Phase 3 Project

Instructor: Morgan Jones
Student: Deanna Gould
Presentation Date: September 12, 2023
Class: Data Science Flex

Overview

SyriaTel Communications is looking to understand what contributes to their customer churn rate. Customer churn decreases revenue due to losing customers, and it also contributes to overall expenses in order to generate new customers. This model will provide insight as to why customers churn, and hopefully prevent this from happening.

This jupyter notebook will touch on a few different types of models; Logistic Regression, RandomForest, and XGBoost. This dataset consists of 3,333 rows, where 2,850 customers have not churned, and 483 customers have churned.

About the data

`state` : state the plan is in
`account length` : length of time the account has been open
`area code` : area code of the phone plan
`phone number` : phone number
`international plan` : indicator of international plan
`voicemail plan` : indicator of a voicemail plan
`number vmail messages` : number of voicemail messages
`total day minutes` : minutes spent during the day
`total day calls` : number of calls during the day
`total day charge` : total charge for day calls
`total eve minutes` : minutes spent during the evening
`total eve calls` : number of calls during the evening
`total eve charge` : total charge for evening calls
`total night minutes` : minutes spent at night
`total night calls` : amount of night calls
`total night charge` : total charge for night calls
`total intl minutes` : total international minutes
`total intl calls` : number of international calls
`total intl charge` : total international charge
`customer service calls` : amount of customer service calls
`churn` : whether a customer churns or not

Importing Libraries

In [1]:
```
# Import libraries for analysis
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

# Import libraries for processing
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import (confusion_matrix, classification_report, ConfusionM

# Import libraries for modeling
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV, train_test_split, cross_val_sc
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from xgboost import plot_importance
```

Creating Functions

In [2]:
```
# Creating a confusion matrix that will be used several times

def conf_matrix(estimator, Xtr, ytr):
    cm1 = plot_confusion_matrix(estimator, Xtr, ytr, cmap = 'flare')
    cm2 = plot_confusion_matrix(estimator, Xtr, ytr, normalize = 'true', cmap =
    return cm1, cm2
```

In [3]:
```
# Creating a function for grid scores

def grid_scores(estimator, Xtr, ytr):
    f1_score = estimator.best_score_
    best_params = estimator.best_params_
    best_estimator = estimator.best_estimator_.score(Xtr, ytr)
    print('Average F1 Score: ', f1_score)
    print('Best Parameters: ', best_params)
    print('Best Estimator Score: ', best_estimator)
```

Data Analysis

In [4]:
```
# Creating DataFrame

df = pd.read_csv('Data/churndata.csv')
df.head()
```

Out[4]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota eve call
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	...	11
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	8

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	tota ev call
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	12

5 rows × 21 columns

In [5]:
```
# Getting shape of DataFrame

df.shape
```

Out[5]:
```
(3333, 21)
```

In [6]:
```
# Checking what datatypes are in the df

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 [7]:
```
# Running describe and checking if anything stands out

df.describe()
```

Out[7]:

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tot mi
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0(
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.9;
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.7

	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	tot mi
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.0(
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.6(
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.4(
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.3(
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.7(

In [8]:
```
# Getting count of churn rate

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

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

In [9]:
```
# Finding churn rate percentages

df['churn'].value_counts(normalize=True)
```

Out[9]:
```
False    0.855086
True     0.144914
Name: churn, dtype: float64
```

In [10]:
```
# Creating a copy of the original df before making any changes

df2 = df.copy()
```

In [11]:
```
# Finding a count of null values

df2.isna().sum()
```

Out[11]:
```
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 [12]:

```
# Dropping columns that won't be relevant to the model

df2.drop(columns = ['area code', 'phone number'], axis=1, inplace=True)
```

In [13]:

```
# Looking at the df after making changes

df2.head()
```

Out[13]:

	state	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
0	KS	128	no	yes	25	265.1	110	45.07	197.4	99	16.78
1	OH	107	no	yes	26	161.6	123	27.47	195.5	103	16.62
2	NJ	137	no	no	0	243.4	114	41.38	121.2	110	10.30
3	OH	84	yes	no	0	299.4	71	50.90	61.9	88	5.26
4	OK	75	yes	no	0	166.7	113	28.34	148.3	122	12.61

In [14]:

```
# Changing text values to binary values

df2['international plan'], df2['voice mail plan'] = (df2['international plan'].m
                                                      df2['voice mail plan'].map(
```

In [15]:

```
# Checking the df again after making changes

df2.head()
```

Out[15]:

	state	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
0	KS	128	0	1	25	265.1	110	45.07	197.4	99	16.78
1	OH	107	0	1	26	161.6	123	27.47	195.5	103	16.62
2	NJ	137	0	0	0	243.4	114	41.38	121.2	110	10.30
3	OH	84	1	0	0	299.4	71	50.90	61.9	88	5.26
4	OK	75	1	0	0	166.7	113	28.34	148.3	122	12.61

In [16]:

```
# Changing churn from a boolean datatype to an integer

df2['churn'] = df2['churn'].astype('int64')
```

In [17]:

```
# Triple checking the df for changes made

df2.head()
```

Out[17]:

	state	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
0	KS	128	0	1	25	265.1	110	45.07	197.4	99	16.78
1	OH	107	0	1	26	161.6	123	27.47	195.5	103	16.62

	state	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
2	NJ	137	0	0	0	243.4	114	41.38	121.2	110	10.30
3	OH	84	1	0	0	299.4	71	50.90	61.9	88	5.26
4	OK	75	1	0	0	166.7	113	28.34	148.3	122	12.61

In [18]:

```
# Getting value counts for churn

df2['churn'].value_counts()
```

Out[18]:

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

In [19]:

```
# Comparing the value counts for international/voice mail plan between the curre

print(df['international plan'].value_counts())
print(df['voice mail plan'].value_counts())
print(df2['international plan'].value_counts())
print(df2['voice mail plan'].value_counts())
```

```
no     3010
yes     323
Name: international plan, dtype: int64
no     2411
yes     922
Name: voice mail plan, dtype: int64
0    3010
1     323
Name: international plan, dtype: int64
0    2411
1     922
Name: voice mail plan, dtype: int64
```

In [20]:

```
# Verifying the datatypes in the df

df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 19 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   international plan      3333 non-null   int64  
 3   voice mail plan         3333 non-null   int64  
 4   number vmail messages   3333 non-null   int64  
 5   total day minutes       3333 non-null   float64
 6   total day calls         3333 non-null   int64  
 7   total day charge        3333 non-null   float64
 8   total eve minutes       3333 non-null   float64
 9   total eve calls         3333 non-null   int64  
 10  total eve charge        3333 non-null   float64
 11  total night minutes     3333 non-null   float64
 12  total night calls       3333 non-null   int64  
 13  total night charge      3333 non-null   float64
 14  total intl minutes      3333 non-null   float64
 15  total intl calls        3333 non-null   int64  
```

```
 16  total intl charge       3333 non-null   float64
 17  customer service calls  3333 non-null   int64
 18  churn                   3333 non-null   int64
dtypes: float64(8), int64(10), object(1)
memory usage: 494.9+ KB
```

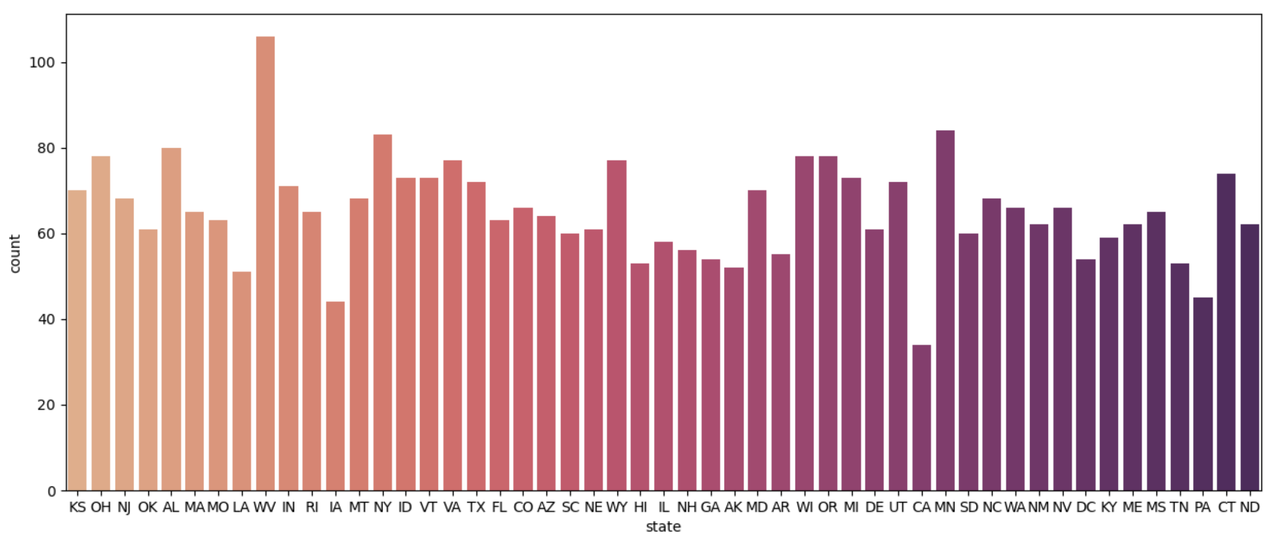
In [21]:
```
# Looking at the current df

df2
```

Out[21]:

	state	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	tc cha
0	KS	128	0	1	25	265.1	110	45.07	197.4	99	16
1	OH	107	0	1	26	161.6	123	27.47	195.5	103	16
2	NJ	137	0	0	0	243.4	114	41.38	121.2	110	10
3	OH	84	1	0	0	299.4	71	50.90	61.9	88	5
4	OK	75	1	0	0	166.7	113	28.34	148.3	122	12
...	...	...	...	...	...	...	...	...	...	...	
3328	AZ	192	0	1	36	156.2	77	26.55	215.5	126	18
3329	WV	68	0	0	0	231.1	57	39.29	153.4	55	13
3330	RI	28	0	0	0	180.8	109	30.74	288.8	58	24
3331	CT	184	1	0	0	213.8	105	36.35	159.6	84	13
3332	TN	74	0	1	25	234.4	113	39.85	265.9	82	22

3333 rows × 19 columns

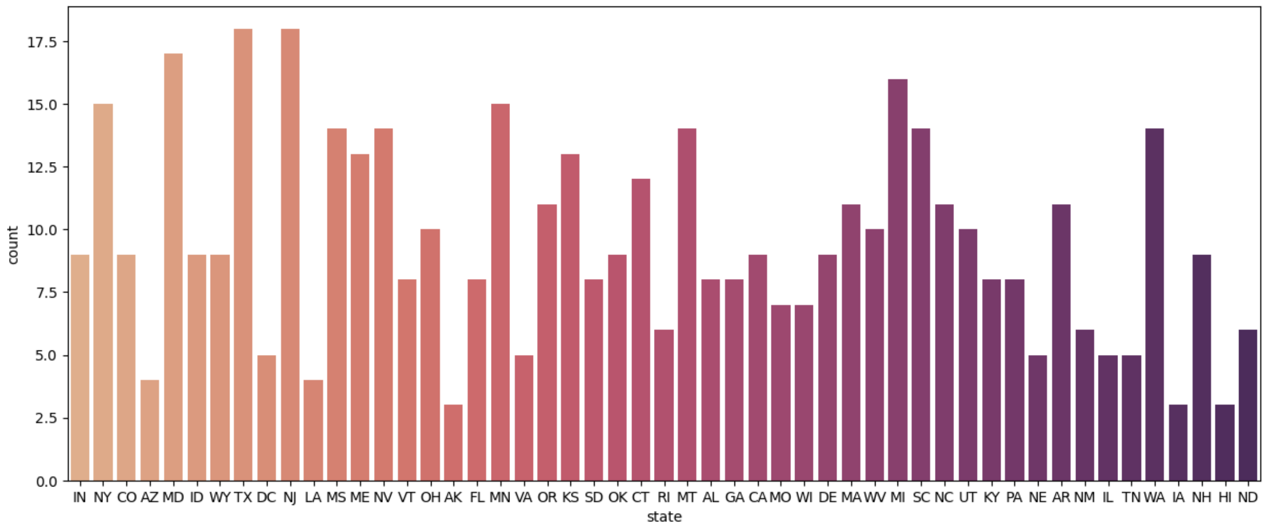
In [22]:
```
# Plotting the customers by state in a bar plot

fig, axes = plt.subplots(figsize = (15, 6))
sns.countplot(x = 'state', data=df2, palette='flare');
```



In [23]:
```
# Creating another df where churn is true
```

```
df3 = df2[df2['churn'] == 1]
```

In [24]:
```
# Checking to see if there is a higher churn based on state

fig, axes = plt.subplots(figsize = (15, 6))
sns.countplot(x = 'state', data = df3, palette = 'flare');
```



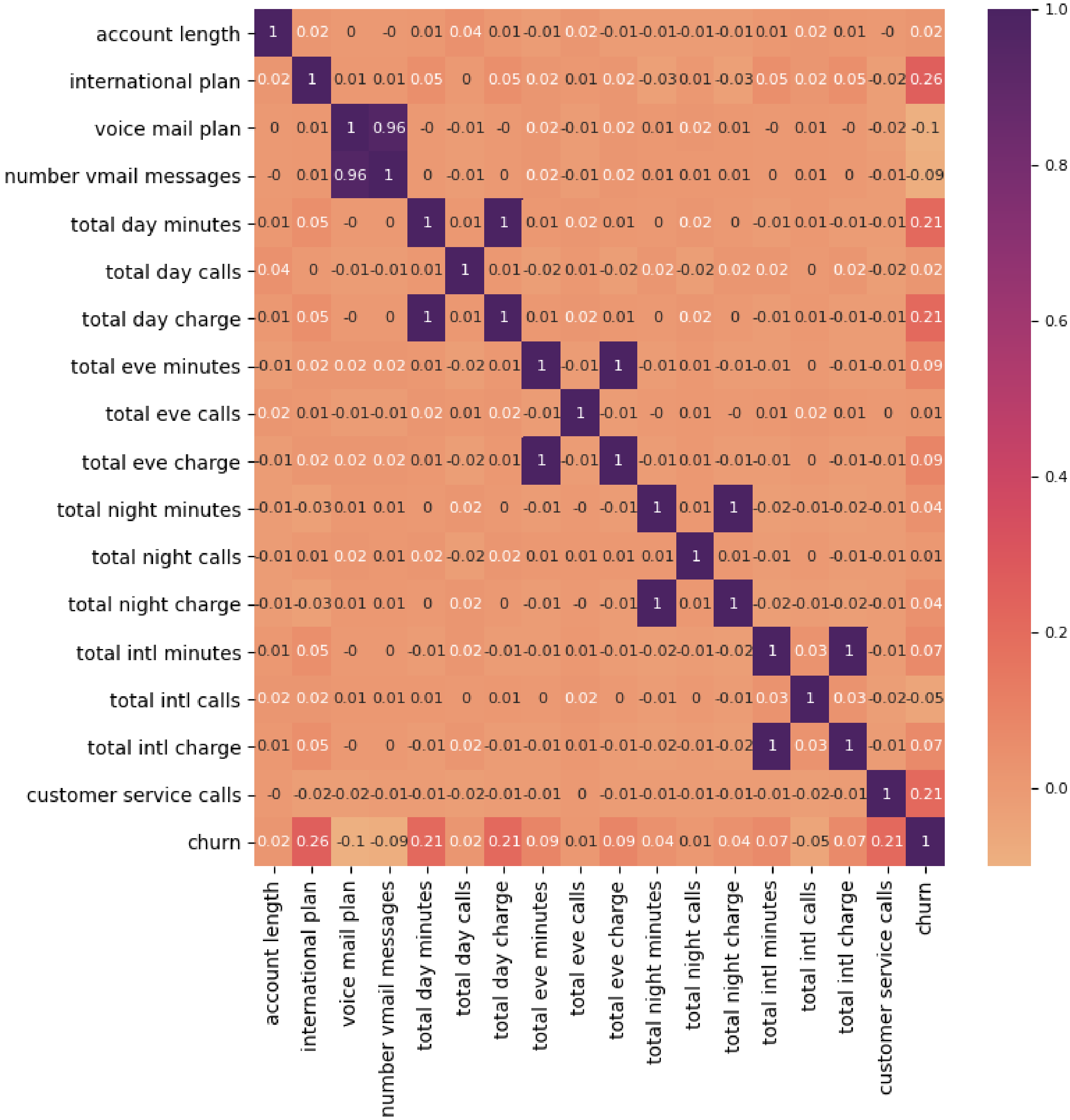
Based on the par plot above, there are some states with higher churn, but the bar plot is still hard to read, so I'm going to print the states with the highest churn.

In [25]:
```
# Sorting the new df by states with highest churn

state_churn = df2.groupby('state')['churn'].mean().reset_index()

# Creating a variable for the top 10 states
top_10 = state_churn.sort_values(by = 'churn', ascending = False).head(10)

print(top_10)
```

```
   state     churn
31    NJ  0.264706
4     CA  0.264706
43    TX  0.250000
20    MD  0.242857
40    SC  0.233333
22    MI  0.219178
25    MS  0.215385
33    NV  0.212121
47    WA  0.212121
21    ME  0.209677
```

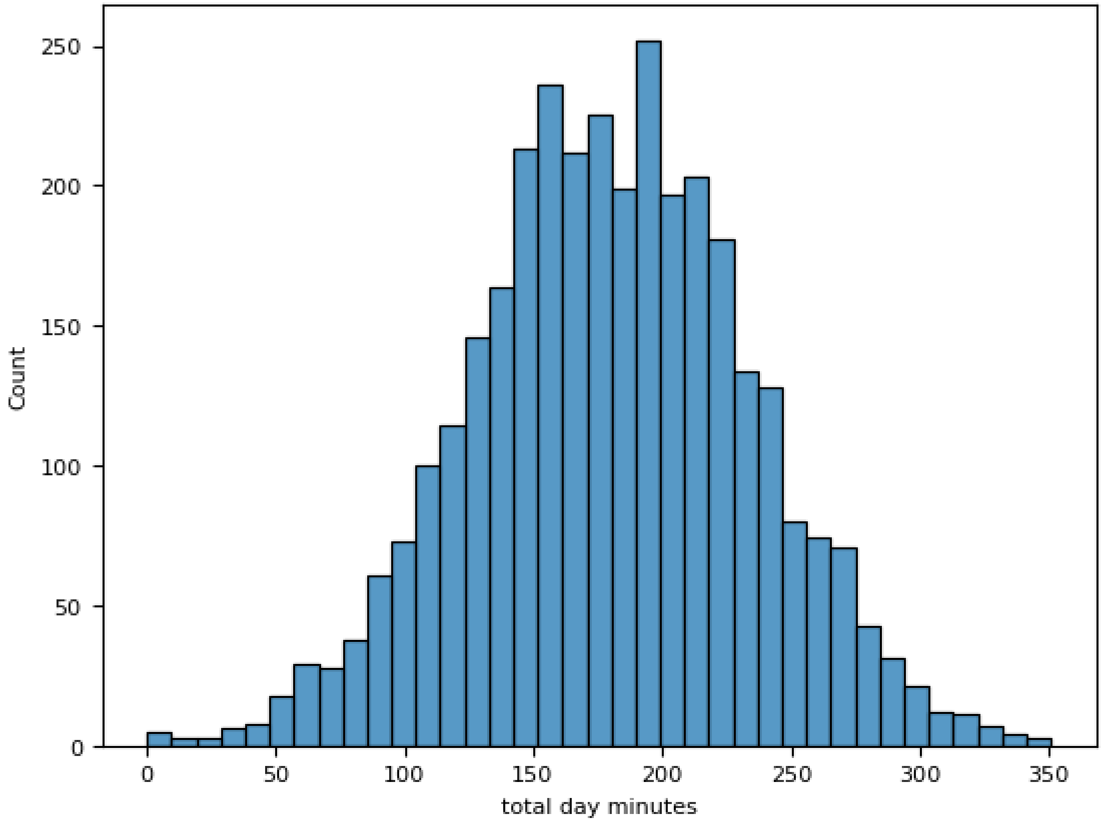
Based on the churn rates above, these states have extemely high churn. Considering the function created above based on some brief market research, I will be considering anything

In [26]:
```
# Creating a heatmap

fig, axes = plt.subplots(figsize = (8, 8))
plt.rcParams.update({'font.size':8})
sns.heatmap(data=df2.corr().round(2), cmap='flare', annot=True);
```



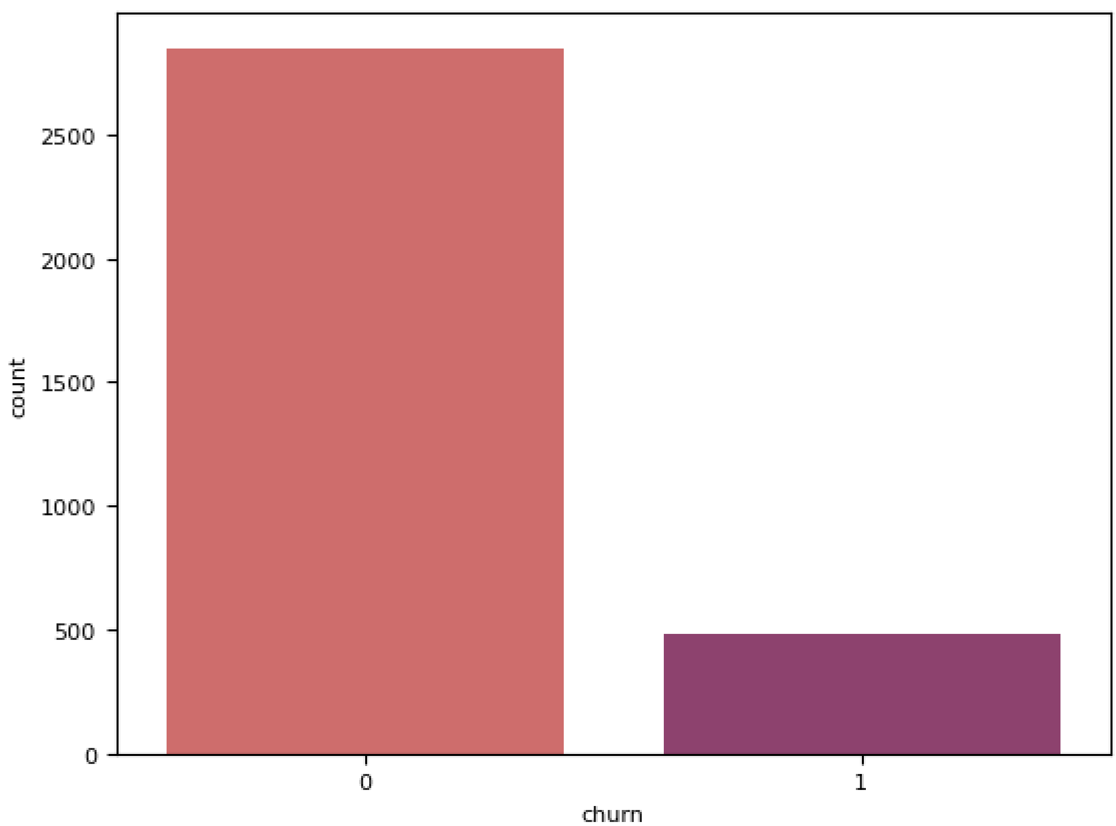
Based on the heatmap above, I can see that these are pay as you go phone plans, so charges are assoicated with having an international plan and the minutes used. Along with that, there is a correlation between customer service calls and churn.

In [27]:
```
# Creating a histogram to look at the distribution of total day minutes

ax = sns.histplot(x='total day minutes', data = df2)
```



In [28]:

```
# Visualizing the churn rate in a bar plot

sns.countplot(x='churn', data = df2, palette='flare');
```



Because the quantity of plans that churn is drastically different than the quantity of those that don't, I will have to keep classes in mind, and make sure they are a true represenative of the dataset.

Preprocessing

In [29]:
```
# Creating variables for a train test split

X = df2.drop('churn', axis = 1)
y = df2['churn']

# Calling the train test split on my data

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, rand
```

In [30]:
```
# Looking at the X_train df

X_train.head()
```

Out[30]:

	state	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	to cha
2654	ME	66	0	0	0	207.7	85	35.31	196.7	112	16
3162	UT	81	0	0	0	129.9	121	22.08	230.1	105	19
2333	NM	16	0	0	0	144.8	84	24.62	164.9	141	14
553	UT	61	1	0	0	78.2	103	13.29	195.9	149	16
1921	DE	136	0	0	0	101.7	105	17.29	202.8	99	17

In [31]:
```
# Separating the numerical and categorical variables in the X_train df

X_train_n = X_train.drop('state', axis = 1)
X_train_c = X_train[['state']]
```

In [32]:
```
# Looking at the numerical X_train df

X_train_n.head()
```

Out[32]:

	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	m
2654	66	0	0	0	207.7	85	35.31	196.7	112	16.72	
3162	81	0	0	0	129.9	121	22.08	230.1	105	19.56	
2333	16	0	0	0	144.8	84	24.62	164.9	141	14.02	
553	61	1	0	0	78.2	103	13.29	195.9	149	16.65	
1921	136	0	0	0	101.7	105	17.29	202.8	99	17.24	

In [33]:
```
# Checking that state is the only variable in the categorical df

X_train_c.head()
```

Out[33]:

	state
2654	ME
3162	UT
2333	NM
553	UT
1921	DE

In [34]:
```
# Establishing a pipieline use standard scaler and one hot encode the categorica

n_pipe = Pipeline(steps = [('scaler', StandardScaler())])
c_pipe = Pipeline(steps = [('ohe', OneHotEncoder(sparse = False, handle_unknown

# Calling ColumnTransformer to combine the num and cat df's after the prior step

cf = ColumnTransformer(transformers = [('state_pipe', c_pipe, X_train_c.columns)
```

Modeling

Logistic Regression Baseline

Model 1

In [35]:
```
# Creating a baseline model with logistic regression

lr_base = Pipeline(steps = [('cf', cf),
                            ('lr', LogisticRegression(class_weight = 'balanced',
```

In [36]:
```
# Fitting the baseline model by X_train and y_train

lr_base.fit(X_train, y_train)
```

Out[36]:
```
Pipeline(steps=[('cf',
                 ColumnTransformer(transformers=[('state_pipe',
                                                  Pipeline(steps=[('ohe',
                                                                   OneHotEncoder
(handle_unknown='ignore',

sparse=False))]),
                                                  Index(['state'], dtype='objec
t')),
                                                 ('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScale
r())]),
                                                  Index(['account length', 'inte
rnational 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'],
       dtype='object'))])),
                ('lr',
                 LogisticRegression(class_weight='balanced', random_state=1))])
```

In [37]:
```
# Predicting variables for the lr baseline and calling them

lr_base_pred = lr_base.predict(X_train)
lr_base_pred
```

Out[37]:
```
array([1, 0, 0, ..., 0, 1, 0])
```

Model 1 Check

In [38]:
```
# Getting the cross validation score and running it 10 times

lr_base_cv = cross_val_score(lr_base, X_train, y_train, scoring = 'f1', cv = 10)
print('CV Scores: ', lr_base_cv, '\n')
print('Average of CV Scores: ', lr_base_cv.mean())
```

```
CV Scores:  [0.42592593 0.5        0.52941176 0.39215686 0.47863248 0.43333333
 0.4137931  0.43396226 0.47863248 0.56842105]

Average of CV Scores:  0.4654269264205995
```

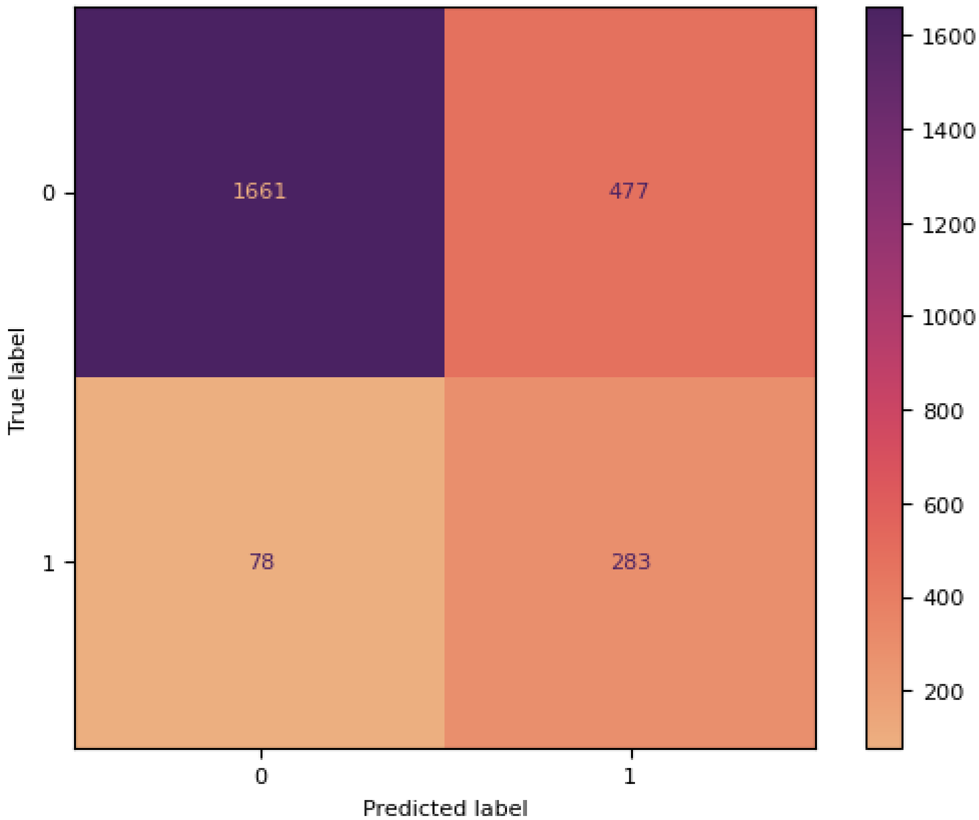
The average f1 score for the first model is 0.46, which is not a model that will accurately predict a test set, so I'm going to keep trying some different models, and also print a classification report to look at other scores for this model.

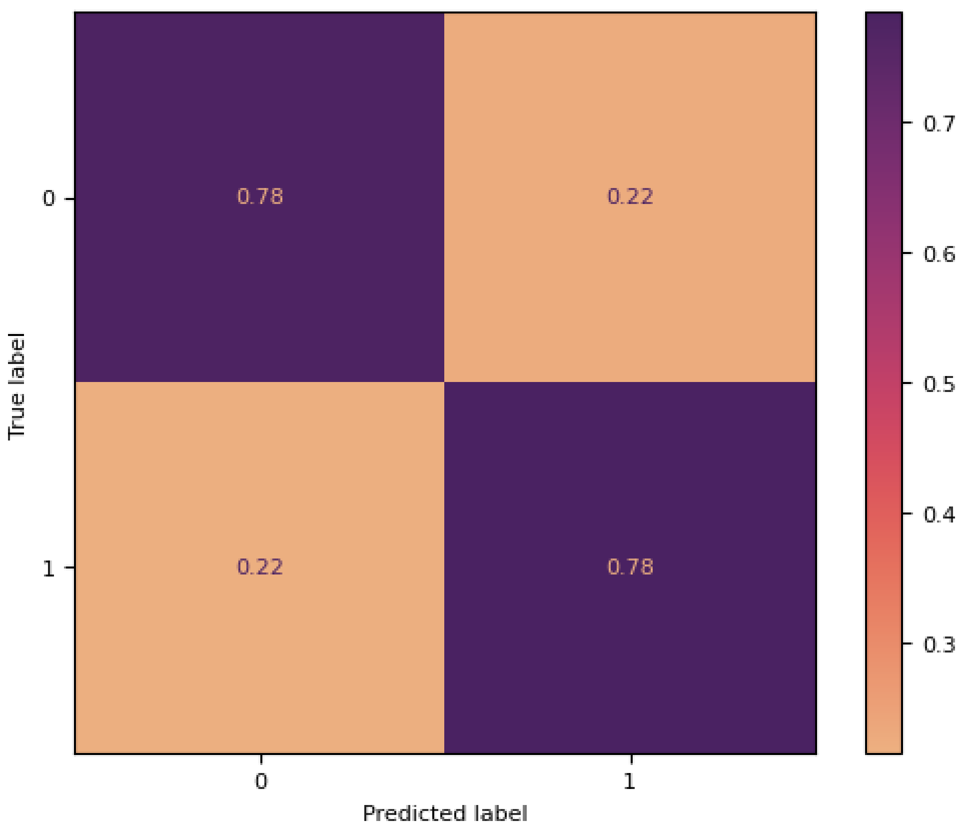
In [39]:
```
# Now performing classification report

print(classification_report(y_train, lr_base_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.78      0.86      2138
           1       0.37      0.78      0.50       361

    accuracy                           0.78      2499
   macro avg       0.66      0.78      0.68      2499
weighted avg       0.87      0.78      0.81      2499
```

Picking the correct measure to determine the effectiveness of a model is dependent on the business problem that needs to be solved. Precision is the percenage of true positives, accounting for false positives. That being said, the amount of false positives would be more important than the amount of false negatives or true negatives. Recall is a percentage of true positives that compares the amount of false negatives. F1 combines precision and recall. Accuracy considers everything, including true negatives, but because of that, the accuracy score can be inflated if there is a proportionally high amount of true negatives in the dataset. I will be using the F1 score to determine the efficacy of the models, and that is why I used the f1 score in my grid scores function as well.

In [40]:
```
# Plotting the confusion matrix of the train set

conf_matrix(lr_base, X_train, y_train);
```





The confusion matrix is a great way to visualize accuracy. In the bottom right section of the grid, there is the amount of true predicted positives. In this case, this is the amount of customers that were predicted to churn that *did* actually churn. In the top right section of the grid, is the amount of predicted customers that churned, but actually did *not* churn, also known as false positives. In the top left is the amount of customers that were predicted to churn, that did *not* churn (true negatives). In the bottom left is the number of people who were predicted to not churn, but did churn, which would be classified as false negatives. False negatives can be detrimental to SyriaTel.

Logistic Regression GridSearchCV

Model 2

In [41]:
```
# Define the grid

lrgrid = [{
    'lr__C': [0.0001, 0.001, 0.01, 0.1, 1.0],
    'lr__penalty': ['l1', 'l2'],
    'lr__solver': ['liblinear']
}]
```

In [42]:
```
# Define a grid search

lrgs = GridSearchCV(estimator = lr_base,
                    param_grid = lrgrid,
                    scoring = 'f1',
                    cv = 10
)
```

In [43]:
```
# Fitting the model

lrgs.fit(X_train, y_train)
```

Out[43]:
```
GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('cf',
                                        ColumnTransformer(transformers=[('state_
pipe',
                                                                         Pipelin
e(steps=[('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse=False))]),
                                                                         Index
(['state'], dtype='object')),
                                                                        ('num',
                                                                         Pipelin
e(steps=[('scaler',

StandardScaler())]),
                                                                         Index
(['account length', 'international plan', 'voice mail plan',
       'number vmail messages', 'total da...
       'total eve charge', 'total night minutes', 'total night calls',
       'total night charge', 'total intl minutes', 'total intl calls',
       'total intl charge', 'customer service calls'],
      dtype='object'))])),
                                       ('lr',
                                        LogisticRegression(class_weight='balance
d',
                                                           random_state=1))]),
             param_grid=[{'lr__C': [0.0001, 0.001, 0.01, 0.1, 1.0],
                          'lr__penalty': ['l1', 'l2'],
                          'lr__solver': ['liblinear']}],
             scoring='f1')
```

In [44]:
```python
# Predicting the model on the train set

lrgs_preds = lrgs.predict(X_train)
```

Model 2 Check

In [45]:
```python
# Printing the classification report for the log regression gridsearch model

print(classification_report(y_train, lrgs_preds))
```

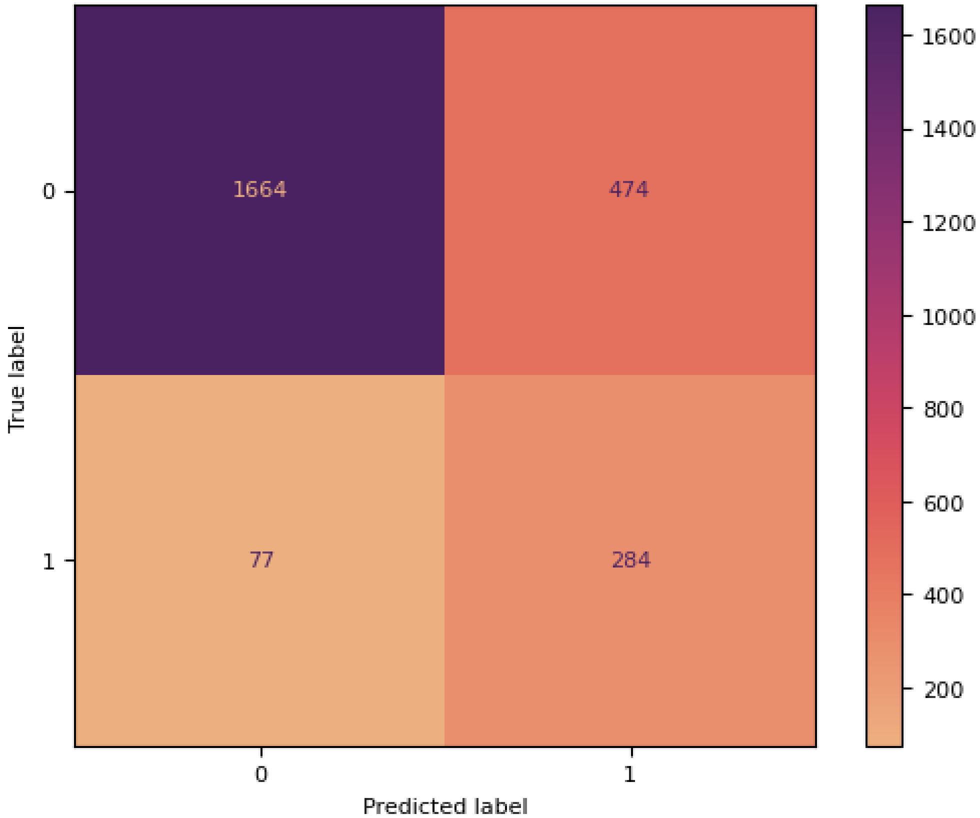
```
              precision    recall  f1-score   support

           0       0.96      0.78      0.86      2138
           1       0.37      0.79      0.51       361

    accuracy                           0.78      2499
   macro avg       0.67      0.78      0.68      2499
weighted avg       0.87      0.78      0.81      2499
```

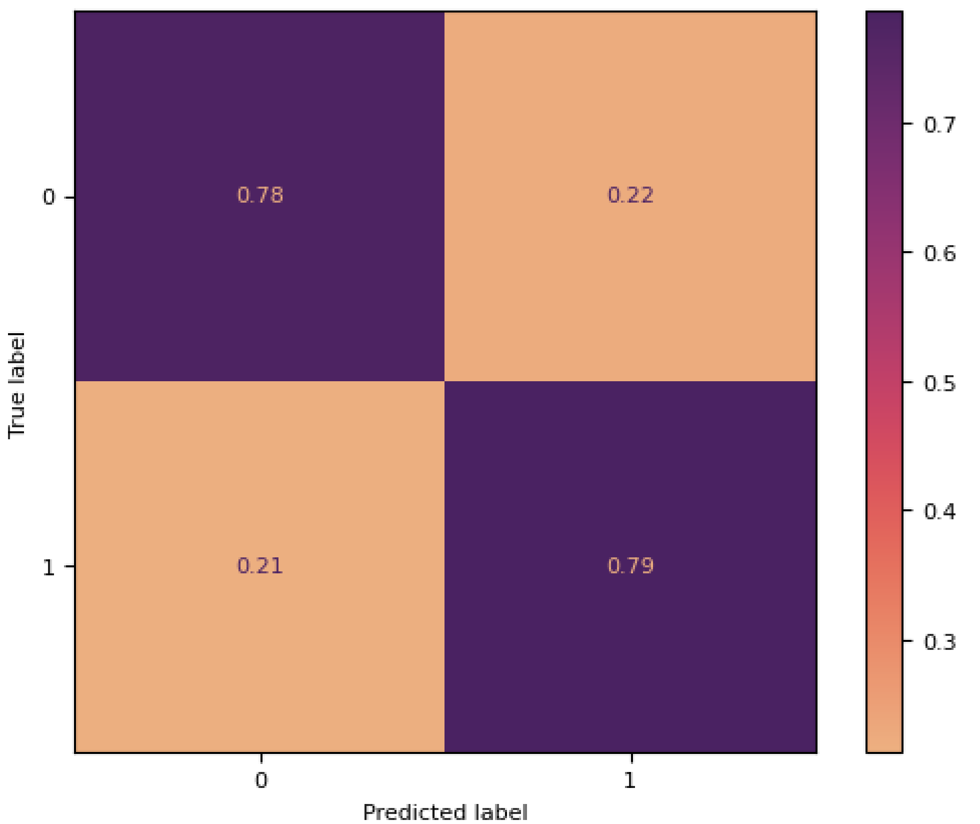
In [46]:
```python
# Calling the grid scores function

grid_scores(lrgs, X_train, y_train)
```

```
Average F1 Score:  0.4933176962847427
Best Parameters:  {'lr__C': 0.1, 'lr__penalty': 'l2', 'lr__solver': 'liblinear'}
Best Estimator Score:  0.7795118047218887
```

This F1 score is slightly better than the original logistic regression model, but still can't accurately predict a test set.

In [47]:
```
# Calling the confusion matrix function

conf_matrix(lrgs, X_train, y_train);
```





RandomForest Baseline

Model 3

In [48]:
```
# Creating a baseline for a RandomForest model

rf_base = Pipeline(steps = [('cf', cf),
                            ('rf', RandomForestClassifier(class_weight = 'balanc
```

In [49]:
```
# Fitting the rf baseline

rf_base.fit(X_train, y_train)
```

Out[49]:
```
Pipeline(steps=[('cf',
                 ColumnTransformer(transformers=[('state_pipe',
                                                  Pipeline(steps=[('ohe',
                                                                   OneHotEncoder
(handle_unknown='ignore',

sparse=False))]),
                                                  Index(['state'], dtype='objec
t')),
                                                 ('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScale
r())]),
                                                  Index(['account length', 'inte
rnational 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'],
      dtype='object'))])),
                ('rf',
                 RandomForestClassifier(class_weight='balanced',
                                        random_state=1))])
```

In [50]:
```
# Now performing classification report

print(classification_report(y_train, lr_base_pred))
```

```
              precision    recall  f1-score   support

           0       0.96      0.78      0.86      2138
           1       0.37      0.78      0.50       361

    accuracy                           0.78      2499
   macro avg       0.66      0.78      0.68      2499
weighted avg       0.87      0.78      0.81      2499
```

In [51]:
```
# Running .predict on the train set

rfbase_pred = rf_base.predict(X_train)
```

Model 3 Check

In [52]:
```
# Getting the cross validation score and running it 10 times

rf_base_cv = cross_val_score(rf_base, X_train, y_train, scoring = 'f1', cv = 10)
print('CV Scores: ', rf_base_cv, '\n')
print('Average of CV Scores: ', rf_base_cv.mean())
```

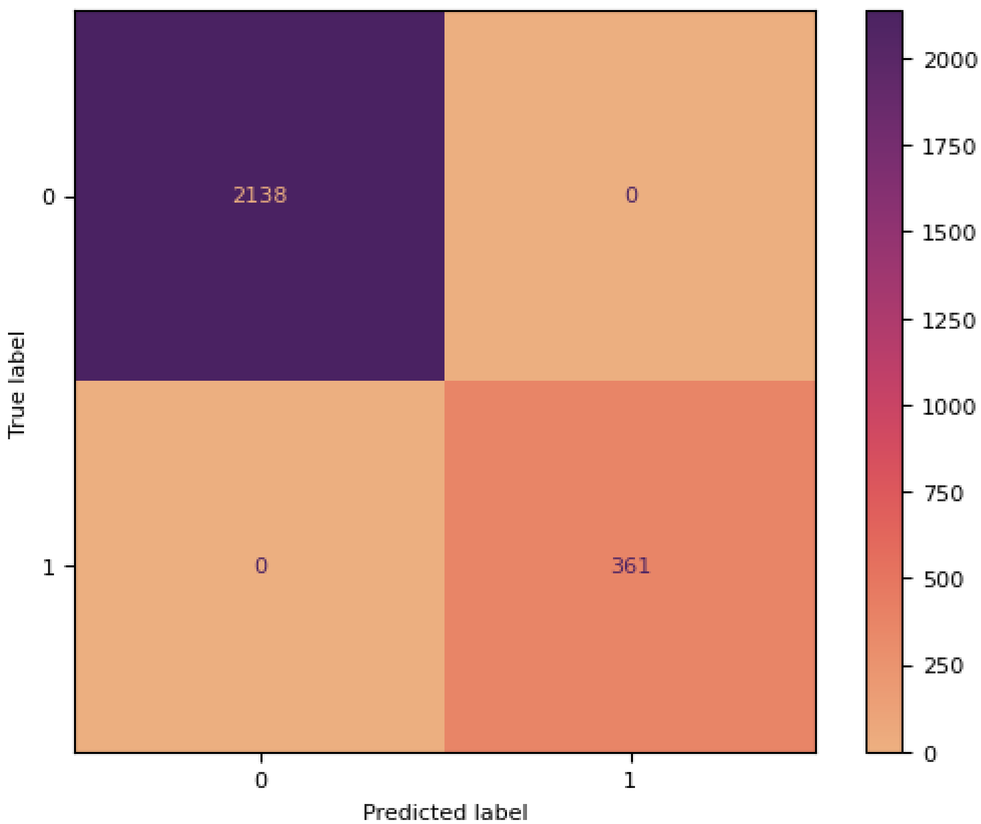
```
CV Scores:  [0.65454545 0.66666667 0.73684211 0.61538462 0.59259259 0.59259259
 0.57692308 0.75862069 0.79365079 0.65454545]

Average of CV Scores:  0.6642364041819577
```

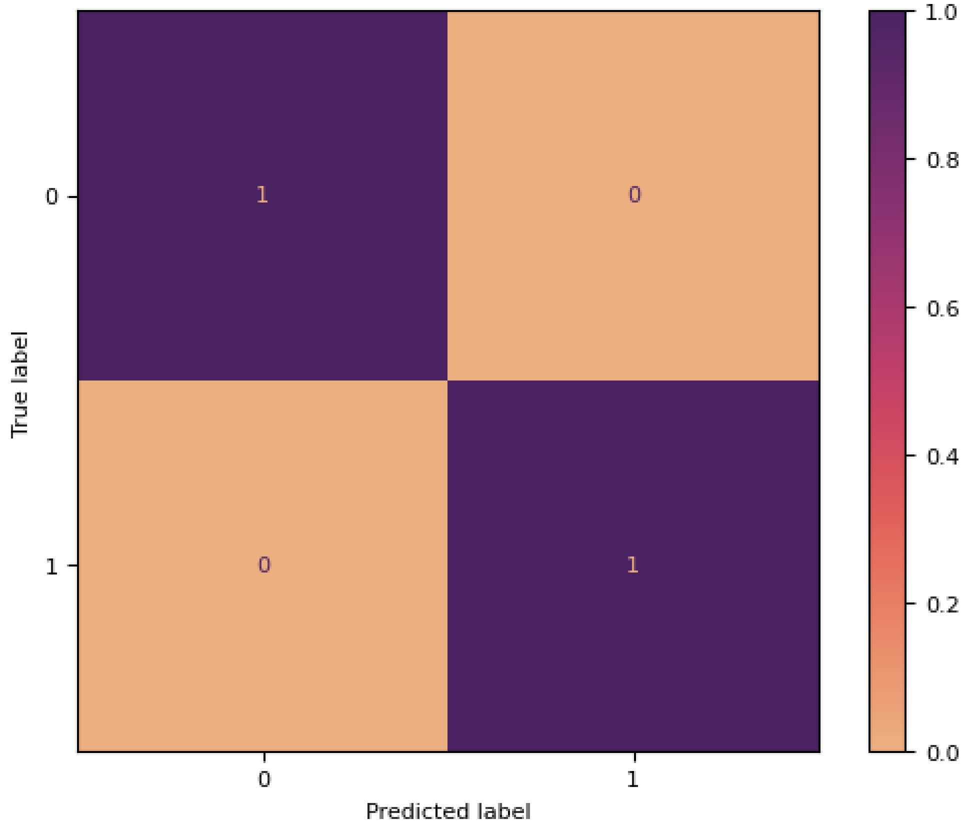
The F1 score from this model is much better than both of the logistic regression models, but still has room to improve. For the next version of the model, when I perform grid search, it will hopefully improve and I will also look at the accuracy score from the confusion matrix.

In [53]:
```
# Now performing classification report

print(classification_report(y_train, rfbase_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2138
           1       1.00      1.00      1.00       361

    accuracy                           1.00      2499
   macro avg       1.00      1.00      1.00      2499
weighted avg       1.00      1.00      1.00      2499
```

In [54]:
```
# Calling the confusion matrix

conf_matrix(rf_base, X_train, y_train);
```





Even though the accuracy score for this model is 1, that doesn't mean it's getting a true depiction of how the model will test. The model is likely overfitting.

RandomForest GridSearchCV

Model 4

In [55]:
```
# Defining rf grid

rfgridparams = [{'rf__max_depth': [3, 5, 7],
          'rf__min_samples_split': [2, 5, 10],
          'rf__min_samples_leaf': [3, 5, 7]
}]
```

In [56]:
```
# Defining the grid search for rf

rfgridsearch = GridSearchCV(estimator = rf_base,
                            param_grid = rfgridparams,
                            scoring = 'f1',
                            cv = 5)
```

In [57]:
```
# Running .fit on the rfgridsearch model

rfgridsearch.fit(X_train, y_train)
```

Out[57]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('cf',
                                        ColumnTransformer(transformers=[('state_
pipe',
                                                                          Pipelin
e(steps=[('ohe',
```

```
OneHotEncoder(handle_unknown='ignore',

sparse=False))]),
                                                                                    Index
(['state'], dtype='object')),
                                                                                   ('num',
                                                                                    Pipelin
e(steps=[('scaler',

StandardScaler())]),
                                                                                    Index
(['account length', 'international plan', 'voice mail plan',
       'number vmail messages', 'total day...
       'total eve charge', 'total night minutes', 'total night calls',
       'total night charge', 'total intl minutes', 'total intl calls',
       'total intl charge', 'customer service calls'],
      dtype='object'))])),
                                       ('rf',
                                        RandomForestClassifier(class_weight='bal
anced',
                                                                          random_state=
1))]),
             param_grid=[{'rf__max_depth': [3, 5, 7],
                          'rf__min_samples_leaf': [3, 5, 7],
                          'rf__min_samples_split': [2, 5, 10]}],
             scoring='f1')
```

In [58]:
```python
# Running .predict on the rfgridsearch model

rfgs_pred = rfgridsearch.predict(X_train)
```

Model 4 Check

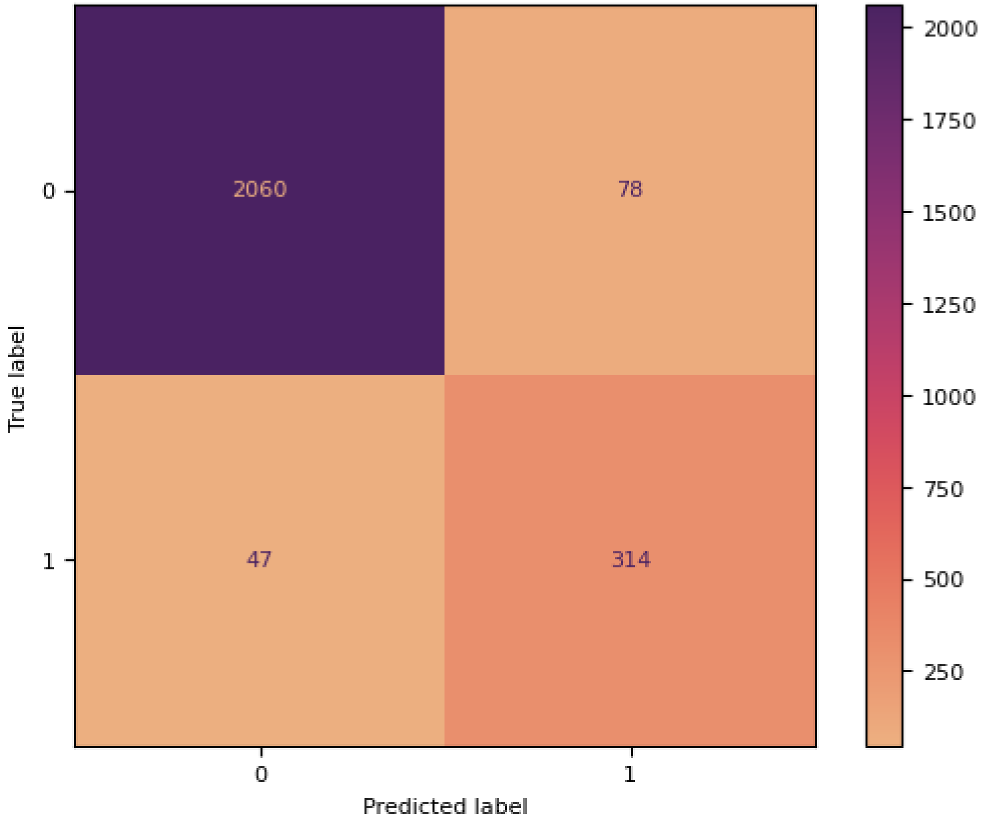
In [59]:
```python
# Calling the grid scores function

grid_scores(rfgridsearch, X_train, y_train)
```

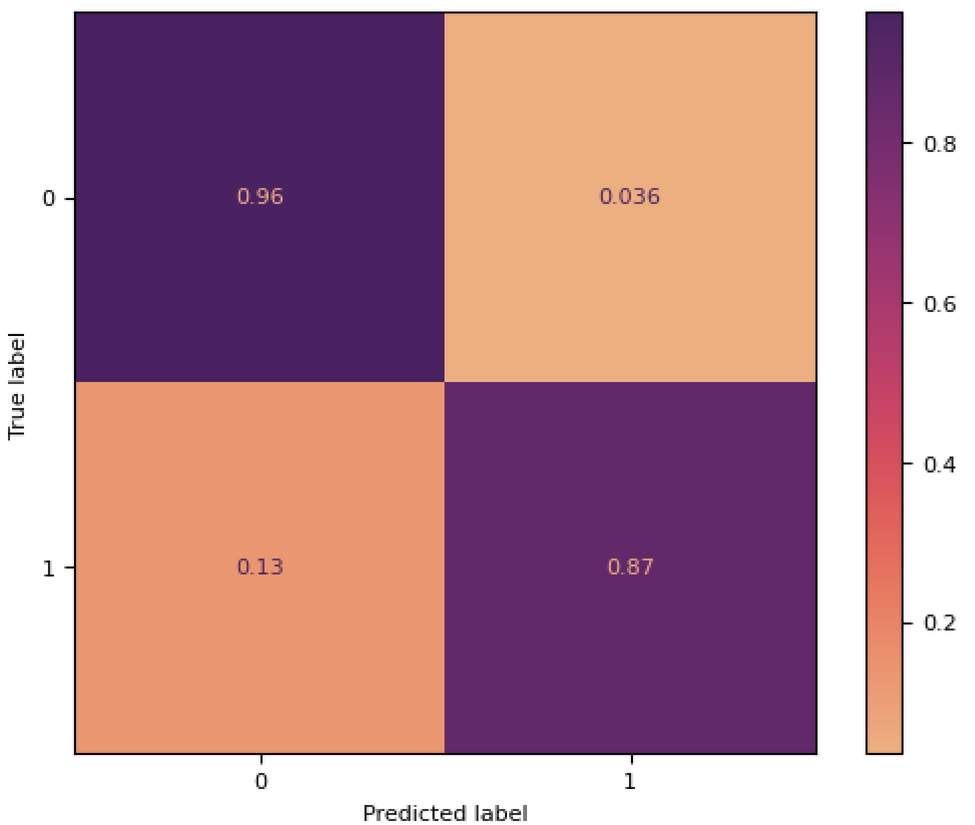
```
Average F1 Score:  0.7510796712723176
Best Parameters:  {'rf__max_depth': 7, 'rf__min_samples_leaf': 3, 'rf__min_sampl
es_split': 10}
Best Estimator Score:  0.9499799919967987
```

In [60]:
```python
# Printing the classification report

print(classification_report(y_train, rfgs_pred))
```

```
              precision    recall  f1-score   support

           0       0.98      0.96      0.97      2138
           1       0.80      0.87      0.83       361

    accuracy                           0.95      2499
   macro avg       0.89      0.92      0.90      2499
weighted avg       0.95      0.95      0.95      2499
```

In [61]:
```python
# Calling the confusion matrix

conf_matrix(rfgridsearch, X_train, y_train);
```





Using GridSearch has helped the RandomForest baseline drastically by bringing the F1 score to 0.75, and the accuracy score, which can be seen in the classification report as well as the confusion matrix above is 0.87. However, I'm going to try XGBoost to see if there is any improvement from here.

XGBoost Baseline

Model 5

In [62]:
```
# Importing counter to get an estimate for the xgb classifier

from collections import Counter

# count examples in each class

counter = Counter(y)

# estimate scale_pos_weight value

estimate = counter[0] / counter[1]
print('Estimate: %.3f' % estimate)
```

```
Estimate: 5.901
```

In [63]:
```
# Creating the pipeline for the XGBoost models

xgb_base = Pipeline(steps = [('cf', cf),
                             ('xgb', XGBClassifier(scale_pos_weight = estimate, r
```

In [64]:
```
# Fitting the baseline XGBoost model

xgb_base.fit(X_train, y_train)
```

Out[64]:
```
Pipeline(steps=[('cf',
                 ColumnTransformer(transformers=[('state_pipe',
                                                  Pipeline(steps=[('ohe',
                                                                   OneHotEncoder
(handle_unknown='ignore',

sparse=False))]),
                                                  Index(['state'], dtype='objec
t')),
                                                 ('num',
                                                  Pipeline(steps=[('scaler',
                                                                   StandardScale
r())]),
                                                  Index(['account length', 'inte
rnational plan', 'voice mail plan',
       'number vmail messages', 'total day minutes', 'total day calls',
       'to...
                               importance_type='gain',
                               interaction_constraints='',
                               learning_rate=0.300000012, max_delta_step=0,
                               max_depth=6, min_child_weight=1, missing=nan,
                               monotone_constraints='()', n_estimators=100,
                               n_jobs=0, num_parallel_tree=1, random_state=1,
                               reg_alpha=0, reg_lambda=1,
                               scale_pos_weight=5.900621118012422, subsample=1,
                               tree_method='exact', validate_parameters=1,
                               verbosity=None))])
```

In [65]:
```
# Using .predict on the train set

xgb_base_preds = xgb_base.predict(X_train)
```

Model 5 Check

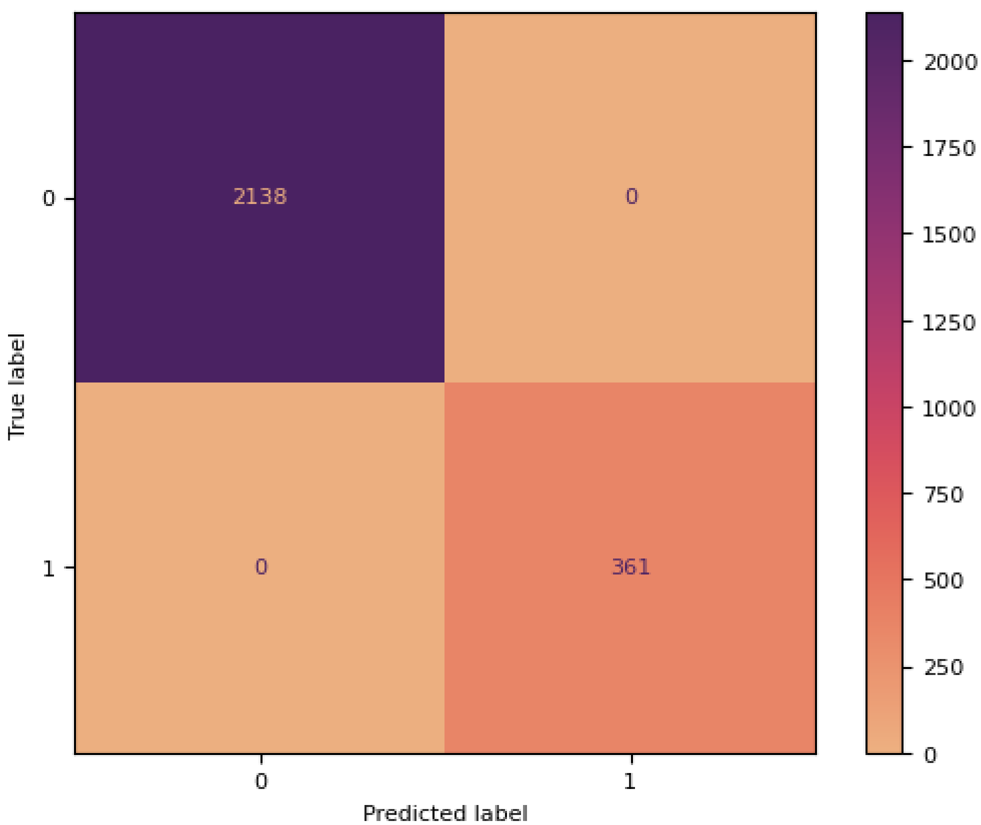
In [66]:
```
# Getting the cross validation score and running it 10 times

xgb_base_cv = cross_val_score(xgb_base, X_train, y_train, scoring = 'f1', cv = 1
print('CV Scores: ', xgb_base_cv, '\n')
print('Average of CV Scores: ', xgb_base_cv.mean())
```

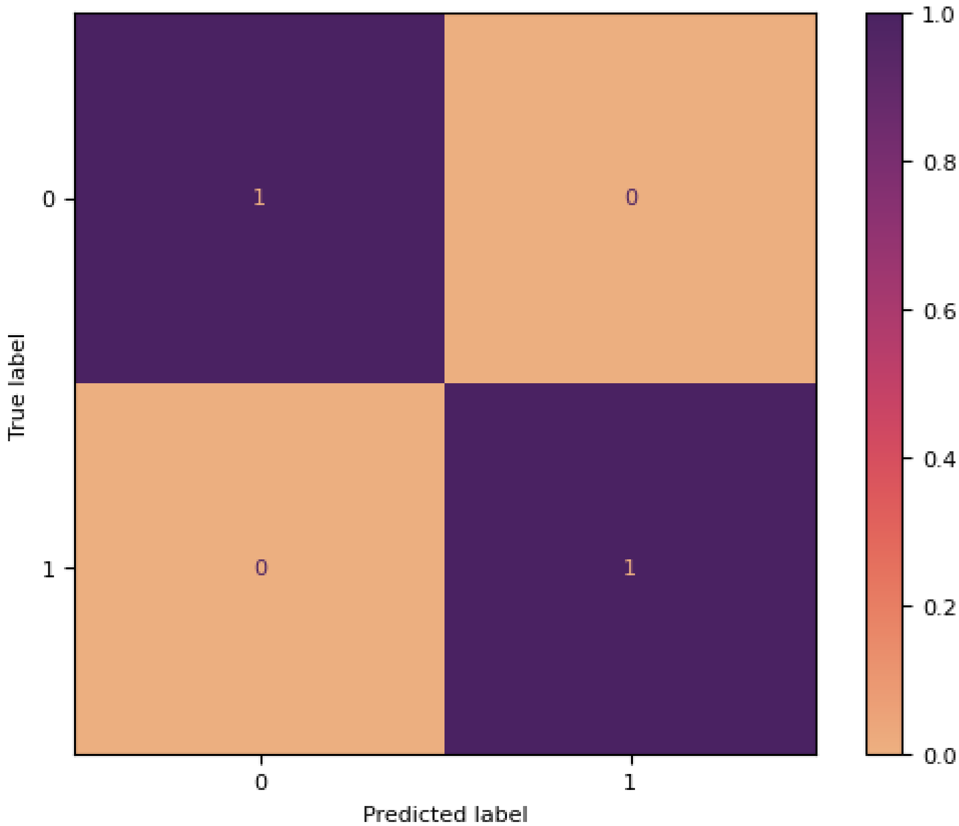
```
CV Scores:  [0.87878788 0.85714286 0.94444444 0.75       0.76470588 0.84057971
 0.78787879 0.79411765 0.88311688 0.82857143]

Average of CV Scores:  0.8329345519498972
```

In [67]:
```
# Now performing classification report

print(classification_report(y_train, xgb_base_preds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2138
           1       1.00      1.00      1.00       361

    accuracy                           1.00      2499
   macro avg       1.00      1.00      1.00      2499
weighted avg       1.00      1.00      1.00      2499
```

In [68]:
```
# Calling the confusion matrix again

conf_matrix(xgb_base, X_train, y_train);
```





Again, even though the accuracy score for this version of XGBoost is 1, it isn't a reliable score and is overfitting because cross validation hasn't been done. I will include GridSearchCV in my next version of XGBoost before I determine the best model to test the train set on.

XGBoost GridSearchCV

Model 6

In [69]:
```
# Creating parameters for the gridsearch

xgbgridparams = [{
    'xgb__max_depth': [3, 5, 7],
    'xgb__n_estimators': [100, 200, 300],
    'xgb__learning_rate': [0.1, 0.01, 0.001],
    'xgb__subsample': [0.8, 0.9, 1.0],
    'xgb__colsample_bytree': [0.8, 0.9, 1.0]
}]
```

In [70]:
```
# Creating the xgboost gridsearch model

xgbgridsearch = GridSearchCV(estimator = xgb_base,
                             param_grid = xgbgridparams,
                             scoring = 'f1',
                             cv = 5)
```

In [71]:
```
# Fitting the model

xgbgridsearch.fit(X_train, y_train)
```

Out[71]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('cf',
```

```
                                                        ColumnTransformer(transformers=[('state_
pipe',
                                                                                          Pipelin
e(steps=[('ohe',

OneHotEncoder(handle_unknown='ignore',

sparse=False))]),
                                                                                          Index
(['state'], dtype='object')),
                                                                                         ('num',
                                                                                          Pipelin
e(steps=[('scaler',

StandardScaler())]),
                                                                                          Index
(['account length', 'international plan', 'voice mail plan',
       'number vmail messages', 'total day...
                                                       num_parallel_tree=1,
                                                       random_state=1,
                                                       reg_alpha=0, reg_lambda=1,
                                                       scale_pos_weight=5.9006211
18012422,
                                                       subsample=1,
                                                       tree_method='exact',
                                                       validate_parameters=1,
                                                       verbosity=None))]),
             param_grid=[{'xgb__colsample_bytree': [0.8, 0.9, 1.0],
                          'xgb__learning_rate': [0.1, 0.01, 0.001],
                          'xgb__max_depth': [3, 5, 7],
                          'xgb__n_estimators': [100, 200, 300],
                          'xgb__subsample': [0.8, 0.9, 1.0]}],
             scoring='f1')
```

In [72]:
```python
# Calling .predict on the train set

xgb_gspreds = xgbgridsearch.predict(X_train)
```

Model 6 Check

In [73]:
```python
# Checking the model with the grid scores function

grid_scores(xgbgridsearch, X_train, y_train)
```

```
Average F1 Score:  0.8486047286047285
Best Parameters:  {'xgb__colsample_bytree': 0.8, 'xgb__learning_rate': 0.1, 'xgb
__max_depth': 5, 'xgb__n_estimators': 300, 'xgb__subsample': 0.9}
Best Estimator Score:  1.0
```

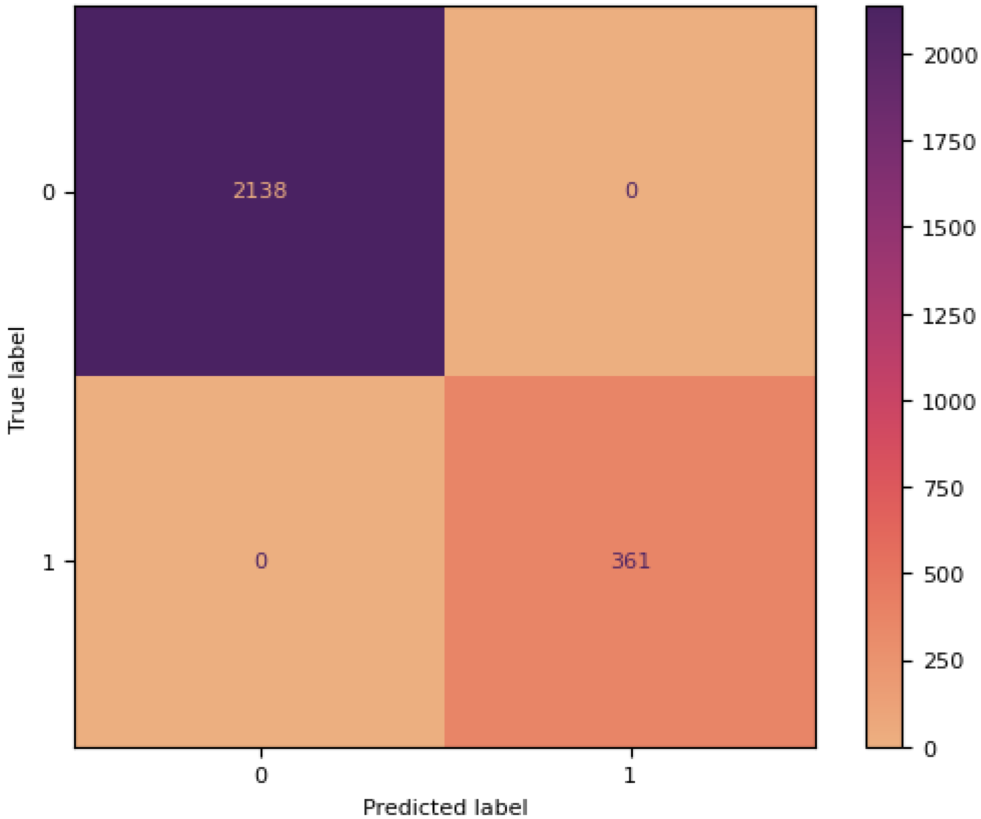
This model has provided my best F1 score yet, and since the F1 score is so high, the accuracy score being 1 isn't due to overfitting the model.

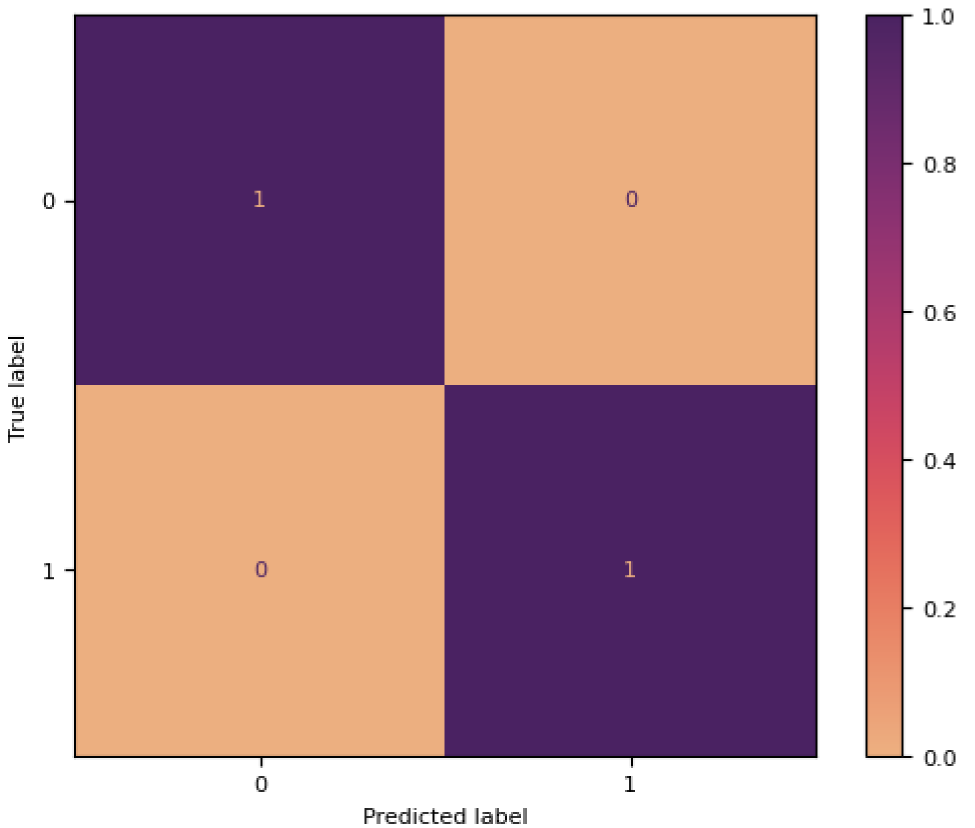
In [74]:
```python
# Printing the classification report

print(classification_report(y_train, xgb_gspreds))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2138
           1       1.00      1.00      1.00       361

    accuracy                           1.00      2499
   macro avg       1.00      1.00      1.00      2499
```

```
weighted avg       1.00      1.00      1.00      2499
```

In [75]:
```
# Calling the confusion matrix

conf_matrix(xgbgridsearch, X_train, y_train);
```





Best Model Tests

Now that I've tried three different algorithms, I'm going to test the test set on each one to determine which model is best to make predictions on.

Best Logistic Regression Model

Logistic Regression Grid Search Test

In [76]:
```
# Using .predict on the test set

lrgs_test_preds = lrgs.predict(X_test)
```

In [77]:
```
# Printing the classification report

print(classification_report(y_test, lrgs_test_preds))
```

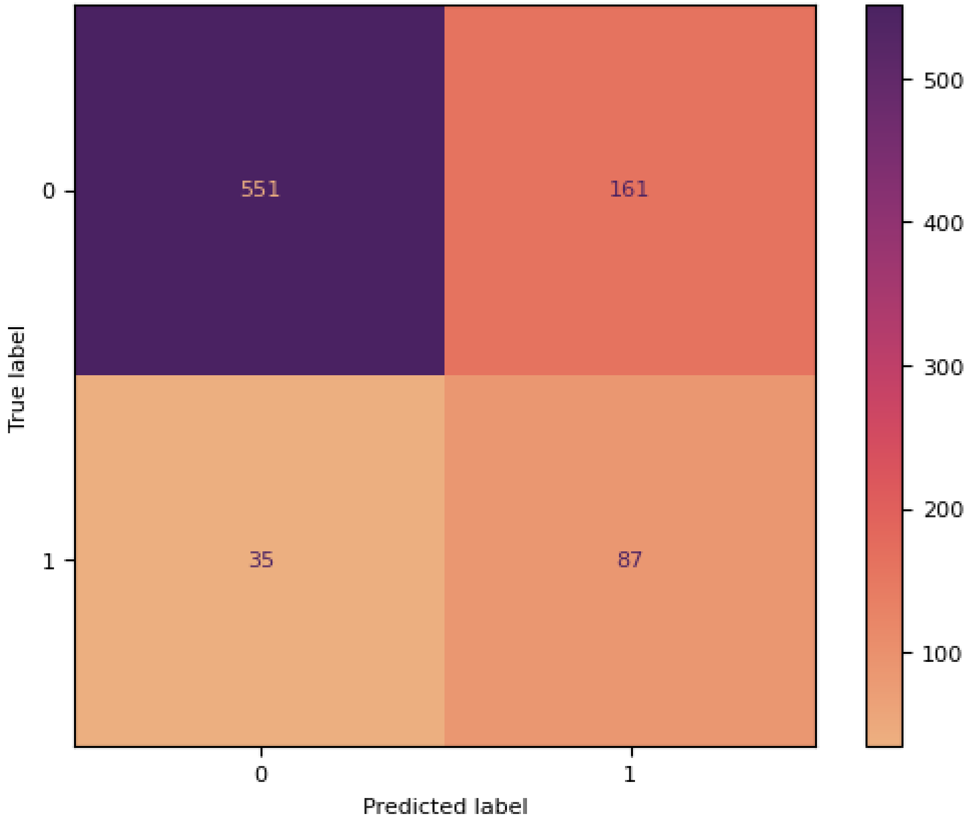
```
              precision    recall  f1-score   support

           0       0.94      0.77      0.85       712
           1       0.35      0.71      0.47       122

    accuracy                           0.76       834
   macro avg       0.65      0.74      0.66       834
weighted avg       0.85      0.76      0.79       834
```

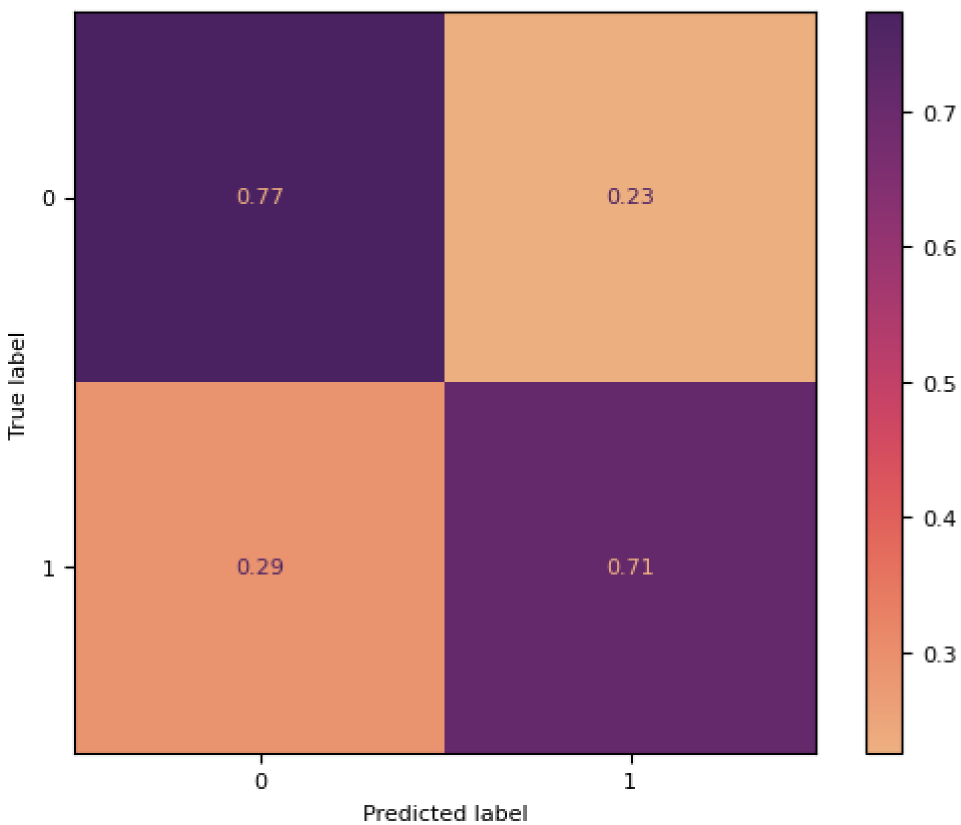
In [78]:
```
# Calling the grid scores function

grid_scores(lrgs, X_test, y_test)
```

```
Average F1 Score:  0.4933176962847427
Best Parameters:  {'lr__C': 0.1, 'lr__penalty': 'l2', 'lr__solver': 'liblinear'}
Best Estimator Score:  0.7649880095923262
```

An average F1 score of 0.49 is not a score that I feel comfortable making predictions on, even though the accuracy score is higher, which can be seen in the confusion matrix below as well.

In [79]:
```
conf_matrix(lrgs, X_test, y_test);
```





Since the test set performed worse than the train set for Logistic Regression Grid Search, the first model was likely overfitting on the train dataset.

Best RandomForest

In [80]:
```
# Predicting on the test set

rfgs_test_preds = rfgridsearch.predict(X_test)
```

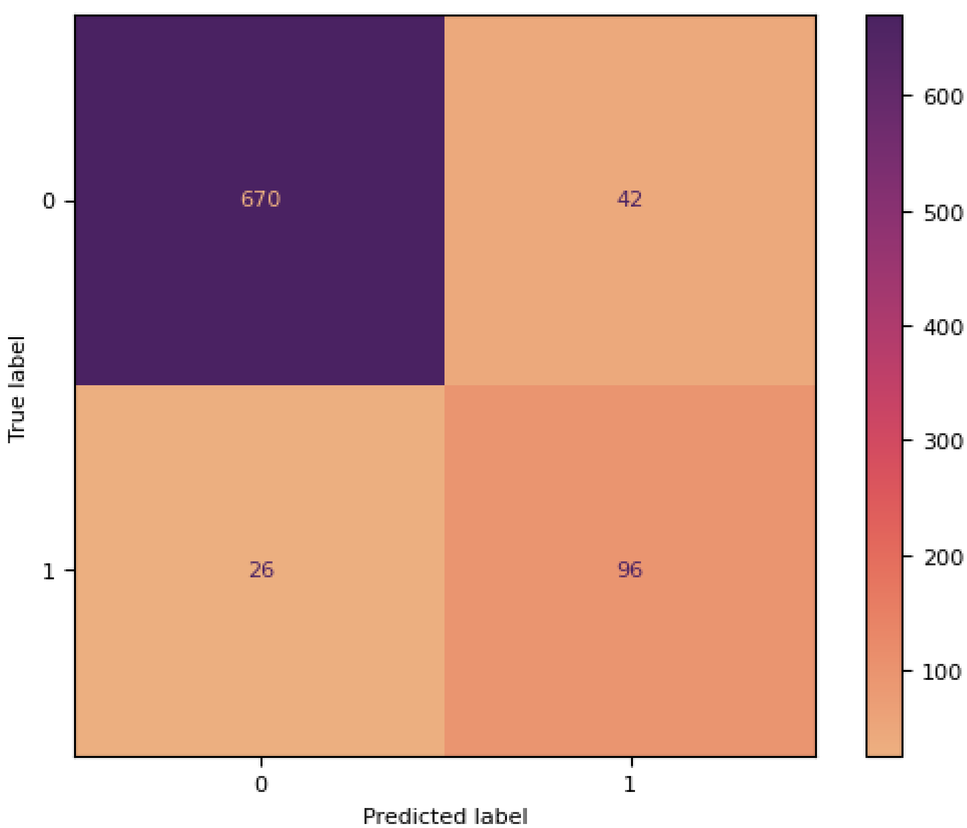
In [81]:
```
# Printing the classification report

print(classification_report(y_test, rfgs_test_preds))
```

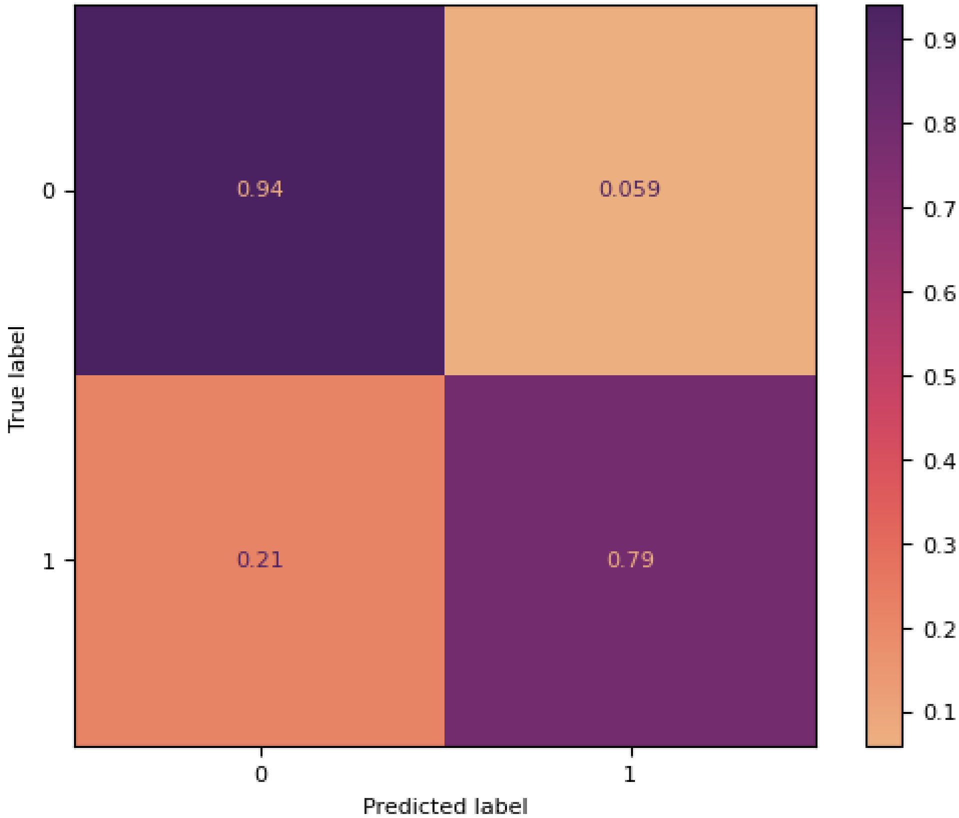
```
              precision    recall  f1-score   support

           0       0.96      0.94      0.95       712
           1       0.70      0.79      0.74       122

    accuracy                           0.92       834
   macro avg       0.83      0.86      0.85       834
weighted avg       0.92      0.92      0.92       834
```

In [82]:
```
# Calling the grid scores function

grid_scores(rfgridsearch, X_test, y_test)
```

```
Average F1 Score:  0.7510796712723176
Best Parameters:  {'rf__max_depth': 7, 'rf__min_samples_leaf': 3, 'rf__min_sampl
es_split': 10}
Best Estimator Score:  0.9184652278177458
```

In [83]:
```
# Calling the confusion matrix function

conf_matrix(rfgridsearch, X_test, y_test);
```





Best XGBoost

In [84]:
```
# Making predictions on the test set

xgb_test_preds = xgbgridsearch.predict(X_test)
```

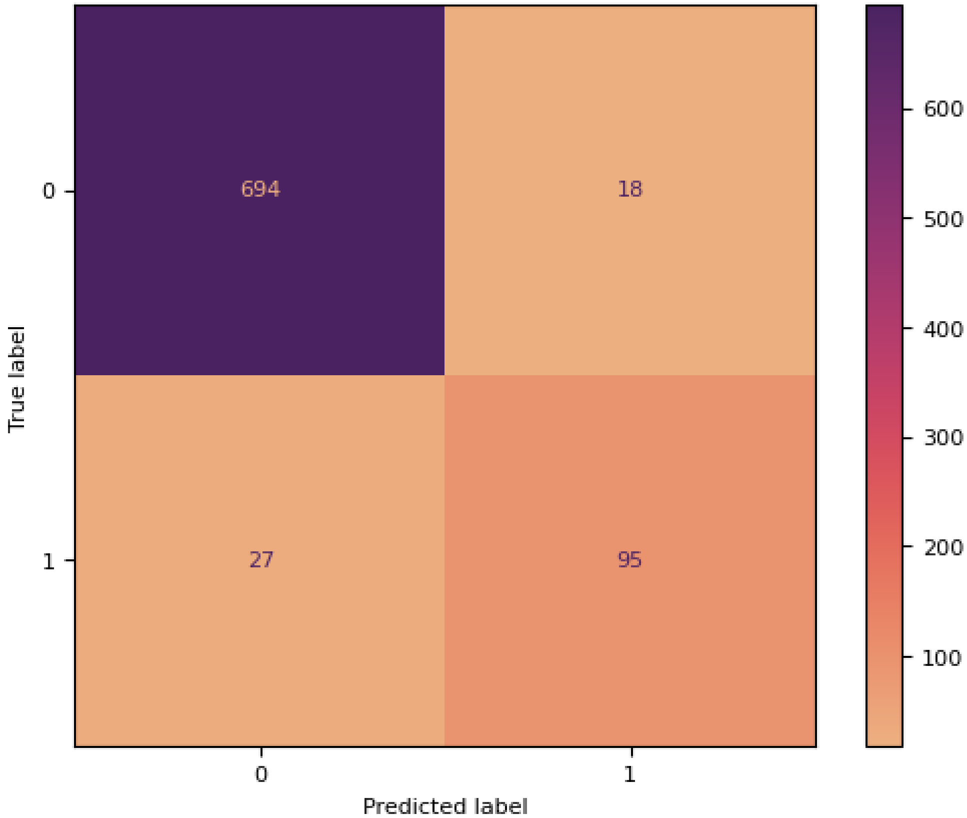
In [85]:
```
# Printing the classification report

print(classification_report(y_test, xgb_test_preds))
```

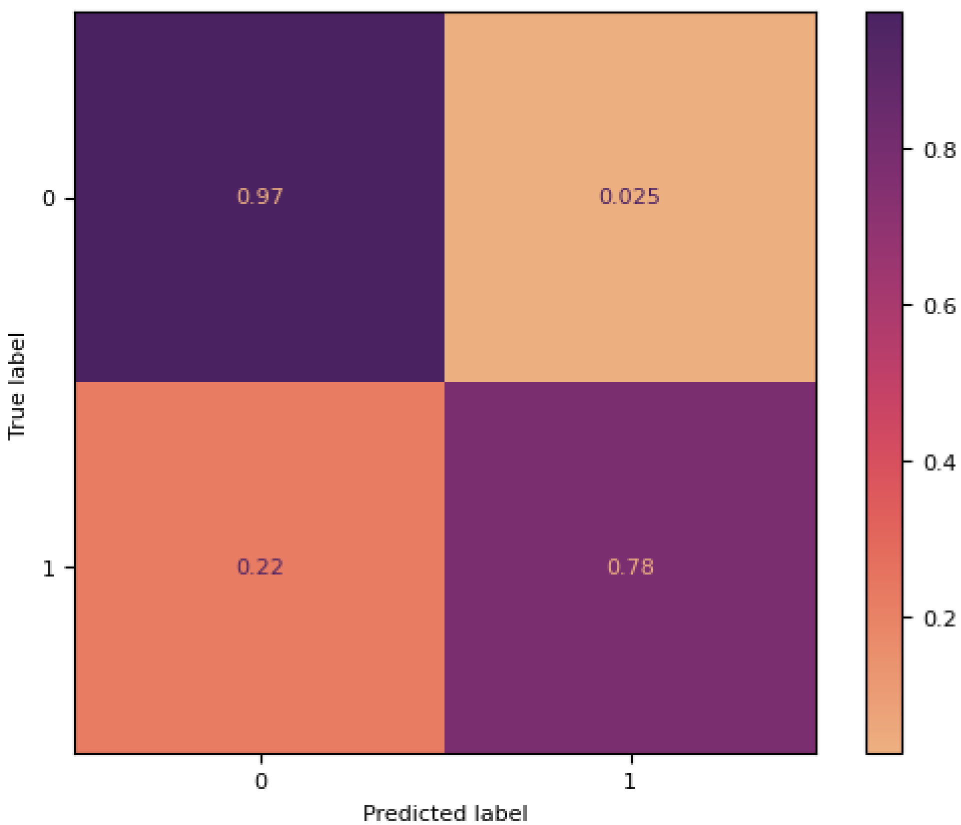
```
              precision    recall  f1-score   support

           0       0.96      0.97      0.97       712
           1       0.84      0.78      0.81       122

    accuracy                           0.95       834
   macro avg       0.90      0.88      0.89       834
weighted avg       0.94      0.95      0.95       834
```

In [86]:
```
# Calling the final grid scores function

grid_scores(xgbgridsearch, X_test, y_test)
```

```
Average F1 Score:  0.8486047286047285
Best Parameters:  {'xgb__colsample_bytree': 0.8, 'xgb__learning_rate': 0.1, 'xgb
__max_depth': 5, 'xgb__n_estimators': 300, 'xgb__subsample': 0.9}
Best Estimator Score:  0.9460431654676259
```

In [87]:
```
# Looking at the confusion matrices

conf_matrix(xgbgridsearch, X_test, y_test);
```





Exploring Best Model

XGBoost GridSearchCV performed the best on the test set, so I'm going to look into the predictions that I can gain from this model.

In [88]:

```
# Creating a variable and calling the best estimator method

xgb_model = xgbgridsearch.best_estimator_.named_steps['xgb']

# Creating a variable for feature importances

feature_importances = xgb_model.feature_importances_

# Get feature names from the one-hot encoder and standard scaler
ohe_features = (cf.named_transformers_['state_pipe'].named_steps["ohe"].get_feat

feature_names =  list(ohe_features) + list(X_train_n.columns)

# Create a dictionary of feature names and their importances
feature_importance_dict = dict(zip(feature_names, feature_importances))

# Print the feature names and their importances
for feature, importance in feature_importance_dict.items():
    print(f"{feature}: {importance}")
```

```
state_AK: 0.006379331927746534
state_AL: 0.005569261498749256
state_AR: 0.018804864957928658
state_AZ: 0.009688476100564003
state_CA: 0.02534690871834755
state_CO: 0.031444430351257324
state_CT: 0.017340730875730515
state_DC: 0.03316567465662956
state_DE: 0.001692904974333942
state_FL: 0.041745010763406754
state_GA: 0.0
state_HI: 0.0
state_IA: 0.0
state_ID: 0.011697923764586449
state_IL: 0.018375717103481293
state_IN: 0.0211940947920084
state_KS: 0.020016834139823914
state_KY: 0.0
state_LA: 0.01140831969678402
state_MA: 0.0
state_MD: 0.005986622069031
state_ME: 0.015439913608133793
state_MI: 0.003815826261416O776
state_MN: 0.0
state_MO: 0.018123386427760124
state_MS: 0.0147816622629761 7
state_MT: 0.014181727543473244
state_NC: 0.004359089769423008
state_ND: 0.011531082913279533
state_NE: 0.0
state_NH: 0.021404724568128586
state_NJ: 0.010708256624639034
state_NM: 0.007187947165220976
state_NV: 0.0
state_NY: 0.005386174190789461
state_OH: 0.013750975951552391
state_OK: 0.0
state_OR: 0.002271309494972229
state_PA: 0.020226968452334404
state_RI: 0.014171048067510128
state_SC: 0.010580657050013542
state_SD: 0.03244459256529808
state_TN: 0.0
```

```
state_TX: 0.017221080139279366
state_UT: 0.012109100818634033
state_VA: 0.0060489969328045845
state_VT: 0.006221654359251261
state_WA: 0.019996995106339455
state_WI: 0.006977899000048637
state_WV: 0.009278814308345318
state_WY: 0.0291795264929533
account length: 0.009914528578519821
international plan: 0.06908272206783295
voice mail plan: 0.03225608915090561
number vmail messages: 0.03520926460623741
total day minutes: 0.0328681617975235
total day calls: 0.01012807060033083
total day charge: 0.016810061410069466
total eve minutes: 0.01622755080461502
total eve calls: 0.011228754185140133
total eve charge: 0.01205381378531456
total night minutes: 0.011095334775745869
total night calls: 0.008796892128884792
total night charge: 0.009949104860424995
total intl minutes: 0.02049739845097065
total intl calls: 0.02201530709862709
total intl charge: 0.013967745937407017
customer service calls: 0.060642797499895096
```

In [103…

```
# Putting the feature importances into a dataframe

ftr_importance = pd.DataFrame(({'Columns': feature_names, 'Importances': feature
```

In [104…

```
# Viewing the dataframe

ftr_importance
```

Out[104…

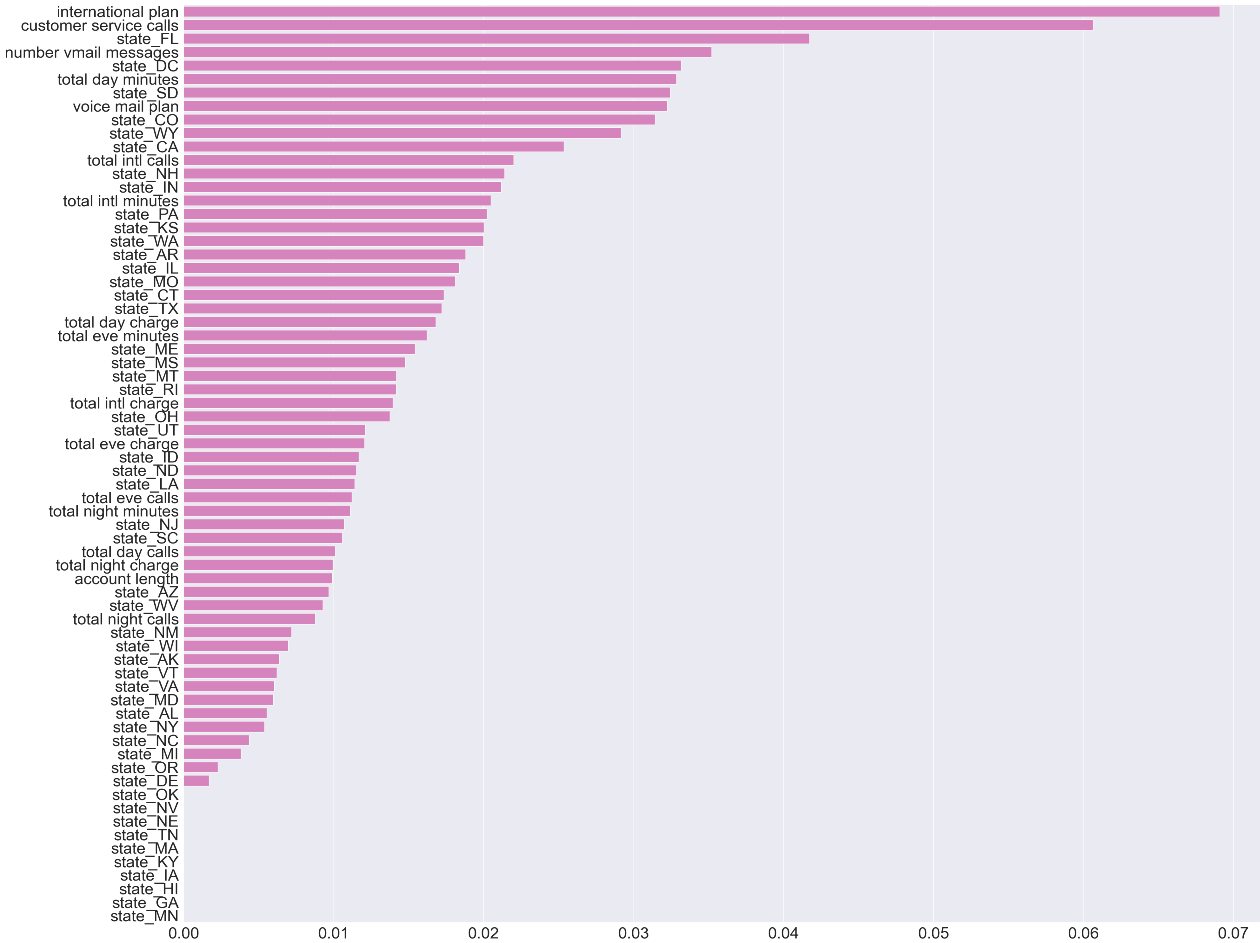
	Columns	Importances
0	state_AK	0.006379
1	state_AL	0.005569
2	state_AR	0.018805
3	state_AZ	0.009688
4	state_CA	0.025347
...	...	...
63	total night charge	0.009949
64	total intl minutes	0.020497
65	total intl calls	0.022015
66	total intl charge	0.013968
67	customer service calls	0.060643

68 rows × 2 columns

In [105…

```
# Plotting the features on a bar plot

sns.barplot(x=feature_importances, y=feature_names, data=ftr_importance, color="
            ftr_importance.sort_values('Importances', ascending = False).Columns
```

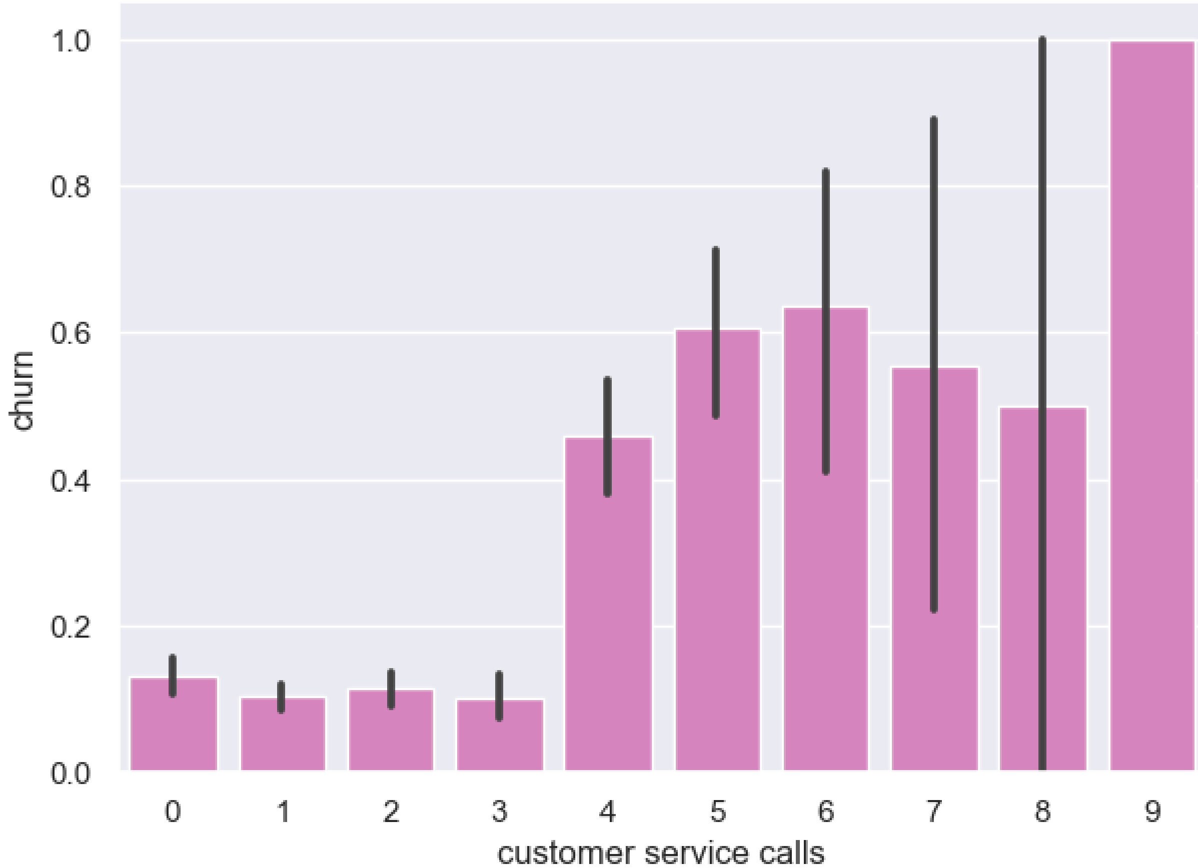
```
sns.set(rc={'figure.figsize':(40,34)},font_scale = 3)
plt.show();
```



Based on the barplot above, I can see all of the feature importances ranked most important to least important. That being said, I can't say whether or not they have positive or negative impacts, but I can make assumptions and realize their level of importance.

In [107…

```
# Plotting churn based on the amount of customer service calls

sns.barplot(x = df2['customer service calls'], y = df2['churn'], color = 'tab:pi
sns.set(rc={'figure.figsize':(5,5)},font_scale= 1)
plt.show();
```



Conclusion

In [108…

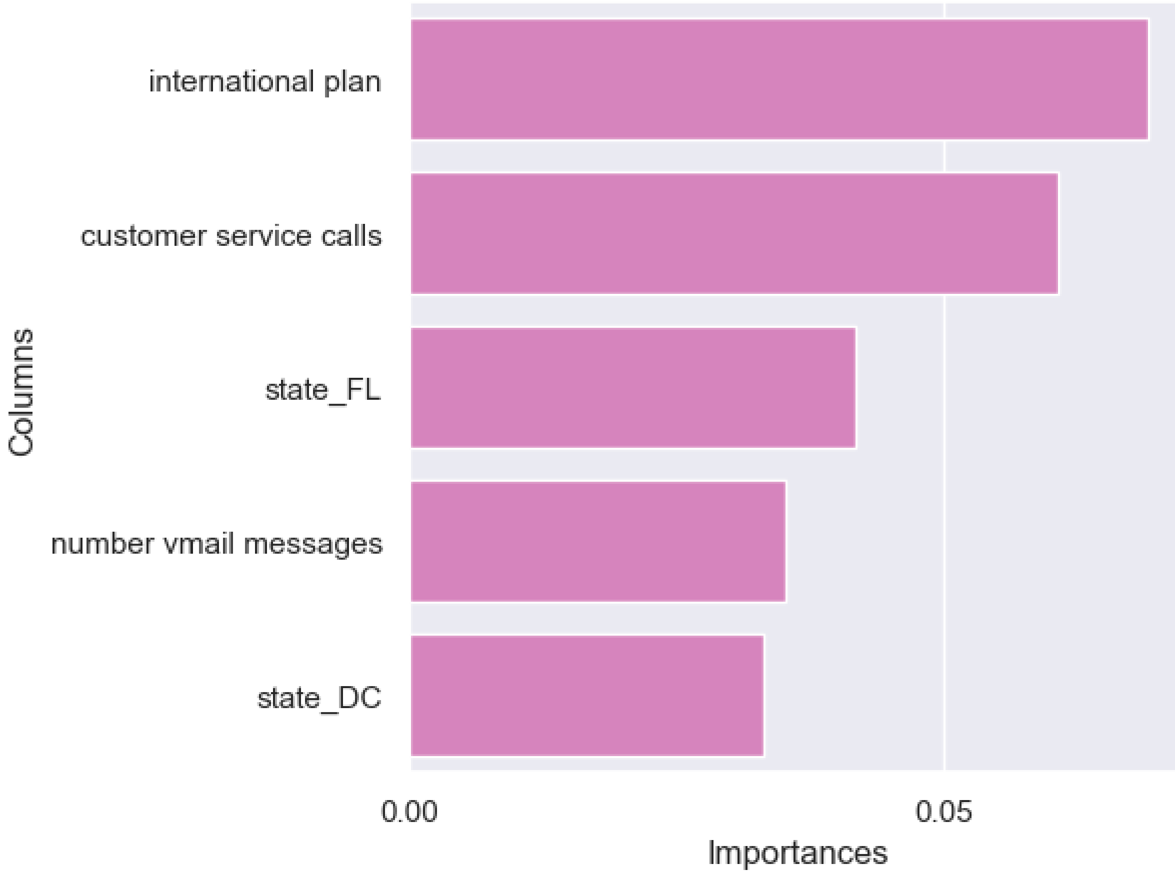
```
# Sorting the feature importances
top5_ftrs = ftr_importance.sort_values(by = 'Importances', ascending = False).he
top5_ftrs
```

Out[108…

	Columns	Importances
52	international plan	0.069083
67	customer service calls	0.060643
9	state_FL	0.041745
54	number vmail messages	0.035209
7	state_DC	0.033166

In [109…

```
# Sorting the feature importances
top5_ftrs = ftr_importance.sort_values(by = 'Importances', ascending = False).he

# Plotting the top 5 features on a bar plot

sns.barplot(x='Importances', y='Columns', data=top5_ftrs, color="tab:pink", orde
            top5_ftrs.sort_values('Importances', ascending = False).Columns, ori
sns.set(rc={'figure.figsize':(40,34)},font_scale = 3)
plt.show();
```



Based on this barplot of the top 5 feature importances, I can show more clearly that some contributing factors are having an international plan, the amount of customer service calls, the state of Florida and DC, as well as the number of voicemail messages. I'm not sure if SyriaTel charges an additional amount for the number of voicemail messages, but this model is depicting it as a top 5 feature.

With that being said, SyriaTel could look more into where their customers are making most of their calls to internationally, and either have a promotion, or lower costs in general to those countries. In addition to that, if customer service calls aren't recorded already, they may want to look into doing so for training purposes. This could be something that NLP could be used for if they do record those calls. There may also be more competition in DC and Florida for international plans, so a promotiion in those states could be beneficial to SyriaTel. If there is in fact a charge for the number of voicemail messages, it may be worth making a voicemail plan slightly more expensive to cause less of a shock in case someone receives a lot of voicemails within the month.