PROBLEM DEFINITION

SyriaTel is a telecommunications company with at least more than 3000 subscribers. The company offers a variety of services which include normal local calls, international calls and voicemail. However, the market conditions seem to make blows to the company quite frequently with a noted customer churn. This poses a threat to SyriaTel as it would mean low turnover and ultimate business decline.

In this regard, SyriaTel have shared their customer dataset that would help in understanding the different patterns portrayed. Further, the company is interested in reducing how much money is lost because of customers who don't stick around very long.

This project uses binary classification to create and predict models that help define the patterns and suggest a resolution in how money lost can be reduced in SyriaTel company.

Stakeholder: SyriaTel

Objectives of the project:

1. Identify Key Factors Influencing Customer Churn: Determine the most significant factors that contribute to customer churn.
2. Build a Predictive Model for Customer Churn: Develop and validate a machine learning model to predict whether a customer will churn.
3. Develop Customer Retention Strategies: Formulate actionable strategies to retain customers identified as high risk for churn.

Conclusions from the study were drawn as follows:

- It is noted that customers who have higher usage during the day ("total day minutes" and "total day charge") and those who frequently contact customer service ("customer service calls") are more likely to churn. This suggests that dissatisfaction with service quality or billing issues during peak hours may drive churn.
- According to the feature importances, Total day minutes, Total day charge, Customer service calls, International plan,Total eve charge are the top contributing factors to customer churning or not.
- Based on the identified key factors influencing customer churn, actionable strategies can be formulated to retain customers identified as high risk for churn.

BUSINESS UNDERSTANDING

For this project, I chose the "SyriaTel Customer Churn" dataset. The dataset provides various customer-related information such as 'state', 'account length', 'area code', 'phone number', 'international plan', 'voice mail plan', 'number vmail messages', and several other features related to call duration, charges, and customer service interactions. This suggests that the dataset covers a wide range of customer attributes.

This dataset is particularly suitable for the objectives, as it provides the necessary information to understand customer behavior and predict churn.

SyriaTel Customer Churn" dataset has 3333 rows and 21 columns. The dataset contains data including: state: The state code where the customer resides.

- account length: The number of days the account has been active.
- area code: The area code of the customer’s phone number.
- phone number: The customer’s phone number.
- international plan: Whether the customer has an international plan.
- voice mail plan: Whether the customer has a voice mail plan.
- number vmail messages: Number of voice mail messages.
- total day minutes, total day calls, total day charge: Usage metrics during the day.
- total eve minutes, total eve calls, total eve charge: Usage metrics during the evening.
- total night minutes, total night calls, total night charge: Usage metrics during the night.
- total intl minutes, total intl calls, total intl charge: International usage metrics.
- customer service calls: Number of calls to customer service.
- churn: Whether the customer has churned or not (target variable).

In [2]:
```
# Importing relevant libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [50]:
```
# Loading the dataset
df = pd.read_csv("Churn in Telecom's dataset.csv")
```

EXPLORATORY DATA ANALYSIS

In [51]:
```
# Reading the first rows and columns to understand the data
df.head()
```

Out[51]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tot ev cal
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	...	1(
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	1'
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	...	1:

5 rows × 21 columns

In [52]:
```
#Determining the rows and columns to understand the data
df.tail()
```

Out[52]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...

5 rows × 21 columns

In [53]:
```
# Examining the dataset's shape
df.shape
```

Out[53]:
```
(3333, 21)
```

In [54]:
```
# Examining the columns
df.columns
```

Out[54]:
```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

In [55]:

```
# Examining the data types
df.dtypes
```

Out[55]:

```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

In [56]:

```
# Checking the dataset's info
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
```

Descriptive Analysis

In [57]:
```
df.describe()
```

Out[57]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total e' minut
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0000(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.9803
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.7138
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0000(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6000(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4000(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3000(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7000(

DATA CLEANING, UNDERSTANDING & PREPARATION

We now check if there are any duplicates

In [58]:
```
# We now check if there are any duplicates
duplicate_count = df.duplicated().sum()
duplicate_count
```

Out[58]: 0

No duplicates found

Now we determine if there are null values in the dataset

In [59]:
```
# Now we determine if there are null values in the dataset
df.isna().sum()
```

Out[59]:
```
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 has no missing value

Proceed to drop irrelevant columns, I dropped phone number since it was a unique identifier without any use for the analysis

In [60]:
```
# Dropping irrelevant columns
df = df.drop(columns=['phone number'])
df.head()
```

Out[60]:

unt gth	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
128	415	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7
107	415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4
137	415	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6
84	408	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9
75	415	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9

We inspect the columns to see if the above code has worked.

In [61]:
```
df.columns
```

Out[61]:
```
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')
```

Finally, I converted the area code to object since it wouldn't serve well if calculated as it is a geaograhical aspect

In [69]:
```
# converting int64 to object
df['area code'] = df['area code'].astype(object)
df.info()
```

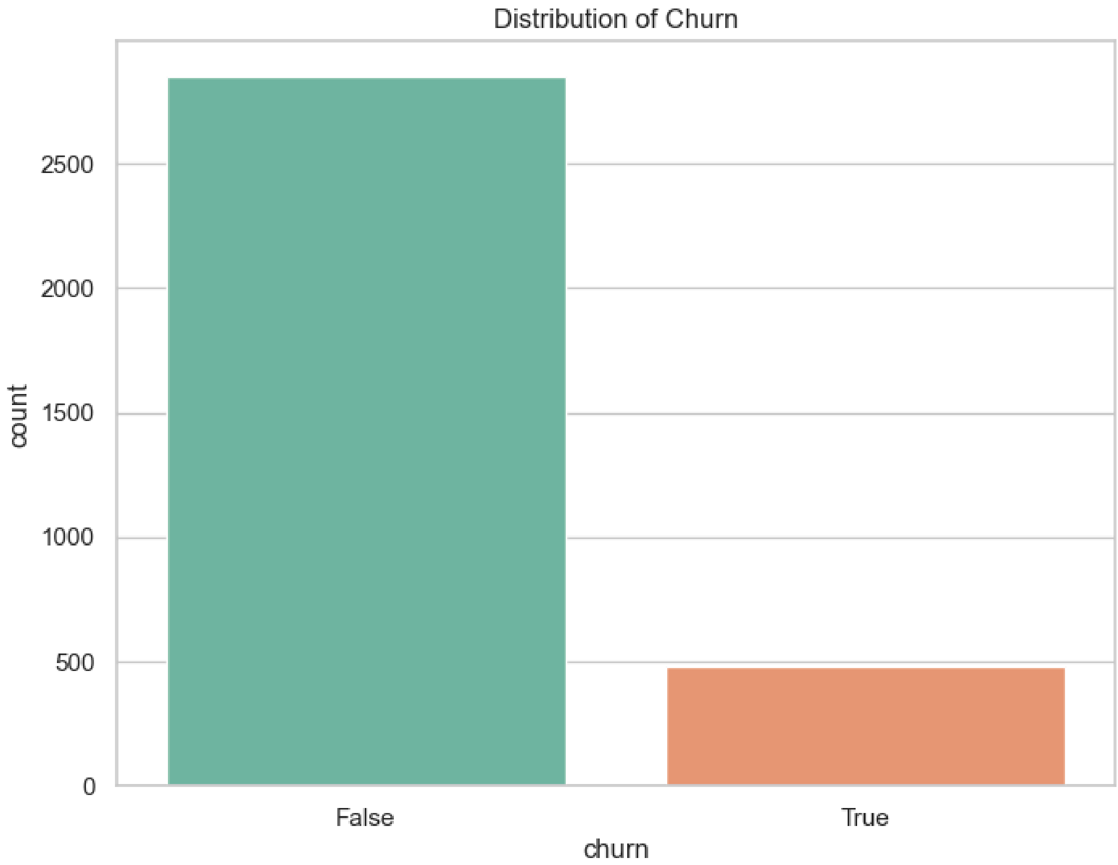
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   object
 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), float64(8), int64(7), object(4)
memory usage: 498.1+ KB
```

DATA VISUALIZATION AND EXPLORATION

I proceeded to visualize the dataset from different angles of univariate (singly), bivariate(one element vs target variable) and multivariate (3 or more elements vs the target variable: churn) analyses, as follows.

Univariate analysis

In [13]:
```
# Checking the target variable distribution: churn
sns.set(style="whitegrid")
plt.figure(figsize=(8, 6))
sns.countplot(df, x='churn', palette='Set2')
plt.title('Distribution of Churn')
plt.show()
```

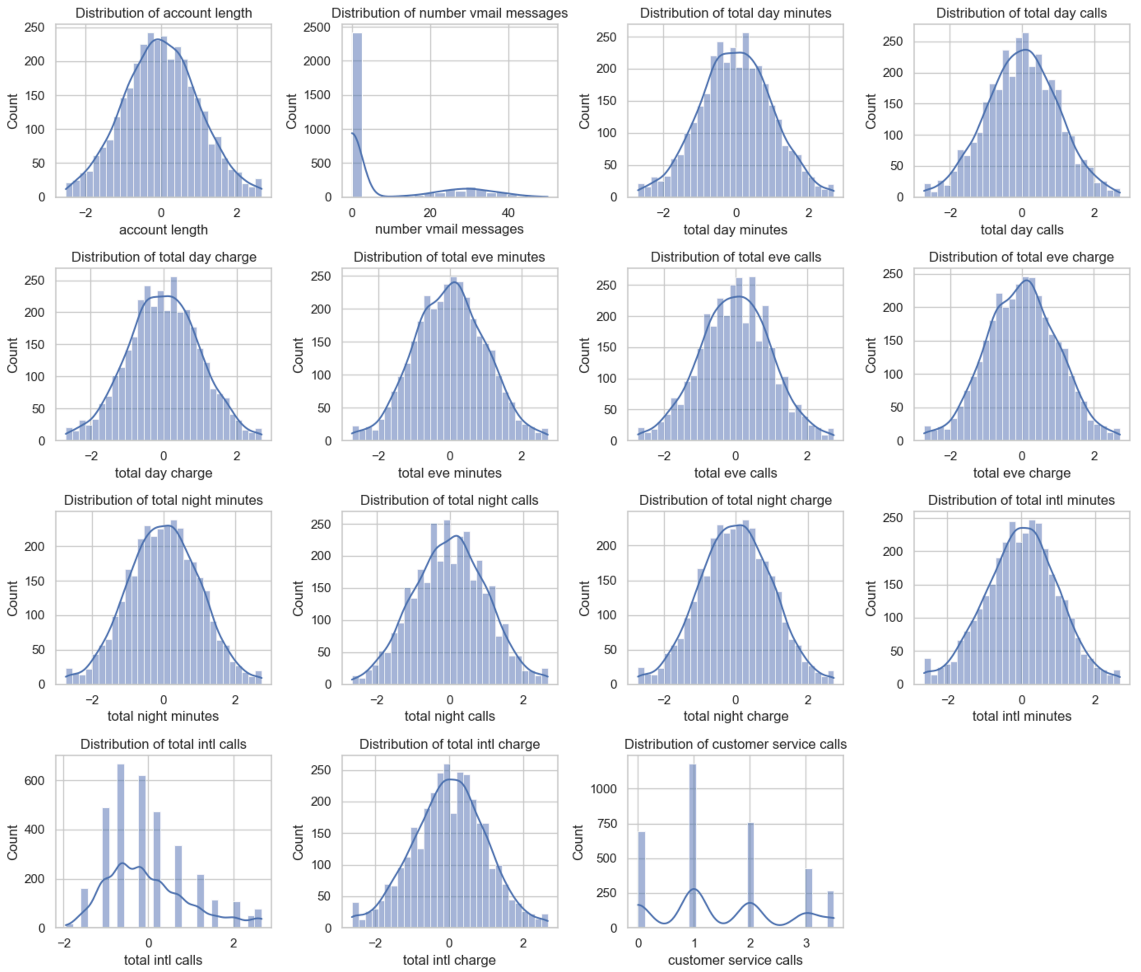


Observation:

According to the churn distribution among the subscribers, about 500 have exited SyriaTel which is a worry to the company.

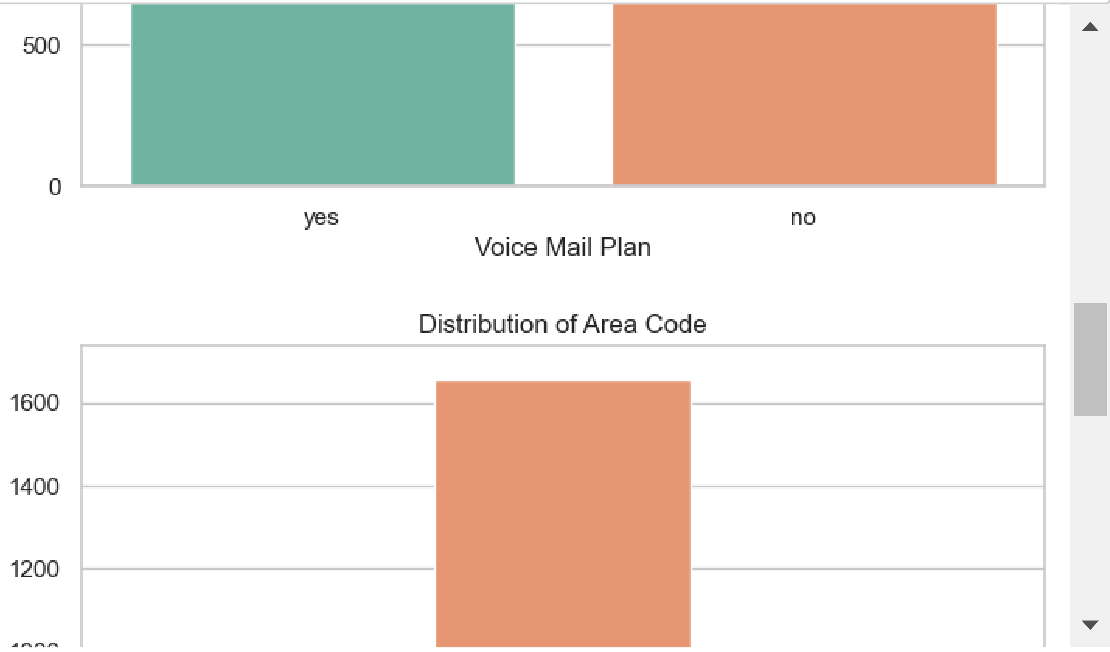
In [176]:

```
# Checking numerical features distribution

numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns

# Set the figure size
plt.figure(figsize=(14, 12))

# Loop through each numerical column and create a subplot for its distribution
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(4, 4, i)
    sns.histplot(df[col], kde=True)
    plt.title(f'Distribution of {col}')

# Adjust the layout to prevent overlap
plt.tight_layout()

# Show the plot
plt.show()
```



In [71]:

```
# Checking the category features distribution by visualization

sns.set(style="whitegrid")

# Visualize the distribution of 'International Plan'
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='international plan', palette='Set2')
plt.title('Distribution of International Plan')
plt.xlabel('International Plan')
plt.ylabel('Count')
plt.show()

# Visualize the distribution of 'Voice Mail Plan'
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='voice mail plan', palette='Set2')
plt.title('Distribution of Voice Mail Plan')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Count')
plt.show()

# Visualize the distribution of 'Area Code'
plt.figure(figsize=(8, 6))
sns.countplot(data=df, x='area code', palette='Set2')
plt.title('Distribution of Area Code')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.show()

# Visualize the distribution of 'State'
plt.figure(figsize=(16, 8))
sns.countplot(data=df, x='state', palette='Set3', order=df['state'].value_coun
plt.title('Distribution of State')
plt.xlabel('State')
plt.ylabel('Count')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Observations:

International plan: about 300(9%) subscribers have also subscribed to the international plan.

Voice mail plan: about 900(27%) customers have subscribed to the voice mail plan.

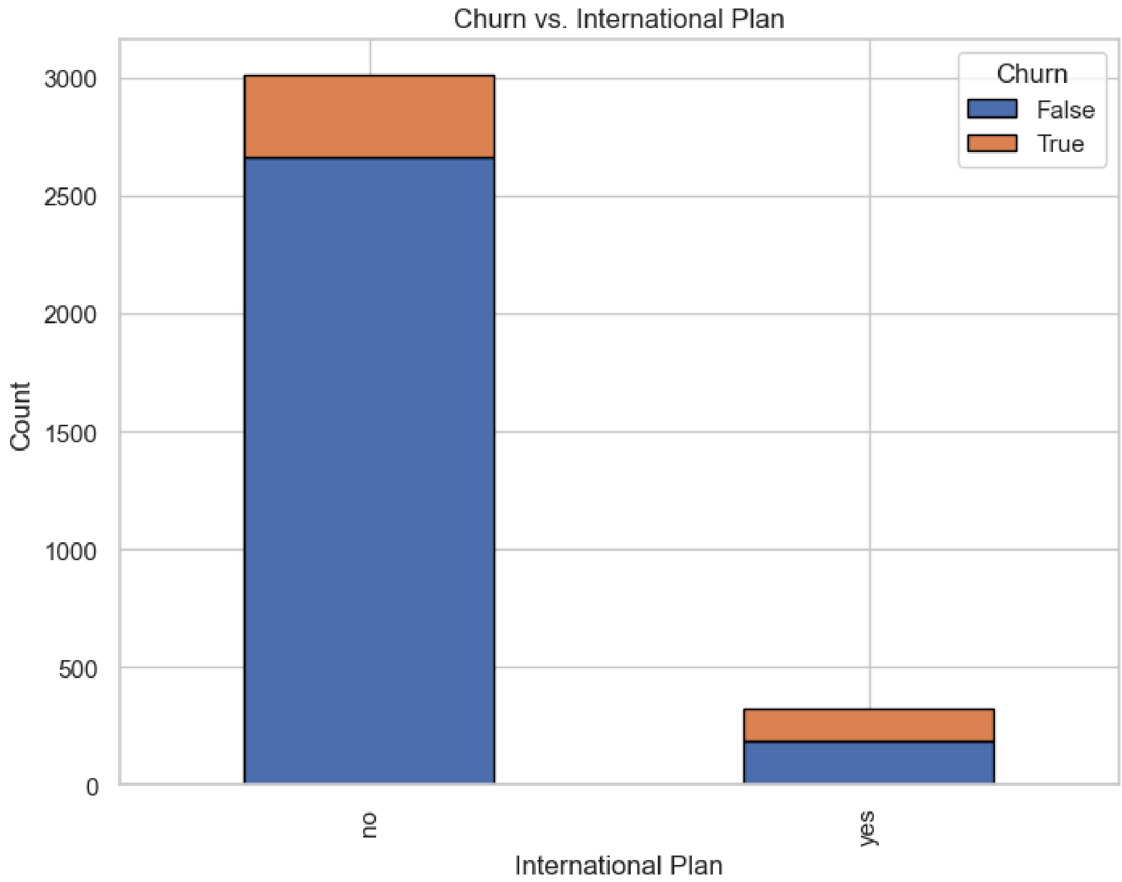
SyriaTel has the highest subscribers from Area code 415, with majority from West Virginia (WV)

Bivariate Analysis

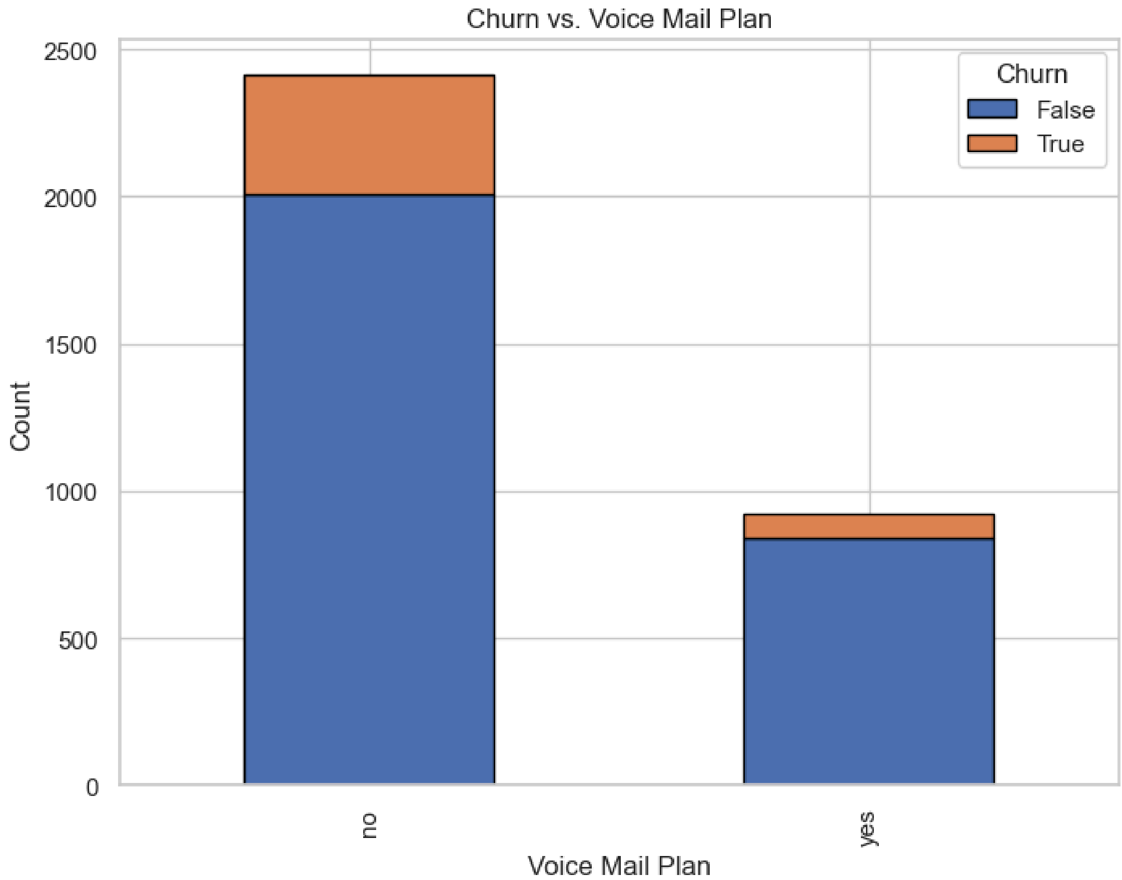
In [72]:

```
# using groupby to visualize churn vs categorical features

# churn vs international plan
plt.figure(figsize=(8, 6))
df.groupby(['international plan', 'churn']).size().unstack().plot(kind='bar',
plt.title('Churn vs. International Plan')
plt.xlabel('International Plan')
plt.ylabel('Count')
plt.legend(title='Churn', loc='upper right')
plt.show()
# The majority of customers who churn do not have an international plan.


# churn vs voice mail plan
plt.figure(figsize=(8, 6))
df.groupby(['voice mail plan', 'churn']).size().unstack().plot(kind='bar', sta
plt.title('Churn vs. Voice Mail Plan')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Count')
plt.legend(title='Churn', loc='upper right')
plt.show()
# Majority of churn seems to occur among customers without a voice mail plan.

# churn vs area code
plt.figure(figsize=(8, 6))
df.groupby(['area code', 'churn']).size().unstack().plot(kind='bar', stacked=T
plt.title('Churn vs. Area Code')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.legend(title='Churn', loc='upper right')
plt.show()
# Churn rates vary across different area codes. While some area codes have hig
# What could be the reason behind this?

# churn vs state
plt.figure(figsize=(16, 8))
df.groupby(['state', 'churn']).size().unstack().plot(kind='bar', stacked=True,
# plt.title('Churn vs. State')
plt.xlabel('State')
plt.ylabel('Count')
plt.legend(title='Churn', loc='upper right')
plt.tight_layout()
plt.show()
# Some states have higher churn rates compared to others.
# Factors such as regional competition, service quality, and marketing strateg
```

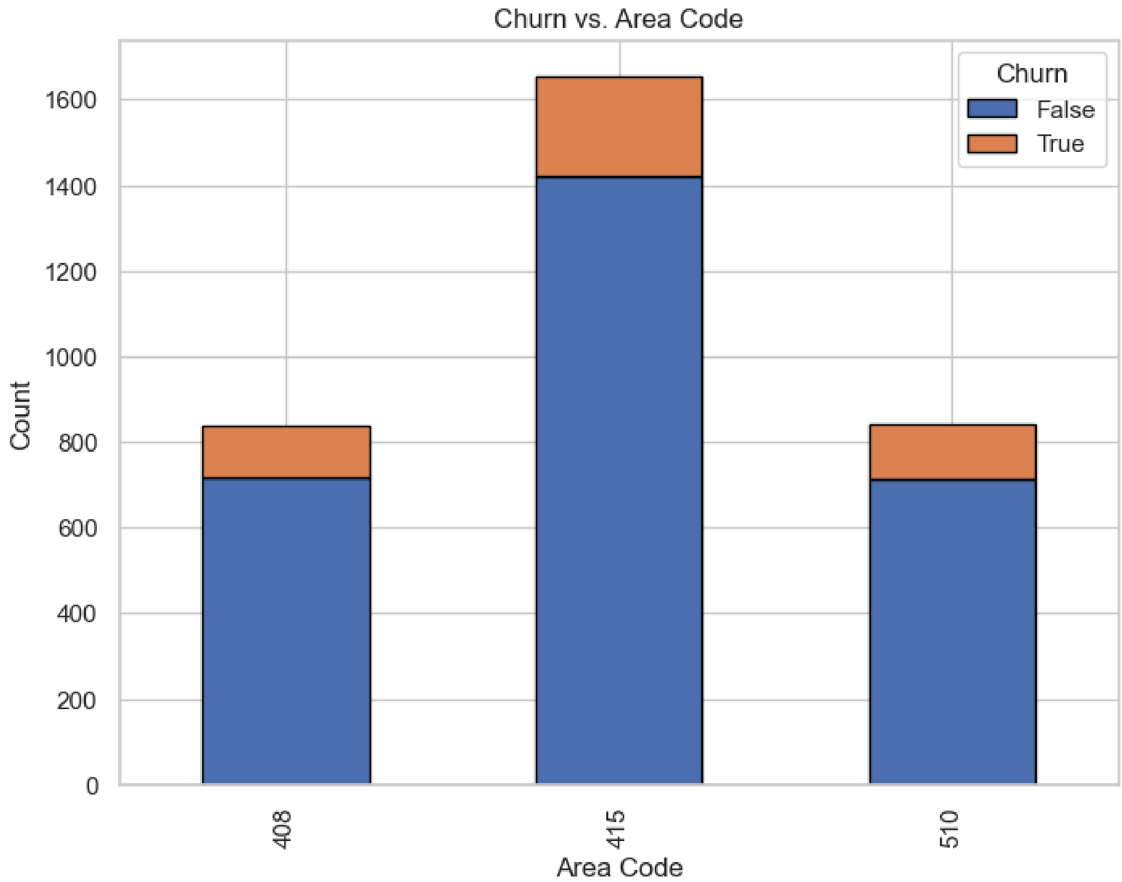
```
<Figure size 800x600 with 0 Axes>
```



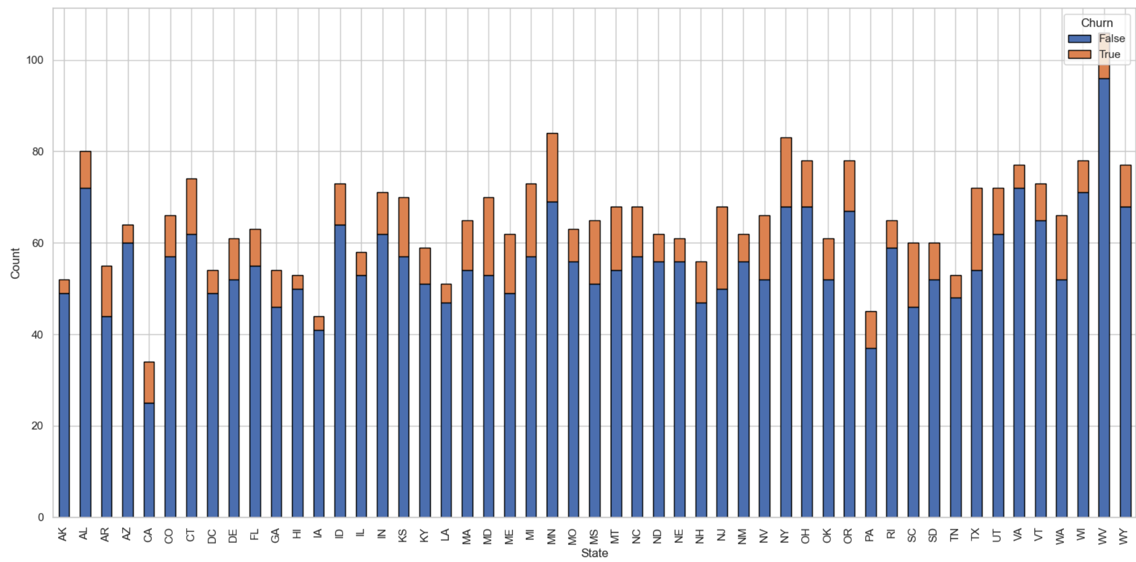

```
<Figure size 800x600 with 0 Axes>
```




```
<Figure size 800x600 with 0 Axes>
```



`<Figure size 1600x800 with 0 Axes>`

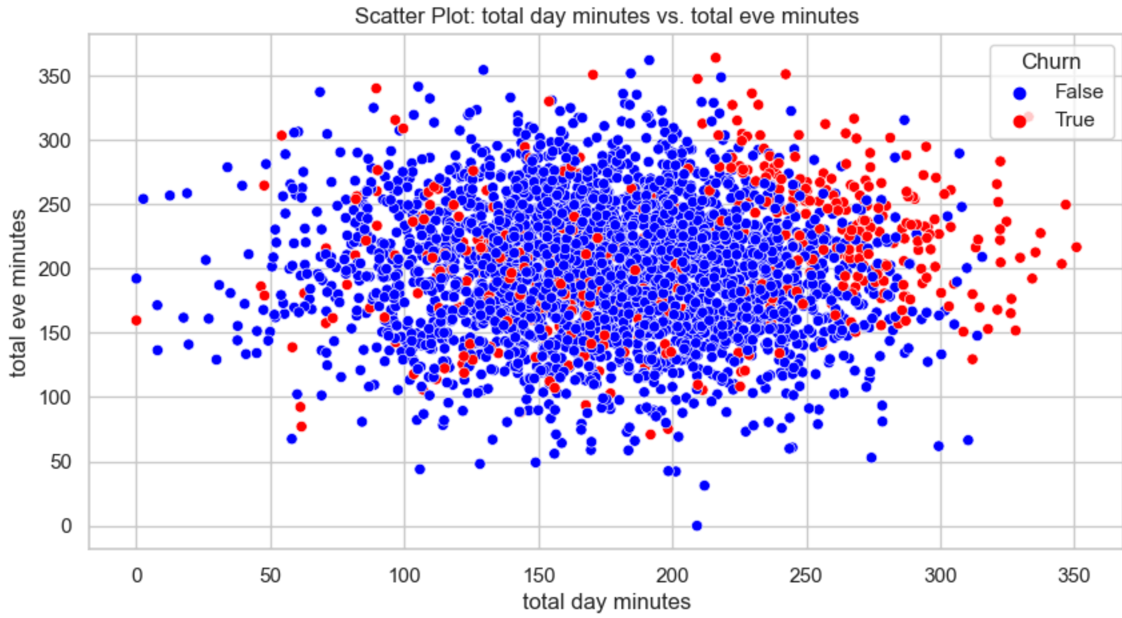


A large number of subscribers without a voice mail plan have churned(exited) SyriaTel.

Area code 415 has the most churn customers.

In [73]:

```
# Select two features for the 2D scatter plot
feature1 = 'total day minutes'
feature2 = 'total eve minutes'

plt.figure(figsize=(10, 5))
sns.scatterplot(data=df, x=feature1, y=feature2, hue='churn', palette={0: 'blu
plt.title(f'Scatter Plot: {feature1} vs. {feature2}')
plt.xlabel(feature1)
plt.ylabel(feature2)
plt.legend(title='Churn', loc='upper right')
plt.show()
```

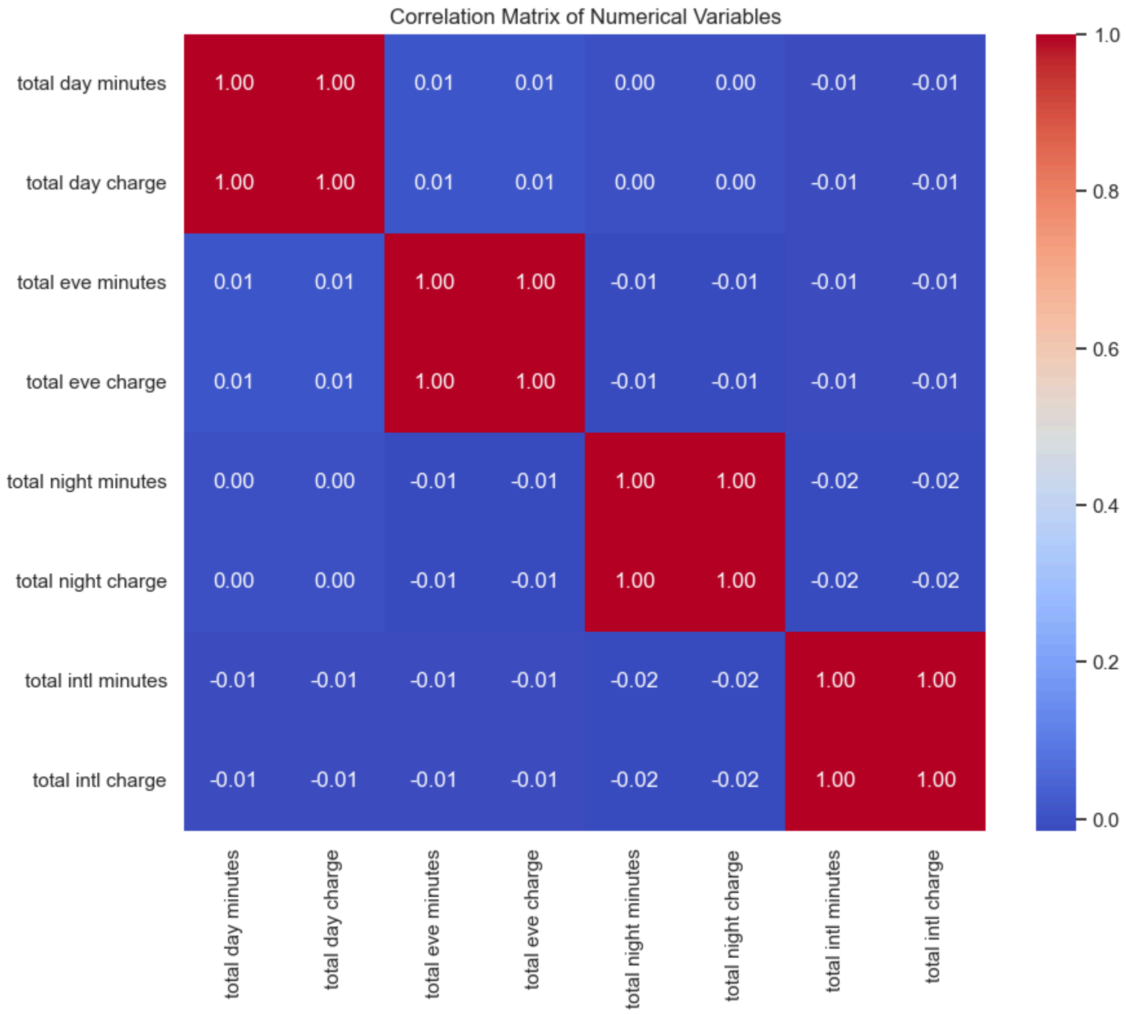


Most of the subscribers have not churned, however, those who have churned spend more day minutes compared to non-churn customers.

MULTIVARIATE ANALYSIS

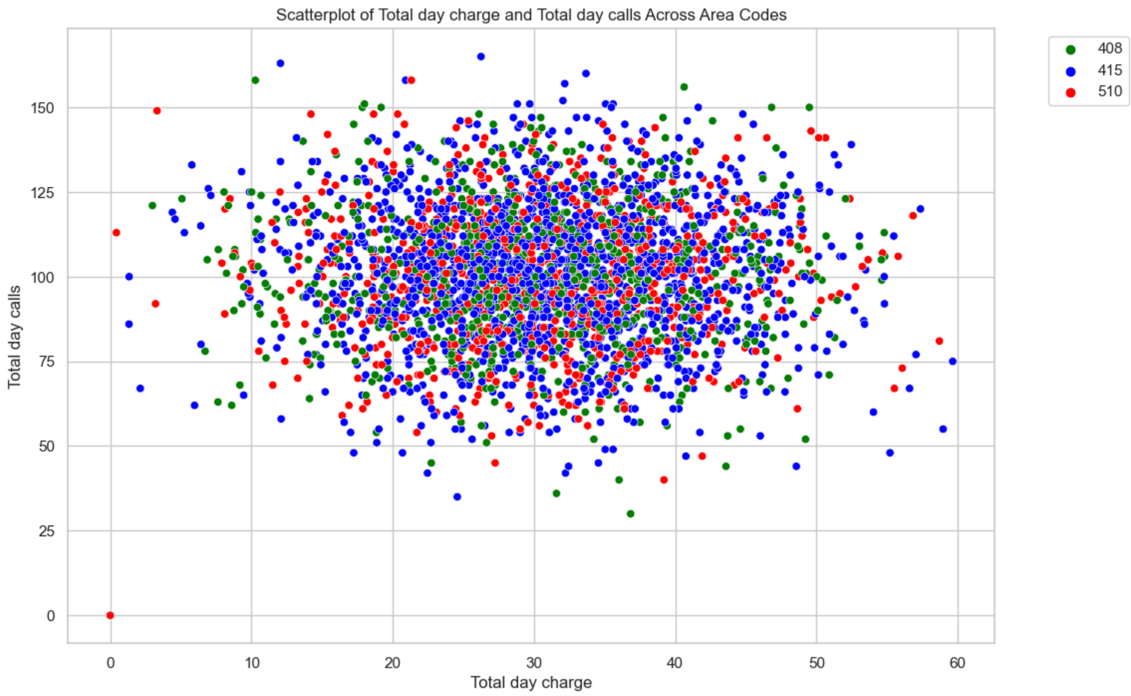
In [18]:

```
plt.figure(figsize=(10, 8))
correlation_matrix = df[['total day minutes', 'total day charge', 'total eve m
                         'total eve charge', 'total night minutes', 'total nig
                         'total intl minutes', 'total intl charge']].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix of Numerical Variables')
plt.show()

# There is a positive correlation between charges and minutes where it
# increases along with the total day minutes.
```



Observation: Total minutes(day, evening, and night) have a very positive correlation with total charge(day, evening, and night)

In [74]:
```
plt.figure(figsize=(12, 8))
sns.scatterplot(x='total day charge', y='total day calls', hue='area code', da
plt.xlabel('Total day charge')
plt.ylabel('Total day calls')
plt.title('Scatterplot of Total day charge and Total day calls Across Area Cod
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')  # Adjust legend positi
plt.show()
```



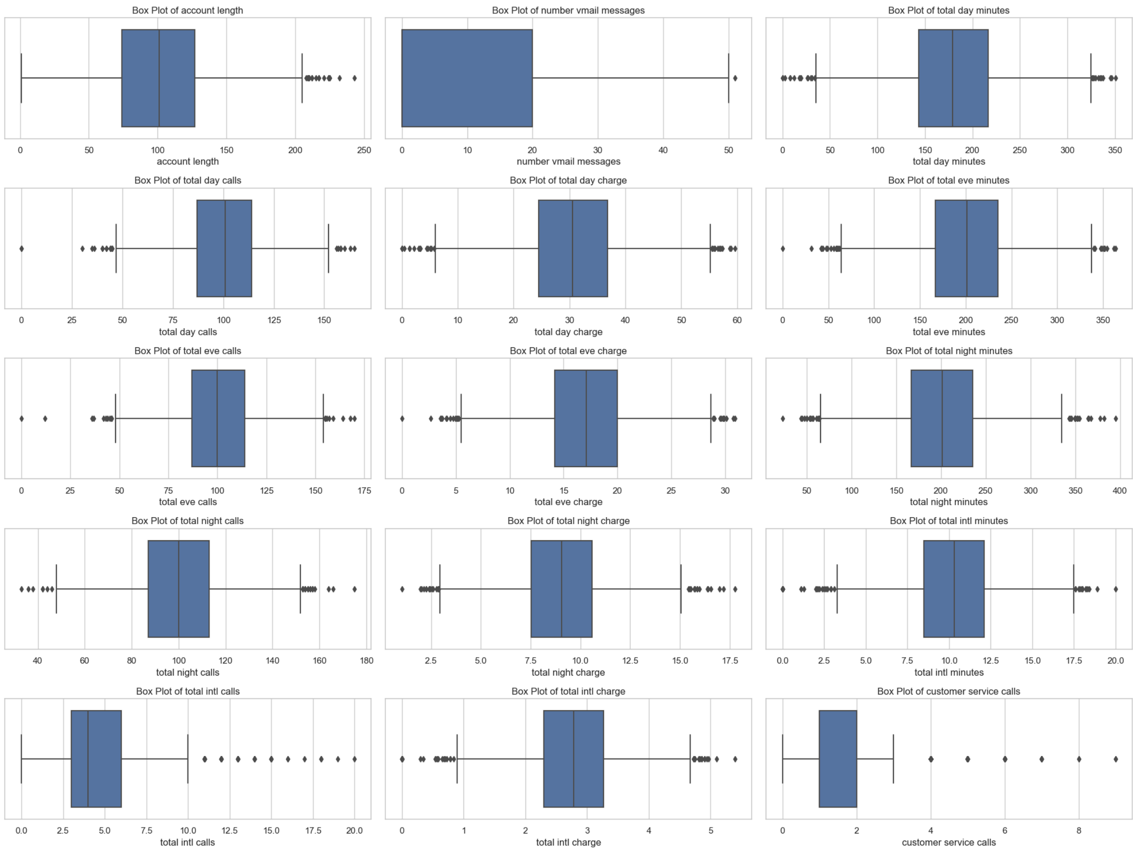
Observation:

There is a high total day calls and day harge from area code 415

Checking for outliers in the data

In [77]:

```
# Identify outliers among the numerical features
numerical_features = [
    'account length', 'number vmail messages', 'total day minutes', 'total day
    'total day charge', 'total eve minutes', 'total eve calls', 'total eve cha
    'total night minutes', 'total night calls', 'total night charge', 'total i
    'total intl calls', 'total intl charge', 'customer service calls'
]

# Plot box plots for numerical features
plt.figure(figsize=(20, 15))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(x=df[feature])
    plt.title(f'Box Plot of {feature}')
plt.tight_layout()
plt.show()
```



I chose to handle outliers through flooring since this method modifies extreme values to be within a reasonable range without necessarily removing them.

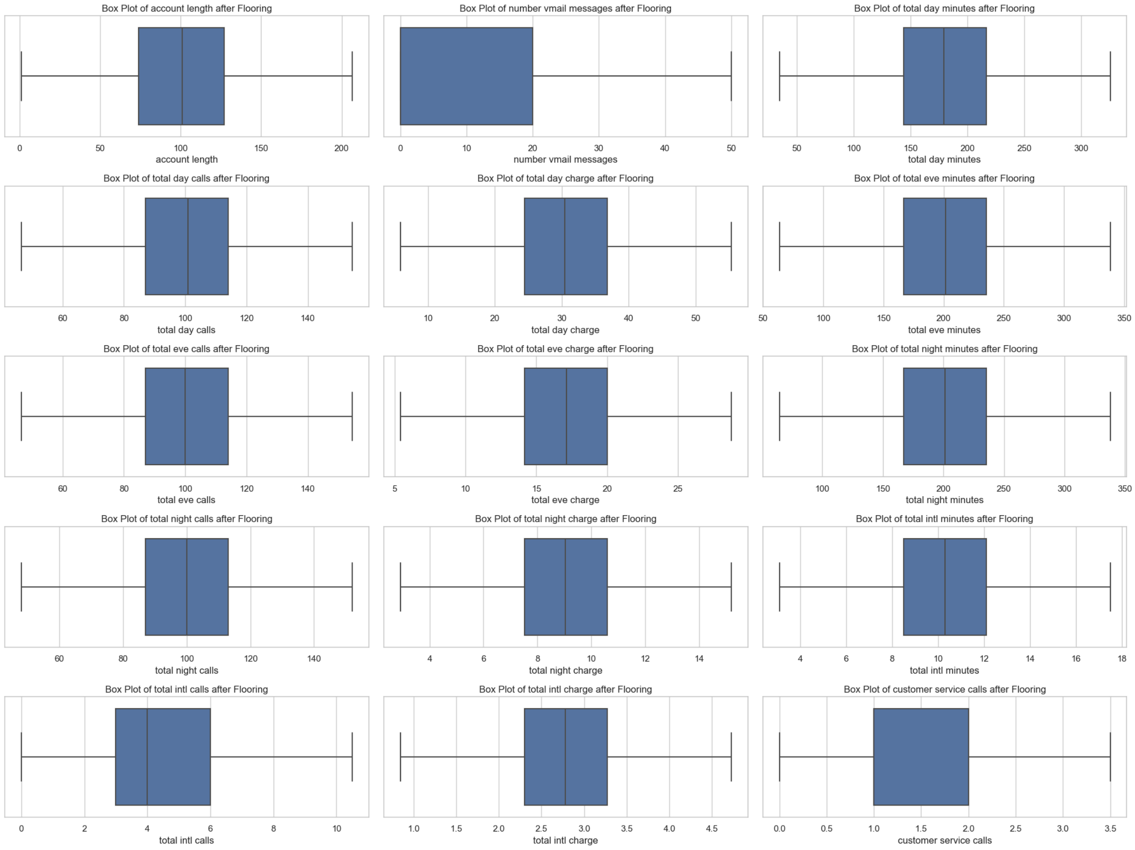
In [82]:
```
# Handling outliers through flooring
def cap_floor_outliers(df, feature):
    Q1 = df[feature].quantile(0.25)
    Q3 = df[feature].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df[feature] = df[feature].apply(lambda x: upper_bound if x > upper_bound e
    return df
# Cap or floor outliers in the dataset
for feature in numerical_features:
    df = cap_floor_outliers(df, feature)
```

In [83]:
```
# Assessing the new dataset without outliers
df.describe()
```

Out[83]:

	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total e' cal
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0000(
mean	101.003300	8.098710	179.816157	100.473597	30.569292	201.009541	100.1341'
std	39.644112	13.687436	54.152190	19.863740	9.205865	50.401365	19.7585(
min	1.000000	0.000000	34.650000	46.500000	5.890000	63.550000	46.5000(
25%	74.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.0000(
50%	101.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.0000(
75%	127.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.0000(
max	206.500000	50.000000	325.450000	154.500000	55.330000	338.350000	154.5000(

In [84]:
```
# Plot box plots for numerical features after flooring
plt.figure(figsize=(20, 15))
for i, feature in enumerate(numerical_features, 1):
    plt.subplot(5, 3, i)
    sns.boxplot(x=df[feature])
    plt.title(f'Box Plot of {feature} after Flooring')
plt.tight_layout()
plt.show()
```



This approach will help retain most of the data while mitigating the influence of extreme values, providing a more robust dataset for modeling customer churn prediction.

I further checked for multicollinearity to enhance the model and drop features that have a strong correlation.

In [103]:
```
# Checking for multicollinearity

from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import StandardScaler

numerical_features = ['account length', 'total day minutes', 'total eve minute
                      'total night minutes', 'total intl minutes', 'total day
                      'total eve calls', 'total night calls', 'total intl call
                      'total day charge', 'total eve charge', 'total night cha
                      'total intl charge']

# Standardize the numerical features
scaler = StandardScaler()
df[numerical_features] = scaler.fit_transform(df[numerical_features])

# Calculate VIF for each numerical feature
vif_data = pd.DataFrame()
vif_data["Feature"] = numerical_features
vif_data["VIF"] = [variance_inflation_factor(df[numerical_features].values, i)

# Display the VIF values
print(vif_data)


# There is low to moderate multicollinearity among the numerical features
# which is manageable and thus no other column was dropped.
```

```
              Feature           VIF
0      account length  1.003405e+00
1   total day minutes  1.039471e+07
2   total eve minutes  2.215568e+06
3 total night minutes  5.958165e+05
4  total intl minutes  6.250972e+04
5     total day calls  1.004386e+00
6     total eve calls  1.002592e+00
7   total night calls  1.001953e+00
8    total intl calls  1.002096e+00
9    total day charge  1.039472e+07
10   total eve charge  2.215567e+06
11 total night charge  5.958159e+05
12  total intl charge  6.250999e+04
```

There is low to moderate multicollinearity among the numerical features which is manageable and thus no other column was dropped.

DATA PREPROCESSING

In [144]:
```
# Importing necessary libraries
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_r
from sklearn.neighbors import KNeighborsClassifier
```

Encoding categorical variables

In [85]:

```python
# Check the unique values in the 'state' column
print(df['state'].unique())

# Perform one-hot encoding for the 'state' column
df_encoded = pd.get_dummies(df, columns=['state'])

# Display the first few rows to verify the changes
df_encoded.head()
```

```
['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'LA' 'WV' 'IN' 'RI' 'IA' 'MT' 'NY'
 'ID' 'VT' 'VA' 'TX' 'FL' 'CO' 'AZ' 'SC' 'NE' 'WY' 'HI' 'IL' 'NH' 'GA'
 'AK' 'MD' 'AR' 'WI' 'OR' 'MI' 'DE' 'UT' 'CA' 'MN' 'SD' 'NC' 'WA' 'NM'
 'NV' 'DC' 'KY' 'ME' 'MS' 'TN' 'PA' 'CT' 'ND']
```

Out[85]:

	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	...	st
0	128.0	415	no	yes	25.0	265.1	110.0	45.07	197.40	99.0	...	
1	107.0	415	no	yes	26.0	161.6	123.0	27.47	195.50	103.0	...	
2	137.0	415	no	no	0.0	243.4	114.0	41.38	121.20	110.0	...	
3	84.0	408	yes	no	0.0	299.4	71.0	50.90	63.55	88.0	...	
4	75.0	415	yes	no	0.0	166.7	113.0	28.34	148.30	122.0	...	

5 rows × 70 columns

In [86]:

```python
# Performing one-hot encoding for the 'state' column
df_encoded = pd.get_dummies(df, columns=['area code'])

# Display the first few rows to verify the changes
df_encoded.head()
```

Out[86]:

	state	account length	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	...	t n c
0	KS	128.0	no	yes	25.0	265.1	110.0	45.07	197.40	99.0	...	[illegible]
1	OH	107.0	no	yes	26.0	161.6	123.0	27.47	195.50	103.0	...	[illegible]
2	NJ	137.0	no	no	0.0	243.4	114.0	41.38	121.20	110.0	...	[illegible]
3	OH	84.0	yes	no	0.0	299.4	71.0	50.90	63.55	88.0	...	[illegible]
4	OK	75.0	yes	no	0.0	166.7	113.0	28.34	148.30	122.0	...	[illegible]

5 rows × 22 columns

In [177]:

```
# Identifying categorical features
categorical_features = ['international plan', 'voice mail plan']

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Apply label encoding to each categorical feature
for feature in categorical_features:
    df[feature] = label_encoder.fit_transform(df[feature])

df.head()
```

Out[177]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minu
0	16	0.681078	415	0	1	25.0	1.575128	0.479660	1.575396	-0.071
1	35	0.151286	415	0	1	26.0	-0.336439	1.134217	-0.336715	-0.109
2	31	0.908132	415	0	0	0.0	1.174346	0.681062	1.174505	-1.583
3	35	-0.428963	408	1	0	0.0	2.208623	-1.484012	2.208783	-2.727
4	36	-0.656017	415	1	0	0.0	-0.242246	0.630711	-0.242196	-1.045

I then progressed to scaling my data so it can be used for the modeling phase.

In [97]:

```
# Scaling numerical features
numerical_features = ['account length', 'total day minutes', 'total eve minute
                      'total night minutes', 'total intl minutes', 'total day
                      'total eve calls', 'total night calls', 'total intl call
                      'total day charge', 'total eve charge', 'total night cha
                      'total intl charge']

# Initializing the StandardScaler
scaler = StandardScaler()

# Instantiating and fitting the scaler
df[numerical_features] = scaler.fit_transform(df[numerical_features])

df.head()
```

Out[97]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minu
0	16	0.681078	415	0	1	25.0	1.575128	0.479660	1.575396	-0.071
1	35	0.151286	415	0	1	26.0	-0.336439	1.134217	-0.336715	-0.109
2	31	0.908132	415	0	0	0.0	1.174346	0.681062	1.174505	-1.583
3	35	-0.428963	408	1	0	0.0	2.208623	-1.484012	2.208783	-2.727
4	36	-0.656017	415	1	0	0.0	-0.242246	0.630711	-0.242196	-1.045

In this data pre-processing phase, the last task was to perform a test train split for modeling.

In [98]:
```
# Performing the Test Train Split
X = df.drop(columns=['churn'])
y = df['churn']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Display the shapes of the split datasets
print("Training set shape:", X_train.shape, y_train.shape)
print("Testing set shape:", X_test.shape, y_test.shape)
```

```
Training set shape: (2666, 19) (2666,)
Testing set shape: (667, 19) (667,)
```

MODELING

1. Baseline model

I proceeded to use logistic regression for the baseline model, since it works well with binary classification.

In [162]:
```
from sklearn.linear_model import LogisticRegression

# Initializing the logistic regression model
baseline_model = LogisticRegression(random_state=42, max_iter=100)

# Train the model on the training data
baseline_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = baseline_model.predict(X_test)

# Evaluate the model's performance
baseline_model_accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", baseline_model_accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

```
Accuracy: 0.8605697151424287

Confusion Matrix:
 [[557   9]
 [ 84  17]]

Classification Report:
               precision    recall  f1-score   support

       False       0.87      0.98      0.92       566
        True       0.65      0.17      0.27       101

    accuracy                           0.86       667
   macro avg       0.76      0.58      0.60       667
weighted avg       0.84      0.86      0.82       667


C:\Users\DELL\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.py:4
60: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://sciki
t-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regres
sion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession)
  n_iter_i = _check_optimize_result(
```

Model evaluation

Precision:

For the "False" class, the precision is 0.87. This means that when the model predicts "False," it is correct 87% of the time. For the "True" class, the precision is 0.65. This means that when the model predicts "True," it is correct 65% of the time.

Recall:

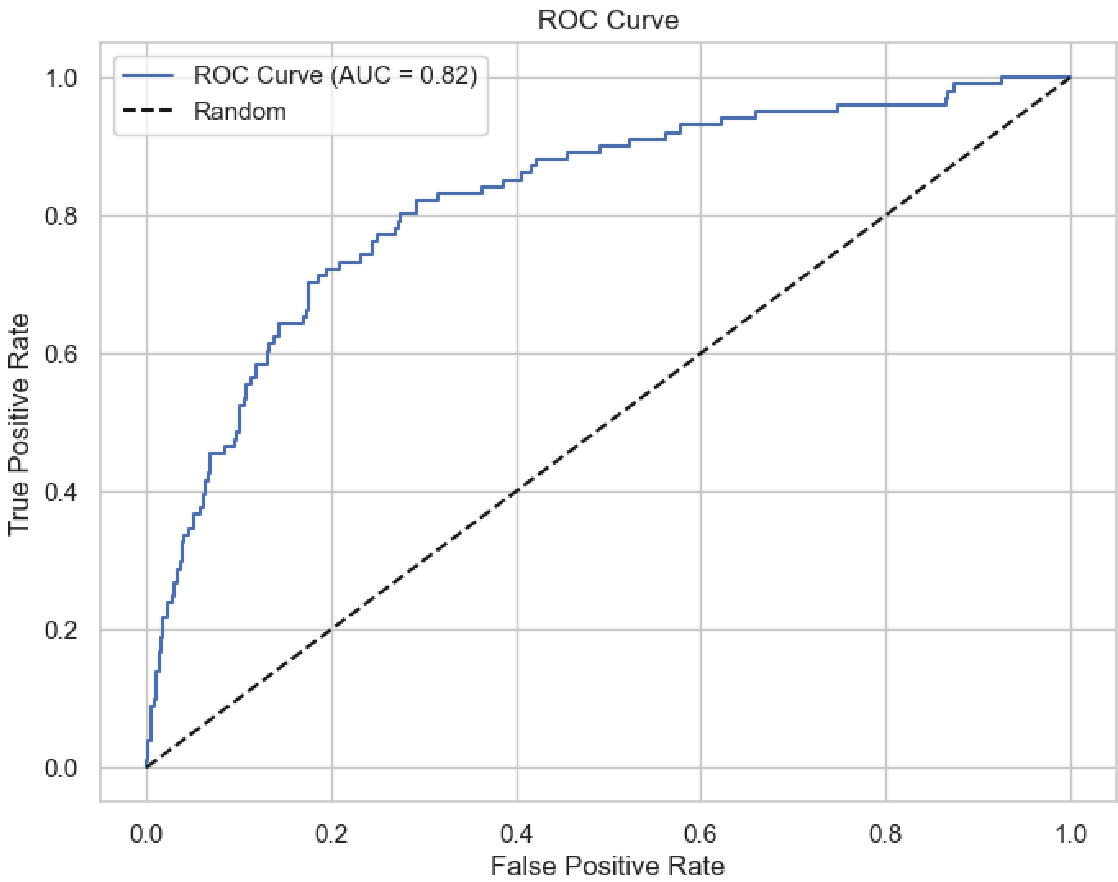
For the "False" class, the recall is 0.98. This means that 98% of the actual "False" instances are correctly identified by the model. For the "True" class, the recall is 0.17. This means that only 17% of the actual "True" instances are correctly identified by the model. This is relatively low, indicating that the model misses a lot of true positive cases.

F1-score:

The F1-score for the "False" class is 0.92, which is a harmonic mean of precision and recall,

In [173]:

```
from sklearn.metrics import roc_curve, roc_auc_score

# Get predicted probabilities for the positive class (churn)
y_prob = baseline_model.predict_proba(X_test)[:, 1]

# Compute false positive rate (FPR), true positive rate (TPR), and thresholds
fpr, tpr, thresholds = roc_curve(y_test, y_prob)

# Compute area under the ROC curve (AUC)
auc = roc_auc_score(y_test, y_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.grid(True)
plt.show()
```



The logistic regression model has an accuracy of 86%. My model has relatively high accuracy, indicating that it performs well in terms of overall correctness. However, the precision is relatively low, suggesting that there is a high rate of false positives among the predicted churn cases. This could indicate that the model is incorrectly labeling some non-churners as churners. The recall is moderate, indicating that the model is moderately successful at capturing actual churn cases, but there is room for improvement. The specificity is relatively high, indicating that the model is good at correctly identifying non-churn cases.

From the ROC curve plot, the model has a relative good performance with area under the curve being relatively close to 1.

However, I chose to explore other models to check performance and churn prediction for better results.

2. DecisionTree Classifier

In [113]:

```python
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
# Initializing the Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Train the model
dt_model.fit(X_train, y_train)

# Make predictions
y_pred = dt_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
class_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("\nConfusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

```
Accuracy: 0.9190404797601199

Confusion Matrix:
 [[538  28]
 [ 26  75]]

Classification Report:
               precision    recall  f1-score   support

       False       0.95      0.95      0.95       566
        True       0.73      0.74      0.74       101

    accuracy                           0.92       667
   macro avg       0.84      0.85      0.84       667
weighted avg       0.92      0.92      0.92       667
```

For the class labeled "False":

Precision: 0.95 - This means that when the model predicts "False," it is correct 95% of the time. Recall: 0.95 - This means that 95% of the actual "False" instances are correctly identified by the model. F1-score: 0.95 - This is the harmonic mean of precision and recall, indicating a high level of accuracy for this class. Support: 566 - This is the number of actual instances of this class in the test set.

For the class labeled "True":

Precision: 0.73 - This means that when the model predicts "True," it is correct 73% of the time. Recall: 0.74 - This means that 74% of the actual "True" instances are correctly identified by the model. F1-score: 0.74 - This is the harmonic mean of precision and recall, indicating a moderate level of accuracy for this class. Support: 101 - This is the number of actual instances of this class in the test set.

Class Imbalance: The class "True" (churn) has fewer instances (101) compared to "False" (non-churn) with 566 instances. Despite this, the model performs reasonably well on the minority class.

Precision and Recall for "True" class: The precision and recall for the "True" class are lower compared to the "False" class, indicating that there is room for improvement in identifying churn

The Decision Tree model is performing well on this dataset. However, we further evaluation metrics or techniques to fine-tune the model for a more accurate performance, using grid searchCV.

In [115]:

```
from sklearn.model_selection import GridSearchCV

# I define the parameter grid
param_grid = {
    'max_depth': [None, 5, 10, 15],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [None, 'sqrt', 'log2'],  # Corrected options for max_featu
    'criterion': ['gini', 'entropy']
}

# Initialize the GridSearchCV object
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(random_state=42),
                           param_grid=param_grid,
                           scoring='accuracy',
                           cv=5,
                           n_jobs=-1)

# Perform grid search
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_

# Train the final model using the best hyperparameters
final_model = DecisionTreeClassifier(random_state=42, **best_params)
final_model.fit(X_train, y_train)

# Evaluate the final model
final_accuracy = final_model.score(X_test, y_test)

print("Best Hyperparameters:", best_params)
print("Final Model Accuracy:", final_accuracy)

print("Training Accuracy (Regularized Random Forest):", train_accuracy_rf_regu
print("Testing Accuracy (Regularized Random Forest):", test_accuracy_rf_regula
```

```
Best Hyperparameters: {'criterion': 'gini', 'max_depth': 5, 'max_features': N
one, 'min_samples_leaf': 1, 'min_samples_split': 2}
Final Model Accuracy: 0.9370314842578711
```

In [156]:

```
# Predict on the test set
y_pred = final_model.predict(X_test)

# Calculate the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Generate a classification report
class_report = classification_report(y_test, y_pred)

print("Confusion Matrix:\n", conf_matrix)
print("\nClassification Report:\n", class_report)
```

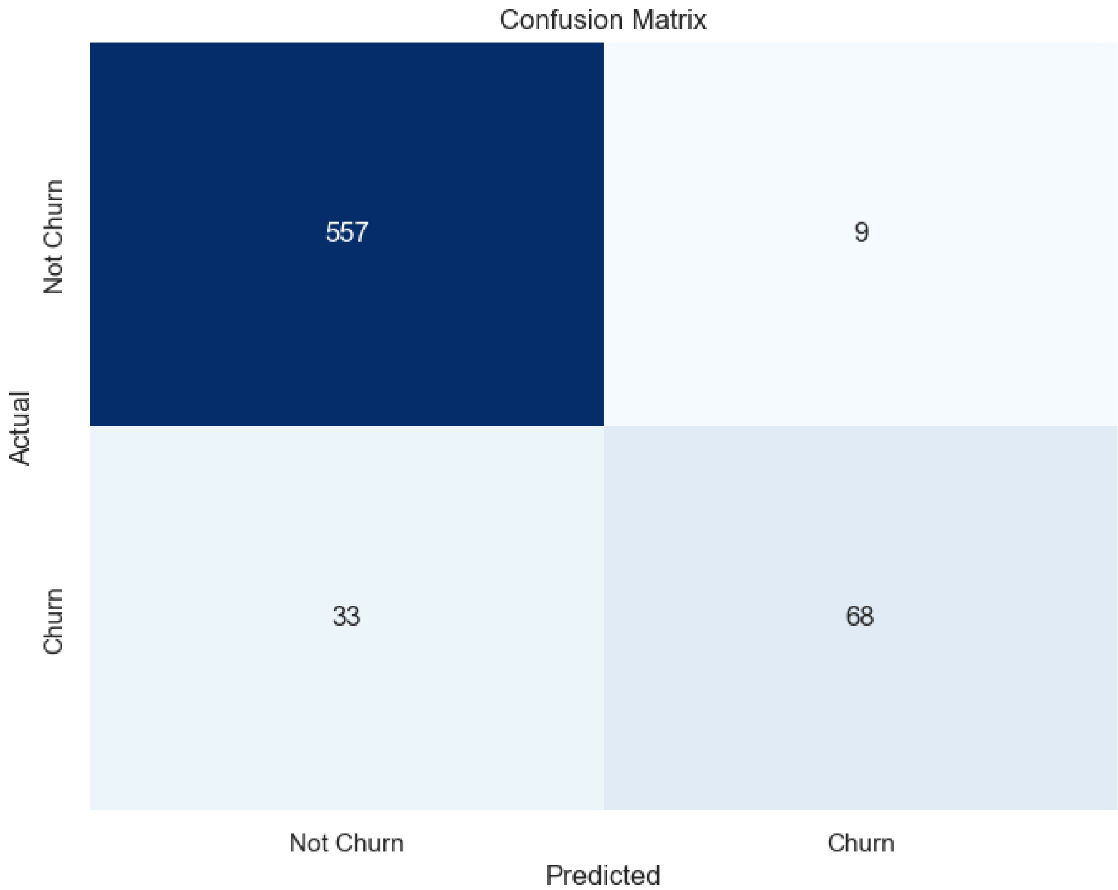
```
Confusion Matrix:
 [[557   9]
 [ 33  68]]

Classification Report:
               precision    recall  f1-score   support

       False       0.94      0.98      0.96       566
        True       0.88      0.67      0.76       101

    accuracy                           0.94       667
   macro avg       0.91      0.83      0.86       667
weighted avg       0.93      0.94      0.93       667
```

Improved Performance

High Accuracy: The model's accuracy of 94% is very high. Improved Precision for "True" class: Precision for the "True" class (churn) has increased to 0.88, meaning fewer false positives compared to previous models. Improved Recall for "False" class: Recall for the "False" class (non-churn) remains high at 0.98, indicating that the model is very good at identifying non-churn customers.

In [157]:

```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Churn', 'Churn'],
            yticklabels=['Not Churn', 'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



The model's accuracy seems to have improved. This I further tested the test and train scores to check for overfitting.

In [131]:

```
# Predict on training and testing data
y_train_pred = grid_search.best_estimator_.predict(X_train)
y_test_pred = grid_search.best_estimator_.predict(X_test)

# Calculate training and testing accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Training Accuracy: {train_accuracy:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
print('\nHooray!! The model no longer overfits and has an overall improvement!
```

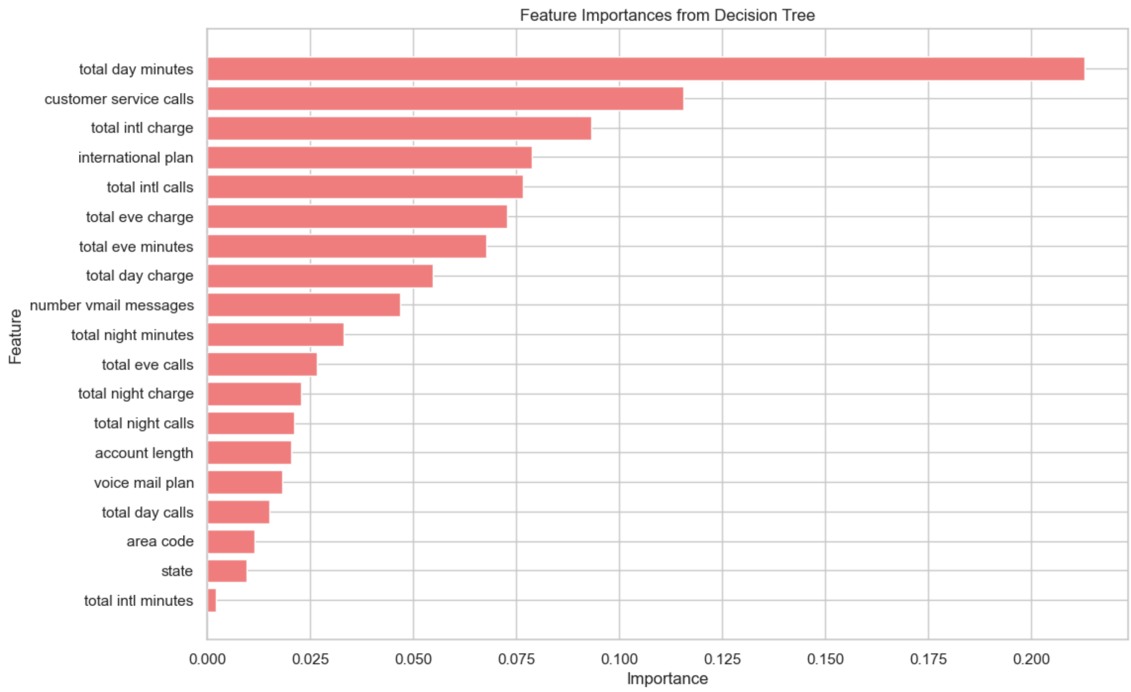
```
Training Accuracy: 0.9572
Test Accuracy: 0.9370

Hooray!! The model no longer overfits and has an overall improvement!
```

Feature importances from the Decision Tree Model

In [128]:

```
# Printing the feature importances of the decision tree model to determine the
# features that are are worth considering in churn or not churn
importances = dt_model.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame({'feature': feature_names, 'importance':

# Sort the DataFrame by importance
feature_importance_df = feature_importance_df.sort_values(by='importance', asc

# Print the feature importances DataFrame
print(feature_importance_df)

# Plot the feature importances
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['feature'], feature_importance_df['importance']
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Decision Tree')
plt.gca().invert_yaxis() # Highest importance at the top
plt.show()
```

```
                  feature  importance
6        total day minutes    0.212792
18  customer service calls    0.115501
17        total intl charge   0.093281
3        international plan   0.078900
16         total intl calls   0.076623
11         total eve charge   0.072863
9         total eve minutes   0.067832
8          total day charge   0.054761
5     number vmail messages   0.046925
12      total night minutes   0.033205
10          total eve calls   0.026551
14       total night charge   0.022754
13        total night calls   0.021134
1            account length   0.020415
4           voice mail plan   0.018267
7           total day calls   0.015156
2                 area code   0.011433
0                     state   0.009505
15       total intl minutes   0.002102
```



3. Random Forest Classifier

I then proceed to use a different model to increase variation in prediction models. Thus, I chose to use Random Forest Classifier as in the following code.

In [137]:

```
# Random forest
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

rf_accuracy = accuracy_score(y_test, y_pred_rf)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
class_report_rf = classification_report(y_test, y_pred_rf)

print("Random Forest Accuracy:", rf_accuracy)
print("\nConfusion Matrix:\n", conf_matrix_rf)
print("\nClassification Report:\n", class_report_rf)

print("\n")
# Predictions on training data
y_train_pred_rf = rf_model.predict(X_train)
train_accuracy_rf = accuracy_score(y_train, y_train_pred_rf)

# Predictions on testing data
test_accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Training Accuracy (Random Forest):", train_accuracy_rf)
print("Testing Accuracy (Random Forest):", test_accuracy_rf)
print('\n')
print('The model seems to be overfitting and I thus proceed to induce regulari
print('to prevent overfitting and improve the model.')
```

```
Random Forest Accuracy: 0.9475262368815592

Confusion Matrix:
 [[561   5]
 [ 30  71]]

Classification Report:
               precision    recall  f1-score   support

       False       0.95      0.99      0.97       566
        True       0.93      0.70      0.80       101

    accuracy                           0.95       667
   macro avg       0.94      0.85      0.89       667
weighted avg       0.95      0.95      0.94       667



Training Accuracy (Random Forest): 1.0
Testing Accuracy (Random Forest): 0.9475262368815592


The model seems to be overfitting and I thus proceed to induce regularization
to prevent overfitting and improve the model.
```

Precision:

Non-Churn (False): 0.95 - Out of all the customers predicted as non-churn, 95% actually did not churn. Churn (True): 0.93 - Out of all the customers predicted as churn, 93% actually churned.

Recall:

Non-Churn (False): 0.99 - Out of all the customers who did not churn, the model correctly identified 99% of them. Churn (True): 0.70 - Out of all the customers who churned, the model correctly identified 70% of them.

F1-Score:

Non-Churn (False): 0.97 - The F1-score is the harmonic mean of precision and recall for non-churn, indicating high accuracy. Churn (True): 0.80 - The F1-score for churn indicates a good balance between precision and recall, but with room for improvement.

The model worked well but with a training accuracy of 100%, it is overfitting. To improve it, I chose to use regularization to balance bias-Trade off and vairance and to control the model's complexity.

In [158]:
```python
# Initializing the Random Forest classifier with regularization parameters
rf_model_regularized = RandomForestClassifier(max_depth=10, min_samples_split=

# Train the regularized model on the training data
rf_model_regularized.fit(X_train, y_train)

# Predictions on the testing data
y_pred_rf_regularized = rf_model_regularized.predict(X_test)

# Evaluate the regularized model
rf_accuracy_regularized = accuracy_score(y_test, y_pred_rf_regularized)
conf_matrix_rf_regularized = confusion_matrix(y_test, y_pred_rf_regularized)
class_report_rf_regularized = classification_report(y_test, y_pred_rf_regulari

print("Regularized Random Forest Accuracy:", rf_accuracy_regularized)
print("\nConfusion Matrix:\n", conf_matrix_rf_regularized)
print("\nClassification Report:\n", class_report_rf_regularized)

print('\n')

# Predictions on training data
y_train_pred_rf_regularized = rf_model_regularized.predict(X_train)
train_accuracy_rf_regularized = accuracy_score(y_train, y_train_pred_rf_regula

# Predictions on testing data
test_accuracy_rf_regularized = accuracy_score(y_test, y_pred_rf_regularized)

print("Training Accuracy (Regularized Random Forest):", train_accuracy_rf_regu
print("Testing Accuracy (Regularized Random Forest):", test_accuracy_rf_regula
```

```
Regularized Random Forest Accuracy: 0.9430284857571214

Confusion Matrix:
 [[561   5]
 [ 33  68]]

Classification Report:
               precision    recall  f1-score   support

       False       0.94      0.99      0.97       566
        True       0.93      0.67      0.78       101

    accuracy                           0.94       667
   macro avg       0.94      0.83      0.87       667
weighted avg       0.94      0.94      0.94       667



Training Accuracy (Regularized Random Forest): 0.9756189047261815
Testing Accuracy (Regularized Random Forest): 0.9430284857571214
```

Precision:

Non-Churn (False): 0.94 - Out of all the customers predicted as non-churn, 94% actually did not churn. Churn (True): 0.93 - Out of all the customers predicted as churn, 93% actually churned.

Recall:

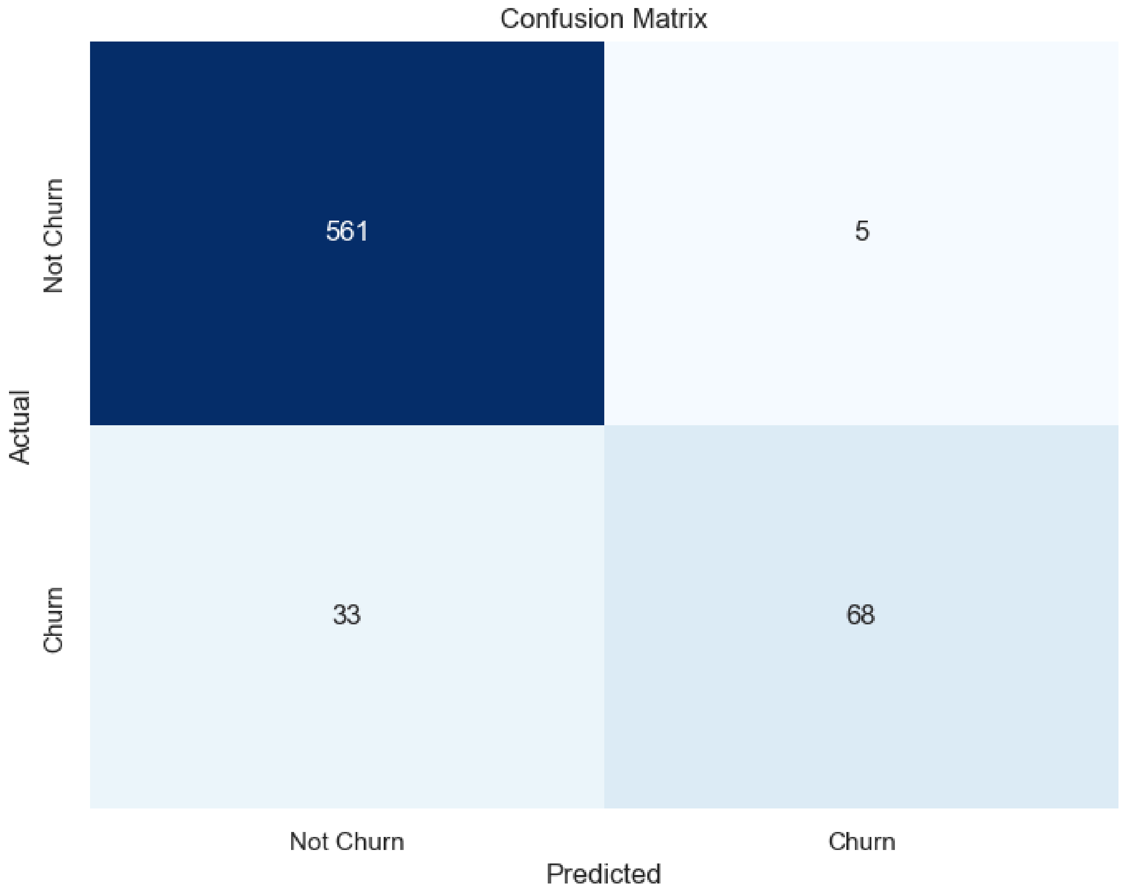
Non-Churn (False): 0.99 - Out of all the customers who did not churn, the model correctly identified 99% of them. Churn (True): 0.67 - Out of all the customers who churned, the model correctly identified 67% of them.

F1-Score:

Non-Churn (False): 0.97 - The F1-score is the harmonic mean of precision and recall for non-churn, indicating high accuracy. Churn (True): 0.78 - The F1-score for churn indicates a relatively good balance between precision and recall.

Then I printed the confusion matrix to check prediction performance.

In [160]:
```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf_regularized, annot=True, fmt='d', cmap='Blues', cba
            xticklabels=['Not Churn', 'Churn'],
            yticklabels=['Not Churn', 'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```

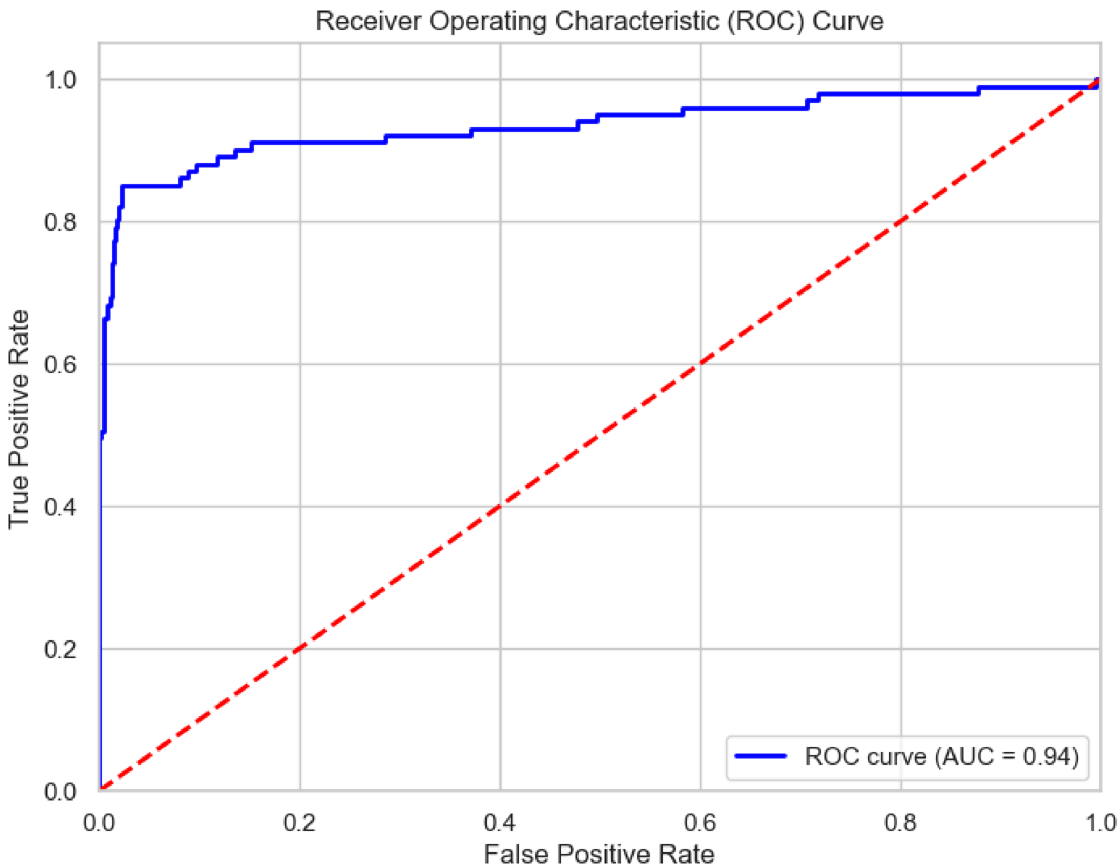


The confusion matrix show an improvement since there is no high accuracy compared to the overfitted model.

I proceeded to plot the ROC curve to determine the model's performance and accuracy in prediction.

In [170]:

```
from sklearn.metrics import roc_curve, auc
# Calculate the probabilities for each class
y_prob_rf_regularized = rf_model_regularized.predict_proba(X_test)

# Compute ROC curve and AUC for class 1 (churn)
fpr, tpr, thresholds = roc_curve(y_test, y_prob_rf_regularized[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (AUC = %0.2f)' % roc_a
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



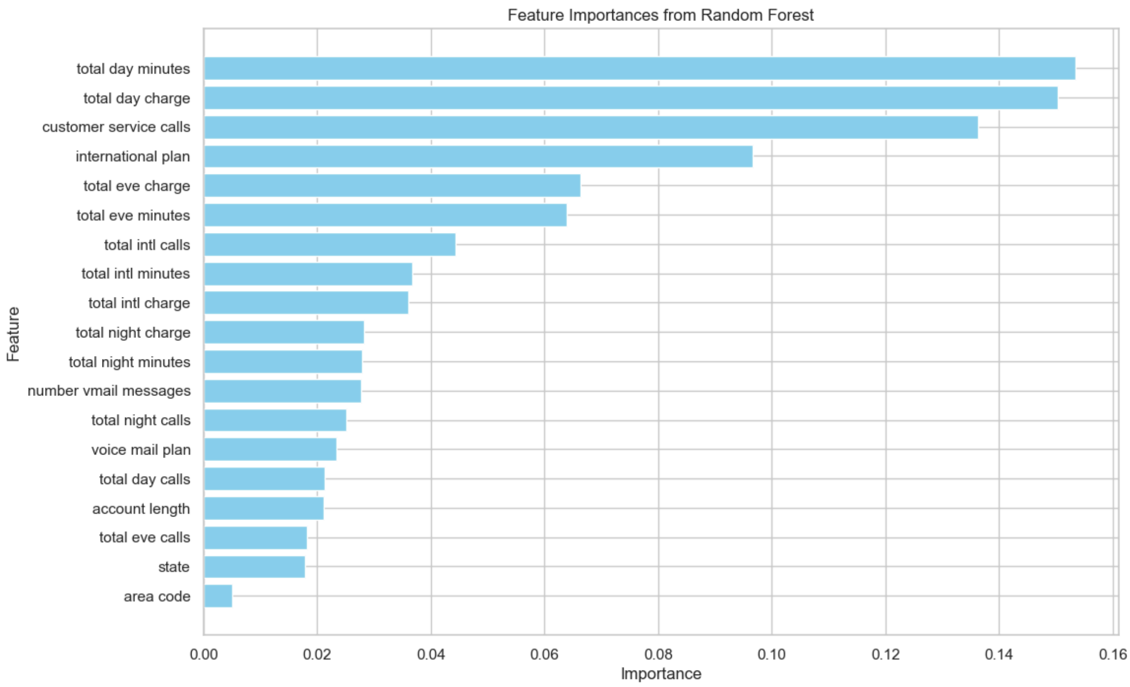
From the ROC curve above, the model performance has a good prediction since the area under the curve is 0.94, which is close to 1.

Printing feature importances from the Random forest model

In [127]:

```
# Printing the feature importances of the Random forest model to determine the
# features that are are worth considering in churn or not churn
importances = rf_model_regularized.feature_importances_
feature_names = X.columns
feature_importance_df = pd.DataFrame({'feature': feature_names, 'importance':
feature_importance_df = feature_importance_df.sort_values(by='importance', asc
print(feature_importance_df)

# Plot the feature importances
plt.figure(figsize=(12, 8))
plt.barh(feature_importance_df['feature'], feature_importance_df['importance']
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.title('Feature Importances from Random Forest')
plt.gca().invert_yaxis() # Highest importance at the top
plt.show()
```

```
                   feature  importance
6         total day minutes    0.153395
8          total day charge    0.150309
18   customer service calls    0.136246
3        international plan    0.096654
11         total eve charge    0.066361
9         total eve minutes    0.063936
16         total intl calls    0.044318
15       total intl minutes    0.036781
17        total intl charge    0.036059
14       total night charge    0.028233
12      total night minutes    0.027941
5     number vmail messages    0.027763
13        total night calls    0.025043
4           voice mail plan    0.023307
7           total day calls    0.021374
1            account length    0.021214
10          total eve calls    0.018224
0                     state    0.017760
2                 area code    0.005081
```



4. K-Nearest Neighbors

Further, I chose to use anaother model, KNN, for prediction. I also thought scaling my data would be necessary for KNN model since it is a distance-based algorithm

In [174]:
```
# Fit and transform the training data, transform the test data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [167]:
```
# Initialize the KNN model with number of neighbors = 5
knn = KNeighborsClassifier(n_neighbors=5)

# Fit the model to the training data
knn.fit(X_train_scaled, y_train)

# Predict on the training data
y_train_pred = knn.predict(X_train_scaled)

# Predict on the test data
y_test_pred = knn.predict(X_test_scaled)

# Calculate training accuracy
train_accuracy = accuracy_score(y_train, y_train_pred)
print(f'Training Accuracy: {train_accuracy:.4f}')

# Calculate test accuracy
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_accuracy:.4f}')

# Confusion matrix and classification report
conf_matrix = confusion_matrix(y_test, y_test_pred)
class_report = classification_report(y_test, y_test_pred)

print("Confusion Matrix:")
print(conf_matrix)
print("\nClassification Report:")
print(class_report)
```

```
Training Accuracy: 0.9122
Test Accuracy: 0.8891
Confusion Matrix:
[[560   6]
 [ 68  33]]

Classification Report:
              precision    recall  f1-score   support

       False       0.89      0.99      0.94       566
        True       0.85      0.33      0.47       101

    accuracy                           0.89       667
   macro avg       0.87      0.66      0.70       667
weighted avg       0.88      0.89      0.87       667
```

Precision:

Non-Churn (False): 0.89 - Out of all the customers predicted as non-churn, 89% actually did not churn. Churn (True): 0.85 - Out of all the customers predicted as churn, 85% actually churned.

Recall:

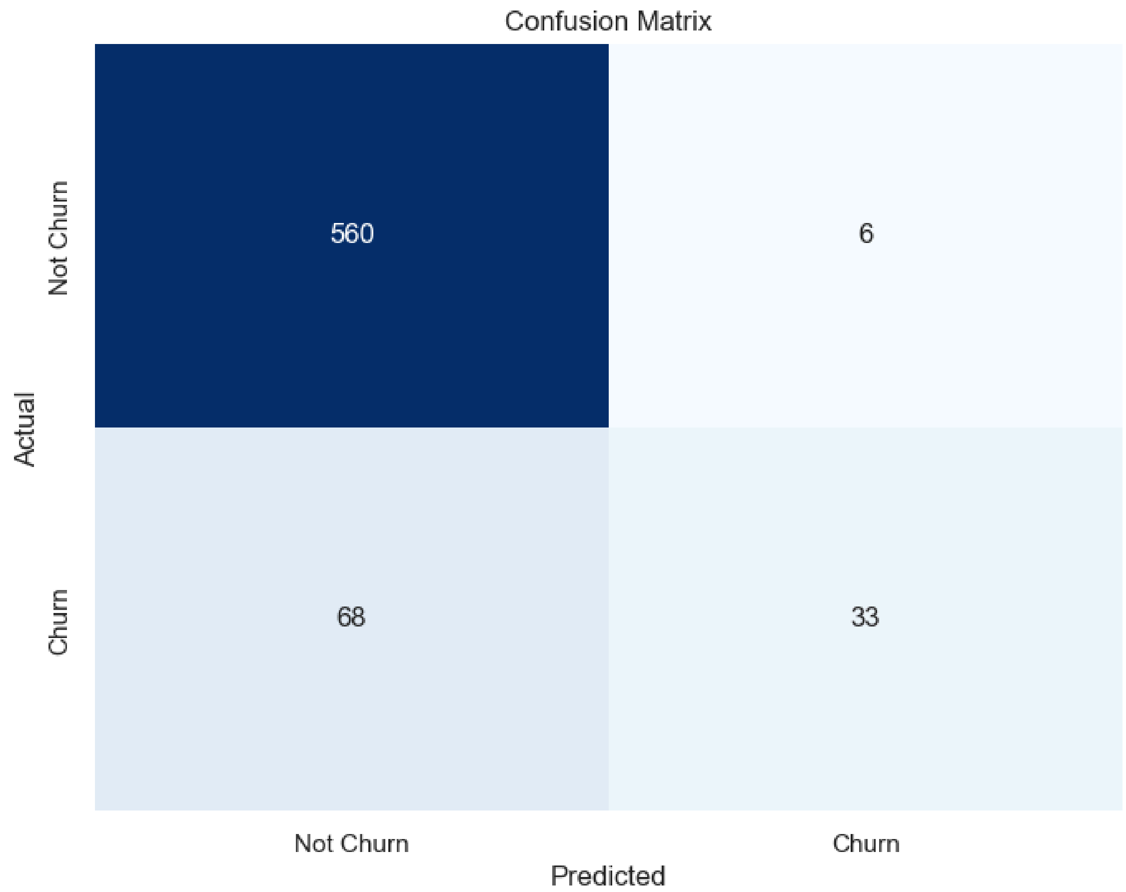
Non-Churn (False): 0.99 - Out of all the customers who did not churn, the model correctly identified 99% of them. Churn (True): 0.33 - Out of all the customers who churned, the model correctly identified 33% of them.

F1-Score:

Non-Churn (False): 0.94 - The F1-score is the harmonic mean of precision and recall for non-churn, indicating high accuracy. Churn (True): 0.47 - The F1-score for churn indicates moderate

In [153]:

```python
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Churn', 'Churn'],
            yticklabels=['Not Churn', 'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



- The model is highly effective in identifying customers who will not churn, with 560 out of 566 non-churn customers correctly classified.
- False Positives (FP): Only 6 customers who are non-churn were incorrectly predicted as churn. This means the model is quite precise in predicting non-churn customers.
- False Negatives (FN): There are 68 customers who were predicted to stay but actually churned. This indicates a weakness in identifying all potential churners, which could be critical for retention strategies.
- True Positives (TP):The model correctly identified 33 churners out of 101 actual churners. This indicates that while the model has some capability in identifying churners, it misses a significant portion.

In [164]:

```
# Importing the relevant libraries for the code
from sklearn.pipeline import make_pipeline
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV, cross_val_score
# Create a pipeline
pipeline = make_pipeline(SVC())

# Set up the parameter grid for GridSearchCV
param_grid = {
    'svc__C': [0.1, 1, 10, 100],
    'svc__gamma': [1, 0.1, 0.01, 0.001],
    'svc__kernel': ['linear', 'rbf']
}

# Initialize GridSearchCV with the pipeline and parameter grid
grid_search = GridSearchCV(pipeline, param_grid, refit=True, verbose=3, cv=5)

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Best hyperparameters from GridSearchCV
print("Best Hyperparameters:", grid_search.best_params_)

# Evaluate the model on test and train sets
knn_test_accuracy = grid_search.score(X_test, y_test)
knn_train_accuracy = grid_search.score(X_train, y_train)
print("Test Accuracy:", knn_test_accuracy)
print("Train Accuracy:", knn_train_accuracy)

# Perform cross-validation
cross_val_scores = cross_val_score(grid_search.best_estimator_, X_train, y_tra
print("Cross-validation scores:", cross_val_scores)
print("Mean cross-validation score:", np.mean(cross_val_scores))
```

```
total time=   1.4s
[CV 5/5] END svc__C=1, svc__gamma=0.001, svc__kernel=linear;, score=0.856
total time=   2.3s
[CV 1/5] END svc__C=1, svc__gamma=0.001, svc__kernel=rbf;, score=0.856 tot
al time=   0.1s
[CV 2/5] END svc__C=1, svc__gamma=0.001, svc__kernel=rbf;, score=0.857 tot
al time=   0.1s
[CV 3/5] END svc__C=1, svc__gamma=0.001, svc__kernel=rbf;, score=0.857 tot
al time=   0.1s
[CV 4/5] END svc__C=1, svc__gamma=0.001, svc__kernel=rbf;, score=0.857 tot
al time=   0.1s
[CV 5/5] END svc__C=1, svc__gamma=0.001, svc__kernel=rbf;, score=0.856 tot
al time=   0.1s
[CV 1/5] END svc__C=10, svc__gamma=1, svc__kernel=linear;, score=0.860 tot
al time=   2.5s
[CV 2/5] END svc__C=10, svc__gamma=1, svc__kernel=linear;, score=0.874 tot
al time=   3.0s
[CV 3/5] END svc__C=10, svc__gamma=1, svc__kernel=linear;, score=0.865 tot
al time=   2.0s
[CV 4/5] END svc__C=10, svc__gamma=1, svc__kernel=linear;, score=0.848 tot
```

Key Observations:

Model Performance: The logistic regression model shows good performance with high accuracy on both training and test datasets, and consistent cross-validation scores.

Generalization: The small gap between train and test accuracy indicates that the model generalizes well to unseen data.

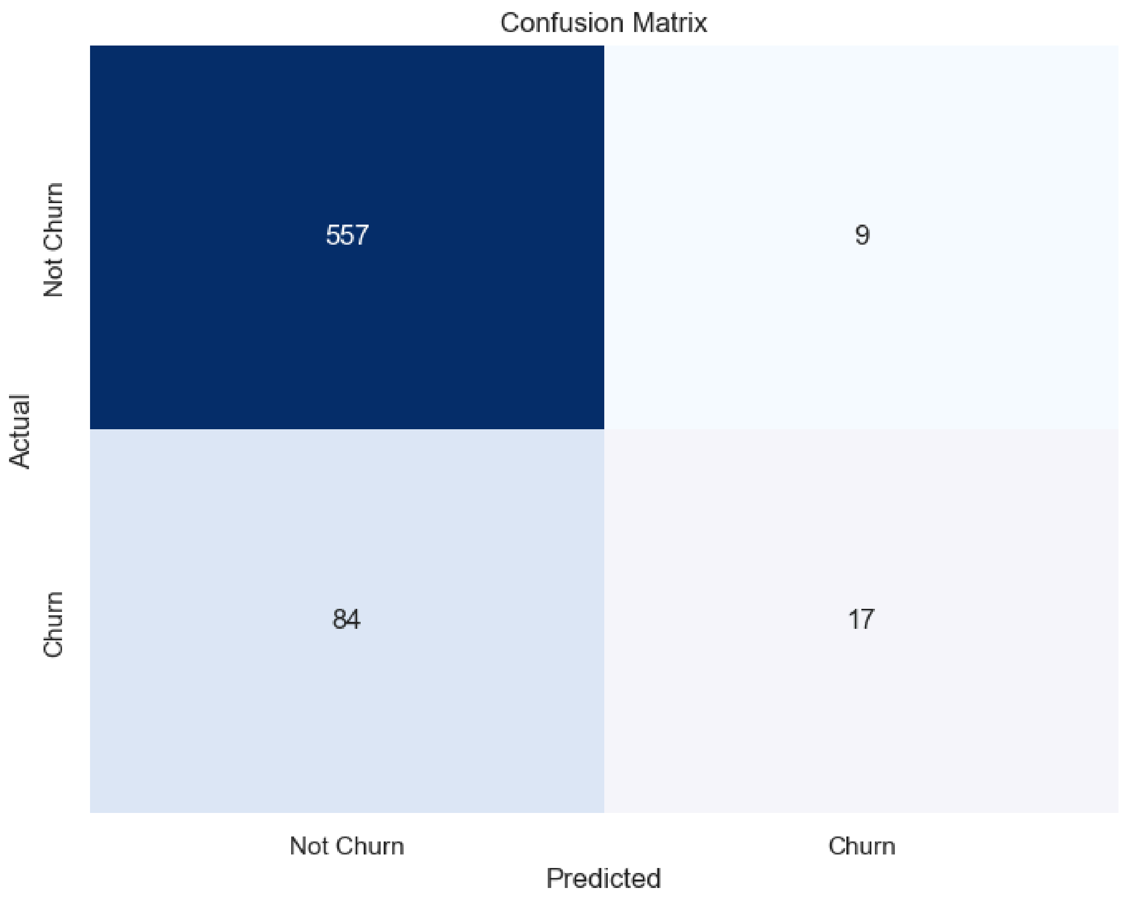
Cross-Validation Insight: The mean cross-validation score supports the robustness of the model, providing confidence in its predictive capability.

The mean cross-validation score of 89.12% served as a valuable tool for assessing and improving the model's performance and generalization ability.

The train accuracy is 89.12% and test accuracy is 86.96%. Not bad, the model is much better now.

In [151]:

```
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=['Not Churn', 'Churn'],
            yticklabels=['Not Churn', 'Churn'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



The final step would then be to check which model of the 4 worked best and why.

In [178]:
```
# Checking the most appropriate variable to predict churn and factors that may

print('Logistic Regression (baseline_model_accuracy):', baseline_model_accurac
print("Decision tree accuracy:", final_accuracy)
print('Random Forest accuracy:', rf_accuracy_regularized)
print('K-Nearest Neighbors accuracy:', knn_train_accuracy)
```

```
Logistic Regression (baseline_model_accuracy): 0.8605697151424287
Decision tree accuracy: 0.9370314842578711
Random Forest accuracy: 0.9430284857571214
K-Nearest Neighbors accuracy: 0.8912228057014253
```

Conclusions

It is noted that customers who have higher usage during the day ("total day minutes" and "total day charge") and those who frequently contact customer service ("customer service calls") are more likely to churn. This suggests that dissatisfaction with service quality or billing issues during peak hours may drive churn.

From the 4 models tested on the dataset, The Random Forest produces the best results with an accuracy of 94.3%.

According to the feature importances, Total day minutes, Total day charge, Customer service calls, International plan,Total eve charge are the top contributing factors to customer churning or not.

Based on the identified key factors influencing customer churn, actionable strategies can be formulated to retain customers identified as high risk for churn.

Limitations of the model

Despite providing feature importances, the random forest may cause a lack of interpretability due to complexity of the hyperparemeter sensitivity. This may hinder the ability to fully understand the key factors influencing customer churn, especially if stakeholders require detailed insights into the drivers of churn. Further, relative importance of features for prediction may not always reflect the true causal relationships between features and the target variable.

Recommendations

- Proactive customer service: Since customer service calls are a significant factor, providing proactive and effective customer support can help address issues or concerns promptly, potentially reducing churn.
- Personalized offers or incentives: Identifying customers with international plans and offering personalized discounts or incentives may encourage them to stay with the telecommunications company.
- Monitoring usage patterns: Monitoring total day minutes and charges can help identify customers who are using the service extensively, potentially indicating dissatisfaction or a need for alternative plans. Offering tailored solutions or upgrades may help retain these customers.
- Targeted communication: Utilizing the insights from the predictive model, targeted communication strategies can be implemented to reach out to customers at high risk of churn. This may involve personalized outreach campaigns, targeted promotions, or loyalty programs aimed at retaining these customers.
- Feedback mechanisms: Implementing effective feedback mechanisms to gather insights from churned customers can help identify underlying issues and inform strategies for continuous improvement.