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 [80]:

```
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 [ ]:

```
# importing libraries
import requests, zipfile, io

#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
```

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 [81]:

```
#Using the read_csv method to load the dataset
wdi = pd.read_csv("WDIData.csv")
```

In [82]:

```
#Printing the shape of the dataframe to help on future analysis
wdi.shape
```

Out[82]:

```
(379368, 66)
```

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

In [83]:

```
#Using the head method to see the top 5 rows from the dataframe
wdi.head()
```

Out[83]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964	1965
0	Arab World	ARB	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	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
2	Arab World	ARB	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS	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
4	Arab World	ARB	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS	NaN	NaN	NaN	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 [84]:

```
import urllib
urllib.request.urlretrieve("https://www.rug.nl/ggdc/docs/pwt90.xlsx", "pwt90.xlsx")
```

Out[84]:

```
('pwt90.xlsx', <http.client.HTTPMessage at 0x1921db8eac0>)
```

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 [85]:

```
#Using read_excel to read the excel file. On the parameter sheet_name we specify the sh
eet.
pwt = pd.read_excel("pwt90.xlsx", sheet_name="Data")
```

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

In [86]:

```
pwt.head()
```

Out[86]:

	countrycode	country	currency_unit	year	rgdpe	rgdpo	pop	emp	avh	hc	...	csh
0	ABW	Aruba	Aruban Guilder	1950	NaN	NaN	NaN	NaN	NaN	NaN	...	N
1	ABW	Aruba	Aruban Guilder	1951	NaN	NaN	NaN	NaN	NaN	NaN	...	N
2	ABW	Aruba	Aruban Guilder	1952	NaN	NaN	NaN	NaN	NaN	NaN	...	N
3	ABW	Aruba	Aruban Guilder	1953	NaN	NaN	NaN	NaN	NaN	NaN	...	N
4	ABW	Aruba	Aruban Guilder	1954	NaN	NaN	NaN	NaN	NaN	NaN	...	N

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 [87]:

```
wdi.columns
```

Out[87]:

```
Index(['Country Name', 'Country Code', 'Indicator Name', 'Indicator Code',
       '1960', '1961', '1962', '1963', '1964', '1965', '1966', '1967', '19
68',
       '1969', '1970', '1971', '1972', '1973', '1974', '1975', '1976', '19
77',
       '1978', '1979', '1980', '1981', '1982', '1983', '1984', '1985', '19
86',
       '1987', '1988', '1989', '1990', '1991', '1992', '1993', '1994', '19
95',
       '1996', '1997', '1998', '1999', '2000', '2001', '2002', '2003', '20
04',
       '2005', '2006', '2007', '2008', '2009', '2010', '2011', '2012', '20
13',
       '2014', '2015', '2016', '2017', '2018', '2019', '2020', 'Unnamed: 6
5'],
      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 [88]:

```
#Listing the values on the column
wdi["Country Name"]
```

Out[88]:

```
0         Arab World
1         Arab World
2         Arab World
3         Arab World
4         Arab World
             ...
379363      Zimbabwe
379364      Zimbabwe
379365      Zimbabwe
379366      Zimbabwe
379367      Zimbabwe
Name: Country Name, Length: 379368, dtype: object
```

In [89]:

```
#Dopping the duplicates and listing the unique "Country Name" values
wdi["Country Name"].drop_duplicates()
```

Out[89]:

```
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 [90]:

```
cnames = wdi["Country Name"].drop_duplicates()
```

In [91]:

```
cnames.head(50)
```

Out[91]:

```
0                                            Arab World
1437                              Caribbean small states
2874                        Central Europe and the Baltics
4311                            Early-demographic dividend
5748                                     East Asia & Pacific
7185           East Asia & Pacific (excluding high income)
8622            East Asia & Pacific (IDA & IBRD countries)
10059                                           Euro area
11496                                Europe & Central Asia
12933        Europe & Central Asia (excluding high income)
14370         Europe & Central Asia (IDA & IBRD countries)
15807                                        European Union
17244            Fragile and conflict affected situations
18681              Heavily indebted poor countries (HIPC)
20118                                          High income
21555                                            IBRD only
22992                                    IDA & IBRD total
24429                                           IDA blend
25866                                            IDA only
27303                                           IDA total
28740                            Late-demographic dividend
30177                            Latin America & Caribbean
31614    Latin America & Caribbean (excluding high income)
33051    Latin America & the Caribbean (IDA & IBRD coun...
34488         Least developed countries: UN classification
35925                                  Low & middle income
37362                                           Low income
38799                                  Lower middle income
40236                           Middle East & North Africa
41673    Middle East & North Africa (excluding high inc...
43110    Middle East & North Africa (IDA & IBRD countries)
44547                                        Middle income
45984                                        North America
47421                                       Not classified
48858                                         OECD members
50295                                   Other small states
51732                           Pacific island small states
53169                             Post-demographic dividend
54606                              Pre-demographic dividend
56043                                       Small states
57480                                         South Asia
58917                               South Asia (IDA & IBRD)
60354                                    Sub-Saharan Africa
61791           Sub-Saharan Africa (excluding high income)
63228            Sub-Saharan Africa (IDA & IBRD countries)
64665                                  Upper middle income
66102                                              World
67539                                        Afghanistan
68976                                            Albania
70413                                            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 [92]:

```
#Fist filtering only the 47 first values
cnames = cnames.head(47)
```

In [93]:

```
#Filtering only the rows with countries on the wdi dataframe. Usin the isin method with
not operator (~)
wdi = wdi[~wdi['Country Name'].isin(cnames)]
```

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

In [94]:

```
wdi.head()
```

Out[94]:

	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	1963	1964
67539	Afghanistan	AFG	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	NaN	NaN	NaN	NaN	NaN
67540	Afghanistan	AFG	Access to electricity (% of population)	EG.ELC.ACCS.ZS	NaN	NaN	NaN	NaN	NaN
67541	Afghanistan	AFG	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS	NaN	NaN	NaN	NaN	NaN
67542	Afghanistan	AFG	Access to electricity, urban (% of urban popul...	EG.ELC.ACCS.UR.ZS	NaN	NaN	NaN	NaN	NaN
67543	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS	NaN	NaN	NaN	NaN	NaN

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 [95]:

```
#Reseting the index and use the inplace parameter to change the original dataframe
wdi.reset_index(inplace=True)
```

Show that the indexes have been reseted.

In [96]:

```
wdi.head(30)
```

Out[96]:

	index	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	19
0	67539	Afghanistan	AFG	Access to clean fuels and technologies for coo...	EG.CFT.ACCS.ZS	NaN	NaN	NaN	N
1	67540	Afghanistan	AFG	Access to electricity (% of population)	EG.ELC.ACCS.ZS	NaN	NaN	NaN	N
2	67541	Afghanistan	AFG	Access to electricity, rural (% of rural popul...	EG.ELC.ACCS.RU.ZS	NaN	NaN	NaN	N
3	67542	Afghanistan	AFG	Access to electricity, urban (% of urban popul...	EG.ELC.ACCS.UR.ZS	NaN	NaN	NaN	N
4	67543	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.ZS	NaN	NaN	NaN	N
5	67544	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.FE.ZS	NaN	NaN	NaN	N
6	67545	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.MA.ZS	NaN	NaN	NaN	N
7	67546	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.OL.ZS	NaN	NaN	NaN	N
8	67547	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.40.ZS	NaN	NaN	NaN	N
9	67548	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.PL.ZS	NaN	NaN	NaN	N
10	67549	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.60.ZS	NaN	NaN	NaN	N
11	67550	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.SO.ZS	NaN	NaN	NaN	N
12	67551	Afghanistan	AFG	Account ownership at a financial institution o...	FX.OWN.TOTL.YG.ZS	NaN	NaN	NaN	N

	index	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	19
13	67552	Afghanistan	AFG	Adequacy of social insurance programs (% of to...	per_si_allsi.adq_pop_tot	NaN	NaN	NaN	N
14	67553	Afghanistan	AFG	Adequacy of social protection and labor progra...	per_allsp.adq_pop_tot	NaN	NaN	NaN	N
15	67554	Afghanistan	AFG	Adequacy of social safety net programs (% of t...	per_sa_allsa.adq_pop_tot	NaN	NaN	NaN	N
16	67555	Afghanistan	AFG	Adequacy of unemployment benefits and ALMP (% ...	per_lm_alllm.adq_pop_tot	NaN	NaN	NaN	N
17	67556	Afghanistan	AFG	Adjusted net enrollment rate, primary (% of pr...	SE.PRM.TENR	NaN	NaN	NaN	N
18	67557	Afghanistan	AFG	Adjusted net enrollment rate, primary, female ...	SE.PRM.TENR.FE	NaN	NaN	NaN	N
19	67558	Afghanistan	AFG	Adjusted net enrollment rate, primary, male (%...	SE.PRM.TENR.MA	NaN	NaN	NaN	N
20	67559	Afghanistan	AFG	Adjusted net national income (annual % growth)	NY.ADJ.NNTY.KD.ZG	NaN	NaN	NaN	N
21	67560	Afghanistan	AFG	Adjusted net national income (constant 2010 US$)	NY.ADJ.NNTY.KD	NaN	NaN	NaN	N
22	67561	Afghanistan	AFG	Adjusted net national income (current US$)	NY.ADJ.NNTY.CD	NaN	NaN	NaN	N
23	67562	Afghanistan	AFG	Adjusted net national income per capita (annua...	NY.ADJ.NNTY.PC.KD.ZG	NaN	NaN	NaN	N
24	67563	Afghanistan	AFG	Adjusted net national income per capita (const...	NY.ADJ.NNTY.PC.KD	NaN	NaN	NaN	N
25	67564	Afghanistan	AFG	Adjusted net national income per capita (curre...	NY.ADJ.NNTY.PC.CD	NaN	NaN	NaN	N

	index	Country Name	Country Code	Indicator Name	Indicator Code	1960	1961	1962	19
26	67565	Afghanistan	AFG	Adjusted net savings, excluding particulate em...	NY.ADJ.SVNX.GN.ZS	NaN	NaN	NaN	N
27	67566	Afghanistan	AFG	Adjusted net savings, excluding particulate em...	NY.ADJ.SVNX.CD	NaN	NaN	NaN	N
28	67567	Afghanistan	AFG	Adjusted net savings, including particulate em...	NY.ADJ.SVNG.GN.ZS	NaN	NaN	NaN	N
29	67568	Afghanistan	AFG	Adjusted net savings, including particulate em...	NY.ADJ.SVNG.CD	NaN	NaN	NaN	N

30 rows × 67 columns

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 [97]:

```
#Select the columsn, then use drop_duplicates method and at the end head(5) to get the
 top 5 records
wdi[['Indicator Name', 'Indicator Code']].drop_duplicates().head(5)
```

Out[97]:

	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 [98]:

```
#Select the columsn, then use drop_duplicates method and at the end tail(5) to get the
 bottom 5 records
wdi[['Indicator Name', 'Indicator Code']].drop_duplicates().tail(5)
```

Out[98]:

	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 [99]:

```
#Using the same aproach from the previous questions and at the end the set_index method to define the new Index
indicators = wdi[['Indicator Name', 'Indicator Code']].drop_duplicates().set_index('Indicator Code')
indicators
```

Out[99]:

Indicator Code	Indicator Name
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 [100]:

```
#Funcion that receives a dataframe, a colum name, and a term to search and return the i
ndex of the first ocurrence
def find_index(df,column, term):
    #First search for the term on the dataset. The parameter regex=False avoid python t
o try
    #using a regex patter to make the search.
    result = df[df[column].str.contains(term,regex=False)]
    #If find anything return the index of the first occurrence, if not return None
    if(len(result) > 0):
        return result.index[0]
    else:
        return None
```

In [101]:

```
#1. 'Population', find the 'Indicator Code' of the total population in a country;
population_code = find_index(indicators,'Indicator Name','Population, total')
#2. 'GDP', find the GDP measured in current US Dollars;
gdp_code = find_index(indicators,'Indicator Name','GDP (current US$)')
#3. 'GINI index'
gni_code = find_index(indicators,'Indicator Name','Gini index')
```

In [102]:

```
print(population_code)
print(gdp_code)
print(gni_code)
```

```
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 [103]:

```
wdi_sample = wdi[['Indicator Code', 'Country Code', '2012']]
```

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 [104]:

```
#Create a list with the indicators code an then use the isin method to select rows with
the codes from the list
indicators_code = [population_code,gdp_code,gni_code]
wdi_sample = wdi_sample.loc[wdi_sample['Indicator Code'].isin(indicators_code)]
```

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 [105]:

```
wdi_sample = pd.pivot_table(wdi_sample,values="2012",index="Country Code", columns=["In
dicator Code"])
```

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

In [106]:

```
wdi_sample.rename(columns={population_code: "Population", gdp_code: "GDP", gni_code: "G
INI"}, inplace=True)
```

In [107]:

```
wdi_sample
```

Out[107]:

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 [108]:

```
pwt_sample = pwt[pwt['year'] == 2012][['countrycode', 'hc']]
pwt_sample.rename(columns= {'countrycode': 'Country Code'},inplace=True)
pwt_sample.head()
```

Out[108]:

	Country Code	hc
62	ABW	NaN
127	AGO	1.431295
192	AIA	NaN
257	ALB	2.917346
322	ARE	2.723864

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 [109]:

```
#Setting the Country Code as the index on teh pwt_sample dataframe
pwt_sample = pwt_sample.set_index('Country Code')
```

In [110]:

```
#Concatenating the two dataframes
data = pd.concat([pwt_sample,wdi_sample], axis=1)
```

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 [111]:

```
print("Question 1")
countries = data[(data['Population'] >= 9000000) & (data['Population'] <= 11000000)].in
dex
print(list(countries))
```

```
Question 1
['ARE', 'AZE', 'BDI', 'BEN', 'BLR', 'BOL', 'CZE', 'DOM', 'GIN', 'HTI', 'HU
N', 'PRT', 'RWA', 'SWE', 'TUN', 'SSD']
```

In [112]:

```
print("Question 2")
std = data.loc[countries]['GDP'].std()
print("The standard deviation is ",std,", and the average is ",data.loc[countries]['GDP'].mean())
```

```
Question 2
The standard deviation is  156266663181.872 , and the average is  112227662509.42097
```

In [113]:

```
print("Question 3")
#Creating a new column that 'GDP Per Capita' by dividing the GDP per the Population
data['GDP Per Capita'] = data['GDP'] / data['Population']
#Then use the funcions sdt() and mean() to calculate both standard deviation and the average
mean = data['GDP Per Capita'].mean()
print("The average of the GDP per capita is: ", mean, ", and the std is ", data['GDP Per Capita'].std())
```

```
Question 3
The average of the GDP per capita is:  17172.39989920186 , and the std is 24961.890982797377
```

1. 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?

In [114]:

```
#Create a function the receives a row from a dataframe and then look for the column Pop
ulation.
# Return the classification of the row according with the Population size
def classify_country(row):
    population = row['Population']
    if population < 1000000:
        return "Tiny"
    elif population < 5000000:
        return "Very Small"
    elif population < 15000000:
        return "Small"
    elif population < 30000000:
        return "Medium"
    elif population < 100000000:
        return "Large"
    else:
        return "Huge"

#Call the apply function on the dataframe passing the previous function to create a new
column with
#the population size classification
data['Population Classification'] = data.apply(classify_country,axis=1)

print("Question 4")
#For each classification print the mean an the std
classifications = ["Tiny", "Very Small","Small","Medium","Large","Huge"]
for c in classifications:
    c_data = data[data['Population Classification'] == c]
    mean = c_data['GDP Per Capita'].mean()
    print("Classification: ", c, " - Average: ", mean, " - Sdt: ", c_data['GDP Per Capi
ta'].std())
```

```
Question 4
Classification:  Tiny  - Average:  27360.25818746288  - Sdt:  35964.116036
50782
Classification:  Very Small  - Average:  13828.15087331703  - Sdt:  16991.
276306225987
Classification:  Small  - Average:  16293.521093060706  - Sdt:  23499.3592
91061017
Classification:  Medium  - Average:  9840.172205668905  - Sdt:  16578.3879
84082838
Classification:  Large  - Average:  12752.302640204414  - Sdt:  15771.2370
1816868
Classification:  Huge  - Average:  14048.187428541969  - Sdt:  18482.95668
9989445
```

In [115]:

```
print("Question 5")
#Using teh apply function to identify if the column is above the mean or not
def classify_column_above_mean(df,column):
    mean = df[column].mean()
    return df.apply(lambda row: None if row[column] is None else (1 if row[column] > me
an else 0),axis=1)
print("Testing the function")
classify_column_above_mean(data,'Population')
```

```
Question 5
Testing the function
```

Out[115]:

```
ABW    0
AGO    0
AIA    0
ALB    0
ARE    0
      ..
TUV    0
VIR    0
VUT    0
WSM    0
XKX    0
Length: 219, dtype: int64
```

In [116]:

```
print("Question 6")
#Using the previous function to create the new column, and the use the groupby function
and aggregate by the mean
data['HC_Above_Mean'] = classify_column_above_mean(data,'hc')
data.groupby(['Population Classification','HC_Above_Mean'])['GDP Per Capita'].mean()
```

```
Question 6
```

Out[116]:

```
Population Classification  HC_Above_Mean
Huge                       0                2713.892188
                           1               27649.341717
Large                      0                3434.865808
                           1               24219.917202
Medium                     0                1322.910398
                           1               19906.027069
Small                      0                3860.592297
                           1               33462.803716
Tiny                       0               24916.712161
                           1               40494.318080
Very Small                 0                9379.995570
                           1               18498.713941
Name: GDP Per Capita, dtype: float64
```

In [117]:

```
print("Question 7")
#Using the previous function to create the new column, and the use the groupby function
and aggregate by the mean
data['GINI_Above_Mean'] = classify_column_above_mean(data,'GINI')
data.groupby(['GINI_Above_Mean','HC_Above_Mean'])['GDP Per Capita'].mean()
```

```
Question 7
```

Out[117]:

```
GINI_Above_Mean  HC_Above_Mean
0                0                12742.936744
                 1                30774.334264
1                0                 4200.770667
                 1                10044.431693
Name: GDP Per Capita, dtype: float64
```

In [119]:

```
print("Question 8")
#First select the Max value of the GDP per Capita column
max_gdp_per_capita = data[(data['GINI_Above_Mean']) == 0 & (data['HC_Above_Mean'] == 0
)]['GDP Per Capita'].max()
#Then use this value to select the country code associated with this value
country_code = data[data['GDP Per Capita'] == max_gdp_per_capita].index[0]
#The use the code to select the name of the Country
wdi[wdi['Country Code'] == country_code]['Country Name'].reset_index().loc[0:0]['Countr
y Name']
```

```
Question 8
```

Out[119]:

```
0    Monaco
Name: Country Name, dtype: object
```

In [120]:

```
print("Question 9")
#For each classification print the mean an the std
classifications = ["Tiny", "Very Small","Small","Medium","Large","Huge"]
for c in classifications:
    index = data[(data['GINI_Above_Mean'] == 0) & (data['HC_Above_Mean'] == 0) & (data[
'Population Classification'] == c)]
    index =  index['GDP Per Capita'].idxmax()
    country_name = wdi[wdi['Country Code'] == index]['Country Name'].values[0]
    print("Classification:", c, " - Country:", country_name)
```

```
Question 9
Classification: Tiny  - Country: Monaco
Classification: Very Small  - Country: Kuwait
Classification: Small  - Country: Portugal
Classification: Medium  - Country: Angola
Classification: Large  - Country: Iran, Islamic Rep.
Classification: Huge  - Country: Nigeria
```

In [122]:

```
print("Question 10")
#Creating a list with Country Code, Country Name and with the years
columns = ['Country Code', 'Country Name']
columns = columns + [str(i) for i in range(1980,2011)]
#Use the list to select the data from the dataframe
wdi_pct = wdi[wdi['Indicator Code'] == gdp_code][columns]
#Using the melt method to unpivot the columns in rows
wdi_pct = wdi_pct.melt(id_vars=['Country Code', 'Country Name'], var_name='Year', value
_name='GDP')
#Changing the index from the dataframe
wdi_pct = wdi_pct.set_index(['Country Code', 'Country Name','Year'])
#Using the pct_change function to calculate the % and then the groupby function to aggr
egate by the max value
wdi_pct = wdi_pct.groupby(level=[0,1]).pct_change().groupby(level=[1]).max()
#Then use the idxmax to find the country with the largest GDP
print("The Country with largest % increase between 1980 and 2010 was ", wdi_pct.loc[wdi
_pct.idxmax()].index.values[0])
```

```
Question 10
The Country with largest % increase between 1980 and 2010 was  Congo, Dem.
Rep.
```

In [123]:

```
print("Question 11")
#Using the same aproach of the last questions, but instead of use the max to aggregate
 the values on
#the group by, I used the mean(), so we will have the mean GDP. And on the pct_change u
se the periods parameter with the value
#3 to look for the increase on the past years
columns = ['Country Code', 'Country Name']
columns = columns + [str(i) for i in range(1980,2011)]
wdi_pct = wdi[wdi['Indicator Code'] == gdp_code][columns]
wdi_pct = wdi_pct.melt(id_vars=['Country Code', 'Country Name'], var_name='Year', value
_name='GDP')
wdi_pct = wdi_pct.set_index(['Country Code', 'Country Name','Year'])
wdi_pct = wdi_pct.groupby(level=[0,1]).pct_change(periods=3).groupby(level=[1,2]).mean
()
country_name = wdi_pct.loc[wdi_pct.idxmax()].index.values[0]
print("The Country with largest average % increase between 1980 and 2010 was ", country
_name)
```

```
Question 11
The Country with largest average % increase between 1980 and 2010 was  ('E
quatorial Guinea', '2005')
```

In [124]:

```
print("Question 12")
#Filter the rows from the years from 1980 to 2010
pwt_pct = pwt[(pwt['year'] >= 1980) & (pwt['year'] <= 2010)][['countrycode', 'country',
'year','hc']]
#Change the indexes
pwt_pct = pwt_pct.set_index(['countrycode', 'country','year'])
#Combine the pct_change and the groupby function
pwt_pct = pwt_pct.groupby(level=[0,1]).pct_change(periods=3).groupby(level=[1,2]).mean
()
#Use the idxmax to selct the rows with the max values
country_name = pwt_pct.loc[pwt_pct.idxmax()].index.values[0]
print("The Country with largest HC average % increase between 1980 and 2010 was ", coun
try_name)
```

```
Question 12
The Country with largest HC average % increase between 1980 and 2010 was
('Botswana', 1985)
```

In [ ]:

```

```