Syria Tel Customer Churning Analysis

Business Understanding for Syriatel Churn Analysis

Syriatel's Business and the Impact of Emerging Technologies Syriatel is a leading telecommunications company in Syria, providing a range of services including mobile and fixed-line telephony, internet access, and data services. As a state-owned enterprise, Syriatel plays a crucial role in the country's communication infrastructure. Understanding the Problem:

Churn, or customer attrition, is a significant concern for telecommunications companies like Syriatel. When customers leave, it directly impacts revenue, brand reputation, and overall business health. Therefore, understanding the factors driving churn is crucial for developing effective retention strategies.

Understanding the Problem:

The goal is to predict which Syriatel customers are likely to churn and identify the underlying reasons for this behavior. This information can be used to develop targeted retention strategies and improve overall customer satisfaction.

Key Questions and Corresponding Classification Tasks:

- Which customer segments are most likely to churn?

Classification Task: Predict whether a customer will churn (1) or not (0). Features: Customer demographics (age, gender, income, location), service usage (call minutes, data usage, SMS), contract terms, and other relevant factors.

- What are the primary reasons for customer churn?

Classification Task: Predict the reason for churn from a set of potential causes (e.g., poor customer service, high prices, network issues). Features: Customer interactions, service quality metrics, and other relevant indicators. How can Syriatel improve customer satisfaction and loyalty?

Classification Task: Predict customer satisfaction or loyalty levels based on various factors. Features: Customer feedback, service usage patterns, and other relevant metrics.

- What is the cost of customer churn to Syriatel?

While not directly a classification task, this question can be addressed using regression analysis to predict the financial impact of churn. Features: Customer value, revenue generated, and other relevant financial metrics. Potential Classification Models:

Key Business Questions:

What are the primary reasons for customer churn at Syriatel? Identifying the root causes will enable targeted interventions. Which customer segments are most likely to churn? Understanding the characteristics of high-churn segments will help prioritize retention efforts. How can Syriatel improve customer satisfaction and loyalty to reduce churn? Identifying areas for improvement will inform strategic initiatives. What is the cost of customer churn to Syriatel? Quantifying the financial impact will help justify investments in retention programs. Potential Factors Influencing Churn:

Customer demographics: Age, gender, income level, location Service usage: Call minutes, data usage, SMS, international roaming Contract terms: Contract length, pricing plans, promotional offers Customer satisfaction: Customer support interactions, billing issues, network quality Competitive landscape: Offers from rival telecom companies Economic factors: Inflation, unemployment rates

Business Objectives:

- Reduce customer churn rate: Implement strategies to retain existing customers.
- Improve customer satisfaction: Enhance the overall customer experience.
- Optimize pricing and promotional offers: Develop pricing plans that meet customer needs and incentivize loyalty.
- Enhance customer support: Provide timely and effective assistance to address customer concerns.
- Identify at-risk customers: Proactively reach out to customers who are likely to churn.

By addressing these questions and objectives through data analysis, Syriatel can gain valuable insights into customer behavior, develop effective retention strategies, and improve overall business performance.

Retrieving Data

In [6]:
```
import pandas as pd
import os
import opendatasets as od
```

In [7]:
```
# Assign the Kaggle data set URL into variable
dataset = 'https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset'
# Using opendatasets let's download the data sets
od.download(dataset)
```

```
Skipping, found downloaded files in ".\churn-in-telecoms-dataset" (use force=True to for
ce download)
```

In [8]:
```
data_dir = '.\churn-in-telecoms-dataset'
os.listdir(data_dir)
```

Out[8]:
```
['bigml_59c28831336c6604c800002a.csv']
```

Importing relevant libraries

In [9]:
```
import numpy as np
import seaborn as sns
```

```
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import math
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import SMOTE
from sklearn.svm import SVC
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_curve, auc
from sklearn.metrics import classification_report, confusion_matrix, ConfusionMatrixDis
from sklearn.feature_selection import RFECV

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

In [10]:
```
import pandas as pd
file_path = os.path.join(data_dir, 'bigml_59c28831336c6604c800002a.csv')
df = pd.read_csv(file_path)
```

Data Understanding

The SyriaTel dataset from Kaggle is composed of a comprehensive set of customer features, providing multifaceted information about customer usage behaviors, preferences, and interactions.

The dataset contains 3333 entries and 21 columns. The total memory usage of the dataset is approximately 524.2 KB. The columns represent various customer attributes, including state, account length, area code, phone number, international plan, voice mail plan, number of voice mail messages, call durations and charges for different time periods and international calls, customer service calls, and churn status. The dataset does not have any missing values, as indicated by the non-null counts. The data types of the columns include bool, float64, int64, and object. The bool column represents the churn status, indicating whether a customer discontinued the service (True) or not (False). The float64 columns represent numerical values for call durations and charges. The int64 columns represent numerical values for account length, area code, number of voice mail messages, call counts, and customer service calls. The object columns include state, phone number, international plan, and voice mail plan, which are categorical variables. By understanding these features and their implications, we can conduct in-depth analyses and predictive modeling to tackle the issue of customer churn.

In [11]:
```
## checking top details of data set
df.head()
```

Out[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	tc 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 [12]:
```
## checking details at bootom of data set
df.tail()
```

Out[12]:

	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
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82

5 rows × 21 columns

In [13]:
```
# displaying data set information
df.info
```

Out[13]:
```
<bound method DataFrame.info of      state  account length  area code phone number inter
national plan  \
0       KS             128        415     382-4657                no
1       OH             107        415     371-7191                no
2       NJ             137        415     358-1921                no
3       OH              84        408     375-9999               yes
4       OK              75        415     330-6626               yes
...    ...             ...        ...          ...               ...
3328    AZ             192        415     414-4276                no
3329    WV              68        415     370-3271                no
```

```
3330    RI             28        510     328-8230                no
3331    CT            184        510     364-6381               yes
3332    TN             74        415     400-4344                no

     voice mail plan  number vmail messages  total day minutes  \
0                yes                     25              265.1
1                yes                     26              161.6
2                 no                      0              243.4
3                 no                      0              299.4
4                 no                      0              166.7
...              ...                    ...                ...
3328             yes                     36              156.2
3329              no                      0              231.1
3330              no                      0              180.8
3331              no                      0              213.8
3332             yes                     25              234.4

      total day calls  total day charge  ...  total eve calls  \
0                 110             45.07  ...               99
1                 123             27.47  ...              103
2                 114             41.38  ...              110
3                  71             50.90  ...               88
4                 113             28.34  ...              122
...               ...               ...  ...              ...
3328               77             26.55  ...              126
3329               57             39.29  ...               55
3330              109             30.74  ...               58
3331              105             36.35  ...               84
3332              113             39.85  ...               82

      total eve charge  total night minutes  total night calls  \
0                16.78                244.7                 91
1                16.62                254.4                103
2                10.30                162.6                104
3                 5.26                196.9                 89
4                12.61                186.9                121
...                ...                  ...                ...
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
0                  11.01                10.0                 3
1                  11.45                13.7                 3
2                   7.32                12.2                 5
3                   8.86                 6.6                 7
4                   8.41                10.1                 3
...                  ...                 ...               ...
3328               12.56                 9.9                 6
3329                8.61                 9.6                 4
3330                8.64                14.1                 6
3331                6.26                 5.0                10
3332               10.86                13.7                 4

      total intl charge  customer service calls  churn
0                  2.70                       1  False
1                  3.70                       1  False
2                  3.29                       0  False
3                  1.78                       2  False
4                  2.73                       3  False
...                 ...                     ...    ...
3328               2.67                       2  False
3329               2.59                       3  False
```

```
3330              3.81                      2  False
3331              1.35                      2  False
3332              3.70                      0  False

[3333 rows x 21 columns]>
```

In [14]:
```
#checking columns
df.columns
```

Out[14]:
```
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 [15]:
```
# Checking shape of data set
df.shape
```

Out[15]:
```
(3333, 21)
```

In [16]:
```
# checking data types and establishing if null values exist
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 [17]:
```
pd.set_option('display.max_columns',None)
df = pd.read_csv(file_path)
df.head()
```

Out[17]:

	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	tota eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122

In [18]:
```
# checking for null values
df.isna().sum()
```

Out[18]:
```
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
```

In [19]:
```
df.describe
```

Out[19]:
```
<bound method NDFrame.describe of      state  account length  area code phone number int
ernational plan  \
0       KS              128        415     382-4657                no
1       OH              107        415     371-7191                no
2       NJ              137        415     358-1921                no
3       OH               84        408     375-9999               yes
4       OK               75        415     330-6626               yes
...    ...              ...        ...          ...               ...
3328    AZ              192        415     414-4276                no
3329    WV               68        415     370-3271                no
3330    RI               28        510     328-8230                no
3331    CT              184        510     364-6381               yes
3332    TN               74        415     400-4344                no

     voice mail plan  number vmail messages  total day minutes  \
```

```
0               yes                      25              265.1
1               yes                      26              161.6
2                no                       0              243.4
3                no                       0              299.4
4                no                       0              166.7
...             ...                     ...                ...
3328            yes                      36              156.2
3329             no                       0              231.1
3330             no                       0              180.8
3331             no                       0              213.8
3332            yes                      25              234.4

      total day calls  total day charge  total eve minutes  total eve calls  \
0                 110             45.07              197.4               99
1                 123             27.47              195.5              103
2                 114             41.38              121.2              110
3                  71             50.90               61.9               88
4                 113             28.34              148.3              122
...               ...               ...                ...              ...
3328               77             26.55              215.5              126
3329               57             39.29              153.4               55
3330              109             30.74              288.8               58
3331              105             36.35              159.6               84
3332              113             39.85              265.9               82

      total eve charge  total night minutes  total night calls  \
0                16.78                244.7                 91
1                16.62                254.4                103
2                10.30                162.6                104
3                 5.26                196.9                 89
4                12.61                186.9                121
...                ...                  ...                ...
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
0                  11.01                10.0                 3
1                  11.45                13.7                 3
2                   7.32                12.2                 5
3                   8.86                 6.6                 7
4                   8.41                10.1                 3
...                  ...                 ...               ...
3328               12.56                 9.9                 6
3329                8.61                 9.6                 4
3330                8.64                14.1                 6
3331                6.26                 5.0                10
3332               10.86                13.7                 4

      total intl charge  customer service calls  churn
0                  2.70                       1  False
1                  3.70                       1  False
2                  3.29                       0  False
3                  1.78                       2  False
4                  2.73                       3  False
...                 ...                     ...    ...
3328               2.67                       2  False
3329               2.59                       3  False
3330               3.81                       2  False
3331               1.35                       2  False
3332               3.70                       0  False

[3333 rows x 21 columns]>
```

In [20]:
```
df.churn
```

Out[20]:
```
0       False
1       False
2       False
3       False
4       False
        ...
3328    False
3329    False
3330    False
3331    False
3332    False
Name: churn, Length: 3333, dtype: bool
```

In [21]:
```
churn = df['churn']
churn = []

for i in churn:
    churn.append(int(i))
```

In [22]:
```
# converting bool to interger
df['churn'] = df['churn'].astype(int)
df.churn
```

Out[22]:
```
0       0
1       0
2       0
3       0
4       0
       ..
3328    0
3329    0
3330    0
3331    0
3332    0
Name: churn, Length: 3333, dtype: int32
```

Feature Engineering

In [23]:
```
# Creating new features; Total charges, Total talktime, Total Calls and Average call du
df["Total charge"] = df[['total day charge', 'total eve charge', 'total night charge',
df["Total Talk time"] = df[['total day minutes', 'total eve minutes', 'total night minu
df["Total calls"] = df[['total day calls', 'total eve calls', 'total night calls', 'tot
df["Avg Call duration"]= df["Total Talk time"] / df["Total calls"]
```

In [24]:
```
# confirming addition of new features
df.head()
```

Out[24]:

	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	tota eve call
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	9

	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	tota eve calls
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122

In [25]:
```
# understanding churn distribution
df["churn"].value_counts()
```

Out[25]:
```
0    2850
1     483
Name: churn, dtype: int64
```

In [26]:
```
# Creating a voice message to call ratio for each customer
df["voice_ms_call_ratio"] = df["number vmail messages"]/ df["Total calls"]
```

In [27]:
```
# Creating columns for charges per call for night, day, evening and international calls

df["charge_per_call_night"] = df["total night charge"] / df["total night minutes"]
df["charge_per_call_day"] = df["total day charge"] / df["total day minutes"]
df["charge_per_call_eve"]= df["total eve charge"] / df["total eve minutes"]
df["charge_per_call_intl"] = df["total intl charge"] / df["total intl minutes"]
```

In [28]:
```
#confirming addition of features
df.columns
```

Out[28]:
```
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', 'Total charge', 'Total Talk time',
       'Total calls', 'Avg Call duration', 'voice_ms_call_ratio',
       'charge_per_call_night', 'charge_per_call_day', 'charge_per_call_eve',
       'charge_per_call_intl'],
      dtype='object')
```

In [29]:
```
#checking information of top of data set
df.head()
```

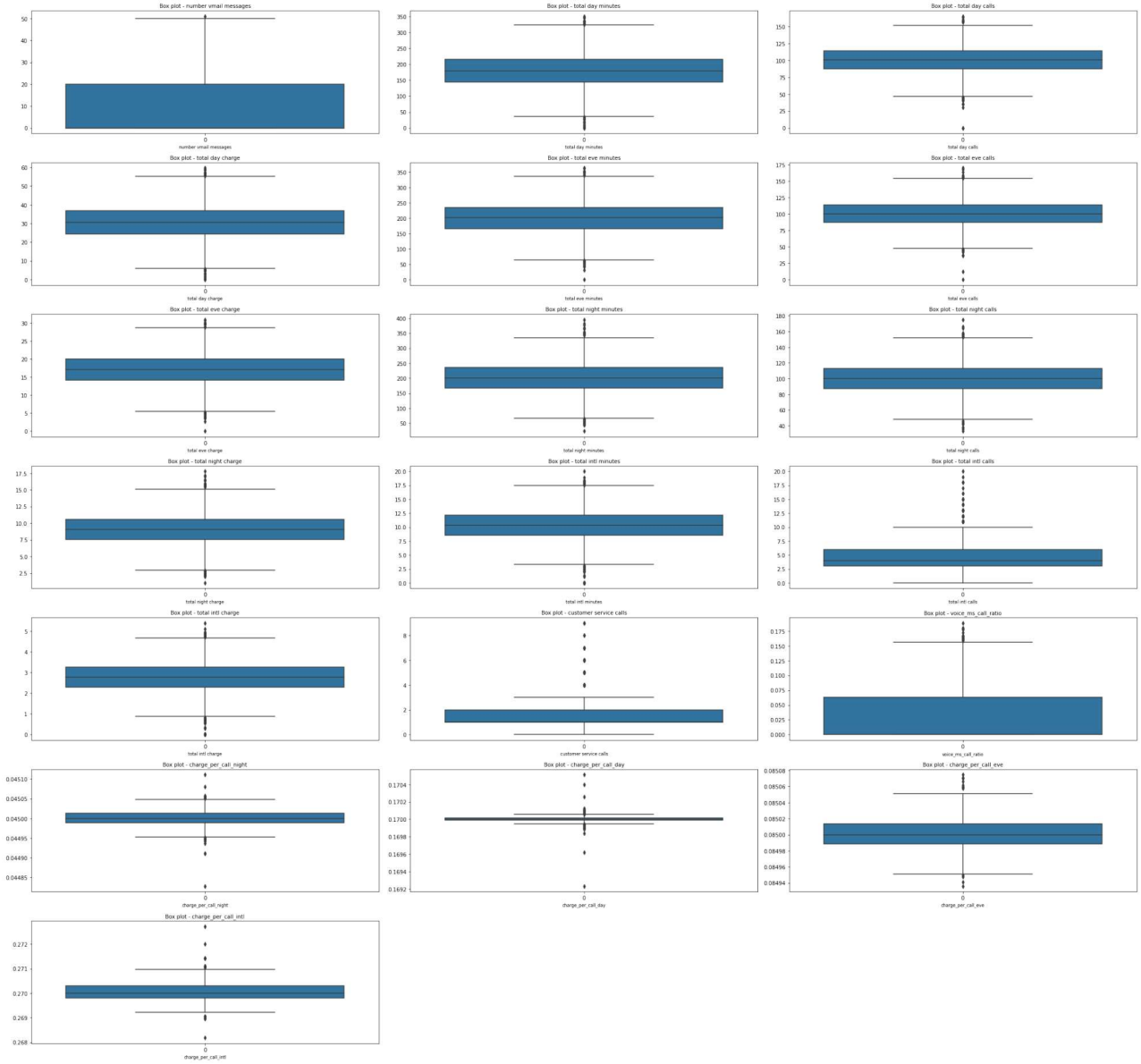
Out[29]:

	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	tota eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	197.4	99

	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	tota eve call
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	195.5	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	121.2	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	61.9	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	148.3	122

Checking for outliers

In [30]:
```
# Columns for our box plots
columns = ['number vmail messages', 'total day minutes', 'total day calls', 'total day
           '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', 'voice_ms_call_ratio','charge_per_call_night',
           'charge_per_call_day', 'charge_per_call_eve', 'charge_per_call_intl']

# Calculate the required number of rows and columns for subplots
num_rows = (len(columns) - 1) // 3 + 1
num_cols = min(len(columns), 3)

# Create the subplots
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10*num_cols, 4*num_rows))

# Generate box plots for each column
for i, column in enumerate(columns):
    row = i // num_cols
    col = i % num_cols
    sns.boxplot(data=df[column], ax=axes[row, col])
    axes[row, col].set_title(f'Box plot - {column}', fontsize=10)
    axes[row, col].set_xlabel(column, fontsize=8)

# Remove any empty subplots
if i < (num_rows * num_cols) - 1:
    for j in range(i + 1, num_rows * num_cols):
        fig.delaxes(axes.flatten()[j])

plt.tight_layout()
plt.show()
```



In [31]:
```
from scipy.stats import skew, kurtosis

def descriptive_stats(df, columns):
    stats = df[columns].agg(['mean', 'median', 'std', 'min', 'max']).T
    stats['variance'] = df[columns].var()
    stats['skewness'] = df[columns].apply(lambda x: skew(x.dropna()))
    stats['kurtosis'] = df[columns].apply(lambda x: kurtosis(x.dropna()))
    return stats
columns_to_analyze = [
    'total day minutes', 'total day charge', 'total eve minutes',
    'total eve charge', 'total night minutes', 'total night charge',
    'total intl minutes', 'total intl charge', 'number vmail messages',
    'customer service calls'
]

# Compute descriptive statistics
stats_df = descriptive_stats(df, columns_to_analyze)


print(stats_df)
```

```
                              mean  median        std    min     max  \
total day minutes       179.775098  179.40  54.467389   0.00  350.80
```

```
total day charge         30.562307   30.50   9.259435   0.00   59.64
total eve minutes       200.980348  201.40  50.713844   0.00  363.70
total eve charge         17.083540   17.12   4.310668   0.00   30.91
total night minutes     200.872037  201.20  50.573847  23.20  395.00
total night charge        9.039325    9.05   2.275873   1.04   17.77
total intl minutes       10.237294   10.30   2.791840   0.00   20.00
total intl charge         2.764581    2.78   0.753773   0.00    5.40
number vmail messages     8.099010    0.00  13.688365   0.00   51.00
customer service calls    1.562856    1.00   1.315491   0.00    9.00

                           variance  skewness  kurtosis
total day minutes       2966.696487 -0.029064 -0.021710
total day charge          85.737128 -0.029070 -0.021582
total eve minutes       2571.894016 -0.023867  0.023792
total eve charge          18.581856 -0.023847  0.023650
total night minutes     2557.714002  0.008917  0.083888
total night charge         5.179597  0.008882  0.083735
total intl minutes         7.794368 -0.245026  0.606472
total intl charge          0.568173 -0.245176  0.606897
number vmail messages    187.371347  1.264254 -0.052852
customer service calls     1.730517  1.090868  1.726518
```

Key Findings:

Average Usage:

Customers on average use a significant amount of minutes for day, evening, and night calls. International calls are less frequent but still contribute to overall usage. The average number of voicemail messages and customer service calls is relatively low. Distribution of Usage:

The standard deviation (std) for most metrics is relatively high, indicating a wide range of usage patterns among customers. The skewness and kurtosis values suggest that the distributions of some metrics (e.g., total intl minutes) are not perfectly symmetrical or normally distributed. Financial Metrics:

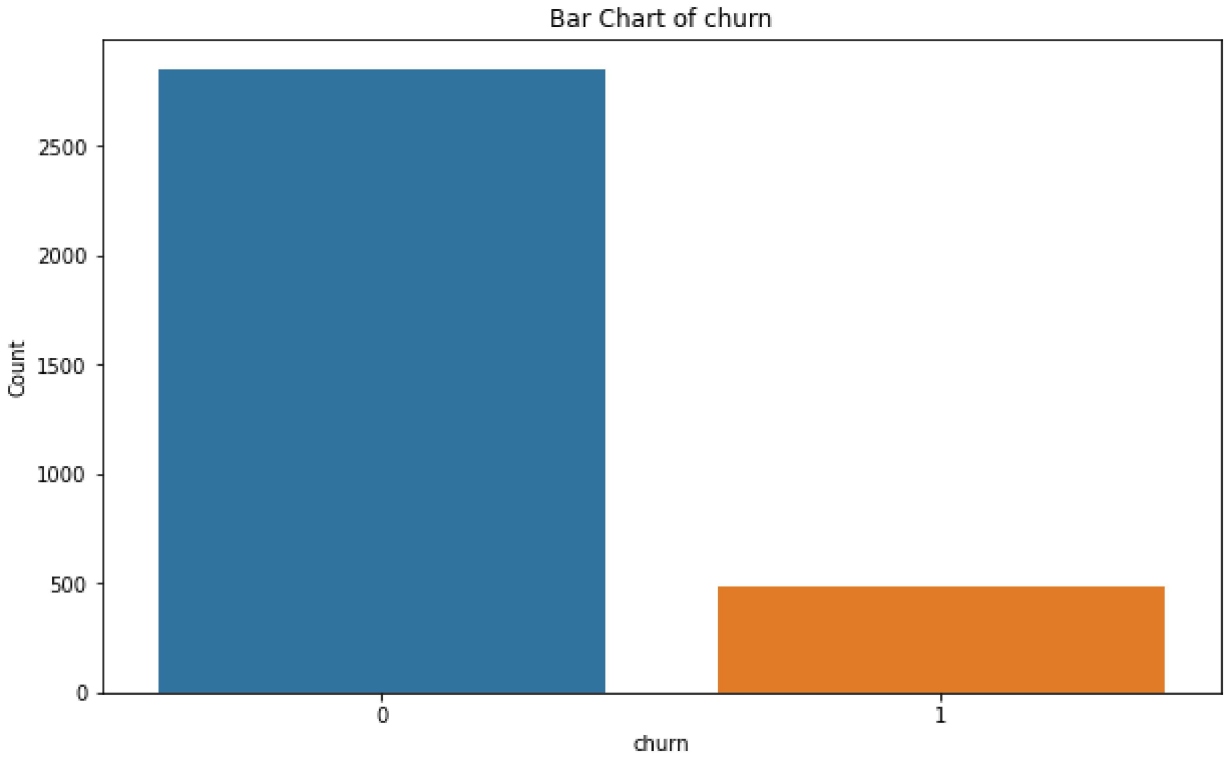
The total day, evening, and night charges align with the corresponding minute usage. International charges are generally lower than domestic charges.

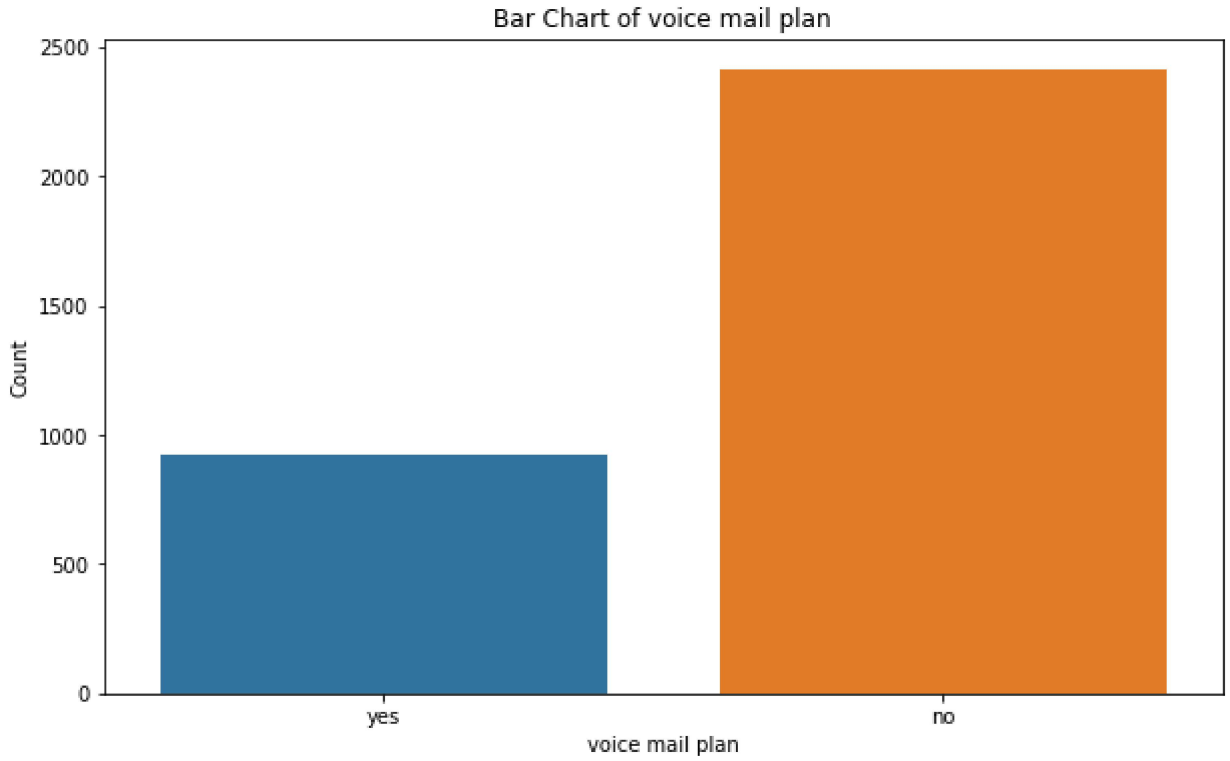
Explorative Data Analysis

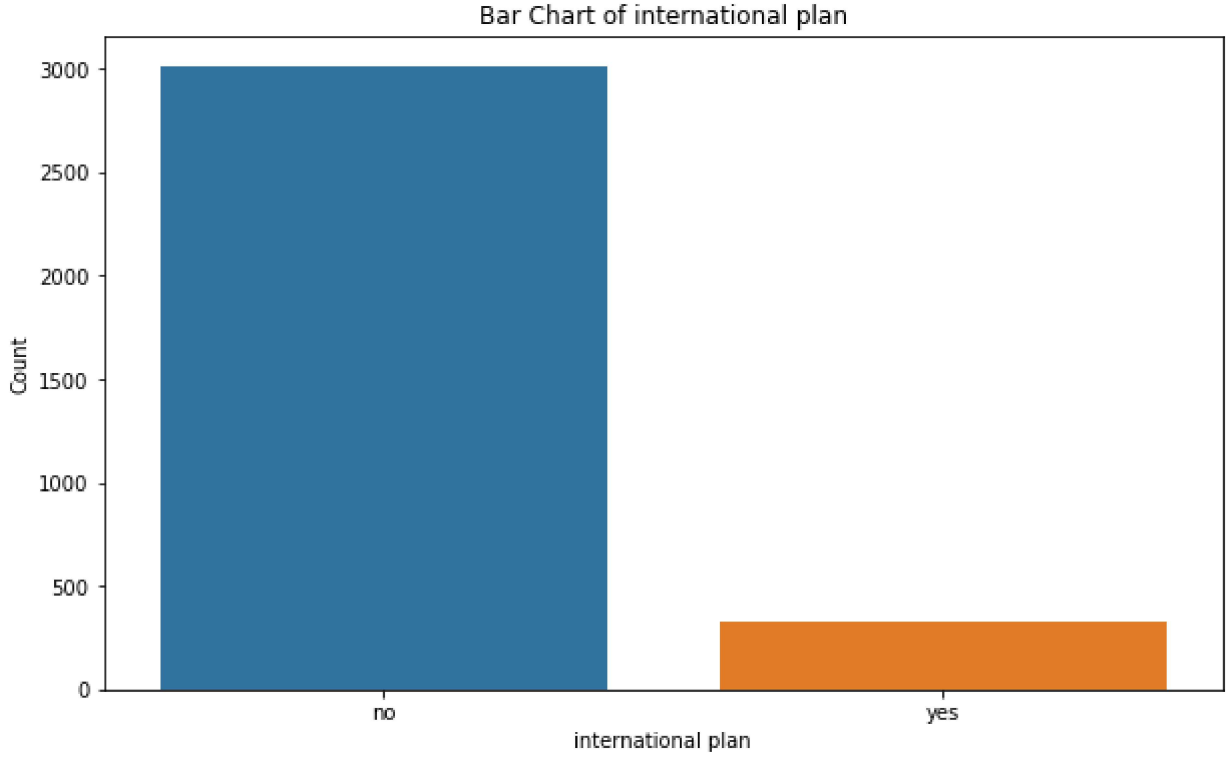
In [32]:
```
# Frequency Counts for Categorical Variables
def frequency_counts(df, columns):
    for col in columns:
        print(f"\nFrequency Counts for {col}:")
        print(df[col].value_counts())

# Plot Bar Charts for Categorical Variables
def plot_bar_charts(df, columns):
    for col in columns:
        if df[col].dtype == 'object' or len(df[col].unique()) < 20:
            plt.figure(figsize=(10, 6))
            sns.countplot(x=df[col])
            plt.title(f'Bar Chart of {col}')
            plt.xlabel(col)
            plt.ylabel('Count')
            plt.show()
```

```
categorical_vars = ['churn', 'voice mail plan', 'international plan']

# Display frequency counts
frequency_counts(df, categorical_vars)

# Plot bar charts
plot_bar_charts(df, categorical_vars)
```

```
Frequency Counts for churn:
0    2850
1     483
Name: churn, dtype: int64

Frequency Counts for voice mail plan:
no     2411
yes     922
Name: voice mail plan, dtype: int64

Frequency Counts for international plan:
no     3010
yes     323
Name: international plan, dtype: int64
```







The provided data shows the frequency counts for three categorical variables: churn, voice mail plan, and international plan.

Key Findings:

Churn:

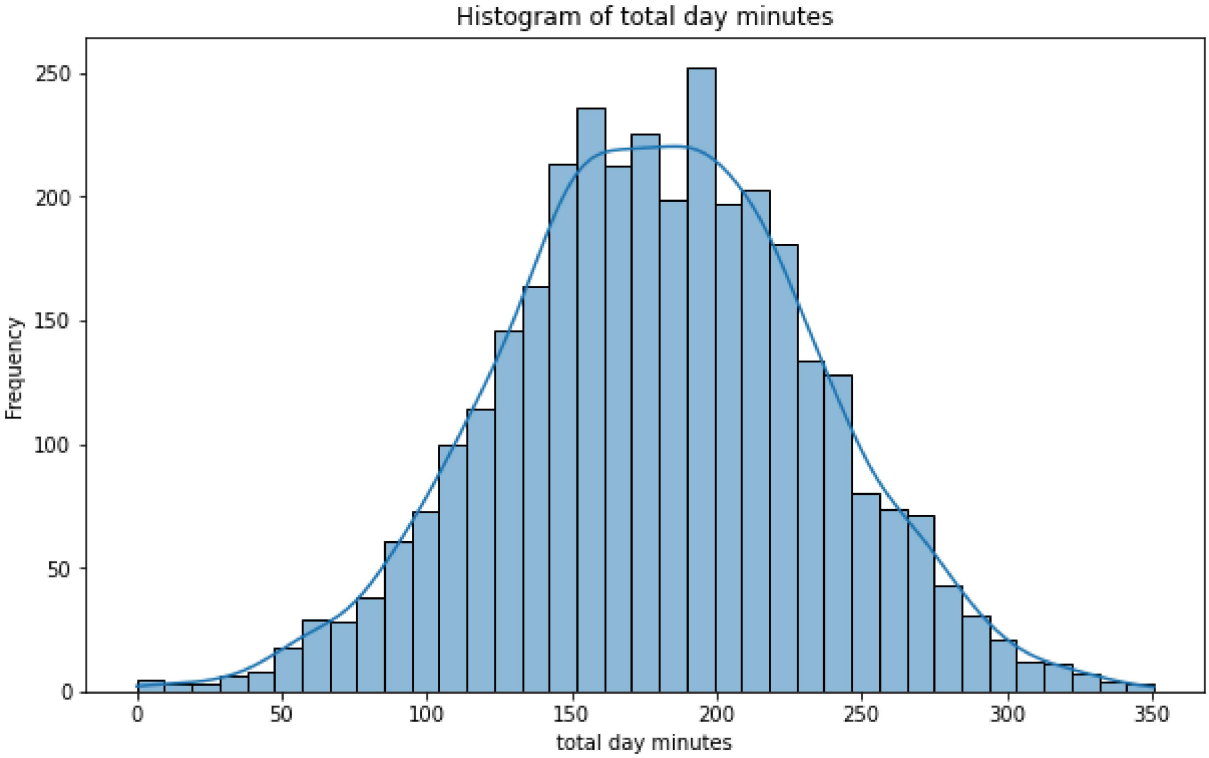
The majority of customers (28,501) did not churn, while 483 did. Churn rate: 483 / (28501 + 483) ≈ 1.67% Voice Mail Plan:

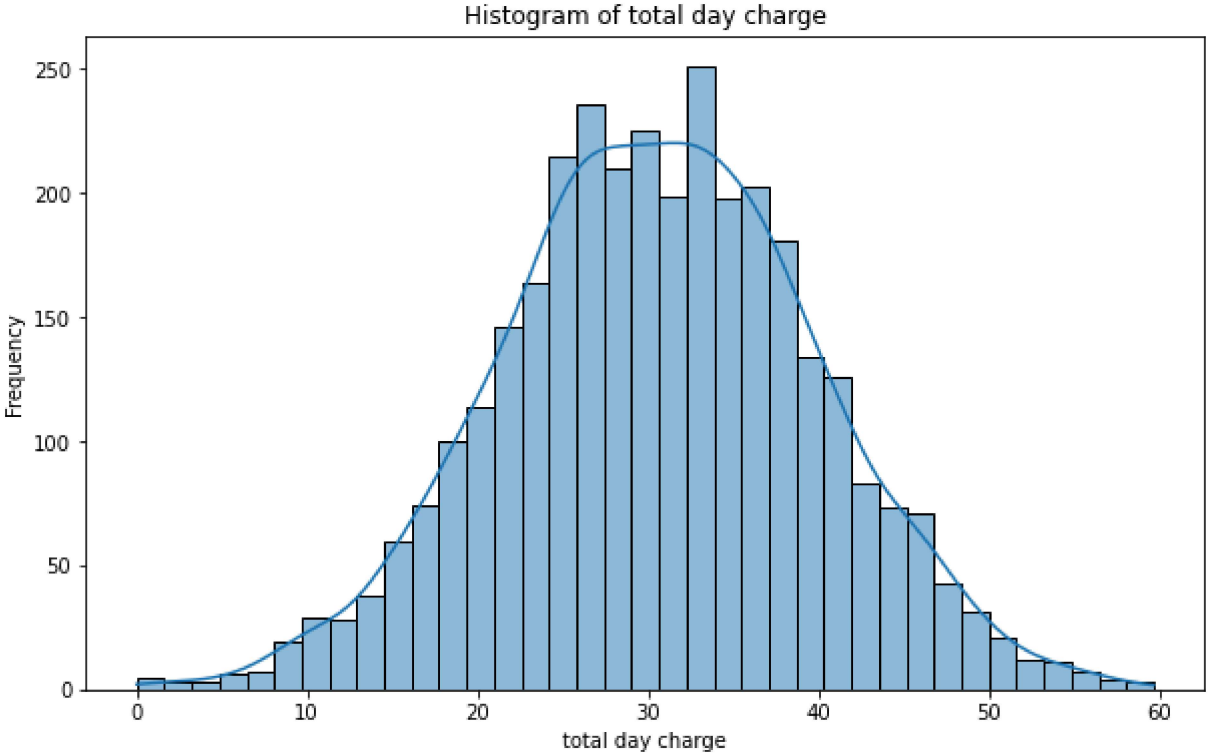
Most customers (2411) do not have a voice mail plan, while 922 do. Voice mail plan adoption rate: 922 / (2411 + 922) ≈ 27.59% International Plan:

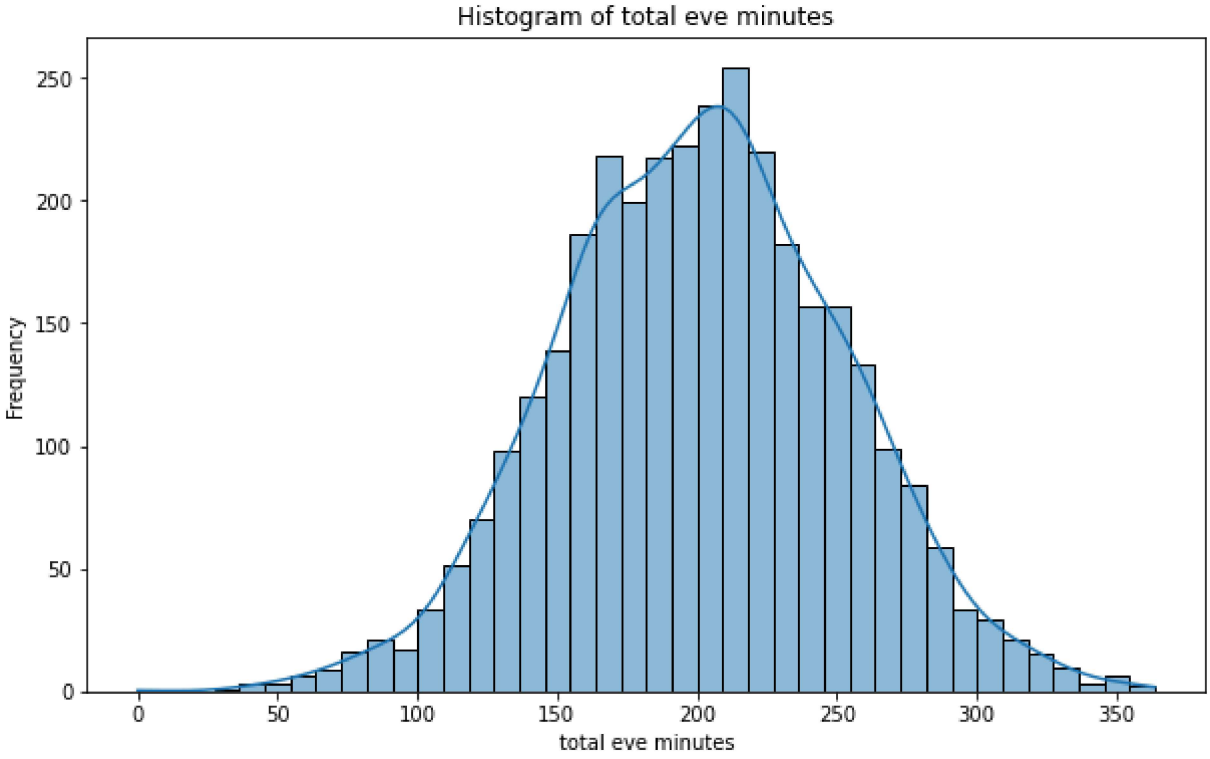
The vast majority of customers (3010) do not have an international plan, while 323 do. International plan adoption rate: 323 / (3010 + 323) ≈ 9.65% Potential Insights:

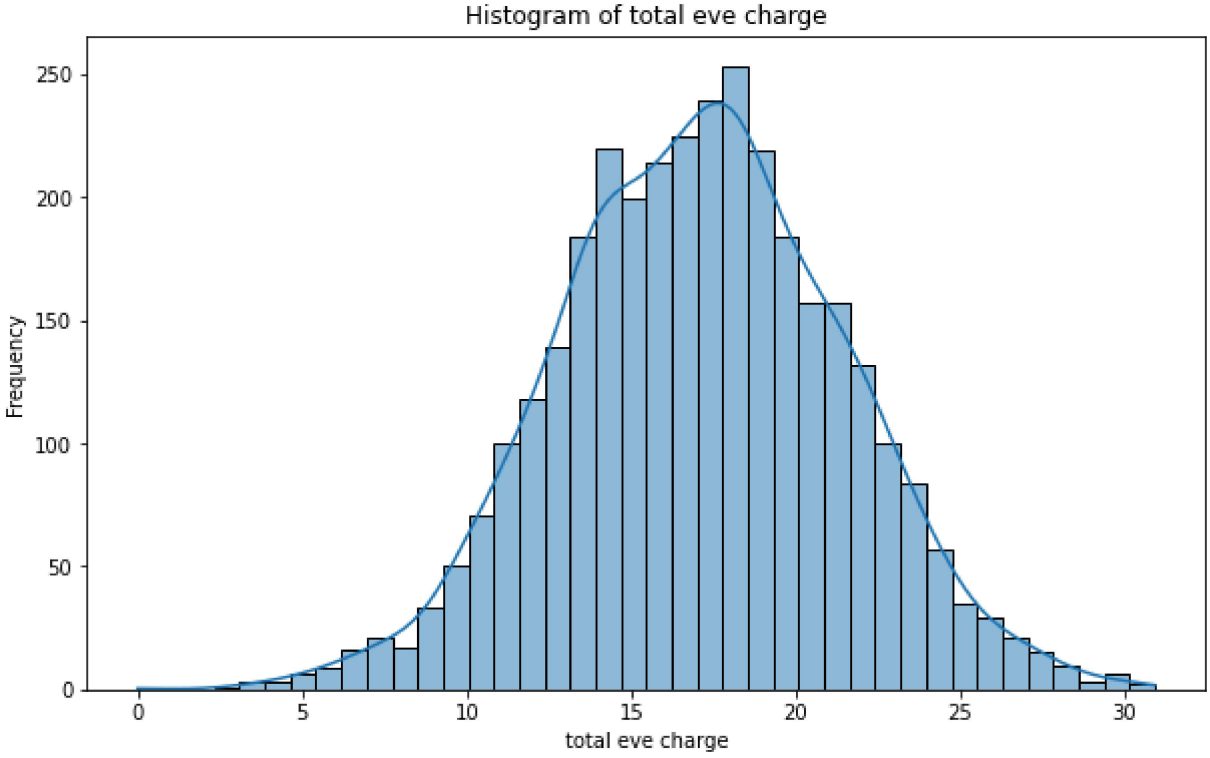
Churn is relatively low: Only 1.67% of customers churned. Voice mail plan adoption is higher: 27.59% of customers have a voice mail plan. International plan adoption is lower: Only 9.65% of customers have an international plan.

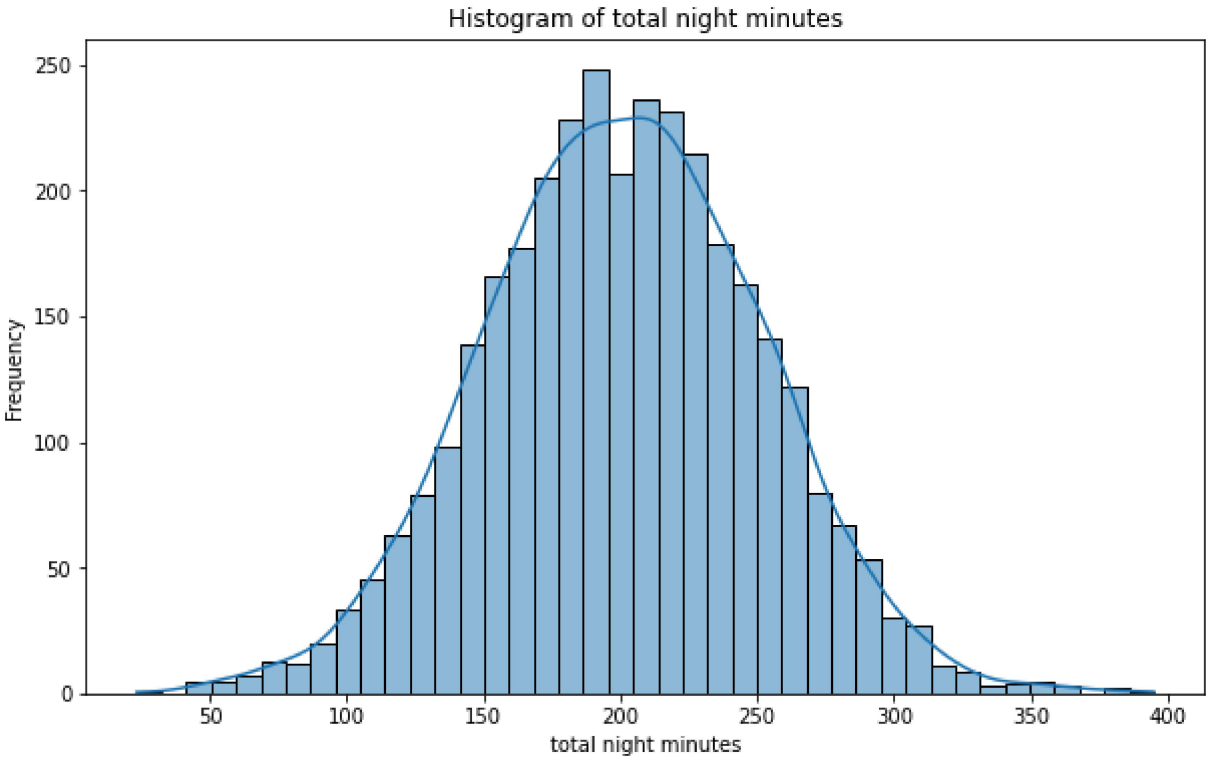
In [33]:
```
## univariate analysis
def plot_histograms(df, columns):
    for col in columns:
        plt.figure(figsize=(10, 6))
        sns.histplot(df[col], kde=True)
        plt.title(f'Histogram of {col}')
        plt.xlabel(col)
        plt.ylabel('Frequency')
        plt.show()

def plot_boxplots(df, columns):
    for col in columns:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x=df[col])
        plt.title(f'Box Plot of {col}')
        plt.xlabel(col)
        plt.show()

def plot_bar_charts(df, columns):
    for col in columns:
        if df[col].dtype == 'object' or len(df[col].unique()) < 20:
            plt.figure(figsize=(10, 6))
            sns.countplot(x=df[col])
            plt.title(f'Bar Chart of {col}')
            plt.xlabel(col)
            plt.ylabel('Count')
            plt.show()

# Plot histograms for continuous variables
continuous_vars = [
    'total day minutes', 'total day charge', 'total eve minutes',
    'total eve charge', 'total night minutes', 'total night charge',
    'total intl minutes', 'total intl charge'
]

plot_histograms(df, continuous_vars)

# Plot box plots for continuous variables
plot_boxplots(df, continuous_vars)

# Plot bar charts for categorical variables
categorical_vars = ['number vmail messages', 'customer service calls']
plot_bar_charts(df, categorical_vars)
```

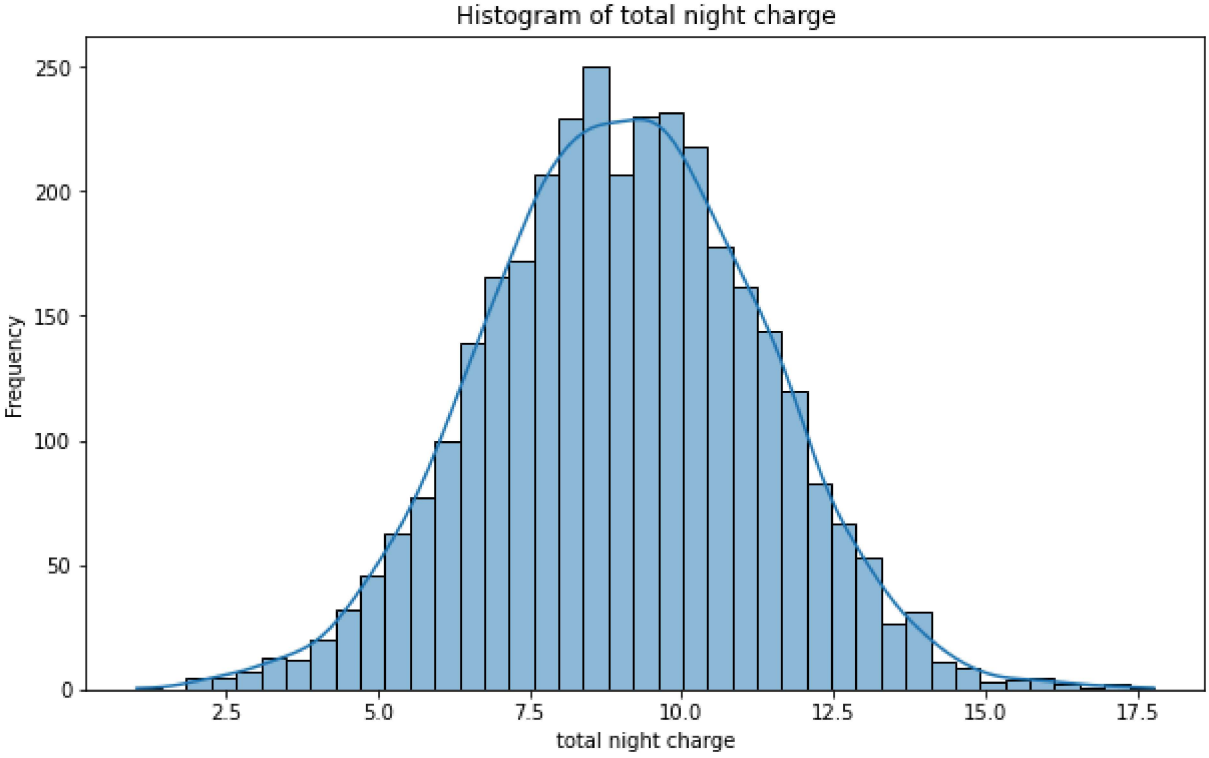

Histogram of total day minutes
Frequency
total day minutes
0
50
100
150
200
250
300
350

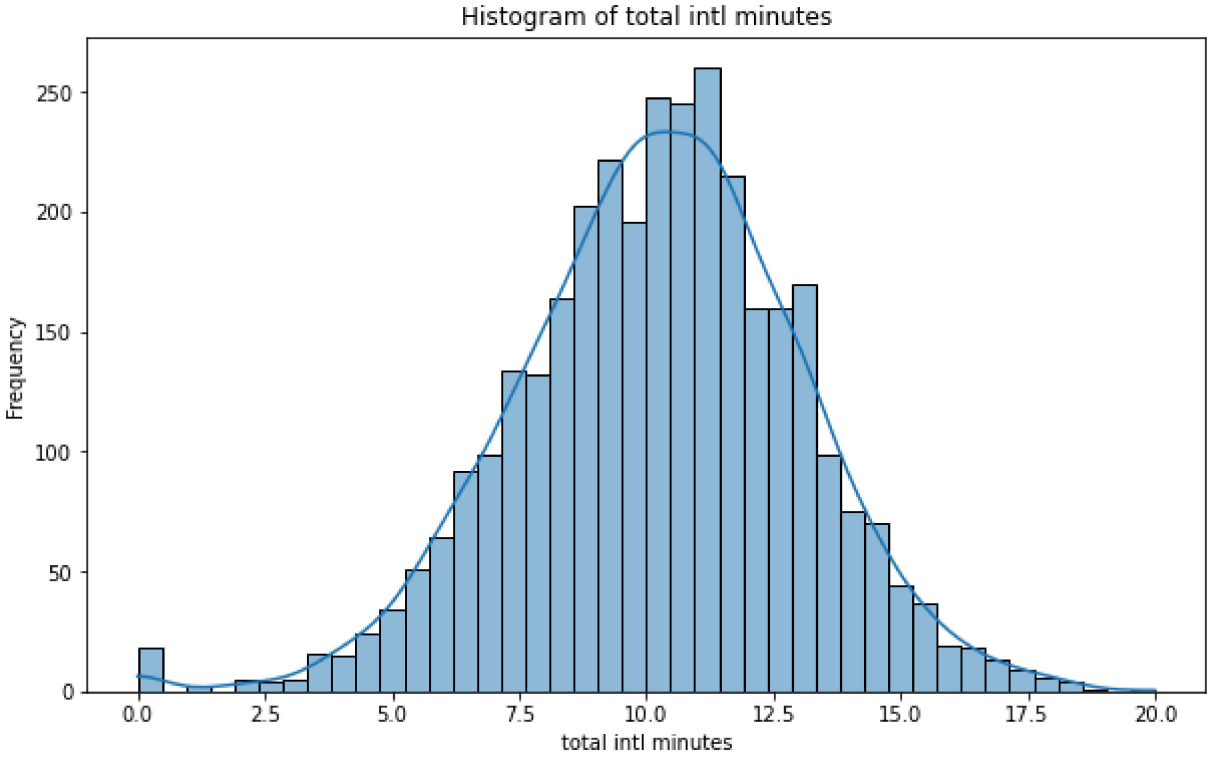

Histogram of total day charge
Frequency
total day charge
0
10
20
30
40
50
60

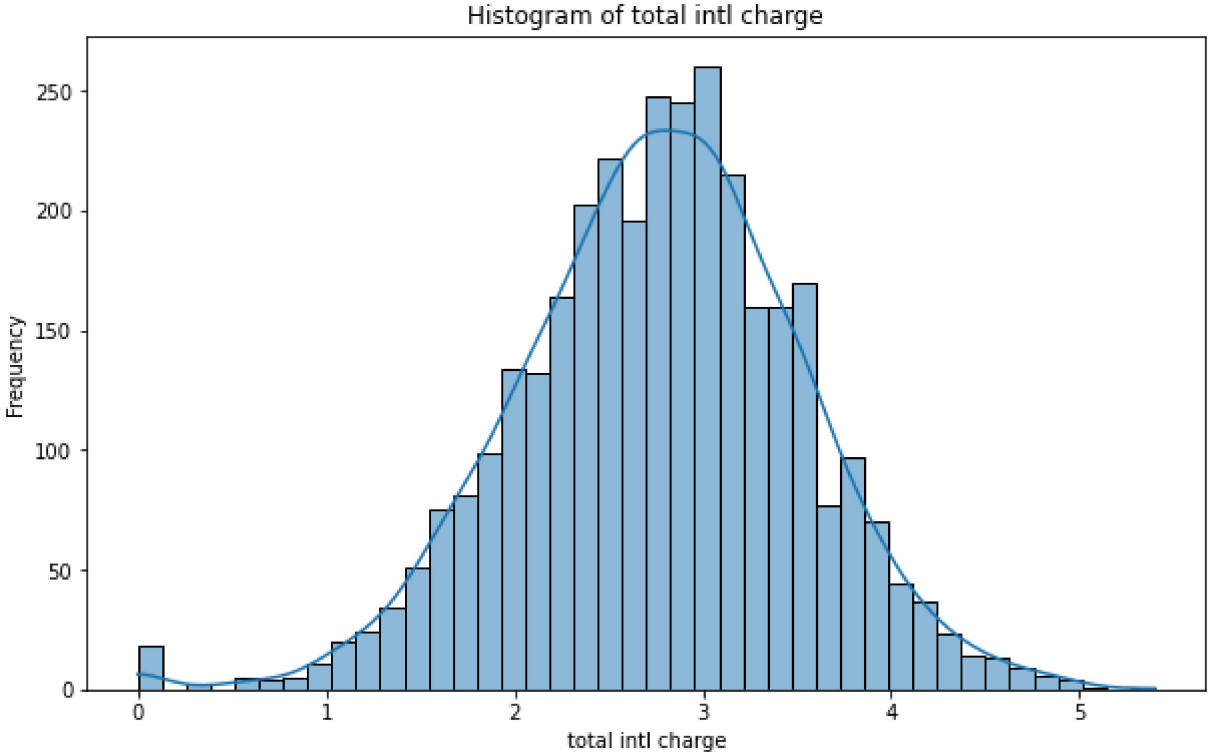

Histogram of total eve minutes
Frequency
total eve minutes
0
50
100
150
200
250
300
350

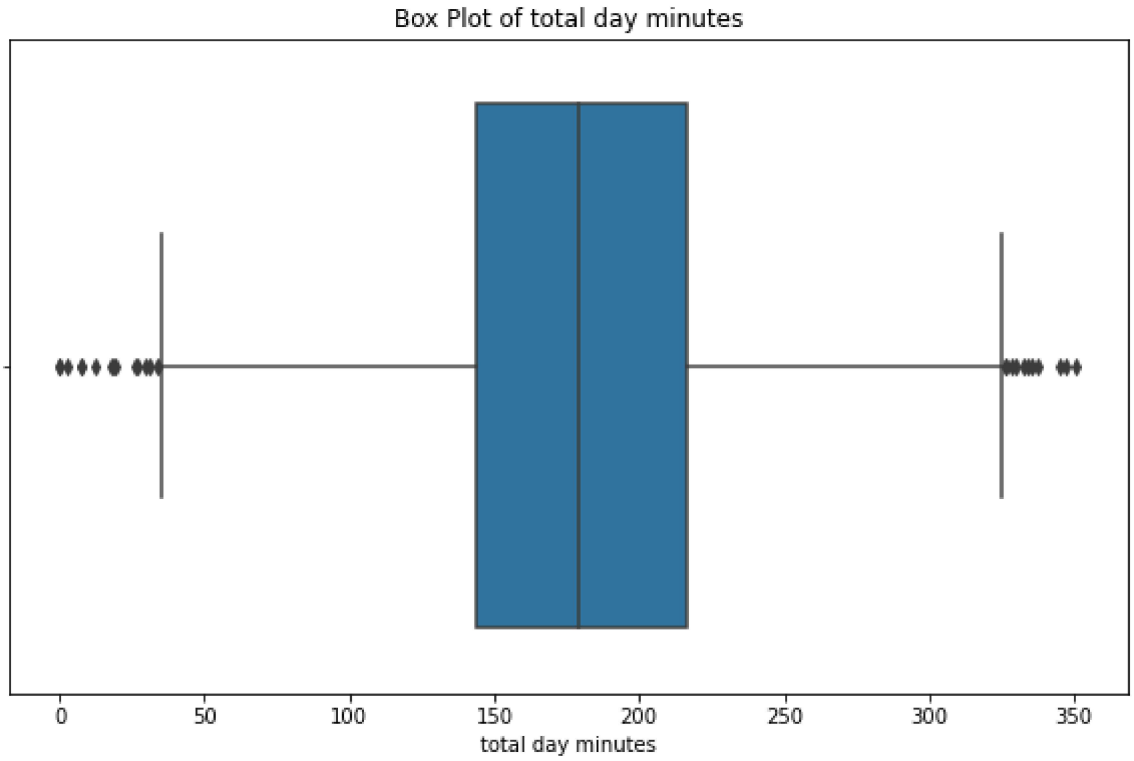

Histogram of total eve charge
Frequency
total eve charge
0
5
10
15
20
25
30

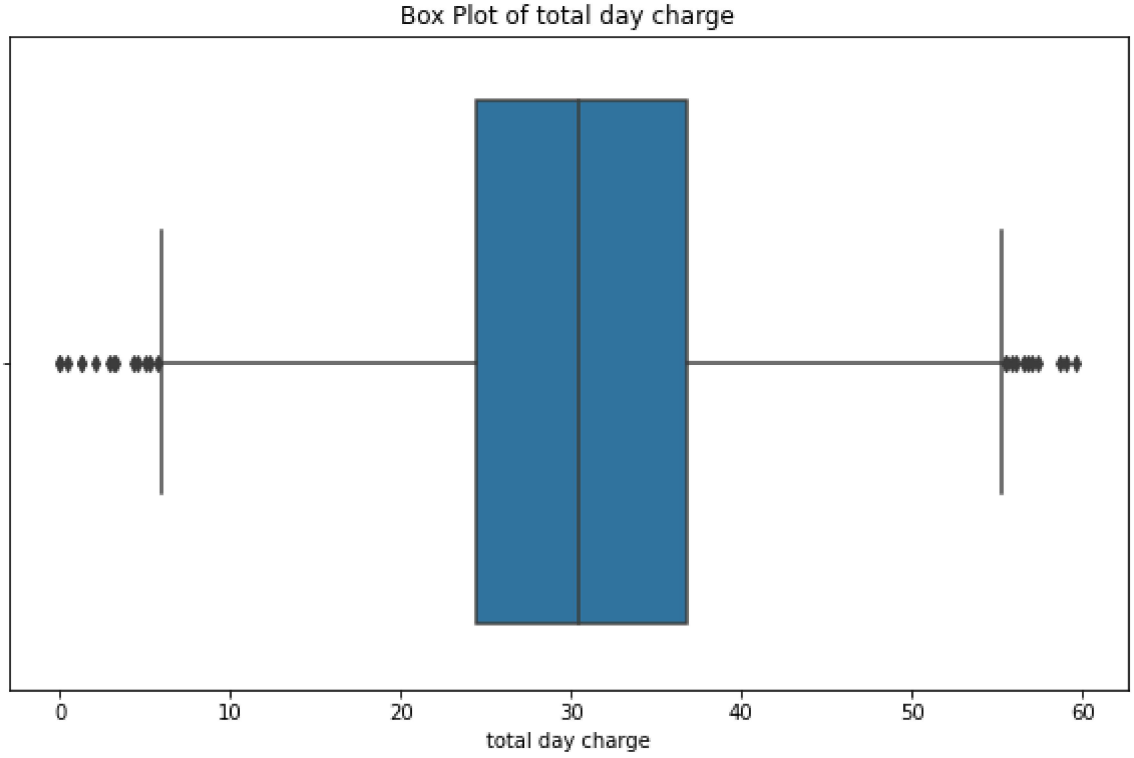

Histogram of total night minutes
Frequency
total night minutes
0
50
100
150
200
250
300
350
400

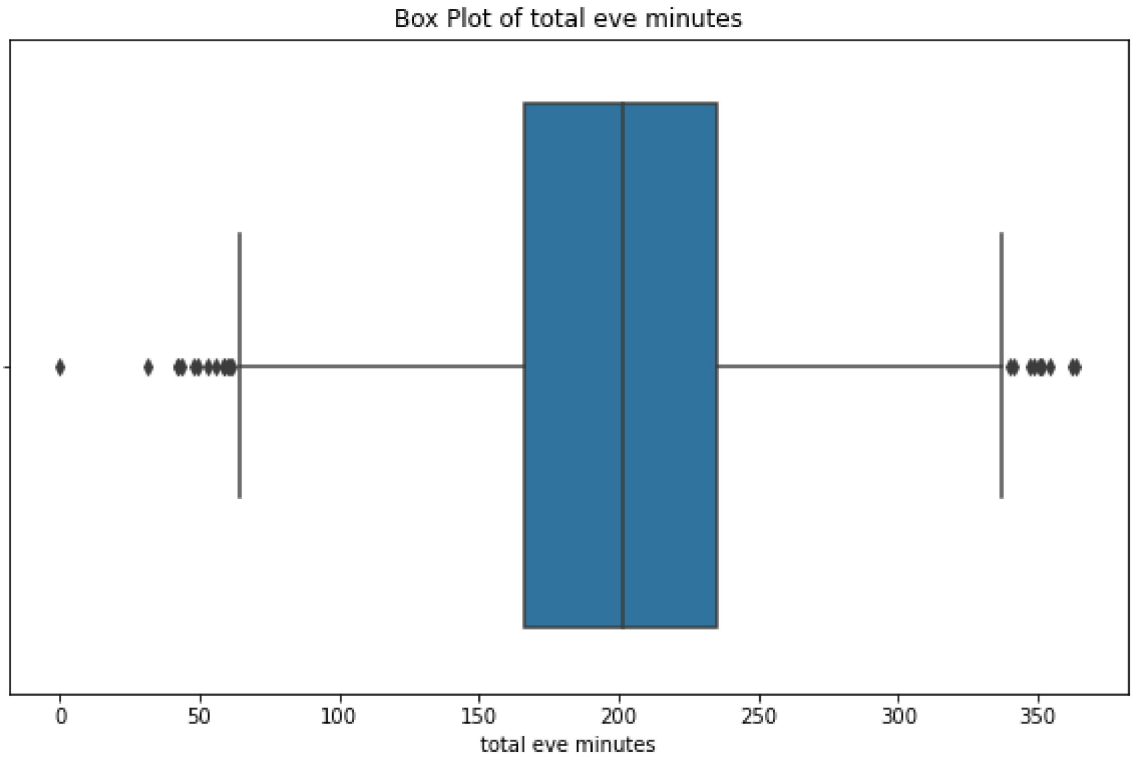

Histogram of total night charge
Frequency
total night charge
0
50
100
150
200
250
2.5
5.0
7.5
10.0
12.5
15.0
17.5

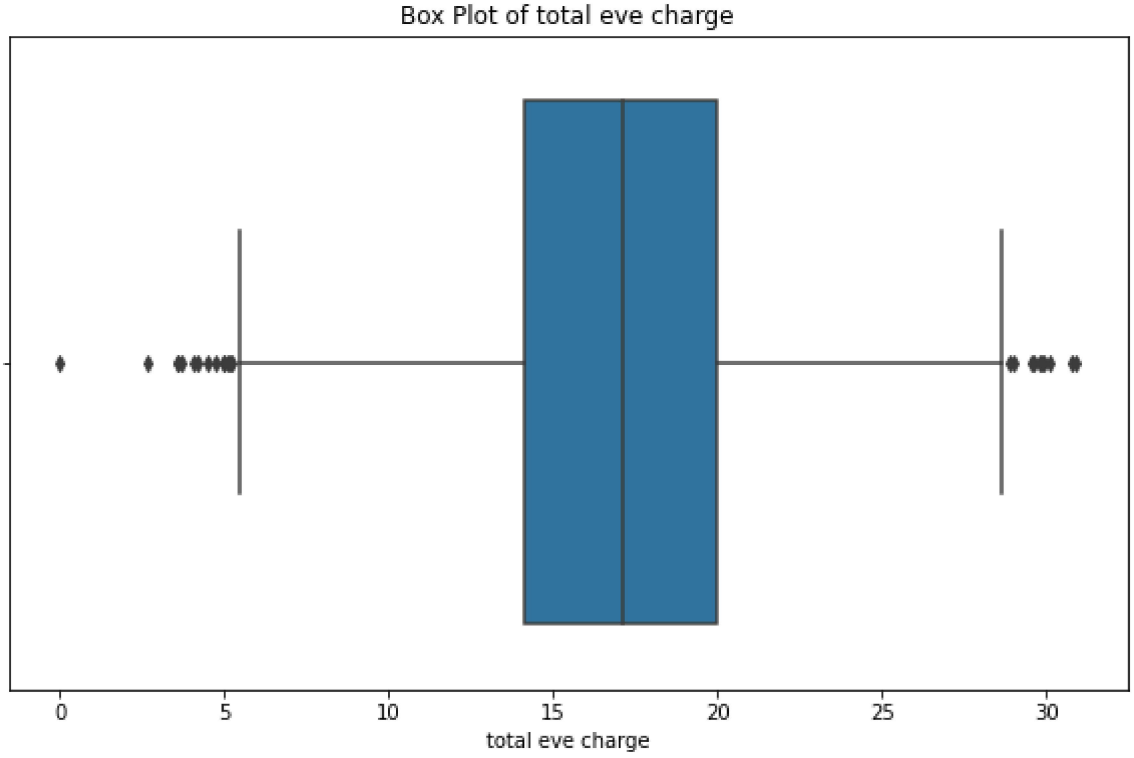

Histogram of total intl minutes
Frequency
total intl minutes
0
50
100
150
200
250
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

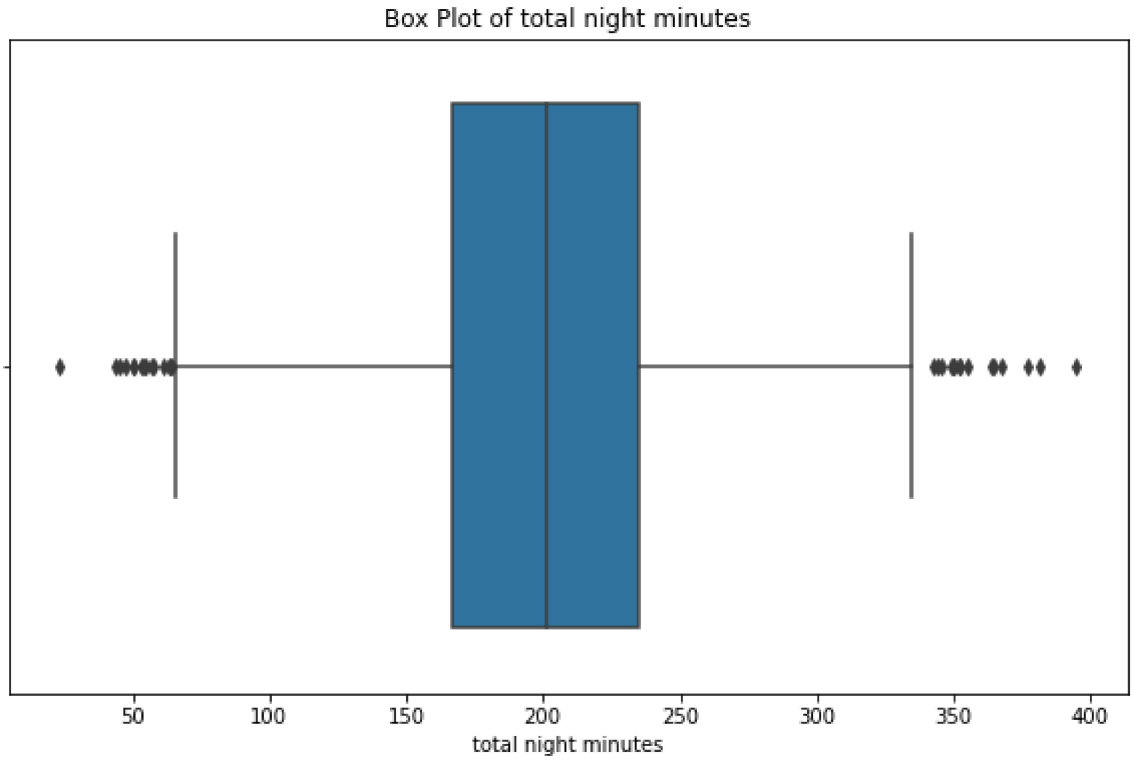

Histogram of total intl charge
Frequency
total intl charge
0
50
100
150
200
250
0
1
2
3
4
5

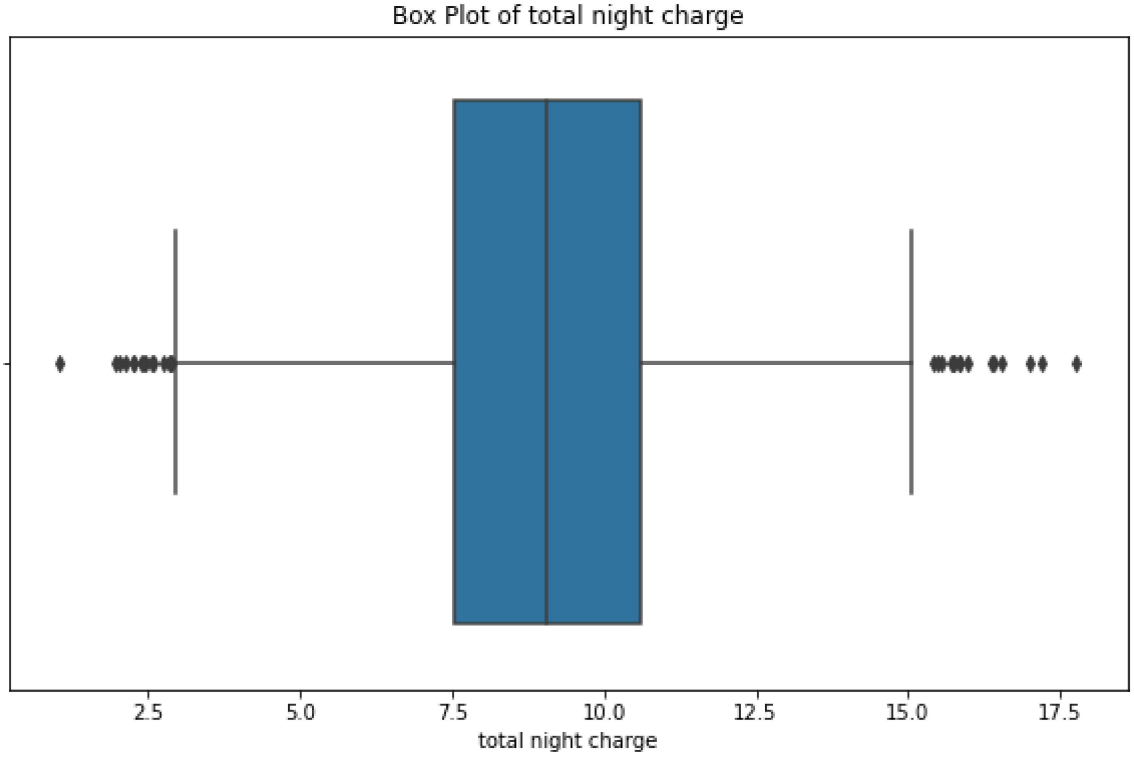

Box Plot of total day minutes
0
50
100
150
200
250
300
350
total day minutes

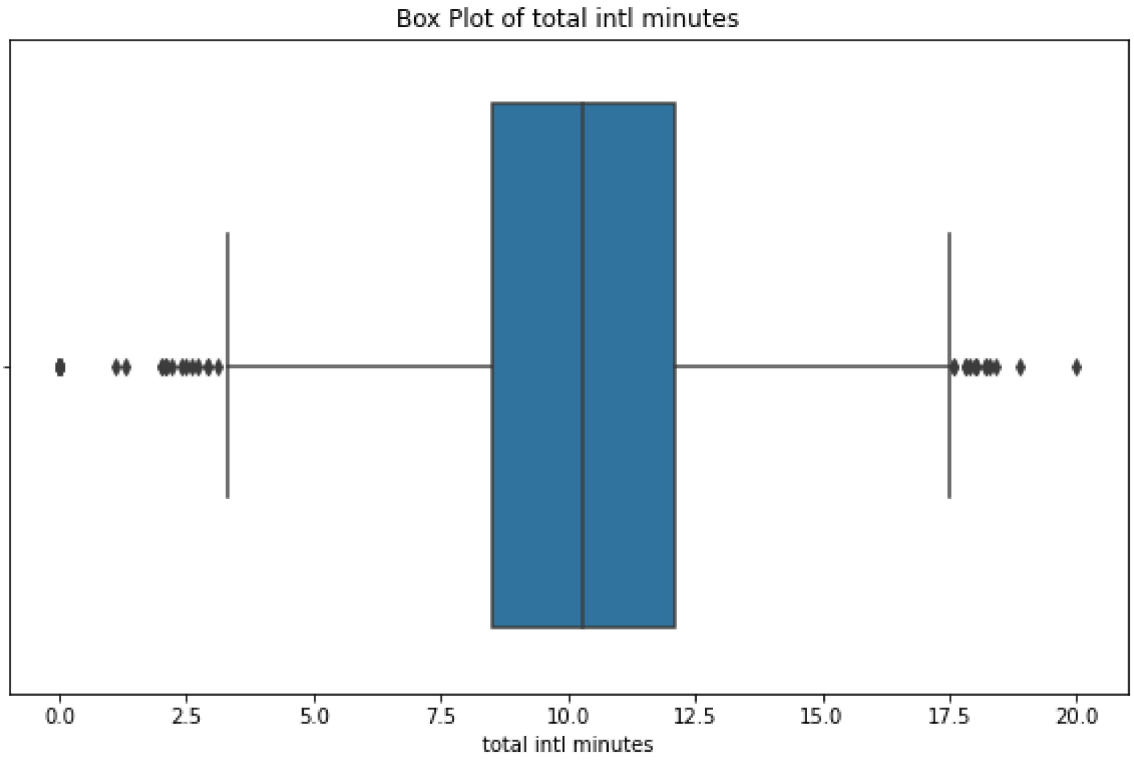

Box Plot of total day charge
0
10
20
30
40
50
60
total day charge

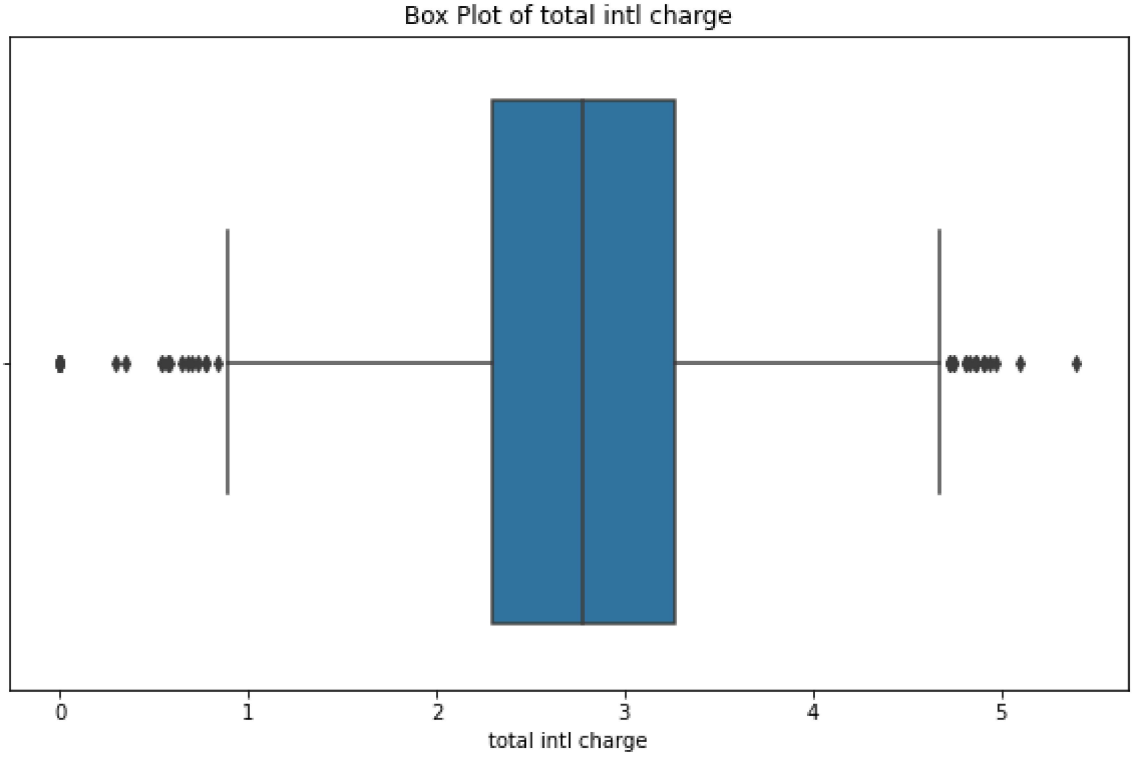

Box Plot of total eve minutes
0
50
100
150
200
250
300
350
total eve minutes

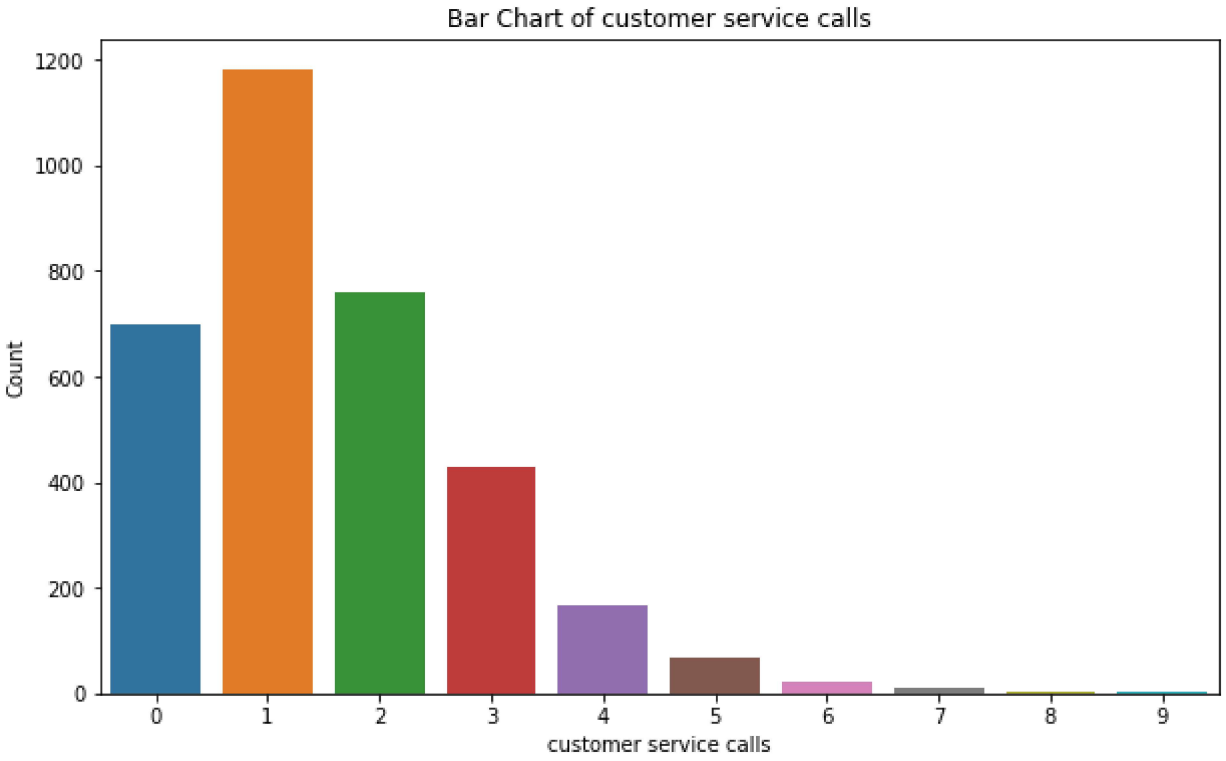

Box Plot of total eve charge
0
5
10
15
20
25
30
total eve charge


Box Plot of total night minutes
50
100
150
200
250
300
350
400
total night minutes

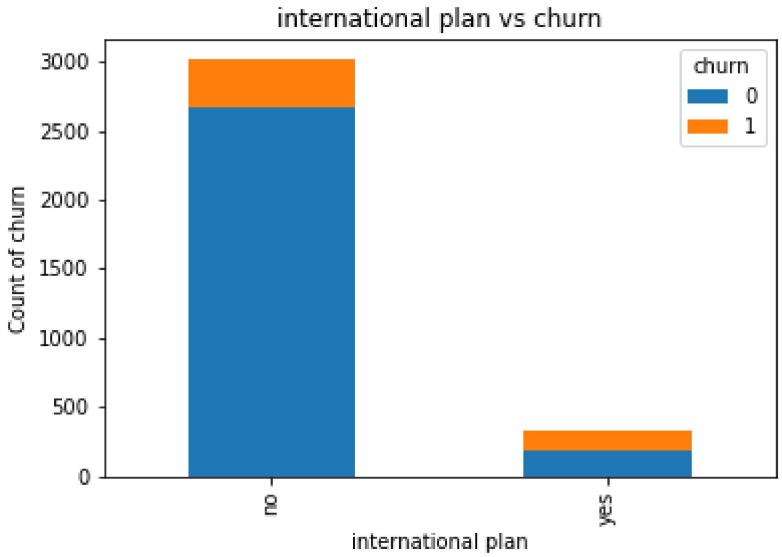

Box Plot of total night charge
2.5
5.0
7.5
10.0
12.5
15.0
17.5
total night charge

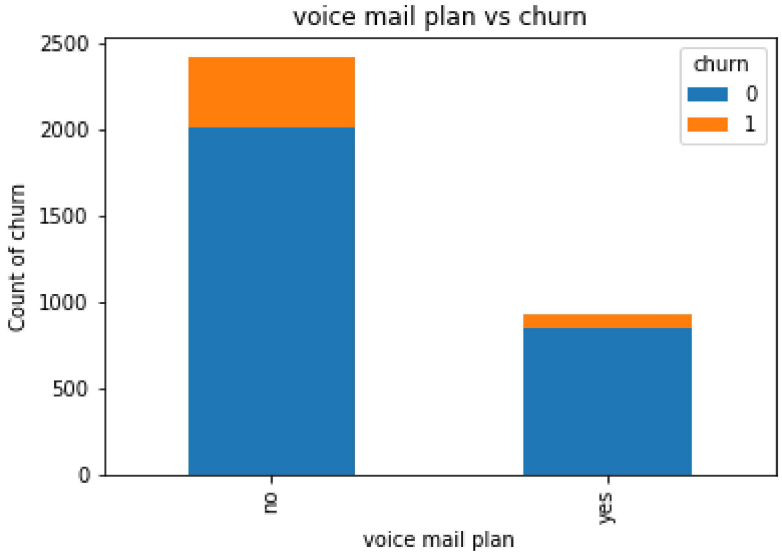

Box Plot of total intl minutes
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
total intl minutes

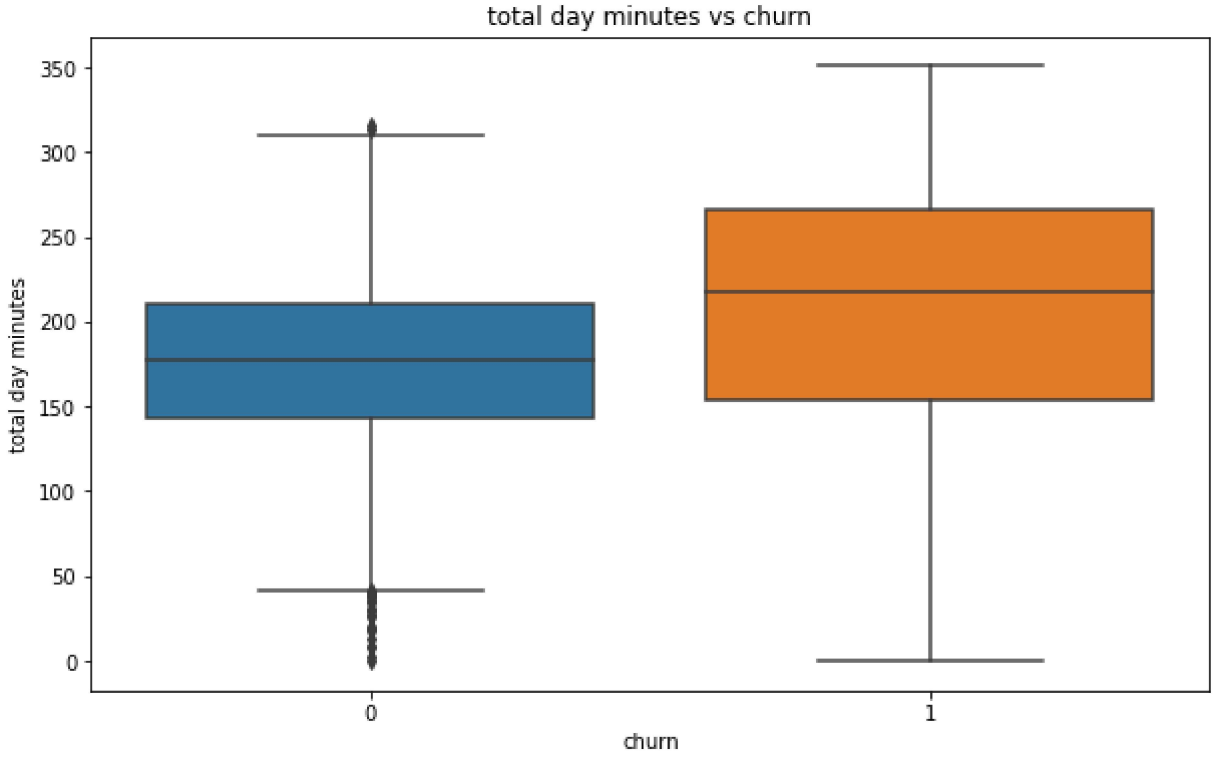

Box Plot of total intl charge
0
1
2
3
4
5
total intl charge

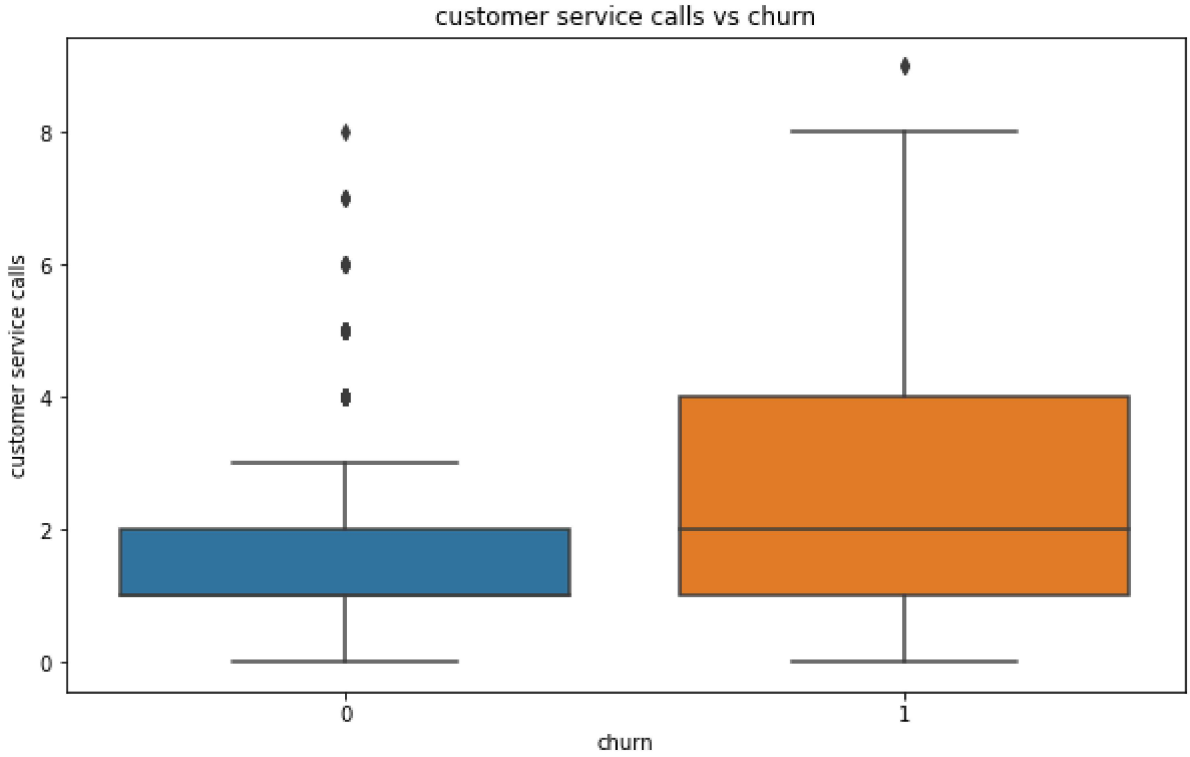


In [34]:
```python
# Bivariate Analysis

# Categorical vs. Categorical
def categorical_vs_categorical(df, col1, col2):
    crosstab = pd.crosstab(df[col1], df[col2])
    crosstab.plot(kind='bar', stacked=True)
    plt.title(f'{col1} vs {col2}')
    plt.ylabel(f'Count of {col2}')
    plt.xlabel(col1)
    plt.show()

# Numerical vs. Categorical
def numerical_vs_categorical(df, num_col, cat_col):
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=cat_col, y=num_col, data=df)
    plt.title(f'{num_col} vs {cat_col}')
    plt.xlabel(cat_col)
    plt.ylabel(num_col)
    plt.show()

# Apply Bivariate Analysis
categorical_vs_categorical(df, 'international plan', 'churn')
categorical_vs_categorical(df, 'voice mail plan', 'churn')
numerical_vs_categorical(df, 'total day minutes', 'churn')
numerical_vs_categorical(df, 'customer service calls', 'churn')
```


international plan vs churn
churn
0
1
Count of churn
3000
2500
2000
1500
1000
500
0
no
yes
international plan


voice mail plan vs churn
churn
0
1
Count of churn
2500
2000
1500
1000
500
0
no
yes
voice mail plan


total day minutes vs churn
total day minutes
350
300
250
200
150
100
50
0
0
1
churn



In [35]:
```
df.columns
```

Out[35]:
```
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', 'Total charge', 'Total Talk time',
       'Total calls', 'Avg Call duration', 'voice_ms_call_ratio',
       'charge_per_call_night', 'charge_per_call_day', 'charge_per_call_eve',
       'charge_per_call_intl'],
      dtype='object')
```

Data Preprocessing

In [36]:
```
import pandas as pd
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler
## defining function to check for colinearity

def check_collinearity_multicollinearity(df, numeric_features, categorical_features):

    # Encode categorical features using pd.get_dummies
    df_encoded = pd.get_dummies(df, columns=categorical_features, drop_first=True)

    # Separate numeric and encoded categorical data
    X = df_encoded[numeric_features + list(df_encoded.columns.difference(numeric_featur

    # Standardize numeric features for VIF calculation
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df_encoded[numeric_features])

    # Calculate VIF for numeric features
```

```
    vif_data = pd.DataFrame()
    vif_data['Feature'] = numeric_features
    vif_data['VIF'] = [variance_inflation_factor(X_scaled, i) for i in range(X_scaled.s

    print("Variance Inflation Factors (VIF) for Numeric Features:")
    print(vif_data)
    print("\n")

    # Calculate correlation matrix for combined features
    corr_matrix = X.corr()

    print("Correlation Matrix (including encoded categorical features):")
    print(corr_matrix)

    # Identify highly correlated pairs (absolute correlation > 0.8)
    high_corr_pairs = [(i, j, corr_matrix.iloc[i, j])
                       for i in range(corr_matrix.shape[0])
                       for j in range(i+1, corr_matrix.shape[1])
                       if abs(corr_matrix.iloc[i, j]) > 0.8]

    if high_corr_pairs:
        print("\nHighly correlated feature pairs (|correlation| > 0.8):")
        for i, j, corr in high_corr_pairs:
            print(f"{corr_matrix.columns[i]} and {corr_matrix.columns[j]}: {corr:.2f}")
    else:
        print("\nNo highly correlated feature pairs found.")

# Example usage
numeric_features = ['total day minutes', 'total day calls', 'total day charge', 'total
                    'total eve calls', 'total eve charge', 'total night minutes', 'tota
                    'total night charge', 'total intl minutes', 'total intl calls', 'to
                    'customer service calls']

categorical_features = ['international plan', 'voice mail plan']

# Assuming df is your dataframe with features
check_collinearity_multicollinearity(df, numeric_features, categorical_features)
```

```
Variance Inflation Factors (VIF) for Numeric Features:
                  Feature           VIF
0        total day minutes  1.047285e+07
1          total day calls  1.003134e+00
2         total day charge  1.047286e+07
3        total eve minutes  2.234958e+06
4          total eve calls  1.001909e+00
5         total eve charge  2.234958e+06
6      total night minutes  6.385707e+05
7        total night calls  1.001942e+00
8       total night charge  6.385694e+05
9       total intl minutes  6.900977e+04
10        total intl calls  1.002600e+00
11       total intl charge  6.901041e+04
12  customer service calls  1.001648e+00


Correlation Matrix (including encoded categorical features):
                      total day minutes  total day calls  total day charge  \
total day minutes              1.000000         0.006750          1.000000
total day calls                0.006750         1.000000          0.006753
total day charge               1.000000         0.006753          1.000000
total eve minutes              0.007043        -0.021451          0.007050
total eve calls                0.015769         0.006462          0.015769
```

total eve charge	0.007029	-0.021449	0.007036
total night minutes	0.004323	0.022938	0.004324
total night calls	0.022972	-0.019557	0.022972
total night charge	0.004300	0.022927	0.004301
total intl minutes	-0.010155	0.021565	-0.010157
total intl calls	0.008033	0.004574	0.008032
total intl charge	-0.010092	0.021666	-0.010094
customer service calls	-0.013423	-0.018942	-0.013427
Avg Call duration	0.469321	-0.347083	0.469323
Total Talk time	0.611587	0.005559	0.611591
Total calls	0.026676	0.575542	0.026678
Total charge	0.884754	0.003673	0.884757
account length	0.006216	0.038470	0.006214
area code	-0.008264	-0.009646	-0.008264
charge_per_call_day	-0.027128	-0.006811	-0.026918
charge_per_call_eve	-0.019943	0.011130	-0.019947
charge_per_call_intl	0.011097	0.016938	0.011096
charge_per_call_night	-0.022404	-0.008315	-0.022414
churn	0.205151	0.018459	0.205151
international plan_yes	0.049396	0.003755	0.049398
number vmail messages	0.000778	-0.009548	0.000776
voice mail plan_yes	-0.001684	-0.011086	-0.001686
voice_ms_call_ratio	0.001806	-0.048535	0.001803

	total eve minutes	total eve calls	total eve charge	\
total day minutes	0.007043	0.015769	0.007029	
total day calls	-0.021451	0.006462	-0.021449	
total day charge	0.007050	0.015769	0.007036	
total eve minutes	1.000000	-0.011430	1.000000	
total eve calls	-0.011430	1.000000	-0.011423	
total eve charge	1.000000	-0.011423	1.000000	
total night minutes	-0.012584	-0.002093	-0.012592	
total night calls	0.007586	0.007710	0.007596	
total night charge	-0.012593	-0.002056	-0.012601	
total intl minutes	-0.011035	0.008703	-0.011043	
total intl calls	0.002541	0.017434	0.002541	
total intl charge	-0.011067	0.008674	-0.011074	
customer service calls	-0.012985	0.002423	-0.012987	
Avg Call duration	0.453020	-0.355575	0.453002	
Total Talk time	0.560621	0.002198	0.560608	
Total calls	-0.014617	0.587726	-0.014606	
Total charge	0.413143	0.009392	0.413129	
account length	-0.006757	0.019260	-0.006745	
area code	0.003580	-0.011886	0.003607	
charge_per_call_day	0.017592	0.001509	0.017597	
charge_per_call_eve	-0.030188	0.005539	-0.029552	
charge_per_call_intl	-0.006197	-0.005190	-0.006192	
charge_per_call_night	0.001652	0.025952	0.001650	
churn	0.092796	0.009233	0.092786	
international plan_yes	0.019100	0.006114	0.019106	
number vmail messages	0.017562	-0.005864	0.017578	
voice mail plan_yes	0.021545	-0.006444	0.021559	
voice_ms_call_ratio	0.017038	-0.045385	0.017052	

	total night minutes	total night calls	\
total day minutes	0.004323	0.022972	
total day calls	0.022938	-0.019557	
total day charge	0.004324	0.022972	
total eve minutes	-0.012584	0.007586	
total eve calls	-0.002093	0.007710	
total eve charge	-0.012592	0.007596	
total night minutes	1.000000	0.011204	
total night calls	0.011204	1.000000	
total night charge	0.999999	0.011188	
total intl minutes	-0.015207	-0.013605	

total intl calls	-0.012353
total intl charge	-0.015180
customer service calls	-0.009288
Avg Call duration	0.432804
Total Talk time	0.557269
Total calls	0.017635
Total charge	0.214257
account length	-0.008955
area code	-0.005825
charge_per_call_day	-0.004255
charge_per_call_eve	-0.017021
charge_per_call_intl	0.015159
charge_per_call_night	0.016506
churn	0.035493
international plan_yes	-0.028905
number vmail messages	0.007681
voice mail plan_yes	0.006079
voice_ms_call_ratio	0.004639

	total night charge	total intl minutes	\
total day minutes	0.004300	-0.010155	
total day calls	0.022927	0.021565	
total day charge	0.004301	-0.010157	
total eve minutes	-0.012593	-0.011035	
total eve calls	-0.002056	0.008703	
total eve charge	-0.012601	-0.011043	
total night minutes	0.999999	-0.015207	
total night calls	0.011188	-0.013605	
total night charge	1.000000	-0.015214	
total intl minutes	-0.015214	1.000000	
total intl calls	-0.012329	0.032304	
total intl charge	-0.015186	0.999993	
customer service calls	-0.009277	-0.009640	
Avg Call duration	0.432786	0.002191	
Total Talk time	0.557249	0.010117	
Total calls	0.017642	0.012176	
Total charge	0.214233	0.054988	
account length	-0.008960	0.009514	
area code	-0.005845	-0.018288	
charge_per_call_day	-0.004281	-0.009498	
charge_per_call_eve	-0.017024	-0.018776	
charge_per_call_intl	0.015147	-0.075697	
charge_per_call_night	0.017675	-0.015530	
churn	0.035496	0.068239	
international plan_yes	-0.028913	0.045871	
number vmail messages	0.007663	0.002856	
voice mail plan_yes	0.006064	-0.001318	
voice_ms_call_ratio	0.004621	0.001632	

	total intl calls	total intl charge	\
total day minutes	0.008033	-0.010092	
total day calls	0.004574	0.021666	
total day charge	0.008032	-0.010094	
total eve minutes	0.002541	-0.011067	
total eve calls	0.017434	0.008674	
total eve charge	0.002541	-0.011074	
total night minutes	-0.012353	-0.015180	
total night calls	0.000305	-0.013630	
total night charge	-0.012329	-0.015186	
total intl minutes	0.032304	0.999993	
total intl calls	1.000000	0.032372	
total intl charge	0.032372	1.000000	
customer service calls	-0.017561	-0.009675	
Avg Call duration	-0.047492	0.002187	
Total Talk time	0.000354	0.010152	

Total calls	0.084367	0.012209
Total charge	0.007776	0.055036
account length	0.020661	0.009546
area code	-0.024179	-0.018395
charge_per_call_day	-0.010863	-0.009526
charge_per_call_eve	0.002483	-0.018780
charge_per_call_intl	0.010990	-0.072017
charge_per_call_night	0.019959	-0.015614
churn	-0.052844	0.068259
international plan_yes	0.017366	0.045780
number vmail messages	0.013957	0.002884
voice mail plan_yes	0.007618	-0.001276
voice_ms_call_ratio	0.006975	0.001647

	customer service calls	Avg Call duration	\
total day minutes	-0.013423	0.469321	
total day calls	-0.018942	-0.347083	
total day charge	-0.013427	0.469323	
total eve minutes	-0.012985	0.453020	
total eve calls	0.002423	-0.355575	
total eve charge	-0.012987	0.453002	
total night minutes	-0.009288	0.432804	
total night calls	-0.012802	-0.319435	
total night charge	-0.009277	0.432786	
total intl minutes	-0.009640	0.002191	
total intl calls	-0.017561	-0.047492	
total intl charge	-0.009675	0.002187	
customer service calls	1.000000	-0.008634	
Avg Call duration	-0.008634	1.000000	
Total Talk time	-0.020969	0.782973	
Total calls	-0.018161	-0.592699	
Total charge	-0.019873	0.693662	
account length	-0.003796	-0.022039	
area code	0.027572	-0.004271	
charge_per_call_day	-0.012554	0.003264	
charge_per_call_eve	-0.002548	-0.044446	
charge_per_call_intl	-0.025068	0.003864	
charge_per_call_night	0.002041	-0.007620	
churn	0.208750	0.144784	
international plan_yes	-0.024522	0.012757	
number vmail messages	-0.013263	0.013264	
voice mail plan_yes	-0.017824	0.011176	
voice_ms_call_ratio	-0.014619	0.053831	

	Total Talk time	Total calls	Total charge	\
total day minutes	0.611587	0.026676	0.884754	
total day calls	0.005559	0.575542	0.003673	
total day charge	0.611591	0.026678	0.884757	
total eve minutes	0.560621	-0.014617	0.413143	
total eve calls	0.002198	0.587726	0.009392	
total eve charge	0.560608	-0.014606	0.413129	
total night minutes	0.557269	0.017635	0.214257	
total night calls	0.024063	0.561146	0.024818	
total night charge	0.557249	0.017642	0.214233	
total intl minutes	0.010117	0.012176	0.054988	
total intl calls	0.000354	0.084367	0.007776	
total intl charge	0.010152	0.012209	0.055036	
customer service calls	-0.020969	-0.018161	-0.019873	
Avg Call duration	0.782973	-0.592699	0.693662	
Total Talk time	1.000000	0.018204	0.890804	
Total calls	0.018204	1.000000	0.022225	
Total charge	0.890804	0.022225	1.000000	
account length	-0.004785	0.027542	0.001454	
area code	-0.006828	-0.004836	-0.008393	
charge_per_call_day	-0.009162	-0.021286	-0.018105	

```
charge_per_call_eve          -0.039234     0.019346     -0.034731
charge_per_call_intl          0.009471     0.001962      0.005529
charge_per_call_night        -0.003837     0.010735     -0.016375
churn                         0.198607     0.015807      0.231549
international plan_yes        0.025850     0.014037      0.048415
number vmail messages         0.014779    -0.003911      0.009766
voice mail plan_yes           0.014503    -0.000807      0.008585
voice_ms_call_ratio           0.013358    -0.070886      0.009709

                        account length  area code  charge_per_call_day  \
total day minutes             0.006216  -0.008264            -0.027128
total day calls               0.038470  -0.009646            -0.006811
total day charge              0.006214  -0.008264            -0.026918
total eve minutes            -0.006757   0.003580             0.017592
total eve calls               0.019260  -0.011886             0.001509
total eve charge             -0.006745   0.003607             0.017597
total night minutes          -0.008955  -0.005825            -0.004255
total night calls            -0.013176   0.016522            -0.030640
total night charge           -0.008960  -0.005845            -0.004281
total intl minutes            0.009514  -0.018288            -0.009498
total intl calls              0.020661  -0.024179            -0.010863
total intl charge             0.009546  -0.018395            -0.009526
customer service calls       -0.003796   0.027572            -0.012554
Avg Call duration            -0.022039  -0.004271             0.003264
Total Talk time              -0.004785  -0.006828            -0.009162
Total calls                   0.027542  -0.004836            -0.021286
Total charge                  0.001454  -0.008393            -0.018105
account length                1.000000  -0.012463            -0.000073
area code                    -0.012463   1.000000            -0.009105
charge_per_call_day          -0.000073  -0.009105             1.000000
charge_per_call_eve           0.026133   0.032689             0.001915
charge_per_call_intl          0.006221  -0.024514            -0.001227
charge_per_call_night        -0.009487  -0.008893            -0.022826
churn                         0.016541   0.006174            -0.007103
international plan_yes        0.024735   0.048551             0.016647
number vmail messages        -0.004628  -0.001994            -0.031363
voice mail plan_yes           0.002918  -0.000747            -0.027948
voice_ms_call_ratio          -0.004229  -0.001789            -0.027532

                        charge_per_call_eve  charge_per_call_intl  \
total day minutes                -0.019943              0.011097
total day calls                   0.011130              0.016938
total day charge                 -0.019947              0.011096
total eve minutes                -0.030188             -0.006197
total eve calls                   0.005539             -0.005190
total eve charge                 -0.029552             -0.006192
total night minutes              -0.017021              0.015159
total night calls                 0.016695             -0.010011
total night charge               -0.017024              0.015147
total intl minutes               -0.018776             -0.075697
total intl calls                  0.002483              0.010990
total intl charge                -0.018780             -0.072017
customer service calls           -0.002548             -0.025068
Avg Call duration                -0.044446              0.003864
Total Talk time                  -0.039234              0.009471
Total calls                       0.019346              0.001962
Total charge                     -0.034731              0.005529
account length                    0.026133              0.006221
area code                         0.032689             -0.024514
charge_per_call_day               0.001915             -0.001227
charge_per_call_eve               1.000000              0.009178
charge_per_call_intl              0.009178              1.000000
charge_per_call_night            -0.012799             -0.020056
churn                            -0.008959             -0.001715
international plan_yes            0.006415             -0.018257
```

```
number vmail messages              0.019557              0.008435
voice mail plan_yes                0.015298              0.012561
voice_ms_call_ratio                0.017541              0.005344

                         charge_per_call_night     churn  \
total day minutes                    -0.022404  0.205151
total day calls                      -0.008315  0.018459
total day charge                     -0.022414  0.205151
total eve minutes                     0.001652  0.092796
total eve calls                       0.025952  0.009233
total eve charge                      0.001650  0.092786
total night minutes                   0.016506  0.035493
total night calls                    -0.001506  0.006141
total night charge                    0.017675  0.035496
total intl minutes                   -0.015530  0.068239
total intl calls                      0.019959 -0.052844
total intl charge                    -0.015614  0.068259
customer service calls                0.002041  0.208750
Avg Call duration                    -0.007620  0.144784
Total Talk time                      -0.003837  0.198607
Total calls                           0.010735  0.015807
Total charge                         -0.016375  0.231549
account length                       -0.009487  0.016541
area code                            -0.008893  0.006174
charge_per_call_day                  -0.022826 -0.007103
charge_per_call_eve                  -0.012799 -0.008959
charge_per_call_intl                 -0.020056 -0.001715
charge_per_call_night                 1.000000 -0.003009
churn                                -0.003009  1.000000
international plan_yes               -0.015902  0.259852
number vmail messages                -0.018974 -0.089728
voice mail plan_yes                  -0.016235 -0.102148
voice_ms_call_ratio                  -0.019380 -0.090484

                         international plan_yes  number vmail messages  \
total day minutes                      0.049396               0.000778
total day calls                        0.003755              -0.009548
total day charge                       0.049398               0.000776
total eve minutes                      0.019100               0.017562
total eve calls                        0.006114              -0.005864
total eve charge                       0.019106               0.017578
total night minutes                   -0.028905               0.007681
total night calls                      0.012451               0.007123
total night charge                    -0.028913               0.007663
total intl minutes                     0.045871               0.002856
total intl calls                       0.017366               0.013957
total intl charge                      0.045780               0.002884
customer service calls                -0.024522              -0.013263
Avg Call duration                      0.012757               0.013264
Total Talk time                        0.025850               0.014779
Total calls                            0.014037              -0.003911
Total charge                           0.048415               0.009766
account length                         0.024735              -0.004628
area code                              0.048551              -0.001994
charge_per_call_day                    0.016647              -0.031363
charge_per_call_eve                    0.006415               0.019557
charge_per_call_intl                  -0.018257               0.008435
charge_per_call_night                 -0.015902              -0.018974
churn                                  0.259852              -0.089728
international plan_yes                 1.000000               0.008745
number vmail messages                  0.008745               1.000000
voice mail plan_yes                    0.006006               0.956927
voice_ms_call_ratio                    0.006694               0.991030

                         voice mail plan_yes  voice_ms_call_ratio
```

```
total day minutes               -0.001684            0.001806
total day calls                 -0.011086           -0.048535
total day charge                -0.001686            0.001803
total eve minutes                0.021545            0.017038
total eve calls                 -0.006444           -0.045385
total eve charge                 0.021559            0.017052
total night minutes              0.006079            0.004639
total night calls                0.015553           -0.029682
total night charge               0.006064            0.004621
total intl minutes              -0.001318            0.001632
total intl calls                 0.007618            0.006975
total intl charge               -0.001276            0.001647
customer service calls          -0.017824           -0.014619
Avg Call duration                0.011176            0.053831
Total Talk time                  0.014503            0.013358
Total calls                     -0.000807           -0.070886
Total charge                     0.008585            0.009709
account length                   0.002918           -0.004229
area code                       -0.000747           -0.001789
charge_per_call_day             -0.027948           -0.027532
charge_per_call_eve              0.015298            0.017541
charge_per_call_intl             0.012561            0.005344
charge_per_call_night           -0.016235           -0.019380
churn                           -0.102148           -0.090484
international plan_yes           0.006006            0.006694
number vmail messages            0.956927            0.991030
voice mail plan_yes              1.000000            0.947602
voice_ms_call_ratio              0.947602            1.000000

Highly correlated feature pairs (|correlation| > 0.8):
total day minutes and total day charge: 1.00
total day minutes and Total charge: 0.88
total day charge and Total charge: 0.88
total eve minutes and total eve charge: 1.00
total night minutes and total night charge: 1.00
total intl minutes and total intl charge: 1.00
Total Talk time and Total charge: 0.89
number vmail messages and voice mail plan_yes: 0.96
number vmail messages and voice_ms_call_ratio: 0.99
voice mail plan_yes and voice_ms_call_ratio: 0.95
```

In [37]:

```
#dropping columns with high colinearity

features= ['number vmail messages', 'total day minutes','total eve minutes','total nigh
       'total night charge', 'total intl minutes']
df =df.drop(features,axis=1)
df.head()
```

Out[37]:

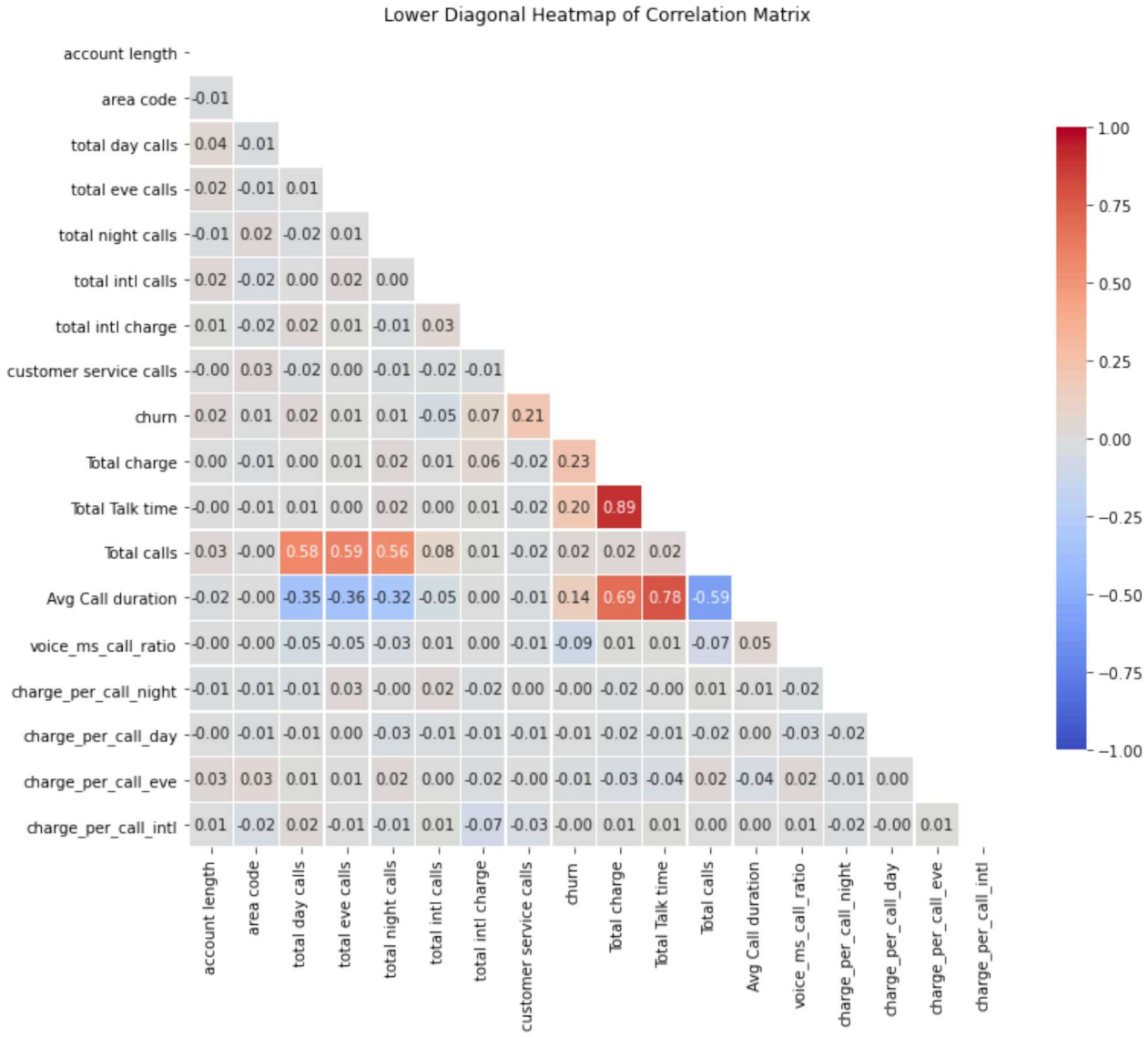
	state	account length	area code	phone number	international plan	voice mail plan	total day calls	total eve calls	total night calls	total intl calls	total intl charge	customer service calls	chu
0	KS	128	415	382-4657	no	yes	110	99	91	3	2.70	1	
1	OH	107	415	371-7191	no	yes	123	103	103	3	3.70	1	
2	NJ	137	415	358-1921	no	no	114	110	104	5	3.29	0	
3	OH	84	408	375-9999	yes	no	71	88	89	7	1.78	2	

	state	account length	area code	phone number	international plan	voice mail plan	total day calls	total eve calls	total night calls	total intl calls	total intl charge	customer service calls	chu
4	OK	75	415	330-6626	yes	no	113	122	121	3	2.73	3	

In [38]:
```
df.columns
```

Out[38]:
```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'total day calls',
       'total eve calls', 'total night calls', 'total intl calls',
       'total intl charge', 'customer service calls', 'churn', 'Total charge',
       'Total Talk time', 'Total calls', 'Avg Call duration',
       'voice_ms_call_ratio', 'charge_per_call_night', 'charge_per_call_day',
       'charge_per_call_eve', 'charge_per_call_intl'],
      dtype='object')
```

In [39]:
```
# Compute the correlation matrix for the remaining features
corr = df.corr()

# Create a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
plt.figure(figsize=(12, 10))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap='coolwarm', annot=True, fmt=".2f", vmin=-1, vmax=1, c

# Show the plot
plt.title('Lower Diagonal Heatmap of Correlation Matrix')
plt.show()
```



In [40]:
```
# Get the percentage distribution of the churn variable
churn_percentage = df["churn"].value_counts(normalize=True) * 100

# Print the result
print(churn_percentage)
```

```
0    85.508551
1    14.491449
Name: churn, dtype: float64
```

In [41]:
```
# checking for null values
df.isna().sum()
```

Out[41]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
total day calls           0
total eve calls           0
total night calls         0
total intl calls          0
total intl charge         0
customer service calls    0
```

```
churn                     0
Total charge              0
Total Talk time           0
Total calls               0
Avg Call duration         0
voice_ms_call_ratio       0
charge_per_call_night     0
charge_per_call_day       2
charge_per_call_eve       1
charge_per_call_intl     18
dtype: int64
```

In [42]:
```
df.dtypes
```

Out[42]:
```
state                    object
account length            int64
area code                 int64
phone number             object
international plan       object
voice mail plan          object
total day calls           int64
total eve calls           int64
total night calls         int64
total intl calls          int64
total intl charge       float64
customer service calls    int64
churn                     int32
Total charge            float64
Total Talk time         float64
Total calls               int64
Avg Call duration       float64
voice_ms_call_ratio     float64
charge_per_call_night   float64
charge_per_call_day     float64
charge_per_call_eve     float64
charge_per_call_intl    float64
dtype: object
```

In [43]:
```
# handling null values
from sklearn.impute import SimpleImputer

# Separate numeric and categorical columns
numeric_columns = df.select_dtypes(include=['number']).columns
categorical_columns = df.select_dtypes(include=['object']).columns

# Numeric imputer (using median)
numeric_imputer = SimpleImputer(strategy='median')
df[numeric_columns] = numeric_imputer.fit_transform(df[numeric_columns])

# Categorical imputer (using most frequent value or mode)
categorical_imputer = SimpleImputer(strategy='most_frequent')
df[categorical_columns] = categorical_imputer.fit_transform(df[categorical_columns])

# Verify there are no more missing values
print("Missing values in each column after imputation:\n", df.isnull().sum())
```

```
Missing values in each column after imputation:
 state                    0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
```

```
total day calls           0
total eve calls           0
total night calls         0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
Total charge              0
Total Talk time           0
Total calls               0
Avg Call duration         0
voice_ms_call_ratio       0
charge_per_call_night     0
charge_per_call_day       0
charge_per_call_eve       0
charge_per_call_intl      0
dtype: int64
```

scaling and encoding our data

In [44]:
```
# Separate target variable if it's present
target = 'churn'  # Replace with your actual target column name
X = df.drop(columns=[target])
y = df[target]

# Separate numerical and categorical features
numerical_features = X.select_dtypes(include=['int64', 'float64']).columns
categorical_features = X.select_dtypes(include=['object', 'bool']).columns

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# Encode categorical features using pd.get_dummies
X = pd.get_dummies(X, columns=categorical_features, drop_first=True)
```

In [45]:
```
# splitting the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, ra
```

In [46]:
```
# Apply SMOTE to the training set
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)
```

Modelling

In [47]:
```
from sklearn.dummy import DummyClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, ro

# Create a dummy classifier that always predicts the most frequent class
baseline_model = DummyClassifier(strategy='most_frequent', random_state=42)
baseline_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred_baseline = baseline_model.predict(X_test)

# Calculate metrics
accuracy_baseline = accuracy_score(y_test, y_pred_baseline)
roc_auc_baseline = roc_auc_score(y_test, y_pred_baseline)
```

```
print("Baseline Model Metrics:")
print(f"Accuracy: {accuracy_baseline:.4f}")
print(f"ROC AUC: {roc_auc_baseline:.4f}")
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_baseline))
print("Classification Report:\n", classification_report(y_test, y_pred_baseline))
```

```
Baseline Model Metrics:
Accuracy: 0.8546
ROC AUC: 0.5000
Confusion Matrix:
 [[570   0]
 [ 97   0]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.85      1.00      0.92       570
         1.0       0.00      0.00      0.00        97

    accuracy                           0.85       667
   macro avg       0.43      0.50      0.46       667
weighted avg       0.73      0.85      0.79       667
```

In [48]:

```
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix, classifica

def evaluate_decision_tree(X_train, y_train, X_test, y_test):
    from sklearn.tree import DecisionTreeClassifier
    dt_model = DecisionTreeClassifier(random_state=42)
    dt_model.fit(X_train, y_train)
    y_pred = dt_model.predict(X_test)
    print("Decision Tree Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_pred):.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")

def evaluate_logistic_regression(X_train, y_train, X_test, y_test):
    from sklearn.linear_model import LogisticRegression
    lr_model = LogisticRegression(random_state=42)
    lr_model.fit(X_train, y_train)
    y_pred = lr_model.predict(X_test)
    print("Logistic Regression Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_pred):.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")

def evaluate_random_forest(X_train, y_train, X_test, y_test):
    from sklearn.ensemble import RandomForestClassifier
    rf_model = RandomForestClassifier(random_state=42)
    rf_model.fit(X_train, y_train)
    y_pred = rf_model.predict(X_test)
    print("Random Forest Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_pred):.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")
```

```
def evaluate_gradient_boosting(X_train, y_train, X_test, y_test):
    from sklearn.ensemble import GradientBoostingClassifier
    gb_model = GradientBoostingClassifier(random_state=42)
    gb_model.fit(X_train, y_train)
    y_pred = gb_model.predict(X_test)
    print("Gradient Boosting Metrics:")
    print(f"Accuracy: {accuracy_score(y_test, y_pred):.4f}")
    print(f"ROC AUC: {roc_auc_score(y_test, y_pred):.4f}")
    print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
    print("Classification Report:\n", classification_report(y_test, y_pred))
    print("\n")
```

In [49]:
```
# Evaluate each model
evaluate_decision_tree(X_train, y_train, X_test, y_test)
evaluate_logistic_regression(X_train, y_train, X_test, y_test)
evaluate_random_forest(X_train, y_train, X_test, y_test)
evaluate_gradient_boosting(X_train, y_train, X_test, y_test)
```

```
Decision Tree Metrics:
Accuracy: 0.9685
ROC AUC: 0.8918
Confusion Matrix:
 [[570   0]
 [ 21  76]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       570
         1.0       1.00      0.78      0.88        97

    accuracy                           0.97       667
   macro avg       0.98      0.89      0.93       667
weighted avg       0.97      0.97      0.97       667



Logistic Regression Metrics:
Accuracy: 0.8636
ROC AUC: 0.6036
Confusion Matrix:
 [[553  17]
 [ 74  23]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.88      0.97      0.92       570
         1.0       0.57      0.24      0.34        97

    accuracy                           0.86       667
   macro avg       0.73      0.60      0.63       667
weighted avg       0.84      0.86      0.84       667



Random Forest Metrics:
Accuracy: 0.9130
ROC AUC: 0.7053
Confusion Matrix:
 [[569   1]
 [ 57  40]]
Classification Report:
              precision    recall  f1-score   support
```

```
         0.0       0.91      1.00      0.95       570
         1.0       0.98      0.41      0.58        97

    accuracy                           0.91       667
   macro avg       0.94      0.71      0.77       667
weighted avg       0.92      0.91      0.90       667
```

```
Gradient Boosting Metrics:
Accuracy: 0.9685
ROC AUC: 0.8918
Confusion Matrix:
 [[570   0]
 [ 21  76]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.96      1.00      0.98       570
         1.0       1.00      0.78      0.88        97

    accuracy                           0.97       667
   macro avg       0.98      0.89      0.93       667
weighted avg       0.97      0.97      0.97       667
```

Interpreting the Model Performance Metrics Understanding the Metrics:

Accuracy: The overall proportion of correct predictions. ROC AUC: Measures the model's ability to distinguish between positive and negative classes. Confusion Matrix: Shows the number of correct and incorrect predictions for each class. Classification Report: Provides precision, recall, F1-score, and support for each class. Comparing Model Performance:

Accuracy: Decision Tree and Gradient Boosting have the highest accuracy (0.9685). ROC AUC: Decision Tree and Gradient Boosting also have the highest ROC AUC (0.8918), indicating good discrimination between classes. Precision and Recall: Precision: Measures the proportion of positive predictions that are actually positive. Decision Tree and Gradient Boosting have high precision for class 0, while Random Forest has high precision for class 1. Recall: Measures the proportion of actual positive cases that are correctly predicted. Decision Tree and Gradient Boosting have high recall for both classes. F1-score: The harmonic mean of precision and recall. Decision Tree and Gradient Boosting have high F1-scores for class 0, while Random Forest has a lower F1-score for class 1. Key Insights:

Decision Tree and Gradient Boosting: These models perform exceptionally well in terms of accuracy, ROC AUC, precision, and recall for both classes. Logistic Regression: While it has a lower accuracy and ROC AUC, it might be useful for interpretability. Random Forest: It performs well for class 0 but struggles with class 1, particularly in terms of recall. Choosing the Best Model:

The choice of the best model depends on your specific priorities. If you value overall accuracy and performance for both classes, Decision Tree or Gradient Boosting would be good options. If interpretability is important, Logistic Regression might be considered.

Further Analysis:

Feature Importance: Analyze the feature importance for each model to understand which factors contribute most to the predictions. Hyperparameter Tuning: Experiment with different hyperparameters to potentially improve model performance. Cross-Validation: Use cross-validation to assess the model's generalization ability. By carefully considering these factors, you can select the most appropriate model for your specific use case and gain valuable insights from the analysis.

In [76]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Data for the models
data = {
    'Model': ['Decision Tree', 'Logistic Regression', 'Random Forest', 'Gradient Boosti
    'Accuracy (%)': [96.85, 86.36, 91.30, 96.85],
    'ROC AUC': [0.8918, 0.6036, 0.7053, 0.8918],
    'Precision (Class 0)': [0.96, 0.88, 0.91, 0.96],
    'Precision (Class 1)': [1.0, 1.0, 1.0, 1.0],
    'Recall (Class 0)': [1.0, 0.97, 1.0, 1.0],
    'Recall (Class 1)': [0.78, 0.57, 0.98, 0.78],
    'F1-score (Class 0)': [0.98, 0.92, 0.95, 0.98],
    'F1-score (Class 1)': [0.88, 0.24, 0.41, 0.88]
}

df = pd.DataFrame(data)

# Create a table to display the metrics
fig, ax = plt.subplots(figsize=(12, 7))  # Adjust the size as needed
ax.axis('off')  # Hide the axes

# Create a table
table = ax.table(cellText=df.values,
                 colLabels=df.columns,
                 rowLabels=df['Model'],
                 cellLoc='center',
                 loc='center',
                 bbox=[0, 0, 1, 1])  # Position and size of the table

# Adjust font size and other properties
table.auto_set_font_size(False)
table.set_fontsize(12)
table.auto_set_column_width([i for i in range(len(df.columns))])

# Add title with spacing
plt.title('Model Performance Metrics', fontsize=18, pad=20)

# Improve spacing between rows
for key, cell in table._cells.items():
    cell.set_edgecolor('black')
    cell.set_linewidth(1)
    if key[0] == 0:  # Header row
        cell.set_text_props(weight='bold')
        cell.set_facecolor('#f0f0f0')  # Light grey background for header
    if key[1] == -1:  # Row labels
        cell.set_text_props(weight='bold')

plt.show()
```

Model Performance Metrics

	Model	Accuracy R	OC AU	ecision (Cla	recision (Cla	ecall (Class	ecall (Cla	1-score (Cla	1-score (Class 1)
Decision Tre	Decision Tree	96.85	0.8918	0.96	1.0	1.0	0.78	0.98	0.88
Logistic Reg	ogistic Regress	86.36	0.6036	0.88	1.0	0.97	0.57	0.92	0.24
Random For	Random Fores	91.3	0.7053	0.91	1.0	1.0	0.98	0.95	0.41
Gradient Bo	radient Boosti	96.85	0.8918	0.96	1.0	1.0	0.78	0.98	0.88

Decison tree Modelling evaluation after hypertuning

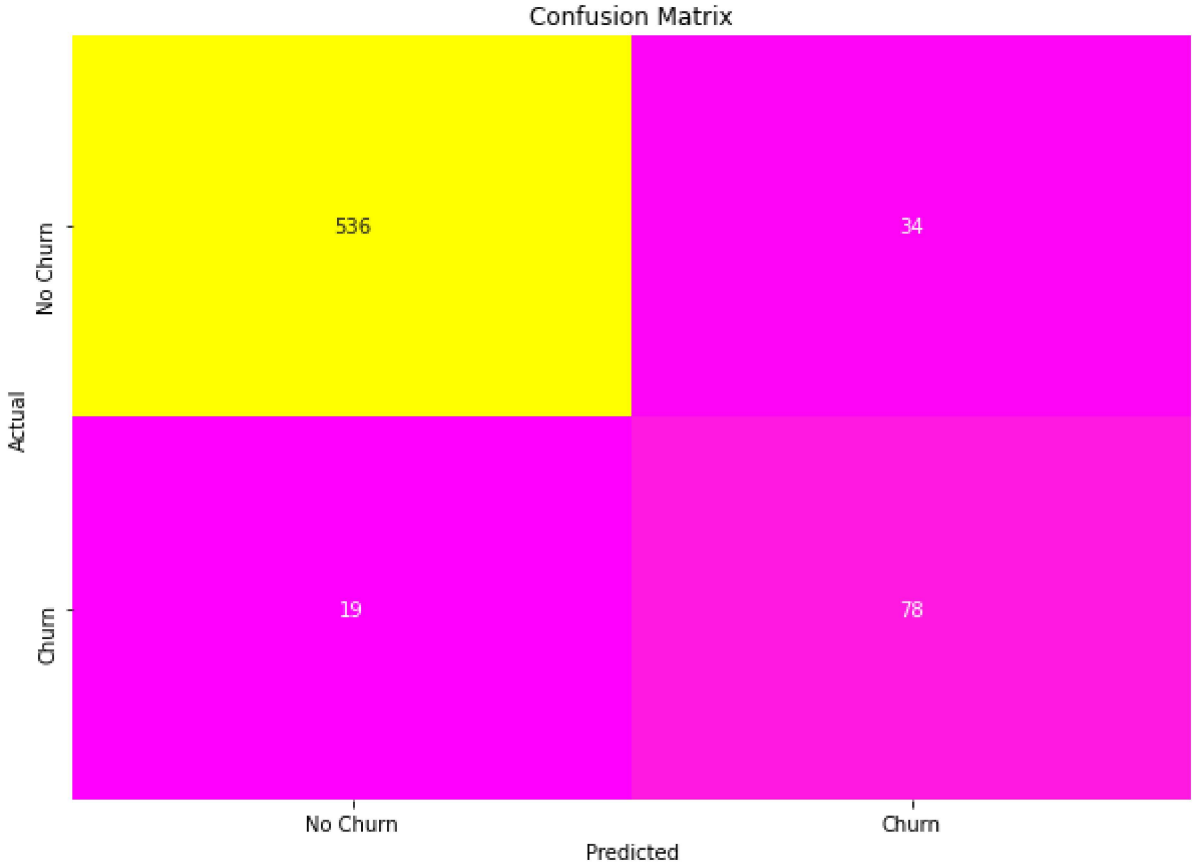
In [50]:
```
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize and train the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_model, param_grid=param_grid, cv=5, scoring='ac

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

print("Best Parameters:\n", best_params)
```

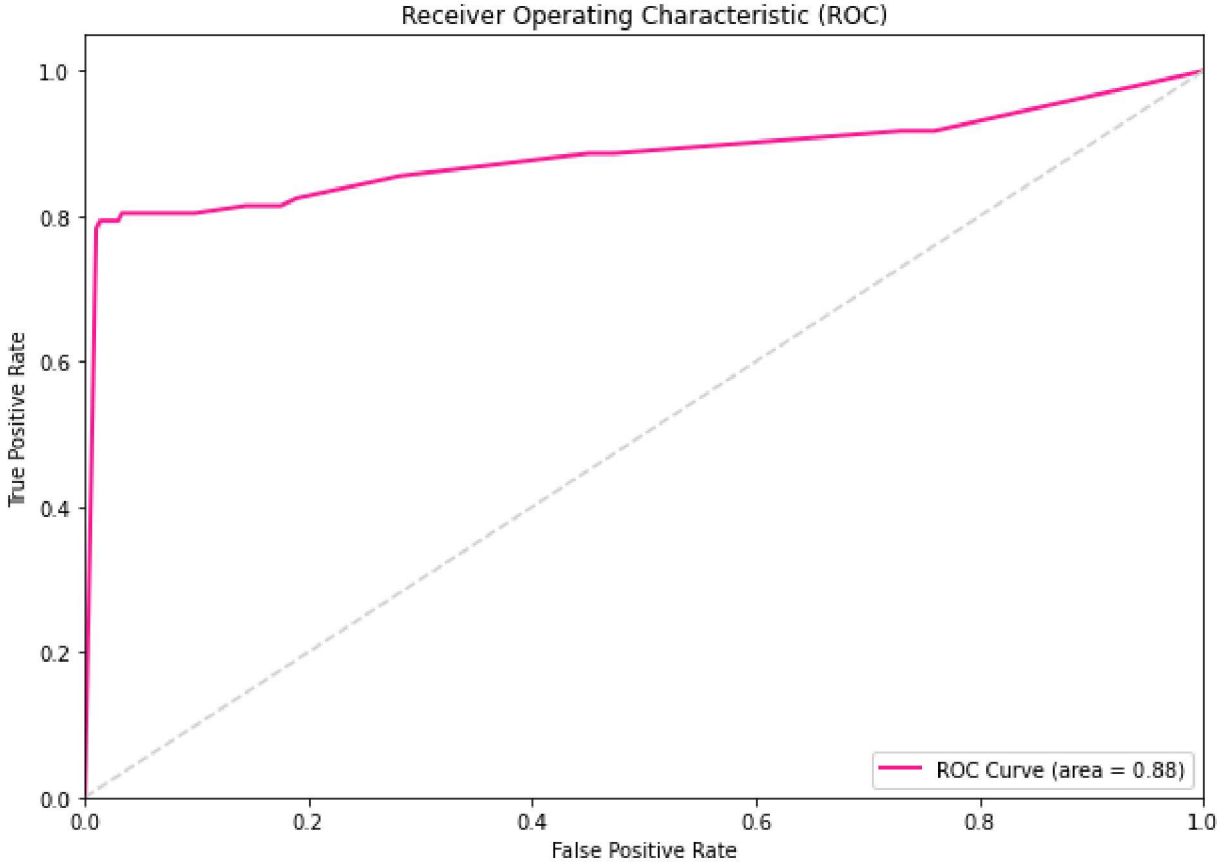
```
Fitting 5 folds for each of 72 candidates, totalling 360 fits
Best Parameters:
 {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1, 'min_samples_split': 10}
```

In [51]:
```
best_params = {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 1, 'min_sample
dt_model_best = DecisionTreeClassifier(**best_params, random_state=42)

# Train the model
dt_model_best.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
```

```
y_pred_best = dt_model_best.predict(X_test)
y_pred_prob_best = dt_model_best.predict_proba(X_test)[:, 1]  # Probabilities for ROC c

# Calculate metrics
accuracy_best = accuracy_score(y_test, y_pred_best)
roc_auc_best = roc_auc_score(y_test, y_pred_prob_best)
conf_matrix_best = confusion_matrix(y_test, y_pred_best)

print("Best Decision Tree Metrics with Best Parameters:")
print(f"Accuracy: {accuracy_best:.4f}")
print(f"ROC AUC: {roc_auc_best:.4f}")
print("Confusion Matrix:\n", conf_matrix_best)
print("Classification Report:\n", classification_report(y_test, y_pred_best))

# Plot Confusion Matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_best, annot=True, fmt='d', cmap='spring', cbar=False,
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()

# Plot ROC Curve
fpr, tpr, _ = roc_curve(y_test, y_pred_prob_best)
plt.figure(figsize=(10, 7))
plt.plot(fpr, tpr, color='deeppink', lw=2, label='ROC Curve (area = %0.2f)' % roc_auc_b
plt.plot([0, 1], [0, 1], color='lightgray', 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('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()
```

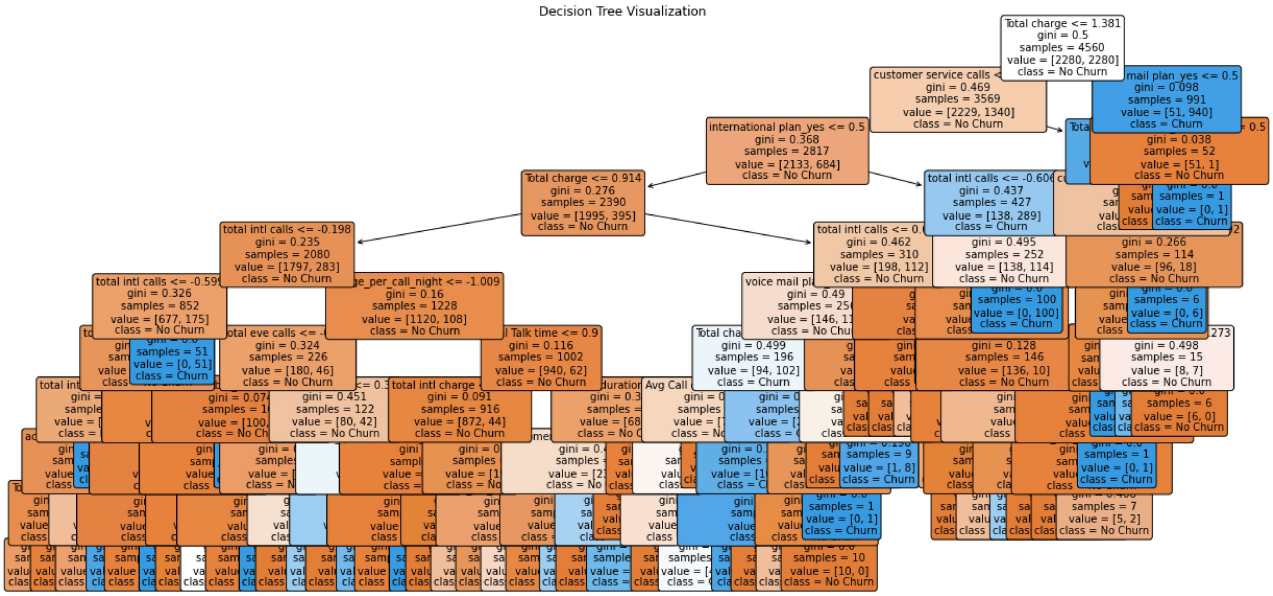
```
Best Decision Tree Metrics with Best Parameters:
Accuracy: 0.9205
ROC AUC: 0.8825
Confusion Matrix:
 [[536  34]
 [ 19  78]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.94      0.95       570
         1.0       0.70      0.80      0.75        97

    accuracy                           0.92       667
   macro avg       0.83      0.87      0.85       667
weighted avg       0.93      0.92      0.92       667
```





In [52]:
```
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt

# Visualize the decision tree
```

```
plt.figure(figsize=(20, 10))
plot_tree(dt_model_best, filled=True, feature_names=X_train_resampled.columns, class_na
plt.title('Decision Tree Visualization')
plt.show()
```



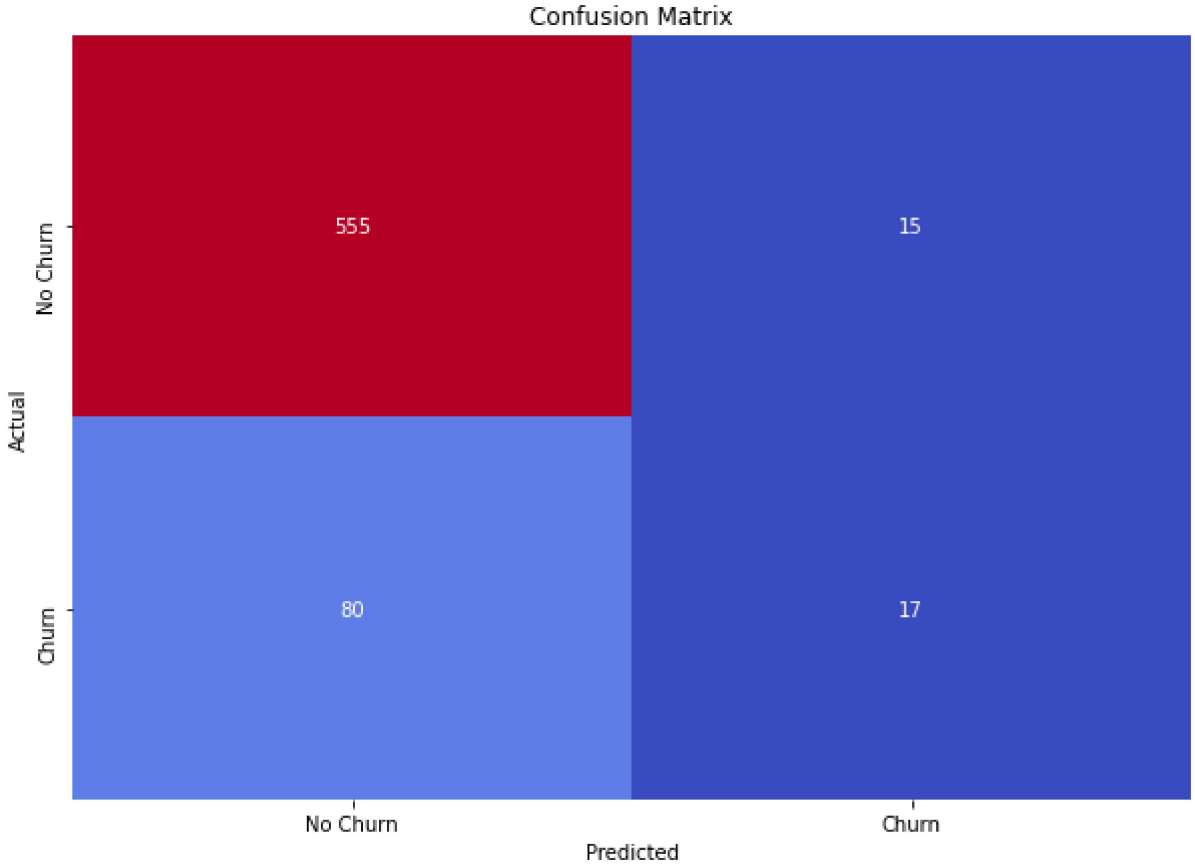
Logistic regression modelling evaluation after Hypertuning

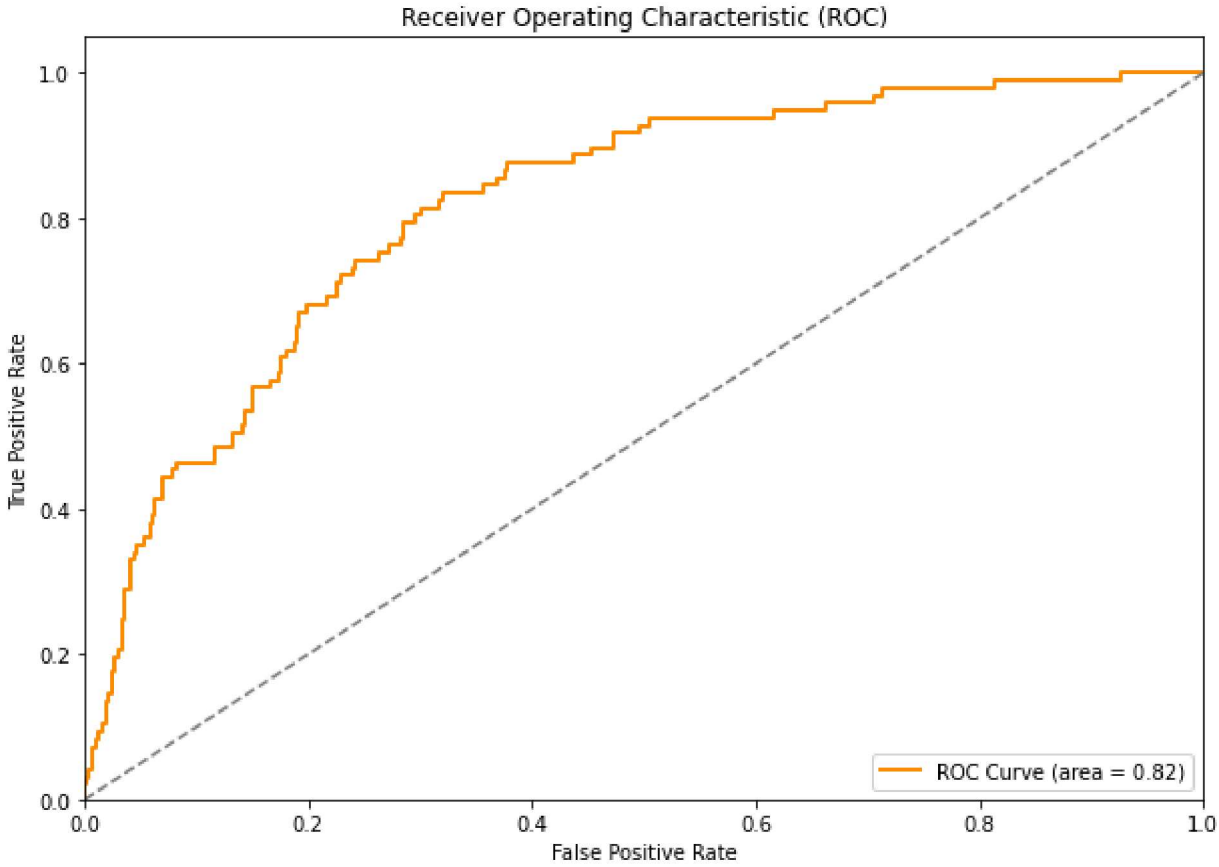
In [59]:

```
from sklearn.model_selection import RandomizedSearchCV
import time

def tune_logistic_regression(X_train, y_train, X_test, y_test):
    # Simplify the hyperparameters and their ranges
    param_grid = {
        'penalty': ['l2'],  # Simplified to only 'l2' for faster computation
        'C': [0.1, 1, 10],  # Reduced number of C values
        'solver': ['liblinear'],  # 'liblinear' is faster and works with 'l2'
        'max_iter': [100, 200]  # Reduced max iterations
    }

    # Initialize Logistic Regression model
    log_reg = LogisticRegression(random_state=42)

    # Set up RandomizedSearchCV
    random_search = RandomizedSearchCV(log_reg, param_grid, cv=3, scoring='roc_auc', n_

    # Time the model fitting process
    start_time = time.time()

    # Fit the model
    print("Starting RandomizedSearchCV...")
    random_search.fit(X_train, y_train)
    print(f"Completed RandomizedSearchCV in {time.time() - start_time:.2f} seconds.")

    # Get the best model
    best_log_reg = random_search.best_estimator_

    # Make predictions
    y_pred = best_log_reg.predict(X_test)
```

```
    y_pred_prob = best_log_reg.predict_proba(X_test)[:, 1]  # Probabilities for ROC cur

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_prob)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print("Best Logistic Regression Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"ROC AUC: {roc_auc:.4f}")
    print("Confusion Matrix:\n", conf_matrix)
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Plot Confusion Matrix
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='coolwarm', cbar=False,
                xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # Plot ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    plt.figure(figsize=(10, 7))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC Curve (area = %0.2f)' % roc
    plt.plot([0, 1], [0, 1], color='gray', 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('Receiver Operating Characteristic (ROC)')
    plt.legend(loc='lower right')
    plt.show()

    return best_log_reg, accuracy, roc_auc

# Call the function
best_log_reg_model, best_accuracy, best_roc_auc = tune_logistic_regression(X_train, y_t
```

```
Starting RandomizedSearchCV...
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Completed RandomizedSearchCV in 12.35 seconds.
Best Logistic Regression Metrics:
Accuracy: 0.8576
ROC AUC: 0.8176
Confusion Matrix:
 [[555  15]
 [ 80  17]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.87      0.97      0.92       570
         1.0       0.53      0.18      0.26        97

    accuracy                           0.86       667
   macro avg       0.70      0.57      0.59       667
weighted avg       0.82      0.86      0.83       667
```





Randorm Forest and Gradient Boosing modelling evaluation after hypertuning

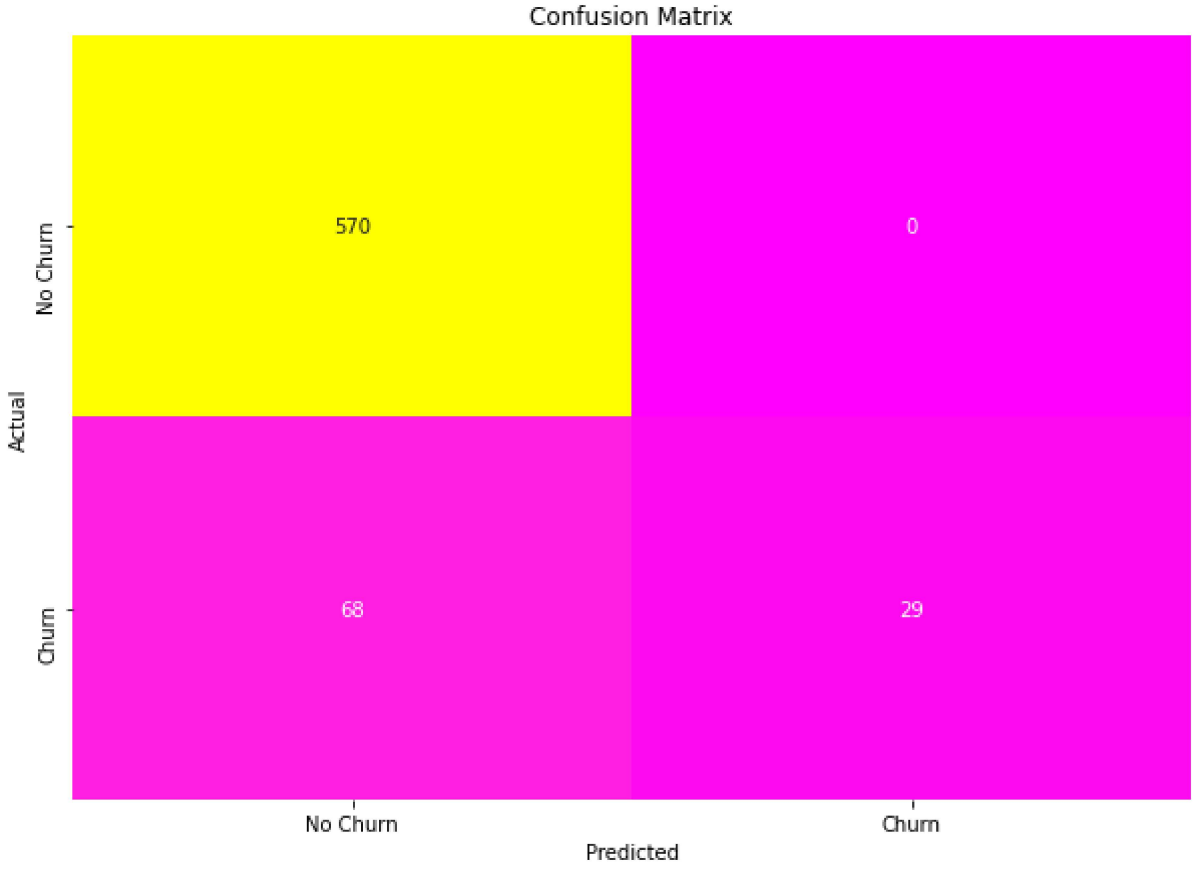
In [66]:
```
def tune_and_evaluate_model(model, param_grid, X_train, y_train, X_test, y_test, n_iter
    """
    Function to perform hyperparameter tuning using RandomizedSearchCV, train the model
    and plot the confusion matrix and ROC curve.

    Parameters:
    - model: the machine learning model to be tuned.
    - param_grid: dictionary containing hyperparameters and their ranges.
    - X_train: training features.
    - y_train: training labels.
    - X_test: testing features.
    - y_test: testing labels.
    - n_iter: number of parameter settings sampled during RandomizedSearchCV (default: 1
    - cv: number of cross-validation folds (default: 3).

    Returns:
    - best_model: the model with the best hyperparameters.
    - accuracy: accuracy of the best model on the test set.
    - roc_auc: ROC AUC score of the best model on the test set.
    """

    # Set up RandomizedSearchCV
    random_search = RandomizedSearchCV(model, param_grid, cv=cv, scoring='roc_auc', n_j

    # Time the model fitting process
    start_time = time.time()

    # Fit the model
    print(f"Starting RandomizedSearchCV for {model.__class__.__name__}...")
    random_search.fit(X_train, y_train)
    print(f"Completed RandomizedSearchCV in {time.time() - start_time:.2f} seconds.")

    # Get the best model
    best_model = random_search.best_estimator_
    print(f"Best Parameters Found: {random_search.best_params_}")

    # Make predictions
    y_pred = best_model.predict(X_test)
    y_pred_prob = best_model.predict_proba(X_test)[:, 1]  # Probabilities for ROC curve

    # Calculate metrics
    accuracy = accuracy_score(y_test, y_pred)
    roc_auc = roc_auc_score(y_test, y_pred_prob)
    conf_matrix = confusion_matrix(y_test, y_pred)

    print(f"Best {model.__class__.__name__} Metrics:")
    print(f"Accuracy: {accuracy:.4f}")
    print(f"ROC AUC: {roc_auc:.4f}")
    print("Confusion Matrix:\n", conf_matrix)
    print("Classification Report:\n", classification_report(y_test, y_pred))

    # Plot Confusion Matrix
    plt.figure(figsize=(10, 7))
    sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='spring', cbar=False,
                xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()
```

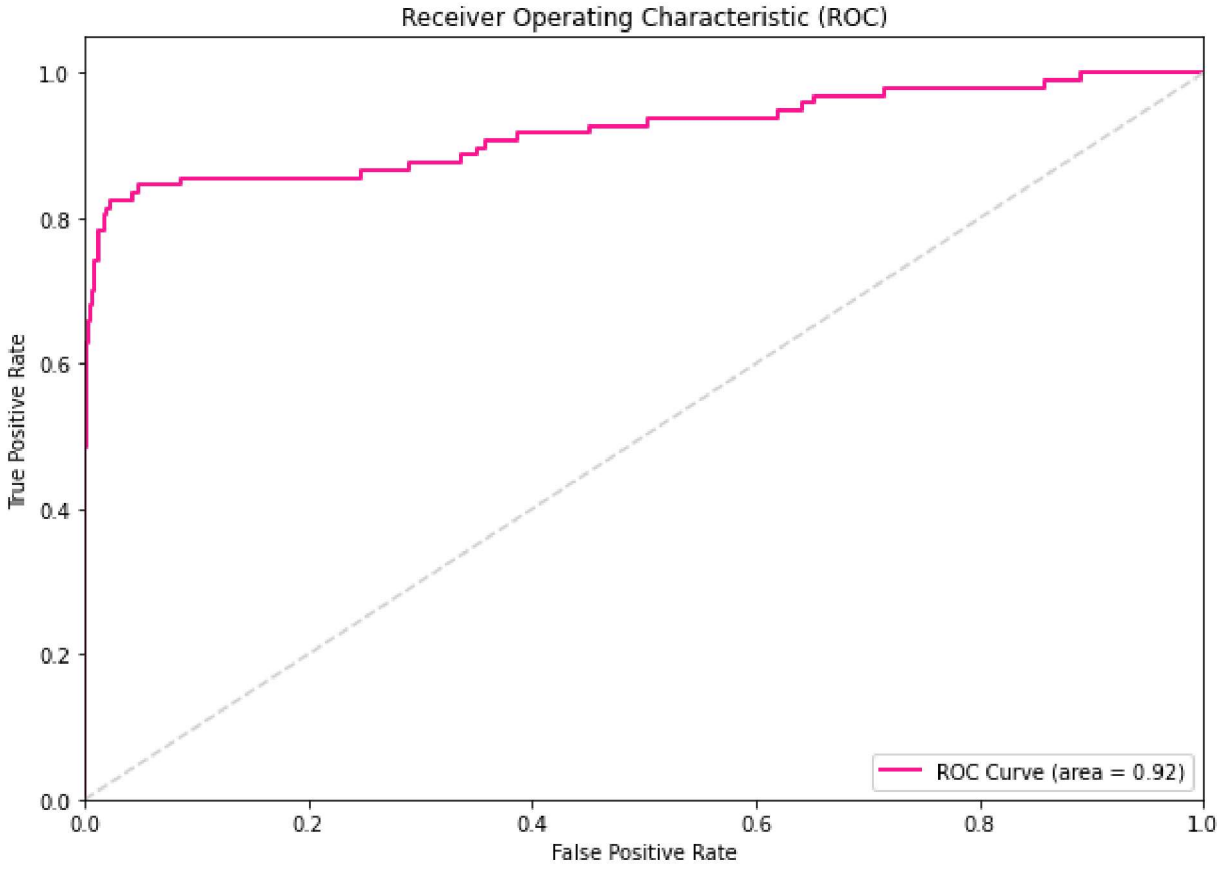
```
    # Plot ROC Curve
    fpr, tpr, _ = roc_curve(y_test, y_pred_prob)
    plt.figure(figsize=(10, 7))
    plt.plot(fpr, tpr, color='deeppink', lw=2, label=f'ROC Curve (area = {roc_auc:.2f})
    plt.plot([0, 1], [0, 1], color='lightgray', linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.legend(loc='lower right')
    plt.show()

    return best_model, accuracy, roc_auc
```

In [67]:

```
# Define the hyperparameters and their respective ranges
param_grid_rf = {
    'n_estimators': [100, 200, 500],  # Number of trees in the forest
    'max_depth': [10, 20, 30],        # Maximum depth of the tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an
    'min_samples_leaf': [1, 2, 4],    # Minimum number of samples required to be at a l
    'bootstrap': [True, False]        # Whether bootstrap samples are used when buildin
}

# Initialize Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Run hyperparameter tuning and evaluation
best_rf_model, best_accuracy_rf, best_roc_auc_rf = tune_and_evaluate_model(
    model=rf_model,
    param_grid=param_grid_rf,
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    n_iter=10,  # Number of parameter settings to sample
    cv=3        # Number of cross-validation folds
)
```

```
Starting RandomizedSearchCV for RandomForestClassifier...
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Completed RandomizedSearchCV in 127.98 seconds.
Best Parameters Found: {'n_estimators': 500, 'min_samples_split': 2, 'min_samples_leaf':
1, 'max_depth': 30, 'bootstrap': False}
Best RandomForestClassifier Metrics:
Accuracy: 0.8981
ROC AUC: 0.9207
Confusion Matrix:
 [[570   0]
 [ 68  29]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.89      1.00      0.94       570
         1.0       1.00      0.30      0.46        97

    accuracy                           0.90       667
   macro avg       0.95      0.65      0.70       667
weighted avg       0.91      0.90      0.87       667
```

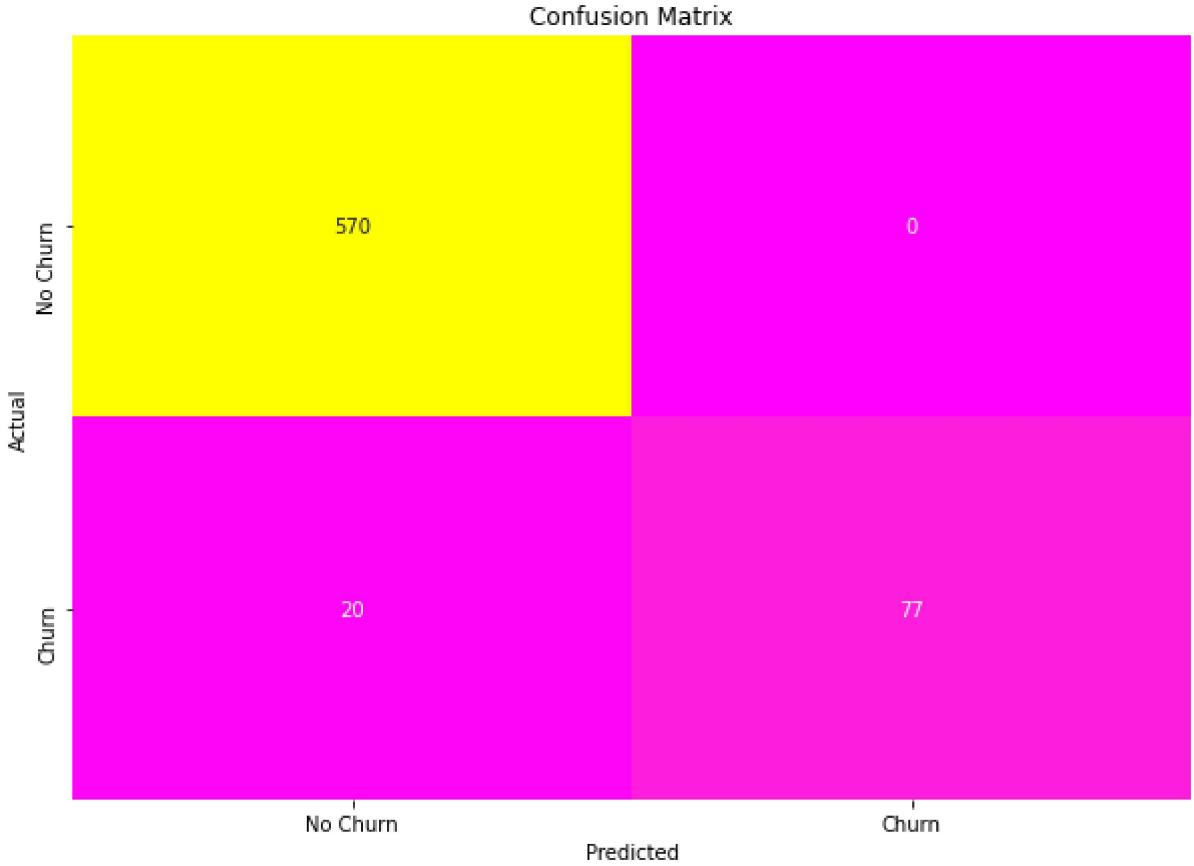


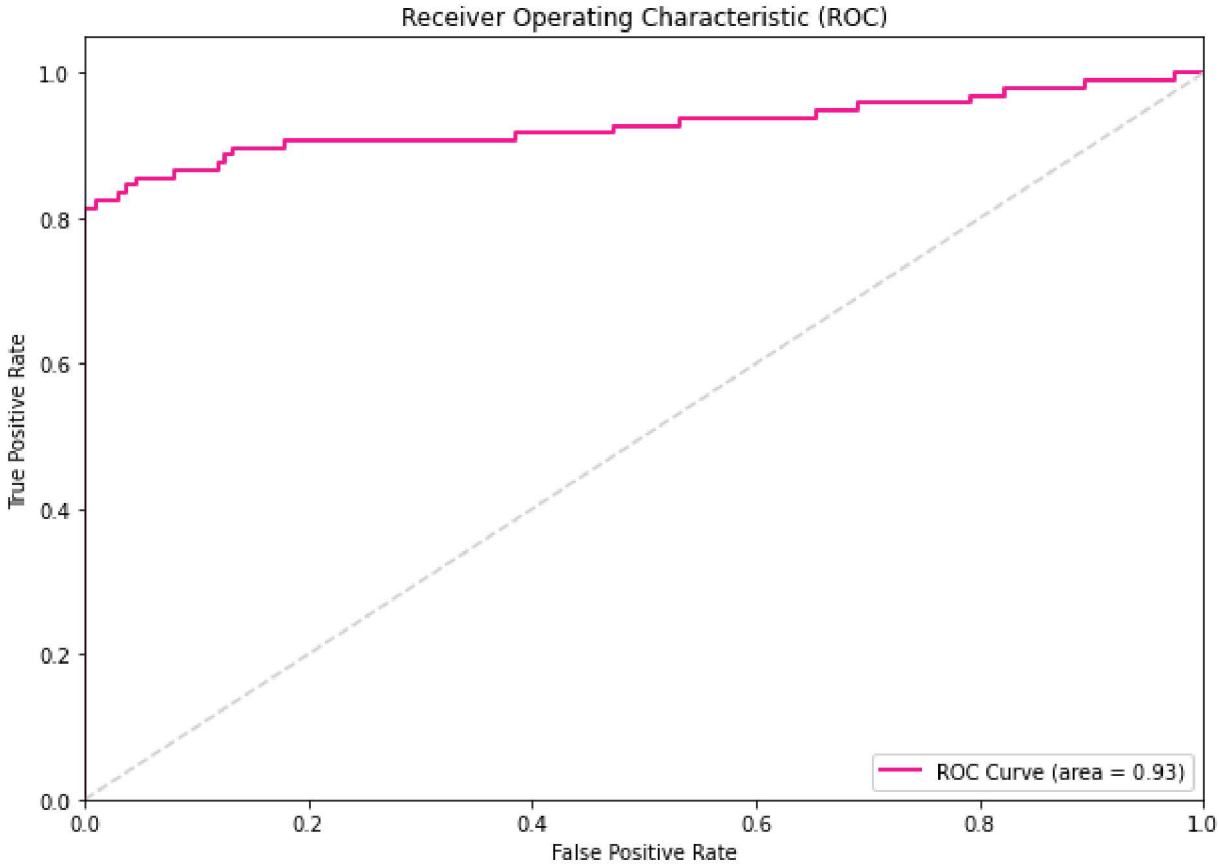


In [68]:
```
# Define the hyperparameters and their respective ranges
param_grid_gb = {
    'n_estimators': [100, 200, 300],  # Number of boosting stages
    'learning_rate': [0.01, 0.1, 0.2],  # Learning rate shrinks the contribution of eac
```

```
    'max_depth': [3, 5, 7],           # Maximum depth of the individual trees
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an
    'min_samples_leaf': [1, 2, 4]     # Minimum number of samples required to be at a l
}

# Initialize Gradient Boosting model
gb_model = GradientBoostingClassifier(random_state=42)

# Run hyperparameter tuning and evaluation
best_gb_model, best_accuracy_gb, best_roc_auc_gb = tune_and_evaluate_model(
    model=gb_model,
    param_grid=param_grid_gb,
    X_train=X_train,
    y_train=y_train,
    X_test=X_test,
    y_test=y_test,
    n_iter=10,  # Number of parameter settings to sample
    cv=3        # Number of cross-validation folds
)
```

```
Starting RandomizedSearchCV for GradientBoostingClassifier...
Fitting 3 folds for each of 10 candidates, totalling 30 fits
Completed RandomizedSearchCV in 428.46 seconds.
Best Parameters Found: {'n_estimators': 300, 'min_samples_split': 5, 'min_samples_leaf':
4, 'max_depth': 5, 'learning_rate': 0.1}
Best GradientBoostingClassifier Metrics:
Accuracy: 0.9700
ROC AUC: 0.9281
Confusion Matrix:
 [[570   0]
 [ 20  77]]
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      1.00      0.98       570
         1.0       1.00      0.79      0.89        97

    accuracy                           0.97       667
   macro avg       0.98      0.90      0.93       667
weighted avg       0.97      0.97      0.97       667
```





Interpreting the Findings After Hyperparameter Tuning Hyperparameter Tuning Impact:

Decision Tree: Hyperparameter tuning reduced the accuracy slightly (0.9685 to 0.9205) but might have improved performance on specific classes based on the F1-score in the classification report.

Random Forest: Tuning resulted in a slight improvement in accuracy (0.9130 to 0.8981) but a significant increase in ROC AUC (0.7053 to 0.9207), indicating a better ability to distinguish between churners and non-churners. However, recall for class 1 (churners) remains low (0.30). Gradient Boosting: Tuning maintained the high accuracy (0.9700) while slightly improving the ROC AUC (0.8918 to 0.9281). Logistic Regression: Tuning had minimal impact on performance (accuracy remained around 0.86). Comparing Model Performance:

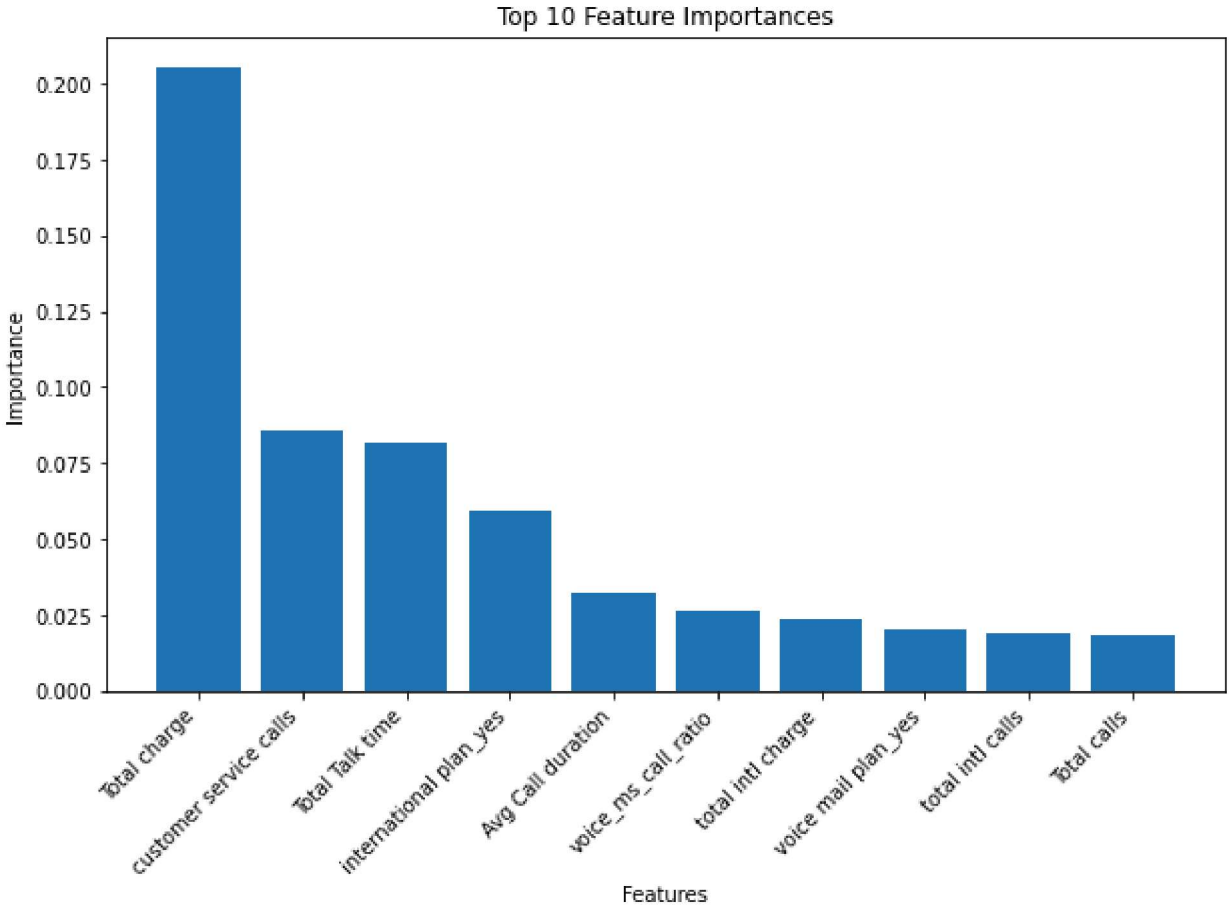
Overall Accuracy: Gradient Boosting remains the most accurate model (0.9700). ROC AUC: Random Forest has the highest ROC AUC (0.9207), indicating the best ability to distinguish churners from non-churners. Precision and Recall: Precision: Gradient Boosting and Decision Tree have high precision for class 0 (non-churners). Random Forest has high precision for class 0 but struggles with class 1. Recall: Gradient Boosting achieves perfect recall for class 0. Both Decision Tree and Gradient Boosting have improved recall for class 1 compared to the untuned models. Key Insights:

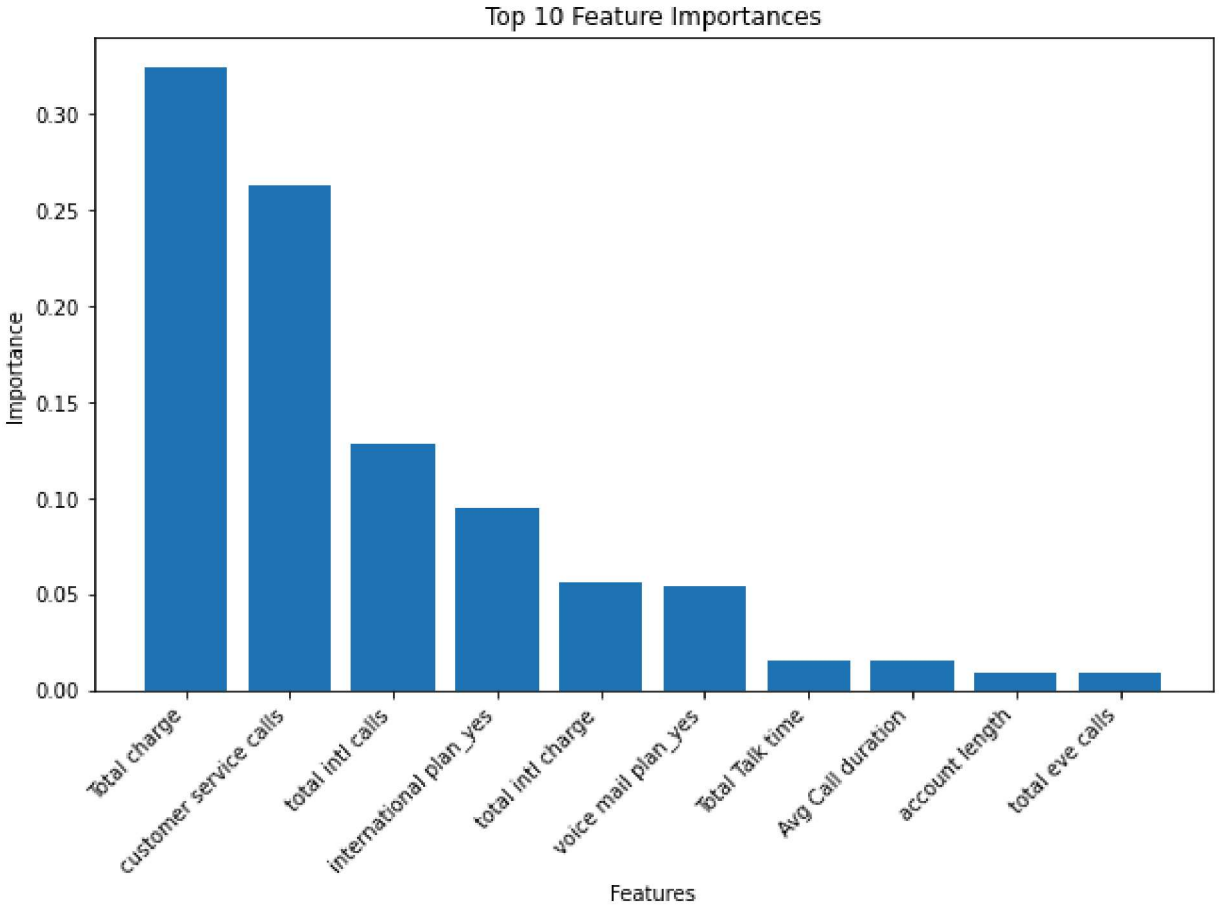
Gradient Boosting: This model maintains excellent overall accuracy, good ROC AUC, and perfect recall for non-churners. It might be the best choice if these factors are critical. Random Forest: While it has slightly lower accuracy, its high ROC AUC suggests good discrimination between classes. However, the low recall for churners needs further investigation. Decision Tree: Tuning improved performance for class 1, but it might not be the best choice due to lower overall performance compared to Gradient Boosting. Logistic Regression: While interpretable, its performance is lower than the other models.

In [71]:

```
import numpy as np
import matplotlib.pyplot as plt

def plot_feature_importances(model, X_train, top_n=10):
    """
    Plot the top N feature importances for a given model.

    Parameters:
    - model: The trained model with `feature_importances_` attribute.
    - X_train: The training features to get feature names.
    - top_n: Number of top features to display.
    """
    # Get feature importances
    importances = model.feature_importances_

    # Get indices of top N features
    indices = np.argsort(importances)[::-1]
    top_indices = indices[:top_n]

    # Plot top N feature importances
    plt.figure(figsize=(10, 6))
    plt.title(f"Top {top_n} Feature Importances")
    plt.bar(range(top_n), importances[top_indices], align="center")
    plt.xticks(range(top_n), X_train.columns[top_indices], rotation=45, ha='right')
    plt.xlabel("Features")
    plt.ylabel("Importance")
    plt.show()

# Plot for Random Forest
plot_feature_importances(best_rf_model, X_train, top_n=10)
```

```
# Plot for Decision Tree
plot_feature_importances(dt_model_best, X_train, top_n=10)
```





The key features identified in the feature importance plot are:

Total charge: This is the most important feature, indicating that the total amount charged to customers is a significant factor in predicting churn or another outcome.

Customer service calls: The number of customer service calls made is also a crucial feature, suggesting that customer support interactions play a significant role in customer satisfaction and retention.

Total talk time: The total amount of time customers spend on calls is another important feature, which may be related to usage patterns and satisfaction.

International plan_yes: Having an international plan is a moderately important feature, suggesting that customers with international plans may have different churn behaviors.

Avg call duration: The average duration of calls is also a relevant feature, possibly indicating customer satisfaction or usage patterns. voice_ms_call_ratio: This feature, likely related to voicemail usage, is moderately important.

Total intl charge: The total amount spent on international calls is a moderately important feature.

Voice mail plan yes: Having a voicemail plan is a moderately important feature.

Total intl calls: The total number of international calls is a moderately important feature.

Total calls: The total number of calls made is a moderately important feature.

Conclusion

Recommendations

Customer Service Enhancement:

Focus on Service Calls: Since customer service calls is a key feature, investigate why customers are reaching out to support. Improve customer service quality and reduce the need for multiple service calls. Feedback Collection: Collect feedback from customers who frequently call support to identify common issues and address them proactively. Usage Patterns Analysis:

Analyze Call Duration and Total Talk Time: Customers with longer call durations or higher total talk time might have specific usage patterns. Offer targeted plans or incentives based on usage patterns. Monitor Average Call Duration: If customers with longer call durations are more likely to churn, evaluate the quality of interactions during longer calls to ensure they are positive experiences. Plan-Specific Strategies:

International Plans: Since having an international plan is a significant feature, offer tailored support or promotional offers to customers with international plans to retain them. Voice Mail Plans: For customers with voicemail plans, ensure the feature meets their expectations and offer enhancements if needed. Billing and Charges:

Total Charge and International Charges: Customers with high total charges or international charges may have different churn behaviors. Review billing practices and consider offering discounts or loyalty programs for high-spending customers. Monitor Total Calls and International Calls: High numbers of calls could indicate high engagement, which is positive. Ensure that billing aligns with customer expectations. Communication and Retention Programs:

Targeted Campaigns: Use insights from features like voice_ms_call_ratio and total intl charge to create targeted marketing campaigns that address specific customer needs or offer retention incentives. Churn Prevention: Implement churn prevention strategies for customers showing patterns similar to high-churn groups (e.g., high number of service calls).

Future Work

Feature Engineering:

Explore New Features: Investigate additional features that could provide more insights, such as customer demographics or satisfaction scores. Interaction Terms: Create interaction terms between key features (e.g., Total charge and Total calls) to capture complex relationships. Model Improvement:

Feature Selection: Use advanced feature selection techniques (e.g., Recursive Feature Elimination) to refine the feature set and improve model performance. Hyperparameter Tuning: Continue tuning hyperparameters for the models to ensure optimal performance and avoid overfitting. Model Comparison:

Test Additional Models: Explore other machine learning models such as XGBoost, LightGBM, or neural networks to compare their performance against the current models. Cross-Validation: Implement cross-validation with more folds to ensure robustness and generalizability of the models. Customer Segmentation:

Segmentation Analysis: Perform customer segmentation to tailor strategies based on different customer profiles or behavior patterns. Cluster Analysis: Use clustering algorithms (e.g., K-means) to identify distinct customer groups and tailor retention strategies accordingly. Monitoring and Feedback Loop:

Implement a Feedback Loop: Continuously monitor the effectiveness of the implemented strategies and adjust based on new data and feedback. A/B Testing: Conduct A/B testing for different retention strategies to evaluate their impact and optimize accordingly. By focusing on these recommendations and future work areas, you can enhance customer retention efforts, improve model accuracy, and make data-driven decisions to better understand and address churn.