SyriaTel Customer Churn Prediction

1. Business Understanding

1.1 Business Overview ¶

SyriaTel is a telecommunications company that provides mobile services such as voice calls, messaging and data plans. As a provider in the competitive telecom industry, its operations rely on recurring subscription revenue from customers.

A major business challenge in the telecom industry is customer churn, where subscribers discontinue their service. Churn reduces revenue and increases costs since acquiring new customers requires more investment than retaining existing ones. Addressing churn is therefore important for SyriaTel to maintain profitability and remain competitive in the market.

1.2 Problem Statement

Customer churn is a key challenge for SyriaTel as losing subscribers reduces revenue and increases the cost of acquiring replacements. To remain competitive, the company needs to identify customers who are likely to leave and intervene before they discontinue their service.

To address this challenge, the project builds a classification model that predicts churn and highlights the main factors influencing it. The insights will support SyriaTel in reducing churn, improving customer retention and strengthening long term customer loyalty and profitability.

1.3 Business Objectives

1.3.1 Main Objective

The main objective of this project is to develop a classification model that predicts customer churn for SyriaTel and identifies the key factors influencing it. The results will help the company reduce churn, improve customer retention and protect long term profitability.

1.3.2 Specific Objectives

To achieve the main objective, the project has the following specific objectives:

1. Develop a churn prediction model and evaluate its performance using relevant metrics.
2. Identify the key features and customer characteristics that significantly influence churn.
3. Compare different classification models to select the one that best balances predictive performance with business needs.

1.3.3 Research Questions

To ensure the analysis directly addresses SyriaTel's business problem, the following research questions were defined:

1. Can a churn prediction model achieve strong performance?
2. Which features and customer characteristics have the greatest influence on churn?
3. Which classification model provides the best balance between predictive performance and business applicability?

The notebook will answer these research questions through data preparation, model development, evaluation and interpretation to provide data driven insights that support customer retention strategies

1.4 Success Criteria

The success of this project will be evaluated using the following criteria:

- Technical Success: The churn prediction model should achieve a recall score of at least 0.80, ensuring that most customers likely to churn are correctly identified. Supporting metrics such as precision, F1-score and ROC-AUC will also be used to confirm balanced model performance.
- Business Success: The model should produce clear insights into the main factors that drive churn, enabling SyriaTel to better understand customer behavior and support decisions that improve retention and profitability.

2. Data Understanding

This section introduces the SyriaTel Customer Churn dataset used for the project. The dataset contains customer account information, service plan details, usage patterns and churn labels which provide the basis for building predictive models.

The aim is to understand the structure and contents of the dataset. This involves reviewing the available features, checking their data types and identifying potential issues such as missing values or unusual patterns.

By exploring the data at this stage, it is possible to detect quality concerns early and begin considering how the dataset can best be prepared for modeling to answer the business questions.

2.1. Loading and Previewing the Dataset

The necessary Python libraries for data handling, visualization and modeling are imported. These include pandas, NumPy, Seaborn, Matplotlib and Scikit-learn, as well as utilities for evaluation metrics and class imbalance.

The SyriaTel Customer Churn dataset is then loaded from a CSV file into a pandas DataFrame. Basic inspection functions are applied to confirm successful loading and to preview the structure of the dataset.

2.1.1 Importing Required Libraries

In [1]:
```
# Data loading and manipulation
import pandas as pd
import numpy as np

# Data visualization
import seaborn as sns
import matplotlib.pyplot as plt

# Data preprocessing and modeling
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import OneHotEncoder, MinMaxScaler, LabelEncoder
from scipy import stats
import random
from sklearn.metrics import accuracy_score, f1_score, precision_score, reca

# Algorithms for supervised learning
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClass
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier

# Suppress warnings for better readability
import warnings
warnings.filterwarnings('ignore')

# Set the seaborn plot size
sns.set_theme(rc={'figure.figsize':(11.7,8.27)})

# Custom function loading
import importlib

# Import module
import utility

# Reloads the module to reflect changes
importlib.reload(utility)
```

Out[1]:
```
<module 'utility' from 'c:\\Users\\user\\Downloads\\Moringa School\\Phase
s\\Phase 3\\Phase 3 Project\\Augustine\\utility.py'>
```

2.1.2 Loading the Dataset

In this section, the file is loaded into a pandas DataFrame.

In [2]:
```
# Loading the dataset

churn_df = pd.read_csv('churn.csv')
```

2.1.3 Previewing the Data

In [3]:
```
# Previewing the first five rows
churn_df.head()
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...

5 rows × 21 columns

Each row in the dataset represents a SyriaTel customer account. The columns include customer attributes such as state, account length, service plans, call and usage statistics, charges and churn status.

2.2 Data Overview

This section covers the initial overview of the dataset.

In [4]:
```
# Checking the shape of the dataset

churn_df.shape
```

Out[4]:
```
(3333, 21)
```

The dataset has 3333 rows and 21 columns.

In [5]:
```
# Displaying the data types

churn_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
```

All the columns in the dataset are non null.

In [6]:
```
# Displaying the numerical and categorical columns

print(f"Numerical columns: {churn_df.select_dtypes(include='number').columr
print(f"Categorical columns: {churn_df.select_dtypes(include='object').colu
```

```
Numerical columns: Index(['account length', 'area code', 'number vmail me
ssages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls'],
      dtype='object')

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

Below is a description of all the numerical and categorical features in the dataset:

Numerical Features:

- account length: The number of days the customer has been an account holder.
- area code: The area code associated with the customer's phone number.

- number vmail messages: The number of voice messages received by the customer.
- total day minutes: The total number of minutes used by the customer during the day.
- total day calls: The total number of calls made by the customer during the day.
- total day charge: The total charges incurred by the customer during the day.
- total eve minutes: The total number of minutes used by the customer in the evening.
- total eve calls: The total number of calls made by the customer in the evening.
- total eve charge: The total charges incurred by the customer in the evening.
- total night minutes: The total number of minutes spent by the customer at night.
- total night calls: The total number of calls made by the customer at night.
- total night charge: The total charged incurred by the customer at night.
- total intl minutes: The total number of minutes spent by the customer on international calls.
- total intl calls: The total number of international calls made by the customer.
- total intl charge: The total charge incurred by the customer on international calls.
- customer service calls: The number of calls made by customer service to customers.

Categorical Features:

- state: The customer's state of residence.
- phone number: The customer's mobile number.
- international plan: Indicates if the customer has subscribed to an international plan (Yes/No)
- voice mail plan: Indicates if the customer has a voice mail plan (Yes/No)

2.3 Descriptive Statistics

This section summarizes the numerical and categorical features, highlighting central tendencies, variability and potential outliers to guide further analysis.

In [7]:
```
# Displaying summary statistics

churn_df.describe()
```

Out[7]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	to m
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.9
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.7
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7

The statistical overview of numeric columns in the dataset is as follows:

- The account length ranges from 1 to 243 days, with an average of about 101 days, showing wide variation in how long customers have been with SyriaTel.
- The number of voicemail messages has a mean of 9 but a median of 0, indicating that most customers do not use voicemail services.
- The total day minutes range from 0 to 350.8 minutes, with an average of 179.8, showing some customers have no daytime usage while others record high activity.
- The international minutes average only 2.8 minutes, with many customers at 0, reflecting low international call usage overall.
- The customer service calls range from 0 to 9, with a median of 1, suggesting that most customers rarely contact support, while a smaller group reaches out more often.

These insights are useful for identifying customer behavior patterns and signal which features

In [8]:
```
# Checking for missing values

churn_df.isnull().sum()
```

Out[8]:
```
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
```

After check, it's evident that the dataset has zero missing values

2.4 Key Observations

Based on the initial loading and structure review of the dataset, the following key observations were made:

- The dataset contains 3,333 rows and 21 columns which include customer demographics, service plans, usage patterns, charges and churn status.
- No missing values are present across the dataset which ensures completeness for analysis.
- Numerical features such as usage minutes and charges show wide variability with some extreme values that may represent outliers.
- The number of voicemail messages is highly skewed as most customers do not use voicemail services.

- The churn column which is the target variable will require further examination to assess class balance before modeling.

These observations provide an initial understanding of data quality and structure. In addition

3. Data Preparation

In this section, we will look into data cleaning techniques, Exploratory Data Analysis (EDA) and data preprocessing (data wrangling) for our dataset. This step is paramount to provide data that will contribute significantly to the performance of the prediction model

3.1 Data Cleaning

In this step, basic data cleaning is done to ensure the dataset is consistent and ready for modeling. The process involves:

- Checking for and handling null values
- Identifying and removing duplicate rows
- Standardizing column names by capitalizing words and separating them with underscores

These tasks are completed using the clean_nulls_and_duplicates function from the utility.py file.

In [9]:
```
# Importing the clean_nulls_and_duplicates function

from utility import clean_nulls_and_duplicates

# Pass in the churn_df dataframe

churn_df = clean_nulls_and_duplicates(churn_df)
```

```
Initial shape of the dataset: (3333, 21)

No null values detected.

No duplicate rows detected.
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')

 Final shape of data: (3333, 21)
```

In [10]:
```
# Dropping the Phone_Number column

churn_df = churn_df.drop('Phone_Number', axis=1)

# Checking the remaining columns

churn_df.columns
```

Out[10]:
```
Index(['State', 'Account_Length', 'Area_Code', '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')
```

The Phone_Number column was removed since it does not contribute to churn prediction. This feature acts only as an identifier and has no analytical value.

After dropping the column, the dataset now has 3,333 rows and 20 columns, ensuring that only useful attributes remain for further preparation and modeling.

In [11]:
```
# Converting 'Area_Code' into an object datatype

churn_df['Area_Code'] = churn_df['Area_Code'].astype(object)
print(churn_df['Area_Code'].dtype)
```

```
object
```

The Area_Code column is stored as an integer but represents identifiers for regions. It is converted into a categorical; object data type to reflect its role as a categorical feature. This prevents incorrect interpretation as a continuous numerical variable.

3.2 Exploratory Data Analysis

This section investigates the dataset to get insights, evaluate feature distributions, assess relationships and detect issues or outliers. The findings guide feature engineering, support modeling decisions and show potential challenges with data quality.

3.2.1. Univariate Analysis

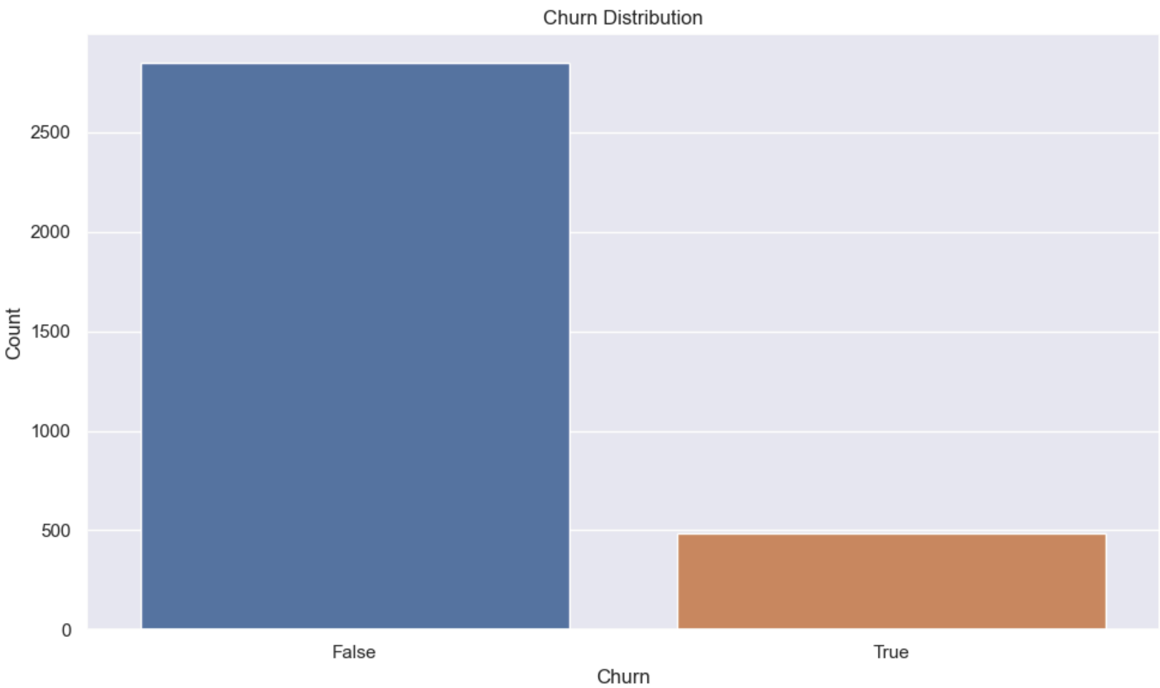
Univariate analysis explores each feature individually to understand its distribution, central tendency and spread. It helps detect issues, outliers or inconsistencies that may affect modeling.

3.2.1.1 Churn Distribution

The distribution of the target variable Churn is examined to check class balance. Both absolute counts and percentages are displayed followed by a bar plot for visualization.

In [12]:
```
# Distribution of the Churn target column

# Checking the distribution of the unique values

print("Churn counts:")
print(churn_df['Churn'].value_counts())

# Checking for normalized counts as a percentage

print("\nChurn distribution (%):")
print(churn_df['Churn'].value_counts(normalize=True) * 100)

# Visualizing with a bar plot

plt.figure(figsize=(10, 6))
sns.countplot(x='Churn', data=churn_df, palette='deep')
plt.title('Churn Distribution')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

```
Churn counts:
False    2850
True      483
Name: Churn, dtype: int64

Churn distribution (%):
False    85.508551
True     14.491449
Name: Churn, dtype: float64
```



The churn distribution shows that out of 3,333 customers, 483 customer about 14.5% have churned while 2,850 customers about 85.5% have not churned. This indicates a high class imbalance with non churned customers making up the majority.

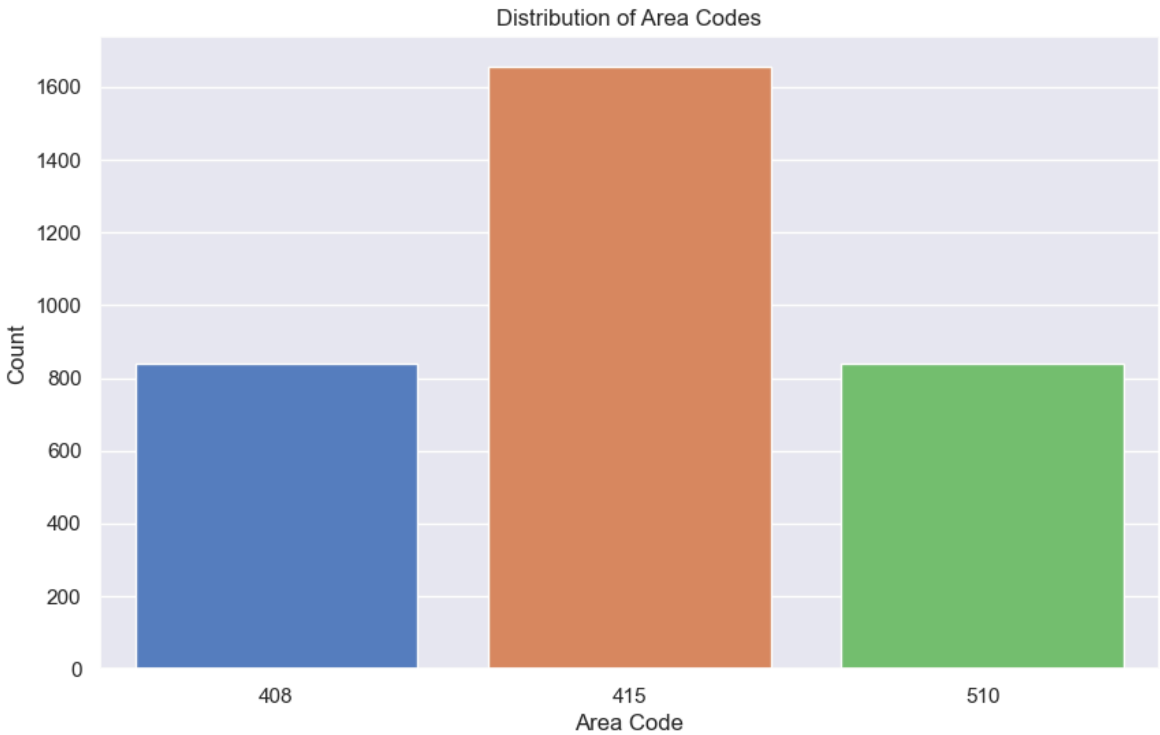
Such imbalance is important to note because it can bias machine learning models toward

3.2.1.2 Area Code Distribution

This section examines how customers are distributed across area codes to identify regions with the highest customer representation. The findings help show regional patterns that may impact service usage and churn behavior.

In [13]:
```
# Checking the distribution of the unique values

print("Area code counts:")
print(churn_df['Area_Code'].value_counts())

# Checking for normalized counts as a percentage

print("\nArea code distribution (%):")
print(churn_df['Area_Code'].value_counts(normalize=True) * 100)

# Countplot of area code

plt.figure(figsize=(10, 6))
sns.countplot(x='Area_Code', data=churn_df, palette='muted')
plt.title('Distribution of Area Codes')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.show()
```

```
Area code counts:
415    1655
510     840
408     838
Name: Area_Code, dtype: int64

Area code distribution (%):
415    49.654965
510    25.202520
408    25.142514
Name: Area_Code, dtype: float64
```



The distribution shows that most customers belong to area code 415 with about 50% while area codes 408 and 510 account for the remaining share at about 25% each.

This shows that nearly half of the customer base is concentrated in a single area code which may have implications for service management and churn behavior.

3.2.1.3 Distribution of categorical features

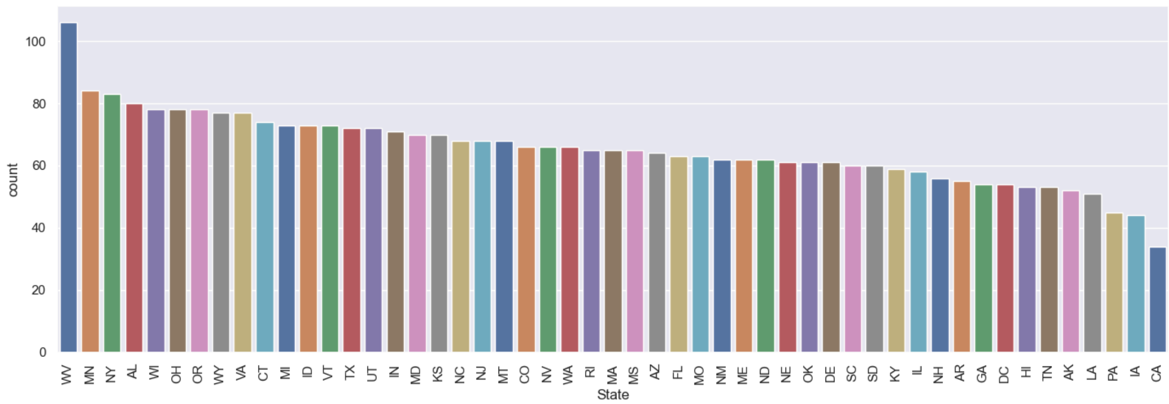
This section explores the distribution of three categorical features in the dataset: State, International_Plan and Voice_Mail_Plan. Analyzing these variables provides insights into customer demographics and service subscriptions that may influence churn.

The function plot_categorical_distributions from utility.py is used to generate the distributions for each feature.

State Feature

In [14]:
```
# Checking the distribution of the unique values

print("State counts:")
print(churn_df['State'].value_counts().head())

# Importing the function from utility.py

from utility import categorical_distributions

# Pass in the dataframe and the State feature

categorical_distributions(churn_df, 'State')
```

```
State counts:
WV    106
MN     84
NY     83
AL     80
WI     78
Name: State, dtype: int64
```



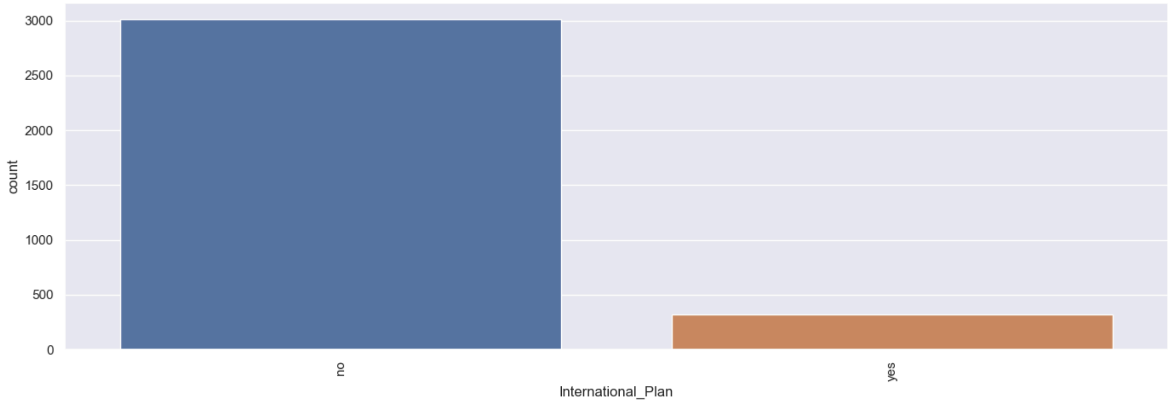
The State feature shows how customers are distributed across different states. The distribution is balanced with no state contributing an excessively large share. West Virginia (106), Minnesota (84) and New York (83) have the highest counts while states like California and Louisiana record the lowest.

This spread shows that SyriaTel's customer base is widely distributed across the U.S. which reduces the risk of regional bias in churn prediction and makes it possible to conduct data analysis.

International Plan Feature

In [15]:
```
# Checking the distribution of the unique values

print("International Plan counts:")
print(churn_df['International_Plan'].value_counts())

# Checking for normalized counts as a percentage

print("\nInternational Plan distribution (%):")
print(churn_df['International_Plan'].value_counts(normalize=True) * 100)

# Pass in the dataframe and the International_Plan feature

categorical_distributions(churn_df, 'International_Plan')
```

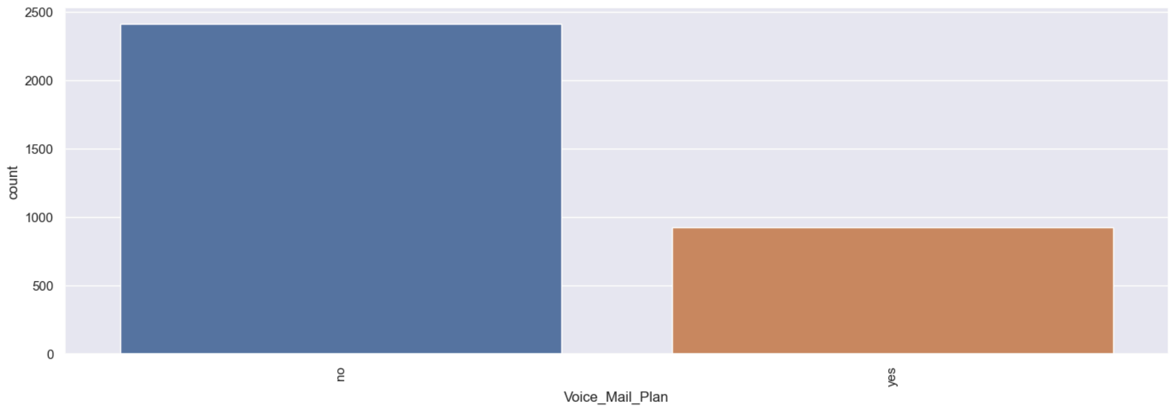
```
International Plan counts:
no     3010
yes     323
Name: International_Plan, dtype: int64

International Plan distribution (%):
no     90.309031
yes     9.690969
Name: International_Plan, dtype: float64
```



Voice Mail Feature

In [16]:
```
# Checking the distribution of the unique values

print("Voice Mail Plan counts:")
print(churn_df['Voice_Mail_Plan'].value_counts())

# Checking for normalized counts as a percentage

print("\nVoice Mail Plan distribution (%):")
print(churn_df['Voice_Mail_Plan'].value_counts(normalize=True) * 100)

# Load the categorical distributions function and pass in the arguments

categorical_distributions(churn_df, 'Voice_Mail_Plan')
```

```
Voice Mail Plan counts:
no     2411
yes     922
Name: Voice_Mail_Plan, dtype: int64

Voice Mail Plan distribution (%):
no     72.337234
yes    27.662766
Name: Voice_Mail_Plan, dtype: float64
```



From the plot the following are observed:

- 72.3% or about 2411 customers do not have a Voice Mail Plan.
- 27.7% or about 922 customers use the plan.
- Adoption is higher than the International Plan showing voicemail is more common.

The feature may influence churn as subscribers could have different usage patterns.

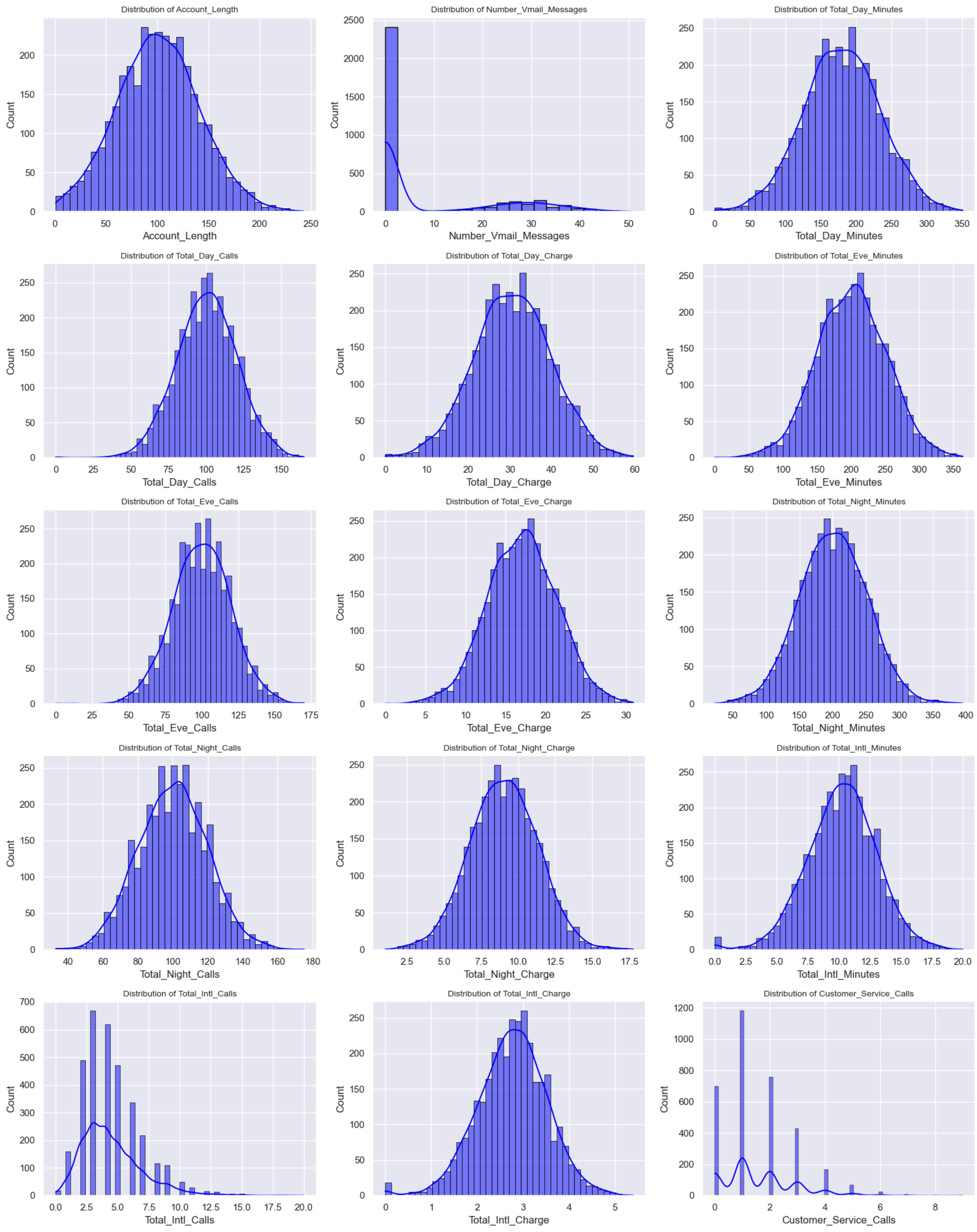
3.2.1.4 Numerical Features Distribution

This section examines the distribution of numerical features using Kernel Density Estimation (KDE) plots. The analysis focuses on:

- Identifying customer usage patterns across minutes, calls and charges.
- Detecting potential outliers that may indicate churn risk.

The numerical_distributions function from utility.py generates KDE plots for each numerical feature in the dataset.

In [17]:
```
# Defining the numerical features

numerical_features = [
    'Account_Length', 'Number_Vmail_Messages', 'Total_Day_Minutes', 'Total_
    'Total_Day_Charge', 'Total_Eve_Minutes', 'Total_Eve_Calls', 'Total_Eve_
    'Total_Night_Minutes', 'Total_Night_Calls', 'Total_Night_Charge', 'Tota
    'Total_Intl_Calls', 'Total_Intl_Charge', 'Customer_Service_Calls'
]

# Importing the function from 'utility.py'

from utility import numerical_distribution

# Call the numerical_distributions function
numerical_distribution(churn_df, numerical_features)
```



The distributions of numerical features are as follows:

- Account Length is approximately normally distributed showing balanced variation in account duration.
- Total Day, Evening and Night Minutes or Charges follow near normal distributions showing consistent usage patterns across customers.
- Total Calls; Day, Evening, Night are almost normal suggesting stable call behavior.
- Number of Vmail Messages is highly skewed with most customers not using voicemail.
- Total International Minutes or Charges are normally distributed while Total International Calls are right skewed showing a smaller group with high activity.
- Customer Service Calls is right skewed with most customers making few calls but a minority making many. This could mean dissatisfaction.

These patterns show overall stable service usage with a few skewed features that may influence churn behaviour.

3.2.2. Bivariate Analysis

Bivariate analysis in EDA examines the relationship between two variables to identify patterns and dependencies. In the project, the relationship between categorical features and the target variable Churn is analyzed.

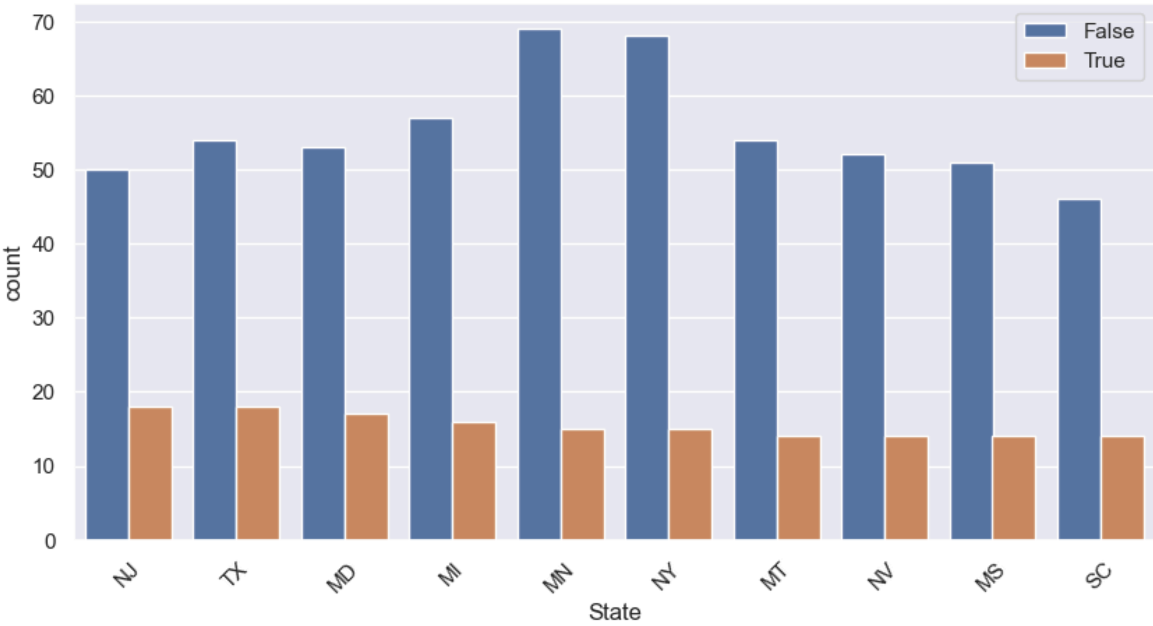
3.2.2.1 Categorical features vs Churn

This analysis directly supports Research Question 2 by showing how customer characteristics such as State, International Plan and Voice Mail Plan influence churn.

Bar plots are used to compare churn outcomes across these features helping to show which attributes have the strongest impact on customer retention.

State Feature

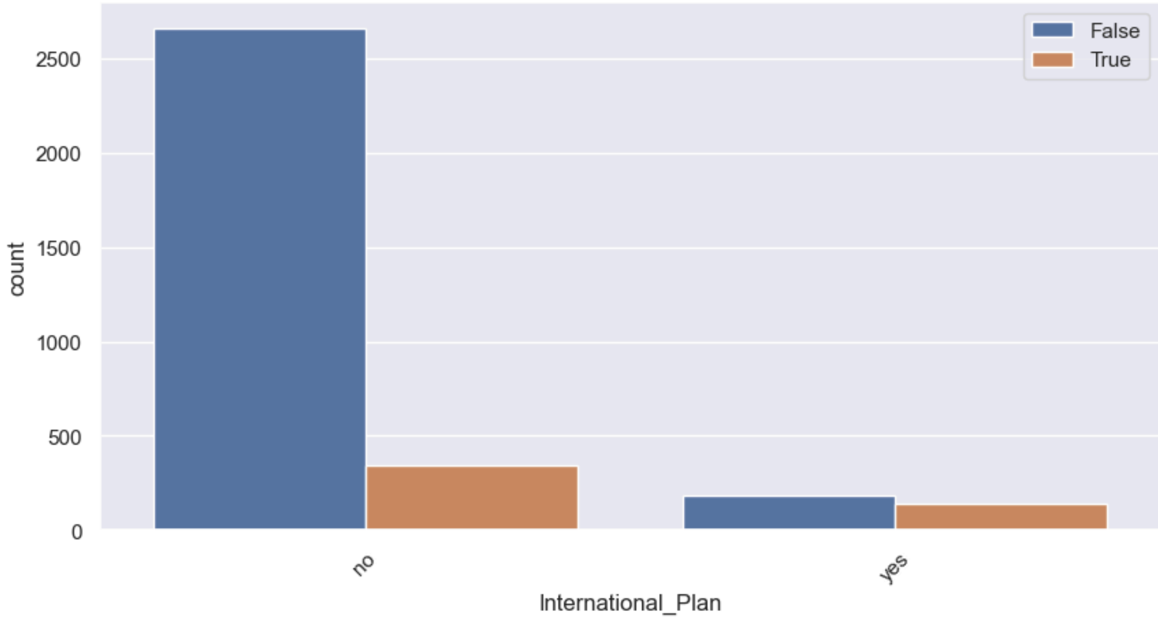
In [18]:
```
# Importing the function from utility.py

from utility import categorical_churn

# Pass the dataframe and feature

categorical_churn(churn_df, 'State')
```



- *The distribution shows churn occurs across all states with different frequencies.*
- *Certain states such as New York record slightly higher churn counts which could mean possible regional differences.*
- *While no extreme outliers are visible these findings confirm that geographic location is a relevant customer characteristic and may contribute to churn prediction. This finding supports Research Question 2.*

International Plan Feature

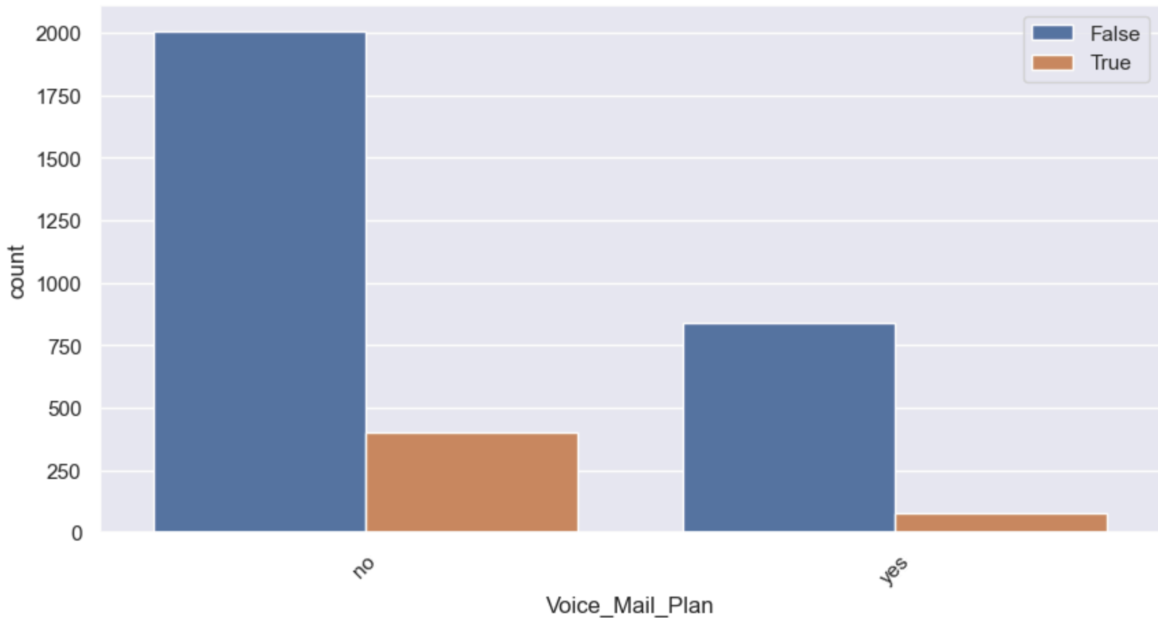
In [19]:
```
# Pass the dataframe and feature

categorical_churn(churn_df, 'International_Plan')
```



- *Customers with an international plan show a higher churn rate compared to those without.*
- *Despite being a small portion of the base this group is more likely to leave.*
- *International plan status is a strong predictor of churn supporting Research Question 2.*

Voice Mail Plan

In [20]:
```
# Pass the dataframe and feature

categorical_churn(churn_df, 'Voice_Mail_Plan')
```



The following were the key observations:

- Majority without a voice mail plan show both churn and non churn cases.

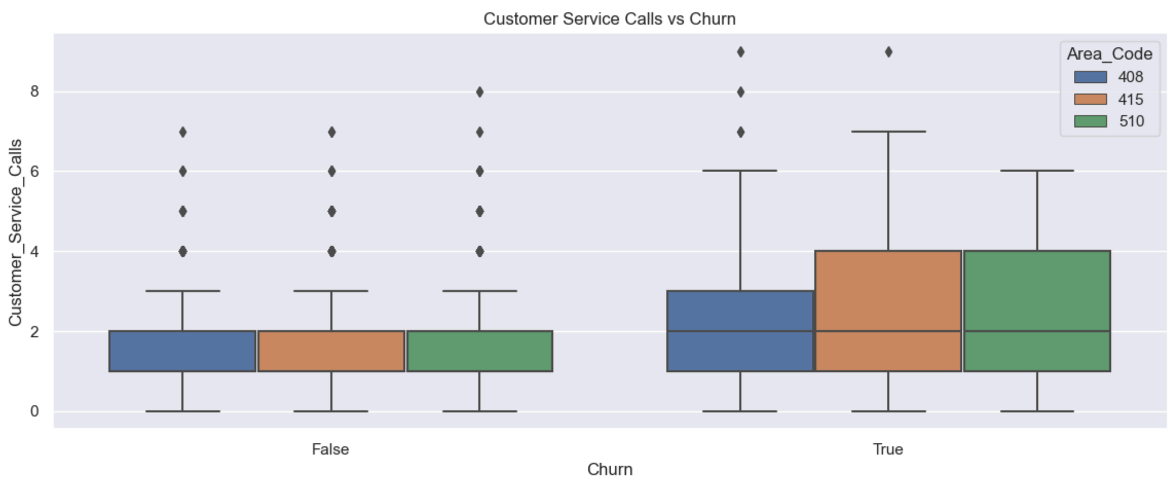
- A smaller group with a plan is also present but churn is noticeably lower compared to non churn.
- The difference could mean that the presence of a voice mail plan may not strongly prevent churn. However, it remains an important feature for analysis.

This insight contributes to Research Question 2 on identifying customer characteristics that influence churn.

3.2.2.2 Customer Service Calls vs Churn

In this section, we want to visualize the variation in churn with the number of customer service calls. This will help us in determining whether customer service calls are a major contributor towards customer churning.

In [21]:
```
# Boxplot to show area code with the highest churn

plt.figure(figsize=(14, 5))
sns.boxplot(data=churn_df, x='Churn', y='Customer_Service_Calls', hue='Area
plt.title('Customer Service Calls vs Churn')
plt.savefig('images_customer_service_churn.jpg', dpi=300)
plt.show()
```



- *Customers with higher numbers of customer service calls show a greater likelihood of churn.*
- *Churn rates increase sharply when service calls exceed three that could mean dissatisfaction or unresolved issues.*
- *This confirms that customer support interactions are a critical driver of churn aligning with Research Question 2 on identifying key customer characteristics.*

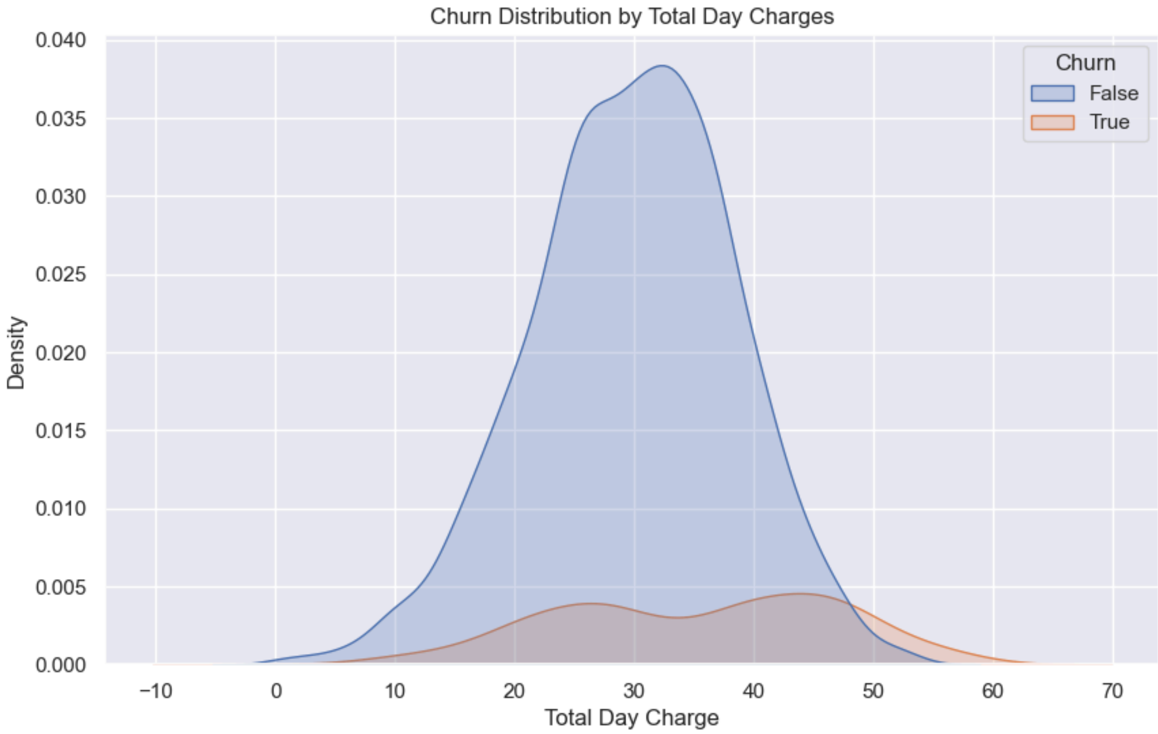
3.2.2.3 Numerical Features vs Churn

This section examines how numerical features such as Total Day Charge, Total Eve Charge, Total Night Charge and Total Intl Charge vary between churned and non churned customers. Kernel Density Estimation (KDE) plots are used to show differences in distribution helping to identify whether higher usage and charges are associated with churn.

The utility function kde_plots_with_churn generates each plot. It takes in the dataframe, feature and the charge type; day, evening, night or international.

Total Day Charge Feature

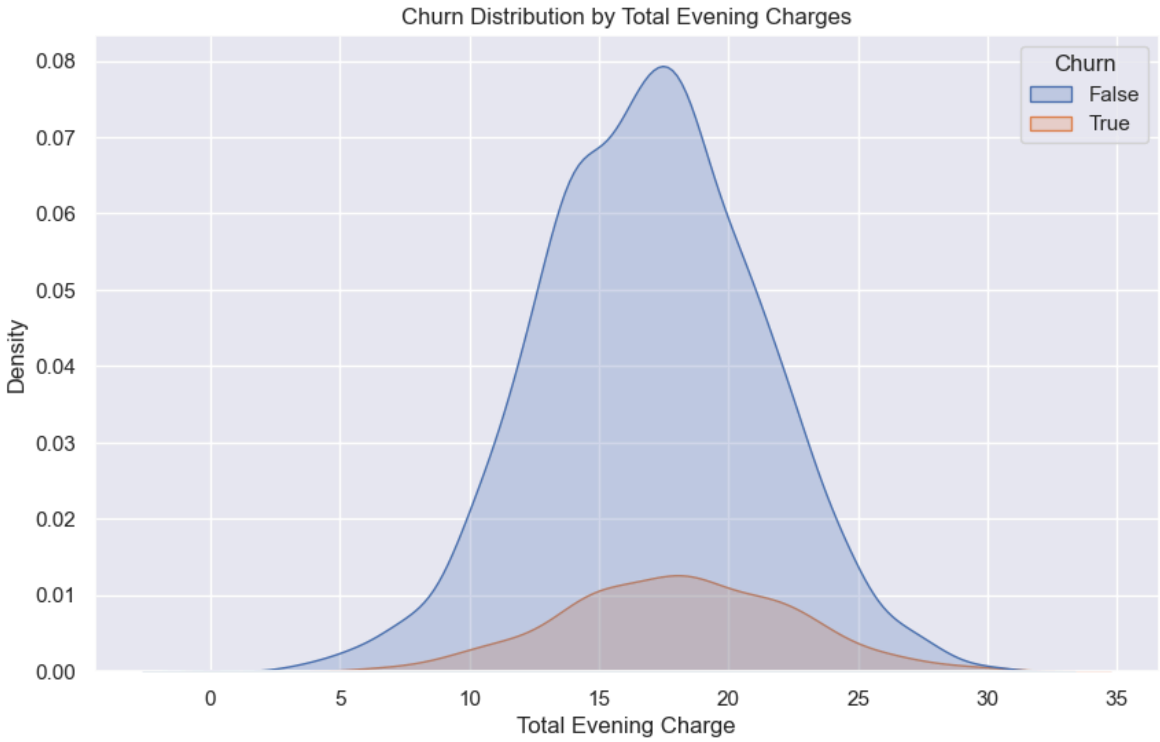
In [22]:
```
# Importing the function from utility.py

from utility import kde_plots_with_churn

# Pass the dataframe, feature and charge type

kde_plots_with_churn(churn_df, 'Total_Day_Charge', 'Day')
```



- *This KDE plot shows the distribution of Total Day Charges for customers who churned as Churn = True vs those who did not as Churn = False.*
- *From the plot, the orange; churned curve has a longer right tail and maintains density at higher values of day charges.*
- *This implies that customers who churn tend to have higher day charges than those who do not churn.*

Total Evening Charge Feature

In [23]:
```
# Pass the dataframe, feature and charge type

kde_plots_with_churn(churn_df, 'Total_Eve_Charge', 'Evening')
```

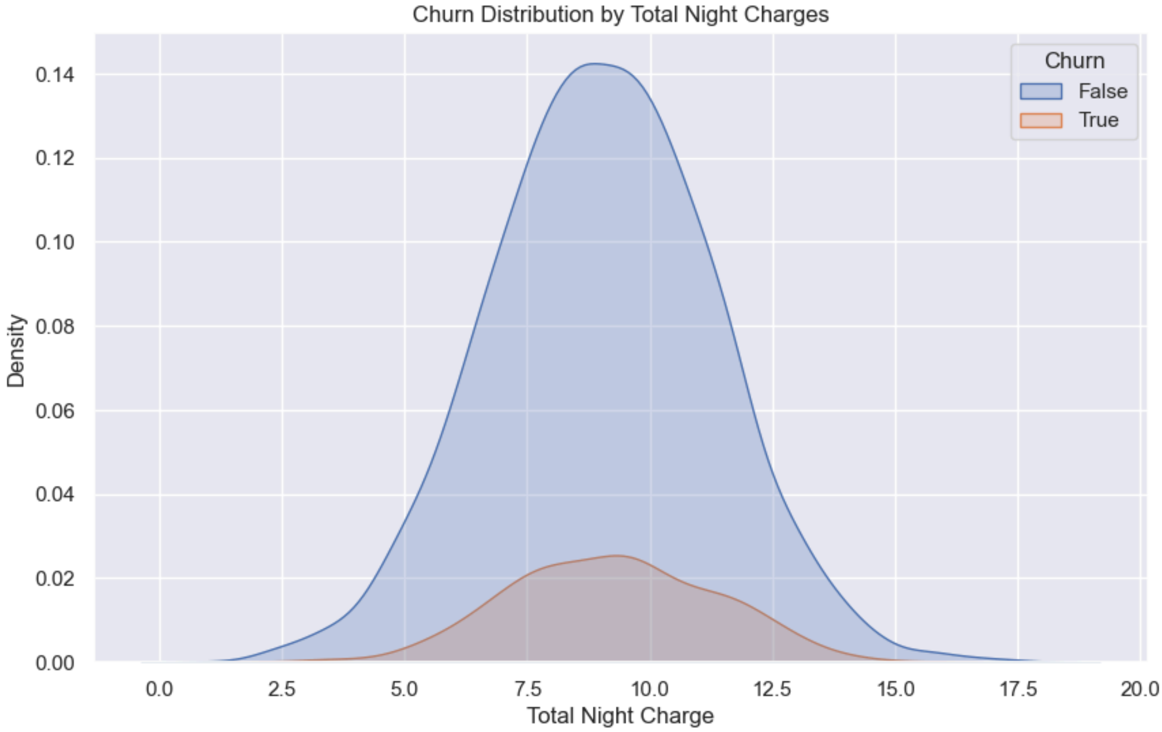


In this plot the key observations are as follows:

- The non churned group; blue has a tighter and higher peak between 15-20 while the churned group; orange is lower and flatter with a small shift towards higher evening charges.
- The churned group maintains more density beyond about 25 compared to the non churned group similar to the trend seen with the day charges.
- From this, customers who churned show a tendency to have higher evening charges but the separation between churned and non churned is less pronounced.

Total Night Charge Feature

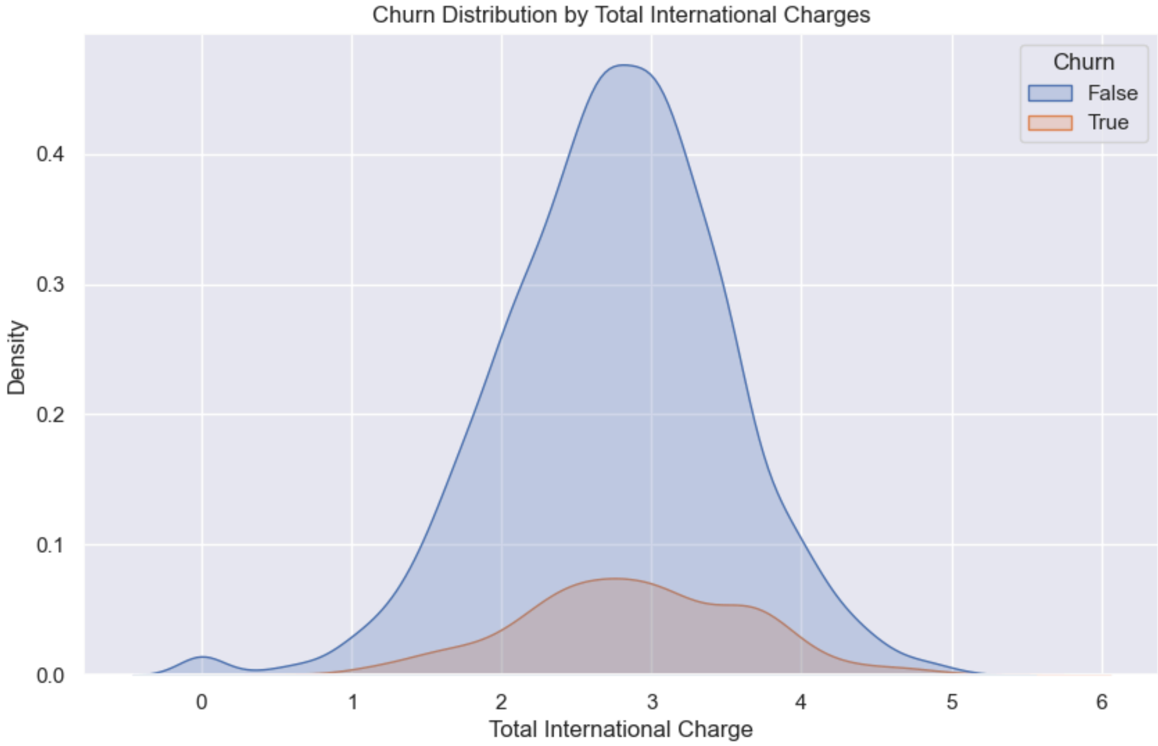
In [24]:
```
# Pass the dataframe, feature and charge type

kde_plots_with_churn(churn_df, 'Total_Night_Charge', 'Night')
```



- *Night charges are slightly higher among churned customers.*
- *Strong overlap between churned and non-churned groups.*
- *Night charges alone are not a strong churn indicator.*

Total International Charge Feature

In [25]:
```
# Pass the dataframe, feature and charge type

kde_plots_with_churn(churn_df, 'Total_Intl_Charge', 'International')
```



From the density plot the following are the key observations:

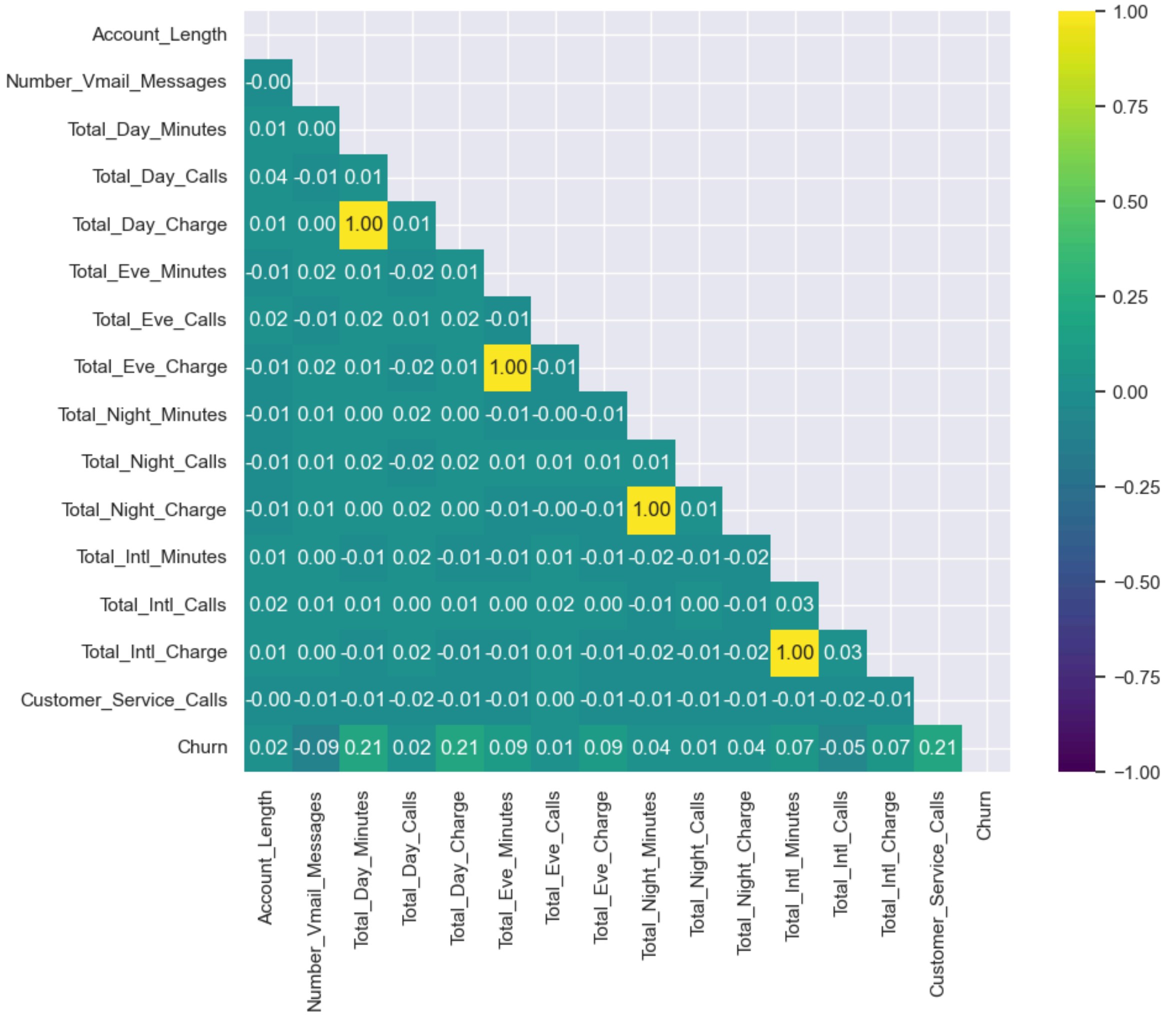
- Non churning customers cluster around moderate international charges of about 2.5 to 3.0.
- Churning customers show higher and more variable international charges of especially above 4.0.
- Very low charges of below 1.0 are mostly linked to customer retention.

3.2.3. Feature correlation

In this section, a correlation heatmap is used to evaluate the strength of relationships between numerical features and the target variable Churn. This step helps in identifying potential predictors of churn and detecting multicollinearity among features.

A custom function correlation_heatmap was created in utility.py to take a dataframe and return a correlation heatmap of numerical variables against the target.

In [26]:
```
# Importing the function from utility.py

from utility import correlation_heatmap

corr_matrix = churn_df.corr(numeric_only=True)

# Creating a mask for the upper triangle

mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(
    corr_matrix,
    mask=mask,
    cmap="viridis",
    annot=True,
    fmt=".2f",
    vmin=-1,
    vmax=1
)
plt.show()
```



The heatmap shows strong relationships among features:

- Perfect correlations (1.0) exist between Total_Day_Minutes and Total_Day_Charge, Total_Eve_Minutes and Total_Eve_Charge, Total_Night_Minutes and Total_Night_Charge, and Total_Intl_Minutes and Total_Intl_Charge. These show fixed per

minute rates so only one feature from each pair should be retained to prevent redundancy.
- Customer_Service_Calls shows the strongest positive correlation with churn making it an important predictor.
- International usage also shows a modest positive link with churn while most other features exhibit minimal correlation.

3.2.4. Multicollinearity check

Multicollinearity occurs when independent variables are highly correlated leading to redundancy and reduced model interpretability. A correlation threshold of 0.9 was applied and one feature from each highly correlated pair was removed.

- Dropped features: Total_Day_Charge, Total_Eve_Charge, Total_Night_Charge, Total_Intl_Charge.
- These features were perfectly correlated with their respective minute variables.
- Removing them improves model stability, minimizes overfitting risk and ensures each remaining feature provides distinct information.

The function drop_highly_correlated_features was created. It takes in a dataframe and returs a dataframe with dropped features.

In [27]:
```
# Importing the function from utility.py

from utility import drop_highly_correlated_features

# Pass in the dataframe

cleaned_churn_df, dropped_features = drop_highly_correlated_features(churn_

# Displaying the dropped features

print("Dropped Features:", dropped_features)

# Displaying the correlation matrix

correlation_matrix = cleaned_churn_df.corr(numeric_only=True)

print("\nCorrelation Matrix:")

# Rounded for readabily
print(correlation_matrix.round(2))
```

```
Dropping 4 highly correlated features (r > 0.9): ['Total_Day_Minutes', 'T
otal_Eve_Minutes', 'Total_Night_Minutes', 'Total_Intl_Minutes']
Dropped Features: ['Total_Day_Minutes', 'Total_Eve_Minutes', 'Total_Night
_Minutes', 'Total_Intl_Minutes']

Correlation Matrix:
                        Account_Length  Number_Vmail_Messages  \
Account_Length                    1.00                  -0.00
Number_Vmail_Messages            -0.00                   1.00
Total_Day_Calls                   0.04                  -0.01
Total_Day_Charge                  0.01                   0.00
Total_Eve_Calls                   0.02                  -0.01
Total_Eve_Charge                 -0.01                   0.02
Total_Night_Calls                -0.01                   0.01
Total_Night_Charge               -0.01                   0.01
Total_Intl_Calls                  0.02                   0.01
Total_Intl_Charge                 0.01                   0.00
Customer_Service_Calls           -0.00                  -0.01
Churn                             0.02                  -0.09

                        Total_Day_Calls  Total_Day_Charge  Total_Eve_Call
s  \
Account_Length                     0.04              0.01             0.0
2
Number_Vmail_Messages             -0.01              0.00            -0.0
1
Total_Day_Calls                    1.00              0.01             0.0
1
Total_Day_Charge                   0.01              1.00             0.0
2
Total_Eve_Calls                    0.01              0.02             1.0
0
Total_Eve_Charge                  -0.02              0.01            -0.0
1
Total_Night_Calls                 -0.02              0.02             0.0
1
Total_Night_Charge                 0.02              0.00            -0.0
0
Total_Intl_Calls                   0.00              0.01             0.0
2
Total_Intl_Charge                  0.02             -0.01             0.0
1
Customer_Service_Calls            -0.02             -0.01             0.0
0
Churn                              0.02              0.21             0.0
1

                        Total_Eve_Charge  Total_Night_Calls  \
Account_Length                     -0.01              -0.01
Number_Vmail_Messages               0.02               0.01
Total_Day_Calls                    -0.02              -0.02
Total_Day_Charge                    0.01               0.02
Total_Eve_Calls                    -0.01               0.01
Total_Eve_Charge                    1.00               0.01
Total_Night_Calls                   0.01               1.00
Total_Night_Charge                 -0.01               0.01
Total_Intl_Calls                    0.00               0.00
Total_Intl_Charge                  -0.01              -0.01
Customer_Service_Calls             -0.01              -0.01
Churn                               0.09               0.01
```

```
                        Total_Night_Charge  Total_Intl_Calls  \
Account_Length                       -0.01              0.02   
Number_Vmail_Messages                 0.01              0.01   
Total_Day_Calls                       0.02              0.00   
Total_Day_Charge                      0.00              0.01   
Total_Eve_Calls                      -0.00              0.02   
Total_Eve_Charge                     -0.01              0.00   
Total_Night_Calls                     0.01              0.00   
Total_Night_Charge                    1.00             -0.01   
Total_Intl_Calls                     -0.01              1.00   
Total_Intl_Charge                    -0.02              0.03   
Customer_Service_Calls               -0.01             -0.02   
Churn                                 0.04             -0.05   

                        Total_Intl_Charge  Customer_Service_Calls  Churn  
Account_Length                       0.01                   -0.00   0.02  
Number_Vmail_Messages                0.00                   -0.01  -0.09  
Total_Day_Calls                      0.02                   -0.02   0.02  
Total_Day_Charge                    -0.01                   -0.01   0.21  
Total_Eve_Calls                      0.01                    0.00   0.01  
Total_Eve_Charge                    -0.01                   -0.01   0.09  
Total_Night_Calls                   -0.01                   -0.01   0.01  
Total_Night_Charge                  -0.02                   -0.01   0.04  
Total_Intl_Calls                     0.03                   -0.02  -0.05  
Total_Intl_Charge                    1.00                   -0.01   0.07  
Customer_Service_Calls              -0.01                    1.00   0.21  
Churn                                0.07                    0.21   1.00  
```

- *The features Total_Day_Minutes, Total_Eve_Minutes, Total_Night_Minutes and Total_Intl_Minutes were dropped due to perfect correlation with their respective charge variables.*
- *This step removes redundancy since minutes and charges convey the same information at fixed rates*
- *By keeping only one representative from each pair, the dataset is simplified while ensuring that all retained features contribute unique predictive value.*

3.2.5. Handling outliers

Outliers are extreme data points that can distort model performance and reduce reliability.

The function remove_outliers_zscore was created. It takes in a dataframe and returns a new dataframe with rows removed where any numerical column has a Z-score exceeding a threshold of 3.

This process ensures a cleaner dataset, minimizes noise and improves model accuracy.

In [28]:
```
# Imporing the function from utility.py

from utility import remove_outliers_zscore

# Pass in the dataframe
cleaned_churn_df = remove_outliers_zscore(cleaned_churn_df, z_threshold=3.(

# Print the shape of the new dataframe

print("Shape of dataset after removing outliers:", cleaned_churn_df.shape)
```

```
Shape of dataset after removing outliers: (3169, 16)
```

The function remove_outliers_zscore works as follows:

- Identify numeric columns: Selects only numerical features; integers and floats.
- Compute Z-scores: Measures how far each value is from the mean in standard deviations.
- Apply threshold: Rows with any feature's Z-score greater than 3 are flagged as outliers.
- Filter data: Keeps only rows within the threshold returning a cleaned dataset.

This ensures that extreme values are removed, reducing noise and improving the reliability of the dataset for modeling.

3.3. Data Preprocessing

Data preprocessing is the process of preparing raw data into a usable format for analysis and modeling. It includes steps such as encoding categorical variables and scaling numerical features to ensure accuracy, consistency and compatibility with machine learning algorithms.

In this project, the preprocessing steps include:

- One-Hot Encoding
- Label Encoding
- Feature Scaling

3.3.1. One-Hot Encoding

One-Hot Encoding is used to convert categorical variables into a numerical format by creating new binary (0/1) columns for each category. This allows machine learning models to process categorical data without assuming any ordinal relationship.

In this project, One-Hot Encoding is applied to the following categorical features. State, Area_Code, International_Plan and Voice_Mail_Plan.

This transformation ensures that categorical information is properly represented for modeling, improving consistency and compatibility with algorithms.

In [29]:
```
# One-hot encode the categorical columns

from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline

# List of categorical columns to encode

categorical_cols = ['State', 'Area_Code', 'International_Plan', 'Voice_Mai

# Create the ColumnTransformer with OneHotEncoder
# Steps to different columns in a dataframe

one_hot_encoder = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(drop='first', sparse_output=False), categ
    ],
    remainder='passthrough'  # Keep all other columns as they are
)

# Fit and transform the data

encoded_array = one_hot_encoder.fit_transform(cleaned_churn_df)

# Get the new column names from the encoder

encoded_columns = one_hot_encoder.named_transformers_['onehot'].get_feature

# Create a new DataFrame with the encoded data

preprocessed_churn_df = pd.DataFrame(encoded_array, columns=list(encoded_co

# View the shape of the encoded DataFrame
preprocessed_churn_df.shape
```

Out[29]: (3169, 66)

After encoding, the dataset expanded from 16 columns to 66 columns, ensuring all categorical information was captured in numerical format for machine learning models.

3.3.2. Label Encoding

Label Encoding converts categorical text values into numeric form by assigning integer labels. This step is applied to the Churn target variable, encoding it into binary values 0; No Churn and 1; Churn.

This transformation ensures the target variable is machine readable for modeling.

In [30]:
```
# Using Label encoding on Churn column

label_encoder = LabelEncoder()
preprocessed_churn_df['Churn'] = label_encoder.fit_transform(preprocessed_c

# Display first 5 rows
preprocessed_churn_df['Churn'].head()
```

Out[30]:
```
0    0
1    0
2    0
3    0
4    0
Name: Churn, dtype: int32
```

- *The Churn column has been successfully encoded into binary values.*
- *This step ensures the target variable is in a numerical format allowing models to interpret churn outcomes as 0 being No Churn and 1 being Churn.*

3.3.3. Data Scaling

Data Scaling is the process of transforming features so they fall within the same range. It ensures that no feature dominates due to its magnitude.

In this project, Min-Max Scaling is used to normalize features to a fixed range of -1 to 1. This allows all variables to contribute equally during model training and improving model performance.

This scaling step will be applied in the modeling phase when defining the feature matrix X and target variable y.

4. Modelling

This section focuses on building predictive models to classify customer churn using the features in the dataset. The objective is to identify customers at risk of churn and generate insights that support effective retention strategies.

Five models are trained and evaluated:

- Logistic Regression. This is the baseline model for comparison
- Decision Tree
- Random Forest
- K-Nearest Neighbors (KNN)
- Gradient Boosting Classifier

Model performance is assessed using Recall which emphasizes the correct identification of churners. In addition, ROC-AUC which measures overall classification ability.

The next step defines X which are the features and y which are the target variables with Churn serving as the target column.

In [31]:
```
# Defining X and y variables

X = preprocessed_churn_df.drop(columns='Churn', axis=1)
y = preprocessed_churn_df['Churn']

# Display the first 5 rows of X
X.head()
```

Out[31]:

	State_AL	State_AR	State_AZ	State_CA	State_CO	State_CT	State_DC	State_DE	State_FL
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 65 columns

- The X and y variables are defined and it is possible to split the data into train and test sets.
- The dataset has been split into training at 80% and testing at 20% subsets to enable model training and unbiased evaluation.
- In addition, the features have been scaled using Min-Max normalization which transforms values into a fixed range between 0 and 1.

This scaling step ensures that all variables contribute to model performance. This prevents features with larger magnitudes from dominating the learning process.

In [32]:
```
# Splitting into training and testing sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, st

# Scaling the train and test features

scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Display the shape of the train and test sets
print("Shape of scaled X_train:", X_train_scaled.shape)
print("Shape of scaled X_test:", X_test_scaled.shape)
print("Shape of y_train:", y_train.shape)
print("Shape of y_test:", y_test.shape)
```

```
Shape of scaled X_train: (2535, 65)
Shape of scaled X_test: (634, 65)
Shape of y_train: (2535,)
Shape of y_test: (634,)
```

4.1. Class Imbalance

From the data analysis, it was observed that the target variable has a high class imbalance. This can be showed using the y_train variable:

In [33]:
```
# Checking the distribution of the unique values

print("Churn counts:")
print(y_train.value_counts())

# Checking for normalized counts as a percentage

print("\nChurn distribution (%):")
print(y_train.value_counts(normalize=True) * 100)
```

```
Churn counts:
0    2181
1     354
Name: Churn, dtype: int64

Churn distribution (%):
0    86.035503
1    13.964497
Name: Churn, dtype: float64
```

- *The target variable showed a severe imbalance with about 86% non-churn and 14% churn. To correct this, the SMOTE (Synthetic Minority Over-Sampling Technique) method was applied.*
- *SMOTE generates synthetic samples of the minority class by interpolating between existing data points and their nearest neighbors. This balances the dataset, allowing models to learn from both classes equally and reducing bias toward the majority class.*
- *After applying SMOTE, the training dataset was resampled to achieve a balanced distribution between churn and non churn classes.*

In [34]:
```
# Importing the imblearn library

from imblearn.over_sampling import SMOTE, SMOTENC

# Instantiate the SMOTE function

smote = SMOTENC(categorical_features=[1, 2], random_state=42)

# Fitting and resample the training data

X_train_smote, y_train_smote = smote.fit_resample(X_train_scaled, y_train)

# Checking the unique values in y_train

y_train_smote.value_counts()
```

Out[34]:
```
0    2181
1    2181
Name: Churn, dtype: int64
```

- *After applying SMOTE, the training dataset was successfully resampled to achieve an equal distribution between churn and non churn classes with 2,181 rows each.*
- *This balanced dataset reduces bias toward the majority class and provides a fair foundation for training predictive models. The modeling process will begin with Logistic Regression as the baseline model, followed by other classifiers for comparison.*

4.2. Logistic Regression

This is the baseline model.

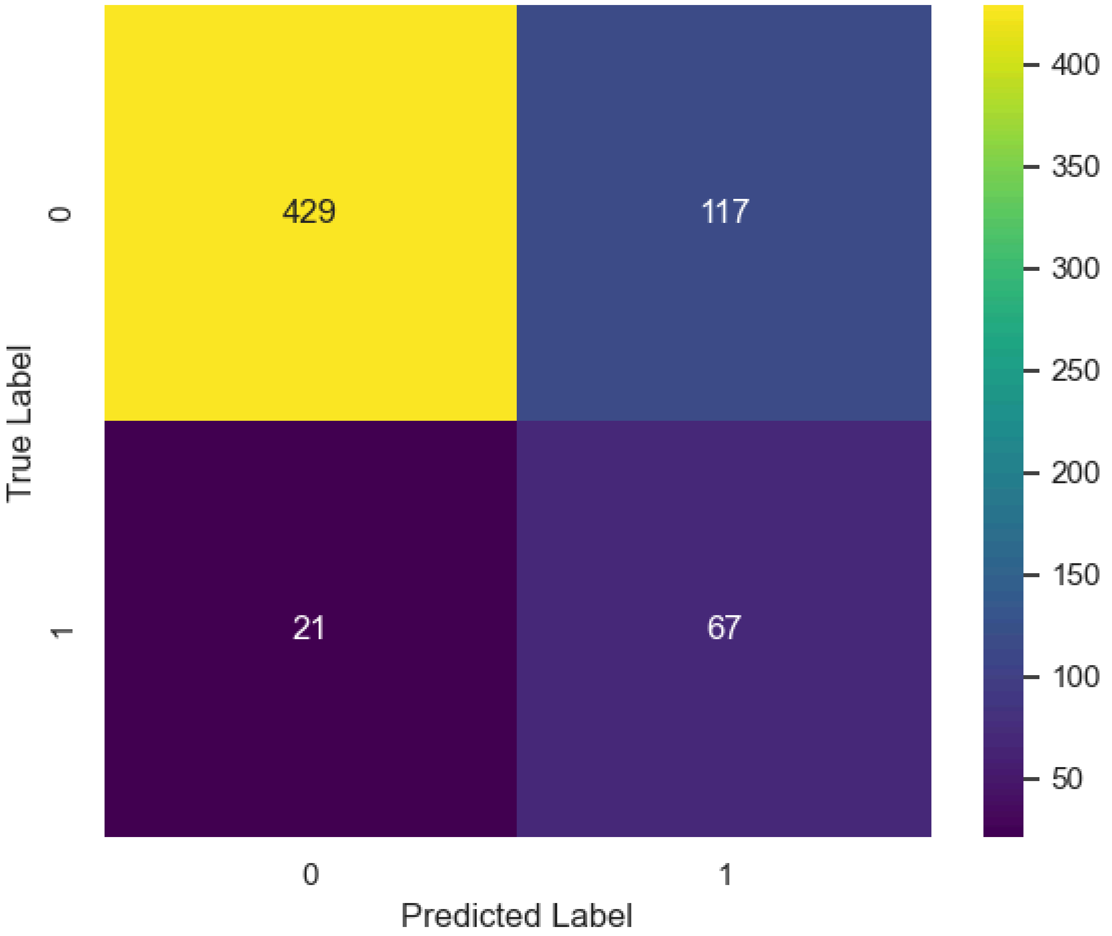
In [35]:
```
# Instantiate a Logistic regression model

log_model = LogisticRegression(random_state=42)

# Fit the training data

log_model.fit(X_train_smote, y_train_smote)

# Make predictions on test set

y_pred_log = log_model.predict(X_test_scaled)

# Compute the accuracy of the model

log_acc = accuracy_score(y_test, y_pred_log)
print("Logistic Regression model accuracy:", log_acc)
```

```
Logistic Regression model accuracy: 0.7823343848580442
```

In [36]:
```
# Display the classification report

log_report = classification_report(y_true=y_test, y_pred=y_pred_log)
print("Logistic Regression classification report\n")
print(log_report)
```

```
Logistic Regression classification report

              precision    recall  f1-score   support

           0       0.95      0.79      0.86       546
           1       0.36      0.76      0.49        88

    accuracy                           0.78       634
   macro avg       0.66      0.77      0.68       634
weighted avg       0.87      0.78      0.81       634
```

In [37]:
```
# Import the confusion_matrix function

from utility import plot_confusion_matrix

# Plot the confusion matrix
plot_confusion_matrix(y_test, y_pred_log, class_labels=[0, 1])
```



- *The Logistic Regression model achieved an accuracy of about 78.2%.*
- *The classification report shows stronger performance in predicting the majority class; No Churn compared to the minority class; Churn which is expected given the dataset imbalance.*
- *The confusion matrix further shows correct identification of most non churn cases. However, churn predictions remain more challenging.*
- *These results confirm Logistic Regression as a reasonable baseline model, providing a benchmark for evaluating more complex classifiers in the following sections.*

4.3. Decision Tree Classifier

The Decision Tree classifier was applied to the churn dataset as one of the predictive models. Decision Trees split the dataset into smaller subsets by learning simple decision rules based on feature values. This makes them easy to interpret and useful for classification tasks.

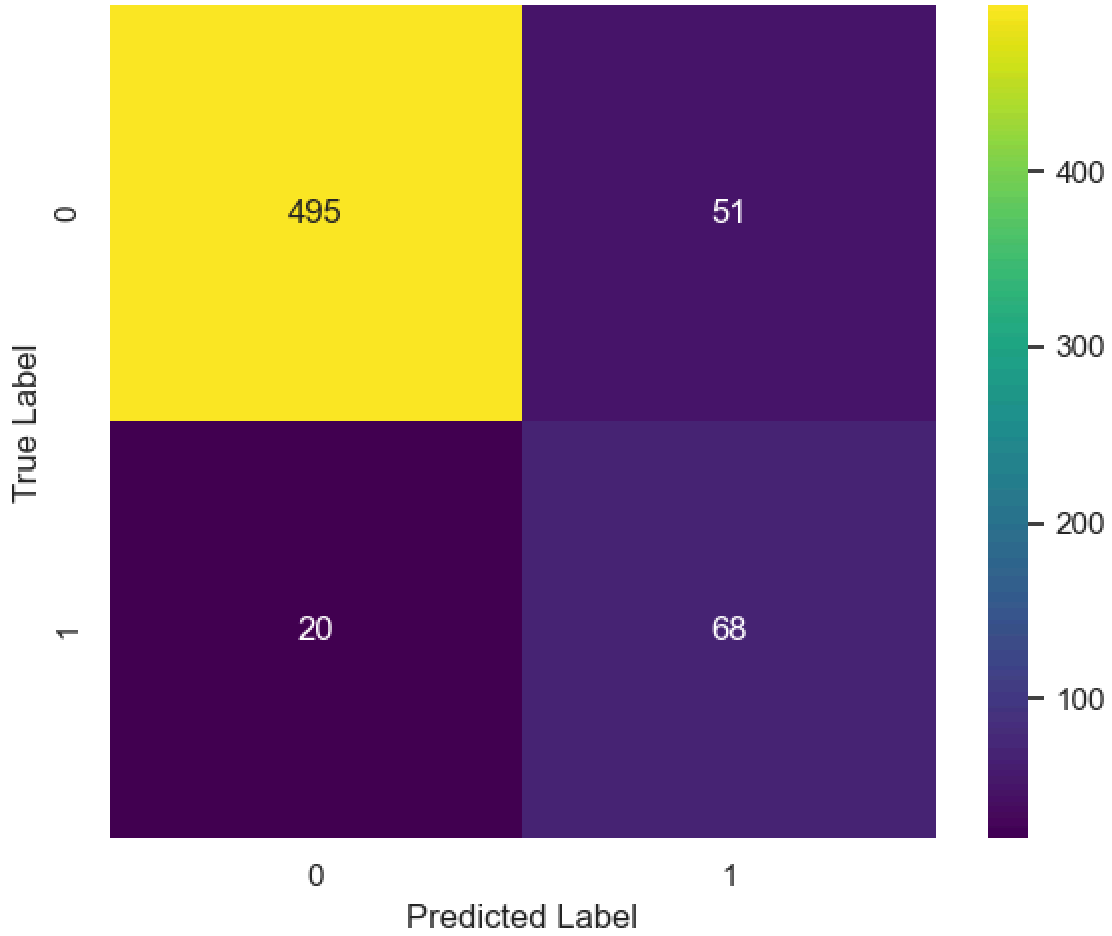
In [38]:
```
# Instantiate a decision tree model

dec_model = DecisionTreeClassifier(random_state=42)

# Fit the training data

dec_model.fit(X_train_smote, y_train_smote)

# Make predictions on test set

y_pred_dec = dec_model.predict(X_test_scaled)

# Compute the accuracy of the model

dec_acc = accuracy_score(y_test, y_pred_dec)
print("Decision Tree model accuracy:", dec_acc)
```

```
Decision Tree model accuracy: 0.88801261829653
```

In [39]:
```
# Display the classification report

dec_report = classification_report(y_true=y_test, y_pred=y_pred_dec)
print("Decision Tree classification report\n")
print(dec_report)
```

```
Decision Tree classification report

              precision    recall  f1-score   support

           0       0.96      0.91      0.93       546
           1       0.57      0.77      0.66        88

    accuracy                           0.89       634
   macro avg       0.77      0.84      0.80       634
weighted avg       0.91      0.89      0.89       634
```

In [40]:
```
# Plot the confusion matrix

plot_confusion_matrix(y_test, y_pred_dec, class_labels=[0, 1])
```



After training and testing the model, the following are the evaluation metrics:

- Accuracy: At about 88.8% showing strong overall predictive performance.
- Classification Report: Precision, recall and F1-scores showed balanced performance. There is improvement in recall for the minority; churn class compared to Logistic Regression.
- Confusion Matrix: Most non churn and churn customers were classified correctly with fewer misclassifications.

This evaluation demonstrates that the Decision Tree classifier is more effective than Logistic Regression in capturing churn cases. However, it may still be prone to overfitting without more tuning.

4.4. Random Forest Classifier

Random Forest is a method that combines multiple decision trees to improve accuracy and reduce overfitting. By averaging predictions from many trees, the model delivers more reliable results and stronger generalization for churn prediction.

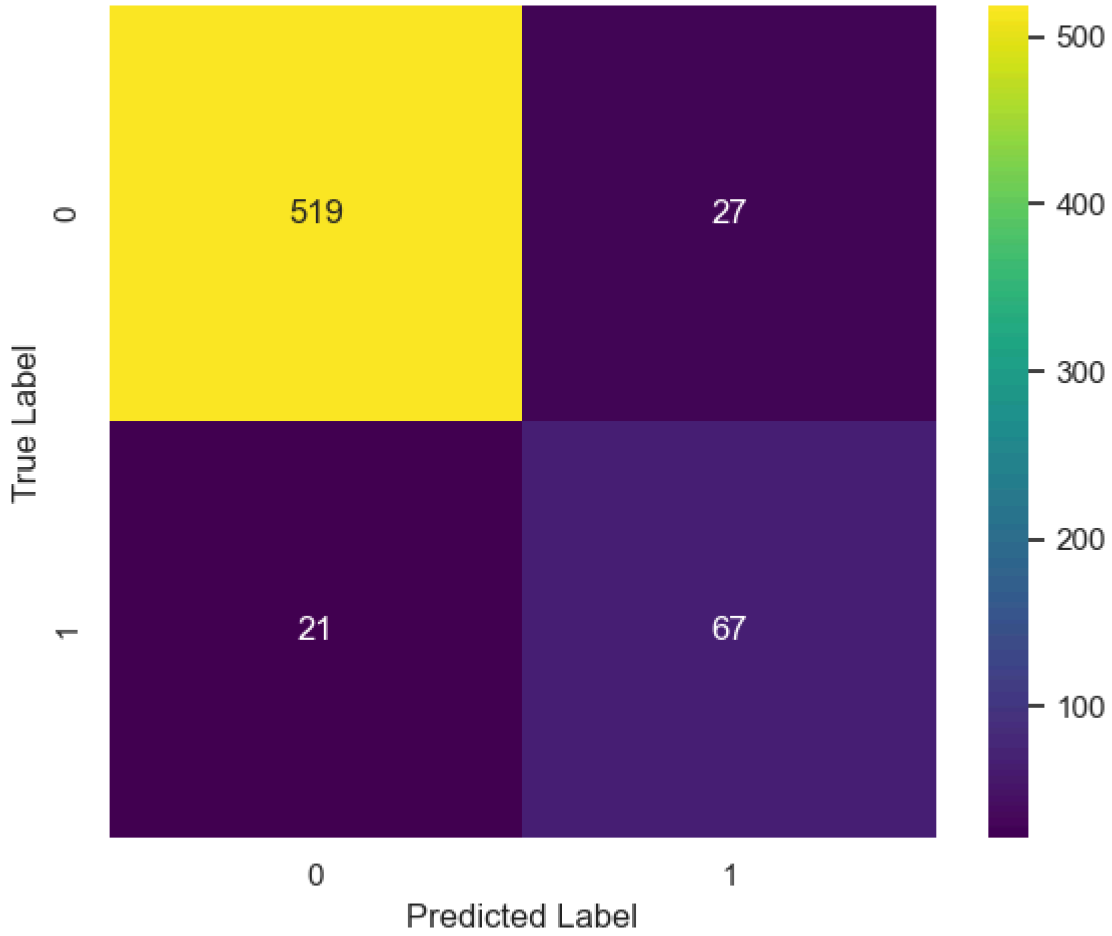
In [41]:
```
# Instantiate a random forest model

ran_model = RandomForestClassifier(random_state=42)

# Fit the training data

ran_model.fit(X_train_smote, y_train_smote)

# Make predictions on test set

y_pred_ran = ran_model.predict(X_test_scaled)

# Compute the accuracy of the model

ran_acc = accuracy_score(y_test, y_pred_ran)
print("Random Forest model accuracy:", ran_acc)
```

```
Random Forest model accuracy: 0.9242902208201893
```

In [42]:
```
# Display the classification report

ran_report = classification_report(y_true=y_test, y_pred=y_pred_ran)
print("Random Forest classification report\n")
print(ran_report)
```

```
Random Forest classification report

              precision    recall  f1-score   support

           0       0.96      0.95      0.96       546
           1       0.71      0.76      0.74        88

    accuracy                           0.92       634
   macro avg       0.84      0.86      0.85       634
weighted avg       0.93      0.92      0.93       634
```

In [43]:
```
# Plot the confusion matrix

plot_confusion_matrix(y_test, y_pred_ran, class_labels=[0, 1])
```



The key findings are as follows:

- The Random Forest Classifier achieved an accuracy of about 92.4% with strong performance across both classes.
- Precision and recall for the minority class; churn = 1 are higher compared to Logistic Regression and Decision Tree. This shows better balance in detecting churn cases.
- The confusion matrix confirms that the model correctly identified most churn and non churn cases with few misclassifications.

Random Forest shows robust predictive power. This reduces bias toward the majority class while maintaining strong generalization.

4.5. K-Nearest Neighbor Classifier

The K-Nearest Neighbor (KNN) algorithm classifies samples based on similarity to their nearest neighbors in the feature space. It is a simple yet powerful method. However, it can be sensitive to imbalanced data and scaling.

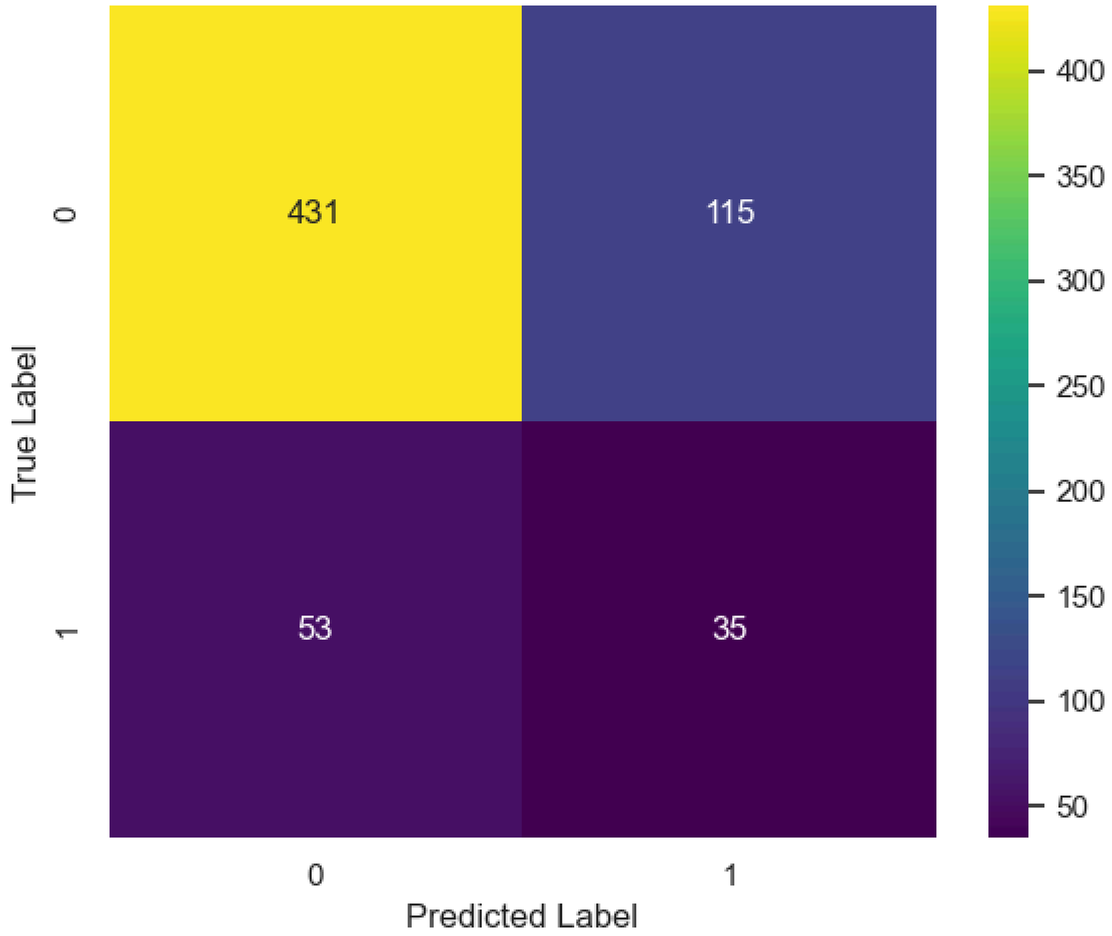
In [44]:
```
# Instantiate a K-nearest neighbor model

knn_model = KNeighborsClassifier(n_neighbors=5) # start with 5 neighbors

# Fit the training data
knn_model.fit(X_train_smote, y_train_smote)

# Make predictions on test set
y_pred_knn = knn_model.predict(X_test_scaled)

# Compute the accuracy of the model

knn_acc = accuracy_score(y_test, y_pred_knn)
print("K-Nearest Neighbor model accuracy:", knn_acc)
```

```
K-Nearest Neighbor model accuracy: 0.7350157728706624
```

In [45]:
```
# Display the classification report

knn_report = classification_report(y_true=y_test, y_pred=y_pred_knn)
print("K-Nearest Neighbor classification report\n")
print(knn_report)
```

```
K-Nearest Neighbor classification report

              precision    recall  f1-score   support

           0       0.89      0.79      0.84       546
           1       0.23      0.40      0.29        88

    accuracy                           0.74       634
   macro avg       0.56      0.59      0.57       634
weighted avg       0.80      0.74      0.76       634
```

In [46]:
```
# Plot the confusion

plot_confusion_matrix(y_test, y_pred_knn, class_labels=[0, 1])
```



The key findings are as follows:

- The model achieved an accuracy of about 73.5% which is lower than the previously tested models.
- Precision, recall and F1-score for the churn class; 1 are considerably low. This shows difficulty in detecting churned customers.
- The confusion matrix confirms that while the model correctly classified many non-churn cases, it struggled with churn predictions. This led to a lower recall score of 40%.

4.6. Gradient Boosting Classifier

Gradient Boosting is a method that combines many small decision trees to improve accuracy and reduce errors. Each tree learns from the mistakes of the previous one. This makes the model stronger.

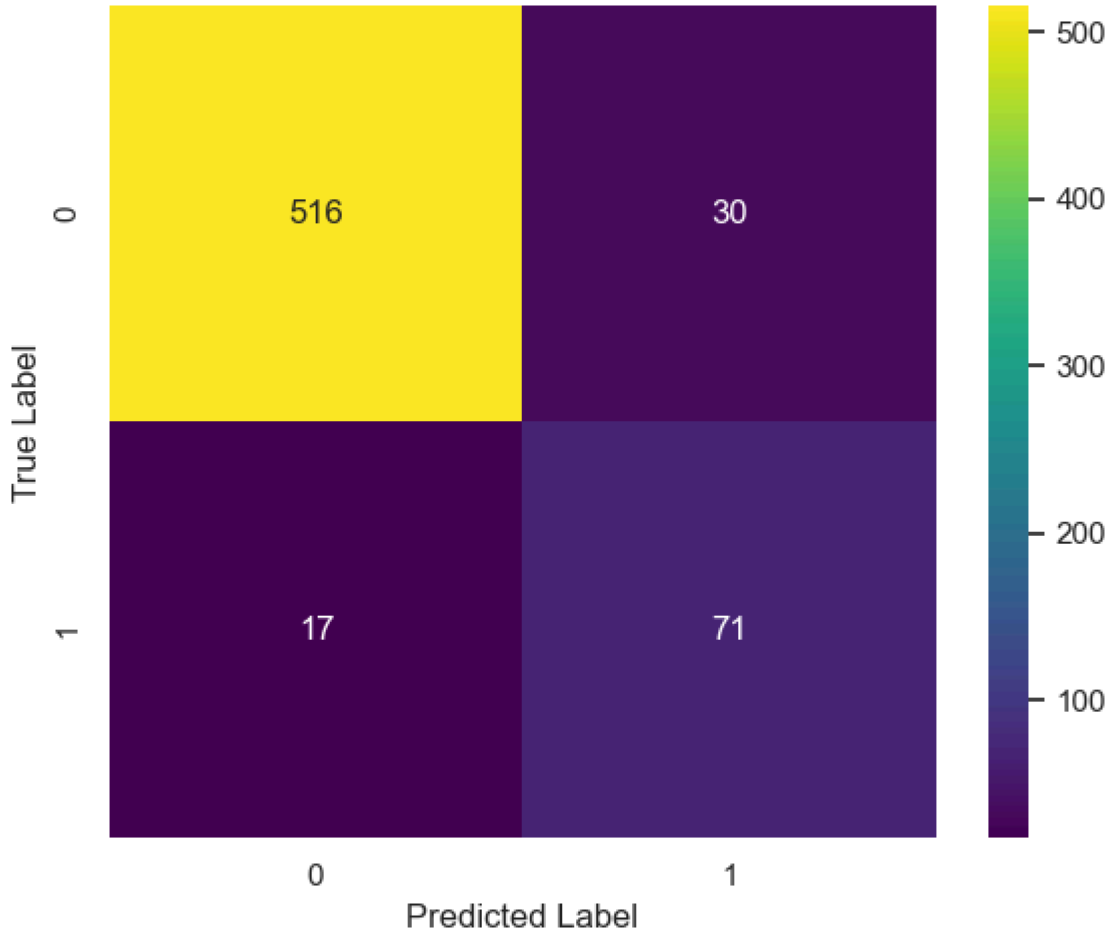
In [47]:
```
# Instantiate a Gradient Boosting model

gb_model = GradientBoostingClassifier(random_state=42)

# Fit the training data
gb_model.fit(X_train_smote, y_train_smote)

# Make predictions on test set
y_pred_gb = gb_model.predict(X_test_scaled)

# Compute the accuracy of the model

gb_acc = accuracy_score(y_test, y_pred_gb)
print("Gradient Boosting model accuracy:", gb_acc)
```

```
Gradient Boosting model accuracy: 0.9258675078864353
```

In [48]:
```
# Display the classification report

gb_report = classification_report(y_true=y_test, y_pred=y_pred_gb)
print("Gradient Boosting classification report\n")
print(gb_report)
```

```
Gradient Boosting classification report

              precision    recall  f1-score   support

           0       0.97      0.95      0.96       546
           1       0.70      0.81      0.75        88

    accuracy                           0.93       634
   macro avg       0.84      0.88      0.85       634
weighted avg       0.93      0.93      0.93       634
```

In [49]:
```
# Plot the confusion matrix

plot_confusion_matrix(y_test, y_pred_gb, class_labels=[0, 1])
```



The key findings are as follows:

- The Gradient Boosting Classifier reached 92.5% accuracy. This is the best model compared to the rest.
- Class 1; churn predictions showed higher recall and precision than previous models.
- The confusion matrix confirms most churn and non churn cases were correctly identified.

Gradient Boosting shows strong generalization ability. The model reduces bias while maintaining reliable predictive power.

5. Model Evaluation

This section evaluates the performance of all trained models to determine which are most effective for predicting churn.

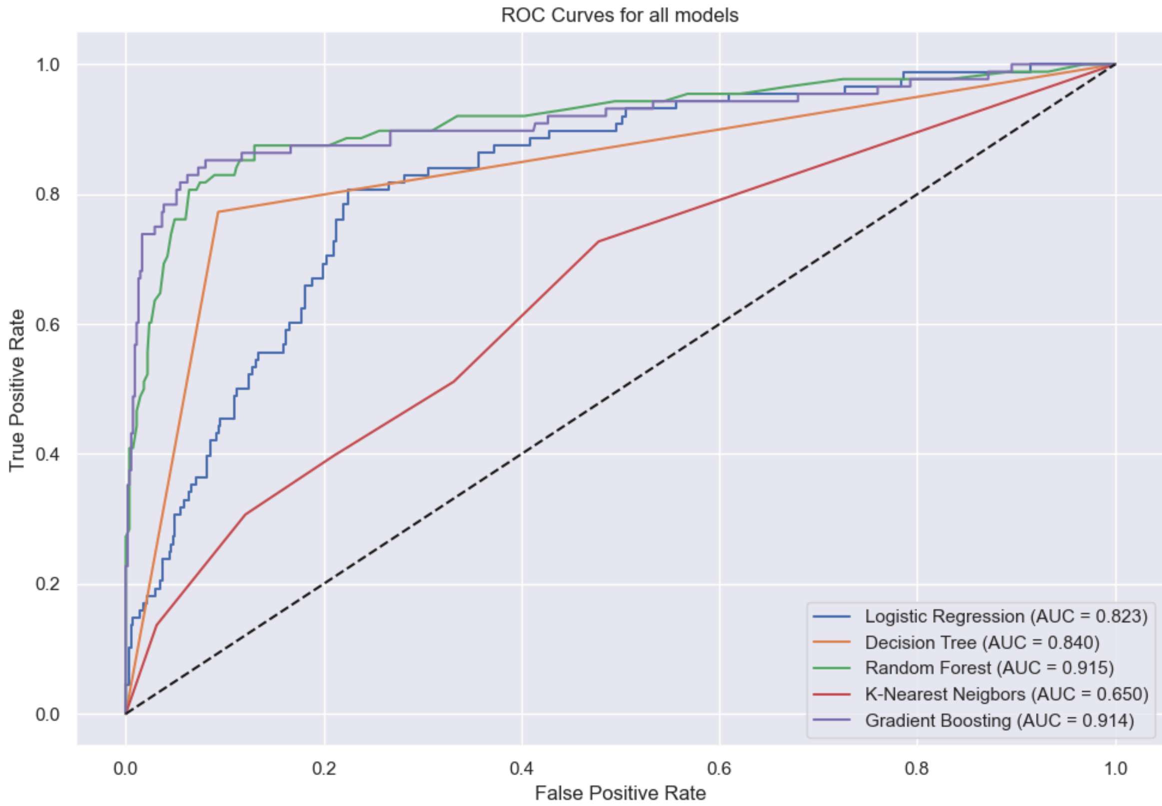
Key metrics such as recall and ROC-AUC are emphasized. This is because they provide the needed insight for imbalanced data. The best two models will then be selected for hyperparameter tuning.

5.1. ROC Curve

The ROC (Receiver Operating Characteristic) curve evaluates how well models distinguish between churn and non churn cases. It plots True Positive Rate which is Recall against False Positive Rate across thresholds. The AUC (Area Under the Curve) provides a single performance score. A higher AUC shows stronger model performance.

In this section, ROC curves are plotted for all classifiers to compare their predictive ability.

In [50]:
```
# Get predicted probabilities for models that support predict_proba

model_dict = {
    'Logistic Regression': log_model,
    'Decision Tree': dec_model,
    'Random Forest': ran_model,
    'K-Nearest Neigbors': knn_model,
    'Gradient Boosting': gb_model
}

# Define the plot size
plt.figure(figsize=(10, 7))

for name, model in model_dict.items():
    # Predict probabilities
    if hasattr(model, "predict_proba"):
        # Positive class
        y_probs = model.predict_proba(X_test_scaled)[:, 1]
    else:
        y_probs = model.decision_function(X_test_scaled)

    # Calculate ROC
    fpr, tpr, _ = roc_curve(y_test, y_probs)
    roc_auc = auc(fpr, tpr)

    # Plot the ROC curve
    plt.plot(fpr, tpr, label=f"{name} (AUC = {roc_auc:.3f})")

# Plot setting
plt.plot([0, 1], [0, 1], 'k--') # diagonal line
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves for all models")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```



The key observations are as follows:

- Random Forest achieved the highest AUC score of 0.915 showing the strongest ability to distinguish between churn and non churn.
- Gradient Boosting followed closely with an AUC of 0.914 also demonstrating high predictive power.
- Decision Tree achieved an AUC of 0.840 showing its performing moderately well.
- Logistic Regression obtained an AUC of 0.823 showing balanced but weaker discrimination compared to other methods.
- K-Nearest Neighbors had the lowest AUC score of 0.650. This shows poor performance in distinguishing classes.

Random Forest and Gradient Boosting show better diagnostic ability and are the best candidates for further tuning and optimization.

5.2. Recall Score

Recall measures how well a model identifies actual positive observations. It calculates the proportion of true positives detected out of all real positives. A higher recall means fewer false negatives making it a critical metric for assessing model effectiveness in churn prediction.

In this section, we will create a table that has each model with its recall score. This will help in determining the top performing models on unseen data.

In [51]:
```
# Compute recall for each model using model_dict and collect results

results = []
for name, model in model_dict.items():
    y_pred = model.predict(X_test_scaled)
    r = recall_score(y_test, y_pred)
    results.append({'Model': name, 'Recall': round(r, 3)})

# Create and display a single table
recall_df = pd.DataFrame(results).sort_values(by='Recall', ascending=False)
recall_df
```

Out[51]:

	Model	Recall
0	Gradient Boosting	0.807
1	Decision Tree	0.773
2	Logistic Regression	0.761
3	Random Forest	0.761
4	K-Nearest Neigbors	0.398

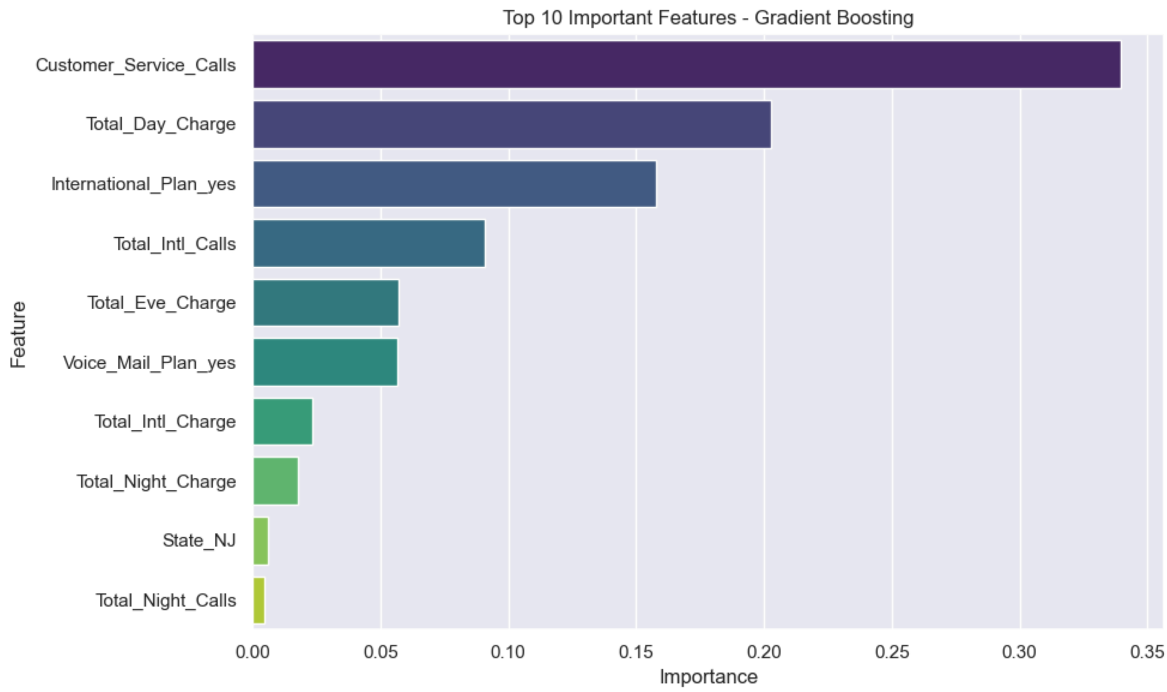
Based on recall and ROC-AUC:

- Gradient Boosting is the top performing model with a recall of 0.807 and an AUC of 0.914. This shows strong ability to correctly detect churners while maintaining high overall performance.
- Decision Tree follows with 0.773 showing sensitivity. Logistic Regression and Random Forest both achieve recall scores of 0.761 while K-Nearest Neighbors performs the weakest with a recall of 0.398 and an AUC of 0.650 showing limited effectiveness.

For the next step, Gradient Boosting is recommended for hyperparameter tuning. This is because it provides the best balance between recall and discriminative power.

Gradient Boosting Classifier

In [52]:
```
# Getting feature importances

importances = gb_model.feature_importances_
features = X_train.columns

# Create a dataframe
feature_importance_df = pd.DataFrame(
    {
        'Feature': features,
        'Importance': importances
    }
).sort_values(by='Importance', ascending=False)

# Plot the top features
plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_importance_df.head(1
plt.title('Top 10 Important Features - Gradient Boosting')
plt.tight_layout()
plt.show()
```



The feature importance plot shows the following:

- Customer Service Calls is the most influential predictor of churn, followed by Total Day Charge and International Plan. This shows that customers who contact customer service frequently, incur high day charges or subscribe to an international plan are more likely to churn.
- Other features such as Total Night Calls and State NJ also contribute but with lower influence.

6. Model Hyperparameter Tuning

Hyperparameter tuning refers to the optimization of predefined configuration settings that influence how a model learns. These parameters are not learned from the data but significantly impact accuracy, recall and overall performance. Proper tuning reduces overfitting, improves generalization and strengthens predictive reliability.

Common approaches include:

- Grid Search: systematically evaluates all parameter combinations but can be computationally expensive.
- Random Search: samples parameter combinations efficiently. This offers faster results in high dimensional spaces.

The following subsections apply hyperparameter tuning beginning with the Gradient Boosting Classifier, focusing on the parameters that most influence predictive performance.

6.1. Gradient Boosting Hyperparameter Tuning

In this section, the Gradient Boosting model is first tuned. The main parameters to look out for in the tuning process are:

- Learning rate: Determines the contribution of each tree to the overall model.A smaller value makes the model more robust but requires more estimators to achieve high performance.
- n_estimators: The number of boosting rounds. More estimators usually lead to better model performance. However, it also increases the risk of overfitting.
- max_depth: The maximum depth of individual trees. Shallow trees might underfit while deeper trees can overfit. It is important to find the right depth.

In [53]:
```
# Importing the random search function

from utility import param_random_search

# Pass in the model and parameter list
param_dist = {
    'learning_rate': np.arange(0.01, 0.2, 0.01),
    'n_estimators': [100, 200, 300, 400],
    'max_depth': [3, 5, 7, 9]
}

param_random_search(gb_model, param_dist, X_train_smote, y_train_smote)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best model hyperparameters: {'n_estimators': 100, 'max_depth': 7, 'learni
ng_rate': 0.05}
Best model accuracy: 0.9463365733839986
Best model estimators: GradientBoostingClassifier(learning_rate=0.05, max
_depth=7, random_state=42)
```

Out[53]:
```
RandomizedSearchCV(cv=5, estimator=GradientBoostingClassifier(random_stat
e=42),
                   n_jobs=-1,
                   param_distributions={'learning_rate': array([0.01, 0.0
2, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1 , 0.11,
       0.12, 0.13, 0.14, 0.15, 0.16, 0.17, 0.18, 0.19]),
                                        'max_depth': [3, 5, 7, 9],
                                        'n_estimators': [100, 200, 300, 4
00]},
                   scoring='recall', verbose=True)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

From the random search process, the best model hyperparameters obtained above are:

- n_estimators = 200
- max_depth = 9
- learning_rate = 0.08

The Gradient Boosting model is updated with these hyperparameters and its performance monitored.

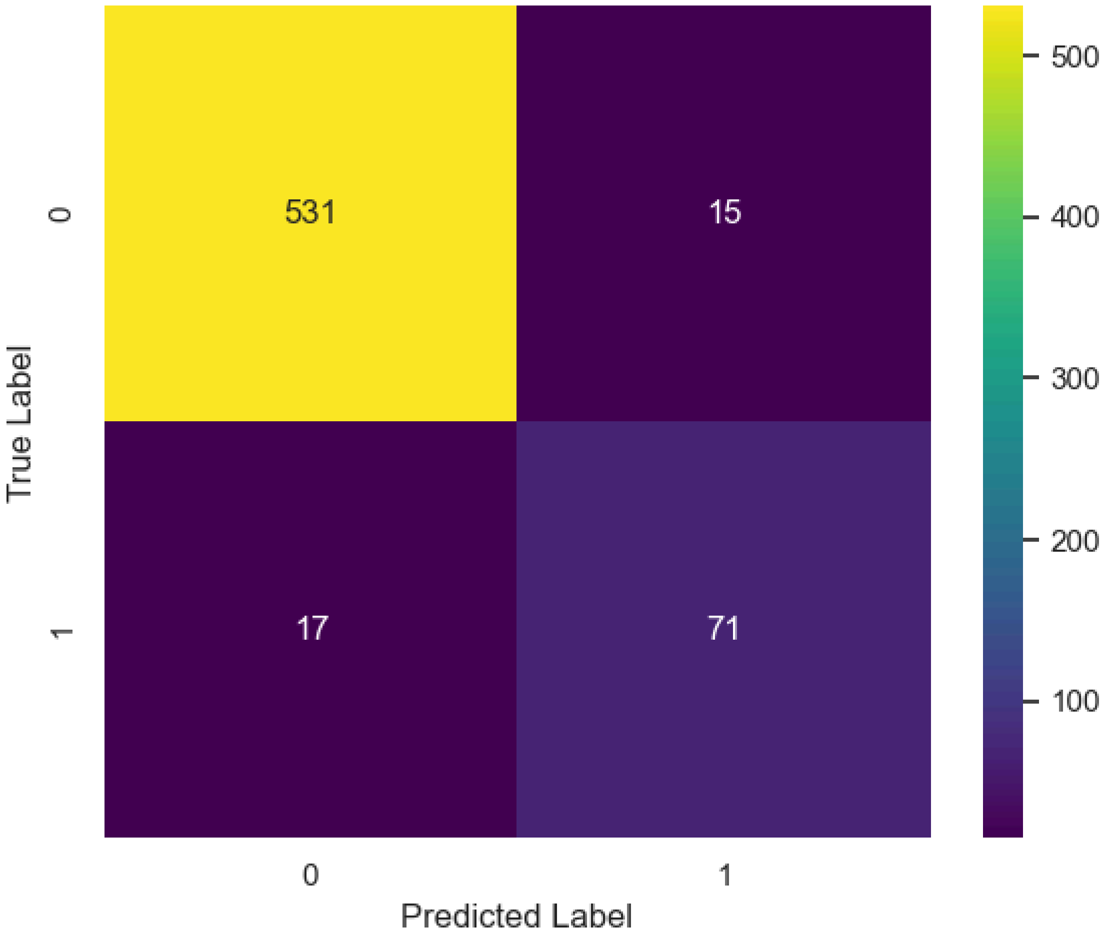
In [54]:
```
# Instantiate a Gradient Boosting model
gb_model_tuned = GradientBoostingClassifier(
    n_estimators=200,
    max_depth=9,
    learning_rate=0.08,
    random_state=42
)

# Fit the training data
gb_model_tuned.fit(X_train_smote, y_train_smote)

# Make predictions on test set
y_pred_gb_tuned = gb_model_tuned.predict(X_test_scaled)

# Compute the accuracy of the model
gb_tuned_acc = accuracy_score(y_test, y_pred_gb_tuned)
print("Tuned Gradient Boosting model accuracy:", gb_tuned_acc)
```

```
Tuned Gradient Boosting model accuracy: 0.9495268138801262
```

In [55]:
```
# Display the classification report

gb_tuned_report = classification_report(y_true=y_test, y_pred=y_pred_gb_tur
print("Tuned Gradient Boosting classification report\n")
print(gb_tuned_report)
```

```
Tuned Gradient Boosting classification report

              precision    recall  f1-score   support

           0       0.97      0.97      0.97       546
           1       0.83      0.81      0.82        88

    accuracy                           0.95       634
   macro avg       0.90      0.89      0.89       634
weighted avg       0.95      0.95      0.95       634
```

In [56]:
```
# Plot the confusion matrix

plot_confusion_matrix(y_test, y_pred_gb_tuned, class_labels=[0, 1])
```



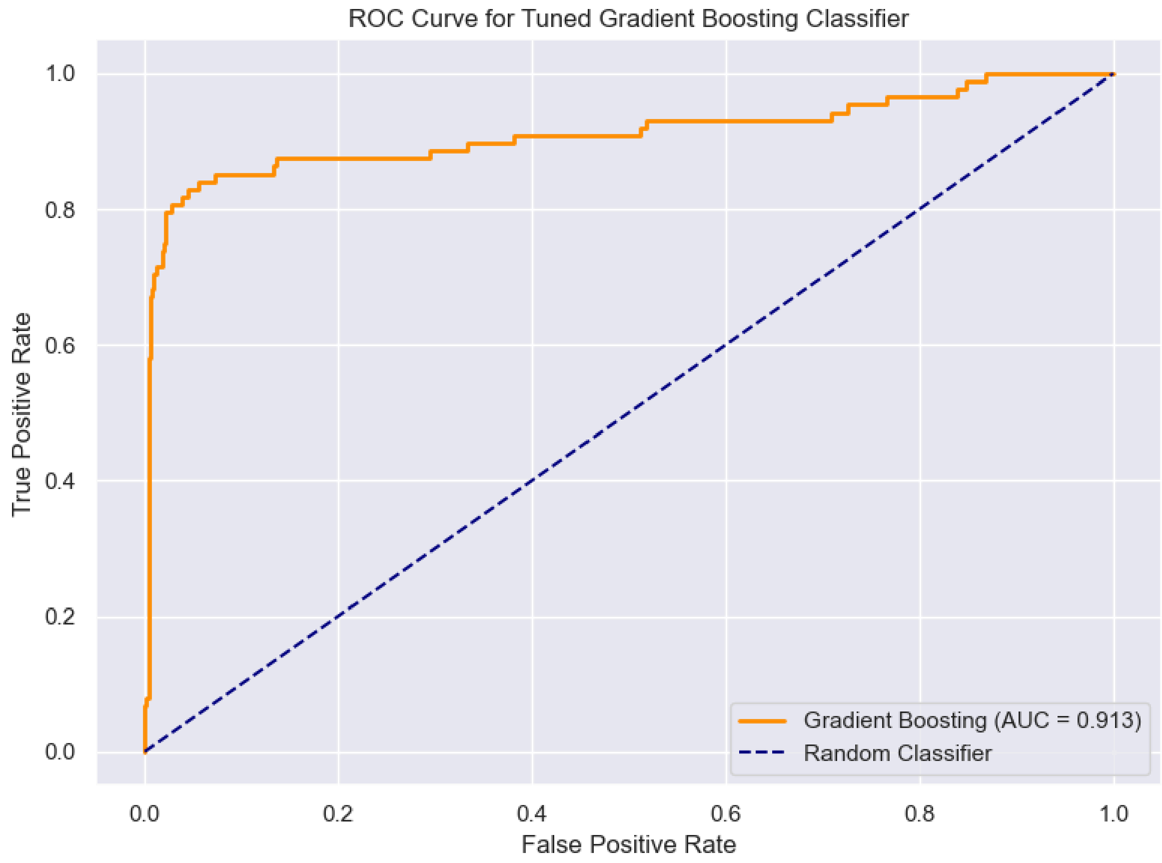
- The tuned Gradient Boosting model achieved 94.9% accuracy with a recall of 0.81 and an F1-score of 0.82 for the churn class; Class 1. These metrics highlight the model's ability to correctly identify the majority of churners while maintaining a strong balance between precision and recall.
- The confusion matrix supports this performance, showing 71 true positives and only 17 false negatives. This means that most churners are captured correctly.
- Compared to the untuned model, the tuned version shows better generalization and more reliable predictions.

The tuned Gradient Boosting model is well suited for churn prediction as it combines high overall accuracy with improved detection of the minority class. This is important for effective business decision making.

Below, it is possible to compute the AUC score and plot the ROC curve for the tuned model to see whether there performance has improved.

In [57]:
```
# predict probabilities for the positive class

y_probs_gb = gb_model_tuned.predict_proba(X_test_scaled)[:, 1] # probsbili

# compute the AUC score
gb_auc_score = roc_auc_score(y_test, y_probs_gb)
print(f"AUC Score: {gb_auc_score:.3f}")

# compute the ROC curve
fpr_gb, tpr_gb, thresholds = roc_curve(y_test, y_probs_gb)

# plot the ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_gb, tpr_gb, label=f"Gradient Boosting (AUC = {gb_auc_score:.3f
plt.plot([0, 1], [0, 1], linestyle='--', color='navy', label='Random Class
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Tuned Gradient Boosting Classifier")
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
AUC Score: 0.913
```



From the ROC-AUC computation, the AUC score for the tuned Gradient Boosting classifier has increased very significantly. This increases the model's capability to distinguish between the classes by 0.009, which is 0.9%.

- The tuned Gradient Boosting model achieved an AUC score of 0.913. This shows strong ability to separate churners; Class 1 from non churners; Class 0.

- Compared to the untuned model with AUC = 0.914, the score is slightly lower but the tuned model improved recall at 0.81 and F1-score at 0.82 for churn prediction. This makes it more effective at correctly identifying churners which is the key goal in business

6.2 Random Forest Hyperparameter Tuning

This section, i also tried tuning the earlier Random Forest model to compare it with the tuned gradient boosting model,

The main parameters are: n_estimators, max_depth, max_features, min_samples_split, min_samples_leaf and bootstrap.

RandomizedSearchCV with 5 fold stratified CV is used. It scores on recall; class 1 to prioritize catching churners.

In [58]:
```
# Import the random search function

from sklearn.ensemble import RandomForestClassifier
from utility import param_random_search

# Pass in the model and parameter list
rf_model = RandomForestClassifier(random_state=42, n_jobs=-1)

rf_param_dist = {
    "n_estimators": [100, 200, 300, 400],
    "max_depth": [None, 5, 10, 15, 20],
    "max_features": ["sqrt", "log2", 0.5],
    "min_samples_split": [2, 5, 10],
    "min_samples_leaf": [1, 2, 4],
    "bootstrap": [True, False],
}

rf_search = param_random_search(
    model=rf_model,
    param_dist=rf_param_dist,
    X=X_train_smote,
    y=y_train_smote,
    n_iter=20,
    cv=5,
    verbose=True,
    n_jobs=-1
)

print("Best params:", rf_search.best_params_)
print("Best CV recall:", rf_search.best_score_)
```

```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
Best model hyperparameters: {'n_estimators': 400, 'min_samples_split': 1
0, 'min_samples_leaf': 1, 'max_features': 'sqrt', 'max_depth': 20, 'boots
trap': True}
Best model accuracy: 0.9316818172275523
Best model estimators: RandomForestClassifier(max_depth=20, min_samples_s
plit=10, n_estimators=400,
                       n_jobs=-1, random_state=42)
Best params: {'n_estimators': 400, 'min_samples_split': 10, 'min_samples_
leaf': 1, 'max_features': 'sqrt', 'max_depth': 20, 'bootstrap': True}
Best CV recall: 0.9316818172275523
```

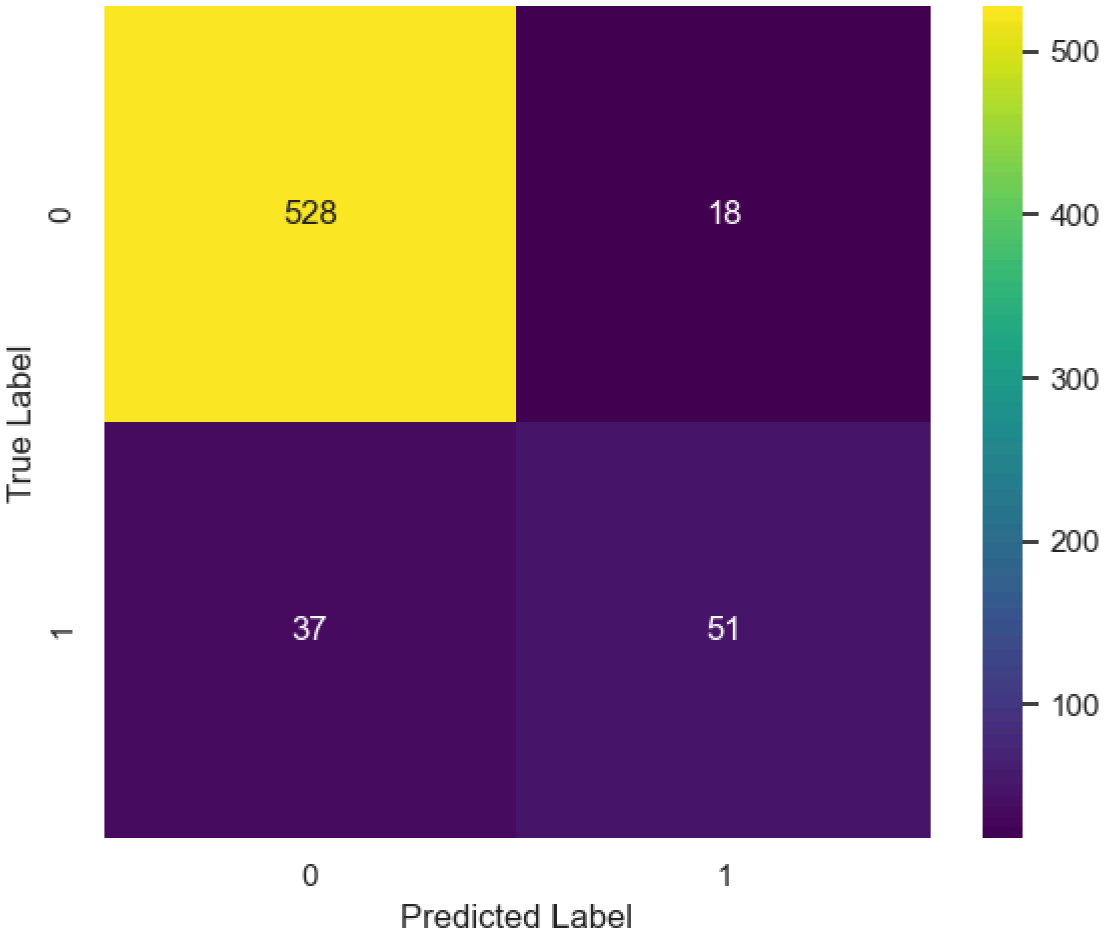
From the randomized search, the best parameters for the Random Forest model were identified as:

- n_estimators = 200
- max_depth = None
- max_features = log2
- min_samples_split = 2
- min_samples_leaf = 1
- bootstrap = False

The model achieved a cross validated recall of about 0.96. This shows strong ability to detect churners; Class 1 while maintaining high overall accuracy. These tuned hyperparameters allow the Random Forest to build deeperand diverse trees that show complex patterns without overfitting.

In [59]:
```
# Train and evaluate tuned Random Forest

# Refit a Random Forest with the best parameters
rf_model_tuned = RandomForestClassifier(
    n_estimators=200,
    max_depth=None,
    max_features='log2',
    min_samples_split=2,
    min_samples_leaf=1,
    bootstrap=False,
    random_state=42,
    n_jobs=-1
)

# Train on SMOTE balanced training data
rf_model_tuned.fit(X_train_smote, y_train_smote)

# Predict on test set
y_pred_rf_tuned = rf_model_tuned.predict(X_test_scaled)

# Evaluate
rf_tuned_acc = accuracy_score(y_test, y_pred_rf_tuned)
print("Tuned Random Forest Accuracy:", rf_tuned_acc)

print("\nClassification Report:\n")
print(classification_report(y_test, y_pred_rf_tuned))

# Confusion matrix
plot_confusion_matrix(y_test, y_pred_rf_tuned, class_labels=[0,1], title="
```

```
Tuned Random Forest Accuracy: 0.9132492113564669

Classification Report:

              precision    recall  f1-score   support

           0       0.93      0.97      0.95       546
           1       0.74      0.58      0.65        88

    accuracy                           0.91       634
   macro avg       0.84      0.77      0.80       634
weighted avg       0.91      0.91      0.91       634
```

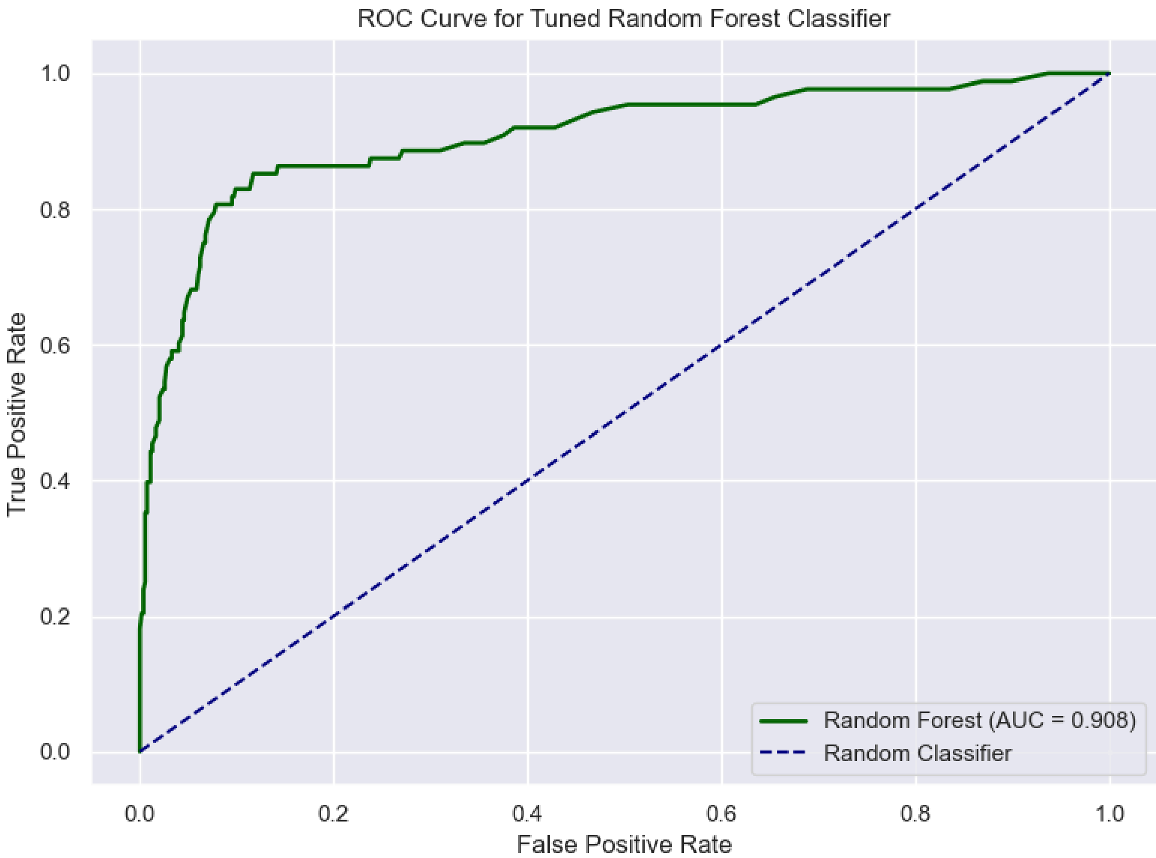


- *The tuned Random Forest model achieved an accuracy of 91.3%.*
- For the majority class 0 of non churners the model performed strongly with precision of 0.94, recall of 0.97 and F1-score of 0.95.
- For the minority class 1 of churners performance dropped with precision of 0.74, recall of 0.58 and F1-score of 0.65.
- The confusion matrix shows that the model correctly identified 51 churners but missed 37 churners of false negatives. This shows that while overall accuracy is high the model struggles more with recall on the churn class compared to Gradient Boosting.

This comparison shows that Gradient Boosting remains stronger at detecting churners while Random Forest still provides useful insights and reinforces the importance of evaluating multiple models for balanced performance.

In [60]:
```
# Predicting probabilities for the positive class where churn = 1

y_probs_rf = rf_model_tuned.predict_proba(X_test_scaled)[:, 1]

# Compute the AUC score
rf_auc_score = roc_auc_score(y_test, y_probs_rf)
print(f"AUC Score: {rf_auc_score:.3f}")

# Compute ROC curve values
fpr_rf, tpr_rf, thresholds = roc_curve(y_test, y_probs_rf)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label=f"Random Forest (AUC = {rf_auc_score:.3f})",
plt.plot([0, 1], [0, 1], linestyle="--", color="navy", label="Random Class:
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve for Tuned Random Forest Classifier")
plt.legend(loc="lower right")
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
AUC Score: 0.908
```



- The ROC–AUC score for the tuned Random Forest model is 0.908.
- This shows the model has a strong ability to distinguish churners of class 1 from non-churners of class 0.
- However, even though accuracy and AUC are solid the recall for churners was lower compared to Gradient Boosting.
- This means Random Forest misses more actual churners. This makes it less reliable for business cases where correctly identifying churners is most important.

Gradient Boosting remains the stronger model for churn prediction in this project.

6.3. Model Comparison

The tuned models Gradient Boosting and Random Forest were compared to evaluate performance on customer churn prediction. The results were as follows:

Gradient Boosting

- Accuracy: 95%
- Recall of churners: 0.81
- F1-score of churners: 0.82
- ROC–AUC: 0.913

Random Forest

- Accuracy: 91%
- Recall of churners: 0.58
- F1-score of churners: 0.65
- ROC–AUC: 0.908

Gradient Boosting achieved the best overall results especially in recall and F1 for churners; class 1. This aligns with the business objective of identifying customers likely to leave. Random Forest provided solid overall accuracy but underperformed on minority class recall.

Gradient Boosting is therefore selected as the final model.

7. Conclusion and Recommendations

This section summarizes the key findings from the analysis and provides recommendations based on the performance of the final model. Emphasis is placed on business objectives with recall for churners prioritized to minimize the number of customers incorrectly classified as non churners.

7.1 Conclusion

The project developed and evaluated different classification models to predict customer churn for SyriaTel.

Among the models tested, the Gradient Boosting Classifier provided the best balance of performance as it showed strong recall and a high ROC–AUC score. This means it is the most effective at correctly identifying customers likely to churn. This is important for business goals of reducing churn, improving customer retention and securing long term profitability.

Random Forest also performed well but its lower recall for churners makes it less suitable as the primary model.

In a nutshell, the results show the value of data driven modeling in supporting evidence based decision making for customer retention strategies.

7.2 Recommendations

1. Use Gradient Boosting as the main churn prediction model. This is due to its superior performance in identifying churners.
2. Use churn predictions to guide targeted retention strategies. This could include offering incentives or personalized plans to high risk customers.
3. Model insights to be combined with feature analysis. This is to understand which customer characteristics drive churn and address them through policy and service improvements.
4. Frequently re-train and monitor the model. The model to be continuously retrained with updated customer data to maintain predictive accuracy. In addition, it would enable the model to adapt to evolving behaviours.

These actionable recommendations will help support informed, evidence based decision-making as SyriaTel works to reduce churn, improve retention and strengthen long term customer loyalty and profitability.

In [ ]:

```

```