SyriaTel Customer Churn Prediction

1.0 Business Understanding

1.1 Introduction: Understanding Customer Churn in the Telecommunications Industry ¶

Customer churn is a significant challenge in the telecommunications industry, where it measures the percentage of customers who discontinue their service over a specified period. High churn rates can signal underlying issues such as poor service quality, inadequate customer support, or lack of transparency in billing. For telecom companies, addressing churn is crucial as acquiring new customers is considerably more expensive than retaining existing ones. Additionally, loyal customers often contribute more to long-term revenue and growth. Thus, reducing churn not only stabilizes revenue but also enhances profitability by fostering customer loyalty.

References:

1. Complete Guide to Reduce Churn in Telecom (https://www.lightico.com/blog/complete-guide-to-reduce-churn-in-telecom/#:~:text=In%20terms%20of%20telecoms%2C%20the,in%20any%20given%20ti - Lightico Blog
2. 2019 Telecom Churn Survey (https://techsee.com/resources/reports/2019-telecom-churn-survey/) - Techsee
3. From Retention to Revenue: How to Reduce Churn Rate in Telecom Industry (https://maxbill.com/blog/from-retention-to-revenue-how-to-reduce-churn-rate-in-telecom-industry/) - Maxbill Blog

1.2 Problem Statement

SyriaTel is currently grappling with challenges related to customer retention. To ensure sustainable growth and enhance its competitive edge, SyriaTel needs to understand and predict customer churn. By analyzing customer usage patterns and demographic information, SyriaTel can identify which customers are at risk of leaving. This early identification will enable the company to implement targeted retention strategies, thereby reducing churn rates and improving overall customer satisfaction and loyalty.

1.3 Objectives

The following are the objectives of this project:

- **Identify key factors leading to customer churn:** Examine the dataset to uncover the most influential features associated with customer churn.
- **Develop predictive models:** Construct and assess predictive models, including logistic regression models and decision tree models, to estimate the likelihood of customer churn based on available data.

- **Provide recommendations:** Deliver insights and recommendations derived from the models to SyriaTel for developing effective customer retention strategies.

1.4 Stakeholders and Usage

- **SyriaTel management:** SyriaTel's management team can utilize the project's findings to better understand customer behavior and the factors influencing churn. This knowledge will inform strategic decisions on customer retention initiatives and resource allocation.
- **Marketing department:** The marketing department can use the churn predictions to tailor marketing campaigns, aiming to address specific issues that contribute to churn. This could involve creating promotions or special offers to retain high-risk customers.

1.5 Conclusion

This project has significant implications for SyriaTel in addressing the challenge of customer churn. By applying advanced data analytics and predictive modeling, SyriaTel can gain valuable insights into the factors driving customer attrition. Implementing the recommendations based on these insights will enable the company to enhance its customer retention strategies, reduce churn rates, and ultimately improve profitability and customer satisfaction. The project's outcomes will help SyriaTel not only stabilize its revenue stream but also strengthen its position in the competitive telecommunications market.

2.0 Data Understanding

The [Churn in Telecom's dataset (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset)](https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset) contains information about whether or not a customer churned from the SyriaTel firm based on certain features in the dataset. Using this dataset, we are able to develop predictive models that help SyriaTel determine whether a customer will abandon their services based on the information provided by the customer.

The dataset contains 3333 rows and 21 columns, with each column representing the following:

- **State:** Represents the U.S. state where the customer resides represented by a two-letter code.
- **Account Length:** The number of days the customer has had an account.
- **Area Code:** The area code of the customer's phone number, which often indicates the geographic region.
- **Phone Number:** The customer's phone number, typically used as a unique identifier for each customer.
- **Internation Plan:** Indicates whether the customer has an international calling plan.
- **Voice Mail Plan:** Indicates whether the customer has a voice mail plan. otherwise false.
- **Number Vmail Messages:** The number of voicemails the customer has sent.
- **Total Day Minutes:** Total number of minutes the customer has been in calls during the day.
- **Total Day Calls:** Total number of calls the user has done during the day.

- **Total Day Charge:** Total amount of money the customer was charged by the Telecom company for calls during the day.
- **Total Eve Minutes:** Total minutes of calls made by the customer during the evening.
- **Total Eve Calls:** Total number of calls the customer has done during the evening.
- **Total Eve Charge:** Total amount of money the customer was charged by the Telecom company for calls during the evening.
- **Total Night Minutes:** Total minutes of calls made by the customer during the night.
- **Total Night Calls:** Total number of calls the customer has done during the night.
- **Total Night Charge:** Total amount of money the customer was charged by the Telecom company for calls during the night.
- **Total Intl Minutes:** Total number of minutes the user has been in international calls.
- **Total Intl Calls:** Total number of international calls the customer has done.
- **Total Intl Charge:** Total amount of money the customer was charged by the Telecom company for international calls.
- **Customer Service Calls:** Number of calls the customer has made to customer service.
- **Churn:** Indicates whether a customer has terminated their contract.

2.1 Load and Explore the Dataset

In [1]:

```
# Importing the relevant libaries
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelE
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.metrics import auc, ConfusionMatrixDisplay, accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.over_sampling import RandomOverSampler
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, precision_score, recall_sco
from matplotlib import pyplot as plt
from sklearn.model_selection import cross_val_score
%matplotlib inline
```

In [2]:

```
# Reading the dataset and displaying the first 10 rows
telcom_data = pd.read_csv('data/telcom_churn.csv')
telcom_data.head(10)
```

Out[2]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	tota da charg
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.0
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.4
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.3
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.9
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.3
5	AL	118	510	391-8027	yes	no	0	223.4	98	37.9
6	MA	121	510	355-9993	no	yes	24	218.2	88	37.0
7	MO	147	415	329-9001	yes	no	0	157.0	79	26.6
8	LA	117	408	335-4719	no	no	0	184.5	97	31.3
9	WV	141	415	330-8173	yes	yes	37	258.6	84	43.9

10 rows × 21 columns

In [3]:
```
# Display basic information about the dataset
telcom_data.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
```

The dataset contains 3333 entries with 21 columns with 16 numerical columns and 5 categorical columns. The phone number column is indicated as a categorical column therefore we should expect no duplicate entries per phone number. The international plan and voice mail plan columns are represented as yes and no and will therefore need to be encoded to the relevant values. The churn column should also be turned into an integer column since it is represented as a boolean. This is necessary when it comes to modelling. The column names should also be changed by removing the whitespaces.

In [4]:
```
# Display a summary of numerical columns in the dataset
telcom_data.describe()
```

Out[4]:

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

In [5]:
```
# Display a summary of categorical columns in the dataset
telcom_data.describe(include='object')
```

Out[5]:

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333
unique	51	3333	2	2
top	WV	409-3428	no	no
freq	106	1	3010	2411

There seems to be a majority of customers from West Virginia (WV). Also, most customers seem to not have an international or voice mail plan. We can look into why this could be during the analysis.

In [6]:

```
# Check value counts of the categorical columns
categorical_columns = telcom_data.select_dtypes(include=['object', 'cat

for column in categorical_columns:
    print(f"Value counts for {column}:")
    print(telcom_data[column].value_counts())
    print()
```

```
Value counts for state:
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
MI     73
VT     73
ID     73
TX     72
UT     72
IN     71
KS     70
MD     70
NC     68
MT     68
NJ     68
WA     66
CO     66
NV     66
MS     65
MA     65
RI     65
AZ     64
MO     63
FL     63
ME     62
ND     62
NM     62
DE     61
OK     61
NE     61
SD     60
SC     60
KY     59
IL     58
NH     56
AR     55
DC     54
GA     54
HI     53
TN     53
AK     52
LA     51
PA     45
IA     44
CA     34
Name: state, dtype: int64

Value counts for phone number:
409-3428    1
388-8797    1
399-6642    1
371-4306    1
411-8140    1
           ..
```

```
390-1760    1
392-6647    1
339-9631    1
371-2418    1
394-3048    1
Name: phone number, Length: 3333, dtype: int64

Value counts for international plan:
no     3010
yes     323
Name: international plan, dtype: int64

Value counts for voice mail plan:
no     2411
yes     922
Name: voice mail plan, dtype: int64

Value counts for churn:
False    2850
True      483
Name: churn, dtype: int64
```

We can keep the state column for our analysis to see how many customers churn by state. The internation plan, voice mail plan and churn columns all contain unique values i.e yes or no, True or false and no other values indicating there is little cleaning required with these columns. There is an evident class imbalance within our dataset with the **2850 customers not churning** and **483 customers churning** represented by False and True respectively. This will be addressed during the data preparation phase.

In [7]:

```
# Looking at the value counts of some additional columns
print(telcom_data['area code'].value_counts())
print(telcom_data['number vmail messages'].value_counts())
print(telcom_data['customer service calls'].value_counts())
```

```
415    1655
510     840
408     838
Name: area code, dtype: int64
0     2411
31      60
29      53
28      51
33      46
27      44
30      44
24      42
26      41
32      41
25      37
23      36
36      34
35      32
22      32
39      30
37      29
34      29
21      28
38      25
20      22
19      19
40      16
42      15
17      14
41      13
16      13
43       9
15       9
18       7
44       7
14       7
45       6
12       6
46       4
13       4
47       3
8        2
48       2
50       2
9        2
11       2
49       1
10       1
4        1
51       1
Name: number vmail messages, dtype: int64
1    1181
2     759
0     697
3     429
4     166
5      66
6      22
7       9
9       2
```

```
8      2
Name: customer service calls, dtype: int64
```

These columns do not contain any unique values that would need to be removed. The area code column has 3 unique area codes as per our dataset and thus we can treat it as a categorical column instead of a numerical column. When it comes to duplicates in the columns, it is expect that for instance with regards to the customer service calls that many customers would make the same amount of calls and many customers can reside or have the same area code. This needs to be considered when it comes to data cleaning.

2.2 Initial Data Cleaning

In [8]:
```
# Convert area_code to categorical just for analysis
telcom_data['area code'] = telcom_data['area code'].astype('category')
```

In [9]:
```
# Removing whitespaces from column names
telcom_data.columns = telcom_data.columns.str.replace(' ','_')
telcom_data.columns
```

Out[9]:
```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_message
s',
       '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_charg
e',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

We removed the whitespaces from the columns to ensure that there is uniformity in naming and also easy access to the relevant columns.

In [10]:
```
# Remove the hyphen in the phone number column
telcom_data['phone_number'] = telcom_data['phone_number'].str.replace('
```

We remove the hyphen from the phone numbers to also ensure unformity within our dataset making it easier to work with and when it comes to analysis if needed.

In [11]:
```python
# Check for missing values in the dataset
telcom_data.isnull().sum()
```

Out[11]:
```
state                     0
account_length            0
area_code                 0
phone_number              0
international_plan        0
voice_mail_plan           0
number_vmail_messages     0
total_day_minutes         0
total_day_calls           0
total_day_charge          0
total_eve_minutes         0
total_eve_calls           0
total_eve_charge          0
total_night_minutes       0
total_night_calls         0
total_night_charge        0
total_intl_minutes        0
total_intl_calls          0
total_intl_charge         0
customer_service_calls    0
churn                     0
dtype: int64
```

The dataset is not missing any values as also indicated in our inital dataset summary. In our inital check we also looked for any unique characters that would otherwise signify missing data which we did not find.

In [12]:
```python
# Check if the dataset has any duplicate values
telcom_data.duplicated().sum()
```

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

There are no duplicated rows. However let us check if there are any duplicated values for the phone number column which should be unique for every customer.

In [13]:
```python
# Check if there are any duplicated values per phone number
telcom_data['phone_number'].duplicated().sum()
```

Out[13]:
```
0
```

There are no duplicated columns per phone number and since our phone number is a unique column identifying a customer in our dataset, we can set this column as the index of our dataset.

In [14]:
```python
# Setting the phone number as our index
telcom_data.set_index('phone_number', inplace=True)
```

In [15]:
```
# Final look at our dataset before analysis
telcom_data.head()
```

Out[15]:

	state	account_length	area_code	international_plan	voice_mail_plan	n
phone_number						
3824657	KS	128	415	no	yes	
3717191	OH	107	415	no	yes	
3581921	NJ	137	415	no	no	
3759999	OH	84	408	yes	no	
3306626	OK	75	415	yes	no	

In [16]:
```
telcom_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3333 entries, 3824657 to 4004344
Data columns (total 20 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   category
 3   international_plan      3333 non-null   object
 4   voice_mail_plan         3333 non-null   object
 5   number_vmail_messages   3333 non-null   int64
 6   total_day_minutes       3333 non-null   float64
 7   total_day_calls         3333 non-null   int64
 8   total_day_charge        3333 non-null   float64
 9   total_eve_minutes       3333 non-null   float64
 10  total_eve_calls         3333 non-null   int64
 11  total_eve_charge        3333 non-null   float64
 12  total_night_minutes     3333 non-null   float64
 13  total_night_calls       3333 non-null   int64
 14  total_night_charge      3333 non-null   float64
 15  total_intl_minutes      3333 non-null   float64
 16  total_intl_calls        3333 non-null   int64
 17  total_intl_charge       3333 non-null   float64
 18  customer_service_calls  3333 non-null   int64
 19  churn                   3333 non-null   bool
dtypes: bool(1), category(1), float64(8), int64(7), object(3)
memory usage: 501.4+ KB
```

We will address any other data cleaning issues when it comes to the data preparation phase. For now the inital cleaning of the data makes it suitable for analysis.

2.3 Feature Relevance and Justification:

Customer Relevance: Features such as account length, service plans, and call metrics are crucial as they directly relate to customer behavior and satisfaction. For example, account length might correlate with customer loyalty, while call metrics can indicate usage patterns.

Feature Selection:

- **Service Plans (International Plan, Voice Mail Plan):** Important for assessing the impact of service features on churn.
- **Call Metrics (Day, Eve, Night, Intl):** Provide insights into usage patterns that might influence churn.
- **Customer Service Calls:** High interaction with customer service could indicate dissatisfaction, leading to churn.

2.4 Data Quality and Limitations:

- **Data Size:** The dataset size (3,333 rows) is manageable but may limit the model's ability to generalize. The class imbalance in the target variable (churn) could also affect model performance and needs addressing through techniques such as SMOTE.
- **Missing Values:** No missing values were found, ensuring completeness.
- **Duplicates:** No duplicate rows or phone numbers, ensuring unique customer representation.
- **Class Imbalance:** The churn class is imbalanced, with a higher percentage of non-churned customers. This imbalance must be addressed to avoid biased model predictions.

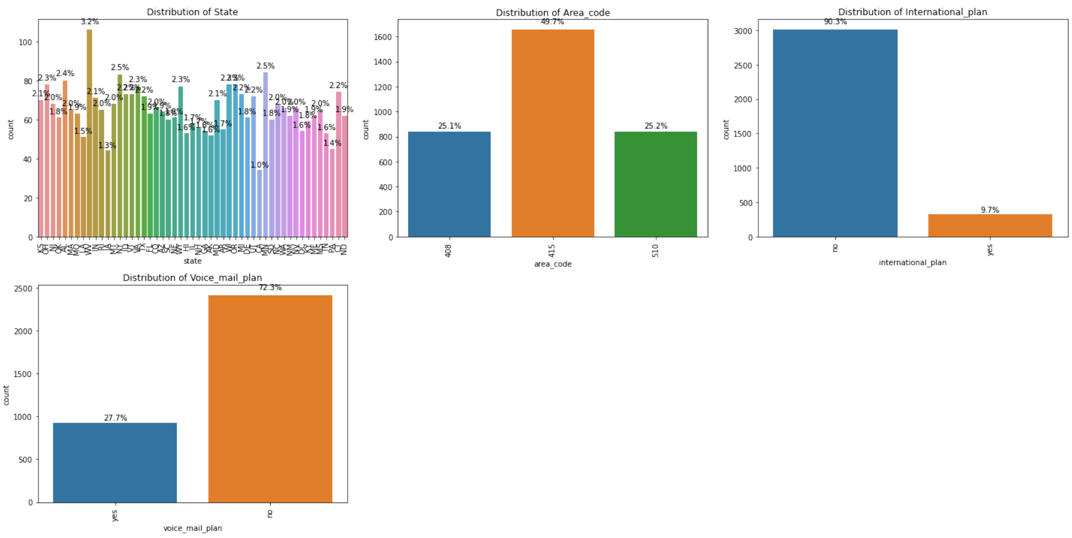
3.0 Exploratory Data Analysis (EDA)

3.1 Univariate Analysis

In [17]:
```
# Identify numerical and categorical columns
numerical_columns = telcom_data.select_dtypes(include=['number']).colum
categorical_columns = [col for col in telcom_data.select_dtypes(include
```

In [18]:

```
# Plot distributions for categorical columns
num_plots = len(categorical_columns)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

plt.figure(figsize=(20, num_rows * 5))

for i, column in enumerate(categorical_columns):
    plt.subplot(num_rows, num_cols, i + 1)
    ax = sns.countplot(data=telcom_data, x=column)
    plt.title(f'Distribution of {column.capitalize()}')
    plt.xticks(rotation=90)

    # Calculate percentages and add text annotations
    total = len(telcom_data)
    for p in ax.patches:
        height = p.get_height()
        percentage = (height / total) * 100
        ax.text(p.get_x() + p.get_width() / 2., height + 0.02 * height,
                f'{percentage:.1f}%',
                ha='center', va='bottom',
                fontsize=10)

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



In [19]:
```
# Get the top 10 states by distribution
top_10_states = telcom_data['state'].value_counts().nlargest(10)
top_10_states
```

Out[19]:
```
WV    106
MN     84
NY     83
AL     80
OH     78
OR     78
WI     78
VA     77
WY     77
CT     74
Name: state, dtype: int64
```

In [20]:
```
# Get the top 10 states with the lowest distribution
bottom_10_states = telcom_data['state'].value_counts().nsmallest(10)
bottom_10_states
```

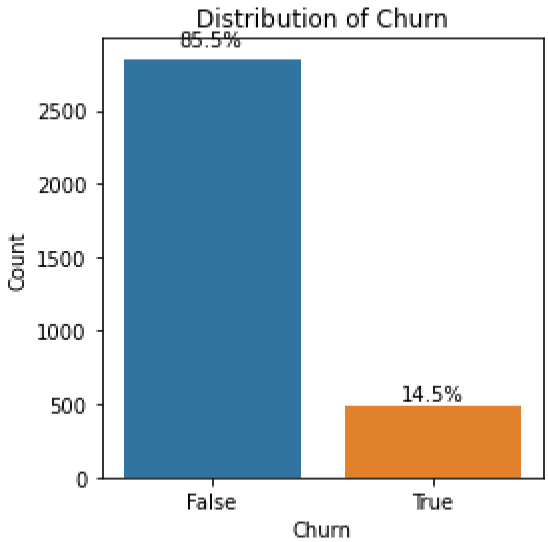
Out[20]:
```
CA    34
IA    44
PA    45
LA    51
AK    52
HI    53
TN    53
DC    54
GA    54
AR    55
Name: state, dtype: int64
```

From the visualization, the state with the most customers is West Virginia with 106 customers and the state with the fewest customers is California with 34 customers. There are some clusters of states with similar customer count. This may indicate regional trends or preferences. Additionally, there may be many factors as to why some states have some more customers than others. This may be due to demographic factors e.g, some states may have a higher average income levels, economic factors such as a lower cost of living in some states that may attract more people and in turn lead to more customers. Moreover some states may have better infrastructure making it easier to access telcom services. SyriaTel can look into some of the factors above to understand the distribution of customers in certain states.

The above can also factor into the area code with majority of customers from the 415 area code. The distribution of area codes can provide insights into customer concentration in specific regions.

The international plan seems not to be a popular service within the telco along with the voice mail plan with over 90% not opting for the international plan and over 70% not opting for the voice mail plan. This may be due to some additional fees, limited need and the rise of alternative communication methods such as social media services or privacy concerns.

In [21]:
```
# Plot count distribution for the 'churn' column
plt.figure(figsize=(4, 4))
ax = sns.countplot(data=telcom_data, x='churn')
plt.title('Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')

# Calculate percentages and add text annotations
total = len(telcom_data)
for p in ax.patches:
    height = p.get_height()
    percentage = (height / total) * 100
    ax.text(p.get_x() + p.get_width() / 2., height + 0.02 * height,
            f'{percentage:.1f}%',
            ha='center', va='bottom',
            fontsize=10)

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

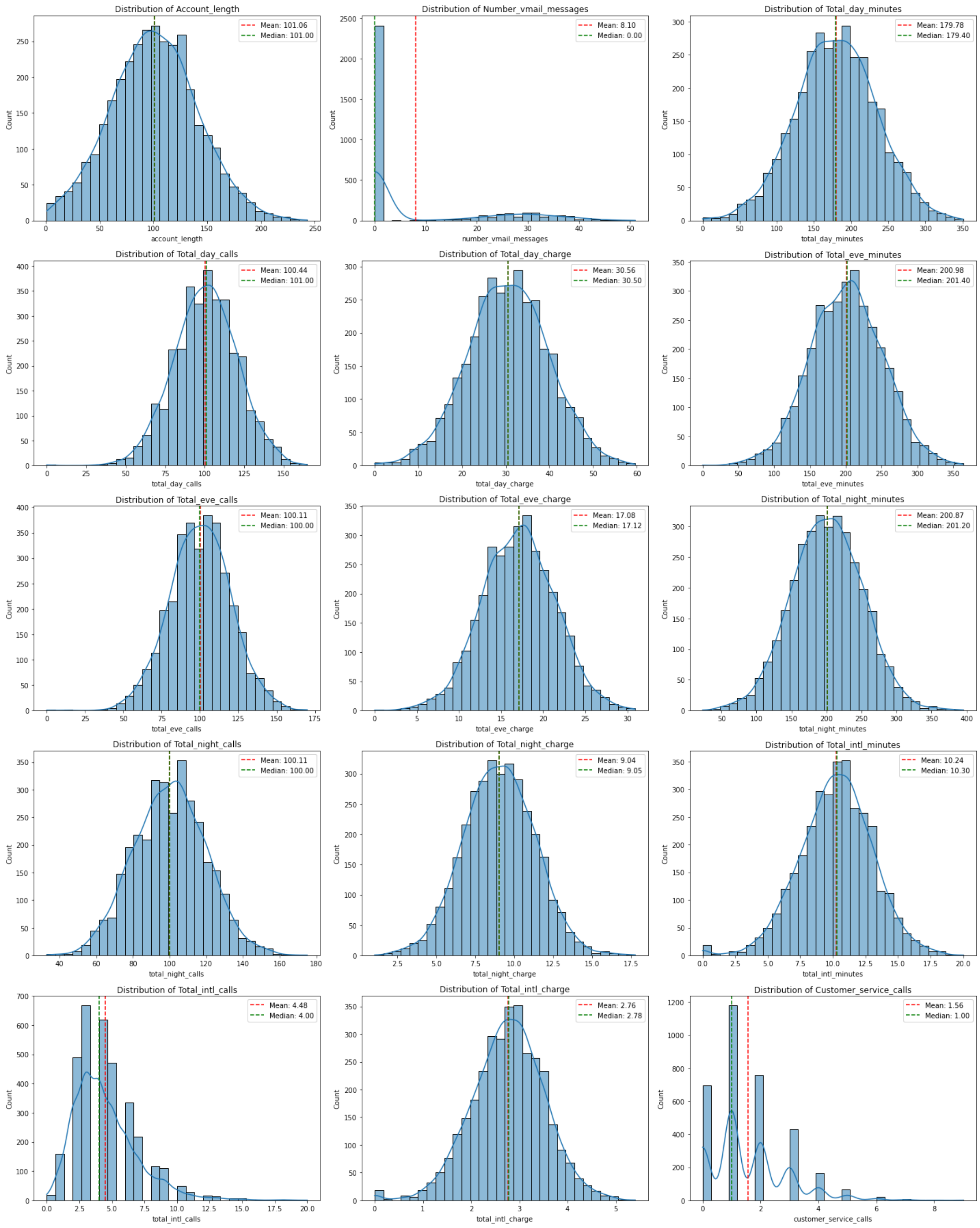


A significant majority of customers (85.5%) have not churned, suggesting that the company has a relatively strong customer retention rate. Only 14.5% of customers have churned, indicating that the company's services and customer experience are generally satisfactory. The high retention rate suggests that customers are generally satisfied with the company's offerings and services. While the overall churn rate is relatively low, identifying the factors contributing to the 14.5% of churn can help the company implement targeted strategies to further improve customer retention.

There is an evident class imbalance that needs to be addressed during preparation before modelling.

In [22]:

```
# Plot distributions for numerical columns
num_plots = len(numerical_columns)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

plt.figure(figsize=(20, num_rows * 5))

for i, column in enumerate(numerical_columns):
    plt.subplot(num_rows, num_cols, i + 1)
    sns.histplot(telcom_data[column], kde=True, bins=30)
    plt.title(f'Distribution of {column.capitalize()}')

    # Calculate mean and median
    mean = telcom_data[column].mean()
    median = telcom_data[column].median()

    # Add mean and median lines
    plt.axvline(mean, color='r', linestyle='--', label=f'Mean: {mean:.2
    plt.axvline(median, color='g', linestyle='--', label=f'Median: {med

    # Add legend
    plt.legend()

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



Many of the distributions exhibit some degree of skewness, indicating that there may be outliers or a concentration of data points in specific ranges. While some distributions appear relatively normal (bell-shaped), others deviate from normality, suggesting that the data might not follow a normal distribution.

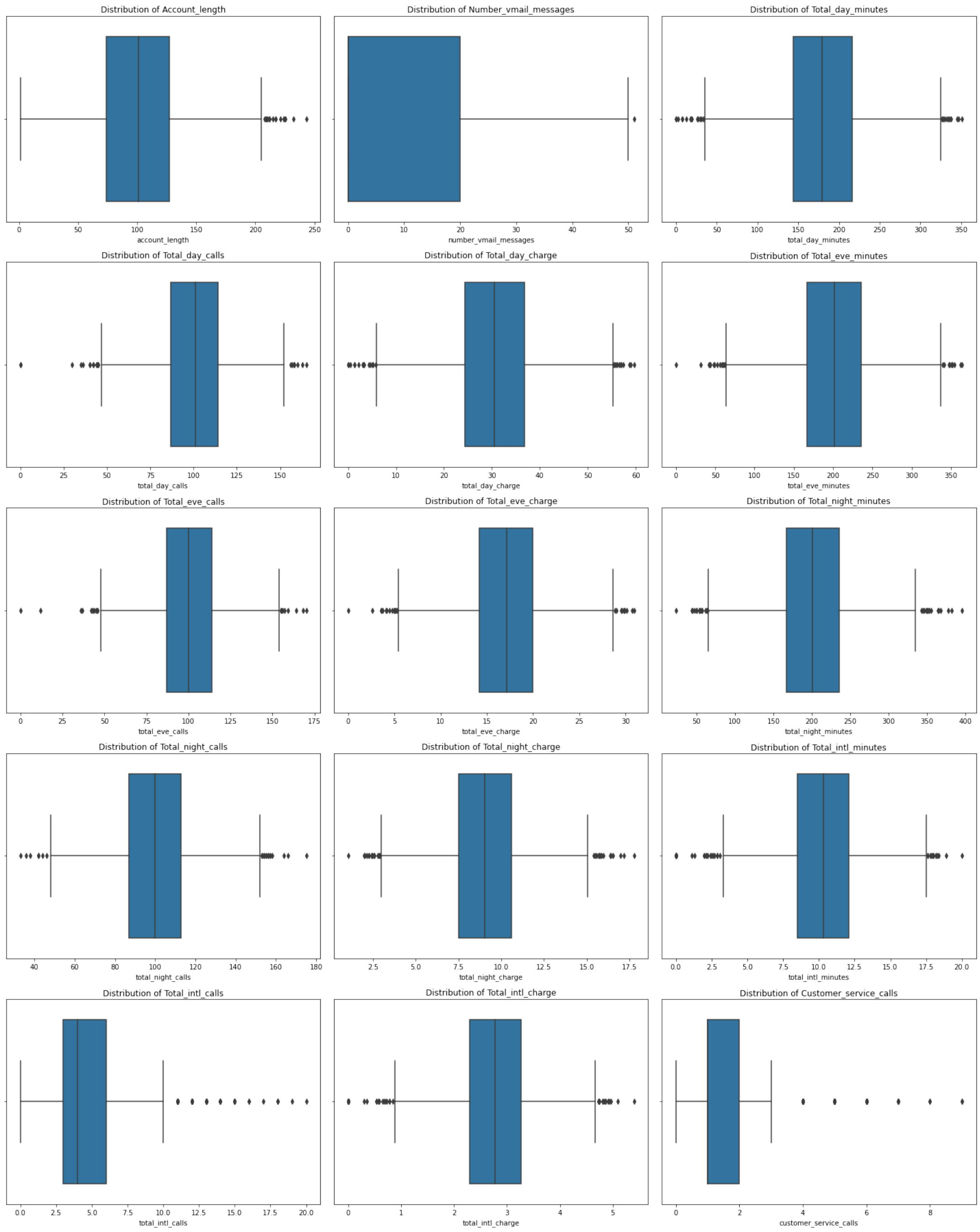
Here are some general observations:

- **Account Length:** The distribution of account length appears roughly normal, with a slight right skew. This suggests that most customers have been with the company for a moderate amount of time, with a smaller number of customers having longer tenures.
- **Number of Vmail Messages:** The distribution of number of voicemail messages is highly skewed to the right, indicating that a small number of customers use voicemail extensively, while the majority of customers use it infrequently or not at all.
- **Total Day Minutes, Calls, and Charge:** The distributions of total day minutes, calls, and charge show a similar pattern, with a slight right skew. This suggests that a majority of customers use a moderate amount of daytime minutes, calls, and incur moderate charges, while a smaller number of customers use significantly more.

- **Total Evening Minutes, Calls, and Charge:** The distributions of total evening minutes, calls, and charge also show a similar pattern, with a slight right skew. This suggests that a majority of customers use a moderate amount of evening minutes, calls, and incur moderate charges, while a smaller number of customers use significantly more.
- **Total Night Minutes, Calls, and Charge:** The distributions of total night minutes, calls, and charge appear relatively normal, with a slight left skew. This suggests that a majority of customers use a moderate amount of night minutes, calls, and incur moderate charges, with a smaller number of customers using significantly less.
- **Total International Minutes, Calls, and Charge:** The distributions of total international minutes, calls, and charge are highly skewed to the right, indicating that a small number of customers use international services extensively, while the majority of customers use them infrequently or not at all.
- **Customer Service Calls:** The distribution of customer service calls is skewed to the right, suggesting that a majority of customers do not require frequent customer service assistance, while a smaller number of customers require more support.

In [23]:

```
# Plot distributions for numerical columns using box plots
num_plots = len(numerical_columns)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols  # Ensure enough rows

plt.figure(figsize=(20, num_rows * 5))

for i, column in enumerate(numerical_columns):
    plt.subplot(num_rows, num_cols, i + 1)
    sns.boxplot(data=telcom_data, x=column)
    plt.title(f'Distribution of {column.capitalize()}')

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



Several of the box plots exhibit outliers, particularly in the distributions of total minutes, calls, and charges. These outliers suggest that there are a small number of customers with extremely high or low usage patterns. The box plots provide a visual representation of the data distribution, including the median, quartiles, and outliers. Some distributions are more symmetrical, while others are skewed, indicating a concentration of data points in certain ranges.

Specific Observations:

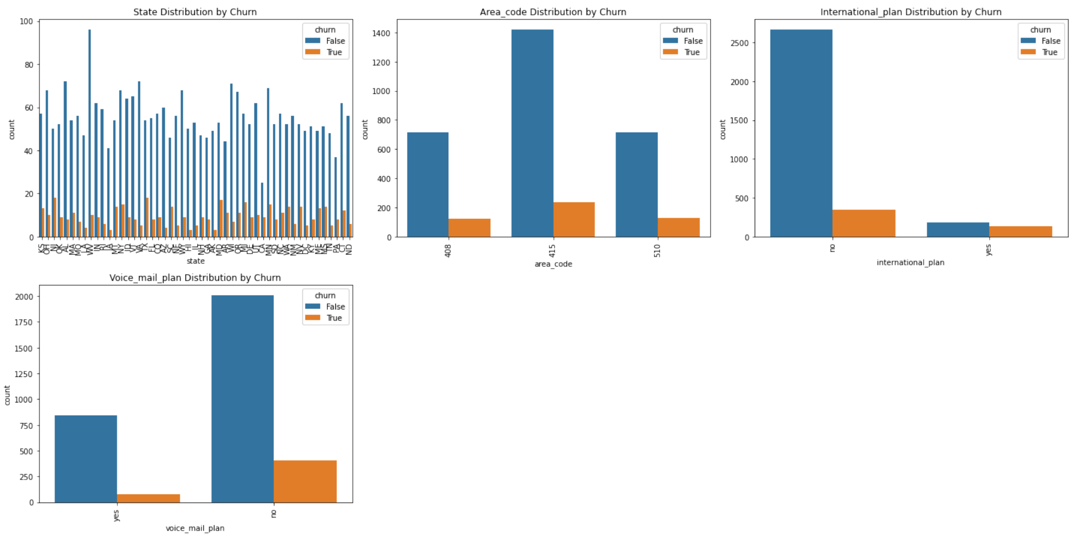
- **Account Length:** The distribution of account length is relatively symmetrical, with a median around 100 months. There are a few outliers on both the high and low ends.
- **Number of Vmail Messages:** The distribution is heavily skewed to the right, with a median close to zero. This indicates that most customers do not use voicemail or use it infrequently.
- **Total Minutes, Calls, and Charges:** The distributions of total minutes, calls, and charges for day, evening, and night usage all show a similar pattern, with a median around the middle of the range and a significant number of outliers.
- **Total International Minutes, Calls, and Charges:** The distributions of total international minutes, calls, and charges are highly skewed to the right, with a median close to zero. This indicates that most customers do not use international services or use them infrequently.
- **Customer Service Calls:** The distribution of customer service calls is skewed to the right, with a median around zero. This suggests that most customers do not require frequent customer service assistance.

Despite the above these outliers contain valuable information which will be important to our models and this we will not be addressing these outliers. Also considering that our dataset is small, removing these outliers may also significantly affect our model's performance.

3.2 Bivariate Analysis

In [24]:

```
# Plot bivariate analysis for categorical columns
num_plots = len(categorical_columns)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols

plt.figure(figsize=(20, num_rows * 5))

for i, column in enumerate(categorical_columns):
    plt.subplot(num_rows, num_cols, i + 1)
    sns.countplot(data=telcom_data, x=column, hue='churn')
    plt.title(f'{column.capitalize()} Distribution by Churn')
    plt.xticks(rotation=90)

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



The following are some observations based on the above:

- **State:** The distribution of churn across states appears relatively uniform, with no clear patterns or outliers. This suggests that state-specific factors might not be a major driver of churn.
- **Area Code:** The distribution of churn by area code shows some variation, with certain area codes having slightly higher or lower churn rates. However, the differences are not significant enough to draw definitive conclusions.
- **International Plan:** Customers without an international plan have a slightly higher churn rate compared to those with, suggesting that the international plan might be more appealing to some customers.
- **Voice Mail Plan:** Customers with voicemail plans have a slightly lower churn rate compared to those without, indicating that the voicemail plan might be a valuable feature for retaining customers.

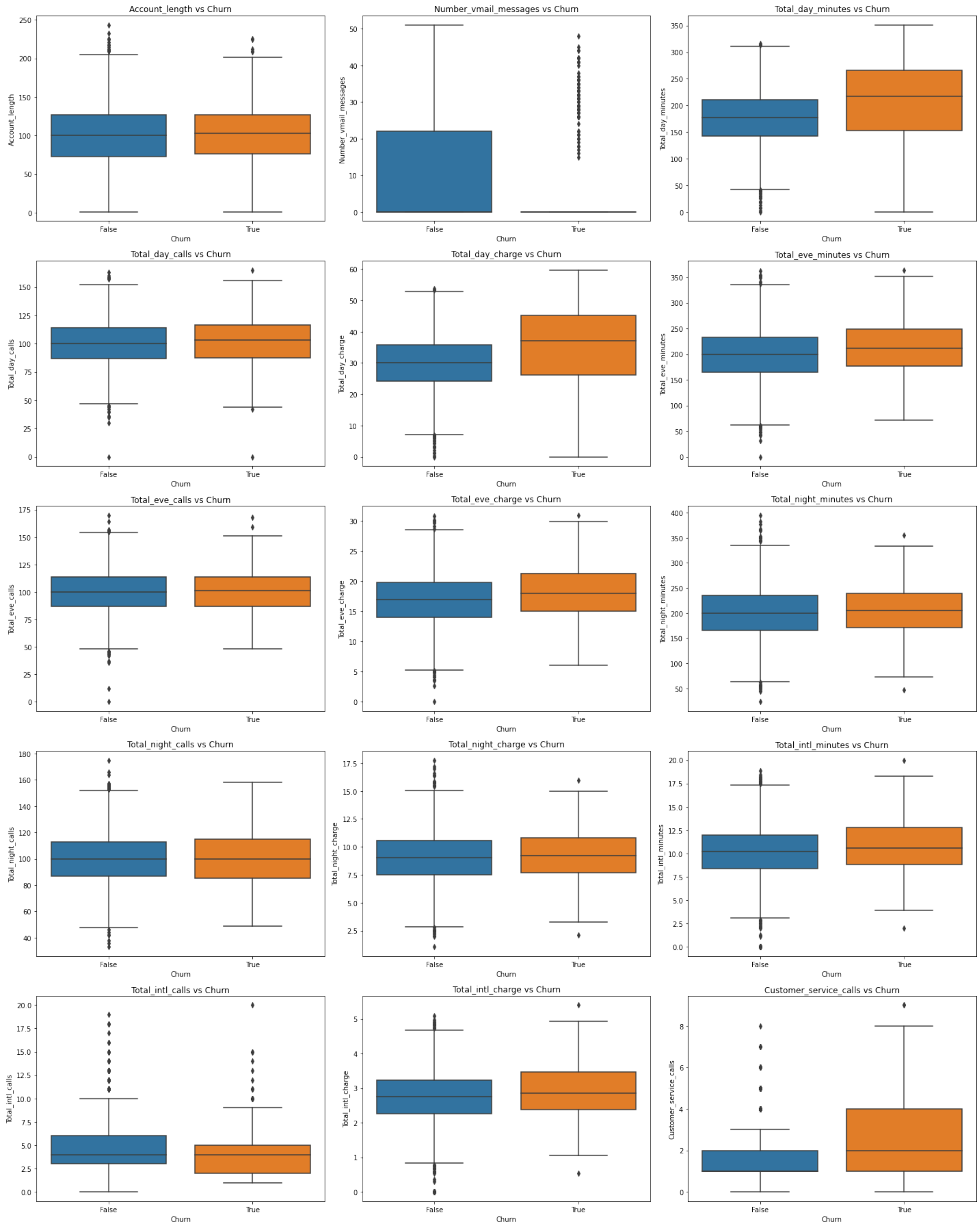
In [25]:

```
# Finding top 5 states by churn
churned_data = telcom_data[telcom_data['churn'] == True]
churn_counts_by_state = churned_data.groupby('state').size().reset_inde
top_5_states_by_churn = churn_counts_by_state.sort_values(by='churn_cou

print(top_5_states_by_churn)
```

```
   state  churn_count
31    NJ           18
43    TX           18
20    MD           17
22    MI           16
23    MN           15
```

In [26]:

```
# Plotting bivariate analysis of numerical columns
# Number of plots
num_plots = len(numerical_columns)
num_cols = 3
num_rows = (num_plots + num_cols - 1) // num_cols  # Calculate rows nee

# Plot distributions for numerical columns using Box Plots
plt.figure(figsize=(20, num_rows * 5))

for i, column in enumerate(numerical_columns):
    plt.subplot(num_rows, num_cols, i + 1)
    sns.boxplot(data=telcom_data, x='churn', y=column)
    plt.title(f'{column.capitalize()} vs Churn')
    plt.xlabel('Churn')
    plt.ylabel(column.capitalize())

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



Based on the box plots, the following insights can be drawn:

1. Account Length: There appears to be a slight overlap in the account length distribution between churned and non-churned customers. However, the median account length is slightly higher for non-churned customers. This might suggest that longer-tenured customers are less likely to churn.

2. Total Day Minutes, Total Eve Minutes, and Total Night Minutes: The distributions for these variables are similar between the two groups, indicating that usage patterns alone might not be a strong predictor of churn.

3. Total Day Charge, Total Eve Charge, and Total Night Charge: There's a slight difference in the distributions of these variables. The median charges are slightly lower for non-churned customers, suggesting that customers who spend less are less likely to churn.

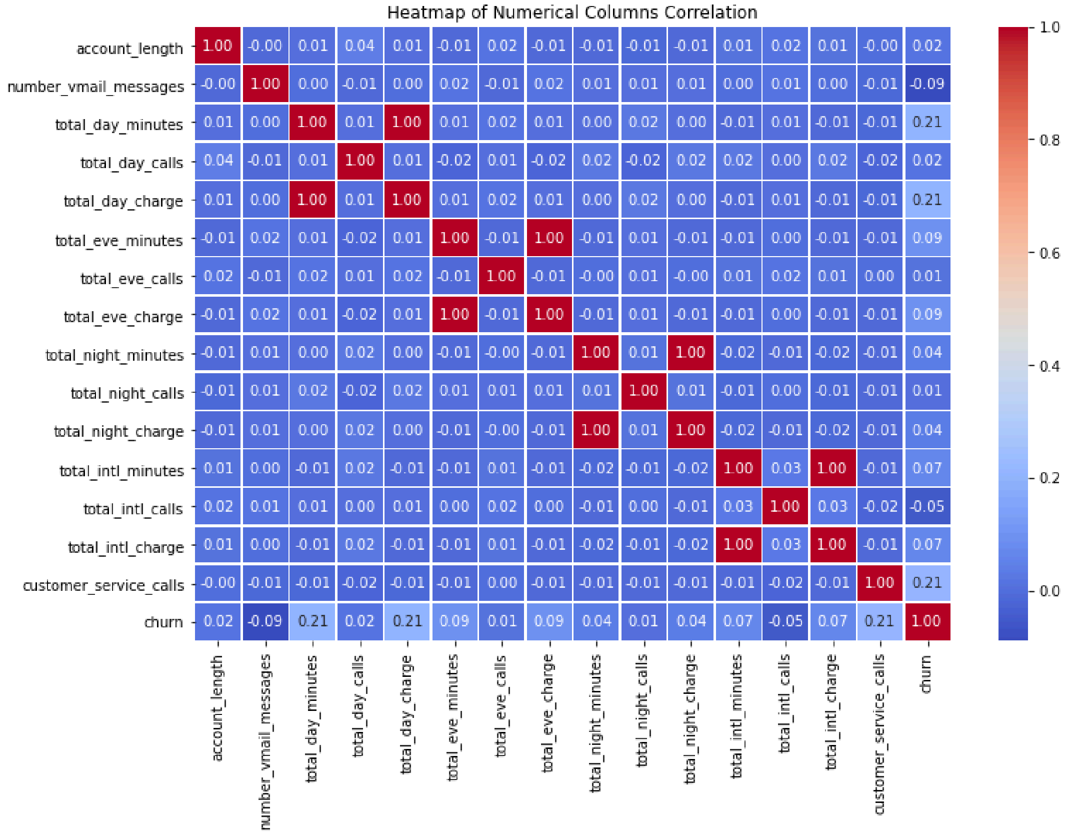
4. Total International Calls and Total International Minutes: The distributions are similar, implying that international usage patterns might not be a significant factor.

5. Customer Service Calls: The distribution for churned customers shows a slightly higher median number of calls. This might indicate that customers who require more customer support are more likely to churn.

3.3 Multivariate Analysis

In [27]:

```
# Calculate the correlation matrix
corr_matrix = telcom_data.corr()

# Plot the heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewi
plt.title('Heatmap of Numerical Columns Correlation')
plt.show()
```



There seems to be a perfect correlation between multiple variables in the dataset. These are: **total_day_charge and total_day_minutes**, **total_eve_charge and total_eve_minutes**, **total_night_charge and total_night_minutes** and **total_intl_charge and total_intl_minutes**. Based on this, we can only include one of each in our models since we need to address multicollinearity. Based on our analysis above we can drop: total_intl_minutes, total_night_minutes, total_day_minutes, total_eve_minutes due to their high multicollinearity and low correlation to the churn target.

4.0 Data Preparation

4.1 Drop Irrelevant Columns

In [28]:
```
# Removing irrelevant columns
telcom_data.drop(['area_code','state','number_vmail_messages','total_da
```

We dropped the above columns since based on our analysis we found that these columns either have a high collinearity with each other and other columns did such as area code, state and number of voice mail messages did not significantly impact our churn target.

4.2 Converting Categorical Variables

In [29]:
```
# Convert categorical varaibles
telcom_data['international_plan'] = LabelEncoder().fit_transform(telcom
telcom_data['voice_mail_plan'] = LabelEncoder().fit_transform(telcom_da
telcom_data['churn'] = telcom_data['churn'].astype(int)
telcom_data.head()
```

Out[29]:

	account_length	international_plan	voice_mail_plan	total_day_calls	tota
phone_number					
3824657	128	0	1	110	
3717191	107	0	1	123	
3581921	137	0	0	114	
3759999	84	1	0	71	
3306626	75	1	0	113	

The international plan and voice mail plan columns are encoded into their relevant labels i.e, 1 being True and 0 being false. The churn column is also converted into an integer column allowing us to use the column directly in our classification models.

4.3 Splitting the Data into Training and Testing Sets

In [30]:
```
# Define features and target variable
X = telcom_data.drop('churn', axis=1)
y = telcom_data['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
```

The dataset is split into train and test with the test set being 30% of our original dataset and the train set being the remaining 70%. We split the dataset initially to prevent data leakage during the other phases in our preprocessing.

4.4 Creating Pipelines for Transformation

In [31]:
```
# Identify numerical and categorical columns
numerical_columns = X_train.select_dtypes(include=['number']).columns
categorical_columns = X_train.select_dtypes(include=['object', 'categor

# Define preprocessing steps
numerical_preprocessor = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_preprocessor = Pipeline(steps=[
    ('encoder', OneHotEncoder(drop='first', sparse=False))
])
```

In [32]:
```
# Combine preprocessors into a ColumnTransformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_preprocessor, numerical_columns),
        ('cat', categorical_preprocessor, categorical_columns)
    ]
)

X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)
```

We identify numerical and categorical columns in our dataset and apply different preprocessing steps for each. The numerical columns are scaled and the categorical columns are encoded if any. We then create a pipeline to apply the relevant preprocessing steps and finally apply the different steps into a single operation using `ColumnTransformer` . Lastly fit apply the transformations on our training data using `fit_transform` on the X_train and apply the learned parameters without re-fitting to the test data that is X_test.

4.5 Handling Class Imbalance

In [33]:
```
# Define SMOTE
smote = SMOTE(random_state=42)

# Fit SMOTE to the training data
X_train_resampled, y_train_resampled = smote.fit_resample(X_train_proce

# Check the class distribution after resampling
print(f'Original class distribution in training set:\n{y_train.value_co
print(f'Resampled class distribution in training set:\n{pd.Series(y_tra
```

```
Original class distribution in training set:
0    1993
1     340
Name: churn, dtype: int64
Resampled class distribution in training set:
1    1993
0    1993
Name: churn, dtype: int64
```

The final step is to apply SMOTE to address class imbalance in the training set to reduce the risk of overfitting to the majority class by creating synthetic examples, which are more generalized compared to simply duplicating existing minority samples.

5.0 Modelling and Evaluation

Here we create two baseline classification models, evaluate them and improve the models by performing hyper-parameter tuning. The intention is to find the best performing model and parameters. To begin we can create functions that can be reused for the different processes.

In [34]:
```
def plot_confusion_matrix(y_true, y_pred, labels, title):
    """
    Plots the confusion matrix.

    Parameters:
    y_true (array-like): True labels.
    y_pred (array-like): Predicted labels.
    labels (list): List of label names.
    title (str): Title of the plot.
    """
    conf_matrix = confusion_matrix(y_true, y_pred)
    print(f"Confusion Matrix for {title}:")
    print(conf_matrix)

    disp = ConfusionMatrixDisplay(conf_matrix, display_labels=labels)
    disp.plot(cmap='Blues', values_format='d')
    plt.title(f'Confusion Matrix for {title}')
    plt.show()
```

In [35]:
```
def print_classification_report(y_true, y_pred, title):
    """
    Prints the classification report.

    Parameters:
    y_true (array-like): True labels.
    y_pred (array-like): Predicted labels.
    title (str): Title for the report.
    """
    class_report = classification_report(y_true, y_pred)
    print(f"Classification Report for {title}:")
    print(class_report)
```

In [36]:

```
def plot_roc_curve(y_true, y_prob, title):
    """
    Plots the ROC curve and calculates AUC.

    Parameters:
    y_true (array-like): True labels.
    y_prob (array-like): Predicted probabilities.
    title (str): Title for the plot.
    """
    fpr, tpr, _ = roc_curve(y_true, y_prob)
    roc_auc = roc_auc_score(y_true, y_prob)

    plt.figure(figsize=(8, 6))
    plt.plot(fpr, tpr, color='blue', lw=2, label=f'{title} (AUC = {roc_
    plt.plot([0, 1], [0, 1], color='grey', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'ROC Curve for {title}')
    plt.legend(loc='lower right')
    plt.show()

    print(f'ROC AUC score for {title}: {roc_auc:.2f}')
```

In [37]:

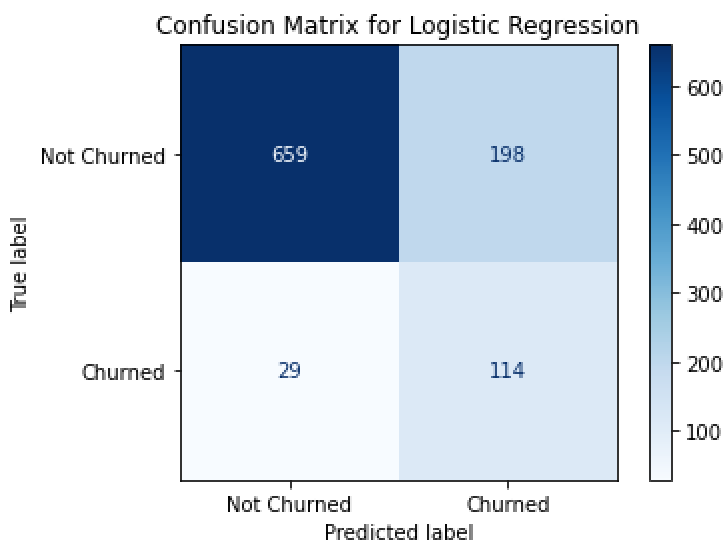
```
def cross_validate_model(model, X, y, cv=5):
    """
    Performs cross-validation and prints AUC scores.

    Parameters:
    model: The model to be evaluated.
    X (array-like): Feature data.
    y (array-like): Target labels.
    cv (int): Number of folds in cross-validation.
    """
    cross_val_scores = cross_val_score(model, X, y, cv=cv, scoring='roc
    print(f"Cross-validated AUC scores: {cross_val_scores}")
    print(f"Mean Cross-validated AUC score: {cross_val_scores.mean()}")
```

In [38]:
```
def print_accuracies(model, X_train, y_train, X_test, y_test, title):
    """
    Prints the training and test accuracies.

    Parameters:
    model: The trained model.
    X_train (array-like): Training feature data.
    y_train (array-like): Training target labels.
    X_test (array-like): Test feature data.
    y_test (array-like): Test target labels.
    title (str): Title for the accuracy report.
    """
    train_accuracy = accuracy_score(y_train, model.predict(X_train))
    test_accuracy = accuracy_score(y_test, model.predict(X_test))

    print(f"{title} Training Accuracy: {train_accuracy:.2f}")
    print(f"{title} Test Accuracy: {test_accuracy:.2f}")
```

5.1 Baseline Logistic Regression Model

In [39]:
```
# Initialize the model
log_reg = LogisticRegression(random_state=42)

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

# Predict on the test set
y_pred_log_reg = log_reg.predict(X_test_processed)
y_prob_log_reg = log_reg.predict_proba(X_test_processed)[:, 1]

plot_confusion_matrix(y_test, y_pred_log_reg, labels=['Not Churned', 'C
```

```
Confusion Matrix for Logistic Regression:
[[659 198]
 [ 29 114]]
```



Confusion Matrix:

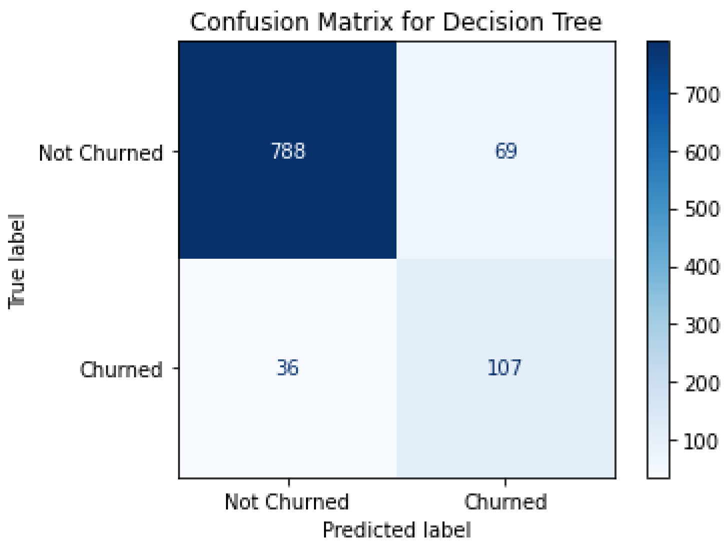
- **True Negatives (TN)**: 659
- **False Positives (FP)**: 198
- **False Negatives (FN)**: 29
- **True Positives (TP)**: 114

5.2 Baseline Decision Tree Model

In [40]:

```
# Initialize the model
decision_tree = DecisionTreeClassifier(random_state=42)

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

# Predict on the test set
y_pred_decision_tree = decision_tree.predict(X_test_processed)
y_prob_decision_tree = decision_tree.predict_proba(X_test_processed)[:,

plot_confusion_matrix(y_test, y_pred_decision_tree, labels=['Not Churne
```

```
Confusion Matrix for Decision Tree:
[[788  69]
 [ 36 107]]
```



Confusion Matrix:

- **True Negatives (TN)**: 788
- **False Positives (FP)**: 69
- **False Negatives (FN)**: 36
- **True Positives (TP)**: 107

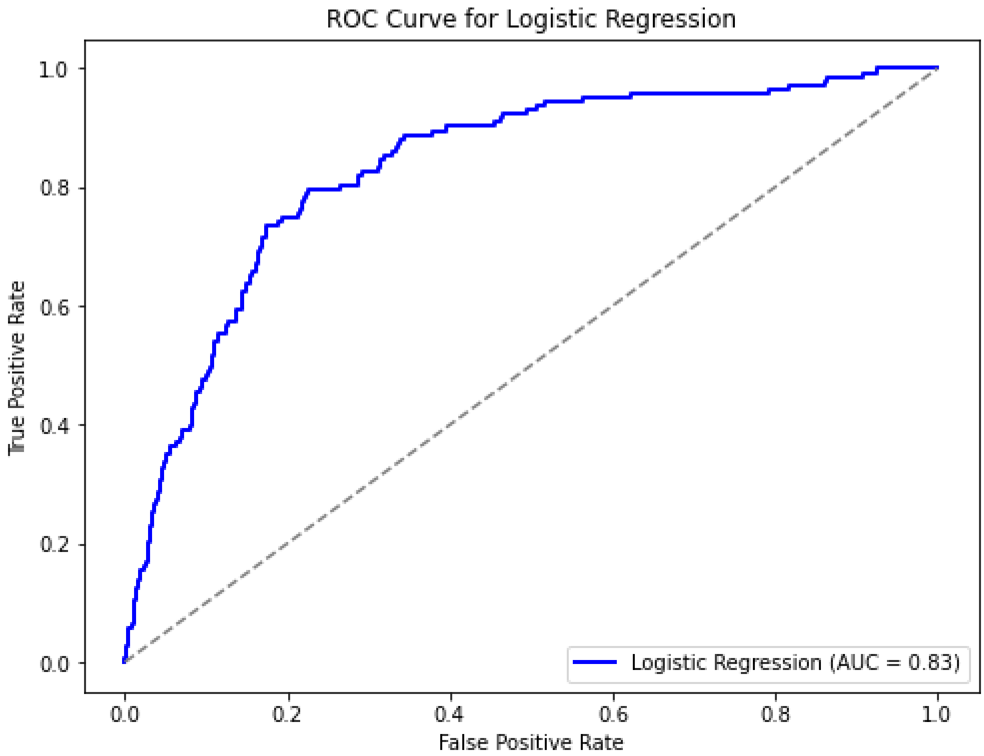
5.3 Evaluating the Baseline Models

5.3.1 Evaluating Baseline Logistic Regression Model

In [41]:

```
# Evaluate Logistic Regression
print_classification_report(y_test, y_pred_log_reg, title='Logistic Reg
plot_roc_curve(y_test, y_prob_log_reg, title='Logistic Regression')
cross_validate_model(log_reg, X_train_processed, y_train)
print_accuracies(log_reg, X_train_resampled, y_train_resampled, X_test_
```

```
Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.96      0.77      0.85       857
           1       0.37      0.80      0.50       143

    accuracy                           0.77      1000
   macro avg       0.66      0.78      0.68      1000
weighted avg       0.87      0.77      0.80      1000
```




```
ROC AUC score for Logistic Regression: 0.83
Cross-validated AUC scores: [0.79802447 0.83676102 0.80583812 0.76237
807 0.80874963]
Mean Cross-validated AUC score: 0.802350263035656
Logistic Regression Training Accuracy: 0.76
Logistic Regression Test Accuracy: 0.77
```

Model Interpretation

Comments on model accuracy: The accuracy of the model is 77%. The training accuracy is 76% whereas the test accuracy is 77%.

Classification Report:

- **Precision:** The precision for class 0 (not churned is) 96% and for the class 1 (churned) is 37%.
- **Recall:** The model correctly identifies 77% of actual class 0 instances and identifies 80% of actual class 1 instances.
- **F1-Score:** The F1-score reflects the balance between precision and recall for class 0. A score of 0.85 is high. The F1-score for class 1 is lower, reflecting a trade-off between precision and recall.
- **Macro Average:** The macro average provides an average performance across all classes, treating each class equally.
- **Weighted Average:** The weighted average accounts for class support, indicating better performance considering class imbalance.

ROC AUC Score:

- The ROC AUC score of 0.83 indicates a good ability to distinguish between classes.

Cross-Validated AUC Scores:

- The cross-validated AUC scores are consistent with the overall ROC AUC score, indicating robust performance.

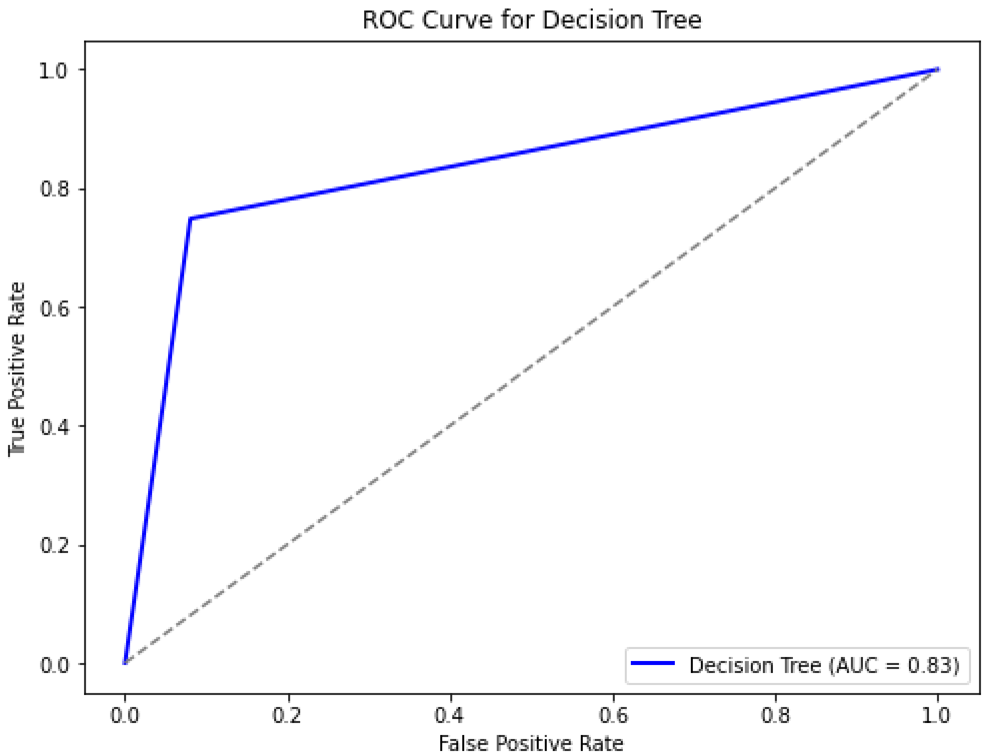
Summary

- **Strengths**: The model effectively distinguishes between classes and performs well in predicting the majority class (class 0).
- **Weaknesses**: The model has lower precision for the minority class (class 1).

5.3.2 Evaluating Baseline Decision Tree Model

In [42]:
```
# Evaluate Decision Tree
print_classification_report(y_test, y_pred_decision_tree, title='Decisi
plot_roc_curve(y_test, y_prob_decision_tree, title='Decision Tree')
cross_validate_model(decision_tree, X_train_processed, y_train)
print_accuracies(decision_tree, X_train_resampled, y_train_resampled, X
```

```
Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.96      0.92      0.94       857
           1       0.61      0.75      0.67       143

    accuracy                           0.90      1000
   macro avg       0.78      0.83      0.80      1000
weighted avg       0.91      0.90      0.90      1000
```




```
ROC AUC score for Decision Tree: 0.83
Cross-validated AUC scores: [0.77624208 0.82019387 0.8327252  0.81902
158 0.83391221]
Mean Cross-validated AUC score: 0.8164189849438852
Decision Tree Training Accuracy: 1.00
Decision Tree Test Accuracy: 0.90
```

Model Interpretation

Comments on model accuracy: The accuracy of the model is 90%. The training accuracy is 100% whereas the test accuracy is 90%. This training accuracy indicates overfitting.

Classification Report:

- **Precision:** The precision for class 0 (not churned is) 96% and for the class 1 (churned) is 61%.
- **Recall:** The model correctly identifies 92% of actual class 0 instances and identifies 75% of actual class 1 instances.
- **F1-Score:** The F1-score reflects the balance between precision and recall for class 0. A score of 0.94 is high. The F1-score for class 1 is lower, reflecting a trade-off between precision and recall.
- **Macro Average:** The macro average provides an average performance across all classes, treating each class equally.
- **Weighted Average:** The weighted average accounts for class support, indicating better performance considering class imbalance.

ROC AUC Score:

- The ROC AUC score of 0.83 indicates a good ability to distinguish between classes.

Cross-Validated AUC Scores:

- The cross-validated AUC scores are consistent with the overall ROC AUC score, indicating robust performance.

Summary

- **Strengths**: The model effectively distinguishes between classes and performs well in predicting the majority class (class 0).
- **Weaknesses**: The model has lower precision for the minority class (class 1). The model has a high accuracy and performs well on the test, however the high training accuracy suggests overfitting where the model performs perfectly on the training data but slightly less on unseen data.

5.4 Comparision of the Two Baseline Models

Here's a comparative analysis of the Decision Tree and Logistic Regression models:

1. Confusion Matrix Comparison:

- The Decision Tree model has more True Negatives and fewer False Positives compared to Logistic Regression, indicating it better identifies non-churn customers.
- Logistic Regression has slightly more True Positives and fewer False Negatives, suggesting it captures more churn cases.

2. Classification Report Comparison:

- **Class 0 (Non-Churn):** Both models have similar precision, but the Decision Tree has a higher recall and F1-score, making it better at predicting non-churn customers.
- **Class 1 (Churn):** Logistic Regression has a higher recall but lower precision, indicating that while it identifies more churn cases, it also has more false positives. The F1-score for the Decision Tree is higher, indicating a better balance between precision and recall.
- **Overall Accuracy:** The Decision Tree outperforms Logistic Regression with a higher overall accuracy.

3. ROC AUC Score Comparison:

- Both models have an identical ROC AUC score of 0.83, indicating similar overall discriminatory power.
- The Decision Tree has a slightly higher mean cross-validated AUC score, indicating more consistent performance across folds.

4. Model Accuracy Comparison:

- The Decision Tree shows a perfect training accuracy, indicating overfitting, while Logistic Regression has a more balanced training accuracy.
- The Decision Tree also has a significantly higher test accuracy, making it the more accurate model on unseen data.

Overall Conclusion:

- **Decision Tree:** While it shows signs of overfitting, it performs better overall, with higher accuracy, better precision for the majority class, and a balanced performance across both classes.
- **Logistic Regression:** It is less prone to overfitting but underperforms in accuracy and F1-score compared to the Decision Tree. However, it captures more churn cases, which could be beneficial if identifying churn is a priority.

For our baseline model, it would be benefitial for us to select the logistic regression model than the decision tree model, with the above in mind.

We now proceed to iterating our models by hypertuning the parameters to see if we can address some of the problems above.

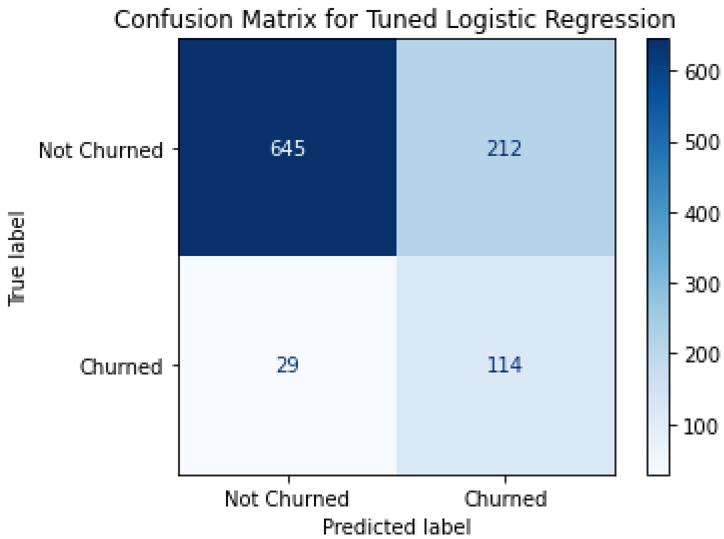
5.5 Logistic Regression with Hyperparameter Tuning

We'll use `GridSearchCV` to find the best parameters for the Logistic Regression model.

In [43]:

```
# Define the parameter grid
log_reg_param_grid = [
    {'penalty': ['l1'], 'C': [0.01, 0.1, 1, 10], 'solver': ['liblinear'
    {'penalty': ['l2'], 'C': [0.01, 0.1, 1, 10], 'solver': ['lbfgs', 'l
    {'penalty': ['elasticnet'], 'C': [0.01, 0.1, 1, 10], 'solver': ['sa
]
# Initialize the GridSearchCV
grid_search_log_reg = GridSearchCV(LogisticRegression(random_state=42),

# Fit the model
grid_search_log_reg.fit(X_train_resampled, y_train_resampled)

# Best parameters
print(f"Best parameters for Logistic Regression: {grid_search_log_reg.b

# Evaluate the tuned model
best_log_reg = grid_search_log_reg.best_estimator_
y_pred_log_reg_tuned = best_log_reg.predict(X_test_processed)
y_prob_log_reg_tuned = best_log_reg.predict_proba(X_test_processed)[:,

plot_confusion_matrix(y_test, y_pred_log_reg_tuned, labels=['Not Churne
```

```
Best parameters for Logistic Regression: {'C': 0.01, 'max_iter': 100,
'penalty': 'l2', 'solver': 'liblinear'}
Confusion Matrix for Tuned Logistic Regression:
[[645 212]
 [ 29 114]]
```



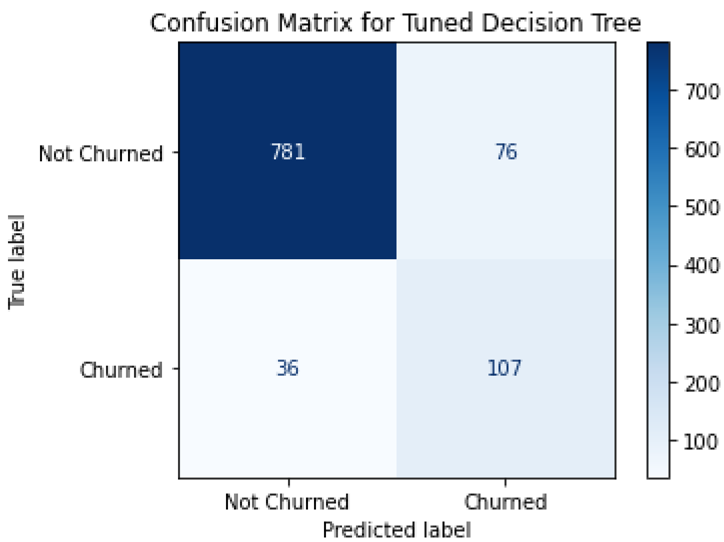
Confusion Matrix:

- **True Negatives (TN)**: 645
- **False Positives (FP)**: 212
- **False Negatives (FN)**: 29
- **True Positives (TP)**: 114

5.6 Decision Tree with Hyperparameter Tuning

In [44]:
```
# Define the parameter grid
decision_tree_param_grid = {
    'criterion': ['gini', 'entropy'],
    'splitter': ['best', 'random'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'auto', 'sqrt', 'log2']
}

# Initialize the GridSearchCV
grid_search_decision_tree = GridSearchCV(DecisionTreeClassifier(random_

# Fit the model
grid_search_decision_tree.fit(X_train_resampled, y_train_resampled)

# Best parameters
print(f"Best parameters for Decision Tree: {grid_search_decision_tree.b

# Evaluate the tuned model
best_decision_tree = grid_search_decision_tree.best_estimator_
y_pred_decision_tree_tuned = best_decision_tree.predict(X_test_processe
y_prob_decision_tree_tuned = best_decision_tree.predict_proba(X_test_pr

plot_confusion_matrix(y_test, y_pred_decision_tree_tuned, labels=['Not
```

```
Best parameters for Decision Tree: {'criterion': 'entropy', 'max_dept
h': 20, 'max_features': None, 'min_samples_leaf': 4, 'min_samples_spl
it': 2, 'splitter': 'random'}
Confusion Matrix for Tuned Decision Tree:
[[781  76]
 [ 36 107]]
```



Confusion Matrix:

- **True Negatives (TN)**: 781
- **False Positives (FP)**: 76
- **False Negatives (FN)**: 36

- **True Positives (TP)**: 107

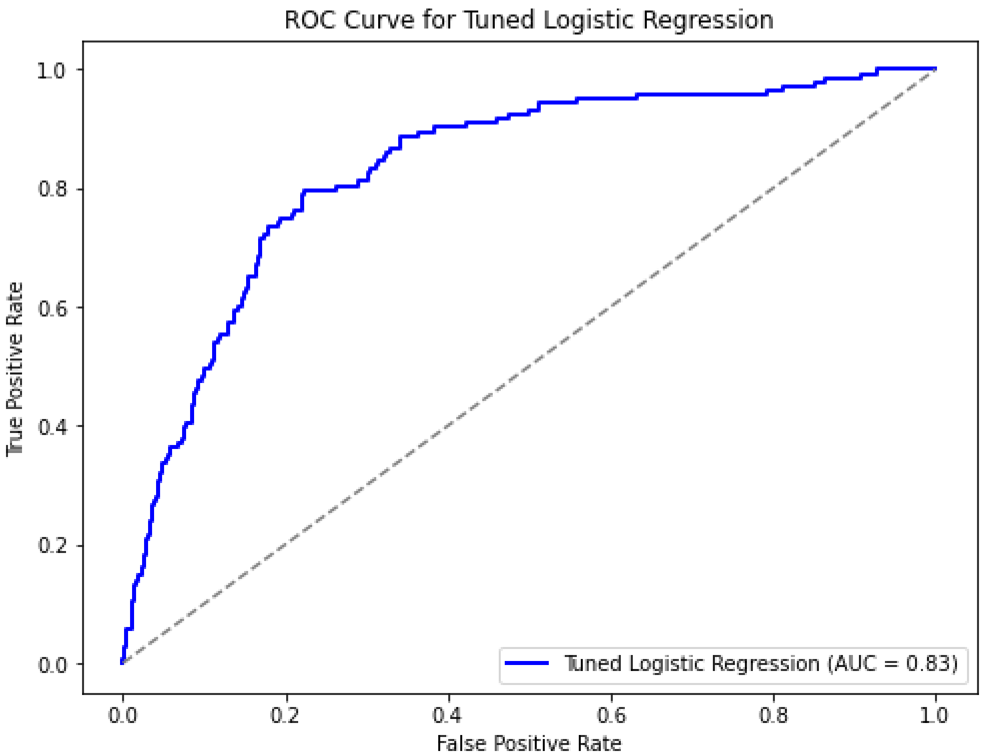
5.7 Evaluating Models with Hyperparameter Tuning

5.7.1 Evaluating Logistic Regression with Hyperparameter Tuning

We'll use `GridSearchCV` to find the best parameters for the Logistic Regression model.

In [45]:
```
# Evaluate Logistic Regression after tuning
print_classification_report(y_test, y_pred_log_reg_tuned, title='Tuned
plot_roc_curve(y_test, y_prob_log_reg_tuned, title='Tuned Logistic Regr
cross_validate_model(best_log_reg, X_train_processed, y_train)
print_accuracies(best_log_reg, X_train_resampled, y_train_resampled, X_
```

```
Classification Report for Tuned Logistic Regression:
              precision    recall  f1-score   support

           0       0.96      0.75      0.84       857
           1       0.35      0.80      0.49       143

    accuracy                           0.76      1000
   macro avg       0.65      0.77      0.66      1000
weighted avg       0.87      0.76      0.79      1000
```




```
ROC AUC score for Tuned Logistic Regression: 0.83
Cross-validated AUC scores: [0.80189444 0.84280554 0.80661212 0.77468
223 0.81488324]
Mean Cross-validated AUC score: 0.8081755156439726
Tuned Logistic Regression Training Accuracy: 0.77
Tuned Logistic Regression Test Accuracy: 0.76
```

Model Interpretation

Comments on model accuracy: The model's accuracy is 76%. The training accuracy is 77%, and the test accuracy is 76%. This indicates that the model generalizes well to unseen data, showing consistent performance across training and test datasets.

Classification Report:

- **Precision:** The precision for class 0 (not churned) is 96%, indicating that when the model predicts a customer has not churned, it is correct 96% of the time. For class 1 (churned), the precision is 35%, meaning the model is less reliable in predicting actual churn.
- **Recall:** The model correctly identifies 75% of actual class 0 instances and 80% of actual class 1 instances. The high recall for class 1 shows that the model is effective at identifying customers who are likely to churn.
- **F1-Score:** The F1-score for class 0 is 0.84, indicating a strong balance between precision and recall. The F1-score for class 1 is 0.49, reflecting a trade-off between precision and recall due to the class imbalance.
- **Macro Average:** The macro average F1-score is 0.66, providing an average performance across both classes, treating each class equally.
- **Weighted Average:** TThe weighted average F1-score is 0.79, which accounts for the class imbalance, indicating overall good performance with higher emphasis on the majority class.

ROC AUC Score:

- The ROC AUC score of 0.83 indicates the model has a good ability to distinguish between customers who will churn and those who will not.

Cross-Validated AUC Scores:

- The cross-validated AUC scores are consistent with the overall ROC AUC score, with a mean of 0.81, indicating that the model's performance is robust across different subsets of the data.

Summary

- **Strengths**: TThe model effectively distinguishes between customers who are likely to churn and those who are not, particularly excelling in identifying non-churners (class 0).
- **Weaknesses**: The model has lower precision for the minority class (class 1), indicating it is less reliable at predicting actual churn, which may lead to more false positives in real-world applications.

Comparison with the Baseline Model: Despite the difference in accuracy with the baseline model, our tuned model does not perform comparatively better to our baseline model. Precision for the churned class is reduced but results in an 0.01% increase in the F1-score. In this case we would go with our baseline model.

Let's see how the hypertuned decision tree model performs.

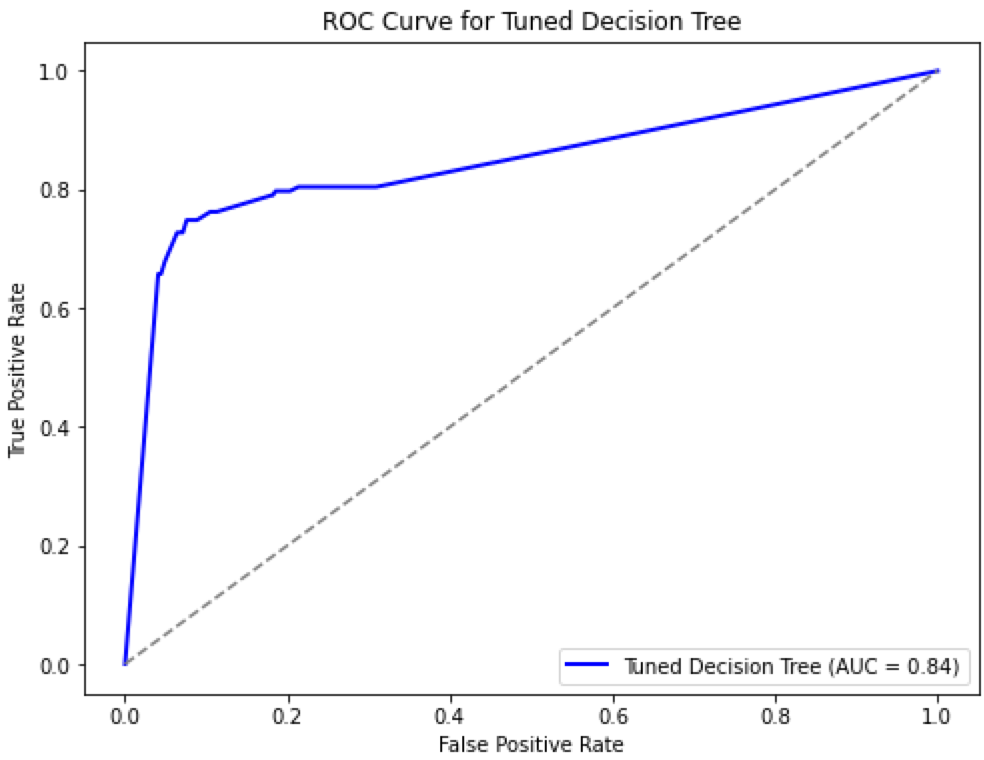
5.7.2 Evaluating Decision Trees with Hyperparameter Tuning

We'll use `GridSearchCV` to find the best parameters for the Decision Tree model.

In [46]:

```
# Evaluate Decision Tree after tuning
print_classification_report(y_test, y_pred_decision_tree_tuned, title='
plot_roc_curve(y_test, y_prob_decision_tree_tuned, title='Tuned Decisic
cross_validate_model(best_decision_tree, X_train_processed, y_train)
print_accuracies(best_decision_tree, X_train_resampled, y_train_resampl
```

```
Classification Report for Tuned Decision Tree:
              precision    recall  f1-score   support

           0       0.96      0.91      0.93       857
           1       0.58      0.75      0.66       143

    accuracy                           0.89      1000
   macro avg       0.77      0.83      0.79      1000
weighted avg       0.90      0.89      0.89      1000
```




```
ROC AUC score for Tuned Decision Tree: 0.84
Cross-validated AUC scores: [0.77075041 0.84492481 0.88528306 0.82724
283 0.87640408]
Mean Cross-validated AUC score: 0.8409210378147556
Tuned Decision Tree Training Accuracy: 0.95
Tuned Decision Tree Test Accuracy: 0.89
```

Model Interpretation

Comments on model accuracy: The accuracy of the model is 89%. The training accuracy is 95% whereas the test accuracy is 89%.

Classification Report:

- **Precision:** The precision for class 0 (not churned is) 96% and for the class 1 (churned) is 58%.
- **Recall:** The model correctly identifies 91% of actual class 0 instances and identifies 75% of actual class 1 instances.
- **F1-Score:** The F1-score reflects the balance between precision and recall for class 0. A score of 0.93 is high. The F1-score for class 1 is 0.66, reflecting a trade-off between precision and recall.
- **Macro Average:** The macro average provides an average performance across all classes, treating each class equally.
- **Weighted Average:** The weighted average accounts for class support, indicating better performance considering class imbalance.

ROC AUC Score:

- The ROC AUC score of 0.84 indicates a good ability to distinguish between classes.

Cross-Validated AUC Scores:

- The cross-validated AUC scores are consistent with the overall ROC AUC score, indicating robust performance.

Summary

- **Strengths**: The model effectively distinguishes between classes and performs well in predicting the majority class (class 0).
- **Weaknesses**: The model has lower precision for the minority class (class 1).

Comparison with the Baseline Model:

- The baseline model has more true negatives and fewer false positives compared to the tuned model. The tuned model has slight increase in false positives, but performance metrics are generally similar.
- Precision for class 1 is lower in the tuned model (58% vs. 61% in the baseline).
- Recall values are consistent across models.
- The tuned model has slightly lower F1-scores for both classes, reflecting the precision-recall trade-offs.
- The tuned model has a slightly better ROC AUC score of 0.84 indicating improved performance in distinguishing between classes.
- The tuned model has a mean 0f of 0.84 cross validated AUC scores better than the baseline of 0.82 showing an improvement in generalization.
- **The baseline model had a Perfect training accuracy (100%) with a test accuracy of 90%.** The tuned model has a slightly lower accuracy of (95%) but consistent test accuracy (89%). **This shows that the hypertuned parameters have addressed overfitting in the model.**

5.8 Feature Importance in Tuned Decision Tree Model

In [47]:

```
# Ensure feature_names is a list of feature names
feature_names = X_train.columns

# Assuming feature_importances_ is the array from your model
feature_importances = best_decision_tree.feature_importances_

# Create a DataFrame for better visualization
feature_importances_df = pd.DataFrame({
    'Feature': feature_names,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

print(feature_importances_df)
```

```
                   Feature  Importance
11  customer_service_calls    0.239547
4         total_day_charge    0.224156
1       international_plan    0.120539
9         total_intl_calls    0.085303
6         total_eve_charge    0.083731
2          voice_mail_plan    0.055234
10       total_intl_charge    0.047618
8       total_night_charge    0.036119
0           account_length    0.032591
7        total_night_calls    0.029564
3          total_day_calls    0.023570
5          total_eve_calls    0.022026
```

6.0 Conclusion and Recommendations

6.1 Conclusions

1. Baseline Models Performance:

- **Logistic Regression:**
  - **Accuracy:** 76%
  - **ROC AUC Score:** 0.83
  - **Strengths:** Good at distinguishing non-churners; strong performance in predicting class 0.
  - **Weaknesses:** Lower precision for churned class (class 1), indicating more false positives.
- **Decision Tree:**
  - **Accuracy:** 90%
  - **ROC AUC Score:** 0.83
  - **Strengths:** Excellent performance on majority class (class 0); higher accuracy and precision compared to Logistic Regression.
  - **Weaknesses:** Overfitting indicated by high training accuracy (100%) and slightly lower precision for churned class (class 1).

2. Hyperparameter Tuning Results:

- **Tuned Logistic Regression:**
  - **Accuracy:** 76%
  - **ROC AUC Score:** 0.83
  - **Strengths:** Consistent performance with baseline model; better generalization with similar performance across training and test sets.
  - **Weaknesses:** Similar issues with precision for class 1; no substantial improvement over the baseline model.
- **Tuned Decision Tree:**
  - **Accuracy:** 89%
  - **ROC AUC Score:** 0.84
  - **Strengths:** Improved ROC AUC score and generalization compared to baseline; reduced overfitting with better performance metrics.
  - **Weaknesses:** Slightly lower precision for class 1 compared to baseline, though still effective.

6.2 Recommendations

Based on the analysis and modelling above, we recommend the following to SyriaTel's management:

1. Given the comparison, SyriaTel should adopt the **Tuned Decision Tree** model as it is preferable due to its improved ROC AUC score, better generalization, and consistent performance across folds. While it exhibits slight issues with precision for class 1, it overall provides a more balanced and accurate prediction compared to the Logistic Regression model.
2. When it comes to predicting whether a customer will churn, SyriaTel needs to focus on the number **customer calls**, **total day charge**, whether or not a customer is subscribed to an **international plan**, **total international calls** and finally **total evening charge**.
3. Implement loyalty programs or offer special incentives for long-tenured customers to reward their loyalty as longer tenured customers are less likely to churn. This could include discounts, personalized offers, or exclusive benefits that encourage them to remain with SyriaTel.
4. Consider training customer service representatives to resolve issues more effectively on the first contact when possible. Additionally, consider offering personalized assistance to customers to improve their overall experience.
5. Offer flexible pricing options for day packages and evening packages since customers who are charged more tend to churn.
6. Increase marketing efforts to highlight the benefits of the international plan, particularly to customers who frequently make international calls or might benefit from such a plan.
7. Investigate the specific area codes with slightly higher churn rates to determine if there are underlying factors (e.g., service quality, local competition) contributing to these variations.

7.0 Next Steps

In order to improve our overall results, we can do the following:

- Investigate additional feature engineering to potentially improve model performance, such as creating new features or performing feature selection.
- Consider using ensemble methods like Random Forests or Gradient Boosting to combine the strengths of multiple models and improve predictive performance.
- Explore techniques to enhance the precision of the minority class, such as adjusting class weights or using ensemble methods like Random Forests.
- To identify other patterns that may affect churn, consider employing the KNN model to segment customers based on their behaviours to generate further insights into which clusters of customers tend to churn and why.