Machine learning 1  
Classification Report

Columns:

# id – unique observation identifier  
# age – age group of the person with the following levels: 18-24, 25-34, 35-44, 45-54, 55-64, 65+  
# gender – gender of the person with the following levels: female, male  
# education – education level of the person with the following levels: Left school before 16 years, Left school at 16 years, Left school at 17 years, Left school at 18 years, Some college or university, no certificate or degree, Professional certificate/ diploma, University degree, Masters degree, Doctorate degree  
# country – country of current residence of the person with the following levels: Australia, Canada, New Zealand, Ireland, UK, USA, Other  
# ethnicity – ethnicity of the person with the following levels: Asian, Black, Mixed-Black/Asian, Mixed-White/Asian, Mixed-White/Black, White, Other  
# personality\_neuroticism – assessment of neuroticism of the person based on psychological tests (0-100)  
# personality\_extraversion – assessment of extraversion of the person based on psychological tests (0-100)  
# personality\_openness – assessment of openness to experience of the person based on psychological tests (0-100)  
# personality\_agreeableness – assessment of agreeableness of the person based on psychological tests (0-100)  
# personality\_conscientiousness – assessment of conscientiousness of the person based on psychological tests (0-100)  
# personality\_impulsiveness – assessment of impulsiveness of the person based on psychological tests (0-100)  
# personality\_sensation – assessment of sensation of the person based on psychological tests (0-100)  
# consumption\_alcohol – declared consumption of alcohol with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_amphetamines – declared consumption of amphetamines with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_caffeine – declared consumption of caffeine with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_cannabis – declared consumption of cannabis with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_chocolate – declared consumption of chocolate with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_mushrooms – declared consumption of magic mushrooms with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_nicotine – declared consumption of nicotine with the following levels: never used, used over a decade ago, used in last decade, used in last year, used in last month, used in last week, used in last day  
# consumption\_cocaine\_last\_month – declared consumption of cocaine in the last month with the following levels: No, Yes (outcome variable, only in the training sample)

# Let's import the necessary libraries  
import json  
import pandas as pd  
import numpy as np  
from sklearn.model\_selection import train\_test\_split  
from sklearn import preprocessing  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, cross\_val\_score, StratifiedKFold, learning\_curve  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.ensemble import BaggingClassifier  
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, roc\_auc\_score, balanced\_accuracy\_score  
from sklearn import tree  
import imblearn  
from os import system  
from IPython.display import Image   
import graphviz  
import pydot  
import matplotlib.pyplot as plt  
import seaborn as sns  
import warnings  
warnings.filterwarnings('ignore')

# We need "consumption\_cocaine\_last\_month" column in test module to compare our predictions based on training results, however we do not have it in the test module.   
# In this case, we can use drugs\_test.csv file as "validation set" to compare our predictions based on training results since we do not need a result column for validations.  
# However we need to split our drugs\_train.csv file into training and test sets later.  
  
# The purpose of the test set is to test the training. If you don't have those data labeled in the same form as the training data, then it's not a test set - Source: Stack Overflow  
  
validation = pd.read\_csv('drugs\_test.csv')  
data = pd.read\_csv('drugs\_train.csv')

data.shape

(1500, 21)

data.head()

id age gender \  
0 train\_0001 45-54 male   
1 train\_0002 25-34 male   
2 train\_0003 18-24 female   
3 train\_0004 25-34 female   
4 train\_0005 18-24 male   
  
 education country \  
0 Masters degree USA   
1 University degree USA   
2 University degree USA   
3 Masters degree USA   
4 Some college or university, no certificate or ... Australia   
  
 ethnicity personality\_neuroticism personality\_extraversion \  
0 Mixed-Black/Asian 57.6 57.3   
1 Mixed-Black/Asian 47.8 67.0   
2 Mixed-Black/Asian 57.6 43.3   
3 Mixed-Black/Asian 71.8 31.2   
4 Mixed-Black/Asian 56.1 62.3   
  
 personality\_openness personality\_agreeableness ... \  
0 50.1 47.8 ...   
1 45.7 47.8 ...   
2 55.3 45.6 ...   
3 43.6 56.3 ...   
4 70.2 66.1 ...   
  
 personality\_impulsiveness personality\_sensation consumption\_alcohol \  
0 42.8 22.4 used in last week   
1 33.8 30.8 used in last week   
2 63.0 62.0 used in last month   
3 63.0 71.1 used in last day   
4 50.4 62.0 used in last week   
  
 consumption\_amphetamines consumption\_caffeine consumption\_cannabis \  
0 used over a decade ago used in last day used in last week   
1 never used used in last week never used   
2 never used used in last day used in last week   
3 never used used in last day used in last decade   
4 never used used in last month used in last month   
  
 consumption\_chocolate consumption\_mushrooms consumption\_nicotine \  
0 used in last day never used used in last week   
1 used in last day never used never used   
2 used in last week used in last year used in last month   
3 used in last day never used used in last decade   
4 used in last day used in last year used in last month   
  
 consumption\_cocaine\_last\_month   
0 No   
1 No   
2 No   
3 No   
4 No   
  
[5 rows x 21 columns]

data.info()  
# Our data has 1500 rows and 20 columns. Data types are object, int64 and float64. There are no missing values.  
# We need to convert object data types to categorical data types and convert categorical data types to numerical data types.

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 1500 entries, 0 to 1499  
Data columns (total 21 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 id 1500 non-null object   
 1 age 1500 non-null object   
 2 gender 1500 non-null object   
 3 education 1500 non-null object   
 4 country 1500 non-null object   
 5 ethnicity 1500 non-null object   
 6 personality\_neuroticism 1500 non-null float64  
 7 personality\_extraversion 1500 non-null float64  
 8 personality\_openness 1500 non-null float64  
 9 personality\_agreeableness 1500 non-null float64  
 10 personality\_conscientiousness 1500 non-null float64  
 11 personality\_impulsiveness 1500 non-null float64  
 12 personality\_sensation 1500 non-null float64  
 13 consumption\_alcohol 1500 non-null object   
 14 consumption\_amphetamines 1500 non-null object   
 15 consumption\_caffeine 1500 non-null object   
 16 consumption\_cannabis 1500 non-null object   
 17 consumption\_chocolate 1500 non-null object   
 18 consumption\_mushrooms 1500 non-null object   
 19 consumption\_nicotine 1500 non-null object   
 20 consumption\_cocaine\_last\_month 1500 non-null object   
dtypes: float64(7), object(14)  
memory usage: 246.2+ KB

# Drop the "id" column since it is not needed for our analysis  
data.drop(['id'], axis=1, inplace=True)  
validation.drop(['id'], axis=1, inplace=True)

# Some of our columns needs normaliaztion, so we will do that here  
numeric\_columns = data.select\_dtypes(include=['int64', 'float64']).columns  
data[numeric\_columns] = data[numeric\_columns].apply(lambda x: (x - x.mean()) / (x.max() - x.min()))  
  
numeric\_columns\_validation = validation.select\_dtypes(include=['int64', 'float64']).columns  
validation[numeric\_columns\_validation] = validation[numeric\_columns\_validation].apply(lambda x: (x - x.mean()) / (x.max() - x.min()))  
  
  
data.head()

age gender education \  
0 45-54 male Masters degree   
1 25-34 male University degree   
2 18-24 female University degree   
3 25-34 female Masters degree   
4 18-24 male Some college or university, no certificate or ...   
  
 country ethnicity personality\_neuroticism \  
0 USA Mixed-Black/Asian 0.060927   
1 USA Mixed-Black/Asian -0.037073   
2 USA Mixed-Black/Asian 0.060927   
3 USA Mixed-Black/Asian 0.202927   
4 Australia Mixed-Black/Asian 0.045927   
  
 personality\_extraversion personality\_openness personality\_agreeableness \  
0 0.072463 -0.030087 -0.02166   
1 0.169463 -0.074087 -0.02166   
2 -0.067537 0.021913 -0.04366   
3 -0.188537 -0.095087 0.06334   
4 0.122463 0.170913 0.16134   
  
 personality\_conscientiousness personality\_impulsiveness \  
0 0.037012 -0.041722   
1 0.060012 -0.131722   
2 -0.000988 0.160278   
3 -0.181988 0.160278   
4 -0.075988 0.034278   
  
 personality\_sensation consumption\_alcohol consumption\_amphetamines \  
0 -0.299163 used in last week used over a decade ago   
1 -0.215163 used in last week never used   
2 0.096837 used in last month never used   
3 0.187837 used in last day never used   
4 0.096837 used in last week never used   
  
 consumption\_caffeine consumption\_cannabis consumption\_chocolate \  
0 used in last day used in last week used in last day   
1 used in last week never used used in last day   
2 used in last day used in last week used in last week   
3 used in last day used in last decade used in last day   
4 used in last month used in last month used in last day   
  
 consumption\_mushrooms consumption\_nicotine consumption\_cocaine\_last\_month   
0 never used used in last week No   
1 never used never used No   
2 used in last year used in last month No   
3 never used used in last decade No   
4 used in last year used in last month No

# Our categorical columns need to be encoded, so we will do that here  
# To not lose any information, we will save the original categorical columns in a json file for reference  
categorical\_columns = data.select\_dtypes(include=['object']).columns  
data[categorical\_columns] = data[categorical\_columns].apply(lambda x: x.astype('category'))  
  
categorical\_columns\_validation = validation.select\_dtypes(include=['object']).columns  
validation[categorical\_columns\_validation] = data[categorical\_columns\_validation].apply(lambda x: x.astype('category'))  
  
def map\_func(column):  
 return dict(enumerate(column.cat.categories))  
  
gender\_map = map\_func(data.gender)  
education\_map = map\_func(data.education)  
country\_map = map\_func(data.country)  
ethnicity\_map = map\_func(data.ethnicity)  
consumption\_alcohol\_map = map\_func(data.consumption\_alcohol)  
consumption\_amphetamines\_map = map\_func(data.consumption\_amphetamines)  
consumption\_caffeine\_map = map\_func(data.consumption\_caffeine)  
consumption\_cannabis\_map = map\_func(data.consumption\_cannabis)  
consumption\_chocolate\_map = map\_func(data.consumption\_chocolate)  
consumption\_mushrooms\_map = map\_func(data.consumption\_mushrooms)  
consumption\_nicotine\_map = map\_func(data.consumption\_nicotine)  
consumption\_cocaine\_last\_month\_map = map\_func(data.consumption\_cocaine\_last\_month)  
  
data\_dict = dict({'gender\_map': gender\_map,  
 'education\_map': education\_map,  
 'country\_map': country\_map,  
 'ethnicity\_map': ethnicity\_map,  
 'consumption\_alcohol\_map': consumption\_alcohol\_map,  
 'consumption\_amphetamines\_map': consumption\_amphetamines\_map,  
 'consumption\_caffeine\_map': consumption\_caffeine\_map,  
 'consumption\_cannabis\_map': consumption\_cannabis\_map,  
 'consumption\_chocolate\_map': consumption\_chocolate\_map,  
 'consumption\_mushrooms\_map': consumption\_mushrooms\_map,  
 'consumption\_nicotine\_map': consumption\_nicotine\_map,  
 'consumption\_cocaine\_last\_month\_map': consumption\_cocaine\_last\_month\_map})  
  
json\_data = json.dumps(data\_dict, indent=4)  
with open('data\_dict.json', 'w+') as f:  
 f.write(json\_data)  
  
data[categorical\_columns] = data[categorical\_columns].apply(lambda x: x.cat.codes)  
validation[categorical\_columns\_validation] = validation[categorical\_columns\_validation].apply(lambda x: x.cat.codes)

# As you see, we have encoded our categorical columns and normalized our numeric columns.  
# Now we can split our data into training and test sets and then we can start our analysis.  
data.head()

age gender education country ethnicity personality\_neuroticism \  
0 3 1 5 6 2 0.060927   
1 1 1 8 6 2 -0.037073   
2 0 0 8 6 2 0.060927   
3 1 0 5 6 2 0.202927   
4 0 1 7 0 2 0.045927   
  
 personality\_extraversion personality\_openness personality\_agreeableness \  
0 0.072463 -0.030087 -0.02166   
1 0.169463 -0.074087 -0.02166   
2 -0.067537 0.021913 -0.04366   
3 -0.188537 -0.095087 0.06334   
4 0.122463 0.170913 0.16134   
  
 personality\_conscientiousness personality\_impulsiveness \  
0 0.037012 -0.041722   
1 0.060012 -0.131722   
2 -0.000988 0.160278   
3 -0.181988 0.160278   
4 -0.075988 0.034278   
  
 personality\_sensation consumption\_alcohol consumption\_amphetamines \  
0 -0.299163 4 6   
1 -0.215163 4 0   
2 0.096837 3 0   
3 0.187837 1 0   
4 0.096837 4 0   
  
 consumption\_caffeine consumption\_cannabis consumption\_chocolate \  
0 1 4 1   
1 4 0 1   
2 1 4 4   
3 1 2 1   
4 3 3 1   
  
 consumption\_mushrooms consumption\_nicotine consumption\_cocaine\_last\_month   
0 0 4 0   
1 0 0 0   
2 5 3 0   
3 0 2 0   
4 5 3 0

X = data.drop('consumption\_cocaine\_last\_month', axis=1)  
y = data.pop('consumption\_cocaine\_last\_month')  
  
Xtrain, Xtest, ytrain, ytest = train\_test\_split(X,y, test\_size=0.25, random\_state=1)

# Building Decision Tree Model  
# default 'gini' criteria to split the tree  
  
  
dTgini = DecisionTreeClassifier(criterion='gini', random\_state=1)  
  
# default 'entropy' criteria to split the tree  
dTentropy = DecisionTreeClassifier(criterion='entropy', random\_state=2)

print('Decision Tree with Gini Index:')  
dTgini.fit(Xtrain, ytrain)  
print('Accuracy of DT with Gini Index: (Train Set) {:.4f}'.format(dTgini.score(Xtrain, ytrain)))  
print('\n')

Decision Tree with Gini Index:  
Accuracy of DT with Gini Index: (Train Set) 1.0000

print('Decision Tree with Entropy:')  
dTentropy.fit(Xtrain, ytrain)  
print('Accuracy of DT with Entropy: (Train Set) {:.4f}'.format(dTentropy.score(Xtest, ytest)))  
print('\n')

Decision Tree with Entropy:  
Accuracy of DT with Entropy: (Train Set) 0.8560

#Visualizing the decision tree as png  
dot\_data = tree.export\_graphviz(dTgini, out\_file=None, feature\_names=X.columns, filled=True, rounded=True, special\_characters=True)  
graph = graphviz.Source(dot\_data)  
graph.render("drugs\_decision\_tree\_gini")  
  
(graph,) = pydot.graph\_from\_dot\_file('drugs\_decision\_tree\_gini')  
graph.write\_png('drugs\_decision\_tree\_gini.png')  
  
# Due to size of the decision tree, we will not be able to visualize it properly but It is nice to save it as PNG file to be able to view it later.

Decision Tree

# Our model may have some overfitting problem, cross validation is a good way to see if we can improve our model.  
# We will use cross validation to see if we can improve our model.  
# We will use 10-fold cross validation to see if we can improve our model.  
  
# The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into.   
# As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model,   
# such as k=10 becoming 10-fold cross-validation.  
# If k=10 the dataset will be divided into 5 equal parts and the below process will run 10 times, each time with a different holdout set.  
  
# We will use 'gini' criteria to split the tree.  
from sklearn.model\_selection import cross\_val\_score  
  
scores = cross\_val\_score(dTgini, Xtest, ytest, cv=10, scoring='accuracy')  
print('Cross Validation Accuracy of DT with Gini Index: {:.4f}'.format(scores.mean()))  
  
# We can see that our model is not overfitting, so we will use 'entropy' criteria to split the tree.  
  
scores = cross\_val\_score(dTentropy, Xtest, ytest, cv=10, scoring='accuracy')  
print('Cross Validation Accuracy of DT with Entropy: {:.4f}'.format(scores.mean()))  
  
# Both of our models are not overfitting

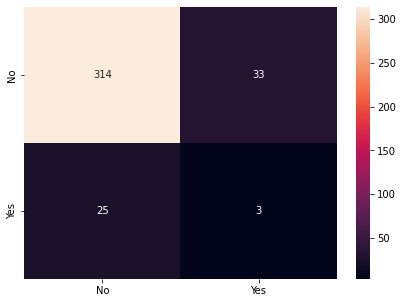
Cross Validation Accuracy of DT with Gini Index: 0.8317  
Cross Validation Accuracy of DT with Entropy: 0.8454

# importance of features in the tree building   
print(pd.DataFrame(dTgini.feature\_importances\_, columns = ['Imp'], index = Xtrain.columns))

Imp  
age 0.041264  
gender 0.000000  
education 0.065945  
country 0.052809  
ethnicity 0.019369  
personality\_neuroticism 0.076485  
personality\_extraversion 0.120869  
personality\_openness 0.098713  
personality\_agreeableness 0.036436  
personality\_conscientiousness 0.103958  
personality\_impulsiveness 0.061303  
personality\_sensation 0.099135  
consumption\_alcohol 0.037160  
consumption\_amphetamines 0.070684  
consumption\_caffeine 0.003802  
consumption\_cannabis 0.021259  
consumption\_chocolate 0.029535  
consumption\_mushrooms 0.024985  
consumption\_nicotine 0.036289

#Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
  
ypredict = dTgini.predict(Xtest)  
  
cm = confusion\_matrix(ytest, ypredict, labels=[0,1])  
  
df\_cm = pd.DataFrame(cm, index = [i for i in ['No', 'Yes']],  
 columns = [i for i in ['No', 'Yes']]  
 )  
  
plt.figure(figsize = (7,5))  
sns.heatmap(df\_cm, annot = True, fmt = 'g')

<AxesSubplot:>

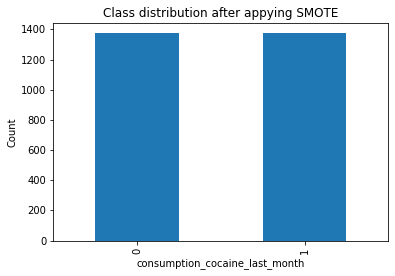


print(classification\_report(ytest, ypredict))  
  
# As we see from our classification report, our predictions are accurate but our data is not very well balanced.

precision recall f1-score support  
  
 0 0.93 0.90 0.92 347  
 1 0.08 0.11 0.09 28  
  
 accuracy 0.85 375  
 macro avg 0.50 0.51 0.50 375  
weighted avg 0.86 0.85 0.85 375

# Imbalanced data is a problem in machine learning.  
# We will use SMOTE to balance our data.  
  
import collections  
  
counter = collections.Counter(y)  
print('Before',counter)  
  
oversample = imblearn.over\_sampling.SMOTE(random\_state=1)  
X, y = oversample.fit\_sample(X, y)  
  
counter = collections.Counter(y)  
print('After',counter)  
  
  
pd.Series(y).value\_counts().plot(kind='bar', title='Class distribution after appying SMOTE', xlabel='consumption\_cocaine\_last\_month', ylabel='Count')  
plt.show()

Before Counter({0: 1373, 1: 127})  
After Counter({0: 1373, 1: 1373})



# As you see from our class distribution, our data is now balanced.  
# Now we will use our resampled data to train our model.  
  
Xtrain, Xtest, ytrain, ytest = train\_test\_split(X,y, test\_size=0.25, random\_state=1)

print('Decision Tree with Gini Index:')  
dTgini.fit(Xtrain, ytrain)  
print('Accuracy of DT with Gini Index: {:.4f}'.format(dTgini.score(Xtest, ytest)))  
print('\n')  
  
# predicted values  
ypredict = dTgini.predict(Xtest)  
  
print('balanced accuracy score: {:.4f}'.format(balanced\_accuracy\_score(ytest, ypredict)))  
print('\n')

Decision Tree with Gini Index:  
Accuracy of DT with Gini Index: 0.8923  
  
  
balanced accuracy score: 0.8922

# Let's use validation set to predict Y axis since we do no have Y axis in validation set  
  
validation\_y = dTgini.predict(validation)  
score = dTgini.score(validation,validation\_y)  
  
# So let's merge the labels with validation data as a new column and try to analyze it.  
validation['consumption\_cocaine\_last\_month'] = validation\_y  
  
validation\_x = validation.drop('consumption\_cocaine\_last\_month', axis=1)  
validation\_y = validation.pop('consumption\_cocaine\_last\_month')  
  
validation.head()

age gender education country ethnicity personality\_neuroticism \  
0 3 1 5 6 2 -0.168255   
1 1 1 8 6 2 0.141960   
2 0 0 8 6 2 0.063776   
3 1 0 5 6 2 0.261758   
4 0 1 7 0 2 0.141960   
  
 personality\_extraversion personality\_openness personality\_agreeableness \  
0 0.244132 0.081794 -0.055461   
1 -0.021868 0.220953 -0.081165   
2 0.028132 -0.191129 0.102434   
3 0.003132 -0.074624 -0.136244   
4 -0.085868 -0.360493 -0.136244   
  
 personality\_conscientiousness personality\_impulsiveness \  
0 -0.124354 0.16154   
1 0.145953 0.09654   
2 0.112664 0.09654   
3 -0.100386 0.16154   
4 -0.368029 -0.04046   
  
 personality\_sensation consumption\_alcohol consumption\_amphetamines \  
0 0.20901 4 6   
1 0.20901 4 0   
2 -0.11399 3 0   
3 -0.03599 1 0   
4 -0.11399 4 0   
  
 consumption\_caffeine consumption\_cannabis consumption\_chocolate \  
0 1 4 1   
1 4 0 1   
2 1 4 4   
3 1 2 1   
4 3 3 1   
  
 consumption\_mushrooms consumption\_nicotine   
0 0 4   
1 0 0   
2 5 3   
3 0 2   
4 5 3

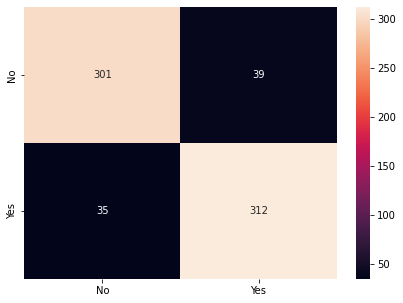
# We can not print a new score based on validation dataset because labels are created based on DecisionTree predict method,   
# if we try to predict another score with validation\_x and validation\_y the result will be 100 percent.  
# So we will take that values as result of our model and not going to use for testing quality of the model.   
# We can assume like validation set is a real life data input coming from a pipeline   
# and there is no point to make further analysis based on it because there is not a "REAL" labels column.

print('Decision Tree with Entropy:')  
dTentropy.fit(Xtrain, ytrain)  
print('Accuracy of DT with Entropy: {:.4f}'.format(dTentropy.score(Xtest, ytest)))  
print('\n')  
  
# predicted values  
ypredict = dTentropy.predict(Xtest)  
  
print('balanced accuracy score: {:.4f}'.format(balanced\_accuracy\_score(ytest, ypredict)))  
print('\n')

Decision Tree with Entropy:  
Accuracy of DT with Entropy: 0.8879  
  
  
balanced accuracy score: 0.8877

#Confusion Matrix  
from sklearn.metrics import confusion\_matrix  
  
ypredict = dTgini.predict(Xtest)  
  
cm = confusion\_matrix(ytest, ypredict, labels=[0,1])  
  
df\_cm = pd.DataFrame(cm, index = [i for i in ['No', 'Yes']],  
 columns = [i for i in ['No', 'Yes']]  
 )  
  
plt.figure(figsize = (7,5))  
sns.heatmap(df\_cm, annot = True, fmt = 'g')

<AxesSubplot:>

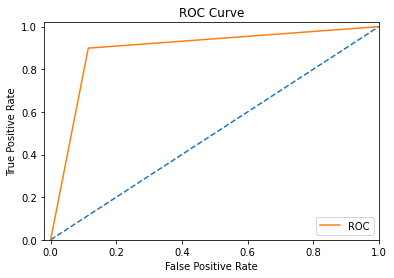


print(classification\_report(ytest, ypredict))

precision recall f1-score support  
  
 0 0.90 0.89 0.89 340  
 1 0.89 0.90 0.89 347  
  
 accuracy 0.89 687  
 macro avg 0.89 0.89 0.89 687  
weighted avg 0.89 0.89 0.89 687

# After balancing the data classification report shows our "real" accuracy is around .89.

#ROC curve Decison Tree  
  
dTprob = dTgini.predict\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest, dTprob)  
  
plt.plot([0,1],[0,1], linestyle='--')  
  
plt.plot(fpr, tpr, label='ROC')  
\_ = plt.xlabel('False Positive Rate')  
\_ = plt.ylabel('True Positive Rate')  
\_ = plt.title('ROC Curve')  
\_ = plt.xlim([-0.02, 1])  
\_ = plt.ylim([0, 1.02])  
\_ = plt.legend(loc="lower right")



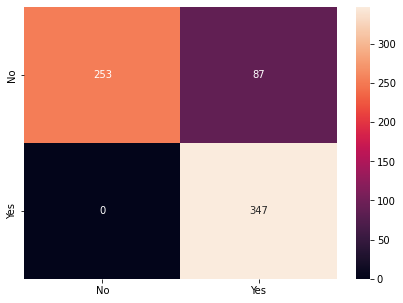
#The more that the ROC curve hugs the top left corner of the plot, the better the model does at classifying the data into categories.  
#Our model has high AUC, which indicates that it has the very wide area under the curve and is the nice model at correctly classifying observations into categories.  
decisionTree\_score = roc\_auc\_score(ytest, dTprob)  
print('AUC:',roc\_auc\_score(ytest, dTprob))

AUC: 0.8922147821664689

# KNN model with K=5 and K=10  
  
from sklearn.neighbors import KNeighborsClassifier  
  
knn5 = KNeighborsClassifier(n\_neighbors=5)  
knn10 = KNeighborsClassifier(n\_neighbors=10)  
  
knn5.fit(Xtrain, ytrain)  
knn10.fit(Xtrain, ytrain)  
  
print('KNN with K=5:')  
print('Accuracy of KNN with K=5: {:.4f}'.format(knn5.score(Xtest, ytest)))  
print('\n')  
  
print('KNN with K=10:')  
print('Accuracy of KNN with K=10: {:.4f}'.format(knn10.score(Xtest, ytest)))  
print('\n')  
  
# predicted values  
ypredict = knn5.predict(Xtest)  
  
print('balanced accuracy score: {:.4f}'.format(balanced\_accuracy\_score(ytest, ypredict)))  
print('\n')

KNN with K=5:  
Accuracy of KNN with K=5: 0.8734  
  
  
KNN with K=10:  
Accuracy of KNN with K=10: 0.8574  
  
  
balanced accuracy score: 0.8721

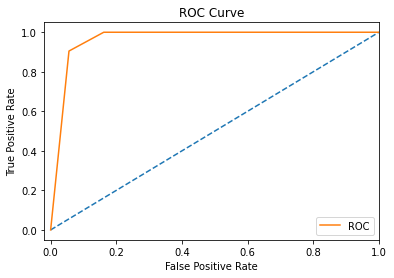
#Confusion Matrix  
cm = confusion\_matrix(ytest, ypredict, labels = [0, 1])  
  
df\_cm = pd.DataFrame(cm, index = [i for i in ['No', 'Yes']],  
 columns = [i for i in ['No', 'Yes']])  
  
plt.figure(figsize=(7,5))  
sns.heatmap(df\_cm, annot = True, fmt='g');



print('Classification Report:\n', classification\_report(ytest, ypredict))

Classification Report:  
 precision recall f1-score support  
  
 0 1.00 0.74 0.85 340  
 1 0.80 1.00 0.89 347  
  
 accuracy 0.87 687  
 macro avg 0.90 0.87 0.87 687  
weighted avg 0.90 0.87 0.87 687

prob = knn5.predict\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest, prob)  
  
plt.plot([0,1],[0,1],linestyle='--')  
plt.plot(fpr, tpr, label='ROC')  
\_=plt.xlabel('False Positive Rate')  
\_=plt.ylabel('True Positive Rate')  
\_=plt.title('ROC Curve')  
\_=plt.xlim([-0.02,1])  
\_=plt.legend(loc='lower right')



knn5\_score = roc\_auc\_score(ytest, prob)  
print('AUC:',roc\_auc\_score(ytest, prob))

AUC: 0.9643668418375996

# After 0.93 percent of accuracy I would love to perform a cross validation check to test overfitting  
  
scores = cross\_val\_score(knn5, Xtest, ytest, cv=10, scoring='accuracy')  
print('Cross Validation Accuracy of KNN: {:.4f}'.format(scores.mean()))  
  
# Cross validation score shows us a more realistic accuracy rate

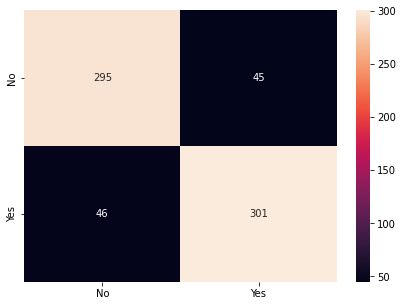
Cross Validation Accuracy of KNN: 0.7786

#adabossting  
adaBoost = AdaBoostClassifier(  
 #base\_estimator=dTgini1,  
 n\_estimators=50,  
 random\_state=2)  
adaBoost = adaBoost.fit(Xtrain, ytrain)  
  
ypredict= adaBoost.predict(Xtest)  
print('Train Score:',adaBoost.score(Xtrain, ytrain))  
print('Test Score:',adaBoost.score(Xtest, ytest))

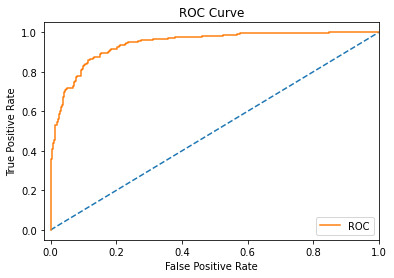
Train Score: 0.893152015541525  
Test Score: 0.8675400291120815

print('Confusion Matrix:')  
  
cm = confusion\_matrix(ytest, ypredict, labels = [0,1])  
  
df\_cm = pd.DataFrame(cm, index = [i for i in ['No','Yes']],  
 columns = [i for i in ['No', 'Yes']])  
plt.figure(figsize=(7,5))  
sns.heatmap(df\_cm, annot=True, fmt='g');

Confusion Matrix:



prob = adaBoost.predict\_log\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest, prob)  
plt.plot([0,1],[0,1],linestyle='--')  
plt.plot(fpr, tpr, label='ROC')  
\_=plt.xlabel('False Positive Rate')  
\_=plt.ylabel('True Positive Rate')  
\_=plt.title('ROC Curve')  
\_=plt.xlim([-0.02,1])  
\_=plt.legend(loc='lower right')



adaBoost\_score = roc\_auc\_score(ytest, prob)  
print('AUC:', roc\_auc\_score(ytest, prob))

AUC: 0.9416511273097135

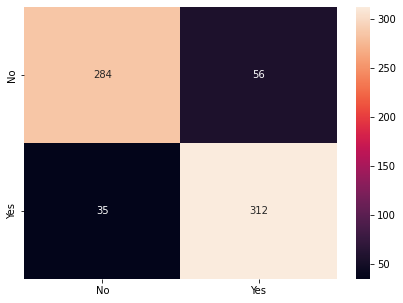
# Again, I feel like we need a cross-validation check.  
  
scores = cross\_val\_score(adaBoost, Xtest, ytest, cv=10, scoring='accuracy')  
print('Cross Validation Accuracy of adaBoost: {:.4f}'.format(scores.mean()))  
  
# This time, Cross validation accuracy is same as test accuracy

Cross Validation Accuracy of Adaboost: 0.8618

#GradientBoost  
gradientBoostingC = GradientBoostingClassifier(learning\_rate = 0.1,  
 n\_estimators = 25,  
 random\_state = 3)  
gradientBoostingC = gradientBoostingC.fit(Xtrain, ytrain)  
  
ypredict = gradientBoostingC.predict(Xtest)  
print('Train Score:', gradientBoostingC.score(Xtrain, ytrain))  
print('Test Score:', gradientBoostingC.score(Xtest, ytest))

Train Score: 0.8936376881981545  
Test Score: 0.8675400291120815

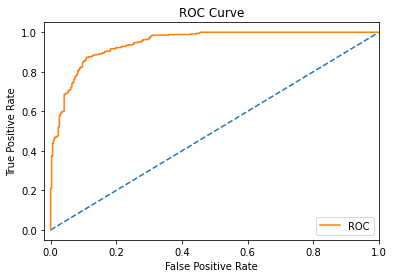
cm = confusion\_matrix(ytest, ypredict, labels=[0,1])  
df\_cm = pd.DataFrame(cm, index = [i for i in ['No','Yes']],  
 columns = [i for i in ['No','Yes']])  
plt.figure(figsize=(7,5))  
sns.heatmap(df\_cm, annot = True, fmt = 'g');



print(classification\_report(ytest, ypredict))

precision recall f1-score support  
  
 0 0.89 0.84 0.86 340  
 1 0.85 0.90 0.87 347  
  
 accuracy 0.87 687  
 macro avg 0.87 0.87 0.87 687  
weighted avg 0.87 0.87 0.87 687

prob = gradientBoostingC.predict\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest,prob)  
plt.plot([0,1],[0,1],linestyle='--')  
plt.plot(fpr, tpr, label='ROC')  
\_=plt.xlabel('False Positive Rate')  
\_=plt.ylabel('True Positive Rate')  
\_=plt.title('ROC Curve')  
\_=plt.xlim([-0.02,1])  
\_=plt.legend(loc='lower right')



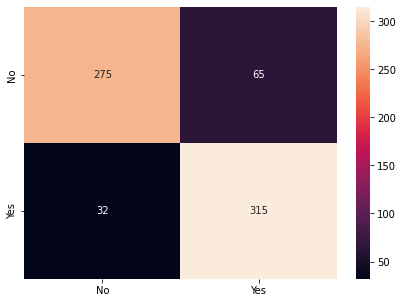
gradientBoost\_score = roc\_auc\_score(ytest, prob)  
print('AUC:', roc\_auc\_score(ytest, prob))

AUC: 0.9459569418545517

#Ensemble Random Forest Classifier  
  
RFC = RandomForestClassifier(n\_estimators=50,   
 random\_state=4,   
 max\_features='log2',  
 max\_depth=5)  
RFC = RFC.fit(Xtrain, ytrain)  
  
ypredict = RFC.predict(Xtest)  
  
print('Train score:', RFC.score(Xtrain, ytrain))  
print('Test Score:', RFC.score(Xtest, ytest))

Train score: 0.8902379796017484  
Test Score: 0.858806404657933

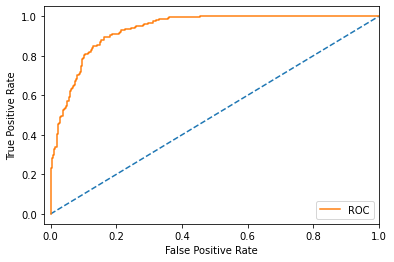
cm = confusion\_matrix(ytest, ypredict, labels=[0,1])  
df\_cm = pd.DataFrame(cm, index=[i for i in ['No','Yes']],  
 columns = [i for i in ['No', 'Yes']])  
plt.figure(figsize=(7,5))  
sns.heatmap(df\_cm, annot = True, fmt = 'g');



print(classification\_report(ytest, ypredict))

precision recall f1-score support  
  
 0 0.90 0.81 0.85 340  
 1 0.83 0.91 0.87 347  
  
 accuracy 0.86 687  
 macro avg 0.86 0.86 0.86 687  
weighted avg 0.86 0.86 0.86 687

prob = RFC.predict\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest, prob)  
  
plt.plot([0,1],[0,1], linestyle ='--')  
  
plt.plot(fpr, tpr, label='ROC')  
\_=plt.xlabel('False Positive Rate')  
\_=plt.ylabel('True Positive Rate')  
\_=plt.xlim([-0.02,1])  
\_=plt.legend(loc='lower right')



rfc\_score = roc\_auc\_score(ytest, prob)  
print('AUC:', roc\_auc\_score(ytest, prob))

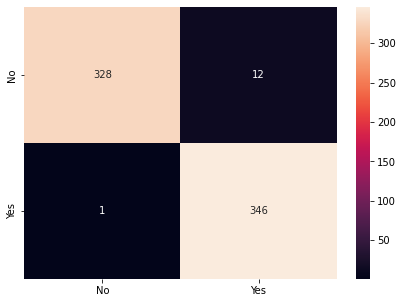
AUC: 0.9331496863875234

# SVM Classifier with RBF Kernel and GridSearchCV to find best parameters  
  
param\_grid = {'C': [0.1, 1, 10, 100, 1000],  
 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],  
 'kernel': ['rbf']}  
grid = GridSearchCV(SVC(probability=True), param\_grid, refit=True, verbose=3)  
grid.fit(Xtrain, ytrain)  
  
#print(grid.best\_params\_)  
#print(grid.best\_estimator\_)  
grid\_predictions = grid.predict(Xtest)

Fitting 5 folds for each of 25 candidates, totalling 125 fits  
[CV 1/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.505 total time= 0.8s  
[CV 2/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.507 total time= 0.7s  
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.507 total time= 0.7s  
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.507 total time= 0.7s  
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.509 total time= 0.7s  
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.869 total time= 0.6s  
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.862 total time= 0.6s  
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.828 total time= 0.6s  
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.857 total time= 0.6s  
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.827 total time= 0.6s  
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.847 total time= 0.5s  
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.794 total time= 0.5s  
[CV 3/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.774 total time= 0.5s  
[CV 4/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.808 total time= 0.5s  
[CV 5/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.793 total time= 0.5s  
[CV 1/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.740 total time= 0.7s  
[CV 2/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.665 total time= 0.7s  
[CV 3/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.687 total time= 0.7s  
[CV 4/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.675 total time= 0.7s  
[CV 5/5] END ....C=0.1, gamma=0.001, kernel=rbf;, score=0.689 total time= 0.7s  
[CV 1/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.500 total time= 0.7s  
[CV 2/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.502 total time= 0.7s  
[CV 3/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.502 total time= 0.7s  
[CV 4/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.502 total time= 0.7s  
[CV 5/5] END ...C=0.1, gamma=0.0001, kernel=rbf;, score=0.501 total time= 0.7s  
[CV 1/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.879 total time= 0.8s  
[CV 2/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.854 total time= 0.8s  
[CV 3/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.840 total time= 0.8s  
[CV 4/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.864 total time= 0.7s  
[CV 5/5] END ..........C=1, gamma=1, kernel=rbf;, score=0.854 total time= 0.8s  
[CV 1/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.942 total time= 0.5s  
[CV 2/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.932 total time= 0.4s  
[CV 3/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.942 total time= 0.4s  
[CV 4/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.930 total time= 0.4s  
[CV 5/5] END ........C=1, gamma=0.1, kernel=rbf;, score=0.925 total time= 0.4s  
[CV 1/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.852 total time= 0.4s  
[CV 2/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.867 total time= 0.4s  
[CV 3/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.820 total time= 0.4s  
[CV 4/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.847 total time= 0.4s  
[CV 5/5] END .......C=1, gamma=0.01, kernel=rbf;, score=0.837 total time= 0.4s  
[CV 1/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.823 total time= 0.5s  
[CV 2/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.767 total time= 0.5s  
[CV 3/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.762 total time= 0.5s  
[CV 4/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.779 total time= 0.5s  
[CV 5/5] END ......C=1, gamma=0.001, kernel=rbf;, score=0.781 total time= 0.5s  
[CV 1/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.731 total time= 0.7s  
[CV 2/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.660 total time= 0.7s  
[CV 3/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.680 total time= 0.7s  
[CV 4/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.675 total time= 0.7s  
[CV 5/5] END .....C=1, gamma=0.0001, kernel=rbf;, score=0.689 total time= 0.7s  
[CV 1/5] END .........C=10, gamma=1, kernel=rbf;, score=0.893 total time= 0.7s  
[CV 2/5] END .........C=10, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 3/5] END .........C=10, gamma=1, kernel=rbf;, score=0.850 total time= 0.8s  
[CV 4/5] END .........C=10, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 5/5] END .........C=10, gamma=1, kernel=rbf;, score=0.869 total time= 0.8s  
[CV 1/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.951 total time= 0.4s  
[CV 2/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.964 total time= 0.4s  
[CV 3/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.954 total time= 0.4s  
[CV 4/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.949 total time= 0.4s  
[CV 5/5] END .......C=10, gamma=0.1, kernel=rbf;, score=0.939 total time= 0.4s  
[CV 1/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.908 total time= 0.3s  
[CV 2/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.900 total time= 0.4s  
[CV 3/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.852 total time= 0.3s  
[CV 4/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.896 total time= 0.3s  
[CV 5/5] END ......C=10, gamma=0.01, kernel=rbf;, score=0.873 total time= 0.3s  
[CV 1/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.850 total time= 0.4s  
[CV 2/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.830 total time= 0.4s  
[CV 3/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.796 total time= 0.4s  
[CV 4/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.818 total time= 0.4s  
[CV 5/5] END .....C=10, gamma=0.001, kernel=rbf;, score=0.825 total time= 0.4s  
[CV 1/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.801 total time= 0.6s  
[CV 2/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.750 total time= 0.5s  
[CV 3/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.757 total time= 0.5s  
[CV 4/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.789 total time= 0.5s  
[CV 5/5] END ....C=10, gamma=0.0001, kernel=rbf;, score=0.769 total time= 0.5s  
[CV 1/5] END ........C=100, gamma=1, kernel=rbf;, score=0.893 total time= 0.7s  
[CV 2/5] END ........C=100, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 3/5] END ........C=100, gamma=1, kernel=rbf;, score=0.850 total time= 0.8s  
[CV 4/5] END ........C=100, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 5/5] END ........C=100, gamma=1, kernel=rbf;, score=0.869 total time= 0.8s  
[CV 1/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.949 total time= 0.4s  
[CV 2/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.964 total time= 0.4s  
[CV 3/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.949 total time= 0.4s  
[CV 4/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.934 total time= 0.4s  
[CV 5/5] END ......C=100, gamma=0.1, kernel=rbf;, score=0.934 total time= 0.4s  
[CV 1/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.915 total time= 0.5s  
[CV 2/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.913 total time= 0.5s  
[CV 3/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.898 total time= 0.4s  
[CV 4/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.915 total time= 0.5s  
[CV 5/5] END .....C=100, gamma=0.01, kernel=rbf;, score=0.898 total time= 0.5s  
[CV 1/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.862 total time= 0.4s  
[CV 2/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.862 total time= 0.4s  
[CV 3/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.820 total time= 0.4s  
[CV 4/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.852 total time= 0.4s  
[CV 5/5] END ....C=100, gamma=0.001, kernel=rbf;, score=0.847 total time= 0.4s  
[CV 1/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.823 total time= 0.5s  
[CV 2/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.803 total time= 0.5s  
[CV 3/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.774 total time= 0.5s  
[CV 4/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.806 total time= 0.5s  
[CV 5/5] END ...C=100, gamma=0.0001, kernel=rbf;, score=0.779 total time= 0.4s  
[CV 1/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.893 total time= 0.7s  
[CV 2/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 3/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.850 total time= 0.8s  
[CV 4/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.871 total time= 0.8s  
[CV 5/5] END .......C=1000, gamma=1, kernel=rbf;, score=0.869 total time= 0.8s  
[CV 1/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.949 total time= 0.4s  
[CV 2/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.964 total time= 0.4s  
[CV 3/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.949 total time= 0.4s  
[CV 4/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.934 total time= 0.4s  
[CV 5/5] END .....C=1000, gamma=0.1, kernel=rbf;, score=0.934 total time= 0.4s  
[CV 1/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.927 total time= 1.0s  
[CV 2/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.930 total time= 1.0s  
[CV 3/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.913 total time= 0.9s  
[CV 4/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.922 total time= 1.0s  
[CV 5/5] END ....C=1000, gamma=0.01, kernel=rbf;, score=0.900 total time= 1.0s  
[CV 1/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.883 total time= 0.7s  
[CV 2/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.881 total time= 0.7s  
[CV 3/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.840 total time= 0.6s  
[CV 4/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.864 total time= 0.6s  
[CV 5/5] END ...C=1000, gamma=0.001, kernel=rbf;, score=0.873 total time= 0.6s  
[CV 1/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.857 total time= 0.5s  
[CV 2/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.830 total time= 0.5s  
[CV 3/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.801 total time= 0.5s  
[CV 4/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.818 total time= 0.5s  
[CV 5/5] END ..C=1000, gamma=0.0001, kernel=rbf;, score=0.825 total time= 0.4s

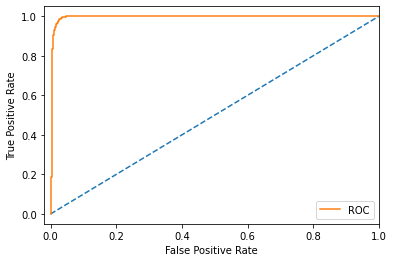
print('Train Score:', grid.score(Xtrain, ytrain))  
print('Test Score:', grid.score(Xtest, ytest))  
  
cm = confusion\_matrix(ytest, grid\_predictions, labels=[0,1])  
df\_cm = pd.DataFrame(cm, index=[i for i in ['No','Yes']],  
 columns = [i for i in ['No', 'Yes']])  
plt.figure(figsize=(7,5))  
sns.heatmap(df\_cm, annot = True, fmt = 'g');

Train Score: 0.9975716367168529  
Test Score: 0.9810771470160117



print(classification\_report(ytest, grid\_predictions))  
  
prob = grid.predict\_proba(Xtest)[:,1]  
  
fpr, tpr, threshold = roc\_curve(ytest, prob)  
  
plt.plot([0,1],[0,1], linestyle ='--')  
  
plt.plot(fpr, tpr, label='ROC')  
  
\_=plt.xlabel('False Positive Rate')  
  
\_=plt.ylabel('True Positive Rate')  
  
\_=plt.xlim([-0.02,1])  
  
\_=plt.legend(loc='lower right')  
  
svm\_score = roc\_auc\_score(ytest, prob)  
  
print('AUC:', roc\_auc\_score(ytest, prob))

precision recall f1-score support  
  
 0 1.00 0.96 0.98 340  
 1 0.97 1.00 0.98 347  
  
 accuracy 0.98 687  
 macro avg 0.98 0.98 0.98 687  
weighted avg 0.98 0.98 0.98 687  
  
AUC: 0.9958806577385997



# Conclusion  
  
# The data is originally contains categorical and numerical data. I have converted categorical data into numerical data using LabelEncoder.   
# The data is imbalanced. I have used SMOTE to balance the data.  
# I used normalization on numerical data and standardization on categorical data.  
  
# These preprocessing steps are very important to get good results.   
# Otherwise, the model will not be able to predict the outcome correctly.  
# After preparing the data, the rest is just applying different models and comparing the results.  
# The models are imported from sklearn library.  
  
  
# ROC curve is used to evaluate the performance of the model. AUC is used to compare the performance of different models.  
# When I suspected overfitting I used cross validation to check the accuracy of the model.  
# Confusion matrix and classification report are used to get better insights about the model.  
# The metric used to evaluate the performance of the models are balanced accuracy.  
  
  
  
# I have used 6 different models to predict the outcome.  
# The application of some models are very similar. However, SVM is a little bit different.  
# To be able to use SVM, I had to find the best parameters using GridSearchCV.  
# The best parameters are C=1000, gamma=0.0001, and kernel='rbf'.  
# However SVM overfitted our data and the accuracy was unrealistic.   
  
# We can see that the best model is KNN with 0.85 score, the rest of the models are a bit underperformed or overfit the data.  
# I picked KNN to predict the outcome of the test data in best\_model.ipynb file.  
# The best model is saved in knn10.sav file and the predictions saved in predictions.csv file.  
  
# In the test dataset we do not have a Y axis which is label of the data.  
# So we can not evaluate the performance of the model. However, based on the train,test performance of our KNN model we can assume we have 85% accuracy.