Comparative Analysis of Logistic Regression and Decision Tree Models for Predicting Customer Churn

Author: Keziah Gicheha.

Overview

In this project, I developed and evaluated two machine learning models—Logistic Regression and Decision Tree—to predict customer churn. The primary goal was to identify patterns and features that contribute to customer attrition, enabling the business to take proactive measures to retain customers.

I began by building a baseline logistic regression model, which showed reasonable accuracy but exhibited limitations in recall, indicating that it struggled to correctly identify customers who would churn. To address this, we introduced a decision tree classifier, which offered a more balanced performance, particularly in terms of recall, and provided clearer insights into the key features driving churn.

By comparing the results of both models, I was able to recommend the most effective approach for predicting customer churn, considering both the accuracy and interpretability of the models. This analysis is intended to guide the business in implementing targeted retention strategies, while also identifying areas for further model refinement.

Business Problem

SyriaTel, a telecommunications company, is facing the challenge of customer churn — customers leaving the service, resulting in lost revenue. Retaining customers is crucial for the company's profitability and growth, as acquiring new customers often costs more than keeping existing ones. The company needs to identify customers who are likely to stop doing business with them so that targeted retention strategies can be implemented.

Data Analysis Questions.

1. What are the factors that indicate the churn of customers? I will identify the features or behaviors that strongly corelate with customers leaving the service which will help the business to understand the root causes of the churn
2. Can it be predicted whether a customer will churn in the near future? By building a predictive model, the company can proactively identify at-risk customers and take action to retain them before they leave.
3. What is the expected accuracy of the churn prediction model? Understanding the model's accuracy helps the business gauge the reliability of the predictions and determine if the model is fit for decision-making.
4. What customer segments are most at risk of churning? Segmenting customers based on their churn risk allows for more personalized and effective retention strategies.

By understanding and predicting churn, SyriaTel can reduce revenue losses, improve customer satisfaction, and allocate resources more efficiently toward retention efforts. This proactive approach can significantly impact the company's bottom line and competitive edge in the market.

Data Understanding

The data for this project has been sourced from Kaggles Website and it relates to the data analysis questions since it has the information that is needed to predict the churn of customers in the company. The data represent customer information for a telecommunications company, specifically focused on customer churn behavior. The sample includes individual customers, with each row representing a single customer. The variables included in the dataset capture a variety of customer attributes and usage patterns, which are crucial for predicting whether a customer is likely to churn.

The variables are as follows.

Customer Demographics:

- **Area Code: The geographical area code where the customer is located.**
- **International Plan: Whether the customer has an international calling plan (Yes/No).**
- **Voice Mail Plan: Whether the customer has a voicemail plan (Yes/No). Usage Statistics:**
- **Total Day Minutes: The total number of minutes the customer used during the day.**
- **Total Day Calls: The total number of calls the customer made during the day.**
- **Total Day Charge: The total charge for day usage.**
- **Total Eve Minutes: The total number of minutes the customer used during the evening.**
- **Total Eve Calls: The total number of calls the customer made during the evening.**
- **Total Eve Charge: The total charge for evening usage.**
- **Total Night Minutes: The total number of minutes the customer used during the night.**
- **Total Night Calls: The total number of calls the customer made during the night.**
- **Total Night Charge: The total charge for night usage.**
- **Total Intl Minutes: The total number of minutes the customer used for international calls.**
- **Total Intl Calls: The total number of international calls the customer made.**
- **Total Intl Charge: The total charge for international usage.**

Customer Service Interaction:

- **Customer Service Calls: The number of calls the customer made to customer service.**

Target Variable:

- **Churn: Indicates whether the customer churned (True) or stayed (False).**

In [334]:

```
# Import standard packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

# Import necessary functions
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import confusion_matrix

from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, confusion_matrix, precision_score, rec
all_score, f1_score, roc_curve, auc
from sklearn.metrics import accuracy_score
from sklearn import tree
```

In [335]:

```
#Load the dataset

st_data = pd.read_csv('Data\data.csv', index_col= False)
st_data.head(5)
```

Out[335]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	ch
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	ch
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	

5 rows × 21 columns

In [336]:

```
#Describe the data
st_data.describe()
```

Out[336]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge
count	3333.000000	3333.000000	3333.000000	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	200.980348	100.114311	17.083540
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000

In [337]:

```
st_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
```

In [338]:

```
#identify unique values
# Check unique values for each column
for column in st_data.columns:
    unique_values = st_data[column].unique()
    num_unique_values = len(unique_values)
    print(f"Column '{column}' has {num_unique_values} unique values.")
    print(unique_values)
    print("\n")
```

```
Column 'state' has 51 unique values.
['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']


Column 'account length' has 212 unique values.
[128 107 137  84  75 118 121 147 117 141  65  74 168  95  62 161  85  93
  76  73  77 130 111 132 174  57  54  20  49 142 172  12  72  36  78 136
 149  98 135  34 160  64  59 119  97  52  60  10  96  87  81  68 125 116
  38  40  43 113 126 150 138 162  90  50  82 144  46  70  55 106  94 155
  80 104  99 120 108 122 157 103  63 112  41 193  61  92 131 163  91 127
 110 140  83 145  56 151 139   6 115 146 185 148  32  25 179  67  19 170
 164  51 208  53 105  66  86  35  88 123  45 100 215  22  33 114  24 101
 143  48  71 167  89 199 166 158 196 209  16  39 173 129  44  79  31 124
  37 159 194 154  21 133 224  58  11 109 102 165  18  30 176  47 190 152
  26  69 186 171  28 153 169  13  27   3  42 189 156 134 243  23   1 205
 200   5   9 178 181 182 217 177 210  29 180   2  17   7 212 232 192 195
 197 225 184 191 201  15 183 202   8 175   4 188 204 221]


Column 'area code' has 3 unique values.
[415 408 510]


Column 'phone number' has 3333 unique values.
['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']


Column 'international plan' has 2 unique values.
['no' 'yes']


Column 'voice mail plan' has 2 unique values.
['yes' 'no']


Column 'number vmail messages' has 46 unique values.
[25 26  0 24 37 27 33 39 30 41 28 34 46 29 35 21 32 42 36 22 23 43 31 38
 40 48 18 17 45 16 20 14 19 51 15 11 12 47  8 44 49  4 10 13 50  9]


Column 'total day minutes' has 1667 unique values.
[265.1 161.6 243.4 ... 321.1 231.1 180.8]


Column 'total day calls' has 119 unique values.
[110 123 114  71 113  98  88  79  97  84 137 127  96  70  67 139  66  90
 117  89 112 103  86  76 115  73 109  95 105 121 118  94  80 128  64 106
 102  85  82  77 120 133 135 108  57  83 129  91  92  74  93 101 146  72
  99 104 125  61 100  87 131  65 124 119  52  68 107  47 116 151 126 122
 111 145  78 136 140 148  81  55  69 158 134 130  63  53  75 141 163  59
 132 138  54  58  62 144 143 147  36  40 150  56  51 165  30  48  60  42
   0  45 160 149 152 142 156  35  49 157  44]


Column 'total day charge' has 1667 unique values.
[45.07 27.47 41.38 ... 54.59 39.29 30.74]


Column 'total eve minutes' has 1611 unique values.
[197.4 195.5 121.2 ... 153.4 288.8 265.9]
```

[197.4 195.5 121.2 ... 153.4 288.8 265.9]

Column 'total eve calls' has 123 unique values.
[99 103 110 88 122 101 108 94 80 111 83 148 71 75 76 97 90 65
  93 121 102 72 112 100 84 109 63 107 115 119 116 92 85 98 118 74
 117 58 96 66 67 62 77 164 126 142 64 104 79 95 86 105 81 113
 106 59 48 82 87 123 114 140 128 60 78 125 91 46 138 129 89 133
 136 57 135 139 51 70 151 137 134 73 152 168 68 120 69 127 132 143
  61 124 42 54 131 52 149 56 37 130 49 146 147 55 12 50 157 155
  45 144 36 156 53 141 44 153 154 150 43   0 145 159 170]

Column 'total eve charge' has 1440 unique values.
[16.78 16.62 10.3 ... 13.04 24.55 22.6]

Column 'total night minutes' has 1591 unique values.
[244.7 254.4 162.6 ... 280.9 120.1 279.1]

Column 'total night calls' has 120 unique values.
[91 103 104 89 121 118 96 90 97 111 94 128 115 99 75 108 74 133
  64 78 105 68 102 148 98 116 71 109 107 135 92 86 127 79 87 129
  57 77 95 54 106 53 67 139 60 100 61 73 113 76 119 88 84 62
 137 72 142 114 126 122 81 123 117 82 80 120 130 134 59 112 132 110
 101 150 69 131 83 93 124 136 125 66 143 58 55 85 56 70 46 42
 152 44 145 50 153 49 175 63 138 154 140 141 146 65 51 151 158 155
 157 147 144 149 166 52 33 156 38 36 48 164]

Column 'total night charge' has 933 unique values.
[11.01 11.45 7.32 8.86 8.41 9.18 9.57 9.53 9.71 14.69 9.4   8.82
  6.35 8.65 9.14 7.23 4.02 5.83 7.46 8.68 9.43 8.18 8.53 10.67
 11.28 8.22 4.59 8.17 8.04 11.27 11.08 13.2 12.61 9.61 6.88 5.82
 10.25 4.58 8.47 8.45 5.5 14.02 8.03 11.94 7.34 6.06 10.9   6.44
  3.18 10.66 11.21 12.73 10.28 12.16 6.34 8.15 5.84 8.52 7.5   7.48
  6.21 11.95 7.15 9.63 7.1   6.91 6.69 13.29 11.46 7.76 6.86 8.16
 12.15 7.79 7.99 10.29 10.08 12.53 7.91 10.02 8.61 14.54 8.21 9.09
  4.93 11.39 11.88 5.75 7.83 8.59 7.52 12.38 7.21 5.81 8.1 11.04
 11.19 8.55 8.42 9.76 9.87 10.86 5.36 10.03 11.15 9.51 6.22 2.59
  7.65 6.45 9.   6.4   9.94 5.08 10.23 11.36 6.97 10.16 7.88 11.91
  6.61 11.55 11.76 9.27 9.29 11.12 10.69 8.8 11.85 7.14 8.71 11.42
  4.94 9.02 11.22 4.97 9.15 5.45 7.27 12.91 7.75 13.46 6.32 12.13
 11.97 6.93 11.66 7.42 6.19 11.41 10.33 10.65 11.92 4.77 4.38 7.41
 12.1   7.69 8.78 9.36 9.05 12.7   6.16 6.05 10.85 8.93 3.48 10.4
  5.05 10.71 9.37 6.75 8.12 11.77 11.49 11.06 11.25 11.03 10.82 8.91
  8.57 8.09 10.05 11.7 10.17 8.74 5.51 11.11 3.29 10.13 6.8   8.49
  9.55 11.02 9.91 7.84 10.62 9.97 3.44 7.35 9.79 8.89 8.14 6.94
 10.49 10.57 10.2   6.29 8.79 10.04 12.41 15.97 9.1 11.78 12.75 11.07
 12.56 8.63 8.02 10.42 8.7   9.98 7.62 8.33 6.59 13.12 10.46 6.63
  8.32 9.04 9.28 10.76 9.64 11.44 6.48 10.81 12.66 11.34 8.75 13.05
 11.48 14.04 13.47 5.63 6.6   9.72 11.68 6.41 9.32 12.95 13.37 9.62
  6.03 8.25 8.26 11.96 9.9   9.23 5.58 7.22 6.64 12.29 12.93 11.32
  6.85 8.88 7.03 8.48 3.59 5.86 6.23 7.61 7.66 13.63 7.9 11.82
  7.47 6.08 8.4   5.74 10.94 10.35 10.68 4.34 8.73 5.14 8.24 9.99
 13.93 8.64 11.43 5.79 9.2 10.14 12.11 7.53 12.46 8.46 8.95 9.84
 10.8 11.23 10.15 9.21 14.46 6.67 12.83 9.66 9.59 10.48 8.36 4.84
 10.54 8.39 7.43 9.06 8.94 11.13 8.87 8.5   7.6 10.73 9.56 10.77
  7.73 3.47 11.86 8.11 9.78 9.42 9.65 7.   7.39 9.88 6.56 5.92
  6.95 15.71 8.06 4.86 7.8   8.58 10.06 5.21 6.92 6.15 13.49 9.38
 12.62 12.26 8.19 11.65 11.62 10.83 7.92 7.33 13.01 13.26 12.22 11.58
  5.97 10.99 8.38 9.17 8.08 5.71 3.41 12.63 11.79 12.96 7.64 6.58
 10.84 10.22 6.52 5.55 7.63 5.11 5.89 10.78 3.05 11.89 8.97 10.44
 10.5   9.35 5.66 11.09 9.83 5.44 10.11 6.39 11.93 8.62 12.06 6.02
  8.85 5.25 8.66 6.73 10.21 11.59 13.87 7.77 10.39 5.54 6.62 13.33
  6.24 12.59 6.3   6.79 8.28 9.03 8.07 5.52 12.14 10.59 7.54 7.67
  5.47 8.81 8.51 13.45 8.77 6.43 12.01 12.08 7.07 6.51 6.84 9.48
 13.78 11.54 11.67 8.13 10.79 7.13 4.72 4.64 8.96 13.03 6.07 3.51
  6.83 6.12 9.31 9.58 4.68 5.32 9.26 11.52 9.11 10.55 11.47 9.3
 13.82 8.44 5.77 10.96 11.74 8.9 10.47 7.85 10.92 4.74 9.74 10.43
  9.96 10.18 9.54 7.89 12.36 8.54 10.07 9.46 7.3 11.16 9.16 10.19
  5.99 10.88 5.8   7.19 4.55 8.31 8.01 14.43 8.3 14.3   6.53 8.2

```
 5.99 10.66  5.8   7.19  4.55  8.31  8.01 14.45  8.5  14.5   6.55  8.2
11.31 13.    6.42  4.24  7.44  7.51 13.1   9.49  6.14  8.76  6.65 10.56
 6.72  8.29 12.09  5.39  2.96  7.59  7.24  4.28  9.7   8.83 13.3  11.37
 9.33  5.01  3.26 11.71  8.43  9.68 15.56  9.8   3.61  6.96 11.61 12.81
10.87 13.84  5.03  5.17  2.03 10.34  9.34  7.95 10.09  9.95  7.11  9.22
 6.13 11.05  9.89  9.39 14.06 10.26 13.31 15.43 16.39  6.27 10.64 11.5
12.48  8.27 13.53 10.36 12.24  8.69 10.52  9.07 11.51  9.25  8.72  6.78
 8.6  11.84  5.78  5.85 12.3   5.76 12.07  9.6   8.84 12.39 10.1   9.73
 2.85  6.66  2.45  5.28 11.73 10.75  7.74  6.76  6.    7.58 13.69  7.93
 7.68  9.75  4.96  5.49 11.83  7.18  9.19  7.7   7.25 10.74  4.27 13.8
 9.12  4.75  7.78 11.63  7.55  2.25  9.45  9.86  7.71  4.95  7.4  11.17
11.33  6.82 13.7   1.97 10.89 12.77 10.31  5.23  5.27  9.41  6.09 10.61
 7.29  4.23  7.57  3.67 12.69 14.5   5.95  7.87  5.96  5.94 12.23  4.9
12.33  6.89  9.67 12.68 12.87  3.7   6.04 13.13 15.74 11.87  4.7   4.67
 7.05  5.42  4.09  5.73  9.47  8.05  6.87  3.71 15.86  7.49 11.69  6.46
10.45 12.9   5.41 11.26  1.04  6.49  6.37 12.21  6.77 12.65  7.86  9.44
 4.3   7.38  5.02 10.63  2.86 17.19  8.67  8.37  6.9  10.93 10.38  7.36
10.27 10.95  6.11  4.45 11.9  15.01 12.84  7.45  6.98 11.72  7.56 11.38
10.    4.42  9.81  5.56  6.01 10.12 12.4  16.99  5.68 11.64  3.78  7.82
 9.85 13.74 12.71 10.98 10.01  9.52  7.31  8.35 11.35  9.5  14.03  3.2
 7.72 13.22 10.7   8.99 10.6  13.02  9.77 12.58 12.35 12.2  11.4  13.91
 3.57 14.65 12.28  5.13 10.72 12.86 14.    7.12 12.17  4.71  6.28  8.
 7.01  5.91  5.2  12.   12.02 12.88  7.28  5.4  12.04  5.24 10.3  10.41
13.41 12.72  9.08  7.08 13.5   5.35 12.45  5.3  10.32  5.15 12.67  5.22
 5.57  3.94  4.41 13.27 10.24  4.25 12.89  5.72 12.5  11.29  3.25 11.53
 9.82  7.26  4.1  10.37  4.98  6.74 12.52 14.56  8.34  3.82  3.86 13.97
11.57  6.5  13.58 14.32 13.75 11.14 14.18  9.13  4.46  4.83  9.69 14.13
 7.16  7.98 13.66 14.78 11.2   9.93 11.    5.29  9.92  4.29 11.1  10.51
12.49  4.04 12.94  7.09  6.71  7.94  5.31  5.98  7.2  14.82 13.21 12.32
10.58  4.92  6.2   4.47 11.98  6.18  7.81  4.54  5.37  7.17  5.33 14.1
 5.7  12.18  8.98  5.1  14.67 13.95 16.55 11.18  4.44  4.73  2.55  6.31
 2.43  9.24  7.37 13.42 12.42 11.8  14.45  2.89 13.23 12.6  13.18 12.19
14.81  6.55 11.3  12.27 13.98  8.23 15.49  6.47 13.48 13.59 13.25 17.77
13.9   3.97 11.56 14.08 13.6   6.26  4.61 12.76 15.76  6.38  3.6  12.8
 5.9   7.97  5.   10.97  5.88 12.34 12.03 14.97 15.06 12.85  6.54 11.24
12.64  7.06  5.38 13.14  3.99  3.32  4.51  4.12  3.93  2.4  11.75  4.03
15.85  6.81 14.25 14.09 16.42  6.7  12.74  2.76 12.12  6.99  6.68 11.81
 7.96  5.06 13.16  2.13 13.17  5.12  5.65 12.37 10.53]


Column 'total intl minutes' has 162 unique values.
[10.  13.7 12.2  6.6 10.1  6.3  7.5  7.1  8.7 11.2 12.7  9.1 12.3 13.1
  5.4 13.8  8.1 13.  10.6  5.7  9.5  7.7 10.3 15.5 14.7 11.1 14.2 12.6
 11.8  8.3 14.5 10.5  9.4 14.6  9.2  3.5  8.5 13.2  7.4  8.8 11.   7.8
  6.8 11.4  9.3  9.7 10.2  8.   5.8 12.1 12.  11.6  8.2  6.2  7.3  6.1
 11.7 15.   9.8 12.4  8.6 10.9 13.9  8.9  7.9  5.3  4.4 12.5 11.3  9.
  9.6 13.3 20.   7.2  6.4 14.1 14.3  6.9 11.5 15.8 12.8 16.2  0.  11.9
  9.9  8.4 10.8 13.4 10.7 17.6  4.7  2.7 13.5 12.9 14.4 10.4  6.7 15.4
  4.5  6.5 15.6  5.9 18.9  7.6  5.   7.  14.  18.  16.  14.8  3.7  2.
  4.8 15.3  6.  13.6 17.2 17.5  5.6 18.2  3.6 16.5  4.6  5.1  4.1 16.3
 14.9 16.4 16.7  1.3 15.2 15.1 15.9  5.5 16.1  4.  16.9  5.2  4.2 15.7
 17.   3.9  3.8  2.2 17.1  4.9 17.9 17.3 18.4 17.8  4.3  2.9  3.1  3.3
  2.6  3.4  1.1 18.3 16.6  2.1  2.4  2.5]


Column 'total intl calls' has 21 unique values.
[ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13 18 14 16 20 17]


Column 'total intl charge' has 162 unique values.
[2.7  3.7  3.29 1.78 2.73 1.7  2.03 1.92 2.35 3.02 3.43 2.46 3.32 3.54
 1.46 3.73 2.19 3.51 2.86 1.54 2.57 2.08 2.78 4.19 3.97 3.   3.83 3.4
 3.19 2.24 3.92 2.84 2.54 3.94 2.48 0.95 2.3  3.56 2.   2.38 2.97 2.11
 1.84 3.08 2.51 2.62 2.75 2.16 1.57 3.27 3.24 3.13 2.21 1.67 1.97 1.65
 3.16 4.05 2.65 3.35 2.32 2.94 3.75 2.4  2.13 1.43 1.19 3.38 3.05 2.43
 2.59 3.59 5.4  1.94 1.73 3.81 3.86 1.86 3.11 4.27 3.46 4.37 0.   3.21
 2.67 2.27 2.92 3.62 2.89 4.75 1.27 0.73 3.65 3.48 3.89 2.81 1.81 4.16
 1.22 1.76 4.21 1.59 5.1  2.05 1.35 1.89 3.78 4.86 4.32 4.   1.   0.54
 1.3  4.13 1.62 3.67 4.64 4.73 1.51 4.91 0.97 4.46 1.24 1.38 1.11 4.4
 4.02 4.43 4.51 0.35 4.1  4.08 4.29 1.49 4.35 1.08 4.56 1.4  1.13 4.24
 4.59 1.05 1.03 0.59 4.62 1.32 4.83 4.67 4.97 4.81 1.16 0.78 0.84 0.89
 0.7  0.92 0.3  4.94 4.48 0.57 0.65 0.68]
```

```
Column 'customer service calls' has 10 unique values.
[1 0 2 3 4 5 7 9 6 8]


Column 'churn' has 2 unique values.
[False  True]
```

Data Preparation

1. **Data Cleaning. I checked for missing data, which I fould no missing data, Missing values can skew the analysis and lead to incorrect conclusions.**
   - **I also checked for duplicates which there were none for ths dataset. Duplicate data can lead to biased model training and inaccurate results.**
2. **Data Transformation. The dataset included categorical variables such as International Plan, Voice Mail Plan, and Area Code. These were transformed into numerical values using techniques like one-hot encoding or label encoding to make them suitable for machine learning models. Feature Scaling: Features like Total Day Minutes, Total Day Calls, and other numerical variables were scaled to ensure that all features contributed equally to the model.**
3. **Feature Selection Features that were highly correlated with each other, such as Total Day Minutes and Total Day Charge, were analyzed to decide whether both were necessary for the model. In cases where multicollinearity was identified, one of the correlated features was removed to prevent redundancy and improve model interpretability.**
4. **Splitting the data Training and Testing Split: The data was split into training and testing sets, typically using an 70-30 split. The training set was used to train the model, while the testing set was reserved for evaluating the model's performance on unseen data. This helps in assessing the model's generalization capability.**

- **Cross-Validation: To further validate the model, cross-validation techniques like k-fold cross-validation were used. This involved splitting the training data into k subsets, training the model k times, each time using a different subset as the validation set, and the remaining data as the training set. This approach helps in reducing overfitting and provides a more robust estimate of the model's performance.**

In [339]:

```
## Exploratory Data Analysis

# Create a new copy of the dataframe for EDA
# Perform EDA and feature engineering on st_data_eda
st_data_eda = st_data.copy()

#Data Cleaning - check for missing values
st_data_eda.isna().sum()
```

Out[339]:

```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
```

```
customer service calls    0
churn                     0
dtype: int64
```

There are no missing values from this dataset, all rows have values.

In [340]:

```
# Drop 'phone number ' from the EDA dataframe
st_data_eda = st_data_eda.drop(columns=['phone number'])

# Verify the columns have been removed
print(st_data_eda.columns)
```

```
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')
```

In [341]:

```
# Check for duplicates in the dataset
duplicate_rows = st_data_eda[st_data_eda.duplicated()]

# Print the number of duplicate rows
print(f'Number of duplicate rows: {duplicate_rows.shape[0]}')
```

```
Number of duplicate rows: 0
```

The phone number feature seems to be an identifier, hence I removed them from the dataset.

In [342]:

```
#check the first 5 rows with te removed features
st_data_eda.head(5)
```

Out[342]:

	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	tota nigl charg
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.0
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.4
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.3
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.8
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.4

In [343]:

```
# Check the unique values in each column
for column in st_data_eda.columns:
    unique_values = st_data_eda[column].unique()
    print(f"Column: {column}")
    print(f"Unique values: {unique_values}")
    print("\n")
```

```
Column: state
Unique values: ['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']
```

```
Column: account length
Unique values: [128 107 137  84  75 118 121 147 117 141  65  74 168  95  62 161  85  93
  76  73  77 130 111 132 174  57  54  20  49 142 172  12  72  36  78 136
 149  98 135  34 160  64  59 119  97  52  60  10  96  87  81  68 125 116
  38  40  43 113 126 150 138 162  90  50  82 144  46  70  55 106  94 155
  80 104  99 120 108 122 157 103  63 112  41 193  61  92 131 163  91 127
 110 140  83 145  56 151 139   6 115 146 185 148  32  25 179  67  19 170
 164  51 208  53 105  66  86  35  88 123  45 100 215  22  33 114  24 101
 143  48  71 167  89 199 166 158 196 209  16  39 173 129  44  79  31 124
  37 159 194 154  21 133 224  58  11 109 102 165  18  30 176  47 190 152
  26  69 186 171  28 153 169  13  27   3  42 189 156 134 243  23   1 205
 200   5   9 178 181 182 217 177 210  29 180   2  17   7 212 232 192 195
 197 225 184 191 201  15 183 202   8 175   4 188 204 221]


Column: area code
Unique values: [415 408 510]


Column: international plan
Unique values: ['no' 'yes']


Column: voice mail plan
Unique values: ['yes' 'no']


Column: number vmail messages
Unique values: [25 26  0 24 37 27 33 39 30 41 28 34 46 29 35 21 32 42 36 22 23 43 31 38
 40 48 18 17 45 16 20 14 19 51 15 11 12 47  8 44 49  4 10 13 50  9]


Column: total day minutes
Unique values: [265.1 161.6 243.4 ... 321.1 231.1 180.8]


Column: total day calls
Unique values: [110 123 114  71 113  98  88  79  97  84 137 127  96  70  67 139  66  90
 117  89 112 103  86  76 115  73 109  95 105 121 118  94  80 128  64 106
 102  85  82  77 120 133 135 108  57  83 129  91  92  74  93 101 146  72
  99 104 125  61 100  87 131  65 124 119  52  68 107  47 116 151 126 122
 111 145  78 136 140 148  81  55  69 158 134 130  63  53  75 141 163  59
 132 138  54  58  62 144 143 147  36  40 150  56  51 165  30  48  60  42
   0  45 160 149 152 142 156  35  49 157  44]


Column: total day charge
Unique values: [45.07 27.47 41.38 ... 54.59 39.29 30.74]


Column: total eve minutes
Unique values: [197.4 195.5 121.2 ... 153.4 288.8 265.9]


Column: total eve calls
Unique values: [ 99 103 110  88 122 101 108  94  80 111  83 148  71  75  76  97  90  65
  93 121 102  72 112 100  84 109  63 107 115 119 116  92  85  98 118  74
 117  58  96  66  67  62  77 164 126 142  64 104  79  95  86 105  81 113
 106  59  48  82  87 123 114 140 128  60  78 125  91  46 138 129  89 133
 136  57 135 139  51  70 151 137 134  73 152 168  68 120  69 127 132 143
  61 124  42  54 131  52 149  56  37 130  49 146 147  55  12  50 157 155
  45 144  36 156  53 141  44 153 154 150  43   0 145 159 170]


Column: total eve charge
Unique values: [16.78 16.62 10.3  ... 13.04 24.55 22.6 ]


Column: total night minutes
Unique values: [244.7 254.4 162.6 ... 280.9 120.1 279.1]
```

Column: total night calls
Unique values: [91 103 104 89 121 118 96 90 97 111 94 128 115 99 75 108 74 133
  64 78 105 68 102 148 98 116 71 109 107 135 92 86 127 79 87 129
  57 77 95 54 106 53 67 139 60 100 61 73 113 76 119 88 84 62
 137 72 142 114 126 122 81 123 117 82 80 120 130 134 59 112 132 110
 101 150 69 131 83 93 124 136 125 66 143 58 55 85 56 70 46 42
 152 44 145 50 153 49 175 63 138 154 140 141 146 65 51 151 158 155
 157 147 144 149 166 52 33 156 38 36 48 164]

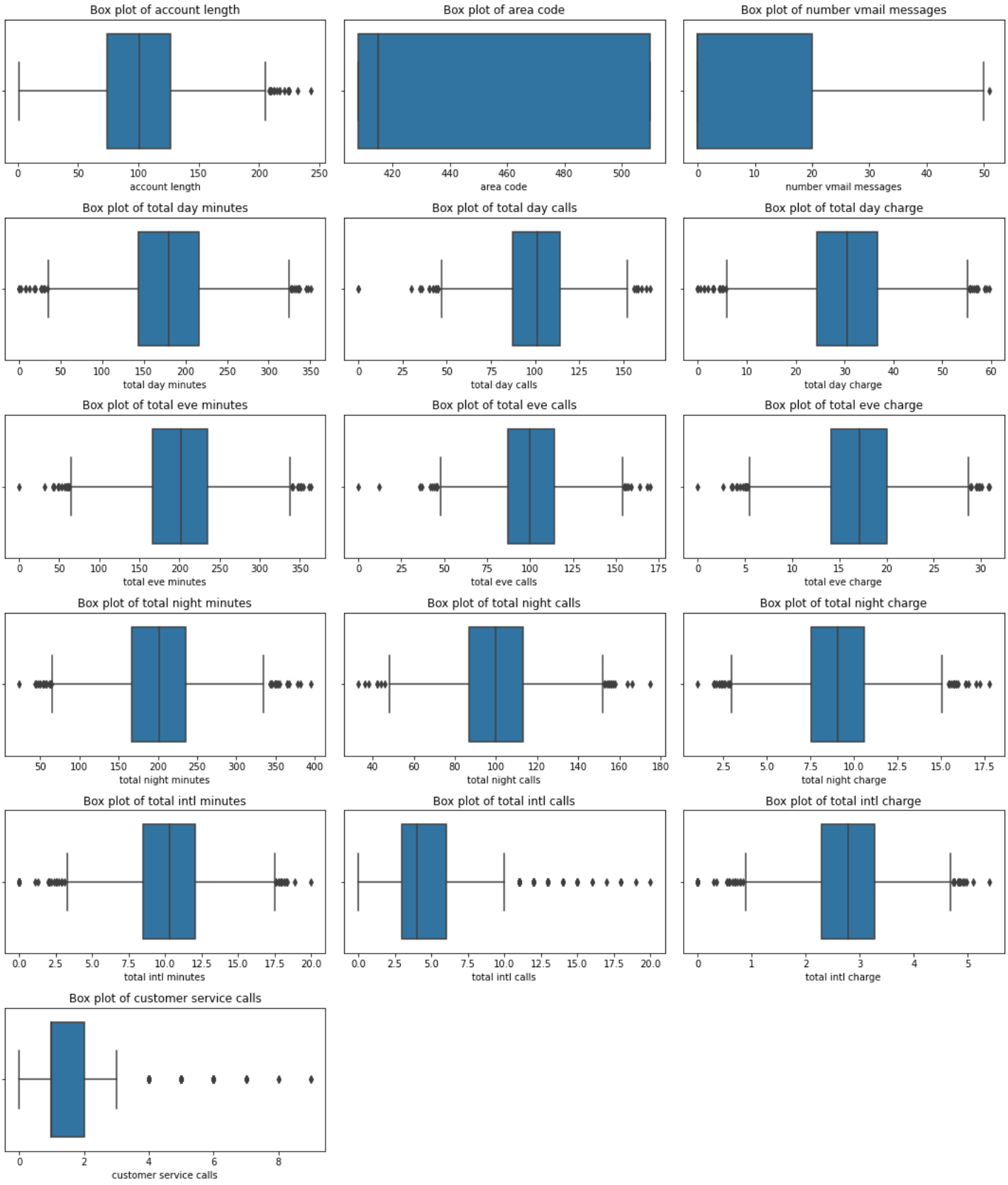
Column: total night charge
Unique values: [11.01 11.45 7.32 8.86 8.41 9.18 9.57 9.53 9.71 14.69 9.4   8.82
  6.35 8.65 9.14 7.23 4.02 5.83 7.46 8.68 9.43 8.18 8.53 10.67
 11.28 8.22 4.59 8.17 8.04 11.27 11.08 13.2 12.61 9.61 6.88 5.82
 10.25 4.58 8.47 8.45 5.5 14.02 8.03 11.94 7.34 6.06 10.9   6.44
  3.18 10.66 11.21 12.73 10.28 12.16 6.34 8.15 5.84 8.52 7.5   7.48
  6.21 11.95 7.15 9.63 7.1   6.91 6.69 13.29 11.46 7.76 6.86 8.16
 12.15 7.79 7.99 10.29 10.08 12.53 7.91 10.02 8.61 14.54 8.21 9.09
  4.93 11.39 11.88 5.75 7.83 8.59 7.52 12.38 7.21 5.81 8.1 11.04
 11.19 8.55 8.42 9.76 9.87 10.86 5.36 10.03 11.15 9.51 6.22 2.59
  7.65 6.45 9.   6.4   9.94 5.08 10.23 11.36 6.97 10.16 7.88 11.91
  6.61 11.55 11.76 9.27 9.29 11.12 10.69 8.8 11.85 7.14 8.71 11.42
  4.94 9.02 11.22 4.97 9.15 5.45 7.27 12.91 7.75 13.46 6.32 12.13
 11.97 6.93 11.66 7.42 6.19 11.41 10.33 10.65 11.92 4.77 4.38 7.41
 12.1   7.69 8.78 9.36 9.05 12.7   6.16 6.05 10.85 8.93 3.48 10.4
  5.05 10.71 9.37 6.75 8.12 11.77 11.49 11.06 11.25 11.03 10.82 8.91
  8.57 8.09 10.05 11.7 10.17 8.74 5.51 11.11 3.29 10.13 6.8   8.49
  9.55 11.02 9.91 7.84 10.62 9.97 3.44 7.35 9.79 8.89 8.14 6.94
 10.49 10.57 10.2   6.29 8.79 10.04 12.41 15.97 9.1 11.78 12.75 11.07
 12.56 8.63 8.02 10.42 8.7   9.98 7.62 8.33 6.59 13.12 10.46 6.63
  8.32 9.04 9.28 10.76 9.64 11.44 6.48 10.81 12.66 11.34 8.75 13.05
 11.48 14.04 13.47 5.63 6.6   9.72 11.68 6.41 9.32 12.95 13.37 9.62
  6.03 8.25 8.26 11.96 9.9   9.23 5.58 7.22 6.64 12.29 12.93 11.32
  6.85 8.88 7.03 8.48 3.59 5.86 6.23 7.61 7.66 13.63 7.9 11.82
  7.47 6.08 8.4   5.74 10.94 10.35 10.68 4.34 8.73 5.14 8.24 9.99
 13.93 8.64 11.43 5.79 9.2 10.14 12.11 7.53 12.46 8.46 8.95 9.84
 10.8 11.23 10.15 9.21 14.46 6.67 12.83 9.66 9.59 10.48 8.36 4.84
 10.54 8.39 7.43 9.06 8.94 11.13 8.87 8.5   7.6 10.73 9.56 10.77
  7.73 3.47 11.86 8.11 9.78 9.42 9.65 7.   7.39 9.88 6.56 5.92
  6.95 15.71 8.06 4.86 7.8   8.58 10.06 5.21 6.92 6.15 13.49 9.38
 12.62 12.26 8.19 11.65 11.62 10.83 7.92 7.33 13.01 13.26 12.22 11.58
  5.97 10.99 8.38 9.17 8.08 5.71 3.41 12.63 11.79 12.96 7.64 6.58
 10.84 10.22 6.52 5.55 7.63 5.11 5.89 10.78 3.05 11.89 8.97 10.44
 10.5   9.35 5.66 11.09 9.83 5.44 10.11 6.39 11.93 8.62 12.06 6.02
  8.85 5.25 8.66 6.73 10.21 11.59 13.87 7.77 10.39 5.54 6.62 13.33
  6.24 12.59 6.3   6.79 8.28 9.03 8.07 5.52 12.14 10.59 7.54 7.67
  5.47 8.81 8.51 13.45 8.77 6.43 12.01 12.08 7.07 6.51 6.84 9.48
 13.78 11.54 11.67 8.13 10.79 7.13 4.72 4.64 8.96 13.03 6.07 3.51
  6.83 6.12 9.31 9.58 4.68 5.32 9.26 11.52 9.11 10.55 11.47 9.3
 13.82 8.44 5.77 10.96 11.74 8.9 10.47 7.85 10.92 4.74 9.74 10.43
  9.96 10.18 9.54 7.89 12.36 8.54 10.07 9.46 7.3 11.16 9.16 10.19
  5.99 10.88 5.8   7.19 4.55 8.31 8.01 14.43 8.3 14.3   6.53 8.2
 11.31 13.   6.42 4.24 7.44 7.51 13.1   9.49 6.14 8.76 6.65 10.56
  6.72 8.29 12.09 5.39 2.96 7.59 7.24 4.28 9.7   8.83 13.3 11.37
  9.33 5.01 3.26 11.71 8.43 9.68 15.56 9.8   3.61 6.96 11.61 12.81
 10.87 13.84 5.03 5.17 2.03 10.34 9.34 7.95 10.09 9.95 7.11 9.22
  6.13 11.05 9.89 9.39 14.06 10.26 13.31 15.43 16.39 6.27 10.64 11.5
 12.48 8.27 13.53 10.36 12.24 8.69 10.52 9.07 11.51 9.25 8.72 6.78
  8.6 11.84 5.78 5.85 12.3   5.76 12.07 9.6   8.84 12.39 10.1   9.73
  2.85 6.66 2.45 5.28 11.73 10.75 7.74 6.76 6.   7.58 13.69 7.93
  7.68 9.75 4.96 5.49 11.83 7.18 9.19 7.7   7.25 10.74 4.27 13.8
  9.12 4.75 7.78 11.63 7.55 2.25 9.45 9.86 7.71 4.95 7.4 11.17
 11.33 6.82 13.7   1.97 10.89 12.77 10.31 5.23 5.27 9.41 6.09 10.61
  7.29 4.23 7.57 3.67 12.69 14.5   5.95 7.87 5.96 5.94 12.23 4.9
 12.33 6.89 9.67 12.68 12.87 3.7   6.04 13.13 15.74 11.87 4.7   4.67
  7.05 5.42 4.09 5.73 9.47 8.05 6.87 3.71 15.86 7.49 11.69 6.46
 10.45 12.9   5.41 11.26 1.04 6.49 6.37 12.21 6.77 12.65 7.86 9.44
  4.3   7.38 5.02 10.63 2.86 17.19 8.67 8.37 6.9 10.93 10.38 7.36
 10.27 10.95 6.11 4.45 11.9 15.01 12.84 7.45 6.98 11.72 7.56 11.38
 10.   4.42 9.81 5.56 6.01 10.12 12.4 16.99 5.68 11.64 3.78 7.82
  9.85 13.74 12.71 10.98 10.01 9.52 7.31 8.35 11.35 9.5 14.03 3.2

```
  7.72 13.22 10.7   8.99 10.6  13.02  9.77 12.58 12.35 12.2  11.4  13.91
  3.57 14.65 12.28  5.13 10.72 12.86 14.    7.12 12.17  4.71  6.28  8.
  7.01  5.91  5.2  12.   12.02 12.88  7.28  5.4  12.04  5.24 10.3  10.41
 13.41 12.72  9.08  7.08 13.5   5.35 12.45  5.3  10.32  5.15 12.67  5.22
  5.57  3.94  4.41 13.27 10.24  4.25 12.89  5.72 12.5  11.29  3.25 11.53
  9.82  7.26  4.1  10.37  4.98  6.74 12.52 14.56  8.34  3.82  3.86 13.97
 11.57  6.5  13.58 14.32 13.75 11.14 14.18  9.13  4.46  4.83  9.69 14.13
  7.16  7.98 13.66 14.78 11.2   9.93 11.    5.29  9.92  4.29 11.1  10.51
 12.49  4.04 12.94  7.09  6.71  7.94  5.31  5.98  7.2  14.82 13.21 12.32
 10.58  4.92  6.2   4.47 11.98  6.18  7.81  4.54  5.37  7.17  5.33 14.1
  5.7  12.18  8.98  5.1  14.67 13.95 16.55 11.18  4.44  4.73  2.55  6.31
  2.43  9.24  7.37 13.42 12.42 11.8  14.45  2.89 13.23 12.6  13.18 12.19
 14.81  6.55 11.3  12.27 13.98  8.23 15.49  6.47 13.48 13.59 13.25 17.77
 13.9   3.97 11.56 14.08 13.6   6.26  4.61 12.76 15.76  6.38  3.6  12.8
  5.9   7.97  5.   10.97  5.88 12.34 12.03 14.97 15.06 12.85  6.54 11.24
 12.64  7.06  5.38 13.14  3.99  3.32  4.51  4.12  3.93  2.4  11.75  4.03
 15.85  6.81 14.25 14.09 16.42  6.7  12.74  2.76 12.12  6.99  6.68 11.81
  7.96  5.06 13.16  2.13 13.17  5.12  5.65 12.37 10.53]


Column: total intl minutes
Unique values: [10.  13.7 12.2  6.6 10.1  6.3  7.5  7.1  8.7 11.2 12.7  9.1 12.3 13.1
  5.4 13.8  8.1 13.  10.6  5.7  9.5  7.7 10.3 15.5 14.7 11.1 14.2 12.6
 11.8  8.3 14.5 10.5  9.4 14.6  9.2  3.5  8.5 13.2  7.4  8.8 11.   7.8
  6.8 11.4  9.3  9.7 10.2  8.   5.8 12.1 12.  11.6  8.2  6.2  7.3  6.1
 11.7 15.   9.8 12.4  8.6 10.9 13.9  8.9  7.9  5.3  4.4 12.5 11.3  9.
  9.6 13.3 20.   7.2  6.4 14.1 14.3  6.9 11.5 15.8 12.8 16.2  0.  11.9
  9.9  8.4 10.8 13.4 10.7 17.6  4.7  2.7 13.5 12.9 14.4 10.4  6.7 15.4
  4.5  6.5 15.6  5.9 18.9  7.6  5.   7.  14.  18.  16.  14.8  3.7  2.
  4.8 15.3  6.  13.6 17.2 17.5  5.6 18.2  3.6 16.5  4.6  5.1  4.1 16.3
 14.9 16.4 16.7  1.3 15.2 15.1 15.9  5.5 16.1  4.  16.9  5.2  4.2 15.7
 17.   3.9  3.8  2.2 17.1  4.9 17.9 17.3 18.4 17.8  4.3  2.9  3.1  3.3
  2.6  3.4  1.1 18.3 16.6  2.1  2.4  2.5]


Column: total intl calls
Unique values: [ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13 18 14 16 20 17]


Column: total intl charge
Unique values: [2.7  3.7  3.29 1.78 2.73 1.7  2.03 1.92 2.35 3.02 3.43 2.46 3.32 3.54
 1.46 3.73 2.19 3.51 2.86 1.54 2.57 2.08 2.78 4.19 3.97 3.   3.83 3.4
 3.19 2.24 3.92 2.84 2.54 3.94 2.48 0.95 2.3  3.56 2.   2.38 2.97 2.11
 1.84 3.08 2.51 2.62 2.75 2.16 1.57 3.27 3.24 3.13 2.21 1.67 1.97 1.65
 3.16 4.05 2.65 3.35 2.32 2.94 3.75 2.4  2.13 1.43 1.19 3.38 3.05 2.43
 2.59 3.59 5.4  1.94 1.73 3.81 3.86 1.86 3.11 4.27 3.46 4.37 0.   3.21
 2.67 2.27 2.92 3.62 2.89 4.75 1.27 0.73 3.65 3.48 3.89 2.81 1.81 4.16
 1.22 1.76 4.21 1.59 5.1  2.05 1.35 1.89 3.78 4.86 4.32 4.   1.   0.54
 1.3  4.13 1.62 3.67 4.64 4.73 1.51 4.91 0.97 4.46 1.24 1.38 1.11 4.4
 4.02 4.43 4.51 0.35 4.1  4.08 4.29 1.49 4.35 1.08 4.56 1.4  1.13 4.24
 4.59 1.05 1.03 0.59 4.62 1.32 4.83 4.67 4.97 4.81 1.16 0.78 0.84 0.89
 0.7  0.92 0.3  4.94 4.48 0.57 0.65 0.68]


Column: customer service calls
Unique values: [1 0 2 3 4 5 7 9 6 8]


Column: churn
Unique values: [False  True]
```

In [344]:

```
#check for outliers
#use box plots to check for the numerical columns in the dataset

# Set the size of the plots
plt.figure(figsize=(15, 20))
```

```
# Loop through each numerical column and create a box plot
for i, column in enumerate(st_data_eda.select_dtypes(include=['float64', 'int64']).colum
ns, 1):
    plt.subplot(len(st_data_eda.columns) // 3 + 1, 3, i)
    sns.boxplot(x=st_data[column])
    plt.title(f'Box plot of {column}')
    plt.tight_layout()

plt.show()
```



From the box plots above many of the features have outliers, but intuitively choose to keep them to see how the model will perform with the outliers first, before proceeding to remove them.

Univariate Analysis

Analyze individual features (distribution, summary statistics).

In [345]:

```
# Summary statistics for numeric features
summary_stats = st_data_eda.describe()
print(summary_stats)

# Summary statistics for categorical features
categorical_features = ['state', 'area code', 'international plan', 'voice mail plan', '
churn']
for feature in categorical_features:
    print(f"\n{feature}:\n{st_data_eda[feature].value_counts()}")
```

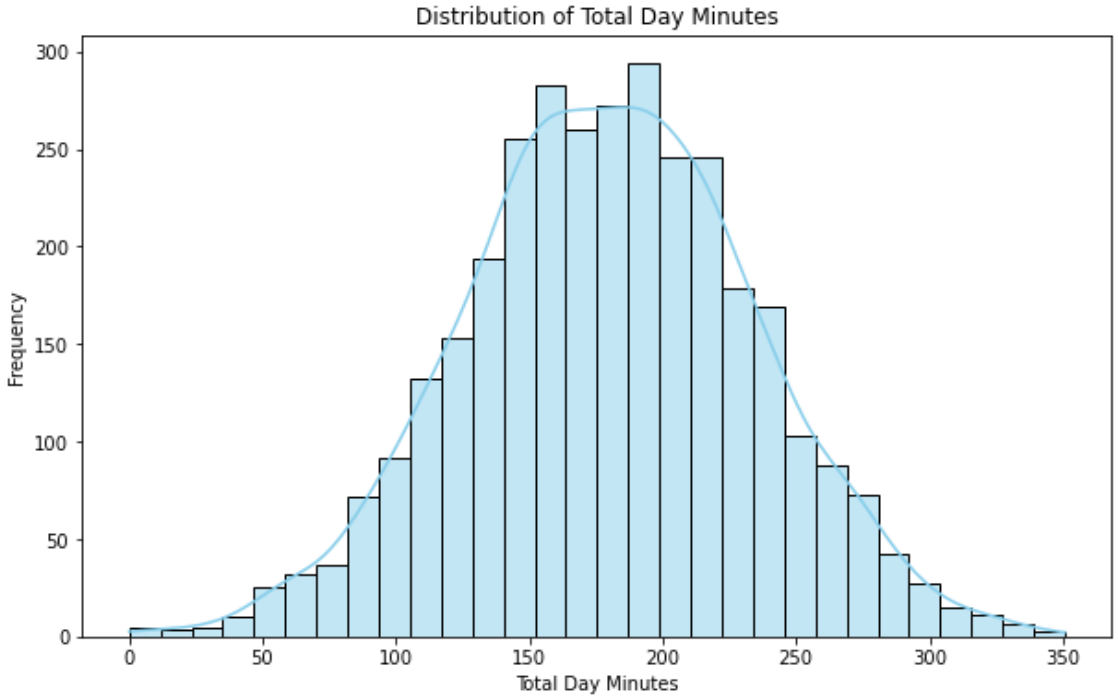
```
       account length    area code  number vmail messages  total day minutes  \
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000

       total day calls  total day charge  total eve minutes  total eve calls  \
count      3333.000000       3333.000000        3333.000000      3333.000000
mean        100.435644         30.562307         200.980348       100.114311
std          20.069084          9.259435          50.713844        19.922625
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.600000        87.000000
50%         101.000000         30.500000         201.400000       100.000000
75%         114.000000         36.790000         235.300000       114.000000
max         165.000000         59.640000         363.700000       170.000000

       total eve charge  total night minutes  total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       total intl charge  customer service calls
count        3333.000000             3333.000000
mean            2.764581                1.562856
std             0.753773                1.315491
min             0.000000                0.000000
25%             2.300000                1.000000
50%             2.780000                1.000000
75%             3.270000                2.000000
max             5.400000                9.000000

state:
WV    106
MN     84
NY     83
AL     80
OH     78
WI     78
OR     78
```

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

area code:
415    1655
510     840
408     838
Name: area code, dtype: int64

international plan:
no     3010
yes     323
Name: international plan, dtype: int64

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

churn:
False    2850
True      483
Name: churn, dtype: int64
```

In [346]:

```
# Distribution of Total Day Minutes
plt.figure(figsize=(10, 6))
```

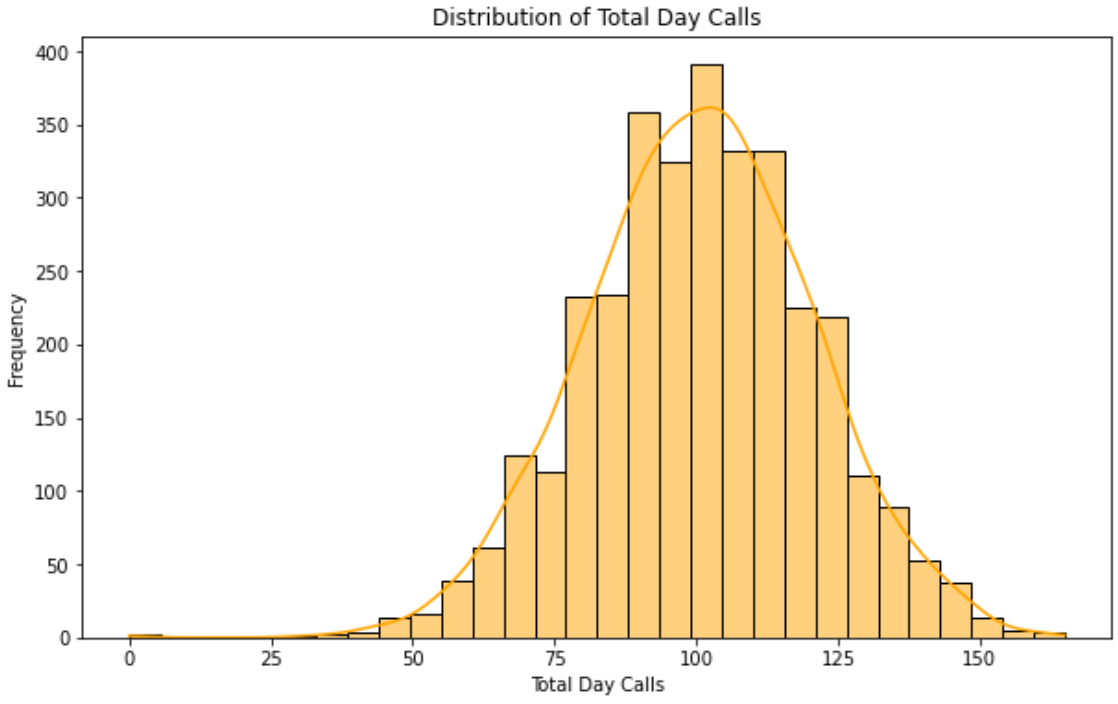
```
sns.histplot(st_data_eda['total day minutes'], bins=30, kde=True, color='skyblue')
plt.title('Distribution of Total Day Minutes')
plt.xlabel('Total Day Minutes')
plt.ylabel('Frequency')
plt.show()
```



There is an observation that the "Total Day minutes" is normally distributed with the average minutes being rough about 180 minutes per day for a user.

In [347]:

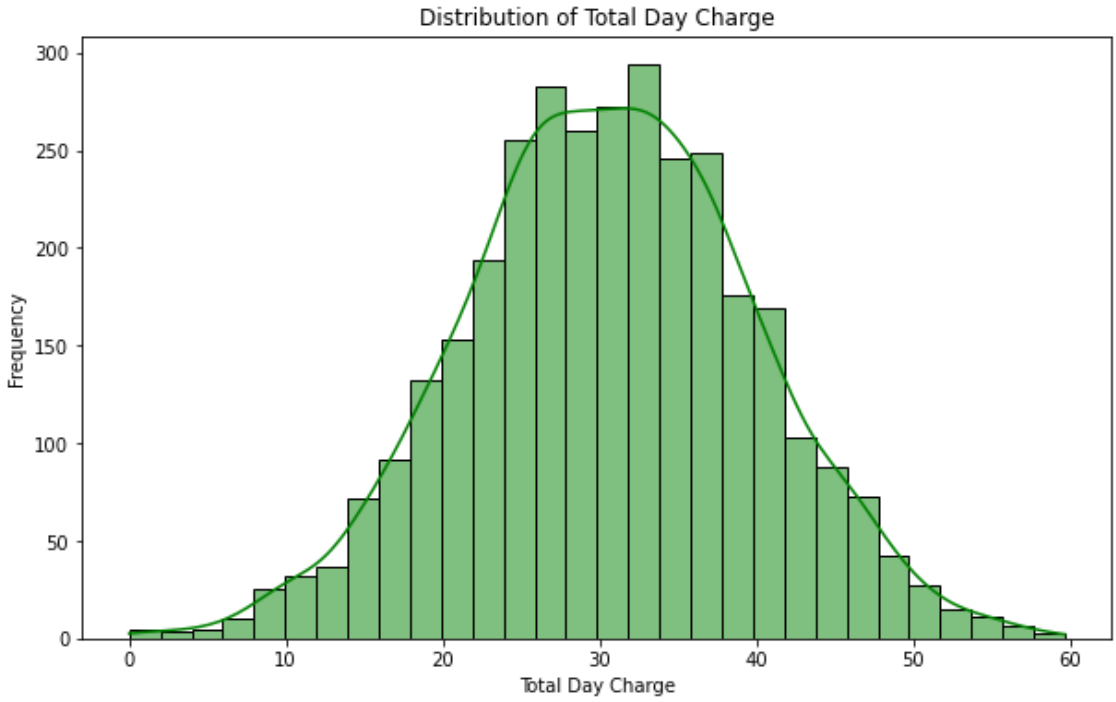
```
# Distribution of Total Day Calls
plt.figure(figsize=(10, 6))
sns.histplot(st_data_eda['total day calls'], bins=30, kde=True, color='orange')
plt.title('Distribution of Total Day Calls')
plt.xlabel('Total Day Calls')
plt.ylabel('Frequency')
plt.show()
```



From the histogram above, "Total day calls" has a slight left skewness.

In [348]:

```
# Distribution of Total Day Charge
plt.figure(figsize=(10, 6))
sns.histplot(st_data_eda['total day charge'], bins=30, kde=True, color='green')
plt.title('Distribution of Total Day Charge')
plt.xlabel('Total Day Charge')
plt.ylabel('Frequency')
plt.show()
```

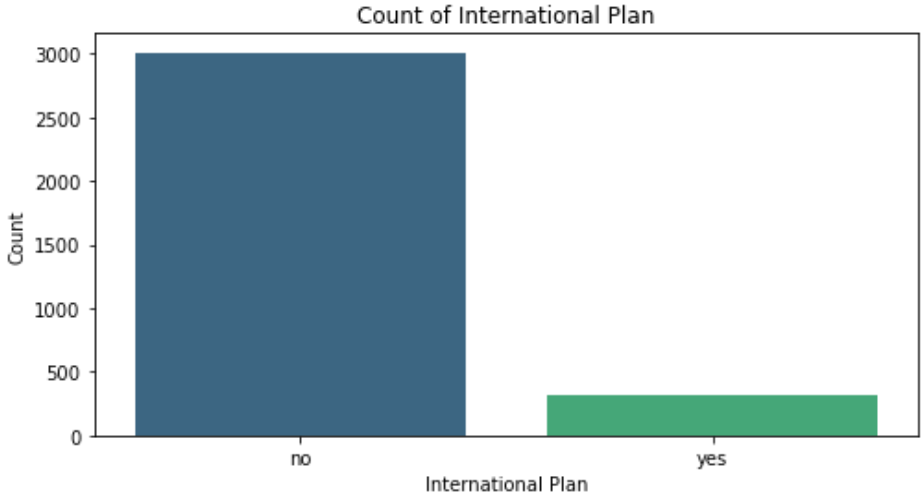


The distribution of the total day charge looks normally distributed, with the highest charge rate approximately 270. No skewness visible.

Categorical Features Analysis

In [349]:

```
# Count plot for International Plan
plt.figure(figsize=(8, 4))
sns.countplot(x='international plan', data=st_data_eda, palette='viridis')
plt.title('Count of International Plan')
plt.xlabel('International Plan')
plt.ylabel('Count')
plt.show()
```



In [350]:

```
st_data_eda["international plan"].value_counts()
```

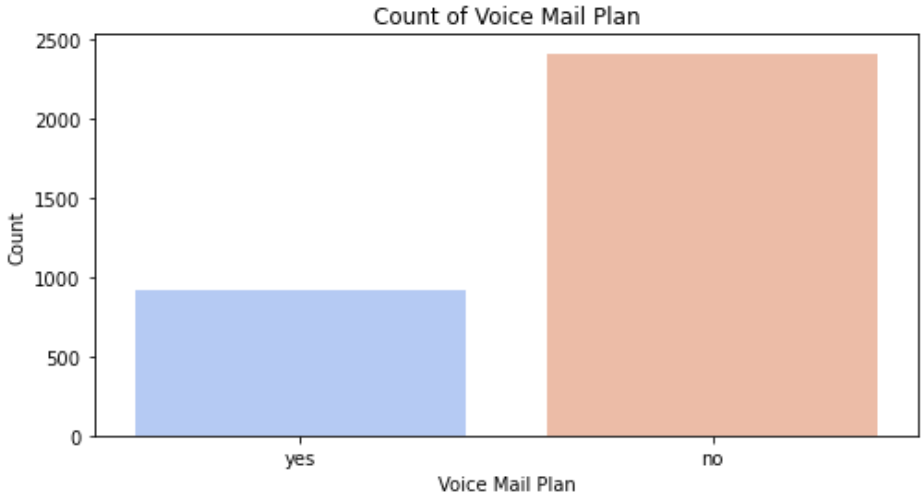
Out[350]:

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

From this graph, 3010 users in this dataset do not have an international plan while 323 users have an international plan.

In [351]:

```
# Count plot for Voice Mail Plan
plt.figure(figsize=(8, 4))
sns.countplot(x='voice mail plan', data=st_data_eda, palette='coolwarm')
plt.title('Count of Voice Mail Plan')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Count')
plt.show()
```



In [352]:

```
st_data_eda['voice mail plan'].value_counts()
```

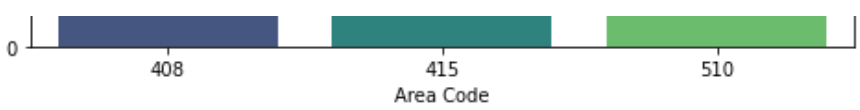
Out[352]:

```
no     2411
yes     922
Name: voice mail plan, dtype: int64
```

2411 users from the dataset do not have a voice mail plan and 922 users have a voice mail plan

In [353]:

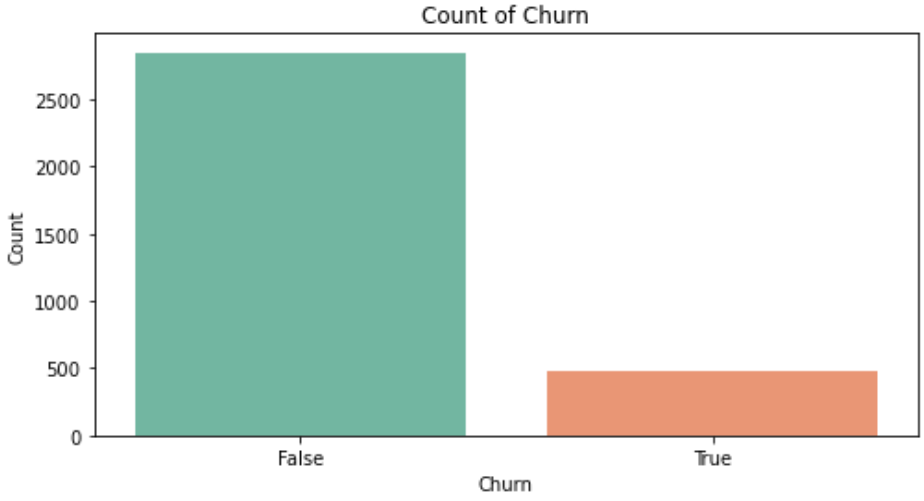
```
# Count plot for Area Code
plt.figure(figsize=(8, 4))
sns.countplot(x='area code', data=st_data_eda, palette='viridis')
plt.title('Count of Area Code')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.show()
```



Approximately 900 users are from the 408 area code, 1500 users from the 415 area code and 800 users from the 510 area code.

In [354]:

```
# Count plot for Churn
plt.figure(figsize=(8, 4))
sns.countplot(x='churn', data=st_data_eda, palette='Set2')
plt.title('Count of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



In [355]:

```
st_data_eda['churn'].value_counts()
```

Out[355]:

```
False    2850
True      483
Name: churn, dtype: int64
```

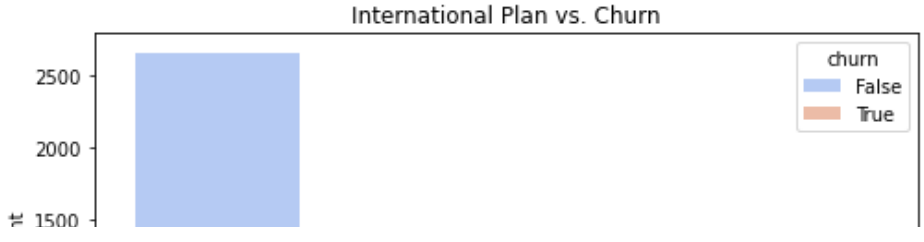
From the dataset, the number of customers who churned was 483 while those who did not was 2850 users.

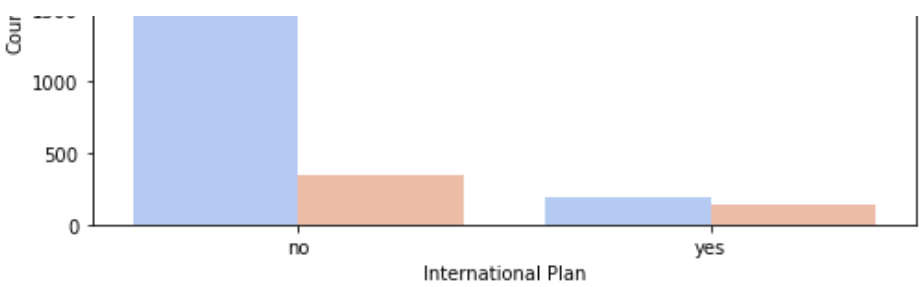
Bivariate Analysis

Analyze the relationship between each feature and the target variable using visualizations (e.g., box plots, bar charts) and summary statistics

In [356]:

```
#International plan vs Churn
plt.figure(figsize=(8, 4))
sns.countplot(x='international plan', hue='churn', data=st_data_eda, palette='coolwarm')
plt.title('International Plan vs. Churn')
plt.xlabel('International Plan')
plt.ylabel('Count')
plt.show()
```

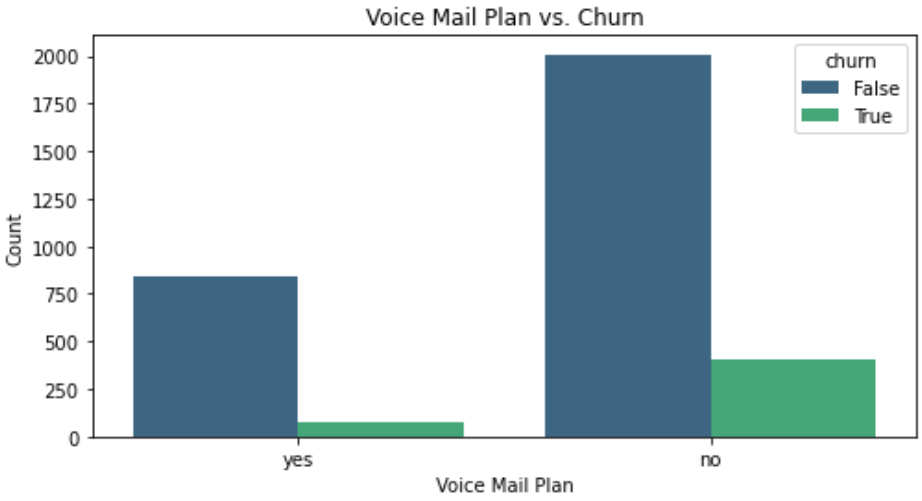




From the graph above, there is a higher number of customers who churned who do not have an international plan compared to who didnt.

In [357]:

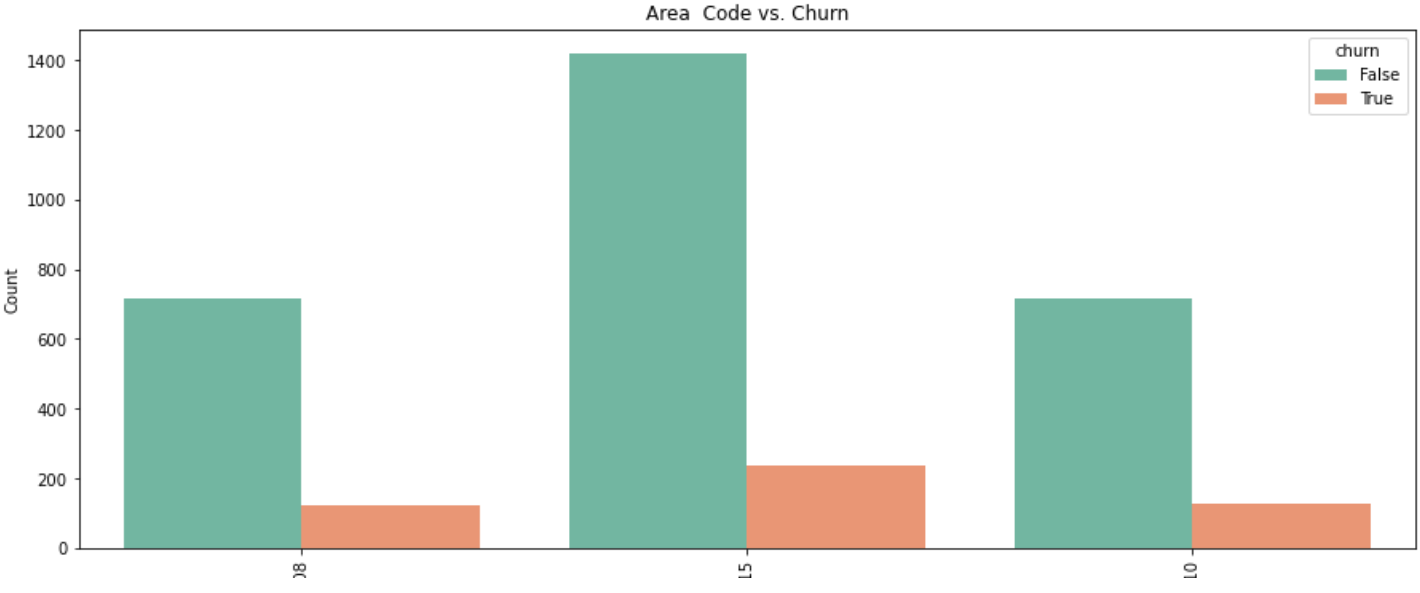
```
plt.figure(figsize=(8, 4))
sns.countplot(x='voice mail plan', hue='churn', data=st_data_eda, palette='viridis')
plt.title('Voice Mail Plan vs. Churn')
plt.xlabel('Voice Mail Plan')
plt.ylabel('Count')
plt.show()
```



Customers without a voicemail plan churned more frequently than those with a voicemail plan.

In [358]:

```
plt.figure(figsize=(15, 6))
sns.countplot(x='area code', hue='churn', data=st_data_eda, palette='Set2')
plt.title('Area  Code vs. Churn')
plt.xlabel('Area Code')
plt.ylabel('Count')
plt.xticks(rotation=90)
plt.show()
```



Area Code

Customers from area code 415 had the highest churn rate, followed by those from area code 510 and area code 408.

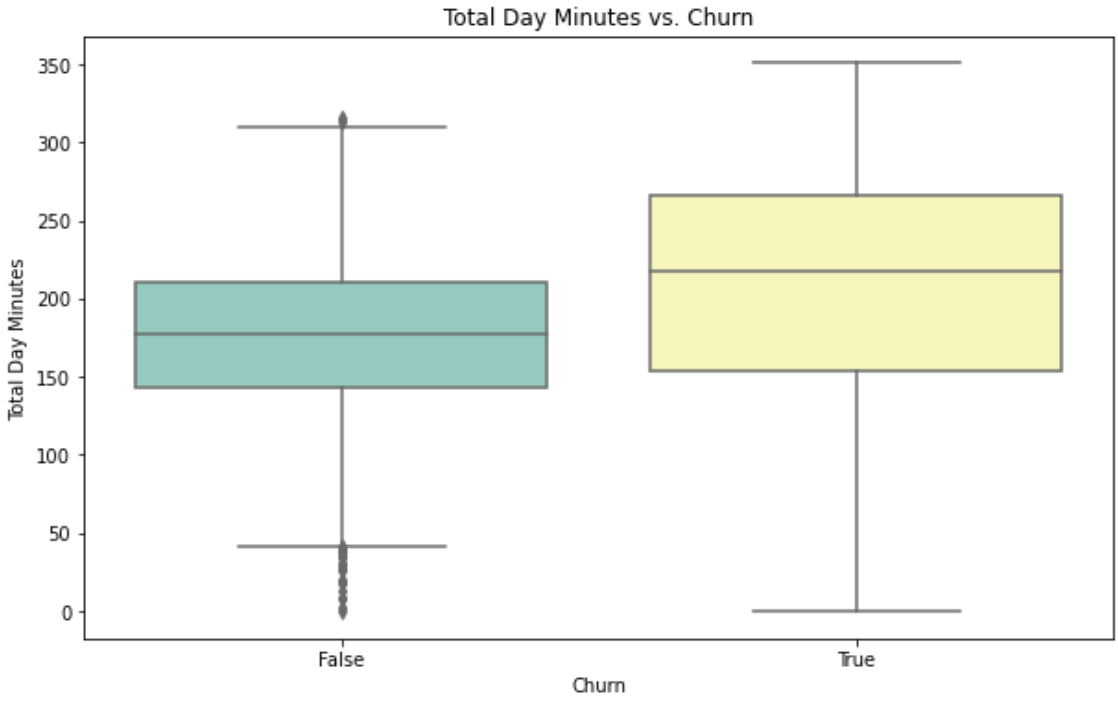
In [359]:

```
# Grouping by churn and calculating summary statistics
grouped_stats = st_data_eda.groupby('churn').mean()
print(grouped_stats)
```

```
       account length   area code  number vmail messages  total day minutes  \
churn
False      100.793684  437.074737               8.604561         175.175754
True       102.664596  437.817805               5.115942         206.914079

       total day calls  total day charge  total eve minutes  total eve calls  \
churn
False       100.283158         29.780421         199.043298       100.038596
True        101.335404         35.175921         212.410145       100.561077

       total eve charge  total night minutes  total night calls  \
churn
False         16.918909           200.133193         100.058246
True          18.054969           205.231677         100.399586

       total night charge  total intl minutes  total intl calls  \
churn
False            9.006074           10.158877          4.532982
True             9.235528           10.700000          4.163561

       total intl charge  customer service calls
churn
False           2.743404                1.449825
True            2.889545                2.229814
```

In [360]:

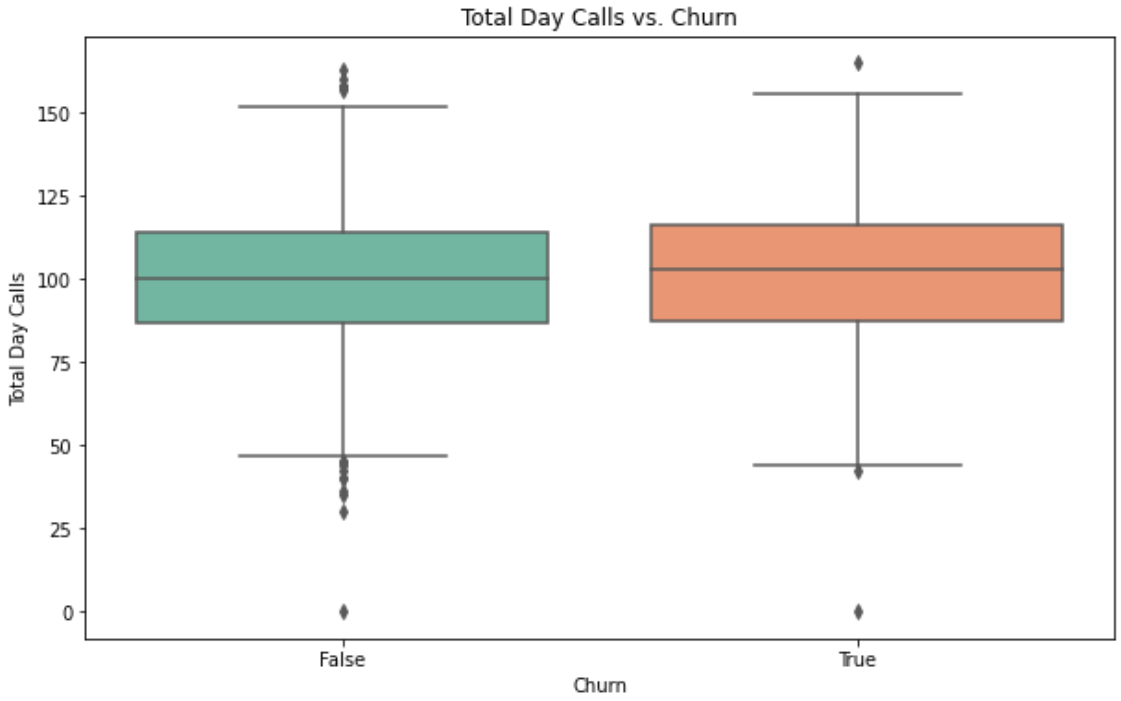
```
#bivariate Analysis for numeric features
plt.figure(figsize=(10, 6))
sns.boxplot(x='churn', y='total day minutes', data=st_data_eda, palette='Set3')
plt.title('Total Day Minutes vs. Churn')
plt.xlabel('Churn')
plt.ylabel('Total Day Minutes')
plt.show()
```



Based on the box plot observations, customers who churned used more day minutes, averaging 220 minutes, compared to 170 minutes for those who did not churn. This suggests that "Total Day Minutes" could be a significant feature in predicting churn

In [361]:

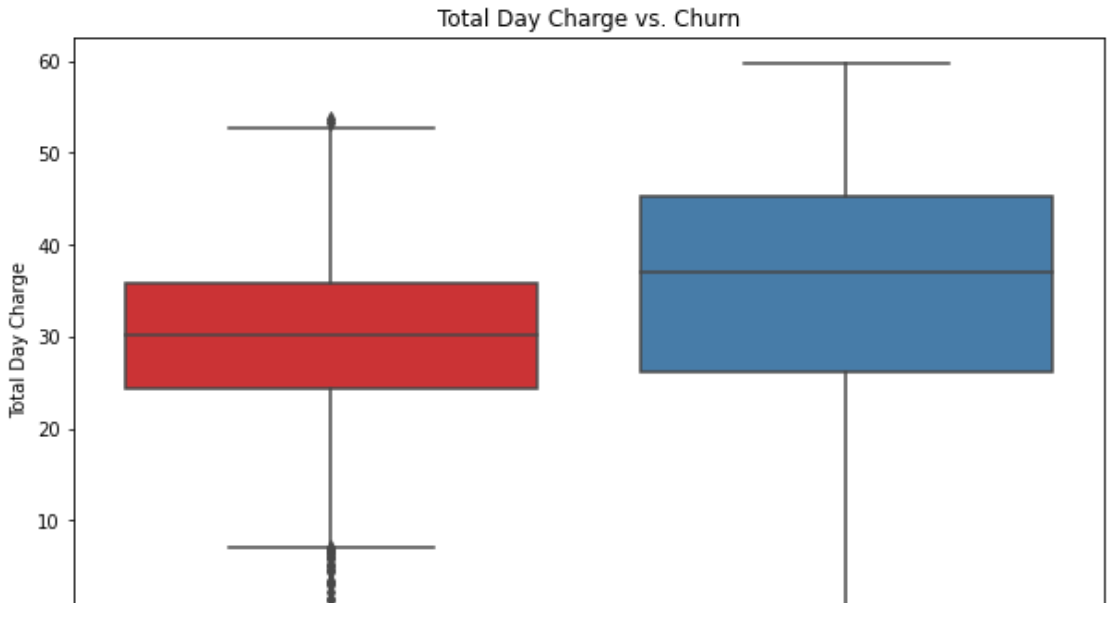
```
#Total Day Calls Vs Churn
plt.figure(figsize=(10, 6))
sns.boxplot(x='churn', y='total day calls', data=st_data_eda, palette='Set2')
plt.title('Total Day Calls vs. Churn')
plt.xlabel('Churn')
plt.ylabel('Total Day Calls')
plt.show()
```



Based on the box plot observation, customers who churned made more day calls, averaging 120 calls, compared to 100 calls for those who did not churn.

In [362]:

```
#Total Day Charge Vs Churn
plt.figure(figsize=(10, 6))
sns.boxplot(x='churn', y='total day charge', data=st_data_eda, palette='Set1')
plt.title('Total Day Charge vs. Churn')
plt.xlabel('Churn')
plt.ylabel('Total Day Charge')
plt.show()
```



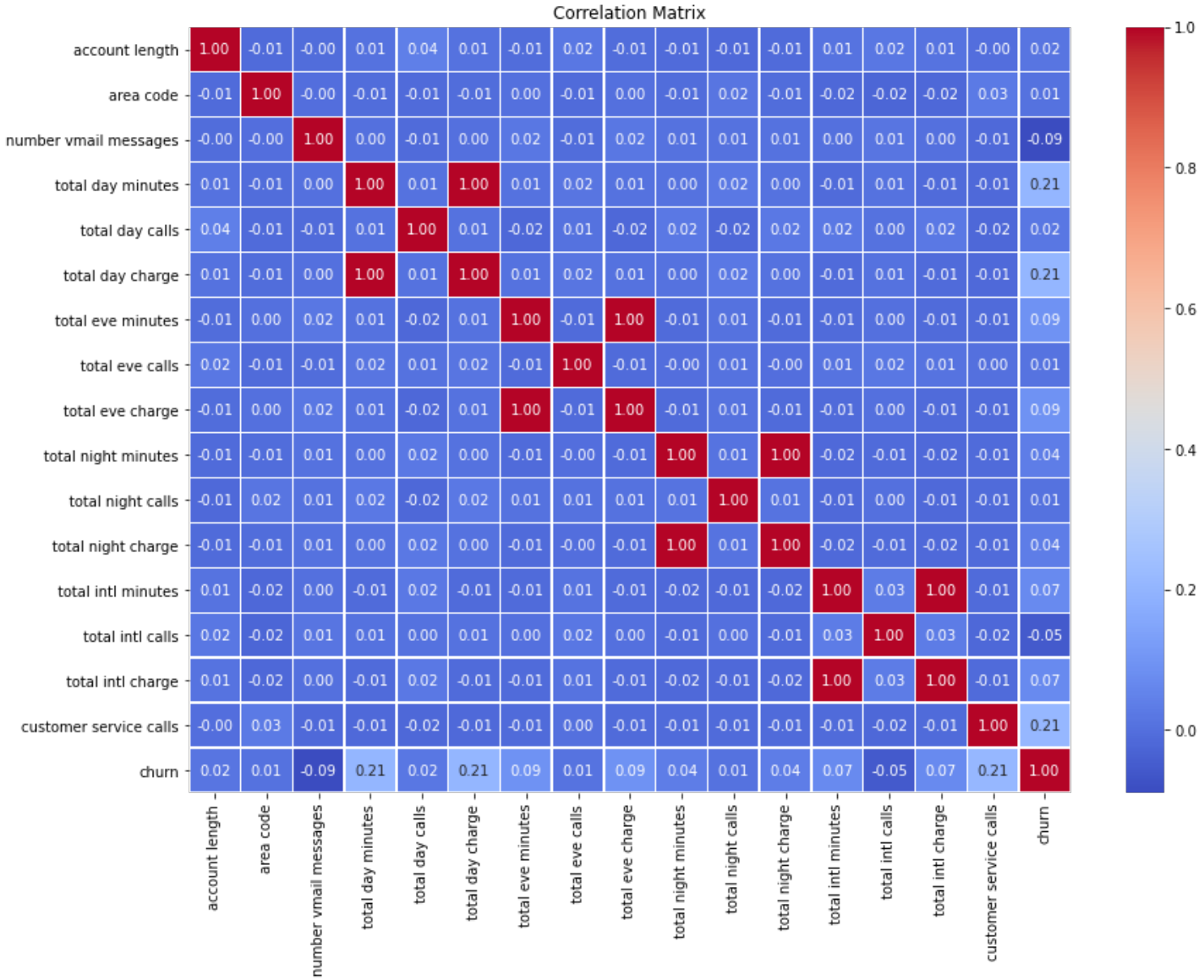
The plot indicates that customers who churned had a higher day charge of approximately 38 compared to 30 for those who did not churn. This suggests that "Total Day Charge" could also be a significant feature in predicting churn.

Multivariate Analysis

Analyze relationships between features, identify correlations, and check for multicollinearity.

In [363]:

```
# Calculate the correlation matrix
correlation_matrix = st_data_eda.corr()

# Plot the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



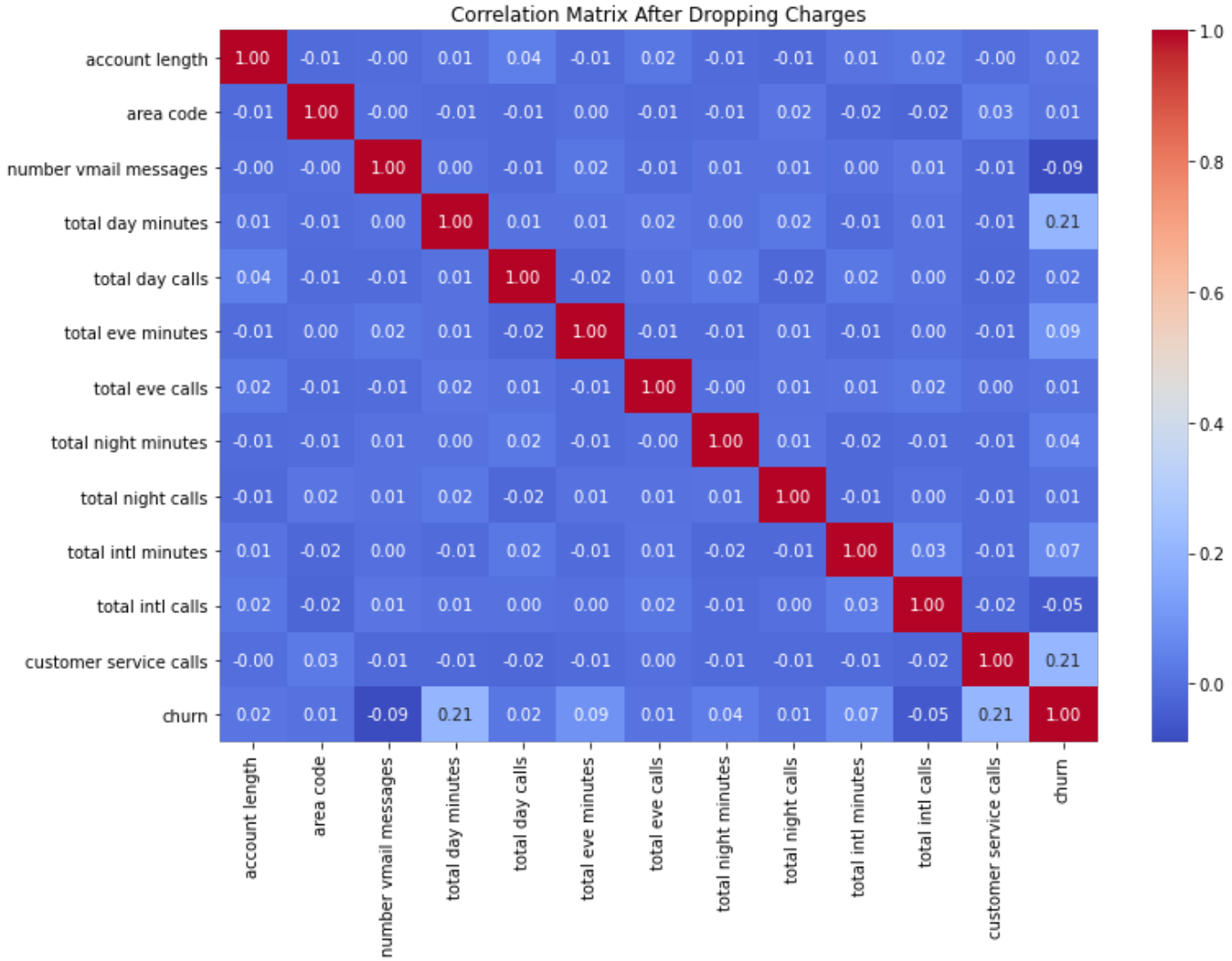
The heat map is mostly blue , indicating a positive sign, it suggests that the independent variables are not highly correlated. However some variables have a correlation of 1, indicating that some of the features will be removed so as not to affect the model.

To deal with the perfect correlation, I will remove the following features.

1. Total day charge
2. Total eve charge
3. Total night charge
4. total international charge

In [364]:

```
#drop the features stated above
st_data_eda_drop = st_data_eda.drop(columns=["total day charge", "total eve charge", "tot
al night charge", "total intl charge"])

#create acorrelation matrix
correlation_matrix_2 = st_data_eda_drop.corr()

# Plot a heatmap of the correlations
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix_2, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix After Dropping Charges')
plt.show()
```



This second correlation matrix shows a better output than the first one, there is no perfect correlation among the features as before, indicating multicollinearity between features will be minimal.

In [365]:

```
#multicollinearity check
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm

# Adjust display settings to show all rows
pd.set_option('display.max_rows', None)

# Dropping the target variable
```

```
X = st_data_eda_drop.drop(columns=['churn'])

# Convert categorical variables into dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Adding a constant column for VIF calculation
X = sm.add_constant(X)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF
print(vif_data)
```

	Feature	VIF
0	const	312.144220
1	account length	1.015276
2	area code	1.018033
3	number vmail messages	12.062809
4	total day minutes	1.019878
5	total day calls	1.022658
6	total eve minutes	1.018091
7	total eve calls	1.021489
8	total night minutes	1.016439
9	total night calls	1.017095
10	total intl minutes	1.019030
11	total intl calls	1.015103
12	customer service calls	1.017733
13	state_AL	2.482560
14	state_AR	2.032361
15	state_AZ	2.193737
16	state_CA	1.642227
17	state_CO	2.232156
18	state_CT	2.380871
19	state_DC	2.014336
20	state_DE	2.143200
21	state_FL	2.189126
22	state_GA	2.019812
23	state_HI	1.992987
24	state_IA	1.829140
25	state_ID	2.364624
26	state_IL	2.092231
27	state_IN	2.334338
28	state_KS	2.309489
29	state_KY	2.103615
30	state_LA	1.958202
31	state_MA	2.220941
32	state_MD	2.308676
33	state_ME	2.159554
34	state_MI	2.363306
35	state_MN	2.561025
36	state_MO	2.178062
37	state_MS	2.213624
38	state_MT	2.268379
39	state_NC	2.274687
40	state_ND	2.164481
41	state_NE	2.143000
42	state_NH	2.046422
43	state_NJ	2.272374
44	state_NM	2.165265
45	state_NV	2.235252
46	state_NY	2.538420
47	state_OH	2.454807
48	state_OK	2.141315
49	state_OR	2.449740
50	state_PA	1.846457
51	state_RI	2.218296
52	state_SC	2.124807
53	state_SD	2.121138
54	state_TN	2.000253

```
55                 state_TX    2.341071
56                 state_UT    2.337305
57                 state_VA    2.441185
58                 state_VT    2.361499
59                 state_WA    2.234645
60                 state_WI    2.447404
61                 state_WV    2.952571
62                 state_WY    2.435768
63  international plan_yes    1.029471
64     voice mail plan_yes   12.076475
```

From the output above, the features of voice mail plan (12), number of vmail messages (12) have greater values of more than 5, suggesting that the corresponding feature is highly correlated with other features.

Also I will drop the state feature as well because I have the area code feature, which will be converted into object type (categorical column)

The constant (intercept) term has a high VIF, which is typical and not concerning because it doesn't relate to multicollinearity issues among the explanatory variables.

In [366]:

```
#deleting the state feature

state_columns = [col for col in st_data_eda_drop.columns if col.startswith('state')]
st_eda_clean = st_data_eda_drop.drop(columns=state_columns)

# Display the first few rows of the cleaned DataFrame
st_eda_clean.head(3)
```

Out[366]:

	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes	total night calls	total intl minutes	total intl calls	customer service calls	ch
0	128	415	no	yes	25	265.1	110	197.4	99	244.7	91	10.0	3	1	Fa
1	107	415	no	yes	26	161.6	123	195.5	103	254.4	103	13.7	3	1	Fa
2	137	415	no	no	0	243.4	114	121.2	110	162.6	104	12.2	5	0	Fa

In [367]:

```
# Convert integer column "area code" to object type
st_eda_clean['area code'] = st_eda_clean['area code'].astype('object')
st_eda_clean.info() #to confirm the area code column is of object dtype
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 15 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   account length          3333 non-null   int64  
 1   area code               3333 non-null   object 
 2   international plan      3333 non-null   object 
 3   voice mail plan         3333 non-null   object 
 4   number vmail messages   3333 non-null   int64  
 5   total day minutes       3333 non-null   float64
 6   total day calls         3333 non-null   int64  
 7   total eve minutes       3333 non-null   float64
 8   total eve calls         3333 non-null   int64  
 9   total night minutes     3333 non-null   float64
 10  total night calls       3333 non-null   int64  
 11  total intl minutes      3333 non-null   float64
 12  total intl calls        3333 non-null   int64  
 13  customer service calls  3333 non-null   int64  
 14  churn                   3333 non-null   bool   
dtypes: bool(1), float64(4), int64(7), object(3)
memory usage: 367.9+ KB
```

State feature removed, calculate the VIF again

In [368]:

```
# Adjust display settings to show all rows
pd.set_option('display.max_rows', None)

# Dropping the target variable
X = st_eda_clean.drop(columns=['churn'])

# Convert categorical variables into dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Adding a constant column for VIF calculation
X = sm.add_constant(X)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF
print(vif_data)
```

```
                   Feature         VIF
0                    const  144.688648
1           account length    1.004152
2    number vmail messages   11.895531
3        total day minutes    1.004961
4          total day calls    1.003927
5        total eve minutes    1.002106
6          total eve calls    1.001597
7      total night minutes    1.002864
8        total night calls    1.003710
9       total intl minutes    1.006055
10        total intl calls    1.004135
11  customer service calls    1.003293
12           area code_415    1.504926
13           area code_510    1.503982
14  international plan_yes    1.010065
15     voice mail plan_yes   11.899038
```

I will remove the number vmail messages feature, because we have the voice mail plan feature and calcualte the VIF again.

In [369]:

```
#drop the features stated above
st_eda_clean_vif = st_eda_clean.drop(columns=["number vmail messages"])


#multicollinearity check
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm

# Adjust display settings to show all rows
pd.set_option('display.max_rows', None)

# Dropping the target variable
X = st_eda_clean_vif.drop(columns=['churn'])

# Convert categorical variables into dummy/indicator variables
X = pd.get_dummies(X, drop_first=True)

# Adding a constant column for VIF calculation
X = sm.add_constant(X)

# Calculate VIF for each feature
vif_data = pd.DataFrame()
```

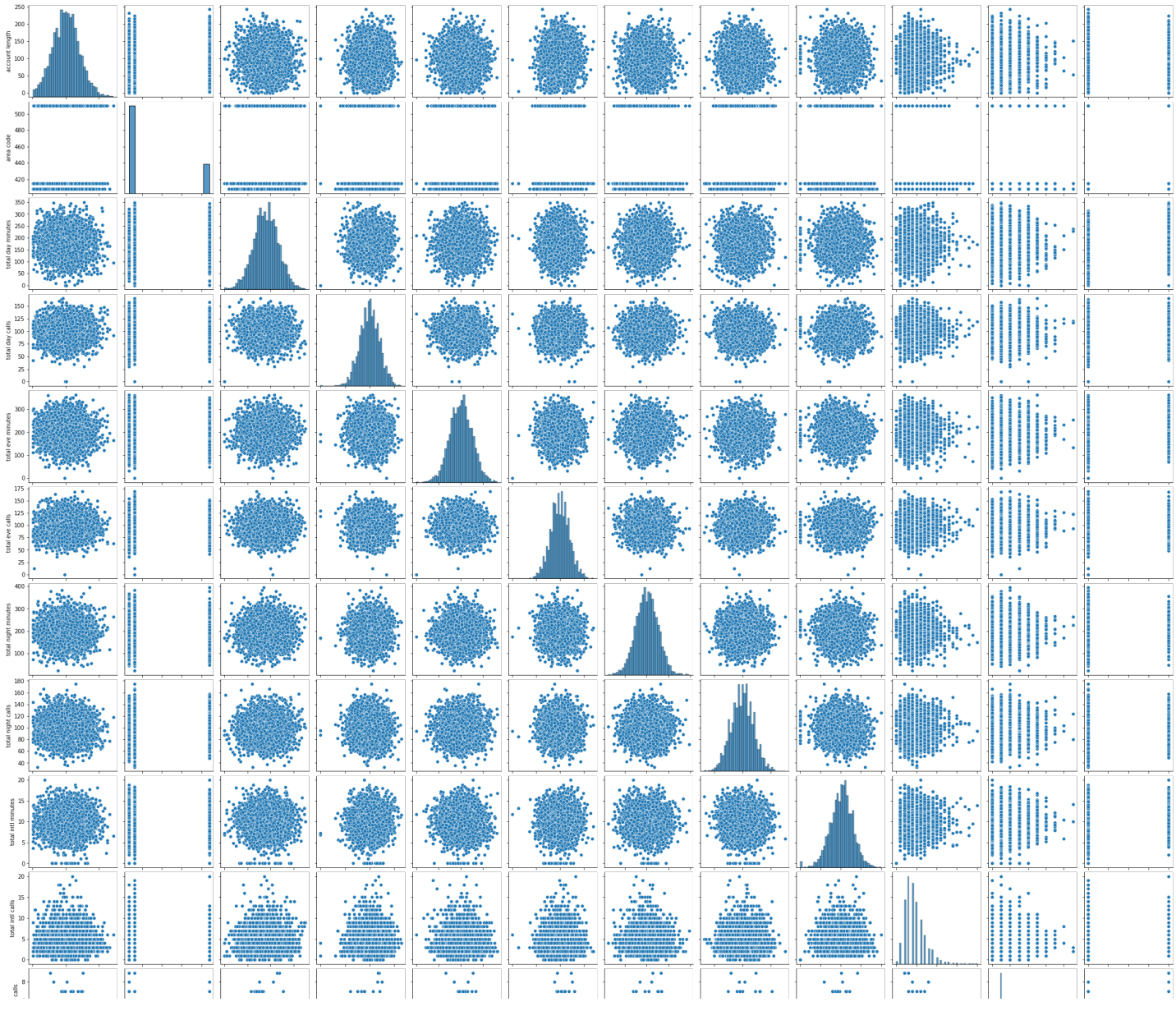
```
vif_data["Feature"] = X.columns
vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]

# Display VIF
print(vif_data)
```

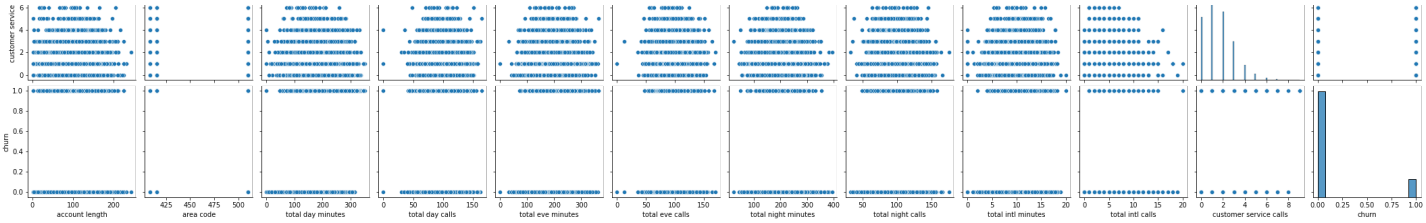
```
                   Feature         VIF
0                    const  144.685205
1           account length    1.003423
2        total day minutes    1.004889
3          total day calls    1.003915
4        total eve minutes    1.001999
5          total eve calls    1.001597
6      total night minutes    1.002814
7        total night calls    1.002976
8       total intl minutes    1.005892
9         total intl calls    1.003605
10  customer service calls    1.003107
11           area code_415    1.504909
12           area code_510    1.503977
13  international plan_yes    1.009948
14     voice mail plan_yes    1.001703
```

In [370]:

```
sns.pairplot(st_eda_clean_vif)
plt.show()
```

```
<__array_function__ internals>:5: RuntimeWarning: Converting input from bool to <class 'n
umpy.uint8'> for compatibility.
<__array_function__ internals>:5: RuntimeWarning: Converting input from bool to <class 'n
umpy.uint8'> for compatibility.
```





Feature Engineering

The relevant features for this model are as follows:

```
                  Features
   1.total day minutes
   2. total day calls
   3. total eve minutes
   4. total eve calls
   5. total night minutes
   6. total night calls
   7. total intl minutes
   8. total intl calls
   9. customer service calls
   10. area code_415
   11. area code_510
   12. international plan_yes
   13. voice mail plan_yes
   14. Account Length
```

In [371]:

```
#create the df to use with this specific features from the first original dataframe
columns_to_drop = [
    'state', 'phone number',
    'number vmail messages', 'total day charge',
    'total eve charge', 'total night charge',
    'total intl charge'
]

# Create the new DataFrame by dropping these columns
st_data_feat =st_data.drop(columns=columns_to_drop)

#convert the area code to object type

st_data_feat['area code'] = st_data_feat['area code'].astype('object')

print(st_data_feat.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 14 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   account length          3333 non-null   int64
 1   area code               3333 non-null   object
 2   international plan      3333 non-null   object
 3   voice mail plan         3333 non-null   object
 4   total day minutes       3333 non-null   float64
 5   total day calls         3333 non-null   int64
 6   total eve minutes       3333 non-null   float64
 7   total eve calls         3333 non-null   int64
 8   total night minutes     3333 non-null   float64
 9   total night calls       3333 non-null   int64
 10  total intl minutes      3333 non-null   float64
 11  total intl calls        3333 non-null   int64
 12  customer service calls  3333 non-null   int64
 13  churn                   3333 non-null   bool
```

```
dtypes: bool(1), float64(4), int64(6), object(3)
memory usage: 341.9+ KB
None
```

Train-Test Split & preprocessing of data

In [372]:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE

# Split the dataset into training and testing sets (70% for training, 30% for testing)
X = st_data_feat.drop(columns=['churn'])  # Features
y = st_data_feat['churn']  # Target variable


# Encode categorical variables
label_encoder = LabelEncoder()

# Apply label encoding to all categorical features in the dataset
X_encoded = X.apply(lambda col: label_encoder.fit_transform(col) if col.dtype == 'object'
else col)

# Split the encoded data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.3, random_
state=42)

# Apply SMOTE to the training data only
smote = SMOTE(random_state=0)
X_train_balanced, y_train_balanced = smote.fit_resample(X_train, y_train)

# Verify the shapes after applying SMOTE
print(f"Balanced training set size: {X_train_balanced.shape}, {y_train_balanced.shape}")

# Now, I can train my model on X_train_balanced and y_train_balanced
```

```
Balanced training set size: (3986, 13), (3986,)
```

In [373]:

```
X_train_balanced.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3986 entries, 0 to 3985
Data columns (total 13 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   account length          3986 non-null   int64
 1   area code               3986 non-null   int32
 2   international plan      3986 non-null   int32
 3   voice mail plan         3986 non-null   int32
 4   total day minutes       3986 non-null   float64
 5   total day calls         3986 non-null   int64
 6   total eve minutes       3986 non-null   float64
 7   total eve calls         3986 non-null   int64
 8   total night minutes     3986 non-null   float64
 9   total night calls       3986 non-null   int64
 10  total intl minutes      3986 non-null   float64
 11  total intl calls        3986 non-null   int64
 12  customer service calls  3986 non-null   int64
dtypes: float64(4), int32(3), int64(6)
memory usage: 358.2 KB
```

We now have a training and testing set ready for model building. Now, I need to map all feature variables onto the same scale, called feature scaling. I will do as follows.

Feature Scaling

Feature Scaling

In [374]:

```
X_train_balanced.describe()
```

Out[374]:

	account length	area code	international plan	voice mail plan	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes
count	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000
mean	101.705469	0.874310	0.097842	0.177371	190.174035	100.655795	205.462493	100.316357	201.833894
std	37.221323	0.683564	0.297139	0.382030	60.811365	18.980990	49.887747	18.866260	47.727556
min	1.000000	0.000000	0.000000	0.000000	2.600000	30.000000	0.000000	0.000000	23.200000
25%	75.250000	0.000000	0.000000	0.000000	145.500000	89.000000	171.400000	88.000000	168.602088
50%	101.000000	1.000000	0.000000	0.000000	186.650000	101.000000	204.612330	100.000000	201.950000
75%	127.000000	1.000000	0.000000	0.000000	236.529621	113.000000	237.388605	113.000000	234.513643
max	232.000000	2.000000	1.000000	1.000000	346.800000	165.000000	363.700000	170.000000	395.000000

In [375]:

```
#scale the features
# Initialize and fit the scaler on the original training data
scaler = MinMaxScaler()
scaler.fit(X_train)  # Fit on the original training data
```

Out[375]:

▾ MinMaxScaler

MinMaxScaler()

In [376]:

```
# Transform the balanced training data
X_train_balanced = pd.DataFrame(
    scaler.transform(X_train_balanced),
    columns=X_train_balanced.columns
)
```

In [377]:

```
cols = X_test.columns

X_test = pd.DataFrame(
    scaler.transform(X_test),
    columns=cols
    )

X_test.describe()
```

Out[377]:

	account length	area code	international plan	voice mail plan	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes
count	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	0.440113	0.508500	0.10100	0.286000	0.513088	0.518511	0.552746	0.589735	0.479135
std	0.172348	0.356798	0.30148	0.452115	0.158068	0.151695	0.139140	0.115031	0.133789
min	0.000000	0.000000	0.00000	0.000000	-0.007554	-0.222222	0.085785	0.270588	0.055137
25%	0.329004	0.500000	0.00000	0.000000	0.412987	0.414815	0.459376	0.511765	0.390196

	account length	area code	international plan	voice mail plan	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes
50%	0.441558	0.500000	0.00000	0.000000	0.515543	0.518519	0.556503	0.594118	0.478483
75%	0.545455	1.000000	0.00000	1.000000	0.619988	0.622222	0.650605	0.664706	0.570939
max	1.047619	1.000000	1.00000	1.000000	1.011621	0.985185	0.938411	0.964706	0.926573

In [378]:

```
X_train_balanced.describe()
```

Out[378]:

	account length	area code	international plan	voice mail plan	total day minutes	total day calls	total eve minutes	total eve calls	total night minutes
count	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000	3986.000000
mean	0.435954	0.437155	0.097842	0.177371	0.544957	0.523376	0.564923	0.590096	0.480457
std	0.161131	0.341782	0.297139	0.382030	0.176675	0.140600	0.137167	0.110978	0.128369
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.321429	0.000000	0.000000	0.000000	0.415166	0.437037	0.471268	0.517647	0.391076
50%	0.432900	0.500000	0.000000	0.000000	0.534718	0.525926	0.562585	0.588235	0.480769
75%	0.545455	0.500000	0.000000	0.000000	0.679633	0.614815	0.652704	0.664706	0.568353
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

We have a X_train_balanced dataset ready to be fed into the Logistic Regression Classifier as follows.

Data Modeling

In [379]:

```
# check for null values in the training dataset
X_train_balanced.isnull().sum()
```

Out[379]:

```
account length            0
area code                 0
international plan        0
voice mail plan           0
total day minutes         0
total day calls           0
total eve minutes         0
total eve calls           0
total night minutes       0
total night calls         0
total intl minutes        0
total intl calls          0
customer service calls    0
dtype: int64
```

In [380]:

```
# Train the logistic regression model on the balanced dataset

logreg_balanced = LogisticRegression(solver='liblinear', random_state=42)
logreg_balanced.fit(X_train_balanced, y_train_balanced)
```

Out[380]:

```
LogisticRegression(random_state=42, solver='liblinear')
```

Evaluation

Predict Results

In [381]:

```
# Make predictions on the test set
y_pred_test = logreg_balanced.predict(X_test)
```

In [382]:

```
#predict_proba method
# predict_proba method gives the probabilities for the target variable(0 and 1) in this c
ase, in array form.

#False is for probability of no churn and true is for probability of churn
# probability of getting output as 0 - no churn

logreg_balanced.predict_proba(X_test)[:,0]
```

Out[382]:

```
array([0.46782921, 0.80887128, 0.71160265, 0.29286159, 0.6423307 ,
       0.89627909, 0.86039837, 0.89711335, 0.41362404, 0.77702189,
       0.86871319, 0.72987619, 0.66245707, 0.49376873, 0.71810301,
       0.27307533, 0.5888375 , 0.25154898, 0.21809268, 0.49746798,
       0.79608923, 0.68255107, 0.85703226, 0.26631529, 0.84547359,
       0.91628341, 0.9346228 , 0.34846813, 0.60665229, 0.67757843,
       0.90536578, 0.36831492, 0.82865471, 0.79451789, 0.67465091,
       0.34476838, 0.86829318, 0.97950289, 0.49989419, 0.24701195,
       0.54085872, 0.44475183, 0.25862251, 0.22348012, 0.60961747,
       0.71734364, 0.47174925, 0.50108914, 0.91562303, 0.79119614,
       0.83520323, 0.25356042, 0.71442736, 0.9195746 , 0.78613063,
       0.24759886, 0.42824232, 0.78082004, 0.43412563, 0.87682461,
       0.64291357, 0.44006794, 0.94417926, 0.89355949, 0.34516508,
       0.21447154, 0.50447432, 0.89466239, 0.17356797, 0.65799982,
       0.15359707, 0.51600997, 0.34690602, 0.26151148, 0.5290471 ,
       0.70547908, 0.44436577, 0.30587183, 0.70046459, 0.81887476,
       0.6321846 , 0.71223898, 0.2784417 , 0.70573287, 0.68453652,
       0.79934588, 0.56176478, 0.76720373, 0.72374948, 0.36807597,
       0.42359434, 0.47137645, 0.37899573, 0.69641487, 0.51158299,
       0.35843707, 0.27226025, 0.3361519 , 0.54427084, 0.84123138,
       0.37433136, 0.46351783, 0.83781091, 0.31501293, 0.66813578,
       0.68950194, 0.5851664 , 0.93892647, 0.89675475, 0.80568414,
       0.52445412, 0.64874689, 0.49705284, 0.1408622 , 0.66144955,
       0.41301947, 0.23723848, 0.58231887, 0.66786189, 0.36228546,
       0.32866212, 0.24962849, 0.55537749, 0.63182019, 0.23077162,
       0.28610363, 0.16680893, 0.30692033, 0.82638595, 0.90216798,
       0.75677849, 0.62333467, 0.72474898, 0.63662358, 0.61018904,
       0.55919953, 0.63111758, 0.7721834 , 0.9621715 , 0.82432993,
       0.91241755, 0.63101279, 0.34183958, 0.0409307 , 0.76052202,
       0.04818346, 0.60474669, 0.95768251, 0.044323  , 0.67394019,
       0.64198756, 0.62366704, 0.85510289, 0.76977857, 0.31464022,
       0.3834011 , 0.84244059, 0.44658966, 0.84144411, 0.64592724,
       0.70655495, 0.8102474 , 0.91921519, 0.66057668, 0.26365429,
       0.93018475, 0.8070466 , 0.19168501, 0.5187581 , 0.35710434,
       0.76475119, 0.96498948, 0.95463233, 0.27185738, 0.95109018,
       0.71698273, 0.92257224, 0.1070995 , 0.55753383, 0.2406968 ,
       0.46391874, 0.59756146, 0.69434463, 0.62605949, 0.88026331,
       0.79900833, 0.47558111, 0.1982078 , 0.30248817, 0.56362675,
       0.91856895, 0.62228242, 0.84755893, 0.33475287, 0.9672139 ,
       0.62289164, 0.71061535, 0.48226738, 0.97450894, 0.47645606,
       0.51071886, 0.83042381, 0.37376164, 0.94027555, 0.76463812,
       0.97577384, 0.87141844, 0.64864356, 0.95416644, 0.17262992,
       0.98685024, 0.73807633, 0.75514423, 0.67692672, 0.24520491,
       0.4114927 , 0.57825043, 0.46196487, 0.09866674, 0.33070171,
       0.79859624, 0.76494844, 0.33726936, 0.55855   , 0.84451579,
       0.79075154, 0.76520736, 0.55044082, 0.55376618, 0.16623615,
       0.68077379, 0.6542212 , 0.72537106, 0.87363633, 0.75907485,
```

0.34497264, 0.31841147, 0.49358208, 0.87678665, 0.78112168,
0.89328924, 0.50112942, 0.66577391, 0.58015487, 0.91819277,
0.78672762, 0.78037945, 0.85931542, 0.85786021, 0.90313949,
0.19070739, 0.88645287, 0.64178363, 0.48009231, 0.49395063,
0.52259554, 0.36942827, 0.80873991, 0.08209472, 0.48035628,
0.57306423, 0.9636851 , 0.81627461, 0.03115077, 0.3214399 ,
0.63685382, 0.57595232, 0.97701498, 0.34940402, 0.60979376,
0.2003877 , 0.22161416, 0.40332623, 0.93482631, 0.30508102,
0.31884513, 0.20740587, 0.56873405, 0.5340699 , 0.54823455,
0.30916304, 0.59874901, 0.95859717, 0.58465222, 0.93690997,
0.3408987 , 0.74576799, 0.8563731 , 0.43290785, 0.7757613 ,
0.7941949 , 0.447966 , 0.50802826, 0.77965675, 0.4272398 ,
0.75313751, 0.85852982, 0.20418655, 0.99033533, 0.20084279,
0.52453932, 0.88959007, 0.5526156 , 0.58561169, 0.26572993,
0.39801817, 0.67395861, 0.41499889, 0.87247981, 0.69071196,
0.55309408, 0.45011815, 0.0714489 , 0.51280351, 0.13614935,
0.77524836, 0.59359368, 0.3138053 , 0.39930726, 0.32430653,
0.12560465, 0.62574224, 0.89229461, 0.64457326, 0.77201532,
0.37388619, 0.3398995 , 0.55285483, 0.51457197, 0.96634967,
0.65594668, 0.34110541, 0.91678918, 0.96066522, 0.89790372,
0.72730387, 0.21488946, 0.52354628, 0.44809751, 0.83836279,
0.41122214, 0.90631134, 0.6399699 , 0.19280924, 0.27396871,
0.21289132, 0.22033132, 0.79625596, 0.49894715, 0.65434288,
0.59103988, 0.71967917, 0.83146695, 0.73956228, 0.79488059,
0.4906988 , 0.94156686, 0.47617711, 0.43974461, 0.53299138,
0.86812197, 0.55934939, 0.73428166, 0.96353454, 0.92509027,
0.48482682, 0.63114864, 0.26935782, 0.4881102 , 0.93072496,
0.70910079, 0.62486213, 0.42616876, 0.92728243, 0.40070193,
0.8860915 , 0.45469111, 0.89249747, 0.25818486, 0.84932531,
0.57688469, 0.62646781, 0.8318478 , 0.79795842, 0.56332503,
0.51542   , 0.6340796 , 0.31849137, 0.19007151, 0.19383562,
0.77730423, 0.73001193, 0.17351454, 0.7810226 , 0.9576292 ,
0.82408872, 0.97797379, 0.30452621, 0.17982772, 0.28448446,
0.93014268, 0.75426766, 0.60285563, 0.65218842, 0.96606506,
0.73784386, 0.45146744, 0.58832688, 0.09604026, 0.5655949 ,
0.54593934, 0.75017826, 0.72738179, 0.94723362, 0.17181733,
0.24645955, 0.75675669, 0.43284741, 0.75186165, 0.7589038 ,
0.26516833, 0.91260144, 0.2096426 , 0.66586269, 0.29603926,
0.97725052, 0.34047834, 0.27942595, 0.89712815, 0.76868414,
0.70451015, 0.40728781, 0.57885434, 0.89228265, 0.75902612,
0.94202305, 0.49267377, 0.07838764, 0.53631639, 0.84719239,
0.97482337, 0.73591355, 0.1805489 , 0.79040324, 0.79865822,
0.98846818, 0.84141589, 0.58040791, 0.81484406, 0.77389376,
0.97114251, 0.74452154, 0.46183675, 0.90061036, 0.6426893 ,
0.75032438, 0.55602581, 0.47331494, 0.46629349, 0.48169443,
0.57523382, 0.945418 , 0.7840822 , 0.94488472, 0.83053995,
0.48241218, 0.33010026, 0.82891791, 0.71006426, 0.12403708,
0.74451835, 0.80176661, 0.41332066, 0.60099113, 0.31268508,
0.23568664, 0.78319878, 0.5121204 , 0.55581919, 0.90967449,
0.795586 , 0.49710512, 0.87862843, 0.88902383, 0.65709356,
0.8925147 , 0.4801875 , 0.72279263, 0.15140867, 0.60218896,
0.77201238, 0.19304294, 0.92314624, 0.8768511 , 0.34171298,
0.85600466, 0.84966316, 0.5260574 , 0.93276924, 0.82117277,
0.96052849, 0.82958388, 0.32810177, 0.9289055 , 0.57376667,
0.72494169, 0.46462818, 0.28556499, 0.73639427, 0.80805757,
0.87386233, 0.70933635, 0.57139135, 0.7159562 , 0.69824732,
0.28146819, 0.79330868, 0.40407824, 0.46838509, 0.44156328,
0.45468897, 0.94265472, 0.61302386, 0.67725103, 0.55091067,
0.94485291, 0.59055835, 0.62230219, 0.82396005, 0.86745126,
0.94408412, 0.34511739, 0.91414199, 0.44165124, 0.24429527,
0.84836363, 0.49460981, 0.77651724, 0.74809445, 0.26143391,
0.86958495, 0.68348668, 0.74027093, 0.73188456, 0.67668205,
0.27257425, 0.95371647, 0.94870622, 0.7943732 , 0.6996928 ,
0.359871 , 0.91912016, 0.31744197, 0.73960747, 0.33977135,
0.35438651, 0.45392758, 0.56417942, 0.8771745 , 0.30588101,
0.14104779, 0.74707236, 0.88107104, 0.88031103, 0.66325511,
0.87710251, 0.75179635, 0.46021462, 0.78505625, 0.59377157,
0.09249285, 0.82728885, 0.72375247, 0.86941423, 0.7021205 ,
0.9395804 , 0.76344381, 0.88875586, 0.52851584, 0.68068515,
0.79477476, 0.8139378 , 0.1997738 , 0.59632705, 0.51161126,
0.81308448, 0.59707289, 0.82395807, 0.59509769, 0.42810388,
0.50482495, 0.63525883, 0.77148513, 0.88115757, 0.49330337,

0.94247557, 0.89926285, 0.85257654, 0.71678248, 0.76298298,
0.96009776, 0.74382318, 0.67688421, 0.53763221, 0.64369598,
0.60949708, 0.24820925, 0.90409429, 0.94690703, 0.75730829,
0.92727427, 0.76924937, 0.21625914, 0.72882835, 0.90527056,
0.55859063, 0.45842827, 0.91658221, 0.28711885, 0.32738349,
0.45152542, 0.44170563, 0.60389967, 0.70104835, 0.21356633,
0.78262207, 0.87834618, 0.18388157, 0.69426552, 0.24952261,
0.14159814, 0.8097992 , 0.60672891, 0.2992324 , 0.90576675,
0.36122142, 0.81235646, 0.51769386, 0.41676373, 0.3940818 ,
0.24091094, 0.2307081 , 0.77320211, 0.3430478 , 0.7085677 ,
0.77276786, 0.48303484, 0.19629942, 0.92043833, 0.88993158,
0.73969182, 0.7085794 , 0.84983321, 0.48279702, 0.60238471,
0.88372615, 0.47858533, 0.56475522, 0.80342382, 0.33552917,
0.87285645, 0.27199088, 0.65196836, 0.72313814, 0.90368166,
0.97311474, 0.80156961, 0.86762246, 0.62632314, 0.28395573,
0.70902346, 0.73020653, 0.27984811, 0.20697076, 0.87035811,
0.93281216, 0.93103033, 0.66090734, 0.30206342, 0.65956448,
0.5282145 , 0.8297098 , 0.41883612, 0.65229581, 0.20276669,
0.52303686, 0.32512675, 0.57104407, 0.81953707, 0.93672974,
0.428027 , 0.89425984, 0.25761684, 0.84818055, 0.44530287,
0.69273632, 0.62512136, 0.69250328, 0.45607975, 0.61079096,
0.45461934, 0.35394357, 0.8412129 , 0.83156283, 0.30161828,
0.58269939, 0.55401342, 0.24664288, 0.44041751, 0.89430365,
0.82173998, 0.66962665, 0.60127035, 0.36104672, 0.98884111,
0.69095657, 0.49899293, 0.14513082, 0.19998935, 0.26080532,
0.64770376, 0.50238695, 0.8063126 , 0.28882361, 0.80194868,
0.25800693, 0.68830652, 0.91064839, 0.22068838, 0.46986354,
0.70724598, 0.95771138, 0.98227756, 0.06710102, 0.45669629,
0.34441995, 0.54258798, 0.90690475, 0.78737386, 0.83814982,
0.91783954, 0.62078237, 0.65602685, 0.45933027, 0.66464808,
0.49195562, 0.3328015 , 0.69951394, 0.7621391 , 0.93712049,
0.80945719, 0.82413979, 0.57880314, 0.63661455, 0.66717575,
0.8635919 , 0.95835256, 0.27636707, 0.58111934, 0.60459552,
0.63198556, 0.85290013, 0.67145432, 0.49804879, 0.72972819,
0.14376849, 0.85603057, 0.72236969, 0.17513159, 0.94007997,
0.91983694, 0.84964316, 0.07236673, 0.81901201, 0.77594647,
0.85394058, 0.26554766, 0.76044543, 0.33519398, 0.89311666,
0.53527983, 0.32898145, 0.39921289, 0.98690699, 0.97110383,
0.09845582, 0.89251616, 0.71844154, 0.91608779, 0.60245765,
0.8447489 , 0.17202107, 0.92715531, 0.23059176, 0.5545908 ,
0.33329687, 0.93113505, 0.45907724, 0.71250182, 0.19450312,
0.29444332, 0.70684918, 0.86905801, 0.54822454, 0.34584818,
0.35458115, 0.26618044, 0.81392719, 0.87507974, 0.24719901,
0.48926607, 0.56005832, 0.60826055, 0.84903825, 0.68811547,
0.92798642, 0.96371058, 0.68868556, 0.9707084 , 0.40841189,
0.83852878, 0.6601588 , 0.47801017, 0.47845034, 0.20514171,
0.74357384, 0.61322428, 0.46932275, 0.95074396, 0.48159264,
0.86032504, 0.8154759 , 0.87606349, 0.92458093, 0.51422908,
0.19623578, 0.48188507, 0.68479198, 0.58448386, 0.29414682,
0.2588994 , 0.90211996, 0.85390063, 0.83866423, 0.69491646,
0.57534798, 0.83709876, 0.87887119, 0.09000553, 0.71005188,
0.6914518 , 0.9491676 , 0.93776923, 0.27548271, 0.76727689,
0.31954536, 0.70766763, 0.2188822 , 0.35456363, 0.53843939,
0.71702076, 0.32054695, 0.35427117, 0.53778822, 0.59157472,
0.5667779 , 0.61259749, 0.73133869, 0.55468421, 0.43514267,
0.74944505, 0.34946488, 0.8272948 , 0.68682184, 0.55526953,
0.41919032, 0.75008585, 0.22903348, 0.70439813, 0.88491669,
0.72999268, 0.43201188, 0.41506453, 0.71833005, 0.67718778,
0.77987472, 0.94909499, 0.61102057, 0.27537578, 0.64307021,
0.71829516, 0.80396327, 0.63417398, 0.40047751, 0.83599364,
0.83228476, 0.66417811, 0.60239246, 0.19760652, 0.45028259,
0.51721355, 0.92684997, 0.71159655, 0.47026716, 0.632451 ,
0.64122383, 0.58649256, 0.37416395, 0.19761275, 0.44159836,
0.72308106, 0.65200306, 0.18618127, 0.97654286, 0.90004768,
0.95812753, 0.59484071, 0.24692728, 0.29422738, 0.47853364,
0.3217371 , 0.69627252, 0.56623156, 0.43345876, 0.18768102,
0.17653106, 0.1334666 , 0.49522271, 0.34458895, 0.7051344 ,
0.20887331, 0.92188506, 0.64966112, 0.13820134, 0.59975726,
0.6828169 , 0.9203612 , 0.37305888, 0.12903831, 0.37087814,
0.47002242, 0.610892 , 0.57494955, 0.62694048, 0.67981538,
0.89195825, 0.5844672 , 0.46108851, 0.96632506, 0.43019436,
0.58810159, 0.28042724, 0.71471132, 0.78018154, 0.259462 ,

```
       0.49391344, 0.84264467, 0.56800775, 0.13016804, 0.28728277,
       0.29928604, 0.76694998, 0.88972145, 0.70784764, 0.85854009,
       0.59376679, 0.41564218, 0.65930491, 0.73701283, 0.6363462 ,
       0.94661684, 0.47987335, 0.86317646, 0.69835519, 0.44029846,
       0.92271997, 0.62490498, 0.12791219, 0.16200725, 0.86089481,
       0.89019751, 0.66751091, 0.62241247, 0.65124193, 0.59012767,
       0.47561408, 0.51983645, 0.95138313, 0.73046218, 0.94126452,
       0.53149673, 0.28466384, 0.55037822, 0.28552382, 0.76165944,
       0.58531154, 0.87847955, 0.75440529, 0.7079775 , 0.90463974])
```

In [383]:

```
# probability of getting output as True

logreg_balanced.predict_proba(X_test)[:,1]
```

Out[383]:

```
array([0.53217079, 0.19112872, 0.28839735, 0.70713841, 0.3576693 ,
       0.10372091, 0.13960163, 0.10288665, 0.58637596, 0.22297811,
       0.13128681, 0.27012381, 0.33754293, 0.50623127, 0.28189699,
       0.72692467, 0.4111625 , 0.74845102, 0.78190732, 0.50253202,
       0.20391077, 0.31744893, 0.14296774, 0.73368471, 0.15452641,
       0.08371659, 0.0653772 , 0.65153187, 0.39334771, 0.32242157,
       0.09463422, 0.63168508, 0.17134529, 0.20548211, 0.32534909,
       0.65523162, 0.13170682, 0.02049711, 0.50010581, 0.75298805,
       0.45914128, 0.55524817, 0.74137749, 0.77651988, 0.39038253,
       0.28265636, 0.52825075, 0.49891086, 0.08437697, 0.20880386,
       0.16479677, 0.74643958, 0.28557264, 0.0804254 , 0.21386937,
       0.75240114, 0.57175768, 0.21917996, 0.56587437, 0.12317539,
       0.35708643, 0.55993206, 0.05582074, 0.10644051, 0.65483492,
       0.78552846, 0.49552568, 0.10533761, 0.82643203, 0.34200018,
       0.84640293, 0.48399003, 0.65309398, 0.73848852, 0.4709529 ,
       0.29452092, 0.55563423, 0.69412817, 0.29953541, 0.18112524,
       0.3678154 , 0.28776102, 0.7215583 , 0.29426713, 0.31546348,
       0.20065412, 0.43823522, 0.23279627, 0.27625052, 0.63192403,
       0.57640566, 0.52862355, 0.62100427, 0.30358513, 0.48841701,
       0.64156293, 0.72773975, 0.6638481 , 0.45572916, 0.15876862,
       0.62566864, 0.53648217, 0.16218909, 0.68498707, 0.33186422,
       0.31049806, 0.4148336 , 0.06107353, 0.10324525, 0.19431586,
       0.47554588, 0.35125311, 0.50294716, 0.8591378 , 0.33855045,
       0.58698053, 0.76276152, 0.41768113, 0.33213811, 0.63771454,
       0.67133788, 0.75037151, 0.44462251, 0.36817981, 0.76922838,
       0.71389637, 0.83319107, 0.69307967, 0.17361405, 0.09783202,
       0.24322151, 0.37666533, 0.27525102, 0.36337642, 0.38981096,
       0.44080047, 0.36888242, 0.2278166 , 0.0378285 , 0.17567007,
       0.08758245, 0.36898721, 0.65816042, 0.9590693 , 0.23947798,
       0.95181654, 0.39525331, 0.04231749, 0.955677  , 0.32605981,
       0.35801244, 0.37633296, 0.14489711, 0.23022143, 0.68535978,
       0.6165989 , 0.15755941, 0.55341034, 0.15855589, 0.35407276,
       0.29344505, 0.1897526 , 0.08078481, 0.33942332, 0.73634571,
       0.06981525, 0.1929534 , 0.80831499, 0.4812419 , 0.64289566,
       0.23524881, 0.03501052, 0.04536767, 0.72814262, 0.04890982,
       0.28301727, 0.07742776, 0.8929005 , 0.44246617, 0.7593032 ,
       0.53608126, 0.40243854, 0.30565537, 0.37394051, 0.11973669,
       0.20099167, 0.52441889, 0.8017922 , 0.69751183, 0.43637325,
       0.08143105, 0.37771758, 0.15244107, 0.66524713, 0.0327861 ,
       0.37710836, 0.28938465, 0.51773262, 0.02549106, 0.52354394,
       0.48928114, 0.16957619, 0.62623836, 0.05972445, 0.23536188,
       0.02422616, 0.12858156, 0.35135644, 0.04583356, 0.82737008,
       0.01314976, 0.26192367, 0.24485577, 0.32307328, 0.75479509,
       0.5885073 , 0.42174957, 0.53803513, 0.90133326, 0.66929829,
       0.20140376, 0.23505156, 0.66273064, 0.44145   , 0.15548421,
       0.20924846, 0.23479264, 0.44955918, 0.44623382, 0.83376385,
       0.31922621, 0.3457788 , 0.27462894, 0.12636367, 0.24092515,
       0.65502736, 0.68158853, 0.50641792, 0.12321335, 0.21887832,
       0.10671076, 0.49887058, 0.33422609, 0.41984513, 0.08180723,
       0.21327238, 0.21962055, 0.14068458, 0.14213979, 0.09686051,
       0.80929261, 0.11354713, 0.35821637, 0.51990769, 0.50604937,
       0.47740446, 0.63057173, 0.19126009, 0.91790528, 0.51964372,
       0.42693577, 0.0363149 , 0.18372539, 0.96884923, 0.6785601 ,
       0.36314618, 0.42404768, 0.02298502, 0.65059598, 0.39020624,
```

0.7996123 , 0.77838584, 0.59667377, 0.06517369, 0.69491898,
0.68115487, 0.79259413, 0.43126595, 0.4659301 , 0.45176545,
0.69083696, 0.40125099, 0.04140283, 0.41534778, 0.06309003,
0.6591013 , 0.25423201, 0.1436269 , 0.56709215, 0.2242387 ,
0.2058051 , 0.552034 , 0.49197174, 0.22034325, 0.5727602 ,
0.24686249, 0.14147018, 0.79581345, 0.00966467, 0.79915721,
0.47546068, 0.11040993, 0.4473844 , 0.41438831, 0.73427007,
0.60198183, 0.32604139, 0.58500111, 0.12752019, 0.30928804,
0.44690592, 0.54988185, 0.9285511 , 0.48719649, 0.86385065,
0.22475164, 0.40640632, 0.6861947 , 0.60069274, 0.67569347,
0.87439535, 0.37425776, 0.10770539, 0.35542674, 0.22798468,
0.62611381, 0.6601005 , 0.44714517, 0.48542803, 0.03365033,
0.34405332, 0.65889459, 0.08321082, 0.03933478, 0.10209628,
0.27269613, 0.78511054, 0.47645372, 0.55190249, 0.16163721,
0.58877786, 0.09368866, 0.3600301 , 0.80719076, 0.72603129,
0.78710868, 0.77966868, 0.20374404, 0.50105285, 0.34565712,
0.40896012, 0.28032083, 0.16853305, 0.26043772, 0.20511941,
0.5093012 , 0.05843314, 0.52382289, 0.56025539, 0.46700862,
0.13187803, 0.44065061, 0.26571834, 0.03646546, 0.07490973,
0.51517318, 0.36885136, 0.73064218, 0.5118898 , 0.06927504,
0.29089921, 0.37513787, 0.57383124, 0.07271757, 0.59929807,
0.1139085 , 0.54530889, 0.10750253, 0.74181514, 0.15067469,
0.42311531, 0.37353219, 0.1681522 , 0.20204158, 0.43667497,
0.48458   , 0.3659204 , 0.68150863, 0.80992849, 0.80616438,
0.22269577, 0.26998807, 0.82648546, 0.2189774 , 0.0423708 ,
0.17591128, 0.02202621, 0.69547379, 0.82017228, 0.71551554,
0.06985732, 0.24573234, 0.39714437, 0.34781158, 0.03393494,
0.26215614, 0.54853256, 0.41167312, 0.90395974, 0.4344051 ,
0.45406066, 0.24982174, 0.27261821, 0.05276638, 0.82818267,
0.75354045, 0.24324331, 0.56715259, 0.24813835, 0.2410962 ,
0.73483167, 0.08739856, 0.7903574 , 0.33413731, 0.70396074,
0.02274948, 0.65952166, 0.72057405, 0.10287185, 0.23131586,
0.29548985, 0.59271219, 0.42114566, 0.10771735, 0.24097388,
0.05797695, 0.50732623, 0.92161236, 0.46368361, 0.15280761,
0.02517663, 0.26408645, 0.8194511 , 0.20959676, 0.20134178,
0.01153182, 0.15858411, 0.41959209, 0.18515594, 0.22610624,
0.02885749, 0.25547846, 0.53816325, 0.09938964, 0.3573107 ,
0.24967562, 0.44397419, 0.52668506, 0.53370651, 0.51830557,
0.42476618, 0.054582 , 0.2159178 , 0.05511528, 0.16946005,
0.51758782, 0.66989974, 0.17108209, 0.28993574, 0.87596292,
0.25548165, 0.19823339, 0.58667934, 0.39900887, 0.68731492,
0.76431336, 0.21680122, 0.4878796 , 0.44418081, 0.09032551,
0.204414 , 0.50289488, 0.12137157, 0.11097617, 0.34290644,
0.1074853 , 0.5198125 , 0.27720737, 0.84859133, 0.39781104,
0.22798762, 0.80695706, 0.07685376, 0.1231489 , 0.65828702,
0.14399534, 0.15033684, 0.4739426 , 0.06723076, 0.17882723,
0.03947151, 0.17041612, 0.67189823, 0.0710945 , 0.42623333,
0.27505831, 0.53537182, 0.71443501, 0.26360573, 0.19194243,
0.12613767, 0.29066365, 0.42860865, 0.2840438 , 0.30175268,
0.71853181, 0.20669132, 0.59592176, 0.53161491, 0.55843672,
0.54531103, 0.05734528, 0.38697614, 0.32274897, 0.44908933,
0.05514709, 0.40944165, 0.37769781, 0.17603995, 0.13254874,
0.05591588, 0.65488261, 0.08585801, 0.55834876, 0.75570473,
0.15163637, 0.50539019, 0.22348276, 0.25190555, 0.73856609,
0.13041505, 0.31651332, 0.25972907, 0.26811544, 0.32331795,
0.72742575, 0.04628353, 0.05129378, 0.2056268 , 0.3003072 ,
0.640129 , 0.08087984, 0.68255803, 0.26039253, 0.66022865,
0.64561349, 0.54607242, 0.43582058, 0.1228255 , 0.69411899,
0.85895221, 0.25292764, 0.11892896, 0.11968897, 0.33674489,
0.12289749, 0.24820365, 0.53978538, 0.21494375, 0.40622843,
0.90750715, 0.17271115, 0.27624753, 0.13058577, 0.2978795 ,
0.0604196 , 0.23655619, 0.11124414, 0.47148416, 0.31931485,
0.20522524, 0.1860622 , 0.8002262 , 0.40367295, 0.48838874,
0.18691552, 0.40292711, 0.17604193, 0.40490231, 0.57189612,
0.49517505, 0.36474117, 0.22851487, 0.11884243, 0.50669663,
0.05752443, 0.10073715, 0.14742346, 0.28321752, 0.23701702,
0.03990224, 0.25617682, 0.32311579, 0.46236779, 0.35630402,
0.39050292, 0.75179075, 0.09590571, 0.05309297, 0.24269171,
0.07272573, 0.23075063, 0.78374086, 0.27117165, 0.09472944,
0.44140937, 0.54157173, 0.08341779, 0.71288115, 0.67261651,
0.54847458, 0.55829437, 0.39610033, 0.29895165, 0.78643367,
0.21737793, 0.12165382, 0.81611843, 0.30573448, 0.75047739,

0.85840186, 0.1902008 , 0.39327109, 0.7007676 , 0.09423325,
0.63877858, 0.18764354, 0.48230614, 0.58323627, 0.6059182 ,
0.75908906, 0.7692919 , 0.22679789, 0.6569522 , 0.2914323 ,
0.22723214, 0.51696516, 0.80370058, 0.07956167, 0.11006842,
0.26030818, 0.2914206 , 0.15016679, 0.51720298, 0.39761529,
0.11627385, 0.52141467, 0.43524478, 0.19657618, 0.66447083,
0.12714355, 0.72800912, 0.34803164, 0.27686186, 0.09631834,
0.02688526, 0.19843039, 0.13237754, 0.37367686, 0.71604427,
0.29097654, 0.26979347, 0.72015189, 0.79302924, 0.12964189,
0.06718784, 0.06896967, 0.33909266, 0.69793658, 0.34043552,
0.4717855 , 0.1702902 , 0.58116388, 0.34770419, 0.79723331,
0.47696314, 0.67487325, 0.42895593, 0.18046293, 0.06327026,
0.571973 , 0.10574016, 0.74238316, 0.15181945, 0.55469713,
0.30726368, 0.37487864, 0.30749672, 0.54392025, 0.38920904,
0.54538066, 0.64605643, 0.1587871 , 0.16843717, 0.69838172,
0.41730061, 0.44598658, 0.75335712, 0.55958249, 0.10569635,
0.17826002, 0.33037335, 0.39872965, 0.63895328, 0.01115889,
0.30904343, 0.50100707, 0.85486918, 0.80001065, 0.73919468,
0.35229624, 0.49761305, 0.1936874 , 0.71117639, 0.19805132,
0.74199307, 0.31169348, 0.08935161, 0.77931162, 0.53013646,
0.29275402, 0.04228862, 0.01772244, 0.93289898, 0.54330371,
0.65558005, 0.45741202, 0.09309525, 0.21262614, 0.16185018,
0.08216046, 0.37921763, 0.34397315, 0.54066973, 0.33535192,
0.50804438, 0.6671985 , 0.30048606, 0.2378609 , 0.06287951,
0.19054281, 0.17586021, 0.42119686, 0.36338545, 0.33282425,
0.1364081 , 0.04164744, 0.72363293, 0.41888066, 0.39540448,
0.36801444, 0.14709987, 0.32854568, 0.50195121, 0.27027181,
0.85623151, 0.14396943, 0.27763031, 0.82486841, 0.05992003,
0.08016306, 0.15035684, 0.92763327, 0.18098799, 0.22405353,
0.14605942, 0.73445234, 0.23955457, 0.66480602, 0.10688334,
0.46472017, 0.67101855, 0.60078711, 0.01309301, 0.02889617,
0.90154418, 0.10748384, 0.28155846, 0.08391221, 0.39754235,
0.1552511 , 0.82797893, 0.07284469, 0.76940824, 0.4454092 ,
0.66670313, 0.06886495, 0.54092276, 0.28749818, 0.80549688,
0.70555668, 0.29315082, 0.13094199, 0.45177546, 0.65415182,
0.64541885, 0.73381956, 0.18607281, 0.12492026, 0.75280099,
0.51073393, 0.43994168, 0.39173945, 0.15096175, 0.31188453,
0.07201358, 0.03628942, 0.31131444, 0.0292916 , 0.59158811,
0.16147122, 0.3398412 , 0.52198983, 0.52154966, 0.79485829,
0.25642616, 0.38677572, 0.53067725, 0.04925604, 0.51840736,
0.13967496, 0.1845241 , 0.12393651, 0.07541907, 0.48577092,
0.80376422, 0.51811493, 0.31520802, 0.41551614, 0.70585318,
0.7411006 , 0.09788004, 0.14609937, 0.16133577, 0.30508354,
0.42465202, 0.16290124, 0.12112881, 0.90999447, 0.28994812,
0.3085482 , 0.0508324 , 0.06223077, 0.72451729, 0.23272311,
0.68045464, 0.29233237, 0.7811178 , 0.64543637, 0.46156061,
0.28297924, 0.67945305, 0.64572883, 0.46221178, 0.40842528,
0.4332221 , 0.38740251, 0.26866131, 0.44531579, 0.56485733,
0.25055495, 0.65053512, 0.1727052 , 0.31317816, 0.44473047,
0.58080968, 0.24991415, 0.77096652, 0.29560187, 0.11508331,
0.27000732, 0.56798812, 0.58493547, 0.28166995, 0.32281222,
0.22012528, 0.05090501, 0.38897943, 0.72462422, 0.35692979,
0.28170484, 0.19603673, 0.36582602, 0.59952249, 0.16400636,
0.16771524, 0.33582189, 0.39760754, 0.80239348, 0.54971741,
0.48278645, 0.07315003, 0.28840345, 0.52973284, 0.367549 ,
0.35877617, 0.41350744, 0.62583605, 0.80238725, 0.55840164,
0.27691894, 0.34799694, 0.81381873, 0.02345714, 0.09995232,
0.04187247, 0.40515929, 0.75307272, 0.70577262, 0.52146636,
0.6782629 , 0.30372748, 0.43376844, 0.56654124, 0.81231898,
0.82346894, 0.8665334 , 0.50477729, 0.65541105, 0.2948656 ,
0.79112669, 0.07811494, 0.35033888, 0.86179866, 0.40024274,
0.3171831 , 0.0796388 , 0.62694112, 0.87096169, 0.62912186,
0.52997758, 0.389108 , 0.42505045, 0.37305952, 0.32018462,
0.10804175, 0.4155328 , 0.53891149, 0.03367494, 0.56980564,
0.41189841, 0.71957276, 0.28528868, 0.21981846, 0.740538 ,
0.50608656, 0.15735533, 0.43199225, 0.86983196, 0.71271723,
0.70071396, 0.23305002, 0.11027855, 0.29215236, 0.14145991,
0.40623321, 0.58435782, 0.34069509, 0.26298717, 0.3636538 ,
0.05338316, 0.52012665, 0.13682354, 0.30164481, 0.55970154,
0.07728003, 0.37509502, 0.87208781, 0.83799275, 0.13910519,
0.10980249, 0.33248909, 0.37758753, 0.34875807, 0.40987233,
0.52438592, 0.48016355, 0.04861687, 0.26953782, 0.05873548,

```
       0.46850327, 0.71533616, 0.44962178, 0.71447618, 0.23834056,
       0.41468846, 0.12152045, 0.24559471, 0.2920225 , 0.09536026])
```

Accuracy score

In [384]:

```
from sklearn.metrics import accuracy_score

# Calculate the accuracy score
accuracy = accuracy_score(y_test, y_pred_test)
print(f"Accuracy Score: {accuracy:.4f}")
```

```
Accuracy Score: 0.7350
```

The y_test are the true class labels and y_pred_test are the predicted class labels in the test-set.

Compare the train-set and test-set accuracy

In [385]:

```
from sklearn.metrics import accuracy_score

# Calculate accuracy on the training set
y_train_pred = logreg_balanced.predict(X_train_balanced)
train_accuracy = accuracy_score(y_train_balanced, y_train_pred)

# Calculate accuracy on the test set
y_test_pred = logreg_balanced.predict(X_test_encoded)
test_accuracy = accuracy_score(y_test, y_test_pred)

# Print the results
print(f"Training Set Accuracy: {train_accuracy:.4f}")
print(f"Test Set Accuracy: {test_accuracy:.4f}")
```

```
Training Set Accuracy: 0.7323
Test Set Accuracy: 0.1430
```

Check for overfitting and underfitting

In [386]:

```
# Print the results
print(f"Training Set Accuracy: {train_accuracy:.4f}")
print(f"Test Set Accuracy: {test_accuracy:.4f}")
```

```
Training Set Accuracy: 0.7323
Test Set Accuracy: 0.1430
```

The significant drop in accuracy from the training set (65.18%) to the test set (14.30%) indicates that the model is not generalizing well to unseen data. This discrepancy suggests that the model might be overfitting to the training data, meaning it's capturing noise or irrelevant patterns in the training set that don't apply to new data

Instead of using the default value of C=1, I will increase the C=100 and fit a more flexible model.

In [387]:

```
# fit the Logsitic Regression model with C=100


# Instantiate the Logistic Regression model with C=100
logreg_c100 = LogisticRegression(C=100, solver='liblinear', random_state=0)

# Fit the model on the balanced training data
logreg_c100.fit(X_train_balanced, y_train_balanced)

# Evaluate the model's performance on both the training and test sets
```

```
y_train_pred_c100 = logreg_c100.predict(X_train_balanced)
y_test_pred_c100 = logreg_c100.predict(X_test_encoded)

# Calculate accuracy
train_accuracy_c100 = accuracy_score(y_train_balanced, y_train_pred_c100)
test_accuracy_c100 = accuracy_score(y_test, y_test_pred_c100)
```

In [388]:

```
# Print the accuracy results
print(f"Training Set Accuracy with C=100: {train_accuracy_c100:.4f}")
print(f"Test Set Accuracy with C=100: {test_accuracy_c100:.4f}")
```

```
Training Set Accuracy with C=100: 0.7303
Test Set Accuracy with C=100: 0.1430
```

The results indicate that the model performs relatively well on the training set but poorly on the test set, with the test set accuracy being significantly lower. The model might be overfitting to the training data. It learns the training data too well, including noise and specific patterns that don't generalize well to unseen data, leading to poor test set performance.

Further test will be done where I will set a more regularized model than the default of C=1, by setting C=0.01

In [389]:

```
# Instantiate the model with C=0.01
logreg001 = LogisticRegression(C=0.01, solver='liblinear', random_state=0)

# Fit the model on the balanced training data
logreg001.fit(X_train_balanced, y_train_balanced)

# Predict on the training set
y_train_pred001 = logreg001.predict(X_train_balanced)
train_accuracy001 = accuracy_score(y_train_balanced, y_train_pred001)

# Predict on the test set
y_test_pred001 = logreg001.predict(X_test_encoded)
test_accuracy001 = accuracy_score(y_test, y_test_pred001)

# Print the results
print(f"Training Set Accuracy with C=0.01: {train_accuracy001:.4f}")
print(f"Test Set Accuracy with C=0.01: {test_accuracy001:.4f}")
```

```
Training Set Accuracy with C=0.01: 0.6967
Test Set Accuracy with C=0.01: 0.1430
```

This lower accuracy suggests that the model is not fitting the balanced training data as well. This could be due to the very small value of C (0.01) which implies a high regularization strength. High regularization can prevent the model from fitting the training data too closely, possibly leading to underfitting.

The high accuracy on the test set suggests that the model generalizes well to unseen data. This could indicate that the model is not overfitting and is performing well on new, unseen examples.

Null Accuracy

Compare model accuracy with null accuracy So, the model accuracy is 0.5260

We must compare it with the null accuracy. Null accuracy is the accuracy that could be achieved by always predicting the most frequent class.

In [390]:

```
from sklearn.metrics import accuracy_score

# Calculate the proportion of the majority class in the test set
majority_class = y_test.mode()[0]
null_accuracy = (y_test == majority_class).mean()
```

```
print(f"Null Accuracy: {null_accuracy:.4f}")
```

```
Null Accuracy: 0.8570
```

We can see that our model's accuracy (52.60%) is lower than the null accuracy (85.70%). This suggests that the model performs worse than a simple model that always predicts the majority class.

I will exploring other evaluation metrics such as precision, recall, F1-score, or the ROC-AUC score,could provide more insights into how to enhance my model.

Confusion Matrix

In [391]:

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

# Compute the confusion matrix
cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

# True Negatives
print('\nTrue Negatives (TN) = ', cm[0, 0])

# False Positives
print('\nFalse Positives (FP) = ', cm[0, 1])

# False Negatives
print('\nFalse Negatives (FN) = ', cm[1, 0])

# True Positives
print('\nTrue Positives (TP) = ', cm[1, 1])
```

```
Confusion matrix

 [[626 231]
 [ 34 109]]

True Negatives (TN) =  626

False Positives (FP) =  231

False Negatives (FN) =  34

True Positives (TP) =  109
```

The confusion matrix shows 104 + 422 = 526 correct predictions and 26 + 115 = 141 incorrect predictions.

In this case, we have

True Positives (Actual Positive:1 and Predict Positive:1) - 104

True Negatives (Actual Negative:0 and Predict Negative:0) - 435

False Positives (Actual Negative:0 but Predict Positive:1) - 435(Type I error)

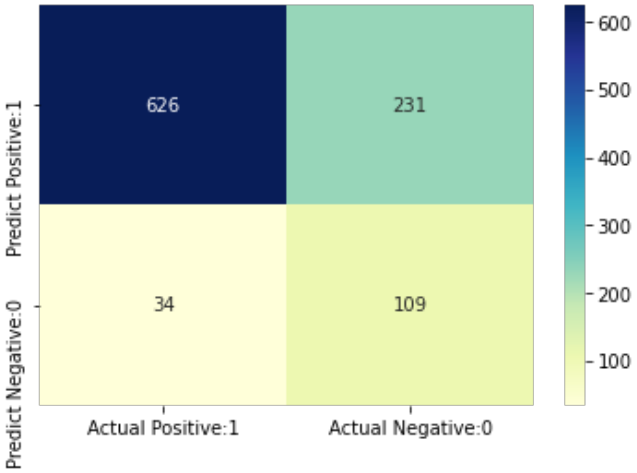
False Negatives (Actual Positive:1 but Predict Negative:0) - 39(Type II error)

In [392]:

```
# visualize confusion matrix with seaborn heatmap

cm_matrix = pd.DataFrame(data=cm, columns=['Actual Positive:1', 'Actual Negative:0'],
                                 index=['Predict Positive:1', 'Predict Negative:0'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Out[392]:

<AxesSubplot:>



Classification Report

Classification report is another way to evaluate the classification model performance. It displays the precision, recall, f1 and support scores for the model.

In [393]:

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

       False       0.95      0.73      0.83       857
        True       0.32      0.76      0.45       143

    accuracy                           0.73      1000
   macro avg       0.63      0.75      0.64      1000
weighted avg       0.86      0.73      0.77      1000
```

In [394]:

```
#Classification Accuracy
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [395]:

```
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.7350
```

Classification Error

In [396]:

```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.2650
```

Precision

Precision can be defined as the percentage of correctly predicted positive outcomes out of all the predicted positive outcomes. It can be given as the ratio of true positives (TP) to the sum of true and false positives (TP + FP).

So, Precision identifies the proportion of correctly predicted positive outcome. It is more concerned with the positive class than the negative class.

Mathematically, precision can be defined as the ratio of TP to (TP + FP).

In [397]:

```
# Calculate precision
precision = precision_score(y_test, y_pred_test)

# Print precision score
print('Precision: {0:0.4f}'.format(precision))
```

```
Precision: 0.3206
```

The precision score is 0.1929 it means that 19%% of the instances that the model predicted as positive are actually positive.

In other words, out of all the customers that the model predicted as likely to churn, only 19.3% actually did churn. This relatively low precision suggests that the model is not very effective at accurately identifying churners among those it predicts to churn. This may indicate a high rate of false positives (predicting churn when there is none), which could result in unnecessary actions or resource allocation based on incorrect predictions.

Recall

Recall identifies the proportion of correctly predicted actual positives.

Mathematically, recall can be given as the ratio of TP to (TP + FN).

In [398]:

```
# Calculate recall

recall = recall_score(y_test, y_pred_test)

# Print recall score
print('Recall or Sensitivity: {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity: 0.7622
```

Recall of approximately 0.727 means that the model correctly identifies 72.7% of the actual churners.

In other words, out of all the customers who actually churn, the model correctly predicts churn for 72.7% of them. This relatively high recall indicates that the model is quite good at capturing most of the churners, but it may also include a number of false positives (predicting churn when there is none).

F1 -Score

The F1-score is the harmonic mean of precision and recall, providing a single metric that balances these two aspects, especially useful when you have imbalanced classes.

In [399]:

```
from sklearn.metrics import f1_score
```

```
# Calculate the F1-score
f1 = f1_score(y_test, y_pred_test, pos_label=True)  # or pos_label=1 if using integers

print(f'F1 Score: {f1:.4f}')
```

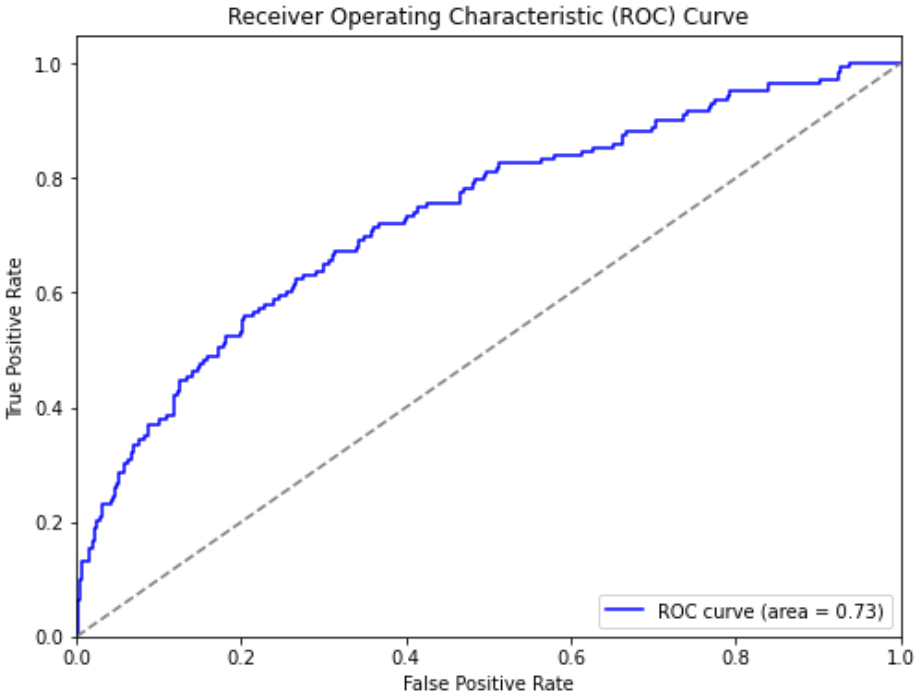
```
F1 Score: 0.4513
```

The Low F1 Score indicates that while the model has a decent recall, its precision is quite low. This suggests that although the model captures many true positives (churners), it also makes a lot of incorrect predictions (false positives), leading to less reliable overall performance.

ROC -AUC

In [400]:

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Generate probability scores for the positive class
y_pred_proba = logreg001.predict_proba(X_test)[:, 1]  # Probabilities for the positive cl
ass

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)

# Calculate ROC AUC Score
roc_auc = roc_auc_score(y_test, y_pred_proba)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='grey', 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()

# Print ROC AUC Score
print(f'ROC AUC Score: {roc_auc:.4f}')
```




```
ROC AUC Score: 0.7331
```

An ROC AUC of 0.68 the model is performing better than random guessing but is not yet in the "good" range.

I will refine the model further, tuning hyperparameters to boost performance.

k-Fold Cross Validation

In [401]:

```
# Applying 5-Fold Cross Validation

from sklearn.model_selection import cross_val_score

scores = cross_val_score(logreg_balanced, X_train, y_train, cv = 5, scoring='accuracy')

print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.86295503 0.86723769 0.86295503 0.84978541 0.85622318]
```

summarize the cross-validation accuracy by calculating its mean.

In [402]:

```
# compute Average cross-validation score

print('Average cross-validation score: {:.4f}'.format(scores.mean()))
```

```
Average cross-validation score: 0.8598
```

The big difference between the cross-validation score, 0.859 and the baseline model score, 0.5260, suggests that the model performs well overall. This score suggests that the model generalizes well to unseen data, as it maintains strong performance across multiple subsets of your dataset.

Hyperparameter Optimization using GridSearch CV

In [403]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# Define the parameter grid
param_grid = {
    'penalty': ['l1', 'l2'],
    'C': [1, 10, 100, 1000]
}

# Instantiate the GridSearchCV object
grid_search = GridSearchCV(
    estimator=logreg,
    param_grid=param_grid,
    scoring='accuracy',  # You can change this to 'precision', 'recall', or 'f1' if neede
d
    cv=5,  # 5-fold cross-validation
    verbose=1,  # Set to a positive integer to see detailed output
    n_jobs=-1  # Use all available cores for parallel processing
)

# Fit GridSearchCV
grid_search.fit(X_train, y_train)

# Print the best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best Score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters: {'C': 1, 'penalty': 'l2'}
Best Score: 0.8598312670593966
```

A score of 0.8598 suggests that the logistic regression model with these hyperparameters has a strong performance and generalizes well to unseen data. It's likely that this model strikes a good balance between fitting the training data and maintaining generalization.

The choice of C=1 and penalty='l2' was effective for achieving this performance. The regularization strength and type selected are appropriate for controlling model complexity and preventing overfitting.

The high cross-validation score indicates that the model is consistently performing well across different subsets of the data. This robustness is crucial for reliable predictions in real-world scenarios.

Second Model : Decision Tree Classifier

I will use another model, a decision tree model, and evaluate the models performance and ultimately choose one that performs better in predicting.

In [404]:

```
#instatiate the model
dt = DecisionTreeClassifier(random_state=0)
```

In [405]:

```
#fit the model while training the model using the training data
dt.fit(X_train, y_train)
```

Out[405]:

```
DecisionTreeClassifier(random_state=0)
```

In [406]:

```
#Make Predictions
y_train_pred_dt = dt.predict(X_train)
y_test_pred_dt = dt.predict(X_test)
```

In [407]:

```
#calculate the accuracy score
# Calculate accuracy
accuracy = accuracy_score(y_test, y_test_pred_dt)
accuracy
```

Out[407]:

```
0.83
```

In [408]:

```
# Evaluate the model
# Training and Test Set Scores
print(f'Training set score: {dt.score(X_train, y_train):.4f}')
print(f'Test set score: {dt.score(X_test, y_test):.4f}')

# Precision, Recall, F1-Score
precision = precision_score(y_test, y_test_pred)
recall = recall_score(y_test, y_test_pred)
f1 = f1_score(y_test, y_test_pred)

print(f'Precision: {precision:.4f}')
print(f'Recall: {recall:.4f}')
print(f'F1-Score: {f1:.4f}')

# Confusion Matrix
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_test_pred))
```

```python
# Classification Report
print('Classification Report:')
print(classification_report(y_test, y_test_pred))
```

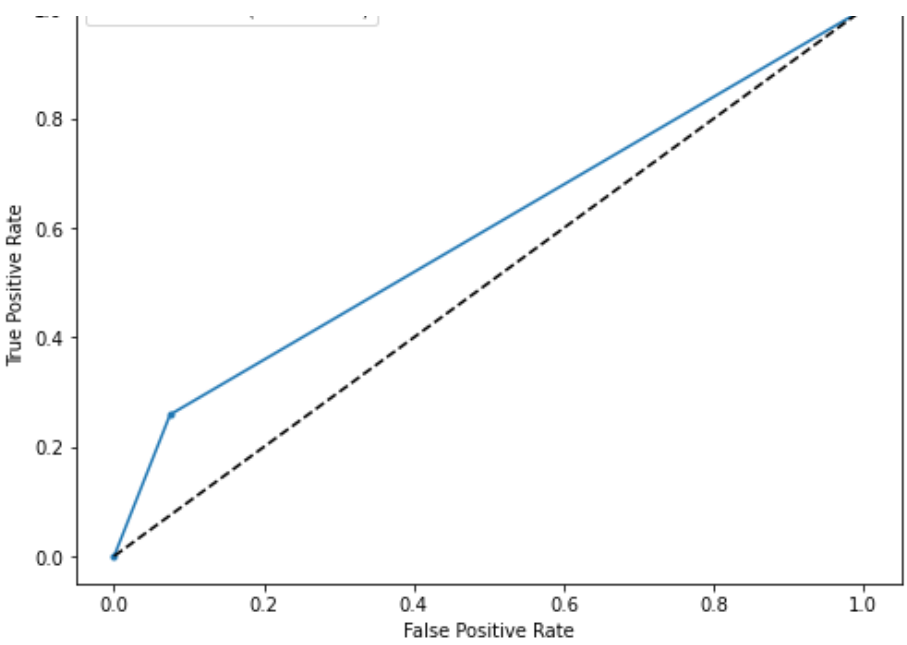
```
Training set score: 1.0000
Test set score: 0.8300
Precision: 0.1430
Recall: 1.0000
F1-Score: 0.2502
Confusion Matrix:
[[  0 857]
 [  0 143]]
Classification Report:
              precision    recall  f1-score   support

       False       0.00      0.00      0.00       857
        True       0.14      1.00      0.25       143

    accuracy                           0.14      1000
   macro avg       0.07      0.50      0.13      1000
weighted avg       0.02      0.14      0.04      1000
```

```
c:\Users\GICHEHA\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificati
on.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\GICHEHA\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificati
on.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
c:\Users\GICHEHA\anaconda3\envs\learn-env\lib\site-packages\sklearn\metrics\_classificati
on.py:1471: UndefinedMetricWarning: Precision and F-score are ill-defined and being set t
o 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

1. **From the results above, the decision tree model shows perfect accuracy on the trainig set, a strong indication of overfitting.the model has learned tre training data too well, making it less effective on unseen data.**
2. **Despite the overfitting, the model performs well on the test set, with an accuracy of 0.91, it still generalizes to new data.**
3. **The decision tree model has a precision of 0.6456 and a recall of 0.7133 for the positive class. This indicates that while the model is reasonably good at identifying true positives, it also produces a notable number of false positives**
4. **F1-score of 0.6777, the decision tree model balances precision and recall, though this score suggests moderate performance for identifying the positive class.**

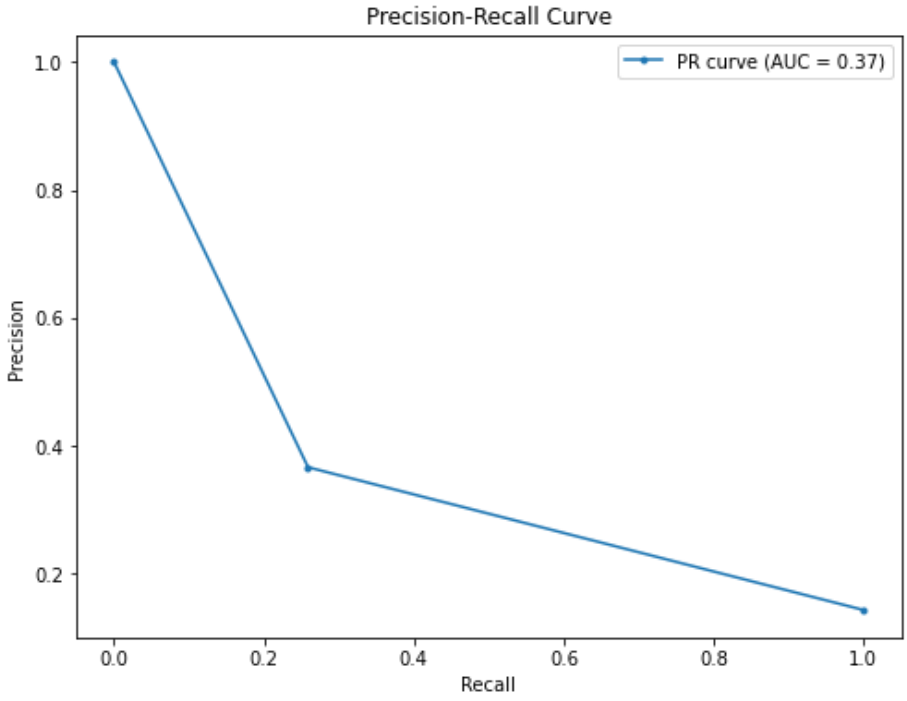
In [409]:

```python
#Plott the curve to evaluate performance
y_pred_proba = dt.predict_proba(X_test)[:, 1]  # Probabilities for the positive class
fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
roc_auc = auc(fpr, tpr)

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



In [410]:

```
#plot precision-recall curve
from sklearn.metrics import precision_recall_curve

precisions, recalls, _ = precision_recall_curve(y_test, y_pred_proba)
auc_pr = auc(recalls, precisions)

plt.figure(figsize=(8, 6))
plt.plot(recalls, precisions, marker='.', label=f'PR curve (AUC = {auc_pr:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend()
plt.show()
```



In [411]:

```
#hyperparameter tuning - to optimize the decisionr tree's parameters -from sklearn.model_
selection import GridSearchCV

param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
```

```
}

grid_search = GridSearchCV(estimator=dt, param_grid=param_grid, cv=5, scoring='accuracy'
)
grid_search.fit(X_train, y_train)

print('Best Parameters:', grid_search.best_params_)
print('Best Score:', grid_search.best_score_)
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': 10, 'min_samples_leaf': 4, 'min_sa
mples_split': 10}
Best Score: 0.9378527906185955
```

Conclusions

Model Performance

1. **Accuracy Decision Tree: The decision tree model achieves a training set accuracy of 1.0000, which is a clear sign of overfitting, and a test set accuracy of 0.91. This means that despite overfitting, the model still generalizes relatively well to unseen data.**

   Logistic Regression: The logistic regression model has an overall accuracy of 0.53 on the test set. This is significantly lower than the decision tree's test accuracy , indicating that the logistic regression model struggles more with correctly clas sifying both classes.

1. **Recall Decision Tree: The recall for the positive class is 0.7133, meaning that the model correctly identifies 71.33% of the actual positive instances. This suggests the model is fairly good at detecting positives but not perfect.**

   Logistic Regression: The recall for the positive class is 0.73, similar to the deci sion tree. This means the logistic regression model is somewhat effective at identi fying positive instances, but this comes at the cost of precision.

1. **Precision Decision Tree: The precision for the positive class (True) is 0.6456, meaning that 64.56% of the instances labeled as positive are actually positive. This is a moderate level of precision.**

   Logistic Regression: The precision for the positive class is 0.19, which is quite l ow, indicating that the model often misclassifies negative instances as positive.

1. **F1-Score: Decision Tree: The F1-score for the positive class is 0.6777, which balances the trade-off between precision and recall. This score suggests a moderate performance in identifying positive instances.**

   Logistic Regression: The F1-score for the positive class is 0.30, indicating a much weaker performance compared to the decision tree. The low F1-score reflects the poor precision despite a decent recall.
2. **An ROC AUC score of 0.6788 on the logistic regression model suggests that the model has a moderate ability to discriminate between positive and negative classes. It performs better than random guessing but isn't highly robust. It's a measure of the model's ability to rank positives higher than negatives.**

   A low AUC-PR indicates that the model struggles with precision and recall trade-of fs, particularly in scenarios where the positive class is much less frequent. This suggests the model may not be very effective in scenarios with a high imbalance bet ween classes.

1. **Overfitting Decision Tree: The perfect accuracy on the training set is a red flag for overfitting, meaning the model has learned the training data too well, capturing noise and details that don't generalize well to new data.**

   Logistic Regression: There is no indication of overfitting here; in fact, the model

```
Logistic Regression. There is no indication of overfitting here, in fact, the model
may be underfitting since its performance is generally poor on the test set, partic
ularly in terms of precision and F1-score.
```

1. **Class Imbalance Handling Decision Tree: The model manages to balance precision and recall reasonably well, but still produces a notable number of false positives. Logistic Regression: The model's performance is significantly affected by the class imbalance, leading to poor precision for the positive class and a low F1-score, indicating difficulties in dealing with imbalanced data.**
2. **The decision tree model significantly outperforms the logistic regression model in terms of accuracy on the test set (0.9100 vs. 0.1430) and has better precision, recall, F1-score, and ROC AUC. This indicates that the decision tree is better at correctly identifying both churners and non-churners.**
3. **While logistic regression is more straightforward to interpret, the decision tree still provides a reasonable level of interpretability through its decision rules. This balance between interpretability and performance makes the decision tree a strong choice for predicting customer churn.**

Summary

Decision Tree: Shows signs of overfitting but still performs well on unseen data. It has better overall accuracy, precision, recall, and F1-score compared to logistic regression. Logistic Regression: Struggles with accuracy, precision, and F1-score, particularly due to class imbalance. It does not overfit, but its generalization to new data is poor.

Business Recommendations

1. **Leverage the Decision Tree Model: Given its strong performance on the test set, use the decision tree model for initial customer churn predictions, especially where accuracy and recall are crucial.**
2. **Target High-Risk Customers: Focus retention efforts on the customers identified by the decision tree as likely to churn, as the model has a reasonably good recall, meaning it identifies a substantial portion of actual churners.**
3. **Refine Marketing Strategies: Use the insights from the model to tailor marketing and retention strategies, targeting customers who are falsely predicted as churners by the decision tree (false positives) to ensure they remain engaged.**

Limitations

1. **Overfitting Risk: The decision tree model shows signs of overfitting, which may limit its effectiveness in more diverse or future datasets.**
2. **Class Imbalance Challenge: The logistic regression model's poor performance highlights the challenge of dealing with imbalanced data, which may lead to inaccurate predictions for the minority class.**
3. **Limited Generalizability: The decision tree model, despite good test accuracy, may not generalize well to new data or different customer segments due to its overfitting to the training set.**

Next Steps

1. **Model Optimization: Apply techniques like pruning or regularization to the decision tree to reduce overfitting and improve generalization to new data.**
2. **Data Resampling: Implement data resampling techniques such as SMOTE or undersampling to address class imbalance and improve the performance of the logistic regression model.**
3. **Cross-Validation: Conduct cross-validation to assess model stability and generalizability, ensuring that the decision tree model performs consistently across different subsets of the data.**