Data Source: http://data.un.org/Explorer.aspx?d=EDATA

Description:

The aim of our projecto is to create a model able to predict the future consumption and production of energy of a given country given the "characteristics" of this country, that is: population, population grow, GDP, GNI, Exports, imports, and other more exotic characteristics like chicken stocks, cattle stocks, Fertility rate, enrolment to primary and secondary school among others.

In [1]:

```
#Importing dependencies
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [230]:

```
#Variables to be imported: we split our variables into those related with Consumption, Production
and Characteristics

CONSUMPTION = [
'Consumption by chemical and petrochemical.csv', \
'Consumption by commercial and public services.csv', \
'Consumption by construction.csv', \
'Consumption by households.csv', \
'Consumption by manufacturing construction and non-fuel industry.csv', \
'Consumption by mining and quarrying.csv', \
'BiodieselCon.csv', \
'BiogasCons.csv', \
'BiogasolineCons.csv', \
'FuelCons.csv', \
'by transport.csv', \
'con_in_Agric_for_fishing.csv', \
'Final energy consumption.csv', \
'textile_leather.csv']

PRODUCTION = [
'Electricity - total solar production.csv', \
'Electricity - total wind production.csv', \
'Electricity Gross production.csv', \
'Biodieselprod.csv', \
'BiogasolineProd.csv', \
'BiogasProd.csv', \
'FuelPro.csv', \
'GeoThermalPro.csv', \
'Hydropro.csv', \
'NaturalPro.csv', \
'NuclearPro.csv']

CHARACTERISTICS = [
'exports.csv', \
'imports.csv', \
'Consumer food prices.csv',\
'Poverty as ratio of population.csv',\
'losses.csv', \
'Cash Surplus.csv', \
'Exports of goods.csv', \
'GDP.csv', \
'GNI.csv', \
'Inflation.csv', \
'Internet Usage.csv', \
'Imports of goods.csv',\
'Precipitation.csv',\
'Sugar cane Area harvested (ha).csv', \
'Agricultural area (1000 ha).csv', \
'Country area (1000 ha).csv', \
'Forest Area (1000 ha).csv', \
'Cattle Stocks.csv', \
'Chickens Stocks.csv', \
'CO2 emissions (metric tons per capita).csv', \
'Fertility rate, total (births per woman).csv', \
'Gross enrolment ratio, primary and secondary, gender parity index (GPI).csv', \
'High-technology exports (% of manufactured exports).csv', \
'Improved water source (% of population with access).csv', \
'Life expectancy at birth, total (years).csv', \
'Mobile cellular subscriptions (per 100 people).csv', \
'Population growth (annual %).csv', \
'Population (total).csv', \
'Total debt service (% of exports of goods, services and primary income).csv', \
'Foreign direct investment.csv', \
'Income share held by lowest 20%.csv',\
'Government expenditure on education as ratio of GDP.csv']

vars = CONSUMPTION + PRODUCTION + CHARACTERISTICS
```

In [3]:

```
#These variable will help us to rename some countries
dic = [('Brunei Darussalam' , 'Brunei'), \
        ('Korea, Republic of' , 'South Korea'), \
        ('T.F.Yug.Rep. Macedonia', 'Macedonia'), \
        ('State of Palestine' , 'Palestine'), \
        ('Russian Federation' , 'Russia'), \
        ('Republic of Moldova' , 'Moldova'), \
        ('Czechia' , 'Czech Republic'), \
        ('Syrian Arab Republic' , 'Syrian'), \
        ('Ethiopia, incl. Eritrea' , 'Ethiopia'), \
        ('China, Hong Kong SAR' , 'Hong Kong'), \
        ('Bonaire, St Eustatius, Saba' , 'Bonaire'), \
        ('Iran, Islamic Rep.', 'Iran'), \
        ('Korea, Dem. Rep.', 'North Korea'), \
        ('Micronesia (Fed. States of)', 'Micronesia'),\
        ("Lao People's Dem. Rep.", 'Lao'), \
        ('Northern Mariana Islands, Saba' , 'Mariana Islands'), \
        ('United Rep. of Tanzania' , 'Tanzania')]
```

In [4]:

```
#The following "countries" will not be taken into consideration in our analysis; either because th
ey are not "real" or contain
#little information

remove_list = ['1', \
                '2', \
                'Arab World', \
                'Dominican Rep.', \
                'China, Hong Kong Special Administrative Region',\
                'East Asia & Pacific (all income levels)', \
                'East Asia & Pacific (developing only)', \
                'Europe & Central Asia (all income levels)', \
                'Europe & Central Asia (developing only)', \
                'European Union', \
                'Euro area',\
                'fnSeqID',\
                'Former Czechoslovakia', \
                'Former Ethiopia', \
                'Former Netherlands Antilles', \
                'Former Sudan', \
                'Former USSR', \
                'Former Yugoslavia', \
                'German Dem. R. (former)', \
                'Germany, Fed. R. (former)', \
                'Heavily indebted poor countries (HIPC)', \
                'High income', \
                'Holy See', \
                'l',\
                'Latin America & Caribbean (all income levels)', \
                'Latin America & Caribbean (developing only)', \
                'Least developed countries: UN classification', \
                'Libyan Arab Jamahiriya', \
                'Low & middle income', \
                'Low income', \
                'Lower middle income', \
                'Middle East & North Africa (all income levels)', \
                'Middle East & North Africa (developing only)', \
                'Middle income', \
                'Neth. Antilles (former)', \
                'North America',\
                'OECD members', \
                'Occupied Palestinian Territory', \
                'Other Asia',\
                'Pacific Islands (former)', \
                'Réunion', \
                'Saint Helena', \
                'Saint Kitts and Nevis', \
                'Saint Lucia', \
                'Saint Pierre and Miquelon', \
                'Saint Vincent and the Grenadines', \
                'Sint Maarten (Dutch part)', \
                'South Asia', \
                'St. Helena and Depend.', \
                'St. Kitts and Nevis', \
                'St. Kitts-Nevis', \
                'St. Lucia', \
                'St. Pierre-Miquelon', \
                'St. Vincent and the Grenadines', \
                'St. Vincent-Grenadines', \
                'Sudan (former)',\
                'The former Yugoslav Republic of Macedonia', \
                'Sub-Saharan Africa (all income levels)', \
                'Sub-Saharan Africa (developing only)', \
                'Upper middle income', \
                'Yemen, Dem. (former)',\
                'Yemen: Former Democratic Yemen',\
                'World', \
                'footnoteSeqID', \
                'West Bank and Gaza',\
                'Yugoslavia, SFR (former)',\
                'Kosovo',\
                'Falkland Is. (Malvinas)',\
                'Wallis and Futuna Is.',\
                'American Samoa',\
                'United States Virgin Islands','Channel Islands']
```

In [5]:

```
#The following dictinary is to add up the information of repeated countries
dict_2={'Bahamas, The': 'Bahamas',
        'Bolivia (Plur. State of)': 'Bolivia',
        'Bolivia (Plurinational State of)': 'Bolivia',
        'Central African Republic': 'Central African Rep.',
        'China, Macao SAR': 'China',
        'China, Macao Special Administrative Region': 'China',
        'Macao SAR, China': 'China',
        'Macao Special Administrative Region of China': 'China',
        'Congo, Dem. Rep.': 'Congo',
        'Congo, Rep.': 'Congo',
        'Dem. Rep. of the Congo': 'Congo',
        'Democratic Republic of the Congo': 'Congo',
        'Egypt, Arab Rep.': 'Egypt' ,
        'Gambia, The': 'Gambia' ,
        'Germany, Fed. R.': 'Germany',
        'Hong Kong SAR, China': 'Hong Kong',
        'Hong Kong Special Administrative Region of China' : 'Hong Kong',
        'Iran (Islamic Rep. of)': 'Iran',
        'Iran, Islamic Republic of': 'Iran',
        'Republic of Korea': 'South Korea',
        'Korea, Rep.': 'South Korea',
        'Kyrgyz Republic': 'Kyrgyzstan',
        'Lao PDR': 'Lao',
        'Macedonia, FYR' : 'Macedonia',
        'Macedonia, The former Yugoslav Rep. of' : 'Macedonia',
        'Micronesia (Federated States of)': 'Micronesia',
        'Micronesia, Fed. Sts.': 'Micronesia',
        'Slovak Republic': 'Slovakia',
        'Venezuela, RB' : 'Venezuela',
         'China, Hong Kong Special Administrative Region': 'China',
         'Congo (Democratic Republic of the)': 'Congo',
         'Korea (Rep. of)': 'South Korea',
         'Lao P.D.R.': 'Lao',
         'Micronesia (Fed. States of)': 'Micronesia',
         'Syrian':  'Syria',
         'T.F.Y.R. Macedonia': 'Macedonia',
         'USSR (former)': 'Russia',
         'United Kingdom of Great Britain and Northern Ireland': 'United Kingdom',
         'United Republic of Tanzania': 'Tanzania',
         'United Republic of Tanzania: Mainland': 'Tanzania',
         'United Republic of Tanzania: Zanzibar': 'Tanzania',
         'United States of America': 'United States',
         'United States Virgin Is.': 'United States Virgin Islands',
         'Venezuela (Bolivar. Rep.)':  'Venezuela',
         'Venezuela (Bolivarian Republic of)':  'Venezuela',
         'Vietnam': 'Viet Nam',
         'Yemen Arab Rep. (former)':  'Yemen',
         'Yemen, Rep.':  'Yemen',
        'Yemen: Former Yemen Arab Republic':  'Yemen',
        "China, People's Republic of" : 'China',
        'Macao, China' : 'China',
        'Hong Kong, China' : 'Hong Kong',
        "Dem. People's Rep. of Korea" : 'North Korea',
        "Democratic People's Republic of Korea" : 'North Korea',
        "Korea, Dem.Ppl's.Rep." : 'North Korea',
        "Lao People's Democratic Republic" : 'Lao',
        'Iranblic of' : 'Iran',
        'Northern Mariana Islands' : 'Northern Marianas Islands',
        'Northern Marianas Islands' : 'North America',
        'Faeroe Islands' : 'Denmark',
        'British Virgin Islands' : 'United Kingdom',
        'Channel Islands' : 'United Kingdom',
        'Bonaire' : 'Netherlands',
        'Netherlands Antilles' : 'Netherlands',
        'Virgin Islands (U.S.)' : 'North America',
       'American Samoa': 'North America',
       "Côte d'Ivoire" : "Cote d'Ivoire",
       'Czechoslovakia (former)': 'Czech Republic',
       'United States Virgin Islands' : 'United States'}
```

Data Cleaning Process

In [6]:

```
for j in range(len(vars)): #For all
    print('Processing: ', j ,vars[j])

    temp = pd.read_csv('Data/'+vars[j]) #importing data set
    col = temp.columns.values # columns names


#==========================================================================================
==============
    #Units of variable
    if 'Unit' in col:
        unit = temp.loc[0]['Unit']
    else:
        unit = ''

    #We create a dictionary containing the names of the variables and an abbreviations of these
    #Without units
    if (('Commodity - Transaction' in col) or ('Item' in col)):
        if 'Commodity - Transaction' in col and ('-' in  temp.loc[0]['Commodity - Transaction']):
            var = temp.loc[0]['Commodity - Transaction'].split('-')[1] #name of variable
            #var = var + ' (' + unit + ')'
            if j == 0:
                dict_var = {'var'+'-'+str(j) : var}
            else:
                dict_var.update({'var'+'-'+str(j) : var})
        elif  'Item' in col and ('-' in  temp.loc[0]['Item']):
            var = temp.loc[0]['Item'].split('-')[1] #name of variable
            #var = var + ' (' + unit + ')'
            if j == 0:
                dict_var = {'var'+'-'+str(j) : var}
            else:
                dict_var.update({'var'+'-'+str(j) : var})
    else:
        var = vars[j].split('.')[0];
        dict_var.update({'var'+'-'+str(j) : var})

    #We drop these columns since we already have their information in the dictionary just created
    if 'Quantity' in col:
        temp=temp.drop(list(set(col)-set({'Country or Area','Year','Quantity'})), axis=1)
    elif 'Value' in col:
        temp=temp.drop(list(set(col)-set({'Country or Area','Year','Value'})), axis=1)
    else:
        print('Variable not imported. Different type \n')
        continue


#==================================================================================================
===================

    temp.columns = [col[0].split()[0],'Year','var'+'-'+str(j)] #changing the names of the columns
    #temp = temp.fillna(0) #filling missing values with 0

    #Here we change the names of some countries, we simplify their names
    for i in range(len(dic)):
        temp['Country'] = temp['Country'].str.replace(dic[i][0], dic[i][1])

    #Cheking the countries list
    Countries = list(set(temp['Country']))

    #Dropping this unnecessary countries
    Countries = list(set(Countries) - set(remove_list))

    #Constructing a dictionary to create the final data frame

dummy=temp[temp['Country']==Countries[0]].drop('Country',axis=1).reset_index(drop=True).set_index(
'Year')
    var = {Countries[0] : dummy} #dictionary
    for i in Countries[1:]:
        var[i] = temp[temp['Country']==i].drop('Country',axis=1).reset_index(drop=True).set_index('
Year')


#==================================================================================================
======================
    #In our csv file there are countries that are the same but have different name, these countrie
s are summarize in the dict_4.
    #We need to sum up the information containen in those countries as one:
    for i in list(Countries):
        if i in dict_2:
            if dict_2[i] in set(var.keys()):
                df1=var[dict_2[i]]
                df2=var[i]
                df1=df1.join(df2,lsuffix='_0', rsuffix='_1')
                var[dict_2[i]]=df1.sum(axis=1)
                del var[i]
            else: #If the country in our dictionary is not created yet, we create it
                var[dict_2[i]] = var[i]
                del var[i]

#==================================================================================================
======================
    df1 = pd.concat(var)

    if 0 in df1.columns.values:
            df1.drop(0,axis=1,inplace=True)

    if j == 0:
        df = df1
    else:
        df=df.join(df1, how='left')
```

```
Processing:  0 Consumption by chemical and petrochemical.csv
```

```
C:\Users\User\Anaconda3\lib\site-packages\pandas\core\indexes\api.py:57: RuntimeWarning:
unorderable types: int() < str(), sort order is undefined for incomparable objects
  union = _union_indexes(indexes)
```

```
Processing:  1 Consumption by commercial and public services.csv
```

```
C:\Users\User\Anaconda3\lib\site-packages\pandas\core\indexes\api.py:87: RuntimeWarning:
unorderable types: int() < str(), sort order is undefined for incomparable objects
  result = result.union(other)
```

```
Processing:  2 Consumption by construction.csv
Processing:  3 Consumption by households.csv
Processing:  4 Consumption by manufacturing construction and non-fuel industry.csv
Processing:  5 Consumption by mining and quarrying.csv
Processing:  6 BiodieselCon.csv
Processing:  7 BiogasCons.csv
Processing:  8 BiogasolineCons.csv
Processing:  9 FuelCons.csv
Processing:  10 by transport.csv
Processing:  11 con_in_Agric_for_fishing.csv
Processing:  12 Final energy consumption.csv
Processing:  13 textile_leather.csv
Processing:  14 Electricity - total solar production.csv
Processing:  15 Electricity - total wind production.csv
Processing:  16 Electricity Gross production.csv
Processing:  17 Biodieselprod.csv
Processing:  18 BiogasolineProd.csv
Processing:  19 BiogasProd.csv
Processing:  20 FuelPro.csv
Processing:  21 GeoThermalPro.csv
Processing:  22 Hydropro.csv
Processing:  23 NaturalPro.csv
Processing:  24 NuclearPro.csv
Processing:  25 exports.csv
Processing:  26 imports.csv
Processing:  27 Consumer food prices.csv
Processing:  28 Poverty as ratio of population.csv
Processing:  29 losses.csv
Processing:  30 Cash Surplus.csv
Processing:  31 Exports of goods.csv
Processing:  32 GDP.csv
Processing:  33 GNI.csv
Processing:  34 Inflation.csv
Processing:  35 Internet Usage.csv
Processing:  36 Imports of goods.csv
Processing:  37 Precipitation.csv
Processing:  38 Sugar cane Area harvested (ha).csv
Processing:  39 Agricultural area (1000 ha).csv
Processing:  40 Country area (1000 ha).csv
Processing:  41 Forest Area (1000 ha).csv
Processing:  42 Cattle Stocks.csv
Processing:  43 Chickens Stocks.csv
Processing:  44 CO2 emissions (metric tons per capita).csv
Processing:  45 Fertility rate, total (births per woman).csv
Processing:  46 Gross enrolment ratio, primary and secondary, gender parity index (GPI).csv
Processing:  47 High-technology exports (% of manufactured exports).csv
Processing:  48 Improved water source (% of population with access).csv
Processing:  49 Life expectancy at birth, total (years).csv
Processing:  50 Mobile cellular subscriptions (per 100 people).csv
Processing:  51 Population growth (annual %).csv
Processing:  52 Population (total).csv
Processing:  53 Total debt service (% of exports of goods, services and primary income).csv
Processing:  54 Foreign direct investment.csv
Processing:  55 Income share held by lowest 20%.csv
Processing:  56 Government expenditure on education as ratio of GDP.csv
```

In [7]:

```
df = df[~df.index.duplicated(keep='first')] #drop some repeated level 1 indexes
```

In [8]:

```
#Renaming some variables
dict_var.update({'var-32' : 'GDP','var-33' : 'GNI', 'var-17': ' Biodiesel Production',
 'var-18': ' Biogasoline production',
 'var-19': ' Biogas Production',
 'var-20': ' Fuel Production',
 'var-23': ' Natural Gas Production',
 'var-4': ' Consumption by manufacturing and construction',
 'var-45': 'Fertility rate',
 'var-46': 'Enrolment primary and secondary school',
 'var-53': 'Total debt service',
 'var-6': ' Biodiesel consumption',
 'var-7': ' Biogas consumption',
 'var-8': ' Biogasoline consumption',
 'var-9': ' Fuel consumption',
 'var-47': 'High-technology exports',
 'var-48': 'Improved water source',
 'var-56': 'Government expenditure on education'})
```

In [9]:

```
#These variables will we usefull to know how to split or DataFrame into "production",
"consumption" and "characteristics"

production = ['var-0', 'var-1', 'var-2', 'var-3', 'var-4', 'var-5', 'var-6', 'var-7', 'var-8', 'var
-9', 'var-10',\
              'var-11', 'var-12', 'var-13']

consumption = ['var-14', 'var-15', 'var-16', 'var-17', 'var-18', 'var-19', 'var-20', 'var-21', 'var
-22', 'var-23', 'var-24']

characteristics = ['var-25', 'var-26', 'var-27', 'var-28', 'var-29', 'var-30', 'var-31', 'var-32',
'var-33', 'var-34', \
                   'var-35', 'var-36', 'var-37', 'var-38', 'var-39', 'var-40', 'var-41', 'var-42',
var-43', 'var-44', \
                   'var-45', 'var-46', 'var-47', 'var-48', 'var-49', 'var-50', 'var-51', 'var-52',
var-53', 'var-54', \
                   'var-55', 'var-56']
```

In [18]:

```
dg=df # We will work with this variable as our data frame
```

In [12]:

```
dict_var #Dictionary of variables
```

Out[12]:

```
{'var-0': ' Consumption by chemical and petrochemical',
 'var-1': ' Consumption by commercial and public services',
 'var-10': ' Consumption by transport',
 'var-11': ' Consumption in agriculture, forestry and fishing',
 'var-12': ' Final energy consumption',
 'var-13': ' Consumption by textile and leather',
 'var-14': ' total solar production',
 'var-15': ' total wind production',
 'var-16': ' Gross production',
 'var-17': ' Biodiesel Production',
 'var-18': ' Biogasoline production',
 'var-19': ' Biogas Production',
 'var-2': ' Consumption by construction',
 'var-20': ' Fuel Production',
 'var-21': ' total geothermal production',
 'var-22': ' total hydro production',
 'var-23': ' Natural Gas Production',
 'var-24': ' total nuclear production',
 'var-25': ' exports',
 'var-26': ' imports',
 'var-27': 'Consumer food prices',
 'var-28': 'Poverty as ratio of population',
 'var-29': ' Losses',
 'var-3': ' Consumption by households',
 'var-30': 'Cash Surplus',
 'var-31': 'Exports of goods',
 'var-32': 'GDP',
 'var-33': 'GNI',
 'var-34': 'Inflation',
 'var-35': 'Internet Usage',
 'var-36': 'Imports of goods',
 'var-37': 'Precipitation',
 'var-38': 'Sugar cane Area harvested (ha)',
 'var-39': 'Agricultural area (1000 ha)',
 'var-4': ' Consumption by manufacturing and construction',
 'var-40': 'Country area (1000 ha)',
 'var-41': 'Forest Area (1000 ha)',
 'var-42': 'Cattle Stocks',
 'var-43': 'Chickens Stocks',
 'var-44': 'CO2 emissions (metric tons per capita)',
 'var-45': 'Fertility rate',
 'var-46': 'Enrolment primary and secondary school',
 'var-47': 'High-technology exports',
 'var-48': 'Improved water source',
 'var-49': 'Life expectancy at birth, total (years)',
 'var-5': ' Consumption by mining and quarrying',
 'var-50': 'Mobile cellular subscriptions (per 100 people)',
 'var-51': 'Population growth (annual %)',
 'var-52': 'Population (total)',
 'var-53': 'Total debt service',
 'var-54': 'Foreign direct investment',
 'var-55': 'Income share held by lowest 20%',
 'var-56': 'Government expenditure on education',
 'var-6': ' Biodiesel consumption',
 'var-7': ' Biogas consumption',
 'var-8': ' Biogasoline consumption',
 'var-9': ' Fuel consumption'}
```

Exploratory Analysis

Exploring the information contained in our data set we can see that we have a lot of missing values.

In [21]:

```
dg.isnull().sum(axis=0)
```

Out[21]:

```
var-0       24
var-1      429
var-2      748
var-3      106
var-4      128
var-5      705
var-6     1292
var-7     1115
var-8     1366
var-9      136
var-10     239
var-11     208
var-12     128
var-13     656
var-14    1111
var-15     819
var-16     152
var-17    1317
var-18    1435
var-19     884
var-20     263
var-21    1482
var-22     190
var-23    1002
var-24    1177
var-25     450
var-26     426
var-27     665
var-28    1473
var-29     128
var-30     499
var-31     113
var-32      89
var-33      89
var-34      79
var-35     242
var-36     113
var-37     876
var-38    1499
var-39      83
```

```
var-40      109
var-41       83
var-42       79
var-43       79
var-44      231
var-45      153
var-46      482
var-47      252
var-48       43
var-49      152
var-50       75
var-51       79
var-52       76
var-53     1149
var-54       86
var-55     1117
var-56      737
dtype: int64
```

We will use a variation of our data frame, concretely, we will use the one with the mean value of each variable and all the correspondent years.

In [22]:

```
dg=dg.mean(level=1)
```

In [23]:

```
dg #This will we our main Data Frame
```

Out[23]:

	var-0	var-1	var-2	var-3	var-4	var-5	var-6	var-7	
Year									
1999.0	11302.442857	50495.034783	1056.125000	42003.014925	58854.924242	3544.545455	109.000000	4100.047619	3765
2000.0	11676.927536	55515.079545	1070.939394	42732.909091	59864.030769	3516.264706	106.285714	3392.416667	3543
2001.0	11891.705882	55780.711111	1026.771429	43947.215385	59440.390625	3221.222222	96.777778	3554.692308	2309
2002.0	10657.318841	55824.642391	1831.750000	44624.666667	56586.661538	3206.833333	92.083333	4227.148148	1844
2003.0	10823.942029	59277.704545	1920.457143	45561.303030	58071.246154	3096.750000	117.166667	3569.250000	1702
2004.0	11361.257143	61167.431818	1932.228571	46889.283582	58905.848485	3202.944444	199.342314	3736.464286	1386
2005.0	11586.923693	59599.610648	1826.105263	47331.548382	58413.578358	3655.675000	259.451160	3800.655172	1256
2006.0	11928.825822	58817.153540	2171.050000	47765.717647	59790.710158	3761.166667	281.371430	4057.172414	1177
2007.0	12767.810995	55571.972812	1739.847800	49712.911658	61437.496004	3191.365000	324.114040	4193.272727	1288
2008.0	12117.542639	52552.192549	1585.615556	49460.752729	59564.459011	2980.800545	362.337056	4048.529412	1546
2009.0	11775.429330	52355.157672	1485.842593	49643.327697	54169.972159	2756.605357	423.699453	4513.971429	1641
2010.0	12219.262399	51329.141418	1919.210909	50768.861831	55929.345528	2916.420690	462.022249	1926.000000	1831
2011.0	12525.540654	50226.597455	1838.014255	49493.429958	56599.922804	2965.179102	516.437987	2574.282051	1808
2012.0	12984.715957	50317.696912	1862.704618	49647.220776	56776.027095	3045.286051	581.159053	2838.175000	1829
2013.0	13600.451683	51183.171282	1781.963860	51782.444824	59962.238990	3204.487194	551.749865	3422.243902	1834
2014.0	13583.056075	49817.341227	1819.843966	50709.355842	58691.207622	3143.408619	585.047702	3814.926829	1911
2015.0	13679.195591	49891.164836	1906.828345	51077.861289	58885.625211	3223.746031	595.590327	3915.024390	2005
1990.0	12257.000000	38625.820513	971.160000	42548.045455	60974.302326	3846.961538	8.000000	814.833333	9225
1991.0	13166.041667	41615.300000	904.038462	46032.044444	66596.681818	3789.851852	8.000000	1178.785714	9596
1992.0	12193.803279	37467.914894	1319.161290	39197.087719	62320.403509	3519.100000	4.500000	1013.714286	4681
1993.0	12078.919355	39626.217391	1269.129032	40652.620690	62197.810345	3388.806452	7.250000	940.333333	4413
1994.0	11913.828125	41216.695652	1231.774194	40319.516667	60327.833333	3310.064516	31.750000	1036.066667	4757

Year	var-0	var-1	var-2	var-3	var-4	var-5	var-6	var-7	
1995.0	12062.707692	42043.893617	1211.218750	40948.213115	60519.606557	3419.500000	53.500000	1099.533333	4965
1996.0	12392.738462	44635.021739	1212.000000	42537.693548	60958.311475	3553.187500	78.000000	1131.562500	4719
1997.0	12244.378788	46660.826087	1080.250000	42081.507937	60909.274194	3479.636364	104.750000	1137.312500	4845
1998.0	11825.469697	50943.500000	1068.812500	43567.047619	61094.435484	3554.484848	95.250000	1113.764706	3723
2016.0	916.616597	9748.787315	929.300000	13746.535278	12691.438056	1248.220000	85.170800	3.000000	755.

27 rows × 57 columns

In [24]:

```
dg.isnull().sum(axis=1)
```

Out[24]:

```
Year
1999.0     0
2000.0     0
2001.0     0
2002.0     0
2003.0     0
2004.0     0
2005.0     0
2006.0     0
2007.0     0
2008.0     0
2009.0     1
2010.0     1
2011.0     1
2012.0     1
2013.0     1
2014.0     3
2015.0    10
1990.0     0
1991.0     1
1992.0     0
1993.0     0
1994.0     0
1995.0     1
1996.0     0
1997.0     0
1998.0     0
2016.0    26
dtype: int64
```

In [16]:

```
dg.isnull().sum(axis=0)
```

Out[16]:

```
var-0     0
var-1     0
var-2     0
var-3     0
var-4     0
var-5     0
var-6     0
var-7     0
var-8     0
var-9     0
var-10    0
var-11    0
var-12    0
var-13    0
var-14    0
var-15    0
var-16    0
var-17    0
var-18    0
var-19    0
var-20    0
```

```
var-21    0
var-22    0
var-23    0
var-24    1
var-25    0
var-26    0
var-27    8
var-28    3
var-29    0
var-30    2
var-31    1
var-32    0
var-33    0
var-34    1
var-35    2
var-36    1
var-37    1
var-38    3
var-39    1
var-40    1
var-41    1
var-42    0
var-43    0
var-44    3
var-45    2
var-46    1
var-47    1
var-48    1
var-49    2
var-50    1
var-51    1
var-52    1
var-53    2
var-54    1
var-55    2
var-56    2
dtype: int64
```

We still can see that our Data Frame have some sissing values, we will fill those values with the mean with respect of all the available years.

In [25]:

```
dg=dg.fillna(dg.mean()) #This will be our main Data Frame
dg
```

Out[25]:

	var-0	var-1	var-2	var-3	var-4	var-5	var-6	var-7	
Year									
1999.0	11302.442857	50495.034783	1056.125000	42003.014925	58854.924242	3544.545455	109.000000	4100.047619	3765
2000.0	11676.927536	55515.079545	1070.939394	42732.909091	59864.030769	3516.264706	106.285714	3392.416667	3543
2001.0	11891.705882	55780.711111	1026.771429	43947.215385	59440.390625	3221.222222	96.777778	3554.692308	2309
2002.0	10657.318841	55824.642391	1831.750000	44624.666667	56586.661538	3206.833333	92.083333	4227.148148	1844
2003.0	10823.942029	59277.704545	1920.457143	45561.303030	58071.246154	3096.750000	117.166667	3569.250000	1702
2004.0	11361.257143	61167.431818	1932.228571	46889.283582	58905.848485	3202.944444	199.342314	3736.464286	1386
2005.0	11586.923693	59599.610648	1826.105263	47331.548382	58413.578358	3655.675000	259.451160	3800.655172	1256
2006.0	11928.825822	58817.153540	2171.050000	47765.717647	59790.710158	3761.166667	281.371430	4057.172414	1177
2007.0	12767.810995	55571.972812	1739.847800	49712.911658	61437.496004	3191.365000	324.114040	4193.272727	1288
2008.0	12117.542639	52552.192549	1585.615556	49460.752729	59564.459011	2980.800545	362.337056	4048.529412	1546
2009.0	11775.429330	52355.157672	1485.842593	49643.327697	54169.972159	2756.605357	423.699453	4513.971429	1641
2010.0	12219.262399	51329.141418	1919.210909	50768.861831	55929.345528	2916.420690	462.022249	1926.000000	1831
2011.0	12525.540654	50226.597455	1838.014255	49493.429958	56599.922804	2965.179102	516.437987	2574.282051	1808

Year	var-0	var-1	var-2	var-3	var-4	var-5	var-6	var-7	
2012.0	12984.715957	50317.696912	1862.704618	49647.220776	56776.027095	3045.286051	581.159053	2838.175000	1829
2013.0	13600.451683	51183.171282	1781.963860	51782.444824	59962.238990	3204.487194	551.749865	3422.243902	1834
2014.0	13583.056075	49817.341227	1819.843966	50709.355842	58691.207622	3143.408619	585.047702	3814.926829	1911
2015.0	13679.195591	49891.164836	1906.828345	51077.861289	58885.625211	3223.746031	595.590327	3915.024390	2005
1990.0	12257.000000	38625.820513	971.160000	42548.045455	60974.302326	3846.961538	8.000000	814.833333	9225
1991.0	13166.041667	41615.300000	904.038462	46032.044444	66596.681818	3789.851852	8.000000	1178.785714	9596
1992.0	12193.803279	37467.914894	1319.161290	39197.087719	62320.403509	3519.100000	4.500000	1013.714286	4681
1993.0	12078.919355	39626.217391	1269.129032	40652.620690	62197.810345	3388.806452	7.250000	940.333333	4413
1994.0	11913.828125	41216.695652	1231.774194	40319.516667	60327.833333	3310.064516	31.750000	1036.066667	4757
1995.0	12062.707692	42043.893617	1211.218750	40948.213115	60519.606557	3419.500000	53.500000	1099.533333	4965
1996.0	12392.738462	44635.021739	1212.000000	42537.693548	60958.311475	3553.187500	78.000000	1131.562500	4719
1997.0	12244.378788	46660.826087	1080.250000	42081.507937	60909.274194	3479.636364	104.750000	1137.312500	4845
1998.0	11825.469697	50943.500000	1068.812500	43567.047619	61094.435484	3554.484848	95.250000	1113.764706	3723
2016.0	916.616597	9748.787315	929.300000	13746.535278	12691.438056	1248.220000	85.170800	3.000000	755.

27 rows × 57 columns

In [26]:

```
round(dg.describe(),2)
```

Out[26]:

	var-0	var-1	var-2	var-3	var-4	var-5	var-6	var-7	var-8	var-9	...	var-47	var-48	var-49	v
count	27.00	27.00	27.00	27.00	27.00	27.00	27.00	27.00	27.00	27.00	...	27.00	27.00	27.00	2
mean	11760.51	48603.92	1480.45	44621.56	57797.55	3249.72	227.40	2635.30	3124.72	1489.68	...	13.04	94.13	73.36	5
std	2297.37	10262.53	397.39	7269.90	9330.73	488.17	207.31	1432.00	2265.54	673.02	...	1.32	1.61	1.55	4
min	916.62	9748.79	904.04	13746.54	12691.44	1248.22	4.50	3.00	755.78	344.38	...	10.74	91.93	71.31	0
25%	11726.18	43339.46	1075.59	42309.60	58242.41	3120.08	81.59	1122.66	1672.11	950.57	...	12.17	92.63	71.97	7
50%	12078.92	50495.03	1485.84	45561.30	59564.46	3223.75	109.00	3392.42	1911.99	1377.17	...	13.04	94.06	73.36	5
75%	12459.14	55543.53	1834.88	49568.38	60933.79	3531.82	393.02	3864.98	4547.25	1940.35	...	14.00	95.84	74.59	1
max	13679.20	61167.43	2171.05	51782.44	66596.68	3846.96	595.59	4513.97	9596.00	2857.63	...	15.56	96.37	75.91	1

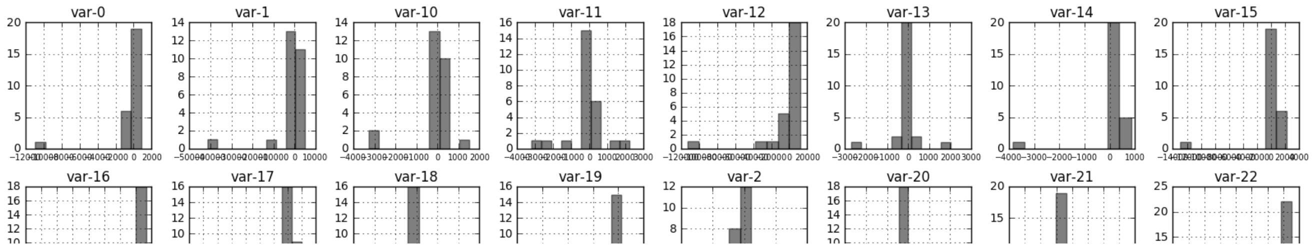
8 rows × 57 columns

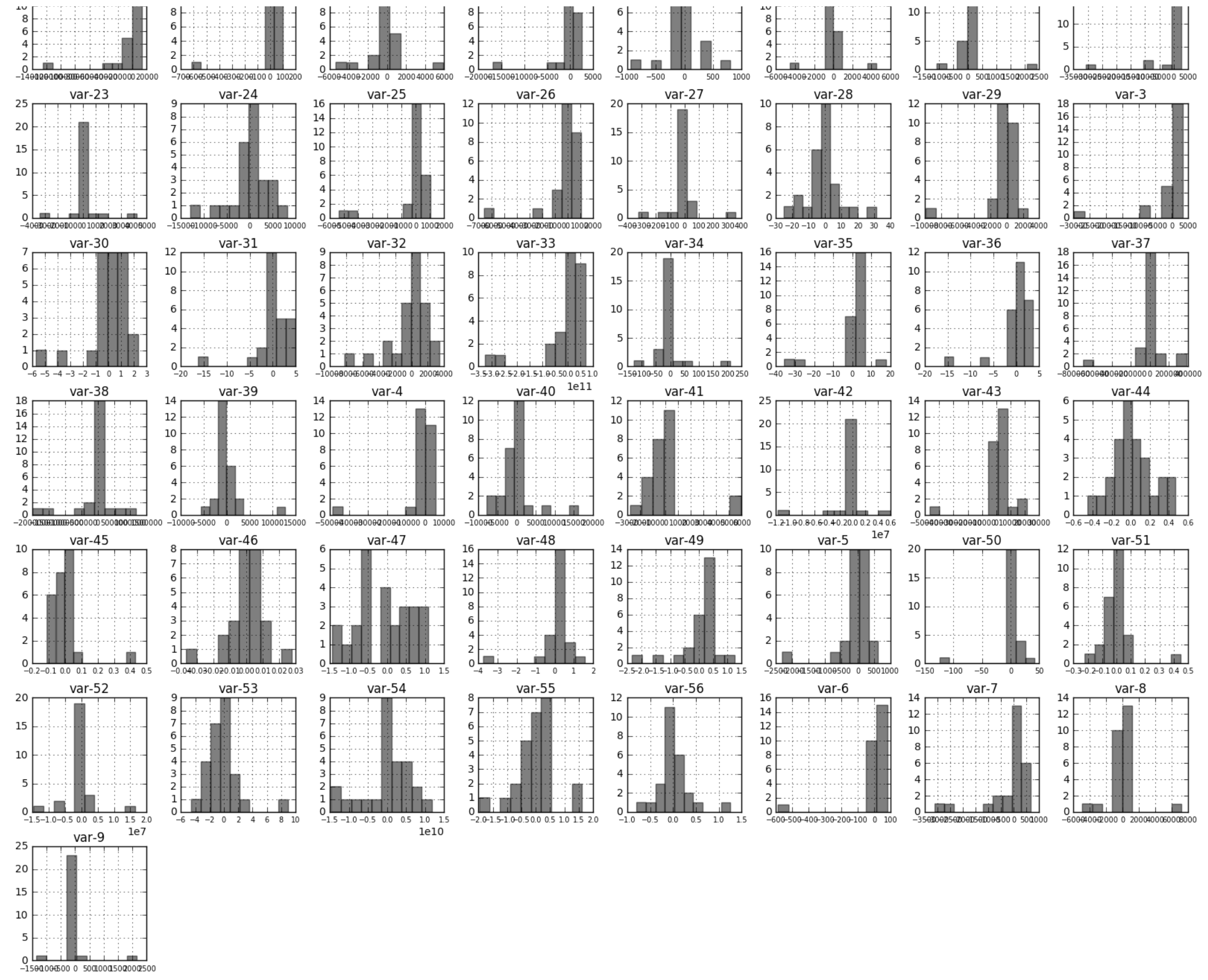
We can see that our information varies a lot with respect to the mean and variance

In [27]:

```
%matplotlib inline
fig = plt.figure()
dg.diff().hist(color='k', alpha=0.5, figsize=(20, 20), xlabelsize = 7)
plt.savefig('histogram.png')
```

```
<matplotlib.figure.Figure at 0x27b129c80f0>
```





With respect to the distribution of our data, we can see from the histograms shown above that it's distribution in most cases is not normal and it tends to be focus of a particular value.

Correlations

Next we explore the correlations between our variables. We won't take into consideration correlation between variables related with production of energy and consumption of energy, and any conbination of those, and only correlation between variables related with "characteristics" of a country and consumption or production.

In [28]:

```
#We correlate the information for all year and all variables
g = dg.reset_index(drop=True)

#Correlating with all variables
cor = g.corr().abs() #Considering either possitive or negative correlations
cor.values[[np.arange(cor.shape[0])]*2] = 0 #Setting the values in the diagonal equal to zero
cor = cor.where(np.triu(np.ones(cor.shape)).astype(np.bool)) #removing the lower triangular part
cor = cor.unstack() #converting into an array
cor = cor.sort_values(kind="quicksort",ascending =False) #Sorting the array
cor = cor.to_frame().reset_index() #This DataFrame contains the correlation of all variables

#We will not consider correlations between "consumption" and "production" or any conbination of su
ch sets of variables
com = cor.isin(consumption+production) #This DF will help us to compare
cor = cor[(com['level_0'] != com['level_1'])] #DataFrame containing our correlations
cor.dropna(inplace=True)

#Changing the names of 'var-X' to it's real name
A=cor['level_0'].map(dict_var)
B=cor['level_1'].map(dict_var)
C=cor[0]
```

```
cor = pd.DataFrame([A,B,C]).T.rename(columns={'level_0':'Variable #1','level_1':'Variable #2', 0:'C
orrelation' })
```

In [29]:

```
cor.head(10)
```

Out[29]:

	Variable #1	Variable #2	Correlation
12	imports	Consumption by households	0.96767
13	Mobile cellular subscriptions (per 100 people)	Biodiesel Production	0.965561
15	imports	Final energy consumption	0.962085
16	Mobile cellular subscriptions (per 100 people)	Biodiesel consumption	0.958667
22	imports	Gross production	0.945788
27	Losses	Gross production	0.935805
30	exports	Consumption by households	0.934213
31	Improved water source	Biodiesel Production	0.933495
34	Chickens Stocks	total wind production	0.930463
37	Improved water source	Biodiesel consumption	0.927918

We can see some interesting correlation in our data! We show only the 10 most correlated variables, either possitive or negative correlated

Supervised Learning: Model 1 - Multivariable Regression

We will use Gradient Boosting as our first Supervised model to fit our information into one model that could provides us the prediction for consumption and production of energy with respected to the characteristics of a given country.

We split our information. We will use information from 1990 until 2011 to train our model and we will test it's accuracy with the information contained in the years 2012-2016

In [30]:

```
#We will use SciKit lear to fit a model. We import some libraries
from sklearn.datasets import make_regression
from sklearn.multioutput import MultiOutputRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.preprocessing import Imputer
```

In [31]:

```
dg.sort_index(inplace=True) #We sort the year located in the index
```

In [32]:

```
#We will train our model for the following years
dg.index.values[:22]
```

Out[32]:

```
array([1990., 1991., 1992., 1993., 1994., 1995., 1996., 1997., 1998.,
       1999., 2000., 2001., 2002., 2003., 2004., 2005., 2006., 2007.,
       2008., 2009., 2010., 2011.])
```

In [35]:

```
#We will test our model with the following years
year_tested = dg.index.values[22:27]
```

In [34]:

```
#Input Data (X)
X = dg[characteristics].values

#Target values  (Y)
Y1 = dg[production].values
Y2 = dg[consumption].values

#We need to handle the missing values! We do this using a Scikit Library called Imputer(), we will
use the mean of each column
imputer = Imputer()

#Filling missing values
X = imputer.fit_transform(X) # X
Y1 = imputer.fit_transform(Y1) # Y
Y2 = imputer.fit_transform(Y2) # Y

#Training variables
X_train = X[:22, ...]
Y1_train = Y1[:22, ...]
Y2_train = Y2[:22, ...]

#Testing variables
X_test = X[22:27, ...]
Y1_test = Y1[22:27, ...]
Y2_test = Y2[22:27, ...]


for i in range(5):
    #We will test with the information transformed
    test = X_test[i].reshape(1, -1)

    #Results expected:
    out_1 = Y1_test[i].reshape(1, -1)
    out_2 = Y2_test[i].reshape(1, -1)

    #Production model
    Pred_production = MultiOutputRegressor(GradientBoostingRegressor(random_state=0)).fit(X_train,
Y1_train).predict(test)

    #Consumption model
    Pred_consumption = MultiOutputRegressor(GradientBoostingRegressor(random_state=0)).fit(X_train,
Y2_train).predict(test)

    #Overall relative error in our prediction for PRODUCTION
    err_pro = round((abs(Pred_production - out_1)/out_1).mean()*100,0)
    print('Prediction for production error '+ str(year_tested[i]) + ' :' + str(err_pro) + '% ' )

    #overall relative error in our prediction for CONSUMPTION
    err_con = round((abs(Pred_consumption - out_2)/out_2).mean()*100,0)
    print('Prediction for consumption error '+ str(year_tested[i]) + ' :'  + str(err_con) + '% '+ '
\n')
```

```
Prediction for production error 2012.0 :4.0%
Prediction for consumption error 2012.0 :9.0%

Prediction for production error 2013.0 :13.0%
Prediction for consumption error 2013.0 :19.0%

Prediction for production error 2014.0 :16.0%
Prediction for consumption error 2014.0 :23.0%

Prediction for production error 2015.0 :18.0%
Prediction for consumption error 2015.0 :30.0%

Prediction for production error 2016.0 :7637.0%
Prediction for consumption error 2016.0 :16723.0%
```

We can see that our model performs fairly well for the years 2012-2015, but we get a major discrepancy in 2016. We will exclude this year in the following analisys.

Supervised Learning: Model 2 - LSTM using TensorFlow

For our next model, we will use a Long short-term memory (LSTM) Recurrent Neural Network. Our aim is to try to exploit the hidden information contained in all variables through all the available years. Since the LSTM RNN "learns" from the past, we will try to train it in an adequate way to perform predictions on the future consumption and production of energy.

We have gotten our code from github and used Tensor Flow to create, train and test our models.

Code source: https://github.com/hzy46/TensorFlow-Time-Series-Examples

In [42]:

```
dg.drop(2016,inplace=True) #We drop 2016 since we see that the information contained there is not
representative
```

In [43]:

```
#We normalice our data to have a better performance in our model
from sklearn import preprocessing

x = dg.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
dg_scaled = pd.DataFrame(x_scaled)
dg_scaled.to_csv('Energy_dataframe_scaled.csv',header=False)
```

Running the following code using python and TensorFlow will give us the graphs shown bellow

```
import abc

from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

from os import path

import numpy
import tensorflow as tf

from tensorflow.contrib.timeseries.python.timeseries import estimators as ts_estimators
from tensorflow.contrib.timeseries.python.timeseries import model as ts_model
import matplotlib
matplotlib.use("agg")
import matplotlib.pyplot as plt

class _LSTMModel(ts_model.SequentialTimeSeriesModel):
  """A time series model-building example using an RNNCell."""

  def __init__(self, num_units, num_features, dtype=tf.float32):
    """Initialize/configure the model object.
    Note that we do not start graph building here. Rather, this object is a
    configurable factory for TensorFlow graphs which are run by an Estimator.
    Args:
      num_units: The number of units in the model's LSTMCell.
      num_features: The dimensionality of the time series (features per
        timestep).
      dtype: The floating point data type to use.
    """
    super(_LSTMModel, self).__init__(
        # Pre-register the metrics we'll be outputting (just a mean here).
        train_output_names=["mean"],
        predict_output_names=["mean"],
        num_features=num_features,
        dtype=dtype)
    self._num_units = num_units
    # Filled in by initialize_graph()
    self._lstm_cell = None
    self._lstm_cell_run = None
    self._predict_from_lstm_output = None

  def initialize_graph(self, input_statistics):
    """Save templates for components, which can then be used repeatedly.
    This method is called every time a new graph is created. It's safe to start
    adding ops to the current default graph here, but the graph should be
    constructed from scratch.
    Args:
      input_statistics: A math_utils.InputStatistics object.
    """
    super(_LSTMModel, self).initialize_graph(input_statistics=input_statistics)
    self._lstm_cell = tf.nn.rnn_cell.LSTMCell(num_units=self._num_units)
    # Create templates so we don't have to worry about variable reuse.
    self._lstm_cell_run = tf.make_template(
        name_="lstm_cell",
        func_=self._lstm_cell,
        create_scope_now_=True)
    # Transforms LSTM output into mean predictions.
    self._predict_from_lstm_output = tf.make_template(
        name_="predict_from_lstm_output",
        func_=lambda inputs: tf.layers.dense(inputs=inputs, units=self.num_features),
        create_scope_now_=True)

  def get_start_state(self):
    """Return initial state for the time series model."""
    return (
        # Keeps track of the time associated with this state for error checking.
        tf.zeros([], dtype=tf.int64),
        # The previous observation or prediction.
        tf.zeros([self.num_features], dtype=self.dtype),
        # The state of the RNNCell (batch dimension removed since this parent
        # class will broadcast).
        [tf.squeeze(state_element, axis=0)
         for state_element
         in self._lstm_cell.zero_state(batch_size=1, dtype=self.dtype)])

  def _transform(self, data):
    """Normalize data based on input statistics to encourage stable training."""
    mean, variance = self._input_statistics.overall_feature_moments
    return (data - mean) / variance

  def _de_transform(self, data):
    """Transform data back to the input scale."""
    mean, variance = self._input_statistics.overall_feature_moments
    return data * variance + mean

  def _filtering_step(self, current_times, current_values, state, predictions):
    """Update model state based on observations.
    Note that we don't do much here aside from computing a loss. In this case
    it's easier to update the RNN state in _prediction_step, since that covers
    running the RNN both on observations (from this method) and our own
    predictions. This distinction can be important for probabilistic models,
    where repeatedly predicting without filtering should lead to low-confidence
    predictions.
    Args:
      current_times: A [batch size] integer Tensor.
      current_values: A [batch size, self.num_features] floating point Tensor
        with new observations.
      state: The model's state tuple.
      predictions: The output of the previous `_prediction_step`.
    Returns:
      loss (note that we could also return other measures of goodness of fit,
      although only "loss" will be optimized).
    """
    state_from_time, prediction, lstm_state = state
    with tf.control_dependencies(
            [tf.assert_equal(current_times, state_from_time)]):
      transformed_values = self._transform(current_values)
      # Use mean squared error across features for the loss.
      predictions["loss"] = tf.reduce_mean(
          (prediction - transformed_values) ** 2, axis=-1)
      # Keep track of the new observation in model state. It won't be run
      # through the LSTM until the next _imputation_step.
      new_state_tuple = (current_times, transformed_values, lstm_state)
    return (new_state_tuple, predictions)

  def _prediction_step(self, current_times, state):
    """Advance the RNN state using a previous observation or prediction."""
    _, previous_observation_or_prediction, lstm_state = state
    lstm_output, new_lstm_state = self._lstm_cell_run(
        inputs=previous_observation_or_prediction, state=lstm_state)
    next_prediction = self._predict_from_lstm_output(lstm_output)
    new_state_tuple = (current_times, next_prediction, new_lstm_state)
    return new_state_tuple, {"mean": self._de_transform(next_prediction)}

  def _imputation_step(self, current_times, state):
    """Advance model state across a gap."""
    # Does not do anything special if we're jumping across a gap. More advanced
    # models, especially probabilistic ones, would want a special case that
    # depends on the gap size.
    return state

  def _exogenous_input_step(
          self, current_times, current_exogenous_regressors, state):
    """Update model state based on exogenous regressors."""
    raise NotImplementedError(
        "Exogenous inputs are not implemented for this example.")


if __name__ == '__main__':
  tf.logging.set_verbosity(tf.logging.INFO)
  csv_file_name = path.join("./Energy_dataframe_scaled.csv")
  reader = tf.contrib.timeseries.CSVReader(
      csv_file_name,
      column_names=((tf.contrib.timeseries.TrainEvalFeatures.TIMES,) + (tf.contrib.timeserie
s.TrainEvalFeatures.VALUES,) * 57))
  train_input_fn = tf.contrib.timeseries.RandomWindowInputFn(
      reader, batch_size=4, window_size=10)

  #num_units: The number of units in the model's LSTMCell.

  estimator = ts_estimators.TimeSeriesRegressor(
      model=_LSTMModel(num_features=57, num_units=250),  #250 stands for number of cells in
our LSTM model
      optimizer=tf.train.AdamOptimizer(0.0005))

  estimator.train(input_fn=train_input_fn, steps=200) #200 is steps to predict
  evaluation_input_fn = tf.contrib.timeseries.WholeDatasetInputFn(reader)
  evaluation = estimator.evaluate(input_fn=evaluation_input_fn, steps=1)

  # Predict starting after the evaluation
  (predictions,) = tuple(estimator.predict(
      input_fn=tf.contrib.timeseries.predict_continuation_input_fn(
          evaluation, steps=5))) #We will predict the information for the next 5 years

  observed_times = evaluation["times"][0]
```

```
observed = evaluation["observed"][0, :, :]
evaluated_times = evaluation["times"][0]
evaluated = evaluation["mean"][0]
predicted_times = predictions['times']
predicted = predictions["mean"]
```

Unsupervised Learning: Clustering using Folium

Using the information extracted from k-means clustering for the year 2015 we will picture the clusters in a map

Countries Codes: We will get the countries codes iso_a3 to plot the clusters in a world map usin Folium

In [45]:

```
Countries = set(df.index.levels[0])

#This is just a temporary set to avoid errors while getting the ISO abbreviation of our countries
C_temp = set(Countries) - {'Brunei','Russia','Kosovo','Northern Marianas Islands',\
                          'Serbia and Montenegro','Venezuela','Central African Rep.','Syria',\
                         'Macedonia', 'Cape Verde','Faeroe Islands','British Virgin Islands',\
                         'United Kingdom','Bonaire','Netherlands Antilles','Moldova','Czech
Republic',\
                         'Iran','Falkland Is. (Malvinas)','Virgin Islands (U.S.)',\
                         'Micronesia','Tanzania','North Korea',"Cote d'Ivoire",'South
Korea','Bolivia','North America',\
                             'Czechoslovakia (former)','United States Virgin Islands','Channel
Islands'}

#Some codes of countries codes are introduce manually to avoid changing the name to a longer one
codes={'Macedonia' : 'MKD',
                'United Kingdom' : 'GBR',
                'Republic of Moldova' : 'MDA',
                'South korea' : 'KOR',
                'Brunei' : 'BRN',
                'Czech Republic' : 'CZE',
                "Côte d'Ivoire" : 'CIV'}

#We create a dictionary containing the names and codes of all countries using ****from iso3166 imp
ort countries***
from iso3166 import countries #Library to get the ISO codes of the countries

datadict = {i:list(countries.get(i))[2] for i in C_temp} #Dictionary containing iso_a3 codes for al
l of our countries
datadict.update(codes)
```

In [160]:

```
import json
import os
import folium
```

In [161]:

```
world = os.path.join('','world-Low-res.json')
geo_json_data = json.load(open(world))
```

In [162]:

```
clusters =  pd.read_csv('Data/'+'Cluster_2015.csv',usecols =['Country','Cluster 2015'] )
clusters.set_index('Country',inplace=True)
df2 = pd.DataFrame.from_dict(datadict, orient='index') #DataFrame containing the iso_a3 and
country name
df3 = pd.concat([df2, clusters], axis=1).reset_index().drop('index',axis=1).rename(columns={0:'Coun
try'}).set_index('Country')
```

In [163]:

```
#We get the dictionary of the codes and countries names from our Json file
d = {} #dictionary containing the iso_a3 codes for all the countries listed in the json file
for i in range(len(geo_json_data['features'])):
    d[geo_json_data['features'][i]['properties']['iso_a3']]=geo_json_data['features'][i]['propertie
s']['name_sort']
df1=pd.DataFrame.from_dict(d,orient='index') #From the JsonFile
```

In [180]:

```
df4 = df1.join(df3, how='outer').drop(0,axis=1).fillna(0,axis=1).reset_index().dropna().set_index('
index')
df_dict=df4['Cluster 2015'].to_dict()
del df_dict['-99']
```

In [228]:

```
import branca.colormap as cm

step = cm.StepColormap(['black','y','g','b','r'], vmin=0., vmax=5., index=[0,1,2,3,4], caption='ste
p')
step
```

Out[228]:

In [229]:

```
m = folium.Map( **{'location': [20, 0], 'zoom_start': 1.5}, tiles=None)

folium.GeoJson(geo_json_data,
        style_function=lambda feature: {
        'fillColor': step(df_dict[feature['properties']['adm0_a3_is']]),
        'color': 'black',
        'weight': 1,
        'dashArray': '2, 2'
    }).add_to(m)
m
```

Out[229]:

In [ ]:

```

```

In [ ]:

```

```