Phase 3 Project

- Author: Jonathan Holt
- Flatiron Data Science
- 7/19/21 Cohort

Business Problem

SyriaTel is a telecomunications company that is concerned with the amount of customers that are leaving their service. (ie, Churn). They have provided a dataset of their most recent data which has 14.5% of the customers leaving during the time period captured in the dataset. I have been tasked with analyzing the data and looking for any areas where Churn is significant, and make recommendations as to what SyriaTel can do to greatly reduce the rate of churn in its customers.

The Data

The dataset that I was given to work with contains information for 3333 accounts, including:

- State
- Account Length
- Area Code
- Phone Number
- Extra Plans (VoiceMail and/or International)
- Minutes Used (Day, Evening, Night, International)
- Number of Calls (Day, Evening, Night, International)
- Number of VoiceMail Messages
- Total Amount Charged for minutes used (Day, Evening, Night, International)
- Number of calls to Customer Service

and most importantly:

- Churn: Customers who cancelled their service.

Questions to Answer

1. What is the Baseline Churn Rate?
2. What factors contribute to churn?
3. What factors have the biggest impact on churn?
4. What can be done to identify when a customer is at risk for churn?
5. What can be done to prevent churn?

What I will be looking for in my models

1. **High Recall Score:** I want my model to be able to predict which customers are at risk of churning. If it is tuned to be too sensitive in this area, that is fine. I would rather flag customers that aren't going to churn rather than focus on the customers that are likely to stay, and ending up with unexpected churn. I will keep this in balance by checking F1 Score.

2. **Good F1 Score:**While I am ultimately not concerned with Precision (how well the model predicts customers that will stay), a good F1 score means that the model is performing well on both Recall and Precision. Since Recall and Precision are inverses of each other, a good F1 score ensures that the model isn't skewed too far toward one or the other. (ie, a model that predicts EVERY customer will churn would have perfect Recall, but would be useless).

3. **High Cross Validation Score:** This ensures that the model isn't overly trained on the test data and that it does a good job of predicted unseen and unknown data. (ie, the test set).

4. **Area Under the Curve (AUC):** The ROC AUC Score measures the Area under the ROC curve, which means that it classifies the true positive rate against the false positive rate. The higher the score, the better performing the model is. That said, here is the scale that I will use to evaluate my models:

- **.69 or less: Model performs only slightly better than guessing and is worthless for my analysis.**
- **.70 - .79: Model still isn't performing very well, but is at minimum acceptable levels.**
- **.80 - .89: Model is performing fairly well. My goal is to be in this range or better.**
- **.90 - .99: Model is performing very well. I would be very happy to have a final model in this range.**

Data Preparation

Importing

In [1]:

```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import matplotlib.ticker as mtick
from sklearn.model_selection import train_test_split, cross_val_score, GridSearc
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, mean_squared_error, mean_squared_log
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_sc
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import plot_confusion_matrix
```

```python
from xgboost import XGBClassifier

import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

In [2]:
```python
pd.set_option('display.max_rows', 1000) #change the amount of rows displayed
plt.style.use('fivethirtyeight')
```

In [3]:
```python
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df.head()
```

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tot ev cal
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	9
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	10
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	1
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	8
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	12

5 rows × 21 columns

Fixing column names

In [4]:
```python
df.columns = df.columns.str.replace(' ','_')
df.columns
```

Out[4]:
```
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')
```

Inital Data Exploration

In [5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account_length          3333 non-null   int64
 2   area_code               3333 non-null   int64
```

```
 3   phone_number            3333 non-null   object
 4   international_plan      3333 non-null   object
 5   voice_mail_plan         3333 non-null   object
 6   number_vmail_messages   3333 non-null   int64
 7   total_day_minutes       3333 non-null   float64
 8   total_day_calls         3333 non-null   int64
 9   total_day_charge        3333 non-null   float64
 10  total_eve_minutes       3333 non-null   float64
 11  total_eve_calls         3333 non-null   int64
 12  total_eve_charge        3333 non-null   float64
 13  total_night_minutes     3333 non-null   float64
 14  total_night_calls       3333 non-null   int64
 15  total_night_charge      3333 non-null   float64
 16  total_intl_minutes      3333 non-null   float64
 17  total_intl_calls        3333 non-null   int64
 18  total_intl_charge       3333 non-null   float64
 19  customer_service_calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [6]:
```
df.isna().sum()
```

Out[6]:
```
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
```

In [7]:
```
df.phone_number.duplicated().sum()
```

Out[7]:
```
0
```

In [8]:
```
df.churn.value_counts()
```

Out[8]:
```
False    2850
True      483
Name: churn, dtype: int64
```

Initial Analysis

Base Churn Rate: 14.49%

- There are 3333 records.
- No missing values, and no duplicates.
- The number of customers who churned was 483.
- Therefore the base churn rate is (483 / 3333) = 14.49%

Target Feature Class Imbalance

- Churn is very imbalanced as the amount of customers who stay with the company is nearly 6X higher than the number of customers who leave.
- Class imbalance will need to be addressed in my models through one of (or a combination of) these options:

1. Class weight parameters
2. Oversampling or undersampling
3. Synthetic Minority Oversampling (SMOTE)

Cleaning and Preprocessing

In [9]:
```
test_df = df.copy()
test_df.head(2)
```

Out[9]:

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number
0	KS	128	415	382-4657	no	yes	
1	OH	107	415	371-7191	no	yes	

2 rows × 21 columns

Changing False to 0 and True to 1

In [10]:
```
test_df["churn"] = test_df["churn"].astype(int)
test_df.churn.value_counts()
```

Out[10]:
```
0    2850
1     483
Name: churn, dtype: int64
```

Dropping Uneccesary Columns

- These are all identifiers.

In [11]:
```
test_df = test_df.drop(columns=['state','phone_number', 'area_code'], axis=1)
```

Adding Column: Total Charge

- **This column captures the total charge that the customer will see on their bill from all charges contained within the dataset.**

In [12]:
```
total_charge = test_df.apply(lambda x: x['total_day_charge'] + x['total_eve_char
                             +x['total_intl_charge'], axis=1)
test_df['total_charge']= total_charge
```

Slicing out object type Features

In [13]:
```
cont_features = [col for col in test_df.columns if test_df[col].dtype in [np.flo
feature_df = test_df.loc[:, cont_features]
feature_df.head()
```

Out[13]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
0	128	25	265.1	110	45.07
1	107	26	161.6	123	27.47
2	137	0	243.4	114	41.38
3	84	0	299.4	71	50.90
4	75	0	166.7	113	28.34

One Hot Encoding

In [14]:
```
need_to_encode = test_df[['international_plan', 'voice_mail_plan', 'customer_ser
ohe = OneHotEncoder()
ohe.fit(need_to_encode)

ohe_1 = ohe.transform(need_to_encode).toarray()

ohe_df = pd.DataFrame(ohe_1, columns=ohe.get_feature_names(need_to_encode.column
ohe_df.head(2)
```

Out[14]:

	international_plan_no	international_plan_yes	voice_mail_plan_no	voice_mail_plan_yes	custor
0	1.0	0.0	0.0	1.0	
1	1.0	0.0	0.0	1.0	

In [15]:
```
# Combining everything together
cleaned_df = pd.concat([pd.DataFrame(feature_df), ohe_df], axis=1)
cleaned_df.head(2)
```

Out[15]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
0	128	25	265.1	110	45.07
1	107	26	161.6	123	27.47

2 rows × 31 columns

Dropping one value for categoricals

In [16]:
```
#Dropping a few of the redundant values.
cleaned_df_2 = cleaned_df.copy()
cleaned_df_2 = cleaned_df_2.drop(['international_plan_no', 'voice_mail_plan_no']
```

Creating Variable: Customer Service Calls High

- Customer Service Calls High = 4 or more Customer Service Calls.
- This is to determine if a high number of Customer Service Calls affects Churn.
- I'm introducing only the High Customer Service Calls variable as any account that isn't in the "High" range is in the "Low" range.

In [17]:
```
cs_calls_high= cleaned_df_2.apply(lambda x: x['customer_service_calls_4'] + x['c
                              + x['customer_service_calls_6'] + x['customer_serv
                             + x['customer_service_calls_8']+ x['customer_service

cleaned_df_2['cs_calls_high'] = cs_calls_high
```

In [18]:
```
cleaned_df_2 = cleaned_df_2.drop(['customer_service_calls_0', 'customer_service_
                          'customer_service_calls_4','customer_service_calls_5','cust
                          'customer_service_calls_7','customer_service_calls_8','cust
                          , axis=1)
cleaned_df_2.head(2)
```

Out[18]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge
0	128	25	265.1	110	45.07
1	107	26	161.6	123	27.47

In [19]:
```
total_charge = cleaned_df_2.apply(lambda x: x['total_day_charge'] + x['total_eve
                              +x['total_intl_charge'], axis=1)
cleaned_df_2['total_charge']= total_charge
```

Creating Model DataFrame

In [20]:
```
model_df = cleaned_df_2.copy()
model_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   account_length         3333 non-null   int64
 1   number_vmail_messages  3333 non-null   int64
 2   total_day_minutes      3333 non-null   float64
 3   total_day_calls        3333 non-null   int64
 4   total_day_charge       3333 non-null   float64
```

```
 5   total_eve_minutes       3333 non-null   float64
 6   total_eve_calls         3333 non-null   int64
 7   total_eve_charge        3333 non-null   float64
 8   total_night_minutes     3333 non-null   float64
 9   total_night_calls       3333 non-null   int64
 10  total_night_charge      3333 non-null   float64
 11  total_intl_minutes      3333 non-null   float64
 12  total_intl_calls        3333 non-null   int64
 13  total_intl_charge       3333 non-null   float64
 14  customer_service_calls  3333 non-null   int64
 15  churn                   3333 non-null   int64
 16  total_charge            3333 non-null   float64
 17  international_plan_yes  3333 non-null   float64
 18  voice_mail_plan_yes     3333 non-null   float64
 19  cs_calls_high           3333 non-null   float64
dtypes: float64(12), int64(8)
memory usage: 520.9 KB
```

In [21]:
```
model_df=model_df.drop(['customer_service_calls','total_day_charge', 'total_eve_
                                          , axis=1)
```

These are now redundant as that information is now being captured in different fields and I don't want extra noise in my model.

In [22]:
```
model_df.head(2)
```

Out[22]:

	account_length	number_vmail_messages	total_day_minutes	total_day_calls	total_eve_minute
0	128	25	265.1	110	197.4
1	107	26	161.6	123	195.5

Dealing With Churn Class Imbalance

- **Always use class weight parameter in Decision Tree Classifier**
- **Always stratify Train Test Split.**
- **Add SMOTE to Training Sets.**

In [23]:
```
balanced_df = model_df.copy()

X = balanced_df.drop(['churn'], axis=1)
y = balanced_df['churn']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, stratif

smote = SMOTE(random_state=23)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)
```

Metrics Function

- **A Function to quickly calulate and display the metrics that I care about.**

1. Recall
2. F1 Score
3. ROC AUC Score

In [24]:
```
def get_metrics(clf, y_pred):

    clf_rcl = recall_score(y_test, y_pred) * 100
    print('Recall is :{0}'.format(clf_rcl))
    clf_f1 = f1_score(y_test, y_pred) * 100
    print('F1 Score is :{0}'.format(clf_f1))
    false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pr
    clf_roc_auc = auc(false_positive_rate, true_positive_rate)
    print('ROC AUC is :{0}'.format(round(clf_roc_auc, 2)))
```

Baseline Decision Tree

In [25]:
```
dt1 = DecisionTreeClassifier(random_state=23, class_weight="balanced")
dt1.fit(X_train_resampled, y_train_resampled)
dt1_y_pred = dt1.predict(X_test)
```

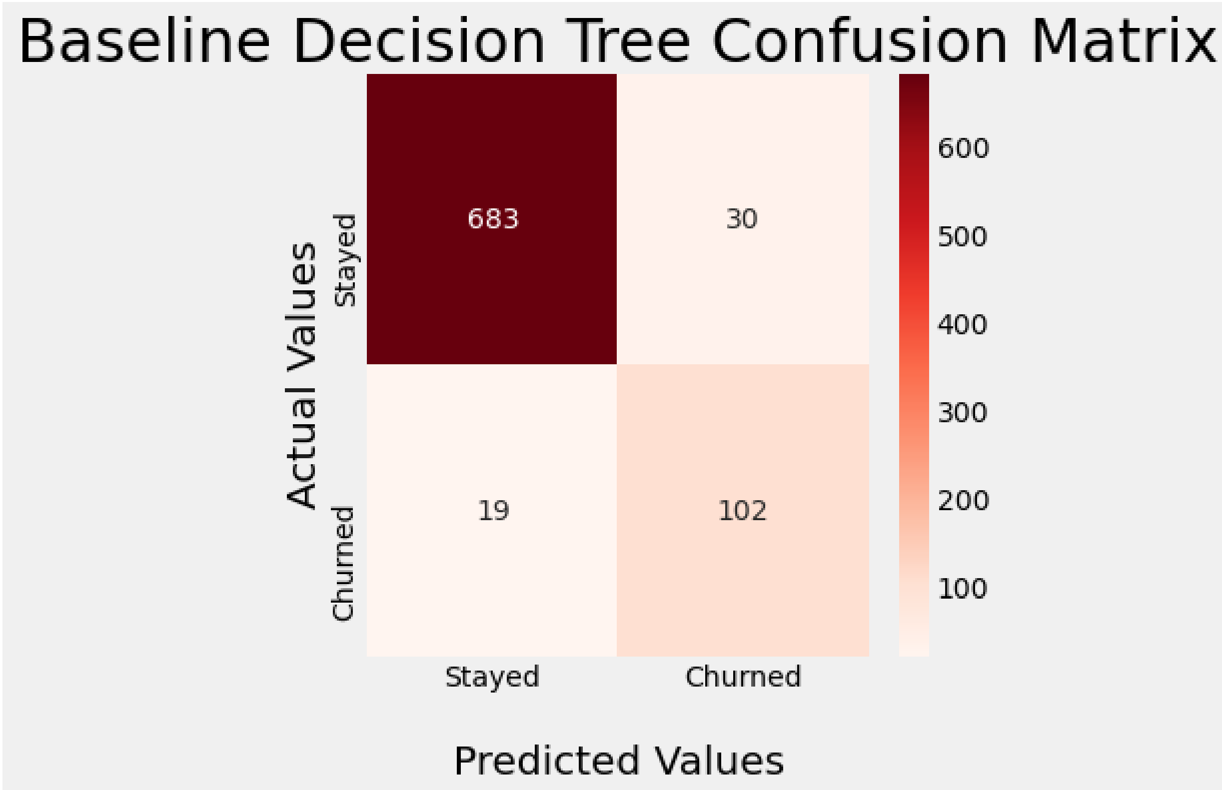
In [26]:
```
get_metrics(dt1, dt1_y_pred)
```

```
Recall is :84.29752066115702
F1 Score is :80.63241106719367
ROC AUC is :0.9
```

In [27]:
```
dt1_cv_score = np.mean(cross_val_score(dt1, X_train_resampled, y_train_resampled
dt1_cv_score
```

Out[27]:
```
0.9513336947237007
```

In [28]:
```
dt1_matrix = confusion_matrix(y_test, dt1_y_pred)

fig, ax = plt.subplots(figsize=(5,5))

ax = sns.heatmap(dt1_matrix, annot=True, cmap='Reds', fmt='d')

ax.set_title('Baseline Decision Tree Confusion Matrix', fontsize = 30);
ax.set_xlabel('\nPredicted Values',fontsize = 20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Stayed','Churned'])
ax.yaxis.set_ticklabels(['Stayed','Churned'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Analysis:

- **All of my work in preprocessing has paid off. The first model is already performing well.**
- **Recall is decent, F1 score is okay, and ROC AUC score is very good.**
- **Still, I will use GridSearch to further optimize my Decision Tree, and then use Random Forests to see if I can further refine and improve my model.**

Refining Decision Tree through GridSearchCV

In [29]:
```
dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 3, 4, 5, 6],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 3, 4, 5, 6]
}
```

In [30]:
```
# Instantiate GridSearchCV
dt2 = DecisionTreeClassifier(random_state=23)

dt_grid_search = GridSearchCV(dt2, dt_param_grid, cv=3, scoring = 'recall')

# Fit to the data
dt_grid_search.fit(X_train_resampled, y_train_resampled)
```

Out[30]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(random_state=23),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [None, 2, 3, 4, 5, 6],
                         'min_samples_leaf': [1, 2, 3, 4, 5, 6],
```

```
                            'min_samples_split': [2, 5, 10]},
              scoring='recall')
```

In [31]:
```
dt_grid_search.best_params_
```

Out[31]:
```
{'criterion': 'entropy',
 'max_depth': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2}
```

Decision Tree 2 (Using Grid Search Parameters)

In [32]:
```
dt2 = DecisionTreeClassifier(criterion='entropy', max_depth=None, min_samples_sp
                             min_samples_leaf=1, class_weight='balanced', random
dt2.fit(X_train_resampled, y_train_resampled)
dt2_y_pred = dt2.predict(X_test)
```

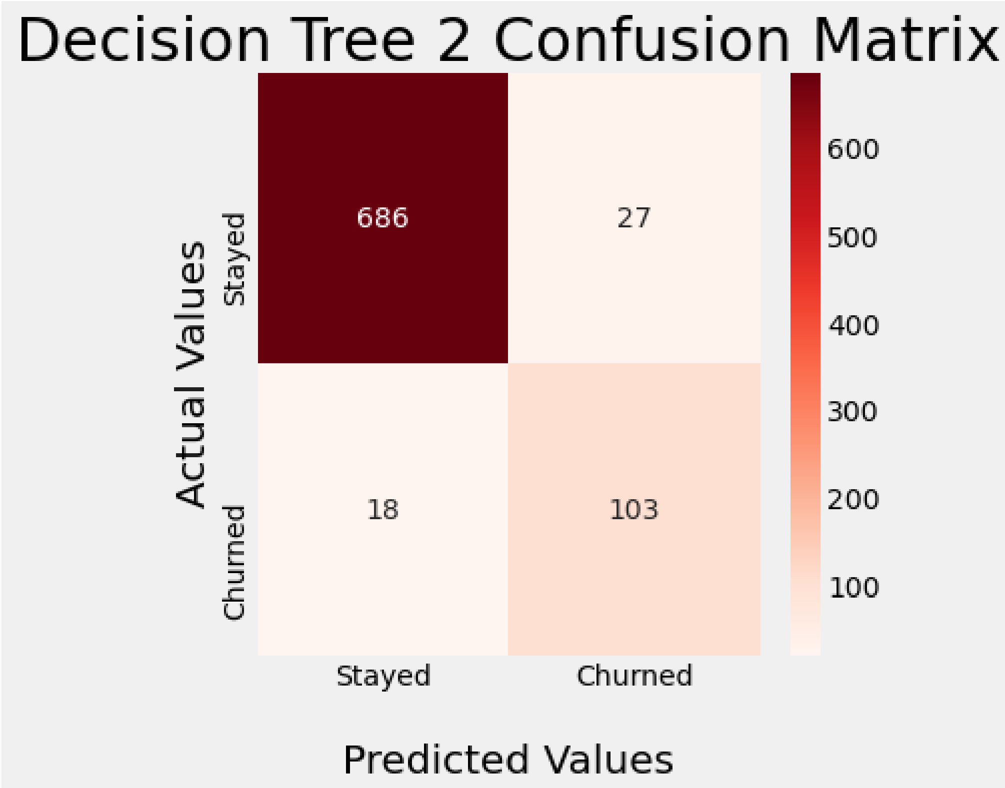
In [33]:
```
get_metrics(dt2, dt2_y_pred)
```

```
Recall is :85.12396694214877
F1 Score is :82.07171314741036
ROC AUC is :0.91
```

In [34]:
```
dt2_cv_score = np.mean(cross_val_score(dt2, X_train_resampled, y_train_resampled
dt2_cv_score
```

Out[34]:
```
0.9588203889874499
```

In [35]:
```
dt2_matrix = confusion_matrix(y_test, dt2_y_pred)

fig, ax = plt.subplots(figsize=(5,5))

ax = sns.heatmap(dt2_matrix, annot=True, cmap='Reds', fmt='d')

ax.set_title('Decision Tree 2 Confusion Matrix', fontsize = 30);
ax.set_xlabel('\nPredicted Values',fontsize = 20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Stayed','Churned'])
ax.yaxis.set_ticklabels(['Stayed','Churned'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Analysis:

- **Decision Tree 2 performs slightly better than Decision Tree 1 on all metrics.**

Random Forests 1

- **Moving on to a more Complex model to see if I get better performance.**

In [36]:
```
rf1_clf = RandomForestClassifier(random_state=23, class_weight="balanced")
rf1_clf.fit(X_train_resampled, y_train_resampled)
rf1_y_pred = rf1_clf.predict(X_test)
```

In [37]:
```
get_metrics(rf1_clf, rf1_y_pred)
```

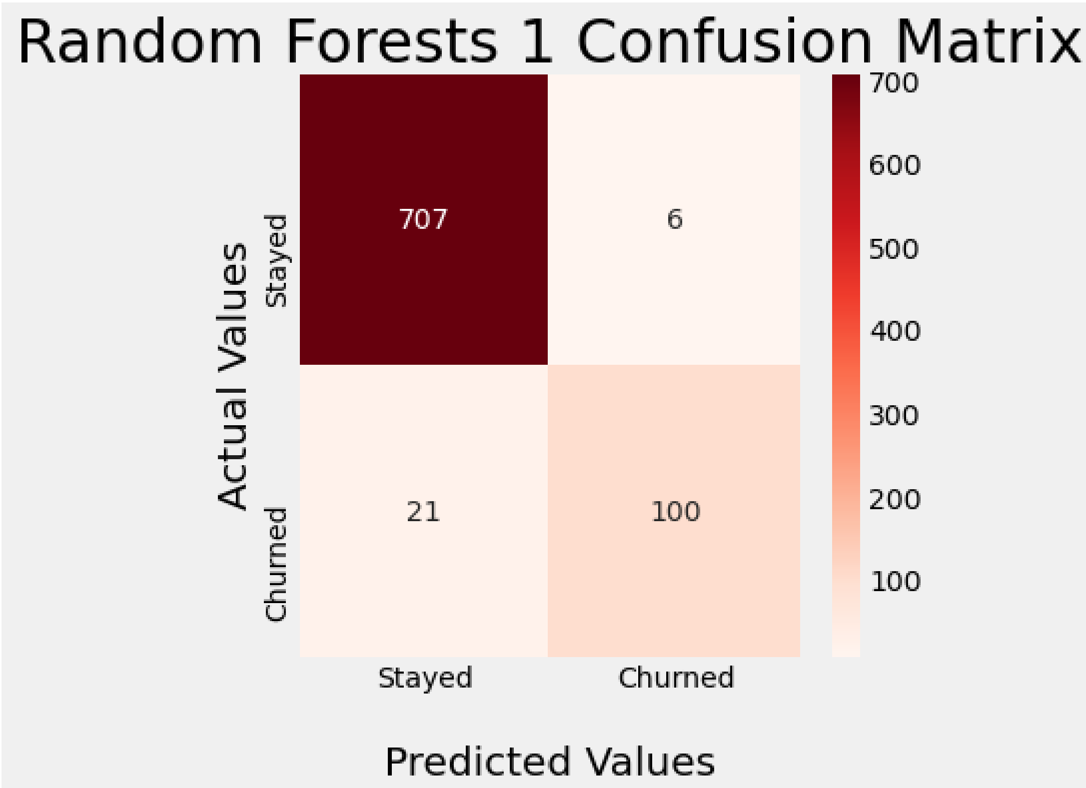
```
Recall is :82.64462809917356
F1 Score is :88.10572687224669
ROC AUC is :0.91
```

In [38]:
```
rf1_cv_score = np.mean(cross_val_score(rf1_clf, X_train_resampled, y_train_resam
rf1_cv_score
```

Out[38]:
```
0.971224127735068
```

In [39]:
```
rf1_matrix = confusion_matrix(y_test, rf1_y_pred)

fig, ax = plt.subplots(figsize=(5,5))

ax = sns.heatmap(rf1_matrix, annot=True, cmap='Reds', fmt='d')

ax.set_title('Random Forests 1 Confusion Matrix', fontsize = 30);
```

```
ax.set_xlabel('\nPredicted Values',fontsize = 20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Stayed','Churned'])
ax.yaxis.set_ticklabels(['Stayed','Churned'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Analysis

- My Random Forests model performs better than my Decision Trees in ROC AUC Score and F1 Score.
- The Recall Score is a slightly lower than my Decision Trees, but I hope that running a Grid Search will improve this score.

GridSearch CV

In [40]:
```
rf_param_grid = {
    'n_estimators': [10, 30, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 2, 6, 10],
    'min_samples_split': [5, 10],
    'min_samples_leaf': [3, 6]
}
```

In [41]:
```
rf2_clf = RandomForestClassifier(random_state=23)


rf1_grid_search= GridSearchCV(rf2_clf, rf_param_grid, scoring = 'recall', cv=3)
```

```
rf1_grid_search.fit(X_train_resampled, y_train_resampled)

print("")
print(f"Random Forest  Optimal Parameters: {rf1_grid_search.best_params_}")
```

```
Random Forest  Optimal Parameters: {'criterion': 'gini', 'max_depth': None, 'min
_samples_leaf': 3, 'min_samples_split': 5, 'n_estimators': 30}
```

Random Forests 2 (Using Parameters from GridSearchCV)

In [42]:
```
rf2_clf = RandomForestClassifier(criterion= 'gini', max_depth= None, min_samples
                                 min_samples_split= 5, n_estimators= 30, random_s
                                 class_weight='balanced')
rf2_clf.fit(X_train_resampled, y_train_resampled)
rf2_y_pred = rf2_clf.predict(X_test)
```

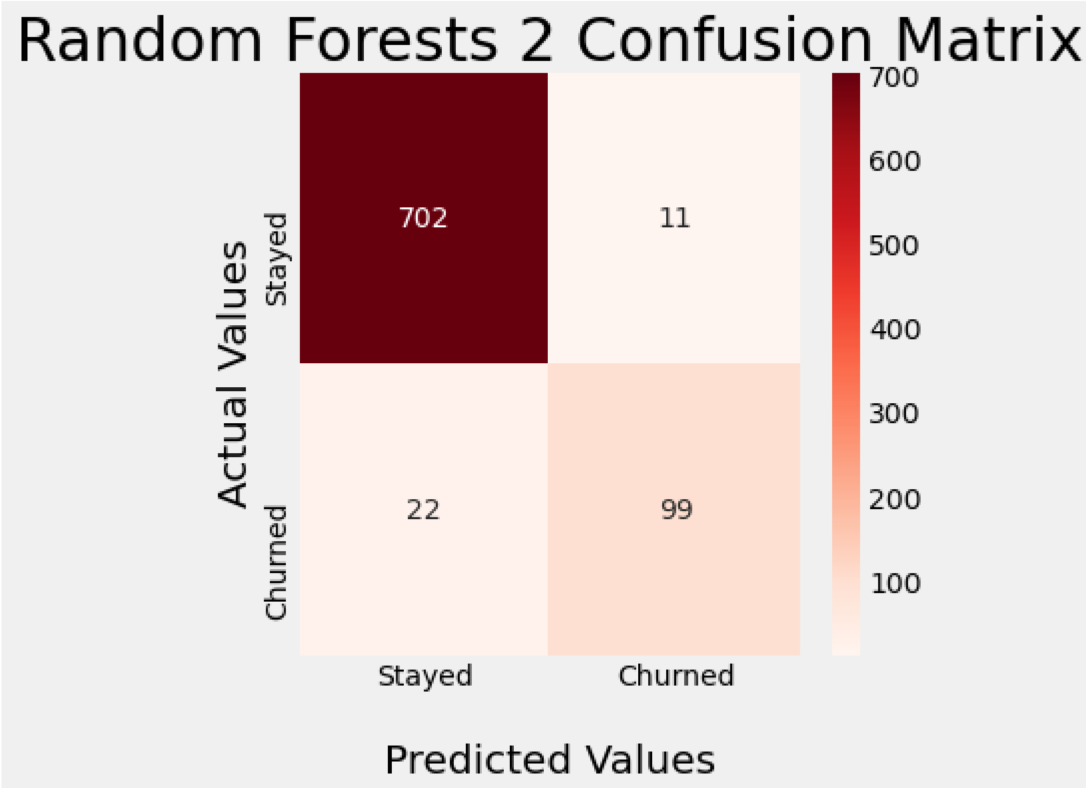
In [43]:
```
get_metrics(rf2_clf, rf2_y_pred)
```

```
Recall is :81.81818181818183
F1 Score is :85.71428571428572
ROC AUC is :0.9
```

In [44]:
```
rf2_cv_score = np.mean(cross_val_score(rf2_clf, X_train_resampled, y_train_resam
rf2_cv_score
```

Out[44]:
```
0.9660757934161247
```

In [45]:
```
rf2_matrix = confusion_matrix(y_test, rf2_y_pred)

fig, ax = plt.subplots(figsize=(5,5))

ax = sns.heatmap(rf2_matrix, annot=True, cmap='Reds', fmt='d' )

ax.set_title('Random Forests 2 Confusion Matrix', fontsize = 30);
ax.set_xlabel('\nPredicted Values',fontsize = 20)
ax.set_ylabel('Actual Values ', fontsize=20);

## Ticket labels - List must be in alphabetical order
ax.xaxis.set_ticklabels(['Stayed','Churned'])
ax.yaxis.set_ticklabels(['Stayed','Churned'])

## Display the visualization of the Confusion Matrix.
plt.show()
```



Analysis:

- **Even though I used GridSearch and set it to prioritize Recall, this model doesn't do as well as the previous ones.**

XGBOOST Model

- **I will now try incorporating Gradient Boosting to see if that can improve my model.**

In [46]:
```
# Instantiate XGBClassifier
clf = XGBClassifier(random_state=23)

# Fit XGBClassifier
xg1 = clf.fit(X_train_resampled, y_train_resampled)

# Predict on training and test sets
training_preds = clf.predict(X_train_resampled)
xg1_y_pred = clf.predict(X_test)
```

In [47]:
```
get_metrics(xg1, xg1_y_pred)
```

```
Recall is :84.29752066115702
F1 Score is :91.4798206278027
ROC AUC is :0.92
```

In [48]:
```
xg1_cv_score = np.mean(cross_val_score(xg1, X_train_resampled, y_train_resampled
xg1_cv_score
```

Out[48]:
```
0.9794115907746895
```

Analysis:

- **The XGBoost Model performs closer to the Decision Tree Models. It has the highest F1 Score so far, but Recall is still my biggest criteria and it is slightly below Decision Tree 2.**
- **I will run GridSearch CV and see if that improves model performance.**

GridSearch

In [49]:
```
boost_param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [100],
}
```

In [50]:
```
xg2 = XGBClassifier(random_state=23)

grid_clf = GridSearchCV(xg2, boost_param_grid, scoring='recall', cv=3, n_jobs=1)
grid_clf.fit(X_train_resampled, y_train_resampled)

best_parameters = grid_clf.best_params_

print('Grid Search found the following optimal parameters: ')
for param_name in sorted(best_parameters.keys()):
    print('%s: %r' % (param_name, best_parameters[param_name]))
```

```
Grid Search found the following optimal parameters:
learning_rate: 0.2
max_depth: 6
min_child_weight: 1
n_estimators: 100
subsample: 0.7
```

In [51]:
```
xg2 = XGBClassifier(learning_rate= 0.2, max_depth=6, min_child_weight=1,
                                n_estimators=100, subsample=0.7, random_state=23
xg2.fit(X_train_resampled, y_train_resampled)
xg2_y_pred = xg2.predict(X_test)
```

In [52]:
```
get_metrics(xg2, xg2_y_pred)
```

```
Recall is :85.12396694214877
F1 Score is :90.35087719298247
ROC AUC is :0.92
```

In [53]:
```
xg2_cv_score = np.mean(cross_val_score(xg2, X_train_resampled, y_train_resampled
xg2_cv_score
```

Out[53]:
```
0.9761367369735199
```

Choosing a Final Model

XGBoost Model 2 is my best performing model.

- **It is tied for best Recall Score with Decision Tree 2 at 85.12.**

- **It has the second highest F1 Score at 90.35, just slightly below XGBoost Model 1.**
- **It has the highest Area Under ROC Curve at .9228.**

XGBoost Model 2 is my final model, and will be used for final analysis and recommendations.

Feature Importance

- **I will now see which features had the most importance in relation to Churn in my Final Model.**

In [54]:
```
feature_names = list(X)
feature_names
```

Out[54]:
```
['account_length',
 'number_vmail_messages',
 'total_day_minutes',
 'total_day_calls',
 'total_eve_minutes',
 'total_eve_calls',
 'total_night_minutes',
 'total_night_calls',
 'total_intl_minutes',
 'total_intl_calls',
 'total_charge',
 'international_plan_yes',
 'voice_mail_plan_yes',
 'cs_calls_high']
```

In [55]:
```
xg2_importance = xg2.feature_importances_
xg2_importance
```

Out[55]:
```
array([0.01555622, 0.01647341, 0.01180071, 0.01209497, 0.01489897,
       0.01386264, 0.01720872, 0.01302678, 0.02353546, 0.05303666,
       0.13529545, 0.19107383, 0.10892226, 0.3732139 ], dtype=float32)
```

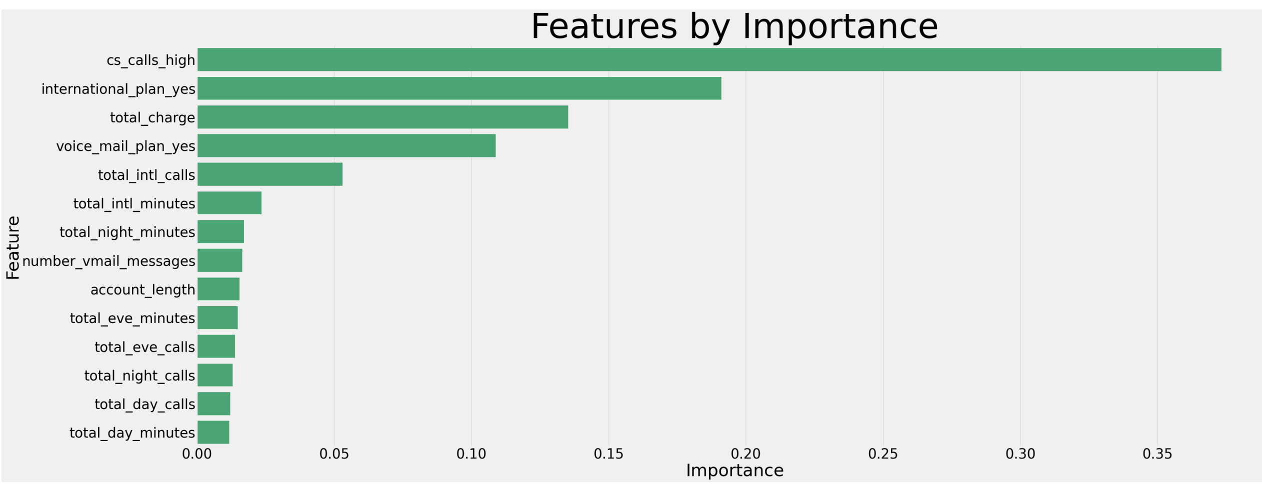
In [56]:
```
#feature_importance_df = pd.DataFrame(rf2_importance, feature_names)
feature_importance_df = pd.DataFrame(xg2_importance, feature_names)
feature_importance_df= feature_importance_df.reset_index()
feature_importance_df.rename(columns={'index': 'Feature', 0: 'Importance'}, inpl
feature_importance_df = feature_importance_df.sort_values('Importance', ascendin
feature_importance_df
```

Out[56]:

	Feature	Importance
13	cs_calls_high	0.373214
11	international_plan_yes	0.191074
10	total_charge	0.135295
12	voice_mail_plan_yes	0.108922
9	total_intl_calls	0.053037
8	total_intl_minutes	0.023535
6	total_night_minutes	0.017209
1	number_vmail_messages	0.016473

	Feature	Importance
0	account_length	0.015556
4	total_eve_minutes	0.014899
5	total_eve_calls	0.013863
7	total_night_calls	0.013027
3	total_day_calls	0.012095
2	total_day_minutes	0.011801

In [57]:
```
# plot feature importance
fig, ax = plt.subplots(figsize=(50,20))
p = sns.barplot(data=feature_importance_df, x='Importance', y='Feature', color =
p.set_xlabel("Importance", fontsize = 50)

p.set_ylabel("Feature", fontsize = 50)
plt.xticks(fontsize=40)
plt.yticks(fontsize=40)

p.set_title("Features by Importance", fontsize = 100)
plt.figsize=(30,20)
plt.savefig('images/project_3_Feature_Importance')

plt.show();
```



Analysis

The Top 4 features with importance in relation to churn are:

1. **A High Amount of Customer Service Calls**
2. **Whether or not Customer has International Plan.**
3. **Total Charge that Customer has.**
4. **Whether or not Customer has a Voice Mail Plan.**

All other features have (at most) half of the feature significance as the top 4. However, it is important to note that features 5 & 6 are both related to the International Plan.

I will take a closer look at each of these features as they have the most impact on churn by far.

All other features have (at most) half of the feature significance as the top 4. I will take a closer look at each of these features as they have the most impact on churn by far.

Analyzing Churn Rate in Important Features

Customer Service Calls

In [58]:
```
analysis_df = cleaned_df.copy()
```

In [59]:
```
df.customer_service_calls.describe()
```

Out[59]:
```
count    3333.000000
mean        1.562856
std         1.315491
min         0.000000
25%         1.000000
50%         1.000000
75%         2.000000
max         9.000000
Name: customer_service_calls, dtype: float64
```

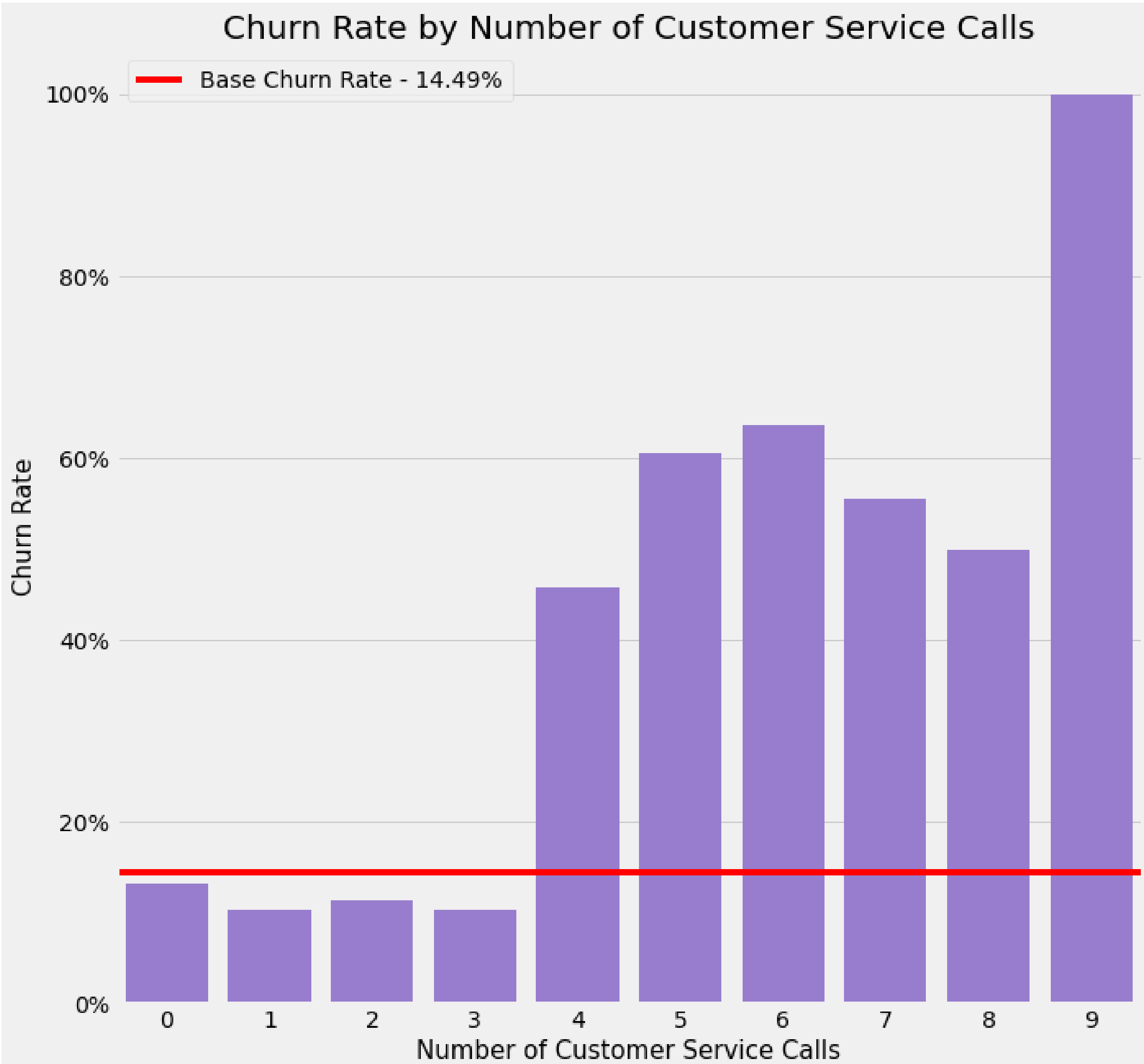
In [60]:
```
cs_churn_df = analysis_df.groupby('customer_service_calls')['churn'].sum().reset
cs_churn_df = cs_churn_df.rename(columns={"customer_service_calls": "#_of_calls"
variable_1 = analysis_df['customer_service_calls'].value_counts().reset_index()
variable_1 = variable_1.rename(columns={"index": "#_of_calls", "customer_service
cs_churn_df = cs_churn_df.merge(variable_1, on='#_of_calls')
churn_rate = cs_churn_df.apply(lambda x: x['churn'] / x['#_of_accounts'], axis=1
cs_churn_df['churn_rate']= churn_rate
cs_churn_df
```

Out[60]:

	#_of_calls	churn	#_of_accounts	churn_rate
0	0	92	697	0.131994
1	1	122	1181	0.103302
2	2	87	759	0.114625
3	3	44	429	0.102564
4	4	76	166	0.457831
5	5	40	66	0.606061
6	6	14	22	0.636364
7	7	5	9	0.555556
8	8	1	2	0.500000
9	9	2	2	1.000000

In [61]:
```
fig, ax = plt.subplots(figsize=(10,10))
p = sns.barplot(x="#_of_calls", y="churn_rate", data=cs_churn_df, color='mediump

p.set_xlabel("Number of Customer Service Calls", fontsize = 15)
```

```
p.set_ylabel("Churn Rate", fontsize = 15)

ax.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1, decimals=None, symbo
p.set_title("Churn Rate by Number of Customer Service Calls", fontsize = 20)
plt.figsize=(30,20)
line = plt.axhline(y=.145, color='red')
plt.savefig('images/project_3_CS_Churn_Rate')
ax.legend([line], ['Base Churn Rate - 14.49%'])

plt.show();
```



Analysis:

- **There is a very strong relationship between the number of Customer Service Calls and Churn Rate.**
- **If there are 0-3 calls, those customers are below the avg. churn rate.**
- **At 4 Calls, the Churn Rate jumps to 45.7%, 4X the avg. churn rate.**
- **The Mode for Customer Service Calls is 1, with 2 or more calls being in the top quartile.**
- **Over Half of all customers make 1 or less customer service calls. (1878 of 3333: 56%)**

- **Hypothesis is that customers that are unhappy (and therefore more likely to cancel their service) are calling customer service more often.**

International Plan

In [62]:
```
intl_df = analysis_df[['international_plan_yes', 'international_plan_no', 'churn
intl_churn_df = intl_df.groupby('churn').sum().reset_index()
intl_churn_df = intl_churn_df.transpose()
intl_churn_df = intl_churn_df.rename(columns={0: "stayed", 1: "churned"})
intl_churn_df['total'] = intl_churn_df.apply(lambda x: x['stayed'] + x['churned'
intl_churn_df['churn_rate'] = intl_churn_df.apply(lambda x: x['churned'] / x['to
intl_churn_df
```

Out[62]:

	stayed	churned	total	churn_rate
churn	0.0	1.0	1.0	1.000000
international_plan_yes	186.0	137.0	323.0	0.424149
international_plan_no	2664.0	346.0	3010.0	0.114950

In [63]:
```
df2 = df.copy()
df2 = df2[['international_plan', 'total_intl_minutes', 'total_intl_calls', 'tota
           'customer_service_calls', 'total_day_minutes', 'total_day_charge', 'ch
```

In [64]:
```
df2.groupby('international_plan').mean()
```

Out[64]:

	total_intl_minutes	total_intl_calls	total_intl_charge	customer_service_calls	t
international_plan					
no	10.195349	4.465449	2.753279	1.573422	
yes	10.628173	4.609907	2.869907	1.464396	

Analysis:

NOTE: Data shows that Customers without the international plan were still able to make international calls. I am operating under the assumption that the data is correct and that there is a seperate International Plan, as indicated bty the "International Plan" column. I am also assuming that the data contained in that field is accurate.

- **only 323 people (9.5% of customers) have international plans. But those that do have a high rate of churn.**
- **churn rate for customers with an international plan is 42.4% vs 11.5% for those without an international plan.**
- **nearly 4X increase in churn rate.**
- **customers without an international plan are actually under the avg. churn rate, but are close to it.**

- **International Minutes and the Number of International Calls were the 5th and 6th most important features. There is definitely something wrong with SyriaTel's International Plan. This will be reflected in my Recommendations.**

Total Charge

In [65]:
```
analysis_df['total_charge'].describe()
```

Out[65]:
```
count    3333.000000
mean       59.449754
std        10.502261
min        22.930000
25%        52.380000
50%        59.470000
75%        66.480000
max        96.150000
Name: total_charge, dtype: float64
```

In [66]:
```
charge_df = analysis_df[['total_charge', 'churn']]
charge_df['charge_group'] = pd.qcut(analysis_df['total_charge'], 200)
group_counts = charge_df.charge_group.value_counts().reset_index()
group_counts = group_counts.rename(columns={"index": "charge_group", "charge_gro
charge_df= charge_df.groupby('charge_group').mean()
charge_df= charge_df.rename(columns={'total_charge': 'group_mean'})
charge_df.head()
```

```
<ipython-input-66-f666cffbefb5>:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stab
le/user_guide/indexing.html#returning-a-view-versus-a-copy
  charge_df['charge_group'] = pd.qcut(analysis_df['total_charge'], 200)
```

Out[66]:

	group_mean	churn
charge_group		
(22.929, 32.497]	28.372353	0.235294
(32.497, 33.853]	33.329412	0.117647
(33.853, 35.72]	34.832500	0.062500
(35.72, 37.436]	36.645882	0.058824
(37.436, 38.752]	38.102941	0.000000

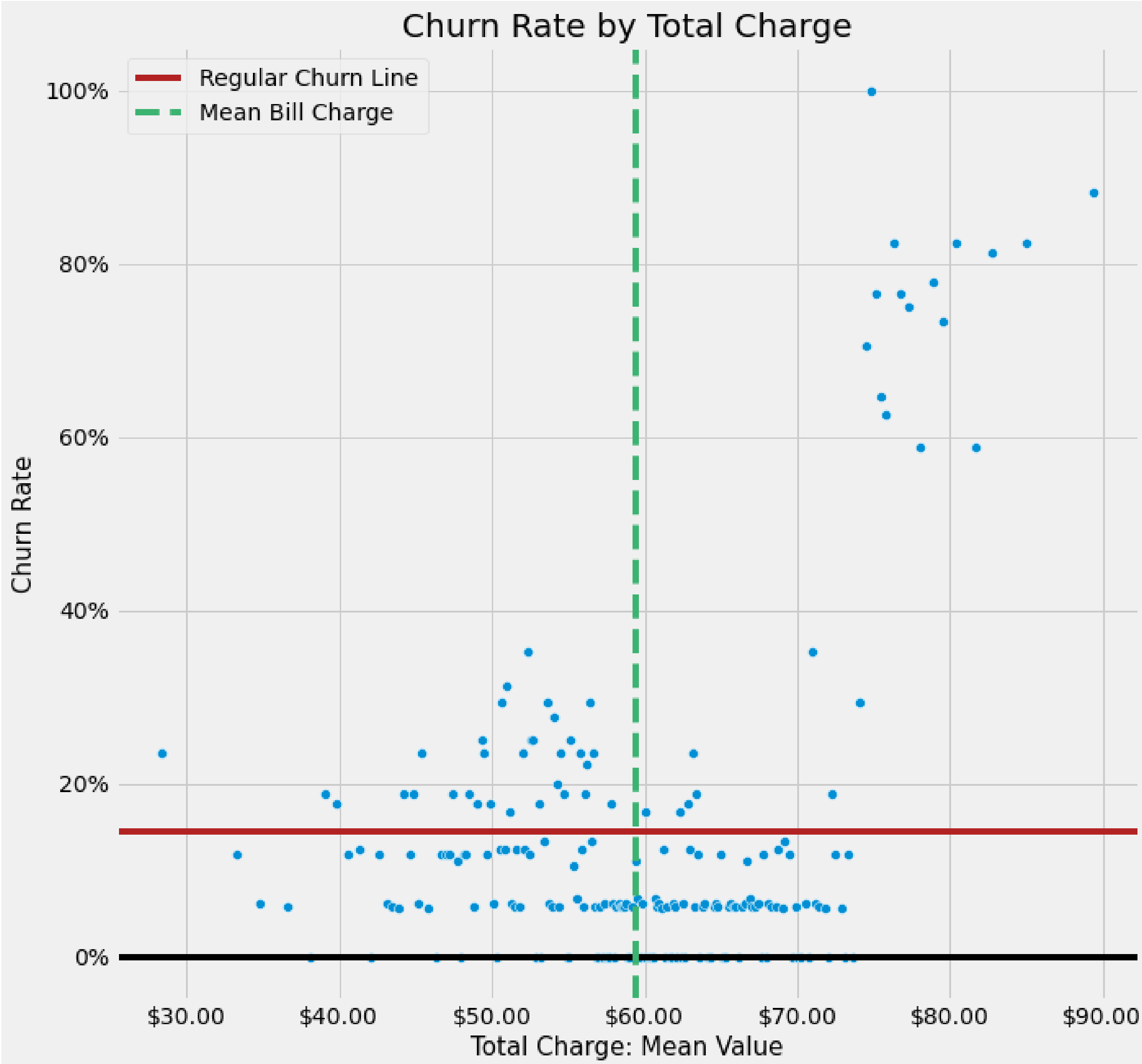
In [67]:
```
charge_df = charge_df.reset_index()
charge_df = charge_df.merge(group_counts, on='charge_group')
charge_df.head()
```

Out[67]:

	charge_group	group_mean	churn	#_of_accounts
0	(22.929, 32.497]	28.372353	0.235294	17
1	(32.497, 33.853]	33.329412	0.117647	17
2	(33.853, 35.72]	34.832500	0.062500	16
3	(35.72, 37.436]	36.645882	0.058824	17

	charge_group	group_mean	churn	#_of_accounts
4	(37.436, 38.752]	38.102941	0.000000	17

In [68]:
```
import matplotlib.ticker as mtick

fig, ax = plt.subplots(figsize=(10,10))
p = sns.scatterplot(x="group_mean", y="churn", data=charge_df);

p.set_xlabel("Total Charge: Mean Value", fontsize = 15)
p.set_ylabel("Churn Rate", fontsize = 15)
ax.yaxis.set_major_formatter(mtick.PercentFormatter(xmax=1, decimals=None, symbo
#p.xaxis.set_major_formatter(display_millions)
ax.xaxis.set_major_formatter('${x:1.2f}')

p.set_title("Churn Rate by Total Charge", fontsize = 20)
plt.figsize=(30,20)

line_1 = plt.axhline(y=.145, color='firebrick')
line_2 = plt.axhline(y=0, color='black')
line_3 = plt.axvline(x=59.45, linestyle='--',color='mediumseagreen')
#line_4 = plt.axvline(x=74.00, color='skyblue')

ax.legend([line_1, line_3], ['Regular Churn Line', 'Mean Bill Charge'])
plt.savefig('images/project_3_total_charge_churn')


plt.show();
```



Analysis

- **Total Charge of $74 per month leads to Churn Rate of roughly 70% or greater!**
- **This affects aprox 240 customers (15 groups of 16)**
- **While there a a good amount of customers above the average churn line, if you add an extra 10%, almost all are within that range until you get to the extreme outliers.**

Voice Mail Plan

In [69]:
```
vm_df = analysis_df[['voice_mail_plan_yes', 'voice_mail_plan_no', 'churn']]
vm_df = vm_df.groupby('churn').sum().reset_index()
vm_df = vm_df.transpose()
vm_df = vm_df.rename(columns={0: "stayed", 1: "churned"})
vm_df['total'] = vm_df.apply(lambda x: x['stayed'] + x['churned'], axis=1)
vm_df['churn_rate'] = vm_df.apply(lambda x: x['churned'] / x['total'], axis=1)
vm_df[1:3]
```

Out[69]:

	stayed	churned	total	churn_rate

	stayed	churned	total	churn_rate
voice_mail_plan_yes	842.0	80.0	922.0	0.086768
voice_mail_plan_no	2008.0	403.0	2411.0	0.167151

Analysis:

- **323 people (27.6% of customers) have a voicemail plan.**
- **Customers that do NOT have a voicemail plan have twice the churn rate of customers that do.**
- **The churn rate for customers without voicemail is slightly higher than the base churn rate, but since the churn rate for customers with voicemail is significantly lower, that gives this good overall significance.**

Conclusions

Questions to Answer: Revisited

What is the Baseline Churn Rate?

- **14.49%**
- **This is the percentage of churn that occured in the dataset I was provided.**

What Factors Contribute to Churn? Which has the biggest impact?

In [70]:
```
feature_importance_df[0:4]
```

Out[70]:

	Feature	Importance
13	cs_calls_high	0.373214
11	international_plan_yes	0.191074
10	total_charge	0.135295
12	voice_mail_plan_yes	0.108922

The 4 factors that have the biggest impact on Churn (in order) are:

1. **Total Amount Charged**
2. **A High Number of Customer Service Calls.**
3. **Customer having an international plan.**
4. **Customer not having a voicemail plan.**

- **All other features have significantly less impact on Churn. (<.05 importance)**

What can be done to identify when a customer is at risk for churn?

Based on my analysis, here is where customers "cross the line" into being at a high risk for churn:

- **Having a Total Charge of $74 or more.**
- **Calling Customer Service 4 or more times.**
- **Having an international plan.**
- **Not Having a Voice Mail plan.**

Recommendations

Recommendation #1: Increased Focus on Customer Service.

- **There is a sharp increase in Churn when a Customer reaches their 4th call to customer service. In order to retain more customers, SyriaTel should focus on resolving whatever issues that customers bring up with Customer Service. If all questions are answered, and issues are explained and addressed, this should lead to happier customers, less customer service calls, and less churn.**
- **Of course, the call itself isn't the issue. Customer service calls are a sign that something is wrong, and the more that a customer calls, the more likely they are to be having problems with the service and/or paying their bills.**
- **I recommend that SyriaTel analyze any data that they have on Customer Service calls to see what issues customers were bringing up and at what frequency. Proactively dealing with these issues will likely cause a decrease in churn.**

Recommendation #2: Take a good look at your international plan and see why it increases the amount of Churn

- **Customers without the international plan are able to make international calls.**
- **Customers with the international plan end up leaving.**
- **I don't have data on how much the international plan costs or how it is used, but it is causing higher churn.**
- **Perhaps it costs too much, or doesn't give an advantage over not having the plan, or is inferior to the competition.**
- **International Minutes and Number of International Calls also have feature importance as well so they should also be investigated.**

Recommendation #3: Offer a Flat Price Model to Combat High Customer Charges

- **Making more money is good, but there is a strong correlation between churn and high charge. This indicates that customers are likely being charged per minute. SyriaTel**

would ultimately make MORE money by RETAINING the customers that they already have.

- **By charging a flat fee, it eliminates any surprise that the customer has, which should result in less customer service calls, and less churn.**
- **The flat fee could be offered in tiers.**
- **The point of this recommendation is that customers know how much their bill is each month, even if they go over on minutes, etc.**

Recommendation #4: Encourage Customers to get a Voice Mail Plan

- **Also, analyze to see why there is such a big difference in churn rate when customers don't have a voice mail plan.**

Recommendation #5: Set up a system which identifies when a customer is getting close to any of the thresholds identified above.

Please Note: These recommenations are based on the way that everything is currently set up. If my other recommendations are followed, many of these issues would already be taken care of.

Green: Low Risk of Customer Churn.

- **0-1 Customer Service Calls.**
- **Customer Bill is $60/month or less.**
- **Customer does not have International Plan.**
- **Customer has Voice Mail Plan**

Yellow: Account is begining to show warning signs of churn.

- **2-3 Customer Service Calls**
- **Customer Bill is above $60/month (the mean value)**

Red: Account is at high risk of churn.

- **4 or more Customer Service Calls**
- **Customer Monthly Bill is at $74 or higher.**
- **Customer has International Plan (in it's current form. See Recommendation #2)**
- **Customer does not have a Voice Mail Plan**

Summary

I was tasked with analyzing the data provided to me by SyriaTel in relation to customers leaving their service. In doing so, I determined that the most important questions to

answer were:

1. **What is the Baseline Churn Rate?**
2. **Which features contribute to churn?**
3. **Which features have the biggest impact on churn?**
4. **What can be done to identify when a customer is at risk for churn?**
5. **What can be done to prevent churn?**

After developing an appropriate model (XGBoost, using GridSearch CV to tune parameters), I was able to determine that the 4 features with the largest impact on customer churn were:

1. **Total Charge**
2. **High Amount of Customer Service Calls**
3. **The presence of an International Plan.**
4. **The presence of a Voice Mail Plan.**

Based on these factors, I made the following recommendations for SyriaTel to implement in order to greatly reduce the amount of churn:

1. **Improve Customer Service: Get to the root of customers' issues and resolve them.**
2. **Take a good look at the international plan that is currently offered and see why customers who have it have such a high rate of churn. Change the plan as necessary to prevent this from happening going forward.**
3. **Change the pricing model from "minutes used" to a flat rate service so that customers will know what to expect to pay each month, while still retaining profit for SyriaTel.**
4. **Incentivize getting a VoiceMail Plan. Also investigate why it helps retain customers.**
5. **Put a system into place for identifying when a customer is at high risk for churn and then be proactive in intervening and helping fix anything that may be leading to churn.**

SyriaTel will always have to deal with churn, but if they deal with the features which have the greatest impact on churn, their average churn rate will be significantly lower. I have also provided some metrics for them to use to better identify when customers are at an increased chance of churn.