Notebook

June 7, 2024

0.0.1 Dataset of Choice:

I chose the `SyriaTel Customer Churn Dataset` primarily because it falls within the telecommunications industry. Telecom companies, including industry giants like Safaricom, are among the most prominent enterprises in Africa. Therefore, targeting them as potential future clients or employers, it makes sense to gain familiarity with handling datasets specific to their industry.

The subject dataset was downloaded from Kaggle

0.0.2 Business Understanding

`Project Overview:` This project aims to analyse and predict the customer churn rate in SyriaTel, to assist the Customer Service and Sales & Marketing Teams in devicing techniques to reduce churn. `Objectives:` 1. Analyze customer churn data to identify key factors contributing to customer churn. 2. Build a predictive model to forecast customer churn. 3. Provide actionable insights and recommendations to the Customer Service and Sales & Marketing teams..

Key Business Questions

1. What is the overall churn rate?
2. What are the most significant factors contributing to customer churn?
3. How can the identified factors be addressed to reduce churn?
4. Which ML model would be most suitable to predict churn?

Our Stakeholders

1. `The Sales & Marketing Team:` Interested in identifying customers at risk of churning to implement targeted retention campaigns.
2. `The Customer Service Team:` Needs to understand common issues leading to churn to improve service quality.

0.0.3 Data Understanding

Let's take a look at our data and try to familiarize ourselves with it.

[1]:
```python
# Import the necessary libaries

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

```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from imblearn.over_sampling import SMOTE
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
↪f1_score, roc_auc_score, confusion_matrix
```

[2]:
```
#load and Inspect the first 5 rows of our dataset.
data = pd.read_csv('SyriaTel.csv')
data.head()
```

[2]:
```
  state  account length  area code phone number international plan  \
0    KS             128        415     382-4657                 no
1    OH             107        415     371-7191                 no
2    NJ             137        415     358-1921                 no
3    OH              84        408     375-9999                yes
4    OK              75        415     330-6626                yes

  voice mail plan  number vmail messages  total day minutes  total day calls  \
0             yes                     25              265.1              110
1             yes                     26              161.6              123
2              no                      0              243.4              114
3              no                      0              299.4               71
4              no                      0              166.7              113

   total day charge  …  total eve calls  total eve charge  \
0             45.07  …               99             16.78
1             27.47  …              103             16.62
2             41.38  …              110             10.30
3             50.90  …               88              5.26
4             28.34  …              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
```

```
0                 10.0                     3                2.70
1                 13.7                     3                3.70
2                 12.2                     5                3.29
3                  6.6                     7                1.78
4                 10.1                     3                2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]
```

[3]:
```python
#let's see how big our raw dataset is
data.shape
```

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

Column Breakdown Here we see the a breakdown of the 21 columns in our dataset and what they represent:

`state:`The state where the customer resides. `account length:` The number of days the account has been active. `area code:` The area code of the customer's phone number. `phone number:` The customer's phone number. `international plan:` Whether the customer has an international plan (yes/no). `voice mail plan:` Whether the customer has a voice mail plan (yes/no). `number vmail messages:` The number of voice mail messages. `total day minutes:` Total minutes of calls during the day. `total day calls:` Total number of calls during the day. `total day charge:` Total charge for calls during the day. `total eve minutes:` Total minutes of calls during the evening. `total eve calls:` Total number of calls during the evening. `total eve charge:` Total charge for calls during the evening. `total night minutes:` Total minutes of calls during the night. `total night calls:` Total number of calls during the night. `total night charge:` Total charge for calls during the night. `total intl minutes:` Total minutes of international calls. `total intl calls:` Total number of international calls. `total intl charge:` Total charge for international calls. `customer service calls:` Number of calls to customer service. `churn:` Whether the customer has churned (True/False).

[4]:
```python
#Checking for any missing values in the dataset
data.isna().sum()
```

[4]:
```
state                    0
account length           0
area code                0
phone number             0
international plan       0
voice mail plan          0
number vmail messages    0
```

```
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

[5]:
```
#Checking for any duplicates in the dataset
data.duplicated().sum()
```

[5]: 0

[6]:
```
# Checking for summary infor on the dataset
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
```

Comments: We have a dataset of 21 columns and 3,333 rows(entries). The columns description has been provided above for reference. Our dataset is notably clean, it neither has duplicated values nor does it have missing values. Out of the 21 columns, 16 are numeric(floats and integers) and our Target (Churn column) is of boolean type(Yes/No) hence indicating our task is a classification problem.

Summary Statistics

[7]:
```
data.describe()
```

[7]:
```
       account length    area code  number vmail messages  total day minutes  \
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000

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

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

       total night charge  total intl minutes  total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
```

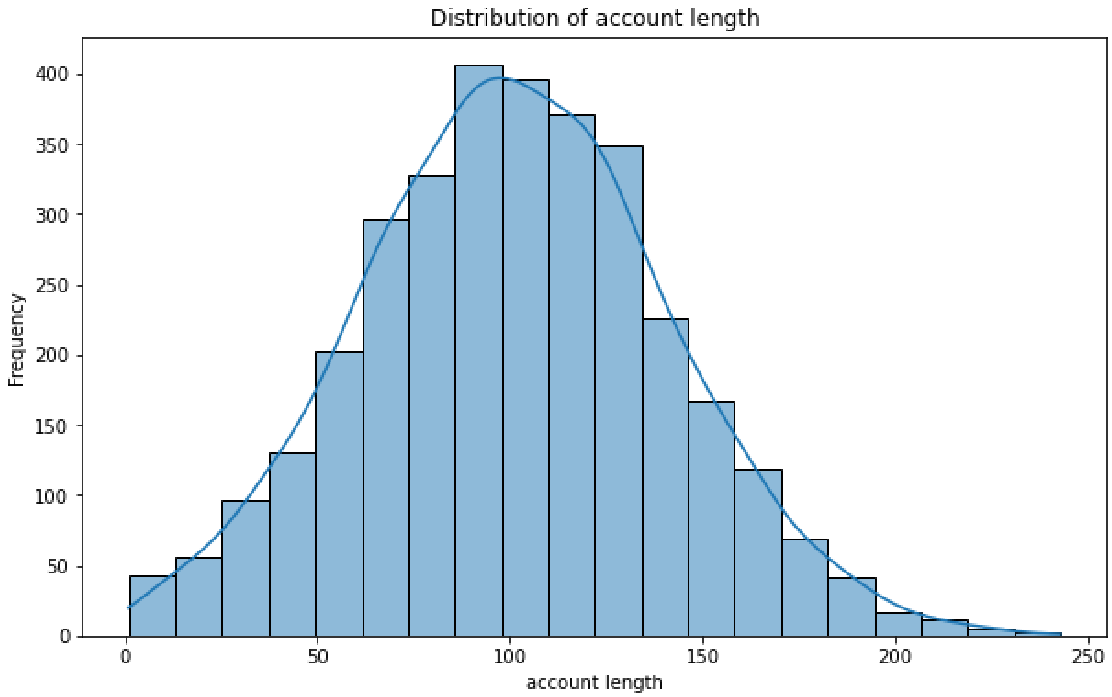
```
std        2.275873       2.791840       2.461214
min        1.040000       0.000000       0.000000
25%        7.520000       8.500000       3.000000
50%        9.050000      10.300000       4.000000
75%       10.590000      12.100000       6.000000
max       17.770000      20.000000      20.000000

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

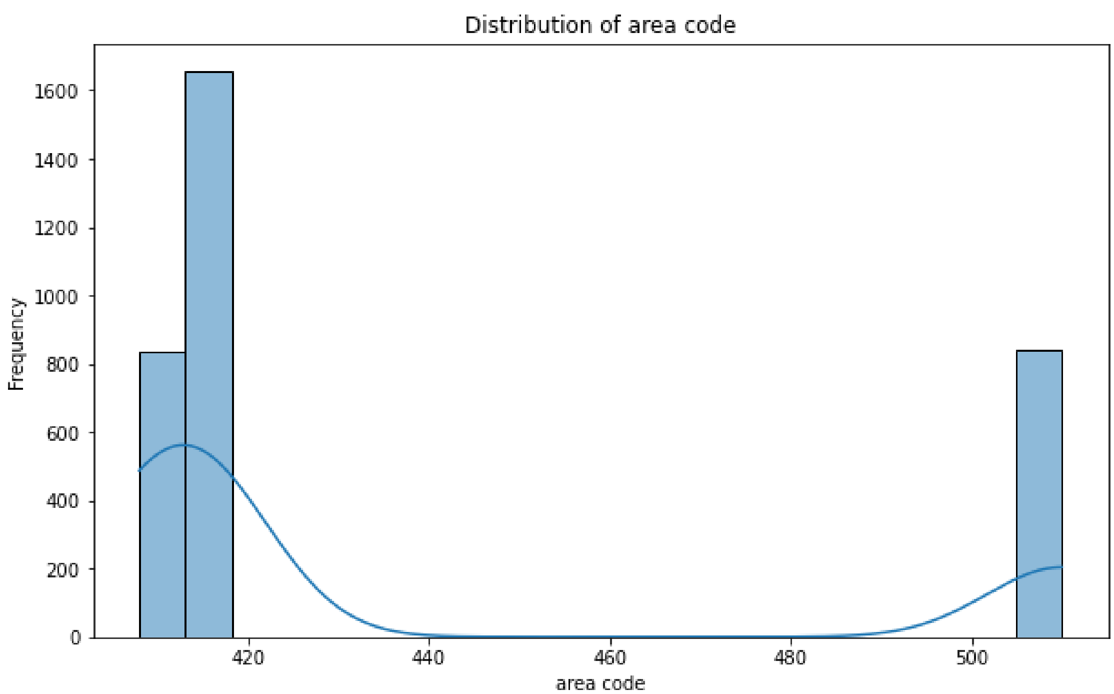
Comments: There are features with many zero values (e.g., voice mail messages, international minutes), suggesting that many customers do not use these services. This could be an important factor in understanding customer behavior and churn.

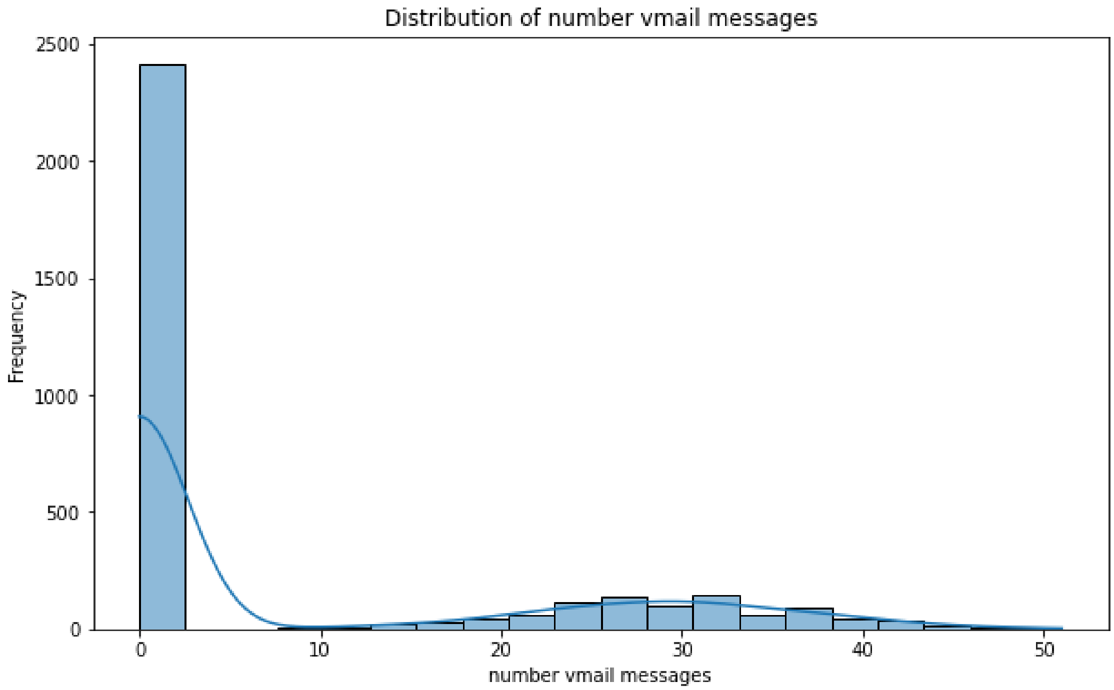
0.0.4 Univariate Analysis

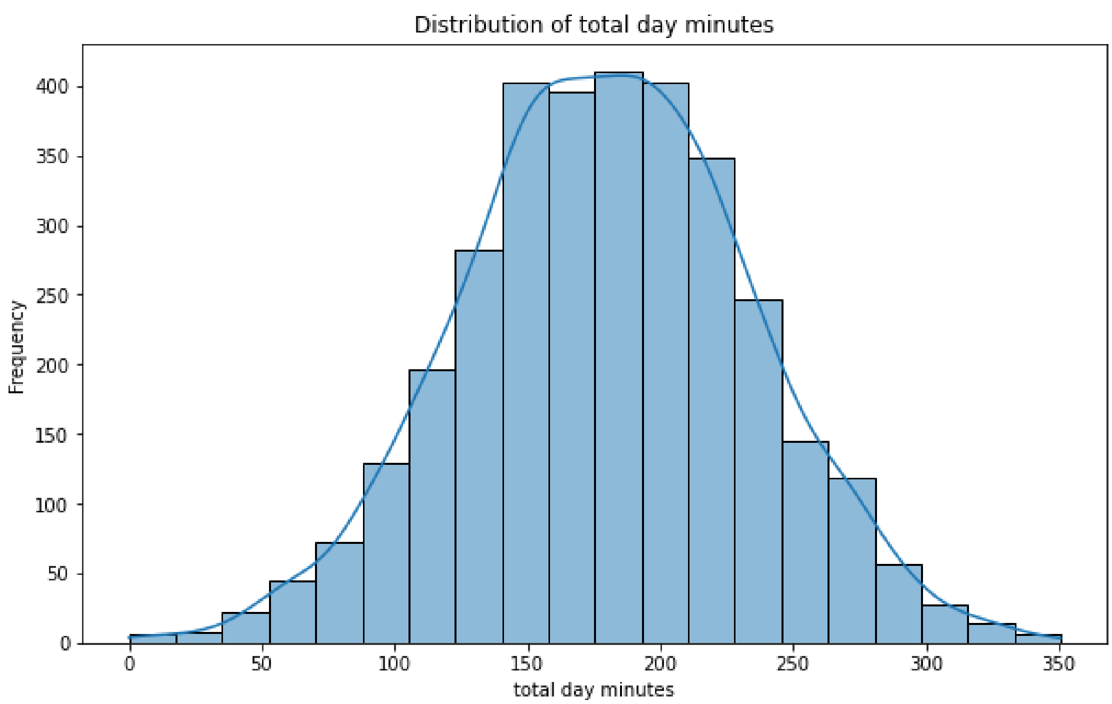
Let's further explore the distribution of these individual features

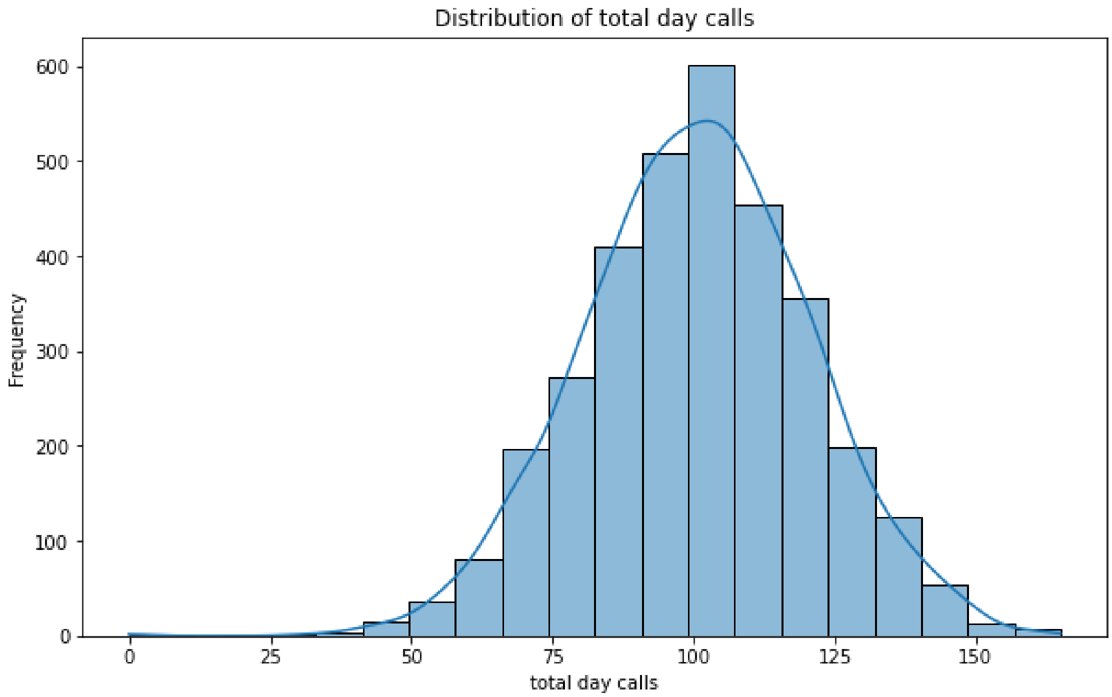
[8]:
```python
# Univariate Analysis For all features except the phone no.
# Univariate Analysis for Numerical Features
numerical_features = data.select_dtypes(include = ['int64', 'float64'])

# Histograms for numerical features
for feature in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.histplot(data[feature], bins=20, kde=True)
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Frequency')
    plt.show()
```


Distribution of account length
Frequency
account length
0
50
100
150
200
250
300
350
400
0
50
100
150
200
250

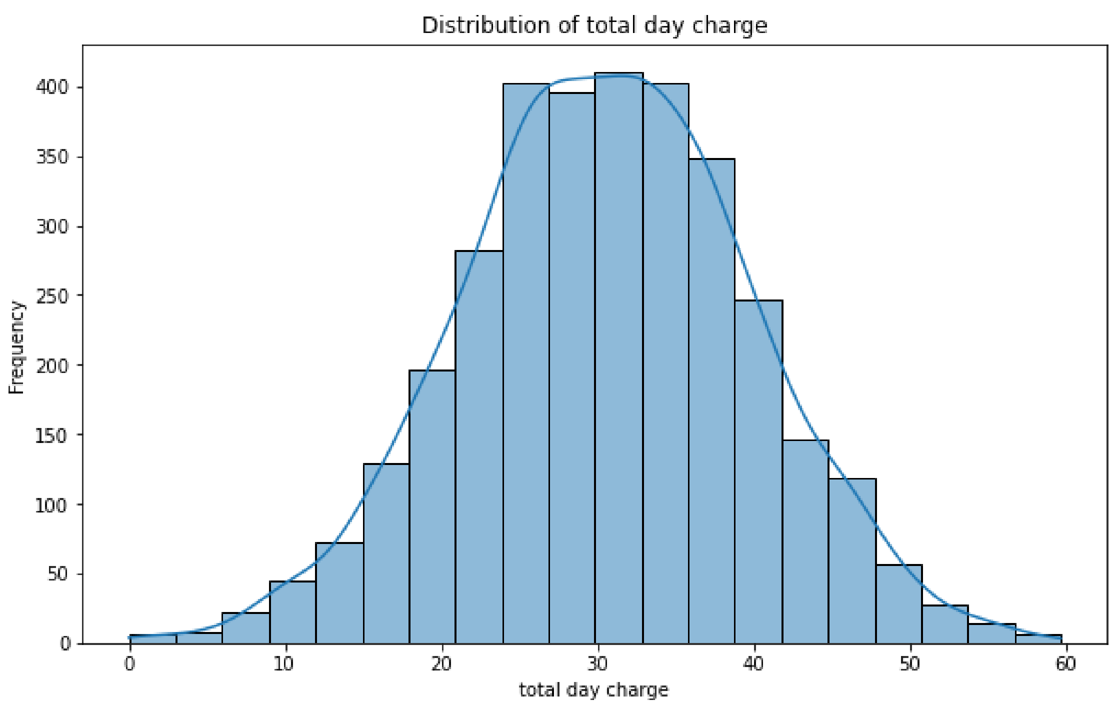

Distribution of area code
Frequency
area code
0
200
400
600
800
1000
1200
1400
1600
420
440
460
480
500


Distribution of number vmail messages
Frequency
2500
2000
1500
1000
500
0
0
10
20
30
40
50
number vmail messages

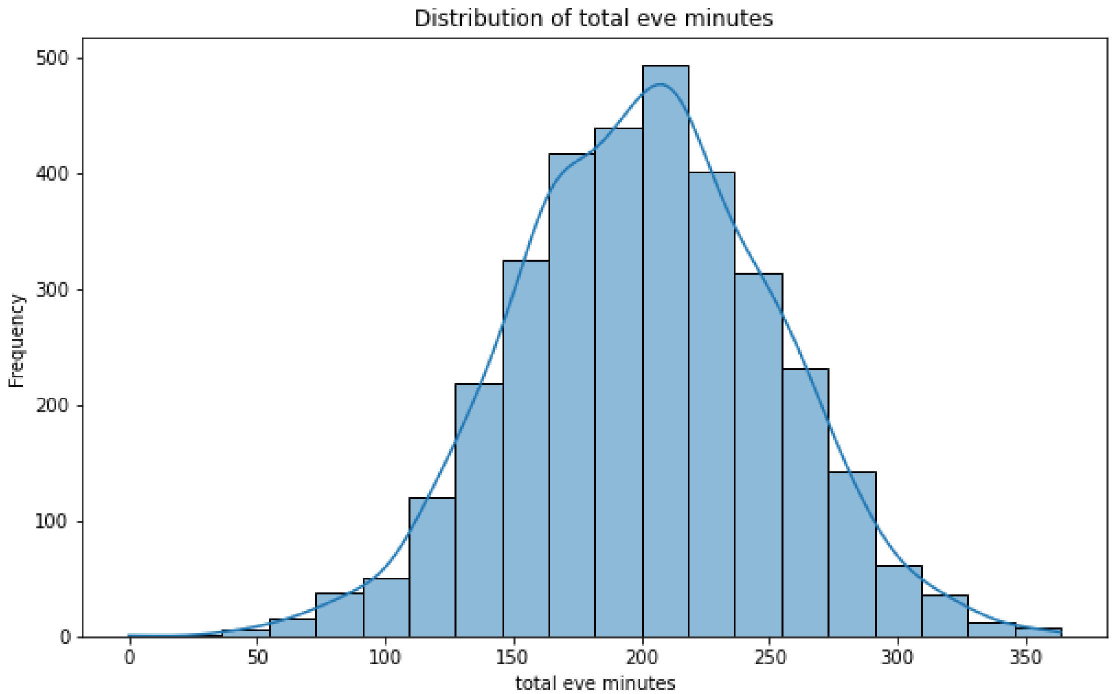

Distribution of total day minutes
Frequency
400
350
300
250
200
150
100
50
0
0
50
100
150
200
250
300
350
total day minutes

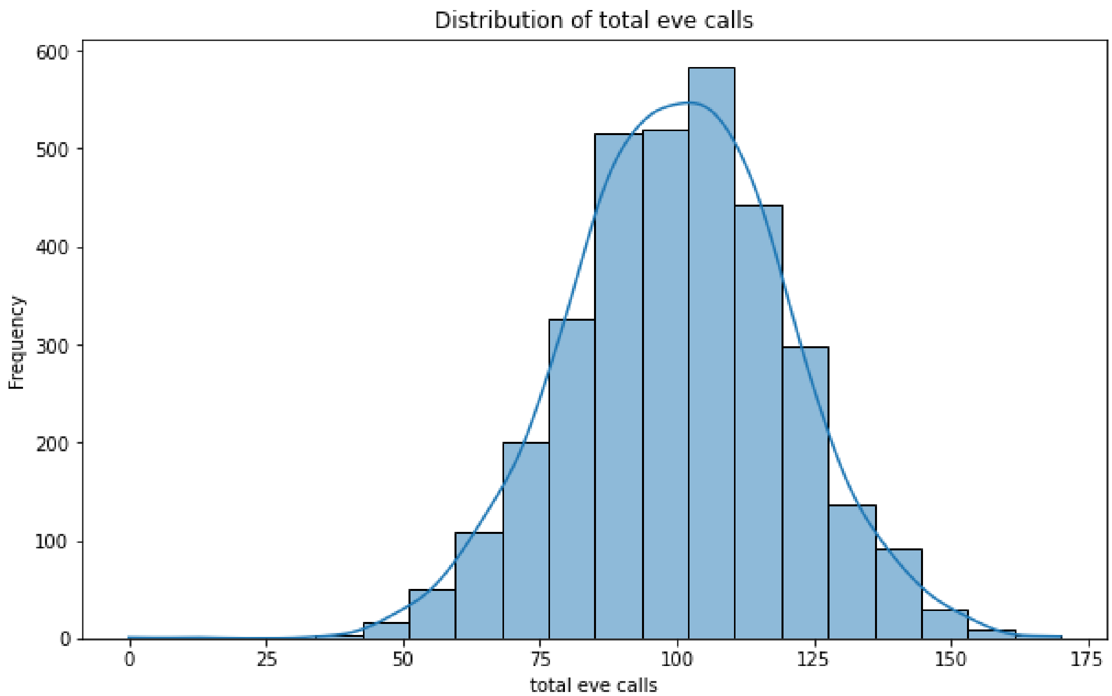


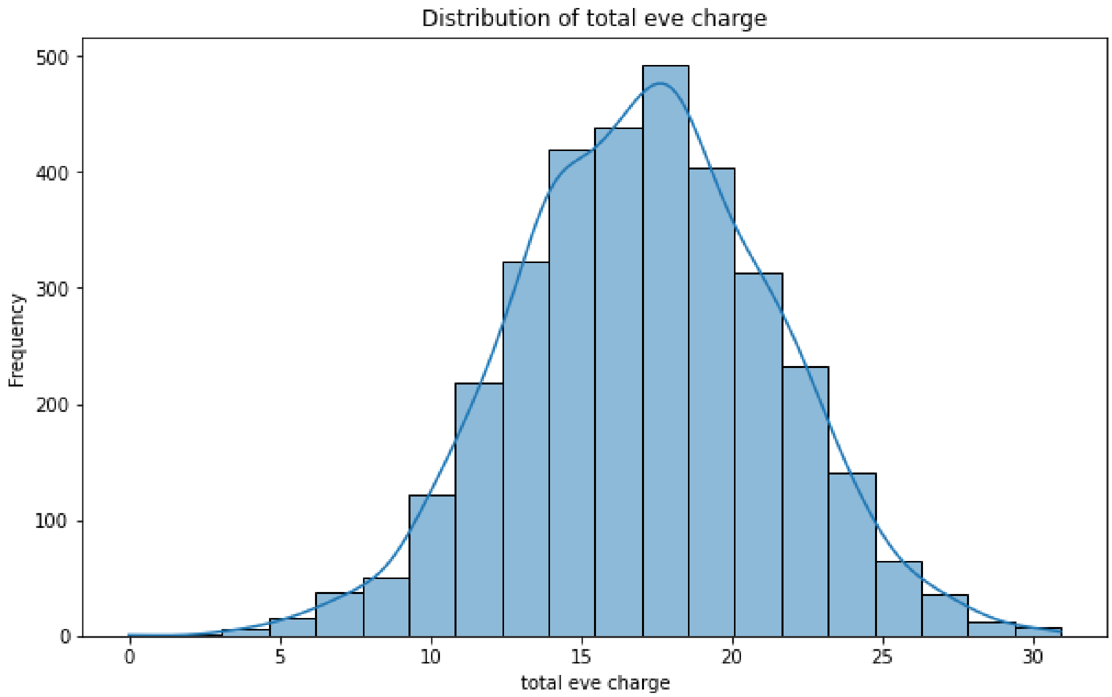
Distribution of total day calls
Frequency
total day calls
0
100
200
300
400
500
600
0
25
50
75
100
125
150

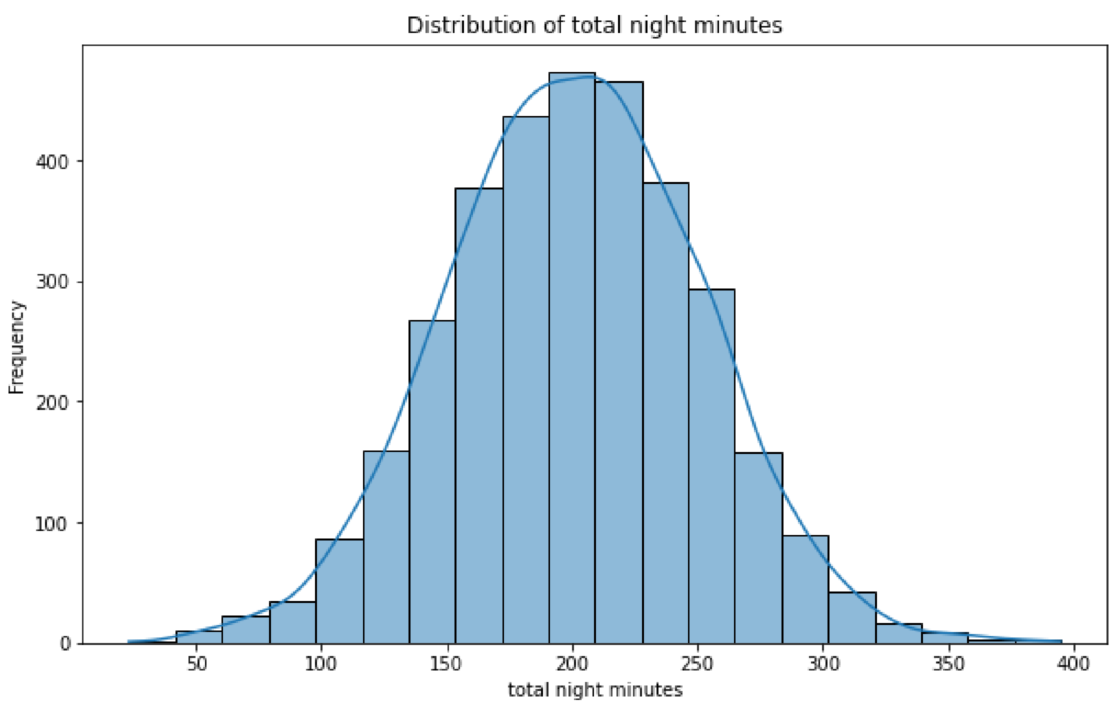


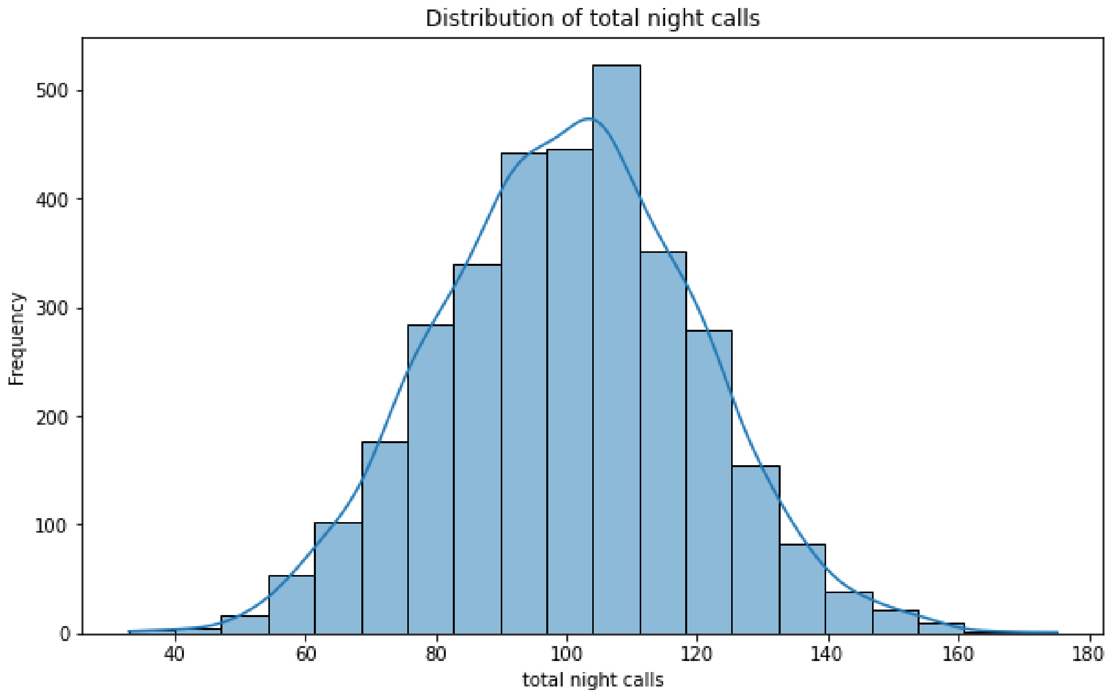
Distribution of total day charge
Frequency
total day charge
0
50
100
150
200
250
300
350
400
0
10
20
30
40
50
60


Distribution of total eve minutes
Frequency
total eve minutes
0
100
200
300
400
500
0
50
100
150
200
250
300
350

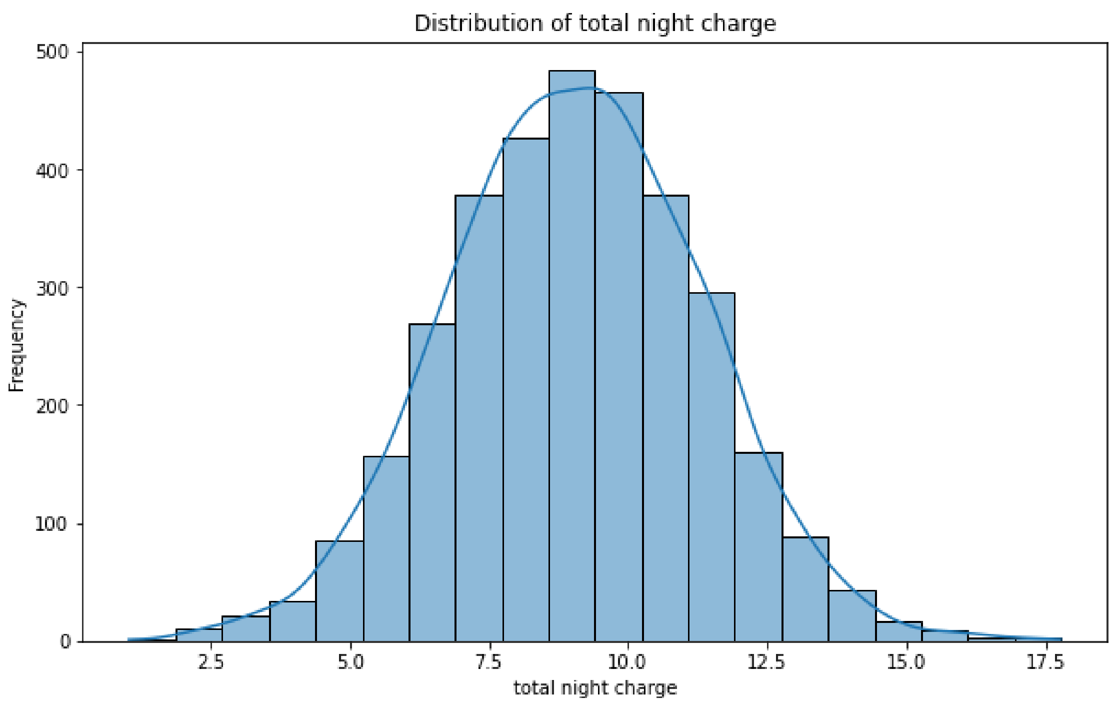

Distribution of total eve calls
Frequency
total eve calls
0
100
200
300
400
500
600
0
25
50
75
100
125
150
175


Distribution of total eve charge
Frequency
0
100
200
300
400
500
0
5
10
15
20
25
30
total eve charge

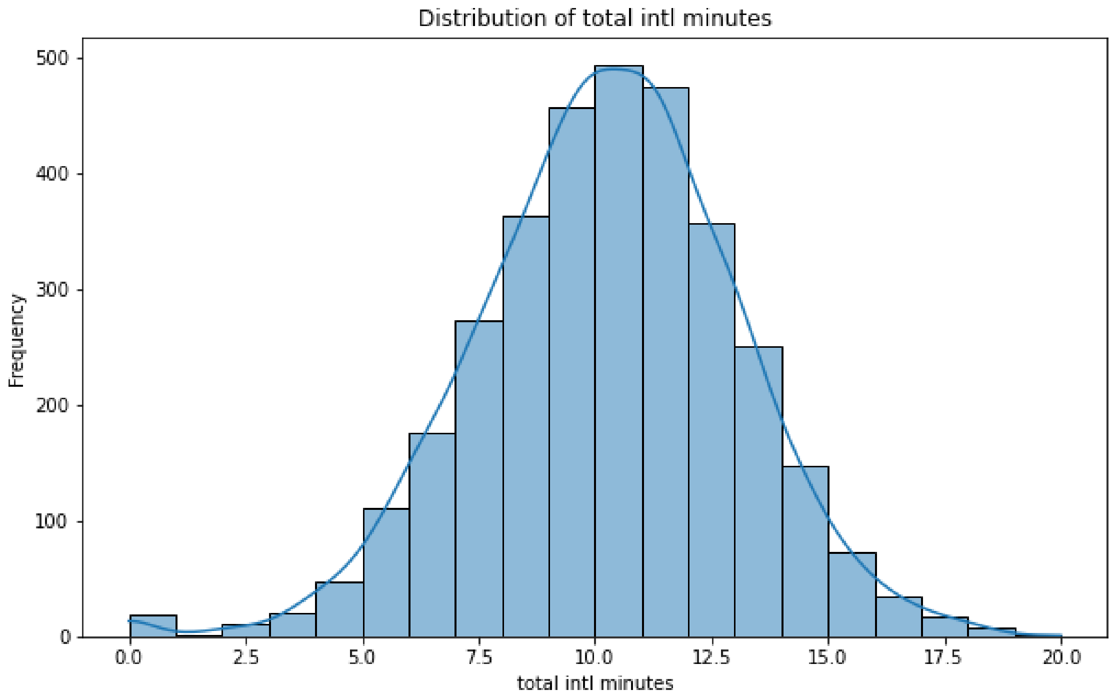

Distribution of total night minutes
Frequency
0
100
200
300
400
50
100
150
200
250
300
350
400
total night minutes

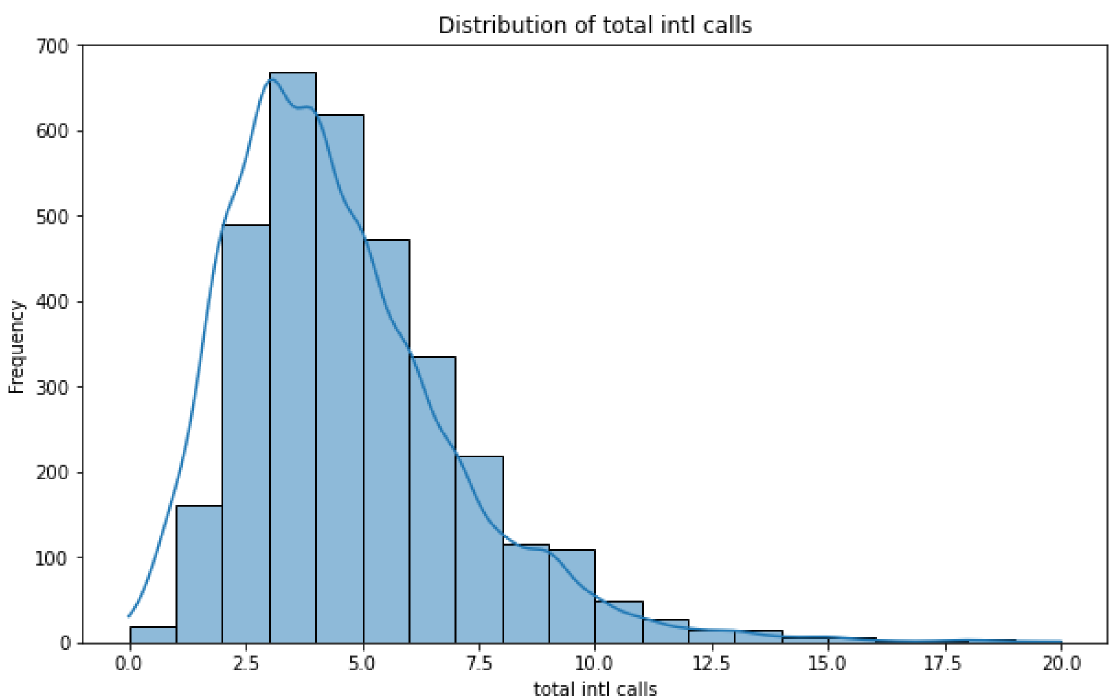


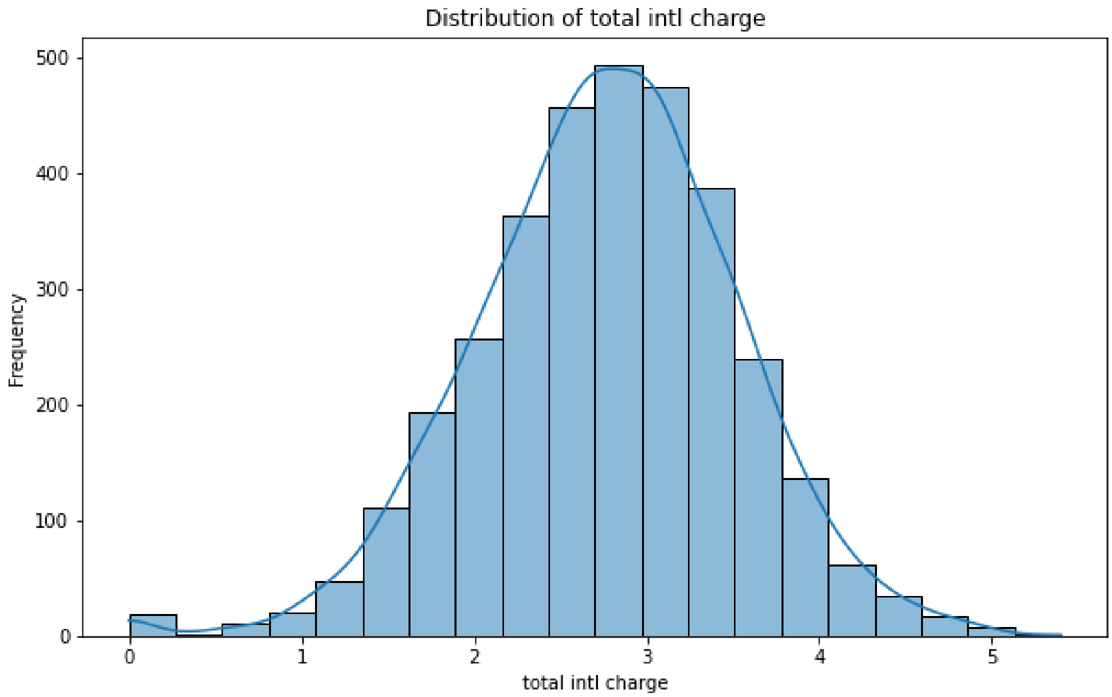
Distribution of total night calls
Frequency
total night calls
0
100
200
300
400
500
40
60
80
100
120
140
160
180

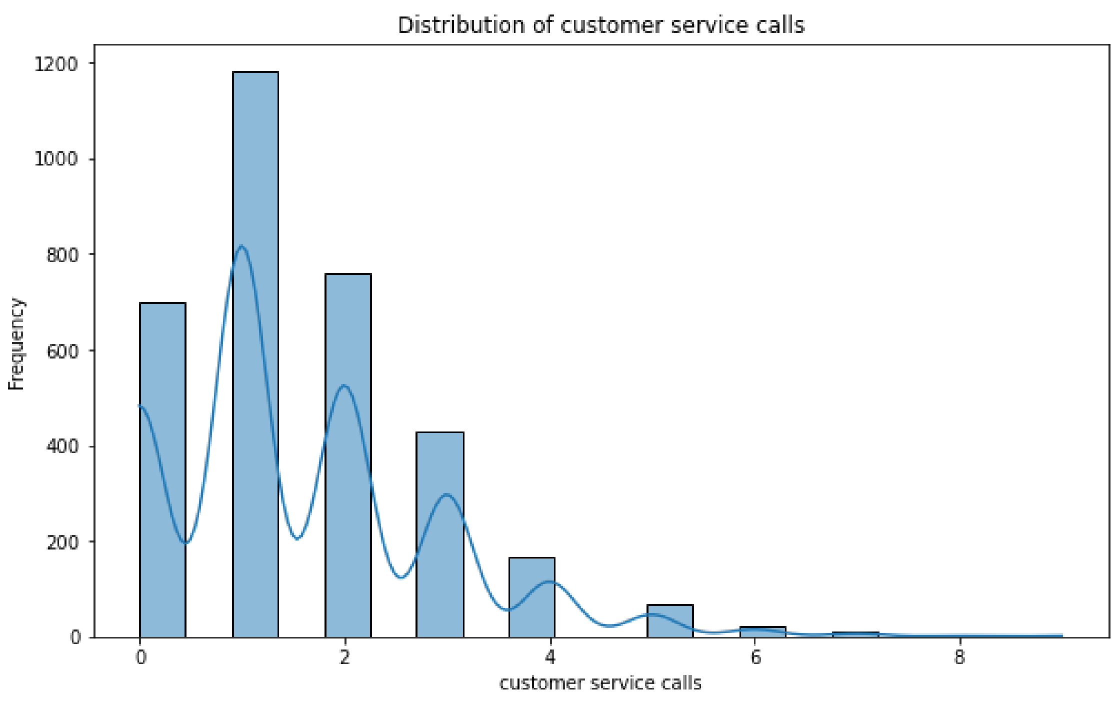


Distribution of total night charge
Frequency
total night charge
0
100
200
300
400
500
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Distribution of total intl minutes
Frequency
total intl minutes
0
100
200
300
400
500
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

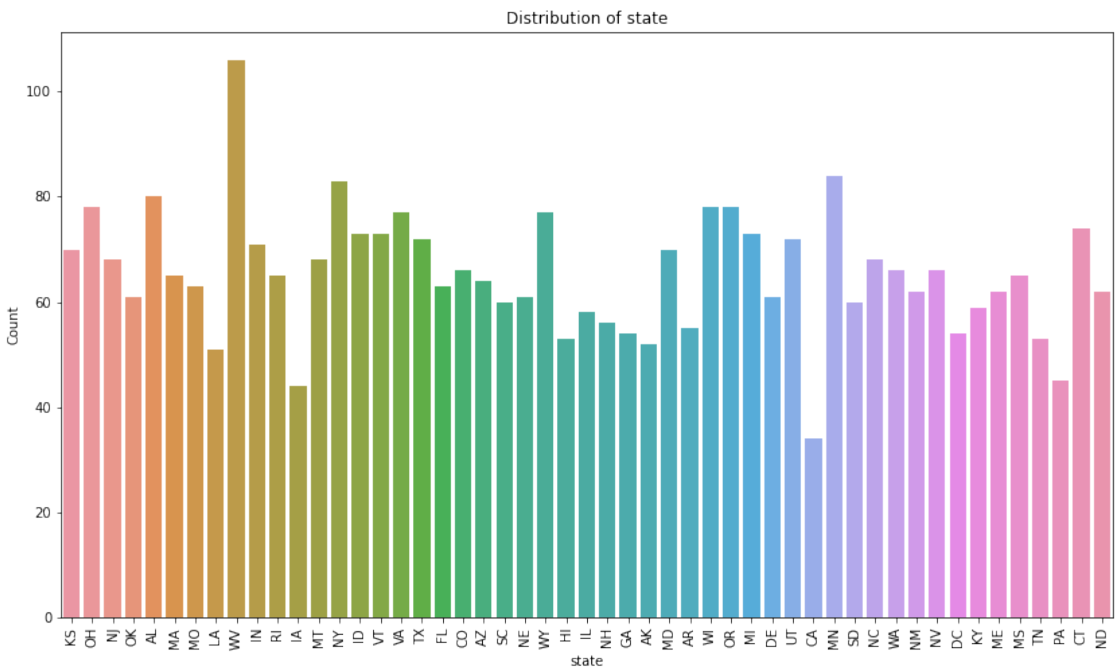

Distribution of total intl calls
Frequency
total intl calls
0
100
200
300
400
500
600
700
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

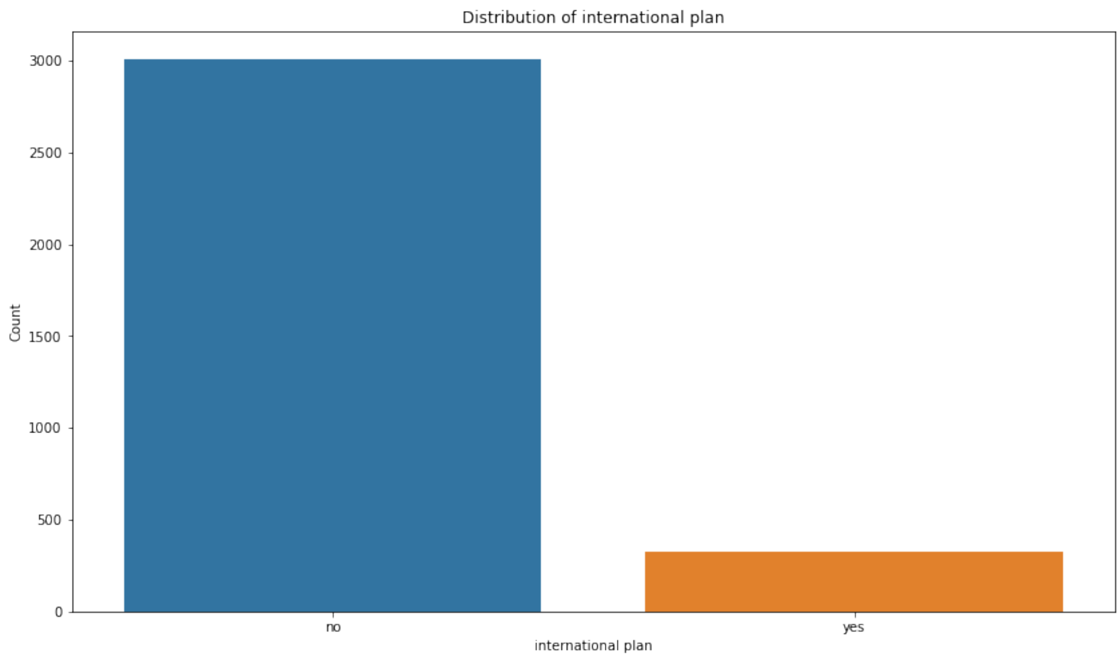


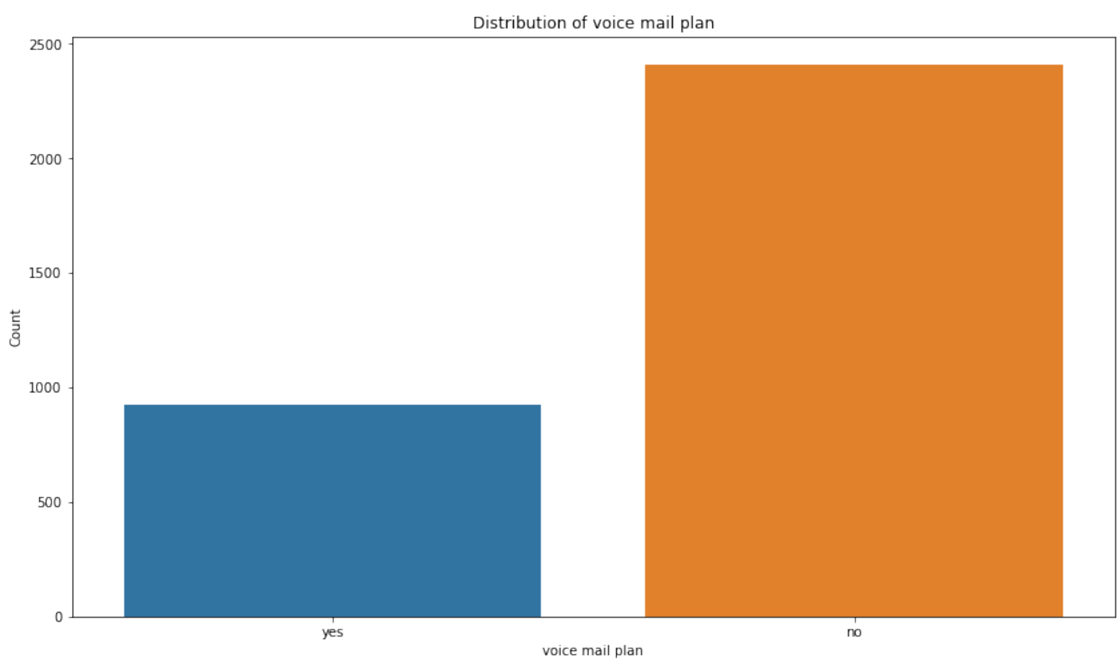


Comments: Majority of the numerical features exhibit a normal distribution.

[9]:
```python
# Univariate Analysis for Categorical Columns
categorical_features = data[['state', 'international plan', 'voice mail plan']]
# Bar plots for categorical features
for feature in categorical_features:
    plt.figure(figsize=(14, 8))
    sns.countplot(x=data[feature])
    plt.title(f'Distribution of {feature}')
    plt.xlabel(feature)
    plt.ylabel('Count')
    plt.xticks(rotation=90 if feature == 'state' else 0)
    plt.show()
```





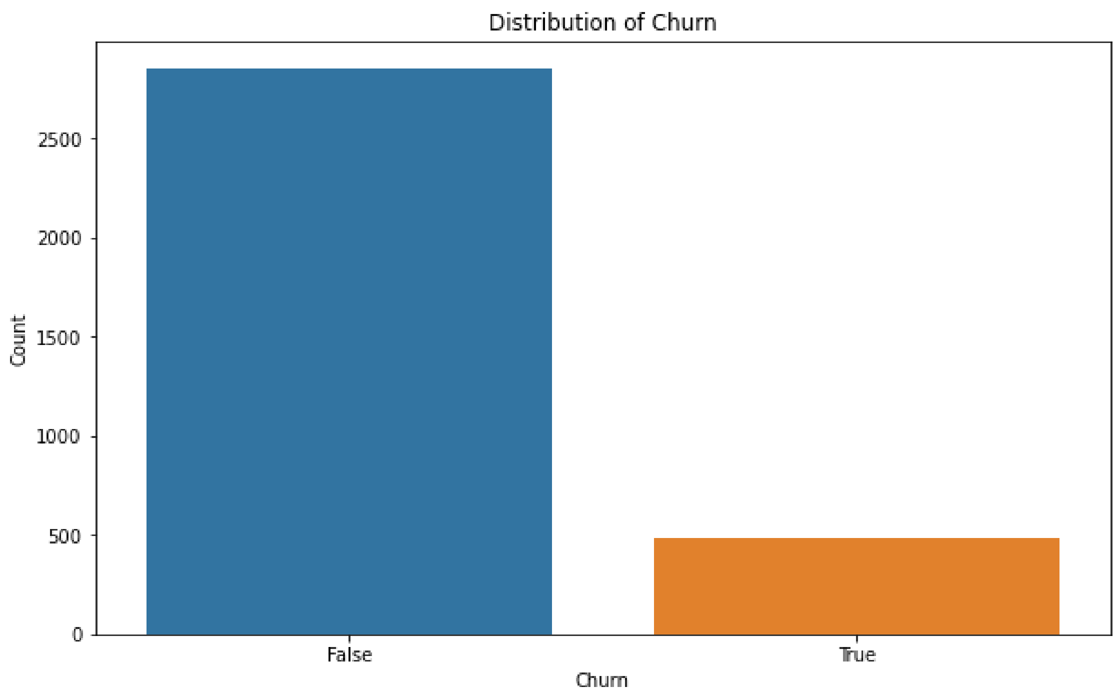


Comments: Based on the plots above, it is evident that most customers do not utilize the voicemail and international plans. Further analysis will be conducted to explore the impact of these features on the churn rate.

The distribution of customers across different states is generally even, with West Virginia (WV) having the highest number of customers and California (CA) having the lowest.

0.0.5 Target Variable Analysis

[10]:
```
# Univariate Analysis for Target Variable
plt.figure(figsize=(10, 6))
sns.countplot(x=data['churn'])
plt.title('Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```



[11]:
```
#Let's calculate the churn rate.
churn_rate = (data["churn"].mean())* 100

print(f"Churn Rate: {churn_rate:.2f}%")
print(data['churn'].value_counts())
```

```
Churn Rate: 14.49%
False    2850
True      483
Name: churn, dtype: int64
```

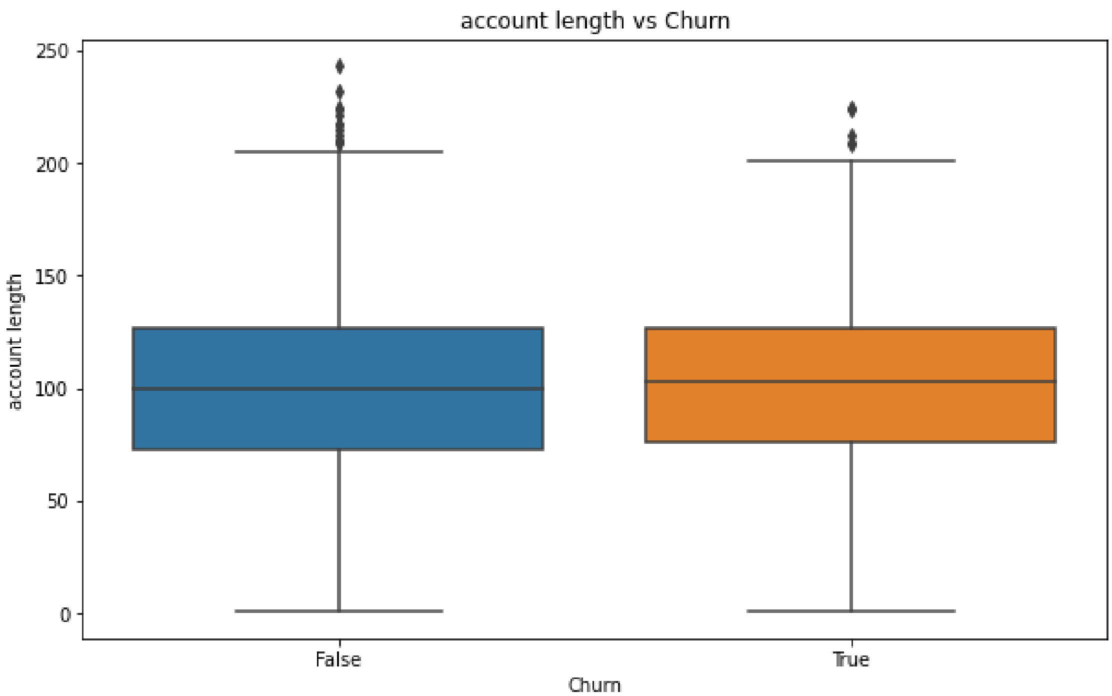
Comments: SyriaTel's churn rate of 14.49% is way below the industrial standard of 30 - 35%(source = Google) but we can still explore our data further and derive more insights to lower it even further.

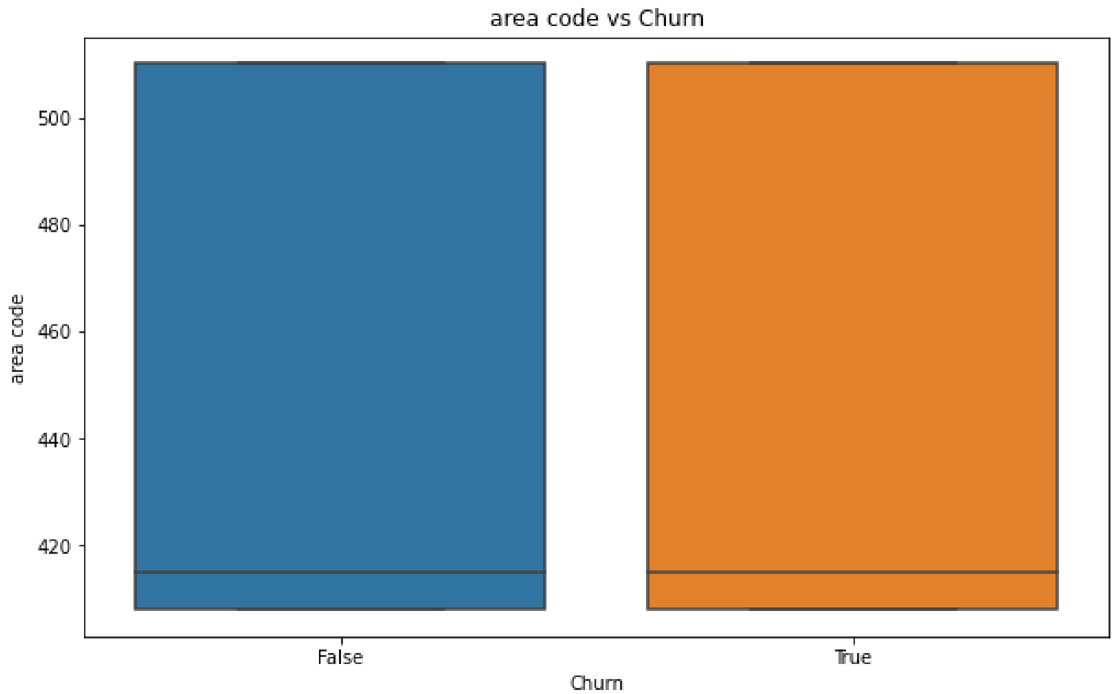
NB: It is evident from the visual above that we have a class imbalance issue that we will have to remedy before the modelling process is started.

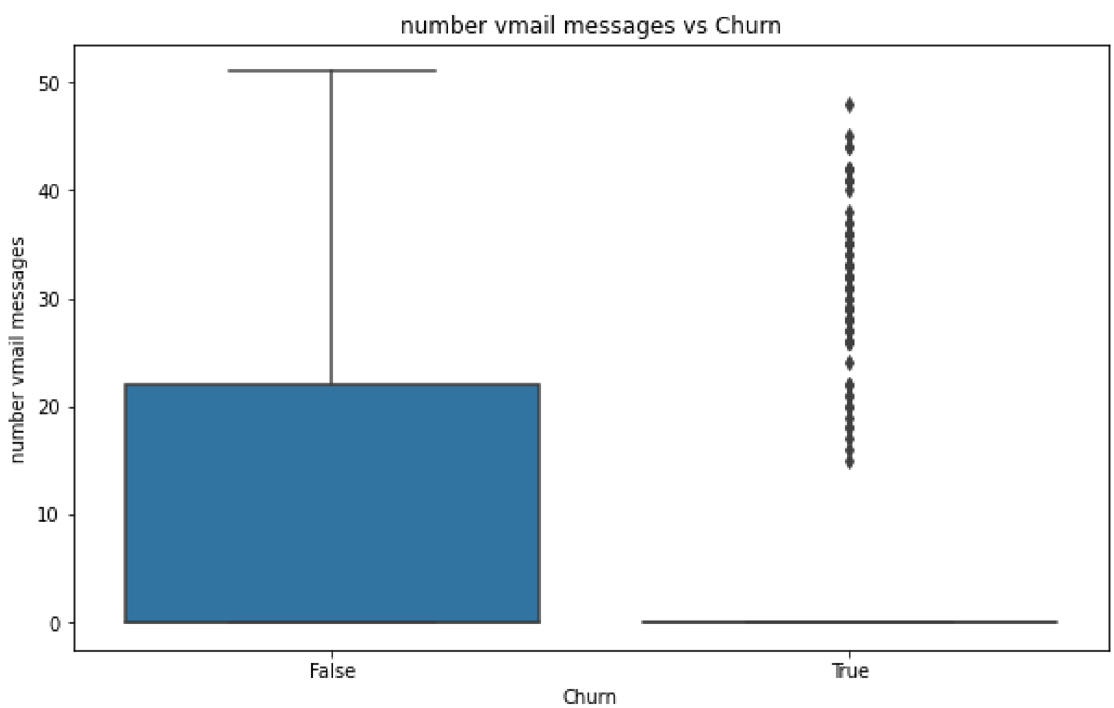
0.0.6 Bi_variate Analysis

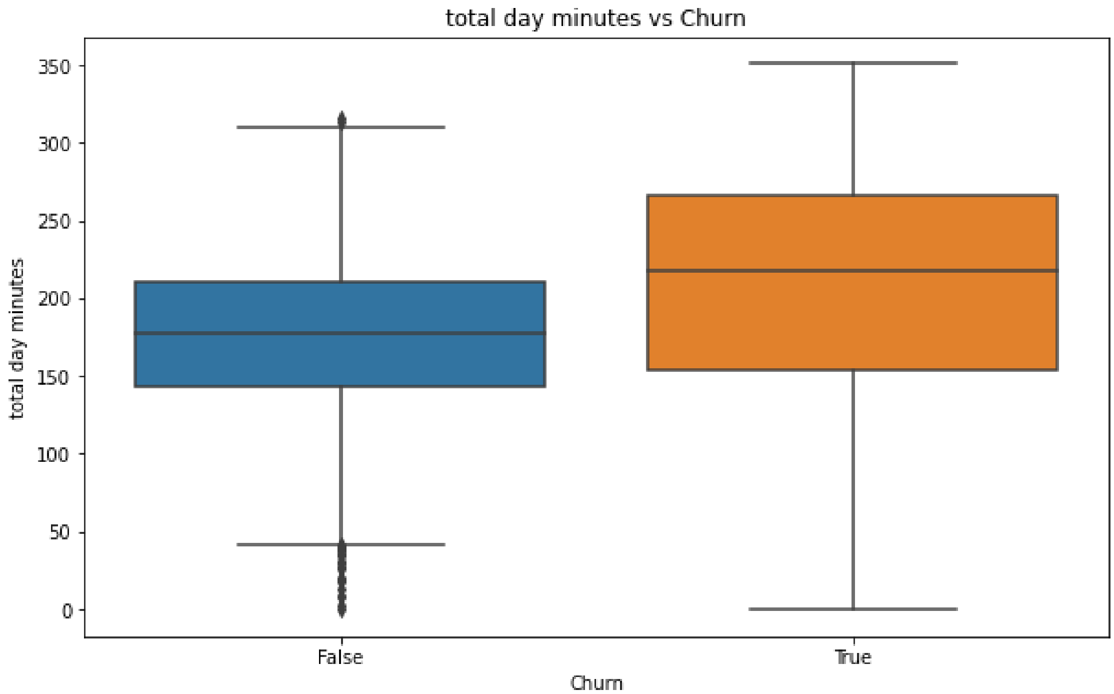
Let's see how some of these features relate/affect the target variable(Churn).

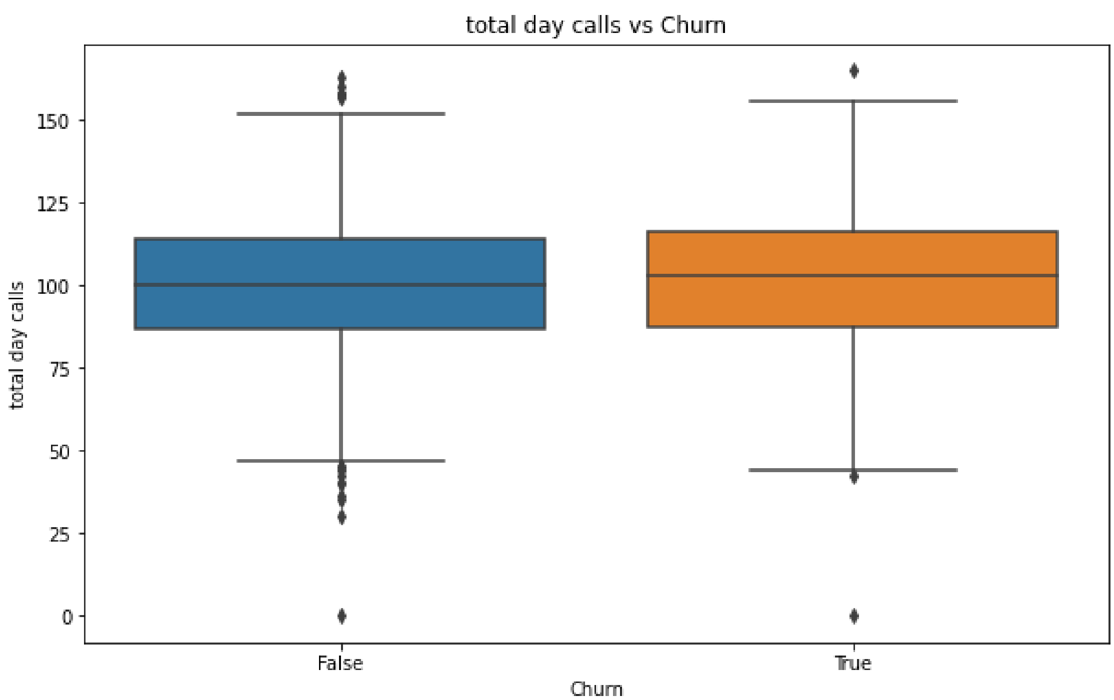
[12]:
```
#Bivariate analysis for numerical features
for feature in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.boxplot(x='churn', y=feature, data=data)
    plt.title(f'{feature} vs Churn')
    plt.xlabel('Churn')
    plt.ylabel(feature)
    plt.show()
```

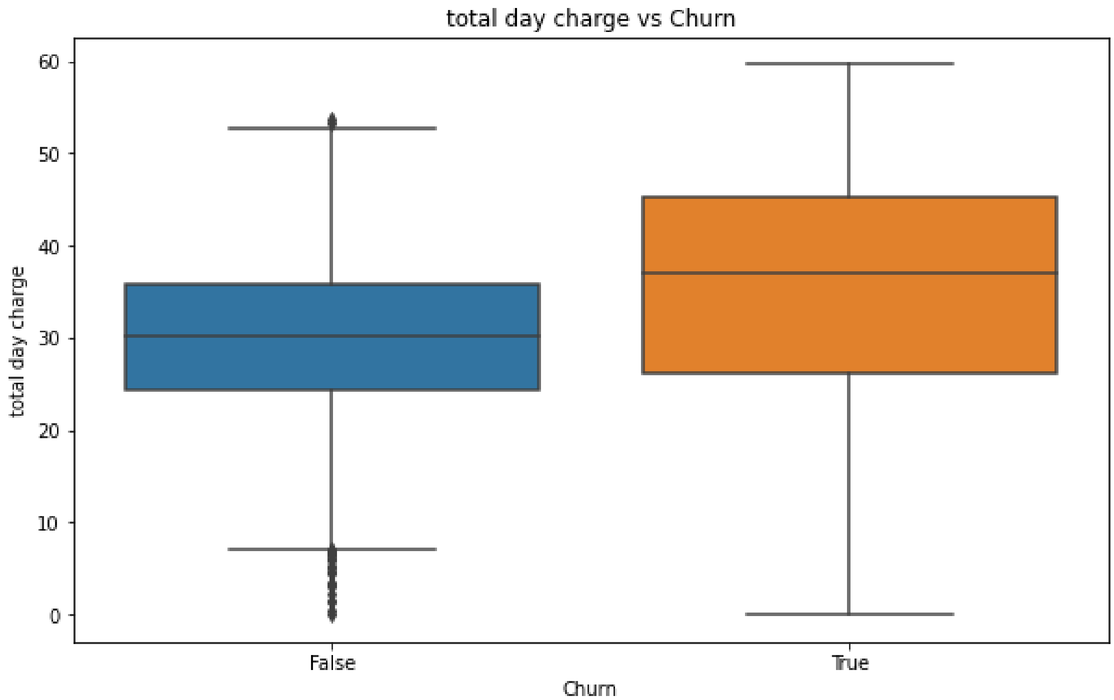


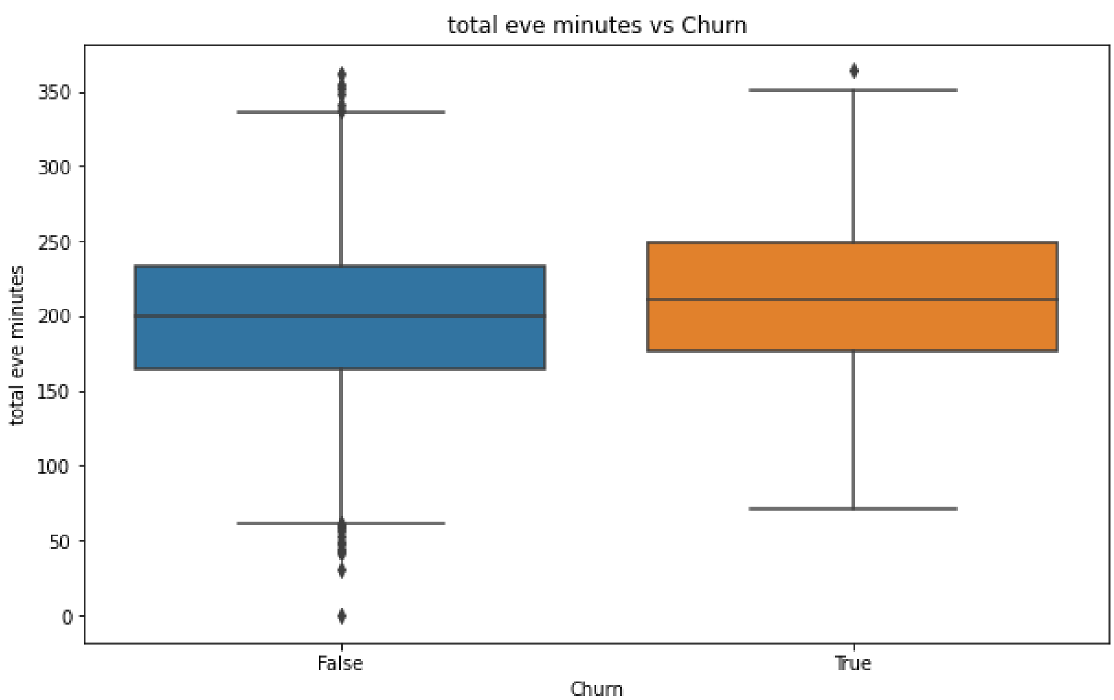

area code vs Churn
500
480
460
440
420
area code
False
True
Churn

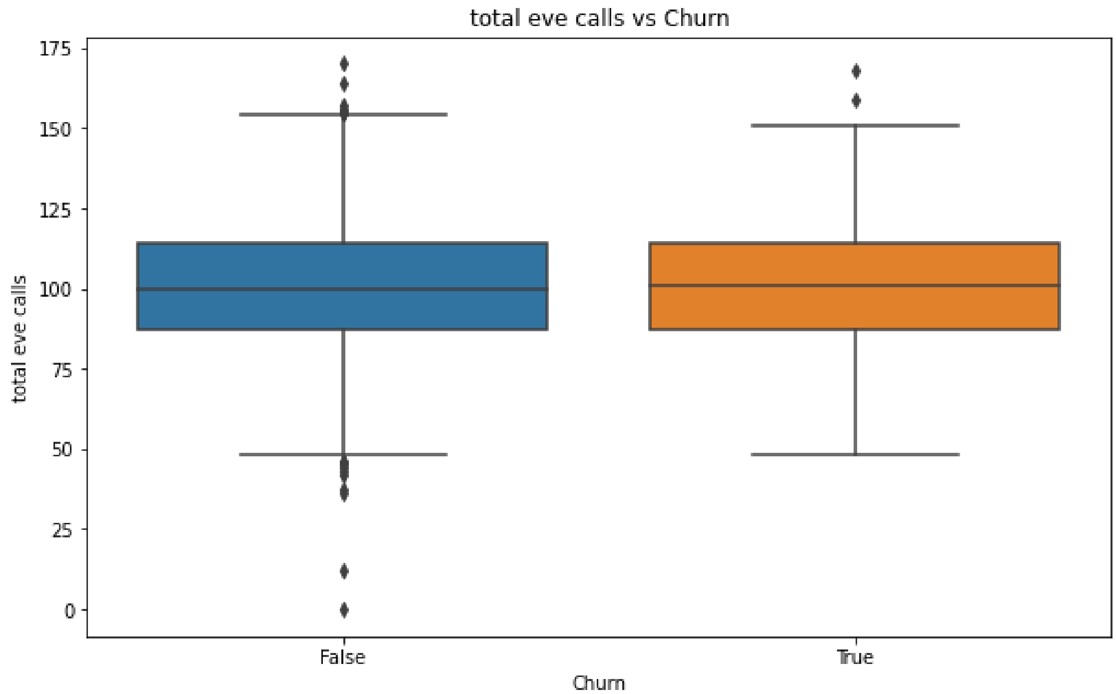

number vmail messages vs Churn
50
40
30
20
10
0
number vmail messages
False
True
Churn

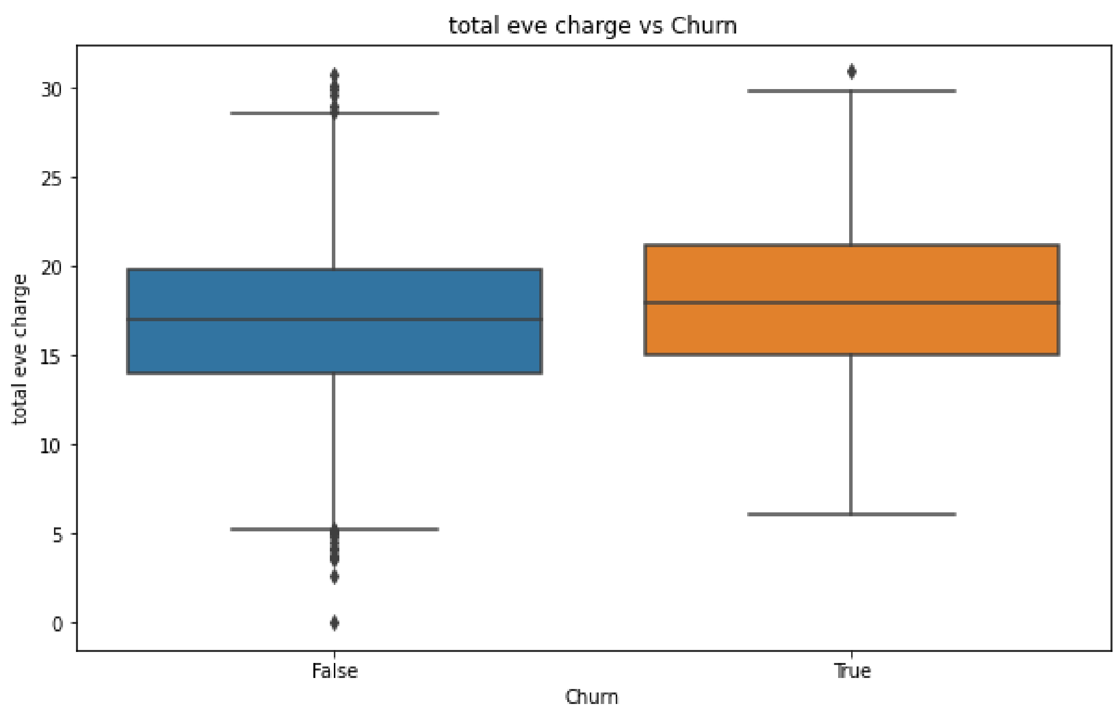

total day minutes vs Churn
total day minutes
Churn
False
True
0
50
100
150
200
250
300
350

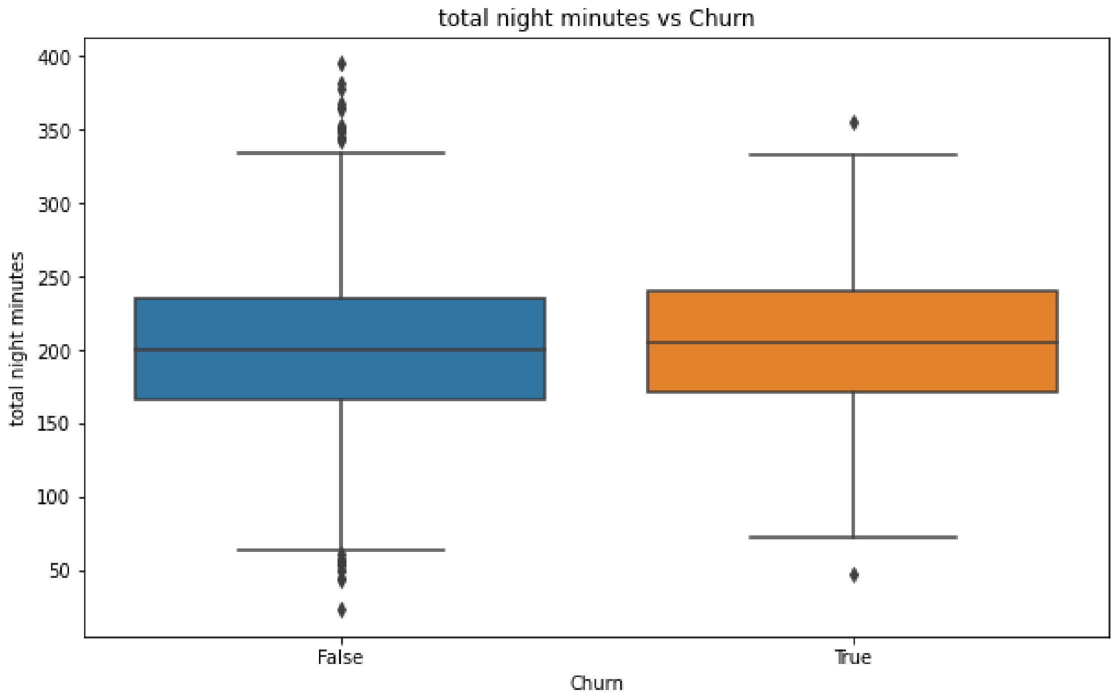

total day calls vs Churn
total day calls
Churn
False
True
0
25
50
75
100
125
150

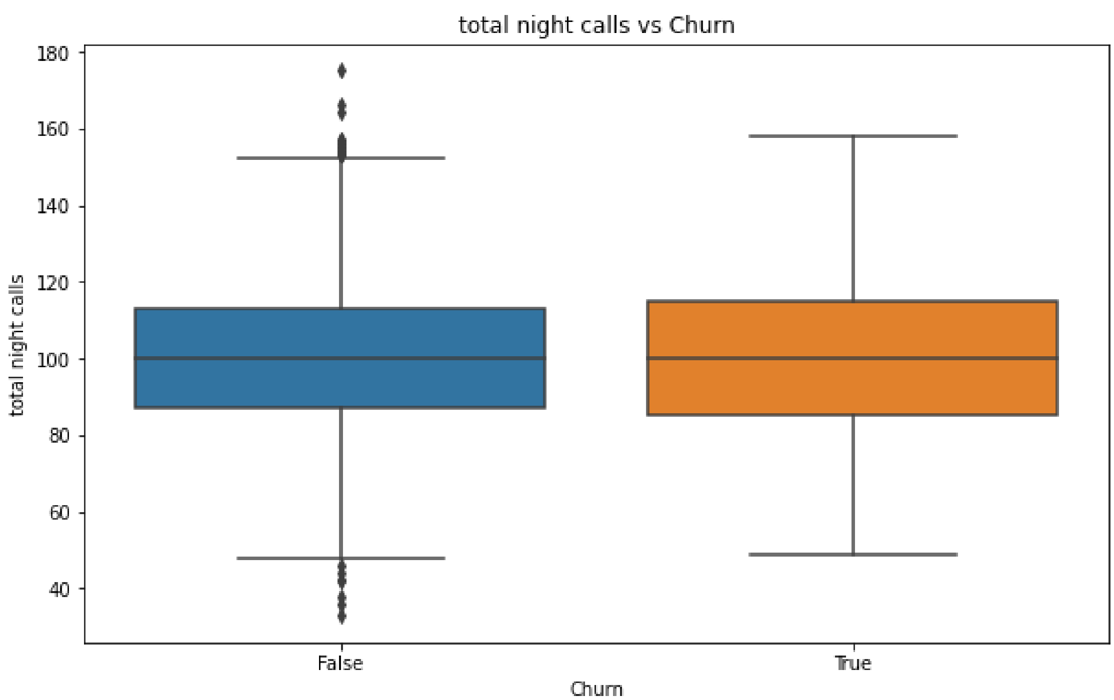

total day charge vs Churn
total day charge
60
50
40
30
20
10
0
False
True
Churn

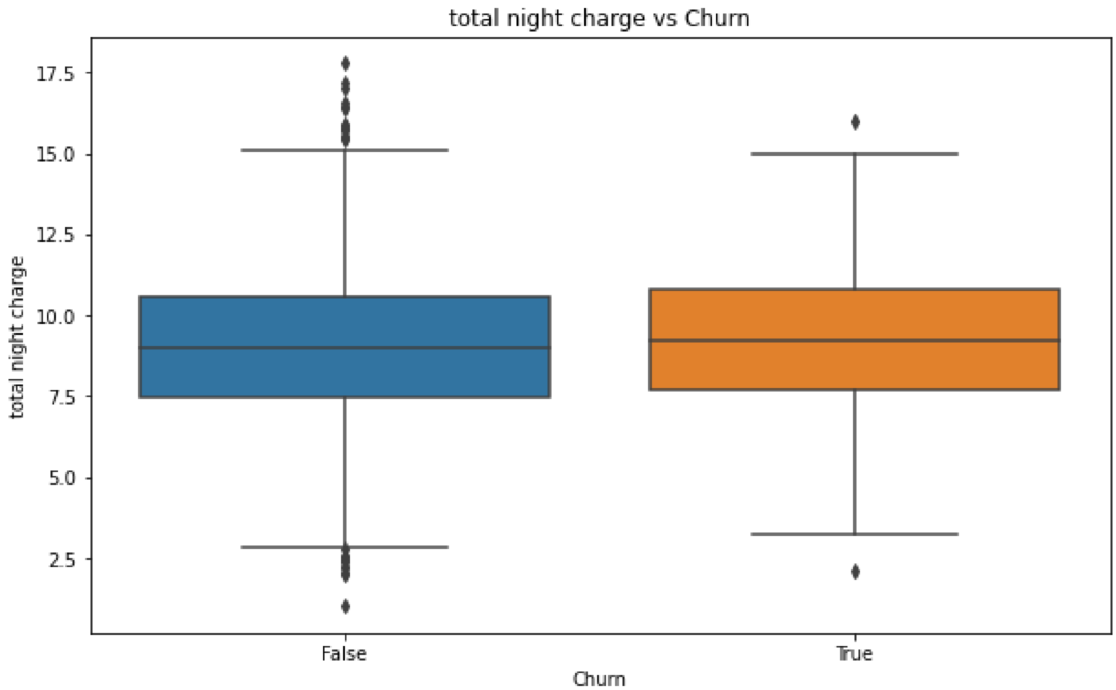

total eve minutes vs Churn
total eve minutes
350
300
250
200
150
100
50
0
False
True
Churn

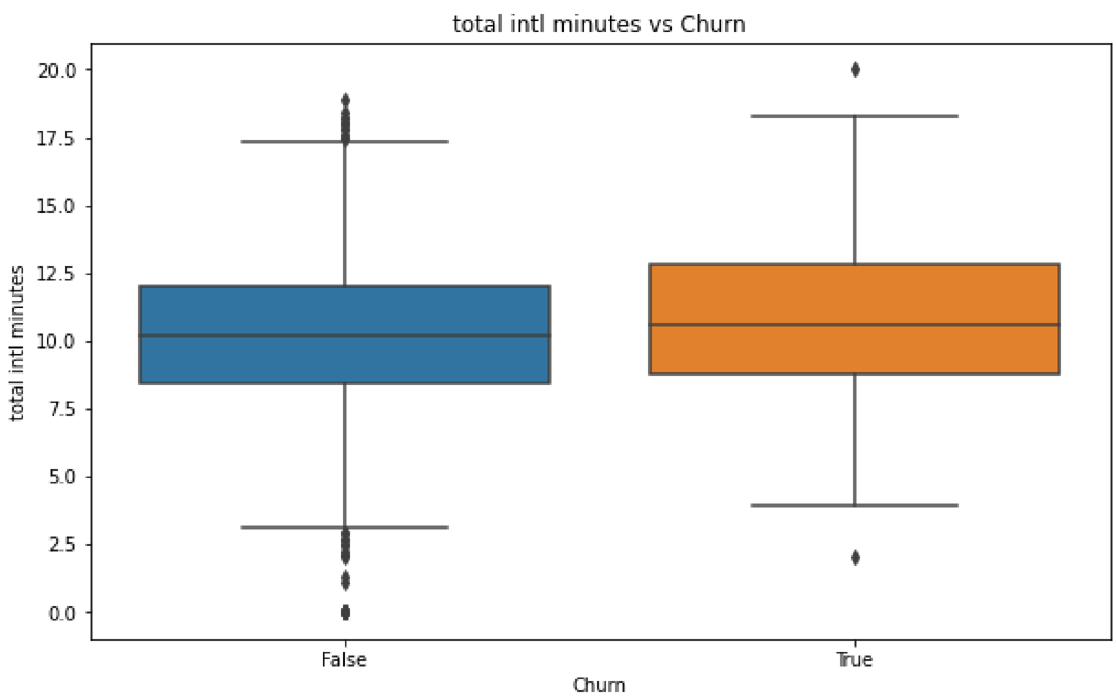

total eve calls vs Churn
175
150
125
100
75
50
25
0
total eve calls
False
True
Churn

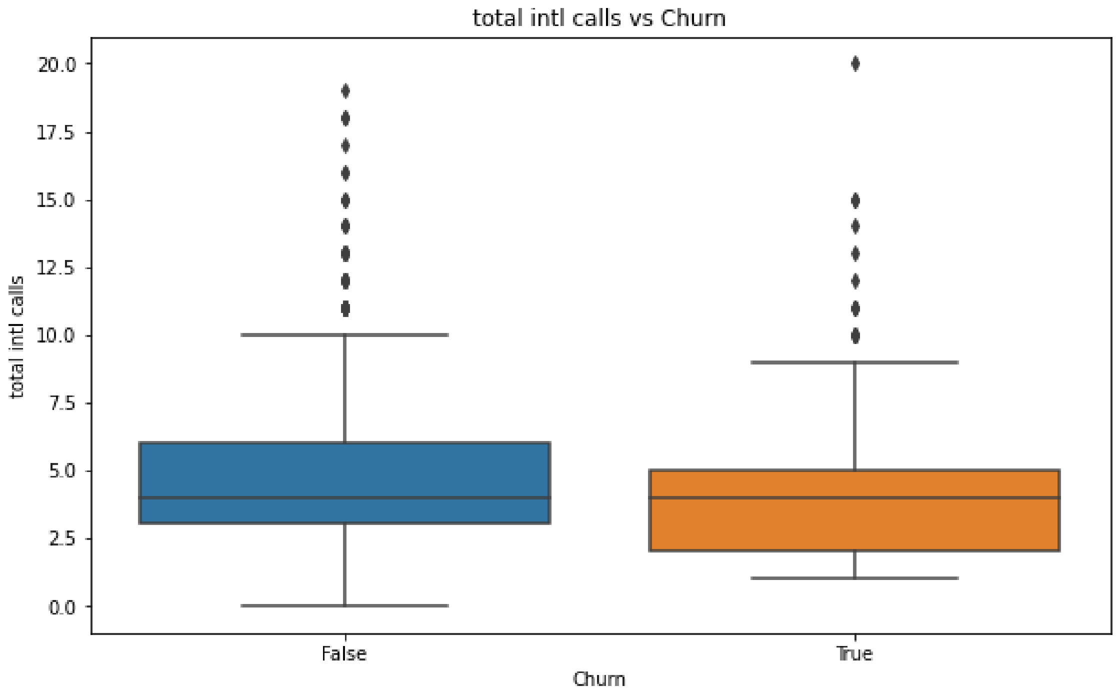

total eve charge vs Churn
30
25
20
15
10
5
0
total eve charge
False
True
Churn

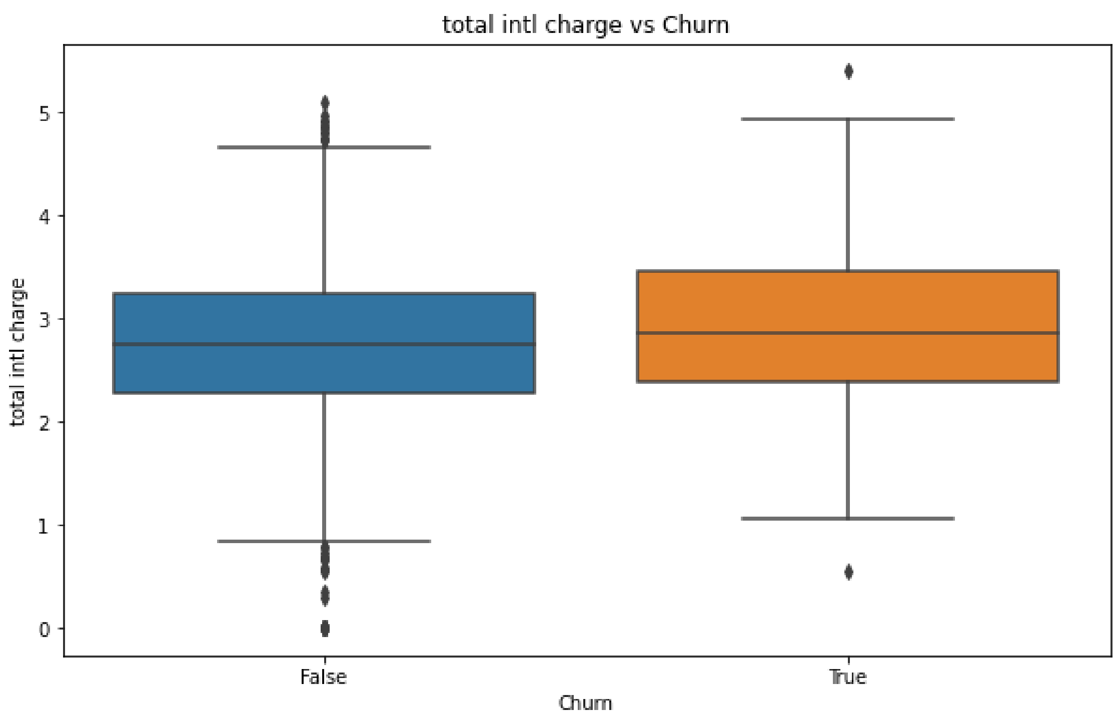

total night minutes vs Churn
400
350
300
250
200
150
100
50
total night minutes
False
True
Churn

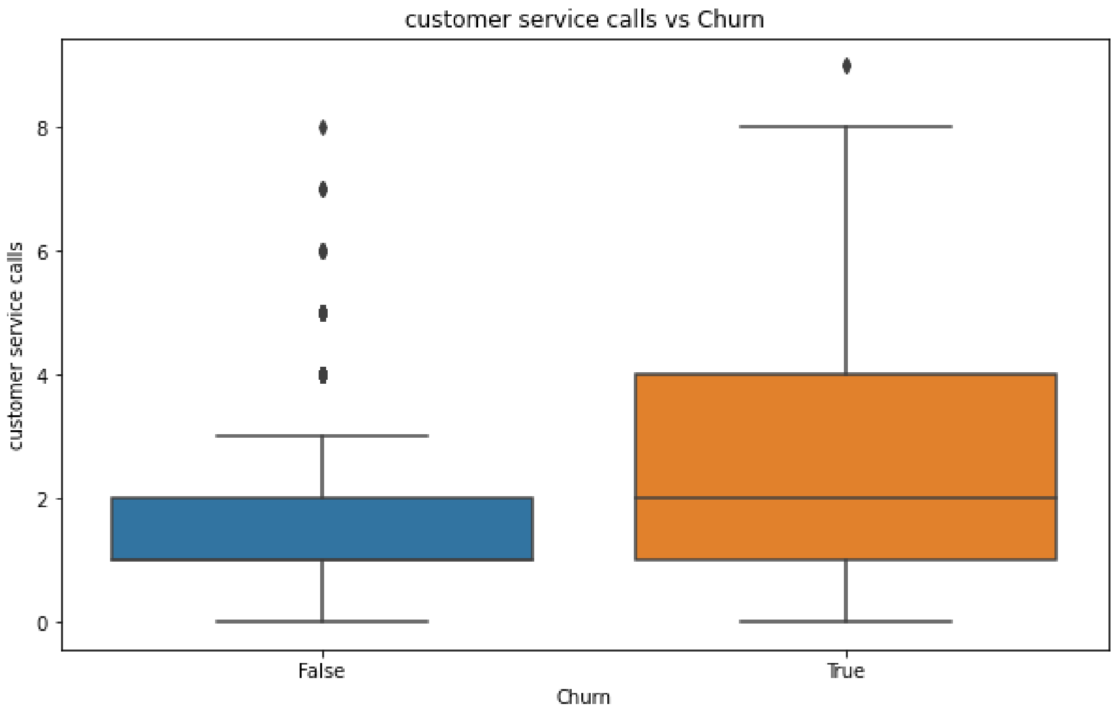

total night calls vs Churn
180
160
140
120
100
80
60
40
total night calls
False
True
Churn


total night charge vs Churn
17.5
15.0
12.5
10.0
7.5
5.0
2.5
total night charge
False
True
Churn


total intl minutes vs Churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
total intl minutes
False
True
Churn


total intl calls vs Churn
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
total intl calls
False
True
Churn

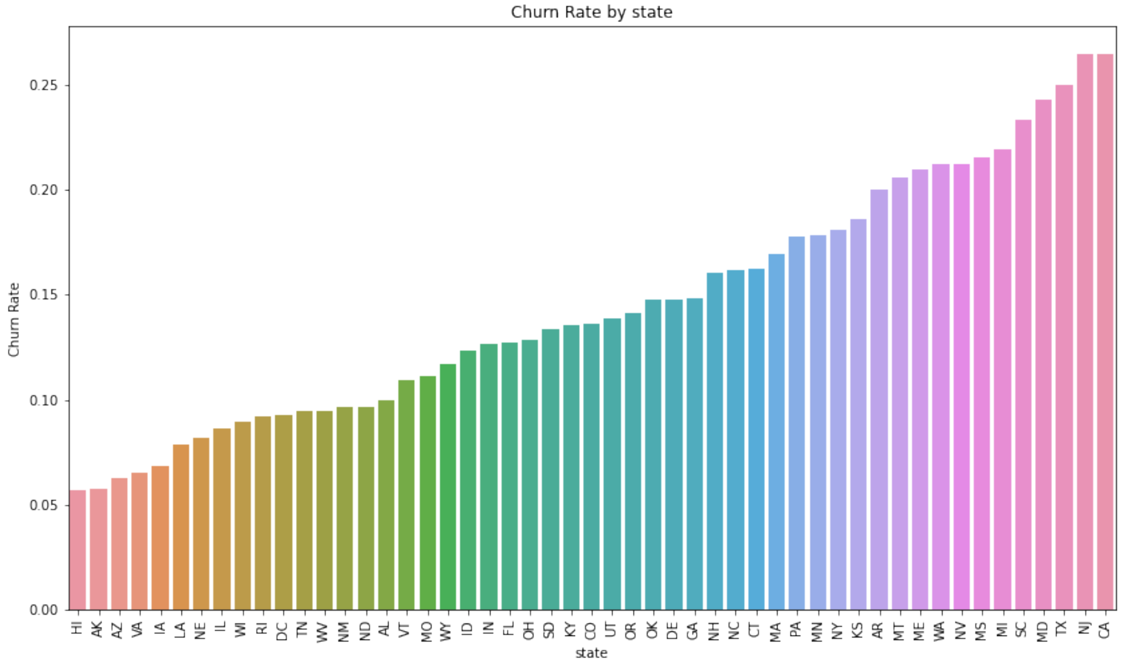

total intl charge vs Churn
5
4
3
2
1
0
total intl charge
False
True
Churn

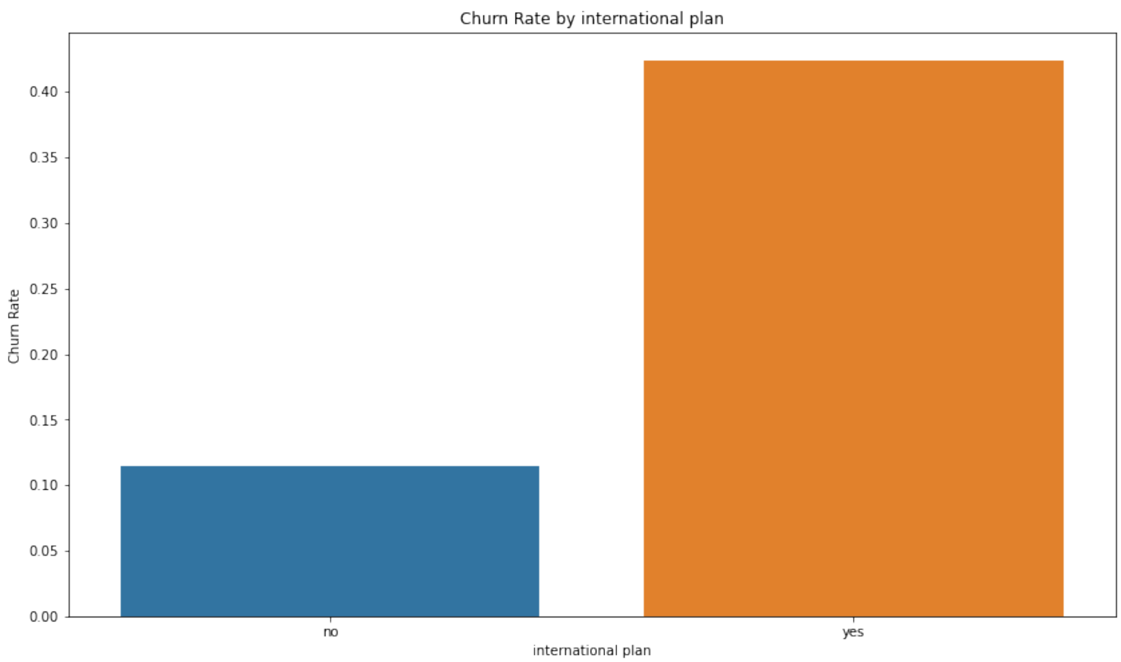


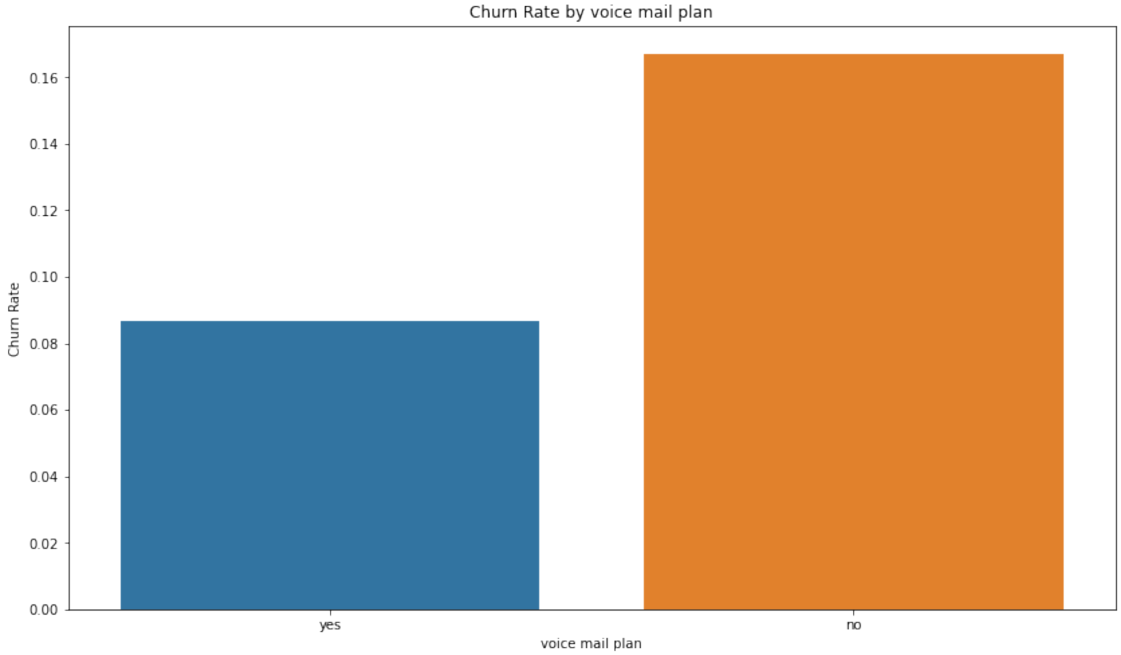
Comments: It's interesting that customers with higher usage ie Total minutes (Day, Evening, Night, International) tend to churn as compared to low and average usage customers. Similar to the minutes features, higher charges seem to be associated with customers who churn, reflecting the direct relationship between usage and charges. This will however require more investigation

The only clear observation is that customers who have churned tend to have made more customer service calls, indicating that frequent customer service interactions might be linked to dissatisfaction and churn. This also reveals that the outliers exhibited in this column are of importance in the analysis.

[13]:
```
# Generate bar plots for categorical features
for feature in categorical_features:
    plt.figure(figsize=(14, 8))
    churn_rate = data.groupby(feature)['churn'].mean().sort_values()
    sns.barplot(x=churn_rate.index, y=churn_rate.values)
    plt.title(f'Churn Rate by {feature}')
    plt.xlabel(feature)
    plt.ylabel('Churn Rate')
    plt.xticks(rotation=90 if feature == 'state' else 0)
    plt.show()
```


Churn Rate by state
Churn Rate
state
0.00
0.05
0.10
0.15
0.20
0.25
HI AK AZ VA IA LA NE IL WI RI DC TN WV NM ND AL VT MO WY ID IN FL OH SD KY CO UT OR OK DE GA NH NC CT MA PA MN NY KS AR MT ME WA NV MS MI SC MD TX NJ CA

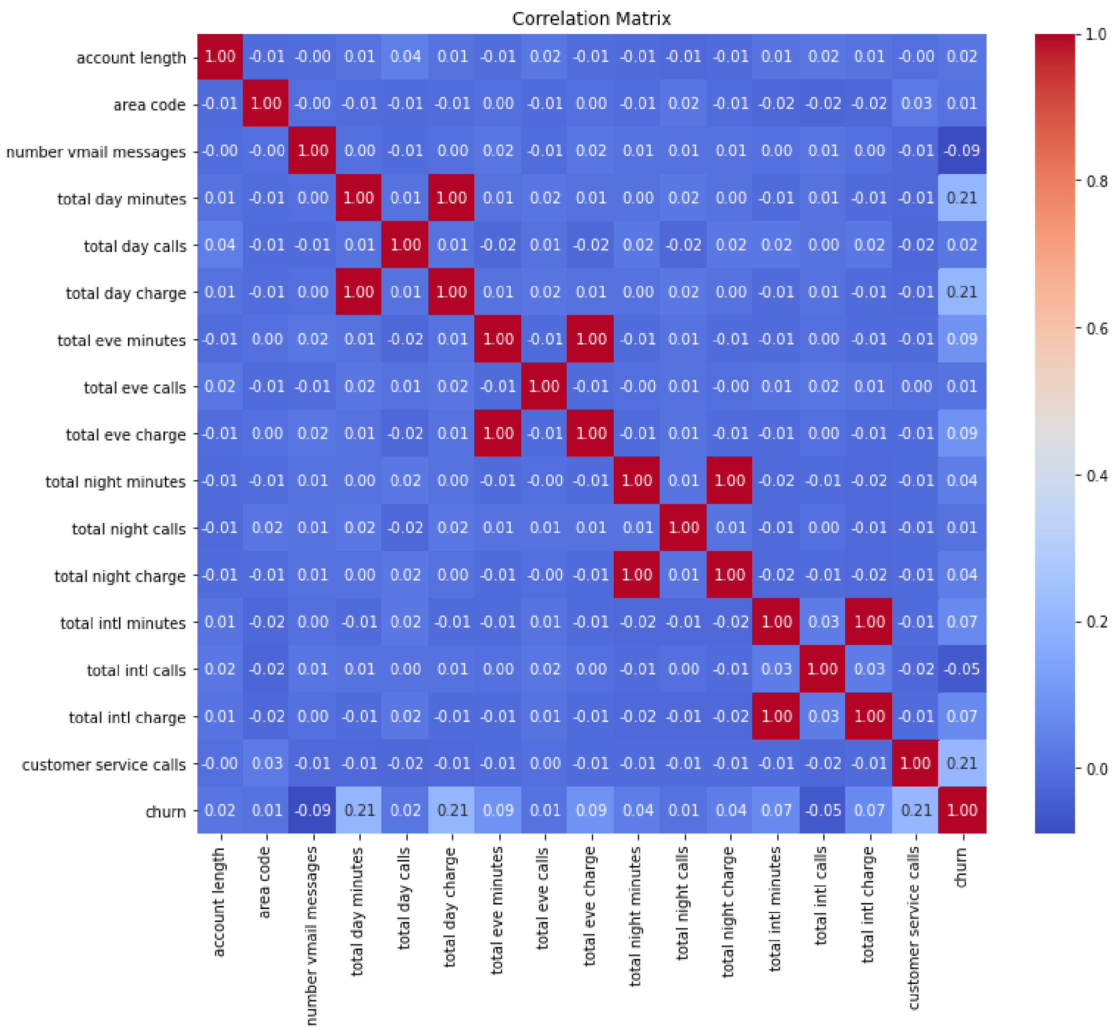

Churn Rate by international plan
Churn Rate
international plan
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
no
yes



Comments: Customers with an international plan are more likely to churn, hence indicating some dissatisfaction with the service, given the observations from earlier this could be due to the charges on international calls.

Customers with Voice mail plans tend not to churn as opposed to those without. This could indicate the service is satisfactory to the customers.

[14]:
```
# Let's check for collinearity of features to each other and to the target␣
↪variable

# Correlation matrix
plt.figure(figsize=(12, 10))
correlation_matrix = data.corr()

# Heatmap of the correlation matrix
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```



As per the graphic above, all the charge columns extremely correlated to the minutes columns to a value of 1. This ideally indicates we have to drop all the minutes columns and keep charge or vice versa.
This will be executed in the data preparation for modelling section.

0.0.7 Modelling

Data preparation for modelling

[15]:
```
#Let's remind ourselves of how our dataset looks again before we start␣
↪preparing it for modelling
data.head()
```

[15]:
```
  state  account length  area code phone number international plan  \
0    KS             128        415     382-4657                 no
1    OH             107        415     371-7191                 no
2    NJ             137        415     358-1921                 no
```

```
3    OH              84       408     375-9999                yes
4    OK              75       415     330-6626                yes

  voice mail plan  number vmail messages  total day minutes  total day calls  \
0             yes                     25              265.1              110
1             yes                     26              161.6              123
2              no                      0              243.4              114
3              no                      0              299.4               71
4              no                      0              166.7              113

   total day charge  …  total eve calls  total eve charge  \
0             45.07  …               99             16.78
1             27.47  …              103             16.62
2             41.38  …              110             10.30
3             50.90  …               88              5.26
4             28.34  …              122             12.61

   total night minutes  total night calls  total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   total intl minutes  total intl calls  total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   customer service calls  churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

[5 rows x 21 columns]
```

Column drops Based on domain knowledge we will drop the phone number column since it would not make a good predictor of churn and we will drop the state column as well since we already have another location column(area code).

Also based on the multi_collinearity check done above We will drop all the minute columns(Total Day minutes, Total Evening Minutes, Total Night Minutes and Total International Minutes) Due to extremely high multi_Collinearity with the charge columns

```
[16]: data = data.drop(columns =['phone number', 'state', 'total day minutes', 'total␣
       ↪eve minutes', 'total night minutes', 'total intl minutes'],axis = 1)
      data. head()
```

```
[16]:    account length  area code international plan voice mail plan  \
      0             128        415                 no             yes
      1             107        415                 no             yes
      2             137        415                 no              no
      3              84        408                yes              no
      4              75        415                yes              no

         number vmail messages  total day calls  total day charge  total eve calls  \
      0                     25              110             45.07               99
      1                     26              123             27.47              103
      2                      0              114             41.38              110
      3                      0               71             50.90               88
      4                      0              113             28.34              122

         total eve charge  total night calls  total night charge  total intl calls  \
      0             16.78                 91               11.01                 3
      1             16.62                103               11.45                 3
      2             10.30                104                7.32                 5
      3              5.26                 89                8.86                 7
      4             12.61                121                8.41                 3

         total intl charge  customer service calls  churn
      0               2.70                       1  False
      1               3.70                       1  False
      2               3.29                       0  False
      3               1.78                       2  False
      4               2.73                       3  False
```

Label encoding Below, I will do what most will consider blasphemous when it comes to modelling, I will do a label encoding before a train_test_split. The basis of this is that we're just translating the values (yes/no, False/True) to 0's and 1's and not transforming them.Ideally this is supposed to lead to data leakage but I base my reasoning on this kaggle correspondence on the subject issue.

```
[17]: label_encoder = LabelEncoder()
      data['churn'] = label_encoder.fit_transform(data['churn'])
      data['international plan'] = label_encoder.fit_transform(data['international␣
       ↪plan'])
      data['voice mail plan'] = label_encoder.fit_transform(data['voice mail plan'])
```

```
[18]: #Let's see how the dataset looks after the encoding above.
      data.head()
```

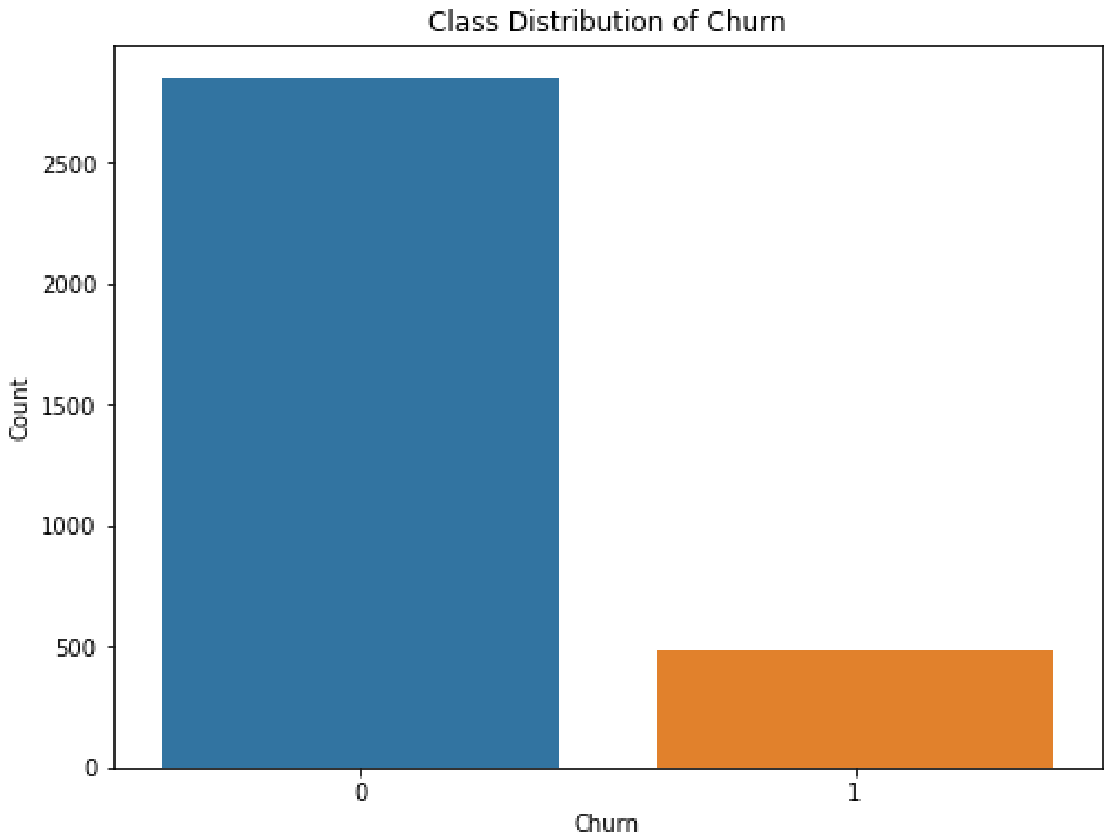
[18]:
```
   account length  area code  international plan  voice mail plan  \
0             128        415                   0                1
1             107        415                   0                1
2             137        415                   0                0
3              84        408                   1                0
4              75        415                   1                0

   number vmail messages  total day calls  total day charge  total eve calls  \
0                     25              110             45.07               99
1                     26              123             27.47              103
2                      0              114             41.38              110
3                      0               71             50.90               88
4                      0              113             28.34              122

   total eve charge  total night calls  total night charge  total intl calls  \
0             16.78                 91               11.01                 3
1             16.62                103               11.45                 3
2             10.30                104                7.32                 5
3              5.26                 89                8.86                 7
4             12.61                121                8.41                 3

   total intl charge  customer service calls  churn
0               2.70                       1      0
1               3.70                       1      0
2               3.29                       0      0
3               1.78                       2      0
4               2.73                       3      0
```

Let's revisit the issue of class imbalance seen earlier in our analysis of the target variable.

[19]:
```python
class_counts = data['churn'].value_counts()
print("Class Distribution:\n", class_counts)
print("\nClass Proportions:\n", class_counts / len(data))

# Visualize class distribution
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(8, 6))
sns.countplot(x='churn', data=data)
plt.title('Class Distribution of Churn')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()
```

```
Class Distribution:
 0    2850
1     483
```

```
Name: churn, dtype: int64

Class Proportions:
 0    0.855086
1    0.144914
Name: churn, dtype: float64
```



As seen above we do have a major issue of class imbalance that has to be addressed before we begin the modelling process. we will use SMOTE as our method to resolve the issue.

We will also define our Target and Predictors

[20]:
```
# Defining of the target(y) and the predictors(X)
X = data.drop(columns = ['churn'])
y = data['churn']

# Split the data into training and testing sets to (70:30 ratio)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
↪random_state=42)

# Previous original class distribution(y_train)
```

```python
print('Original Class Distribution: \n')
print(y_train.value_counts())

# Handle class imbalance using SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train, y_train)

# Let's see how the class look after applying SMOTE
print("Class Distribution After Resampling:\n", pd.Series(y_train_resampled).
 ↪value_counts())
```

```
Original Class Distribution:

0    1993
1     340
Name: churn, dtype: int64
Class Distribution After Resampling:
 1    1993
0    1993
Name: churn, dtype: int64
```

Baseline Model. Since our data is already prepared for modelling. We will begin the process by comparing 3 classifiers(DT,KNN & LR) all using their default parameters and with the data scaled using the Standard scaler. The best performing model will be chosen as our Baseline model. We will use accuracy and F1_score as the metric on this particular section. And we will also print out a confusion matrix in the process. May the best model win.

[21]:
```python
# Construct pipelines with default values

# Default Decision Tree Model
pipe_dt = Pipeline([('scl', StandardScaler()),
                    ('clf', DecisionTreeClassifier(random_state = 42))])
# Default KNN Model
pipe_knn = Pipeline([('scl', StandardScaler()),
                     ('clf', KNeighborsClassifier())])
# Default Logistic Regression Model
pipe_lr = Pipeline([('scl', StandardScaler()),
                    ('clf', LogisticRegression(random_state = 42))])

# List of pipelines for ease of iteration
pipelines = [pipe_dt, pipe_knn, pipe_lr]

# Dictionary of pipelines and classifier types for ease of reference
pipe_dict = {0: 'Decision Tree', 1: 'K-Nearest Neighbors', 2: 'Logistic␣
 ↪Regression'}

# Fit the pipelines and compare metrics
```

```python
print("Training and evaluating baseline models...\n")
for idx, pipe in enumerate(pipelines):
    pipe.fit(X_train_resampled, y_train_resampled)
    y_pred = pipe.predict(X_test)
    # We will use Accuracy and F1_score as a combined metric for our models.
    print(f'{pipe_dict[idx]} pipeline metrics:')
    print(f'Accuracy: {accuracy_score(y_test, y_pred):.3f}')
    print(f'F1-Score: {f1_score(y_test, y_pred):.3f}')
    print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred)}\n')
```

```
Training and evaluating baseline models…

Decision Tree pipeline metrics:
Accuracy: 0.800
F1-Score: 0.537
Confusion Matrix:
 [[684 173]
 [ 27 116]]

K-Nearest Neighbors pipeline metrics:
Accuracy: 0.744
F1-Score: 0.448
Confusion Matrix:
 [[640 217]
 [ 39 104]]

Logistic Regression pipeline metrics:
Accuracy: 0.705
F1-Score: 0.420
Confusion Matrix:
 [[598 259]
 [ 36 107]]
```

It's quite evident the `Decision Tree Model` is the best classifier amongst the three. Hence We will proceed to tune this particular model below and see if it will perform better. We will use Gridsearch to find the best hyperparameters.

Decision Tree Classifier

[22]:
```python
# let's see the default parameters that got the model to an accuracy of 0.800
default_params = pipe_dt.steps[1][1].get_params()
print(default_params)
```

```
{'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None,
'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0,
'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 2,
'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': 42,
'splitter': 'best'}
```

[23]:
```python
# Let's tune the model for optimal performance
"""
My tuning process:
 - Instatiate a model with the default parameters
 - Apply GridsearchCV to identify the optimal parameters
 - Find the optimal model
 - Score the model by it's accuracy, f1_score and print out a confusion matrix␣
↪of the same.
"""
#Defining parameter ranges for the 3 main parameters
param_range_max_depth = [None, 1,2,3,4,5]
param_range_min_samples_split = [2, 3, 4, 5]
param_range_min_samples_leaf = [1, 2, 3, 4, 5]

# Set grid search params
grid_params = [{'clf__criterion': ['gini', 'entropy'],
                'clf__min_samples_leaf': param_range_min_samples_leaf,
                'clf__max_depth': param_range_max_depth,
                'clf__min_samples_split': param_range_min_samples_split}]

# Construct grid search
gs_dt = GridSearchCV(pipe_dt, grid_params, cv=3, scoring='f1', verbose=2,␣
↪n_jobs=-1)

# Fit using grid search
gs_dt.fit(X_train_resampled, y_train_resampled)

# Get the best estimator
best_dt = gs_dt.best_estimator_

# Predict with the best estimator
y_pred_dt = best_dt.predict(X_test)

# Model evaluation
print("Best Decision Tree Model Metrics:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_dt):.3f}')
print(f'F1-Score: {f1_score(y_test, y_pred_dt):.3f}')
print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred_dt)}')

# Output the best parameters
print(f'Best Parameters: {gs_dt.best_params_}')
```

```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    3.5s
[Parallel(n_jobs=-1)]: Done 616 tasks      | elapsed:    6.3s
```

```
Best Decision Tree Model Metrics:
```

```
Accuracy: 0.803
F1-Score: 0.543
Confusion Matrix:
 [[686 171]
 [ 26 117]]
Best Parameters: {'clf__criterion': 'entropy', 'clf__max_depth': None,
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2}
```

```
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed:    6.7s finished
```

Off the results above, by attempting 2160 fits with different parameter combinations, the verdict is: changing the criterion from `gini` to `entropy` increases the accuracy of the model from 0.800 to 0.803. This is while the model used defaults on all the other parameters. Let's however try to see if we could get better results with an ensemble model, specifically `A Random Forest Model`. We will create one with it's default parameters then attempt to tune it.

Random Forest Classifier

[24]:
```
"""
My tuning process:
 - Instatiate a model with the default parameters
 - Apply GridsearchCV to identify the optimal parameters
 - Find the optimal model
 - Score the model by it's accuracy, f1_score and print out a confusion matrix␣
 ↪of the same.
"""



# Construct pipeline for Random Forest with default values
pipe_rf = Pipeline([('scl', StandardScaler()),
                    ('clf', RandomForestClassifier(random_state = 42))])

# Define parameter grid for Grid Search
param_grid = {
    'clf__n_estimators': [50, 100, 200],
    'clf__max_features': ['auto', 'sqrt', 'log2'],
    'clf__max_depth': [None, 10, 20, 30, 40, 50],
    'clf__min_samples_split': [2, 5, 10],
    'clf__min_samples_leaf': [1, 2, 4]
}

# Perform Grid Search with cross-validation
gs_rf = GridSearchCV(pipe_rf, param_grid, cv=3, scoring='f1', verbose=2,␣
 ↪n_jobs=-1)
gs_rf.fit(X_train_resampled, y_train_resampled)

# Get the best estimator
best_rf = gs_rf.best_estimator_
```

```python
# Predict with the best estimator
y_pred_rf = best_rf.predict(X_test)

# Model Evaluation
print("Best Random Forest Model Metrics:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_rf):.3f}')
print(f'F1-Score: {f1_score(y_test, y_pred_rf):.3f}')
print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred_rf)}')

# Output the best parameters
print(f'Best Parameters: {gs_rf.best_params_}')
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

```
Fitting 3 folds for each of 486 candidates, totalling 1458 fits
```

```
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    5.5s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   35.0s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  2.5min
[Parallel(n_jobs=-1)]: Done 997 tasks      | elapsed:  3.8min
[Parallel(n_jobs=-1)]: Done 1442 tasks      | elapsed:  4.9min
[Parallel(n_jobs=-1)]: Done 1458 out of 1458 | elapsed:  4.9min finished
```

```
Best Random Forest Model Metrics:
Accuracy: 0.926
F1-Score: 0.748
Confusion Matrix:
 [[816  41]
 [ 33 110]]
Best Parameters: {'clf__max_depth': 20, 'clf__max_features': 'auto',
'clf__min_samples_leaf': 1, 'clf__min_samples_split': 2, 'clf__n_estimators':
200}
```

The tuned random forest model has the best accuracy on the test data so far. The accuracy is at 0.926 compared to the 0.803 of the tuned decision tree. Let's see how a `XGboost model` will perform before we settle on a classifier.

XGBoost Classifier

[25]:
```python
"""
My tuning process:
 - Instatiate a model with the default parameters
 - Apply GridsearchCV to identify the optimal parameters
 - Find the optimal model
 - Score the model by it's accuracy, f1_score and print out a confusion matrix␣
 ↪of the same.
"""
# Construct pipeline for XGBoost
pipe_xgb = Pipeline([('scl', StandardScaler()),
```

```
                    ('clf', XGBClassifier(random_state = 42))])

#Define parameter grid for Grid Search
param_grid = {
    'clf__n_estimators': [50, 100, 200, 300],
    'clf__learning_rate': [0.01, 0.1, 0.2],
    'clf__max_depth': [3, 5, 10, 20, 30, 50],
    'clf__subsample': [0.6, 0.8, 1.0],
    'clf__colsample_bytree': [0.6, 0.8, 1.0]
}

# Perform Grid Search with cross-validation
gs_xgb = GridSearchCV(pipe_xgb, param_grid, cv=3, scoring='f1', verbose=2,␣
 ↪n_jobs=-1)
gs_xgb.fit(X_train_resampled, y_train_resampled)

# Get the best estimator
best_xgb = gs_xgb.best_estimator_

# Predict with the best estimator
y_pred_xgb = best_xgb.predict(X_test)

# Model evaluation
print("Best XGBoost Model Metrics:")
print(f'Accuracy: {accuracy_score(y_test, y_pred_xgb):.3f}')
print(f'F1-Score: {f1_score(y_test, y_pred_xgb):.3f}')
print(f'Confusion Matrix:\n {confusion_matrix(y_test, y_pred_xgb)}')

# Output the best parameters
print(f'Best Parameters: {gs_xgb.best_params_}')
```

```
Fitting 3 folds for each of 648 candidates, totalling 1944 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    1.4s
[Parallel(n_jobs=-1)]: Done 146 tasks      | elapsed:   22.0s
[Parallel(n_jobs=-1)]: Done 349 tasks      | elapsed:   55.4s
[Parallel(n_jobs=-1)]: Done 632 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 997 tasks      | elapsed:  2.8min
[Parallel(n_jobs=-1)]: Done 1442 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 1944 out of 1944 | elapsed:  6.2min finished
```

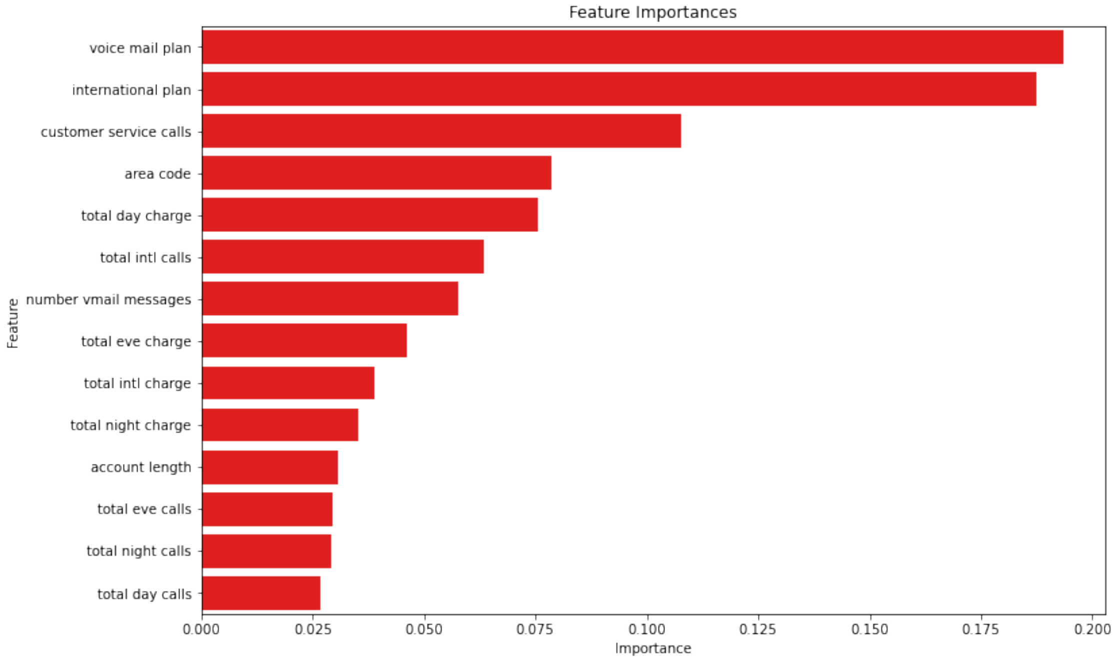
```
Best XGBoost Model Metrics:
Accuracy: 0.933
F1-Score: 0.771
Confusion Matrix:
 [[820  37]
 [ 30 113]]
Best Parameters: {'clf__colsample_bytree': 1.0, 'clf__learning_rate': 0.1,
```

```
'clf__max_depth': 30, 'clf__n_estimators': 200, 'clf__subsample': 0.8}
```

So far, with XGBoost classifier has proven to be the best as expected at an accuracy of 0.933. It will be my recommendation as the best model to predict customer churn rate in SyriaTel.

Feature Importance Now let's look at the most impactful features to customer churn prediction using our preferred model (The XGBoost Classifier).

[26]:
```
# Let's access the model within the pipeline
model = best_xgb.named_steps['clf']

# Let's calculate feature importances
feature_importances = model.feature_importances_

# Creating a list of feature names
feature_list = list(X_train_resampled.columns)

# Creatinf a Panda DataFrame using feature_list as an index
relative_importances = pd.DataFrame(index=feature_list,␣
 ↪data=feature_importances, columns=["importance"])

# Sorting values
relative_importances = relative_importances.sort_values(by="importance",␣
 ↪ascending=False)

# Reseting index to have 'index' as a column
result = relative_importances.reset_index()

# Plot feature importances with sorted values
plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y="index", data=result, color='red')
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```



Voice mail plan, International plan and No. of customer service calls are the 3 most significant factors in predicting customer churn rate in SyriaTel. This revelation coupled with the bivariate analysis of these 3 features against churn will form part of our recommendations to the company and specifically to the 2 main stakeholder departments (Customer service and Sales & Marketing Teams) within.

Conclusions: **Overall Churn Rate:** The customer churn rate for SyriaTel is 14.49%, which is relatively low compared to the industry standard of 30-35%.

Significant Factors Affecting Churn: `International Plan:` Customers with an international plan have a higher churn rate, indicating dissatisfaction with this service. `Customer Service Calls:` Customers who make multiple calls to customer service have a significantly higher chance of churning, suggesting unresolved issues or dissatisfaction with complaint resolution.

Predictive Model: The XGBoost model has proven to be the most effective for predicting customer churn, with an accuracy of 93.3%.

Recommendations: **To The Customer Service Team:**

`Improve on complaint resolution:` Focus on improving the quality of customer service interactions, especially for customers making multiple calls. This could involve training the customer service personnel and creating effective customer service SOPs(Standard Operiting Procedures). Consider proactive outreach to customers who have called multiple times to ensure their issues have been resolved satisfactorily.

`Monitor and address frequent callers:` Identify customers who frequently contact customer service and monitor their interactions closely. Follow-up to ensure their issues are fully resolved.

`Implement a customer feedback system:` Implement a feedback system to gather customer opinions after intering with the customer service team. Use this data to continually improve the quality of service.

To The Sales & Marketing Team:

`Review the International plan:` Reevaluate the international plan. Conduct surveys to understand the specific pain points customers are experiencing with this plan.This should be based on how the Voice mail plan is rolled out since customers with the later tend not to churn. Market both plans more to ensure a higher adoption rate. Offer a revised plan or additional benefits to address common issues and enhance customer satisfaction.

`Targeted marketing campaigns:` Develop targeted campaigns for customers with international plans and frequent customer care callers to address their specific needs and reduce churn. Highlight improvements and new features in marketing materials to reassure existing customers of the value they are receiving.

`Promotional offers and customer sensitization:` Provide special promotions or discounts to customers identified as highly likely to churn. This can include loyalty programs or incentives for long-term commitment. Create educational content to help customers better understand and utilize their plans. This can include tutorials and/or webinars.

Further steps: `Model Deployment:` Explore the possibility of integrating the model to Syria-Tel's systems to allow for seamless access to newly generated data for better churn predictions over time.