Final Project Submission

Please fill out:

- Student name: Tia Plagata
- Student pace: full time
- Scheduled project review date/time:
- Instructor name: Rafael Carrasco
- Blog post URL:

Modeling (Model and iNterpretation)

Modeling Outline

- Metric to use
- Experimenting with a First Model
- Model Selection
- Building a Custom Pipeline
- Model Tuning & GridSearch
- Feature Importance
- Confusion Matrix Analysis

In [1]:

```
# Import Statements
import pandas as pd
import numpy as np

import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BaseEstimator
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassif
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, make_scorer, recall_score

from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline, Pipeline

import matplotlib.pyplot as plt
import seaborn as sns
```

Metric to use: Recall

Recall would be a better metric for this dataset because with churn rate, we can implement more custo
retention strategies, and misidentifying someone as 'exited' and hitting them with a strategy to keep th

Import Training Data

(From saved CSV during EDA)

In [2]:

```
df_train = pd.read_csv('/Users/jordanrjohnson/DataScienceCourseMaterial/phas

df_train.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	...
2682	DC	55	510	354-5058	yes	no	0	106.1	77	18.04	...
3304	IL	71	510	330-7137	yes	no	0	186.1	114	31.64	...
757	UT	112	415	358-5953	no	no	0	115.8	108	19.69	...
2402	NY	77	415	388-9285	no	yes	33	143.0	101	24.31	...
792	NV	69	510	397-6789	yes	yes	33	271.5	98	46.16	...

5 rows × 21 columns

In [3]:

```
df_train.shape
```

Out[3]:

```
(2999, 21)
```

In [4]:

```
df_train.columns
```

Out[4]:

```
Index(['state', 'account length', 'area code', 'phone number',
       'international plan', 'voice mail plan', 'number vmail message
s',
       'total day minutes', 'total day calls', 'total day charge',
       'total eve minutes', 'total eve calls', 'total eve charge',
       'total night minutes', 'total night calls', 'total night charg
e',
       'total intl minutes', 'total intl calls', 'total intl charge',
       'customer service calls', 'churn'],
      dtype='object')
```

In [5]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2999 entries, 2682 to 1061
Data columns (total 21 columns):
state                     2999 non-null object
account length            2999 non-null int64
area code                 2999 non-null int64
phone number              2999 non-null object
international plan        2999 non-null object
voice mail plan           2999 non-null object
number vmail messages     2999 non-null int64
total day minutes         2999 non-null float64
total day calls           2999 non-null int64
total day charge          2999 non-null float64
total eve minutes         2999 non-null float64
total eve calls           2999 non-null int64
total eve charge          2999 non-null float64
total night minutes       2999 non-null float64
total night calls         2999 non-null int64
total night charge        2999 non-null float64
total intl minutes        2999 non-null float64
total intl calls          2999 non-null int64
total intl charge         2999 non-null float64
customer service calls    2999 non-null int64
churn                     2999 non-null bool
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 495.0+ KB
```

In [6]:

```
df_train['churn'].value_counts()
```

Out[6]:

```
False    2569
True      430
Name: churn, dtype: int64
```

Build an Initial Model

AKA sanity check. This model is only built to see if it's possible to build a model using this dataset.

In [7]:

```
# Functions
def transform_df(df):

    """
    Transforms yes and no values in certain columns of the df to 1s and 0s,
    Returns the dataframe.
    """

    df['international plan'] = df['international plan'].apply(lambda x: 1 if
    df['voice mail plan'] = df['voice mail plan'].apply(lambda x: 1 if x.low

    return df


def plot_conf_matrix(y_true, y_pred):

    """
    Plots a prettier confusion matrix than matplotlib.
    """

    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, cmap=sns.color_palette('Blues_d'), fmt='0.5g
    plt.xlabel('Predictions')
    plt.ylabel('Actuals')
    plt.ylim([0,2])
    plt.show()
```

In [8]:

```
features_to_use = ['account length', 'international plan', 'voice mail plan'
                   'total day charge', 'total eve charge', 'total night char
                   'customer service calls']
target = ['churn']
```

In [9]:

```
df_train_transformed = transform_df(df_train)
df_train_transformed.head()
```

Out[9]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
2682	DC	55	510	354-5058	1	0	0	106.1	77	18.04	...
3304	IL	71	510	330-7137	1	0	0	186.1	114	31.64	...
757	UT	112	415	358-5953	0	0	0	115.8	108	19.69	...
2402	NY	77	415	388-9285	0	1	33	143.0	101	24.31	...
792	NV	69	510	397-6789	1	1	33	271.5	98	46.16	...

5 rows × 21 columns

In [10]:

```
X = df_train_transformed[features_to_use]
y = df_train_transformed[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, ran
X_train.shape, X_test.shape
```

Out[10]:

```
((2249, 9), (750, 9))
```

In [11]:

```
# Use Smote to resample and fix the class imbalance problem
smote = SMOTE()
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)
```

In [12]:

```
rf1 = RandomForestClassifier()
rf1.fit(X_train_resampled, y_train_resampled)
```

Out[12]:

```
RandomForestClassifier()
```

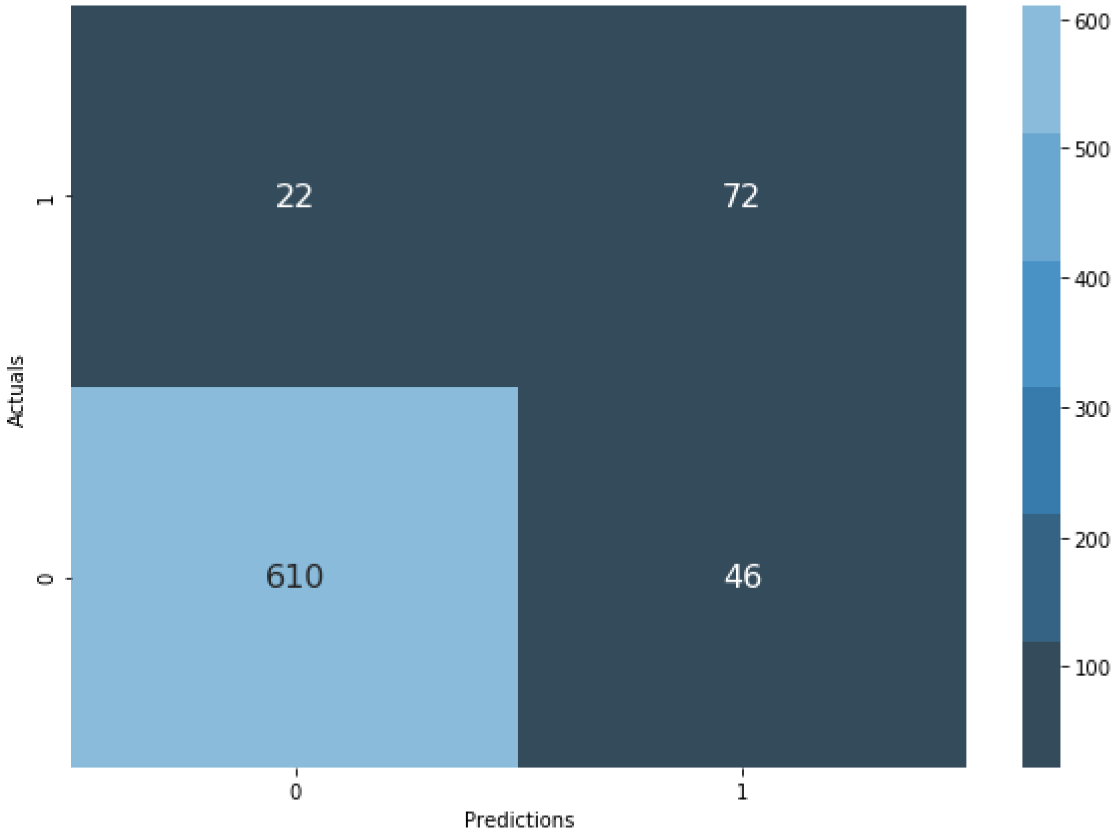
In [13]:

```
y_preds_test = rf1.predict(X_test)
y_preds_train = rf1.predict(X_train_resampled)

print('Training Recall:', recall_score(y_train_resampled, y_preds_train))
print('Testing Recall:', recall_score(y_test, y_preds_test))
```

```
Training Recall: 1.0
Testing Recall: 0.7659574468085106
```

In [14]:

```
plot_conf_matrix(y_test, y_preds_test)
# We want to reduce that 25 because those are our False Negatives (people wh
```



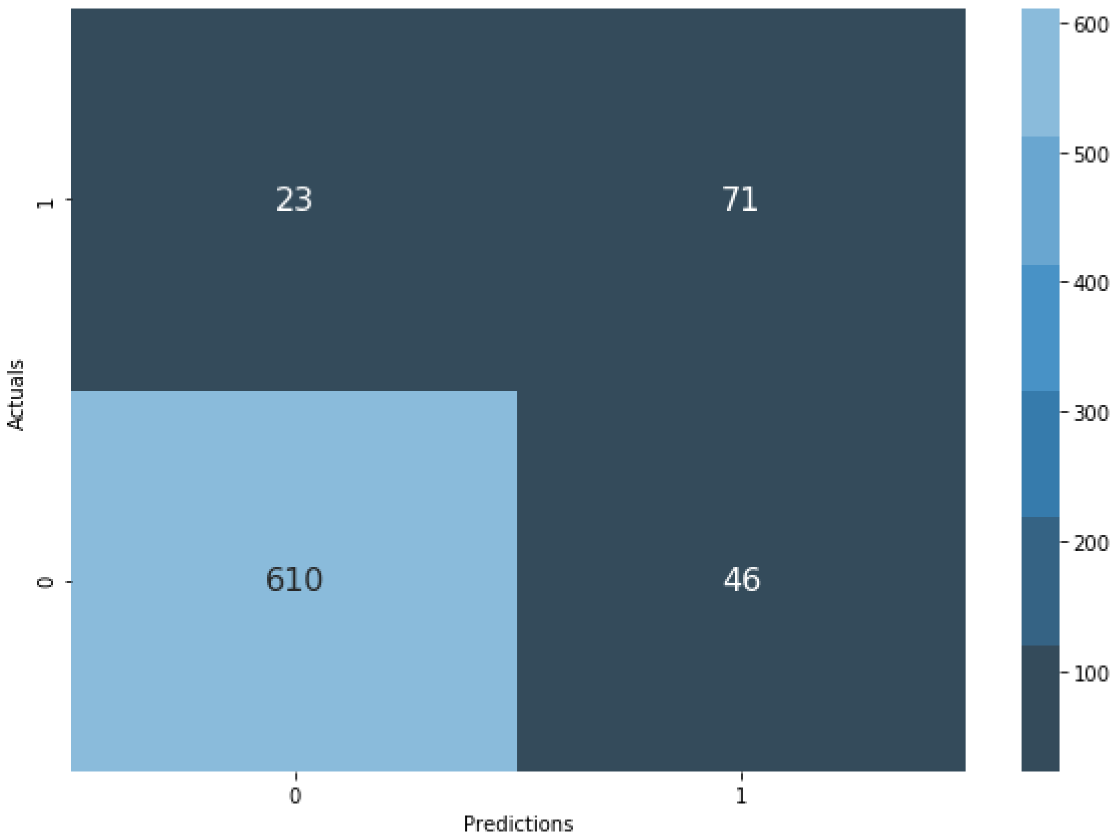
Model Selection:

Perform Test to Select Best Classifier

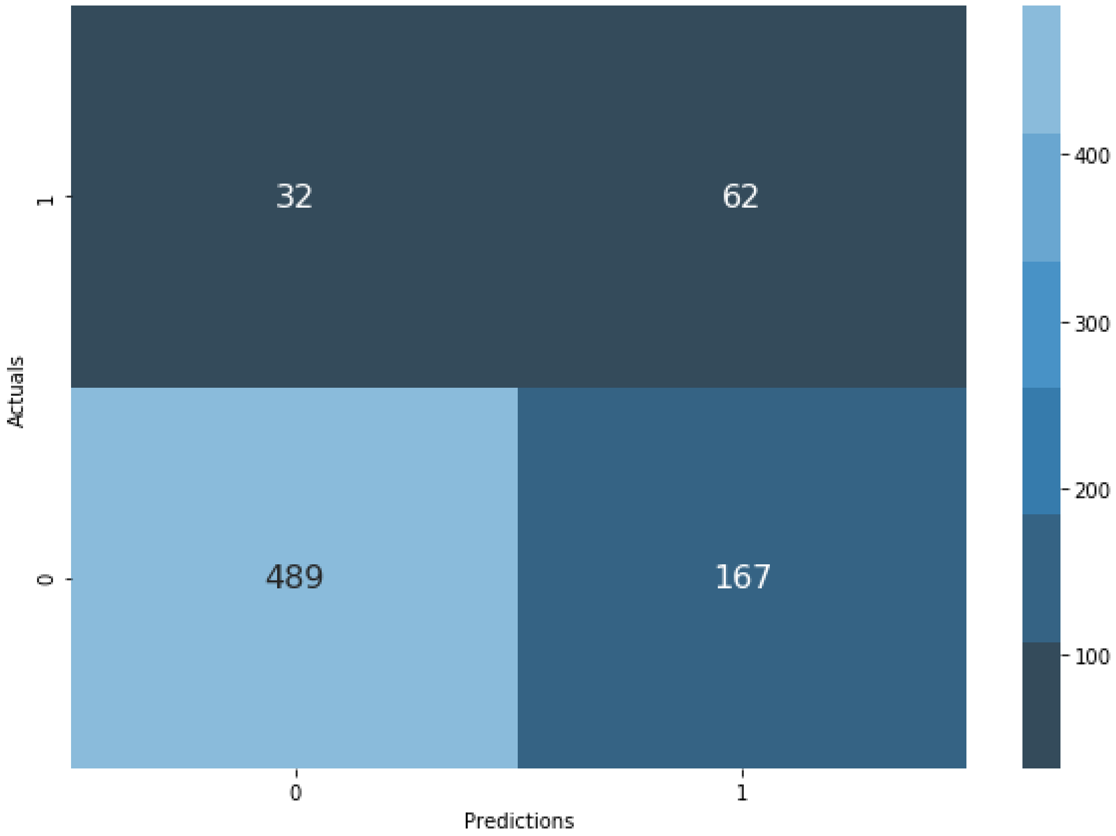
In [15]:

```
rf = RandomForestClassifier()
knn = KNeighborsClassifier()
gboost = GradientBoostingClassifier()
gbayes = GaussianNB()
svm = SVC()

models = [rf, knn, gboost, gbayes, svm]

for model in models:
    model.fit(X_train_resampled, y_train_resampled)
    y_preds_test = model.predict(X_test)
    y_preds_train = model.predict(X_train_resampled)
    print('Model:', model)
    print('Training Recall:', recall_score(y_train_resampled, y_preds_train)
    print('Testing Recall:', recall_score(y_test, y_preds_test))
    plot_conf_matrix(y_test, y_preds_test)
    print('\n ---------------------------------- \n')
```

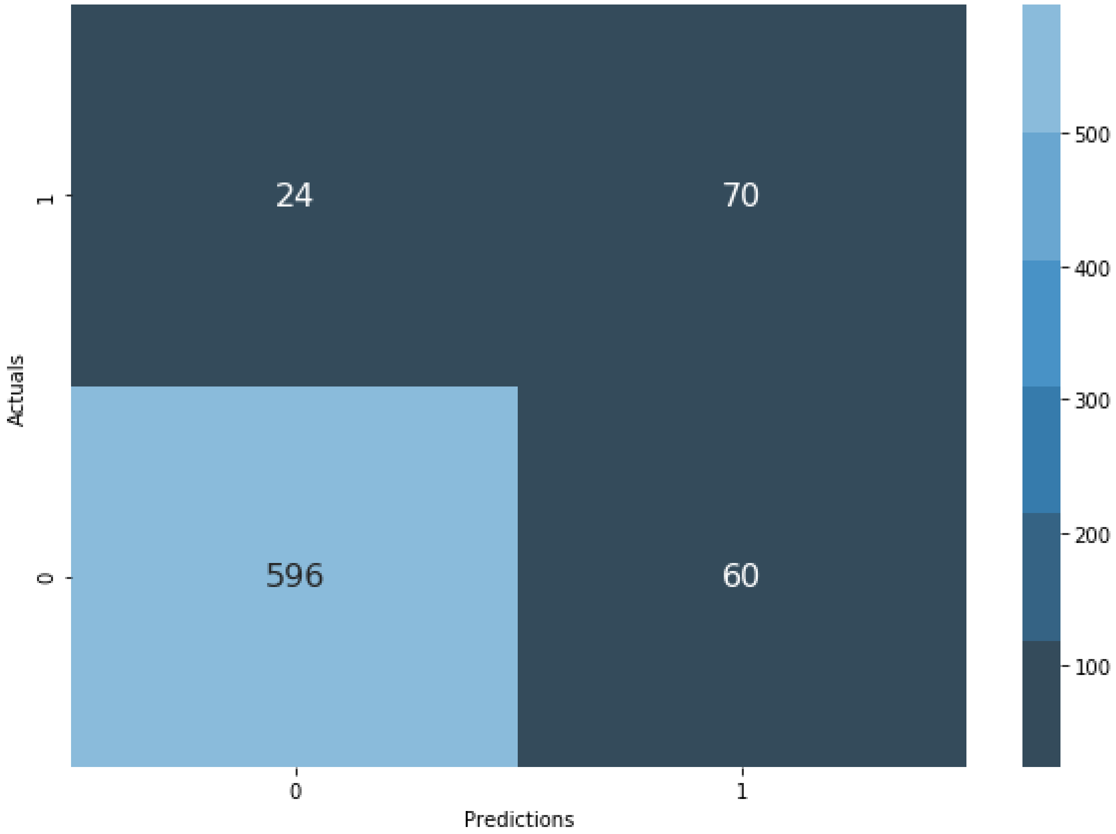
```
Model: RandomForestClassifier()
Training Recall: 1.0
Testing Recall: 0.7553191489361702
```



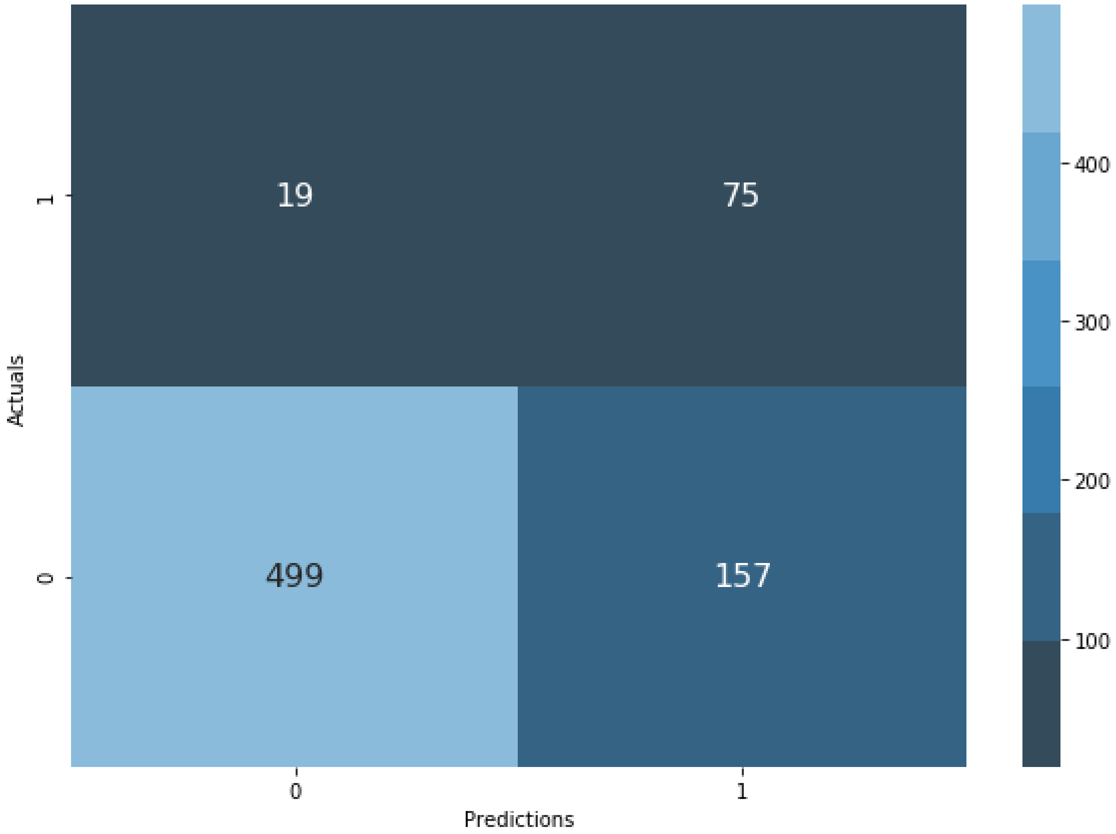
```
----------------------------------

Model: KNeighborsClassifier()
Training Recall: 0.9759539989545217
Testing Recall: 0.6595744680851063
```



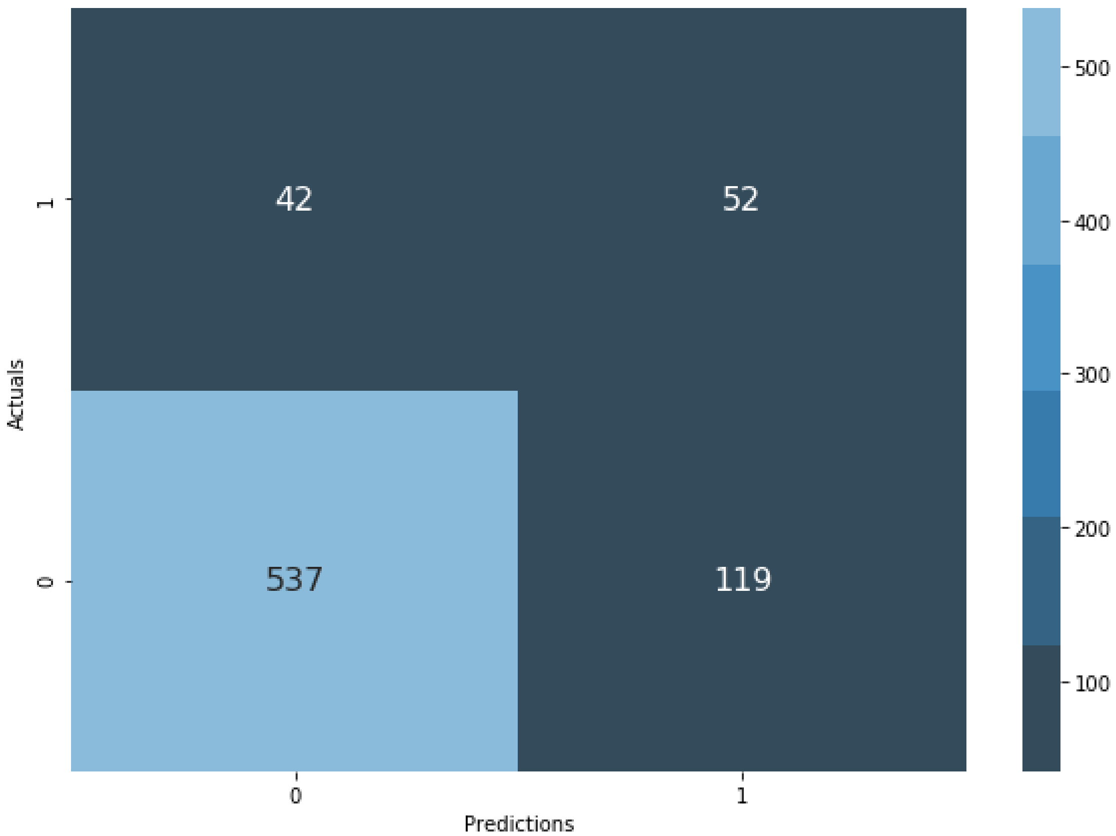

```
----------------------------------

Model: GradientBoostingClassifier()
Training Recall: 0.7731312075274438
Testing Recall: 0.7446808510638298
```




```
 --------------------------------

Model: GaussianNB()
Training Recall: 0.7647673810768426
Testing Recall: 0.7978723404255319
```




```
----------------------------------

Model: SVC()
Training Recall: 0.48980658651332987
Testing Recall: 0.5531914893617021
```



```
------------------------------------
```

Notes:

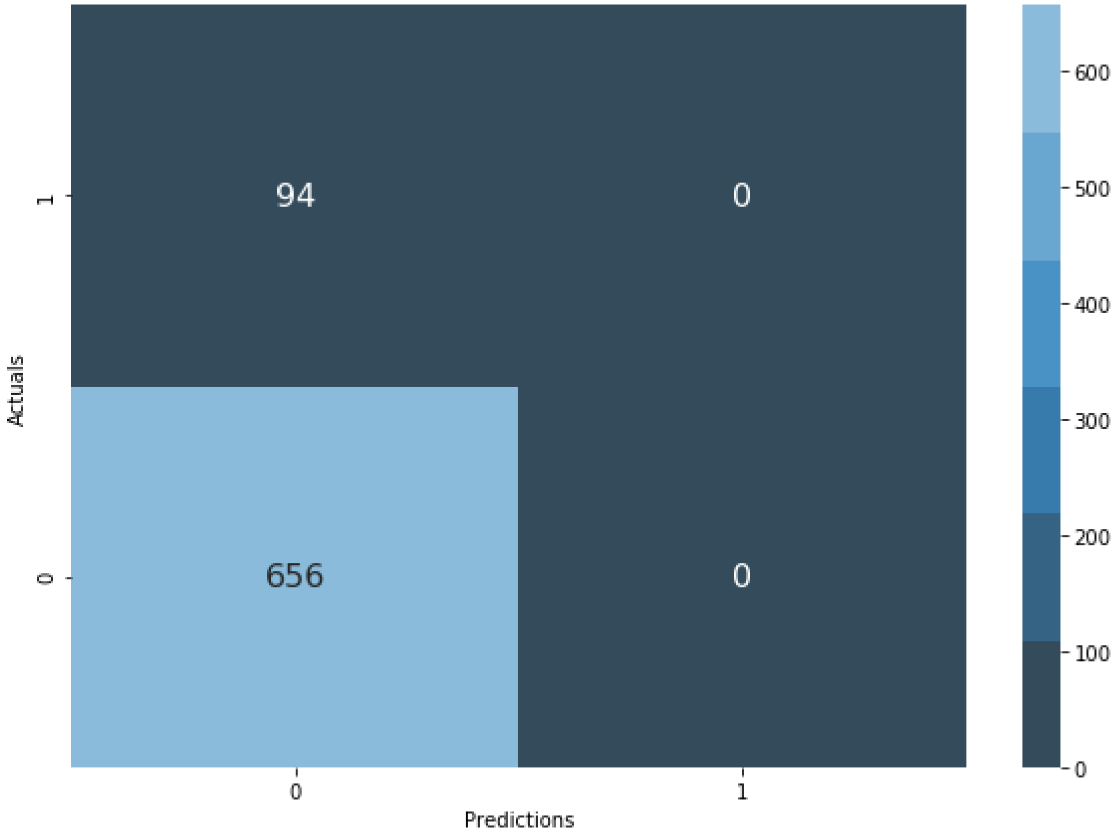
The best performing classifiers here were Gradient Boost, and Gaussian Naive Bayes. They both had t lowest False Negatives and the least overfitting. I would like to redo the KNN model with a scaler to se helps it perform better (distance is very much affected by the scales of the features).

It is also important to note that I did not do any hypertuning yet, nor feature engineering. I am only che initally which model might do the best with this dataset.

In [16]:

```
# Redo the KNN model with feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_resampled)
X_test_scaled = scaler.transform(X_test)

knn.fit(X_train_scaled, y_train_resampled)

y_preds_test = model.predict(X_test_scaled)
y_preds_train = model.predict(X_train_scaled)
print('New KNN Model:')
print('Training Recall:', recall_score(y_train_resampled, y_preds_train))
print('Testing Recall:', recall_score(y_test, y_preds_test))
plot_conf_matrix(y_test, y_preds_test)
```

```
New KNN Model:
Training Recall: 0.0
Testing Recall: 0.0
```



Not good. It just predicted everything as not churning. Therefore, I will proceed with Gradient Boosting

Set up the Pipeline

Steps

1) Choose Columns (need to build a class for this)

2) Transform categorical columns and build features (need a custom class for this too)

3) Deal with class imbalance using SMOTE (imblearn's pipeline will only apply this step to training set)

4) Build model and predict (use existing sklearn methods)

In [17]:

```
# Functions to build features

def categorize_state(state):
    if state in ['AK', 'AZ', 'DC', 'HI', 'IA', 'IL', 'LA', 'NE', 'NM', 'RI',
        state = 1
    elif state in ['AL', 'CO', 'FL', 'ID', 'IN', 'KY', 'MO', 'NC', 'ND', 'NH
        state = 2
    elif state in ['CT', 'DE', 'GA', 'KS', 'MA', 'MN', 'MS', 'MT', 'NV', 'NY
        state = 3
    else:
        state = 4
    return state


def build_features(X):
    X['total charge'] = X['total day charge'] + X['total eve charge'] + X['t
    X['total minutes'] = X['total day minutes'] + X['total eve minutes'] + X
    X['total calls'] = X['total day calls'] + X['total eve calls'] + X['tota
    X['avg minutes per domestic call'] = (X['total minutes'] - X['total intl
    X['competition'] = X['state'].apply(categorize_state)
    return X
```

In [18]:

```
# Build custom classes to build into the pipeline

class SelectColumnsTransformer(BaseEstimator):

    def __init__(self, columns=None):
        self.columns = columns

    def transform(self, X, **transform_params):
        cpy_df = X[self.columns].copy()
        return cpy_df

    def fit(self, X, y=None, **fit_params):
        return self


class Transform_Categorical(BaseEstimator):

    def transform(self, X, y=None, **transform_params):
        try:
            X['international plan'] = X['international plan'].apply(self.yes_
            X['voice mail plan'] = X['voice mail plan'].apply(self.yes_no_fu
        except:
            pass
        return X

    def fit(self, X, y=None, **fit_params):
        return self

    @staticmethod
    def yes_no_func(x):
        return 1 if x.lower() == 'yes' else 0
```

In [19]:

```
df_with_features = build_features(df_train)
df_with_features.head()
```

Out[19]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
2682	DC	55	510	354-5058	1	0	0	106.1	77	18.04	...
3304	IL	71	510	330-7137	1	0	0	186.1	114	31.64	...
757	UT	112	415	358-5953	0	0	0	115.8	108	19.69	...
2402	NY	77	415	388-9285	0	1	33	143.0	101	24.31	...
792	NV	69	510	397-6789	1	1	33	271.5	98	46.16	...

5 rows × 26 columns

In [20]:

```
features_to_use = ['account length', 'international plan', 'voice mail plan'
                   'total charge', 'customer service calls', 'competition',
                   'avg minutes per domestic call', 'total calls', 'total mi
target = ['churn']
```

In [21]:

```
pipeline = Pipeline(steps= [
                    ("ColumnTransformer", SelectColumnsTransformer(columns=f
                    ("TransformCategorical", Transform_Categorical()),
                    ("SMOTE", SMOTE()),
                    ("GradientBooster", GradientBoostingClassifier())
                    ])
```

In [22]:

```
X_train = df_with_features.drop(columns=['churn'])
y_train = df_with_features[target]
```

In [23]:

```
pipeline.fit(X_train, y_train)
```

Out[23]:

```
Pipeline(steps=[('ColumnTransformer',
                 SelectColumnsTransformer(columns=['account length',
                                                   'international pla
n',
                                                   'voice mail plan',
                                                   'number vmail messa
ges',
                                                   'total charge',
                                                   'customer service c
alls',
                                                   'competition',
                                                   'avg minutes per do
mestic '
                                                   'call',
                                                   'total calls',
                                                   'total minutes'])),
                ('TransformCategorical', Transform_Categorical()),
                ('SMOTE', SMOTE()),
                ('GradientBooster', GradientBoostingClassifier())])
```

In [24]:

```
# Bring in validation set to test
df_validation = pd.read_csv('/Users/jordanrjohnson/DataScienceCourseMaterial
df_validation.head()
```

Out[24]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...
2360	IN	68	415	386-9724	no	no	0	222.1	107	37.76	...
600	MI	102	510	336-4656	no	no	0	102.6	89	17.44	...
1501	AZ	72	510	407-9830	no	no	0	272.4	88	46.31	...
1114	TN	108	408	352-1127	no	yes	15	165.1	85	28.07	...
517	OK	52	408	389-4780	no	no	0	214.7	68	36.50	...

5 rows × 21 columns

In [25]:

```
df_valid_transformed = build_features(df_validation)
X_valid = df_valid_transformed.drop(columns='churn')
y_valid = df_valid_transformed['churn']
```

In [26]:

```
pipeline.score(X_valid, y_valid)
```

Out[26]:

```
0.9131736526946108
```

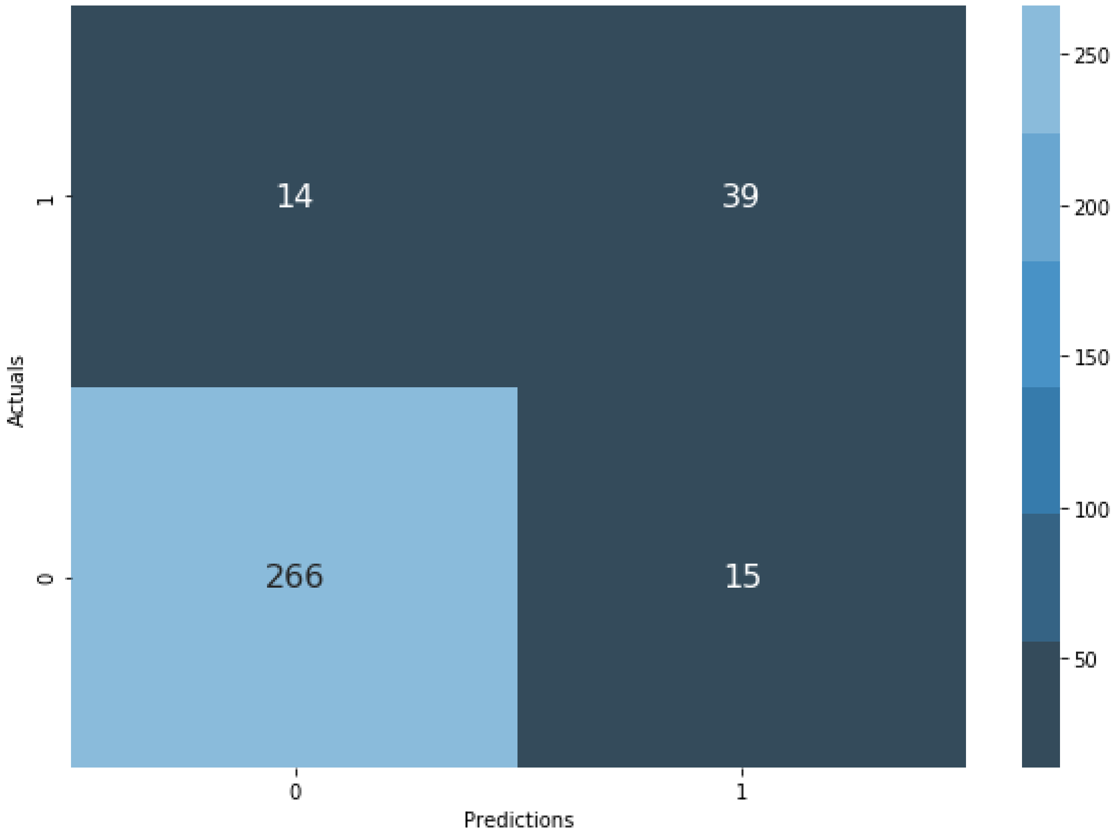
In [27]:

```
y_preds = pipeline.predict(X_valid)
```

In [28]:

```
print(recall_score(y_valid, y_preds))
print('Confusion Matrix for Model Before Tuning')
plot_conf_matrix(y_valid, y_preds)
```

```
0.7358490566037735
Confusion Matrix for Model Before Tuning
```



Model Tuning & GridSearch

In [29]:

```
param_grid = {
            "ColumnTransformer__columns": [['account length', 'international
                                            'number vmail messages', 'total d
                                            'total day charge', 'total eve mi
                                            'total eve charge', 'total night :
                                            'total night charge', 'total intl
                                            'total intl charge', 'customer se
                                           ['account length', 'international :
                                            'number vmail messages', 'total d
                                            'total day charge', 'total eve mi
                                            'total eve charge', 'total night :
                                            'total night charge', 'total intl
                                            'total intl charge', 'customer se
                                            'total minutes', 'total calls', '
                                            'competition']],
        "SMOTE__sampling_strategy": [1],
        "GradientBooster__loss": ['deviance', 'exponential'],
        "GradientBooster__n_estimators": [100, 150],
        "GradientBooster__max_depth": [3, 5],
        "GradientBooster__max_features": ['auto', 8, None]
}
```

In [199]:

```
# Warning! This takes a long time to run!
gs_pipeline = GridSearchCV(pipeline, param_grid=param_grid, verbose=2, scori
gs_pipeline.fit(X_train, y_train)
```

...

In [200]:

```
gs_pipeline.best_estimator_
```

Out[200]:

```
Pipeline(steps=[('ColumnTransformer',
                 SelectColumnsTransformer(columns=['account length',
                                                   'international pla
n',
                                                   'voice mail plan',
                                                   'number vmail messa
ges',
                                                   'total day minute
s',
                                                   'total day calls',
                                                   'total day charge',
                                                   'total eve minute
s',
                                                   'total eve calls',
                                                   'total eve charge',
                                                   'total night minute
s',
                                                   'total night call
s',
                                                   'total night charg
e',
                                                   'total intl minute
s',
                                                   'total intl calls',
                                                   'total intl charg
e',
                                                   'customer service c
alls',
                                                   'total charge',
                                                   'total minutes',
                                                   'total calls',
                                                   'avg minutes per do
mestic '
                                                   'call',
                                                   'competition'])),
                ('TransformCategorical', Transform_Categorical()),
                ('SMOTE', SMOTE(sampling_strategy=1)),
                ('GradientBooster', GradientBoostingClassifier(max_dep
th=5))])
```

In [201]:

```
gs_pipeline.best_params_
```

Out[201]:

```
{'ColumnTransformer__columns': ['account length',
  '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',
  'total charge',
  'total minutes',
  'total calls',
  'avg minutes per domestic call',
  'competition'],
 'GradientBooster__loss': 'deviance',
 'GradientBooster__max_depth': 5,
 'GradientBooster__max_features': None,
 'GradientBooster__n_estimators': 100,
 'SMOTE__sampling_strategy': 1}
```

In [202]:

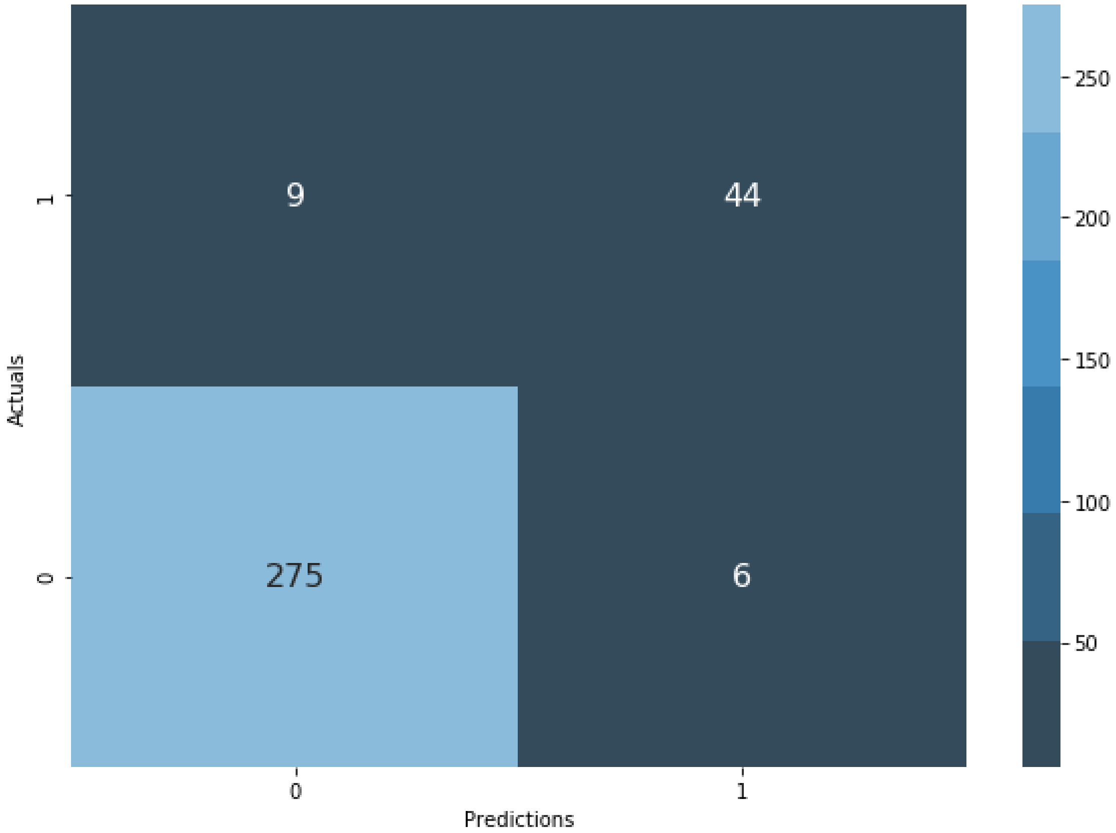
```
best_model = gs_pipeline.best_estimator_
y_validation_preds = best_model.predict(X_valid)
recall_score(y_valid, y_validation_preds)
```

Out[202]:

```
0.8301886792452831
```

In [203]:

```
plot_conf_matrix(y_valid, y_validation_preds)
```



Feature Importance

In [38]:

```
def plot_feature_importances(X, model):
    features = X.columns
    feat_imp_scores = model.feature_importances_
    plt.figure(figsize=(10, 8))
    plt.bar(features, feat_imp_scores, zorder=2, alpha=0.8)
    plt.grid(zorder=0)
    plt.xticks(rotation=90)
    plt.xlabel('Feature Importance')
    plt.ylabel('Features')
    plt.title('Feature Importances of Model')
    plt.show()
```

In [229]:

```
best_model.steps[3][1].feature_importances_
```

Out[229]:

```
array([0.01044691, 0.05775256, 0.06502698, 0.06725046, 0.0081423 ,
       0.009758  , 0.00580963, 0.01031463, 0.0136483 , 0.00922068,
       0.01039439, 0.01047657, 0.01041452, 0.03448355, 0.05809022,
       0.02795012, 0.16297136, 0.38006398, 0.0145925 , 0.00951009,
       0.01522518, 0.00845706])
```

In [230]:

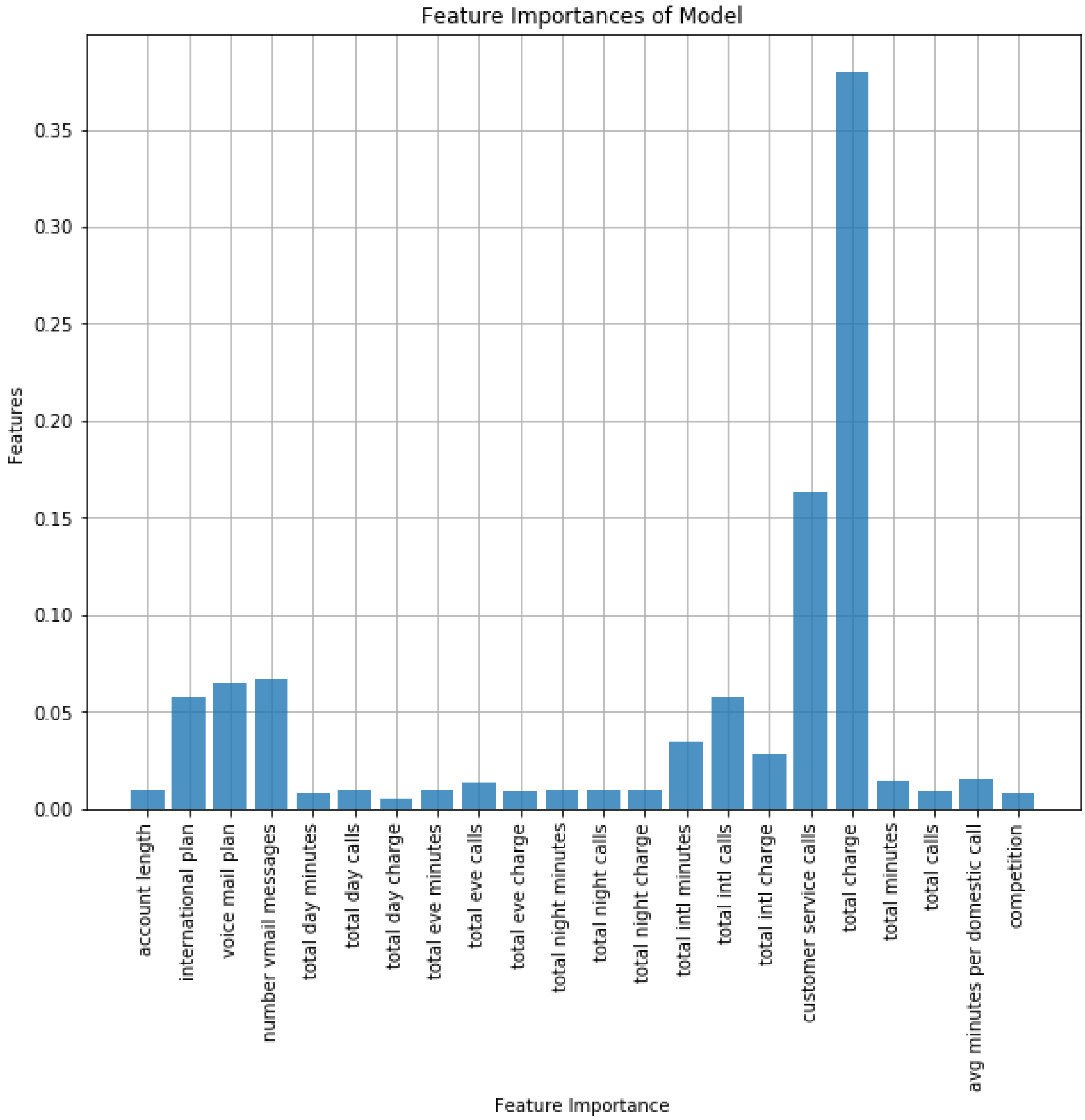
```
best_model.steps[0][1].columns
```

Out[230]:

```
['account length',
 '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',
 'total charge',
 'total minutes',
 'total calls',
 'avg minutes per domestic call',
 'competition']
```

In [231]:

```
plot_feature_importances(X=best_model.steps[0][1], model=best_model.steps[3]
```



New Model with Most Important Features

I will train a new model with just the top 8 features and test if these perform better or similar to our pre model to potentially save resources.

In [30]:

```
param_grid = {
            "ColumnTransformer__columns": [['total charge', 'customer servic
                                            'voice mail plan', 'international 
                                             'total intl minutes', 'total intl
        "SMOTE__sampling_strategy": [1],
        "GradientBooster__n_estimators": [100, 150],
        "GradientBooster__max_depth": [3, 5],
        "GradientBooster__max_features": [None]
}
```

In [31]:

```
gs_pipeline = GridSearchCV(pipeline, param_grid=param_grid, verbose=2, scori
gs_pipeline.fit(X_train, y_train)
```

...

In [32]:

```
gs_pipeline.best_params_
```

Out[32]:

```
{'ColumnTransformer__columns': ['total charge',
  'customer service calls',
  'number vmail messages',
  'voice mail plan',
  'international plan',
  'total intl calls',
  'total intl minutes',
  'total intl charge'],
 'GradientBooster__max_depth': 3,
 'GradientBooster__max_features': None,
 'GradientBooster__n_estimators': 100,
 'SMOTE__sampling_strategy': 1}
```

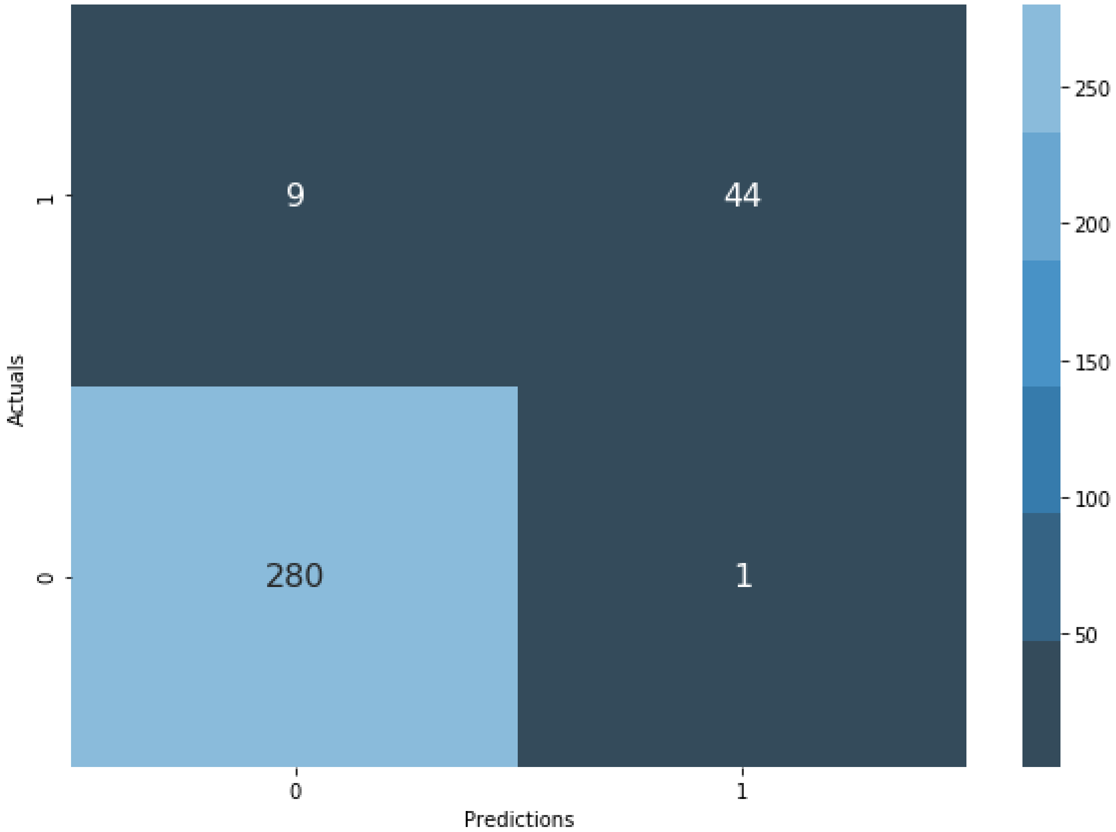
In [33]:

```
best_model = gs_pipeline.best_estimator_
y_validation_preds = best_model.predict(X_valid)
```

In [48]:

```
print('Final Testing Recall:', recall_score(y_valid, y_validation_preds))
plot_conf_matrix(y_valid, y_validation_preds)
```

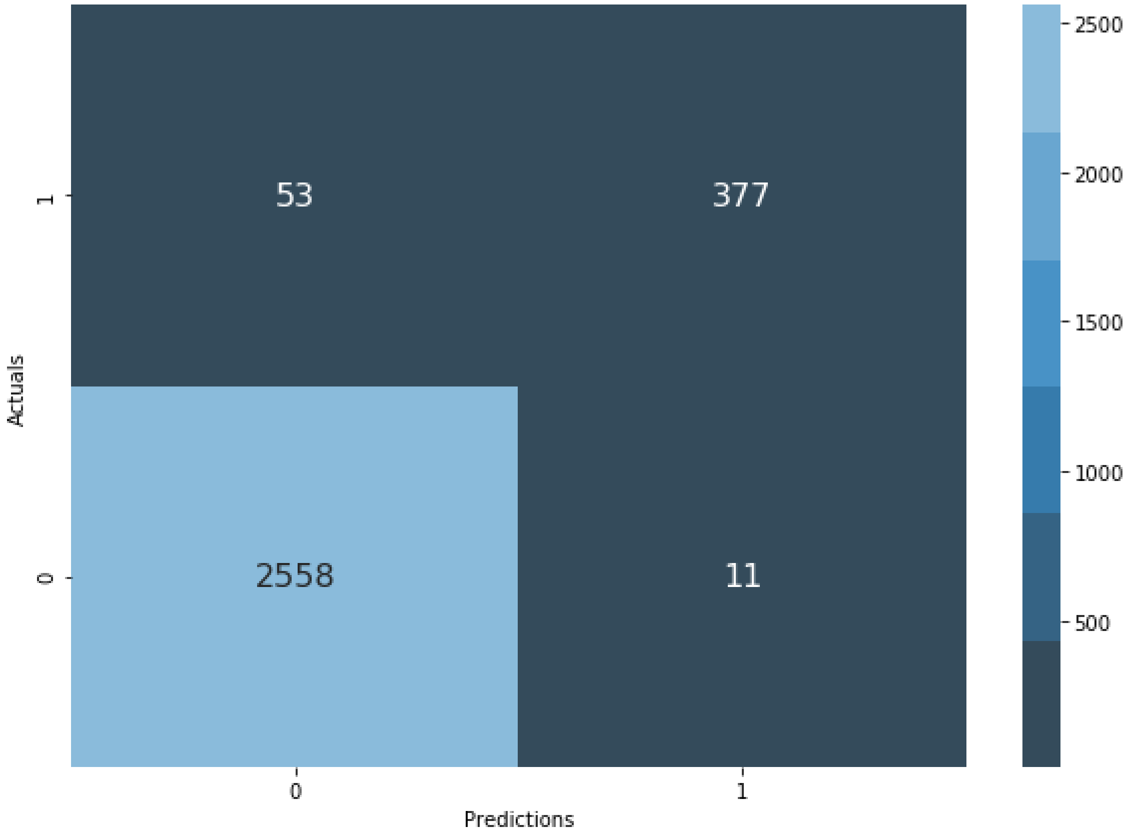
```
Final Testing Recall: 0.8301886792452831
```



In [46]:

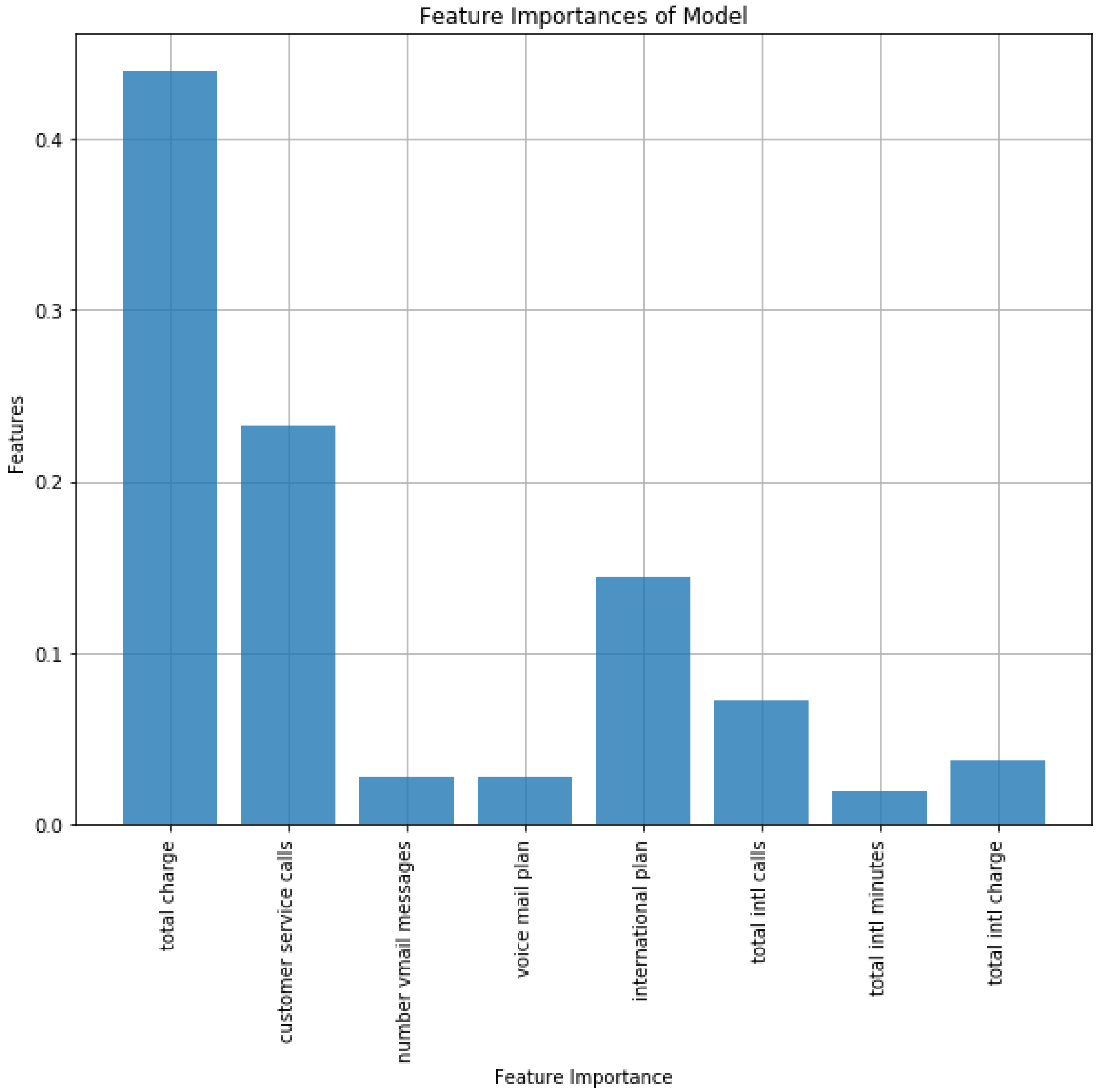
```
y_training_preds = best_model.predict(X_train)
print('Final Training Recall', recall_score(y_train, y_training_preds))
plot_conf_matrix(y_train, y_training_preds)
```

```
Final Training Recall 0.8767441860465116
```



In [39]:

```
plot_feature_importances(X=best_model.steps[0][1], model=best_model.steps[3]
```



Confusion Matrix & Cost Benefit Analysis

Let's say the cost of a False Positive is having to give a customer a discount of 50% off one month of t service when they were not actually going to churn. For this analysis we will say that the **cost of a FP = USD per customer (-25)**.

Alternatively, the cost of a False Negative is losing that customer (and their monthly payment of 50 USD having to go out and get a new customer (customer acqusition cost of 50 USD). Therefore, we will say **cost of a FN = 100 USD per customer (-100)**.

The benefit of a True Positive is keeping that customer on and having them continue paying their 50 US monthly payment, minus the 50% discount. **Benefit of TP = 25**

The **benefit of a True Negative = 0** since they were not going to churn and we predicted that, so we d offer any discounts.

These costs and benefits are reflected in the function below.

In [40]:

```
def cost_benefit_analysis(model, X_test, y_test):
    y_preds = model.predict(X_test)
    label_dict = {"TP":0, "FP": 0, "TN": 0, "FN": 0}
    for yt, yp in zip(y_test, y_preds):
        if yt==yp:
            if yt==1:
                label_dict["TP"] += 1
            else:
                label_dict["TN"] += 1
        else:
            if yp==1:
                label_dict["FP"] += 1
            else:
                label_dict["FN"] += 1
    cb_dict = {"TP": 25, "FP": -25, "TN": 0, "FN": -100}

    total = 0
    for key in label_dict.keys():
        total += cb_dict[key]*label_dict[key]
    return cb_dict, label_dict, total / sum(label_dict.values())
```

In [41]:

```
cb_dict, label_dict, expected_value = cost_benefit_analysis(best_model, X_va
```

In [42]:

```
print(cb_dict, label_dict)
```

```
{'TP': 25, 'FP': -25, 'TN': 0, 'FN': -100} {'TP': 44, 'FP': 1, 'TN': 2
80, 'FN': 9}
```

In [43]:

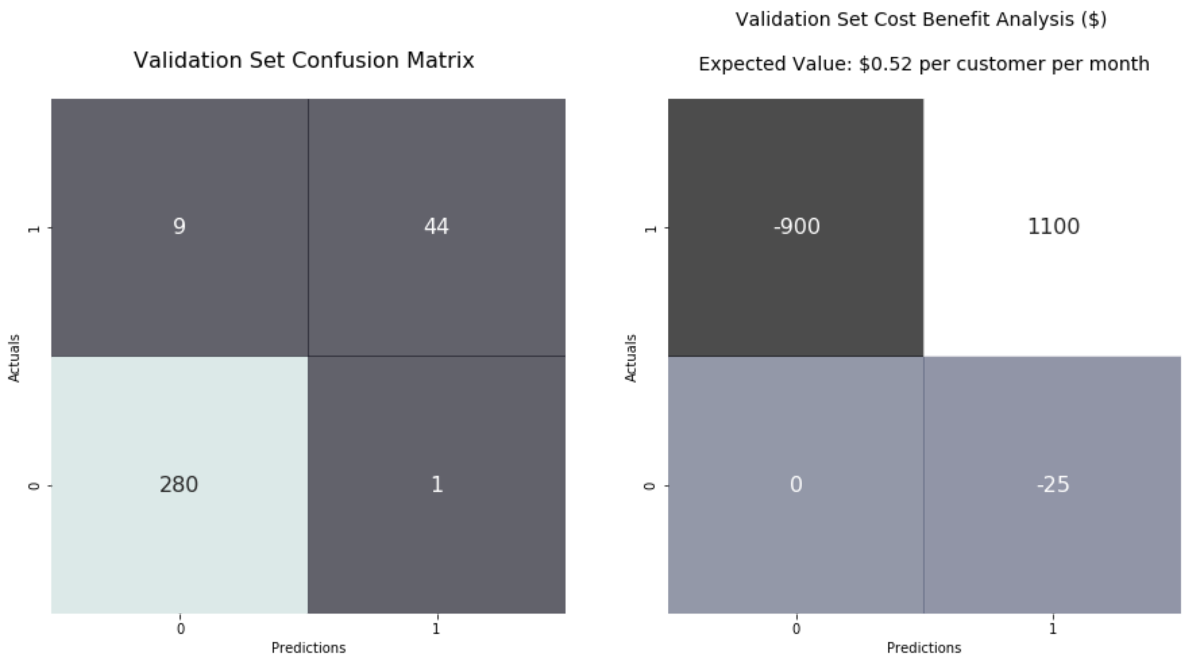
```
# Put the cost benefit values in an array to plot
cb_array = [[cb_dict['TN']*label_dict['TN'],
            cb_dict['FP']*label_dict['FP']],
            [cb_dict['FN']*label_dict['FN'],
            cb_dict['TP']*label_dict['TP']]]
cb_array
```

Out[43]:

```
[[0, -25], [-900, 1100]]
```

In [49]:

```
cm = confusion_matrix(y_valid, y_validation_preds)

fig, axes = plt.subplots(1, 2, figsize=(15, 7))

sns.heatmap(cm, annot=True, cmap=sns.color_palette('bone'), fmt='0.5g', cbar
            annot_kws={'size': 16}, alpha=.7, ax=axes[0])

sns.heatmap(cb_array, annot=True, fmt='0.5g', cmap='bone', cbar=False,
            annot_kws={'size': 16}, alpha=.7, ax=axes[1])

plt.xlabel('Predictions')
plt.ylabel('Actuals')
axes[0].set_ylabel('Actuals')
axes[0].set_xlabel('Predictions')
axes[0].set_ylim([0,2])
axes[1].set_ylim([0,2])
axes[0].set_title('Validation Set Confusion Matrix \n', fontdict={'size': 16
axes[1].set_title(f'Validation Set Cost Benefit Analysis ($) \n\n Expected V
          fontdict={'size': 14})
plt.show()
```



Based on this cost benefit analysis, our expected value from this strategy is 52 cents per customer pe
That may not seem like much, but for millions of customers it would add up. The good news here is th
this model predicting churn, we are not LOSING money! We can see the breakdown of each cost and
multiplied by the number of TP, TN, FP, FNs on the confusion matrix above.

Conclusion

The final model had the following recall scores:

In [144]:

```
print('Validation Recall Score', round(recall_score(y_valid, y_validation_pr
print('Training Recall Score', round(recall_score(y_train, y_training_preds)
```

```
Validation Recall Score 0.83
Training Recall Score 0.88
```

Since these recall scores are so close, we can assume the model is slightly overfit, but overall very goc recall. This model produced only 9 (2%) false negatives for the validation set. It produced only 1 (0.003 positive from the validation set, but if our customer retention strategy is to keep these customers enga not a bad thing to keep a customer engaged who is mispredicted as potentially exiting.

Finally, based on the cost benefit analysis of this model's predictive ability and the SyriaTel Communic strategy for customer retention, the expected value is 52 cents per customer per month. Over the cour year and millions of customers nationwide, we can conclude that this strategy would make us a lot of r the long run.

In [ ]:

```

```