- Student name: Charles Ondieki Otwori
- Student pace: Part-time
- Scheduled project review date/time: December 23, 2024
- Instructor name: Daniel Ekale
- Github link : [https://github.com/charlesot/DSF-PT08P3-Phase-3-project-.git (https://github.com/charlesot/DSF-PT08P3-Phase-3-project-.git)](https://github.com/charlesot/DSF-PT08P3-Phase-3-project-.git)

1 1. Project overview

SyriaTel is telecommunication company that is interested in reducing the number of customer churn/attrition to reduce loss of revenue. The project will enable the reduction of churn through analytics of historical data and development of machine learning models to predict customers who are likely to churn and develop a mitigation strategy.

2 2. Business and Data understanding

Telecom companies experience customer churn/attrition when customers voluntarily cease their relationship with the company. This leads to financial loss due to reduction in revenue and loss of market share. It is more expensive to acquire a new customer than to main existing ones. During churn the company also loses the future revenue from the customer and the resources spent to acquire the customer. Thus, reducing profitability.Beyond financial loss, high customer churn can indicate a deeper problem with the company in terms of customer service, product appeal or quality of processes. This can erode the company's reputation and further erode the market share.

2.1 The objective of the project is to predict customer churn using public dataset with customer usage patterns and if the customer has churned or not.

- Analytic of churn will provide insights on why customers churn and will help in identifying customers who are likely to leave so that a targeted strategy can be developed to convince them to stay
- The Stakeholder audience for the project is SyriaTel telecommunication company executives from marketing, sales and innovations departments.

2.2 The predictors (features) include the following

- account length
- international plan

- voice mail plan
- number of voice mail messages
- total day minutes used
- day calls made
- total day charge
- total evening minutes
- total evening calls
- total evening charge
- total night minutes
- total night calls
- total night charge
- total international minutes used
- total international calls made
- total international charge
- number customer service calls made

2.3 Target Variable:

Churn; If the customer has churned (1 = yes; 0 = no) 3. Modelling 4. Evaluation 5. Conclusion

3 3. Data Preparation

3.1 Loading Python packages

In [179]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve, accuracy_score, confusion_mat
import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="xgboost")
```

3.2 Data Loading

In [180]:
```
# Loading the CSV data to pandas data frame
df = pd.read_csv('./data/bigml_59c28831336c6604c800002a.csv')
```

3.3 Data Understanding

In [181]:
```
# Finding the number of columns and rowa in the data set
df.shape
```

Out[181]:
```
(3333, 21)
```

In [182]:
```
# checking the first 5 rows of the data set
df.head()
```

Out[182]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	mi
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	

5 rows × 21 columns

In [183]:
```
# checking the last 5 rows of the data set
df.tail()
```

Out[183]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge
3328	AZ	192	415	414-4276	no	yes	36	156.2	77	26.55	...	126	18.32	279.1	83	12.56
3329	WV	68	415	370-3271	no	no	0	231.1	57	39.29	...	55	13.04	191.3	123	8.61
3330	RI	28	510	328-8230	no	no	0	180.8	109	30.74	...	58	24.55	191.9	91	8.64
3331	CT	184	510	364-6381	yes	no	0	213.8	105	36.35	...	84	13.57	139.2	137	6.26
3332	TN	74	415	400-4344	no	yes	25	234.4	113	39.85	...	82	22.60	241.4	77	10.86

5 rows × 21 columns

In [184]:
```
# checking the summary information of the data set in terms of columns, missing values and data types
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
```

In [185]:
```
# Dropping  phone number Column which will not be used in the modelling
df= df.drop( columns= ['phone number'], axis=1)
```

In [186]:
```
# printing all the column names
df.columns
```

Out[186]:
```
Index(['state', 'account length', 'area code', 'international plan',
       'voice mail plan', 'number vmail messages', 'total day minutes',
       'total day calls', 'total day charge', 'total eve minutes',
       'total eve calls', 'total eve charge', 'total night minutes',
       'total night calls', 'total night charge', 'total intl minutes',
       'total intl calls', 'total intl charge', 'customer service calls',
       'churn'],
      dtype='object')
```

In [187]:
```
# printing the unique values in all the columns

for column in df.columns:
    print (column,df[column].unique())
    print ('-'*80)
```

```
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']
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
area code [415 408 510]
--------------------------------------------------------------------------------
international plan ['no' 'yes']
--------------------------------------------------------------------------------
voice mail plan ['yes' 'no']
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
total day minutes [265.1 161.6 243.4 ... 321.1 231.1 180.8]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
total day charge [45.07 27.47 41.38 ... 54.59 39.29 30.74]
--------------------------------------------------------------------------------
total eve minutes [197.4 195.5 121.2 ... 153.4 288.8 265.9]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
total eve charge [16.78 16.62 10.3  ... 13.04 24.55 22.6 ]
--------------------------------------------------------------------------------
total night minutes [244.7 254.4 162.6 ... 280.9 120.1 279.1]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
total intl calls [ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13 18 14 16 20 17]
--------------------------------------------------------------------------------
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]
--------------------------------------------------------------------------------
customer service calls [1 0 2 3 4 5 7 9 6 8]
--------------------------------------------------------------------------------
churn [False  True]
--------------------------------------------------------------------------------
```

In [188]:

```
# Identify categorical and numerical columns
categorical_features = [col for col in df.columns if df[col].dtype == 'object' or df[col].dtype.name == 'bo
numerical_features = [col for col in df.columns if df[col].dtype == 'int64' or df[col].dtype.name == 'float

# Create separate DataFrames for categorical and numerical columns
categorical_df = df[categorical_features]
numerical_df = df[numerical_features]
```

In [189]:

```
categorical_df.head()
```

Out[189]:

	state	international plan	voice mail plan	churn
0	KS	no	yes	False
1	OH	no	yes	False
2	NJ	no	no	False
3	OH	yes	no	False
4	OK	yes	no	False

In [190]:

```
# checking the Class distribution of the Target column
churn_counts = df['churn'].value_counts()

# Calculate the percentage of each value
churn_percentage = df['churn'].value_counts(normalize=True) * 100

print(churn_counts)
print('-'*30)
print(churn_percentage)
```

```
churn
False    2850
True      483
Name: count, dtype: int64
------------------------------
churn
False    85.508551
True     14.491449
Name: proportion, dtype: float64
```

In [191]:
```
# checking duplicate values
df.duplicated().sum()
```

Out[191]:
```
0
```

In [192]:
```
# Checking null values
df.isnull().sum()
```

Out[192]:
```
state                     0
account length            0
area code                 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
```

3.4 Insights

1. Dropped the Phone number columns because it will not be used in the model
2. No missing values
3. No duplicated values
4. Class imbalance in the target column (churn) with 85.5% of False and 14.5% of True. This can be addressed by using oversampling techniques.

4 4. Exploratory Data Analysis

In [193]:
```
df.head(2)
```

Out[193]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	to i minut
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	1(
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	1:

In [194]:
```
# Checking the descriptive statistics
df.describe().T
```

Out[194]:

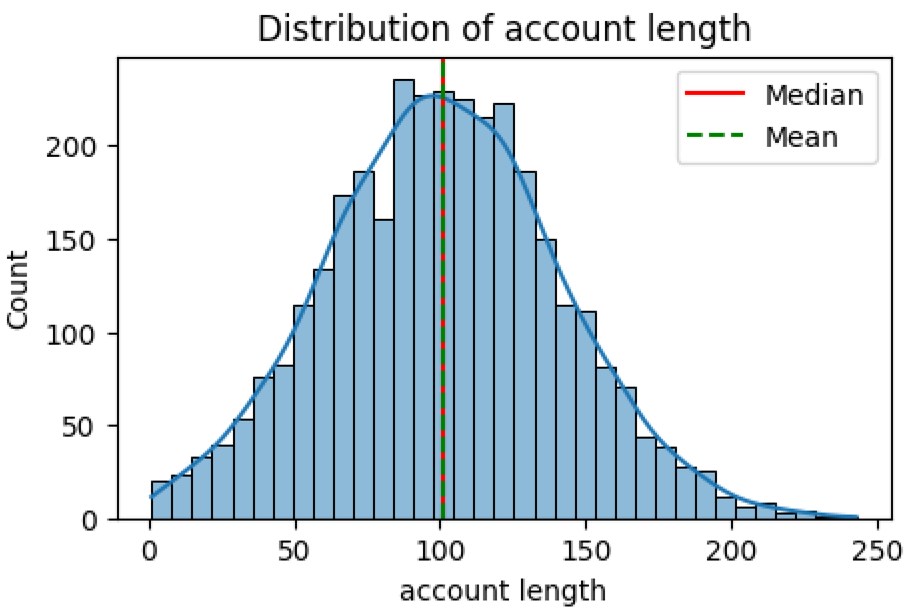
	count	mean	std	min	25%	50%	75%	max
account length	3333.0	101.064806	39.822106	1.00	74.00	101.00	127.00	243.00
area code	3333.0	437.182418	42.371290	408.00	408.00	415.00	510.00	510.00
number vmail messages	3333.0	8.099010	13.688365	0.00	0.00	0.00	20.00	51.00
total day minutes	3333.0	179.775098	54.467389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

4.1 Numerical features analysis

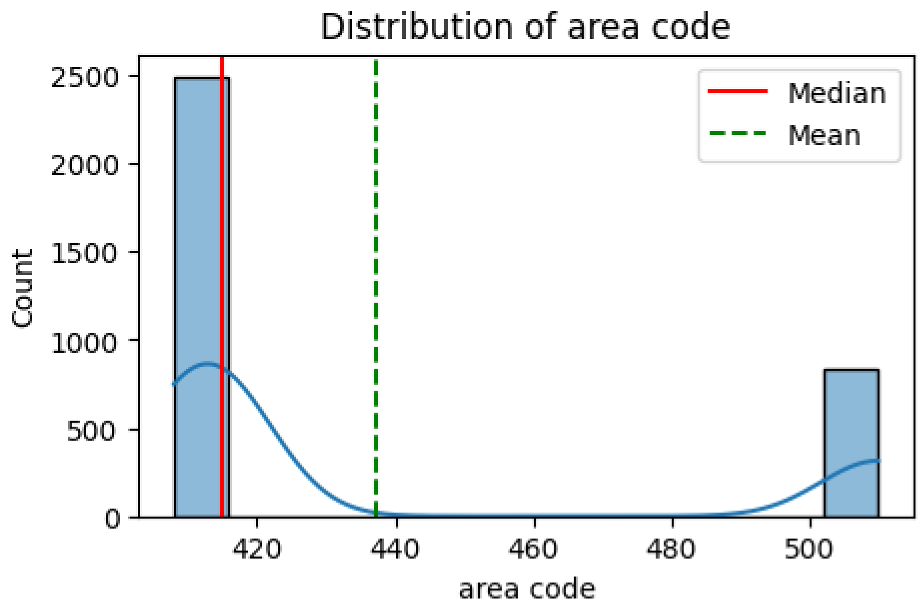
Distribution of the numerical features

In [195]:
```
def plot_histogram (df,column_name):
  plt.figure(figsize=(5, 3))
  sns.histplot(df[column_name], kde = True)
  plt.title (f'Distribution of {column_name}')

# calculate  the mean and median values for the columns
  col_mean = np.mean(df[column_name].mean())
  col_median = np.median(df[column_name].median())

# Vertical lines for mean and median label='Mean')
  plt.axvline(col_median, color='red', linestyle= '-', label='Median')
  plt.axvline(col_mean, color='green', linestyle= '--', label='Mean')

  plt.legend(loc='best')
  plt.show()
```

In [196]:
```
plot_histogram(df,'account length')
```

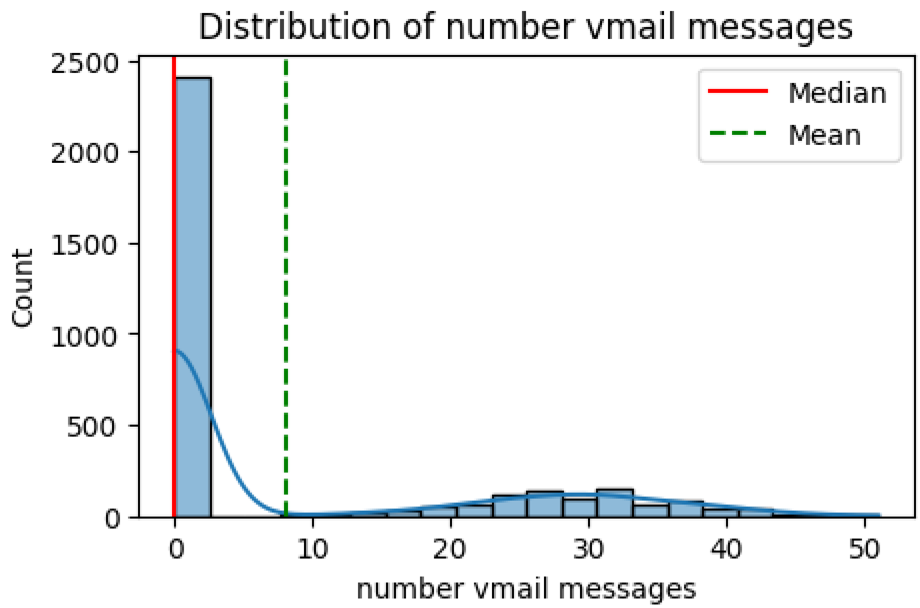


In [197]:
```
plot_histogram(df,'area code')
```

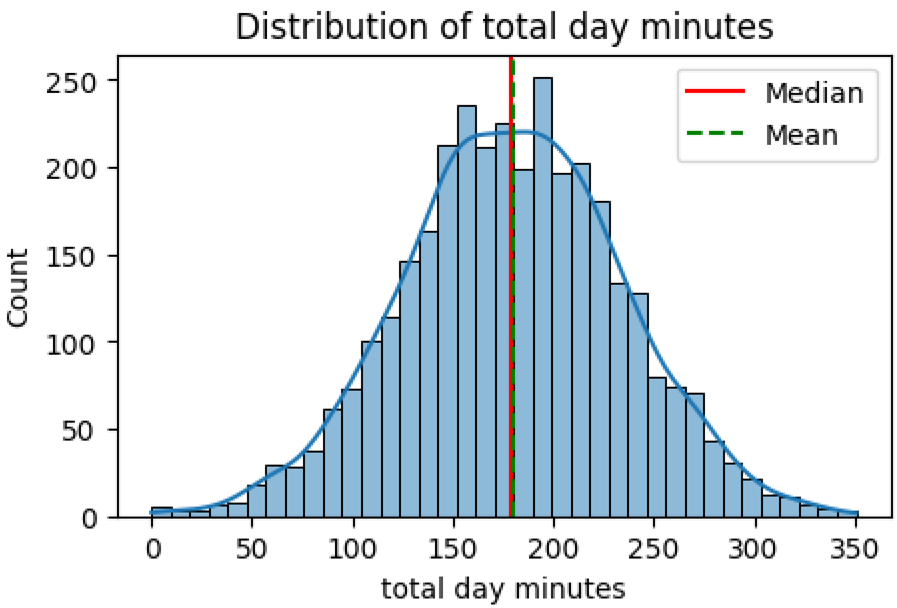


In [198]:

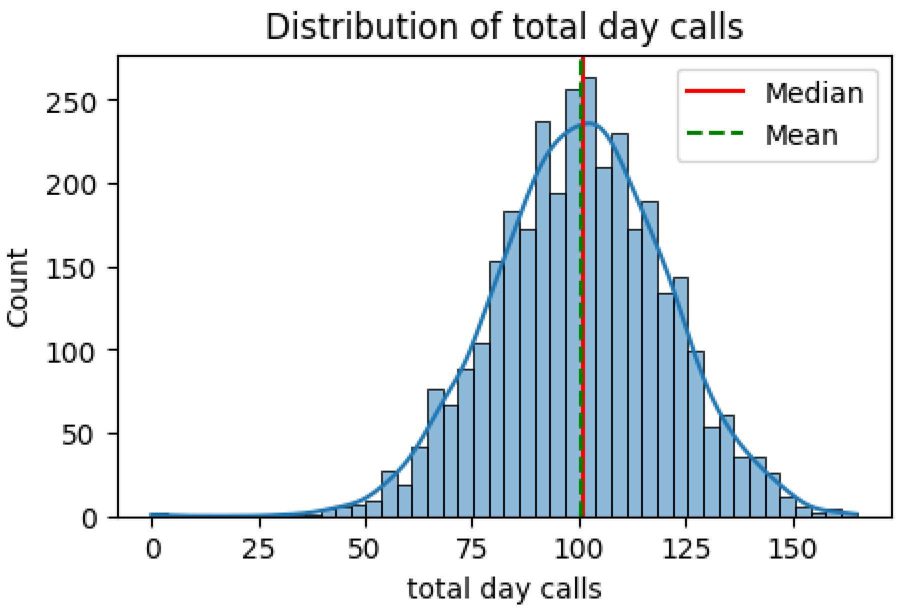
```
plot_histogram(df,'number vmail messages')
```



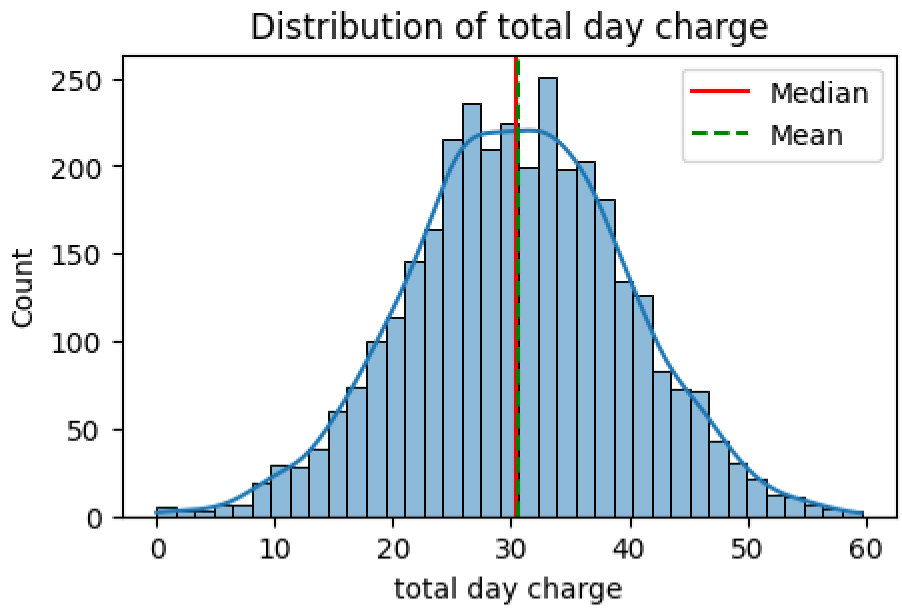
In [199]:
```
plot_histogram(df,'total day minutes')
```



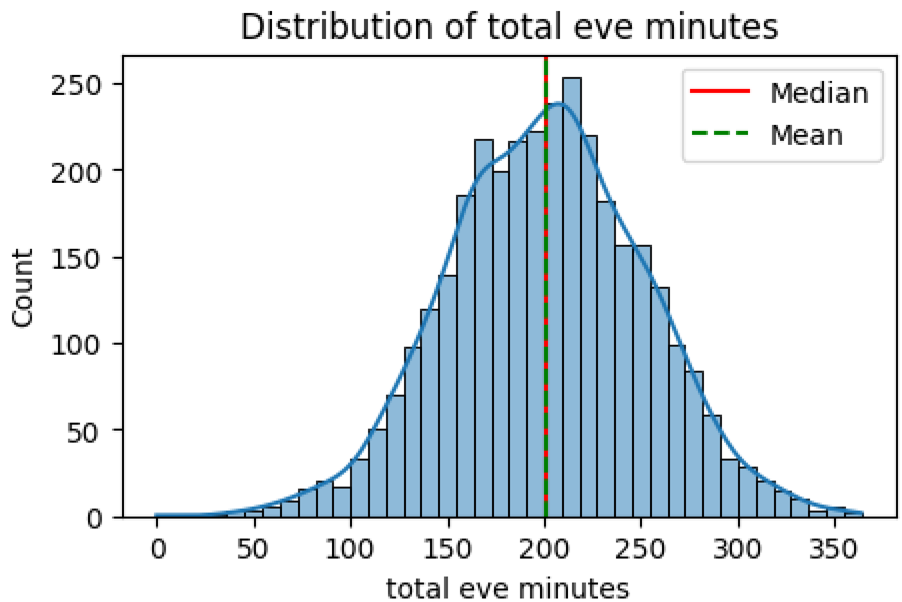
In [200]:
```
plot_histogram(df,'total day calls')
```



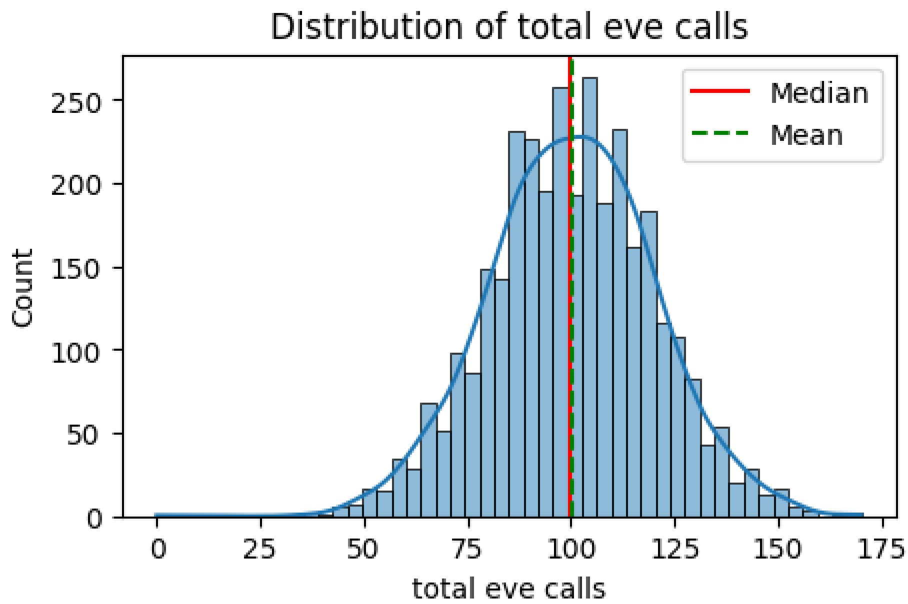
In [201]:
```
plot_histogram(df,'total day charge')
```



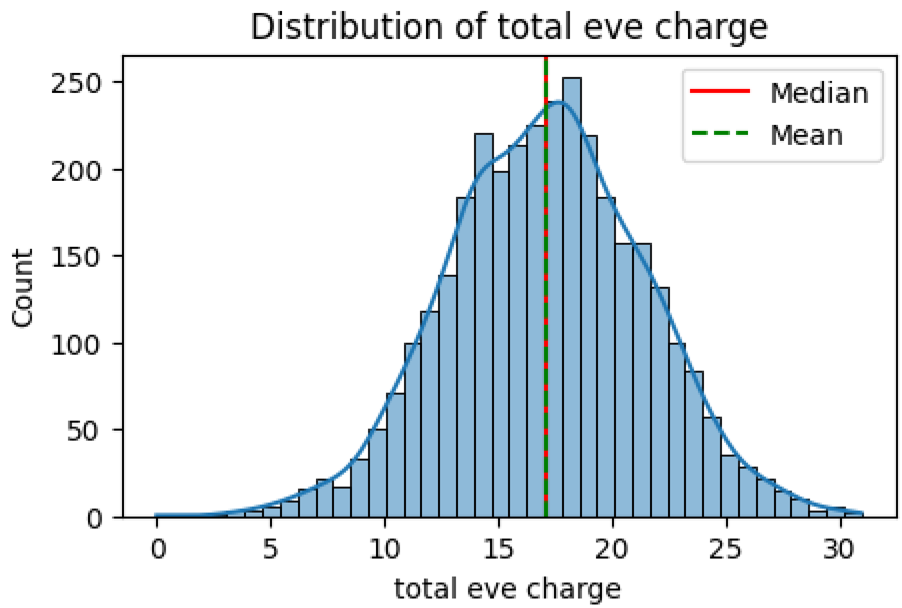
In [202]:
```
plot_histogram(df,'total eve minutes')
```



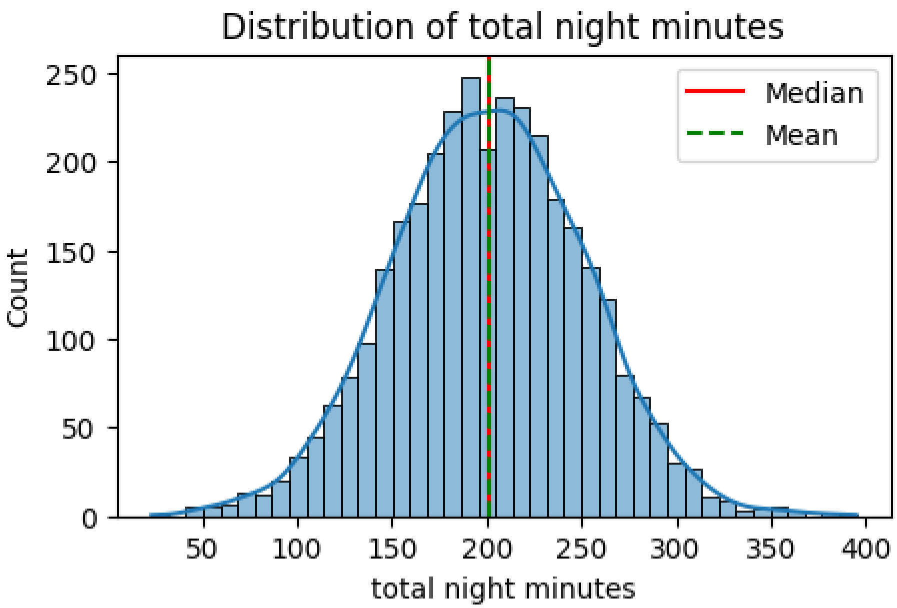
In [203]:
```
plot_histogram(df,'total eve calls')
```



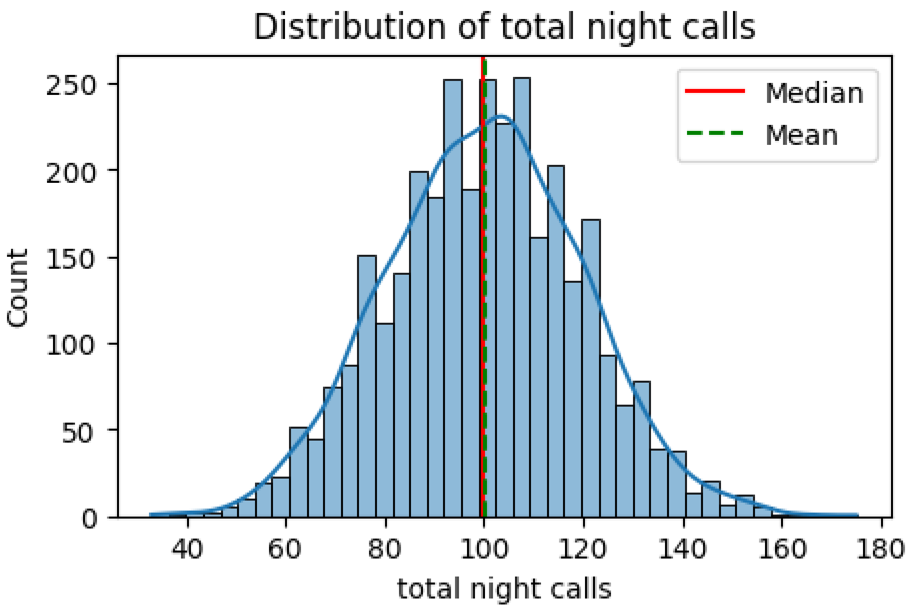
In [204]:
```
plot_histogram(df,'total eve charge')
```



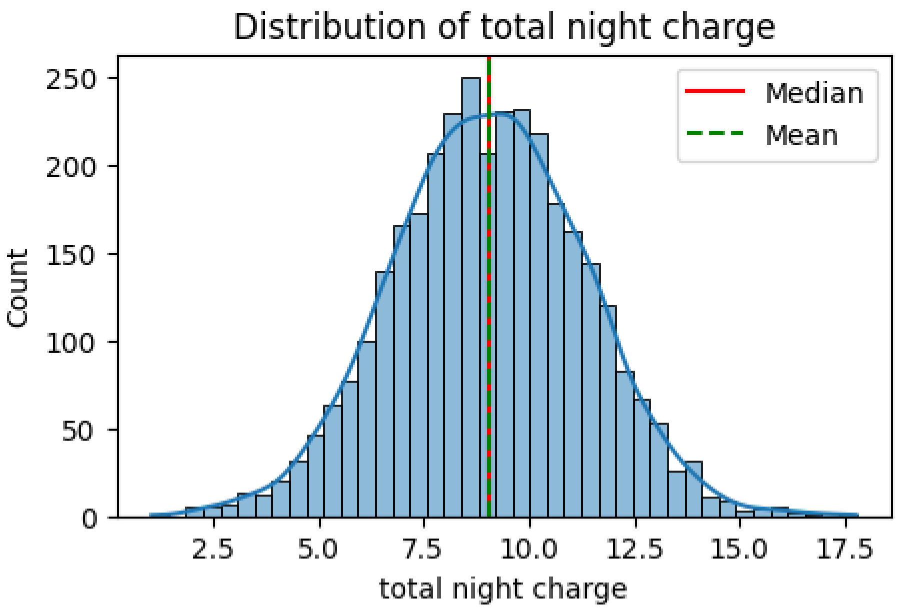
In [205]:
```
plot_histogram(df,'total night minutes')
```



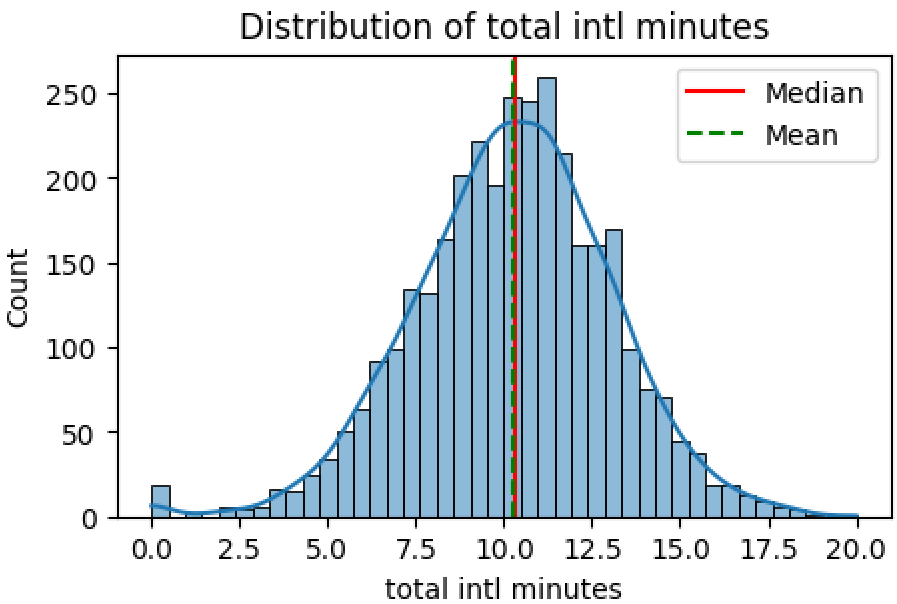
In [206]:
```
plot_histogram(df,'total night calls')
```



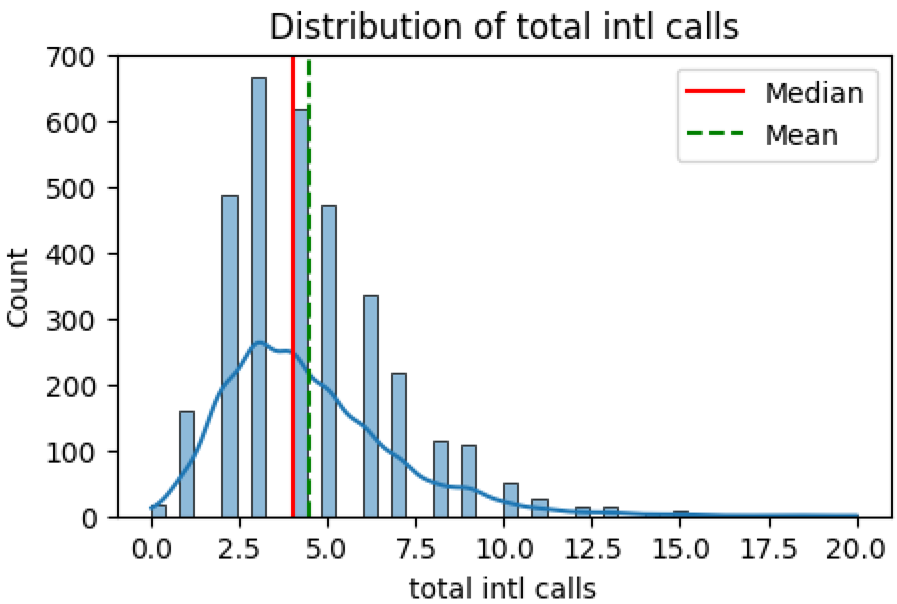
In [207]:
```
plot_histogram(df,'total night charge')
```



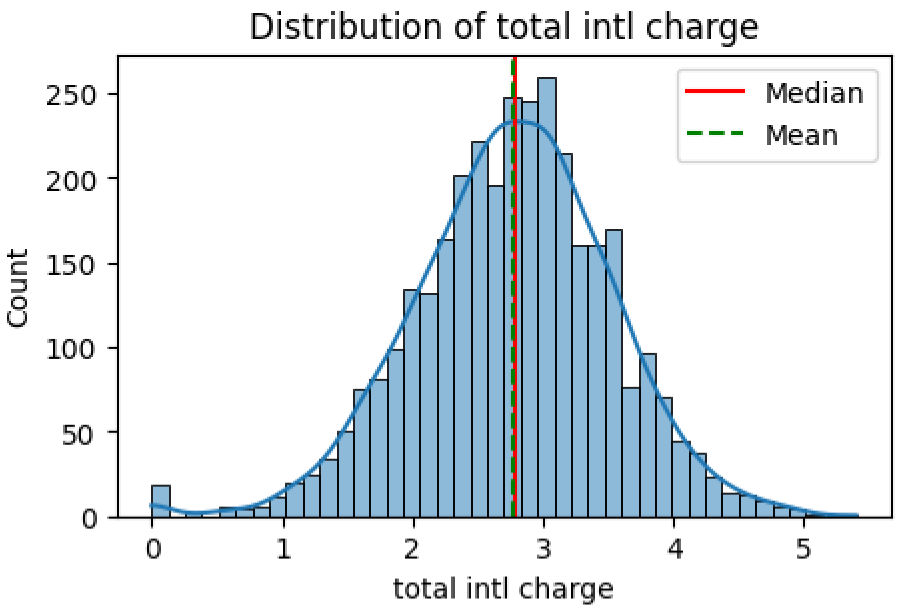
In [208]:
```
plot_histogram(df,'total intl minutes')
```



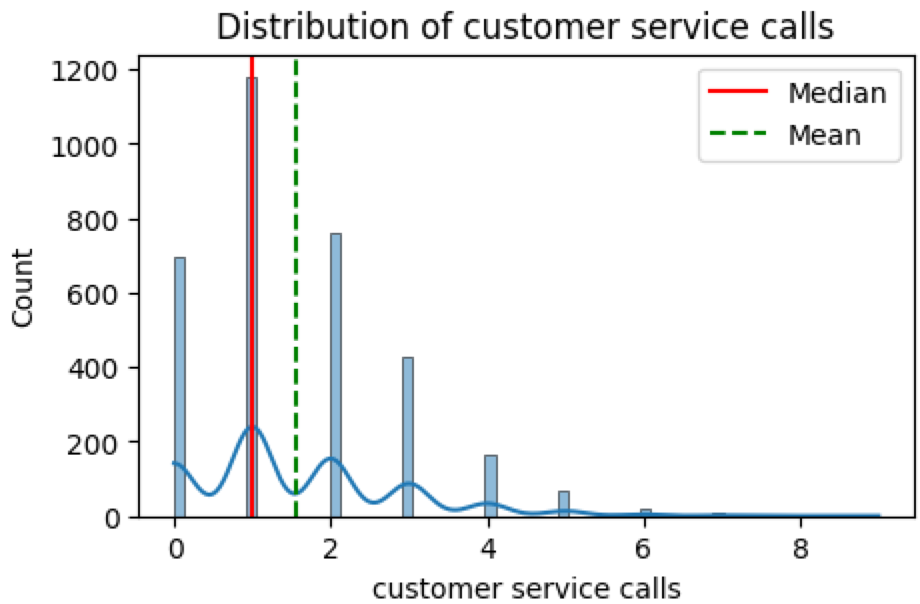
In [209]:
```
plot_histogram(df,'total intl calls')
```



In [210]:
```
plot_histogram(df,'total intl charge')
```



In [211]:
```
plot_histogram(df,'customer service calls')
```



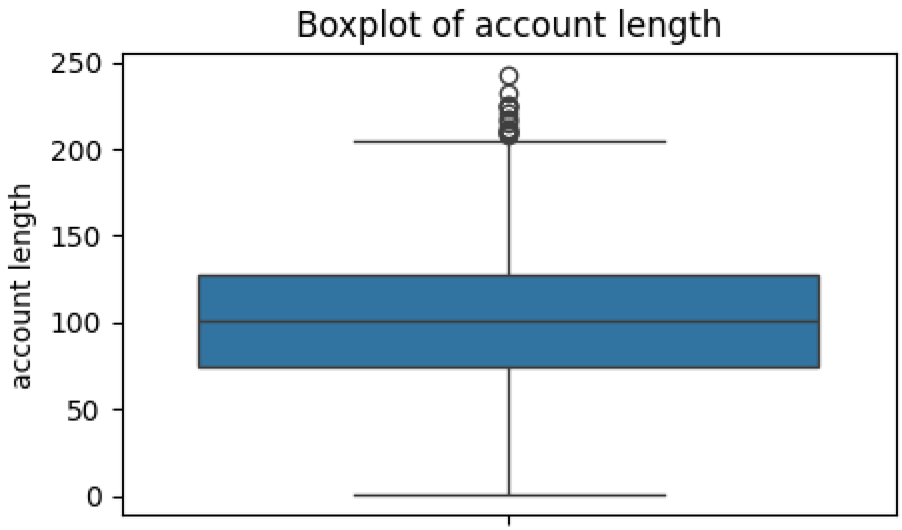
4.1.1 Insights from the histograms

- Several features have a normal distribution : account length, Total day minutes, Total day calls, Total day charge, Total eve minutes, Total eve calls, Total eve charges, Total night minutes, Total night calls, Total night charges, Total intel charges, Total intel calls
- Left Skew Area code, Number of voice mail messages, Total international calls ,Customer service calls

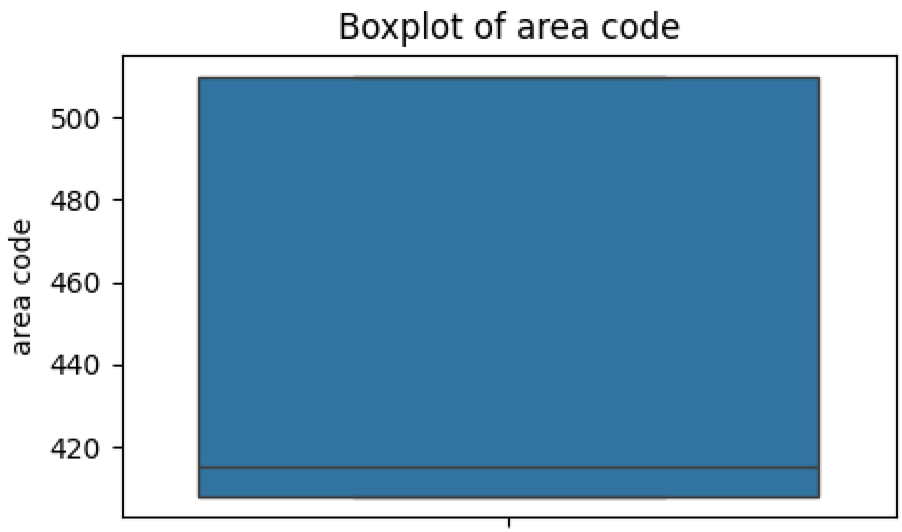
4.1.2 Box plot for numerical features

In [212]:
```
def plot_boxplot(df, column_name):
    plt.figure(figsize=(5,3))
    sns.boxplot(y=df[column_name])
    plt.title(f'Boxplot of {column_name}')
    plt.ylabel(column_name)
    plt.show
```

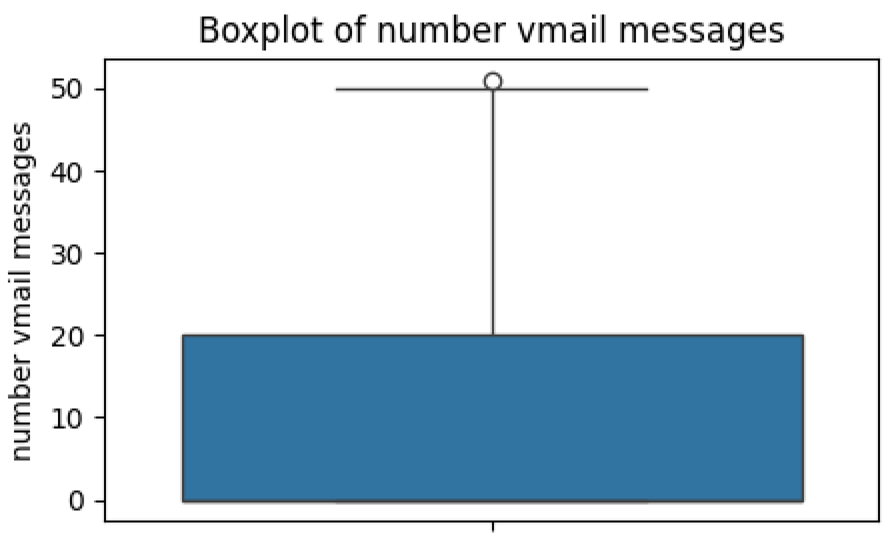
In [213]:
```
plot_boxplot(df,'account length')
```



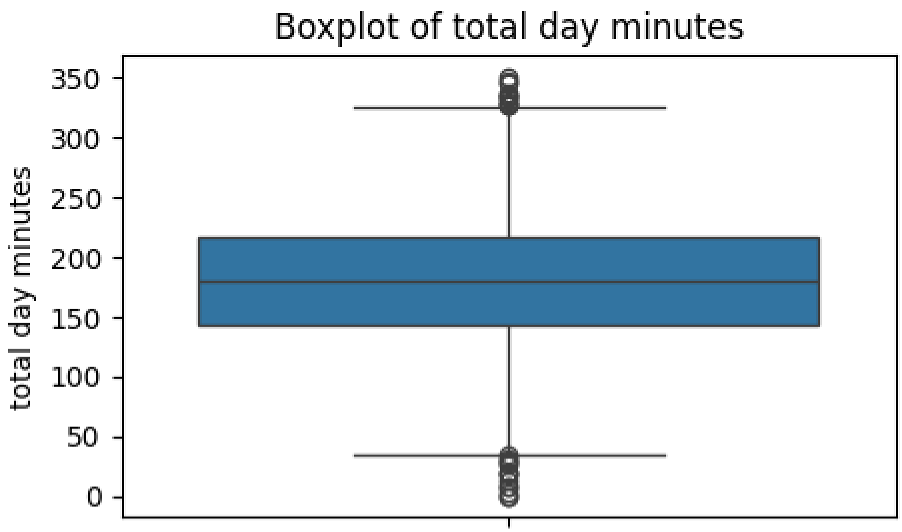
In [214]:
```
plot_boxplot(df,'area code')
```



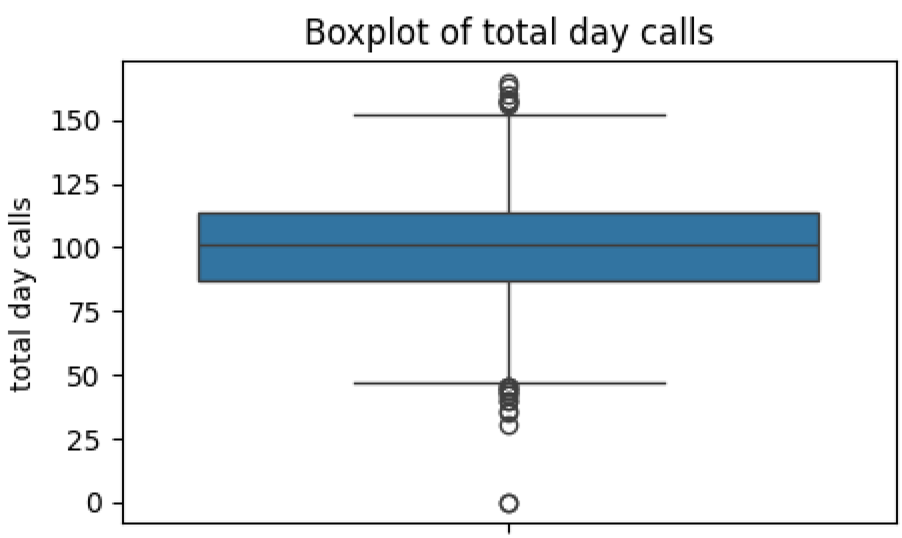
In [215]:
```
plot_boxplot(df,'number vmail messages')
```



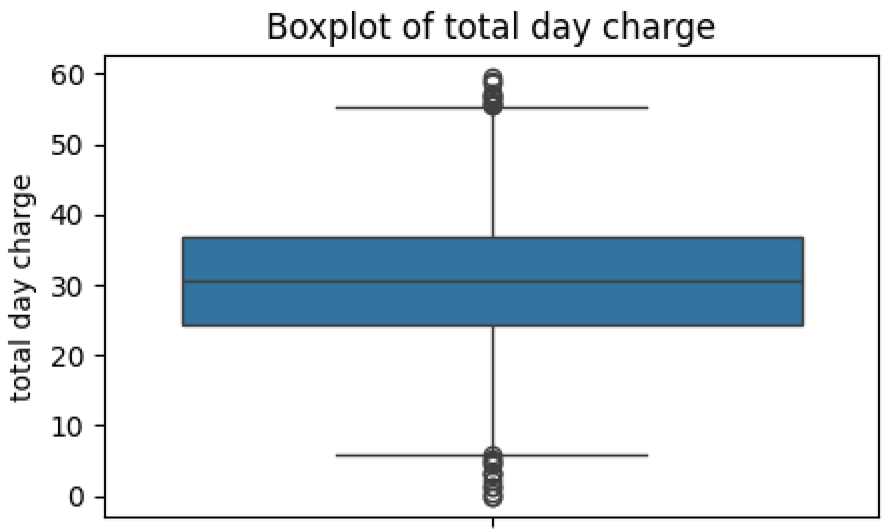
In [216]:
```
plot_boxplot(df,'total day minutes')
```



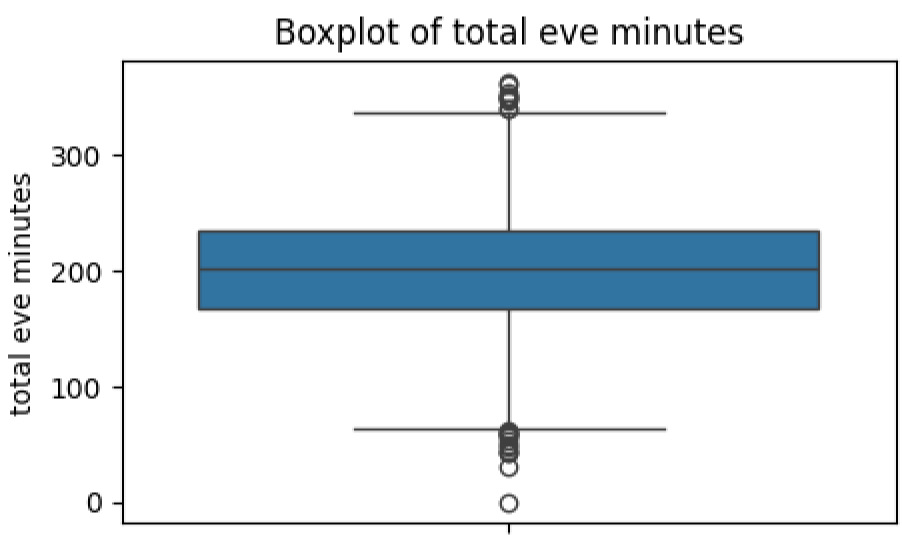
In [217]:
```
plot_boxplot(df,'total day calls')
```



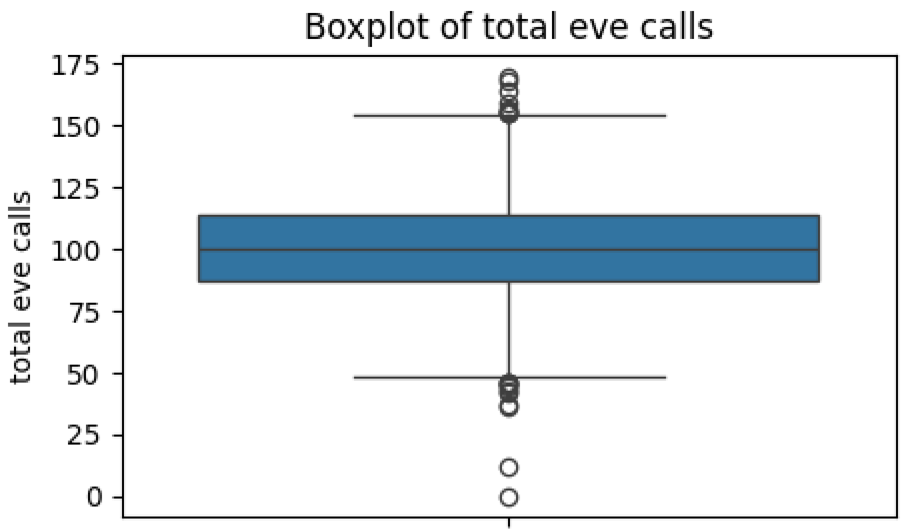
In [218]:
```
plot_boxplot(df,'total day charge')
```



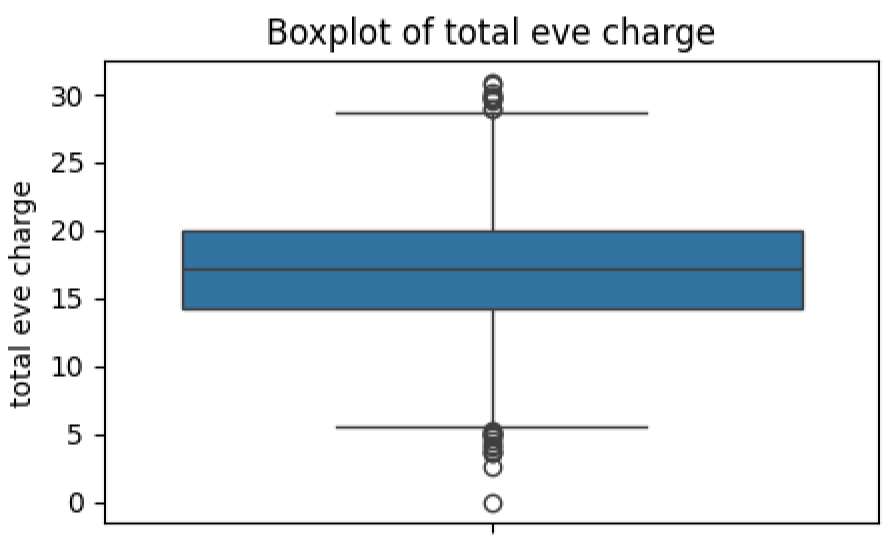
In [219]:
```
plot_boxplot(df,'total eve minutes')
```



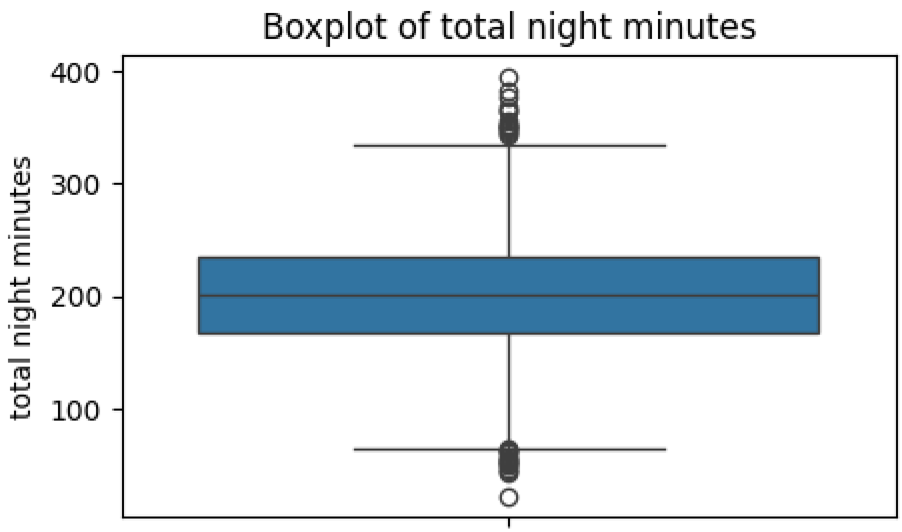
In [220]:
```
plot_boxplot(df,'total eve calls')
```



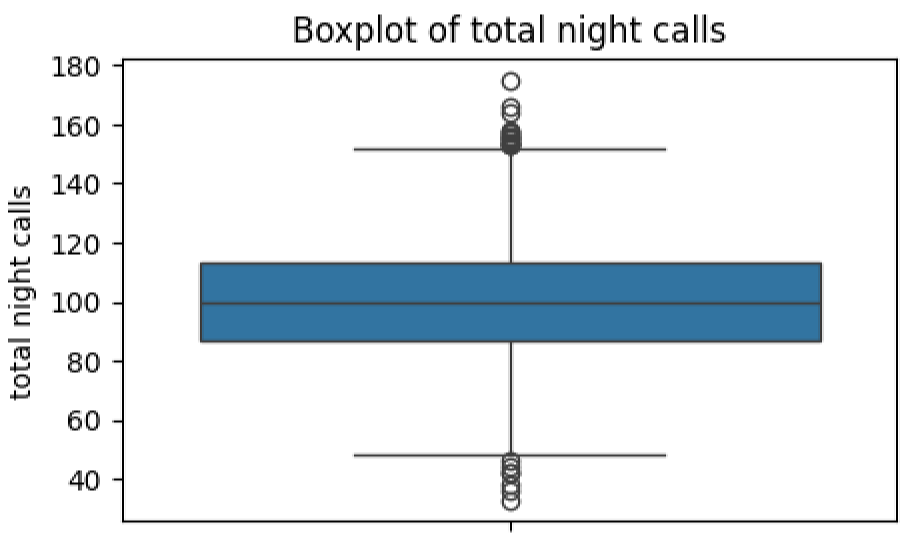
In [221]:
```
plot_boxplot(df,'total eve charge')
```



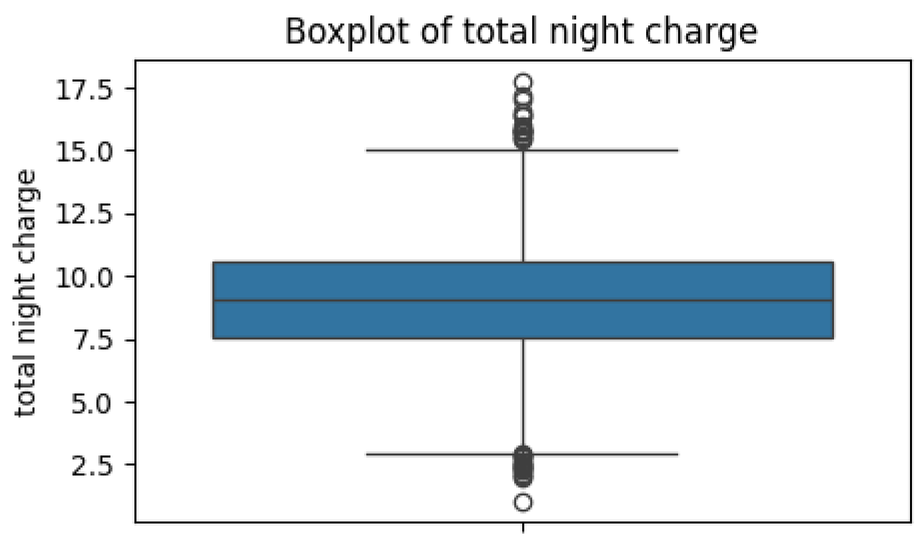
In [222]:
```
plot_boxplot(df,'total night minutes')
```



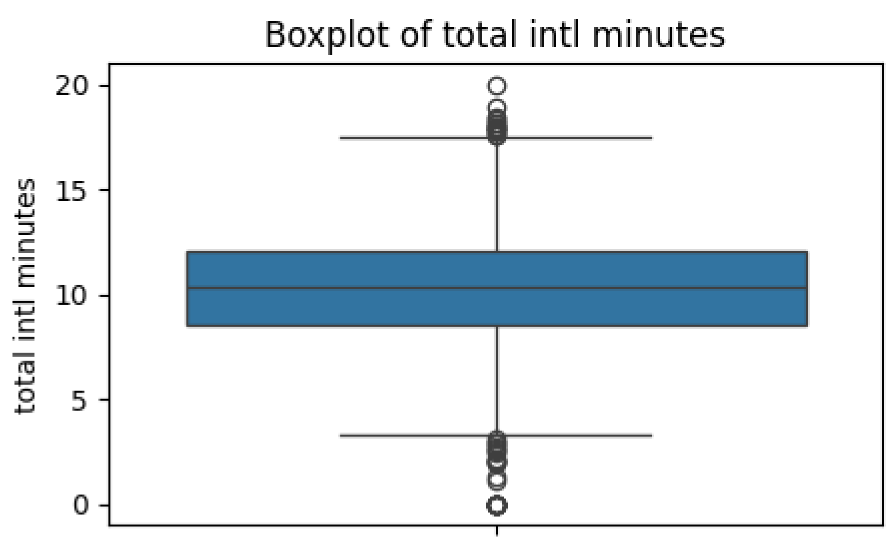
In [223]:
```
plot_boxplot(df,'total night calls')
```



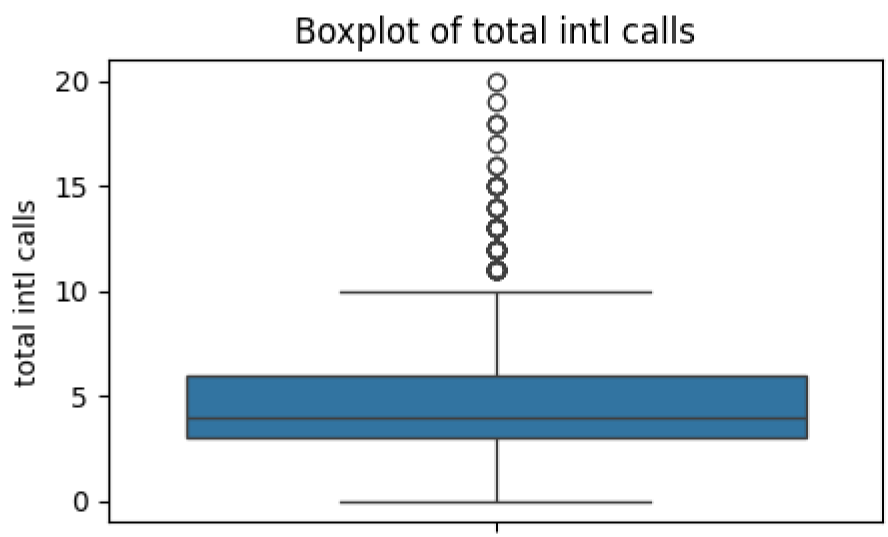
In [224]:
```
plot_boxplot(df,'total night charge')
```



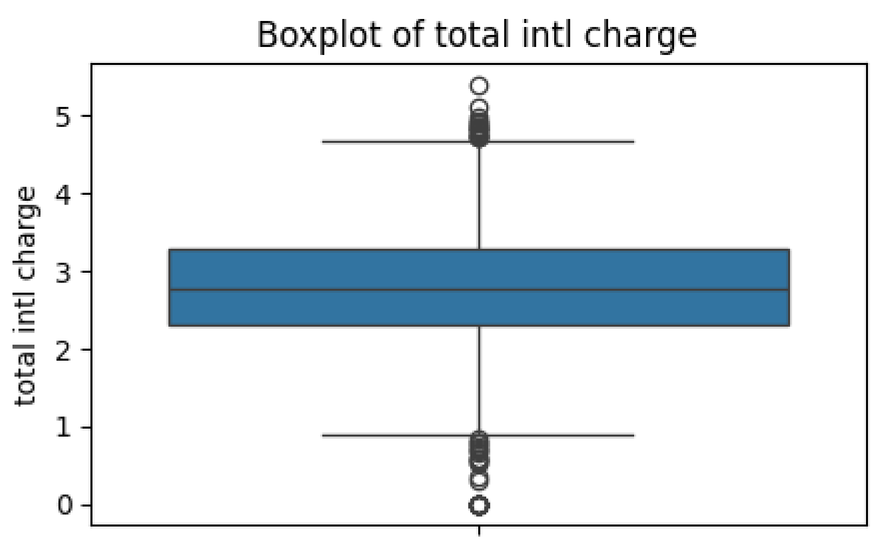
In [225]:
```
plot_boxplot(df,'total intl minutes')
```



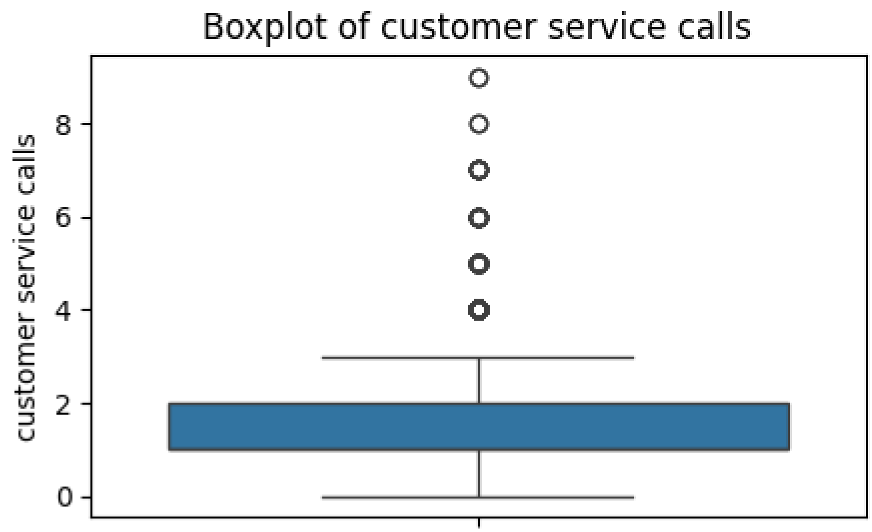
In [226]:
```
plot_boxplot(df,'total intl calls')
```



In [227]:
```
plot_boxplot(df,'total intl charge')
```



In [228]:
```
plot_boxplot(df,'customer service calls')
```

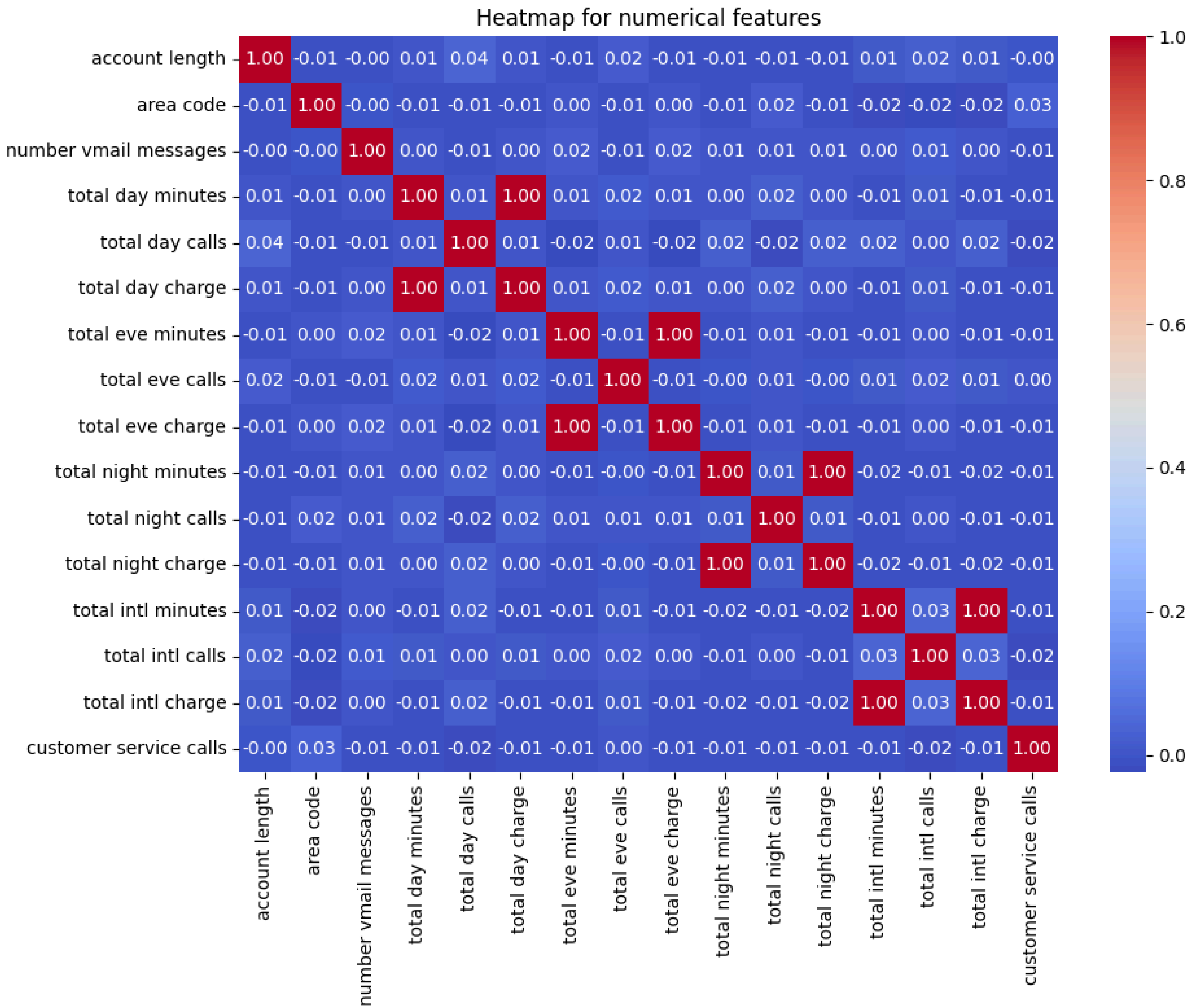


4.2 Insights from the boxplots

- The numerical features have outliers apart from area code, and number of vcall messages

4.2.1 Correlation heatmap for numerical features

In [229]:
```
plt.figure(figsize=(10,8))
sns.heatmap(numerical_df.corr(),annot= True,cmap='coolwarm', fmt='.2f')
plt.title('Heatmap for numerical features')
plt.tight_layout()
plt.show()
```


Heatmap for numerical features
account length
area code
number vmail messages
total day minutes
total day calls
total day charge
total eve minutes
total eve calls
total eve charge
total night minutes
total night calls
total night charge
total intl minutes
total intl calls
total intl charge
customer service calls
1.0
0.8
0.6
0.4
0.2
0.0

- The numerical featues have low correlations

4.3 Categorical features analysis

In [230]:
```
categorical_df.head()
```

Out[230]:

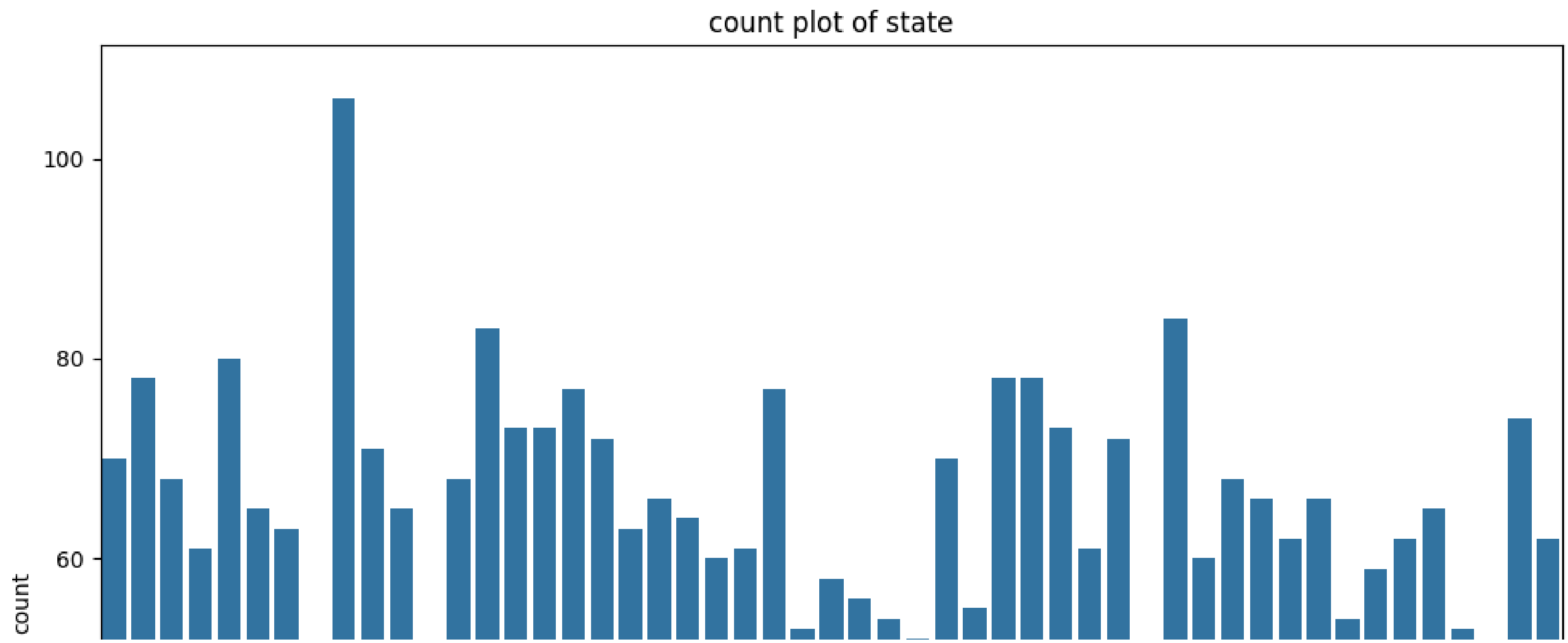
	state	international plan	voice mail plan	churn
0	KS	no	yes	False
1	OH	no	yes	False
2	NJ	no	no	False
3	OH	yes	no	False
4	OK	yes	no	False

In [231]:
```
categorical_features
```

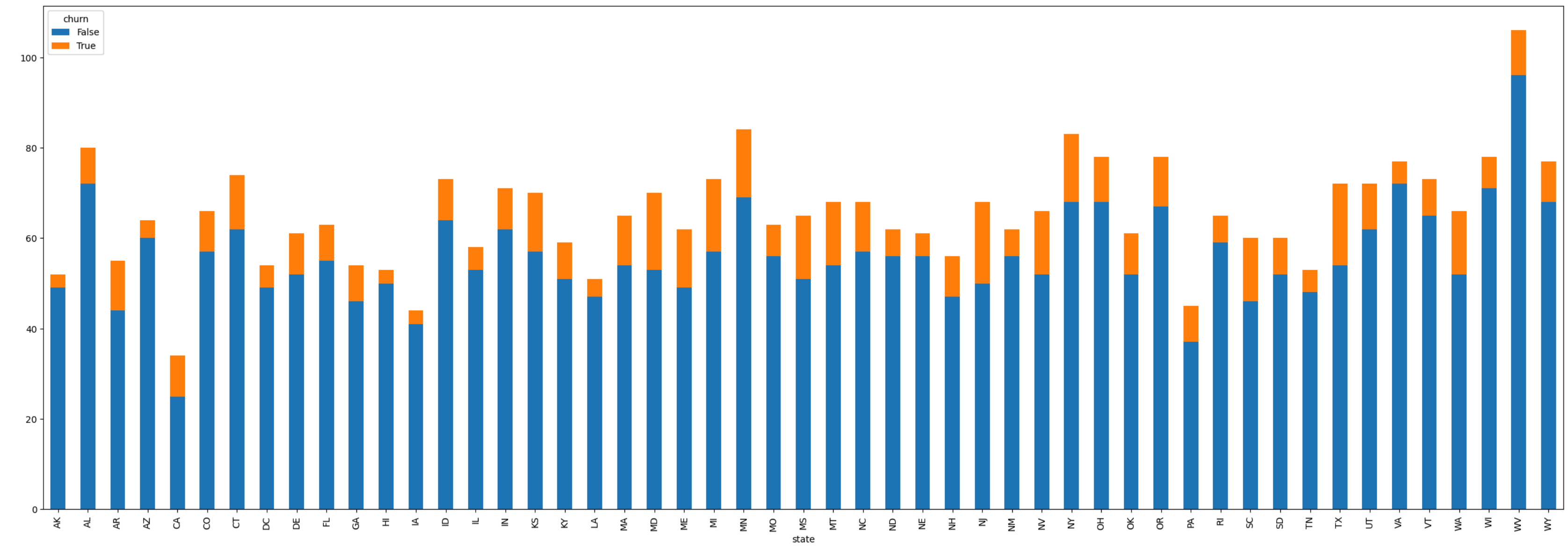
Out[231]:
```
['state', 'international plan', 'voice mail plan', 'churn']
```

In [232]:

```
for col in categorical_features:
    plt.figure(figsize=(10,8))
    sns.countplot(x=col, data=categorical_df)
    plt.title(f'count plot of {col}')
    plt.xticks(rotation=45)  # Rotate x-axis ticks
    plt.tight_layout()
    plt.show()
```



In [233]:
```python
df.groupby(["state", "churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(30,10));
```



5 5. Data Preprocessing

5.0.1 Label encoding of the Target column

In [234]:
```python
df['churn']= df['churn'].astype(int)
```

In [235]:
```python
df.head(2)
```

Out[235]:

	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	to i minut
0	KS	128	415	no	yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	1(
1	OH	107	415	no	yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	1:

In [236]:
```
# separate feature and target
X = df.drop(columns=['churn'],axis =1)
y = df['churn']
```

5.0.2 Hot encoding for categorical features

In [237]:
```
# identifying categorical columns and numerical columns
categorical_columns = ['state', 'international plan', 'voice mail plan', 'area code']
numerical_columns = X.columns.difference(categorical_columns)
X_categorical = df[categorical_columns]
```

In [238]:
```
# One-Hot Encoding Categorical columns
ohe = OneHotEncoder(drop='first', sparse_output=False)  # drop='first' avoids multicollinearity
X_cat_encoded = ohe.fit_transform(X_categorical)
X_cat_encoded
```

Out[238]:
```
array([[0., 0., 0., ..., 1., 1., 0.],
       [0., 0., 0., ..., 1., 1., 0.],
       [0., 0., 0., ..., 0., 1., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 1., 1., 0.]])
```

In [239]:

```
# Retrieve the feature names after encoding for proper labeling
encoded_feature_names = ohe.get_feature_names_out(categorical_columns)

# Convert the encoded data to a DataFrame with appropriate column names
X_cat_encoded_df = pd.DataFrame(X_cat_encoded, columns=encoded_feature_names, index=df.index)
X_cat_encoded_df.head()
```

Out[239]:

	state_AL	state_AR	state_AZ	state_CA	state_CO	state_CT	state_DC	state_DE	state_FL	state_GA	...	state_VA	state_VT	state_
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

5 rows × 54 columns

In [240]:
```
# combining categorical and numerical features
transformed_feature_names = np.concatenate([encoded_feature_names, numerical_columns])
transformed_feature_names
```

Out[240]:
```
array(['state_AL', 'state_AR', 'state_AZ', 'state_CA', 'state_CO',
       'state_CT', 'state_DC', 'state_DE', 'state_FL', 'state_GA',
       'state_HI', 'state_IA', 'state_ID', 'state_IL', 'state_IN',
       'state_KS', 'state_KY', 'state_LA', 'state_MA', 'state_MD',
       'state_ME', 'state_MI', 'state_MN', 'state_MO', 'state_MS',
       'state_MT', 'state_NC', 'state_ND', 'state_NE', 'state_NH',
       'state_NJ', 'state_NM', 'state_NV', 'state_NY', 'state_OH',
       'state_OK', 'state_OR', 'state_PA', 'state_RI', 'state_SC',
       'state_SD', 'state_TN', 'state_TX', 'state_UT', 'state_VA',
       'state_VT', 'state_WA', 'state_WI', 'state_WV', 'state_WY',
       'international plan_yes', 'voice mail plan_yes', 'area code_415',
       'area code_510', 'account length', 'customer service calls',
       'number vmail messages', 'total day calls', 'total day charge',
       'total day minutes', 'total eve calls', 'total eve charge',
       'total eve minutes', 'total intl calls', 'total intl charge',
       'total intl minutes', 'total night calls', 'total night charge',
       'total night minutes'], dtype=object)
```

5.0.3 Standardization of numerical columns

In [241]:
```
#Instanciate standardization
scaler = StandardScaler()
X_numerical= scaler.fit_transform(X[numerical_columns])
X_numerical
```

Out[241]:
```
array([[ 0.67648946, -0.42793202,  1.23488274, ..., -0.46549436,
         0.86602851,  0.86674322],
       [ 0.14906505, -0.42793202,  1.30794844, ...,  0.14782467,
         1.05938994,  1.05857074],
       [ 0.9025285 , -1.1882185 , -0.59175986, ...,  0.19893459,
        -0.75557074, -0.75686906],
       ...,
       [-1.83505538,  0.33235445, -0.59175986, ..., -0.46549436,
        -0.17548645, -0.1774313 ],
       [ 2.08295458,  0.33235445, -0.59175986, ...,  1.88556193,
        -1.22139599, -1.21962822],
       [-0.67974475, -1.1882185 ,  1.23488274, ..., -1.18103324,
         0.80010984,  0.80148231]])
```

In [242]:
```
# convert to dataframe
X_numerical_df = pd.DataFrame(X_numerical, columns= numerical_columns,index=X.index)
X_numerical_df.head(2)
```

Out[242]:

	account length	customer service calls	number vmail messages	total day calls	total day charge	total day minutes	total eve calls	total eve charge	total eve minutes	total intl calls	total intl charge	total intl minutes	t n c
0	0.676489	-0.427932	1.234883	0.476643	1.567036	1.566767	-0.055940	-0.070427	-0.07061	-0.601195	-0.085690	-0.085008	-0.465
1	0.149065	-0.427932	1.307948	1.124503	-0.334013	-0.333738	0.144867	-0.107549	-0.10808	-0.601195	1.241169	1.240482	0.147

5.0.4 Combining procesed Data frames

In [243]:
```
X_processed = pd.concat([X_cat_encoded_df, X_numerical_df], axis=1)
X_processed
```

Out[243]:

	state_AL	state_AR	state_AZ	state_CA	state_CO	state_CT	state_DC	state_DE	state_FL	state_GA	...	total day minutes	total eve calls
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.566767	-0.055940
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.333738	0.144867
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.168304	0.496279
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	2.196596	-0.608159
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.240090	1.098699
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3328	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	-0.432895	1.299506
3329	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.942447	-2.264816
3330	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.018820	-2.114211
3331	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	...	0.624778	-0.808966
3332	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	1.003042	-0.909370

3333 rows × 69 columns

5.1 Train and test data split

In [244]:
```
# Split  train and test data
X_train, X_test, y_train, y_test = train_test_split(X_processed, y, test_size=0.2, random_state=42)
```

In [245]:
```
print(f'Train data shape: {X_train.shape}')

print(f'Test data shape: {X_test.shape}')
```

```
Train data shape: (2666, 69)
Test data shape: (667, 69)
```

In [246]:
```
y_train.value_counts()
```

Out[246]:
```
churn
0    2284
1     382
Name: count, dtype: int64
```

- There is a significant class imbalance

5.1.1 Handling the class imbalance with SMOTE (Synthetic Minority Over-sampling Technique)

In [247]:
```
smote = SMOTE(random_state=42)
X_train_bal, y_train_bal = smote.fit_resample(X_train, y_train)
y_train_bal.value_counts()
```

Out[247]:
```
churn
0    2284
1    2284
Name: count, dtype: int64
```

6 6. Modelling

6.1 6.1. Logistic Regression

6.1.1 Training Logistc Regression model

In [248]:
```
Log_model = LogisticRegression( solver = 'newton-cg', max_iter=2000, random_state= 42,)
Log_model.fit(X_train_bal,y_train_bal)
```

Out[248]:
```
LogisticRegression(max_iter=2000, random_state=42, solver='newton-cg')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

6.1.2 Making a prediction

In [249]:
```
y_pred =Log_model.predict(X_test)
```

6.1.3 Evaluate logistic model

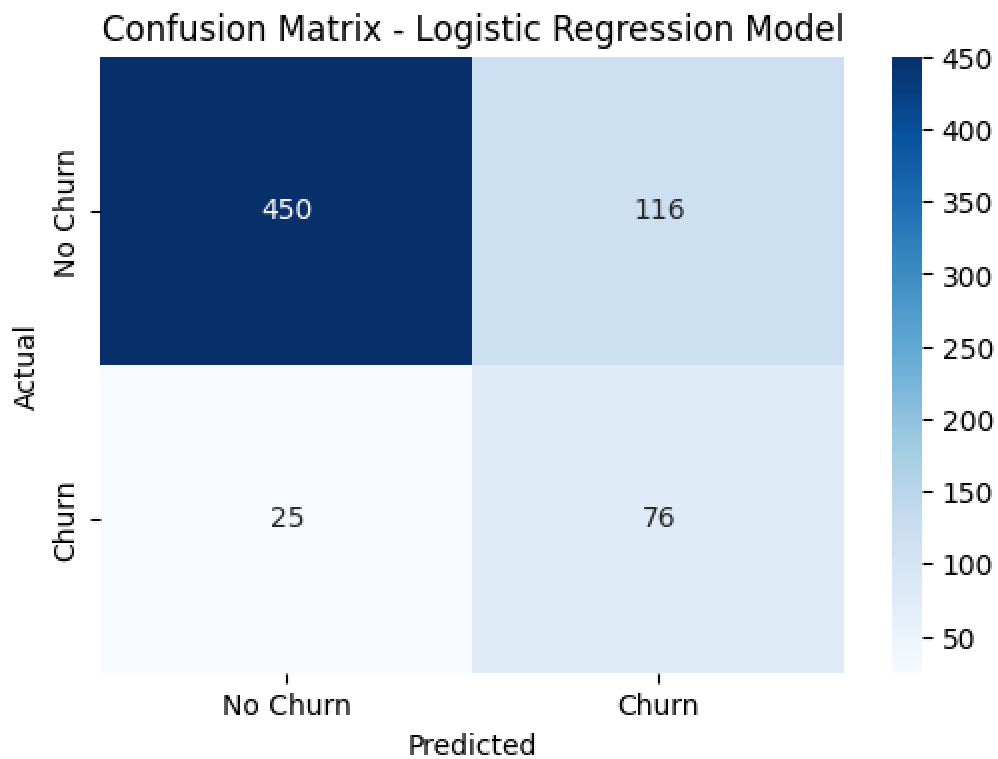
In [250]:
```
accuracy= accuracy_score(y_test,y_pred)
confusion = confusion_matrix(y_test,y_pred)
Report = classification_report(y_test,y_pred)
y_proba_logreg = Log_model.predict_proba(X_test)[:, 1]

roc_auc= roc_auc_score(y_test, y_proba_logreg)
# Print Model Results with Titles and Separators
print("Logistic Regression Model Evaluation")
print('-' * 55)
print("Accuracy:", accuracy)
print('-' * 55)
print("Confusion Matrix:")
print(confusion)
print('-' * 55)
print("Classification Report:")
print(Report)
print('-' * 55)
print("ROC AUC Score:", roc_auc)
print('-' * 55)
```

```
Logistic Regression Model Evaluation
-------------------------------------------------------
Accuracy: 0.7886056971514243
-------------------------------------------------------
Confusion Matrix:
[[450 116]
 [ 25  76]]
-------------------------------------------------------
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.80      0.86       566
           1       0.40      0.75      0.52       101

    accuracy                           0.79       667
   macro avg       0.67      0.77      0.69       667
weighted avg       0.86      0.79      0.81       667

-------------------------------------------------------
ROC AUC Score: 0.8307560438022599
-------------------------------------------------------
```

6.1.4 Visualization of Logistic regression Model Evaluation

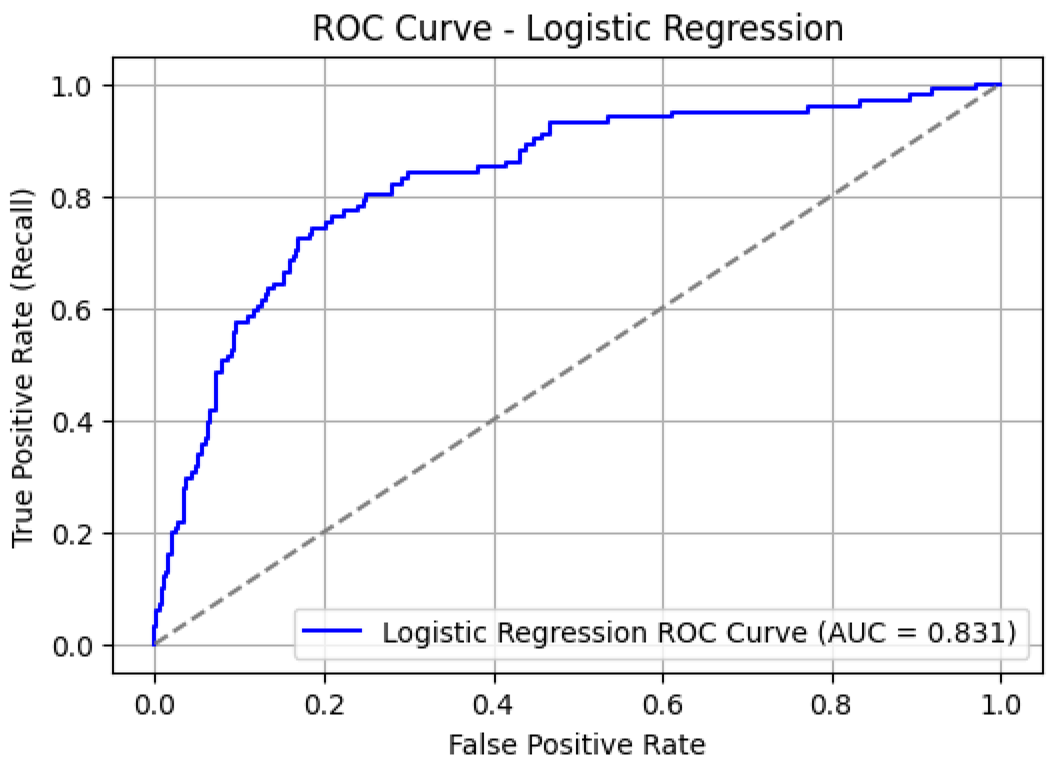
6.1.4.1 Logistic Regression Model Confusion Matrix

In [251]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(confusion, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title("Confusion Matrix - Logistic Regression Model")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



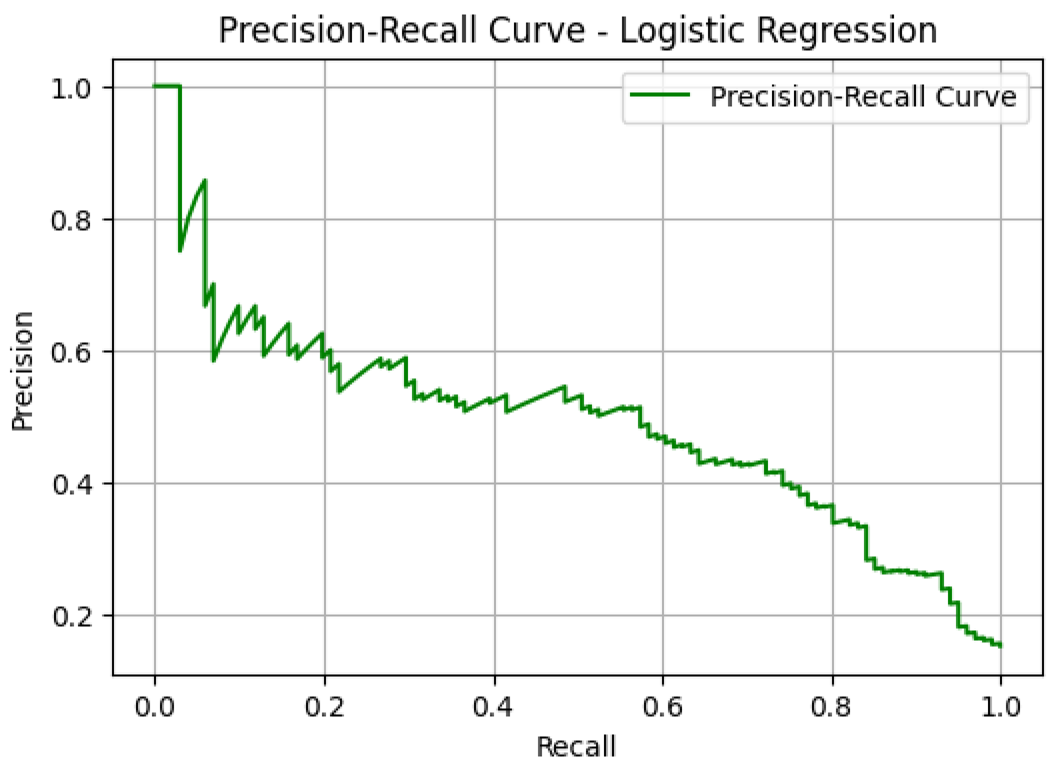
6.1.4.2 Logistic Regression model ROC curve

In [252]:
```
# Calculate the Predicted Probabilities for Positive Class
y_proba_logreg = Log_model.predict_proba(X_test)[:, 1]  # Probabilities for class 1 (churn)

# Calculate ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba_logreg)  # True and false positive rates

# Step 3: Compute ROC-AUC
roc_auc= roc_auc_score(y_test, y_proba_logreg)

#Plot the ROC curve
roc_auc = roc_auc_score(y_test, y_proba_logreg)
fpr, tpr, _ = roc_curve(y_test, y_proba_logreg)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label=f'Logistic Regression ROC Curve (AUC = {roc_auc:.3f})')
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc="lower right")
plt.grid()
plt.show()
```



6.1.4.3 Logic Regression Model Precision-Recall Curve

In [253]:
```
precision, recall, _ = precision_recall_curve(y_test, y_proba_logreg)
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='green', label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - Logistic Regression')
plt.legend()
plt.grid()
plt.show()
```



6.2 Interpretation of the Evaluation of the logistic regression model

- The Accuracy score was 0.79 meaning that the model classified correctly 79% of the test sample
- The ROC AUC score was 0.83
- Confusion Matrix
  - 450 (True Negatives) - customers who did not churn and were correctly predicted not to churn

    - 116 (False Positives) - customers who did not churn but were incorrectly predicted to churn
    - 25 (False Negatives) - customers who churned but were incorrectly predicted not to churn
    - 76 (True Positives) - customers who churned and were correctly predicted to churn
- The model over predicts churn(false positive) and misses churn cases (false negative)
- Precision
    - class 0 : model predicts non churn correct at 94% of the time
    - class 1 : model predicts churn correct at 40% of the time
    - low precison for churn show model has high false positives
- Recall
    - class 0 : model predicts non churn correct at 86% of the time
    - class 1 : model predicts churn correct at 75% of the time
    - High recall for churn at the expense of precision
- F1-Score
    - weighted average of precision and recall, taking into account both classes
    - 52% for churn
- THe logistic regression model performs poorly
    - The performance is due to class imbalance of the test sample, needs further test models using the Decision Tree classifier. To further enhance the model performance, ensemble methods like e.g XGBoost and Random Forest which can handle the class imbalance better will be explored

6.3 6.2 Decision Tree

6.3.1 Initializing Decision Tree classifier

In [254]:
```
decision_tree = DecisionTreeClassifier(
    criterion = 'gini', # gini impurity criterion
    max_depth=5,  # maximum depth of the tree
    min_samples_split=10,  # minimum number of samples required to split an internal node. It is set at 10 
    random_state=42  # random state for reproducibility
)
```

6.3.2 Fitting Decision Tree

In [255]:
```
decision_tree.fit(X_train,y_train)
```

Out[255]:
```
DecisionTreeClassifier(max_depth=5, min_samples_split=10, random_state=42)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

6.3.3 Decision tree prediction

In [256]:
```
y_pred_dt = decision_tree.predict(X_test)
y_proba_dt = decision_tree.predict_proba(X_test)[:,1]
```

6.3.4 Decision tree model Evaluation

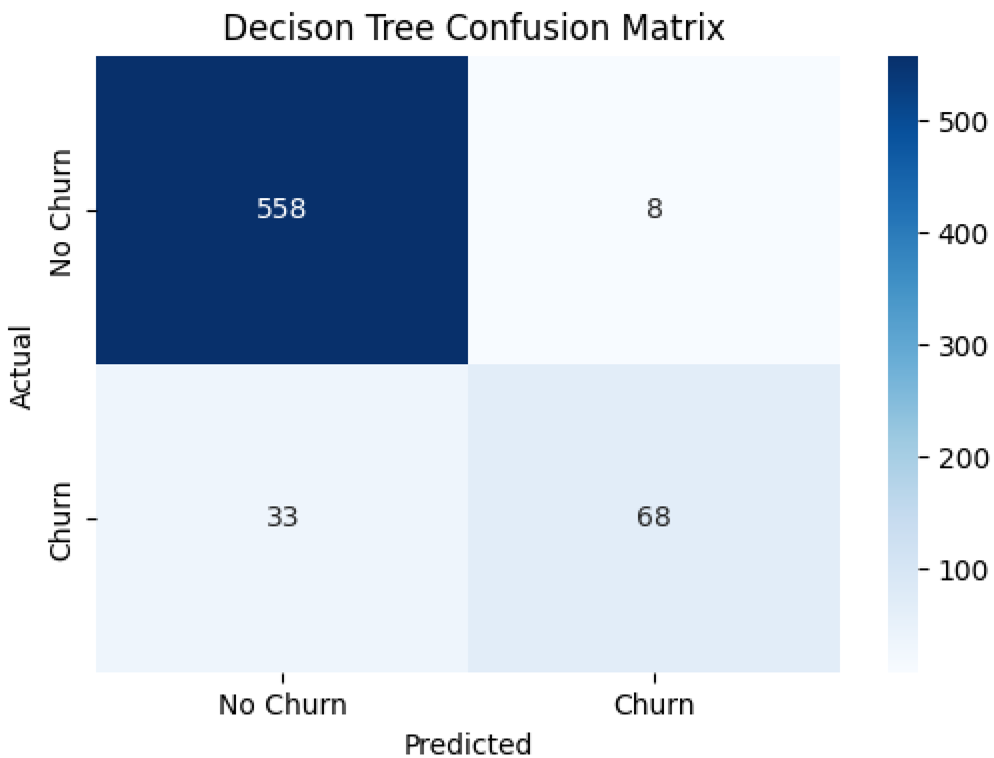
In [257]:
```
Accuracy_dt= accuracy_score(y_test, y_pred_dt)
confusion_dt = confusion_matrix(y_test, y_pred_dt)
Report_dt = classification_report(y_test, y_pred_dt)
roc_auc_dt = roc_auc_score(y_test, y_pred_dt)


print("Decision Tree Results")
print('-' * 55)
print("Accuracy:", Accuracy_dt)
print('-' * 55)
print("Confusion Matrix:\n", confusion_dt)
print('-' * 55)
print("Classification Report:\n", Report_dt)
print('-' * 55)
print("ROC AUC Score:", roc_auc_dt)
print('-' * 55)
```

```
Decision Tree Results
-------------------------------------------------------
Accuracy: 0.9385307346326837
-------------------------------------------------------
Confusion Matrix:
 [[558   8]
 [ 33  68]]
-------------------------------------------------------
Classification Report:
               precision    recall  f1-score   support

           0       0.94      0.99      0.96       566
           1       0.89      0.67      0.77       101

    accuracy                           0.94       667
   macro avg       0.92      0.83      0.87       667
weighted avg       0.94      0.94      0.93       667

-------------------------------------------------------
ROC AUC Score: 0.8295665255571495
-------------------------------------------------------
```

6.4 Visualization of the Decision Tree Evaluations

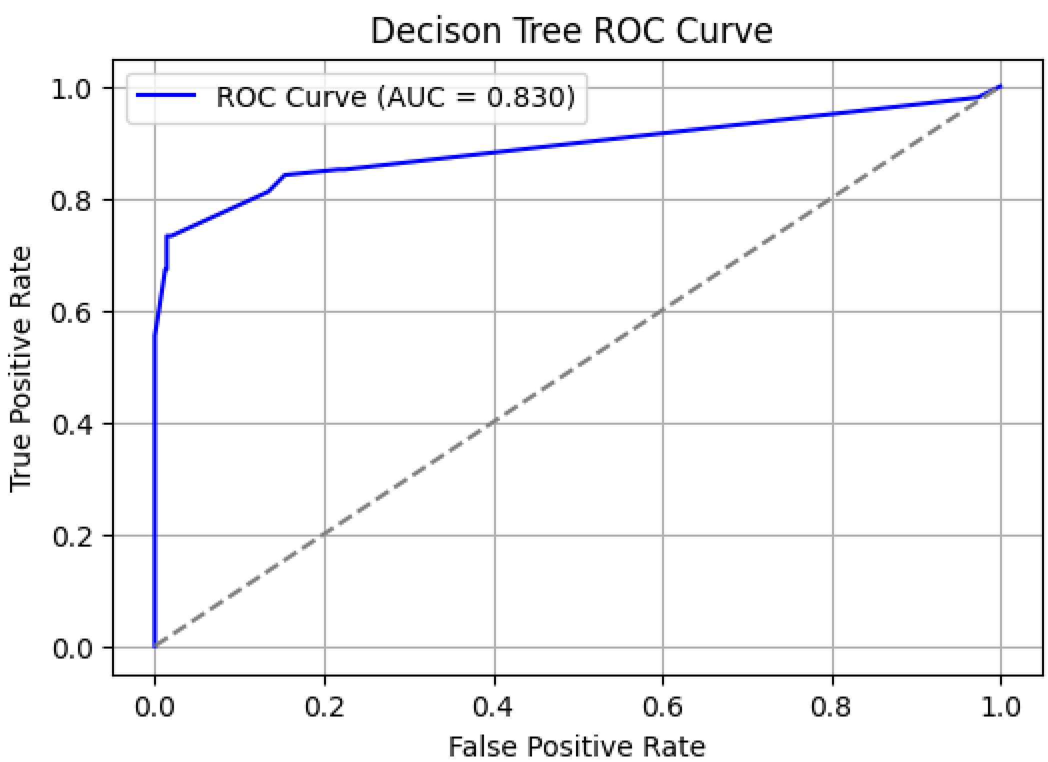
6.4.1 Decision Tree confusion Matrix plot

In [258]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_dt, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title(" Decison Tree Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



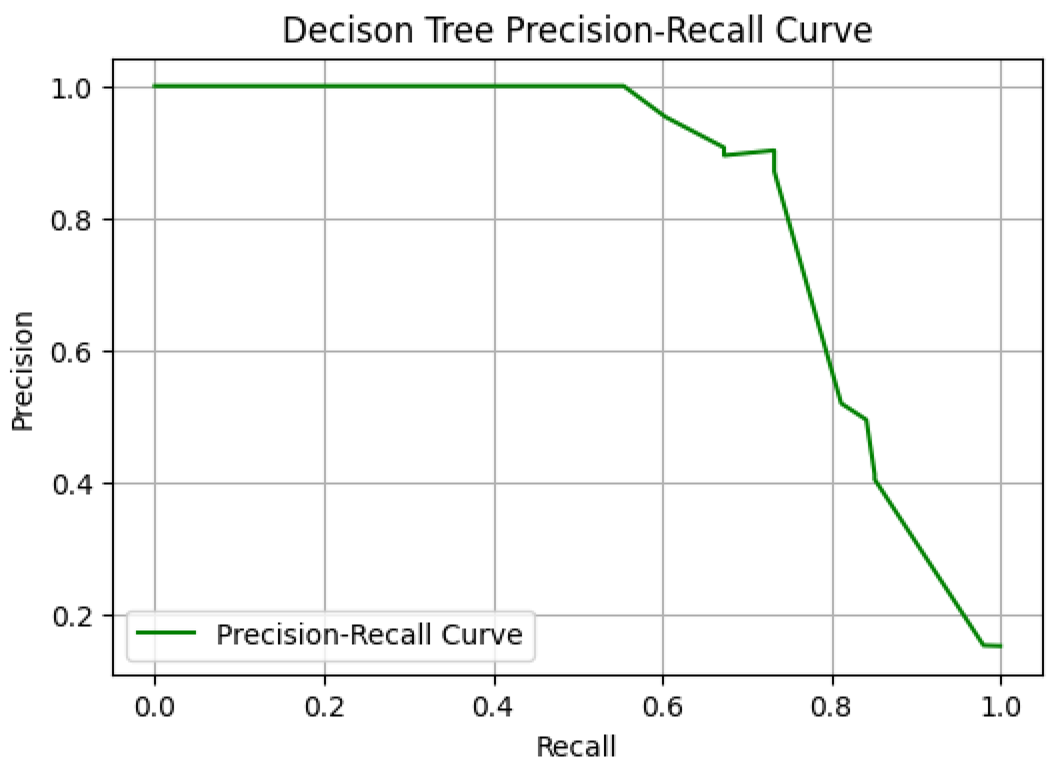
6.4.2 Decision Tree ROC curve

In [259]:
```
fpr, tpr, _ = roc_curve(y_test, y_proba_dt)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC = {:.3f})'.format(roc_auc_dt))
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Decison Tree ROC Curve ')
plt.grid(True)
plt.legend()
plt.show()
```



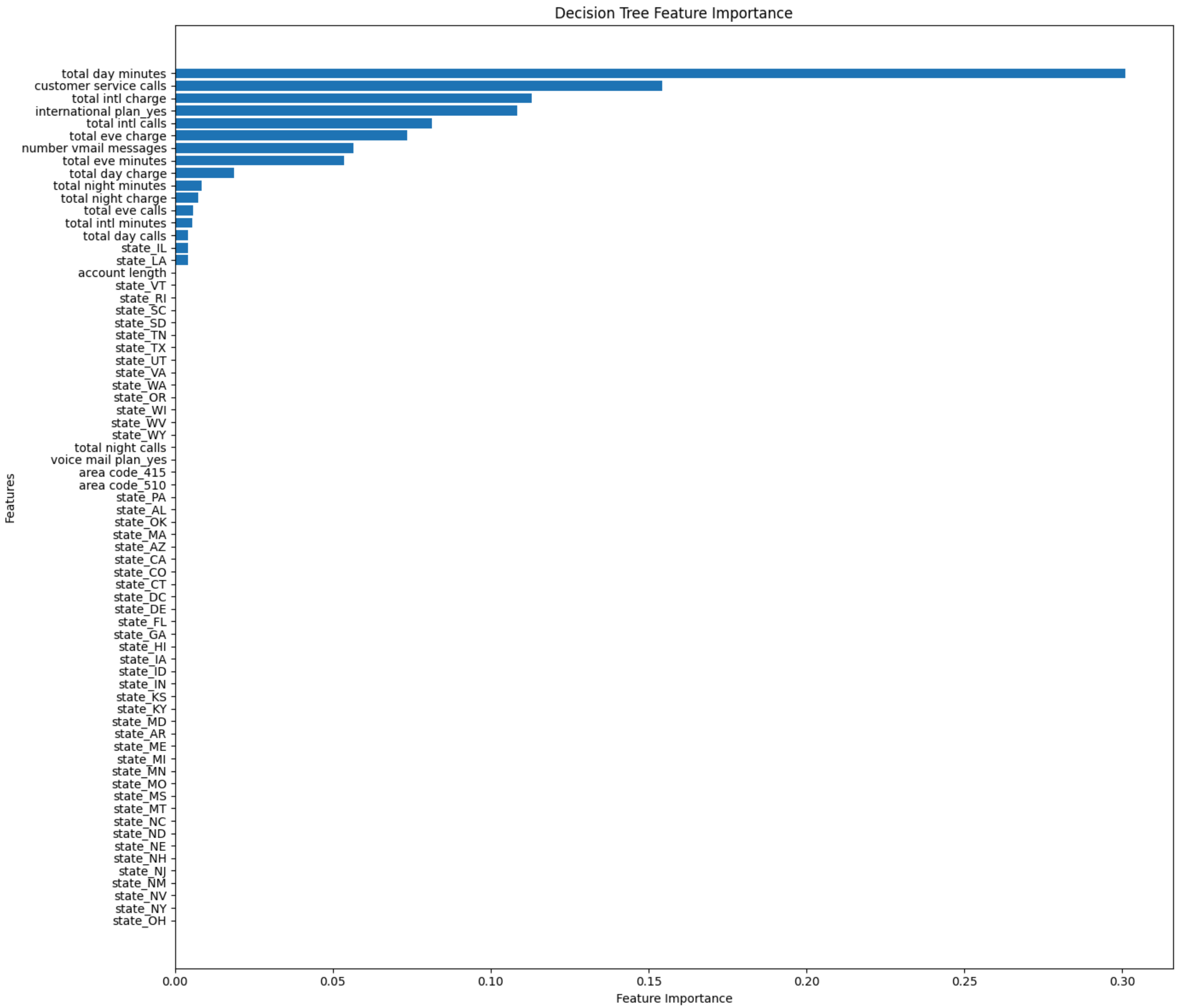
6.4.3 Decision Tree Precision Recall Plot

In [260]:
```
precision, recall, _ = precision_recall_curve(y_test, y_proba_dt)
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='green', label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Decison Tree Precision-Recall Curve')
plt.grid(True)
plt.legend()
plt.show()
```



6.4.4 Visualization of Decision Tree Feature importance

In [261]:
```
# Get feature importance scores
feature_importances = decision_tree.feature_importances_

# Create a DataFrame for feature importance
importance_df = pd.DataFrame({
    'Feature': X_processed.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(14, 12))
plt.barh(importance_df['Feature'], importance_df['Importance'], align='center')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Decision Tree Feature Importance")
plt.gca().invert_yaxis()  # Reverse the order to show the most important feature on top
plt.tight_layout()
plt.show()
```


Decision Tree Feature Importance
total day minutes
customer service calls
total intl charge
international plan_yes
total intl calls
total eve charge
number vmail messages
total eve minutes
total day charge
total night minutes
total night charge
total eve calls
total intl minutes
total day calls
state_IL
state_LA
account length
state_VT
state_RI
state_SC
state_SD
state_TN
state_TX
state_UT
state_VA
state_WA
state_OR
state_WI
state_WV
state_WY
total night calls
voice mail plan_yes
area code_415
area code_510
state_PA
state_AL
state_OK
state_MA
state_AZ
state_CA
state_CO
state_CT
state_DC
state_DE
state_FL
state_GA
state_HI
state_IA
state_ID
state_IN
state_KS
state_KY
state_MD
state_AR
state_ME
state_MI
state_MN
state_MO
state_MS
state_MT
state_NC
state_ND
state_NE
state_NH
state_NJ
state_NM
state_NV
state_NY
state_OH
Features
0.00
0.05
0.10
0.15
0.20
0.25
0.30
Feature Importance

6.5 Interpretation of Decison Tree

- The Accuracy score was 0.94 meaning that the model classified correctly 94% of the test sample
- The ROC AUC score was 0.83
- Confusion Matrix
  - 558 (True Negatives) - customers who did not churn and were correctly predicted not to churn
  - 8 (False Positives) - customers who did not churn but were incorrectly predicted to churn
  - 33 (False Negatives) - customers who churned but were incorrectly predicted not to churn
  - 68 (True Positives) - customers who churned and were correctly predicted to churn
- The model predicts churn(false positive) better than and churn cases (false negative)
- Precision
  - class 0 : model predicts non churn correct at 94% of the time
  - class 1 : model predicts churn correct at 89% of the time
  - low precison for churn show model has a low false positive
- Recall
  - class 0 : model predicts non churn correct at 99% of the time
  - class 1 : model predicts churn correct at 66% % of the time
  - High recall for churn at the expense of precision
- F1-Score
  - weighted average of precision and recall, taking into account both classes
  - 77% for churn
  - 96% for non churn
- The following have been identified to have high Feature importance by Decision Tree
  - Total day minutes
  - customer service calls
  - international call plan yes
  - Total international calls
  - total evening charges
  - Total numver of voice mail messages
  - total evening minutes

6.5.1 The Comparison of the Decision Tree Model and the logistic regression

- Decision tree model performs better than the Logistic regression in Accuracy 0.94 vs the accuracy score of Logistic regression of 0.79.
- Both have similar ROC AUC score of 0.83
- Decision tree model performs better than the Logistic regression in precision of churn 0.89 vs the precision score of Logistic regression of 0.40
- Decision tree model performs better than the Logistic regression in in F1 score

7 Ensemble Methods

7.1 6.3. XGBoost Classifier

7.1.1 Tuning the XGBoot Classifier

In [262]:
```
xgboot = XGBClassifier(use_label_encoder= False, eval_metric ='logloss',random_state = 42 )
# eval_metric is a performance metric for ebvaluation during training
# random_state ensures reproducibility
```

7.1.2 Parameters Tuning for XGBoost model

In [263]:
```
param_grid_xgb = {
    'n_estimators': [100, 200, 300], # number of boosting rounds(trees)
    'max_depth': [3, 5, 7],           # The maximun  depth of each tree
    'learning_rate': [0.01, 0.1, 0.2], # Step-size shrinkage to prevent overfitting
    'subsample': [0.8, 1.0],          #random sample for growing tree
    'colsample_bytree': [0.8, 1.0]    # Fraction of features to randomly sample for each tree
}
```

7.1.3 Grid Search cross validation for XGBoost

In [264]:
```
grid_search_xgb = GridSearchCV(xgboot, param_grid_xgb, cv=5, scoring='f1_macro', n_jobs=-1, verbose=2)
grid_search_xgb.fit(X_train_bal, y_train_bal) # training XGBoost model on SMOTE balanced training data
```

```
Fitting 5 folds for each of 108 candidates, totalling 540 fits
```

Out[264]:
```
GridSearchCV(cv=5,
             estimator=XGBClassifier(base_score=None, booster=None,
                                     callbacks=None, colsample_bylevel=None,
                                     colsample_bynode=None,
                                     colsample_bytree=None, device=None,
                                     early_stopping_rounds=None,
                                     enable_categorical=False,
                                     eval_metric='logloss', feature_types=None,
                                     gamma=None, grow_policy=None,
                                     importance_type=None,
                                     interaction_constraints=None,
                                     learning_rate=...
                                     max_leaves=None, min_child_weight=None,
                                     missing=nan, monotone_constraints=None,
                                     multi_strategy=None, n_estimators=None,
                                     n_jobs=None, num_parallel_tree=None,
                                     random_state=42, ...),
             n_jobs=-1,
             param_grid={'colsample_bytree': [0.8, 1.0],
                         'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 5, 7],
                         'n_estimators': [100, 200, 300],
                         'subsample': [0.8, 1.0]},
             scoring='f1_macro', verbose=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

7.1.4 XGBoost Model prediction

In [265]:
```
best_xgb = grid_search_xgb.best_estimator_
y_pred_xgb = best_xgb.predict(X_test)
```

7.1.5 XGBoost Model Evaluation

In [266]:

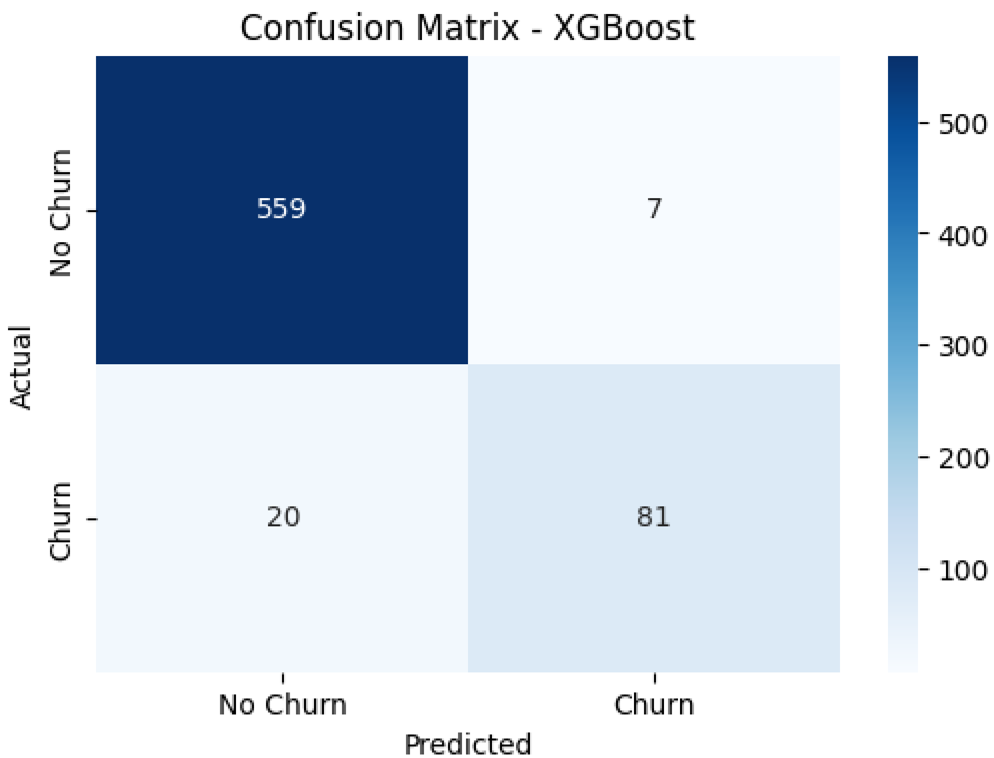
```
Accuracy_xgb= accuracy_score(y_test, y_pred_xgb)
confusion_xgb = confusion_matrix(y_test, y_pred_xgb)
Report_xgb = classification_report(y_test, y_pred_xgb)
roc_auc_xgb = roc_auc_score(y_test, y_pred_xgb)
y_proba_xgb = best_xgb.predict_proba(X_test)[:, 1]


print("XGBoost Results")
print('-' * 55)
print("Accuracy:", Accuracy_xgb)
print('-' * 55)
print("Confusion Matrix:\n", confusion_xgb)
print('-' * 55)
print("Classification Report:\n", Report_xgb)
print('-' * 55)
print("ROC AUC Score:", roc_auc_xgb)
print('-' * 55)
```

```
XGBoost Results
-------------------------------------------------------
Accuracy: 0.95952023988006
-------------------------------------------------------
Confusion Matrix:
 [[559   7]
 [ 20  81]]
-------------------------------------------------------
Classification Report:
               precision    recall  f1-score   support

           0       0.97      0.99      0.98       566
           1       0.92      0.80      0.86       101

    accuracy                           0.96       667
   macro avg       0.94      0.89      0.92       667
weighted avg       0.96      0.96      0.96       667

-------------------------------------------------------
ROC AUC Score: 0.8948063534268622
-------------------------------------------------------
```

7.2 XGBoost model Evaluation Visualization

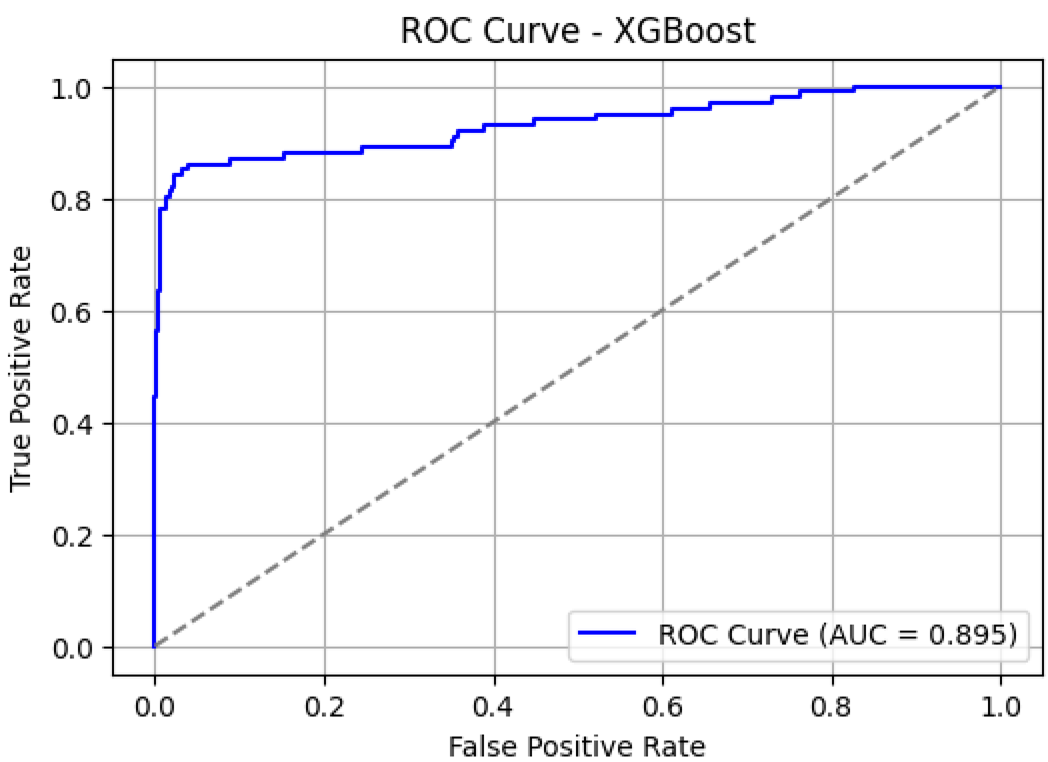
7.2.1 XGBoost Confusion matrix

In [267]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_xgb, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title("Confusion Matrix - XGBoost")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



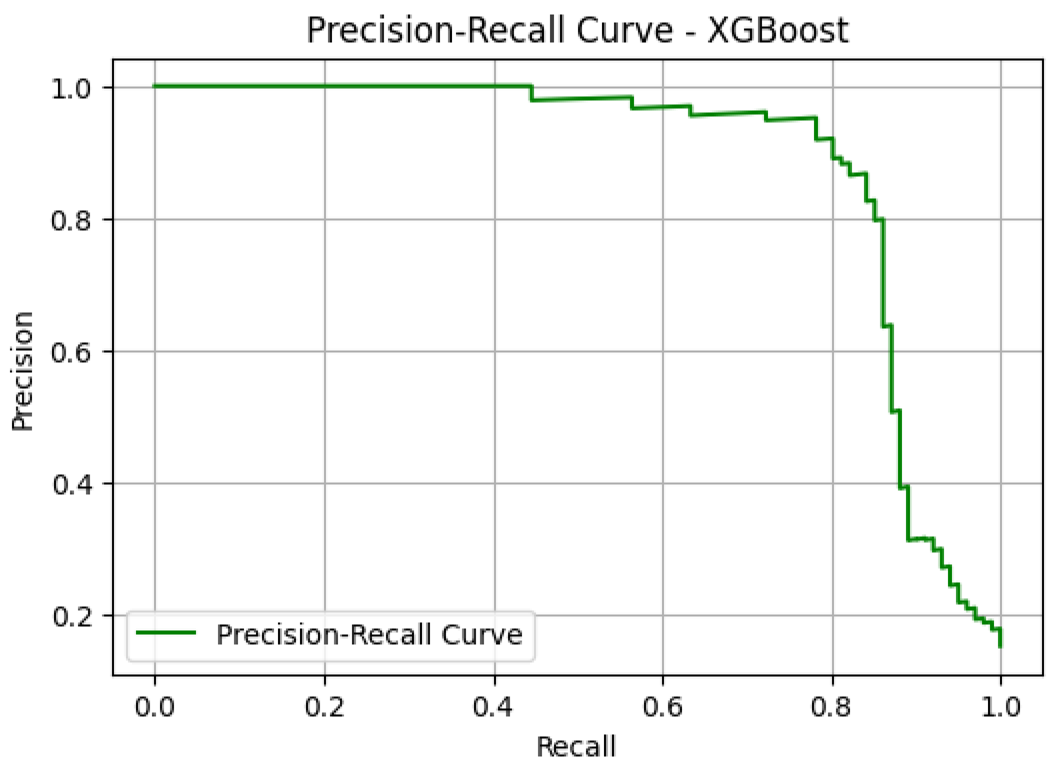
7.2.2 XGBoost ROC curve plot

In [268]:
```
fpr, tpr, _ = roc_curve(y_test, y_proba_xgb)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC = {:.3f})'.format(roc_auc_xgb))
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - XGBoost')
plt.grid(True)
plt.legend()
plt.show()
```



7.2.3 XGBoost Precision Recall curve plot

In [269]:
```
precision, recall, _ = precision_recall_curve(y_test, y_proba_xgb)
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='green', label='Precision-Recall Curve')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve - XGBoost')
plt.grid(True)
plt.legend()
plt.show()
```

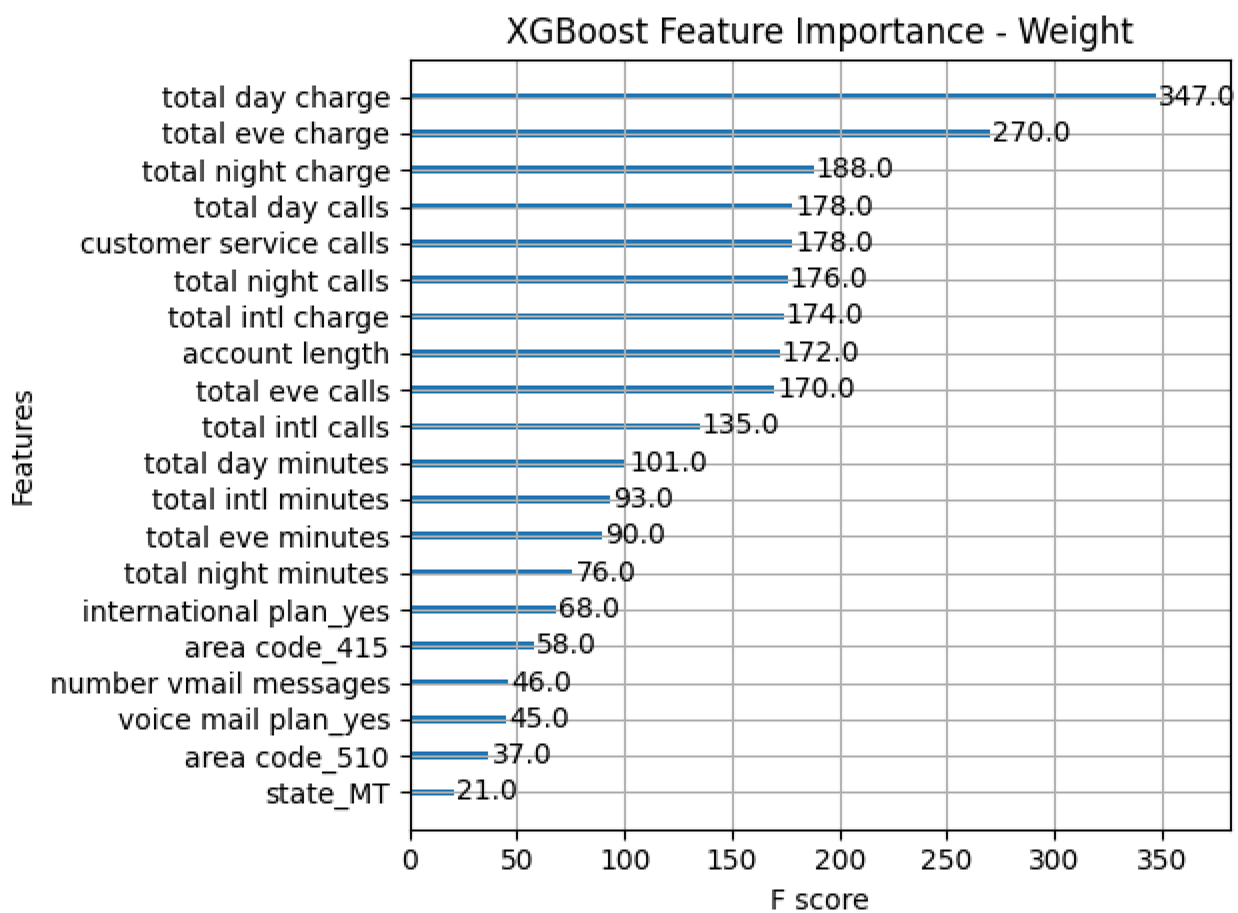


7.2.4 XGBoost Feature importance

In [270]:

```
from xgboost import plot_importance
plt.figure(figsize=(10, 8))
plot_importance(best_xgb, importance_type='weight', max_num_features= 20)
plt.title("XGBoost Feature Importance - Weight")
plt.tight_layout()
plt.show()
```

```
<Figure size 1000x800 with 0 Axes>
```



7.3 Interpretation of the Evaluation of the XGBoost

- The Accuracy score was 0.96 meaning that the model classified correctly 96% of the test sample

- Confusion Matrix
  - 558 (True Negatives) - customers who did not churn and were correctly predicted not to churn
  - 8 (False Positives) - customers who did not churn but were incorrectly predicted to churn
  - 20 (False Negatives) - customers who churned but were incorrectly predicted not to churn
  - 81 (True Positives) - customers who churned and were correctly predicted to churn
- Precision
  - class 0 : model predicts non churn correct at 97% of the time
  - class 1 : model predicts churn correct at 91% of the time
- Recall
  - class 0 : model predicts non churn correct at 99% of the time
  - class 1 : model predicts churn correct at 80% of the time
- F1-Score
  - weighted average of precision and recall, taking into account both classes
  - 85% for churn
- The following were weighted highest in feature importance
  - Total day charge
  - Total evening charge
  - Customer service calls
  - Total international calls
  - Total night charge s
  - Total day calls

7.4 6.4 Random Forest Model

7.4.1 Defining Random forest model for churn

In [271]:
```
rf = RandomForestClassifier(random_state=42, class_weight='balanced')
```

7.4.2 Hyper parameter Tuning for Random Forest Model

In [272]:
```
param_grid_rf = {
    'n_estimators': [100, 200, 300],  # Number of trees in the forest
    'max_depth': [10, 20, None],      # Maximum depth of each tree
    'min_samples_split': [2, 5],      # Minimum number of samples to split a node
    'min_samples_leaf': [1, 2]
     }     # Minimum number of samples for a leaf
```

7.4.3 Grid Search cross validation for Random forest model

In [273]:
```
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1_macro', n_jobs=-1, verbose=2)
grid_search_rf.fit(X_train_bal, y_train_bal) # training Random forest model on SMOTE balanced data set
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
```

Out[273]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(class_weight='balanced',
                                              random_state=42),
             n_jobs=-1,
             param_grid={'max_depth': [10, 20, None],
                         'min_samples_leaf': [1, 2],
                         'min_samples_split': [2, 5],
                         'n_estimators': [100, 200, 300]},
             scoring='f1_macro', verbose=2)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

7.4.4 Random Forest model prediction

In [274]:
```
best_rf_model = grid_search_rf.best_estimator_
```

7.4.5 Random Forest model Prediction on Test data

In [275]:
```
y_pred_rf = best_rf_model.predict(X_test)
```

7.5 Random Forest model Evaluation

In [276]:

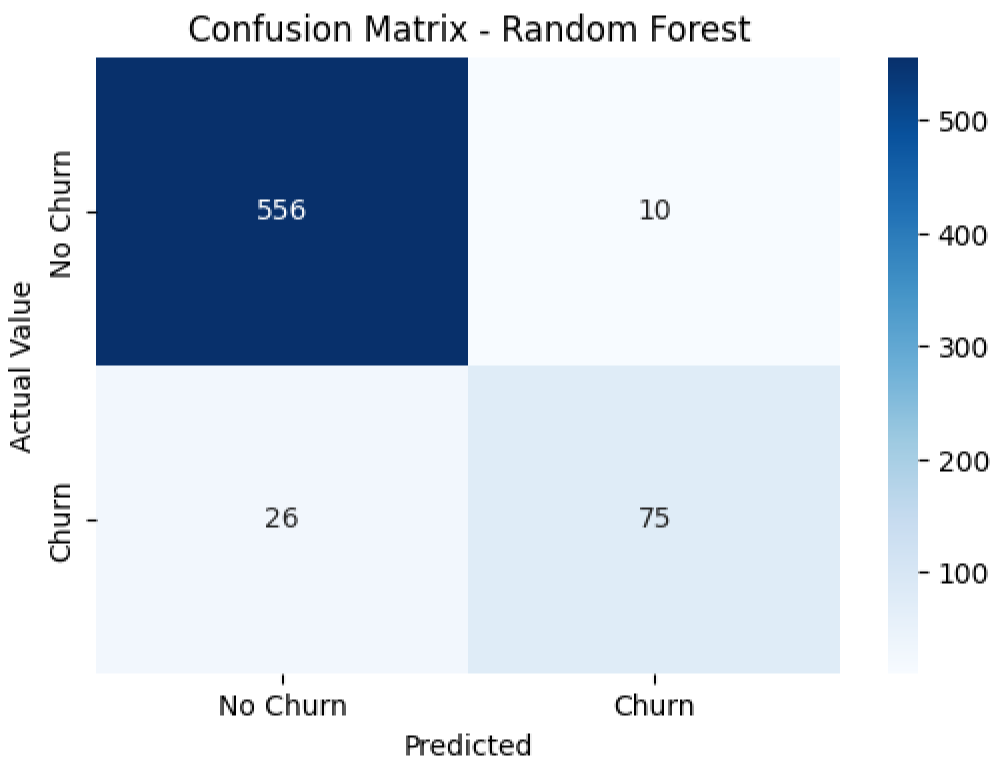
```
# Evaluation
accuracy_rf = accuracy_score(y_test, y_pred_rf)
confusion_rf = confusion_matrix(y_test, y_pred_rf)
classification_report_rf = classification_report(y_test, y_pred_rf)
roc_auc_rf = roc_auc_score(y_test, y_pred_rf)
y_proba_rf = best_rf_model.predict_proba(X_test)[:, 1]

# results
print ("Random forest results")
print('-' * 55)
print('Best Parameters:',grid_search_rf.best_params_)
print("Accuracy:", accuracy_rf)
print('-' * 55)
print("Confusion Matrix:\n", confusion_rf)
print('-' * 55)
print("Classification Report:\n", classification_report_rf)
print('-' * 55)
print("ROC AUC Score:", roc_auc_rf)
```

```
Random forest results
-------------------------------------------------------
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Accuracy: 0.9460269865067467
-------------------------------------------------------
Confusion Matrix:
 [[556  10]
 [ 26  75]]
-------------------------------------------------------
Classification Report:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97       566
           1       0.88      0.74      0.81       101

    accuracy                           0.95       667
   macro avg       0.92      0.86      0.89       667
weighted avg       0.94      0.95      0.94       667

-------------------------------------------------------
ROC AUC Score: 0.8624532064513871
```

7.6 Random Forest model Evaluation Visualization

7.6.1 Random Forest model confusion matrix plot

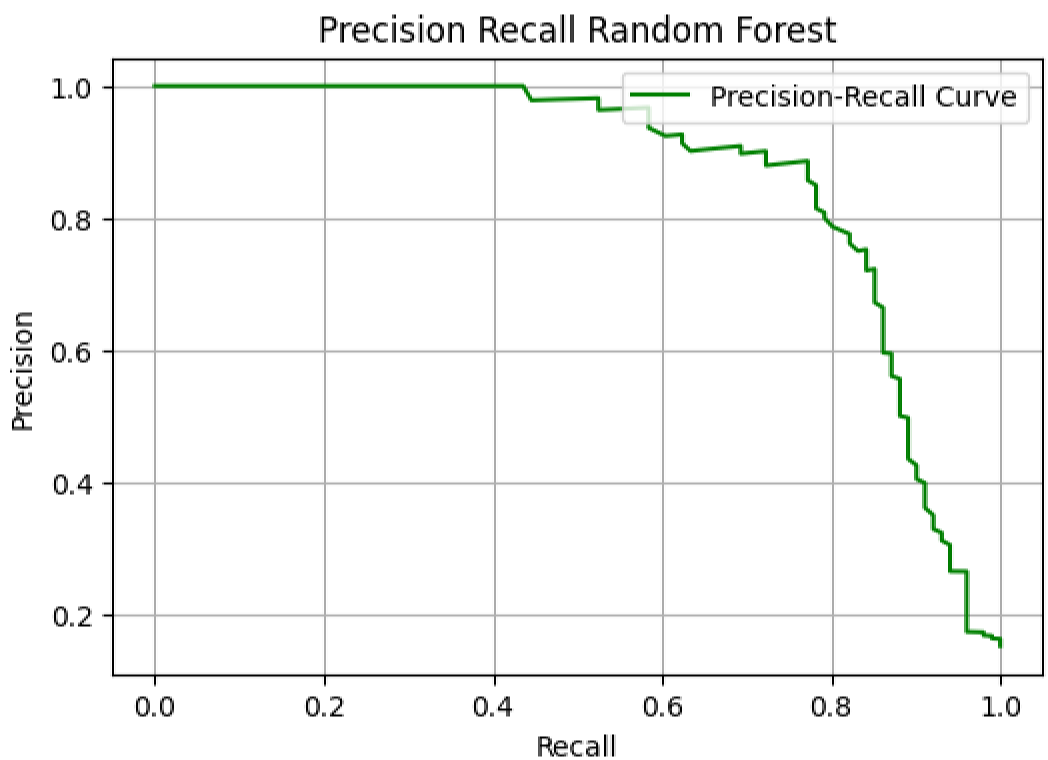
In [277]:
```
plt.figure(figsize=(6, 4))
sns.heatmap(confusion_rf, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Churn', 'Churn'], yticklabels=['No Churn', 'Churn'])
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual Value')
plt.show()
```



7.6.2 Random Forest model Precision recall curve plot

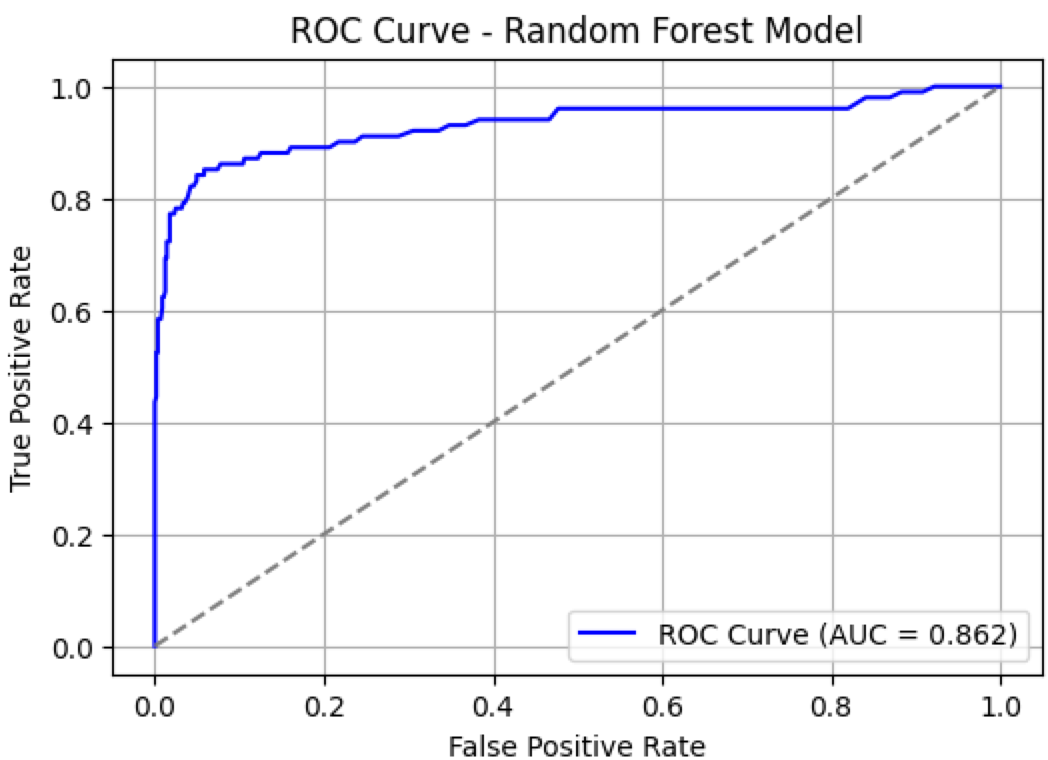
In [278]:

```
precision, recall, _ = precision_recall_curve(y_test, y_proba_rf)
plt.figure(figsize=(6, 4))
plt.plot(recall, precision, color='green', label='Precision-Recall Curve')
plt.title('Precision Recall Random Forest')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```



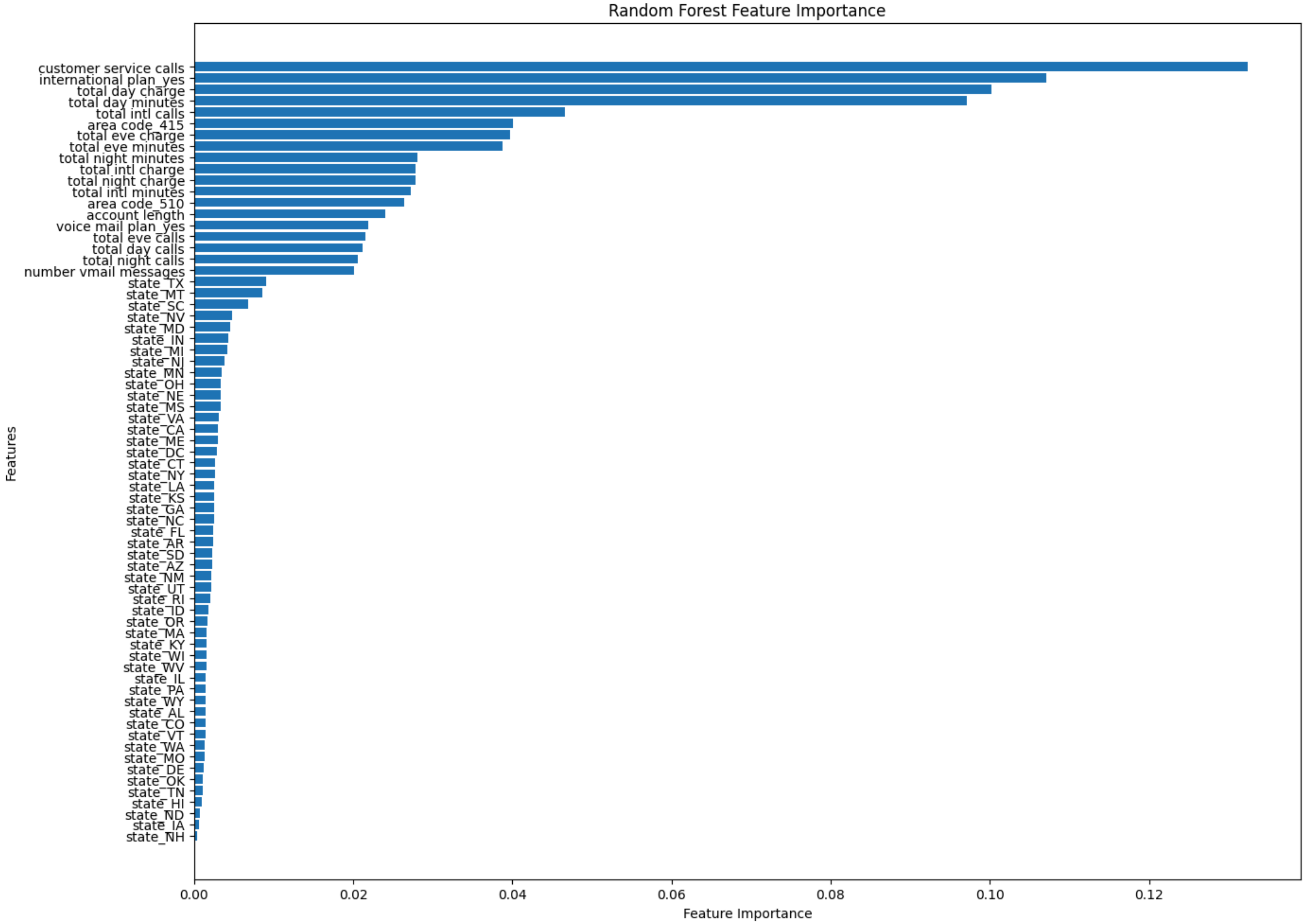
7.6.3 Random Forest ROC curve plot

In [279]:
```
fpr, tpr, _ = roc_curve(y_test, y_proba_rf)
plt.figure(figsize=(6, 4))
plt.plot(fpr, tpr, color='blue', label='ROC Curve (AUC = {:.3f})'.format(roc_auc_rf))
plt.plot([0, 1], [0, 1], linestyle='--', color='grey')
plt.title ('ROC Curve - Random Forest Model')
plt.grid(True)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()
```



7.6.4 Random Forest model Feature importance plot

In [280]:
```
# Get feature importance scores
feature_importances = best_rf_model.feature_importances_

# Create a DataFrame for feature importance
importance_df = pd.DataFrame({
    'Feature': X_processed.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(14, 10))
plt.barh(importance_df['Feature'], importance_df['Importance'], align='center')
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Random Forest Feature Importance")
plt.gca().invert_yaxis()  # Reverse the order to show the most important feature on top
plt.tight_layout()
plt.show()
```



7.6.5 Interpretation of Random Forest Evaluation

The Accuracy score was 0.95 meaning that the model classified correctly 96% of the test sample

- Confusion Matrix

- 556 (True Negatives) - customers who did not churn and were correctly predicted not to churn
- 10 (False Positives) - customers who did not churn but were incorrectly predicted to churn
- 26 (False Negatives) - customers who churned but were incorrectly predicted not to churn
- 75 (True Positives) - customers who churned and were correctly predicted to churn

- Precision
  - class 0 : model predicts non churn correct at 96% of the time
  - class 1 : model predicts churn correct at 88% of the time
- Recall
  - class 0 : model predicts non churn correct at 98% of the time
  - class 1 : model predicts churn correct at 74% of the time
- F1-Score
  - weighted average of precision and recall, taking into account both classes
  - 81% for churn
- The following were weighted highest in feature importance
  - Total day charge
  - Total evening charge
  - Customer service calls
  - Total international calls
  - Total night charge s
  - Total day calls

7.6.6 XGBoot performed better than Random forest Ensember in all parameters

8 7. Conclusion and recommendation

- XGBoost model had the best overall performance, hoever it will need further tuning
- The organization can also consider Decision tree which performed better than Logistic regression. However, The model will need with further tuning to address recall.
- Further analysis of customer feedback will be important to understand reasons so that mitigation measures can be implemented
- Features which were weighted highest in importance need to be analysed further to refine model. This include phone charges, international rates
- The model will help identify customeers who are likey to churn so that the reasons for attriction can be addressed to ensure retention of clientelle
- The limitation of the analysis was use of historical data with significant class imbalance , the model can be further defined as progressively