1 Overview

This notebook represents efforts to predict trends in the telecommunication data that was investigated in the EDA notebook. For initial context, it is recommended to first read the README file. It is helpful to scan through the EDA first before this notebook as well to understand some of the strategies implemented in this notebook.

The problem: SyriaTel (a telecommunications company in Syria) is using a US-based telecom dataset surrounding the idea of customer churn in order to decrease losses involved in losing customers. Main considerations for this problem:

- Business consideration: Identify customers that are likely to churn and implement a strategy, such as discounted rates for x number of months, to retain these customers.
- Evaluation consideration: More money will be lost on a False Negative (customer that is predicted to be retained but actually churned) than on a False Positive (customer that is predicted to churn but actually is retained) because it is assumed that it is more costly to replace the False Negative with a new customer and lose the stream of income from this churned customer than it is to implement a retention strategy (i.e. discounted rate) on a customer that does not need it and simply lose the difference between what they would have paid versus the discounted payment stream. For this reason, an F-2 Score will be used to evaluate model success because this metric penalizes False Negatives more than False Positives.
- It should also be noted that F-2 Score is used instead of another metric like recall because F-2 will penalize both FPs and FNs, which is important because both of these cases negatively affect the company.

2 Modeling Preparation

2.1 Imports and Some Dtype Conversions

In [1]:
```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn

%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression

from imblearn.over_sampling import SMOTE

from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn import tree
from sklearn.svm import SVC
from sklearn import svm

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import fbeta_score, accuracy_score, make_scorer
from sklearn.metrics import plot_confusion_matrix
```

In [2]:
```
# import dataset
df = pd.read_csv('..\\data\\telecom_data.csv')

# Handle object types for international plan and voice mail plan
df.loc[df['international plan'] == 'no', 'international plan'] = 0
df.loc[df['international plan'] == 'yes', 'international plan'] = 1

df.loc[df['voice mail plan'] == 'no', 'voice mail plan'] = 0
df.loc[df['voice mail plan']== 'yes', 'voice mail plan'] = 1

# Change churn to values: 1 (churned/True) 0 (no churn/False)
df.loc[df['churn'] == True, 'churn'] = 1
df.loc[df['churn'] == False, 'churn'] = 0

# going to create backup df and drop phone number from original df
# phone number could be used as unique id, but it doesn't seem necessary
df_backup = df.copy()
df = df.drop(['phone number'], axis=1)

# casting int values to churn, voice mail plan, and international plan cols
objs = ['international plan', 'voice mail plan', 'churn']

for o in objs:
    df = df.astype({o: int})

# dropping area code
df = df.drop(['area code'], axis=1)

# dropping various columns due to perfect (or exceptionally high) correlat
df = df.drop(['total intl minutes', 'number vmail messages'], axis=1)

# check df
df.head()
```

Out[2]:

	state	account length	international plan	voice mail plan	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
0	KS	128	0	1	265.1	110	45.07	197.4	99	16.78	244.7
1	OH	107	0	1	161.6	123	27.47	195.5	103	16.62	254.4
2	NJ	137	0	0	243.4	114	41.38	121.2	110	10.30	162.6
3	OH	84	1	0	299.4	71	50.90	61.9	88	5.26	196.9
4	OK	75	1	0	166.7	113	28.34	148.3	122	12.61	186.9

Handle international calls to bin them into categories easier.

In [3]:

```
# handle international calls to bin them into categories easier.
df['total intl calls'].value_counts()
```

Out[3]:

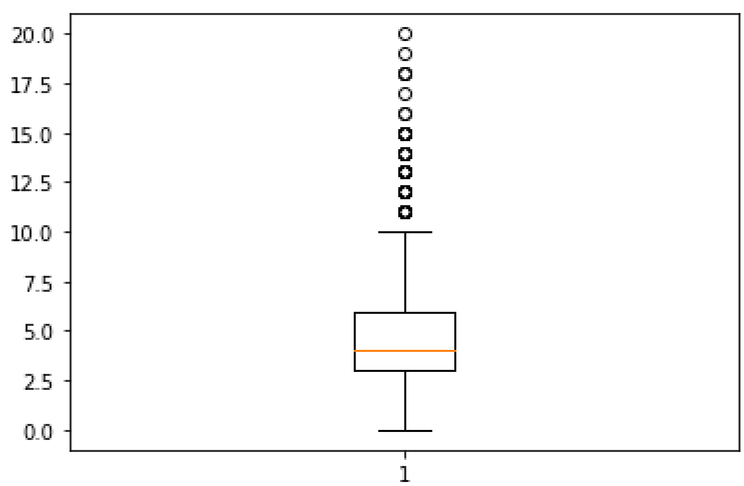
```
3     668
4     619
2     489
5     472
6     336
7     218
1     160
8     116
9     109
10     50
11     28
0      18
12     15
13     14
15      7
14      6
18      3
16      2
19      1
17      1
20      1
Name: total intl calls, dtype: int64
```

In [4]:

```
plt.boxplot(df['total intl calls'])
```

Out[4]:

```
{'whiskers': [<matplotlib.lines.Line2D at 0x1637183e0a0>,
  <matplotlib.lines.Line2D at 0x1637183e400>],
 'caps': [<matplotlib.lines.Line2D at 0x1637183e760>,
  <matplotlib.lines.Line2D at 0x1637183eac0>],
 'boxes': [<matplotlib.lines.Line2D at 0x16371823d00>],
 'medians': [<matplotlib.lines.Line2D at 0x1637183ee20>],
 'fliers': [<matplotlib.lines.Line2D at 0x1637184a1c0>],
 'means': []}
```



In [5]:
```
df['total intl calls'].describe()
```

Out[5]:
```
count    3333.000000
mean        4.479448
std         2.461214
min         0.000000
25%         3.000000
50%         4.000000
75%         6.000000
max        20.000000
Name: total intl calls, dtype: float64
```

In [6]:
```
# range is 0-20 for international calls with most concentrated from 0-10
# I will bin into cats: low, moderate, and high with values <3, 3-6, and >6

list_tmp = []

for index, row in df.iterrows():
    if row['total intl calls'] < 3:
        list_tmp.append('low')
    elif row['total intl calls'] > 6:
        list_tmp.append('high')
    else:
        list_tmp.append('moderate')

df['total_intl_calls'] = list_tmp

df['total_intl_calls'].describe()
```

Out[6]:
```
count          3333
unique            3
top        moderate
freq           2095
Name: total_intl_calls, dtype: object
```

2.2 What would be the % chance of guessing correctly if the customer was assumed to not churn?

In [7]:
```
1-df.churn.mean()
```

Out[7]:
```
0.8550855085508551
```

Guessing "not churned" for every customer would yield about an 85.5% chance of guessing correctly.

2.3 Handle datatypes

Handle state and total_intl_calls object types. Turn these into integers for later modeling.

In [8]:
```
states = ["AL", "AK", "AZ", "AR", "CA", "CO", "CT", "DC", "DE", "FL", "GA"
          "HI", "ID", "IL", "IN", "IA", "KS", "KY", "LA", "ME", "MD", "MA"
          "MI", "MN", "MS", "MO", "MT", "NE", "NV", "NH", "NJ", "NM", "NY"
          "NC", "ND", "OH", "OK", "OR", "PA", "RI", "SC", "SD", "TN", "TX"
          "UT", "VT", "VA", "WA", "WV", "WI", "WY"]
```

In [9]:
```
state_int = []
for i, row in df.iterrows():
    state_int.append(states.index(row['state']))
```

In [10]:
```
df['state_int'] = state_int
```

States in the dataset are alphabetically ordered such that Alabama is 0 and Wyoming is 50.

In [11]:
```
intl_calls_int = []
for i, row in df.iterrows():
    if row['total_intl_calls'] == 'low':
        intl_calls_int.append(0)
    elif row['total_intl_calls'] == 'moderate':
        intl_calls_int.append(1)
    else:
        intl_calls_int.append(2)

df['intl_calls_bins'] = intl_calls_int
```

Total international calls is now split into 0, 1, and 2 instead of low, moderate, and high. The new columns correspond to the following: 0 represents customers with less than 3 international calls, 1 represents customers with between 3 and 6 international calls, and 6 represents customers with over 6 international calls.

2.4 Feature Engineering: Totals for Charge, Minutes, and Calls

To reduce the number of variables and redundancies in the dataset, total calls, total charge, and total minutes will represent totals for each category. These new categories will not include international call data.

In [12]:
```
df['total_calls'] = df.apply(lambda x: x['total night calls'] + x['total da
                               axis=1)
df['total_charge']= df.apply(lambda x: x['total day charge'] + x['total eve
                                      axis=1)
df['total_minutes'] = df.apply(lambda x: x['total day minutes'] + x['total 
                                      axis=1)
```

It was observed from the EDA that total minutes and total charge are perfectly correlated. Total minutes will be dropped from the dataframe.

In [13]:
```
df = df.drop(['total_minutes'], axis=1)
```

2.5 Define X and y and Normalize the Data

In [14]:
```
y = df['churn']

# drop redundancies now that there are total charge, minutes, and calls col
# dropping any other columns that have been replaced with new ones.

X = df.drop(['churn', 'total intl calls',
             'total_intl_calls', 'state', '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'], axis=1)
```

2.6 Train-test-split and SMOTE

The target feature has imbalance because customers that churned only represent about 14.5% of the data. Synthetic Minority Oversampling Technique (SMOTE) will be implemented to create balance in the target feature such that the models do not overlook the minority class (churned customers).

In [15]:
```
# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=27)

# Fixing class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)
```

2.7 OHE and Normalize

Avoiding data leakage by transforming after the split

In [16]:
```
# Define categorical columns for OHE
cats = ['state_int', 'international plan', 'intl_calls_bins',
        'customer service calls', 'voice mail plan']
```

In [17]:
```python
# Seperate data into categorical for train and test sets
X_train_cats = X_train_resampled[cats]
X_test_cats = X_test[cats]

# handle categorical values
ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

# OHE for training categoricals
ohe.fit(X_train_cats)
X_train_ohe = pd.DataFrame(
    ohe.transform(X_train_cats),
    index=X_train_cats.index,
    columns=ohe.get_feature_names()
)

# OHE for testing categoricals
ohe.fit(X_test_cats)
X_test_ohe = pd.DataFrame(
    ohe.transform(X_test_cats),
    index=X_test_cats.index,
    columns=ohe.get_feature_names()
)

# Scaling variables to work well with OHE data -scale train and test data
X_train_numerics = X_train_resampled.drop(cats, axis=1)
X_test_numerics = X_test.drop(cats, axis=1)

scaler = MinMaxScaler()

scaler.fit(X_train_numerics)
X_train_scaled = pd.DataFrame(
    scaler.transform(X_train_numerics),
    index=X_train_numerics.index,
    columns=X_train_numerics.columns
)

scaler.fit(X_train_numerics)
X_test_scaled = pd.DataFrame(
    scaler.transform(X_test_numerics),
    index=X_test_numerics.index,
    columns=X_test_numerics.columns
)


# Concatenate and replace X_train and X_test with OHE+scaled data
X_train_resampled = pd.DataFrame()
X_train_resampled = pd.concat([X_train_scaled, X_train_ohe], axis=1)

X_test = pd.DataFrame()
X_test = pd.concat([X_test_scaled, X_test_ohe], axis=1)
```

To clarify the new confusing column names ('x0_0' and so on), the following list is provided to retain what each column refers to:

- First 51 columns (x0_0-x0_50) refer to the states

- The next two columns (x1_0 & x1_1) refer to whether or not the customer has an international plan.
- Columns x2_0-x2_2 refer to the frequency (low, moderate, high) of international calls.
- Columns x3_0-x3_9 refer to how many calls to customer service the customer has completed.
- The last two columns indicate whether or not the customer has a voice mail plan.

2.8 Confusion Matrix Helper Function

In [18]:
```
# This function prints out a string of totals for TP, TN, FP, & FN
def conf_matrix(y_true, y_pred):
    cm = {'TP': 0, 'TN': 0, 'FP': 0, 'FN': 0}

    for ind, label in enumerate(y_true):
        pred = y_pred[ind]
        if label == 1:
            # CASE: TP
            if label == pred:
                cm['TP'] += 1
            # CASE: FN
            else:
                cm['FN'] += 1
        else:
            # CASE: TN
            if label == pred:
                cm['TN'] += 1
            # CASE: FP
            else:
                cm['FP'] += 1
    return cm
```

3 Baseline Model - Logistic Regression

3.1 Fit data to model

In [19]:
```
# model
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear
                            random_state=1)
model_log = logreg.fit(X_train_resampled, y_train_resampled)
model_log
```

Out[19]:
```
LogisticRegression(C=1000000000000.0, fit_intercept=False, random_state=1,
                   solver='liblinear')
```

3.2 Evaluate

In [20]:

```
y_hat_train = logreg.predict(X_train_resampled)

train_residuals = np.abs(y_train_resampled - y_hat_train)
print(pd.Series(train_residuals, name="Residuals (counts)").value_counts()
print()
print(pd.Series(train_residuals, name="Residuals (proportions)").value_cou
```

```
0    3364
1     878
Name: Residuals (counts), dtype: int64

0    0.793022
1    0.206978
Name: Residuals (proportions), dtype: float64
```

In [21]:

```
print(confusion_matrix(y_train_resampled, y_hat_train))
print(classification_report(y_train_resampled, y_hat_train))
print("The accuracy score is" + " "+ str(accuracy_score(y_train_resampled,
```

```
[[1653  468]
 [ 410 1711]]
              precision    recall  f1-score   support

           0       0.80      0.78      0.79      2121
           1       0.79      0.81      0.80      2121

    accuracy                           0.79      4242
   macro avg       0.79      0.79      0.79      4242
weighted avg       0.79      0.79      0.79      4242

The accuracy score is 0.793022159358793
```

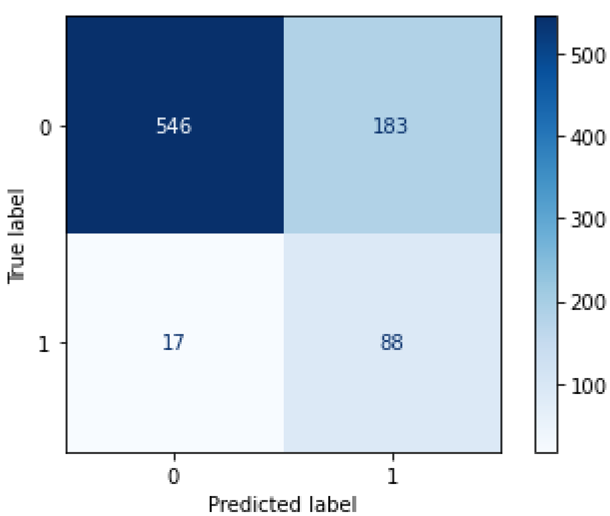
In [22]:
```
y_hat_test = logreg.predict(X_test)

test_residuals = np.abs(y_test - y_hat_test)
print(pd.Series(test_residuals, name="Residuals (counts)").value_counts())
print()
print(pd.Series(test_residuals, name="Residuals (proportions)").value_coun

print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print("The accuracy score is" + " "+ str(accuracy_score(y_test, y_hat_test
```

```
0    634
1    200
Name: Residuals (counts), dtype: int64

0    0.760192
1    0.239808
Name: Residuals (proportions), dtype: float64
[[546 183]
 [ 17  88]]
              precision    recall  f1-score   support

           0       0.97      0.75      0.85       729
           1       0.32      0.84      0.47       105

    accuracy                           0.76       834
   macro avg       0.65      0.79      0.66       834
weighted avg       0.89      0.76      0.80       834

The accuracy score is 0.7601918465227818
```

In [23]:
```
conf_matrix(y_test, y_hat_test)
```

Out[23]:
```
{'TP': 88, 'TN': 546, 'FP': 183, 'FN': 17}
```

In [24]:
```
plot_confusion_matrix(logreg, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [25]:
```
# create list for storing F-Beta2 scores for each model
fbeta2_scores = []

f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
print(fbeta2_scores)
```

```
[0.637]
```

Logistic Regression results:

- Both accuracy scores are below a the strategy of simply guessing "not churned" for every customer.
- It will be key to reduce the number of customers that are identified as retained but actually churned. Ideally everyone that can be identified as churning will be retained with some kind of strategy.
- This model produces a lot of false positives, which ultimately penalized the F2 score.

4 Model 2 - RF

4.1 Fit model

In [26]:
```
# random forest model
forest = RandomForestClassifier(n_estimators=100, max_depth=13, random_stat
forest.fit(X_train_resampled, y_train_resampled)
```

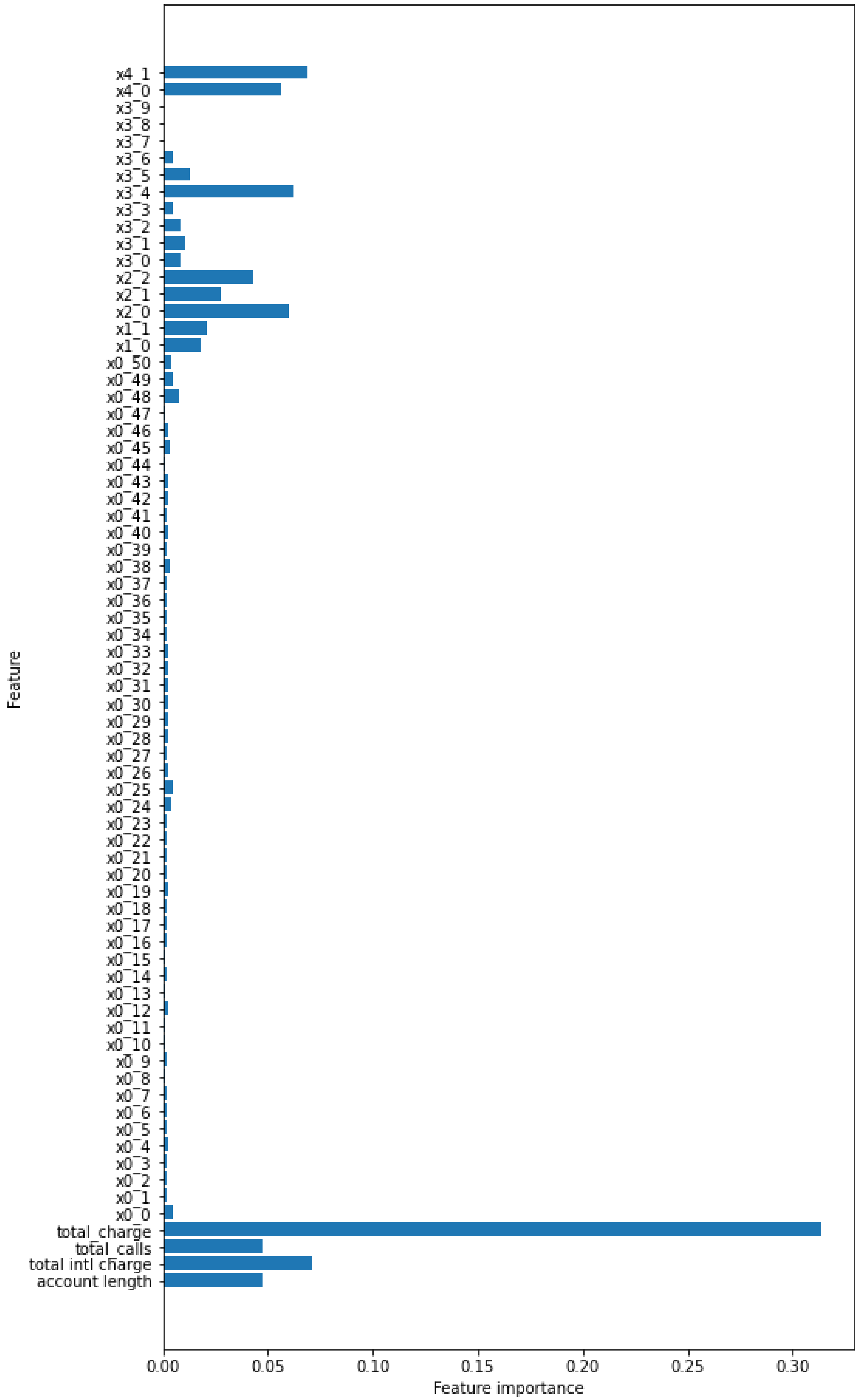
Out[26]:
```
RandomForestClassifier(max_depth=13, random_state=1)
```

4.2 Evaluate

In [27]:
```
def plot_feature_importances(model):
    n_features = X_train_resampled.shape[1]
    plt.figure(figsize=(8,16))
    plt.barh(range(n_features), model.feature_importances_, align='center'
    plt.yticks(np.arange(n_features), X_train_resampled.columns.values)
    plt.xlabel('Feature importance')
    plt.ylabel('Feature')
```

In [28]:

```
plot_feature_importances(forest)
```



In [29]:
```
# Training Accuracy
forest.score(X_train_resampled, y_train_resampled)
```

Out[29]: 0.9005186232909005

In [30]:
```
# Testing Accuracy
forest.score(X_test, y_test)
```

Out[30]: 0.8812949640287769

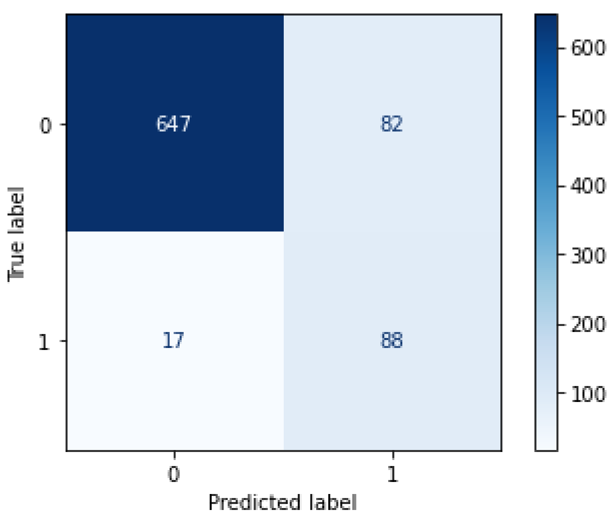
In [31]:
```
y_hat_train = forest.predict(X_train_resampled)

train_residuals = np.abs(y_train_resampled - y_hat_train)
print(pd.Series(train_residuals, name="Residuals (counts)").value_counts()]
print()
print(pd.Series(train_residuals, name="Residuals (proportions)").value_cour

print(confusion_matrix(y_train_resampled, y_hat_train))
print(classification_report(y_train_resampled, y_hat_train))
print("The accuracy score is" + " "+ str(accuracy_score(y_train_resampled,
```

```
0    3820
1     422
Name: Residuals (counts), dtype: int64

0    0.900519
1    0.099481
Name: Residuals (proportions), dtype: float64
[[2016  105]
 [ 317 1804]]
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      2121
           1       0.94      0.85      0.90      2121

    accuracy                           0.90      4242
   macro avg       0.90      0.90      0.90      4242
weighted avg       0.90      0.90      0.90      4242

The accuracy score is 0.9005186232909005
```

In [32]:
```
y_hat_test = forest.predict(X_test)

test_residuals = np.abs(y_test - y_hat_test)
print(pd.Series(test_residuals, name="Residuals (counts)").value_counts())
print()
print(pd.Series(test_residuals, name="Residuals (proportions)").value_coun
 
print(confusion_matrix(y_test, y_hat_test))
print(classification_report(y_test, y_hat_test))
print("The accuracy score is" + " "+ str(accuracy_score(y_test, y_hat_test]
```

```
0    735
1     99
Name: Residuals (counts), dtype: int64

0    0.881295
1    0.118705
Name: Residuals (proportions), dtype: float64
[[647  82]
 [ 17  88]]
              precision    recall  f1-score   support

           0       0.97      0.89      0.93       729
           1       0.52      0.84      0.64       105

    accuracy                           0.88       834
   macro avg       0.75      0.86      0.78       834
weighted avg       0.92      0.88      0.89       834

The accuracy score is 0.8812949640287769
```

In [33]:
```
plot_confusion_matrix(forest, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [34]:
```
f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[34]:
```
[0.637, 0.746]
```

5 Model 3 - Optimize RF Model

5.1 Fit and use GridSearchCV to find best parameters

In [35]:
```
# FBeta2 Scorer
ftwo_scorer = make_scorer(fbeta_score, beta=2)
```

In [36]:
```
# Using a pipeline to select for optimal parameters in the RF Classifier
pipe = Pipeline([('clf', RandomForestClassifier(random_state=1))])

grid_params = [{'clf__n_estimators': [100, 1000],
                'clf__criterion': ['gini', 'entropy'],
                'clf__max_depth': [None, 1, 5, 7, 11, 13, 17, 19],
                'clf__min_samples_split': [2, 5, 10],
                'clf__min_samples_leaf': [2, 3, 4, 5]
               }]

# grid search
gs = GridSearchCV(estimator=pipe,
                  param_grid=grid_params,
                  scoring=ftwo_scorer,
                  cv=5)

# Fit using grid search
gs.fit(X_train_resampled, y_train_resampled)

# Best F-2 Score
print('Best F-2 Score: %.3f' % gs.best_score_)

# Best params
print('\nBest params:\n', gs.best_params_)
```

```
Best F-2 Score: 0.817

Best params:
 {'clf__criterion': 'entropy', 'clf__max_depth': None, 'clf__min_samples_leaf':
2, 'clf__min_samples_split': 2, 'clf__n_estimators': 100}
```

This GridSearchCV code took my machine about 40 minutes to run.

In [37]:
```
forest = RandomForestClassifier(n_estimators=100, criterion='entropy',
                                max_depth=None, min_samples_leaf=2,
                                min_samples_split=2, random_state=1)
forest.fit(X_train_resampled, y_train_resampled)
```

Out[37]:
```
RandomForestClassifier(criterion='entropy', min_samples_leaf=2, random_state=1)
```

5.2 Evaluate

In [38]:
```
# Training Accuracy
forest.score(X_train_resampled, y_train_resampled)
```

Out[38]:
```
0.9594530881659594
```

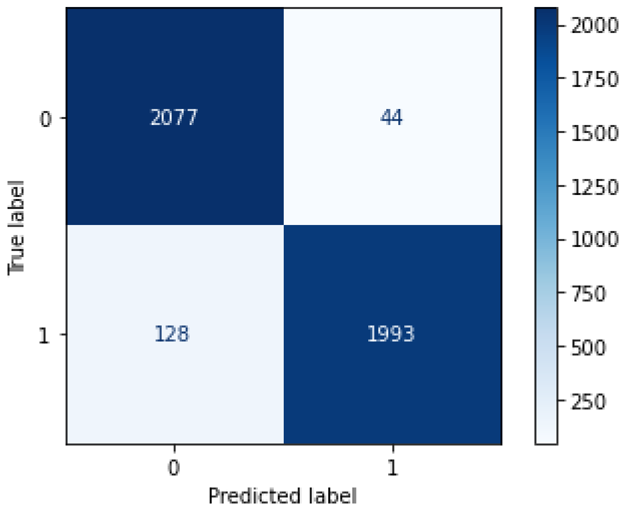
In [39]:
```
# Testing Accuracy
forest.score(X_test, y_test)
```

Out[39]:
```
0.8980815347721822
```

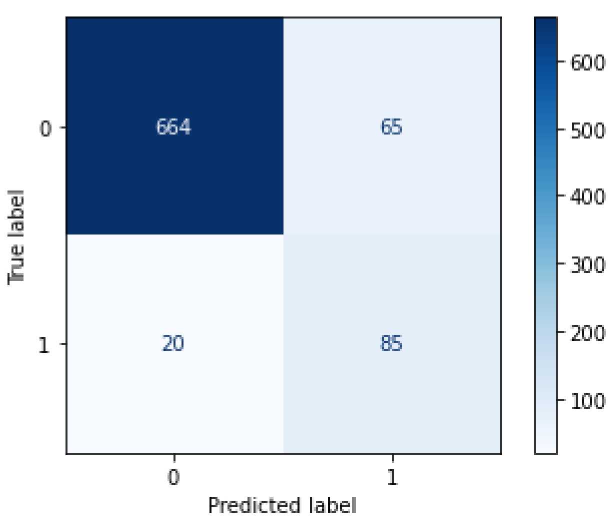
In [40]:
```
print(classification_report(y_train_resampled, y_hat_train))
print(classification_report(y_test, y_hat_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      2121
           1       0.94      0.85      0.90      2121

    accuracy                           0.90      4242
   macro avg       0.90      0.90      0.90      4242
weighted avg       0.90      0.90      0.90      4242

              precision    recall  f1-score   support

           0       0.97      0.89      0.93       729
           1       0.52      0.84      0.64       105

    accuracy                           0.88       834
   macro avg       0.75      0.86      0.78       834
weighted avg       0.92      0.88      0.89       834
```

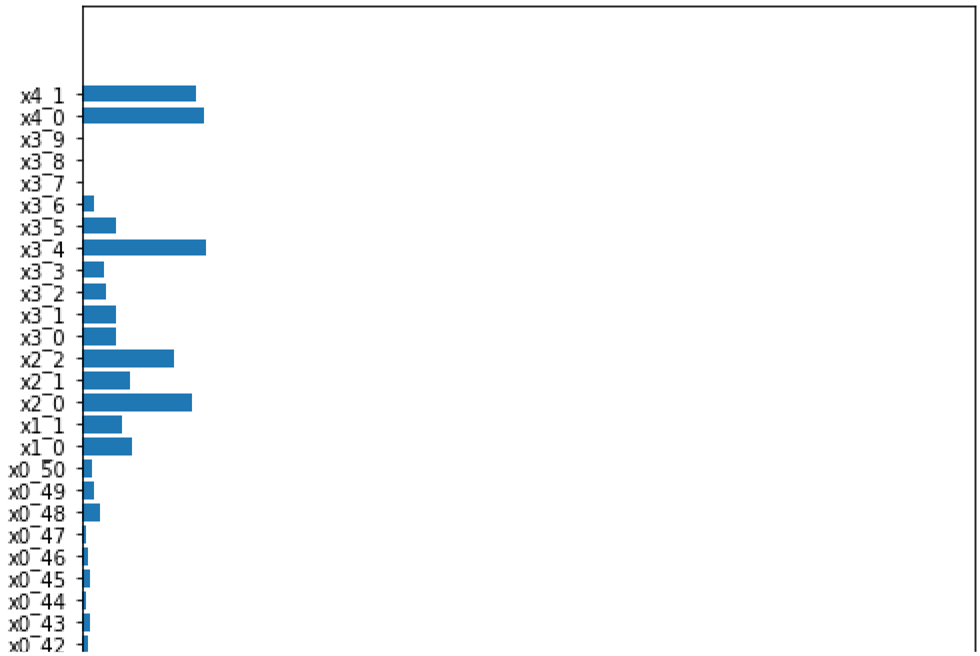
In [41]:
```
plot_confusion_matrix(forest, X_train_resampled, y_train_resampled,
                      cmap=plt.cm.Blues)
plt.show()
```



In [42]:
```
plot_confusion_matrix(forest, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [43]:
```
plot_feature_importances(forest)
```



In [44]:
```
y_hat_test = forest.predict(X_test)
f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[44]:
```
[0.637, 0.746, 0.746]
```

6 Model 4 - Optimize RF Model (2nd Attempt)

Attempting to reduce runtime by decreasing parameters a bit, but this code still does take a while to run.

To avoid overfitting in the RF Model, the following strategies will be utilized:

- Use less features (use something like 50% of total features for the max features parameter).
- Keep n_estimators large because the more trees there are, the less likely it is to overfit.

6.1 Optimize RF Parameters and Fit

In [45]:
```
# Using a pipeline to select for optimal parameters in the RF Classifier
pipe = Pipeline([('clf', RandomForestClassifier(random_state=1))])

grid_params = [{'clf__n_estimators': [100],
                'clf__criterion': ['entropy'],
                'clf__max_depth': [None, 1, 7, 15],
                'clf__min_samples_split': [2, 5, 10],
                'clf__min_samples_leaf': [2, 3, 4, 5],
                'clf__max_features': [37]     #this is 50% of total feature
              }]

# grid search
gs = GridSearchCV(estimator=pipe,
                  param_grid=grid_params,
                  scoring=ftwo_scorer,
                  cv=5)

# Fit using grid search
gs.fit(X_train_resampled, y_train_resampled)

# Best F-2 Score
print('Best F-2 Score: %.3f' % gs.best_score_)

# Best params
print('\nBest params:\n', gs.best_params_)
```

```
Best F-2 Score: 0.848

Best params:
 {'clf__criterion': 'entropy', 'clf__max_depth': None, 'clf__max_features': 37,
'clf__min_samples_leaf': 2, 'clf__min_samples_split': 5, 'clf__n_estimators': 1
00}
```

In [46]:
```
forest = RandomForestClassifier(n_estimators=100, criterion='entropy',
                                max_depth=None, min_samples_leaf=2,
                                min_samples_split=5, random_state=1,
                                max_features=37)
forest.fit(X_train_resampled, y_train_resampled)
```

Out[46]:
```
RandomForestClassifier(criterion='entropy', max_features=37, min_samples_leaf=
2,
                       min_samples_split=5, random_state=1)
```

6.2 Evaluate

In [47]:
```
# Training Accuracy
forest.score(X_train_resampled, y_train_resampled)
```

Out[47]:
```
0.9929278642149929
```

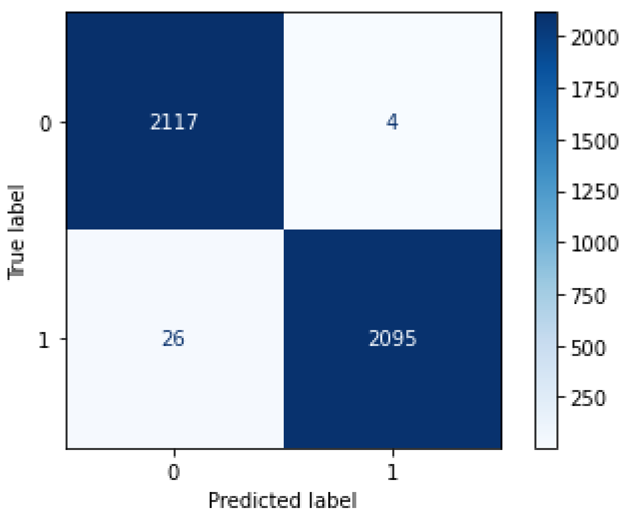
In [48]:
```
# Testing Accuracy
forest.score(X_test, y_test)
```

Out[48]:
```
0.9160671462829736
```

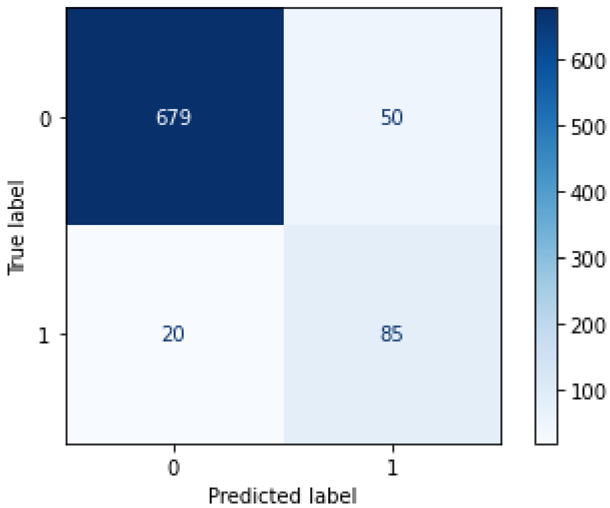
In [49]:
```
print(classification_report(y_train_resampled, y_hat_train))
print(classification_report(y_test, y_hat_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      2121
           1       0.94      0.85      0.90      2121

    accuracy                           0.90      4242
   macro avg       0.90      0.90      0.90      4242
weighted avg       0.90      0.90      0.90      4242

              precision    recall  f1-score   support

           0       0.97      0.91      0.94       729
           1       0.57      0.81      0.67       105

    accuracy                           0.90       834
   macro avg       0.77      0.86      0.80       834
weighted avg       0.92      0.90      0.91       834
```

In [50]:
```
plot_confusion_matrix(forest, X_train_resampled, y_train_resampled,
                      cmap=plt.cm.Blues)
plt.show()
```



In [51]:
```
plot_confusion_matrix(forest, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [52]:
```
y_hat_test = forest.predict(X_test)
f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[52]:
```
[0.637, 0.746, 0.746, 0.766]
```

7 Model 5 - AdaBoost

The motivation for this model is to reduce runtime expense found in the RF model optimization processes and to perform better than the SVM model.

7.1 Instantiate Model and Fit

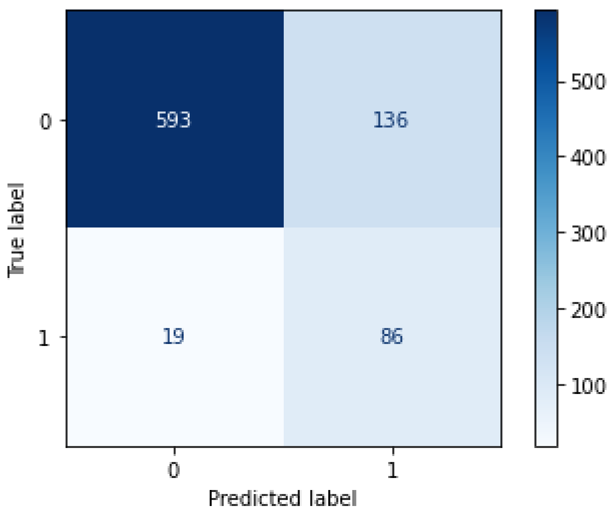
In [53]:
```python
from sklearn.ensemble import AdaBoostClassifier
adaboost_clf = AdaBoostClassifier(random_state=42)
adaboost_clf.fit(X_train_resampled, y_train_resampled)
```

Out[53]:
```
AdaBoostClassifier(random_state=42)
```

7.2 Evaluate

In [54]:
```python
# AdaBoost model predictions for test and train sets
adaboost_train_preds = adaboost_clf.predict(X_train_resampled)
adaboost_test_preds = adaboost_clf.predict(X_test)
```

In [55]:
```python
plot_confusion_matrix(adaboost_clf, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [56]:
```python
print(classification_report(y_test, adaboost_test_preds))
print("The accuracy score is" + " "+ str(accuracy_score(y_test, adaboost_te
```

```
              precision    recall  f1-score   support

           0       0.97      0.81      0.88       729
           1       0.39      0.82      0.53       105

    accuracy                           0.81       834
   macro avg       0.68      0.82      0.71       834
weighted avg       0.90      0.81      0.84       834

The accuracy score is 0.8141486810551559
```

In [57]:
```
print(classification_report(y_train_resampled, adaboost_train_preds))
print("The accuracy score is" + " "+ str(accuracy_score(y_train_resampled,
```

```
              precision    recall  f1-score   support

           0       0.80      0.83      0.82      2121
           1       0.82      0.80      0.81      2121

    accuracy                           0.81      4242
   macro avg       0.81      0.81      0.81      4242
weighted avg       0.81      0.81      0.81      4242

The accuracy score is 0.8135313531353136
```

In [58]:
```
f = fbeta_score(y_test, adaboost_test_preds, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[58]:
```
[0.637, 0.746, 0.746, 0.766, 0.67]
```

Comments on AdaBoost Model:

- It is very fast
- It has a much higher rate of FPs compared to the RF models.
- It does not have as good of scoring (by any metric of comparison) as the RF models

7.3 Tune Model with GridSearchCV and Run Again

In [59]:
```
gs = GridSearchCV(estimator=adaboost_clf,
                  param_grid={
                      'n_estimators': [25, 50, 100, 500],
                      'learning_rate': [.001, .01, .1, 1]
                  }, scoring=ftwo_scorer, cv=5)

gs.fit(X_train_resampled, y_train_resampled)
gs.best_params_
```

Out[59]:
```
{'learning_rate': 1, 'n_estimators': 100}
```

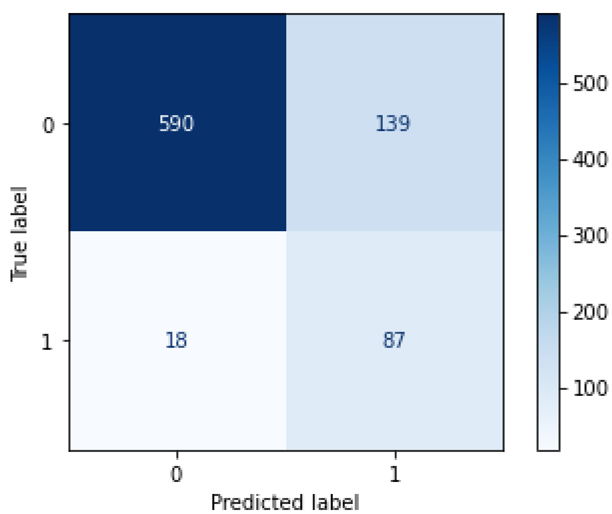
In [60]:
```
adaboost_clf = AdaBoostClassifier(learning_rate=1, n_estimators=100,
                                  random_state=42)
adaboost_clf.fit(X_train_resampled, y_train_resampled)
```

Out[60]:
```
AdaBoostClassifier(learning_rate=1, n_estimators=100, random_state=42)
```

In [61]:
```
# AdaBoost model predictions
adaboost_train_preds = adaboost_clf.predict(X_train_resampled)
adaboost_test_preds = adaboost_clf.predict(X_test)

print(classification_report(y_train_resampled, adaboost_train_preds))
print(classification_report(y_test, adaboost_test_preds))
```

```
              precision    recall  f1-score   support

           0       0.81      0.83      0.82      2121
           1       0.83      0.80      0.82      2121

    accuracy                           0.82      4242
   macro avg       0.82      0.82      0.82      4242
weighted avg       0.82      0.82      0.82      4242

              precision    recall  f1-score   support

           0       0.97      0.81      0.88       729
           1       0.38      0.83      0.53       105

    accuracy                           0.81       834
   macro avg       0.68      0.82      0.70       834
weighted avg       0.90      0.81      0.84       834
```

In [62]:
```
plot_confusion_matrix(adaboost_clf, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.show()
```



In [63]:
```
f = fbeta_score(y_test, adaboost_test_preds, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[63]: `[0.637, 0.746, 0.746, 0.766, 0.67, 0.673]`

8 RF Without State Column

The 'State' column is probably not particularly useful for SyriaTel, and none of the states seem to have super strong feature importance by themselves.

8.1 Remove State Column and Prepare Dataset to Model

In [64]:
```
# Define X and y
y = df['churn']

# drop state_int column as well this time
X = df.drop(['churn', 'total intl calls',
             'total_intl_calls', 'state', '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', 'state_int'], axis=

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=27)

# Fixing class imbalance with SMOTE
smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_sample(X_train, y_train)

# define categorical columns
cats = ['international plan', 'intl_calls_bins',
        'customer service calls', 'voice mail plan']

# Seperate data into categorical for train and test sets
X_train_cats = X_train_resampled[cats]
X_test_cats = X_test[cats]

# handle categorical values
ohe = OneHotEncoder(handle_unknown="ignore", sparse=False)

ohe.fit(X_train_cats)
X_train_ohe = pd.DataFrame(
    ohe.transform(X_train_cats),
    index=X_train_cats.index,
    columns=ohe.get_feature_names()
)

ohe.fit(X_test_cats)
X_test_ohe = pd.DataFrame(
    ohe.transform(X_test_cats),
    index=X_test_cats.index,
    columns=ohe.get_feature_names()
)

# Scaling variables to work well with OHE data
X_train_numerics = X_train_resampled.drop(cats, axis=1)
X_test_numerics = X_test.drop(cats, axis=1)

scaler = MinMaxScaler()

scaler.fit(X_train_numerics)
X_train_scaled = pd.DataFrame(
    scaler.transform(X_train_numerics),
    index=X_train_numerics.index,
    columns=X_train_numerics.columns
)

scaler.fit(X_train_numerics)
```

```
X_test_scaled = pd.DataFrame(
    scaler.transform(X_test_numerics),
    index=X_test_numerics.index,
    columns=X_test_numerics.columns
)


# Concatenate
X_train_resampled = pd.DataFrame()
X_train_resampled = pd.concat([X_train_scaled, X_train_ohe], axis=1)

X_test = pd.DataFrame()
X_test = pd.concat([X_test_scaled, X_test_ohe], axis=1)
```

8.2 Fit Model

In [65]:

```
# random forest model
forest = RandomForestClassifier(n_estimators=100, max_depth=13,
                                random_state=69)
forest.fit(X_train_resampled, y_train_resampled)
```

Out[65]:

```
RandomForestClassifier(max_depth=13, random_state=69)
```

8.3 Evaluate

In [66]:

```
print(classification_report(y_train_resampled, y_hat_train))
print(classification_report(y_test, y_hat_test))
```

```
              precision    recall  f1-score   support

           0       0.86      0.95      0.91      2121
           1       0.94      0.85      0.90      2121

    accuracy                           0.90      4242
   macro avg       0.90      0.90      0.90      4242
weighted avg       0.90      0.90      0.90      4242

              precision    recall  f1-score   support

           0       0.97      0.93      0.95       729
           1       0.63      0.81      0.71       105

    accuracy                           0.92       834
   macro avg       0.80      0.87      0.83       834
weighted avg       0.93      0.92      0.92       834
```

In [67]:

```
# Training Accuracy
forest.score(X_train_resampled, y_train_resampled)
```

Out[67]:

```
0.9429514380009429
```

In [68]:
```
# Testing Accuracy
forest.score(X_test, y_test)
```

Out[68]: 0.9304556354916067

In [69]:
```
final_cols = ['account length', 'total intl charge',
              'total_calls', 'total_charge',
              'no international plan', 'has international plan',
              'low international calls', 'moderate international calls',
              'high international calls', '0 customer service calls',
              '1 customer service call', '2 customer service calls',
              '3 customer service calls',
              '4 customer service calls', '5 customer service calls',
              '6 customer service calls', '7 customer service calls',
              '8 customer service calls', '9 customer service calls',
              'no voice mail plan', 'has voice mail plan']
```

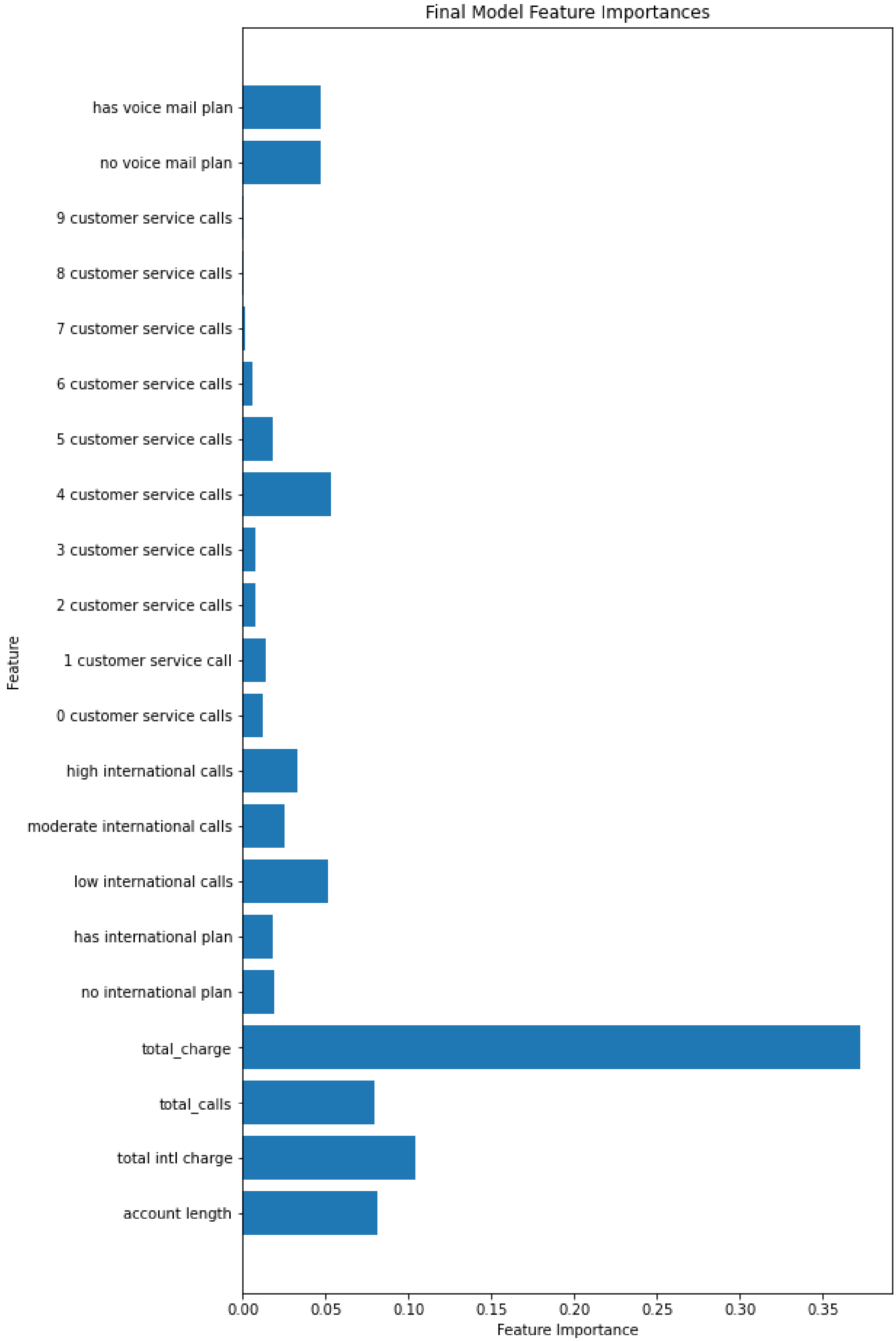
In [70]:
```
# Create table of feature importances and values
df_final_feature_importances = pd.DataFrame(index=(final_cols))
# df_final_feature_importances.set_axis([final_cols], axis=1)
df_final_feature_importances['feature importances values'] = np.round(fores
df_final_feature_importances = df_final_feature_importances.sort_values(
    by=['feature importances values'], ascending=False)
df_final_feature_importances
```

Out[70]:

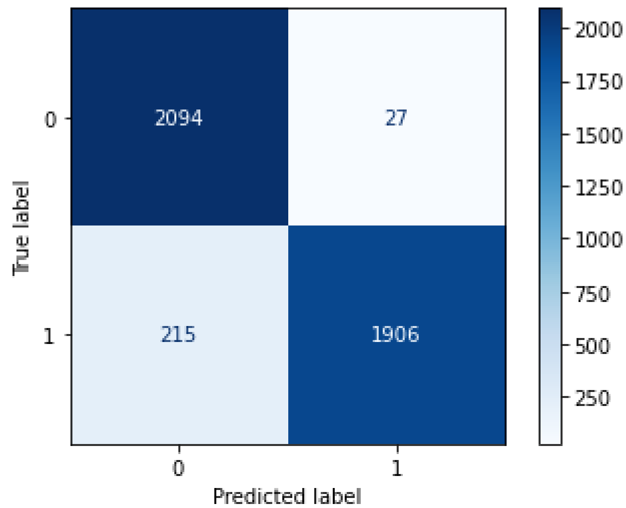
	feature importances values
total_charge	0.373
total intl charge	0.104
account length	0.081
total_calls	0.079
4 customer service calls	0.053
low international calls	0.051
no voice mail plan	0.047
has voice mail plan	0.047
high international calls	0.033
moderate international calls	0.025
no international plan	0.019
has international plan	0.018
5 customer service calls	0.018
1 customer service call	0.014
0 customer service calls	0.012
2 customer service calls	0.008
3 customer service calls	0.008
6 customer service calls	0.006
7 customer service calls	0.001
8 customer service calls	0.001
9 customer service calls	0.000

In [71]:
```
# creating new feature importances plot function to allow for more
# comprehensability for this plot, which will be used elsewhere as well.

def plot_feature_importances_final(model):
    n_features = X_train_resampled.shape[1]
    plt.figure(figsize=(8,16))
    plt.barh(range(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), final_cols)
    plt.xlabel('Feature Importance')
    plt.ylabel('Feature')
    plt.title('Final Model Feature Importances')
```

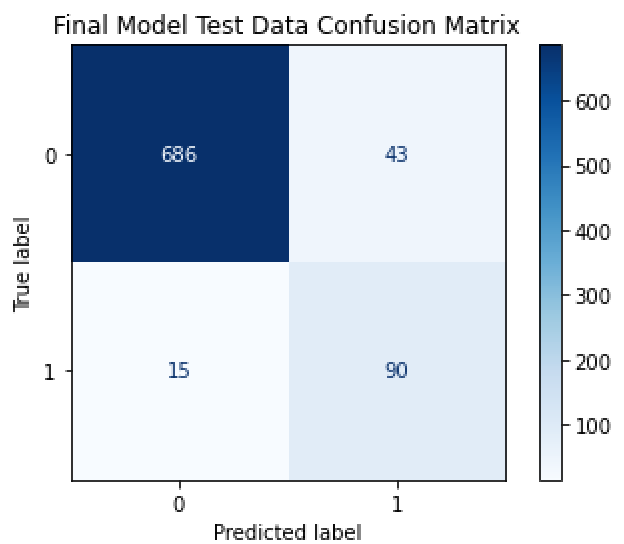
In [72]:
```
1 plot_feature_importances_final(forest)
```



In [73]:
```
plot_confusion_matrix(forest, X_train_resampled, y_train_resampled,
                      cmap=plt.cm.Blues)
plt.show()
```



In [74]:
```
plot_confusion_matrix(forest, X_test, y_test,
                      cmap=plt.cm.Blues)
plt.title("Final Model Test Data Confusion Matrix")
plt.show()
```



In [75]:
```
y_hat_test = forest.predict(X_test)
f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[75]:
```
[0.637, 0.746, 0.746, 0.766, 0.67, 0.673, 0.814]
```

8.4 Tune Model with GridSearchCV

In [76]:
```
forest = RandomForestClassifier(random_state=1)

gs = GridSearchCV(estimator=forest,
                  param_grid={
                      'n_estimators': [10, 100, 1000],
                      'criterion': ['entropy', 'gini'],
                      'max_depth': [None, 1, 7, 15, 23, 31],
                      'min_samples_split': [2, 5, 10],
                      'min_samples_leaf': [2, 3, 4, 5, 6]
                  }, scoring=ftwo_scorer, cv=5)

gs.fit(X_train_resampled, y_train_resampled)
```

Out[76]:
```
GridSearchCV(cv=5, estimator=RandomForestClassifier(random_state=1),
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': [None, 1, 7, 15, 23, 31],
                         'min_samples_leaf': [2, 3, 4, 5, 6],
                         'min_samples_split': [2, 5, 10],
                         'n_estimators': [10, 100, 1000]},
             scoring=make_scorer(fbeta_score, beta=2))
```

Approximately 42 minute runtime for this GridSearchCV.

In [77]:
```
# Best F-2 Score
print('Best F2 Score: %.3f' % gs.best_score_)

# Best params
print('\nBest params:\n', gs.best_params_)
```

```
Best F2 Score: 0.839

Best params:
 {'criterion': 'entropy', 'max_depth': 23, 'min_samples_leaf': 2, 'min_samples_
split': 2, 'n_estimators': 1000}
```

In [78]:
```
forest = RandomForestClassifier(n_estimators=1000, criterion='entropy',
                                min_samples_leaf=2, min_samples_split=2,
                                max_depth=23, random_state=42)
forest.fit(X_train_resampled, y_train_resampled)
```

Out[78]:
```
RandomForestClassifier(criterion='entropy', max_depth=23, min_samples_leaf=2,
                       n_estimators=1000, random_state=42)
```

8.5 Re-Evaluate

In [79]:
```
print(classification_report(y_train_resampled, y_hat_train, digits=4))
print(classification_report(y_test, y_hat_test, digits=4))
```

```
              precision    recall  f1-score   support

           0     0.8641    0.9505    0.9053      2121
           1     0.9450    0.8505    0.8953      2121

    accuracy                         0.9005      4242
   macro avg     0.9046    0.9005    0.9003      4242
weighted avg     0.9046    0.9005    0.9003      4242

              precision    recall  f1-score   support

           0     0.9786    0.9410    0.9594       729
           1     0.6767    0.8571    0.7563       105

    accuracy                         0.9305       834
   macro avg     0.8276    0.8991    0.8579       834
weighted avg     0.9406    0.9305    0.9339       834
```

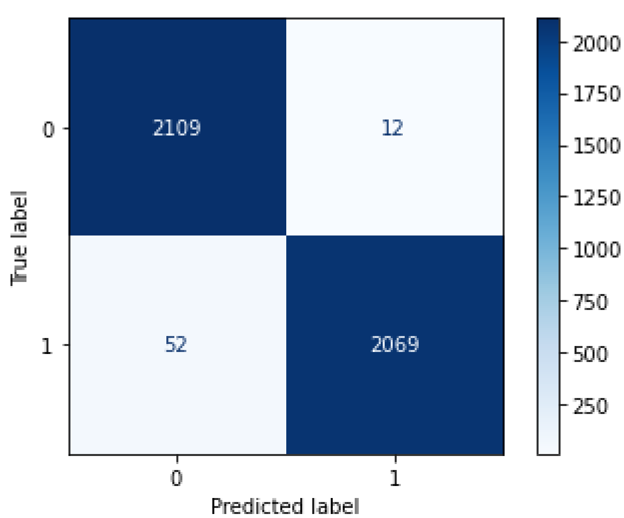
In [80]:
```
# Training Accuracy
forest.score(X_train_resampled, y_train_resampled)
```

Out[80]: 0.9849127769919849

In [81]:
```
# Testing Accuracy
forest.score(X_test, y_test)
```

Out[81]: 0.9280575539568345

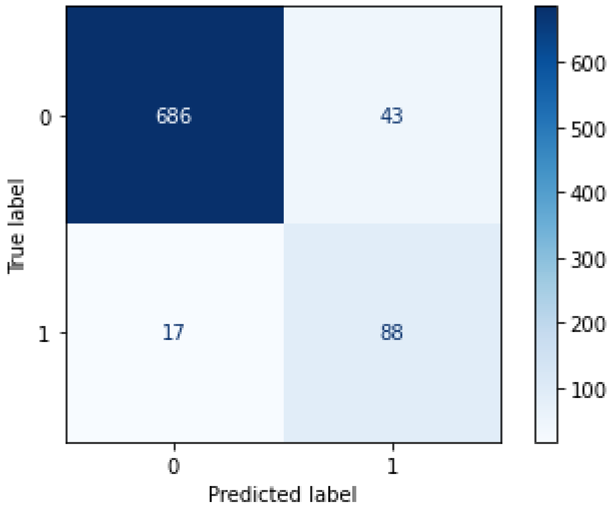
In [82]:
```
plot_confusion_matrix(forest, X_train_resampled, y_train_resampled,
                      cmap=plt.cm.Blues)
plt.show()
```



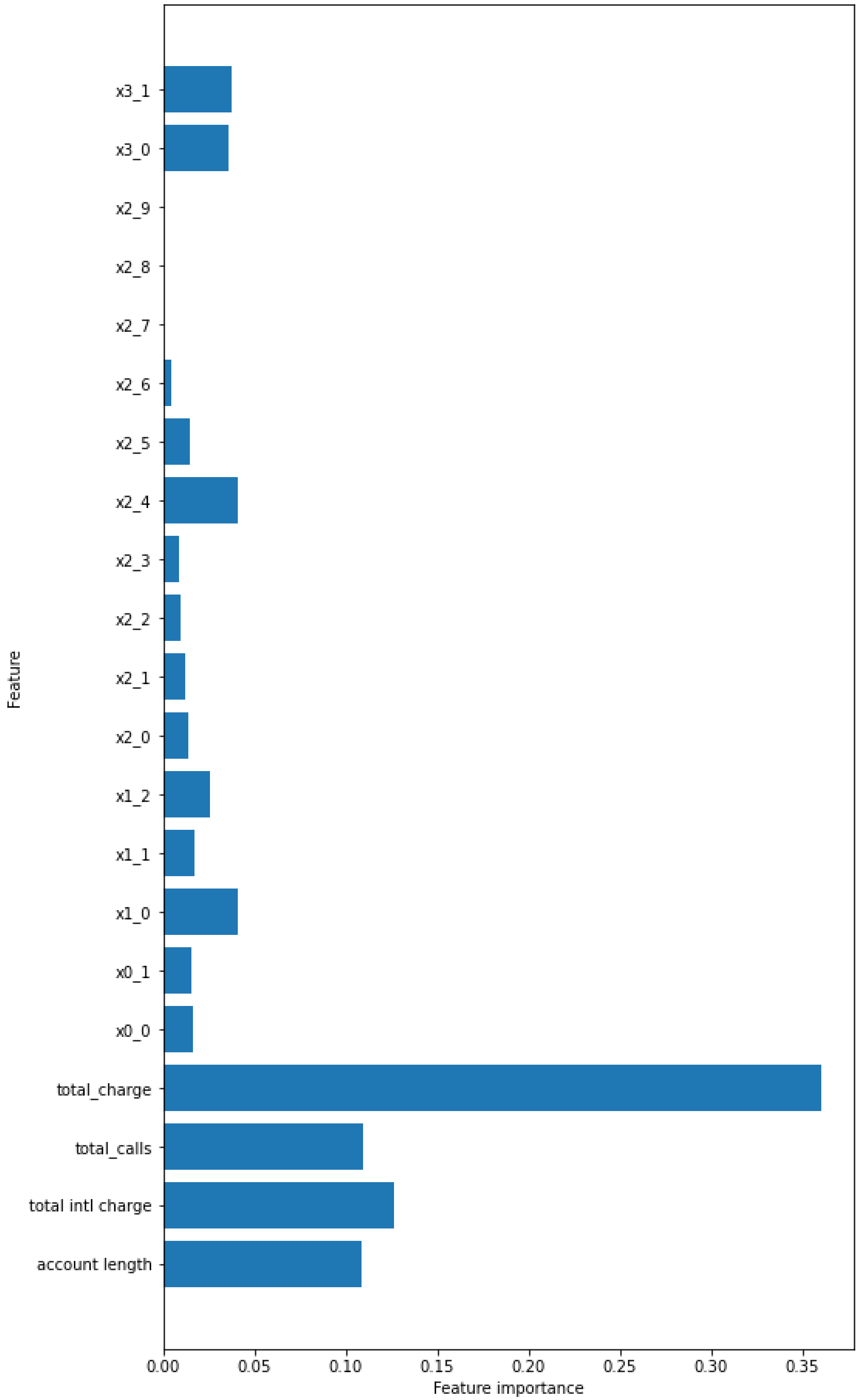
In [83]:
```
y_pred = forest.predict(X_test)
conf_matrix(y_test, y_pred)
```

Out[83]:
```
{'TP': 88, 'TN': 686, 'FP': 43, 'FN': 17}
```

In [84]:
```
plot_confusion_matrix(forest, X_test, y_test, cmap=plt.cm.Blues)
plt.show()
```



In [85]:
```
plot_feature_importances(forest)
```



In [86]:
```
y_hat_test = forest.predict(X_test)
f = fbeta_score(y_test, y_hat_test, beta=2.0)
fbeta2_scores.append(round(f, 3))
fbeta2_scores
```

Out[86]:
```
[0.637, 0.746, 0.746, 0.766, 0.67, 0.673, 0.814, 0.799]
```

Suppressing the following code because it produces a large file size for the image it makes. The following code creates an image of a single tree in the forest in the section 8 tuned model.

In [87]:
```
# try to visualize the tree
# adapted from: https://towardsdatascience.com/how-to-visualize-a-decision-
# credits to Will Koehresen for the code above (link above)

# from sklearn.tree import export_graphviz
# # Export as dot file
# export_graphviz(forest.estimators_[3], out_file='tree.dot',
#                 feature_names = X_train_resampled.columns.to_list(),
#                 class_names = 'churn',
#                 rounded = True, proportion = False,
#                 precision = 3, filled = True)

# # Convert to png using system command (requires Graphviz)
# from subprocess import call
# call(['dot', '-Tpng', 'tree.dot', '-o', 'tree.png', '-Gdpi=600'])

# # Display in jupyter notebook
# from IPython.display import Image
# Image(filename = 'tree.png')
```

9 Final Model Selection and Discussion

9.1 Overview

The goal of this project is to reduce customer churn rate for SyriaTel. This dataset is based in the US, so it is important for model results to extrapolate well to an application in Syria (where SyriaTel is based). To reduce customer churn rate, it is important to identify customers that are likely to churn, why these customers churn, and deploy a customer retention strategy on these customers.

Main evaluation metrics to use and the basis of justification for utilizing these metrics:

- The main goal is to implement retention strategies such as offering discounted rates, better plans, etc. to customers identified as likely to churn.
- Customers that are classified as churning by the model will be offered a retention strategy that will probably cost money to implement, but it can be justified by retaining the customer in the long term.
- Accuracy is important for minimizing total false negatives and false positives.
- False negatives (customers identified as retained but are not actually retained) are important to minimize because these are customers that "fell through the cracks" and left without any attempt to save their business.
- False positives are important to minimize because these customers will potentially represent an unneeded cost because they will be offered one of the retention strategies, which costs the company money, even though they don't need incentive to stay with the company. This loss could be countered with some type of survey to identify dissatisfied customers, but it is important to decrease the amount of false positives.
- False negatives (customers identified as not churning but actually churned) should be penalized more because these customers do not have the potential to be retained with any

particular strategy, they simply fall through the cracks. This cost is greater than a decent retention strategy because the retention strategy will signify an initial cost, but it will allow for the customer to be retained for a longer time.

Other justifications for model selection:

- The model must have potential to extrapolate from this dataset (from a US-based company) to the stakeholder (SyriaTel).
- It is ideal for the model to be deployed somewhat quickly so that customers can be classified as likely to churn or not at the end of various time periods, and so that customer retention strategies can be deployed quickly.

Potential Retention Strategies:

- If the customer has an international plan, then offer a certain amount of months for free. The international plan in this dataset doesn't seem to do anything (check EDA notebook), and it probably wouldn't be of any cost for this particular company to offer international plans free for an extended period. But this idea is one strategy for retaining customers with international plans.
- Offer a discount for call charges. Total charge was an important feature in most models, so it will be useful to offer discounts to retain customers.
- Create better customer service methods and offer solutions quickly to customers that call multiple times.
- Have an effective international plan.

9.2 Customer Retention Strategy and Evaluation Metric

Main Retention Strategy to Consider:

- Offer customers identified as likely to churn the option to have a discount such as 6 months at 50% off.

Items to consider:

- Assumptions have to be made in order to evaluate how effective the retention strategy will be.
- Assumption 1: average cost to acquire a new customer is around 300 dollars. Sources: https://www.forbes.com/sites/forbestechcouncil/2020/10/30/acquiring-subscribers-is-only-half-the-battle/https://www.entrepreneur.com/article/225415 (https://www.forbes.com/sites/forbestechcouncil/2020/10/30/acquiring-subscribers-is-only-half-the-battle/https://www.entrepreneur.com/article/225415)
- Average phone bill cost (in the US) for a single user: 70 dollars/month. Source: https://www.usmobile.com/blog/cut-cell-phone-bill/ (https://www.usmobile.com/blog/cut-cell-phone-bill/)
- Assume that customers that leave the service will cost the company 370 dollars (cost of one month of service + cost to acquire new customer).
- For False Positive customers (model identifies customer as churning, but they do not churn), assume the cost to the company is (35 dollars* 6 months) = 210 dollars.

- For False Negative customers (model identifies as retained but they churn), assume the cost to the company is 370 dollars.
- For True Positive customers, assume the rate of retention (after offering the discount) is 80%.
- The expected value gained for any True Positive customer is .8(-210+370) + .2(-370) = 54 dollars.

Discussion of Metric to Use (F-Beta2):

- A False Negative costs about 1.762 times as much as a False Positive from the above calculations. The metric will need to implement recognition of this discrepancy.
- Could use expected amount of money gained or lost.
- Use F-Beta with a beta value > 1 to penalize False Negatives more than False Positives.
- Due to the nature of making many assumptions in the previous calculations, F-Beta2 score will be used to compare model performance between each other. This scoring method ensures that FN's are penalized more than FP's. F-Beta2 will not necessarily penalize FN's exactly 1.762 times more than FP's, but it will give a good comparison between models that will offer a good estimation that doesn't have to rely on the assumptions made.

9.3 Final Model and Inferences from the Model

In [88]:
```
fbeta2_scores
```

Out[88]:
```
[0.637, 0.746, 0.746, 0.766, 0.67, 0.673, 0.814, 0.799]
```

The first RF model without the state column performed the best in terms of F-2 Score.

9.3.1 Review the top feature importances of the model

In [89]:
```
df_final_feature_importances.head(10)
```

Out[89]:

	feature importances values
total_charge	0.373
total intl charge	0.104
account length	0.081
total_calls	0.079
4 customer service calls	0.053
low international calls	0.051
no voice mail plan	0.047
has voice mail plan	0.047
high international calls	0.033
moderate international calls	0.025

By far the most important feature is total charge. It will be important to focus on discounting costs of customers that are predicted to churn. It would be worthwhile to investigate better overall methods to decrease costs for customers. The second most important feature is total international charge, and the third most important feature is total calls. It seems like there may be a common theme with either charging too much or not providing comprehensive enough packages for calling. There are various business strategies that could reduce churn in these cases such as rewarding customers for utilizing the service more or offering more comprehensive plans for customers such as unlimited calling.

9.3.2 Estimate money saved with this model

Without a retention strategy, how much money is lost? SyriaTel has around 8 million customers (according to its LinkedIn page), so let's assume that and account for lost money without a strategy. With the assumptions made in section 10.2, 370 dollars is lost for customers that are not identified as churning. 14.5% of customers in the original dataset churned. With eight million total customers, that would represent a loss of business from 1.16 million customers and $429.2M.

In the final model, the frequency of FN's (customers that churn but are predicted as retained) is about 1.80%. The frequency of TP's (TP's / total observations) is about 10.80%. The frequency of FP's is about 5.16%. Therefore, FN's will account for 144,000 customers, FP's will account for 412,800 customers, and TP's will account for 864,000 customers. The expected loss (derived from the calculations in section 10.2) equates to 53,280,000 dollars from FN's and 86,688,000 dollars from FP's. The expected gain from TP's is 46,656,000 dollars. In total, SyriaTel is expected to lose about 93,312,000 dollars with this retention strategy, but SyriaTel will save 335,888,000 dollars by implementing this strategy in comparison to having no strategy.

9.3.3 Conclusions

With the retention strategy covered in this notebook, SyriaTel is estimated to save just under 336 million dollars. Obviously SyriaTel is based in Syria and probably does not use dollars, and the estimations for these calculations is based somewhat on US telecommunications data (reference section 10.2). However, the point of this notebook is that the model created allows for SyriaTel to save an enormous amount of money. There are also other considerations that can be inferred from this model that SyriaTel can make use of. For instance, this dataset highlights the need for good customer service, especially after the customer has contacted customer service previously. The dataset also highlights the need for a comprehensive international plan (if that plan is offered) because many customers with an international plan ended up churning in this dataset.

Further considerations:

- Tune estimation parameters to account for telecommunication data in Syria.
- Use a larger telecommunications dataset. There were 3,333 datapoints in this set, which is by no means a large dataset. A larger dataset will allow for more confidence in the predictive models covered in this notebook.
- Tune the hyperparameters in the model and re-adjust the model after more data has been collected.

- Ideally this model and various parameters would be adjusted and predictions would be created again after another time period has surpassed and customer churn rate can be re-evaluated.
- There are other macroeconomic variables in Syria that will affect customer retention. Syria is not the most stable of areas, and the EU has even imposed sanctions on SyriaTel in the past (reference: [https://www.reuters.com/article/us-syria-eu-sanctions-idUSTRE78N1DY20110924 (https://www.reuters.com/article/us-syria-eu-sanctions-idUSTRE78N1DY20110924)](https://www.reuters.com/article/us-syria-eu-sanctions-idUSTRE78N1DY20110924))