SyriaTel Customer Churn: A Data-Driven Analysis for Strategic Intervention

Business Understanding

Customer churn is one of the most pressing challenges for telecommunications companies, as retaining existing customers is significantly more cost-effective than acquiring new ones. SyriaTel, like many telecom providers, experiences a steady loss of customers who discontinue their services.

SyriaTel, a telecom provider, wants to understand **which customers are most likely to churn** and **what factors contribute to churn**. By predicting churn early, the company can design retention strategies such as personalized offers, improved customer service, or plan adjustments to reduce customer loss.

Stakeholders

- **SyriaTel Management:** Interested in reducing churn rates to improve profitability.
- **Customer Retention Team:** Needs insights to design effective interventions for possible churners.

The key question is:
“Can we predict which customers are likely to churn, and what behaviors signal higher churn risk?”

Problem statement

Currently, SyriaTel lacks a reliable method to **predict which customers are at risk of leaving**. Without such a system, the company cannot take proactive steps to retain customers, leading to revenue loss and reduced market competitiveness.

Goals and objectives

Goal

The primary goal of this project is to develop a **machine learning classification model** that predicts whether a customer will churn, enabling SyriaTel to take proactive actions that reduce churn and improve profitability.

Objectives

To achieve this goal, the project will:

1. **Build predictive models** using customer account details, usage behavior, and service interactions.
2. **Identify key factors** that contribute most to churn, such as call usage, international plans, and customer service calls.
3. **Evaluate model performance** using appropriate classification metrics (precision,F1_score and recall).
4. **Provide actionable recommendations** for SyriaTel's customer retention strategies based on model insights.

Metric of success

The success of this project will be evaluated using the following **classification metrics** and their operational impact:

- **F1 Score:** The primary metric, balancing precision and recall, ensures the model effectively identifies customers likely to churn while minimizing false positives.
- **Precision:** Critical for retention campaigns, high precision ensures interventions target only customers most likely to churn, optimizing marketing and support resources.
- **Business Impact:** Success is measured not only by model metrics but also by actionable outcomes, such as reduced churn rates, improved retention in high-risk segments, and cost savings from targeted interventions.

Thresholds for Success:

- **F1 Score ≥ 75%**
- **Precision ≥ 80%**

2: Data Undestanding

2.1: Data Overview

The dataset used in this analysis is sourced from the **SyriaTel Customer Churn Dataset** on Kaggle (https://www.kaggle.com/becksddf/churn-in-telecoms-dataset). It contains information on **3,333 customers** of SyriaTel, a telecommunications company. The dataset is designed for a **binary classification problem** with the **Target variable:** `churn` (whether a customer left the company — `True` / `False`).

The core of the dataset is customer account details, usage behavior (calls, minutes, charges), and interactions with customer service. Together, these variables enable robust analysis of churn drivers and prediction.

Column Name Meanings

Column Name	Meaning
`state`	The U.S. state the customer resides in.
`account length`	Number of days the account has been active.
`area code`	Customer’s assigned telephone area code.
`phone number`	Customer’s phone number (unique identifier).
`international plan`	Whether the customer has an international calling plan (`yes` / `no`).
`voice mail plan`	Whether the customer has a voicemail plan (`yes` / `no`).
`number vmail messages`	Number of voicemail messages recorded.
`total day minutes`	Total number of minutes of calls made during the day.
`total day calls`	Total number of calls made during the day.
`total day charge`	Total charges for daytime calls.
`total eve minutes`	Total minutes of evening calls.
`total eve calls`	Total number of evening calls.
`total eve charge`	Total charges for evening calls.
`total night minutes`	Total minutes of night calls.

Column Name	Meaning
total night calls	Total number of night calls.
total night charge	Total charges for night calls.
total intl minutes	Total minutes of international calls.
total intl calls	Total number of international calls.
total intl charge	Total charges for international calls.
customer service calls	Number of calls made to customer service.

2.2: Data Description

2.2.1: Importing the dataset

In [1]:
```
#importing the necessary libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
import re
#sklearn libraries
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, roc_auc_score
from sklearn.model_selection import StratifiedKFold, cross_validate
from sklearn.metrics import make_scorer, average_precision_score,classification_report
```

In [2]:

```
#Reading the dataset and checking top five rows and last five rows
data = pd.read_csv('original_data/churn_telecom.csv')
data
```

Out[2]:

	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	tota nigh charge
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83	12.56
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123	8.61
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91	8.64
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137	6.26
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77	10.86

3333 rows × 21 columns

2.2.2: Basic Structure

In [3]:
```
#data shape
data.shape
```

Out[3]:
```
(3333, 21)
```

In [4]:
```
#column names
data.columns
```

Out[4]:
```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail messages',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charge',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

2.2.3: Overview of column types and non-null values

In [5]:
```
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
```

2.2.4: Summary statistics numerical

In [6]:

```
data.describe(include='number').T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
account length	3333.0	101.064806	39.822106	1.00	74.00	101.00	127.00	243.00
area code	3333.0	437.182418	42.371290	408.00	408.00	415.00	510.00	510.00
number vmail messages	3333.0	8.099010	13.688365	0.00	0.00	0.00	20.00	51.00
total day minutes	3333.0	179.775098	54.467389	0.00	143.70	179.40	216.40	350.80
total day calls	3333.0	100.435644	20.069084	0.00	87.00	101.00	114.00	165.00
total day charge	3333.0	30.562307	9.259435	0.00	24.43	30.50	36.79	59.64
total eve minutes	3333.0	200.980348	50.713844	0.00	166.60	201.40	235.30	363.70
total eve calls	3333.0	100.114311	19.922625	0.00	87.00	100.00	114.00	170.00
total eve charge	3333.0	17.083540	4.310668	0.00	14.16	17.12	20.00	30.91
total night minutes	3333.0	200.872037	50.573847	23.20	167.00	201.20	235.30	395.00
total night calls	3333.0	100.107711	19.568609	33.00	87.00	100.00	113.00	175.00
total night charge	3333.0	9.039325	2.275873	1.04	7.52	9.05	10.59	17.77
total intl minutes	3333.0	10.237294	2.791840	0.00	8.50	10.30	12.10	20.00
total intl calls	3333.0	4.479448	2.461214	0.00	3.00	4.00	6.00	20.00
total intl charge	3333.0	2.764581	0.753773	0.00	2.30	2.78	3.27	5.40
customer service calls	3333.0	1.562856	1.315491	0.00	1.00	1.00	2.00	9.00

2.2.5: Summary statistics categorical

In [7]:
```
data.describe(include='O')
```

Out[7]:

	state	phone number	international plan	voice mail plan
count	3333	3333	3333	3333
unique	51	3333	2	2
top	WV	382-4657	no	no
freq	106	1	3010	2411

2.2.6: Missing values

In [8]:
```
data.isna().mean()*100
```

Out[8]:
```
state                     0.0
account length            0.0
area code                 0.0
phone number              0.0
international plan        0.0
voice mail plan           0.0
number vmail messages     0.0
total day minutes         0.0
total day calls           0.0
total day charge          0.0
total eve minutes         0.0
total eve calls           0.0
total eve charge          0.0
total night minutes       0.0
total night calls         0.0
total night charge        0.0
total intl minutes        0.0
total intl calls          0.0
total intl charge         0.0
customer service calls    0.0
churn                     0.0
dtype: float64
```

2.2.7: Duplicates

In [9]:
```
data.duplicated().sum()
```

Out[9]: 0

2.3: Data Summary

The dataset consists of **3,333 customer records** with **21 attributes `[16 numeric, 4 categorical and 1 boolean]`**, capturing demographic details, account information, service usage, and churn status (whether a customer left or not). Features include customer state, account length, area code, subscription plans (international and voicemail), usage metrics (day, evening, night, international minutes/calls/charges), number of customer service calls, and the binary target variable `churn`.

Numerical summaries show that customers, on average, have an account length of about **101 days**, make around **100 calls per time period (day/evening/night)**, and use approximately **180–200 minutes daily**. International usage is relatively low with a mean equal to 10 minutes and 4 to 5 calls, while the average number of customer service calls is **1.6**, with some customers contacting up to **9 times**.

Categorical summaries reveal that most customers do not subscribe to an international plan (**~90% "no"**) or voicemail plan (**~72% "no"**). The dataset covers customers across **51 U.S. states**, but each phone number is unique, making it unsuitable as a predictive feature.

The target variable `churn` is imbalanced: **14.5% of customers churned (483 records)**, while **85.5% did not churn (2,850 records)**. This imbalance is important to consider when training classification models, as it may bias predictions toward the majority class.

The dataset contains **no missing values** or duplicates, making it clean and ready for modeling. This is highly relevant for our churn analysis, as it ensures the model can fully leverage all historical usage and interaction records without significant preprocessing.

Overall, the dataset provides a balanced mix of **service usage, subscription plan details, and customer interaction features**, which are directly relevant for predicting churn and identifying factors that drive customer retention.

3: Data Preparation

3.1: Data Cleaning

3.1.1: Making a copy of the data

In [10]:
```
#copy
df = data.copy()
df.shape
```

Out[10]: (3333, 21)

3.1.2: Dropping unpredictive column(s)

In [11]:
```
df.drop(columns=['phone number'], inplace=True)
df.shape
```

Out[11]: (3333, 20)

The decision to drop `phone number` column is because it contains unique identifiers for each customer and does not provide any meaningful information for predicting churn. Keeping it would lead to aspects such as overfitting

3.1.3: Formating column names

In [12]:
```
df.columns
```

Out[12]:
```
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 [13]:

```
#creating mapper
col_map = {
    'state': 'state',
    'account length': 'acc_length',
    'area code': 'area_code',
    'international plan': 'intl_plan',
    'voice mail plan': 'vmail_plan',
    'number vmail messages': 'num_vm_msgs',
    'total day minutes': 'ttl_day_mins',
    'total day calls': 'ttl_day_calls',
    'total day charge': 'ttl_day_charge',
    'total eve minutes': 'ttl_eve_mins',
    'total eve calls': 'ttl_eve_calls',
    'total eve charge': 'ttl_eve_charge',
    'total night minutes': 'ttl_ngt_mins',
    'total night calls': 'ttl_ngt_calls',
    'total night charge': 'ttl_ngt_charge',
    'total intl minutes': 'ttl_intl_mins',
    'total intl calls': 'ttl_intl_calls',
    'total intl charge': 'ttl_intl_charge',
    'customer service calls': 'cust_service_calls',
    'churn': 'churn'
}

# Applying mapping
df.rename(columns=col_map, inplace=True)
df.columns
```

Out[13]:

```
Index(['state', 'acc_length', 'area_code', 'intl_plan', 'vmail_plan',
       'num_vm_msgs', 'ttl_day_mins', 'ttl_day_calls', 'ttl_day_charge',
       'ttl_eve_mins', 'ttl_eve_calls', 'ttl_eve_charge', 'ttl_ngt_mins',
       'ttl_ngt_calls', 'ttl_ngt_charge', 'ttl_intl_mins', 'ttl_intl_calls',
       'ttl_intl_charge', 'cust_service_calls', 'churn'],
      dtype='object')
```

3.1.4: Checking Unique Items

In [14]:
```
for col in df.columns:
    print(f"Unique items in {col}:\n")
    print(df[col].unique())
    print("***********")
```

```
Unique items in state:

['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']
***********
Unique items in acc_length:

[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]
***********
Unique items in area_code:

[415 408 510]
***********
Unique items in intl_plan:

['no' 'yes']
***********
Unique items in vmail_plan:

['yes' 'no']
***********
Unique items in num_vm_msgs:

[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]
***********
Unique items in ttl_day_mins:

[265.1 161.6 243.4 ... 321.1 231.1 180.8]
***********
```

```
Unique items in ttl_day_calls:

[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]
***********
Unique items in ttl_day_charge:

[45.07 27.47 41.38 ... 54.59 39.29 30.74]
***********
Unique items in ttl_eve_mins:

[197.4 195.5 121.2 ... 153.4 288.8 265.9]
***********
Unique items in ttl_eve_calls:

[ 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]
***********
Unique items in ttl_eve_charge:

[16.78 16.62 10.3  ... 13.04 24.55 22.6 ]
***********
Unique items in ttl_ngt_mins:

[244.7 254.4 162.6 ... 280.9 120.1 279.1]
***********
Unique items in ttl_ngt_calls:

[ 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]
***********
Unique items in ttl_ngt_charge:

[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]
***********
Unique items in ttl_intl_mins:
```

```
[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]
***********
Unique items in ttl_intl_calls:

[ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13 18 14 16 20 17]
***********
Unique items in ttl_intl_charge:

[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]
***********
Unique items in cust_service_calls:

[1 0 2 3 4 5 7 9 6 8]
***********
Unique items in churn:

[False  True]
***********
```

3.1.5: Changing Datatype to object

In [15]:
```
df["churn"] = df["churn"].map({True: "Churned", False: "Not Churned"})
```

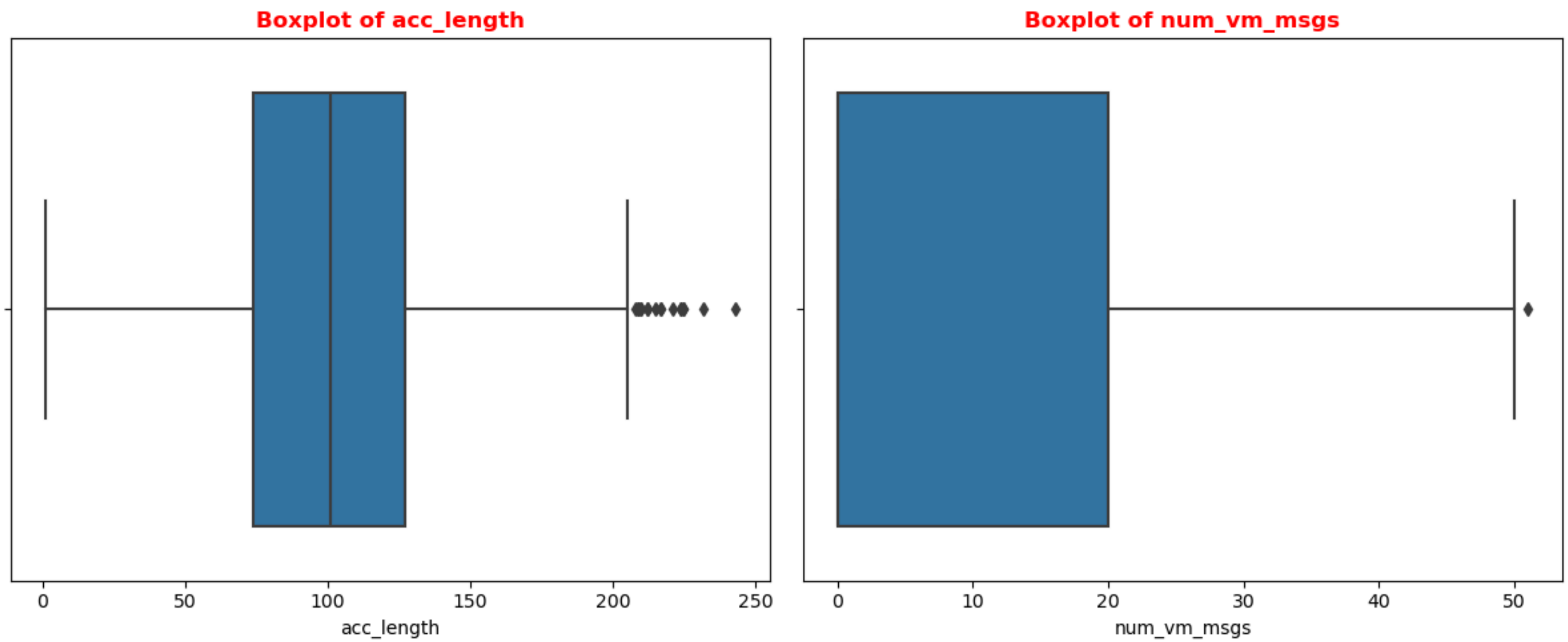
In [16]:
```
df['area_code'] = df['area_code'].astype(str)
```

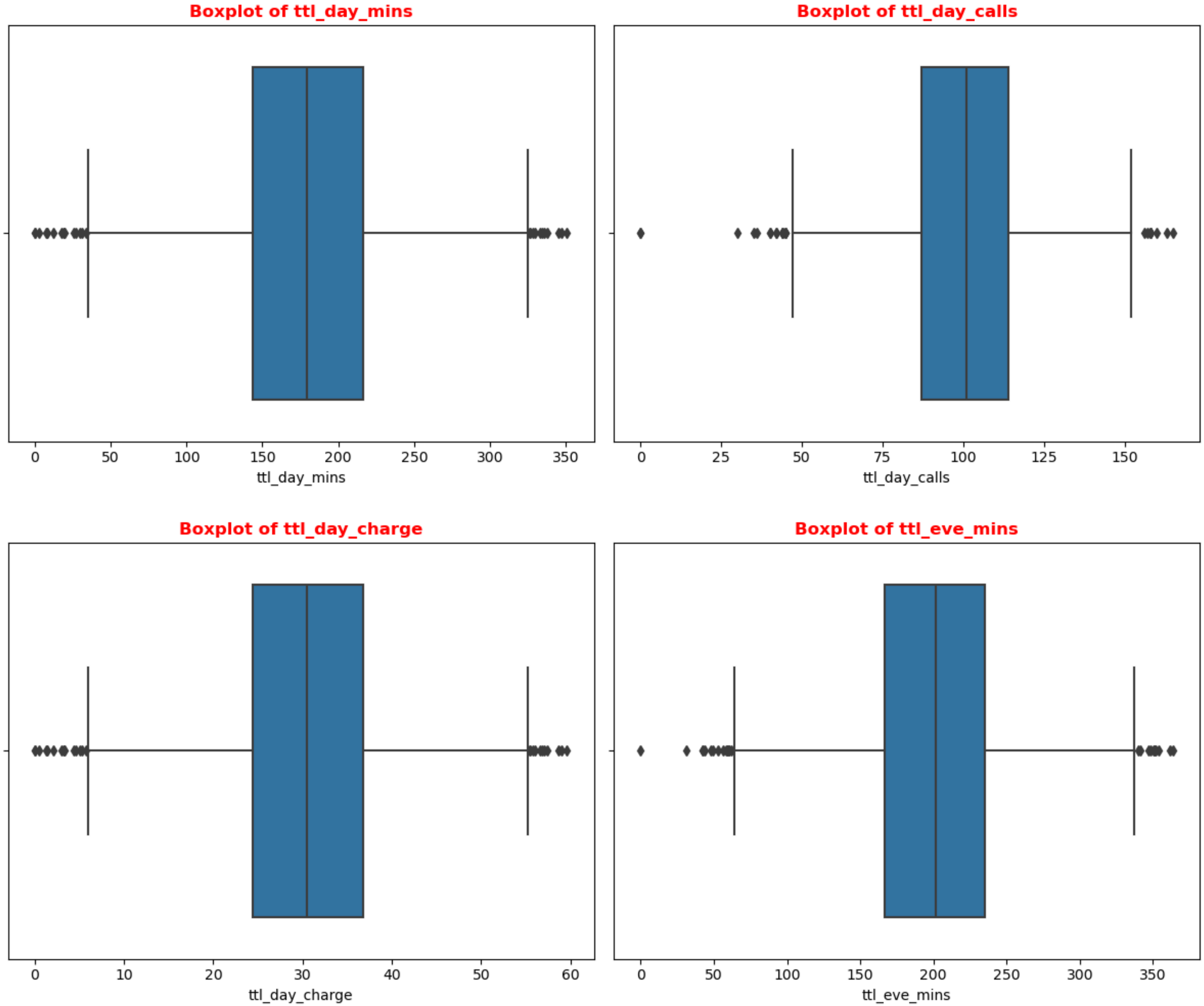
- **churn**: Converted from boolean (`True/False`) to categorical strings `"Churned"` and `"Not Churned"`).This makes the column more interpretable for humans and avoids plotting errors.
- **area_code**: Converted from numeric to string because it represents a **categorical label** rather than a quantitative value. Treating it as a string prevents meaningless arithmetic operations and ensures proper handling during **EDA** and **modeling**.

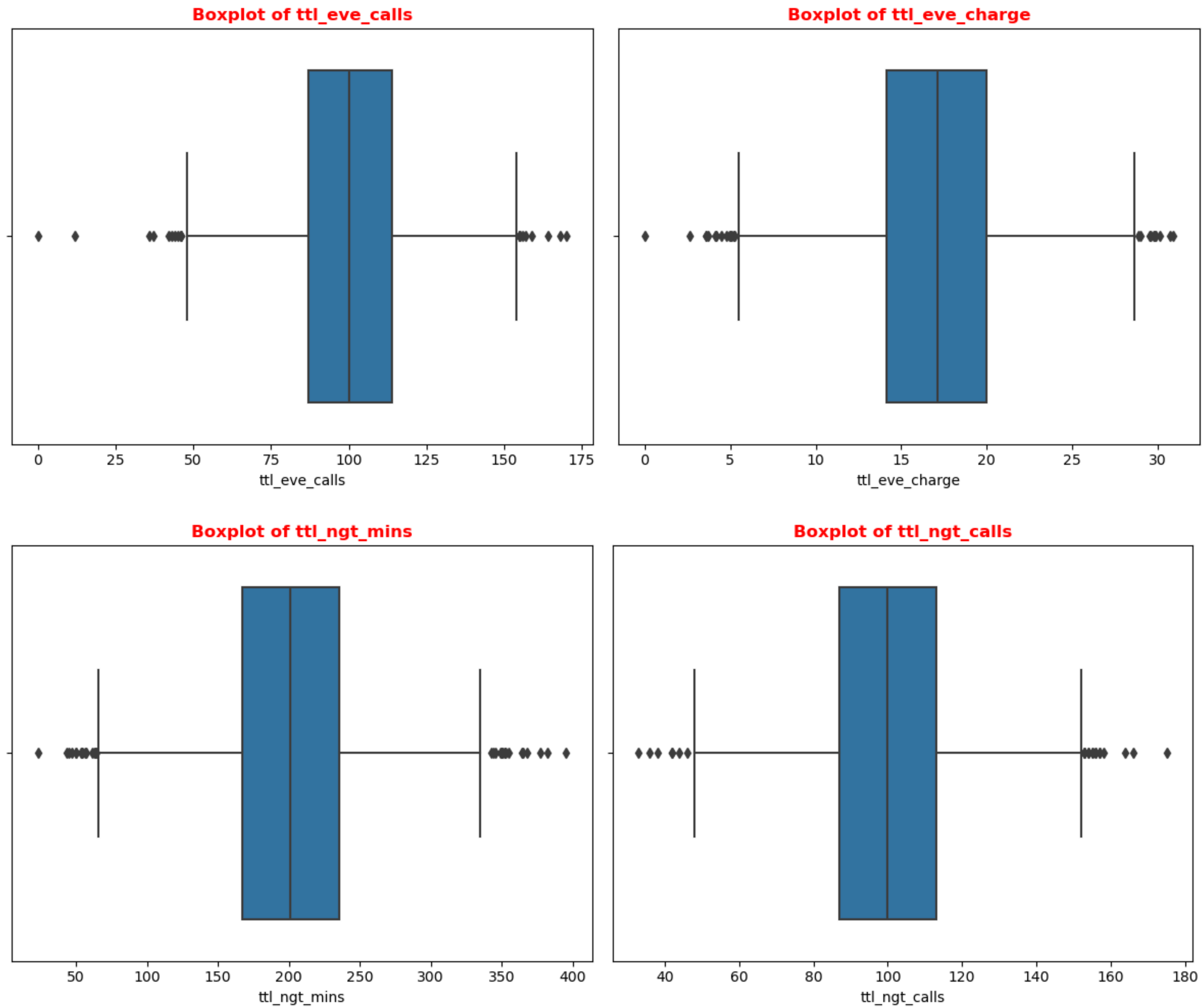
3.1.6: Checking for Outliers

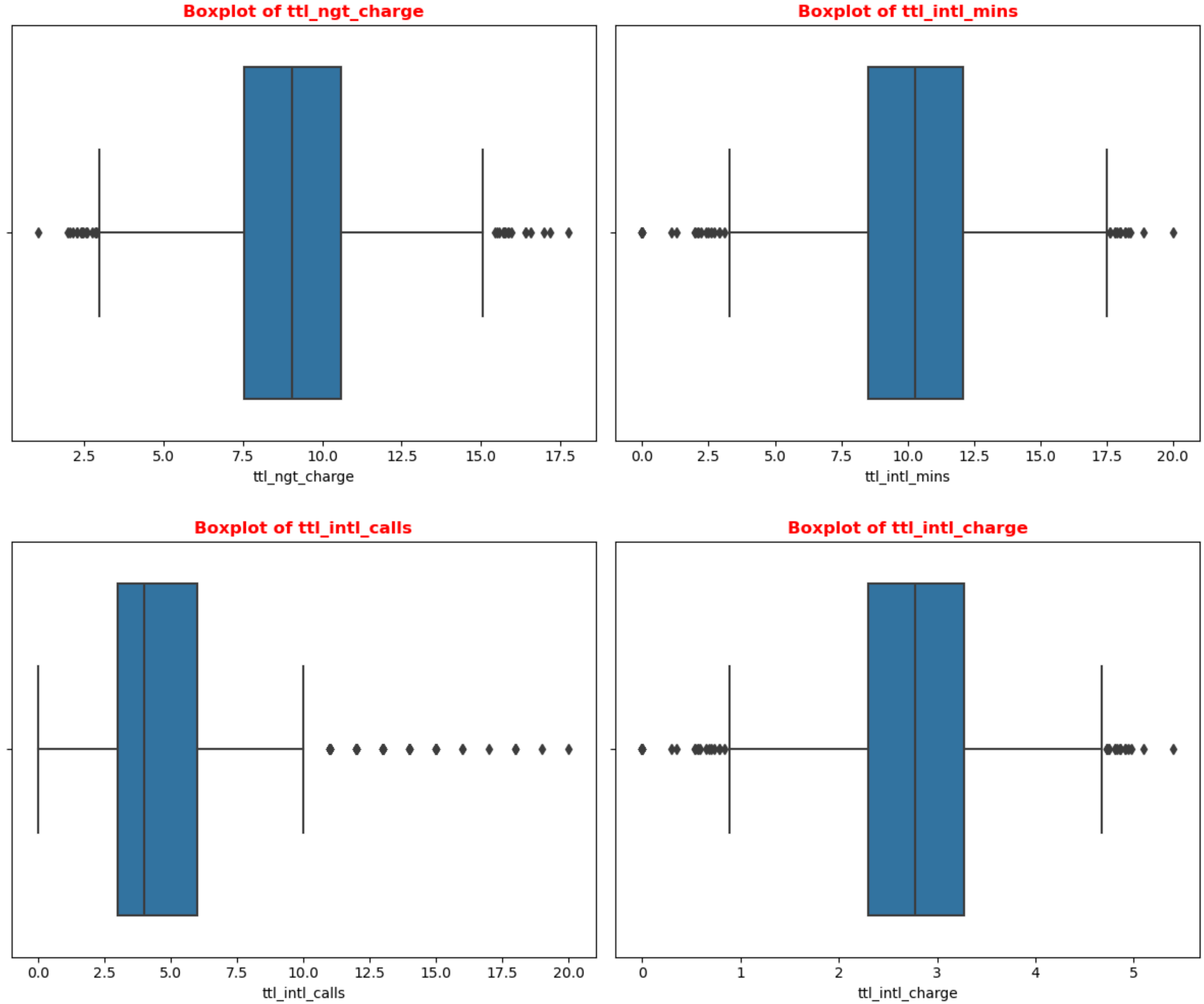
In [17]:
```
#storing numeric columns
num_cols = df.select_dtypes(include='number')
```

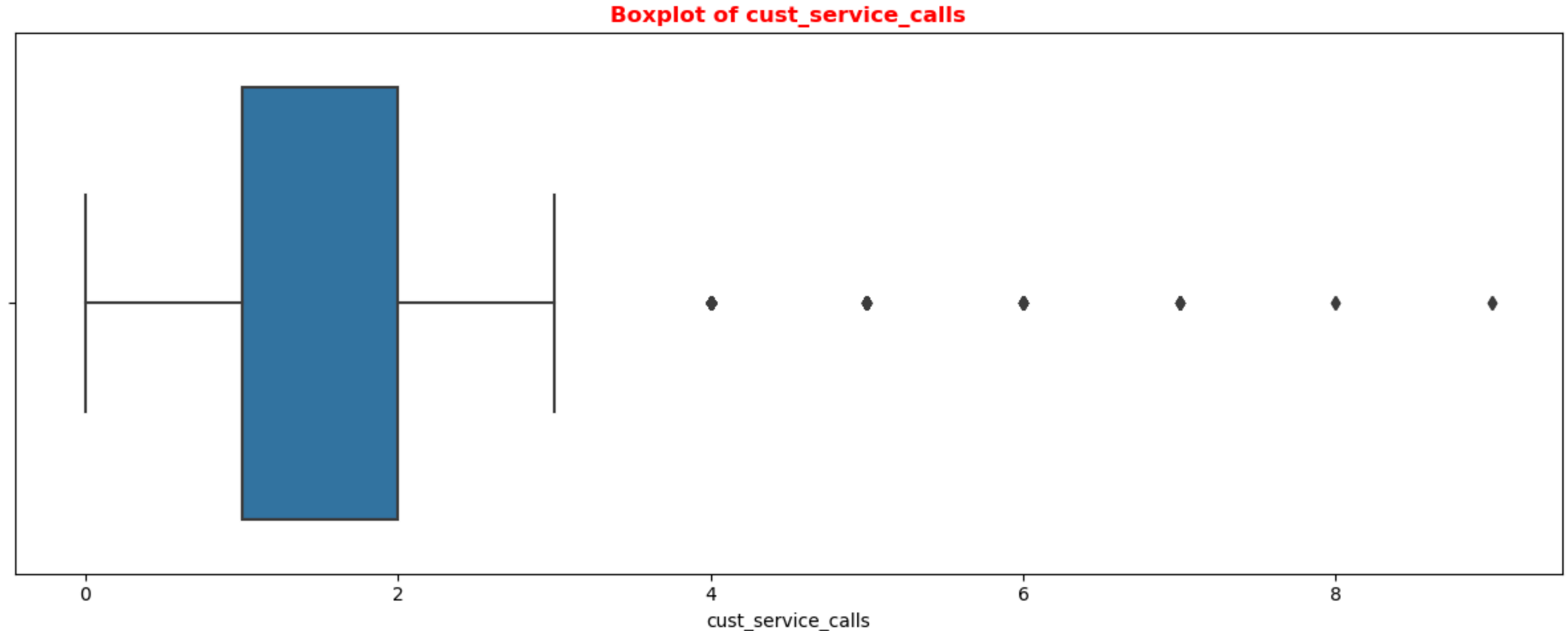
In [18]:
```
num_cols_list = num_cols.columns.tolist()
n = len(num_cols_list)
for i in range(0, n, 2):
    cols_pair = num_cols_list[i:i+2]
    fig, axes = plt.subplots(1, len(cols_pair), figsize=(12, 5))
    axes = np.atleast_1d(axes)
    for j, col in enumerate(cols_pair):
        sns.boxplot(x=num_cols[col], ax=axes[j])
        axes[j].set_title(f'Boxplot of {col}', fontweight='bold', color='red')
    plt.tight_layout()
    plt.show()
```




Boxplot of ttl_day_mins
0
50
100
150
200
250
300
350
ttl_day_mins
Boxplot of ttl_day_calls
0
25
50
75
100
125
150
ttl_day_calls
Boxplot of ttl_day_charge
0
10
20
30
40
50
60
ttl_day_charge
Boxplot of ttl_eve_mins
0
50
100
150
200
250
300
350
ttl_eve_mins


Boxplot of ttl_eve_calls
0
25
50
75
100
125
150
175
ttl_eve_calls
Boxplot of ttl_eve_charge
0
5
10
15
20
25
30
ttl_eve_charge
Boxplot of ttl_ngt_mins
50
100
150
200
250
300
350
400
ttl_ngt_mins
Boxplot of ttl_ngt_calls
40
60
80
100
120
140
160
180
ttl_ngt_calls


Boxplot of ttl_ngt_charge
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ttl_ngt_charge
Boxplot of ttl_intl_mins
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
ttl_intl_mins
Boxplot of ttl_intl_calls
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
ttl_intl_calls
Boxplot of ttl_intl_charge
0
1
2
3
4
5
ttl_intl_charge



Decision to Retain Outliers

Outliers were detected in several numeric variables, but I have retained them because they reflect meaningful customer behavior rather than errors. Since this data is collected through **Network Management Systems (NMS)** using automated **counters**, it is **reliable** and **free** from manual entry **errors**. High call minutes, charges, or frequent service calls can indicate genuine patterns such as heavy users or dissatisfied customers, which are critical for understanding churn. Moreover, extreme values are expected in telecom usage data and often represent real-world customer segments, such as business users or international callers, making them important to preserve in the analysis.

3.2: Feature Engineering

Engineered features will help us improve our model metrics as it defines relationships which would be missed by the models. Additional, these columns will simply the exploratory data visualization.

3.2.1: Creating Additional features

In [19]:
```
#creating a copy of original dataframe
df1 = df.copy()
df1.head()
```

Out[19]:

	state	acc_length	area_code	intl_plan	vmail_plan	num_vm_msgs	ttl_day_mins	ttl_day_calls	ttl_day_charge	ttl_eve_mins	ttl_eve
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	

In [20]:
```
#checking shape before
df1.shape
```

Out[20]: (3333, 20)

In [21]:
```
#Combining related features
df1['total_calls'] = (df1['ttl_day_calls'] + df1['ttl_eve_calls'] + df1['ttl_ngt_calls'] + df1['ttl_intl_cal
df1['total_mins'] = (df1['ttl_day_mins'] + df1['ttl_eve_mins'] + df1['ttl_ngt_mins'] + df1['ttl_intl_mins'])
df1['total_charge'] = (df1['ttl_day_charge'] + df1['ttl_eve_charge'] + df1['ttl_ngt_charge'] + df1['ttl_intl
df1.shape
```

Out[21]: (3333, 23)

3.2.3: Saving the cleaned dataset with additional features

In [22]:
```
df1.to_csv("cleaned_data/data_final.csv", index=False)
```

4: Exploratory Data Analysis

In [23]:

```
#importing the extended dataset
data = pd.read_csv('cleaned_data/data_final.csv',dtype={"area_code": str})
data.head()
```

Out[23]:

	state	acc_length	area_code	intl_plan	vmail_plan	num_vm_msgs	ttl_day_mins	ttl_day_calls	ttl_day_charge	ttl_eve_mins	...	ttl_
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	...	
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	...	
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	...	
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	...	
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	...	

5 rows × 23 columns

In [24]:

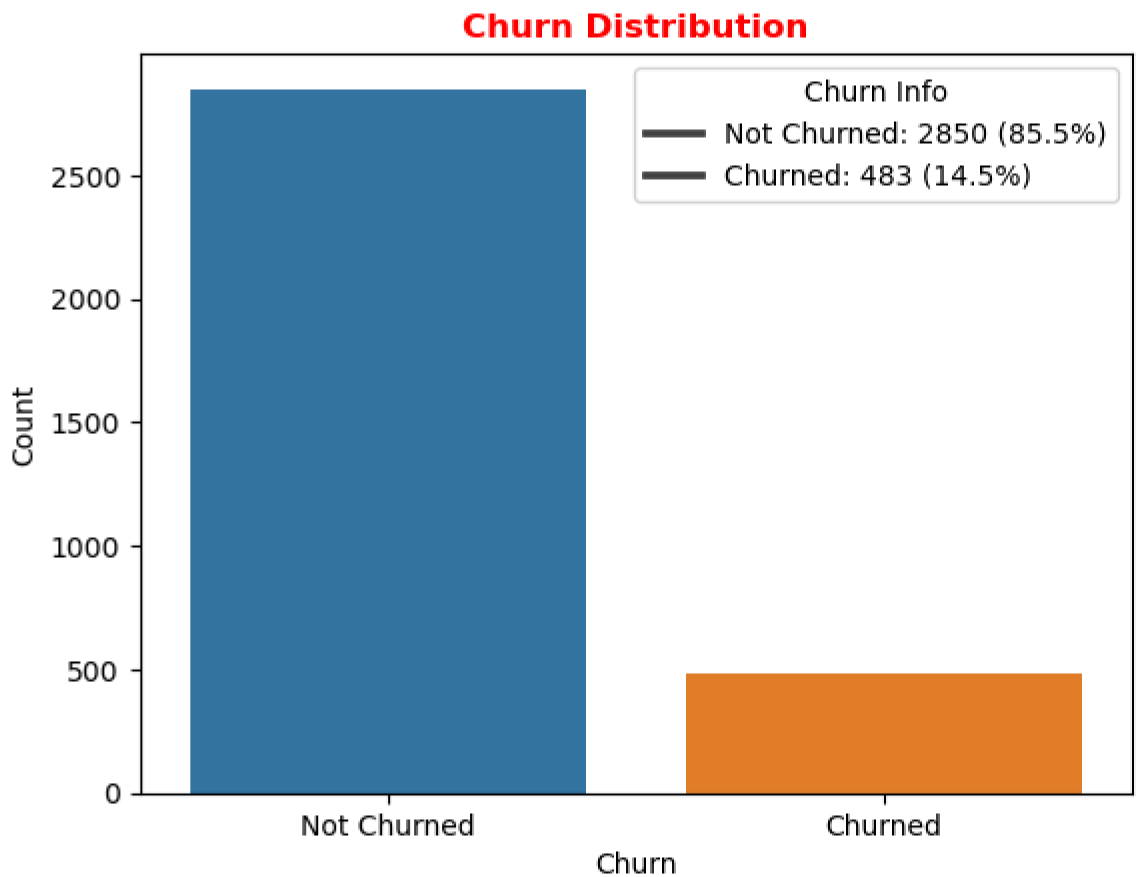
```
#checking shape
data.shape
```

Out[24]: (3333, 23)

4.1: Univariate Analysis

4.1.1: Churn distribution

In [25]:
```
churn_cnts = data['churn'].value_counts()
labels = data['churn'].value_counts(normalize=True)*100
sns.barplot(x=churn_cnts.index, y=churn_cnts.values)
plt.title("Churn Distribution", fontweight='bold',color='red')
legend_labels = [
    f"{idx}: {count} ({perc:.1f}%)"
    for idx, count, perc in zip(churn_cnts.index, churn_cnts.values, labels)
]
plt.legend(title="Churn Info", labels=legend_labels)
plt.xlabel("Churn")
plt.ylabel("Count")
plt.show()
```

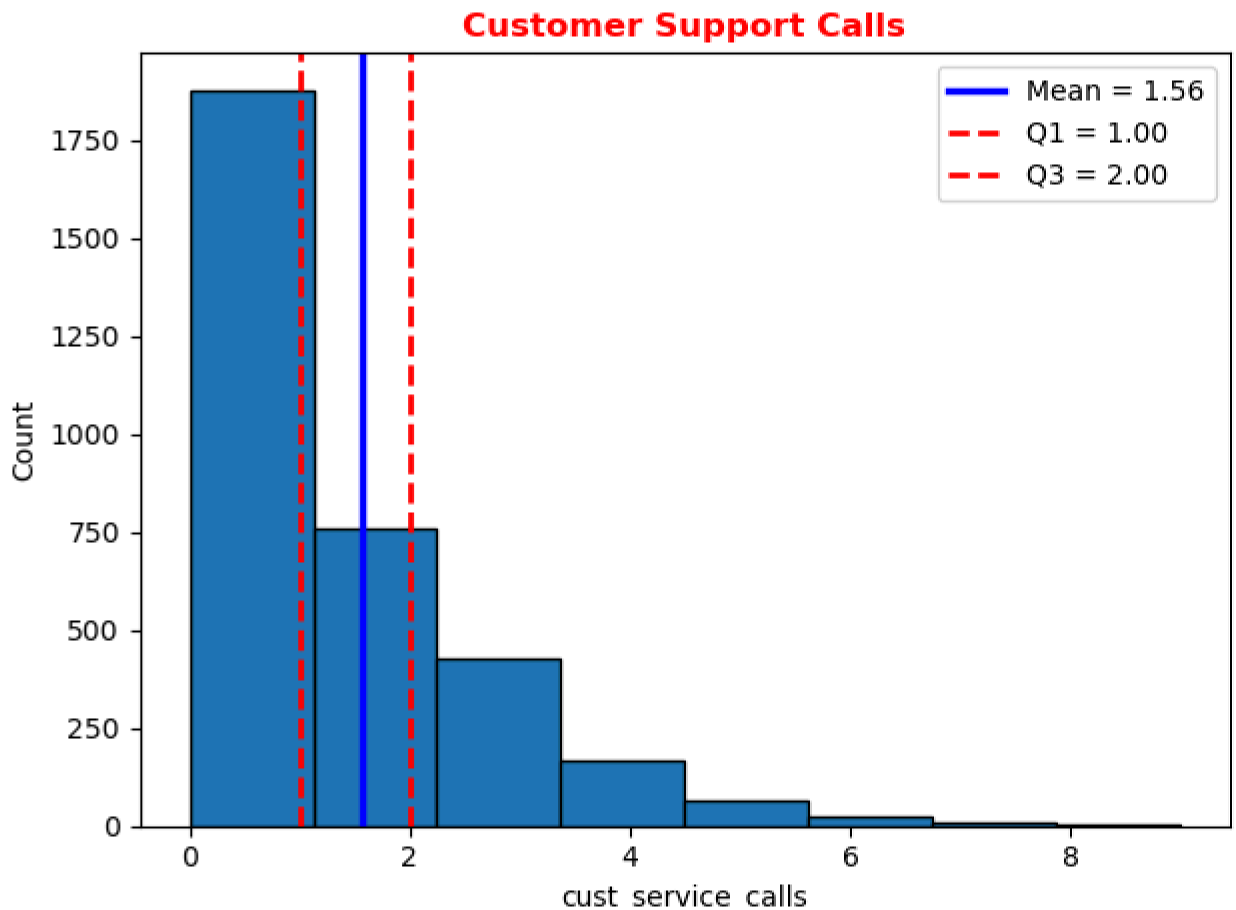


From the barplot, we can see there is class imbalance in our target variable as `85%` of the dataset did `not churn` while `14.5%` customers did `churn`.

4.1.2: How often customers call Support

In [26]:

```
# calculating statistics
mean_calls = data['cust_service_calls'].mean()
q1 = data['cust_service_calls'].quantile(0.25)
q3 = data['cust_service_calls'].quantile(0.75)
iqr = q3 - q1

#plotting the graph
plt.hist(data['cust_service_calls'], bins=8, edgecolor='black')
plt.title("Customer Support Calls", fontweight='bold', color='red')
plt.xlabel("cust_service_calls")
plt.ylabel("Count")

#adding mean and iqr lines
plt.axvline(mean_calls, color='blue', linestyle='-', linewidth=2.5, label=f"Mean = {mean_calls:.2f}")
plt.axvline(q1, color='red', linestyle='--', linewidth=2, label=f"Q1 = {q1:.2f}")
plt.axvline(q3, color='red', linestyle='--', linewidth=2, label=f"Q3 = {q3:.2f}")

#formating
plt.legend()
plt.tight_layout()
plt.show();
```



Majority of customers in the dataset `did not place` customer support calls since `more` than `two-thirds` of the dataset have `zero` calls. The customers who made calls average atleast `one phone call` with the highest being `8` calls. The distribution is `skewed` to the left showing that high number of support calls are `rare` .

4.1.3: What is the proportion of customers on International Plan

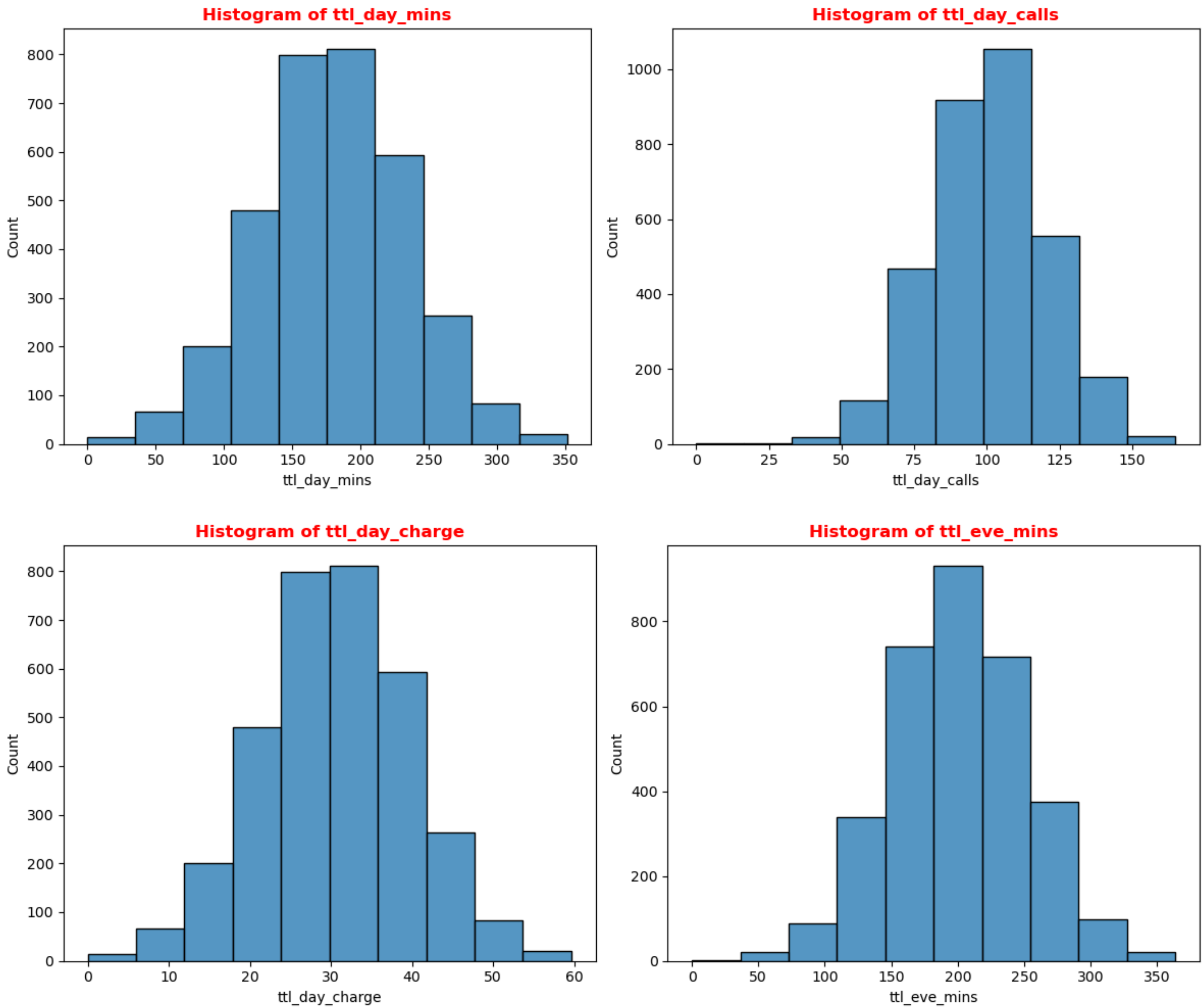
In [27]:
```
plt.figure(figsize=(7,5))
sns.countplot(x='intl_plan', data=data)
plt.title("Number of Customers with International Plan", fontweight='bold', color='red')
plt.xlabel("International Plan")
plt.ylabel("Count")
plt.tight_layout()
plt.show();
```

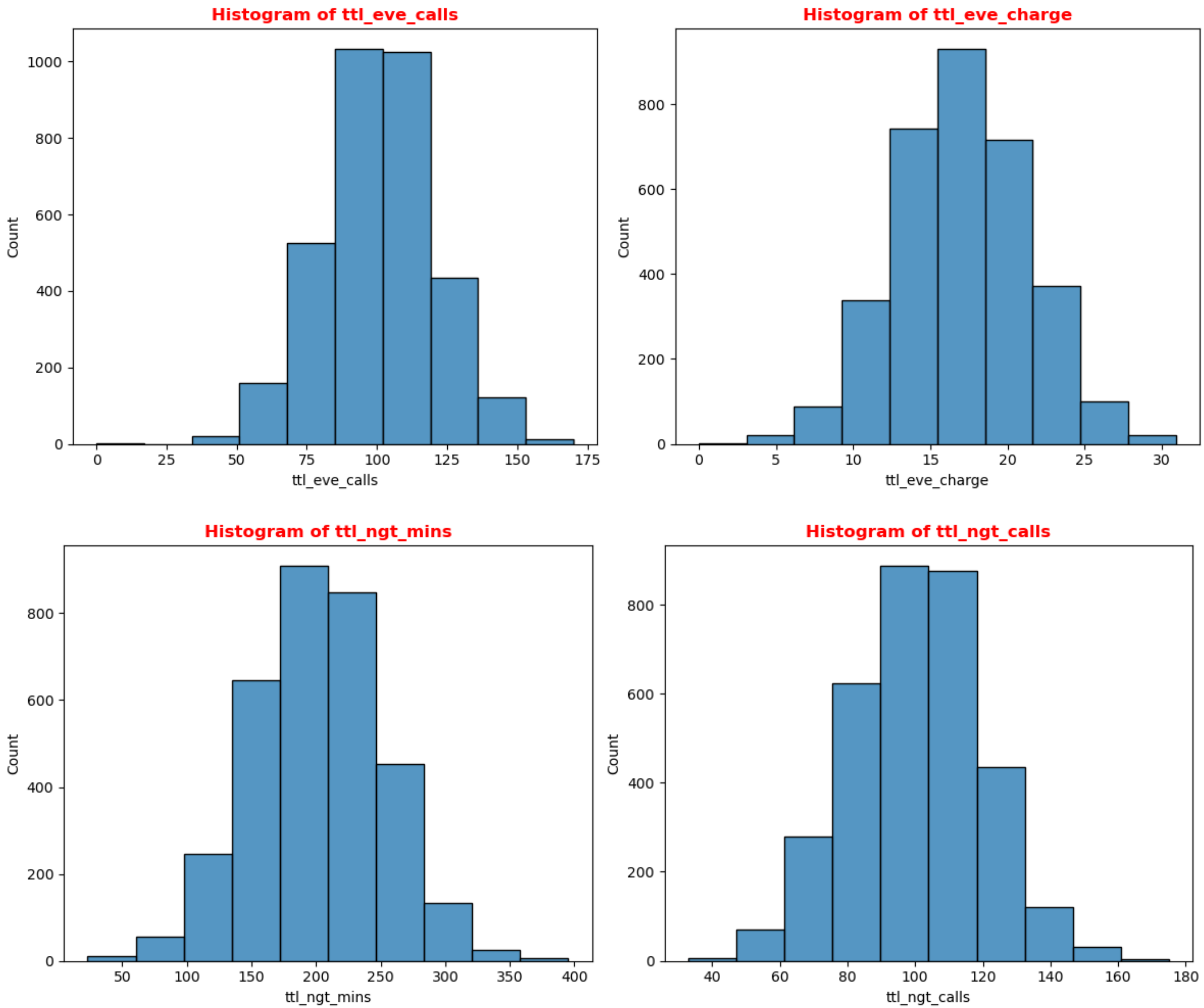
From our dataset,only about `10%` of customers have subscribed to an `international plan package` while the majority have no international plan

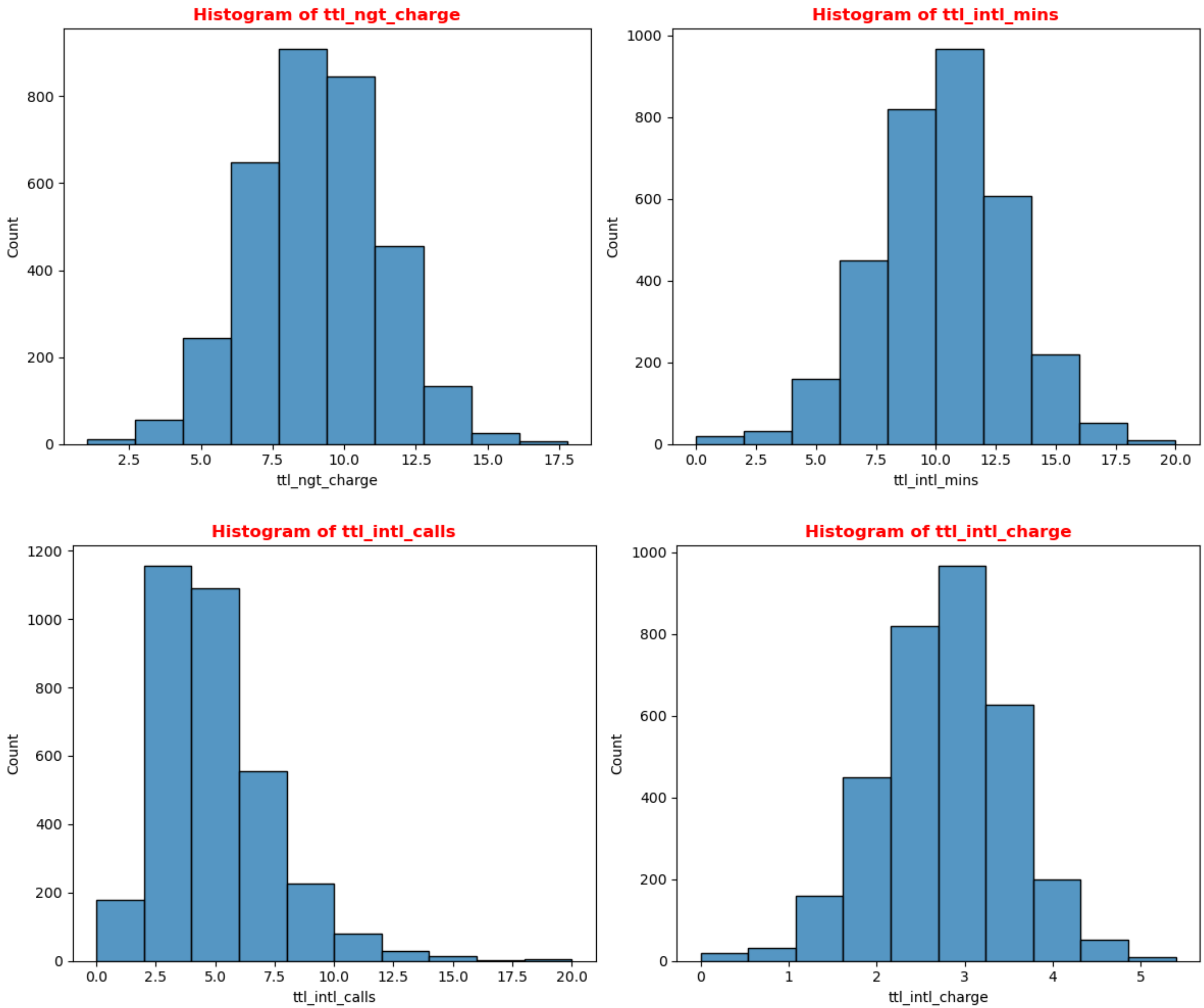
4.1.4: What is the distribution of Numerical features

In [28]:

```
numeric_cols = ['ttl_day_mins',
                'ttl_day_calls',
                'ttl_day_charge',
                'ttl_eve_mins',
                'ttl_eve_calls',
                'ttl_eve_charge',
                'ttl_ngt_mins',
                'ttl_ngt_calls',
                'ttl_ngt_charge',
                'ttl_intl_mins',
                'ttl_intl_calls',
                'ttl_intl_charge']

n = len(numeric_cols)
for i in range(0, n, 2):
    cols_pair = numeric_cols[i:i+2]
    fig, axes = plt.subplots(1, len(cols_pair), figsize=(12, 5))
    axes = np.atleast_1d(axes)

    for j, col in enumerate(cols_pair):
        sns.histplot(data[col], bins=10, kde=False, edgecolor='black', ax=axes[j])
        axes[j].set_title(f'Histogram of {col}', fontweight='bold', color='red')
        axes[j].set_xlabel(col)
        axes[j].set_ylabel("Count")

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


Histogram of ttl_day_mins
Count
ttl_day_mins
0 50 100 150 200 250 300 350
0 100 200 300 400 500 600 700 800
Histogram of ttl_day_calls
Count
ttl_day_calls
0 25 50 75 100 125 150
0 200 400 600 800 1000
Histogram of ttl_day_charge
Count
ttl_day_charge
0 10 20 30 40 50 60
0 100 200 300 400 500 600 700 800
Histogram of ttl_eve_mins
Count
ttl_eve_mins
0 50 100 150 200 250 300 350
0 200 400 600 800


Histogram of ttl_eve_calls
Count
ttl_eve_calls
0 25 50 75 100 125 150 175
0 200 400 600 800 1000
Histogram of ttl_eve_charge
Count
ttl_eve_charge
0 5 10 15 20 25 30
0 200 400 600 800
Histogram of ttl_ngt_mins
Count
ttl_ngt_mins
50 100 150 200 250 300 350 400
0 200 400 600 800
Histogram of ttl_ngt_calls
Count
ttl_ngt_calls
40 60 80 100 120 140 160 180
0 200 400 600 800


Histogram of ttl_ngt_charge
Count
0
200
400
600
800
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ttl_ngt_charge
Histogram of ttl_intl_mins
Count
0
200
400
600
800
1000
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
ttl_intl_mins
Histogram of ttl_intl_calls
Count
0
200
400
600
800
1000
1200
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
ttl_intl_calls
Histogram of ttl_intl_charge
Count
0
200
400
600
800
1000
0
1
2
3
4
5
ttl_intl_charge

Distribution of Numeric Variables

--- The histograms show the distribution of telecom usage variables such as `total minutes` , `number of calls` , and `charges` across different time periods (`day` , `evening` , `night` , and `international`).

--- Most variables appear roughly **normally distributed**, with `call totals` being **narrower** while `minutes` and `charges` show **wider** variance.

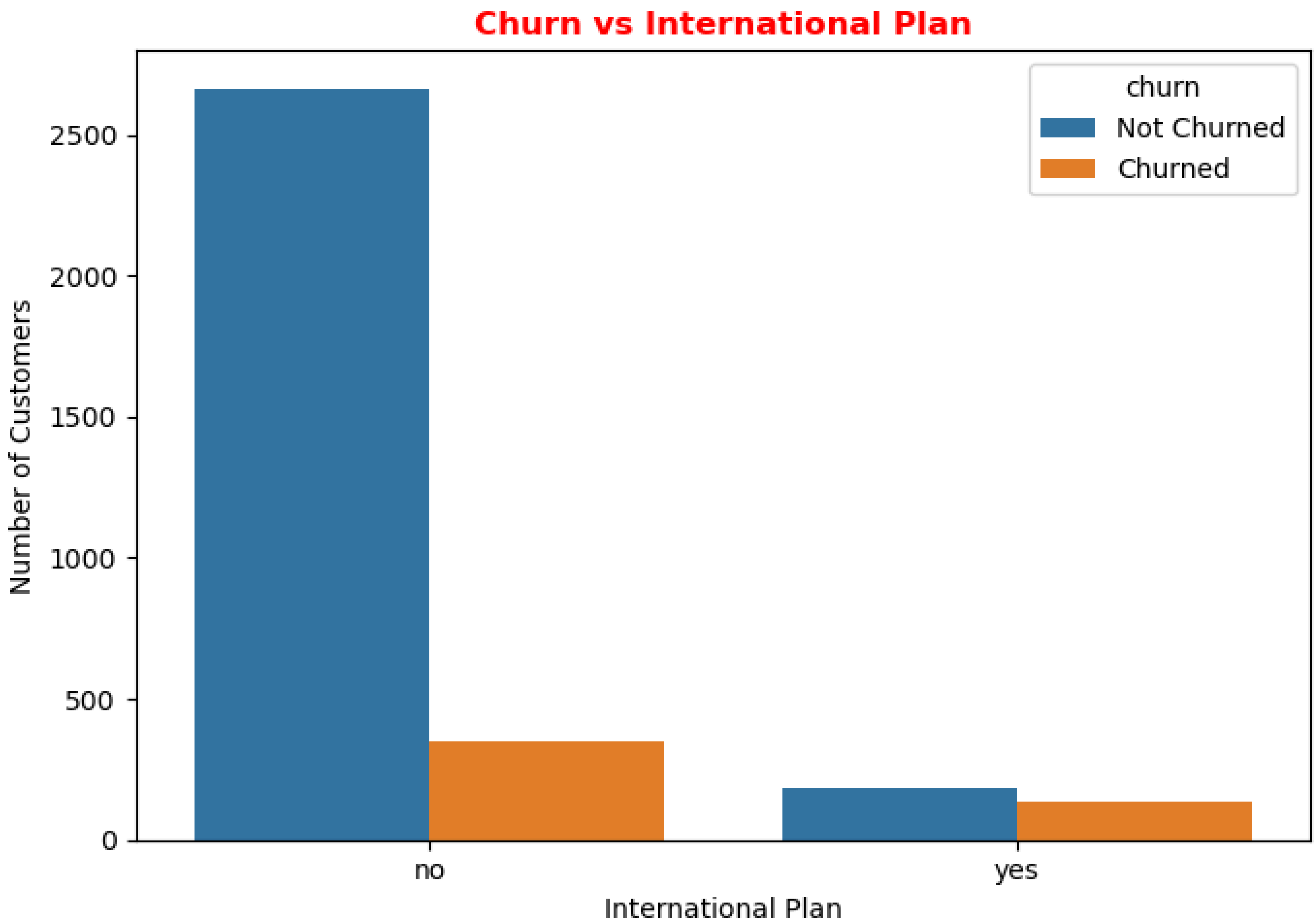
--- Worth noting is that `charges` follow the **same distribution** as their corresponding `minutes` since they are directly derived from call durations.

--- Overall, the distributions look well behaved, with most values falling within reasonable ranges for telecom usage.

4.2: Bivariate analysis

4.2.1: Do customers with international plan churn more?

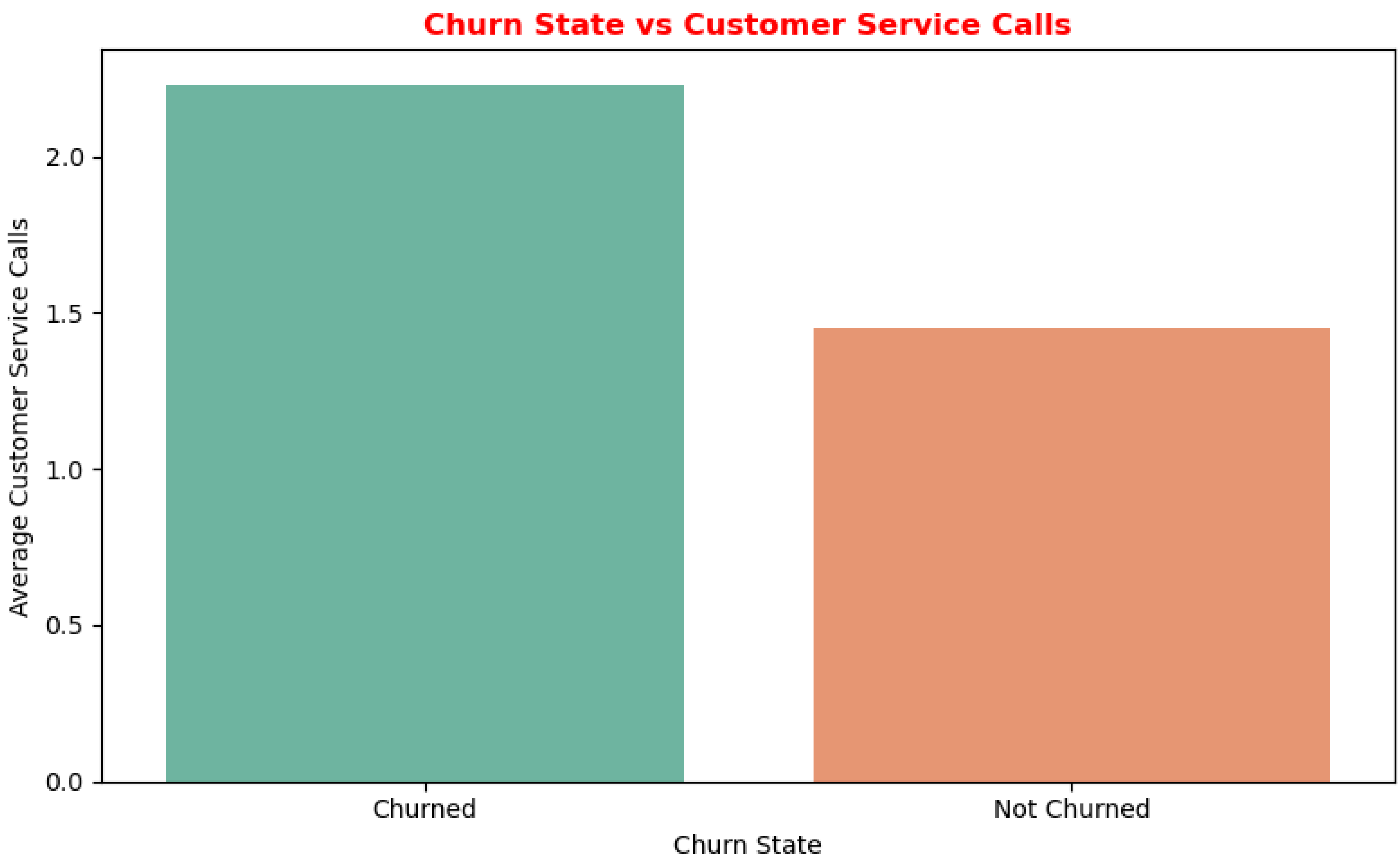
In [29]:
```
plt.figure(figsize=(7,5))
sns.countplot(data=data, x="intl_plan", hue="churn")
plt.title("Churn vs International Plan", fontweight='bold', color='red')
plt.xlabel("International Plan")
plt.ylabel("Number of Customers")
plt.tight_layout()
plt.show();
```



Most customers do **not** have an international plan. Among the customers without an international plan, churn rates are relatively low. On the customers **with** an international plan, churn is noticeably higher relative to their group size. This suggests that having an international plan is associated with a higher likelihood of churn since this customers require a higher quality of service.

4.2.2: What is the distribution of Numerical features

In [30]:
```
churn_rate = data.groupby("churn")["cust_service_calls"].mean().reset_index()
plt.figure(figsize=(8,5))
sns.barplot(data=churn_rate, x="churn", y="cust_service_calls", palette="Set2")
plt.title("Churn State vs Customer Service Calls", fontweight='bold', color='red')
plt.xlabel("Churn State")
plt.ylabel("Average Customer Service Calls")
plt.tight_layout()
plt.show();
```

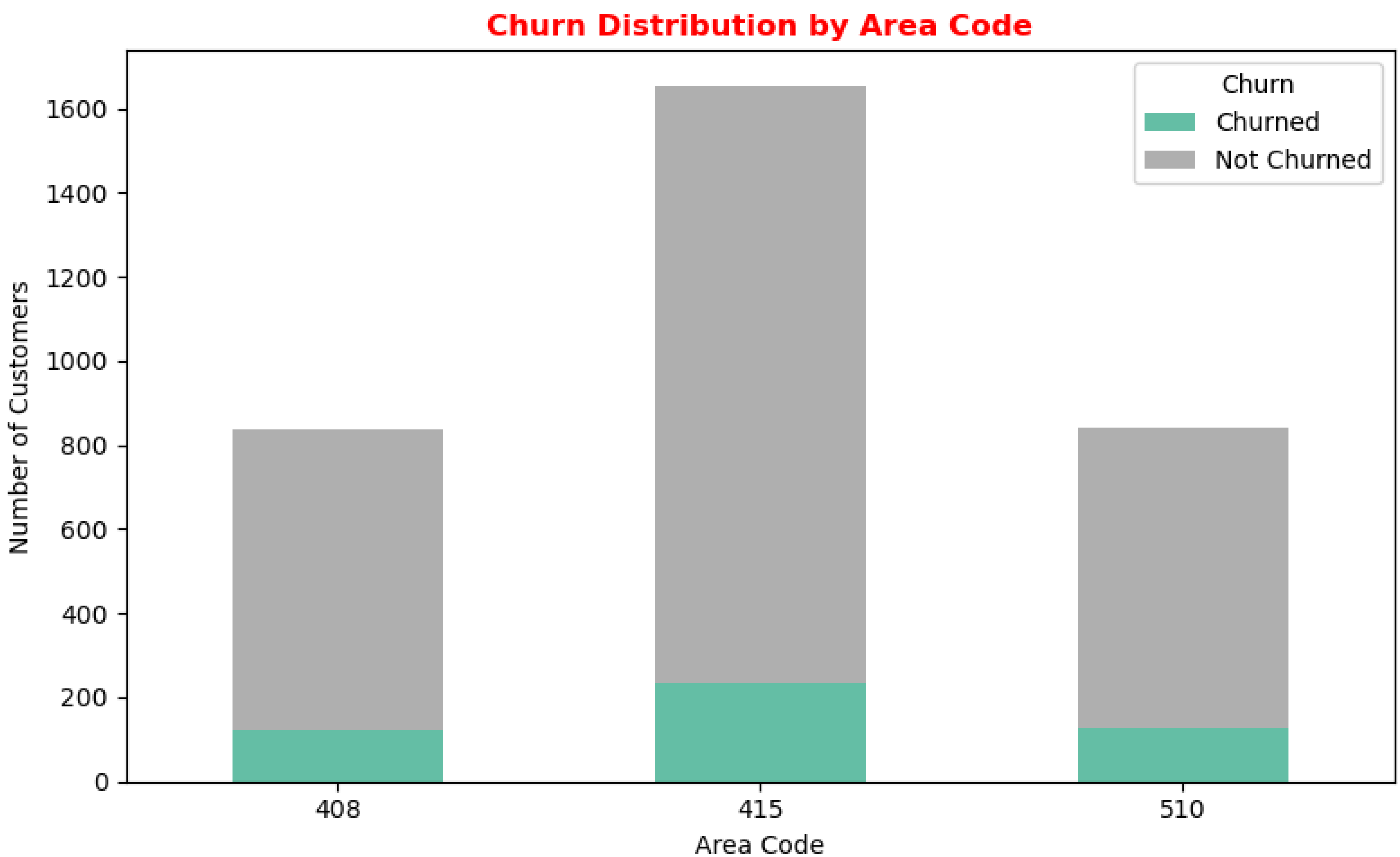


Customers who churned made more **customer service calls** on average than those who did not, indicating a potential correlation between frequent service interactions and underlying dissatisfaction or frustration. This pattern suggests that repeated contact with customer support may be a signal of unresolved issues or negative experiences, which could ultimately drive customers to leave.

4.2.3: What is the Churn Distribution by Area Code?

In [31]:
```
churn_area = pd.crosstab(data['area_code'],data['churn'])
plt.figure(figsize=(8,5))
churn_area.plot(kind="bar", stacked=True, colormap='Set2', figsize=(8,5))
plt.title("Churn Distribution by Area Code", fontweight='bold', color='red')
plt.xlabel("Area Code")
plt.ylabel("Number of Customers")
plt.xticks(rotation=0)
plt.legend(title="Churn")
plt.tight_layout()
plt.show();
```

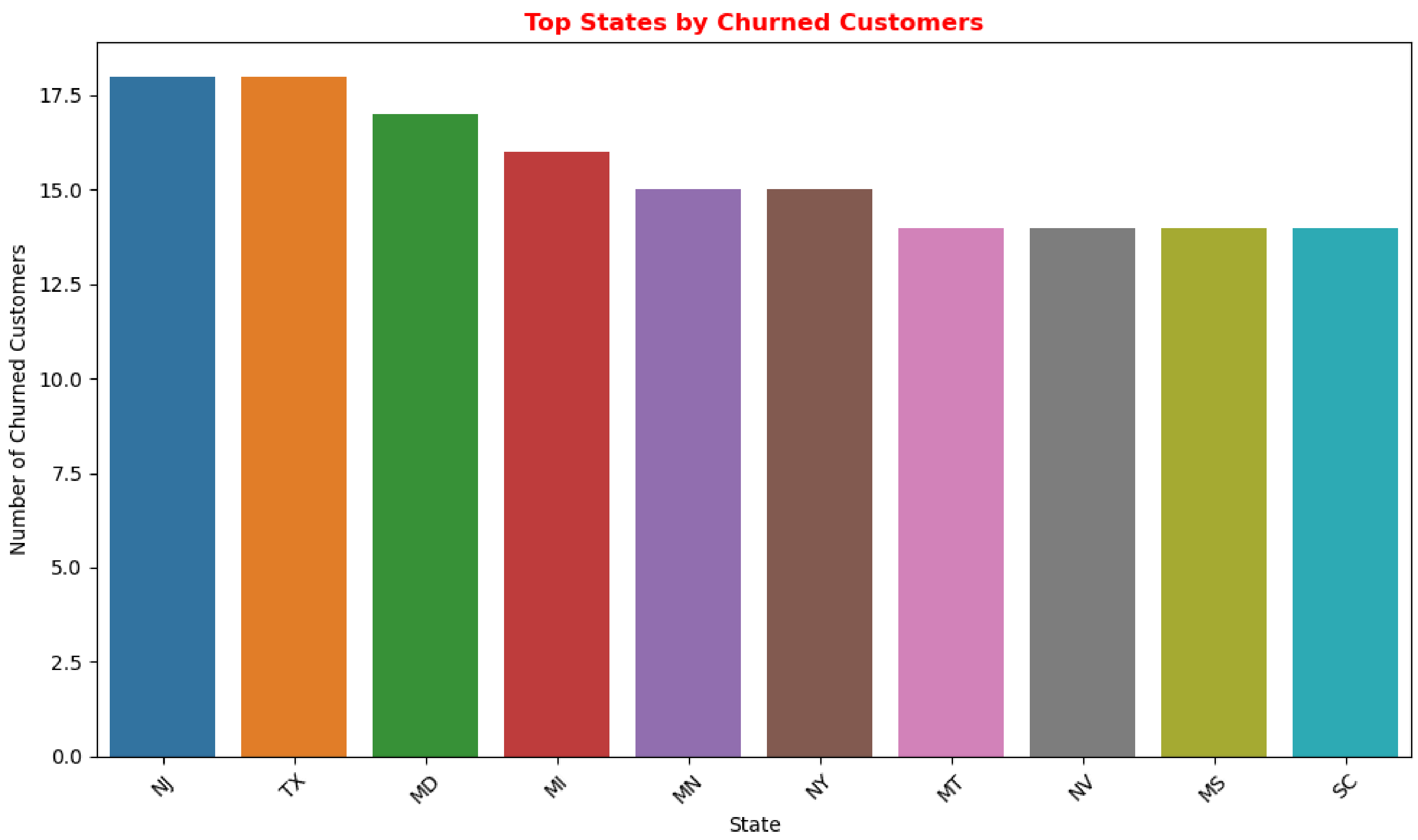
```
<Figure size 800x500 with 0 Axes>
```



Area code **415** has the highest total customer count and a notably large churn segment, which may point to underlying issues in service quality or customer satisfaction. In contrast, area codes **408** and **510** have similar customer volumes but exhibit lower churn rates, suggesting that customers in these regions may be receiving **better service** or are more effectively retained. The elevated churn in **415** could be driven by factors such as `longer wait times, unresolved service issues, inconsistent support experiences, or regional infrastructure and staffing challenges`. Meanwhile, the lower churn observed in 408 and 510 may reflect more `responsive customer service, efficient issue resolution, and stronger engagement efforts` that contribute to improved customer retention.

4.2.4: What is the Churn Distribution by State?

In [32]:
```
state_churn = data[data["churn"] == "Churned"].groupby("state")["churn"].count().reset_index()
state_churn = state_churn.rename(columns={"churn": "churn_count"})
top_states = state_churn.sort_values("churn_count", ascending=False).head(10)

plt.figure(figsize=(10,6))
sns.barplot(data=top_states, x="state", y="churn_count")
plt.title("Top States by Churned Customers", fontweight='bold', color='red')
plt.xlabel("State")
plt.ylabel("Number of Churned Customers")
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

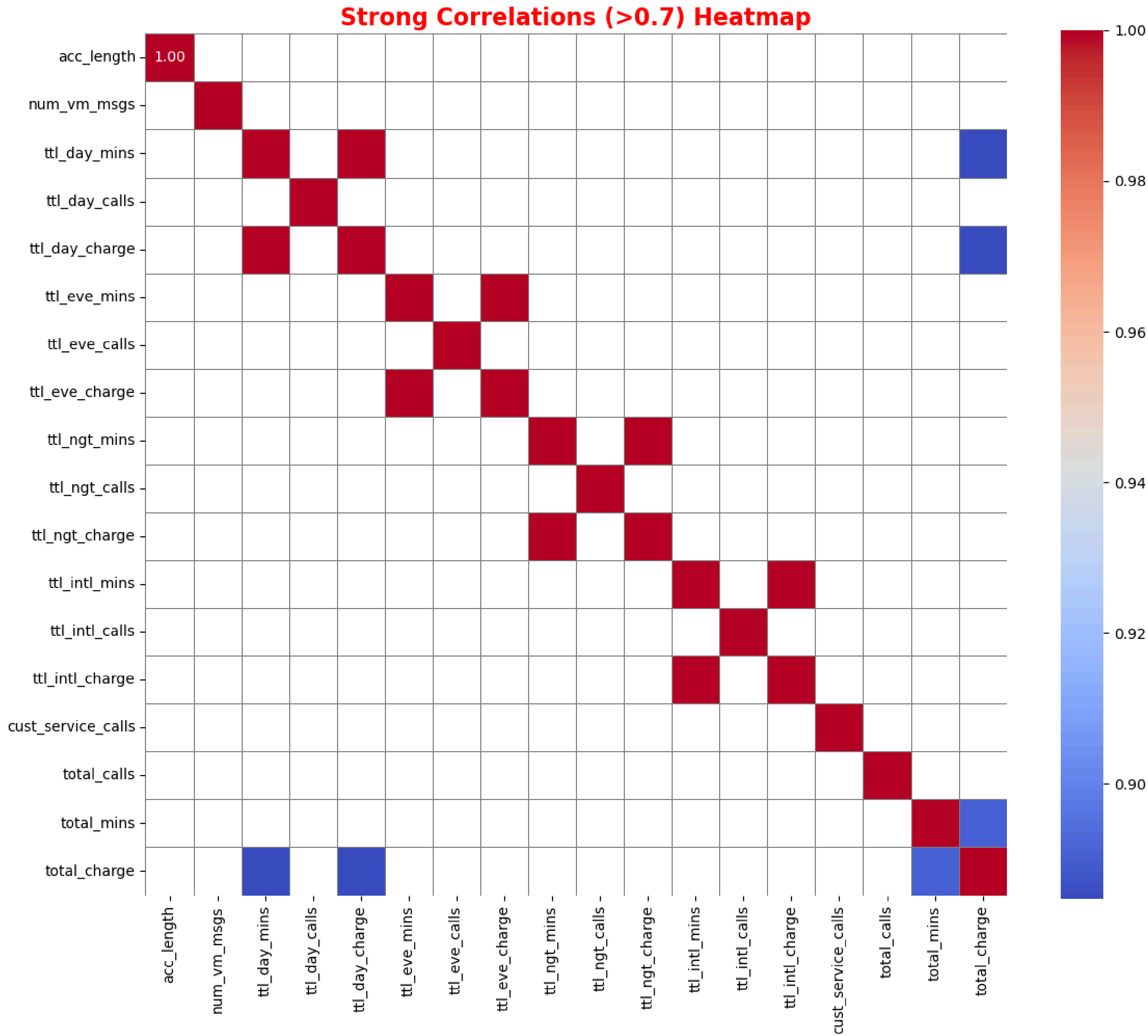


New Jersey has the highest number of churned customers, followed closely by **Texas**. States like **Maryland** and **Minnesota** also have elevated churn, while Montana, New York, Mississippi, and South Carolina have the lowest among the top ten. This highlights key regions where customer retention efforts may need to be strengthened.

4.2.5: What is the association of numerical features?

In [33]:

```
import numpy as np
numeric_data = data.select_dtypes(include=['int64', 'float64'])
corr = numeric_data.corr()
mask = corr < 0.7
plt.figure(figsize=(12,10))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm", square=True,linewidths=0.5, linecolor='gray', mask
plt.title("Strong Correlations (>0.7) Heatmap", fontweight='bold', color='red', fontsize=16)
plt.tight_layout()
plt.show();
```


Strong Correlations (>0.7) Heatmap
1.00
acc_length
num_vm_msgs
ttl_day_mins
ttl_day_calls
ttl_day_charge
ttl_eve_mins
ttl_eve_calls
ttl_eve_charge
ttl_ngt_mins
ttl_ngt_calls
ttl_ngt_charge
ttl_intl_mins
ttl_intl_calls
ttl_intl_charge
cust_service_calls
total_calls
total_mins
total_charge
1.00
0.98
0.96
0.94
0.92
0.90

The color gradient in the heatmap ranges from blue (negative correlation) to red strong positive correlation, with the diagonal line showing perfect self-correlation (1.00) for each variable.

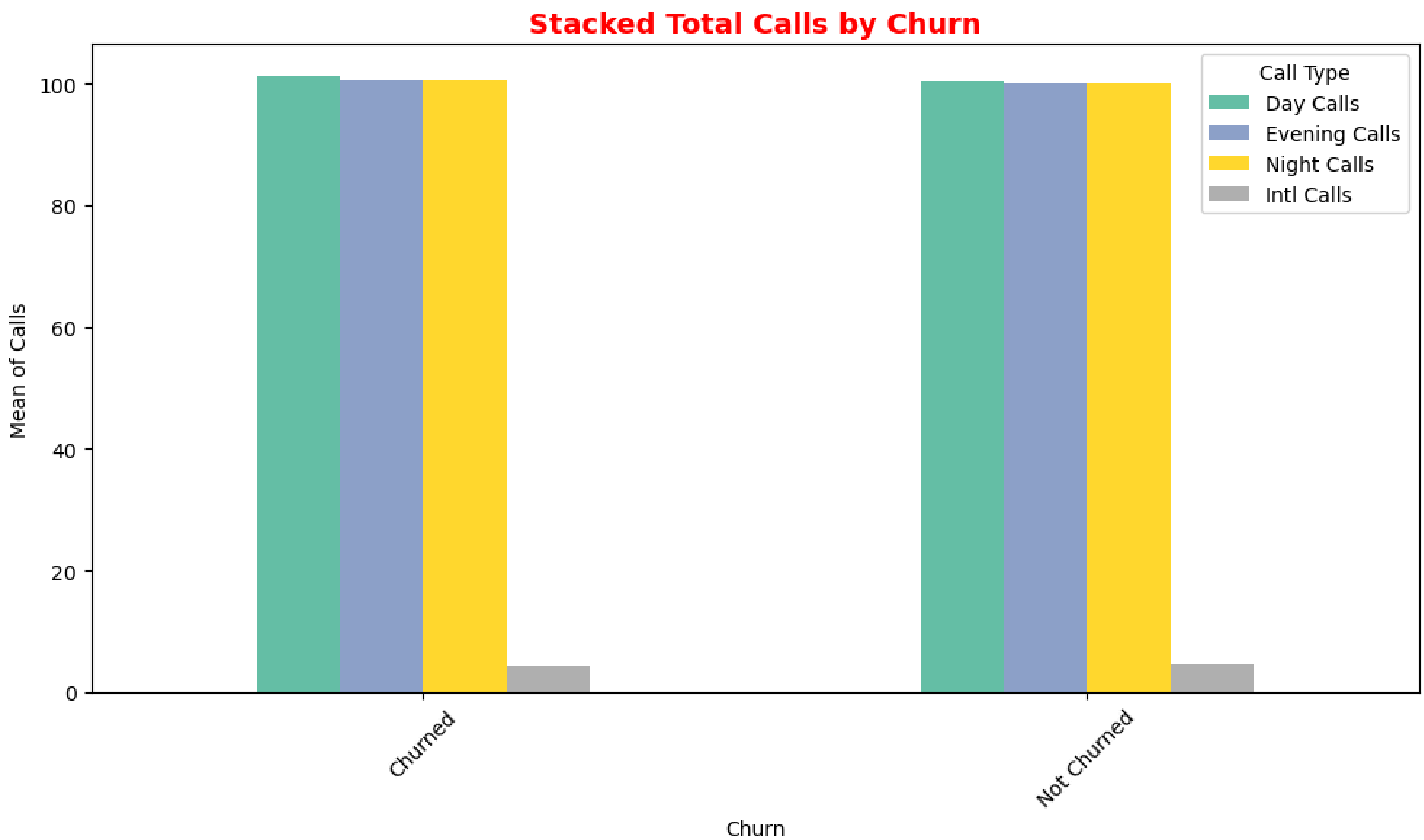
- **Strong correlations** are evident between:
  - `ttl_day_mins` and `ttl_day_charge`
  - `ttl_eve_mins` and `ttl_eve_charge`
  - `ttl_ngt_mins` and `ttl_ngt_charge`
  - `ttl_intl_mins` and `ttl_intl_charge`
  - `total_mins` and `total_charge`

The heatmap validates the expected patterns such as the link between minutes and charges(elements of **multicollinearity**) but also indicates that most of the other features in the dataset show little linear dependency on each other. As a result during modelling, we need to drop the feature engineered column and other columns with a perfect correlation.

4.3: Multivariate analysis

4.3.1: Churn vs Total Calls (Day/Eve/Night/Intl)

In [34]:
```
call_columns = ['ttl_day_calls', 'ttl_eve_calls', 'ttl_ngt_calls', 'ttl_intl_calls']
calls_churn = data.groupby('churn')[call_columns].mean()
calls_churn.plot(kind='bar', stacked=False, figsize=(10,6), colormap='Set2')
plt.title("Stacked Total Calls by Churn", fontsize=14, fontweight='bold', color='red')
plt.xlabel("Churn")
plt.ylabel("Mean of Calls")
plt.legend(title="Call Type", labels=['Day Calls', 'Evening Calls', 'Night Calls', 'Intl Calls'])
plt.xticks(rotation=45)
plt.tight_layout()
plt.show();
```

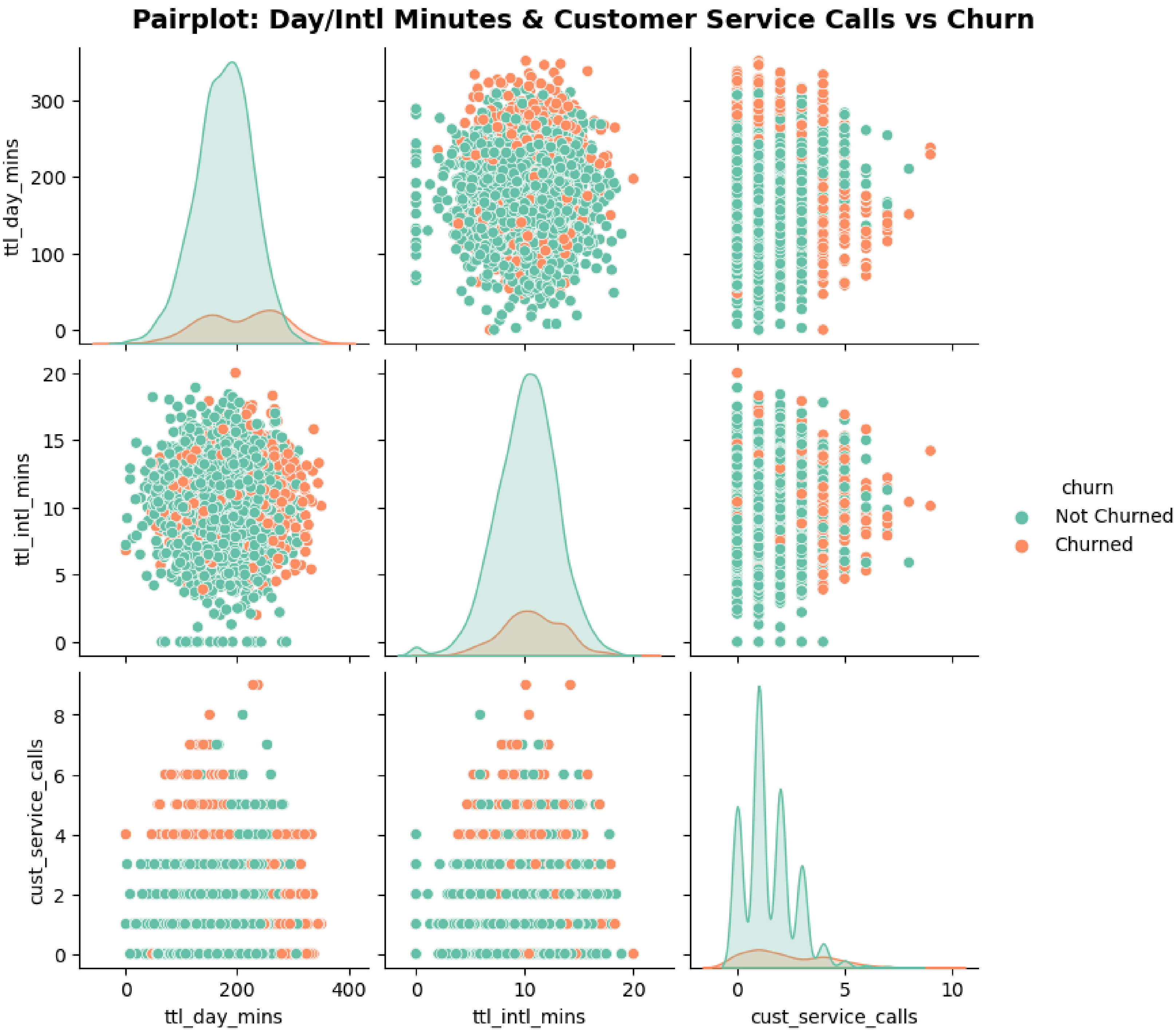


The chart compares the average number of calls made by **churned** and **non-churned customers**, revealing that both groups exhibit similar usage patterns across **Day**, **Evening**, and **Night calls**. **International calls** make up the smallest portion of total call volume for both segments. Overall, the consistency in call behavior between churned and retained customers suggests that churn may not be directly influenced by call frequency alone.

4.3.2: Day/Intl Minutes & Customer Service Calls vs Churn

In [35]:

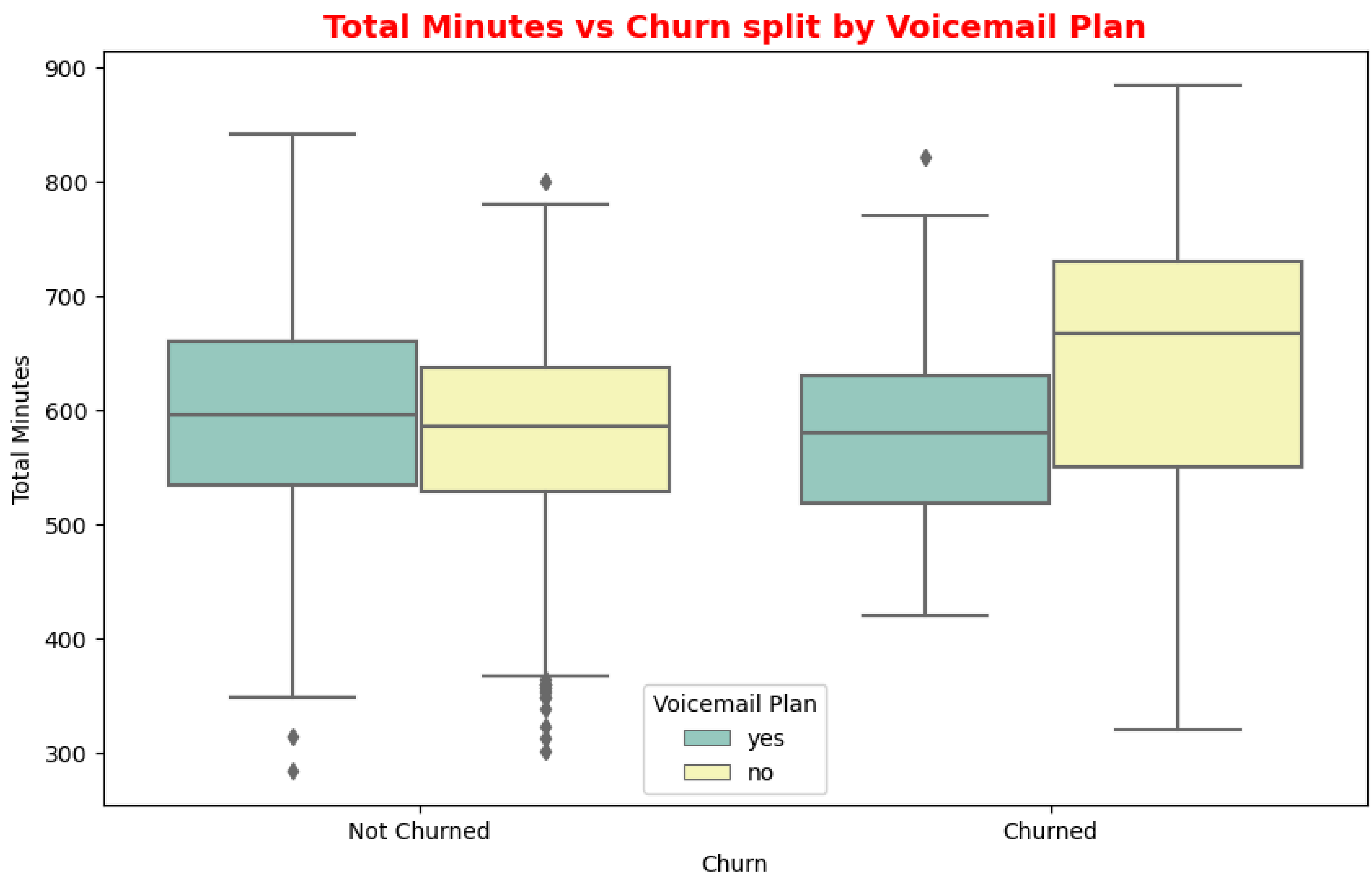
```
selected_features = ['ttl_day_mins', 'ttl_intl_mins', 'cust_service_calls', 'churn']
sns.pairplot(data[selected_features], hue='churn', palette='Set2', diag_kind='kde')
plt.suptitle("Pairplot: Day/Intl Minutes & Customer Service Calls vs Churn", y=1.02, fontsize=14, fontweight
plt.show()
```


Pairplot: Day/Intl Minutes & Customer Service Calls vs Churn
ttl_day_mins
ttl_intl_mins
cust_service_calls
churn
Not Churned
Churned
0
100
200
300
400
5
10
15
20
2
4
6
8

This pairplot visualizes the relationships between three key variables— `ttl_day_mins` , `ttl_intl_mins` , and `cust_service_calls` in relation to **customer churn**. There is a visible spread in `cust_service_calls` among churned customers, suggesting a potential link between frequent service interactions and churn. `ttl_day_mins` and `ttl_intl_mins` show distinct usage patterns, but less separation between churned and non-churned groups. KDE curves help highlight differences in distribution, especially for `cust_service_calls` .

4.3.3: Total Minutes vs Churn split by Voicemail Plan

In [36]:
```
plt.figure(figsize=(10,6))
sns.boxplot(data=data, x='churn', y='total_mins', hue='vmail_plan', palette="Set3")
plt.title("Total Minutes vs Churn split by Voicemail Plan", fontsize=14, fontweight='bold', color='red')
plt.xlabel("Churn")
plt.ylabel("Total Minutes")
plt.legend(title="Voicemail Plan")
plt.show()
```

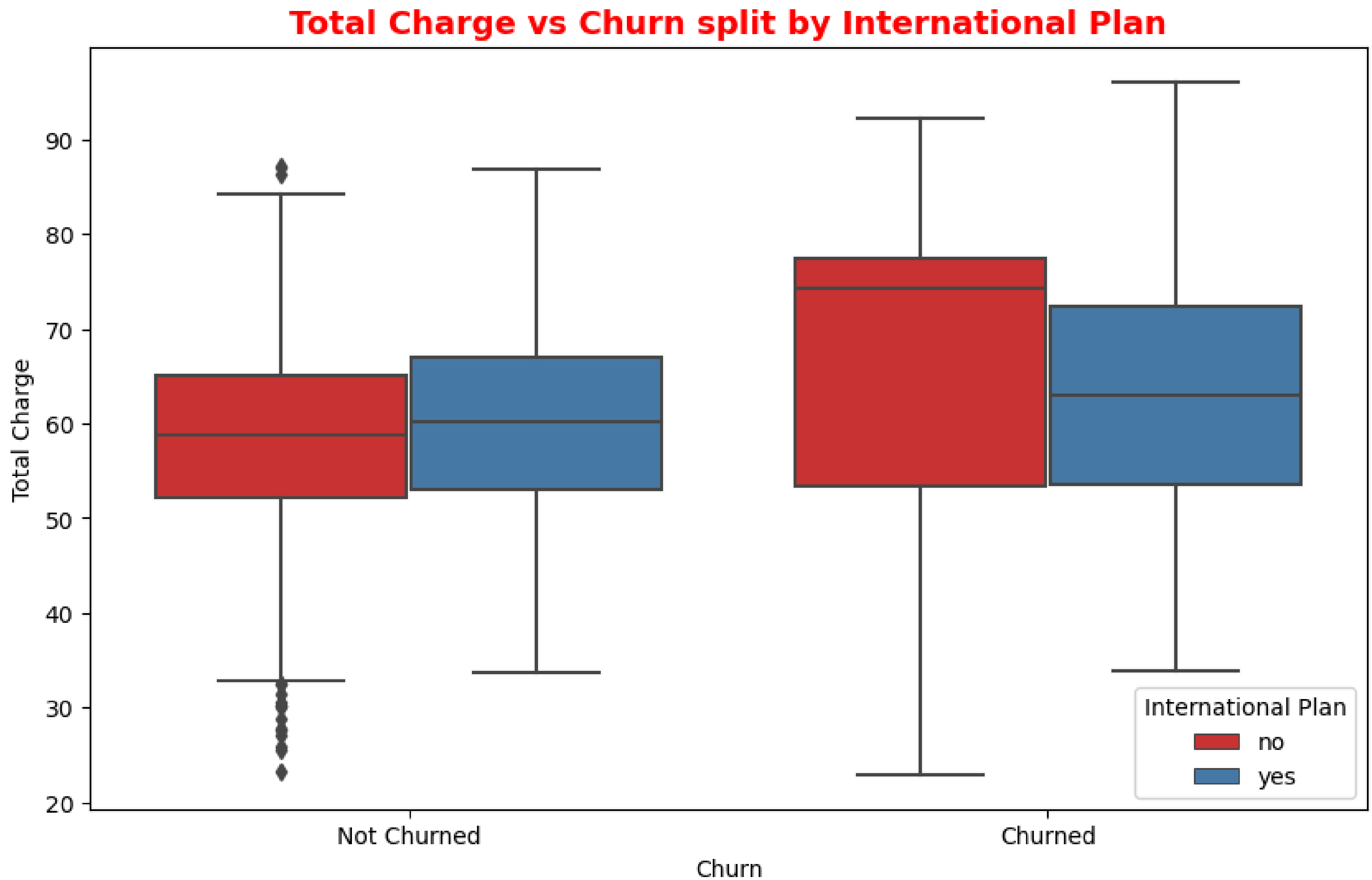


The box plot shows how **total minutes used** by customers vary by **churn status** and **voicemail plan**. **Churned customers without a voicemail plan** exhibit a **wider spread of usage**, **more outliers**, and a **slightly higher median**, suggesting that **heavy usage without voicemail** may be linked to **dissatisfaction**. In contrast, **non-churned customers with a voicemail plan** show **more consistent usage patterns** and **fewer extreme values**, indicating that **voicemail access** may support a **stable and satisfactory experience**. Overall, the differences suggest that **voicemail plans** could play a **meaningful role in retaining customers**, particularly those with **high usage**.

4.3.4: Total Charge vs Churn split by International Plan

In [37]:

```
plt.figure(figsize=(10,6))
sns.boxplot(data=data, x='churn', y='total_charge', hue='intl_plan', palette="Set1")
plt.title("Total Charge vs Churn split by International Plan", fontsize=14, fontweight='bold', color='red')
plt.xlabel("Churn")
plt.ylabel("Total Charge")
plt.legend(title="International Plan")
plt.show()
```



This box plot compares the distribution of **total charges** between **churned** and **non-churned customers**, divided by **international plan status**. **Churned customers** generally have **higher total charges**, and among them, those **without an international plan** show a **wider range and higher median**. In contrast, **non-churned customers** tend to have **lower and more consistent charges**, particularly those **with an international plan**, though a few **outliers** exist in the **non-churned group without an international plan**. These patterns suggest that **high charges** may be linked to **churn**, especially for customers lacking **international plan benefits**, and that offering or optimizing **international plans** could help retain **high-usage customers**.

5: Inferential Analysis

5.1: Chi-Square Test of Independence

-- Are customers with a voicemail plan more or less likely to churn compared to those without it?

5.1.1: Hypotheses

- **Null Hypothesis (H_0):**
  There is **no association** between having a voicemail plan and customer churn.
- **Alternative Hypothesis (H_1):**
  There **is an association** between having a voicemail plan and customer churn. The likelihood of churn differs between customers with and without a voicemail plan.

5.1.2: Test

In [38]:

```
from scipy.stats import chi2_contingency
contingency_table = pd.crosstab(data['vmail_plan'], data['churn'])

chi2, p, dof, expected = chi2_contingency(contingency_table)
print("Chi-Square Test Statistic:", chi2)
print("Degrees of Freedom:", dof)
print("P-value:", p)
```

```
Chi-Square Test Statistic: 34.13166001075673
Degrees of Freedom: 1
P-value: 5.15063965903898e-09
```

5.1.3: Inference

Since the **p-value** is much smaller than 0.05, `we reject the null hypothesis (H₀)` . This indicates a significant association between having a **voicemail plan** and **customer churn**. In other words, customers with and without voicemail plans exhibit different churn behaviors, suggesting that voicemail plan status plays a role in the likelihood of churn.

5.2: Independent Samples T-test

-- Does mean charges differ between churned and non churned customers?

5.2.1: Hypotheses

- **Null Hypothesis (H_0):**
  The mean total charges are the same for churned and non-churned customers.
- **Alternative Hypothesis (H_1):**
  The mean total charges differ between churned and non-churned customers.

5.2.2: Test

In [39]:

```
from scipy.stats import ttest_ind

churned = data[data['churn'] == 'Churned']['total_charge']
not_churned = data[data['churn'] == 'Not Churned']['total_charge']
t_stat, p_val = ttest_ind(churned, not_churned, equal_var=False)
print("T-test Statistic:", t_stat)
print("P-value:", p_val)
```

```
T-test Statistic: 10.526419982942551
P-value: 9.114972608480788e-24
```

5.2.3: Inference

Since the p-value is **much smaller than 0.05**, we reject the null hypothesis (H_0). This means there is a **statistically significant difference** in mean total charges between churned and non-churned customers. Customers who churn tend to have **higher or different total charges** compared to those who do not, indicating that total charges are strongly associated with churn behavior.

5.3: One-Way ANOVA

-- Do churn differ across area codes?

5.3.1: Hypotheses

- **Null Hypothesis (H_0):**
  The mean churn rate is the same across all area codes.
- **Alternative Hypothesis (H_1):**
  At least one area code has a different mean churn rate compared to the others.

5.3.2: Test

In [40]:
```
#mapping the churn into numeric
data['churn_num'] = data['churn'].map({'Not Churned': 0, 'Churned': 1})
from scipy.stats import f_oneway
groups = [group['churn_num'].values for name, group in data.groupby('area_code')]
f_stat, p_val = f_oneway(*groups)
print("ANOVA F-statistic:", f_stat)
print("P-value:", p_val)
```

```
ANOVA F-statistic: 0.08869516885890079
P-value: 0.9151266513306314
```

5.3.3: Inference

Since the p-value is **much greater than 0.05**, we fail to reject the null hypothesis (H_0). This indicates that there is **no statistically significant difference** in mean churn rate across the different area codes. Customers from different area codes exhibit **similar churn behavior**, suggesting that area code alone does not play a role in predicting churn in this dataset.

6: Modeling

6.1: Data Preprocessing

6.1.1: Importing data

In [41]:
```
#importing the data again
data = pd.read_csv('cleaned_data/data_final.csv',dtype={"area_code": str})
data.head()
```

Out[41]:

	state	acc_length	area_code	intl_plan	vmail_plan	num_vm_msgs	ttl_day_mins	ttl_day_calls	ttl_day_charge	ttl_eve_mins	...	ttl
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	...	
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	...	
2	NJ	137	415	no	no	0	243.4	114	41.38	121.2	...	
3	OH	84	408	yes	no	0	299.4	71	50.90	61.9	...	
4	OK	75	415	yes	no	0	166.7	113	28.34	148.3	...	

5 rows × 23 columns

6.1.2: Label Encoding the Target Variable & Binary Columns

In [42]:
```
#Returning the Target Variable into Boolean as it was changed to string during EDA for interpretability
data["churn"] = df["churn"].map({"Churned": True, "Not Churned": False})
```

In [43]:
```
#Label Encoding binary columns
label = LabelEncoder()
data['churn'] = label.fit_transform(data['churn'])
data['intl_plan'] = label.fit_transform(data['intl_plan'])
data['vmail_plan'] = label.fit_transform(data['vmail_plan'])
```

6.1.3: One hot Encoding Non_Binary Categorical columns

In [44]:
```
#defining categorical columns
cat_cols = ['state', 'area_code']
```

In [45]:
```
for col in cat_cols:
    print(data[col].unique())
```

```
['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']
['415' '408' '510']
```

In [46]:
```
#one hot encoding
ohe = OneHotEncoder(drop='first', sparse_output=False)
encoded = ohe.fit_transform(data[['state', 'area_code']])
encoded_df = pd.DataFrame(encoded, columns=ohe.get_feature_names_out(['state', 'area_code']))
encoded_df.head()
```

Out[46]:

	state_AL	state_AR	state_AZ	state_CA	state_CO	state_CT	state_DC	state_DE	state_FL	state_GA	...	state_TX	state_UT	state
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

5 rows × 52 columns

In [47]:

```
#Merging back to original df and dropping columns
data = pd.concat([data.drop(['state', 'area_code'], axis=1), encoded_df], axis=1)
data.head()
```

Out[47]:

	acc_length	intl_plan	vmail_plan	num_vm_msgs	ttl_day_mins	ttl_day_calls	ttl_day_charge	ttl_eve_mins	ttl_eve_calls	ttl_eve_cha
0	128	0	1	25	265.1	110	45.07	197.4	99	1(
1	107	0	1	26	161.6	123	27.47	195.5	103	1(
2	137	0	0	0	243.4	114	41.38	121.2	110	1(
3	84	1	0	0	299.4	71	50.90	61.9	88	!
4	75	1	0	0	166.7	113	28.34	148.3	122	1:

5 rows × 73 columns

6.1.4: Creating two dataframes to test impact of Multicollinearity

In [48]:

```
#df with engineered and multicollinearity columns
df1 = data.copy()
#df without engineered and multicollinearity columns
eng_cols = ['total_calls', 'total_mins',
       'total_charge','ttl_day_charge','ttl_eve_charge','ttl_ngt_charge','ttl_intl_charge']
df2 = data.drop(columns=eng_cols, axis=1)
```

6.2: Model_1: Multicollinearity

In [49]:

```
# separating features and target
X = df1.drop("churn",axis=1)
y = df1.churn

#splitting to train and test set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42,stratify=y)

#Scaling featues
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#fitting models
models = {'logistic regression':LogisticRegression(random_state=42,max_iter=1000),
          'Decision tree': DecisionTreeClassifier(random_state=42),
          'Random Forest':RandomForestClassifier(random_state=42,n_estimators=100),
          'Xgboost':XGBClassifier(random_state=42,use_label_encoder=False,eval_metric='logloss')
          }
model_results = []
for name, model in models.items():
    model.fit(X_train_scaled,y_train)
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:,1]

    model_results.append({'Model':name,
                    'Train_Score':model.score(X_train_scaled,y_train),
                    'Test_Score':model.score(X_test_scaled,y_test),
                    'Accuracy':accuracy_score(y_test,y_pred)*100,
                    'Recall':recall_score(y_test,y_pred)*100,
                    'Precision':precision_score(y_test,y_pred)*100,
                    'F1 score':f1_score(y_test,y_pred)*100,
                    'AUC':roc_auc_score(y_test,y_prob)*100
                })
model_results_df = pd.DataFrame(model_results).round(2)[["Model", "Train_Score", "Test_Score", "Accuracy", "
model_results_df
```

Out[49]:

	Model	Train_Score	Test_Score	Accuracy	Recall	Precision	F1 score	AUC
0	logistic regression	0.87	0.86	86.21	27.27	55.00	36.46	78.91
1	Decision tree	1.00	0.94	94.36	79.34	81.36	80.33	88.13
2	Random Forest	1.00	0.96	96.40	76.03	98.92	85.98	90.10
3	Xgboost	1.00	0.97	97.12	80.17	100.00	88.99	90.63

Summary

- All models achieved strong performance with accuracy above 85%. **XGBoost** recorded the highest test accuracy (97.12%), while **Logistic Regression** had the lowest (86.21%).
- **Logistic Regression** suffers from very low recall (27.27%), meaning it misses most churned customers. Tree-based models perform much better: Decision Tree (79.34%), Random Forest (76.03%), and XGBoost (80.17%).
- **XGBoost** achieved the highest precision (100), with **Random Forest** having a precision score of (98.92%). In contrast, Logistic Regression lagged behind (55.00%).
- Looking at the **F1 Score** (balancing precision and recall), **XGBoost** leads (88.99%), followed by **Random Forest** (85.98%) and **Decision Tree** (80.33%). Logistic Regression again performs the weakest (36.46%).
- **AUC scores** reinforce this pattern: XGBoost (90.63%) and Random Forest (90.10%) are highest, Decision Tree is slightly lower (88.13%), and Logistic Regression trails (78.91%).
- All tree-based models (Decision Tree, Random Forest, XGBoost) show perfect training scores (1.00) but slightly lower test scores, indicating some degree of **overfitting**, though performance on unseen data remains strong.
- The presence of **perfect training scores (1.00)** for Decision Tree, Random Forest, and XGBoost suggests that the models may be memorizing the training data rather than generalizing. This could be driven by **multicollinearity** as identified in our heatmap before causing instability in predictions.

6.3: Model_2: No Multicollinearity

In [50]:

```
#using df2 where we removed the feature engineered columns and perfectly related features
# separating features and target
X = df2.drop("churn",axis=1)
y = df2.churn

#splitting to train and test set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42,stratify=y)

#Scaling featues
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#fitting models
models = {'logistic regression':LogisticRegression(random_state=42,max_iter=1000),
          'Decision tree': DecisionTreeClassifier(random_state=42),
          'Random Forest':RandomForestClassifier(random_state=42,n_estimators=100),
          'Xgboost':XGBClassifier(random_state=42,use_label_encoder=False,eval_metric='logloss')
          }
model_2_results = []
for name, model in models.items():
    model.fit(X_train_scaled,y_train)
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:,1]

    model_2_results.append({'Model':name,
                    'Train_Score':model.score(X_train_scaled,y_train),
                    'Test_Score':model.score(X_test_scaled,y_test),
                    'Accuracy':accuracy_score(y_test,y_pred)*100,
                    'Recall':recall_score(y_test,y_pred)*100,
                    'Precision':precision_score(y_test,y_pred)*100,
                    'F1 score':f1_score(y_test,y_pred)*100,
                    'AUC':roc_auc_score(y_test,y_prob)*100
                })
model_2_results_df = pd.DataFrame(model_2_results).round(2)[["Model", "Train_Score", "Test_Score", "Accuracy
model_2_results_df
```

Out[50]:

	Model	Train_Score	Test_Score	Accuracy	Recall	Precision	F1 score	AUC
0	logistic regression	0.87	0.86	86.21	27.27	55.00	36.46	78.89
1	Decision tree	1.00	0.91	91.37	66.94	71.68	69.23	81.23
2	Random Forest	1.00	0.92	92.33	52.89	90.14	66.67	89.11
3	Xgboost	1.00	0.95	94.72	73.55	88.12	80.18	89.16

Summary (After Handling Multicollinearity)

- All models continue to perform well, with accuracy above 85%. **XGBoost** achieved the highest test accuracy (94.72%), while **Logistic Regression** remains the lowest (86.21%).
- **Logistic Regression** still has very low recall (27.27%), missing most churned customers. Tree-based models show stronger recall: Decision Tree (66.94%), Random Forest (52.89%), and XGBoost (73.55%).
- **Random Forest** achieved the highest precision (90.14%), followed by **XGBoost** (88.12%) and Decision Tree (71.68%). Logistic Regression again lagged behind (55.00%).
- In terms of **F1 Score**, **XGBoost** leads (80.18%), followed by Decision Tree (69.23%) and Random Forest (66.67%). Logistic Regression remains weak (36.46%).
- **AUC scores** show that ensemble models continue to be strong: Random Forest (89.11%) and XGBoost (89.16%) perform best, Decision Tree is lower (81.23%), and Logistic Regression is weakest (78.89%).
- After handling **multicollinearity**, test scores are slightly lower than before, but results are more realistic and generalizable. The gap between training and test performance has narrowed, indicating **reduced overfitting** and improved model reliability.
- Overall, **XGBoost** provides the best balance across metrics, making it the strongest candidate for deployment in predicting customer churn.

Metrics Decision

For our imbalanced dataset **14.5% churned**, the following metrics are key and will inform the rest of the analysis:

Selected Metrics

- **F1 Score**: A Balance between recall and precision, especially crucial for our imbalanced dataset. Provides a single metric summarizing our models ability to correctly identify customers who churn while avoiding false positives.

- **Precision**: The main goal is to correctly identify churned customers while minimizing false positives. High precision ensures retention efforts focus on customers who are very likely to churn, avoiding unnecessary spend on those unlikely to churn.

Dropped Metrics

- **Recall**: Measures how many actual churners are captured. We accept missing a few customers to optimize retention efforts on customers highly likely to churn.
- **Accuracy** is less useful based on our imbalanced dataset. Since a model predicting all “Not churn” could still have a 85% plus accuracy but fail the main business goal.

Model Selection Rationale

- Based on Model_2 results without multicollinearity and considering class imbalance exists on the dataset, **XGBoost** and **Random Forest** will be used for further testing and analysis.
- **XGBoost** offers the highest F1 score (80.18%) and second best precision (88.12%) for detecting churned customers.
- **Random Forest** Has the highest precision (90.14%) for highly reliable predictions and it is generally robust to noise.
- **Excluded models:** Decision Tree and Logistic Regression were excluded due to overfitting and poor recall and F1 on this imbalanced dataset.
- **Next steps:** Applying class imbalance techniques and reevaluate precision, recall, and F1 to ensure high-risk churners are correctly identified.

6.4: Model_3: Handling Class Imbalance

In [51]:
```
# separating features and target
X = df2.drop("churn",axis=1)
y = df2.churn

#splitting to train and test set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25,random_state=42,stratify=y)


#balance the training set
smote = SMOTE(random_state=42)
X_train_smote, y_train_smote = smote.fit_resample(X_train,y_train)

#Scaling featues
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_smote)
X_test_scaled = scaler.transform(X_test)

#fitting models
models = {'Decision tree': DecisionTreeClassifier(random_state=42),
          'Random Forest':RandomForestClassifier(random_state=42,n_estimators=100),
          'Xgboost':XGBClassifier(random_state=42,use_label_encoder=False,eval_metric='logloss')
          }
model_3_results = []
for name, model in models.items():
    model.fit(X_train_scaled,y_train_smote)
    y_pred = model.predict(X_test_scaled)
    y_prob = model.predict_proba(X_test_scaled)[:,1]

    model_3_results.append({'Model':name,
                    'Train_Score':model.score(X_train_scaled,y_train_smote),
                    'Test_Score':model.score(X_test_scaled,y_test),
                    'Accuracy':accuracy_score(y_test,y_pred)*100,
                    'Recall':recall_score(y_test,y_pred)*100,
                    'Precision':precision_score(y_test,y_pred)*100,
                    'F1 score':f1_score(y_test,y_pred)*100,
                    'AUC':roc_auc_score(y_test,y_prob)*100
                })
model_3_results_df = pd.DataFrame(model_3_results).round(2)[["Model", "Train_Score", "Test_Score", "Accuracy
model_3_results_df
```

Out[51]:

	Model	Train_Score	Test_Score	Accuracy	Recall	Precision	F1 score	AUC
0	Decision tree	1.0	0.86	85.73	62.81	50.67	56.09	76.22
1	Random Forest	1.0	0.90	90.29	48.76	75.64	59.30	86.65
2	Xgboost	1.0	0.94	94.24	73.55	84.76	78.76	89.53

Summary

Our two selected models performed strongly on the key metrics, especially **F1 score** and **precision**.

- **F1 Score** is highest for **XGBoost (78.76%)**, followed by **Random Forest (59.30%)**, showing that both models provide a good balance between capturing churners and avoiding false positives.
- **Precision**, critical for this operational scenario, is highest for **XGBoost (84.76%)**, with **Random Forest close behind (75.64%)**, meaning these models are highly reliable when predicting churn and help minimize unnecessary retention efforts.
- **Impact of SMOTE Resampling:** Applying SMOTE to balance the training set slightly altered model performance. Both models saw a decrease in both metrics compared to the previous model. This shows that resampling helped address class imbalance but did not universally improve all metrics.

6.5: Applying Cross Validation

In [52]:

```
# Defining custom scoring metrics
scoring_metrics = {
    'accuracy': 'accuracy',
    'recall': 'recall',
    'precision': 'precision',
    'f1': 'f1',
    'roc_auc': 'roc_auc',
    'pr_auc': 'average_precision'
}

# stratified K-fold cross-validation
kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

# Performing cross-validation for each model
cv_summary = {}
for model_name, model_instance in models.items():
    results = cross_validate(model_instance, X, y, cv=kf, scoring=scoring_metrics, return_train_score=True)

    # Computing mean test scores for each metric
    mean_test_scores = {metric: results[f'test_{metric}'].mean() * 100 for metric in scoring_metrics.keys()}
    mean_test_scores['train_accuracy'] = results['train_accuracy'].mean() * 100
    cv_summary[model_name] = mean_test_scores

# Converting results to DataFrame
cv_summary_df = pd.DataFrame(cv_summary).T.round(2)
cv_summary_df
```

Out[52]:

	accuracy	recall	precision	f1	roc_auc	pr_auc	train_accuracy
Decision tree	91.81	72.47	71.64	71.87	83.78	55.98	100.0
Random Forest	93.25	55.71	96.07	70.25	91.61	84.95	100.0
Xgboost	95.62	77.23	91.54	83.53	91.61	87.08	100.0

Summary:

- Our models **XGBoost** and **Random Forest**, performed well on the key operational metrics: **F1 score** and **precision**.

- **F1 Score** is highest for **XGBoost (83.53%)**, followed by **Random Forest (70.25%)**, showing that both models metrics have improved in comparison without cross validation.
- **Precision**, is now highest for **Random Forest (96.07%)**, with **XGBoost** even though improved by +7 **(91.54%)**.
- Both models are strong, but the overall pick is **XGBoost** since it slightly better captures churners overall (higher F1) compared to **Random Forest** is more conservative.

6.6: Applying Hyperparameter tuning with grid search

In [53]:

```
xgb = XGBClassifier(random_state=42,use_label_encoder=False,eval_metric='logloss')
param_grid_xgb = {'learning_rate':[0.01,0.05,0.1],
               'max_depth':[3,5,7,9],
               'n_estimators':[100,200,300]
}
grid_search_xgb = GridSearchCV(scoring='f1',param_grid=param_grid_xgb, cv=5, estimator=xgb)
grid_search_xgb.fit(X_train_scaled,y_train_smote)
```

Out[53]:

```
▸      GridSearchCV
▸ estimator: XGBClassifier
     ▸ XGBClassifier
```

In [54]:

```
best_xgb = grid_search_xgb.best_estimator_
print(f'The best parameters are: {grid_search_xgb.best_params_}')
```

```
The best parameters are: {'learning_rate': 0.05, 'max_depth': 9, 'n_estimators': 300}
```

In [55]:

```
# y pred and y_prob
y_best_pred = best_xgb.predict(X_test_scaled)
y_best_prob = best_xgb.predict_proba(X_test_scaled)[:,1]
report_dict = classification_report(y_test, y_best_pred, output_dict=True)
report_df = pd.DataFrame(report_dict).transpose()
report_df
```

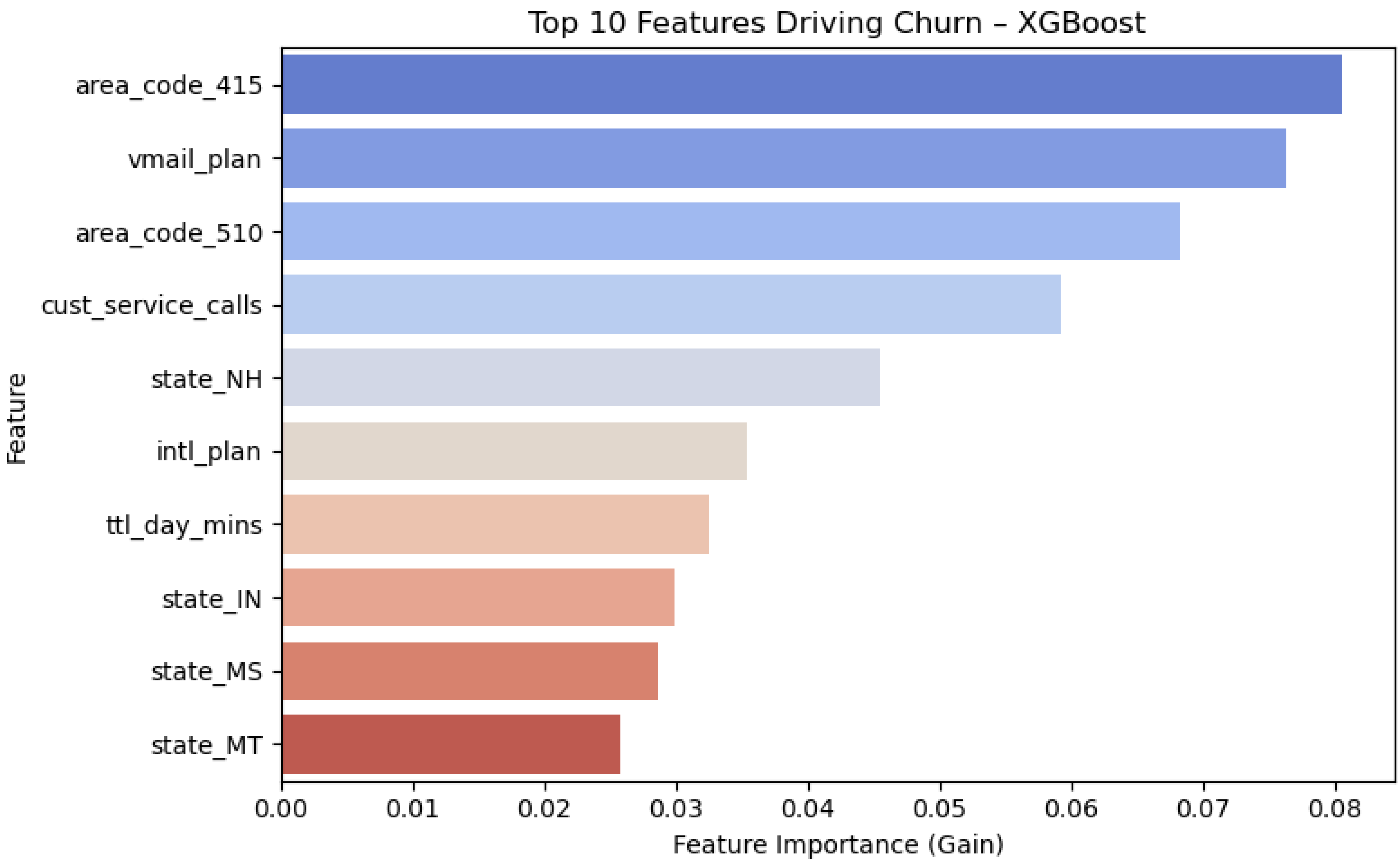
Out[55]:

	precision	recall	f1-score	support
0	0.955923	0.973352	0.964559	713.000000
1	0.824074	0.735537	0.777293	121.000000
accuracy	0.938849	0.938849	0.938849	0.938849
macro avg	0.889998	0.854445	0.870926	834.000000
weighted avg	0.936794	0.938849	0.937389	834.000000

Final XGBoost Model Performance Interpretation

- The model is **highly precise** for both classes **Churn class Precision (0.82) and Non_Churn Class Precision (0.95)**, ensuring that predictions are reliable.
- **Churn class F1-Score (0.77)** indicates a strong balance between capturing actual churners and minimizing false positives.
- This configuration is suitable for operational deployment where **precision and F1** are key to targeting high-risk customers effectively.

In [56]:
```
# Extracting feature importances from the trained XGBoost model
feature_names = X.columns
feature_importances = best_xgb.feature_importances_

# Creating a DataFrame of top 10 features by importance
top_features = (pd.Series(feature_importances, index=feature_names).sort_values(ascending=False).head(10))

plt.figure(figsize=(8, 5))
sns.barplot(x=top_features.values, y=top_features.index, palette='coolwarm')
plt.title('Top 10 Features Driving Churn – XGBoost')
plt.xlabel('Feature Importance (Gain)')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()
```



Final Model Gain Importance of top 10 Features

1. **Area Code 415** – customers with this area code show higher churn likelihood, Possibly regional effects.
2. **Voice Mail Plan** – Not having a voice mail plan increases churn risk, indicating value-add plan importance.
3. **Area Code 510** – Another regional factor contributing to churn, though slightly weaker than 415.
4. **Customer Service Calls** – High number of calls is a strong early warning for dissatisfaction.
5. **State = NH** – Geographic effect
6. **International Plan** – Presence of an international plan increases churn risk as plan holders are more critical of quality of service.
7. **Total Day Minutes** – Heavy daytime usage correlates with higher churn, possibly due to frustration with pricing.
8. **State = IN** – Regional factor contributing modestly to churn likelihood.

9. **State = MS** – Minor geographic influence.
10. **State = MT** – Weakest among top 10, but still detectable in the model.

Business takeaway: churn is concentrated among customers with **No Voice Mail plan**, **high daytime usage**, **multiple service calls**, and certain **regional characteristics** i.e Area Code 415 and 510. Retention strategies should focus on these high-risk segments with targeted offers and proactive engagement

7: Conclusion and Recommendations

7.1: Conclusion

Insights from EDA

- Customers with **international plans** and those making frequent **customer service calls** are more likely to churn, suggesting dissatisfaction or unmet service expectations.
- Higher **total charges** are associated with churn, especially for customers without international plans, indicating that cost sensitive users are at risk.

Inferential Analysis Insights

- **Chi square** and **T-tests** confirmed that **voicemail plans** and **total charges** are significantly associated with churn.
- **One way ANOVA** showed that **area code** alone is not a statistically significant factor for predicting churn, reinforcing that churn drivers are more behavioral than geographic. However, this is disproved in our feature importance during modelling

Modeling Insights

- **XGBoost** and **Random Forest** are the top performing models after feature engineering and handling class imbalance.
- **XGBoost** achieves the highest **F1 score (78%)**, balancing recall and precision. Final model **Precision (82.4%)**
- Feature engineering led to additional multicollinearity and the columns had to be dropped during modeling.
- **Area Code**, **Voice Mail Plan**, **International plan** and **Customer service callls** are the major drivers of churn risk
- **SMOTE resampling** successfully addressed class imbalance without compromising predictive performance.

7.2: Recommendations

1. Focus Retention on At-Risk Segments

- Identify customers with **International Plans** and those making **frequent customer support calls**.
- Offer proactive outreach, loyalty rewards, or usage based incentives to these high risk groups.

2. Improve Quality of Service and Billing

- Optimize customer service to reduce repeat or unresolved support calls.
- Implement **flatrate billing** for heavy users, allowing them to enjoy more minutes and data after a certain threshold without incurring unexpectedly high charges.

3. Leverage Plan Features to Reduce Churn

- Encourage adoption of **voicemail plans** for high_usage customers, as those without voicemail show higher churn and usage variability.
- Offer **customized international or bundled plans** to customers with heavy international calling to reduce churn driven by high charges.
- Educate customers on plan benefits to highlight convenience, predictability, and cost savings.

4. Regional and State-Level Retention Strategies

- Investigate and address service quality or infrastructure issues in **area code 415**, where churn is elevated despite high customer volume.
- Deploy **localized promotions** and faster issue resolution to strengthen loyalty in at-risk states.