Project Final (2)

October 19, 2022

1 Final Project

by Mohammad Fanous

This notebook serves as my final project for this class.

As a start, I downloaded a public data set off Kaggle, that records the urban population values for countries in the world between 1960 and 2020. I will read it in shortly.

Through working on this data set, I am interested in learning about the population changes for different countries, especially war-torn countries and countries in crises. I hope to answer the following question: Would there be a relationship between the unusal urban population trends and the ongoing national and international events? Short answer should be yes, but let's test that and learn something new!

I believe this question is important as while getting to an answer, we would certainly learn something new about a country's history or even its present situation. And in general, population analysis could be crucial for understanding a certain population's style of living, history, or possibly traditions. It is also important when dealing with overpopulation issues.

2 First Element: Reading in the Dataset

Now let's get started! First, I'll import stuff we might need and read in the dataset (will call the resulting pandas table, "pop") in the two cells below:

[1]:
```
import matplotlib.pyplot as plt #importing the libraries we would need in this␣
↪notebook
import pandas as pd
import numpy as np
```

[2]:
```
pop=pd.read_csv('urban_total.csv') #reading in the csv file urban_total.csv as␣
↪a pandas table and calling it pop
```

Nice! Now, let's take a look at pop:

[3]:
```
pop #printing the pandas table pop
```

[3]:
```
                    Country Name Country Code    Indicator Name  \
0                          Aruba          ABW  Urban population
```

```
1    Africa Eastern and Southern          AFE  Urban population
2                    Afghanistan          AFG  Urban population
3     Africa Western and Central          AFW  Urban population
4                         Angola          AGO  Urban population
..                           …          …                 …
261                       Kosovo          XKX  Urban population
262                    Yemen Rep.          YEM  Urban population
263                  South Africa          ZAF  Urban population
264                       Zambia          ZMB  Urban population
265                     Zimbabwe          ZWE  Urban population

    Indicator Code        1960        1961        1962        1963  \
0      SP.URB.TOTL     27525.0     28139.0     28537.0     28763.0
1      SP.URB.TOTL  19239140.0  20049454.0  20897622.0  21807831.0
2      SP.URB.TOTL    755835.0    796271.0    839385.0    885227.0
3      SP.URB.TOTL  14141671.0  14813809.0  15527606.0  16290977.0
4      SP.URB.TOTL    569223.0    597286.0    628376.0    660175.0
..             …           …           …           …           …
261    SP.URB.TOTL         NaN         NaN         NaN         NaN
262    SP.URB.TOTL    483697.0    510127.0    538117.0    567679.0
263    SP.URB.TOTL   7971773.0   8200255.0   8427007.0   8662568.0
264    SP.URB.TOTL    557193.0    599672.0    645119.0    695944.0
265    SP.URB.TOTL    476164.0    500665.0    528409.0    567387.0

           1964        1965  …         2012         2013         2014  \
0       28922.0     29080.0  …      44059.0      44351.0      44666.0
1    22780115.0  23806156.0  …  181061819.0  188513993.0  196270604.0
2      934134.0    986074.0  …    7528589.0    7865068.0    8204880.0
3    17102150.0  17961784.0  …  158838089.0  165607296.0  172602709.0
4      691526.0    721552.0  …   15383123.0   16130308.0   16900844.0
..           …           …  …           …            …            …
261         NaN         NaN  …         NaN          NaN          NaN
262    598799.0    631541.0  …    8065869.0    8439119.0    8822595.0
263   8906583.0   9158948.0  …   33428280.0   34248628.0   35078456.0
264    762426.0    834489.0  …    5837266.0    6099735.0    6372742.0
265    609177.0    653686.0  …    4306228.0    4359432.0    4416224.0

           2015         2016         2017         2018         2019  \
0       44978.0      45293.0      45614.0      45949.0      46294.0
1    204322110.0  212669776.0  221319466.0  230276235.0  239539517.0
2      8535606.0    8852834.0    9164768.0    9476982.0    9797274.0
3    179836016.0  187305752.0  195016097.0  202961117.0  211133789.0
4     17691524.0   18502164.0   19332895.0   20184724.0   21061028.0
..           …            …            …            …            …
261         NaN          NaN          NaN          NaN          NaN
262   9215168.0    9615916.0   10024986.0   10442487.0   10869523.0
263  35905875.0   36726640.0   37540921.0   38348227.0   39149715.0
```

```
264    6654568.0    6944320.0    7243007.0    7551639.0    7871715.0
265    4473872.0    4531238.0    4589452.0    4650597.0    4717307.0

            2020  Unnamed: 65
0        46654.0          NaN
1    249112475.0          NaN
2     10131490.0          NaN
3    219531155.0          NaN
4     21962884.0          NaN
..           …          …
261          NaN          NaN
262   11306428.0          NaN
263   39946775.0          NaN
264    8204576.0          NaN
265    4792105.0          NaN

[266 rows x 66 columns]
```

3 Second Element: Working on a Subset of the Data

Here I will take subsets of the data (create smaller tables for specific countries) that I will probably use later on in the notebook. Before I do that, however, I would have to fix the table a bit, because it is hard to work with it as it is right now. I wonder if there's a way to list all the years in one "Year" column instead of in separate columns (and of course their population values in a single "Population" column). After some research, it looks like we can fix it using the code below:

[4]:
```
'''Fixing the columns of the table pop'''
newpop=pop.melt(id_vars=["Country Name", "Country Code",'Indicator␣
 ↪Name','Indicator Code'],
        var_name="Year",
        value_name="Population") #found and used a method that fixes the table␣
 ↪columns, called "melt"
newpop #print out the new fixed (thinner) table
```

[4]:
```
                    Country Name Country Code    Indicator Name  \
0                          Aruba          ABW  Urban population
1      Africa Eastern and Southern          AFE  Urban population
2                      Afghanistan          AFG  Urban population
3       Africa Western and Central          AFW  Urban population
4                           Angola          AGO  Urban population
…                              …          …                …
16487                       Kosovo          XKX  Urban population
16488                  Yemen Rep.          YEM  Urban population
16489                South Africa          ZAF  Urban population
16490                      Zambia          ZMB  Urban population
16491                    Zimbabwe          ZWE  Urban population
```

```
      Indicator Code          Year  Population
0        SP.URB.TOTL          1960     27525.0
1        SP.URB.TOTL          1960  19239140.0
2        SP.URB.TOTL          1960    755835.0
3        SP.URB.TOTL          1960  14141671.0
4        SP.URB.TOTL          1960    569223.0
…                …           …          …
16487    SP.URB.TOTL  Unnamed: 65         NaN
16488    SP.URB.TOTL  Unnamed: 65         NaN
16489    SP.URB.TOTL  Unnamed: 65         NaN
16490    SP.URB.TOTL  Unnamed: 65         NaN
16491    SP.URB.TOTL  Unnamed: 65         NaN

[16492 rows x 6 columns]
```

Nice!! That new table "newpop" looks much better! But have you noticed those annoying "NaN"s by the end. I'll drop them for my peace of mind (and of course to avoid any issues beforehand):

[5]:
```
newpop=newpop.dropna() #drop the NaN's because they're annoying
newpop #printing the new table without the NaN's
```

[5]:
```
                      Country Name Country Code    Indicator Name  \
0                            Aruba          ABW  Urban population
1      Africa Eastern and Southern          AFE  Urban population
2                      Afghanistan          AFG  Urban population
3       Africa Western and Central          AFW  Urban population
4                           Angola          AGO  Urban population
…                              …           …                 …
16220                        Samoa          WSM  Urban population
16222                  Yemen Rep.           YEM  Urban population
16223                South Africa          ZAF  Urban population
16224                      Zambia          ZMB  Urban population
16225                    Zimbabwe          ZWE  Urban population

      Indicator Code  Year  Population
0        SP.URB.TOTL  1960     27525.0
1        SP.URB.TOTL  1960  19239140.0
2        SP.URB.TOTL  1960    755835.0
3        SP.URB.TOTL  1960  14141671.0
4        SP.URB.TOTL  1960    569223.0
…                …    …          …
16220    SP.URB.TOTL  2020     35494.0
16222    SP.URB.TOTL  2020  11306428.0
16223    SP.URB.TOTL  2020  39946775.0
16224    SP.URB.TOTL  2020   8204576.0
16225    SP.URB.TOTL  2020   4792105.0
```

```
[16001 rows x 6 columns]
```

Perfect! Now we can easily take subsets of the data:

For now, I'll create three subset tables: for Lebanon, Syria, and Ukraine and call them: popLB, popSA, and popUE respectively:

[6]:
```
'''Creating a smaller table with only Lebanon's data'''
popLB=newpop[newpop['Country Name'].str.contains("Lebanon")] #create the first␣
 ↪subset table for data about Lebanon and call it popLB
popLB #print this table
```

[6]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
130        Lebanon          LBN  Urban population    SP.URB.TOTL  1960
396        Lebanon          LBN  Urban population    SP.URB.TOTL  1961
662        Lebanon          LBN  Urban population    SP.URB.TOTL  1962
928        Lebanon          LBN  Urban population    SP.URB.TOTL  1963
1194       Lebanon          LBN  Urban population    SP.URB.TOTL  1964
…              …            …               …              …     …
15026      Lebanon          LBN  Urban population    SP.URB.TOTL  2016
15292      Lebanon          LBN  Urban population    SP.URB.TOTL  2017
15558      Lebanon          LBN  Urban population    SP.URB.TOTL  2018
15824      Lebanon          LBN  Urban population    SP.URB.TOTL  2019
16090      Lebanon          LBN  Urban population    SP.URB.TOTL  2020

       Population
130      764264.0
396      821157.0
662      880861.0
928      942378.0
1194    1004429.0
…             …
15026   5926427.0
15292   6030303.0
15558   6076955.0
15824   6084990.0
16090   6069524.0

[61 rows x 6 columns]
```

[7]:
```
'''Creating a smaller table with only Syria's data'''
popSA=newpop[newpop['Country Name'].str.contains("Syria")] #create the first␣
 ↪subset table for data about Lebanon and call it popSA
popSA #print this table
```

[7]:
```
                Country Name Country Code    Indicator Name Indicator Code  \
227    Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
493    Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
759    Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
1025   Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
1291   Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
…                       …          …               …              …
15123  Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
15389  Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
15655  Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
15921  Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL
16187  Syrian Arab Republic          SYR  Urban population    SP.URB.TOTL

       Year  Population
227    1960   1683373.0
493    1961   1765941.0
759    1962   1854808.0
1025   1963   1948103.0
1291   1964   2046282.0
…      …         …
15123  2016   9227932.0
15389  2017   9146183.0
15655  2018   9177784.0
15921  2019   9358017.0
16187  2020   9708489.0

[61 rows x 6 columns]
```

[8]:
```
'''Creating a smaller table with only Ukraine's data'''
popUE=newpop[newpop['Country Name'].str.contains("Ukraine")] #create the first␣
 ↪subset table for data about Lebanon and call it popUE
popUE #print this smaller table
```

[8]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
248        Ukraine          UKR  Urban population    SP.URB.TOTL  1960
514        Ukraine          UKR  Urban population    SP.URB.TOTL  1961
780        Ukraine          UKR  Urban population    SP.URB.TOTL  1962
1046       Ukraine          UKR  Urban population    SP.URB.TOTL  1963
1312       Ukraine          UKR  Urban population    SP.URB.TOTL  1964
…            …          …               …              …    …
15144      Ukraine          UKR  Urban population    SP.URB.TOTL  2016
15410      Ukraine          UKR  Urban population    SP.URB.TOTL  2017
15676      Ukraine          UKR  Urban population    SP.URB.TOTL  2018
15942      Ukraine          UKR  Urban population    SP.URB.TOTL  2019
16208      Ukraine          UKR  Urban population    SP.URB.TOTL  2020

       Population
```

```
248     19963641.0
514     20541164.0
780     21129701.0
1046    21721790.0
1312    22308886.0
…              …
15144   31122532.0
15410   31043768.0
15676   30946609.0
15942   30836427.0
16208   30721277.0

[61 rows x 6 columns]
```

Great, these 3 tables look good!

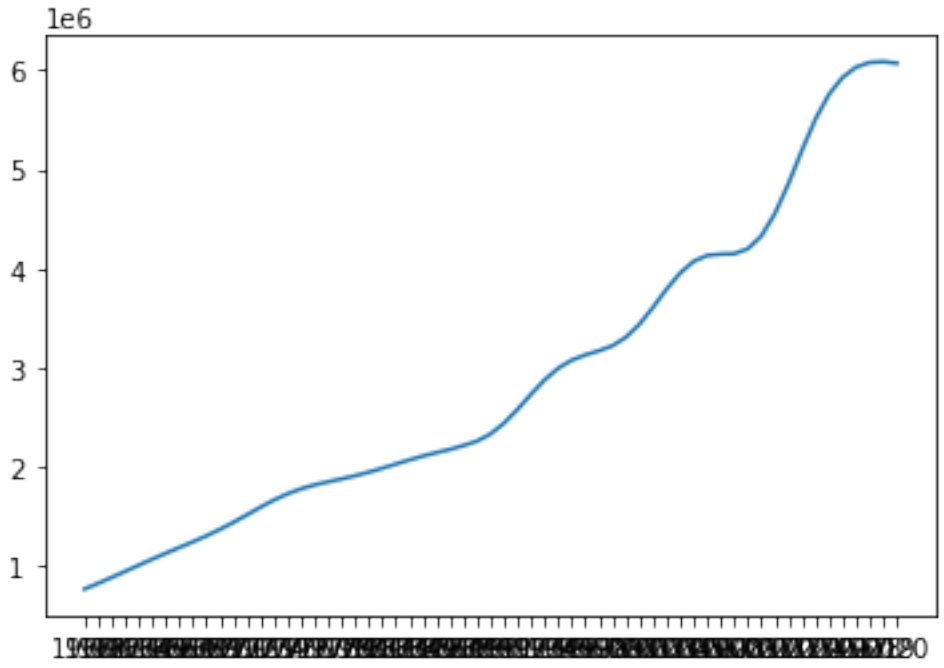
4 Third Element: Plotting One Parameter Vs Another

Now, let's see how the plots will look like for these 3 countries:

For Lebanon:

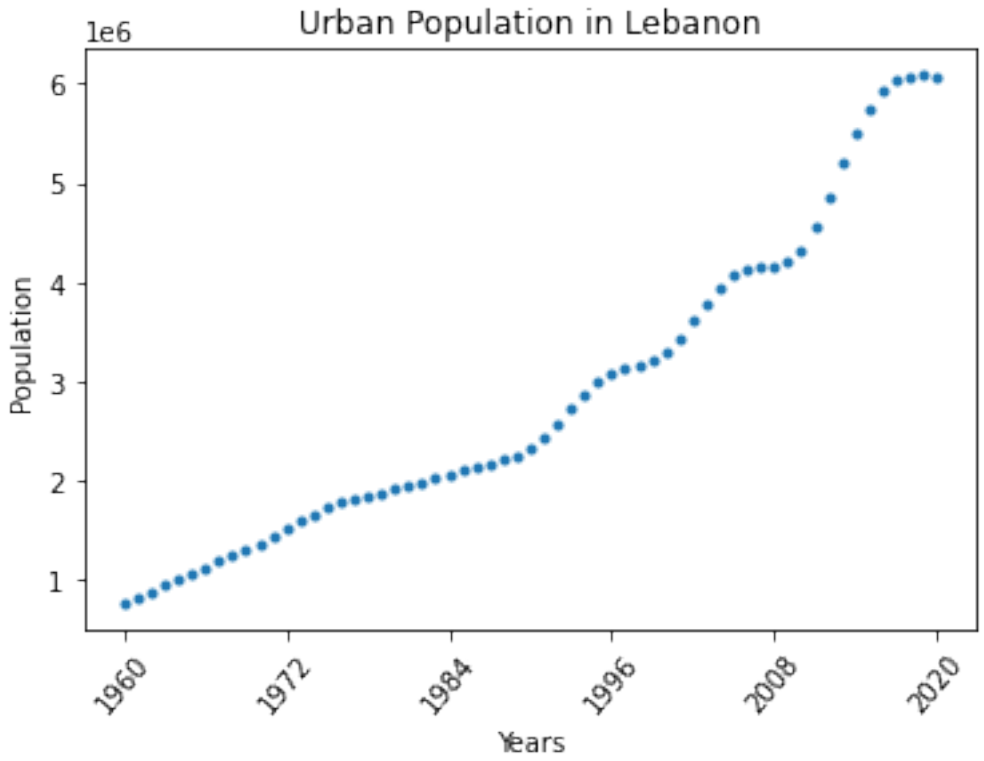
[9]:
```
plt.plot(popLB.Year,popLB.Population) #plotting the urban population vs years
↪(1960-2020) for Lebanon
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7f5874fca910>]
```



Oh, the terribly-looking x-axis again… but that makes sense because I have 60 years to be displayed on a 4 cm x-axis… Now let's use Andrea's method of fixing it in class 11: simply drop the NaN's which I already did, and use lin-space to create, for example, 6 equally-spaced years:

[10]:
```
'''Fixing the x-axis for Lebanon's plot'''
popLB.index=popLB.index-min(popLB.index) #getting the index to start from zero
desind=np.linspace(min(popLB.index),max(popLB.index),6,dtype=int) #creating 6␣
 ↪equally spaced indices
plt.plot(popLB['Year'],popLB['Population'],'.') #plotting the plot urban pop vs␣
 ↪years for Lebanon
plt.xticks(popLB.Year[desind],rotation=50) #using xticks to print 6 equally␣
 ↪spaced dates at the indices generated above
plt.xlabel('Years') #label x-axis
plt.ylabel('Population') #label y-axis
plt.title('Urban Population in Lebanon'); #add title
```

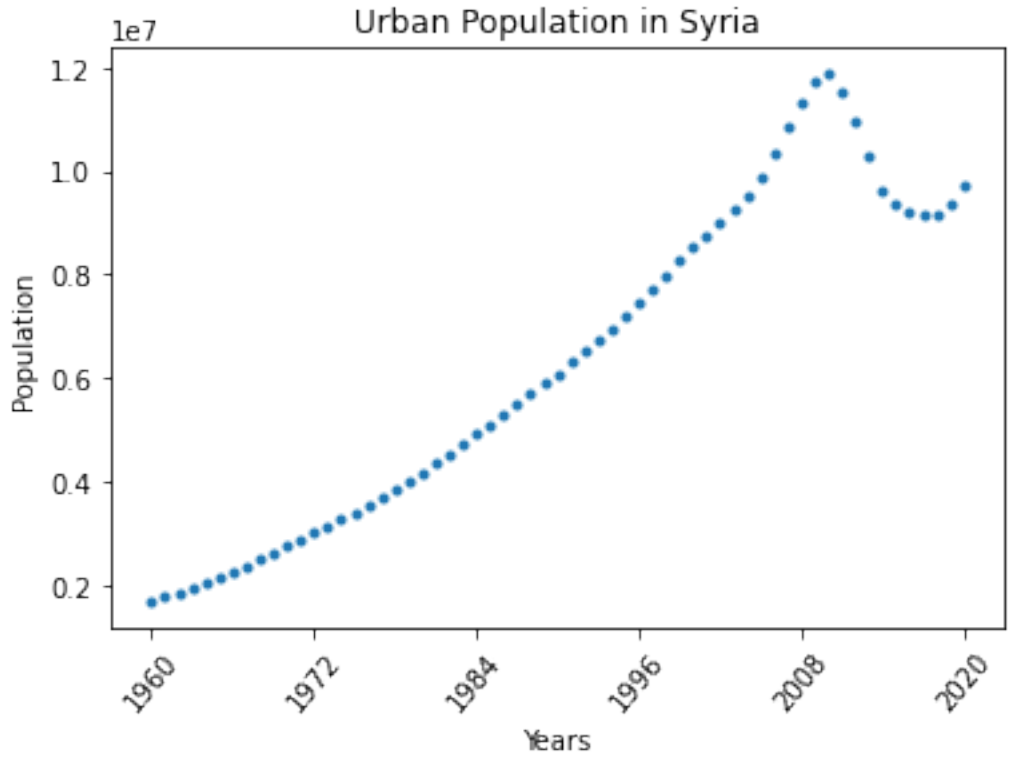


It worked! That looks much clearer. I'll do the same thing for the remaining plots.

For Syria:

[11]:
```
'''Fixing the x-axis for Syria's plot'''
popSA.index=popSA.index-min(popSA.index) #getting the index to start from zero
```

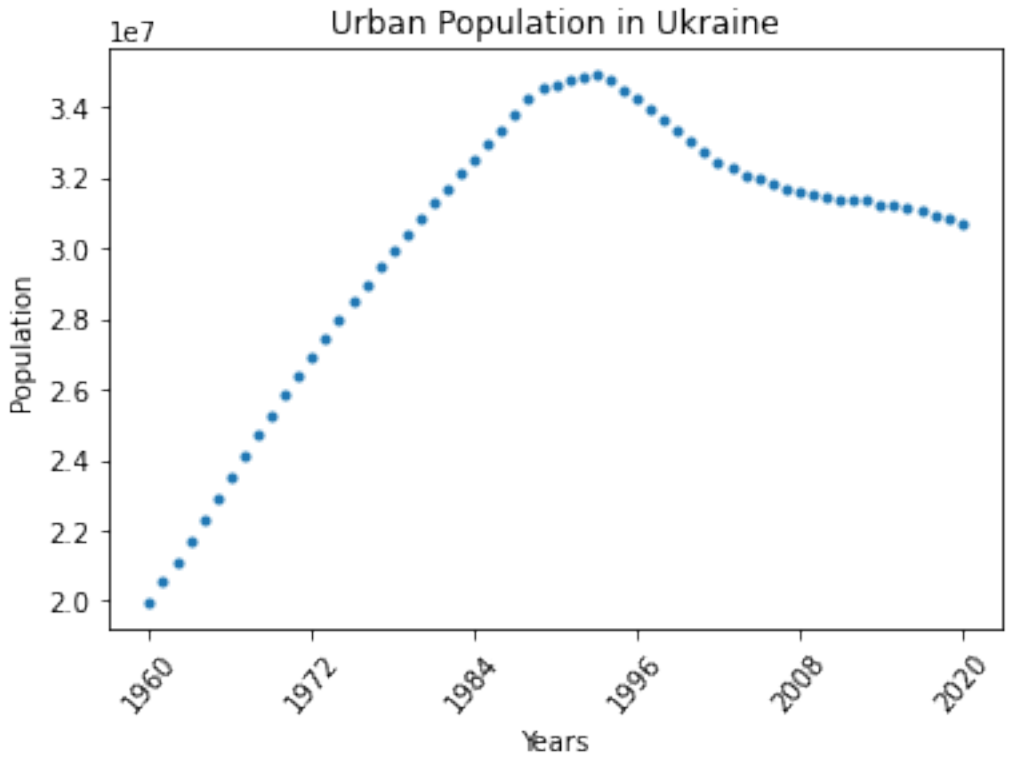
```
desind=np.linspace(min(popSA.index),max(popSA.index),6,dtype=int) #creating 6␣
 ↪equally spaced indices
plt.plot(popSA['Year'],popSA['Population'],'.') #plotting the plot urban pop vs␣
 ↪years for Syria
plt.xticks(popSA.Year[desind],rotation=50) #using xticks to print 6 equally␣
 ↪spaced dates at the indices generated above
plt.xlabel('Years') #label x-axis
plt.ylabel('Population') #label y-axis
plt.title('Urban Population in Syria'); #add title
```



Great, now for Ukraine:

[12]:
```
'''Fixing the x-axis for Ukraine's plot'''
popUE.index=popUE.index-min(popUE.index) #getting the index to start from zero
desind=np.linspace(min(popUE.index),max(popUE.index),6,dtype=int) #creating 6␣
 ↪equally spaced indices
plt.plot(popUE['Year'],popUE['Population'],'.') #plotting the plot urban pop vs␣
 ↪years for Ukraine
plt.xticks(popUE.Year[desind],rotation=50) #using xticks to print 6 equally␣
 ↪spaced dates at the indices generated above
plt.xlabel('Years') #label x-axis
plt.ylabel('Population') #label y-axis
```

```
plt.title('Urban Population in Ukraine'); #add title
```



5 Fourth Element : class and method to fix the x-axis

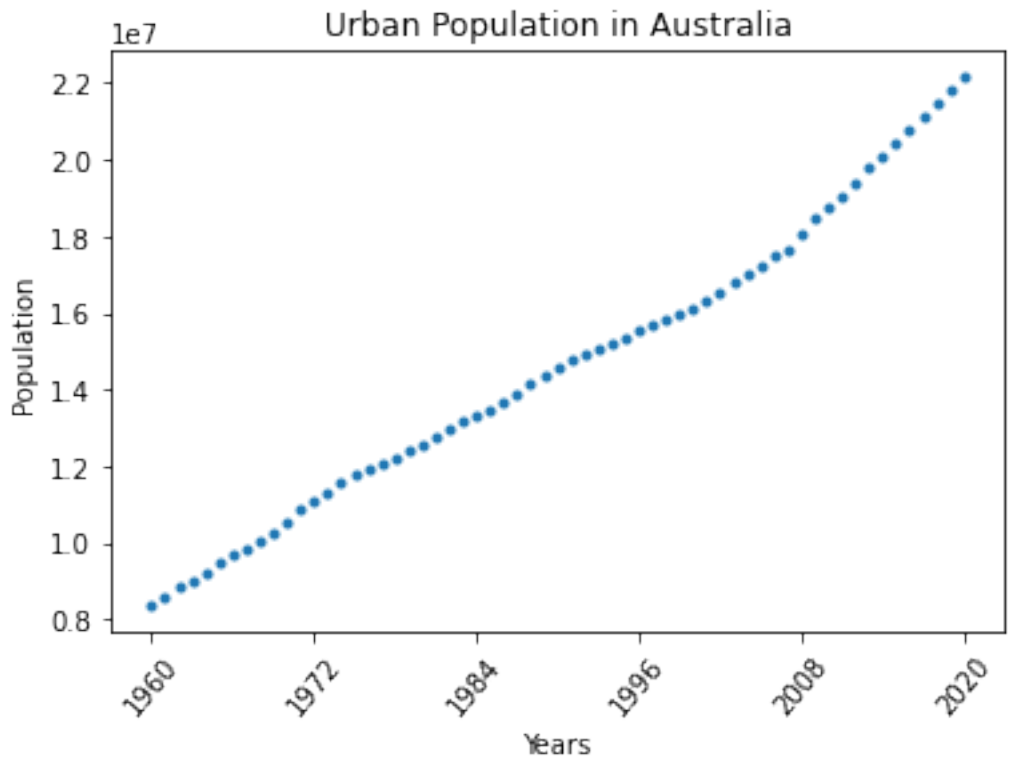
While plotting the 3 plots above, I noticed that I was merely changing the subtable's name each time. So, I thought we could make that more general by creating a class.
The class "fixplot" below contains a method called "fixing" that is supposed to take a specific country name, create a subset table for that country and plot its population vs the years with a good-looking x-axis for sure (including appropriate labels and title):

[13]:
```
'''Create a class that accept a specific country name, create a subset with the␣
↪country's data, plots its population vs time, and of course fixes the␣
↪x-axis'''
class fixplot: #defining the class "fixplot"
    def __init__(self, xvalues, yvalues):  #the default method that must be␣
↪present "__init__"
        self.xdata = xvalues
        self.ydata = yvalues
    def fixing(CountryName): #define the method fixing that does all that's␣
↪mentioned in the comment block above
```

```
        pop1=newpop[newpop['Country Name'].str.contains(CountryName)] #creating␣
↪a subset of the data (for a specific country)
        pop1.index=pop1.index-min(pop1.index) #starting the index from zero
        desind=np.linspace(min(pop1.index),max(pop1.index),6,dtype=int)␣
↪#creating 6 equally spaced indices
        plt.plot(pop1['Year'],pop1['Population'],'.') #plotting the plot urban␣
↪pop vs years for the chosen country
        plt.xticks(pop1.Year[desind],rotation=50) #using xticks to print 6␣
↪equally spaced dates at the indices generated above
        plt.xlabel('Years') #label x-axis
        plt.ylabel('Population') #label y-axis
        plt.title('Urban Population in %s' %CountryName); #add a title to the␣
↪plot
```

Great, now let's see if it works. I'll try it on Australia to check:

[14]:
```
fixplot.fixing('Australia') #testing the method "fixing" in the class above,␣
↪turns out it works!
```



It does!

6 Fifth Element: Correlation Analysis

In this section, I want to test if there's a correlation between the urban populations in Lebanon and Syria between 2011 and 2014. I'll tell why later on in the conclusions section.

First, let's make another smaller subset of the 2010-2015 population values for each country:

[15]:
```
popLBs=popLB['Population'].loc[(popLB.Year>'2010')&(popLB.Year<'2015')]␣
 ↪#finding Lebanon's population data between 2011 and 2014
popLBs #printing the result
```

[15]:
```
13566    4550937.0
13832    4852949.0
14098    5191037.0
14364    5506402.0
Name: Population, dtype: float64
```

[16]:
```
popSAs=popSA['Population'].loc[(popSA.Year>'2010')&(popSA.Year<'2015')]␣
 ↪#finding Syria's population data between 2011 and 2014
popSAs #printing the result
```

[16]:
```
13566    11506454.0
13832    10946032.0
14098    10284372.0
14364     9636016.0
Name: Population, dtype: float64
```

[17]:
```
popSAs.corr(popLBs, method='pearson') #finding the correlation between the two.
```

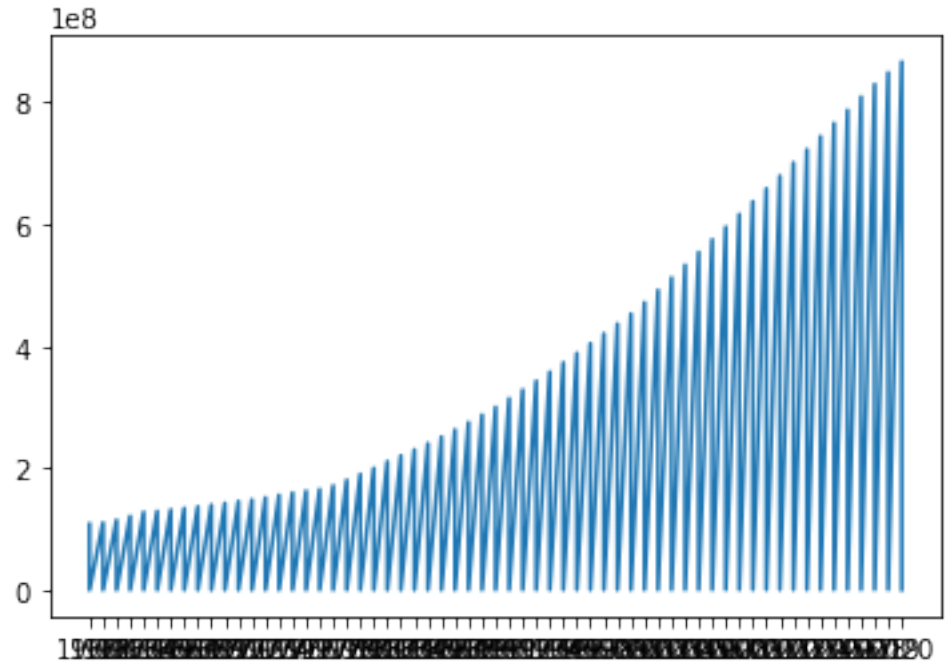
[17]:
```
-0.9997577037470852
```

As I expected, there's a strong negative correlation between the populations of the two countries and if we go on into later years this negative correlation decreases, which makes sense to me. I will tell you about all I'm thinking in the conclusions section.

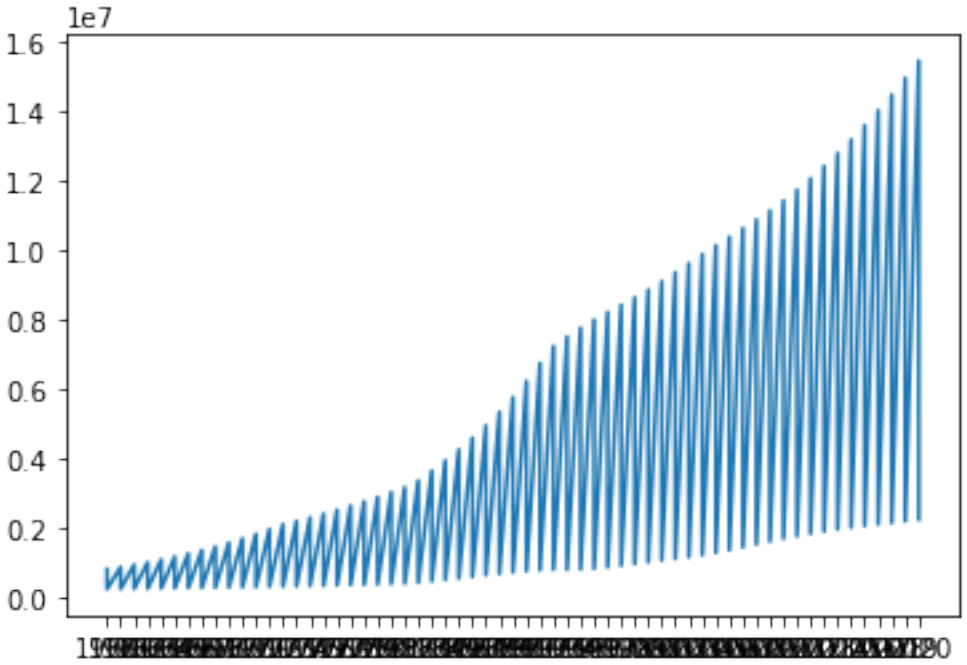
7 Surprising Result + Class fix

While I was plotting the populations for China and Sudan, I had a weird-looking odd graph. My code was something like this:

[18]:
```
popCA=newpop[newpop['Country Name'].str.contains('China')] #smaller table for␣
 ↪China's data called popCA
plt.plot(popCA.Year,popCA.Population); #plot China's population data vs the␣
 ↪years
```



[19]:
```
popSN=newpop[newpop['Country Name'].str.contains('Sudan')] #smaller table for␣
 ↪Sudan's data called popSN
plt.plot(popSN.Year,popSN.Population); #plot Sudan's population data vs the␣
 ↪years
```



I was very sure there was an error somewhere in my code because, you know, urban population does not work like that i.e., change drastically from one year to another. So, first thing I thought about was looking at what the tables for both countries look like:

[20]:
```
popCA
```

[20]:
```
              Country Name Country Code    Indicator Name Indicator Code  \
40                   China          CHN  Urban population    SP.URB.TOTL
96      Hong Kong SAR China          HKG  Urban population    SP.URB.TOTL
146         Macao SAR China          MAC  Urban population    SP.URB.TOTL
306                  China          CHN  Urban population    SP.URB.TOTL
362     Hong Kong SAR China          HKG  Urban population    SP.URB.TOTL
…                    …          …               …             …
15790   Hong Kong SAR China          HKG  Urban population    SP.URB.TOTL
15840       Macao SAR China          MAC  Urban population    SP.URB.TOTL
16000                China          CHN  Urban population    SP.URB.TOTL
16056   Hong Kong SAR China          HKG  Urban population    SP.URB.TOTL
16106       Macao SAR China          MAC  Urban population    SP.URB.TOTL

       Year   Population
40     1960  108085352.0
96     1960    2620415.0
146    1960     159890.0
306    1961  110327936.0
362    1961    2702168.0
…      …           …
15790  2019    7507400.0
15840  2019     640446.0
16000  2020  866705688.0
16056  2020    7481800.0
16106  2020     649342.0

[183 rows x 6 columns]
```

[21]:
```
popSN
```

[21]:
```
       Country Name Country Code    Indicator Name Indicator Code  Year  \
206           Sudan          SDN  Urban population    SP.URB.TOTL  1960
216     South Sudan          SSD  Urban population    SP.URB.TOTL  1960
472           Sudan          SDN  Urban population    SP.URB.TOTL  1961
482     South Sudan          SSD  Urban population    SP.URB.TOTL  1961
738           Sudan          SDN  Urban population    SP.URB.TOTL  1962
…             …          …               …             …    …
15644   South Sudan          SSD  Urban population    SP.URB.TOTL  2018
15900         Sudan          SDN  Urban population    SP.URB.TOTL  2019
```

```
15910  South Sudan          SSD  Urban population    SP.URB.TOTL  2019
16166        Sudan          SDN  Urban population    SP.URB.TOTL  2020
16176  South Sudan          SSD  Urban population    SP.URB.TOTL  2020

       Population
206      810732.0
216      248681.0
472      872667.0
482      252989.0
738      939444.0
…             …
15644   2152927.0
15900  14957232.0
15910   2201250.0
16166  15458183.0
16176   2261021.0

[122 rows x 6 columns]
```

And there, I realized what was wrong. It turns out there are independent territories in China and Sudan and their populations were mixing up, that's why I was getting the odd graphs above. I solved the issue by changing the code a bit, so instead of str.contains I used "loc":

[22]:
```
popCA=newpop.loc[newpop['Country Name']=='China'] #finding the data for the␣
 ↪country specifically called "China"
popSN=newpop.loc[newpop['Country Name']=='Sudan'] #finding the data for the␣
 ↪country specificaly called "Sudan"
```

[23]:
```
popCA #printing the new (fixed) table
```

[23]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
40           China          CHN  Urban population    SP.URB.TOTL  1960
306          China          CHN  Urban population    SP.URB.TOTL  1961
572          China          CHN  Urban population    SP.URB.TOTL  1962
838          China          CHN  Urban population    SP.URB.TOTL  1963
1104         China          CHN  Urban population    SP.URB.TOTL  1964
…              …            …                …              …    …
14936        China          CHN  Urban population    SP.URB.TOTL  2016
15202        China          CHN  Urban population    SP.URB.TOTL  2017
15468        China          CHN  Urban population    SP.URB.TOTL  2018
15734        China          CHN  Urban population    SP.URB.TOTL  2019
16000        China          CHN  Urban population    SP.URB.TOTL  2020

        Population
40     108085352.0
306    110327936.0
572    114685540.0
```

```
838    121162226.0
1104   127791981.0
…              …
14936  787376534.0
15202  809246214.0
15468  829760595.0
15734  848982855.0
16000  866705688.0

[61 rows x 6 columns]
```

[24]:
```
popSN #printing the new (fixed) table
```

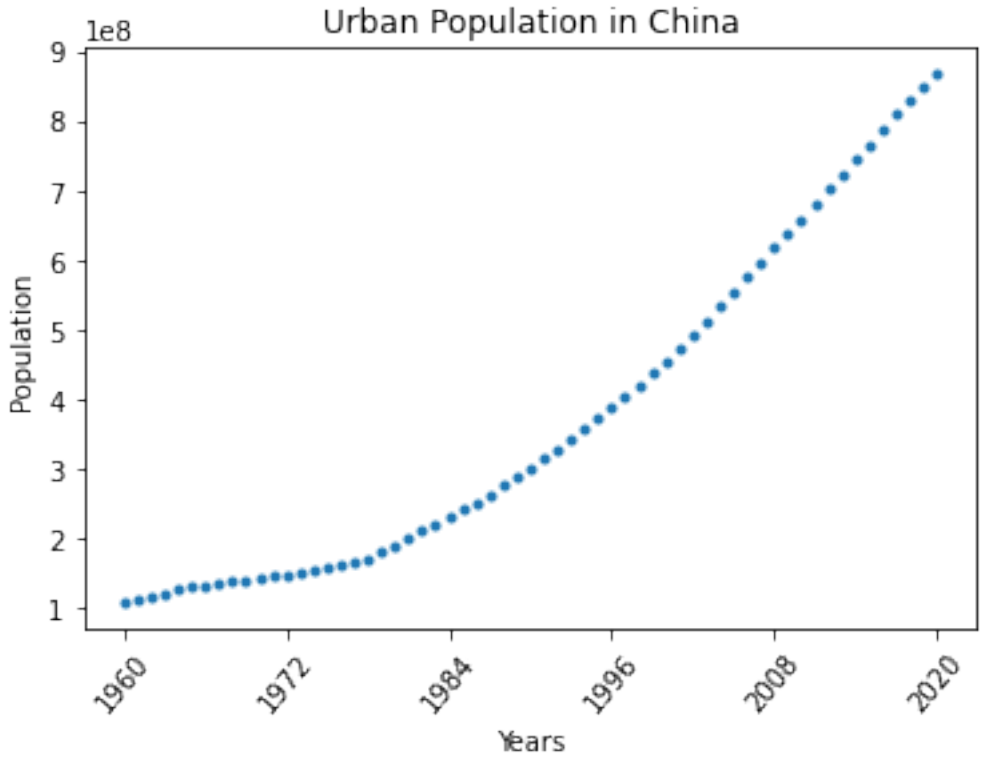
[24]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
206          Sudan          SDN  Urban population    SP.URB.TOTL  1960
472          Sudan          SDN  Urban population    SP.URB.TOTL  1961
738          Sudan          SDN  Urban population    SP.URB.TOTL  1962
1004         Sudan          SDN  Urban population    SP.URB.TOTL  1963
1270         Sudan          SDN  Urban population    SP.URB.TOTL  1964
…              …            …                …              …    …
15102        Sudan          SDN  Urban population    SP.URB.TOTL  2016
15368        Sudan          SDN  Urban population    SP.URB.TOTL  2017
15634        Sudan          SDN  Urban population    SP.URB.TOTL  2018
15900        Sudan          SDN  Urban population    SP.URB.TOTL  2019
16166        Sudan          SDN  Urban population    SP.URB.TOTL  2020

       Population
206      810732.0
472      872667.0
738      939444.0
1004    1011470.0
1270    1089191.0
…             …
15102  13596343.0
15368  14027565.0
15634  14480887.0
15900  14957232.0
16166  15458183.0

[61 rows x 6 columns]
```

Great, that fixed the issue, now I'll print the plots but makes me think I should change the class a little:

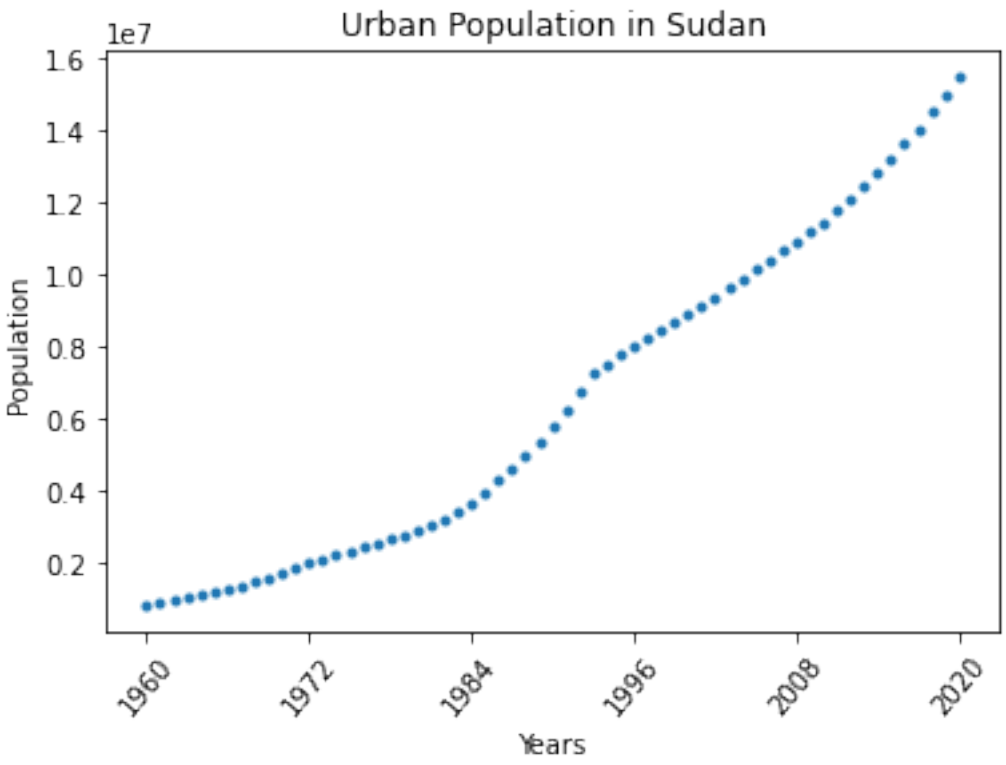
[25]:
```
'''Fixing the class "fixplot" a little so that it works fine with the above␣
 ↪issue'''
class fixplot: #defining the class fixplot
```

```
    def __init__(self, xvalues, yvalues):  #defining __init__
        self.xdata = xvalues
        self.ydata = yvalues
    def fixing(CountryName): #defining the method fixing that takes CountryName
        pop1=newpop.loc[newpop['Country Name']==CountryName] #small table with␣
↪the chosen country data but more exact/specific with the country's name this␣
↪time.
        pop1.index=pop1.index-min(pop1.index) #getting the indices to start␣
↪from zero
        desind=np.linspace(min(pop1.index),max(pop1.index),6,dtype=int) ␣
↪#creating 6 equally spaced indices
        plt.plot(pop1['Year'],pop1['Population'],'.') #plotting the plot urban␣
↪pop vs years for the chosen country
        plt.xticks(pop1.Year[desind],rotation=50) #using xticks to print 6␣
↪equally spaced dates at the indices generated above
        plt.xlabel('Years') #label x-axis
        plt.ylabel('Population') #label y-axis
        plt.title('Urban Population in %s' %CountryName); #give the plot a title
```

[26]:
```
fixplot.fixing('China') #checking if the new fixed method works properly
```



[27]:
```
fixplot.fixing("Sudan") #checking that for Sudan too
```



Amazing, everything looks fine now!

8 Conclusions

–For Lebanon: If we look at the top, we can notice that there's a slight decrease (or beginning of decrease) in urban population. Being a resident of Lebanon, I can tell you it's because of the economic and political crises Lebanon has been going through since 2019. I am pretty sure more recent data would show more decline because things have been becoming worse rather than improving, unfortunately.

–For Ukraine: The web says that Ukraine's population (and obviously its urban population) has been declining since the 1990s because of high emigration rates, coupled with high death rates and low birth rates. Population shrinkage averaged 300,000 people annually since 1993. In 2007, the country's rate of population decline was the fourth highest in the world.

–For Syria: Over the last decade, Syrians have faced mass displacement due to the devastating civil war that has taken place since 2012 up until today. However, that things are currently relatively better there, the urban population is going back to increasing– as seen in the plot. (I know that, last summer, Lebanese Security Forces were coordinating the return of many Syrian families to the "safe" regions in Syria)

–Correlation Analysis: Now, that I have talked about both Syria and Lebanon, I can tell you about the meaning behind the correlation analysis result we got earlier in the notebook. (or at least what I think it means) So 2011-2014 was peak war time in Syria and as I would expect, many Syrians

seeked refuge in Lebanon, which is a neighboring country to Syria. So the decrease of the Syrian urban population during war period should be strongly and negatively correlated with the urban population increase during that same period, which is consistent with what we got!

9 Truth Test Suit

To prove that our code is working correctly, we would need to run some truth tests. I'll run 3 tests:

9.1 Truth Test 1

Here, I will test whether the correlation analysis code was working fine:

[28]:
```
'''Creating a fake data set for truth testing purposes'''
myfakedata={'x':[1,2,3,4,5],'y':[-2,-4,-6,-8,-10]} #creating a fake data of x␣
 ↪and y values and calling it myfakedata
myfakedata=pd.DataFrame(myfakedata) #changing myfakedata into a pandas table
```

[29]:
```
myfakedata #printing myfakedata
```

[29]:
```
   x   y
0  1  -2
1  2  -4
2  3  -6
3  4  -8
4  5 -10
```

[30]:
```
myfakedata['x'].corr(myfakedata['y'], method='pearson') #finding the␣
 ↪correlation between x and y
```

[30]:
```
-0.9999999999999999
```

Nice! we got a -1 correlation which means my code has passed the first test!

10 Truth Test 2

For the second test, I will be testing the loc function:

To get the subtable for Lebanon, we used the code:

[31]:
```
popLB=newpop.loc[newpop['Country Name']=='Lebanon'] #getting the data for␣
 ↪Lebanon by specifying the country name
popLB #printing the outcome
```

[31]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
130        Lebanon          LBN  Urban population    SP.URB.TOTL  1960
396        Lebanon          LBN  Urban population    SP.URB.TOTL  1961
662        Lebanon          LBN  Urban population    SP.URB.TOTL  1962
928        Lebanon          LBN  Urban population    SP.URB.TOTL  1963
```

```
1194       Lebanon          LBN  Urban population    SP.URB.TOTL  1964
…            …          …              …             …   …
15026      Lebanon          LBN  Urban population    SP.URB.TOTL  2016
15292      Lebanon          LBN  Urban population    SP.URB.TOTL  2017
15558      Lebanon          LBN  Urban population    SP.URB.TOTL  2018
15824      Lebanon          LBN  Urban population    SP.URB.TOTL  2019
16090      Lebanon          LBN  Urban population    SP.URB.TOTL  2020

       Population
130      764264.0
396      821157.0
662      880861.0
928      942378.0
1194    1004429.0
…             …
15026   5926427.0
15292   6030303.0
15558   6076955.0
15824   6084990.0
16090   6069524.0

[61 rows x 6 columns]
```

Nice, we have 61 rows and 6 columns. Now, if the loc function is working properly, we should get the same number of rows (and columns) if we changed the criteria from country name to country code for instance. So to test this, I can say:

[32]:
```
popLB=newpop.loc[newpop['Country Code']=='LBN'] #getting the data for Lebanon␣
 ↪by specifying the country code
popLB #printing the outcome
```

[32]:
```
      Country Name Country Code    Indicator Name Indicator Code  Year  \
130        Lebanon          LBN  Urban population    SP.URB.TOTL  1960
396        Lebanon          LBN  Urban population    SP.URB.TOTL  1961
662        Lebanon          LBN  Urban population    SP.URB.TOTL  1962
928        Lebanon          LBN  Urban population    SP.URB.TOTL  1963
1194       Lebanon          LBN  Urban population    SP.URB.TOTL  1964
…            …          …              …             …   …
15026      Lebanon          LBN  Urban population    SP.URB.TOTL  2016
15292      Lebanon          LBN  Urban population    SP.URB.TOTL  2017
15558      Lebanon          LBN  Urban population    SP.URB.TOTL  2018
15824      Lebanon          LBN  Urban population    SP.URB.TOTL  2019
16090      Lebanon          LBN  Urban population    SP.URB.TOTL  2020

       Population
130      764264.0
396      821157.0
```

```
662      880861.0
928      942378.0
1194    1004429.0
…             …
15026   5926427.0
15292   6030303.0
15558   6076955.0
15824   6084990.0
16090   6069524.0

[61 rows x 6 columns]
```

Great! I got the exact same table with the same number of rows and columns, which means code passed the second truth test!

11 Truth Test 3

For this one, I will just add a fake column with some country names and use loc and prove that I am really getting the columns I am asking for through my code:

[33]:
```
'''Modifying myfakedata for the third truth test'''
myfakedata={'x':[1,2,3,4,5],'y':[-2,-4,-6,-8,-10],'Country Name':
 ↪['Lebanon','Syria','Lebanon','Lebanon','Syria']} #adding a new fake column␣
 ↪with random country names
myfakedata=pd.DataFrame(myfakedata) #making the new fake data set, a pandas␣
 ↪table
```

[34]:
```
myfakedata
```

[34]:
```
   x   y Country Name
0  1  -2      Lebanon
1  2  -4        Syria
2  3  -6      Lebanon
3  4  -8      Lebanon
4  5 -10        Syria
```

Now, let's see if I'll actually get all the rows for Lebanon, for example, when we ask for that using my code:

[35]:
```
'''Performing the truth test'''
fakeLB=myfakedata.loc[myfakedata['Country Name']=='Lebanon'] #getting the fake␣
 ↪data about Lebanon
fakeLB #printing the (fake) outcome
```

[35]:
```
   x  y Country Name
0  1 -2      Lebanon
2  3 -6      Lebanon
```

```
3  4 -8      Lebanon
```

Nice! The code returned the rows pertaining to Lebanon, as expected, and for this, it passed the third test!

12 Fourth Test:

For this last test, it will be more of a data test (checked with Andrea and she was ok with this). I will basically look up the urban population from a credible source online for a certain country and see if it matches my data:

[37]:
```
'''Performing the fourth truth test'''
desind=popLB.loc[popLB['Year']=='2020'] #getting the desired row containing the␣
 ↪population value in Lebanon for 2020
desind['Population'] #printing the population value for that row
```

[37]:
```
16090    6069524.0
Name: Population, dtype: float64
```

https://www.macrotrends.net/countries/LBN/lebanon/urban-population#:~:text=Lebanon%20urban%20popula

https://tradingeconomics.com/lebanon/urban-population-wb-data.html

These two websites clearly state the number we got above for the urban population in Lebanon for the year 2020, which adds to the data's credibility. (we could do that for more countries but due to the limitations of time and space, I will just use this example)

all tests should be related to the code you used. you should fix test above by making a correlation analysis with the fake data instead of slope other ideas: -use country code with loc, you should get the same number of columns and rows -check population online and make sure it matches what we have -add column with country names to myfakedata and then show that loc really does call them