**Introduction**

Decision trees are one of the simplest forms of supervised learning, that is a number of predictor variables are used to predict a value or classify a data (assign it a particular class in our case whether a country is a Democracy or is not in this case it is an anoncracy, autocracy or unknown, this categorization is based on the polity score). The class Democracy is based on the following split of polity scores in the Gapminder dataset:

def polityscore\_cat (row):

if (row['polityscore'] >=6 and row['polityscore'] <= 10 ) :

return 'Democracy'

elif (row['polityscore'] >=-5 and row['polityscore'] <= 5 ) :

return 'Anocracy'

elif (row['polityscore'] >=-10 and row['polityscore'] <= -6 ) :

return 'Autocracy'

else :

return 'NA'

Countries between 6 and 10 are democracies.

Countries between -5 and 5 are anoncracies (these are essentially as states in various degrees of failure, i.e non functioning democracies, low level insurgencies etc).

Countries between -10 and -6 are autocracies (absolute monarchies, dictatorships etc, authoritarian regimes).

Then the different categories are decoded to 1 for democracy and everything else gets decoded as 0

(i.e anoncracies, autocracies and non-rated countries)

The predictor variables used are:

'incomeperperson','armedforcesrate','femaleemployrate',

'internetuserate',

'European', 'African','Asian', 'Mid\_East',

'North\_American', 'Carribean\_Central\_America',

'OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American',

'Is\_Nato\_Country','Eu\_Member'

The predictor variables; 'incomeperperson','armedforcesrate','femaleemployrate', 'internetuserate' are non-derived and supplied as part of the Gapminder dataset.

The predictor variables are calculated based on the country name and are derived from Wikipedia, They are scored 1 meaning the condition is true or 0, so in the case of African predictor variable a 1 indicates is African country and 0 indicates it’s not:

'European': is a European country.

'African': is a African country (Egypt is included).

'Asian': is an Asian Country (Note caucuses countries (Georgia, Armenia) and Cyprus are include as European).

'Mid\_East' : Is a middle eastern country.

'North\_American' : Is in North America (Includes central America and Caribbean, essentially the CONCACAF countries).

'Carribean\_Central\_America': Is in America and Caribbean.

'OPEC' Is a member of OPEC or is an observer member (Eritrea).

'Arab\_League' is a member of the Arab League.

'ASEAN\_ARF' is a member of ASEAN (ASSOCIATION OF SOUTHEAST ASIAN NATIONS) regional forum members.

'South\_American' is located in South America (below Panama).

'Is\_Nato\_Country' is a member of NATO.

'Eu\_Member' is a member of the EU.

Advantages of a decision tree are they are interpretable by humans, some can use continuous and categorical variables and can be used to model intricate and difficult relationships.

Disadvantages They can be computationally costly especially when using large datasets.

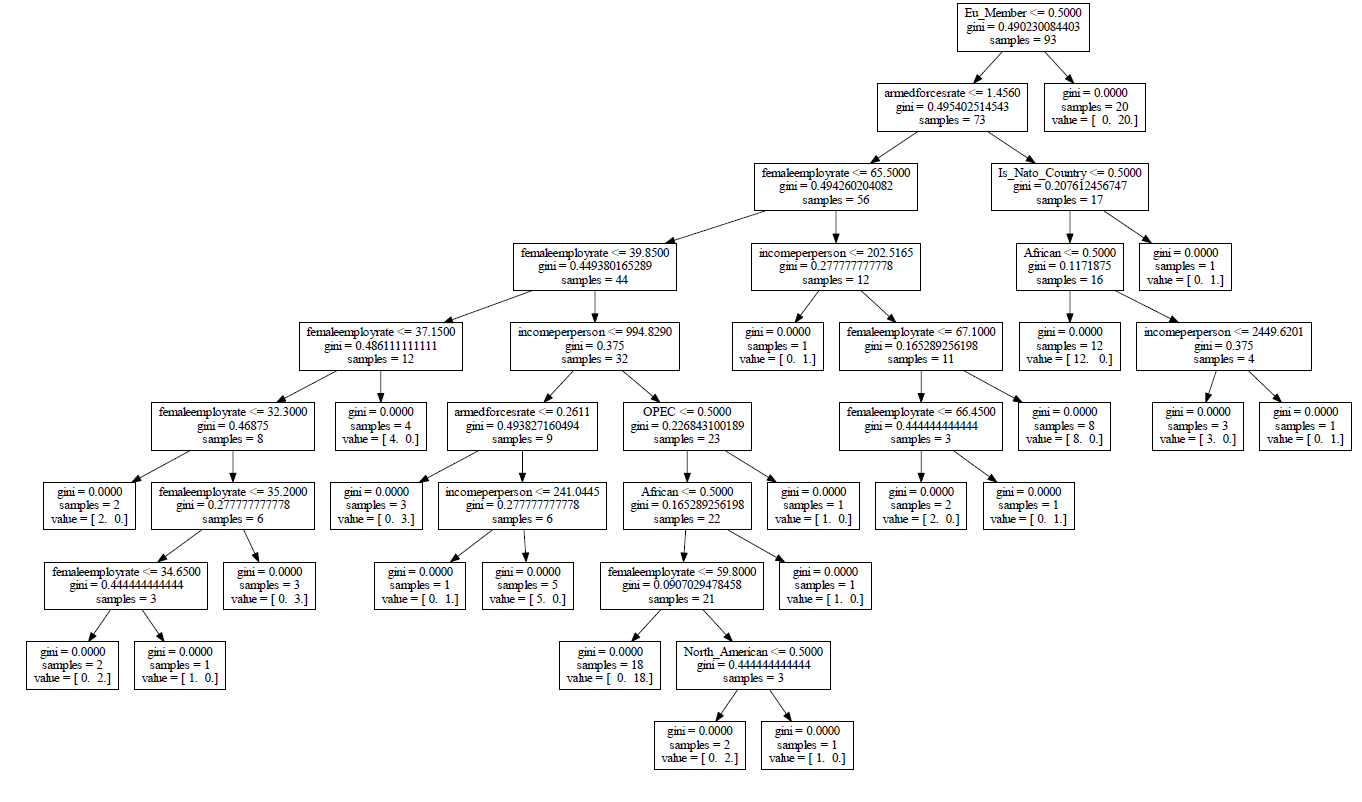
**How a decision tree works?**

Creation of a decision tree takes place from class-labeled training tuples, these were described in the previous section. A decision tree is a flow-chart diagram, where all internal (non-leaf) node denotes a test on an attribute, each branch shows the result of a test, and each leaf (or terminal) node shows a class label. The highest node in a tree is referred to as the root node. Various information criterion can be used to calculate the optimal split:

1. Gini impurity
2. Information gain
3. Variance Reduction

**Interpretation of the tree**

The derived tree is shown below:



The predictive accuracy achiever by our decision tree was not very high 55%, (Predictive accuracy: TP+TN/(TP+TN+FP+FN)), where :

TP : True positive

TN : True negative

FP : False positive

FN : False negative

The confusion matric is shown below:

ACTUAL Positive Negative

Positive ## array ([[21, 11], TP FP PRED

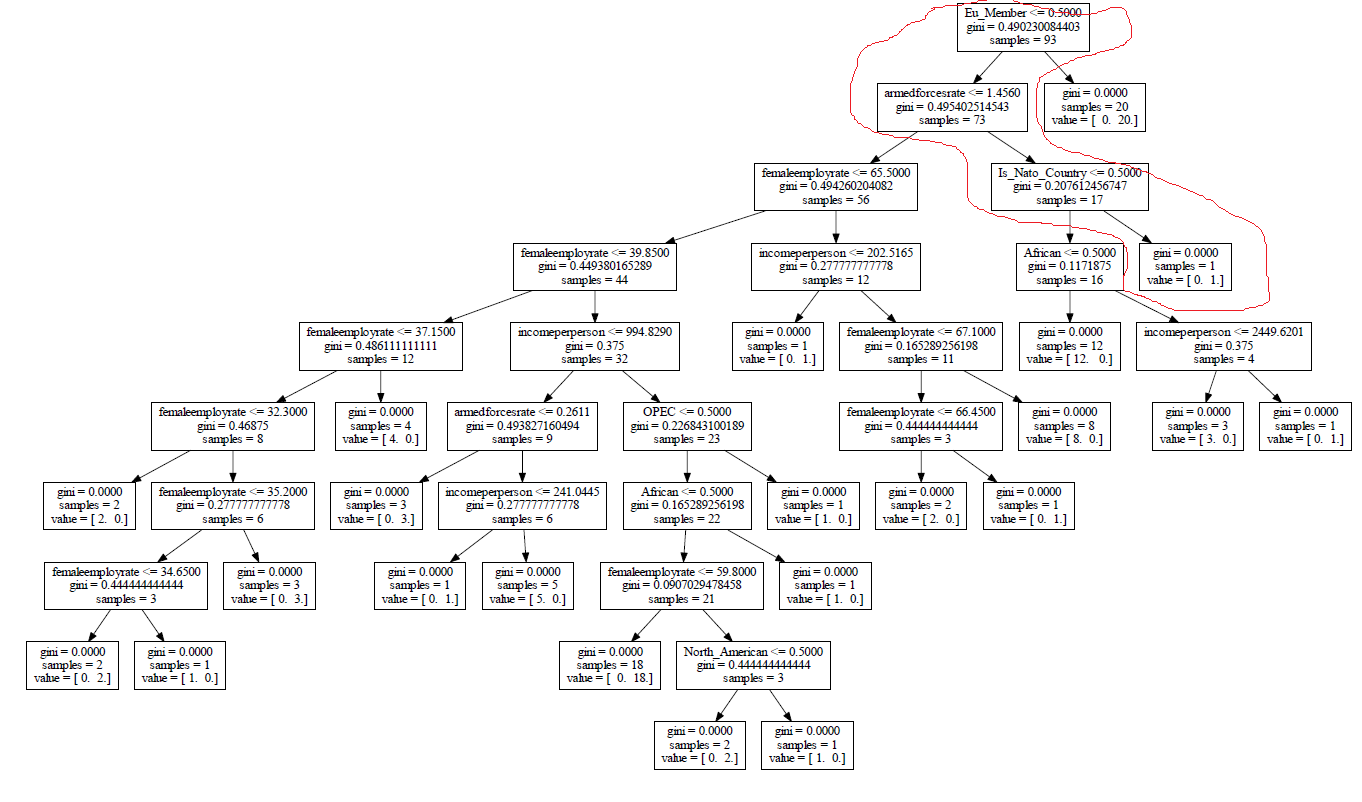
Negative## [17, 14]]) FN TN

##sklearn.metrics.accuracy\_score(tar\_test, predictions)

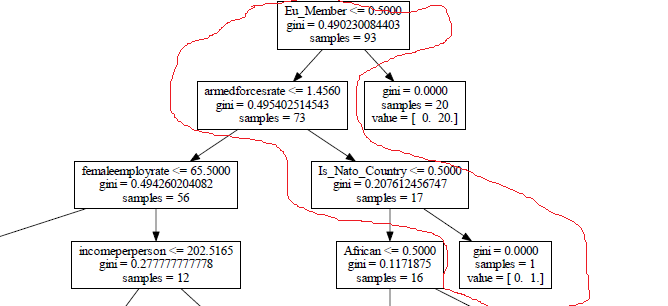
##Out[386]: 0.55555555555555558

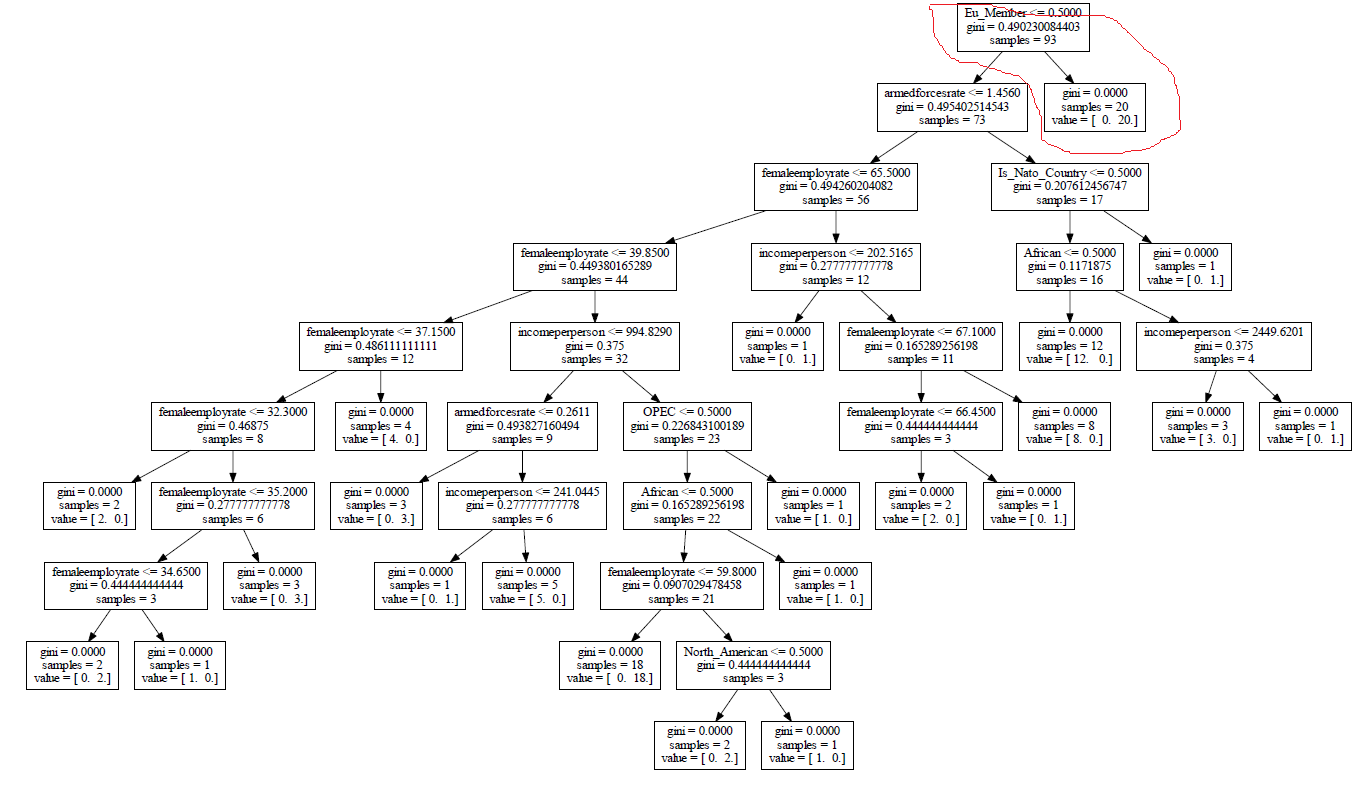
Pred.accuracy= (21+14)/(21+14+11+17)

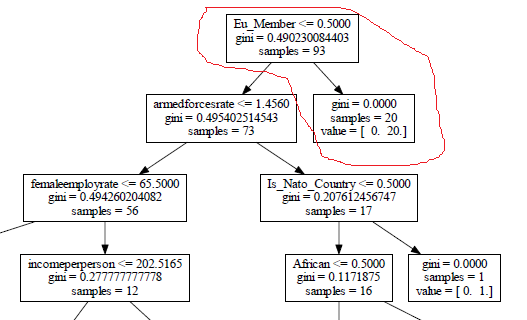
Creation of a decision tree takes place from class-labeled training tuples, these were described in the previous section. A decision tree is a flow-chart diagram where all internal (non-leaf) node denotes a test on an attribute, each branch shows the result of a test, and each leaf (or terminal) node shows a class label. The highest node in a tree is referred to as the root node. We can view all data (all rows) starting in a single bin at the top of the tree. All features in the subset are used to see how the data can be partitioned in the most informatively effective manner – which utilizes the gini measure by default, but this can be configured to use entropy at your preference.



At the root of the tree, we see the most informative condition is EUmembership <=0.5. If this condition is true, we follow the left branch of the tree to get to the 73. Following the right brnach where the condition is not true that is to say they are EU\_Membership is greater than 0.5 i.e they are EU\_member states they are Democracies.

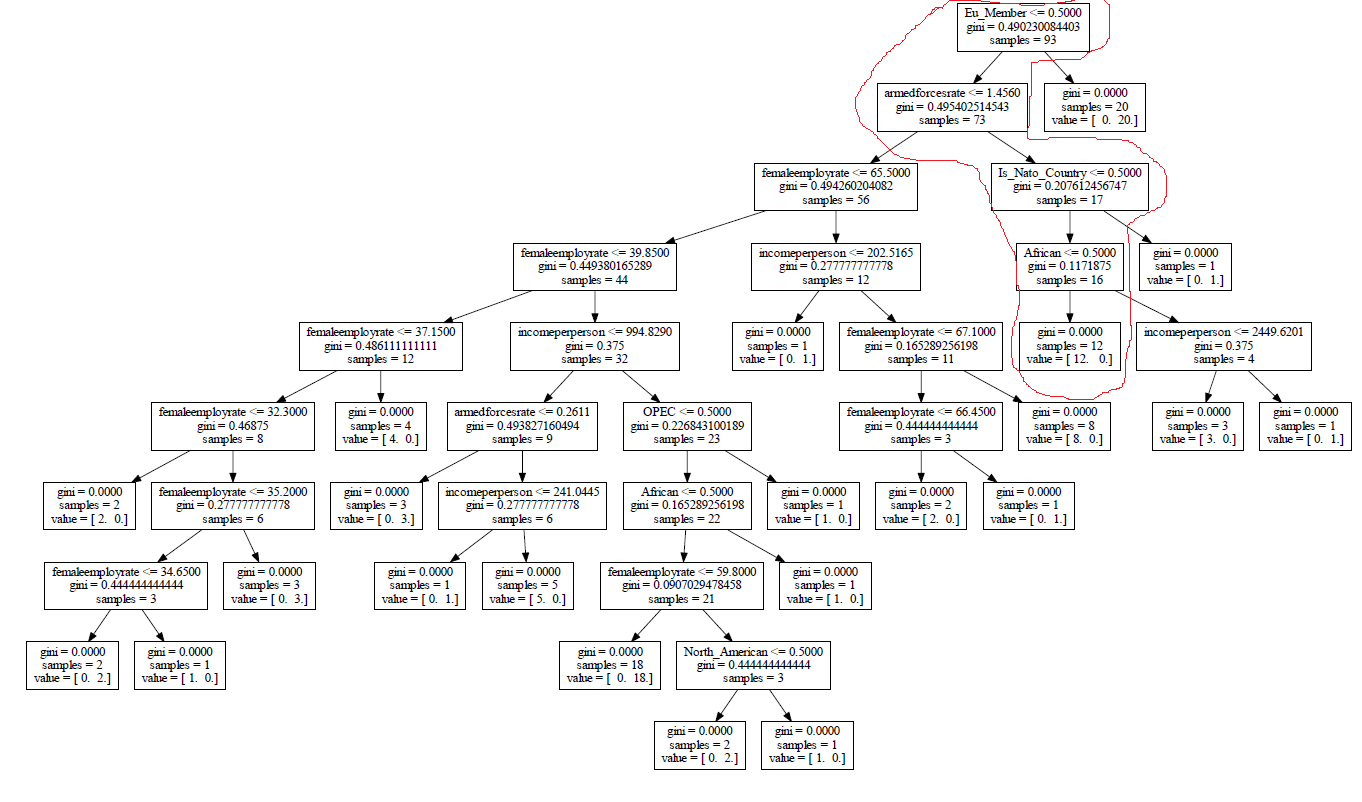


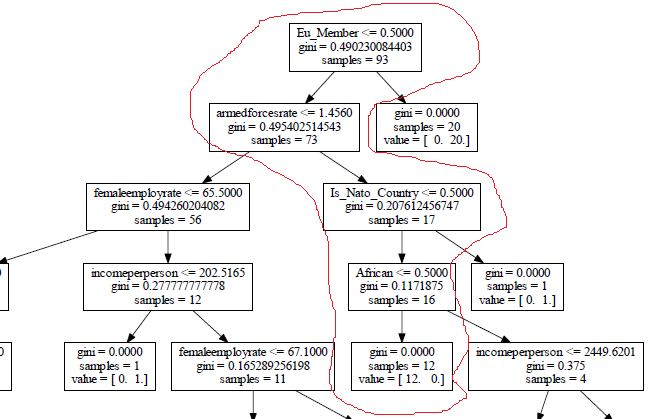




The other 73 samples, of the 93 total, go to the left bin. On the left branch at the next node if we go right and the armed forces rate is over 1.456 and they are a NATO country (Is\_Nato\_Country >= 0.5) they are also a democracy.

If the armed forces rate is over 1.4560 and they are not a NATO Country or not an African country they are Not a Democracy.





This splitting continues until such a stage that the split spawns a bin with only a single class – for example the bin described above is not split again and results in 12 samples of value 0, class 'Not a Democracy'.

**Code:**

# -\*- coding: utf-8 -\*-

"""

Created on Fri Jun 03 12:27:51 2016

@author: Peter

"""

import os

import pandas

import numpy

import sklearn

import matplotlib

import matplotlib.pyplot as plt

import sys; print(sys.path)

from seaborn import \*

import seaborn as sns

import ggplot

from ggplot import \*

import scipy

from pandas import Series, DataFrame

import pandas as pd

import numpy as np

import os

import matplotlib.pylab as plt

from sklearn.cross\_validation import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import classification\_report

import sklearn.metrics

import pydot

import graphviz

apath='C:\Users\Peter\Desktop\Gapminder'

print(apath)

os.chdir('C:\Users\Peter\Desktop\Gapminder')

##check the directory has changed

os.getcwd()

##read in the file

data = pandas.read\_csv('gapminder.csv', low\_memory=False)

##lets convert the data to numeric

data['incomeperperson'] = data['incomeperperson'].convert\_objects(convert\_numeric=True)

data['alcconsumption'] = data['alcconsumption'].convert\_objects(convert\_numeric=True)

data['armedforcesrate'] = data['armedforcesrate'].convert\_objects(convert\_numeric=True)

data['breastcancerper100th'] = data['breastcancerper100th'].convert\_objects(convert\_numeric=True)

data['co2emissions'] = data['co2emissions'].convert\_objects(convert\_numeric=True)

data['femaleemployrate'] = data['femaleemployrate'].convert\_objects(convert\_numeric=True)

data['hivrate'] = data['hivrate'].convert\_objects(convert\_numeric=True)

data['internetuserate'] = data['internetuserate'].convert\_objects(convert\_numeric=True)

data['lifeexpectancy'] = data['lifeexpectancy'].convert\_objects(convert\_numeric=True)

data['oilperperson'] = data['oilperperson'].convert\_objects(convert\_numeric=True)

data['polityscore'] = data['polityscore'].convert\_objects(convert\_numeric=True)

data['relectricperperson'] = data['relectricperperson'].convert\_objects(convert\_numeric=True)

data['suicideper100th'] = data['suicideper100th'].convert\_objects(convert\_numeric=True)

data['employrate'] = data['employrate'].convert\_objects(convert\_numeric=True)

data['urbanrate'] = data['urbanrate'].convert\_objects(convert\_numeric=True)

bins = [0, 1000, 5000, 10000, 20000,50000,200000]

group\_names = ['Very Low Income,0-1000', 'Low Income,1000-5000', 'Okay Income,5000-10000', 'Good Income,10000-20000','Great Income,20000-50000','50,000-200,000']

categories = pandas.cut(data['incomeperperson'], bins, labels=group\_names)

data['categories'] = pandas.cut(data['incomeperperson'], bins, labels=group\_names)

##data.dtypes chk

##now encode european countries

def EUROPEAN (row):

if row['country'] == 'Albania' :

return 'Europe'

elif row['country'] == 'Andorra' :

return 'Europe'

elif row['country'] == 'Armenia' :

return 'Europe'

elif row['country'] == 'Azerbaijan' :

return 'Europe'

elif row['country'] == 'Austria' :

return 'Europe'

elif row['country'] == 'Belarus' :

return 'Europe'

elif row['country'] == 'Belgium' :

return 'Europe'

elif row['country'] == 'Bosnia' :

return 'Europe'

elif row['country'] == 'Bulgaria' :

return 'Europe'

elif row['country'] == 'Croatia' :

return 'Europe'

elif row['country'] == 'Cyprus' :

return 'Europe'

elif row['country'] == 'Czech Republic' :

return 'Europe'

elif row['country'] == 'Denmark' :

return 'Europe'

elif row['country'] == 'Estonia' :

return 'Europe'

elif row['country'] == 'Finland' :

return 'Europe'

elif row['country'] == 'France' :

return 'Europe'

elif row['country'] == 'Georgia' :

return 'Europe'

elif row['country'] == 'Germany' :

return 'Europe'

elif row['country'] == 'Greece' :

return 'Europe'

elif row['country'] == 'Hungary' :

return 'Europe'

elif row['country'] == 'Iceland' :

return 'Europe'

elif row['country'] == 'Ireland' :

return 'Europe'

elif row['country'] == 'Italy' :

return 'Europe'

elif row['country'] == 'Kazakhstan' :

return 'Europe'

elif row['country'] == 'Kosovo' :

return 'Europe'

elif row['country'] == 'Latvia' :

return 'Europe'

elif row['country'] == 'Liechtenstein' :

return 'Europe'

elif row['country'] == 'Lithuania' :

return 'Europe'

elif row['country'] == 'Luxembourg' :

return 'Europe'

elif row['country'] == 'Macedonia' :

return 'Europe'

elif row['country'] == 'Malta' :

return 'Europe'

elif row['country'] == 'Moldova' :

return 'Europe'

elif row['country'] == 'Monaco' :

return 'Europe'

elif row['country'] == 'Montenegro' :

return 'Europe'

elif row['country'] == 'Netherlands' :

return 'Europe'

elif row['country'] == 'Norway' :

return 'Europe'

elif row['country'] == 'Poland' :

return 'Europe'

elif row['country'] == 'Portugal' :

return 'Europe'

elif row['country'] == 'Romania' :

return 'Europe'

elif row['country'] == 'Russia' :

return 'Europe'

elif row['country'] == 'San Marino' :

return 'Europe'

elif row['country'] == 'Serbia' :

return 'Europe'

elif row['country'] == 'Slovak Republic' :

return 'Europe'

elif row['country'] == 'Slovenia' :

return 'Europe'

elif row['country'] == 'Spain' :

return 'Europe'

elif row['country'] == 'Sweden' :

return 'Europe'

elif row['country'] == 'Switzerland' :

return 'Europe'

elif row['country'] == 'Turkey' :

return 'Europe'

elif row['country'] == 'Ukraine' :

return 'Europe'

elif row['country'] == 'United Kingdom' :

return 'Europe'

else :

return 'Not\_In\_Europe'

data['European'] = data.apply (lambda row: EUROPEAN (row),axis=1)

##check it worked

##'''

##Out[24]:

##Europe 45

##Not-In-Europe 168

##dtype: int64

##'''

data['European'].value\_counts(sort=False, dropna=False)

data['country']

##checked working

def African (row):

if row['country'] == 'Algeria' :

return 'Africa'

elif row['country'] == 'Angola' :

return 'Africa'

elif row['country'] == 'Benin' :

return 'Africa'

elif row['country'] == 'Botswana' :

return 'Africa'

elif row['country'] == 'Burkina Faso' :

return 'Africa'

elif row['country'] == 'Burundi' :

return 'Africa'

elif row['country'] == 'Cameroon' :

return 'Africa'

elif row['country'] == 'Cape Verde' :

return 'Africa'

elif row['country'] == 'Central African Republic' :

return 'Africa'

elif row['country'] == 'Chad' :

return 'Africa'

elif row['country'] == 'Comoros' :

return 'Africa'

elif row['country'] == 'Congo, Dem. Rep.' :

return 'Africa'

elif row['country'] == 'Congo, Rep.' :

return 'Africa'

elif row['country'] == 'Djibouti' :

return 'Africa'

elif row['country'] == 'Equatorial Guinea' :

return 'Africa'

elif row['country'] == 'Eritrea' :

return 'Africa'

elif row['country'] == 'Ethiopia' :

return 'Africa'

elif row['country'] == 'Egypt' :

return 'Africa'

elif row['country'] == 'Gabon' :

return 'Africa'

elif row['country'] == 'Gambia' :

return 'Africa'

elif row['country'] == 'Ghana' :

return 'Africa'

elif row['country'] == "Cote d'Ivoire":

return 'Africa'

elif row['country'] == "Guinea-Bissau":

return 'Africa'

elif row['country'] == "Guinea":

return 'Africa'

elif row['country'] == "Kenya":

return 'Africa'

elif row['country'] == "Lesotho":

return 'Africa'

elif row['country'] == "Liberia":

return 'Africa'

elif row['country'] == "Libya":

return 'Africa'

elif row['country'] == "Madagascar":

return 'Africa'

elif row['country'] == "Malawi":

return 'Africa'

elif row['country'] == "Mali":

return 'Africa'

elif row['country'] == "Mauritania":

return 'Africa'

elif row['country'] == "Mauritius":

return 'Africa'

elif row['country'] == "Morocco":

return 'Africa'

elif row['country'] == "Mozambique":

return 'Africa'

elif row['country'] == "Namibia":

return 'Africa'

elif row['country'] == "Niger":

return 'Africa'

elif row['country'] == "Nigeria":

return 'Africa'

elif row['country'] == "Rwanda":

return 'Africa'

elif row['country'] == 'Sao Tome and Principe':

return 'Africa'

elif row['country'] == 'Senegal':

return 'Africa'

elif row['country'] == 'Seychelles':

return 'Africa'

elif row['country'] == 'Sierra Leone':

return 'Africa'

elif row['country'] == 'Somalia':

return 'Africa'

elif row['country'] == 'South Sudan':

return 'Africa'

elif row['country'] == 'South Africa':

return 'Africa'

elif row['country'] == 'Sudan':

return 'Africa'

elif row['country'] == 'Swaziland':

return 'Africa'

elif row['country'] == 'Tanzania':

return 'Africa'

elif row['country'] == 'Togo':

return 'Africa'

elif row['country'] == 'Tunisia':

return 'Africa'

elif row['country'] == 'Uganda':

return 'Africa'

elif row['country'] == 'Zambia':

return 'Africa'

elif row['country'] == 'Zimbabwe':

return 'Africa'

elif row['country'] == 'Somaliland':

return 'Africa'

else :

return 'Not\_In\_Africa'

data['African'] = data.apply (lambda row: African (row),axis=1)

data['African'].value\_counts(sort=False, dropna=False)

def Asian (row):

if row['country'] == 'Afganistan' :

return 'Asia'

elif row['country'] == 'Armenia' :

return 'Asia'

elif row['country'] == 'Bahrain' :

return 'Asia'

elif row['country'] == 'Bangladesh' :

return 'Asia'

elif row['country'] == 'Bhutan' :

return 'Asia'

elif row['country'] == 'Brunei' :

return 'Asia'

elif row['country'] == 'Cambodia' :

return 'Asia'

elif row['country'] == 'China' :

return 'Asia'

elif row['country'] == 'Georgia' :

return 'Asia'

elif row['country'] == 'India' :

return 'Asia'

elif row['country'] == 'Iran' :

return 'Asia'

elif row['country'] == 'Indonesia' :

return 'Asia'

elif row['country'] == 'Iraq' :

return 'Asia'

elif row['country'] == 'Israel' :

return 'Asia'

elif row['country'] == 'Japan' :

return 'Asia'

elif row['country'] == 'Jordan' :

return 'Asia'

elif row['country'] == 'Kazakhstan' :

return 'Asia'

elif row['country'] == 'Korea, Dem. Rep.' :

return 'Asia'

elif row['country'] == 'Korea, Rep.' :

return 'Asia'

elif row['country'] == 'Kuwait' :

return 'Asia'

elif row['country'] == 'Kyrgyzstan' :

return 'Asia'

elif row['country'] == 'Laos' :

return 'Asia'

elif row['country'] == 'Lebanon' :

return 'Asia'

elif row['country'] == 'Malaysia' :

return 'Asia'

elif row['country'] == 'Maldives' :

return 'Asia'

elif row['country'] == 'Mongolia' :

return 'Asia'

elif row['country'] == 'Myanmar' :

return 'Asia'

elif row['country'] == 'Nepal' :

return 'Asia'

elif row['country'] == 'Oman' :

return 'Asia'

elif row['country'] == 'Pakistan' :

return 'Asia'

elif row['country'] == 'Philippines' :

return 'Asia'

elif row['country'] == 'Qatar' :

return 'Asia'

elif row['country'] == 'Saudi Arabia' :

return 'Asia'

elif row['country'] == 'Singapore' :

return 'Asia'

elif row['country'] == 'Sri Lanka' :

return 'Asia'

elif row['country'] == 'Syria' :

return 'Asia'

elif row['country'] == 'Tajikistan' :

return 'Asia'

elif row['country'] == 'Thailand' :

return 'Asia'

elif row['country'] == 'Timor-Leste' :

return 'Asia'

elif row['country'] == 'Turkey' :

return 'Asia'

elif row['country'] == 'Turkmenistan' :

return 'Asia'

elif row['country'] == 'United Arab Emirates' :

return 'Asia'

elif row['country'] == 'Uzbekistan' :

return 'Asia'

elif row['country'] == 'Vietnam' :

return 'Asia'

elif row['country'] == 'Yemen' :

return 'Asia'

else :

return 'Not\_In\_Asia'

data['Asian'] = data.apply (lambda row: Asian (row),axis=1)

data['Asian'].value\_counts(sort=False, dropna=False)

##data[['country','incomeperperson','polityscore\_cat']][(data.European=='Europe') & (data.polityscore\_cat!='NA') ]

##data[(data.Asian=='Asian')]

def Mid\_East (row):

if row['country'] == 'Bahrain' :

return 'Middle\_East'

elif row['country'] == 'Cyprus' :

return 'Middle\_East'

elif row['country'] == 'Egypt' :

return 'Middle\_East'

elif row['country'] == 'Iran' :

return 'Middle\_East'

elif row['country'] == 'Iraq' :

return 'Middle\_East'

elif row['country'] == 'Israel' :

return 'Middle\_East'

elif row['country'] == 'Jordan' :

return 'Middle\_East'

elif row['country'] == 'Kuwait' :

return 'Middle\_East'

elif row['country'] == 'Lebanon' :

return 'Middle\_East'

elif row['country'] == 'Oman' :

return 'Middle\_East'

elif row['country'] == 'Qatar' :

return 'Middle\_East'

elif row['country'] == 'Saudi Arabia' :

return 'Middle\_East'

elif row['country'] == 'Syria' :

return 'Middle\_East'

elif row['country'] == 'Turkey' :

return 'Middle\_East'

elif row['country'] == 'United Arab Emirates' :

return 'Middle\_East'

elif row['country'] == 'Yemen' :

return 'Middle\_East'

else :

return 'Not\_In\_Middle\_East'

data['Mid\_East'] = data.apply (lambda row: Mid\_East (row),axis=1)

data['Mid\_East'].value\_counts(sort=False, dropna=False)

def North\_American (row):

if row['country'] == 'Antigua and Barbuda' :

return 'North\_America'

elif row['country'] == 'Bahamas' :

return 'North\_America'

elif row['country'] == 'Barbados' :

return 'North\_America'

elif row['country'] == 'Belize' :

return 'North\_America'

elif row['country'] == 'Canada' :

return 'North\_America'

elif row['country'] == 'Costa Rica' :

return 'North\_America'

elif row['country'] == 'Cuba' :

return 'North\_America'

elif row['country'] == 'Dominica' :

return 'North\_America'

elif row['country'] == 'Dominican Republic' :

return 'North\_America'

elif row['country'] == 'El Salvador' :

return 'North\_America'

elif row['country'] == 'Grenada' :

return 'North\_America'

elif row['country'] == 'Guatemala' :

return 'North\_America'

elif row['country'] == 'Haiti' :

return 'North\_America'

elif row['country'] == 'Honduras' :

return 'North\_America'

elif row['country'] == 'Jamaica' :

return 'North\_America'

elif row['country'] == 'Mexico' :

return 'North\_America'

elif row['country'] == 'Nicaragua' :

return 'North\_America'

elif row['country'] == 'Panama' :

return 'North\_America'

elif row['country'] == 'Panama' :

return 'North\_America'

elif row['country'] == 'Saint Kitts and Nevis' :

return 'North\_America'

elif row['country'] == 'Saint Lucia' :

return 'North\_America'

elif row['country'] == 'Saint Vincent and the Grenadines' :

return 'North\_America'

elif row['country'] == 'Trinidad and Tobago' :

return 'North\_America'

elif row['country'] == 'United States' :

return 'North\_America'

else :

return 'Not\_In\_North\_America'

data['North\_American'] = data.apply (lambda row: North\_American (row),axis=1)

data['North\_American'].value\_counts(sort=False, dropna=False)

def Carribean\_Central\_America (row):

if row['country'] == 'Antigua and Barbuda' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Bahamas' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Barbados' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Belize' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Costa Rica' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Cuba' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Dominica' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Dominican Republic' :

return 'Carribean\_Central\_American'

elif row['country'] == 'El Salvador' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Grenada' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Guatemala' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Haiti' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Honduras' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Jamaica' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Nicaragua' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Panama' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Saint Kitts and Nevis' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Saint Lucia' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Saint Vincent and the Grenadines' :

return 'Carribean\_Central\_American'

elif row['country'] == 'Trinidad and Tobago' :

return 'Carribean\_Central\_American'

else :

return 'Not\_In\_Carribean\_Central\_American'

data['Carribean\_Central\_America'] = data.apply (lambda row: Carribean\_Central\_America (row),axis=1)

data['Carribean\_Central\_America'].value\_counts(sort=False, dropna=False)

##Algeria, Angola, Ecuador, Iran, Iraq, Kuwait, Libya, Nigeria, Qatar, Saudi Arabia, United Arab Emirates and Venezuela

def OPEC (row):

if row['country'] == 'Algeria' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Angola' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Ecuador' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Iran' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Iraq' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Kuwait' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Libya' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Nigeria' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Qatar' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Saudi Arabia' :

return 'OPEC\_MEMBER'

elif row['country'] == 'United Arab Emirates' :

return 'OPEC\_MEMBER'

elif row['country'] == 'Venezuela' :

return 'OPEC\_MEMBER'

else :

return 'Not\_In\_OPEC'

data['OPEC'] = data.apply (lambda row: OPEC (row),axis=1)

data['OPEC'].value\_counts(sort=False, dropna=False)

def Arab\_League (row):

if row['country'] == 'Algeria' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Bahrain' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Comoros' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Djibouti' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Egypt' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Iraq' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Jordan' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Kuwait' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Lebanon' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Libya' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Mauritania' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Morocoo' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Oman' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'West Bank and Gaza' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Qatar' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Saudi Arabia' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Somalia' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Sudan' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Syria' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Tunisia' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'United Arab Emirates' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Yemen' :

return 'Arab\_League\_MEMBER'

elif row['country'] == 'Eritrea' :

return 'Arab\_League\_MEMBER'

else :

return 'Not\_In\_Arab\_League'

##

data['Arab\_League'] = data.apply (lambda row: Arab\_League (row),axis=1)

data['Arab\_League'].value\_counts(sort=False, dropna=False)

##ASEAN is a regional grouping with security, economic and social aspects

def ASEAN\_ARF (row):

if row['country'] == 'Australia' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Bangladesh' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Brunei' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Cambodia' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Canada' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'China' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'India' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Indonesia' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Japan' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Korea, Dem. Rep.' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Korea, Rep.' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Laos' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Malaysia' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Myanmar' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Mongolia' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'New Zealand' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Pakistan' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Papua New Guinea' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Phillipines' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Russian Federation' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Singapore' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Sri Lanka' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Thailand' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Timor-Leste' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'United States' :

return 'ASEAN\_ARF\_MEMBER'

elif row['country'] == 'Vietnam' :

return 'ASEAN\_ARF\_MEMBER'

else :

return 'Not\_In\_ASEAN\_ARF'

data['ASEAN\_ARF'] = data.apply (lambda row: ASEAN\_ARF (row),axis=1)

data['ASEAN\_ARF'].value\_counts(sort=False, dropna=False)

def South\_American (row):

if row['country'] == 'Argentina' :

return 'South\_America'

elif row['country'] == 'Bolivia' :

return 'South\_America'

elif row['country'] == 'Brazil' :

return 'South\_America'

elif row['country'] == 'Chile' :

return 'South\_America'

elif row['country'] == 'Colombia' :

return 'South\_America'

elif row['country'] == 'Ecuador' :

return 'South\_America'

elif row['country'] == 'Guyana' :

return 'South\_America'

elif row['country'] == 'Paraguay' :

return 'South\_America'

elif row['country'] == 'Peru' :

return 'South\_America'

elif row['country'] == 'Suriname' :

return 'South\_America'

elif row['country'] == 'Uruguay' :

return 'South\_America'

elif row['country'] == 'Venezuala' :

return 'South\_America'

else :

return 'Not\_South\_America'

data['South\_American'] = data.apply (lambda row: South\_American (row),axis=1)

data['South\_American'].value\_counts(sort=False, dropna=False)

##

###http://www.nato.int/cps/en/natohq/topics\_52044.htm

##NATO data

Nato\_Countries = pandas.DataFrame({ 'country' : ('Albania','Belgium','Bulgaria','Canada','Croatia','Czech Republic','Denmark','Estonia','France','Germany','Greece','Hungary','Iceland','Italy','Latvia','Lithuania','Luxembourg','Netherlands','Norway','Poland','Portugal','Romania','Slovak Republic','Slovenia','Spain','Turkey','United Kingdom','United States'),

'Year\_Joined' : (2009,1949,2004,1949,2009,1999,1949,2004,1949,1955,1952,1999,1949,1949,2004,2004,1949,1949,1949,1999,1949,2004,2004,2004,1982,1952,1949,1949),

'Is\_Nato\_Country' : 'Nato\_Member'

})

##Enhanced data join NATO data

data.columns.values

data=pandas.merge(data, Nato\_Countries,how='left',on='country')

##data.columns.values

##check that all column values have been added

data['Is\_Nato\_Country']=data['Is\_Nato\_Country'].fillna('Not\_in\_Nato')

##

data['Is\_Nato\_Country'].value\_counts(sort=False, dropna=False)

data.columns.values

##year joined needs to be renamed

data.rename(columns={'Year\_Joined': 'Year\_Joined\_Nato'}, inplace=True)

## change columns names

data.columns.values

def EUMEMBER (row):

if row['country'] == 'Austria' :

return 'EU'

elif row['country'] == 'Belgium' :

return 'EU'

elif row['country'] == 'Bulgaria' :

return 'EU'

elif row['country'] == 'Croatia' :

return 'EU'

elif row['country'] == 'Cyprus' :

return 'EU'

elif row['country'] == 'Czech Republic' :

return 'EU'

elif row['country'] == 'Denmark' :

return 'EU'

elif row['country'] == 'Estonia' :

return 'EU'

elif row['country'] == 'Finland' :

return 'EU'

elif row['country'] == 'France' :

return 'EU'

elif row['country'] == 'Germany' :

return 'EU'

elif row['country'] == 'Greece' :

return 'EU'

elif row['country'] == 'Hungary' :

return 'EU'

elif row['country'] == 'Ireland' :

return 'EU'

elif row['country'] == 'Italy' :

return 'EU'

elif row['country'] == 'Latvia' :

return 'EU'

elif row['country'] == 'Lithuania' :

return 'EU'

elif row['country'] == 'Luxembourg' :

return 'EU'

elif row['country'] == 'Malta' :

return 'EU'

elif row['country'] == 'Netherlands' :

return 'EU'

elif row['country'] == 'Poland' :

return 'EU'

elif row['country'] == 'Portugal' :

return 'EU'

elif row['country'] == 'Romania' :

return 'EU'

elif row['country'] == 'Slovak Republic' :

return 'EU'

elif row['country'] == 'Slovenia' :

return 'EU'

elif row['country'] == 'Spain' :

return 'EU'

elif row['country'] == 'Sweden' :

return 'EU'

elif row['country'] == 'United Kingdom' :

return 'EU'

else :

return 'Not\_In\_EU'

data['Eu\_Member'] = data.apply (lambda row: EUMEMBER (row),axis=1)

data['Eu\_Member'].value\_counts(sort=False, dropna=False)

data.columns.values

import time

##check how to calc time

print (time.strftime("%Y"))

##write unction to calculate the age of NATO countries based on the current date

def AGE\_YEARS (row):

current\_year=time.strftime("%Y")

if row['Year\_Joined\_Nato'] >0 :

return (int(current\_year)-int(row['Year\_Joined\_Nato']))

else :

return 0

##calculate the age of NATO countries

data['Years\_In\_Nato'] = data.apply (lambda row: AGE\_YEARS (row),axis=1)

##calculate the age of NAto countries

print("distribution of years in NATO for Nato Countries")

pp2=data['Years\_In\_Nato'].value\_counts(sort=False, dropna=False)

print(pp2)

##Mean number of years in NATO by european or Non EU

print("Mean years of countries in NATO for European and Non European Countries")

pp3=data[data['Years\_In\_Nato']>0]['Years\_In\_Nato'].groupby(data['Eu\_Member']).mean()

print(pp3)

##Count of countries in EU who are not in not in NATO

print("Count of countries in NATO for European and Non European Countries")

pp4=data[data['Years\_In\_Nato']>0]['Years\_In\_Nato'].groupby(data['Eu\_Member']).count()

print(pp4)

def EU\_NATO (Nato\_Membership,EU\_Membership):

if Nato\_Membership == 'Nato\_Member' and EU\_Membership== 'EU' :

return 'Nato\_And\_EU'

elif Nato\_Membership == 'Nato\_Member' and EU\_Membership == 'Not-In-EU':

return 'Nato\_Not\_In\_EU'

elif Nato\_Membership != 'Nato\_Member' and EU\_Membership== 'EU' :

return 'Not\_In\_Nato\_In\_EU'

else :

return 'Not\_In\_Nato\_Not\_In\_EU'

EU\_NATO('Nato\_Member','EU')

##test

##apply the function

data['NATO\_EU\_MEMBERSHIP'] = data.apply (lambda row: EU\_NATO(row['Is\_Nato\_Country'],row['Eu\_Member']),axis=1)

data.columns.values

def polityscore\_cat (row):

if (row['polityscore'] >=6 and row['polityscore'] <= 10 ) :

return 'Democracy'

elif (row['polityscore'] >=-5 and row['polityscore'] <= 5 ) :

return 'Anocracy'

elif (row['polityscore'] >=-10 and row['polityscore'] <= -6 ) :

return 'Autocracy'

else :

return 'NA'

##calculate the age of NATO countries

##data['Years\_In\_Nato'] = data.apply (lambda row: AGE\_YEARS (row),axis=1)

data['polityscore\_cat'] = data.apply (lambda row: polityscore\_cat (row),axis=1)

data.columns.values

##array(['country', 'incomeperperson', 'alcconsumption', 'armedforcesrate',

## 'breastcancerper100th', 'co2emissions', 'femaleemployrate',

## 'hivrate', 'internetuserate', 'lifeexpectancy', 'oilperperson',

## 'polityscore', 'relectricperperson', 'suicideper100th',

## 'employrate', 'urbanrate', 'categories', 'European', 'African',

## 'Asian', 'Mid\_East', 'North\_American', 'Carribean\_Central\_America',

## 'OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American',

## 'Is\_Nato\_Country', 'Year\_Joined\_Nato', 'Eu\_Member', 'Years\_In\_Nato',

## 'NATO\_EU\_MEMBERSHIP', 'polityscore\_cat'], dtype=object)

##data = data.drop('Is\_Nato\_Country\_y', 1)

##data.rename(columns={'Is\_Nato\_Country\_x': 'Is\_Nato\_Country'}, inplace=True)

data['European'].value\_counts(sort=False, dropna=False)

data['European'].replace("Europe",1,inplace=True)

data['European'].replace("Not\_In\_Europe",0,inplace=True)

data['African'].value\_counts(sort=False, dropna=False)

data['African'].replace("Africa",1,inplace=True)

data['African'].replace("Not\_In\_Africa",0,inplace=True)

data['African'].value\_counts(sort=False, dropna=False)

data['Asian'].value\_counts(sort=False, dropna=False)

data['Asian'].replace("Asia",1,inplace=True)

data['Asian'].replace("Not\_In\_Asia",0,inplace=True)

data['Asian'].value\_counts(sort=False, dropna=False)

##'Mid\_East'

data['Mid\_East'].value\_counts(sort=False, dropna=False)

data['Mid\_East'].replace("Middle\_East",1,inplace=True)

data['Mid\_East'].replace("Not\_In\_Middle\_East",0,inplace=True)

data['Mid\_East'].value\_counts(sort=False, dropna=False)

data['North\_American'].value\_counts(sort=False, dropna=False)

data['North\_American'].replace("North\_America",1,inplace=True)

data['North\_American'].replace("Not\_In\_North\_America",0,inplace=True)

data['North\_American'].value\_counts(sort=False, dropna=False)

data['Carribean\_Central\_America'].value\_counts(sort=False, dropna=False)

data['Carribean\_Central\_America'].replace("Carribean\_Central\_American",1,inplace=True)

data['Carribean\_Central\_America'].replace("Not\_In\_Carribean\_Central\_American",0,inplace=True)

data['Carribean\_Central\_America'].value\_counts(sort=False, dropna=False)

data['OPEC'].replace("OPEC\_MEMBER",1,inplace=True)

data['OPEC'].replace("Not\_In\_OPEC",0,inplace=True)

data['OPEC'].value\_counts(sort=False, dropna=False)

data['Arab\_League'].value\_counts(sort=False, dropna=False)

data['Arab\_League'].replace("Not\_In\_Arab\_League",0,inplace=True)

data['Arab\_League'].replace("Arab\_League\_MEMBER",1,inplace=True)

data['Arab\_League'].value\_counts(sort=False, dropna=False)

##'ASEAN\_ARF'

data['ASEAN\_ARF'].value\_counts(sort=False, dropna=False)

data['ASEAN\_ARF'].replace("Not\_In\_ASEAN\_ARF",0,inplace=True)

data['ASEAN\_ARF'].replace("ASEAN\_ARF\_MEMBER",1,inplace=True)

data['ASEAN\_ARF'].value\_counts(sort=False, dropna=False)

##'South\_American'

data['South\_American'].value\_counts(sort=False, dropna=False)

data['South\_American'].replace("Not\_South\_America",0,inplace=True)

data['South\_American'].replace("South\_America",1,inplace=True)

data['South\_American'].value\_counts(sort=False, dropna=False)

##'Is\_Nato\_Country'

data['Is\_Nato\_Country'].value\_counts(sort=False, dropna=False)

data['Is\_Nato\_Country'].replace("Not\_in\_Nato",0,inplace=True)

data['Is\_Nato\_Country'].replace("Nato\_Member",1,inplace=True)

data['Is\_Nato\_Country'].value\_counts(sort=False, dropna=False)

##'Eu\_Member'

data['Eu\_Member'].value\_counts(sort=False, dropna=False)

data['Eu\_Member'].replace("Not\_In\_EU",0,inplace=True)

data['Eu\_Member'].replace("EU",1,inplace=True)

data['Eu\_Member'].value\_counts(sort=False, dropna=False)

##'polityscore\_cat'

data['polityscore\_cat'].value\_counts(sort=False, dropna=False)

data['polityscore\_cat'].replace("Anocracy",0,inplace=True)

data['polityscore\_cat'].replace("Autocracy",0,inplace=True)

data['polityscore\_cat'].replace("NA",0,inplace=True)

data['polityscore\_cat'].replace("Democracy",1,inplace=True)

data['polityscore\_cat'].value\_counts(sort=False, dropna=False)

##

data1=data[['incomeperperson','armedforcesrate','femaleemployrate',

'internetuserate',

'European', 'African','Asian', 'Mid\_East',

'North\_American', 'Carribean\_Central\_America',

'OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American',

'Is\_Nato\_Country','Eu\_Member','polityscore\_cat']]

data\_clean1 = data1.dropna() ## drop all na values cant handle nulls

data\_clean1.dtypes

data\_clean.describe()

##'European', 'African','Asian', 'Mid\_East', 'North\_American', 'Carribean\_Central\_America','OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American','Is\_Nato\_Country'

predictors = data\_clean1[['incomeperperson','armedforcesrate','femaleemployrate','European', 'African','Asian', 'Mid\_East', 'North\_American', 'Carribean\_Central\_America','OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American','Is\_Nato\_Country','Eu\_Member']]

targets = data\_clean1.polityscore\_cat

pred\_train, pred\_test, tar\_train, tar\_test = train\_test\_split(predictors, targets, test\_size=.4)

pred\_train.shape

pred\_test.shape

tar\_train.shape

tar\_test.shape

#Build model on training data

classifier=DecisionTreeClassifier()

classifier=classifier.fit(pred\_train,tar\_train)

##

predictions=classifier.predict(pred\_test)

sklearn.metrics.confusion\_matrix(tar\_test,predictions)

sklearn.metrics.accuracy\_score(tar\_test, predictions)

## array([[21, 11],

## [17, 14]])

##sklearn.metrics.accuracy\_score(tar\_test, predictions)

##Out[386]: 0.55555555555555558

from sklearn import tree

#from StringIO import StringIO

from io import StringIO

#from StringIO import StringIO

from IPython.display import Image

import pydotplus

from sklearn.externals.six import StringIO

import pydot

from sklearn.externals.six import StringIO

with open("gapminder3.dot", 'w') as f:

f = tree.export\_graphviz(classifier, out\_file=f)

os.unlink('gapminder3.dot')

dot\_data = StringIO()

tree.export\_graphviz(classifier, out\_file=dot\_data)

graph = pydot.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_pdf("gapminder.pdf")

predictors.columns.values.tolist()

np.unique(targets)

print sklearn.\_\_version\_\_

from IPython.display import Image

dot\_data = StringIO()

tree.export\_graphviz(classifier, out\_file=dot\_data,feature\_names=predictors.columns.values.tolist())

graph = pydot.graph\_from\_dot\_data(dot\_data.getvalue())

graph.write\_pdf("gapminder4.pdf")

Image(graph.create\_png())