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.

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	PaymentMethod	MonthlyCharges	TotalCharge
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29.85	29.8
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56.95	1889.5
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53.85	108.1
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42.30	1840.7
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70.70	151.6

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

	customerID	gender	SeniorCitizen	Partner	Dependents
2	3668-QPYBK	Male	0	No	No
3	7795-CFOCW	Male	0	No	No
4	9237-HQITU	Female	0	No	No

In [5]:

```python
internet_data = pd.read_csv("internet_data.csv")
internet_data.head()
```

Out[5]:

	customerID	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	Stre
0	7590-VHVEG	No phone service	DSL	No	Yes	No	No	
1	5575-GNVDE	No	DSL	Yes	No	Yes	No	
2	3668-QPYBK	No	DSL	Yes	Yes	No	No	
3	7795-CFOCW	No phone service	DSL	Yes	No	Yes	Yes	
4	9237-HQITU	No	Fiber optic	No	No	No	No	

Combining all data files into one consolidated dataframe

In [6]:

```python
# Merging on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')
```

In [7]:

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

Step 2: Inspecting the Dataframe

In [8]:

```python
# Let's see the head of our master dataset
telecom.head()
```

Out[8]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29.85	29.8
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56.95	1889.
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53.85	108.1

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42.30	1840.75
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70.70	151.65

5 rows × 21 columns

In [9]:

```
# Let's check the dimensions of the dataframe
telecom.shape
```

Out[9]:

```
(7043, 21)
```

In [10]:

```
# 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]:

```
# 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
```

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	PaymentMethod	MonthlyCharges	TotalCharges
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85	29.8
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95	1889.
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85	108.1
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30	1840.7
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70	151.6

5 rows × 21 columns

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

In [14]:

```
# Creating a dummy variable for some of the categorical variables and dropping th
e 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)
```

In [15]:

```
telecom.head()
```

Out[15]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharges	TotalCharges
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85	29.8
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95	1889.5
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85	108.1
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30	1840.7
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70	151.6

5 rows × 29 columns

In [16]:

```
# Creating dummy variables for the remaining categorical variables and dropping t
he 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)
# 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	PaymentMethod	MonthlyCharges	TotalCharge
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29.85	29.8
1	5575-GNVDE	34	1	One year	0	Mailed check	56.95	1889.
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53.85	108.1
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42.30	1840.7
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70.70	151.6

5 rows × 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','Inter
netService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
        'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
```

In [19]:

```
#The varaible was imported as a string we need to convert it to float
telecom['TotalCharges'] = pd.to_numeric(telecom['TotalCharges'], errors = 'coerce'
)#Depreciated ->telecom['TotalCharges'].convert_objects(convert_numeric=True)
```

In [20]:

```
telecom.info()
```

```
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

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

	tenure	MonthlyCharges	SeniorCitizen	TotalCharges
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
```

```
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

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	SeniorCitizen	Partner	Dependents	C
0	1	0	1	29.85	29.85	0	1	0	
1	34	1	0	56.95	1889.50	0	0	0	
2	2	1	1	53.85	108.15	0	0	0	
3	45	0	0	42.30	1840.75	0	0	0	
4	2	1	1	70.70	151.65	0	0	0	

5 rows × 30 columns

In [29]:

```python
# 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=0.3, random_state=100)
```

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[['tenure','MonthlyCharges','TotalCharges']])

X_train.head()
```

Out[32]:

	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	SeniorCitizen	Partner	Dependen
879	0.019693	1	1	-0.338074	-0.276449	0	0	
5790	0.305384	0	1	-0.464443	-0.112702	0	1	
6498	-1.286319	1	1	0.581425	-0.974430	0	0	
880	-0.919003	1	1	1.505913	-0.550676	0	0	
2784	-1.163880	1	1	1.106854	-0.835971	0	0	

5 rows × 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

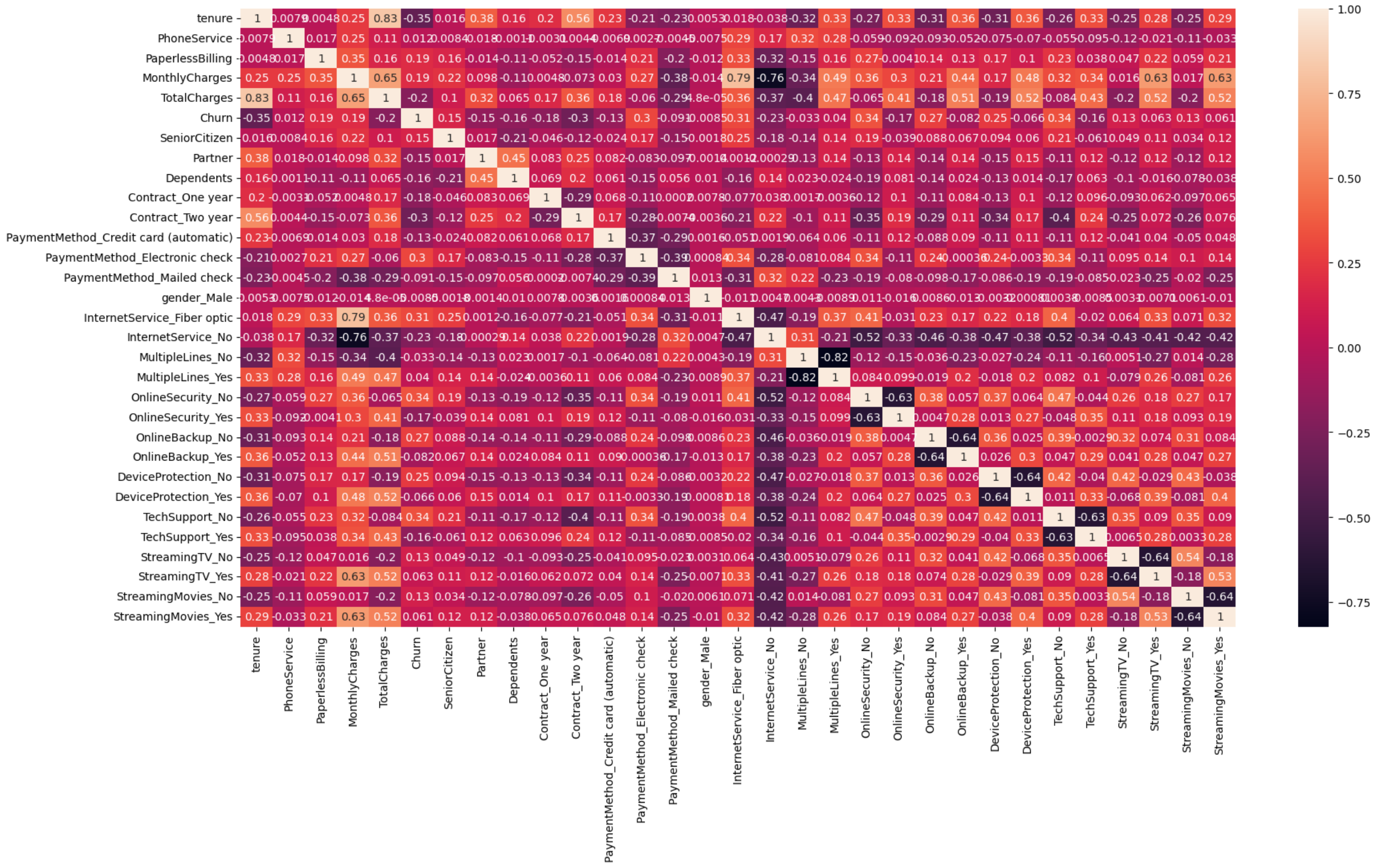
Step 6: Looking at Correlations

In [34]:

```
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
plt.style.use('default')
```

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]:

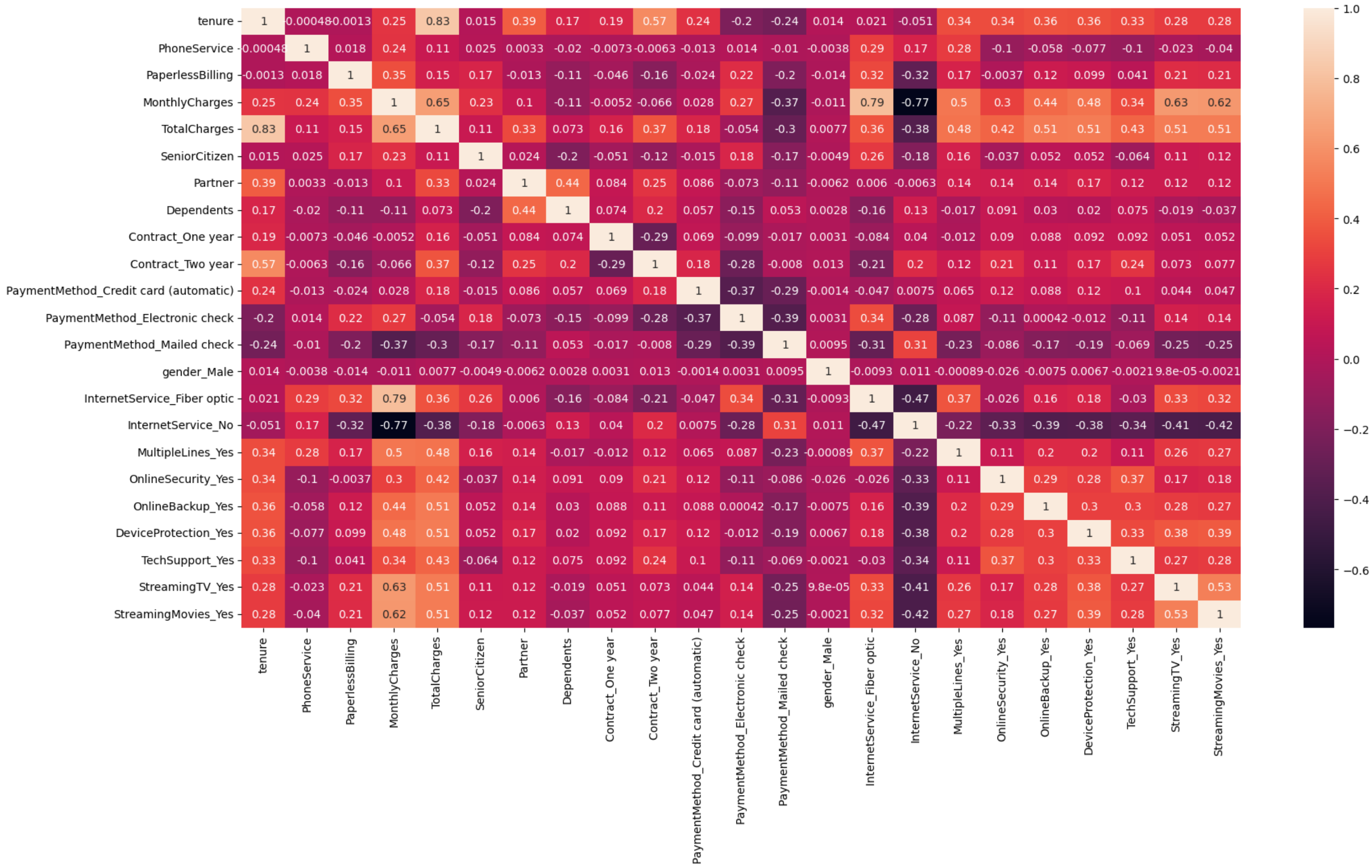
```
X_test = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','De
viceProtection_No','TechSupport_No',
                      'StreamingTV_No','StreamingMovies_No'], 1)
X_train = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','
DeviceProtection_No','TechSupport_No',
                        '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()
```



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 [64]:

```
from statsmodels import api as sm
```

In [65]:

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

Out[65]:

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, 15 Feb 2023	**Deviance:**	4009.4
Time:	01:40:14	**Pearson chi2:**	6.07e+03
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.2844
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

Step 8: Feature Selection Using RFE

In [47]:

```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [49]:

```
from sklearn.feature_selection import RFE
rfe = RFE(estimator=logreg, n_features_to_select=15)             # running RFE wit
13 variables as output
rfe = rfe.fit(X_train, y_train)
```

In [59]:

```
rfe.get_support()
```

Out[59]:

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

In [60]:

```
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[60]:

```
[('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)]
```

In [61]:

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

In [62]:

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

Out[62]:

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

Assessing the model with StatsModels

In [66]:

```
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[66]:

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, 15 Feb 2023	**Deviance:**	4022.2
Time:	01:40:22	**Pearson chi2:**	6.25e+03
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.2825
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

In [67]:

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

Out[67]:

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

```
6623    0.801788
4465    0.228194
5364    0.504575
dtype: float64
```

In [68]:

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

Out[68]:

```
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

In [69]:

```
y_train_pred_final = pd.DataFrame({'Churn':y_train.values, 'Churn_Prob':y_train_pr
ed})
y_train_pred_final['CustID'] = y_train.index
y_train_pred_final.head()
```

Out[69]:

	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

In [70]:

```
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()
```

Out[70]:

	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

In [71]:

```
from sklearn import metrics
```

In [72]:

```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.
predicted )
print(confusion)
```

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

In [53]:

```
# Predicted     not_churn    churn
# Actual
# not_churn        3270       365
# churn            579        708
```

In [73]:

```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicte
d))
```

```
0.8102397399431126
```

Checking VIFs

In [74]:

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

In [75]:

```
# 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
```

Out[75]:

	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

	Features	VIF
11	MultipleLines_Yes	2.35
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.

In [84]:

```
col = col.drop('MonthlyCharges', 1)
col
```

Out[84]:

```
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', 'TechSupport_Yes', 'StreamingTV_Yes',
       'StreamingMovies_Yes'],
      dtype='object')
```

In [85]:

```
# 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[85]:

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:**	-2023.1
Date:	Wed, 15 Feb 2023	**Deviance:**	4046.2
Time:	01:42:48	**Pearson chi2:**	5.80e+03
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.2790
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.5971	0.130	-12.296	0.000	-1.852	-1.343
tenure	-1.3286	0.180	-7.401	0.000	-1.681	-0.977

PaperlessBilling	0.3533	0.089	3.958	0.000	0.178	0.528
TotalCharges	0.4347	0.186	2.340	0.019	0.071	0.799
SeniorCitizen	0.4569	0.099	4.601	0.000	0.262	0.652
Contract_One year	-0.7289	0.127	-5.729	0.000	-0.978	-0.480
Contract_Two year	-1.3277	0.210	-6.322	0.000	-1.739	-0.916
PaymentMethod_Credit card (automatic)	-0.3870	0.112	-3.442	0.001	-0.607	-0.167
PaymentMethod_Mailed check	-0.3618	0.110	-3.274	0.001	-0.578	-0.145
InternetService_Fiber optic	0.6888	0.109	6.297	0.000	0.474	0.903
InternetService_No	-0.9555	0.156	-6.120	0.000	-1.262	-0.649
MultipleLines_Yes	0.1700	0.094	1.814	0.070	-0.014	0.354
TechSupport_Yes	-0.4371	0.101	-4.307	0.000	-0.636	-0.238
StreamingTV_Yes	0.2881	0.096	2.996	0.003	0.100	0.477
StreamingMovies_Yes	0.1944	0.096	2.031	0.042	0.007	0.382

In [86]:

```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [87]:

```
y_train_pred[:10]
```

Out[87]:

```
array([0.22790197, 0.22864388, 0.67489226, 0.61586836, 0.66226032,
       0.41819928, 0.28813321, 0.7951366 , 0.17433167, 0.51908788])
```

In [88]:

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

In [89]:

```
# 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()
```

Out[89]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.227902	879	0
1	0	0.228644	5790	0
2	1	0.674892	6498	1
3	1	0.615868	880	1
4	1	0.662260	2784	1

In [90]:

```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted))
```

```
0.8057700121901666
```

So overall the accuracy hasn't dropped much.

Let's check the VIFs again

In [91]:

```
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
```

Out[91]:

	Features	VIF
2	TotalCharges	7.46
0	tenure	6.90
5	Contract_Two year	3.07
8	InternetService_Fiber optic	2.96
13	StreamingMovies_Yes	2.62
12	StreamingTV_Yes	2.59
1	PaperlessBilling	2.55
9	InternetService_No	2.44
10	MultipleLines_Yes	2.27
11	TechSupport_Yes	1.95
4	Contract_One year	1.79
7	PaymentMethod_Mailed check	1.63
6	PaymentMethod_Credit card (automatic)	1.42
3	SeniorCitizen	1.31

In [92]:

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

Out[92]:

```
Index(['tenure', 'PaperlessBilling', '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')
```

In [93]:

```
# 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[93]:

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:**	-2025.9
Date:	Wed, 15 Feb 2023	**Deviance:**	4051.9
Time:	01:43:03	**Pearson chi2:**	5.25e+03
No. Iterations:	7	**Pseudo R-squ. (CS):**	0.2782
Covariance Type:	nonrobust		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-1.6577	0.127	-13.094	0.000	-1.906	-1.410
tenure	-0.9426	0.065	-14.480	0.000	-1.070	-0.815
PaperlessBilling	0.3455	0.089	3.877	0.000	0.171	0.520
SeniorCitizen	0.4597	0.100	4.613	0.000	0.264	0.655
Contract_One year	-0.7218	0.127	-5.702	0.000	-0.970	-0.474
Contract_Two year	-1.2987	0.208	-6.237	0.000	-1.707	-0.891
PaymentMethod_Credit card (automatic)	-0.3874	0.113	-3.442	0.001	-0.608	-0.167
PaymentMethod_Mailed check	-0.3307	0.110	-3.020	0.003	-0.545	-0.116
InternetService_Fiber optic	0.8052	0.097	8.272	0.000	0.614	0.996
InternetService_No	-0.9726	0.155	-6.261	0.000	-1.277	-0.668
MultipleLines_Yes	0.2097	0.092	2.279	0.023	0.029	0.390
TechSupport_Yes	-0.4046	0.101	-4.019	0.000	-0.602	-0.207
StreamingTV_Yes	0.3390	0.094	3.619	0.000	0.155	0.523
StreamingMovies_Yes	0.2428	0.093	2.598	0.009	0.060	0.426

In [94]:

```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [95]:

```
y_train_pred[:10]
```

Out[95]:

```
array([0.24581699, 0.26536078, 0.66940978, 0.63097033, 0.68291606,
       0.39952622, 0.27582791, 0.79816753, 0.19878625, 0.52911878])
```

In [96]:

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

In [97]:

```
# 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()
```

Out[97]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.245817	879	0
1	0	0.265361	5790	0
2	1	0.669410	6498	1
3	1	0.630970	880	1
4	1	0.682916	2784	1

In [98]:

```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicte
d))
```

```
0.8061763510767981
```

The accuracy is still practically the same.

Let's now check the VIFs again

In [99]:

```
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
```

Out[99]:

	Features	VIF
4	Contract_Two year	2.98
7	InternetService_Fiber optic	2.67
12	StreamingMovies_Yes	2.54
11	StreamingTV_Yes	2.51
1	PaperlessBilling	2.45
9	MultipleLines_Yes	2.24
0	tenure	2.04
8	InternetService_No	2.03
10	TechSupport_Yes	1.92
3	Contract_One year	1.78

	Features	VIF
6	PaymentMethod_Maile[illegible]	[illegible]
5	PaymentMethod_Credit card (automatic)	1.42
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 [100]:

```
# 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
```

Out[100]:

```
array([[3278,  357],
       [ 597,  690]], dtype=int64)
```

In [74]:

```
# Actual/Predicted     not_churn    churn
        # not_churn        3269      366
        # churn            595       692
```

In [101]:

```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[101]:

```
0.8061763510767981
```

Metrics beyond simply accuracy

In [102]:

```
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 [104]:

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

Out[104]:

```
0.5361305361305362
```

In [105]:

```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[105]:

```
0.9017881705639614
```

In [106]:

In [106]:

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

0.09821182943603851

In [107]:

```
# positive predictive value
print (TP / float(TP+FP))
```

0.6590257879656161

In [108]:

```
# Negative predictive value
print (TN / float(TN+ FN))
```

0.8459354838709677

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 [109]:

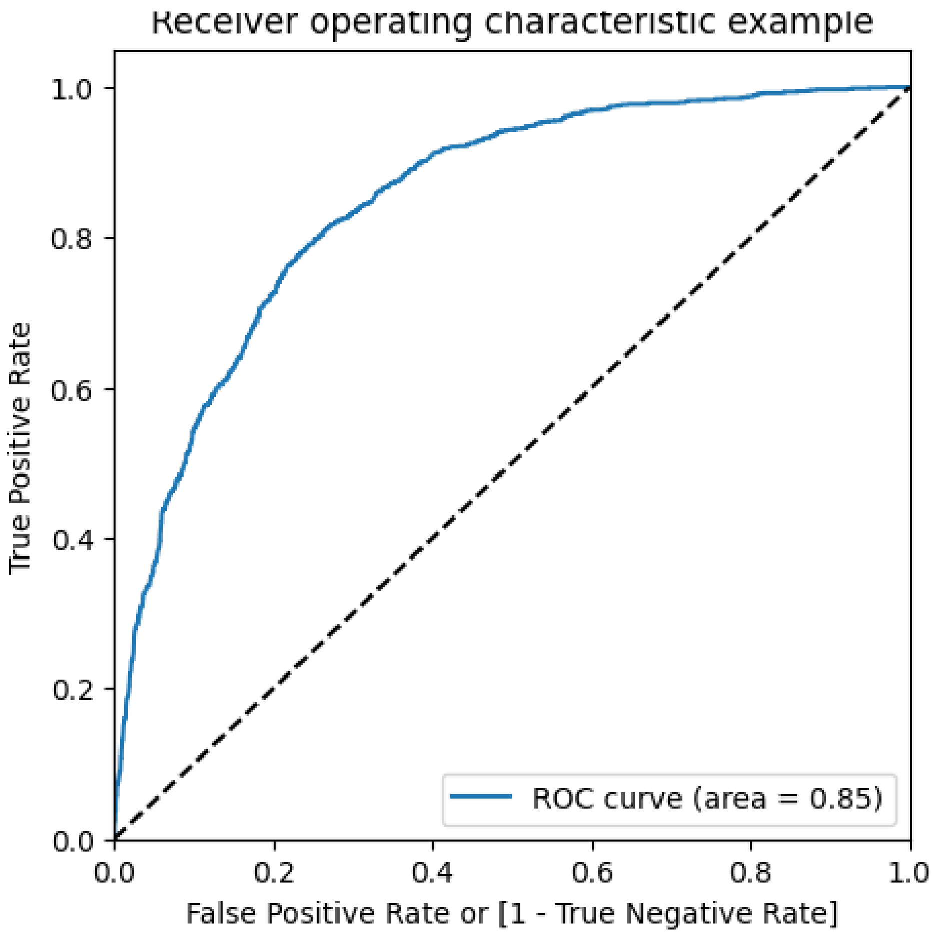
```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %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 [110]:

```
fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Churn,
y_train_pred_final.Churn_Prob, drop_intermediate = False )
```

In [111]:

```
draw_roc(y_train_pred_final.Churn, y_train_pred_final.Churn_Prob)
```



Step 10: Finding Optimal Cutoff Point

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

In [112]:

```
# 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 e
lse 0)
y_train_pred_final.head()
```

Out[112]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.245817	879	0	1	1	1	0	0	0	0	0	0	0
1	0	0.265361	5790	0	1	1	1	0	0	0	0	0	0	0
2	1	0.669410	6498	1	1	1	1	1	1	1	1	0	0	0
3	1	0.630970	880	1	1	1	1	1	1	1	1	0	0	0
4	1	0.682916	2784	1	1	1	1	1	1	1	1	0	0	0

In [113]:

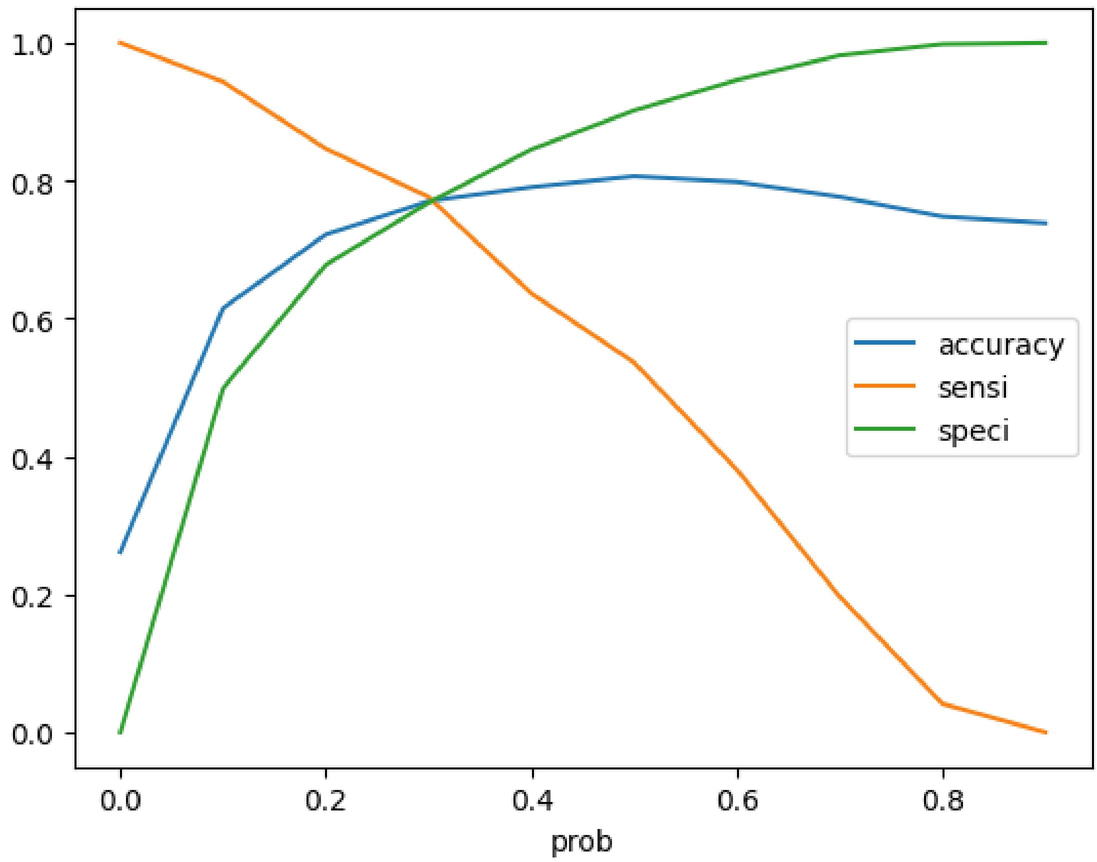
```
# Now let's calculate accuracy sensitivity and specificity for various
probability cutoffs.
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix

# 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  0.614994  0.943279  0.498762
0.2   0.2  0.721861  0.846154  0.677854
0.3   0.3  0.770012  0.776224  0.767813
0.4   0.4  0.790532  0.636364  0.845117
0.5   0.5  0.806176  0.536131  0.901788
0.6   0.6  0.798050  0.380730  0.945805
0.7   0.7  0.776310  0.196581  0.981568
0.8   0.8  0.747867  0.041181  0.998074
0.9   0.9  0.738521  0.000000  1.000000
```

In [114]:

```
# 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 [115]:

```
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()
```

Out[115]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	final_predicted
0	0	0.245817	879	0	1	1	1	0	0	0	0	0	0	0	0
1	0	0.265361	5790	0	1	1	1	0	0	0	0	0	0	0	0
2	1	0.669410	6498	1	1	1	1	1	1	1	1	0	0	0	1
3	1	0.630970	880	1	1	1	1	1	1	1	1	0	0	0	1
4	1	0.682916	2784	1	1	1	1	1	1	1	1	0	0	0	1

In [116]:

```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn,
y_train_pred_final.final_predicted)
```

Out[116]:

```
0.7700121901665989
```

In [117]:

```
confusion2 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final
.final_predicted )
confusion2
```

Out[117]:

```
array([[2791,  844],
       [ 288,  999]], dtype=int64)
```

In [118]:

```
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 [119]:

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

Out[119]:

```
0.7762237762237763
```

In [120]:

```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[120]:

```
0.7678129298486933
```

In [121]:

```
# Calculate false postive rate - predicting churn when customer does not have chur
ned
print(FP/ float(TN+FP))
```

```
0.23218707015130674
```

In [122]:

```
# Positive predictive value
print (TP / float(TP+FP))
```

```
0.5420510037981552
```

In [123]:

```
# Negative predictive value
print (TN / float(TN+ FN))
```

```
0.9064631373822669
```

Precision and Recall

In [97]:

```
#Looking at the confusion matrix again
```

In [124]:

```
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.
predicted )
confusion
```

Out[124]:

```
array([[3278,  357],
       [ 597,  690]], dtype=int64)
```

Precision

TP / TP + FP

In [125]:

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

Out[125]:

```
0.6590257879656161
```

Recall

TP / TP + FN

In [126]:

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

Out[126]:

```
0.5361305361305362
```

Using sklearn utilities for the same

In [127]:

```python
from sklearn.metrics import precision_score, recall_score
```

In [128]:

```python
?precision_score
```

```
Signature:
precision_score(
    y_true,
    y_pred,
    *,
    labels=None,
    pos_label=1,
    average='binary',
    sample_weight=None,
    zero_division='warn',
)
Docstring:
Compute the precision.

The precision is the ratio ``tp / (tp + fp)`` where ``tp`` is the number of
true positives and ``fp`` the number of false positives. The precision is
intuitively the ability of the classifier not to label as positive a sample
that is negative.

The best value is 1 and the worst value is 0.

Read more in the :ref:`User Guide <precision_recall_f_measure_metrics>`.

Parameters
----------
y_true : 1d array-like, or label indicator array / sparse matrix
    Ground truth (correct) target values.

y_pred : 1d array-like, or label indicator array / sparse matrix
    Estimated targets as returned by a classifier.

labels : array-like, default=None
    The set of labels to include when ``average != 'binary'``, and their
    order if ``average is None``. Labels present in the data can be
    excluded, for example to calculate a multiclass average ignoring a
    majority negative class, while labels not present in the data will
    result in 0 components in a macro average. For multilabel targets,
    labels are column indices. By default, all labels in ``y_true`` and
    ``y_pred`` are used in sorted order.

    .. versionchanged:: 0.17
       Parameter `labels` improved for multiclass problem.

pos_label : str or int, default=1
    The class to report if ``average='binary'`` and the data is binary.
    If the data are multiclass or multilabel, this will be ignored;
    setting ``labels=[pos_label]`` and ``average != 'binary'`` will report
    scores for that label only.

average : {'micro', 'macro', 'samples', 'weighted', 'binary'} or None,
default='binary'
    This parameter is required for multiclass/multilabel targets.
    If ``None``, the scores for each class are returned. Otherwise, this
```

```
    determines the type of averaging performed on the data:

    ``'binary'``:
        Only report results for the class specified by ``pos_label``.
        This is applicable only if targets (``y_{true,pred}``) are binary.
    ``'micro'``:
        Calculate metrics globally by counting the total true positives,
        false negatives and false positives.
    ``'macro'``:
        Calculate metrics for each label, and find their unweighted
        mean.  This does not take label imbalance into account.
    ``'weighted'``:
        Calculate metrics for each label, and find their average weighted
        by support (the number of true instances for each label). This
        alters 'macro' to account for label imbalance; it can result in an
        F-score that is not between precision and recall.
    ``'samples'``:
        Calculate metrics for each instance, and find their average (only
        meaningful for multilabel classification where this differs from
        :func:`accuracy_score`).

sample_weight : array-like of shape (n_samples,), default=None
    Sample weights.

zero_division : "warn", 0 or 1, default="warn"
    Sets the value to return when there is a zero division. If set to
    "warn", this acts as 0, but warnings are also raised.

Returns
-------
precision : float (if average is not None) or array of float of shape
(n_unique_labels,)
    Precision of the positive class in binary classification or weighted
    average of the precision of each class for the multiclass task.

See Also
--------
precision_recall_fscore_support : Compute precision, recall, F-measure and
    support for each class.
recall_score :  Compute the ratio ``tp / (tp + fn)`` where ``tp`` is the
    number of true positives and ``fn`` the number of false negatives.
PrecisionRecallDisplay.from_estimator : Plot precision-recall curve given
    an estimator and some data.
PrecisionRecallDisplay.from_predictions : Plot precision-recall curve given
    binary class predictions.
multilabel_confusion_matrix : Compute a confusion matrix for each class or
    sample.

Notes
-----
When ``true positive + false positive == 0``, precision returns 0 and
raises ``UndefinedMetricWarning``. This behavior can be
modified with ``zero_division``.

Examples
--------
>>> from sklearn.metrics import precision_score
>>> y_true = [0, 1, 2, 0, 1, 2]
>>> y_pred = [0, 2, 1, 0, 0, 1]
>>> precision_score(y_true, y_pred, average='macro')
0.22...
>>> precision_score(y_true, y_pred, average='micro')
0.33...
>>> precision_score(y_true, y_pred, average='weighted')
```

```
0.22...
>>> precision_score(y_true, y_pred, average=None)
array([0.66..., 0.        , 0.        ])
>>> y_pred = [0, 0, 0, 0, 0, 0]
>>> precision_score(y_true, y_pred, average=None)
array([0.33..., 0.        , 0.        ])
>>> precision_score(y_true, y_pred, average=None, zero_division=1)
array([0.33..., 1.        , 1.        ])
>>> # multilabel classification
>>> y_true = [[0, 0, 0], [1, 1, 1], [0, 1, 1]]
>>> y_pred = [[0, 0, 0], [1, 1, 1], [1, 1, 0]]
>>> precision_score(y_true, y_pred, average=None)
array([0.5, 1. , 1. ])
File:
c:\users\dell\appdata\local\packages\pythonsoftwarefoundation.python.3.9_qbz5n2kfra
\localcache\local-packages\python39\site-
packages\sklearn\metrics\_classification.py
Type:      function
```

In [129]:

```
precision_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[129]:

```
0.6590257879656161
```

In [130]:

```
recall_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[130]:

```
0.5361305361305362
```

Precision and recall tradeoff

In [131]:

```
from sklearn.metrics import precision_recall_curve
```

In [132]:

```
y_train_pred_final.Churn, y_train_pred_final.predicted
```

Out[132]:

```
(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
```

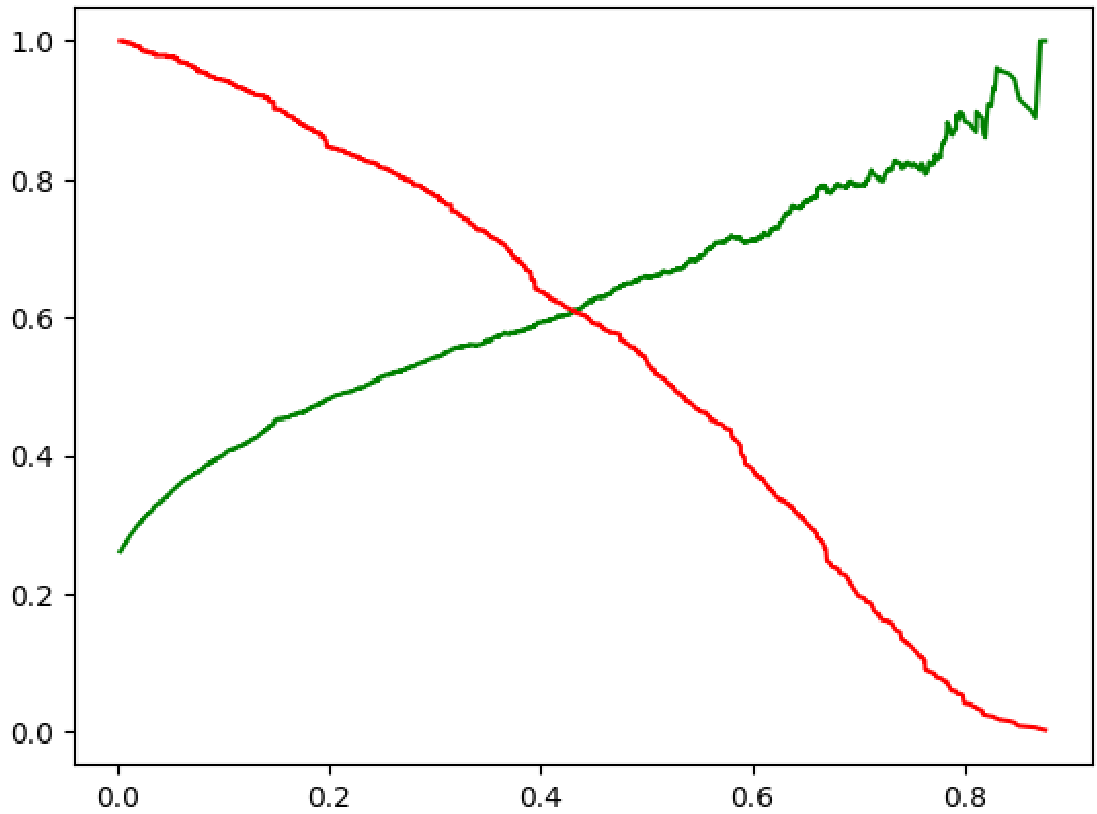
```
        ..
 4917     0
 4918     0
 4919     0
 4920     0
 4921     0
 Name: predicted, Length: 4922, dtype: int64)
```

In [133]:

```
p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn,
y_train_pred_final.Churn_Prob)
```

In [134]:

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



Step 11: Making predictions on the test set

In [135]:

```
X_test[['tenure','MonthlyCharges','TotalCharges']] = scaler.transform(X_test[['ten
ure','MonthlyCharges','TotalCharges']])
```

In [136]:

```
X_test = X_test[col]
X_test.head()
```

Out[136]:

	tenure	PaperlessBilling	SeniorCitizen	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMe
942	-	1	0	0	0	1	

	tenure	PaperlessBilling	SeniorCitizen	Contract_One year	Contract_Two year	PaymentMethod_Credit card (automatic)	PaymentMe
	0.347623						
3730	0.999203	1	0	0	0		
1761	1.040015	1	0	0	1	1	
2283	-1.286319	1	0	0	0	0	
1872	0.346196	0	0	0	1	0	

In [137]:

```
X_test_sm = sm.add_constant(X_test)
```

Making predictions on the test set

In [138]:

```
y_test_pred = res.predict(X_test_sm)
```

In [139]:

```
y_test_pred[:10]
```

Out[139]:

```
942     0.419725
3730    0.260232
1761    0.008650
2283    0.592626
1872    0.013989
1970    0.692893
2532    0.285289
1616    0.008994
2485    0.602307
5914    0.145153
dtype: float64
```

In [140]:

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

In [141]:

```
# Let's see the head
y_pred_1.head()
```

Out[141]:

	0
942	0.419725
3730	0.260232
1761	0.008650
2283	0.592626
1872	0.013989

In [142]:

```
# Converting y_test to dataframe
```

```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
```

In [143]:

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

In [144]:

```
# 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 [145]:

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

In [146]:

```
y_pred_final.head()
```

Out[146]:

	Churn	CustID	0
0	0	942	0.419725
1	1	3730	0.260232
2	0	1761	0.008650
3	1	2283	0.592626
4	0	1872	0.013989

In [147]:

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

In [149]:

```
# Rearranging the columns
y_pred_final = y_pred_final[['CustID','Churn','Churn_Prob']]
```

In [150]:

```
# Let's see the head of y_pred_final
y_pred_final.head()
```

Out[150]:

	CustID	Churn	Churn_Prob
0	942	0	0.419725
1	3730	1	0.260232
2	1761	0	0.008650
3	2283	1	0.592626
4	1872	0	0.013989

In [151]:

In [151]:

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

In [152]:

```
y_pred_final.head()
```

Out[152]:

	CustID	Churn	Churn_Prob	final_predicted
0	942	0	0.419725	0
1	3730	1	0.260232	0
2	1761	0	0.008650	0
3	2283	1	0.592626	1
4	1872	0	0.013989	0

In [153]:

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

Out[153]:

```
0.7838862559241706
```

In [154]:

```
confusion2 = metrics.confusion_matrix(y_pred_final.Churn, y_pred_final.final_predi
cted )
confusion2
```

Out[154]:

```
array([[1286,  242],
       [ 214,  368]], dtype=int64)
```

In [155]:

```
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 [156]:

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

Out[156]:

```
0.6323024054982818
```

In [157]:

```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[157]:

```
0.8416230366492147
```

In [ ]:

```
```