Final Project Submission

- Student name: Kamile Yagci
- Student pace: self paced
- Scheduled project review date/time: Fri Jan 7, 2022 10:30am – 11:15am (CST)
- Instructor name: Claude Fried
- Blog URL: [https://kamileyagci.github.io/ (https://kamileyagci.github.io/)](https://kamileyagci.github.io/)

SyriaTel Customer Churn Study

Overview

In this study, I will analyze the 'SyriaTel Customer Churn' data. The SyriaTel is a telecommunication company. The purpose of the study is to predict whether a customer will ("soon") stop doing business with SyriaTel.

Business Problem

The telecommincation company, SyriaTel, hired me to analyze the Chustomer Churn data. The company wants to understand the customer's decision to discontinue their business with SyriaTel. The results of the analysis will be used make business decisions for improving the company finances.

This study will

- Search for the predictable pattern for customer decision on stop or continue doing business with SyriaTel
- Choose a model which will best identify the customers who will stop doing business with SyriaTel

Data

Load

I use SyriaTel Customer Churn data for this study. The data file is downloaded from Kaggle.

The file name is 'bigml_59c28831336c6604c800002a.csv'.

In [526]:
```
# Import base libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')
```

In [351]:
```
# Import data
df = pd.read_csv('bigml_59c28831336c6604c800002a.csv')
df.head()
```

Out[351]:

	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
0	KS	128	415	382-4657	no	yes	25	265.1	110	45.07	...	99
1	OH	107	415	371-7191	no	yes	26	161.6	123	27.47	...	103
2	NJ	137	415	358-1921	no	no	0	243.4	114	41.38	...	110
3	OH	84	408	375-9999	yes	no	0	299.4	71	50.90	...	88
4	OK	75	415	330-6626	yes	no	0	166.7	113	28.34	...	122

5 rows × 21 columns

Scrub / Explore

I will first look at the data closely.

In [352]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   phone number            3333 non-null   object 
 4   international plan      3333 non-null   object 
 5   voice mail plan         3333 non-null   object 
 6   number vmail messages   3333 non-null   int64  
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64  
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64  
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64  
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64  
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64  
 20  churn                   3333 non-null   bool   
dtypes: bool(1), float64(8), int64(8), object(4)
memory usage: 524.2+ KB
```

In [353]:
```
df.isna().sum()
```

Out[353]:
```
state                     0
account length            0
area code                 0
phone number              0
international plan        0
voice mail plan           0
number vmail messages     0
total day minutes         0
total day calls           0
total day charge          0
total eve minutes         0
total eve calls           0
total eve charge          0
total night minutes       0
total night calls         0
total night charge        0
total intl minutes        0
total intl calls          0
total intl charge         0
customer service calls    0
churn                     0
dtype: int64
```

I will remove the column 'phone number' from dataset. Most digits in the phone number is random, and it will not have much use in modeling. This variable will also be a problem in dummy variable creation, because all values will be unique.

In [354]:
```
df = df.drop('phone number', axis=1)
```

I will convert 'international plan', 'voice mail plan', and 'churn' variables to binary.

In [355]:
```
# Convert to binary
df['international plan'] = df['international plan'].map({'yes':1 ,'no':0})
df['voice mail plan'] = df['voice mail plan'].map({'yes':1 ,'no':0})
df['churn'] = df['churn'].map({True:1 ,False:0})
df.head()
```

Out[355]:

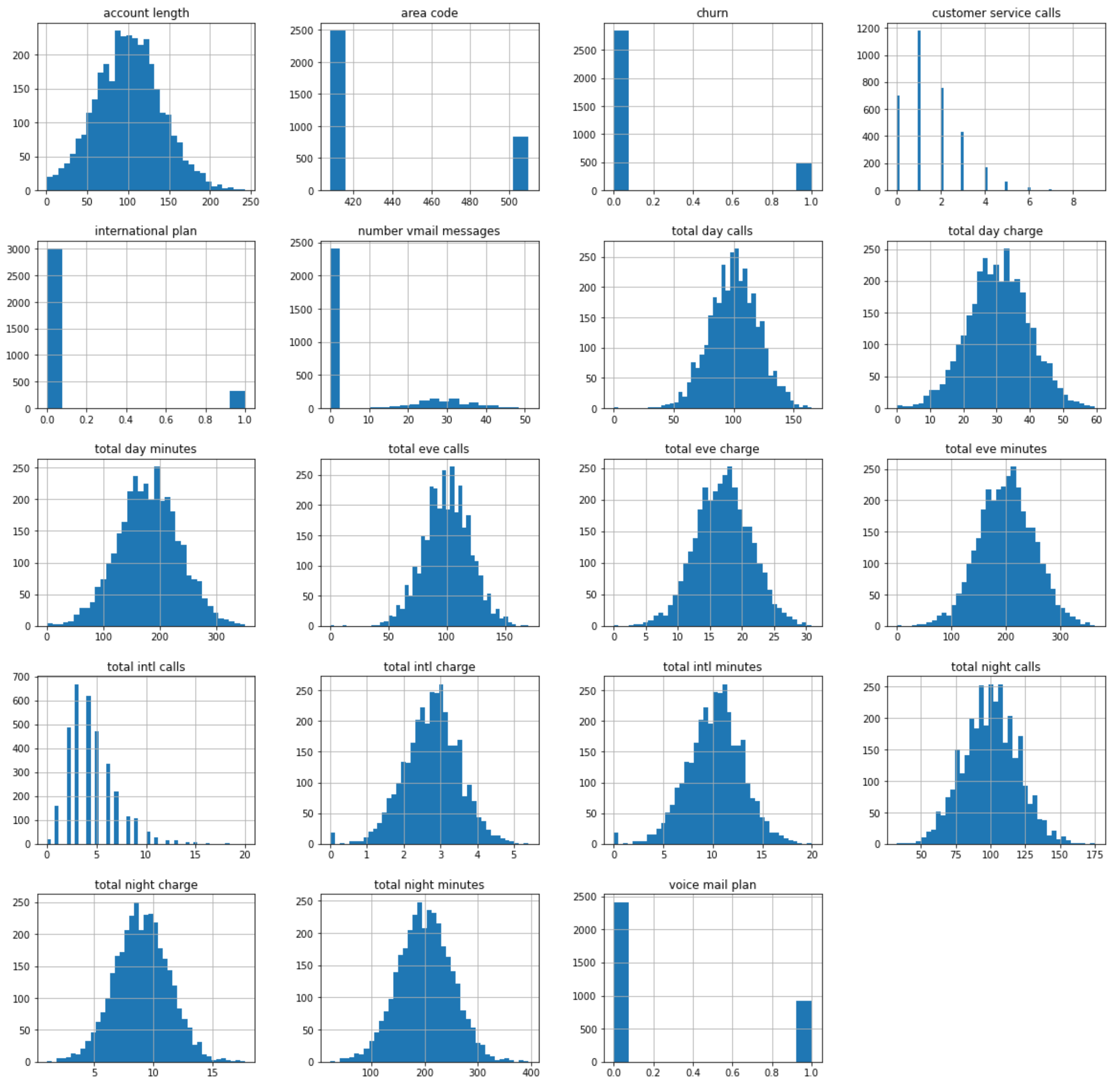
	state	account length	area code	international plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	ch
0	KS	128	415	0	1	25	265.1	110	45.07	197.4	99	
1	OH	107	415	0	1	26	161.6	123	27.47	195.5	103	
2	NJ	137	415	0	0	0	243.4	114	41.38	121.2	110	
3	OH	84	408	1	0	0	299.4	71	50.90	61.9	88	
4	OK	75	415	1	0	0	166.7	113	28.34	148.3	122	

In [356]:
```
df.info()
```

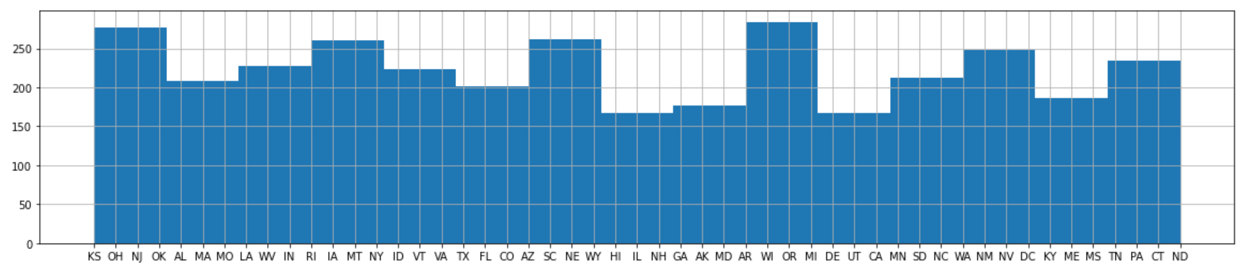
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype  
---  ------                  --------------  -----  
 0   state                   3333 non-null   object 
 1   account length          3333 non-null   int64  
 2   area code               3333 non-null   int64  
 3   international plan      3333 non-null   int64  
 4   voice mail plan         3333 non-null   int64  
 5   number vmail messages   3333 non-null   int64  
 6   total day minutes       3333 non-null   float64
 7   total day calls         3333 non-null   int64  
 8   total day charge        3333 non-null   float64
 9   total eve minutes       3333 non-null   float64
 10  total eve calls         3333 non-null   int64  
 11  total eve charge        3333 non-null   float64
 12  total night minutes     3333 non-null   float64
 13  total night calls       3333 non-null   int64  
 14  total night charge      3333 non-null   float64
 15  total intl minutes      3333 non-null   float64
 16  total intl calls        3333 non-null   int64  
 17  total intl charge       3333 non-null   float64
 18  customer service calls  3333 non-null   int64  
 19  churn                   3333 non-null   int64  
dtypes: float64(8), int64(11), object(1)
memory usage: 520.9+ KB
```

Let's see distributions for all varaiables.

In [357]:
```
df.hist(figsize=(20,20), bins='auto')
plt.savefig('images/histograms_All.png')
```



In [358]:
```
df['state'].hist(figsize=(20,4), bins='auto')
plt.savefig('images/histogram_state.png')
```



Now, the binary variables have type int64. I will change the dtype to object for these variables, to make them available for dummy variable creation.

The variable 'area code' is also dtype int64, however it is a categorical variable. I will also change it to object.

In [359]:
```
df = df.astype({'international plan': 'object'})
df = df.astype({'voice mail plan': 'object'})
df = df.astype({'area code': 'object'})
```

In [360]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 20 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   object 
 3   international plan      3333 non-null   object 
 4   voice mail plan         3333 non-null   object 
 5   number vmail messages   3333 non-null   int64  
 6   total day minutes       3333 non-null   float64
 7   total day calls         3333 non-null   int64  
 8   total day charge        3333 non-null   float64
 9   total eve minutes       3333 non-null   float64
 10  total eve calls         3333 non-null   int64  
 11  total eve charge        3333 non-null   float64
 12  total night minutes     3333 non-null   float64
 13  total night calls       3333 non-null   int64  
 14  total night charge      3333 non-null   float64
 15  total intl minutes      3333 non-null   float64
 16  total intl calls        3333 non-null   int64  
 17  total intl charge       3333 non-null   float64
 18  customer service calls  3333 non-null   int64  
 19  churn                   3333 non-null   int64  
dtypes: float64(8), int64(8), object(4)
memory usage: 520.9+ KB
```

In [361]:
```
df.describe()
```

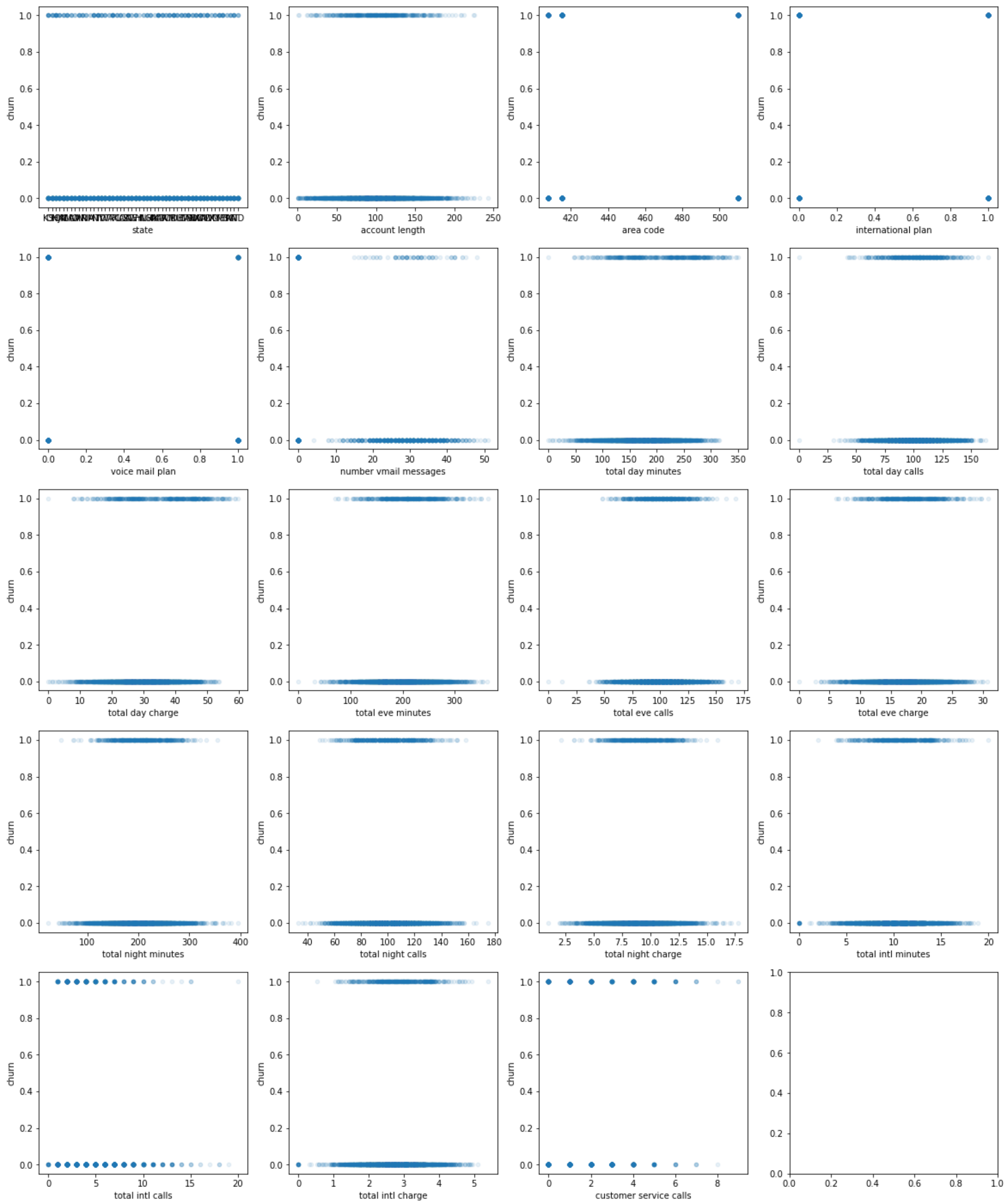
Out[361]:

	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000
mean	101.064806	8.099010	179.775098	100.435644	30.562307	200.980348	100.114311
std	39.822106	13.688365	54.467389	20.069084	9.259435	50.713844	19.922625
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	74.000000	0.000000	143.700000	87.000000	24.430000	166.600000	87.000000
50%	101.000000	0.000000	179.400000	101.000000	30.500000	201.400000	100.000000
75%	127.000000	20.000000	216.400000	114.000000	36.790000	235.300000	114.000000
max	243.000000	51.000000	350.800000	165.000000	59.640000	363.700000	170.000000

The traget variable for this study is 'churn'. Let's check the scatter matrixes.

In [362]:

```
fig, axes = plt.subplots(5, 4, figsize=(20, 25))

for ax, col in zip(axes.flatten(), df.columns[:-1]):
    df.plot.scatter(col, 'churn', alpha=0.1, ax=ax)

plt.savefig('images/scatters_All.png')
```



It is hard to recognize any patterns for 'churn' in these plots.

We will now look at the models to derive patterns and predictions.

Model

In this study, we are trying to predict customer's decision on stopping the business with the company. The prediction will be True (1) or False (0). Therefore we will use binary classification model.

Pre-process

The target variable is 'churn': activity of customers leaving the company and discarding the services offered

The rest of the variables in the dataset will be predictors. I will also create dummy variables from categorical variables.

Let's create the target data series (y) and predictor dataframe (X).

In [363]:
```
# Assign target and predictor
y = df['churn']
X = df.drop('churn', axis=1)

X = pd.get_dummies(X)
X.head()
```

Out[363]:

	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	...	state_WI
0	128	25	265.1	110	45.07	197.4	99	16.78	244.7	91	...	0
1	107	26	161.6	123	27.47	195.5	103	16.62	254.4	103	...	0
2	137	0	243.4	114	41.38	121.2	110	10.30	162.6	104	...	0
3	84	0	299.4	71	50.90	61.9	88	5.26	196.9	89	...	0
4	75	0	166.7	113	28.34	148.3	122	12.61	186.9	121	...	0

5 rows × 73 columns

Next, I will seperate the data into train and test splits. I will allocate 25% of the data for testing. I will also assign a random state for repeatability.

In [364]:

```
# Sepearate data into train and test splist
from sklearn.model_selection import train_test_split

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

print('X_train shape = ', X_train.shape)
print('y_train shape = ', y_train.shape)
print('X_test shape = ', X_test.shape)
print('y_test shape = ', y_test.shape)
```

```
X_train shape =  (2499, 73)
y_train shape =  (2499,)
X_test shape =  (834, 73)
y_test shape =  (834,)
```

The data values have different ranges, so I need to normalize/scale each variable in train and test data (X) before modeling.

In [365]:

```
# Scale/Normalize the predictor variables
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert to Dataframe
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)
X_train_scaled.head()
```

Out[365]:

	account length	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes
0	-1.404508	-0.584700	-1.883677	1.330852	-1.884170	1.037727	0.401340	1.037905	1.069609
1	0.366388	-0.584700	0.294083	0.529165	0.293703	0.516178	0.401340	0.517286	2.214376
2	0.518179	1.685101	1.056392	-1.875896	1.056666	0.093407	0.849774	0.094283	-0.077125
3	2.010792	-0.584700	-0.679156	1.681590	-0.679320	-0.402459	0.650470	-0.403094	-0.322994
4	0.290493	-0.584700	0.484660	1.080325	0.484172	-0.718549	-0.296224	-0.719184	-1.186487

5 rows × 73 columns

Evaluation Metrics

In the next steps, I will use several classifiers to model the data. I will check their performance using the evaluation metrics:

precision:

- Number of True Positives / Number of Predicted Positives
- How precise our predictions are?

recall:

- Nuber of True Positives / Number of Actual Total Positives
- What percentage of the classes we're interested in were actually captured by the model?

accuracy:

- (Number of True Positives + Number of True Negatives) / (Number of Total Observations)
- Out of all the predictions our model made, what percentage were correct?

f1-score:

- 2 * (Precision * Recall) / (Precision + Recall)
- Harmonic Mean of Precision and Recall.

Source: Flatiron Data Science Curriculum, Evaluation Metrics

Since my business problem is focusing on identfying the customers who stop doing business, I am interested mainly on the 'recall' metrics. However, when optimizing my model, I should also pay attention to the 'precision'. I want my predictions to be true, to be precise. The recall and precision are inversely proportional. Therefore, I choose to use the f1-score, Harmonic Mean of Precision and Recall, as the main metric for evaluating the performance of the model.

Logistic Regression

I start with Logistic Regression. I instantiate the model with default parameters and fit on training data. Then I will check the evaluation metrics both for training and testing data.

In [366]:
```
# Import, Instantiate a LogisticRegression and fit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_scaled, y_train)

# Predict
#y_train_pred = logreg.predict(X_train_scaled)
#y_test_pred = logreg.predict(X_test_scaled)
```

Out[366]:
```
LogisticRegression(random_state=42)
```

In [367]:
```python
#Print out the evaluation metrics for training and testing data
from sklearn.metrics import confusion_matrix, plot_confusion_matrix, classi

print('Training Data:\n', classification_report(y_train, logreg.predict(X_t
print('Testing Data:\n', classification_report(y_test, logreg.predict(X_tes
```

```
Training Data:
               precision    recall  f1-score   support

           0       0.89      0.97      0.93      2141
           1       0.64      0.27      0.37       358

    accuracy                           0.87      2499
   macro avg       0.76      0.62      0.65      2499
weighted avg       0.85      0.87      0.85      2499

Testing Data:
               precision    recall  f1-score   support

           0       0.88      0.97      0.92       709
           1       0.56      0.22      0.32       125

    accuracy                           0.86       834
   macro avg       0.72      0.60      0.62       834
weighted avg       0.83      0.86      0.83       834
```

My observations from the printed results:

- The metrics look similar for both training and testing data, just training is a bit better; so slight overfitting.
- The precision - recall - f1 scores are low (for churn=1), so the model prediction performance is not good.
- The high accuracy score is high, but misleading. It is caused by the imbalanced dataset.

Resampling

Class imbalance effects the performance of the classification model.

In [368]:
```python
print('Original whole data class distribution:')
print(y.value_counts())
print('Original whole data class distribution, normalized:')
print(y.value_counts(normalize=True))
```

```
Original whole data class distribution:
0    2850
1     483
Name: churn, dtype: int64
Original whole data class distribution, normalized:
0    0.855086
1    0.144914
Name: churn, dtype: float64
```

According to the dataset, 85.5% of the customers do continue with SyriaTel and 14.5% of customers stop business. If we make a prediction that, all customers will continue, then we will have 85.5% accuracy. This explains the high accuracy score of the model, despite the other low metric values.

I will use SMOTE to create a synthetic training sample to take care of imbalance.

In [369]:
```
# Import SMOTE, resample
from imblearn.over_sampling import SMOTE

smote = SMOTE()
X_train_scaled_resampled, y_train_resampled = smote.fit_resample(X_train_sc

print('Original training data class distribution:')
print(y_train.value_counts())
print('Synthetic training data class distribution:')
print(y_train_resampled.value_counts())
```

```
Original training data class distribution:
0    2141
1     358
Name: churn, dtype: int64
Synthetic training data class distribution:
1    2141
0    2141
Name: churn, dtype: int64
```

In [377]:
```
# New model after resampling
logreg = LogisticRegression(random_state=42)
logreg.fit(X_train_scaled_resampled, y_train_resampled)

print('Training Data:\n', classification_report(y_train_resampled, logreg.p
print('Testing Data:\n', classification_report(y_test, logreg.predict(X_tes
```

```
Training Data:
               precision    recall  f1-score   support

           0       0.81      0.78      0.79      2141
           1       0.79      0.81      0.80      2141

    accuracy                           0.80      4282
   macro avg       0.80      0.80      0.80      4282
weighted avg       0.80      0.80      0.80      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.95      0.78      0.85       709
           1       0.38      0.77      0.51       125

    accuracy                           0.78       834
   macro avg       0.66      0.77      0.68       834
weighted avg       0.86      0.78      0.80       834
```

- After resampling, the Logistic Regression Model performance is clearly improved.
- The performance in training data is better than test data. This is a sign of overfitting.

Parameter Tuning

I initially used the default paremeters for the Logistic Regression model. I will now apply parameter tuning with GridSearchCV. It will determine the best parameter combination for the given parameter grid.

In [371]:
```
print('Default parameters:')
logreg.get_params()
```

```
Default parameters:
```

Out[371]:
```
{'C': 1.0,
 'class_weight': None,
 'dual': False,
 'fit_intercept': True,
 'intercept_scaling': 1,
 'l1_ratio': None,
 'max_iter': 100,
 'multi_class': 'auto',
 'n_jobs': None,
 'penalty': 'l2',
 'random_state': 42,
 'solver': 'lbfgs',
 'tol': 0.0001,
 'verbose': 0,
 'warm_start': False}
```

In [378]:
```python
# Tuning Logistic Regression model with GridSearchCV
from sklearn.model_selection import GridSearchCV

logreg_param_grid = {
    'solver': ['lbfgs', 'liblinear'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000, 1e5, 1e10, 1e15, 1e20],
}

logreg_gs = GridSearchCV(logreg, logreg_param_grid, cv=5, scoring='f1')
#logreg_gs.fit(X_train_scaled, y_train)
logreg_gs.fit(X_train_scaled_resampled, y_train_resampled)

#score_logreg_gs = logreg_gs.score(X_test_scaled, y_test)
#print('f1-score for test data:', score_logreg_gs)

print('Parameter Tuning Results:\n')
print("Best Parameter Combination:", logreg_gs.best_params_)
print('Training Data:\n', classification_report(y_train_resampled, logreg_g
print('Testing Data:\n', classification_report(y_test, logreg_gs.predict(X_
```

```
Parameter Tuning Results:

Best Parameter Combination: {'C': 0.01, 'solver': 'liblinear'}
Training Data:
               precision    recall  f1-score   support

           0       0.81      0.76      0.79      2141
           1       0.78      0.83      0.80      2141

    accuracy                           0.79      4282
   macro avg       0.80      0.79      0.79      4282
weighted avg       0.80      0.79      0.79      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.95      0.76      0.85       709
           1       0.37      0.79      0.51       125

    accuracy                           0.77       834
   macro avg       0.66      0.78      0.68       834
weighted avg       0.87      0.77      0.80       834
```

- It looks like the parameter tuning, with the given parameter grid, didn't improve the performance much.
- Overfitting is observed.

K-Nearest Neighbors

In [384]:
```
# Import, Instantiate, fit KNeighborsClassifier,
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier()
#knn.fit(X_train_scaled, y_train) # #f1 = 0.15 on test data
knn.fit(X_train_scaled_resampled, y_train_resampled) #Resampled data with S

print('Training Data:\n', classification_report(y_train_resampled, knn.pred
print('Testing Data:\n', classification_report(y_test, knn.predict(X_test_s
```

```
Training Data:
               precision    recall  f1-score   support

           0       1.00      0.82      0.90      2141
           1       0.85      1.00      0.92      2141

    accuracy                           0.91      4282
   macro avg       0.92      0.91      0.91      4282
weighted avg       0.92      0.91      0.91      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.92      0.73      0.81       709
           1       0.29      0.62      0.39       125

    accuracy                           0.72       834
   macro avg       0.60      0.67      0.60       834
weighted avg       0.82      0.72      0.75       834
```

Observations:

- The fitting on resampled training data has a better performance. The f1-score for test data increased from 0.15 to 0.39. (The results for resampled data is not shown here, but tested).
- Overfitting observed.

Parameter Tuning

In [374]:
```
print('Default parameters:')
knn.get_params()
```

```
Default parameters:
```

Out[374]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

```
In [387]: # Tuning KNN model with GridSearchCV
          # Takes about 10 minutes om my PC

          knn_param_grid = {
              'n_neighbors': [3, 4, 5, 6, 7, 8],
              'p': [1, 2, 3, 4]
          }

          knn_gs = GridSearchCV(knn, knn_param_grid, cv=5, scoring='f1')
          #knn_gs.fit(X_train_scaled, y_train)
          knn_gs.fit(X_train_scaled_resampled, y_train_resampled) #Lower performance,

          #score_knn_gs = knn_gs.score(X_test_scaled, y_test)
          #print('f1-score for test data:', score_knn_gs)

          print('Parameter Tuning Results:\n')
          print("Best Parameter Combination:", knn_gs.best_params_)
          print('Training Data:\n', classification_report(y_train_resampled, knn_gs.p
          print('Testing Data:\n', classification_report(y_test, knn_gs.predict(X_tes
```

```
Parameter Tuning Results:

Best Parameter Combination: {'n_neighbors': 4, 'p': 1}
Training Data:
               precision    recall  f1-score   support

           0       0.99      0.94      0.96      2141
           1       0.94      0.99      0.97      2141

    accuracy                           0.97      4282
   macro avg       0.97      0.97      0.97      4282
weighted avg       0.97      0.97      0.97      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.89      0.85      0.87       709
           1       0.32      0.39      0.35       125

    accuracy                           0.79       834
   macro avg       0.61      0.62      0.61       834
weighted avg       0.80      0.79      0.79       834
```

- Parameter tuning, with the given parameter ranges, didn't improve the KNN model performance.
- Overfitting observed.

Decision Tress

I will firstly use DecisionTreeClassifier with default parameters, then apply GridSearchCV to find the optimum parameteres.

In [388]:
```python
# Import, Instantiate, fit DecisionTreeClassifier,
from sklearn.tree import DecisionTreeClassifier

dt = DecisionTreeClassifier(random_state=42)
#dt.fit(X_train_scaled, y_train)
dt.fit(X_train_scaled_resampled, y_train_resampled)

print('Training Data:\n', classification_report(y_train_resampled, dt.predi
print('Testing Data:\n', classification_report(y_test, dt.predict(X_test_sc
```

```
Training Data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.94      0.93      0.93       709
           1       0.62      0.67      0.64       125

    accuracy                           0.89       834
   macro avg       0.78      0.80      0.79       834
weighted avg       0.89      0.89      0.89       834
```

Parameter Tuning

In [389]:
```python
print('Default parameters:')
dt.get_params()
```

```
Default parameters:
```

Out[389]:
```
{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'random_state': 42,
 'splitter': 'best'}
```

In [398]:

```python
# Tuning Decision Trees model with GridSearchCV
# Takes more than 10 minutes om my PC

dt_param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, 6],
    'min_samples_split': [2, 3, 4, 5, 6],
    #'min_samples_leaf': [1, 2, 3, 4, 5, 6]
}

dt_gs = GridSearchCV(dt, dt_param_grid, cv=5, scoring='f1')
#dt_gs.fit(X_train_scaled, y_train)
dt_gs.fit(X_train_scaled_resampled, y_train_resampled)

#score_dt_gs = dt_gs.score(X_test_scaled, y_test)
#print('f1-score for test data:', score_dt_gs)

print('Parameter Tuning Results:\n')
print("Best Parameter Combination:", dt_gs.best_params_)
print('Training Data:\n', classification_report(y_train_resampled, dt_gs.pr
print('Testing Data:\n', classification_report(y_test, dt_gs.predict(X_test
```

```
Parameter Tuning Results:

Best Parameter Combination: {'criterion': 'gini', 'max_depth': 6, 'min_sa
mples_split': 2}
Training Data:
               precision    recall  f1-score   support

           0       0.88      0.94      0.91      2141
           1       0.94      0.87      0.90      2141

    accuracy                           0.91      4282
   macro avg       0.91      0.91      0.91      4282
weighted avg       0.91      0.91      0.91      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.95      0.94      0.94       709
           1       0.68      0.70      0.69       125

    accuracy                           0.91       834
   macro avg       0.81      0.82      0.82       834
weighted avg       0.91      0.91      0.91       834
```

- The parameter tuning improved the Decision Trees performance a little.
- Overfitting observed.

Random Forests

Let's try an ensemble method Random Forests, which uses DecisionTreeClassifier.

In [392]:
```python
# Import, Instantiate, fit RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(random_state=42)
rf.fit(X_train_scaled, y_train)
rf.fit(X_train_scaled_resampled, y_train_resampled) #No change in f1 score

print('Training Data:\n', classification_report(y_train_resampled, rf.predi
print('Testing Data:\n', classification_report(y_test, rf.predict(X_test_sc
```

```
Training Data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.95      0.97      0.96       709
           1       0.79      0.71      0.75       125

    accuracy                           0.93       834
   macro avg       0.87      0.84      0.85       834
weighted avg       0.93      0.93      0.93       834
```

Parameter Tuning

In [393]:
```python
print('Default parameters:')
rf.get_params()
```

```
Default parameters:
```

Out[393]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

```
In [399]: # Tuning Random Forest model with GridSearchCV

rf_param_grid = {
    'n_estimators': [10, 30, 100],
    'criterion': ['gini', 'entropy'],
    'max_depth': [2, 3, 4, 5, 6],
    'min_samples_split': [2, 3, 4, 5, 6],
    #'min_samples_leaf': [3, 6],
    'max_features': [4, 5, 6, 7, 8]
}

rf_gs = GridSearchCV(rf, rf_param_grid, cv=5, scoring='f1')
rf_gs.fit(X_train_scaled_resampled, y_train_resampled)

#score_rf_gs = rf_gs.score(X_test_scaled, y_test)
#print('f1-score on test data:', score_rf_gs)

print('Parameter Tuning Results:\n')
print("Best Parameter Combination:", rf_gs.best_params_)
print('Training Data:\n', classification_report(y_train_resampled, rf_gs.pr
print('Testing Data:\n', classification_report(y_test, rf_gs.predict(X_test
```

```
Parameter Tuning Results:

Best Parameter Combination: {'criterion': 'gini', 'max_depth': 6, 'max_fe
atures': 8, 'min_samples_split': 6, 'n_estimators': 10}
Training Data:
               precision    recall  f1-score   support

           0       0.90      0.92      0.91      2141
           1       0.92      0.89      0.90      2141

    accuracy                           0.91      4282
   macro avg       0.91      0.91      0.91      4282
weighted avg       0.91      0.91      0.91      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.94      0.92      0.93       709
           1       0.61      0.69      0.65       125

    accuracy                           0.89       834
   macro avg       0.78      0.81      0.79       834
weighted avg       0.89      0.89      0.89       834
```

- The paremeter tuning didn't improve the performance of Random Forest model.
- Overfitting observed.

XGBoost

In [395]:
```python
# Import, Instantiate, fit XGBClassifier
from xgboost import XGBClassifier
import xgboost as xgb

xgb = XGBClassifier(random_state=42, eval_metric='logloss') #'logloss' is d
#xgb.fit(X_train_scaled, y_train)
xgb.fit(X_train_scaled_resampled, y_train_resampled)

print('Training Data:\n', classification_report(y_train_resampled, xgb.pred
print('Testing Data:\n', classification_report(y_test, xgb.predict(X_test_s
```

```
Training Data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00      2141
           1       1.00      1.00      1.00      2141

    accuracy                           1.00      4282
   macro avg       1.00      1.00      1.00      4282
weighted avg       1.00      1.00      1.00      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.95      0.99      0.97       709
           1       0.90      0.73      0.81       125

    accuracy                           0.95       834
   macro avg       0.93      0.86      0.89       834
weighted avg       0.95      0.95      0.94       834
```

Parameter Tuning

In [396]:
```python
print('Default parameters:')
xgb.get_params()
```

```
Default parameters:
```

Out[396]:
```
{'objective': 'binary:logistic',
 'use_label_encoder': True,
 'base_score': 0.5,
 'booster': 'gbtree',
 'colsample_bylevel': 1,
 'colsample_bynode': 1,
 'colsample_bytree': 1,
 'gamma': 0,
 'gpu_id': -1,
 'importance_type': 'gain',
 'interaction_constraints': '',
 'learning_rate': 0.300000012,
 'max_delta_step': 0,
 'max_depth': 6,
 'min_child_weight': 1,
 'missing': nan,
 'monotone_constraints': '()',
 'n_estimators': 100,
 'n_jobs': 4,
 'num_parallel_tree': 1,
 'random_state': 42,
 'reg_alpha': 0,
 'reg_lambda': 1,
 'scale_pos_weight': 1,
 'subsample': 1,
 'tree_method': 'exact',
 'validate_parameters': 1,
 'verbosity': None,
 'eval_metric': 'logloss'}
```

In [400]:
```python
# Tuning XGBClassifier with GridSearchCV
# Takes more than 10 minutes om my PC

from sklearn.model_selection import GridSearchCV

xgb_param_grid = {
    'learning_rate': [0.1, 0.2],
    'max_depth': [2, 3, 4, 5, 6],
    'min_child_weight': [1, 2],
    'subsample': [0.5, 0.7],
    'n_estimators': [30, 100],
}

xgb_gs = GridSearchCV(xgb, xgb_param_grid, cv=5, scoring='f1')
xgb_gs.fit(X_train_scaled_resampled, y_train_resampled)

#score_xgb_gs = xgb_gs.score(X_test_scaled, y_test)
#print('f1-score on test data:', score_xgb_gs)

print('Parameter Tuning Results:\n')
print("Best Parameter Combination:", xgb_gs.best_params_)
print('Training Data:\n', classification_report(y_train_resampled, xgb_gs.p
print('Testing Data:\n', classification_report(y_test, xgb_gs.predict(X_tes
```

```
Parameter Tuning Results:

Best Parameter Combination: {'learning_rate': 0.1, 'max_depth': 6, 'min_c
hild_weight': 1, 'n_estimators': 100, 'subsample': 0.7}
Training Data:
               precision    recall  f1-score   support

           0       0.98      1.00      0.99      2141
           1       1.00      0.98      0.99      2141

    accuracy                           0.99      4282
   macro avg       0.99      0.99      0.99      4282
weighted avg       0.99      0.99      0.99      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97       709
           1       0.89      0.75      0.81       125

    accuracy                           0.95       834
   macro avg       0.92      0.87      0.89       834
weighted avg       0.95      0.95      0.95       834
```

- The parameter tuning didn't effect the XGBoost performance much.
- Overfitting observed.

Compare the models

At this section, I will compare the classification models to choose the best one to identify the customers who will study doing business with SyriaTel .

I will now look evaluation metrics like precision, recall, accuracy and f1.

I will also plot ROC curves and calculate AUC for each model.

- ROC: Receiver Operating Characteristic curve illustrates the true positive rate against the false positive rate.
- AUC: Area Under Curve

I will use the optimal/best parameter set to instantiate my models. For some models, the GridSearchCV selected the parameters which causes large ovefitting; so low performance on test data. I used Default parameters for these models.

Optimum parameter sets, with f1-score used for tuning

Logictic Regression: {'C': 0.01, 'solver': 'liblinear'}

KNN: Default

Decision Trees: {'criterion': 'gini', 'max_depth': 6, 'min_samples_split': 2}

Random Forest: Default

XGBoost: {'learning_rate': 0.1, 'max_depth': 6, 'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.7}

In [420]:

```
# InInstantiate models with optimum parameters
from sklearn.metrics import roc_auc_score, roc_curve, auc
from sklearn.metrics import precision_score, recall_score, accuracy_score,

logreg_best = LogisticRegression(C=0.01, solver='liblinear', random_state=4
knn_best = KNeighborsClassifier()
dt_best = DecisionTreeClassifier(criterion='gini', max_depth=6, min_samples
rf_best = RandomForestClassifier(random_state=42)
xgb_best = XGBClassifier(learning_rate=0.1, max_depth=6, min_child_weight=1
                         subsample=0.7, random_state=42, eval_metric='loglo

model_list = [logreg_best, knn_best, dt_best, rf_best, xgb_best]
model_names = ['Logistic Regression', 'K-Nearest Neighbor', 'Decision Trees
```

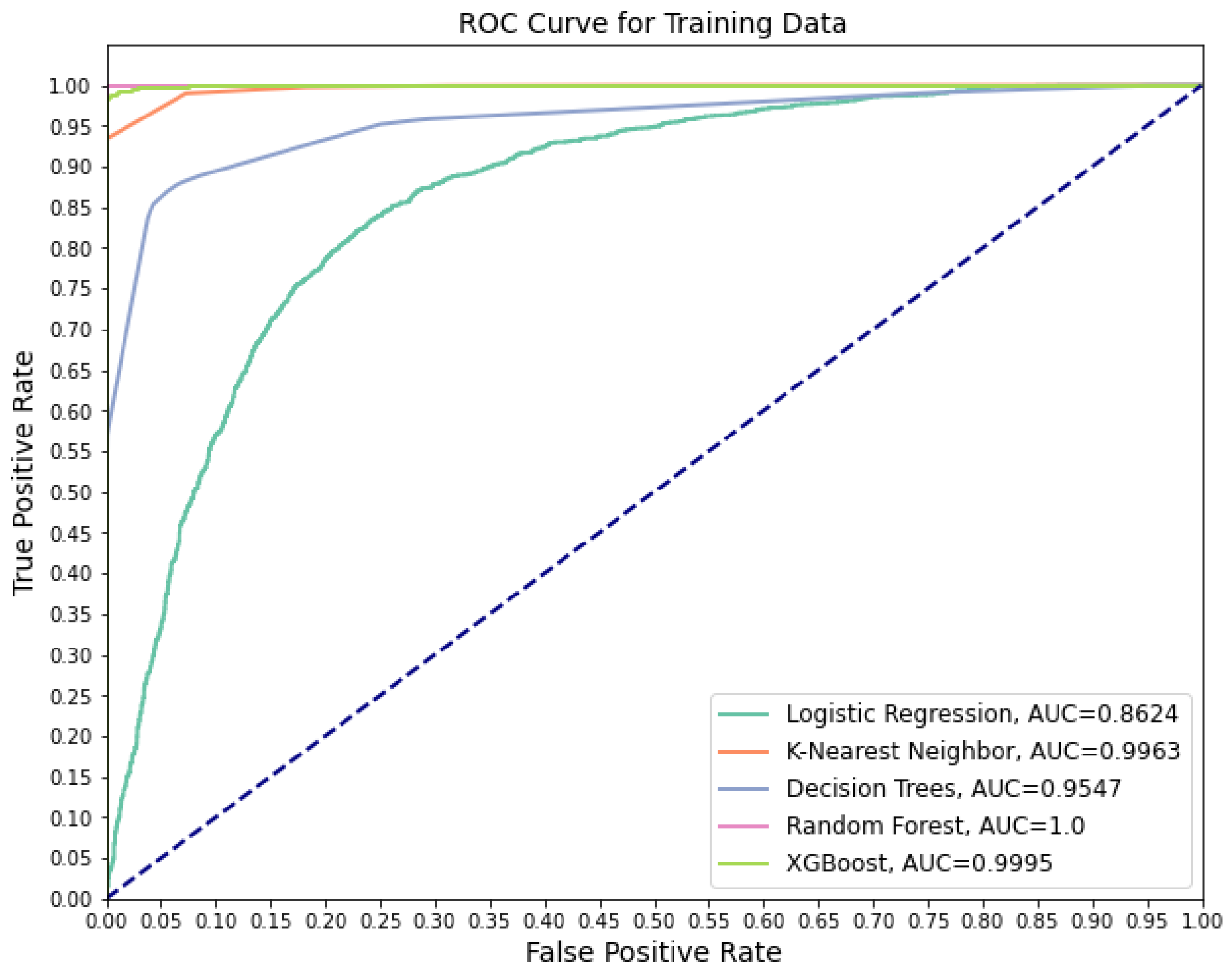
In [427]:

```
def model_scores(dataset_type, X_scaled, y_true):
    """
    dataset_type = 'Testing' or 'Training'
    X_scaled = X_test_scaled or X_train_scaled
    y_true = y_train or y_test

    """
    colors = sns.color_palette('Set2')
    plt.figure(figsize=(10, 8))

    model_scores_list = []

    for n, clf in enumerate(model_list):
        #print(n)

        clf.fit(X_train_scaled_resampled, y_train_resampled)
        #clf.fit(X_train_scaled, y_train)

        y_pred = clf.predict(X_scaled)

        #y_score = clf.decision_function(X_scaled)
        y_prob = clf.predict_proba(X_scaled) #Probability estimates for eac
        fpr, tpr, thresholds = roc_curve(y_true, y_prob[:,1])
        auc_score = auc(fpr, tpr)
        plt.plot(fpr, tpr, color=colors[n], lw=2, label=f'{model_names[n]},

        fit_scores = {'model': model_names[n],
                      'precision': round(precision_score(y_true, y_pred),
                      'recall': round(recall_score(y_true, y_pred),3),
                      'accuracy': round(accuracy_score(y_true, y_pred),3)
                      'f1': round(f1_score(y_true, y_pred),3),
                      'auc': round(auc_score,3)
                     }

        model_scores_list.append(fit_scores)

    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.yticks([i/20.0 for i in range(21)])
    plt.xticks([i/20.0 for i in range(21)])
    plt.xlabel('False Positive Rate', fontsize=14)
    plt.ylabel('True Positive Rate', fontsize=14)
    plt.title(f'ROC Curve for {dataset_type} Data', fontsize=14)
    plt.legend(loc='lower right', fontsize=12)
    #plt.show()
    plt.savefig(f'images/ROC_Curve_{dataset_type}.png')


    model_scores_df = pd.DataFrame(model_scores_list)
    model_scores_df = model_scores_df.set_index('model')
    #print(model_scores_df)

    return model_scores_df
```

```
In [549]: model_scores('Training', X_train_scaled_resampled, y_train_resampled)
```

Out[549]:

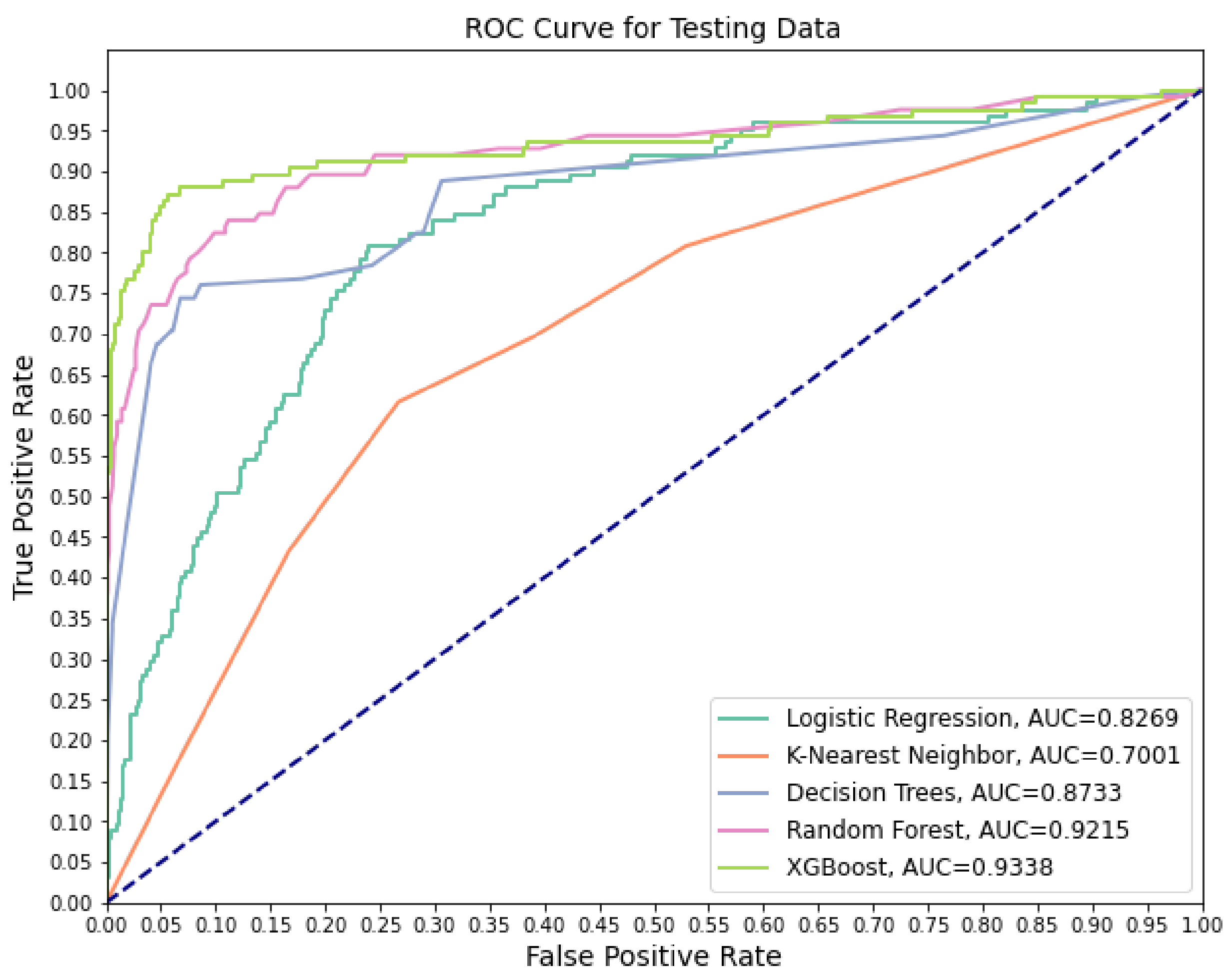
model	precision	recall	accuracy	f1	auc
Logistic Regression	0.777	0.825	0.794	0.801	0.862
K-Nearest Neighbor	0.845	0.998	0.908	0.915	0.996
Decision Trees	0.940	0.869	0.907	0.903	0.955
Random Forest	1.000	1.000	1.000	1.000	1.000
XGBoost	0.998	0.984	0.991	0.991	1.000



In [550]:
```
model_scores('Testing', X_test_scaled, y_test)
```

Out[550]:

model	precision	recall	accuracy	f1	auc
Logistic Regression	0.372	0.792	0.769	0.506	0.827
K-Nearest Neighbor	0.289	0.616	0.716	0.394	0.700
Decision Trees	0.677	0.704	0.905	0.690	0.873
Random Forest	0.788	0.712	0.928	0.748	0.922
XGBoost	0.887	0.752	0.948	0.814	0.934



Which model is best on identinfying churn customers?

According to the test data evaluation metrics, the XGBoost classifier has overall best performance. It also has the best 'recall' and 'f1 score', which matters most for my study.

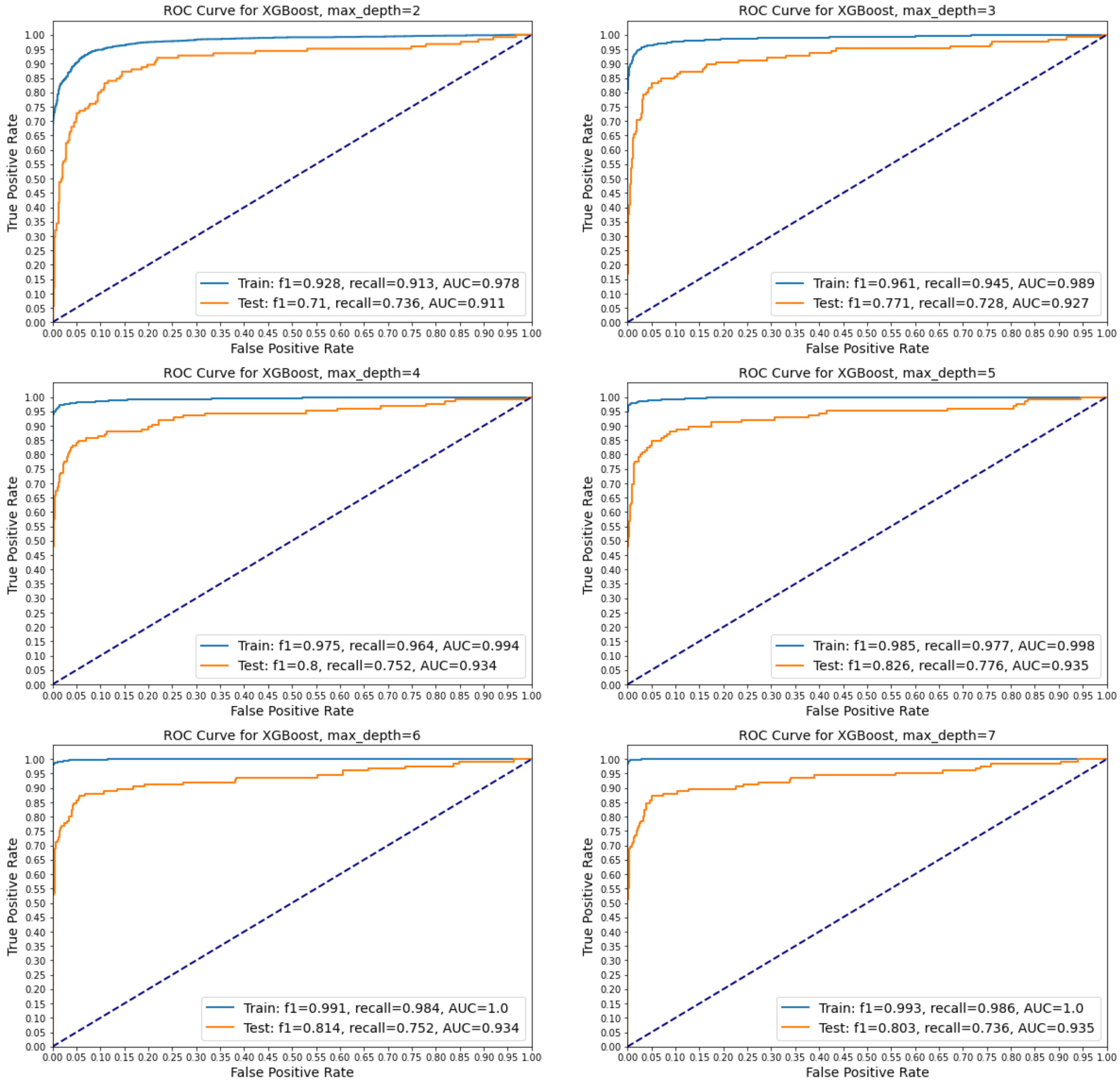
I choose the XGBoost Classiffier as the best model.

Overfitting in XGBoost model

The XGBoost model performed better in training data than the test data. This is overfitting. The decreasing the 'max_depth' can help to minimize the overfitting. I will plot ROC curve for several max_depth values to observe the overfitting.

In [548]:

```
fig, axes = plt.subplots(3, 2, figsize=(20, 20))
#plt.tight_layout(pad=5)

depths = [2, 3, 4, 5, 6, 7]

for ax, d in zip(axes.flat, depths):

    clf = XGBClassifier(learning_rate=0.1, max_depth=d, min_child_weight=1,
                         subsample=0.7, random_state=42, eval_metric='loglo

    clf.fit(X_train_scaled_resampled, y_train_resampled)

    y_train_pred = clf.predict(X_train_scaled_resampled)
    y_train_prob = clf.predict_proba(X_train_scaled_resampled) #Probability
    fpr_train, tpr_train, thresholds_train = roc_curve(y_train_resampled, y
    auc_train = round(auc(fpr_train, tpr_train),3)
    f1_train = round(f1_score(y_train_resampled, y_train_pred),3)
    recall_train = round(recall_score(y_train_resampled, y_train_pred),3)
    ax.plot(fpr_train, tpr_train, lw=2, label=f'Train: f1={f1_train}, recal

    y_test_pred = clf.predict(X_test_scaled)
    y_test_prob = clf.predict_proba(X_test_scaled) #Probability estimates f
    fpr_test, tpr_test, thresholds_test = roc_curve(y_test, y_test_prob[:,1
    auc_test = round(auc(fpr_test, tpr_test),3)
    f1_test = round(f1_score(y_test, y_test_pred),3)
    recall_test = round(recall_score(y_test, y_test_pred),3)
    ax.plot(fpr_test, tpr_test, lw=2, label=f'Test: f1={f1_test}, recall={r

    ax.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    ax.set_xlim([0.0, 1.0])
    ax.set_ylim([0.0, 1.05])
    ax.set_yticks([i/20.0 for i in range(21)])
    ax.set_xticks([i/20.0 for i in range(21)])
    ax.set_xlabel('False Positive Rate', fontsize=14)
    ax.set_ylabel('True Positive Rate', fontsize=14)
    ax.set_title(f'ROC Curve for XGBoost, max_depth={d}', fontsize=14)
    ax.legend(loc='lower right', fontsize=14)

plt.savefig(f'images/ROC_Curve_XGBoost_maxDepth.png')
```



The overfitting decreased a little bit, when max_depth is 4 or 5. The performance of the model with max_depth = 5 is better. I decide on the optimum max_depth = 5.

Final Model

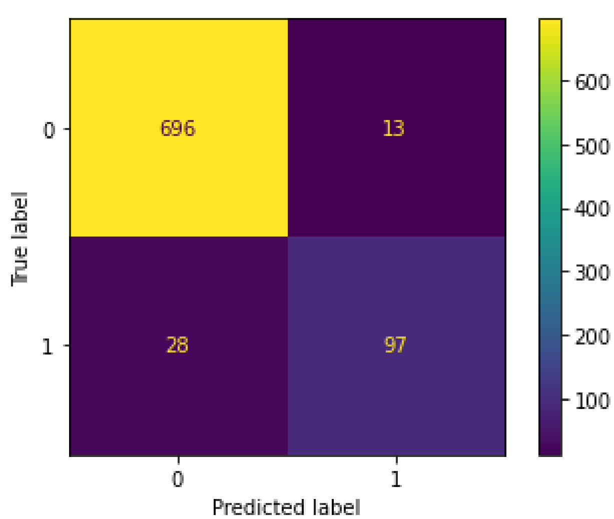
I will create my final model with XGBoost Classifier with the below parameters.

{'learning_rate': 0.1, 'max_depth': 5, 'min_child_weight': 1, 'n_estimators': 100, 'subsample': 0.7}

In [442]:
```
# Instantiate and fit Final Model

xgb_final = XGBClassifier(learning_rate=0.1, max_depth=5, min_child_weight=
                          subsample=0.7, random_state=42, eval_metric='loglo

xgb_final.fit(X_train_scaled_resampled, y_train_resampled)

print('Final Model:\n')
print('Training Data:\n', classification_report(y_train_resampled, xgb_fina
print('Testing Data:\n', classification_report(y_test, xgb_final.predict(X_
```

```
Final Model:

Training Data:
               precision    recall  f1-score   support

           0       0.98      0.99      0.99      2141
           1       0.99      0.98      0.98      2141

    accuracy                           0.99      4282
   macro avg       0.99      0.99      0.99      4282
weighted avg       0.99      0.99      0.99      4282

Testing Data:
               precision    recall  f1-score   support

           0       0.96      0.98      0.97       709
           1       0.88      0.78      0.83       125

    accuracy                           0.95       834
   macro avg       0.92      0.88      0.90       834
weighted avg       0.95      0.95      0.95       834
```

In [444]:
```
# Confusion Matrix on Test Data

plot_confusion_matrix(xgb_final, X_test_scaled, y_test)
plt.savefig('images/confusion_matrix_XGB.png')
```



In [471]:
```python
# The list of important features
len(X)
len(xgb_final.feature_importances_)
fetaure_importance_df = pd.DataFrame(zip(X.columns, xgb_final.feature_impor
fetaure_importance_df.columns = ['predictors', 'importance']
fetaure_importance_df.set_index('predictors')
fetaure_importance_df.sort_values(by='importance', ascending=False, inplace
fetaure_importance_df.head(10)
```

Out[471]:

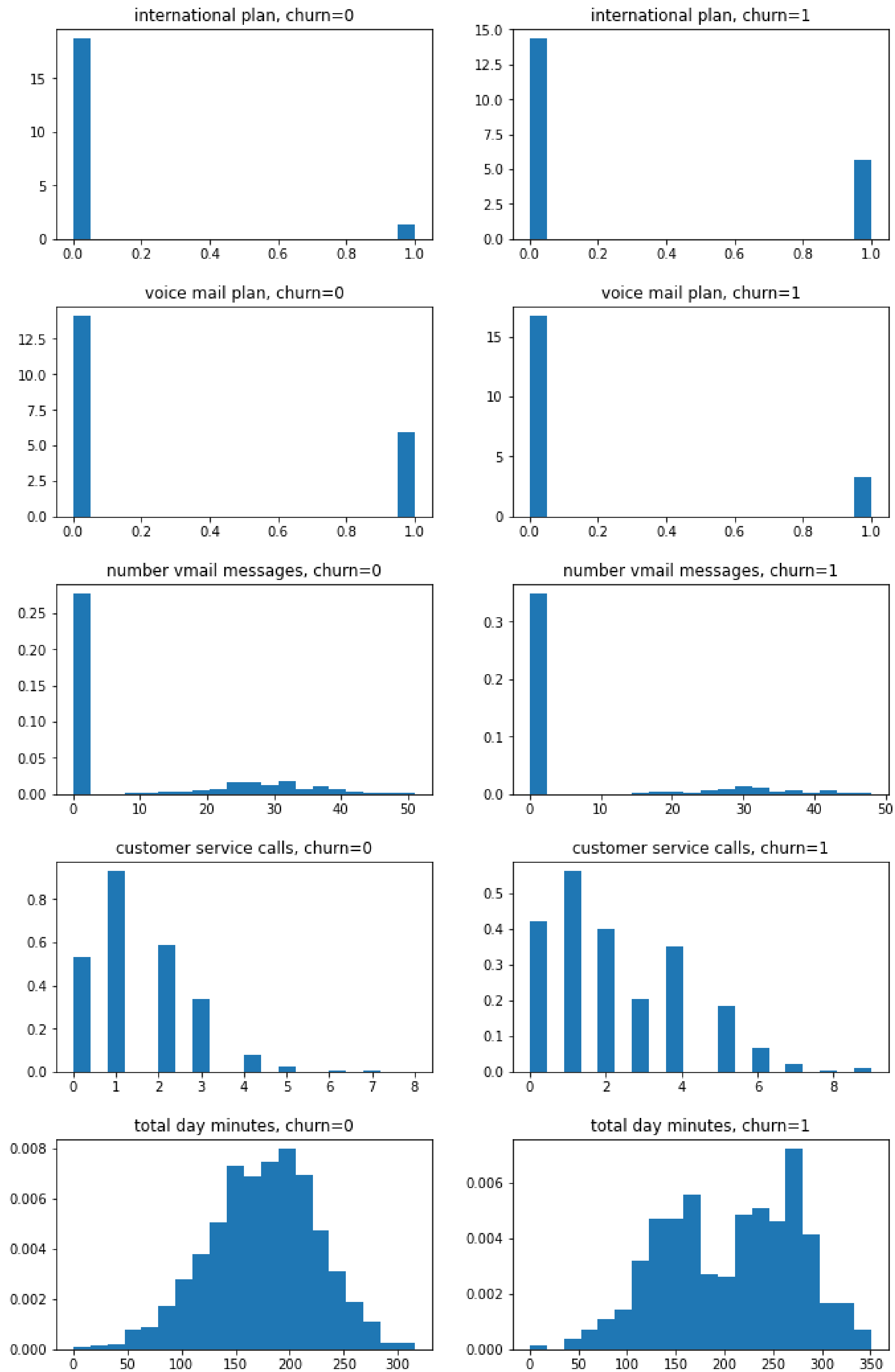
	predictors	importance
69	international plan_0	0.099607
71	voice mail plan_0	0.090104
14	customer service calls	0.076601
1	number vmail messages	0.056749
2	total day minutes	0.044972
67	area code_415	0.034792
7	total eve charge	0.033760
68	area code_510	0.032788
41	state_MT	0.024700
66	area code_408	0.024159

In [520]:
```python
df.columns
```

Out[520]:
```
Index(['state', 'account length', 'area code', 'international plan',
       'voice mail plan', 'number vmail messages', 'total day minutes',
       'total day calls', 'total day charge', 'total eve minutes',
       'total eve calls', 'total eve charge', 'total night minutes',
       'total night calls', 'total night charge', 'total intl minutes',
       'total intl calls', 'total intl charge', 'customer service calls',
       'churn'],
      dtype='object')
```

In [545]:

```
imp_features = ['international plan', 'voice mail plan', 'number vmail mess
                'customer service calls', 'total day minutes']

fig, axes = plt.subplots(len(imp_features), 2, figsize=(10,15))
plt.tight_layout(pad=3)

for n, feat in enumerate(imp_features):

    churn_0 = df[df.churn==0][feat]
    churn_1 = df[df.churn==1][feat]

    axes[n,0].hist(churn_0, bins=20, density=1)
    axes[n,0].set_title(f'{feat}, churn=0')
    axes[n,1].hist(churn_1, bins=20, density=1)
    axes[n,1].set_title(f'{feat}, churn=1')

plt.savefig('images/histograms_importantFeatures.png')
```


international plan, churn=0
international plan, churn=1
voice mail plan, churn=0
voice mail plan, churn=1
number vmail messages, churn=0
number vmail messages, churn=1
customer service calls, churn=0
customer service calls, churn=1
total day minutes, churn=0
total day minutes, churn=1

Interpret

The summary of Final Model performance:

- It successfully indentifies the 78% of the true churn customers. (recall)
- Among the model predicted churn customers, 88% of them are true churn customers. (precision)
- The Harmonic Mean of Precision and Recall (f1-score) is 83%.

The identification numbers on test data:

- Identification numbers:
  - Number of true positives: 97
  - Number of true negatives: 696
  - Number of false positives: 13
  - Number of false negatives: 28
- It identifies 97 out of 125 churn customers correctly.
- 97 out of 110 predicted churn customers are real churn.

Characteristic of churn customers:

- The churn customers are more likely to have international plan than continuous customers.
- The churn customers are less likely to have voice mail plan than continuous customers.
- The churn customers have less voice mail messages than continuous customers (as a result of less voice mail plan)
- The churn customers have more customer service calls than continuous customers.
- The churn customers have more total day minutes than continuous customers.

Future Work

- Improve the XGBT model (final model) performance
  - Search each parameter separately to understand the effect on performance
  - Obtain a more sensitive/informed range for each parameter to be used in grid search
  - Study the effect of other hyperparameters
- Use weighted f1-score, with more weight on recall than precision
  - to compare model performance
  - and for parameter tuning