Pyplot Formatting

Democratizing Data Science Learning

DPhi

Learning Objectives

Plotting consecutive plots

Making the plots descriptive

Formatting the plots

Changing the size of the plots

Multiple Plots in one figure

Pyplot Formatting

Tired of those plain, boring graphs? Want them to convey more information at once?
This is the perfect module for you then!

We'll be working on the same dataset i.e on the Standard Metropolitan Data

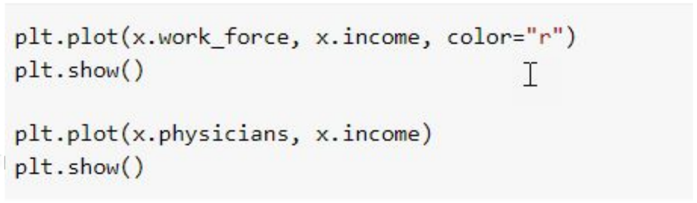
DPhi

Plotting consecutive plots using Matplotlib

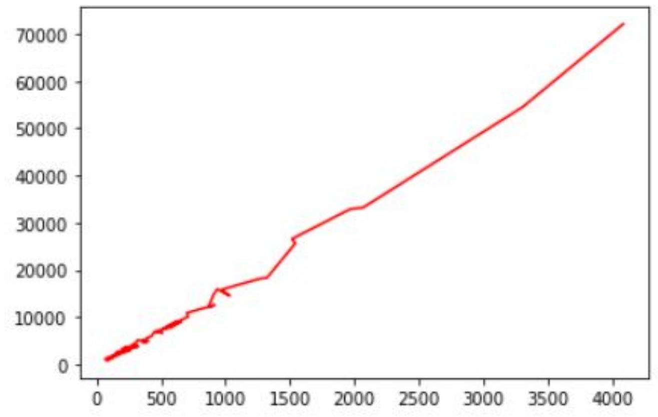
Plotting data on multiple consecutive figures can be done by calling the corresponding graphing functions and displaying each figure consecutively:

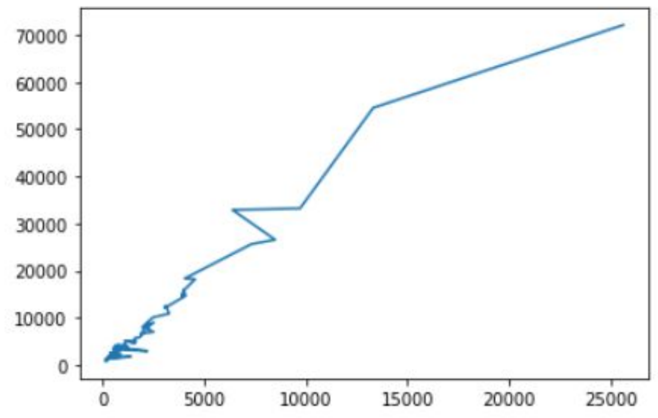
First Plot

Second Plot



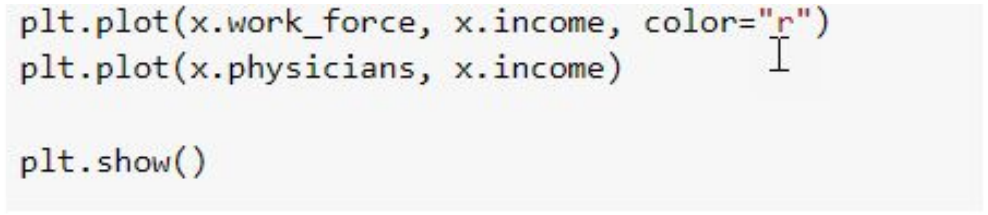
```
plt.plot(x.work_force, x.income, color="r")
plt.show()

plt.plot(x.physicians, x.income)
plt.show()
```



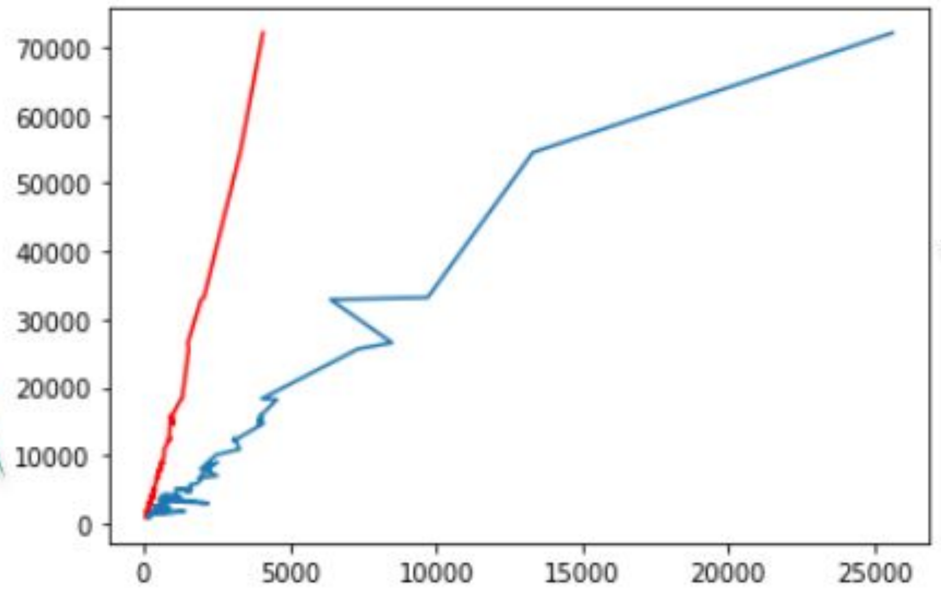


Combining the 2 plots into 1

What happens if you don't use plt.show() after the first figure? Both variables will be plotted in the same figure:



```
plt.plot(x.work_force, x.income, color="r")
plt.plot(x.physicians, x.income)

plt.show()
```



Both share the same axes

Instead of creating a separate image for plot2, it displays both the plots in the same figure

Why should we make plots descriptive?

The plots that we created in the previous module do the job but do you think any person who looks at those will be able to understand what they represent?

It's a good practice to make all the plots descriptive and easy to understand using Title, Labels and Legends.

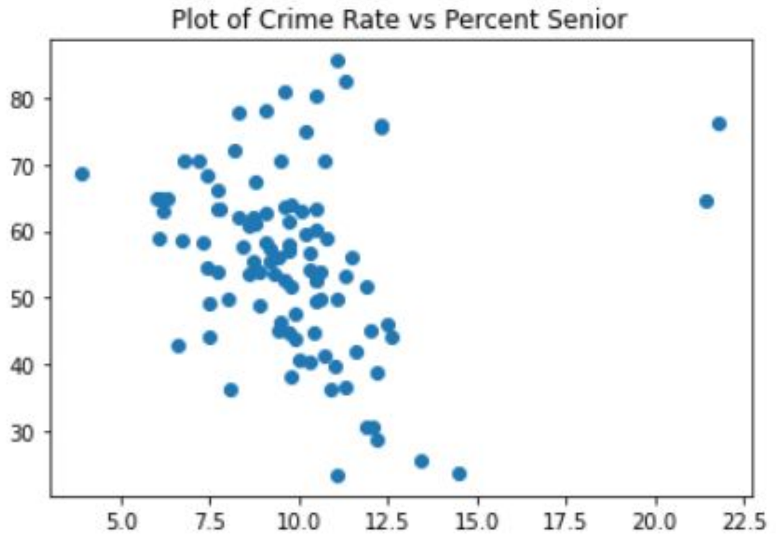
DPhi

1. Adding a Title

Your graph isn't complete without a title that summarizes what the graph itself depicts. The title is usually placed in the center, either above or below the graph. The proper form for a graph title is "y-axis variable vs. x-axis variable."

Adding title to a plot is very straightforward. It can be done with the title function as follows:

```
plt.scatter(x.percent_senior, x.crime_rate)

plt.title('Plot of Crime Rate vs Percent Senior') # Adding a title to the plot
plt.show()
```

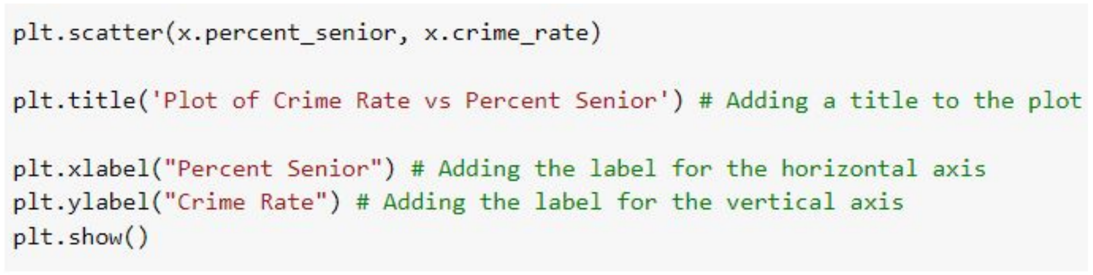


2. Adding Labels

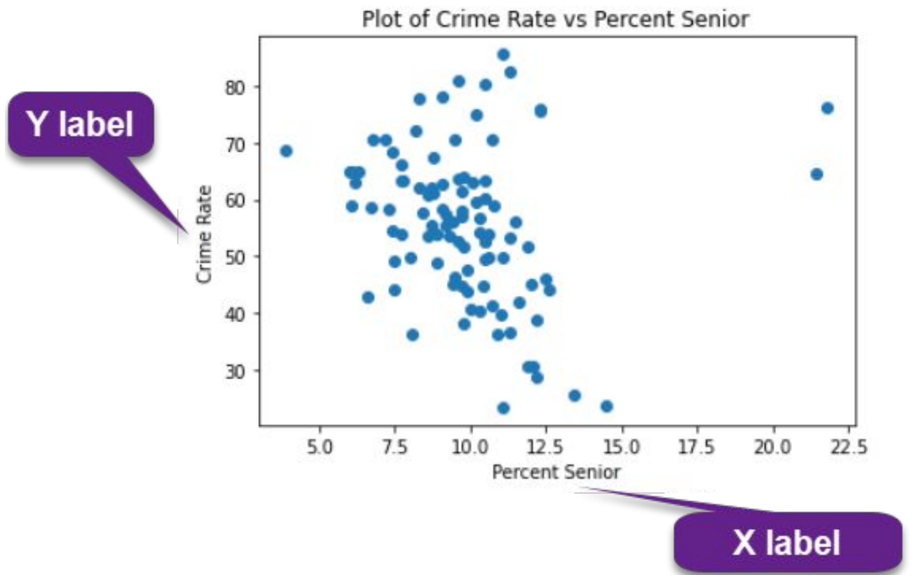
Axis labels are the variable names or descriptions that represent a particular axis. The axis label usually appears below or to the left of the axis.

To add a label to the X axis, xlabel function is used.

Similarly, to add a label to the Y axis, ylabel function is used.



```
plt.scatter(x.percent_senior, x.crime_rate)

plt.title('Plot of Crime Rate vs Percent Senior') # Adding a title to the plot

plt.xlabel("Percent Senior") # Adding the label for the horizontal axis
plt.ylabel("Crime Rate") # Adding the label for the vertical axis
plt.show()
```



DPhi

3. Adding a legend

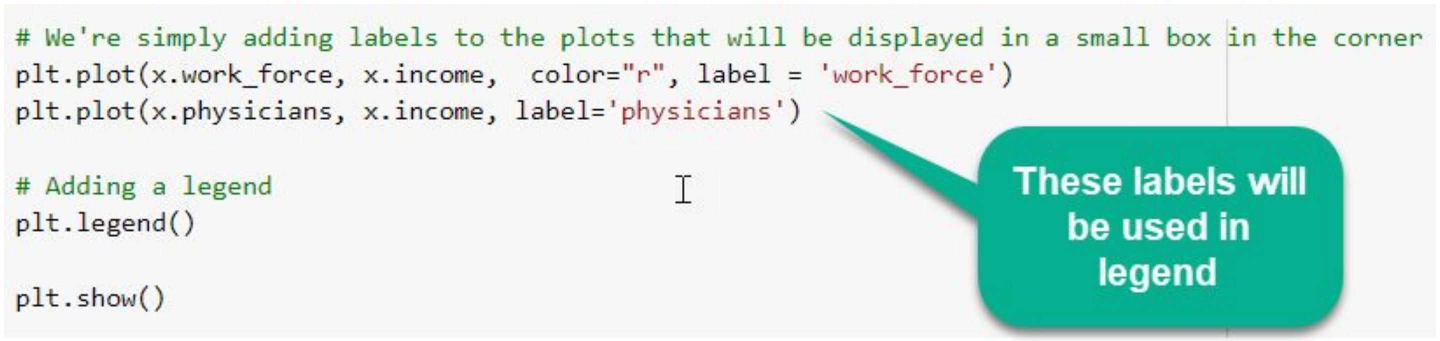
Now that you've added what the graph, X and Y axis represent, you're already halfway through the process of making your graphs intuitive.

You plotted the income of both work_force and physicians on the same graph above. How would a layman know which plot represents the work_force and which physicians? That's the problem solved by legend.

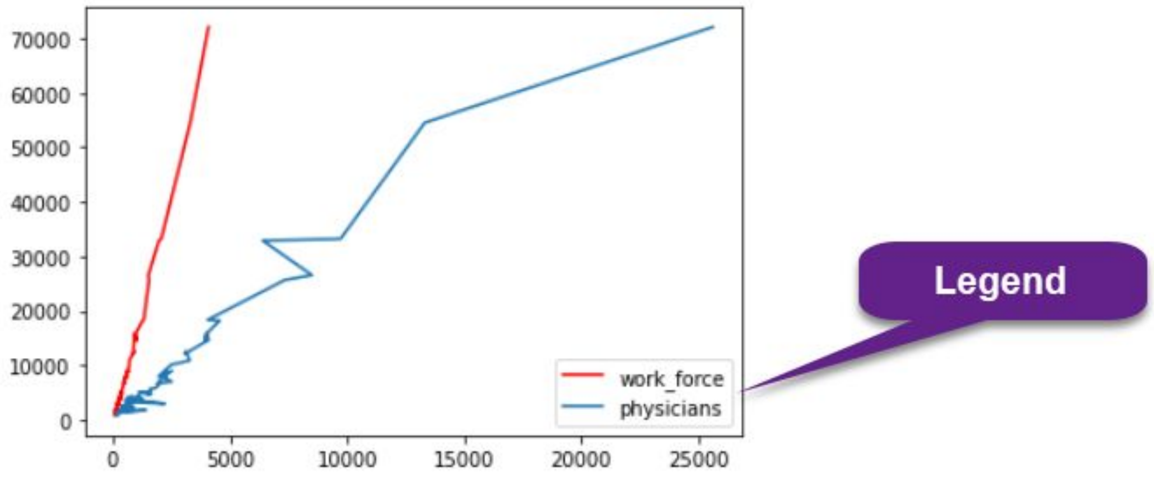
DPhi

Adding a legend

A legend is an area describing the elements of the graph. In the matplotlib library, there's a function called legend which is used to place a legend on the axes, as follows:



```
# We're simply adding labels to the plots that will be displayed in a small box in the corner
plt.plot(x.work_force, x.income,  color="r", label = 'work_force')
plt.plot(x.physicians, x.income, label='physicians')

# Adding a legend
plt.legend()

plt.show()
```



Formatting Plots

The first adjustment you might wish to make to a plot is to control the line colors and styles.

Look at all the above graphs you've created till now. What is the color and style of the plots where it is not explicitly specified?
All **blue solid lines.**
Blue solid line is thus the default formatting style.

The plt.plot() function takes additional arguments that can be used to specify the formatting.

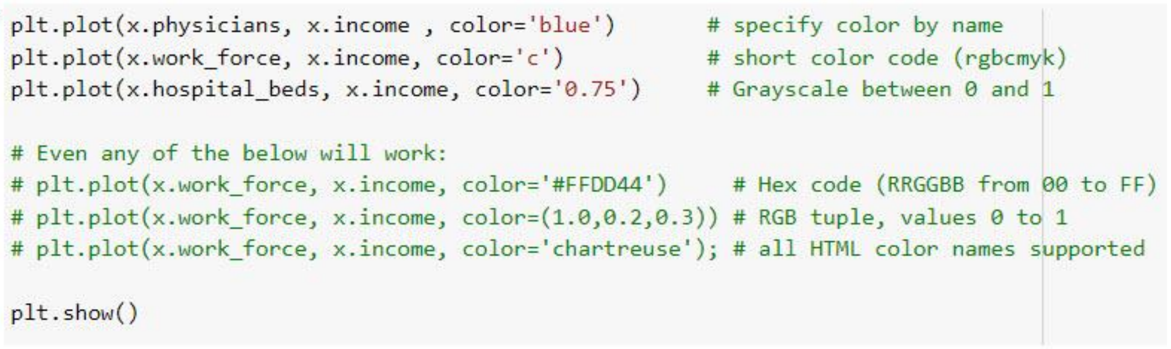
You've already specified the colour of a few plots before using the 'color' argument. Colours can be provided in a number of ways apart from that.

DPhi

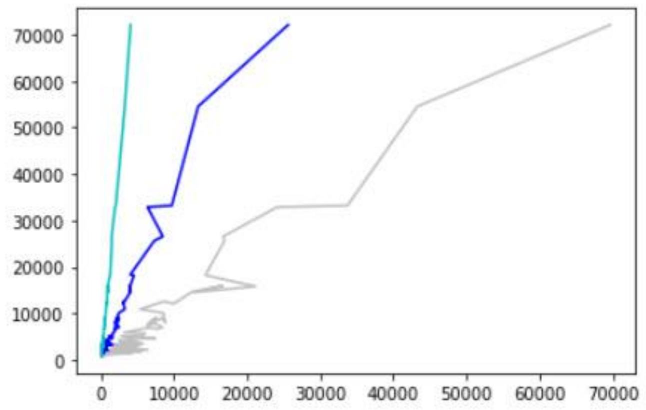
1. Changing Colours

To adjust the color, you can use the color keyword, which accepts a string argument representing virtually any imaginable color. The color can be specified in a variety of ways:

Multiple methods of specifying colors



```
plt.plot(x.physicians, x.income , color='blue')       # specify color by name
plt.plot(x.work_force, x.income, color='c')           # short color code (rgbcmyk)
plt.plot(x.hospital_beds, x.income, color='0.75')     # Grayscale between 0 and 1

# Even any of the below will work:
# plt.plot(x.work_force, x.income, color='#FFDD44')     # Hex code (RRGGBB from 00 to FF)
# plt.plot(x.work_force, x.income, color=(1.0,0.2,0.3)) # RGB tuple, values 0 to 1
# plt.plot(x.work_force, x.income, color='chartreuse'); # all HTML color names supported

plt.show()
```

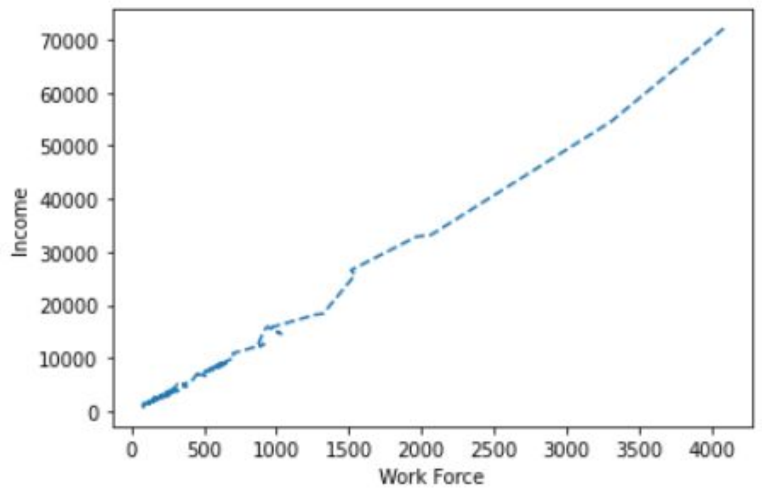


If no color is specified, Matplotlib will automatically cycle through a set of default colors for multiple lines

2. Changing Line Styles

Similarly, the line style can be adjusted using the linestyle keyword. You can make the lines dashed, dotted or a combination of both.

```
plt.plot(x.work_force, x.income , linestyle='dashed')
# plt.plot(x.work_force, x.income, linestyle='--') # shortcut for dashed line

## Other styles:
# plt.plot(x.work_force, x.income, linestyle='solid') # default style
# plt.plot(x.work_force, x.income, linestyle='-')  # shortcut for solid line

# plt.plot(x.work_force, x.income, linestyle='dashdot')
# plt.plot(x.work_force, x.income, linestyle='-.') # shortcut for dashdot line

# plt.plot(x.work_force, x.income, linestyle='dotted')
# plt.plot(x.work_force, x.income, linestyle=':');  # shortcut for dotted line
plt.xlabel("Work Force")
plt.ylabel("Income")
plt.show()
```

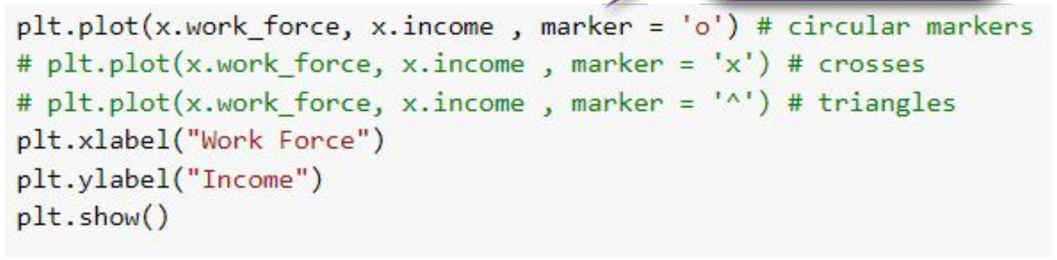


3. Adding Markers

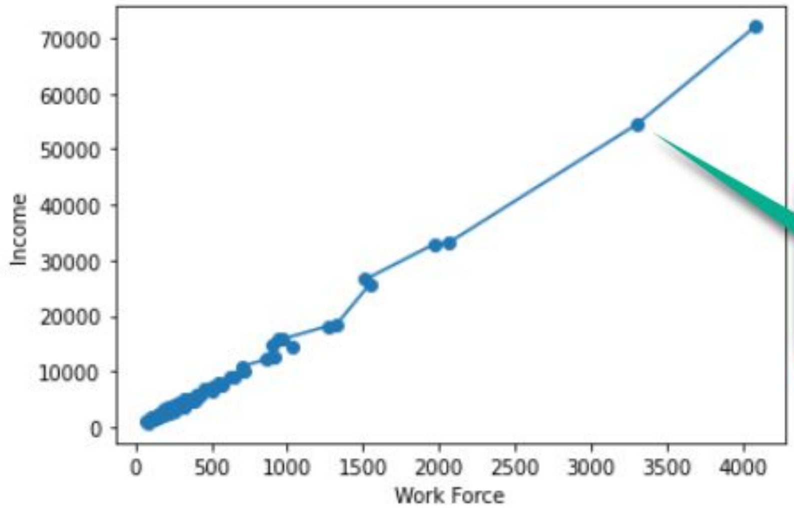
So you've plotted lines that pass through the points in the dataset. But we can't really see those points in the lines above, can we?

Adding markers to a line plot can be a useful way to distinguish multiple lines or to highlight particular data points.

Specifying the marker to be used



```
plt.plot(x.work_force, x.income , marker = 'o') # circular markers
# plt.plot(x.work_force, x.income , marker = 'x') # crosses
# plt.plot(x.work_force, x.income , marker = '^') # triangles
plt.xlabel("Work Force")
plt.ylabel("Income")
plt.show()
```

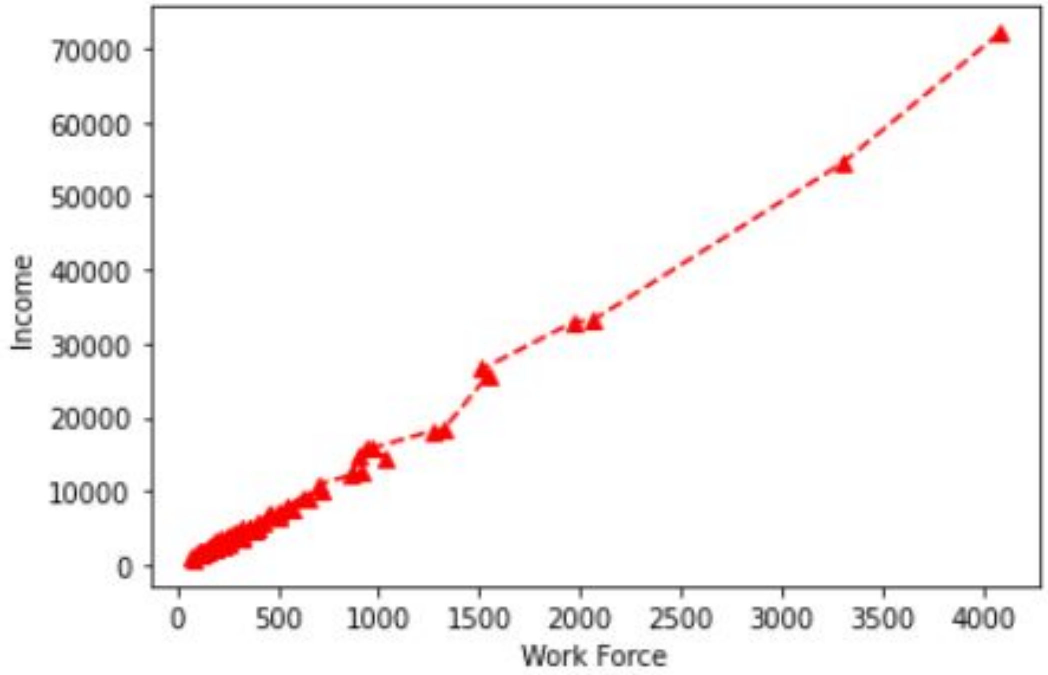


Circular markers represent each data point

Combining the 3 formatting options

Let's plot a red dashed line with triangular markers!

```
plt.plot(x.work_force, x.income, linestyle='--', marker='^', color='r')
plt.xlabel("Work Force")
plt.ylabel("Income")
plt.show()
```



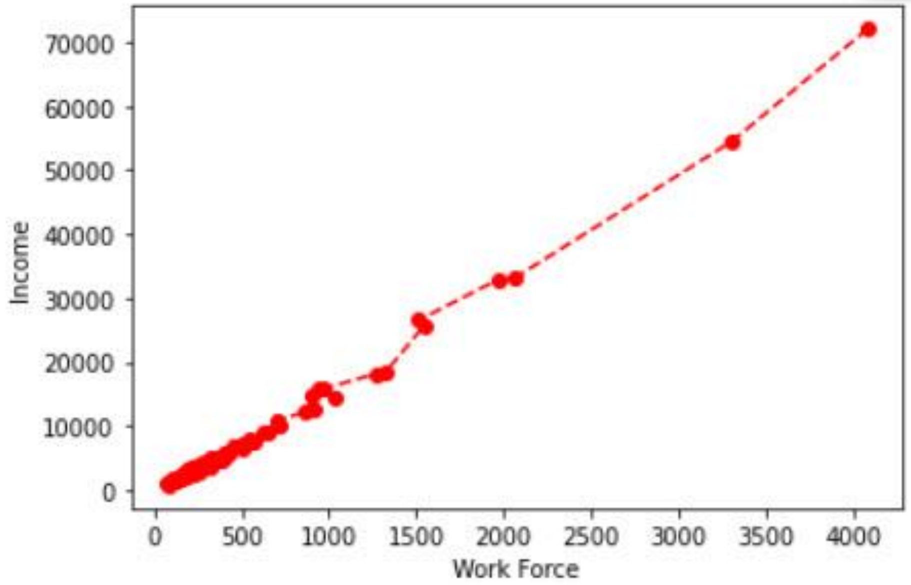
Shortcut for above

For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The letters and symbols of the format string are from MATLAB, and you concatenate a color string with a line style string. The default format string is 'b-', which is a solid blue line.

These linestyle and color codes can be combined into a single non-keyword argument to the plt.plot() function:

```
plt.plot(x.work_force, x.income, '--ro')  # ro = red circles
plt.xlabel("Work Force")
plt.ylabel("Income")
plt.show()
```

-- : Dashed line
r : red color
o : circular marker



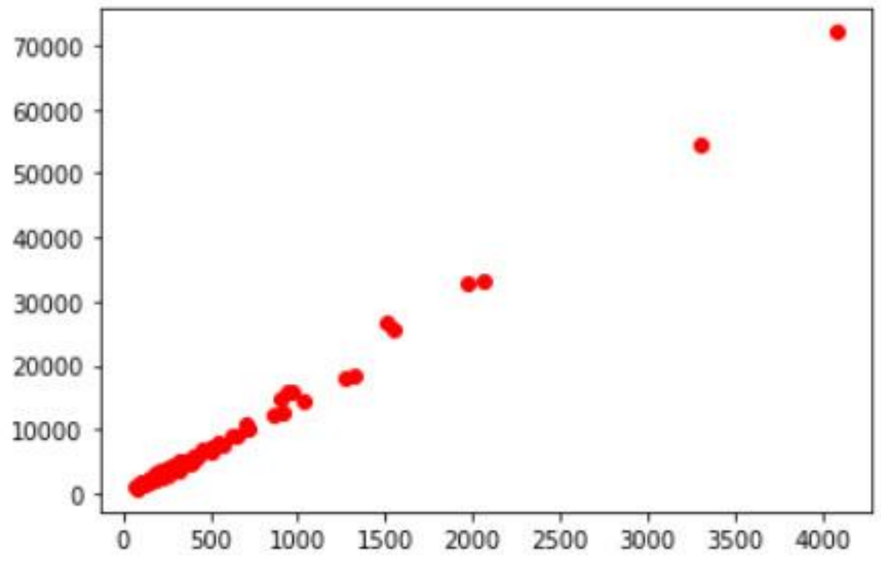
DPhi

Scatter Plot with plot function

It turns out that the plot function can produce scatter plots as well. Just don't mention any linestyle.

Didn't provide any linestyle

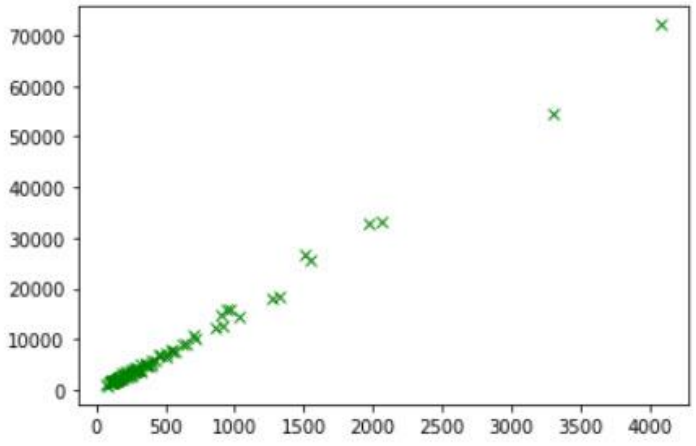
```
plt.plot(x.work_force, x.income, 'ro') # ro = red circles
plt.show()
```



DPhi

More examples

```
plt.plot(x.work_force, x.income, "gx") # gx = green x
plt.show()
```



There are plenty of other options. You can try the following:

```
plt.plot(x.work_force, x.income, "go") # green circles
plt.plot(x.work_force, x.income, "g^") # green traingles
plt.plot(x.work_force, x.income, "ro") # red circles
plt.plot(x.work_force, x.income, "rx") # red x symbol
plt.plot(x.work_force, x.income, "b^") # blue ^ symbol
plt.plot(x.work_force, x.income, "go--", linewidth=3) # green circles and dashed lines of width 3.
```

These single-character color codes reflect the standard abbreviations in the RGB (Red/Green/Blue) and CMYK (Cyan/Magenta/Yellow/blacK) color systems, commonly used for digital color graphics.

Changing Plot Size

You may feel that the output images are not of appropriate size and might prefer to enlarge or sometimes downsize them.

In such a case, the size of the figure that contains the graph can be changed with the figsize argument as follows:

```
plt.figure(figsize=(new_width_pixels, new_height_pixels))
```

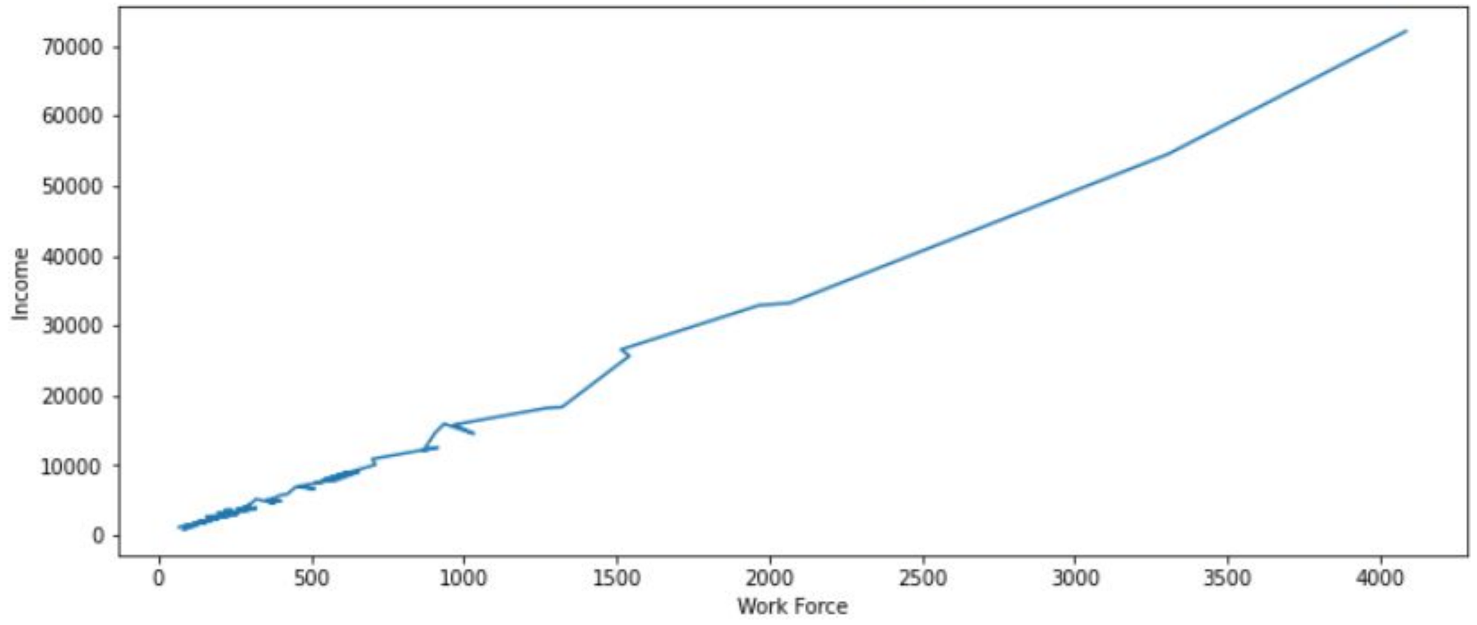
DPhi

Changing Plot Size

Let's look at this example:

```
plt.figure(figsize=(12,5)) # 12x5 plot

plt.plot(x.work_force, x.income)
plt.xlabel("Work Force")
plt.ylabel("Income")
plt.show()
```

We got a 12x5 plot!



DPhi

Multiple Plots in 1 Figure

We can make multiple graphics in one figure. This goes very well for comparing charts or for sharing data from several types of charts easily with a single image.

The `.subplot()` method is used to add multiple plots in one figure. It takes three arguments:

- nrows: number of rows in the figure
- ncols: number of columns in the figure
- index: index of the plot

Let's see how variables are plotted with different row and column configurations in the figures.

DPhi

Multiple Plots in 1 Figure

The function subplot creates a figure and a set of subplots. It is a wrapper function to make it convenient to create common layouts of subplots, including the enclosing figure object, in a single call.

You can even give a Title to your subplots using the suptitle method! No, the 'p' is not a typing mistake.

The subplot function returns a figure and an Axes object or an array of Axes objects.

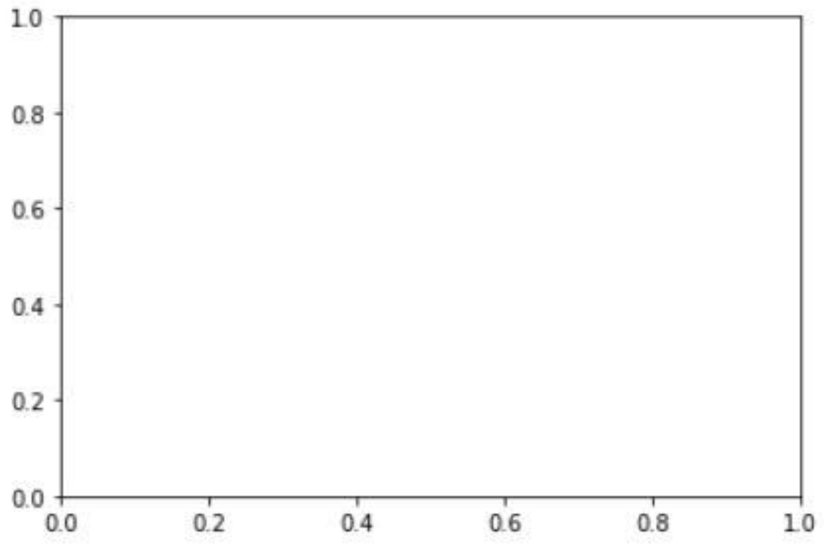
DPhi

Changing Plot Size

If we call the subplot function without any parameters - like we do in the following example - a Figure object and one Axes object will be returned:

```
fig, ax = plt.subplots()
print(fig, ax)
```

```
Figure(432x288) AxesSubplot(0.125,0.125;0.775x0.755)
```



1 row and 2 columns - Method 1

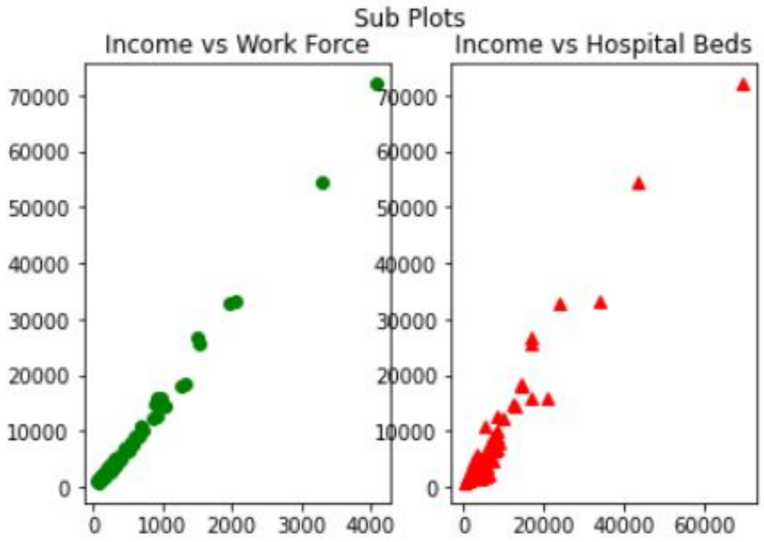
Providing the no. of rows, no, of columns and index of the subplot

```
plt.subplot(1,2,1)  # row, column, index
plt.plot(x.work_force, x.income, "go")
plt.title("Income vs Work Force")

plt.subplot(1,2,2)  # row, column, index

plt.plot(x.hospital_beds, x.income, "r^")
plt.title("Income vs Hospital Beds")

plt.suptitle("Sub Plots") # Add a centered title to the figure.
plt.show()
```

index = 1 means this subplot will be displayed first

index = 2 means this subplot will be displayed at the 2nd position

Giving a common centered title to the subplots



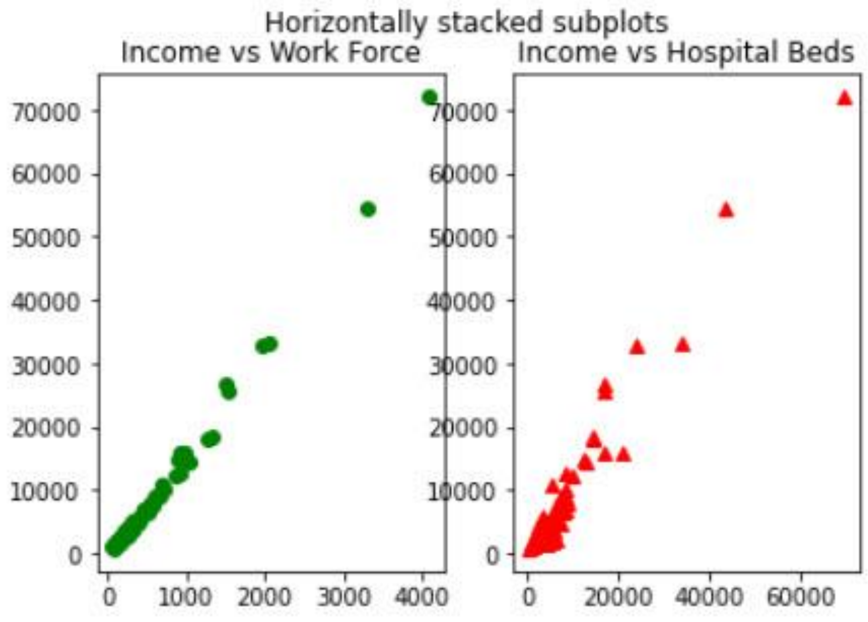
DPhi

1 row and 2 columns - Method 2

Using figure and axis to create subplots

Setting individual title for each subplot

```
fig, (ax1,ax2) = plt.subplots(1,2)  # row, column

fig.suptitle('Horizontally stacked subplots')
ax1.plot(x.work_force, x.income, "go")
ax1.set_title("Income vs Work Force")

ax2.plot(x.hospital_beds, x.income, "r^")
ax2.set_title("Income vs Hospital Beds")

plt.show()
```



Look at the next slide to remove the overlap of Y values.

DPhi

Overlapping Values

PS. Aren't those overlapping axis values annoying? Matplotlib has a solution for that too.

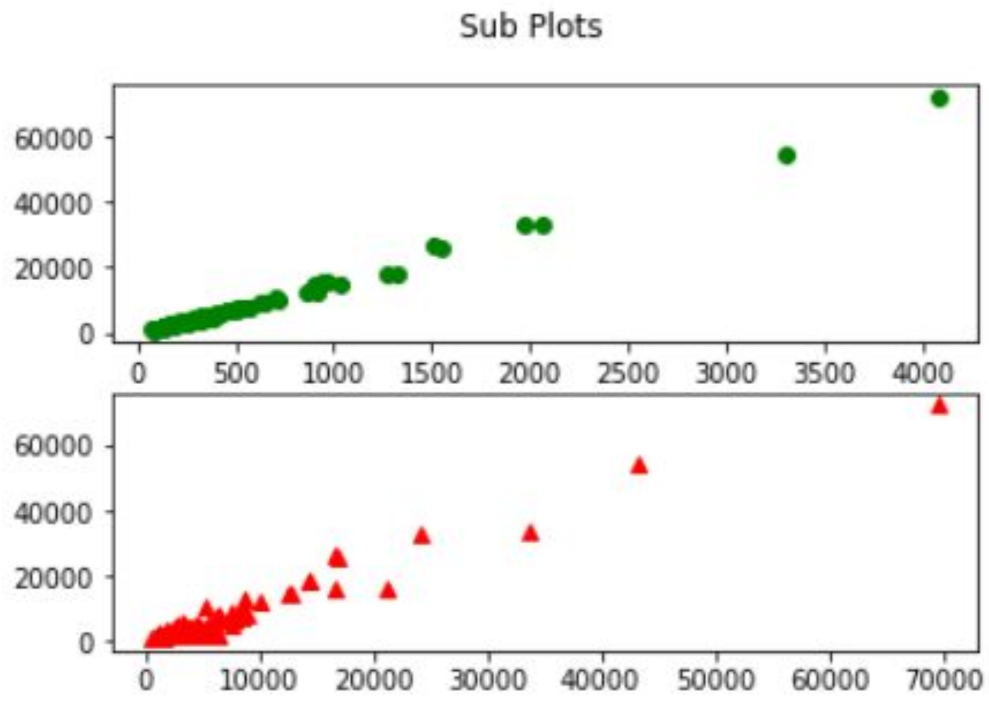
Simply put `sharey=True` inside the subplot function. The two subplots will share the y axis values then.

Try it out!

DPhi

2 rows and 1 column

```
plt.subplot(2,1,1) # row, column, index
plt.plot(x.work_force, x.income, "go")

plt.subplot(2,1,2) # row, column, index
plt.plot(x.hospital_beds, x.income, "r^")

plt.suptitle("Sub Plots")
plt.show()
```

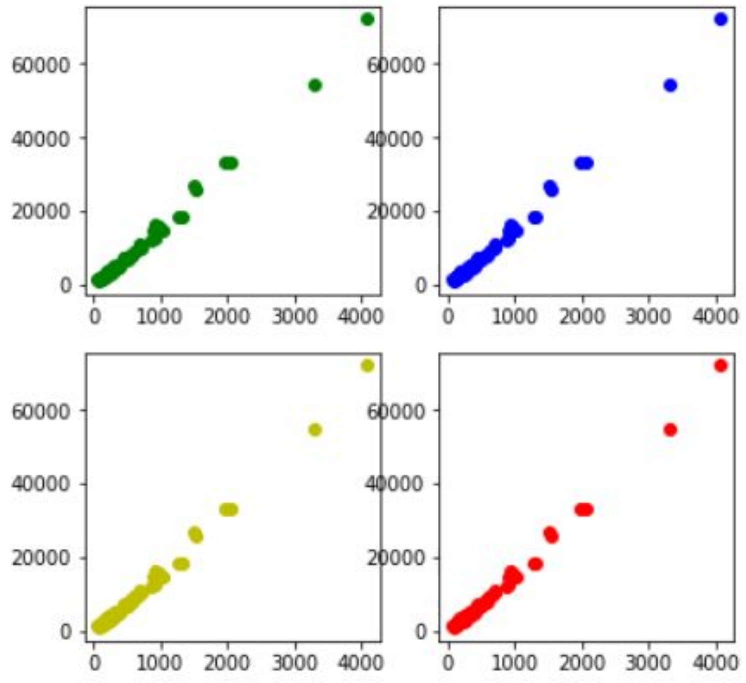


DPhi

2 rows and 2 columns

Think of this as a 2X2 grid. You can access the different positions of the axis and plot your graphs there as follows:

```
fig, ax = plt.subplots(nrows=2, ncols=2, figsize=(6,6)) #creating a grid of 2 rows, 2 columns and 6x6 figure size
ax[0,0].plot(x.work_force, x.income, "go") # The top-left axes
ax[0,1].plot(x.work_force, x.income, "bo") # The top-right axes
ax[1,0].plot(x.work_force, x.income, "yo") # The bottom-left axes
ax[1,1].plot(x.work_force, x.income, "ro") # The bottom-right axes

plt.show()
```



hi

Slide Download Link

You can download these slides from the below link:

https://docs.google.com/presentation/d/1AvTu8Ny14KQP9MFEzPggr2fYmg0IoX0MJlyImG4VhLY/edit?usp=sharing

DPhi

That’s it for this unit. Thank you!

Feel free to post any queries on Discuss.

DPhi