1 Predicting Telecommunications Customer Churn

According to [Profitwell (https://www.profitwell.com/customer-retention/industry-rates)](https://www.profitwell.com/customer-retention/industry-rates), the average churn rate within the telecommunications industry is 22% (churn rate referring to the rate customers close their accounts or end their business relationship). This project develops a machine learning classification algorithm to predict customer churn.

2 Business Understanding

The telecommunications company, SyriaTel, is faced with the problem of better predicting when its customers will soon churn. They need a solution that will predict whether a customer will ("soon") stop doing business with SyriaTel. This will be valuable to SyriaTel, so that they may better understand their churn rate and identify areas they may address to improve its churn rate.

Finding predictable patterns using a classification model will benefit SyriaTel's business practices to minimize customer churn.

To determine which classification model best predicts potential customer churn, **I will evaluate models' performance using the F1 score**. More on this metric to follow.

3 Data Understanding

The data source for this project comes from [SyriaTel's churn data (https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset)](https://www.kaggle.com/datasets/becksddf/churn-in-telecoms-dataset). This data is suitable for the project because it includes key performance indicators and data points from SyriaTel related to its customers and their accounts as well as whether the customer churned or not.

The data consists of 3,333 observations with 21 features and no missing values.

Explanation of Features

- `state` : the state the user lives in
- `account length` : the number of days the user has this account
- `area code` : the code of the area the user lives in
- `phone number` : the phone number of the user
- `international plan` : true if the user has the international plan, otherwise false
- `voice mail plan` : true if the user has the voice mail plan, otherwise false
- `number vmail messages` : the number of voice mail messages the user has sent
- `total day minutes` : total number of minutes the user has been in calls during the day
- `total day calls` : total number of calls the user has done during the day
- `total day charge` : total amount of money the user was charged by the Telecom company for calls during the day
- `total eve minutes` : total number of minutes the user has been in calls during the evening
- `total eve calls` : total number of calls the user has done during the evening
- `total eve charge` : total amount of money the user was charged by the Telecom company for calls during the evening
- `total night minutes` : total number of minutes the user has been in calls during the night
- `total night calls` : total number of calls the user has done during the night
- `total night charge` : total amount of money the user was charged by the Telecom company for calls during the night
- `total intl minutes` : total number of minutes the user has been in international calls
- `total intl calls` : total number of international calls the user has done
- `total intl charge` : total amount of money the user was charged by the Telecom company for international calls
- `customer service calls` : number of customer service calls the user has done
- `churn` : true if the user terminated the contract, otherwise false

Further descriptive statistics to follow.

4 Data Preparation

4.1 Importing Libraries and Data

In [166]:

```python
# Import needed base libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import machine learning libraries

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from sklearn.metrics import plot_confusion_matrix, f1_score, classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression


import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

executed in 143ms, finished 19:19:36 2022-05-24

In [2]:

```python
# Import dataset

df = pd.read_csv('telecom.csv')
```

executed in 54ms, finished 14:53:34 2022-05-24

In [3]:

```python
# Preview the data to ensure it loaded correctly

df.head()
```

executed in 48ms, finished 14:53:34 2022-05-24

Out[3]:

	state	account length	area code	phone number	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	...	total eve calls	total eve charge	total night minutes	total night calls	total night charge	tot ir minute
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99	16.78	244.7	91	11.01	10
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103	16.62	254.4	103	11.45	13
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110	10.30	162.6	104	7.32	12
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88	5.26	196.9	89	8.86	6
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122	12.61	186.9	121	8.41	10

5 rows × 21 columns

4.2 Data Import

In [4]:

```python
# Shape of the data

df.shape

# Indicates 3,333 observations with 21 features
```

executed in 12ms, finished 14:53:34 2022-05-24

Out[4]:

```
(3333, 21)
```

In [5]:

```
df.info()

# No missing values
```

executed in 35ms, finished 14:53:35 2022-05-24

```
<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
```

A couple of notes about our data thus far:

1. `phone number` is essentially a unique ID for each observation, so that column can be dropped
2. `state`, `international plan`, and `voice mail plan` have string values (`churn` is a boolean value). For the latter two that have yes/no, I convert to 1 and 0, respectively. For `state`, I use `LabelEncoder`.

For efficiency and reproducibility, I write a preprocessing function to handle this work.

4.3 Data Preprocessing

In [8]:

```
# Write a preprocessing data function

def preprocess_data(df):
    pre_df = df.copy()
    
    # Replace the spaces in the column names with underscores
    pre_df.columns = [s.replace(" ", "_") for s in pre_df.columns]
    
    # Convert string columns to integers
    pre_df["international_plan"] = pre_df["international_plan"].apply(lambda x: 0 if x=="no" else 1)
    pre_df["voice_mail_plan"] = pre_df["voice_mail_plan"].apply(lambda x: 0 if x=="no" else 1)
    pre_df = pre_df.drop(["phone_number"], axis=1)
    
    # Initialize labelencoder()
    le = LabelEncoder()
    le.fit(pre_df['state'])
    pre_df['state'] = le.transform(pre_df['state'])
    
    return pre_df, le
```

executed in 7ms, finished 14:53:35 2022-05-24

In [9]:

```
# Apply the preprocessing function to our data

pre_df, _ = preprocess_data(df)
pre_df.head(3)
```

executed in 52ms, finished 14:53:35 2022-05-24

Out[9]:

	state	account_length	area_code	international_plan	voice_mail_plan	number_vmail_messages	total_day_minutes	total_day_calls	total_da
0	16	128	415	0	1	25	265.1	110	
1	35	107	415	0	1	26	161.6	123	
2	31	137	415	0	0	0	243.4	114	

4.4 Exploratory Data Analysis

As noted above, this dataset has 3,333 observations with 21 different features. The only feature that will not be included in the analysis is each observation's phone number as this functions as a unique identifier and thus, does not add value to any predictive model.

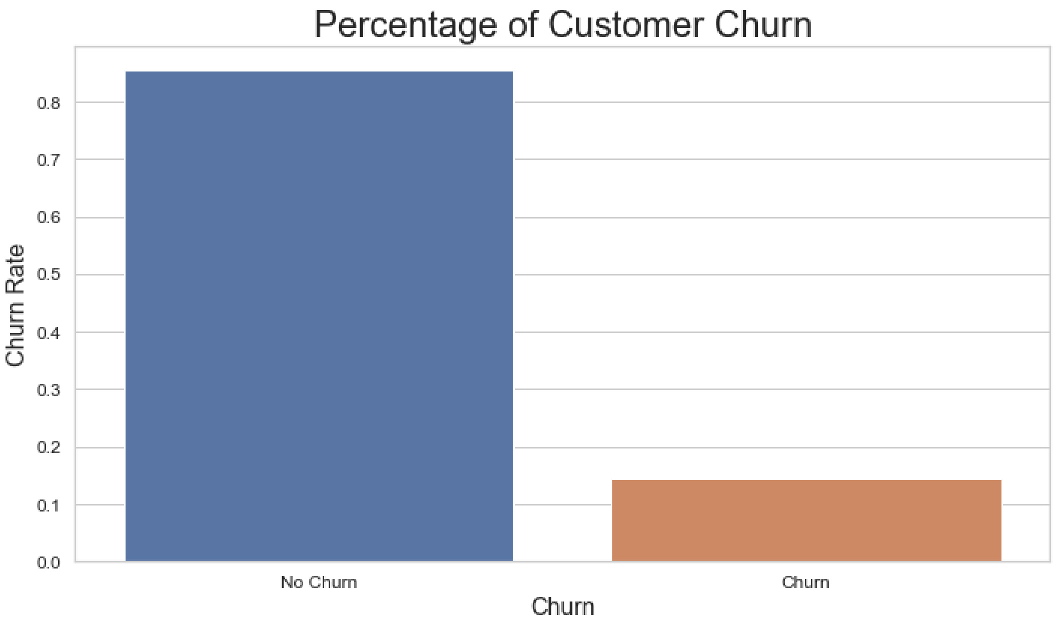
4.4.1 Data Visualizations

Distribution of Churn

In [323]:

```
churn_perc = pd.DataFrame(df.churn.value_counts()/len(df.churn))

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = [0, 1], y = 'churn', data = churn_perc, ax = ax)
plt.title('Percentage of Customer Churn', fontsize = 24)
ax.tick_params(axis = 'both', labelsize = 12)
plt.xlabel('Churn', fontsize = 16)
plt.ylabel('Churn Rate', fontsize = 16)
ax.set_xticklabels(['No Churn', 'Churn'])
plt.tight_layout()
```

executed in 184ms, finished 16:49:01 2022-05-27



In [300]:

```
pre_df["churn"].value_counts(normalize = True)
```

executed in 147ms, finished 16:15:05 2022-05-27

Out[300]:

```
False    0.855086
True     0.144914
Name: churn, dtype: float64
```

Here we can see of the 3,333 observations (or accounts), 86% have not churned whereas 14% have churned.

Further, this initial visualization indicates that we have an unbalanced dataset; a baseline model that always chooses the majority class (no churn) would have an accuracy of 85.5%. In the modeling to address the business problem, I address the imbalance by using class weights, which will promote the minority class in our analyses, and use the F1 score to measure effectiveness of the models.

The F1 score is the harmonic mean of two other metrics, precision and recall, and is suited well to evaluate imbalanced datasets.

Precision summarizes the fraction of examples assigned the positive class that belong to the positive class whereas the recall summarizes how well the positive class was predicted and is the same calculation as sensitivity. Both precision and recall values fall in the range [0,1], with 0 indicating no precision/recall and 1 perfect precision/recall. These values can be combined into one metric, the F1 score, which is the harmonic average of the precision and recall scores. The F1 score also ranges [0,1].

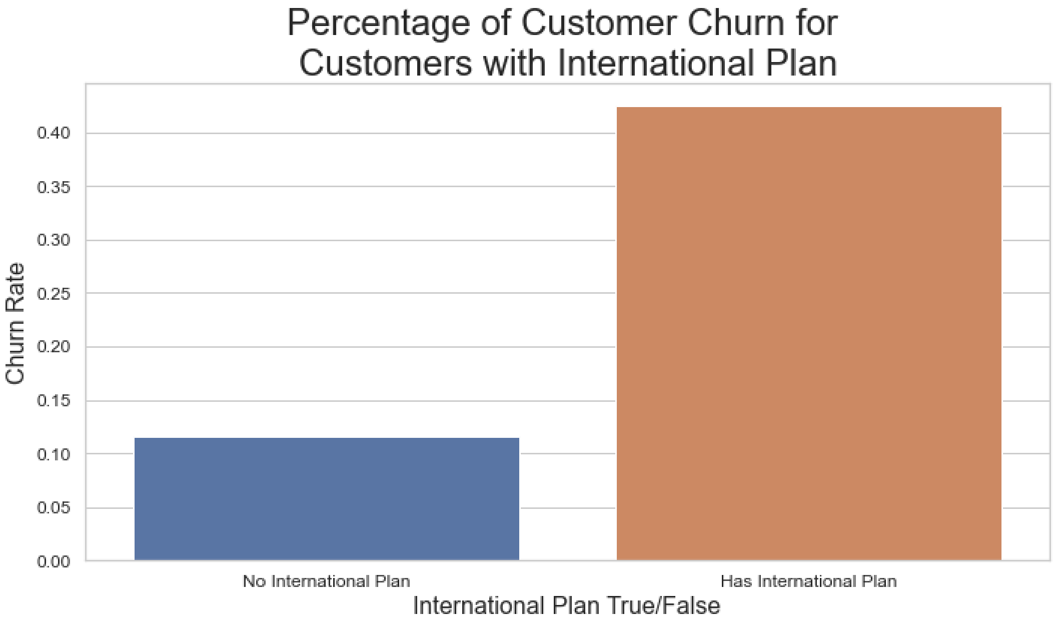
The closer to 1 the F1 score, the more perfect the model is when classifying samples.

Churn By International Plan

In [330]:

```
int_plan_churn = pd.DataFrame(df.groupby(['international plan'])['churn'].mean())

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = [0, 1], y = 'churn', data = int_plan_churn, ax = ax)
plt.title('Percentage of Customer Churn for \nCustomers with International Plan', fontsize = 24)
ax.tick_params(axis = 'both', labelsize = 12)
plt.xlabel('International Plan True/False', fontsize = 16)
plt.ylabel('Churn Rate', fontsize = 16)
ax.set_xticklabels(['No International Plan', 'Has International Plan'])
plt.tight_layout()
```

executed in 192ms, finished 17:13:31 2022-05-27



In [385]:

```
pre_df.groupby("international_plan")["churn"].value_counts(normalize = True)
```

executed in 21ms, finished 19:10:46 2022-05-28

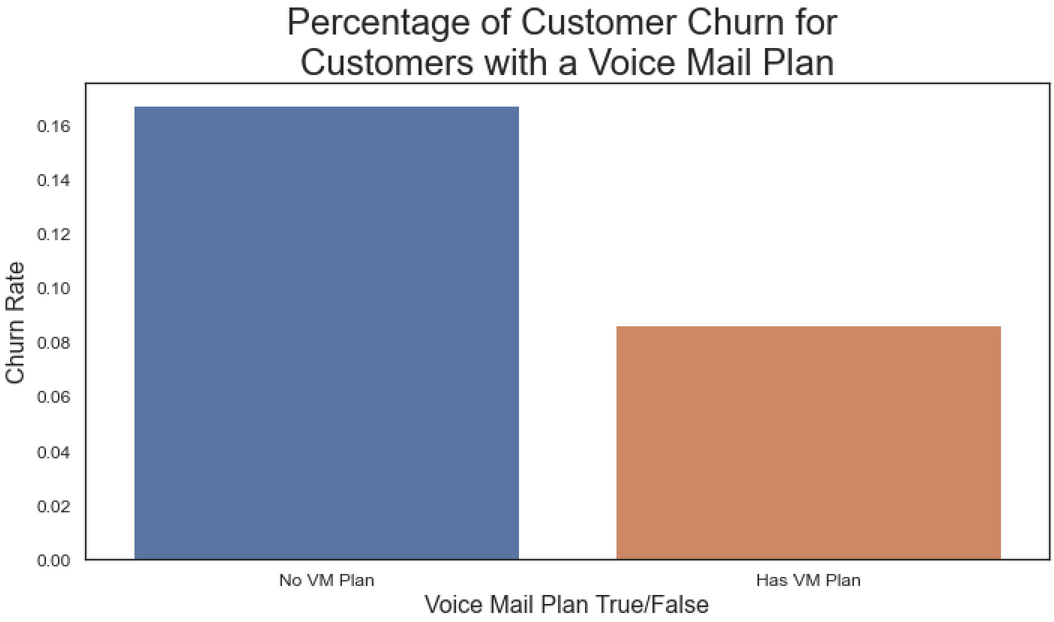
Out[385]:

```
international_plan  churn
0                   False    0.885050
                    True     0.114950
1                   False    0.575851
                    True     0.424149
Name: churn, dtype: float64
```

Here we can see that the churn rate for those who do not have an international plan is only 11.5% whereas the churn rate for those who do have an international rate is 42.4%

Churn By Voice Mail Plan

In [411]:
```
vm_plan_churn = pd.DataFrame(df.groupby(['voice mail plan'])['churn'].mean())

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = [0, 1], y = 'churn', data = vm_plan_churn, ax = ax)
plt.title('Percentage of Customer Churn for \nCustomers with a Voice Mail Plan', fontsize = 24)
ax.tick_params(axis = 'both', labelsize = 12)
plt.xlabel('Voice Mail Plan True/False', fontsize = 16)
plt.ylabel('Churn Rate', fontsize = 16)
ax.set_xticklabels(['No VM Plan', 'Has VM Plan'])
plt.tight_layout()
```
executed in 199ms, finished 20:54:43 2022-05-28



In [407]:
```
df.groupby("voice mail plan")["churn"].value_counts(normalize = True)
```
executed in 19ms, finished 20:52:37 2022-05-28

Out[407]:
```
voice mail plan  churn
no               False    0.832849
                 True     0.167151
yes              False    0.913232
                 True     0.086768
Name: churn, dtype: float64
```

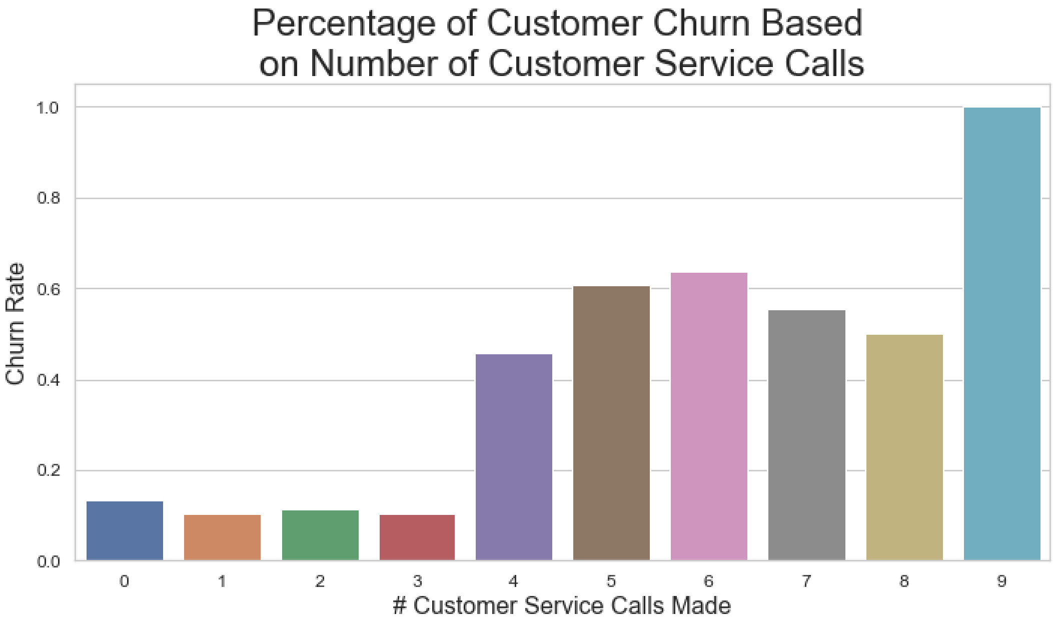
Customers without a voice mail plan have a churn rate of 16.7%

Churn By Customer Service Calls

In [334]:

```
cust_calls = pd.DataFrame(df.groupby(['customer service calls'])['churn'].mean())

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], y = 'churn', data = cust_calls, ax = ax)
plt.title('Percentage of Customer Churn Based \non Number of Customer Service Calls', fontsize = 24)
ax.tick_params(axis = 'both', labelsize = 12)
plt.xlabel('# Customer Service Calls Made', fontsize = 16)
plt.ylabel('Churn Rate', fontsize = 16)
plt.tight_layout()
```

executed in 252ms, finished 17:13:59 2022-05-27

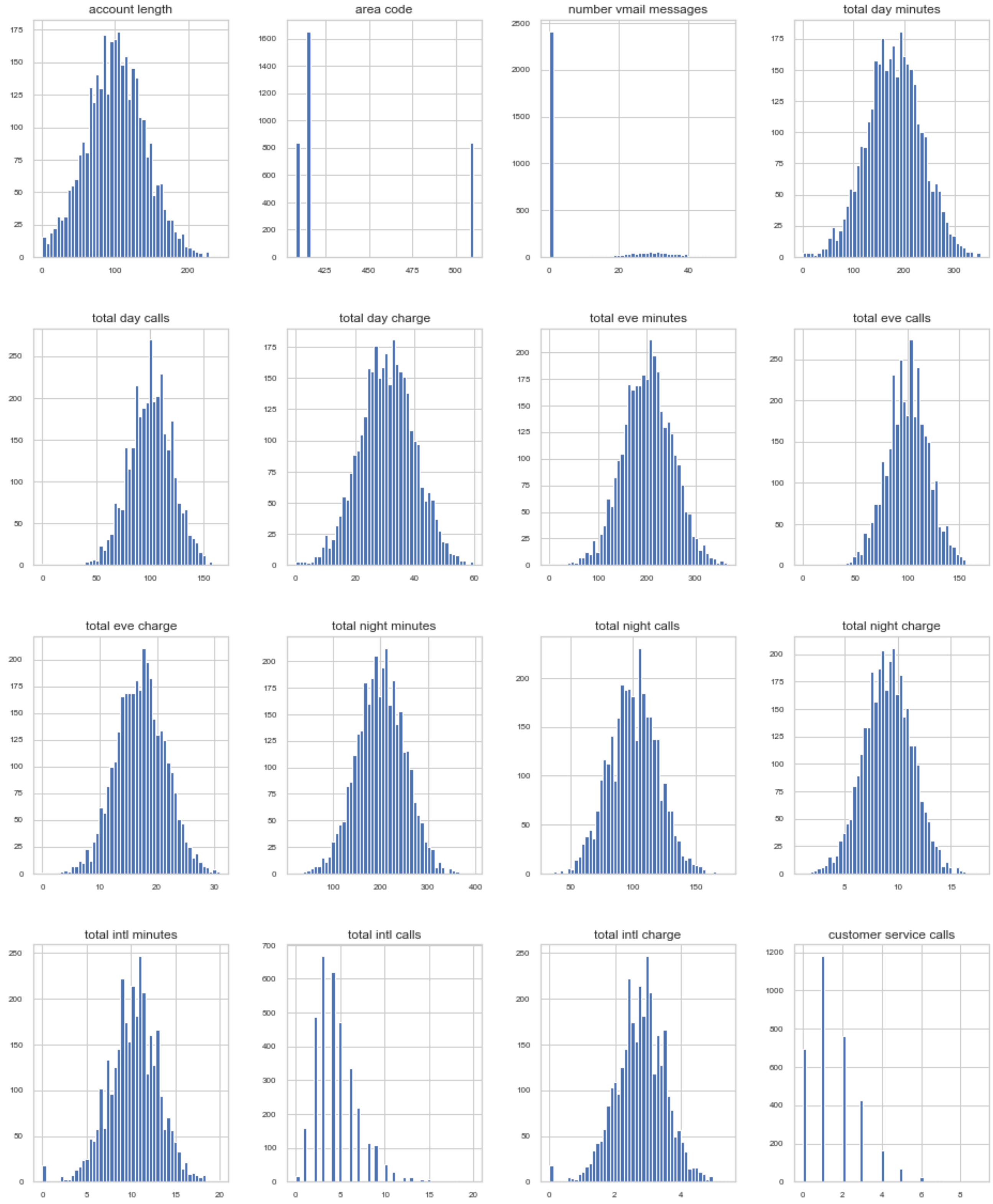


Here we can see for those customers calling customer service four or more times, they have a higher churn rate than those who call customer service three or fewer times.

Distribution of Continuous Features

In [17]:
```
df[df.select_dtypes(exclude = object and bool).columns].hist(figsize=(16, 20),
                                                             bins = 50, xlabelsize = 8, ylabelsize = 8);
```

executed in 5.49s, finished 14:53:46 2022-05-24



While this project does not utilize linear regression and thus distribution normality is not an assumption we need to verify and meet, this visualization is still valuable to identify if there are any significant outliers or skew that may need to be addressed in data processing.

Finally, a note on the limitations of the dataset:

This dataset does not include any information related to dates, so I do not know how recent or out-of-date the data points are. Not knowing the timeframe of the data may impact conclusions in that data from 20 years ago may not accurately reflect data from 6 months ago.

5 Data Modeling

5.1 Prepare the Data for Modeling

In [21]:

```
# Split the data

y = pre_df["churn"]
X = pre_df.drop(["churn"], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 47, stratify = y)
```

executed in 37ms, finished 14:53:46 2022-05-24

Here, I set the stratify parameter to "yes" because we have an unbalanced dataset. Stratifying our train-test-split ensures that relative class frequencies are approximately preserved in each train and validation fold.

In [22]:

```
# Standardize the data

# Instantiate StandardScaler
scaler = scaler = StandardScaler()

# Transform the training and test sets
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

executed in 17ms, finished 14:53:46 2022-05-24

5.2 Functions for Efficiency

In [336]:

```
# Write a function that will print the model metrics

def print_metrics(clf):
    # Predict on training and test sets
    training_preds = clf.predict(X_train_scaled)
    test_preds = clf.predict(X_test_scaled)

    # F1-score of training and test sets (include weighted scoring due to
    # imbalanced data)
    clf_training_f1 = f1_score(y_train, training_preds, average = 'weighted')
    clf_test_f1 = f1_score(y_test, test_preds, average = 'weighted')
    clf_f1_delta = clf_training_f1 - clf_test_f1

    print('Training F1-Score: {:.2}'.format(clf_training_f1))
    print('Validation F1-Score: {:.2}'.format(clf_test_f1))
    print('F1-Score Delta: {:.2}'.format(clf_f1_delta))
```

executed in 8ms, finished 17:17:07 2022-05-27

In [341]:

```
# Write a function to plot the feature importances, used with decision tree,
# random forest, and xgboost

def plot_feature_importances(model):
    sns.set_theme(style = "whitegrid")

    pd.Series(model.feature_importances_,
              index = X_train.columns).nlargest(10).sort_values(ascending = True).plot(kind = 'barh')
```

executed in 11ms, finished 17:20:47 2022-05-27

In [342]:

```
# Write a function to plot the feature importances based on GridSearch,
# used with decision tree, random forest, and xgboost

def plot_gs_feature_importances(model):
    sns.set_theme(style = "whitegrid")

    pd.Series(model.best_estimator_.feature_importances_,
              index = X_train.columns).nlargest(10).sort_values(ascending = True).plot(kind = 'barh')
```

executed in 9ms, finished 17:20:48 2022-05-27

In [343]:

```
# Write a function to print the best parameters found during a GridSearchCV

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

executed in 11ms, finished 17:20:49 2022-05-27

In [344]:

```
# Write a function to print the model metrics of the model with the best parameters
# based on the GridSearch, including confusion matric and classification report

def print_gs_metrics(clf):

    # Set SNS
    sns.set_theme(style = "white")

    # Predict on training and test sets
    training_preds = clf.predict(X_train_scaled)
    test_preds = clf.predict(X_test_scaled)

    # F1-score of training and test sets, including 'weighted' for imbalance
    clf_training_f1 = f1_score(y_train, training_preds, average = 'weighted')
    clf_test_f1 = f1_score(y_test, test_preds, average = 'weighted')
    clf_f1_delta = clf_training_f1 - clf_test_f1

    # Print all scores
    print('Training F1-Score: {:.2}'.format(clf_training_f1))
    print('Validation F1-Score: {:.2}'.format(clf_test_f1))
    print('F1-Score Delta: {:.2}'.format(clf_f1_delta))

    print('\n','-'*30,'\n')

    # Print training classification report
    print('Training Classification Report')
    print(classification_report(y_train, training_preds))

    print('\n','-'*30,'\n')

    # Print testing classification report
    print('Testing Classification Report')
    print(classification_report(y_test, test_preds))

    print('\n','-'*30,'\n')

    # Plot testing confusion matrix
    print('Confusion Matrix')
    plot_confusion_matrix(clf, X_test_scaled, y_test, normalize = 'true', cmap = 'Blues');
```

executed in 10ms, finished 17:20:55 2022-05-27

5.3 Baseline Model

5.3.1 Decision Tree Stump

I begin the predictive modeling process by creating a baseline model from which to build from. The baseline model in this case is a decision stump (using a Decision Tree classifier) with just one split, i.e., a max depth of 1. This classifier will be a weak learner.

In [399]:

```
# Set seed for reproducibility

SEED = 47
```

executed in 11ms, finished 19:50:48 2022-05-28

In [400]:

```
# Initialize the decision tree stump

baseline = DecisionTreeClassifier(max_depth = 1,
                                  class_weight = 'balanced',
                                  random_state = SEED)

# Fit the data to the classifier
baseline.fit(X_train_scaled, y_train)

# Print the metrics
print_metrics(baseline)
```

executed in 22ms, finished 19:50:50 2022-05-28

```
Training F1-Score: 0.83
Validation F1-Score: 0.81
F1-Score Delta: 0.024
```

5.4 Vanilla Models

In this section, I create vanilla models using a decision tree classifier, logistic regression, k-Nearest Neighbors classifier, random forest classifier, and eXtreme Gradient Boost (XGBoost) classifier.

Each model has its own advantages and disadvantages, which is why I will include each to best determine the strongest predictive model for the stakeholder.

These models' parameters are mostly defaults and not tuned for specificity or improvement. This allows for more opportunity to determine which model(s) will be the best to recommend.

5.4.1 Decision Tree Classifier

[Decision Trees (https://scikit-learn.org/stable/modules/tree.html)](https://scikit-learn.org/stable/modules/tree.html) are a non-parametric supervised learning method used for classification and regression with the goal of creating a model that predicts the value of a target variable by learning simple decision rules inferred from the data features; in the case of this project, the decision tree classifier will attempt to predict customer churn.

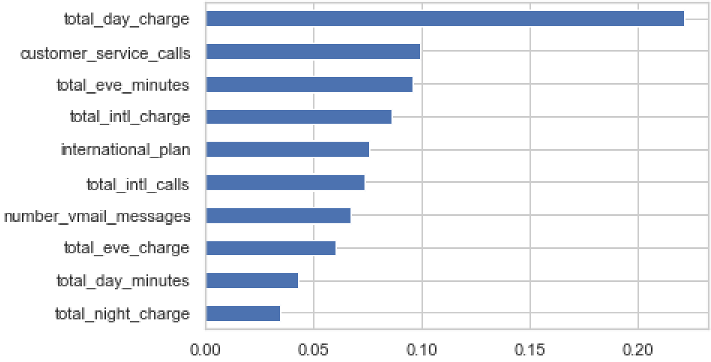
In [388]:

```
# Initialize decision tree classifier

dt = DecisionTreeClassifier(class_weight = 'balanced',
                            random_state = SEED).fit(X_train_scaled, y_train)

print_metrics(dt)
```

executed in 48ms, finished 19:22:14 2022-05-28

```
Training F1-Score: 1.0
Validation F1-Score: 0.91
F1-Score Delta: 0.086
```

In [348]:
```
# Fit the data to the classifier

dt = DecisionTreeClassifier().fit(X_train_scaled, y_train)
plot_feature_importances(dt)
```
executed in 209ms, finished 17:21:05 2022-05-27



5.4.2 Logistic Regression

Logistic regression (https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression) is a linear model that is used for classification and models the probability of one event or class (out of two alternatives) taking place and that the target variable is categorical, e.g., a customer churns (1) or does not churn (0). Thus, logistic regression is an applicable model to our business problem and may be of value.

In [389]:
```
# Initialize the logistic regression classifier and fit the data

lr = LogisticRegression(class_weight = 'balanced',
                        random_state = SEED).fit(X_train_scaled, y_train)

print_metrics(lr)
```
executed in 126ms, finished 19:26:16 2022-05-28

```
Training F1-Score: 0.8
Validation F1-Score: 0.79
F1-Score Delta: 0.013
```

5.4.3 K-Nearest Neighbors Classifier

The neighbors-based classification (https://scikit-learn.org/stable/modules/neighbors.html#classification) is a type of instance-based learning that computes classification from a simple majority vote of the nearest neighbors of each point: a query point is assigned the data class which has the most representatives within the nearest neighbors of the point. For this modeling, k-Nearest Neighbors implements learning based on the nearest neighbors of each query point, where k is an integer value specified by the user.

In [390]:
```
# Initialize the KNN classifier and fit the data

knn = KNeighborsClassifier().fit(X_train_scaled, y_train)

print_metrics(knn)
```
executed in 369ms, finished 19:26:31 2022-05-28

```
Training F1-Score: 0.92
Validation F1-Score: 0.87
F1-Score Delta: 0.05
```

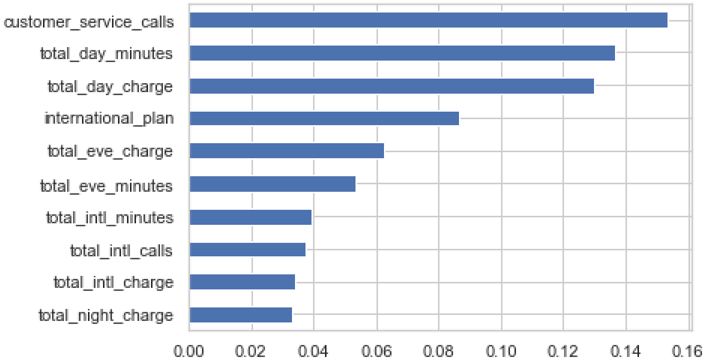
5.4.4 Random Forest Classifier

Random Forest Classifiers (https://scikit-learn.org/stable/modules/ensemble.html#forest) are a type of ensemble method, which means a diverse set of classifiers is created by introducing randomness in the classifier construction; in this case, he prediction of the ensemble is given as the averaged prediction of the individual decision tree classifiers. In random forests, each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set.

In [391]:
```
# Initialize the random forest classifier and fit the data

rf = RandomForestClassifier(class_weight = 'balanced',
                            random_state = SEED).fit(X_train_scaled, y_train)

print_metrics(rf)
```
executed in 633ms, finished 19:26:43 2022-05-28

```
Training F1-Score: 1.0
Validation F1-Score: 0.94
F1-Score Delta: 0.064
```

In [353]:
```
plot_feature_importances(rf)
```
executed in 328ms, finished 17:22:25 2022-05-27



5.4.5 XGBoost

From its documentation, XGBoost (https://xgboost.readthedocs.io/en/latest/index.html) is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework.

Gradient Boosting alogrithms are a more advanced boosting algorithm that makes use of Gradient Descent. It starts with a weak learner that makes predictions on the dataset. The algorithm then checks this learner's performance, identifying examples that it got right and wrong. The model then calculates the Residuals for each data point, to determine how far off the mark each prediction was. The model then combines these residuals with a Loss Function to calculate the overall loss.

XGBoost, or eXtreme Gradient Boosting, provides a parallel tree boosting that solve many data science problems in a fast and accurate way. It is a stand-alone library that implements popular gradient boosting algorithms in the fastest, most performant way possible. In fact, XGBoost provides best-in-class performance compared to other classification algorithms.

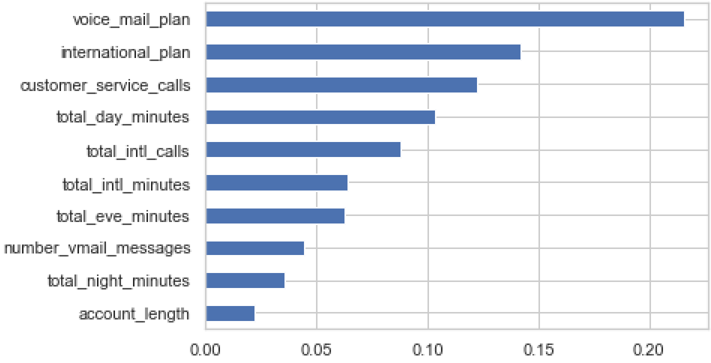
In [392]:
```
# Instantiate XGBClassifier and and fit the data
xgb = XGBClassifier(seed = SEED).fit(X_train_scaled, y_train)

print_metrics(xgb)
```
executed in 377ms, finished 19:27:04 2022-05-28

```
Training F1-Score: 1.0
Validation F1-Score: 0.94
F1-Score Delta: 0.056
```

In [356]:

```
xgb_clf = XGBClassifier().fit(X_train_scaled, y_train)
plot_feature_importances(xgb_clf)
```

executed in 433ms, finished 17:22:57 2022-05-27



5.4.6 Summary Results Table

In [393]:

```
# Build a table of the results from the set of vanilla models

classifiers = [dt, lr, knn, rf, xgb]
classifier_names = ["Decision Tree", "Logistic Regression", "KNN", "Random Forest", "XGBoost"]

results_table = pd.DataFrame(columns=["Training F1", "Validation F1"])
for (i, clf), name in zip(enumerate(classifiers), classifier_names):
    X_pred = clf.predict(X_train_scaled)
    y_pred = clf.predict(X_test_scaled)
    row = []
    row.append(f1_score(y_train, X_pred, average = 'weighted'))
    row.append(f1_score(y_test, y_pred, average = 'weighted'))
    row = [float("%.2f" % r) for r in row]
    results_table.loc[name] = row

results_table['Delta'] = results_table['Training F1'] - results_table['Validation F1']
```

executed in 474ms, finished 19:27:35 2022-05-28

In [394]:

```
# Display the results table

results_table
```

executed in 31ms, finished 19:27:36 2022-05-28

Out[394]:

	Training F1	Validation F1	Delta
Decision Tree	1.00	0.91	0.09
Logistic Regression	0.80	0.79	0.01
KNN	0.92	0.87	0.05
Random Forest	1.00	0.94	0.06
XGBoost	1.00	0.94	0.06

With our vanilla models, both Random Forest and XGBoost performed the best, without any tuning of hyperparameters.

On the test data, both RF and XGB had an F1 score value of 0.94, the closet to 1.0 out of all the other vanilla models.

Compared to the baseline model, with an F1 score value of 0.81 on the test data, our RF and XGB vanilla models are already showing improvement in predictive performance.

Furthermore, we are not seeing evidence of significant overfitting, which is good.

5.5 Preparing for Model Tuning

Following the first set of vanilla model scores, there is certainly room for improvement in our predictive modeling. While the vanilla RF and XGBoost models had an F1 score of 0.91 on the testing data, it may be possible with further tuning of all five models that we can improve that F1 score.

To best solve our business problem of predicting customer churn, these iterative models will attempt to improve results by utilizing `GridSearchCV` to find the best performing hyperparameters for each model.

In this section, I also include more information on feature importances, classification reports, and confusion matrices.

5.5.1 Feature Importances

Feature Importances (https://towardsdatascience.com/understanding-feature-importance-and-how-to-implement-it-in-python-ff0287b20285) refer to techniques that calculate a score for all the input features for a given model; the scores simply represent the "importance" of each feature with a higher score indicating that the specific feature will have a larger effect on the model that is being used to predict a certain variable.

5.5.2 Classification Report

The **Classification Report** (https://scikit-learn.org/stable/modules/model_evaluation.html#classification-report) is a printed report showing the main classification metrics: precision, recall, and the F1 score, along with the sample size (indicated as 'support').

Remember, precision is the ability of the classifier not to label as positive a sample that is negative (what percent of the predictions were correct?), and recall is the ability of the classifier to find all the positive samples (what percent of the positive cases were caught?). The F1 score can be interpreted as a weighted harmonic mean of the precision and recall that reaches its best value at 1 and its worst score at 0.

The report also includes the macro average (averaging the unweighted mean per label) and weighted average (averaging the support-weighted mean per label).

The classification report is meaningful to this business problem because the model eventually chosen to predict customer churn should correctly determine a given class.

5.5.3 Confusion Matrix

The **Confusion Matrix** (https://scikit-learn.org/stable/modules/model_evaluation.html#confusion-matrix) is a function evaluates classification accuracy by computing the confusion matrix with each row corresponding to the true class. Since our data is imbalanced, the included confusion matrices will be normalized such that it will show the percentage prediction of each class made by the model for that specific true label.

The higher the diagonal values of the confusion matrix the better, indicating many correct predictions of True Positives and True Negatives.

5.6 Model Tuning

5.6.1 Decision Tree Classifier

In [284]:

```python
# Identify parameters to search for the decision tree classifier

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

executed in 61ms, finished 18:24:52 2022-05-25

In [360]:

```
# Initialize the decision tree classifier, run the GridSearch, and fit the data to the model

dt = DecisionTreeClassifier(class_weight = 'balanced', random_state = SEED)

dt_gs_clf = GridSearchCV(dt, dt_param_grid, scoring = 'f1_weighted')
dt_gs_clf.fit(X_train_scaled, y_train)

print_best_params(dt_gs_clf)
```

executed in 4.74s, finished 17:24:49 2022-05-27

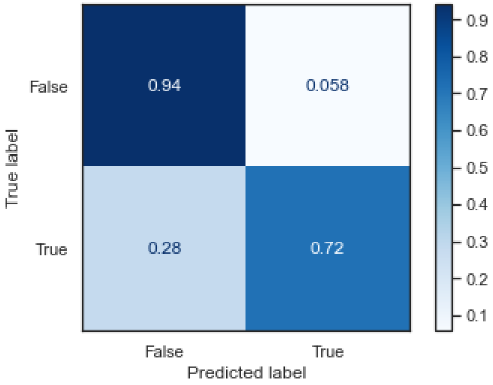
```
Grid Search found the following optimal parameters:
criterion: 'entropy'
max_depth: 6
min_samples_leaf: 1
min_samples_split: 2
```

In [361]:

```
print_gs_metrics(dt_gs_clf)
```

executed in 239ms, finished 17:26:03 2022-05-27

```
Training F1-Score: 0.96
Validation F1-Score: 0.91
F1-Score Delta: 0.044

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.98      0.97      0.97      2137
        True       0.82      0.88      0.85       362

    accuracy                           0.95      2499
   macro avg       0.90      0.93      0.91      2499
weighted avg       0.96      0.95      0.96      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.94      0.95       713
        True       0.68      0.72      0.70       121

    accuracy                           0.91       834
   macro avg       0.82      0.83      0.82       834
weighted avg       0.91      0.91      0.91       834


 ------------------------------

Confusion Matrix
```

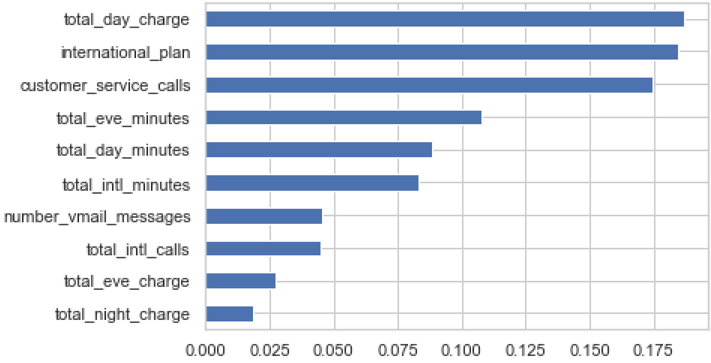


5.6.1.1 Results Summary

With this iterative model of the decision tree classifer, the F1-score on the training data is 0.96 whereas the F1-score on the test data is 0.91. With a delta of 0.04, this model has minimal overfitting. Further, its F1-score on the test data is improved from the baseline decision tree model (0.81) and the vanilla decision tree model (0.91).

The confusion matrix also indicates that this model correctly predicted 94% of True Negatives and 72% of True Positives.

In [362]:
```
sns.set_theme(style = "whitegrid")
plot_gs_feature_importances(dt_gs_clf)
```
executed in 202ms, finished 17:26:18 2022-05-27



The features that have the largest effect on the model are:

1. the total amount of money charged by SyriaTel per day
2. the number of calls the customer made to customer service
3. whether the customer has an international plan
4. the total number of international minutes
5. the number of international calls

5.6.2 Logistic Regression

In [365]:
```
# Identify parameters to search for the logistic regression model

lr_param_grid = {
    'C': [1e-1, 1e2, 1e4],
    'penalty': ['l2', 'l1'],
    'solver': ['lbfgs', 'liblinear', 'sag'],
    'max_iter': [1e-1, 1e2, 1e4]
}
```
executed in 7ms, finished 17:33:13 2022-05-27

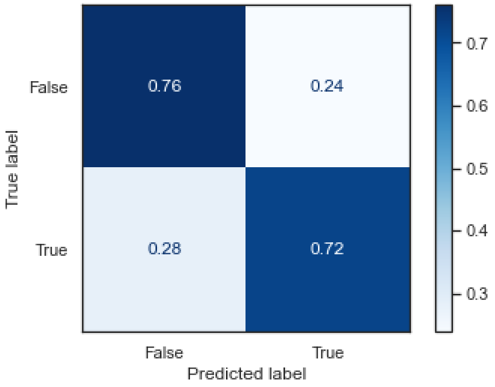
In [366]:
```
# Initialize the logistic regression, run the GridSearch, and fit the data to the model

lr = LogisticRegression(class_weight = 'balanced', random_state = SEED)

lr_gs_clf = GridSearchCV(lr, lr_param_grid, scoring = 'f1_weighted')
lr_gs_clf.fit(X_train_scaled, y_train)

print_best_params(lr_gs_clf)
```
executed in 5m 7s, finished 17:38:23 2022-05-27

```
Grid Search found the following optimal parameters:
C: 0.1
max_iter: 100.0
penalty: 'l2'
solver: 'lbfgs'
```

```
In [367]: print_gs_metrics(lr_gs_clf)
```

executed in 203ms, finished 17:38:35 2022-05-27

```
Training F1-Score: 0.8
Validation F1-Score: 0.79
F1-Score Delta: 0.017

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.95      0.78      0.86      2137
        True       0.37      0.76      0.49       362

    accuracy                           0.78      2499
   macro avg       0.66      0.77      0.67      2499
weighted avg       0.87      0.78      0.80      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.94      0.76      0.84       713
        True       0.34      0.72      0.46       121

    accuracy                           0.76       834
   macro avg       0.64      0.74      0.65       834
weighted avg       0.85      0.76      0.79       834


 ------------------------------

Confusion Matrix
```



5.6.2.1 Results Summary

With this iterative model of the logistic regression model, the F1-score on the training data is 0.80 and the F1-score on the test data is 0.79. With a delta of 0.02, this model does not seem to be overfitting the data. Its F1-score on the test data (0.79) is actually a tiny bit lower than that of the F1 score of the baseline model (0.81).

Additionally, the confusion matrix shows overall weaker predictions than the tuned decision tree model, correctly predicting only 76% of True Negatives and 72% of True Positives.

5.6.3 K-Nearest Neighbor Classifier

In [41]:

```
# Identify parameters to search for the KNN classifier model

knn_param_grid = {
    'n_neighbors': [3, 5, 7],
    'algorithm': ['ball_tree', 'kd_tree', 'brute'],
    'metric': ['minkowski', 'euclidean', 'manhattan']
}
```

executed in 10ms, finished 14:57:29 2022-05-24

In [370]:

```
# Initialize the KNN classifier, run the GridSearch, and fit the data to the model

knn = KNeighborsClassifier()

knn_gs_clf = GridSearchCV(knn, knn_param_grid, scoring = 'f1_weighted')
knn_gs_clf.fit(X_train_scaled, y_train)

print_best_params(knn_gs_clf)
```

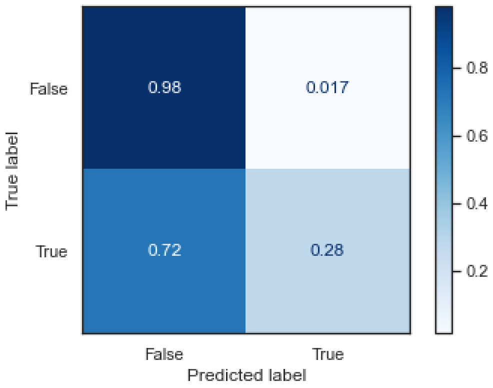
executed in 4.74s, finished 17:39:33 2022-05-27

```
Grid Search found the following optimal parameters:
algorithm: 'ball_tree'
metric: 'manhattan'
n_neighbors: 3
```

```
In [371]: print_gs_metrics(knn_gs_clf)
```

executed in 450ms, finished 17:39:35 2022-05-27

```
Training F1-Score: 0.94
Validation F1-Score: 0.86
F1-Score Delta: 0.079

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.94      0.99      0.97      2137
        True       0.94      0.64      0.76       362

    accuracy                           0.94      2499
   macro avg       0.94      0.82      0.86      2499
weighted avg       0.94      0.94      0.94      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.89      0.98      0.93       713
        True       0.74      0.28      0.41       121

    accuracy                           0.88       834
   macro avg       0.81      0.63      0.67       834
weighted avg       0.87      0.88      0.86       834


 ------------------------------

Confusion Matrix
```



5.6.3.1 Results Summary

With this iterative model of the KNN classifier, the F1-score on the training data is 0.94 whereas the F1-score on the test data is 0.86. With a delta of 0.08, this model does not seem to be overfitting the data. Its F1-score on the test data (0.86) is a bit better than that of the baseline (0.81).

However, there is significant concern with this model's confusion matrix, indicating this model is correctly predicting True Positives only 28% of the time and predicting False Positives 72% of the time. This is not good.

5.6.4 Random Forest Classifier

In [54]:

```
# Identify parameters to search for the random forest classifier model

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

executed in 5ms, finished 14:57:42 2022-05-24

In [372]:

```
# Initialize the random forest classifier, run the GridSearch, and fit the data to the model

rf = RandomForestClassifier(class_weight = 'balanced', random_state = SEED)

rf_gs_clf = GridSearchCV(rf, rf_param_grid, scoring = 'f1_weighted')
rf_gs_clf.fit(X_train_scaled, y_train)

print_best_params(rf_gs_clf)
```

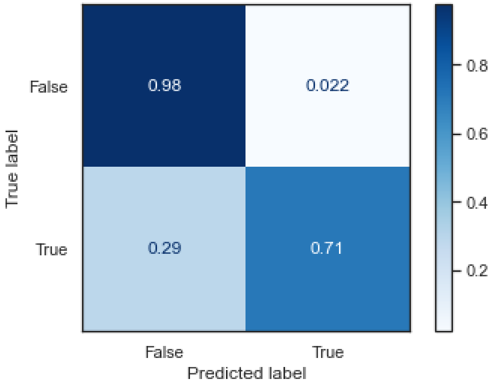
executed in 1m 38.8s, finished 17:41:25 2022-05-27

```
Grid Search found the following optimal parameters:
criterion: 'entropy'
max_depth: None
min_samples_leaf: 3
min_samples_split: 2
```

```
In [373]: print_gs_metrics(rf_gs_clf)
```

executed in 263ms, finished 17:41:34 2022-05-27

```
Training F1-Score: 1.0
Validation F1-Score: 0.94
F1-Score Delta: 0.059

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2137
        True       0.98      0.99      0.99       362

    accuracy                           1.00      2499
   macro avg       0.99      0.99      0.99      2499
weighted avg       1.00      1.00      1.00      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.98      0.96       713
        True       0.84      0.71      0.77       121

    accuracy                           0.94       834
   macro avg       0.90      0.84      0.87       834
weighted avg       0.94      0.94      0.94       834


 ------------------------------

Confusion Matrix
```

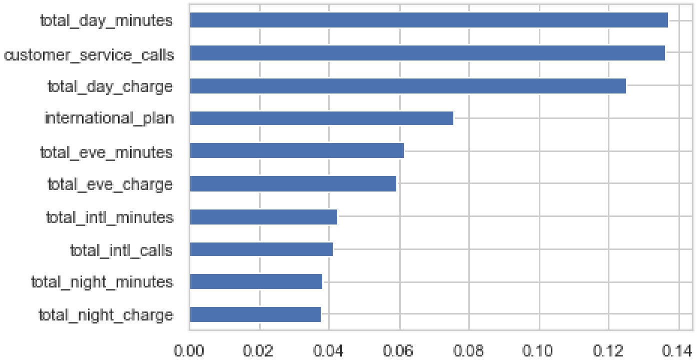


5.6.4.1 Results Summary

With this iterative model of the random forest classifier, the F1-score on the training data is 1.0 and the F1-score on the test data is 0.94, the strongest yet. With a delta of 0.06, this model does not seem to be overfitting the data. Its F1-score on the test data (0.94) is higher than the baseline model (0.81).

Further, this model is correctly predicting 98% of True Negatives and 71% of True Positives. Its Type 1 Error (False Positives) are being predicted 29% of the time.

In [374]:
```
plot_gs_feature_importances(rf_gs_clf)
```
executed in 229ms, finished 17:41:44 2022-05-27



The features that have the largest effect on the model are:

1. the total number of minutes used per day
2. the amount of money charged by SyriaTel per day
3. the number of calls the customer made to customer service
4. whether the customer has an international plan
5. total number of minutes used per night.

Four of the five top features of this Random Forest classifier are the same as those of the Decision Tree classifier.

5.6.5 XGBoost

In [286]:
```
# Identify parameters to search for the XGBoost classifier model

xgb_param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [2, 6, 10],
    'min_child_weight': [0, 1, 2],
    'subsample': [0.3, 0.5, 0.7],
    'n_estimators': [10, 30, 100],
}
```
executed in 52ms, finished 19:35:35 2022-05-25

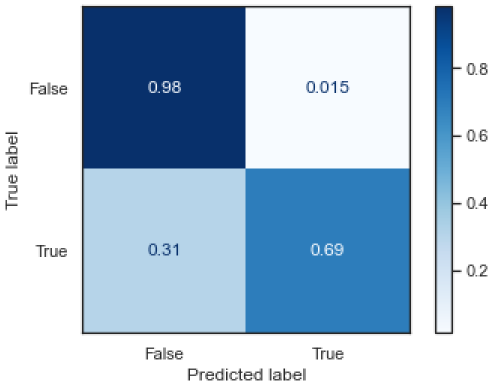
In [375]:
```
# Initialize the XGBoost classifier, run the GridSearch, and fit the data to the model

xgb_clf = XGBClassifier(seed = SEED)

xgb_gs_clf = GridSearchCV(xgb_clf, xgb_param_grid, scoring = 'f1_weighted')
xgb_gs_clf.fit(X_train_scaled, y_train)

print_best_params(rf_gs_clf)
```
executed in 1m 14.4s, finished 17:43:30 2022-05-27

```
Grid Search found the following optimal parameters:
criterion: 'entropy'
max_depth: None
min_samples_leaf: 3
min_samples_split: 2
```

```
In [376]: print_gs_metrics(xgb_gs_clf)
```

executed in 220ms, finished 17:43:34 2022-05-27

```
Training F1-Score: 0.98
Validation F1-Score: 0.94
F1-Score Delta: 0.045

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.98      1.00      0.99      2137
        True       1.00      0.90      0.94       362

    accuracy                           0.98      2499
   macro avg       0.99      0.95      0.97      2499
weighted avg       0.99      0.98      0.98      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.98      0.97       713
        True       0.88      0.69      0.78       121

    accuracy                           0.94       834
   macro avg       0.92      0.84      0.87       834
weighted avg       0.94      0.94      0.94       834


 ------------------------------

Confusion Matrix
```

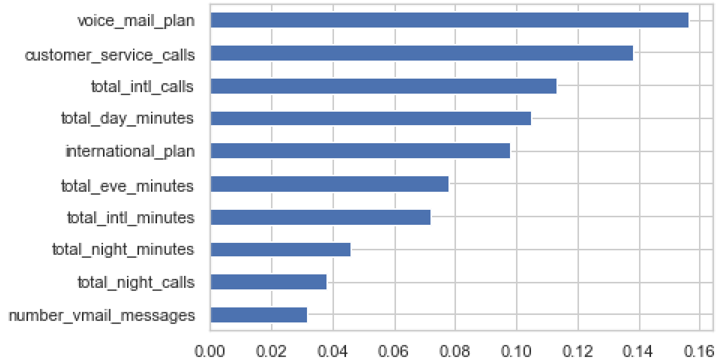


5.6.5.1 Results Summary

With this iterative model of the XGBoost classifier, the F1-score on the training data is 0.98 and the F1-score on the test data is 0.94. With a delta of 0.05, this model does not seem to be overfitting the data. Its F1-score is higher (0.94) than the baseline decision stump model (0.81).

Looking at the confusion matrix, the XGBoost model is correctly predicting 98% of True Negatives and 69% of True Positives. Its Type 1 Error (False Positives) are being predicted 31% of the time. The True Positive prediction percentage (69%) is marginally lower than the tuned random forest model and its percentage of Type 1 errors (31%) is marginally higher than the tuned random forest model as well.

In [377]:
```
plot_gs_feature_importances(xgb_gs_clf)
```
```
executed in 221ms, finished 17:43:44 2022-05-27
```



The features that have the largest effect on this XGBoost model are:

1. the number of calls the customer made to customer service
2. whether the customer has a voice mail plan
3. the total number of minutes used per day
4. whether the customer has an international plan
5. the total number of international calls made

Two of the five top features of this XGBoost classifier are the same as those of both the Decision Tree classifier and Random Forest classifier (number of calls made to customer service and whether the customer has an international plan).

5.6.6 Summary Results Table

In [395]:
```
# Build a table of the results from the set of tuned models

classifiers = [lr_gs_clf, knn_gs_clf, dt_gs_clf, rf_gs_clf, xgb_gs_clf]
classifier_names = ["Decision Tree", "Logistic Regression", "KNN", "Random Forest", "XGBoost"]

accs = []
recalls = []
precision = []
results_table = pd.DataFrame(columns=["accuracy", "precision", "recall", "f1"])
for (i, clf), name in zip(enumerate(classifiers), classifier_names):
    y_pred = clf.predict(X_test_scaled)
    row = []

    # positive class for each metric
    row.append(accuracy_score(y_test, y_pred))
    row.append(precision_score(y_test, y_pred, average = 'weighted'))
    row.append(recall_score(y_test, y_pred, average = 'weighted'))
    row.append(f1_score(y_test, y_pred, average = 'weighted'))
    row = ["%.3f" % r for r in row]
    results_table.loc[name] = row
```
```
executed in 284ms, finished 19:50:00 2022-05-28
```

In [396]:
```
results_table
```
```
executed in 30ms, finished 19:50:02 2022-05-28
```

Out[396]:

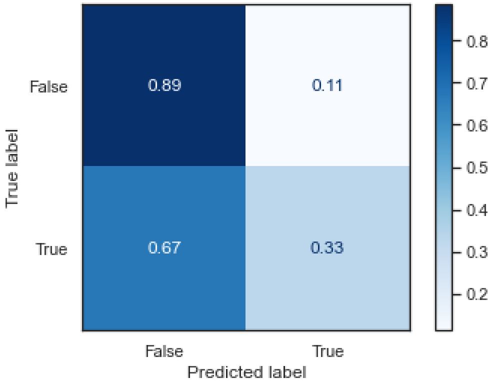
	accuracy	precision	recall	f1
Decision Tree	0.755	0.854	0.755	0.787
Logistic Regression	0.881	0.868	0.881	0.858
KNN	0.910	0.912	0.910	0.911
Random Forest	0.939	0.936	0.939	0.937
XGBoost	0.942	0.940	0.942	0.939

6 Evaluation

```
In [401]: print('Baseline Model Metrics')
          print_gs_metrics(baseline)
```

executed in 309ms, finished 19:52:14 2022-05-28

```
Baseline Model Metrics
Training F1-Score: 0.83
Validation F1-Score: 0.81
F1-Score Delta: 0.024

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.90      0.89      0.90      2137
        True       0.41      0.44      0.42       362

    accuracy                           0.83      2499
   macro avg       0.66      0.67      0.66      2499
weighted avg       0.83      0.83      0.83      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.89      0.89      0.89       713
        True       0.33      0.33      0.33       121

    accuracy                           0.81       834
   macro avg       0.61      0.61      0.61       834
weighted avg       0.81      0.81      0.81       834


 ------------------------------

Confusion Matrix
```

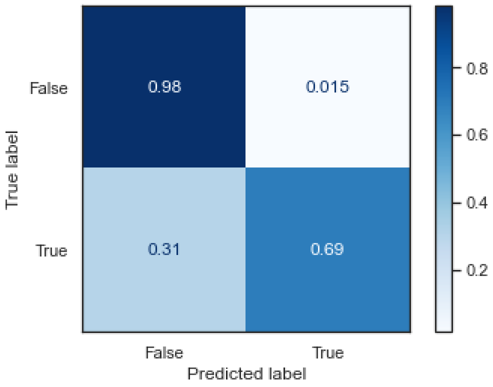


In [402]:

```
print('Final Model Metrics')
print_gs_metrics(xgb_gs_clf)
```

executed in 274ms, finished 19:53:34 2022-05-28

```
Final Model Metrics
Training F1-Score: 0.98
Validation F1-Score: 0.94
F1-Score Delta: 0.045

 ------------------------------

Training Classification Report
              precision    recall  f1-score   support

       False       0.98      1.00      0.99      2137
        True       1.00      0.90      0.94       362

    accuracy                           0.98      2499
   macro avg       0.99      0.95      0.97      2499
weighted avg       0.99      0.98      0.98      2499


 ------------------------------

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.98      0.97       713
        True       0.88      0.69      0.78       121

    accuracy                           0.94       834
   macro avg       0.92      0.84      0.87       834
weighted avg       0.94      0.94      0.94       834


 ------------------------------

Confusion Matrix
```



The final model selected to address the business problem of predicting customer churn is the tuned XGBoost classifier. Remember, the XGBoost classifier provides parallel tree boosting that can solve many data science problems in a fast and accurate way.

The F1-score is the best metric to measure the performance of the predictive modeling for this business case because it is the harmonic mean of precision and recall, two measures important to predicting a binary class (churn or no churn), averaged to best evaluate imbalanced data.

The F1-score of this final model (the tuned XGBoost model) is 0.94 whereas the baseline model's F1-score is 0.81. Further, this final model correctly predicts True Negatives 98% of the time and True Positives 69% of the time.

A couple limitations of XGBoost that must be noted: XGBoost is more likely to overfit than bagging does (i.e. random forest) and it does not perform as well on sparse and unstructured data. However, in this business case, our data is robust with few outliers (based on exploratory analysis) and a selection of hyperparameters that perform the best.

Lastly, this final model does predict Type 1 errors (False Positives) 31% of the time, so that must be considered in predicting customer churn on future, unseen data.

7 Recommendations for Future Work

- Provide a larger dataset: The predictive modeling for SyriaTel can be improved upon for more enhanced performance with a larger dataset from which the models can be trained. Further, the dataset should include dates/time periods to understand when the data had been collected.
- Further Analysis of Feature Importances: Whether SyriaTel's customers have a voice mail plan or an international plan as well as the number of international calls made or number of calls made to customer service all had more influence on the churn rate than other features.
- Evaluate Customer Service: Conduct customer service surveys for more information on why customers are calling
- Conduct industry benchmarking to determine how SyriaTel's voice mail and international plans compare to its offerings