# 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 | MonthlyCharge |
|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | 1 | No | Month-to-month | Yes | Electronic check | 29.8 |
| 1 | 5575-GNVDE | 34 | Yes | One year | No | Mailed check | 56.9 |
| 2 | 3668-QPYBK | 2 | Yes | Month-to-month | Yes | Mailed check | 53.8 |
| 3 | 7795-CFOCW | 45 | No | One year | No | Bank transfer (automatic) | 42.3 |
| 4 | 9237-HQITU | 2 | Yes | Month-to-month | Yes | Electronic check | 70.7 |

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

Out[4]:

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

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

Out[5]:

| | customerID | MultipleLines | InternetService | OnlineSecurity | OnlineBackup | DeviceProtection | Te |
|---|---|---|---|---|---|---|---|
| 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]:
```
# Merging on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')
```

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

## Step 2: Inspecting the Dataframe

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

Out[8]:

| | customerID | tenure | PhoneService | Contract | PaperlessBilling | PaymentMethod | MonthlyCharge |
|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | 1 | No | Month-to-month | Yes | Electronic check | 29.8 |
| 1 | 5575-GNVDE | 34 | Yes | One year | No | Mailed check | 56.9 |
| 2 | 3668-QPYBK | 2 | Yes | Month-to-month | Yes | Mailed check | 53.8 |
| 3 | 7795-CFOCW | 45 | No | One year | No | Bank transfer (automatic) | 42.3 |
| 4 | 9237-HQITU | 2 | Yes | Month-to-month | Yes | Electronic check | 70.7 |

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):
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
```

**The unique values for every feature are printed to the console to get a deeper understanding about the feature values.**

In [ ]:
```
# Looping through columns to get unique values
for i in telecom.columns:
    print(f"Unique {i} count: {telecom[i].nunique()}")
    print(f" {telecom[i].unique()}\n")
```

## 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', 'Dependent
s']

# 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 | MonthlyCharge |
|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | 1 | 0 | Month-to-month | 1 | Electronic check | 29.8 |
| 1 | 5575-GNVDE | 34 | 1 | One year | 0 | Mailed check | 56.9 |
| 2 | 3668-QPYBK | 2 | 1 | Month-to-month | 1 | Mailed check | 53.8 |
| 3 | 7795-CFOCW | 45 | 0 | One year | 0 | Bank transfer (automatic) | 42.3 |
| 4 | 9237-HQITU | 2 | 1 | Month-to-month | 1 | Electronic check | 70.7 |

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 the first one.
dummy1 = pd.get_dummies(telecom[['Contract', 'PaymentMethod', 'gender', 'InternetService']], drop_first=True)

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

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

Out[15]:

| | customerID | tenure | PhoneService | Contract | PaperlessBilling | PaymentMethod | MonthlyCharge |
|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | 1 | 0 | Month-to-month | 1 | Electronic check | 29.8 |
| 1 | 5575-GNVDE | 34 | 1 | One year | 0 | Mailed check | 56.9 |
| 2 | 3668-QPYBK | 2 | 1 | Month-to-month | 1 | Mailed check | 53.8 |
| 3 | 7795-CFOCW | 45 | 0 | One year | 0 | Bank transfer (automatic) | 42.3 |
| 4 | 9237-HQITU | 2 | 1 | Month-to-month | 1 | Electronic check | 70.7 |

5 rows × 29 columns

In [16]:
```
# Creating dummy variables for the remaining categorical variables and droppin
g the level with big names.

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

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

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

# Creating dummy variables for the variable 'DeviceProtection'.
dp = pd.get_dummies(telecom['DeviceProtection'], prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'], 1)
# 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 | MonthlyCharge |
|---|---|---|---|---|---|---|---|
| 0 | 7590-VHVEG | 1 | 0 | Month-to-month | 1 | Electronic check | 29.8 |
| 1 | 5575-GNVDE | 34 | 1 | One year | 0 | Mailed check | 56.9 |
| 2 | 3668-QPYBK | 2 | 1 | Month-to-month | 1 | Mailed check | 53.8 |
| 3 | 7795-CFOCW | 45 | 0 | One year | 0 | Bank transfer (automatic) | 42.3 |
| 4 | 9237-HQITU | 2 | 1 | Month-to-month | 1 | Electronic check | 70.7 |

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','I
nternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
```

In [21]:
```
telecom['TotalCharges'] = pd.to_numeric(telecom['TotalCharges'], errors = 'coe
rce')
telecom = telecom[~np.nan(telecom['TotalCharges'])]
```

In [22]:
```
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7032 entries, 0 to 7042
Data columns (total 32 columns):
customerID                               7032 non-null object
tenure                                   7032 non-null int64
PhoneService                             7032 non-null int64
PaperlessBilling                         7032 non-null int64
MonthlyCharges                           7032 non-null float64
TotalCharges                             7032 non-null float64
Churn                                    7032 non-null int64
SeniorCitizen                            7032 non-null int64
Partner                                  7032 non-null int64
Dependents                               7032 non-null int64
Contract_One year                        7032 non-null uint8
Contract_Two year                        7032 non-null uint8
PaymentMethod_Credit card (automatic)    7032 non-null uint8
PaymentMethod_Electronic check           7032 non-null uint8
PaymentMethod_Mailed check               7032 non-null uint8
gender_Male                              7032 non-null uint8
InternetService_Fiber optic              7032 non-null uint8
InternetService_No                       7032 non-null uint8
MultipleLines_No                         7032 non-null uint8
MultipleLines_Yes                        7032 non-null uint8
OnlineSecurity_No                        7032 non-null uint8
OnlineSecurity_Yes                       7032 non-null uint8
OnlineBackup_No                          7032 non-null uint8
OnlineBackup_Yes                         7032 non-null uint8
DeviceProtection_No                      7032 non-null uint8
DeviceProtection_Yes                     7032 non-null uint8
TechSupport_No                           7032 non-null uint8
TechSupport_Yes                          7032 non-null uint8
StreamingTV_No                           7032 non-null uint8
StreamingTV_Yes                          7032 non-null uint8
StreamingMovies_No                       7032 non-null uint8
StreamingMovies_Yes                      7032 non-null uint8
dtypes: float64(2), int64(7), object(1), uint8(22)
memory usage: 755.4+ KB
```

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

### Checking for Outliers

In [23]:
```
# Checking for outliers in the continuous variables
num_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharge
s']]
```

In [24]:
```
# Checking outliers at 25%, 50%, 75%, 90%, 95% and 99%
num_telecom.describe(percentiles=[.25, .5, .75, .90, .95, .99])
```

Out[24]:

| | tenure | MonthlyCharges | SeniorCitizen | TotalCharges |
|---|---|---|---|---|
| count | 7032.000000 | 7032.000000 | 7032.000000 | 7032.000000 |
| mean | 32.421786 | 64.798208 | 0.162400 | 2283.300441 |
| std | 24.545260 | 30.085974 | 0.368844 | 2266.771362 |
| min | 1.000000 | 18.250000 | 0.000000 | 18.800000 |
| 25% | 9.000000 | 35.587500 | 0.000000 | 401.450000 |
| 50% | 29.000000 | 70.350000 | 0.000000 | 1397.475000 |
| 75% | 55.000000 | 89.862500 | 0.000000 | 3794.737500 |
| 90% | 69.000000 | 102.645000 | 1.000000 | 5976.640000 |
| 95% | 72.000000 | 107.422500 | 1.000000 | 6923.590000 |
| 99% | 72.000000 | 114.734500 | 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 [25]:
```
# Adding up the missing values (column-wise)
telecom.isnull().sum()
```

Out[25]:
```
customerID                               0
tenure                                   0
PhoneService                             0
PaperlessBilling                         0
MonthlyCharges                           0
TotalCharges                             0
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 [26]:
```
# Checking the percentage of missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Out[26]:
```
customerID                                 0.0
tenure                                     0.0
PhoneService                               0.0
PaperlessBilling                           0.0
MonthlyCharges                             0.0
TotalCharges                               0.0
Churn                                      0.0
SeniorCitizen                              0.0
Partner                                    0.0
Dependents                                 0.0
Contract_One year                          0.0
Contract_Two year                          0.0
PaymentMethod_Credit card (automatic)      0.0
PaymentMethod_Electronic check             0.0
PaymentMethod_Mailed check                 0.0
gender_Male                                0.0
InternetService_Fiber optic                0.0
InternetService_No                         0.0
MultipleLines_No                           0.0
MultipleLines_Yes                          0.0
OnlineSecurity_No                          0.0
OnlineSecurity_Yes                         0.0
OnlineBackup_No                            0.0
OnlineBackup_Yes                           0.0
DeviceProtection_No                        0.0
DeviceProtection_Yes                       0.0
TechSupport_No                             0.0
TechSupport_Yes                            0.0
StreamingTV_No                             0.0
StreamingTV_Yes                            0.0
StreamingMovies_No                         0.0
StreamingMovies_Yes                        0.0
dtype: float64
```

In [27]:
```
# Removing NaN TotalCharges rows
telecom = telecom[~np.isnan(telecom['TotalCharges'])]
```

In [28]:
```
# Checking percentage of missing values after removing the missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Out[28]:
```
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 [29]:
```
from sklearn.model_selection import train_test_split
```

In [30]:
```
# Putting feature variable to X
X = telecom.drop(['Churn','customerID'], axis=1)

X.head()
```

Out[30]:

| | 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 [31]:
```
# Putting response variable to y
y = telecom['Churn']

y.head()
```

Out[31]:
```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

In [32]:
```
# 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 [33]:
```
from sklearn.preprocessing import StandardScaler
```

In [34]:
```
scaler = StandardScaler()

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

X_train.head()
```

Out[34]:

| | tenure | PhoneService | PaperlessBilling | MonthlyCharges | TotalCharges | SeniorCitizen | Pa |
|---|---|---|---|---|---|---|---|
| 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 [35]:
```
### Checking the Churn Rate
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn
```

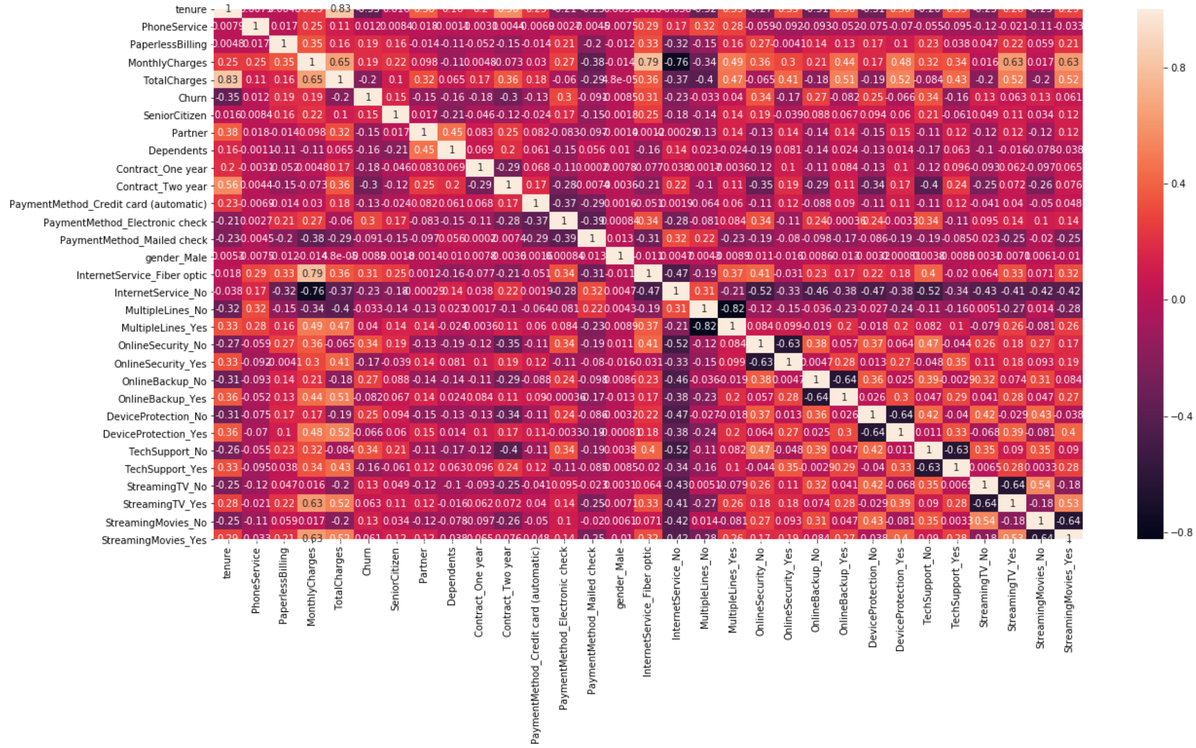
Out[35]: 26.578498293515356

We have almost 27% churn rate

## Step 6: Looking at Correlations

In [36]:
```
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [37]:
```python
# Let's see the correlation matrix
plt.figure(figsize = (20,10))        # Size of the figure
sns.heatmap(telecom.corr(),annot = True)
plt.show()
```



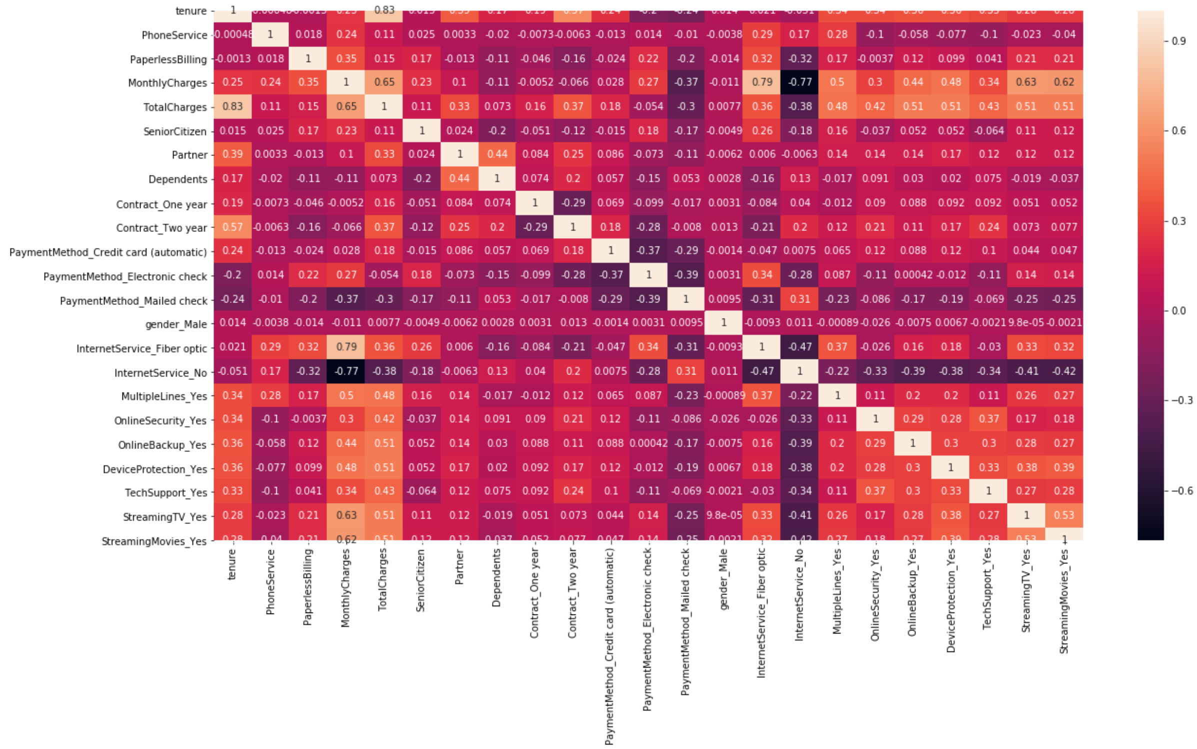
### Dropping highly correlated dummy variables

In [38]:
```python
X_test = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No'
,'DeviceProtection_No','TechSupport_No',
                       'StreamingTV_No','StreamingMovies_No'], 1)
X_train = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_N
o','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 [39]:
```
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 [40]:
```
import statsmodels.api as sm
```

In [41]:
```
# Logistic regression model
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomi
al())
logm1.fit().summary()
```

Out[41]:

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, 23 Apr 2020 | **Deviance:** | 4009.4 |
| **Time:** | 15:33:07 | **Pearson chi2:** | 6.07e+03 |
| **No. Iterations:** | 7 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -3.9382 | 1.546 | -2.547 | 0.011 | -6.969 | -0.908 |
| **tenure** | -1.5172 | 0.189 | -8.015 | 0.000 | -1.888 | -1.146 |
| **PhoneService** | 0.9507 | 0.789 | 1.205 | 0.228 | -0.595 | 2.497 |
| **PaperlessBilling** | 0.3254 | 0.090 | 3.614 | 0.000 | 0.149 | 0.502 |
| **MonthlyCharges** | -2.1806 | 1.160 | -1.880 | 0.060 | -4.454 | 0.092 |
| **TotalCharges** | 0.7332 | 0.198 | 3.705 | 0.000 | 0.345 | 1.121 |
| **SeniorCitizen** | 0.3984 | 0.102 | 3.924 | 0.000 | 0.199 | 0.597 |
| **Partner** | 0.0374 | 0.094 | 0.399 | 0.690 | -0.146 | 0.221 |
| **Dependents** | -0.1430 | 0.107 | -1.332 | 0.183 | -0.353 | 0.067 |
| **Contract_One year** | -0.6578 | 0.129 | -5.106 | 0.000 | -0.910 | -0.405 |
| **Contract_Two year** | -1.2455 | 0.212 | -5.874 | 0.000 | -1.661 | -0.830 |
| **PaymentMethod_Credit card (automatic)** | -0.2577 | 0.137 | -1.883 | 0.060 | -0.526 | 0.011 |
| **PaymentMethod_Electronic check** | 0.1615 | 0.113 | 1.434 | 0.152 | -0.059 | 0.382 |
| **PaymentMethod_Mailed check** | -0.2536 | 0.137 | -1.845 | 0.065 | -0.523 | 0.016 |
| **gender_Male** | -0.0346 | 0.078 | -0.442 | 0.658 | -0.188 | 0.119 |
| **InternetService_Fiber optic** | 2.5124 | 0.967 | 2.599 | 0.009 | 0.618 | 4.407 |
| **InternetService_No** | -2.7792 | 0.982 | -2.831 | 0.005 | -4.703 | -0.855 |
| **MultipleLines_Yes** | 0.5623 | 0.214 | 2.628 | 0.009 | 0.143 | 0.982 |
| **OnlineSecurity_Yes** | -0.0245 | 0.216 | -0.113 | 0.910 | -0.448 | 0.399 |
| **OnlineBackup_Yes** | 0.1740 | 0.212 | 0.822 | 0.411 | -0.241 | 0.589 |
| **DeviceProtection_Yes** | 0.3229 | 0.215 | 1.501 | 0.133 | -0.099 | 0.744 |
| **TechSupport_Yes** | -0.0305 | 0.216 | -0.141 | 0.888 | -0.455 | 0.394 |
| **StreamingTV_Yes** | 0.9598 | 0.396 | 2.423 | 0.015 | 0.183 | 1.736 |
| **StreamingMovies_Yes** | 0.8484 | 0.396 | 2.143 | 0.032 | 0.072 | 1.624 |

## Step 8: Feature Selection Using RFE

In [42]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

In [43]:
```
from sklearn.feature_selection import RFE
rfe = RFE(logreg, 15)             # running RFE with 13 variables as output
rfe = rfe.fit(X_train, y_train)
```

In [44]:
```
rfe.support_
```

Out[44]:
```
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 [45]:
```
list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

Out[45]:
```
[('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 [46]:
```
col = X_train.columns[rfe.support_]
```

In [47]:
```
X_train.columns[~rfe.support_]
```

Out[47]:
```
Index(['PhoneService', 'Partner', 'Dependents',
       'PaymentMethod_Electronic check', 'gender_Male', 'OnlineSecurity_Yes',
       'OnlineBackup_Yes', 'DeviceProtection_Yes'],
      dtype='object')
```

### *Assessing the model with StatsModels*

In [48]:
```
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[48]:

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:** | Thu, 23 Apr 2020 | **Deviance:** | 4022.2 |
| **Time:** | 15:33:14 | **Pearson chi2:** | 6.25e+03 |
| **No. Iterations:** | 7 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -2.2462 | 0.189 | -11.879 | 0.000 | -2.617 | -1.876 |
| **tenure** | -1.5596 | 0.187 | -8.334 | 0.000 | -1.926 | -1.193 |
| **PaperlessBilling** | 0.3436 | 0.090 | 3.832 | 0.000 | 0.168 | 0.519 |
| **MonthlyCharges** | -0.9692 | 0.199 | -4.878 | 0.000 | -1.359 | -0.580 |
| **TotalCharges** | 0.7421 | 0.197 | 3.764 | 0.000 | 0.356 | 1.128 |
| **SeniorCitizen** | 0.4296 | 0.100 | 4.312 | 0.000 | 0.234 | 0.625 |
| **Contract_One year** | -0.6830 | 0.128 | -5.342 | 0.000 | -0.934 | -0.432 |
| **Contract_Two year** | -1.2931 | 0.211 | -6.138 | 0.000 | -1.706 | -0.880 |
| **PaymentMethod_Credit card (automatic)** | -0.3724 | 0.113 | -3.308 | 0.001 | -0.593 | -0.152 |
| **PaymentMethod_Mailed check** | -0.3723 | 0.111 | -3.345 | 0.001 | -0.591 | -0.154 |
| **InternetService_Fiber optic** | 1.5865 | 0.216 | 7.342 | 0.000 | 1.163 | 2.010 |
| **InternetService_No** | -1.6897 | 0.216 | -7.830 | 0.000 | -2.113 | -1.267 |
| **MultipleLines_Yes** | 0.3779 | 0.104 | 3.640 | 0.000 | 0.174 | 0.581 |
| **TechSupport_Yes** | -0.2408 | 0.109 | -2.210 | 0.027 | -0.454 | -0.027 |
| **StreamingTV_Yes** | 0.5796 | 0.114 | 5.102 | 0.000 | 0.357 | 0.802 |
| **StreamingMovies_Yes** | 0.4665 | 0.111 | 4.197 | 0.000 | 0.249 | 0.684 |

In [49]:
```
# Getting the predicted values on the train set
y_train_pred = res.predict(X_train_sm)
y_train_pred[:10]
```

Out[49]:
```
879     0.192642
5790    0.275624
6498    0.599507
880     0.513571
2784    0.648233
3874    0.414846
5387    0.431184
6623    0.801788
4465    0.228194
5364    0.504575
dtype: float64
```

In [50]:
```
y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred[:10]
```

Out[50]:
```
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 [51]:
```
y_train_pred_final = pd.DataFrame({'Churn':y_train.values, 'Churn_Prob':y_trai
n_pred})
y_train_pred_final['CustID'] = y_train.index
y_train_pred_final.head()
```

Out[51]:

| | 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 [52]:
```
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[52]:

| | 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 [53]:
```
from sklearn import metrics
```

In [54]:
```
# Confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_fi
nal.predicted )
print(confusion)
```

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

In [55]:
```
# Predicted     not_churn    churn
# Actual
# not_churn        3270      365
# churn            579       708
```

In [56]:
```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.pred
icted))
```

```
0.8102397399431126
```

### Checking VIFs

In [57]:
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [58]:
```
# 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[58]:

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

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

In [59]:
```
col = col.drop('PhoneService', 1)
col
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-59-00b1d68f1759> in <module>
----> 1 col = col.drop('PhoneService', 1)
      2 col

~\AppData\Local\Continuum\anaconda3\lib\site-packages\pandas\core\indexes\bas
e.py in drop(self, labels, errors)
   5338         if mask.any():
   5339             if errors != "ignore":
-> 5340                 raise KeyError("{} not found in axis".format(labels[m
ask]))
   5341             indexer = indexer[~mask]
   5342         return self.delete(indexer)

KeyError: "['PhoneService'] not found in axis"
```

In [60]:
```
# 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[60]:

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:** | Thu, 23 Apr 2020 | **Deviance:** | 4022.2 |
| **Time:** | 15:33:43 | **Pearson chi2:** | 6.25e+03 |
| **No. Iterations:** | 7 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -2.2462 | 0.189 | -11.879 | 0.000 | -2.617 | -1.876 |
| **tenure** | -1.5596 | 0.187 | -8.334 | 0.000 | -1.926 | -1.193 |
| **PaperlessBilling** | 0.3436 | 0.090 | 3.832 | 0.000 | 0.168 | 0.519 |
| **MonthlyCharges** | -0.9692 | 0.199 | -4.878 | 0.000 | -1.359 | -0.580 |
| **TotalCharges** | 0.7421 | 0.197 | 3.764 | 0.000 | 0.356 | 1.128 |
| **SeniorCitizen** | 0.4296 | 0.100 | 4.312 | 0.000 | 0.234 | 0.625 |
| **Contract_One year** | -0.6830 | 0.128 | -5.342 | 0.000 | -0.934 | -0.432 |
| **Contract_Two year** | -1.2931 | 0.211 | -6.138 | 0.000 | -1.706 | -0.880 |
| **PaymentMethod_Credit card (automatic)** | -0.3724 | 0.113 | -3.308 | 0.001 | -0.593 | -0.152 |
| **PaymentMethod_Mailed check** | -0.3723 | 0.111 | -3.345 | 0.001 | -0.591 | -0.154 |
| **InternetService_Fiber optic** | 1.5865 | 0.216 | 7.342 | 0.000 | 1.163 | 2.010 |
| **InternetService_No** | -1.6897 | 0.216 | -7.830 | 0.000 | -2.113 | -1.267 |
| **MultipleLines_Yes** | 0.3779 | 0.104 | 3.640 | 0.000 | 0.174 | 0.581 |
| **TechSupport_Yes** | -0.2408 | 0.109 | -2.210 | 0.027 | -0.454 | -0.027 |
| **StreamingTV_Yes** | 0.5796 | 0.114 | 5.102 | 0.000 | 0.357 | 0.802 |
| **StreamingMovies_Yes** | 0.4665 | 0.111 | 4.197 | 0.000 | 0.249 | 0.684 |

In [61]:
```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [62]:
```
y_train_pred[:10]
```

Out[62]:
```
array([0.19264205, 0.27562384, 0.59950707, 0.51357126, 0.64823272,
       0.41484553, 0.43118361, 0.80178789, 0.22819404, 0.50457542])
```

In [63]:
```
y_train_pred_final['Churn_Prob'] = y_train_pred
```

In [64]:
```
# 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[64]:

| | 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 [65]:
```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.pred
icted))
```

```
0.8102397399431126
```

So overall the accuracy hasn't dropped much.

***Let's check the VIFs again***

In [66]:
```
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[66]:

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

In [67]:
```
# Let's drop TotalCharges since it has a high VIF
col = col.drop('TotalCharges')
col
```

Out[67]:
```
Index(['tenure', 'PaperlessBilling', 'MonthlyCharges', 'SeniorCitizen',
       'Contract_One year', 'Contract_Two year',
       'PaymentMethod_Credit card (automatic)', 'PaymentMethod_Mailed check',
       'InternetService_Fiber optic', 'InternetService_No',
       'MultipleLines_Yes', 'TechSupport_Yes', 'StreamingTV_Yes',
       'StreamingMovies_Yes'],
      dtype='object')
```

In [68]:
```
# 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[68]:

Generalized Linear Model Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Churn | **No. Observations:** | 4922 |
| **Model:** | GLM | **Df Residuals:** | 4907 |
| **Model Family:** | Binomial | **Df Model:** | 14 |
| **Link Function:** | logit | **Scale:** | 1.0000 |
| **Method:** | IRLS | **Log-Likelihood:** | -2018.5 |
| **Date:** | Thu, 23 Apr 2020 | **Deviance:** | 4037.1 |
| **Time:** | 15:33:56 | **Pearson chi2:** | 5.25e+03 |
| **No. Iterations:** | 7 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -2.1697 | 0.186 | -11.663 | 0.000 | -2.534 | -1.805 |
| **tenure** | -0.9137 | 0.065 | -13.982 | 0.000 | -1.042 | -0.786 |
| **PaperlessBilling** | 0.3332 | 0.089 | 3.726 | 0.000 | 0.158 | 0.508 |
| **MonthlyCharges** | -0.7106 | 0.184 | -3.854 | 0.000 | -1.072 | -0.349 |
| **SeniorCitizen** | 0.4407 | 0.100 | 4.404 | 0.000 | 0.245 | 0.637 |
| **Contract_One year** | -0.6821 | 0.127 | -5.374 | 0.000 | -0.931 | -0.433 |
| **Contract_Two year** | -1.2558 | 0.208 | -6.034 | 0.000 | -1.664 | -0.848 |
| **PaymentMethod_Credit card (automatic)** | -0.3774 | 0.113 | -3.348 | 0.001 | -0.598 | -0.156 |
| **PaymentMethod_Mailed check** | -0.3207 | 0.110 | -2.917 | 0.004 | -0.536 | -0.105 |
| **InternetService_Fiber optic** | 1.5264 | 0.213 | 7.166 | 0.000 | 1.109 | 1.944 |
| **InternetService_No** | -1.5165 | 0.208 | -7.278 | 0.000 | -1.925 | -1.108 |
| **MultipleLines_Yes** | 0.3872 | 0.104 | 3.739 | 0.000 | 0.184 | 0.590 |
| **TechSupport_Yes** | -0.2426 | 0.109 | -2.224 | 0.026 | -0.456 | -0.029 |
| **StreamingTV_Yes** | 0.5779 | 0.113 | 5.126 | 0.000 | 0.357 | 0.799 |
| **StreamingMovies_Yes** | 0.4667 | 0.110 | 4.226 | 0.000 | 0.250 | 0.683 |

In [69]:
```
y_train_pred = res.predict(X_train_sm).values.reshape(-1)
```

In [70]:
```
y_train_pred[:10]
```

Out[70]:
```
array([0.22669491, 0.32279486, 0.61112237, 0.56497818, 0.68324356,
       0.38658503, 0.36867571, 0.80505887, 0.25567371, 0.52400218])
```

In [71]:
```
y_train_pred_final['Churn_Prob'] = y_train_pred
```

In [72]:
```
# 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[72]:

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

In [73]:
```
# Let's check the overall accuracy.
print(metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.pred
icted))
```

```
0.8063795205201137
```

The accuracy is still practically the same.

***Let's now check the VIFs again***

In [74]:
```
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[74]:

| | Features | VIF |
|---|---|---|
| 2 | MonthlyCharges | 10.63 |
| 8 | InternetService_Fiber optic | 5.44 |
| 9 | InternetService_No | 5.15 |
| 5 | Contract_Two year | 3.13 |
| 12 | StreamingTV_Yes | 2.79 |
| 13 | StreamingMovies_Yes | 2.79 |
| 1 | PaperlessBilling | 2.76 |
| 0 | tenure | 2.38 |
| 10 | MultipleLines_Yes | 2.38 |
| 11 | TechSupport_Yes | 1.94 |
| 4 | Contract_One year | 1.85 |
| 7 | PaymentMethod_Mailed check | 1.69 |
| 6 | PaymentMethod_Credit card (automatic) | 1.45 |
| 3 | SeniorCitizen | 1.33 |

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

In [75]:
```
# Let's take a look at the confusion matrix again
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_fi
nal.predicted )
confusion
```

Out[75]:
```
array([[3278,  357],
       [ 596,  691]], dtype=int64)
```

In [76]:
```
# Actual/Predicted     not_churn    churn
        # not_churn        3269       366
        # churn            595        692
```

In [77]:
```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[77]:
```
0.8063795205201137
```

# Metrics beyond simply accuracy

In [78]:
```
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 [79]:
```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

Out[79]:
```
0.5369075369075369
```

In [80]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[80]:
```
0.9017881705639614
```

In [81]:
```
# Calculate false postive rate - predicting churn when customer does not have
 churned
print(FP/ float(TN+FP))
```

```
0.09821182943603851
```

In [82]:
```
# positive predictive value
print (TP / float(TP+FP))
```

```
0.6593511450381679
```

In [83]:
```
# Negative predictive value
print (TN / float(TN+ FN))
```

```
0.8461538461538461
```

## Step 9: Plotting the ROC Curve

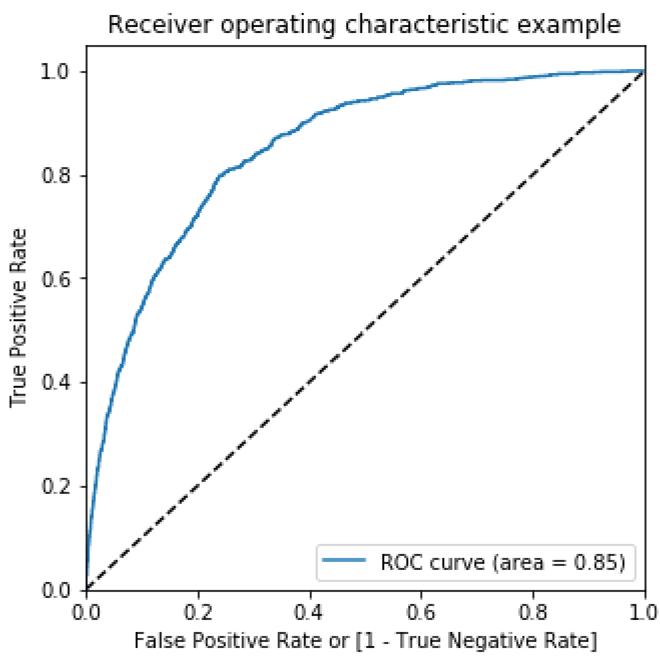
An ROC curve demonstrates several things:

- 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 [192]:
```
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 [85]:
```
fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Churn, y_train_pr
ed_final.Churn_Prob, drop_intermediate = False )
```

In [86]:
```
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 [87]:
```
# Let's create columns with different probability cutoffs
numbers = [float(x)/10 for x in range(10)]
for i in numbers:
    y_train_pred_final[i]= y_train_pred_final.Churn_Prob.map(lambda x: 1 if x
> i else 0)
y_train_pred_final.head()
```

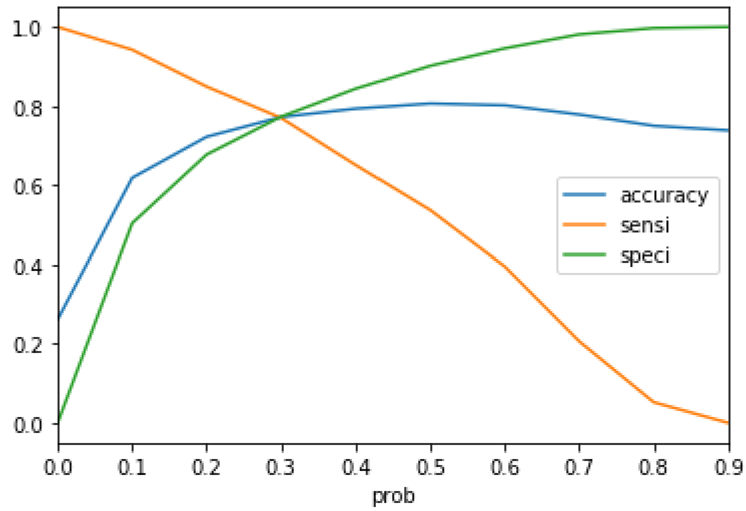
Out[87]:

| | 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 | 0.226695 | 879 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0.322795 | 5790 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0.611122 | 6498 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 3 | 1 | 0.564978 | 880 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 4 | 1 | 0.683244 | 2784 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

In [88]:
```
# Now let's calculate accuracy sensitivity and specificity for various probabi
lity 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_fina
l[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

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

```
     prob  accuracy     sensi     speci
0.0   0.0  0.261479  1.000000  0.000000
0.1   0.1  0.618448  0.942502  0.503714
0.2   0.2  0.722267  0.849262  0.677304
0.3   0.3  0.772247  0.770008  0.773040
0.4   0.4  0.793377  0.651127  0.843741
0.5   0.5  0.806380  0.536908  0.901788
0.6   0.6  0.801910  0.394716  0.946080
0.7   0.7  0.778545  0.205905  0.981293
0.8   0.8  0.750102  0.052059  0.997249
0.9   0.9  0.738521  0.000000  1.000000
```

In [89]:
```
# 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 [90]:
```
y_train_pred_final['final_predicted'] = y_train_pred_final.Churn_Prob.map( lam
bda x: 1 if x > 0.3 else 0)

y_train_pred_final.head()
```

Out[90]:

| | 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_pı |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.226695 | 879 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 1 | 0 | 0.322795 | 5790 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 2 | 1 | 0.611122 | 6498 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | |
| 3 | 1 | 0.564978 | 880 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | |
| 4 | 1 | 0.683244 | 2784 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | |

In [91]:
```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Churn, y_train_pred_final.final_pred
icted)
```

Out[91]: 0.7722470540430719

In [92]:
```
confusion2 = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_f
inal.final_predicted )
confusion2
```

Out[92]:
```
array([[2810,  825],
       [ 296,  991]], dtype=int64)
```

In [93]:
```
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 [94]:
```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

Out[94]:
```
0.77000777000777
```

In [95]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[95]:
```
0.7730398899587345
```

In [96]:
```
# Calculate false postive rate - predicting churn when customer does not have
 churned
print(FP/ float(TN+FP))
```

```
0.22696011004126548
```

In [97]:
```
# Positive predictive value
print (TP / float(TP+FP))
```

```
0.545704845814978
```

In [98]:
```
# Negative predictive value
print (TN / float(TN+ FN))
```

```
0.9047005795235029
```

## Precision and Recall

In [99]:
```
#Looking at the confusion matrix again
```

In [100]:
```
confusion = metrics.confusion_matrix(y_train_pred_final.Churn, y_train_pred_fi
nal.predicted )
confusion
```

Out[100]:
```
array([[3278,  357],
       [ 596,  691]], dtype=int64)
```

***Precision***

TP / TP + FP

In [101]:
```
confusion[1,1]/(confusion[0,1]+confusion[1,1])
```

Out[101]: 0.6593511450381679

***Recall***

TP / TP + FN

In [102]:
```
confusion[1,1]/(confusion[1,0]+confusion[1,1])
```

Out[102]: 0.5369075369075369

Using sklearn utilities for the same

In [103]:
```
from sklearn.metrics import precision_score, recall_score
```

In [104]:
```
?precision_score
```

In [105]:
```
precision_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[105]: 0.6593511450381679

In [106]:
```
recall_score(y_train_pred_final.Churn, y_train_pred_final.predicted)
```

Out[106]: 0.5369075369075369

## Precision and recall tradeoff

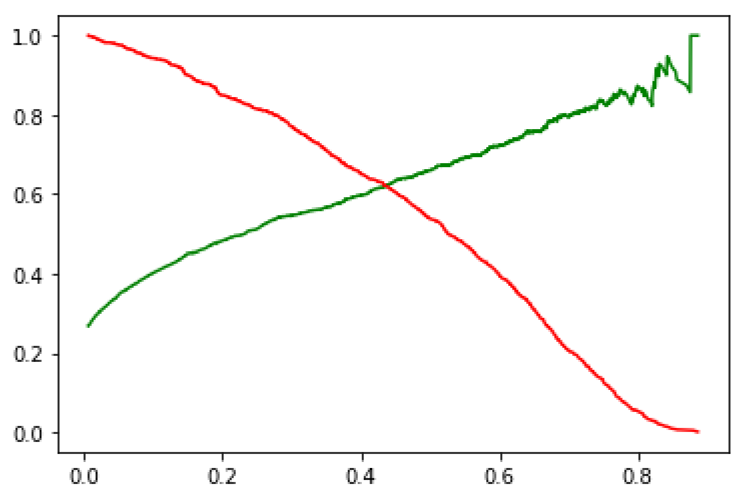
In [107]:
```
from sklearn.metrics import precision_recall_curve
```

In [108]:
```
y_train_pred_final.Churn, y_train_pred_final.predicted
```

Out[108]:
```
(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 [109]:
```
p, r, thresholds = precision_recall_curve(y_train_pred_final.Churn, y_train_pr
ed_final.Churn_Prob)
```

In [110]:
```
plt.plot(thresholds, p[:-1], "g-")
plt.plot(thresholds, r[:-1], "r-")
plt.show()
```




## Step 11: Making predictions on the test set

In [111]:
```
X_test[['tenure','MonthlyCharges','TotalCharges']] = scaler.transform(X_test[[
'tenure','MonthlyCharges','TotalCharges']])
```

In [112]:
```
X_test = X_test[col]
X_test.head()
```

Out[112]:

| | tenure | PaperlessBilling | MonthlyCharges | SeniorCitizen | Contract_One year | Contract_Two year | F |
|---|---|---|---|---|---|---|---|
| 942 | -0.347623 | 1 | 0.499951 | 0 | 0 | 0 | |
| 3730 | 0.999203 | 1 | 1.319685 | 0 | 0 | 0 | |
| 1761 | 1.040015 | 1 | -1.342374 | 0 | 0 | 1 | |
| 2283 | -1.286319 | 1 | 0.223935 | 0 | 0 | 0 | |
| 1872 | 0.346196 | 0 | -1.500335 | 0 | 0 | 1 | |

In [113]:
```
X_test_sm = sm.add_constant(X_test)
```

Making predictions on the test set

In [114]:
```
y_test_pred = res.predict(X_test_sm)
```

In [115]:
```
y_test_pred[:10]
```

Out[115]:
```
942     0.435743
3730    0.248518
1761    0.009998
2283    0.595171
1872    0.014889
1970    0.697307
2532    0.284275
1616    0.009756
2485    0.598246
5914    0.131993
dtype: float64
```

In [116]:
```
# Converting y_pred to a dataframe which is an array
y_pred_1 = pd.DataFrame(y_test_pred)
```

In [117]:
```
# Let's see the head
y_pred_1.head()
```

Out[117]:

| | 0 |
|---|---|
| 942 | 0.435743 |
| 3730 | 0.248518 |
| 1761 | 0.009998 |
| 2283 | 0.595171 |
| 1872 | 0.014889 |

In [118]:
```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
```

In [119]:
```
# Putting CustID to index
y_test_df['CustID'] = y_test_df.index
```

In [120]:
```
# 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 [121]:
```
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df, y_pred_1],axis=1)
```

In [122]:
```
y_pred_final.head()
```

Out[122]:

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

In [123]:
```
# Renaming the column
y_pred_final= y_pred_final.rename(columns={ 0 : 'Churn_Prob'})
```

In [124]:
```
# Rearranging the columns
y_pred_final = y_pred_final[['CustID','Churn','Churn_Prob']]
```

In [125]:
```
# Let's see the head of y_pred_final
y_pred_final.head()
```

Out[125]:

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

In [126]:
```
y_pred_final['final_predicted'] = y_pred_final.Churn_Prob.map(lambda x: 1 if x
> 0.42 else 0)
```

In [127]:
```
y_pred_final.head()
```

Out[127]:

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

In [128]:
```
# Let's check the overall accuracy.
metrics.accuracy_score(y_pred_final.Churn, y_pred_final.final_predicted)
```

Out[128]:
```
0.7848341232227488
```

In [129]:
```
confusion2 = metrics.confusion_matrix(y_pred_final.Churn, y_pred_final.final_p
redicted )
confusion2
```

Out[129]:
```
array([[1288,  240],
       [ 214,  368]], dtype=int64)
```

In [130]:
```
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 [131]:
```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

Out[131]:
```
0.6323024054982818
```

In [132]:
```
# Let us calculate specificity
TN / float(TN+FP)
```

Out[132]:
```
0.8429319371727748
```

## Using Decision Trees

In [97]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, rand
om_state=100)
```

In [98]:
```
X_train.shape, X_test.shape
```

Out[98]:
```
((4922, 30), (2110, 30))
```

In [99]:
```
from sklearn.tree import DecisionTreeClassifier
```

In [100]:
```
dt_base = DecisionTreeClassifier(random_state=42, max_depth=4)
```

In [101]:
```
dt_base.fit(X_train, y_train)
```

Out[101]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [102]:
```
y_train_pred = dt_base.predict(X_train)
y_test_pred = dt_base.predict(X_test)
```

In [103]:
```
from sklearn.metrics import classification_report
```

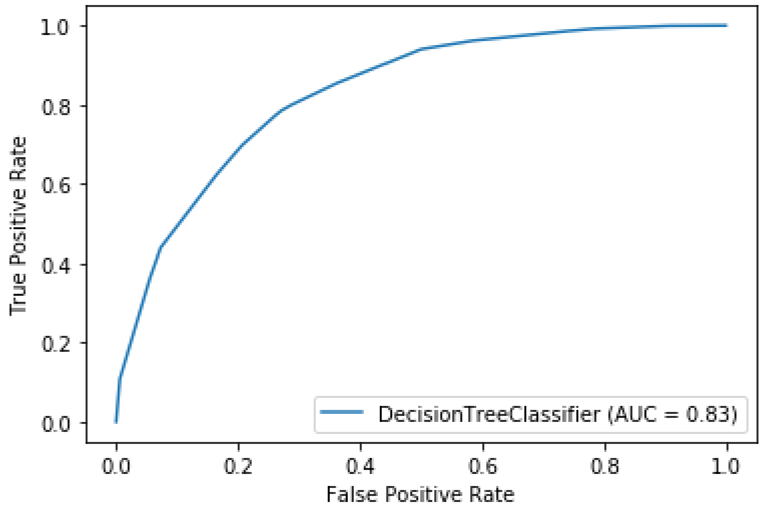
In [104]:
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.92      0.86      1528
           1       0.67      0.45      0.54       582

    accuracy                           0.79      2110
   macro avg       0.74      0.68      0.70      2110
weighted avg       0.78      0.79      0.77      2110
```

### Plot the ROC curve

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

In [106]:
```
plot_roc_curve(dt_base, X_train, y_train, drop_intermediate=False)
plt.show()
```




### Hyper-parameter tuning for the Decision Tree

In [93]:
```
from sklearn.model_selection import GridSearchCV
```

In [94]:
```
dt = DecisionTreeClassifier(random_state=42)
```

In [96]:
```
params = {
    "max_depth": [2,3,5,10,20],
    "min_samples_leaf": [5,10,20,50,100,500]
}
```

In [107]:
```
grid_search = GridSearchCV(estimator=dt,
                           param_grid=params,
                           cv=4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
```

In [108]:
```
%%time
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 30 candidates, totalling 120 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    6.7s

Wall time: 7.25 s

[Parallel(n_jobs=-1)]: Done 120 out of 120 | elapsed:    7.1s finished
```

Out[108]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=Non
e,
                                              criterion='gini', max_depth=Non
e,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=42,
                                              splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 500]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

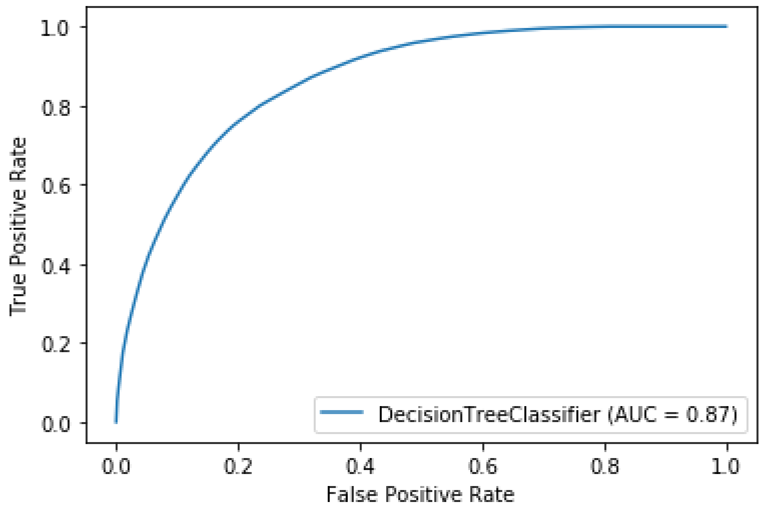
In [109]:
```
grid_search.best_score_
```

Out[109]:
```
0.7986589658747929
```

In [110]:
```
dt_best = grid_search.best_estimator_
dt_best
```

Out[110]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=10, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=50, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=42, splitter='best')
```

In [111]:
```
plot_roc_curve(dt_best, X_train, y_train)
plt.show()
```




# Using Random Forest

In [112]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [115]:
```
rf = RandomForestClassifier(n_estimators=10, max_depth=4, max_features=5, rand
om_state=100, oob_score=True)
```

In [116]:
```
%%time
rf.fit(X_train, y_train)
```

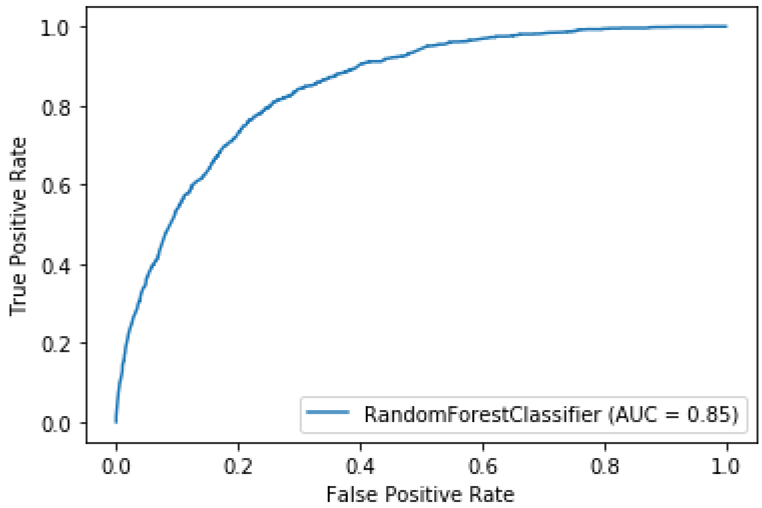
```
Wall time: 82.2 ms
```

Out[116]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=4, max_features=5,
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=True, random_state=100, verbose
=0,
                       warm_start=False)
```

In [117]:
```
rf.oob_score_
```

Out[117]:
```
0.7850467289719626
```

In [118]:
```
plot_roc_curve(rf, X_train, y_train)
plt.show()
```




## Hyper-parameter tuning for the Random Forest

In [119]:
```
rf = RandomForestClassifier(random_state=42, n_jobs=-1)
```

In [120]:
```
params = {
    'max_depth': [2,3,5,10,20],
    'min_samples_leaf': [5,10,20,50,100,200],
    'n_estimators': [10, 25, 50, 100]
}
```

In [121]:
```
grid_search = GridSearchCV(estimator=rf,
                           param_grid=params,
                           cv = 4,
                           n_jobs=-1, verbose=1, scoring="accuracy")
```

In [122]:
```
%%time
grid_search.fit(X_train, y_train)
```

```
Fitting 4 folds for each of 120 candidates, totalling 480 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done 184 tasks      | elapsed:   13.6s
[Parallel(n_jobs=-1)]: Done 434 tasks      | elapsed:   29.6s
[Parallel(n_jobs=-1)]: Done 480 out of 480 | elapsed:   32.4s finished

Wall time: 32.9 s
```

Out[122]:
```
GridSearchCV(cv=4, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=Non
e,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=100, n_jobs=-1,
                                              oob_score=False, random_state=4
2,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10, 20],
                         'min_samples_leaf': [5, 10, 20, 50, 100, 200],
                         'n_estimators': [10, 25, 50, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring='accuracy', verbose=1)
```

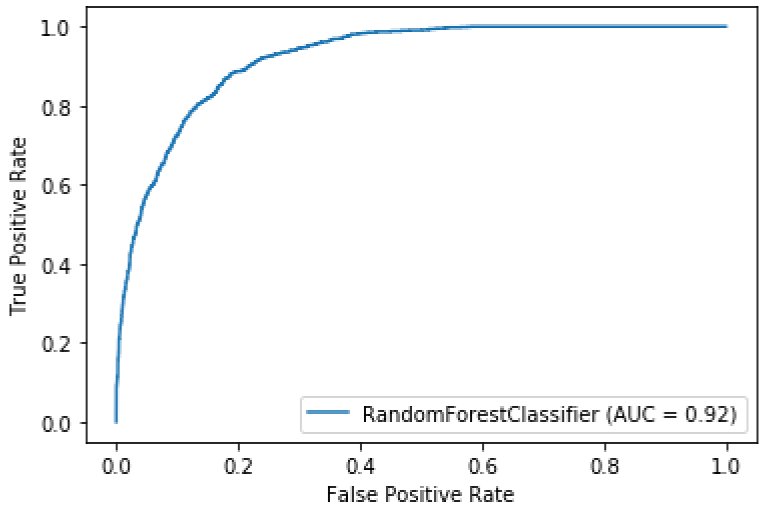
In [123]:
```
grid_search.best_score_
```

Out[123]:
```
0.8047555361824943
```

In [124]:
```
rf_best = grid_search.best_estimator_
rf_best
```

Out[124]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=10, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=5, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=-1, oob_score=False, random_state=42, verbose=
0,
                       warm_start=False)
```

In [125]:
```
plot_roc_curve(rf_best, X_train, y_train)
plt.show()
```




In [126]:
```
rf_best.feature_importances_
```

Out[126]:
```
array([0.18467568, 0.00428761, 0.01680924, 0.10468866, 0.1391503 ,
       0.01617061, 0.00919805, 0.01009522, 0.0351729 , 0.0500925 ,
       0.00752117, 0.04194471, 0.00788822, 0.01184186, 0.07714897,
       0.01677421, 0.00910346, 0.00906618, 0.06686565, 0.01208793,
       0.0221372 , 0.01153073, 0.01726205, 0.00776034, 0.06110446,
       0.01178177, 0.01092192, 0.00918687, 0.00861561, 0.00911594])
```

In [127]:
```
imp_df = pd.DataFrame({
    "Varname": X_train.columns,
    "Imp": rf_best.feature_importances_
})
```

In [128]:
```
imp_df.sort_values(by="Imp", ascending=False)
```

Out[128]:

| | Varname | Imp |
|---|---|---|
| 0 | tenure | 0.184676 |
| 4 | TotalCharges | 0.139150 |
| 3 | MonthlyCharges | 0.104689 |
| 14 | InternetService_Fiber optic | 0.077149 |
| 18 | OnlineSecurity_No | 0.066866 |
| 24 | TechSupport_No | 0.061104 |
| 9 | Contract_Two year | 0.050092 |
| 11 | PaymentMethod_Electronic check | 0.041945 |
| 8 | Contract_One year | 0.035173 |
| 20 | OnlineBackup_No | 0.022137 |
| 22 | DeviceProtection_No | 0.017262 |
| 2 | PaperlessBilling | 0.016809 |
| 15 | InternetService_No | 0.016774 |
| 5 | SeniorCitizen | 0.016171 |
| 19 | OnlineSecurity_Yes | 0.012088 |
| 13 | gender_Male | 0.011842 |
| 25 | TechSupport_Yes | 0.011782 |
| 21 | OnlineBackup_Yes | 0.011531 |
| 26 | StreamingTV_No | 0.010922 |
| 7 | Dependents | 0.010095 |
| 6 | Partner | 0.009198 |
| 27 | StreamingTV_Yes | 0.009187 |
| 29 | StreamingMovies_Yes | 0.009116 |
| 16 | MultipleLines_No | 0.009103 |
| 17 | MultipleLines_Yes | 0.009066 |
| 28 | StreamingMovies_No | 0.008616 |
| 12 | PaymentMethod_Mailed check | 0.007888 |
| 23 | DeviceProtection_Yes | 0.007760 |
| 10 | PaymentMethod_Credit card (automatic) | 0.007521 |
| 1 | PhoneService | 0.004288 |

In [ ]:

In [ ]:

In [ ]:

```

```