**Introduction**

In our model we are trying to use data combined from transparency international (data to do with how corrupt a country is) and Gapminder to predict polity score (how democratic a country is):

1.      ‘incomeperperson’ :average income per person.

2.      'armedforcesrate’ : armed forces rate as a % of population.

3.      'femaleemployrate’ :female employee rate.

4.      'internetuserate’: internet use rate.

5.      'European’ : Is an European country.

6.      'African’ : Is an African country.

7.      'Asian’ : Is an Asian country.

8.      'Mid\_East’ : Is a Mid-East country.

9.      'North\_American’ : Is a north American country (includes central America and carribean, basically the CONCACAF countries).

10.   'Carribean\_Central\_America’ : Is carribean or central American country.

11.   'OPEC’: Is a member of OPEC.

12.   'Arab\_League’: Is a member of the Arab league.

13.   'ASEAN\_ARF’: Is an ASEAN regional forum.

14.   'South\_American’: Is a South American country.

15.   'Is\_Nato\_Country’: Is a member of NATO.

16.   'Eu\_Member’: Is a member of the EU.

17. 'CPI2015' from the transparency international dataset, the transparency international score for 2015.

18. 'PRS International Country Risk Guide' score. Built by ICRG.

19. 'World Economic Forum EOS (Executive Opinion Survey)' .

20. Number of years a member of NATO (if a country is not in NATO, this is 0)..

The value you are trying to predict is polity score (not political category a derived categorical value based on polity score). All Code is listed in the code section.

**Data pre-processing**

Besides joining the two datasets (Gapminder and Transparency international) together merging on country, the only other operations were to:

1. Give the transparency international countries consistent country names between the two datasets.
2. Standardize the predictor variables so as to make a determination of which predictor had a greater effect on the model.
3. The standardization is done using the scale unction having a mean of 0 and a standard deviation of 1.

**Building our model**

Lasso regression is a type of contraction and selection method for linear regression, it uses L1 regularization, that is it increases a penalty equivalent to the absolute value of the magnitude of coefficients. It minimizes the usual sum of squared errors, with a bound on the sum of the absolute values of the coefficients.

LAR algorithm adds predictor most correlated with response variable and moves towards Least Squared Estimation until there is equally correlated with residual and adds it to model, LAR continues with this is repeated for all variables. The LARS algorithm is much akin to the forward stepwise regression, but instead of the addition variables at each iteration, the estimated parameters are increased in a direction equiangular to each one's correlations with the residual.

The advantages of the LARS algorithm are:

1. It is comparable in terms of computation resources to forward selection.
2. It creates a full piecewise linear solution path, which is extremely useful in conjunction with cross-validation in attempts to optimize the model.
3. It is easily modified to produce solutions for other estimators, like the lasso.

The disadvantage of using this method are:

1. It is sensitive to noise due to iterative refitting of residuals.

**Calculating the optimal alpha (lambda value)**

The optimal alpha value for the the model is assessed using cross validation in conjunction with LARS (LassoLarsCV), this is a more optimal solution than using LassoCV as it explores more useful values of alpha when compared to LassoCV.

I used 5 random folds 4 (the 4 remaining folds, the first being used as a validation set) folds are used for training and 1 (the first fold) used to test. The model which produces the lowest mean squared error as the best model to validate using the test dataset.

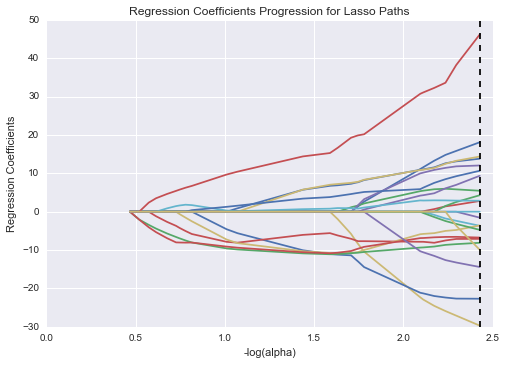
The model is fit using the following code:

*model=sklearn.linear\_model.LassoLarsCV(cv=5, precompute=False).fit(pred\_train,tar\_train)*

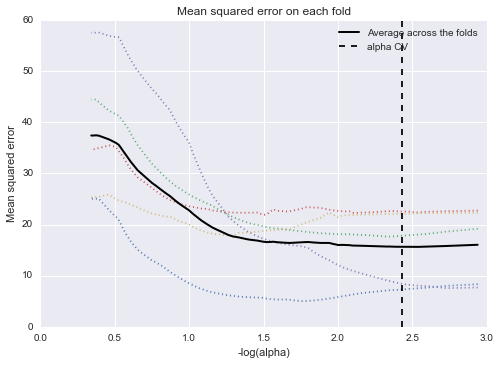
**Diagnostic plots of our model**

Two diagnostic plots were used to look at the lasso selection path for our model these were:

1. A plot of regerssion coefficients progression for Lasso Path.
2. A plot of change Mean Square error at each change in penalty parameter.



This plot shows the relative importance of predictors at each step of the selection process under lasso, how the coefficients changes on addition of another predictor as well as what stage a predictor entered the selection process model. CPI 2015 (the tranparency score) had the largest positive regression value, we can see entering the selection process first as it is the most important.



From the plot we can see that there is variability across each cross validation fold but the change in MSE across all folds follows the same pattern, it decreases rapidly and then levels off.

Note the penalty parameter is referred to as alphas in scikit. Model.alphas. dashed line. This is shown as a vertical line.

**Model Coefficients**

The model coefficients are shown below. We can see the most significant coefficients are CPI2015 (Positive) and income per person.

dict(zip(predictors.columns, model.coef\_)) ##dictionaries and lists

### Out[49]:

{'ASEAN\_ARF': 1.0751280260662057,

'African': -0.80103427955213391,

'Arab\_League': -0.75445326325981865,

'Asian': -0.18027408104575077,

'CPI2015': 5.1488730346502063,

'Carribean\_Central\_America': 1.7511806395204483,

'Eu\_Member': 0.3186323959143092,

'European': 0.48294668353681142,

'Is\_Nato\_Country': 1.1682007573664959,

'Mid\_East': -0.39695605502390469,

'North\_American': 0.0,

'OPEC': -0.86977861918196875,

'PRS International Country Risk Guide': -1.0914128237785916,

'South\_American': 1.5441052599359877,

'World Economic Forum EOS': -1.6287945034420015,

'Years\_In\_Nato': -0.4787168743731352,

'alcconsumption': 0.61054750969745375,

'armedforcesrate': 0.32722934011118593,

'employrate': -2.4374967009233841,

'femaleemployrate': 1.2879949088390379,

'incomeperperson': 1.9394683296765849,

'internetuserate': -3.2651731839242308,

'lifeexpectancy': -0.40401776086788732}

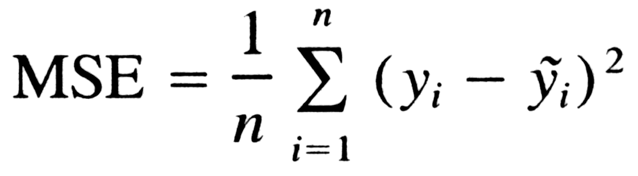
Only one of the coefficients have shrunk to 0 (after applying lasso regression penalty), North America. As we have standardized all predictors on the same scale, so we can tell which are the most important (which are the strongest predictors of polity score).

**Training and test error**

There was significant difference in MSE (Mean Squared Error) error between training and test data indicating I may have over-fitted the model.

The R2 calculated showed significant difference between test and training set indicating our model may suffer from overfitting. The R2 value is how much of the variance in the data we can explain.

**MSE of training and test data**



MSE is mean squared error (MSE) measures the mean of the squares of the errors, the difference between the estimated value and the actual value, it is given by the equation above.

**training data MSE**

7.86094638508

**test data MSE**

21.5213663612

The MSE values were not stable across the training and test set. This would indicate that our model is still over-fitted on our training set. The MSE is higher in the test data.

**training data R-square**

0.776969617669

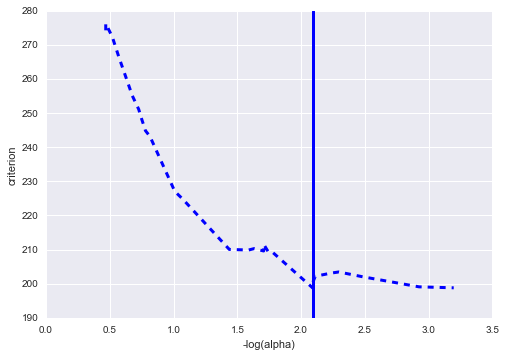
**test data R-square**

0.314393635405

Again the difference in R2 between our training and test data we can explain

**Using AIC criterion**

An alternative to using LassoLarsCV was to use an information criterion selection method, in this case I used (AIC) Akaike information criterion (measure of the relativistic quality criteria of a model) to select an ideal value of alpha regularization parameter.Below we see a diagnostic plot of AIC criterion against log alpha model using the AIC as the information criterion.



**Regression coefficients for AIC selection criteria Lasso Model**

'ASEAN\_ARF': 0.4817277075830127,

'African': -0.88743192067681231,

'Arab\_League': -0.77102456171737688,

'Asian': 0.0,

'CPI2015': 3.4261676420330516,

'Carribean\_Central\_America': 1.3852468150993107,

'Eu\_Member': 0.32737115619068258,

'European': 0.0,

'Is\_Nato\_Country': 0.64445163494424318,

'Mid\_East': -0.64918792797127278,

'North\_American': 0.0,

'OPEC': -1.0037125526070625,

'PRS International Country Risk Guide': 0.0,

'South\_American': 1.1666702294227076,

'World Economic Forum EOS': -1.1639115442413683,

'Years\_In\_Nato': 0.0,

'alcconsumption': 0.59855758131369263,

'armedforcesrate': 0.0,

'employrate': -2.2695726938628469,

'femaleemployrate': 1.0671515028671372,

'incomeperperson': 1.191656220279911,

'internetuserate': -2.4535120774767076,

'lifeexpectancy': 0.0

Note when we use AIC criterion a lot more of the coefficients are regualirzed to 0.

**Training and test error for AIC selection Lasso Model**

Again there was significant difference in MSE (Mean Squared Error) error between training and test data indicating I may have over-fitted the model.

The R2 calculated showed significant difference between test and training set indicating our model may suffer from overfitting. The R2 value is how much of the variance in the data we can explain. The data is shown below.

**training data MSE**

8.70899397216

**test data MSE**

18.4602168991

**training data R-square**

0.752908853441

**test data R-square**

0.411912701759

Again we see the same problems with our model that we may have over-fitted our model to our training set as our model dos not fit well to our test with substantially higher MSE error and lower R2 value.

**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

# Feature Importance

from sklearn.ensemble import ExtraTreesClassifier

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)

#Build model on training data

from sklearn import tree

#from StringIO import StringIO

from io import StringIO

#from StringIO import StringIO

from IPython.display import Image

import pydot ##maygey an error here

from sklearn.externals.six import StringIO

import pydot

##

###

###

##

##lets try

##

##Ok lets see what the best features are

##note to one self HIV rates missing froma lot of countries

datatransparency = pandas.read\_csv('CPI\_2015\_DATA.csv', low\_memory=False)

##w['female'] = w['female'].map({'female': 1, 'male': 0})

datatransparency.columns.values

data.columns.values

## Dont use map

## datatransparency['Country']= datatransparency['Country'].map(

## {"The United States Of America":"United States",

## "C“te dïIvoire":"Cote d'Ivoire",

## "Korea (South)":"Korea, Rep.",

## "Korea (North)":"Korea, Dem. Rep.",

## "Czech Republic":"Czech Rep.",

## "Democratic Republic of the Congo":"Congo, Dem. Rep.",

## "The FYR of Macedonia": "Macedonia, FYR",

## "Hong Kong":"Hong Kong, China"

## })

def country\_consistent (row):

if row['Country'] == "The United States Of America" :

return "United Sates"

elif row['Country'] == "C“te dïIvoire" :

return "Cote d'Ivoire"

elif row['Country'] == "Korea (South)" :

return "Korea, Rep."

elif row['Country'] == "Korea (North)" :

return "Korea, Dem. Rep."

elif row['Country'] == "Korea (South)" :

return "Korea, Rep."

elif row['Country'] == "Czech Republic" :

return "Czech Rep."

elif row['Country'] == "Democratic Republic of the Congo" :

return "Congo, Dem. Rep."

elif row['Country'] == "The FYR of Macedonia" :

return "Macedonia, FYR"

elif row['Country'] == "Hong Kong" :

return "Hong Kong, China"

else :

return row['Country']

datatransparency['Country'] = datatransparency.apply (lambda row: country\_consistent(row),axis=1)

##calculate the age of NATO countries

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

##

##ok after eyeballing in excel they all look ok

##merge the two datasets

datafullset=data.merge(datatransparency,left\_on='country',right\_on='Country',how='left')

datafullset.columns.values

datafullset.count

## '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', 'Rank', 'CPI2015',

## 'Country', 'Region', 'wbcode', 'World Bank CPIA',

## 'World Economic Forum EOS', 'Bertelsmann Foundation TI',

## 'African Dev Bank', 'IMD World Competitiveness Yearbook',

## 'Bertelsmann Foundation SGI', 'World Justice Project ROL',

## 'PRS International Country Risk Guide',

## 'Economist Intelligence Unit', 'IHS Global Insight',

## 'PERC Asia Risk Guide', 'Freedom House NIT', 'CPI2015(2)', 'Rank2',

## 'Number of Sources', 'Std Deviation of Sources', 'Standard Error',

## 'Minimum', 'Maximum', 'Lower CI', 'Upper CI', 'Country2'

data2=datafullset[['incomeperperson', 'alcconsumption', 'armedforcesrate',

'femaleemployrate',

'internetuserate', 'lifeexpectancy',

'employrate', 'European', 'African',

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

'OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American', 'Eu\_Member','Is\_Nato\_Country','Years\_In\_Nato',

'CPI2015','World Economic Forum EOS','PRS International Country Risk Guide',

'polityscore']]

data2.count

data\_clean2 = data2.dropna() ## drop all na values cant handle nulls

data\_clean2.count

from sklearn import preprocessing

## standardize the dataset

data\_clean2['incomeperperson']=preprocessing.scale(data\_clean2['incomeperperson'].astype('float64'))

data\_clean2['alcconsumption']=preprocessing.scale(data\_clean2['alcconsumption'].astype('float64'))

data\_clean2['armedforcesrate']=preprocessing.scale(data\_clean2['armedforcesrate'].astype('float64'))

data\_clean2['femaleemployrate']=preprocessing.scale(data\_clean2['femaleemployrate'].astype('float64'))

data\_clean2['internetuserate']=preprocessing.scale(data\_clean2['internetuserate'].astype('float64'))

data\_clean2['lifeexpectancy']=preprocessing.scale(data\_clean2['lifeexpectancy'].astype('float64'))

data\_clean2['employrate']=preprocessing.scale(data\_clean2['employrate'].astype('float64'))

data\_clean2['European']=preprocessing.scale(data\_clean2['European'].astype('float64'))

data\_clean2['African']=preprocessing.scale(data\_clean2['African'].astype('float64'))

data\_clean2['Asian']=preprocessing.scale(data\_clean2['Asian'].astype('float64'))

data\_clean2['Mid\_East']=preprocessing.scale(data\_clean2['Mid\_East'].astype('float64'))

data\_clean2['North\_American']=preprocessing.scale(data\_clean2['North\_American'].astype('float64'))

data\_clean2['Carribean\_Central\_America']=preprocessing.scale(data\_clean2['Carribean\_Central\_America'].astype('float64'))

data\_clean2['OPEC']=preprocessing.scale(data\_clean2['OPEC'].astype('float64'))

data\_clean2['Arab\_League']=preprocessing.scale(data\_clean2['Arab\_League'].astype('float64'))

data\_clean2['ASEAN\_ARF']=preprocessing.scale(data\_clean2['ASEAN\_ARF'].astype('float64'))

data\_clean2['South\_American']=preprocessing.scale(data\_clean2['South\_American'].astype('float64'))

data\_clean2['Eu\_Member']=preprocessing.scale(data\_clean2['Eu\_Member'].astype('float64'))

data\_clean2['Is\_Nato\_Country']=preprocessing.scale(data\_clean2['Is\_Nato\_Country'].astype('float64'))

data\_clean2['Years\_In\_Nato']=preprocessing.scale(data\_clean2['Years\_In\_Nato'].astype('float64'))

data\_clean2['CPI2015']=preprocessing.scale(data\_clean2['CPI2015'].astype('float64'))

data\_clean2['World Economic Forum EOS']=preprocessing.scale(data\_clean2['World Economic Forum EOS'].astype('float64'))

data\_clean2['PRS International Country Risk Guide']=preprocessing.scale(data\_clean2['PRS International Country Risk Guide'].astype('float64'))

##

###

###check the standardization worked

target = data\_clean2.polityscore

# standardize predictors to have mean=0 and sd=1

predictors=data\_clean2[['incomeperperson', 'alcconsumption', 'armedforcesrate',

'femaleemployrate',

'internetuserate', 'lifeexpectancy',

'employrate', 'European', 'African',

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

'OPEC', 'Arab\_League', 'ASEAN\_ARF', 'South\_American', 'Eu\_Member','Is\_Nato\_Country','Years\_In\_Nato',

'CPI2015','World Economic Forum EOS','PRS International Country Risk Guide']]

###

###

pred\_train, pred\_test, tar\_train, tar\_test = train\_test\_split(predictors, target,

test\_size=.3, random\_state=123)

import sklearn.linear\_model

# specify the lasso regression model

# change cv to 5

model=sklearn.linear\_model.LassoLarsCV(cv=5, precompute=False).fit(pred\_train,tar\_train)

# print variable names and regression coefficients

dict(zip(predictors.columns, model.coef\_))

## {'ASEAN\_ARF': 1.0751280260662057,

## 'African': -0.80103427955213391,

## 'Arab\_League': -0.75445326325981865,

## 'Asian': -0.18027408104575077,

## 'CPI2015': 5.1488730346502063,

## 'Carribean\_Central\_America': 1.7511806395204483,

## 'Eu\_Member': 0.3186323959143092,

## 'European': 0.48294668353681142,

## 'Is\_Nato\_Country': 1.1682007573664959,

## 'Mid\_East': -0.39695605502390469,

## 'North\_American': 0.0,

## 'OPEC': -0.86977861918196875,

## 'PRS International Country Risk Guide': -1.0914128237785916,

## 'South\_American': 1.5441052599359877,

## 'World Economic Forum EOS': -1.6287945034420015,

## 'Years\_In\_Nato': -0.4787168743731352,

## 'alcconsumption': 0.61054750969745375,

## 'armedforcesrate': 0.32722934011118593,

## 'employrate': -2.4374967009233841,

## 'femaleemployrate': 1.2879949088390379,

## 'incomeperperson': 1.9394683296765849,

## 'internetuserate': -3.2651731839242308,

## 'lifeexpectancy': -0.40401776086788732}

# plot coefficient progression

m\_log\_alphas = -np.log10(model.alphas\_)

ax = plt.gca()

plt.plot(m\_log\_alphas, model.coef\_path\_.T)

plt.axvline(-np.log10(model.alpha\_), linestyle='--', color='k',

label='alpha CV')

plt.ylabel('Regression Coefficients')

plt.xlabel('-log(alpha)')

plt.title('Regression Coefficients Progression for Lasso Paths')

# plot mean square error for each fold

m\_log\_alphascv = -np.log10(model.cv\_alphas\_)

plt.figure()

plt.plot(m\_log\_alphascv, model.cv\_mse\_path\_, ':')

plt.plot(m\_log\_alphascv, model.cv\_mse\_path\_.mean(axis=-1), 'k',

label='Average across the folds', linewidth=2)

plt.axvline(-np.log10(model.alpha\_), linestyle='--', color='k',

label='alpha CV')

plt.legend()

plt.xlabel('-log(alpha)')

plt.ylabel('Mean squared error')

plt.title('Mean squared error on each fold')

# MSE from training and test data

from sklearn.metrics import mean\_squared\_error

train\_error = mean\_squared\_error(tar\_train, model.predict(pred\_train))

test\_error = mean\_squared\_error(tar\_test, model.predict(pred\_test))

print ('training data MSE')

print(train\_error)

print ('test data MSE')

print(test\_error)

# R-square from training and test data

rsquared\_train=model.score(pred\_train,tar\_train)

rsquared\_test=model.score(pred\_test,tar\_test)

print ('training data R-square')

print(rsquared\_train)

print ('test data R-square')

print(rsquared\_test)

##

from sklearn.linear\_model import LassoCV, LassoLarsCV, LassoLarsIC

##next we try and fit using AIC criterion

model\_aic = LassoLarsIC(criterion='aic')

model\_aic.fit(pred\_train,tar\_train)

alpha\_aic\_ = model\_aic.alpha\_

def plot\_ic\_criterion(model, name, color):

alpha\_ = model.alpha\_

alphas\_ = model.alphas\_

criterion\_ = model.criterion\_

plt.plot(-np.log10(alphas\_), criterion\_, '--', color=color,

linewidth=3, label='%s criterion' % name)

plt.axvline(-np.log10(alpha\_), color=color, linewidth=3,

label='alpha: %s estimate' % name)

plt.xlabel('-log(alpha)')

plt.ylabel('criterion')

plt.figure()

plot\_ic\_criterion(model\_aic, 'AIC', 'b')

dict(zip(predictors.columns, model\_aic.coef\_))

## {'ASEAN\_ARF': 0.4817277075830127,

## 'African': -0.88743192067681231,

## 'Arab\_League': -0.77102456171737688,

## 'Asian': 0.0,

## 'CPI2015': 3.4261676420330516,

## 'Carribean\_Central\_America': 1.3852468150993107,

## 'Eu\_Member': 0.32737115619068258,

## 'European': 0.0,

## 'Is\_Nato\_Country': 0.64445163494424318,

## 'Mid\_East': -0.64918792797127278,

## 'North\_American': 0.0,

## 'OPEC': -1.0037125526070625,

## 'PRS International Country Risk Guide': 0.0,

## 'South\_American': 1.1666702294227076,

## 'World Economic Forum EOS': -1.1639115442413683,

## 'Years\_In\_Nato': 0.0,

## 'alcconsumption': 0.59855758131369263,

## 'armedforcesrate': 0.0,

## 'employrate': -2.2695726938628469,

## 'femaleemployrate': 1.0671515028671372,

## 'incomeperperson': 1.191656220279911,

## 'internetuserate': -2.4535120774767076,

## 'lifeexpectancy': 0.0}

from sklearn.metrics import mean\_squared\_error

train\_error\_aic = mean\_squared\_error(tar\_train, model\_aic.predict(pred\_train))

test\_error\_aic = mean\_squared\_error(tar\_test, model\_aic.predict(pred\_test))

print ('training data MSE')

print(train\_error\_aic)

print ('test data MSE')

print(test\_error\_aic)

# R-square from training and test data

rsquared\_train\_aic=model\_aic.score(pred\_train,tar\_train)

rsquared\_test\_aic=model\_aic.score(pred\_test,tar\_test)

print ('training data R-square')

print(rsquared\_train\_aic)

print ('test data R-square')

print(rsquared\_test\_aic)