Customer churn

Business Objective:

Customer churn is a big problem for telecommunications companies. Indeed, their annual churn rates are usually higher than 10%. For that reason, they develop strategies to keep as many clients as possible. This is a classification project since the variable to bepredicted is binary (churn or loyal customer). The goal here is to model churn probability, conditioned on the customer features.

Data Set Details:

```
Each row corresponds to a client of a telecommunications
company for whom it has collected information about the type of plan they
have contracted, the minutes they have talked, or the charge they pay every
month.
The data set includes the following variables:

● state: Categorical, for the 51 states and the District of Columbia.
● Area.code
● account.length: how long the account has been active.
● voice.plan: yes or no, voicemail plan.
● voice.messages: number of voicemail messages.
● intl.plan: yes or no, international plan.
● intl.mins: minutes customer used service to make international calls.
● intl.calls: total number of international calls.
● intl.charge: total international charge.
● day.mins: minutes customer used service during the day.
● day.calls: total number of calls during the day.
● day.charge: total charge during the day.
● eve.mins: minutes customer used service during the evening.
● eve.calls: total number of calls during the evening.
● eve.charge: total charge during the evening.
● night.mins: minutes customer used service during the night.
● night.calls: total number of calls during the night.
● night.charge: total charge during the night.
● customer.calls: number of calls to customer service.
● churn: Categorical, yes or no. Indicator of whether the customer has
left the company (yes or no).
```

In [1]:

```
#Importing Neccesary Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings

%matplotlib inline
warnings.filterwarnings("ignore")
```

In [2]:

```
#Importing dataset
data = pd.read_csv('Churn.csv')
```

In [3]:

```
data.head()
```

Out[3]:

	Unnamed: 0	state	area.code	account.length	voice.plan	voice.messages	intl.plan	int
0	1	KS	area_code_415	128	yes	25	no	
1	2	OH	area_code_415	107	yes	26	no	
2	3	NJ	area_code_415	137	no	0	no	
3	4	OH	area_code_408	84	no	0	yes	
4	5	OK	area_code_415	75	no	0	yes	

5 rows × 21 columns

In [4]:

```
df=data.drop(['Unnamed: 0'], axis=1)
```

In [5]:

```
df.head()
```

Out[5]:

	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	intl.c
0	KS	area_code_415	128	yes	25	no	10.0	
1	OH	area_code_415	107	yes	26	no	13.7	
2	NJ	area_code_415	137	no	0	no	12.2	
3	OH	area_code_408	84	no	0	yes	6.6	
4	OK	area_code_415	75	no	0	yes	10.1	

In [6]:

```
#Rows and columns
df.shape
```

Out[6]:

```
(5000, 20)
```

In [7]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   state           5000 non-null   object 
 1   area.code       5000 non-null   object 
 2   account.length  5000 non-null   int64  
 3   voice.plan      5000 non-null   object 
 4   voice.messages  5000 non-null   int64  
 5   intl.plan       5000 non-null   object 
 6   intl.mins       5000 non-null   float64
 7   intl.calls      5000 non-null   int64  
 8   intl.charge     5000 non-null   float64
 9   day.mins        5000 non-null   float64
 10  day.calls       5000 non-null   int64  
 11  day.charge      5000 non-null   object 
 12  eve.mins        5000 non-null   object 
 13  eve.calls       5000 non-null   int64  
 14  eve.charge      5000 non-null   float64
 15  night.mins      5000 non-null   float64
 16  night.calls     5000 non-null   int64  
 17  night.charge    5000 non-null   float64
 18  customer.calls  5000 non-null   int64  
 19  churn           5000 non-null   object 
dtypes: float64(6), int64(7), object(7)
memory usage: 781.4+ KB
```

In [8]:

```
df.dtypes
```

Out[8]:

```
state               object
area.code           object
account.length       int64
voice.plan          object
voice.messages       int64
intl.plan           object
intl.mins          float64
intl.calls           int64
intl.charge        float64
day.mins           float64
day.calls            int64
day.charge          object
eve.mins            object
eve.calls            int64
eve.charge         float64
night.mins         float64
night.calls          int64
night.charge       float64
customer.calls       int64
churn               object
dtype: object
```

Report

- we have 2 features with wrong data type which is day.charge and eve.mins

In [9]:

```
df['day.charge']= df['day.charge'].astype(float)
```

In [10]:

```
df['eve.mins']= df['eve.mins'].astype(float)
```

In [11]:

```
df.dtypes
```

Out[11]:

```
state               object
area.code           object
account.length       int64
voice.plan          object
voice.messages       int64
intl.plan           object
intl.mins          float64
intl.calls           int64
intl.charge        float64
day.mins           float64
day.calls            int64
day.charge         float64
eve.mins           float64
eve.calls            int64
eve.charge         float64
night.mins         float64
night.calls          int64
night.charge       float64
customer.calls       int64
churn               object
dtype: object
```

In [12]:

```
#Extracting cat feature from the dataset
cat_feature = [feature for feature in df.columns if df[feature].dtypes == 'O']
```

In [13]:

```
cat_feature
```

Out[13]:

```
['state', 'area.code', 'voice.plan', 'intl.plan', 'churn']
```

In [14]:

```
#Extracting cat feature from the dataset
num_feature = [feature for feature in df.columns if df[feature].dtypes != 'O']
```

In [15]:

```
num_feature
```

Out[15]:

```
['account.length',
 'voice.messages',
 'intl.mins',
 'intl.calls',
 'intl.charge',
 'day.mins',
 'day.calls',
 'day.charge',
 'eve.mins',
 'eve.calls',
 'eve.charge',
 'night.mins',
 'night.calls',
 'night.charge',
 'customer.calls']
```

In [16]:

```
#checking null values if there are any
df.isnull().sum()
```

Out[16]:

```
state              0
area.code          0
account.length     0
voice.plan         0
voice.messages     0
intl.plan          0
intl.mins          0
intl.calls         0
intl.charge        0
day.mins           0
day.calls          0
day.charge         7
eve.mins          24
eve.calls          0
eve.charge         0
night.mins         0
night.calls        0
night.charge       0
customer.calls     0
churn              0
dtype: int64
```

In [17]:

```
#Extracting unique value from the dataset
for feature in df.columns:
    print(f"feature {feature} has {df[feature].unique()} unique values \n")
```

```
feature state has ['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'] unique values

feature area.code has ['area_code_415' 'area_code_408' 'area_code_510'] un
ique values

feature account.length has [128 107 137  84  75 118 121 147 117 141  65  7
4 168  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   4 188 204 221 187  14 238 216
 222 233] unique values

feature voice.plan has ['yes' 'no'] unique values

feature voice.messages has [25 26  0 24 37 27 33 39 30 41 28 34 46 29 35 2
1 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  6 52]
unique values

feature intl.plan has ['no' 'yes'] unique values

feature intl.mins has [10.  13.7 12.2  6.6 10.1  6.3  7.5  7.1  8.7 11.2 1
2.7  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 18.7 16.8  0.4 19.3 19.2 19.7
 18.5 17.7] unique values

feature intl.calls has [ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13 18
14 16 20 17] unique values

feature intl.charge has [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 5.05 4.54 0.11 5.21 5.18 5.32
 5.   4.78] unique values

feature day.mins has [265.1 161.6 243.4 ... 188.7   7.2 170. ] unique valu
es

feature day.calls has [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  50  34  39  46] unique value
s

feature day.charge has [45.07 27.47 41.38 ... 32.08  1.22 28.9 ] unique va
lues

feature eve.mins has [197.4 195.5 121.2 ... 302.3 280.6 340.3] unique valu
es

feature eve.calls has [ 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  47 169  38]
unique values

feature eve.charge has [16.78 16.62 10.3  ... 25.7  23.85 28.93] unique va
lues

feature night.mins has [244.7 254.4 162.6 ...  75.1 297.5 224.4] unique va
lues

feature night.calls has [ 91 103 104  89 121 118  96  90  97 111  94 128 1
15  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  40 168 161 159 160 170
  41  12 165  43   0] unique values

feature night.charge has [11.01 11.45  7.32 ...  4.65  3.65  3.38] unique
values

feature customer.calls has [1 0 2 3 4 5 7 9 6 8] unique values

feature churn has ['no' 'yes'] unique values
```

In [18]:

```
df.columns
```

Out[18]:

```
Index(['state', 'area.code', 'account.length', 'voice.plan', 'voice.messag
es',
       'intl.plan', 'intl.mins', 'intl.calls', 'intl.charge', 'day.mins',
       'day.calls', 'day.charge', 'eve.mins', 'eve.calls', 'eve.charge',
       'night.mins', 'night.calls', 'night.charge', 'customer.calls', 'chu
rn'],
      dtype='object')
```

In [19]:

```
df[df['state']=='IA'].head(50)
```

Out[19]:

	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	i
12	IA	area_code_408	168	no	0	no	11.2	
14	IA	area_code_415	62	no	0	no	13.1	
50	IA	area_code_408	52	no	0	no	7.8	
100	IA	area_code_510	98	yes	21	no	4.4	
150	IA	area_code_408	113	no	0	no	10.6	
162	IA	area_code_510	141	yes	36	no	10.5	
227	IA	area_code_408	126	yes	27	no	9.6	
303	IA	area_code_415	158	no	0	no	9.1	
328	IA	area_code_510	76	no	0	no	12.5	
489	IA	area_code_415	130	no	0	no	12.3	
699	IA	area_code_415	98	no	0	no	11.0	
855	IA	area_code_510	66	no	0	no	6.2	
887	IA	area_code_408	128	no	0	no	7.6	
912	IA	area_code_510	45	no	0	no	10.6	
941	IA	area_code_510	63	no	0	no	6.5	
1113	IA	area_code_415	152	no	0	no	10.6	
1175	IA	area_code_415	134	yes	32	no	8.6	
1206	IA	area_code_510	92	yes	25	no	9.7	
1234	IA	area_code_408	86	no	0	no	9.8	
1266	IA	area_code_415	42	no	0	no	9.0	
1300	IA	area_code_510	46	no	0	no	10.2	
1356	IA	area_code_415	118	no	0	no	11.3	
1494	IA	area_code_415	129	no	0	no	4.9	
1527	IA	area_code_510	36	no	0	no	9.0	
1530	IA	area_code_510	81	no	0	no	8.0	
1605	IA	area_code_415	73	no	0	no	11.1	
1838	IA	area_code_408	1	yes	26	no	8.1	
2031	IA	area_code_510	130	no	0	no	11.4	
2035	IA	area_code_510	81	no	0	no	8.7	
2040	IA	area_code_510	105	yes	15	no	9.7	
2085	IA	area_code_415	75	no	0	no	11.1	
2149	IA	area_code_415	120	yes	33	no	11.6	
2261	IA	area_code_408	100	no	0	no	9.4	
2416	IA	area_code_510	113	no	0	no	11.9	
2561	IA	area_code_510	143	yes	33	no	7.7	
2570	IA	area_code_415	64	yes	43	no	8.5	
2632	IA	area_code_415	89	yes	35	no	11.8	

	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	i
2661	IA	area_code_415	197	no	0	no	9.5	
2677	IA	area_code_415	44	no	0	no	7.3	
2783	IA	area_code_415	79	yes	17	no	9.1	
2856	IA	area_code_415	123	no	0	no	10.0	
3125	IA	area_code_510	40	no	0	no	10.5	
3193	IA	area_code_415	88	no	0	no	10.8	
3308	IA	area_code_415	45	no	0	no	13.3	
3338	IA	area_code_415	117	no	0	no	6.9	
3365	IA	area_code_415	92	no	0	no	10.0	
3424	IA	area_code_408	82	no	0	no	10.3	
3432	IA	area_code_408	74	no	0	no	9.0	
3476	IA	area_code_408	93	no	0	no	12.6	
3481	IA	area_code_510	157	no	0	no	13.6	

In [20]:

```
# Observation in few columns we have anonymous value "Nan" we will replace it with np.na
```

In [21]:

```
#Basic stats about the dataset
df.describe().T
```

Out[21]:

	count	mean	std	min	25%	50%	75%	max
account.length	5000.0	100.258600	39.694560	1.0	73.000	100.00	127.00	243.00
voice.messages	5000.0	7.755200	13.546393	0.0	0.000	0.00	17.00	52.00
intl.mins	5000.0	10.261780	2.761396	0.0	8.500	10.30	12.00	20.00
intl.calls	5000.0	4.435200	2.456788	0.0	3.000	4.00	6.00	20.00
intl.charge	5000.0	2.771196	0.745514	0.0	2.300	2.78	3.24	5.40
day.mins	5000.0	180.288900	53.894699	0.0	143.700	180.10	216.20	351.50
day.calls	5000.0	100.029400	19.831197	0.0	87.000	100.00	113.00	165.00
day.charge	4993.0	30.653501	9.166356	0.0	24.430	30.62	36.75	59.76
eve.mins	4976.0	200.580326	50.554637	0.0	166.275	201.00	234.10	363.70
eve.calls	5000.0	100.191000	19.826496	0.0	87.000	100.00	114.00	170.00
eve.charge	5000.0	17.054322	4.296843	0.0	14.140	17.09	19.90	30.91
night.mins	5000.0	200.391620	50.527789	0.0	166.900	200.40	234.70	395.00
night.calls	5000.0	99.919200	19.958686	0.0	87.000	100.00	113.00	175.00
night.charge	5000.0	9.017732	2.273763	0.0	7.510	9.02	10.56	17.77
customer.calls	5000.0	1.570400	1.306363	0.0	1.000	1.00	2.00	9.00

In [22]:

```
#Checking unique values for target varible
df['churn'].value_counts()
```

Out[22]:

```
no     4293
yes     707
Name: churn, dtype: int64
```

In [23]:

```
100*df['churn'].value_counts()/len(data['churn'])
```

Out[23]:

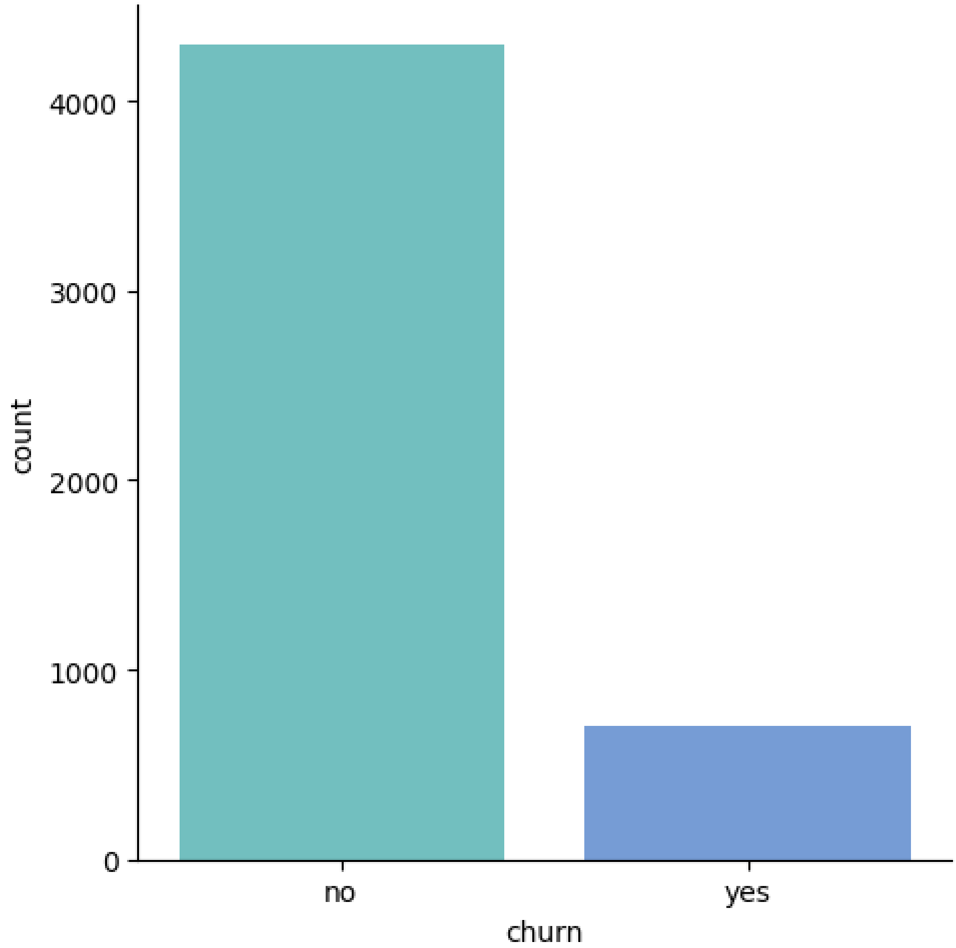
```
no     85.86
yes    14.14
Name: churn, dtype: float64
```

Visualization of target feature on unique values

In [24]:

```
yes = df[df['churn']=='yes'].shape[0]
no = df[df['churn']=='no'].shape[0]
print("No: " + str(no) + ", yes: " + str(yes))
sns.catplot(data=df, x="churn", kind="count", palette="winter_r", alpha=.6)
plt.show()
```

```
yes: 4293, yes: 707
```



Report

The target feature are highly imbalanced

Class imbalance is a scenario that arises when we have unequal distribution of class in a dataset i.e. the no. of data points in the no churn (majority class) very large compared to that of the yes churn (minority class)

If the imbalanced data is not treated beforehand, then this will degrade the performance of the classifier model.

Hence we should handle imbalanced data with certain methods.

How to handle Imbalance Data ?

Resampling data is one of the most commonly preferred approaches to deal with an imbalanced dataset. There are broadly two types of methods for this i) Undersampling ii) Oversampling. In most cases, oversampling is preferred over undersampling techniques. The reason being, in undersampling we tend to remove instances from data that may be carrying some important information.

SMOTE: Synthetic Minority Oversampling Technique

SMOTE is an oversampling technique where the synthetic samples are generated for the minority class.

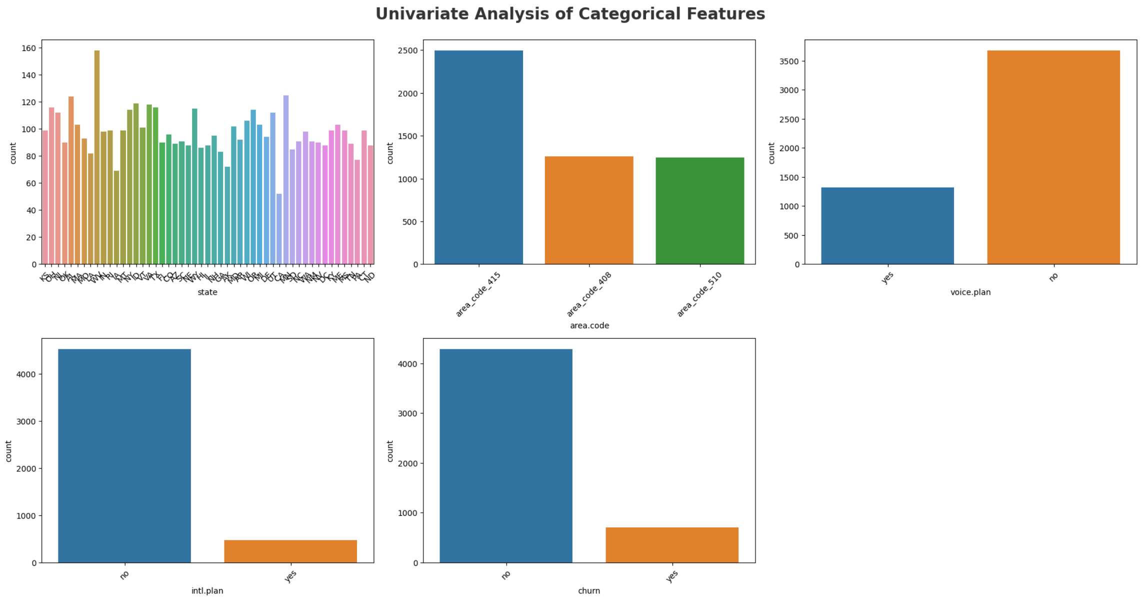
Hybridization techniques involve combining both undersampling and oversampling techniques. This is done to optimize the performance of classifier models for the samples created as part of these techniques.

It only duplicates the data and it won't add and new information. Hence we look at some different techniques.

Visulization of Categorical feature

In [25]:

```
# categorical columns
plt.figure(figsize=(20, 15))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight='bol
for i in range(0, len(cat_feature)):
    plt.subplot(3, 3, i+1)
    sns.countplot(x=df[cat_feature[i]])
    plt.xlabel(cat_feature[i])
    plt.xticks(rotation=45)
    plt.tight_layout()
```

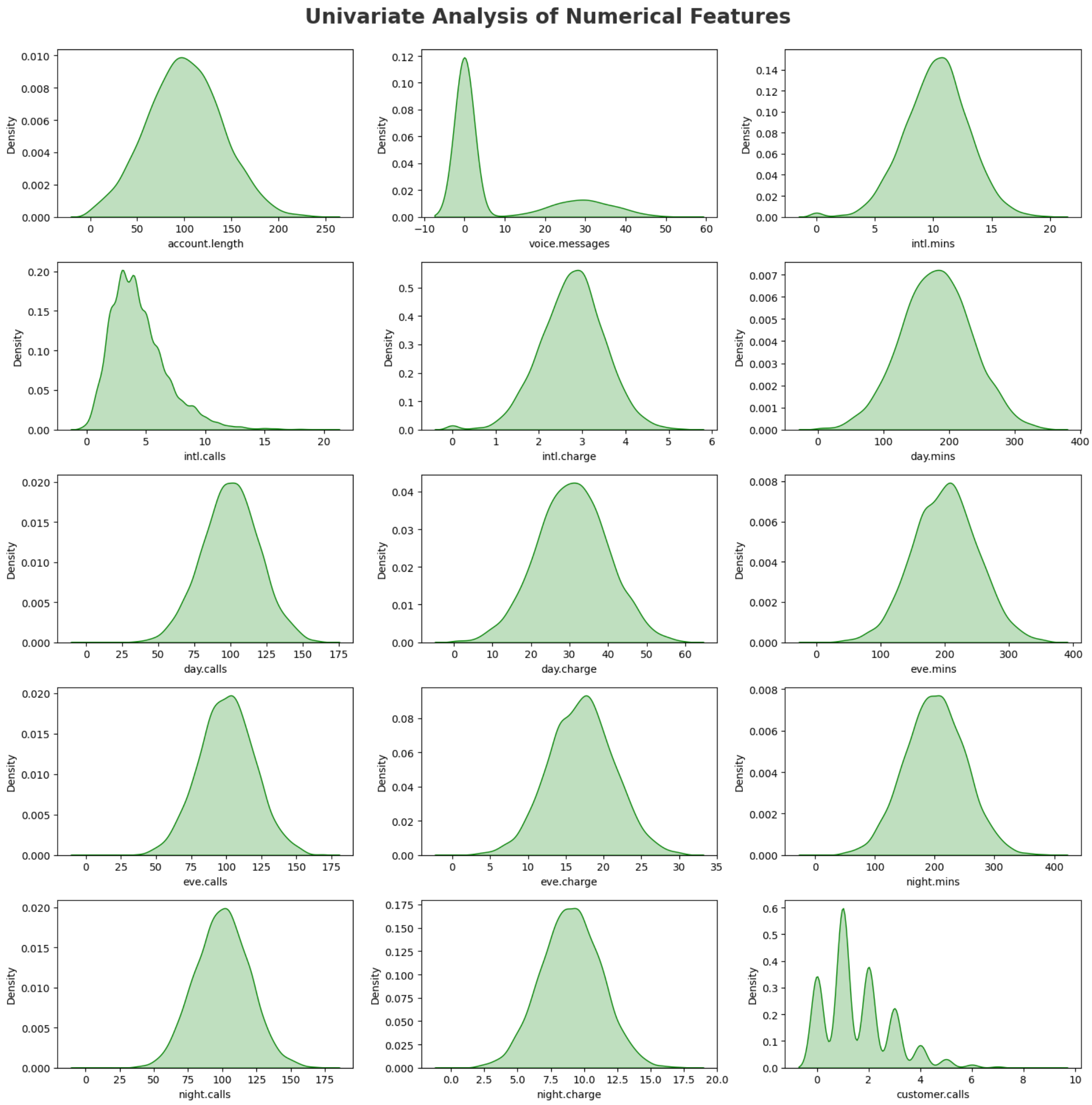


Visualization of numerical features

In [26]:

```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold'

for i in range(0, len(num_feature)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[num_feature[i]],shade=True, color='g')
    plt.xlabel(num_feature[i])
    plt.tight_layout()
```



Report

Most of the features are normally distrubuted only few features are little bit skewed towards right or left

Check Multicollinearity in Numerical features

In [27]:

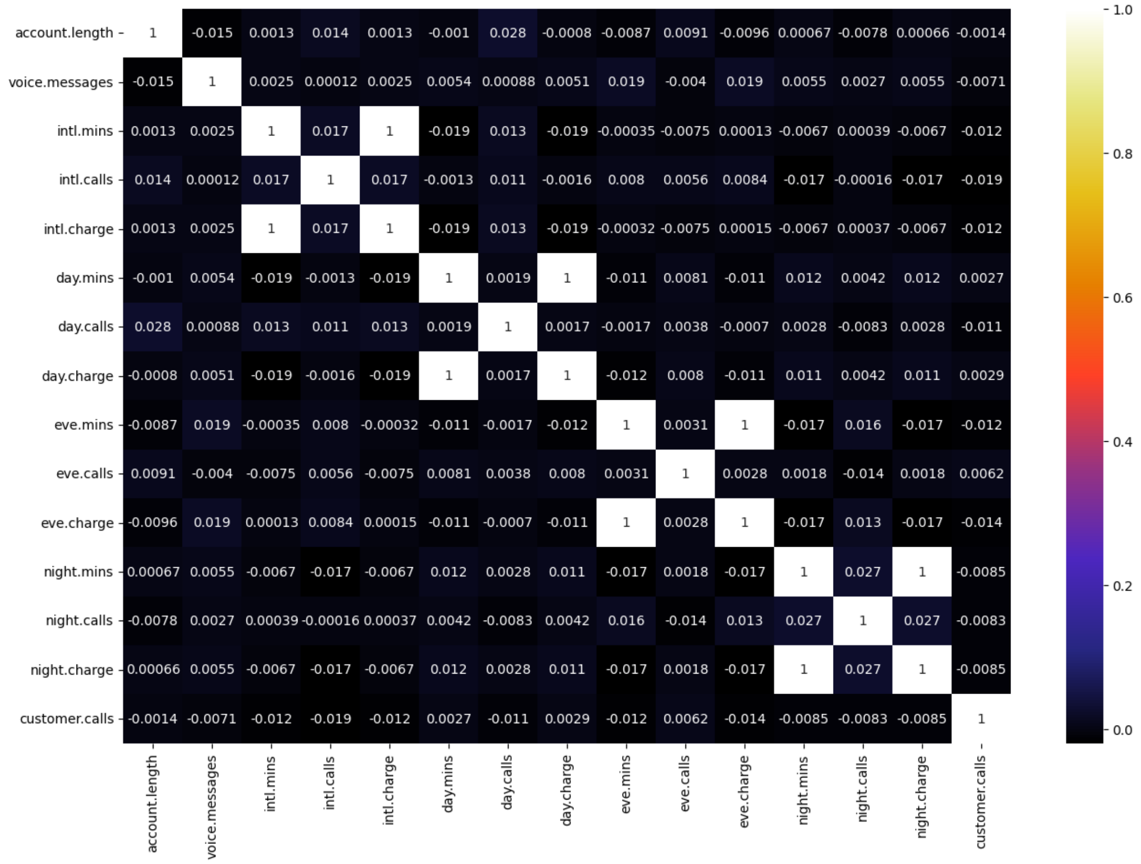
```
df.corr()
```

Out[27]:

	account.length	voice.messages	intl.mins	intl.calls	intl.charge	day.mins
account.length	1.000000	-0.014575	0.001291	0.014277	0.001292	-0.001017
voice.messages	-0.014575	1.000000	0.002463	0.000124	0.002505	0.005381
intl.mins	0.001291	0.002463	1.000000	0.016791	0.999993	-0.019486
intl.calls	0.014277	0.000124	0.016791	1.000000	0.016900	-0.001303
intl.charge	0.001292	0.002505	0.999993	0.016900	1.000000	-0.019415
day.mins	-0.001017	0.005381	-0.019486	-0.001303	-0.019415	1.000000
day.calls	0.028240	0.000883	0.013097	0.010893	0.013161	0.001935
day.charge	-0.000800	0.005140	-0.019295	-0.001599	-0.019225	1.000000
eve.mins	-0.008706	0.018917	-0.000347	0.008001	-0.000324	-0.010931
eve.calls	0.009143	-0.003954	-0.007458	0.005574	-0.007507	0.008128
eve.charge	-0.009587	0.019496	0.000132	0.008393	0.000155	-0.010760
night.mins	0.000668	0.005541	-0.006721	-0.017214	-0.006655	0.011799
night.calls	-0.007825	0.002676	0.000391	-0.000156	0.000368	0.004236
night.charge	0.000656	0.005535	-0.006717	-0.017182	-0.006650	0.011783
customer.calls	-0.001445	-0.007086	-0.012122	-0.019147	-0.012180	0.002733

In [28]:

```
plt.figure(figsize = (15,10))
sns.heatmap(df.corr(), cmap="CMRmap", annot=True)
plt.show()
```



Report

feature customer.calls if negatively correlated with feature voice.messages, intl.min, intl.call, day.call, nights.min, night.charge, night.calls

feature intl.charge and feature intl.min ,feature night.mins and night.charge are highly correlated likewise most of the features are negative correlated

In [29]:

```
Top_10_state = df['state'].value_counts().head(10)
Top_10_state
```

Out[29]:

```
WV    158
MN    125
AL    124
ID    119
VA    118
OH    116
TX    116
WY    115
NY    114
OR    114
Name: state, dtype: int64
```

In [30]:

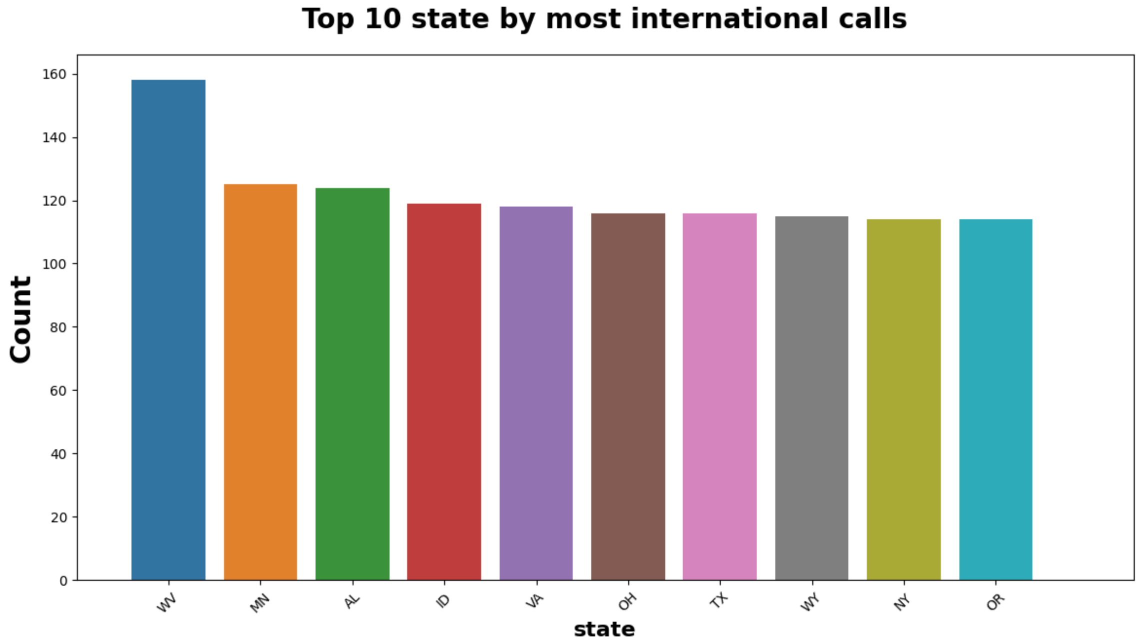
```
Top_10_state_intlcalls =Top_10_state.index
Top_10_state_intlcalls
```

Out[30]:

```
Index(['WV', 'MN', 'AL', 'ID', 'VA', 'OH', 'TX', 'WY', 'NY', 'OR'], dtype
='object')
```

In [31]:

```
plt.subplots(figsize=(14,7))
sns.countplot(x="state", data=df,order = Top_10_state_intlcalls)
plt.title("Top 10 state by most international calls", weight="bold",fontsize=20, pad=20)
plt.ylabel("Count", weight="bold", fontsize=20)
plt.xlabel("state", weight="bold", fontsize=16)
plt.xticks(rotation= 45)
plt.xlim(-1,10.5)
plt.show()
```



Check the mean international calls for state WV which have highest number of international calls

In [32]:

```
WV = df[df['state'] == 'WV']['intl.calls'].mean()
print(f"state WV has average {round(WV)} international calls every month ")
```

```
state WV has average 4 international calls every month
```

Report

- As per the graph state WV has the most no of international calls among all other states
- Following WV we have MN and AL
- Mean International calls from State WV are 4 per month

In [33]:

```
df.columns
```

Out[33]:

```
Index(['state', 'area.code', 'account.length', 'voice.plan', 'voice.messag
es',
       'intl.plan', 'intl.mins', 'intl.calls', 'intl.charge', 'day.mins',
       'day.calls', 'day.charge', 'eve.mins', 'eve.calls', 'eve.charge',
       'night.mins', 'night.calls', 'night.charge', 'customer.calls', 'chu
rn'],
      dtype='object')
```

In [34]:

```
International_details = df.groupby('state')['intl.mins', 'intl.calls', 'intl.charge'].su
International_details
```

Out[34]:

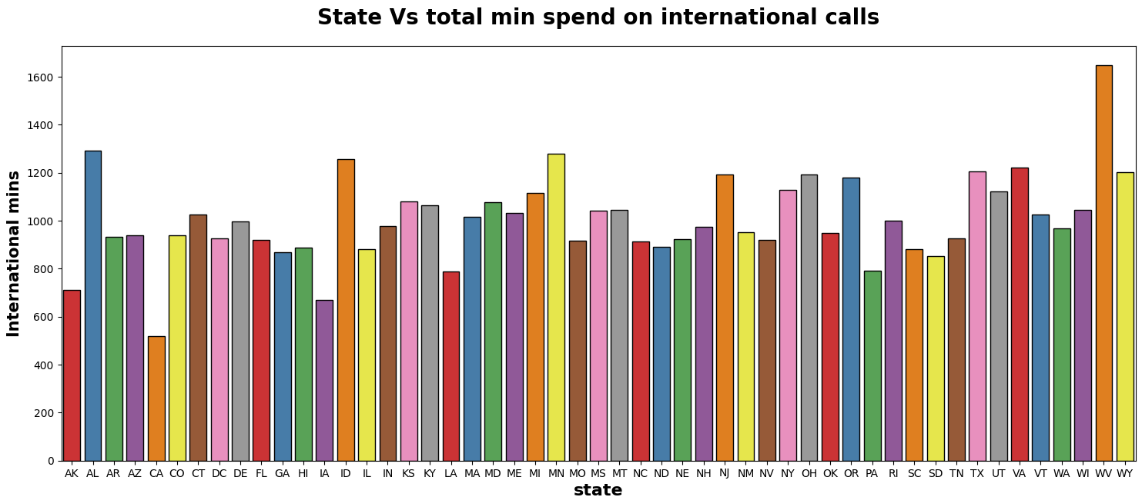
	state	intl.mins	intl.calls	intl.charge
0	AK	711.5	335	192.15
1	AL	1291.2	558	348.65
2	AR	933.9	437	252.16
3	AZ	938.5	423	253.44
4	CA	518.7	226	140.09
5	CO	938.8	416	253.52
6	CT	1027.2	403	277.35
7	DC	925.3	349	249.90
8	DE	997.0	397	269.31
9	FL	920.0	382	248.44
10	GA	869.3	358	234.77
11	HI	889.0	420	240.08
12	IA	670.6	291	181.10
13	ID	1256.6	553	339.35
14	IL	883.0	347	238.46
15	IN	979.0	400	264.36
16	KS	1080.8	437	291.84
17	KY	1063.0	405	287.05
18	LA	789.2	378	213.18
19	MA	1016.5	475	274.53
20	MD	1076.3	452	290.65
21	ME	1032.7	449	278.90
22	MI	1115.7	491	301.25
23	MN	1279.1	526	345.42
24	MO	917.4	465	247.75
25	MS	1041.9	429	281.38
26	MT	1046.1	452	282.48
27	NC	912.5	400	246.42
28	ND	892.8	415	241.11
29	NE	922.9	361	249.25
30	NH	973.5	423	262.91
31	NJ	1192.3	511	322.01
32	NM	951.7	422	257.07
33	NV	920.8	391	248.61
34	NY	1127.2	519	304.34
35	OH	1191.5	494	321.76
36	OK	947.7	419	255.99

	state	intl.mins	intl.calls	intl.charge
37	OR	1180.6	483	318.83
38	PA	791.5	309	213.69
39	RI	999.9	433	270.01
40	SC	881.9	381	238.18
41	SD	851.9	393	230.05
42	TN	927.2	395	250.38
43	TX	1205.0	492	325.38
44	UT	1121.1	515	302.72
45	VA	1220.4	564	329.57
46	VT	1024.6	486	276.69
47	WA	969.0	450	261.71
48	WI	1045.7	431	282.38
49	WV	1647.3	701	444.88
50	WY	1201.6	534	324.48

In [35]:

```
plt.subplots(figsize=(18,7))
sns.barplot(x=International_details.state, y=International_details['intl.mins'],ec = "bl
plt.title("State Vs total min spend on international calls", weight="bold",fontsize=20,
plt.ylabel("International mins", weight="bold", fontsize=15)
plt.xlabel("state", weight="bold", fontsize=16)

plt.show()
```



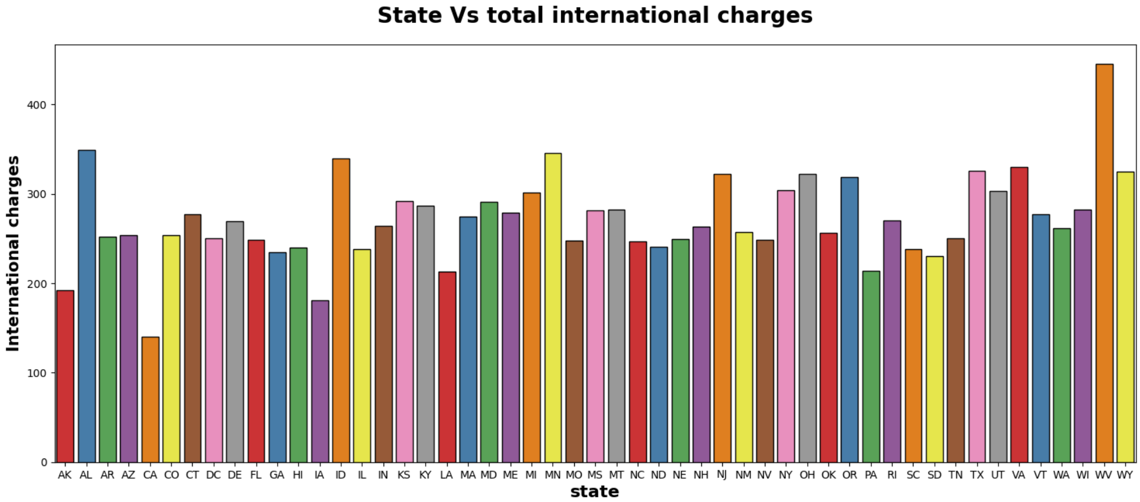
Report

- state WV has total above 1600 mins spend on international calls all over time

In [36]:

```
plt.subplots(figsize=(18,7))
sns.barplot(x=International_details.state, y=International_details['intl.charge'],ec = "
plt.title("State Vs total international charges", weight="bold",fontsize=20, pad=20)
plt.ylabel("International charges", weight="bold", fontsize=15)
plt.xlabel("state", weight="bold", fontsize=16)

plt.show()
```



Report

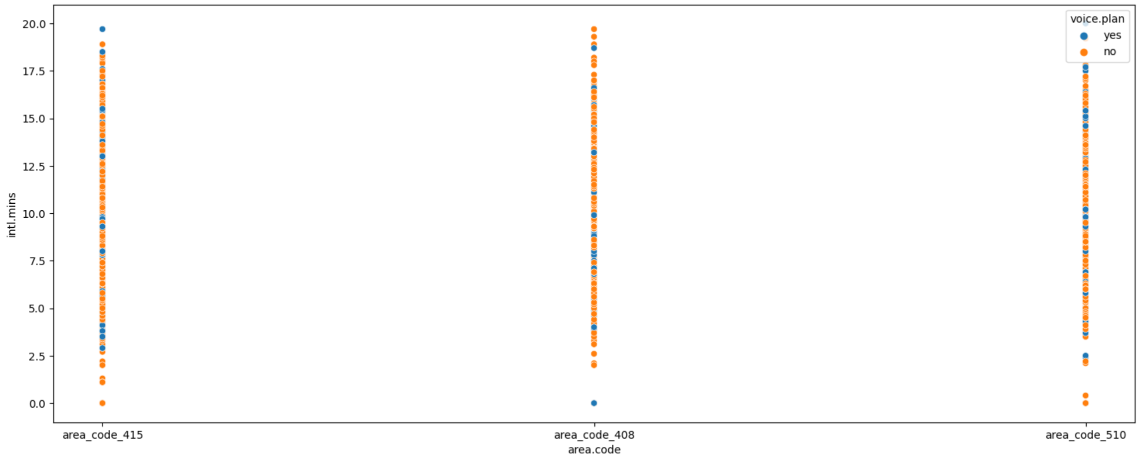
- Its obious that those state which spent more time on international calls have more charge rate which is state 'WV'

In [37]:

```
plt.subplots(figsize=(18,7))
sns.scatterplot(x=df['area.code'], y=df['intl.mins'], hue=df['voice.plan'])
```

Out[37]:

```
<AxesSubplot: xlabel='area.code', ylabel='intl.mins'>
```



Type *Markdown* and LaTeX: α^2

Report

- Higher the totol duration of call during international call greater the international charge

In [39]:

```
df['TotalDaysMins'] = df['day.mins']+df['eve.mins']+df['night.mins']
```

In [40]:

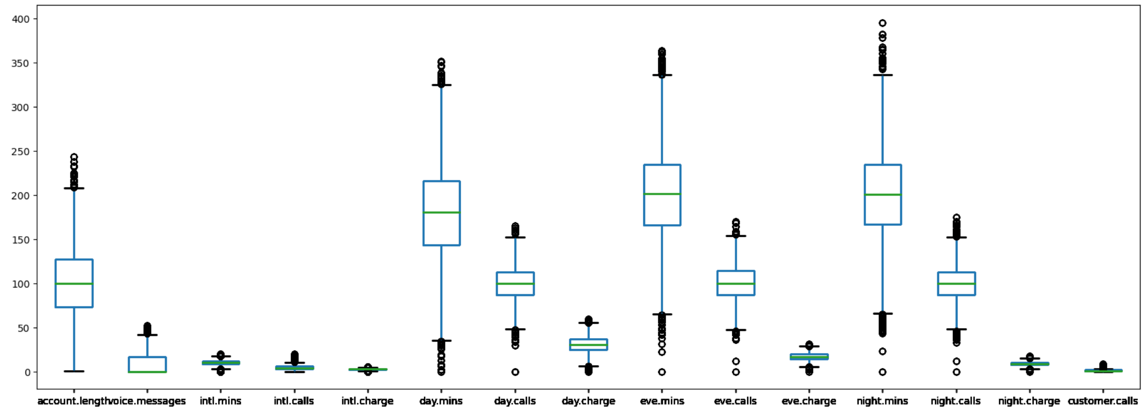
```
df['TotalCharge'] = df['day.charge']+df['eve.charge']+df['night.charge']
```

In [41]:

```
df['TotalDaysCalls'] = df['day.calls']+df['eve.calls']+df['night.calls']
```

In [45]:

```
plt.subplots(figsize=(20,7))
for feature in num_feature:
    df.boxplot(column=num_feature, grid=False)
```



Report

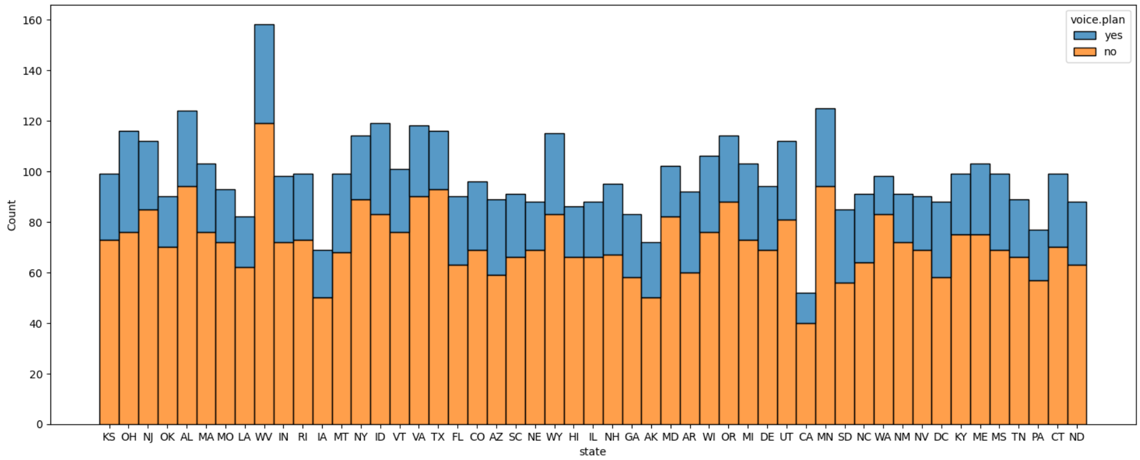
- The no. of 'no' from class churn is greater than no of 'yes', which means loyal customers are higher than churn
- Most of the loyal customer purchased the voice plan
- Rate of Loyal customers if higher than churn customers

In [47]:

```
plt.subplots(figsize=(18,7))
sns.histplot(data=df, x="state", hue="voice.plan", multiple="stack")
```

Out[47]:

```
<AxesSubplot: xlabel='state', ylabel='Count'>
```



Report

- We can say no of customers with voice plan is less compare to customers who have voice plan

Feature Engineering

In [48]:

```
cat_feature
```

Out[48]:

```
['state', 'area.code', 'voice.plan', 'intl.plan', 'churn']
```

In [49]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 23 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   state           5000 non-null   object 
 1   area.code       5000 non-null   object 
 2   account.length  5000 non-null   int64  
 3   voice.plan      5000 non-null   object 
 4   voice.messages  5000 non-null   int64  
 5   intl.plan       5000 non-null   object 
 6   intl.mins       5000 non-null   float64
 7   intl.calls      5000 non-null   int64  
 8   intl.charge     5000 non-null   float64
 9   day.mins        5000 non-null   float64
 10  day.calls       5000 non-null   int64  
 11  day.charge      4993 non-null   float64
 12  eve.mins        4976 non-null   float64
 13  eve.calls       5000 non-null   int64  
 14  eve.charge      5000 non-null   float64
 15  night.mins      5000 non-null   float64
 16  night.calls     5000 non-null   int64  
 17  night.charge    5000 non-null   float64
 18  customer.calls  5000 non-null   int64  
 19  churn           5000 non-null   object 
 20  TotalDaysMins   4976 non-null   float64
 21  TotalCharge     4993 non-null   float64
 22  TotalDaysCalls  5000 non-null   int64  
dtypes: float64(10), int64(8), object(5)
memory usage: 898.6+ KB
```

In [50]:

```
df.isnull().sum()
```

Out[50]:

```
state              0
area.code          0
account.length     0
voice.plan         0
voice.messages     0
intl.plan          0
intl.mins          0
intl.calls         0
intl.charge        0
day.mins           0
day.calls          0
day.charge         7
eve.mins          24
eve.calls          0
eve.charge         0
night.mins         0
night.calls        0
night.charge       0
customer.calls     0
churn              0
TotalDaysMins     24
TotalCharge        7
TotalDaysCalls     0
dtype: int64
```

Filling null values

In [51]:

```
df['TotalDaysMins']=df['TotalDaysMins'].fillna(df['TotalDaysMins'].mean())
```

In [52]:

```
df['TotalCharge']=df['TotalCharge'].fillna(df['TotalCharge'].mean())
```

In [53]:

```
df.isnull().sum()
```

Out[53]:

```
state               0
area.code           0
account.length      0
voice.plan          0
voice.messages      0
intl.plan           0
intl.mins           0
intl.calls          0
intl.charge         0
day.mins            0
day.calls           0
day.charge          7
eve.mins           24
eve.calls           0
eve.charge          0
night.mins          0
night.calls         0
night.charge        0
customer.calls      0
churn               0
TotalDaysMins       0
TotalCharge         0
TotalDaysCalls      0
dtype: int64
```

In [54]:

```
df
```

Out[54]:

	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	i
0	KS	area_code_415	128	yes	25	no	10.0	
1	OH	area_code_415	107	yes	26	no	13.7	
2	NJ	area_code_415	137	no	0	no	12.2	
3	OH	area_code_408	84	no	0	yes	6.6	
4	OK	area_code_415	75	no	0	yes	10.1	
...	...	...	...	...	...	...	...	
4995	HI	area_code_408	50	yes	40	no	9.9	
4996	WV	area_code_415	152	no	0	no	14.7	
4997	DC	area_code_415	61	no	0	no	13.6	
4998	DC	area_code_510	109	no	0	no	8.5	
4999	VT	area_code_415	86	yes	34	no	9.3	

5000 rows × 23 columns

Dropping Already merged columns

In [55]:

```
df.drop(['day.calls'],axis=1,inplace=True)
```

In [56]:

```
df.drop(['eve.calls'],axis=1,inplace=True)
```

In [57]:

```
df.drop(['night.calls'],axis=1,inplace=True)
```

In [58]:

```
df.drop(['day.charge'],axis=1,inplace=True)
```

In [59]:

```
df.drop(['eve.charge'],axis=1,inplace=True)
```

In [60]:

```
df.drop(['night.charge'],axis=1,inplace=True)
```

In [61]:

```
df.drop(['day.mins'],axis=1,inplace=True)
```

In [62]:

```
df.drop(['eve.mins'],axis=1,inplace=True)
```

In [63]:

```
df.drop(['night.mins'],axis=1,inplace=True)
```

Encoding categorical columns

In [64]:

```
from sklearn.preprocessing import LabelEncoder
```

In [65]:

```
le=LabelEncoder()
```

In [66]:

```
for i in cat_feature:
       df[i]=le.fit_transform(df[i])
```

Splitting data into x and y

In [67]:

```
x=df.drop('churn',axis=1)
```

In [68]:

```
x
```

Out[68]:

	state	area.code	account.length	voice.plan	voice.messages	intl.plan	intl.mins	intl.ca
0	16	1	128	1	25	0	10.0	
1	35	1	107	1	26	0	13.7	
2	31	1	137	0	0	0	12.2	
3	35	0	84	0	0	1	6.6	
4	36	1	75	0	0	1	10.1	
...	...	...	...	...	...	...	...	
4995	11	0	50	1	40	0	9.9	
4996	49	1	152	0	0	0	14.7	
4997	7	1	61	0	0	0	13.6	
4998	7	2	109	0	0	0	8.5	
4999	46	1	86	1	34	0	9.3	

5000 rows × 13 columns

In [69]:

```
y=df['churn']
y
```

Out[69]:

```
0       0
1       0
2       0
3       0
4       0
       ..
4995    0
4996    1
4997    0
4998    0
4999    0
Name: churn, Length: 5000, dtype: int32
```

Scaling Data

In [70]:

```
from sklearn.preprocessing import StandardScaler
```

In [71]:

```
sc=StandardScaler()
```

In [72]:

```
x=sc.fit_transform(x)
```

Splitting data into train and test

In [73]:

```
from sklearn.model_selection import train_test_split
```

In [74]:

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.20,random_state=1)
```

As we have imabalanced data we will do oversampling by smote as it will balanced out our data and we'll won't face data loss and there will be no more duplicate values in our data.

In [75]:

```
from imblearn.over_sampling import SMOTE
os = SMOTE(sampling_strategy=1.0)
```

In [76]:

```
xtrain_os, ytrain_os=os.fit_resample(xtrain,ytrain)
```

In [77]:

```
xtrain_os.shape
```

Out[77]:

```
(6862, 13)
```

In [78]:

```
xtest.shape
```

Out[78]:

```
(1000, 13)
```

In [79]:

```
ytrain_os.shape
```

Out[79]:

```
(6862,)
```

In [80]:

```
ytest.shape
```

Out[80]:

```
(1000,)
```

Logistic Regression

In [81]:

```
from sklearn.linear_model import LogisticRegression
```

In [82]:

```
lr=LogisticRegression()
```

In [83]:

```
lr.fit(xtrain_os,ytrain_os)
```

Out[83]:

```
▾ LogisticRegression
LogisticRegression()
```

In [84]:

```
y_pred_train=lr.predict(xtrain_os)
y_pred_test=lr.predict(xtest)
```

In [85]:

```
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
```

Accuracy Score of Logistic Regression

In [86]:

```
print("Train Data")
print(accuracy_score(ytrain_os,y_pred_train))
print("Test Data")
print(accuracy_score(ytest,y_pred_test))
```

```
Train Data
0.7892742640629554
Test Data
0.791
```

Classification Report of Logistic Regression

In [87]:

```
print("Train Data")
print(classification_report(ytrain_os,y_pred_train))
print("Test Data")
print(classification_report(ytest,y_pred_test))
```

```
Train Data
              precision    recall  f1-score   support

           0       0.80      0.77      0.79      3431
           1       0.78      0.81      0.79      3431

    accuracy                           0.79      6862
   macro avg       0.79      0.79      0.79      6862
weighted avg       0.79      0.79      0.79      6862

Test Data
              precision    recall  f1-score   support

           0       0.95      0.80      0.87       862
           1       0.37      0.73      0.49       138

    accuracy                           0.79      1000
   macro avg       0.66      0.77      0.68      1000
weighted avg       0.87      0.79      0.82      1000
```

In [88]:

```
print("Train Data")
print(confusion_matrix(ytrain_os,y_pred_train))
print("Test Data")
print(confusion_matrix(ytest,y_pred_test))
```

```
Train Data
[[2649  782]
 [ 664 2767]]
Test Data
[[690 172]
 [ 37 101]]
```

Decision Tree

In [89]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [90]:

```
dt=DecisionTreeClassifier()
```

In [91]:

```
dt.fit(xtrain_os,ytrain_os)
```

Out[91]:

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [92]:

```
def my_model(clf):
    clf.fit(xtrain_os,ytrain_os)
    y_train_pred=clf.predict(xtrain_os)
    y_test_pred=clf.predict(xtest)
    print('Train Data')
    print(classification_report(ytrain_os,y_train_pred))
    print('Test Data')
    print(classification_report(ytest,y_test_pred))
```

In [93]:

```
my_model(dt)
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3431
           1       1.00      1.00      1.00      3431

    accuracy                           1.00      6862
   macro avg       1.00      1.00      1.00      6862
weighted avg       1.00      1.00      1.00      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      0.94      0.96       862
           1       0.69      0.82      0.75       138

    accuracy                           0.93      1000
   macro avg       0.83      0.88      0.85      1000
weighted avg       0.93      0.93      0.93      1000
```

Hyper parameter tunning for Decision Tree

In [94]:

```
param_grid={'criterion': ['gini', 'entropy'],
            'splitter': ['best', 'random'],
            'max_depth': list(range(3, 51)),
            'min_samples_split': list(range(2, 50)),
            'min_samples_leaf': list(range(1, 50)),
            'max_features': ['auto', 'log2', None]
}
```

In [95]:

```
from sklearn.model_selection import RandomizedSearchCV
```

In [ ]:

```

```

In [96]:

```
clf=RandomizedSearchCV(dt,param_distributions=param_grid,n_iter=10,scoring='f1',n_jobs=-
```

In [97]:

```
clf.fit(xtrain_os,ytrain_os)
```

Out[97]:

```
▸ RandomizedSearchCV
▸ estimator: DecisionTreeClassifier
  ▸ DecisionTreeClassifier
```

In [98]:

```
clf.best_params_
```

Out[98]:

```
{'splitter': 'random',
 'min_samples_split': 3,
 'min_samples_leaf': 1,
 'max_features': None,
 'max_depth': 41,
 'criterion': 'gini'}
```

In [99]:

```
dt1=DecisionTreeClassifier(splitter= 'best',min_samples_split=46,min_samples_leaf=17,cri
```

In [100]:

```
my_model(dt1)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.94      0.97      0.95      3431
           1       0.97      0.94      0.95      3431

    accuracy                           0.95      6862
   macro avg       0.95      0.95      0.95      6862
weighted avg       0.95      0.95      0.95      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      0.96      0.96       862
           1       0.75      0.81      0.78       138

    accuracy                           0.94      1000
   macro avg       0.86      0.88      0.87      1000
weighted avg       0.94      0.94      0.94      1000
```

Random Forest Classifier

In [101]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [102]:

```
rf=RandomForestClassifier()
```

In [103]:

```
my_model(rf)
```

```
Train Data
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      3431
           1       1.00      1.00      1.00      3431

    accuracy                           1.00      6862
   macro avg       1.00      1.00      1.00      6862
weighted avg       1.00      1.00      1.00      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      1.00      0.98       862
           1       0.97      0.80      0.88       138

    accuracy                           0.97      1000
   macro avg       0.97      0.90      0.93      1000
weighted avg       0.97      0.97      0.97      1000
```

Hyper Parameter Tunning for Random Forest Classifier

In [104]:

```
param_grid1={'criterion': ['gini', 'entropy'],
             'max_depth': list(range(3, 51)),
             'min_samples_split': list(range(2, 50)),
             'min_samples_leaf': list(range(1, 50)),
             'max_features': ['auto', 'log2', None]
}
```

In [105]:

```
clf1=RandomizedSearchCV(rf,param_distributions=param_grid1,n_iter=10,scoring='f1',n_jobs
```

In [106]:

```
clf1.fit(xtrain_os,ytrain_os)
```

Out[106]:

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [107]:

```
clf1.best_params_
```

Out[107]:

```
{'min_samples_split': 17,
 'min_samples_leaf': 6,
 'max_features': 'log2',
 'max_depth': 13,
 'criterion': 'entropy'}
```

In [108]:

```
rf1=RandomForestClassifier(min_samples_split=44,min_samples_leaf=1,criterion='entropy',m
```

In [109]:

```
my_model(rf1)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      3431
           1       0.99      0.95      0.97      3431

    accuracy                           0.97      6862
   macro avg       0.97      0.97      0.97      6862
weighted avg       0.97      0.97      0.97      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       862
           1       0.91      0.83      0.87       138

    accuracy                           0.96      1000
   macro avg       0.94      0.91      0.92      1000
weighted avg       0.96      0.96      0.96      1000
```

AdaBoost Classifier

In [110]:

```
from sklearn.ensemble import AdaBoostClassifier
```

In [111]:

```
ada=AdaBoostClassifier()
```

In [112]:

```
my_model(ada)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.90      0.93      0.91      3431
           1       0.92      0.90      0.91      3431

    accuracy                           0.91      6862
   macro avg       0.91      0.91      0.91      6862
weighted avg       0.91      0.91      0.91      6862

Test Data
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       862
           1       0.60      0.69      0.64       138

    accuracy                           0.89      1000
   macro avg       0.78      0.81      0.79      1000
weighted avg       0.90      0.89      0.90      1000
```

Hyper parameter tunning for AdaBoost classifier

In [113]:

```
param_grid2={'n_estimators':[20,50,70,100,120],
            'learning_rate':(0.1,0.01,0.001,1,),
             }
```

In [114]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
ran_clf=RandomizedSearchCV(ada,param_distributions=param_grid2,cv=10,n_jobs=-1)
ran_clf.fit(xtrain_os,ytrain_os)
```

Out[114]:

```
▸      RandomizedSearchCV
▸ estimator: AdaBoostClassifier
     ▸ AdaBoostClassifier
```

In [115]:

```
ran_clf.best_params_
```

Out[115]:

```
{'n_estimators': 120, 'learning_rate': 1}
```

In [116]:

```
ada1=AdaBoostClassifier(n_estimators= 50,learning_rate= 1)
```

In [117]:

```
my_model(ada1)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.90      0.93      0.91      3431
           1       0.92      0.90      0.91      3431

    accuracy                           0.91      6862
   macro avg       0.91      0.91      0.91      6862
weighted avg       0.91      0.91      0.91      6862

Test Data
              precision    recall  f1-score   support

           0       0.95      0.93      0.94       862
           1       0.60      0.69      0.64       138

    accuracy                           0.89      1000
   macro avg       0.78      0.81      0.79      1000
weighted avg       0.90      0.89      0.90      1000
```

Support Vector Machine Classifier

In [118]:

```
from sklearn.svm import SVC
```

In [119]:

```
svc=SVC()
```

In [120]:

```
my_model(svc)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      3431
           1       0.94      0.90      0.92      3431

    accuracy                           0.93      6862
   macro avg       0.93      0.93      0.93      6862
weighted avg       0.93      0.93      0.93      6862

Test Data
              precision    recall  f1-score   support

           0       0.96      0.95      0.96       862
           1       0.71      0.78      0.74       138

    accuracy                           0.93      1000
   macro avg       0.84      0.86      0.85      1000
weighted avg       0.93      0.93      0.93      1000
```

Gradient Boosting Classifier

In [121]:

```
from sklearn.ensemble import GradientBoostingClassifier
```

In [122]:

```
gb=GradientBoostingClassifier()
```

In [123]:

```
my_model(gb)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.94      0.99      0.97      3431
           1       0.99      0.94      0.97      3431

    accuracy                           0.97      6862
   macro avg       0.97      0.97      0.97      6862
weighted avg       0.97      0.97      0.97      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       862
           1       0.95      0.82      0.88       138

    accuracy                           0.97      1000
   macro avg       0.96      0.91      0.93      1000
weighted avg       0.97      0.97      0.97      1000
```

Hyper Parameter tunning for Gradient Boosting

In [124]:

```
gb_grid={'n_estimators':[50,100,120,150],
            'learning_rate':(0.1,0.01,0.001)
           }
```

In [125]:

```
GB_clf1=RandomizedSearchCV(gb,param_distributions=gb_grid,cv=10,n_jobs=-1)
GB_clf1.fit(xtrain_os,ytrain_os)
```

Out[125]:

```
▸          RandomizedSearchCV
▸ estimator: GradientBoostingClassifier
     ▸ GradientBoostingClassifier
```

In [126]:

```
GB_clf1.best_params_
```

Out[126]:

```
{'n_estimators': 120, 'learning_rate': 0.1}
```

In [127]:

```
gb1=GradientBoostingClassifier(n_estimators= 150,learning_rate=0.1)
```

In [128]:

```
my_model(gb1)
```

```
Train Data
              precision    recall  f1-score   support

           0       0.96      1.00      0.98      3431
           1       1.00      0.96      0.98      3431

    accuracy                           0.98      6862
   macro avg       0.98      0.98      0.98      6862
weighted avg       0.98      0.98      0.98      6862

Test Data
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       862
           1       0.96      0.81      0.88       138

    accuracy                           0.97      1000
   macro avg       0.96      0.90      0.93      1000
weighted avg       0.97      0.97      0.97      1000
```

Cross Validation

1) K-Fold Cross Validation

In [129]:

```
from sklearn.model_selection import KFold
model=GradientBoostingClassifier(random_state=42)
kfold_validation=KFold(10)
import numpy as np
from sklearn.model_selection import cross_val_score
results=cross_val_score(model,x,y,cv=kfold_validation)
print(results)
print(round(np.mean(results),2))
```

```
[0.97   0.97   0.968 0.97   0.972 0.956 0.986 0.96   0.97   0.986]
0.97
```

2) Stratified K-Fold Cross Validation

In [130]:

```
from sklearn.model_selection import StratifiedKFold
skfold=StratifiedKFold(n_splits=5)
model=GradientBoostingClassifier(random_state=42)
scores=cross_val_score(model,x,y,cv=skfold)
print(round(np.mean(scores),2))
```

```
0.97
```

Model Deployment

In [131]:

```
final_df=df[['voice.plan','voice.messages','intl.plan','intl.mins','intl.calls','intl.ch
final_df
```

Out[131]:

	voice.plan	voice.messages	intl.plan	intl.mins	intl.calls	intl.charge	customer.calls	To
0	1	25	0	10.0	3	2.70	1	
1	1	26	0	13.7	3	3.70	1	
2	0	0	0	12.2	5	3.29	0	
3	0	0	1	6.6	7	1.78	2	
4	0	0	1	10.1	3	2.73	3	
...	...	...	...	...	...	...	...	
4995	1	40	0	9.9	5	2.67	2	
4996	0	0	0	14.7	2	3.97	3	
4997	0	0	0	13.6	4	3.67	1	
4998	0	0	0	8.5	6	2.30	0	
4999	1	34	0	9.3	16	2.51	0	

5000 rows × 11 columns

In [132]:

```
final_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   voice.plan       5000 non-null   int32  
 1   voice.messages   5000 non-null   int64  
 2   intl.plan        5000 non-null   int32  
 3   intl.mins        5000 non-null   float64
 4   intl.calls       5000 non-null   int64  
 5   intl.charge      5000 non-null   float64
 6   customer.calls   5000 non-null   int64  
 7   TotalDaysMins    5000 non-null   float64
 8   TotalCharge      5000 non-null   float64
 9   TotalDaysCalls   5000 non-null   int64  
 10  churn            5000 non-null   int32  
dtypes: float64(4), int32(3), int64(4)
memory usage: 371.2 KB
```

In [133]:

```
final_df.isnull().sum()
```

Out[133]:

```
voice.plan        0
voice.messages    0
intl.plan         0
intl.mins         0
intl.calls        0
intl.charge       0
customer.calls    0
TotalDaysMins     0
TotalCharge       0
TotalDaysCalls    0
churn             0
dtype: int64
```

In [134]:

```
x=final_df.drop(['churn'],axis=1)
y=final_df['churn']
```

In [149]:

```
from sklearn.model_selection import train_test_split
```

In [150]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.20,random_state=123)
```

In [151]:

```
from imblearn.over_sampling import SMOTE
os = SMOTE(sampling_strategy=1.0)
```

In [152]:

```
x_train_os, y_train_os=os.fit_resample(x_train,y_train)
```

In [153]:

```
x_train_os.shape,y_train_os.shape
```

Out[153]:

```
((6894, 10), (6894,))
```

In [154]:

```
x_test.shape,y_test.shape
```

Out[154]:

```
((1000, 10), (1000,))
```

In [155]:

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn import metrics
```

In [159]:

```
pipe=Pipeline(steps=[('step1',LogisticRegression(solver='liblinear'))])
pipe.fit(x_train_os,y_train_os)
```

Out[159]:

```
▸ Pipeline
▸ LogisticRegression
```

In [160]:

```
y_pred=pipe.predict(x_test)
```

In [161]:

```
pipe.predict(x_test)[10]
```

Out[161]:

```
1
```

In [162]:

```
pipe1=Pipeline(steps=[('step1',GradientBoostingClassifier())])
pipe1.fit(x_train_os,y_train_os)
```

Out[162]:

```
Pipeline
GradientBoostingClassifier
```

In [163]:

```
pipe1.predict(x_test)[10]
```

Out[163]:

```
0
```

In [164]:

```
import pickle
pickle.dump(pipe,open('pipe.pkl','wb'))
```

In [165]:

```
final_df.sample(10)
```

Out[165]:

	voice.plan	voice.messages	intl.plan	intl.mins	intl.calls	intl.charge	customer.calls	To
4139	1	31	0	12.5	3	3.38	2	
873	0	0	0	9.3	1	2.51	1	
1048	0	0	0	7.7	3	2.08	0	
3309	0	0	1	12.0	4	3.24	4	
695	0	0	0	11.6	9	3.13	1	
4536	0	0	0	14.6	4	3.94	2	
2631	1	22	0	12.4	2	3.35	2	
3394	1	22	0	11.4	7	3.08	2	
2722	0	0	0	10.1	5	2.73	2	
2155	0	0	0	12.0	5	3.24	2	

In [ ]: