phase-3-project-2

May 23, 2024

0.1 Data Science Part Time 06

Student Name : **Lydia Masabarakiza**

0.2 Overview

This project aims to develop a predictive model for SyriaTel, a telecommunications company, to forecast customer churn. Customer churn, where customers cease using the company's services, is a significant issue impacting revenue and growth. By identifying which customers are likely to churn, SyriaTel can implement targeted strategies to improve retention and customer satisfaction.

0.3 Business Understanding

SyriaTel, a prominent telecommunications company, aims to improve customer retention and reduce revenue loss caused by customer churn. Churn occurs when customers stop using the company's services, and it is crucial for SyriaTel to predict which customers are likely to churn soon. By identifying these customers in advance, SyriaTel can implement targeted retention strategies to improve customer satisfaction and loyalty.

0.4 Problem Statement

The objective of this project is to develop a predictive model that can accurately classify whether a customer will stop using SyriaTel's services in the near future. This is a binary classification problem where the target variable indicates churn status: '1' for churn and '0' for non-churn. By analyzing customer data and identifying patterns associated with churn, we can provide actionable insights to SyriaTel for preemptive retention efforts.

Objectives 1. **Develop Predictive Models**: Create models to predict customer churn using the provided dataset.

2. **Analyze Customer Data**: Examine customer demographics, usage patterns, and service interactions to identify churn predictors.

3. **Provide Actionable Insights**: Offer insights and recommendations based on the model's findings to aid SyriaTel in reducing churn.

0.5 1. Data Understanding

1. The dataset includes 3333 records and 21 features, such as customer demographics, account information, usage statistics, and service interactions.

2. **Target Variable**: The target variable is 'churn', indicating whether a customer has churned (False) or not (True).
3. **No Missing Values**: There are no missing values in the dataset.
4. **Feature Types**: The dataset has a mix of categorical (e.g., state, international plan) and numerical (e.g., total day minutes, customer service calls) features.

0.6 2. Data Preparation

[ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix,␣
↪accuracy_score
```

[ ]:
```
# Loading the data set
df = pd.read_csv("/content/bigml_59c28831336c6604c800002a.csv")
df.head()
```

[ ]:
```
  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]
```

[ ]:
```
df.tail()
```

[ ]:
```
     state  account length  area code phone number international plan  \
3328    AZ             192        415     414-4276                 no
3329    WV              68        415     370-3271                 no
3330    RI              28        510     328-8230                 no
3331    CT             184        510     364-6381                yes
3332    TN              74        415     400-4344                 no

     voice mail plan  number vmail messages  total day minutes  \
3328             yes                     36              156.2
3329              no                      0              231.1
3330              no                      0              180.8
3331              no                      0              213.8
3332             yes                     25              234.4

      total day calls  total day charge  …  total eve calls  \
3328               77             26.55  …              126
3329               57             39.29  …               55
3330              109             30.74  …               58
3331              105             36.35  …               84
3332              113             39.85  …               82
```

```
      total eve charge  total night minutes  total night calls  \
3328             18.32                279.1                 83
3329             13.04                191.3                123
3330             24.55                191.9                 91
3331             13.57                139.2                137
3332             22.60                241.4                 77

      total night charge  total intl minutes  total intl calls  \
3328               12.56                 9.9                 6
3329                8.61                 9.6                 4
3330                8.64                14.1                 6
3331                6.26                 5.0                10
3332               10.86                13.7                 4

      total intl charge  customer service calls  churn
3328               2.67                       2  False
3329               2.59                       3  False
3330               3.81                       2  False
3331               1.35                       2  False
3332               3.70                       0  False

[5 rows x 21 columns]
```

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
```

```
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

[ ]:
```
# Checking missing values
df.isnull().sum()
```

[ ]:
```
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
```

[ ]:
```
df.dtypes
```

[ ]:
```
state                      object
account length              int64
area code                   int64
phone number               object
international plan         object
voice mail plan            object
number vmail messages       int64
total day minutes         float64
total day calls             int64
total day charge          float64
total eve minutes         float64
total eve calls             int64
```

```
total eve charge          float64
total night minutes       float64
total night calls           int64
total night charge        float64
total intl minutes        float64
total intl calls            int64
total intl charge         float64
customer service calls      int64
churn                        bool
dtype: object
```

[ ]:
```
unique_values = df.nunique()
print("\nNumber of unique values in each column:")
print(unique_values)
```

```
Number of unique values in each column:
state                       51
account length             212
area code                    3
phone number              3333
international plan           2
voice mail plan              2
number vmail messages       46
total day minutes         1667
total day calls            119
total day charge          1667
total eve minutes         1611
total eve calls            123
total eve charge          1440
total night minutes       1591
total night calls          120
total night charge         933
total intl minutes         162
total intl calls            21
total intl charge          162
customer service calls      10
churn                        2
dtype: int64
```

[ ]:
```
for column in df.columns:
    if pd.api.types.is_numeric_dtype(df[column]):
        unique_values = df[column].unique()
        print(f"Unique values in numeric column '{column}': {unique_values}")
```

```
Unique values in numeric column 'account length': [128 107 137  84  75 118 121
147 117 141  65  74 168  95  62 161  85  93
  76  73  77 130 111 132 174  57  54  20  49 142 172  12  72  36  78 136
```

```
 149  98 135  34 160  64  59 119  97  52  60  10  96  87  81  68 125 116
  38  40  43 113 126 150 138 162  90  50  82 144  46  70  55 106  94 155
  80 104  99 120 108 122 157 103  63 112  41 193  61  92 131 163  91 127
 110 140  83 145  56 151 139   6 115 146 185 148  32  25 179  67  19 170
 164  51 208  53 105  66  86  35  88 123  45 100 215  22  33 114  24 101
 143  48  71 167  89 199 166 158 196 209  16  39 173 129  44  79  31 124
  37 159 194 154  21 133 224  58  11 109 102 165  18  30 176  47 190 152
  26  69 186 171  28 153 169  13  27   3  42 189 156 134 243  23   1 205
 200   5   9 178 181 182 217 177 210  29 180   2  17   7 212 232 192 195
 197 225 184 191 201  15 183 202   8 175   4 188 204 221]
Unique values in numeric column 'area code': [415 408 510]
Unique values in numeric column 'number vmail messages': [25 26  0 24 37 27 33
39 30 41 28 34 46 29 35 21 32 42 36 22 23 43 31 38
 40 48 18 17 45 16 20 14 19 51 15 11 12 47  8 44 49  4 10 13 50  9]
Unique values in numeric column 'total day minutes': [265.1 161.6 243.4 …
321.1 231.1 180.8]
Unique values in numeric column 'total day calls': [110 123 114  71 113  98  88
79  97  84 137 127  96  70  67 139  66  90
 117  89 112 103  86  76 115  73 109  95 105 121 118  94  80 128  64 106
 102  85  82  77 120 133 135 108  57  83 129  91  92  74  93 101 146  72
  99 104 125  61 100  87 131  65 124 119  52  68 107  47 116 151 126 122
 111 145  78 136 140 148  81  55  69 158 134 130  63  53  75 141 163  59
 132 138  54  58  62 144 143 147  36  40 150  56  51 165  30  48  60  42
   0  45 160 149 152 142 156  35  49 157  44]
Unique values in numeric column 'total day charge': [45.07 27.47 41.38 … 54.59
39.29 30.74]
Unique values in numeric column 'total eve minutes': [197.4 195.5 121.2 …
153.4 288.8 265.9]
Unique values in numeric column 'total eve calls': [ 99 103 110  88 122 101 108
94  80 111  83 148  71  75  76  97  90  65
  93 121 102  72 112 100  84 109  63 107 115 119 116  92  85  98 118  74
 117  58  96  66  67  62  77 164 126 142  64 104  79  95  86 105  81 113
 106  59  48  82  87 123 114 140 128  60  78 125  91  46 138 129  89 133
 136  57 135 139  51  70 151 137 134  73 152 168  68 120  69 127 132 143
  61 124  42  54 131  52 149  56  37 130  49 146 147  55  12  50 157 155
  45 144  36 156  53 141  44 153 154 150  43   0 145 159 170]
Unique values in numeric column 'total eve charge': [16.78 16.62 10.3  … 13.04
24.55 22.6 ]
Unique values in numeric column 'total night minutes': [244.7 254.4 162.6 …
280.9 120.1 279.1]
Unique values in numeric column 'total night calls': [ 91 103 104  89 121 118
96  90  97 111  94 128 115  99  75 108  74 133
  64  78 105  68 102 148  98 116  71 109 107 135  92  86 127  79  87 129
  57  77  95  54 106  53  67 139  60 100  61  73 113  76 119  88  84  62
 137  72 142 114 126 122  81 123 117  82  80 120 130 134  59 112 132 110
 101 150  69 131  83  93 124 136 125  66 143  58  55  85  56  70  46  42
 152  44 145  50 153  49 175  63 138 154 140 141 146  65  51 151 158 155
 157 147 144 149 166  52  33 156  38  36  48 164]
```

```
Unique values in numeric column 'total night charge': [11.01 11.45  7.32  8.86
8.41  9.18  9.57  9.53  9.71 14.69  9.4   8.82
  6.35  8.65  9.14  7.23  4.02  5.83  7.46  8.68  9.43  8.18  8.53 10.67
 11.28  8.22  4.59  8.17  8.04 11.27 11.08 13.2  12.61  9.61  6.88  5.82
 10.25  4.58  8.47  8.45  5.5  14.02  8.03 11.94  7.34  6.06 10.9   6.44
  3.18 10.66 11.21 12.73 10.28 12.16  6.34  8.15  5.84  8.52  7.5   7.48
  6.21 11.95  7.15  9.63  7.1   6.91  6.69 13.29 11.46  7.76  6.86  8.16
 12.15  7.79  7.99 10.29 10.08 12.53  7.91 10.02  8.61 14.54  8.21  9.09
  4.93 11.39 11.88  5.75  7.83  8.59  7.52 12.38  7.21  5.81  8.1  11.04
 11.19  8.55  8.42  9.76  9.87 10.86  5.36 10.03 11.15  9.51  6.22  2.59
  7.65  6.45  9.    6.4   9.94  5.08 10.23 11.36  6.97 10.16  7.88 11.91
  6.61 11.55 11.76  9.27  9.29 11.12 10.69  8.8  11.85  7.14  8.71 11.42
  4.94  9.02 11.22  4.97  9.15  5.45  7.27 12.91  7.75 13.46  6.32 12.13
 11.97  6.93 11.66  7.42  6.19 11.41 10.33 10.65 11.92  4.77  4.38  7.41
 12.1   7.69  8.78  9.36  9.05 12.7   6.16  6.05 10.85  8.93  3.48 10.4
  5.05 10.71  9.37  6.75  8.12 11.77 11.49 11.06 11.25 11.03 10.82  8.91
  8.57  8.09 10.05 11.7  10.17  8.74  5.51 11.11  3.29 10.13  6.8   8.49
  9.55 11.02  9.91  7.84 10.62  9.97  3.44  7.35  9.79  8.89  8.14  6.94
 10.49 10.57 10.2   6.29  8.79 10.04 12.41 15.97  9.1  11.78 12.75 11.07
 12.56  8.63  8.02 10.42  8.7   9.98  7.62  8.33  6.59 13.12 10.46  6.63
  8.32  9.04  9.28 10.76  9.64 11.44  6.48 10.81 12.66 11.34  8.75 13.05
 11.48 14.04 13.47  5.63  6.6   9.72 11.68  6.41  9.32 12.95 13.37  9.62
  6.03  8.25  8.26 11.96  9.9   9.23  5.58  7.22  6.64 12.29 12.93 11.32
  6.85  8.88  7.03  8.48  3.59  5.86  6.23  7.61  7.66 13.63  7.9  11.82
  7.47  6.08  8.4   5.74 10.94 10.35 10.68  4.34  8.73  5.14  8.24  9.99
 13.93  8.64 11.43  5.79  9.2  10.14 12.11  7.53 12.46  8.46  8.95  9.84
 10.8  11.23 10.15  9.21 14.46  6.67 12.83  9.66  9.59 10.48  8.36  4.84
 10.54  8.39  7.43  9.06  8.94 11.13  8.87  8.5   7.6  10.73  9.56 10.77
  7.73  3.47 11.86  8.11  9.78  9.42  9.65  7.    7.39  9.88  6.56  5.92
  6.95 15.71  8.06  4.86  7.8   8.58 10.06  5.21  6.92  6.15 13.49  9.38
 12.62 12.26  8.19 11.65 11.62 10.83  7.92  7.33 13.01 13.26 12.22 11.58
  5.97 10.99  8.38  9.17  8.08  5.71  3.41 12.63 11.79 12.96  7.64  6.58
 10.84 10.22  6.52  5.55  7.63  5.11  5.89 10.78  3.05 11.89  8.97 10.44
 10.5   9.35  5.66 11.09  9.83  5.44 10.11  6.39 11.93  8.62 12.06  6.02
  8.85  5.25  8.66  6.73 10.21 11.59 13.87  7.77 10.39  5.54  6.62 13.33
  6.24 12.59  6.3   6.79  8.28  9.03  8.07  5.52 12.14 10.59  7.54  7.67
  5.47  8.81  8.51 13.45  8.77  6.43 12.01 12.08  7.07  6.51  6.84  9.48
 13.78 11.54 11.67  8.13 10.79  7.13  4.72  4.64  8.96 13.03  6.07  3.51
  6.83  6.12  9.31  9.58  4.68  5.32  9.26 11.52  9.11 10.55 11.47  9.3
 13.82  8.44  5.77 10.96 11.74  8.9  10.47  7.85 10.92  4.74  9.74 10.43
  9.96 10.18  9.54  7.89 12.36  8.54 10.07  9.46  7.3  11.16  9.16 10.19
  5.99 10.88  5.8   7.19  4.55  8.31  8.01 14.43  8.3  14.3   6.53  8.2
 11.31 13.    6.42  4.24  7.44  7.51 13.1   9.49  6.14  8.76  6.65 10.56
  6.72  8.29 12.09  5.39  2.96  7.59  7.24  4.28  9.7   8.83 13.3  11.37
  9.33  5.01  3.26 11.71  8.43  9.68 15.56  9.8   3.61  6.96 11.61 12.81
 10.87 13.84  5.03  5.17  2.03 10.34  9.34  7.95 10.09  9.95  7.11  9.22
  6.13 11.05  9.89  9.39 14.06 10.26 13.31 15.43 16.39  6.27 10.64 11.5
 12.48  8.27 13.53 10.36 12.24  8.69 10.52  9.07 11.51  9.25  8.72  6.78
```

```
  8.6  11.84  5.78  5.85 12.3   5.76 12.07  9.6   8.84 12.39 10.1   9.73
  2.85  6.66  2.45  5.28 11.73 10.75  7.74  6.76  6.    7.58 13.69  7.93
  7.68  9.75  4.96  5.49 11.83  7.18  9.19  7.7   7.25 10.74  4.27 13.8
  9.12  4.75  7.78 11.63  7.55  2.25  9.45  9.86  7.71  4.95  7.4  11.17
 11.33  6.82 13.7   1.97 10.89 12.77 10.31  5.23  5.27  9.41  6.09 10.61
  7.29  4.23  7.57  3.67 12.69 14.5   5.95  7.87  5.96  5.94 12.23  4.9
 12.33  6.89  9.67 12.68 12.87  3.7   6.04 13.13 15.74 11.87  4.7   4.67
  7.05  5.42  4.09  5.73  9.47  8.05  6.87  3.71 15.86  7.49 11.69  6.46
 10.45 12.9   5.41 11.26  1.04  6.49  6.37 12.21  6.77 12.65  7.86  9.44
  4.3   7.38  5.02 10.63  2.86 17.19  8.67  8.37  6.9  10.93 10.38  7.36
 10.27 10.95  6.11  4.45 11.9  15.01 12.84  7.45  6.98 11.72  7.56 11.38
 10.    4.42  9.81  5.56  6.01 10.12 12.4  16.99  5.68 11.64  3.78  7.82
  9.85 13.74 12.71 10.98 10.01  9.52  7.31  8.35 11.35  9.5  14.03  3.2
  7.72 13.22 10.7   8.99 10.6  13.02  9.77 12.58 12.35 12.2  11.4  13.91
  3.57 14.65 12.28  5.13 10.72 12.86 14.    7.12 12.17  4.71  6.28  8.
  7.01  5.91  5.2  12.   12.02 12.88  7.28  5.4  12.04  5.24 10.3  10.41
 13.41 12.72  9.08  7.08 13.5   5.35 12.45  5.3  10.32  5.15 12.67  5.22
  5.57  3.94  4.41 13.27 10.24  4.25 12.89  5.72 12.5  11.29  3.25 11.53
  9.82  7.26  4.1  10.37  4.98  6.74 12.52 14.56  8.34  3.82  3.86 13.97
 11.57  6.5  13.58 14.32 13.75 11.14 14.18  9.13  4.46  4.83  9.69 14.13
  7.16  7.98 13.66 14.78 11.2   9.93 11.    5.29  9.92  4.29 11.1  10.51
 12.49  4.04 12.94  7.09  6.71  7.94  5.31  5.98  7.2  14.82 13.21 12.32
 10.58  4.92  6.2   4.47 11.98  6.18  7.81  4.54  5.37  7.17  5.33 14.1
  5.7  12.18  8.98  5.1  14.67 13.95 16.55 11.18  4.44  4.73  2.55  6.31
  2.43  9.24  7.37 13.42 12.42 11.8  14.45  2.89 13.23 12.6  13.18 12.19
 14.81  6.55 11.3  12.27 13.98  8.23 15.49  6.47 13.48 13.59 13.25 17.77
 13.9   3.97 11.56 14.08 13.6   6.26  4.61 12.76 15.76  6.38  3.6  12.8
  5.9   7.97  5.   10.97  5.88 12.34 12.03 14.97 15.06 12.85  6.54 11.24
 12.64  7.06  5.38 13.14  3.99  3.32  4.51  4.12  3.93  2.4  11.75  4.03
 15.85  6.81 14.25 14.09 16.42  6.7  12.74  2.76 12.12  6.99  6.68 11.81
  7.96  5.06 13.16  2.13 13.17  5.12  5.65 12.37 10.53]
Unique values in numeric column 'total intl minutes': [10.  13.7 12.2  6.6 10.1
6.3  7.5  7.1  8.7 11.2 12.7  9.1 12.3 13.1
  5.4 13.8  8.1 13.  10.6  5.7  9.5  7.7 10.3 15.5 14.7 11.1 14.2 12.6
 11.8  8.3 14.5 10.5  9.4 14.6  9.2  3.5  8.5 13.2  7.4  8.8 11.   7.8
  6.8 11.4  9.3  9.7 10.2  8.   5.8 12.1 12.  11.6  8.2  6.2  7.3  6.1
 11.7 15.   9.8 12.4  8.6 10.9 13.9  8.9  7.9  5.3  4.4 12.5 11.3  9.
  9.6 13.3 20.   7.2  6.4 14.1 14.3  6.9 11.5 15.8 12.8 16.2  0.  11.9
  9.9  8.4 10.8 13.4 10.7 17.6  4.7  2.7 13.5 12.9 14.4 10.4  6.7 15.4
  4.5  6.5 15.6  5.9 18.9  7.6  5.   7.  14.  18.  16.  14.8  3.7  2.
  4.8 15.3  6.  13.6 17.2 17.5  5.6 18.2  3.6 16.5  4.6  5.1  4.1 16.3
 14.9 16.4 16.7  1.3 15.2 15.1 15.9  5.5 16.1  4.  16.9  5.2  4.2 15.7
 17.   3.9  3.8  2.2 17.1  4.9 17.9 17.3 18.4 17.8  4.3  2.9  3.1  3.3
  2.6  3.4  1.1 18.3 16.6  2.1  2.4  2.5]
Unique values in numeric column 'total intl calls': [ 3  5  7  6  4  2  9 19  1
10 15  8 11  0 12 13 18 14 16 20 17]
Unique values in numeric column 'total intl charge': [2.7  3.7  3.29 1.78 2.73
1.7  2.03 1.92 2.35 3.02 3.43 2.46 3.32 3.54
```

```
 1.46 3.73 2.19 3.51 2.86 1.54 2.57 2.08 2.78 4.19 3.97 3.   3.83 3.4
 3.19 2.24 3.92 2.84 2.54 3.94 2.48 0.95 2.3  3.56 2.   2.38 2.97 2.11
 1.84 3.08 2.51 2.62 2.75 2.16 1.57 3.27 3.24 3.13 2.21 1.67 1.97 1.65
 3.16 4.05 2.65 3.35 2.32 2.94 3.75 2.4  2.13 1.43 1.19 3.38 3.05 2.43
 2.59 3.59 5.4  1.94 1.73 3.81 3.86 1.86 3.11 4.27 3.46 4.37 0.   3.21
 2.67 2.27 2.92 3.62 2.89 4.75 1.27 0.73 3.65 3.48 3.89 2.81 1.81 4.16
 1.22 1.76 4.21 1.59 5.1  2.05 1.35 1.89 3.78 4.86 4.32 4.   1.   0.54
 1.3  4.13 1.62 3.67 4.64 4.73 1.51 4.91 0.97 4.46 1.24 1.38 1.11 4.4
 4.02 4.43 4.51 0.35 4.1  4.08 4.29 1.49 4.35 1.08 4.56 1.4  1.13 4.24
 4.59 1.05 1.03 0.59 4.62 1.32 4.83 4.67 4.97 4.81 1.16 0.78 0.84 0.89
 0.7  0.92 0.3  4.94 4.48 0.57 0.65 0.68]
Unique values in numeric column 'customer service calls': [1 0 2 3 4 5 7 9 6 8]
Unique values in numeric column 'churn': [False  True]
```

[ ]:
```
df.duplicated().sum()
```

[ ]:
```
0
```

There are no duplicates in the data set

[ ]:
```
df.shape
```

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

The data set has 3333 rows and 21 columns

0.7 3. Exploratory Data Analysis

A. Descriptive Statistics

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

[ ]:
```
       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
```

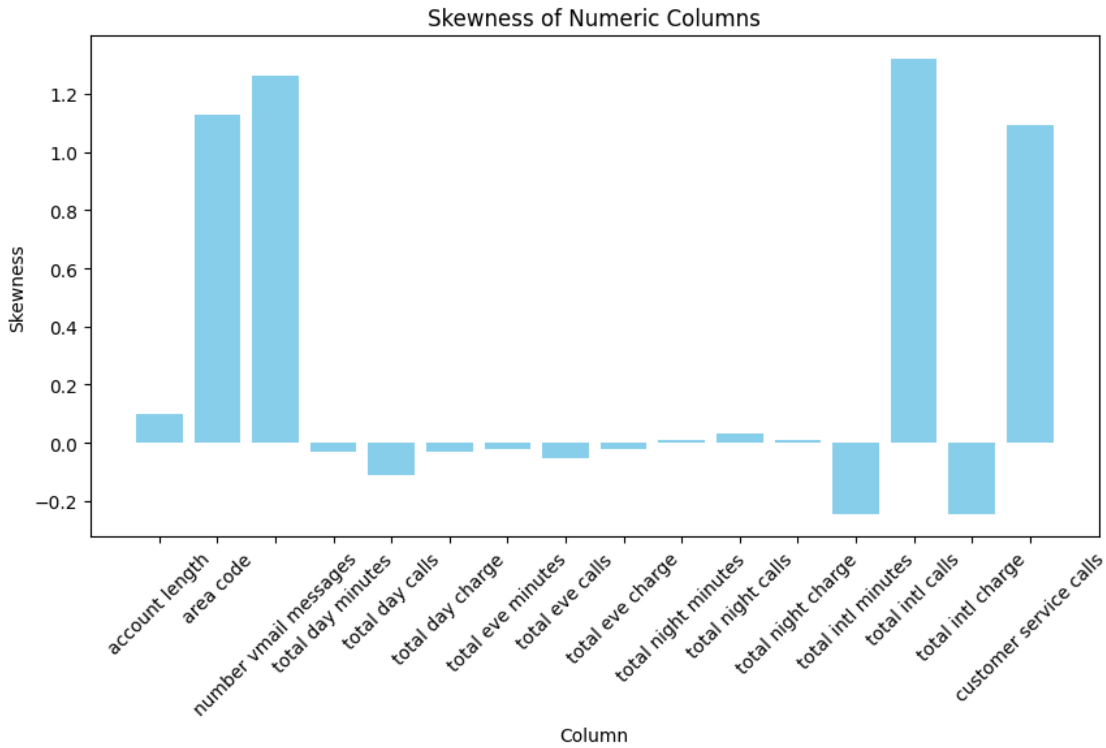
The output shows the summary statistics of the data

[ ]:
```python
# Skewness and Kurtosis
# Filter numeric columns
numeric_columns = df.select_dtypes(include=['number'])
skewness = numeric_columns.skew()
kurtosis = numeric_columns.kurt()
print("Skewness:\n", skewness)
print("Kurtosis:\n", kurtosis)
```

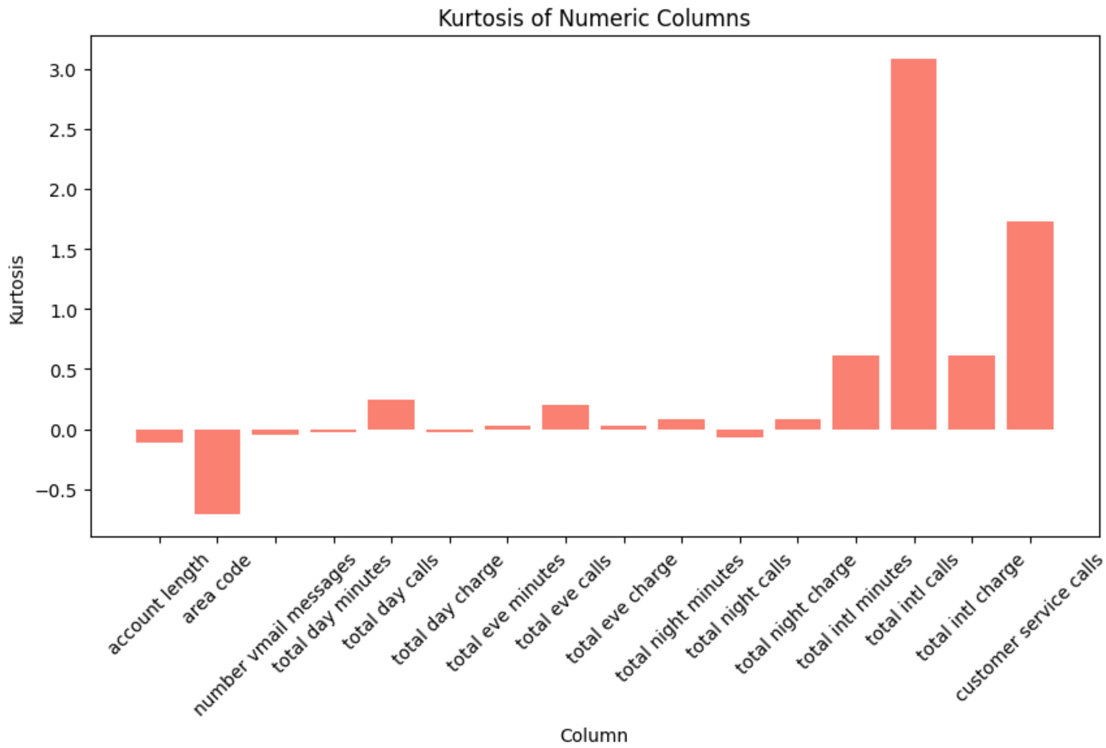
```
Skewness:
 account length            0.096606
area code                 1.126823
number vmail messages     1.264824
total day minutes        -0.029077
```

```
total day calls          -0.111787
total day charge         -0.029083
total eve minutes        -0.023877
total eve calls          -0.055563
total eve charge         -0.023858
total night minutes       0.008921
total night calls         0.032500
total night charge        0.008886
total intl minutes       -0.245136
total intl calls          1.321478
total intl charge        -0.245287
customer service calls    1.091359
dtype: float64
Kurtosis:
 account length           -0.107836
area code                -0.705632
number vmail messages    -0.051129
total day minutes        -0.019940
total day calls           0.243182
total day charge         -0.019812
total eve minutes         0.025630
total eve calls           0.206156
total eve charge          0.025487
total night minutes       0.085816
total night calls        -0.072020
total night charge        0.085663
total intl minutes        0.609185
total intl calls          3.083589
total intl charge         0.609610
customer service calls    1.730914
dtype: float64
```

[ ]:
```python
# Plotting skewness
plt.figure(figsize=(10, 5))
plt.bar(skewness.index, skewness.values, color='skyblue')
plt.title('Skewness of Numeric Columns')
plt.xlabel('Column')
plt.ylabel('Skewness')
plt.xticks(rotation=45)
plt.show()
```

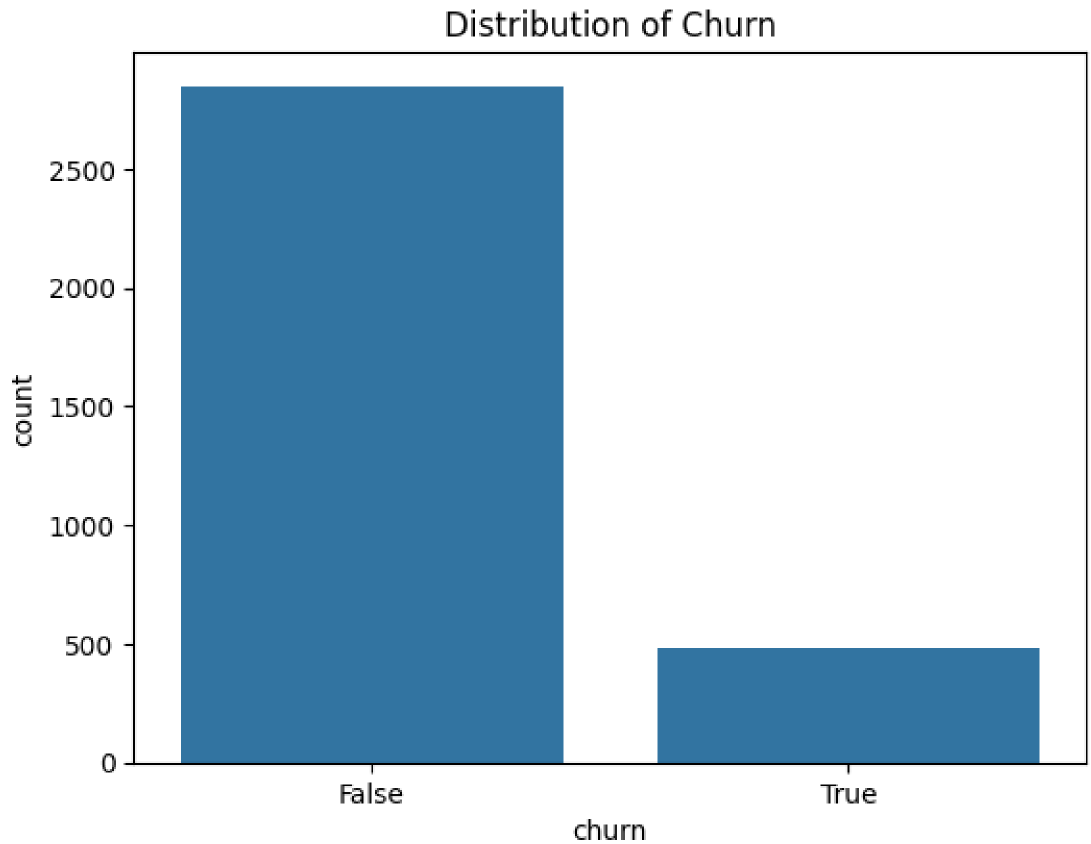


[ ]:
```python
# Plotting kurtosis
plt.figure(figsize=(10, 5))
plt.bar(kurtosis.index, kurtosis.values, color='salmon')
plt.title('Kurtosis of Numeric Columns')
plt.xlabel('Column')
plt.ylabel('Kurtosis')
plt.xticks(rotation=45)
plt.show()
```



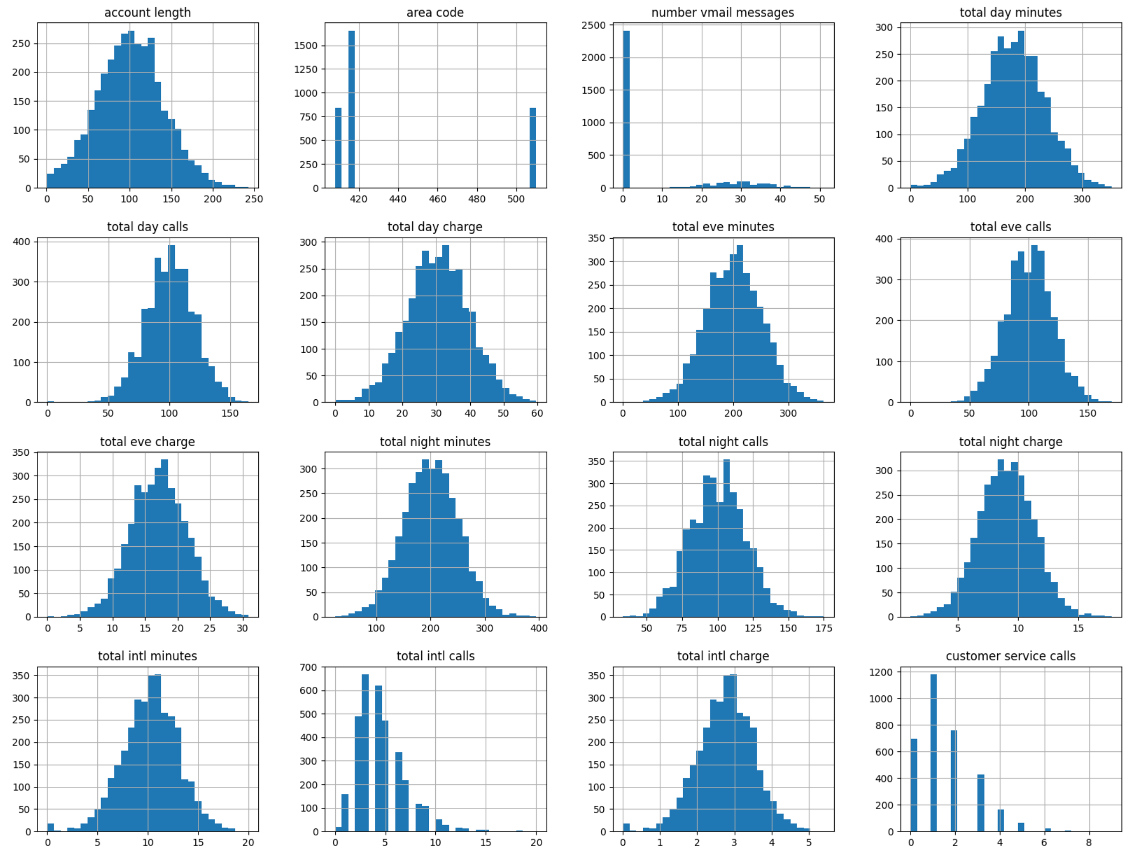
B. Univariate Analysis

```
[ ]: # Target variable distribution
sns.countplot(x='churn', data=df)
plt.title('Distribution of Churn')
plt.show()
```



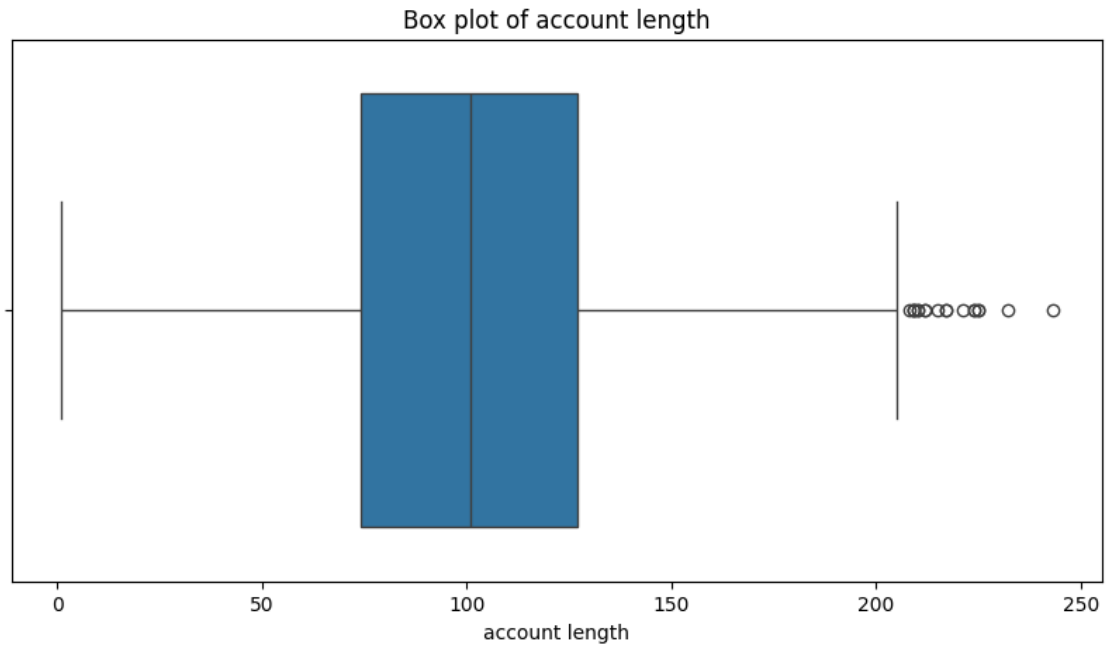
There are more users that have stayed with the company than those that have left.

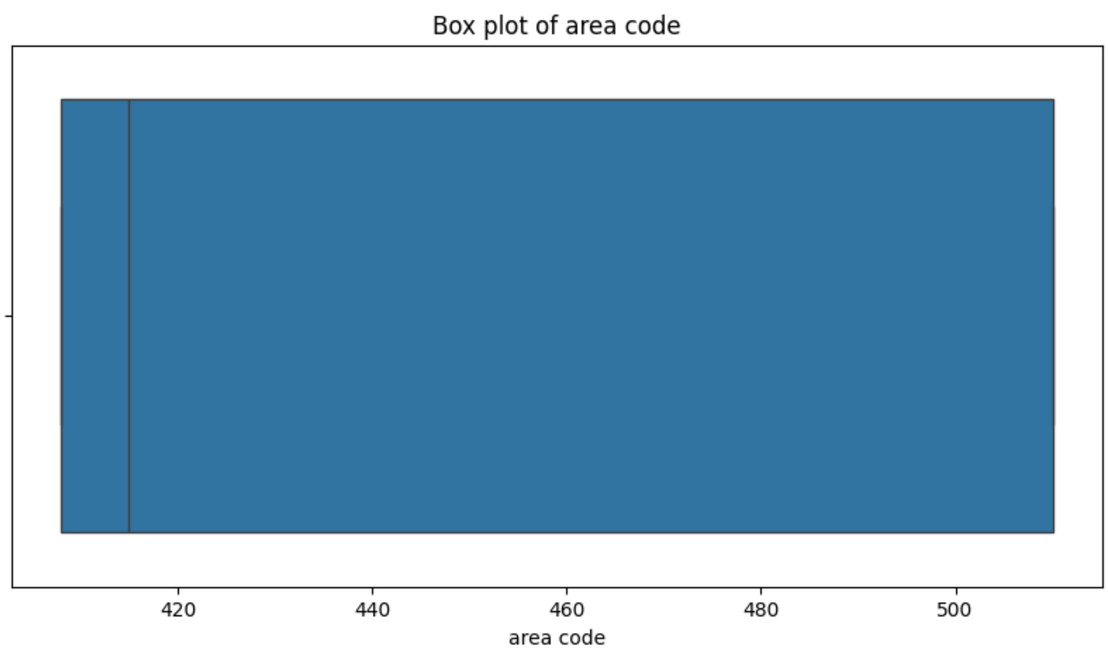
[ ]:
```
# Numeric Features Distribution
df.hist(bins=30, figsize=(20, 15))
plt.show()
```

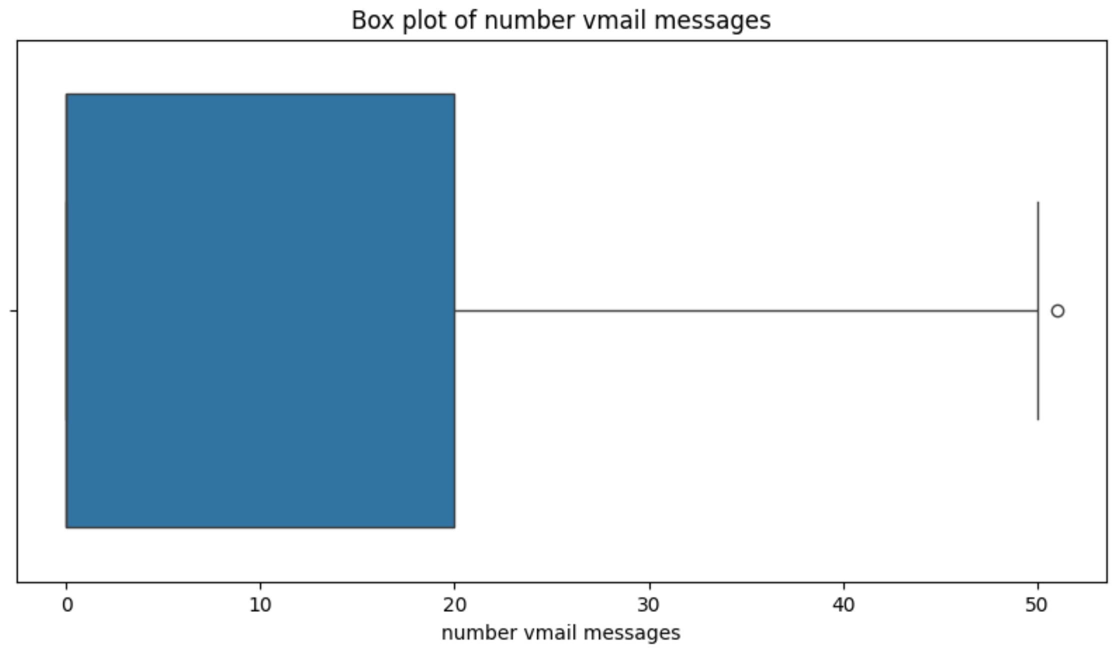


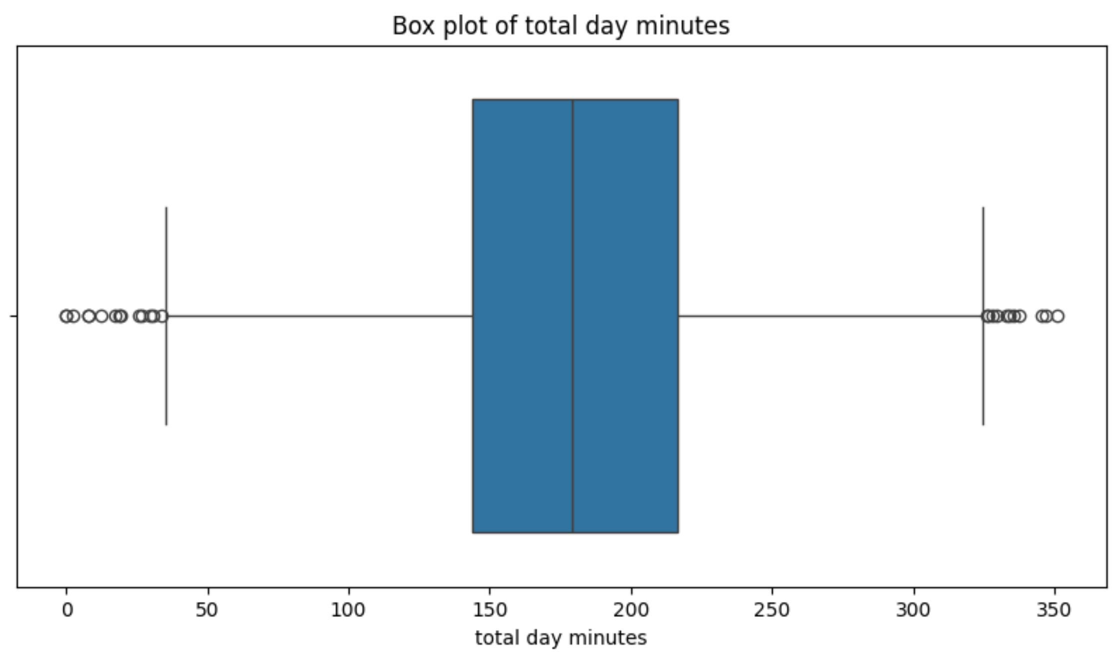
All the columns have a normal distrubution except from total international calls, customer service calls and area code.

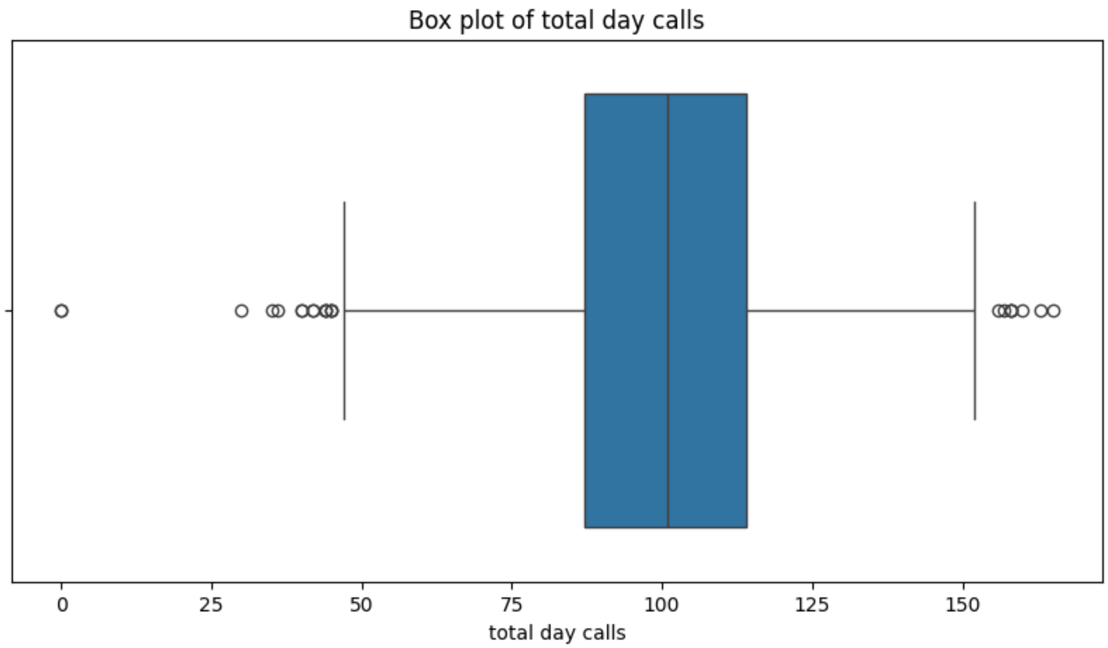
[ ]:
```
# Boxplots for numeric variables
for column in df.select_dtypes(include=['float64', 'int64']).columns:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x=df[column])
    plt.title(f'Box plot of {column}')
    plt.show()
```

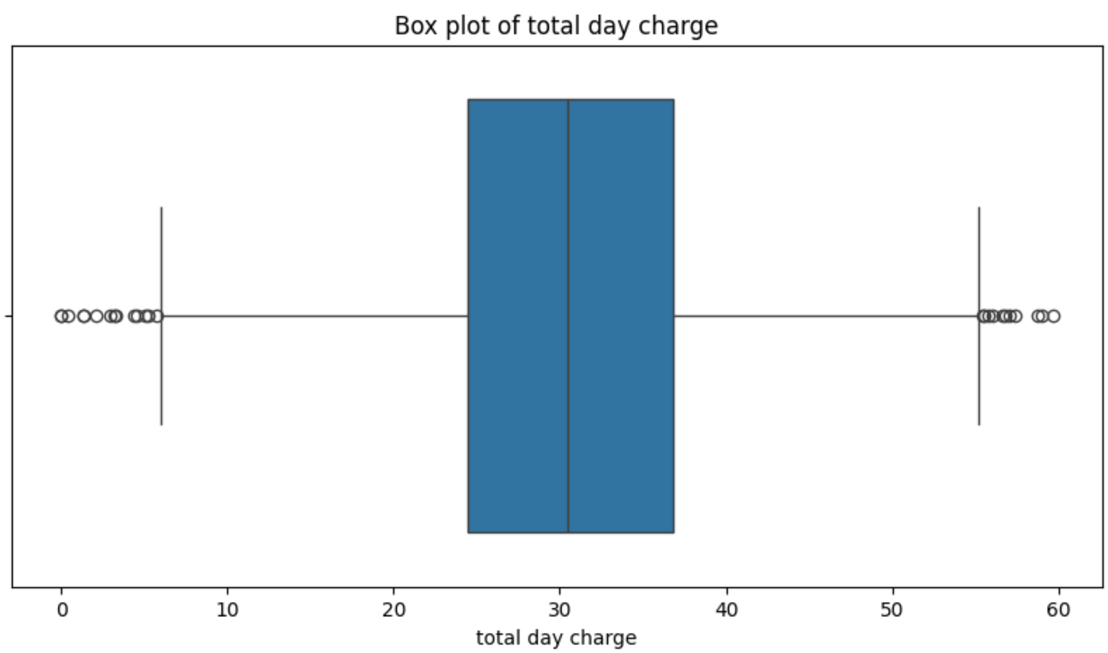

Box plot of account length
0
50
100
150
200
250
account length

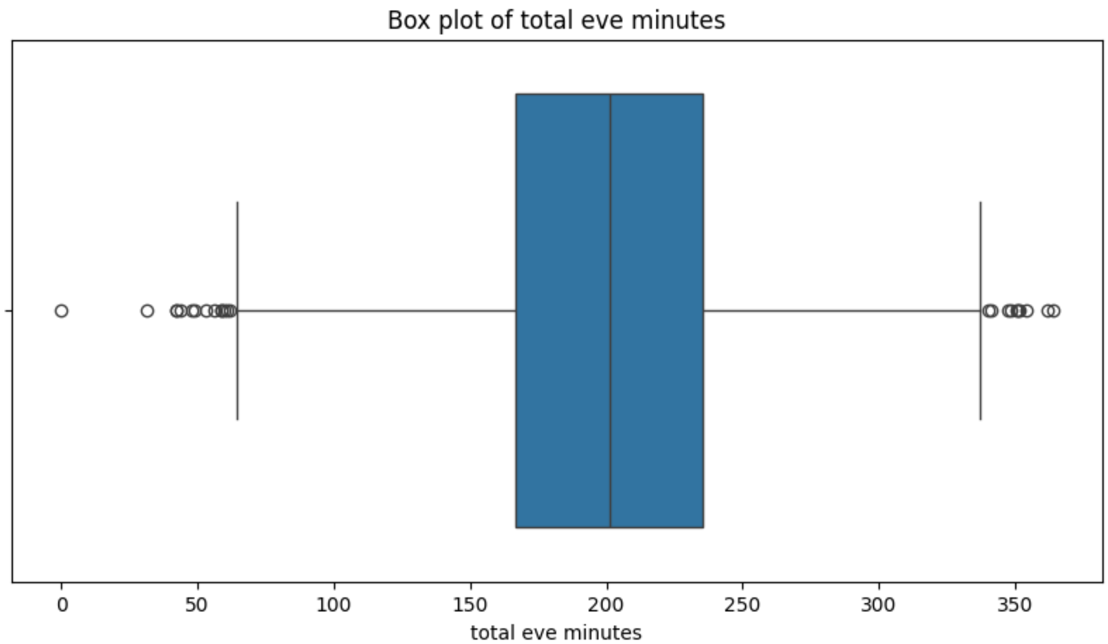

Box plot of area code
420
440
460
480
500
area code

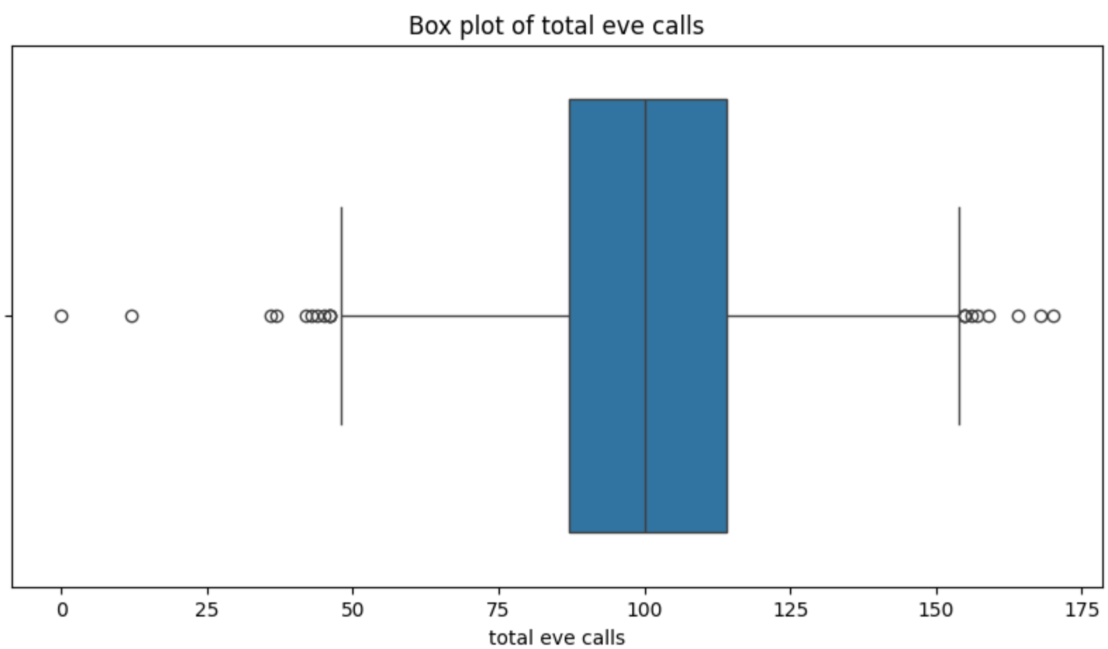

Box plot of number vmail messages
0
10
20
30
40
50
number vmail messages

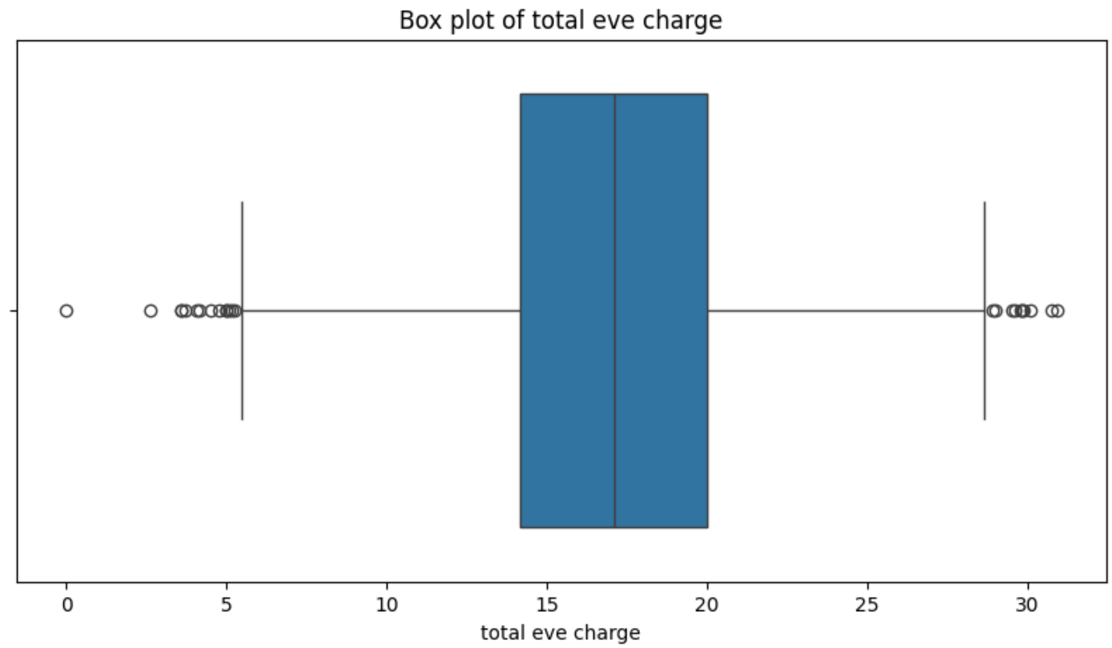

Box plot of total day minutes
0
50
100
150
200
250
300
350
total day minutes

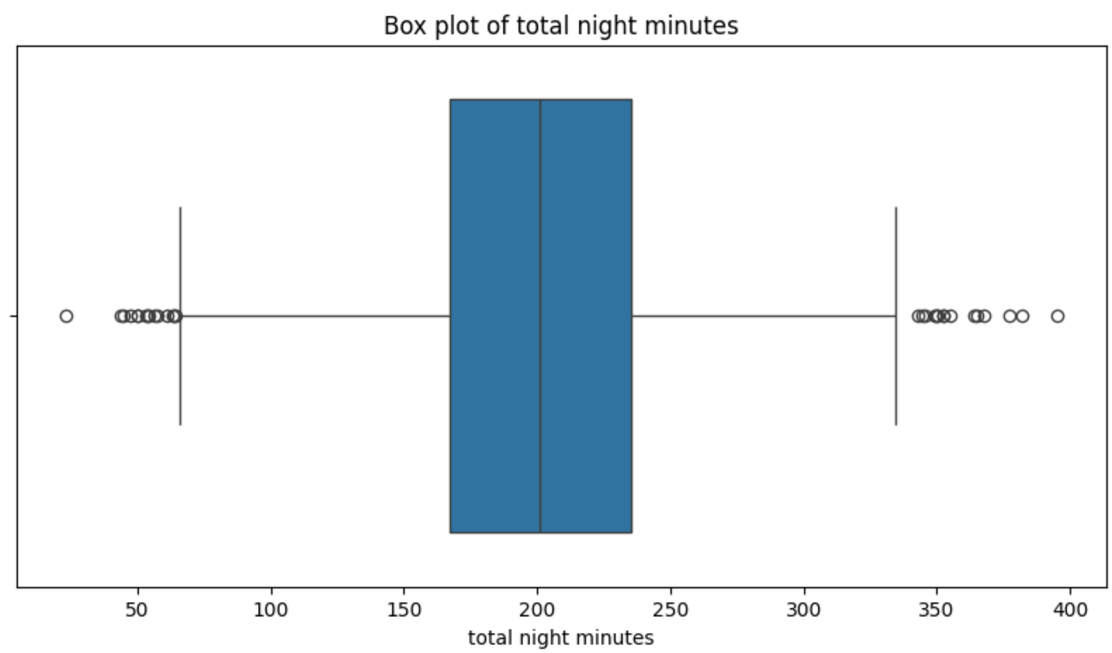

Box plot of total day calls
0
25
50
75
100
125
150
total day calls

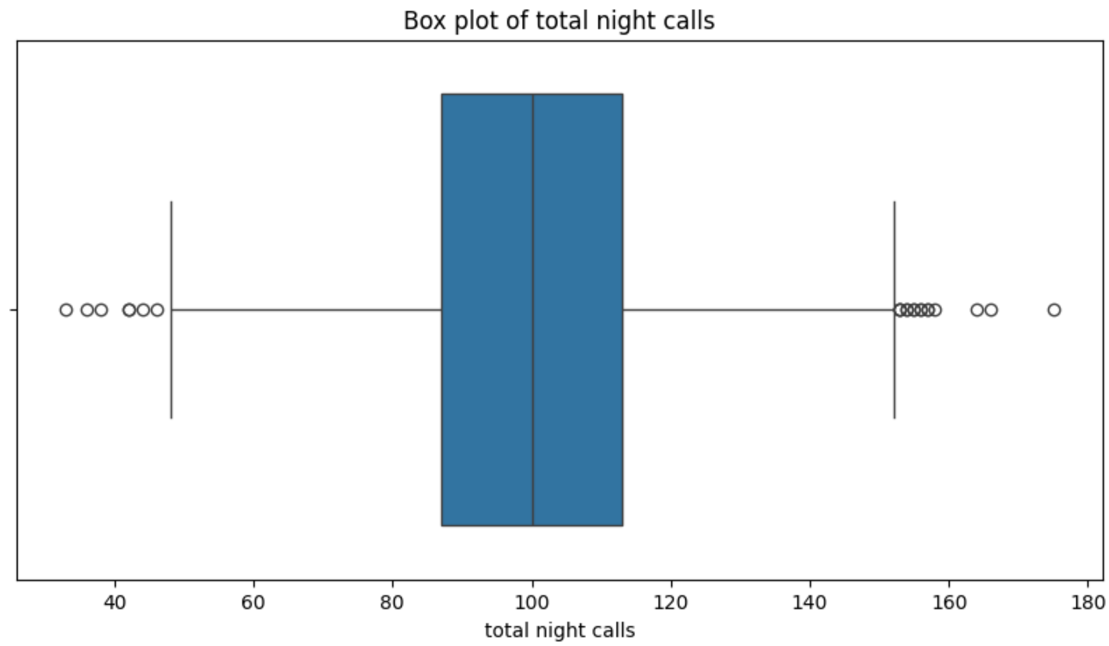

Box plot of total day charge
0
10
20
30
40
50
60
total day charge

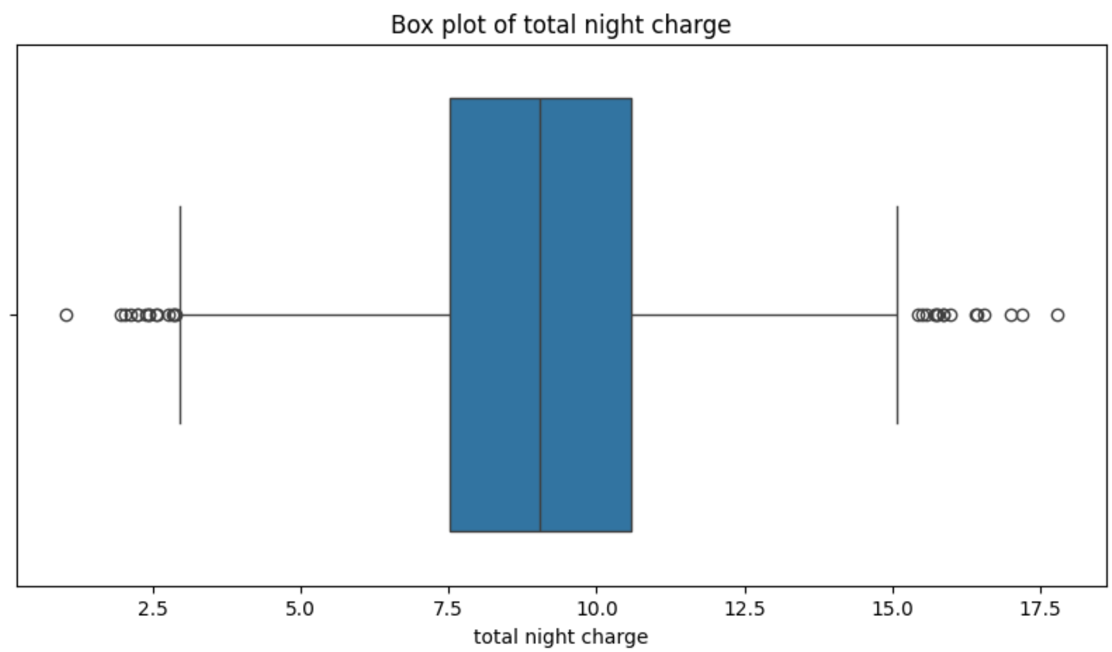

Box plot of total eve minutes
0
50
100
150
200
250
300
350
total eve minutes

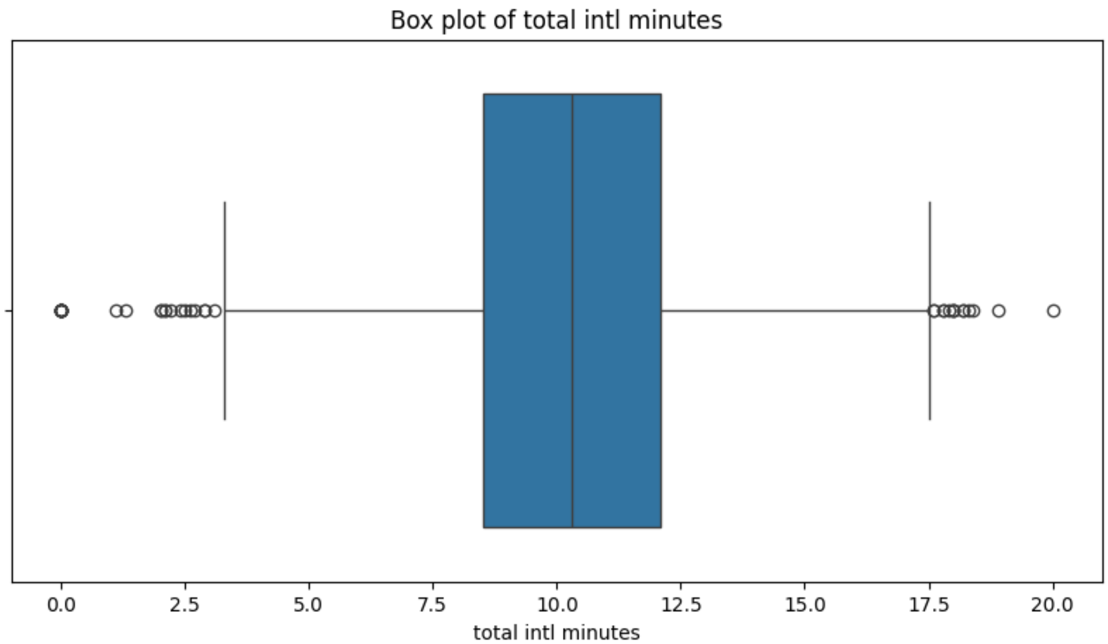

Box plot of total eve calls
0
25
50
75
100
125
150
175
total eve calls

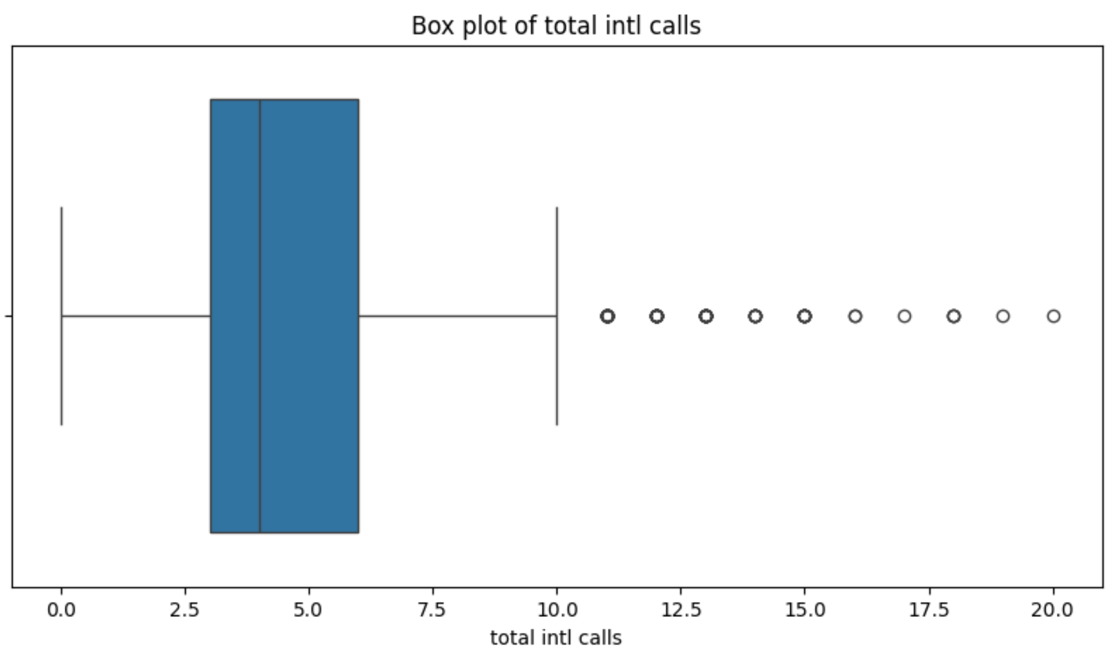

Box plot of total eve charge
0
5
10
15
20
25
30
total eve charge

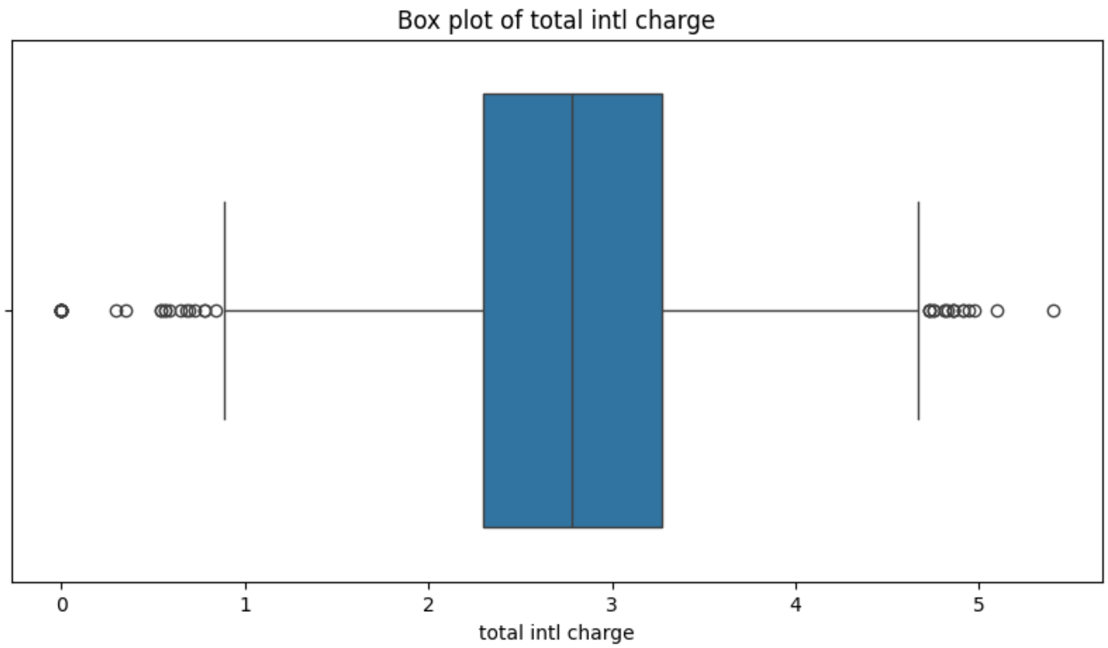

Box plot of total night minutes
50
100
150
200
250
300
350
400
total night minutes

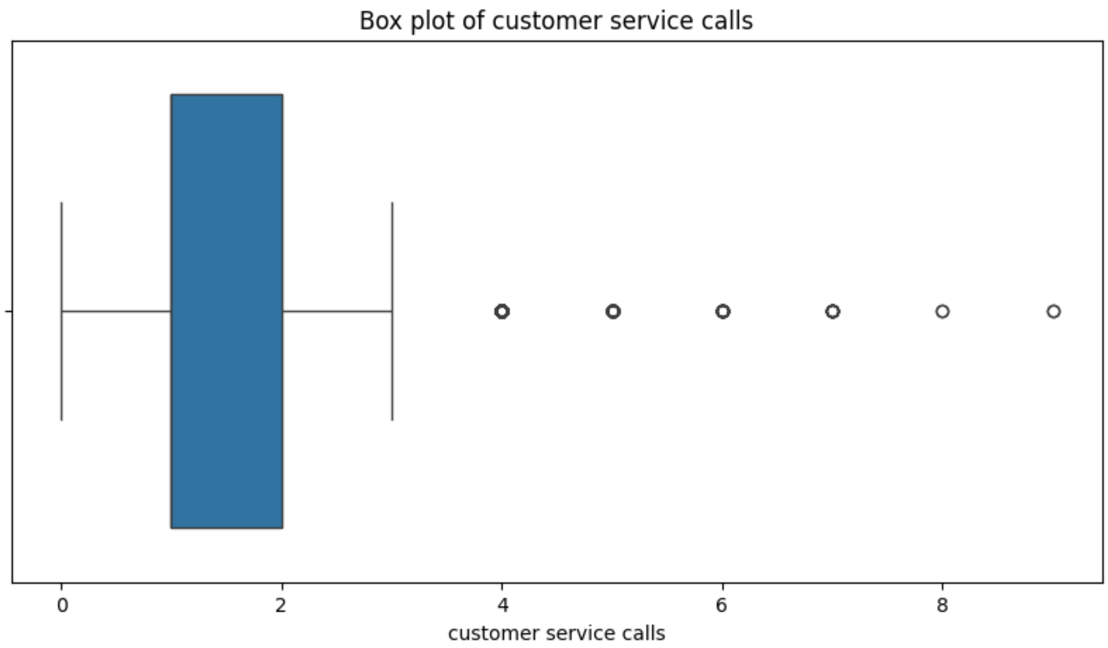

Box plot of total night calls
40
60
80
100
120
140
160
180
total night calls


Box plot of total night charge
2.5
5.0
7.5
10.0
12.5
15.0
17.5
total night charge


Box plot of total intl minutes
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
total intl minutes

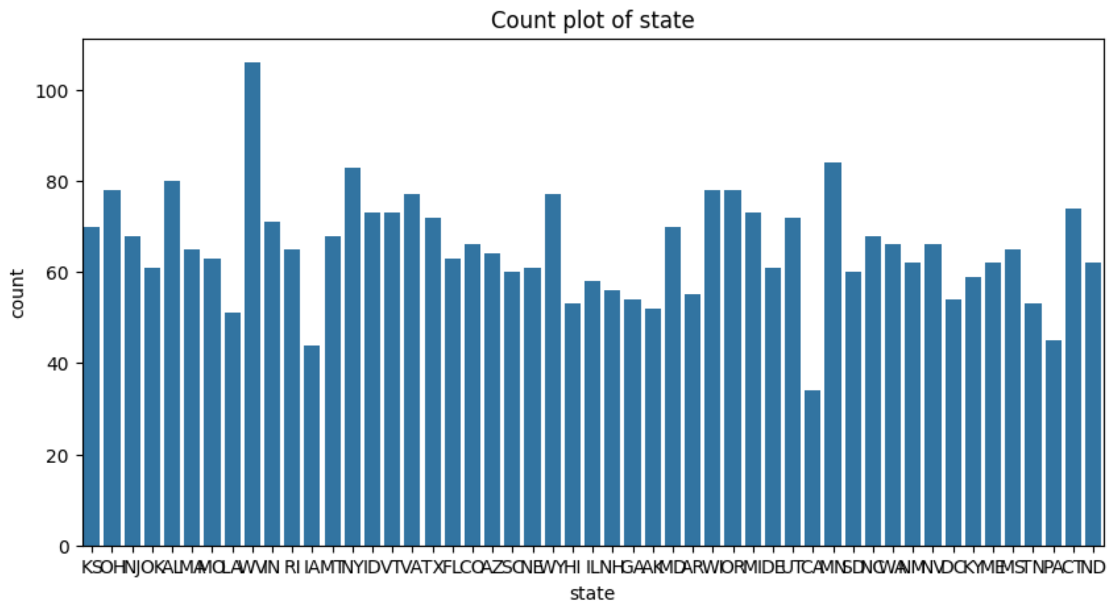

Box plot of total intl calls
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
total intl calls

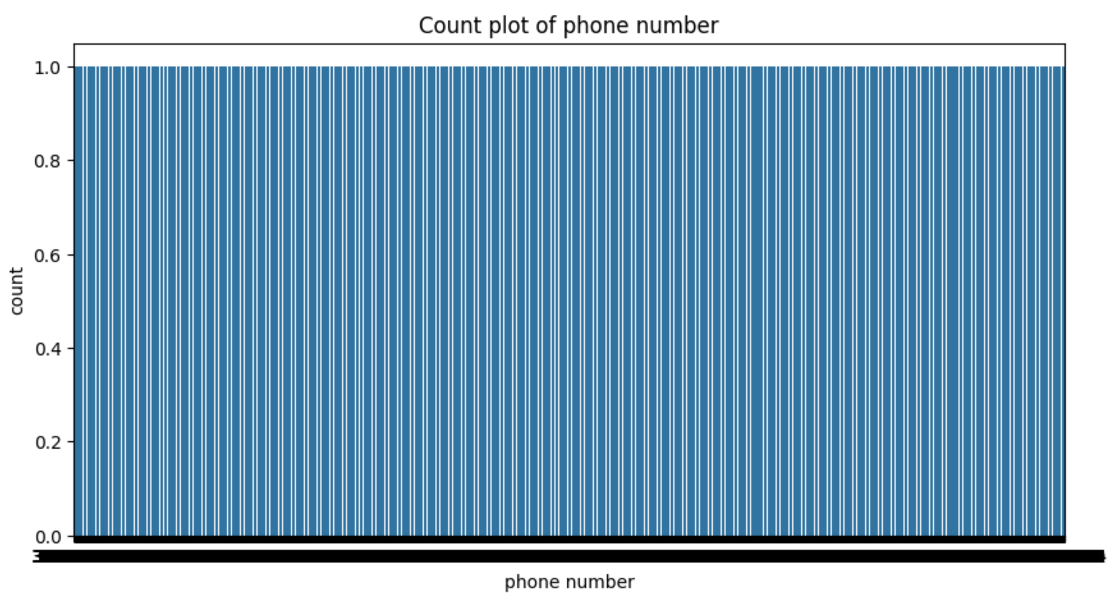


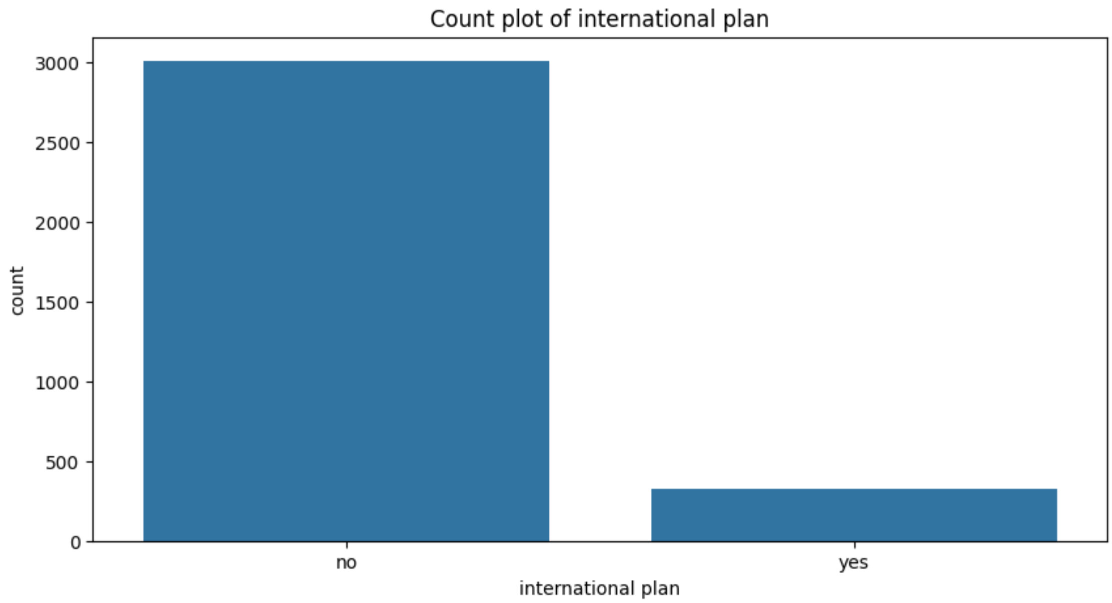


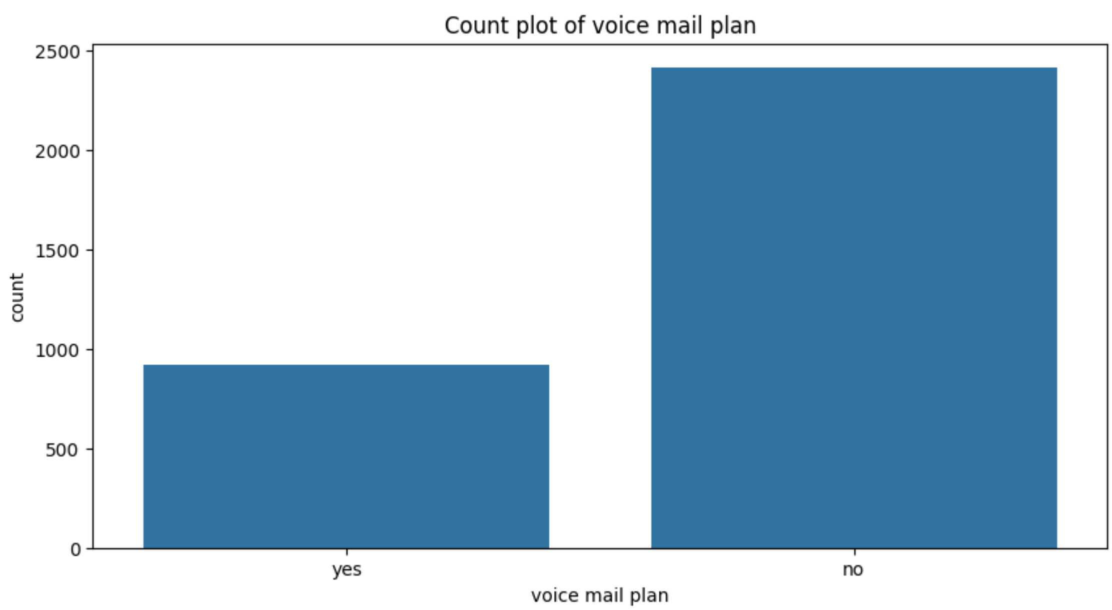
[ ]:
```
categorical_columns = df.select_dtypes(include=['object', 'bool']).columns
for column in categorical_columns:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=df[column])
```

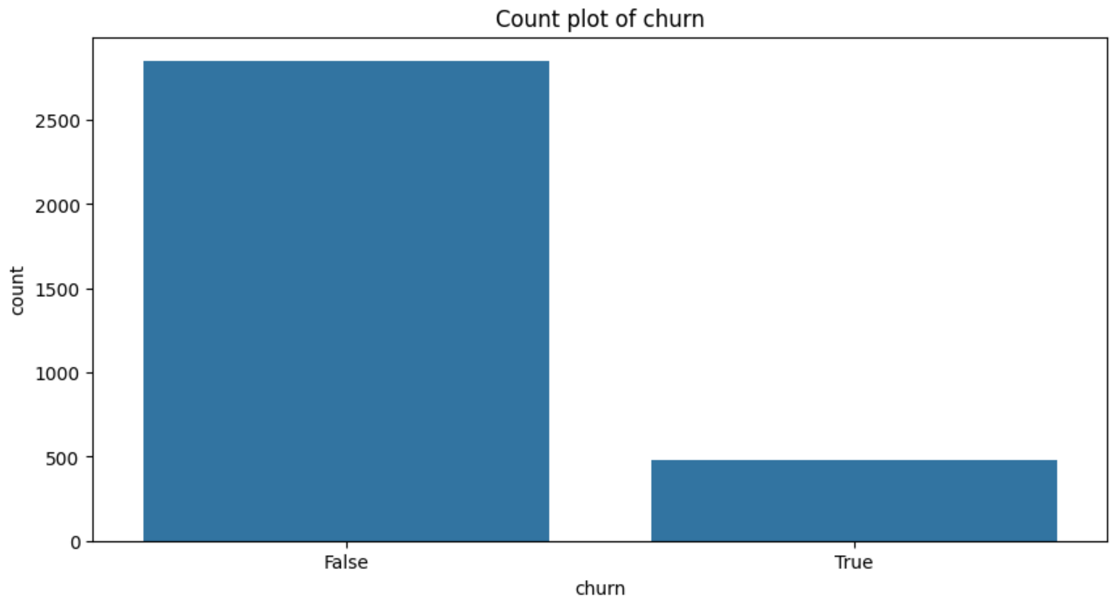
```
    plt.title(f'Count plot of {column}')
    plt.show()
```





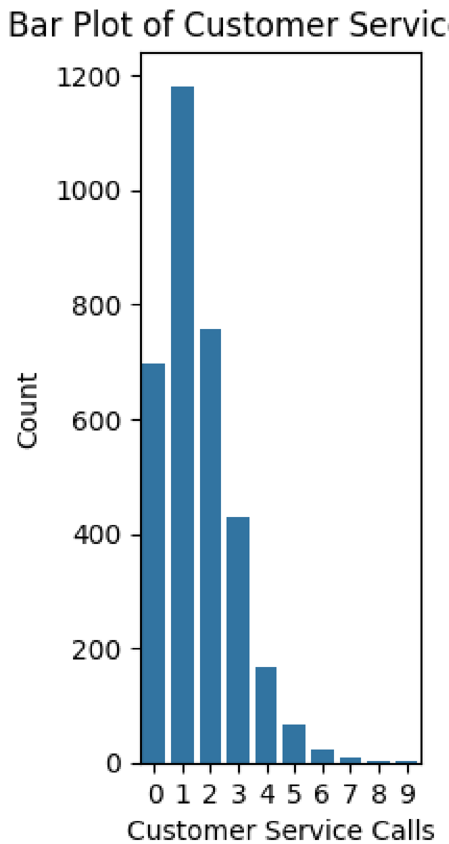

Count plot of international plan
3000
2500
2000
1500
1000
500
0
count
no
yes
international plan


Count plot of voice mail plan
2500
2000
1500
1000
500
0
count
yes
no
voice mail plan




```
[ ]: # Bar Plot
     plt.subplot(1, 3, 2)
     sns.countplot(x='customer service calls', data=df)
     plt.title('Bar Plot of Customer Service Calls')
     plt.xlabel('Customer Service Calls')
     plt.ylabel('Count')
```

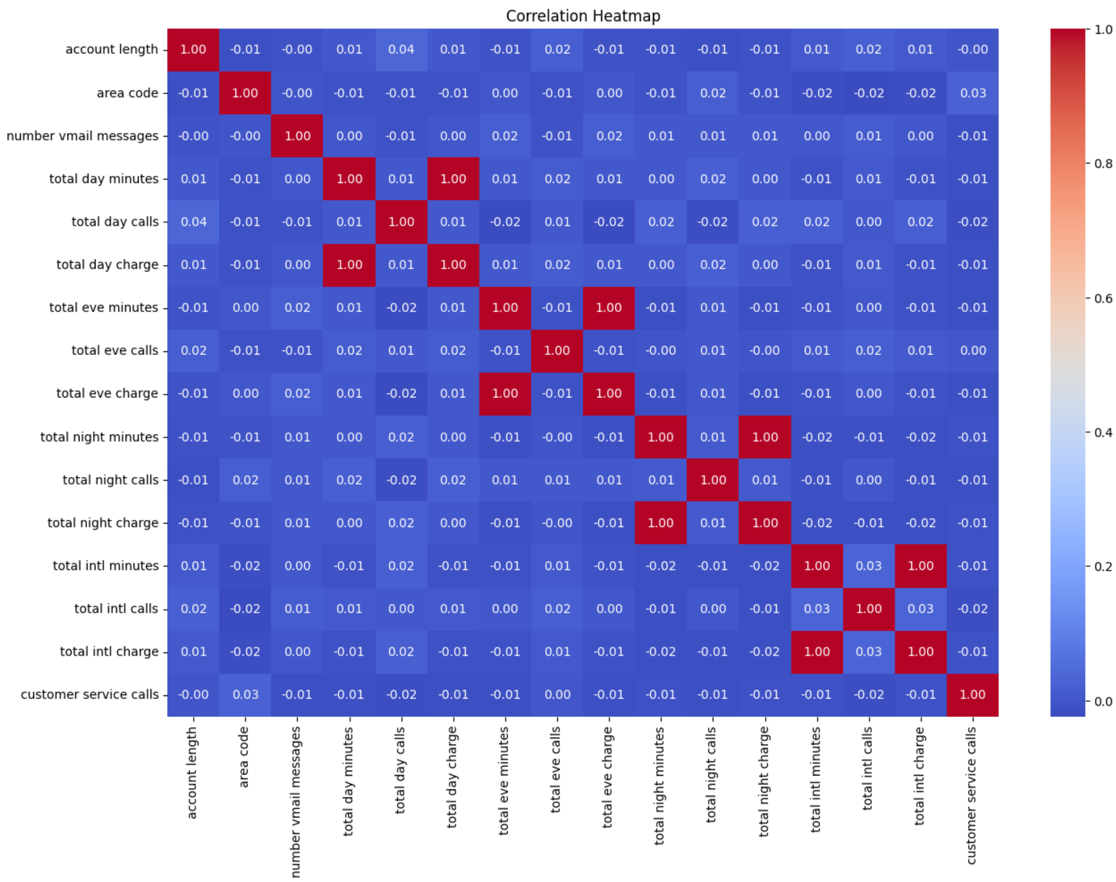
```
[ ]: Text(0, 0.5, 'Count')
```



The most number of customer service calls was 1.

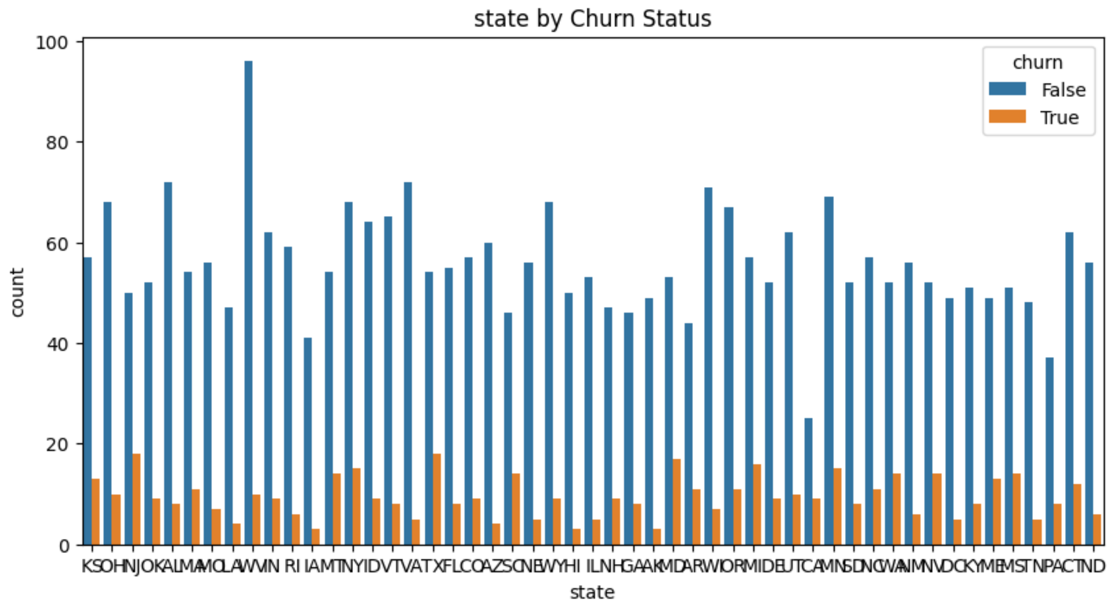
0.7.1 C. Bivariate Analysis

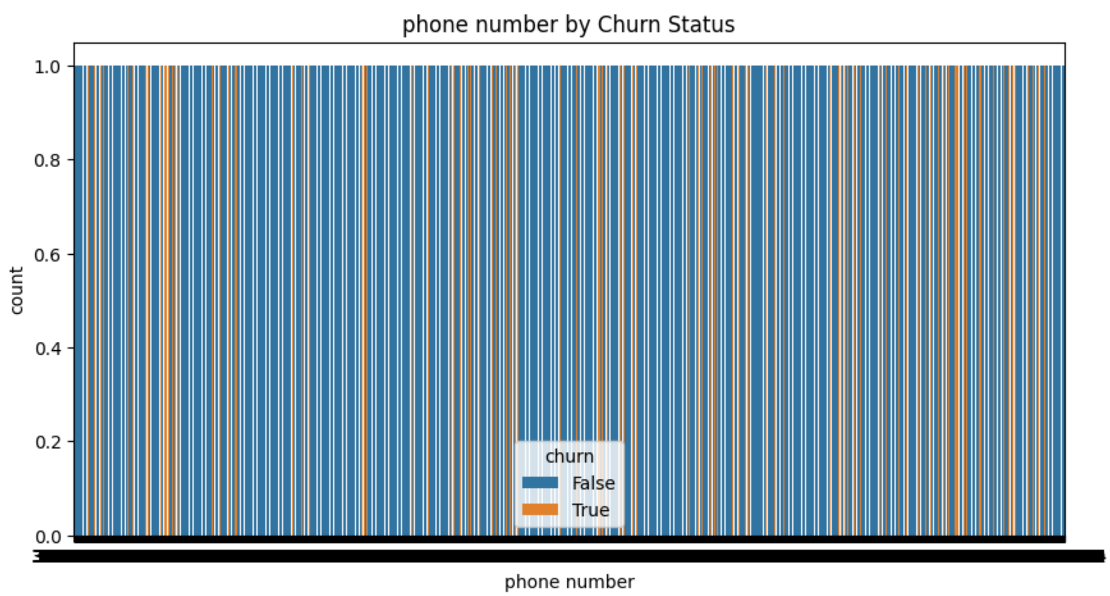
[ ]:
```
# Correlation heatmap for numerical features
plt.figure(figsize=(15, 10))
sns.heatmap(numeric_columns.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

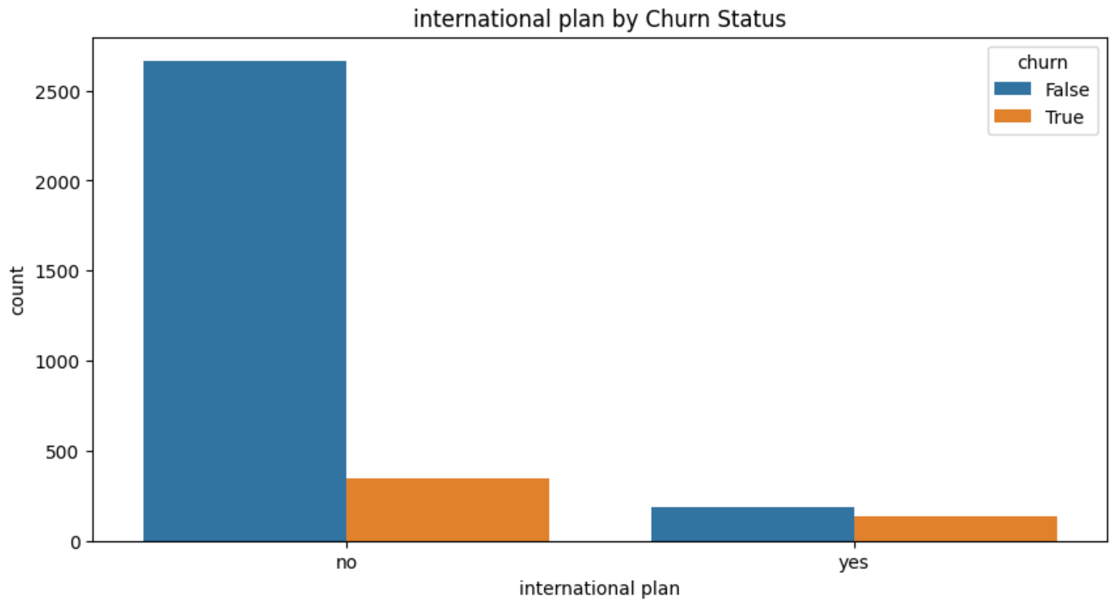


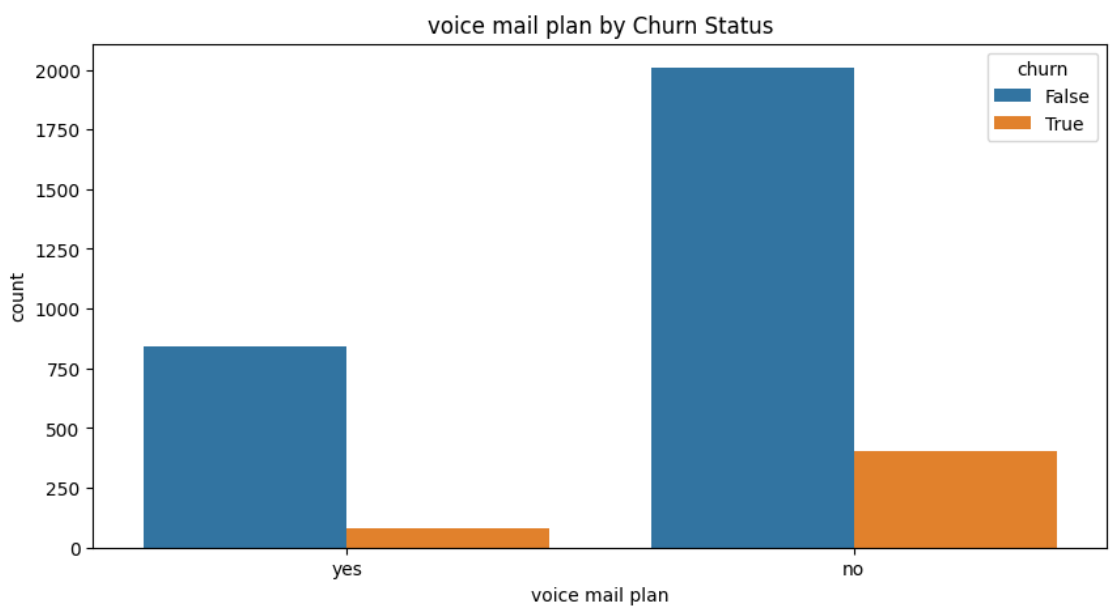
The correlation between features is very low. With most of the values being in the negatives

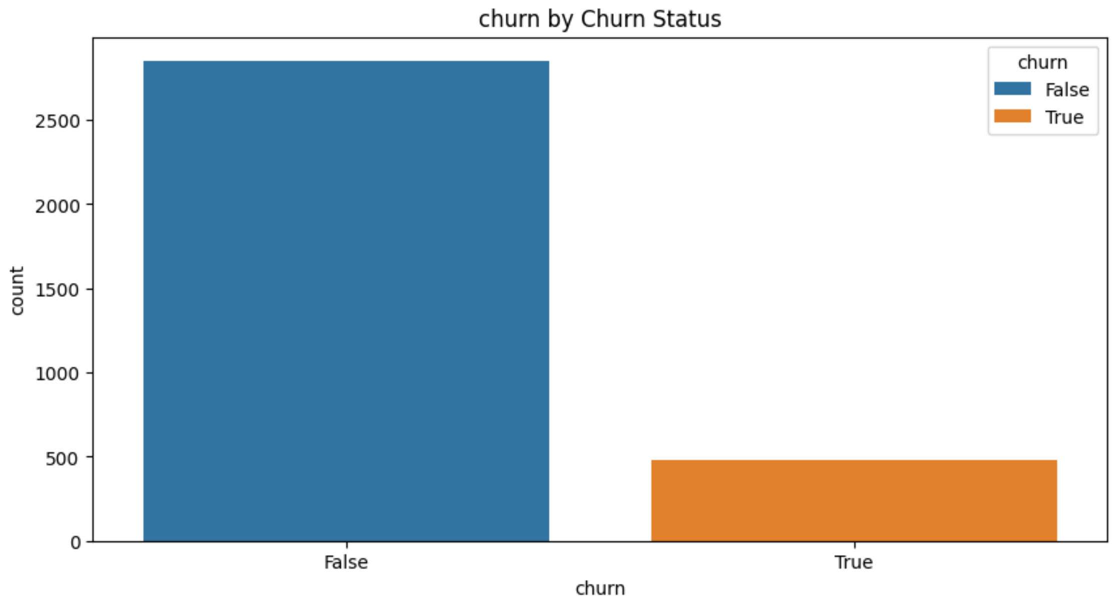
[ ]:
```python
# Bar plots of categorical features against churn
for column in categorical_columns:
    plt.figure(figsize=(10, 5))
    sns.countplot(x=column, hue='churn', data=df)
    plt.title(f'{column} by Churn Status')
    plt.show()
```


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


phone number by Churn Status
churn
False
True
count
phone number

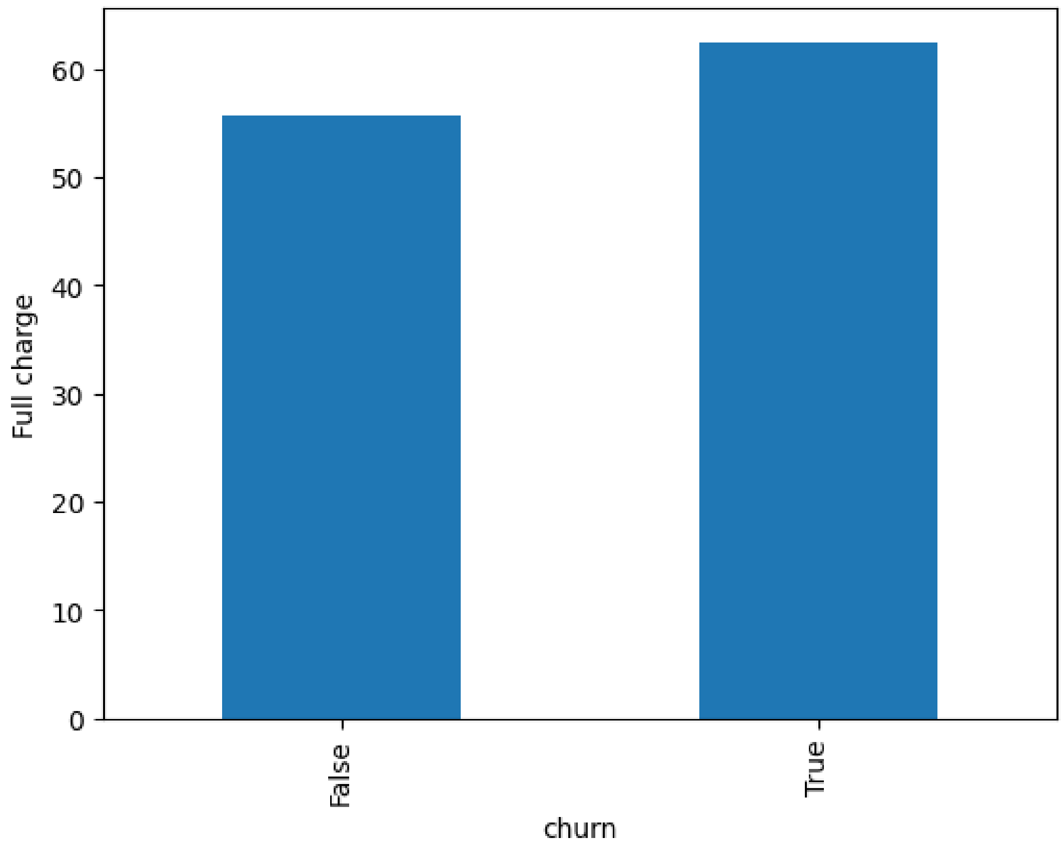

international plan by Churn Status
churn
False
True
count
2500
2000
1500
1000
500
0
no
yes
international plan


voice mail plan by Churn Status
churn
False
True
count
2000
1750
1500
1250
1000
750
500
250
0
yes
no
voice mail plan



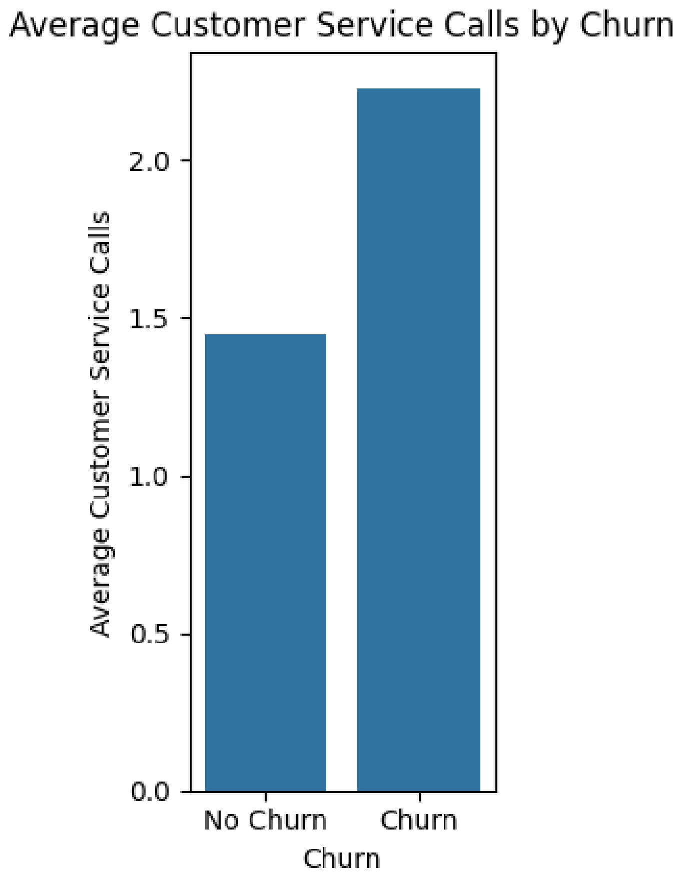

```
[ ]: # Creating a column full charge
     df['full charge'] = df['total day charge'] + df['total eve charge'] + df['total␣
      ↪night charge']
```

```
[ ]: df.groupby('churn')['full charge'].mean().plot(kind='bar')
     plt.ylabel('Full charge')
     plt.show();
```



The people with the most charges had churned (stopped using the telecommunication services)

[ ]:
```
# Bar Plot of Average Customer Service Calls by Churn
plt.subplot(1, 3, 3)
avg_calls_by_churn = df.groupby('churn')['customer service calls'].mean().
 ↪reset_index()
sns.barplot(x='churn', y='customer service calls', data=avg_calls_by_churn)
plt.title('Average Customer Service Calls by Churn')
plt.xlabel('Churn')
plt.ylabel('Average Customer Service Calls')
plt.xticks([0, 1], ['No Churn', 'Churn'])

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



The customers that churned had a more customer service calls than those that had left.

[ ]:
```
# Total calls by state
df['total_calls'] = df['total day calls'] + df['total eve calls'] + df['total␣
↪night calls'] + df['total intl calls']
calls_by_state = df.groupby('state')['total_calls'].sum().reset_index().
↪sort_values(by='total_calls', ascending=False)

# Total international calls by state
intl_calls_by_state = df.groupby('state')['total intl calls'].sum().
↪reset_index().sort_values(by='total intl calls', ascending=False)

# Total charges by state
df['total_charges'] = df['total day charge'] + df['total eve charge'] +␣
↪df['total night charge'] + df['total intl charge']
charges_by_state = df.groupby('state')['total_charges'].sum().reset_index().
↪sort_values(by='total_charges', ascending=False)
```

[ ]:
```
calls_by_state = df.groupby('state')['total_calls'].sum().reset_index()
calls_by_state_sorted = calls_by_state.sort_values(by='total_calls',␣
 ↪ascending=False)
print(calls_by_state_sorted)
```

```
   state  total_calls
49    WV        32523
23    MN        25807
34    NY        25092
1     AL        24070
35    OH        24060
50    WY        23751
37    OR        23497
45    VA        23483
48    WI        23463
13    ID        22722
22    MI        22657
6     CT        22492
46    VT        22146
43    TX        22144
15    IN        22096
44    UT        21761
16    KS        21426
27    NC        21172
31    NJ        20970
20    MD        20868
26    MT        20377
47    WA        20084
33    NV        20023
25    MS        19911
9     FL        19797
3     AZ        19671
19    MA        19602
39    RI        19525
5     CO        19434
28    ND        19394
24    MO        19348
32    NM        19278
21    ME        19075
36    OK        18930
29    NE        18718
40    SC        18397
8     DE        18392
17    KY        17971
14    IL        17752
41    SD        17517
10    GA        17087
```

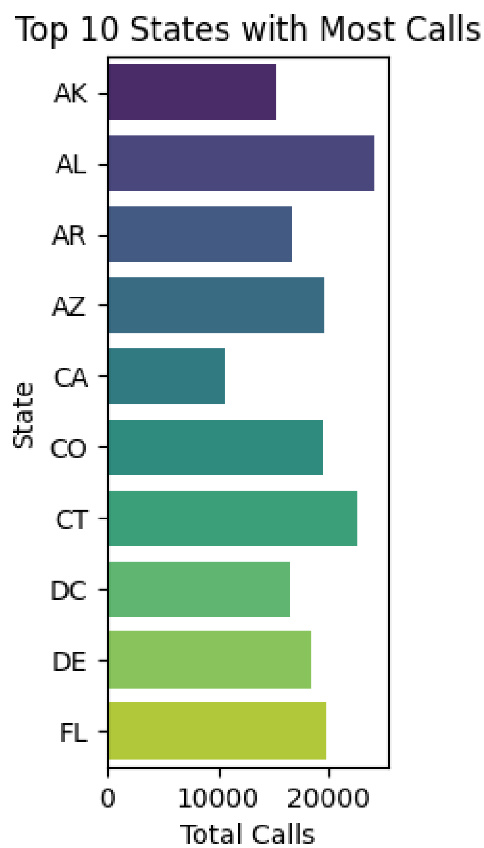
```
2     AR          16705
30    NH          16585
7     DC          16401
11    HI          16188
42    TN          16102
18    LA          15523
0     AK          15288
38    PA          13637
12    IA          13528
4     CA          10582
```

The state West Virgina had the most calls with 32523 and california had the least amount of calls with 10582

[ ]:
```python
# State with most calls
plt.subplot(1, 3, 1)
sns.barplot(x='total_calls', y='state', data=calls_by_state.head(10),␣
 ↪palette='viridis')
plt.title('Top 10 States with Most Calls')
plt.xlabel('Total Calls')
plt.ylabel('State')
```

```
<ipython-input-27-cf5c19390214>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total_calls', y='state', data=calls_by_state.head(10),
palette='viridis')
```

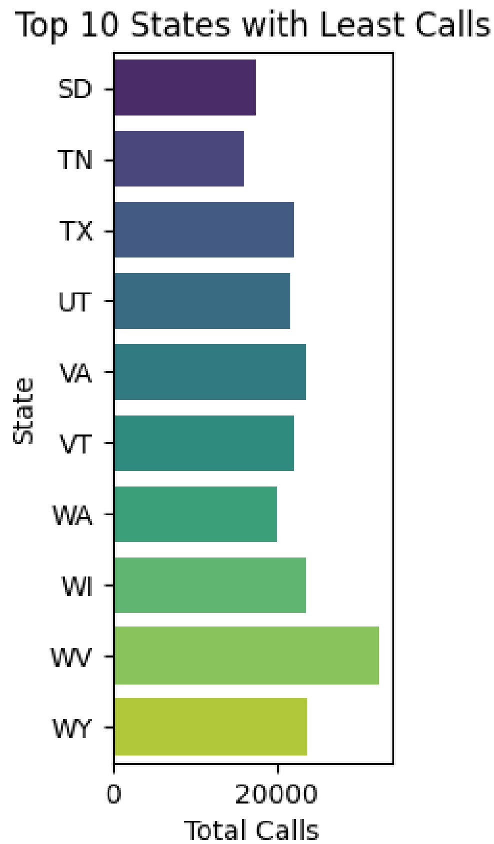
[ ]:
```
Text(0, 0.5, 'State')
```



[ ]:
```
# State with most calls
plt.subplot(1, 3, 1)
sns.barplot(x='total_calls', y='state', data=calls_by_state.tail(10),␣
  ↪palette='viridis')
plt.title('Top 10 States with Least Calls')
plt.xlabel('Total Calls')
plt.ylabel('State')
```

```
<ipython-input-28-6d1c32d9f557>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total_calls', y='state', data=calls_by_state.tail(10),
palette='viridis')
```

[ ]: Text(0, 0.5, 'State')



[ ]:
```
# Calculate total international calls for each state
int_calls_by_state = df.groupby('state')['total intl calls'].sum().reset_index()

# Sort the results by total international calls in descending order
int_calls_by_state_sorted = int_calls_by_state.sort_values(by='total intl␣
↪calls', ascending=False)

# Display the result
print(int_calls_by_state_sorted)
```

```
   state  total intl calls
49    WV               468
34    NY               385
50    WY               383
1     AL               376
45    VA               365
23    MN               364
35    OH               341
37    OR               338
46    VT               336
```

```
13    ID    333
44    UT    333
43    TX    333
22    MI    332
48    WI    322
31    NJ    319
16    KS    315
3     AZ    311
20    MD    310
26    MT    309
24    MO    306
28    ND    305
6     CT    304
15    IN    304
39    RI    302
32    NM    300
47    WA    294
25    MS    294
36    OK    293
19    MA    289
33    NV    286
21    ME    285
27    NC    280
5     CO    271
41    SD    269
30    NH    264
8     DE    262
2     AR    258
29    NE    257
17    KY    255
14    IL    252
9     FL    251
40    SC    251
0     AK    250
11    HI    245
18    LA    237
42    TN    230
10    GA    219
7     DC    211
12    IA    208
38    PA    174
4     CA    151
```

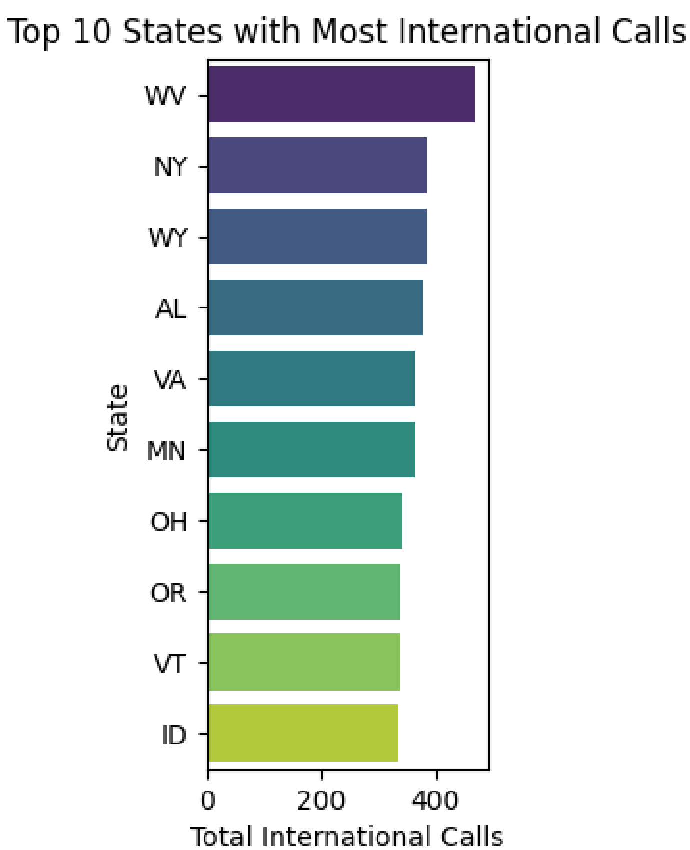
Similar to the total calls West Virgina had the most international calls totalling 468 and california with the least totalling to 151.

[ ]:
```
# State with most international calls
plt.subplot(1, 3, 2)
```

```
sns.barplot(x='total intl calls', y='state', data=intl_calls_by_state.head(10),␣
  ↪palette='viridis')
plt.title('Top 10 States with Most International Calls')
plt.xlabel('Total International Calls')
plt.ylabel('State')
```

```
<ipython-input-30-dcbe714c85d8>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total intl calls', y='state',
data=intl_calls_by_state.head(10), palette='viridis')
```

[ ]: Text(0, 0.5, 'State')

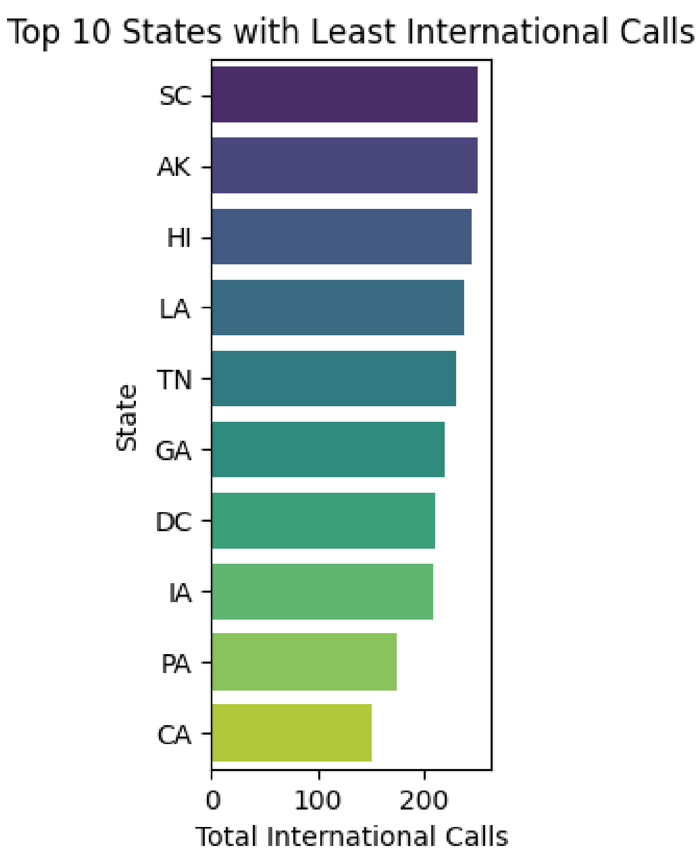


[ ]:
```
# State with most international calls
plt.subplot(1, 3, 2)
```

```
sns.barplot(x='total intl calls', y='state', data=intl_calls_by_state.tail(10),
    palette='viridis')
plt.title('Top 10 States with Least International Calls')
plt.xlabel('Total International Calls')
plt.ylabel('State')
```

```
<ipython-input-31-808dccbd9689>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total intl calls', y='state',
data=intl_calls_by_state.tail(10), palette='viridis')
```

[ ]: Text(0, 0.5, 'State')



[ ]:
```
# Sort the results by total international calls in descending order
charges_by_state_sorted = charges_by_state.sort_values(by='total_charges',
    ascending=False)
```

```python
# Display the result
print(charges_by_state_sorted)
```

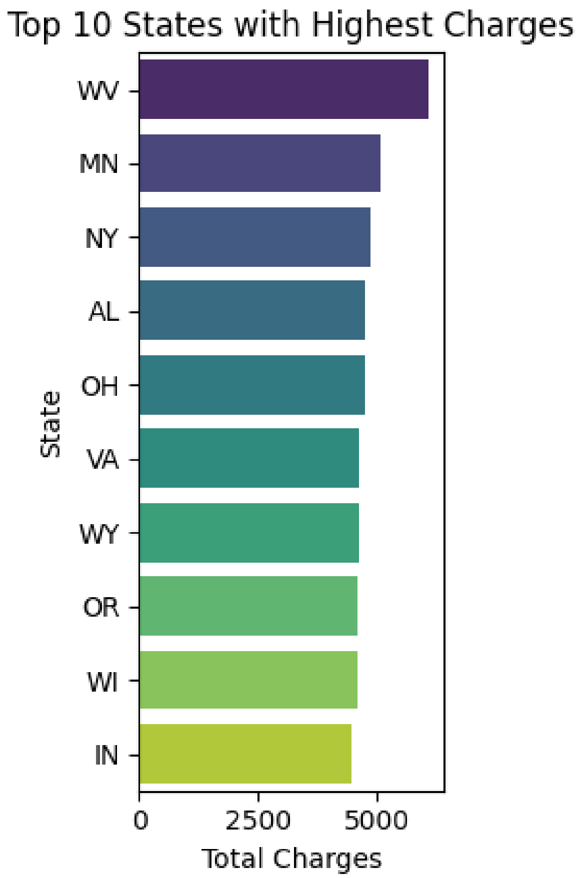
```
   state  total_charges
49    WV        6079.48
23    MN        5071.19
34    NY        4843.77
1     AL        4755.43
35    OH        4733.69
45    VA        4615.30
50    WY        4611.81
37    OR        4592.00
48    WI        4589.03
15    IN        4457.93
46    VT        4413.51
22    MI        4374.32
6     CT        4372.22
20    MD        4345.30
16    KS        4332.04
13    ID        4296.81
43    TX        4282.55
44    UT        4258.63
31    NJ        4244.56
27    NC        4097.17
26    MT        3974.84
19    MA        3946.19
47    WA        3917.56
33    NV        3916.43
5     CO        3905.78
25    MS        3846.68
28    ND        3788.76
39    RI        3787.24
9     FL        3775.89
21    ME        3731.58
41    SD        3675.33
24    MO        3662.91
32    NM        3656.85
29    NE        3632.22
3     AZ        3630.93
8     DE        3619.46
36    OK        3579.67
40    SC        3443.28
17    KY        3423.34
14    IL        3359.53
30    NH        3310.22
10    GA        3256.26
```

```
2     AR        3249.19
42    TN        3181.15
7     DC        3130.37
11    HI        3077.15
18    LA        2994.87
0     AK        2982.21
38    PA        2698.29
12    IA        2594.69
4     CA        2030.42
```

West Virgina has the highest charges amounting to 6079.48 and California has the least charges amounting to 2030.42. This is expected beacuse West Virginia has the highest amount of calls and California has the least amount of calls.

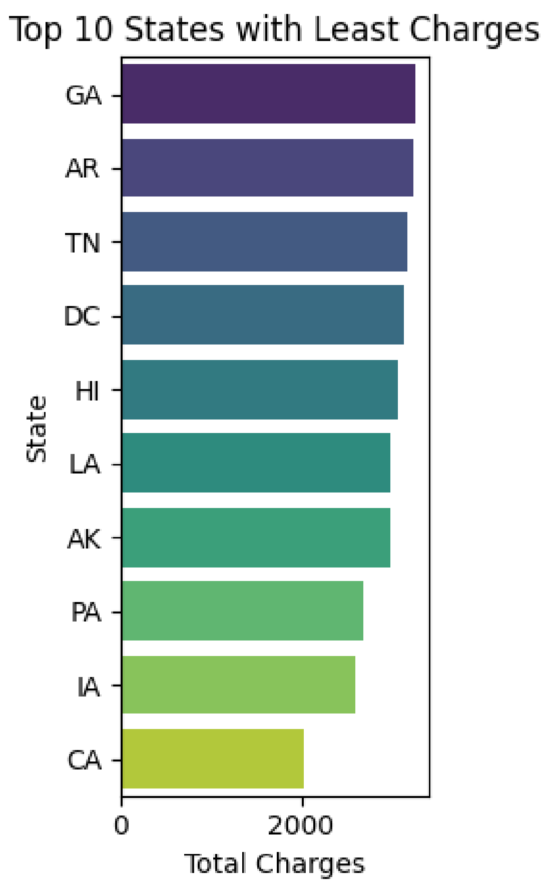
[ ]:
```
# State with highest charges
plt.subplot(1, 3, 3)
sns.barplot(x='total_charges', y='state', data=charges_by_state.head(10),␣
 ↪palette='viridis')
plt.title('Top 10 States with Highest Charges')
plt.xlabel('Total Charges')
plt.ylabel('State')

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

```
<ipython-input-33-217f8e4aa2df>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total_charges', y='state', data=charges_by_state.head(10),
palette='viridis')
```




```
[ ]: # State with least charges
plt.subplot(1, 3, 3)
sns.barplot(x='total_charges', y='state', data=charges_by_state.tail(10),␣
 ↪palette='viridis')
plt.title('Top 10 States with Least Charges')
plt.xlabel('Total Charges')
plt.ylabel('State')

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

```
<ipython-input-34-0b3a7742245b>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='total_charges', y='state', data=charges_by_state.tail(10),
palette='viridis')
```



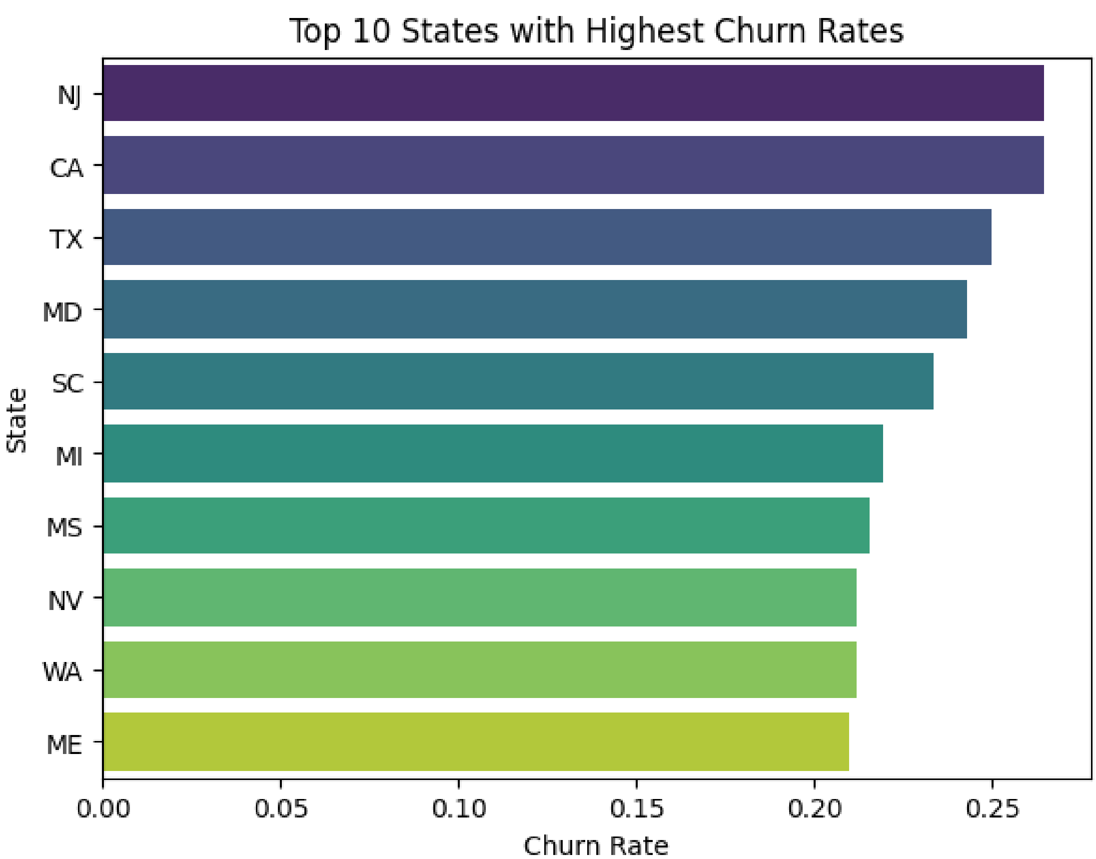
[ ]:
```python
# Calculate churn rate for each state
churn_by_state = df.groupby('state')['churn'].mean().reset_index()
churn_by_state.columns = ['state', 'churn_rate']

# Sort the states by churn rate
churn_by_state_sorted = churn_by_state.sort_values(by='churn_rate',␣
↪ascending=False)

# Get the top 10 states with the highest churn rates
top_10_highest_churn = churn_by_state_sorted.head(10)

# Get the top 10 states with the lowest churn rates
top_10_lowest_churn = churn_by_state_sorted.tail(10)
```

[ ]:
```python
# Plotting the top 10 states with the highest churn rates
plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
```

```
sns.barplot(x='churn_rate', y='state', data=top_10_highest_churn,␣
  ↪palette='viridis')
plt.title('Top 10 States with Highest Churn Rates')
plt.xlabel('Churn Rate')
plt.ylabel('State')
```

```
<ipython-input-36-1231ae5c048b>:5: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='churn_rate', y='state', data=top_10_highest_churn,
palette='viridis')
```

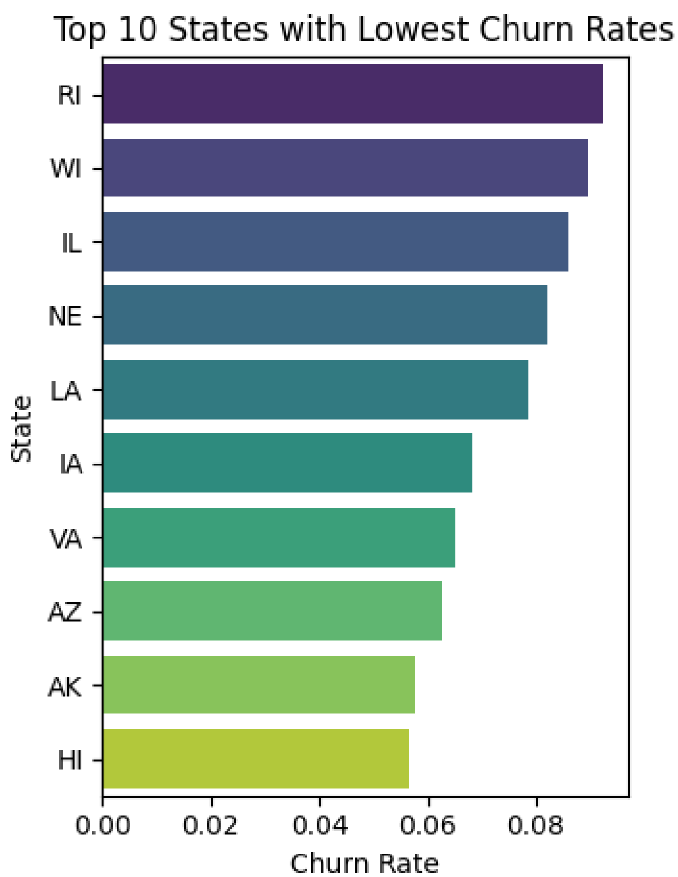
[ ]: Text(0, 0.5, 'State')



The graph shows the top 10 states with the highest churn rates

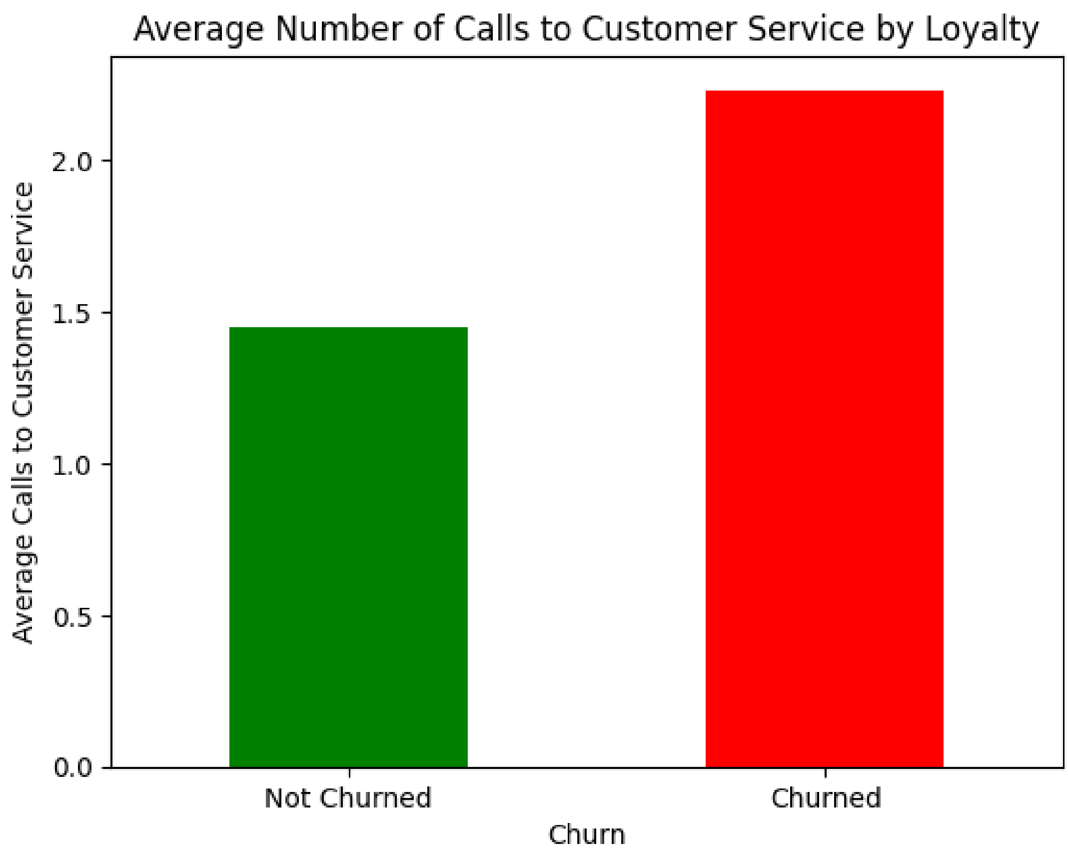
[ ]:
```python
# Plotting the top 10 states with the lowest churn rates
plt.subplot(1, 2, 2)
sns.barplot(x='churn_rate', y='state', data=top_10_lowest_churn,␣
 ↪palette='viridis')
plt.title('Top 10 States with Lowest Churn Rates')
plt.xlabel('Churn Rate')
plt.ylabel('State')

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

```
<ipython-input-37-cde508b0a998>:3: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x='churn_rate', y='state', data=top_10_lowest_churn,
palette='viridis')
```



The graph shows the Top 10 states with the lowest chrun rates

[ ]:
```
# Average number of customer service calls
avg_calls = df.groupby('churn')['customer service calls'].mean()


avg_calls.plot(kind='bar', color=['green', 'red'])
plt.title('Average Number of Calls to Customer Service by Loyalty')
plt.xlabel('Churn')
plt.ylabel('Average Calls to Customer Service')
plt.xticks([0, 1], ['Not Churned', 'Churned'], rotation=0)
plt.show()
```

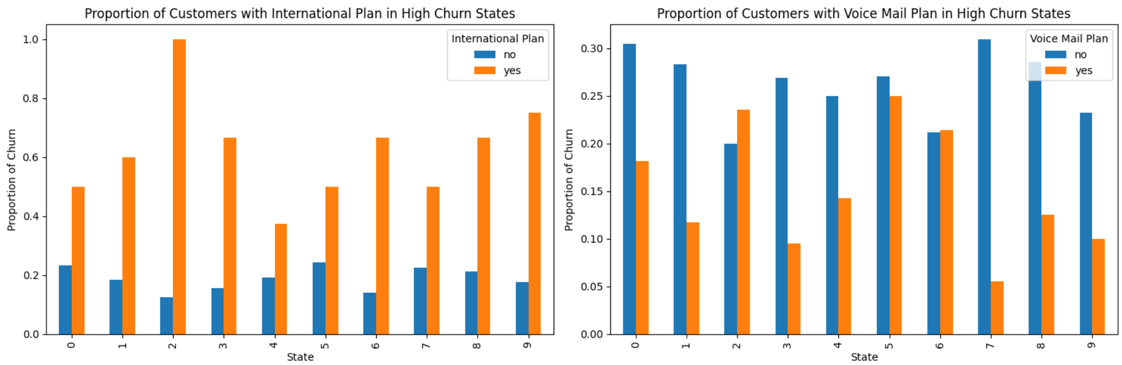


The users that left the telecommunication service called customer service more than those that stayed.

Multivariate analysis

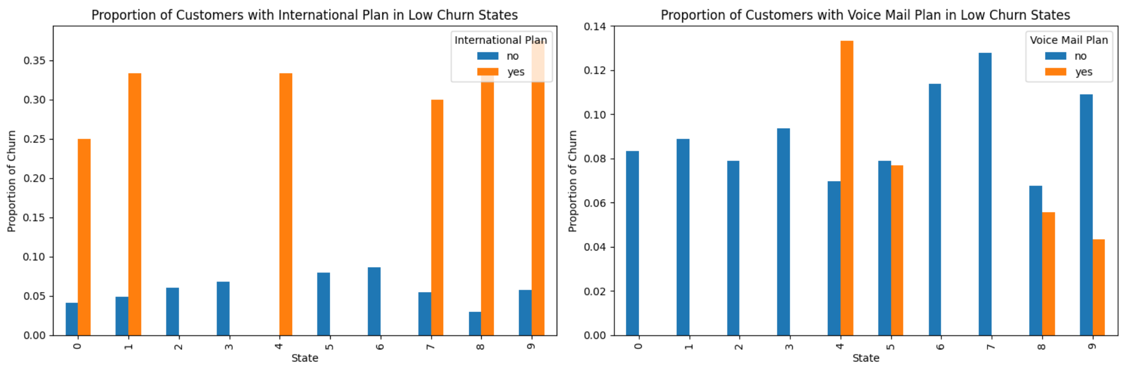
[ ]:
```
# Step 1: Calculate churn rate for each state
churn_by_state = df.groupby('state')['churn'].mean().reset_index()
churn_by_state.columns = ['state', 'churn_rate']
```

```
# Step 2: Identify the states with the highest churn rates (top 10)
top_states_by_churn = churn_by_state.sort_values(by='churn_rate',␣
↪ascending=False).head(10)
top_states = top_states_by_churn['state'].tolist()

# Step 3: Filter the original dataframe to include only customers from the top␣
↪churn states
top_churn_df = df[df['state'].isin(top_states)]

# Step 4: Create plots to compare the proportion of customers with an␣
↪international plan and voice mail plan in these states
plt.figure(figsize=(15, 5))

# Proportion of customers with an international plan
plt.subplot(1, 2, 1)
intl_plan_by_churn = top_churn_df.groupby(['state', 'international␣
↪plan'])['churn'].mean().unstack().reset_index()
intl_plan_by_churn.plot(kind='bar', stacked=False, ax=plt.gca())
plt.title('Proportion of Customers with International Plan in High Churn␣
↪States')
plt.xlabel('State')
plt.ylabel('Proportion of Churn')
plt.legend(title='International Plan', loc='upper right')

# Proportion of customers with a voice mail plan
plt.subplot(1, 2, 2)
vm_plan_by_churn = top_churn_df.groupby(['state', 'voice mail plan'])['churn'].
↪mean().unstack().reset_index()
vm_plan_by_churn.plot(kind='bar', stacked=False, ax=plt.gca())
plt.title('Proportion of Customers with Voice Mail Plan in High Churn States')
plt.xlabel('State')
plt.ylabel('Proportion of Churn')
plt.legend(title='Voice Mail Plan', loc='upper right')

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



[ ]:
```
# Step 1: Calculate churn rate for each state
churn_by_state = df.groupby('state')['churn'].mean().reset_index()
churn_by_state.columns = ['state', 'churn_rate']

# Step 2: Identify the states with the lowest churn rates (bottom 10)
lowest_states_by_churn = churn_by_state.sort_values(by='churn_rate').head(10)
lowest_states = lowest_states_by_churn['state'].tolist()

# Step 3: Filter the original dataframe to include only customers from the lowest churn states
lowest_churn_df = df[df['state'].isin(lowest_states)]

# Step 4: Create plots to compare the proportion of customers with an international plan and voice mail plan in these states
plt.figure(figsize=(15, 5))

# Proportion of customers with an international plan
plt.subplot(1, 2, 1)
intl_plan_by_churn = lowest_churn_df.groupby(['state', 'international plan'])['churn'].mean().unstack().reset_index()
intl_plan_by_churn.plot(kind='bar', stacked=False, ax=plt.gca())
plt.title('Proportion of Customers with International Plan in Low Churn States')
plt.xlabel('State')
plt.ylabel('Proportion of Churn')
plt.legend(title='International Plan', loc='upper right')

# Proportion of customers with a voice mail plan
plt.subplot(1, 2, 2)
vm_plan_by_churn = lowest_churn_df.groupby(['state', 'voice mail plan'])['churn'].mean().unstack().reset_index()
vm_plan_by_churn.plot(kind='bar', stacked=False, ax=plt.gca())
plt.title('Proportion of Customers with Voice Mail Plan in Low Churn States')
plt.xlabel('State')
```

```
plt.ylabel('Proportion of Churn')
plt.legend(title='Voice Mail Plan', loc='upper right')

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



Majority of the users in the states with Low churn rates had an international plan. However they did not have a voice mail plan.

0.8 EDA Summary

Univariate Analysis

- Target Variable Distribution: The count plot shows a higher number of non-churned customers compared to churned ones, indicating class imbalance.
- Numeric Features Distribution: Histograms reveal that most features follow a normal distribution except for 'total international calls', 'customer service calls', and 'area code'.
- Boxplots: Highlight the presence of outliers in numerical features.

Bivariate Analysis

- Correlation Heatmap: Displays low correlations among most features.
- Categorical Features vs. Churn: Count plots indicate how different categorical features (e.g., 'international plan', 'voice mail plan') relate to churn status.
- Full Charge Analysis: Higher average charges correlate with a higher likelihood of churn.

0.9 Data Preprocessing

Preparing the data before we start modeling the data. For this I will perform one hot encoding, spiliting and feature scaling.

[ ]:
```
df.columns
```

[ ]:
```
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', 'full charge', 'total_calls',
       'total_charges'],
      dtype='object')
```

[ ]:
```
columns_to_drop = ['account length', 'area code', 'phone number']
data = df.drop(columns=columns_to_drop)
data.columns
```

[ ]:
```
Index(['state', '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', 'full charge',
       'total_calls', 'total_charges'],
      dtype='object')
```

0.10 One Hot Encoing

[ ]:
```
# Binary encoding for binary categorical columns
data['international plan'] = data['international plan'].map({'no': 0, 'yes': 1})
data['voice mail plan'] = data['voice mail plan'].map({'no': 0, 'yes': 1})
data['churn'] = data['churn'].astype(int)
```

[ ]:
```
# Viewing the data after one hot encoding
data.head()
```

[ ]:
```
  state  international plan  voice mail plan  number vmail messages  \
0    KS                   0                1                     25
1    OH                   0                1                     26
2    NJ                   0                0                      0
3    OH                   1                0                      0
4    OK                   1                0                      0

   total day minutes  total day calls  total day charge  total eve minutes  \
0              265.1              110             45.07              197.4
1              161.6              123             27.47              195.5
2              243.4              114             41.38              121.2
3              299.4               71             50.90               61.9
4              166.7              113             28.34              148.3
```

```
   total eve calls  total eve charge  …  total night calls  \
0               99             16.78  …                 91
1              103             16.62  …                103
2              110             10.30  …                104
3               88              5.26  …                 89
4              122             12.61  …                121

   total night charge  total intl minutes  total intl calls  \
0               11.01                10.0                 3
1               11.45                13.7                 3
2                7.32                12.2                 5
3                8.86                 6.6                 7
4                8.41                10.1                 3

   total intl charge  customer service calls  churn  full charge  total_calls  \
0               2.70                       1      0        72.86          303
1               3.70                       1      0        55.54          332
2               3.29                       0      0        59.00          333
3               1.78                       2      0        65.02          255
4               2.73                       3      0        49.36          359

   total_charges
0          75.56
1          59.24
2          62.29
3          66.80
4          52.09

[5 rows x 21 columns]
```

0.11 Feature Selection

[ ]:
```
# Selecting the features
selected_features = ['international plan', 'voice mail plan', 'number vmail␣
 ↪messages',
                     'total intl calls', 'total_calls', 'total_charges',␣
 ↪'customer service calls',
                     'total intl calls']

# Selecting the target variable
target_variable = 'churn'

# Creating X (features) and y (target)
X = data[selected_features]
y = data[target_variable]
```

0.12 Spilting the data

[ ]:
```
# Splitting the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

0.13 1. Logistic Regression

[ ]:
```
# Initializing and fitting the logistic regression model
model= LogisticRegression()
model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

[ ]: LogisticRegression()

[ ]:
```
# Making predictions
y_pred = model.predict(X_test)
```

0.14 Evaluation

[ ]:
```
# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.8545727136431784
```

[ ]:
```
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92       566
           1       0.57      0.16      0.25       101
```

```
    accuracy                           0.85       667
   macro avg       0.72      0.57      0.58       667
weighted avg       0.82      0.85      0.82       667
```

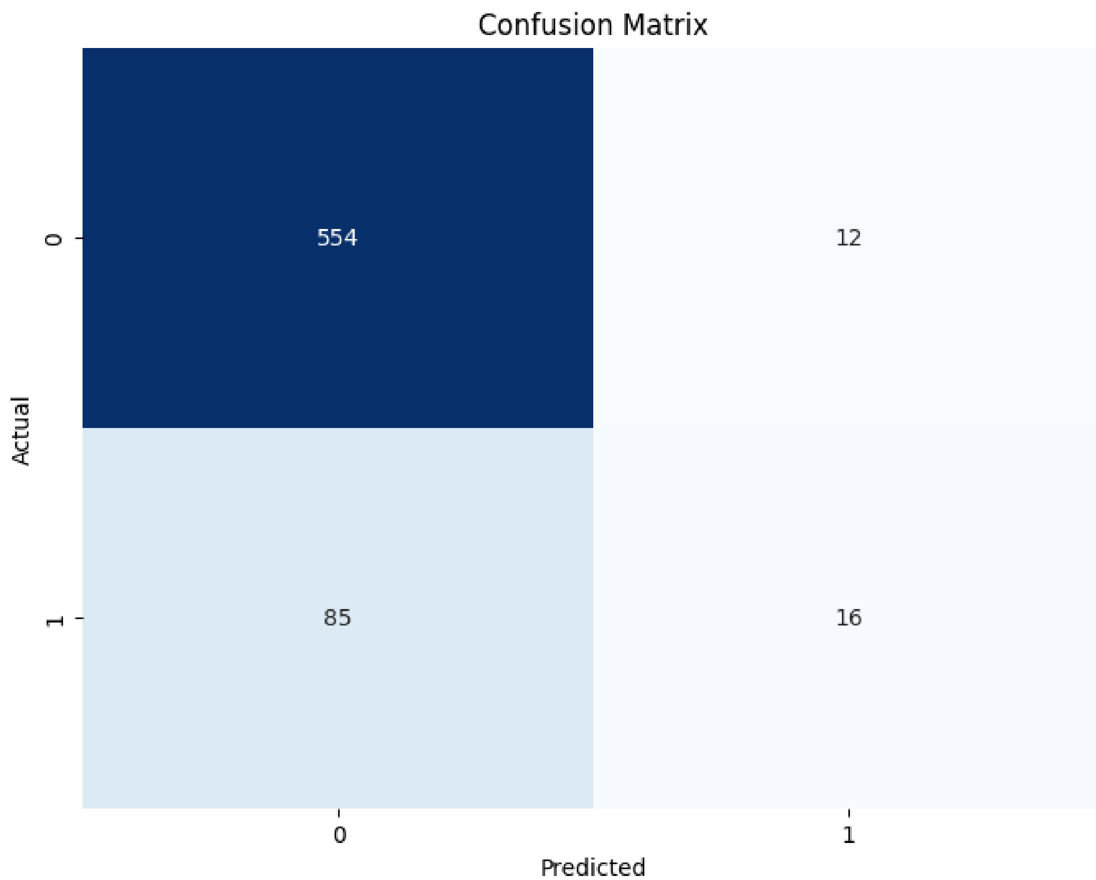
The model shows high precision for class 0 (non-churned customers), indicating that when it predicts a customer will not churn, it is correct 87% of the time.

However, the recall for class 1 (churned customers) is low, indicating that the model misses many churned customers.

The F1-score for class 1 is also relatively low, reflecting the imbalance between precision and recall. The overall accuracy of 85% seems high, but it might be misleading due to the class imbalance. It's important to consider precision, recall, and F1-score for each class to get a better understanding of the model's performance, especially in imbalanced datasets like this one.

[ ]:

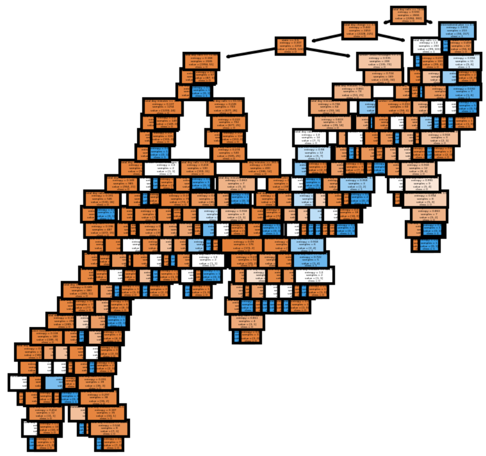
```
from sklearn.metrics import confusion_matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting the Confusion Matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



0.15 2. Decision Trees

[ ]:
```
# Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(criterion = "entropy")

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

# Make predictions
y_pred = dt_classifier.predict(X_test)
```

[ ]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn import tree
fig, axes = plt.subplots(nrows = 1,ncols = 1, figsize = (3,3), dpi=300)
tree.plot_tree(dt_classifier,
               feature_names = data.columns,
               class_names=np.unique(y).astype('str'),
               filled = True)
plt.show()
```



It is quiet tricky to visualize the decision tree because there are a lot of features.

0.16 Evaluation

[ ]:
```python
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9325337331334332
```

[ ]:
```python
# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

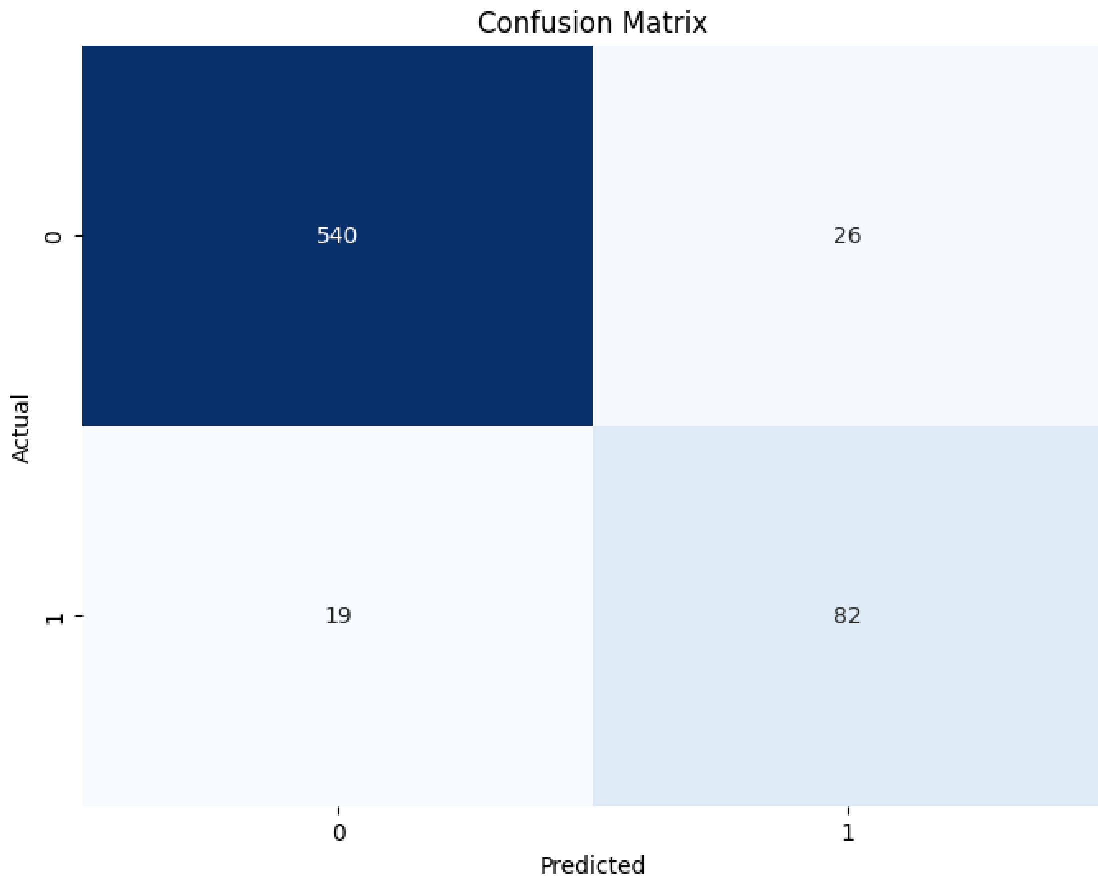
```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.95      0.96       566
           1       0.76      0.81      0.78       101

    accuracy                           0.93       667
   macro avg       0.86      0.88      0.87       667
weighted avg       0.93      0.93      0.93       667
```

Interpretation: The precision of 0.75 for class 1 suggests that when the model predicts a customer will churn, it is correct about 75% of the time. The recall of 0.81 for class 1 indicates that the model correctly identifies about 81% of all actual churned customers. The F1-score of 0.78 for class 1 balances precision and recall, providing a single metric to evaluate the model's performance. The accuracy of 0.93 indicates that the model performs well overall, correctly classifying 93% of the instances in the test dataset. Macro avg and weighted avg provide aggregate metrics that consider both classes and their respective support.

[ ]:
```python
# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[540  26]
 [ 19  82]]
```

[ ]:
```python
# Plotting Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



Correct Predictions: The model correctly predicted 538 instances of non-churned customers and 82 instances of churned customers.

Incorrect Predictions:

The model incorrectly predicted 28 instances as churned customers when they are actually non-churned customers (false positives). The model incorrectly predicted 19 instances as non-churned customers when they are actually churned customers (false negatives). Interpretation: The confusion matrix suggests that the model is performing well overall, with a large number of true positives and true negatives. The false positives and false negatives are relatively low, indicating that the model's errors are minimal. However, it's essential to consider the specific context and implications of false positives and false negatives in your application. For example, false positives may result in unnecessary interventions for customers who are not actually at risk of churning, while false negatives may lead to missed opportunities to intervene with customers who are at risk of churning.

0.17 Hyperparameter Tuning for Decision Trees

[ ]:
```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.metrics import classification_report

# Assuming X and y are your features and target variable
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Create a decision tree classifier object
dt_classifier = DecisionTreeClassifier()

# Define the hyperparameters to tune
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
```

[ ]:
```
# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=dt_classifier, param_grid=param_grid,␣
↪cv=5, scoring='accuracy')
```

[ ]:
```
# Perform hyperparameter tuning
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'criterion': 'entropy', 'max_depth': 10,
'min_samples_leaf': 2, 'min_samples_split': 10}
```

```
[ ]: # Get the best model
best_dt_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_dt_model.predict(X_test)
```

0.18 Evaluation

```
[ ]: # Evaluate the model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       566
           1       0.94      0.80      0.87       101

    accuracy                           0.96       667
   macro avg       0.95      0.90      0.92       667
weighted avg       0.96      0.96      0.96       667
```

Improvement in Precision and F1-score: After hyperparameter tuning, there is a notable improvement in precision for class 1, indicating that the model's ability to correctly predict churned customers has significantly increased. The F1-score, which balances precision and recall, also shows improvement, indicating a better overall performance in classifying churned customers.

Slight Decrease in Recall: Although there was an increase in precision, the recall for class 1 slightly decreased after hyperparameter tuning. This suggests that the model may miss a few more actual churned customers compared to before tuning.

Significant Increase in Accuracy: The overall accuracy of the model improved from 0.93 to 0.96 after hyperparameter tuning. This indicates that the model's ability to correctly classify both churned and non-churned customers improved significantly.

Balanced Evaluation Metrics: The macro avg and weighted avg of precision, recall, and F1-score also show improvements after hyperparameter tuning, indicating a more balanced performance across both classes.

0.19 KNN

```
[ ]: # Create a KNN classifier object
from sklearn.neighbors import KNeighborsClassifier
knn_classifier = KNeighborsClassifier()  # You can specify the number of
 ↪neighbors (k) here

# Train the model
```

```
knn_classifier.fit(X_train, y_train)

# Make predictions on the test set
y_pred = knn_classifier.predict(X_test)
```

[ ]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```
Accuracy: 0.9115442278860569

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       566
           1       0.94      0.45      0.60       101

    accuracy                           0.91       667
   macro avg       0.92      0.72      0.78       667
weighted avg       0.91      0.91      0.90       667
```

Interpretation:

The precision and recall trade-off: The model achieves high precision for both classes, indicating that the predictions are reliable. However, the recall for class 1 (churned customers) is relatively low, indicating that the model struggles to correctly identify churned customers.

F1-score: The F1-score for class 1 is moderate, indicating a balance between precision and recall, but it could be improved, especially in terms of recall.

Class imbalance: There's a significant class imbalance, with a larger number of instances for non-churned customers compared to churned customers. This can affect the model's performance, especially for the minority class (churned customers).

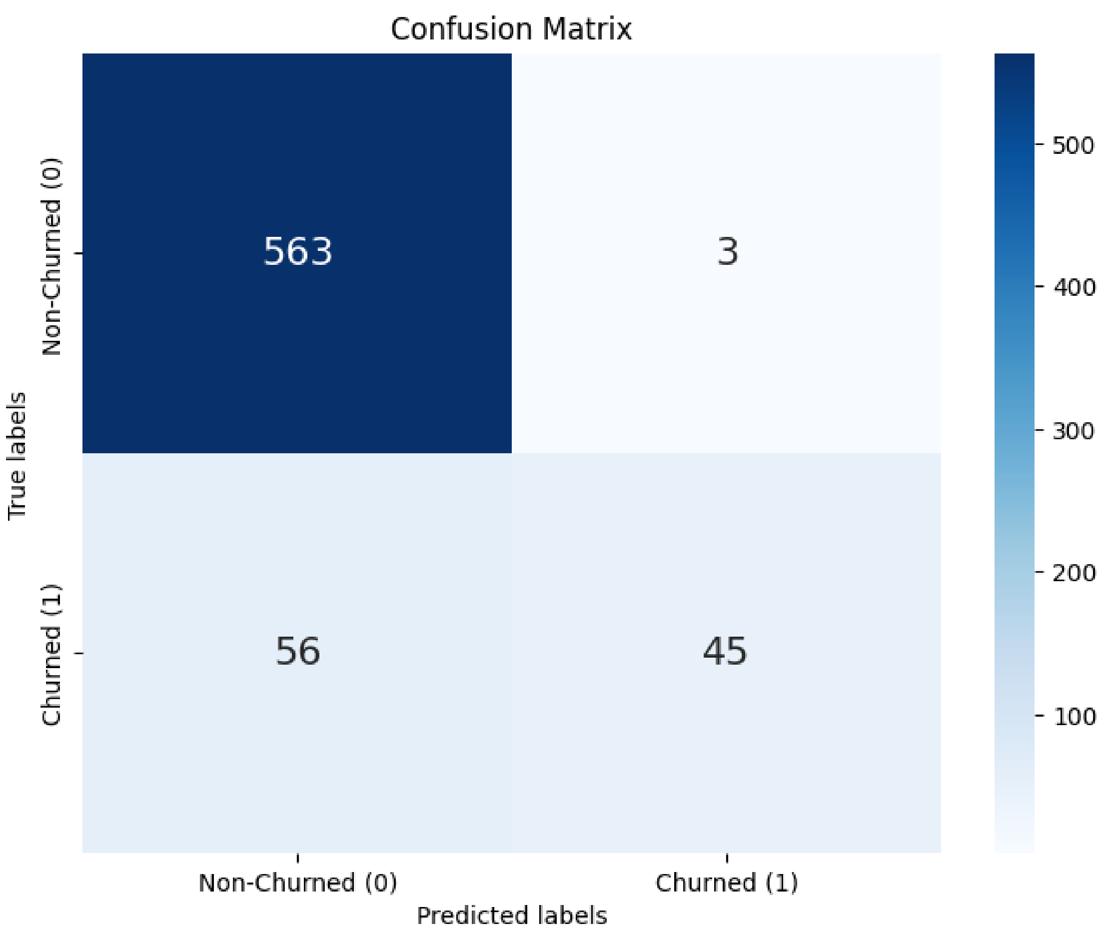
Overall accuracy: The model achieves a relatively high accuracy, but it's important to consider the context and implications of misclassifications, especially for churned customers. Further optimization may be needed to improve the model's performance, particularly in terms of recall for churned customers.

[ ]:
```
# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
```

```
[[563   3]
 [ 56  45]]
```

[ ]:
```python
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})

# Add labels, title, and ticks
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.xticks(ticks=[0.5, 1.5], labels=['Non-Churned (0)', 'Churned (1)'])
plt.yticks(ticks=[0.5, 1.5], labels=['Non-Churned (0)', 'Churned (1)'])

# Show plot
plt.show()
```



0.20 Improving model performance

[ ]:
```
# Create a KNN classifier object
knn_classifier = KNeighborsClassifier()

# Define the hyperparameters to tune
param_grid = {
    'n_neighbors': [3, 5, 7, 9],  # number of neighbors
    'weights': ['uniform', 'distance'],  # weight function used in prediction
    'p': [1, 2]  # power parameter for Minkowski distance
}

# Initialize GridSearchCV
grid_search = GridSearchCV(estimator=knn_classifier, param_grid=param_grid,␣
 ↪cv=5, scoring='accuracy')

# Perform hyperparameter tuning
grid_search.fit(X_train, y_train)

# Print the best hyperparameters found
print("Best Hyperparameters:", grid_search.best_params_)
```

```
Best Hyperparameters: {'n_neighbors': 5, 'p': 2, 'weights': 'distance'}
```

[ ]:
```
# Get the best model
best_knn_model = grid_search.best_estimator_

# Make predictions on the test set
y_pred = best_knn_model.predict(X_test)
```

0.21 Evaluation

[ ]:
```
# Evaluate the model
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

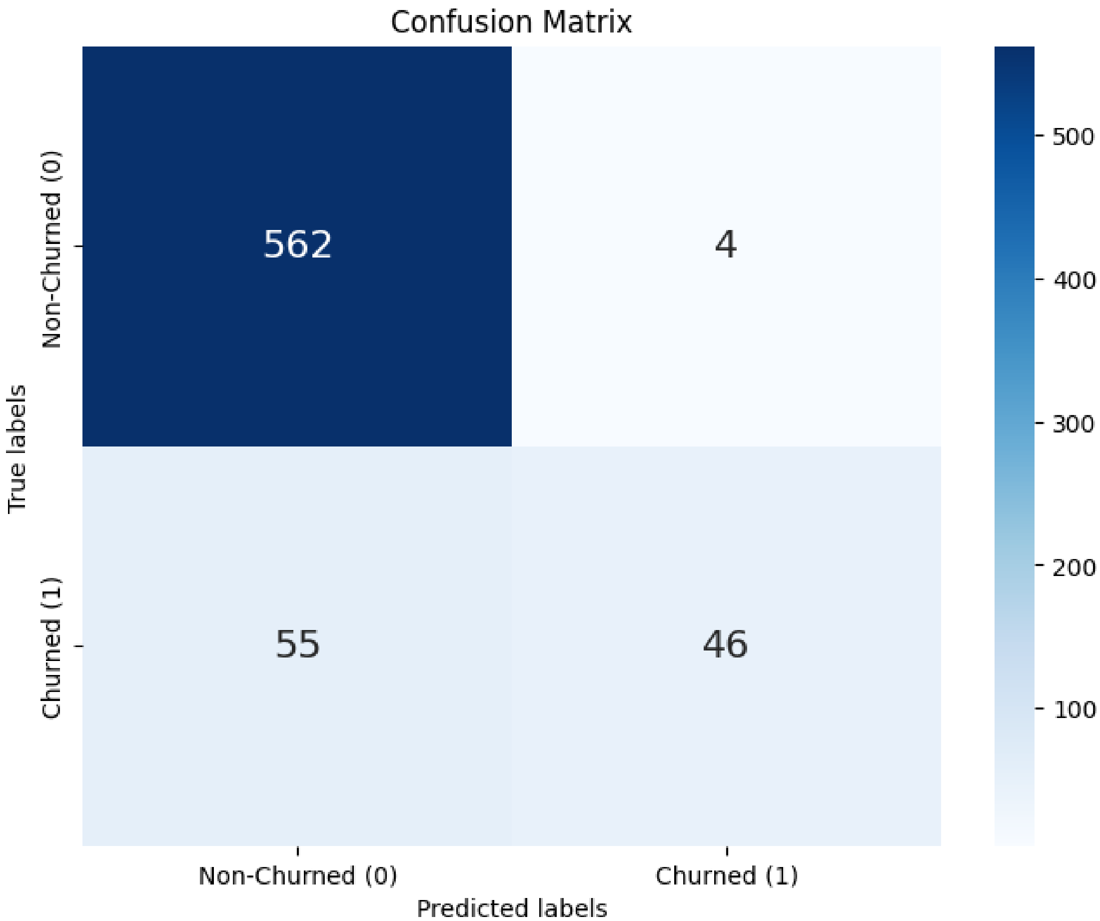
```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       566
           1       0.92      0.46      0.61       101

    accuracy                           0.91       667
   macro avg       0.92      0.72      0.78       667
```

```
weighted avg       0.91      0.91      0.90       667
```

[ ]:
```python
# Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
Confusion Matrix:
[[562   4]
 [ 55  46]]
```

[ ]:
```python
# Calculate confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 16})

# Add labels, title, and ticks
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.xticks(ticks=[0.5, 1.5], labels=['Non-Churned (0)', 'Churned (1)'])
plt.yticks(ticks=[0.5, 1.5], labels=['Non-Churned (0)', 'Churned (1)'])

# Show plot
plt.show()
```



Interpretation:

Precision: Before tuning, the precision for predicting churned customers was higher (0.94) compared to after tuning (0.92), indicating that a higher proportion of predicted churned customers were actually churned before tuning. However, the difference is relatively small.

Recall: Recall measures the ability of the model to correctly identify all actual churned customers. Before tuning, recall for churned customers was 0.45, and after tuning, it slightly increased to 0.46. This indicates a minor improvement in the model's ability to capture actual churned customers.

F1-Score: F1-score is the harmonic mean of precision and recall, providing a balance between them. Both precision and recall contribute to the F1-score. The F1-score for predicting churned customers improved marginally from 0.60 to 0.61 after tuning.

Accuracy: The overall accuracy remained the same at 0.91 before and after tuning. This suggests that while some improvements were observed in precision, recall, and F1-score for predicting churned customers, they were not significant enough to affect the overall accuracy of the model.

In summary, while hyperparameter tuning led to slight improvements in precision, recall, and F1-score for predicting churned customers, these improvements did not substantially impact the overall accuracy of the model

0.22 Model Comparison

In this project I used three different models and evaluated each.

Logistic Regression: Simpler model with decent performance but struggles with class imbalance.

Decision Tree: Better balance between precision and recall, good interpretability but might overfit.

KNN: High accuracy and precision but indicates potential overfitting and imbalance handling issues.

##**Recommendations Improve Data Balance**: Implement techniques like SMOTE (Synthetic Minority Over-sampling Technique) to balance the classes in the dataset.

Model Ensemble: Combine multiple models (e.g., using stacking or voting classifiers) to leverage their individual strengths.

Customer Segmentation: Use clustering techniques to segment customers and apply targeted retention strategies for different segments.

Develop a loyalty program that offers benefits like discounts, priority service, or exclusive offers for long-term customers.

Use CRM (Customer Relationship Management) tools to track customer history and preferences.

Reduce the number of calls to customer service by providing customers with self-service tools.

0.23 Next Steps

Deployment: Develop a deployment pipeline to integrate the best model into SyriaTel's system for real-time churn prediction.

Monitoring and Maintenance: Continuously monitor model performance and update it with new data to maintain its accuracy over time.

A/B Testing: Implement A/B tests to measure the effectiveness of retention strategies suggested by the model.

Customer Feedback: Collect feedback from customers to refine the model and retention strategies further.

Regular Updates: Schedule regular updates and model retraining sessions to ensure the model adapts to any changes in customer behavior.