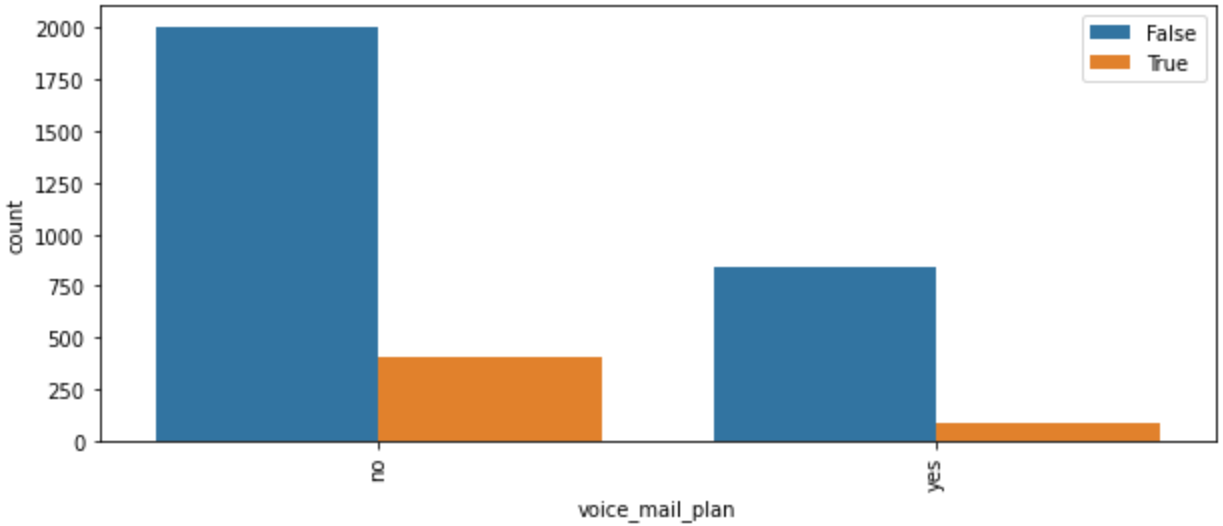
In [26]:
```
plot_categorical_distribution(df, 'international_plan')
```



Most of the customers who churned didn't have an international plan.

Voice Mail plan

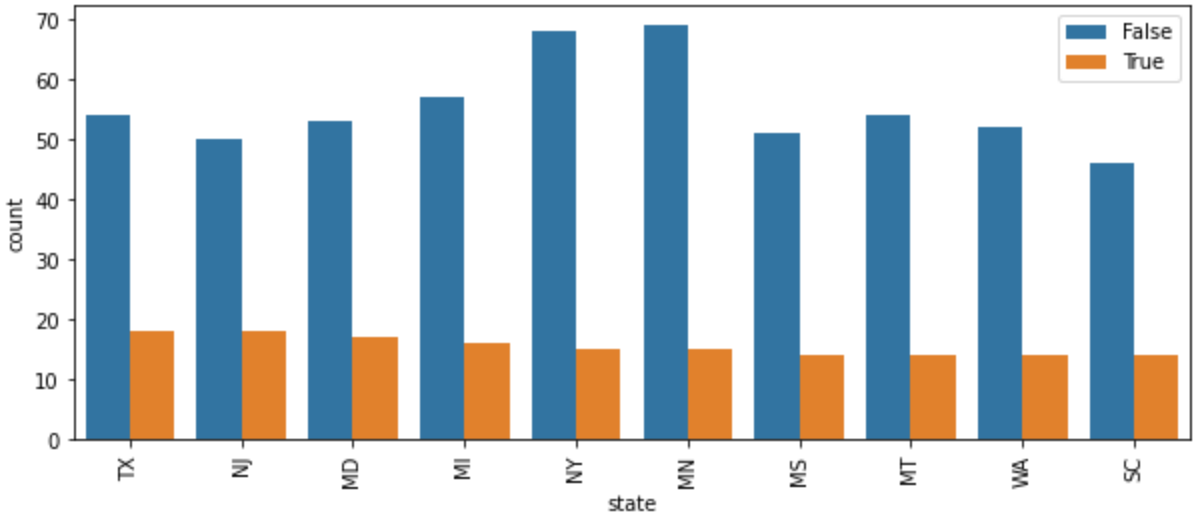
In [27]:
```
plot_categorical_distribution(df, 'voice_mail_plan')
```



Most of the customers who churned didn't have an voicemail plan.

State

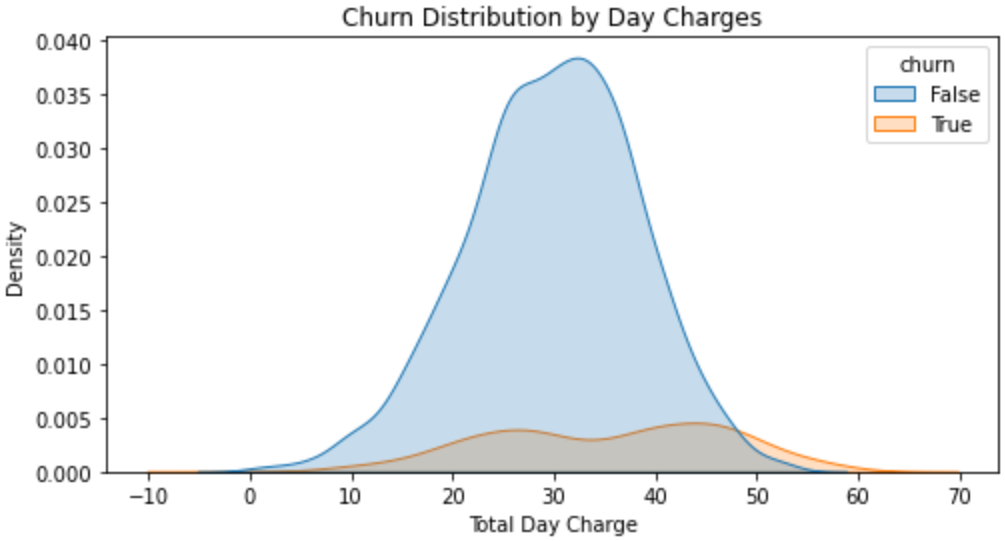
In [28]:
```
plot_categorical_distribution(df, 'state')
```



Amongst all the customers that churned, most of them are from `Texas` , `New Jersey` , `Maryland` , `Miami` and `NewYork` .

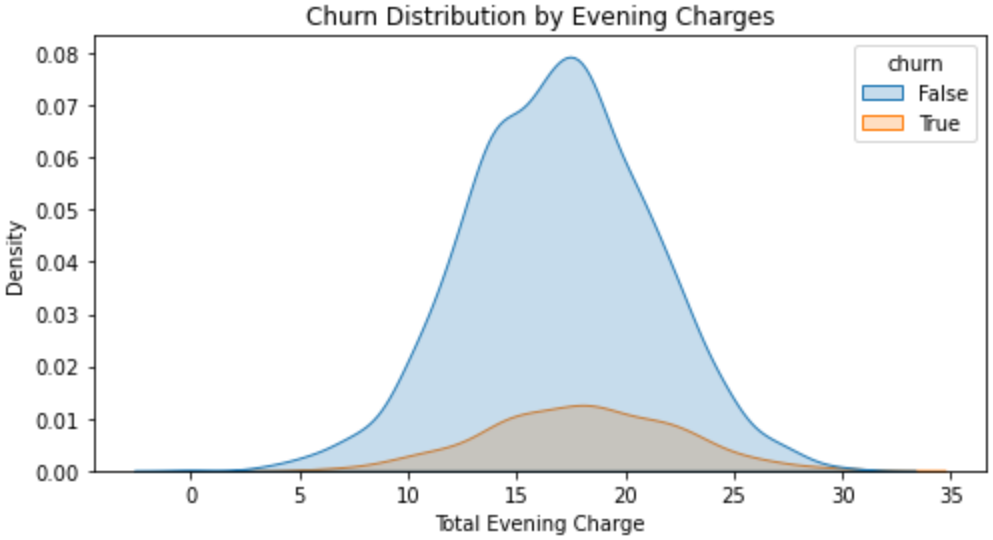
In [29]:
```
def plot_churn_kde(data, x_column, charge_type):
    """
    A function to plot features based on churn rate
    """
    plt.figure(figsize=(8, 4))
    sns.kdeplot(data=data, x=x_column, hue='churn', fill=True)
    plt.xlabel(f'Total {charge_type} Charge')
    plt.ylabel('Density')
    plt.title(f'Churn Distribution by {charge_type} Charges')
    plt.show()
```

In [30]:
```
# Churn by day charges
plot_churn_kde(df, 'total_day_charge', 'Day')
```



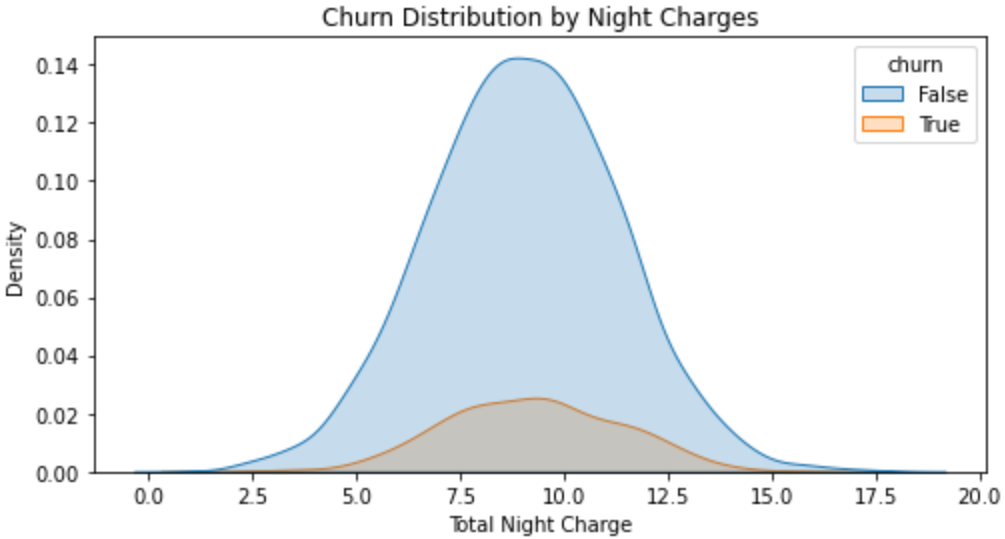
The KDE plot of churn versus total day charges indicates that customers who have churned generally incur higher day charges compared to those who stayed. This pattern suggests that higher daytime costs may be a contributing factor to customer dissatisfaction and could increase the likelihood of churn.

In [31]:
```
# Churn by evening charges
plot_churn_kde(df, 'total_eve_charge', 'Evening')
```



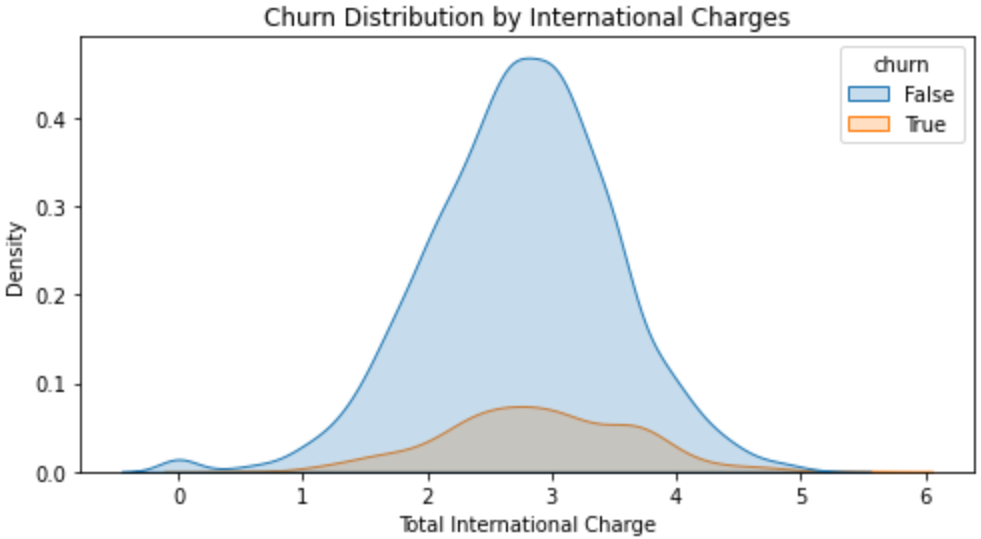
The KDE plot for churn versus evening charges reveals a pattern similar to that of day charges; customers who have churned typically exhibit higher total evening charges compared to those who remained.

In [32]:
```
# Churn by night charges
plot_churn_kde(df, 'total_night_charge', 'Night')
```



The KDE plot for churn based on night charges follows a similar trend observed with day and evening charges; customers who churned generally incurred higher total night charges compared to those who stayed

In [33]:
```
plot_churn_kde(df, 'total_intl_charge', 'International')
```



The plot indicates that customers with higher total international charges appear to have a slightly increased tendency to churn.

Dealing with Outliers

Outliers can negatively influence the performance of predictive models by adding noise and distorting the training process. By removing values that fall beyond 3 standard deviations from the mean, we enhance the model's ability to capture underlying patterns and improve its accuracy on unseen data.

In [34]:
```
def drop_numerical_outliers(df, z_thresh=3):
    constrains = df.select_dtypes(include=[np.number]).apply(lambda x: np.abs(stats.zscore(x)) < z_thresh) \
        .all(axis=1)
    df.drop(df.index[~constrains], inplace=True)

drop_numerical_outliers(df)
print(df.shape)
```

```
(3169, 20)
```

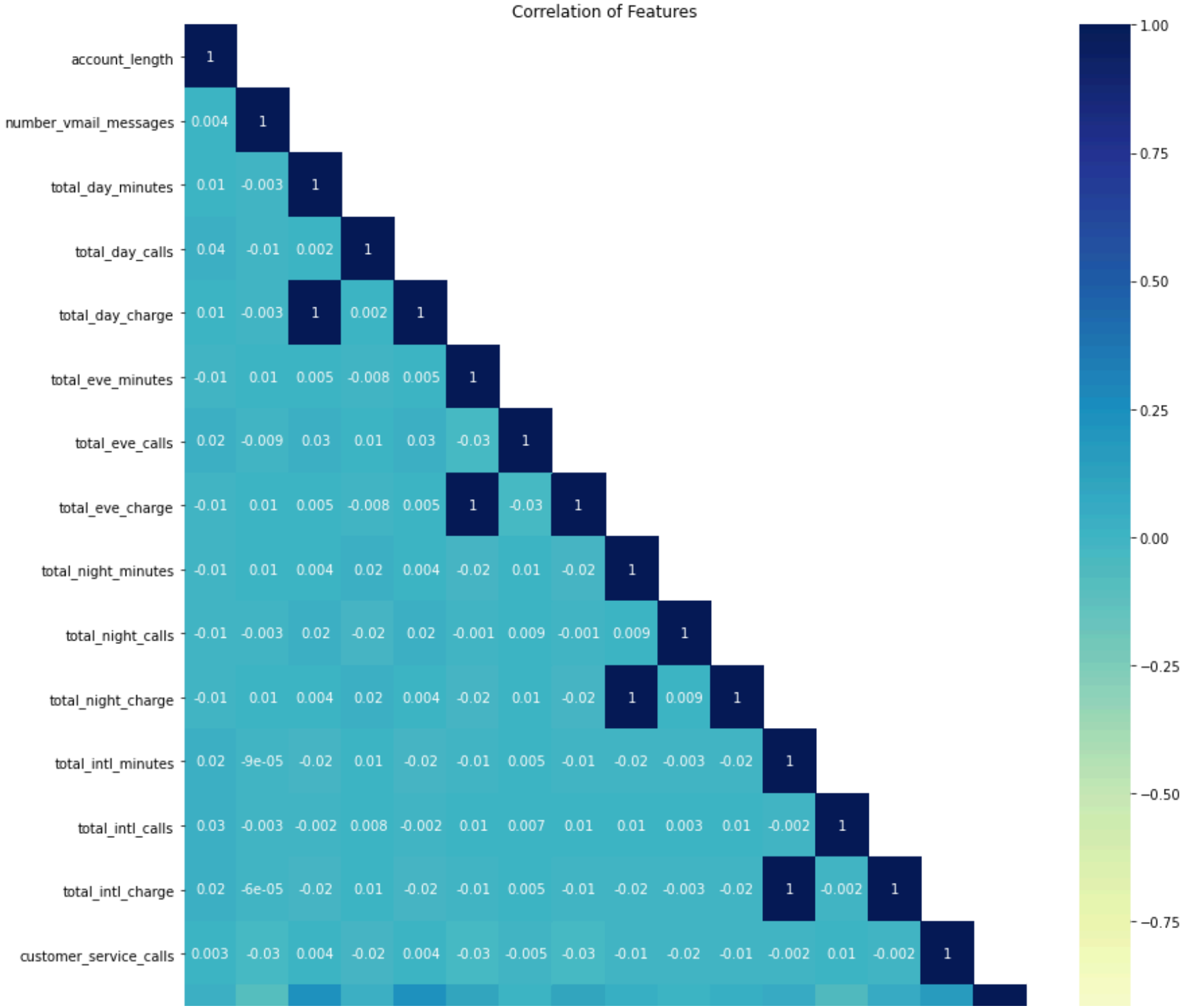
Features Correlation

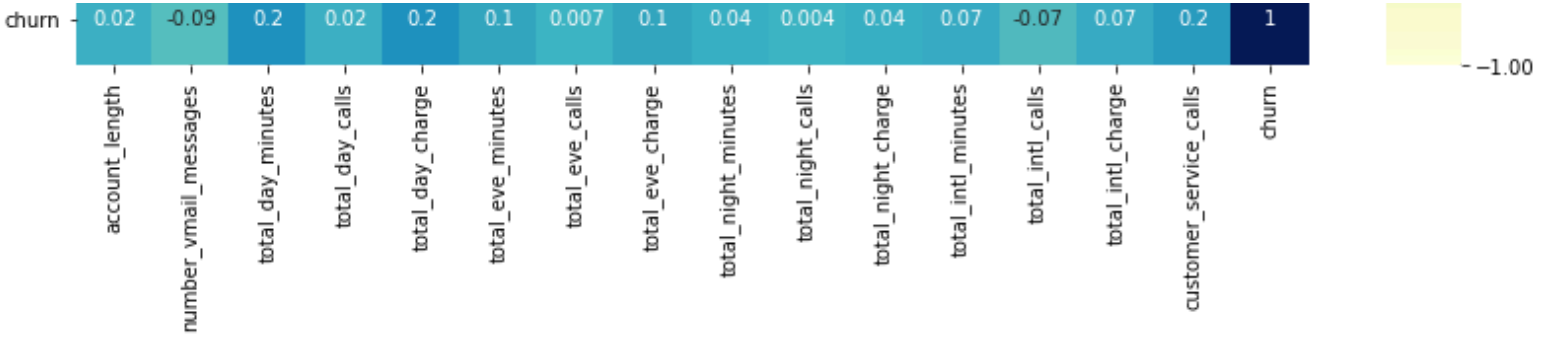
Examining which features exhibit a strong correlation with the target variable (churn).

In [35]:
```
# Using a Heatmap to find the correlation between features
def corrmatrix(df):
    ''' This function plots a correlation matrix for a given dataframe '''
    plt.figure(figsize=(14,14))
```

```
    corr = df.corr()

    # Create mask to only show bottom triangle
    corr_tri = corr.where(np.tril(np.ones(corr.shape)).astype(np.bool))

    sns.heatmap(data = corr_tri, center = 0, cmap = "YlGnBu", annot = True, fmt='.1g',vmin=-1);
    plt.title('Correlation of Features')
    plt.show()
```

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


Correlation of Features
account_length
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
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75



While most features show low correlation with each other, a few exhibit perfect positive correlation:

- Total day charge and total day minutes
- Total eve charge and total eve minutes
- Total night charge and total night minutes
- Total int charge and total int minutes

This strong correlation is expected, as the charges are directly calculated based on the number of minutes used in each time category.

Multicollinearity check

To assess multicollinearity among features, a correlation matrix was used. Multicollinearity arises when two or more features are highly correlated with one another, which can lead to problems during modeling, such as model instability, overfitting, and inaccurate coefficient estimates. To counter this, features with correlation values exceeding 0.9 were identified and removed.

In [37]:
```
# Calculate the correlation matrix and obtain absolute value
corr_matrix = df.corr().abs()

# Create and apply a True/False mask
mask = np.triu(np.ones_like(corr_matrix, dtype=bool))
tri_df = corr_matrix.mask(mask)

# List column names of highly correlated features (r > 0.90)
to_drop = [c for c in tri_df.columns if any(tri_df[c] >  0.90)]

# Drop the features
df = df.drop(to_drop, axis=1)
```

Feature Engineering

This phase involves transforming raw data into features that more effectively capture the underlying problem to the predictive models, resulting in improved model accuracy on unseen data. In this phase, we'll perform Label Encoding, One Hot Encoding and Feature Scaling the data.

Label Encoding

It is a technique used to convert categorical variables into numerical value by assigning a unique integer to each category.

In [38]:
```
# Convert columns with 'yes' or 'no' to binary using LabelEncoder
label_encoder = LabelEncoder()
df['churn'] = label_encoder.fit_transform(df['churn'])
```

One Hot Encoding

It is a technique used to convert categorical variables into a set of binary features by creating a new feature for each category, and then assigning a value of 1 to the feature if the category is present and 0 if it is not.

In [39]:
```
df = pd.get_dummies(df,columns = ['state', 'area_code','international_plan','voice_mail_plan'])
df.head()
```

Out[39]:

	account_length	number_vmail_messages	total_day_calls	total_day_charge	total_eve_calls	total_eve_charge	total_night_calls	total_night_charge
0	128	25	110	45.07	99	16.78	91	11.0
1	107	26	123	27.47	103	16.62	103	11.4
2	137	0	114	41.38	110	10.30	104	7.3
3	84	0	71	50.90	88	5.26	89	8.8
4	75	0	113	28.34	122	12.61	121	8.4

5 rows × 70 columns

Feature Scaling the data

Feature Scaling is a preprocessing technique used to transform numerical features into a comparable range. It helps in reducing the impact of outliers and standardizing the variables.

A commonly used method is Min-Max Normalization, which rescales the data so that the minimum value becomes 0 and the maximum becomes 1, with all other values proportionally adjusted within this range.

In [40]:
```
scaler = MinMaxScaler()

def scaling(columns):
    return scaler.fit_transform(df[columns].values.reshape(-1,1))

for i in df.select_dtypes(include=[np.number]).columns:
    df[i] = scaling(i)
df.head()
```

Out[40]:

	account_length	number_vmail_messages	total_day_calls	total_day_charge	total_eve_calls	total_eve_charge	total_night_calls	total_night_charge
0	0.587963	0.510204	0.576271	0.773956	0.487179	0.490082	0.422414	0.64364
1	0.490741	0.530612	0.686441	0.450248	0.521368	0.483858	0.525862	0.67597
2	0.629630	0.000000	0.610169	0.706088	0.581197	0.238040	0.534483	0.37252
3	0.384259	0.000000	0.245763	0.881184	0.393162	0.042007	0.405172	0.48567
4	0.342593	0.000000	0.601695	0.466250	0.683761	0.327888	0.681034	0.45260

5 rows × 70 columns

Modeling

In this phase, we will build a model that can accurately classify and predict whether a customer is likely to churn based on the features in our dataset.The primary evaluation metric for success will be the recall score, with a target threshold of 80% or higher.Achieving this ensures the model is effective at correctly identifying churned customers, which is critical for proactive retention strategies.

To meet the objectives outlined in the project proposal, we will implement and compare the performance of the following machine learning algorithms:

- Logistic Regression
- Decision Tree
- Random Forest
- XG Boost

We will also be using the `ROC_AUC` metric to evaluate the overall discriminatory power of the models.

To deal with class imbalance, we will be using `SMOTE` to generate synthetic examples of the minority class in our dataset and help the models learn balanced decision boundaries.

Let's define X and y

In [41]:
```
#Define X and y
X = df.drop("churn", axis=1)
y = df["churn"]
```

Train-Test Split

We are splitting the data into train and test sets using a test_size of 0.25

In [42]:
```
#splitting the data into train and test sets
X_train,X_test,y_train,y_test = train_test_split(X,y, test_size=0.25, random_state=123)
```

Applying SMOTE to Resolve Unbalanced 'churn' Feature

Synthetic Minority Oversampling Technique (SMOTE) is an advanced oversampling method used to address class imbalance by generating synthetic examples for the minority class. Unlike random oversampling, which can lead to overfitting, SMOTE works by creating new instances through interpolation between neighboring minority class examples in the feature space.The technique aims to balance class distribution by randomly increasing minority class examples by replicating them.

In [43]:
```
#instantiate SMOTENC
from imblearn.over_sampling import SMOTE, SMOTENC

smote = SMOTENC(categorical_features = [1,2],random_state = 123)
resampled_X_train, resampled_y_train = smote.fit_resample(X_train,y_train)
```

Logistic Regression

Logistic Regression is a statistical method used for binary classification problems, where the target variable has two possible outcomes (in this case, churn or no churn). It models the relationship between one or more independent variables and a binary dependent variable by estimating the likelihood that a given input belongs to a particular class. Its goal is to estimate the probability of an instance belonging to a specific class based on the values of the independent variables.

In [44]:
```
#instantiate the logistic regression
logreg = LogisticRegression(random_state=123)
```

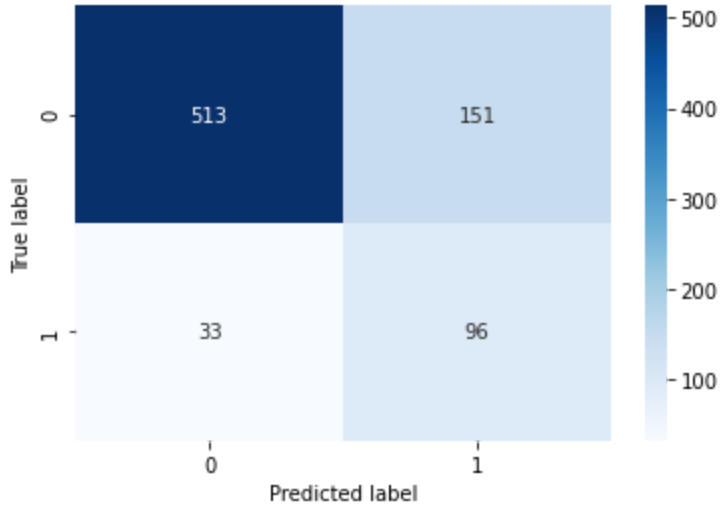
In [45]:
```
# Fit the model on the training data
logreg.fit(resampled_X_train, resampled_y_train)
```

Out[45]:
```
LogisticRegression(random_state=123)
```

In [46]:
```
#predict on the labels of test set
y_pred_log = logreg.predict(X_test)
```

In [47]:
```
def plot_confusion_matrix(y_true, y_pred, classes):
    """
    Plots a confusion matrix.
    """
    cm = confusion_matrix(y_true, y_pred)
    plt.figure()
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=classes, yticklabels=classes)
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.show()
```

In [48]:
```
plot_confusion_matrix(y_test, y_pred_log, [0,1])
```



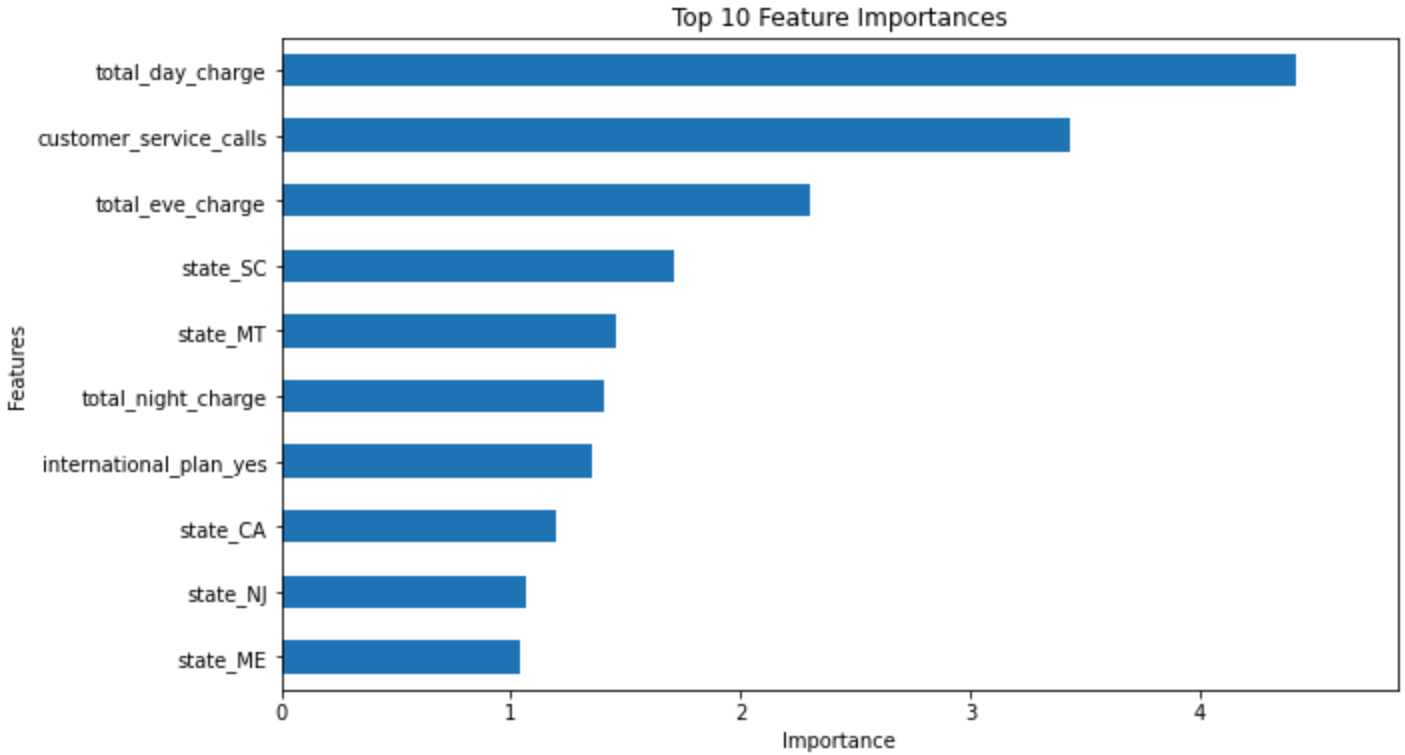
In [49]:
```
print(classification_report(y_test,y_pred_log))
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.77      0.85       664
         1.0       0.39      0.74      0.51       129
```

```
              accuracy                           0.77       793
             macro avg       0.66      0.76      0.68       793
          weighted avg       0.85      0.77      0.79       793
```

In [50]:

```
# Feature Importances
importance = logreg.coef_[0]
feature_names = resampled_X_train.columns
feature_importances = pd.Series(importance,index=feature_names)
feature_importances = feature_importances.sort_values(ascending=False)
plt.figure(figsize=(10, 6))

# Select the top 10 features
top_features = feature_importances[:10]
top_features.sort_values().plot(kind='barh')
plt.xlabel('Importance')
plt.ylabel('Features')
plt.title('Top 10 Feature Importances')

# Set the xlim to the maximum importance value
plt.xlim(0, max(top_features)* 1.1)
plt.show()
```



The logistic regression model has a recall score of 0.74, which is a strong result for a baseline model meaning that the model can identify around 74% of customers who actually churn.

The confusion matrix shows a higher count of true positives and true negatives compared to false positives and false negatives, suggesting the model is making accurate predictions more often and is not overfitting.

Based on feature importance, the top three predictors of churn identified by the model are: `total_day_charge` , `customer_service_calls` , `total_eve_charge` .

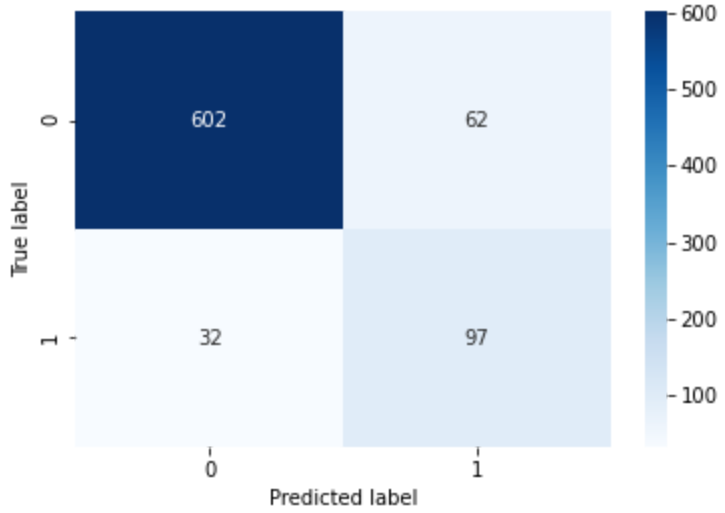
Decision Tree Classifier

A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. Decision trees work by splitting the dataset into smaller subsets based on feature values, forming a tree-like structure. Each internal node represents a decision based on a feature, and each leaf node represents a class label. The decision tree then predicts the class of a new data point by following the path down the tree that corresponds to the values of its features.

In [51]:
```
#Instantiate DecisionTreeClassifier
dt_clf = DecisionTreeClassifier(random_state=123)
```

In [52]:
```
#Fit on the training data
dt_clf.fit(resampled_X_train,resampled_y_train)

#predict on the test set
y_pred_dt = dt_clf.predict(X_test)
```

In [53]:
```
plot_confusion_matrix(y_test, y_pred_dt, [0,1])
```



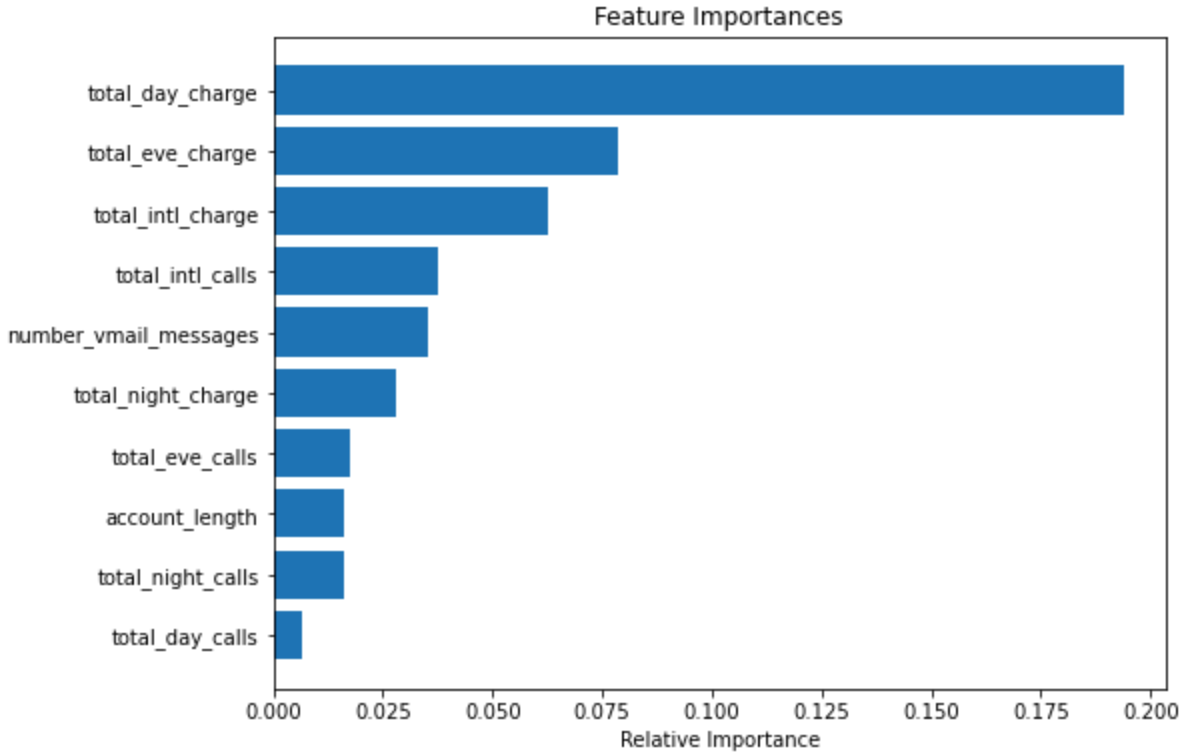
In [54]:
```
print(classification_report(y_test,y_pred_dt))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.91      0.93       664
         1.0       0.61      0.75      0.67       129

    accuracy                           0.88       793
   macro avg       0.78      0.83      0.80       793
weighted avg       0.89      0.88      0.89       793
```

Let's outline the Feature Importances

In [55]:
```
# Feature Importances
feature_names = list(resampled_X_train.columns)
importances = dt_clf.feature_importances_[0:10]
```

```
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



The Decision Tree model achieved a recall score of 0.73, which is fairly strong but slightly lower than the baseline Logistic Regression model. This means it correctly identifies about 73% of actual churn cases

The confusion matrix indicates that the model predicts true positives and true negatives more frequently than false positives and false negatives. This suggests the model is making reliable predictions and shows no signs of overfitting.

Based on the model's feature importance scores, the top three predictors of churn are: `total_day_charge`, `total_eve_charge`, `total_intl_charge`.

Random Forest Classifier

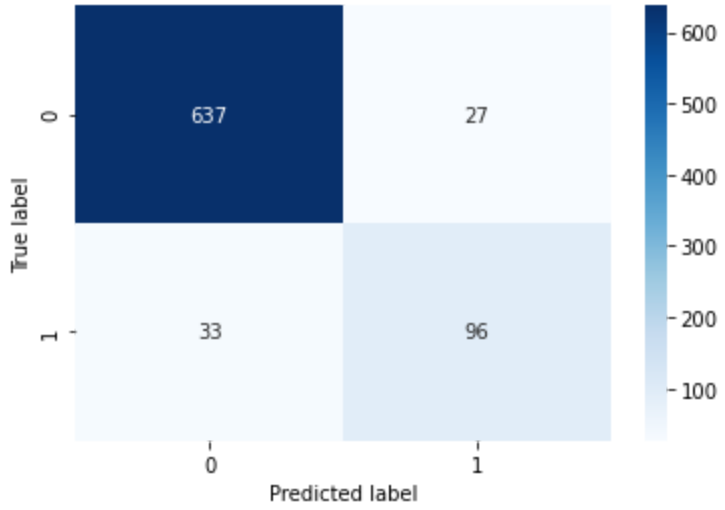
Random Forest is an ensemble learning method used for classification, regression, and other tasks. It builds multiple decision trees during training and combines their outputs to improve accuracy and robustness. For classification tasks, it predicts the class based on the majority vote from all trees, while for regression, it averages their predictions. This approach reduces overfitting, increases generalization, and typically results in better performance compared to a single decision tree.

In [56]:
```
#Instantiate the classifier
rf_clf= RandomForestClassifier(random_state=123)

#Fit on the training data
rf_clf.fit(resampled_X_train,resampled_y_train)
```

Out[56]:
```
RandomForestClassifier(random_state=123)
```

In [57]:
```
#predict on the test data
y_pred_rf = rf_clf.predict(X_test)
```

In [58]:
```
plot_confusion_matrix(y_test, y_pred_rf, [0,1])
```

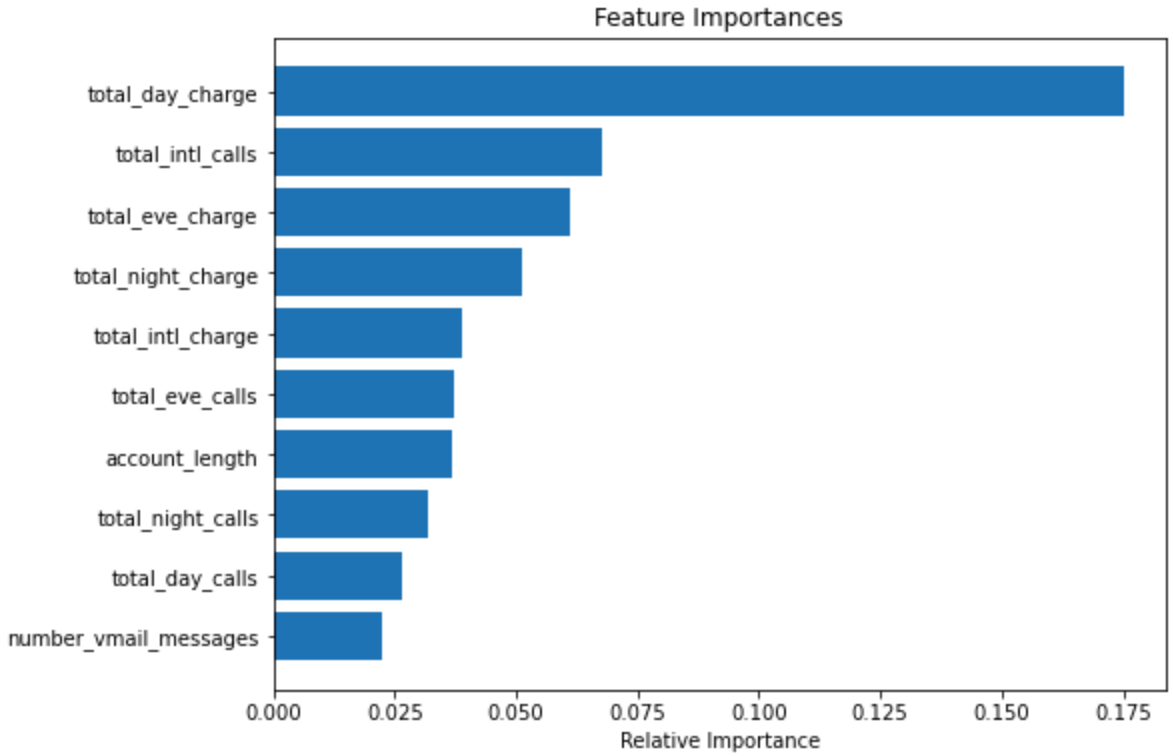


In [59]:
```
print(classification_report(y_test,y_pred_rf))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.96       664
```

```
           1.0       0.78      0.74      0.76       129

      accuracy                           0.92       793
     macro avg       0.87      0.85      0.86       793
  weighted avg       0.92      0.92      0.92       793
```

In [60]:
```
feature_names = list(resampled_X_train.columns)
importances = rf_clf.feature_importances_[0:10]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



The Random Forest Classifier achieved a recall score of 0.74, matching the performance of the Logistic Regression model and outperforming the Decision Tree. This indicates that the model correctly identifies approximately 74% of customers who are likely to churn.

The confusion matrix reveals a higher number of true positives and true negatives compared to false predictions, suggesting that the model is making reliable decisions and is not overfitting the data.

According to the feature importance scores, the top three predictors of churn in this model are `total_day_charge`, `total_intl_calls`, `total_eve_charge`.

XGBoost

XGBoost (Extreme Gradient Boosting) is a highly efficient and scalable machine learning algorithm commonly used for classification and regression tasks.It utilizes gradient boosting and ensemble learning techniques to combine multiple weak models, typically decision trees, into a strong predictive model.

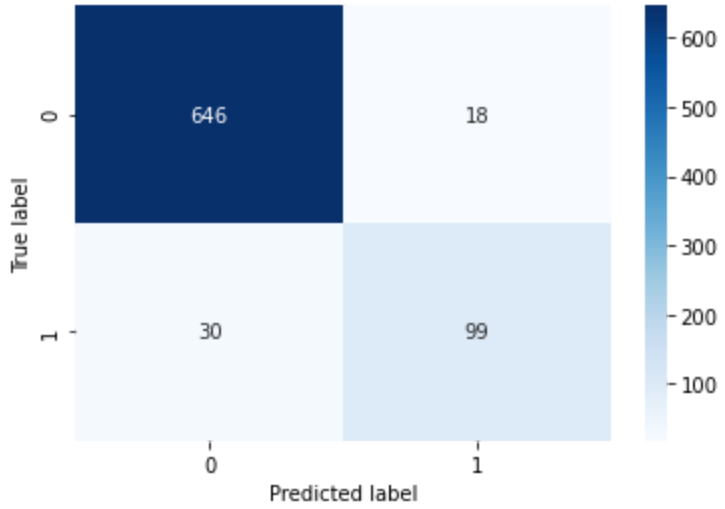
In [61]:
```
from xgboost import XGBClassifier
```

In [62]:
```
#instantiate XGBClassifier
xg_clf = XGBClassifier(random_state=123)

#Fit on the training data
xg_clf.fit(resampled_X_train,resampled_y_train)
```

Out[62]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.300000012, max_delta_step=0, max_depth=6,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=123,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [63]:
```
#predict on the test data
y_pred_xg = xg_clf.predict(X_test)
```

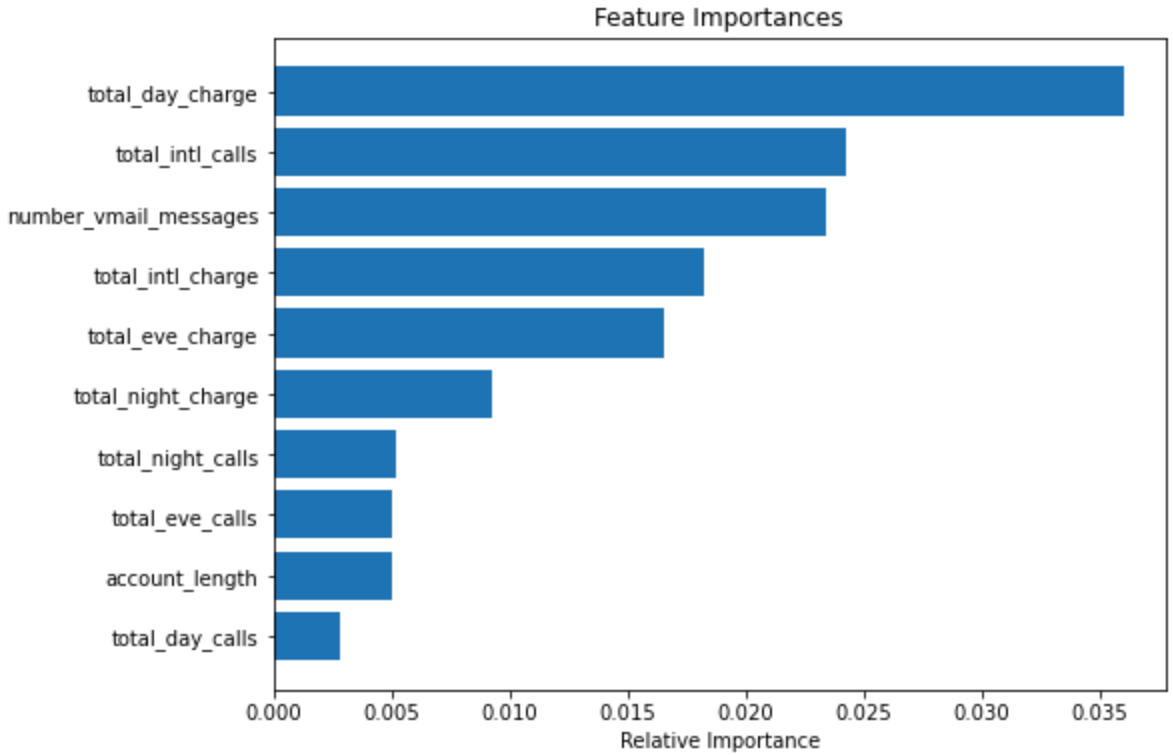
In [64]:
```
plot_confusion_matrix(y_test, y_pred_xg, [0,1])
```



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

```
              precision    recall  f1-score   support

         0.0       0.96      0.97      0.96       664
         1.0       0.85      0.77      0.80       129

    accuracy                           0.94       793
   macro avg       0.90      0.87      0.88       793
weighted avg       0.94      0.94      0.94       793
```

In [66]:
```
feature_names = list(resampled_X_train.columns)
importances = xg_clf.feature_importances_[0:10]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



The XGBoost Classifier achieved a recall score of 0.77, outperforming all previous models. This indicates that it can correctly identify approximately 77% of customers who are likely to churn, making it the most effective model so far in terms of recall.

The confusion matrix confirms that the model has more true positives and true negatives than false predictions, suggesting it is performing reliably without signs of overfitting.

Based on the model's feature importance scores, the top three drivers of churn prediction are: `total_day_charge` , `total_intl_calls` , `number_vmail_messages` .

Model Evaluation

In this phase, we will evaluate the performance of our models using two key metrics: Recall Score and ROC AUC. These metrics help us determine how well each model identifies churned customers and distinguishes between the classes.

After the evaluation, we will select the top two performing models and apply hyperparameter tuning to optimize their performance further and to obtain the best model.

Models Comparison - Recall Score

Recall score measures the proportion of actual positive instances (such as churned customers) that the model correctly identifies.

A higher recall indicates that the model is more effective at detecting customers who are likely to churn, which is especially important in churn prediction tasks where missing a churned customer can have significant business implications.

In [67]:
```
np.random.seed(123)

classifiers = [LogisticRegression(),
               RandomForestClassifier(),
               DecisionTreeClassifier(),
               XGBClassifier()]

# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'recall'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(resampled_X_train, resampled_y_train)
    y_pred = model.predict(X_test)

    recall = recall_score(y_test, y_pred)

    result_table = result_table.append({'classifiers': cls.__class__.__name__,
                                        'recall': recall}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

result_table
```

Out[67]:

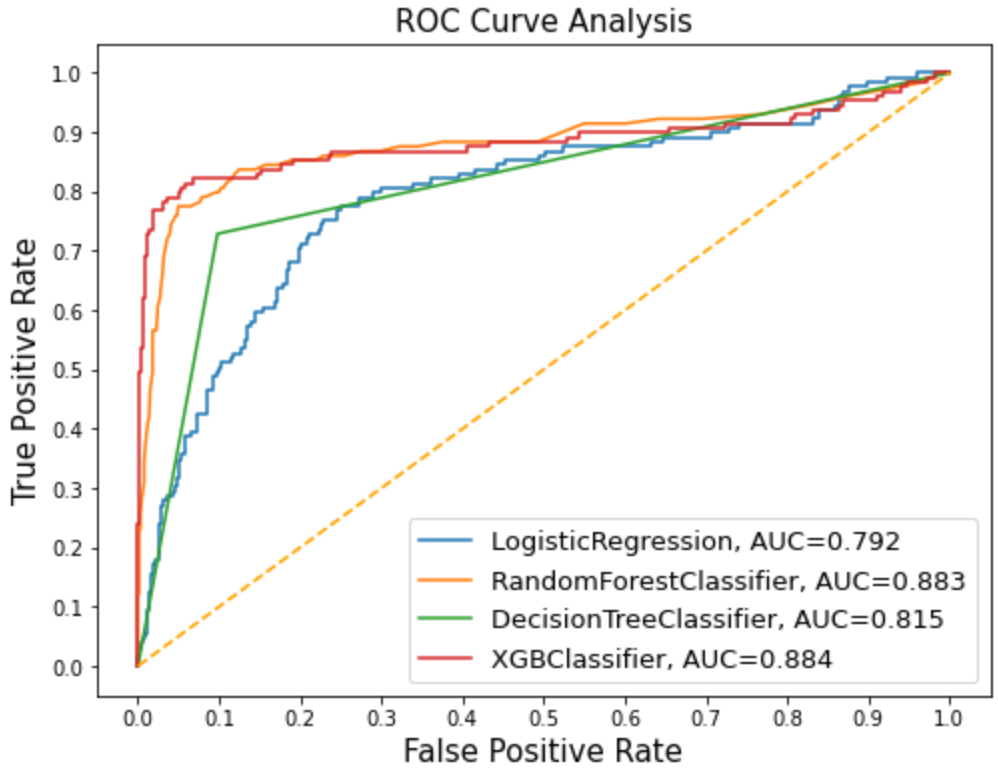
classifiers	recall
LogisticRegression	0.744186
RandomForestClassifier	0.744186
DecisionTreeClassifier	0.728682
XGBClassifier	0.767442

The results table indicates that the XGBoost Classifier achieved the highest recall score, followed by the Random Forest Classifier and Logistic Regression. The Decision Tree Classifier recorded the lowest recall score of 0.73 among all the models evaluated.

Models Comparison - ROC Curve

In [68]:
```
np.random.seed(123)
classifiers = [LogisticRegression(),
               RandomForestClassifier(),
               DecisionTreeClassifier(),
               XGBClassifier()]


# Define a result table as a DataFrame
result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

# Train the models and record the results
for cls in classifiers:
    model = cls.fit(resampled_X_train, resampled_y_train)
    yproba = model.predict_proba(X_test)[::,1]

    fpr, tpr, _ = roc_curve(y_test,  yproba)
    auc = roc_auc_score(y_test, yproba)

    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)

# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)

fig = plt.figure(figsize=(8,6))

for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)
```

```
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)

plt.title('ROC Curve Analysis', fontsize=15)
plt.legend(prop={'size':13}, loc='lower right')

plt.show()
```



The ROC curve analysis demonstrates that the XGBoost Classifier outperforms the other models, followed by the Random Forest, Decision Tree, and Logistic Regression classifiers. XGBoost achieved the highest AUC score of 0.884, while Logistic Regression recorded the lowest at 0.79.

The ROC (Receiver Operating Characteristic) curve plots the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. The AUC (Area Under the Curve) summarizes this performance — a higher AUC indicates a better ability of the model to distinguish between the positive and negative classes.

Model Tuning

Based on the evaluation of the models using recall scores and ROC AUC, it is observed that the XGBoost classifier and the RandomForest classifier have shown the strongest performance. To further improve their performance, hyperparameter tuning will be applied using GridSearch.

Tuning RandomForest

In [69]:
```
#Define the hyperparameter grid
param_grid = {
    "max_depth": [8,15,20],
    "n_estimators":[500,1000],
    "min_samples_split":[5,10,15],
    "criterion":['entropy','gini']
    }
# Create an instance of the RandomForest classifier
rf = RandomForestClassifier(random_state = 123)
# Create GridSearchCV object with the defined parameter grid and scoring metric
grid_search = GridSearchCV(rf, param_grid, cv=3, n_jobs=-1, verbose=False)

#Fit the GridsearchCV object to the training data
grid_search.fit(resampled_X_train,resampled_y_train)

#print the best parameters
print(grid_search.best_params_)
```

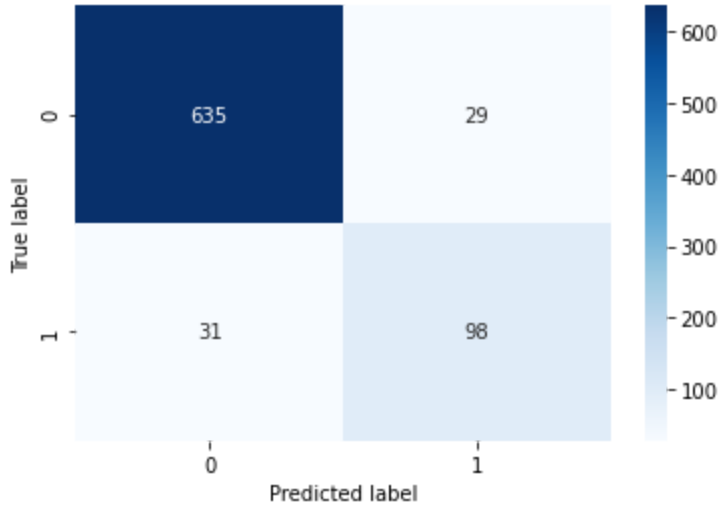
```
{'criterion': 'entropy', 'max_depth': 20, 'min_samples_split': 5, 'n_estimators': 1000}
```

In [70]:
```
# Create an instance of the RandomForest classifier with best params
rf_tuned = RandomForestClassifier(criterion = 'entropy',
                                  max_depth = 20,
                                  min_samples_split = 5,
                                  n_estimators = 1000,
                                  random_state=123)
#Fit the model on the training data
rf_tuned.fit(resampled_X_train, resampled_y_train)
```

Out[70]:
```
RandomForestClassifier(criterion='entropy', max_depth=20, min_samples_split=5,
                       n_estimators=1000, random_state=123)
```

In [71]:
```
#Predict on the test data
y_pred_tuned = rf_tuned.predict(X_test)
```

In [72]:
```
#The confusion matrix
plot_confusion_matrix(y_test, y_pred_tuned, [0,1])
```

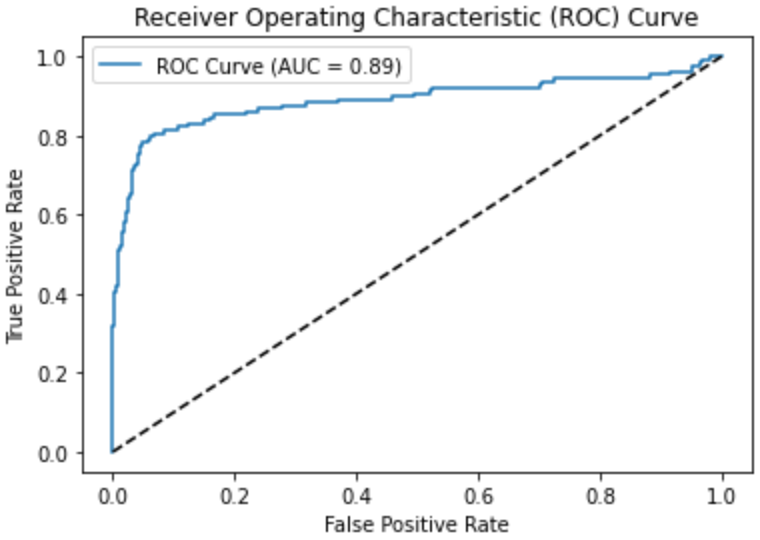


In [73]:
```
#print the classification report
print(classification_report(y_test,y_pred_tuned))
```

```
              precision    recall  f1-score   support

         0.0       0.95      0.96      0.95       664
         1.0       0.77      0.76      0.77       129

    accuracy                           0.92       793
   macro avg       0.86      0.86      0.86       793
weighted avg       0.92      0.92      0.92       793
```

In [74]:
```
# Get the predicted probabilities for the positive class
y_proba = rf_tuned.predict_proba(X_test)[:, 1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Compute the AUC score
auc_score = roc_auc_score(y_test, y_proba)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC Curve (AUC = {:.2f})'.format(auc_score))
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for random classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
```

```
plt.legend()
plt.show()
```



Based on the ROC curve and the recall metric, the **tuned Random Forest model** demonstrates strong performance in distinguishing between churned and non-churned customers. With a **recall score of 0.76**, the model effectively identifies **76% of actual churned customers**, making it a reliable tool for detecting customer attrition and supporting retention strategies.

Tuning XGBoost

In [75]:
```
#Define the hyperparameter grid
param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [1,2,5,10],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100,200],
}
#Create an instance of XGBoost Classifier
xgb = XGBClassifier(random_state = 123)
# Create GridSearchCV object
grid_search = GridSearchCV(xgb, param_grid, cv=3, scoring = 'recall',n_jobs=1)
#Fit the GridsearchCV object to the training data
grid_search.fit(resampled_X_train,resampled_y_train)

#print the best parameters
print(grid_search.best_params_)
```

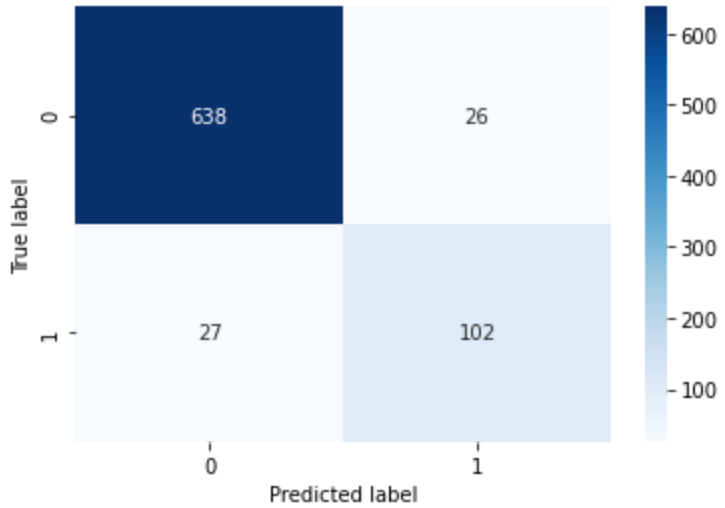
```
{'learning_rate': 0.1, 'max_depth': 10, 'min_child_weight': 1, 'n_estimators': 200, 'subsample': 0.7}
```

In [76]:
```
#Create an instance of XGBoost Classifier with best params
xgb_tuned = XGBClassifier(learning_rate = 0.1,
                          max_depth = 10,
                          min_child_weight = 1,
                          n_estimators = 100,
                          subsample = 0.5,
                          random_state = 123)

#Fit on the training data
xgb_tuned.fit(resampled_X_train,resampled_y_train)
```

Out[76]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
              importance_type='gain', interaction_constraints='',
              learning_rate=0.1, max_delta_step=0, max_depth=10,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=100, n_jobs=0, num_parallel_tree=1, random_state=123,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=0.5,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

In [77]:
```
#Predict on the test data
y_pred_xgt = xgb_tuned.predict(X_test)
```

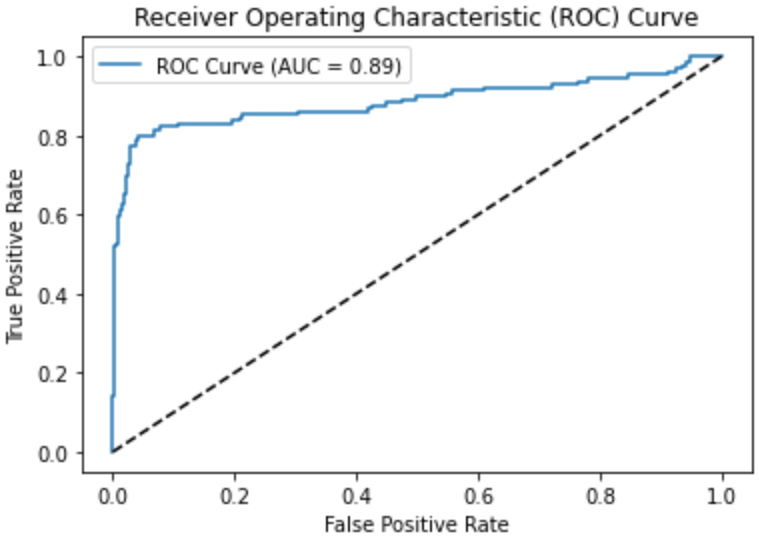
In [78]:
```
#Confusion matrix
plot_confusion_matrix(y_test, y_pred_xgt, [0,1])
```



In [79]:
```
#Classification report
print(classification_report(y_test, y_pred_xgt))
```

```
              precision    recall  f1-score   support

         0.0       0.96      0.96      0.96       664
         1.0       0.80      0.79      0.79       129

    accuracy                           0.93       793
   macro avg       0.88      0.88      0.88       793
weighted avg       0.93      0.93      0.93       793
```

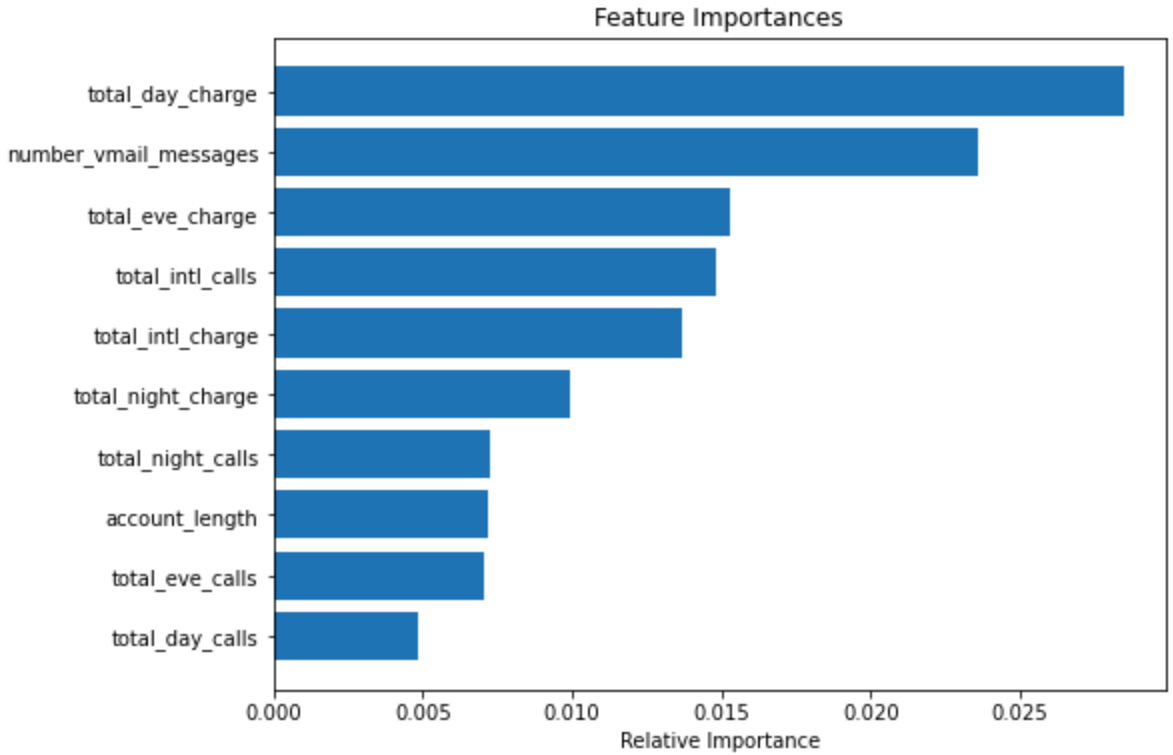
In [80]:
```
# Get the predicted probabilities for the positive class
y_proba = xgb_tuned.predict_proba(X_test)[:, 1]

# Compute the false positive rate, true positive rate, and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_proba)

# Compute the AUC score
auc_score = roc_auc_score(y_test, y_proba)

# Plot the ROC curve
plt.plot(fpr, tpr, label='ROC Curve (AUC = {:.2f})'.format(auc_score))
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line for XGB classifier
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



Based on the ROC curve and the recall metric, the tuned XGBoost model slightly outperforms the Random Forest model in distinguishing between churned and non-churned customers. With a recall score of 0.79, it correctly identifies 79% of actual churned customers, coming very close to our target recall of 0.80. This strong performance highlights XGBoost's effectiveness in predicting customer churn and supporting proactive retention efforts.

Let's outline the most important features

In [81]:
```
feature_names = list(resampled_X_train.columns)
importances = xgb_tuned.feature_importances_[0:10]
indices = np.argsort(importances)

plt.figure(figsize=(8,6))
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], align='center')
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show()
```



Based on the model's feature importance scores, the top three drivers of churn prediction are:
`total_day_charge` , `number_vmail_messages` , `total_eve_charges`

Conclusion.

This project aimed to build a predictive model to identify customers likely to churn, enabling Syriatel to proactively implement customer retention strategies. By following a comprehensive data science workflow — from business understanding to model evaluation — we uncovered valuable insights into customer behavior and developed high-performing classification models.

Our analysis suggests that a machine learning-based approach is viable for identifying customers at churn risk. The fine-tuned XGBoost model stands out with its predictive proficiency.

The key features associated witth churn prediction were total_day_charge, number_vmail_messages and total_eve_charges.

Recommendations.

SyriaTel Company should implement the following:

1. Make use of XGBoost as the primary model for predicting customer churn in a real-time system that flags high-risk customers that are likely to churn.

1. Offer discounts or promotional offers to customers in area code 415 and 510, as these areas have a higher churn rate. This can help incentivize customers to stay with the company.

1. Improve customer service quality and reduce the number of customer service calls by enhancing training programs for customer service representatives to ensure prompt and effective resolution of customer issues, leading to higher customer satisfaction and reduced churn.

1. Evaluate the pricing structure for day, evening, night, and international charges. Consider introducing customized bundles or discounts for these segments to reduce dissatisfaction related to billing.

1. Focus on customer retention strategies in states with higher churn rates, such as Texas, New Jersey, Maryland, Miami, and New York. This can involve targeted marketing campaigns, personalized offers, or improved customer support.

1. Promote Voicemail and International Plans with targeted offers to increase stickiness and engagement.