logistic regression115

September 17, 2025

0.0.1 Importing and Merging Data

[3]:
```
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

[5]:
```
# Importing Pandas and NumPy
import pandas as pd, numpy as np
```

[7]:
```
# Importing all datasets
churn_data = pd.read_csv("churn_data.csv")
churn_data.head()
```

[7]:
```
   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
```

[11]:
```
customer_data = pd.read_csv("customer_data.csv")
customer_data.head()
```

[11]:
```
   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
```

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

```
[13]:    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

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

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

0.0.2 reading and understanding

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

```
[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 OnlineSecurity  \
     0     Yes         No  No phone service             DSL             No
     1      No         No                No             DSL            Yes
     2      No         No                No             DSL            Yes
```

```
3       No          No  No phone service             DSL             Yes
4       No          No                No     Fiber optic              No

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

[5 rows x 21 columns]
```

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

[9]:
```
(7043, 21)
```

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

[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

[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):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   customerID        7043 non-null   object
 1   tenure            7043 non-null   int64
 2   PhoneService      7043 non-null   object
 3   Contract          7043 non-null   object
 4   PaperlessBilling  7043 non-null   object
 5   PaymentMethod     7043 non-null   object
 6   MonthlyCharges    7043 non-null   float64
 7   TotalCharges      7043 non-null   object
 8   Churn             7043 non-null   object
 9   gender            7043 non-null   object
```

```
 10  SeniorCitizen     7043 non-null   int64
 11  Partner           7043 non-null   object
 12  Dependents        7043 non-null   object
 13  MultipleLines     7043 non-null   object
 14  InternetService   7043 non-null   object
 15  OnlineSecurity    7043 non-null   object
 16  OnlineBackup      7043 non-null   object
 17  DeviceProtection  7043 non-null   object
 18  TechSupport       7043 non-null   object
 19  StreamingTV       7043 non-null   object
 20  StreamingMovies   7043 non-null   object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.2+ MB
```

0.0.3 Data Preparation

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

[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)
```

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

[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 OnlineSecurity  \
0        1           0  No phone service             DSL             No
1        0           0                No             DSL            Yes
```

```
2        0            0                No             DSL             Yes
3        0            0  No phone service             DSL             Yes
4        0            0                No     Fiber optic              No

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

[5 rows x 21 columns]
```

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

[14]:
```python
# Creating a dummy variable for some of the categorical variables and dropping␣
↪the first one.
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender',␣
↪'InternetService']], drop_first=True)

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

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

[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 Contract_Two year  \
0          No               No                  0                 0
1          No               No                  1                 0
2          No               No                  0                 0
3          No               No                  1                 0
4          No               No                  0                 0

  PaymentMethod_Credit card (automatic) PaymentMethod_Electronic check  \
```

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

  PaymentMethod_Mailed check gender_Male InternetService_Fiber optic  \
0                          0           0                           0
1                          1           1                           0
2                          1           1                           0
3                          0           1                           0
4                          0           0                           1

  InternetService_No
0                  0
1                  0
2                  0
3                  0
4                  0

[5 rows x 29 columns]
```

[16]:

```
# Creating dummy variables for the remaining categorical variables and dropping␣
↪the level with big names.

# 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)
```

```python
# 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)
```

[17]:

```python
telecom.head()
```

[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

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

[19]:
```
#The varaible was imported as a string we need to convert it to float
telecom['TotalCharges'] = pd.to_numeric(telecom['TotalCharges'],errors='coerce')
```

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

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 32 columns):
 #   Column                                 Non-Null Count  Dtype
---  ------                                 --------------  -----
 0   customerID                             7043 non-null   object
 1   tenure                                 7043 non-null   int64
 2   PhoneService                           7043 non-null   int64
 3   PaperlessBilling                       7043 non-null   int64
 4   MonthlyCharges                         7043 non-null   float64
 5   TotalCharges                           7032 non-null   float64
 6   Churn                                  7043 non-null   int64
 7   SeniorCitizen                          7043 non-null   int64
 8   Partner                                7043 non-null   int64
 9   Dependents                             7043 non-null   int64
 10  Contract_One year                      7043 non-null   uint8
 11  Contract_Two year                      7043 non-null   uint8
 12  PaymentMethod_Credit card (automatic)  7043 non-null   uint8
 13  PaymentMethod_Electronic check         7043 non-null   uint8
 14  PaymentMethod_Mailed check             7043 non-null   uint8
 15  gender_Male                            7043 non-null   uint8
 16  InternetService_Fiber optic            7043 non-null   uint8
```

```
 17  InternetService_No                      7043 non-null   uint8
 18  MultipleLines_No                        7043 non-null   uint8
 19  MultipleLines_Yes                       7043 non-null   uint8
 20  OnlineSecurity_No                       7043 non-null   uint8
 21  OnlineSecurity_Yes                      7043 non-null   uint8
 22  OnlineBackup_No                         7043 non-null   uint8
 23  OnlineBackup_Yes                        7043 non-null   uint8
 24  DeviceProtection_No                     7043 non-null   uint8
 25  DeviceProtection_Yes                    7043 non-null   uint8
 26  TechSupport_No                          7043 non-null   uint8
 27  TechSupport_Yes                         7043 non-null   uint8
 28  StreamingTV_No                          7043 non-null   uint8
 29  StreamingTV_Yes                         7043 non-null   uint8
 30  StreamingMovies_No                      7043 non-null   uint8
 31  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

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

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

[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

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

[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

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

[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
```

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

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

[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

0.0.4 Step 4: Test-Train Split

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

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

X.head()
```

[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  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 30 columns]
```

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

y.head()
```

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

[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=0.3, random_state=100)
```

0.0.5 Step 5: Feature Scaling

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

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

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

X_train.head()
```

[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  OnlineBackup_Yes  \
879                   0  …                0                 1
5790                  0  …                0                 1
6498                  0  …                0                 1
880                   0  …                0                 1
2784                  0  …                1                 0

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

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

      StreamingMovies_Yes
879                     0
5790                    1
6498                    0
880                     1
2784                    1

[5 rows x 30 columns]
```

[33]:

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

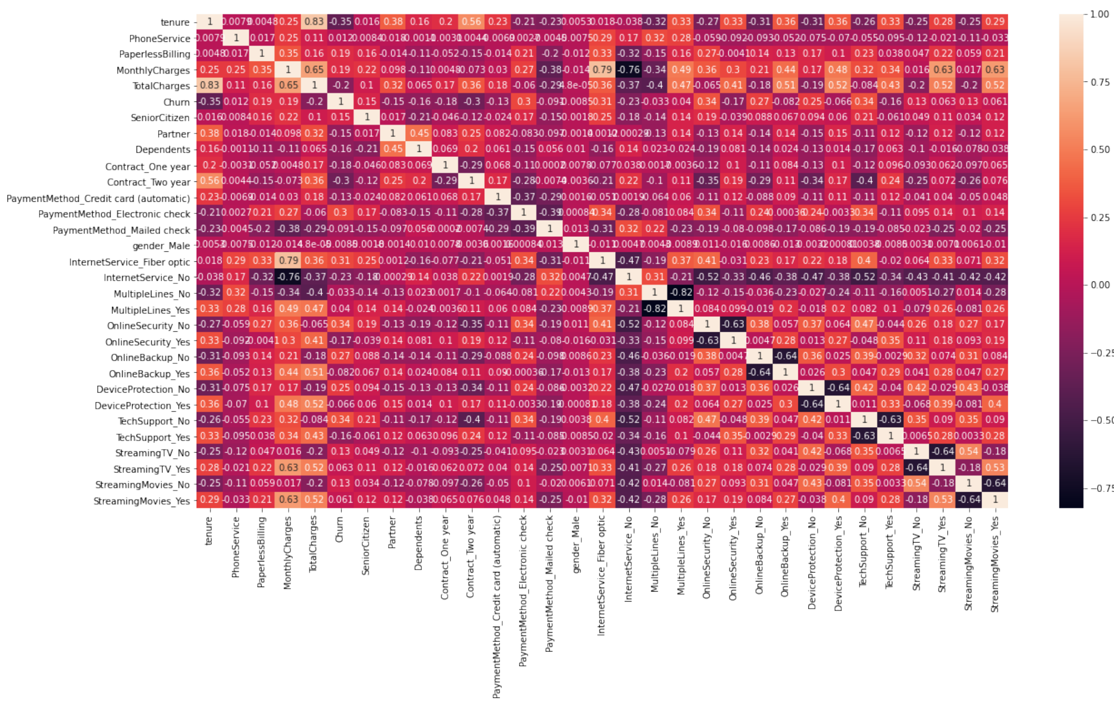
[33]: 26.578498293515356

We have almost 27% churn rate

0.0.6 Step 6: Looking at Correlations

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

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

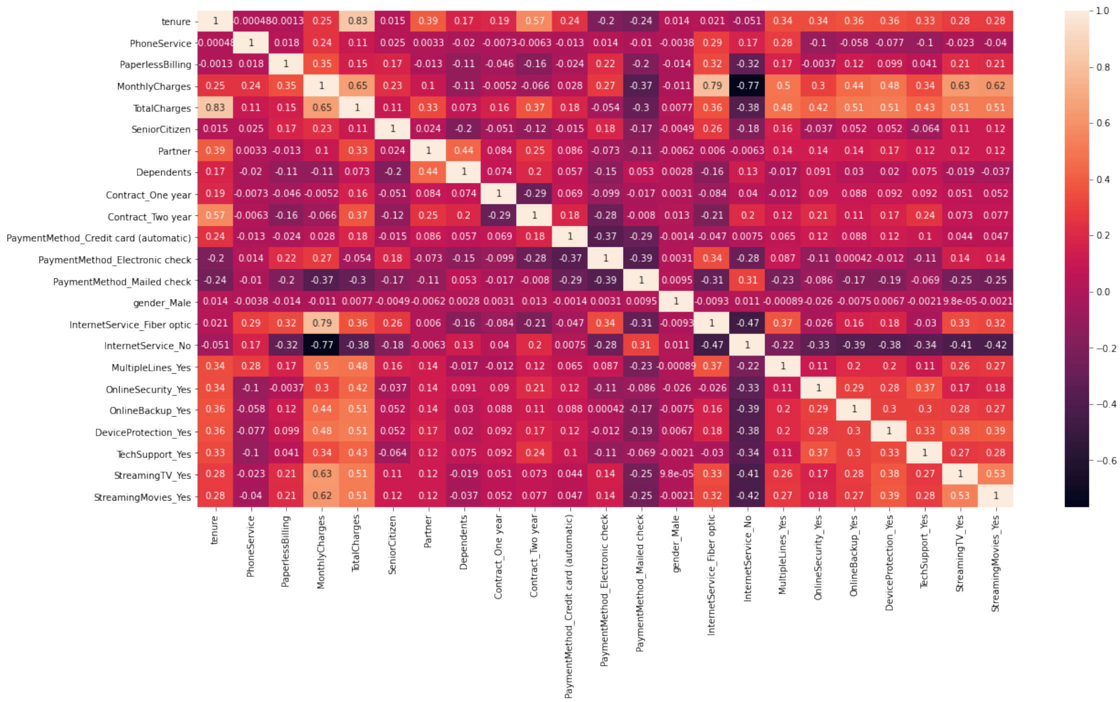


Dropping highly correlated dummy variables

[36]:
```python
X_test = X_test.
 ↪drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechS
                    'StreamingTV_No','StreamingMovies_No'], 1)
X_train = X_train.
 ↪drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechS
                     'StreamingTV_No','StreamingMovies_No'], 1)
```

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

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



0.0.7 Step 7: Model Building

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

Running Your First Training Model

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

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

[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:                Wed, 03 Jun 2020   Deviance:                       4009.4
Time:                        11:16:04   Pearson chi2:                 6.07e+03
No. Iterations:                     7
Covariance Type:            nonrobust
================================================================================
=========================
                                            coef    std err          z
P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
-------------------------
const                                    -3.9382      1.546     -2.547
0.011      -6.969      -0.908
tenure                                   -1.5172      0.189     -8.015
0.000      -1.888      -1.146
PhoneService                              0.9507      0.789      1.205
0.228      -0.595       2.497
PaperlessBilling                          0.3254      0.090      3.614
0.000       0.149       0.502
MonthlyCharges                           -2.1806      1.160     -1.880
0.060      -4.454       0.092
TotalCharges                              0.7332      0.198      3.705
0.000       0.345       1.121
SeniorCitizen                             0.3984      0.102      3.924
0.000       0.199       0.597
Partner                                   0.0374      0.094      0.399
0.690      -0.146       0.221
Dependents                               -0.1430      0.107     -1.332
0.183      -0.353       0.067
Contract_One year                        -0.6578      0.129     -5.106
0.000      -0.910      -0.405
Contract_Two year                        -1.2455      0.212     -5.874
0.000      -1.661      -0.830
PaymentMethod_Credit card (automatic)    -0.2577      0.137     -1.883
0.060      -0.526       0.011
PaymentMethod_Electronic check            0.1615      0.113      1.434
0.152      -0.059       0.382
PaymentMethod_Mailed check               -0.2536      0.137     -1.845
0.065      -0.523       0.016
gender_Male                              -0.0346      0.078     -0.442
0.658      -0.188       0.119
InternetService_Fiber optic               2.5124      0.967      2.599
0.009       0.618       4.407
InternetService_No                       -2.7792      0.982     -2.831
0.005      -4.703      -0.855
```

```
MultipleLines_Yes                               0.5623      0.214      2.628
0.009      0.143      0.982
OnlineSecurity_Yes                             -0.0245      0.216     -0.113
0.910     -0.448      0.399
OnlineBackup_Yes                                0.1740      0.212      0.822
0.411     -0.241      0.589
DeviceProtection_Yes                            0.3229      0.215      1.501
0.133     -0.099      0.744
TechSupport_Yes                                -0.0305      0.216     -0.141
0.888     -0.455      0.394
StreamingTV_Yes                                 0.9598      0.396      2.423
0.015      0.183      1.736
StreamingMovies_Yes                             0.8484      0.396      2.143
0.032      0.072      1.624
================================================================================
=========================
"""
```

0.0.8 Step 8: Feature Selection Using RFE

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

[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)
```

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

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

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

[43]:
```
[('tenure', True, 1),
 ('PhoneService', False, 3),
 ('PaperlessBilling', True, 1),
 ('MonthlyCharges', True, 1),
 ('TotalCharges', True, 1),
 ('SeniorCitizen', True, 1),
 ('Partner', False, 7),
 ('Dependents', False, 6),
 ('Contract_One year', True, 1),
 ('Contract_Two year', True, 1),
 ('PaymentMethod_Credit card (automatic)', True, 1),
 ('PaymentMethod_Electronic check', False, 4),
```

```
 ('PaymentMethod_Mailed check', True, 1),
 ('gender_Male', False, 8),
 ('InternetService_Fiber optic', True, 1),
 ('InternetService_No', True, 1),
 ('MultipleLines_Yes', True, 1),
 ('OnlineSecurity_Yes', False, 2),
 ('OnlineBackup_Yes', False, 5),
 ('DeviceProtection_Yes', False, 9),
 ('TechSupport_Yes', True, 1),
 ('StreamingTV_Yes', True, 1),
 ('StreamingMovies_Yes', True, 1)]
```

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

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

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

Assessing the model with StatsModels

[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()
```

[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:                -2011.1
Date:                Wed, 03 Jun 2020   Deviance:                       4022.2
Time:                        11:16:05   Pearson chi2:                 6.25e+03
No. Iterations:                     7
Covariance Type:            nonrobust
================================================================================
=========================
                                            coef    std err          z
P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
-------------------------
const                                    -2.2462      0.189    -11.879
```

```
0.000      -2.617      -1.876
tenure                                      -1.5596      0.187     -8.334
0.000      -1.926      -1.193
PaperlessBilling                             0.3436      0.090      3.832
0.000       0.168       0.519
MonthlyCharges                              -0.9692      0.199     -4.878
0.000      -1.359      -0.580
TotalCharges                                 0.7421      0.197      3.764
0.000       0.356       1.128
SeniorCitizen                                0.4296      0.100      4.312
0.000       0.234       0.625
Contract_One year                           -0.6830      0.128     -5.342
0.000      -0.934      -0.432
Contract_Two year                           -1.2931      0.211     -6.138
0.000      -1.706      -0.880
PaymentMethod_Credit card (automatic)       -0.3724      0.113     -3.308
0.001      -0.593      -0.152
PaymentMethod_Mailed check                  -0.3723      0.111     -3.345
0.001      -0.591      -0.154
InternetService_Fiber optic                  1.5865      0.216      7.342
0.000       1.163       2.010
InternetService_No                          -1.6897      0.216     -7.830
0.000      -2.113      -1.267
MultipleLines_Yes                            0.3779      0.104      3.640
0.000       0.174       0.581
TechSupport_Yes                             -0.2408      0.109     -2.210
0.027      -0.454      -0.027
StreamingTV_Yes                              0.5796      0.114      5.102
0.000       0.357       0.802
StreamingMovies_Yes                          0.4665      0.111      4.197
0.000       0.249       0.684
================================================================================
=========================
"""
```

[47]:

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

[47]:

```
879     0.192642
5790    0.275624
6498    0.599507
880     0.513571
2784    0.648233
3874    0.414846
5387    0.431184
6623    0.801788
```

```
4465    0.228194
5364    0.504575
dtype: float64
```

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

[48]:
```
array([0.19264205, 0.27562384, 0.59950707, 0.51357126, 0.64823272,
       0.41484553, 0.43118361, 0.80178789, 0.22819404, 0.50457542])
```

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

[49]:
```
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()
```

[49]:

	Churn	Churn_Prob	CustID
0	0	0.192642	879
1	0	0.275624	5790
2	1	0.599507	6498
3	1	0.513571	880
4	1	0.648233	2784

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

[50]:
```
y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1␣
 ↪if x > 0.5 else 0)

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

[50]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.192642	879	0
1	0	0.275624	5790	0
2	1	0.599507	6498	1
3	1	0.513571	880	1
4	1	0.648233	2784	1

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

[52]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Churn,␣
 ↪y_train_pred_final.predicted )
print(confusion)
```

```
[[3275  360]
 [ 574  713]]
```

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

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

```
0.8102397399431126
```

Checking VIFs

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

[56]:
```
# Create a dataframe that will contain the names of all the feature variables␣
 ↪and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in␣
 ↪range(X_train[col].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[56]:

	Features	VIF
2	MonthlyCharges	14.85
3	TotalCharges	10.42
0	tenure	7.38
9	InternetService_Fiber optic	5.61
10	InternetService_No	5.27
6	Contract_Two year	3.14
13	StreamingTV_Yes	2.79
14	StreamingMovies_Yes	2.79
1	PaperlessBilling	2.76
11	MultipleLines_Yes	2.38
12	TechSupport_Yes	1.95
5	Contract_One year	1.85
8	PaymentMethod_Mailed check	1.73
7	PaymentMethod_Credit card (automatic)	1.45
4	SeniorCitizen	1.33

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.

[57]:
```
# 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()
```

[57]:
```
<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:                -2011.1
Date:                Wed, 03 Jun 2020   Deviance:                       4022.2
Time:                        11:16:05   Pearson chi2:                 6.25e+03
No. Iterations:                     7
Covariance Type:            nonrobust
================================================================================
=========================
                                            coef    std err          z
P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
-------------------------
const                                    -2.2462      0.189    -11.879
0.000      -2.617      -1.876
tenure                                   -1.5596      0.187     -8.334
0.000      -1.926      -1.193
PaperlessBilling                          0.3436      0.090      3.832
0.000       0.168       0.519
MonthlyCharges                           -0.9692      0.199     -4.878
0.000      -1.359      -0.580
TotalCharges                              0.7421      0.197      3.764
0.000       0.356       1.128
SeniorCitizen                             0.4296      0.100      4.312
0.000       0.234       0.625
Contract_One year                        -0.6830      0.128     -5.342
0.000      -0.934      -0.432
Contract_Two year                        -1.2931      0.211     -6.138
0.000      -1.706      -0.880
PaymentMethod_Credit card (automatic)    -0.3724      0.113     -3.308
0.001      -0.593      -0.152
PaymentMethod_Mailed check               -0.3723      0.111     -3.345
0.001      -0.591      -0.154
InternetService_Fiber optic               1.5865      0.216      7.342
0.000       1.163       2.010
InternetService_No                       -1.6897      0.216     -7.830
0.000      -2.113      -1.267
```

```
MultipleLines_Yes                              0.3779      0.104      3.640
0.000      0.174      0.581
TechSupport_Yes                               -0.2408      0.109     -2.210
0.027     -0.454     -0.027
StreamingTV_Yes                                0.5796      0.114      5.102
0.000      0.357      0.802
StreamingMovies_Yes                            0.4665      0.111      4.197
0.000      0.249      0.684
================================================================================
=========================
"""
```

[58]:
```python
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

[59]:
```python
y_train_pred[:10]
```

[59]:
```
array([0.19264205, 0.27562384, 0.59950707, 0.51357126, 0.64823272,
       0.41484553, 0.43118361, 0.80178789, 0.22819404, 0.50457542])
```

[60]:
```python
y_train_pred_final['Churn_Prob'] = y_train_pred
```

[61]:
```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 > 0.5 else 0)
y_train_pred_final.head()
```

[61]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.192642	879	0
1	0	0.275624	5790	0
2	1	0.599507	6498	1
3	1	0.513571	880	1
4	1	0.648233	2784	1

[62]:
```python
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.
 ↪predicted))
```

```
0.8102397399431126
```

So overall the accuracy hasn't dropped much.

Let's check the VIFs again

[63]:
```python
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in␣
 ↪range(X_train[col].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
```

```
vif
```

[63]:

```
                                 Features    VIF
2                          MonthlyCharges  14.85
3                            TotalCharges  10.42
0                                  tenure   7.38
9             InternetService_Fiber optic   5.61
10                     InternetService_No   5.27
6                       Contract_Two year   3.14
13                        StreamingTV_Yes   2.79
14                    StreamingMovies_Yes   2.79
1                        PaperlessBilling   2.76
11                       MultipleLines_Yes   2.38
12                        TechSupport_Yes   1.95
5                       Contract_One year   1.85
8              PaymentMethod_Mailed check   1.73
7   PaymentMethod_Credit card (automatic)   1.45
4                           SeniorCitizen   1.33
```

[64]:

```
# Let's drop TotalCharges since it has a high VIF
col = col.drop('TotalCharges')
col
```

[64]:

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

[65]:

```
# 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()
```

[65]:

```
<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:                -2018.5
Date:                Wed, 03 Jun 2020   Deviance:                       4037.1
```

```
Time:                         11:16:05   Pearson chi2:                 5.25e+03
No. Iterations:                      7
Covariance Type:             nonrobust
================================================================================
=========================
                                            coef    std err          z
P>|z|      [0.025      0.975]
--------------------------------------------------------------------------------
-------------------------
const                                    -2.1697      0.186    -11.663
0.000      -2.534      -1.805
tenure                                   -0.9137      0.065    -13.982
0.000      -1.042      -0.786
PaperlessBilling                          0.3332      0.089      3.726
0.000       0.158       0.508
MonthlyCharges                           -0.7106      0.184     -3.854
0.000      -1.072      -0.349
SeniorCitizen                             0.4407      0.100      4.404
0.000       0.245       0.637
Contract_One year                        -0.6821      0.127     -5.374
0.000      -0.931      -0.433
Contract_Two year                        -1.2558      0.208     -6.034
0.000      -1.664      -0.848
PaymentMethod_Credit card (automatic)    -0.3774      0.113     -3.348
0.001      -0.598      -0.156
PaymentMethod_Mailed check               -0.3207      0.110     -2.917
0.004      -0.536      -0.105
InternetService_Fiber optic               1.5264      0.213      7.166
0.000       1.109       1.944
InternetService_No                       -1.5165      0.208     -7.278
0.000      -1.925      -1.108
MultipleLines_Yes                         0.3872      0.104      3.739
0.000       0.184       0.590
TechSupport_Yes                          -0.2426      0.109     -2.224
0.026      -0.456      -0.029
StreamingTV_Yes                           0.5779      0.113      5.126
0.000       0.357       0.799
StreamingMovies_Yes                       0.4667      0.110      4.226
0.000       0.250       0.683
================================================================================
=========================
"""
```

[66]:
```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

[67]:
```
y_train_pred[:10]
```

[67]: array([0.22669491, 0.32279486, 0.61112237, 0.56497818, 0.68324356,
       0.38658503, 0.36867571, 0.80505887, 0.25567371, 0.52400218])

[68]:
```
y_train_pred_final['Churn_Prob'] = y_train_pred
```

[69]:
```
# 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 > 0.5 else 0)
y_train_pred_final.head()
```

[69]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.226695	879	0
1	0	0.322795	5790	0
2	1	0.611122	6498	1
3	1	0.564978	880	1
4	1	0.683244	2784	1

[70]:
```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.
↪predicted))
```

0.8063795205201137

The accuracy is still practically the same.

Let's now check the VIFs again

[71]:
```
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in␣
↪range(X_train[col].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[71]:

	Features	VIF
2	MonthlyCharges	10.63
8	InternetService_Fiber optic	5.44
9	InternetService_No	5.15
5	Contract_Two year	3.13
12	StreamingTV_Yes	2.79
13	StreamingMovies_Yes	2.79
1	PaperlessBilling	2.76
0	tenure	2.38
10	MultipleLines_Yes	2.38
11	TechSupport_Yes	1.94
4	Contract_One year	1.85
7	PaymentMethod_Mailed check	1.69
6	PaymentMethod_Credit card (automatic)	1.45

```
3                          SeniorCitizen   1.33
```

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

[72]:
```python
# Let's take a look at the confusion matrix again
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.predicted )
confusion
```

[72]:
```
array([[3278,  357],
       [ 596,  691]])
```

[73]:
```python
# Actual/Predicted     not_churn    churn
        # not_churn        3269      366
        # churn            595       692
```

[74]:
```python
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

[74]:
```
0.8063795205201137
```

0.1 Metrics beyond simply accuracy

[75]:
```python
TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

[76]:
```python
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

[76]:
```
0.5369075369075369
```

[77]:
```python
# Let us calculate specificity
TN / float(TN+FP)
```

[77]:
```
0.9017881705639614
```

[78]:
```python
# Calculate false postive rate - predicting churn when customer does not have churned
print(FP/ float(TN+FP))
```

```
0.09821182943603851
```

[79]:
```python
# positive predictive value
print (TP / float(TP+FP))
```

```
0.6593511450381679
```

[80]:
```python
# Negative predictive value
print (TN / float(TN+ FN))
```

0.8461538461538461

0.1.1 Step 9: Plotting the ROC Curve

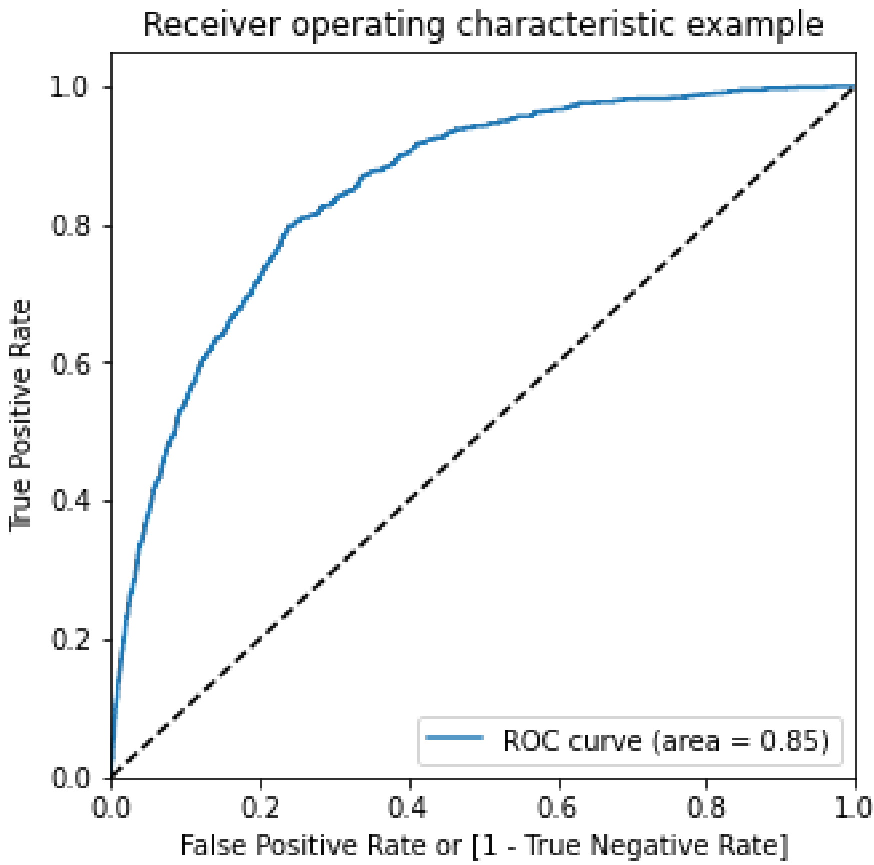
An ROC curve demonstrates several things:

- 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.

[81]:
```python
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
```

[82]:
```python
fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Churn,␣
↪y_train_pred_final.Churn_Prob, drop_intermediate = False )
```

[83]:
```python
draw_roc(y_train_pred_final.Churn, y_train_pred_final.Churn_Prob)
```



0.1.2 Step 10: Finding Optimal Cutoff Point

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

[84]:
```
# 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 0)
y_train_pred_final.head()
```

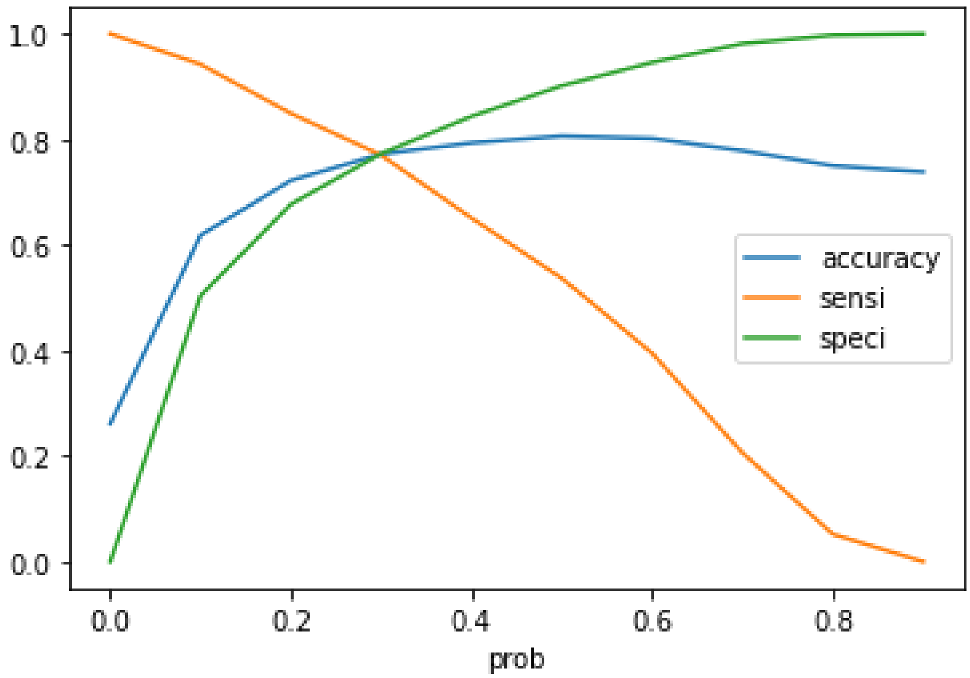
[84]:
```
   Churn  Churn_Prob  CustID  predicted  0.0  0.1  0.2  0.3  0.4  0.5  0.6  \
0      0    0.226695     879          0    1    1    1    0    0    0    0
1      0    0.322795    5790          0    1    1    1    1    0    0    0
2      1    0.611122    6498          1    1    1    1    1    1    1    1
3      1    0.564978     880          1    1    1    1    1    1    1    0
4      1    0.683244    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
```

[85]:
```
# Now let's calculate accuracy sensitivity and specificity for various␣
↪probability cutoffs.
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  0.618448  0.942502  0.503714
0.2   0.2  0.722267  0.849262  0.677304
0.3   0.3  0.772247  0.770008  0.773040
0.4   0.4  0.793377  0.651127  0.843741
0.5   0.5  0.806380  0.536908  0.901788
0.6   0.6  0.801910  0.394716  0.946080
0.7   0.7  0.778545  0.205905  0.981293
0.8   0.8  0.750102  0.052059  0.997249
0.9   0.9  0.738521  0.000000  1.000000
```

[86]:
```
# 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.

[87]:
```
y_train_pred_final['final_predicted'] = y_train_pred_final.Churn_Prob.map(␣
↪lambda x: 1 if x > 0.3 else 0)

y_train_pred_final.head()
```

[87]:
```
   Churn  Churn_Prob  CustID  predicted  0.0  0.1  0.2  0.3  0.4  0.5  0.6  \
0      0    0.226695     879          0    1    1    1    0    0    0    0
1      0    0.322795    5790          0    1    1    1    1    0    0    0
2      1    0.611122    6498          1    1    1    1    1    1    1    1
3      1    0.564978     880          1    1    1    1    1    1    1    0
4      1    0.683244    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                1
2    0    0    0                1
3    0    0    0                1
4    0    0    0                1
```

[88]:
```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.
↪final_predicted)
```

[88]: 0.7722470540430719

[89]:
```
confusion2 = metrics.confusion_matrix(y_train_pred_final.Churn,␣
↪y_train_pred_final.final_predicted )
confusion2
```

[89]:
```
array([[2810,  825],
       [ 296,  991]])
```

[90]:
```
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

[91]:
```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

[91]: 0.77000777000777

[92]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

[92]: 0.7730398899587345

[93]:
```
# Calculate false postive rate - predicting churn when customer does not have␣
↪churned
print(FP/ float(TN+FP))
```

0.22696011004126548

[94]:
```
# Positive predictive value
print (TP / float(TP+FP))
```

0.545704845814978

[95]:
```
# Negative predictive value
print (TN / float(TN+ FN))
```

0.9047005795235029

0.2 Precision and Recall

[96]:
```
#Looking at the confusion matrix again
```

[97]:
```
confusion = metrics.confusion_matrix(y_train_pred_final.Churn,␣
↪y_train_pred_final.predicted )
confusion
```

[97]:
```
array([[3278,  357],
       [ 596,  691]])
```

Precision TP / TP + FP

[98]:
```
confusion[1,1]/(confusion[0,1]+confusion[1,1])
```

[98]:
```
0.6593511450381679
```

Recall TP / TP + FN

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

[99]:
```
0.5369075369075369
```

Using sklearn utilities for the same

[100]:
```
from sklearn.metrics import precision_score, recall_score
```

[101]:
```
?precision_score
```

[102]:
```
precision_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

[102]:
```
0.6593511450381679
```

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

[103]:
```
0.5369075369075369
```

0.2.1 Precision and recall tradeoff

[104]:
```
from sklearn.metrics import precision_recall_curve
```

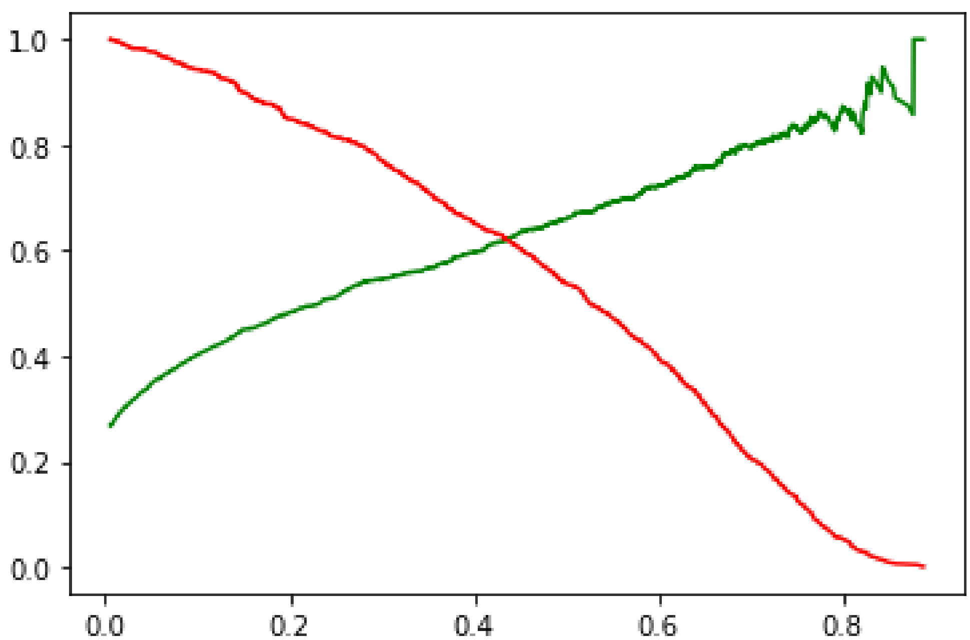
[105]:
```
y_train_pred_final.Churn, y_train_pred_final.predicted
```

[105]:
```
(0       0
 1       0
 2       1
 3       1
 4       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
       ..
4917    0
4918    0
4919    0
4920    0
4921    0
Name: predicted, Length: 4922, dtype: int64)
```

[106]:
```python
p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn,
 ↪y_train_pred_final.Churn_Prob)
```

[107]:
```python
plt.plot(thresholds, p[:-1], "g-")
plt.plot(thresholds, r[:-1], "r-")
plt.show()
```



0.2.2 Step 11: Making predictions on the test set

[108]:
```python
X_test[['tenure','MonthlyCharges','TotalCharges']] = scaler.
 ↪transform(X_test[['tenure','MonthlyCharges','TotalCharges']])
```

[109]:
```python
X_test = X_test[col]
X_test.head()
```

[109]:
```
        tenure  PaperlessBilling  MonthlyCharges  SeniorCitizen  \
942  -0.347623                 1        0.499951              0
3730  0.999203                 1        1.319685              0
1761  1.040015                 1       -1.342374              0
2283 -1.286319                 1        0.223935              0
1872  0.346196                 0       -1.500335              0

      Contract_One year  Contract_Two year  \
942                   0                  0
3730                  0                  0
1761                  0                  1
2283                  0                  0
1872                  0                  1

      PaymentMethod_Credit card (automatic)  PaymentMethod_Mailed check  \
942                                       1                           0
3730                                      1                           0
1761                                      1                           0
2283                                      0                           1
1872                                      0                           0

      InternetService_Fiber optic  InternetService_No  MultipleLines_Yes  \
942                             1                   0                  0
3730                            1                   0                  1
1761                            0                   1                  1
2283                            1                   0                  0
1872                            0                   1                  0

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

[110]:
```
X_test_sm = sm.add_constant(X_test)
```

Making predictions on the test set

[111]:
```
y_test_pred = res.predict(X_test_sm)
```

[112]:
```
y_test_pred[:10]
```

[112]:
```
942     0.435743
3730    0.248518
1761    0.009998
2283    0.595171
1872    0.014889
```

```
1970    0.697307
2532    0.284275
1616    0.009756
2485    0.598246
5914    0.131993
dtype: float64
```

[113]:
```
# Converting y_pred to a dataframe which is an array
y_pred_1 = pd.DataFrame(y_test_pred)
```

[114]:
```
# Let's see the head
y_pred_1.head()
```

[114]:

	0
942	0.435743
3730	0.248518
1761	0.009998
2283	0.595171
1872	0.014889

[115]:
```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
```

[116]:
```
# Putting CustID to index
y_test_df['CustID'] = y_test_df.index
```

[117]:
```
# 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)
```

[118]:
```
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

[119]:
```
y_pred_final.head()
```

[119]:

	Churn	CustID	0
0	0	942	0.435743
1	1	3730	0.248518
2	0	1761	0.009998
3	1	2283	0.595171
4	0	1872	0.014889

[120]:
```
# Renaming the column
y_pred_final= y_pred_final.rename(columns={ 0 : 'Churn_Prob'})
```

[121]:
```
# Rearranging the columns
y_pred_final = y_pred_final.reindex(['CustID','Churn','Churn_Prob'], axis=1)
```

[122]:
```
# Let's see the head of y_pred_final
y_pred_final.head()
```

[122]:

	CustID	Churn	Churn_Prob
0	942	0	0.435743
1	3730	1	0.248518
2	1761	0	0.009998
3	2283	1	0.595171
4	1872	0	0.014889

[123]:
```
y_pred_final['final_predicted'] = y_pred_final.Churn_Prob.map(lambda x: 1 if x␣
↪> 0.42 else 0)
```

[124]:
```
y_pred_final.head()
```

[124]:

	CustID	Churn	Churn_Prob	final_predicted
0	942	0	0.435743	1
1	3730	1	0.248518	0
2	1761	0	0.009998	0
3	2283	1	0.595171	1
4	1872	0	0.014889	0

[125]:
```
# Let's check the overall accuracy.
metrics.accuracy_score(y_pred_final.Churn, y_pred_final.final_predicted)
```

[125]: 0.7848341232227488

[126]:
```
confusion2 = metrics.confusion_matrix(y_pred_final.Churn, y_pred_final.
↪final_predicted )
confusion2
```

[126]:
```
array([[1288,  240],
       [ 214,  368]])
```

[127]:
```
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

[128]:
```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

[128]: 0.6323024054982818

[129]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

[129]: 0.8429319371727748