مقرر التحليل العددي

د. يمار الحموي

م.اية خيربك

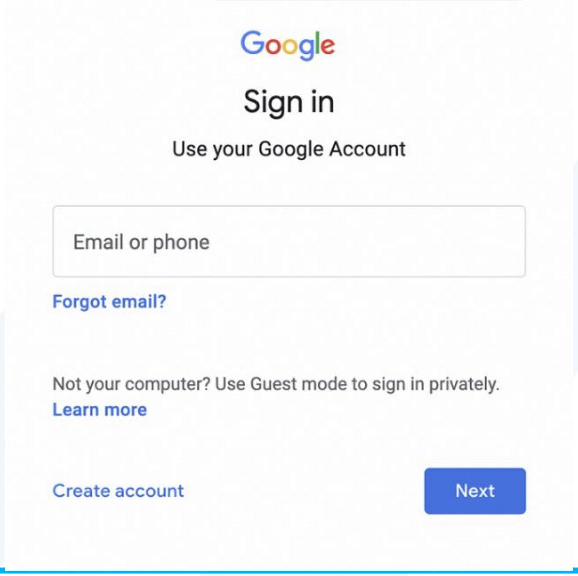
م. ندى جنيدي

العملي

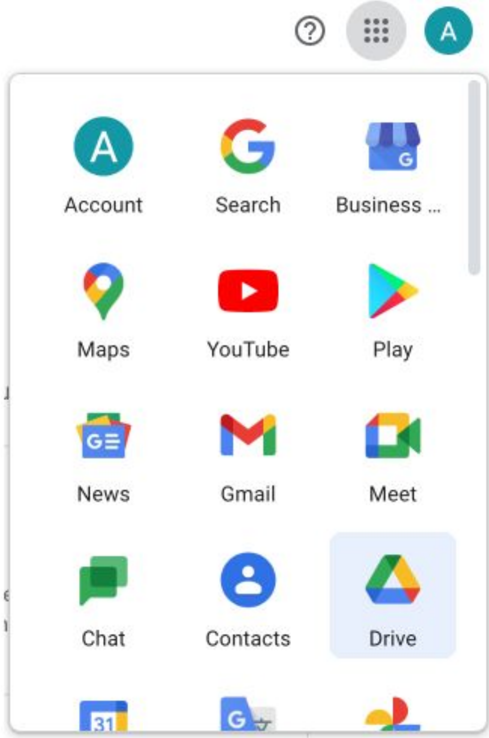
الفصل الثاني 2022-2023

- التعامل مع Colab :

- **Step 1 : Create a Google Colab Notebook.**
  firstly you need to have google account

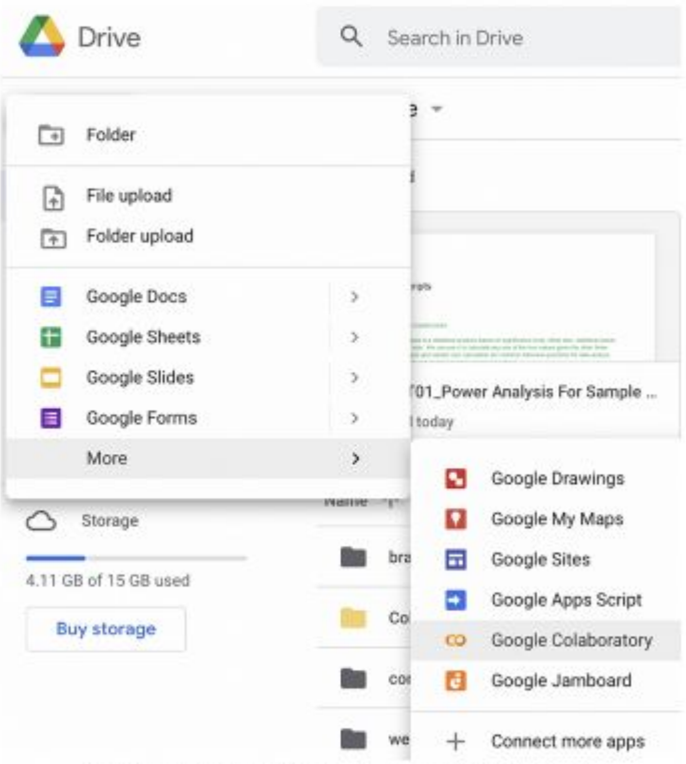


Step 1.2: Click the 9 dots icon on the upper-right corner and select **Drive**.



Select Google Drive for Colab — Image from GrabNGoInfo.com

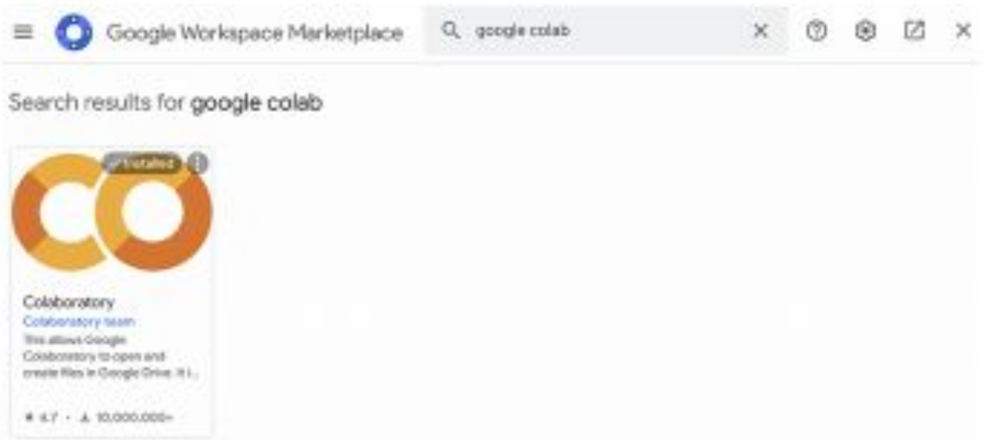
Step 1.3: Click **New** -> **More** -> **Google Colaboratory** to open a new Colab Notebook.



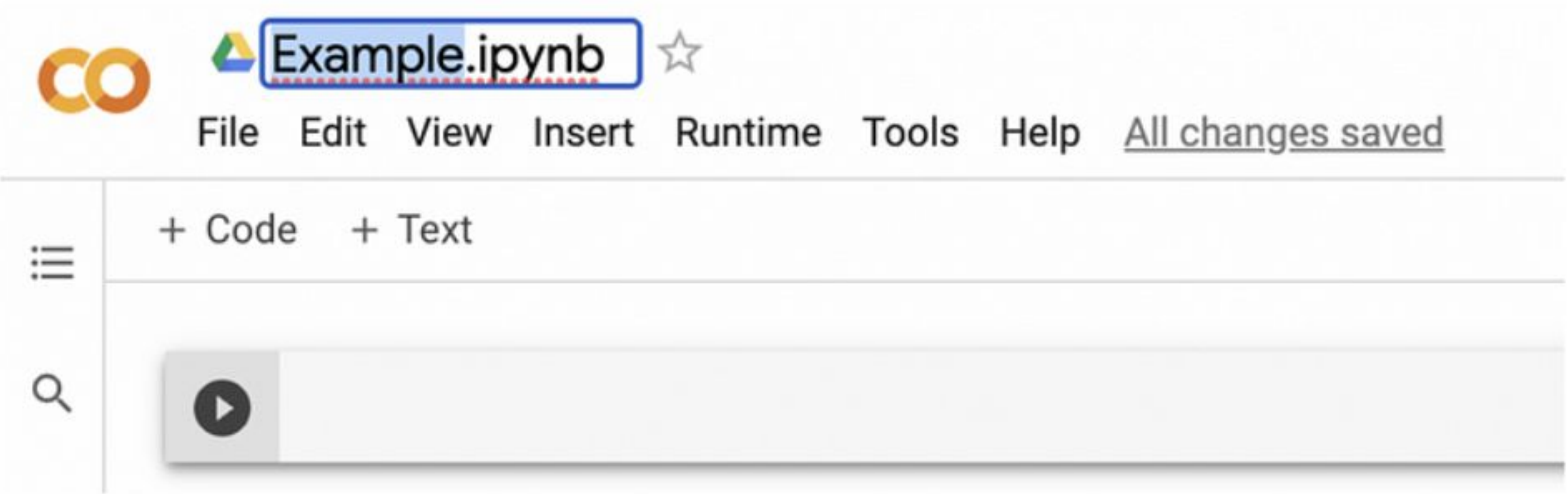
Create Google Colab Notebook — Image from GrabNGoInfo.com

If you do not see Google Colaboratory in the list, click **Connect more apps** and search **Google Colab** in the search bar.

Install Google Colab and click **New -> More -> Google Colaboratory** to open a new Colab Notebook.

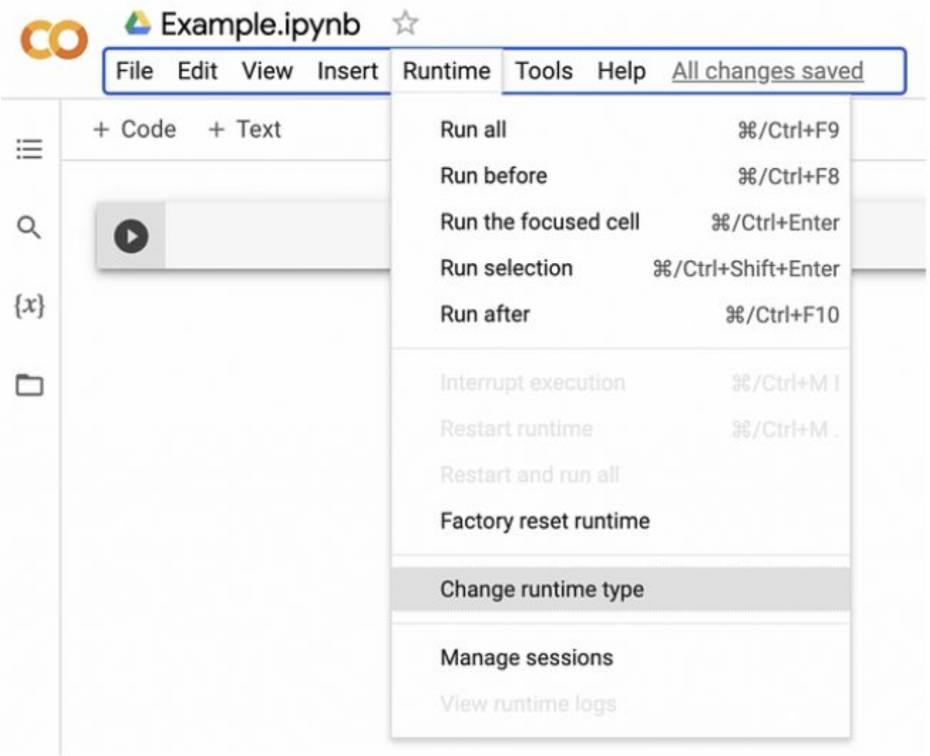


Step 1.4: Click the file name one the upper-left corner and change the file name.



Step 2 (Optional): Set Up Runtime

The 2nd step is to set up the run time. The default run time uses CPUs, but you can change the run time by clicking **Runtime** -> **Change runtime type**.

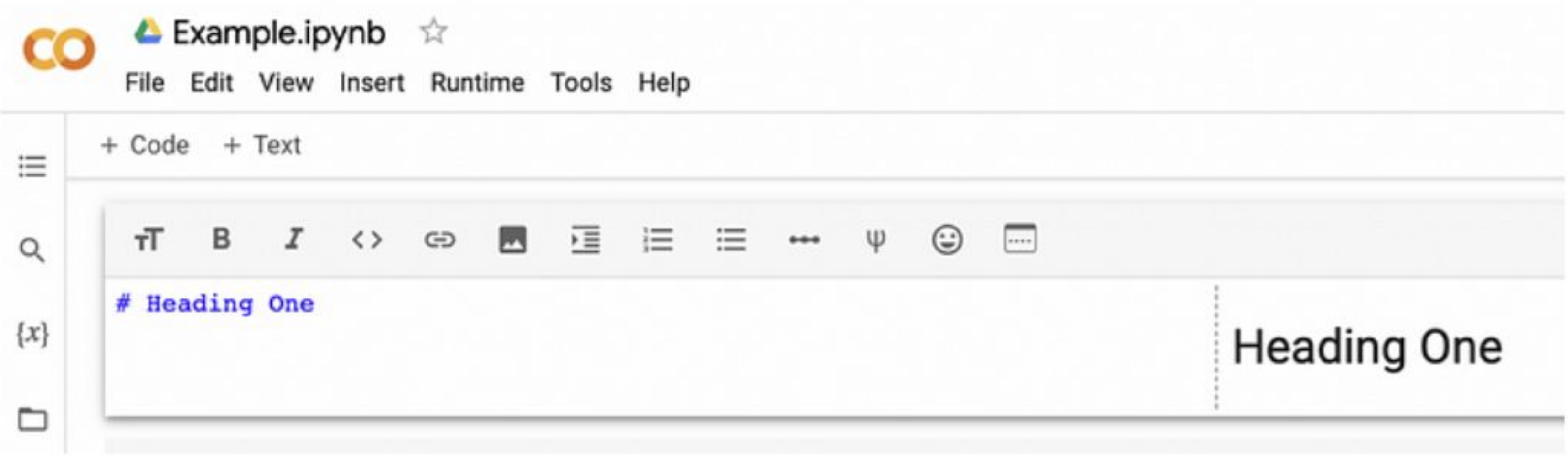


Step 3: Create and Run Cells

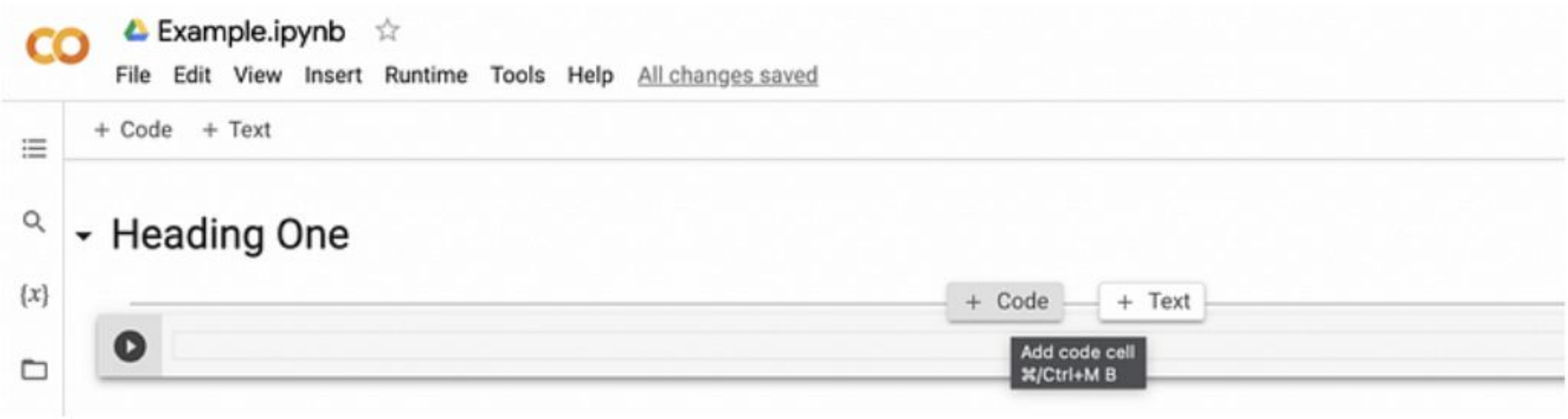
There are two types of cells in the Google Colab notebook, text cells, and code cells.

Step 3.1: To add a new text cell, hover the mouse in the middle until + Code and **+Text** show up. Click **+Text**.

Step 3.2: Type text in the newly added text cell. You can use markdown to format the text, and the rendered text shows on the right-hand side. The cell renders automatically when clicking outside the cell.



Step 3.3: To add a new code cell, hover the mouse in the middle until **+Code** and **+Text** show up. Click **+Code**.



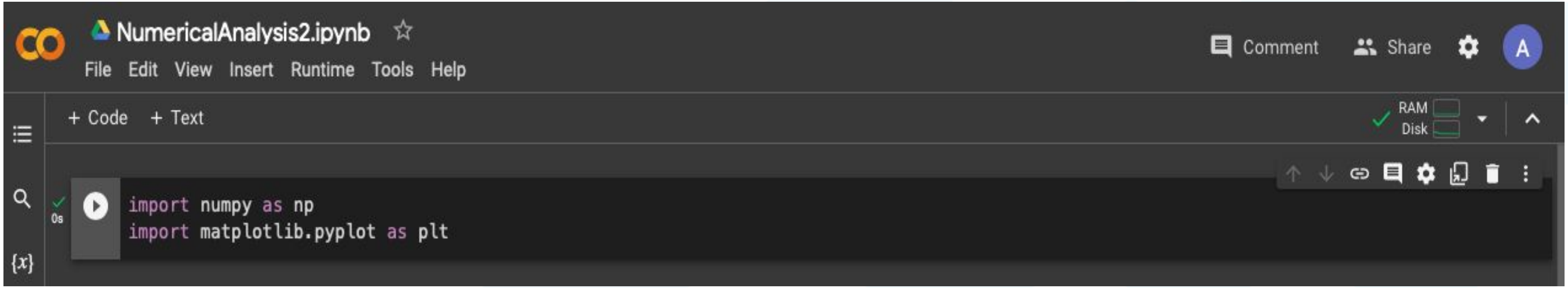
Step 3.4: Type Python code in the newly added code cell, and click the run button (a black circle with a white triangle in it) to run the code. Here we entered `2+3` and get the results of `5`.

Plotting using Colab + Python

2D Plotting

In Python, the *matplotlib* is the most important package that to make a plot

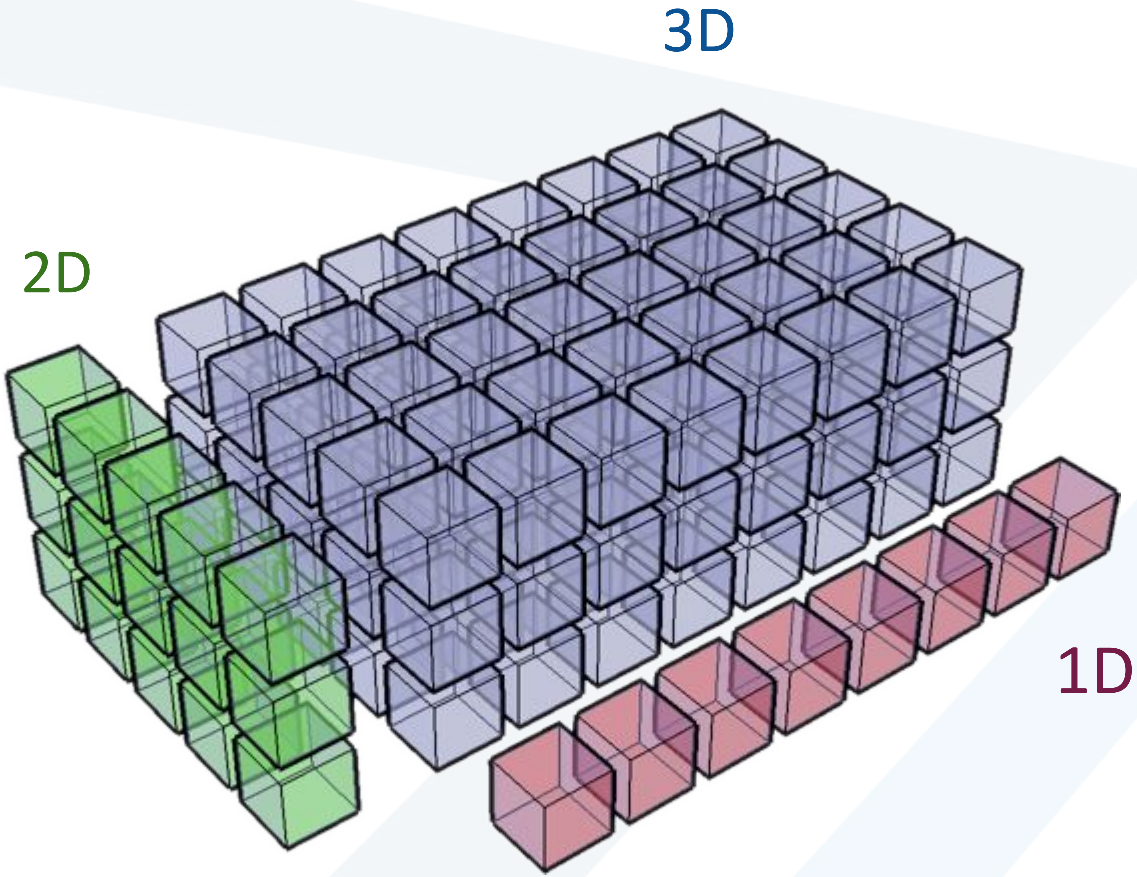
Usually the first thing we need to do to make a plot is to import the matplotlib package.




```
import numpy as np
import matplotlib.pyplot as plt
```

إنشاء مصفوفةNumpy

```
import numpy as np
```



باستخدام مكتبة **Matplotlib** نستطيع توليد رسوم بيانية وأشكال خاصة بالبيانات عبر القليل من الشيفرة البرمجية بالبايثون، وتتجلى فائدة هذه المكتبات أثناء عمليات تحليل البيانات وتجهيز التقارير الإحصائية وعمليات تنقيب البيانات وتعليم الالة.

المنحنى البياني في مكتبة Matplotlib

من أجل القيام برسم منحنى بياني, نستورد وحدة pyplot من مكتبة **Matplotlib.**

تحتوي pyplot على مجموعة من الوظائف والدوال التي تتشابه مع أوامر برنامج **MATLAB** في الشكل والغرض. عند استيراد المكتبة من الافضل اعطاء الوحدة مُسمًا سهلًا في الكتابة حتى لا نكتب اسم الوحدة في كل مرة نحتاجها فيه، والشائع هنا أن نسميها ب plt.

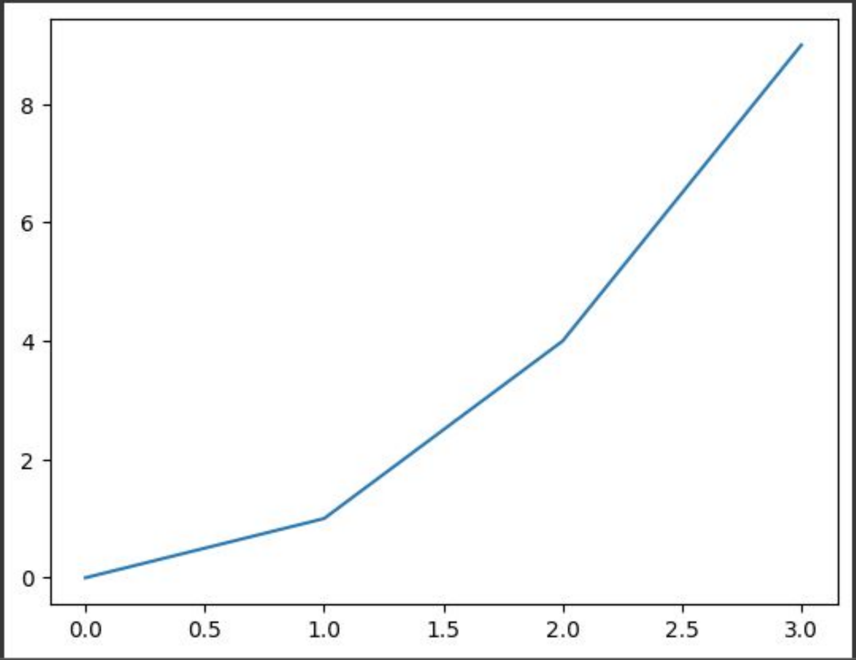
بعد ذلك نُجهز البيانات التي نريد عرضها في الشكل.

- **The basic plotting function is *plot(x,y)*.**
- **The *plot* function takes in two lists/arrays, x and y, and produces a visual display of the respective points in x and y.**

```
x = [0, 1, 2, 3]
y = [0, 1, 4, 9]
plt.plot(x, y)
plt.show()
```

[1]
```python
import numpy as np
import matplotlib.pyplot as plt
```

```python
x = [0, 1, 2, 3]
y = [0, 1, 4, 9]
plt.plot(x, y)
plt.show()
```



- You will notice in the above figure that by default, the plot function connects each point with a blue line.
- To make the function look smooth, use a finer discretization points.
- The *plt.plot* function did the main job to plot the figure, and *plt.show()* is telling Python that we are done plotting and please show the figure.
- Also, you can see some buttons beneath the plot that you could use it to move the line, zoom in or out, save the figure.
- Note that, before you plot the next figure, you need to turn off the interactive plot by pressing the *stop interaction* button on the top right of the figure.
- Otherwise, the next figure will be plotted in the same frame. Or we could simply using the magic function *%matplotlib inline* to turn off the interactive features.

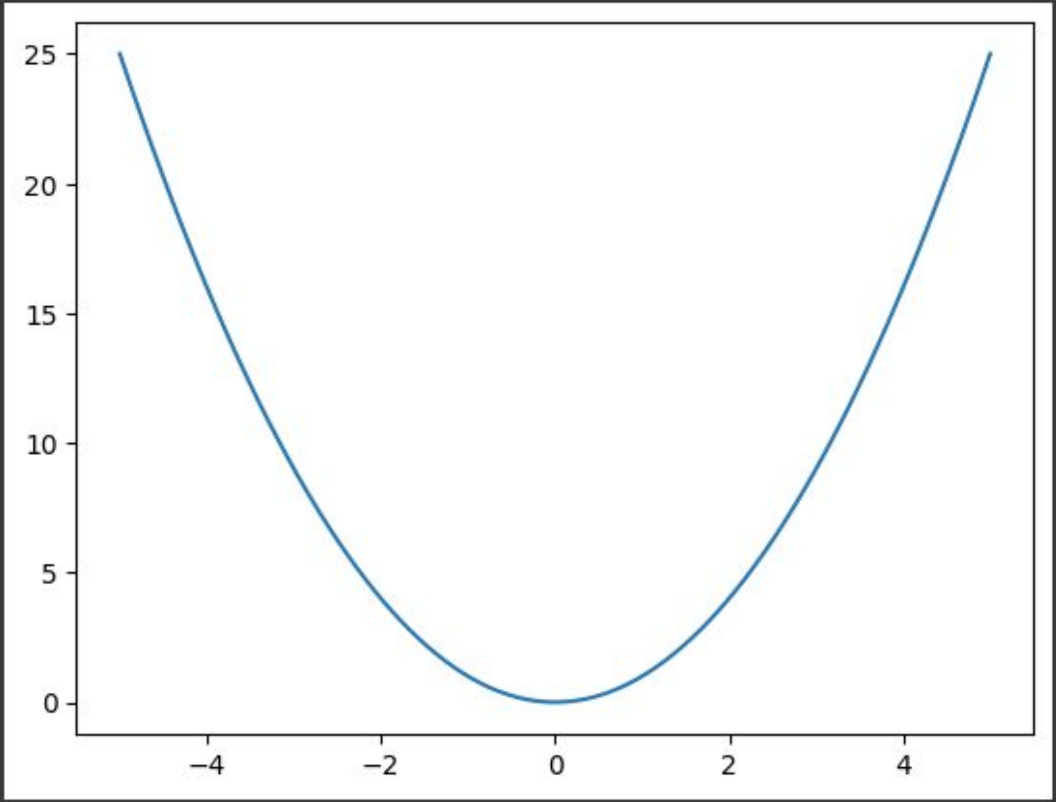
Make a plot of the function

$f(x) = x\hat{}2\ for\ -5 \leq x \leq 5$

```
x = np.linspace(-5,5, 100)
plt.plot(x, x**2)
plt.show()
```

للحصول على مصفوفة تبدأ برقم معين وتنتهي برقم اخر مع تحديد عدد عناصر المصفوفة بحيث يكون مدى قيمة عناصرها بين الرقمين نستخدم الدالة np.linespace

```
x = np.linspace(-5,5, 100)
plt.plot(x, x**2)
plt.show()
```



To change the marker or line, you can put a third input argument into plot, which is a string that specifies the color and line style to be used in the plot.

For example, *plot(x,y,’ro’)* will plot the elements of x against the elements of y using red, r, circles, ‘o’.

جامعة المنارة

Symbol	Description	Symbol	Description
b	blue	T	T
g	green	s	square
r	red	d	diamond
c	cyan	v	triangle (down)
m	magenta	^	triangle (up)
y	yellow	<	triangle (left)
k	black	>	triangle (right)
w	white	p	pentagram
.	point	h	hexagram

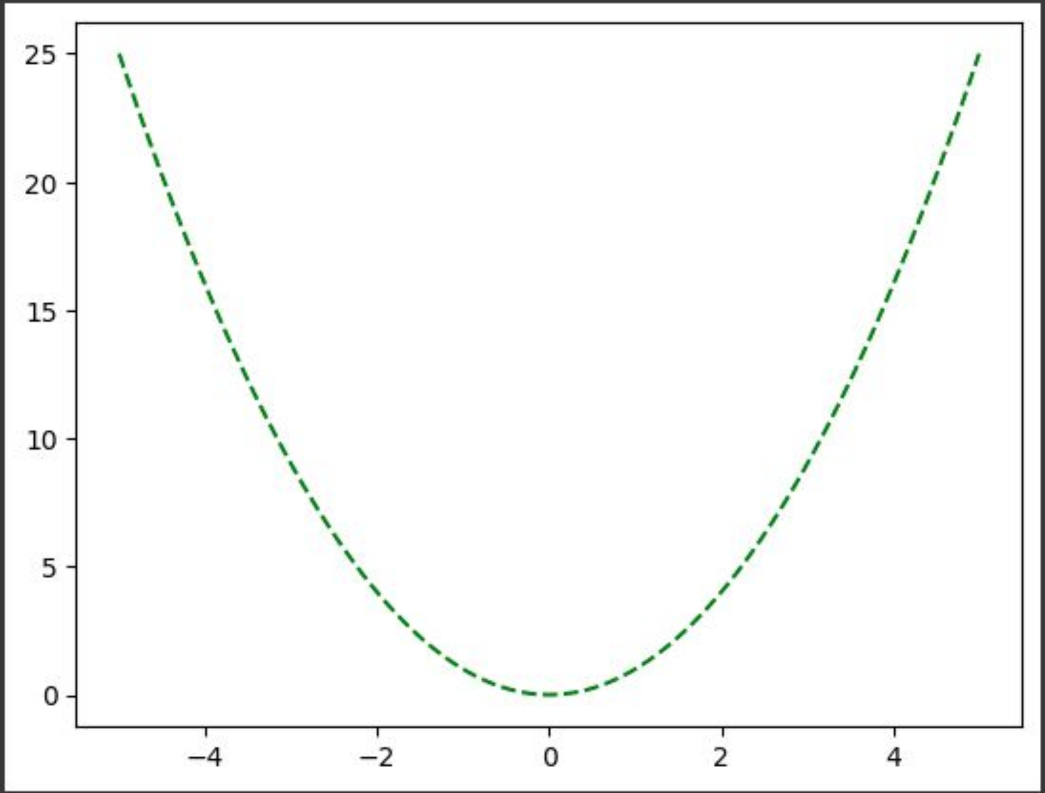
Symbol	Description	Symbol	Description
o	circle	-	solid
x	x-mark	:	dotted
+	plus	-.	dashdot
*	star	–	dashed

Make a plot of the function

$f(x) = x$^2 for −5 ≤ x ≤ 5 using a dashed green line

```
x = np.linspace(-5,5, 100)
plt.plot(x, x**2, 'g--')
plt.show()
```

```
x = np.linspace(-5,5, 100)
plt.plot(x, x**2, 'g--')
plt.show()
```



Before the *plt.show()* statement, you can add in and plot more datasets within one figure.

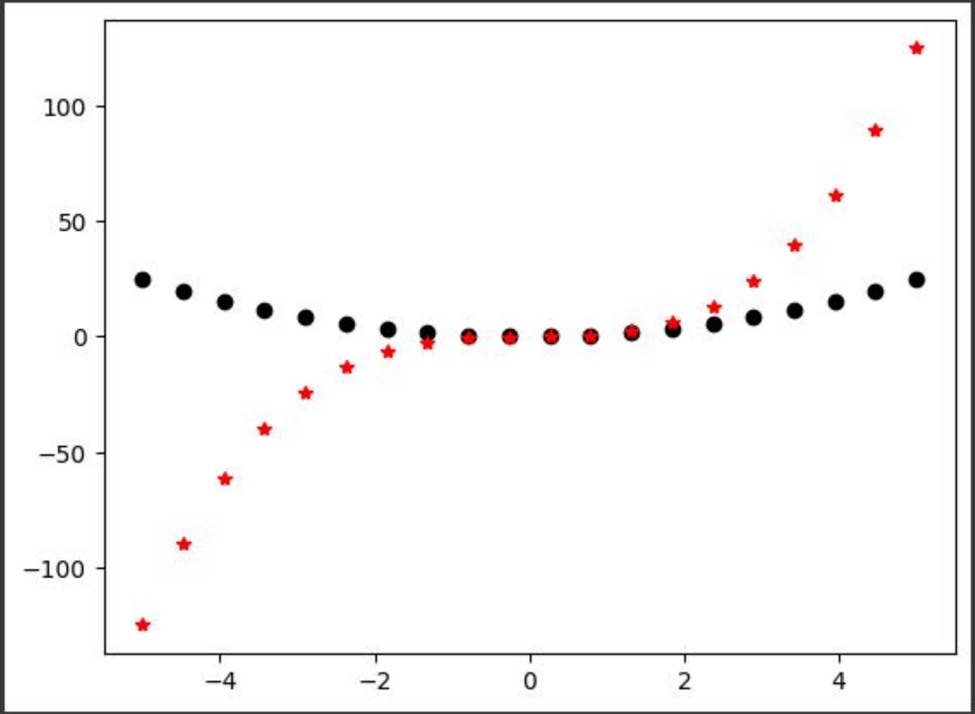
Make a plot of the function

$f(x) = x^2$ *and* $g(x) = x^3$ *for* $-5 \leq x \leq 5$

Use different colors and markers for each function.

```
x = np.linspace(-5,5,20)
plt.plot(x, x**2, 'ko')
plt.plot(x, x**3, 'r*')
plt.show()
```

```
x = np.linspace(-5,5,20)
plt.plot(x, x**2, 'ko')
plt.plot(x, x**3, 'r*')
plt.show()
```



It is customary in engineering and science to always give your plot a title and axis labels so that people know what your plot is about.

Besides, sometimes you want to change the size of the figure as well.

You can add a title to your plot using the *title* function, which takes as input a string and puts that string as the title of the plot.

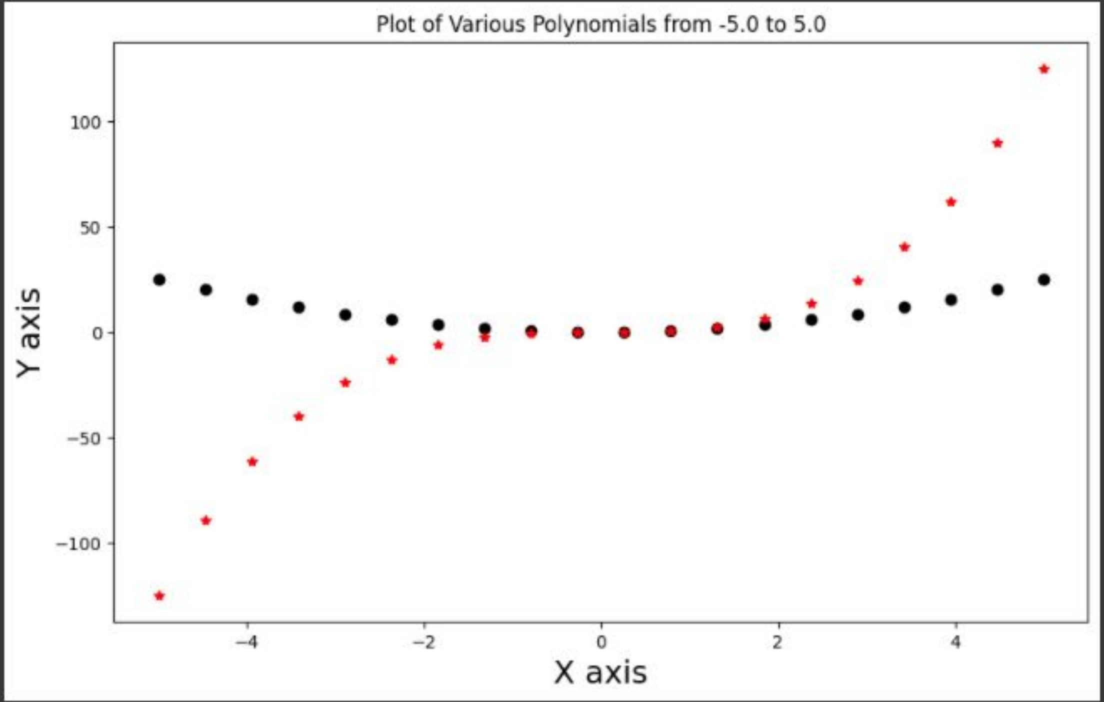
The functions *xlabel* and *ylabel* work in the same way to name your axis labels.

For changing the size of the figure, we could create a figure object and resize it. Note, every time we call *plt.figure* function, we create a new figure object to draw something on it.

Add a title and axis labels to the previous plot. And make the figure larger with width 10 inches, and height 6 inches.

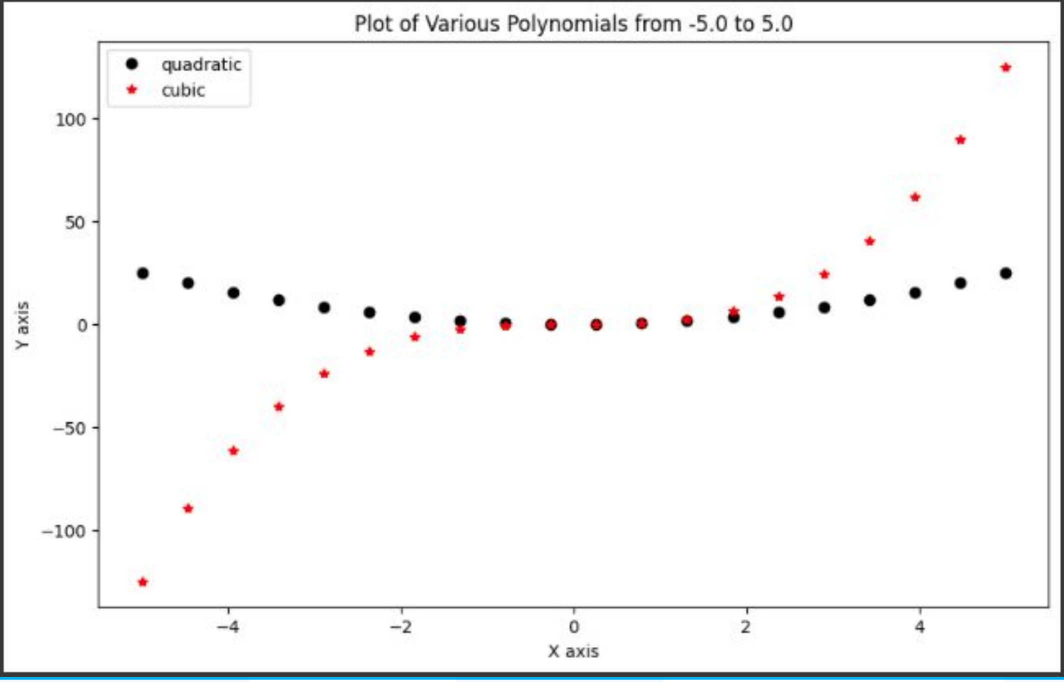
```
plt.figure(figsize = (10,6))
x = np.linspace(-5,5,20)
plt.plot(x, x**2, 'ko')
plt.plot(x, x**3, 'r*')
plt.title(f'Plot of Various Polynomials from {x[0]} to {x[-1]}')
plt.xlabel('X axis', fontsize = 18)
plt.ylabel('Y axis', fontsize = 18)
plt.show()
```

```python
plt.figure(figsize = (10,6))

x = np.linspace(-5,5,20)
plt.plot(x, x**2, 'ko')
plt.plot(x, x**3, 'r*')
plt.title(f'Plot of Various Polynomials from {x[0]} to {x[-1]}')
plt.xlabel('X axis', fontsize = 18)
plt.ylabel('Y axis', fontsize = 18)
plt.show()
```



You can add a legend to your plot by using the *legend* function. And add a *label* argument in the *plot* function. The legend function also takes argument of *loc* to indicate where to put the legend, try to change it from 0 to 10.

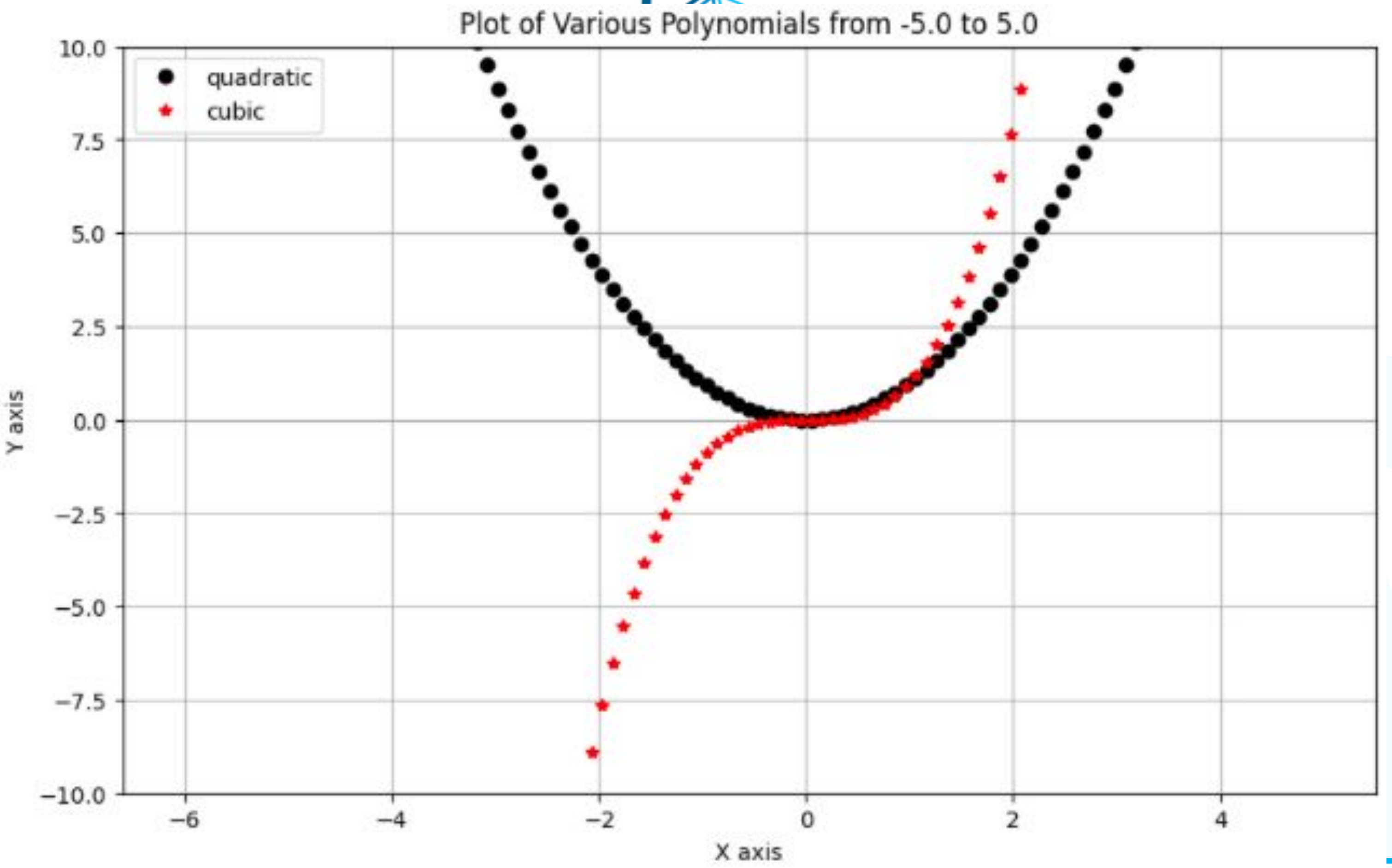
```python
plt.figure(figsize = (10,6))

x = np.linspace(-5,5,20)
plt.plot(x, x**2, 'ko', label = 'quadratic')
plt.plot(x, x**3, 'r*', label = 'cubic')
plt.title(f'Plot of Various Polynomials from {x[0]} to {x[-1]}')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend(loc = 2)
plt.show()
```



Finally, you can further customize the appearance of your plot to change the limits of each axis using the *xlim* or *ylim* function. Also, you can use the *grid* function to turn on the grid of the figure.

Change the limits of the plot so that x is visible from -6 to 6 and y is visible from -10 to 10. Turn the grid on.

```python
plt.figure(figsize = (10,6))

x = np.linspace(-5,5,100)
plt.plot(x, x**2, 'ko', label = 'quadratic')
plt.plot(x, x**3, 'r*', label = 'cubic')
plt.title(f'Plot of Various Polynomials from {x[0]} to {x[-1]}')
plt.xlabel('X axis')
plt.ylabel('Y axis')
plt.legend(loc = 2)
plt.xlim(-6.6)
plt.ylim(-10,10)
plt.grid()
plt.show()
```


Plot of Various Polynomials from -5.0 to 5.0
quadratic
cubic
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−6
−4
−2
0
2
4
Y axis
X axis

We can create a **table of plots** on a single figure using the ***subplot* function**.

The *subplot* function takes three inputs:

- the number of rows of plots.
- the number of columns of plots.
- which plot all calls to plotting functions should plot.

You can move to a different subplot by calling the subplot again with a different entry for the plot location.

There are several other plotting functions that plot x versus y data.

Some of them are ***scatter***, ***bar***, ***loglog***, ***semilogx***, **and** ***semilogy***.

scatter **works exactly the same as plot except it defaults to red circles (i.e.,** ***plot(x,y,'ro')*** **is equivalent to** ***scatter(x,y)*****).**

The ***bar*** **function plots bars centered at x with height y.**

The ***loglog, semilogx, and semilogy*** **functions plot the data in x and y with the x and y axis on a log scale, the x axis on a log scale and the y axis on a linear scale, and the y axis on a log scale and the x axis on a linear scale, respectively.**

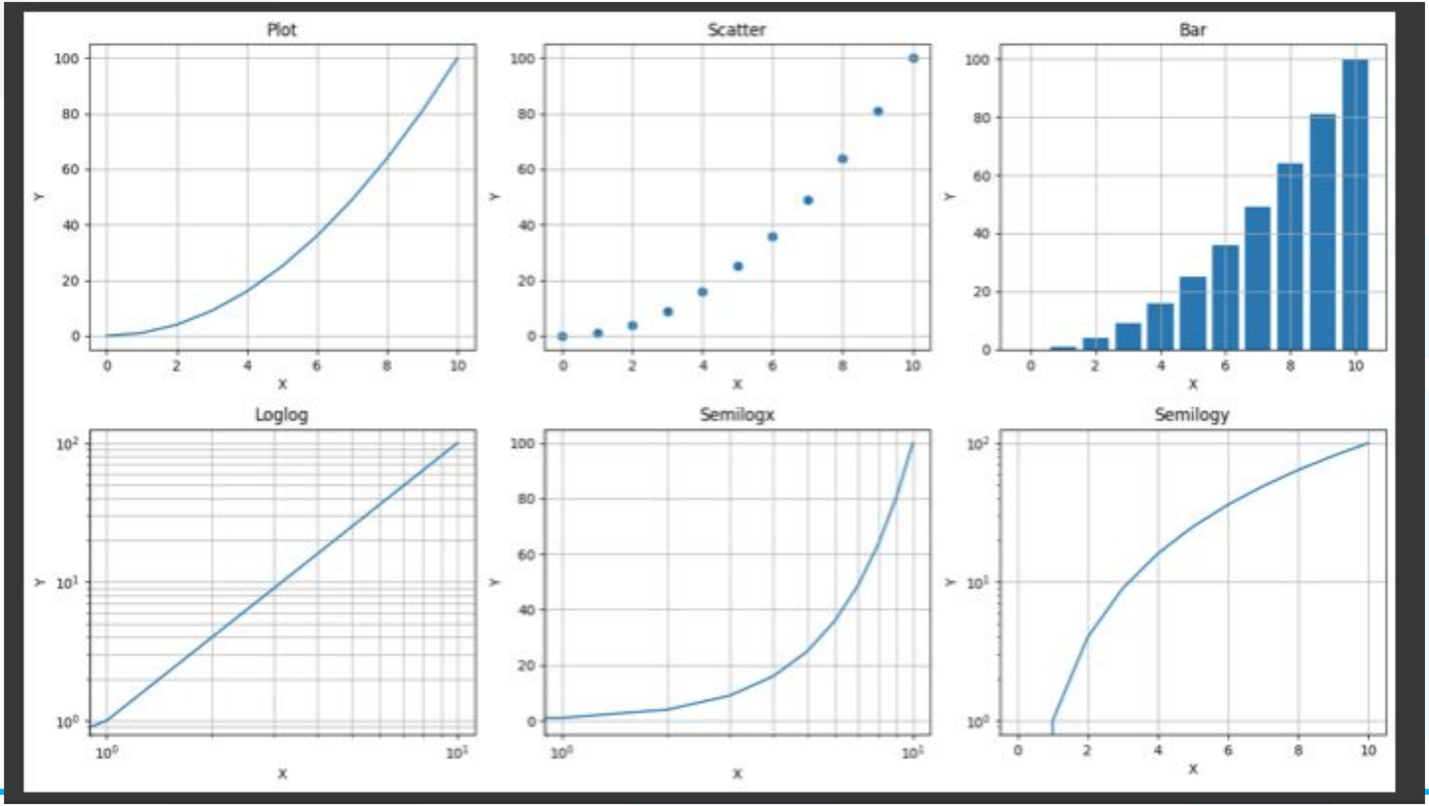
Given the lists x = np.arange(11) and y = x^2.

Create a 2 by 3 subplot where each subplot plots x versus y using *plot*, *scatter*, *bar*, *loglog*, *semilogx*, and *semilogy*.

Title and label each plot appropriately.

Use a grid, but a legend is not necessary.

```python
x = np.arange(11)
y = x**2

plt.figure(figsize = (14, 8))

plt.subplot(2, 3, 1)
plt.plot(x,y)
plt.title('Plot')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()

plt.subplot(2, 3, 2)
plt.scatter(x,y)
plt.title('Scatter')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()

plt.subplot(2, 3, 3)
plt.bar(x,y)
plt.title('Bar')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()

plt.subplot(2, 3, 4)
plt.loglog(x,y)
plt.title('Loglog')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(which='both')

plt.subplot(2, 3, 5)
plt.semilogx(x,y)
plt.title('Semilogx')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(which='both')

plt.subplot(2, 3, 6)
plt.semilogy(x,y)
plt.title('Semilogy')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid()

plt.tight_layout()

plt.show()
```

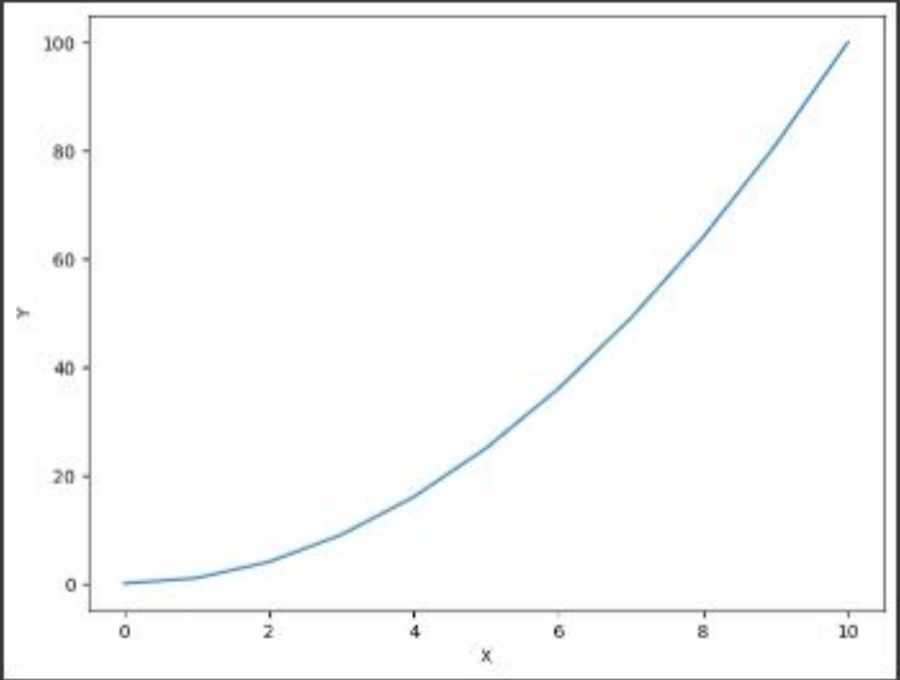

Plot
Scatter
Bar
Loglog
Semilogx
Semilogy
X
Y
0
2
4
6
8
10
20
40
60
80
100
10^0
10^1
10^2

We could see that at the end of our plot, we used `plt.tight_layout` to make the sub-figures not overlap with each other, you can try and see the effect without this statement.

Besides, sometimes, you want to save the figures as a specific format, such as pdf, jpeg, png, and so on. You can do this with the function `plt.savefig`.

```
plt.figure(figsize = (8,6))
plt.plot(x,y)
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('image.pdf')
```

```
plt.figure(figsize = (8,6))
plt.plot(x,y)
plt.xlabel('X')
plt.ylabel('Y')
plt.savefig('image.pdf')
```



Thanks