# Line plot (1)

With matplotlib, you can create a bunch of different plots in Python. The most basic plot is the line plot. A general recipe is given here.

import matplotlib.pyplot as plt
plt.plot(x,y)
plt.show()

In the video, you already saw how much the world population has grown over the past years. Will it continue to do so? The world bank has estimates of the world population for the years 1950 up to 2100. The years are loaded in your workspace as a list called year, and the corresponding populations as a list called pop.

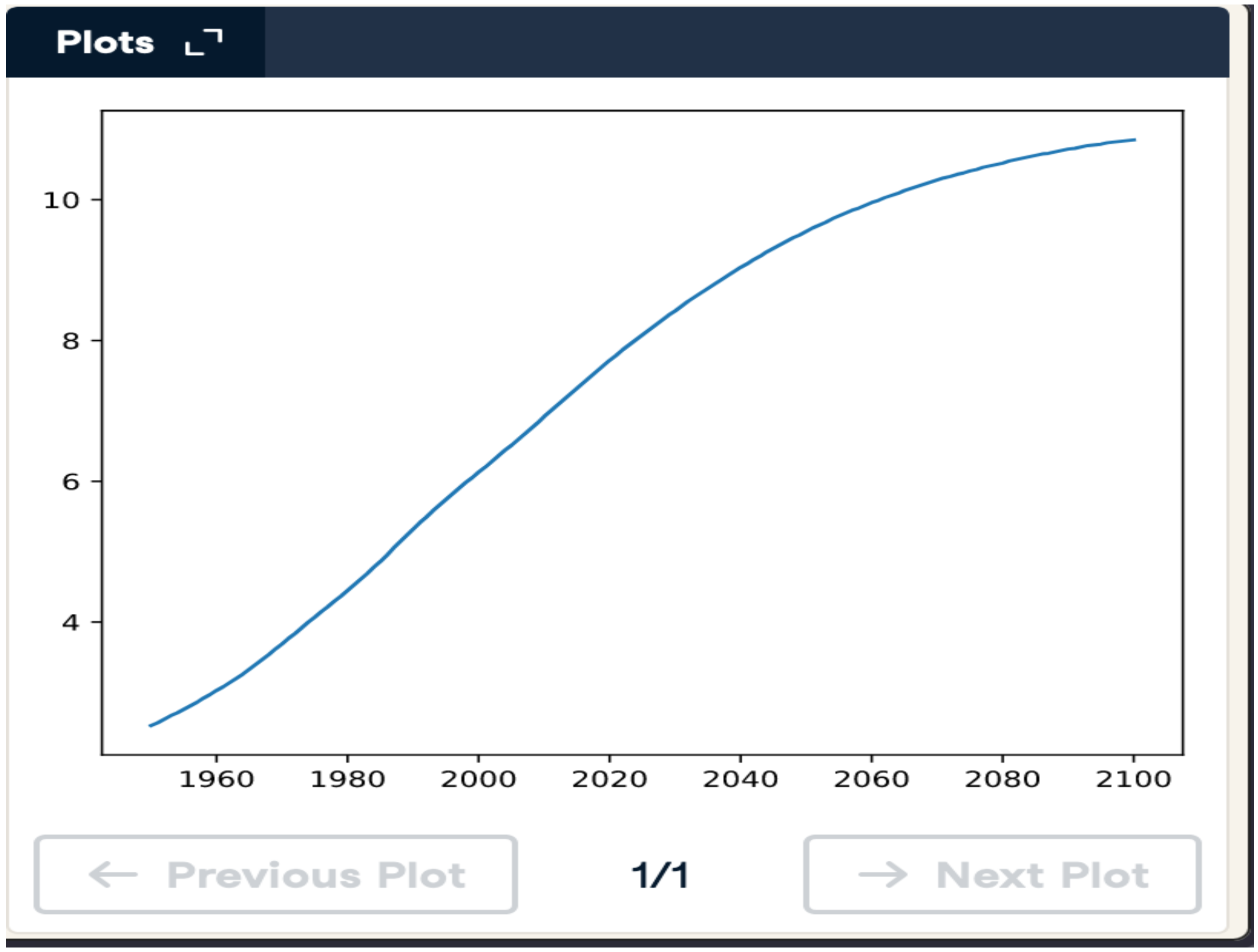
Instructions

- print() the last item from both the year and the pop list to see what the predicted population for the year 2100 is. Use two print() functions.
- Before you can start, you should import matplotlib.pyplot as plt. pyplot is a sub-package of matplotlib, hence the dot.
- Use plt.plot() to build a line plot. year should be mapped on the horizontal axis, pop on the vertical axis. Don't forget to finish off with the show() function to actually display the plot.

```
# Print the last item from year and pop

print(year[-1])

print(pop[-1])




# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt
```

```
# Make a line plot: year on the x-axis, pop on the y-axis

plt.plot(year,pop)


# Display the plot with plt.show()

plt.show()
```

**Graphical Plot**



**Python Shell**

```
# Print the last item from year and pop

print(year[-1])

print(pop[-1])


# Import matplotlib.pyplot as plt

import matplotlib.pyplot as plt


# Make a line plot: year on the x-axis, pop on the y-axis

plt.plot(year,pop)


# Display the plot with plt.show()

plt.show()

2100

10.85
```

## Line plot (3)

Now that you've built your first line plot, let's start working on the data that professor Hans Rosling used to build his beautiful bubble chart. It was collected in 2007. Two lists are available for you:

- life_exp which contains the life expectancy for each country and
- gdp_cap, which contains the GDP per capita (i.e. per person) for each country expressed in US Dollars.

GDP stands for Gross Domestic Product. It basically represents the size of the economy of a country. Divide this by the population and you get the GDP per capita.

matplotlib.pyplot is already imported as plt, so you can get started straight away.

**Instructions**

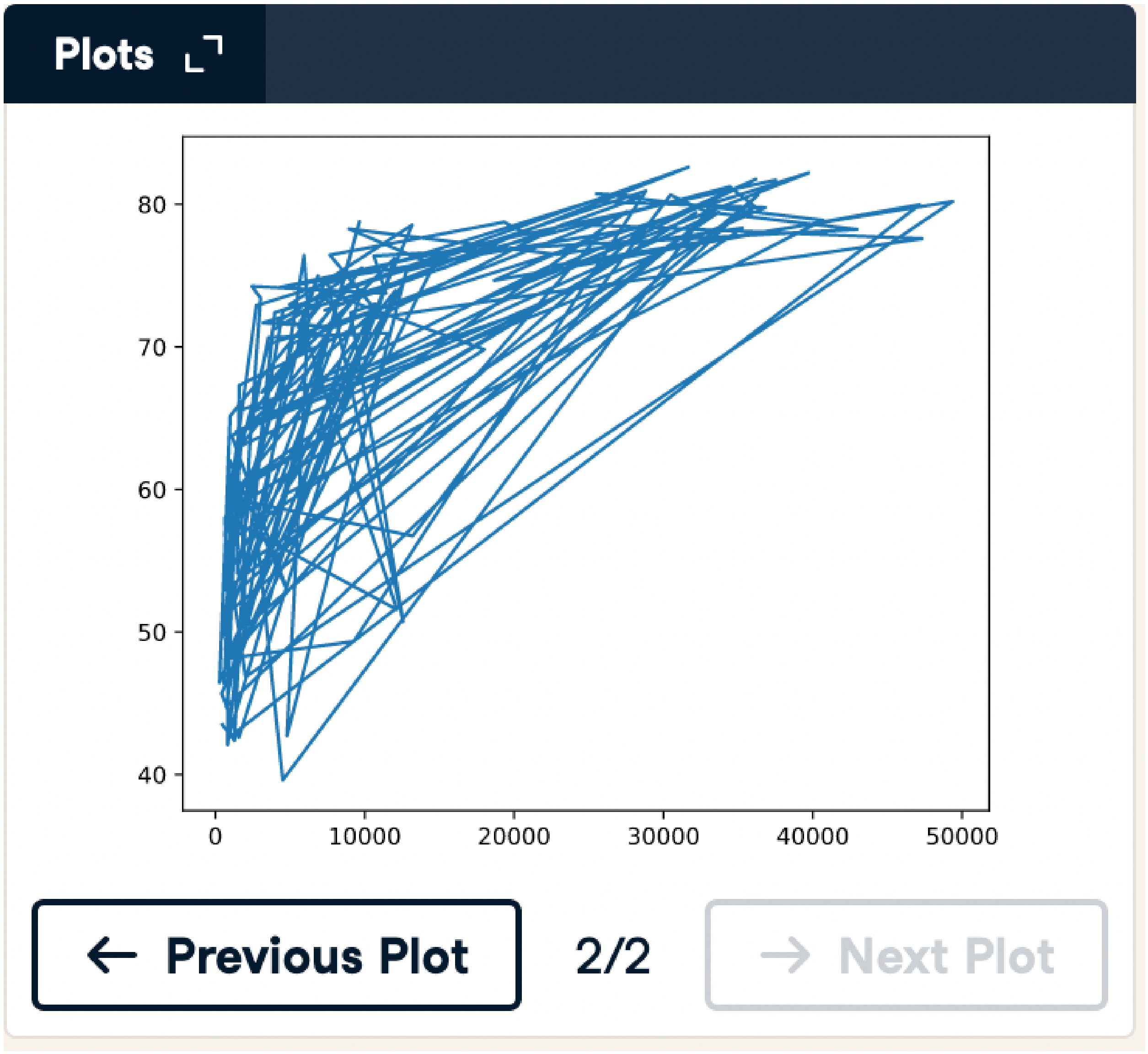
- Print the last item from both the list gdp_cap, and the list life_exp; it is information about Zimbabwe.
- Build a line chart, with gdp_cap on the x-axis, and life_exp on the y-axis. Does it make sense to plot this data on a line plot?
- Don't forget to finish off with a plt.show() command, to actually display the plot.

```
# Print the last item of gdp_cap and life_exp
print(gdp_cap[-1])
print(life_exp[-1])


# Make a line plot, gdp_cap on the x-axis, life_exp on the
y-axis
plt.plot(gdp_cap, life_exp)

# Display the plot
plt.show()
```

**Python Shell**

```
# Print the last item of gdp_cap and life_exp
print(gdp_cap[-1])
print(life_exp[-1])


# Make a line plot, gdp_cap on the x-axis, life_exp on the y-axis
plt.plot(gdp_cap, life_exp)

# Display the plot
plt.show()
469.70929810000007
43.487
```

**Plot**


Plots
80
70
60
50
40
0
10000
20000
30000
40000
50000
← Previous Plot
2/2
→ Next Plot

# Scatter Plot (1)

When you have a time scale along the horizontal axis, the line plot is your friend. But in many other cases, when you're trying to assess if there's a correlation between two variables, for example, the scatter plot is the better choice. Below is an example of how to build a scatter plot.

```
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.show()
```

Let's continue with the gdp_cap versus life_exp plot, the GDP and life expectancy data for different countries in 2007. Maybe a scatter plot will be a better alternative?

Again, the matplotlib.pyplot package is available as plt.

**Instructions**

- **Change the line plot that's coded in the script to a scatter plot.**
- **A correlation will become clear when you display the GDP per capita on a logarithmic scale. Add the line plt.xscale('log').**
- **Finish off your script with plt.show() to display the plot.**

**Python Script**

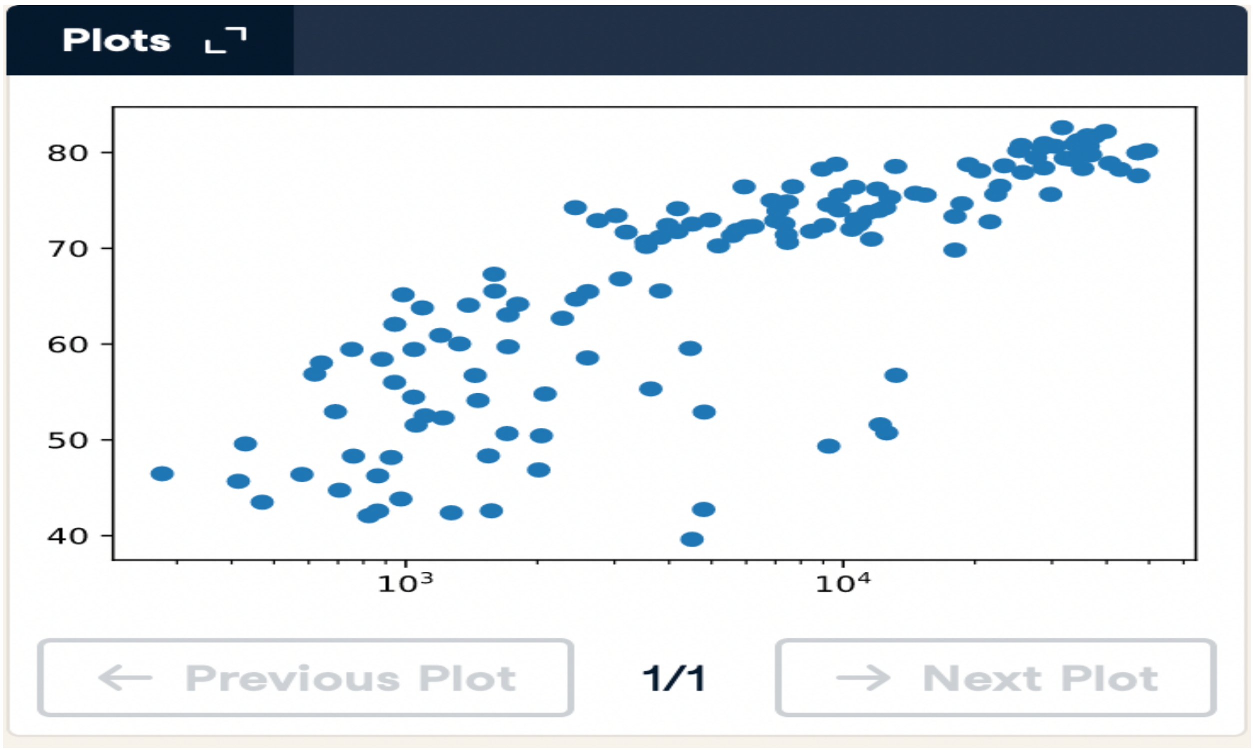
```
# Change the line plot below to a scatter plot
plt.scatter(gdp_cap, life_exp)

# Put the x-axis on a logarithmic scale
plt.xscale('log')

# Show plot
plt.show()
```

**Python Shell**

```
# Print the last item of gdp_cap and life_exp
print(gdp_cap[-1])
print(life_exp[-1])
```

```
# Make a line plot, gdp_cap on the x-axis, life_exp on the y-axis
plt.plot(gdp_cap, life_exp)

# Display the plot
plt.show()
# Change the line plot below to a scatter plot
plt.scatter(gdp_cap, life_exp)

# Put the x-axis on a logarithmic scale
plt.xscale('log')

# Show plot
plt.show()
```

**Plot**



# Scatter plot (2)

In the previous exercise, you saw that the higher GDP usually corresponds to a higher life expectancy. In other words, there is a positive correlation.

Do you think there's a relationship between population and life expectancy of a country? The list life_exp from the previous exercise is already available. In addition, now also pop is available, listing the corresponding populations for the countries in 2007. The populations are in millions of people.

Instructions

- Start from scratch: import matplotlib.pyplot as plt.
- Build a scatter plot, where pop is mapped on the horizontal axis, and life_exp is mapped on the vertical axis.
- Finish the script with plt.show() to actually display the plot. Do you see a correlation?

Python Shell

```python
# Import package

import matplotlib.pyplot as plt



# Print Life Expectancy Statistics

print (life_exp)



#Print Population Statistics

print (pop)



# Build Scatter plot

plt.scatter(pop, life_exp)
```

```
# Show plot

plt.show()
```

```
# Import package

import matplotlib.pyplot as plt



# Print Life Expectancy Statistics

print (life_exp)



#Print Population Statistics

print (pop)



# Build Scatter plot

plt.scatter(pop, life_exp)



# Show plot

plt.show()
```

```
[43.828, 76.423, 72.301, 42.731, 75.32, 81.235, 79.829, 75.635, 64.062, 79.441, 56.728,
65.554, 74.852, 50.728, 72.39, 73.005, 52.295, 49.58, 59.723, 50.43, 80.653,
44.74100000000001, 50.651, 78.553, 72.961, 72.889, 65.152, 46.462, 55.322, 78.782, 48.328,
75.748, 78.273, 76.486, 78.332, 54.791, 72.235, 74.994, 71.33800000000002, 71.878,
51.57899999999999, 58.04, 52.947, 79.313, 80.657, 56.735, 59.448, 79.406, 60.022, 79.483,
70.259, 56.007, 46.38800000000001, 60.916, 70.19800000000001, 82.208,
73.33800000000002, 81.757, 64.69800000000001, 70.65, 70.964, 59.545, 78.885, 80.745,
80.546, 72.567, 82.603, 72.535, 54.11, 67.297, 78.623, 77.58800000000002, 71.993, 42.592,
45.678, 73.952, 59.44300000000001, 48.303, 74.241, 54.467, 64.164, 72.801, 76.195, 66.803,
74.543, 71.164, 42.082, 62.069, 52.90600000000001, 63.785, 79.762, 80.204, 72.899, 56.867,
46.859, 80.196, 75.64, 65.483, 75.53699999999998, 71.752, 71.421, 71.688, 75.563, 78.098,
78.74600000000002, 76.442, 72.476, 46.242, 65.528, 72.777, 63.062, 74.002,
42.56800000000001, 79.972, 74.663, 77.926, 48.159, 49.339, 80.941, 72.396, 58.556, 39.613,
80.884, 81.70100000000002, 74.143, 78.4, 52.517, 70.616, 58.42, 69.819, 73.923, 71.777,
51.542, 79.425, 78.242, 76.384, 73.747, 74.249, 73.422, 62.698, 42.38399999999999, 43.487]
```

```
[31.889923, 3.600523, 33.333216, 12.420476, 40.301927, 20.434176, 8.199783, 0.708573,
150.448339, 10.392226, 8.078314, 9.119152, 4.552198, 1.639131, 190.010647, 7.322858,
14.326203, 8.390505, 14.131858, 17.696293, 33.390141, 4.369038, 10.238807, 16.284741,
1318.683096, 44.22755, 0.71096, 64.606759, 3.80061, 4.133884, 18.013409, 4.493312,
11.416987, 10.228744, 5.46812, 0.496374, 9.319622, 13.75568, 80.264543, 6.939688,
0.551201, 4.906585, 76.511887, 5.23846, 61.083916, 1.454867, 1.688359, 82.400996,
22.873338, 10.70629, 12.572928, 9.947814, 1.472041, 8.502814, 7.483763, 6.980412,
9.956108, 0.301931, 1110.396331, 223.547, 69.45357, 27.499638, 4.109086, 6.426679,
58.147733, 2.780132, 127.467972, 6.053193, 35.610177, 23.301725, 49.04479, 2.505559,
3.921278, 2.012649, 3.193942, 6.036914, 19.167654, 13.327079, 24.821286, 12.031795,
3.270065, 1.250882, 108.700891, 2.874127, 0.684736, 33.757175, 19.951656, 47.76198,
2.05508, 28.90179, 16.570613, 4.115771, 5.675356, 12.894865, 135.031164, 4.627926,
3.204897, 169.270617, 3.242173, 6.667147, 28.674757, 91.077287, 38.518241, 10.642836,
3.942491, 0.798094, 22.276056, 8.860588, 0.199579, 27.601038, 12.267493, 10.150265,
6.144562, 4.553009, 5.447502, 2.009245, 9.118773, 43.997828, 40.448191, 20.378239,
42.292929, 1.133066, 9.031088, 7.554661, 19.314747, 23.174294, 38.13964, 65.068149,
5.701579, 1.056608, 10.276158, 71.158647, 29.170398, 60.776238, 301.139947, 3.447496,
26.084662, 85.262356, 4.018332, 22.211743, 11.746035, 12.311143]
```

## Build a histogram (1)

life_exp, the list containing data on the life expectancy for different countries in 2007, is available in your Python shell.

To see how life expectancy in different countries is distributed, let's create a histogram of life_exp.

matplotlib.pyplot is already available as plt.

**Instructions**

- Use plt.hist() to create a histogram of the values in life_exp. Do not specify the number of bins; Python will set the number of bins to 10 by default for you.

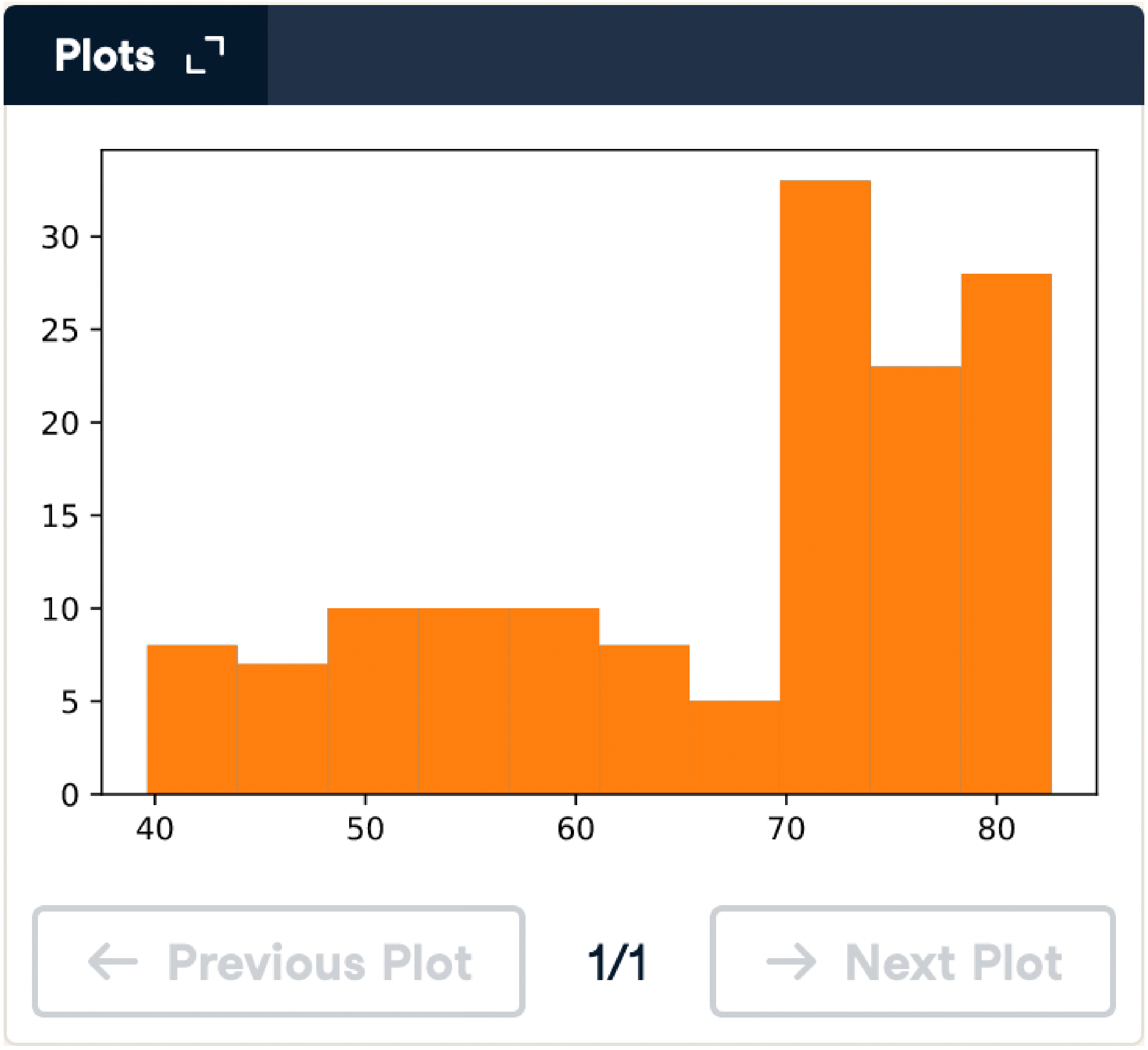
- Add plt.show() to actually display the histogram. Can you tell which bin contains the most observations?

**Python Code**

```
# Create histogram of life_exp data
plt.hist(life_exp)

# Display histogram
plt.show()
```

**Python Shell**

**Plot**



## Build a histogram (2): bins

In the previous exercise, you didn't specify the number of bins. By default, Python sets the number of bins to 10 in that case. The number of bins is pretty important. Too few bins will oversimplify reality and won't show you the details. Too many bins will overcomplicate reality and won't show the bigger picture.

To control the number of bins to divide your data in, you can set the bins argument.

That's exactly what you'll do in this exercise. You'll be making two plots here. The code in the script already includes plt.show() and plt.clf() calls; plt.show() displays a plot; plt.clf() cleans it up again so you can start afresh.

As before, life_exp is available and matplotlib.pyplot is imported as plt.

Instructions

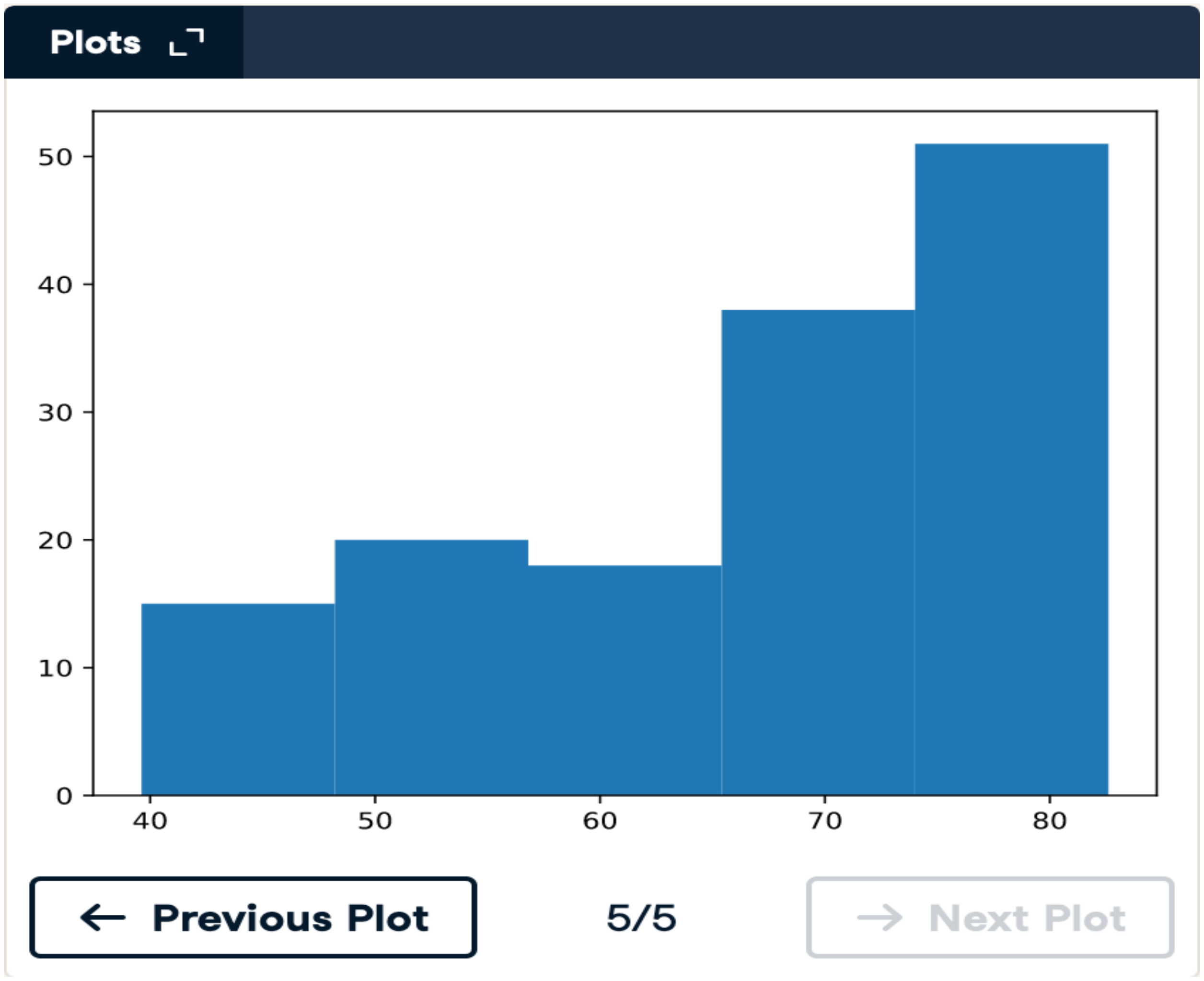
- 
- Build another histogram of life_exp, this time with 20 bins. Is this better?

**Python Code**

- Build a histogram of life_exp, with 5 bins. Can you tell which bin contains the most observations?

```
# Build histogram with 5 bins
plt.hist(life_exp, bins=5)

# Show and clean up plot
plt.show()
plt.clf()

# Build histogram with 20 bins
#plt.hist(life_exp, bins=20)
```

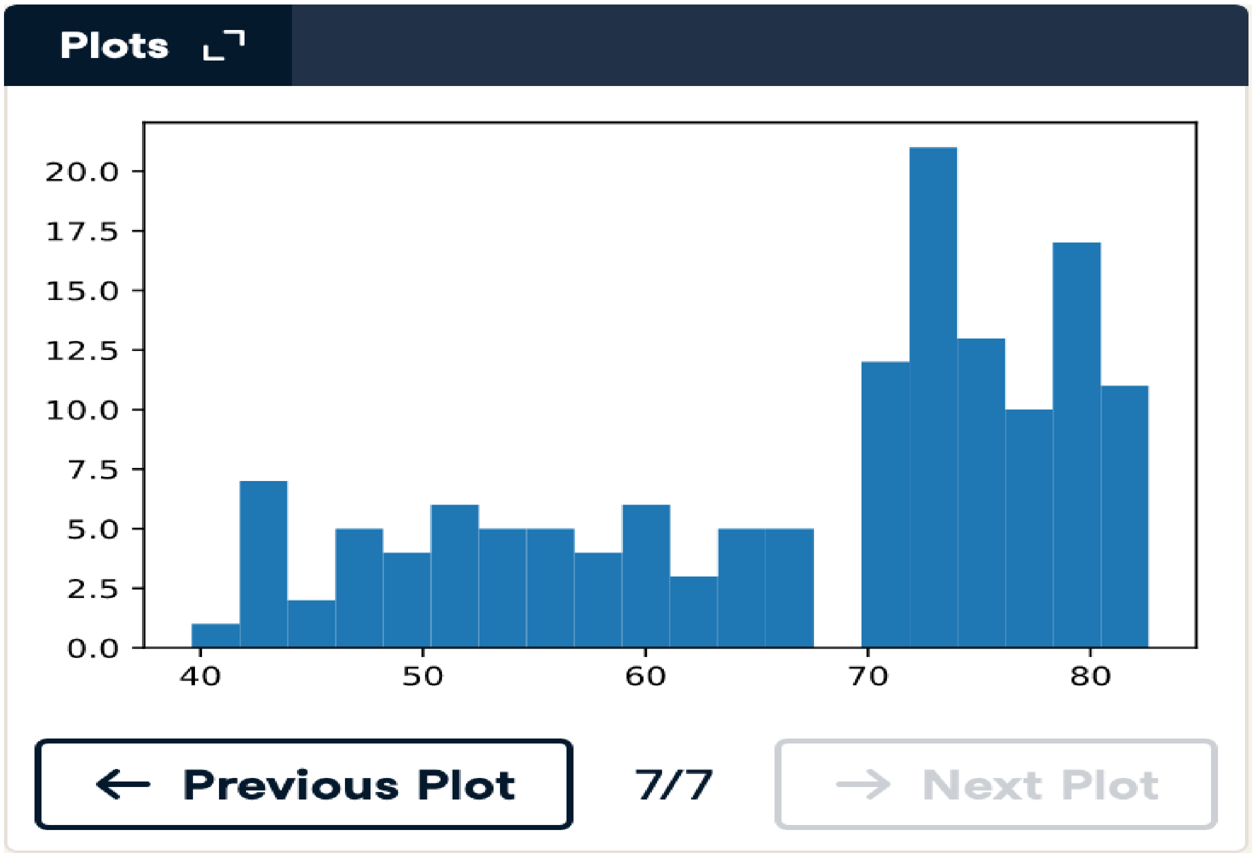
```
# Show and clean up again
#plt.show()
#plt.clf()
```



```
# Build histogram with 5 bins
plt.hist(life_exp, bins=5)

# Show and clean up plot
plt.show()
plt.clf()

# Build histogram with 20 bins
plt.hist(life_exp, bins=20)
```

```
# Show and clean up again
plt.show()
plt.clf()
```

**Plot Diagram**



# Build a histogram (3): compare

**In the video, you saw population pyramids for the present day and for the future. Because we were using a histogram, it was very easy to make a comparison.**

**Let's do a similar comparison. life_exp contains life expectancy data for different countries in 2007. You also have access to a second list now, life_exp1950, containing similar data for 1950. Can you make a histogram for both datasets?**

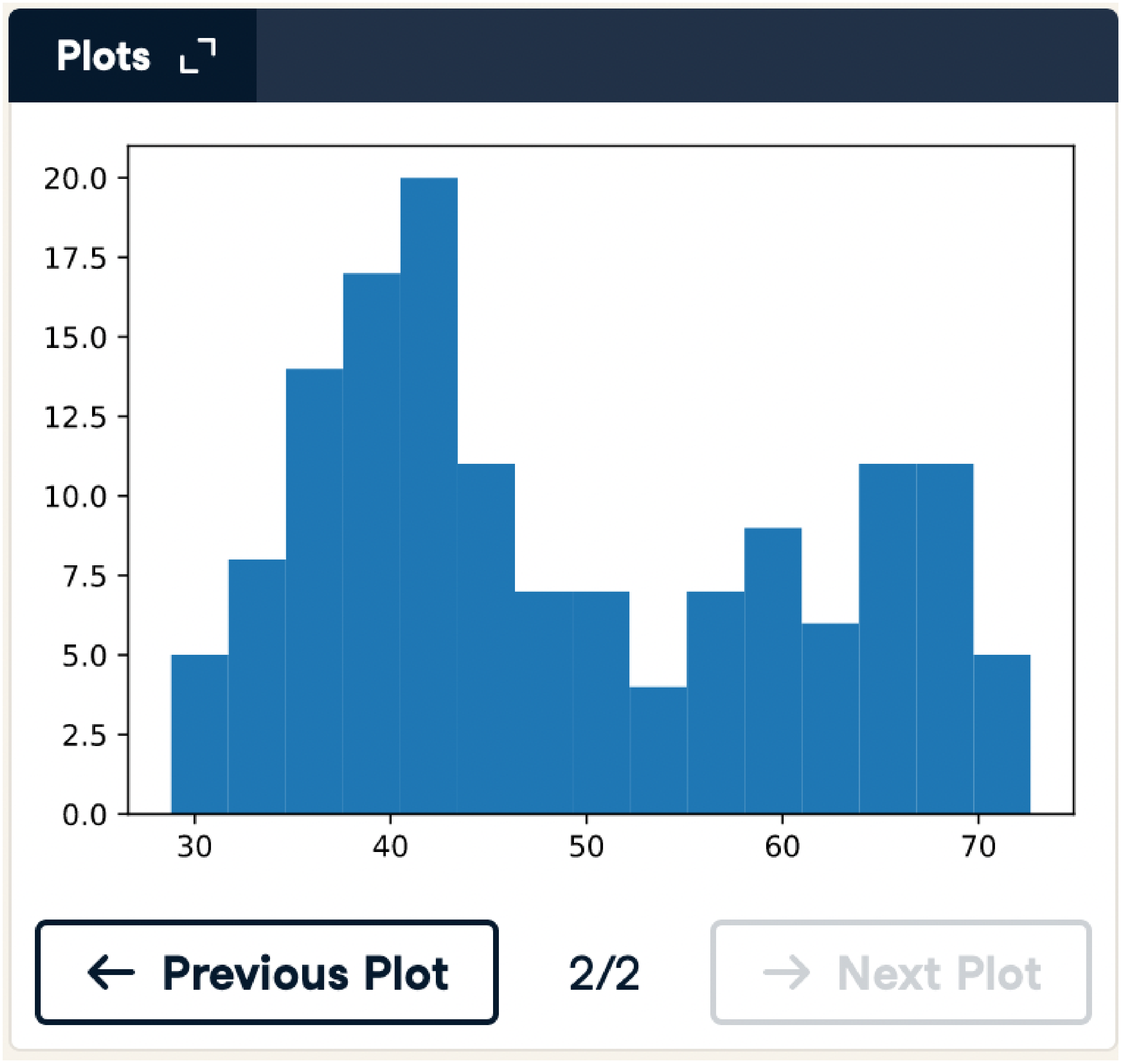
**You'll again be making two plots. The plt.show() and plt.clf() commands to render everything nicely are already included. Also matplotlib.pyplot is imported for you, as plt.**

**Instructions**

- **Build a histogram of life_exp with 15 bins.**
- **Build a histogram of life_exp1950, also with 15 bins. Is there a big difference with the histogram for the 2007 data?**

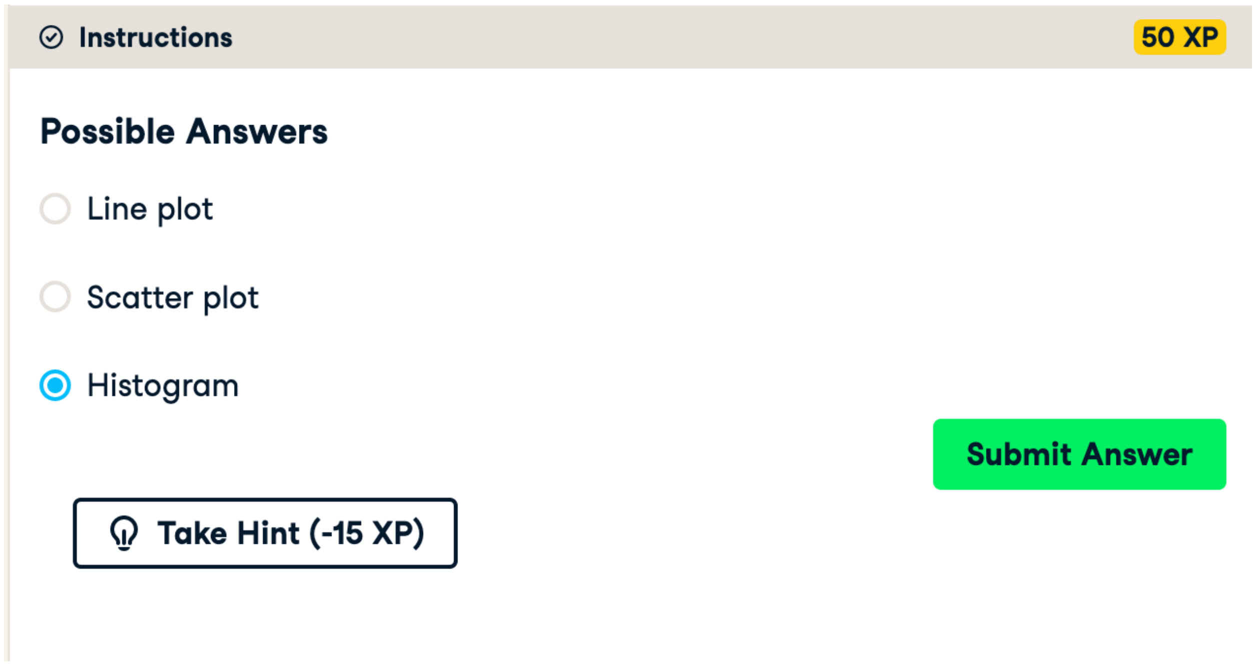
**Python Shell**

```
# Histogram of life_exp, 15 bins
plt.hist(life_exp, bins=15)

# Show and clear plot
plt.show()
plt.clf()

# Histogram of life_exp1950, 15 bins
plt.hist(life_exp1950, bins=15)

# Show and clear plot again
plt.show()
plt.clf()
```


Plots
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
30
40
50
60
70
Previous Plot
2/2
Next Plot

## Choose the right plot (1)

**You're a professor teaching Data Science with Python, and you want to visually assess if the grades on your exam follow a particular distribution. Which plot do you use?**

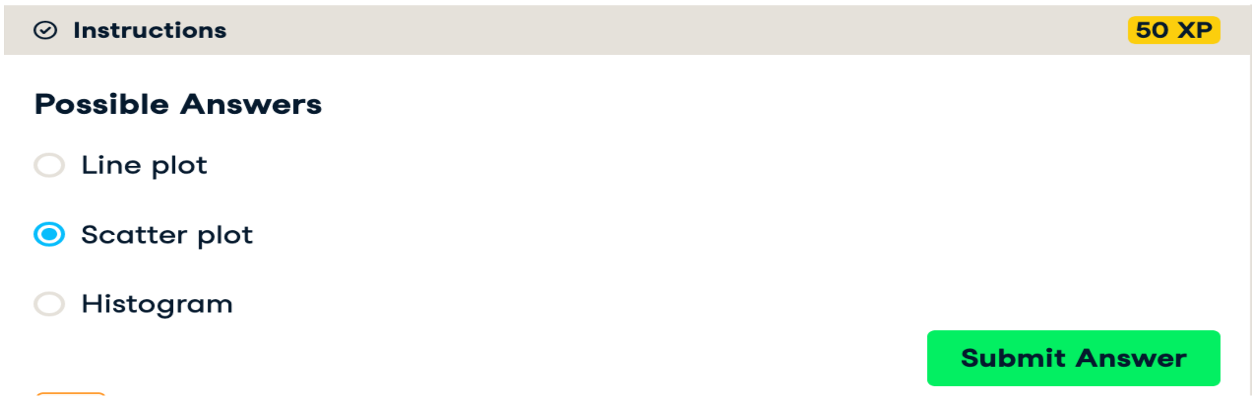

Instructions 50 XP

Possible Answers

- Line plot
- Scatter plot
- [✓ Histogram]

## Choose the right plot (2)

**You're a professor in Data Analytics with Python, and you want to visually assess if longer answers on exam questions lead to higher grades. Which plot do you use?**


Instructions 50 XP

Possible Answers

- Line plot
- [✓ Scatter plot]
- Histogram

# Customization

## Labels

It's time to customize your own plot. This is the fun part, you will see your plot come to life!

You're going to work on the scatter plot with world development data: GDP per capita on the x-axis (logarithmic scale), life expectancy on the y-axis. The code for this plot is available in the script.

As a first step, let's add axis labels and a title to the plot. You can do this with the xlabel(), ylabel() and title() functions, available in matplotlib.pyplot. This sub-package is already imported as plt.

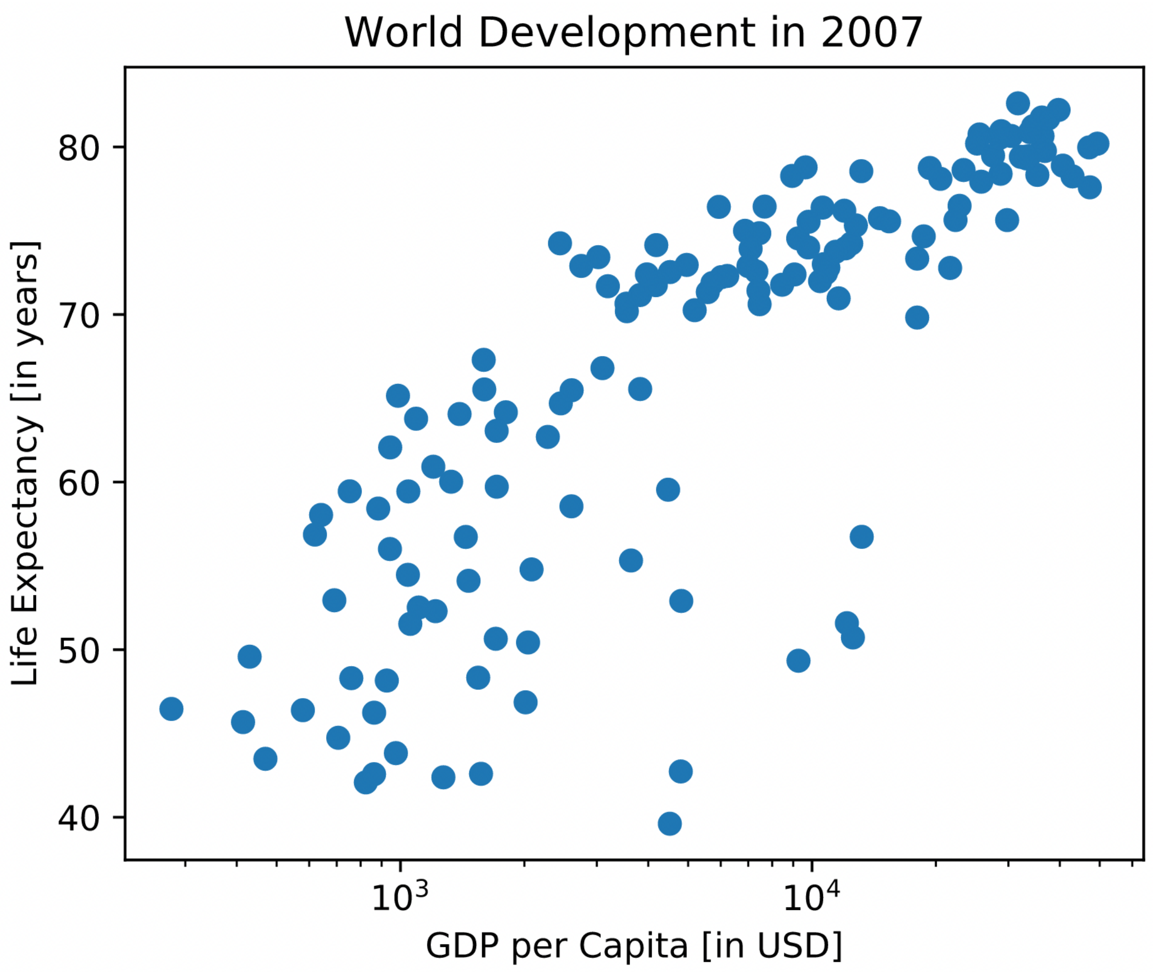
**Instructions**

- The strings xlab and ylab are already set for you. Use these variables to set the label of the x- and y-axis.
- The string title is also coded for you. Use it to add a title to the plot.
- After these customizations, finish the script with plt.show() to actually display the plot.

```
# Basic scatter plot, log scale

plt.scatter(gdp_cap, life_exp)

plt.xscale('log')



# Strings

xlab = 'GDP per Capita [in USD]'
```

```
ylab = 'Life Expectancy [in years]'

title = 'World Development in 2007'



# Add axis labels

plt.xlabel(xlab)

plt.ylabel(ylab)



# Add title

plt.title(title)






# After customizing, display the plot

plt.show()
```

**`Plot`**


World Development in 2007
Life Expectancy [in years]
GDP per Capita [in USD]
80
70
60
50
40
$10^3$
$10^4$

## Ticks

The customizations you've coded up to now are available in the script, in a more concise form.

In the video, Hugo has demonstrated how you could control the y-ticks by specifying two arguments:

```
plt.yticks([0,1,2], ["one","two","three"])
```

In this example, the ticks corresponding to the numbers 0, 1 and 2 will be replaced by *one*, *two* and *three*, respectively.

Let's do a similar thing for the x-axis of your world development chart, with the xticks() function. The tick values 1000, 10000 and 100000 should be replaced by 1k, 10k and 100k. To this end, two lists have already been created for you: tick_val and tick_lab.

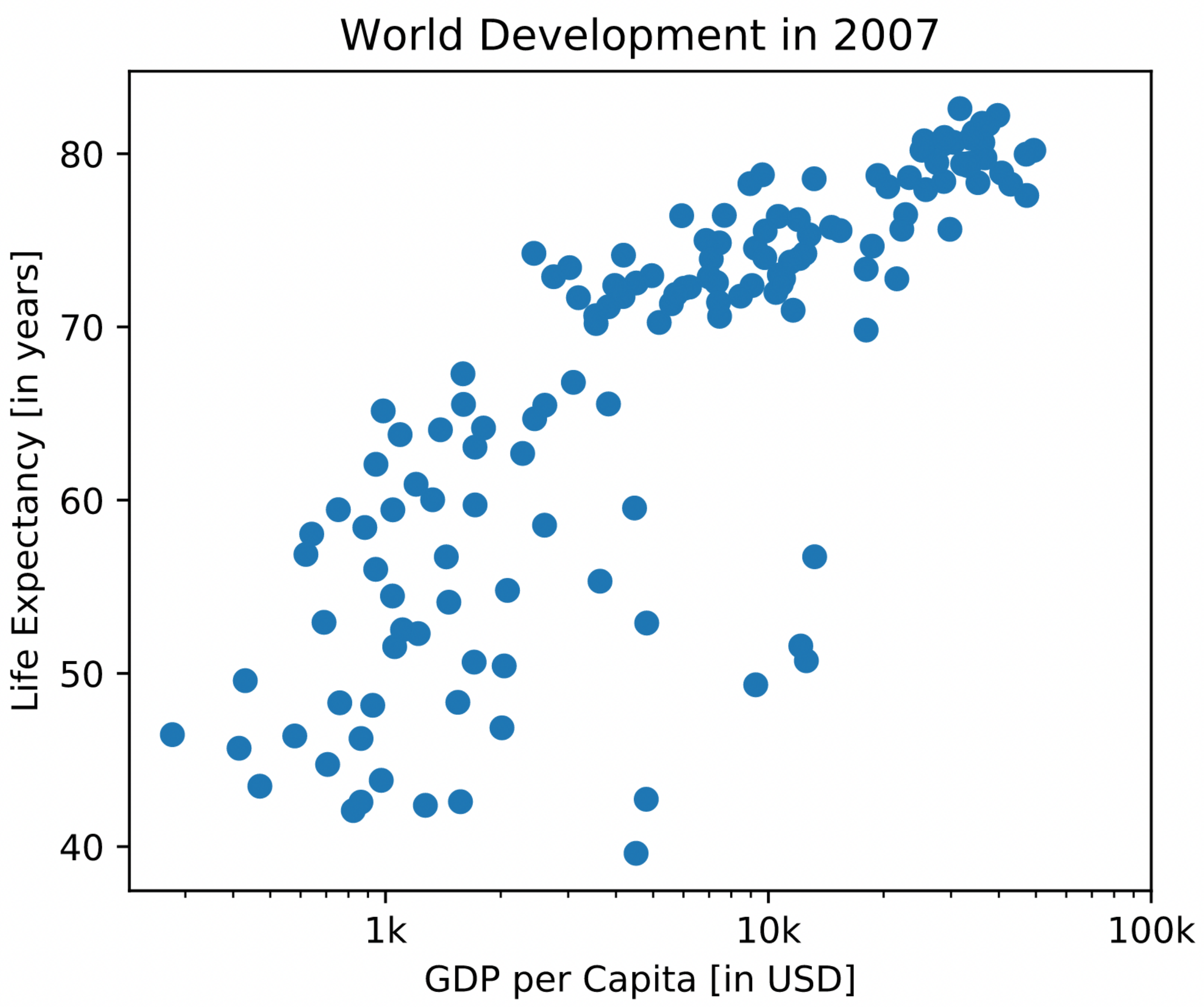
## Instructions

- Use tick_val and tick_lab as inputs to the xticks() function to make the the plot more readable.
- As usual, display the plot with plt.show() after you've added the customizations.

**Script.py**

```
# Scatter plot
plt.scatter(gdp_cap, life_exp)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')

# Definition of tick_val and tick_lab
tick_val = [1000, 10000, 100000]
tick_lab = ['1k', '10k', '100k']

# Adapt the ticks on the x-axis
plt.xticks(tick_val, tick_lab)

# After customizing, display the plot
plt.show()
```

**Plot**



## Sizes

Right now, the scatter plot is just a cloud of blue dots, indistinguishable from each other. Let's change this. Wouldn't it be nice if the size of the dots corresponds to the population?

To accomplish this, there is a list pop loaded in your workspace. It contains population numbers for each country expressed in millions. You can see that this list is added to the scatter method, as the argument s, for size.

## Instructions

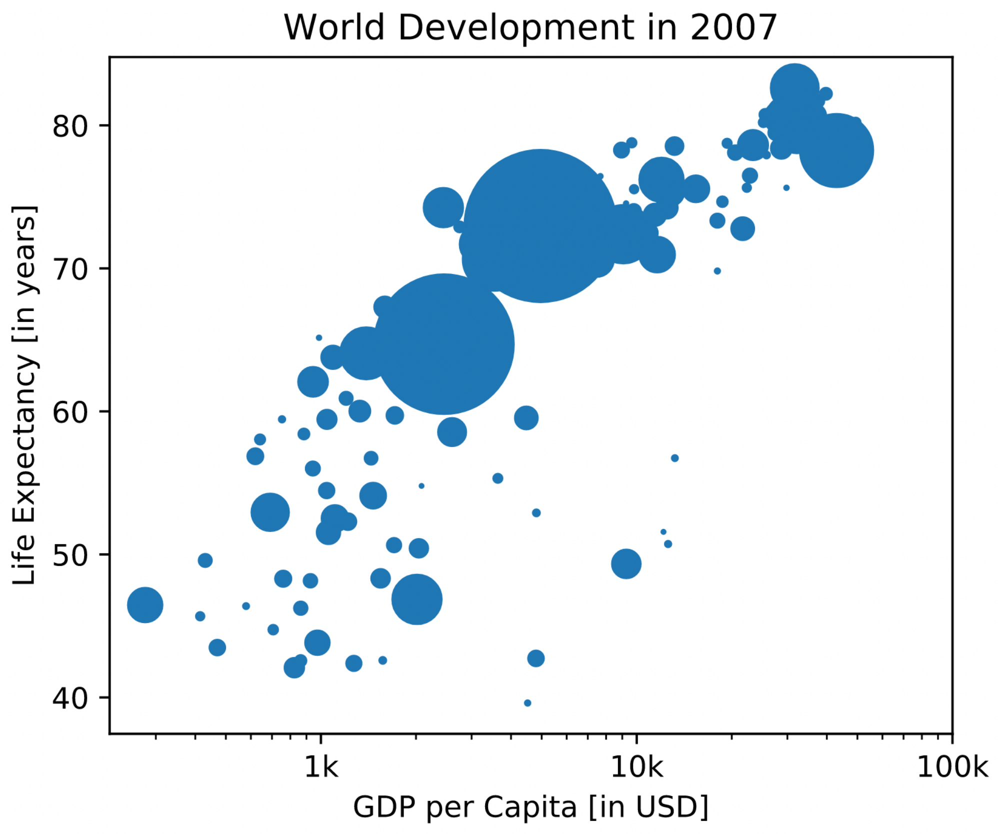
- Run the script to see how the plot changes.
- Looks good, but increasing the size of the bubbles will make things stand out more.
    - Import the numpy package as np.
    - Use np.array() to create a numpy array from the list pop. Call this Numpy array np_pop.
    - Double the values in np_pop setting the value of np_pop equal to np_pop * 2. Because np_pop is a Numpy array, each array element will be doubled.
    - Change the s argument inside plt.scatter() to be np_pop instead of pop.

**Script.py**

```
# Import numpy as np

import numpy as np
```

```
# Store pop as a numpy array: np_pop

np_pop = np.array(pop)


# Double np_pop

np_pop = np_pop*2


# Update: set s argument to np_pop

plt.scatter(gdp_cap, life_exp, s = np_pop)


# Previous customizations

plt.xscale('log')

plt.xlabel('GDP per Capita [in USD]')

plt.ylabel('Life Expectancy [in years]')

plt.title('World Development in 2007')

plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])


# Display the plot

plt.show()
```

```
Plot
```


World Development in 2007
Life Expectancy [in years]
GDP per Capita [in USD]
40
50
60
70
80
1k
10k
100k

## Colors

The code you've written up to now is available in the script on the right.

The next step is making the plot more colorful! To do this, a list col has been created for you. It's a list with a color for each corresponding country, depending on the continent the country is part of.

How did we make the list col you ask? The Gapminder data contains a list continent with the continent each country belongs to. A dictionary is constructed that maps continents onto colors:

```
dict = {

    'Asia':'red',

    'Europe':'green',

    'Africa':'blue',

    'Americas':'yellow',

    'Oceania':'black'

}
```

Instructions

- Add c = col to the arguments of the plt.scatter() function.
- Change the opacity of the bubbles by setting the alpha argument to 0.8 inside plt.scatter(). Alpha can be set from zero to one, where zero is totally transparent, and one is not at all transparent.

Script.py

```
print (gdp_cap)

print (life_exp)

print (col)

# Specify c and alpha inside plt.scatter()

plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2,
c=col, alpha=0.8)



# Previous customizations

plt.xscale('log')

plt.xlabel('GDP per Capita [in USD]')

plt.ylabel('Life Expectancy [in years]')

plt.title('World Development in 2007')

plt.xticks([1000,10000,100000], ['1k','10k','100k'])



# Show the plot

plt.show()
```

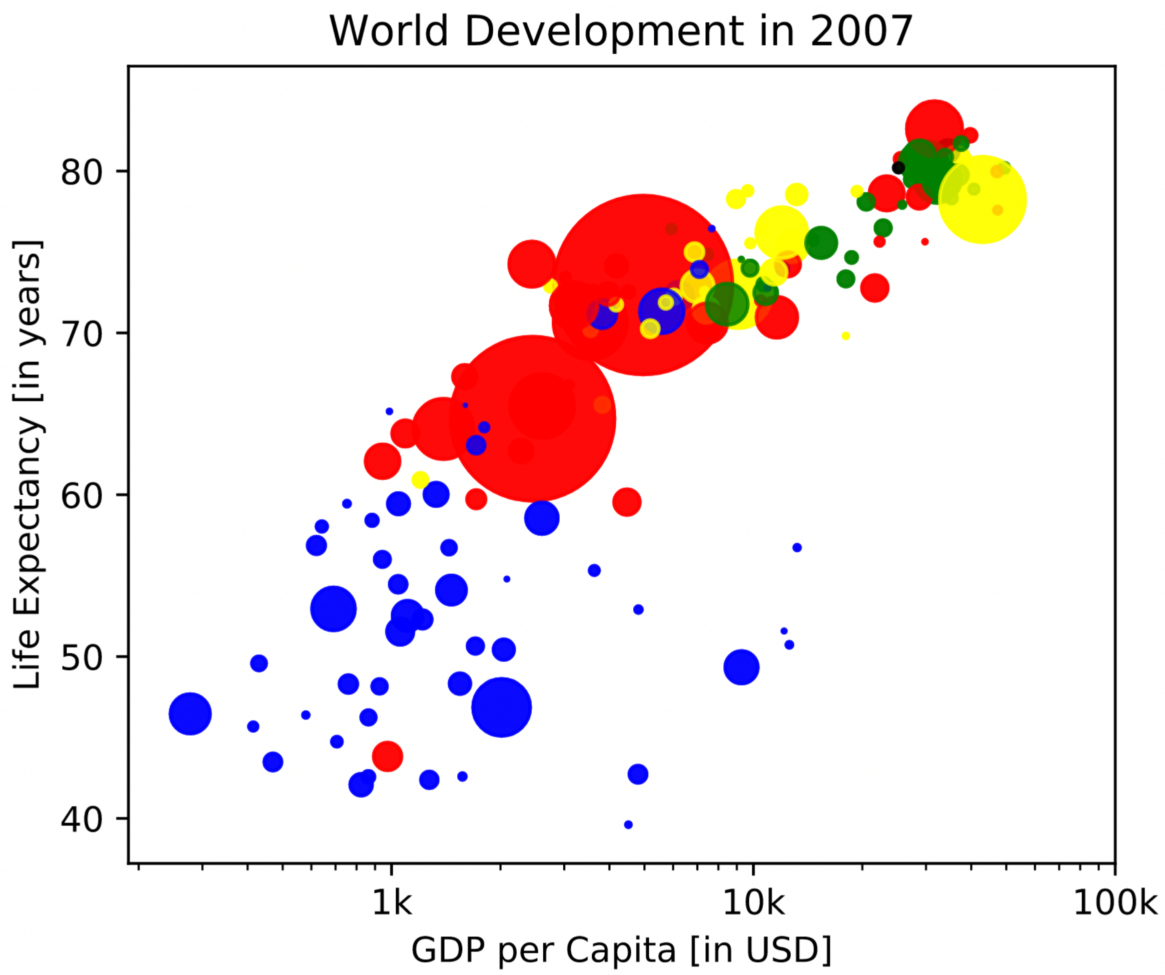
[redacted]

Python Shell

```
# Import numpy as np
import numpy as np


# Store pop as a numpy array: np_pop
np_pop = np.array(pop)

# Double np_pop
np_pop = np_pop*2

# Update: set s argument to np_pop
plt.scatter(gdp_cap, life_exp, s = np_pop)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000, 10000, 100000],['1k', '10k', '100k'])

# Display the plot
plt.show()
print (col)
# Specify c and alpha inside plt.scatter()
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2,
c=col, alpha=0.8)

# Previous customizations
```

```
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Show the plot
plt.show()
['red', 'green', 'blue', 'blue', 'yellow', 'black', 'green',
'red', 'red', 'green', 'blue', 'yellow', 'green', 'blue',
'yellow', 'green', 'blue', 'blue', 'red', 'blue', 'yellow',
'blue', 'blue', 'yellow', 'red', 'yellow', 'blue', 'blue',
'blue', 'yellow', 'blue', 'green', 'yellow', 'green',
'green', 'blue', 'yellow', 'yellow', 'blue', 'yellow',
'blue', 'blue', 'blue', 'green', 'green', 'blue', 'blue',
'green', 'blue', 'green', 'yellow', 'blue', 'blue', 'yellow',
'yellow', 'red', 'green', 'green', 'red', 'red', 'red',
'red', 'green', 'red', 'green', 'yellow', 'red', 'red',
'blue', 'red', 'red', 'red', 'red', 'blue', 'blue', 'blue',
'blue', 'blue', 'red', 'blue', 'blue', 'blue', 'yellow',
'red', 'green', 'blue', 'blue', 'red', 'blue', 'red',
'green', 'black', 'yellow', 'blue', 'blue', 'green', 'red',
'red', 'yellow', 'yellow', 'yellow', 'red', 'green', 'green',
'yellow', 'blue', 'green', 'blue', 'blue', 'red', 'blue',
'green', 'blue', 'red', 'green', 'green', 'blue', 'blue',
'green', 'red', 'blue', 'blue', 'green', 'green', 'red',
'red', 'blue', 'red', 'blue', 'yellow', 'blue', 'green',
'blue', 'green', 'yellow', 'yellow', 'yellow', 'red', 'red',
'red', 'blue', 'blue']
print (gdp_cap)
print (life_exp)
print (col)
# Specify c and alpha inside plt.scatter()
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2,
c=col, alpha=0.8)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Show the plot
plt.show()
[974.5803384, 5937.029525999998, 6223.367465, 4797.231267,
12779.37964, 34435.367439999995, 36126.4927, 29796.04834,
1391.253792, 33692.60508, 1441.284873, 3822.137084,
7446.298803, 12569.85177, 9065.800825, 10680.79282,
1217.032994, 430.0706916, 1713.778686, 2042.09524,
36319.23501, 706.016537, 1704.063724, 13171.63885,
4959.114854, 7006.580419, 986.1478792, 277.5518587,
3632.557798, 9645.06142, 1544.750112, 14619.222719999998,
8948.102923, 22833.30851, 35278.41874, 2082.4815670000007,
6025.374752000015, 6873.262326000001, 5581.180998,
5728.353514, 12154.08975, 641.3695236000002, 690.8055759,
33207.0844, 30470.0167, 13206.48452, 752.7497265,
32170.37442, 1327.60891, 27538.41188, 5186.050003,
942.6542111, 579.2317429999998, 1201.637154,
3548.3308460000007, 39724.97867, 18008.94444, 36180.78919,
2452.210407, 3540.651564, 11605.71449, 4471.061906,
40675.99635, 25523.2771, 28569.7197, 7320.8802620000015,
31656.06806, 4519.461171, 1463.249282, 1593.06548,
23348.139730000006, 47306.98978, 10461.05868, 1569.331442,
414.5073415, 12057.49928, 1044.770126, 759.3499101,
12451.6558, 1042.581557, 1803.151496, 10956.99112,
11977.57496, 3095.7722710000007, 9253.896111, 3820.17523,
823.6856205, 944.0, 4811.060429, 1091.359778, 36797.93332,
25185.00911, 2749.320965, 619.6768923999998, 2013.977305,
49357.19017, 22316.19287, 2605.94758, 9809.185636,
4172.838464, 7408.905561, 3190.481016, 15389.924680000002,
20509.64777, 19328.70901, 7670.122558, 10808.47561,
863.0884639000002, 1598.435089, 21654.83194, 1712.472136,
9786.534714, 862.5407561000002, 47143.17964, 18678.31435,
25768.25759, 926.1410683, 9269.657808, 28821.0637,
3970.095407, 2602.394995, 4513.480643, 33859.74835,
37506.41907, 4184.548089, 28718.27684, 1107.482182,
7458.396326999998, 882.9699437999999, 18008.50924,
7092.923025, 8458.276384, 1056.380121, 33203.26128,
42951.65309, 10611.46299, 11415.80569, 2441.576404,
3025.349798, 2280.769906, 1271.211593, 469.70929810000007]


[43.828, 76.423, 72.301, 42.731, 75.32, 81.235, 79.829,
75.635, 64.062, 79.441, 56.728, 65.554, 74.852, 50.728,
72.39, 73.005, 52.295, 49.58, 59.723, 50.43, 80.653,
44.74100000000001, 50.651, 78.553, 72.961, 72.889, 65.152,
46.462, 55.322, 78.782, 48.328, 75.748, 78.273, 76.486,
78.332, 54.791, 72.235, 74.994, 71.33800000000002, 71.878,
51.57899999999999, 58.04, 52.947, 79.313, 80.657, 56.735,
59.448, 79.406, 60.022, 79.483, 70.259, 56.007,
46.38800000000001, 60.916, 70.19800000000001, 82.208,
73.33800000000002, 81.757, 64.69800000000001, 70.65, 70.964,
59.545, 78.885, 80.745, 80.546, 72.567, 82.603, 72.535,
54.11, 67.297, 78.623, 77.58800000000002, 71.993, 42.592,
45.678, 73.952, 59.44300000000001, 48.303, 74.241, 54.467,
64.164, 72.801, 76.195, 66.803, 74.543, 71.164, 42.082,
62.069, 52.90600000000001, 63.785, 79.762, 80.204, 72.899,
56.867, 46.859, 80.196, 75.64, 65.483, 75.53699999999998,
71.752, 71.421, 71.688, 75.563, 78.098, 78.74600000000002,
76.442, 72.476, 46.242, 65.528, 72.777, 63.062, 74.002,
42.56800000000001, 79.972, 74.663, 77.926, 48.159, 49.339,
80.941, 72.396, 58.556, 39.613, 80.884, 81.70100000000002,
74.143, 78.4, 52.517, 70.616, 58.42, 69.819, 73.923, 71.777,
51.542, 79.425, 78.242, 76.384, 73.747, 74.249, 73.422,
62.698, 42.38399999999999, 43.487]

['red', 'green', 'blue', 'blue', 'yellow', 'black', 'green',
'red', 'red', 'green', 'blue', 'yellow', 'green', 'blue',
'yellow', 'green', 'blue', 'blue', 'red', 'blue', 'yellow',
'blue', 'blue', 'yellow', 'red', 'yellow', 'blue', 'blue',
'blue', 'yellow', 'blue', 'green', 'yellow', 'green',
'green', 'blue', 'yellow', 'yellow', 'blue', 'yellow',
'blue', 'blue', 'blue', 'green', 'green', 'blue', 'blue',
'green', 'blue', 'green', 'yellow', 'blue', 'blue', 'yellow',
'yellow', 'red', 'green', 'green', 'red', 'red', 'red',
'red', 'green', 'red', 'green', 'yellow', 'red', 'red',
'blue', 'red', 'red', 'red', 'red', 'blue', 'blue', 'blue',
'blue', 'blue', 'red', 'blue', 'blue', 'blue', 'yellow',
'red', 'green', 'blue', 'blue', 'red', 'blue', 'red',
'green', 'black', 'yellow', 'blue', 'blue', 'green', 'red',
'red', 'yellow', 'yellow', 'yellow', 'red', 'green', 'green',
'yellow', 'blue', 'green', 'blue', 'blue', 'red', 'blue',
'green', 'blue', 'red', 'green', 'green', 'blue', 'blue',
'green', 'red', 'blue', 'blue', 'green', 'green', 'red',
'red', 'blue', 'red', 'blue', 'yellow', 'blue', 'green',
'blue', 'green', 'yellow', 'yellow', 'yellow', 'red', 'red',
'red', 'blue', 'blue']
```

**Plot**



## Additional Customizations

If you have another look at the script, under # Additional Customizations, you'll see that there are two [plt.text()](plt.text()) functions now. They add the words "India" and "China" in the plot.

Python Shell

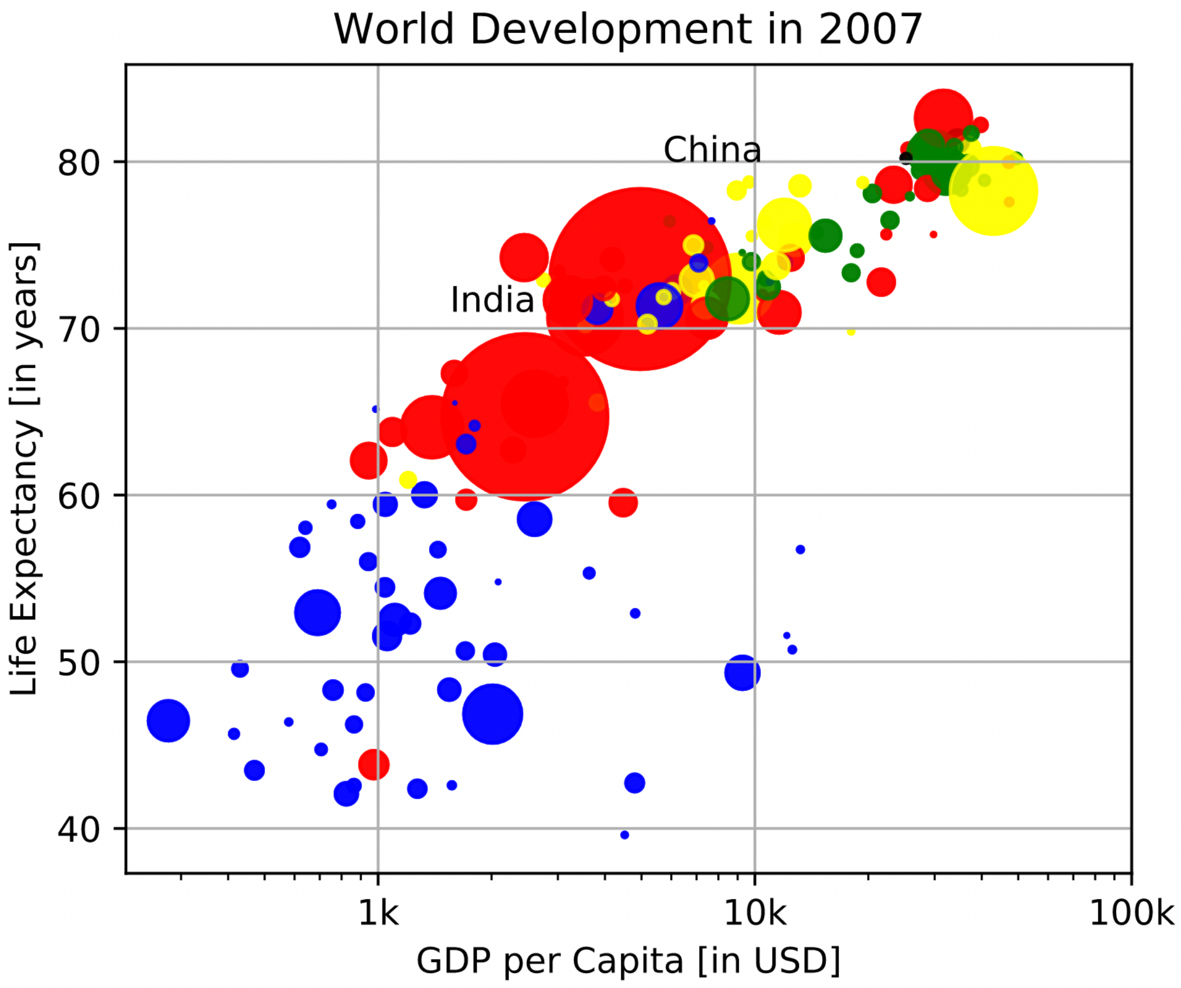
```
# Numpy Array (Double Size)
np_pop = np.array(pop)
np_pop = np_pop*2
print (np_pop)

# Scatter plot
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2,
c = col, alpha = 0.8)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Additional customizations
plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')

# Add grid() call
plt.grid(True)

# Show the plot
plt.show()
[6.37798460e+01 7.20104600e+00 6.66664320e+01 2.48409520e+01
 8.06038540e+01 4.08683520e+01 1.63995660e+01 1.41714600e+00
 3.00896678e+02 2.07844520e+01 1.61566280e+01 1.82383040e+01
 9.10439600e+00 3.27826200e+00 3.80021294e+02 1.46457160e+01
 2.86524060e+01 1.67810100e+01 2.82637160e+01 3.53925860e+01
 6.67802820e+01 8.73807600e+00 2.04776140e+01 3.25694820e+01
 2.63736619e+03 8.84551000e+01 1.42192000e+00 1.29213518e+02
 7.60122000e+00 8.26776800e+00 3.60268180e+01 8.98662400e+00
 2.28339740e+01 2.04574880e+01 1.09362400e+01 9.92748000e-01
 1.86392440e+01 2.75113600e+01 1.60529086e+02 1.38793760e+01
 1.10240200e+00 9.81317000e+00 1.53023774e+02 1.04769200e+01
 1.22167832e+02 2.90973400e+00 3.37671800e+00 1.64801992e+02
 4.57466760e+01 2.14125800e+01 2.51458560e+01 1.98956280e+01
 2.94408200e+00 1.70056280e+01 1.49675260e+01 1.39608240e+01
 1.99122160e+01 6.03862000e-01 2.22079266e+03 4.47094000e+02
 1.38907140e+02 5.49992760e+01 8.21817200e+00 1.28533580e+01
 1.16295466e+02 5.56026400e+00 2.54935944e+02 1.21063860e+01
 7.12203540e+01 4.66034500e+01 9.80895800e+01 5.01111800e+00
 7.84255600e+00 4.02529800e+00 6.38788400e+00 1.20738280e+01
 3.83353080e+01 2.66541580e+01 4.96425720e+01 2.40635900e+01
 6.54013000e+00 2.50176400e+00 2.17401782e+02 5.74825400e+00
 1.36947200e+00 6.75143500e+01 3.99033120e+01 9.55239600e+01
 4.11016000e+00 5.78035800e+01 3.31412260e+01 8.23154200e+00
 1.13507120e+01 2.57897300e+01 2.70062328e+02 9.25585200e+00
 6.40979400e+00 3.38541234e+02 6.48434600e+00 1.33342940e+01
 5.73495140e+01 1.82154574e+02 7.70364820e+01 2.12856720e+01
 7.88498200e+00 1.59618800e+00 4.45521120e+01 1.77211760e+01
 3.99158000e-01 5.52020760e+01 2.45349860e+01 2.03005300e+01
 1.22891240e+01 9.10601800e+00 1.08950040e+01 4.01849000e+00
 1.82375460e+01 8.79956560e+01 8.08963820e+01 4.07564780e+01
 8.45858580e+01 2.26613200e+00 1.80621760e+01 1.51093220e+01
 3.86294940e+01 4.63485880e+01 7.62792800e+01 1.30136298e+02
 1.14031580e+01 2.11321600e+00 2.05523160e+01 1.42317294e+02
 5.83407960e+01 1.21552476e+02 6.02279894e+02 6.89499200e+00
 5.21693240e+01 1.70524712e+02 8.03666400e+00 4.44234860e+01
 2.34920700e+01 2.46222860e+01]
```

Script.py

```python
# Numpy Array (Double Size)
np_pop = np.array(pop)
np_pop = np_pop*2
print (np_pop)
```

```python
# Scatter plot
plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c
= col, alpha = 0.8)

# Previous customizations
plt.xscale('log')
plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]')
plt.title('World Development in 2007')
plt.xticks([1000,10000,100000], ['1k','10k','100k'])

# Additional customizations
plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')

# Add grid() call
plt.grid(True)

# Show the plot
plt.show()
```

**Plot**


World Development in 2007
China
India
80
70
60
50
40
1k
10k
100k
GDP per Capita [in USD]
Life Expectancy [in years]

## Interpretation

If you have a look at your colorful plot, it's clear that people live longer in countries with a higher GDP per capita. No high income countries have really short life expectancy, and no low income countries have very long life expectancy. Still, there is a huge difference in life expectancy between countries on the same income level. Most people live in middle income countries where difference in lifespan is huge between countries; depending on how income is distributed and how it is used.

What can you say about the plot?

```
np_pop = np.array(pop) np_pop = np_pop*2 print (np_pop)
```

```
# Scatter plot plt.scatter(x = gdp_cap, y = life_exp, s =
np.array(pop) * 2, c = col, alpha = 0.8)
```

```
# Previous customizations
```

```
plt.xscale('log') plt.xlabel('GDP per Capita [in USD]')
plt.ylabel('Life Expectancy [in years]') plt.title('World
Development in 2007') plt.xticks([1000,10000,100000],
['1k','10k','100k'])
```

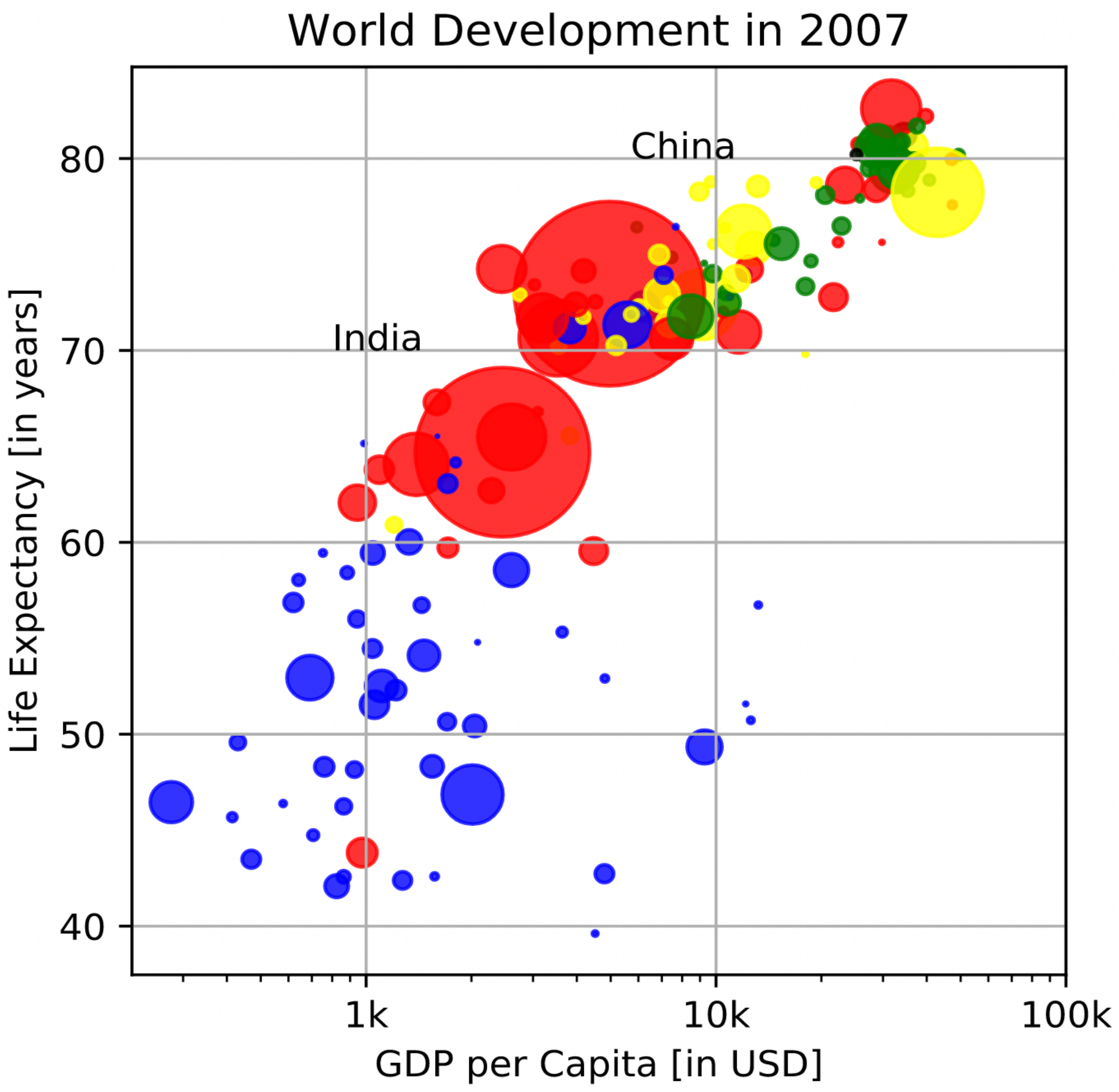
```
# Additional customizations plt.text(1550, 71, 'India')
plt.text(5700, 80, 'China')
```

```
# Add grid() call
```

```
plt.grid(True)
```

```
# Show the plot
```

```
plt.show()
```


World Development in 2007
China
India
80
70
60
50
40
1k
10k
100k
Life Expectancy [in years]
GDP per Capita [in USD]