Choxy1 Final commit

41c7052 · yesterday History

Preview Code Blame 5055 lines (5055 loc) · 784 KB

Raw

1.0. Business Understanding

Overview

The global telecommunications industry is known for intense competition and unpredictable business conditions, making it challenging for less resilient companies. Customer retention fluctuates dramatically due to factors such as economic downturns, rising costs, competitive alternatives, rapid technological advancements, globalization, government intervention, and various other influences.

Problem Statement

Syria Telecommunications is grappling with the same challenges as other major telecom companies, but possibly at a more severe level. While the industry standard churn rate falls between 5% and 7%, Syria Telecommunications has seen its churn rate spike to nearly 15% at the time of data collection. In response, the company's management has tasked the Data Science Department with the responsibility of gathering, cleaning, and analyzing data to uncover the reasons behind this alarming rate and to propose practical solutions to address the issue.

Business Problem

Despite the potential for booming profits and increased market share, Syria Telecommunications is experiencing a decline in customer retention. This downward trend threatens to steer the company away from its business objectives and hinder its growth.

Objectives

To identify factors leading to increased churn rates

To create a classification model that predicts whether a customer will churn with a recall of over 80%

To give customer retention recommendations

2.0 Data understanding

The data was sourced from Kaggle.

There are 3333 records and 21 features in the data.

Associated columns included are:

State: The location of the customer.

Account Length: The number of days the account was held by the customer.

Area Code: The area code of the customer.

Phone Number: Phone number assigned to the user.

International Plan: Indicator of whether the customer has an international plan.

Voice Mail Plan: Indicator of whether the customer has a voicemail plan.

Number Vmail Messages: Number of voicemails sent.

Total Day Minutes: Number of minutes the customer has been in calls during the day.

Total Day Calls: Total calls made during the day.

Total Day Charge: Billed charge to the customer for all day calls.

Total Eve Minutes: Number of minutes the customer has been in calls during the evening.

Total Eve Calls: Total calls made during the evening.

Total Eve Charge: Billed charge to the customer for all evening calls.

Total Night Minutes: Number of minutes the customer has been in calls during the night.

Total Night Calls: Total calls made during the night.

Total Night Charge: Billed charge to the customer for all night calls.

Total Intl Minutes: Total number minutes on international calls.

Total Intl Calls: Total internation calls made.

Total Intl Charge: Billed charge to the customer for all international calls.

Customer Service Calls: Number of calls made to customer service.

Churn: Indication of whether the customer terminated their contract.

3.0 Data Preparation

We explore the data by understanding aspects of the data - including, loading the data, identifying the shape, features and data types, and understanding the summary statistics.

In [3]:

```
# Data Exploration
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Preprocessing and Metric Evaluation
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.metrics import recall_score, accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from imblearn.over_sampling import SMOTE

# Models
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestClassifier
#import xgboost as xb
```

In [4]:

```
class DataPreparation():
    """
    This class takes a dataframe and returns some basic information.
    Consider making this ONE METHOD that returns everything.
    """
    def __init__(self, data):
        self.data = data

    def read_head(self):
        """Returns the first 5 rows"""
        return self.data.head()

    def read_columns(self):
        """Returns the columns of the DataFrame"""
        return self.data.columns

    def read_info(self):
        """Returns the features, datatypes and non-null count"""
        return self.data.info()

    def read_describe(self):
        """Returns the statistical summary of the dataset"""
        return self.data.describe()

    def read_shape(self):
        """Returns the number of rows and columns"""
        return self.data.shape

    def read_corr(self):
        """Returns a correlation dataframe"""
        return self.data.corr()

    def read_corr_wrt_target(self, target='churn'):
        """Returns a Series containing the correlation of features with respect to target"""
        return self.data.corr()[target].sort_values(ascending=False)

    def read_multicollinearity(self, target='churn'):
        """Returns a correlation dataframe without the target"""
        return self.data.corr().iloc[0:-1, 0:-1]

    def read_na(self):
        """Returns the sum of all null values per feature"""
        return self.data.isna().sum()

    def read_duplicated(self):
        """Returns the sum of all duplicated records"""
        return self.data.duplicated().sum()
```

In [5]:

```
# The data in DataFrame
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
```

In [6]:

```
# A data preparation Object Instatiated
dp = DataPreparation(data=df)

# First 5 lines of the DataFrame
dp.read_head()
```

Out[6]:

	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	total night charge	mir
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	

5 rows × 21 columns

In [7]:

```
# Explore the column names
dp.read_columns()
```

Out[7]:

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

In [8]:

```
# Explore features and their datatypes
dp.read_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 [9]:

```
# Explore the statistical summary
dp.read_describe()
```

Out[9]:

	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	total night minutes
count	3333.000000	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	200.872037
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625	4.310668	50.573847
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	23.200000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000	14.160000	167.000000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000	17.120000	201.200000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000	20.000000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000	30.910000	395.000000

In [10]:

```
# Explore any correlations
dp.read_corr()
```

Out[10]:

	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	total night minutes	total night calls	tota nigh charg
account length	1.000000	-0.012463	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.019260	-0.006745	-0.008955	-0.013176	-0.00896
area code	-0.012463	1.000000	-0.001994	-0.008264	-0.009646	-0.008264	0.003580	-0.011886	0.003607	-0.005825	0.016522	-0.00584
number vmail messages	-0.004628	-0.001994	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.005864	0.017578	0.007681	0.007123	0.00766
total day minutes	0.006216	-0.008264	0.000778	1.000000	0.006750	1.000000	0.007043	0.015769	0.007029	0.004323	0.022972	0.00430
total day calls	0.038470	-0.009646	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.006462	-0.021449	0.022938	-0.019557	0.02292
total day charge	0.006214	-0.008264	0.000776	1.000000	0.006753	1.000000	0.007050	0.015769	0.007036	0.004324	0.022972	0.00430
total eve minutes	-0.006757	0.003580	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.011430	1.000000	-0.012584	0.007586	-0.01259
total eve calls	0.019260	-0.011886	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.000000	-0.011423	-0.002093	0.007710	-0.00205
total eve charge	-0.006745	0.003607	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.011423	1.000000	-0.012592	0.007596	-0.01260
total night minutes	-0.008955	-0.005825	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.002093	-0.012592	1.000000	0.011204	0.99999

minutes												
total night calls	-0.013176	0.016522	0.007123	0.022972	-0.019557	0.022972	0.007586	0.007710	0.007596	0.011204	1.000000	0.01118
total night charge	-0.008960	-0.005845	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.002056	-0.012601	0.999999	0.011188	1.00000
total intl minutes	0.009514	-0.018288	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.008703	-0.011043	-0.015207	-0.013605	-0.01521
total intl calls	0.020661	-0.024179	0.013957	0.008033	0.004574	0.008032	0.002541	0.017434	0.002541	-0.012353	0.000305	-0.01232
total intl charge	0.009546	-0.018395	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.008674	-0.011074	-0.015180	-0.013630	-0.01518
customer service calls	-0.003796	0.027572	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.002423	-0.012987	-0.009288	-0.012802	-0.00927
churn	0.016541	0.006174	-0.089728	0.205151	0.018459	0.205151	0.092796	0.009233	0.092786	0.035493	0.006141	0.03549

In [11]:

```python
# Investigating feature correlation within themselves
dp.read_multicollinearity()
```

Out[11]:

	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	total night minutes	total night calls	tota nigh charg
account length	1.000000	-0.012463	-0.004628	0.006216	0.038470	0.006214	-0.006757	0.019260	-0.006745	-0.008955	-0.013176	-0.00896
area code	-0.012463	1.000000	-0.001994	-0.008264	-0.009646	-0.008264	0.003580	-0.011886	0.003607	-0.005825	0.016522	-0.00584
number vmail messages	-0.004628	-0.001994	1.000000	0.000778	-0.009548	0.000776	0.017562	-0.005864	0.017578	0.007681	0.007123	0.00766
total day minutes	0.006216	-0.008264	0.000778	1.000000	0.006750	1.000000	0.007043	0.015769	0.007029	0.004323	0.022972	0.00430
total day calls	0.038470	-0.009646	-0.009548	0.006750	1.000000	0.006753	-0.021451	0.006462	-0.021449	0.022938	-0.019557	0.02292
total day charge	0.006214	-0.008264	0.000776	1.000000	0.006753	1.000000	0.007050	0.015769	0.007036	0.004324	0.022972	0.00430
total eve minutes	-0.006757	0.003580	0.017562	0.007043	-0.021451	0.007050	1.000000	-0.011430	1.000000	-0.012584	0.007586	-0.01259
total eve calls	0.019260	-0.011886	-0.005864	0.015769	0.006462	0.015769	-0.011430	1.000000	-0.011423	-0.002093	0.007710	-0.00205
total eve charge	-0.006745	0.003607	0.017578	0.007029	-0.021449	0.007036	1.000000	-0.011423	1.000000	-0.012592	0.007596	-0.01260
total night minutes	-0.008955	-0.005825	0.007681	0.004323	0.022938	0.004324	-0.012584	-0.002093	-0.012592	1.000000	0.011204	0.99999
total night calls	-0.013176	0.016522	0.007123	0.022972	-0.019557	0.022972	0.007586	0.007710	0.007596	0.011204	1.000000	0.01118
total night charge	-0.008960	-0.005845	0.007663	0.004300	0.022927	0.004301	-0.012593	-0.002056	-0.012601	0.999999	0.011188	1.00000
total intl minutes	0.009514	-0.018288	0.002856	-0.010155	0.021565	-0.010157	-0.011035	0.008703	-0.011043	-0.015207	-0.013605	-0.01521
total intl calls	0.020661	-0.024179	0.013957	0.008033	0.004574	0.008032	0.002541	0.017434	0.002541	-0.012353	0.000305	-0.01232
total intl charge	0.009546	-0.018395	0.002884	-0.010092	0.021666	-0.010094	-0.011067	0.008674	-0.011074	-0.015180	-0.013630	-0.01518
customer service calls	-0.003796	0.027572	-0.013263	-0.013423	-0.018942	-0.013427	-0.012985	0.002423	-0.012987	-0.009288	-0.012802	-0.00927

4.0 Data Cleaning

Missing Values

In [12]:

```python
# Find the number of missing values
dp.read_na()
```

Out[12]:

```
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 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
```

Duplicates

In [13]:

```
# Check for Duplicates
dp.read_duplicated()
```

Out[13]:

```
0
```

Conversion of feature datatypes

Convert the area code into a Categorical Dtype using pandas pd.Categorical

In [14]:

```
# Conversion of the 'area code' into a Categorical dtype
df['area code'] = pd.Categorical(df['area code'])
df['area code'].dtype
```

Out[14]:

```
CategoricalDtype(categories=[408, 415, 510], ordered=False)
```

Conversion of feature values

Convert the churn values using pandas .replace method

In [15]:

```
df['churn'] = df['churn'].replace([False, True], ['False', 'True'])
```

Drop unnecessary columns

Drop the phone number feature

In [16]:

```
df = df.drop('phone number', axis=1)

# Test
df.columns
```

Out[16]:

```
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')
```

5.0. Data Exploration

Split the data between the categorical columns and the numerical columns

In [17]:

```
# Split the Categorical columns from the Numerical Columns
num_cols = df.select_dtypes(include=['number', 'float', 'int']).columns
cat_cols = df.select_dtypes(exclude=['number', 'float', 'int']).columns
```

In [18]:

```
def view_value_counts(cat_columns):
    for item in cat_columns:
        print(item.capitalize())
        values = df[item].value_counts()
        print(values[:10] if len(values) > 10 else values)
        print("")

view_value_counts(cat_cols)
```

```
State
WV    106
MN     84
NY     83
AL     80
OH     78
WI     78
OR     78
VA     77
WY     77
CT     74
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
```

Comment

Our top customers are come from West Virginia, Minesota and Virginia.

The area code with the highest clientelle base 415.

The ratio between customers without an international plan against those who do is around 10 to 1.

The ratio between customers who don't churn against those who do is around 17 to 1.

In [19]:
```
df.groupby('churn').median().loc[:,]
```

Out[19]:

churn	account length	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
False	100	0	177.2	100	30.12	199.6	100	16.97	200.25	100	9.01	10.2	4	2.75	1
True	103	0	217.6	103	36.99	211.3	101	17.96	204.80	100	9.22	10.6	4	2.86	2

In [20]:
```
df.groupby('churn').mean().loc[:,]
```

Out[20]:

churn	account length	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	to m
False	100.793684	8.604561	175.175754	100.283158	29.780421	199.043298	100.038596	16.918909	200.133193	100.058246	9.006074	10.
True	102.664596	5.115942	206.914079	101.335404	35.175921	212.410145	100.561077	18.054969	205.231677	100.399586	9.235528	10.

In [21]:
```
# Maintain seperate datasets were 'churn' is either True or False
stayed_df = df[df['churn']=='False']
left_df = df[df['churn']!='False']
```

Which are some differentiating features between the customers who stayed versus those who left?

Area Code

In [22]:
```
df['area code'].value_counts(normalize=True)
```

Out[22]:
```
415    0.496550
510    0.252025
408    0.251425
Name: area code, dtype: float64
```

In [23]:
```
stayed_df['area code'].value_counts(), left_df['area code'].value_counts()
```

Out[23]:
```
(415    1419
 408     716
 510     715
 Name: area code, dtype: int64,
 415    236
 510    125
 408    122
 Name: area code, dtype: int64)
```

In [24]:
```
stayed_df['area code'].value_counts(normalize=True), left_df['area code'].value_counts(normalize=True)
```

Out[24]:
```
(415    0.497895
 408    0.251228
 510    0.250877
 Name: area code, dtype: float64,
 415    0.488613
 510    0.258799
 408    0.252588
 Name: area code, dtype: float64)
```

Area code 415 represents the area code where most of our customers stayed and left at the same time. The ratios between the stayed versus left customers seems consistent

Customer Service Calls

In [25]:
```
stayed_df['customer service calls'].mean(), left_df['customer service calls'].mean()
```

Out[25]:
```
(1.4498245614035088, 2.229813664596273)
```

In [26]:
```
csc_states = df.groupby(['state']).sum()[['customer service calls']].sort_values(by='customer service calls',ascending=False)
csc_states
```

Out[26]:

	customer service calls
state	
WV	159
NY	142
OR	135
MN	130
VT	127
AL	125
VA	123
ID	122
IN	120
MI	119

In [27]:
```
avg_css_states = df.groupby(['state']).mean()[['customer service calls']].sort_values(by='customer service calls',ascending=F
avg_css_states
```

Out[27]:

	customer service calls
state	
AR	1.981818
GA	1.925926
CO	1.787879
OK	1.786885
ME	1.741935
VT	1.739726
OR	1.730769
KY	1.711864
NY	1.710843
MD	1.700000

The mean of the number of calls made to the customer service center were considerably in those made by the customers who stayed.

The state with the highest number of calls to the service center in absolute terms was West Virginia, Oregon and New York.

From an average perspective, the highest averages per state to the customer service centers were from Arizona, Georgia and Colarado.

Domestic Plans

Minutes purchased throughout the three periods - day, evening and night

In [28]:
```
stayed_min = (stayed_df['total day minutes'] + stayed_df['total eve minutes'] + stayed_df['total night minutes']).mean()
left_min = (left_df['total day minutes'] + left_df['total eve minutes'] + left_df['total night minutes']).mean()

stayed_min, left_min
```

Out[28]:
```
(574.352245614035, 624.555900621118)
```

The mean of the minutes purchased by the customers for domestic usage who left was significantly more than those who stayed

Total Charges across the three periods - day, evening and night

In [29]:
```
stayed_charges = (stayed_df['total day charge'] + stayed_df['total eve charge'] + stayed_df['total night charge']).mean()
left_charges = (left_df['total day charge'] + left_df['total eve charge'] + left_df['total night charge']).mean()

stayed_charges, left_charges
```

Out[29]:
```
(55.70540350877194, 62.466418219461694)
```

The mean of the charges for domestic usage for the customers who left was significantly more than those who stayed.

International Plans

International Plans Opted

In [30]:
```
stayed_df['international plan'].value_counts(normalize=True), left_df['international plan'].value_counts(normalize=True)
```

Out[30]:
```
(no     0.934737
 yes    0.065263
 Name: international plan, dtype: float64,
 no     0.716356
 yes    0.283644
 Name: international plan, dtype: float64)
```

6% of the proportion of consumers who stayed had an international plan while 28% of the customers who left had an international plan connected with their account.

Total International Plans - Minutes and Charges

In [31]:
```
df.groupby(['churn', 'international plan']).mean().iloc[:, -4:-1]
```

Out[31]:

churn	international plan	total intl minutes	total intl calls	total intl charge
False	no	10.185473	4.493243	2.750586
	yes	9.777957	5.102151	2.640538
True	no	10.271387	4.251445	2.774017
	yes	11.782482	3.941606	3.181314

In [32]:
```
stayed_df['total intl minutes'].mean(), left_df['total intl minutes'].mean()
```

Out[32]:
```
(10.158877192982455, 10.700000000000001)
```

In [33]:
```
stayed_df['total intl charge'].mean(), left_df['total intl charge'].mean()
```

Out[33]:
```
(2.7434035087719297, 2.8895445134575573)
```

The data indicates higher average values in terms of minutes purchased and bill charged to the consumers who left over those who stayed.

Account Lengths

Account Lengths Exploration

In [34]:
```
stayed_df['account length'].mean(), left_df['account length'].mean()
```

Out[34]:
```
(100.79368421052632, 102.66459627329192)
```

Account Lengths and States

In [35]:
```
df.groupby(['state']).mean()[['account length']].sort_values(by='account length', ascending=False)[:10]
```

Out[35]:

state	account length
FL	109.571429
OK	108.262295
LA	108.235294
KS	106.785714
ND	106.209677
VA	105.935065
WY	105.740260
DC	105.722222
HI	105.471698
SD	105.450000

Account Lengths against Customer Service Calls

In [36]:
```
df.groupby(['churn', 'customer service calls']).mean()[['account length']]
```

Out[36]:

churn	customer service calls	account length

False	0	101.550413
	1	101.386213
	2	99.197917
	3	100.142857
	4	100.333333
	5	109.192308
	6	100.875000
	7	125.000000
	8	64.000000
True	0	99.673913
	1	105.196721
	2	99.436782
	3	112.727273
	4	105.421053
	5	98.250000
	6	84.071429
	7	109.000000
	8	103.000000
	9	102.500000

The customers who left held on their accounts longer than the customers who stayed.

On average, customers from Florida, Oklahoma and Los Angeles had the highest account lengths than all other customers in America.

In addition, customers who left and had longer account lengths seemed to have made more calls to the service centers than the customers who stayed.

COMMENTS

Area codes 415, 408 and 510 share proportions 50%, 25%, and 25% respectively representing the proportion of customers in each area code. The proportions hold true when comparing the data for the customers who churned and those who did not.

The mean number of calls to custmomer service were significantly more in the customers who left than those who stayed.

The mean amount of day minutes and their mean charges for customers who left were significantly more than the customers who stayed.

Only 6% of the customers who stayed did had an international plan whereas 28% of the customers who left had an international plan.

The minutes spent on international calls were higher in the customers who left over the customers who stayed.

The average international call charge was higher in the customers who churned over the customers who stayed.

Account lengths held by the customers who left were slightly more than those held by the customers who stayed.

6.0 Data Visualisation

Univariate Analysis

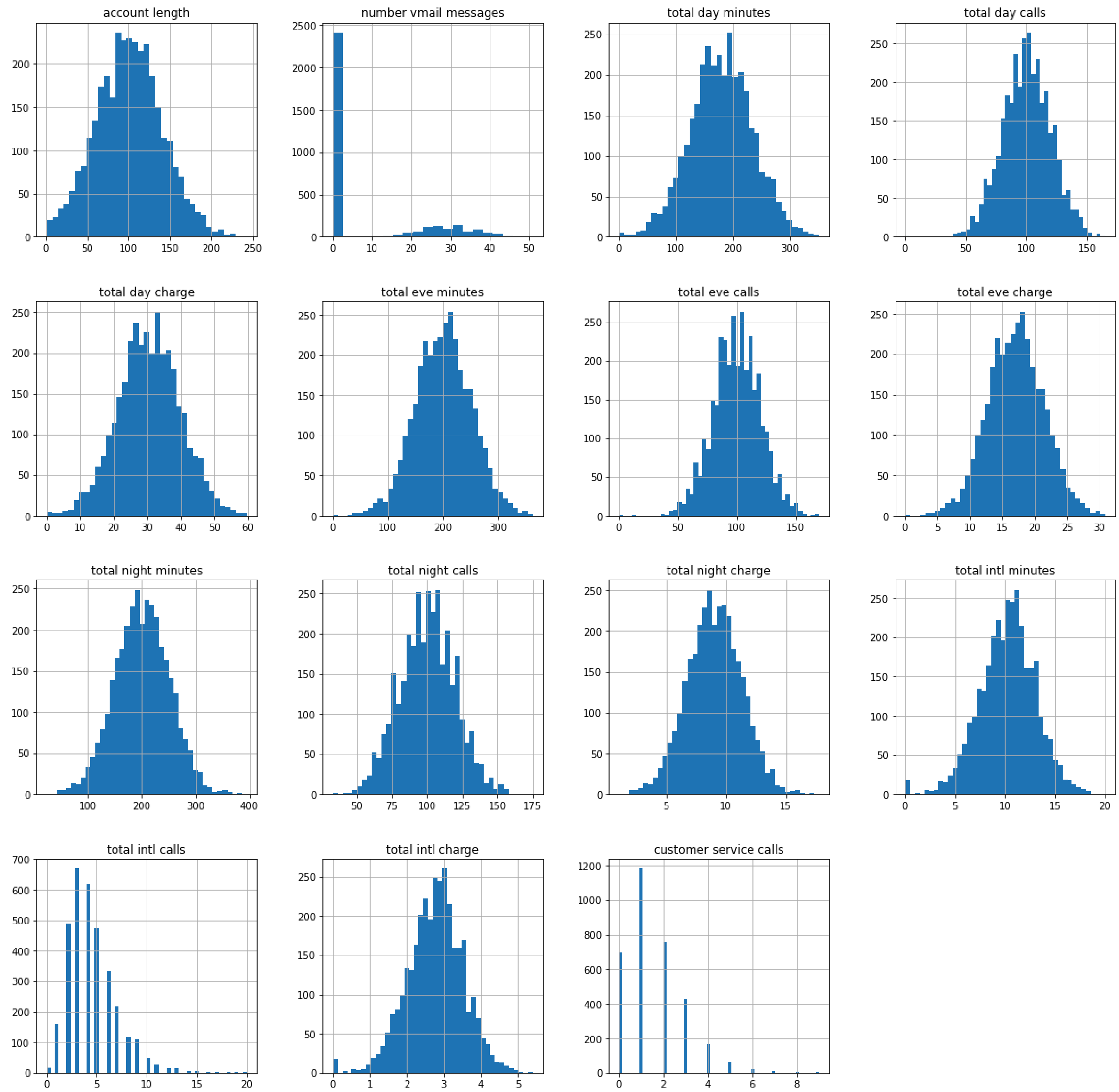
In [37]:

```python
def univariate_plot(col):
    # Style
#     plt.style.use('_mpl-gallery')
    if col in num_cols:

        # plot:
        fig, ax = plt.subplots(figsize=(8, 6))

        ax.hist(df[col], bins=20, linewidth=0.5, edgecolor="white")

        plt.suptitle(f'Distribution of the {col.capitalize()}', fontsize=14)

        ax.set_xlabel(f'{col.capitalize()}')

        ax.set_ylabel('Distribution')

        plt.show()

    else:

        # plot:
        fig, ax = plt.subplots(figsize=(8, 6))

        data = df[col].value_counts(sort=True)

        ax.bar(data.index[:10] if len(data) > 10 else data.index,
               data.values[:10] if len(data) > 10 else data.values,
               linewidth=0.5, edgecolor="white")

        plt.suptitle(f'Distribution of the {col.capitalize()}', fontsize=14)

        ax.set_ylabel('Distribution')

        plt.show()
```

```
plt.show()
```

In [38]:

```
df.hist(bins='auto',figsize=(20,20));
```



COMMENT

Almost all of the distributions reveal a normal distribution of the features.

It is worth noting that the distribution of customer service calls and total intl callsare right-tailed.

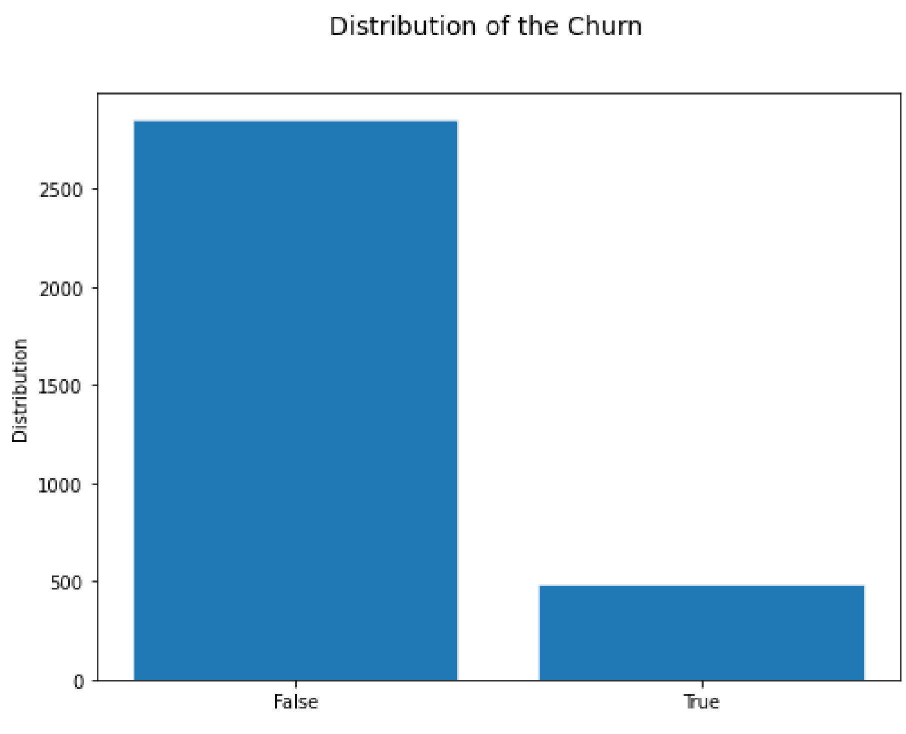
The number vmail messages feature also displays a strange distribution illustrating that alot users opted out of sending/ receiving voice mail messages and very few other people decided to have the voice mail messages.

Visualisation of Categorical features

Churn

In [39]:

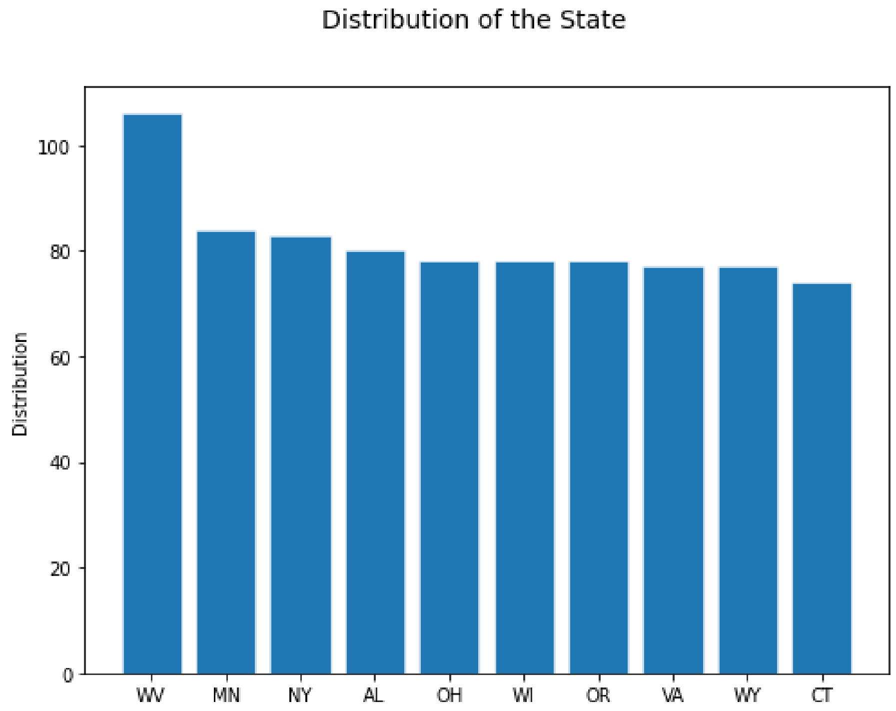
```
univariate_plot('churn')
```



Almost 15% the customers churned while the rest stayed with Syria Telecommunications.

State

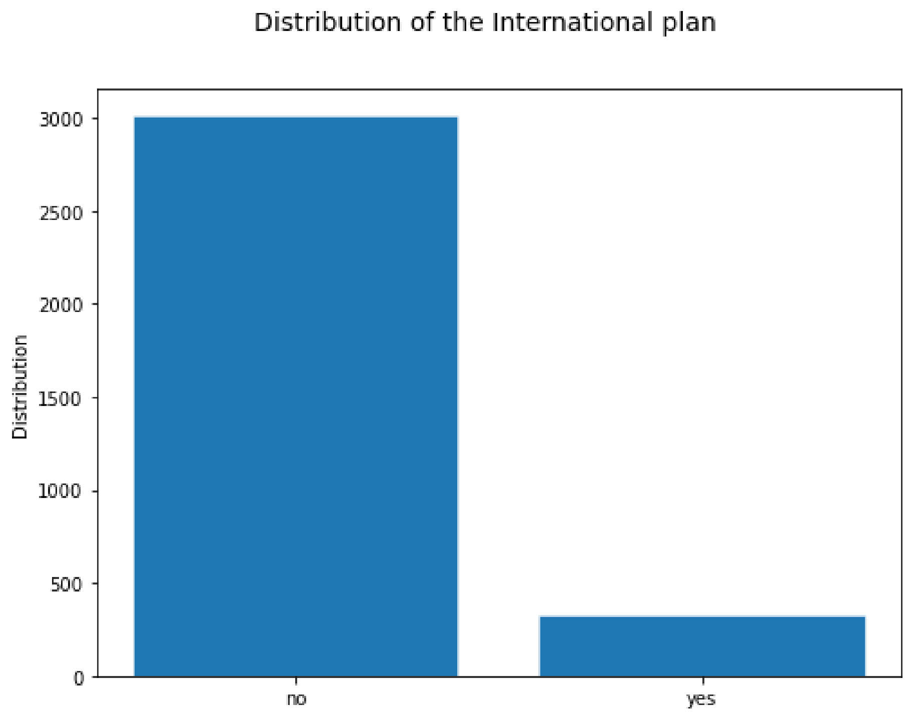
In [40]:
```python
univariate_plot('state')
```



West Virginia, Minnesota and New York contain most of Syria Telecommunications customers

International Plan

In [41]:
```python
univariate_plot('international plan')
```



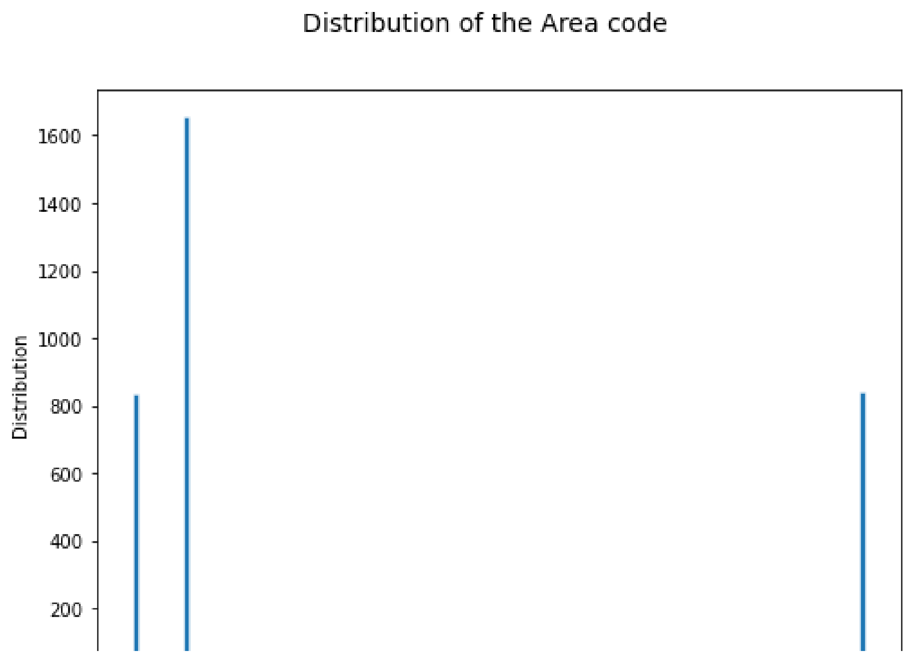
In [42]:
```python
df['international plan'].value_counts(normalize=True)
```

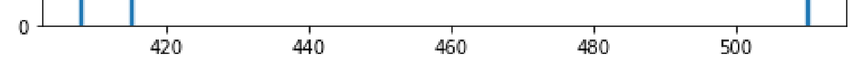
Out[42]:
```
no     0.90309
yes    0.09691
Name: international plan, dtype: float64
```

Near 10% of the customers have an international plan

Area Code

In [43]:
```python
univariate_plot('area code')
```



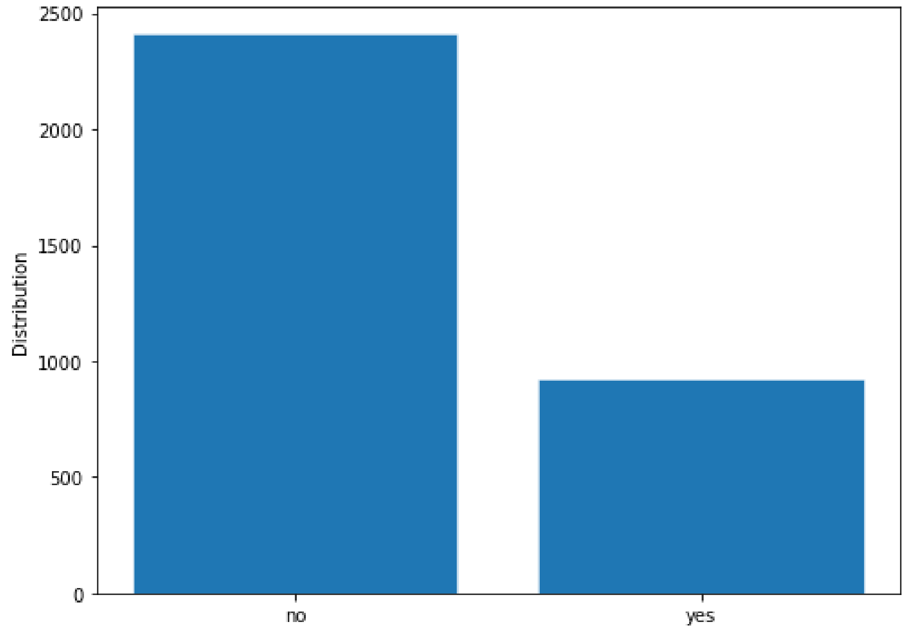


Most of the customers came from the 410 area code region.

Voice Mail plans

In [44]:
```
univariate_plot('voice mail plan')
```

Distribution of the Voice mail plan



In [45]:
```
df['voice mail plan'].value_counts(normalize=True)
```

Out[45]:
```
no      0.723372
yes     0.276628
Name: voice mail plan, dtype: float64
```

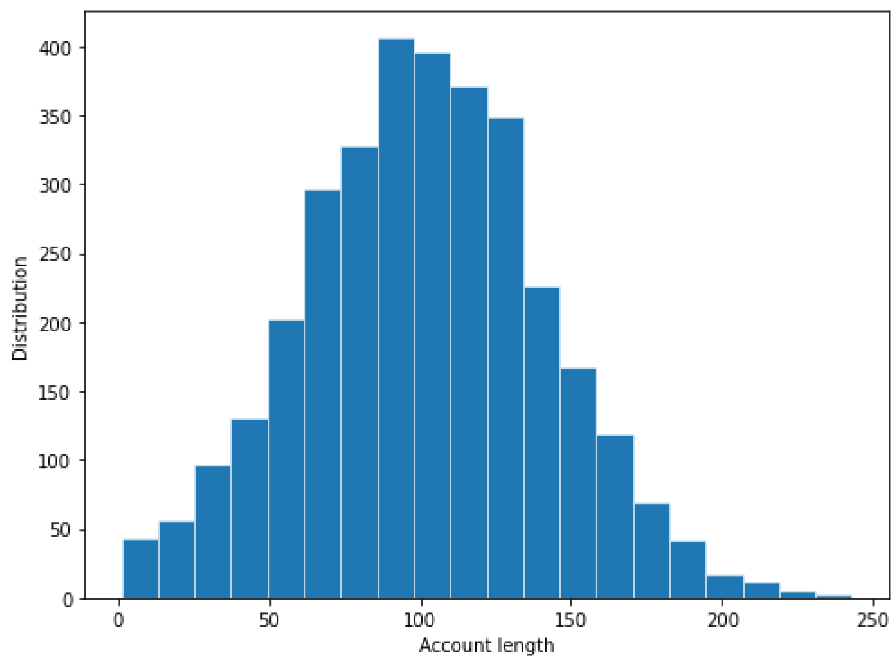
Nearly 28% of the customers have a voice mail plan.

Visualisation of Numerical columns

Account Length

In [46]:
```
univariate_plot('account length')
```

Distribution of the Account length

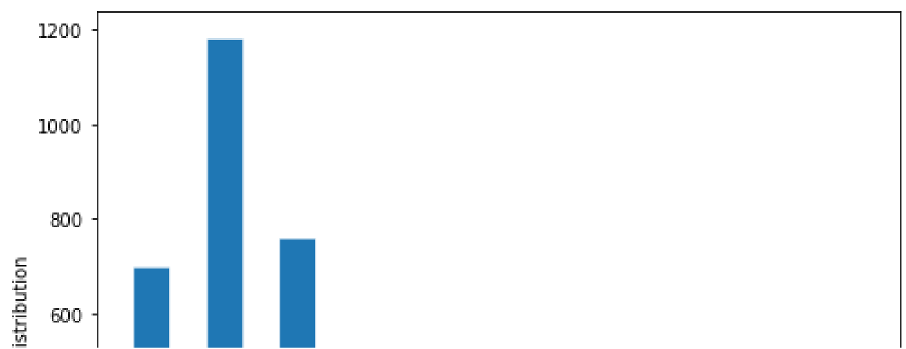


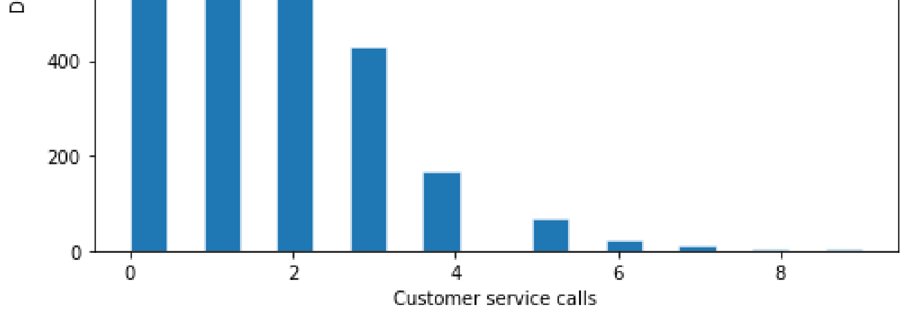
A normal distribuition of the account length feature with the mean and median being around 100.

Customer Service Calls

In [47]:
```
univariate_plot('customer service calls')
```

Distribution of the Customer service calls

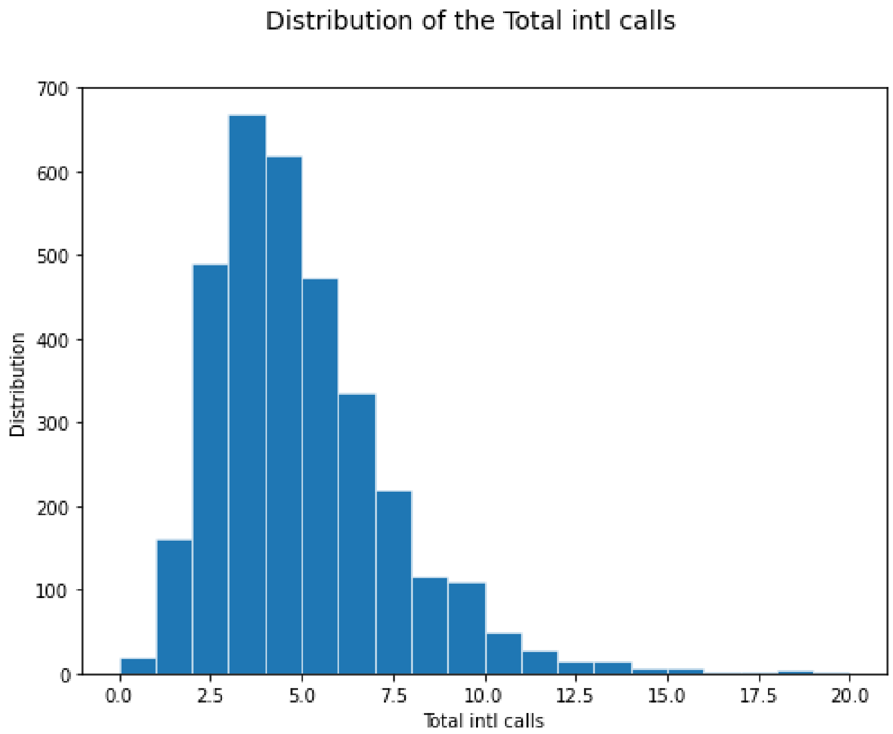




A right tailed customer service showing a lot of the customers have called the customer service at least 2 and fewer people have called more than twice.

International Calls

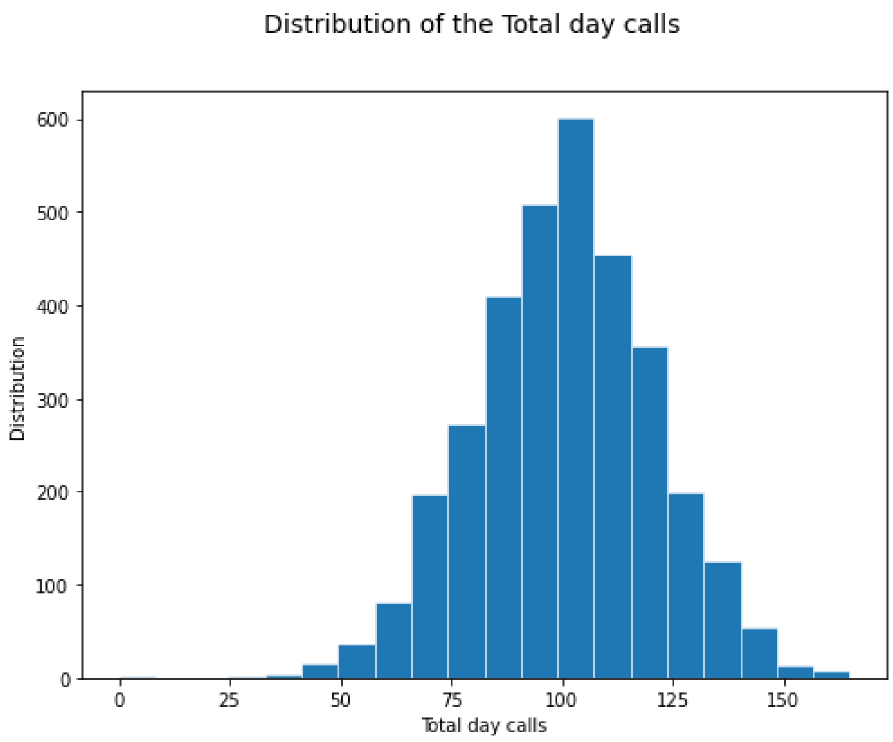
In [48]:
```
univariate_plot('total intl calls')
```



This right tailed feature shows it peaked at the 3 call mark and a sharp declined soon follows. Very few people made more than 3 calls.

Total Day Calls

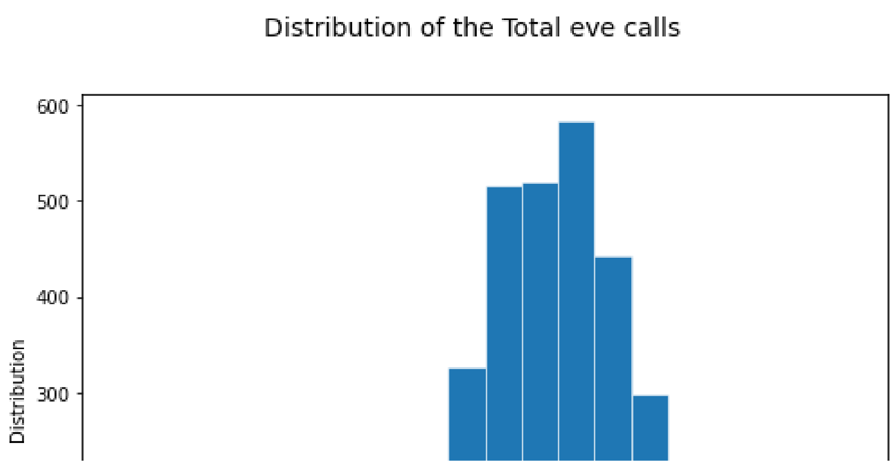
In [49]:
```
univariate_plot('total day calls')
```

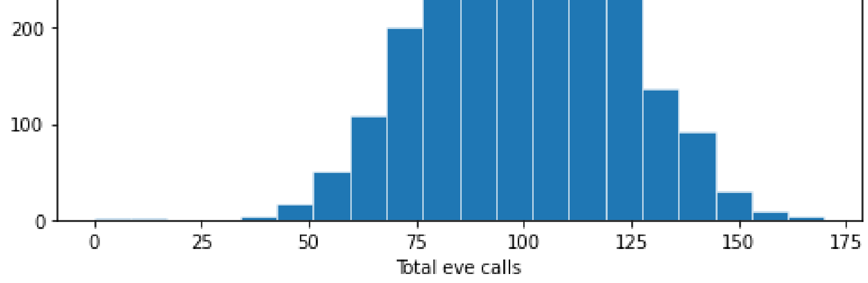


A near normal distribution of the total day calls feature peaking at around 100 day calls.

Total Evening Calls

In [50]:
```
univariate_plot('total eve calls')
```

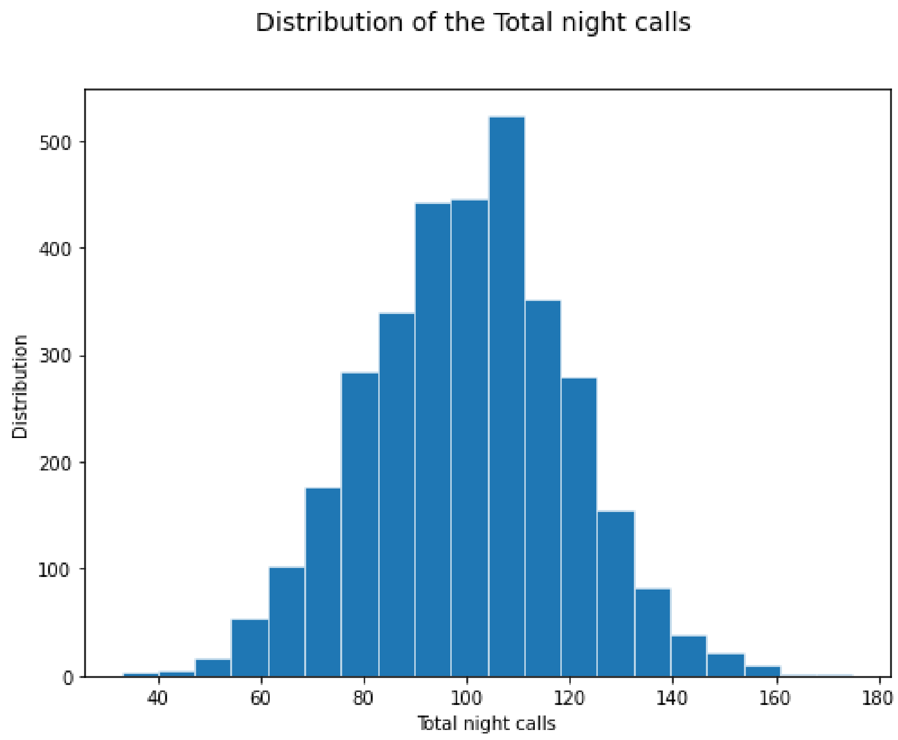




A near normal distribution of the total eve calls feature peaking at around 100 evening calls.

Total Night Calls

In [51]:
```
univariate_plot('total night calls')
```



COMMENTS

The number of people who stayed outweighed the number of people who churned.

The univariate analysis visually told us that most of our customers came from teh 410 area code.

The state with the highest number of customers was Washington

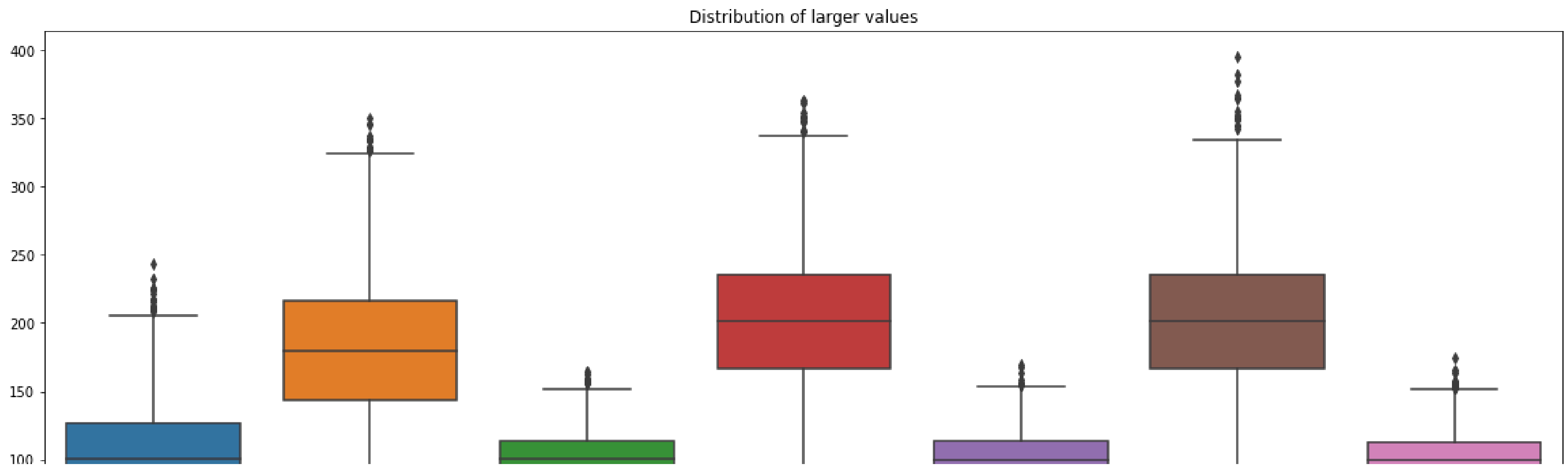
The distributions of the Account Length was normally distributed

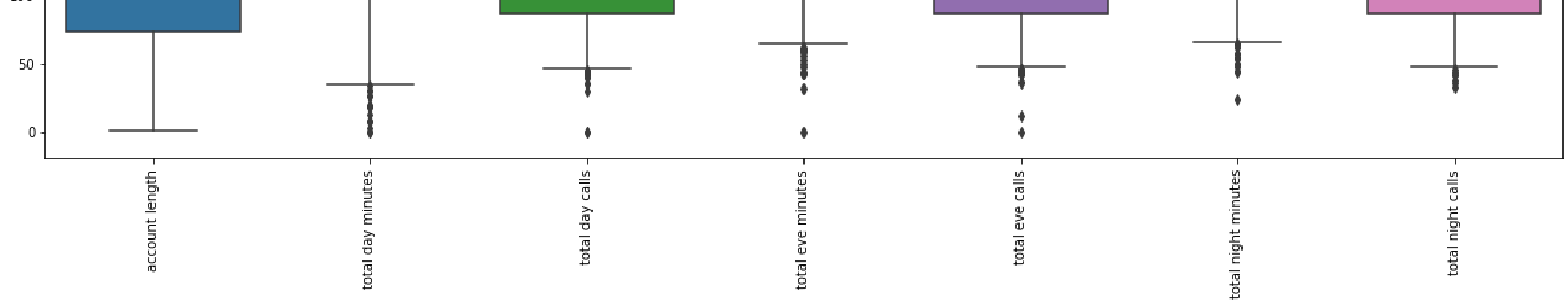
The distributions of the Customer Service calls and Total Intl Calls were right tailed.

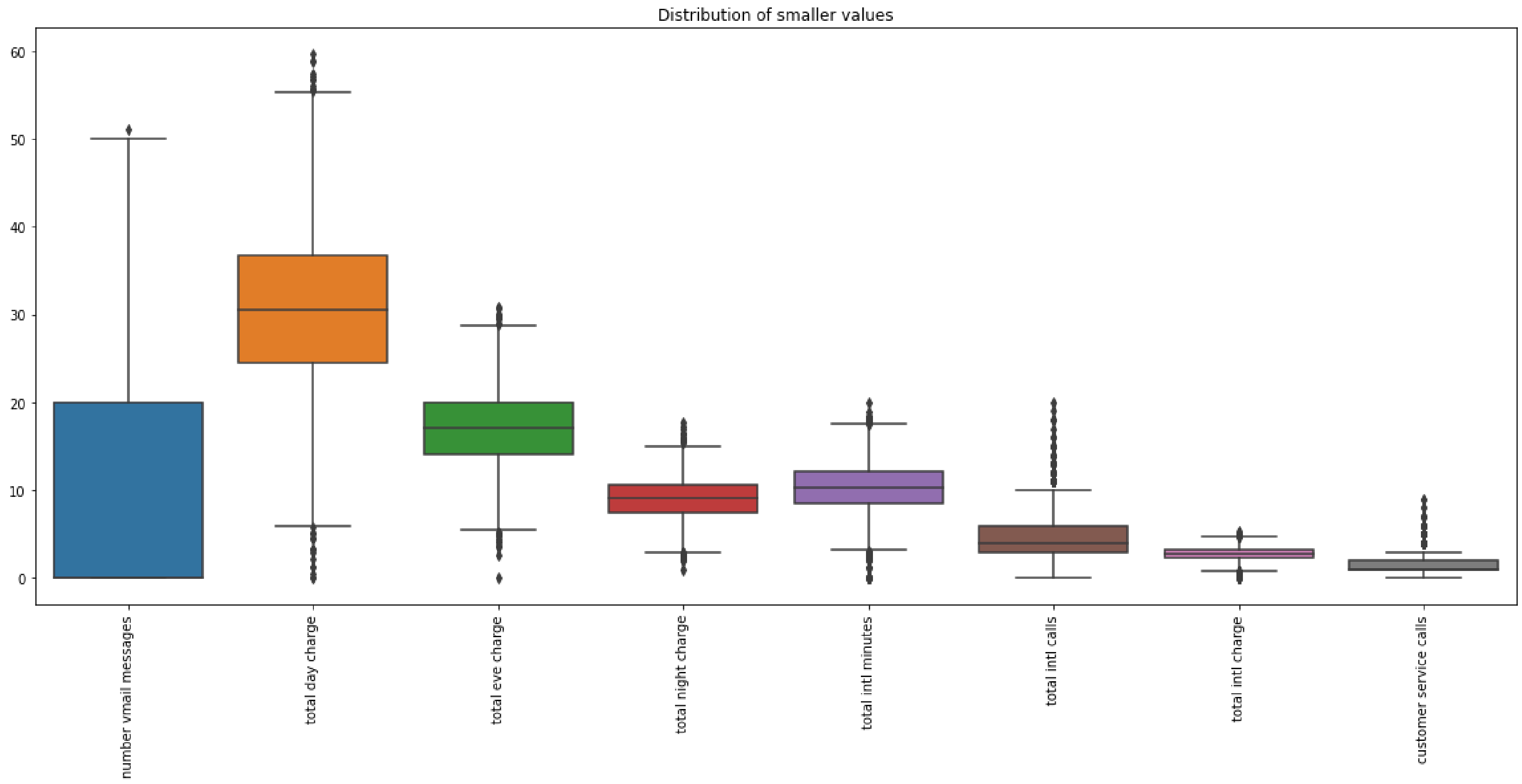
The distributions of the Total Day Calls, Total Eve Calls and Total Night were left-tailed.

Visualisation of Outliers

In [52]:
```
#Checking for outliers in the data
# List of columns for the first boxplot
large_box = ['account length','total day minutes','total day calls', 'total eve minutes',
             'total eve calls','total night minutes','total night calls']

# List of columns for the second boxplot
small_box = ['number vmail messages', 'total day charge', 'total eve charge', 'total night charge',
             'total intl minutes', 'total intl calls', 'total intl charge', 'customer service calls']

def plot_boxplots(lists, larger=True, title=None):
    fig, axes = plt.subplots(figsize=(20, 8))

    sns.boxplot(data=df[lists], ax=axes)
    axes.set_xticklabels(axes.get_xticklabels(), rotation=90)

    if title == None:
        axes.set_title('Distribution of' + (' larger values' if larger == True else ' smaller values'))

    # Show the plot
    plt.show()

plot_boxplots(large_box)
plot_boxplots(small_box, larger=False)
```





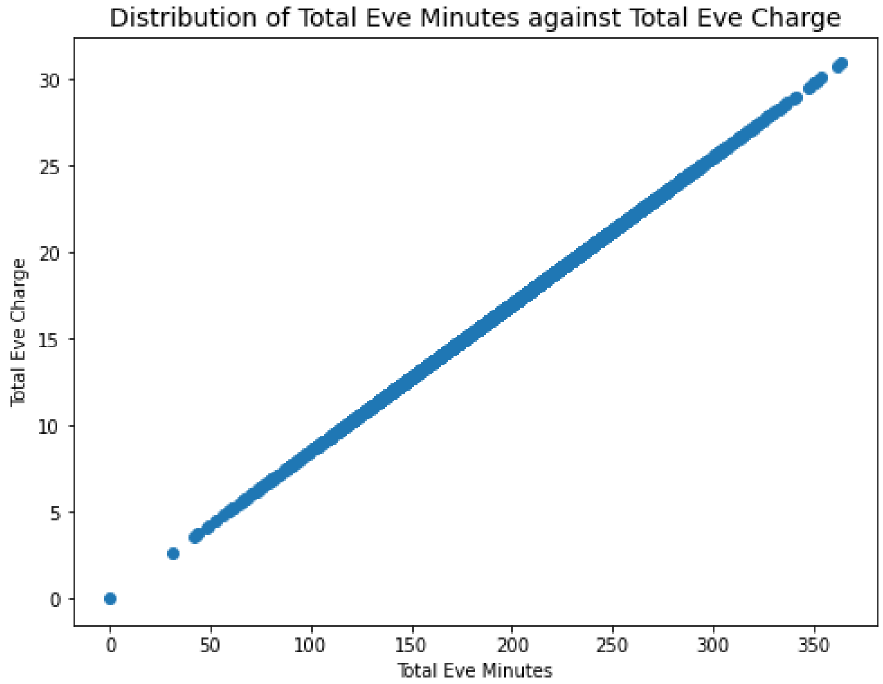


Bivariate Analysis

In [53]:
```python
def plot_scatter(first_col, second_col='churn'):
    if first_col in num_cols:
        plt.figure(figsize=(8,6))
        plt.scatter(df[first_col], df[second_col])
        plt.xlabel(f'{first_col.title()}')
        plt.ylabel(f'{second_col.title()}')
        plt.title(f'Distribution of {first_col.title()} against {second_col.title()}', fontsize=14)

def plot_graph(dataset, title, ylabel):
    plt.figure(figsize=(8,6))
    plt.bar([state for state in dataset.index],
            [rate[0] for rate in dataset.values])
    plt.title(title, fontsize=14)
    plt.ylabel(dataset.columns[0])
    plt.show()
```

Evening Minutes against Minutes

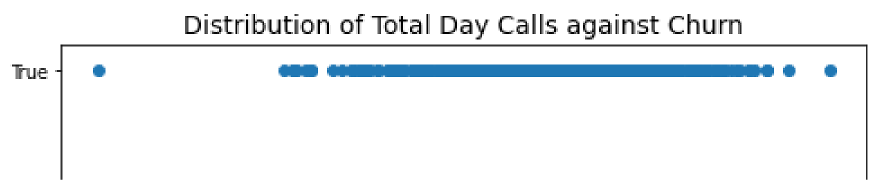
In [54]:
```python
plot_scatter('total eve minutes', 'total eve charge')
```

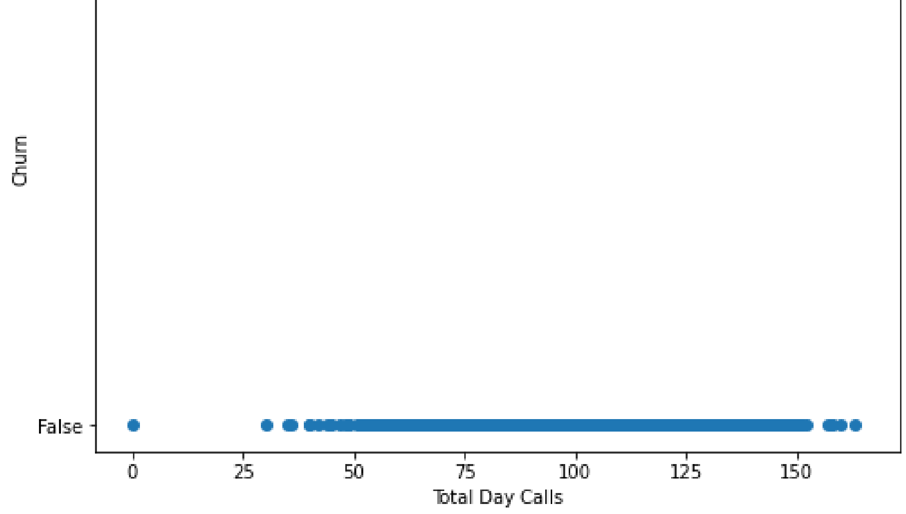


There seems to be a strong correlation between Total Evening Minutes against Total Evening Charge.

Total Day Calls against Churn

In [55]:
```python
plot_scatter('total day calls')
```



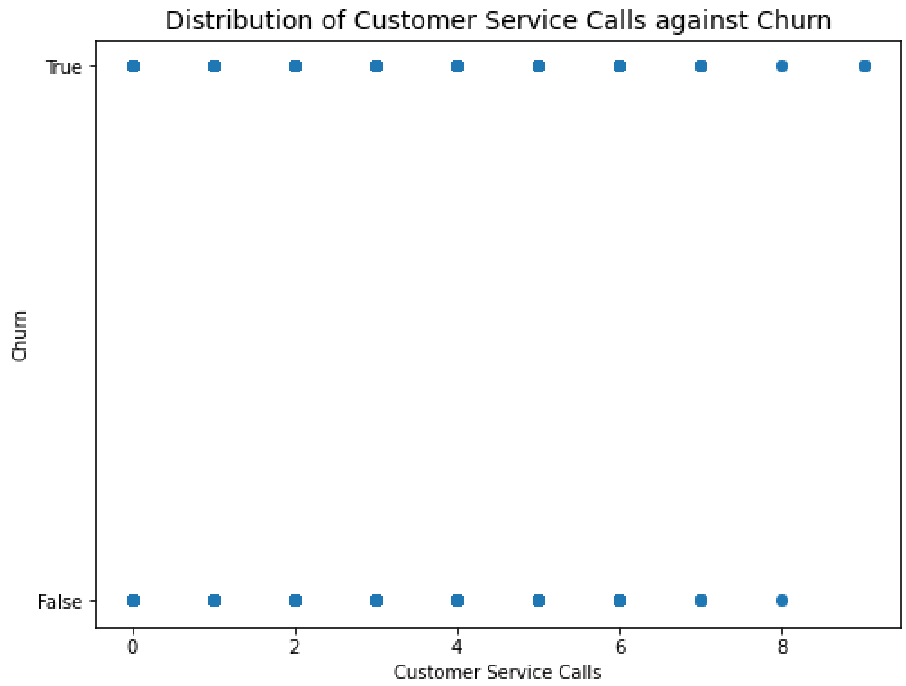


There seems to be a poor correlation between Total Day Calls against Churn.

Customer Service Calls against Churn

In [56]:

```
plot_scatter('customer service calls')
```

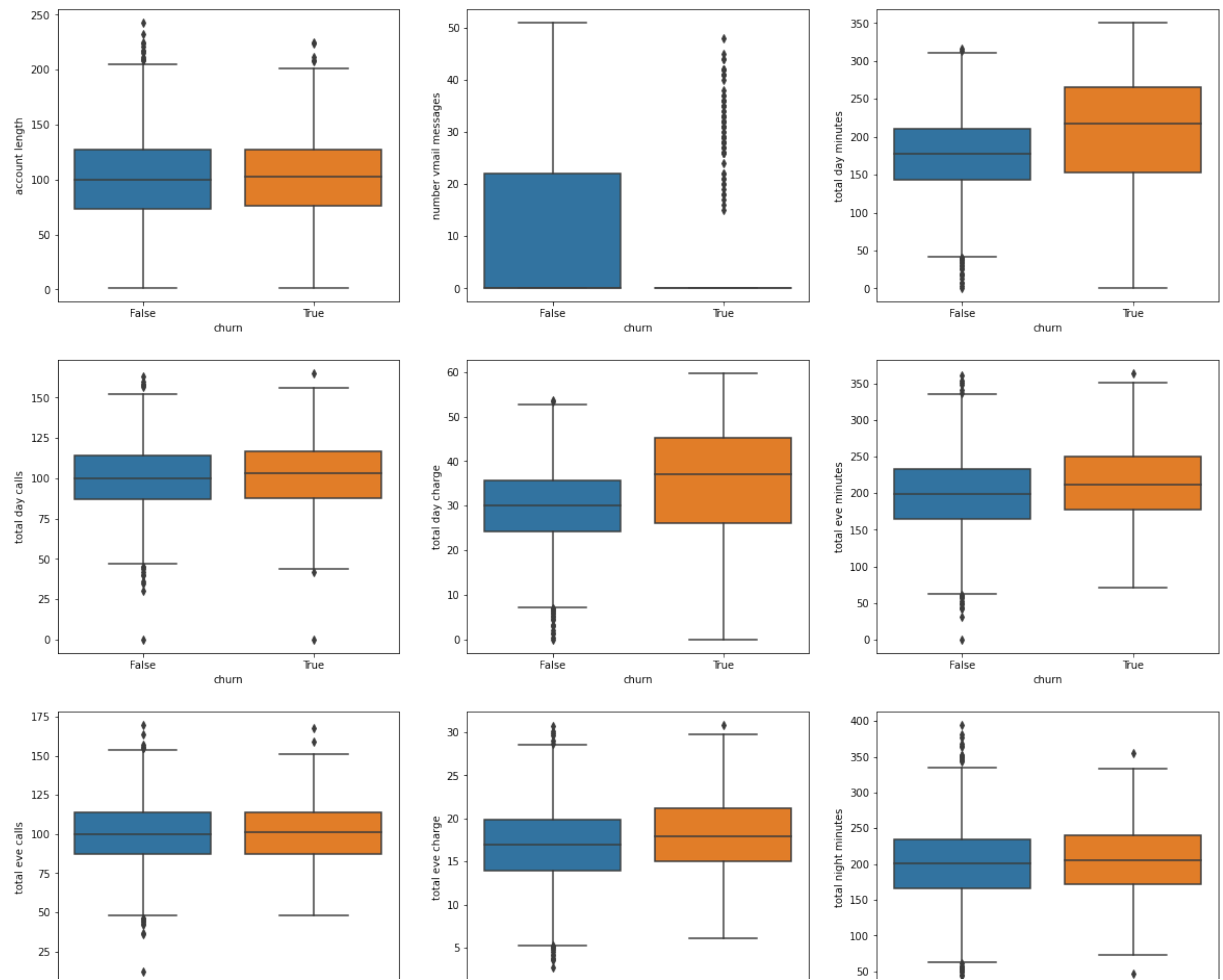


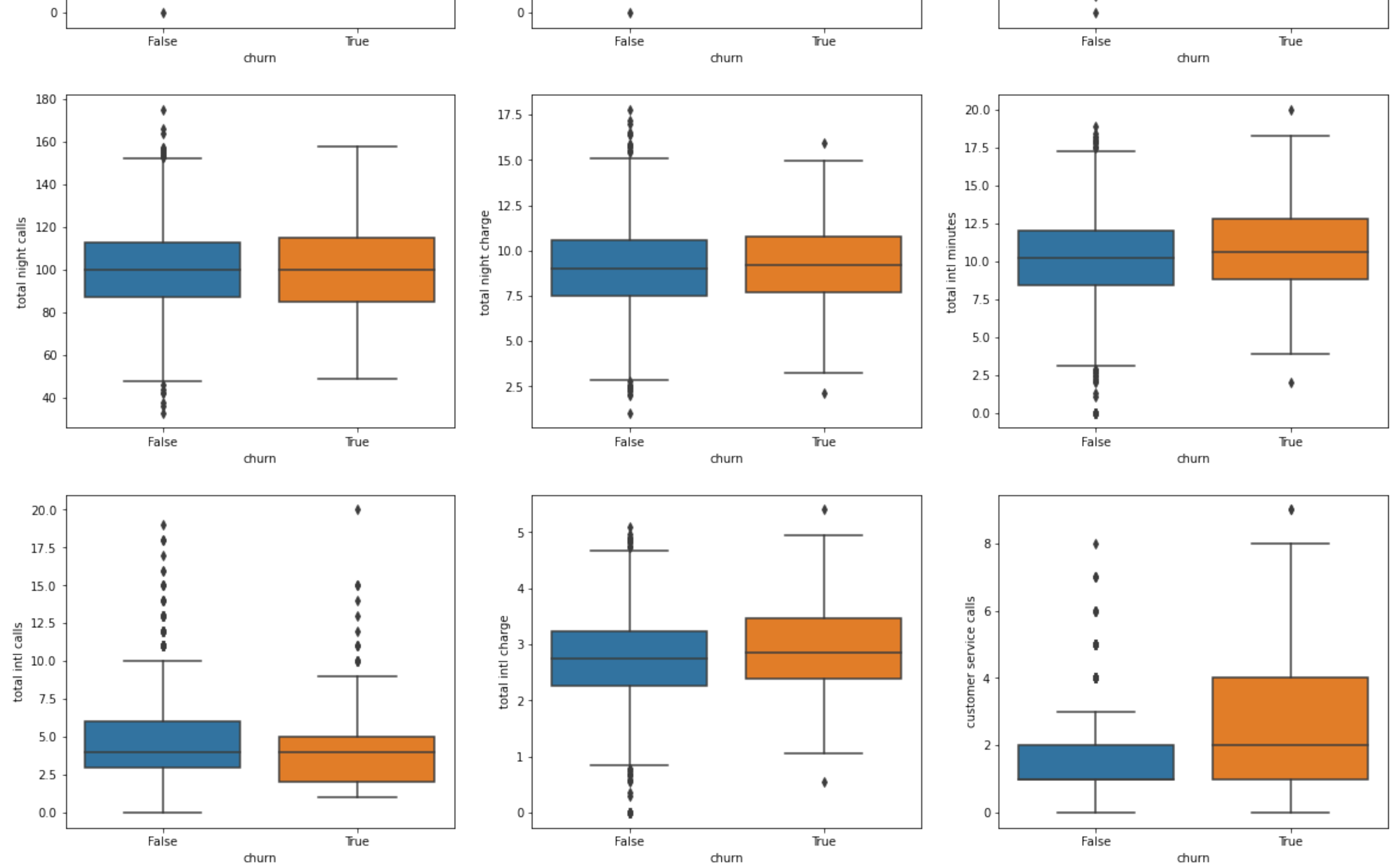
There seems to be a poor correlation between Customer Service Calls against Churn.

Bivariate Analysis of Numerical Columns against Churn Feature

In [57]:

```
fig, ax = plt.subplots(5, 3, figsize=(20, 30))
for column, plot in zip(num_cols, ax.flatten()):
    sns.boxplot(x=df['churn'], y=df[column] ,ax=plot);
```





COMMENTS

The bivariate visualisations against churn really draw out the key distinctions between customers who churned verses thoos who stayed.

Total day minutes ,total eve minutes, total day charge, total eve charge, total intl calls, total intl charge and customer service calls show a wider range of values between those who stayed and the ones who left.

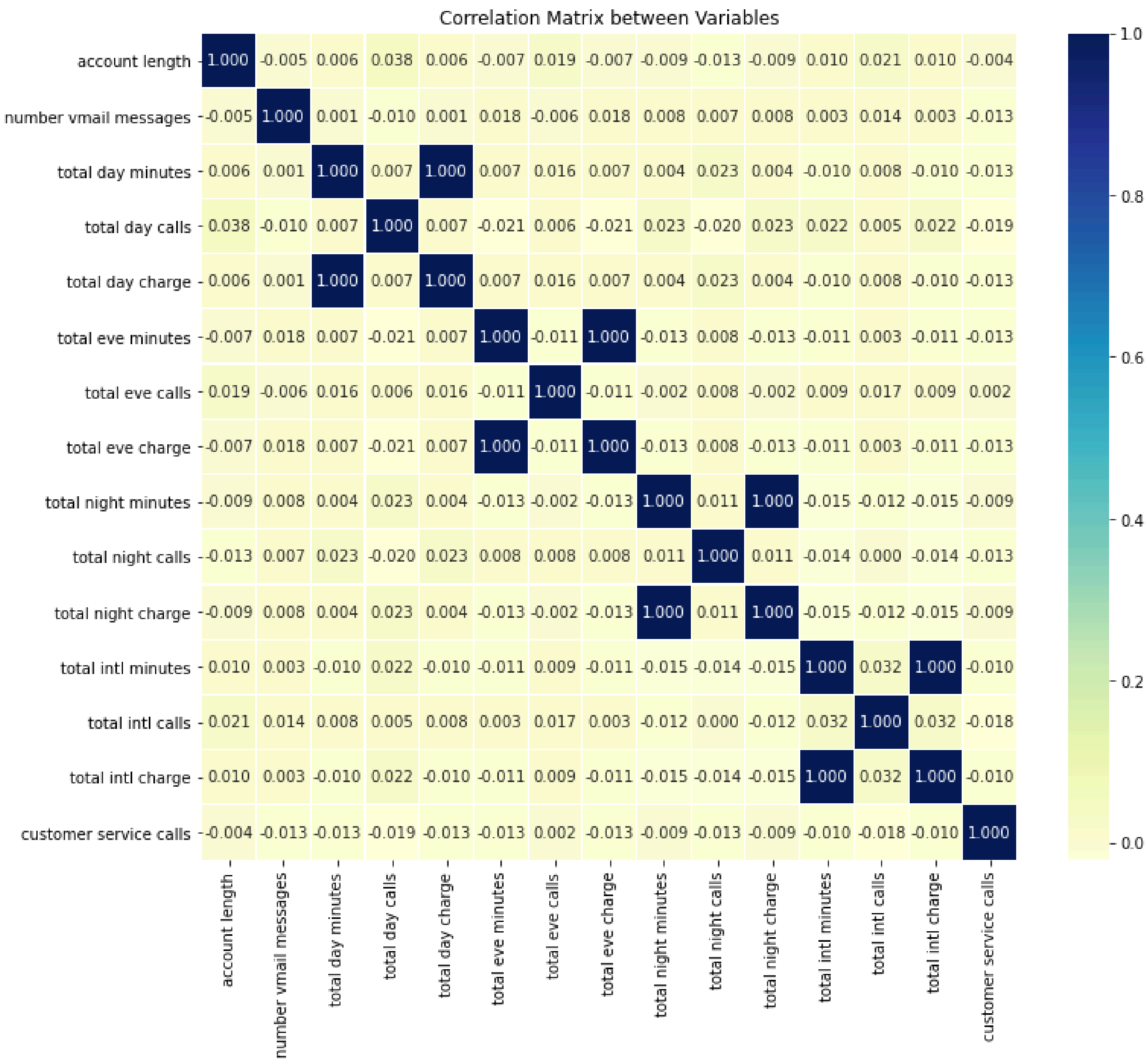
Churners made more calls to the customer service center, paid more in terms of International Call Rates even though they made less calls, paid significantly more total day charges and day minutes and had less voice mail messages. They had slightly more night calls.

Other factors such as total night charge, were nearly similar between customers who stayed against those who left.

Multivariate Analysis

In [58]:

```
# Calculate the correlation matrix
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='YlGnBu', fmt=".3f", linewidths=0.1)
plt.title('Correlation Matrix between Variables')
plt.show();
```



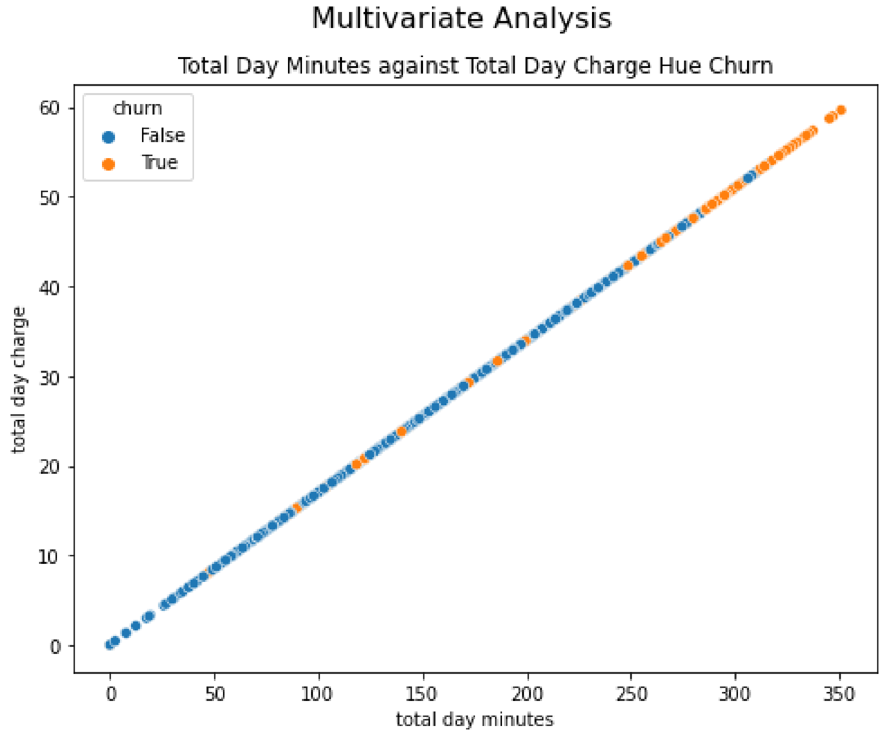
Multivariate Function

Multivariate Function

In [59]:
```
def plot_multivariate(x, y, hue='churn'):
    plt.figure(figsize=(8,6))
    plt.suptitle('Multivariate Analysis', fontsize=16)
    plt.title(f'{x.title()} against {y.title()} Hue {hue.title()}')
    sns.scatterplot(x=x, y=y, hue=hue, data=df)
    plt.show();
```

Day Minutes against Day charge

In [60]:
```
plot_multivariate(x='total day minutes', y='total day charge', hue='churn')
```



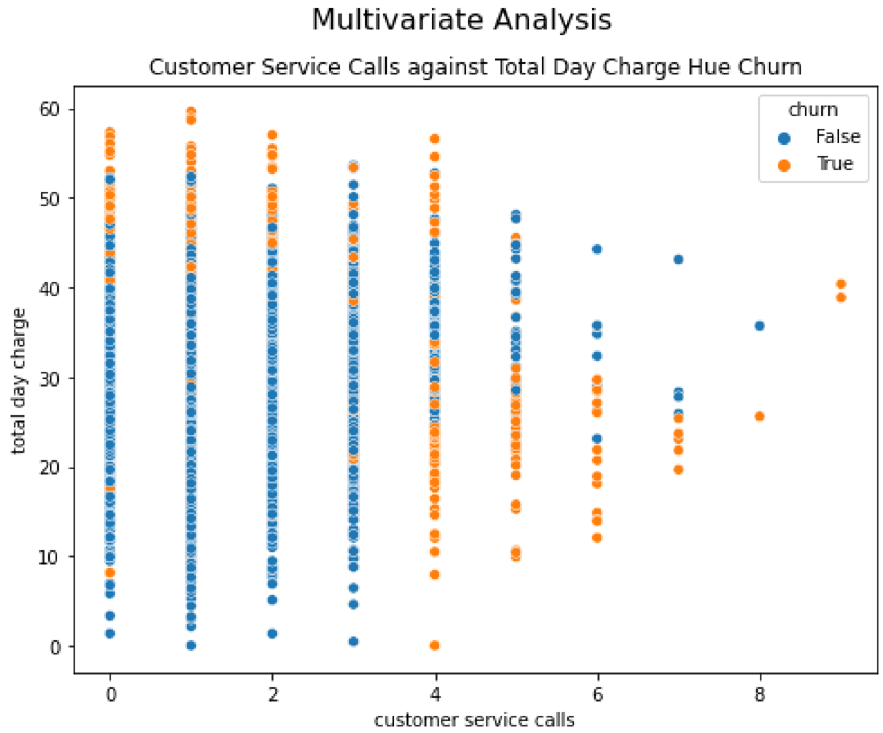
High paying customers churn more than the customers who are below the mean day charge value.

Day Charge against Customer Service Calls

In [61]:
```
df['total day charge'].mean()
```

Out[61]: 30.562307230723075

In [62]:
```
plot_multivariate(x='customer service calls', y='total day charge', hue='churn')
```



Consumers paying higher than the average churn more on low customer service call rates than those who pay less. Inversely, customers who pay less than the mean day charge churn more on higher values of the customer service call rate

7.0. Modelling

Applying the Train-Test Split

In [63]:
```
# Defining the target
y = df.churn

# Defining the predictors
X = df[df.columns[:-1]]
```

In [64]:
```
def preprocessing(test_size=0.2):

    # Imprelementing the Train Test Split
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=8)
```

```
    # Separate X data into continuous vs. categorical
    X_train_cont = X_train.select_dtypes(include='number')
    X_test_cont = X_test.select_dtypes(include='number')
    X_train_cat = X_train.select_dtypes(exclude='number')
    X_test_cat = X_test.select_dtypes(exclude='number')

    # Scale continuous values using MinMaxScaler
    scaler = StandardScaler()
    X_train_cont = scaler.fit_transform(X_train_cont)
    X_test_cont = scaler.transform(X_test_cont)

    # Dummy encode categorical values using OneHotEncoder
    ohe = OneHotEncoder(handle_unknown='ignore')
    X_train_cat = ohe.fit_transform(X_train_cat)
    X_test_cat = ohe.transform(X_test_cat)

    # Combine everything back together
    X_train_preprocessed = np.asarray(np.concatenate([X_train_cont, X_train_cat.todense()], axis=1))
    X_test_preprocessed = np.asarray(np.concatenate([X_test_cont, X_test_cat.todense()], axis=1))

    # Label Encoding the target
    # All 0's represent False and 1's represent True
    y_train = LabelEncoder().fit_transform(y_train)
    y_test = LabelEncoder().fit_transform(y_test)

    # Synthetic Minority Oversampling Technique
    smote = SMOTE()
    X_train_resampled, y_train_resampled = smote.fit_resample(X_train_preprocessed, y_train)

    return X_train_resampled, X_test_preprocessed, y_train_resampled, y_test

X_train_resampled, X_test_preprocessed, y_train_resampled, y_test = preprocessing()
```

Creating Metric Functions

In [65]:

```
scoring_keeper = []

def scoring_function(name, model, splits=0.2):
    """
    This function only requires a model.

    The function will fit it by itself and provide metrics for the SMOTE dataset and non-SMOTE dataset
    """
    X_train_resampled, X_test_preprocessed, y_train_resampled, y_test = preprocessing(test_size=splits)

    smote_model = model.fit(X_train_resampled, y_train_resampled)

    # Predict X train and X Test
    smote_train_pred = smote_model.predict(X_train_resampled)
    smote_test_pred = smote_model.predict(X_test_preprocessed)

    cross_val_scores_smote = cross_val_score(smote_model, X_train_resampled, y_train_resampled).mean()

    # Compare model scores
    print(f"""
    Metrics
        Train Accuracy Score: {accuracy_score(y_train_resampled, smote_train_pred)}
        Test Accuracy Score: {accuracy_score(y_test, smote_test_pred)}
        Train Recall Score: {recall_score(y_train_resampled, smote_train_pred)}
        Test Recall Score: {recall_score(y_test, smote_test_pred)}
    """)

    # Keep a record of the results
    scoring_keeper.append(
        {'name':name, 'model':model,
         'Accuracy': round(accuracy_score(y_test, smote_test_pred), 4),
         'CV Score': round(cross_val_scores_smote, 4),
         'Recall': round(recall_score(y_test, smote_test_pred), 4)
        }
    )
```

Classification Models

Logistic Regression Model

In [66]:

```
scoring_function(name='Logistic Regression', model=LogisticRegression(random_state=8, max_iter=200))
```

```
Metrics
    Train Accuracy Score: 0.7953549517966696
    Test Accuracy Score: 0.7766116941529235
    Train Recall Score: 0.7992988606485539
    Test Recall Score: 0.7070707070707071
```

K-Nearest Neighbors

In [67]:

```
scoring_function(name='KNN', model=KNeighborsClassifier())
```

```
Metrics
    Train Accuracy Score: 0.9211218229623137
    Test Accuracy Score: 0.7451274362818591
    Train Recall Score: 0.9982471516213848
    Test Recall Score: 0.7474747474747475
```

The K Nearest Neighbors Classification Model improved on the Logistic Regression producing an accurate prediction of 78% of churners but missed 22% who were misclassified

Decision Trees

In [69]:
```
scoring_function(name='Decision Tree', model=DecisionTreeClassifier(random_state=8))
```

```
Metrics
    Train Accuracy Score: 1.0
    Test Accuracy Score: 0.8950524737631185
    Train Recall Score: 1.0
    Test Recall Score: 0.7373737373737373
```

The Decision Tree Classification Model did not improve on the K Nearest Neighbors Classifier producing an accurate prediction of 76% of churners but missed 24% who were misclassified. It's accuracy however increased showing improved results and learning.

Random Forest Model

In [70]:
```
scoring_function(name='Random Forest', model=RandomForestClassifier(random_state=8))
```

```
Metrics
    Train Accuracy Score: 1.0
    Test Accuracy Score: 0.9385307346326837
    Train Recall Score: 1.0
    Test Recall Score: 0.7474747474747475
```

The Random Forest Classification Model did not improve on the Decision Tree Classifier producing an accurate prediction of 76% of churners but missed 24% who were misclassified. It's test accuracy however increased showing improved results and learning.

Evaluation of Models and their Scores

In [71]:
```
for item in scoring_keeper:
    print('\t', item['name'])
    print('\tRecall Score',item['Recall'])
    print('\tCV Score',item['CV Score'])
    print("")
```

```
     Logistic Regression
    Recall Score 0.7071
    CV Score 0.7857

     KNN
    Recall Score 0.7475
    CV Score 0.8834

     Decision Tree
    Recall Score 0.7374
    CV Score 0.9299

     Random Forest
    Recall Score 0.7475
    CV Score 0.9691
```

Train Test Split Selection

In [72]:
```
def test_split(range_of_values, models_list, names_of_models):
    record = []
    for split in range_of_values:
        X_train, X_test, y_train, y_test = preprocessing(test_size=split)
        for item,model in enumerate(models_list):
            model.fit(X_train, y_train)
            train_score = cross_val_score(model, X_train, y_train).mean()
            test_score = cross_val_score(model, X_test, y_test).mean()
            record.append({'split':split, 'name':names_of_models[item],
                          'train_score':train_score,
                           'test_score':test_score})
    return record
```

In [73]:
```
# Obtain the names and the models from the score keeping list we populated
names_list = []
model_list = []
for model in scoring_keeper:
    names_list.append(model['name'])
    model_list.append(model['model'])

range_values = [0.5, 0.4, 0.3, 0.2, 0.1]

results = test_split(range_values, model_list, names_list)
results
```

Out[73]:
```
[{'split': 0.5,
  'name': 'Logistic Regression',
  'train_score': 0.8070995590910492,
  'test_score': 0.8554260847674022},
 {'split': 0.5,
  'name': 'KNN',
  'train_score': 0.8843992230135973,
  'test_score': 0.8818345291399183},
 {'split': 0.5,
  'name': 'Decision Tree',
  'train_score': 0.927631116455462,
  'test_score': 0.892632752512992},
 {'split': 0.5,
  'name': 'Random Forest',
  'train_score': 0.9652335584127278,
```

```
  'test_score': 0.9184082285878693},
 {'split': 0.4,
  'name': 'Logistic Regression',
  'train_score': 0.7828327672893864,
  'test_score': 0.853079327532314},
 {'split': 0.4,
  'name': 'KNN',
  'train_score': 0.8810114081828058,
  'test_score': 0.8830587705218103},
 {'split': 0.4,
  'name': 'Decision Tree',
  'train_score': 0.9276209409067293,
  'test_score': 0.9017909943397819},
 {'split': 0.4,
  'name': 'Random Forest',
  'train_score': 0.9665993997501106,
  'test_score': 0.9197938666891948},
 {'split': 0.3,
  'name': 'Logistic Regression',
  'train_score': 0.777139549436796,
  'test_score': 0.8480000000000001},
 {'split': 0.3,
  'name': 'KNN',
  'train_score': 0.8741886733416772,
  'test_score': 0.869},
 {'split': 0.3,
  'name': 'Decision Tree',
  'train_score': 0.9344721526908636,
  'test_score': 0.8939999999999999},
 {'split': 0.3,
  'name': 'Random Forest',
  'train_score': 0.9694871714643304,
  'test_score': 0.9},
 {'split': 0.2,
  'name': 'Logistic Regression',
  'train_score': 0.7808953517418958,
  'test_score': 0.8560543148917068},
 {'split': 0.2,
  'name': 'KNN',
  'train_score': 0.8762057800580312,
  'test_score': 0.8710582426214792},
 {'split': 0.2,
  'name': 'Decision Tree',
  'train_score': 0.9382145808112835,
  'test_score': 0.8635618897991245},
 {'split': 0.2,
  'name': 'Random Forest',
  'train_score': 0.9684503564497223,
  'test_score': 0.9010548759959599},
 {'split': 0.1,
  'name': 'Logistic Regression',
  'train_score': 0.7874099653731118,
  'test_score': 0.808457711442786},
 {'split': 0.1,
  'name': 'KNN',
  'train_score': 0.8845716927547718,
  'test_score': 0.8623699683401176},
 {'split': 0.1,
  'name': 'Decision Tree',
  'train_score': 0.9345236679970185,
  'test_score': 0.8714156490275894},
 {'split': 0.1,
  'name': 'Random Forest',
  'train_score': 0.9702778642663723,
  'test_score': 0.8833559475350519}]
```

In [74]:

```
# Best model based on training score
max_model_train = max(results, key=lambda x:x['train_score'])

# Best model based on test score
max_model_test = max(results, key=lambda x:x['test_score'])

# View models
max_model_train,max_model_test
```

Out[74]:

```
({'split': 0.1,
  'name': 'Random Forest',
  'train_score': 0.9702778642663723,
  'test_score': 0.8833559475350519},
 {'split': 0.4,
  'name': 'Random Forest',
  'train_score': 0.9665993997501106,
  'test_score': 0.9197938666891948})
```

Hyperparameter Tuning

GridSearchCV

In [75]:

```
# Variables that will populate our Grid Search Params
ranges = list(np.arange(80,120,10))
criterion = ["gini", "entropy"]
min_samples_split=list(np.arange(2, 5))
min_samples_leaf=list(np.arange(1, 5))

# The grid search param_dict
param_grid = dict(n_estimators=ranges, criterion=criterion,
                  min_samples_split= min_samples_split,
                  min_samples_leaf = min_samples_leaf )

print(param_grid)
```

```
{'n_estimators': [80, 90, 100, 110], 'criterion': ['gini', 'entropy'], 'min_samples_split': [2, 3, 4], 'min_samples_leaf': [1, 2, 3, 4]}
```

In [82]:
```
grid = GridSearchCV(
    RandomForestClassifier(),
    param_grid, cv=10,
    scoring='accuracy',
    return_train_score=False)
grid.fit(X_train_resampled, y_train_resampled)
```

Out[82]:
```
GridSearchCV(cv=10, estimator=RandomForestClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'min_samples_leaf': [1, 2, 3, 4],
                         'min_samples_split': [2, 3, 4],
                         'n_estimators': [80, 90, 100, 110]},
             scoring='accuracy')
```

In [83]:
```
# Train Data Score
print(grid.score(X_train_resampled, y_train_resampled))

# Test Data Score
grid.score(X_test_preprocessed, y_test)
```

```
1.0
```

Out[83]:
```
0.9400299850074962
```

In [84]:
```
# Saving the best parameters of the RandomForest in a variable
rf_parameters = grid.best_params_
rf_parameters
```

Out[84]:
```
{'criterion': 'entropy',
 'min_samples_leaf': 1,
 'min_samples_split': 3,
 'n_estimators': 80}
```

In [85]:
```
# Clear the scoring list to make room for the new models
scoring_keeper.clear()
```

Hyperparameter Tuning and Train Test Split

In [86]:
```
# RandomForest with the best scores with the Train Test Split and best parameters from GridSearchCV
scoring_function(name='RandomForest TTS 0.1', splits=0.1,
                 model=RandomForestClassifier(
                     criterion=rf_parameters['criterion'],
                     min_samples_leaf=rf_parameters['min_samples_leaf'],
                     min_samples_split=rf_parameters['min_samples_split'],
                     n_estimators=rf_parameters['n_estimators'],
                 )
                )
```

```
Metrics
    Train Accuracy Score: 1.0
    Test Accuracy Score: 0.9550898203592815
    Train Recall Score: 1.0
    Test Recall Score: 0.8245614035087719
```

COMMENTS

After hyperparameter tuning and picking the right train-test split value, the RandomForest gave us the metrics we required as per the objectives we set out to achieve.

Random Forest procuced an accuracy score of 100% and a recall score of 80.7%.

This high recall score means that the model would be able to predict correctly the customers who would churn. It will however miss very few who might be misclassified as non-churners.

However, corrective action and a deep focus on retaining the 80% would prove advantageous to Syria Telecommunications.

8.0. Conclusions and Recommendations

Based on the findings from our churn model, the following recommendations are proposed:

Targeted Retention Programs: Offering rewards and discounts to customers can help lower the churn rate. Long-term customers or those who have used a certain amount of talk-time should be prioritized to foster loyalty. This approach could also attract new customers and enhance the company’s visibility.

Training for Customer Care Agents: As representatives of the organization, customer care agents should be trained to follow specific guidelines that make customers feel valued and understood. Proper training will help ensure even dissatisfied customers are listened to and their concerns are addressed.

Service Improvements: Syria Telecommunications should enhance its services by reviewing pricing strategies. Reducing call costs or offering fixed rates beyond a certain usage threshold would be beneficial. Bundling services, such as international plans or voicemail, with a specific amount of minutes could also be attractive. Additionally, addressing technical issues in problem-prone areas promptly would reduce customer complaints.

Ongoing Customer Feedback Collection: Customer feedback is a valuable resource that Syria Telecommunications should leverage. The company should actively encourage feedback from both current and exiting customers to identify areas where users feel neglected or dissatisfied.