PfDS - Homework 2 - 2020 - Pandas

This week the homework assignment will test your ability to load and manipulate data with Pandas.
The goal is to develop some intuition on how to filter, arrange, and merge data. This will be usefull for the next homework assignment as well.
Fill the empty cells with your code and deliver a copy of this notebook to Moodle.

Remember to change the name of the notebook to "HW2-student_id.ipynb", replacing student_id by your student_id. Submit your homework with both a PDF of html version and the ipynb.
Remember to comment your code.

In [1]:

```
import numpy as np
import scipy
import pandas as pd
```

Download and Load the World Development Indicators data set

We will work with the World Development Indicators data set.
We download this data set from the world bank databank.
Hence, the very first step is to download the data to your computer, you can do this by running the following cell.

NOTE This cell may timeout on slower connections. If you recieve an error you will need to download the file manually by pasting the URL into your browser. After download the zip archive you will need to move it to the same folder as this notebook and then unzip it to have acees to the required files.

Alternatively you can copy and paste the url inside the .get() method into your browser.

In [2]:

```
# importing libraries
import requests, zipfile, io
from os import path

if not path.exists("WDIData.csv"): # Check if the file has already been downloaded
    #note this can take several minutes depending on your internet connection
    r = requests.get('http://databank.worldbank.org/data/download/WDI_csv.zip')
    z = zipfile.ZipFile(io.BytesIO(r.content))
    z.extractall()

    # let us free the variales we used above
    del z
    del r
else:
    print("File already in workspace.")
```

```
File already in workspace.
```

The above code downloads a zip archive to the working folder, which by default is the the location of this notebook in your computer.
Secondly, and since the document downloaded is a zip archive, it extracts the documents from the archive.
The contents include multiple .csv files, however we will be working only with the document 'WDIData.csv'.

In the cell bellow, use Pandas to open the file "WDIData.csv" and save it to a variable called 'wdi'.
NOTE If you see strange characters in the headings or text you may need to specify the option enconding, "ISO-8859-1" has worked previously. Find more information at https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)

In [3]:

```
# Upon opening the file, the encoding is UTF8-BOM
wdi = pd.read_csv( 'WDIData.csv', sep=',', encoding='utf-8-sig', quotechar = '"')
```

Check the top of the datframe to ensure it loaded correctly.

In [4]:
```
wdi.head()
```

Out[4]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	2012	2013	2014	2015	2016	2017	2018	2019	2020
0	Arab World	ARB	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	Arab World	ARB	Access to electricity (% of population)	EG.ELC.ACCS.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	Arab World	ARB	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	Arab World	ARB	Access to electricity, urban (% of urban popul...	EG.ELC.ACCS.UR.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	Arab World	ARB	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	30.27713	NaN	NaN	37.165211	NaN	NaN	NaN

5 rows × 66 columns

Download and Load the Penn World Table V9.0

We will additionally use data from the pwt v9.0 tables.
Again run the following cell to download the dataset. This time using the library urllib.

In [5]:
```
import urllib
from os import path

if not path.exists('pwt90.xlsx'): # Check if the file has already been downloaded
    urllib.request.urlretrieve("https://www.rug.nl/ggdc/docs/pwt90.xlsx", "pwt90.xlsx")
else:
    print("File already in workspace.")
```

```
File already in workspace.
```

In the following cell, open and read the file 'pwt90.xlsx' and save it into variable 'pwt'.
Remember that pandas has a method to read excel files, different to the moethod to read csv files, and secondly we need to specity the sheet we want to load data from.

In [6]:
```
# The data that we want to load is in the sheet name "Data"
pwt = pd.read_excel( 'pwt90.xlsx', sheet_name='Data' )
```

Check the top of the datframe to ensure it loaded correctly.

In [7]:
```
pwt.head()
```

Out[7]:

	countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp	avh	hc	...	csh_g	csh_x	csh_m	csh_r	pl_c	pl_i	pl_g	pl_x	pl_m	pl_k
0	ABW	Aruba	Aruban Guilder	1950	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	ABW	Aruba	Aruban Guilder	1951	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	ABW	Aruba	Aruban Guilder	1952	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	ABW	Aruba	Aruban Guilder	1953	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	ABW	Aruba	Aruban Guilder	1954	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

5 rows × 47 columns

Data Wrangling

Now that we have loaded our data into variable 'wdi', we are ready to start playing with it.
Start by printing all column values in the cell bellow.

In [8]:

```
# Assuming the "column values" actually means "column names"
wdi.columns
```

Out[8]:

```
Index(['Country Name', 'Country Code', 'Indicator Name', 'Indicator Code',
       '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '1968',
       '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '1977',
       '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '1986',
       '1987', '1988', '1989', '1990', '1991', '1992', '1993', '1994', '1995',
       '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '2004',
       '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '2013',
       '2014', '2015', '2016', '2017', '2018', '2019', '2020', 'Unnamed: 65'],
      dtype='object')
```

Next, list the values in the column 'Country Name'.
You will get a list with repeated values, delete all duplicates to ease your analysis.

Tip: see the method '.drop_duplicates()' https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop_duplicates.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.drop_duplicates.html).

In [9]:

```
wdi['Country Name'].drop_duplicates()
```

Out[9]:

```
0                               Arab World
1437                Caribbean small states
2874        Central Europe and the Baltics
4311            Early-demographic dividend
5748                   East Asia & Pacific
                        ...
372183              Virgin Islands (U.S.)
373620                 West Bank and Gaza
375057                        Yemen, Rep.
376494                             Zambia
377931                           Zimbabwe
Name: Country Name, Length: 264, dtype: object
```

You might notice that while the bottom rows represent Countries, the top rows represent aggregates of countries (e.g., world regions).
However we will be only interested in working with country-level data, and as such we need to filter out all unecessary rows.

Save all the values of column 'Country Name' in variable 'cnames'.
Delete all duplicate values.
Print the first 50 values in cnames (remember you can use slice here).

```
In [10]: # Resetting index so we can see the index of the World Regions
         cnames = wdi['Country Name'].drop_duplicates().reset_index(drop=True)
         cnames[0:50]
```

```
Out[10]: 0                                            Arab World
         1                                Caribbean small states
         2                        Central Europe and the Baltics
         3                            Early-demographic dividend
         4                                   East Asia & Pacific
         5           East Asia & Pacific (excluding high income)
         6            East Asia & Pacific (IDA & IBRD countries)
         7                                             Euro area
         8                                 Europe & Central Asia
         9         Europe & Central Asia (excluding high income)
         10         Europe & Central Asia (IDA & IBRD countries)
         11                                       European Union
         12             Fragile and conflict affected situations
         13               Heavily indebted poor countries (HIPC)
         14                                          High income
         15                                            IBRD only
         16                                     IDA & IBRD total
         17                                            IDA blend
         18                                             IDA only
         19                                            IDA total
         20                            Late-demographic dividend
         21                            Latin America & Caribbean
         22    Latin America & Caribbean (excluding high income)
         23    Latin America & the Caribbean (IDA & IBRD coun...
         24         Least developed countries: UN classification
         25                                  Low & middle income
         26                                           Low income
         27                                  Lower middle income
         28                           Middle East & North Africa
         29    Middle East & North Africa (excluding high inc...
         30    Middle East & North Africa (IDA & IBRD countries)
         31                                        Middle income
         32                                        North America
         33                                       Not classified
         34                                         OECD members
         35                                   Other small states
         36                          Pacific island small states
         37                            Post-demographic dividend
         38                             Pre-demographic dividend
         39                                         Small states
         40                                           South Asia
         41                              South Asia (IDA & IBRD)
         42                                   Sub-Saharan Africa
         43           Sub-Saharan Africa (excluding high income)
         44            Sub-Saharan Africa (IDA & IBRD countries)
         45                                  Upper middle income
         46                                                World
         47                                          Afghanistan
         48                                              Albania
         49                                              Algeria
         Name: Country Name, dtype: object
```

You can verify, that the first 47 values in cnames 'Country Name' do not correspond to countries, but aggregates.
In the next cell filter out, from 'wdi', rows in which 'Country Name' represents an aggregate of countries.

Tip1 : You can use the negation of .isin() to perform a boolean filter over the rows of the DataFrame, see an example at https://erikrood.com/Python_References/rows_cols_python.html (https://erikrood.com/Python_References/rows_cols_python.html)
Tip2 : You can also perform this action by slicing out all rows unecessary rows.

```
In [11]: # Generate a Pandas Series with all the Country Names which represent a region
         Regions = cnames[0:47]

         # Extract from the DF the Country Names which are not in the previous Series
         wdi = wdi[ ~wdi['Country Name'].isin(Regions) ]
```

Check the top of the wdi dataframe now only has countries and not aggregates of countries.

In [12]:
```
wdi.head()
```

Out[12]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965	...	2012	2013	2014	2015	2016	
67539	Afghanistan	AFG	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	24.080000	26.170000	27.990000	30.100000	32.44000	
67540	Afghanistan	AFG	Access to electricity (% of population)	EG.ELC.ACCS.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	69.100000	68.933266	89.500000	71.500000	97.70000	9
67541	Afghanistan	AFG	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	60.849157	61.282199	86.500512	64.573354	97.09936	9
67542	Afghanistan	AFG	Access to electricity, urban (% of urban popul...	EG.ELC.ACCS.UR.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	95.000000	92.673767	98.700000	92.500000	99.50000	9
67543	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	9.961000	NaN	NaN	1

5 rows × 66 columns

Reset the indexes of 'wdi', you can use the method reset_index(), see https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.reset_index.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.reset_index.html).
Perform this operation In Place.

In [13]:
```
# drop Parameter will remove the previous index from the Dataframe
# inplace parameter will remove the need to assign the result to a new DF
wdi.reset_index(drop=True, inplace=True)
```

Show that the indexes have been reseted.

In [14]:
```
wdi.index
# The index is now a range between 0 and 311829
# Since we used the drop parameter, the sentence below does not apply to our dataframe
```

Out[14]:
```
RangeIndex(start=0, stop=311829, step=1)
```

Note that when reseting the index, pandas appends a new column at the begining of the data frame, which holds the previous index values.

Indicator Codes and Indicator Name

Select the columns 'Indicator Name' and 'Indicator Code'.
Delete all duplicates, and then Print the top 5 and bottom 5 values.
Tip: You should be able to do everything in a single line of code for the top 5 values and a single line for the bottom 5 values.

In [15]:
```
# Extract the 2 columns requested from the DF
# Drop duplicates
# Use head(5) to extract the first 5 elements
wdi[['Indicator Name','Indicator Code']].drop_duplicates().head(5)
```

Out[15]:

	Indicator Name	Indicator Code
0	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS
1	Access to electricity (% of population)	EG.ELC.ACCS.ZS
2	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS
3	Access to electricity, urban (% of urban popul...	EG.ELC.ACCS.UR.ZS
4	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS

In [16]:
```
# Extract the 2 columns requested from the DF
# Drop duplicates
# Use tail(5) to extract the last 5 elements
wdi[['Indicator Name','Indicator Code']].drop_duplicates().tail(5)
```

Out[16]:

	Indicator Name	Indicator Code
1432	Women who believe a husband is justified in be...	SG.VAW.NEGL.ZS
1433	Women who believe a husband is justified in be...	SG.VAW.REFU.ZS
1434	Women who were first married by age 15 (% of w...	SP.M15.2024.FE.ZS
1435	Women who were first married by age 18 (% of w...	SP.M18.2024.FE.ZS
1436	Women's share of population ages 15+ living wi...	SH.DYN.AIDS.FE.ZS

Create a new DataFrame named 'indicators' made up of the columns 'Indicator Name' and 'Indicator Code'.
Delete all duplicated entries.
Set the column 'Indicator Code' as the index of 'indicators'.
The ouput should be a DataFrame with 1437 rows.
Try to perform all these steps in a single line of code.

In [17]:
```
# Extract the 2 columns requested from the DF
# Drop duplicates
# Set the index of the dataframe as the 'Indicator Code' column
indicators = wdi[['Indicator Name','Indicator Code']].drop_duplicates().set_index('Indicator Code')
indicators
```

Out[17]:

	Indicator Name
Indicator Code	
EG.CFT.ACCS.ZS	Access to clean fuels and technologies for coo...
EG.ELC.ACCS.ZS	Access to electricity (% of population)
EG.ELC.ACCS.RU.ZS	Access to electricity, rural (% of rural popul...
EG.ELC.ACCS.UR.ZS	Access to electricity, urban (% of urban popul...
FX.OWN.TOTL.ZS	Account ownership at a financial institution o...
...	...
SG.VAW.NEGL.ZS	Women who believe a husband is justified in be...
SG.VAW.REFU.ZS	Women who believe a husband is justified in be...
SP.M15.2024.FE.ZS	Women who were first married by age 15 (% of w...
SP.M18.2024.FE.ZS	Women who were first married by age 18 (% of w...
SH.DYN.AIDS.FE.ZS	Women's share of population ages 15+ living wi...

1437 rows × 1 columns

The 'indicators' DataFrame can operate now as a dictionary.
By passing an 'Indicator Code' (key) it returns the associated 'Indicator Name' (value).

Using 'indicators' DataFrame, find the 'Indicator Code' associated with the following observables:

1. 'Population', find the 'Indicator Code' of the total population in a country;
2. 'GDP', find the GDP measured in current US Dollars;
3. 'GINI index'

Hint: You can use the method STRING.str.contains('substring') to check whether a string contains a substring, like "GDP", also note that the match is case sensitive.

In [18]:
```
# By analyzing the results for the Indicator Names which contain 'Population',
# we find that the Indicator Name that matches the criteria defined in the exercise is
# 'Population, total'. So, we use the contain method to search for the Indicator Code

# Then, to extract the Indicator Code, we use the index attribute and take out the first row
IndCode_Population = indicators.loc[ indicators['Indicator Name'].str.contains('Population, total') ].index[0]
print("The indicator code for Total Population is {}.".format(IndCode_Population))
```

```
The indicator code for Total Population is SP.POP.TOTL.
```

In [19]:
```
# Same logic as above, by investigating the dataset, we find that the indicator name
# that we need to search is the 'GDP (current US$)'
# In this case, the regex=False is necessary so that the method does not mistake the
# string inside parenthesis as a capture group
IndCode_GDP = indicators.loc[ indicators['Indicator Name'].str.contains('GDP (current US$)', regex = False) ].index[0]
print("The indicator code for GDP in US$ is {}.".format(IndCode_GDP))
```

```
The indicator code for GDP in US$ is NY.GDP.MKTP.CD.
```

```
In [20]: # Same logic as above, by investigating the dataset, we find that the indicator name
         # that we need to search is the 'Gini index'
         IndCode_GINI = indicators.loc[ indicators['Indicator Name'].str.contains('Gini index', regex = False) ].index[0]
         print("The indicator code for GINI Index is {}.".format(IndCode_GINI))

         The indicator code for GINI Index is SI.POV.GINI.
```

```
In [21]: # Storing indicator codes in a list
         IndicatorCodes = [IndCode_Population,IndCode_GDP,IndCode_GINI]
         IndicatorCodes

Out[21]: ['SP.POP.TOTL', 'NY.GDP.MKTP.CD', 'SI.POV.GINI']
```

Extracting and Cleaning data from WDI and PWT

From 'wdi' extract the columns 'Indicator Code', 'Country Code', and '2012'. Save the output in variable 'wdi_sample'

Tip1: You should be able to perfom all operations in a single line of code.
Tip2: Use the method .loc[] to extract a row with a specified index value, see https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html (https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.loc.html).

```
In [22]: # Using loc to extract all the rows and the 3 columns required
         wdi_sample = wdi.loc[ : , ['Indicator Code', 'Country Code', '2012'] ]
         wdi_sample

Out[22]:
                Indicator Code  Country Code       2012
              0   EG.CFT.ACCS.ZS           AFG  24.080000
              1   EG.ELC.ACCS.ZS           AFG  69.100000
              2 EG.ELC.ACCS.RU.ZS          AFG  60.849157
              3 EG.ELC.ACCS.UR.ZS          AFG  95.000000
              4    FX.OWN.TOTL.ZS          AFG        NaN
            ...              ...           ...        ...
         311824    SG.VAW.NEGL.ZS          ZWE        NaN
         311825    SG.VAW.REFU.ZS          ZWE        NaN
         311826  SP.M15.2024.FE.ZS         ZWE        NaN
         311827  SP.M18.2024.FE.ZS         ZWE        NaN
         311828   SH.DYN.AIDS.FE.ZS        ZWE  59.200000

         311829 rows × 3 columns
```

Select from 'wdi_sample' the lines associated with all the Indicator Codes that you found above, which concern the data of the 'GINI index', 'GDP', and 'Population total'.

```
In [23]: # Extract from the DF the rows where 'Indicator Code' exists in the previously defined list
         wdi_sample = wdi_sample[ wdi_sample['Indicator Code'].isin(IndicatorCodes) ]
         wdi_sample

Out[23]:
                Indicator Code  Country Code          2012
            467 NY.GDP.MKTP.CD           AFG  2.000160e+10
            491    SI.POV.GINI           AFG           NaN
           1061    SP.POP.TOTL           AFG  3.116138e+07
           1904 NY.GDP.MKTP.CD           ALB  1.231983e+10
           1928    SI.POV.GINI           ALB  2.900000e+01
            ...            ...           ...           ...
         309446    SI.POV.GINI           ZMB           NaN
         310016    SP.POP.TOTL           ZMB  1.446512e+07
         310859 NY.GDP.MKTP.CD           ZWE  1.711485e+10
         310883    SI.POV.GINI           ZWE           NaN
         311453    SP.POP.TOTL           ZWE  1.311513e+07

         651 rows × 3 columns
```

Create a pivot table, in which values are the column '2012', the index is the 'Country Code', and the columns are the Indicator Codes.

You can use the function pivot_table() from Pandas, see http://pandas.pydata.org/pandas-docs/stable/generated/pandas.pivot_table.html (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.pivot_table.html).

In [24]:
```python
# Pivot the previously defined dataframe wdi_sample with:
# 1. Index 'Country Code'
# 2. Columns 'Indicator Code'
# 3. Values '2012'
wdi_sample = pd.pivot_table(wdi_sample, values='2012', index=['Country Code'], columns=['Indicator Code'])
wdi_sample
```

Out[24]:

Indicator Code	NY.GDP.MKTP.CD	SI.POV.GINI	SP.POP.TOTL
Country Code			
ABW	2.534637e+09	NaN	102560.0
AFG	2.000160e+10	NaN	31161376.0
AGO	1.280529e+11	NaN	25107931.0
ALB	1.231983e+10	29.0	2900401.0
AND	3.188809e+09	NaN	82427.0
...	...	...	...
XKX	6.499807e+09	29.0	1807106.0
YEM	3.540134e+10	NaN	24473178.0
ZAF	3.963327e+11	NaN	52834005.0
ZMB	2.550306e+10	NaN	14465121.0
ZWE	1.711485e+10	NaN	13115131.0

216 rows × 3 columns

Rename the column names of wdi_sample to 'Population', 'GDP', and 'GINI', accordingly.

In [25]:
```python
# Using a dict, rename the Indicator Codes to the proper measure definition
wdi_sample.rename({'SP.POP.TOTL':'Population', 'NY.GDP.MKTP.CD':'GDP', 'SI.POV.GINI':'GINI'}, axis='columns',inplace=True)
wdi_sample
```

Out[25]:

Indicator Code	GDP	GINI	Population
Country Code			
ABW	2.534637e+09	NaN	102560.0
AFG	2.000160e+10	NaN	31161376.0
AGO	1.280529e+11	NaN	25107931.0
ALB	1.231983e+10	29.0	2900401.0
AND	3.188809e+09	NaN	82427.0
...	...	...	...
XKX	6.499807e+09	29.0	1807106.0
YEM	3.540134e+10	NaN	24473178.0
ZAF	3.963327e+11	NaN	52834005.0
ZMB	2.550306e+10	NaN	14465121.0
ZWE	1.711485e+10	NaN	13115131.0

216 rows × 3 columns

From 'pwt' select only the values of the year 2012.
Then, extract the columns 'countrycode' and 'hc' into a new variable 'pwt_sample'.
Rename 'countrycode' to 'Country Code', so that it matches the same column in 'wdi_sample'
Note that here 'hc' stands for the Human Capital Index.

In [26]:
```
pwt
```

Out[26]:

	countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp	avh	hc	...	csh_g	csh_x	csh_m	csh
0	ABW	Aruba	Aruban Guilder	1950	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
1	ABW	Aruba	Aruban Guilder	1951	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
2	ABW	Aruba	Aruban Guilder	1952	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
3	ABW	Aruba	Aruban Guilder	1953	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
4	ABW	Aruba	Aruban Guilder	1954	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	Na
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
11825	ZWE	Zimbabwe	US Dollar	2010	20652.718750	21053.855469	13.973897	6.298438	NaN	2.372605	...	0.127251	0.214657	-0.454497	0.0144
11826	ZWE	Zimbabwe	US Dollar	2011	20720.435547	21592.298828	14.255592	6.518841	NaN	2.415823	...	0.189860	0.219809	-0.625170	0.0043
11827	ZWE	Zimbabwe	US Dollar	2012	23708.654297	24360.527344	14.565482	6.248271	NaN	2.459828	...	0.178643	0.225631	-0.479897	-0.0769
11828	ZWE	Zimbabwe	US Dollar	2013	27011.988281	28157.886719	14.898092	6.287056	NaN	2.504635	...	0.162252	0.174443	-0.436145	-0.0000
11829	ZWE	Zimbabwe	US Dollar	2014	28495.554688	29149.708984	15.245855	6.499974	NaN	2.550258	...	0.253410	0.147346	-0.349806	-0.0200

11830 rows × 47 columns

In [27]:
```
# Extract the values of pwt in which year == 2012
# Extract the 'countrycode' and 'hc' columns
# Using a dict, rename the columns to match the wdi_sample DF
pwt_sample = pwt.loc[pwt['year'] == 2012][['countrycode','hc']].rename({"countrycode":"Country Code"},axis='columns')
```

In [28]:
```
# Setting index as 'Country Code' to match wdi_sample DF
pwt_sample.set_index('Country Code',inplace=True, drop=True)
pwt_sample
```

Out[28]:

	hc
Country Code	
ABW	NaN
AGO	1.431295
AIA	NaN
ALB	2.917346
ARE	2.723864
...	...
VNM	2.532614
YEM	1.488720
ZAF	2.596012
ZMB	2.330032
ZWE	2.459828

182 rows × 1 columns

Finally, create a new dataframe named 'data' that contains the columns from wdi_sample and pwt_sample, matched by 'Country Code'. Use the method concat(), and make sure both dataframes have the same index ('Country Code').

In [29]:
```
# Using concat to join the 2 dataframes by using the index as the key
data = pd.concat([wdi_sample, pwt_sample], axis=1)
data
```

Out[29]:

	GDP	GINI	Population	hc
ABW	2.534637e+09	NaN	102560.0	NaN
AFG	2.000160e+10	NaN	31161376.0	NaN
AGO	1.280529e+11	NaN	25107931.0	1.431295
ALB	1.231983e+10	29.0	2900401.0	2.917346
AND	3.188809e+09	NaN	82427.0	NaN
...	...	...	...	...
ZMB	2.550306e+10	NaN	14465121.0	2.330032
ZWE	1.711485e+10	NaN	13115131.0	2.459828
AIA	NaN	NaN	NaN	NaN
MSR	NaN	NaN	NaN	NaN
TWN	NaN	NaN	NaN	3.135961

219 rows × 4 columns

Perform the necessary manipualtions to answer the following questions, unless otherwise stated you can use the country codes to represent the countries in your solutions:

1. Which countries have a population size of 10 million habitations +/- 1 million? List just the country codes in your solution.
2. What is the average and the standard deviation in GDP of countries listed in 1?
3. What is the average and the standard deviation in the GDP per capita of countries listed in 1?
4. Consider the following classification of country size:
   Tiny - population < 1 000 000
   Very Small - 1 000 000 <= population < 5 000 000
   Small - 5 000 000 <= population < 15 000 000
   Medium - 15 000 000 <= population < 30 000 000
   Large - 30 000 000 <= population < 100 000 000
   Huge - 100 000 000 <= population
   What is the average and the standard deviation in the GDP per capita of countries in each classification of size?
5. Create a function that will take a dataframe and a column name, and **return** a series with binary values inidicating whether the values from the column are above the mean value of that column (indicated with a value of 1) or 0 otherwise. If the value in the column is missing (NaN) the value in the series should also be missing (NaN). *Hint:* remember how to check if something is None and that we can return None.
6. What is the average GDP per capita of countries grouped both by size classification and whether or not the human capital is above the average?
7. What is the average GDP per capita of countries grouped both by whether or not the human capital is above the average and whether or not the gini coefficient is above average?
8. What is the **name of the country** that has the highest GDP per capita with both a gini coefficient below average and a level of human capital below average?
9. What is the **name of the country** that has the highest GDP per capita with both a gini coefficient below average for its size classification and a level of human capital below average for its size classification?
10. What is the **name of the country** that had the largest % increase in GDP between 1980 and 2010? *HINT:* You will need to use the wdi dataframe to calculate.
11. What is the **name of the country and the year** that had the largest % average increase in GDP over the previous four years, between 1980 and 2010. You will need to use the wdi dataframe to calculate and you should not loop through the dataframe.
12. What is the **name of the country and the year** that had the largest % average increase in hc over the previous four years, between 1980 and 2010. You will need to use the wdi dataframe to calculate and you should not loop through the dataframe.

Write the code necessary to answer each question in a single cell.
Print the answer at the end of that cell and the question number.

In [30]:
```
# Extract from the data DF the rows in which Population falls between 9 million and 11 million
# Extract the index of the resulting DataFrame, which are the country codes
l_countries = list(data[ ( data['Population'] >= 9000000 ) & ( data['Population'] <= 11000000 ) ].index)
print("1. The Country Codes which have a population between 9 million and 11 million are: {}".format(', '.join(l_countries)))
```

```
1. The Country Codes which have a population between 9 million and 11 million are: ARE, AZE, BDI, BEN, BLR, BOL, CZE, DOM, GIN, H
TI, HUN, PRT, RWA, SSD, SWE, TUN
```

In [31]:
```
# Using the list of Country Codes above, use loc to extract the Country Codes by Index and the GCP column
GDP = data.loc[l_countries, 'GDP']

# Use the Series mean and std methods to compute the average and standard deviation
print("2. GDP -> Average(US$) : {:.2f} Standard Deviation(US$) : {:.2f}".format(GDP.mean(),GDP.std()))
```

```
2. GDP -> Average(US$) : 112227662509.42 Standard Deviation(US$) : 156266663181.87
```

In [32]:

```
# Create new column in the data DF which is the GDP per Capita
# by dividing the 2 Series, GDP and Population
data['GDPperCapita'] = data['GDP'] / data['Population']

# Using the list of Country Codes above, use loc to extract the Country Codes by Index and the GCPperCapita column
GDPperCapita = data.loc[l_countries, 'GDPperCapita']

# Use the Series mean and std methods to compute the average and standard deviation
print("3. GDPperCapita -> Average(US$) : {:.2f} Standard Deviation(US$) : {:.2f}".format(GDPperCapita.mean(),
                                                                                         GDPperCapita.std()))
```

```
3. GDPperCapita -> Average(US$) : 11520.59 Standard Deviation(US$) : 16478.47
```

In [33]:

```
# Create a function which Categorizes a Population by it's size
def PopulationClassifier(population):
    if population < 1000000:
        return 'Tiny'
    elif population >= 1000000 and population < 5000000:
        return 'Very Small'
    elif population >= 5000000 and population < 15000000:
        return 'Small'
    elif population >= 15000000 and population < 30000000:
        return 'Medium'
    elif population >= 30000000 and population < 100000000:
        return 'Large'
    else:
        return 'Huge'

# Create a new Column in the Dataframe by applying the previous function to the Population column
data['PopulationClassification'] = data.Population.apply(PopulationClassifier)

# Group by the data by the Population classification and compute the mean and std of the GDP per Capita
print("4.")
data.groupby('PopulationClassification').agg({'GDPperCapita': [np.mean,np.std]})
```

```
4.
```

Out[33]:

	GDPperCapita	
	mean	std
PopulationClassification		
Huge	14048.187429	18482.956690
Large	12752.302640	15771.237018
Medium	9840.172206	16578.387984
Small	16293.521093	23499.359291
Tiny	27360.258187	35964.116037
Very Small	13828.150873	16991.276306

In [34]:

```
# Fundamentally, the problem statement is flawed.
# It is asked to return a Series with Boolean values 1 and 0, while also keeping NaN when
# the source value is NaN.
# NaN is, by definition, a float64 and, a series has a homogeneous datatype. Meaning that,
# the combination of the 2 definitions will cause the problem statement to fail since
# as soon as a missing value is detected, the Series would automatically be a float64 series
# To get around this, i assumed from here on forward that the values 1.0 and 0.0 would mean
# True and False respectively.

def CheckAgainstMean(df,column):
    """
        This function receives a Pandas Dataframe and a column name (which needs to exist)
        and returns a Pandas Series where each element is a float indicating if
        each value in the column is higher or equal than the average of the column.

        Inputs:
            df : Pandas.Dataframe - Dataframe with all the columns
            column : string - Column name to check against the mean
        Outputs:
            Pandas.Series - Pandas series where each element indicates if the value in that
                            position is higher or equal than the average of the column.
                            1.0, Value is equal or higher than the average
                            0.0, Value is lower than the average
                            NaN, Value is undefined
    """
    import math

    if column in df:
        # Computes the average of the column
        ColumnAverage = df[column].mean()
        # Applys the lambda function to every element of the series.
        # The lambda function will perform the comparison with the average while also
        # propagating NaN values
        return df[column].apply(lambda x: None if math.isnan(x) else 1.0 if x >= ColumnAverage else 0.0)
    else:
        raise Exception("Column not found.")


# Testing with the hc column
Validate = CheckAgainstMean(data,'hc')
print("5.")
print(Validate)
```

```
5.
ABW    NaN
AFG    NaN
AGO    0.0
ALB    1.0
AND    NaN
      ...
ZMB    0.0
ZWE    0.0
AIA    NaN
MSR    NaN
TWN    1.0
Name: hc, Length: 219, dtype: float64
```

In [35]:
```
# Create a new column in the DF, which tells us if each value of hc is above or below the average
data['hc_aboveAVG'] = CheckAgainstMean(data,'hc')

# Groups the data by PopulationClassification and wether if the value of hc is higher/lower than the average
# Aggregate the results by computing the mean of the GDPperCapita column
print("6.")
data.groupby(['PopulationClassification','hc_aboveAVG']).agg({'GDPperCapita': [np.mean]})
```

```
6.
```

Out[35]:

		GDPperCapita
		mean
PopulationClassification	**hc_aboveAVG**	
Huge	**0.0**	2713.892188
	1.0	27649.341717
Large	**0.0**	3621.065430
	1.0	24219.917202
Medium	**0.0**	1255.067505
	1.0	19906.027069
Small	**0.0**	3037.758902
	1.0	33462.803716
Tiny	**0.0**	7265.613656
	1.0	40494.318080
Very Small	**0.0**	9405.686778
	1.0	18498.713941

In [36]:
```
# Create a new column in the DF, which tells us if each value of GINI is above or below the average
data['GINI_aboveAVG'] = CheckAgainstMean(data,'GINI')

print("7.")
# Groups the data by the hc_aboveAVG and GINI_aboveAVG columns
# Aggregate the results by computing the mean of the GDPperCapita column
data.groupby(['hc_aboveAVG','GINI_aboveAVG']).agg({'GDPperCapita': [np.mean]})
```

```
7.
```

Out[36]:

		GDPperCapita
		mean
hc_aboveAVG	**GINI_aboveAVG**	
0.0	**0.0**	6790.020549
	1.0	4286.412522
1.0	**0.0**	31043.244113
	1.0	10044.431693

In [37]:
```
# Create a mapping between Country Codes and Country Names
# Removing the duplicates in the DF so we can make sure that there is only 1 value for each Country Code
CountryCodeToCountryName = (wdi[['Country Name','Country Code']]
                             .drop_duplicates(subset=['Country Code'], keep='first')
                             .set_index('Country Code'))

# Extract from the Dataframe the rows where hc and Gini are below average
# Extract the GDPperCapitaColumn
# Extract the label of the maximum value, which is the Country Code
CountryCode = data[ (data['hc_aboveAVG'] == 0.0) & (data['GINI_aboveAVG'] == 0.0) ]['GDPperCapita'].idxmax()

# Using the Country code, access the DF with the mapping and extract the Country Name
# Since we removed the duplicates, the result of this loc will always be a series with 1 element
# So we can just get the values of that series and extract the 0 index to get the Country Name
CountryName = CountryCodeToCountryName.loc[CountryCode].values[0]

print("8. The country is {}.".format(CountryName))
```

```
8. The country is Portugal.
```

In [38]:

```python
# Reworking the previous function created to also work in a series
def CheckAgainstMean(pandasObject,column=None):
    """
        This function receives a Pandas Series/Dataframe and returns a Pandas Series where each element
        is a float indicating if each value in the column is higher or equal than the average
        of the series.

        Inputs:
            pandasObject: Pandas.Series/Pandas.Dataframe - If Dataframe, a column needs to be provided to
                                                           the function
        Outputs:
            Pandas.Series - Pandas series where each element indicates if the value in that
                            position is higher or equal than the average of the serie.
                            1.0, Value is equal or higher than the average
                            0.0, Value is lower than the average
                            NaN, Value is undefined
    """
    import math
    import pandas as pd

    if isinstance(pandasObject,pd.DataFrame):
        if column in pandasObject:
            # Computes the average of the column
            ColumnAverage = pandasObject[column].mean()
            # Applys the lambda function to every element of the series.
            # The lambda function will perform the comparison with the average while also
            # propagating NaN values
            return pandasObject[column].apply(lambda x: None if math.isnan(x) else 1.0 if x >= ColumnAverage else 0.0)
        else:
            raise Exception("Column not in Dataframe.")

    elif isinstance(pandasObject,pd.Series):
        SeriesAverage = pandasObject.mean()
        return pandasObject.apply(lambda x: None if math.isnan(x) else 1.0 if x >= SeriesAverage else 0.0)
    else:
        raise Exception("Not a Pandas Dataframe or a Pandas Series.")

# The following 2 lines of code will apply the same logic, one for hc and the other for GINI
# The goal is to create an aggregation of the HC/GINI by the population size and then,
# use the defined function to check which countries are above/lower than the average within their size
# classification
data['PerPopulationSize_hc_aboveAVG'] = data.groupby('PopulationClassification')['hc'].apply(CheckAgainstMean)
data['PerPopulationSize_GINI_aboveAVG'] = data.groupby('PopulationClassification')['GINI'].apply(CheckAgainstMean)

# From the dataframe extract the rows where the 2 new columns are False
# Extract the GDPperCapita column
# Extract the index of the maximum value of GDP per Capita
CountryCode = (data[ (data['PerPopulationSize_hc_aboveAVG'] == 0.0) &
                   (data['PerPopulationSize_GINI_aboveAVG'] == 0.0) ]['GDPperCapita'].idxmax())

# Using the Country code, access the DF with the mapping and extract the Country Name
# Since we removed the duplicates, the result of this loc will always be a series with 1 element
# So we can just get the values of that serie and extract the 0 index to get the Country Name
CountryName = CountryCodeToCountryName.loc[CountryCode].values[0]

print("9. The country is {}.".format(CountryName))
```

```
9. The country is Iraq.
```

In [39]:

```python
# Creating an Auxiliar Dataframe to help answer the following questions
# Copy wdi and set it's index to 'Country Name'
wdi_aux = wdi.copy().set_index('Country Name')
# Extract from wdi the rows corresponding to the GDP indicator
GDP_PerYear = wdi_aux[wdi_aux['Indicator Code'] == IndCode_GDP]

# From the newly created Dataframe, extract all the rows and just the columns for the years we need to compute
# Use the pct_change method from Pandas which computes the change in percentage between n periods
# By using the axis = 'columns' and periods=1, the method will compute the percentage change between each
# element and it's immediate left column. In this case, the percentage change between 2010 and 1980
GDPPercentChange = GDP_PerYear.loc[:,['1980','2010']].pct_change(axis = 'columns', periods=1)

# All elements of the column '1980' will be NaN since there is nothing to compare it to.
# To answer the question, we need to look at the maximum value of the '2010' column and get the index
CountryName = GDPPercentChange['2010'].idxmax()

CountryMax = GDPPercentChange['2010'].max() * 100.0

print("10. The country with the highest increase in GDP is {}, with and increase of {:.2f}%.".format(CountryName,CountryMax))
```

```
10. The country with the highest increase in GDP is Equatorial Guinea, with and increase of 32114.68%.
```

In [40]:

```
# Exact same logic will be applied as before
# This will create a new Dataframe in which every element is the percentage of change from
# the previous year.
PctChange  = GDP_PerYear.loc[:,'1980':'2010'].pct_change(periods=1,axis = 'columns')

# Create a rolling window in the previous dataframe moving 4 columns at a time and
# compute the mean of that rolling window
PctChange_avg4years = PctChange.rolling(4,axis = 1).mean()

# Max will give us the maximum pct change per year, the idxmax will give us the year where it happened
YearOfMax = PctChange_avg4years.max().idxmax()

# idxmax will give us the country with the maximum pct_change in each year.
# We have the maximum year already computed, so we just take the value using it
CountryOfMax = PctChange_avg4years.idxmax()[YearOfMax]

ValueMax = PctChange_avg4years.max().max() * 100.0

print("11. The country is {}, with a percentage increase of {:.2f}% in the year {}.".format(CountryOfMax,ValueMax,YearOfMax))
```

```
11. The country is Congo, Dem. Rep., with a percentage increase of 72.13% in the year 2000.
```

In [41]:

```
# From the pwt dataframe, extract all rows for the required years and the country, year and hc columns
# Sort Values by country and year, since it is critical that they are in order for the following operations
# set the index of the dataframe as year and drop the previous index

pwt_aux = (pwt [pwt['year'].between(1980,2010)]
               .loc[:,['country','year','hc']]
               .sort_values(by=['country','year'])
               .set_index(['year'], drop=True))

# From the dataframe, group the data by country and use the pct_change method (with axis=0 and periods=1)
# along with the rolling method and the mean to compute for each country the average percentage increase
# of the past 4 years, for each year
pwt_aux['hc_pctChange_avg4years'] = pwt_aux.groupby('country').hc.pct_change(periods = 1).rolling(4).mean()

# Add the country name to the index of the dataframe
pwt_aux.set_index(['country'], append=True, inplace = True)

# Use idxmax to obtain the index (year, country) of the maximum of the average of percentage increase
# in a rolling window of 4 years
YearOfMax, CountryOfMax =  pwt_aux['hc_pctChange_avg4years'].idxmax()
# Get the max value of the column
ValueMax = pwt_aux['hc_pctChange_avg4years'].max() * 100.0

print("12. The country is {}, with an average increase of {:.2f}% in the year {}.".format(CountryOfMax,ValueMax,YearOfMax))
```

```
12. The country is Botswana, with an average increase of 4.42% in the year 1985.
```