Logistic Regression The Telecom Churn Case Study

August 12, 2020

0.1 Telecom Churn Case Study

With 21 predictor variables we need to predict whether a particular customer will switch to another telecom provider or not. In telecom terminology, this is referred to as churning and not churning, respectively.

0.1.1 Step 1: Importing and Merging Data

```
In [1]: # Suppressing Warnings
        import warnings
        warnings.filterwarnings('ignore')
```

```
In [2]: # Importing Pandas and NumPy
        import pandas as pd, numpy as np
```

```
In [3]: # Importing all datasets
        churn_data = pd.read_csv("churn_data.csv")
        churn_data.head()
```

```
Out[3]:    customerID  tenure PhoneService        Contract PaperlessBilling  \
        0  7590-VHVEG       1           No  Month-to-month              Yes
        1  5575-GNVDE      34          Yes        One year               No
        2  3668-QPYBK       2          Yes  Month-to-month              Yes
        3  7795-CFOCW      45           No        One year               No
        4  9237-HQITU       2          Yes  Month-to-month              Yes

                       PaymentMethod  MonthlyCharges TotalCharges Churn
        0           Electronic check           29.85        29.85    No
        1               Mailed check           56.95       1889.5    No
        2               Mailed check           53.85       108.15   Yes
        3  Bank transfer (automatic)           42.30      1840.75    No
        4           Electronic check           70.70       151.65   Yes
```

```
In [4]: customer_data = pd.read_csv("customer_data.csv")
        customer_data.head()
```

```
Out[4]:    customerID  gender  SeniorCitizen Partner Dependents
        0  7590-VHVEG  Female              0     Yes         No
```

```
1  5575-GNVDE    Male              0       No          No
2  3668-QPYBK    Male              0       No          No
3  7795-CFOCW    Male              0       No          No
4  9237-HQITU  Female              0       No          No
```

```
In [5]: internet_data = pd.read_csv("internet_data.csv")
        internet_data.head()
```

```
Out[5]:    customerID     MultipleLines InternetService OnlineSecurity OnlineBackup  \
        0  7590-VHVEG  No phone service             DSL             No          Yes
        1  5575-GNVDE                No             DSL            Yes           No
        2  3668-QPYBK                No             DSL            Yes          Yes
        3  7795-CFOCW  No phone service             DSL            Yes           No
        4  9237-HQITU                No     Fiber optic             No           No

          DeviceProtection TechSupport StreamingTV StreamingMovies
        0               No          No          No              No
        1              Yes          No          No              No
        2               No          No          No              No
        3              Yes         Yes          No              No
        4               No          No          No              No
```

Combining all data files into one consolidated dataframe

```
In [6]: # Merging on 'customerID'
        df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')
```

```
In [7]: # Final dataframe with all predictor variables
        telecom = pd.merge(df_1, internet_data, how='inner', on='customerID')
```

0.1.2 Step 2: Inspecting the Dataframe

```
In [8]: # Let's see the head of our master dataset
        telecom.head()
```

```
Out[8]:    customerID  tenure PhoneService        Contract PaperlessBilling  \
        0  7590-VHVEG       1           No  Month-to-month              Yes
        1  5575-GNVDE      34          Yes        One year               No
        2  3668-QPYBK       2          Yes  Month-to-month              Yes
        3  7795-CFOCW      45           No        One year               No
        4  9237-HQITU       2          Yes  Month-to-month              Yes

                       PaymentMethod  MonthlyCharges TotalCharges Churn  gender  \
        0           Electronic check           29.85        29.85    No  Female
        1               Mailed check           56.95       1889.5    No    Male
        2               Mailed check           53.85       108.15   Yes    Male
        3  Bank transfer (automatic)           42.30      1840.75    No    Male
        4           Electronic check           70.70       151.65   Yes  Female
```

```
        ...        Partner Dependents     MultipleLines InternetService  \
0       ...            Yes         No  No phone service             DSL
1       ...             No         No                No             DSL
2       ...             No         No                No             DSL
3       ...             No         No  No phone service             DSL
4       ...             No         No                No     Fiber optic

  OnlineSecurity OnlineBackup DeviceProtection TechSupport StreamingTV  \
0             No          Yes               No          No          No
1            Yes           No              Yes          No          No
2            Yes          Yes               No          No          No
3            Yes           No              Yes         Yes          No
4             No           No               No          No          No

  StreamingMovies
0              No
1              No
2              No
3              No
4              No

[5 rows x 21 columns]
```

In [9]:
```python
# Let's check the dimensions of the dataframe
telecom.shape
```

Out[9]: (7043, 21)

In [10]:
```python
# let's look at the statistical aspects of the dataframe
telecom.describe()
```

Out[10]:

	tenure	MonthlyCharges	SeniorCitizen
count	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	0.162147
std	24.559481	30.090047	0.368612
min	0.000000	18.250000	0.000000
25%	9.000000	35.500000	0.000000
50%	29.000000	70.350000	0.000000
75%	55.000000	89.850000	0.000000
max	72.000000	118.750000	1.000000

In [11]:
```python
# Let's see the type of each column
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 21 columns):
customerID          7043 non-null object
tenure              7043 non-null int64
```

```
PhoneService        7043 non-null object
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7043 non-null object
Churn               7043 non-null object
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
MultipleLines       7043 non-null object
InternetService     7043 non-null object
OnlineSecurity      7043 non-null object
OnlineBackup        7043 non-null object
DeviceProtection    7043 non-null object
TechSupport         7043 non-null object
StreamingTV         7043 non-null object
StreamingMovies     7043 non-null object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.2+ MB
```

0.1.3 Step 3: Data Preparation

Converting some binary variables (Yes/No) to 0/1

In [12]:
```
# List of variables to map

varlist =  ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']

# Defining the map function
def binary_map(x):
    return x.map({'Yes': 1, "No": 0})

# Applying the function to the housing list
telecom[varlist] = telecom[varlist].apply(binary_map)
```

In [13]:
```
telecom.head()
```

Out[13]:
```
   customerID  tenure  PhoneService        Contract  PaperlessBilling  \
0  7590-VHVEG       1             0  Month-to-month                 1
1  5575-GNVDE      34             1        One year                 0
2  3668-QPYBK       2             1  Month-to-month                 1
3  7795-CFOCW      45             0        One year                 0
4  9237-HQITU       2             1  Month-to-month                 1

               PaymentMethod  MonthlyCharges TotalCharges  Churn  gender  \
0           Electronic check           29.85        29.85      0  Female
```

```
1              Mailed check           56.95       1889.5      0    Male
2              Mailed check           53.85       108.15      1    Male
3  Bank transfer (automatic)          42.30      1840.75      0    Male
4          Electronic check           70.70       151.65      1  Female

        ...        Partner  Dependents     MultipleLines InternetService  \
0       ...              1           0  No phone service             DSL
1       ...              0           0                No             DSL
2       ...              0           0                No             DSL
3       ...              0           0  No phone service             DSL
4       ...              0           0                No     Fiber optic

  OnlineSecurity OnlineBackup DeviceProtection TechSupport StreamingTV  \
0             No          Yes               No          No          No
1            Yes           No              Yes          No          No
2            Yes          Yes               No          No          No
3            Yes           No              Yes         Yes          No
4             No           No               No          No          No

  StreamingMovies
0              No
1              No
2              No
3              No
4              No

[5 rows x 21 columns]
```

For categorical variables with multiple levels, create dummy features (one-hot encoded)

In [14]:
```python
# Creating a dummy variable for some of the categorical variables and dropping the fi
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetServi

# Adding the results to the master dataframe
telecom = pd.concat([telecom, dummy1], axis=1)
```

In [15]:
```python
telecom.head()
```

Out[15]:
```
   customerID  tenure  PhoneService        Contract  PaperlessBilling  \
0  7590-VHVEG       1             0  Month-to-month                 1
1  5575-GNVDE      34             1        One year                 0
2  3668-QPYBK       2             1  Month-to-month                 1
3  7795-CFOCW      45             0        One year                 0
4  9237-HQITU       2             1  Month-to-month                 1

               PaymentMethod  MonthlyCharges TotalCharges  Churn  gender  \
0           Electronic check           29.85        29.85      0  Female
1               Mailed check           56.95       1889.5      0    Male
```

```
2              Mailed check          53.85       108.15      1    Male
3  Bank transfer (automatic)         42.30      1840.75      0    Male
4           Electronic check         70.70       151.65      1  Female

        ...          StreamingTV  StreamingMovies  Contract_One year  \
0       ...                   No               No                  0
1       ...                   No               No                  1
2       ...                   No               No                  0
3       ...                   No               No                  1
4       ...                   No               No                  0

  Contract_Two year PaymentMethod_Credit card (automatic)  \
0                 0                                     0
1                 0                                     0
2                 0                                     0
3                 0                                     0
4                 0                                     0

  PaymentMethod_Electronic check PaymentMethod_Mailed check gender_Male  \
0                              1                          0           0
1                              0                          1           1
2                              0                          1           1
3                              0                          0           1
4                              1                          0           0

  InternetService_Fiber optic InternetService_No
0                           0                  0
1                           0                  0
2                           0                  0
3                           0                  0
4                           1                  0

[5 rows x 29 columns]
```

In [16]:
```
# Creating dummy variables for the remaining categorical variables and dropping the l

# Creating dummy variables for the variable 'MultipleLines'
ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
# Dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'], 1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ml1], axis=1)

# Creating dummy variables for the variable 'OnlineSecurity'.
os = pd.get_dummies(telecom['OnlineSecurity'], prefix='OnlineSecurity')
os1 = os.drop(['OnlineSecurity_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,os1], axis=1)
```

```
# Creating dummy variables for the variable 'OnlineBackup'.
ob = pd.get_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')
ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ob1], axis=1)

# Creating dummy variables for the variable 'DeviceProtection'.
dp = pd.get_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,dp1], axis=1)

# Creating dummy variables for the variable 'TechSupport'.
ts = pd.get_dummies(telecom['TechSupport'], prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,ts1], axis=1)

# Creating dummy variables for the variable 'StreamingTV'.
st =pd.get_dummies(telecom['StreamingTV'], prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,st1], axis=1)

# Creating dummy variables for the variable 'StreamingMovies'.
sm = pd.get_dummies(telecom['StreamingMovies'], prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'], 1)
# Adding the results to the master dataframe
telecom = pd.concat([telecom,sm1], axis=1)
```

In [17]:
```
telecom.head()
```

Out[17]:
```
   customerID  tenure  PhoneService        Contract  PaperlessBilling  \
0  7590-VHVEG       1             0  Month-to-month                 1
1  5575-GNVDE      34             1        One year                 0
2  3668-QPYBK       2             1  Month-to-month                 1
3  7795-CFOCW      45             0        One year                 0
4  9237-HQITU       2             1  Month-to-month                 1

               PaymentMethod  MonthlyCharges TotalCharges  Churn  gender  \
0           Electronic check           29.85        29.85      0  Female
1               Mailed check           56.95       1889.5      0    Male
2               Mailed check           53.85       108.15      1    Male
3  Bank transfer (automatic)           42.30      1840.75      0    Male
4           Electronic check           70.70       151.65      1  Female

          ...          OnlineBackup_No  OnlineBackup_Yes  DeviceProtection_No  \
```

```
0        ...                       0                 1                    1
1        ...                       1                 0                    0
2        ...                       0                 1                    1
3        ...                       1                 0                    0
4        ...                       1                 0                    1

  DeviceProtection_Yes TechSupport_No TechSupport_Yes StreamingTV_No  \
0                    0              1               0              1
1                    1              1               0              1
2                    0              1               0              1
3                    1              0               1              1
4                    0              1               0              1

  StreamingTV_Yes StreamingMovies_No StreamingMovies_Yes
0               0                  1                   0
1               0                  1                   0
2               0                  1                   0
3               0                  1                   0
4               0                  1                   0

[5 rows x 43 columns]
```

Dropping the repeated variables

```
In [18]: # We have created dummies for the below variables, so we can drop them
         telecom = telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','Internet
                'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)

In [19]: #The varaible was imported as a string we need to convert it to float
         telecom['TotalCharges'] = telecom['TotalCharges'].convert_objects(convert_numeric=Tru

In [20]: telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 32 columns):
customerID                              7043 non-null object
tenure                                  7043 non-null int64
PhoneService                            7043 non-null int64
PaperlessBilling                        7043 non-null int64
MonthlyCharges                          7043 non-null float64
TotalCharges                            7032 non-null float64
Churn                                   7043 non-null int64
SeniorCitizen                           7043 non-null int64
Partner                                 7043 non-null int64
Dependents                              7043 non-null int64
Contract_One year                       7043 non-null uint8
Contract_Two year                       7043 non-null uint8
```

```
PaymentMethod_Credit card (automatic)    7043 non-null uint8
PaymentMethod_Electronic check           7043 non-null uint8
PaymentMethod_Mailed check               7043 non-null uint8
gender_Male                              7043 non-null uint8
InternetService_Fiber optic              7043 non-null uint8
InternetService_No                       7043 non-null uint8
MultipleLines_No                         7043 non-null uint8
MultipleLines_Yes                        7043 non-null uint8
OnlineSecurity_No                        7043 non-null uint8
OnlineSecurity_Yes                       7043 non-null uint8
OnlineBackup_No                          7043 non-null uint8
OnlineBackup_Yes                         7043 non-null uint8
DeviceProtection_No                      7043 non-null uint8
DeviceProtection_Yes                     7043 non-null uint8
TechSupport_No                           7043 non-null uint8
TechSupport_Yes                          7043 non-null uint8
StreamingTV_No                           7043 non-null uint8
StreamingTV_Yes                          7043 non-null uint8
StreamingMovies_No                       7043 non-null uint8
StreamingMovies_Yes                      7043 non-null uint8
dtypes: float64(2), int64(7), object(1), uint8(22)
memory usage: 756.6+ KB
```

Now you can see that you have all variables as numeric.

Checking for Outliers

In [21]:
```
# Checking for outliers in the continuous variables
num_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharges']]
```

In [22]:
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_telecom.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[22]:

	tenure	MonthlyCharges	SeniorCitizen	TotalCharges
count	7043.000000	7043.000000	7043.000000	7032.000000
mean	32.371149	64.761692	0.162147	2283.300441
std	24.559481	30.090047	0.368612	2266.771362
min	0.000000	18.250000	0.000000	18.800000
25%	9.000000	35.500000	0.000000	401.450000
50%	29.000000	70.350000	0.000000	1397.475000
75%	55.000000	89.850000	0.000000	3794.737500
90%	69.000000	102.600000	1.000000	5976.640000
95%	72.000000	107.400000	1.000000	6923.590000
99%	72.000000	114.729000	1.000000	8039.883000
max	72.000000	118.750000	1.000000	8684.800000

From the distribution shown above, you can see that there no outliers in your data. The numbers are gradually increasing.

Checking for Missing Values and Inputing Them

```
In [23]: # Adding up the missing values (column-wise)
         telecom.isnull().sum()
```

```
Out[23]: customerID                                0
         tenure                                    0
         PhoneService                              0
         PaperlessBilling                          0
         MonthlyCharges                            0
         TotalCharges                             11
         Churn                                     0
         SeniorCitizen                             0
         Partner                                   0
         Dependents                                0
         Contract_One year                         0
         Contract_Two year                         0
         PaymentMethod_Credit card (automatic)     0
         PaymentMethod_Electronic check            0
         PaymentMethod_Mailed check                0
         gender_Male                               0
         InternetService_Fiber optic               0
         InternetService_No                        0
         MultipleLines_No                          0
         MultipleLines_Yes                         0
         OnlineSecurity_No                         0
         OnlineSecurity_Yes                        0
         OnlineBackup_No                           0
         OnlineBackup_Yes                          0
         DeviceProtection_No                       0
         DeviceProtection_Yes                      0
         TechSupport_No                            0
         TechSupport_Yes                           0
         StreamingTV_No                            0
         StreamingTV_Yes                           0
         StreamingMovies_No                        0
         StreamingMovies_Yes                       0
         dtype: int64
```

It means that 11/7043 = 0.001561834 i.e 0.1%, best is to remove these observations from the analysis

```
In [24]: # Checking the percentage of missing values
         round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

```
Out[24]: customerID                               0.00
         tenure                                   0.00
         PhoneService                             0.00
         PaperlessBilling                         0.00
```

```
MonthlyCharges                           0.00
TotalCharges                             0.16
Churn                                    0.00
SeniorCitizen                            0.00
Partner                                  0.00
Dependents                               0.00
Contract_One year                        0.00
Contract_Two year                        0.00
PaymentMethod_Credit card (automatic)    0.00
PaymentMethod_Electronic check           0.00
PaymentMethod_Mailed check               0.00
gender_Male                              0.00
InternetService_Fiber optic              0.00
InternetService_No                       0.00
MultipleLines_No                         0.00
MultipleLines_Yes                        0.00
OnlineSecurity_No                        0.00
OnlineSecurity_Yes                       0.00
OnlineBackup_No                          0.00
OnlineBackup_Yes                         0.00
DeviceProtection_No                      0.00
DeviceProtection_Yes                     0.00
TechSupport_No                           0.00
TechSupport_Yes                          0.00
StreamingTV_No                           0.00
StreamingTV_Yes                          0.00
StreamingMovies_No                       0.00
StreamingMovies_Yes                      0.00
dtype: float64
```

In [25]:
```
# Removing NaN TotalCharges rows
telecom = telecom[~np.isnan(telecom['TotalCharges'])]
```

In [26]:
```
# Checking percentage of missing values after removing the missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Out[26]:
```
customerID                               0.0
tenure                                   0.0
PhoneService                             0.0
PaperlessBilling                         0.0
MonthlyCharges                           0.0
TotalCharges                             0.0
Churn                                    0.0
SeniorCitizen                            0.0
Partner                                  0.0
Dependents                               0.0
Contract_One year                        0.0
Contract_Two year                        0.0
```

```
PaymentMethod_Credit card (automatic)    0.0
PaymentMethod_Electronic check           0.0
PaymentMethod_Mailed check               0.0
gender_Male                              0.0
InternetService_Fiber optic              0.0
InternetService_No                       0.0
MultipleLines_No                         0.0
MultipleLines_Yes                        0.0
OnlineSecurity_No                        0.0
OnlineSecurity_Yes                       0.0
OnlineBackup_No                          0.0
OnlineBackup_Yes                         0.0
DeviceProtection_No                      0.0
DeviceProtection_Yes                     0.0
TechSupport_No                           0.0
TechSupport_Yes                          0.0
StreamingTV_No                           0.0
StreamingTV_Yes                          0.0
StreamingMovies_No                       0.0
StreamingMovies_Yes                      0.0
dtype: float64
```

Now we don't have any missing values

0.1.4 Step 4: Test-Train Split

In [27]:
```python
from sklearn.model_selection import train_test_split
```

In [28]:
```python
# Putting feature variable to X
X = telecom.drop(['Churn','customerID'], axis=1)

X.head()
```

Out[28]:
```
   tenure  PhoneService  PaperlessBilling  MonthlyCharges  TotalCharges  \
0       1             0                 1           29.85         29.85
1      34             1                 0           56.95       1889.50
2       2             1                 1           53.85        108.15
3      45             0                 0           42.30       1840.75
4       2             1                 1           70.70        151.65

   SeniorCitizen  Partner  Dependents  Contract_One year  Contract_Two year  \
0              0        1           0                  0                  0
1              0        0           0                  1                  0
2              0        0           0                  0                  0
3              0        0           0                  1                  0
4              0        0           0                  0                  0

          ...            OnlineBackup_No  OnlineBackup_Yes  \
0         ...                          0                 1
```

```
1         ...                          1                0
2         ...                          0                1
3         ...                          1                0
4         ...                          1                0

   DeviceProtection_No  DeviceProtection_Yes  TechSupport_No  TechSupport_Yes  \
0                    1                     0               1                0
1                    0                     1               1                0
2                    1                     0               1                0
3                    0                     1               0                1
4                    1                     0               1                0

   StreamingTV_No  StreamingTV_Yes  StreamingMovies_No  StreamingMovies_Yes
0               1                0                   1                    0
1               1                0                   1                    0
2               1                0                   1                    0
3               1                0                   1                    0
4               1                0                   1                    0

[5 rows x 30 columns]
```

In [29]:
```
# Putting response variable to y
y = telecom['Churn']

y.head()
```

Out[29]:
```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

In [30]:
```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0
```

0.1.5 Step 5: Feature Scaling

In [31]:
```
from sklearn.preprocessing import StandardScaler
```

In [32]:
```
scaler = StandardScaler()

X_train[['tenure','MonthlyCharges','TotalCharges']] = scaler.fit_transform(X_train[['

X_train.head()
```

Out[32]:
```
        tenure  PhoneService  PaperlessBilling  MonthlyCharges  TotalCharges  \
879   0.019693             1                 1       -0.338074     -0.276449
5790  0.305384             0                 1       -0.464443     -0.112702
```

```
6498 -1.286319              1                 1         0.581425     -0.974430
880  -0.919003              1                 1         1.505913     -0.550676
2784 -1.163880              1                 1         1.106854     -0.835971

      SeniorCitizen  Partner  Dependents  Contract_One year  \
879               0        0           0                  0
5790              0        1           1                  0
6498              0        0           0                  0
880               0        0           0                  0
2784              0        0           1                  0

      Contract_Two year         ...           OnlineBackup_No  \
879                   0         ...                         0
5790                  0         ...                         0
6498                  0         ...                         0
880                   0         ...                         0
2784                  0         ...                         1

      OnlineBackup_Yes  DeviceProtection_No  DeviceProtection_Yes  \
879                  1                    1                     0
5790                 1                    1                     0
6498                 1                    0                     1
880                  1                    0                     1
2784                 0                    0                     1

      TechSupport_No  TechSupport_Yes  StreamingTV_No  StreamingTV_Yes  \
879                1                0               1                0
5790               1                0               0                1
6498               1                0               1                0
880                0                1               0                1
2784               0                1               0                1

      StreamingMovies_No  StreamingMovies_Yes
879                    1                    0
5790                   0                    1
6498                   1                    0
880                    0                    1
2784                   0                    1

[5 rows x 30 columns]
```

In [33]:
```
### Checking the Churn Rate
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn
```

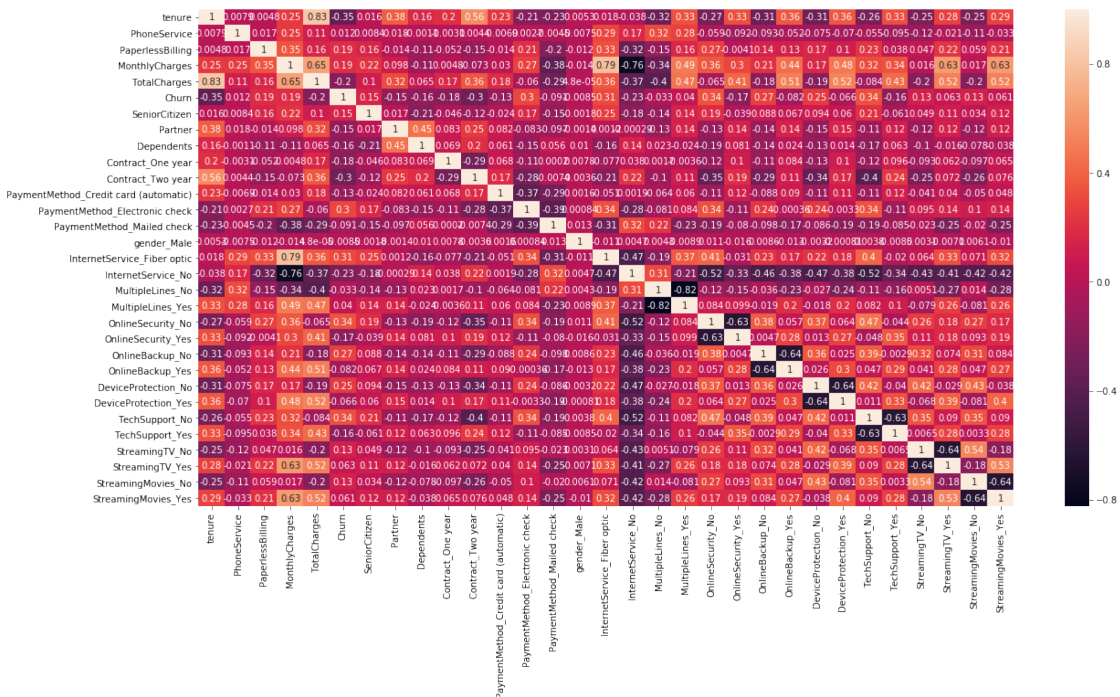
Out[33]:
```
26.578498293515356
```

We have almost 27% churn rate

0.1.6 Step 6: Looking at Correlations

```
In [34]: # Importing matplotlib and seaborn
         import matplotlib.pyplot as plt
         import seaborn as sns
         %matplotlib inline
```

```
In [35]: # Let's see the correlation matrix
         plt.figure(figsize = (20,10))        # Size of the figure
         sns.heatmap(telecom.corr(),annot = True)
         plt.show()
```

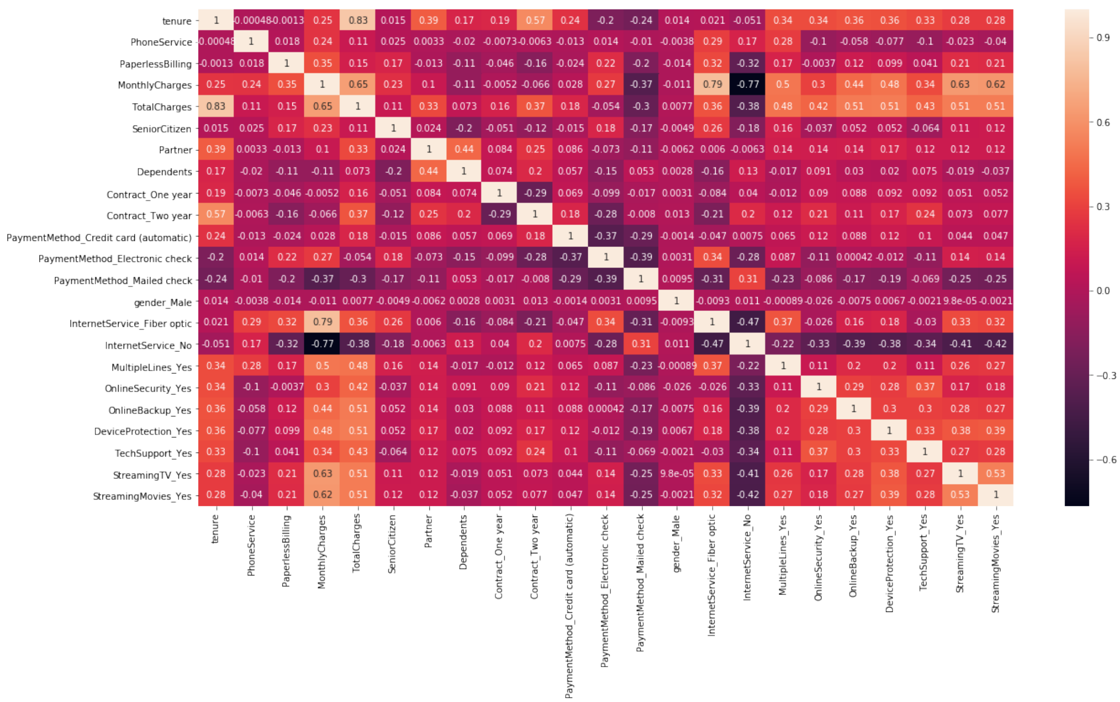


Dropping highly correlated dummy variables

```
In [36]: X_test = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','Devic
                               'StreamingTV_No','StreamingMovies_No'], 1)
         X_train = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','Dev
                                 'StreamingTV_No','StreamingMovies_No'], 1)
```

Checking the Correlation Matrix After dropping highly correlated variables now let's check the correlation matrix again.

```
In [37]: plt.figure(figsize = (20,10))
         sns.heatmap(X_train.corr(),annot = True)
         plt.show()
```



0.1.7 Step 7: Model Building

Let's start by splitting our data into a training set and a test set.

Running Your First Training Model

In [38]:
```
import statsmodels.api as sm
```

In [39]:
```
# Logistic regression model
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

Out[39]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  Churn   No. Observations:                 4922
Model:                            GLM   Df Residuals:                     4898
Model Family:                Binomial   Df Model:                           23
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2004.7
Date:                Thu, 29 Nov 2018   Deviance:                       4009.4
Time:                        11:23:01   Pearson chi2:                 6.07e+03
No. Iterations:                     7   Covariance Type:             nonrobust
==================================================================================
```

```
                                       coef    std err          z      P>|z|
---------------------------------------------------------------------------------
const                               -3.9382      1.546     -2.547      0.011
tenure                              -1.5172      0.189     -8.015      0.000
PhoneService                         0.9507      0.789      1.205      0.228
PaperlessBilling                     0.3254      0.090      3.614      0.000
MonthlyCharges                      -2.1806      1.160     -1.880      0.060
TotalCharges                         0.7332      0.198      3.705      0.000
SeniorCitizen                        0.3984      0.102      3.924      0.000
Partner                              0.0374      0.094      0.399      0.690
Dependents                          -0.1430      0.107     -1.332      0.183
Contract_One year                   -0.6578      0.129     -5.106      0.000
Contract_Two year                   -1.2455      0.212     -5.874      0.000
PaymentMethod_Credit card (automatic)  -0.2577      0.137     -1.883      0.060
PaymentMethod_Electronic check       0.1615      0.113      1.434      0.152
PaymentMethod_Mailed check          -0.2536      0.137     -1.845      0.065
gender_Male                         -0.0346      0.078     -0.442      0.658
InternetService_Fiber optic          2.5124      0.967      2.599      0.009
InternetService_No                  -2.7792      0.982     -2.831      0.005
MultipleLines_Yes                    0.5623      0.214      2.628      0.009
OnlineSecurity_Yes                  -0.0245      0.216     -0.113      0.910
OnlineBackup_Yes                     0.1740      0.212      0.822      0.411
DeviceProtection_Yes                 0.3229      0.215      1.501      0.133
TechSupport_Yes                     -0.0305      0.216     -0.141      0.888
StreamingTV_Yes                      0.9598      0.396      2.423      0.015
StreamingMovies_Yes                  0.8484      0.396      2.143      0.032
=================================================================================
"""
```

0.1.8 Step 8: Feature Selection Using RFE

In [40]:
```python
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [41]:
```python
from sklearn.feature_selection import RFE
rfe = RFE(logreg, 15)             # running RFE with 13 variables as output
rfe = rfe.fit(X_train, y_train)
```

In [42]:
```python
rfe.support_
```

Out[42]:
```
array([ True,  True,  True, False,  True,  True, False, False,  True,
        True,  True, False,  True, False,  True,  True,  True,  True,
       False, False,  True,  True, False])
```

In [43]:
```python
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[43]:
```
[('tenure', True, 1),
 ('PhoneService', True, 1),
 ('PaperlessBilling', True, 1),
```

```
 ('MonthlyCharges', False, 6),
 ('TotalCharges', True, 1),
 ('SeniorCitizen', True, 1),
 ('Partner', False, 8),
 ('Dependents', False, 4),
 ('Contract_One year', True, 1),
 ('Contract_Two year', True, 1),
 ('PaymentMethod_Credit card (automatic)', True, 1),
 ('PaymentMethod_Electronic check', False, 3),
 ('PaymentMethod_Mailed check', True, 1),
 ('gender_Male', False, 9),
 ('InternetService_Fiber optic', True, 1),
 ('InternetService_No', True, 1),
 ('MultipleLines_Yes', True, 1),
 ('OnlineSecurity_Yes', True, 1),
 ('OnlineBackup_Yes', False, 2),
 ('DeviceProtection_Yes', False, 7),
 ('TechSupport_Yes', True, 1),
 ('StreamingTV_Yes', True, 1),
 ('StreamingMovies_Yes', False, 5)]
```

```
In [44]: col = X_train.columns[rfe.support_]
```

```
In [45]: X_train.columns[~rfe.support_]
```

```
Out[45]: Index(['MonthlyCharges', 'Partner', 'Dependents',
                'PaymentMethod_Electronic check', 'gender_Male', 'OnlineBackup_Yes',
                'DeviceProtection_Yes', 'StreamingMovies_Yes'],
               dtype='object')
```

Assessing the model with StatsModels

```
In [46]: X_train_sm = sm.add_constant(X_train[col])
         logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
         res = logm2.fit()
         res.summary()
```

```
Out[46]: <class 'statsmodels.iolib.summary.Summary'>
         """
                          Generalized Linear Model Regression Results
         ==============================================================================
         Dep. Variable:                  Churn   No. Observations:                 4922
         Model:                            GLM   Df Residuals:                     4906
         Model Family:                Binomial   Df Model:                           15
         Link Function:                  logit   Scale:                          1.0000
         Method:                          IRLS   Log-Likelihood:                -2011.8
         Date:                Thu, 29 Nov 2018   Deviance:                       4023.5
         Time:                        11:23:04   Pearson chi2:                 6.22e+03
         No. Iterations:                     7   Covariance Type:             nonrobust
```

```
==========================================================================================
                                             coef    std err          z      P>|z|
------------------------------------------------------------------------------------------
const                                     -1.0343      0.171     -6.053      0.000
tenure                                    -1.5386      0.184     -8.381      0.000
PhoneService                              -0.5231      0.161     -3.256      0.001
PaperlessBilling                           0.3397      0.090      3.789      0.000
TotalCharges                               0.7116      0.188      3.794      0.000
SeniorCitizen                              0.4294      0.100      4.312      0.000
Contract_One year                         -0.6813      0.128     -5.334      0.000
Contract_Two year                         -1.2680      0.211     -6.011      0.000
PaymentMethod_Credit card (automatic)     -0.3775      0.113     -3.352      0.001
PaymentMethod_Mailed check                -0.3760      0.111     -3.389      0.001
InternetService_Fiber optic                0.7421      0.117      6.317      0.000
InternetService_No                        -0.9385      0.166     -5.650      0.000
MultipleLines_Yes                          0.2086      0.096      2.181      0.029
OnlineSecurity_Yes                        -0.4049      0.102     -3.968      0.000
TechSupport_Yes                           -0.3967      0.102     -3.902      0.000
StreamingTV_Yes                            0.2747      0.094      2.911      0.004
==========================================================================================
"""
```

In [47]:
```python
# Getting the predicted values on the train set
y_train_pred = res.predict(X_train_sm)
y_train_pred[:10]
```

Out[47]:
```
879     0.225111
5790    0.274893
6498    0.692126
880     0.504909
2784    0.645261
3874    0.417544
5387    0.420131
6623    0.809427
4465    0.223211
5364    0.512246
dtype: float64
```

In [48]:
```python
y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred[:10]
```

Out[48]:
```
array([0.22511138, 0.27489289, 0.69212611, 0.50490896, 0.6452606 ,
       0.41754449, 0.42013086, 0.80942651, 0.2232105 , 0.51224637])
```

Creating a dataframe with the actual churn flag and the predicted probabilities

In [49]:
```python
y_train_pred_final = pd.DataFrame({'Churn':y_train.values, 'Churn_Prob':y_train_pred})
y_train_pred_final['CustID'] = y_train.index
y_train_pred_final.head()
```

```
Out[49]:    Churn  Churn_Prob  CustID
         0      0    0.225111     879
         1      0    0.274893    5790
         2      1    0.692126    6498
         3      1    0.504909     880
         4      1    0.645261    2784
```

Creating new column 'predicted' with 1 if Churn_Prob > 0.5 else 0

```
In [50]: y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x >

         # Let's see the head
         y_train_pred_final.head()
```

```
Out[50]:    Churn  Churn_Prob  CustID  predicted
         0      0    0.225111     879          0
         1      0    0.274893    5790          0
         2      1    0.692126    6498          1
         3      1    0.504909     880          1
         4      1    0.645261    2784          1
```

```
In [51]: from sklearn import metrics
```

```
In [52]: # Confusion matrix
         confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.pred
         print(confusion)
```

```
[[3270  365]
 [ 579  708]]
```

```
In [53]: # Predicted     not_churn    churn
         # Actual
         # not_churn        3270      365
         # churn            579       708
```

```
In [54]: # Let's check the overall accuracy.
         print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted))
```

```
0.8082080455099553
```

Checking VIFs

```
In [55]: # Check for the VIF values of the feature variables.
         from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
In [56]: # Create a dataframe that will contain the names of all the feature variables and the
         vif = pd.DataFrame()
```

```
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_trai
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

Out[56]:

	Features	VIF
1	PhoneService	8.86
3	TotalCharges	7.37
0	tenure	6.88
9	InternetService_Fiber optic	3.97
6	Contract_Two year	3.28
10	InternetService_No	3.25
2	PaperlessBilling	2.68
11	MultipleLines_Yes	2.53
14	StreamingTV_Yes	2.34
13	TechSupport_Yes	2.08
5	Contract_One year	1.93
12	OnlineSecurity_Yes	1.90
8	PaymentMethod_Mailed check	1.72
7	PaymentMethod_Credit card (automatic)	1.46
4	SeniorCitizen	1.31

There are a few variables with high VIF. It's best to drop these variables as they aren't helping much with prediction and unnecessarily making the model complex. The variable 'PhoneService' has the highest VIF. So let's start by dropping that.

In [57]:
```
col = col.drop('PhoneService', 1)
col
```

Out[57]:
```
Index(['tenure', 'PaperlessBilling', 'TotalCharges', 'SeniorCitizen',
       'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Credit card (automatic)', 'PaymentMethod_Mailed check',
       'InternetService_Fiber optic', 'InternetService_No',
       'MultipleLines_Yes', 'OnlineSecurity_Yes', 'TechSupport_Yes',
       'StreamingTV_Yes'],
      dtype='object')
```

In [58]:
```
# Let's re-run the model using the selected variables
X_train_sm = sm.add_constant(X_train[col])
logm3 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
res = logm3.fit()
res.summary()
```

Out[58]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  Churn   No. Observations:                 4922
```

```
Model:                            GLM   Df Residuals:                     4907
Model Family:                Binomial   Df Model:                           14
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2017.0
Date:                Thu, 29 Nov 2018   Deviance:                       4034.0
Time:                        11:23:05   Pearson chi2:                 5.94e+03
No. Iterations:                     7   Covariance Type:             nonrobust
==================================================================================
                                            coef    std err          z      P>|z|
----------------------------------------------------------------------------------
const                                    -1.3885      0.133    -10.437      0.000
tenure                                   -1.4138      0.179     -7.884      0.000
PaperlessBilling                          0.3425      0.089      3.829      0.000
TotalCharges                              0.5936      0.184      3.225      0.001
SeniorCitizen                             0.4457      0.099      4.486      0.000
Contract_One year                        -0.6905      0.128     -5.411      0.000
Contract_Two year                        -1.2646      0.211     -6.002      0.000
PaymentMethod_Credit card (automatic)    -0.3785      0.113     -3.363      0.001
PaymentMethod_Mailed check               -0.3769      0.111     -3.407      0.001
InternetService_Fiber optic               0.6241      0.111      5.645      0.000
InternetService_No                       -1.0940      0.158     -6.919      0.000
MultipleLines_Yes                         0.1607      0.094      1.712      0.087
OnlineSecurity_Yes                       -0.4094      0.102     -4.016      0.000
TechSupport_Yes                          -0.4085      0.101     -4.025      0.000
StreamingTV_Yes                           0.3077      0.094      3.277      0.001
==================================================================================
"""
```

In [59]:
```python
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [60]:
```python
y_train_pred[:10]
```

Out[60]:
```
array([0.25403236, 0.22497676, 0.69386521, 0.51008735, 0.65172434,
       0.45441958, 0.3272777 , 0.80583357, 0.17618503, 0.50403034])
```

In [61]:
```python
y_train_pred_final['Churn_Prob'] = y_train_pred
```

In [62]:
```python
# Creating new column 'predicted' with 1 if Churn_Prob > 0.5 else 0
y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x >
y_train_pred_final.head()
```

Out[62]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.254032	879	0
1	0	0.224977	5790	0
2	1	0.693865	6498	1
3	1	0.510087	880	1
4	1	0.651724	2784	1

In [63]:
```python
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted))
```

0.8051605038602194

So overall the accuracy hasn't dropped much.

Let's check the VIFs again

```
In [64]: vif = pd.DataFrame()
         vif['Features'] = X_train[col].columns
         vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_trai
         vif['VIF'] = round(vif['VIF'], 2)
         vif = vif.sort_values(by = "VIF", ascending = False)
         vif
```

Out[64]:

	Features	VIF
2	TotalCharges	7.30
0	tenure	6.79
5	Contract_Two year	3.16
8	InternetService_Fiber optic	2.94
9	InternetService_No	2.53
1	PaperlessBilling	2.52
13	StreamingTV_Yes	2.31
10	MultipleLines_Yes	2.27
12	TechSupport_Yes	2.00
4	Contract_One year	1.83
11	OnlineSecurity_Yes	1.80
7	PaymentMethod_Mailed check	1.66
6	PaymentMethod_Credit card (automatic)	1.44
3	SeniorCitizen	1.31

```
In [65]: # Let's drop TotalCharges since it has a high VIF
         col = col.drop('TotalCharges')
         col
```

```
Out[65]: Index(['tenure', 'PaperlessBilling', 'SeniorCitizen', 'Contract_One year',
                'Contract_Two year', 'PaymentMethod_Credit card (automatic)',
                'PaymentMethod_Mailed check', 'InternetService_Fiber optic',
                'InternetService_No', 'MultipleLines_Yes', 'OnlineSecurity_Yes',
                'TechSupport_Yes', 'StreamingTV_Yes'],
               dtype='object')
```

```
In [66]: # Let's re-run the model using the selected variables
         X_train_sm = sm.add_constant(X_train[col])
         logm4 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
         res = logm4.fit()
         res.summary()
```

```
Out[66]: <class 'statsmodels.iolib.summary.Summary'>
         """
```

```
                 Generalized Linear Model Regression Results
==============================================================================
Dep. Variable:                  Churn   No. Observations:                 4922
Model:                            GLM   Df Residuals:                     4908
Model Family:                Binomial   Df Model:                           13
Link Function:                  logit   Scale:                          1.0000
Method:                          IRLS   Log-Likelihood:                -2022.5
Date:                Thu, 29 Nov 2018   Deviance:                       4044.9
Time:                        11:23:06   Pearson chi2:                 5.22e+03
No. Iterations:                     7   Covariance Type:             nonrobust
==============================================================================
                                            coef    std err          z      P>|z|
-----------------------------------------------------------------------------------
const                                    -1.4695      0.130    -11.336      0.000
tenure                                   -0.8857      0.065    -13.553      0.000
PaperlessBilling                          0.3367      0.089      3.770      0.000
SeniorCitizen                             0.4517      0.100      4.527      0.000
Contract_One year                        -0.6792      0.127     -5.360      0.000
Contract_Two year                        -1.2308      0.208     -5.903      0.000
PaymentMethod_Credit card (automatic)    -0.3827      0.113     -3.399      0.001
PaymentMethod_Mailed check               -0.3393      0.110     -3.094      0.002
InternetService_Fiber optic               0.7914      0.098      8.109      0.000
InternetService_No                       -1.1205      0.157     -7.127      0.000
MultipleLines_Yes                         0.2166      0.092      2.355      0.019
OnlineSecurity_Yes                       -0.3739      0.101     -3.684      0.000
TechSupport_Yes                          -0.3611      0.101     -3.591      0.000
StreamingTV_Yes                           0.3995      0.089      4.465      0.000
===================================================================================
"""
```

In [67]:
```python
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [68]:
```python
y_train_pred[:10]
```

Out[68]:
```
array([0.28219274, 0.2681923 , 0.68953115, 0.53421409, 0.67433213,
       0.42980951, 0.31009304, 0.81248467, 0.20462744, 0.50431479])
```

In [69]:
```python
y_train_pred_final['Churn_Prob'] = y_train_pred
```

In [70]:
```python
# Creating new column 'predicted' with 1 if Churn_Prob > 0.5 else 0
y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x >
y_train_pred_final.head()
```

Out[70]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.282193	879	0
1	0	0.268192	5790	0
2	1	0.689531	6498	1
3	1	0.534214	880	1
4	1	0.674332	2784	1

```
In [71]: # Let's check the overall accuracy.
         print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted))
```

```
0.804754164973588
```

The accuracy is still practically the same.

Let's now check the VIFs again

```
In [72]: vif = pd.DataFrame()
         vif['Features'] = X_train[col].columns
         vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_trai
         vif['VIF'] = round(vif['VIF'], 2)
         vif = vif.sort_values(by = "VIF", ascending = False)
         vif
```

Out[72]:

	Features	VIF
4	Contract_Two year	3.07
7	InternetService_Fiber optic	2.60
1	PaperlessBilling	2.44
9	MultipleLines_Yes	2.24
12	StreamingTV_Yes	2.17
8	InternetService_No	2.12
0	tenure	2.04
11	TechSupport_Yes	1.98
3	Contract_One year	1.82
10	OnlineSecurity_Yes	1.78
6	PaymentMethod_Mailed check	1.66
5	PaymentMethod_Credit card (automatic)	1.44
2	SeniorCitizen	1.31

All variables have a good value of VIF. So we need not drop any more variables and we can proceed with making predictions using this model only

```
In [73]: # Let's take a look at the confusion matrix again
         confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.pred
         confusion
```

```
Out[73]: array([[3269,  366],
                [ 595,  692]], dtype=int64)
```

```
In [74]: # Actual/Predicted     not_churn    churn
                 # not_churn        3269       366
                 # churn            595        692
```

```
In [75]: # Let's check the overall accuracy.
         metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

```
Out[75]: 0.804754164973588
```

0.2 Metrics beyond simply accuracy

```
In [76]: TP = confusion[1,1] # true positive
         TN = confusion[0,0] # true negatives
         FP = confusion[0,1] # false positives
         FN = confusion[1,0] # false negatives
```

```
In [77]: # Let's see the sensitivity of our logistic regression model
         TP / float(TP+FN)
```

```
Out[77]: 0.5376845376845377
```

```
In [78]: # Let us calculate specificity
         TN / float(TN+FP)
```

```
Out[78]: 0.8993122420907841
```

```
In [79]: # Calculate false postive rate - predicting churn when customer does not have churned
         print(FP/ float(TN+FP))
```

```
0.10068775790921596
```

```
In [80]: # positive predictive value
         print (TP / float(TP+FP))
```

```
0.6540642722117203
```

```
In [81]: # Negative predictive value
         print (TN / float(TN+ FN))
```

```
0.8460144927536232
```

0.2.1 Step 9: Plotting the ROC Curve

An ROC curve demonstrates several things:

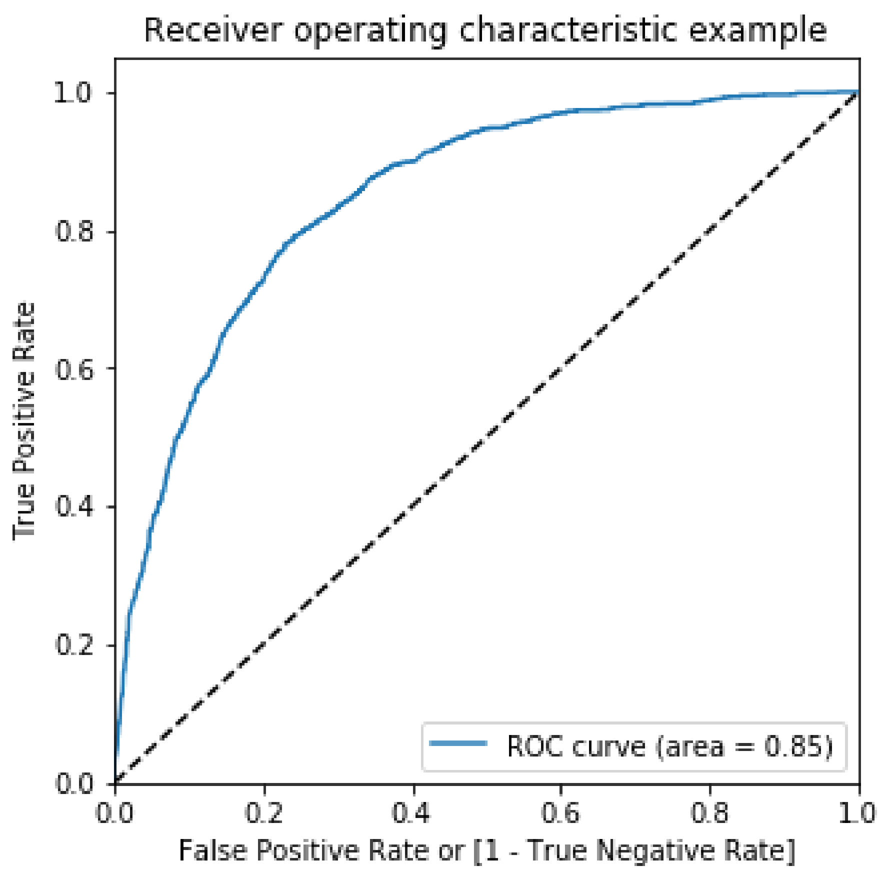
- It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
- The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
- The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

```
In [82]: def draw_roc( actual, probs ):
             fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                                       drop_intermediate = False )
             auc_score = metrics.roc_auc_score( actual, probs )
```

```
        plt.figure(figsize=(5, 5))
        plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
        plt.plot([0, 1], [0, 1], 'k--')
        plt.xlim([0.0, 1.0])
        plt.ylim([0.0, 1.05])
        plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
        plt.ylabel('True Positive Rate')
        plt.title('Receiver operating characteristic example')
        plt.legend(loc="lower right")
        plt.show()

        return None
```

```
In [83]: fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Churn, y_train_pred_final
```

```
In [84]: draw_roc(y_train_pred_final.Churn, y_train_pred_final.Churn_Prob)
```



0.2.2 Step 10: Finding Optimal Cutoff Point

Optimal cutoff probability is that prob where we get balanced sensitivity and specificity

In [85]:
```python
# Let's create columns with different probability cutoffs
numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_pred_final[i]= y_train_pred_final.Churn_Prob.map(lambda x: 1 if x > i else
y_train_pred_final.head()
```

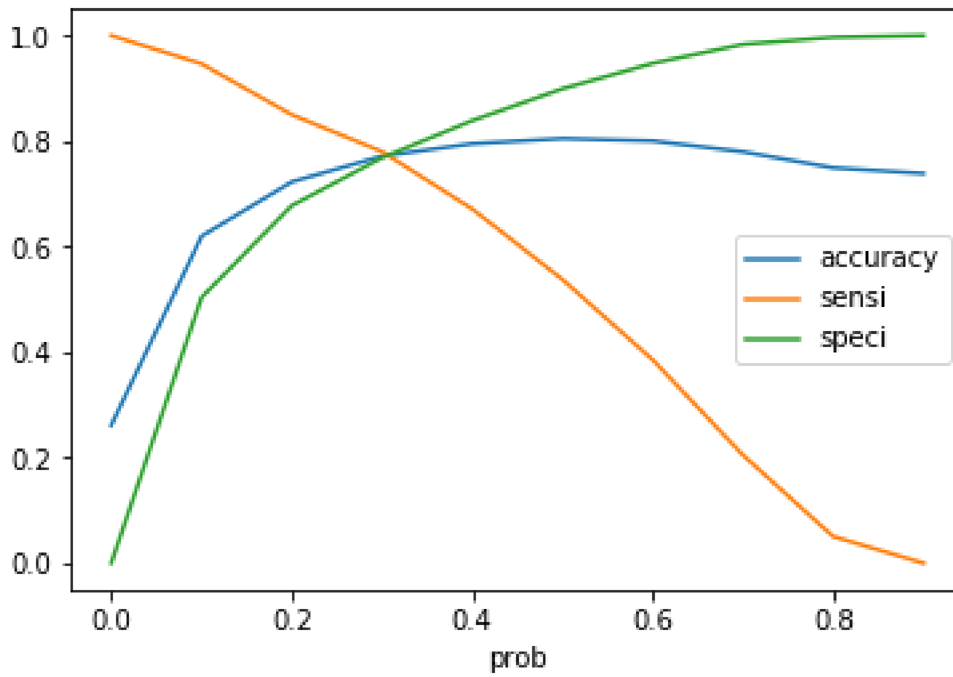
Out[85]:
```
   Churn  Churn_Prob  CustID  predicted  0.0  0.1  0.2  0.3  0.4  0.5  0.6  \
0      0    0.282193     879          0    1    1    1    0    0    0    0
1      0    0.268192    5790          0    1    1    1    0    0    0    0
2      1    0.689531    6498          1    1    1    1    1    1    1    1
3      1    0.534214     880          1    1    1    1    1    1    1    0
4      1    0.674332    2784          1    1    1    1    1    1    1    1

   0.7  0.8  0.9
0    0    0    0
1    0    0    0
2    0    0    0
3    0    0    0
4    0    0    0
```

In [86]:
```python
# Now let's calculate accuracy sensitivity and specificity for various probability cu
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

```
     prob  accuracy     sensi     speci
0.0   0.0  0.261479  1.000000  0.000000
0.1   0.1  0.619667  0.946387  0.503989
0.2   0.2  0.722674  0.850039  0.677579
0.3   0.3  0.771434  0.780109  0.768363
0.4   0.4  0.795002  0.671329  0.838790
0.5   0.5  0.804754  0.537685  0.899312
0.6   0.6  0.800284  0.385392  0.947180
```

```
0.7   0.7  0.779764  0.205128  0.983219
0.8   0.8  0.749289  0.050505  0.996699
0.9   0.9  0.738521  0.000000  1.000000
```

In [87]:
```python
# Let's plot accuracy sensitivity and specificity for various probabilities.
cutoff_df.plot.line(x='prob', y=['accuracy','sensi','speci'])
plt.show()
```



From the curve above, 0.3 is the optimum point to take it as a cutoff probability.

In [88]:
```python
y_train_pred_final['final_predicted'] = y_train_pred_final.Churn_Prob.map( lambda x: 

y_train_pred_final.head()
```

Out[88]:
```
   Churn  Churn_Prob  CustID  predicted  0.0  0.1  0.2  0.3  0.4  0.5  0.6  \
0      0    0.282193     879          0    1    1    1    0    0    0    0
1      0    0.268192    5790          0    1    1    1    0    0    0    0
2      1    0.689531    6498          1    1    1    1    1    1    1    1
3      1    0.534214     880          1    1    1    1    1    1    1    0
4      1    0.674332    2784          1    1    1    1    1    1    1    1

   0.7  0.8  0.9  final_predicted
0    0    0    0                0
1    0    0    0                0
```

```
2    0    0    0              1
3    0    0    0              1
4    0    0    0              1
```

```
In [89]: # Let's check the overall accuracy.
         metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.final_predicted)
```

```
Out[89]: 0.771434376269809
```

```
In [90]: confusion2 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.fi
         confusion2
```

```
Out[90]: array([[2793,  842],
                [ 283, 1004]], dtype=int64)
```

```
In [91]: TP = confusion2[1,1] # true positive
         TN = confusion2[0,0] # true negatives
         FP = confusion2[0,1] # false positives
         FN = confusion2[1,0] # false negatives
```

```
In [92]: # Let's see the sensitivity of our logistic regression model
         TP / float(TP+FN)
```

```
Out[92]: 0.7801087801087802
```

```
In [93]: # Let us calculate specificity
         TN / float(TN+FP)
```

```
Out[93]: 0.768363136176066
```

```
In [94]: # Calculate false postive rate - predicting churn when customer does not have churned
         print(FP/ float(TN+FP))
```

```
0.23163686382393398
```

```
In [95]: # Positive predictive value
         print (TP / float(TP+FP))
```

```
0.5438786565547129
```

```
In [96]: # Negative predictive value
         print (TN / float(TN+ FN))
```

```
0.907997399219766
```

0.3 Precision and Recall

```
In [97]: #Looking at the confusion matrix again
```

```
In [98]: confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.pred
         confusion
```

```
Out[98]: array([[3269,  366],
               [ 595,  692]], dtype=int64)
```

Precision TP / TP + FP

```
In [99]: confusion[1,1]/(confusion[0,1]+confusion[1,1])
```

```
Out[99]: 0.6540642722117203
```

Recall TP / TP + FN

```
In [100]: confusion[1,1]/(confusion[1,0]+confusion[1,1])
```

```
Out[100]: 0.5376845376845377
```

Using sklearn utilities for the same

```
In [101]: from sklearn.metrics import precision_score, recall_score
```

```
In [102]: ?precision_score
```

```
In [103]: precision_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

```
Out[103]: 0.6540642722117203
```

```
In [104]: recall_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

```
Out[104]: 0.5376845376845377
```

0.3.1 Precision and recall tradeoff

```
In [105]: from sklearn.metrics import precision_recall_curve
```

```
In [106]: y_train_pred_final.Churn, y_train_pred_final.predicted
```

```
Out[106]: (0        0
           1        0
           2        1
           3        1
           4        1
           5        0
           6        0
           7        1
           8        0
```

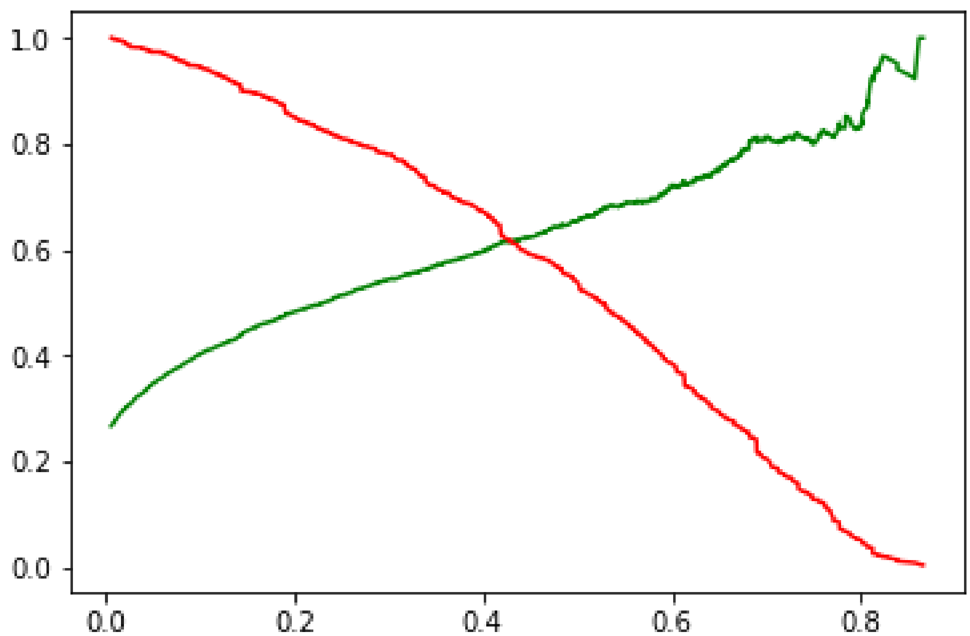
9	1
10	0
11	1
12	1
13	0
14	0
15	0
16	0
17	0
18	0
19	0
20	0
21	0
22	0
23	0
24	0
25	0
26	0
27	0
28	0
29	0
	..
4892	1
4893	1
4894	0
4895	0
4896	0
4897	0
4898	0
4899	0
4900	0
4901	1
4902	0
4903	1
4904	0
4905	0
4906	1
4907	0
4908	0
4909	1
4910	0
4911	0
4912	0
4913	0
4914	0
4915	0
4916	1
4917	0

```
4918    0
4919    0
4920    0
4921    0
Name: Churn, Length: 4922, dtype: int64, 0       0
1       0
2       1
3       1
4       1
5       0
6       0
7       1
8       0
9       1
10      0
11      1
12      1
13      0
14      0
15      0
16      0
17      0
18      0
19      0
20      0
21      0
22      0
23      0
24      0
25      0
26      0
27      0
28      0
29      0
       ..
4892    0
4893    1
4894    0
4895    0
4896    0
4897    0
4898    0
4899    0
4900    0
4901    0
4902    0
4903    0
4904    1
```

```
4905    0
4906    1
4907    0
4908    0
4909    1
4910    0
4911    0
4912    0
4913    0
4914    0
4915    0
4916    0
4917    0
4918    0
4919    0
4920    0
4921    0
Name: predicted, Length: 4922, dtype: int64)
```

In [107]:
```
p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn, y_train_pred_fina
```

In [108]:
```
plt.plot(thresholds, p[:-1], "g-")
plt.plot(thresholds, r[:-1], "r-")
plt.show()
```



0.3.2 Step 11: Making predictions on the test set

```
In [109]: X_test[['tenure','MonthlyCharges','TotalCharges']] = scaler.transform(X_test[['tenure
```

```
In [110]: X_test = X_test[col]
          X_test.head()
```

```
Out[110]:         tenure  PaperlessBilling  SeniorCitizen  Contract_One year  \
          942  -0.347623                 1              0                  0
          3730  0.999203                 1              0                  0
          1761  1.040015                 1              0                  0
          2283 -1.286319                 1              0                  0
          1872  0.346196                 0              0                  0

                Contract_Two year  PaymentMethod_Credit card (automatic)  \
          942                   0                                      1
          3730                  0                                      1
          1761                  1                                      1
          2283                  0                                      0
          1872                  1                                      0

                PaymentMethod_Mailed check  InternetService_Fiber optic  \
          942                            0                            1
          3730                           0                            1
          1761                           0                            0
          2283                           1                            1
          1872                           0                            0

                InternetService_No  MultipleLines_Yes  OnlineSecurity_Yes  \
          942                    0                  0                   0
          3730                   0                  1                   0
          1761                   1                  1                   0
          2283                   0                  0                   0
          1872                   1                  0                   0

                TechSupport_Yes  StreamingTV_Yes
          942                 0                0
          3730                0                1
          1761                0                0
          2283                0                0
          1872                0                0
```

```
In [111]: X_test_sm = sm.add_constant(X_test)
```

Making predictions on the test set

```
In [112]: y_test_pred = res.predict(X_test_sm)
```

```
In [113]: y_test_pred[:10]
```

Out[113]:
```
942     0.397413
3730    0.270295
1761    0.010238
2283    0.612692
1872    0.015869
1970    0.727206
2532    0.302131
1616    0.010315
2485    0.632881
5914    0.126451
dtype: float64
```

In [114]:
```
# Converting y_pred to a dataframe which is an array
y_pred_1 = pd.DataFrame(y_test_pred)
```

In [115]:
```
# Let's see the head
y_pred_1.head()
```

Out[115]:

	0
942	0.397413
3730	0.270295
1761	0.010238
2283	0.612692
1872	0.015869

In [116]:
```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
```

In [117]:
```
# Putting CustID to index
y_test_df['CustID'] = y_test_df.index
```

In [118]:
```
# Removing index for both dataframes to append them side by side
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
```

In [119]:
```
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

In [120]:
```
y_pred_final.head()
```

Out[120]:

	Churn	CustID	0
0	0	942	0.397413
1	1	3730	0.270295
2	0	1761	0.010238
3	1	2283	0.612692
4	0	1872	0.015869

In [121]:
```
# Renaming the column
y_pred_final= y_pred_final.rename(columns={ 0 : 'Churn_Prob'})
```

```
In [122]: # Rearranging the columns
          y_pred_final = y_pred_final.reindex_axis(['CustID','Churn','Churn_Prob'], axis=1)

In [123]: # Let's see the head of y_pred_final
          y_pred_final.head()

Out[123]:    CustID  Churn  Churn_Prob
          0     942      0    0.397413
          1    3730      1    0.270295
          2    1761      0    0.010238
          3    2283      1    0.612692
          4    1872      0    0.015869

In [124]: y_pred_final['final_predicted'] = y_pred_final.Churn_Prob.map(lambda x: 1 if x > 0.42

In [125]: y_pred_final.head()

Out[125]:    CustID  Churn  Churn_Prob  final_predicted
          0     942      0    0.397413                0
          1    3730      1    0.270295                0
          2    1761      0    0.010238                0
          3    2283      1    0.612692                1
          4    1872      0    0.015869                0

In [126]: # Let's check the overall accuracy.
          metrics.accuracy_score(y_pred_final.Churn, y_pred_final.final_predicted)

Out[126]: 0.7834123222748816

In [127]: confusion2 = metrics.confusion_matrix(y_pred_final.Churn, y_pred_final.final_predicte
          confusion2

Out[127]: array([[1294,  234],
                 [ 223,  359]], dtype=int64)

In [128]: TP = confusion2[1,1] # true positive
          TN = confusion2[0,0] # true negatives
          FP = confusion2[0,1] # false positives
          FN = confusion2[1,0] # false negatives

In [129]: # Let's see the sensitivity of our logistic regression model
          TP / float(TP+FN)

Out[129]: 0.6168384879725086

In [130]: # Let us calculate specificity
          TN / float(TN+FP)

Out[130]: 0.8468586387434555
```