- Student name: Michael Holthouser
- Student pace: flex
- Scheduled project review date/time:
- Instructor name: Abhineet Kulkarni
- Blog post URL:

Introduction

SyriaTel has tasked me to provide prediction analysis on whether their customers will churn soon. To churn in its broadest sense according to wikipedia is, "A measure of the number of individuals or items moving out of a collective group over a specific period."

Stakeholder:

- SyriaTel Communications

Data:

The title of this dataset is called "Churn in Telecom's dataset" from [kaggle.com (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset)](https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset)

- Number of records: 3333
- Number of columns: 20
- Target variable: **churn**

Models:

- Baseline model: `Logistic regression`
- Model 2: `Decision tree`
- Model 3: `Random forest`

Evaluation Metric:

I have elected to use **Recall** as my evaluation metric for this particular project. The recall score is true positive divided by the true positive plus the false negative. It is the measure of actual observations which are predicted correctly. I chose this metric because we want to capture as many positives as possible, and is the best metric to use when we have imbalanced data.

Import Libararies

Firstly, we must import the necessary library packages for this project.

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

from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split,  GridSearchCV
from sklearn.preprocessing import OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix, classification_repor
from imblearn.over_sampling import RandomOverSampler


from sklearn.metrics import precision_score, recall_score, accuracy_sco


from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
```

Column Descriptions

- **`state`** , string. 2-letter code of the US state of customer residence
- **`account_length`** , numerical. Number of months the customer has been with the current telco provider
- **`area_code`** , string="area_code_AAA" where AAA = 3 digit area code.
- **`international_plan`** , (yes/no). The customer has international plan.
- **`voice_mail_plan`** , (yes/no). The customer has voice mail plan.
- **`number_vmail_messages`** , numerical. Number of voice-mail messages.
- **`total_day_minutes`** , numerical. Total minutes of day calls.
- **`total_day_calls`** , numerical. Total minutes of day calls.
- **`total_day_charge`** , numerical. Total charge of day calls.
- **`total_eve_minutes`** , numerical. Total minutes of evening calls.
- **`total_eve_calls`** , numerical. Total number of evening calls.
- **`total_eve_charge`** , numerical. Total charge of evening calls.
- **`total_night_minutes`** , numerical. Total minutes of night calls.
- **`total_night_calls`** , numerical. Total number of night calls.
- **`total_night_charge`** , numerical. Total charge of night calls.
- **`total_intl_minutes`** , numerical. Total minutes of international calls.
- **`total_intl_calls`** , numerical. Total number of international calls.

- **`total_intl_charge`** , numerical. Total charge of international calls
- **`number_customer_service_calls`** , numerical. Number of calls to customer service
- **`churn`** , (yes/no). Customer churn - target variable.

Loading The Data

The next step is to extract the data and put in a pandas dataframe, and to print the first 5 rows to see if the data was imported correctly.

In [2]:

```
data = pd.read_csv('churn_telecom.csv')
data.head()
```

Out[2]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122

5 rows × 21 columns

Exploring and Cleaning the data

Below I am exploring the data, and checking what the data types of each columns with the .info() function, the descriptive statistics of the data with the .describe() function, and finally I will check for any missing data using the .isna() function.

In [3]:
```
# inspect how many records there are, and the data types for each colum
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

At first glance, it appears for the most part the columns that are numerical are the correct data type. However, there are a couple columns that are the object data type that need to be changed to something numerical. The "international plan" and the "voice mail plan" have entries in the form of "yes" or "no" need to be changed to 1 for "yes" and 0 for "no". Likewise the "churn" column needs to be changed into a different data type and will the have entries changed to numerical entries such as 1 for "True" and 0 for "False".

In [4]:

```
# Descriptive statistics on each column of the dataset
data.describe()
```

Out[4]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000

The data looks normal without and large outliers glooming.

In [5]:

```
# Inspect the dataset to see if there is any missing data
data.isna().sum()
```

Out[5]:

```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

Great! there is no missing values to take care of. However, I will next add an underscore "_" as that is conventional in the python language.

In [6]:
```
data.columns = data.columns.map(lambda col: col.replace(' ','_'))
print(data.columns)
```

```
Index(['state', 'account_length', 'area_code', 'phone_number',
       'international_plan', 'voice_mail_plan', 'number_vmail_messages',
       'total_day_minutes', 'total_day_calls', 'total_day_charge',
       'total_eve_minutes', 'total_eve_calls', 'total_eve_charge',
       'total_night_minutes', 'total_night_calls', 'total_night_charge',
       'total_intl_minutes', 'total_intl_calls', 'total_intl_charge',
       'customer_service_calls', 'churn'],
      dtype='object')
```

Next I shall check the unique values of the dataset to see if there are any place holders for missing values.

In [7]:

```python
# inspect unique values of columns to identify potention errors or null
for col in data.columns:
    print(f"{col} vals:  {data[col].unique()} \n")
```

```
state vals:  ['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 vals:  [128 107 137  84  75 118 121 147 117 141  65  74 168  95  62 161  85  93
  76  73  77 130 111 132 174  57  54  20  49 142 172  12  72  36  78 136
 149  98 135  34 160  64  59 119  97  52  60  10  96  87  81  68 125 116
  38  40  43 113 126 150 138 162  90  50  82 144  46  70  55 106  94 155
  80 104  99 120 108 122 157 103  63 112  41 193  61  92 131 163  91 127
 110 140  83 145  56 151 139   6 115 146 185 148  32  25 179  67  19 170
 164  51 208  53 105  66  86  35  88 123  45 100 215  22  33 114  24 101
 143  48  71 167  89 199 166 158 196 209  16  39 173 129  44  79  31 124
  37 159 194 154  21 133 224  58  11 109 102 165  18  30 176  47 190 152
  26  69 186 171  28 153 169  13  27   3  42 189 156 134 243  23   1 205
 200   5   9 178 181 182 217 177 210  29 180   2  17   7 212 232 192 195
 197 225 184 191 201  15 183 202   8 175   4 188 204 221]

area_code vals:  [415 408 510]

phone_number vals:  ['382-4657' '371-7191' '358-1921' ... '328-8230' '364-6381' '400-4344']

international_plan vals:  ['no' 'yes']

voice_mail_plan vals:  ['yes' 'no']

number_vmail_messages vals:  [25 26  0 24 37 27 33 39 30 41 28 34 46 29 35 21 32 42 36 22 23 43 31 38
 40 48 18 17 45 16 20 14 19 51 15 11 12 47  8 44 49  4 10 13 50  9]

total_day_minutes vals:  [265.1 161.6 243.4 ... 321.1 231.1 180.8]

total_day_calls vals:  [110 123 114  71 113  98  88  79  97  84 137 127  96  70  67 139  66  90
 117  89 112 103  86  76 115  73 109  95 105 121 118  94  80 128  64 106
 102  85  82  77 120 133 135 108  57  83 129  91  92  74  93 101 146  72
  99 104 125  61 100  87 131  65 124 119  52  68 107  47 116 151 126 122
 111 145  78 136 140 148  81  55  69 158 134 130  63  53  75 141 163  59
 132 138  54  58  62 144 143 147  36  40 150  56  51 165  30  48  60  42
   0  45 160 149 152 142 156  35  49 157  44]

total_day_charge vals:  [45.07 27.47 41.38 ... 54.59 39.29 30.74]

total_eve_minutes vals:  [197.4 195.5 121.2 ... 153.4 288.8 265.9]

total_eve_calls vals:  [ 99 103 110  88 122 101 108  94  80 111  83 148  71  75  76  97  90  65
  93 121 102  72 112 100  84 109  63 107 115 119 116  92  85  98 118  74
 117  58  96  66  67  62  77 164 126 142  64 104  79  95  86 105  81 113
 106  59  48  82  87 123 114 140 128  60  78 125  91  46 138 129  89 133
```

```
 136  57 135 139  51  70 151 137 134  73 152 168  68 120  69 127 132 143
  61 124  42  54 131  52 149  56  37 130  49 146 147  55  12  50 157 155
  45 144  36 156  53 141  44 153 154 150  43   0 145 159 170]

total_eve_charge vals:  [16.78 16.62 10.3  ... 13.04 24.55 22.6 ]

total_night_minutes vals:  [244.7 254.4 162.6 ... 280.9 120.1 279.1]

total_night_calls vals:  [ 91 103 104  89 121 118  96  90  97 111  94 128
115  99  75 108  74 133
  64  78 105  68 102 148  98 116  71 109 107 135  92  86 127  79  87 129
  57  77  95  54 106  53  67 139  60 100  61  73 113  76 119  88  84  62
 137  72 142 114 126 122  81 123 117  82  80 120 130 134  59 112 132 110
 101 150  69 131  83  93 124 136 125  66 143  58  55  85  56  70  46  42
 152  44 145  50 153  49 175  63 138 154 140 141 146  65  51 151 158 155
 157 147 144 149 166  52  33 156  38  36  48 164]

total_night_charge vals:  [11.01 11.45  7.32  8.86  8.41  9.18  9.57  9.5
3  9.71 14.69  9.4   8.82
  6.35  8.65  9.14  7.23  4.02  5.83  7.46  8.68  9.43  8.18  8.53 10.67
 11.28  8.22  4.59  8.17  8.04 11.27 11.08 13.2  12.61  9.61  6.88  5.82
 10.25  4.58  8.47  8.45  5.5  14.02  8.03 11.94  7.34  6.06 10.9   6.44
  3.18 10.66 11.21 12.73 10.28 12.16  6.34  8.15  5.84  8.52  7.5   7.48
  6.21 11.95  7.15  9.63  7.1   6.91  6.69 13.29 11.46  7.76  6.86  8.16
 12.15  7.79  7.99 10.29 10.08 12.53  7.91 10.02  8.61 14.54  8.21  9.09
  4.93 11.39 11.88  5.75  7.83  8.59  7.52 12.38  7.21  5.81  8.1  11.04
 11.19  8.55  8.42  9.76  9.87 10.86  5.36 10.03 11.15  9.51  6.22  2.59
  7.65  6.45  9.    6.4   9.94  5.08 10.23 11.36  6.97 10.16  7.88 11.91
  6.61 11.55 11.76  9.27  9.29 11.12 10.69  8.8  11.85  7.14  8.71 11.42
  4.94  9.02 11.22  4.97  9.15  5.45  7.27 12.91  7.75 13.46  6.32 12.13
 11.97  6.93 11.66  7.42  6.19 11.41 10.33 10.65 11.92  4.77  4.38  7.41
 12.1   7.69  8.78  9.36  9.05 12.7   6.16  6.05 10.85  8.93  3.48 10.4
  5.05 10.71  9.37  6.75  8.12 11.77 11.49 11.06 11.25 11.03 10.82  8.91
  8.57  8.09 10.05 11.7  10.17  8.74  5.51 11.11  3.29 10.13  6.8   8.49
  9.55 11.02  9.91  7.84 10.62  9.97  3.44  7.35  9.79  8.89  8.14  6.94
 10.49 10.57 10.2   6.29  8.79 10.04 12.41 15.97  9.1  11.78 12.75 11.07
 12.56  8.63  8.02 10.42  8.7   9.98  7.62  8.33  6.59 13.12 10.46  6.63
  8.32  9.04  9.28 10.76  9.64 11.44  6.48 10.81 12.66 11.34  8.75 13.05
 11.48 14.04 13.47  5.63  6.6   9.72 11.68  6.41  9.32 12.95 13.37  9.62
  6.03  8.25  8.26 11.96  9.9   9.23  5.58  7.22  6.64 12.29 12.93 11.32
  6.85  8.88  7.03  8.48  3.59  5.86  6.23  7.61  7.66 13.63  7.9  11.82
  7.47  6.08  8.4   5.74 10.94 10.35 10.68  4.34  8.73  5.14  8.24  9.99
 13.93  8.64 11.43  5.79  9.2  10.14 12.11  7.53 12.46  8.46  8.95  9.84
 10.8  11.23 10.15  9.21 14.46  6.67 12.83  9.66  9.59 10.48  8.36  4.84
 10.54  8.39  7.43  9.06  8.94 11.13  8.87  8.5   7.6  10.73  9.56 10.77
  7.73  3.47 11.86  8.11  9.78  9.42  9.65  7.    7.39  9.88  6.56  5.92
  6.95 15.71  8.06  4.86  7.8   8.58 10.06  5.21  6.92  6.15 13.49  9.38
 12.62 12.26  8.19 11.65 11.62 10.83  7.92  7.33 13.01 13.26 12.22 11.58
  5.97 10.99  8.38  9.17  8.08  5.71  3.41 12.63 11.79 12.96  7.64  6.58
 10.84 10.22  6.52  5.55  7.63  5.11  5.89 10.78  3.05 11.89  8.97 10.44
 10.5   9.35  5.66 11.09  9.83  5.44 10.11  6.39 11.93  8.62 12.06  6.02
  8.85  5.25  8.66  6.73 10.21 11.59 13.87  7.77 10.39  5.54  6.62 13.33
  6.24 12.59  6.3   6.79  8.28  9.03  8.07  5.52 12.14 10.59  7.54  7.67
  5.47  8.81  8.51 13.45  8.77  6.43 12.01 12.08  7.07  6.51  6.84  9.48
 13.78 11.54 11.67  8.13 10.79  7.13  4.72  4.64  8.96 13.03  6.07  3.51
  6.83  6.12  9.31  9.58  4.68  5.32  9.26 11.52  9.11 10.55 11.47  9.3
 13.82  8.44  5.77 10.96 11.74  8.9  10.47  7.85 10.92  4.74  9.74 10.43
```

```
  9.96 10.18  9.54  7.89 12.36  8.54 10.07  9.46  7.3  11.16  9.16 10.19
  5.99 10.88  5.8   7.19  4.55  8.31  8.01 14.43  8.3  14.3   6.53  8.2
 11.31 13.    6.42  4.24  7.44  7.51 13.1   9.49  6.14  8.76  6.65 10.56
  6.72  8.29 12.09  5.39  2.96  7.59  7.24  4.28  9.7   8.83 13.3  11.37
  9.33  5.01  3.26 11.71  8.43  9.68 15.56  9.8   3.61  6.96 11.61 12.81
 10.87 13.84  5.03  5.17  2.03 10.34  9.34  7.95 10.09  9.95  7.11  9.22
  6.13 11.05  9.89  9.39 14.06 10.26 13.31 15.43 16.39  6.27 10.64 11.5
 12.48  8.27 13.53 10.36 12.24  8.69 10.52  9.07 11.51  9.25  8.72  6.78
  8.6  11.84  5.78  5.85 12.3   5.76 12.07  9.6   8.84 12.39 10.1   9.73
  2.85  6.66  2.45  5.28 11.73 10.75  7.74  6.76  6.    7.58 13.69  7.93
  7.68  9.75  4.96  5.49 11.83  7.18  9.19  7.7   7.25 10.74  4.27 13.8
  9.12  4.75  7.78 11.63  7.55  2.25  9.45  9.86  7.71  4.95  7.4  11.17
 11.33  6.82 13.7   1.97 10.89 12.77 10.31  5.23  5.27  9.41  6.09 10.61
  7.29  4.23  7.57  3.67 12.69 14.5   5.95  7.87  5.96  5.94 12.23  4.9
 12.33  6.89  9.67 12.68 12.87  3.7   6.04 13.13 15.74 11.87  4.7   4.67
  7.05  5.42  4.09  5.73  9.47  8.05  6.87  3.71 15.86  7.49 11.69  6.46
 10.45 12.9   5.41 11.26  1.04  6.49  6.37 12.21  6.77 12.65  7.86  9.44
  4.3   7.38  5.02 10.63  2.86 17.19  8.67  8.37  6.9  10.93 10.38  7.36
 10.27 10.95  6.11  4.45 11.9  15.01 12.84  7.45  6.98 11.72  7.56 11.38
 10.    4.42  9.81  5.56  6.01 10.12 12.4  16.99  5.68 11.64  3.78  7.82
  9.85 13.74 12.71 10.98 10.01  9.52  7.31  8.35 11.35  9.5  14.03  3.2
  7.72 13.22 10.7   8.99 10.6  13.02  9.77 12.58 12.35 12.2  11.4  13.91
  3.57 14.65 12.28  5.13 10.72 12.86 14.    7.12 12.17  4.71  6.28  8.
  7.01  5.91  5.2  12.   12.02 12.88  7.28  5.4  12.04  5.24 10.3  10.41
 13.41 12.72  9.08  7.08 13.5   5.35 12.45  5.3  10.32  5.15 12.67  5.22
  5.57  3.94  4.41 13.27 10.24  4.25 12.89  5.72 12.5  11.29  3.25 11.53
  9.82  7.26  4.1  10.37  4.98  6.74 12.52 14.56  8.34  3.82  3.86 13.97
 11.57  6.5  13.58 14.32 13.75 11.14 14.18  9.13  4.46  4.83  9.69 14.13
  7.16  7.98 13.66 14.78 11.2   9.93 11.    5.29  9.92  4.29 11.1  10.51
 12.49  4.04 12.94  7.09  6.71  7.94  5.31  5.98  7.2  14.82 13.21 12.32
 10.58  4.92  6.2   4.47 11.98  6.18  7.81  4.54  5.37  7.17  5.33 14.1
  5.7  12.18  8.98  5.1  14.67 13.95 16.55 11.18  4.44  4.73  2.55  6.31
  2.43  9.24  7.37 13.42 12.42 11.8  14.45  2.89 13.23 12.6  13.18 12.19
 14.81  6.55 11.3  12.27 13.98  8.23 15.49  6.47 13.48 13.59 13.25 17.77
 13.9   3.97 11.56 14.08 13.6   6.26  4.61 12.76 15.76  6.38  3.6  12.8
  5.9   7.97  5.   10.97  5.88 12.34 12.03 14.97 15.06 12.85  6.54 11.24
 12.64  7.06  5.38 13.14  3.99  3.32  4.51  4.12  3.93  2.4  11.75  4.03
 15.85  6.81 14.25 14.09 16.42  6.7  12.74  2.76 12.12  6.99  6.68 11.81
  7.96  5.06 13.16  2.13 13.17  5.12  5.65 12.37 10.53]

total_intl_minutes vals:  [10.  13.7 12.2  6.6 10.1  6.3  7.5  7.1  8.7 1
1.2 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 vals:  [ 3  5  7  6  4  2  9 19  1 10 15  8 11  0 12 13
18 14 16 20 17]
```

```
total_intl_charge vals:  [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 vals:  [1 0 2 3 4 5 7 9 6 8]

churn vals:  [False  True]
```

Some things I noticed:

- **States** look good.
- There are only 3 area codes
- "**international_plan**", "**voice_mail_plan**" have yes/no values and will need to be changed to a 1 and 0.
- Phone number can probably be dropped from the dataset because, it shouldn't be a reason why customer is choosing to churn.
- The target variable "**churn**" has a boolean value, and needs to be changed to a 1 and 0.

Map columns: international plan, voice mail plan, and churn

In [8]:
```
data['international_plan'] = data['international_plan'].map({'no': 0, '
data['voice_mail_plan'] = data['voice_mail_plan'].map({'no': 0, 'yes':
data['churn'] = data['churn'].map({False: 0, True: 1})
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account_length          3333 non-null   int64  
 2   area_code               3333 non-null   int64  
 3   phone_number            3333 non-null   object 
 4   international_plan      3333 non-null   int64  
 5   voice_mail_plan         3333 non-null   int64  
 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   int64  
dtypes: float64(8), int64(11), object(2)
memory usage: 546.9+ KB
```

Now that we have dealt with the object data types, we can delete the phone number column, since it will not effect our models.

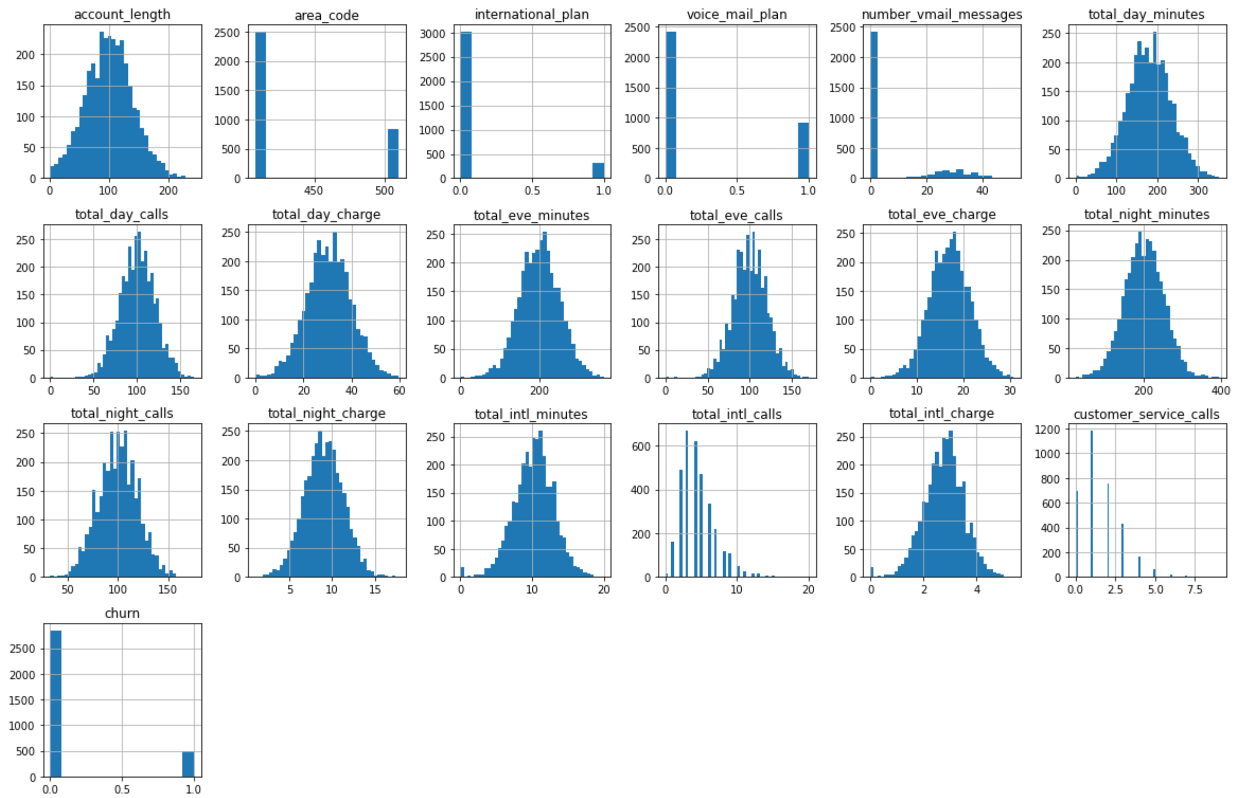
Drop phone number column

In [9]:
```
# Drop the phone_number column from the dataset
data.drop("phone_number", axis=1, inplace=True)
```

Check the distribution of the data

In [10]:

```
data.hist(bins = 'auto', layout = (6,6), figsize = (20,20))
plt.show()
```



It looks like our data is normally distributed.

Exploration and Visualization of Churn Data

Next I am curious and want visualize what the number of customers have churned and their percentages.

In [11]:
```
print("Churn Counts")
print(data["churn"].value_counts())
print()
print("Percentages")
print(data["churn"].value_counts(normalize=True))
```

```
Churn Counts
0    2850
1     483
Name: churn, dtype: int64

Percentages
0    0.855086
1    0.144914
Name: churn, dtype: float64
```

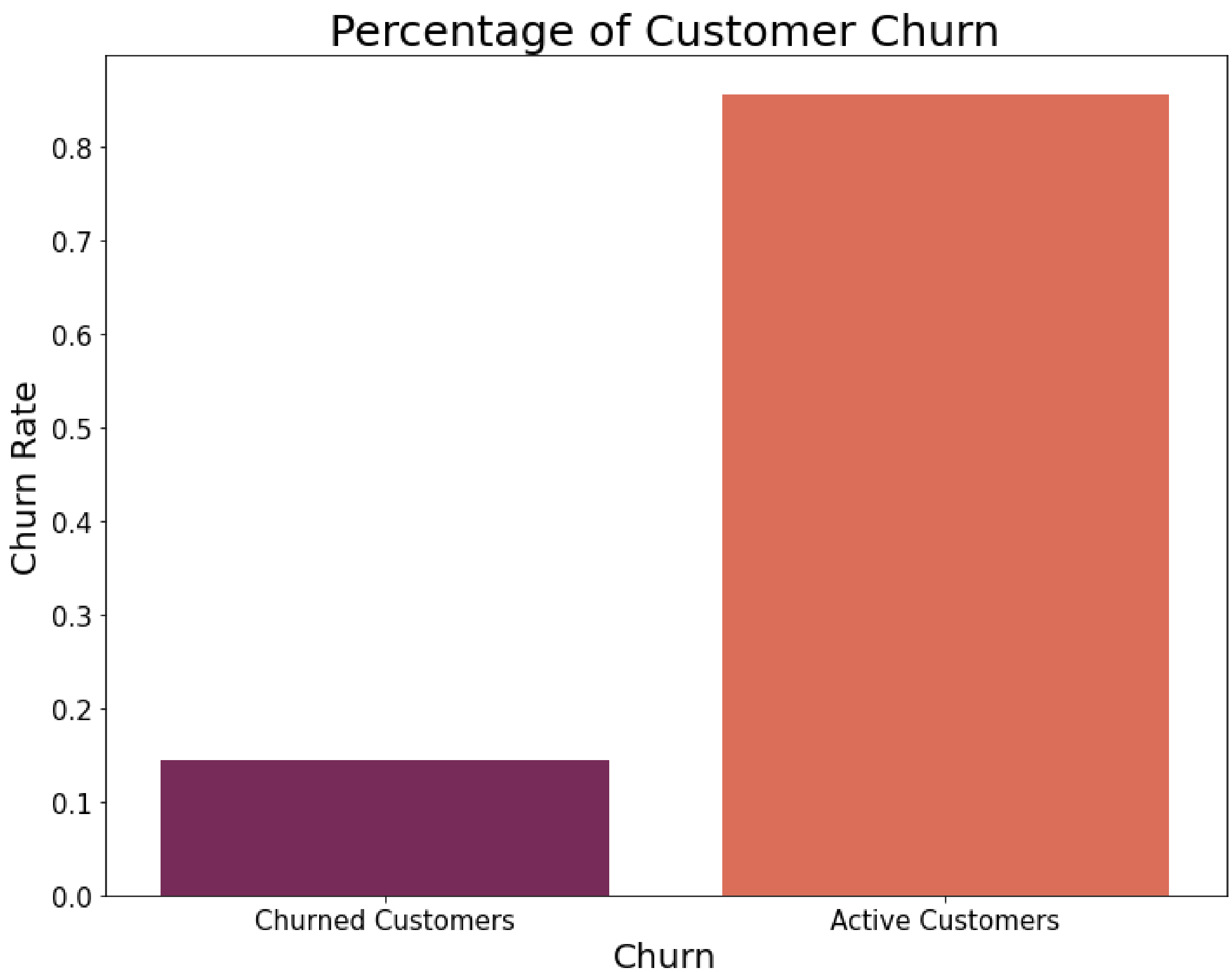
Of the 3333 customers from the data set **14.5%** have terminated their service with SyriaTel. I am curious if a certain area code has a greater number of churns over the other.

Visualization of percentage of customers that have churned

In [12]:

```
# Percentages of current customers vs customer churn
churn_per = data["churn"].value_counts(normalize=True)

# Plot of percentages
fig, ax = plt.subplots(figsize = (10, 8))
sns.barplot(x = [0, 1], y = data["churn"], palette="rocket", data = chu
plt.title('Percentage of Customer Churn', fontsize = 25)
ax.tick_params(axis='both', labelsize=15)
plt.xlabel('Churn', fontsize=20)
plt.ylabel('Churn Rate', fontsize=20)
ax.set_xticklabels(['Churned Customers', 'Active Customers'])
plt.tight_layout()
```



This is a visualization to show the churn rate of the SyriaTel customers. As shown above, the churn rate for this particular dataset was **14.5%**

Visualization of number of customers churned per area code

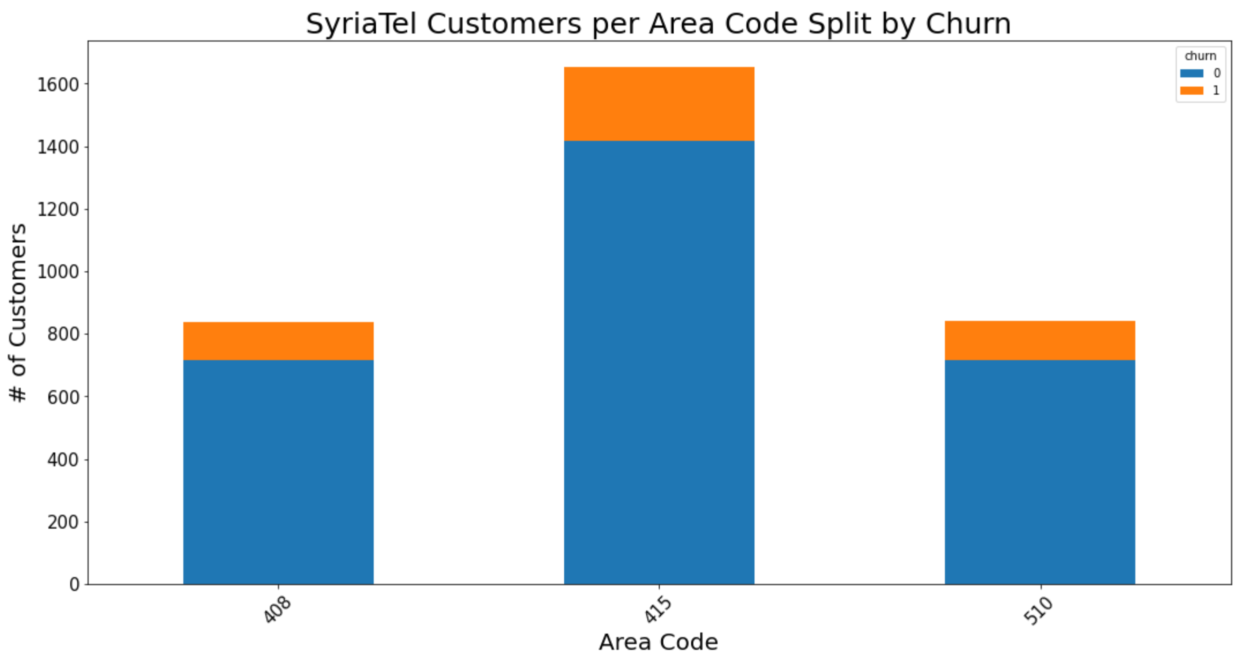
Below I wanted to see if there was any pattern with the churn rate versus the area codes provided in the dataset.

In [13]:
```
# percentage of churn by area code
print(data.groupby(["area_code"])['churn'].mean())

fig, ax = plt.subplots(figsize = (15, 8))
data.groupby(['area_code', 'churn']).size().unstack().plot(kind='bar',
plt.title('SyriaTel Customers per Area Code Split by Churn', fontsize =
ax.tick_params(axis = 'both', labelsize = 15)
plt.xlabel('Area Code', fontsize = 20)
plt.ylabel('# of Customers', fontsize = 20)
plt.tight_layout()

##rotate x-axis to a 45 degree angle
for label in ax.xaxis.get_ticklabels():
    label.set_rotation(45)
```

```
area_code
408    0.145585
415    0.142598
510    0.148810
Name: churn, dtype: float64
```



After further investigation, it is clear that the 415 area code has more customers than the 408 or 510 area codes. However, all three area codes have around the same **churn rate**. Since there is no clear pattern, I believe it is safe to delete the area code column from the dataset as well.

- **churn rate** – is the rate at which customers or clients are leaving a company within a specific period of time.

Drop area code column

Since area codes had no influence on the churn rate, I think it is safe to remove the column from our dataset.

In [14]:
```
# Drop the area_code column from the dataset
data.drop('area_code', axis=1, inplace=True)
```

In [15]:
```
# Check to see if phone_number and area_code columns have been removed
data.columns
```

Out[15]:
```
Index(['state', 'account_length', 'international_plan', 'voice_mail_pla
n',
       'number_vmail_messages', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_minutes', 'total_eve_calls',
       'total_eve_charge', 'total_night_minutes', 'total_night_calls',
       'total_night_charge', 'total_intl_minutes', 'total_intl_calls',
       'total_intl_charge', 'customer_service_calls', 'churn'],
      dtype='object')
```

EDA continued: Correlation between churn and other features

Below I used the .corr() function to see what features had the highest correlation with the target variable, churn.

In [16]:
```
# Correlation with Churn
data.corr().churn.sort_values(ascending=False)
```

Out[16]:
```
churn                     1.000000
international_plan        0.259852
customer_service_calls    0.208750
total_day_minutes         0.205151
total_day_charge          0.205151
total_eve_minutes         0.092796
total_eve_charge          0.092786
total_intl_charge         0.068259
total_intl_minutes        0.068239
total_night_charge        0.035496
total_night_minutes       0.035493
total_day_calls           0.018459
account_length            0.016541
total_eve_calls           0.009233
total_night_calls         0.006141
total_intl_calls         -0.052844
number_vmail_messages    -0.089728
voice_mail_plan          -0.102148
Name: churn, dtype: float64
```

international_plan, customer_service_calls, and total_day_charge/total_day_minutes have the highest correlation with churn. This will be later confirmed when I do the feature importance investigation.

Average churn rate for international plan holders

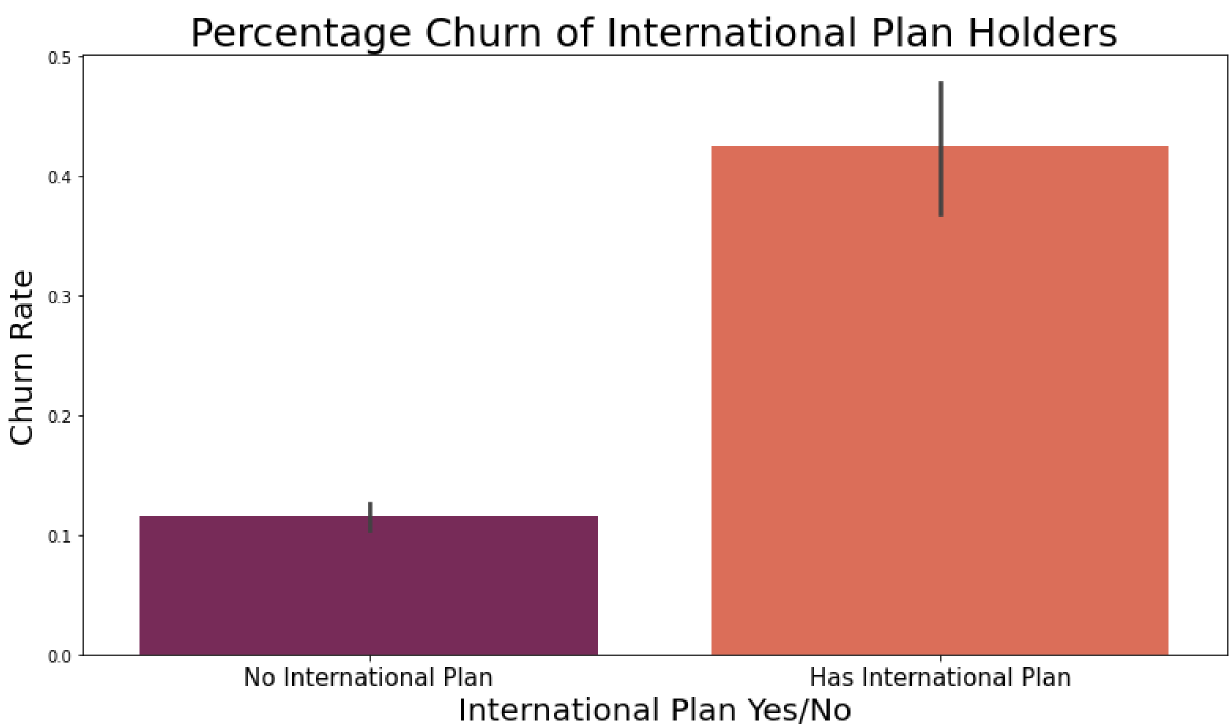
In [17]:
```
inter_plan_churn = pd.DataFrame(data.groupby(['international_plan'])['c
inter_plan_churn
```

Out[17]:

international_plan	churn
0	0.114950
1	0.424149

It appears 42% customers with an international plan with SyriaTel, end up churning. On a business stand point, this may be a worthwhile topic to further investigate.

In [18]:
```
# Internationl plan bar plot
fig, ax = plt.subplots(figsize=(13,7))
sns.barplot(data=data, x=data['international_plan'], y='churn', palette
plt.title('Percentage Churn of International Plan Holders', fontsize=25
plt.xlabel('International Plan Yes/No', fontsize=20)
plt.ylabel('Churn Rate', fontsize=20)
ax.set_xticklabels(['No International Plan', 'Has International Plan'],
plt.tight_layout;
```



For customer service calls, you would imagine the more calls a customer must make to customer service, the likely they are to be unhappy with their phone service. But how many calls on average does it take to increase the likely hood for a customer to churn. Let's take a look.

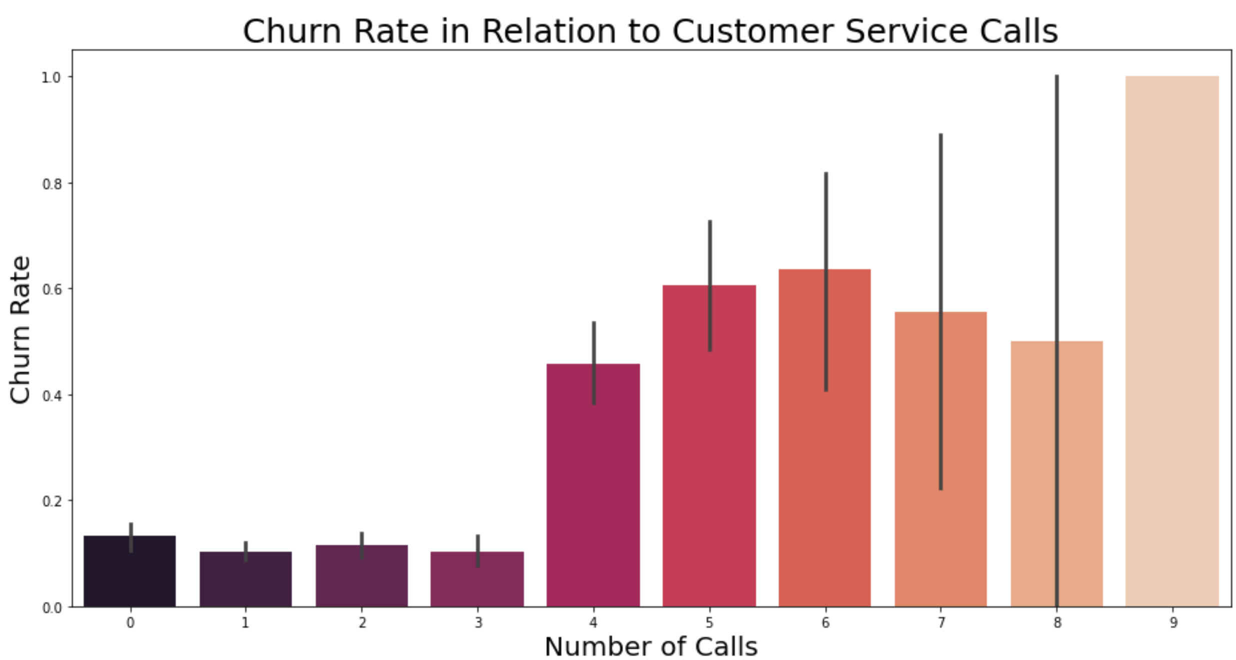
Average churn rate for number of customer service calls

In [19]:
```
cust_serv_calls = pd.DataFrame(data.groupby(['customer_service_calls'])
cust_serv_calls
```

Out[19]:

customer_service_calls	churn
0	0.131994
1	0.103302
2	0.114625
3	0.102564
4	0.457831
5	0.606061
6	0.636364
7	0.555556
8	0.500000
9	1.000000

In [20]:
```
# Bar plot for customer service calls
fig, ax = plt.subplots(figsize=(13,7))
sns.barplot(data=data, x=data['customer_service_calls'], y='churn',pale
plt.title('Churn Rate in Relation to Customer Service Calls', fontsize=
plt.ylabel('Churn Rate', fontsize=20)
plt.xlabel('Number of Calls', fontsize=20)
plt.tight_layout();
```



From the graph above, it is evident that when a customer has to call customer service four times, the likely hood of a customer to churn significantly increases. When a customer needs to call a maximum 9 times, the churn rate reaches 100%. Looking at this in a business perspective, new strategies must be discussed to handle unhappy customers when they are calling customer service by the fourth time.

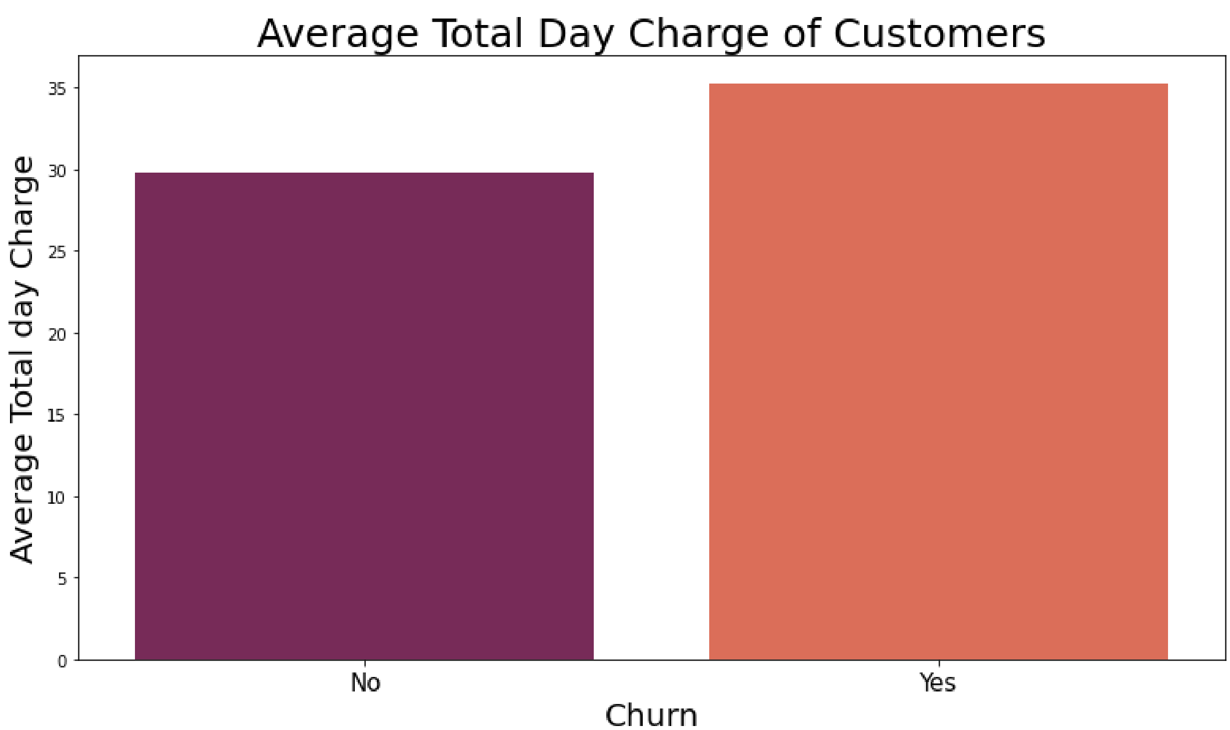
Average total day charge for churned and current customers

In [21]:
```
tdc = data.groupby(data['churn'])\
            ['total_day_charge'].mean().sort_values(ascending=False).rese
tdc.head()
```

Out[21]:

	churn	total_day_charge
0	1	35.175921
1	0	29.780421

In [22]:
```
# total_day_charge plan bar plot
fig, ax = plt.subplots(figsize=(13,7))
sns.barplot(data=tdc, x='churn', y='total_day_charge', palette="rocket"
plt.title('Average Total Day Charge of Customers', fontsize=25)
plt.xlabel('Churn', fontsize=20)
plt.ylabel('Average Total day Charge', fontsize=20)
ax.set_xticklabels(['No', 'Yes'], fontsize=15)
plt.tight_layout;
```



Graph above shows the average day charge a SyriaTel customer has, and when they are on average churning. Per the data, when a customer is around the 35.18 mark for their average day charge, they are more likely to churn. 29.78 and lower is the price the company should strive to be around in order to keep their customers from churning.

Create dummy variables for state column

Before modeling, there is one column that must be dealt with. The state column is categorical, and needs to be converted to dummy variables and added to the dataframe.

In [23]:
```
state_dum = pd.get_dummies(data['state'], drop_first=True)
```

In [24]:
```
data_final = data.drop('state', axis=1)
```

In [25]:
```
data_final = pd.concat([data_final, state_dum], axis=1)
data_final.head()
```

Out[25]:

	account_length	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	t
0	128	0	1	25	265.1	
1	107	0	1	26	161.6	
2	137	0	0	0	243.4	
3	84	1	0	0	299.4	
4	75	1	0	0	166.7	

5 rows × 68 columns

Prepare Data for Modeling

Create X, y variables

```
Below I am creating the target variable y, and the independant
variables X.
```

In [26]:
```
# Create X
X = data_final.drop('churn', axis=1)

# Create y
y = data_final['churn']
```

Train, Test, Split

below I am splitting the data into a training set and test set

In [27]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

Baseline Model

Logistic Regression

In [28]:
```
# Initialize logistic regression
logreg = LogisticRegression(solver= 'liblinear', random_state=42)

# fit the model
logreg.fit(X_train, y_train)
```

Out[28]:
```
LogisticRegression(random_state=42, solver='liblinear')
```

Get Predictions

In [29]:
```
y_hat_train = logreg.predict(X_train)
y_hat_test = logreg.predict(X_test)
```

Classification report of the training data

In [30]:
```
display(confusion_matrix(y_train, y_hat_train))
print(classification_report(y_train, y_hat_train))
```

```
array([[2099,   42],
       [ 278,   80]])

              precision    recall  f1-score   support

           0       0.88      0.98      0.93      2141
           1       0.66      0.22      0.33       358

    accuracy                           0.87      2499
   macro avg       0.77      0.60      0.63      2499
weighted avg       0.85      0.87      0.84      2499
```

In [31]:
```
print("Training Accuracy for Logistic Regression: {:.4}%".format(accura
```

```
Training Accuracy for Logistic Regression: 87.19%
```

After running the first classification report on the training data, our baseline model had a recall score of 22%. This is not very good and leaves a lot of room for improvement.

Check for imbalance

In [32]:
```
# check for imbalance of the training data
print(y_train.value_counts())
print('\n')
print(y_test.value_counts())
```

```
0    2141
1     358
Name: churn, dtype: int64


0    709
1    125
Name: churn, dtype: int64
```

It looks like we are dealing with some imbalance here. We could use SMOTE to deal with imbalance for future work.

Classification report of the testing data

In [33]:
```
display(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
```

```
array([[688,  21],
       [106,  19]])
              precision    recall  f1-score   support

           0       0.87      0.97      0.92       709
           1       0.47      0.15      0.23       125

    accuracy                           0.85       834
   macro avg       0.67      0.56      0.57       834
weighted avg       0.81      0.85      0.81       834
```

In [34]:
```
y_test.value_counts()
```

Out[34]:
```
0    709
1    125
Name: churn, dtype: int64
```

In [35]:
```
print("Test Accuracy for Logistic Regression: {:.4}%".format(accuracy_s
```

```
Test Accuracy for Logistic Regression: 84.77%
```

```
Our test set had an even lower score than our training set with a
recall score of only 15%. On a positive note, we did manage to lower
our false negative score. Let's try a different model.
```

Results:

- Training data recall score: `22%`
- Test data recall score: `15%`

Model 2: Decision Tree

Train the Decision Tree

In [36]:
```
clf = DecisionTreeClassifier(max_depth = 5, min_samples_leaf = 2, min_s

clf.fit(X_train, y_train)
```

Out[36]:
```
DecisionTreeClassifier(max_depth=5, min_samples_leaf=2, min_samples_split
=5,
                       random_state=42)
```

Decision trees require some pruning to become more accurate. For this model I used min_samples_split, and min_samples_leaf.

Evalutate the Predictive Performance

In [37]:
```
# Make prediction for test data
y_pred = clf.predict(X_test)
y_pred_train = clf.predict(X_train)
```

Display classification report for training data

In [38]:
```
#print confusion matrix and classification report
display(confusion_matrix(y_train, y_pred_train))
print(classification_report(y_train, y_pred_train))
```

```
array([[2129,   12],
       [ 101,  257]])
              precision    recall  f1-score   support

           0       0.95      0.99      0.97      2141
           1       0.96      0.72      0.82       358

    accuracy                           0.95      2499
   macro avg       0.96      0.86      0.90      2499
weighted avg       0.95      0.95      0.95      2499
```

In [39]:
```
print("Training Accuracy for Decision Tree: {:.4}%".format(accuracy_sco
```

```
Training Accuracy for Decision Tree: 95.48%
```

We have a much better recall score on our training data using the decision tree model. We improved from our baseline model from 22% previously, to 72% with the decision tree model.

Display classification report for test data

In [40]:
```
#print confusion matrix and classification report
display(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
array([[699,  10],
       [ 43,  82]])

              precision    recall  f1-score   support

           0       0.94      0.99      0.96       709
           1       0.89      0.66      0.76       125

    accuracy                           0.94       834
   macro avg       0.92      0.82      0.86       834
weighted avg       0.93      0.94      0.93       834
```

In [41]:
```
print("Test Accuracy for Decision Tree: {:.4}%".format(accuracy_score(y
```

```
Test Accuracy for Decision Tree: 93.65%
```

We have gotten a slightly lower recall score on our test data, but still scored much better than our baseline model. Also, our false negative number was reduced even further to 43.

Results:

- Training data recall score: `72%`
- Test data recall score: `66%`

Model 3: Random Forest Classifier

Create X and y variables

In [42]:
```
# assign X and y variables
X = data_final.drop('churn', axis =1)
y = data_final['churn']
```

Split the data into training and testing sets

In [43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2
```

Fit a random forest model

In [44]:
```
#Instantiate and fit a RandomForestClassifier
rf = RandomForestClassifier(n_estimators=10, max_depth=5, random_state=
rf.fit(X_train, y_train)
```

Out[44]:
```
RandomForestClassifier(max_depth=5, n_estimators=10, random_state=42)
```

Get prediction and build classification report

Get predictions for training and test data

In [45]:
```
# get predictions
y_pred = rf.predict(X_test)
y_pred_train = rf.predict(X_train)
```

Print confusion matrix and classification report for training data

In [46]:
```
#build a confusion matrix and classification report
display(confusion_matrix(y_train, y_pred_train))
print(classification_report(y_train, y_pred_train))
```

```
array([[2138,    3],
       [ 251,  107]])
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      2141
           1       0.97      0.30      0.46       358

    accuracy                           0.90      2499
   macro avg       0.93      0.65      0.70      2499
weighted avg       0.91      0.90      0.87      2499
```

In [47]:
```
print("Training Accuracy for Random Forest: {:.4}%".format(accuracy_sco
```

```
Training Accuracy for Random Forest: 89.84%
```

On our training data, the random forest model did not perform as well as the decision tree producing a recall score of 30%.

Print confusion matrix and classification report for test data

In [48]:
```
#build a confusion matrix and classification report
display(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
array([[705,   4],
       [ 96,  29]])
              precision    recall  f1-score   support

           0       0.88      0.99      0.93       709
           1       0.88      0.23      0.37       125

    accuracy                           0.88       834
   macro avg       0.88      0.61      0.65       834
weighted avg       0.88      0.88      0.85       834
```

In [49]:
```
print("Test Accuracy for Random Forest: {:.4}%".format(accuracy_score(y
```

```
Test Accuracy for Random Forest: 88.01%
```

On our test data, random forest forest again produced another low recall score of 23%. The false negative number was not improved from the decision tree model.

Results:

- Training data recall score: `30%`
- Test data recall score: `23%`

Tuning the model with GridSearchCV

I am now performing GridSearchCV on the random forest model, to see what hypertuning should be taken place in order to get the best performing random forest model.

In [50]:
```
param_grid = {'criterion':['gini','entropy'],
              'max_depth':[3,6,9,12],
              'min_samples_leaf': [5,10,12],
              'n_estimators': [10, 12, 15, 20]
             }
```

Fit the gridsearch

In [51]:
```
rfc = RandomForestClassifier(n_estimators=10, max_depth=5, random_state
grid_rfc =GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
grid_rfc.fit(X_train, y_train)
```

Out[51]:
```
GridSearchCV(cv=5,
             estimator=RandomForestClassifier(max_depth=5, n_estimators=1
0,
                                              random_state=42),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [3, 6, 9, 12],
                         'min_samples_leaf': [5, 10, 12],
                         'n_estimators': [10, 12, 15, 20]},
             scoring='accuracy')
```

Display the gridsearch results

In [52]:
```
# Print the best parameters
grid_rfc.best_params_
```

Out[52]:
```
{'criterion': 'gini',
 'max_depth': 12,
 'min_samples_leaf': 5,
 'n_estimators': 12}
```

With these recommendations, I will apply this hypertunings to the model to see if we get better results.

Get predictions from gridsearch

In [53]:
```
# get predictions from the gridsearch
y_pred_grid = grid_rfc.predict(X_test)
y_pred_train_grid = grid_rfc.predict(X_train)
```

Print confusion matrix and classification report for training data

In [54]:
```
#print confusion matrix and classification report
display(confusion_matrix(y_train, y_pred_train_grid))
print(classification_report(y_train, y_pred_train_grid))
```

```
array([[2140,    1],
       [ 105,  253]])

              precision    recall  f1-score   support

           0       0.95      1.00      0.98      2141
           1       1.00      0.71      0.83       358

    accuracy                           0.96      2499
   macro avg       0.97      0.85      0.90      2499
weighted avg       0.96      0.96      0.95      2499
```

In [55]:
```
print("Training Accuracy for Random Forest Classifier: {:.4}%".format(a
```

```
Training Accuracy for Random Forest Classifier: 95.76%
```

Great! We saw an improvement in our recall score from 30% to 71% on our training data after using gridSearch

Print confusion matrix and classification report for test data

In [56]:
```
#print confusion matrix and classification report
display(confusion_matrix(y_test, y_pred_grid))
print(classification_report(y_test, y_pred_grid))
```

```
array([[705,   4],
       [ 57,  68]])

              precision    recall  f1-score   support

           0       0.93      0.99      0.96       709
           1       0.94      0.54      0.69       125

    accuracy                           0.93       834
   macro avg       0.93      0.77      0.82       834
weighted avg       0.93      0.93      0.92       834
```

In [57]:
```
print("Test Accuracy for Random Forest Classifier: {:.4}%".format(accur
```

```
Test Accuracy for Random Forest Classifier: 92.69%
```

The recall score took a little dip when running it on our test data, resulting a recall score of 54%. The false negative number did decrease to 57.

Results:

- Training data recall score: `71%`

- Test data recall score: `54%`

Feature Importance

Next we will examine how important each feature ended up being in our final model. In machine learninge, feature selection is an important step. More features equals more complex models that take longer to train and are harder to interpret.

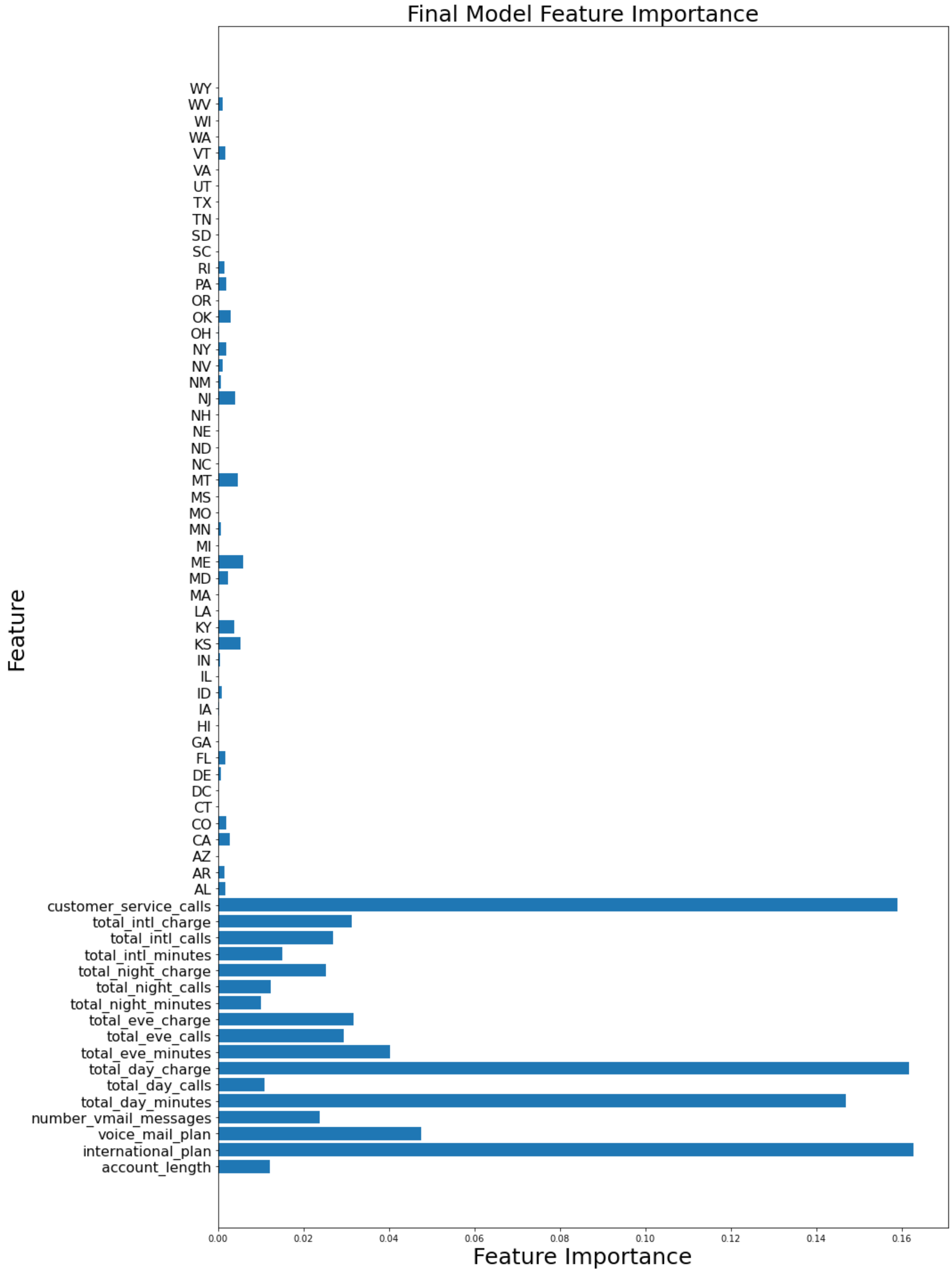
In [58]:
```
# Feature importance
rf.feature_importances_
```

Out[58]:
```
array([1.22494433e-02, 1.62707771e-01, 4.76229153e-02, 2.38905778e-02,
       1.46961828e-01, 1.09306397e-02, 1.61786301e-01, 4.02865581e-02,
       2.94028749e-02, 3.18338451e-02, 1.00894253e-02, 1.22712100e-02,
       2.53034318e-02, 1.49793444e-02, 2.70418095e-02, 3.13637003e-02,
       1.59027372e-01, 1.81158198e-03, 1.61274014e-03, 0.00000000e+00,
       2.79599790e-03, 1.89470501e-03, 0.00000000e+00, 0.00000000e+00,
       5.84599959e-04, 1.72440166e-03, 0.00000000e+00, 0.00000000e+00,
       2.72819229e-04, 9.15745437e-04, 0.00000000e+00, 4.87626773e-04,
       5.26425267e-03, 3.84937889e-03, 0.00000000e+00, 0.00000000e+00,
       2.25138758e-03, 5.96763790e-03, 0.00000000e+00, 7.69026920e-04,
       0.00000000e+00, 0.00000000e+00, 4.58178301e-03, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.99977346e-03,
       7.21375938e-04, 1.16223313e-03, 1.96181241e-03, 2.05621859e-04,
       3.03082977e-03, 0.00000000e+00, 1.97705363e-03, 1.49307190e-03,
       0.00000000e+00, 0.00000000e+00, 1.36076802e-04, 0.00000000e+00,
       0.00000000e+00, 0.00000000e+00, 1.66817014e-03, 0.00000000e+00,
       0.00000000e+00, 1.11004588e-03, 1.20263870e-06])
```

This array full of numbers isn't very helpful. Let's plot the data to see if the important features become more clear.

In [59]:
```
def plot_features_importances(model):
    n_features = X_test.shape[1]
    plt.figure(figsize=(15,20))
    plt.barh(range(n_features), model.feature_importances_, align='cent
    plt.yticks(np.arange(n_features), X_test.columns.values, fontsize =
    plt.xlabel('Feature Importance', fontsize = 25)
    plt.ylabel('Feature', fontsize = 25)
    plt.title('Final Model Feature Importance', fontsize = 25)
    plt.tight_layout()

plot_features_importances(rf)
```



we can see from this feature importance graph that there are three features that the model is weighing more heavily, with little to no weight given to the states.

- **`total_day_charge`**
- **`customer_service_calls`**
- **`international_plan`**

Conclusion

Logistic Regression:

- Recall Score (Training): `22%`
- Recall Score (Test): `15%`

Decision Tree:

- Recall Score (Training): `72%`
- Recall Score (Test): `66%`

Random Forest:

- Recall Score (Training): `30%`
- Recall Score (Test): `23%`

Random Forest with GridSearchCV:

- Recall Score (Training): `71%`
- Recall Score (Test): `54%`

From our findings, I can conclude that the decision tree model was the best testing model having the highest recall score on it's training data as well as it's test data.

Recommendations:

- 42% percent of the customers that have churned had international plans. Further discussion and investigations should be taken place to formulate a plan to retain these customers.
- Customers that have called customer service at least 4 times have a significantly increased chance of churning. Managers must come up with new training techniques to help customer service representatives assist these disgruntled customers.
- Investigate ways to retain customers that have an average total day charge of 35 dollars. Possibly creating more incentives and added perks to their phone plans could sway these customers from terminating their contracts.