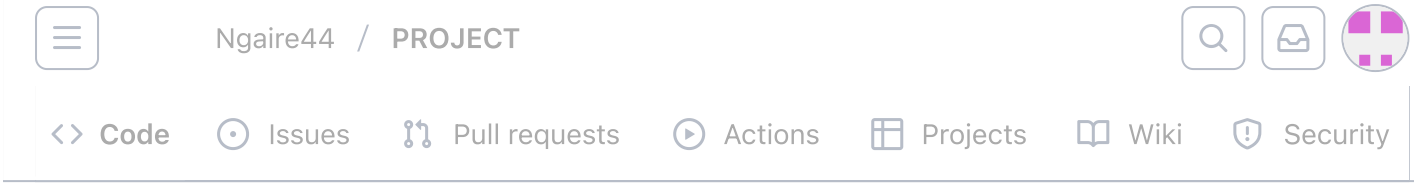


PROJECT / notebook.ipynb

Ngaire44 final project 180d5b7 · 22 minutes ago

5575 lines (5575 loc) · 887 KB

SyriaTel Customer Churn Project

Business Understanding

SyriaTel, a telecommunications company, experiences customer churn at around 15%, meaning thousands of customers leave every year. This results in lost revenue, increased acquisition costs, and weakened customer loyalty. While churn prediction is a common use case, we aim to go beyond prediction by identifying the key behavioral and service-related factors driving churn. This will help SyriaTel target interventions effectively and reduce customer attrition.

This project aims:

- To build a predictive model that identifies customers at high risk of churn
- To identify key factors likely to make a customer churn; service quality, call charges, customer support
- To recommend actions that will reduce churn
- 5% reduction in churn rate through targeted retention strategies

Incase this dataset is imbalanced, accuracy alone is not a good measure. Instead, we will evaluate success using:

- Recall >= 85 (Sensitivity) for churn class - ensure the model identifies as many churners as possible, since missing them means lost revenue.
- Accuracy >=85 so predictions are overall correct
- F1-score - balance between Precision and Recall.
- AUC >= 0.85 -overall ability of the model to discriminate churners vs. non-churners.

Data Understanding

Columns description

- state: The U.S. state where the customer resides
- account length: Number of days the customer has had an account with SyriaTel
- area code: The customer’s telephone area code
- phone number: The unique phone number assigned to the customer.
- international plan: A binary indicator (yes/no) of whether the customer has subscribed to an international calling plan.
- voice mail plan: A binary indicator (yes/no) of whether the customer has subscribed to a voicemail plan.
- number vmail messages: The number of voicemail messages recorded by the customer.
- total day minutes: The total number of minutes the customer used during

daytime calls
- total day calls: The total number of calls made during the day.
- total day charge: The total charges incurred from daytime calls, directly proportional to total day minutes.
- total eve minutes: The total number of minutes the customer used during evening calls
- total eve calls: The total number of calls made during the evening.
- total eve charge: The total charges incurred from evening calls, directly proportional to total eve minutes.
- total night minutes: The total number of minutes the customer used during nighttime calls
- total night calls: The total number of calls made during the night.
- total night charge: The total charges incurred from nighttime calls, directly proportional to total night minutes.
- total intl minutes: The total number of minutes spent on international calls.
- total intl calls: The total number of international calls made by the customer.
- total intl charge: The total charges incurred from international calls, directly proportional to total intl minutes.
- customer service calls: The number of calls the customer made to customer service.
- churn: indicates whether the customer has churned (yes/no); Target variable

In [1]:
```
# import libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

from scipy.stats import chi2_contingency,ttest_ind
from xgboost import XGBClassifier

from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score, roc_auc_score, roc_curve,\
    classification_report, confusion_matrix, accuracy_score,\
          recall_score,precision_score,auc,make_scorer,average_precisi
from sklearn.model_selection import train_test_split,GridSearchCV,\
     cross_validate,StratifiedKFold,StratifiedShuffleSplit
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.calibration import CalibratedClassifierCV
from sklearn.inspection import permutation_importance
from imblearn.over_sampling import SMOTE
```

In [2]:
```
#load datasets
df = pd.read_csv('bigml.csv')
df.head()
```

Out[2]:

state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls

0	KS	128	415	382-4657	no	yes	25	265.1	110
1	OH	107	415	371-7191	no	yes	26	161.6	123
2	NJ	137	415	358-1921	no	no	0	243.4	114
3	OH	84	408	375-9999	yes	no	0	299.4	71
4	OK	75	415	330-6626	yes	no	0	166.7	113

5 rows × 21 columns

In [3]:
```
df.tail()
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	t
3328	AZ	192	415	414-4276	no	yes	36	156.2	
3329	WV	68	415	370-3271	no	no	0	231.1	
3330	RI	28	510	328-8230	no	no	0	180.8	
3331	CT	184	510	364-6381	yes	no	0	213.8	
3332	TN	74	415	400-4344	no	yes	25	234.4	

5 rows × 21 columns

observations: the columns are uniform

In [4]:
```
#shape of the dataset
print(f"The dataset has {df.shape[0]} rows and {df.shape[1]} columns")
```

```
The dataset has 3333 rows and 21 columns
```

In [5]:
```
# investigate the 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
```

observations:

- We will one hot encode the intl plan, voicemail plan and churn
- The dataset has no missing values

In [6]:
```
#stat summary
df.describe()
```

Out[6]:

	account length	area code	number vmail messages	total day minutes	total day calls	
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	333
mean	101.064806	437.182418	8.099010	179.775098	100.435644	3
std	39.822106	42.371290	13.688365	54.467389	20.069084	
min	1.000000	408.000000	0.000000	0.000000	0.000000	
25%	74.000000	408.000000	0.000000	143.700000	87.000000	2
50%	101.000000	415.000000	0.000000	179.400000	101.000000	3
75%	127.000000	510.000000	20.000000	216.400000	114.000000	3
max	243.000000	510.000000	51.000000	350.800000	165.000000	5

In [7]:
```
#unique values
for col in df:
    uniq_values = df[col].unique()
    print(f"{col}\n {df[col].unique()}\n")
```

```
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  9
3
  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]

phone number
 ['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']

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 3
8
 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  9
0
 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  6
5
  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 13
3
```

```
 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   8.8
2
  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
```



```
  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]
```

Data Preparation

In [8]:
```
#make a copy
data = df.copy(deep=True)
```

In [9]:
```
#remove spaces in column names
data.columns = data.columns.str.replace(" ","_")
data.columns
```

Out[9]:
```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_message
s',
       '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_charg
e',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

In [10]:
```
#change churn data type to object
data.churn = data['churn'].astype('O')
```

In [11]:
```
#drop phone numbers
del data['phone_number']
data
```

Out[11]:

	state	account_length	area_code	international_plan	voice_mail_plan	nu
0	KS	128	415	no	yes	
1	OH	107	415	no	yes	
2	NJ	137	415	no	no	
3	OH	84	408	yes	no	
4	OK	75	415	yes	no	
...	...	...	...	...	...	
3328	AZ	192	415	no	yes	
3329	WV	68	415	no	no	
3330	RI	28	510	no	no	
3331	CT	184	510	yes	no	
3332	TN	74	415	no	yes	

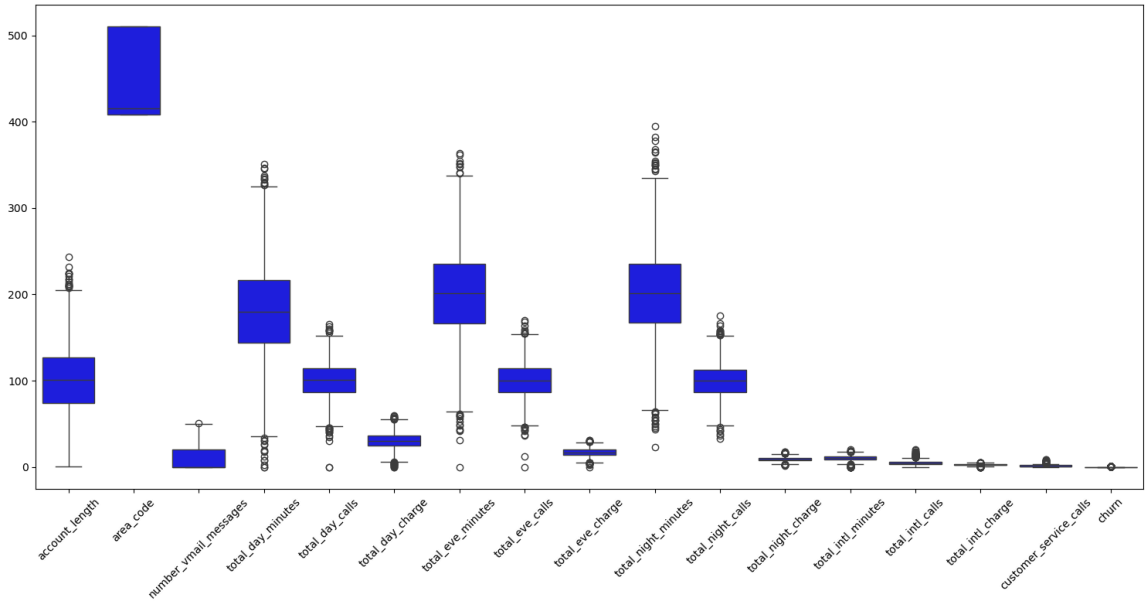
3333 rows × 20 columns

In [12]:
```
#check for duplicates,missing values
print(f"The dataset has {data.duplicated().sum()} duplicated values")
print(f"The dataset has: \n{data.isna().sum()} missing values")
```

```
The dataset has 0 duplicated values
The dataset has:
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 missing values
```

In [13]:
```
#check for outliers
plt.figure(figsize=(15,8))
sns.boxplot(data,color='blue')
plt.xticks(rotation=45)
plt.tight_layout()
```



Observations: The outliers are genuine

- Call times may vary depending on clients' job,location or lifestyle
- Call charges are dependent on time spend on call
- Customer service calls may increase as a result of complains from clients

In [14]:
```
#confirm data types
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account_length          3333 non-null   int64
```

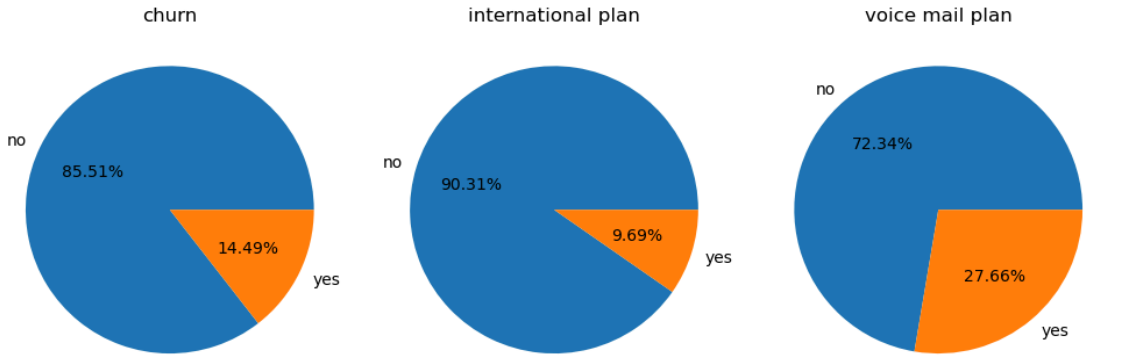
```
 2   area_code               3333 non-null   int64
 3   international_plan      3333 non-null   object
 4   voice_mail_plan         3333 non-null   object
 5   number_vmail_messages   3333 non-null   int64
 6   total_day_minutes       3333 non-null   float64
 7   total_day_calls         3333 non-null   int64
 8   total_day_charge        3333 non-null   float64
 9   total_eve_minutes       3333 non-null   float64
 10  total_eve_calls         3333 non-null   int64
 11  total_eve_charge        3333 non-null   float64
 12  total_night_minutes     3333 non-null   float64
 13  total_night_calls       3333 non-null   int64
 14  total_night_charge      3333 non-null   float64
 15  total_intl_minutes      3333 non-null   float64
 16  total_intl_calls        3333 non-null   int64
 17  total_intl_charge       3333 non-null   float64
 18  customer_service_calls  3333 non-null   int64
 19  churn                   3333 non-null   object
dtypes: float64(8), int64(8), object(4)
memory usage: 520.9+ KB
```

In [15]:
```
#save the cleaned dataset
data.to_csv('clean_customer_churn.csv',index=False)
```

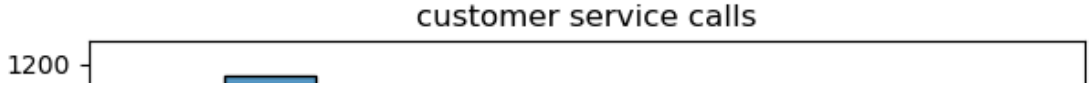
Exploratory Data Analysis

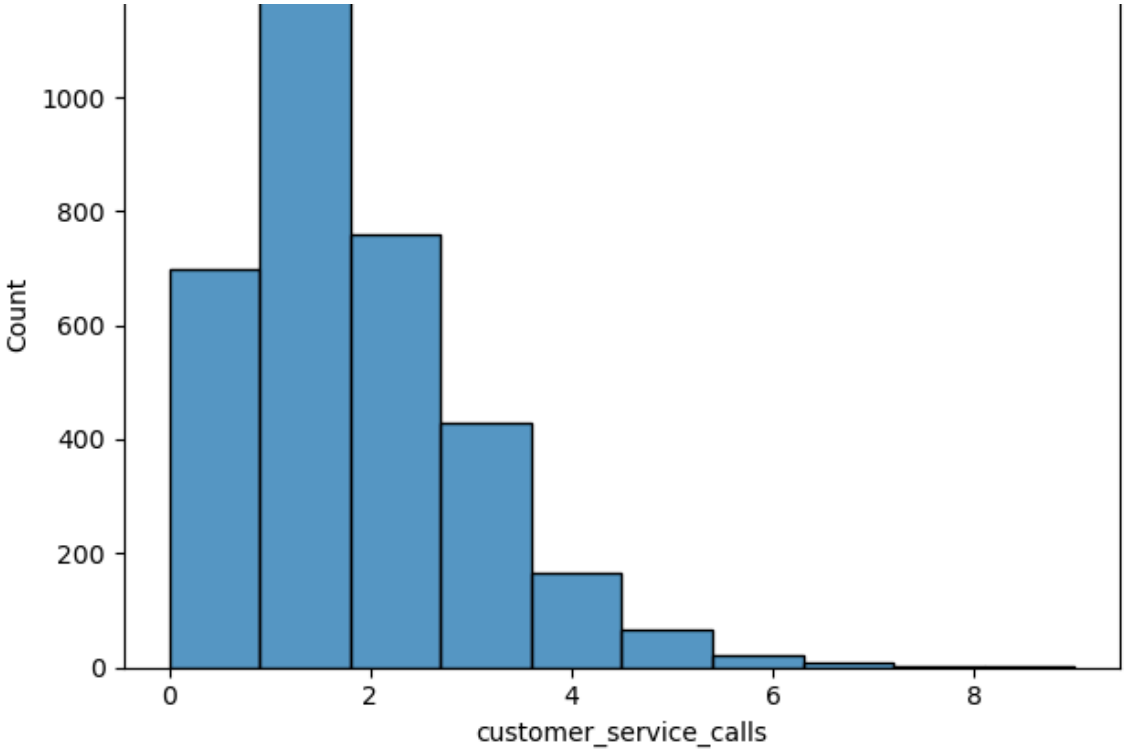
Univariate Analysis

In [16]:
```
#check for imbalance in churn and plans
label = ['no','yes']
fig,(ax1,ax2,ax3) = plt.subplots(1,3,figsize=(10,5))
ax1.pie(data['churn'].value_counts(),autopct='%.2f%%',labels=label)
ax1.set_title("churn")
ax2.pie(data['international_plan'].value_counts(),autopct='%.2f%%',labe
ax2.set_title("international plan")
ax3.pie(data['voice_mail_plan'].value_counts(),autopct='%.2f%%',labels=
ax3.set_title("voice mail plan")
plt.tight_layout();
```



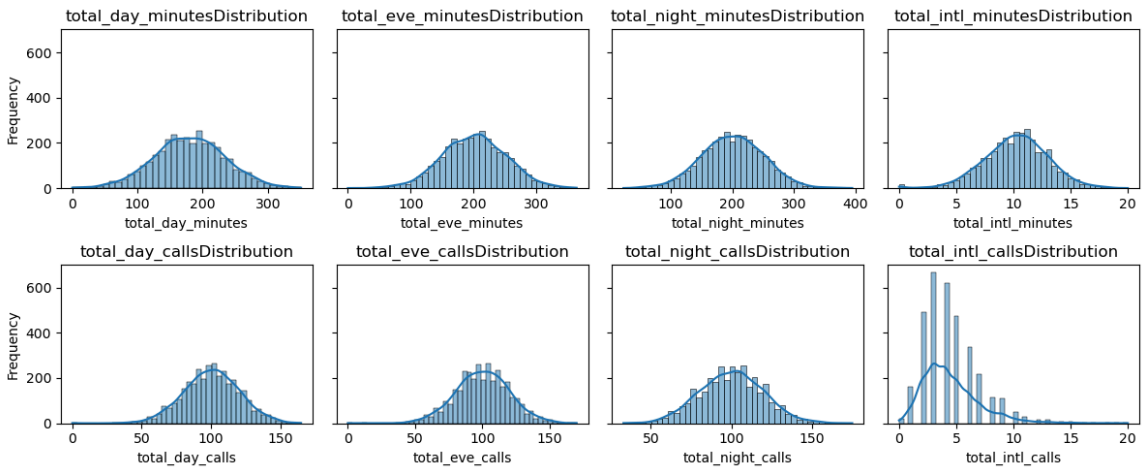
In [17]:
```
sns.histplot(data['customer_service_calls'],bins=10)
plt.title('customer service calls')
plt.tight_layout();
```





In [18]:

```
usage_cols = ['total_day_minutes','total_eve_minutes','total_night_minu
              'total_day_calls','total_eve_calls','total_night_calls',
fig,axes = plt.subplots(2,4,figsize=(12,5),sharey=True)
axes = axes.flatten()

for i,col in enumerate(usage_cols):
    sns.histplot(data[col],bins='auto',kde=True, ax=axes[i])
    axes[i].set_title(f"{col}Distribution")
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Frequency')
    plt.tight_layout();
```



In [19]:

```
data.columns
```

Out[19]:

```
Index(['state', 'account_length', 'area_code', 'international_plan',
       'voice_mail_plan', 'number_vmail_messages', 'total_day_minute
s',
       '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_minute
s',
       'total_intl_calls', 'total_intl_charge', 'customer_service_call
s'
```

```
        'churn'],
      dtype='object')
```

Observations

- 15% of the customers churn; we have an imbalance
- Most customers call twice while few call upto 8 times; Do those who call more than 6 times churn?
- The usage columns are normally distributed apart from total internation calls
- customer service calls column is skewed to the right
- intl plan and voicemail plan columns are imbalanced; we may use SMOTE

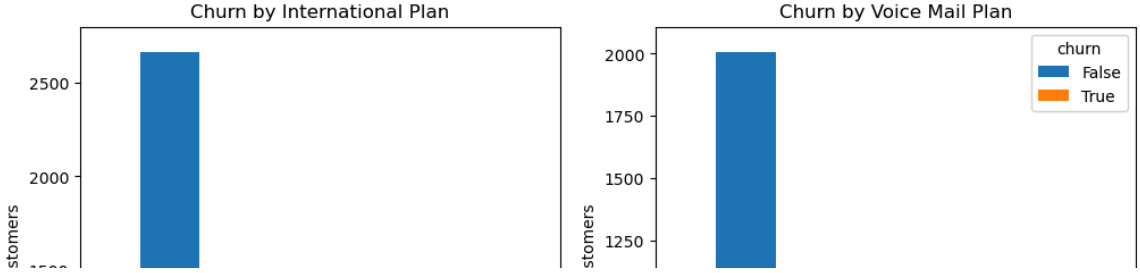
Bivariate analysis

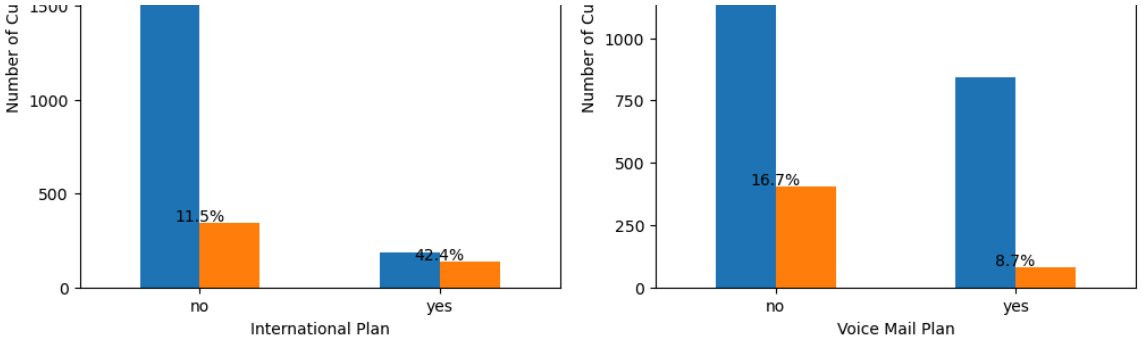
- We will compare features to churn

In [20]:

```
#H0: there is no association between customer plan and churn
# H1: there is an association between customer plan and churn

customer_plan = data[['international_plan','voice_mail_plan']]
chi = {}
for col in customer_plan:
    contigency = pd.crosstab(data[col],data['churn'])
    chi2,p_value,_,_ = chi2_contingency(contigency)
    chi[col] = p_value
print(f"p_value for international plan is {chi['international_plan']:.4

#let's visualize to see the association
fig, axes = plt.subplots(1, 2, figsize=(12, 6))

# Loop through both subplots and columns
for ax, col in zip(axes, customer_plan.columns):
    # Get contingency table
    customer_plan_ct = pd.crosstab(data[col], data['churn'])
    customer_plan_ct.plot(kind='bar', ax=ax, legend=(col == 'voice_mai
    churn_rate = (customer_plan_ct[True] / customer_plan_ct.sum(axis=1
    # Add churn % labels on top of orange bars

    for i in range(len(customer_plan_ct)):
        height = customer_plan_ct.iloc[i][True]
        percent = churn_rate.iloc[i]
        ax.text(i, height + 10, f"{percent:.1f}%", ha='center', color=

    # Titles and axis labels
    ax.set_title(f"Churn by {col.replace('_', ' ').title()}")
    ax.set_xlabel(col.replace('_', ' ').title())
    ax.set_ylabel("Number of Customers")
    ax.set_xticklabels(customer_plan_ct.index, rotation=0);
```

```
p_value for international plan is 0.0000
p-value forvoice mail plan is 0.0000
```





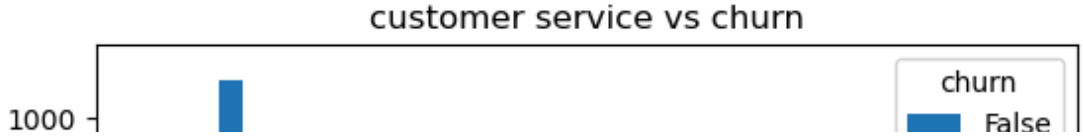
In [21]:

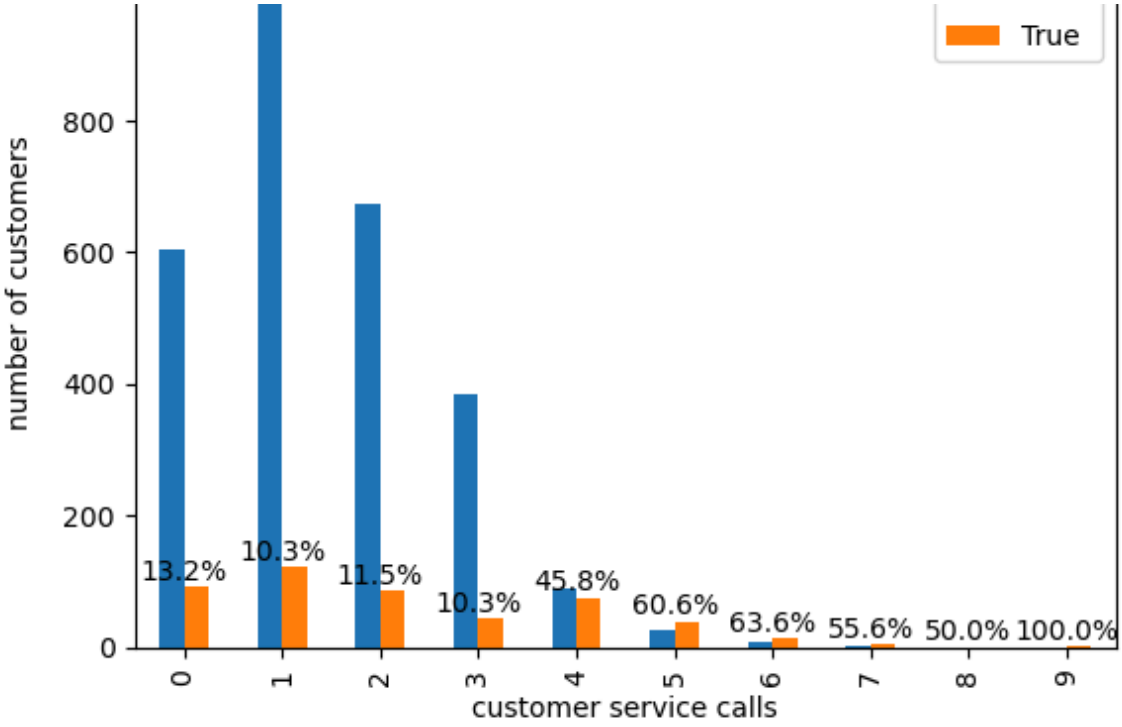
```
# H0: There is no significant difference in means of churn vs non-churn
# H1: There is significant difference in means of churn vs non-churn in

num_cols = ['account_length', 'total_day_minutes','total_day_calls', 't
            'total_night_calls', 'total_intl_minutes','total_intl_cal
            'total_eve_charge','total_night_charge','total_intl_charge
anova= []
for col in num_cols:
    group0 = data[data['churn']==0][col]
    group1 = data[data['churn']==1][col]
    t_stat,p_value = ttest_ind(group0,group1,equal_var=False)
    is_significant = p_value < 0.05
    anova.append({'Feature': col,'p-value': p_value,'Significant (<0.05

anova_df = pd.DataFrame(anova)
anova_df = anova_df.sort_values(by='p-value')
print(anova_df[anova_df['Significant (<0.05)']])
```

	Feature	p-value	Significant (<0.05)
1	total_day_minutes	1.218968e-20	True
11	total_day_charge	1.219876e-20	True
9	customer_service_calls	5.270040e-18	True
10	number_vmail_messages	8.764782e-09	True
3	total_eve_minutes	1.839080e-07	True
12	total_eve_charge	1.842608e-07	True
14	total_intl_charge	9.025887e-05	True
7	total_intl_minutes	9.065715e-05	True
8	total_intl_calls	3.185777e-03	True
13	total_night_charge	3.027154e-02	True
5	total_night_minutes	3.028049e-02	True

In [22]:

```
# customer service by churn
service_ct = pd.crosstab(data['customer_service_calls'],data['churn'])
ax5 = service_ct.plot(kind='bar')

service_churn_rate = (service_ct[True] / service_ct.sum(axis=1)) *100
service_bar_position = range(len(service_ct))
for i in service_bar_position:
    height_service = service_ct.iloc[i][True]
    service_percent = service_churn_rate.iloc[i]
    ax5.text(i, height_service + 10, f"{service_percent:.1f}%", ha='cen

plt.title('customer service vs churn')
plt.xlabel("customer service calls")
plt.ylabel('number of customers')
plt.legend(title='churn');
```



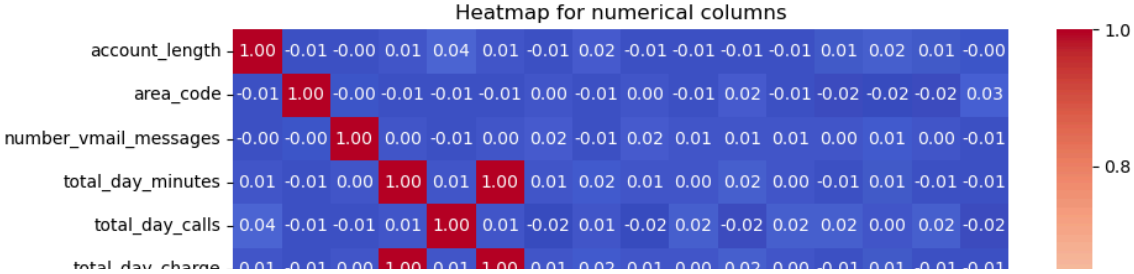


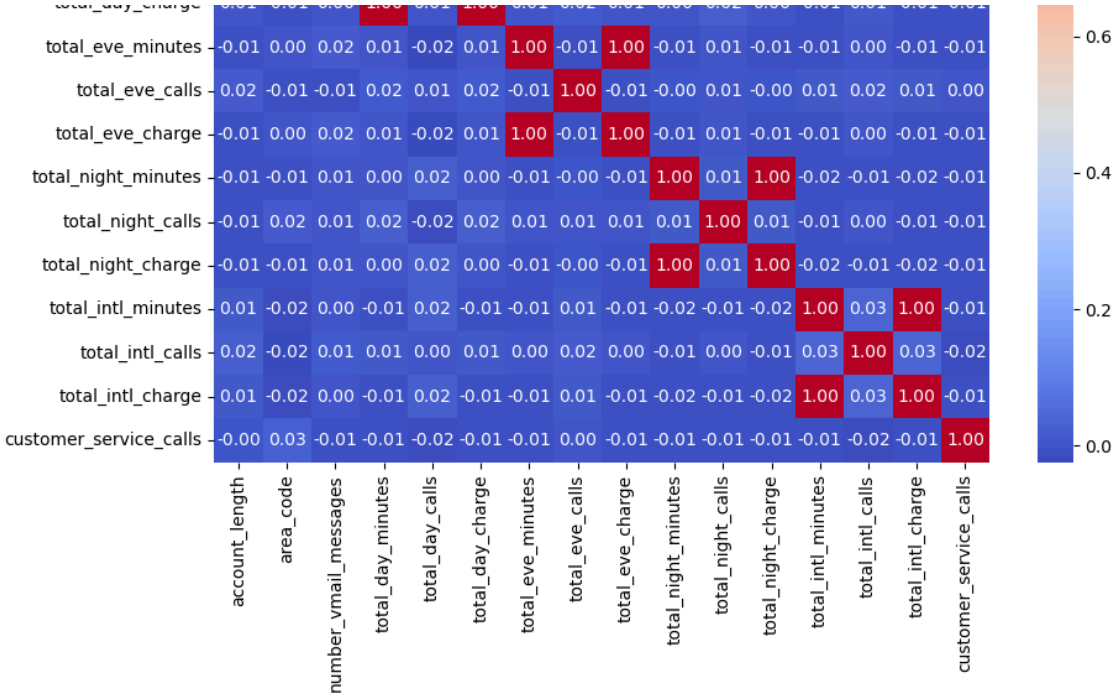
observations:

- Customers' plan are statistically significant, this means that we have an association between customer plan and churn
  - The churn rate for those with international plan is higher by about 30%
  - The churn rate for those without voice mail plan is higher by about 8%; they never get helped
- For numerical values:
  - total day minutes, total day charge, total eve minutes, total eve charge, total night minutes, total night charge, total intl minutes, total intl calls, total intl charge, customer service calls, number vmail messages are statistically significant; we reject H0
- Customer service calls vs churn
  - There is a 50% or more churn for customers who called 5 or more times

Multivariate Analysis

In [23]:
```
corr = data.select_dtypes("number").corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr,cmap='coolwarm',annot=True, fmt='.2f')
plt.title("Heatmap for numerical columns")
plt.tight_layout();
```





Observation:

- total minutes and total charges are highly correlated, this could be because miutes was used to calculate charges
  - since we saw that charges were statistically significant to churn, we will use charges as opposed to minutes

Feature Engineering

In [24]:
```
#aggregate columns
data['total_calls'] =data['total_day_calls']+ data['total_eve_calls']+
data['total_charge'] = data['total_day_charge']+ data['total_eve_charge
data.columns
```

Out[24]:
```
Index(['state', 'account_length', 'area_code', 'international_plan',
       'voice_mail_plan', 'number_vmail_messages', 'total_day_minute
s',
       '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_minute
s',
       'total_intl_calls', 'total_intl_charge', 'customer_service_call
s',
       'churn', 'total_calls', 'total_charge'],
      dtype='object')
```

In [25]:
```
#drop redundant features
data = data.drop(['total_day_minutes','total_eve_minutes','total_night_
```

Modeling

data preprocessing

In [26]:
```
data[:3]
```

Out[26]:

	account_length	area_code	international_plan	voice_mail_plan	number_vmail
0	128	415	no	yes	
1	107	415	no	yes	
2	137	415	no	no	

In [27]:
```
data.info(verbose=False)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Columns: 17 entries, account_length to total_charge
dtypes: float64(5), int64(9), object(3)
memory usage: 442.8+ KB
```

In [28]:
```
#label encode the target variable
le = LabelEncoder()
data['churn'] = le.fit_transform(data['churn'])
```

In [29]:
```
#one hot encode intl plan and voice mail plan features
cat = data[['international_plan','voice_mail_plan']]
cat_ohe = pd.get_dummies(cat,drop_first=True,dtype='int')

#merge the original data with the new cat columns
merged_df = pd.concat([data,cat_ohe],axis=1)
#drop the initial cat columns
merged_df.drop(columns=cat.columns, inplace=True,axis=1)
merged_df[:3]
```

Out[29]:

	account_length	area_code	number_vmail_messages	total_day_calls	total_da
0	128	415	25	110	
1	107	415	26	123	
2	137	415	0	114	

In [30]:
```
# separate features and target
X = merged_df.drop("churn",axis=1)
y = merged_df.churn


#split to train and test set
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,rand
X_train.shape ,X_test.shape ,y_train.shape ,y_test.shape
```

Out[30]: ((2666, 16), (667, 16), (2666,), (667,))

In [31]:
```
#check for correlation
merged_df.corr()['churn'].sort_values(ascending=False)
```

Out[31]:
```
churn                    1.000000
international_plan_yes   0.259852
total_charge             0.231549
customer_service_calls   0.208750
```

```
total_day_charge          0.205151
total_eve_charge          0.092786
total_intl_charge         0.068259
total_night_charge        0.035496
total_day_calls           0.018459
account_length            0.016541
total_calls               0.015807
total_eve_calls           0.009233
area_code                 0.006174
total_night_calls         0.006141
total_intl_calls         -0.052844
number_vmail_messages    -0.089728
voice_mail_plan_yes      -0.102148
Name: churn, dtype: float64
```

In [32]:

```
#balance the training set
smote = SMOTE(random_state=42)
X_train_sm, y_train_sm = smote.fit_resample(X_train,y_train)
#check on whether SMOTEN worked
print(f" Original values \n {y_train.value_counts()}\n")
print(f"Smoted values \n {y_train_sm.value_counts()}")
```

```
 Original values
 churn
0    2280
1     386
Name: count, dtype: int64

Smoted values
 churn
0    2280
1    2280
Name: count, dtype: int64
```

In [33]:

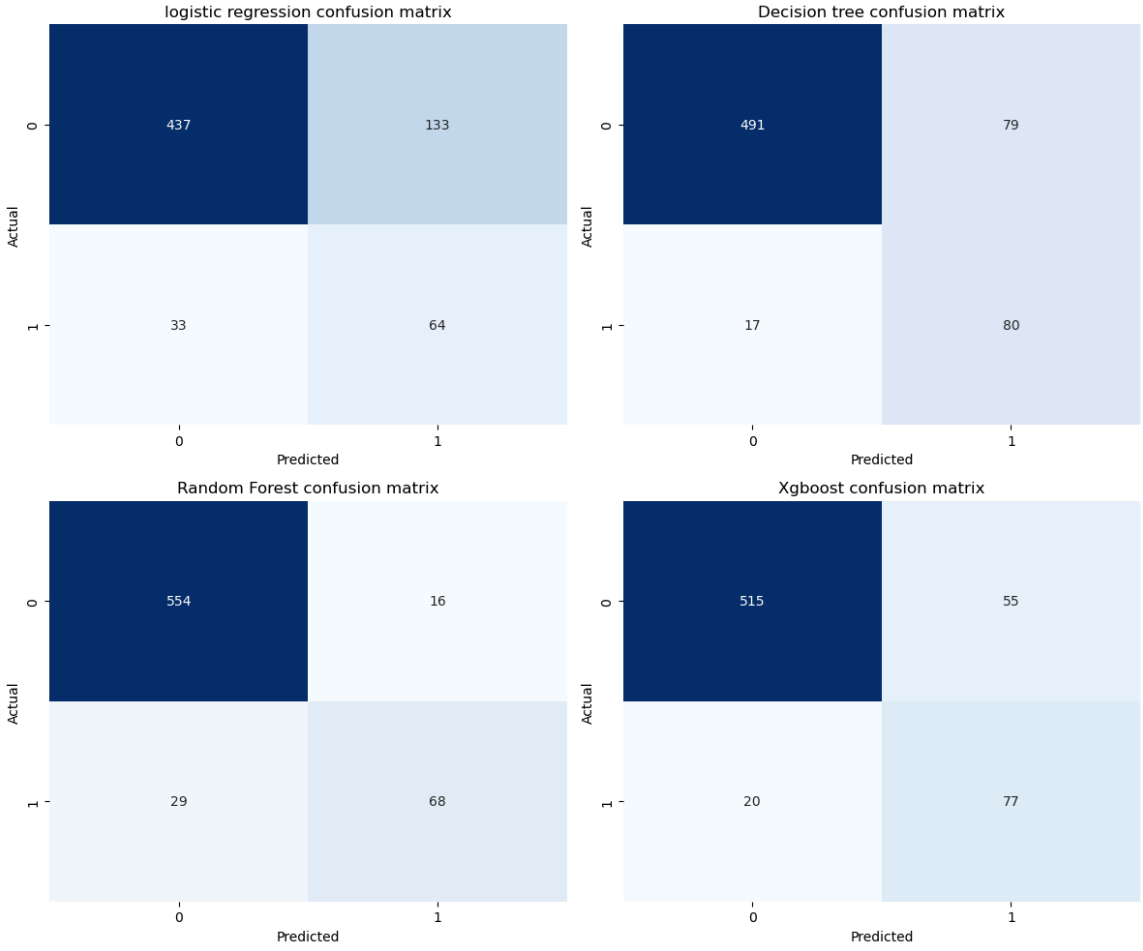
```
#scaling featues
ss = StandardScaler()
X_train_s = ss.fit_transform(X_train_sm)
X_test_s = ss.transform(X_test)
```

In [34]:

```
#fit models
models = {'logistic regression':LogisticRegression(random_state=42,max_
          'Decision tree': DecisionTreeClassifier(random_state=42),
          'Random Forest':RandomForestClassifier(random_state=42,n_esti
          'Xgboost':XGBClassifier(random_state=42,use_label_encoder=Fal
          }
results = []
conf_matrix = {}
for name, model in models.items():
    model.fit(X_train_s,y_train_sm)
    y_pred = model.predict(X_test_s)
    y_prob = model.predict_proba(X_test_s)[:,1]

    results.append({'Model':name,
                    'train score':model.score(X_train_s,y_train_sm),
                    'Accuracy':accuracy_score(y_test,y_pred)*100,
                    'Recall':recall_score(y_test,y_pred)*100,
                    'Precision':precision_score(y_test,y_pred)*100,
                    'F1 score':f1_score(y_test,y_pred)*100,
                    'AUC':roc_auc_score(y_test,y_pred)*100
                })
    conf_matrix[name] = confusion_matrix(y_test,y_pred)
```

```
results_df = pd.DataFrame(results).round(2)
results_df = results_df[["Model",'train score', "Accuracy", "Recall", "
print(results_df.to_string(index=False))

#confusion matrix
fig, axes = plt.subplots(2,2,figsize=(12,10))
axes = axes.flatten()
for i, (name,cm) in enumerate (conf_matrix.items()):
    sns.heatmap(cm, annot=True, cbar=False,cmap='Blues',fmt='d', ax=axe
    axes[i].set_title(f'{name} confusion matrix')
    axes[i].set_xlabel('Predicted')
    axes[i].set_ylabel('Actual')
plt.tight_layout()
plt.show()
```

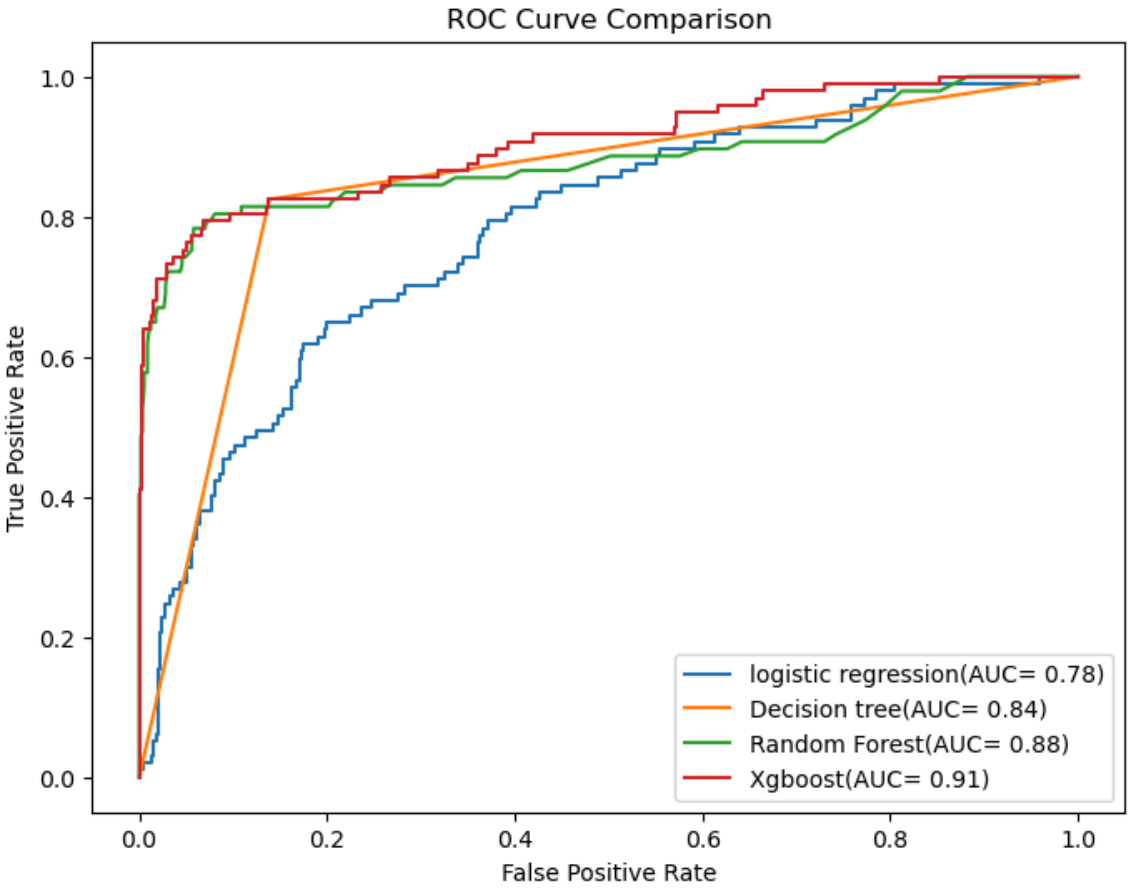
Model	train score	Accuracy	Recall	Precision	F1 score	AUC
logistic regression	0.79	75.11	65.98	32.49	43.54	71.32
Decision tree	1.00	85.61	82.47	50.31	62.50	84.31
Random Forest	1.00	93.25	70.10	80.95	75.14	83.65
Xgboost	1.00	88.76	79.38	58.33	67.25	84.87



Observation:

- Logistic regression misses a fair number of churners; very expensive
- Decision tree has a high recall but
- Random forest would be better if our priority was precision
- Xgboost is the best model with a balance of precision and recall (F1)

In [35]:
```
#plot the roc for all models
plt.figure(figsize=(8,6))
for name,model in models.items():
    y_prob = model.predict_proba(X_test_s)[:,1]
    fpr,tpr,_= roc_curve(y_test,y_prob)
    roc_auc = auc(fpr,tpr)
    plt.plot(fpr,tpr,label=f"{name}(AUC= {roc_auc:.2f})")

plt.title("ROC Curve Comparison")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend();
```



observations:

- Decision tree has a recall closest to our metrics of success
- So far, Xgboost is the best model to use.
- Based on our measure of success, xgboost has a recall of less than 85
- We focus on tuning it to improve the recall but first we will use cross validation

In [36]:
```
#using cross validation
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

scoring = {
    'accuracy': 'accuracy',
    'recall': 'recall',
    'precision': 'precision',
    'f1': 'f1',
}

# ---- Cross-validation ----
cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
```

```
cv_results = {}
for name, model in models.items():
    cv_result = cross_validate(
        model,
        X_res, y_res,
        cv=cv,
        scoring=scoring,
        return_train_score=True
    )
    cv_results[name] = {
        metric: cv_result['test_' + metric].mean() * 100
        for metric in scoring.keys()
    }
    cv_results[name]['train_accuracy'] = cv_result['train_accuracy'].me

# ---- Results DataFrame ----
cv_results_df = pd.DataFrame(cv_results).T.round(2)
print(cv_results_df)
```

	accuracy	recall	precision	f1	train_accuracy
logistic regression	89.60	82.49	96.16	88.80	89.63
Decision tree	86.75	88.84	85.31	87.02	100.00
Random Forest	94.84	92.21	97.35	94.70	100.00
Xgboost	93.96	95.58	92.62	94.07	99.96

observation:

- we will use cross validation to split our dataset
- xgboost seems to be the best model to use
- we will hyperparameter tune to find the right parameters to use

In [37]:

```
#hyperparameter tuning
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X, y)

base_models = {
    "Logistic Regression": LogisticRegression(class_weight="balanced",
    "Decision Tree": DecisionTreeClassifier(random_state=42,class_weigh
    "Random Forest": RandomForestClassifier(random_state=42, n_jobs=-1,
    "XGBoost": XGBClassifier(eval_metric="logloss", random_state=42)
}


param_grids = {
    "Logistic Regression": {
        "C": [0.01, 0.1, 1, 10],
        "penalty": ["l1", "l2"],
        "max_iter": [2000, 5000, 10000]
    },
    "Decision Tree": {
        "max_depth": [3, 5, 10],
        "min_samples_split": [5, 10, 20],
        "min_samples_leaf": [2, 5, 10]
    },
    "Random Forest": {
        "n_estimators": [100, 200],
        "max_depth": [5, 10, 15],
        "min_samples_split": [5, 10],
        "min_samples_leaf": [2, 5]
    },
    "XGBoost": {
        "n_estimators": [100, 200],
```

```
        "max_depth": [3, 5, 7],
        "learning_rate": [0.05, 0.1, 0.2],
        "subsample": [0.8, 1],
        "colsample_bytree": [0.8, 1],
        "reg_lambda": [1, 5],
        "reg_alpha": [0, 1]
    }
}

scoring = {
    "accuracy": "accuracy",
    "recall": "recall",
    "precision": "precision",
    "f1": "f1",
    "roc_auc": "roc_auc",
    "pr_auc": "average_precision",
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

best_models = {}
cv_results = {}

for name, model in base_models.items():

    grid = GridSearchCV(
        model,
        param_grids[name],
        cv=cv,
        scoring="recall",
        n_jobs=-1,
        verbose=0
    )
    grid.fit(X_res, y_res)
    best_model = grid.best_estimator_

    if name == "Decision Tree":
        best_model = CalibratedClassifierCV(best_model, cv=5)
        best_model.fit(X_res, y_res)

    best_models[name] = best_model

    cv_result = cross_validate(
        best_model,
        X_res, y_res,
        cv=cv,
        scoring=scoring,
        return_train_score=True
    )
    cv_results[name] = {
        metric: cv_result["test_" + metric].mean() * 100
        for metric in scoring.keys()
    }
    cv_results[name]["train_accuracy"] = cv_result["train_accuracy"].me
    cv_results[name]["best_params"] = grid.best_params_

cv_results_df = pd.DataFrame(cv_results).T
cv_results_df = cv_results_df.applymap(lambda x: round(x, 2) if isinsta

print(cv_results_df)
```

```
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
```

```
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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
```

```
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
egression
  n_iter_i = _check_optimize_result(
/opt/anaconda3/envs/learn-env/lib/python3.9/site-packages/sklearn/linear
_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (s
tatus=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-r
```

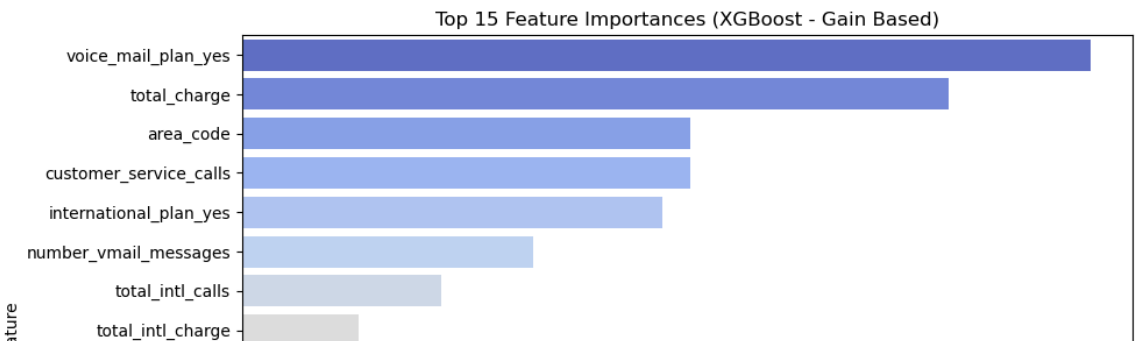
```
egression
  n_iter_i = _check_optimize_result(
                     accuracy  recall  precision     f1  roc_auc  pr_auc
\
Logistic Regression     89.56   83.33      95.20  88.87    95.43   96.54
Decision Tree           87.91   80.67      94.34  86.97    94.30   95.55
Random Forest           92.74   87.86      97.36  92.36    98.10   98.40
XGBoost                 95.05   93.26      96.73  94.96    98.58   98.84

                     train_accuracy  \
Logistic Regression           89.67
Decision Tree                 91.58
Random Forest                 98.88
XGBoost                      100.00

                                                          best_params
Logistic Regression      {'C': 0.1, 'max_iter': 5000, 'penalty': 'l2'}
Decision Tree        {'max_depth': 10, 'min_samples_leaf': 5, 'min_...
Random Forest        {'max_depth': 15, 'min_samples_leaf': 2, 'min_...
XGBoost              {'colsample_bytree': 1, 'learning_rate': 0.2, ...
```

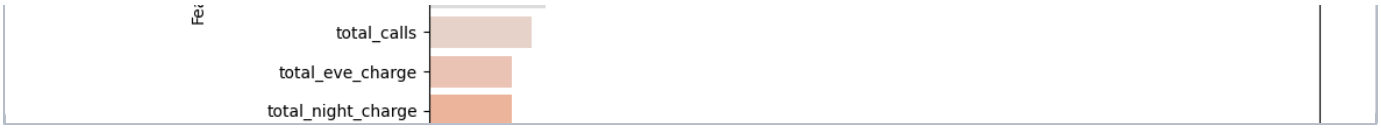
observation: Model train score Accuracy Recall Precision F1 score AUC logistic regression 0.79 75.11 65.98 32.49 43.54 71.32 Decision tree 1.00 85.61 82.47 50.31 62.50 84.31 Random Forest 1.00 93.25 70.10 80.95 75.14 83.65 Xgboost 1.00 88.76 79.38 58.33 67.25 84.87

- Imbalance handled with SMOTE; recall boosted from 65% (Logistic baseline) to >90% (XGBoost).
- Overfitting addressed; Decision Tree pruned with parameters
- from feature importance: international plan, customer service calls, and high day charges.
- Business Value: With XGBoost, we can now identify >93% of churners, directly supporting customer retention strategies.

In [38]:

```
#Feature Importance using XGBoost
best_xgb = best_models["XGBoost"]

xgb_importances = best_xgb.feature_importances_
feat_imp = pd.DataFrame({
    "Feature": X.columns,
    "Importance": xgb_importances
}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10,6))
sns.barplot(x="Importance", y="Feature", data=feat_imp.head(15), palett
plt.title("Top 15 Feature Importances (XGBoost - Gain Based)")
plt.tight_layout()
plt.show()
```




Fe
total_calls
total_eve_charge
total_night_charge