Final Project Submission

- Student name: Heath Rittler
- Student pace: Self paced
- Scheduled project review date/time: 1/31/2023 0930
- Instructor name: Mark Barbour
- Blog post URL: https://medium.com/@heathlikethecandybar/

Introduction

Business Case/ Summary

SyriaTel, a telecommunications company wants to identify the leading factors of why a customer cancels their service. This is also referred to as 'Churn.' If they understand the factors that lead to churn, the company can implement programs to reduce the risk of churn, and increase the lifetime value of and for their customers.

My goal is to build a classifer to predict whether a customer will stop doing business with SyriaTel. I will be using information such as usage, interactions with SyriaTel, and certain features that the customer has purchased. I am mostly focused on reducing the rate of false negatives so the metric in which I will be evaluating my models is called Recall.

Core Field Names and Definitions from Data Source

- `state` - The state in which the account owner resides.
- `account length` -
- `area code` - Primary 3 digit area of the line for the account.
- `phone number` - Primary 7 digit area of the line for the account.
- `international plan` - Indicator denoting whether or not the account has an international feature.
- `voice mail plan` - Indicator denoting whether or not the account has an voice mail feature.
- `number vmail messages` - Usage metric counting the total number of voicemails for the phone number in question.
- `total day minutes` - Usage metric indicating how many minutes (call time) were used between 6:00am and 5:00pm.
- `total day calls` - Usage metric indicating how many calls were used between 6:00am and 5:00pm.

- `total day charge` - Usage metric indicating how much the user was charged for their usage between 6:00am and 5:00pm.
- `total eve minutes` - Usage metric indicating how many minutes (call time) were used between 5:01pm and 8:00pm.
- `total eve calls` - Usage metric indicating how many calls were used between 5:01pm and 8:00pm.
- `total eve charge` - Usage metric indicating how much the user was charged for their usage between 5:01pm and 8:00pm.
- `total night minutes` - Usage metric indicating how many minutes (call time) were used between 8:01pm and 5:59am.
- `total night calls` - Usage metric indicating how many calls were used between 8:01pm and 5:59am.
- `total night charge` - Usage metric indicating how much the user was charged for their usage between 8:01pm and 5:59a.
- `total intl minutes` - Usage metric indicating how many minutes (call time) were used internationally.
- `total intl calls` - Usage metric indicating how many calls were made internationally.
- `total intl charge` - Usage metric indicating how much the user was charged for their international call usage.
- `customer service calls` - The total number of customer service calls made by the user to the Skyvia Customer Service line.
- `churn` - Our target category indicating whether or not the customer churned/ cancelled their plan.

Additional information about the dataset can be found here: https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset

Data Load, Cleaning

Importing Packages

In [278…
```python
# Importing packages for analysis

import pandas as pd
import numpy as np
import seaborn as sns
import statsmodels.api as sm

from matplotlib import pyplot as plt
import matplotlib.ticker as mtick
from matplotlib.colors import ListedColormap
%matplotlib inline

from sklearn.linear_model import LinearRegression, LogisticRegression, RidgeClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler, FunctionTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import MissingIndicator, SimpleImputer
```

```python
from sklearn.compose import ColumnTransformer
from sklearn.dummy import DummyClassifier
from sklearn.metrics import plot_confusion_matrix, confusion_matrix, plot_roc_curve, classification_report, plc
from sklearn.metrics import precision_score, recall_score, accuracy_score, f1_score, precision_recall_curve, rc
from sklearn.neighbors import KNeighborsClassifier, NearestNeighbors
from sklearn.ensemble import AdaBoostRegressor, GradientBoostingRegressor, AdaBoostClassifier, GradientBoosting
from sklearn.feature_selection import RFECV
import xgboost as xgb
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, plot_tree
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImPipeline

plt.style.use('seaborn-talk')
```

Choosing Colors & Templates

In [279...
```python
# Choosing standard colors for project

pal = sns.color_palette("viridis")

color_codes = ['purple', 'darkblue', 'blue', 'bluegreen', 'green', 'lime']

my_cmap = ListedColormap(sns.color_palette(pal).as_hex())

pal.as_hex()
```

Out[279...

Import data

In [280...
```python
# Importing data and viewing the first 5 rows

df = pd.read_csv('data/data.csv')

df.head()
```

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	m
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	

5 rows × 21 columns

```python
# Updating column names to have an _ vs a space
df.columns = [c.replace(' ', '_') for c in df.columns]
```

```python
# Taking a quick peak at our datatypes for any transformations needed.
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
```

Looks like we have a few categorical variables in which we will need to encode. We'll look at that here in a bit. First I want to make sure our dataset is complete. Moving on to checking to see if we need to remove or impute any data before any transformations.

In [283…
```
# Checking for missing values.

df.isna().sum()
```

Out[283…
```
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
```

No missing values. Going to create some columns to add together all of our minutes, calls, and charges. We might be able to reduce the number of dimensions we have if the customer is not using the plan at all. We may drop these later, but let's add them while we are in the mood.

In [284…
```
# Adding column for total calls
```

```python
df['total_calls'] = (df['total_day_calls'] +
                     df['total_eve_calls'] +
                     df['total_night_calls'] +
                     df['total_intl_calls']
                     )
```

In [285...
```python
# Adding column for total minutes

df['total_charges'] = (df['total_day_charge'] +
                       df['total_eve_charge'] +
                       df['total_night_charge'] +
                       df['total_intl_charge']
                        )
```

In [286...
```python
# Adding column for total charges

df['total_minutes'] = (df['total_day_minutes'] +
                       df['total_eve_minutes'] +
                       df['total_night_minutes'] +
                       df['total_intl_minutes']
                        )
```

In [287...
```python
# Adding column for price per minute

df['price_per_minute'] = df['total_day_charge']/df['total_minutes']
```

In [288...
```python
# Quick check to make sure our columns were added correctly and math checks out

df.head()
```

Out[288...

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes
0	KS	128	415	382-4657	no	yes	25	265.1
1	OH	107	415	371-7191	no	yes	26	161.6
2	NJ	137	415	358-1921	no	no	0	243.4
3	OH	84	408	375-9999	yes	no	0	299.4
4	OK	75	415	330-6626	yes	no	0	166.7

5 rows × 25 columns

Calculations are looking good. These will primarily serve as EDA helper stats. Since they are formulas based on existing fields, I would assume that we will see a high level of colinearity within these added metrics.

```
In [289… df.describe()
```

Out[289…

	account_length	area_code	number_vmail_messages	total_day_minutes	total_day_calls	total_day_charge	total_eve_minutes
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	437.182418	8.099010	179.775098	100.435644	30.562307	200.980348
std	39.822106	42.371290	13.688365	54.467389	20.069084	9.259435	50.713844
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	408.000000	0.000000	143.700000	87.000000	24.430000	166.600000
50%	101.000000	415.000000	0.000000	179.400000	101.000000	30.500000	201.400000
75%	127.000000	510.000000	20.000000	216.400000	114.000000	36.790000	235.300000
max	243.000000	510.000000	51.000000	350.800000	165.000000	59.640000	363.700000

Across the board we can already tell that our data is pretty well distributed. We can see this by a quick glance at the relationship between our mean and our median (50%) values. This will reduce the amount of data cleaning, and imputation that we will need to complete later. Roughly looks like our daytime minutes are more expensive than our evening, and night time minutes. Looks like customers interact with customer service on average 1.6 times. Customers pay on average $60 in total, and on average .05 cents a minute. What is difficult to discern from this data set is if this is across one month, or a different time period. That would help us determine how we institute our strategy from a timing perspective. However, we will just discuss the strategies in general, and think of the time as an arbitrary component.

Features of the cell phone plan EDA

Since we know that our target variable is imbalanced (churn accounts for roughly 15% of the total dataset). There are only a few other features that are tied to the account besides the usage, and those are voice_mail_plan, international_plan, and customer_service_calls. Let's dig into those next!

There are 323 accounts that purchased a international plan (roughly 10% of customers.). Going to take a look to see if churn within the international accounts is similar to those accounts that did not purchase international plans.

```
In [290… # Checking our target/ dataset for balance before creating our baseline classifcation model.
```

```python
pd.pivot_table(df,
               values='account_length',
               index='international_plan',
               columns='churn',
               aggfunc='count'
              )
```

churn	False	True
international_plan		
no	2664	346
yes	186	137

As we can see, churn within our customer segment that has the international_plan feature, churn at a higher rate than those without the feature. Going to visualize this relationship next.

```python
# Credit for this little function belongs to Eva Mizer :) Leveraging to make my charts
# easier as I get through my EDA process.

def mini_bar(x, y, y_title, x_title, plot_title):
    mean_df = df[[x, y]].groupby(x, as_index=False).mean()
    print(mean_df)

    #Bar plot to visualize!
    ax = sns.barplot(x=x, y=y, data=mean_df, palette=pal)
    ax.figure.set_size_inches(12,8)
    ax.set_ylabel(y_title)
    ax.set_xlabel(x_title)
    ax.set_title(plot_title)
```

```python
# Visualizing the relationship

mini_bar('international_plan', 'churn', 'International Plan', 'Churn Flag', '% of Customers that Churned by Int
```

```
   international_plan   churn
0                  no  0.114950
1                 yes  0.424149
```

% of Customers that Churned by International Plan
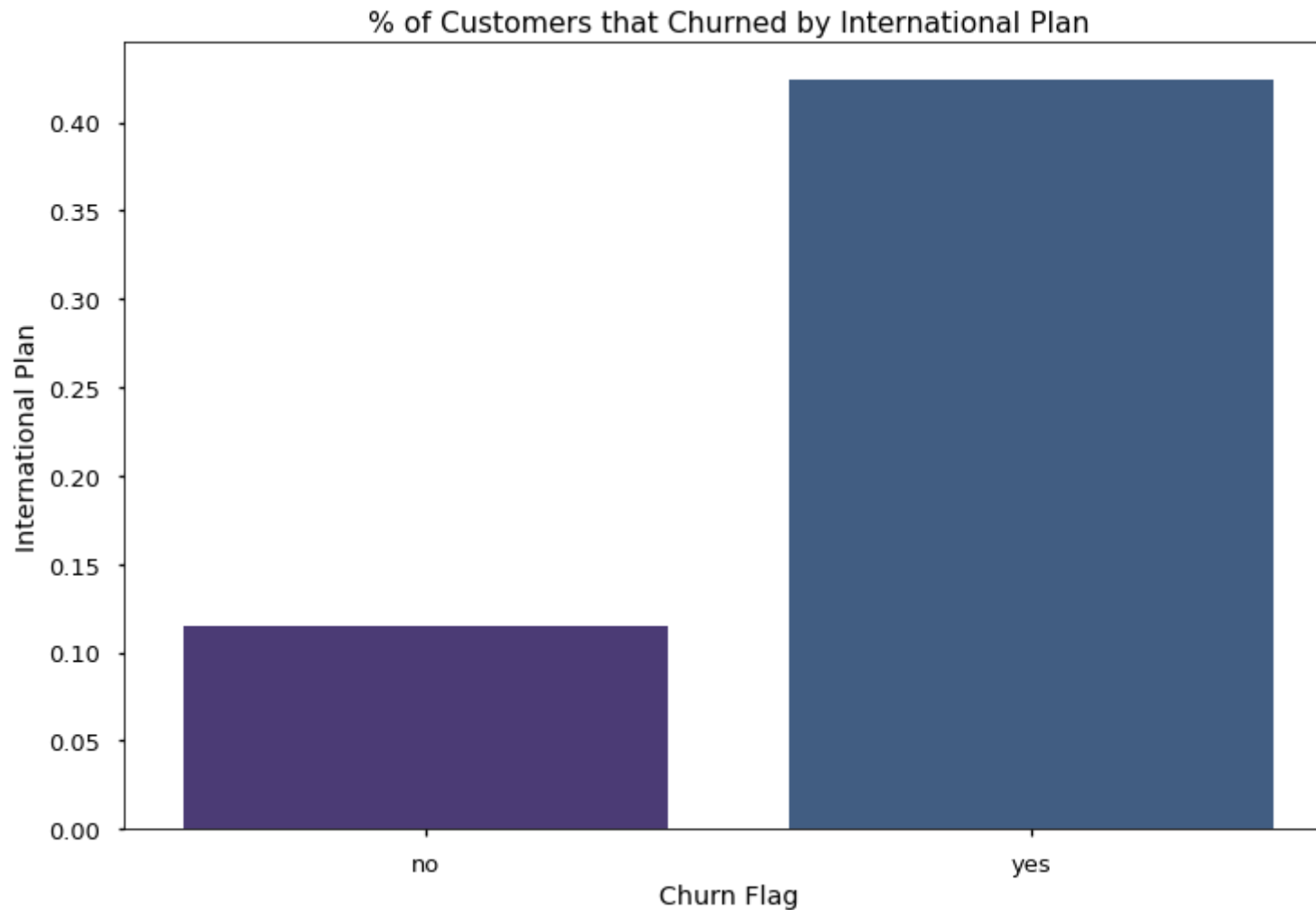
As we mentioned above, the customers with an International plan experience a higher churn percentage than those without; just over 40% of those that had the interntational plan, and about 10% for those without. About 1/3 of the total churn that was experienced were customers that had purchased the international plan. Moving on to the Voice Mail Plan feature next.

```
In [293...   # Creating pivot table for voice mail plan vs churn.

             pd.pivot_table(df,
                            values='account_length',
                            index='voice_mail_plan',
                            columns='churn',
                            aggfunc='count'
                            )
```

churn	False	True
voice_mail_plan		
no	2008	403
yes	842	80

```python
# Visualizing relationship

mini_bar('voice_mail_plan', 'churn', 'Voice Mail Plan', 'Churn Flag', '% of Customers that Churned by Voice Mai
```

```
   voice_mail_plan    churn
0               no  0.167151
1              yes  0.086768
```

% of Customers that Churned by Voice Mail Plan
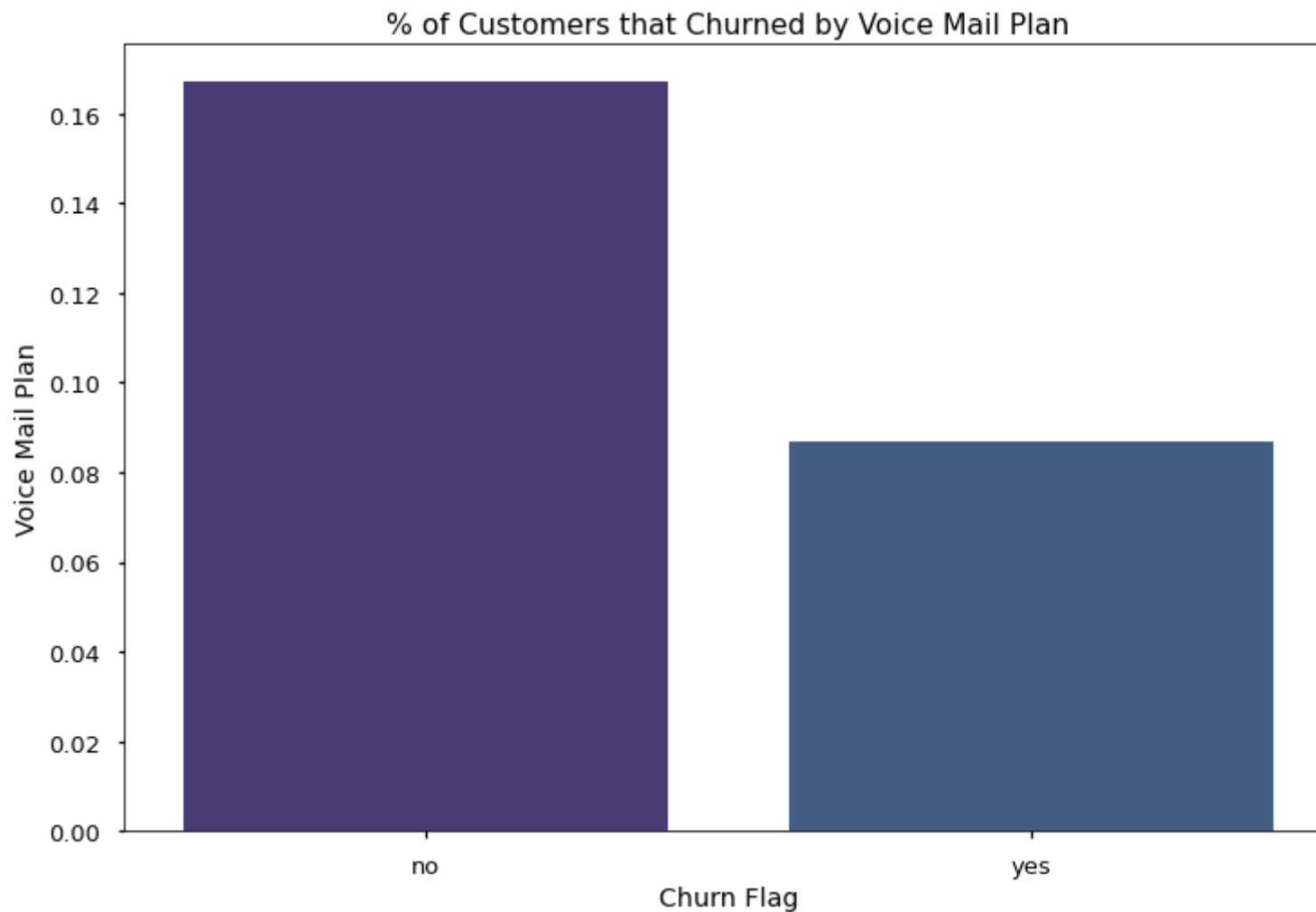
Looks like those plans with a voice mail plan have a lower churn percentage than those without. 9% of those with the voice mail plan vs 17% of those without a voice mail plan churn. We will keep this in mind as we complete our analyses. So as of right now, it would suffice us to say that customers with the voicemail plan were more satisfied than those customers without that feature.

```python
# Now looking at the relationship of churn between the 2 product features we listed
# above.  Looking for any uniqueness within the feature combinations.

pd.pivot_table(df,
            values='account_length',
            index=['voice_mail_plan', 'international_plan'],
            columns='churn',
```

```
                    aggfunc='count'
```

voice_mail_plan	international_plan	churn False	True
no	no	1878	302
	yes	130	101
yes	no	786	44
	yes	56	36

So between our two features above, it looks like:

- No voice mail plan & no interntational plan Churn – **14%**
- No voice mail plan & interntational plan Churn – **44%**
- Voice mail plan & no international plan Churn – **5%**
- Voice mail plan & international plan Churn – **39%**

So customers that had the voice mail plan & no international plan, or no features at all performed be best (5%, and 14% respectively). Again, this could mean that the service of the international plan does meet the expectations of the customers. With that being said, the voice mail feature/ functionality does generally pretty well, and customers see the value in that feature. The same cannot be said for the international feature. This is typically bringing down the experience overall.

```python
# Now that we know customers generally are not satisfied with the international plan
# we are going to dig into whether they pay more per minute to have those features
# enabled.

pd.pivot_table(df,
               values='price_per_minute',
               index=['voice_mail_plan', 'international_plan'],
               columns='churn',
               aggfunc=np.mean
               )
```

voice_mail_plan	international_plan	churn False	True
no	no	0.050518	0.055644
	yes	0.053261	0.053121

churn		False	True
voice_mail_plan	international_plan		
yes	no	0.051141	0.049108
	yes	0.051807	0.053927

Now taking a look at the average price paid per minute between the same feature diagram we looked at above, it looks like those plans with the international plan pay more on average than those without.

Here is the comparison of price between the churn, and non-churn groups that we looked at above:

- `No voice mail plan & no interntational plan Churn` – **10% higher in Churn price per minute vs Non Churn price per minute**
- `No voice mail plan & interntational plan Churn` – **0% difference in Churn price per minute vs Non Churn price per minute**
- `Voice mail plan & no international plan Churn` – **-4% difference in Churn price per minute vs Non Churn price per minute**
- `Voice mail plan & international plan Churn` – **4% higher cost in Churn price per minute vs Non Churn price per minute**

This would indicate that there really isn't much price difference when looking at the per minute costs for these additional features. Or at least the customers that churned weren't paying that much more than the customers that were retained, providing support for the fact that those features value were really based on if they met the expectations of functionality or not.

Demographics/ Customer Categorical Features EDA

```
In [297…
fix, ax = plt.subplots(figsize=(20,12))

df['state'].value_counts().plot(kind='bar', color=pal[0])

ax.set_title('# of Customers by State')
ax.set_xlabel('State', labelpad=15, fontsize=16)
ax.set_ylabel('# of Customers', labelpad= 15, fontsize=16)

plt.show()
```

# of Customers by State
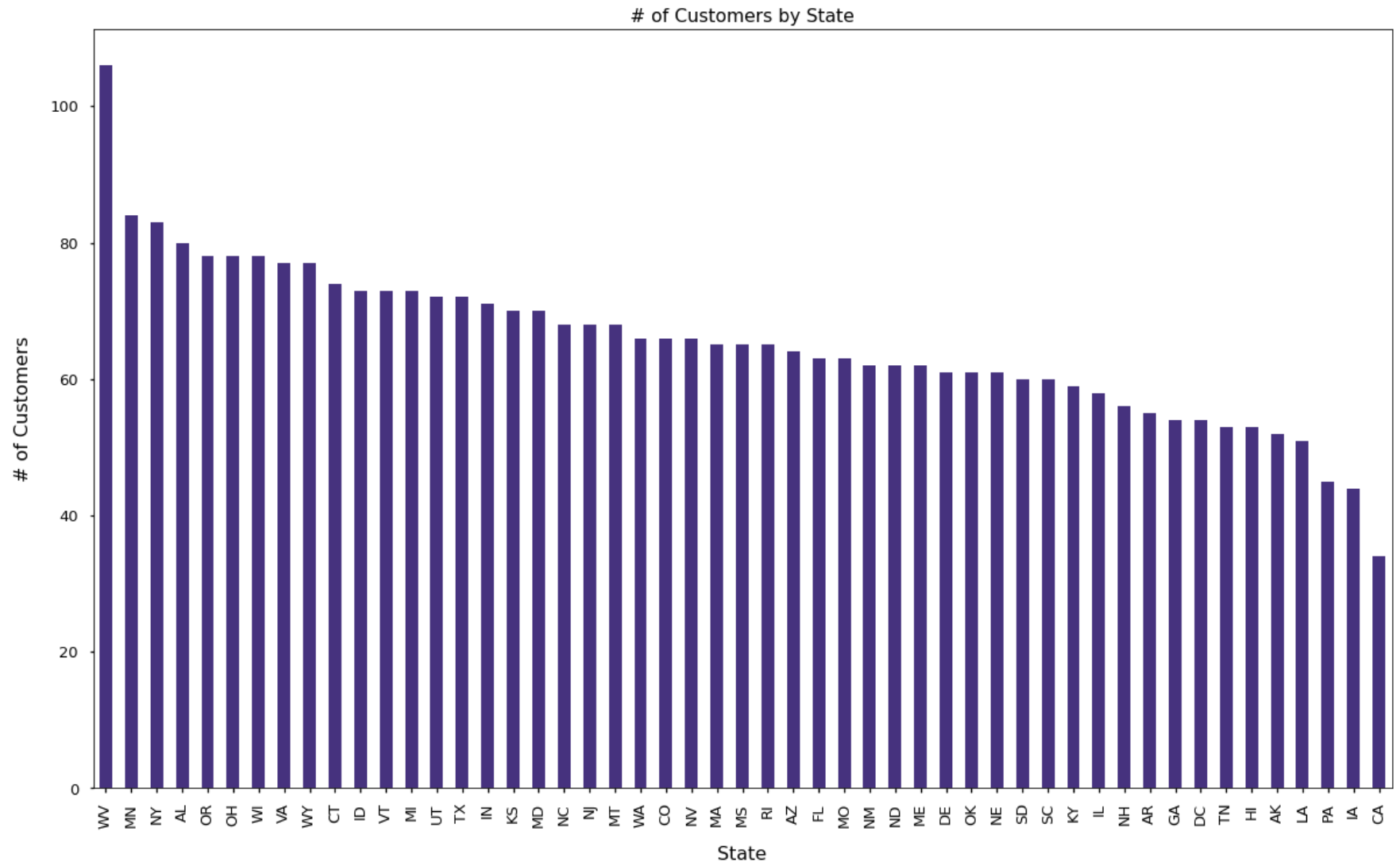
Overall the distribution of states is pretty good! No concerns here, however we will need to encode this categorical column if we want to retain this information. I am not sure if this will help us or if it will cause us to over fit on our training model. I say that because of the account specificity that we will introduce by maintaining this feature.

Since we are thinking about dropping our state column, we still have area_code, and our phone_number_prefix, to give us an idea of where the users are located. Going to split those columns out from the line_number, which won't really help us. We may need to drop the prefix and the area code for the same reasons I mentioned for state above. For now, we will prep the data in case we do want to keep it in.

```
In [298…    # Adjusting out our phone number prefix and line number information to make it easier to work with.

            df[['phone_number_prefix','line_number']] = df.phone_number.str.split("-",expand=True,).astype(int)
            df
```

Out[298…

	state	account_length	area_code	phone_number	international_plan	voice_mail_plan	number_vmail_messages	total_day_minut
0	KS	128	415	382-4657	no	yes	25	265
1	OH	107	415	371-7191	no	yes	26	161
2	NJ	137	415	358-1921	no	no	0	243
3	OH	84	408	375-9999	yes	no	0	299
4	OK	75	415	330-6626	yes	no	0	166
...	...	...	...	...	...	...	...	
3328	AZ	192	415	414-4276	no	yes	36	156
3329	WV	68	415	370-3271	no	no	0	23
3330	RI	28	510	328-8230	no	no	0	180
3331	CT	184	510	364-6381	yes	no	0	213
3332	TN	74	415	400-4344	no	yes	25	234

3333 rows × 27 columns

```
In [299…    # dropping state, and phone number

            df = df.drop(['phone_number',
                          'line_number',
                          'state'
                         ],
                        axis=1
                        )
```

```
In [300…    # Going back to look at our columns that need transformed from string to integer // international plan

            df = df.replace({'international_plan':
                            {'yes': 1,
                             'no': 0
                            }
```

```python
        }
    `
```

In [301...
```python
# Going back to look at our columns that need transformed from string to integer // voice mail plan

df = df.replace({'voice_mail_plan':
                    {'yes': 1,
                     'no': 0
                    }
                }
            )
```

In [302...
```python
# Going back to look at our columns that need transformed from string to integer // voice mail plan

df['churn'] = df['churn'].astype(int)
```

In [303...
```python
# Quick look at our data set to see how we are evolving.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 24 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   account_length         3333 non-null   int64
 1   area_code              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
 19  total_calls            3333 non-null   int64
 20  total_charges          3333 non-null   float64
 21  total_minutes          3333 non-null   float64
```

```
 22   price_per_minute          3333 non-null    float64
 23   phone_number_prefix       3333 non-null    int64
dtypes: float64(11), int64(13)
memory usage: 625.1 KB
```

Usage EDA

Now that we have looked at some of our categorical/ demogrphic based features, next we are going to dig into some of our usage based features to note any outliers, and any patterns between our target, churn vs non-churn customer segments.

In [304...
```python
# Creating a few filtered data sets to make some visuals/ grouping easier if needed.

df_churn = df[df['churn'] == True]
df_active = df[df['churn'] == False]
```

In [305...
```python
# Quick look at distribution across our numerical values.  Looking for outliers, even though
# our data seems to be pretty evenly distributed across our feature sets.

plt.rcParams['figure.max_open_warning'] = 0;

for column in df:
        plt.figure(figsize=(14,1))
        sns.boxplot(data=df_churn, x=column, color=pal[0])
```
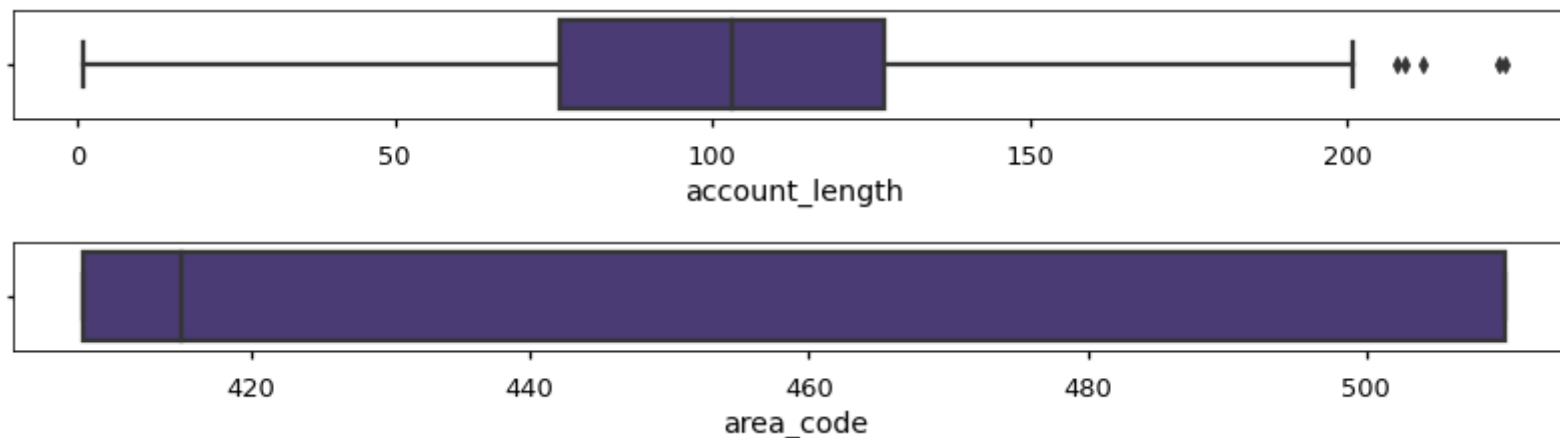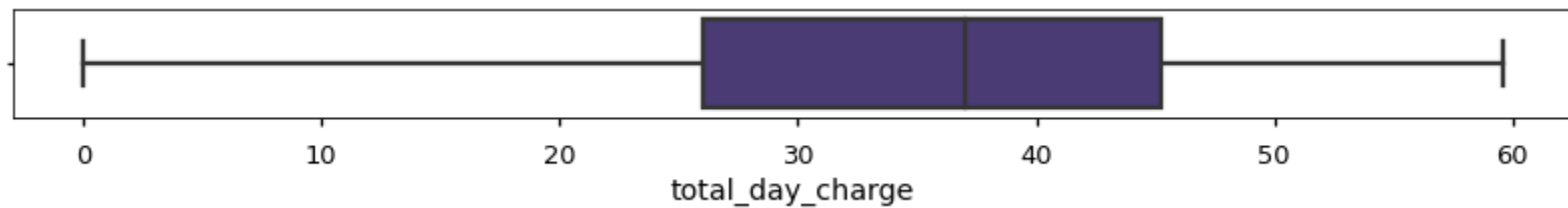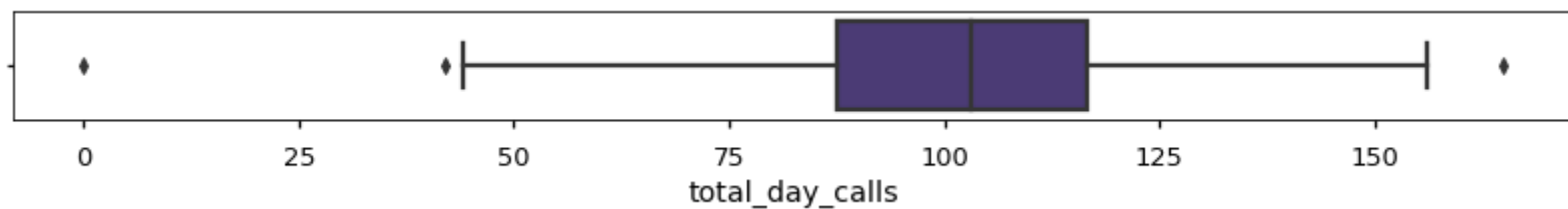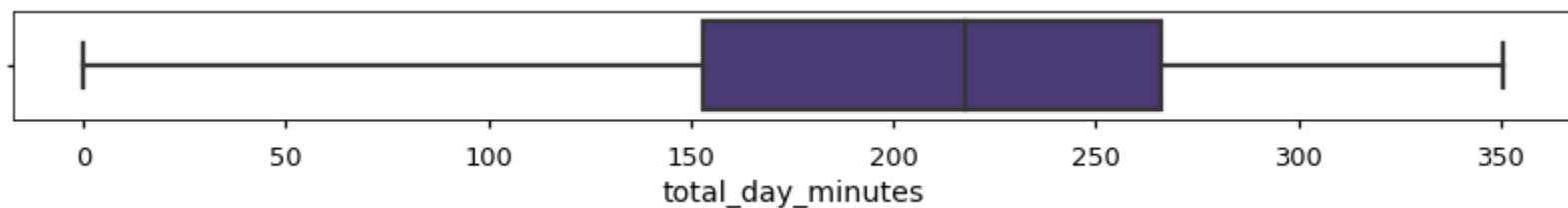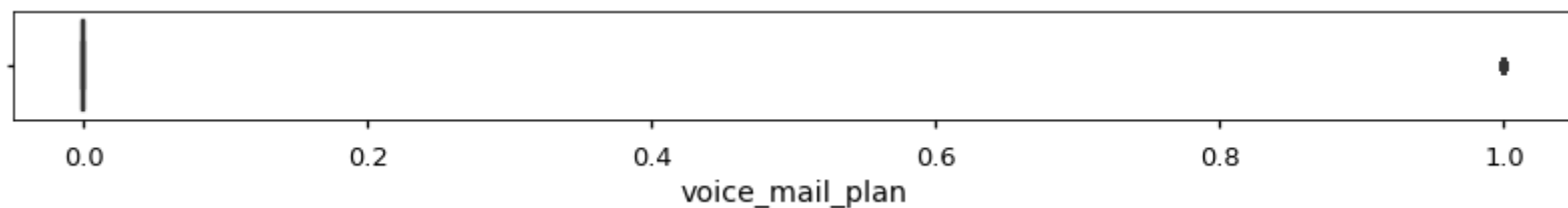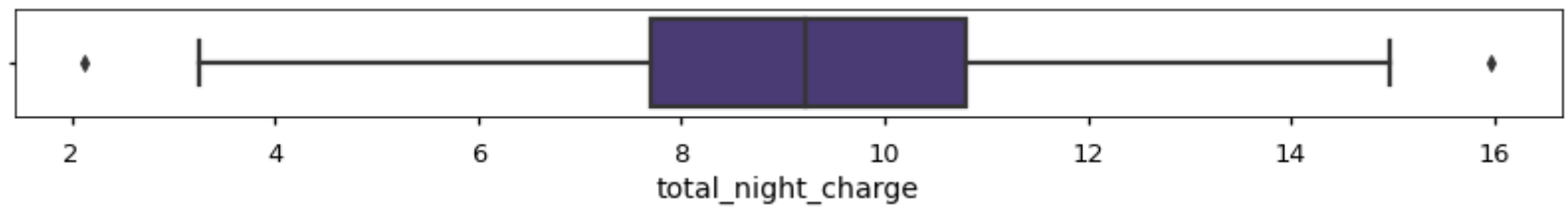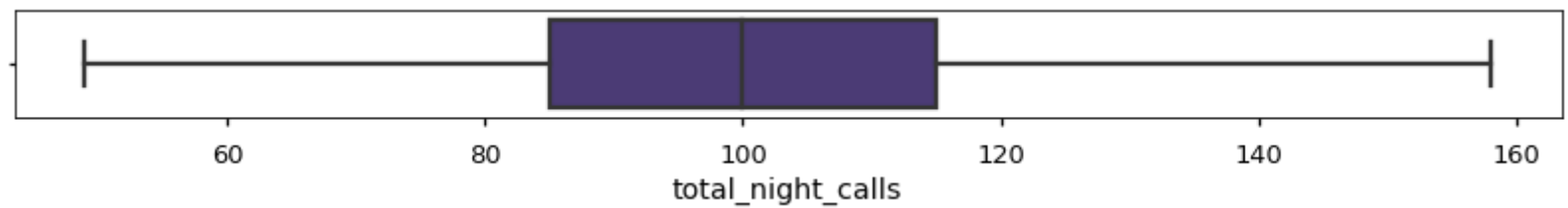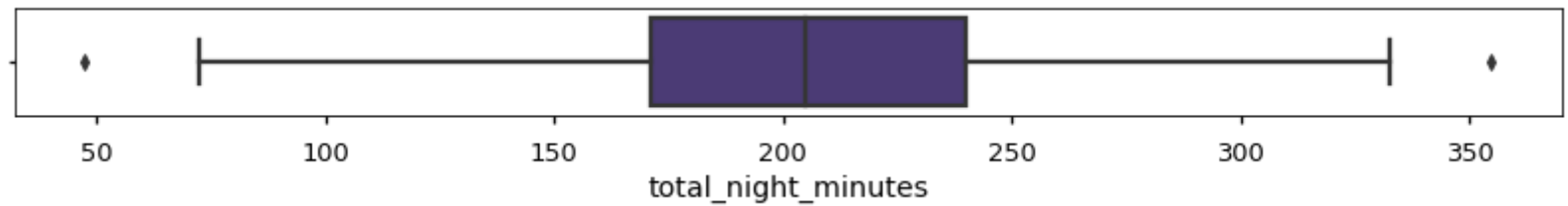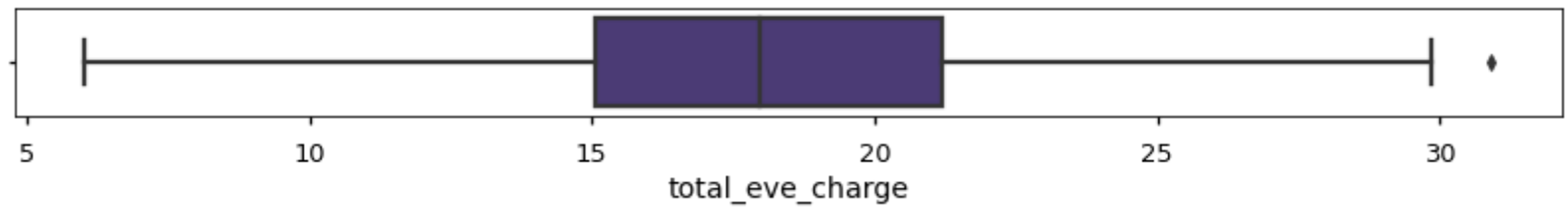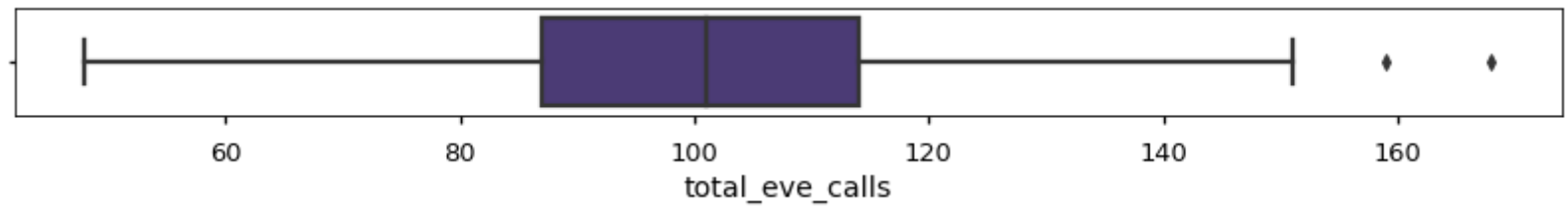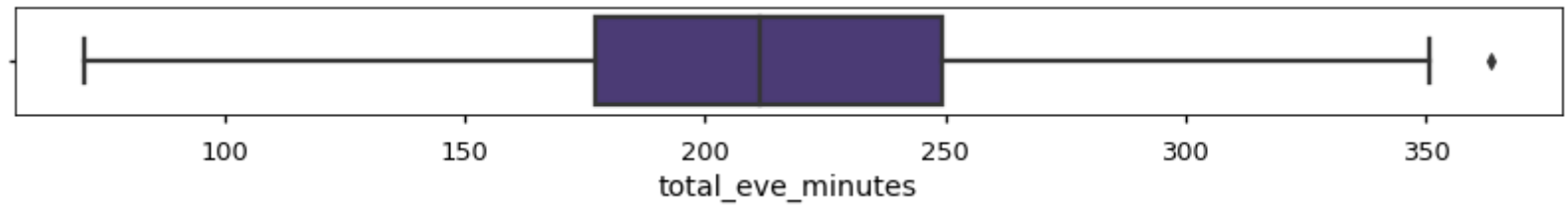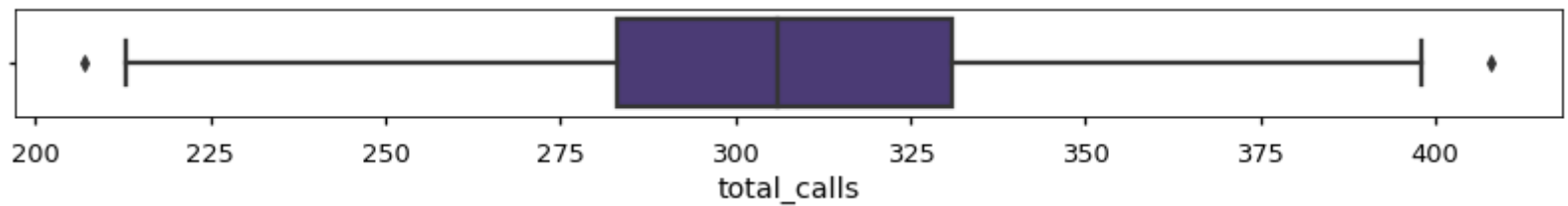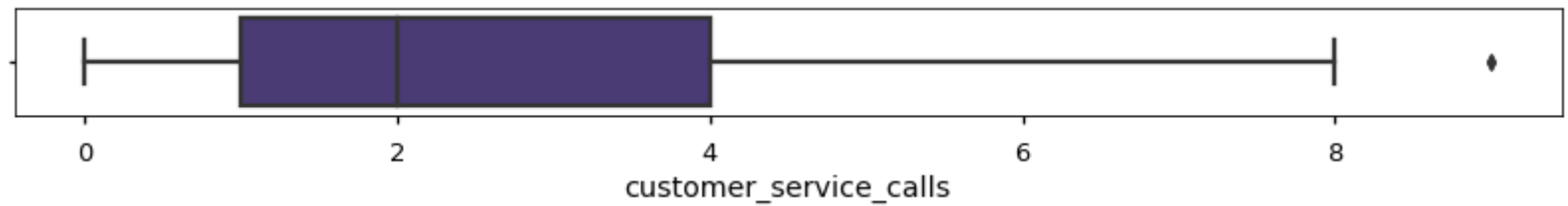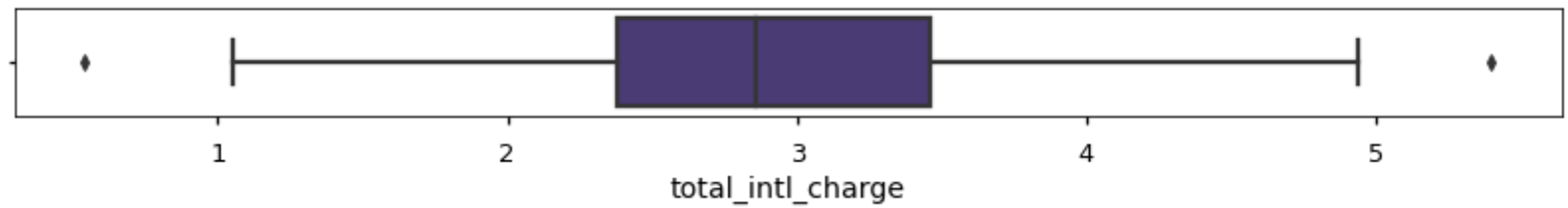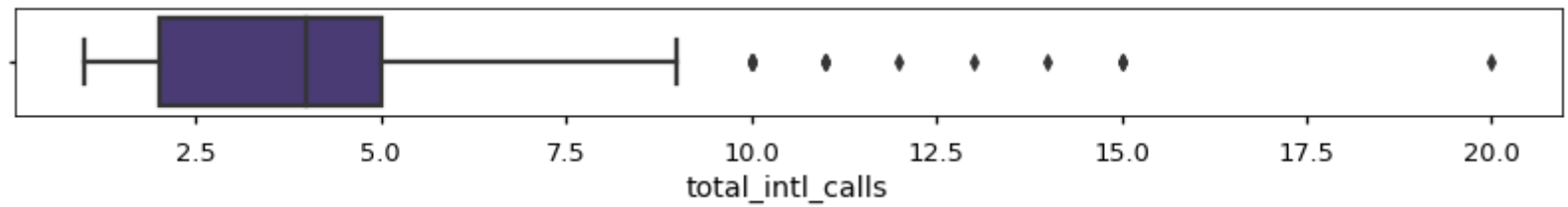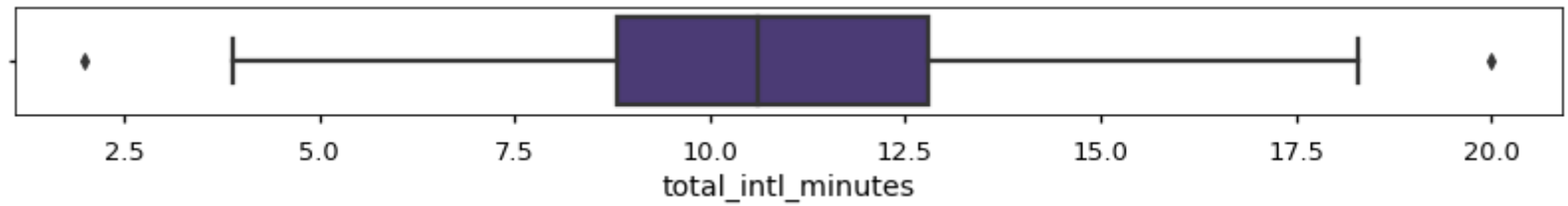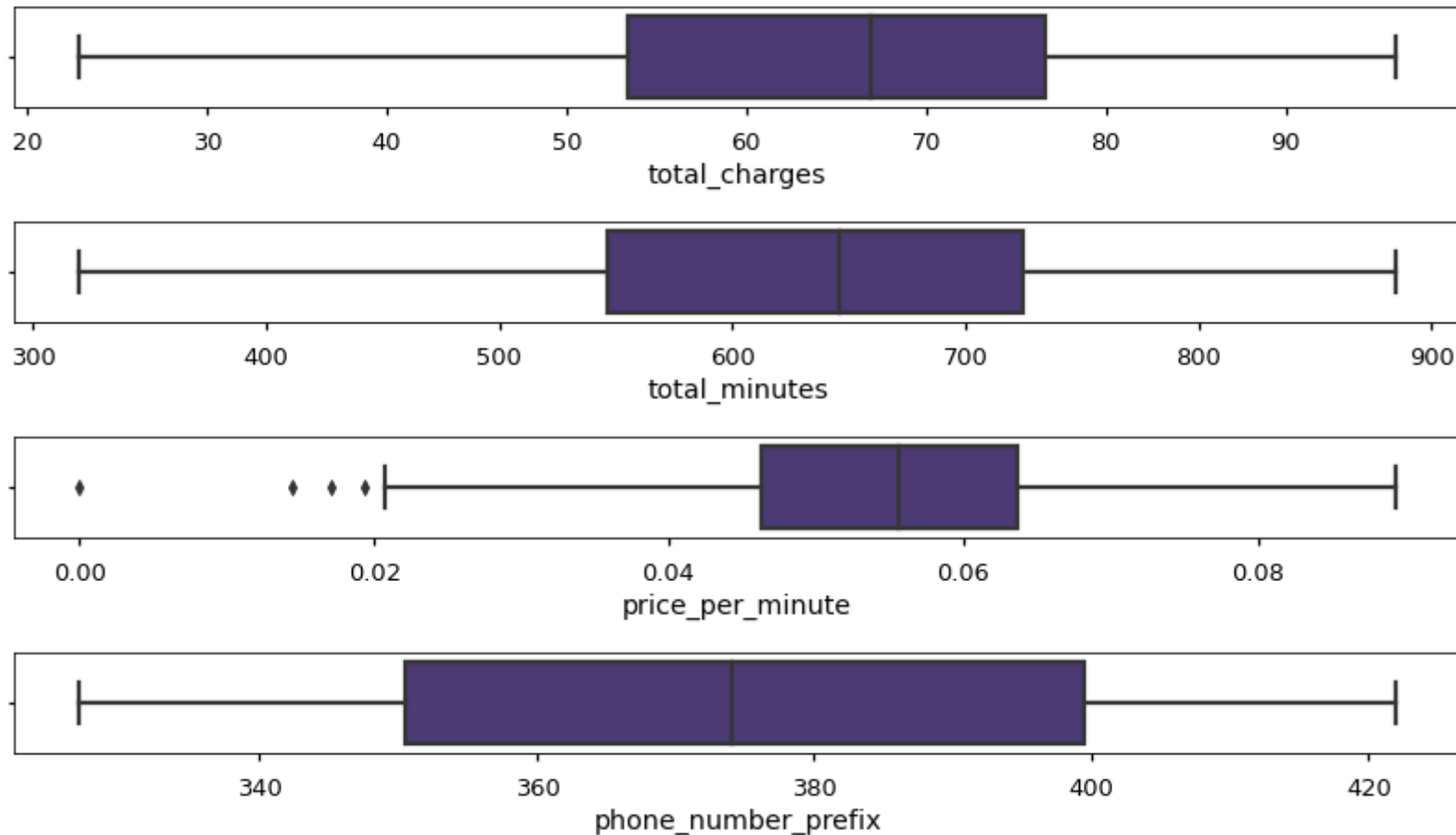
Not a ton of outliers within our numerical features, at least in aggregate. The voice mail messages seems to have the hgiher number of outliers, but that could be for a number of reasons. The sample is smaller, and we will have the impact between the groups that actually have the feature, vs those that don't have the feature impacting the distribution (since our charts are naive).

```
# Going to visualize our price per minute feature between churn and non- churn accounts.

mini_bar('churn', 'price_per_minute', 'Price per Minute', 'Churn Flag', 'Price per Minute by Customer Status');
```

```
   churn  price_per_minute
0      0          0.050840
1      1          0.054393
```
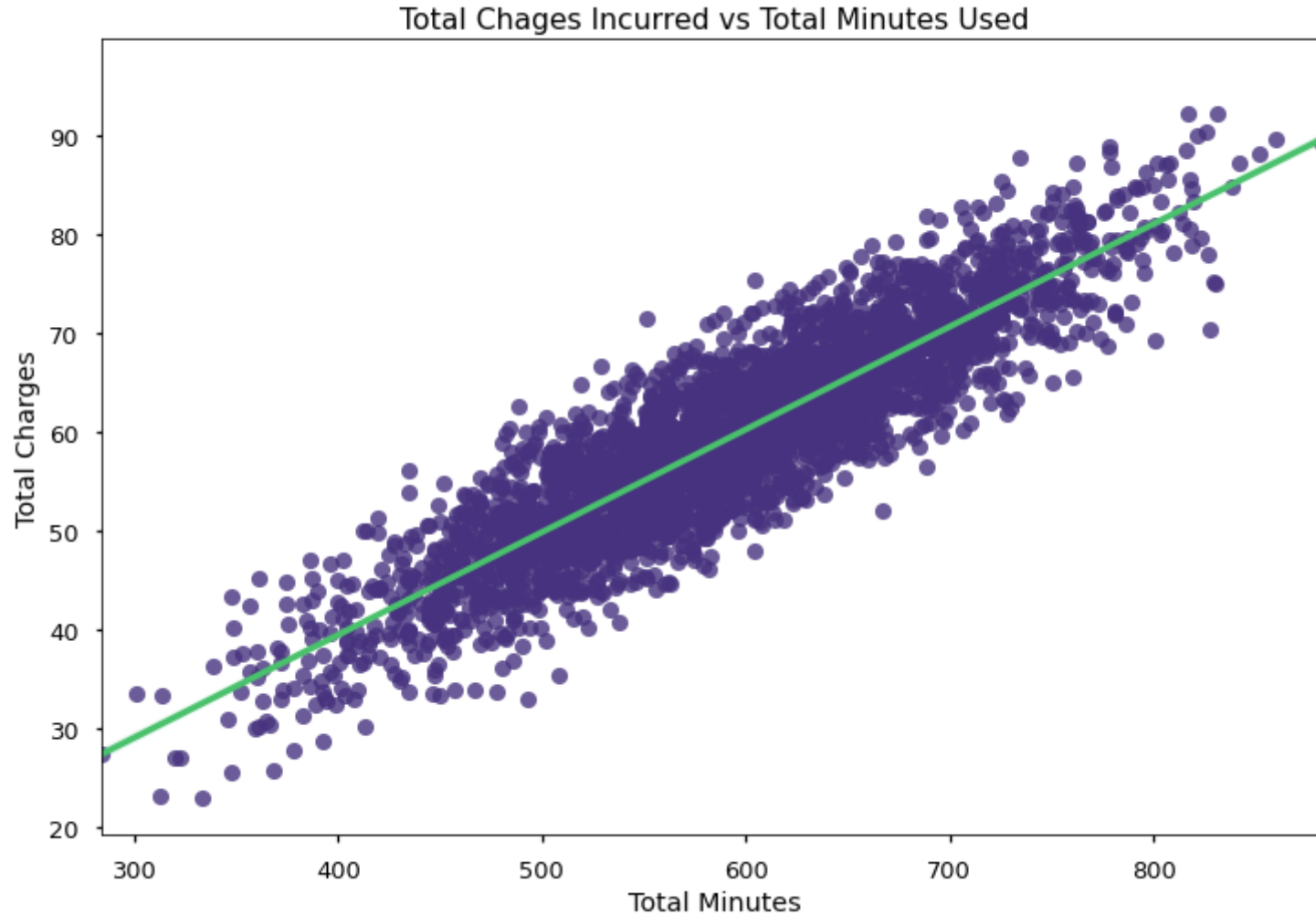
Price per Minute by Customer Status
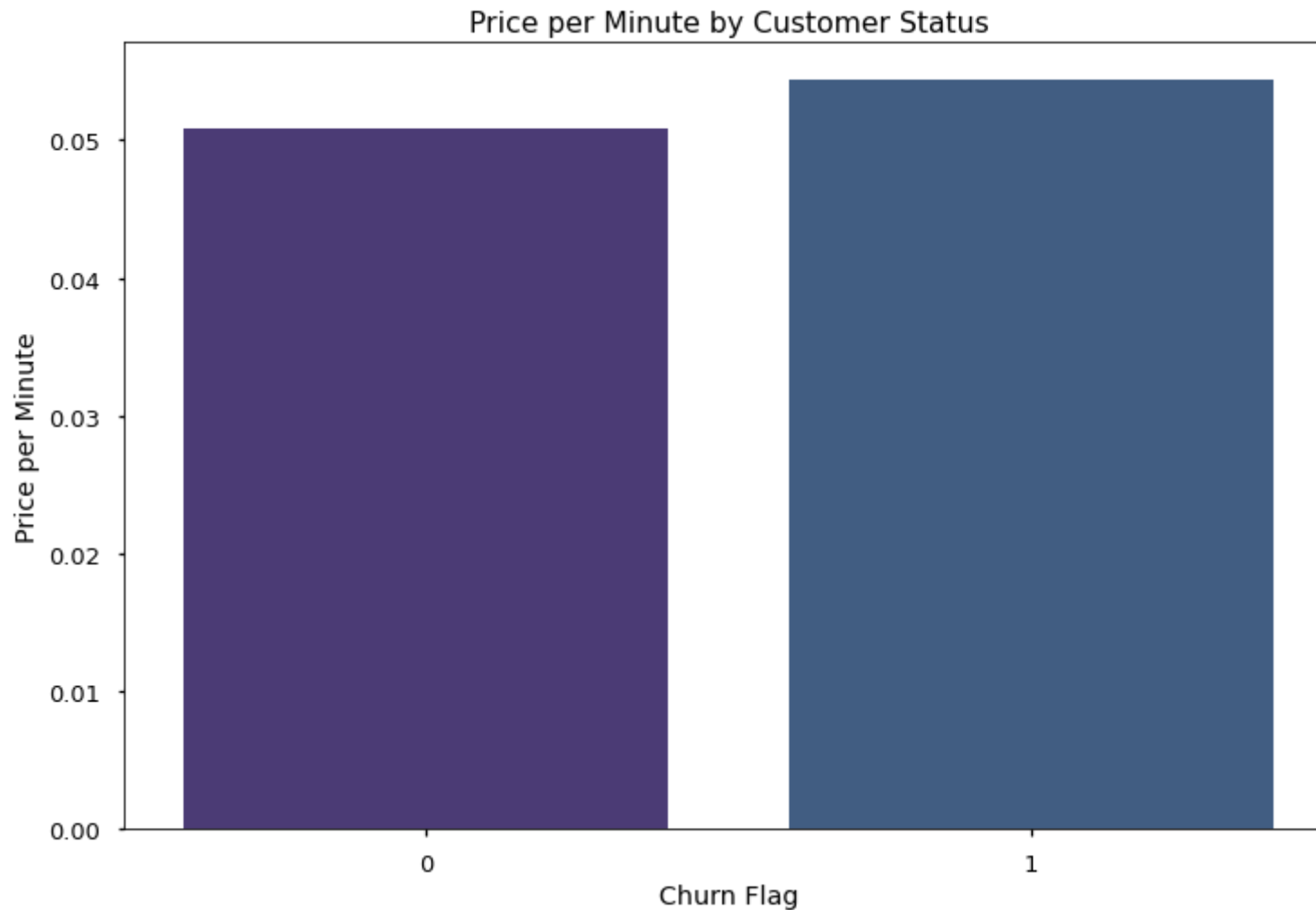
Churned customers paid **7% more** per minute than customers that were retained. Not sure why that might be the case, but could be a result of customers with the international plan included.

```
In [307...
# Creating a scatter plot to look at the relationship between price, and usage.

plt.figure(figsize = (12,8))

rp = sns.regplot(x='total_minutes',
            y='total_charges',
            data=df,
            scatter_kws={"color": pal[0]},
            line_kws={"color": pal[4]})
```

```
rp.set_ylabel('Total Charges')
rp.set_xlabel('Total Minutes')
```

Total Chages Incurred vs Total Minutes Used

This makes sense to me. As the product is used, charges are increasing. What we really want to investigate though is if the price per minute is going down, as the the total minutes go up. If there was a strategic pricing, I think we would want to see the price per minute go down, but the charges stay flat because of the increase in minutes used.

In [308...

```
# Creating a scatter plot to look at the relationship between price, and usage.

plt.figure(figsize = (12,8))
```

```
rp = sns.regplot(x='total_minutes',
                 y='price_per_minute',
                 data=df,
                 scatter_kws={"color": pal[0]},
                 line_kws={"color": pal[4]})

rp.set_ylabel('Total Charges')
rp.set_xlabel('Total Minutes')
rp.set_title('Total Chages Incurred vs Total Minutes Used');
```
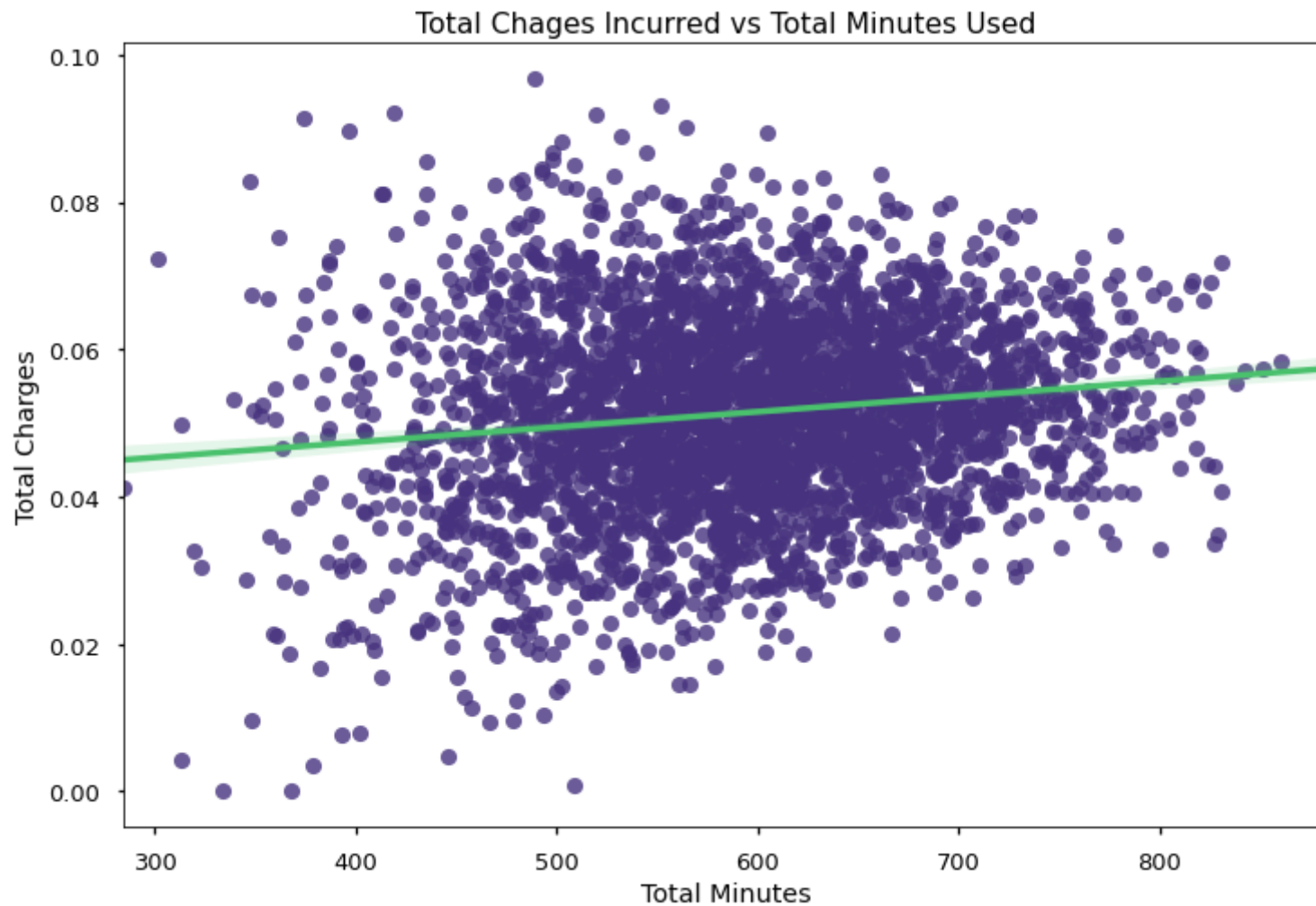

Total Chages Incurred vs Total Minutes Used

This is still showing a positive correlation between our two variables. I think we would actually want to see a negative slope here indicating that the customers that use the product the most, would be getting a slight discount on pricing as usage increases.
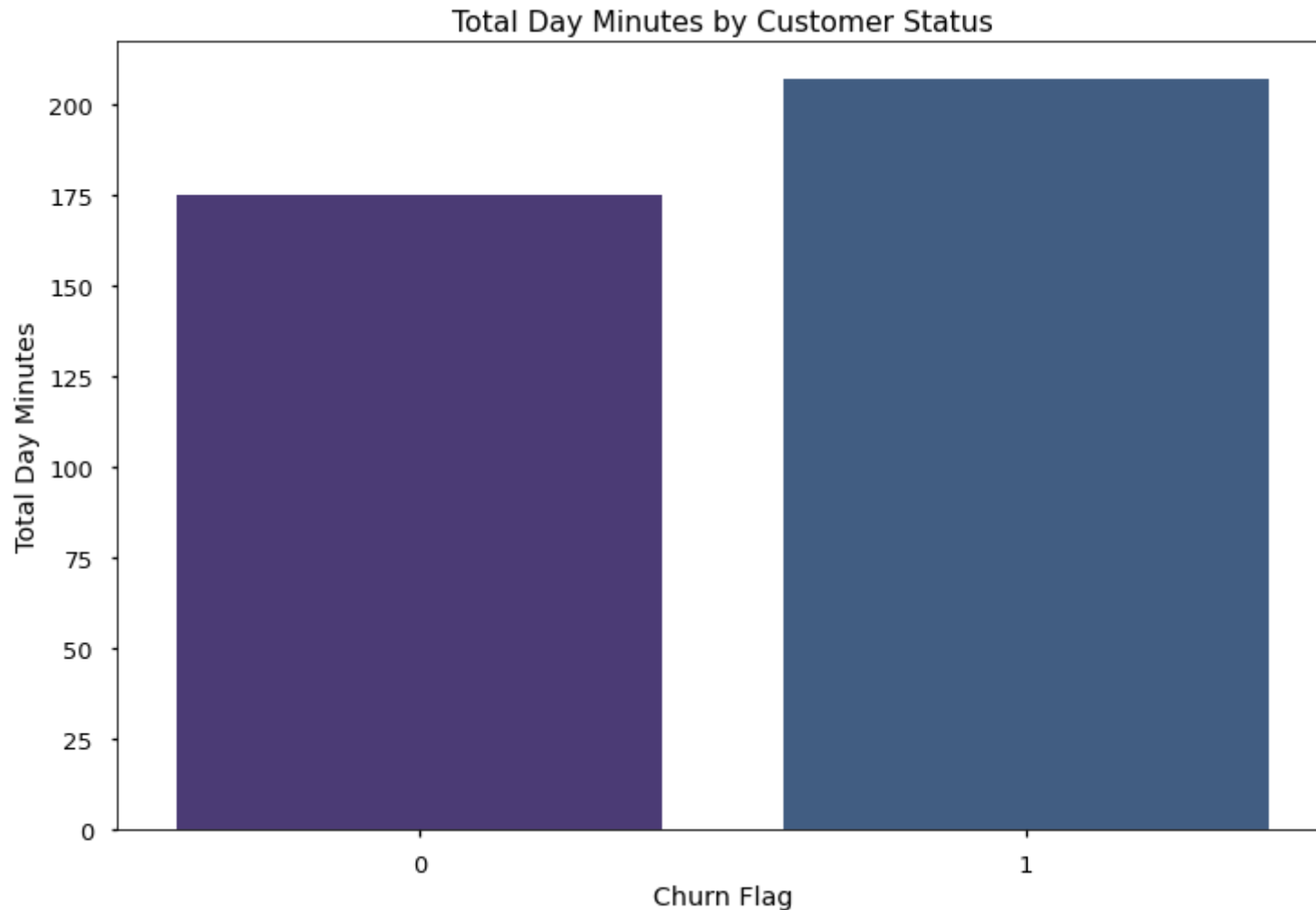
```
# Looking at usage between total day minutes within our target customer segments

mini_bar('churn', 'total_day_minutes', 'Total Day Minutes', 'Churn Flag', 'Total Day Minutes by Customer Status
```

```
   churn  total_day_minutes
0      0         175.175754
1      1         206.914079
```


Customers that churn use the plan during the day **18% more** than the customers that were retained.

```
# Looking at averag customer service calls between our target customer segments.

mini_bar('churn', 'customer_service_calls', 'Customer Service Calls', 'Churn Flag', 'Customer Service Calls by
```

```
     churn  customer_service_calls
0      0                 1.449825
1      1                 2.229814
```

Customer Service Calls by Customer Status
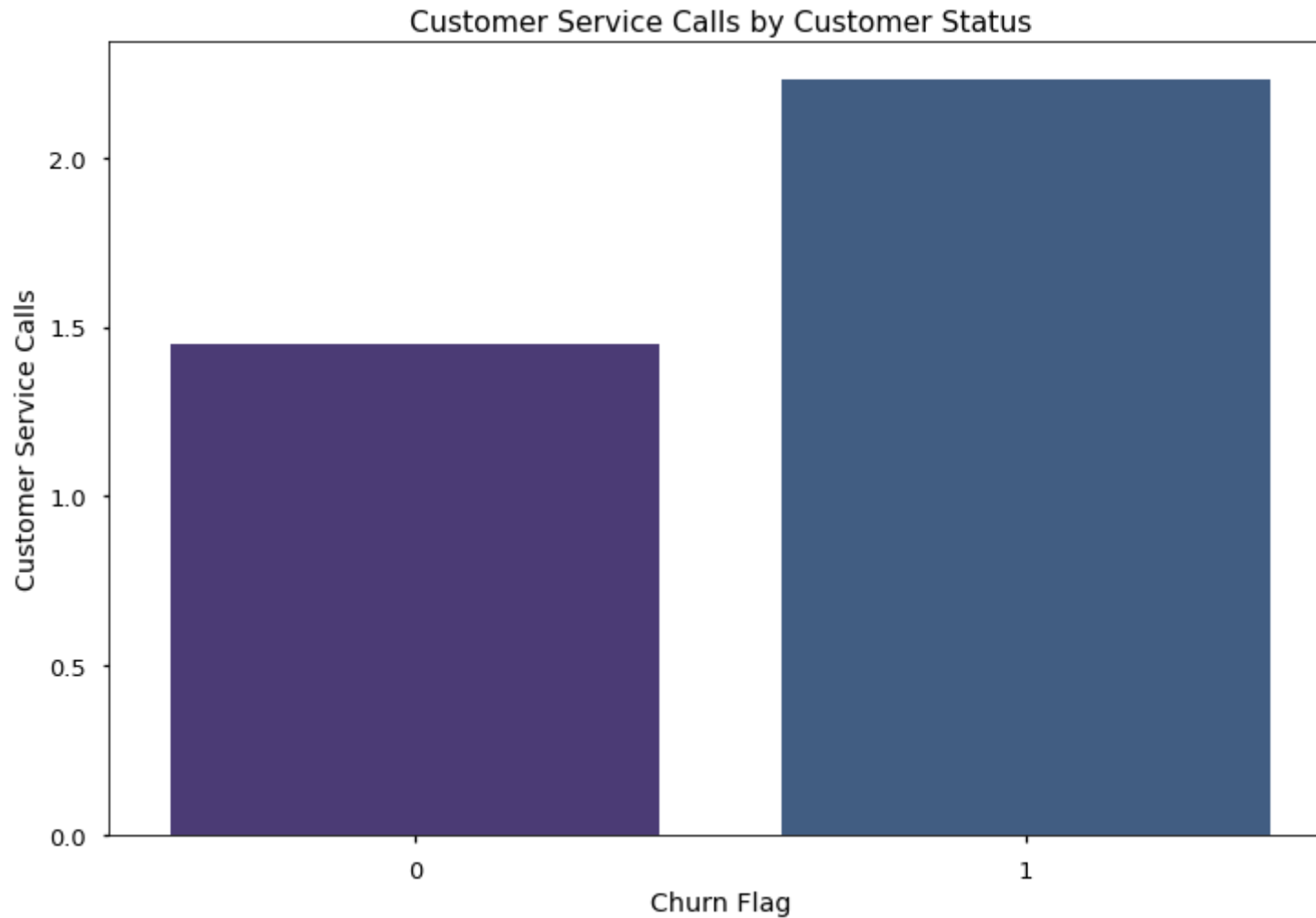
Customers that churned contacted customer service **54% more** than customers that were retained. Going to take a quick look at the % of churn as customer service calls increase.

```python
# Create an object to chart

cs_churn_percent = df['churn'].groupby(df['customer_service_calls']).mean()*100
```

```python
# Create a charge looking at the customer service calls and churn percentage within each.
```

```python
fix, ax = plt.subplots(figsize=(20,12))

cs_churn_percent.plot(kind='bar', color=pal[0])


ax.set_title('Churn % by Number of Customer Service Calls')
ax.set_xlabel('Customer Service Calls', labelpad=15, fontsize=16)
ax.set_ylabel('Churn %', labelpad= 15, fontsize=16)

fmt = '%.0f%%' # Format you want the ticks, e.g. '40%'
yticks = mtick.FormatStrFormatter(fmt)
ax.yaxis.set_major_formatter(yticks)

plt.show()
```

Churn % by Number of Customer Service Calls
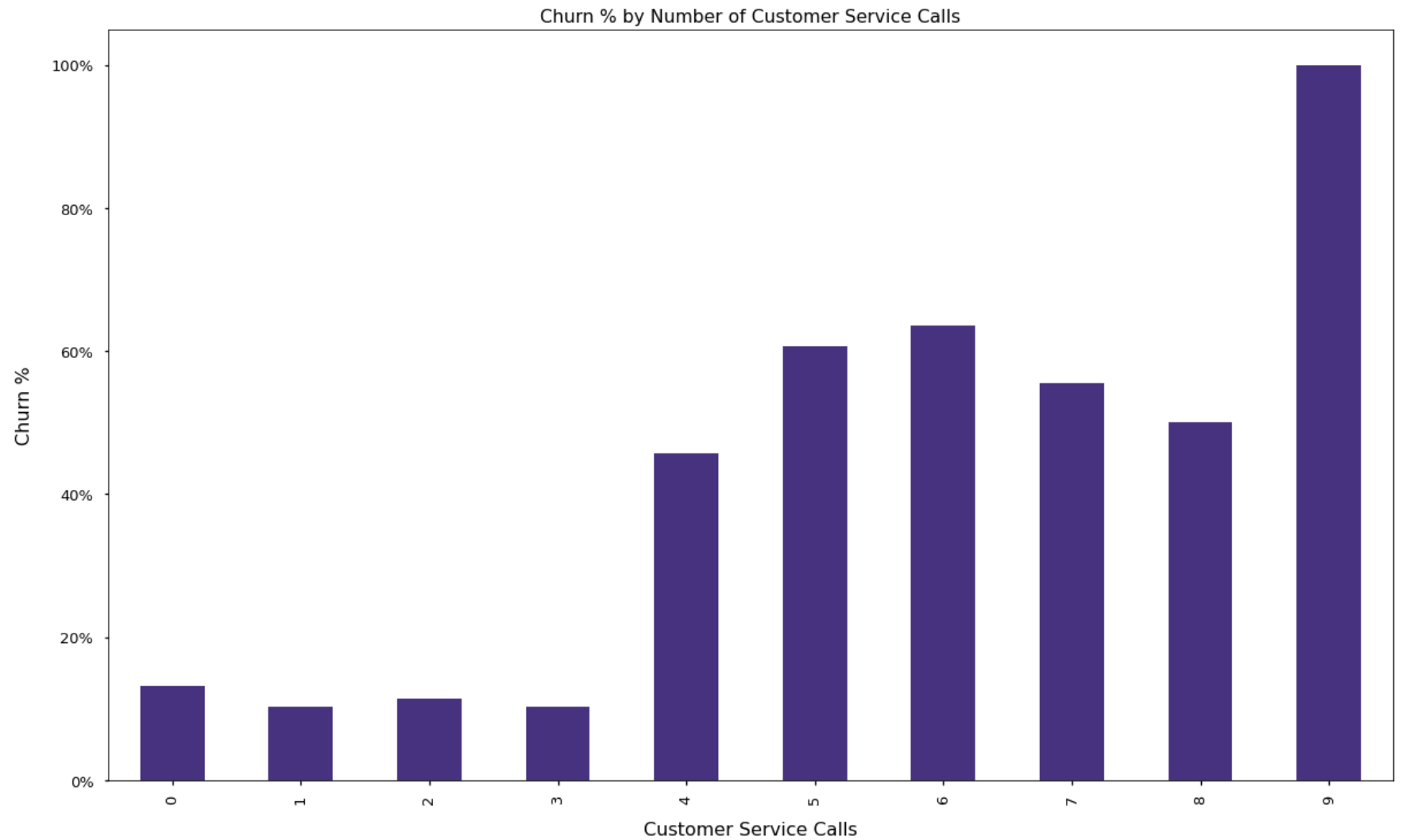
```
# Total international calls by target customer segment

mini_bar('churn', 'total_intl_calls', 'Total International Calls', 'Churn Flag', 'Total International Calls by
```

```
   churn  total_intl_calls
0      0          4.532982
1      1          4.163561
```

Total International Calls by Customer Status
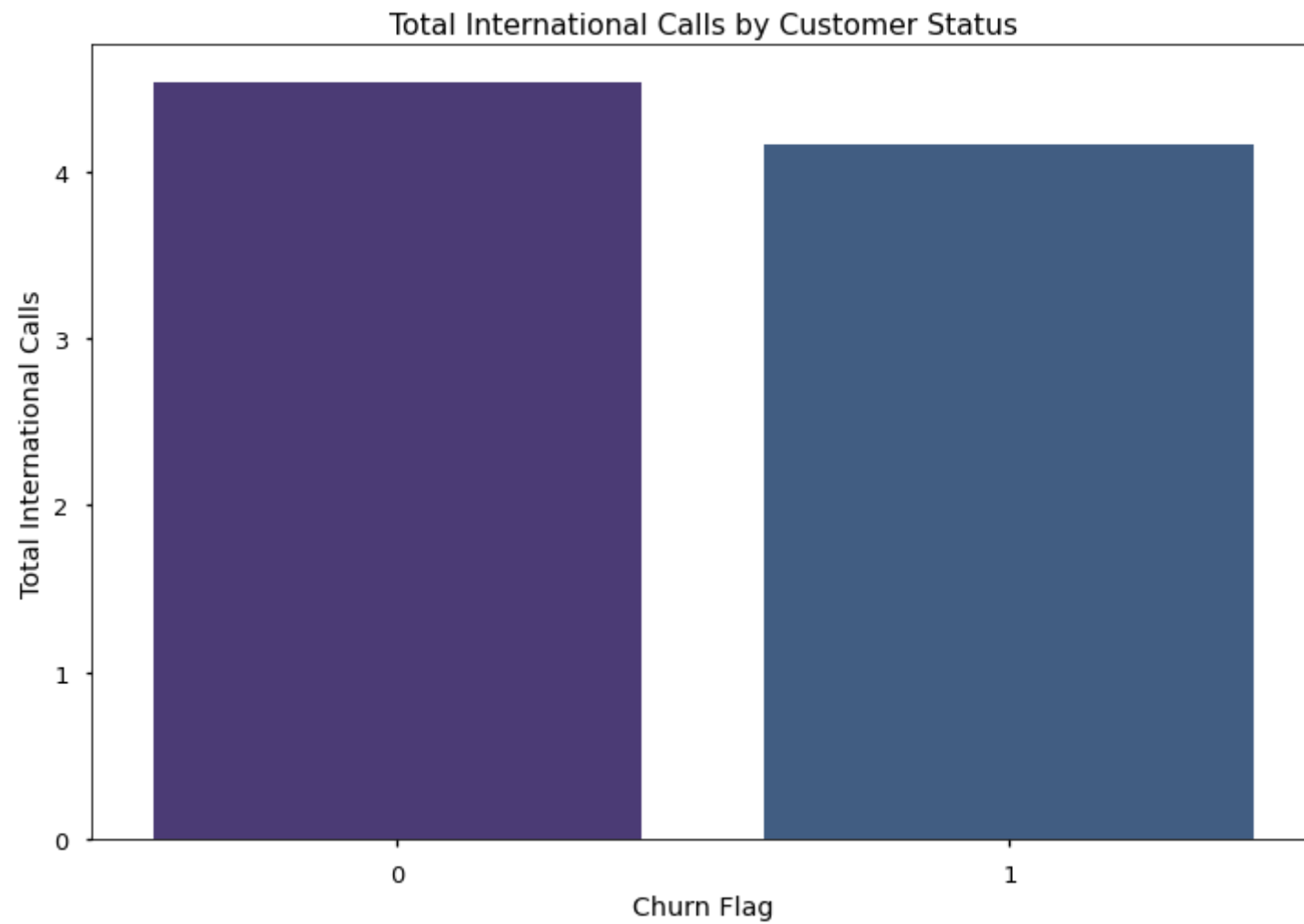
```
# Total night minutes between target customer segments

mini_bar('churn', 'total_night_minutes', 'Total Day Calls', 'Churn Flag', 'Total Day Calls by Customer Status')
```

```
   churn  total_night_minutes
0      0            200.133193
1      1            205.231677
```

Total Day Calls by Customer Status
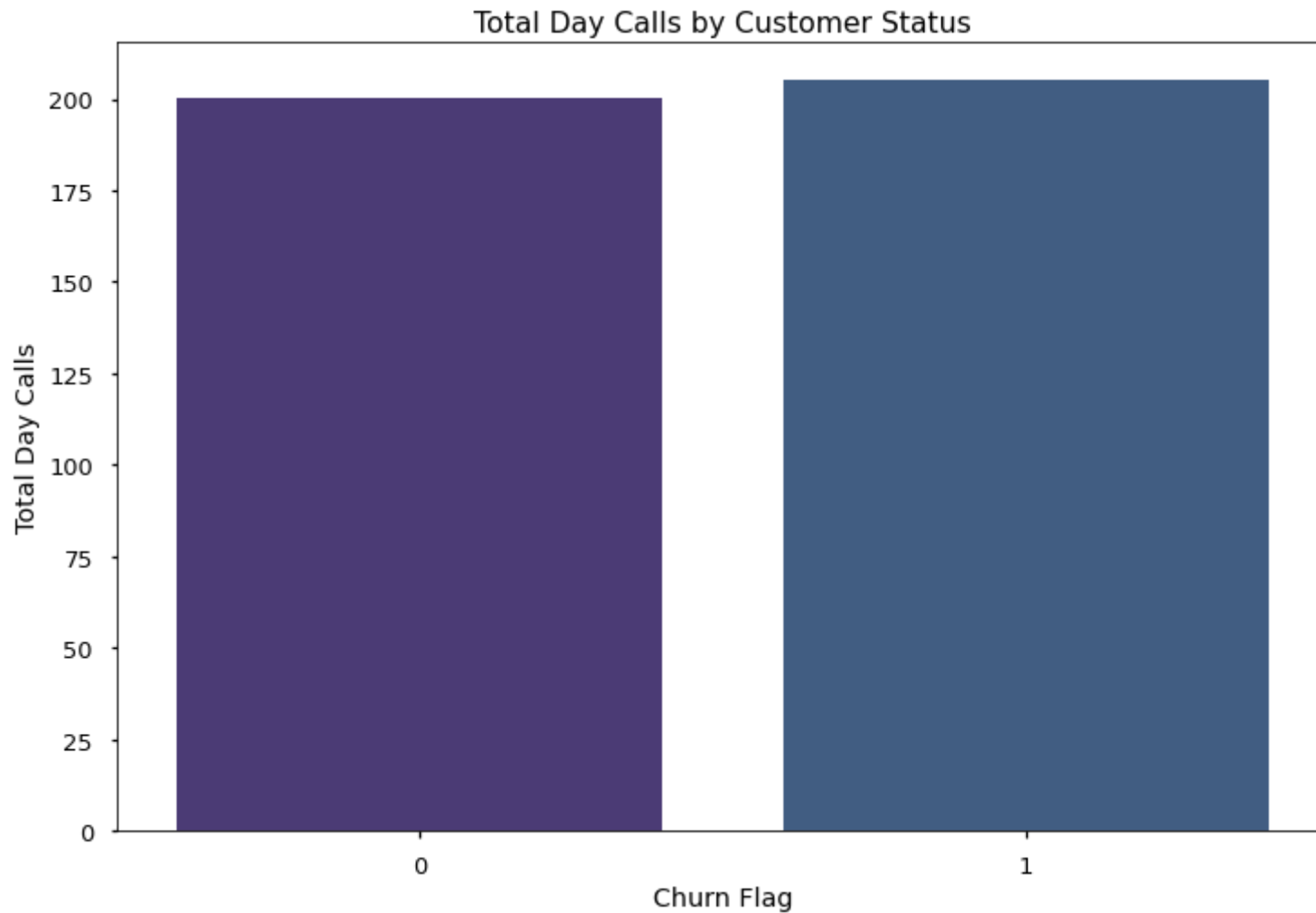
Nothing to really write home about with the last two charts. The difference in usage between our target segments is negligible. Moving on to collinearity before we get into our classification modeling.

Distribution, Linearity, Correlation EDA

In [315…
```python
# Going to look at overall distribution and histograms to visually inspect our data

df.hist(bins='auto',
        figsize=(40,40),
        color=pal[0]
        );
```
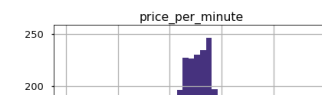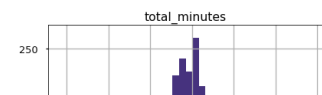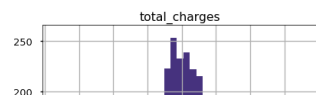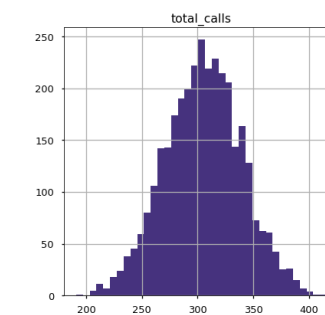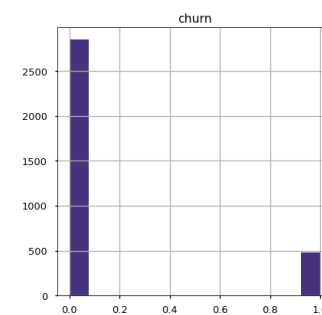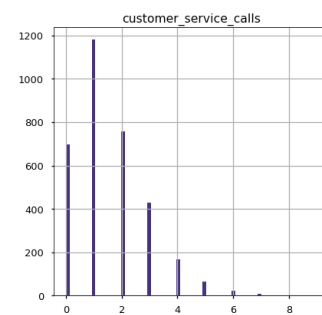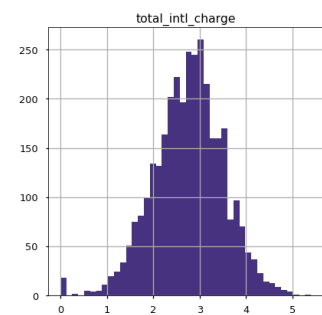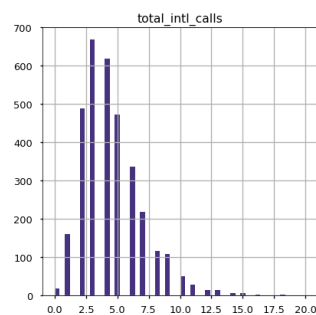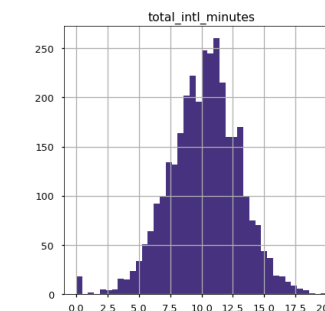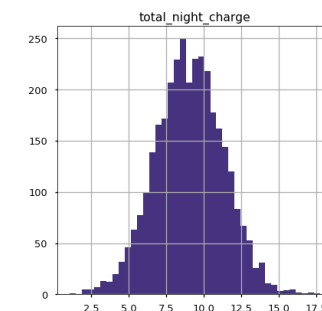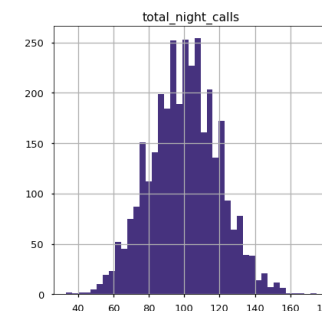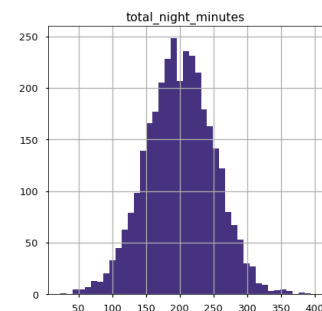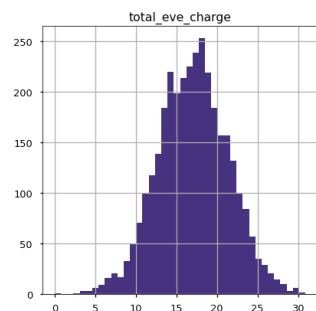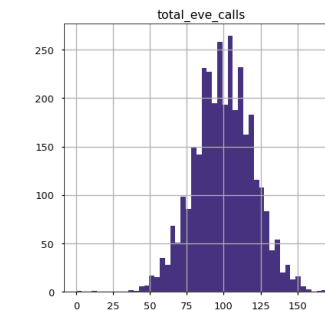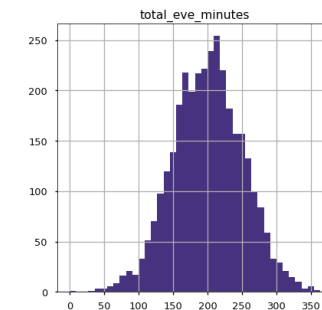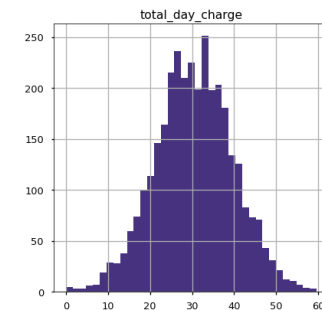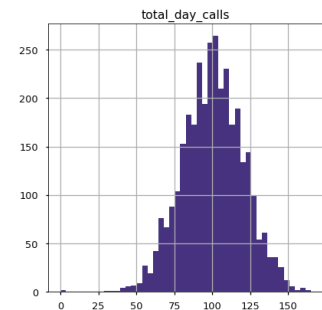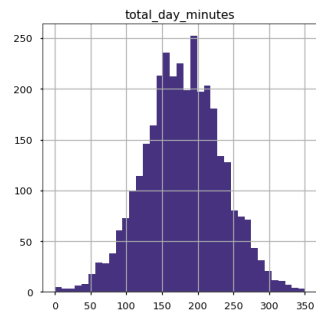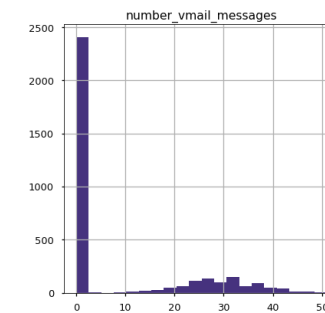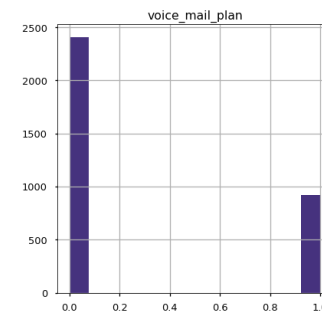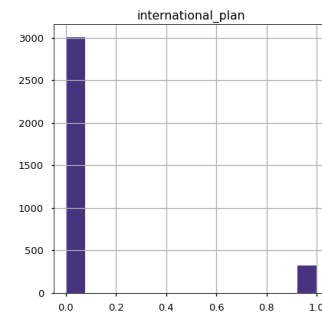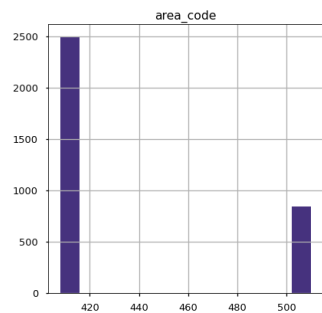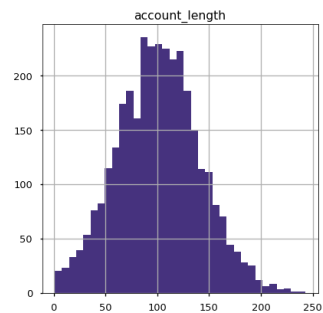
These look great. Reminds me of why I choose this dataset in the first place!

```
# Before summarizing the table above, I am going to take a peek at the area code column vs the prefix columns.

df['area_code'].value_counts()


pd.pivot_table(df,
               values='account_length',
               index='area_code',
               columns='phone_number_prefix',
               aggfunc='count'
               )
```

phone_number_prefix	327	328	329	330	331	332	333	334	335	336	...	413	414	415	416	417	418	419	420	421	422
area_code																					
408	3	9	9	5	6	12	13	11	10	14	...	6	4	10	10	15	6	6	9	5	7
415	10	12	17	20	10	21	25	20	16	20	...	12	15	21	22	21	21	13	18	13	9
510	6	11	11	9	9	11	8	7	8	7	...	10	10	5	5	10	5	8	8	6	3

3 rows × 96 columns

We only have 3 area codes for the all the prefixes that we see. We may want to encode this category since it is categorical feature. I will also encode the phone number prefix, and the state columns before running any models. Going to continue to review.

```
# Updating international and voicemail indicators to boolean, and area code and phone_num prefix to strings

df['area_code'] = df['area_code'].astype(str)
df['phone_number_prefix'] = df['phone_number_prefix'].astype(str)


df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 24 columns):
 #   Column               Non-Null Count   Dtype
---  ------               --------------   -----
 0   account_length       3333 non-null    int64
 1   area_code            3333 non-null    object
 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
19   total_calls            3333 non-null   int64
20   total_charges          3333 non-null   float64
21   total_minutes          3333 non-null   float64
22   price_per_minute       3333 non-null   float64
23   phone_number_prefix    3333 non-null   object
dtypes: float64(11), int64(11), object(2)
memory usage: 625.1+ KB
```

In [318...]
```python
# Creating a heatmap to look at colinearity and potential categories that will lead to churn prediction.

corr = df.corr().abs()

fix, ax = plt.subplots(figsize = (24,15))
matrix = np.triu(corr)
ax.set_title('Feature Correlation Matrix', pad=15, fontsize=15)
heatmap = sns.heatmap(corr, annot=True, cmap=pal, fmt='.2f', mask=matrix, linewidths=1)
plt.show()
```

Feature Correlation Matrix
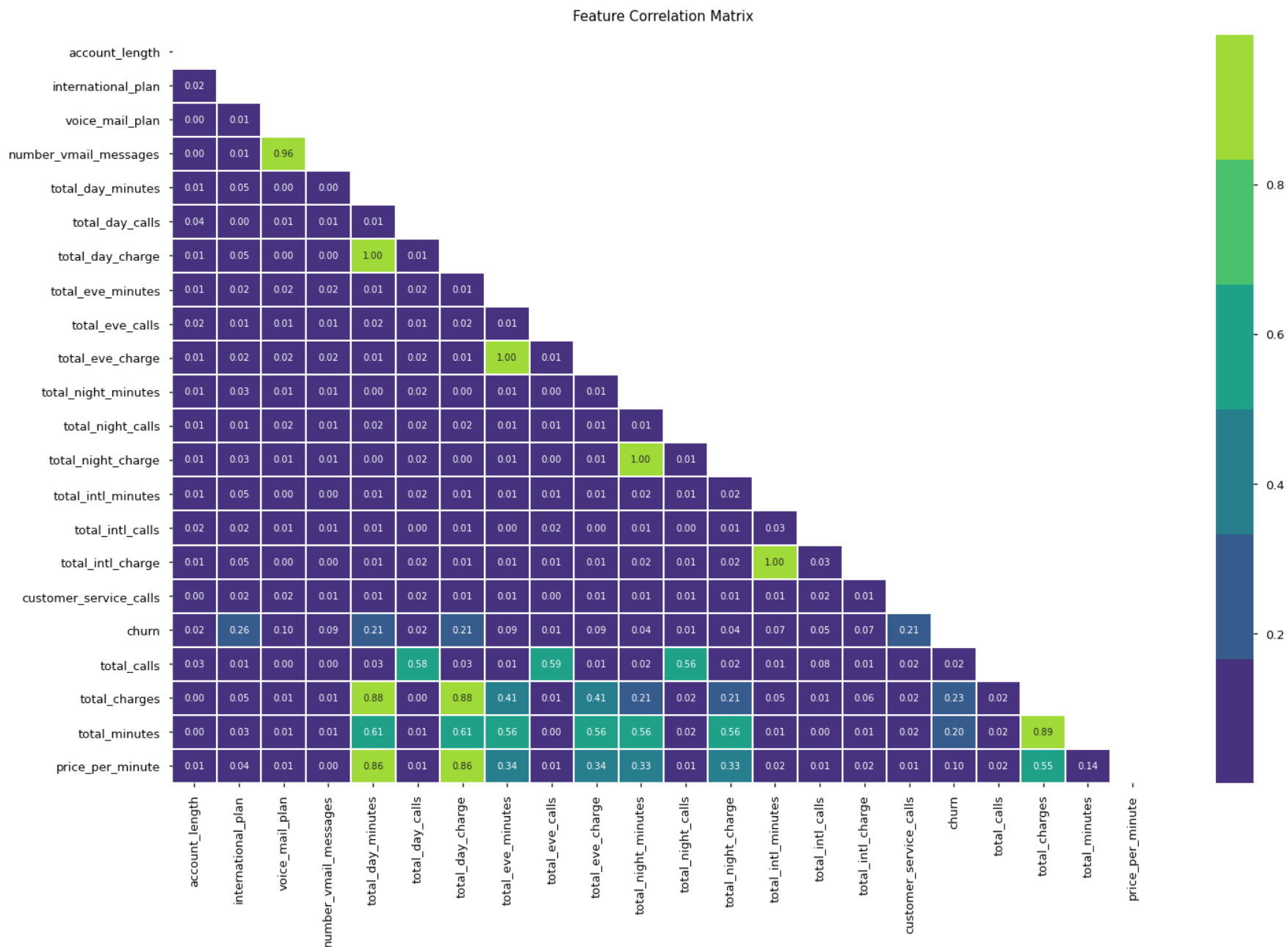
Charges and total minutes are perfectly correlated, which makes sense. Also the number of voicemail messages, and the voice mail plan category are also highly correlated. Meaning if they purchased the plan, it is highly likely that they used it. Day minutes were highly correlated with the total minutes vs evening minutes. It looks like churn had the highest correlation with the international plan

(which we identified as a possibility earlier). Total day minutes, total day charges, and customer service calls. Customer service calls could be related to the actual service that the customer experienced, or could be the fact that the customer had to call in to cancel their plan, thus driving that columns correlation higher.

Going to drop the columns we added earlier, and columns that are dependent on another for its calculation (i.e. charges). They are too dependent on the source features.

```python
df = df.drop(['total_charges',
        'total_intl_charge',
        'total_eve_charge',
        'total_day_charge',
        'total_night_charge',
        'number_vmail_messages',
        'total_calls',
        'total_minutes',
        'price_per_minute'
        ],
        axis=1)
```

```python
# Creating a heatmap to look at colinearity and potential categories that will lead to churn prediction.

corr = df.corr().abs()

fix, ax = plt.subplots(figsize = (24,15))
matrix = np.triu(corr)
ax.set_title('Feature Correlation Matrix', pad=15, fontsize=15)
heatmap = sns.heatmap(corr, annot=True, cmap=pal, fmt='.2f', mask=matrix, linewidths=1)
plt.show()
```

Feature Correlation Matrix
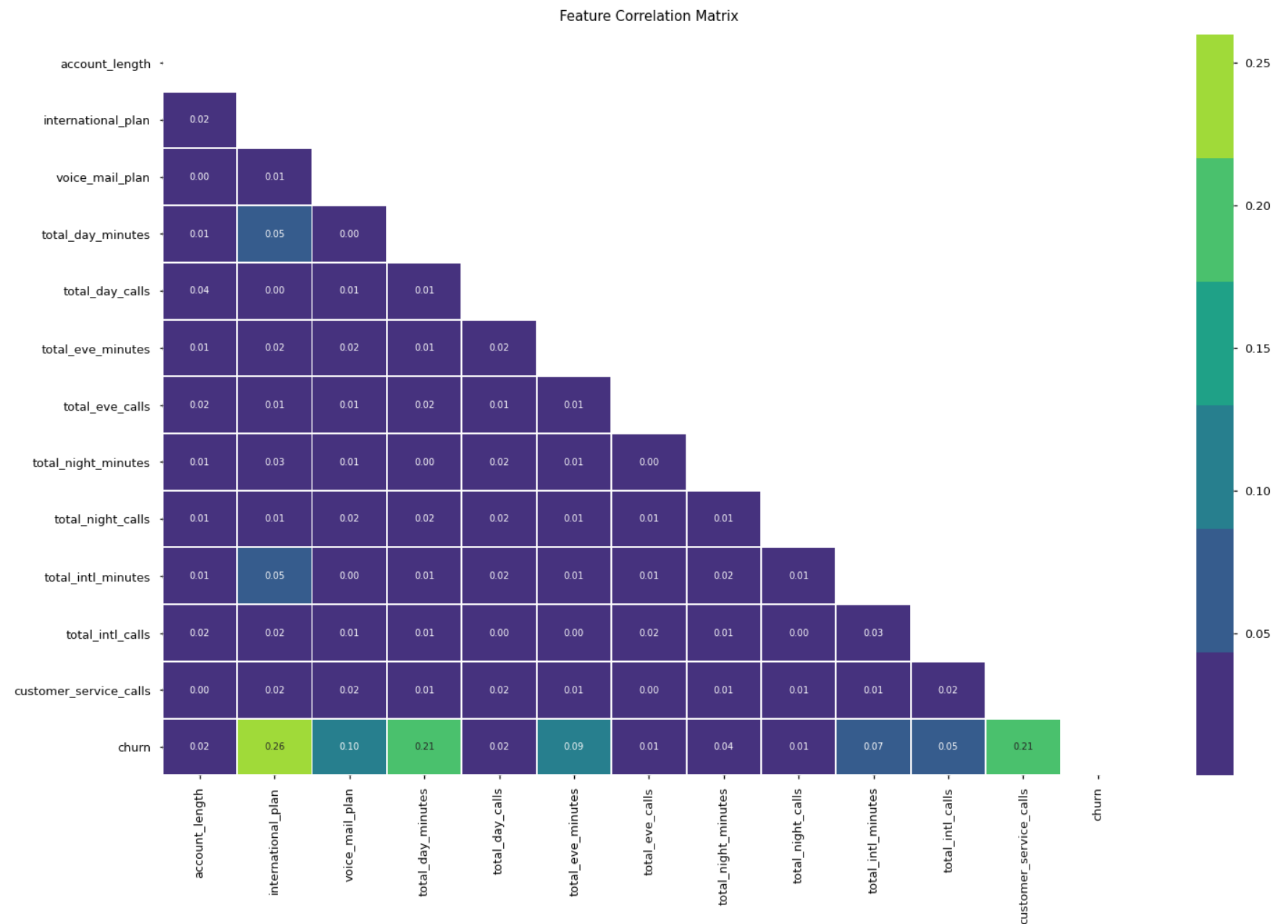
Looking much better. Looks liek that international plan, the total day minutes, and customer serivce calls are all showing some correlation to our target variable of churn. I would expect these to manifest again at the end when we are looking at feature importances of our model.

```python
# Checking our target/ dataset for balance before creating our baseline classifcation model.

print('Raw Counts')
print(df['churn'].value_counts())
print()
print('Percentages')
print(df['churn'].value_counts(normalize=True))
```

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

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

A baseline model that always chose the majority class would have an accuracy of over 85%. Therefore we will want to report additional metrics at the end.

Classfication Modeling

Now we are going to get into our classification modeling. We are going to first try and create a psuedo pipeline to do some of the lifting as we move between our models. In addition, we will be doing some oversampling in order to account for our target imbalance.

```python
# Quick look at our dataframe

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   account_length     3333 non-null   int64
 1   area_code          3333 non-null   object
 2   international_plan  3333 non-null   int64
 3   voice_mail_plan    3333 non-null   int64
 4   total_day_minutes  3333 non-null   float64
 5   total_day_calls    3333 non-null   int64
 6   total_eve_minutes  3333 non-null   float64
 7   total_eve_calls    3333 non-null   int64
 8   total_night_minutes 3333 non-null  float64
 9   total_night_calls  3333 non-null   int64
```

```
 10  total_intl_minutes     3333 non-null   float64
 11  total_intl_calls       3333 non-null   int64
 12  customer_service_calls 3333 non-null   int64
 13  churn                  3333 non-null   int64
 14  phone_number_prefix    3333 non-null   object
dtypes: float64(4), int64(9), object(2)
memory usage: 390.7+ KB
```

In [323…
```python
# Dropping out superfulous columns here before we do our X, y splits for test/ train.

df = df.drop(['area_code', 'phone_number_prefix'], axis=1)
```

In [324…
```python
# Situate target and non-target features

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

# Create splits

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

In [325…
```python
# Also looking at a dummy model with 5 cross validation folds.  Mean accuracy is about
# 86%.  This aligns with our assumption above with the imbalance of our churned
# customer count.

dummy_model = DummyClassifier(strategy='most_frequent')

cv_results = cross_val_score(dummy_model,
                             X_train,
                             y_train,
                             cv=5)

dummy_model.fit(X_train, y_train)

np.mean(cv_results)
```

Out[325… 0.8567430861723446

Case and point from our statement earlier, a most frequent model could capture around 86% of the correct cases. With that being said, the metric we will be evaluating is recall. Recall is calculated by taking the number of True Positives divided by the True Positives and the False Negative classes. Recall is typically good when we want to limit false negatives. For our use case, false negatives would be a customer that churned, and we didn't predict them to churn, thus never giving our partners in Customer Success the opportunity to "save" the client.

Pipeline

```
In [326... # Labeling columns for different preprocessing steps

         categorical_columns = []

         numerical_columns = ['account_length', 'total_day_minutes',
                             'total_day_calls', 'total_eve_minutes', 'total_eve_calls',
                             'total_night_calls', 'total_intl_minutes', 'total_intl_calls',
                             'customer_service_calls', 'total_night_minutes'
                             ]

         binary_columns = ['international_plan', 'voice_mail_plan']
```

```
In [327... df.columns
```

```
Out[327... Index(['account_length', 'international_plan', 'voice_mail_plan',
                'total_day_minutes', 'total_day_calls', 'total_eve_minutes',
                'total_eve_calls', 'total_night_minutes', 'total_night_calls',
                'total_intl_minutes', 'total_intl_calls', 'customer_service_calls',
                'churn'],
               dtype='object')
```

```
In [328... # Check to make sure we have all our columns accounted for

         (len(categorical_columns)+len(numerical_columns)+len(binary_columns)) == (df.shape[1]-1)
```

```
Out[328... True
```

```
In [329... # Ended up not using any sort of OHE for data

         # One Hot Encoding out categorical variables on our training data

         #categorical_Xtr_df = pd.DataFrame(X_train, columns=categorical_columns)
         #
         #ohe = OneHotEncoder(drop='first',
         #                   sparse = False)
         #
         #ohe_data = ohe.fit_transform(categorical_Xtr_df)
         #
         #
         ## Dum dums
         #
```

```python
#ohe_data_df = pd.DataFrame(ohe_data,
#                           columns=ohe.get_feature_names(),
#                           index=categorical_Xtr_df.index)#make sure to pass an index
#
#clean_df = X_train.drop(categorical_Xtr_df, axis=1)
#
#
## Putting humpty back together again
#
#X_train = pd.concat([clean_df, ohe_data_df], axis=1)
#
#
## Quick peek
#
#X_train.head()
```

```python
# Ended up not using any sort of OHE for data

# One Hot Encoding out categorical variables on our testing data

#categorical_Xts_df = pd.DataFrame(X_test, columns=categorical_columns)
#
#ohe = OneHotEncoder(drop='first',
#                    sparse = False)
#
#ohe_data = ohe.fit_transform(categorical_Xts_df)
#
## Dum dums
#
#ohe_data_df = pd.DataFrame(ohe_data,
#                           columns=ohe.get_feature_names(),
#                           index=categorical_Xts_df.index)#make sure to pass an index
#
#clean_df = X_test.drop(categorical_Xts_df, axis=1)
#
## Putting humpty back together again
#
#X_test = pd.concat([clean_df, ohe_data_df], axis=1)
#
## Quick peek
#X_test.head()
```

```python
# Saving a copy of our data frame to reference columns later.

df_X_train_copy = X_train.iloc[:10]
```

```
df X test copy = X test.iloc[:10]
```

Logistic Regression Model

Baseline Log Reg

```python
# Baseline Logistic Regression model

baseline_logreg = ImPipeline(steps=[('sm', SMOTE(random_state=42)),
                                     ('scale', StandardScaler()),
                                     ('estimator', LogisticRegression(random_state=42))])

# Train model

baseline_logreg.fit(X_train, y_train);
```

```python
# Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

divider = ('-----' * 10)
# Capture roc_auc for test, and train

baseline_logreg_roc_score_train = roc_auc_score(y_train, baseline_logreg.predict(X_train))
baseline_logreg_roc_score_test = roc_auc_score(y_test, baseline_logreg.predict(X_test))


# Capture recall scores for test and train

baseline_logreg_recall_score_train_cv = cross_val_score(estimator=baseline_logreg, X=X_train,  y=y_train,
                                            cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


baseline_logreg_recall_score_train = recall_score(y_train, baseline_logreg.predict(X_train))
baseline_logreg_recall_score_test = recall_score(y_test, baseline_logreg.predict(X_test))


# Capture f1 scores for test and train

baseline_logreg_f1_score_train = f1_score(y_train, baseline_logreg.predict(X_train))
baseline_logreg_f1_score_test = f1_score(y_test, baseline_logreg.predict(X_test))
```

```python
# Capture precision scores for test and train

baseline_logreg_precision_score_train = precision_score(y_train, baseline_logreg.predict(X_train))
baseline_logreg_precision_score_test = precision_score(y_test, baseline_logreg.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {baseline_logreg_roc_score_train :.2%}")
print(f" Test ROC Score: {baseline_logreg_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {baseline_logreg_recall_score_train :.2%}")
print(f" Test Recall score: {baseline_logreg_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {baseline_logreg_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {baseline_logreg_f1_score_train :.2%}")
print(f" Test F1 score: {baseline_logreg_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {baseline_logreg_precision_score_train :.2%}")
print(f" Test Precision score: {baseline_logreg_precision_score_test :.2%}")
```

```
 Performance Comparison

 --------------------------------------------------
 Train ROC Score: 70.99%
 Test ROC Score: 72.23%
 --------------------------------------------------
 Train Recall score: 72.35%
 Test Recall score: 72.80%
 Mean Cross Validated Recall Score: 71.51%
 --------------------------------------------------
 Train F1 score: 40.88%
 Test F1 score: 43.65%
 --------------------------------------------------
 Train Precision score: 28.49%
 Test Precision score: 31.16%
 --------------------------------------------------
```

Our model is performing ok. The fact that our scores are so close means that we probably have some slight underfitting going on (since our test is higher than our train). In addition, our precision is pretty poor meaning that we have a high number of false positives. As we mentioned that probably isn't the end of the world but something. we want to keep in mind as we complete additional models.

```python
# Plotting confusion matrix for our baseline logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_logreg, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Logistic Regression Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# Plotting confusion matrix for our baseline logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_logreg, X_test, y_test, ax=ax, cmap='Blues', normalize='true')
ax.set_title("Baseline Logistic Regression Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```


Baseline Logistic Regression Confusion Matrix - Train

Baseline Logistic Regression Confusion Matrix - Test

	Non-churn	churn
Non-churn	0.72	0.28
churn	0.27	0.73

True label / Predicted label

In [335...

```python
# Print classification Scores for the test set

y_pred = baseline_logreg.predict(X_test)
divider = ('-' * 60)
table = classification_report(y_test, y_pred, digits=3)

print('\n', 'Classification Report - Test', '\n')
print(divider)
print(table)
```

Classification Report - Test

```
------------------------------------------------------------
              precision    recall  f1-score   support

           0      0.937     0.717     0.812       709
           1      0.312     0.728     0.436       125

    accuracy                          0.718       834
   macro avg      0.624     0.722     0.624       834
weighted avg      0.844     0.718     0.756       834
```

```
In [336…    # Quick look at the performance of our baseline model.  We'll take a peek
            # at the ROC curve first, even though our metric of interest is recall, and F1.

            fig, ax2 = plt.subplots(figsize=(8,5))
            plot_roc_curve(baseline_logreg, X_train, y_train, ax=ax2, name ='Train ROC curve', color=pal[1])
            plot_roc_curve(baseline_logreg, X_test, y_test, ax=ax2, name ='Test ROC curve', color=pal[4])
            ax2.plot([0, 1], [0, 1], color='lightgray', lw=1, linestyle='-.')
            ax2.set_xlabel('False Positive Rate')
            ax2.set_ylabel('True Positive Rate')
            ax2.set_title('Receiver operating characteristic (ROC) Curve')
            plt.show();
```
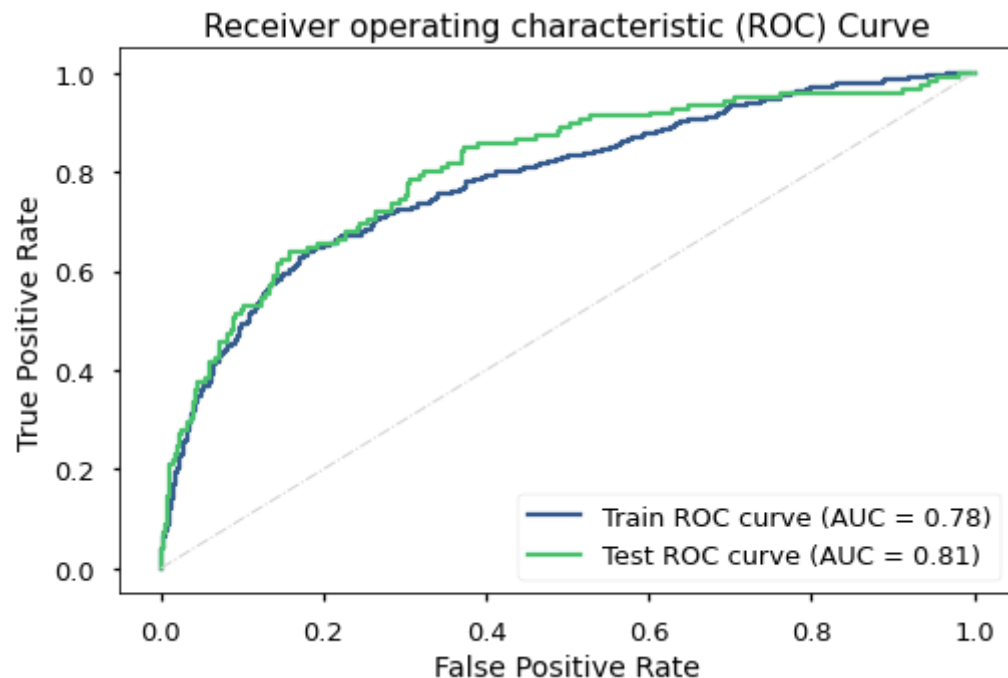


```
In [337…    # Since we are more focused on our precision and recall we are going to look at the precision/ recall curve as

            y_train_score = baseline_logreg.predict(X_train)
            y_test_score = baseline_logreg.predict(X_test)


            # Calculate precision and recall

            precision, recall, thresholds = precision_recall_curve(y_test, y_test_score)
            precision_train, recall_train, thresholds_train = precision_recall_curve(y_train, y_train_score)
```

```python
# Create precision recall curve

fig, ax = plt.subplots(figsize=(8,5))
ax.plot(recall, precision, color=pal[1], label='Test Precision/ Recall curve')
ax.plot(recall_train, precision_train, color=pal[4], label='Train Precision/ Recall curve')


# Add axis labels to plot

ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
ax.legend()

plt.show()
```
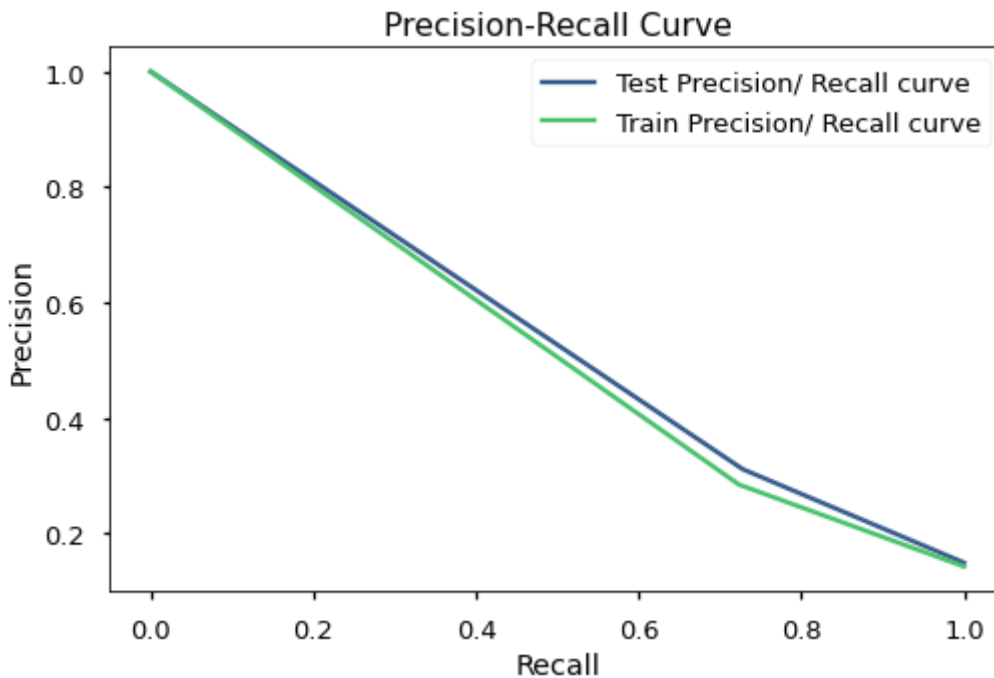

Our model seems to be peforming better on our training data set vs our test data set. This would suggest some overfitting happening. We will try and penalize our model and look into some hyperparameter tuning.

Tuned Logisitic Regression model

Next we will look at tuning our baseline model with some hyperparameters. We will select some random parameters in addition to the default values to see if we can improve the recall score, and ultimately the F1 score of our model.

```python
# Parameters for our gridsearch, model optimization

parameters = {
    'estimator__penalty' : ['l1', 'l2', 'elasticnet'],
    'estimator__fit_intercept':[True, False],
    'estimator__C'        : [20, 50, 100],
    'estimator__solver'   : ['newton-cg', 'lbfgs', 'liblinear'],
    'estimator__max_iter' : [50, 100, 200, 300]
}


# Create the grid, with "logreg_pipeline" as the estimator

best_logreg = GridSearchCV(estimator=baseline_logreg,
                           param_grid=parameters,
                           scoring='recall',
                           cv=5,
                           n_jobs=-1
)
```

In [339…]
```python
# Train the pipeline (tranformations & predictor)

best_logreg.fit(X_train, y_train);


# Let's take a look at our best parameters

best_logreg.best_params_
```

Out[339…]
```
{'estimator__C': 20,
 'estimator__fit_intercept': False,
 'estimator__max_iter': 50,
 'estimator__penalty': 'l1',
 'estimator__solver': 'liblinear'}
```

In [340…]
```python
# Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

# Capture roc_auc for test, and train

best_logreg_roc_score_train = roc_auc_score(y_train, best_logreg.predict(X_train))
best_logreg_roc_score_test = roc_auc_score(y_test, best_logreg.predict(X_test))
```

```python
# Capture recall scores for test and train

best_logreg_recall_score_train_cv = cross_val_score(estimator=best_logreg, X=X_train,  y=y_train,
                                  cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


best_logreg_recall_score_train = recall_score(y_train, best_logreg.predict(X_train))
best_logreg_recall_score_test = recall_score(y_test, best_logreg.predict(X_test))


# Capture f1 scores for test and train

best_logreg_f1_score_train = f1_score(y_train, best_logreg.predict(X_train))
best_logreg_f1_score_test = f1_score(y_test, best_logreg.predict(X_test))


# Capture precision scores for test and train

best_logreg_precision_score_train = precision_score(y_train, best_logreg.predict(X_train))
best_logreg_precision_score_test = precision_score(y_test, best_logreg.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {best_logreg_roc_score_train :.2%}")
print(f" Test ROC Score: {best_logreg_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {best_logreg_recall_score_train :.2%}")
print(f" Test Recall score: {best_logreg_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {best_logreg_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {best_logreg_f1_score_train :.2%}")
print(f" Test F1 score: {best_logreg_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {best_logreg_precision_score_train :.2%}")
print(f" Test Precision score: {best_logreg_precision_score_test :.2%}")
print(divider, '\n')
```

```
Performance Comparison

-------------------------------------------------------------
 Train ROC Score: 70.74%
 Test ROC Score: 72.32%
```

```
-------------------------------------------------------------
 Train Recall score: 73.18%
 Test Recall score: 74.40%
 Mean Cross Validated Recall Score: 72.63%
-------------------------------------------------------------
 Train F1 score: 40.34%
 Test F1 score: 43.36%
-------------------------------------------------------------
 Train Precision score: 27.84%
 Test Precision score: 30.59%
-------------------------------------------------------------
```

Looks like our recall metric is performing better by a few points. Up to 88 on the training set and 90 on the test set. We are probably under fitting a little but not sure if that is a big deal since we are already beating our most frequent baseline of 85%.

In [341…

```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_logreg, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Logistic Regression Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_logreg, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Logistic Regression Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```

Best Logistic Regression Confusion Matrix - Train

	Non-churn	churn
Non-churn	0.68	0.32
churn	0.27	0.73

True label / Predicted label

Best Logistic Regression Confusion Matrix - Test

	Non-churn	churn
Non-churn	0.7	0.3
churn	0.26	0.74

True label / Predicted label
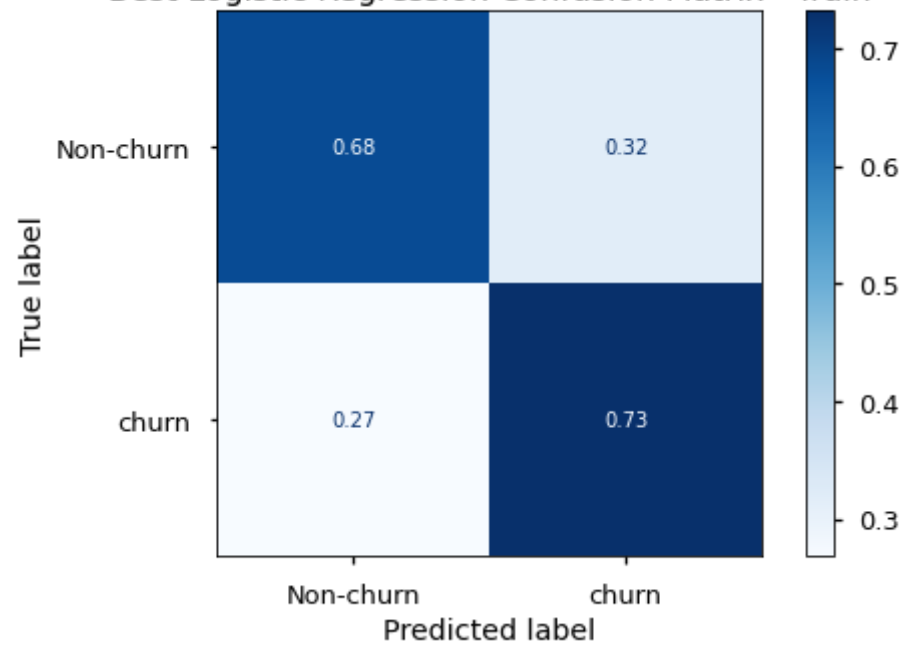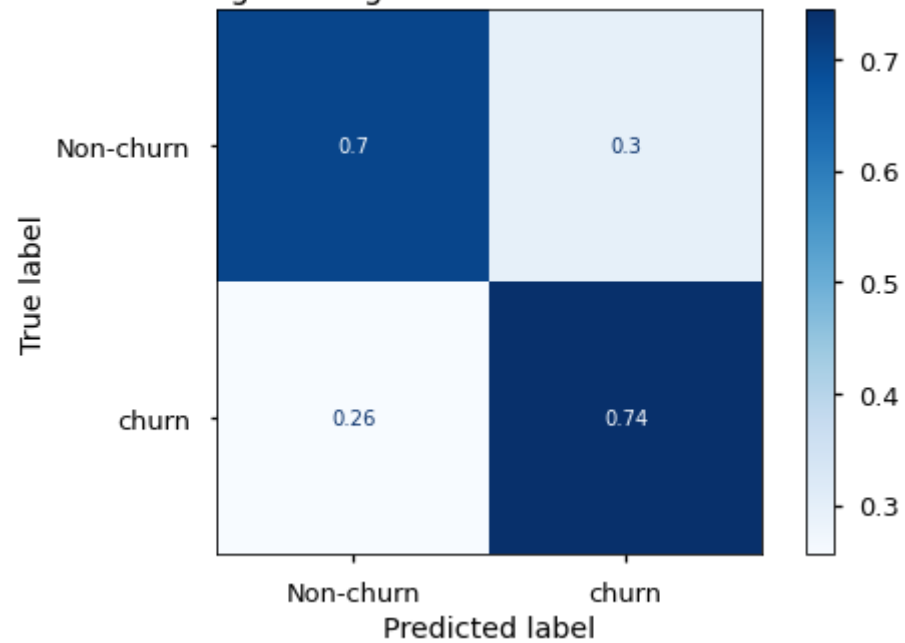
```
In [342...   # Print classification scores for the test set

             y_pred = best_logreg.predict(X_test)
             divider = ('-' * 60)
             table = classification_report(y_test, y_pred, digits=3)

             print('\n', 'Classification Report - Test', '\n')
             print(divider)
             print(table)
```

```
Classification Report - Test

------------------------------------------------------------
              precision    recall  f1-score   support

           0      0.940     0.702     0.804       709
           1      0.306     0.744     0.434       125

    accuracy                          0.709       834
   macro avg      0.623     0.723     0.619       834
weighted avg      0.845     0.709     0.748       834
```

Again peforming better on our recall scores -- however our f1 score is lagging behind our baseline model. Primarily because we aren't as concerned with the false positives as we mentioned before.

```
In [343...   # Checking in on our ROC curve on our tuned model.

             fig, ax2 = plt.subplots(figsize=(8,5))
             plot_roc_curve(best_logreg, X_train, y_train, ax=ax2, name ='Train ROC curve', color=pal[1])
             plot_roc_curve(best_logreg, X_test, y_test, ax=ax2, name ='Test ROC curve', color=pal[4])
             ax2.plot([0, 1], [0, 1], color='lightgray', lw=1, linestyle='-.')
             ax2.set_xlabel('False Positive Rate')
             ax2.set_ylabel('True Positive Rate')
             ax2.set_title('Receiver operating characteristic (ROC) Curve')
             plt.show();
```

Receiver operating characteristic (ROC) Curve

Train ROC curve (AUC = 0.78)
Test ROC curve (AUC = 0.81)
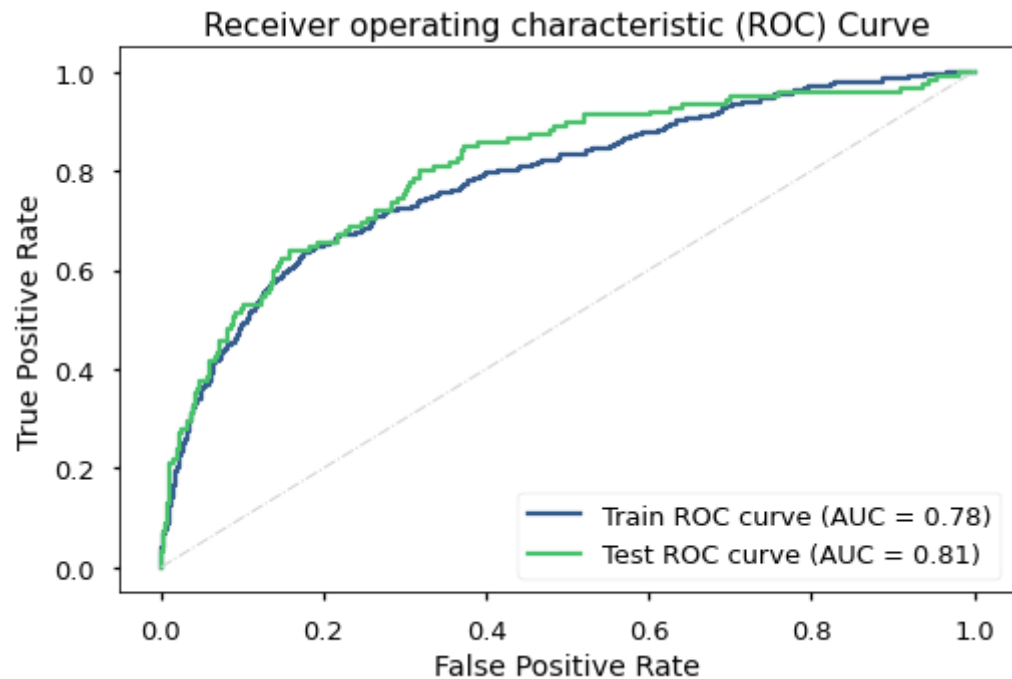
In [344…
```python
# Since we are more focused on our precision and recall we are going to look at the precision/ recall curve as

y_train_score = best_logreg.predict(X_train)
y_test_score = best_logreg.predict(X_test)


# Calculate precision and recall

precision, recall, thresholds = precision_recall_curve(y_test, y_test_score)
precision_train, recall_train, thresholds_train = precision_recall_curve(y_train, y_train_score)


# Create precision recall curve

fig, ax = plt.subplots(figsize=(8,5))
ax.plot(recall, precision, color=pal[1], label='Test Precision/ Recall curve')
ax.plot(recall_train, precision_train, color=pal[4], label='Train Precision/ Recall curve')


# Add axis labels to plot

ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
```

```
ax.set_xlabel('Recall')
ax.legend()

plt.show()
```
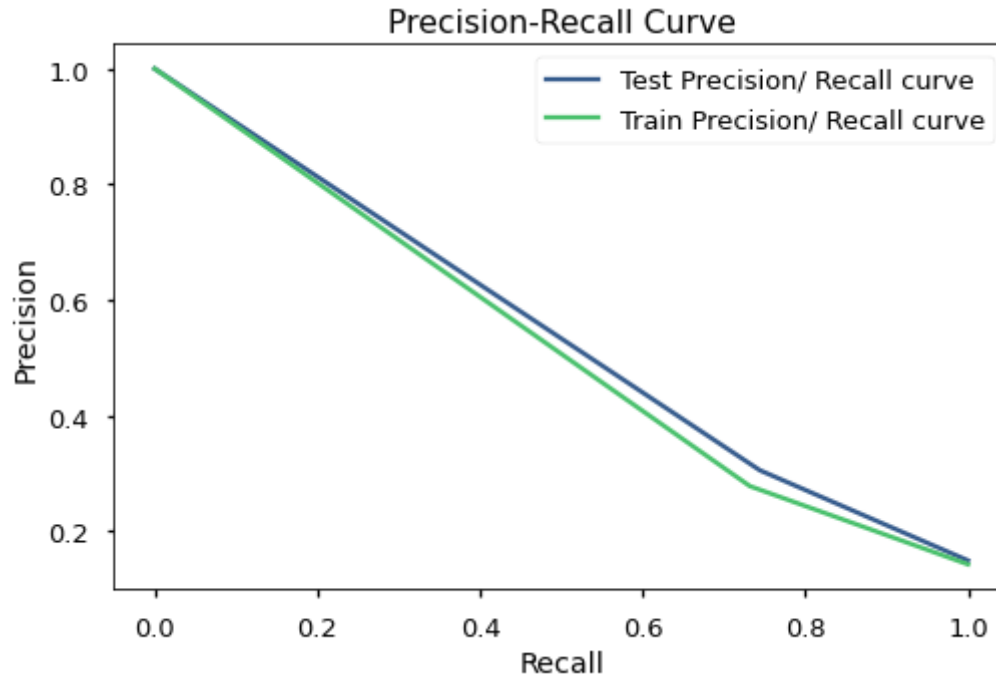

Build Ridge Model

Baseline Ridge Model

First we will start with our baseline model. See if we can out perform out logistic regression baseline and tuned models.

```
In [345…   # Baseline model
           baseline_ridge = ImPipeline(steps=[('sm', SMOTE(random_state=42)),
                                              ('scale', StandardScaler()),
                                              ('estimator', RidgeClassifier(random_state=42))])

           # Train model
           baseline_ridge.fit(X_train, y_train);
```

```python
In [346...  # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

            # Capture roc_auc for test, and train

            baseline_tree_roc_score_train = roc_auc_score(y_train, baseline_tree.predict(X_train))
            baseline_tree_roc_score_test = roc_auc_score(y_test, baseline_tree.predict(X_test))

            # Capture recall scores for test and train

            baseline_ridge_recall_score_train_cv = cross_val_score(estimator=baseline_ridge, X=X_train,  y=y_train,
                                                cv=StratifiedKFold(shuffle=True), scoring='recall').mean()

            baseline_ridge_recall_score_train = recall_score(y_train, baseline_ridge.predict(X_train))
            baseline_ridge_recall_score_test = recall_score(y_test, baseline_ridge.predict(X_test))


            # Capture f1 scores for test and train

            baseline_ridge_f1_score_train = f1_score(y_train, baseline_ridge.predict(X_train))
            baseline_ridge_f1_score_test = f1_score(y_test, baseline_ridge.predict(X_test))


            # Capture precision scores for test and train

            baseline_ridge_precision_score_train = precision_score(y_train, baseline_ridge.predict(X_train))
            baseline_ridge_precision_score_test = precision_score(y_test, baseline_ridge.predict(X_test))

            print('\n', "Performance Comparison", '\n')
            print(divider)

            print(f" Train ROC Score: {best_logreg_roc_score_train :.2%}")
            print(f" Test ROC Score: {best_logreg_roc_score_test :.2%}")
            print(divider)

            print(f" Train Recall score: {baseline_ridge_recall_score_train :.2%}")
            print(f" Test Recall score: {baseline_ridge_recall_score_test :.2%}")
            print(f" Mean Cross Validated Recall Score: {baseline_ridge_recall_score_train_cv :.2%}")
            print(divider)

            print(f" Train F1 score: {baseline_ridge_f1_score_train :.2%}")
            print(f" Test F1 score: {baseline_ridge_f1_score_test :.2%}")
            print(divider)

            print(f" Train Precision score: {baseline_ridge_precision_score_train :.2%}")
```

```
    print(f" Test Precision score: {baseline_ridge_precision_score_test :.2%}")
```

Performance Comparison

```
------------------------------------------------------------
 Train ROC Score: 70.74%
 Test ROC Score: 72.32%
------------------------------------------------------------
 Train Recall score: 73.18%
 Test Recall score: 76.00%
 Mean Cross Validated Recall Score: 73.47%
------------------------------------------------------------
 Train F1 score: 40.40%
 Test F1 score: 43.98%
------------------------------------------------------------
 Train Precision score: 27.90%
 Test Precision score: 30.94%
------------------------------------------------------------
```

In [347...

```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_ridge, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Ridge Regression Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);


# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_ridge, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Ridge Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```

Baseline Ridge Regression Confusion Matrix - Train
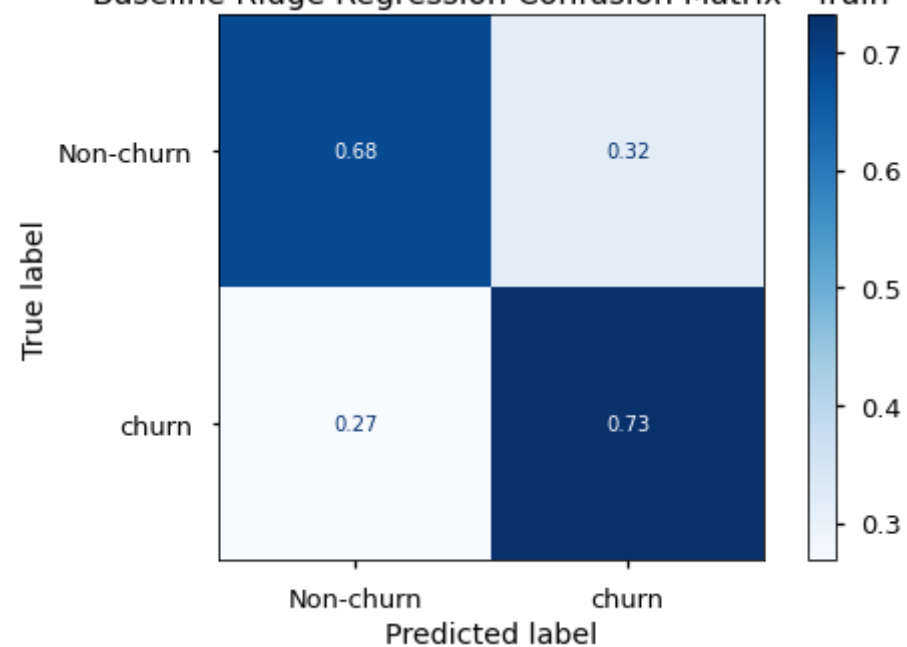
Baseline Ridge Confusion Matrix - Test
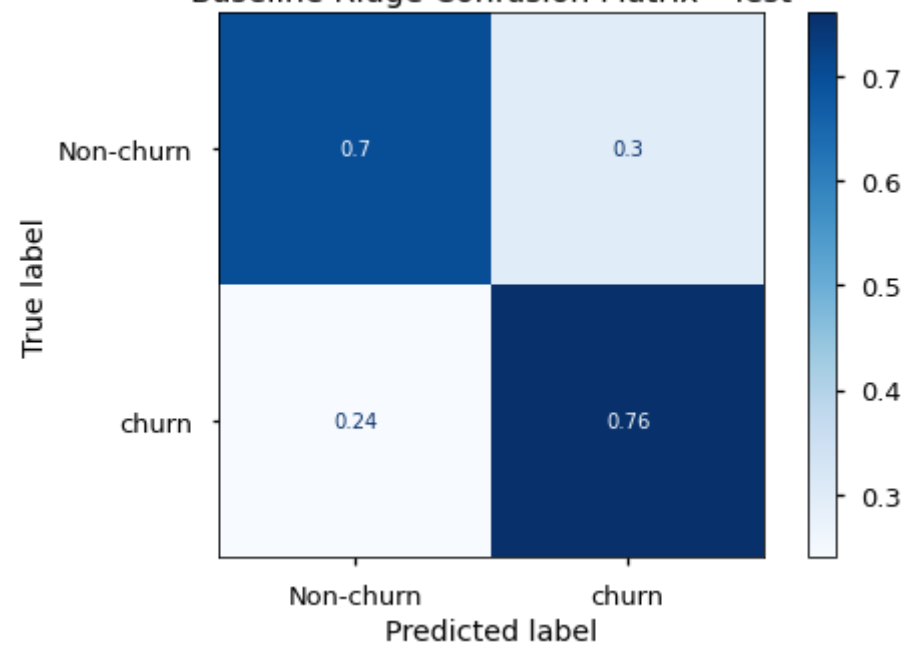
Very similar results as our logistic regression which isn't super suprising. We will continue to move forward with this model to see if the tuned version can perform any better than our baseline or hyper tuned logistic regressions.

Tuned Ridge Model

```
In [348…    # Parameters for our gridsearch, model optimization

            parameters = {
                'estimator__alpha' : ['1', '10', '20', '50'],
                'estimator__fit_intercept':[True, False],
                'estimator__solver'  : ['auto', 'lsqr', 'svd', 'sag', 'saga'],
                'estimator__max_iter'  : [None, 50, 100, 200, 300]
            }

            # Create the grid, with "logreg_pipeline" as the estimator
            best_ridge = GridSearchCV(estimator=baseline_ridge,   # model
                                      param_grid=parameters,
                                      scoring='recall',            # my metric for scoring
                                      cv=5,                        # default folds
                                      n_jobs=-1
            )
```

```
In [349…    # Train the pipeline (tranformations & predictor)

            best_ridge.fit(X_train, y_train)

            # Let's take a look at our best parameters

            best_ridge.best_params_
```

```
Out[349…   {'estimator__alpha': '1',
            'estimator__fit_intercept': True,
            'estimator__max_iter': None,
            'estimator__solver': 'auto'}
```

```
In [350…    # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

            # Capture roc_auc for test, and train

            baseline_tree_roc_score_train = roc_auc_score(y_train, baseline_tree.predict(X_train))
            baseline_tree_roc_score_test = roc_auc_score(y_test, baseline_tree.predict(X_test))

            # Capture recall scores for test and train
```

```python
best_ridge_recall_score_train_cv = cross_val_score(estimator=best_ridge, X=X_train,  y=y_train,
                                    cv=StratifiedKFold(shuffle=True), scoring='recall').mean()

best_ridge_recall_score_train = recall_score(y_train, best_ridge.predict(X_train))
best_ridge_recall_score_test = recall_score(y_test, best_ridge.predict(X_test))


# Capture f1 scores for test and train

best_ridge_f1_score_train = f1_score(y_train, best_ridge.predict(X_train))
best_ridge_f1_score_test = f1_score(y_test, best_ridge.predict(X_test))


# Capture precision scores for test and train

best_ridge_precision_score_train = precision_score(y_train, best_ridge.predict(X_train))
best_ridge_precision_score_test = precision_score(y_test, best_ridge.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)

print(f" Train ROC Score: {best_logreg_roc_score_train :.2%}")
print(f" Test ROC Score: {best_logreg_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {best_ridge_recall_score_train :.2%}")
print(f" Test Recall score: {best_ridge_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {best_ridge_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {best_ridge_f1_score_train :.2%}")
print(f" Test F1 score: {best_ridge_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {best_ridge_precision_score_train :.2%}")
print(f" Test Precision score: {best_ridge_precision_score_test :.2%}")
print(divider, '\n')
```

```
 Performance Comparison

---------------------------------------------------------------
 Train ROC Score: 70.74%
 Test ROC Score: 72.32%
---------------------------------------------------------------
 Train Recall score: 73.18%
 Test Recall score: 76.00%
```

```
Mean Cross Validated Recall Score: 73.76%
---------------------------------------------------------------
Train F1 score: 40.40%
Test F1 score: 43.98%
---------------------------------------------------------------
Train Precision score: 27.90%
Test Precision score: 30.94%
---------------------------------------------------------------
```

In [351…

```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_ridge, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Ridge Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_ridge, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Ridge Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```

Best Ridge Confusion Matrix - Train

	Non-churn	churn
Non-churn	0.68	0.32
churn	0.27	0.73

Best Ridge Confusion Matrix - Test

	Non-churn	churn
Non-churn	0.7	0.3
churn	0.24	0.76
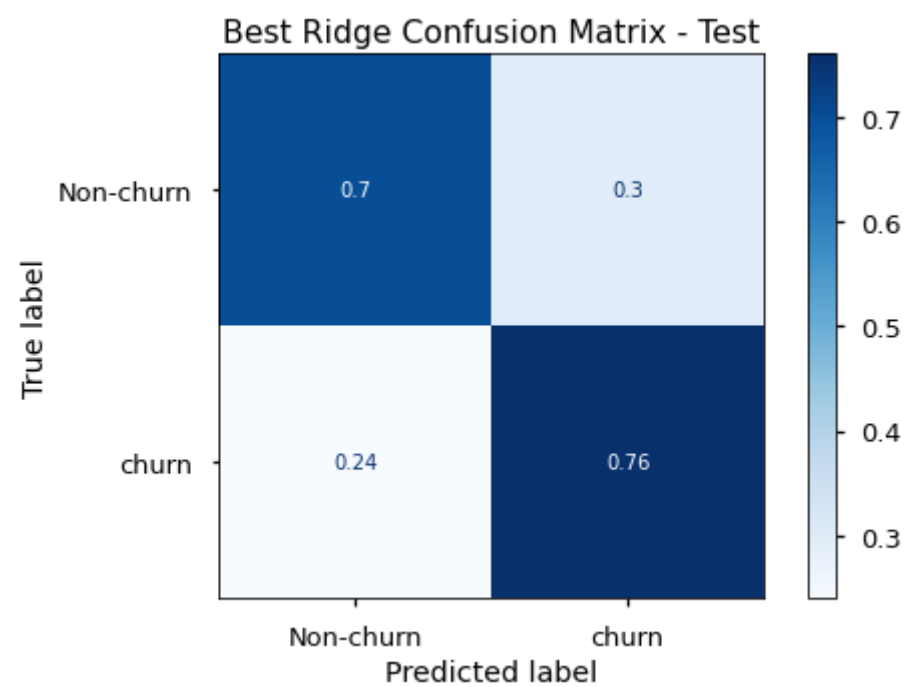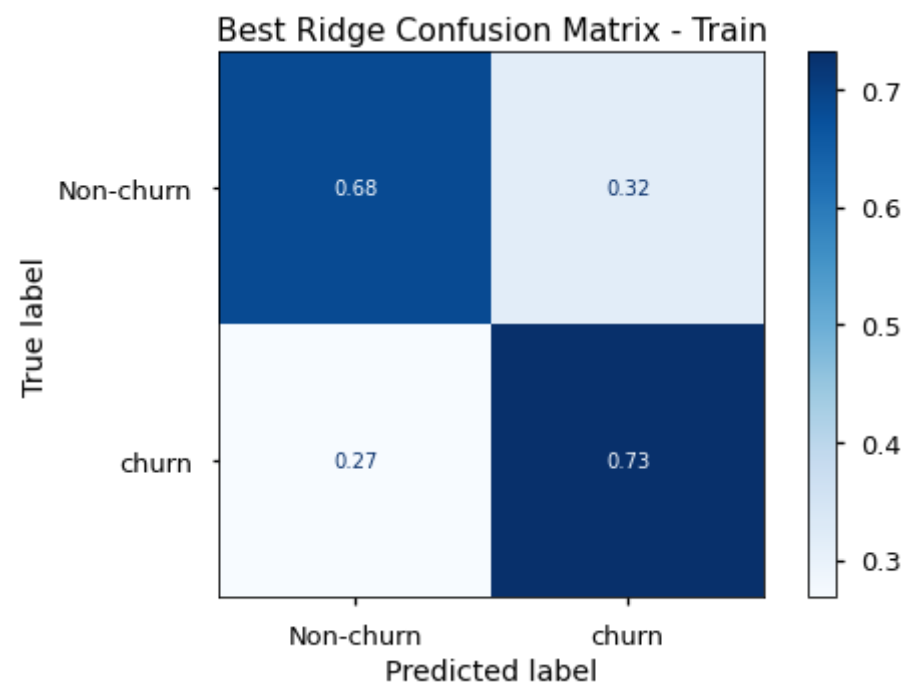
Our tuned ridge model although performing better than it's respective baseline, the tuned results are similar to the baseline and tuned logistic regression models. If it were a choice at this point we would still entertain our core logistic regression model for classification.

Decision Tree Model

Baseline Decision Tree

```python
In [352…  # Baseline model
          baseline_tree = ImPipeline(steps=[('sm', SMOTE(random_state=42)),
                                            ('scale', StandardScaler()),
                                            ('estimator', DecisionTreeClassifier(random_state=42))])

          # Train model
          baseline_tree.fit(X_train, y_train);
```

```python
In [353…  # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

          # Capture roc_auc for test, and train

          baseline_tree_roc_score_train = roc_auc_score(y_train, baseline_tree.predict(X_train))
          baseline_tree_roc_score_test = roc_auc_score(y_test, baseline_tree.predict(X_test))


          # Capture recall scores for test and train

          baseline_tree_recall_score_train_cv = cross_val_score(estimator=baseline_tree, X=X_train,  y=y_train,
                                              cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


          baseline_tree_recall_score_train = recall_score(y_train, baseline_tree.predict(X_train))
          baseline_tree_recall_score_test = recall_score(y_test, baseline_tree.predict(X_test))


          # Capture f1 scores for test and train

          baseline_tree_f1_score_train = f1_score(y_train, baseline_tree.predict(X_train))
          baseline_tree_f1_score_test = f1_score(y_test, baseline_tree.predict(X_test))


          # Capture precision scores for test and train
```

```python
baseline_tree_precision_score_train = precision_score(y_train, baseline_tree.predict(X_train))
baseline_tree_precision_score_test = precision_score(y_test, baseline_tree.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {baseline_tree_roc_score_train :.2%}")
print(f" Test ROC Score: {baseline_tree_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {baseline_tree_recall_score_train :.2%}")
print(f" Test Recall score: {baseline_tree_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {baseline_tree_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {baseline_tree_f1_score_train :.2%}")
print(f" Test F1 score: {baseline_tree_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {baseline_tree_precision_score_train :.2%}")
print(f" Test Precision score: {baseline_tree_precision_score_test :.2%}")
print(divider, '\n')
```

```
Performance Comparison

------------------------------------------------------------
 Train ROC Score: 100.00%
 Test ROC Score: 78.29%
------------------------------------------------------------
 Train Recall score: 100.00%
 Test Recall score: 75.20%
 Mean Cross Validated Recall Score: 74.04%
------------------------------------------------------------
 Train F1 score: 100.00%
 Test F1 score: 53.56%
------------------------------------------------------------
 Train Precision score: 100.00%
 Test Precision score: 41.59%
------------------------------------------------------------
```

In [354...
```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))
```

```
plot_confusion_matrix(baseline_tree, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Decision Tree Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_tree, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Decision Tree Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax set vticklabels(labels).
```


Baseline Decision Tree Confusion Matrix - Train

Baseline Decision Tree Confusion Matrix - Test
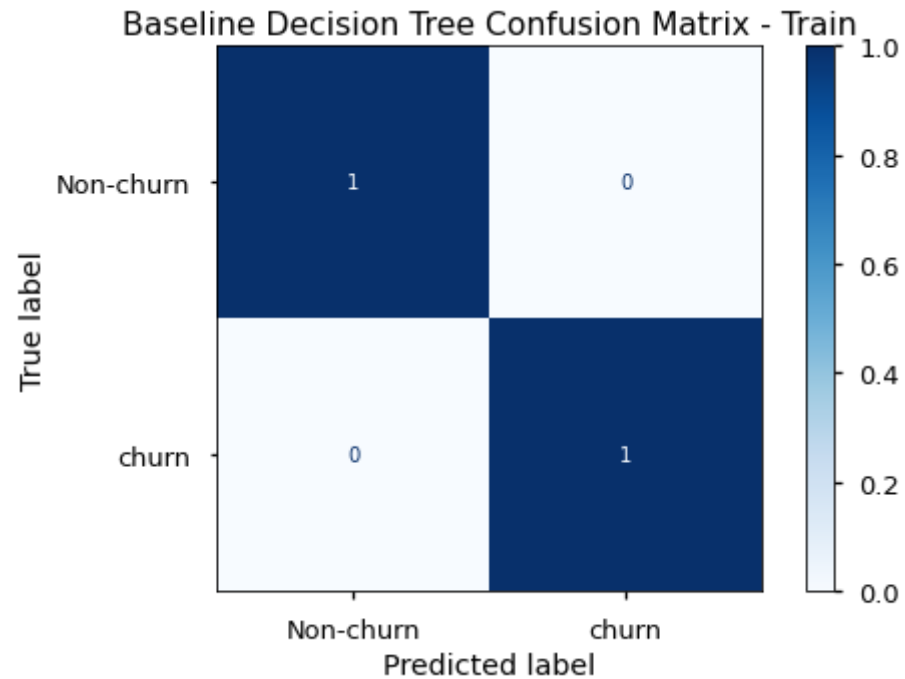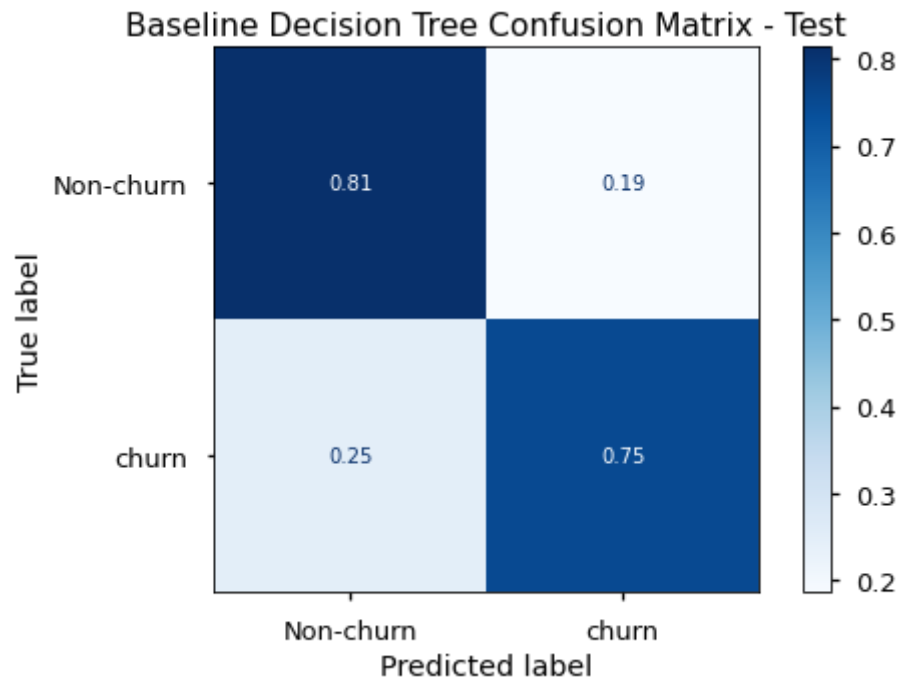
Holy over fitting batman. As you can see with our training data being at 100% there is some clear over fitting happening within our baseline Decision Tree model. This should get better with our tuning. It is worth noting however, that this is the highest F1 score we have seen thus far. Hopefully we can close the gap between our train and test sets since we are seeing a better F1 and a comparable recall score out of the gate.

Tuned Decision Tree Model

```python
# Let's tune this model!

parameters = {
    'estimator__criterion': ['gini', 'entropy'],
    'estimator__max_depth': [3, 5, 10, 15, 20],
    'estimator__max_features': [None, 15,  5],
    'estimator__min_samples_split': [2, 5, 7, 10],
    'estimator__min_samples_leaf': [1, 2, 5]
            }


# Grid with our baseline tree as our estimator

best_tree = GridSearchCV(estimator=baseline_tree,
```

```
                              param_grid=parameters,
                              scoring='recall',
                              cv=5,
                              n_jobs=-1
                          )
```

In [356… # Train the pipeline based on our most appropriate parameters

```
best_tree.fit(X_train, y_train);
best_tree.best_params_
```

Out[356… {'estimator__criterion': 'gini',
           'estimator__max_depth': 20,
           'estimator__max_features': None,
           'estimator__min_samples_leaf': 5,
           'estimator__min_samples_split': 2}

In [357… # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

```
# Capture roc_auc for test, and train

best_tree_roc_score_train = roc_auc_score(y_train, best_tree.predict(X_train))
best_tree_roc_score_test = roc_auc_score(y_test, best_tree.predict(X_test))


# Capture recall scores for test and train

best_tree_recall_score_train_cv = cross_val_score(estimator=best_tree, X=X_train,  y=y_train,
                                    cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


best_tree_recall_score_train = recall_score(y_train, best_tree.predict(X_train))
best_tree_recall_score_test = recall_score(y_test, best_tree.predict(X_test))


# Capture f1 scores for test and train

best_tree_f1_score_train = f1_score(y_train, best_tree.predict(X_train))
best_tree_f1_score_test = f1_score(y_test, best_tree.predict(X_test))


# Capture precision scores for test and train

best_tree_precision_score_train = precision_score(y_train, best_tree.predict(X_train))
best_tree_precision_score_test = precision_score(y_test, best_tree.predict(X_test))
```

```python
print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {best_tree_roc_score_train :.2%}")
print(f" Test ROC Score: {best_tree_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {best_tree_recall_score_train :.2%}")
print(f" Test Recall score: {best_tree_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {best_tree_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {best_tree_f1_score_train :.2%}")
print(f" Test F1 score: {best_tree_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {best_tree_precision_score_train :.2%}")
print(f" Test Precision score: {best_tree_precision_score_test :.2%}")
print(divider, '\n')
```

```
 Performance Comparison

 ------------------------------------------------------------
 Train ROC Score: 93.94%
 Test ROC Score: 80.03%
 ------------------------------------------------------------
 Train Recall score: 92.46%
 Test Recall score: 76.00%
 Mean Cross Validated Recall Score: 73.99%
 ------------------------------------------------------------
 Train F1 score: 84.12%
 Test F1 score: 57.06%
 ------------------------------------------------------------
 Train Precision score: 77.16%
 Test Precision score: 45.67%
 ------------------------------------------------------------
```

In [358…
```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_tree, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Decision Tree Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```

```
# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_tree, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Decision Tree Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```


Best Decision Tree Confusion Matrix - Train

Best Decision Tree Confusion Matrix - Test
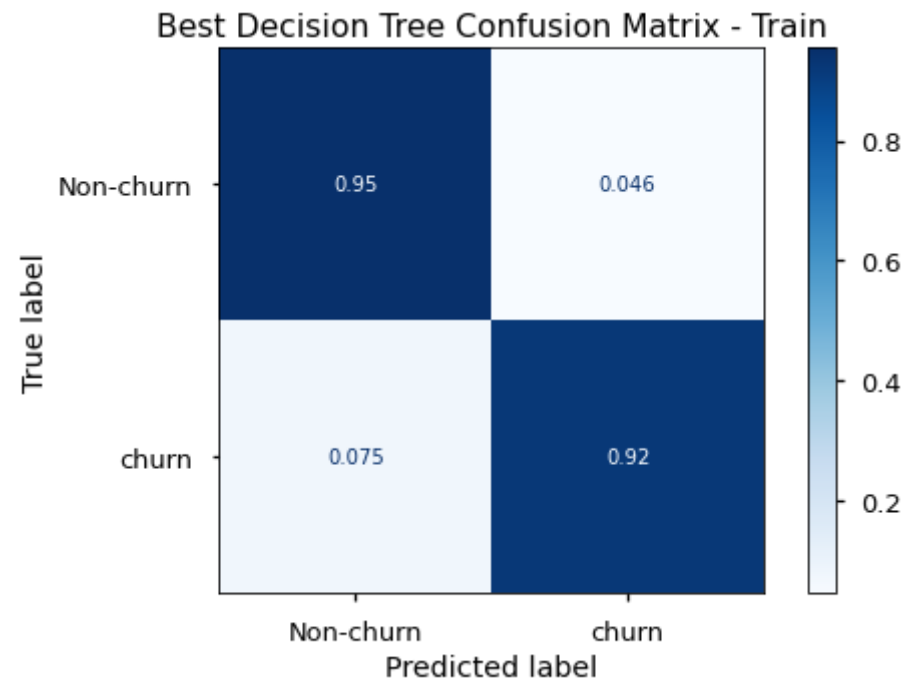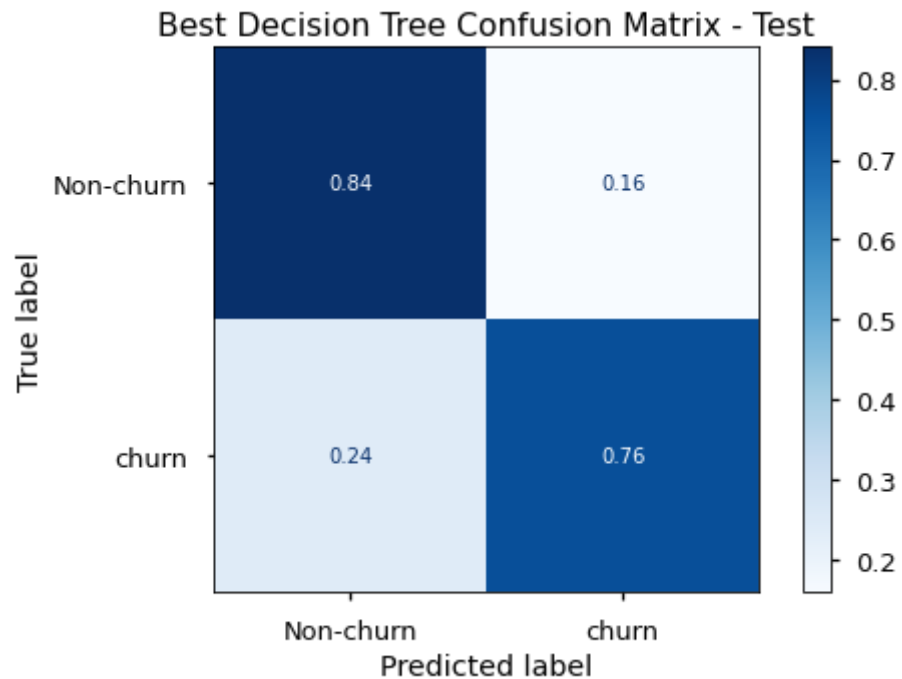
Performance is much better than our baseline tree, and we are already seeing improvement over our baseline model. Some overfitting happening, but not a ton. Going to move forward with additional models, but heading in the right direction.

Random Forest

Baseline Forest

```python
# Moving along to our next model

baseline_RF = ImPipeline(steps=[('sm', SMOTE(random_state=42)),
                                ('scale', StandardScaler()),
                                ('estimator', RandomForestClassifier(random_state=42))])


# Train model

baseline_RF.fit(X_train, y_train);
```

```python
# Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

# Capture roc_auc for test, and train

baseline_RF_roc_score_train = roc_auc_score(y_train, baseline_RF.predict(X_train))
baseline_RF_roc_score_test = roc_auc_score(y_test, baseline_RF.predict(X_test))


# Capture recall scores for test and train

baseline_RF_recall_score_train_cv = cross_val_score(estimator=baseline_RF, X=X_train,  y=y_train,
                                    cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


baseline_RF_recall_score_train = recall_score(y_train, baseline_RF.predict(X_train))
baseline_RF_recall_score_test = recall_score(y_test, baseline_RF.predict(X_test))


# Capture f1 scores for test and train

baseline_RF_f1_score_train = f1_score(y_train, baseline_RF.predict(X_train))
baseline_RF_f1_score_test = f1_score(y_test, baseline_RF.predict(X_test))


# Capture precision scores for test and train

baseline_RF_precision_score_train = precision_score(y_train, baseline_RF.predict(X_train))
baseline_RF_precision_score_test = precision_score(y_test, baseline_RF.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {baseline_RF_roc_score_train :.2%}")
print(f" Test ROC Score: {baseline_RF_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {baseline_RF_recall_score_train :.2%}")
print(f" Test Recall score: {baseline_RF_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {baseline_RF_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {baseline_RF_f1_score_train :.2%}")
print(f" Test F1 score: {baseline_RF_f1_score_test :.2%}")
print(divider)
```

```
print(f" Train Precision score: {baseline_RF_precision_score_train :.2%}")
print(f" Test Precision score: {baseline_RF_precision_score_test :.2%}")
```

Performance Comparison

```
---------------------------------------------------------------
 Train ROC Score: 100.00%
 Test ROC Score: 85.44%
---------------------------------------------------------------
 Train Recall score: 100.00%
 Test Recall score: 76.80%
 Mean Cross Validated Recall Score: 70.40%
---------------------------------------------------------------
 Train F1 score: 100.00%
 Test F1 score: 73.00%
---------------------------------------------------------------
 Train Precision score: 100.00%
 Test Precision score: 69.57%
---------------------------------------------------------------
```

In [361...

```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_RF, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Random Forest Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_RF, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline Random Forest Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```

Baseline Random Forest Confusion Matrix - Train

	Non-churn	churn
Non-churn	1	0
churn	0	1

True label / Predicted label

Baseline Random Forest Confusion Matrix - Test

	Non-churn	churn
Non-churn	0.94	0.059
churn	0.23	0.77

True label / Predicted label
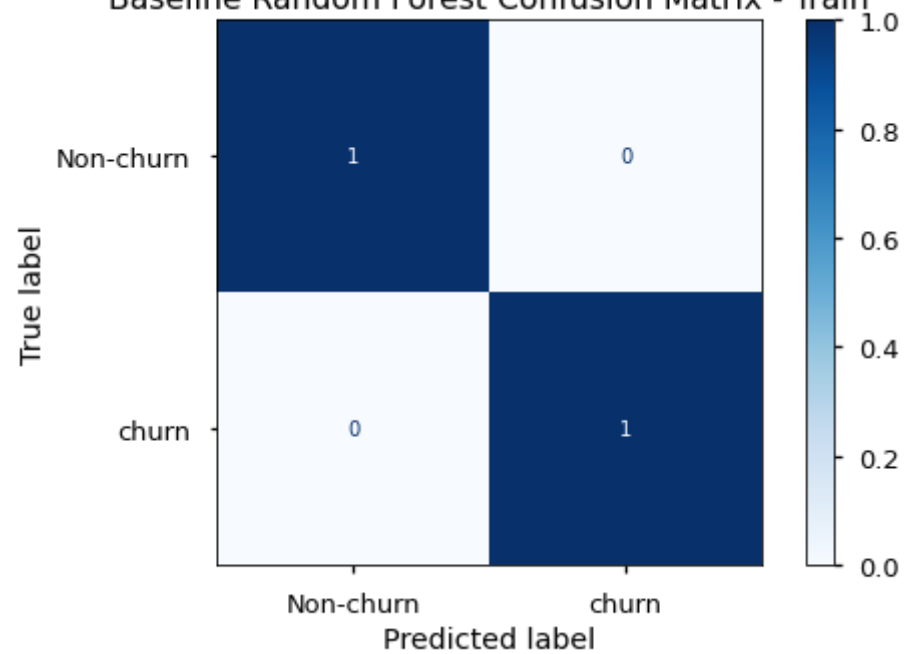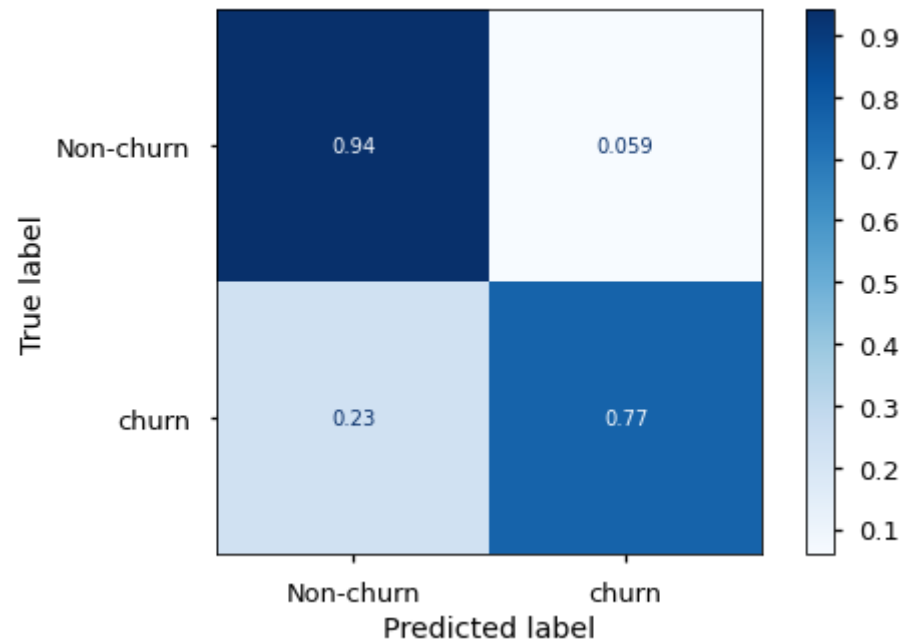
Again, some serious over fitting on our training data however we do see some life now in our F1 score. Our recall is also holding strong in a range that we have seen in our previous models.

Tuned Forest

```python
# Parameters for our gridsearch, model optimization

parameters = {
    #'estimator__n_estimators': [100, 150, 200],
    #'estimator__criterion': ['entropy', 'gini'],
    'estimator__max_depth': [2, 5],
    #'estimator__max_features': [2, 5, 10],
    'estimator__min_samples_split': [10, 20, 50],
    'estimator__min_samples_leaf': [1, 2, 4]
        }

best_RF = GridSearchCV(estimator=baseline_RF,
                       param_grid=parameters,
                       scoring='recall',
                       cv=5,
                       n_jobs=-1
                    )
```

In [362…

```python
# Train the pipeline based on our most appropriate parameters

best_RF.fit(X_train, y_train)
best_RF.best_params_
```

In [363…

```
{'estimator__max_depth': 5,
 'estimator__min_samples_leaf': 1,
 'estimator__min_samples_split': 50}
```

Out[363…

```python
# Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

# Capture roc_auc for test, and train

best_RF_roc_score_train = roc_auc_score(y_train, best_RF.predict(X_train))
best_RF_roc_score_test = roc_auc_score(y_test, best_RF.predict(X_test))


# Capture recall scores for test and train

best_RF_recall_score_train_cv = cross_val_score(estimator=best_RF, X=X_train,  y=y_train,
```

In [364…

```python
                                        cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


best_RF_recall_score_train = recall_score(y_train, best_RF.predict(X_train))
best_RF_recall_score_test = recall_score(y_test, best_RF.predict(X_test))


# Capture f1 scores for test and train

best_RF_f1_score_train = f1_score(y_train, best_RF.predict(X_train))
best_RF_f1_score_test = f1_score(y_test, best_RF.predict(X_test))


# Capture precision scores for test and train

best_RF_precision_score_train = precision_score(y_train, best_RF.predict(X_train))
best_RF_precision_score_test = precision_score(y_test, best_RF.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {best_RF_roc_score_train :.2%}")
print(f" Test ROC Score: {best_RF_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {best_RF_recall_score_train :.2%}")
print(f" Test Recall score: {best_RF_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {best_RF_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {best_RF_f1_score_train :.2%}")
print(f" Test F1 score: {best_RF_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {best_RF_precision_score_train :.2%}")
print(f" Test Precision score: {best_RF_precision_score_test :.2%}")
print(divider, '\n')
```

```
 Performance Comparison

 ------------------------------------------------------------
 Train ROC Score: 85.81%
 Test ROC Score: 83.51%
 ------------------------------------------------------------
 Train Recall score: 80.73%
 Test Recall score: 77.60%
 Mean Cross Validated Recall Score: 73.72%
 ------------------------------------------------------------
```

```
Train F1 score: 68.65%
Test F1 score: 65.32%
-------------------------------------------------------------
Train Precision score: 59.71%
Test Precision score: 56.40%
-------------------------------------------------------------
```

```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_RF, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Random Forest Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_RF, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best Random Forest Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```
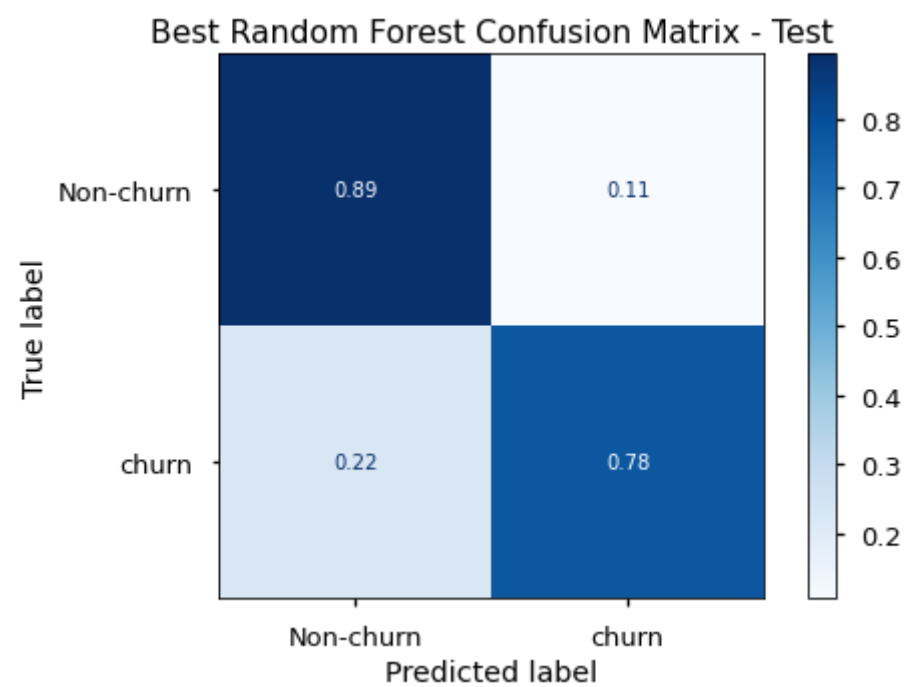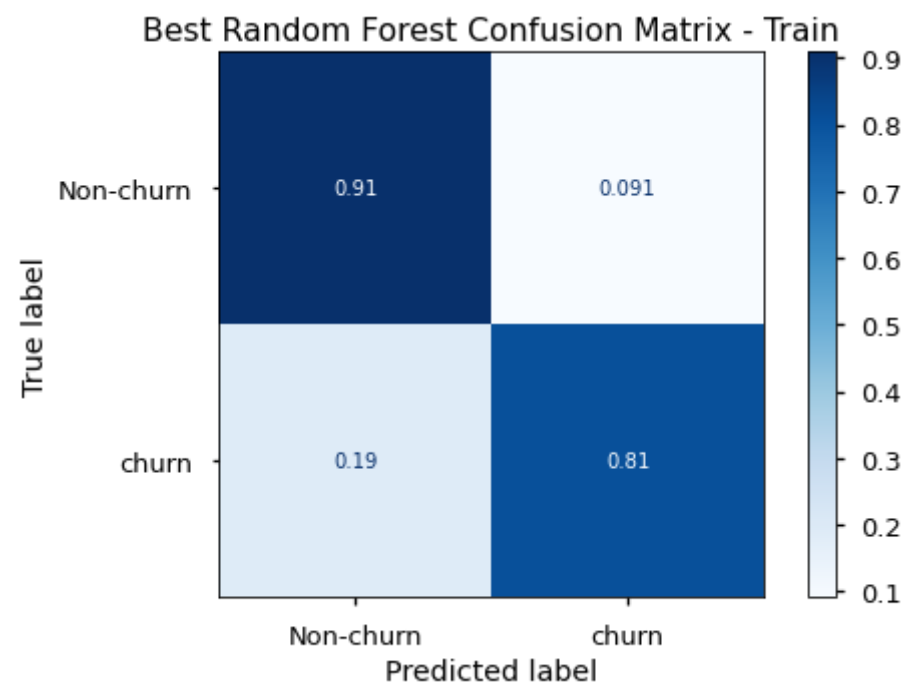
Best Random Forest Confusion Matrix - Train

Best Random Forest Confusion Matrix - Test

This model is looking much better now that we tuned out some of the noise from our baseline model. This model is very similar to our tuned tree model. The test scores on the best random forest model is out performing the best decision tree model. However, our F1 scores are comparable.

XG Boost Model

Baseline XGBoost

```
In [366…   baseline_xgb = ImPipeline(steps=[('sm', SMOTE(random_state=42)),
                                    ('scale', StandardScaler()),
                                    ('estimator', xgb.XGBClassifier(objective="binary:logistic", random_state=42))

           # Train model
           baseline_xgb.fit(X_train, y_train);
```

```
In [367…   # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

           # Capture roc_auc for test, and train

           baseline_xgb_roc_score_train = roc_auc_score(y_train, baseline_xgb.predict(X_train))
           baseline_xgb_roc_score_test = roc_auc_score(y_test, baseline_xgb.predict(X_test))


           # Capture recall scores for test and train

           baseline_xgb_recall_score_train_cv = cross_val_score(estimator=baseline_xgb, X=X_train,  y=y_train,
                                              cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


           baseline_xgb_recall_score_train = recall_score(y_train, baseline_xgb.predict(X_train))
           baseline_xgb_recall_score_test = recall_score(y_test, baseline_xgb.predict(X_test))


           # Capture f1 scores for test and train

           baseline_xgb_f1_score_train = f1_score(y_train, baseline_xgb.predict(X_train))
           baseline_xgb_f1_score_test = f1_score(y_test, baseline_xgb.predict(X_test))


           # Capture precision scores for test and train
```

```python
baseline_xgb_precision_score_train = precision_score(y_train, baseline_xgb.predict(X_train))
baseline_xgb_precision_score_test = precision_score(y_test, baseline_xgb.predict(X_test))

print('\n', "Performance Comparison", '\n')
print(divider)
print(f" Train ROC Score: {baseline_xgb_roc_score_train :.2%}")
print(f" Test ROC Score: {baseline_xgb_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {baseline_xgb_recall_score_train :.2%}")
print(f" Test Recall score: {baseline_xgb_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {baseline_xgb_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {baseline_xgb_f1_score_train :.2%}")
print(f" Test F1 score: {baseline_xgb_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {baseline_xgb_precision_score_train :.2%}")
print(f" Test Precision score: {baseline_xgb_precision_score_test :.2%}")
print(divider, '\n')
```

```
Performance Comparison

------------------------------------------------------------
 Train ROC Score: 100.00%
 Test ROC Score: 86.71%
------------------------------------------------------------
 Train Recall score: 100.00%
 Test Recall score: 79.20%
 Mean Cross Validated Recall Score: 72.92%
------------------------------------------------------------
 Train F1 score: 100.00%
 Test F1 score: 74.72%
------------------------------------------------------------
 Train Precision score: 100.00%
 Test Precision score: 70.71%
------------------------------------------------------------
```

In [368...
```python
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_xgb, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
```

```
ax.set_title("Baseline XGBoost Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(baseline_xgb, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Baseline XGBoost Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax set yticklabels(labels).
```
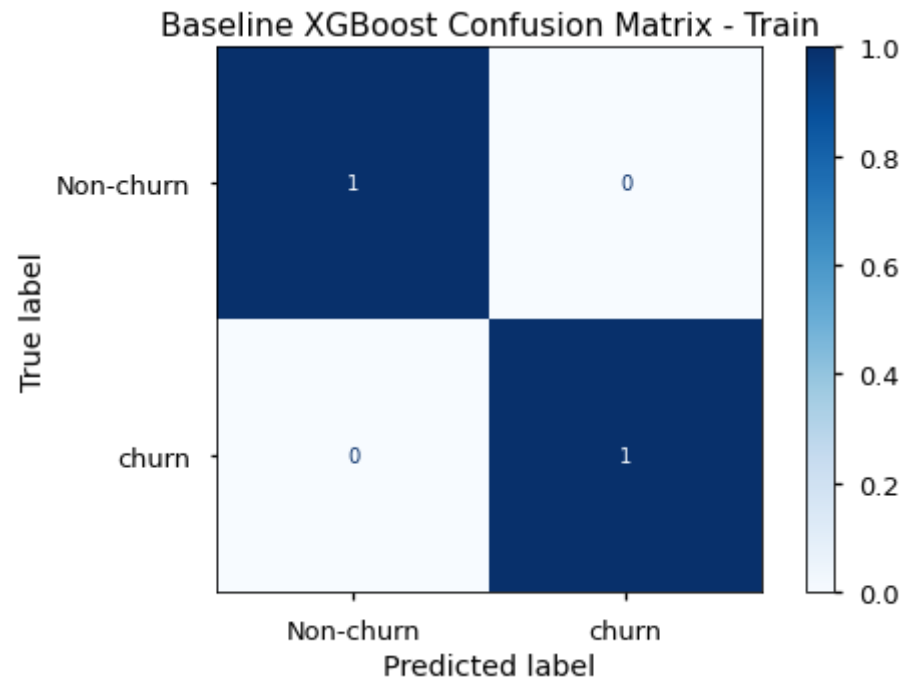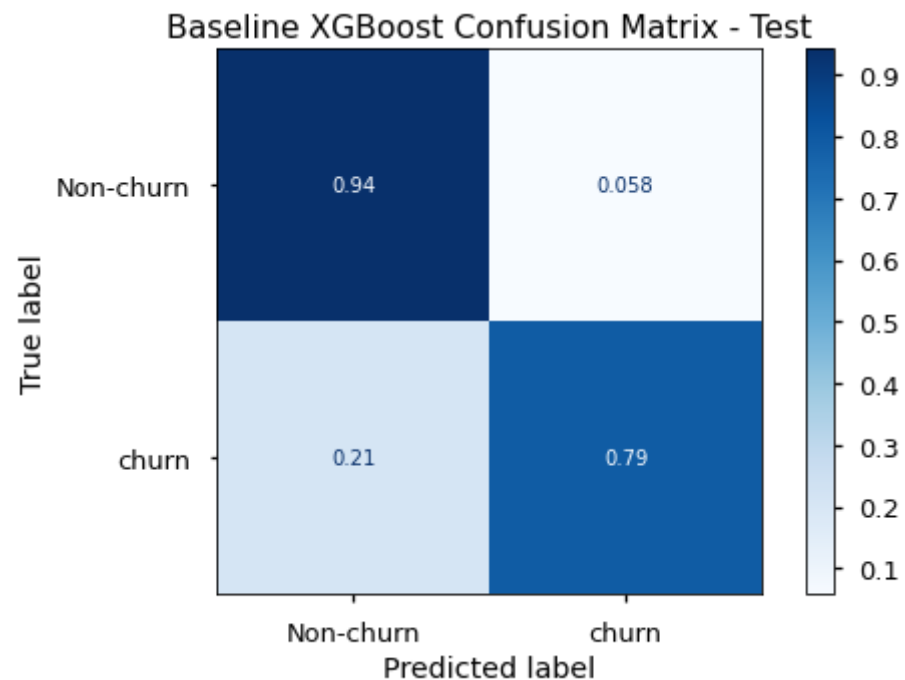

Baseline XGBoost Confusion Matrix - Train

Baseline XGBoost Confusion Matrix - Test

Clear over fitting again, but positive measurements on our F1 test score. Going to tune our parameters again and see if we can create a nice fitting model to predict our customer churn!

Tuned XGBoost

```
In [369…    # Tuning our XGB model.

            parameters = {
                "estimator__n_estimators": [50, 75],
                "estimator__learning_rate": [0.05, 0.1, 0.2],
                "estimator__max_depth": [4, 5, 6],
                'estimator__gamma': [0.5, 1],
                'estimator__min_child_weight': [3, 4, 5],
                'estimator__subsample': [0.5, 0.75],
                'estimator__colsample_bytree':[0.5, 0.75]
                    }

            best_xgb = GridSearchCV(estimator=baseline_xgb,
                                    param_grid=parameters,
                                    scoring='recall',
                                    cv=5,
```

```
                         n_jobs=-1
                                                                                      `

In [370…  # Train the pipeline based on our most appropriate parameters

          best_xgb.fit(X_train, y_train)
          best_xgb.best_params_

Out[370…  {'estimator__colsample_bytree': 0.75,
           'estimator__gamma': 1,
           'estimator__learning_rate': 0.1,
           'estimator__max_depth': 4,
           'estimator__min_child_weight': 4,
           'estimator__n_estimators': 50,
           'estimator__subsample': 0.5}

In [371…  # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

          # Capture roc_auc for test, and train

          best_xgb_roc_score_train = roc_auc_score(y_train, best_xgb.predict(X_train))
          best_xgb_roc_score_test = roc_auc_score(y_test, best_xgb.predict(X_test))


          # Capture recall scores for test and train

          best_xgb_recall_score_train_cv = cross_val_score(estimator=best_xgb, X=X_train,   y=y_train,
                                          cv=StratifiedKFold(shuffle=True), scoring='recall').mean()


          best_xgb_recall_score_train = recall_score(y_train, best_xgb.predict(X_train))
          best_xgb_recall_score_test = recall_score(y_test, best_xgb.predict(X_test))


          # Capture f1 scores for test and train

          best_xgb_f1_score_train = f1_score(y_train, best_xgb.predict(X_train))
          best_xgb_f1_score_test = f1_score(y_test, best_xgb.predict(X_test))


          # Capture precision scores for test and train

          best_xgb_precision_score_train = precision_score(y_train, best_xgb.predict(X_train))
          best_xgb_precision_score_test = precision_score(y_test, best_xgb.predict(X_test))

          print('\n', "Performance Comparison", '\n')
```

```
print(divider)
print(f" Train Roc_Auc Score: {best_xgb_roc_score_train :.2%}")
print(f" Test Roc_Auc Score: {best_xgb_roc_score_test :.2%}")
print(divider)

print(f" Train Recall score: {best_xgb_recall_score_train :.2%}")
print(f" Test Recall score: {best_xgb_recall_score_test :.2%}")
print(f" Mean Cross Validated Recall Score: {best_xgb_recall_score_train_cv :.2%}")
print(divider)

print(f" Train F1 score: {best_xgb_f1_score_train :.2%}")
print(f" Test F1 score: {best_xgb_f1_score_test :.2%}")
print(divider)

print(f" Train Precision score: {best_xgb_precision_score_train :.2%}")
print(f" Test Precision score: {best_xgb_precision_score_test :.2%}")
print(divider, '\n')
```

Performance Comparison

```
-------------------------------------------------------------
 Train Roc_Auc Score: 89.40%
 Test Roc_Auc Score: 86.26%
-------------------------------------------------------------
 Train Recall score: 86.31%
 Test Recall score: 82.40%
 Mean Cross Validated Recall Score: 80.14%
-------------------------------------------------------------
 Train F1 score: 74.64%
 Test F1 score: 69.13%
-------------------------------------------------------------
 Train Precision score: 65.74%
 Test Precision score: 59.54%
-------------------------------------------------------------
```

In [372…
```
# plotting confusion matrix for our tuned logistic regression - Train

labels = ['Non-churn', 'churn']

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_xgb, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best XGBoost Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test
```
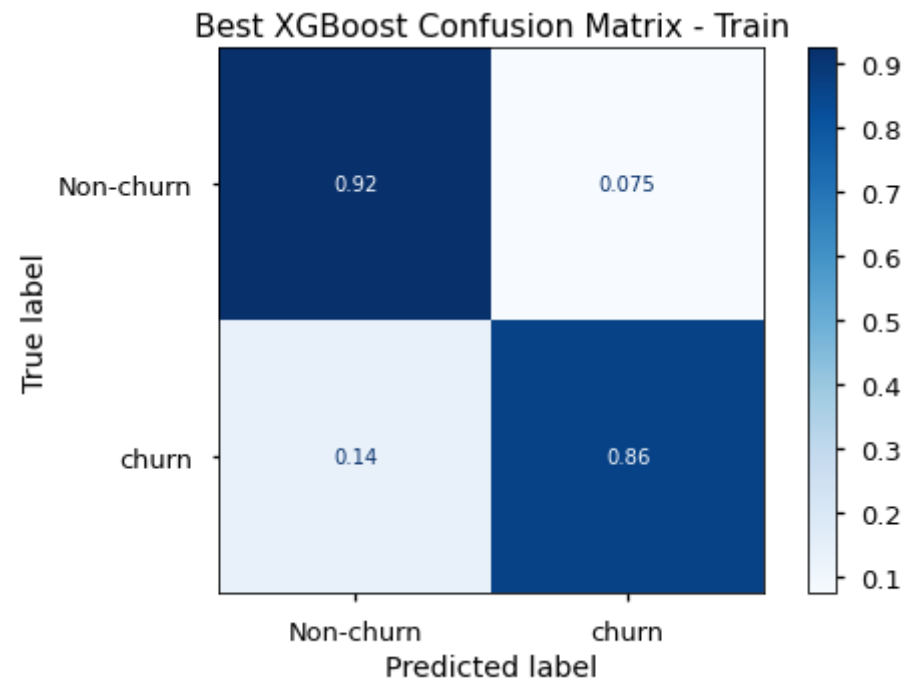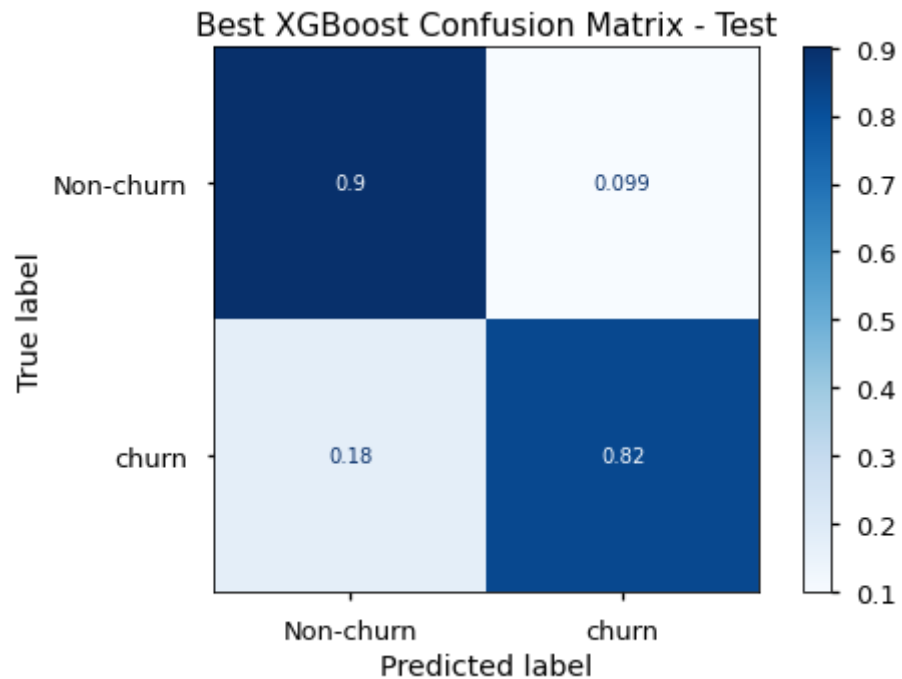
```
fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_xgb, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best XGBoost Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```


Best XGBoost Confusion Matrix - Train

Best XGBoost Confusion Matrix - Test

This is by far our best model. Overfitting seems to be resolved, and both positive recall and f1 scores.

Model Evalution

Comparison between Models

Although the best performing model based on recall only was our tuned logistic regression, and our ridge models, the best performing model from both a Recall and a F1 perspective was our tuned XGBoost model. This model was able to predict our positive cases at a rate of 82%. We will go back and visualize some of these metrics below.

Final Model

So in Summary, our best performing model according to the best recall, and F1 score is our Tuned XGBoost model. As mentioned above, we are able to successfully predict our groups for churn approximately 82% of the time, without sacrificing groups that we are not able to successfully predict, and they churn (false negatives).

Let's first start by taking a look back at our summary metrics and performance of our best performing model.

```
In [373...    # Scoring print out adapted from others -- Eva Mizer, and Aysu Erdemir.

              print('\n', "Performance Comparison", '\n')
              print(divider)
              print(f" Train ROC Score: {best_xgb_roc_score_train :.2%}")
              print(f" Test ROC Score: {best_xgb_roc_score_test :.2%}")
              print(divider)

              print(f" Train Recall score: {best_xgb_recall_score_train :.2%}")
              print(f" Test Recall score: {best_xgb_recall_score_test :.2%}")
              print(f" Mean Cross Validated Recall Score: {best_xgb_recall_score_train_cv :.2%}")
              print(divider)

              print(f" Train F1 score: {best_xgb_f1_score_train :.2%}")
              print(f" Test F1 score: {best_xgb_f1_score_test :.2%}")
              print(divider)

              print(f" Train Precision score: {best_xgb_precision_score_train :.2%}")
              print(f" Test Precision score: {best_xgb_precision_score_test :.2%}")
              print(divider, '\n')
```

```
 Performance Comparison

 ------------------------------------------------------------
 Train ROC Score: 89.40%
 Test ROC Score: 86.26%
 ------------------------------------------------------------
 Train Recall score: 86.31%
 Test Recall score: 82.40%
 Mean Cross Validated Recall Score: 80.14%
 ------------------------------------------------------------
 Train F1 score: 74.64%
 Test F1 score: 69.13%
 ------------------------------------------------------------
 Train Precision score: 65.74%
 Test Precision score: 59.54%
 ------------------------------------------------------------
```

```
In [374...    # plotting confusion matrix for our tuned logistic regression - Train

              labels = ['Non-churn', 'churn']

              fig, ax = plt.subplots(figsize=(8,5))

              plot_confusion_matrix(best_xgb, X_train, y_train, ax=ax,  cmap='Blues', normalize='true')
```

```python
ax.set_title("Best XGBoost Confusion Matrix - Train")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);

# plotting confusion matrix for our tuned logistic regression - Test

fig, ax = plt.subplots(figsize=(8,5))

plot_confusion_matrix(best_xgb, X_test, y_test, ax=ax,  cmap='Blues', normalize='true')
ax.set_title("Best XGBoost Confusion Matrix - Test")
ax.set_xticklabels(labels)
ax.set_yticklabels(labels);
```
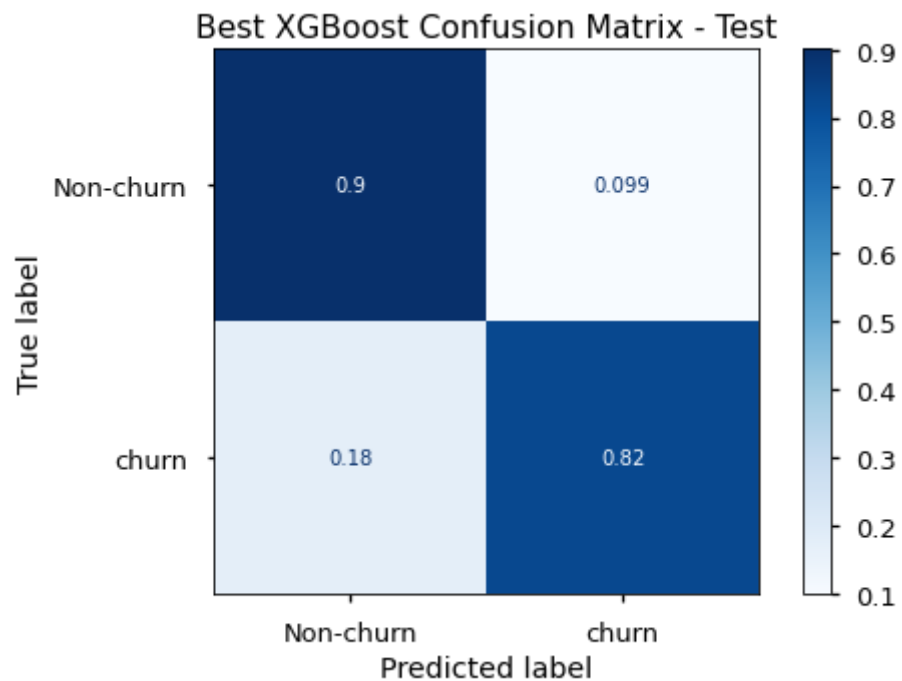

Best XGBoost Confusion Matrix - Train

Best XGBoost Confusion Matrix - Test
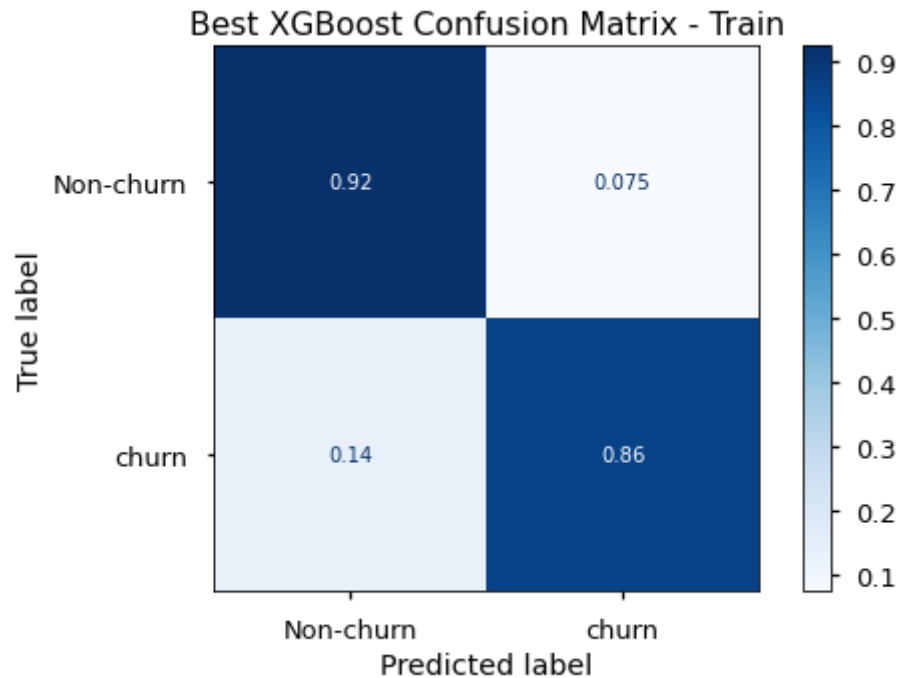
In [375...

```python
# ROC_AUC curve for our best performing model

fig, ax2 = plt.subplots(figsize=(8,5))
plot_roc_curve(best_xgb, X_train, y_train, ax=ax2, name ='Train ROC curve', color=pal[1])
plot_roc_curve(best_xgb, X_test, y_test, ax=ax2, name ='Test ROC curve', color=pal[4])
ax2.plot([0, 1], [0, 1], color='lightgray', lw=1, linestyle='-.')
ax2.set_xlabel('False Positive Rate')
ax2.set_ylabel('True Positive Rate')
ax2.set_title('Receiver operating characteristic (ROC) Curve')
plt.show();
```

Receiver operating characteristic (ROC) Curve

Train ROC curve (AUC = 0.94)
Test ROC curve (AUC = 0.92)

False Positive Rate / True Positive Rate
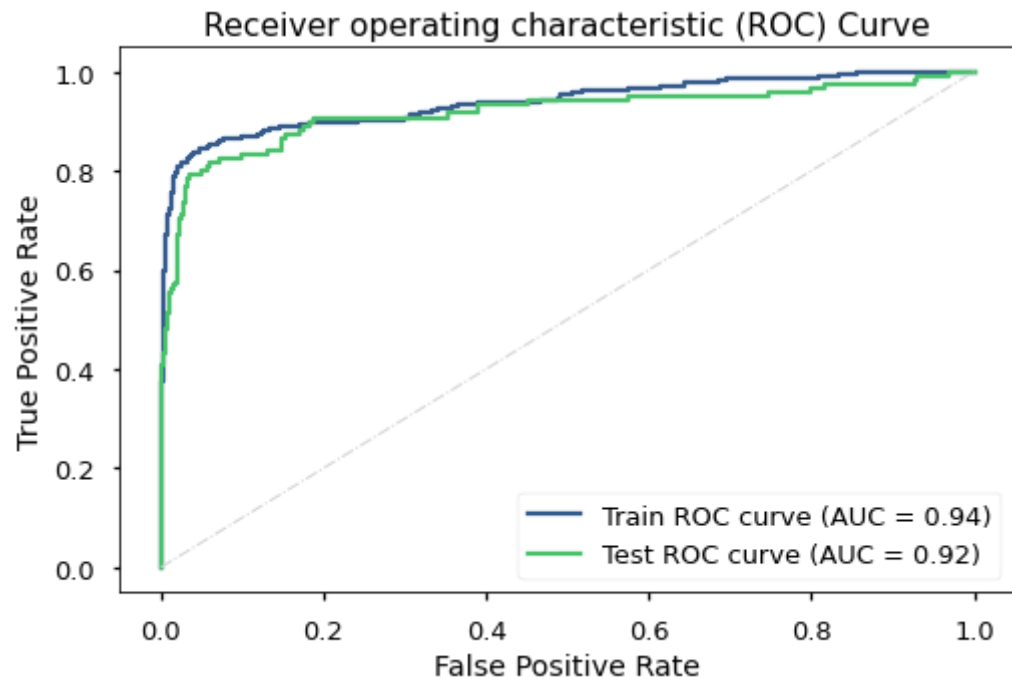
In [376…

```python
# Precision Recall curve for our best performing model

y_train_score = best_xgb.predict(X_train)
y_test_score = best_xgb.predict(X_test)


# Calculate precision and recall

precision, recall, thresholds = precision_recall_curve(y_test, y_test_score)
precision_train, recall_train, thresholds_train = precision_recall_curve(y_train, y_train_score)


# Create precision recall curve

fig, ax = plt.subplots(figsize=(8,5))
ax.plot(recall, precision, color=pal[1], label='Test Precision/ Recall curve')
ax.plot(recall_train, precision_train, color=pal[4], label='Train Precision/ Recall curve')


# Add axis labels to plot

ax.set_title('Precision-Recall Curve')
ax.set_ylabel('Precision')
```
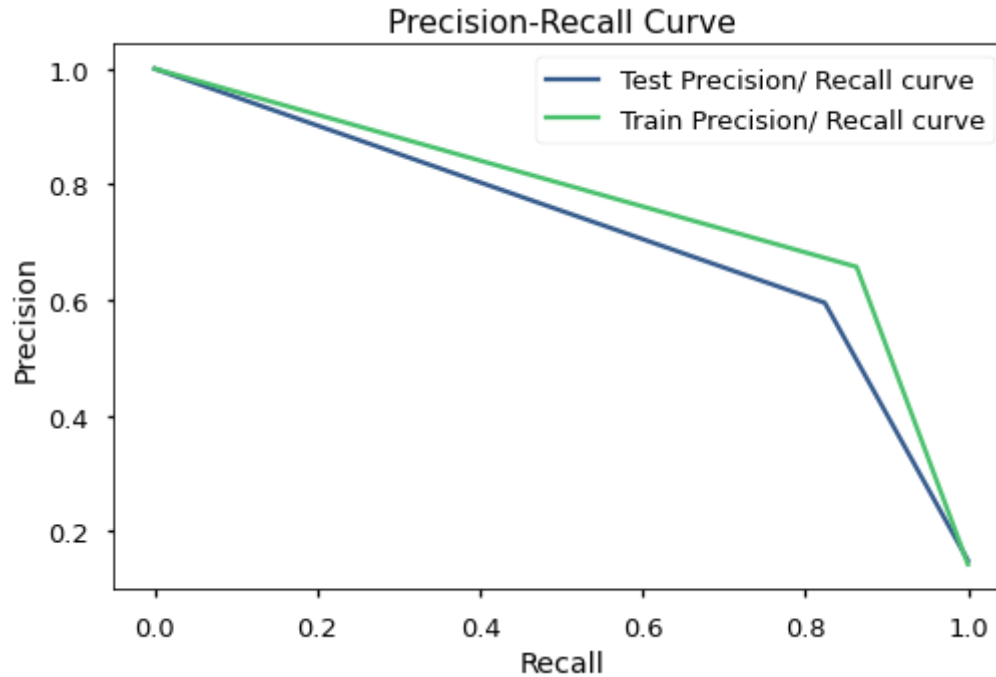
```
ax.set_xlabel('Recall')
ax.legend()

plt.show()
```


Precision-Recall Curve

In [377... # Taking a look at our feature importances

best_xgb.estimator.steps[2][1].feature_importances_

Out[377... array([0.0349853 , 0.14855045, 0.23684838, 0.09348225, 0.02688019,
              0.05152148, 0.03433942, 0.04045242, 0.02911783, 0.04626551,
              0.0796058 , 0.17795102], dtype=float32)

In [378... # Columns on our dataframe

df_X_train_copy.columns

Out[378... Index(['account_length', 'international_plan', 'voice_mail_plan',
              'total_day_minutes', 'total_day_calls', 'total_eve_minutes',
              'total_eve_calls', 'total_night_minutes', 'total_night_calls',
              'total_intl_minutes', 'total_intl_calls', 'customer_service_calls'],
   dtype='object')
```

```python
# Visualize our Feature importances

feature_imp_xgb = pd.DataFrame(pd.Series(best_xgb.estimator.steps[2][1].feature_importances_, index=df_X_train_
feature_imp_xgb = feature_imp_xgb.head(10)
feature_imp_xgb['index'] = feature_imp_xgb.index
feature_imp_xgb.head()

ax = sns.barplot(x=feature_imp_xgb[0], y=feature_imp_xgb['index'], data=feature_imp_xgb, color=pal[0])
ax.set_xlabel('Importance')
ax.set_ylabel('Feature')
ax.set_title('Importance per Feature');
```

Importance per Feature
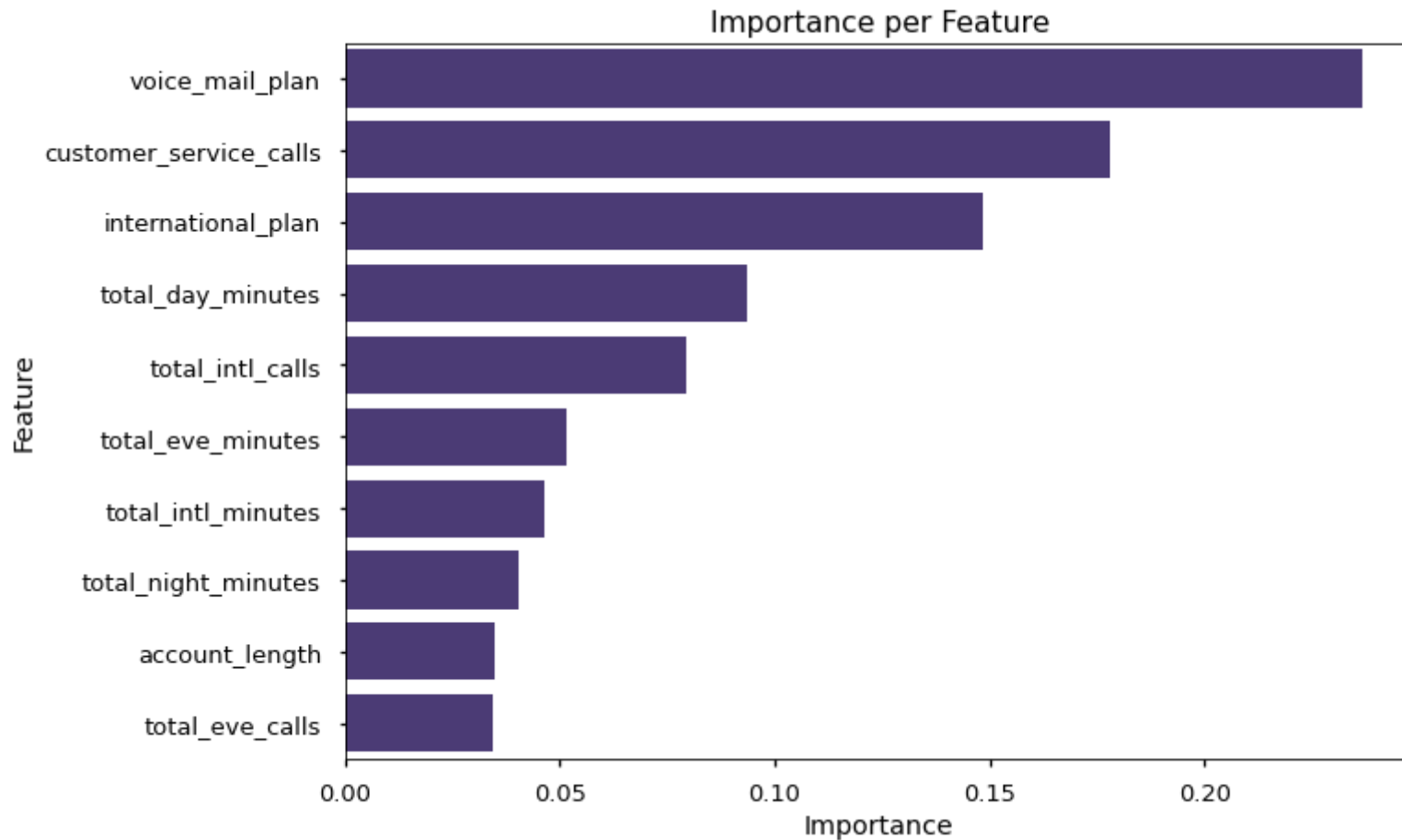
Recommendations & Summary

The top 3 features that lead to churn are customers that have the international plan, folks that engage with customer service frequently, and those with the voice mail plan. Those that are engaging with customer service already, most likely have some other questions or concern about the value that the features/ service are providing. These would be good indicators of risk, and information to understand what issues customers are experiencing.

Having customer service calls on this list, actually will make it easier to identify risk within the customer base. Thus making the outbound efforts to engage with customers with the international plan and the voice mail plan that aren't engaging frequently with customer service.

1. Once a customer reaches 3 customer service calls, flag the account as at risk. Once the account reaches 4 calls, the probability of canceling goes from 10% to 45%.

2. Customers that have the international plan and the voice mail plan should be surveyed to understand product performance and value. Although the voice mail plan performances better when paired with the international plan, it should be noted still that no international plan, and no voice mail plan performs the worst of the possible feature combinations (with those products).

3. Scale pricing to offer some relief for customers that use the plan more often. The pricing scale doesn't incentivize more usage of the product. Another option would be to look at unlimited usage based pricing, to give customers that use the plan more, the relief knowing that if they use the plan more, they won't necessarily be charged more for that usage.

In [ ]: