

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]:

```
1 # Suppressing Warnings
2 import warnings
3 warnings.filterwarnings('ignore')
```

In [2]:

```
1 # Importing Pandas and NumPy
2 import pandas as pd, numpy as np
```

In [3]:

```
1 # Importing all datasets
2 churn_data = pd.read_csv("churn_data.csv")
3 churn_data.head()
```

Out[3]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharg
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70

In [4]:

```
1 customer_data = pd.read_csv("customer_data.csv")
2 customer_data.head()
```

Out[4]:

	customerID	gender	SeniorCitizen	Partner	Dependents
0	7590-VHVEG	Female	0	Yes	No
1	5575-GNVDE	Male	0	No	No
2	3668-QPYBK	Male	0	No	No
3	7795-CFOCW	Male	0	No	No
4	9237-HQITU	Female	0	No	No

In [5]:

```
1 internet_data = pd.read_csv("internet_data.csv")
2 internet_data.head()
```

Out[5]:

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

Combining all data files into one consolidated dataframe

In [6]:

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

In [7]:

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

Step 2: Inspecting the Dataframe

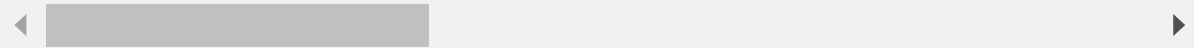
In [8]:

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

Out[8]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharg
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70

5 rows × 21 columns



In [9]:

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

Out[9]:

(7043, 21)

In [10]:

```
1 # Let's Look at the statistical aspects of the dataframe
2 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]:

```
1 # Let's see the type of each column
2 telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 21 columns):
customerID      7043 non-null object
tenure          7043 non-null int64
PhoneService    7043 non-null object
Contract        7043 non-null object
PaperlessBilling 7043 non-null object
PaymentMethod   7043 non-null object
MonthlyCharges  7043 non-null float64
TotalCharges    7043 non-null object
Churn           7043 non-null object
gender          7043 non-null object
SeniorCitizen   7043 non-null int64
Partner         7043 non-null object
Dependents      7043 non-null object
MultipleLines   7043 non-null object
InternetService 7043 non-null object
OnlineSecurity  7043 non-null object
OnlineBackup    7043 non-null object
DeviceProtection 7043 non-null object
TechSupport     7043 non-null object
StreamingTV     7043 non-null object
StreamingMovies 7043 non-null object
dtypes: float64(1), int64(2), object(18)
memory usage: 1.2+ MB
```

Step 3: Data Preparation

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

In [12]:

```
1 # List of variables to map
2
3 varlist = ['PhoneService', 'PaperlessBilling', 'Churn', 'Partner', 'Dependents']
4
5 # Defining the map function
6 def binary_map(x):
7     return x.map({'Yes': 1, 'No': 0})
8
9 # Applying the function to the housing list
10 telecom[varlist] = telecom[varlist].apply(binary_map)
```

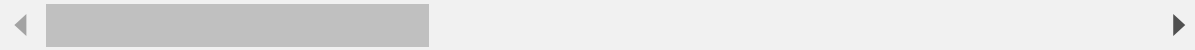
In [13]:

```
1 telecom.head()
```

Out[13]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharg
0	7590-VHVEG	1		Month-to-month	1	Electronic check	29
1	5575-GNVDE	34		One year	0	Mailed check	56
2	3668-QPYBK	2		Month-to-month	1	Mailed check	53
3	7795-CFOCW	45		One year	0	Bank transfer (automatic)	42
4	9237-HQITU	2		Month-to-month	1	Electronic check	70

5 rows × 21 columns



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

In [14]:

```
1 # Creating a dummy variable for some of the categorical variables and dropping the first
2 dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetService
3
4 # Adding the results to the master dataframe
5 telecom = pd.concat([telecom, dummy1], axis=1)
```

In [15]:

```
1 telecom.head()
```

Out[15]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharg
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29
1	5575-GNVDE	34	1	One year	0	Mailed check	56
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70

5 rows × 29 columns



In [16]:

```
1  # Creating dummy variables for the remaining categorical variables and dropping the Lev
2
3  # Creating dummy variables for the variable 'MultipleLines'
4  ml = pd.get_dummies(telecom['MultipleLines'], prefix='MultipleLines')
5  # Dropping MultipleLines_No phone service column
6  ml1 = ml.drop(['MultipleLines_No phone service'], 1)
7  #Adding the results to the master dataframe
8  telecom = pd.concat([telecom,ml1], axis=1)
9
10 # Creating dummy variables for the variable 'OnlineSecurity'.
11 os = pd.get_dummies(telecom['OnlineSecurity'], prefix='OnlineSecurity')
12 os1 = os.drop(['OnlineSecurity_No internet service'], 1)
13 # Adding the results to the master dataframe
14 telecom = pd.concat([telecom,os1], axis=1)
15
16 # Creating dummy variables for the variable 'OnlineBackup'.
17 ob = pd.get_dummies(telecom['OnlineBackup'], prefix='OnlineBackup')
18 ob1 = ob.drop(['OnlineBackup_No internet service'], 1)
19 # Adding the results to the master dataframe
20 telecom = pd.concat([telecom,ob1], axis=1)
21
22 # Creating dummy variables for the variable 'DeviceProtection'.
23 dp = pd.get_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')
24 dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
25 # Adding the results to the master dataframe
26 telecom = pd.concat([telecom,dp1], axis=1)
27
28 # Creating dummy variables for the variable 'TechSupport'.
29 ts = pd.get_dummies(telecom['TechSupport'], prefix='TechSupport')
30 ts1 = ts.drop(['TechSupport_No internet service'], 1)
31 # Adding the results to the master dataframe
32 telecom = pd.concat([telecom,ts1], axis=1)
33
34 # Creating dummy variables for the variable 'StreamingTV'.
35 st =pd.get_dummies(telecom['StreamingTV'], prefix='StreamingTV')
36 st1 = st.drop(['StreamingTV_No internet service'], 1)
37 # Adding the results to the master dataframe
38 telecom = pd.concat([telecom,st1], axis=1)
39
40 # Creating dummy variables for the variable 'StreamingMovies'.
41 sm = pd.get_dummies(telecom['StreamingMovies'], prefix='StreamingMovies')
42 sm1 = sm.drop(['StreamingMovies_No internet service'], 1)
43 # Adding the results to the master dataframe
44 telecom = pd.concat([telecom,sm1], axis=1)
```

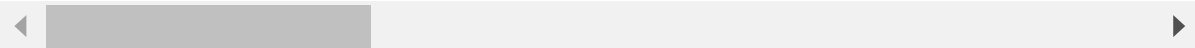
In [17]:

```
1 telecom.head()
```

Out[17]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	MonthlyCharg
0	7590-VHVEG	1	0	Month-to-month	1	Electronic check	29
1	5575-GNVDE	34	1	One year	0	Mailed check	56
2	3668-QPYBK	2	1	Month-to-month	1	Mailed check	53
3	7795-CFOCW	45	0	One year	0	Bank transfer (automatic)	42
4	9237-HQITU	2	1	Month-to-month	1	Electronic check	70

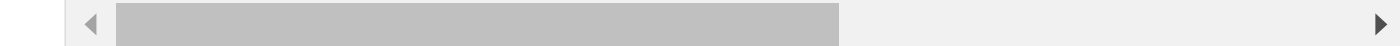
5 rows × 43 columns



Dropping the repeated variables

In [18]:

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



In [19]:

```
1 #The variable was imported as a string we need to convert it to float
2 telecom['TotalCharges'] = telecom['TotalCharges'].convert_objects(convert_numeric=True)
```


In [20]:

```
1 telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 32 columns):
customerID                7043 non-null object
tenure                    7043 non-null int64
PhoneService              7043 non-null int64
PaperlessBilling          7043 non-null int64
MonthlyCharges            7043 non-null float64
TotalCharges              7032 non-null float64
Churn                     7043 non-null int64
SeniorCitizen             7043 non-null int64
Partner                   7043 non-null int64
Dependents                7043 non-null int64
Contract_One year        7043 non-null uint8
Contract_Two year        7043 non-null uint8
PaymentMethod_Credit card (automatic) 7043 non-null uint8
PaymentMethod_Electronic check 7043 non-null uint8
PaymentMethod_Mailed check 7043 non-null uint8
gender_Male               7043 non-null uint8
InternetService_Fiber optic 7043 non-null uint8
InternetService_No        7043 non-null uint8
MultipleLines_No          7043 non-null uint8
MultipleLines_Yes         7043 non-null uint8
OnlineSecurity_No         7043 non-null uint8
OnlineSecurity_Yes        7043 non-null uint8
OnlineBackup_No           7043 non-null uint8
OnlineBackup_Yes          7043 non-null uint8
DeviceProtection_No       7043 non-null uint8
DeviceProtection_Yes      7043 non-null uint8
TechSupport_No            7043 non-null uint8
TechSupport_Yes           7043 non-null uint8
StreamingTV_No            7043 non-null uint8
StreamingTV_Yes           7043 non-null uint8
StreamingMovies_No        7043 non-null uint8
StreamingMovies_Yes       7043 non-null uint8
dtypes: float64(2), int64(7), object(1), uint8(22)
memory usage: 756.6+ KB
```

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

Checking for Outliers

In [21]:

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

In [22]:

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

Out[22]:

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

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

Checking for Missing Values and Inputing Them

In [23]:

```
1 # Adding up the missing values (column-wise)
2 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]:

```
1 # Checking the percentage of missing values
2 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]:

```
1 # Removing NaN TotalCharges rows
2 telecom = telecom[~np.isnan(telecom['TotalCharges'])]
```

In [26]:

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

Out[26]:

customerID	0.0
tenure	0.0
PhoneService	0.0
PaperlessBilling	0.0
MonthlyCharges	0.0
TotalCharges	0.0
Churn	0.0
SeniorCitizen	0.0
Partner	0.0
Dependents	0.0
Contract_One year	0.0
Contract_Two year	0.0
PaymentMethod_Credit card (automatic)	0.0
PaymentMethod_Electronic check	0.0
PaymentMethod_Mailed check	0.0
gender_Male	0.0
InternetService_Fiber optic	0.0
InternetService_No	0.0
MultipleLines_No	0.0
MultipleLines_Yes	0.0
OnlineSecurity_No	0.0
OnlineSecurity_Yes	0.0
OnlineBackup_No	0.0
OnlineBackup_Yes	0.0
DeviceProtection_No	0.0
DeviceProtection_Yes	0.0
TechSupport_No	0.0
TechSupport_Yes	0.0
StreamingTV_No	0.0
StreamingTV_Yes	0.0
StreamingMovies_No	0.0
StreamingMovies_Yes	0.0
dtype: float64	

Now we don't have any missing values

Step 4: Test-Train Split

In [27]:

```
1 from sklearn.model_selection import train_test_split
```

In [28]:

```
1 # Putting feature variable to X
2 X = telecom.drop(['Churn', 'customerID'], axis=1)
3
4 X.head()
```

Out[28]:

	tenure	PhoneService	PaperlessBilling	MonthlyCharges	TotalCharges	SeniorCitizen	Partner
0	1	0	1	29.85	29.85	0	1
1	34	1	0	56.95	1889.50	0	0
2	2	1	1	53.85	108.15	0	0
3	45	0	0	42.30	1840.75	0	0
4	2	1	1	70.70	151.65	0	0

5 rows × 30 columns



In [29]:

```
1 # Putting response variable to y
2 y = telecom['Churn']
3
4 y.head()
```

Out[29]:

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

Name: Churn, dtype: int64

In [30]:

```
1 # Splitting the data into train and test
2 X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3)
```

Step 5: Feature Scaling

In [31]:

```
1 from sklearn.preprocessing import StandardScaler
```

In [32]:

```

1 scaler = StandardScaler()
2
3 X_train[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.fit_transform(X_train[['te
4
5 X_train.head()

```

Out[32]:

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

5 rows × 30 columns

In [33]:

```

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

```

Out[33]:

26.578498293515356

We have almost 27% churn rate

Step 6: Looking at Correlations

In [34]:

```

1 # Importing matplotlib and seaborn
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 %matplotlib inline

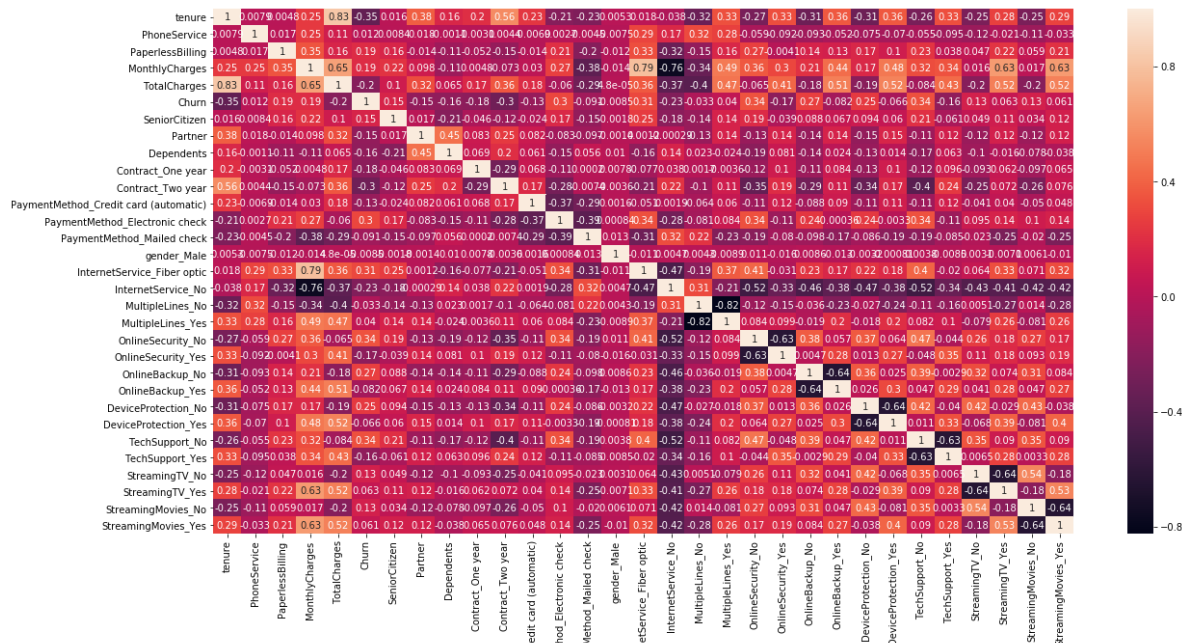
```

In [35]:

```

1 # Let's see the correlation matrix
2 plt.figure(figsize = (20,10)) # Size of the figure
3 sns.heatmap(telecom.corr(),annot = True)
4 plt.show()

```



Dropping highly correlated dummy variables

In [36]:

```

1 X_test = X_test.drop(['MultipleLines_No', 'OnlineSecurity_No', 'OnlineBackup_No', 'DeviceProtection_No',
2                       'StreamingTV_No', 'StreamingMovies_No'], 1)
3 X_train = X_train.drop(['MultipleLines_No', 'OnlineSecurity_No', 'OnlineBackup_No', 'DeviceProtection_No',
4                          'StreamingTV_No', 'StreamingMovies_No'], 1)

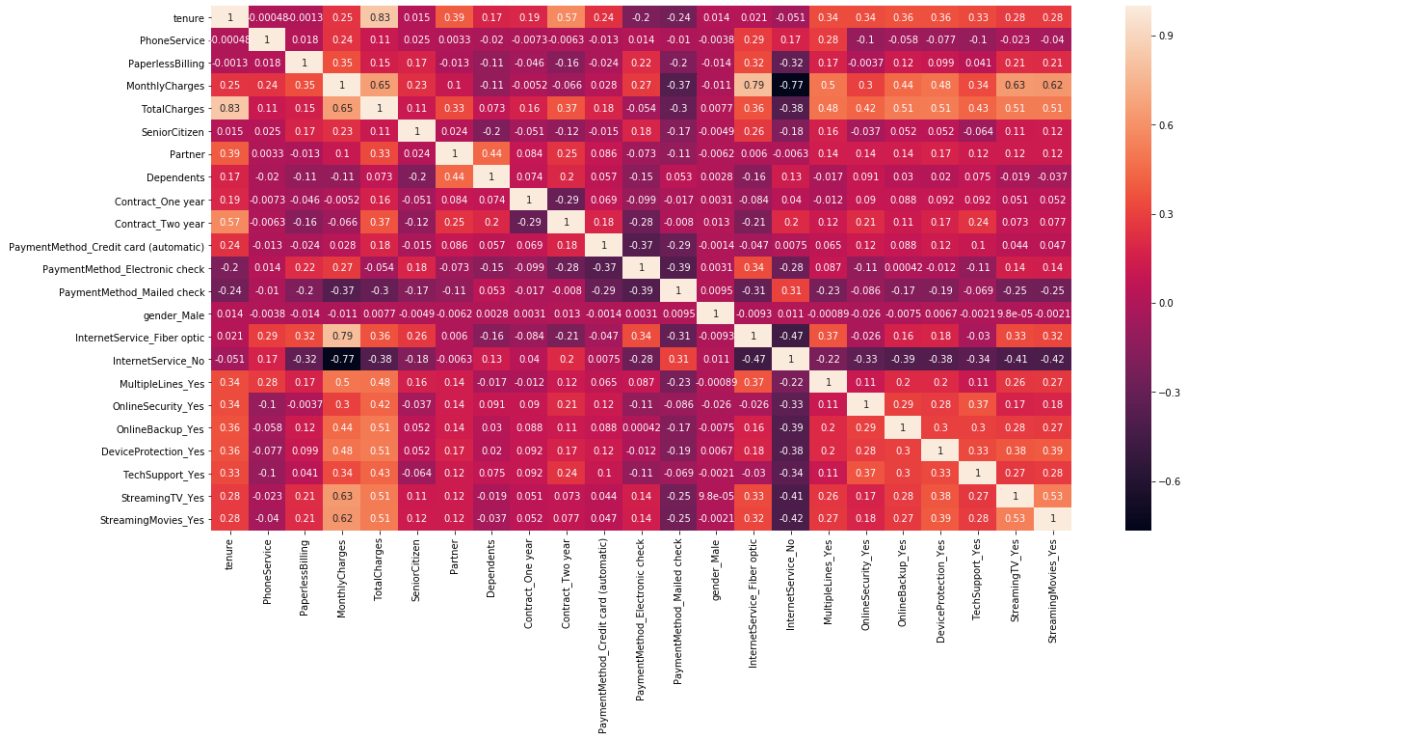
```

Checking the Correlation Matrix

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

In [37]:

```
1 plt.figure(figsize = (20,10))
2 sns.heatmap(X_train.corr(),annot = True)
3 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 [38]:

```
1 import statsmodels.api as sm
```

In [39]:

```

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

```

Out[39]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	No. Observations:	4922
Model:	GLM	Df Residuals:	4898
Model Family:	Binomial	Df Model:	23
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-2004.7
Date:	Thu, 29 Nov 2018	Deviance:	4009.4
Time:	11:23:01	Pearson chi2:	6.07e+03
No. Iterations:	7	Covariance Type:	nonrobust

	coef	std err	z	P> z	[0.025	0.975]
const	-3.9382	1.546	-2.547	0.011	-6.969	-0.908

Step 8: Feature Selection Using RFE

In [40]:

```

1 from sklearn.linear_model import LogisticRegression
2 logreg = LogisticRegression()

```

In [41]:

```

1 from sklearn.feature_selection import RFE
2 rfe = RFE(logreg, 15) # running RFE with 13 variables as output
3 rfe = rfe.fit(X_train, y_train)

```

In [42]:

```
1 rfe.support_
```

Out[42]:

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

In [43]:

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

Out[43]:

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

In [44]:

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

In [45]:

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

Out[45]:

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

Assessing the model with StatsModels

In [46]:

```

1 X_train_sm = sm.add_constant(X_train[col])
2 logm2 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
3 res = logm2.fit()
4 res.summary()

```

Out[46]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	No. Observations:	4922
Model:	GLM	Df Residuals:	4906
Model Family:	Binomial	Df Model:	15
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-2011.8
Date:	Thu, 29 Nov 2018	Deviance:	4023.5
Time:	11:23:04	Pearson chi2:	6.22e+03
No. Iterations:	7	Covariance Type:	nonrobust

	coef	std err	z	P> z	[0.025	0.975]
const	-1.0343	0.171	-6.053	0.000	-1.369	-0.699
tenure	-1.5386	0.184	-8.381	0.000	-1.898	-1.179
PhoneService	-0.5231	0.161	-3.256	0.001	-0.838	-0.208
PaperlessBilling	0.3397	0.090	3.789	0.000	0.164	0.515
TotalCharges	0.7116	0.188	3.794	0.000	0.344	1.079
SeniorCitizen	0.4294	0.100	4.312	0.000	0.234	0.625
Contract_One year	-0.6813	0.128	-5.334	0.000	-0.932	-0.431
Contract_Two year	-1.2680	0.211	-6.011	0.000	-1.681	-0.855
PaymentMethod_Credit card (automatic)	-0.3775	0.113	-3.352	0.001	-0.598	-0.157
PaymentMethod_Mailed check	-0.3760	0.111	-3.389	0.001	-0.594	-0.159
InternetService_Fiber optic	0.7421	0.117	6.317	0.000	0.512	0.972
InternetService_No	-0.9385	0.166	-5.650	0.000	-1.264	-0.613
MultipleLines_Yes	0.2086	0.096	2.181	0.029	0.021	0.396
OnlineSecurity_Yes	-0.4049	0.102	-3.968	0.000	-0.605	-0.205
TechSupport_Yes	-0.3967	0.102	-3.902	0.000	-0.596	-0.197
StreamingTV_Yes	0.2747	0.094	2.911	0.004	0.090	0.460

In [47]:

```

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

```

Out[47]:

```

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

```

In [48]:

```

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

```

Out[48]:

```

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

```

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

In [49]:

```

1 y_train_pred_final = pd.DataFrame({'Churn':y_train.values, 'Churn_Prob':y_train_pred})
2 y_train_pred_final['CustID'] = y_train.index
3 y_train_pred_final.head()

```

Out[49]:

	Churn	Churn_Prob	CustID
0	0	0.225111	879
1	0	0.274893	5790
2	1	0.692126	6498
3	1	0.504909	880
4	1	0.645261	2784

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

In [50]:

```

1 y_train_pred_final['predicted'] = y_train_pred_final.Churn_Prob.map(lambda x: 1 if x >
2
3 # Let's see the head
4 y_train_pred_final.head()

```

Out[50]:

	Churn	Churn_Prob	CustID	predicted
0	0	0.225111	879	0
1	0	0.274893	5790	0
2	1	0.692126	6498	1
3	1	0.504909	880	1
4	1	0.645261	2784	1

In [51]:

```

1 from sklearn import metrics

```

In [52]:

```

1 # Confusion matrix
2 confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.predicted)
3 print(confusion)

```

```

[[3270  365]
 [ 579  708]]

```

In [53]:

```

1 # Predicted      not_churn    churn
2 # Actual
3 # not_churn      3270         365
4 # churn          579         708

```

In [54]:

```

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

```

0.8082080455099553

Checking VIFs

In [55]:

```

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

```

In [56]:

```
1 # Create a dataframe that will contain the names of all the feature variables and their
2 vif = pd.DataFrame()
3 vif['Features'] = X_train[col].columns
4 vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train[
5 vif['VIF'] = round(vif['VIF'], 2)
6 vif = vif.sort_values(by = "VIF", ascending = False)
7 vif
```

Out[56]:

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

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

In [57]:

```
1 col = col.drop('PhoneService', 1)
2 col
```

Out[57]:

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


In [58]:

```

1 # Let's re-run the model using the selected variables
2 X_train_sm = sm.add_constant(X_train[col])
3 logm3 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
4 res = logm3.fit()
5 res.summary()

```

Out[58]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	No. Observations:	4922
Model:	GLM	Df Residuals:	4907
Model Family:	Binomial	Df Model:	14
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-2017.0
Date:	Thu, 29 Nov 2018	Deviance:	4034.0
Time:	11:23:05	Pearson chi2:	5.94e+03
No. Iterations:	7	Covariance Type:	nonrobust

	coef	std err	z	P> z	[0.025	0.975]
const	-1.3885	0.133	-10.437	0.000	-1.649	-1.128
tenure	-1.4138	0.179	-7.884	0.000	-1.765	-1.062
PaperlessBilling	0.3425	0.089	3.829	0.000	0.167	0.518
TotalCharges	0.5936	0.184	3.225	0.001	0.233	0.954
SeniorCitizen	0.4457	0.099	4.486	0.000	0.251	0.640
Contract_One year	-0.6905	0.128	-5.411	0.000	-0.941	-0.440
Contract_Two year	-1.2646	0.211	-6.002	0.000	-1.678	-0.852
PaymentMethod_Credit card (automatic)	-0.3785	0.113	-3.363	0.001	-0.599	-0.158
PaymentMethod_Mailed check	-0.3769	0.111	-3.407	0.001	-0.594	-0.160
InternetService_Fiber optic	0.6241	0.111	5.645	0.000	0.407	0.841
InternetService_No	-1.0940	0.158	-6.919	0.000	-1.404	-0.784
MultipleLines_Yes	0.1607	0.094	1.712	0.087	-0.023	0.345
OnlineSecurity_Yes	-0.4094	0.102	-4.016	0.000	-0.609	-0.210
TechSupport_Yes	-0.4085	0.101	-4.025	0.000	-0.607	-0.210
StreamingTV_Yes	0.3077	0.094	3.277	0.001	0.124	0.492

In [59]:

```

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

```

In [60]:

```
1 y_train_pred[:10]
```

Out[60]:

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

In [61]:

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

In [62]:

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

Out[62]:

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

In [63]:

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

```
0.8051605038602194
```

So overall the accuracy hasn't dropped much.

Let's check the VIFs again

In [64]:

```

1 vif = pd.DataFrame()
2 vif['Features'] = X_train[col].columns
3 vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train[
4 vif['VIF'] = round(vif['VIF'], 2)
5 vif = vif.sort_values(by = "VIF", ascending = False)
6 vif

```

Out[64]:

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

In [65]:

```

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

```

Out[65]:

```

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

```

In [66]:

```

1 # Let's re-run the model using the selected variables
2 X_train_sm = sm.add_constant(X_train[col])
3 logm4 = sm.GLM(y_train,X_train_sm, family = sm.families.Binomial())
4 res = logm4.fit()
5 res.summary()

```

Out[66]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	No. Observations:	4922
Model:	GLM	Df Residuals:	4908
Model Family:	Binomial	Df Model:	13
Link Function:	logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-2022.5
Date:	Thu, 29 Nov 2018	Deviance:	4044.9
Time:	11:23:06	Pearson chi2:	5.22e+03
No. Iterations:	7	Covariance Type:	nonrobust

	coef	std err	z	P> z	[0.025	0.975]
const	-1.4695	0.130	-11.336	0.000	-1.724	-1.215
tenure	-0.8857	0.065	-13.553	0.000	-1.014	-0.758
PaperlessBilling	0.3367	0.089	3.770	0.000	0.162	0.512
SeniorCitizen	0.4517	0.100	4.527	0.000	0.256	0.647
Contract_One year	-0.6792	0.127	-5.360	0.000	-0.927	-0.431
Contract_Two year	-1.2308	0.208	-5.903	0.000	-1.639	-0.822
PaymentMethod_Credit card (automatic)	-0.3827	0.113	-3.399	0.001	-0.603	-0.162
PaymentMethod_Mailed check	-0.3393	0.110	-3.094	0.002	-0.554	-0.124
InternetService_Fiber optic	0.7914	0.098	8.109	0.000	0.600	0.983
InternetService_No	-1.1205	0.157	-7.127	0.000	-1.429	-0.812
MultipleLines_Yes	0.2166	0.092	2.355	0.019	0.036	0.397
OnlineSecurity_Yes	-0.3739	0.101	-3.684	0.000	-0.573	-0.175
TechSupport_Yes	-0.3611	0.101	-3.591	0.000	-0.558	-0.164
StreamingTV_Yes	0.3995	0.089	4.465	0.000	0.224	0.575

In [67]:

```

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

```

In [68]:

```
1 y_train_pred[:10]
```

Out[68]:

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

In [69]:

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

In [70]:

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

Out[70]:

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

In [71]:

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

0.804754164973588

The accuracy is still practically the same.

Let's now check the VIFs again

In [72]:

```

1 vif = pd.DataFrame()
2 vif['Features'] = X_train[col].columns
3 vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train[
4 vif['VIF'] = round(vif['VIF'], 2)
5 vif = vif.sort_values(by = "VIF", ascending = False)
6 vif

```

Out[72]:

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

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

In [73]:

```

1 # Let's take a look at the confusion matrix again
2 confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.predict)
3 confusion

```

Out[73]:

```

array([[3269, 366],
       [ 595, 692]], dtype=int64)

```

In [74]:

```

1 # Actual/Predicted      not_churn      churn
2 # not_churn             3269           366
3 # churn                 595           692

```

In [75]:

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

Out[75]:

0.804754164973588

Metrics beyond simply accuracy

In [76]:

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

In [77]:

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

Out[77]:

0.5376845376845377

In [78]:

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

Out[78]:

0.8993122420907841

In [79]:

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

0.10068775790921596

In [80]:

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

0.6540642722117203

In [81]:

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

0.8460144927536232

Step 9: Plotting the ROC Curve

An ROC curve demonstrates several things:

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

In [82]:

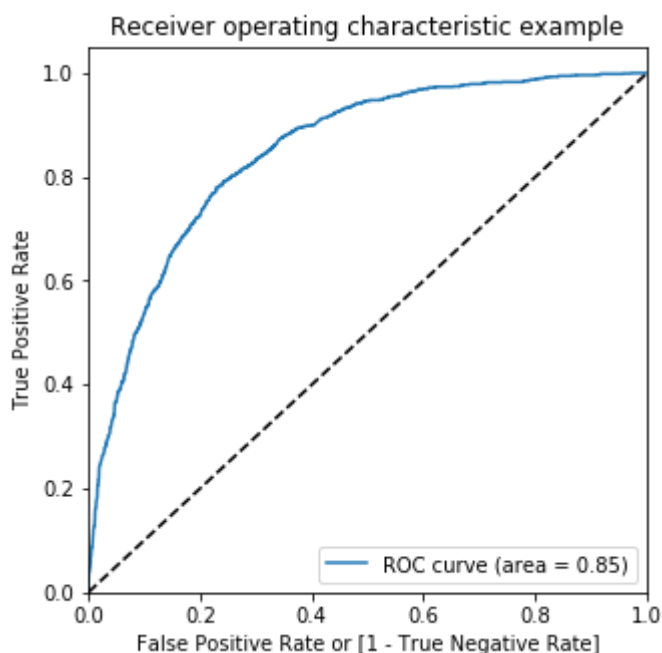
```
1 def draw_roc( actual, probs ):  
2     fpr, tpr, thresholds = metrics.roc_curve( actual, probs,  
3                                             drop_intermediate = False )  
4     auc_score = metrics.roc_auc_score( actual, probs )  
5     plt.figure(figsize=(5, 5))  
6     plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )  
7     plt.plot([0, 1], [0, 1], 'k--')  
8     plt.xlim([0.0, 1.0])  
9     plt.ylim([0.0, 1.05])  
10    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')  
11    plt.ylabel('True Positive Rate')  
12    plt.title('Receiver operating characteristic example')  
13    plt.legend(loc="lower right")  
14    plt.show()  
15  
16    return None
```

In [83]:

```
1 fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Churn, y_train_pred_final.
```


In [84]:

```
1 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 [85]:

```
1 # Let's create columns with different probability cutoffs
2 numbers = [float(x)/10 for x in range(10)]
3 for i in numbers:
4     y_train_pred_final[i]= y_train_pred_final.Churn_Prob.map(lambda x: 1 if x > i else 0)
5 y_train_pred_final.head()
```

Out[85]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0	0	0.282193	879	0	1	1	1	0	0	0	0	0	0	0
1	0	0.268192	5790	0	1	1	1	0	0	0	0	0	0	0
2	1	0.689531	6498	1	1	1	1	1	1	1	1	0	0	0
3	1	0.534214	880	1	1	1	1	1	1	1	0	0	0	0
4	1	0.674332	2784	1	1	1	1	1	1	1	1	0	0	0

In [86]:

```

1 # Now Let's calculate accuracy sensitivity and specificity for various probability cutoff
2 cutoff_df = pd.DataFrame( columns = ['prob', 'accuracy', 'sensi', 'speci'])
3 from sklearn.metrics import confusion_matrix
4
5 # TP = confusion[1,1] # true positive
6 # TN = confusion[0,0] # true negatives
7 # FP = confusion[0,1] # false positives
8 # FN = confusion[1,0] # false negatives
9
10 num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
11 for i in num:
12     cm1 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final[i] )
13     total1=sum(sum(cm1))
14     accuracy = (cm1[0,0]+cm1[1,1])/total1
15
16     speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
17     sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
18     cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
19 print(cutoff_df)

```

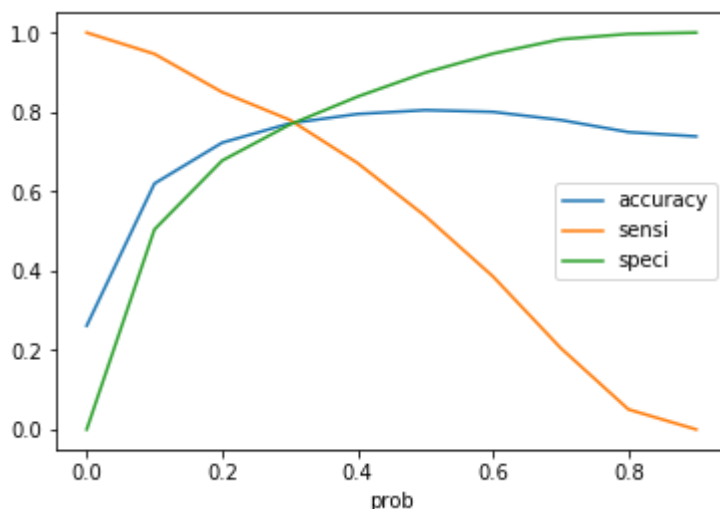
	prob	accuracy	sensi	speci
0.0	0.0	0.261479	1.000000	0.000000
0.1	0.1	0.619667	0.946387	0.503989
0.2	0.2	0.722674	0.850039	0.677579
0.3	0.3	0.771434	0.780109	0.768363
0.4	0.4	0.795002	0.671329	0.838790
0.5	0.5	0.804754	0.537685	0.899312
0.6	0.6	0.800284	0.385392	0.947180
0.7	0.7	0.779764	0.205128	0.983219
0.8	0.8	0.749289	0.050505	0.996699
0.9	0.9	0.738521	0.000000	1.000000

In [87]:

```

1 # Let's plot accuracy sensitivity and specificity for various probabilities.
2 cutoff_df.plot.line(x='prob', y=['accuracy', 'sensi', 'speci'])
3 plt.show()

```



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

In [88]:

```

1 y_train_pred_final['final_predicted'] = y_train_pred_final.Churn_Prob.map( lambda x: 1
2
3 y_train_pred_final.head()

```

Out[88]:

	Churn	Churn_Prob	CustID	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	final_I
0	0	0.282193	879	0	1	1	1	0	0	0	0	0	0	0	
1	0	0.268192	5790	0	1	1	1	0	0	0	0	0	0	0	
2	1	0.689531	6498	1	1	1	1	1	1	1	1	0	0	0	
3	1	0.534214	880	1	1	1	1	1	1	1	0	0	0	0	
4	1	0.674332	2784	1	1	1	1	1	1	1	1	0	0	0	

In [89]:

```

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

```

Out[89]:

0.771434376269809

In [90]:

```

1 confusion2 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.final_predicted)
2 confusion2

```

Out[90]:

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

In [91]:

```

1 TP = confusion2[1,1] # true positive
2 TN = confusion2[0,0] # true negatives
3 FP = confusion2[0,1] # false positives
4 FN = confusion2[1,0] # false negatives

```

In [92]:

```

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

```

Out[92]:

0.7801087801087802

In [93]:

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

Out[93]:

0.768363136176066

In [94]:

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

0.23163686382393398

In [95]:

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

0.5438786565547129

In [96]:

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

0.907997399219766

Precision and Recall

In [97]:

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

In [98]:

```
1 confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_final.prediction)
2 confusion
```

Out[98]:

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

Precision

$TP / TP + FP$

In [99]:

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

Out[99]:

0.6540642722117203

Recall

TP / TP + FN

In [100]:

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

Out[100]:

0.5376845376845377

Using sklearn utilities for the same

In [101]:

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

In [102]:

```
1 ?precision_score
```

In [103]:

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

Out[103]:

0.6540642722117203

In [104]:

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

Out[104]:

0.5376845376845377

Precision and recall tradeoff

In [105]:

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

In [106]:

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

Out[106]:

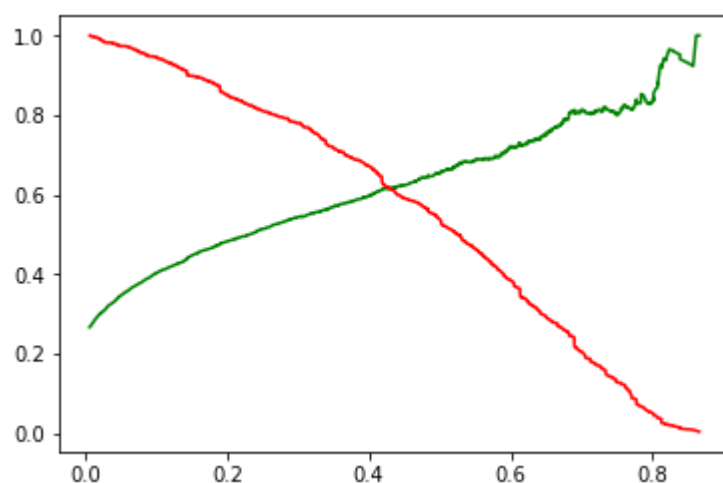
```
(0      0
1      0
2      1
3      1
4      1
5      0
6      0
7      1
8      0
9      1
10     0
11     1
12     1
13     0
14     0
15     0
16     0
17     0
```

In [107]:

```
1 p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn, y_train_pred_final.
```

In [108]:

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



Step 11: Making predictions on the test set

In [109]:

```
1 X_test[['tenure', 'MonthlyCharges', 'TotalCharges']] = scaler.transform(X_test[['tenure',
```

In [110]:

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

Out[110]:

	tenure	PaperlessBilling	SeniorCitizen	Contract_One year	Contract_Two year	PaymentMethod_C card (auton
942	-0.347623	1	0	0	0	
3730	0.999203	1	0	0	0	
1761	1.040015	1	0	0	1	
2283	-1.286319	1	0	0	0	
1872	0.346196	0	0	0	1	

In [111]:

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

Making predictions on the test set

In [112]:

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

In [113]:

```
1 y_test_pred[:10]
```

Out[113]:

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

In [114]:

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

In [115]:

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

Out[115]:

	0
942	0.397413
3730	0.270295
1761	0.010238
2283	0.612692
1872	0.015869

In [116]:

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

In [117]:

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

In [118]:

```
1 # Removing index for both dataframes to append them side by side
2 y_pred_1.reset_index(drop=True, inplace=True)
3 y_test_df.reset_index(drop=True, inplace=True)
```

In [119]:

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

In [120]:

```
1 y_pred_final.head()
```

Out[120]:

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

In [121]:

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

In [122]:

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

In [123]:

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

Out[123]:

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

In [124]:

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

In [125]:

```
1 y_pred_final.head()
```

Out[125]:

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

In [126]:

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

Out[126]:

0.7834123222748816

In [127]:

```
1 confusion2 = metrics.confusion_matrix(y_pred_final.Churn, y_pred_final.final_predicted)
2 confusion2
```

Out[127]:

```
array([[1294,  234],
       [ 223,  359]], dtype=int64)
```

In [128]:

```
1 TP = confusion2[1,1] # true positive
2 TN = confusion2[0,0] # true negatives
3 FP = confusion2[0,1] # false positives
4 FN = confusion2[1,0] # false negatives
```

In [129]:

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

Out[129]:

```
0.6168384879725086
```

In [130]:

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

Out[130]:

```
0.8468586387434555
```