This project consists of 3000 marks and has to be submitted in .ipynb/PDF format for evaluation.

High Level Machine Learning Classification Project Life Cycle

1.Domain Introduction

We have the customer data for a **telecom** company which offers many services like phone, internet, TV Streaming and Movie Streaming.

2.Problem Statement

"Find the Best model to predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs."

3. Data Source

Available at : IBM watson analytics page (https://community.watsonanalytics.com/wp-content/uploads/2015/03/WA_Fn-UseC_-Telco-Customer-Churn.csv?cm_mc_uid=14714377267115403444551&cm_mc_sid_50200000=12578191540344455127&cm_mc_sid_52640

4. Data Description

This data set provides info to help you predict behavior to retain customers. You can analyze all relevant customer data and develop focused customer retention programs.

A telecommunications company is concerned about the number of customers leaving their landline business for cable competitors. They need to understand who is leaving. Imagine that you're an analyst at this company and you have to find out who is leaving and why.

The data set includes information about:

Customers who left within the last month – the column is called Churn

Services that each customer has signed up for – phone, multiple lines, internet, online security, online backup, device protection, tech support, and streaming TV and movies

Customer account information – how long they've been a customer, contract, payment method, paperless billing, monthly charges, and total charges

Demographic info about customers – gender, age range, and if they have partners and dependents

5. Identify the target variable

The Goal is to predict whether or not a particular customer is likely to retain services. This is represented by the Churn column in dataset. Churn=Yes means customer leaves the company, whereas Churn=No implies customer is retained by the company.

6. Read the data

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [2]:

```
df = pd.read_csv('../datasets/WA_Fn-UseC_-Telco-Customer-Churn.csv',index_col='customerID')
```

7. Inspect the data

In [3]:

```
df.head()
```

Out[3]:

customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	I
7590-VHVEG	Female	0	Yes	No	1	No	No phone service	
5575-GNVDE	Male	0	No	No	34	Yes	No	
3668-QPYBK	Male	0	No	No	2	Yes	No	
7795-CFOCW	Male	0	No	No	45	No	No phone service	
9237-HQITU	Female	0	No	No	2	Yes	No	

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 7590-VHVEG to 3186-AJIEK
Data columns (total 20 columns):
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        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
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
dtypes: float64(1), int64(2), object(17)
memory usage: 1.1+ MB
```

df.describe()

In [6]:

```
df.describe(include=object)
```

Out[6]:

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineSecur
count	7043	7043	7043	7043	7043	7043	70
unique	2	2	2	2	3	3	
top	Male	No	No	Yes	No	Fiber optic	
freq	3555	3641	4933	6361	3390	3096	34

8. Data Manipulation

Data Manipulation

In [7]:

```
df.isna().any()
```

Out[7]:

```
gender              False
SeniorCitizen       False
Partner             False
Dependents          False
tenure              False
PhoneService        False
MultipleLines       False
InternetService     False
OnlineSecurity      False
OnlineBackup        False
DeviceProtection    False
TechSupport         False
StreamingTV         False
StreamingMovies     False
Contract            False
PaperlessBilling    False
PaymentMethod       False
MonthlyCharges      False
TotalCharges        False
Churn               False
dtype: bool
```

In [8]:

```
df[df['TotalCharges'].isna()]
```

Out[8]:

	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	I
customerID								

In [9]:

```
len(df[df['TotalCharges'].isna()])
```

Out[9]:

```
0
```

Here we can see that Total Charges is an object variable. Let's Change it to float

In [10]:

```
# We need to convert the Total Charges from object type to Numeric
df['TotalCharges'] = df['TotalCharges'].replace(r'\s+', np.nan, regex=True)
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'])

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7043 entries, 7590-VHVEG to 3186-AJIEK
Data columns (total 20 columns):
gender              7043 non-null object
SeniorCitizen       7043 non-null int64
Partner             7043 non-null object
Dependents          7043 non-null object
tenure              7043 non-null int64
PhoneService        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
Contract            7043 non-null object
PaperlessBilling    7043 non-null object
PaymentMethod       7043 non-null object
MonthlyCharges      7043 non-null float64
TotalCharges        7032 non-null float64
Churn               7043 non-null object
dtypes: float64(2), int64(2), object(16)
memory usage: 1.1+ MB
```

every missing value record comes from customers who has not opted out

** Imputation **

In [11]:

```
df['TotalCharges'] = df['TotalCharges'].fillna((df['TotalCharges'].mean()))
```

** Data formating **

9. Exploratory Data Analysis

In [12]:

```
df_categorical = df.select_dtypes(include=object)

column_categorical = df_categorical.columns
```

In [13]:

```
df_categorical.head()
```

Out[13]:

	gender	Partner	Dependents	PhoneService	MultipleLines	InternetService	OnlineS
customerID							
7590-VHVEG	Female	Yes	No	No	No phone service	DSL	
5575-GNVDE	Male	No	No	Yes	No	DSL	
3668-QPYBK	Male	No	No	Yes	No	DSL	
7795-CFOCW	Male	No	No	No	No phone service	DSL	
9237-HQITU	Female	No	No	Yes	No	Fiber optic	

In [14]:

```
df_numerical = df.select_dtypes(include=np.float)

column_numerical = df_numerical.columns
```

In [15]:

```
df_numerical.head()
```

Out[15]:

	MonthlyCharges	TotalCharges
customerID		
7590-VHVEG	29.85	29.85
5575-GNVDE	56.95	1889.50
3668-QPYBK	53.85	108.15
7795-CFOCW	42.30	1840.75
9237-HQITU	70.70	151.65

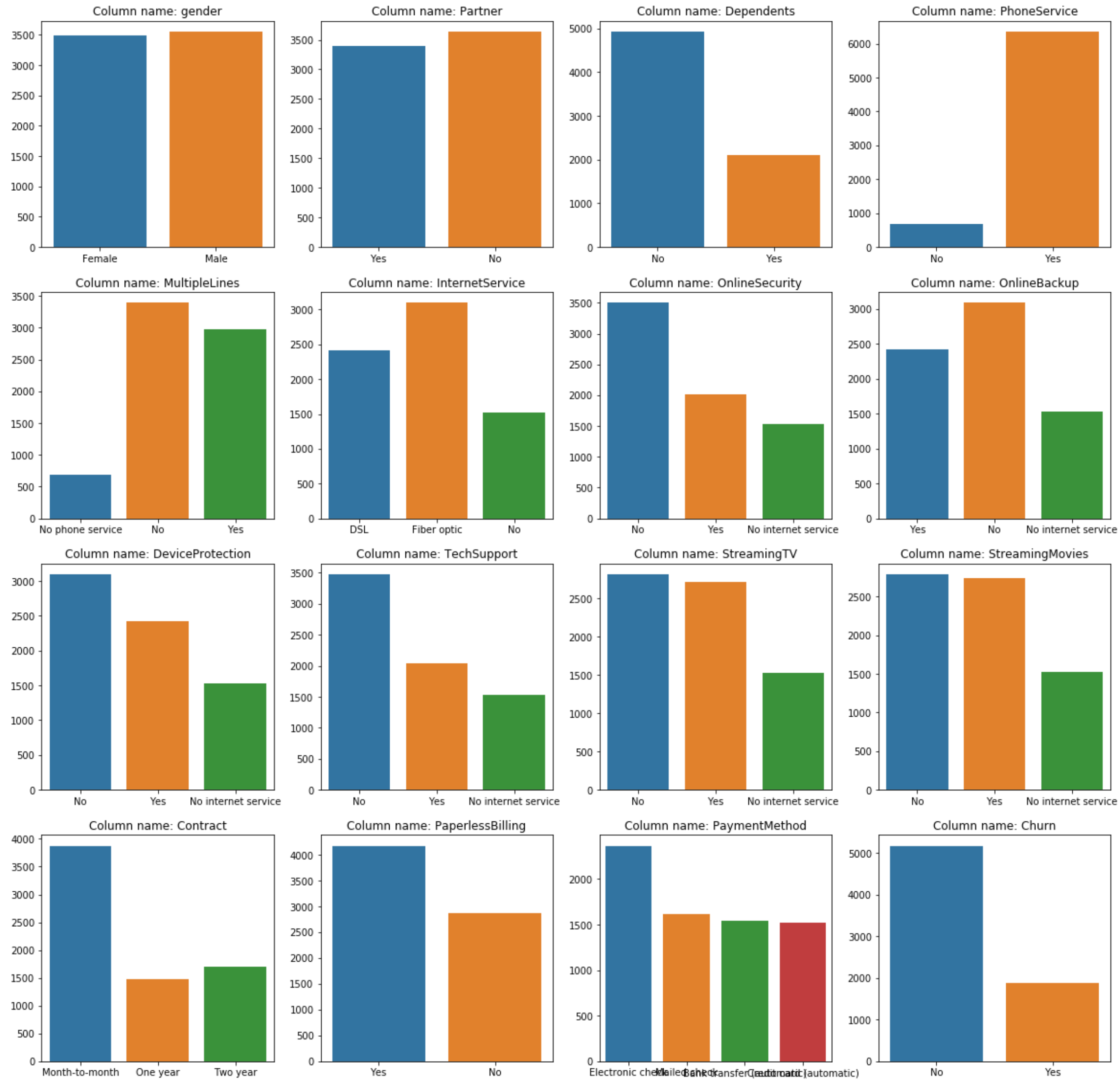
Univariate Analysis

In [16]:

```
def display_plot(df, col_to_exclude, object_mode = True):
    """
     This function plots the count or distribution of each column in the dataframe based on
     @Args
       df: pandas dataframe
       col_to_exclude: specific column to exclude from the plot, used for excluded key
       object_mode: whether to plot on object data types or not (default: True)

     Return
       No object returned but visualized plot will return based on specified inputs
    """
    n = 0
    this = []

    if object_mode:
        nrows = 4
        ncols = 4
        width = 20
        height = 20

    else:
        nrows = 2
        ncols = 2
        width = 14
        height = 10


    for column in df.columns:
        if object_mode:
            if (df[column].dtypes == 'O') & (column != col_to_exclude):
                this.append(column)


        else:
            if (df[column].dtypes != 'O'):
                this.append(column)


    fig, ax = plt.subplots(nrows, ncols, sharex=False, sharey=False, figsize=(width, height
    for row in range(nrows):
        for col in range(ncols):
            if object_mode:
                g = sns.countplot(df[this[n]], ax=ax[row][col])
            else:
                g = sns.distplot(df[this[n]], ax = ax[row][col])



            ax[row,col].set_title("Column name: {}".format(this[n]))
            ax[row, col].set_xlabel("")
            ax[row, col].set_ylabel("")
            n += 1
    plt.show();
    return None
```

In [17]:

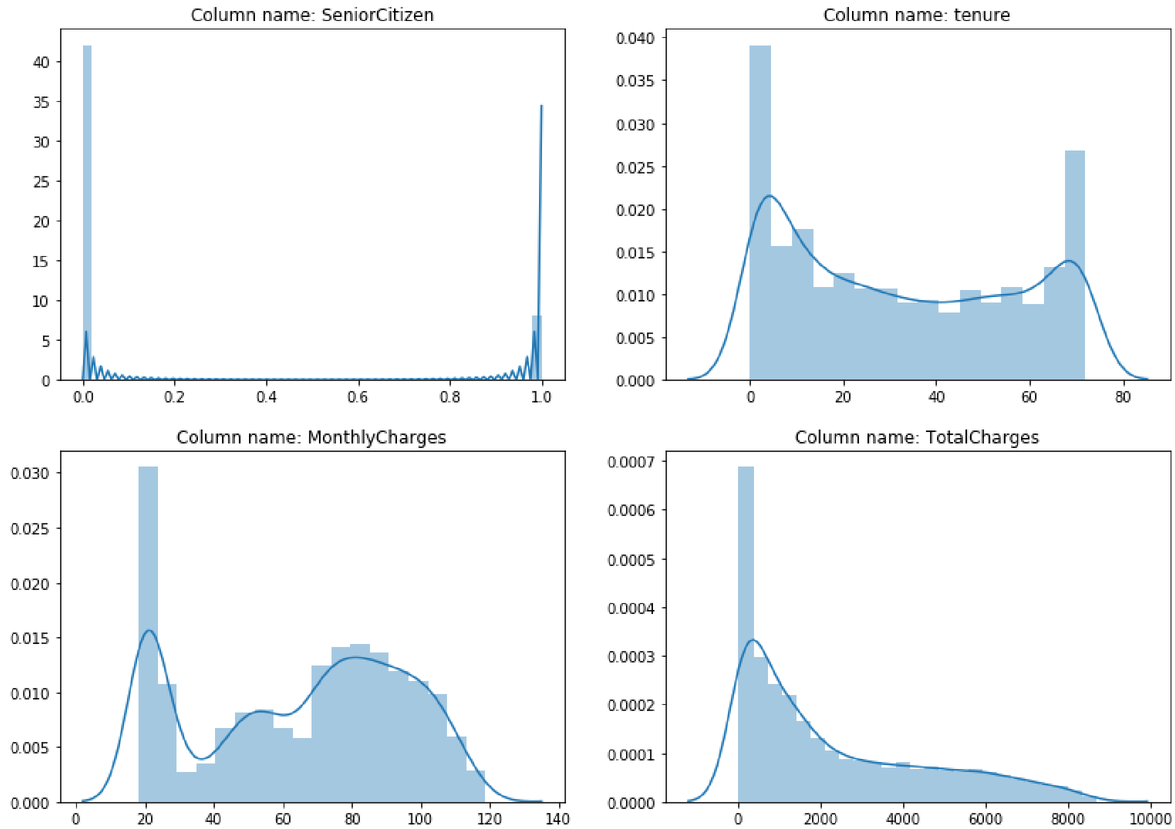
```
display_plot(df, 'customerid', object_mode = True)
```



In [18]:

```
display_plot(df, 'customerid', object_mode = False)
```

```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/scipy/sta
ts/stats.py:1713: FutureWarning: Using a non-tuple sequence for multidimensi
onal indexing is deprecated; use `arr[tuple(seq)]` instead of `arr[seq]`. In
the future this will be interpreted as an array index, `arr[np.array(seq)]`,
which will result either in an error or a different result.
  return np.add.reduce(sorted[indexer] * weights, axis=axis) / sumval
```



feature Engineering

Based on the value of the services the subscribers subscribed to, there are **yes**, **no**, and **no phone / internet service**. These are somewhat related to primary products. Examples are illustrated through *panda crosstab* function below:

1. **Phone service (Primary) and Multiple lines (Secondary)**
    - If the subscribers have phone service, they may have multiple lines (yes or no).
    - But if the subscribers don't have phone service, the subscribers will never have multiple lines.

In [19]:

```
pd.crosstab(index = df["PhoneService"], columns = df["MultipleLines"])
```

Out[19]:

MultipleLines	No	No phone service	Yes
PhoneService			
No	0	682	0
Yes	3390	0	2971

2. **Internet Service (Primary) and other services, let's say streaming TV (secondary)**

   - If the subscribers have Internet services (either DSL or Fiber optic), the subscribers may opt to have other services related to Internet (i.e. streaming TV, device protection).
   - But if the subscribers don't have the Internet services, this secondary service will not be available for the subscribers.

In [20]:

```
pd.crosstab(index = df["InternetService"], columns = df["StreamingTV"])
```

Out[20]:

StreamingTV	No	No internet service	Yes
InternetService			
DSL	1464	0	957
Fiber optic	1346	0	1750
No	0	1526	0

With this conclusion, I opt to transform the feature value of **No Phone / Internet service** to be the same **No** because it can be used another features (hence, **phone service** and **internet service** column) to explain.

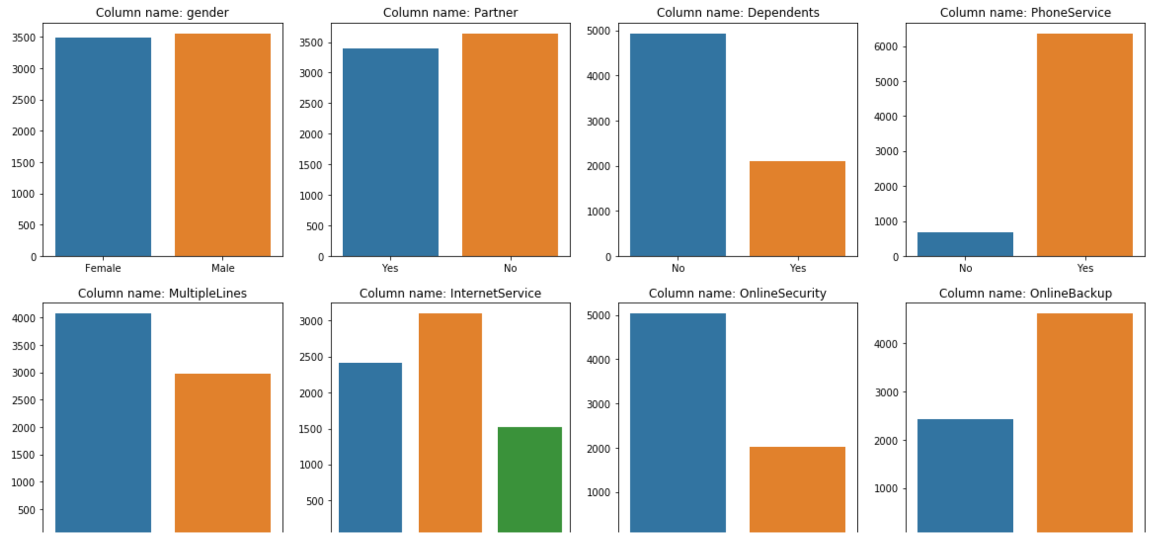
In [21]:

```
def convert_no_service (df):
    col_to_transform = []
    for col in df.columns:
        if (df[col].dtype == 'O') & (col != 'customerid'):
            if len(df[df[col].str.contains("No")][col].unique()) > 1:
                col_to_transform.append(col)

    print("Total column(s) to transform: {}".format(col_to_transform))
    for col in col_to_transform:
        df.loc[df[col].str.contains("No"), col] = 'No'

    return df
```

In [22]:

```
df = convert_no_service(df)

# Let's see the data after transformation.

display_plot(df, 'customerid', object_mode = True)
```

```
Total column(s) to transform: ['MultipleLines', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV', 'StreamingMovies']
```

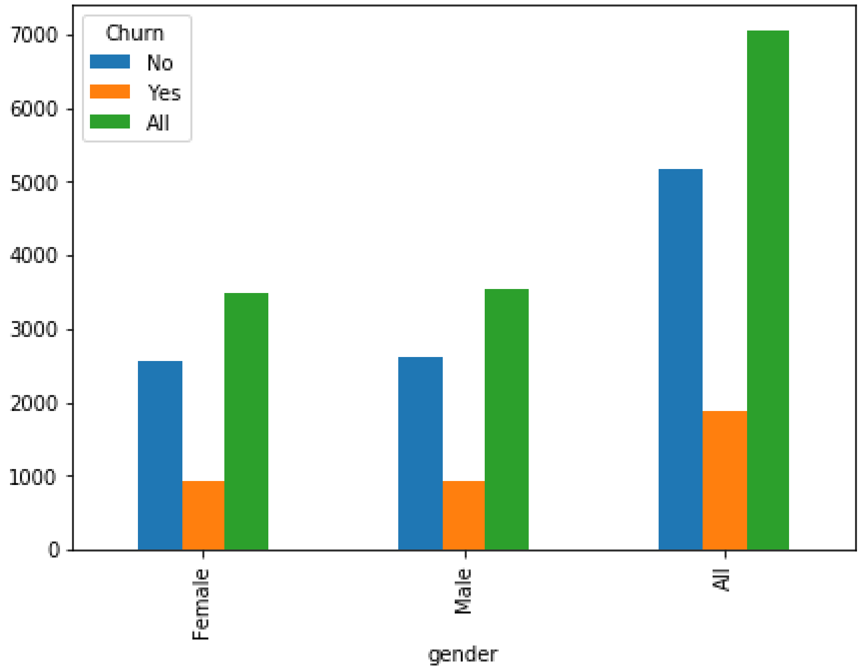


In [ ]:

In [23]:

```
# Now Let's Start Comparing.
# Gender Vs Churn
print(pd.crosstab(df.gender,df.Churn,margins=True))
pd.crosstab(df.gender,df.Churn,margins=True).plot(kind='bar',figsize=(7,5));

print('Percent of Females that Left the Company {0}'.format((939/1869)*100))
print('Percent of Males that Left the Company {0}'.format((930/1869)*100))
```

```
Churn     No   Yes   All
gender
Female  2549   939  3488
Male    2625   930  3555
All     5174  1869  7043
Percent of Females that Left the Company 50.24077046548957
Percent of Males that Left the Company 49.75922953451043
```

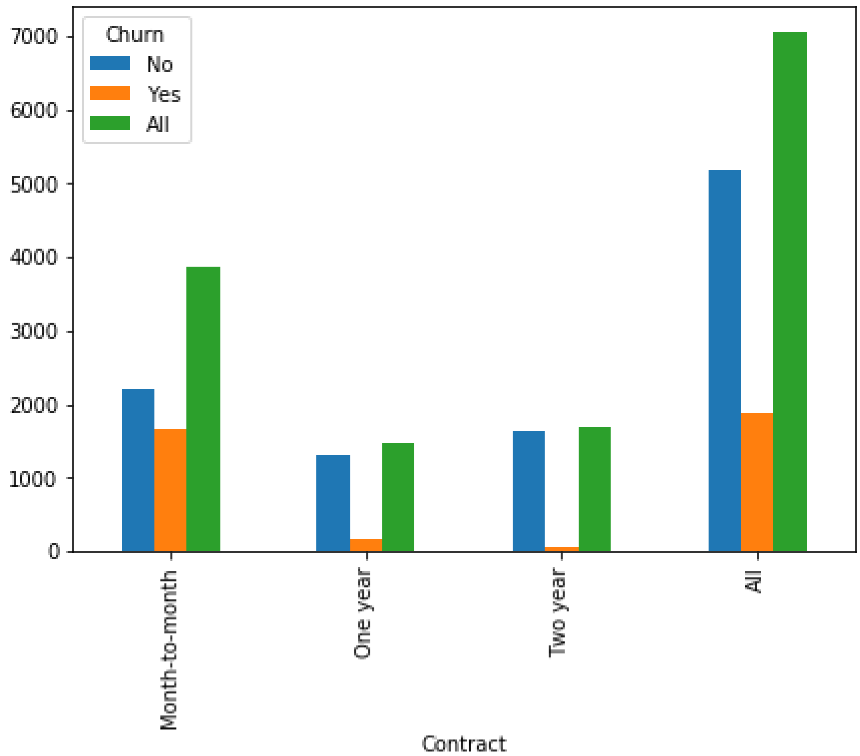


We can See that Gender Does'nt Play an important Role in Predicting Our Target Variable.

In [24]:

```
# Contract Vs Churn
print(pd.crosstab(df.Contract,df.Churn,margins=True))
pd.crosstab(df.Contract,df.Churn,margins=True).plot(kind='bar',figsize=(7,5));

print('Percent of Month-to-Month Contract People that Left the Company {0}'.format((1655/18
print('Percent of One-Year Contract People that Left the Company {0}'.format((166/1869)*100
print('Percent of Two-Year Contract People that Left the Company {0}'.format((48/1869)*100)
```

```
Churn             No   Yes   All
Contract
Month-to-month  2220  1655  3875
One year        1307   166  1473
Two year        1647    48  1695
All             5174  1869  7043
Percent of Month-to-Month Contract People that Left the Company 88.550026752
27395
Percent of One-Year Contract People that Left the Company 8.881754949170679
Percent of Two-Year Contract People that Left the Company 2.568218298555377
```

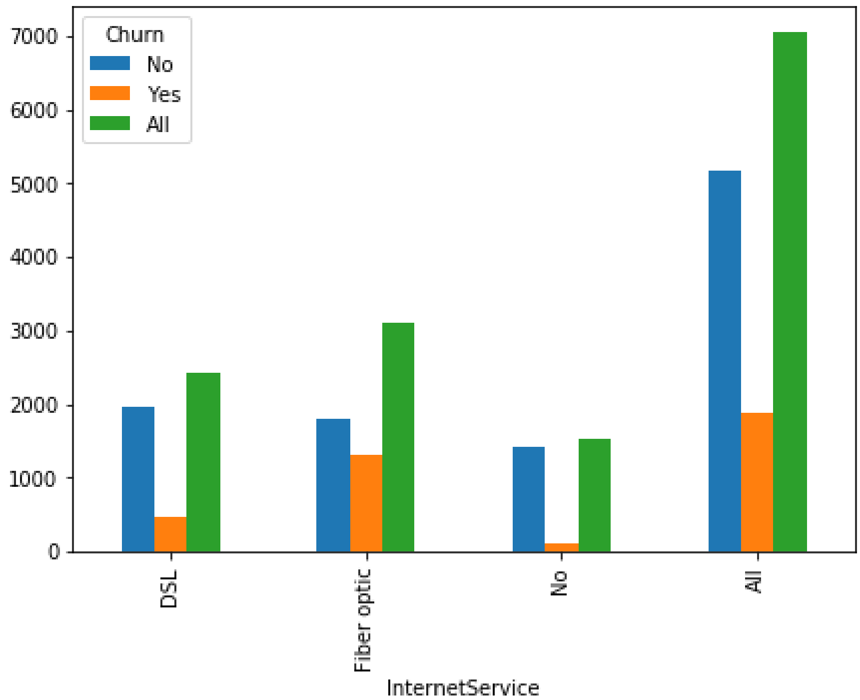


Most of the People that Left were the Ones who had Month-to-Month Contract.

In [25]:

```
# Internet Service Vs Churn
print(pd.crosstab(df.InternetService,df.Churn,margins=True))
pd.crosstab(df.InternetService,df.Churn,margins=True).plot(kind='bar',figsize=(7,5));

print('Percent of DSL Internet-Service People that Left the Company {0}'.format((459/1869)*
print('Percent of Fiber Optic Internet-Service People that Left the Company {0}'.format((12
print('Percent of No Internet-Service People that Left the Company {0}'.format((113/1869)*1
```

```
Churn              No   Yes   All
InternetService
DSL              1962   459  2421
Fiber optic      1799  1297  3096
No               1413   113  1526
All              5174  1869  7043
Percent of DSL Internet-Service People that Left the Company 24.558587479935
795
Percent of Fiber Optic Internet-Service People that Left the Company 69.3953
9860888175
Percent of No Internet-Service People that Left the Company 6.04601391118245
1
```

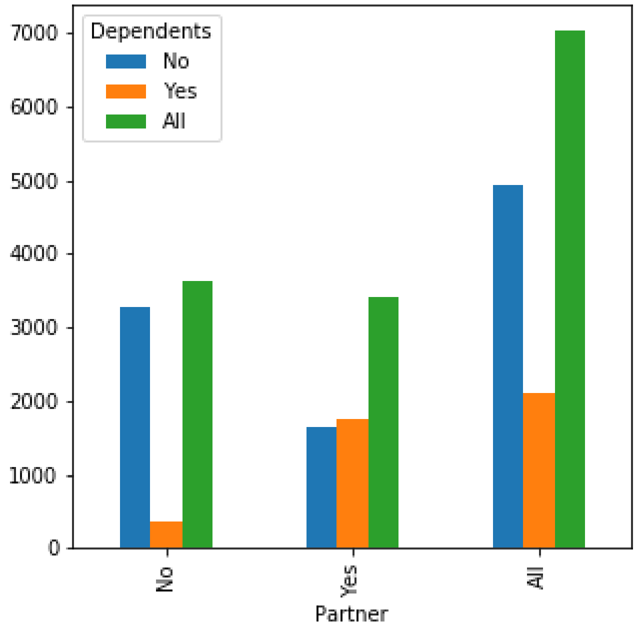


Most of the people That Left had Fiber Optic Internet-Service.

In [26]:

```
# Partner Vs Dependents
print(pd.crosstab(df.Partner,df.Dependents,margins=True))
pd.crosstab(df.Partner,df.Dependents,margins=True).plot(kind='bar',figsize=(5,5));

print('Percent of Partner that had Dependents {0}'.format((1749/2110)*100))
print('Percent of Non-Partner that had Dependents {0}'.format((361/2110)*100))
```

```
Dependents    No   Yes   All
Partner
No          3280   361  3641
Yes         1653  1749  3402
All         4933  2110  7043
Percent of Partner that had Dependents 82.8909952606635
Percent of Non-Partner that had Dependents 17.10900473933649
```

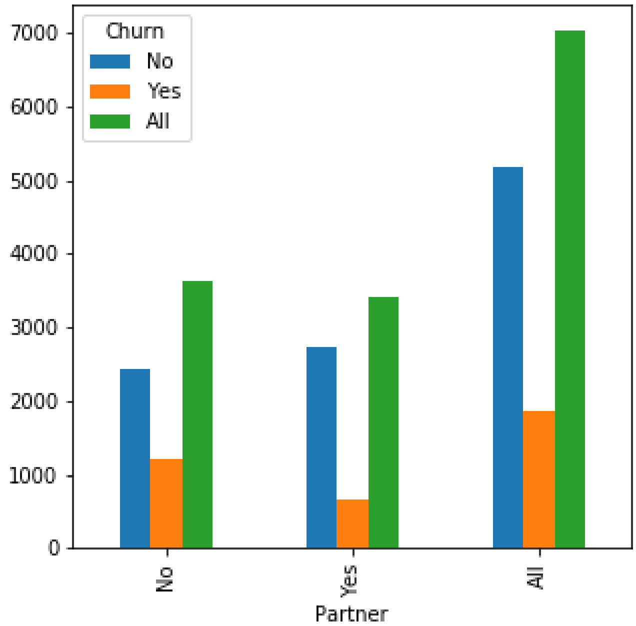


We can See Partners had a much larger percent of Dependents than Non-Partner this tells us that Most Partners might be Married.

In [27]:

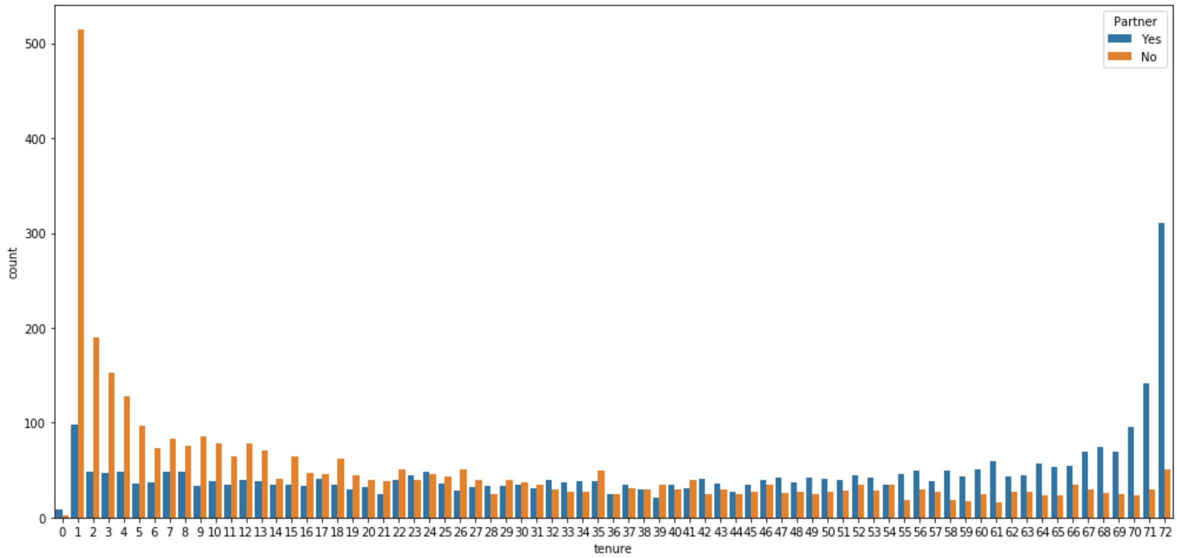
```
# Partner Vs Churn
print(pd.crosstab(df.Partner,df.Churn,margins=True))
pd.crosstab(df.Partner,df.Churn,margins=True).plot(kind='bar',figsize=(5,5));
```

```
Churn       No   Yes   All
Partner
No        2441  1200  3641
Yes       2733   669  3402
All       5174  1869  7043
```



In [28]:

```
plt.figure(figsize=(17,8))
sns.countplot(x=df['tenure'],hue=df.Partner);
```

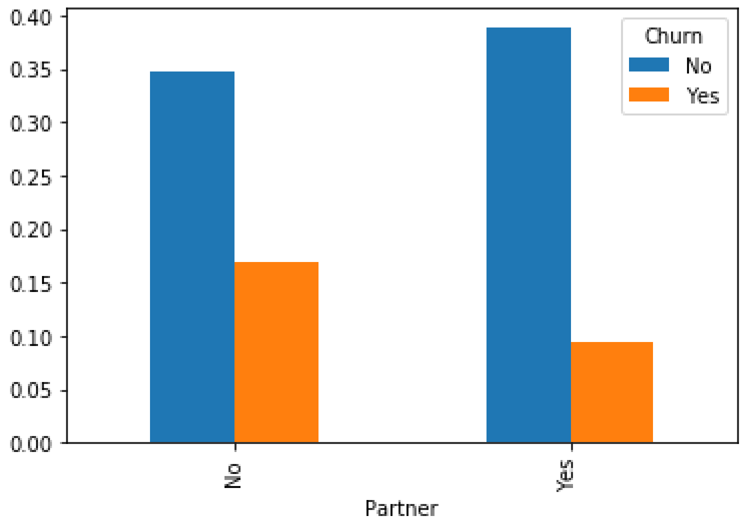


Most of the People that Were Partner will Stay Longer with The Company. So Being a Partner is a Plus-Point For the Company as they will Stay Longer with Them.

In [29]:

```
# Partner Vs Churn
print(pd.crosstab(df.Partner,df.Churn,margins=True))
pd.crosstab(df.Partner,df.Churn,normalize=True).plot(kind='bar');
```

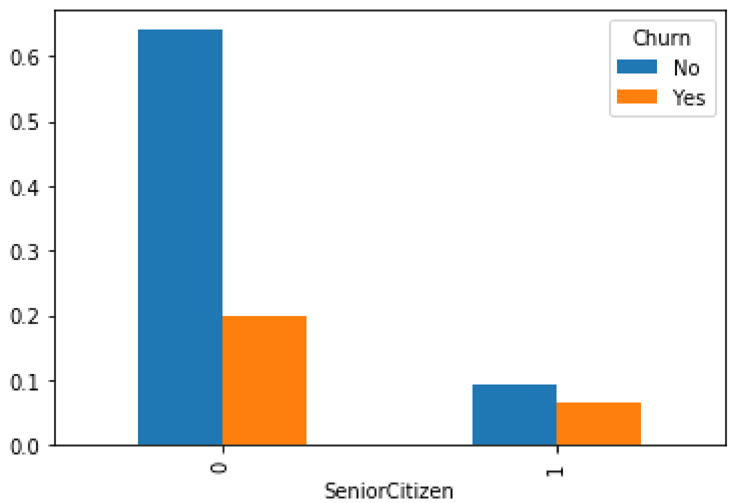
```
Churn      No   Yes   All
Partner
No       2441  1200  3641
Yes      2733   669  3402
All      5174  1869  7043
```



In [30]:

```
# Senior Citizen Vs Churn
print(pd.crosstab(df.SeniorCitizen,df.Churn,margins=True))
pd.crosstab(df.SeniorCitizen,df.Churn,normalize=True).plot(kind='bar');
```

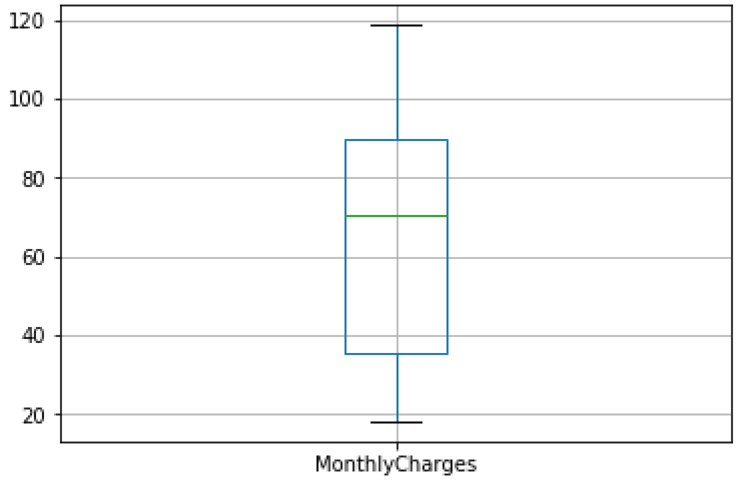
```
Churn            No   Yes   All
SeniorCitizen
0              4508  1393  5901
1               666   476  1142
All            5174  1869  7043
```



Let's Check for Outliers in Monthly Charges And Total Charges Using Box Plots

In [31]:

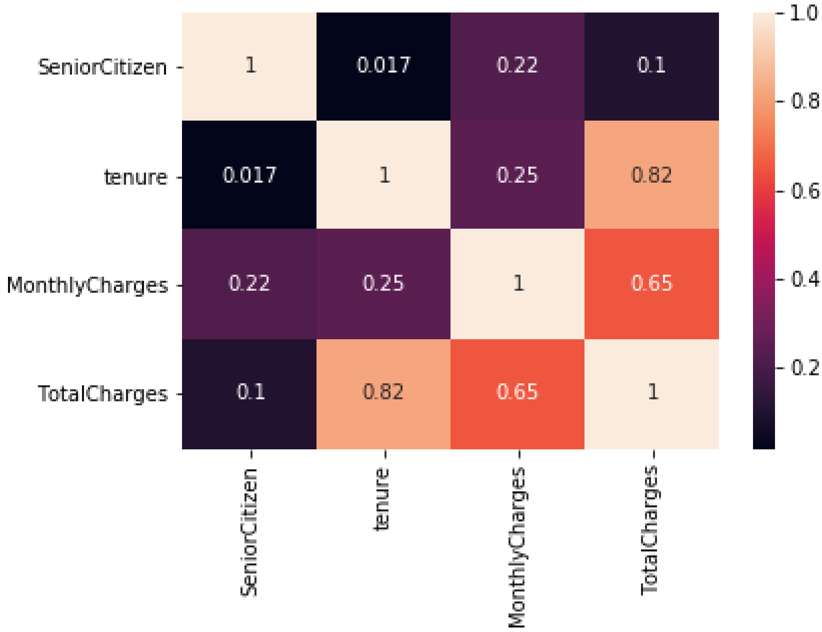
```
df.boxplot('MonthlyCharges');
```



Monthly Charges don't have any Outliers so we don't have to Get into Extracting Information from Outliers.

In [32]:

```
## correlation matrix

# Let's Check the Correaltion Matrix in Seaborn
sns.heatmap(df.corr(),xticklabels=df.corr().columns.values,yticklabels=df.corr().columns.va
```



Here We can See Tenure and Total Charges are correlated and also Monthly charges and Total Charges are also correlated with each other.

we can assume from our domain expertise that , Total Charges ~ Monthly Charges * Tenure + Additional Charges(Tax).

Bucketing

In [34]:

```
#Tenure to categorical column
def tenure_lab(telcom) :

    if telcom["tenure"] <= 12 :
        return "Tenure_0-12"
    elif (telcom["tenure"] > 12) & (telcom["tenure"] <= 24 ):
        return "Tenure_12-24"
    elif (telcom["tenure"] > 24) & (telcom["tenure"] <= 48) :
        return "Tenure_24-48"
    elif (telcom["tenure"] > 48) & (telcom["tenure"] <= 60) :
        return "Tenure_48-60"
    elif telcom["tenure"] > 60 :
        return "Tenure_gt_60"


df["tenure_group"] = df.apply(lambda x:tenure_lab(x),axis = 1)
```

10. Data preprocessing

Encoding categorical variable

In [35]:

```
#replace values
df["SeniorCitizen"] = df["SeniorCitizen"].replace({1:"Yes",0:"No"})
```

In [36]:

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

#customer id col
Id_col     = ['customerID']
#Target columns
target_col = ["Churn"]


#categorical columns
cat_cols   = df.nunique()[df.nunique() < 6].keys().tolist()
cat_cols   = [x for x in cat_cols if x not in target_col]
#numerical columns
num_cols   = [x for x in df.columns if x not in cat_cols + target_col + Id_col]
#Binary columns with 2 values
bin_cols   = df.nunique()[df.nunique() == 2].keys().tolist()
#Columns more than 2 values
multi_cols = [i for i in cat_cols if i not in bin_cols]

#Label encoding Binary columns
le = LabelEncoder()
for i in bin_cols :
    df[i] = le.fit_transform(df[i])

#Duplicating columns for multi value columns
df = pd.get_dummies(data = df,columns = multi_cols )
```

Normalizing features

In [37]:

```
telcom = df
#Scaling Numerical columns
std = StandardScaler()
scaled = std.fit_transform(telcom[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)

#dropping original values merging scaled values for numerical columns
df_telcom_og = telcom.copy()
telcom = telcom.drop(columns = num_cols,axis = 1)
telcom = telcom.merge(scaled,left_index=True,right_index=True,how = "left")
```

```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/p
reprocessing/data.py:617: DataConversionWarning: Data with input dtype int6
4, float64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/b
ase.py:462: DataConversionWarning: Data with input dtype int64, float64 were
all converted to float64 by StandardScaler.
  return self.fit(X, **fit_params).transform(X)
```

spliting train/val/test data

In [38]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from sklearn.metrics import roc_auc_score,roc_curve,scorer
from sklearn.metrics import f1_score
import statsmodels.api as sm
from sklearn.metrics import precision_score,recall_score
from yellowbrick.classifier import DiscriminationThreshold
#splitting train and test data

telcom = df
# target_col = telcom["Churn"]

train,test = train_test_split(telcom,test_size = .25 ,random_state = 111)

##seperating dependent and independent variables
cols    = [i for i in telcom.columns if i not in  target_col]
X_train = train[cols]
y_train = train["Churn"]
X_test  = test[cols]
y_test  = test["Churn"]
```

11. Model Building

In [39]:

```
from sklearn.dummy import DummyClassifier

# Feature Selection and Encoding
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize

# Machine learning
from sklearn import tree , linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, Ridge, Lasso, SGDCla
from sklearn.tree import DecisionTreeClassifier
from xgboost.sklearn import XGBClassifier
```

In [40]:

```
# validation
from sklearn import datasets, model_selection, metrics , preprocessing
```

In [57]:

```
# Grid and Random Search
import scipy.stats as st
from scipy.stats import randint as sp_randint
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
```

In [42]:

```
# Metrics
from sklearn.metrics import precision_recall_fscore_support, roc_curve, auc
```

In [43]:

```
#utilities
import time
import io, os, sys, types, time, datetime, math, random
```

In [47]:

```
# calculate the fpr and tpr for all thresholds of the classification
def plot_roc_curve(y_test, preds):
    fpr, tpr, threshold = metrics.roc_curve(y_test, preds)
    roc_auc = metrics.auc(fpr, tpr)
    plt.title('Receiver Operating Characteristic')
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([-0.01, 1.01])
    plt.ylim([-0.01, 1.01])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.show()



# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, X_test, cv):
    # One Pass
    model = algo.fit(X_train, y_train)
    test_pred = model.predict(X_test)
    if (isinstance(algo, (LogisticRegression,
                          KNeighborsClassifier,
                          GaussianNB,
                          DecisionTreeClassifier,
                          RandomForestClassifier,
                          GradientBoostingClassifier))):
        probs = model.predict_proba(X_test)[:,1]
    else:
        probs = "Not Available"
    acc = round(model.score(X_test, y_test) * 100, 2)
    # CV
    train_pred = model_selection.cross_val_predict(algo,
                                                   X_train,
                                                   y_train,
                                                   cv=cv,
                                                   n_jobs = -1)
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)
    return train_pred, test_pred, acc, acc_cv, probs

# Utility function to report best scores
def report(results, n_top=5):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

Baseline model with DummyClassifier

In [48]:

```
clf = DummyClassifier(strategy='most_frequent',random_state=0)
clf.fit(X_train, y_train)
```

Out[48]:

```
DummyClassifier(constant=None, random_state=0, strategy='most_frequent')
```

In [49]:

```
accuracy = clf.score(X_test, y_test)
accuracy
```

Out[49]:

```
0.7535491198182851
```

In [50]:

```
preds = clf.predict(X_test)


# dummyistic Regression
start_time = time.time()
train_pred_dummy, test_pred_dummy, acc_dummy, acc_cv_dummy, probs_dummy = fit_ml_algo(Dummy
                                                                    X_train,
                                                                    y_train,
                                                                    X_test,
                                                                    10)
dummy_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dummy)
print("Accuracy CV 10-Fold: %s" % acc_cv_dummy)
print("Running Time: %s" % datetime.timedelta(seconds=dummy_time))

print (metrics.classification_report(y_train, train_pred_dummy))

print (metrics.classification_report(y_test, test_pred_dummy))
```

```
Accuracy: 75.35
Accuracy CV 10-Fold: 72.83
Running Time: 0:00:03.575734
              precision    recall  f1-score   support

           0       0.73      1.00      0.84      3847
           1       0.00      0.00      0.00      1435

   micro avg       0.73      0.73      0.73      5282
   macro avg       0.36      0.50      0.42      5282
weighted avg       0.53      0.73      0.61      5282

              precision    recall  f1-score   support

           0       0.75      1.00      0.86      1327
           1       0.00      0.00      0.00       434

   micro avg       0.75      0.75      0.75      1761
   macro avg       0.38      0.50      0.43      1761
weighted avg       0.57      0.75      0.65      1761


/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
etrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score
are ill-defined and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Select Candidate Algorithms

1. KNN

2. Logistic Regression

3. Random Forest

4. Naive Bayes

5. Stochastic Gradient Decent

6. Linear SVC

7. Decision Tree

8. Gradient Boosted Trees

In [51]:

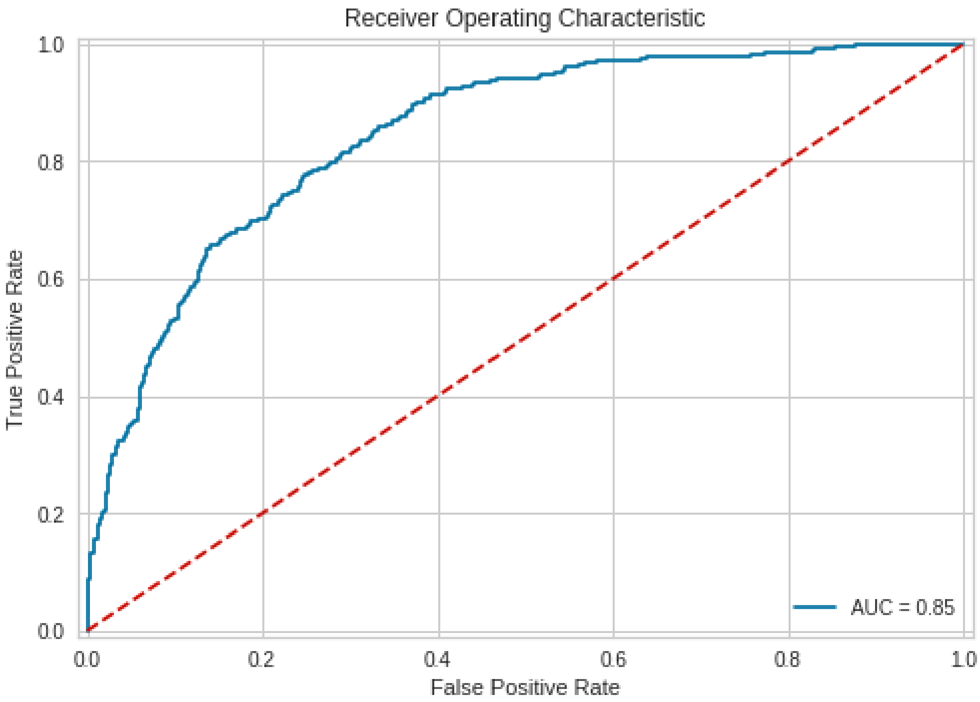
```
# Specify parameters and distributions to sample from
param_dist = {'penalty': ['l2', 'l1'],
                         'class_weight': [None, 'balanced'],
                         'C': np.logspace(-20, 20, 10000),
                         'intercept_scaling': np.logspace(-20, 20, 10000)}



# Run Randomized Search
n_iter_search = 10
lrc = LogisticRegression()
random_search = RandomizedSearchCV(lrc,
                                   n_jobs=-1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(random_search.cv_results_)
```

```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
odel_selection/_split.py:1943: FutureWarning: You should specify a value for
'cv' instead of relying on the default value. The default value will change
from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)

RandomizedSearchCV took 2.69 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.801 (std: 0.001)
Parameters: {'penalty': 'l2', 'intercept_scaling': 0.00033857350174073126,
'class_weight': None, 'C': 0.015624976827451342}

Model with rank: 2
Mean validation score: 0.797 (std: 0.006)
Parameters: {'penalty': 'l1', 'intercept_scaling': 6.798032528158685e-17, 'c
lass_weight': None, 'C': 86.73488747257058}

Model with rank: 3
Mean validation score: 0.797 (std: 0.005)
Parameters: {'penalty': 'l1', 'intercept_scaling': 9.497247583784531e-05, 'c
lass_weight': None, 'C': 131556378962.65248}

Model with rank: 4
Mean validation score: 0.796 (std: 0.004)
Parameters: {'penalty': 'l1', 'intercept_scaling': 210296615.49651128, 'clas
s_weight': None, 'C': 677927292345.3245}

Model with rank: 5
Mean validation score: 0.744 (std: 0.003)
Parameters: {'penalty': 'l2', 'intercept_scaling': 1.1967620273057582, 'clas
s_weight': 'balanced', 'C': 15503.11761737585}


/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/l
inear_model/logistic.py:432: FutureWarning: Default solver will be changed t
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

In [52]:

```
# Logistic Regression
start_time = time.time()
train_pred_log, test_pred_log, acc_log, acc_cv_log, probs_log = fit_ml_algo(LogisticRegress
                                                                              X_train,
                                                                              y_train,
                                                                              X_test,
                                                                              10)
log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))

print (metrics.classification_report(y_train, train_pred_log))

print (metrics.classification_report(y_test, test_pred_log))

plot_roc_curve(y_test, probs_log)
```

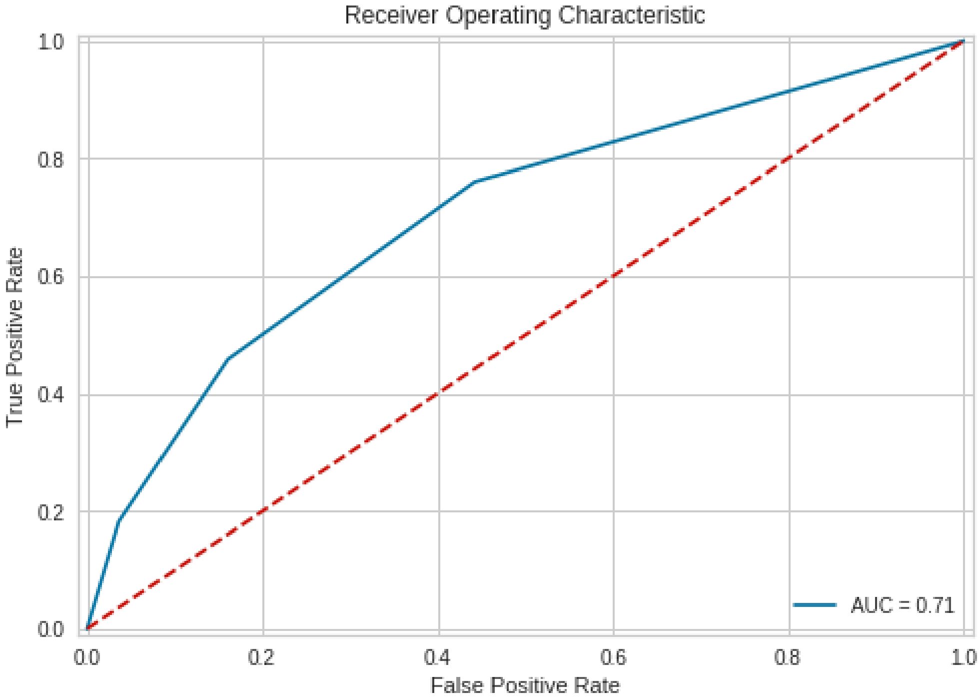
```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/l
inear_model/logistic.py:432: FutureWarning: Default solver will be changed t
o 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/l
inear_model/logistic.py:1296: UserWarning: 'n_jobs' > 1 does not have any ef
fect when 'solver' is set to 'liblinear'. Got 'n_jobs' = 12.
  " = {}.".format(effective_n_jobs(self.n_jobs)))

Accuracy: 80.86
Accuracy CV 10-Fold: 80.1
Running Time: 0:00:00.576369
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      3847
           1       0.67      0.53      0.59      1435

   micro avg       0.80      0.80      0.80      5282
   macro avg       0.75      0.72      0.73      5282
weighted avg       0.79      0.80      0.79      5282

              precision    recall  f1-score   support

           0       0.86      0.89      0.88      1327
           1       0.62      0.56      0.59       434

   micro avg       0.81      0.81      0.81      1761
   macro avg       0.74      0.73      0.73      1761
weighted avg       0.80      0.81      0.81      1761
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.85
0.0
0.2
0.4
0.6
0.8
1.0

In [53]:

```
# k-Nearest Neighbors
start_time = time.time()
train_pred_knn, test_pred_knn, acc_knn, acc_cv_knn, probs_knn = fit_ml_algo(KNeighborsClass

knn_time = (time.time() - start_time)
print("Accuracy: %s" % acc_knn)
print("Accuracy CV 10-Fold: %s" % acc_cv_knn)
print("Running Time: %s" % datetime.timedelta(seconds=knn_time))

print (metrics.classification_report(y_train, train_pred_knn))

print (metrics.classification_report(y_test, test_pred_knn))

plot_roc_curve(y_test, probs_knn)
```

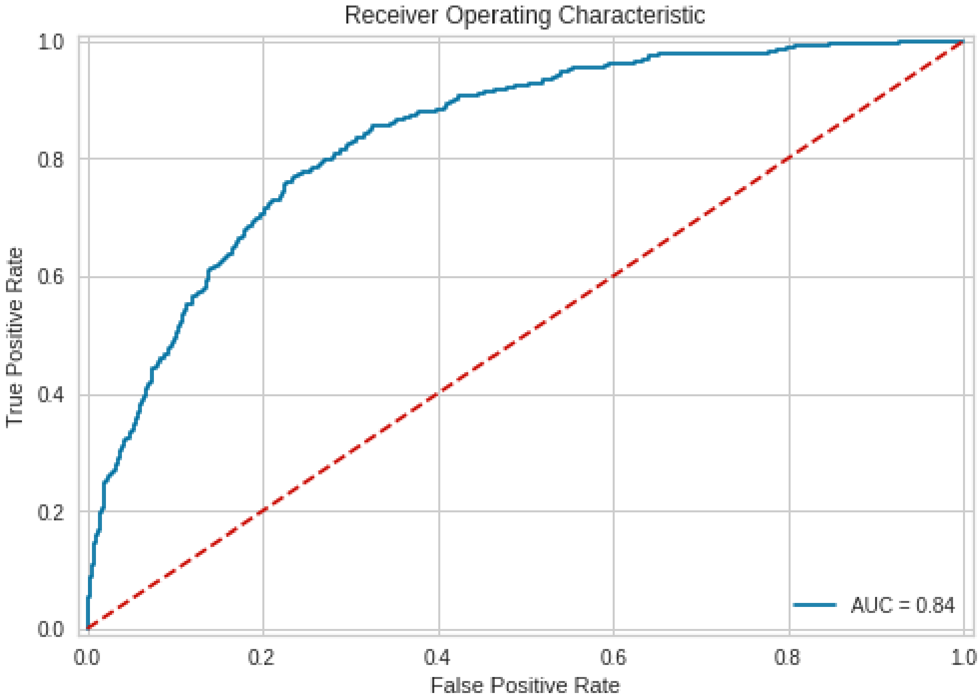
```
Accuracy: 74.56
Accuracy CV 10-Fold: 74.93
Running Time: 0:00:00.601969
              precision    recall  f1-score   support

           0       0.81      0.86      0.83      3847
           1       0.55      0.46      0.50      1435

   micro avg       0.75      0.75      0.75      5282
   macro avg       0.68      0.66      0.67      5282
weighted avg       0.74      0.75      0.74      5282

              precision    recall  f1-score   support

           0       0.83      0.84      0.83      1327
           1       0.48      0.46      0.47       434

   micro avg       0.75      0.75      0.75      1761
   macro avg       0.65      0.65      0.65      1761
weighted avg       0.74      0.75      0.74      1761
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.71
0.0
0.2
0.4
0.6
0.8
1.0

In [54]:

```
# Gaussian Naive Bayes
start_time = time.time()
train_pred_gaussian, test_pred_gaussian, acc_gaussian, acc_cv_gaussian, probs_gau = fit_ml_
                                                                                    X_trai
                                                                                    y_trai
                                                                                    X_test
                                                                                    10)
gaussian_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gaussian)
print("Accuracy CV 10-Fold: %s" % acc_cv_gaussian)
print("Running Time: %s" % datetime.timedelta(seconds=gaussian_time))

print (metrics.classification_report(y_train, train_pred_gaussian))

print (metrics.classification_report(y_test, test_pred_gaussian))

plot_roc_curve(y_test, probs_gau)
```

```
Accuracy: 73.65
Accuracy CV 10-Fold: 74.67
Running Time: 0:00:00.113730
              precision    recall  f1-score   support

           0       0.90      0.73      0.81      3847
           1       0.52      0.78      0.63      1435

   micro avg       0.75      0.75      0.75      5282
   macro avg       0.71      0.76      0.72      5282
weighted avg       0.80      0.75      0.76      5282

              precision    recall  f1-score   support

           0       0.92      0.71      0.80      1327
           1       0.48      0.81      0.60       434

   micro avg       0.74      0.74      0.74      1761
   macro avg       0.70      0.76      0.70      1761
weighted avg       0.81      0.74      0.75      1761
```

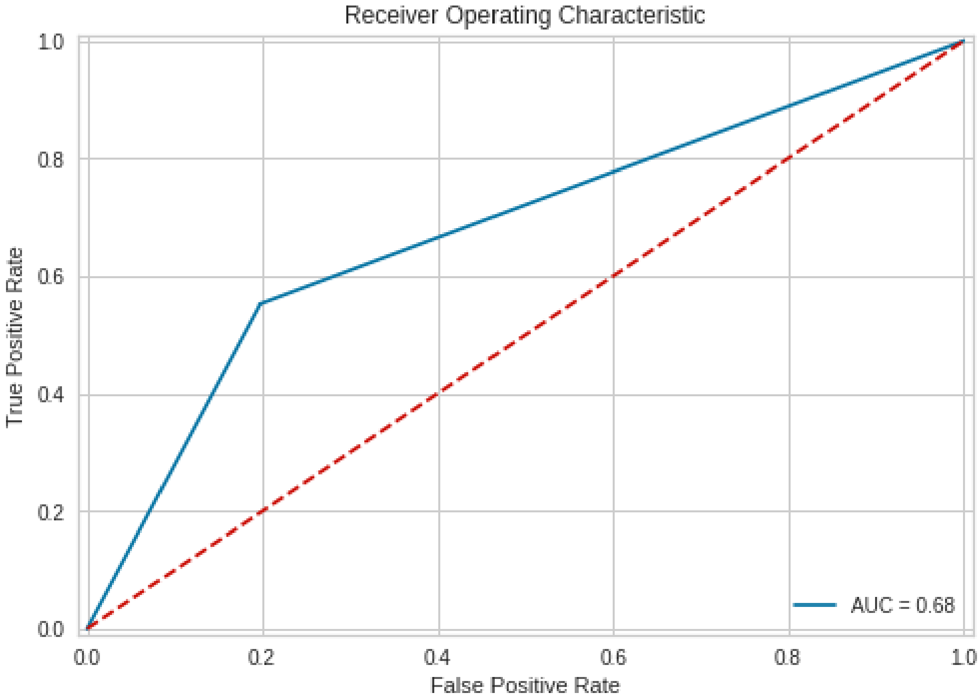


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.84
0.0
0.2
0.4
0.6
0.8
1.0

In [55]:

```
# Decision Tree Classifier
start_time = time.time()
train_pred_dt, test_pred_dt, acc_dt, acc_cv_dt, probs_dt = fit_ml_algo(DecisionTreeClassifi
                                                              X_train,
                                                              y_train,
                                                              X_test,
                                                              10)
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)
print("Accuracy CV 10-Fold: %s" % acc_cv_dt)
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))

print (metrics.classification_report(y_train, train_pred_dt))

print (metrics.classification_report(y_test, test_pred_dt))

plot_roc_curve(y_test, probs_dt)
```

```
Accuracy: 74.11
Accuracy CV 10-Fold: 71.53
Running Time: 0:00:00.152194
              precision    recall  f1-score   support

           0       0.81      0.80      0.80      3847
           1       0.48      0.49      0.48      1435

   micro avg       0.72      0.72      0.72      5282
   macro avg       0.64      0.64      0.64      5282
weighted avg       0.72      0.72      0.72      5282

              precision    recall  f1-score   support

           0       0.84      0.80      0.82      1327
           1       0.48      0.55      0.51       434

   micro avg       0.74      0.74      0.74      1761
   macro avg       0.66      0.68      0.67      1761
weighted avg       0.75      0.74      0.75      1761
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.68

In [58]:

```
# Random Forest Classifier - Random Search for Hyperparameters

# Utility function to report best scores
def report(results, n_top=5):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")

# Specify parameters and distributions to sample from
param_dist = {"max_depth": [10, None],
              "max_features": sp_randint(1, 11),
              "min_samples_split": sp_randint(2, 20),
              "min_samples_leaf": sp_randint(1, 11),
              "bootstrap": [True, False],
              "criterion": ["gini", "entropy"]}


# Run Randomized Search
n_iter_search = 10
rfc = RandomForestClassifier(n_estimators=10)
random_search = RandomizedSearchCV(rfc,
                                   n_jobs = -1,
                                   param_distributions=param_dist,
                                   n_iter=n_iter_search)

start = time.time()
random_search.fit(X_train, y_train)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time.time() - start), n_iter_search))
report(random_search.cv_results_)
```

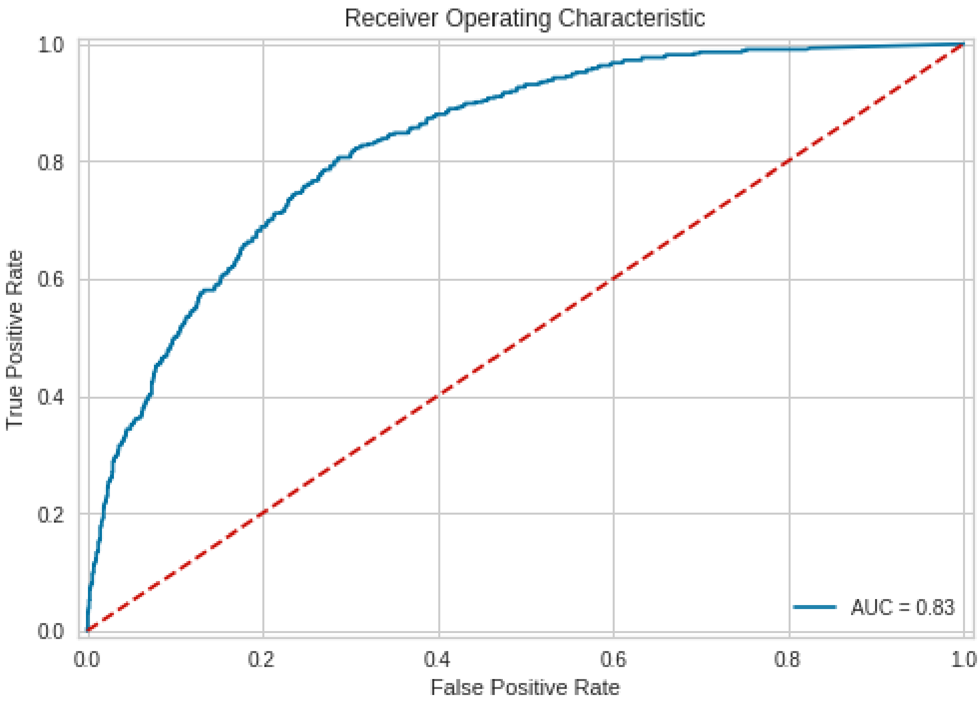
```
/home/ubuntu/.virtualenvs/Data_Science/lib/python3.6/site-packages/sklearn/m
odel_selection/_split.py:1943: FutureWarning: You should specify a value for
'cv' instead of relying on the default value. The default value will change
from 3 to 5 in version 0.22.
  warnings.warn(CV_WARNING, FutureWarning)

RandomizedSearchCV took 0.87 seconds for 10 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.798 (std: 0.006)
Parameters: {'bootstrap': True, 'criterion': 'gini', 'max_depth': 10, 'max_f
eatures': 4, 'min_samples_leaf': 8, 'min_samples_split': 16}

Model with rank: 2
Mean validation score: 0.798 (std: 0.003)
Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': None, 'ma
x_features': 3, 'min_samples_leaf': 8, 'min_samples_split': 6}

Model with rank: 3
Mean validation score: 0.796 (std: 0.001)
Parameters: {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 10, 'm
ax_features': 6, 'min_samples_leaf': 7, 'min_samples_split': 10}
```

```
Model with rank: 4
Mean validation score: 0.795 (std: 0.002)
Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': 10, 'max_
features': 6, 'min_samples_leaf': 3, 'min_samples_split': 13}

Model with rank: 5
Mean validation score: 0.794 (std: 0.001)
Parameters: {'bootstrap': False, 'criterion': 'gini', 'max_depth': 10, 'max_
features': 5, 'min_samples_leaf': 9, 'min_samples_split': 9}
```

In [59]:

```
# Random Forest Classifier
start_time = time.time()
rfc = RandomForestClassifier(n_estimators=10,
                             min_samples_leaf=2,
                             min_samples_split=17,
                             criterion='gini',
                             max_features=8)
train_pred_rf, test_pred_rf, acc_rf, acc_cv_rf, probs_rf = fit_ml_algo(rfc,
                                                              X_train,
                                                              y_train,
                                                              X_test,
                                                              10)
rf_time = (time.time() - start_time)
print("Accuracy: %s" % acc_rf)
print("Accuracy CV 10-Fold: %s" % acc_cv_rf)
print("Running Time: %s" % datetime.timedelta(seconds=rf_time))

print (metrics.classification_report(y_train, train_pred_rf))

print (metrics.classification_report(y_test, test_pred_rf))

plot_roc_curve(y_test, probs_rf)
```

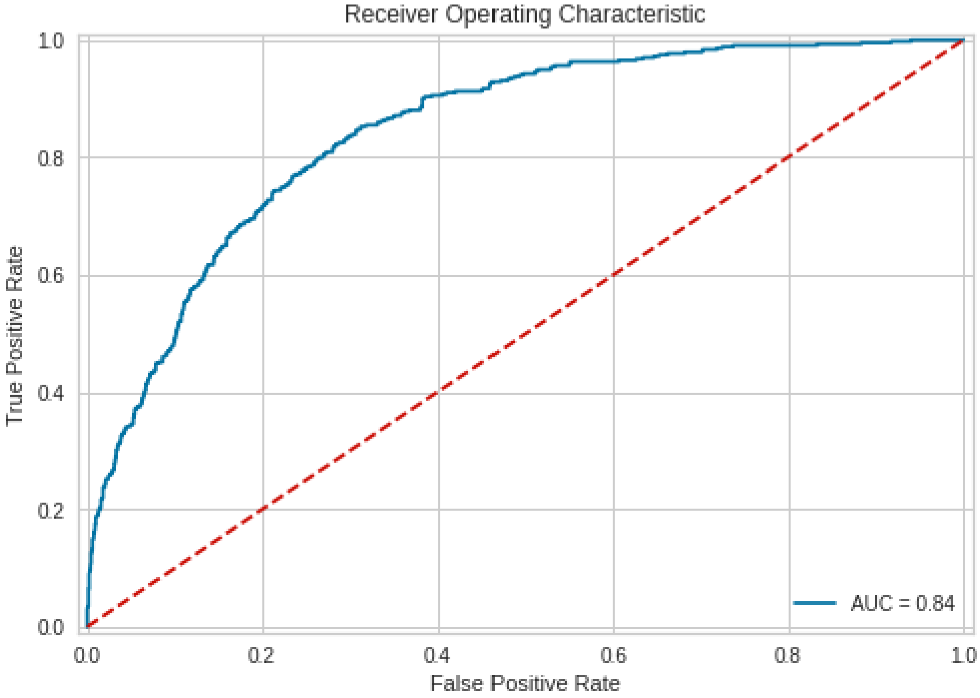
```
Accuracy: 80.01
Accuracy CV 10-Fold: 78.66
Running Time: 0:00:00.250799
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      3847
           1       0.63      0.51      0.57      1435

   micro avg       0.79      0.79      0.79      5282
   macro avg       0.73      0.70      0.71      5282
weighted avg       0.78      0.79      0.78      5282

              precision    recall  f1-score   support

           0       0.85      0.89      0.87      1327
           1       0.61      0.53      0.57       434

   micro avg       0.80      0.80      0.80      1761
   macro avg       0.73      0.71      0.72      1761
weighted avg       0.79      0.80      0.80      1761
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
AUC = 0.83

In [60]:

```
# Gradient Boosting Trees
start_time = time.time()
train_pred_gbt, test_pred_gbt, acc_gbt, acc_cv_gbt, probs_gbt = fit_ml_algo(GradientBoostir
                                                                 X_train,
                                                                 y_train,
                                                                 X_test,
                                                                 10)
gbt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_gbt)
print("Accuracy CV 10-Fold: %s" % acc_cv_gbt)
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))

print (metrics.classification_report(y_train, train_pred_gbt))

print (metrics.classification_report(y_test, test_pred_gbt))

plot_roc_curve(y_test, probs_gbt)
```

```
Accuracy: 80.58
Accuracy CV 10-Fold: 80.18
Running Time: 0:00:02.742447
              precision    recall  f1-score   support

           0       0.84      0.90      0.87      3847
           1       0.67      0.53      0.59      1435

   micro avg       0.80      0.80      0.80      5282
   macro avg       0.75      0.72      0.73      5282
weighted avg       0.79      0.80      0.79      5282

              precision    recall  f1-score   support

           0       0.86      0.89      0.87      1327
           1       0.62      0.55      0.58       434

   micro avg       0.81      0.81      0.81      1761
   macro avg       0.74      0.72      0.73      1761
weighted avg       0.80      0.81      0.80      1761
```


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
AUC = 0.84
0.0
0.2
0.4
0.6
0.8
1.0

In [61]:

```
def xgb_f1(y, t):
    #
    # Function to evaluate the prediction based on F1 score, this will be used as evaluatic
    # Args:
    #   y: label
    #   t: predicted
    #
    # Return:
    #   f1: F1 score of the actual and predicted
    #
    t = t.get_label()
    y_bin = [1. if y_cont > 0.5 else 0. for y_cont in y]   # change the prob to class outpu
    return 'f1', f1_score(t, y_bin)

best_xgb = XGBClassifier(objective = 'binary:logistic',
                         colsample_bylevel = 0.7,
                         colsample_bytree = 0.8,
                         gamma = 1,
                         learning_rate = 0.15,
                         max_delta_step = 3,
                         max_depth = 4,
                         min_child_weight = 1,
                         n_estimators = 50,
                         reg_lambda = 10,
                         scale_pos_weight = 1.5,
                         subsample = 0.9,
                         silent = False,
                         n_jobs = 4
                        )

xgbst = best_xgb.fit(X_train, y_train, eval_metric = xgb_f1, eval_set = [(X_train, y_train)
             early_stopping_rounds = 20)
```

```
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[0]     validation_0-error:0.216585     validation_1-error:0.236229     vali
dation_0-f1:0.642053    validation_1-f1:0.597679
Multiple eval metrics have been passed: 'validation_1-f1' will be used for e
arly stopping.

Will train until validation_1-f1 hasn't improved in 20 rounds.
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 0 pruned nodes, max_depth=4
[1]     validation_0-error:0.217531     validation_1-error:0.242476     vali
dation_0-f1:0.6435      validation_1-f1:0.594492
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 0 pruned nodes, max_depth=4
[2]     validation_0-error:0.216585     validation_1-error:0.236229     vali
dation_0-f1:0.642053    validation_1-f1:0.597679
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 16 extra nodes, 0 pruned nodes, max_depth=4
[3]     validation_0-error:0.216395     validation_1-error:0.235662     vali
dation_0-f1:0.642254    validation_1-f1:0.599034
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 0 pruned nodes, max_depth=4
[4]     validation_0-error:0.211852     validation_1-error:0.227712     vali
dation_0-f1:0.642606    validation_1-f1:0.603363
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
```

```
[5]	validation_0-error:0.213177	validation_1-error:0.230551	vali
dation_0-f1:0.643896	validation_1-f1:0.601179
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[6]	validation_0-error:0.211094	validation_1-error:0.228847	vali
dation_0-f1:0.643884	validation_1-f1:0.598205
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 0 pruned nodes, max_depth=4
[7]	validation_0-error:0.205604	validation_1-error:0.218058	vali
dation_0-f1:0.639681	validation_1-f1:0.604124
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 0 pruned nodes, max_depth=4
[8]	validation_0-error:0.205415	validation_1-error:0.219194	vali
dation_0-f1:0.641559	validation_1-f1:0.604508
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[9]	validation_0-error:0.204468	validation_1-error:0.214651	vali
dation_0-f1:0.640957	validation_1-f1:0.609504
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[10]	validation_0-error:0.204279	validation_1-error:0.210676	vali
dation_0-f1:0.638768	validation_1-f1:0.61233
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
[11]	validation_0-error:0.202764	validation_1-error:0.211811	vali
dation_0-f1:0.643594	validation_1-f1:0.612669
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 4 pruned nodes, max_depth=4
[12]	validation_0-error:0.202953	validation_1-error:0.210108	vali
dation_0-f1:0.643854	validation_1-f1:0.614583
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 4 pruned nodes, max_depth=4
[13]	validation_0-error:0.201817	validation_1-error:0.20954	vali
dation_0-f1:0.645376	validation_1-f1:0.61442
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[14]	validation_0-error:0.200492	validation_1-error:0.207836	vali
dation_0-f1:0.645464	validation_1-f1:0.613924
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[15]	validation_0-error:0.199735	validation_1-error:0.208404	vali
dation_0-f1:0.649152	validation_1-f1:0.615707
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[16]	validation_0-error:0.199924	validation_1-error:0.206701	vali
dation_0-f1:0.645638	validation_1-f1:0.616842
[22:13:42] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[17]	validation_0-error:0.199924	validation_1-error:0.205565	vali
dation_0-f1:0.646823	validation_1-f1:0.618947
[22:13:43] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 4 pruned nodes, max_depth=4
[18]	validation_0-error:0.199546	validation_1-error:0.205565	vali
dation_0-f1:0.648901	validation_1-f1:0.619748
[22:13:43] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 0 pruned nodes, max_depth=4
[19]	validation_0-error:0.200303	validation_1-error:0.204997	vali
dation_0-f1:0.644489	validation_1-f1:0.618796
[22:13:43] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 6 pruned nodes, max_depth=4
[20]	validation_0-error:0.199546	validation_1-error:0.203861	vali
```

```
dation_0-f1:0.646309    validation_1-f1:0.622503
[22:13:43] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 20 extra nodes, 4 pruned nodes, max_depth=4
[21]    validation_0-error:0.199356     validation_1-error:0.204429     vali
dation_0-f1:0.648414    validation_1-f1:0.624217
Stopping. Best iteration:
[1]     validation_0-error:0.217531     validation_1-error:0.242476     vali
dation_0-f1:0.6435      validation_1-f1:0.594492
```

In [62]:

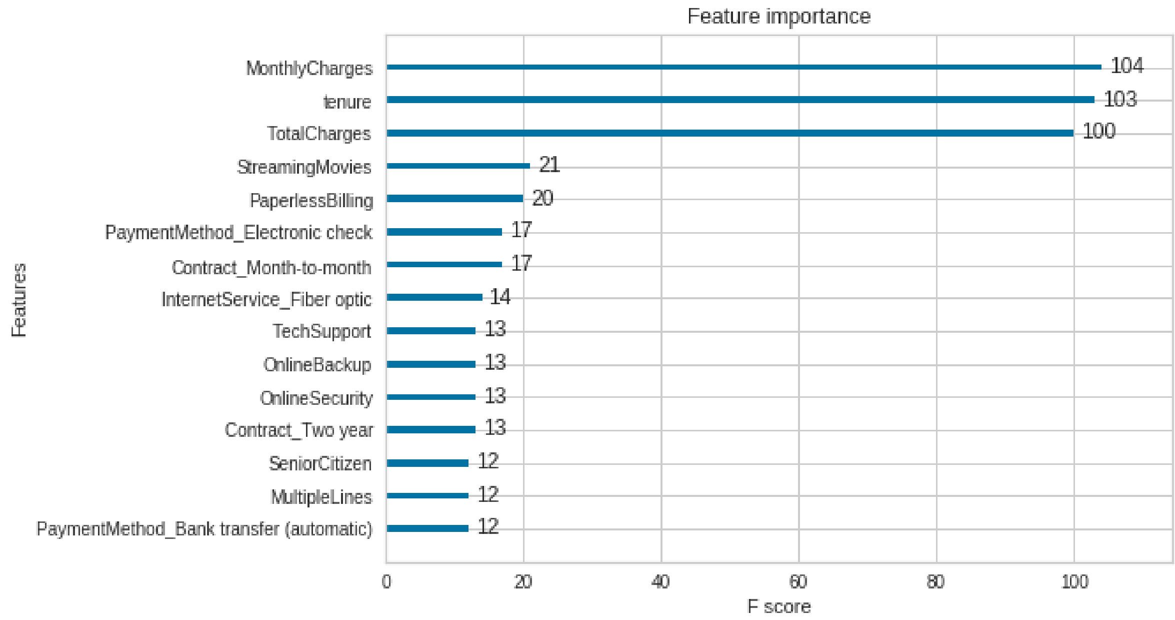
```
train_pred_xgbst, test_pred_xgbst, acc_xgbst, acc_cv_xgbst, probs_xgbst = fit_ml_algo(xgbst
                                                                X_train,
                                                                y_train,
                                                                X_test,
                                                                10)
```

```
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 16 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 6 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 20 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 8 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 30 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
```

```
s, 16 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 16 extra nodes, 12 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 6 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 24 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 16 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 14 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 18 extra nodes, 8 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 22 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 20 extra nodes, 6 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 28 extra nodes, 0 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 16 extra nodes, 4 pruned nodes, max_depth=4
[22:13:46] /workspace/src/tree/updater_prune.cc:74: tree pruning end, 1 root
s, 26 extra nodes, 2 pruned nodes, max_depth=4
```

In [63]:

```
import xgboost as xgb
xgb.plot_importance(best_xgb, max_num_features = 15)
plt.show();
```



Compare all models

In [64]:

```
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',

              'Decision Tree',
              'Gradient Boosting Trees'],
    'Score': [
        acc_knn,
        acc_log,
        acc_rf,
        acc_gaussian,
        acc_dt,
        acc_gbt,

    ]})
models.sort_values(by='Score', ascending=False)
```

Out[64]:

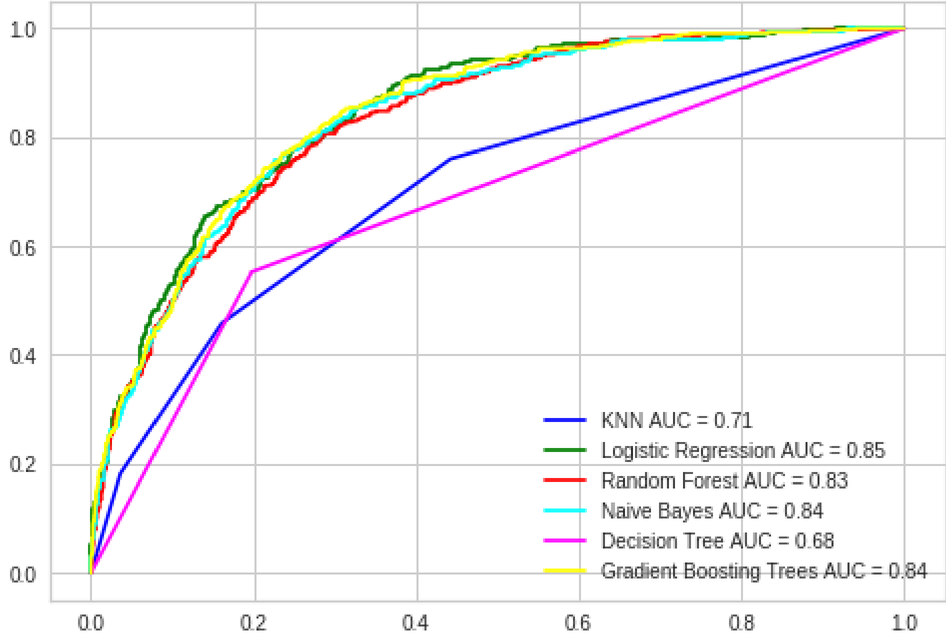
	Model	Score
1	Logistic Regression	80.86
5	Gradient Boosting Trees	80.58
2	Random Forest	80.01
0	KNN	74.56
4	Decision Tree	74.11
3	Naive Bayes	73.65

In [65]:

```
models = [
    'KNN',
    'Logistic Regression',
    'Random Forest',
    'Naive Bayes',
    'Decision Tree',
    'Gradient Boosting Trees',

]
probs = [
    probs_knn,
    probs_log,
    probs_rf,
    probs_gau,
    probs_dt,
    probs_gbt
]
colors = [
    'blue',
    'green',
    'red',
    'cyan',
    'magenta',
    'yellow',
    'black',
]
```

In [66]:

```
def plot_roc_curves(y_test, prob, model):
    fpr, tpr, threshold = metrics.roc_curve(y_test, prob)
    roc_auc = metrics.auc(fpr, tpr)
    plt.plot(fpr, tpr, 'b', label = model + ' AUC = %0.2f' % roc_auc, color=colors[i])
    plt.legend(loc = 'lower right')

for i, model in list(enumerate(models)):
    plot_roc_curves(y_test, probs[i], models[i])

plt.show()
```



Interpretation

[To Do] : Make Conclusions from the above graph and Probability scores from the test dataset

In [ ]:

In [ ]: