Final Project - Predicting Customer Churn in the Wireless Mobile Telecommunication Industry

Sisay Teketele VanJois Scott Zeinab Massudi Gael Ruta Gatera

Table of Contents

Introduction

Wireless telecommunication service is an inseparable part of modern life. Advancement in technology is allowing wireless telecommunication companies to not only enabe their customers to use their services to make calls at a relatively low cost of international and domestic call connections but also provide other functions including providing unlimited text-messages, social media, multi-media messages as well as connecting to internet services at competitivecompletive speed. The understanding of consumer feedback to customer service and interaction at physical stores is paramount to help strengthen the network's product and service offers, boost new acquisitions and minimize churn which is a major problem in a wireless telecommunication company. which consequently has a massive impact on it's productivity and Return On Investment (ROI).

Description of Project

Abstract/ Background

This project was created in understanding and marketing decision about customers who are likely to leave a wireless telecommunication service provider for a competitor is one of its means of survival in already mobile service saturated business. The application of Logistic Regression and Decision Trees, K Nearest Neighbors (KNN), Neural Networks, and Support Vector Machine Learning (SVM) to study customer churn and propose retention decisions for a wireless telecommunication company falls into predictive analyses methods, which enables to better understand the needs (behaviors) of subscribers. The mentioned methods, therefore, were used to predict customer churn and retention decisions. Customers who churn in the first few months of subscribing to services are the most costly to telecommunication companies since they take advantage of different free service offers provided to them during their first 30 days.

Problem Statement

The purpose of this project, using a sample data, is to establish relevant drivers of customers churn in a wireless telecommunications company. The purpose of this project is to examines the relevant customer 's churn drivers (motivations) of mobile phone subscribers in a wireless telecommunication company. This project provides answer to the question that seeks to understand, what are the major drivers of customers churn and retention in a wireless telecommunication company?

Definitions

Customer churn is defined as customers/subscribers switching or termination of their contracts/no contract to another wireless telecommunication provider. Customer churn, therefore, is the movement of subscribers from a service provider to another provider (Geeta & Shashi, 2012). Customer churn is the propensity of customers to cease doing business with a provider. Customer churn is also referred to as customer retention, customer turnover, or customer attrition (Kerdprasop, Kongchai & Kerdprasop, 2013). Losing customers without recovering the associated costs or investments by a business in acquiring them is an acquiredincurred loss to any business. Customer churn occurs when competing companies provides attractive incentives for customers to switch providers. The recent no contract deal with service providers and the implementation of mobile number portability (MNP) is enabling customers to legally retain (port) their original phone number when switching to their new company. The barriers to porting also is even reduced because of no binding contract when customers sign up for a service with a provider. Wireless service providers take customer churn seriously because the cost of retaining current customers is much lower than acquiring new ones (Syam & Hess, 2006). Customer churn, if not understood and mitigated using the right approach, has the potential to result in significant loss or profit to a provider. The prevention of customer churn, especially the profitable subscribers are central to the survival of businesses. A wireless service provider is certain that churning of good customers has an irreversible disadvantage.

Methods

This study focuses on using predictive analysis techniques powered by advanced Machine Learning methods to accurately predict customer churn which through reverse engineering will enable a telecommunication company to increase it's customer retention level.

Essential Libraries & Tools

This section will describe the various modules and library that our methodologies will depend on in order to properly function to manipulate the data. These dependencies were pulled from various sources.

Relative imports

Importing allows a python file or a Python module to access the script from another Python file or module. The `import` statement adds the object to the current scope of your program. (Stackabuse)

- **Scikit-learn (sklearn)**: Contains a number of state-of-the-art machine learning algorithms that were used throughout this project. Scikit-learn is considered the most prominent Python library for machine learning and depends on two other Python Packages, Numpy and SciPy (Muller & Guido, Page 6).
- **OS**: The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux (Python for beginners).
- **Pandas**: This is a Python Library for data wrangling and analysis. Pandas provides a great range of methods to modify and operate on this table; in particular, it allowa SQL-like queries and joins of tables (Muller & Guido, Page 10). This library is mainly used in Step 1 - Data Pre-processing.
- **Numpy**: This is one of the fundamental packages for scientific computing in Python. It contains functionality for multidimensional multidimensional arrays, high-level mathematical functions such as linear algebra operations. In scikit-learn, the Numpy array is the fundamental data structure (Muller & Guido, Page 7).
- **Scipy**: This is a collection of functions for scientific computing in Python. It provides, among other functionality, advanced linear algebra routines, mathematical function optimazation, signal processing, special mathematical functions, and statistical distributions (Muller & Guido, Page 8).
- **'%Matplotlib inline'** performs the necessary behind-the-scenes setup for IPython to work correctly hand in hand with matplotlib (Ipython.readthedocs)
- **Matplotlib**: This is considered the primary scientific plotting library in Python. It provides functions for making publications-quality visualizations such as line charts, histograms, scatter plots and so on (Muller & Guido, Page 9).
- **Seaborn**: This is a library for making statistical graphics in Python. It is built on top of matplotlib and closely integrated with pandas data structures. Seaborn aims to make visualization a central part of exploring and understanding data. Its dataset-oriented plotting functions operate on dataframes and arrays containing whole datasets and internally perform the necessary semantic mapping and statistical aggregation to produce informative plots (seaborn.pydata.org) . Seaborn functionality include but is not limited to:
  - A dataset-oriented API for examining relationships between multiple variables.
  - Specialized support for using categorical variables to show observations or aggregate statistics.
  - Options for visualizing univariate or bivariate distributions and for comparing them between subsets of data.
- **Itertools**: Collection of tools for handling iterators. Simply put, iterators are data types that can be used in a for loop. The most common iterator in Python is the list. (Ridgen)
- **io module**: The default interface to access files and streams in Python 3.0 (docs.python.org)

In [1]:
```
#Importing Dependencies
import os
%matplotlib inline

import sys
print("Python version: {}".format(sys.version))

# Importing panas
import pandas as pd
print("Pandas version: {}".format(pd.__version__))

# Importing numpy
import numpy as np
print("Numpy version: {}".format(np.__version__))

# Importing matplotlib for plotting
import matplotlib as plt
print("MatplotLib version: {}".format(plt.__version__))

# importing seaborn for plotting
import seaborn as sns
sns.set_style('whitegrid')
sns.set()
print("Seaborn version: {}".format(sns.__version__))

# Importing warnings to ignore 'errors'
import warnings
warnings.filterwarnings('ignore')
plt.style.use('classic')

# Importing scipy & libraries that we will use
import scipy as sp
print("Numpy version: {}".format(sp.__version__))
from scipy.stats import skew

# Stats for statistical functions
from scipy import stats

#Importing scipy for stats for pearson distribution Skewness and Kurtosis
from scipy.stats.stats import pearsonr

#Importing scipy for stats for normal distribution
from scipy.stats import norm

# Importing Ipython
import IPython
print("IPython version: {}".format(IPython.__version__))

# Importing sckitit learn & the decision tree module
import sklearn
print("sklearn version: {}".format(sklearn.__version__))
from sklearn import tree

# Import label encoder
from sklearn.preprocessing import LabelEncoder

# Import skewness from stats
from scipy.stats import skew

# Import stats from Scipy
from scipy import stats

# Import stats from Pearson correlation
from scipy.stats.stats import pearsonr

# Import stats  normal distribution
```

```
from scipy.stats import norm

# Import GraphViz
import graphviz

# Import export GraphViz
from sklearn.tree import export_graphviz

# Import export StringIO
from sklearn.externals.six import StringIO

# Import display image
from IPython.display import Image

# Import Pydotplus
import pydotplus

# Importing tenserflow
from tensorflow.keras.utils import to_categorical

import plotly.graph_objs as go
import plotly.offline as py
```

```
Python version: 3.7.1 (default, Dec 14 2018, 13:28:58)
[Clang 4.0.1 (tags/RELEASE_401/final)]
Pandas version: 0.23.4
Numpy version: 1.15.4
MatplotLib version: 3.0.2
Seaborn version: 0.9.0
Numpy version: 1.1.0
IPython version: 7.2.0
sklearn version: 0.21.2

/Users/vanjoisscott/anaconda3/lib/python3.7/site-packages/sklearn/externals/six.py:31: Deprec
ationWarning: The module is deprecated in version 0.21 and will be removed in version 0.23 si
nce we've dropped support for Python 2.7. Please rely on the official version of six (http
s://pypi.org/project/six/).
  "(https://pypi.org/project/six/).", DeprecationWarning)
```

In [2]:
```
#Importing Dependencies
import os
%matplotlib inline

import pandas as pd
import numpy as np


from sklearn import tree
from sklearn.preprocessing import LabelEncoder

import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
plt.style.use('classic')


import seaborn as sns
sns.set_style('whitegrid')
sns.set()
```

Raw Data

In [3]:
```
# Importing the first dataset
sample = pd.read_csv("./finalData.csv", index_col="Store ID", sep =',')
sample.head(5)
```

Out[3]:

Store ID	Date	Account	Market	Service Relate Plans	Tenure	Type of Service Operation	Type of Service Provider	Account Size	New Account	Multi_Service Account	...	TC
20068	1/1/18	908974456	ORA	5B120L1	18	New Connection	Corp	1	True	False	...	
16302	1/1/18	489744457	GSA	5B120L2	22	New Connection	Corp	3	True	True	...	
18830	1/1/18	546685406	HFD	5B120L5	38	New Connection	Corp	4	False	True	...	
17238	1/1/18	997444505	MIA	5B120L1	22	New Connection	Corp	1	True	False	...	
19060	1/1/18	459444459	ALB	TLKTBT25	22	New Connection	Corp	1	True	False	...	

5 rows × 26 columns

In [4]:
```
sample.columns
```

Out[4]:
```
Index(['Date', 'Account', 'Market', 'Service Relate Plans', 'Tenure',
       'Type of Service Operation', 'Type of Service Provider', 'Account Size',
       'New Account', 'Multi_Service Account', 'Total Service Charge',
       'Monthly Service Charge', 'Service Ported', 'TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage'],
      dtype='object')
```

In [5]:
```
# Dataframe shape (columns and rows)
sample.shape
```

Out[5]:
```
(1333823, 26)
```

Data Preprocessing

In [6]:
```
# Dropping duplicates from dataframes
sample = sample.drop_duplicates()
```

In [7]:
```
# Forward Method to replace missing value
sample = sample.fillna(method='ffill')
sample.head()
```

Out[7]:

Store ID	Date	Account	Market	Service Relate Plans	Tenure	Type of Service Operation	Type of Service Provider	Account Size	New Account	Multi_Service Account	...	TC
20068	1/1/18	908974456	ORA	5B120L1	18	New Connection	Corp	1	True	False	...	
16302	1/1/18	489744457	GSA	5B120L2	22	New Connection	Corp	3	True	True	...	
18830	1/1/18	546685406	HFD	5B120L5	38	New Connection	Corp	4	False	True	...	
17238	1/1/18	997444505	MIA	5B120L1	22	New Connection	Corp	1	True	False	...	
19060	1/1/18	459444459	ALB	TLKTBT25	22	New Connection	Corp	1	True	False	...	

5 rows × 26 columns

Label Encoder

The text data we encountered in our data such as from the Service Ported, Multi_Service Account, and New Account columns will need to be processed before being able to appply machine learning algorithms to them which can only read 0's and 1's. LabelEncoder encode labels with a value between 0 and n_classes-1 where n is the number of distinct labels. If a label repeats it assigns the same value to as assigned earlier. It can also be used to transform non-numerical labels (as long as they are hashable and comparable) to numerical labels. In our code we used the "fit_transform(self, y)" method which Fits label encoder and return encoded labels.

In [8]:
```
# label encoding "Service Ported"
le = LabelEncoder()
sample['Service Ported'] = le.fit_transform(sample['Service Ported'].astype(str))
```

In [9]:
```
# label encoding "Multi_Service Account"
le = LabelEncoder()
sample['Multi_Service Account'] = le.fit_transform(sample['Multi_Service Account'].astype(str
))
```

In [10]:
```
# label encoding "New Account"
le = LabelEncoder()
sample['New Account'] = le.fit_transform(sample['New Account'].astype(str))
```

Sampling Data

Returns a random sample of items from an axis of object.

In [11]:
```
sample = sample.rename(columns={'Unnamed: 12': 'Feature Service Charge'})
sample = sample.rename(columns={'Service Ported': 'Service_Ported'})
sample = sample.rename(columns={'Bill Payments': 'Bill_Payments'})
sample = sample.rename(columns={'Staff Count': 'Staff_Count'})
sample = sample.rename(columns={'Outgoing Traffic': 'Outgoing_Traffic'})
sample = sample.rename(columns={'Type of Service Provider': 'Type_of_Service_Provider'})
sample = sample.rename(columns={'New Account': 'New_Account'})
sample = sample.rename(columns={'Multi_Service Account': 'Multi_Service_Account'})
sample = sample.rename(columns={'Multi_Service Account': 'Multi_Service_Account'})
sample = sample.rename(columns={'Feature Service Charge': 'Feature_Service_Charge'})
sample = sample.rename(columns={'Account Size': 'Account Size'})
sample = sample.rename(columns={'Monthly Service Charge': 'Monthly_Service_Charge'})
sample = sample.rename(columns={'Account Size': 'Account_Size'})
sample = sample.rename(columns={'Monthly Service Charge': 'Monthly_Service_Charge'})
sample = sample.rename(columns={'Type of Service Operation': 'Type_of_Service_Operation'})
sample = sample.rename(columns={'Service Relate Plans': 'Service_Relate_Plans'})
sample = sample.rename(columns={'Type of Service Operation': 'Type_of_Service_Operation'})
sample = sample.rename(columns={'Store ID': 'Store_ID'})
```

In [12]:
```
# The first five rows of the sample data
sample.head(5)
```

Out[12]:

Store ID	Date	Account	Market	Service_Relate_Plans	Tenure	Type_of_Service_Operation	Type_of_Service_Provide
20068	1/1/18	908974456	ORA	5B120L1	18	New Connection	Cor
16302	1/1/18	489744457	GSA	5B120L2	22	New Connection	Cor
18830	1/1/18	546685406	HFD	5B120L5	38	New Connection	Cor
17238	1/1/18	997444505	MIA	5B120L1	22	New Connection	Cor
19060	1/1/18	459444459	ALB	TLKTBT25	22	New Connection	Cor

5 rows × 26 columns

In [13]:
```
sample.columns
```

Out[13]:
```
Index(['Date', 'Account', 'Market', 'Service_Relate_Plans', 'Tenure',
       'Type_of_Service_Operation', 'Type_of_Service_Provider', 'Account_Size',
       'New_Account', 'Multi_Service_Account', 'Total Service Charge',
       'Monthly_Service_Charge', 'Service_Ported', 'TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage'],
      dtype='object')
```

Dropping Features

In [14]:
```
# Dropping features from the dataset
sample = sample.drop(['Type_of_Service_Provider','Service_Relate_Plans','Type_of_Service_Operation','Market', 'Date'], axis= 1)
sample.head()
```

Out[14]:

Store ID	Account	Tenure	Account_Size	New_Account	Multi_Service_Account	Total Service Charge	Monthly_Service_Charge	S
20068	908974456	18	1	1	0	85	85	
16302	489744457	22	3	1	1	135	45	
18830	546685406	38	4	0	1	180	45	
17238	997444505	22	1	1	0	85	85	
19060	459444459	22	1	1	0	45	45	

5 rows × 21 columns

In [15]:
```
# Dropping null values from the dataset
sample1 = sample
```

In [16]:
```
# # Dropping Date column
# sample1 = sample.drop(['Date'], axis=1)
```

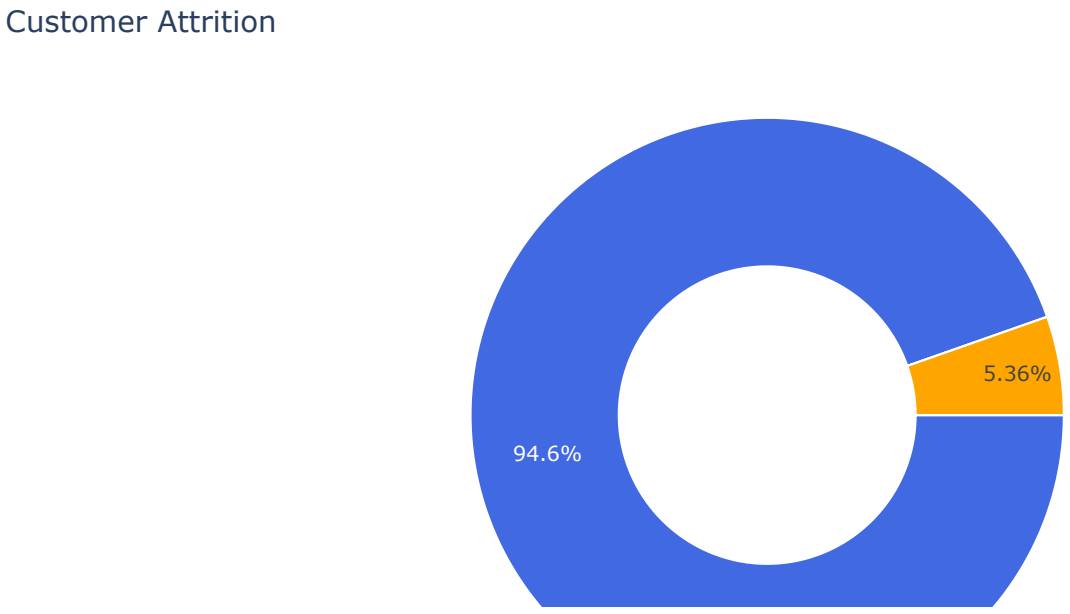
In [17]:
```
# Replacing NaNs with forward fill method
sample1.head()
```

Out[17]:

Store ID	Account	Tenure	Account_Size	New_Account	Multi_Service_Account	Total Service Charge	Monthly_Service_Charge	S
20068	908974456	18	1	1	0	85	85	
16302	489744457	22	3	1	1	135	45	
18830	546685406	38	4	0	1	180	45	
17238	997444505	22	1	1	0	85	85	
19060	459444459	22	1	1	0	45	45	

5 rows × 21 columns

In [25]:
```
#labels
lab = sample1["Service_Ported"].value_counts().keys().tolist()
#values
val = sample1["Service_Ported"].value_counts().values.tolist()

trace = go.Pie(labels = lab ,
               values = val ,
               marker = dict(colors =  [ 'royalblue' ,'orange'],
                             line = dict(color = "white",
                                         width =  1.3)
                            ),
               rotation = 90,
               hoverinfo = "label+value+text",
               hole = .5
              )
layout = go.Layout(dict(title = "Customer Attrition",
                        plot_bgcolor  = "rgb(243,243,243)",
                        paper_bgcolor = "rgb(243,243,243)",
                       )
                   )

data = [trace]
fig = go.Figure(data = data,layout = layout)
py.iplot(fig, image='png')
```



Clean Data

In [26]:
```
# Sample data first five rows
sample1.head()
```

Out[26]:

Store ID	Account	Tenure	Account_Size	New_Account	Multi_Service_Account	Total Service Charge	Monthly_Service_Charge	S
20068	908974456	18	1	1	0	85	85	
16302	489744457	22	3	1	1	135	45	
18830	546685406	38	4	0	1	180	45	
17238	997444505	22	1	1	0	85	85	
19060	459444459	22	1	1	0	45	45	

5 rows × 21 columns

In [27]:
```
sample['Service_Ported'].value_counts(normalize=True)
```

In [28]:
```
sample['New_Account'].value_counts(normalize=True)
```

In [29]:
```
sample['Tenure'].value_counts(normalize=True)
```

Scaling Data

In [18]:
```
#sample = sample.drop(['Market', 'Service_Relate_Plans', 'Type_of_Service_Operation'], axis=1)
sample1.min().values.ravel().reshape(-1,1)
```

Out[18]:
```
array([[400004044],
       [         2],
       [         1],
       [         0],
       [         0],
       [        45],
       [        42],
       [         0],
       [        12],
       [         7],
       [         0],
       [         0],
       [         0],
       [         2],
       [         0],
       [         0],
       [         0],
       [         0],
       [         0],
       [         0],
       [         0]])
```

Scaling Data

Standardization of datasets is a common requirement for many machine learning estimators implemented in scikit-learn; they might behave badly if the individual features do not more or less look like standard normally distributed data: Gaussian with zero mean and unit variance.

In [ ]:
```
# Reading Data and sampling
sample2 = sample1.sample(frac=0.30, replace=True, random_state=1)
```

In [23]:
```
sample2.columns
```

Out[23]:
```
Index(['Account', 'Tenure', 'Account_Size', 'New_Account',
       'Multi_Service_Account', 'Total Service Charge',
       'Monthly_Service_Charge', 'Service_Ported', 'TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage'],
      dtype='object')
```

In [24]:
```
sample2.shape
```

Out[24]:
```
(543, 21)
```

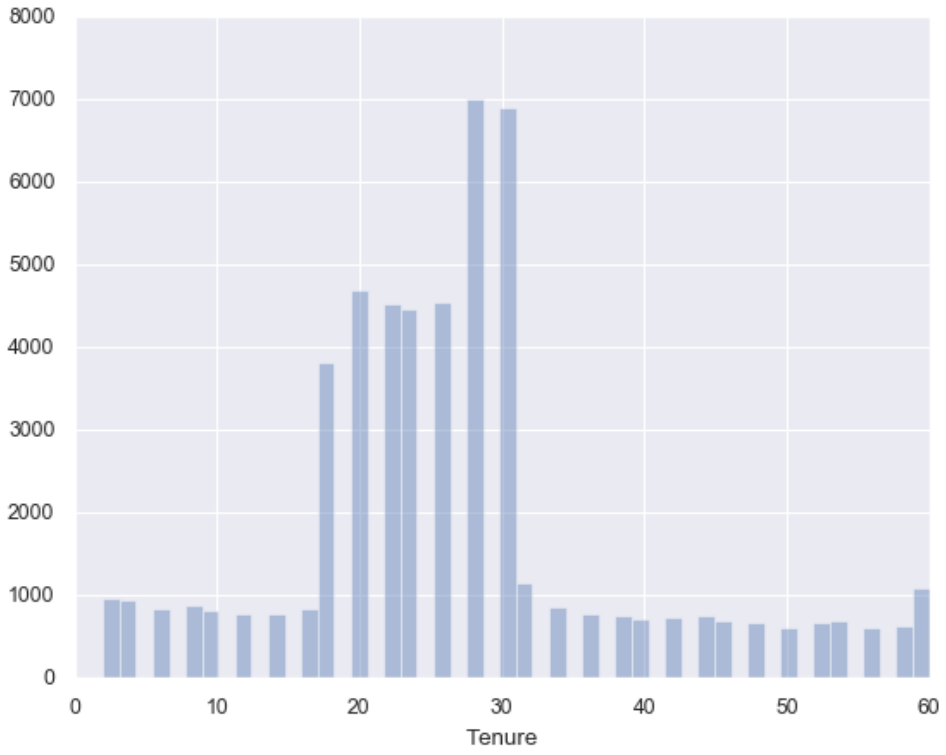
Plots

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. The skewness value can be positive or negative, or undefined.

Kurtosis is a statistical measure that is used to describe the distribution. Whereas skewness differentiates extreme values in one versus the other tail, kurtosis measures extreme values in either tail. Distributions with large kurtosis exhibit tail data exceeding the tails of the normal distribution (e.g., five or more standard deviations from the mean). Distributions with low kurtosis exhibit tail data that are generally less extreme than the tails of the normal distribution.

In [34]:
```
sns.distplot(sample2['Tenure'], kde=False)
```

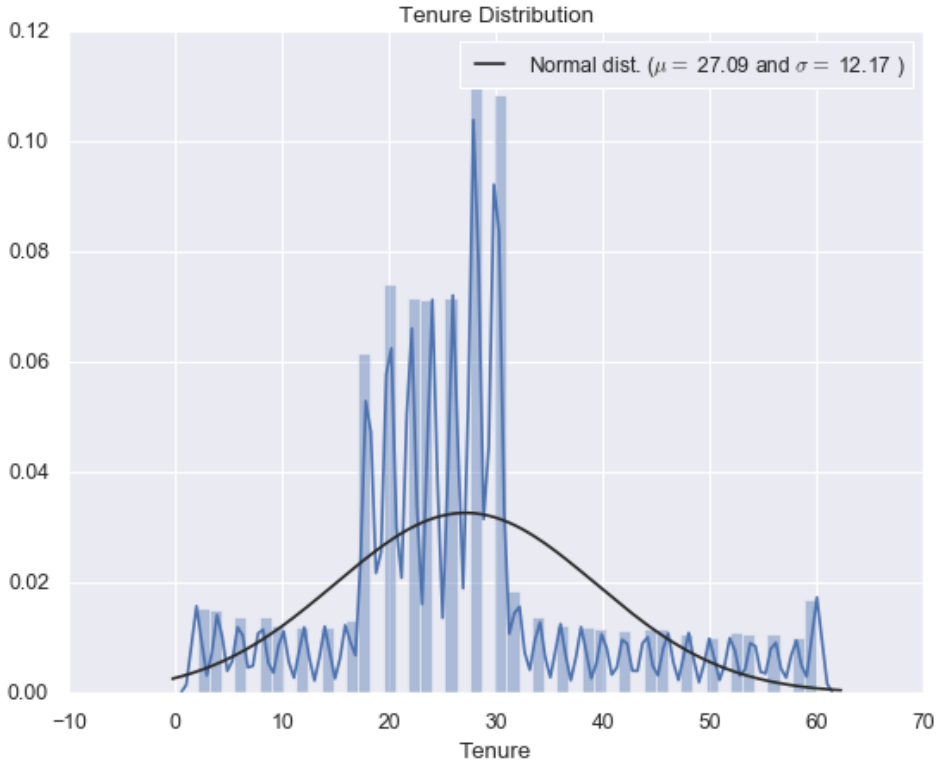
Out[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a46fe47f0>
```



In [35]:
```
# Plot Histogram
sns.distplot(sample1['Tenure'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(sample2['Tenure'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu, sigma)],
           )
#plt.ylabel('Frequency')
plt.title('Tenure Distribution')
```

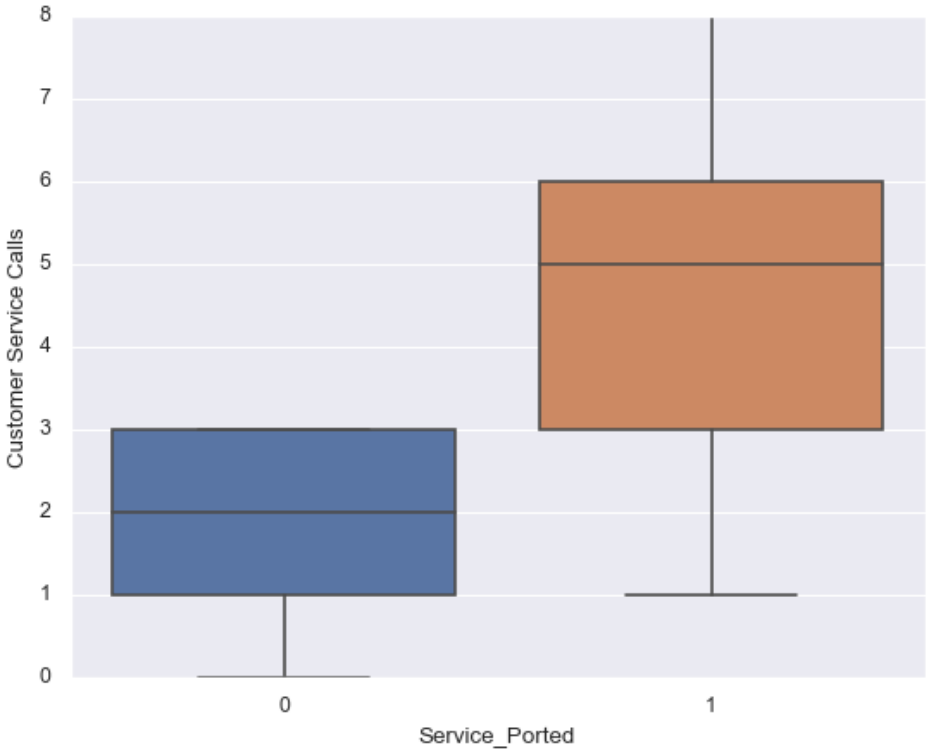
```
 mu = 27.09 and sigma = 12.17
```

Out[35]:
```
Text(0.5, 1.0, 'Tenure Distribution')
```



In [36]:
```
# Plotting box plots
sns.boxplot(x='Service_Ported', y='Customer Service Calls', data=sample2)
```

Out[36]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a4707d0f0>
```



A boxplot is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile (Q1), median, third quartile (Q3), and “maximum”). It can tell you about your outliers and what their values are. It can also tell you if your data is symmetrical, how tightly your data is grouped, and if and how your data is skewed.

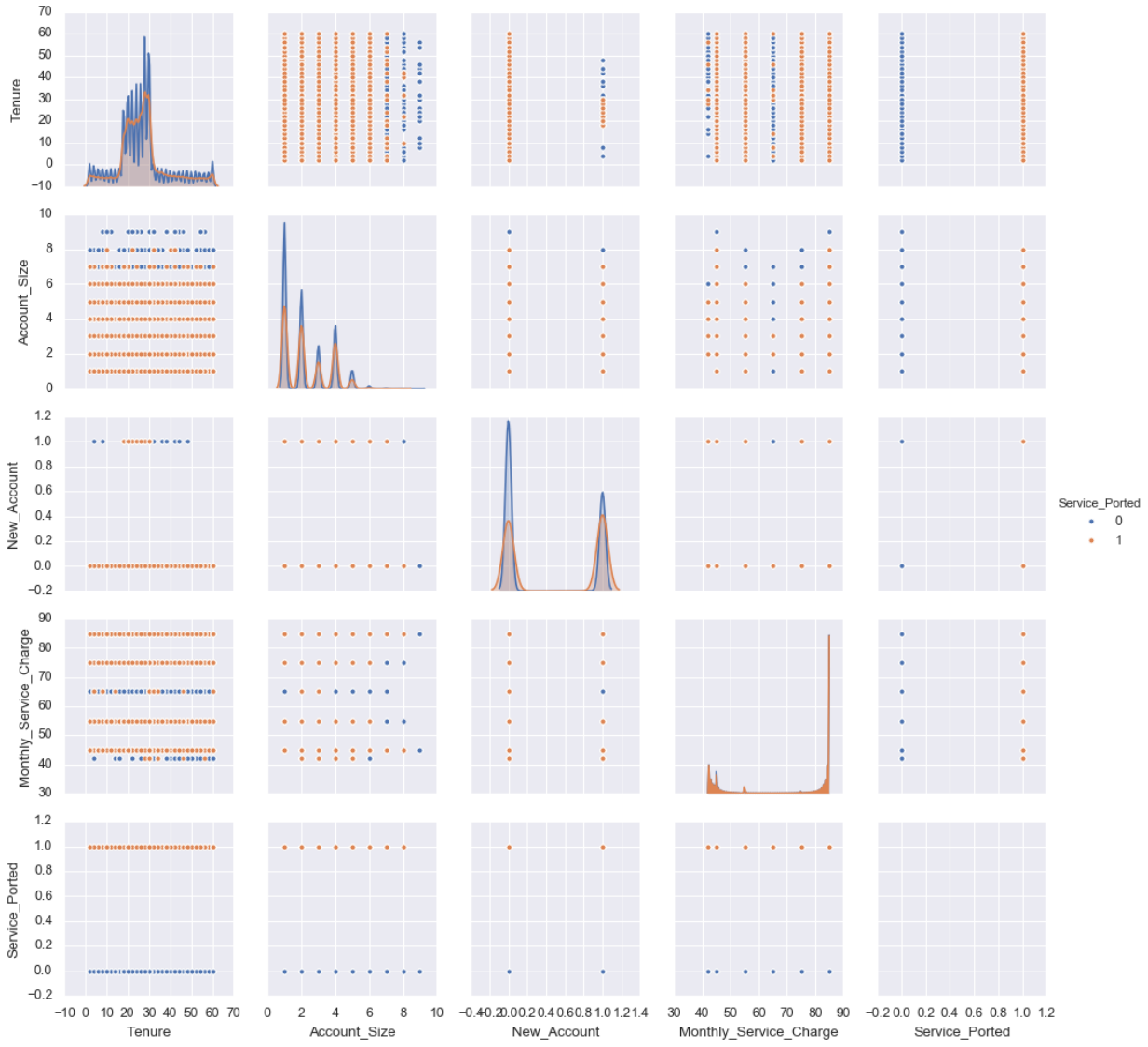
Joint Distribution and Correlation Analysis

In [23]:
```
sample1.columns
```

Out[23]:
```
Index(['Account', 'Tenure', 'Account_Size', 'New_Account',
       'Multi_Service_Account', 'Total Service Charge',
       'Monthly_Service_Charge', 'Service_Ported', 'TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage'],
      dtype='object')
```

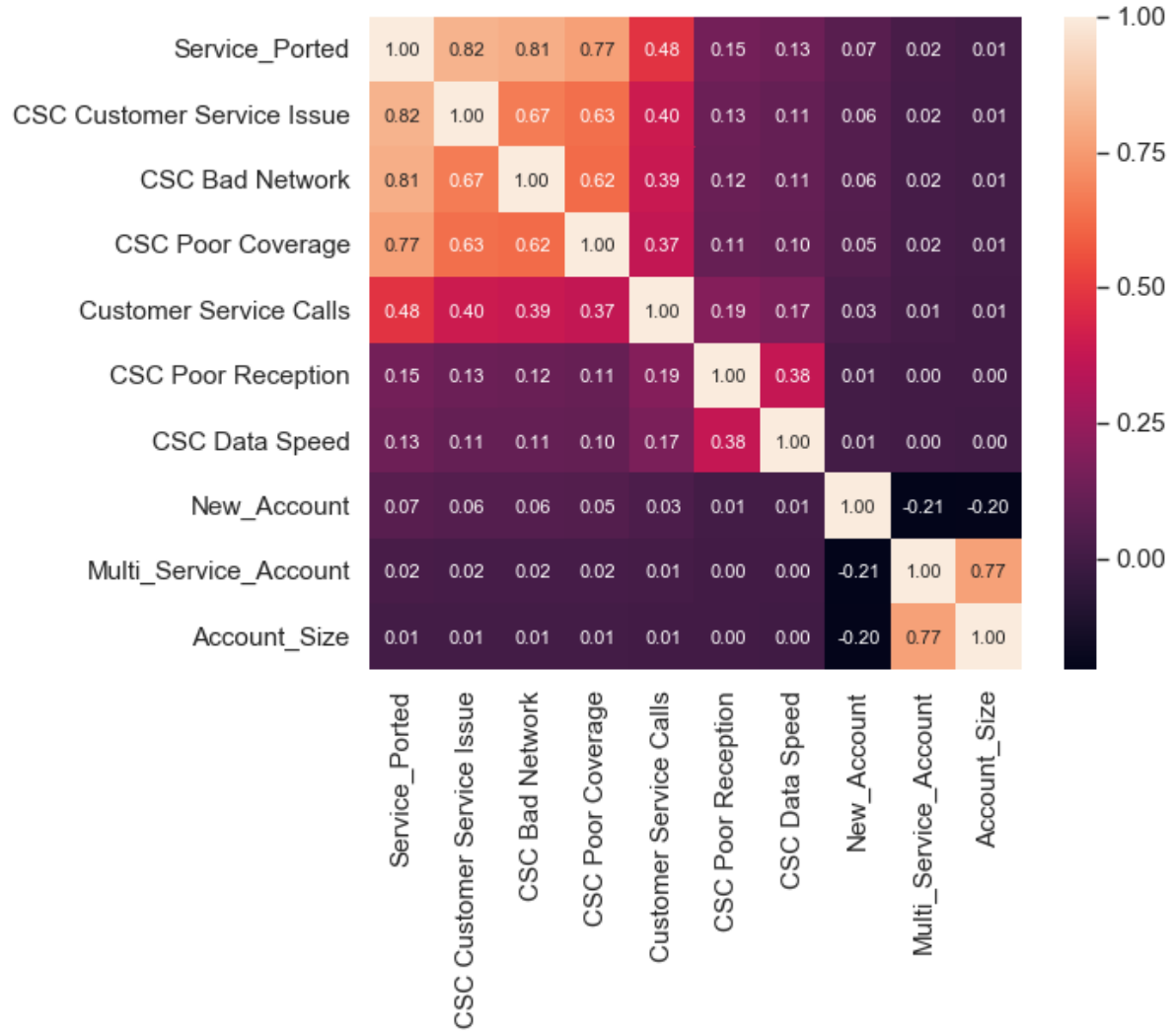
In [24]:
```
#Joint Distribution
example_features = sample1[['Tenure', 'Account_Size', 'New_Account', 'Monthly_Service_Charge'
, 'Service_Ported']]
sns.pairplot(example_features, hue = 'Service_Ported')
```

Out[24]: <seaborn.axisgrid.PairGrid at 0x1a407d55f8>



In [39]:
```
corr = sample2[['Tenure', 'Account_Size', 'New_Account',
       'Multi_Service_Account', 'Monthly_Service_Charge',
       'Service_Ported', 'TOTAL_MSG_COUNT', 'TOTAL_CALLS',
       'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage']].corr()
```

In [40]:
```
# Top 10 Heatmap
k = 10 #number of variables for heatmap
cols = corr.nlargest(k, 'Service_Ported')['Service_Ported'].index
cm = np.corrcoef(sample[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10}, y
ticklabels=cols.values, xticklabels=cols.values)
plt.show()
```



In [41]:
```
most_corr = pd.DataFrame(cols)
most_corr.columns = ['Most Correlated Features']
most_corr
```

Out[41]:

	Most Correlated Features
0	Service_Ported
1	CSC Customer Service Issue
2	CSC Bad Network
3	CSC Poor Coverage
4	Customer Service Calls
5	CSC Poor Reception
6	CSC Data Speed
7	New_Account
8	Multi_Service_Account
9	Account_Size

In [42]:
```
numeric_feats = sample1.dtypes[sample.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = sample1[numeric_feats].apply(lambda x: skew(x.dropna())).sort_values(ascending
=False)
skewness = pd.DataFrame({'Skewed Features' :skewed_feats})
skewness.head()
```

Out[42]:

	Skewed Features
TOTAL_DROPPED_CALLS	113.877763
TOTAL_MSG_COUNT	57.667133
TOTAL_CALLS	18.092953
TOTAL_ANSWERED_CALLS	12.285249
TOTAL_CALL_DURATION_IN_MINUTE	9.093449

In [43]:
```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform".format(skewness.shape[0
]))

from scipy.special import boxcox1p
skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    sample[feat] = boxcox1p(sample[feat], lam)
    sample[feat] += 1
```

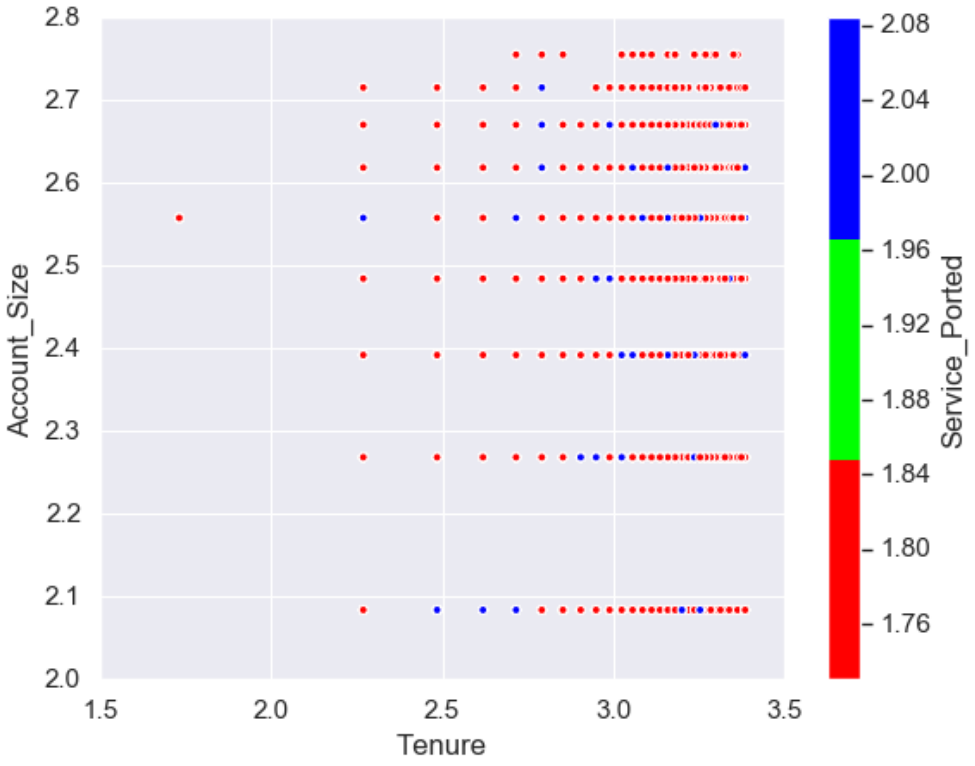
```
There are 21 skewed numerical features to Box Cox transform
```

Churn Visualization

In [44]:
```
# create a custom colormap
from matplotlib.colors import ListedColormap
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])
```

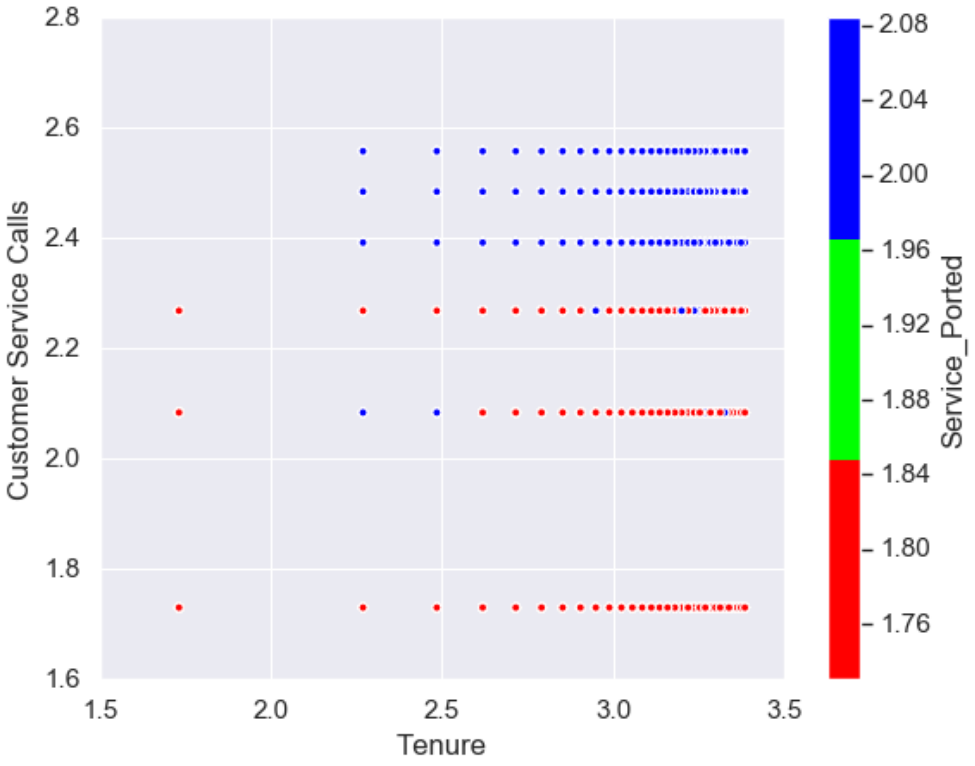
In [66]:
```
# create a scatter plot of Tenure versus Account_Size and color by Service_Ported
sample.plot(kind='scatter', x = 'Tenure', y='Account_Size', c='Service_Ported', colormap=cmap
_bold)
```

Out[66]: <matplotlib.axes._subplots.AxesSubplot at 0x14037a2a0b8>



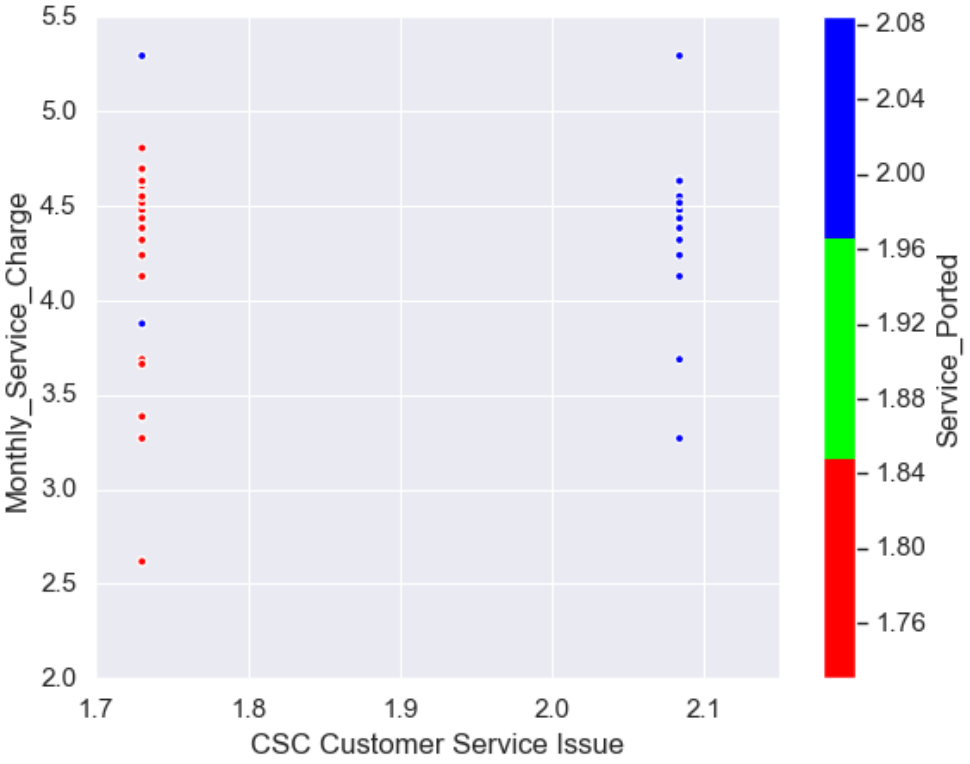
In [67]:
```
# create a scatter plot of Tenure versus Customer Service Calls and color by Service_Ported
sample.plot(kind='scatter', x = 'Tenure', y='Customer Service Calls', c='Service_Ported', col
ormap=cmap_bold)
```

Out[67]: <matplotlib.axes._subplots.AxesSubplot at 0x1403792ac50>



In [68]:
```
# create a scatter plot of CSC Customer Service Issue versus Monthly_Service_Charge and color by Service_Ported
sample.plot(kind='scatter', x = 'CSC Customer Service Issue', y='Monthly_Service_Charge', c= 'Service_Ported', colormap=cmap_bold)
```

Out[68]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x140379c2f98>
```



MACHINE-LEARNING

LOGISTIC REGRESSION

Logistic Regression is a Machine Learning classification algorithm, which used to predict the probability of a categorical dependent variable. In logistic regression, the dependent variable is Service_Ported a binary. This variable contains data coded as 1 (True = service proted (churned)) or 0 = False, service not ported (not churned). In other words, the logistic regression model predicts p(Y=1) as a function of X.

Assuptions:

Binary logistic regression requires the dependent variable to be binary. For a binary regression, the factor level 1 of the dependent variable should represent the desired outcome. Only the meaningful variables should be included. The independent variables should be independent of each other. That is, the model should have little or no multicollinearity. Logistic regression requires quite large sample sizes.

In [25]:
```
# from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression(C=1e9, class_weight='balanced')
feature_cols = ['Tenure']
X = sample2[feature_cols]
y = sample2.Service_Ported
logreg.fit(X, y)
Service_Ported_pred_class = logreg.predict(X)
```

In [26]:
```
logreg.coef_
```

Out[26]:
```
array([[-0.019277]])
```

In [27]:
```
#dir(logreg)
```

In [28]:
```
# print the class predictions
Service_Ported_pred_class
```

Out[28]:
```
array([0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1])
```

In [29]:
```
# add predicted class to DataFrame
sample2['Service_Ported_pred_class'] = Service_Ported_pred_class
```

In [30]:
```
# sort DataFrame by Tenure
sample2.sort_values('Tenure', inplace=True)
```

In [31]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

In [32]:
```
sample2.head()
```

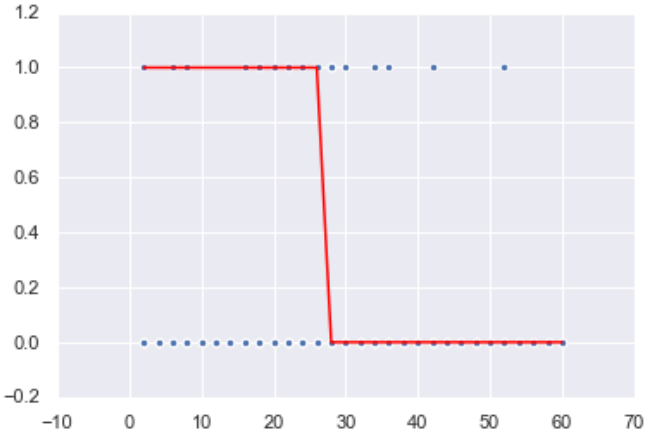
Out[32]:

Store ID	Account	Tenure	Account_Size	New_Account	Multi_Service_Account	Total Service Charge	Monthly_Service_Charge	S
19389	649649947	2	1	0	0	85	85	
17852	946975844	2	1	0	0	85	85	
17460	497485844	2	3	0	1	135	45	
15315	555556748	2	2	0	1	170	85	
18577	407976048	2	3	0	1	255	85	

5 rows × 22 columns

In [33]:
```
# plot the class predictions again
plt.scatter(sample2.Tenure, sample2.Service_Ported)
plt.plot(sample2.Tenure, sample2.Service_Ported_pred_class, color='red')
```

Out[33]:
```
[<matplotlib.lines.Line2D at 0x1a43084390>]
```



Predicted probabilities instead of just the class predictions, to understand how confident we are in a given prediction

In [34]:
```
logreg.predict_proba(X)
```

Out[34]:
```
array([[0.50712502, 0.49287498],
       [0.47822379, 0.52177621],
       [0.3839774 , 0.6160226 ],
       ...,
       [0.56459351, 0.43540649],
       [0.4590251 , 0.5409749 ],
       [0.48785089, 0.51214911]])
```

In [35]:
```
# store the predicted probabilites of class 1
Service_Ported_pred_prob = logreg.predict_proba(X)[:, 1]
```

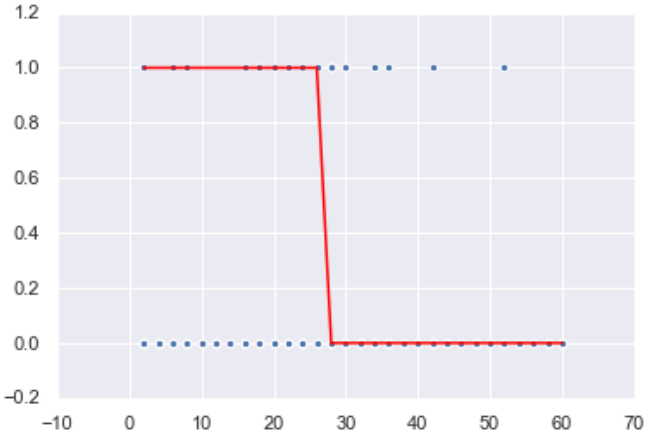
In [36]:
```
Service_Ported_pred_prob
```

Out[36]:
```
array([0.49287498, 0.52177621, 0.6160226 , 0.60686325, 0.50251298,
       0.5409749 , 0.52177621, 0.58832322, 0.49287498, 0.37058594,
       0.35278678, 0.51214911, 0.60686325, 0.51214911, 0.51214911,
       0.58832322, 0.48324228, 0.38874373, 0.52177621, 0.39794333,
       0.49287498, 0.47362201, 0.48324228, 0.49287498, 0.50251298,
       0.5409749 , 0.49287498, 0.50251298, 0.52177621, 0.44490661,
       0.49287498, 0.50251298, 0.34403457, 0.52177621, 0.41655444,
       0.45444717, 0.53138717, 0.49287498, 0.58832322, 0.5409749 ,
       0.5409749 , 0.39794333, 0.50251298, 0.53138717, 0.47362201,
       0.52177621, 0.5409749 , 0.49287498, 0.52177621, 0.59762813,
       0.47362201, 0.48324228, 0.5409749 , 0.34403457, 0.49287498,
       0.48324228, 0.50251298, 0.56952906, 0.51214911, 0.48324228,
       0.35278678, 0.55053239, 0.49287498, 0.53138717, 0.48324228,
       0.49287498, 0.48324228, 0.5409749 , 0.55053239, 0.48324228,
       0.58832322, 0.48324228, 0.52177621, 0.49287498, 0.52177621,
       0.53138717, 0.49287498, 0.42595356, 0.5409749 , 0.50251298,
       0.50251298, 0.5409749 , 0.34403457, 0.48324228, 0.50251298,
       0.49287498, 0.50251298, 0.49287498, 0.48324228, 0.52177621,
       0.41655444, 0.39794333, 0.55053239, 0.48324228, 0.51214911,
       0.6160226 , 0.37058594, 0.47362201, 0.48324228, 0.48324228,
       0.39794333, 0.53138717, 0.45444717, 0.58832322, 0.49287498,
       0.52177621, 0.50251298, 0.53138717, 0.48324228, 0.49287498,
       0.35278678, 0.50251298, 0.3796227 , 0.51214911, 0.60686325,
       0.51214911, 0.39794333, 0.42595356, 0.53138717, 0.49287498,
       0.48324228, 0.52177621, 0.52177621, 0.50251298, 0.35278678,
       0.50251298, 0.48324228, 0.42595356, 0.51214911, 0.49287498,
       0.6160226 , 0.5409749 , 0.48324228, 0.49287498, 0.56952906,
       0.49287498, 0.60686325, 0.52177621, 0.49287498, 0.48324228,
       0.5409749 , 0.6160226 , 0.59762813, 0.49287498, 0.52177621,
       0.53138717, 0.46402129, 0.50251298, 0.51214911, 0.58832322,
       0.52177621, 0.5409749 , 0.48324228, 0.48324228, 0.51214911,
       0.53138717, 0.44490661, 0.34403457, 0.58832322, 0.49287498,
       0.41655444, 0.49287498, 0.49287498, 0.49287498, 0.6160226 ,
       0.44490661, 0.4072156 , 0.48324228, 0.51214911, 0.45444717,
       0.46402129, 0.48324228, 0.60686325, 0.50251298, 0.5409749 ,
       0.46402129, 0.49287498, 0.51214911, 0.48324228, 0.49287498,
       0.49287498, 0.50251298, 0.57895474, 0.38874373, 0.53138717,
       0.52177621, 0.49287498, 0.42595356, 0.45444717, 0.49287498,
       0.5409749 , 0.5409749 , 0.49287498, 0.5409749 , 0.52177621,
       0.53138717, 0.49287498, 0.52177621, 0.39794333, 0.50251298,
       0.49287498, 0.48324228, 0.49287498, 0.5409749 , 0.48324228,
       0.6160226 , 0.53138717, 0.51214911, 0.52177621, 0.42595356,
       0.49287498, 0.49287498, 0.6160226 , 0.57895474, 0.44490661,
       0.52177621, 0.50251298, 0.38874373, 0.56005271, 0.49287498,
       0.49287498, 0.42595356, 0.50251298, 0.52177621, 0.50251298,
       0.36163889, 0.49287498, 0.50251298, 0.48324228, 0.52177621,
       0.50251298, 0.50251298, 0.46402129, 0.53138717, 0.53138717,
       0.44490661, 0.35278678, 0.51214911, 0.49287498, 0.49287498,
       0.50251298, 0.59762813, 0.48324228, 0.60686325, 0.53138717,
       0.46402129, 0.48324228, 0.50251298, 0.52177621, 0.37058594,
       0.59762813, 0.48324228, 0.51214911, 0.34403457, 0.49287498,
       0.47362201, 0.49287498, 0.51214911, 0.5409749 , 0.53138717,
       0.48324228, 0.51214911, 0.57895474, 0.4072156 , 0.45444717,
       0.55053239, 0.53138717, 0.43540649, 0.50251298, 0.47362201,
       0.49287498, 0.49287498, 0.51214911, 0.49287498, 0.49287498,
       0.35278678, 0.49287498, 0.50251298, 0.48324228, 0.51214911,
       0.6160226 , 0.49287498, 0.60686325, 0.5409749 , 0.49287498,
       0.53138717, 0.48324228, 0.36163889, 0.39794333, 0.56952906,
       0.57895474, 0.58832322, 0.53138717, 0.52177621, 0.36163889,
       0.43540649, 0.46402129, 0.53138717, 0.5409749 , 0.48324228,
       0.48324228, 0.34403457, 0.46402129, 0.53138717, 0.50251298,
       0.47362201, 0.49287498, 0.50251298, 0.34403457, 0.34403457,
       0.56005271, 0.41655444, 0.48324228, 0.53138717, 0.52177621,
       0.49287498, 0.49287498, 0.49287498, 0.46402129, 0.46402129,
       0.52177621, 0.51214911, 0.53138717, 0.39794333, 0.48324228,
       0.57895474, 0.48324228, 0.53138717, 0.3796227 , 0.42595356,
```

```
0.41655444, 0.48324228, 0.52177621, 0.53138717, 0.41655444,
0.53138717, 0.48324228, 0.4072156 , 0.49287498, 0.48324228,
0.55053239, 0.59762813, 0.48324228, 0.51214911, 0.46402129,
0.5409749 , 0.59762813, 0.42595356, 0.56005271, 0.53138717,
0.5409749 , 0.50251298, 0.48324228, 0.48324228, 0.49287498,
0.50251298, 0.51214911, 0.3796227 , 0.48324228, 0.52177621,
0.51214911, 0.39794333, 0.51214911, 0.47362201, 0.51214911,
0.51214911, 0.60686325, 0.46402129, 0.45444717, 0.49287498,
0.49287498, 0.48324228, 0.3796227 , 0.5409749 , 0.57895474,
0.48324228, 0.48324228, 0.53138717, 0.48324228, 0.55053239,
0.52177621, 0.48324228, 0.50251298, 0.48324228, 0.48324228,
0.58832322, 0.52177621, 0.51214911, 0.49287498, 0.56952906,
0.34403457, 0.51214911, 0.56952906, 0.48324228, 0.55053239,
0.56005271, 0.50251298, 0.50251298, 0.48324228, 0.49287498,
0.5409749 , 0.50251298, 0.49287498, 0.4072156 , 0.45444717,
0.39794333, 0.48324228, 0.48324228, 0.39794333, 0.49287498,
0.37058594, 0.51214911, 0.3796227 , 0.48324228, 0.42595356,
0.5409749 , 0.49287498, 0.49287498, 0.43540649, 0.48324228,
0.50251298, 0.50251298, 0.50251298, 0.46402129, 0.48324228,
0.49287498, 0.48324228, 0.51214911, 0.5409749 , 0.48324228,
0.48324228, 0.48324228, 0.49287498, 0.5409749 , 0.49287498,
0.49287498, 0.49287498, 0.3796227 , 0.48324228, 0.53138717,
0.60686325, 0.53138717, 0.45444717, 0.50251298, 0.49287498,
0.48324228, 0.59762813, 0.46402129, 0.48324228, 0.50251298,
0.53138717, 0.48324228, 0.53138717, 0.37058594, 0.48324228,
0.48324228, 0.49287498, 0.46402129, 0.48324228, 0.48324228,
0.49287498, 0.48324228, 0.49287498, 0.51214911, 0.39794333,
0.60686325, 0.5409749 , 0.48324228, 0.47362201, 0.50251298,
0.58832322, 0.52177621, 0.36163889, 0.53138717, 0.49287498,
0.52177621, 0.48324228, 0.49287498, 0.49287498, 0.42595356,
0.50251298, 0.52177621, 0.38874373, 0.34403457, 0.50251298,
0.41655444, 0.48324228, 0.35278678, 0.48324228, 0.45444717,
0.51214911, 0.44490661, 0.5409749 , 0.53138717, 0.46402129,
0.51214911, 0.51214911, 0.53138717, 0.6160226 , 0.5409749 ,
0.34403457, 0.50251298, 0.5409749 , 0.49287498, 0.51214911,
0.49287498, 0.53138717, 0.53138717, 0.49287498, 0.5409749 ,
0.34403457, 0.56952906, 0.48324228, 0.49287498, 0.49287498,
0.51214911, 0.60686325, 0.52177621, 0.49287498, 0.36163889,
0.50251298, 0.41655444, 0.49287498, 0.53138717, 0.60686325,
0.47362201, 0.51214911, 0.51214911, 0.48324228, 0.3796227 ,
0.49287498, 0.6160226 , 0.50251298, 0.34403457, 0.50251298,
0.6160226 , 0.57895474, 0.53138717, 0.45444717, 0.49287498,
0.43540649, 0.5409749 , 0.51214911])
```

In [37]:
```
# plot the predicted probabilities
plt.scatter(sample2.Tenure, sample2.Service_Ported)
plt.plot(sample2.Tenure, sample2.Service_Ported_pred_class, color='red')
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x10c4f7be0>]
```



In [38]:
```
# examine some example predictions
logreg.predict_proba(X)
```

Out[38]:
```
array([[0.50712502, 0.49287498],
       [0.47822379, 0.52177621],
       [0.3839774 , 0.6160226 ],
       ...,
       [0.56459351, 0.43540649],
       [0.4590251 , 0.5409749 ],
       [0.48785089, 0.51214911]])
```

In [39]:
```
##Saving the file
```

In [40]:
```
#sample1.to_csv('./Sample1.csv')
```

In [41]:
```
#sample1 = pd.read_csv("./sample2.csv", sep =',')
#sample2.head(2)
```

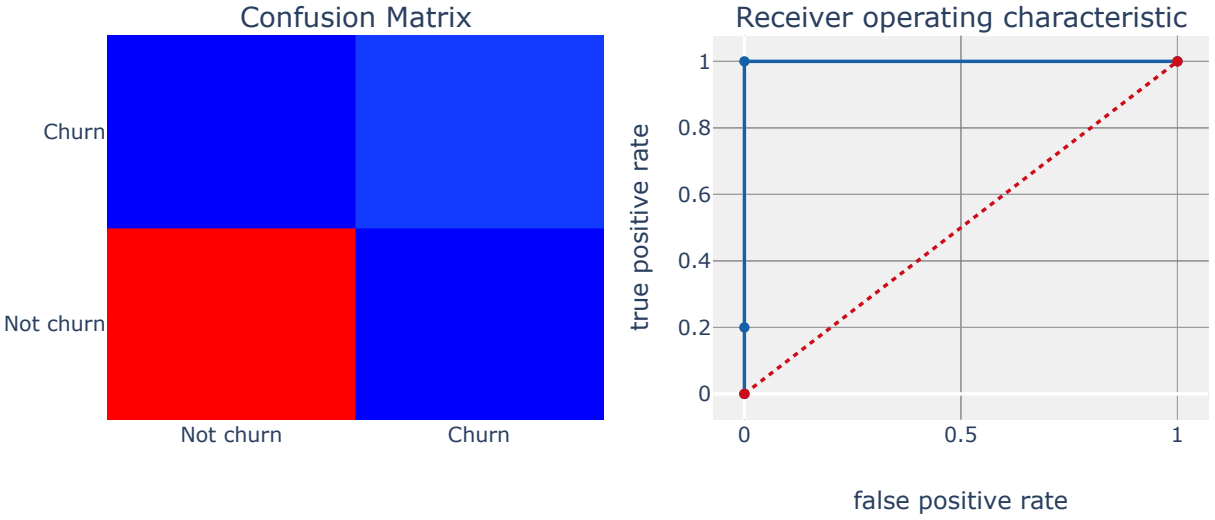
In [42]:
```
import plotly.tools as tls
target_col = ["Service_Ported"]
#sample2 = sample1.sample(frac=0.10, replace=True, random_state=1)
```

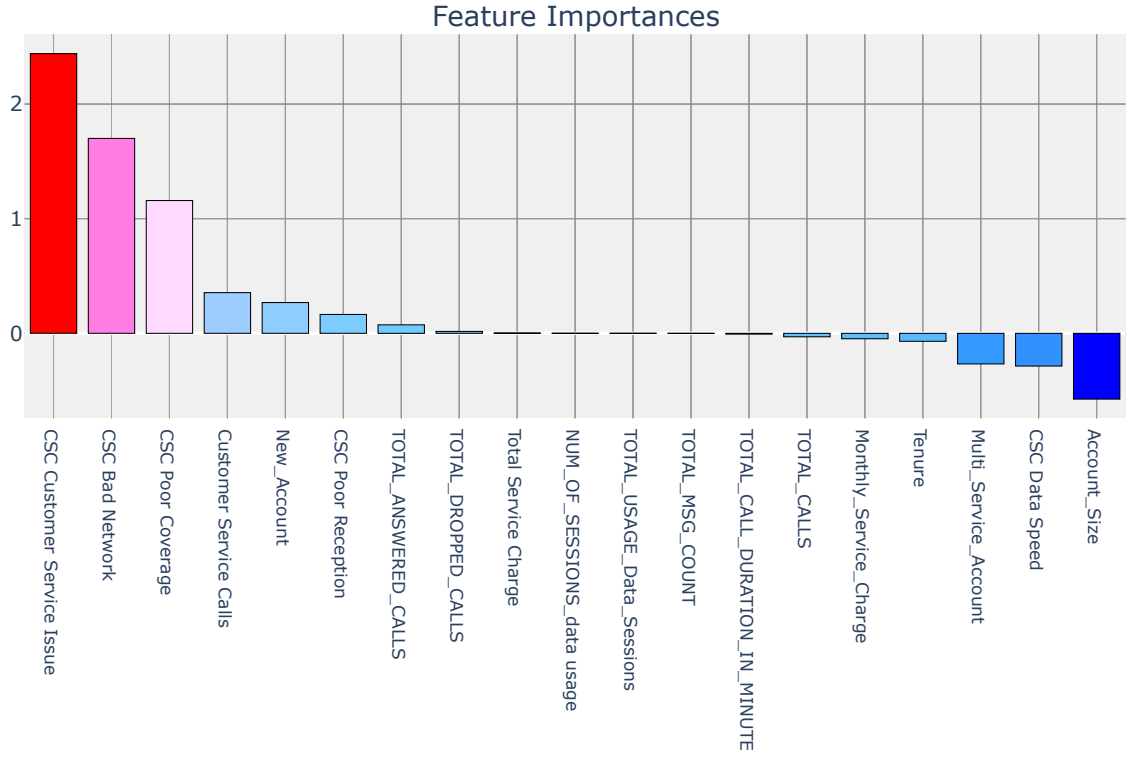
In [43]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from sklearn.metrics import roc_auc_score,roc_curve,scorer
from sklearn.metrics import f1_score
import statsmodels.api as sm
from sklearn.metrics import precision_score,recall_score
from yellowbrick.classifier import DiscriminationThreshold
#splitting train and test data
train,test = train_test_split(sample2,test_size = .25 ,random_state = 111)

##seperating dependent and independent variables
cols    = ['Tenure', 'Account_Size', 'New_Account',
       'Multi_Service_Account', 'Total Service Charge',
       'Monthly_Service_Charge','TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE', 'NUM_OF_SESSIONS_data usage',
       'TOTAL_USAGE_Data_Sessions ', 'Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage']
train_X = train[cols]
train_Y = train[target_col]
test_X  = test[cols]
test_Y  = test[target_col]


#threshold_plot - if True returns threshold plot for model

def mobile_churn_prediction(algorithm,train_X,test_X,
                            train_Y,test_Y,cols,cf,threshold_plot) :

    #model
    algorithm.fit(train_X,train_Y)
    predictions   = algorithm.predict(test_X)
    probabilities = algorithm.predict_proba(test_X)
    #coeffs
    if   cf == "coefficients" :
        coefficients  = pd.DataFrame(algorithm.coef_.ravel())
    elif cf == "features" :
        coefficients  = pd.DataFrame(algorithm.feature_importances_)

    column_df     = pd.DataFrame(cols)
    coef_sumry    = (pd.merge(coefficients,column_df,left_index= True,
                              right_index= True, how = "left"))
    coef_sumry.columns = ["coefficients","features"]
    coef_sumry    = coef_sumry.sort_values(by = "coefficients",ascending = False)

    print (algorithm)
    print ("\n Classification report : \n",classification_report(test_Y,predictions))
    print ("Accuracy   Score : ",accuracy_score(test_Y,predictions))
    #confusion matrix
    conf_matrix = confusion_matrix(test_Y,predictions)
    #roc_auc_score
    model_roc_auc = roc_auc_score(test_Y,predictions)
    print ("Area under curve : ",model_roc_auc,"\n")
    fpr,tpr,thresholds = roc_curve(test_Y,probabilities[:,1])

    #plot confusion matrix
    trace1 = go.Heatmap(z = conf_matrix ,
                        x = ["Not churn","Churn"],
                        y = ["Not churn","Churn"],
                        showscale  = False,colorscale = "Picnic",
                        name = "matrix")

    #plot roc curve
    trace2 = go.Scatter(x = fpr,y = tpr,
```

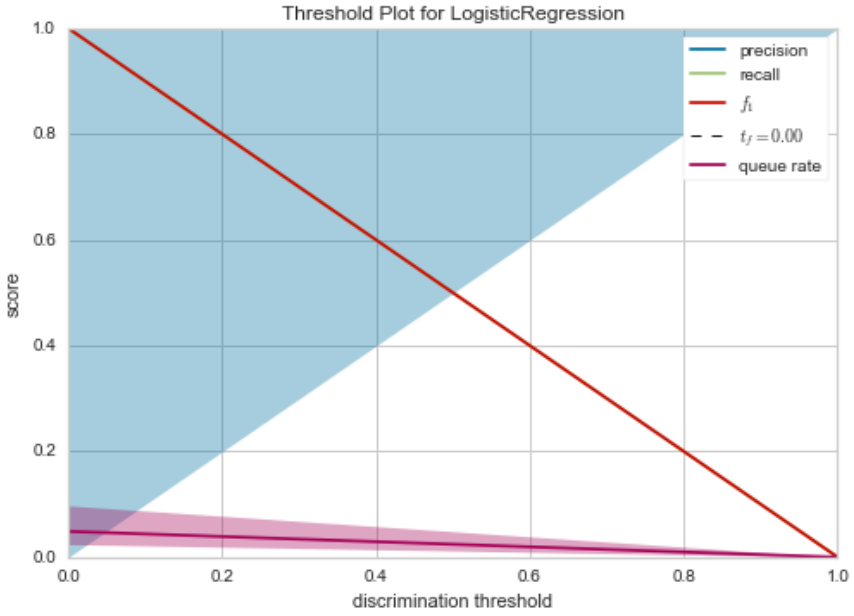
```
                        name = "Roc : " + str(model_roc_auc),
                        line = dict(color = ('rgb(22, 96, 167)'),width = 2))
    trace3 = go.Scatter(x = [0,1],y=[0,1],
                        line = dict(color = ('rgb(205, 12, 24)'),width = 2,
                        dash = 'dot'))

    #plot coeffs
    trace4 = go.Bar(x = coef_sumry["features"],y = coef_sumry["coefficients"],
                    name = "coefficients",
                    marker = dict(color = coef_sumry["coefficients"],
                                  colorscale = "Picnic",
                                  line = dict(width = .6,color = "black")))

    #subplots
    fig = tls.make_subplots(rows=2, cols=2, specs=[[{}, {}], [{'colspan': 2}, None]],
                            subplot_titles=('Confusion Matrix',
                                            'Receiver operating characteristic',
                                            'Feature Importances'))

    fig.append_trace(trace1,1,1)
    fig.append_trace(trace2,1,2)
    fig.append_trace(trace3,1,2)
    fig.append_trace(trace4,2,1)

    fig['layout'].update(showlegend=False, title="Model performance" ,
                         autosize = False,height = 900,width = 800,
                         plot_bgcolor = 'rgba(240,240,240, 0.95)',
                         paper_bgcolor = 'rgba(240,240,240, 0.95)',
                         margin = dict(b = 195))
    fig["layout"]["xaxis2"].update(dict(title = "false positive rate"))
    fig["layout"]["yaxis2"].update(dict(title = "true positive rate"))
    fig["layout"]["xaxis3"].update(dict(showgrid = True,tickfont = dict(size = 10),
                                        tickangle = 90))
    py.iplot(fig)

    if threshold_plot == True :
        visualizer = DiscriminationThreshold(algorithm)
        visualizer.fit(train_X,train_Y)
        visualizer.poof()

logit  = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)

mobile_churn_prediction(logit,train_X,test_X,train_Y.values.ravel(),test_Y.values.ravel(),
                        cols,"coefficients",threshold_plot = True)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

 Classification report :
               precision    recall  f1-score   support

           0       1.00      1.00      1.00       131
           1       1.00      1.00      1.00         5

    accuracy                           1.00       136
   macro avg       1.00      1.00      1.00       136
weighted avg       1.00      1.00      1.00       136

Accuracy   Score :  1.0
Area under curve :  1.0
```

Model performance



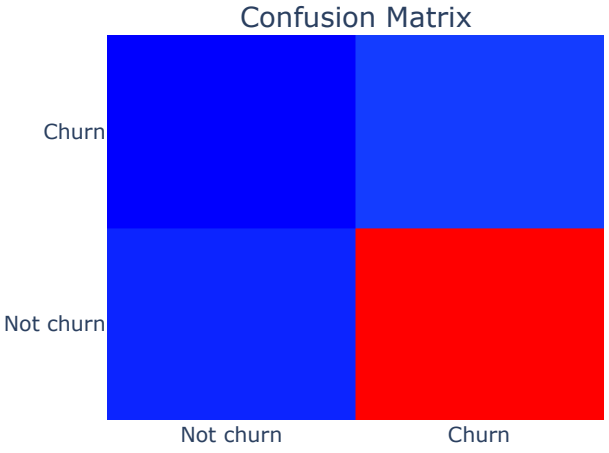


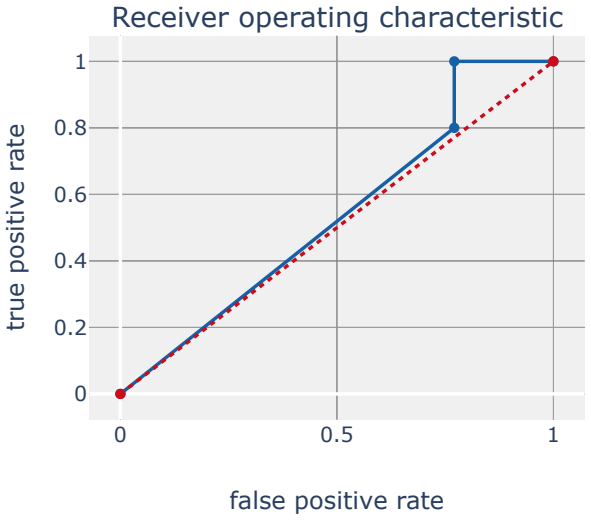

Threshold Plot for LogisticRegression
precision
recall
f_1
$t_f = 0.00$
queue rate
score
discrimination threshold
0.0
0.2
0.4
0.6
0.8
1.0

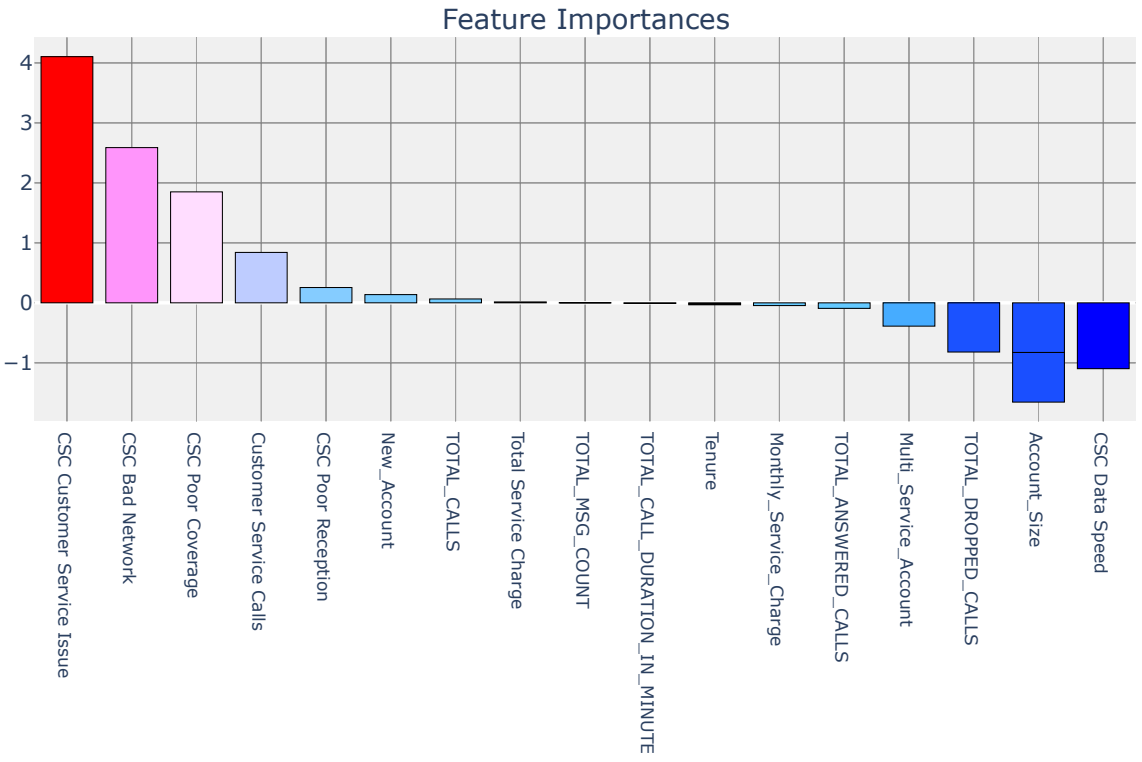
In [46]:
```
from imblearn.over_sampling import SMOTE

cols    =  [ 'Tenure', 'Account_Size', 'New_Account',
       'Multi_Service_Account', 'Total Service Charge',
       'Monthly_Service_Charge', 'TOTAL_MSG_COUNT',
       'TOTAL_CALLS', 'TOTAL_ANSWERED_CALLS', 'TOTAL_DROPPED_CALLS',
       'TOTAL_CALL_DURATION_IN_MINUTE','Customer Service Calls',
       'CSC Poor Reception', 'CSC Bad Network', 'CSC Customer Service Issue',
       'CSC Data Speed', 'CSC Poor Coverage', 'Account_Size', 'Tenure']
smote_X = sample2[cols]
smote_Y = sample2[target_col]

#Split train and test data
smote_train_X,smote_test_X,smote_train_Y,smote_test_Y = train_test_split(smote_X,smote_Y,
                                                                         test_size = .25 ,
                                                                         random_state = 111)

#oversampling minority class using smote
os = SMOTE(random_state = 0)
os_smote_X,os_smote_Y = os.fit_sample(smote_train_X,smote_train_Y)
os_smote_X = pd.DataFrame(data = os_smote_X,columns=cols)
os_smote_Y = pd.DataFrame(data = os_smote_Y,columns=target_col)
###



logit_smote = LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)

mobile_churn_prediction(logit_smote,os_smote_X,test_X,os_smote_Y,test_Y,
                        cols,"coefficients",threshold_plot = True)
```

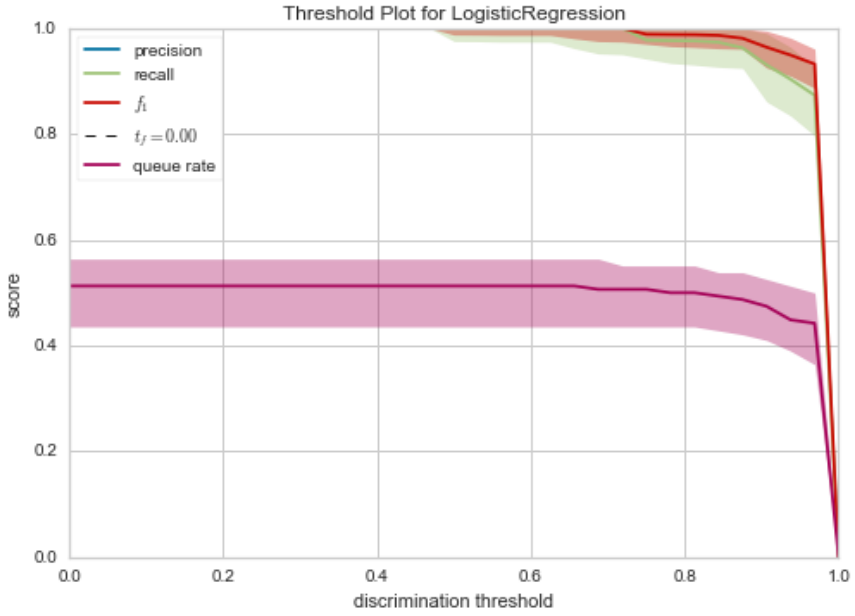
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
                   solver='liblinear', tol=0.0001, verbose=0, warm_start=False)

 Classification report :
               precision    recall  f1-score   support

           0       1.00      0.02      0.04       131
           1       0.04      1.00      0.07         5

    accuracy                           0.06       136
   macro avg       0.52      0.51      0.06       136
weighted avg       0.96      0.06      0.05       136

Accuracy   Score :  0.058823529411764705
Area under curve :  0.5114503816793894
```

Model performance









Decision Tree

Decision Trees are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision (Muller & Guido, page 70). The goal in building a decision tree is to ask as few if/else questions as possible. Each node in a tree either represents a question or a terminal node (also called a leaf) that contains the answer. The edges connect the answers to a question with the next question you would ask. In the machine learning setting, the sequence of if/else statements are called tests (which defer from the traditional test set used to see how generilizable our model is) (Muller & Guido, Page 71).

In [47]:
```
# Load libraries
import pandas as pd
from sklearn.tree import DecisionTreeClassifier # Import Decision Tree Classifier
from sklearn.model_selection import train_test_split # Import train_test_split function
from sklearn import metrics #Import scikit-learn metrics module for accuracy calc
```

In [48]:
```
# Selected features
feature_cols = ['Tenure', 'Customer Service Calls', 'CSC Poor Coverage', 'CSC Bad Network',
'Multi_Service_Account', 'CSC Customer Service Issue']
```

In [49]:
```
# define X and y
X = sample2[feature_cols] # Features
y = sample2.Service_Ported # Target variable
```

In [50]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

In [51]:
```
# Create Decision Tree classifer object
clf = DecisionTreeClassifier()

# Train Decision Tree Classifer
clf = clf.fit(X_train,y_train)
```

In [52]:
```
#Predict the response for test dataset
y_pred = clf.predict(X_test)
y_pred
```

Out[52]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0])
```

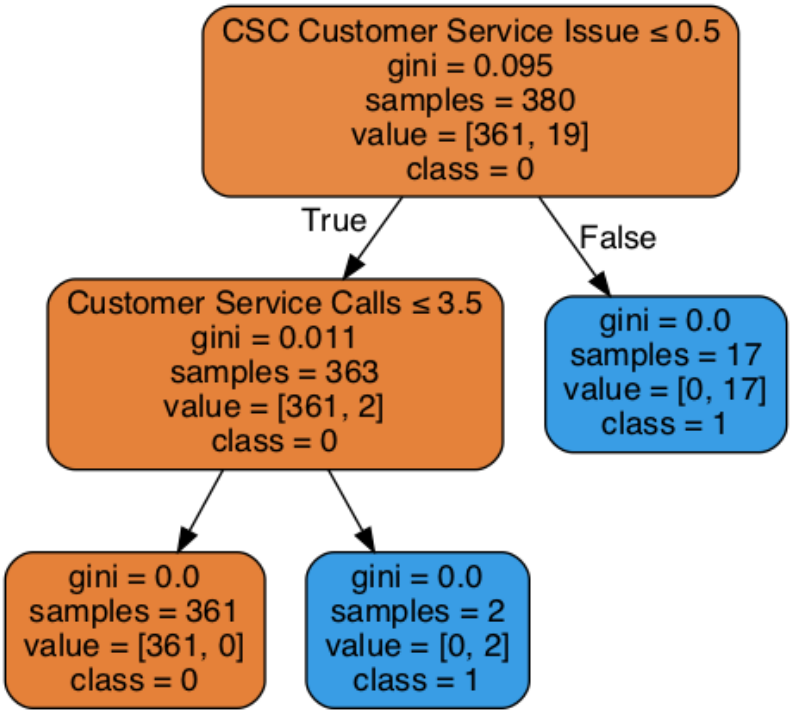
In [53]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 1.0
```

The results indicate an accuracy of 99.5%, which means that the algorithm successfully predicts customer churn 80% of the time.

In [54]:
```
# Decision Tree
dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('Churn.png')
Image(graph.create_png())
```

Out[54]:



- Notice the split in the bottom right: the **same class** is predicted in both of its leaves. That split didn't affect the **classification error rate**, though it did increase the **node purity**, which is important because it increases the accuracy of our predicted probabilities.

In [55]:
```
## Import the random forest model.
from sklearn.ensemble import RandomForestClassifier
## This line instantiates the model.
rf = RandomForestClassifier()
## Fit the model on your training data.
rf.fit(X_train, y_train)
## And score it on your testing data.
rf.score(X_test, y_test)
```

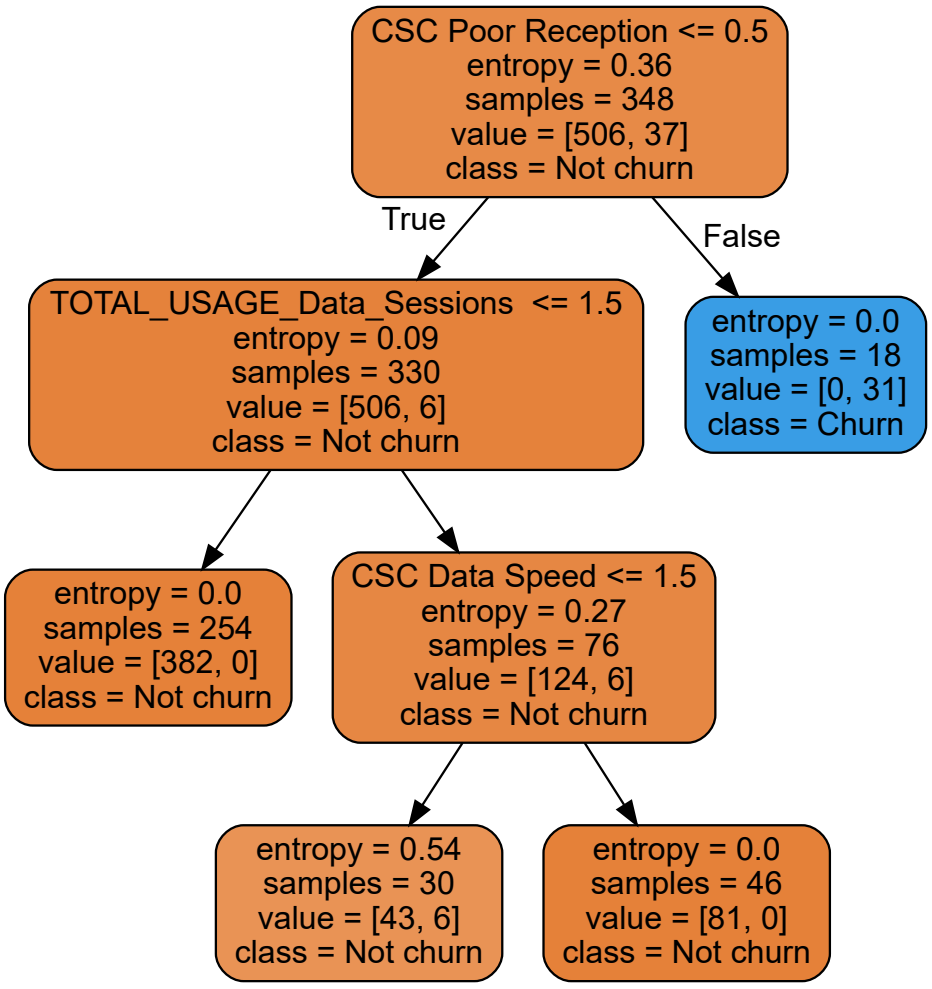
Out[55]: 1.0

In [56]:
```
# Idenfifying feature importance
feature_importances = pd.DataFrame(rf.feature_importances_,
                                   index = X_train.columns,
                                    columns=['Importance']).sort_values('Importance', ascendi
ng=False)
feature_importances
```

Out[56]:

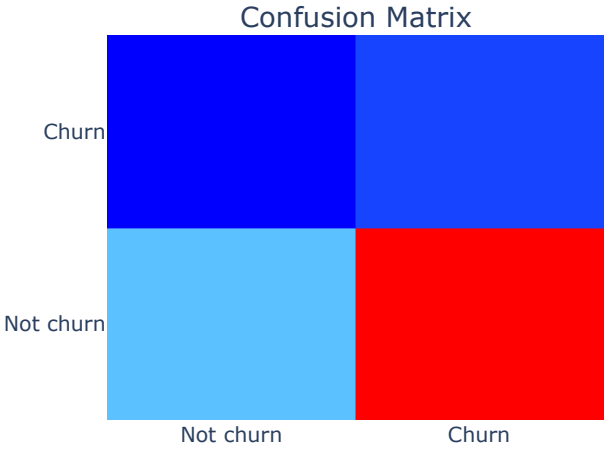
	Importance
CSC Customer Service Issue	0.385675
CSC Poor Coverage	0.273090
CSC Bad Network	0.184807
Customer Service Calls	0.155249
Tenure	0.001179
Multi_Service_Account	0.000000

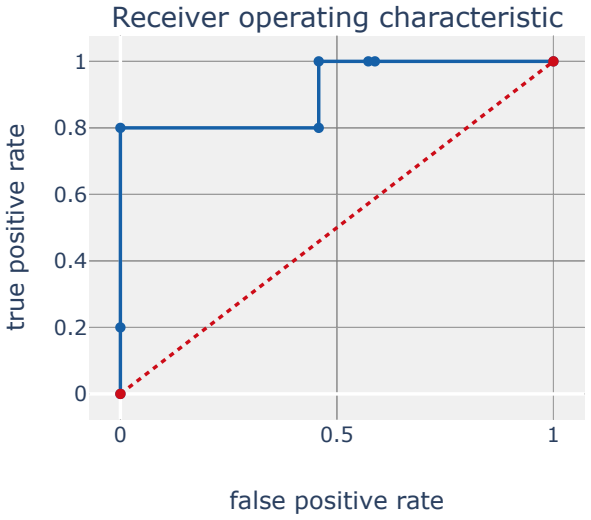
In [57]:
```
from sklearn.ensemble import RandomForestClassifier
from graphviz import Source

#function attributes
#columns  - column used
#nf_estimators   - The number of trees in the forest.
#estimated_tree  - tree number to be displayed
#maximum_depth   - depth of the tree
#criterion_type  - split criterion type ["gini" or "entropy"]
#Model performance - prints performance of model

def plot_tree_randomforest(columns,nf_estimators,
                           estimated_tree,maximum_depth,
                           criterion_type,model_performance = None) :



    #train and test datasets
    rf_x     = sample2[cols]
    rf_y     = sample2['Service_Ported']

    #random forest classifier
    rfc   = RandomForestClassifier(n_estimators = nf_estimators,
                                   max_depth = maximum_depth,
                                   criterion = criterion_type,
                                  )
    rfc.fit(rf_x,rf_y)

    estimated_tree = rfc.estimators_[estimated_tree]

    graph = Source(tree.export_graphviz(estimated_tree,out_file=None,
                                        rounded=True,proportion = False,
                            feature_names = columns,
                            precision  = 2,
                            class_names=["Not churn","Churn"],
                            filled = True))
    display(graph)

    #model performance
    if model_performance == True :
        mobile_churn_prediction(rfc,
                                rf_x,test_X[columns],
                                rf_y,test_Y,
                                columns,"features",threshold_plot = True)


cols1 = [ i for i in train_X.columns if i not in sample2['Service_Ported']]
plot_tree_randomforest(cols1,100,99,3,"entropy",True)
```

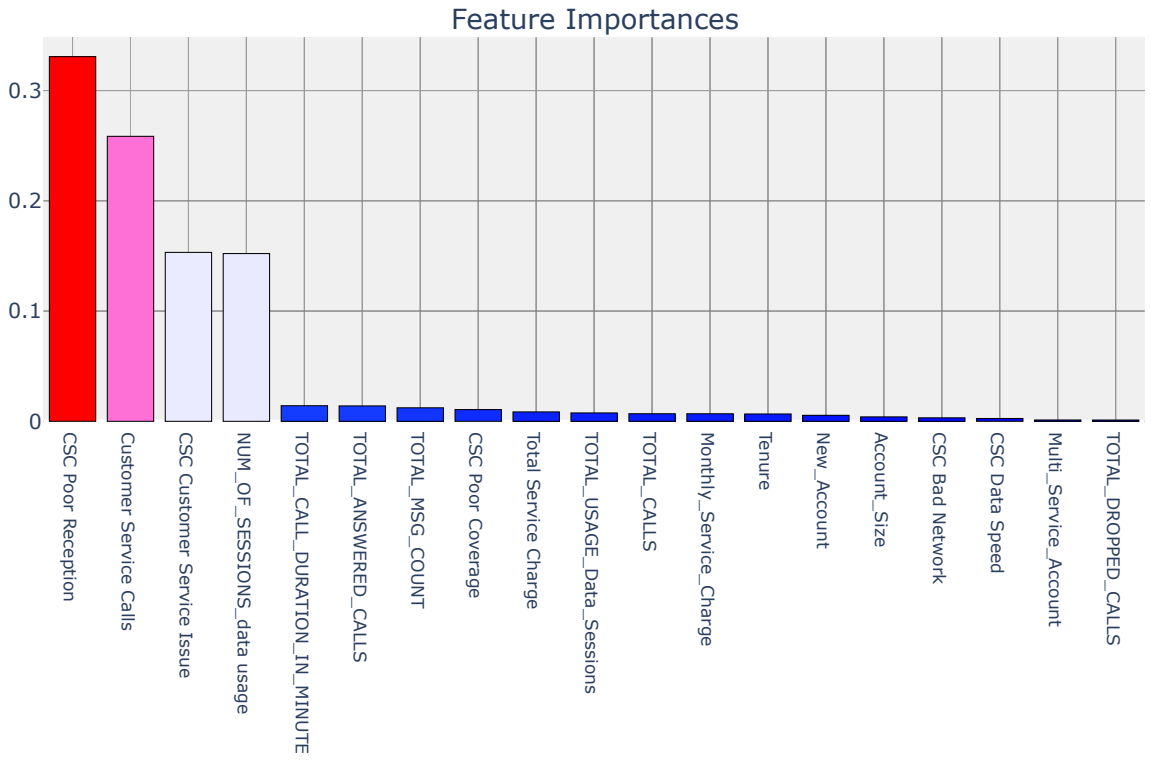


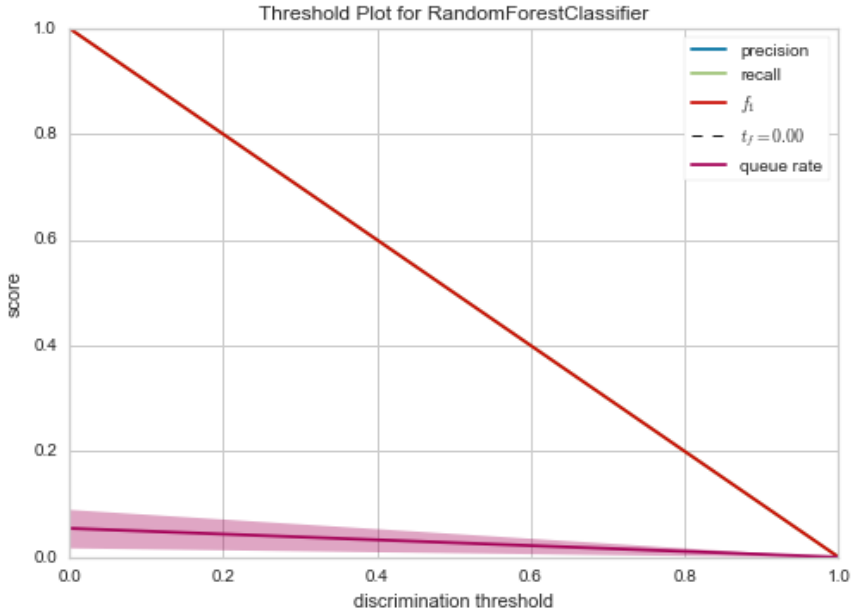

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
                       max_depth=3, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)

 Classification report :
               precision    recall  f1-score   support

           0       1.00      0.15      0.26       131
           1       0.04      1.00      0.08         5

    accuracy                           0.18       136
   macro avg       0.52      0.58      0.17       136
weighted avg       0.96      0.18      0.26       136

Accuracy   Score :  0.18382352941176472
Area under curve :  0.5763358778625954
```

Model performance









K-NEAREST NEIGHBORS AND SCIKIT-LEARN

The k-NN algorithm is considered one of the simpler machine learning algorithms to implement. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset-its "neighbors."

In [58]:
```
sample2.shape
```

Out[58]: (543, 22)

In [59]:
```
sample2.tail()
```

Out[59]:

Store ID	Account	Tenure	Account_Size	New_Account	Multi_Service_Account	Total Service Charge	Monthly_Service_Charge	S
16305	489447770	60	4	0	1	180	45	
14957	658464847	60	1	0	0	85	85	
18978	444054444	60	5	0	1	425	85	
14548	657644744	60	3	0	1	255	85	
20166	999449450	60	1	0	0	85	85	

5 rows × 22 columns

TERMINOLOGY

- **266765 observations**
- **21 features** (p=21)
- **Response**: Service Ported

In [60]:
```
# This is a model, which is Scirkit-learn
from sklearn.neighbors import KNeighborsClassifier
```

In [61]:
```
# store feature matrix in "X"
example_features1 = ['Tenure', 'Customer Service Calls', 'CSC Customer Service Issue', 'CSC P
oor Reception', 'TOTAL_DROPPED_CALLS', 'New_Account']
X = sample2[example_features1]
```

In [62]:
```
# Assigning X and Y variables
X = sample2[example_features1]
y = sample2.Service_Ported
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X, y)
```

Out[62]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=1, p=2,
                     weights='uniform')
```

In [63]:
```
# First five rows of X columns
X.head()
```

Out[63]:

Store ID	Tenure	Customer Service Calls	CSC Customer Service Issue	CSC Poor Reception	TOTAL_DROPPED_CALLS	New_Account
19389	2	5	4	3	0	0
17852	2	2	0	3	0	0
17460	2	3	0	0	1	0
15315	2	1	0	1	0	0
18577	2	1	0	0	0	0

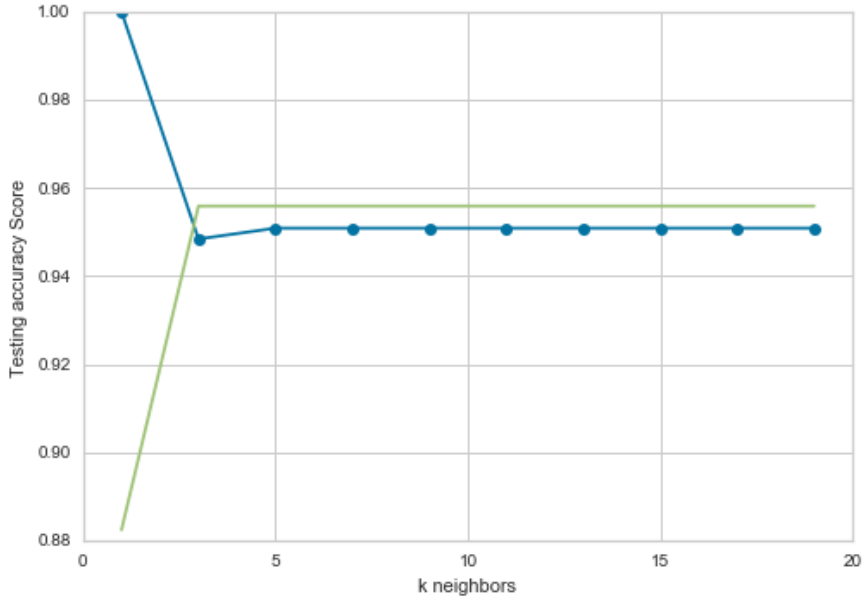
In [ ]:
```

```

In [64]:
```
target = sample2['Service_Ported']
```

In [65]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(sample2, target, random_state=42)
```

In [66]:
```
# Loop through different k values to see which has the highest accuracy
# Note: We only use odd numbers because we don't want any ties
train_scores = []
test_scores = []
for k in range(1, 20, 2):
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    train_score = knn.score(X_train, y_train)
    test_score = knn.score(X_test, y_test)
    train_scores.append(train_score)
    test_scores.append(test_score)
    print(f"k: {k}, Train/Test Score: {train_score:.3f}/{test_score:.3f}")


plt.plot(range(1, 20, 2), train_scores, marker='o')
plt.plot(range(1, 20, 2), test_scores, marker="x")
plt.xlabel("k neighbors")
plt.ylabel("Testing accuracy Score")
plt.show()
```

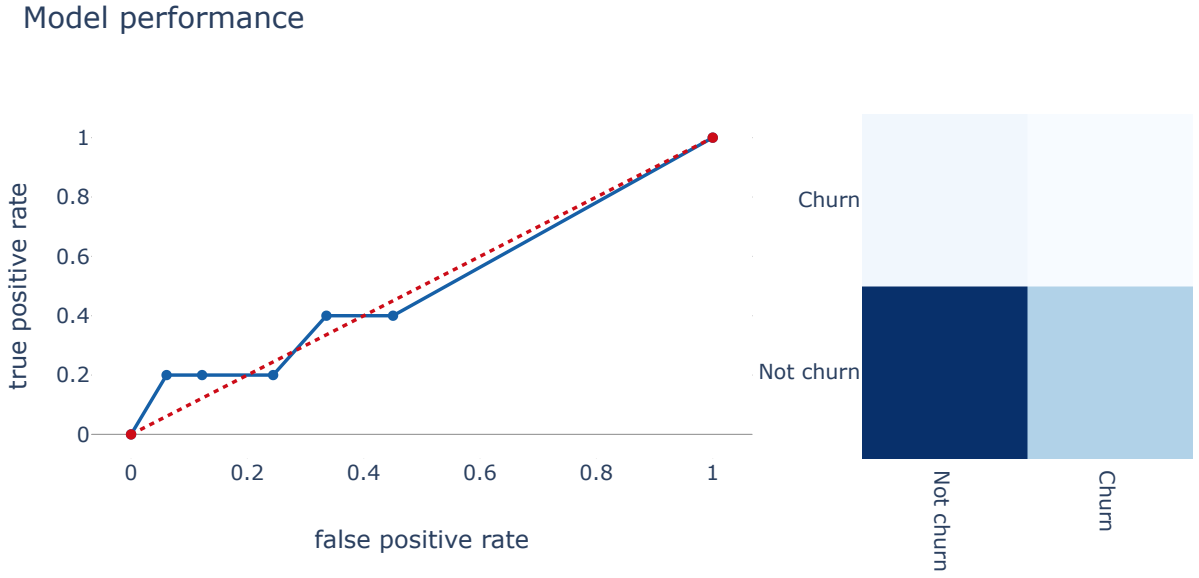
```
k: 1, Train/Test Score: 1.000/0.882
k: 3, Train/Test Score: 0.948/0.956
k: 5, Train/Test Score: 0.951/0.956
k: 7, Train/Test Score: 0.951/0.956
k: 9, Train/Test Score: 0.951/0.956
k: 11, Train/Test Score: 0.951/0.956
k: 13, Train/Test Score: 0.951/0.956
k: 15, Train/Test Score: 0.951/0.956
k: 17, Train/Test Score: 0.951/0.956
k: 19, Train/Test Score: 0.951/0.956
```

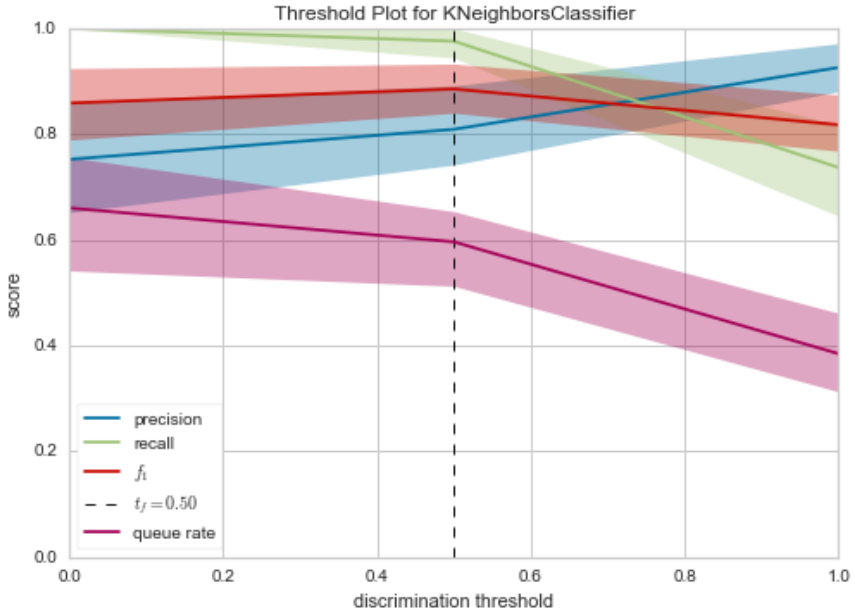


In [67]:
```
def mobile_churn_prediction_alg(algorithm,X_train,X_test,
                                y_train,y_test,threshold_plot = True) :

    #model
    algorithm.fit(X_train,y_train)
    predictions   = algorithm.predict(X_test)
    probabilities = algorithm.predict_proba(X_test)

    print (algorithm)
    print ("\n Classification report : \n",classification_report(y_test,predictions))
    print ("Accuracy Score   : ",accuracy_score(y_test,predictions))
    #confusion matrix
    conf_matrix = confusion_matrix(y_test,predictions)
    #roc_auc_score
    model_roc_auc = roc_auc_score(y_test,predictions)
    print ("Area under curve : ",model_roc_auc)
    fpr,tpr,thresholds = roc_curve(y_test,probabilities[:,1])

    #plot roc curve
    trace1 = go.Scatter(x = fpr,y = tpr,
                        name = "Roc : " + str(model_roc_auc),
                        line = dict(color = ('rgb(22, 96, 167)'),width = 2),
                       )
    trace2 = go.Scatter(x = [0,1],y=[0,1],
                        line = dict(color = ('rgb(205, 12, 24)'),width = 2,
                        dash = 'dot'))

    #plot confusion matrix
    trace3 = go.Heatmap(z = conf_matrix ,x = ["Not churn","Churn"],
                        y = ["Not churn","Churn"],
                        showscale  = False,colorscale = "Blues",name = "matrix",
                        xaxis = "x2",yaxis = "y2"
                       )

    layout = go.Layout(dict(title="Model performance" ,
                            autosize = False,height = 500,width = 800,
                            showlegend = False,
                            plot_bgcolor  = "rgb(243,243,243)",
                            paper_bgcolor = "rgb(243,243,243)",
                            xaxis = dict(title = "false positive rate",
                                         gridcolor = 'rgb(255, 255, 255)',
                                         domain=[0, 0.6],
                                         ticklen=5,gridwidth=2),
                            yaxis = dict(title = "true positive rate",
                                         gridcolor = 'rgb(255, 255, 255)',
                                         zerolinewidth=1,
                                         ticklen=5,gridwidth=2),
                            margin = dict(b=200),
                            xaxis2=dict(domain=[0.7, 1],tickangle = 90,
                                        gridcolor = 'rgb(255, 255, 255)'),
                            yaxis2=dict(anchor='x2',gridcolor = 'rgb(255, 255, 255)')
                           )
                      )
    data = [trace1,trace2,trace3]
    fig = go.Figure(data=data,layout=layout)

    py.iplot(fig)

    if threshold_plot == True :
        visualizer = DiscriminationThreshold(algorithm)
        visualizer.fit(X_train,y_train.values.ravel())
        visualizer.poof()
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
mobile_churn_prediction_alg(knn,os_smote_X,test_X,
                            os_smote_Y,test_Y,threshold_plot = True)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=1, n_neighbors=5, p=2,
                     weights='uniform')

 Classification report :
              precision    recall  f1-score   support

          0       0.96      0.76      0.85       131
          1       0.03      0.20      0.05         5

avg / total       0.93      0.74      0.82       136

Accuracy Score   :  0.7352941176470589
Area under curve :  0.4778625954198473
```





NEURAL NETWORK

In [68]:
```
#Normalized data from above

target = sample2['Service_Ported']
```

In [69]:
```
# Split dataset into training set and test set
X_train, X_test, y_train, y_test = train_test_split(sample2, target, test_size=0.3, random_st
ate=1)
```

In [70]:
```
# Importing Standard Scaler for processing
from sklearn.preprocessing import StandardScaler
X_scaler = StandardScaler().fit(X_train)
```

In [71]:
```
# Spletting the data to train and test
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

In [72]:
```
#from tensorflow.keras.utils import to_categorical
```

In [73]:
```
from tensorflow.keras.models import Sequential
model = Sequential()
```

In [74]:
```
from tensorflow.keras.layers import Dense
number_inputs = 3
number_hidden_nodes = 4
model.add(Dense(units=number_hidden_nodes,
                activation='relu', input_dim=number_inputs))
```

```
WARNING:tensorflow:From /Users/vanjoisscott/anaconda3/lib/python3.7/site-packages/tensorflow/
python/ops/resource_variable_ops.py:435: colocate_with (from tensorflow.python.framework.ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [75]:
```
number_classes = 2
model.add(Dense(units=number_classes, activation='softmax'))
```

In [76]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 4)                 16
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 10
=================================================================
Total params: 26
Trainable params: 26
Non-trainable params: 0
_________________________________________________________________
```

SVM

In [77]:
```
# Idenfifying target variable
target = sample2["Service_Ported"]
target_names = ["negative", "positive"]
```

In [78]:
```
# Idenfifying target variable
data = sample2.drop("Service_Ported", axis=1)
feature_names = data.columns
```

In [79]:
```
# Spletting the data to train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data, target, random_state=42)
```

In [80]:
```
# Support vector machine linear classifier
from sklearn.svm import SVC
# model = SVC(kernel='linear')
# model.fit(X_train, y_train)
```

In [ ]:
```
svc_lin  = SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
               decision_function_shape='ovr', degree=3, gamma=1.0, kernel='linear',
               max_iter=-1, probability=True, random_state=None, shrinking=True,
               tol=0.001, verbose=False)
mobile_churn_prediction(svc_lin,X_train,X_test,y_train,y_test,
                        cols,"coefficients",threshold_plot = False)
```

In [ ]:
```
# Model Accuracy
print('Test Acc: %.3f' % svc_lin.score(X_test, y_test))
```

In [ ]:
```
# Calculate classification report
from sklearn.metrics import classification_report
predictions = model.predict(X_test)
print(classification_report(y_test, predictions,
                            target_names=target_names))
```

In [ ]:
```

```

In [ ]:
```

```

In [73]:
```
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
import plotly.figure_factory as ff

#gives model report in dataframe
def model_report(model,training_x,testing_x,training_y,testing_y,name) :
    model.fit(training_x,training_y)
    predictions  = model.predict(testing_x)
    accuracy     = accuracy_score(testing_y,predictions)
    recallscore  = recall_score(testing_y,predictions)
    precision    = precision_score(testing_y,predictions)
    roc_auc      = roc_auc_score(testing_y,predictions)
    f1score      = f1_score(testing_y,predictions)
    kappa_metric = cohen_kappa_score(testing_y,predictions)

    df = pd.DataFrame({"Model"           : [name],
                       "Accuracy_score"  : [accuracy],
                       "Recall_score"    : [recallscore],
                       "Precision"       : [precision],
                       "f1_score"        : [f1score],
                       "Area_under_curve": [roc_auc],
                       "Kappa_metric"    : [kappa_metric],
                      })
    return df

#outputs for every model
model1 = model_report(logit,train_X,test_X,train_Y,test_Y,
                      "Logistic Regression(Baseline_model)")
model2 = model_report(logit_smote,os_smote_X,test_X,os_smote_Y,test_Y,
                      "Logistic Regression(SMOTE)")

decision_tree = DecisionTreeClassifier(max_depth = 9,
                                       random_state = 123,
                                       splitter  = "best",
                                       criterion = "gini",
                                      )
model3 = model_report(decision_tree,train_X,test_X,train_Y,test_Y,
                      "Decision Tree")
model4 = model_report(knn,os_smote_X,test_X,os_smote_Y,test_Y,
                      "KNN Classifier")
rfc = RandomForestClassifier(n_estimators = 1000,
                             random_state = 123,
                             max_depth = 9,
                             criterion = "gini")
model5 = model_report(rfc,train_X,test_X,train_Y,test_Y,
                      "Random Forest Classifier")

#model6 = model_report(svc_lin,os_smote_X,test_X,os_smote_Y,test_Y,
#                      "SVM Classifier Linear")


#concat all models
model_performances = pd.concat([model1,model2,model3,
                                model4,model5,model6,

                                ],axis = 0).reset_index()

model_performances = model_performances.drop(columns = "index",axis =1)

table  = ff.create_table(np.round(model_performances,4))

py.iplot(table)
```

Model	Accuracy_score	Recall_score	Precision	f1_score	A
Logistic Regression(Baseline model)	0.9985	0.971	1.0	0.9853	0
Logistic Regression(SMOTE)	0.9926	1.0	0.0515	0.0979	0
Decision Tree	0.9993	0.9884	0.9985	0.9934	0
KNN Classifier	0.3283	0.7246	0.0531	0.0989	0

References

Geeta, D. V., & Shashi, M. (2012). Applicability of data mining techniques for churn prediction. *IUP Journal of Information Technology,* 8(1), 22-35. Retrieved from https://searchproquestcom.contentproxy.phoenix.edu/docview/1434101064?accountid=35812

Shubham. Ranjan (2019). Geeks For Geeks *Python | Pandas dataframe.ffill(),* Retrieved from https://www.geeksforgeeks.org/python-pandas-dataframe-ffill/

Author Unknown (2013). Python For Beginners *Python's OS Module,* Retrieved from https://www.pythonforbeginners.com/os/pythons-os-module

Andreas C. Muller & Sarah Guido (2013). Introduction to Machine Learning with Python: *A guide for Data Scientists.,* Retrieved from https://www.pythonforbeginners.com/os/pythons-os-module

Michael Waskom (2012-2018). Seaborn *An Introduction to Seaborn* Retrieved from https://seaborn.pydata.org/introduction.html#introduction

Author Unknown . scikit-learn *5.9. Transforming the prediction target (y),* Retrieved from https://scikit-learn.org/stable/modules/preprocessing_targets.html#preprocessing-targets

Author Unknown . ipython.readthedocs.io *Rich Outputs,* Retrieved from https://ipython.readthedocs.io/en/stable/interactive/plotting.html

Jason Ridgen (2017) Medium *A guide to itertools,* Retrieved from https://medium.com/@jasonrigden/a-guide-to-python-itertools-82e5a306cdf8

Author Unknown (2019) docs.python.org *A guide to itertools,* Retrieved from https://docs.python.org/2/library/io.html

Guest Contributor (2019) docs.python.org *Relative vs Absolute Imports in Python,* Retrieved from https://stackabuse.com/relative-vs-absolute-imports-in-python/