1 Customer Churn

According to Profitwell (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).

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. The F1 score is the harmonic average 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 classifying samples.

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). 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]:
```
# 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]:
```
# Import dataset

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

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

In [3]:
```
# 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]:
```
# 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

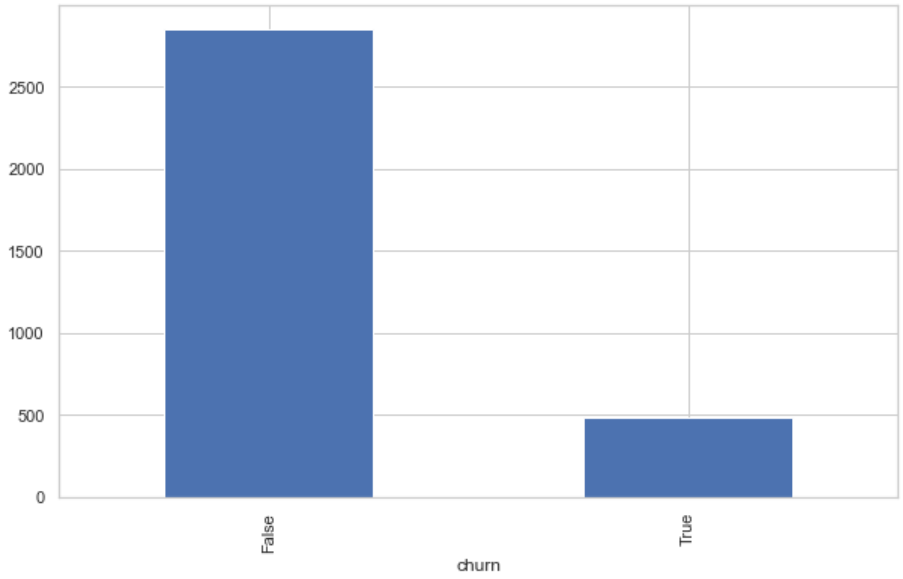
As noted above, this dataset has 3,333 observations with 21 different features. The only feature that will not be included in the analysis is each observation's phone number as this functions as a unique identifier and thus, does not add value to any predictive model.

4.4.1 Data Visualizations

Distribution of Churn

In [10]:
```
sns.set_theme(style = "whitegrid")
pre_df.groupby(["churn"]).size().plot(kind = 'bar', figsize = (10, 6));
```

executed in 324ms, finished 14:53:35 2022-05-24



In [11]:
```
print("Raw Counts")
print(df["churn"].value_counts())
print()
print("Percentages")
print(df["churn"].value_counts(normalize=True))
```

executed in 23ms, finished 14:53:35 2022-05-24

```
Raw Counts
False    2850
True      483
Name: churn, dtype: int64

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

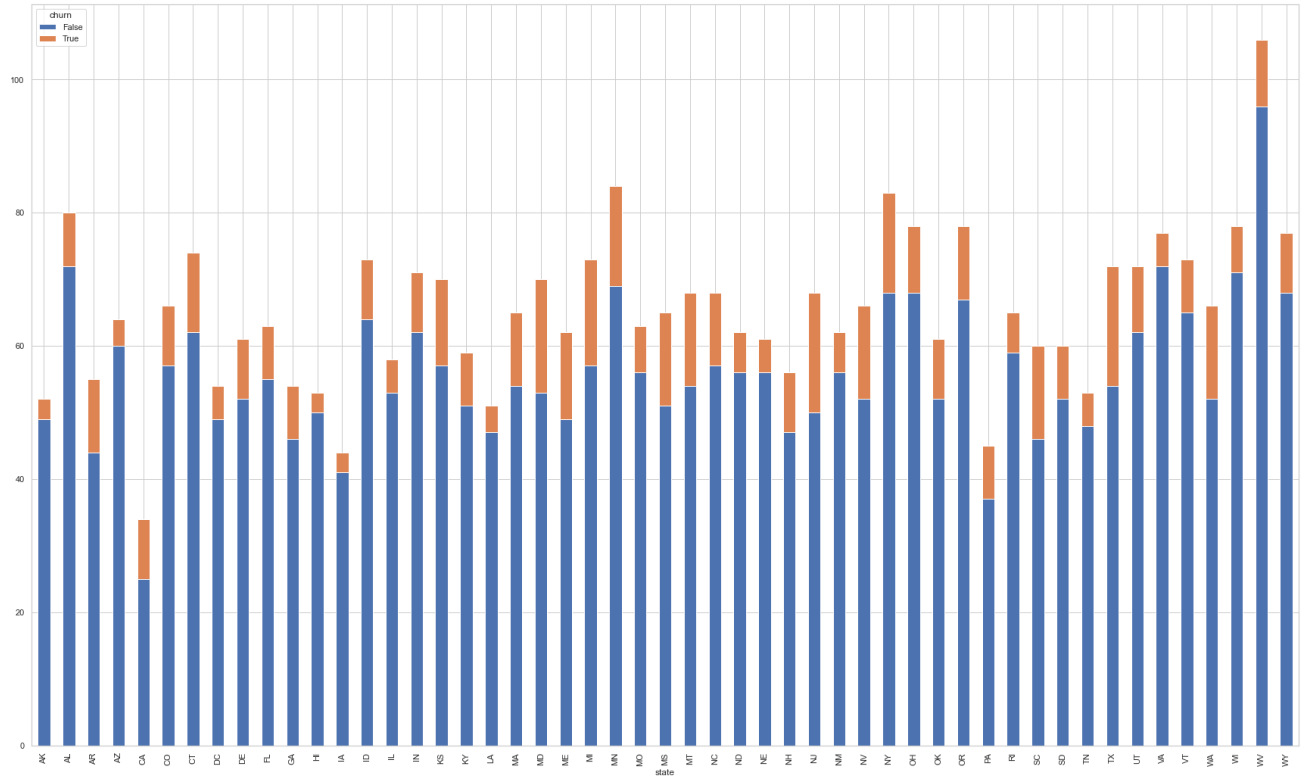
Here we can see of the 3,333 observations (or accounts), 2,850 (86%) have not churned whereas 483 accounts (14%) have churned.

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.

Churn By State

In [73]:
```
df.groupby(["state", "churn"]).size().unstack().plot(
    kind = 'bar', stacked = True, figsize = (30,18));
```

executed in 1.62s, finished 17:23:50 2022-05-24

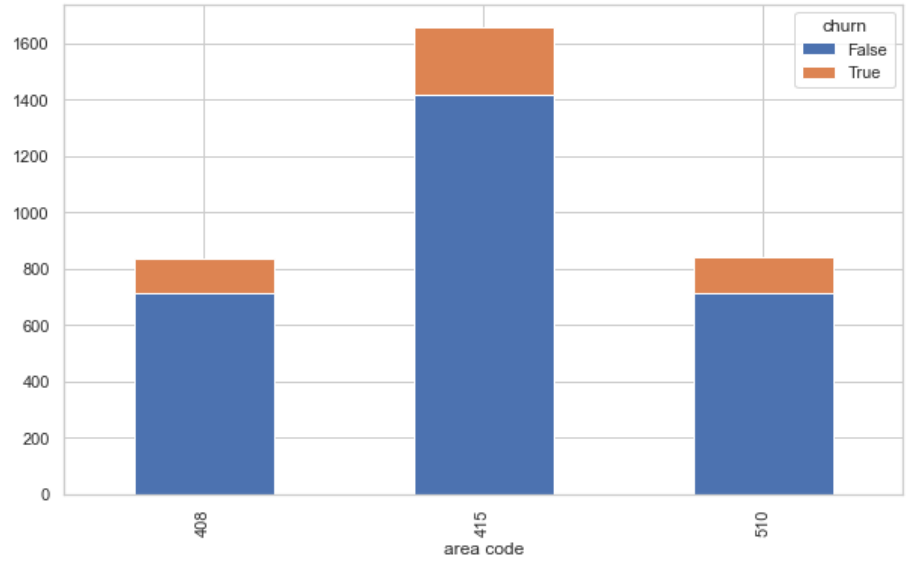


Some states have a smaller proportion of customer churn, such as AK, HI, IA, and LA, whereas other states have a larger proportion of customer churn, such as MD, MI, NJ, TX, and WA. This indicates that `state` will be a feature to include in future analyses for any insight on predicting churn.

Churn By Area Code

In [13]:
```
df.groupby(["area code", "churn"]).size().unstack().plot(
    kind = 'bar', stacked = True, figsize = (10, 6));
```

executed in 606ms, finished 14:53:39 2022-05-24

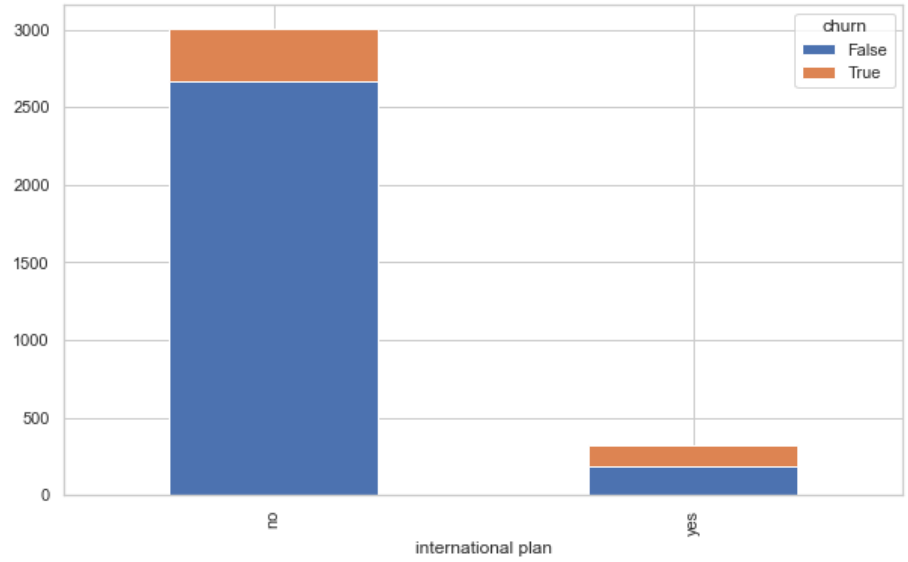


Churn By International Plan

proportion

In [14]:
```
df.groupby(["international plan", "churn"]).size().unstack().plot(
    kind = 'bar', stacked = True, figsize = (10, 6));
```

executed in 428ms, finished 14:53:40 2022-05-24

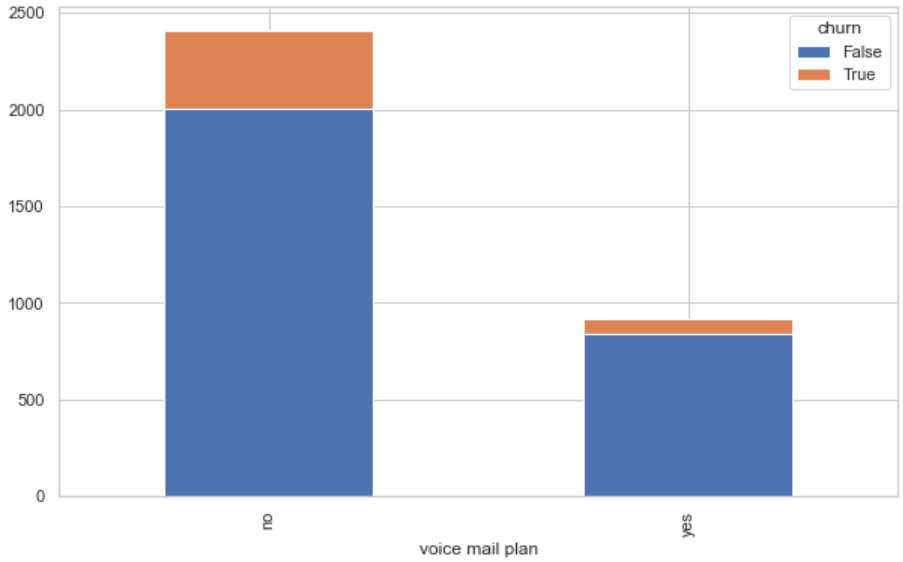


Customers that have an international plan have a much higher percentage of customer churn than do those without an international plan.

Churn By Voice Mail Plan

In [15]:
```
df.groupby(["voice mail plan", "churn"]).size().unstack().plot(
    kind = 'bar', stacked = True, figsize = (10, 6));
```

executed in 389ms, finished 14:53:40 2022-05-24

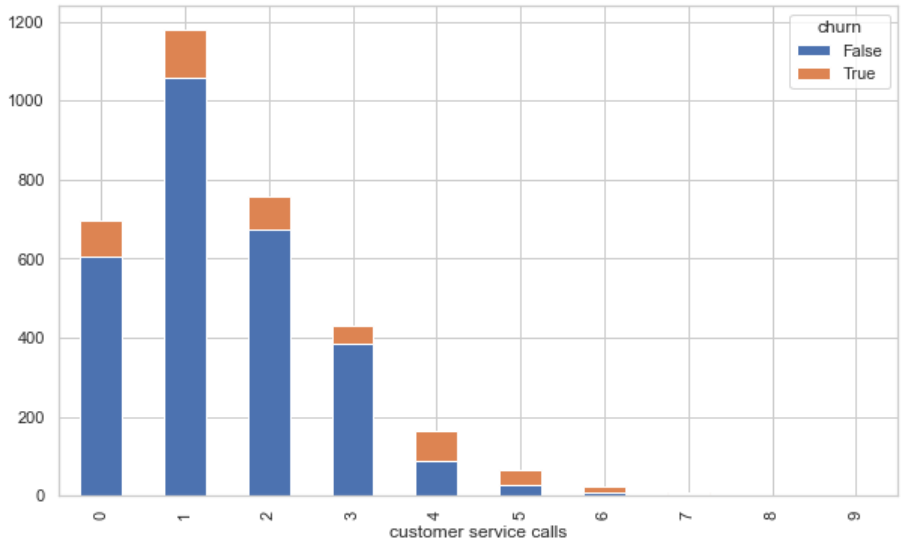


Customers without a voice mail plan also seem to have a higher proportion of customer than churn than those with voice mail plans.

Churn By Customer Service Calls

In [16]:
```
df.groupby(["customer service calls", "churn"]).size().unstack().plot(
    kind = 'bar', stacked = True, figsize = (10, 6));
```

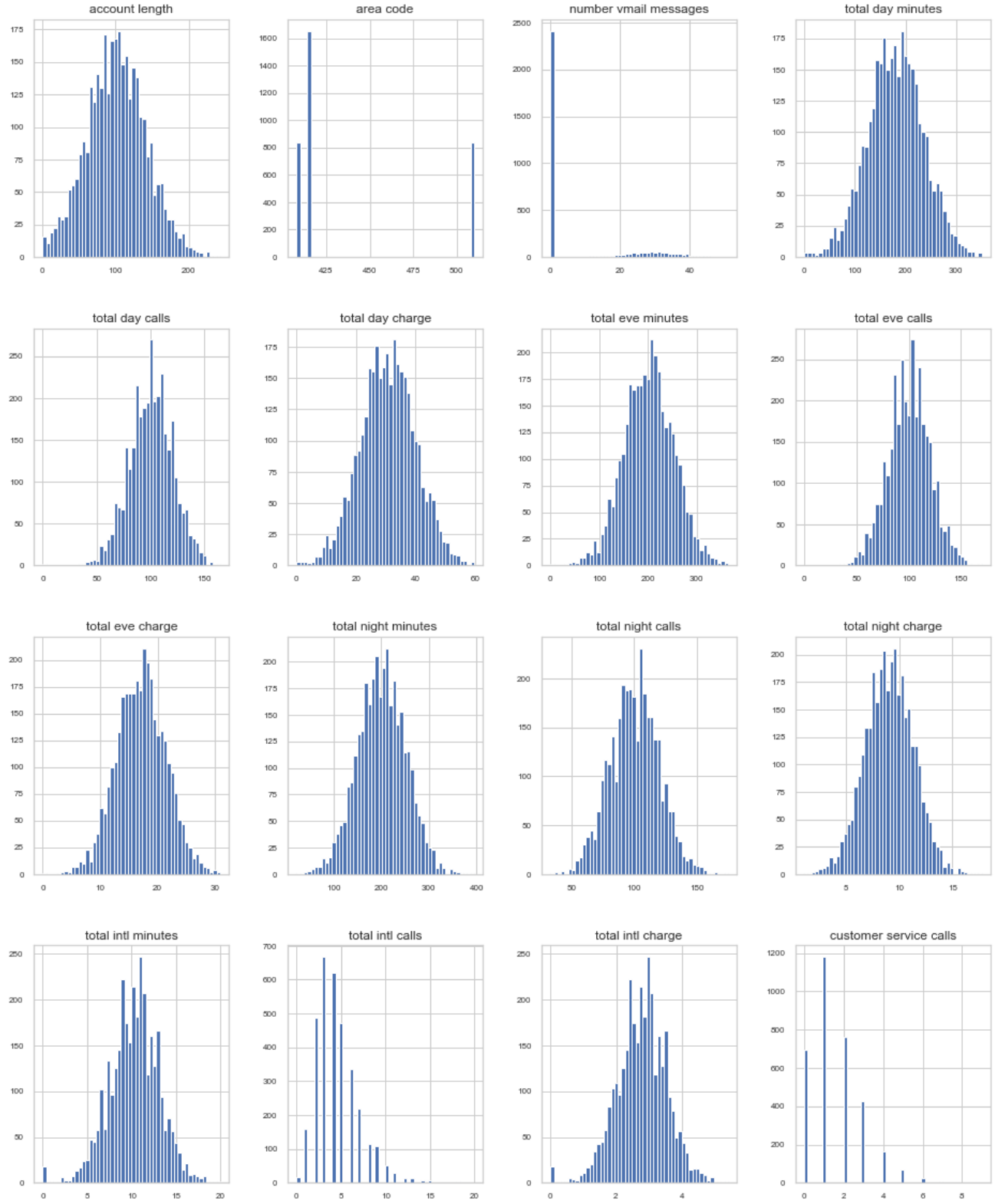
executed in 463ms, finished 14:53:40 2022-05-24



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

Modeling: Notebook demonstrates an iterative approach to model-building.

- Runs and interprets a simple, baseline model for comparison
- Introduces new models that improve on prior models and interprets their results

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 Baseline Model

5.2.1 Decision Tree Stump

In [246]:
```
# Set seed for reproducibility

SEED = 47
```
executed in 11ms, finished 16:30:12 2022-05-25

In [247]:
```
# 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
    clf_training_f1 = f1_score(y_train, training_preds)
    clf_test_f1 = f1_score(y_test, test_preds)
    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 49ms, finished 16:30:34 2022-05-25

In [223]:
```
# Initialize the decision tree stump

baseline = DecisionTreeClassifier(max_depth = 3,
                                  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 33ms, finished 20:35:20 2022-05-24

```
Training F1-Score: 0.62
Validation F1-Score: 0.56
F1-Score Delta: 0.062
```

5.3 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.

In [267]:
```
# 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(5).sort_values(ascending = True).plot(kind = 'barh')
```

executed in 6ms, finished 16:50:43 2022-05-25

5.3.1 Decision Tree Classifier

Decision Trees (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.

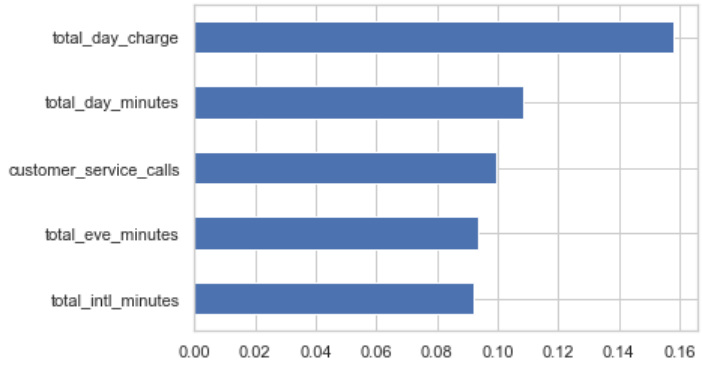
In [224]:
```
# Initialize decision tree classifier

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

print_metrics(dt)
```

executed in 66ms, finished 20:35:26 2022-05-24

```
Training F1-Score: 1.0
Validation F1-Score: 0.7
F1-Score Delta: 0.3
```

In [266]:
```
# Fit the data to the classifier

dt = DecisionTreeClassifier().fit(X_train_scaled, y_train)
plot_feature_importances(dt)
```

executed in 224ms, finished 16:49:59 2022-05-25



5.3.2 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 [268]:
```
# 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 246ms, finished 17:18:53 2022-05-25

```
Training F1-Score: 0.49
Validation F1-Score: 0.46
F1-Score Delta: 0.029
```

5.3.3 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 [226]:
```
# Initialize the KNN classifier and fit the data

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

print_metrics(knn)
```

executed in 352ms, finished 20:35:40 2022-05-24

```
Training F1-Score: 0.67
Validation F1-Score: 0.45
F1-Score Delta: 0.22
```

5.3.4 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.

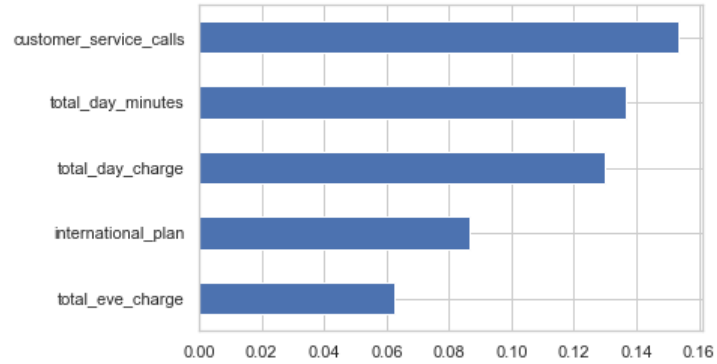
In [227]:
```
# Initialize the random forest classifier and fit the data

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

print_metrics(rf)
```

executed in 775ms, finished 20:35:43 2022-05-24

```
Training F1-Score: 1.0
Validation F1-Score: 0.76
F1-Score Delta: 0.24
```

In [269]:
```
plot_feature_importances(rf)
```

executed in 1.67s, finished 17:40:02 2022-05-25



5.3.5 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.

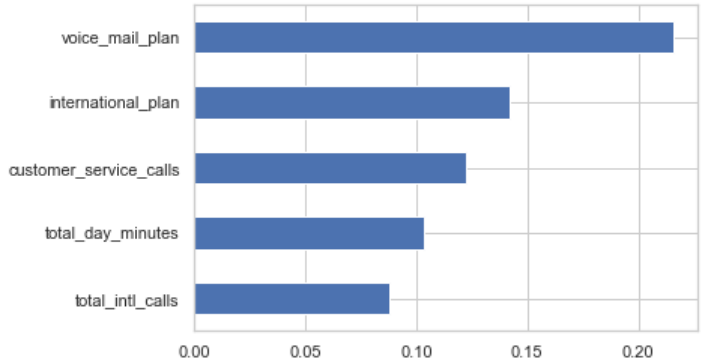
In [228]:
```
# Instantiate XGBClassifier and and fit the data
xgb = XGBClassifier(seed = SEED).fit(X_train_scaled, y_train)

print_metrics(xgb)
```

executed in 439ms, finished 20:35:50 2022-05-24

```
Training F1-Score: 1.0
Validation F1-Score: 0.8
F1-Score Delta: 0.2
```

In [271]:
```
xgb_clf = XGBClassifier().fit(X_train_scaled, y_train)
plot_feature_importances(xgb_clf)
```

executed in 517ms, finished 17:40:23 2022-05-25



5.3.6 Summary Results Table

In [291]:
```
# 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))
    row.append(f1_score(y_test, y_pred))
    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 2.48s, finished 17:06:57 2022-05-26

In [292]:
```
# Display the results table

results_table
```

executed in 39ms, finished 17:06:58 2022-05-26

Out[292]:

	Training F1	Validation F1	Delta
Decision Tree	1.00	0.69	0.31
Logistic Regression	0.49	0.46	0.03
KNN	0.67	0.45	0.22
Random Forest	1.00	0.76	0.24
XGBoost	1.00	0.80	0.20

5.4 Preparing for Model Tuning

- Explicitly justifies model changes based on the results of prior models and the problem context
- Explicitly describes any improvements found from running new models

Following the first set of vanilla model scores, there is certainly room for improvement in our predictive modeling. While the vanilla XGBoost model had an F1 score of 0.8 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.

5.4.1 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.

In [275]:
```
# Write a function to plot the feature importances, 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(5).sort_values(ascending = True).plot(kind = 'barh')
```

executed in 31ms, finished 17:47:33 2022-05-25

In [276]:
```
# 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 19ms, finished 17:47:59 2022-05-25

5.4.2 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.

5.4.3 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.

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

In [293]:
```
# 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
    clf_training_f1 = f1_score(y_train, training_preds)
    clf_test_f1 = f1_score(y_test, test_preds)
    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 23ms, finished 17:11:08 2022-05-26

5.5 Model Tuning

5.5.1 Decision Tree Classifier

In [284]:
```
# 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

when instantiating model, add class_weights parameter

In [159]:
```
# Initialize the decision tree classifier, run the GridSearch, and fit the data to the model

dt = DecisionTreeClassifier()

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)
```

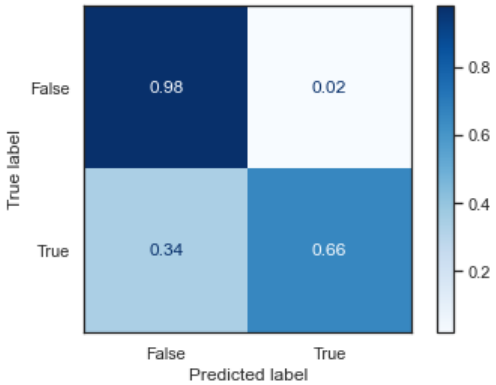
executed in 5.37s, finished 19:08:05 2022-05-24

```
Grid Search found the following optimal parameters:
criterion: 'entropy'
max_depth: 6
min_samples_leaf: 3
min_samples_split: 2
```

In [294]:
```
print_gs_metrics(dt_gs_clf)
```

executed in 1.18s, finished 17:11:29 2022-05-26

```
Training F1-Score: 0.87
Validation F1-Score: 0.74
F1-Score Delta: 0.13

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

Training Classification Report
              precision    recall  f1-score   support

       False       0.97      0.99      0.98      2137
        True       0.93      0.83      0.87       362

    accuracy                           0.97      2499
   macro avg       0.95      0.91      0.93      2499
weighted avg       0.96      0.97      0.96      2499


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.94      0.98      0.96       713
        True       0.85      0.66      0.74       121

    accuracy                           0.93       834
   macro avg       0.90      0.82      0.85       834
weighted avg       0.93      0.93      0.93       834


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

Confusion Matrix
```

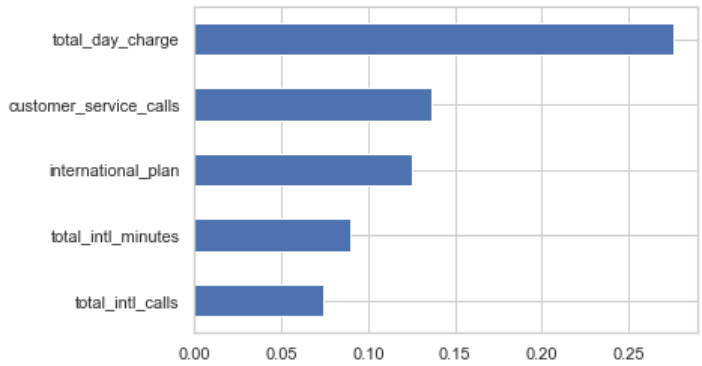


5.5.1.1 Results Summary

With this iterative model of the decision tree classifer, the F1-score on the training data is 0.87 whereas the F1-score on the test data is 0.74. With a delta of 0.13, this model does seem to be overfitting the data, though its F1-score on the test data is improved from the baseline decision tree model (0.56) and the vanilla decision tree model (0.7).

In [279]:
```
sns.set_theme(style = "whitegrid")
plot_gs_feature_importances(dt_gs_clf)
```

executed in 226ms, finished 17:50:16 2022-05-25



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

5.5.2 Logistic Regression

In [163]:
```
# Identify parameters to search for the logistic regression model

lr_param_grid = {
    'C': [1e-1, 1e2, 1e4],
    'penalty': ['l2', 'l1', 'elasticnet'],
    'solver': ['lbfgs', 'liblinear', 'sag', 'saga'],
    'max_iter': [1e-1, 1e2, 1e4]
}
```

executed in 20ms, finished 19:14:13 2022-05-24

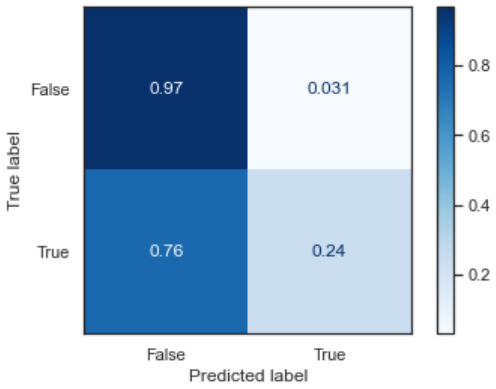
In [164]:
```
# Initialize the logistic regression, run the GridSearch, and fit the data to the model

lr = LogisticRegression()

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 4m 26s, finished 19:18:43 2022-05-24

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

In [295]:
```
print_gs_metrics(lr_gs_clf)
```
executed in 1.31s, finished 17:13:32 2022-05-26

```
Training F1-Score: 0.42
Validation F1-Score: 0.34
F1-Score Delta: 0.081

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

Training Classification Report
              precision    recall  f1-score   support

       False       0.89      0.97      0.93      2137
        True       0.63      0.31      0.42       362

    accuracy                           0.87      2499
   macro avg       0.76      0.64      0.67      2499
weighted avg       0.86      0.87      0.86      2499


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.88      0.97      0.92       713
        True       0.57      0.24      0.34       121

    accuracy                           0.86       834
   macro avg       0.73      0.60      0.63       834
weighted avg       0.84      0.86      0.84       834


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

Confusion Matrix
```



5.5.2.1 Results Summary

With this iterative model of the logistic regression model, the F1-score on the training data is 0.42 whereas the F1-score on the test data is 0.34. With a delta of 0.08, this model does not seem to be overfitting the data; however, its F1-score on the test data is quite low. It is actually lower than both the baseline decision tree model (0.56) and the vanilla logistic regression model (0.46).

5.5.3 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 [202]:
```
# 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)
```

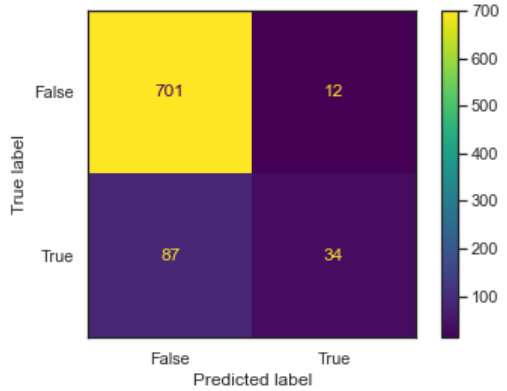
executed in 5.08s, finished 20:13:22 2022-05-24

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

In [234]:
```
print_gs_metrics(knn_gs_clf)
```

executed in 517ms, finished 20:36:47 2022-05-24

```
Training F1-Score: 0.76
Validation F1-Score: 0.41
F1-Score Delta: 0.35

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

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
```



5.5.3.1 Results Summary

With this iterative model of the KNN classifier, the F1-score on the training data is 0.76 whereas the F1-score on the test data is 0.41. With a delta of 0.35, this model does seem to be significantly overfitting the data; in fact, its F1-score on the test data (0.41) is quite low, lower than the baseline decision tree model (0.56) and not all that improved from the vanilla KNN classifier (0.45).

5.5.4 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 [204]:
```
# Initialize the random forest classifier, run the GridSearch, and fit the data to the model

rf = RandomForestClassifier()

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)
```

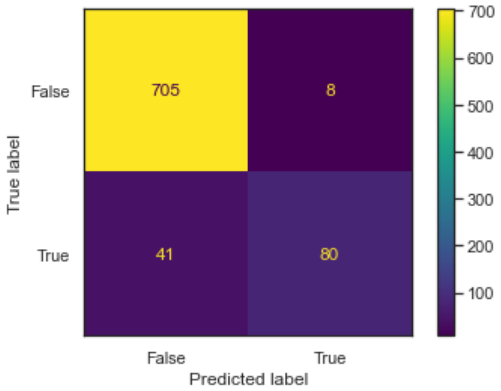
executed in 1m 53.3s, finished 20:15:54 2022-05-24

```
Grid Search found the following optimal parameters:
criterion: 'gini'
max_depth: None
min_samples_leaf: 1
min_samples_split: 2
```

In [235]:
```
print_gs_metrics(rf_gs_clf)
```

executed in 292ms, finished 20:36:52 2022-05-24

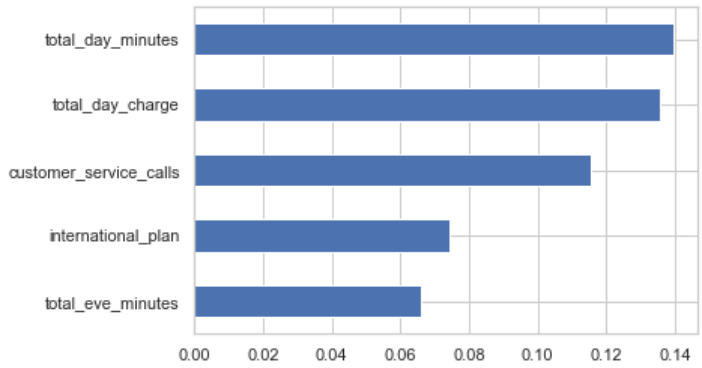
```
Training F1-Score: 1.0
Validation F1-Score: 0.77
F1-Score Delta: 0.23

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

Training Classification Report
              precision    recall  f1-score   support

       False       1.00      1.00      1.00      2137
        True       1.00      1.00      1.00       362

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


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.99      0.97       713
        True       0.91      0.66      0.77       121

    accuracy                           0.94       834
   macro avg       0.93      0.82      0.87       834
weighted avg       0.94      0.94      0.94       834


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

Confusion Matrix
```



5.5.4.1 Results Summary

With this iterative model of the random forest classifier, the F1-score on the training data is 0.77 whereas the F1-score on the test data is 1.0. With a delta of 0.23, this model does seem to be significantly overfitting the data; though, its F1-score on the test data (0.77) is good, higher than the baseline decision tree model (0.56) and not much different from the vanilla random forest classifier (0.76).

In [280]:
```
plot_gs_feature_importances(rf_gs_clf)
```
executed in 243ms, finished 17:52:16 2022-05-25



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.

5.5.5 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

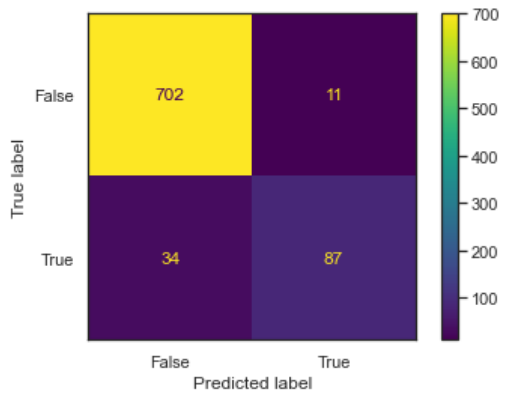
In [287]:
```
# Initialize the XGBoost classifier, run the GridSearch, and fit the data to the model

xgb_clf = XGBClassifier()

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 16.5s, finished 19:36:54 2022-05-25

```
Grid Search found the following optimal parameters:
criterion: 'gini'
max_depth: None
min_samples_leaf: 1
min_samples_split: 2
```

In [236]:
```
print_gs_metrics(xgb_gs_clf)
```

executed in 439ms, finished 20:36:59 2022-05-24

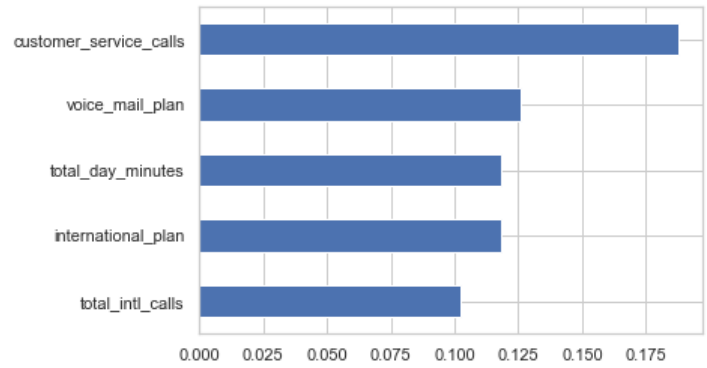
```
Training F1-Score: 0.93
Validation F1-Score: 0.79
F1-Score Delta: 0.14

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

Training Classification Report
              precision    recall  f1-score   support

       False       0.98      1.00      0.99      2137
        True       1.00      0.88      0.93       362

    accuracy                           0.98      2499
   macro avg       0.99      0.94      0.96      2499
weighted avg       0.98      0.98      0.98      2499


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.98      0.97       713
        True       0.89      0.72      0.79       121

    accuracy                           0.95       834
   macro avg       0.92      0.85      0.88       834
weighted avg       0.94      0.95      0.94       834


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

Confusion Matrix
```



5.5.5.1 Results Summary

With this iterative model of the XGBoost classifier, the F1-score on the training data is 0.93 whereas the F1-score on the test data is 0.79. With a delta of 0.14, this model does seem to be overfitting the data; though, its F1-score on the test data (0.79) is the best achieved yet, higher than the baseline decision tree model (0.56) and not much different from the vanilla XGBoost classifier (0.80).

In [281]:
```
plot_gs_feature_importances(xgb_gs_clf)
```
executed in 297ms, finished 17:52:57 2022-05-25



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).

5.5.6 Summary Results Table

In [282]:
```
# 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 = ["Logistic Regression", "KNN", "Decision Tree", "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))
    row.append(recall_score(y_test, y_pred))
    row.append(f1_score(y_test, y_pred))
    row = ["%.3f" % r for r in row]
    results_table.loc[name] = row
```
executed in 285ms, finished 17:54:02 2022-05-25

In [283]:
```
results_table
```
executed in 23ms, finished 17:54:05 2022-05-25

Out[283]:

	accuracy	precision	recall	f1
Logistic Regression	0.863	0.569	0.240	0.337
KNN	0.881	0.739	0.281	0.407
Decision Tree	0.934	0.851	0.661	0.744
Random Forest	0.941	0.909	0.661	0.766
XGBoost	0.946	0.888	0.719	0.795

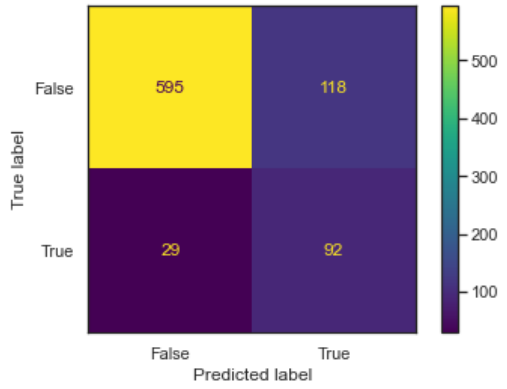
6 Evaluation

In [285]:
```
# Write a function to compare baseline and final/best model

def print_final_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
    clf_training_f1 = f1_score(y_train, training_preds)
    clf_test_f1 = f1_score(y_test, test_preds)
    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);
```
executed in 106ms, finished 19:24:52 2022-05-25

In [243]:
```
print('Baseline Model Metrics')
print_final_metrics(baseline)
```

executed in 273ms, finished 08:17:30 2022-05-25

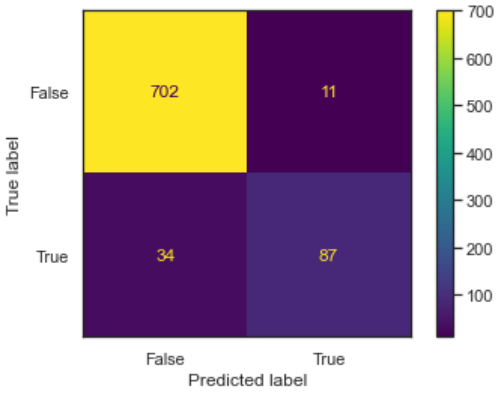
```
Baseline Model Metrics
Training F1-Score: 0.62
Validation F1-Score: 0.56
F1-Score Delta: 0.062

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

Training Classification Report
              precision    recall  f1-score   support

       False       0.97      0.85      0.91      2137
        True       0.49      0.85      0.62       362

    accuracy                           0.85      2499
   macro avg       0.73      0.85      0.76      2499
weighted avg       0.90      0.85      0.86      2499


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.83      0.89       713
        True       0.44      0.76      0.56       121

    accuracy                           0.82       834
   macro avg       0.70      0.80      0.72       834
weighted avg       0.88      0.82      0.84       834


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

Confusion Matrix
```



In [244]:
```
print('Final Model Metrics')
print_final_metrics(xgb_gs_clf)
```

executed in 281ms, finished 08:17:35 2022-05-25

```
Final Model Metrics
Training F1-Score: 0.93
Validation F1-Score: 0.79
F1-Score Delta: 0.14

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

Training Classification Report
              precision    recall  f1-score   support

       False       0.98      1.00      0.99      2137
        True       1.00      0.88      0.93       362

    accuracy                           0.98      2499
   macro avg       0.99      0.94      0.96      2499
weighted avg       0.98      0.98      0.98      2499


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

Testing Classification Report
              precision    recall  f1-score   support

       False       0.95      0.98      0.97       713
        True       0.89      0.72      0.79       121

    accuracy                           0.95       834
   macro avg       0.92      0.85      0.88       834
weighted avg       0.94      0.95      0.94       834


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

Confusion Matrix
```



- **Justifies choice of metrics using context of the real-world problem and consequences of errors**

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.

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.

- **Identifies one final model based on performance on the chosen metrics with validation data**
- **Evaluates the performance of the final model using holdout test data**

On the validation/test data, the XGBoost model has an F1 score of 0.79, which is the highest F1 score produced from the models run in this analysis.

- **Discusses implications of the final model evaluation for solving the real-world problem**

In [ ]:

- Create visuals to be used in presentation
- Recommendations (incl feature importance)

7 Conclusion Section

Future Work - setting yourself up for contract renewal, needing more money, etc.