Customer Retention Classification

Author: Matthew Lipman

Overview

"Do what you do so well that they want to see it again and bring their friends." -Walt Disney

Customer retention is a mainstay for profitability and success of businesses. With the telecommunications industry's ever-changing technologies and rapidly increasing user base, Internet and phone providers need to continually advance their products and services to meet their customers' growing expectations and needs. Using data provided by IBM Watson Analytics about Telco, a telecommunications company, I have analyzed key features driving a customer from leaving the service. With over seven thousand customers and over twenty different features and attributes provided about each customer, this dataset provides a strong basis to classify how and why a customer churns out of the service. By understanding, analyzing, and modeling the data, I am able to correctly classify 86.8% of customers who churned out and left the service based upon how the customers interact with specific features and parameters.

Business Problem

Customer retention is key to driving a company's profitability. Customer retention allows for companies to reduce time and costs spent by the Sales department driving new business, and helps businesses maintain a stable, reliable flow of income. In addition, customers who are happy with their service and perceive that they are getting a strong value for their product, are less likely to leave the service, and more likely to recommend the products and services they are receiveing to their colleagues, family, and friends. As Tamulienea and Gabryteb have pointed out in a case study of Lithuanian mobile operators, customers who are satisfied end up being growth opportunities for future revenue of the business by a process called "relationship marketing." Thus, focusing on retention reduces loss of revenue and inadvertently cultivates future sales. Currently, 36.2% of customers churned out of the business. Additionally, the base of customers who do not churn generate almost 13.2 million dollars while customers who have churned generated just under 2.9 million dollars in total revenue. If the 36.2% of customers who had churned had not churned--based upon the current average lifetime revenue generated per customers--Telco could have received an additional 1.9 million dollars in revenue, not including the additional customers and revenue that could have been achieved through relationship marketing. As a whole, maintaining a strong client base by retaining customers is a key component to the vitality of your business. Through this research and development, I will be answering/addressing the following questions:

1. *Can churn be explained and understood through a classification model?*
2. *If so, which features are associated with churn?*
3. *How can Telco reduce churn and other businesses learn how to reduce customer churn?*

The model I created aims to be able to classify whether or not a customer successfully churns.

(Citation: 19th International Scientific Conference; Economics and Management 2014, ICEM 2014, 23-25 April 2014, Riga, Latvia. Factors influencing customer retention: case study of Lithuanian mobile operators Vilma Tamulienea, Ingrida Gabryteb.)

Data

The dataset used for this project can be found at [https://www.kaggle.com/blastchar/telco-customer-churn (https://www.kaggle.com/blastchar/telco-customer-churn)](https://www.kaggle.com/blastchar/telco-customer-churn). It contains 7,043 rows of 21 features representing 7,043 different individuals who were all Telco customers. The features describe some qualitative features about the customer, the quality and quantity of services used by the customer, and metrics involving how much money was spent by the customer and the length of time the customer remained using the service. The target feature--or feature we are trying to get a deeper understanding of as it relates to customer retention--is "churn." Churn is defined as whether or not the customer churned out of the service and left. This variable is categorical and binary (Yes/No). "Yes" represents that the customer has churned and thus left the service. "No" represents that the customer remained utilizing and paying for the service provided. A comprehensive exploratory data analysis was conducted and is available for viewing in the Jupyter Notebook, "EDA, Preprocessing, and Visualizing Relationships."

Methods

This classification modeling project is in accordance with the CRISP-DM method. I began my work by importing, preprocessing, cleaning, and visualizing the relationships between the features and the target feature, "churn." First I used a relatively minimally pre-processed dataset to create a baseline model. Each subsequential model had an iterative approach using a number of techniques such as Synthetic Minority Oversampling Technique (SMOTE), feature engineering, attempting different classification modeling techniques, and grid searches to address a variety of modeling obstacles such as class imbalances, model complexity reduction, and overfitting. To streamline the modeling process, I created a number of pipelines to assess and compare the various models' metric performances. I identified the metric to best analyze my model, recall, because I was interested in reducing the number of times the model predicted that someone would not churn, but did in fact churn. By reducing the number of these instances, this model is able to minimize costly situations where we did not identify a potential churner and is able to notify the Sales team which products to try to sell to the existing customer.

Results

After conducting eight iterations of the baseline model, I was able to acheive a model recall of 86.8%, meaning 86.8% of customers who churned were correctly classified by the model. The most effective model that classified churn with 86.8% recall used the following attributes:

- SMOTE to address class imbalance
- Feature Engineering to reduce model complexity
- A Decision Tree Classifier
- Hyperparameter tuning to optimize the model's recall and reduce model overfitting

This Decision Tree Classifier indicates that 86.8% of customers churning are able to be successfully predicted by the model. In addition, this Decision Tree illustrates:

1. The contract type is very important to customer retention. Specifically, customers who are on a longer-term contract are strongly associated with not churning. Customers that are on atleast a 1-year contract are less likely to churn and customers that are on atleast a 2-year contract are much less likely to churn.
2. The amount of money spent by a customer for their services is a strong indicator of classifying whether or not they are going to churn. Customers that spend more than 59.65 dollars per month on their biill are likely to churn and customers that spend 89.85 dollars per month on their bill are highly likely to churn.
3. There is a conditionality to these two features. For example, those are are on an atleast 2-year contract and paying less than 59.65 dollars per month on their monthly rate are of the subgroup of individuals who are least likely to churn.
4. Customers who are on a month-to-month contract, pay more than $89.85 per month, are extremely likely to churn if they pay by automatic payments via a credit card.

By honing in on recall, the model ensures that we capture all instances where a customer churns, which makes this the most important metric for the problem at hand because it is the instance where a customer leaves the service and the company loses the most revenue. By focusing on recall, this model avoids false negatives or Type II errors, where we do not predict and identify a customer who will churn.

Concluding Recommendations

1. **Get customers on long term contracts.** In order to reduce churn, I advise Telco to reach out to their population of customers that are on a month-to-month contract and attempt to convert them into a longer term contract. Currently 55.1 percent of customers are not on a long term contract. By increasing the number of customers on a minimum of a 1-year contract, the model forecasts that less customers will churn.
2. **Reduce the monthly bills of customers who pay more than $59.65 per month.** By reaching out to customers who pay more than the threshold for what the model predicts are customers more likely to churn, Telco's retention team can try helping the customer save on their monthly bill. This will increase customer relations by building trust between the customer and service provider, further aiding in relationship marketing.
3. **For customers on month-to-month contracts and pay more than 89.85 dollars per month, have the retention team reach out to the customer and change their payment method from automatic credit card.** If all three of these conditionals are satisfied, customers are extremely likely to churn. With that said, by changing their payment method they become substantially less likely to churn.

Future Work

1. **Model Implementation:** Conduct a short term study on the financial effects of converting customers onto longer term contracts, reducing monthly payments, and changing payment methods.
2. **Finer-tune understanding of the service add-ons:** To keep the complexity of the model more straightforward, this model has feature engineered the add-ons and treated them equally when classifying churn. With more time and understanding, it would benefit the company to see which specific add-ons affect the causality of churn.
3. **Continue to improve model overfitting:** The training data has a higher recall percentage of 5.9 as compared to the testing data. With more time, I would increase the number of cross validations to conduct the hyperparameter tuning, with the hope of finding a better-tuned model and reduce overfitting.

In [1]:
```
import pandas as pd
pd.set_option('display.max_columns', None) # to display all columns
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy_score, roc_curve, auc, precision_score, reca
ll_score, f1_score
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE
from sklearn import tree
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, plot_confus
ion_matrix
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
```

Data Loading and Comprehension

In [2]:
```
df = pd.read_csv('data/WA_Fn-UseC_-Telco-Customer-Churn.csv')
```

In [3]:
```
df.head()
```

Out[3]:

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetServ
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	[
1	5575-GNVDE	Male	0	No	No	34	Yes	No	[
2	3668-QPYBK	Male	0	No	No	2	Yes	No	[
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	[
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber o

In [4]:
```
df.shape
```

Out[4]:
```
(7043, 21)
```

In [5]:
```
# Looking into TotalCharges, which has a value string type for some strange reaso
n
df['TotalCharges'].describe()
```

Out[5]:
```
count      7043
unique     6531
top        20.2
freq         11
Name: TotalCharges, dtype: object
```

In [6]:
```
def preliminary_research(df):
    for col in df.columns:
        unique_vals = df[col].unique()
        if len(unique_vals) < 10:
            print("Unique values for column {}: {}".format(col, unique_vals))
        else:
            if df[col].dtype == 'object':
                print("column {} has values string type".format(col))
            elif df[col].dtype == 'int64':
                print("column {} is numerical".format(col))
            elif df[col].dtype == 'float64':
                print("column {} is numerical".format(col))
    return

preliminary_research(df)
```

```
column customerID has values string type
Unique values for column gender: ['Female' 'Male']
Unique values for column SeniorCitizen: [0 1]
Unique values for column Partner: ['Yes' 'No']
Unique values for column Dependents: ['No' 'Yes']
column tenure is numerical
Unique values for column PhoneService: ['No' 'Yes']
Unique values for column MultipleLines: ['No phone service' 'No' 'Yes']
Unique values for column InternetService: ['DSL' 'Fiber optic' 'No']
Unique values for column OnlineSecurity: ['No' 'Yes' 'No internet service']
Unique values for column OnlineBackup: ['Yes' 'No' 'No internet service']
Unique values for column DeviceProtection: ['No' 'Yes' 'No internet service']
Unique values for column TechSupport: ['No' 'Yes' 'No internet service']
Unique values for column StreamingTV: ['No' 'Yes' 'No internet service']
Unique values for column StreamingMovies: ['No' 'Yes' 'No internet service']
Unique values for column Contract: ['Month-to-month' 'One year' 'Two year']
Unique values for column PaperlessBilling: ['Yes' 'No']
Unique values for column PaymentMethod: ['Electronic check' 'Mailed check' 'Bank
transfer (automatic)'
 'Credit card (automatic)']
column MonthlyCharges is numerical
column TotalCharges has values string type
Unique values for column Churn: ['No' 'Yes']
```

A Breakdown of the Columns

'CustomerID'

- ID numbers which have no impact on churn

'Gender' is binary

- Male
- Female

'SeniorCitizen' is binary

- 0 - no
- 1 - yes

'Partner' is binary

- Yes - customer has a partner
- No - customer do not have a partner

'Dependents' is binary

- Yes - customer has dependent(s)
- No - customer does not have dependents

'Tenure' is numerical

- Represents how long the customer has been using the service

'PhoneService' is binary

- Yes - customer has phone service with company
- no - customer do not have phone service with company

'MultipleLines' is categorical

- Yes - customer has multiple line subscriptions
- No - customer has only 1 line subscription
- No phone service - customer do not have phone service with company

'InternetService' is categorical

- DSL
- Fiber optic
- No - customer do not have internet service with company

'OnlineSecurity' is categorical

- Yes - customer has online security with company
- No - customer do not have online security with company
- No internet service - customer do not have internet service with company

'OnlineBackup' is categorical

- Yes - customer has online backup with company
- No - customer do not have online backup with company
- No internet service - customer do not have internet service with company

'DeviceProtection' is categorical

- Yes - customer has device protection with company
- No - customer do not have device protection with company
- No internet service - customer do not have internet service with company

'TechSupport' is categorical

- Yes - customer has technical support with company
- No - customer do not have technical support with company
- No internet service - customer do not have internet service with company

'StreamingTV' is categorical

- Yes - customer has streaming TV service with company
- No - customer do not have streaming TV service with company
- No internet service - customer do not have internet service with company

'StreamingMovies' is categorical

- Yes - customer has streaming movies service with company
- No - customer do not have streaming movies with company
- No internet service - customer do not have internet service with company

'Contract' is categorical

- Month-to-month - customer is on a no-commitment plan
- One year - customer is on a 1-year contract commitment
- Two year - customer is on a 2-year contract commitment

'PaperlessBilling' is binary

- Yes - only receives bills via email
- No - receives letters in mail with bill

'PaymentMethod' is categorical

- Electronic check
- Mailed check
- Bank transfer (automatic)
- Credit card (automatic)

'MonthlyCharges' is numerical

- count 7043.000000
- mean 64.761692
- std 30.090047
- min 18.250000
- 25% 35.500000

- 50% 70.350000
- 75% 89.850000
- max 118.750000

'TotalCharges' -- NEEDS ATTENTION

- Says that dtype is a string. However, this should be numerical...

'Churn' is binary

- No - customer is still an active customer
- Yes - customer has left service

Data Cleaning

In [7]:
```
# change column names to all lowercase
df.columns = map(str.lower, df.columns)
```

In [8]:
```
# convert "No / Yes" binary options to numerical 0s and 1s
binary_columns = ['partner', 'dependents', 'phoneservice', 'paperlessbilling', 'c
hurn']
df[binary_columns] = df[binary_columns].eq('Yes').mul(1)

# convert the categorical variables that have numeric significance into numerical
df.multiplelines = df.multiplelines.map({'No phone service':0, 'No':1, 'Yes':2})
df.contract = df.contract.map({'Month-to-month':0, 'One year':1, 'Two year':2})

# convert "Male / Female" binary options to numerical 0s and 1s
df['gender'] = df['gender'].eq('Female').mul(1)
```

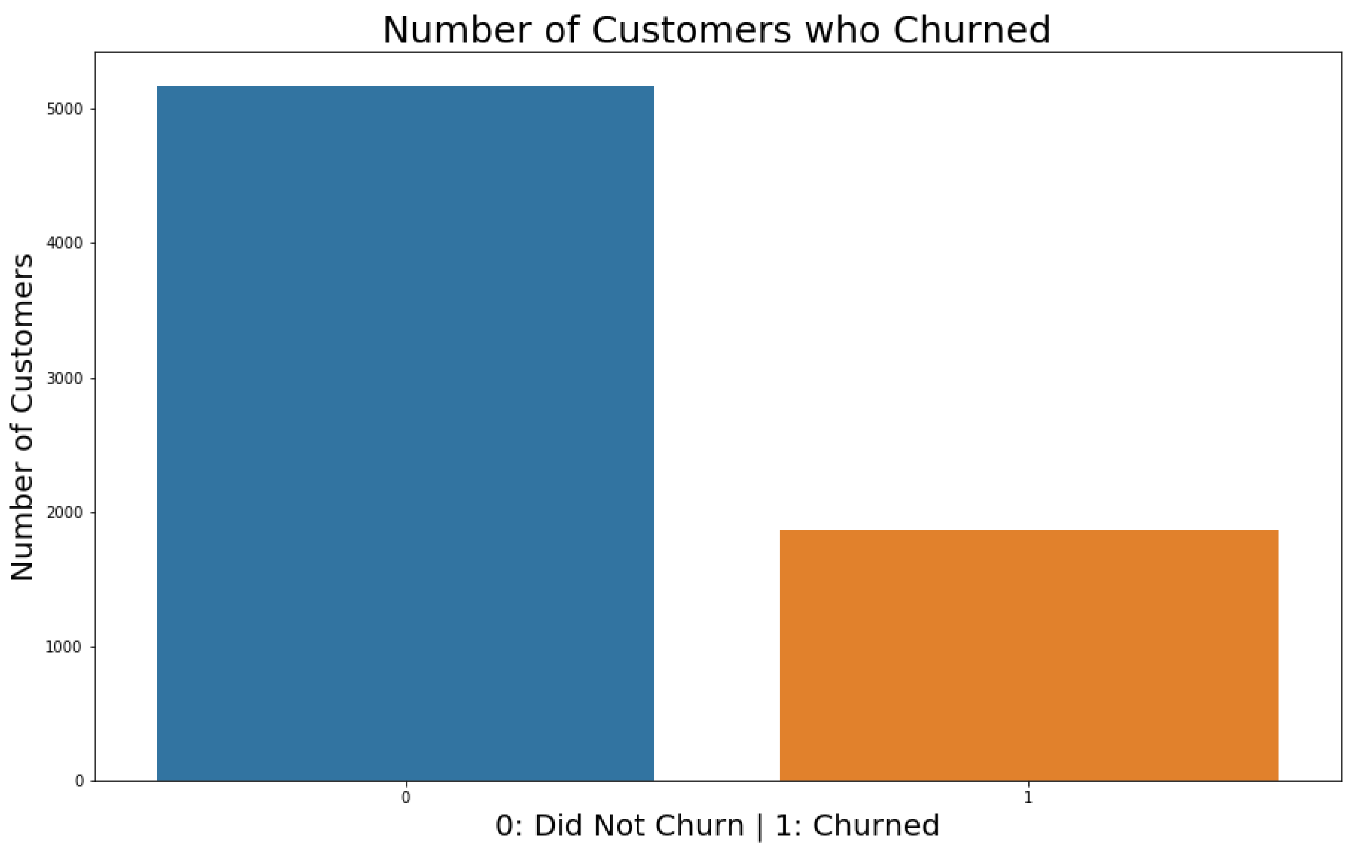
In [9]:
```
df = df.drop(columns = 'customerid') # drop 'customerid' column.
df['totalcharges'] = df['totalcharges'].replace(' ', np.nan, regex=True) # replac
es blank to NaN
df = df.dropna() # drop the NaN values
df['totalcharges'] = df.totalcharges.astype(float) # converts to float
```

In [10]:
```
# create dummy variables
df_dummified = pd.get_dummies(df, drop_first=True, dtype=int)
```

Visualizing Relationships

In [11]:
```
# Class frequency of target variable
fig, ax = plt.subplots(figsize=(15,9))
ax = sns.countplot(x='churn', data=df)
ax.set_xlabel('0: Did Not Churn | 1: Churned', fontsize=20)
ax.set_ylabel('Number of Customers', fontsize=20)
ax.set_title('Number of Customers who Churned', fontsize=25);
df['churn'].value_counts()
```

Out[11]:
```
0    5163
1    1869
Name: churn, dtype: int64
```



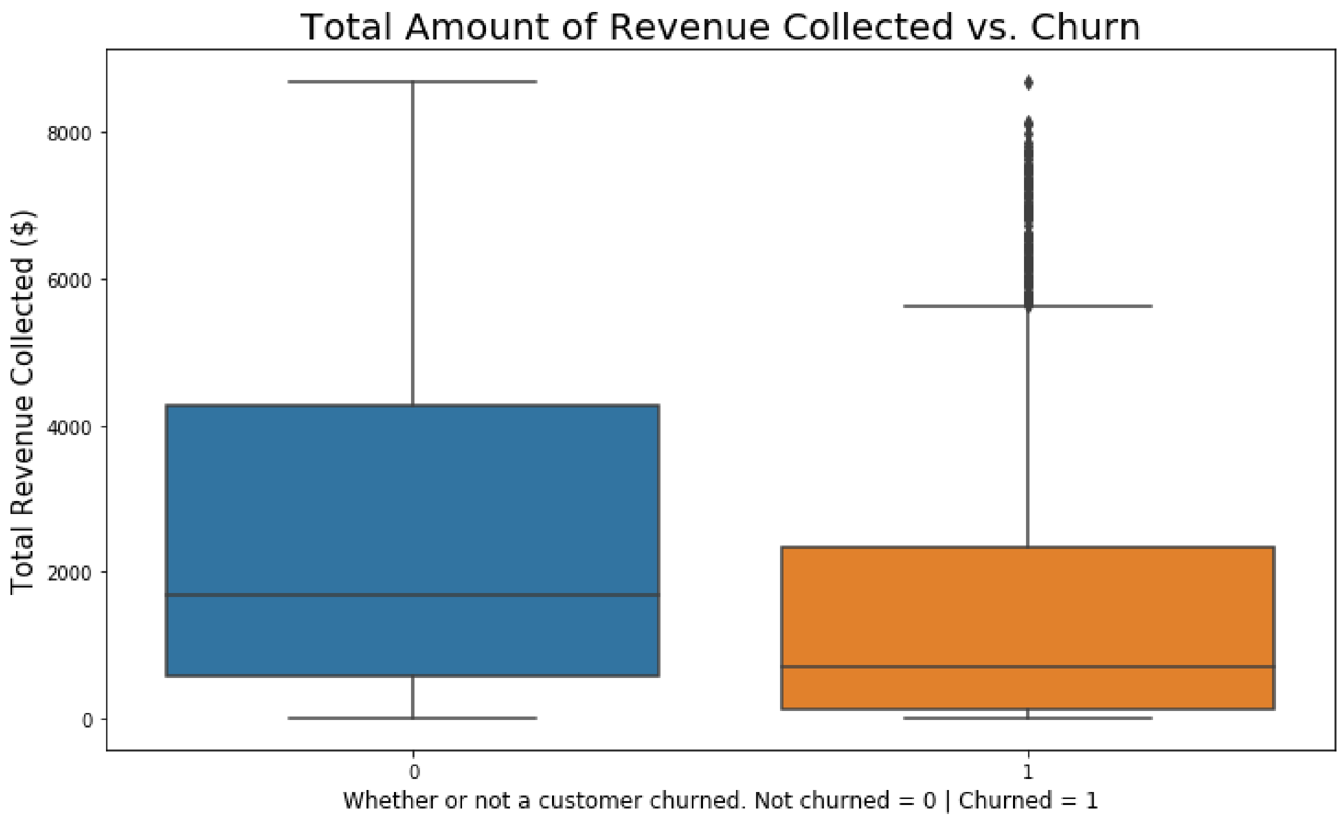
In [12]:
```
percent_that_churned = ((1869/5163)*100)
print('The total percentage of customers that churned:', round(percent_that_churn
ed,1))
```

```
The total percentage of customers that churned: 36.2
```

In [13]:
```
# Looking at the relationship between the total amount of revenue earned and whet
her or not they churned
plot1 = pd.concat([df['totalcharges'], df['churn']], axis=1)
f, ax = plt.subplots(figsize=(12, 7))
fig = sns.boxplot(x='churn', y='totalcharges', data=plot1)
plt.title("Total Amount of Revenue Collected vs. Churn", fontsize=20)
ax.set_ylabel('Total Revenue Collected ($)', fontsize=15)
ax.set_xlabel('Whether or not a customer churned. Not churned = 0 | Churned = 1',
fontsize=12);
df.groupby('churn')[['totalcharges']].describe()
```

Out[13]:

	totalcharges							
	count	mean	std	min	25%	50%	75%	max
churn								
0	5163.0	2555.344141	2329.456984	18.80	577.825	1683.60	4264.125	8672.45
1	1869.0	1531.796094	1890.822994	18.85	134.500	703.55	2331.300	8684.80



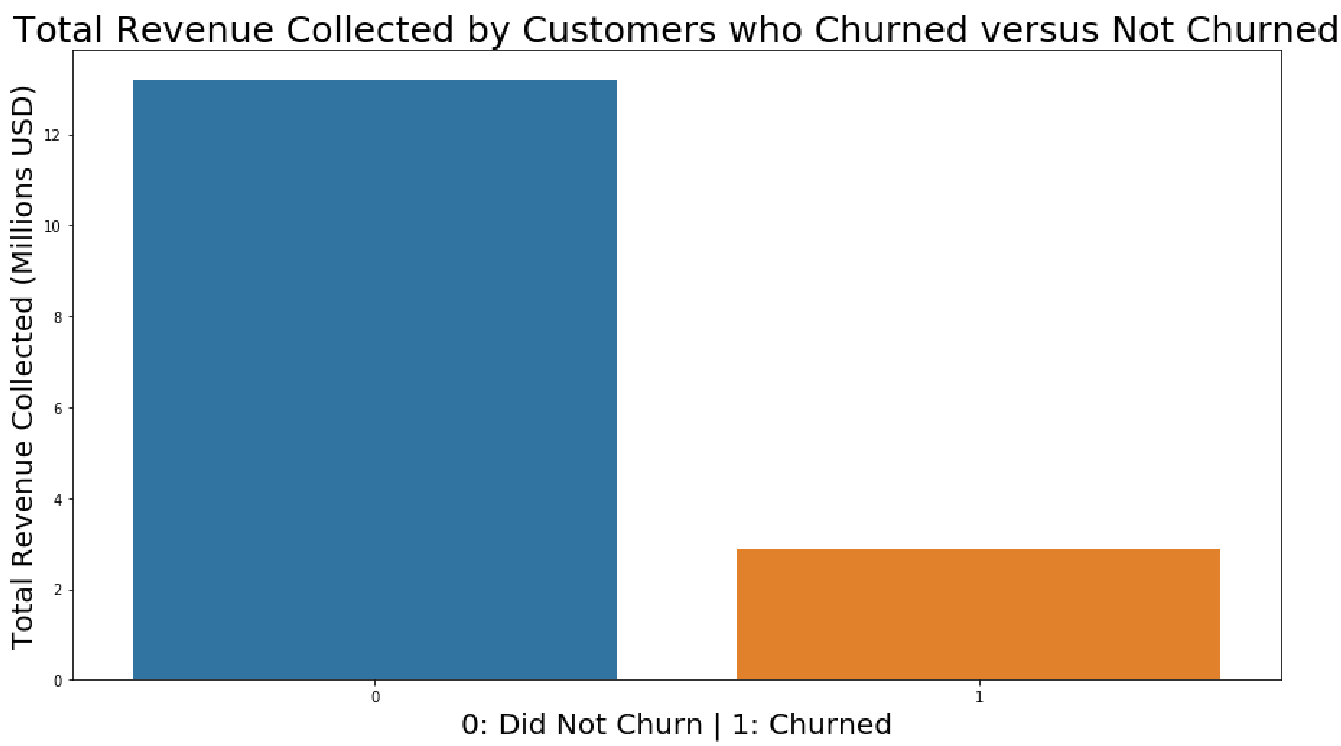
Notes: Customers who do not churn generate a larger revenue than customers who churn. Specifically, customers who do not churn generate on average $1023.54 more revenue than customers who churn.

In [14]:
```
# Projecting how much revenue could have been generated by customers who churned
projection = 1869*2555.34 - 1869*1531.80
projection_in_millions = projection/1000000
print("Revenue that would have been generated if customers who had churned",
      "spent the average lifetime revenue from customers who did not churn:",
      round(projection_in_millions, 3), "million USD.")
```

```
Revenue that would have been generated if customers who had churned spent the av
erage lifetime revenue from customers who did not churn: 1.913 million USD.
```

In [15]:
```
# calculating the total amount of revenue collected by customers who churned vers
us not churned
df_total_revenues = df.groupby(["churn"]).totalcharges.sum().reset_index()
df_total_revenues['revenueinmillions'] = df_total_revenues['totalcharges']/100000
0
print(df_total_revenues)
# Class frequency of target variable
plt.figure(figsize=(15,8))
ax = sns.barplot(x='churn', y='revenueinmillions', data=df_total_revenues)
ax.set_xlabel('0: Did Not Churn | 1: Churned', fontsize=20)
ax.set_ylabel('Total Revenue Collected (Millions USD)', fontsize=20)
ax.set_title('Total Revenue Collected by Customers who Churned versus Not Churne
d', fontsize=25);
```

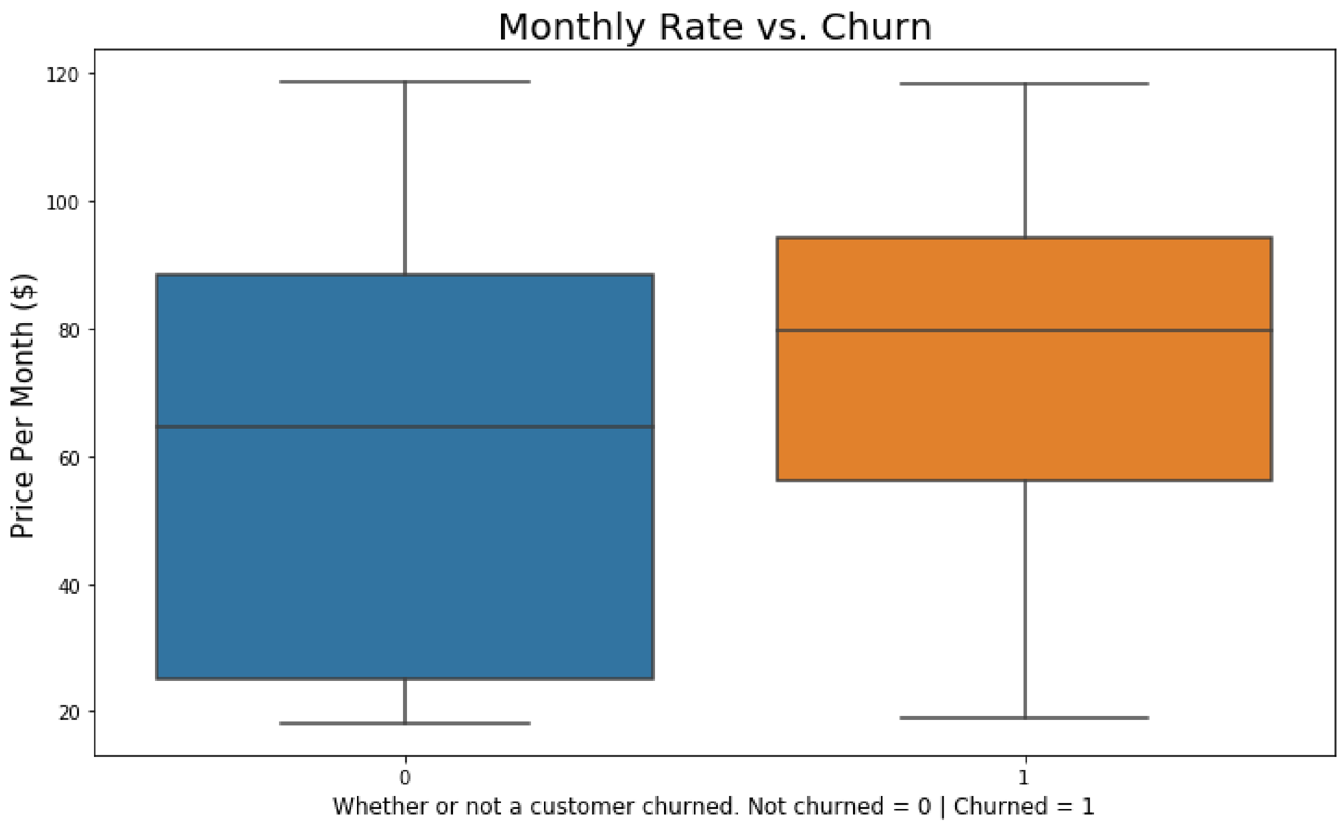
```
   churn  totalcharges  revenueinmillions
0      0    13193241.8          13.193242
1      1     2862926.9           2.862927
```



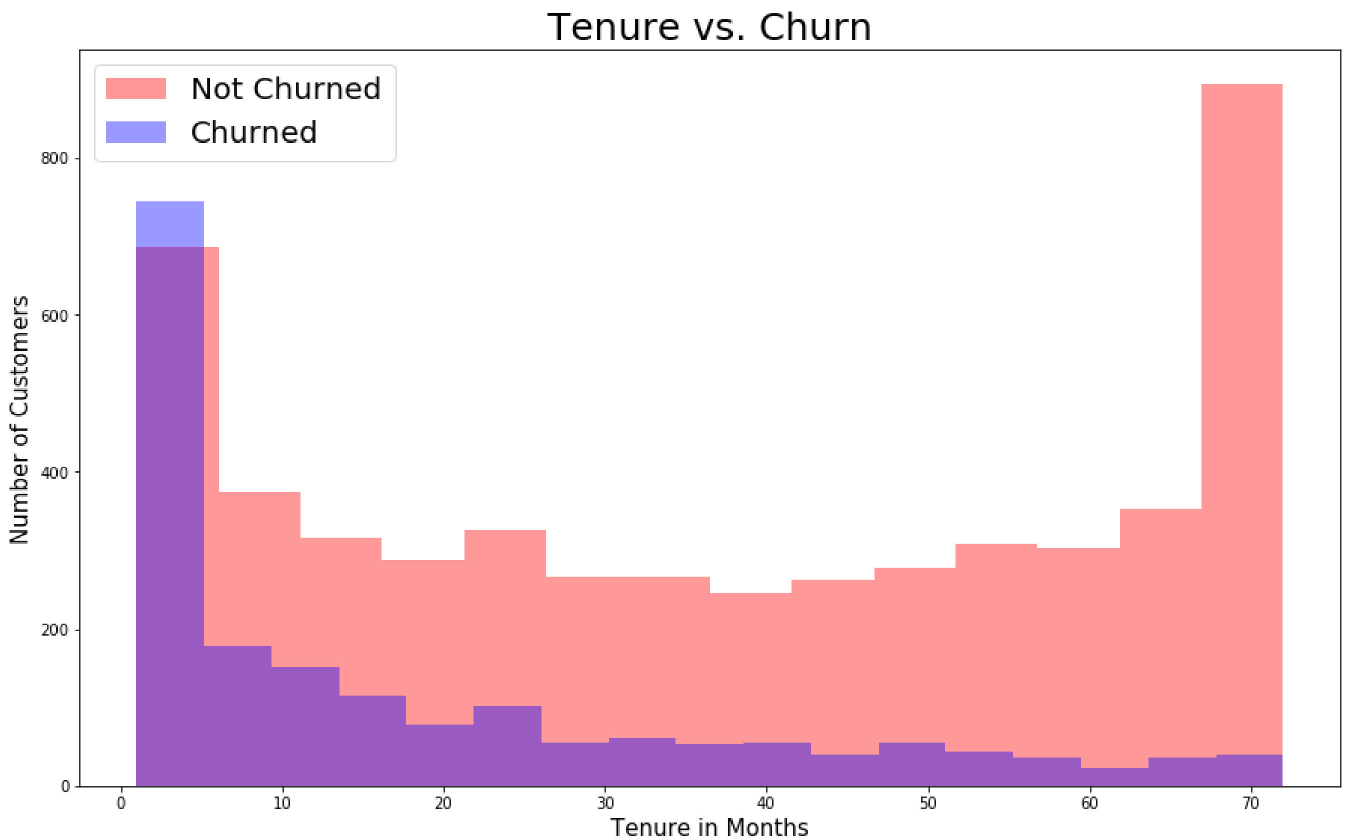
In [16]:

```
# Looking at the relationship between the monthly bill and whether or not they ch
urned
plot1 = pd.concat([df['monthlycharges'], df['churn']], axis=1)
f, ax = plt.subplots(figsize=(12, 7))
fig = sns.boxplot(x='churn', y='monthlycharges', data=plot1)
plt.title("Monthly Rate vs. Churn", fontsize=20)
ax.set_ylabel('Price Per Month ($)', fontsize=15)
ax.set_xlabel('Whether or not a customer churned. Not churned = 0 | Churned = 1',
fontsize=12);
print("The amount of money spent by customers per month who are on the threshold
 of likely to churn:",
       df.monthlycharges.quantile(0.407), "USD")
```

```
The amount of money spent by customers per month who are on the threshold of lik
ely to churn: 59.65 USD
```



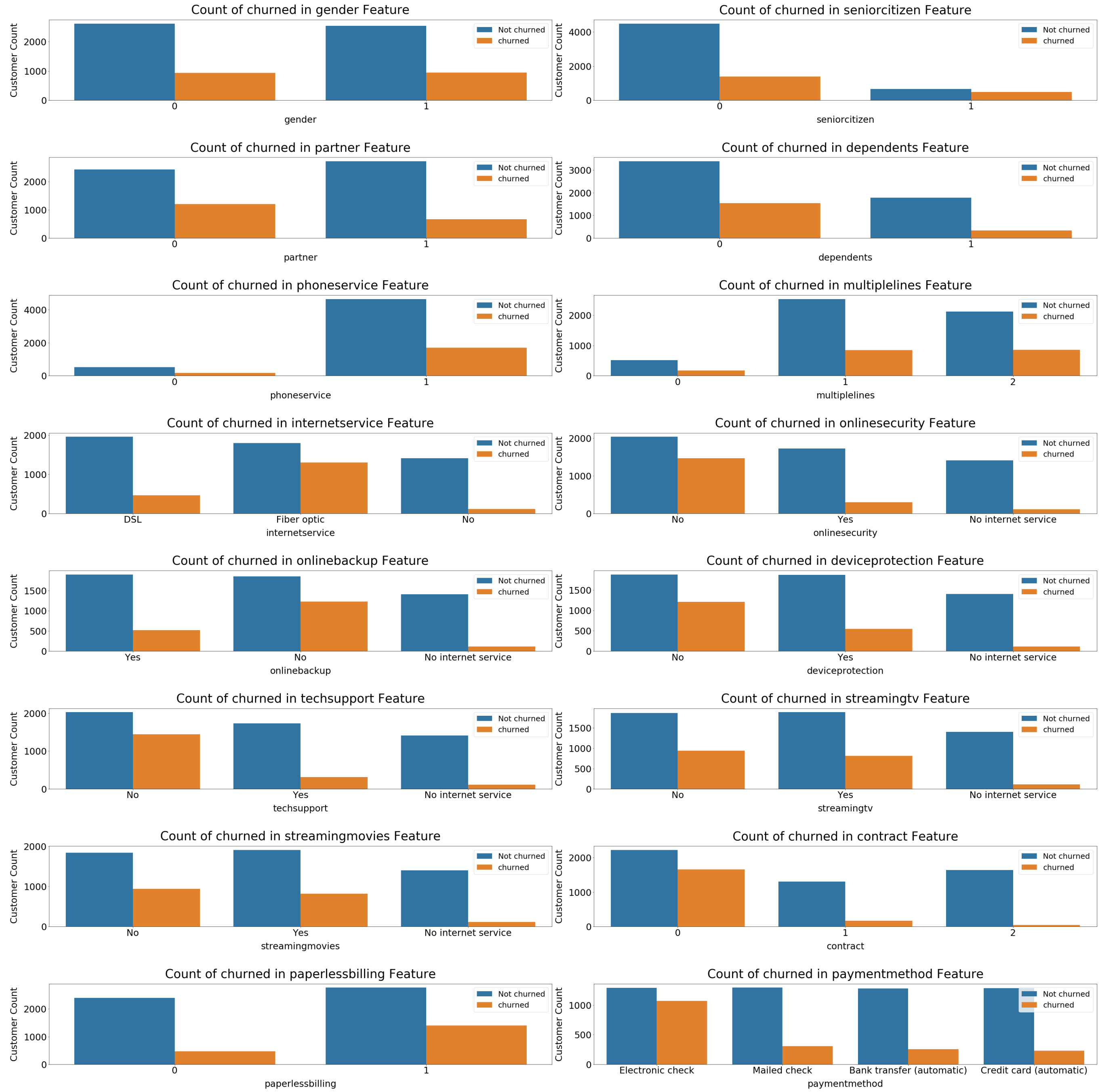
In [17]:
```
# defining variables for histogram
churn = df['churn'] == 1
did_not_churn = df['churn'] == 0

# Histogram looking at customer tenure
fig, ax = plt.subplots(figsize=(15,9))
ax = sns.distplot(df[~churn]['tenure'], label='Not Churned',
                  kde=False, color='red')
sns.distplot(df[churn]['tenure'], label='Churned', kde=False, color='blue')
ax.set_xlabel('Tenure in Months', fontsize=15)
ax.set_ylabel('Number of Customers', fontsize=15)
ax.set_title('Tenure vs. Churn', fontsize=25)
ax.legend(fontsize=20);
```



Notes: The highest frequency of customers who churned did not stay with the service for more than one month. The highest frequency of customers who did not churn stayed with the service for seventy months. As tenure increased, the frequency of customers who churned decreased.

In [18]:
```
cat_features = ['gender', 'seniorcitizen', 'partner', 'dependents',
                'phoneservice', 'multiplelines', 'internetservice',
                'onlinesecurity', 'onlinebackup', 'deviceprotection',
                'techsupport', 'streamingtv', 'streamingmovies',
                'contract', 'paperlessbilling', 'paymentmethod']

def plot_bar_cat_cols(df, cat_features, target_label, hue):
    '''Visualizing all categorical features versus churn'''
    fig, axs = plt.subplots(ncols=2, nrows=8, figsize=(50, 50))
    plt.subplots_adjust(right=1.5, top=1.25)
    for i, feature in enumerate(cat_features, 1):
        plt.subplot(8, 2, i)
        sns.countplot(x=feature, hue=hue, data=df)
        plt.xlabel('{}'.format(feature), size=30, labelpad=15)
        plt.ylabel('Customer Count', size=30, labelpad=15)
        plt.tick_params(axis='x', labelsize=30)
        plt.tick_params(axis='y', labelsize=30)

        plt.legend(['Not {}'.format(target_label), '{}'.format(target_label)], lo
c='upper right', prop={'size': 25})
        plt.title('Count of {} in {} Feature'.format(target_label, feature), size
=40, y=1.05)

    plt.tight_layout(h_pad=5)
    plt.show();
```

```
In [19]: plot_bar_cat_cols(df, cat_features, 'churned', 'churn')
```




```
In [20]: print(df.groupby('contract')[['churn']].describe())
         num_of_folks_not_on_contract = 3875
         percent_not_on_contract = (num_of_folks_not_on_contract/7032)*100
         print("Percentage of customers that are not on a longterm contract:",round(percen
         t_not_on_contract,2))
```

```
           churn
           count      mean       std  min  25%  50%  75%  max
contract
0         3875.0  0.427097  0.494720  0.0  0.0  0.0  1.0  1.0
1         1472.0  0.112772  0.316421  0.0  0.0  0.0  0.0  1.0
2         1685.0  0.028487  0.166408  0.0  0.0  0.0  0.0  1.0
Percentage of customers that are not on a longterm contract: 55.11
```

Create a baseline model: Decision Tree Classifier

In [21]:
```
# Scale the data
scale = MinMaxScaler()
df_dummified = pd.DataFrame(scale.fit_transform(df_dummified.values),
                            columns=df_dummified.columns,index=df_dummified.index
)
```

In [22]:
```
df_dummified.head()
```

Out[22]:

	gender	seniorcitizen	partner	dependents	tenure	phoneservice	multiplelines	contract	paperlessbilling
0	1.0	0.0	1.0	0.0	0.000000	0.0	0.0	0.0	1.(
1	0.0	0.0	0.0	0.0	0.464789	1.0	0.5	0.5	0.(
2	0.0	0.0	0.0	0.0	0.014085	1.0	0.5	0.0	1.(
3	0.0	0.0	0.0	0.0	0.619718	0.0	0.0	0.5	0.(
4	1.0	0.0	0.0	0.0	0.014085	1.0	0.5	0.0	1.(

In [23]:
```
# Create features and labels
X = df_dummified.drop('churn', axis=1)
y = df_dummified['churn']

# Perform an train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_s
tate=0, stratify=y)
```

In [24]:
```
# CREATING A PIPELINE
def pipeline(name_of_pipeline, classifier, X_train, y_train, X_test, y_test):
    '''Creates and displays the pipeline classifiers along with the report of met
rics'''
    name_of_pipeline = Pipeline([('classifier', classifier)])
    name_of_pipeline.fit(X_train, y_train)
    y_pred_test = name_of_pipeline.predict(X_test)
    y_pred_train = name_of_pipeline.predict(X_train)

    report = classification_report(y_test, y_pred_test, output_dict=True)
    df = pd.DataFrame(report).transpose()

    print(df)
    print('\n\n')
    print(name_of_pipeline.fit(X_train, y_train))
    print('\n\n')
    print('Training Precision: ', round(precision_score(y_train, y_pred_train),3
))
    print('Testing Precision: ', round(precision_score(y_test, y_pred_test),3))
    print('\n\n')
    print('Training Recall: ', round(recall_score(y_train, y_pred_train),3))
    print('Testing Recall: ', round(recall_score(y_test, y_pred_test),3))
    print('\n\n')
    print('Training Accuracy: ', round(accuracy_score(y_train, y_pred_train),3))
    print('Testing Accuracy: ', round(accuracy_score(y_test, y_pred_test),3))
    print('\n\n')
    print('Training F1-Score: ', round(f1_score(y_train, y_pred_train),3))
    print('Testing F1-Score: ', round(f1_score(y_test, y_pred_test),3))
    return
```

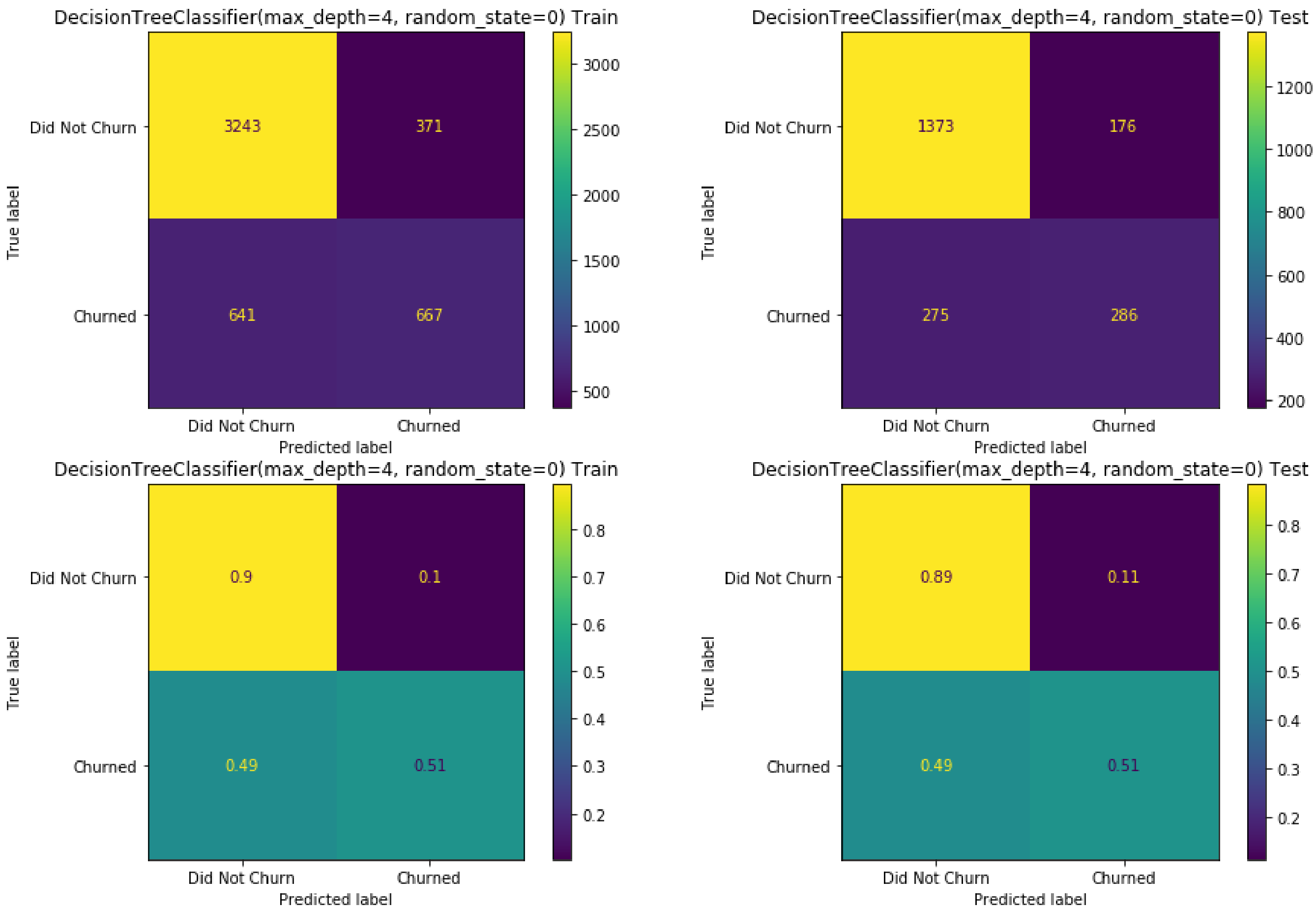
In [25]:
```
# defining the three different classification modeling techniques that will be us
ed throughout this project
dt = DecisionTreeClassifier(random_state=0, max_depth=4)
knn = KNeighborsClassifier()
rf = RandomForestClassifier(random_state=0, n_estimators=100, max_depth=4)
```

In [26]:
```
# CALL THE PIPELINE
pipeline('pipe_1', dt, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score      support
0.0            0.833131  0.886378  0.858930  1549.000000
1.0            0.619048  0.509804  0.559140   561.000000
accuracy       0.786256  0.786256  0.786256     0.786256
macro avg      0.726089  0.698091  0.709035  2110.000000
weighted avg   0.776211  0.786256  0.779223  2110.000000



Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(max_depth=4, random_state=0))])



Training Precision:  0.643
Testing Precision:  0.619



Training Recall:  0.51
Testing Recall:  0.51



Training Accuracy:  0.794
Testing Accuracy:  0.786



Training F1-Score:  0.569
Testing F1-Score:  0.559
```

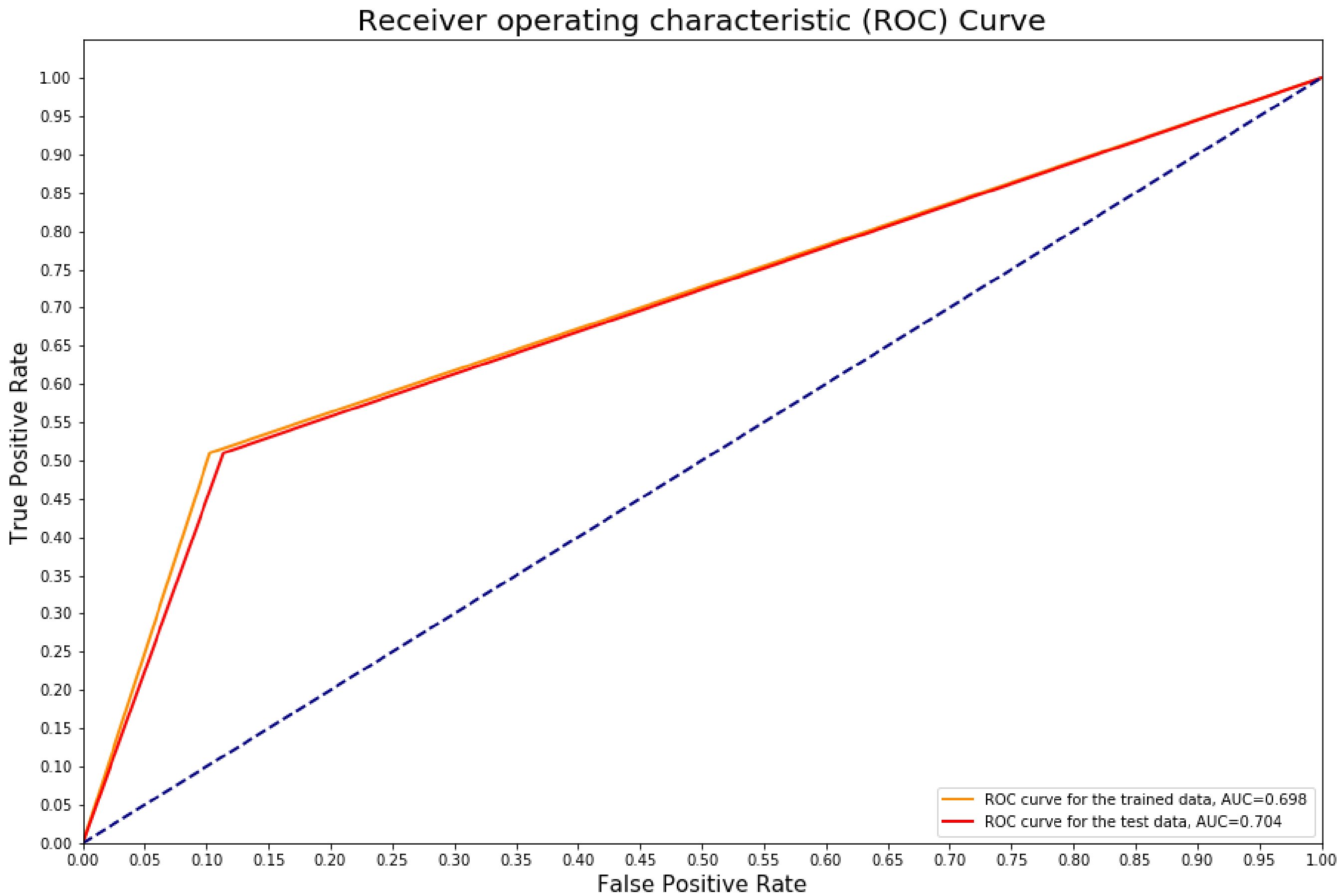
In [27]:
```
def visualizing_confusionmatrix(name_of_pipeline, classifier, X_train, y_train, X
_test, y_test):
    '''Creates confusion matrices of the results from classifier'''
    fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15,10))
    name_of_pipeline = Pipeline([('classifier', classifier)])
    name_of_pipeline.fit(X_train, y_train)
    y_pred_test = name_of_pipeline.predict(X_test)
    y_pred_train = name_of_pipeline.predict(X_train)

    #Plot Training Confusion Matrix
    plot_confusion_matrix(classifier, X_train, y_train, ax=axes[0,0],
                          display_labels=["Did Not Churn","Churned"])
    cm_train = confusion_matrix(y_train, y_pred_train)

    #Plot Normalized Training Confusion Matrix
    plot_confusion_matrix(classifier, X_train, y_train, ax=axes[1,0],
                          display_labels=["Did Not Churn","Churned"],
                          normalize='true')

    #Plot Test Confusion Matrix
    plot_confusion_matrix(classifier, X_test, y_test, ax=axes[0,1],
                          display_labels=["Did Not Churn","Churned"])
    cm_test = confusion_matrix(y_test, y_pred_test)

    #Plot Normalized Test Confusion Matrix
    plot_confusion_matrix(classifier, X_test, y_test, ax=axes[1,1],
                          display_labels=["Did Not Churn","Churned"],
                          normalize='true')

    axes[0,0].title.set_text(f'{classifier} Train')
    axes[0,1].title.set_text(f'{classifier} Test')
    axes[1,0].title.set_text(f'{classifier} Train')
    axes[1,1].title.set_text(f'{classifier} Test')

    plt.grid(False)
    plt.show()
    return
```

```
In [28]: visualizing_confusionmatrix('pipe_1', dt, X_train, y_train, X_test, y_test)
```



In [29]:
```
def createROCCurve(name_of_pipeline, classifier, X_train, y_train, X_test, y_test
):
    '''Creates and plots the ROC'''
    name_of_pipeline = Pipeline([('classifier', classifier)])
    name_of_pipeline.fit(X_train, y_train)
    y_train_score = name_of_pipeline.predict(X_train)

    # Calculate the fpr, tpr, and thresholds for the training set
    train_fpr, train_tpr, thresholds = roc_curve(y_train, y_train_score)

    # Calculate the probability scores of each point in the test set
    y_test_score = name_of_pipeline.predict(X_test)

    # Calculate the fpr, tpr, and thresholds for the test set
    test_fpr, test_tpr, test_thresholds = roc_curve(y_test, y_test_score)

    plt.figure(figsize=(15, 10))
    lw = 2
    plt.plot(train_fpr, train_tpr, color='darkorange', lw=lw,
             label=('ROC curve for the trained data, AUC={:.3f}'.format(auc(test_
fpr, test_tpr))))
    plt.plot(test_fpr, test_tpr, color='red', lw=lw,
             label=('ROC curve for the test data, AUC={:.3f}'.format(auc(train_fp
r, train_tpr))))
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, 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=15)
    plt.ylabel('True Positive Rate', fontsize=15)
    plt.title('Receiver operating characteristic (ROC) Curve', fontsize=20)
    plt.legend(loc='lower right')
    plt.show()
    return
```

```
In [30]: createROCCurve('pipe_1', dt, X_train, y_train, X_test, y_test)
```



Results:

The baseline model is a Decision Tree

- The decision tree classifier has a max_depth=4
- Recall of the test data = 51.0%
- The baseline recall percentage of 51.0% in layman's terms means, "51.0% of customers who churned were correctly classified by the model."

Goals:

Prioritize recall

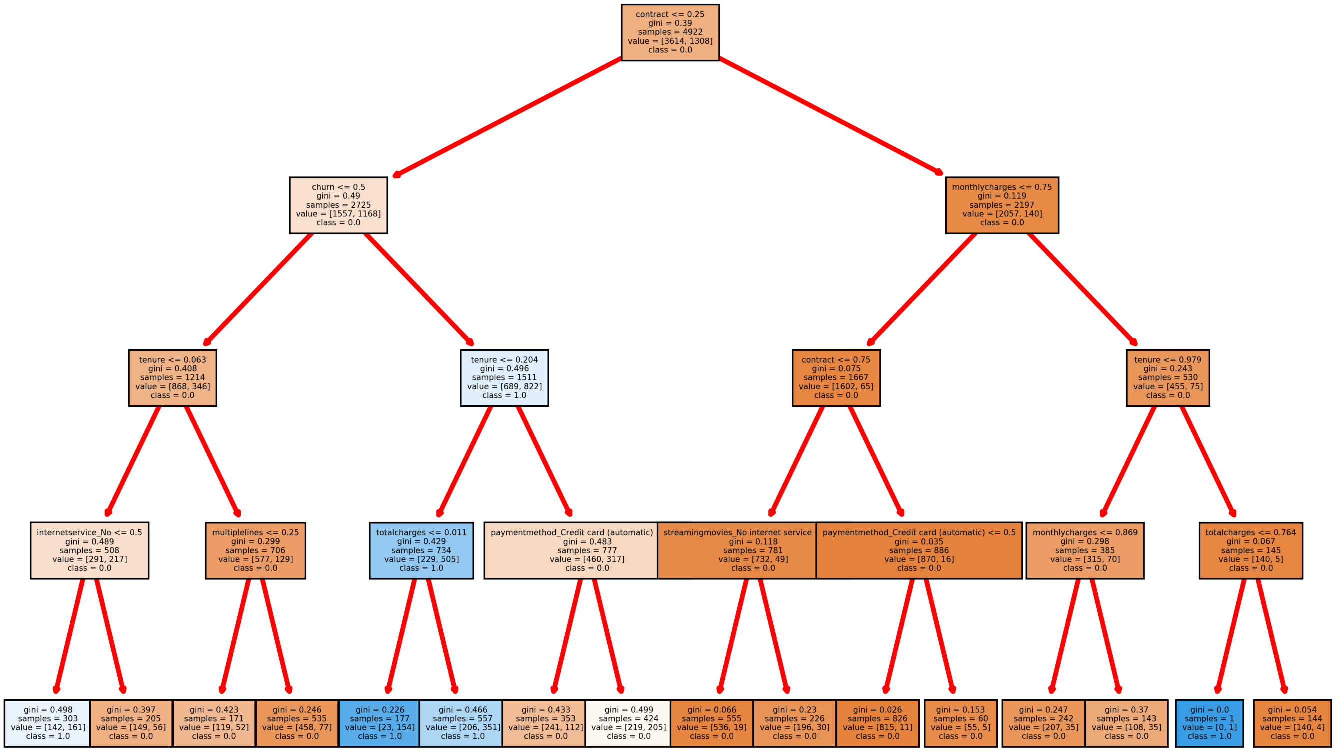
- minimize Type II errors/false negative
- minimize costly situations where the company doesn't identify customers who are going to churn.

Next Steps:

1. Address class imbalance (SMOTE)
2. Simplify the model by identifying and reducing unimportant features:
   - Feature engineering
   - LASSO - least absolute shrinkage and selection operator - L1 Regularization
3. Attempt different types of modeling techniques
   - KNN
   - Random Forests
4. Hyperparameter tuning (GridSearch to create multiple models with different hyperparameters)

In [31]:
```
def create_decisiontree(name_of_pipeline, classifier, dataframe, X_train, y_train
, y):
    '''Creates and plots a decision tree'''
    name_of_pipeline = Pipeline([('classifier', classifier)])
    name_of_pipeline.fit(X_train, y_train)
    fig, axes = plt.subplots(nrows = 1, ncols = 1, figsize = (15,10), dpi=300)
    tree.plot_tree(classifier, fontsize=5, feature_names = dataframe.columns,
                   class_names=np.unique(y).astype('str'), filled = True)
    for decision_box in tree.plot_tree(classifier, fontsize=5, feature_names = da
taframe.columns,
                   class_names=np.unique(y).astype('str'), filled = True):
        arrow = decision_box.arrow_patch
        if arrow is not None:
            arrow.set_edgecolor('red')
            arrow.set_linewidth(3)
    plt.show()
    return
```

```
In [32]: create_decisiontree('pipe_1', dt, df_dummified, X_train, y_train, y)
```



Notes: Important features defining the nodes of the tree:

- 'contract'
- 'monthlycharges'
- 'paymentmethod_Credit card (automatic)'

Model Iteration I - SMOTE and class balancing

```
In [33]: smote = SMOTE(random_state=0, sampling_strategy=1)
         X_train_smote, y_train_smote = smote.fit_sample(X_train, y_train)
         # Preview synthetic sample class distribution
         print('Synthetic sample class distribution: \n')
         print(pd.Series(y_train_smote).value_counts())
```

```
Synthetic sample class distribution:

1.0    3614
0.0    3614
Name: churn, dtype: int64
```

In [34]:
```
pipeline('pipe_2', dt, X_train_smote, y_train_smote, X_test, y_test)
```

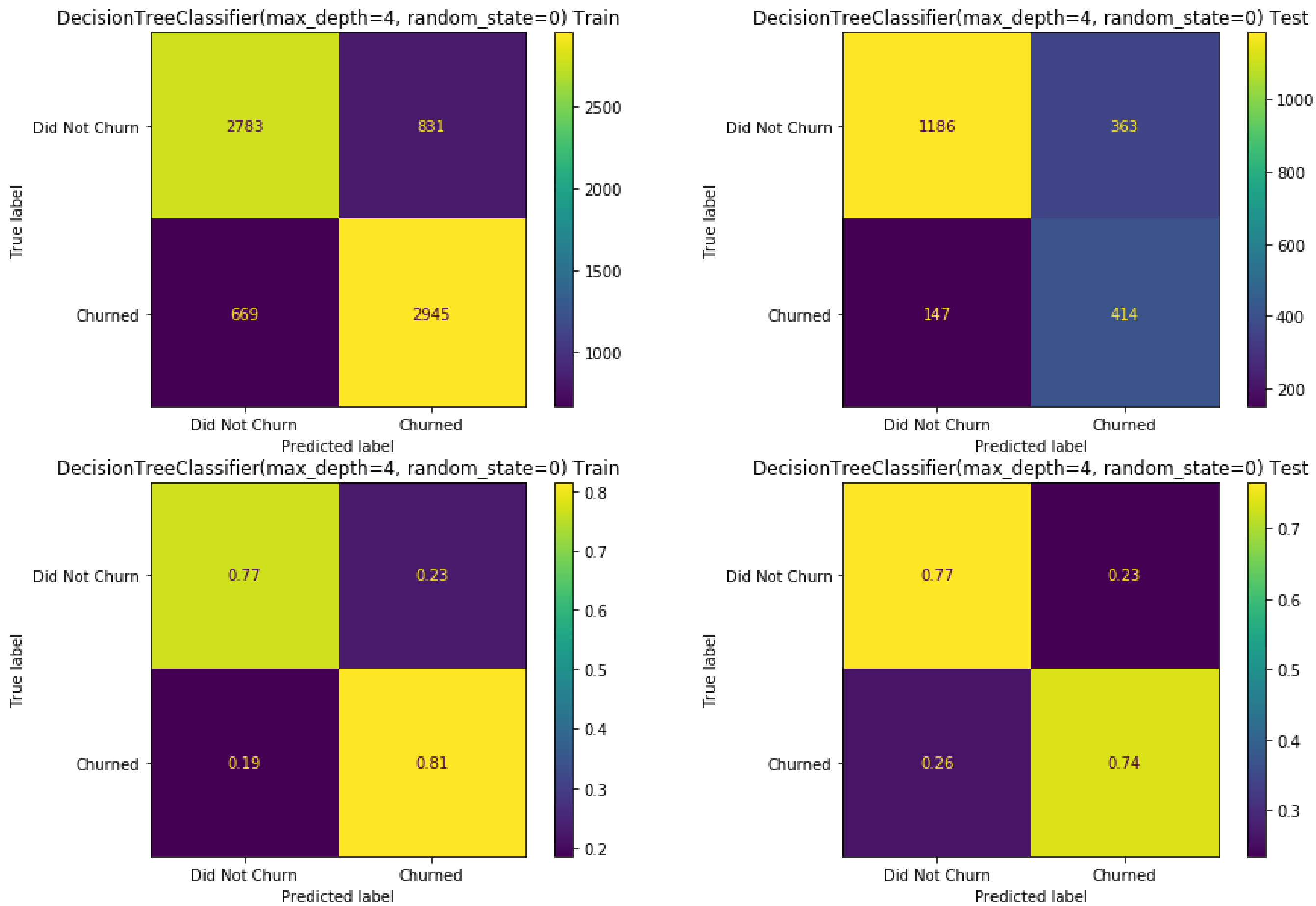
	precision	recall	f1-score	support
0.0	0.889722	0.765655	0.823040	1549.000000
1.0	0.532819	0.737968	0.618834	561.000000
accuracy	0.758294	0.758294	0.758294	0.758294
macro avg	0.711270	0.751812	0.720937	2110.000000
weighted avg	0.794830	0.758294	0.768746	2110.000000

```
Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(max_depth=4, random_state=0))])



Training Precision:  0.78
Testing Precision:  0.533



Training Recall:  0.815
Testing Recall:  0.738



Training Accuracy:  0.792
Testing Accuracy:  0.758



Training F1-Score:  0.797
Testing F1-Score:  0.619
```

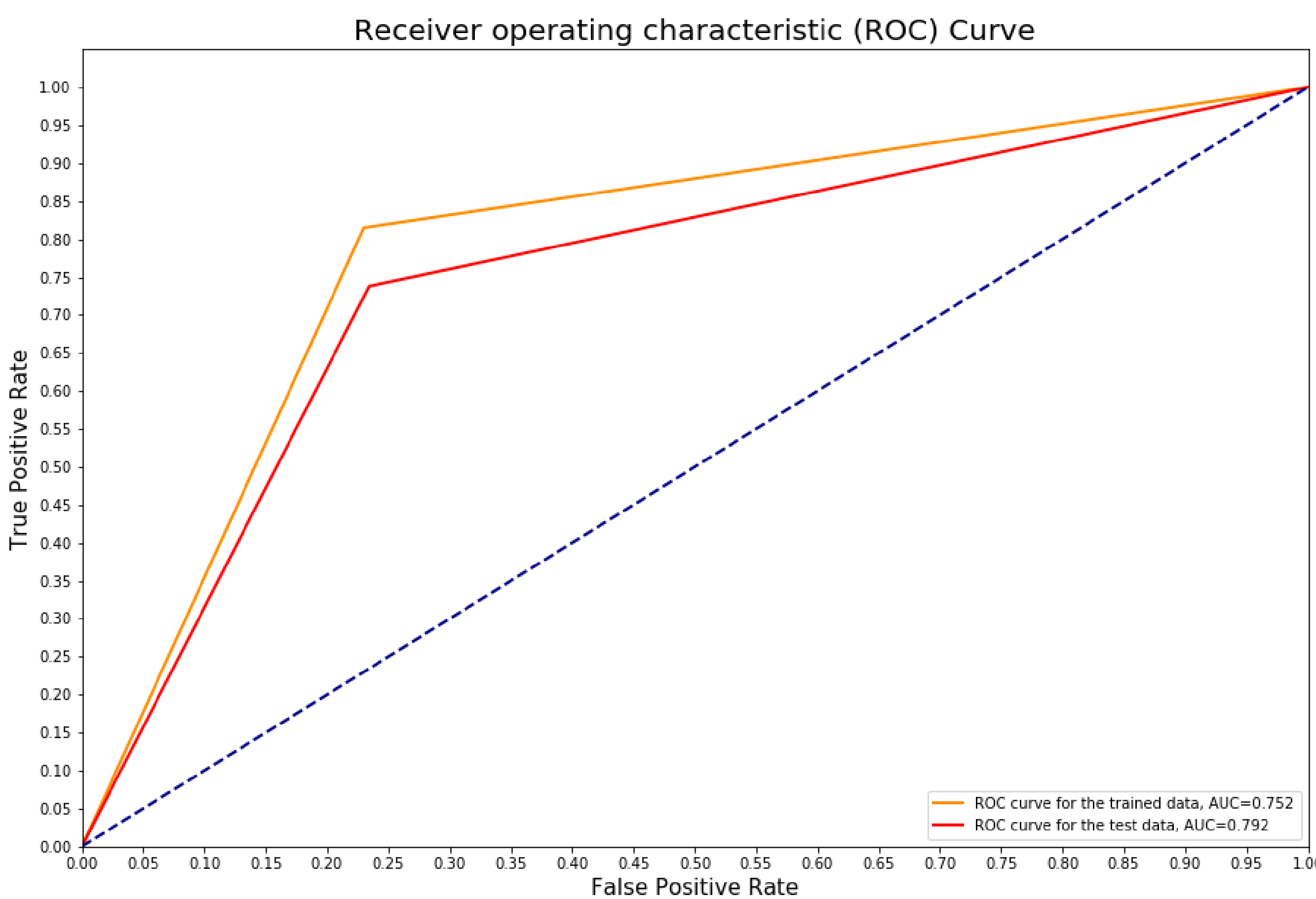
In [35]:
```python
dt_balanced = DecisionTreeClassifier(random_state=0, max_depth=4, class_weight='balanced')

pipeline('pipe_3', dt_balanced, X_train, y_train, X_test, y_test)
```

```
              precision    recall  f1-score     support
0.0            0.908403  0.697870  0.789339  1549.00000
1.0            0.491304  0.805704  0.610398   561.00000
accuracy       0.726540  0.726540  0.726540     0.72654
macro avg      0.699854  0.751787  0.699869  2110.00000
weighted avg   0.797506  0.726540  0.741763  2110.00000



Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(class_weight='balanced', max_depth=4,
                                        random_state=0))])



Training Precision:  0.498
Testing Precision:  0.491



Training Recall:  0.834
Testing Recall:  0.806



Training Accuracy:  0.733
Testing Accuracy:  0.727



Training F1-Score:  0.624
Testing F1-Score:  0.61
```

In [36]:
```
visualizing_confusionmatrix('pipe_2', dt, X_train_smote, y_train_smote, X_test, y_test)
```



In [37]:
```
createROCCurve('pipe_2', dt, X_train_smote, y_train_smote, X_test, y_test)
```



Results:

The first iteration model is a Decision Tree with addressing class imbalance using SMOTE

- The decision tree classifier has a max_depth=4
- Recall of the test data = 73.8%
- The baseline recall percentage of 73.8% in layman's terms means, "73.8% of customers who churned were correctly classified by the model."

Next Steps:

1. Simplify the model by identifying and reducing unimportant features:

- Feature engineering
- LASSO - least absolute shrinkage and selection operator - L1 Regularization

1. Attempt different types of modeling techniques

- KNN
- Random Forests

1. Hyperparameter tuning (GridSearch to create multiple models with different hyperparameters)

Model Iteration II - Feature engineering

In [38]:
```
# can drop 'phoneservice' column because 'multiplelines'
# already asks whether or not customer has phone service
df_featureengineered = df.drop(columns='phoneservice')
```

In [39]:
```
# convert "No / Yes" add-ons services to numerical 0s and 1s
add_on_services = ['onlinesecurity', 'onlinebackup', 'deviceprotection',
                   'techsupport', 'streamingtv', 'streamingmovies']
df_featureengineered[add_on_services] = df[add_on_services].eq('Yes').mul(1)
df_featureengineered[add_on_services] = df_featureengineered[add_on_services].ast
ype(float)
```

In [40]:
```
# create a new column 'numberofaddons', which is the sum of the number of service
add-ons a customer has
df_featureengineered['numberofaddons'] = df_featureengineered[add_on_services].su
m(axis=1)
```

In [41]:
```
# drop all the individual columns relating to the single add-ons
df_featureengineered = df_featureengineered.drop(columns=add_on_services)
```

In [42]:
```
# drop all the column 'tenure' because it is more a qualitative, expressive categ
ory
# moreso about the length of time someone was a customer after the fact
df_featureengineered = df_featureengineered.drop(columns='tenure')
```

In [43]:
```
# create dummy variables
df_featureengineered_dummified = pd.get_dummies(df_featureengineered, drop_first=
True, dtype=int)

# Scale the data
scale = MinMaxScaler()
df_featureengineered_dummified = pd.DataFrame(scale.fit_transform(df_featureengin
eered_dummified.values),
                                              columns=df_featureengineered_dummif
ied.columns,
                                              index=df_featureengineered_dummifie
d.index)
```

In [44]:
```
# Create features and labels
X_fe = df_featureengineered_dummified.drop('churn', axis=1)
y_fe = df_featureengineered_dummified['churn']

# Perform an train_test_split = 70/30 for standard ML
X_fe_train, X_fe_test, y_fe_train, y_fe_test = train_test_split(X_fe, y_fe, test_
size=0.3, random_state=0, stratify=y)
```

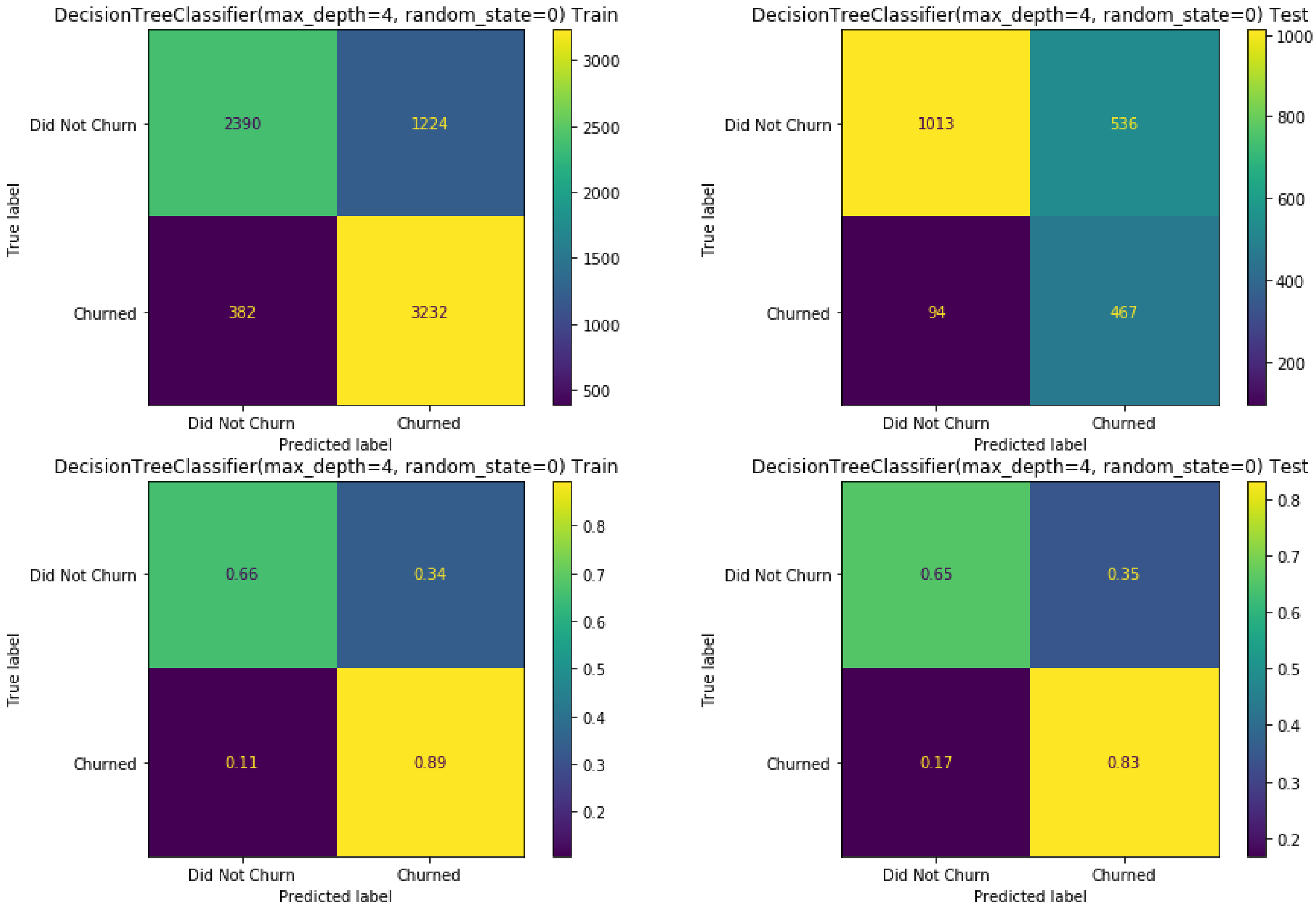
In [45]:
```
# SMOTE the data
smote = SMOTE(random_state=0, sampling_strategy=1)
X_fe_train_smote, y_fe_train_smote = smote.fit_sample(X_fe_train, y_fe_train)
# Preview synthetic sample class distribution
print('Synthetic sample class distribution: \n')
print(pd.Series(y_fe_train_smote).value_counts())
```

```
Synthetic sample class distribution:

1.0    3614
0.0    3614
Name: churn, dtype: int64
```

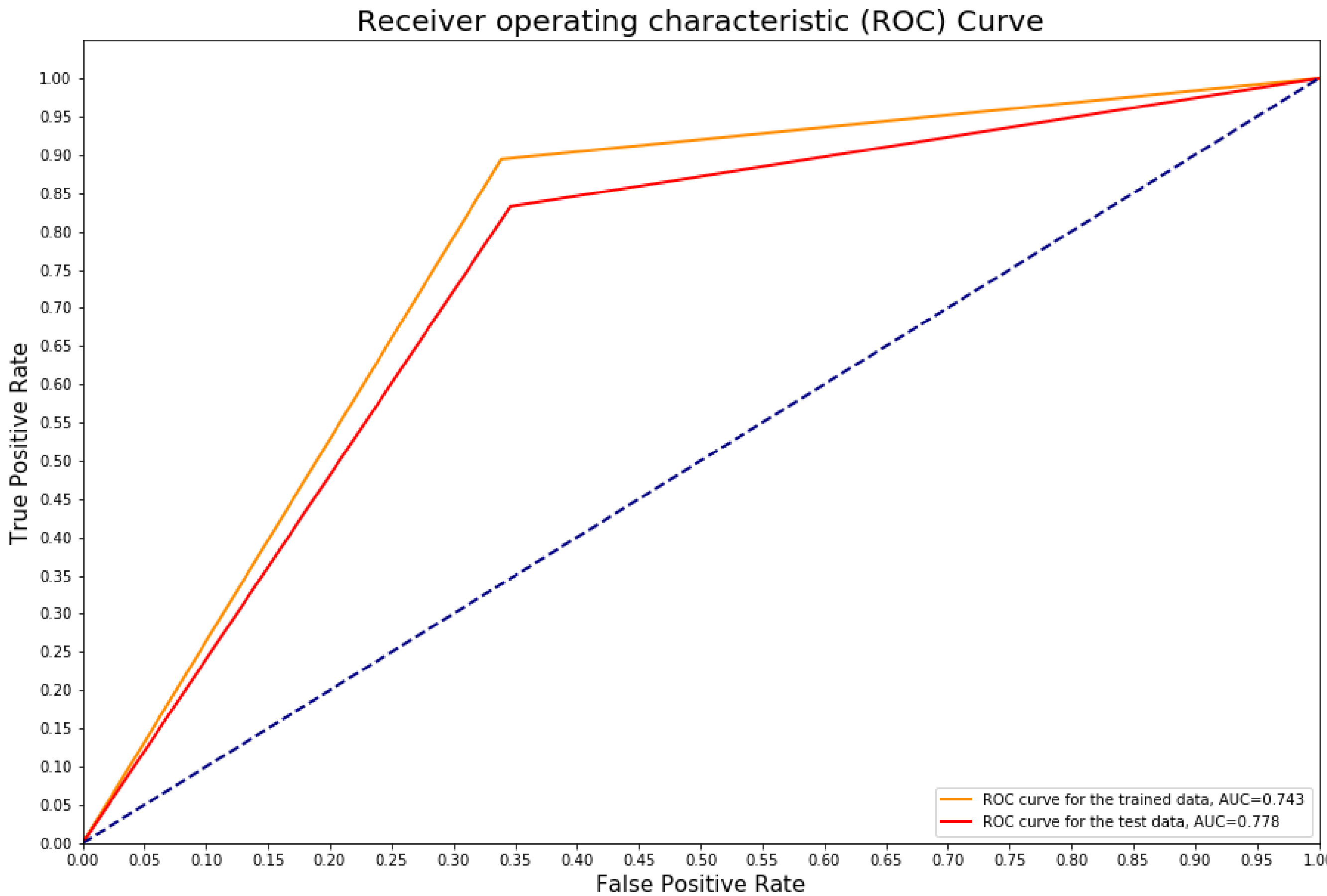
In [46]:
```python
pipeline('pipe_4', dt, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```

```
              precision    recall  f1-score      support
0.0            0.915086  0.653970  0.762801  1549.000000
1.0            0.465603  0.832442  0.597187   561.000000
accuracy       0.701422  0.701422  0.701422     0.701422
macro avg      0.690345  0.743206  0.679994  2110.000000
weighted avg   0.795579  0.701422  0.718768  2110.000000



Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(max_depth=4, random_state=0))])



Training Precision:  0.725
Testing Precision:  0.466



Training Recall:  0.894
Testing Recall:  0.832



Training Accuracy:  0.778
Testing Accuracy:  0.701



Training F1-Score:  0.801
Testing F1-Score:  0.597
```

In [47]:
```
visualizing_confusionmatrix('pipe_4', dt, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```



```
In [48]: createROCCurve('pipe_4', dt, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```



Results:

The second iteration model is a Decision Tree with feature engineering to remove unnecessary variables and addressing class imbalance using SMOTE

- The decision tree classifier has a max_depth=4
- Recall of the test data = 83.2%
- The baseline recall percentage of 83.2% in layman's terms means, "83.2% of customers who churned were correctly classified by the model."
- This model does not have as good as a recall as the previous model.

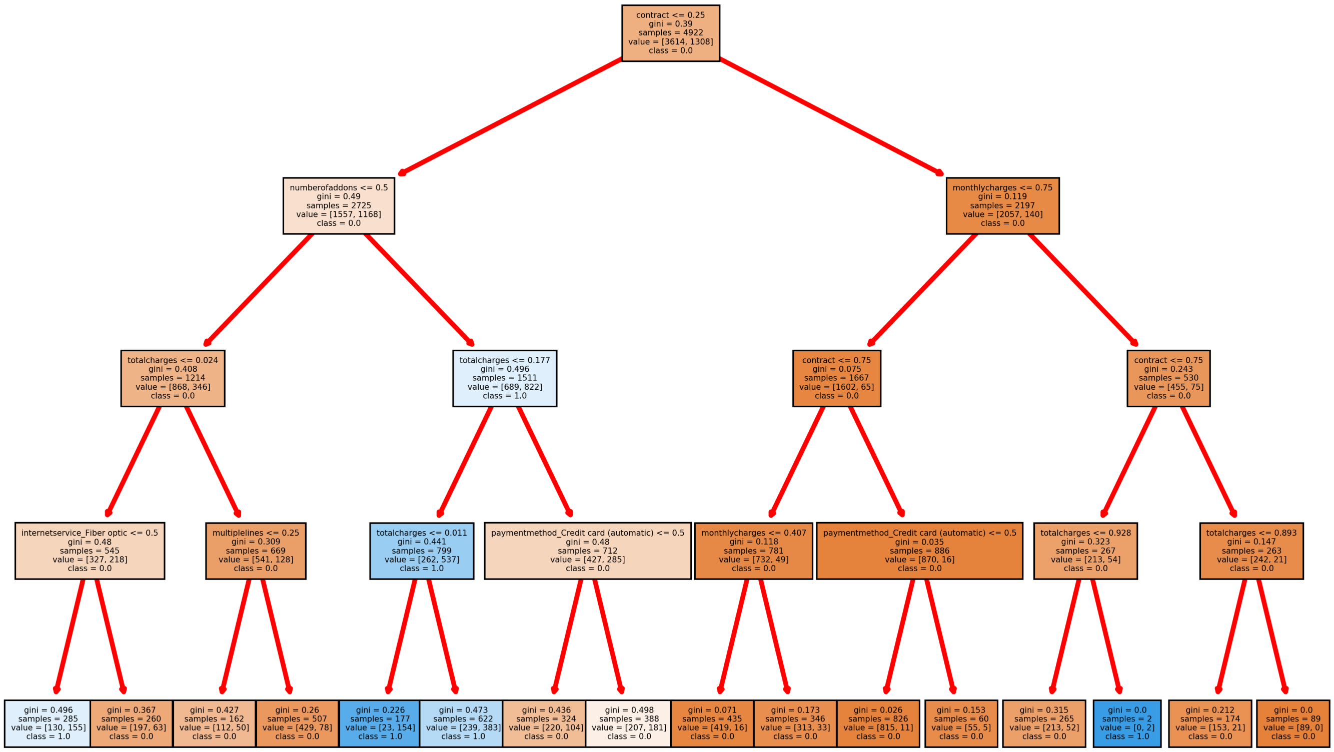
Next Steps:

1. Attempt different types of modeling techniques

- KNN
- Random Forests

1. Hyperparameter tuning (GridSearch to create multiple models with different hyperparameters)

In [49]:
```python
create_decisiontree('pipe_4', dt, df_featureengineered_dummified, X_fe_train, y_fe_train, y_fe)
```



Notes: Important features defining the nodes of the tree:

- 'contract'
- 'monthlycharges'
- 'paymentmethod_Credit card (automatic)'

Model Iteration III - Attempting different classification models

In [50]:
```
# KNN with SMOTE, but no feature engineering
pipeline('pipe_knn', knn, X_train_smote, y_train_smote, X_test, y_test)
```

	precision	recall	f1-score	support
0.0	0.857373	0.686895	0.762724	1549.000000
1.0	0.441887	0.684492	0.537063	561.000000
accuracy	0.686256	0.686256	0.686256	0.686256
macro avg	0.649630	0.685693	0.649893	2110.000000
weighted avg	0.746905	0.686256	0.702726	2110.000000

```
Pipeline(steps=[('classifier', KNeighborsClassifier())])



Training Precision:  0.806
Testing Precision:  0.442



Training Recall:  0.951
Testing Recall:  0.684



Training Accuracy:  0.861
Testing Accuracy:  0.686



Training F1-Score:  0.873
Testing F1-Score:  0.537
```

In [51]:
```python
# KNN with SMOTE and feature engineering
pipeline('pipe_knn_fe', knn, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```

	precision	recall	f1-score	support
0.0	0.860166	0.734668	0.792479	1549.000000
1.0	0.477764	0.670232	0.557864	561.000000
accuracy	0.717536	0.717536	0.717536	0.717536
macro avg	0.668965	0.702450	0.675171	2110.000000
weighted avg	0.758494	0.717536	0.730100	2110.000000

```
Pipeline(steps=[('classifier', KNeighborsClassifier())])



Training Precision:  0.819
Testing Precision:  0.478



Training Recall:  0.929
Testing Recall:  0.67



Training Accuracy:  0.862
Testing Accuracy:  0.718



Training F1-Score:  0.871
Testing F1-Score:  0.558
```

In [52]:

```
# Random Forest with SMOTE, but no feature engineering
pipeline('pipe_rf', rf, X_train_smote, y_train_smote, X_test, y_test)
```

```
              precision    recall  f1-score      support
0.0            0.903614  0.726275  0.805297  1549.00000
1.0            0.509827  0.786096  0.618513   561.00000
accuracy       0.742180  0.742180  0.742180     0.74218
macro avg      0.706721  0.756186  0.711905  2110.00000
weighted avg   0.798915  0.742180  0.755636  2110.00000



Pipeline(steps=[('classifier',
                 RandomForestClassifier(max_depth=4, random_state=0))])



Training Precision:  0.766
Testing Precision:  0.51



Training Recall:  0.861
Testing Recall:  0.786



Training Accuracy:  0.799
Testing Accuracy:  0.742



Training F1-Score:  0.81
Testing F1-Score:  0.619
```

In [53]:
```
# Random Forest with SMOTE and feature engineering
pipeline('pipe_rf_fe', rf, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_te
st)
```

	precision	recall	f1-score	support
0.0	0.903941	0.710781	0.795808	1549.000000
1.0	0.497758	0.791444	0.611149	561.000000
accuracy	0.732227	0.732227	0.732227	0.732227
macro avg	0.700849	0.751112	0.703479	2110.000000
weighted avg	0.795946	0.732227	0.746711	2110.000000

```
Pipeline(steps=[('classifier',
                 RandomForestClassifier(max_depth=4, random_state=0))])
```

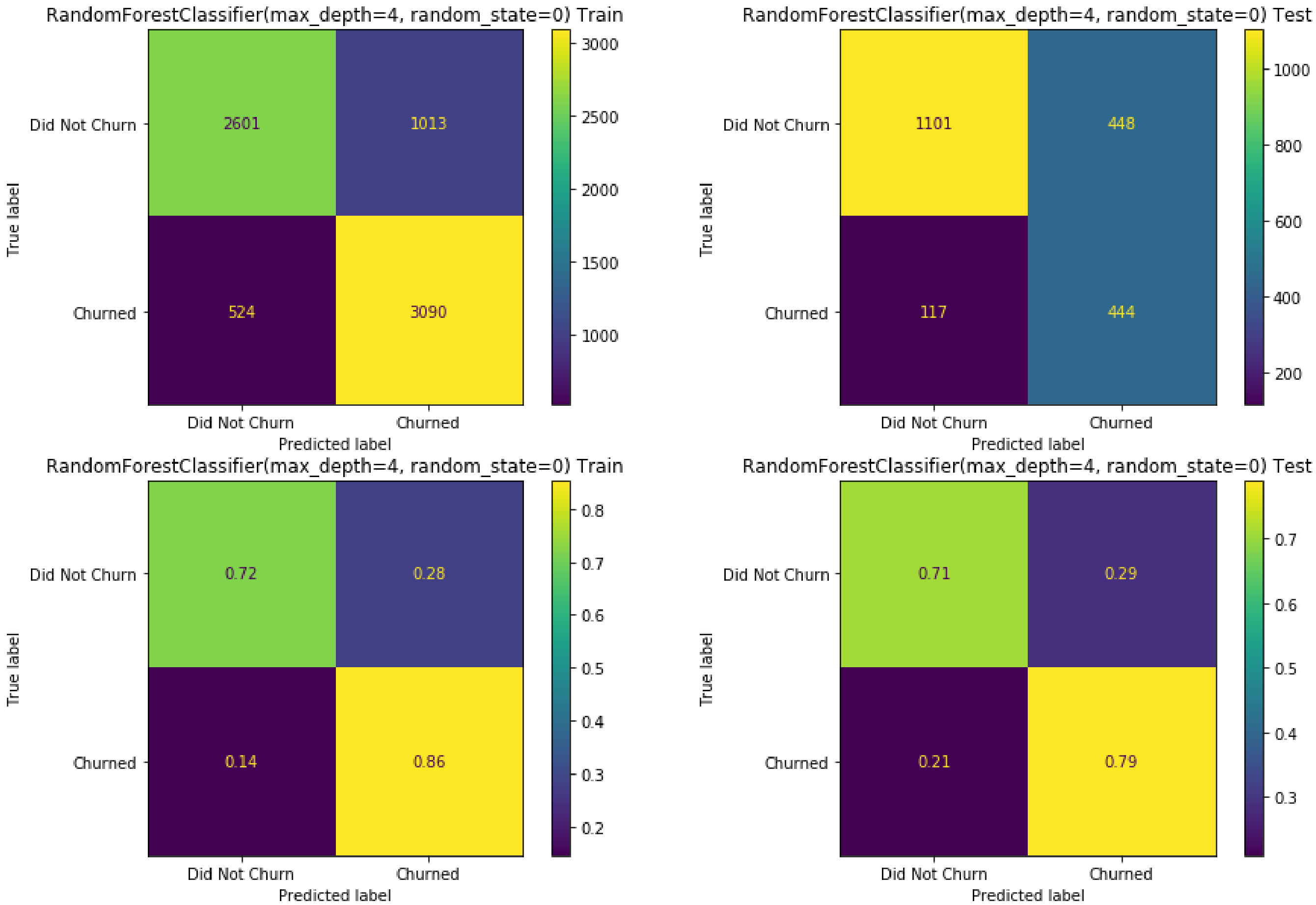
```
Training Precision:  0.753
Testing Precision:  0.498
```

```
Training Recall:  0.855
Testing Recall:  0.791
```

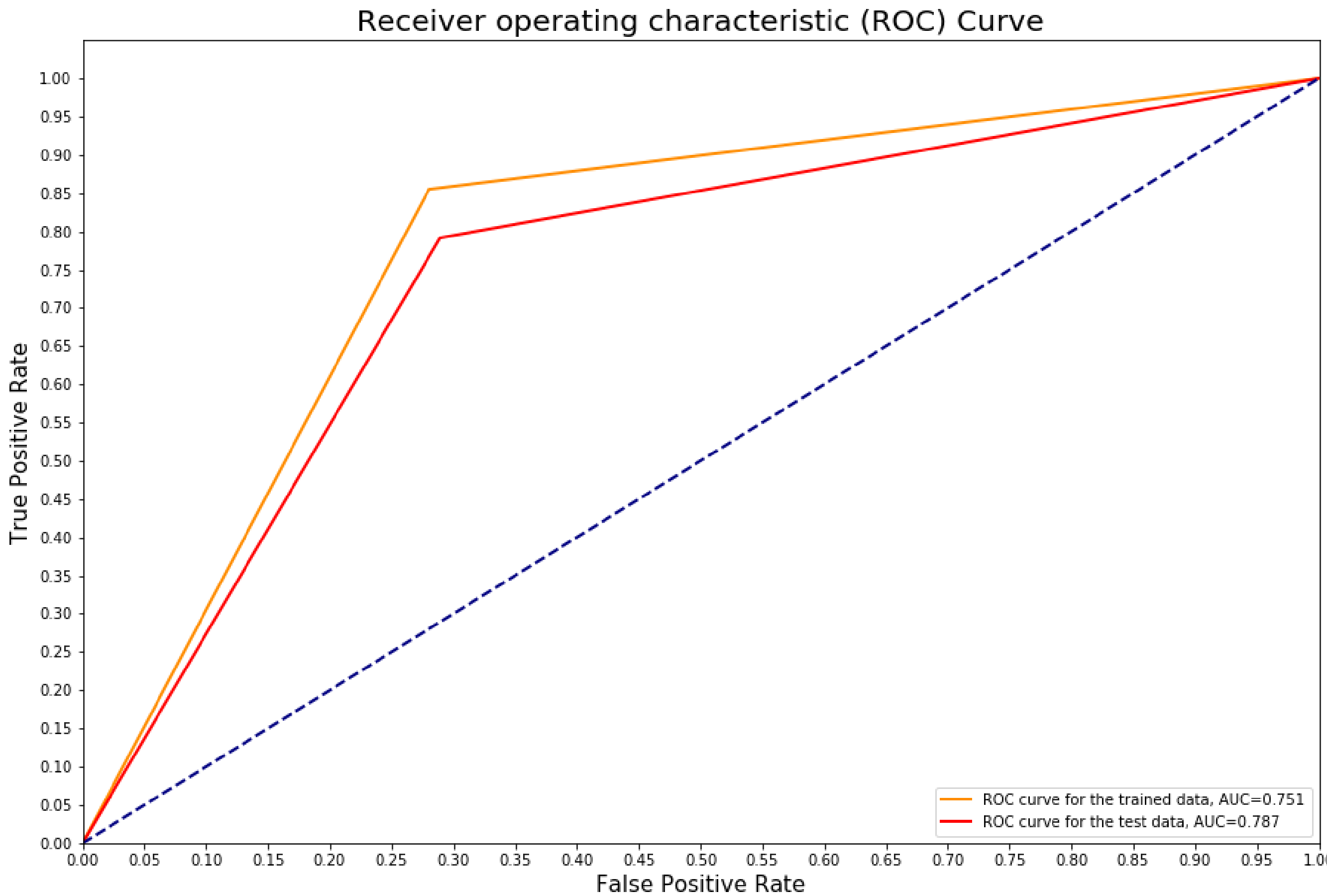
```
Training Accuracy:  0.787
Testing Accuracy:  0.732
```

```
Training F1-Score:  0.801
Testing F1-Score:  0.611
```

In [54]:
```
visualizing_confusionmatrix('pipe_rf_fe', rf, X_fe_train_smote, y_fe_train_smote,
X_fe_test, y_fe_test)
```



In [55]:
```
createROCCurve('pipe_rf_fe', rf, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```



Results:

The third iteration model is a Random Forest with feature engineering to remove unnecessary variables and addressing class imbalance using SMOTE

- The random forest classifier has a max_depth=4
- Recall of the test data = 79.1%
- The recall percentage of 79.1% in layman's terms means, "79.1% of customers who churned were correctly classified by the model."

Next Steps:

1. My best model thus far has been a decision tree; however, I notice that the training recall is substantially larger than the test recall. I am going to try to alter the model parameter, 'class_weights,' to balanced.
2. If this is helpful, I will continue to conduct some hyperparameter tuning using a GridSearch, and create a series of Decision Tree classifiers with different hyperparameters.

Model Iteration IV - Hyperparameter tuning

In [56]:
```
df_featureengineered_dummified.head()
```

Out[56]:

	gender	seniorcitizen	partner	dependents	multiplelines	contract	paperlessbilling	monthlycharges	totalc
0	1.0	0.0	1.0	0.0	0.0	0.0	1.0	0.115423	0.
1	0.0	0.0	0.0	0.0	0.5	0.5	0.0	0.385075	0.
2	0.0	0.0	0.0	0.0	0.5	0.0	1.0	0.354229	0.
3	0.0	0.0	0.0	0.0	0.0	0.5	0.0	0.239303	0.
4	1.0	0.0	0.0	0.0	0.5	0.0	1.0	0.521891	0.

In [57]:
```
from pprint import pprint
# Look at parameters used by our current forest
print('Parameters currently in use:\n')
pprint(dt.get_params())
```

```
Parameters currently in use:

{'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': 4,
 'max_features': None,
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'presort': 'deprecated',
 'random_state': 0,
 'splitter': 'best'}
```

In [58]:
```
# Create the parameter grid based on the results of random search
param_grid = {
    'ccp_alpha': [0, 0.01, 0.02, 0.05, 0.1, 0.5],
    'max_depth': [3, 4, 5],
    'max_features': ['sqrt', None],
    'min_samples_leaf': [1, 2, 3, 10],
    'min_samples_split': [2, 3, 10, 20],
}
# Create a based model
dt = DecisionTreeClassifier(random_state=0)
# Instantiate the grid search model
grid_search = GridSearchCV(estimator = dt, param_grid = param_grid,
                           cv = 3, n_jobs = -1, verbose = 2, scoring='recall')
```

In [59]:
```
# Fit the Decision Tree grid search to the data
grid_search.fit(X_fe_train_smote, y_fe_train_smote)
grid_search.best_params_
```

```
Fitting 3 folds for each of 576 candidates, totalling 1728 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    3.7s
[Parallel(n_jobs=-1)]: Done 928 tasks      | elapsed:   10.4s
[Parallel(n_jobs=-1)]: Done 1728 out of 1728 | elapsed:   17.1s finished
```

Out[59]:
```
{'ccp_alpha': 0.01,
 'max_depth': 3,
 'max_features': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2}
```

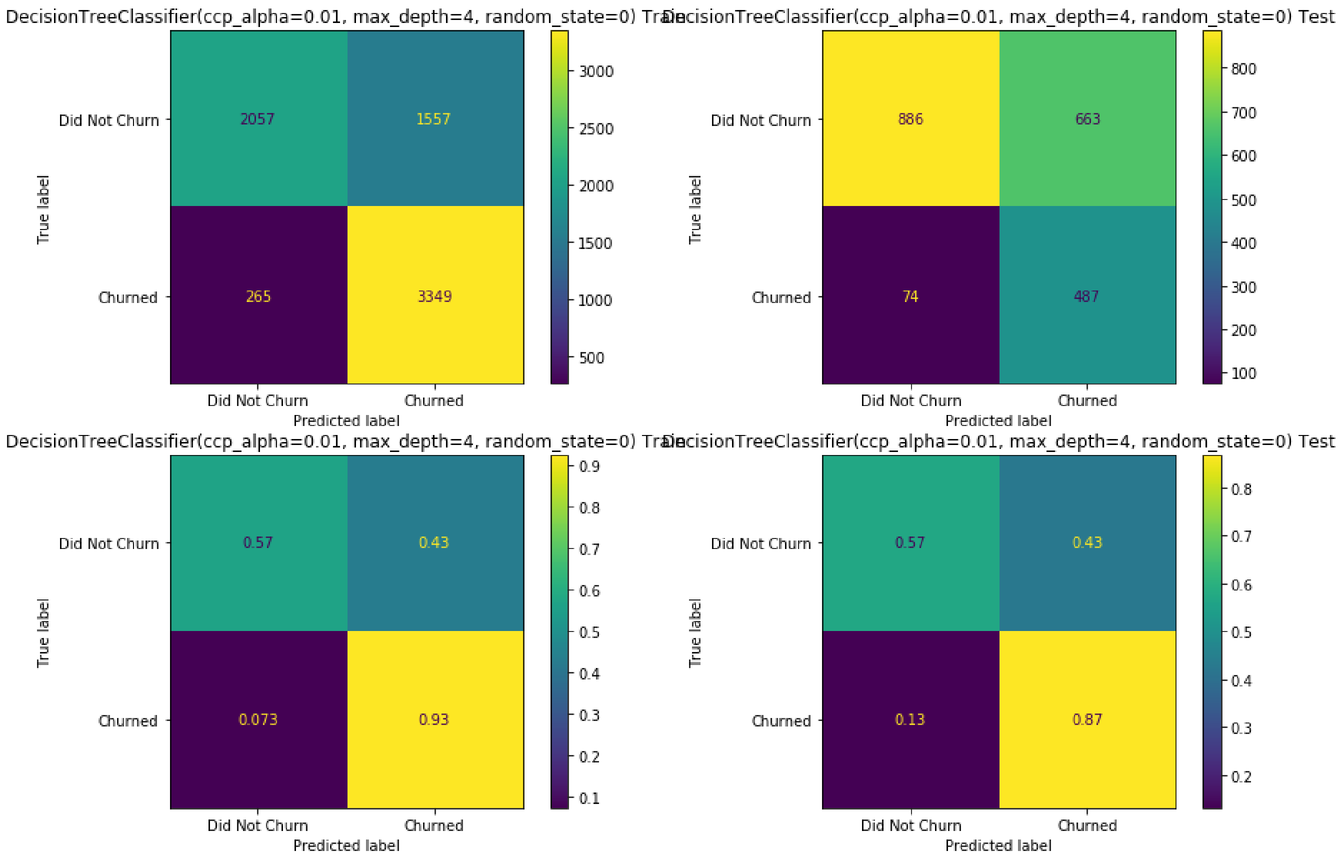
In [60]:
```
dt_best = DecisionTreeClassifier(random_state=0, ccp_alpha=0.01, max_depth=4,
                                 min_samples_split=2, min_samples_leaf=1, max_fea
tures=None)
```

In [61]:
```
pipeline('pipe_dt_fe_best', dt_best, X_fe_train_smote, y_fe_train_smote, X_fe_tes
t, y_fe_test)
```

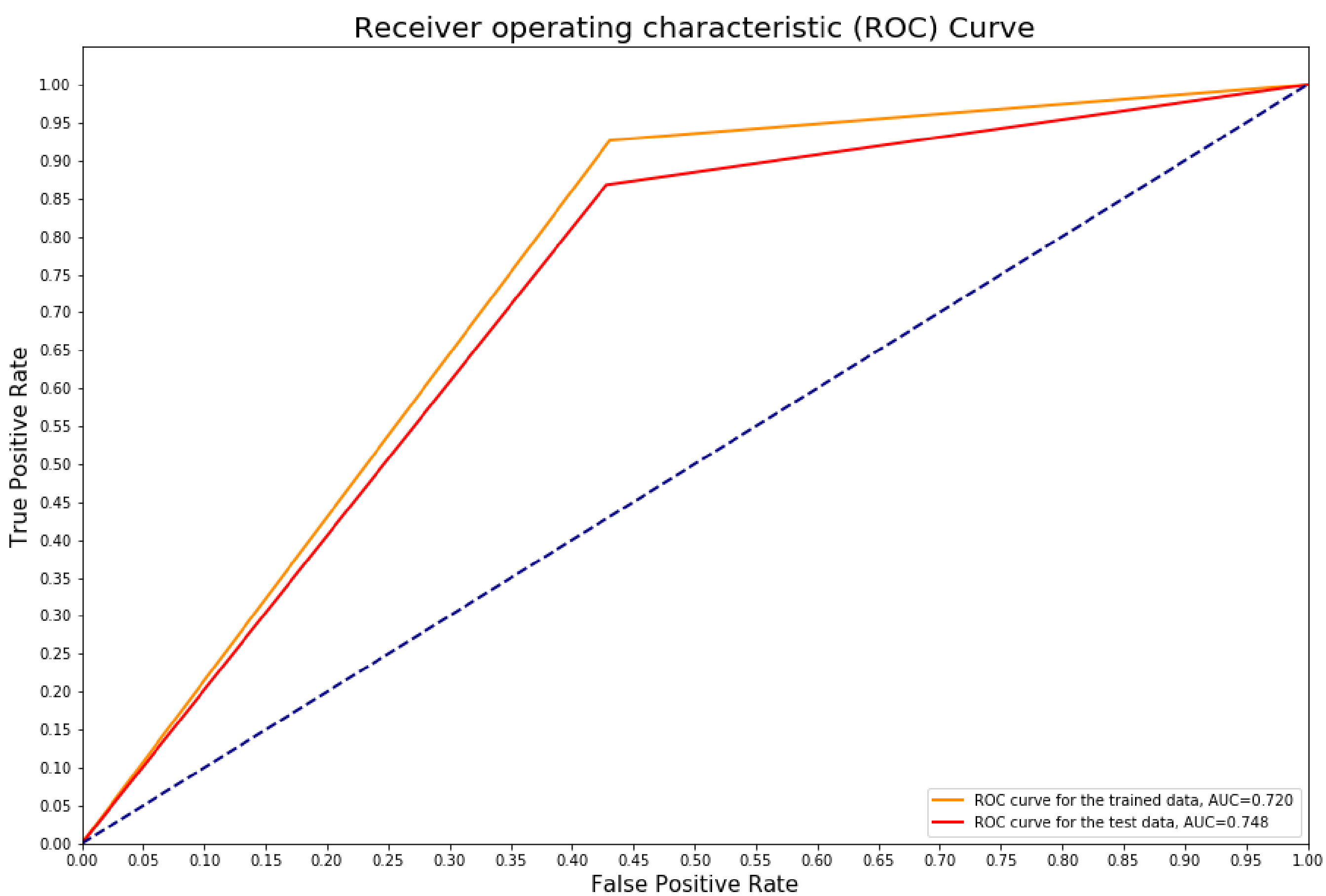
	precision	recall	f1-score	support
0.0	0.922917	0.571982	0.706257	1549.000000
1.0	0.423478	0.868093	0.569258	561.000000
accuracy	0.650711	0.650711	0.650711	0.650711
macro avg	0.673197	0.720037	0.637758	2110.000000
weighted avg	0.790128	0.650711	0.669832	2110.000000

```
Pipeline(steps=[('classifier',
                 DecisionTreeClassifier(ccp_alpha=0.01, max_depth=4,
                                        random_state=0))])



Training Precision:  0.683
Testing Precision:  0.423



Training Recall:  0.927
Testing Recall:  0.868



Training Accuracy:  0.748
Testing Accuracy:  0.651



Training F1-Score:  0.786
Testing F1-Score:  0.569
```

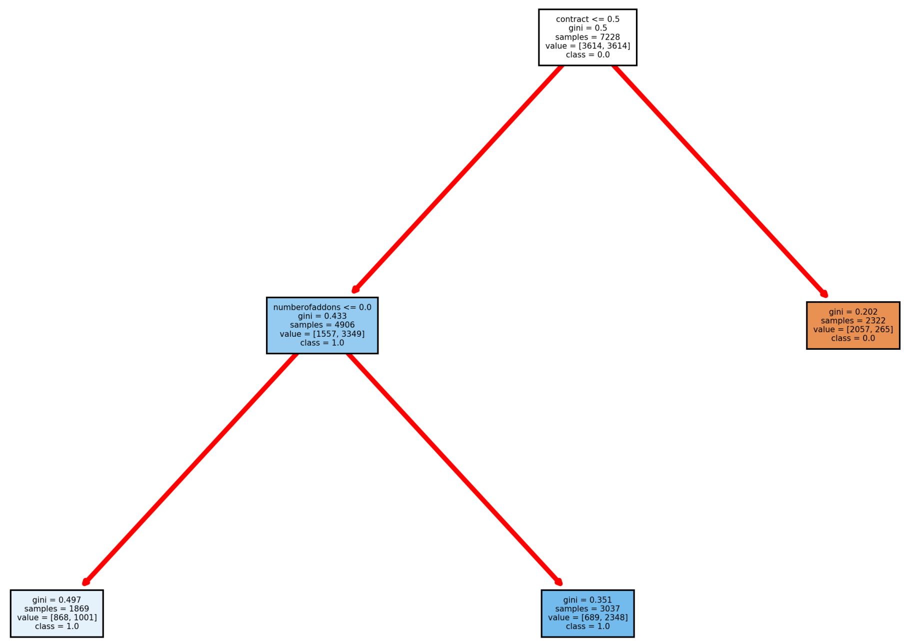
```
In [62]: visualizing_confusionmatrix('pipe_dt_fe_best', dt_best, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```




```
In [63]: createROCCurve('pipe_dt_fe_best', dt_best, X_fe_train_smote, y_fe_train_smote, X_fe_test, y_fe_test)
```



```
In [64]: create_decisiontree('pipe_dt_fe_best', dt_best, df_featureengineered_dummified,
                             X_fe_train_smote, y_fe_train_smote, y_fe)
```



Results:

The fourth iteration model is a Decision Tree with feature engineering to remove unnecessary variables, SMOTE to address class imbalance, and hyperparameter tuning

- The decision tree classifier has a max_depth=3
- One of the most effective parameters that I had adjusted is complexity parameter used for minimal cost-complexity pruning
- Recall of the test data = 86.8%
- The recall percentage of 86.8% in layman's terms means, "86.8% of customers who churned were correctly classified by the model."

Next Steps:

1. Visualize and explore some relationships between the most significant features affecting whether or not a customer churns.

Visualizing the various models and their metrics

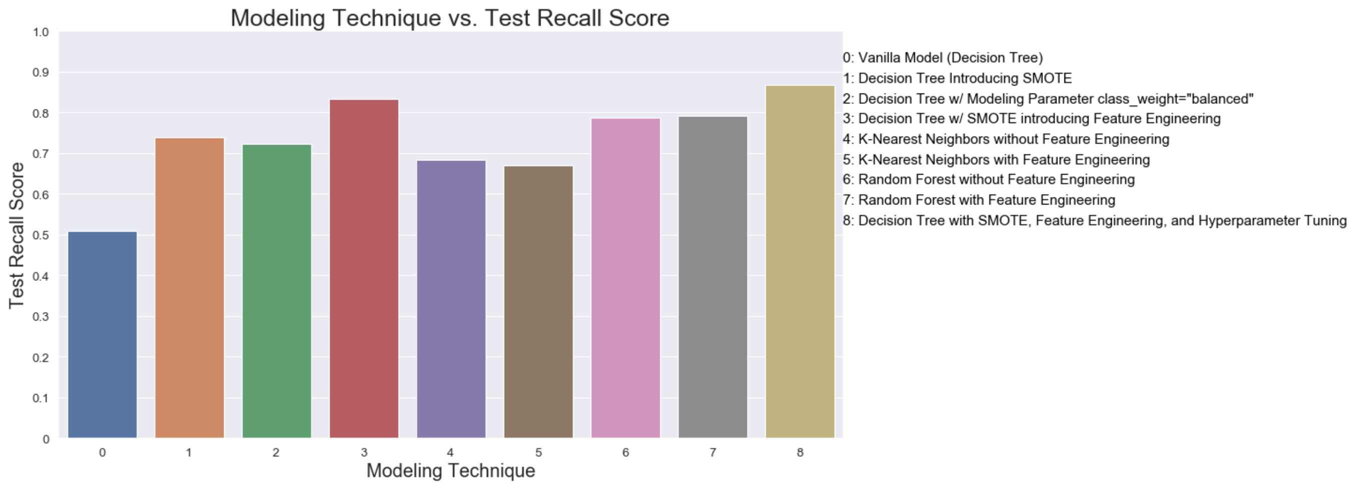
In [65]:
```
data = [['pipe_1', 'Decision Tree', 0.510, 0.510, 0, 'None', 'No', 'Vanilla Model (Decision Tree)'],
        ['pipe_2', 'Decision Tree', 0.815, 0.738, 1, 'SMOTE', 'No', 'Decision Tree introducing SMOTE'],
        ['pipe_3', 'Decision Tree', 0.742, 0.722, 2, 'class_weight="balanced"', 'No', 'Decision Tree with modeling parameter class_weight="Balanced"'],
        ['pipe_4', 'Decision Tree', 0.894, 0.832, 3, 'SMOTE', 'Yes', 'Decision Tree with SMOTE introducing Feature Engineering'],
        ['pipe_knn', 'K-Nearest Neighbors', 0.951, 0.684, 4, 'SMOTE', 'No', 'K-Nearest Neighbors without Feature Engineering'],
        ['pipe_knn_fe', 'K-Nearest Neighbors', 0.929, 0.670, 5, 'SMOTE', 'Yes', 'K-Nearest Neighbors with Feature Engineering'],
        ['pipe_rf', 'Random Forest', 0.861, 0.786, 6, 'SMOTE', 'No', 'Random Forest without Feature Engineering'],
        ['pipe_rf_fe', 'Random Forest', 0.855, 0.791, 7, 'SMOTE', 'Yes', 'Random Forest with Feature Engineering'],
        ['pipe_dt_fe_best', 'Decision Tree', 0.927, 0.868, 8, 'SMOTE', 'Yes', 'Decision Tree with SMOTE, Feature Engineering, and Hyperparameter Tuning']]

# Create the pandas DataFrame
df_pipelines = pd.DataFrame(data, columns = ['Name of Pipeline', 'Name of Classification Modeling Technique',
                                             'Training Recall Score', 'Test Recall Score', 'Model Number',
                                             'Addressed Class Imbalance Using', 'Feature Engineering Implemented', 'Title'])

df_pipelines
```

Out[65]:

	Name of Pipeline	Name of Classification Modeling Technique	Training Recall Score	Test Recall Score	Model Number	Addressed Class Imbalance Using	Feature Engineering Implemented	Titl
0	pipe_1	Decision Tree	0.510	0.510	0	None	No	Vanilla Mod (Decisio Tre
1	pipe_2	Decision Tree	0.815	0.738	1	SMOTE	No	Decision Tre introducir SMOT
2	pipe_3	Decision Tree	0.742	0.722	2	class_weight="balanced"	No	Decision Tre with modelir paramete class_we
3	pipe_4	Decision Tree	0.894	0.832	3	SMOTE	Yes	Decision Tre with SMOT introducir Feature E
4	pipe_knn	K-Nearest Neighbors	0.951	0.684	4	SMOTE	No	K-Neares Neighbor withou Featur Engineerir
5	pipe_knn_fe	K-Nearest Neighbors	0.929	0.670	5	SMOTE	Yes	K-Neares Neighbor with Featur Engineerir
6	pipe_rf	Random Forest	0.861	0.786	6	SMOTE	No	Randor Forest withou Featur Engineerir
7	pipe_rf_fe	Random Forest	0.855	0.791	7	SMOTE	Yes	Randor Forest wit Featur Engineerir
8	pipe_dt_fe_best	Decision Tree	0.927	0.868	8	SMOTE	Yes	Decision Tre with SMOTI Featur Engineering,

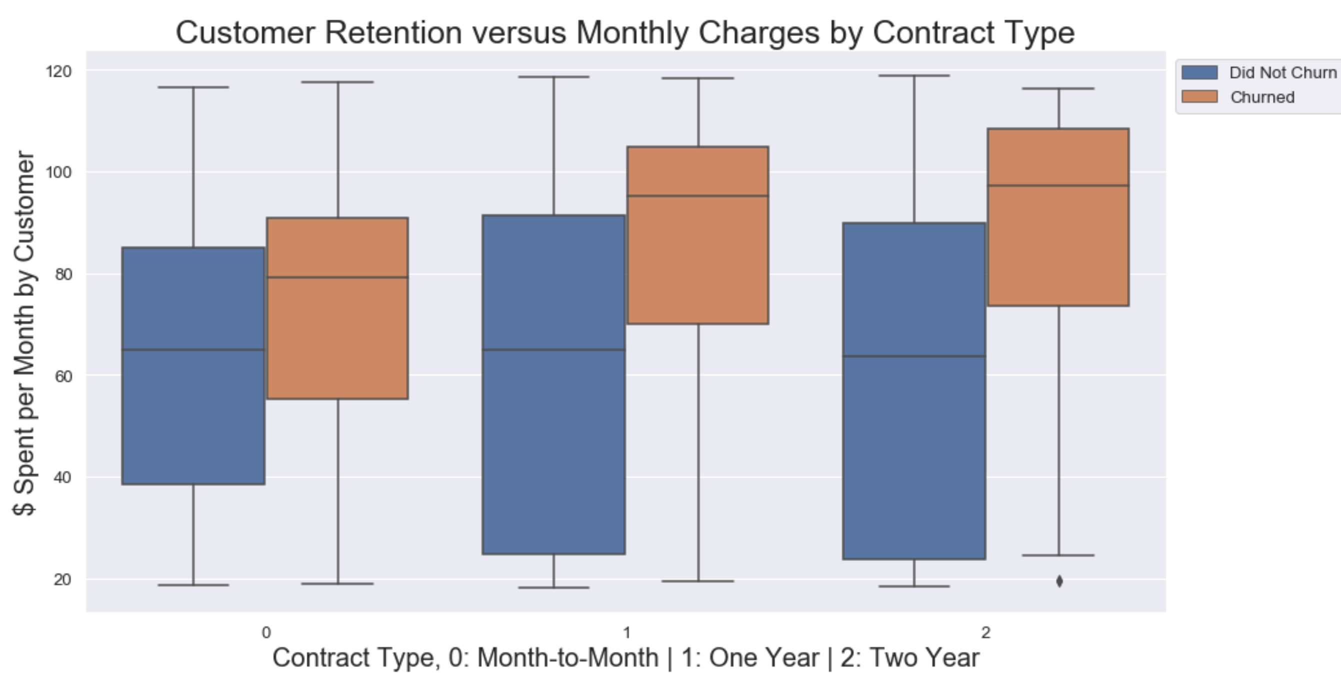
In [66]:

```
plt.figure(figsize=(15,8))
sns.set(font_scale=1.2)
pal = sns.color_palette("husl", 8)
ax = sns.barplot(df_pipelines['Model Number'], df_pipelines['Test Recall Score'])
ax.set_title('Modeling Technique vs. Test Recall Score', fontsize=25)
ax.set_xlabel('Modeling Technique', fontsize=20)
ax.set_ylabel('Test Recall Score', fontsize=20)
ax.text(1, 0.95, '0: Vanilla Model (Decision Tree)', color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.9, '1: Decision Tree Introducing SMOTE', color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.85, '2: Decision Tree w/ Modeling Parameter class_weight="balanced"'
, color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.8, '3: Decision Tree w/ SMOTE introducing Feature Engineering', colo
r='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.75, '4: K-Nearest Neighbors without Feature Engineering', color='bla
ck',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.70, '5: K-Nearest Neighbors with Feature Engineering', color='black'
,
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.65, '6: Random Forest without Feature Engineering', color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.6, '7: Random Forest with Feature Engineering', color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.text(1, 0.55, '8: Decision Tree with SMOTE, Feature Engineering, and Hyperpara
meter Tuning', color='black',
        horizontalalignment='left', fontsize=15,
        verticalalignment='top',
        transform=ax.transAxes)
ax.set_ylim(bottom=0, top=1)
ax.set_yticks((0,.10,.20,.30,.40,.50,.60,.70,.80,.90,1.00))
ax.set_yticklabels((0,.10,.20,.30,.40,.50,.60,.70,.80,.90,1.00));
```



Visualizing Advance Relationships

In [67]:
```
plt.figure(figsize=(15,8))
sns.set(font_scale=1.2)
pal = sns.color_palette("husl", 8)
ax = sns.boxplot(x='contract', y='monthlycharges',
            hue='churn',
            data=df_featureengineered)
ax.set_title('Customer Retention versus Monthly Charges by Contract Type', fontsi
ze=25)
ax.set_xlabel('Contract Type, 0: Month-to-Month | 1: One Year | 2: Two Year', fon
tsize=20)
ax.set_ylabel('$ Spent per Month by Customer', fontsize=20)
leg=plt.legend(bbox_to_anchor=(1, 1))
leg.get_texts()[0].set_text('Did Not Churn')
leg.get_texts()[1].set_text('Churned');
```



In [68]:
```
plt.figure(figsize=(15,8))
sns.set(font_scale=1.2)
pal = sns.color_palette("husl", 8)
ax = sns.boxplot(x='paymentmethod', y='monthlycharges',
            hue='churn',
            data=df)
ax.set_title('Customer Retention versus Monthly Charges by Payment Method', fonts
ize=25)
ax.set_xlabel('Payment Method', fontsize=20)
ax.set_ylabel('$ Spent per Month by Customer', fontsize=20)
leg=plt.legend(bbox_to_anchor=(1, 1))
leg.get_texts()[0].set_text('Did Not Churn')
leg.get_texts()[1].set_text('Churned');
```