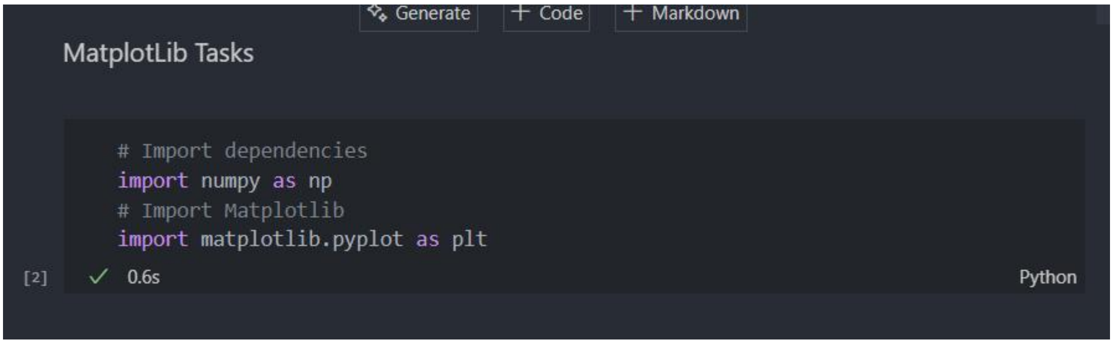
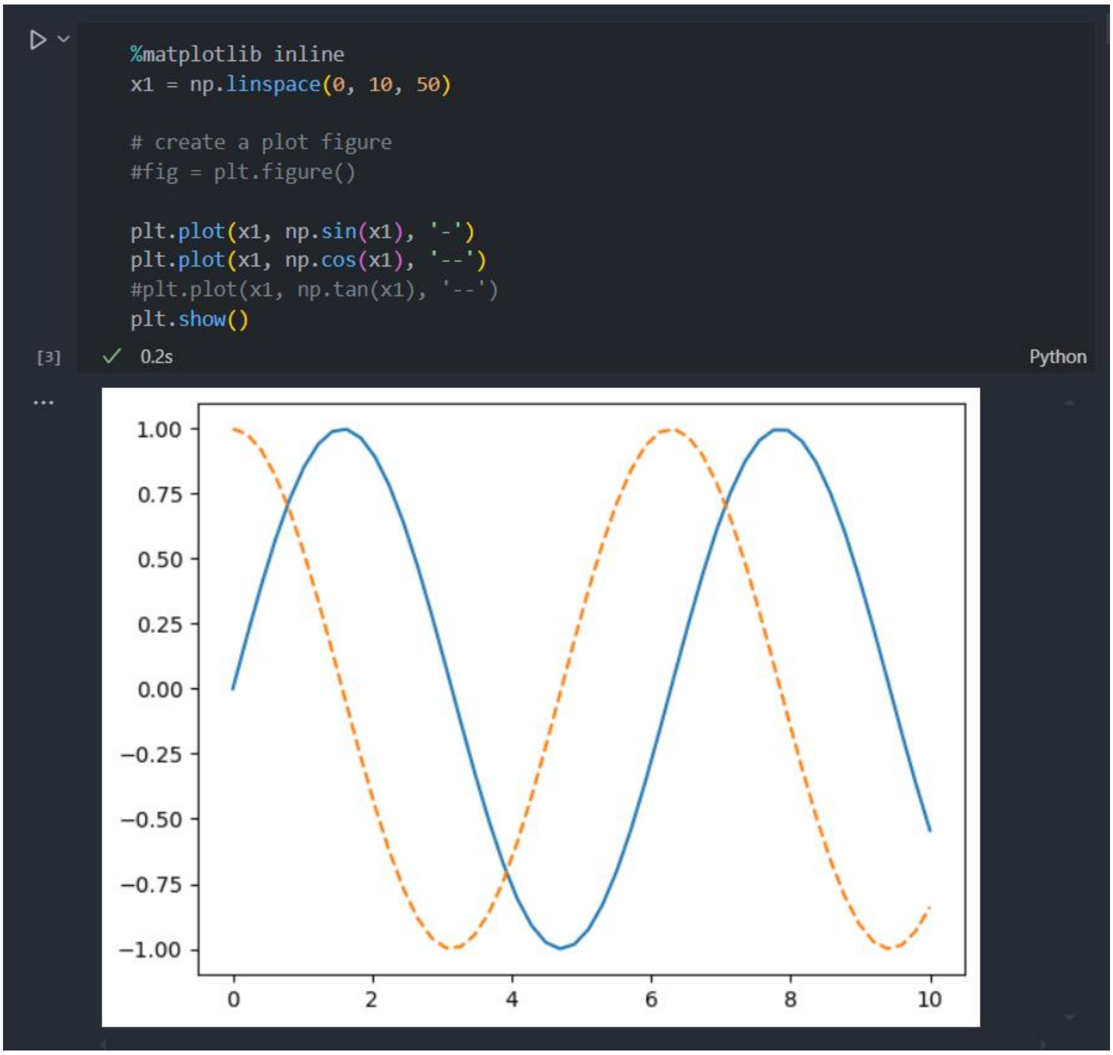
Matplotlib Tasks

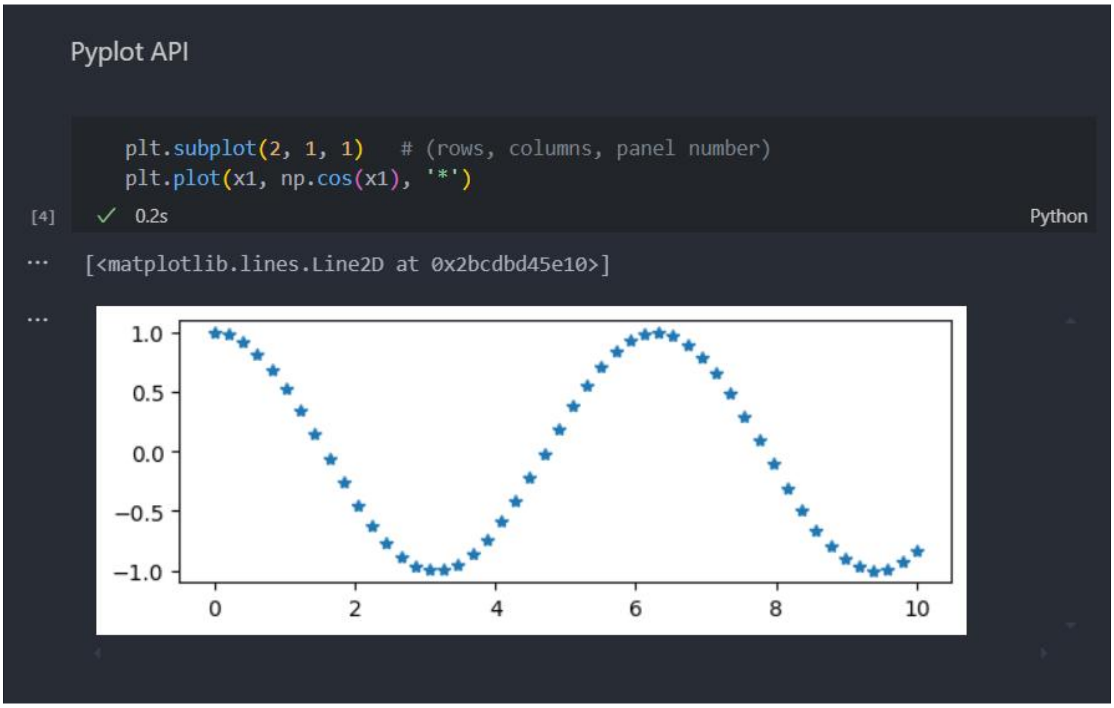


MatplotLib Tasks

```
# Import dependencies
import numpy as np
# Import Matplotlib
import matplotlib.pyplot as plt
```



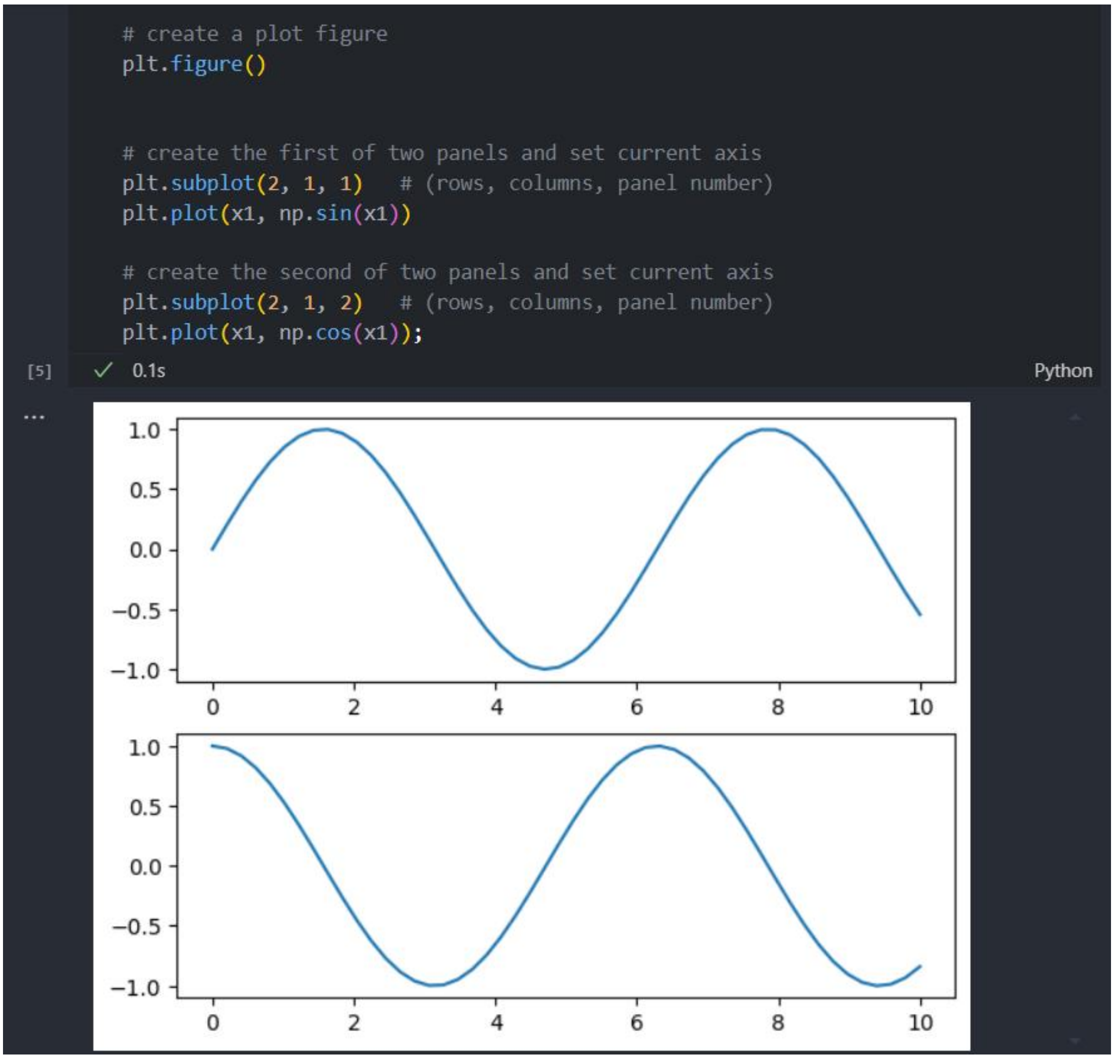
```
%matplotlib inline
x1 = np.linspace(0, 10, 50)

# create a plot figure
#fig = plt.figure()

plt.plot(x1, np.sin(x1), '-')
plt.plot(x1, np.cos(x1), '--')
#plt.plot(x1, np.tan(x1), '--')
plt.show()
```



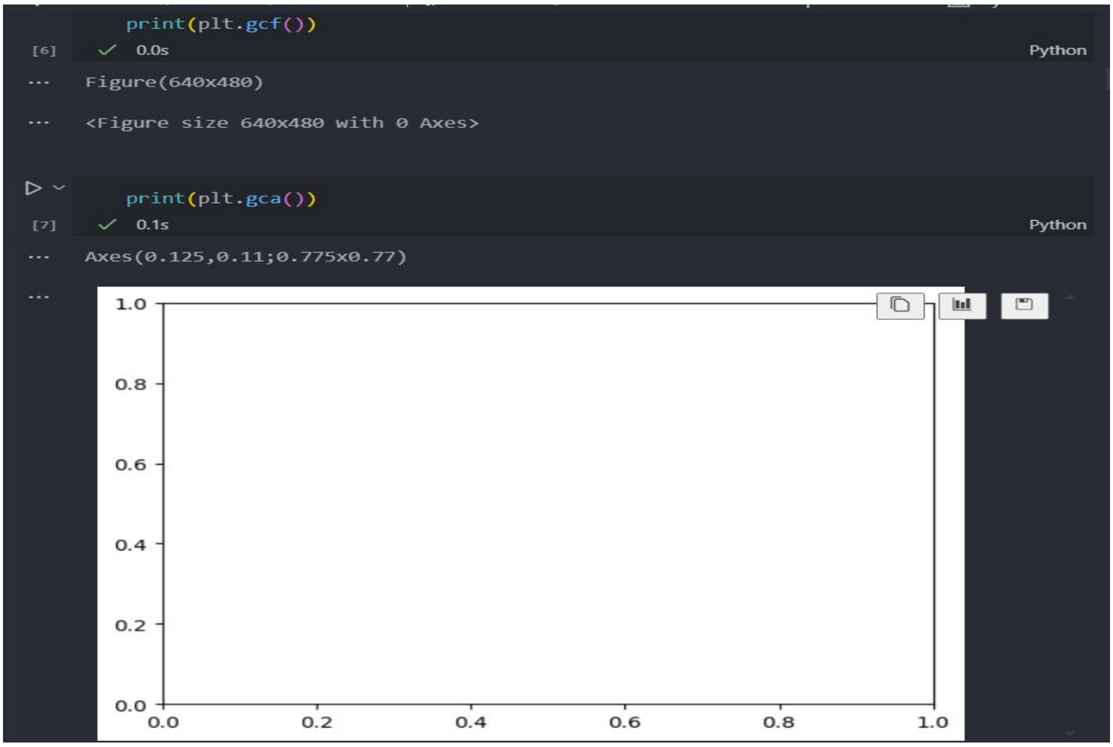
Pyplot API

```
plt.subplot(2, 1, 1)   # (rows, columns, panel number)
plt.plot(x1, np.cos(x1), '*')
```

```
[<matplotlib.lines.Line2D at 0x2bcdbd45e10>]
```



```
# create a plot figure
plt.figure()


# create the first of two panels and set current axis
plt.subplot(2, 1, 1)   # (rows, columns, panel number)
plt.plot(x1, np.sin(x1))

# create the second of two panels and set current axis
plt.subplot(2, 1, 2)   # (rows, columns, panel number)
plt.plot(x1, np.cos(x1));
```



```
print(plt.gcf())
```

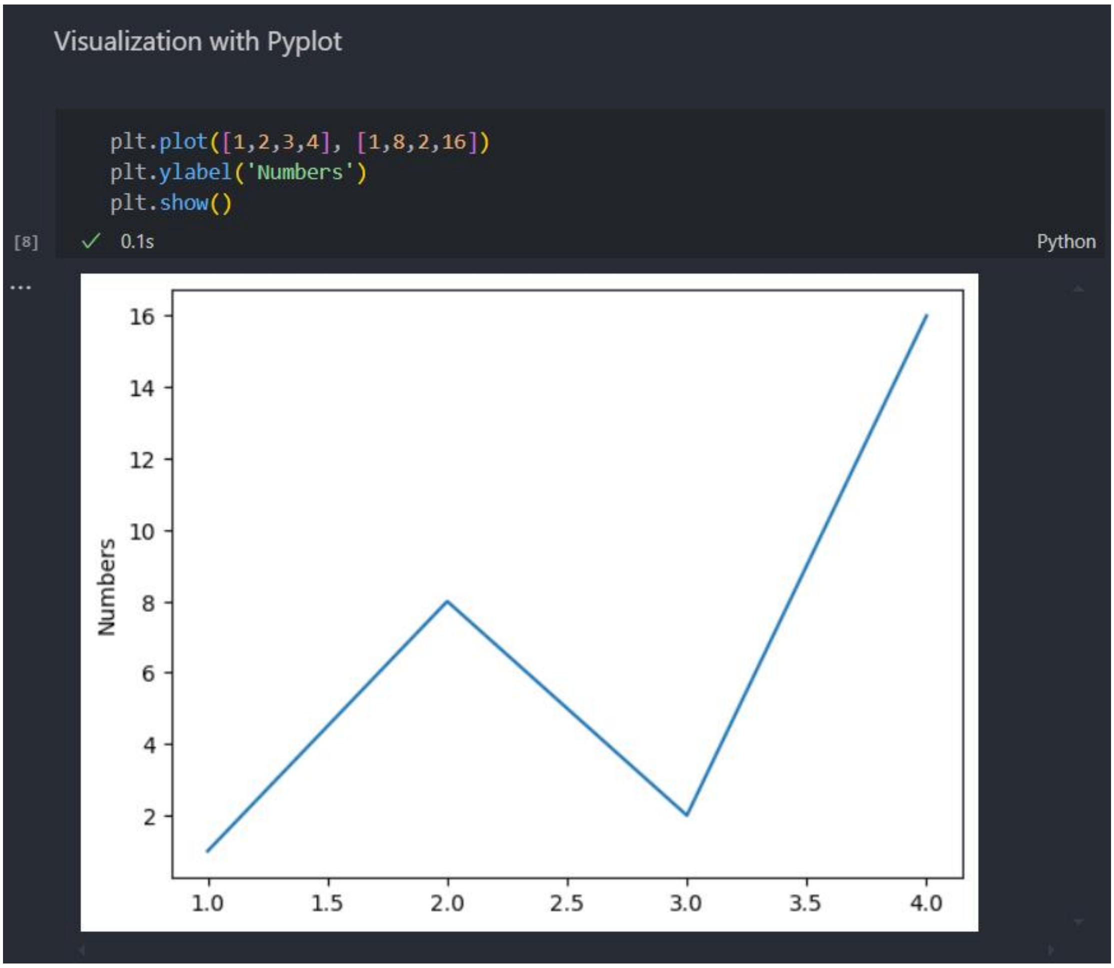
[6] ✓ 0.0s Python

```
Figure(640x480)

<Figure size 640x480 with 0 Axes>
```

```
print(plt.gca())
```

[7] ✓ 0.1s Python

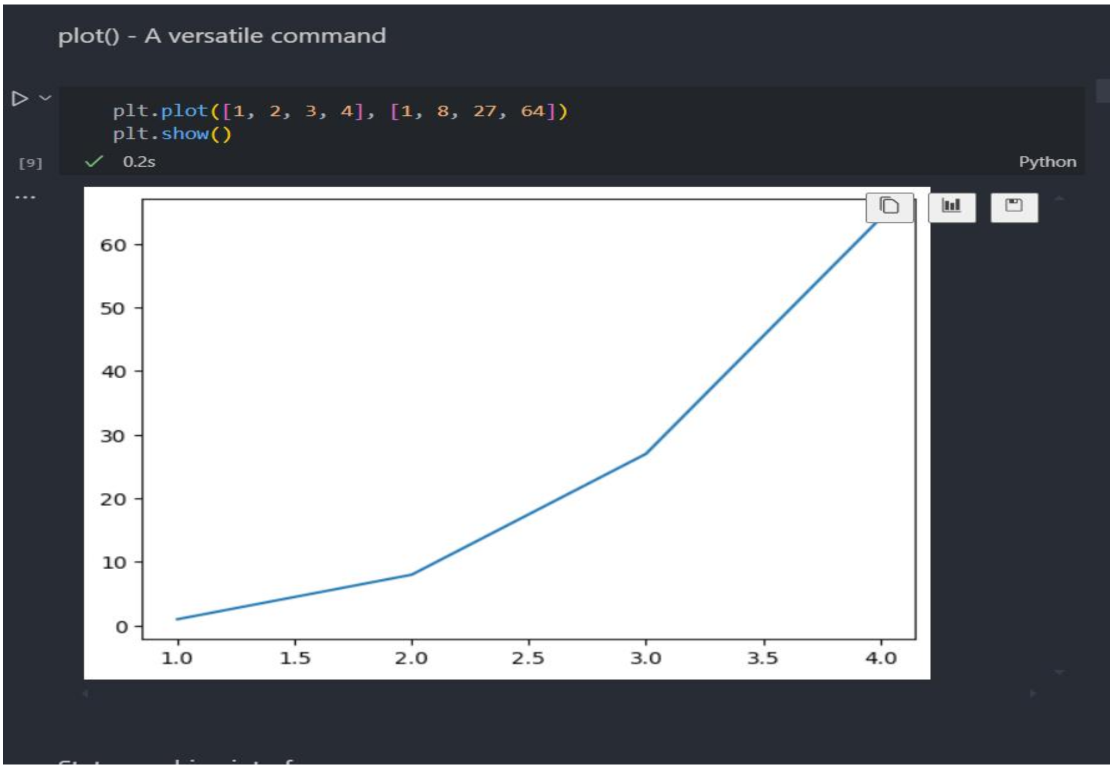
```
Axes(0.125,0.11;0.775x0.77)
```



Visualization with Pyplot

```
plt.plot([1,2,3,4], [1,8,2,16])
plt.ylabel('Numbers')
plt.show()
```

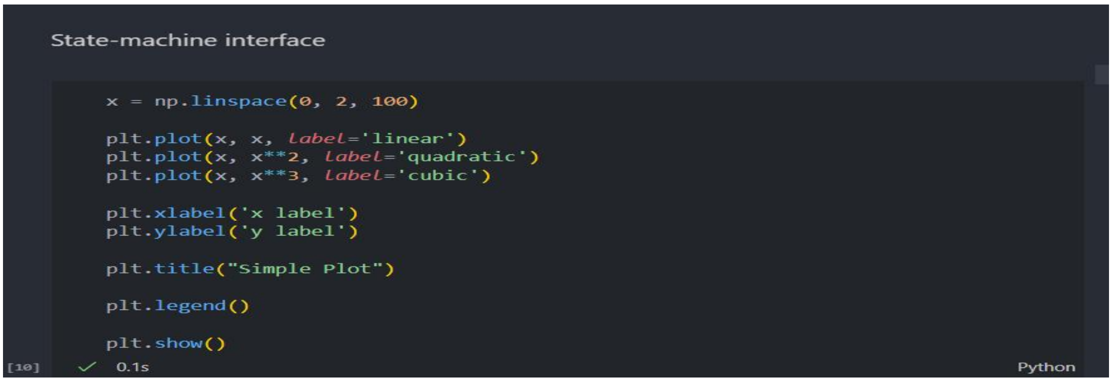
[8] ✓ 0.1s Python



plot() - A versatile command

```
plt.plot([1, 2, 3, 4], [1, 8, 27, 64])
plt.show()
```

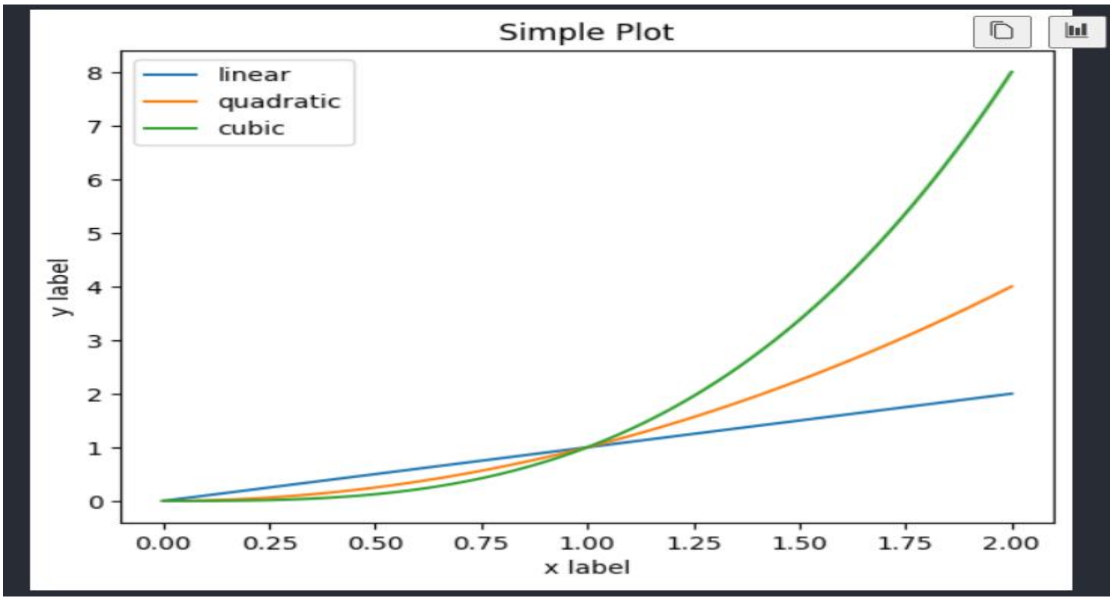
[9] ✓ 0.2s Python

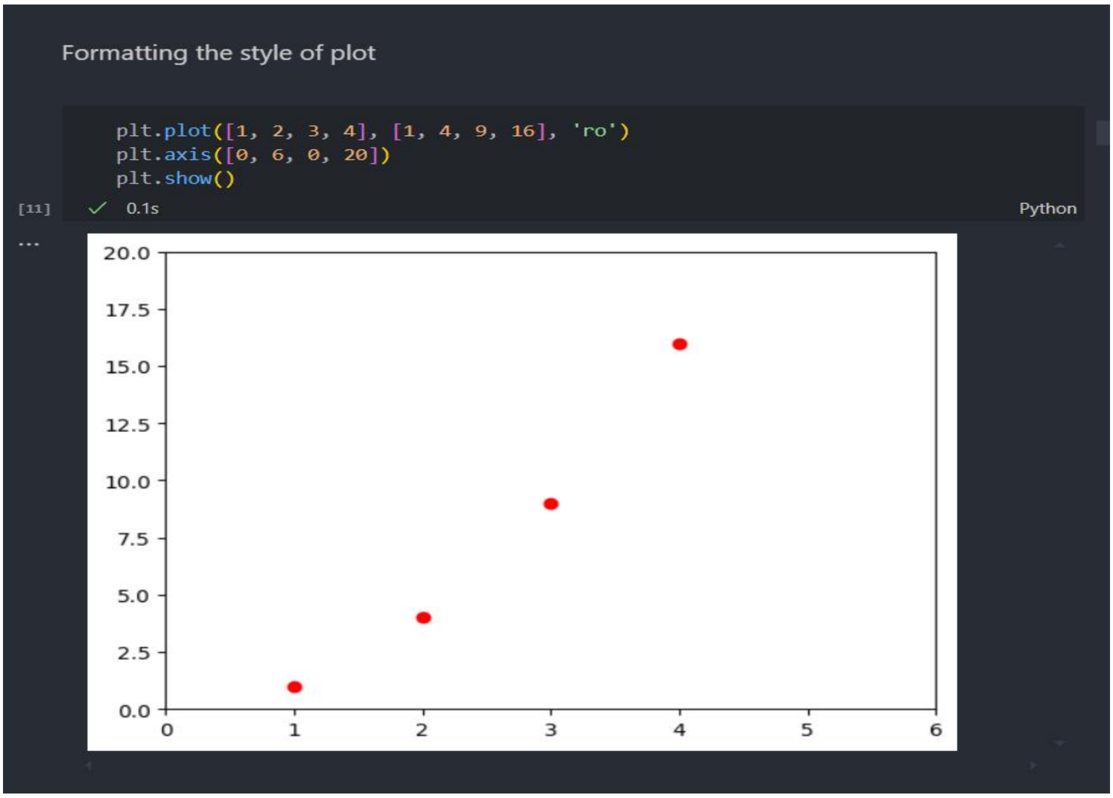


State-machine interface

```
x = np.linspace(0, 2, 100)

plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')

plt.xlabel('x label')
plt.ylabel('y label')

plt.title("Simple Plot")

plt.legend()

plt.show()
```

[10] ✓ 0.1s Python

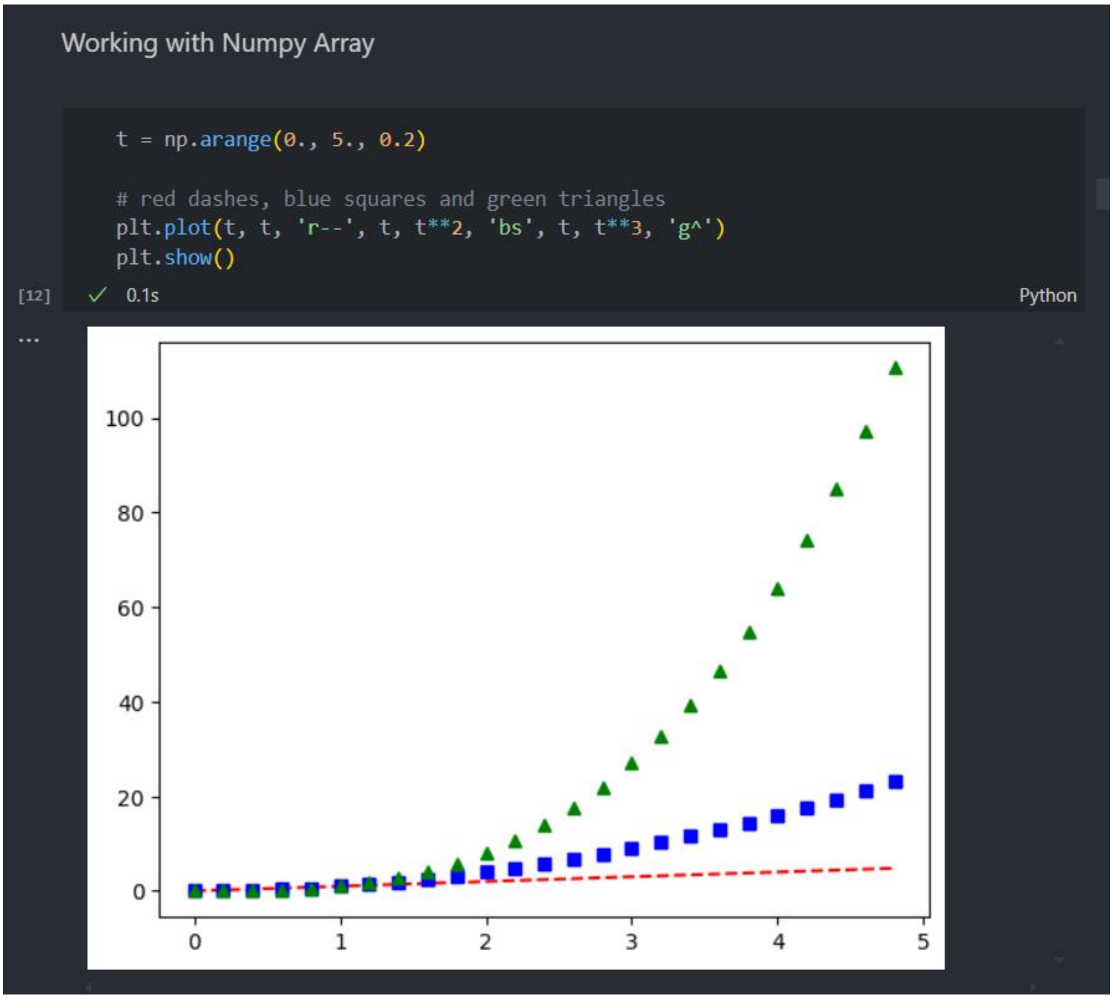




Formatting the style of plot

```
plt.plot([1, 2, 3, 4], [1, 4, 9, 16], 'ro')
plt.axis([0, 6, 0, 20])
plt.show()
```

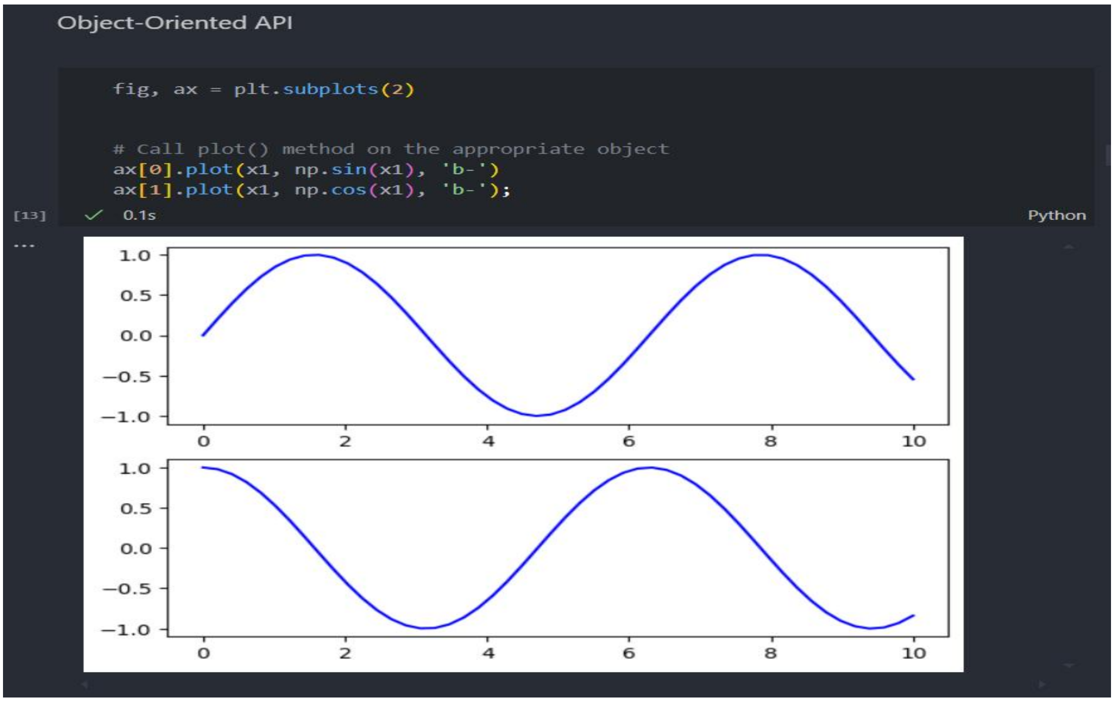
[11] ✓ 0.1s Python



Working with Numpy Array

```
t = np.arange(0., 5., 0.2)

# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```

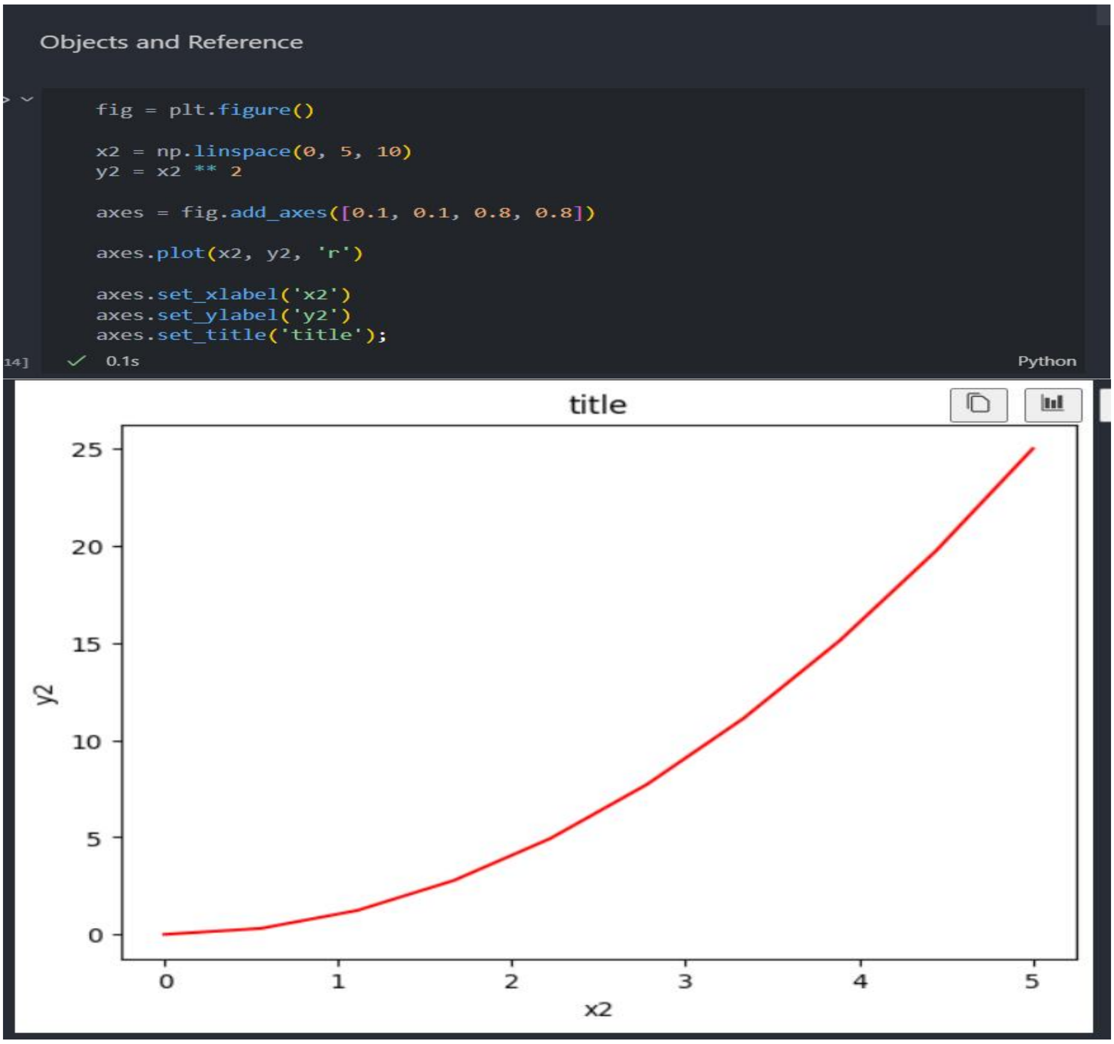
[12] ✓ 0.1s Python



Object-Oriented API

```
fig, ax = plt.subplots(2)


# Call plot() method on the appropriate object
ax[0].plot(x1, np.sin(x1), 'b-')
ax[1].plot(x1, np.cos(x1), 'b-');
```

[13] ✓ 0.1s Python



Objects and Reference

```
fig = plt.figure()

x2 = np.linspace(0, 5, 10)
y2 = x2 ** 2

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])

axes.plot(x2, y2, 'r')

axes.set_xlabel('x2')
axes.set_ylabel('y2')
axes.set_title('title');
```

14] ✓ 0.1s Python

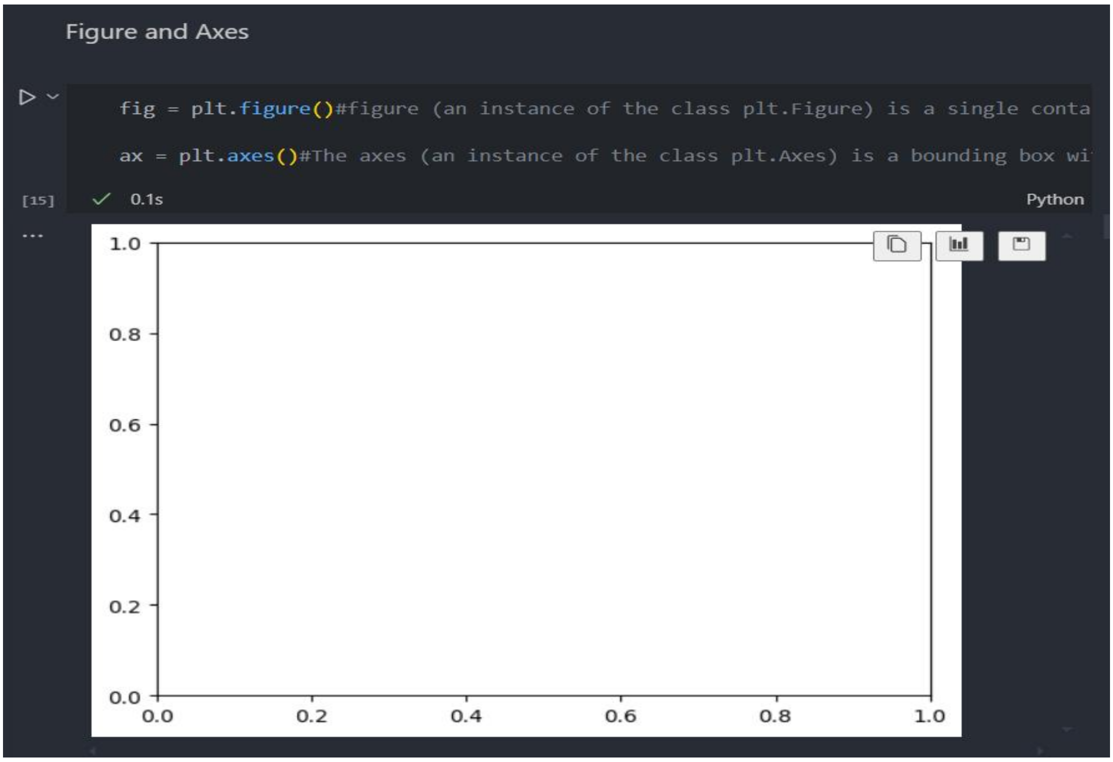
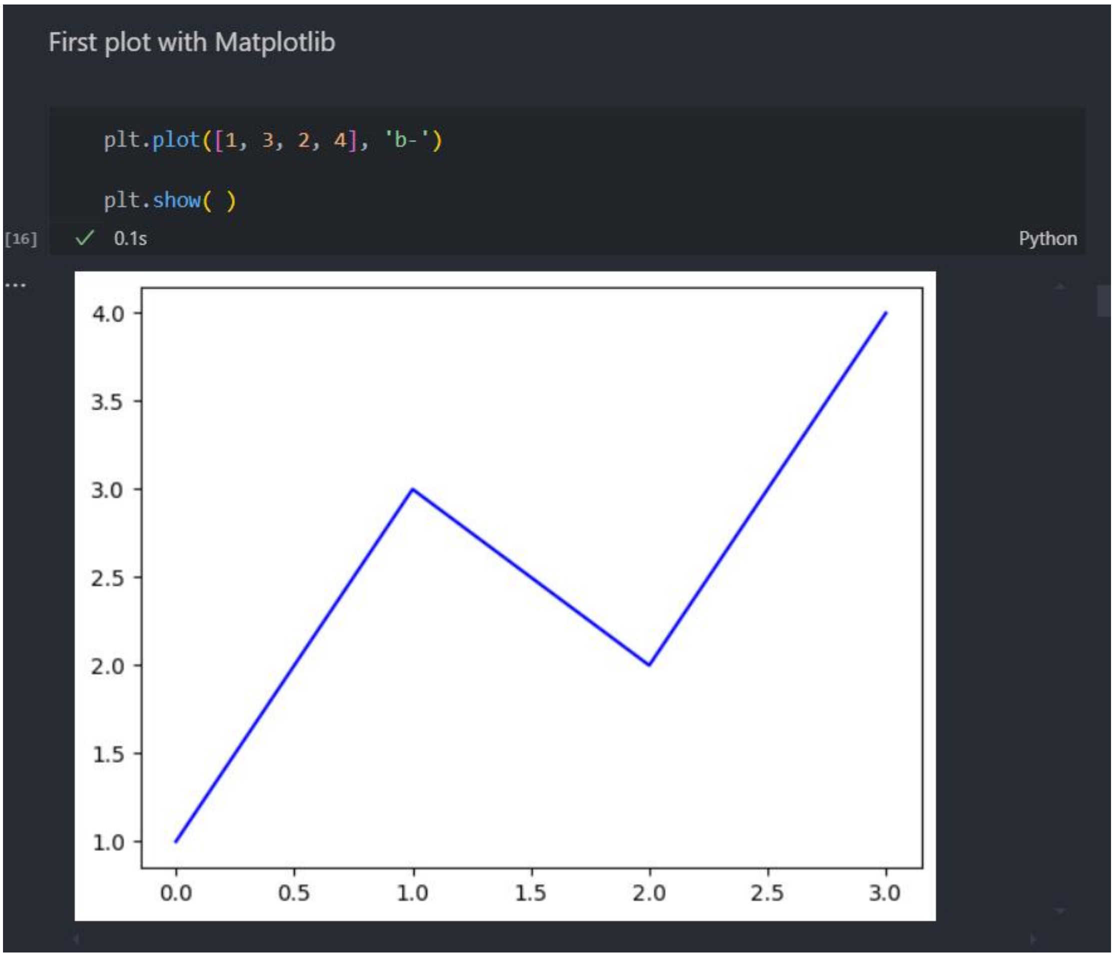


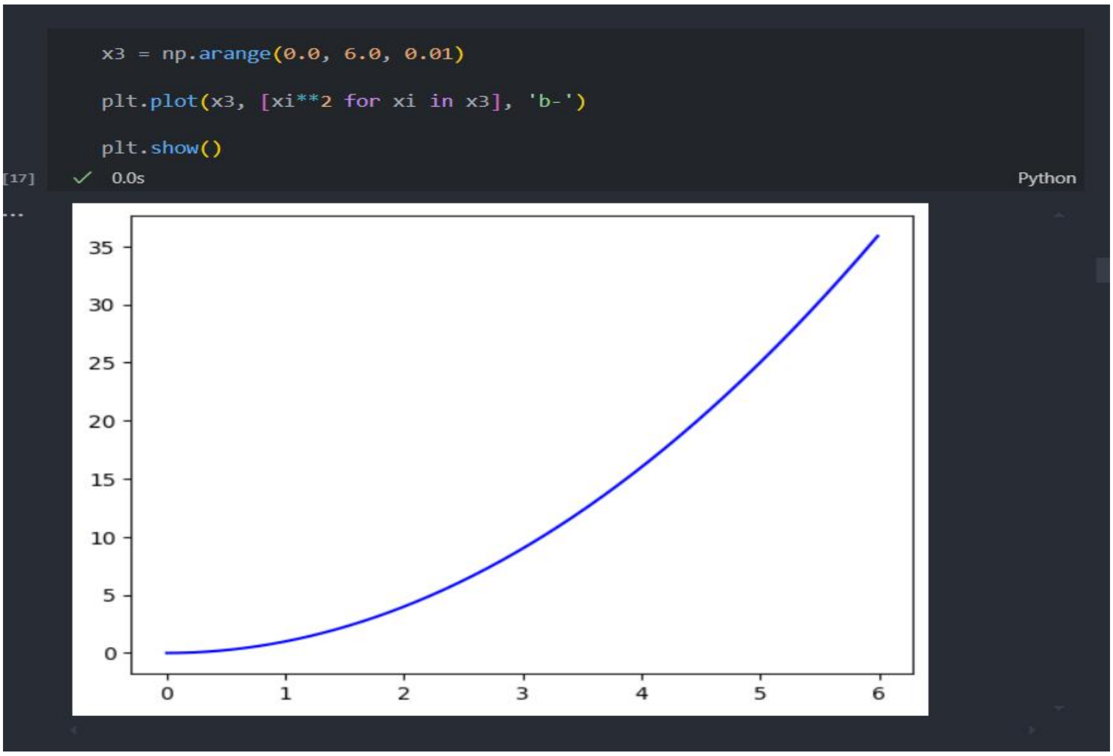
Figure and Axes

```
fig = plt.figure()#figure (an instance of the class plt.Figure) is a single conta
ax = plt.axes()#The axes (an instance of the class plt.Axes) is a bounding box wi
```

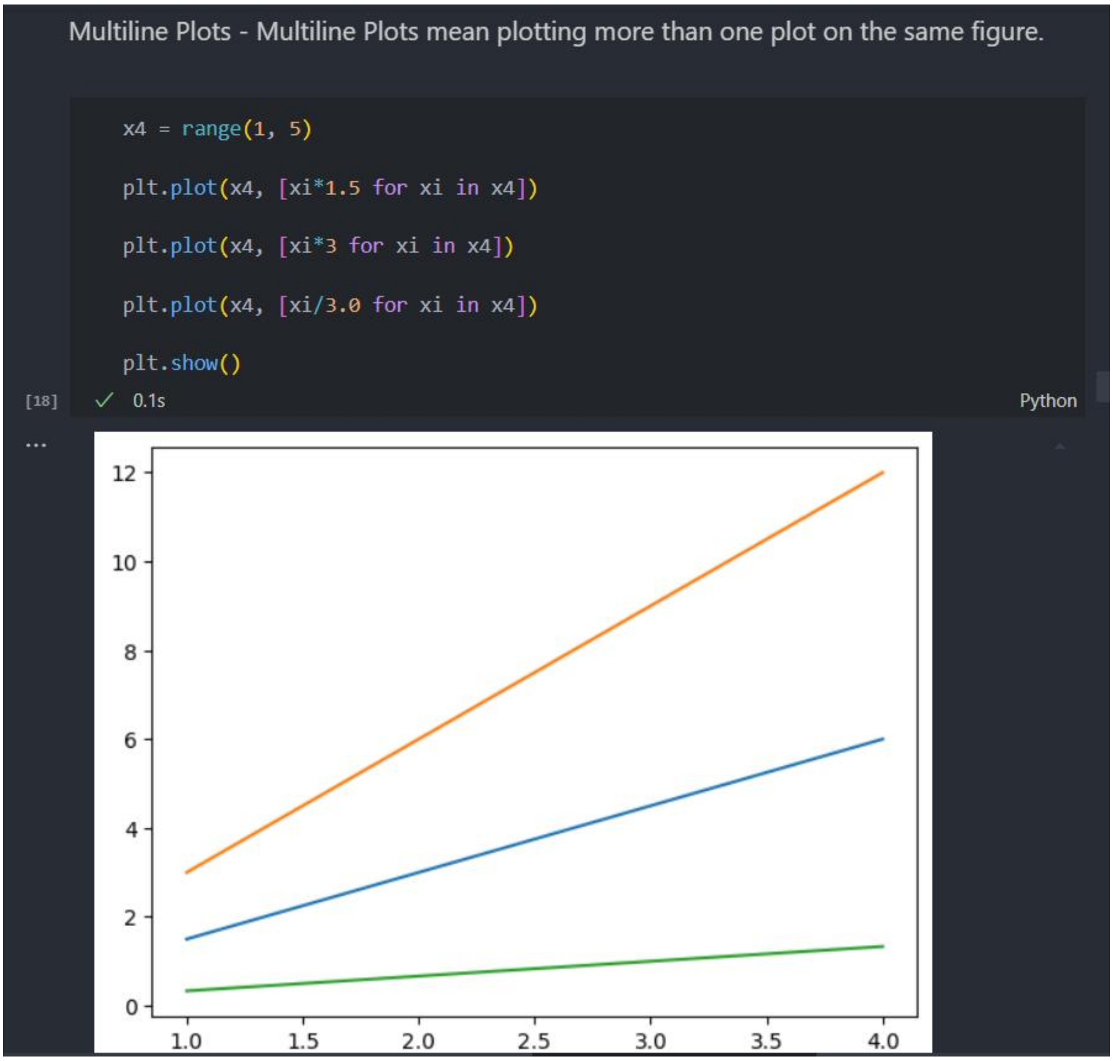


First plot with Matplotlib

```
plt.plot([1, 3, 2, 4], 'b-')
plt.show( )
```

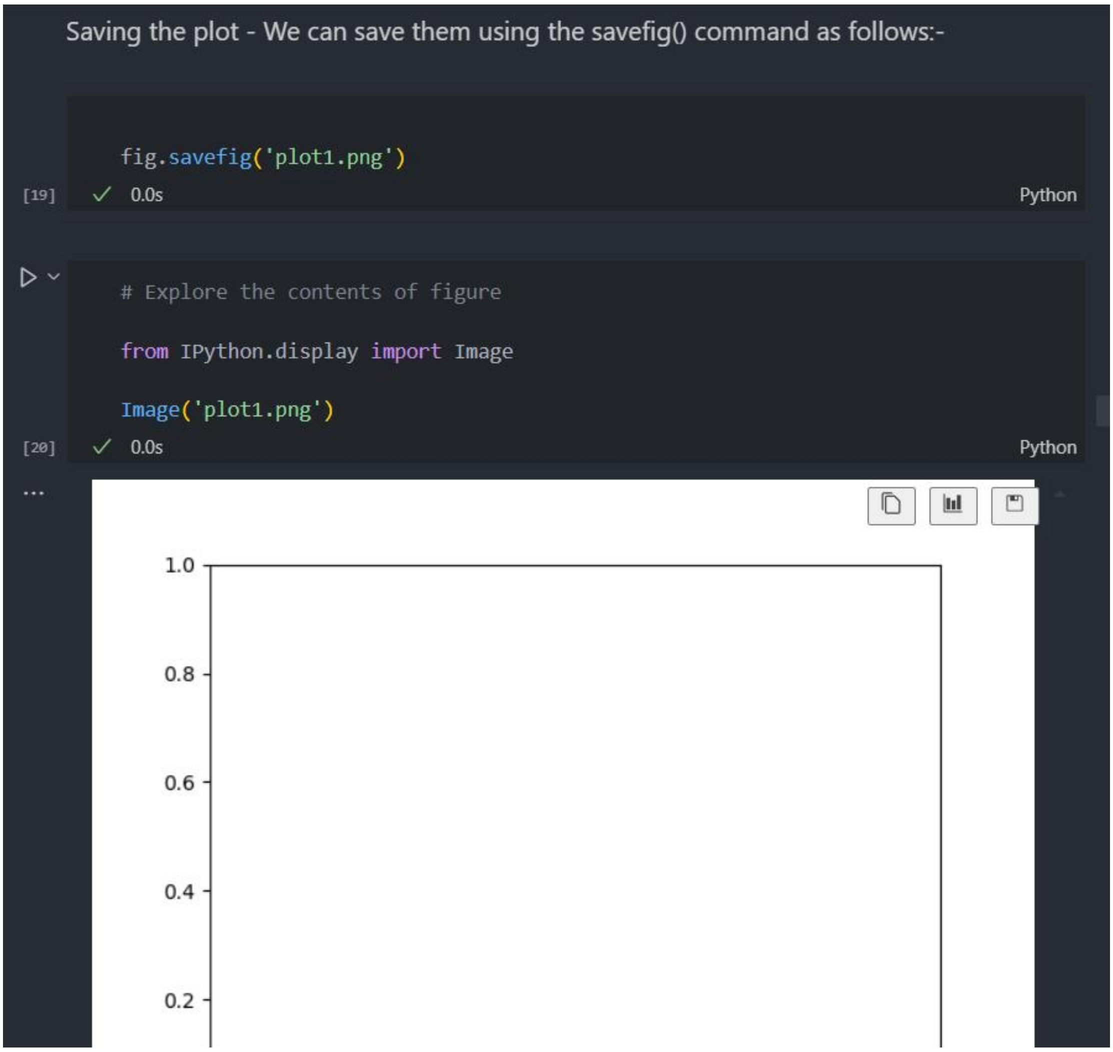


```
x3 = np.arange(0.0, 6.0, 0.01)

plt.plot(x3, [xi**2 for xi in x3], 'b-')

plt.show()
```



Multiline Plots - Multiline Plots mean plotting more than one plot on the same figure.

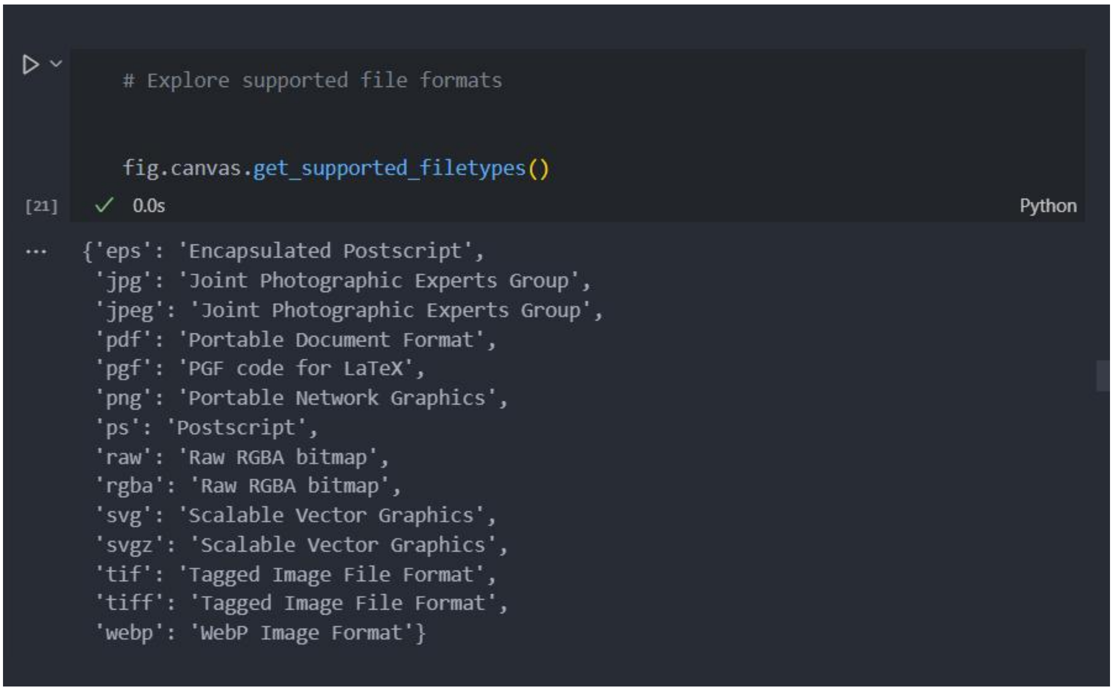
```
x4 = range(1, 5)

plt.plot(x4, [xi*1.5 for xi in x4])

plt.plot(x4, [xi*3 for xi in x4])

plt.plot(x4, [xi/3.0 for xi in x4])

plt.show()
```



Saving the plot - We can save them using the savefig() command as follows:-

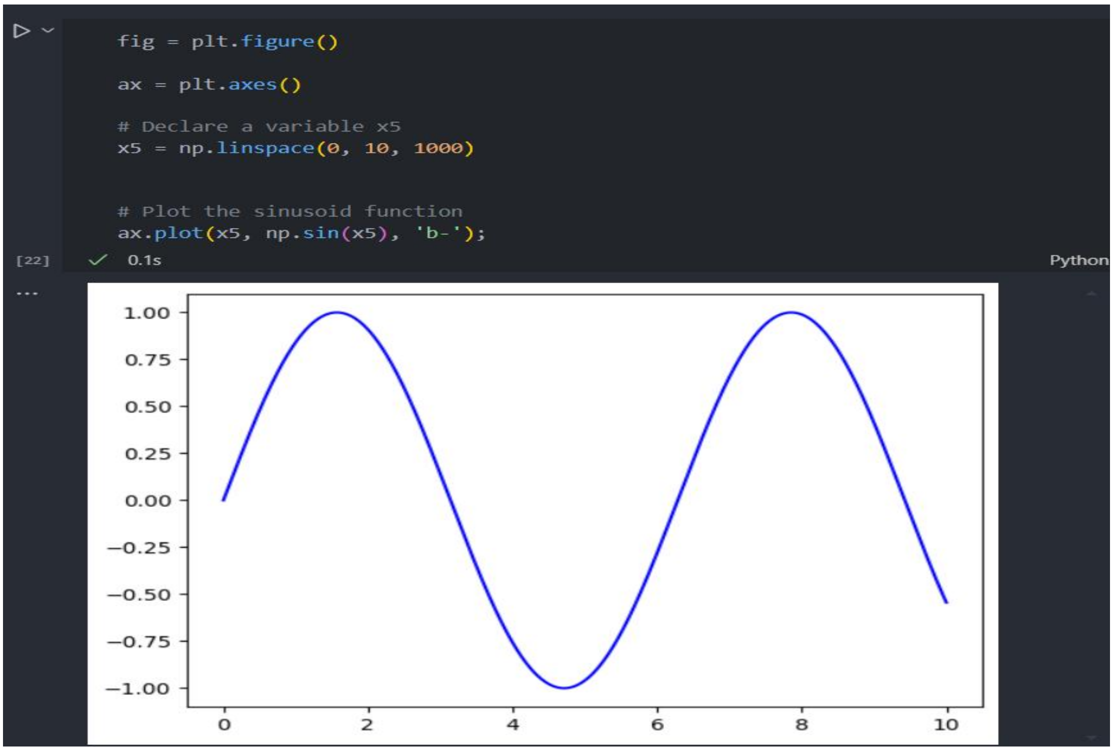
```
fig.savefig('plot1.png')
```

```
# Explore the contents of figure

from IPython.display import Image

Image('plot1.png')
```



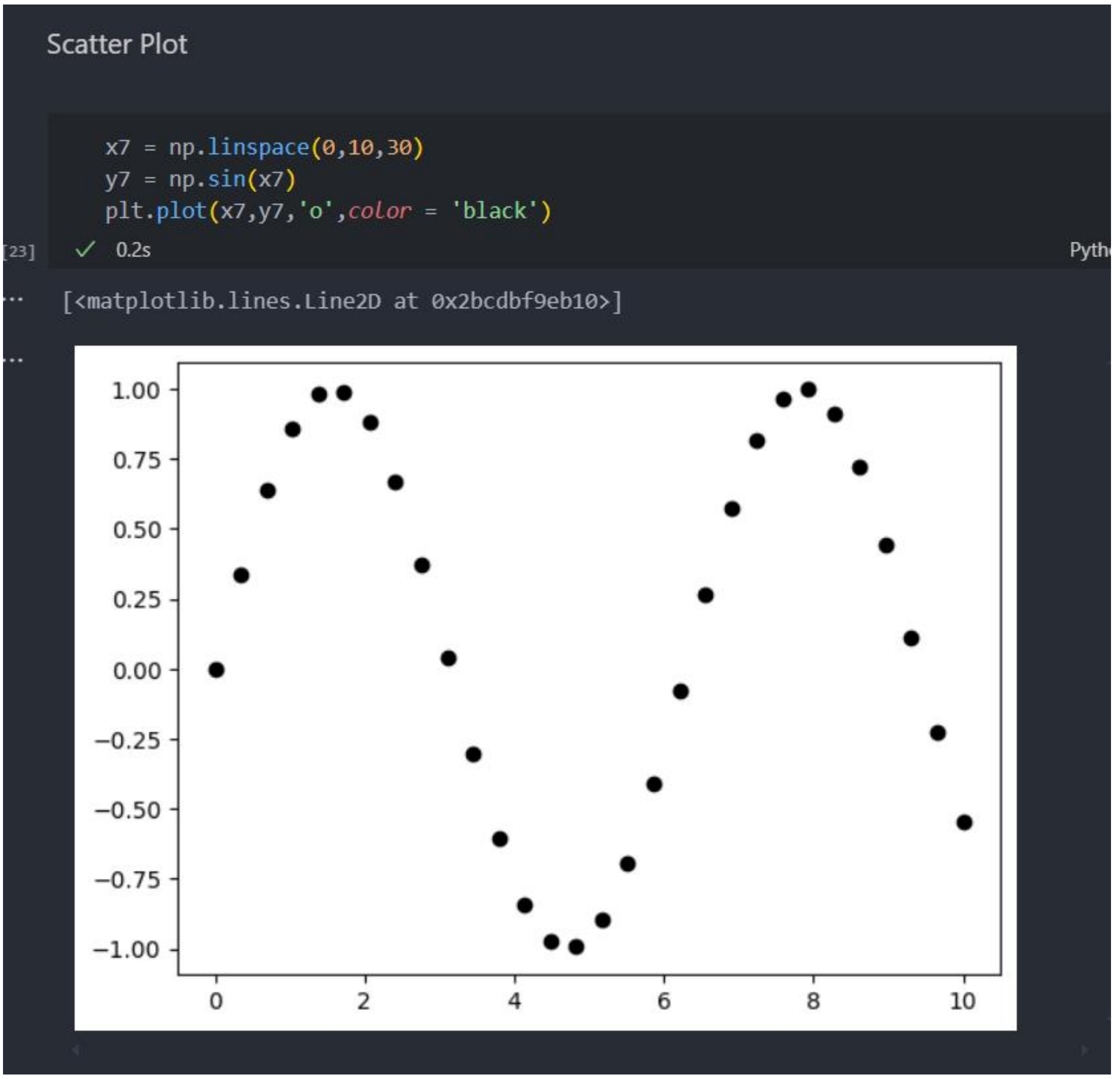
```
# Explore supported file formats


fig.canvas.get_supported_filetypes()
```

```
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format',
 'webp': 'WebP Image Format'}
```



```
fig = plt.figure()

ax = plt.axes()

# Declare a variable x5
x5 = np.linspace(0, 10, 1000)


# Plot the sinusoid function
ax.plot(x5, np.sin(x5), 'b-');
```

[22] ✓ 0.1s Python

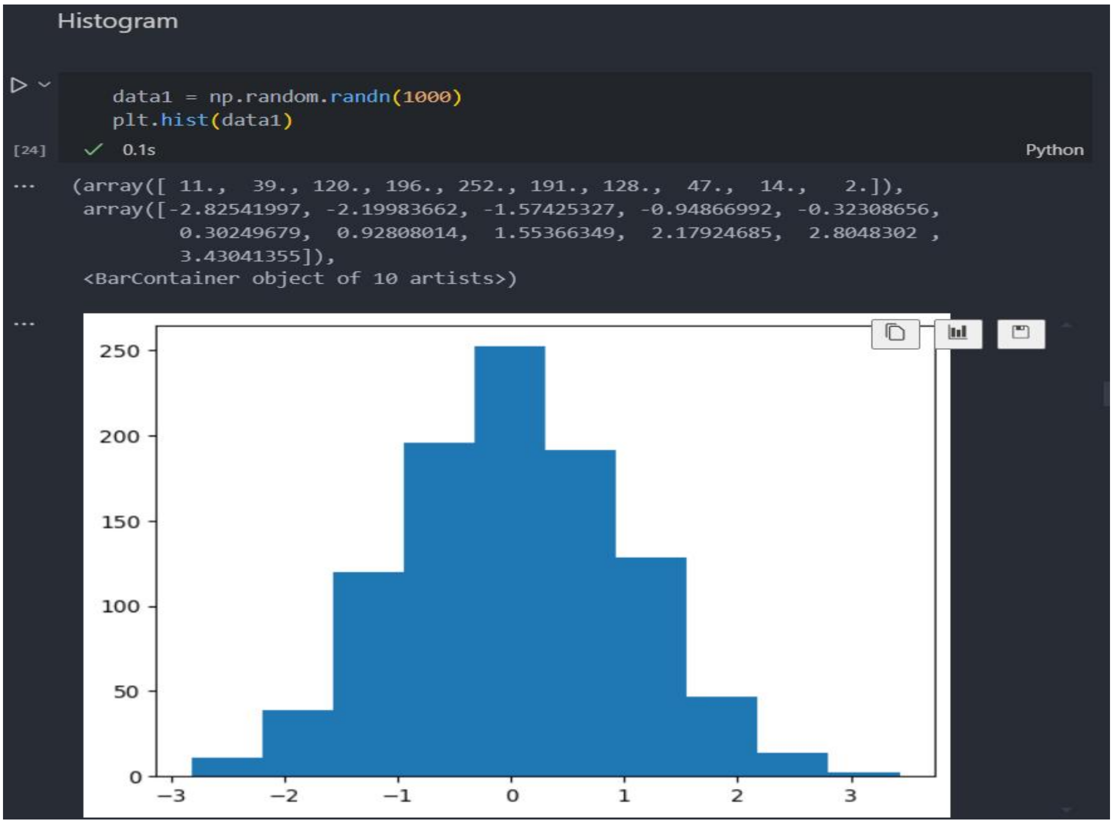


Scatter Plot

```
x7 = np.linspace(0,10,30)
y7 = np.sin(x7)
plt.plot(x7,y7,'o',color = 'black')
```

[23] ✓ 0.2s Pyth

```
[<matplotlib.lines.Line2D at 0x2bcdbf9eb10>]
```

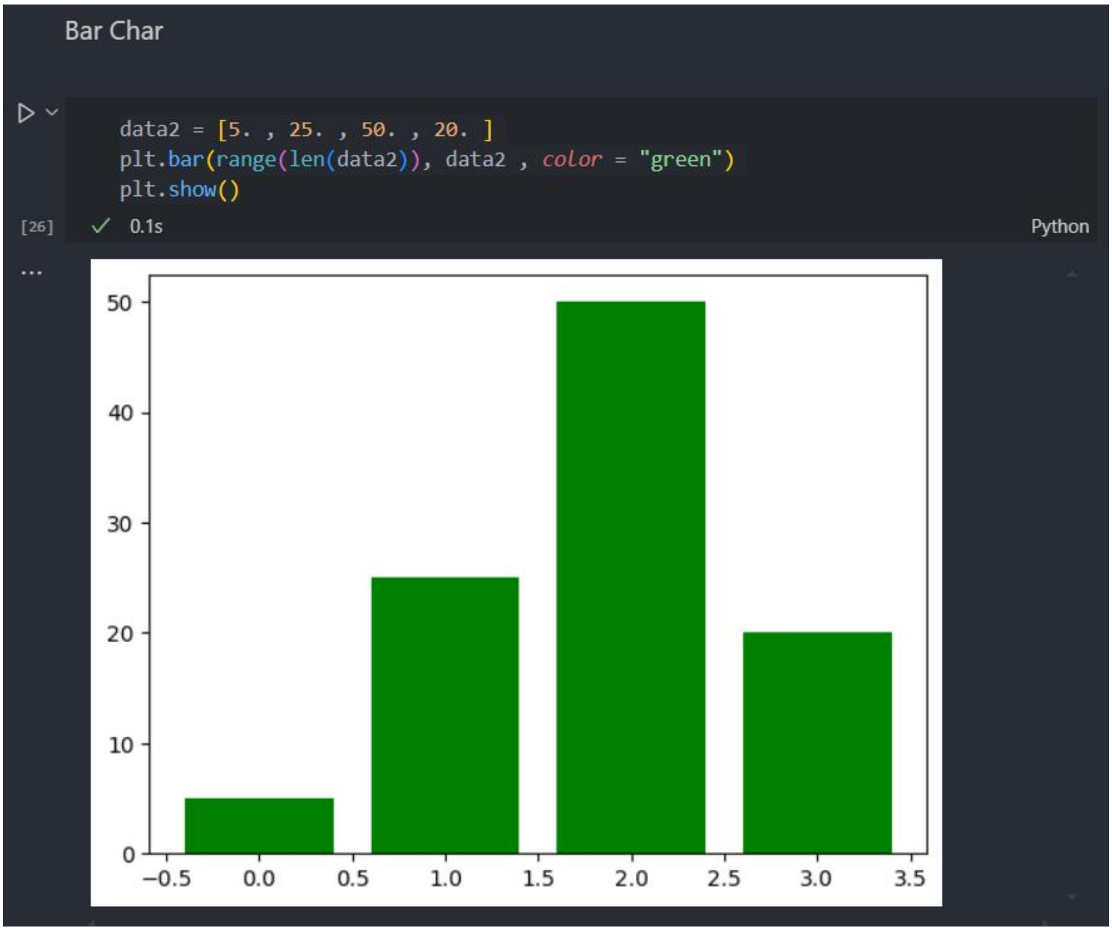


Histogram

```
data1 = np.random.randn(1000)
plt.hist(data1)
```

[24] ✓ 0.1s Python

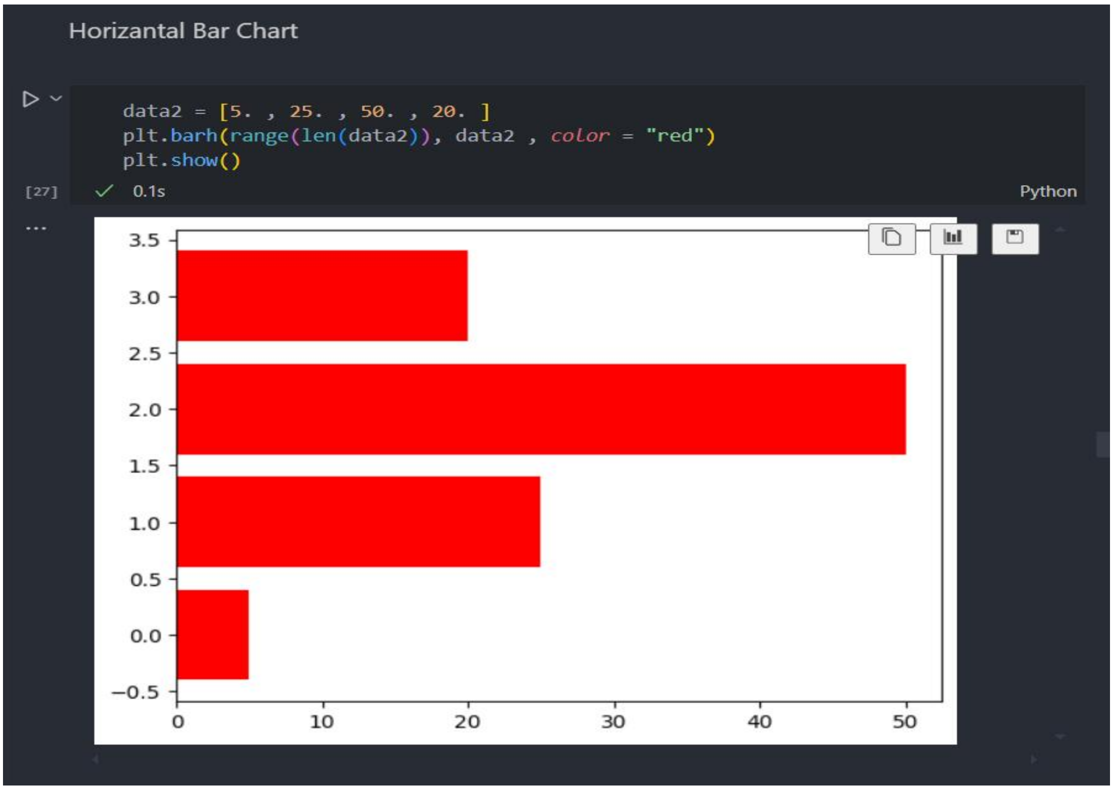
```
(array([ 11.,  39., 120., 196., 252., 191., 128.,  47.,  14.,   2.]),
 array([-2.82541997, -2.19983662, -1.57425327, -0.94866992, -0.32308656,
         0.30249679,  0.92808014,  1.55366349,  2.17924685,  2.8048302 ,
         3.43041355]),
 <BarContainer object of 10 artists>)
```



Bar Char

```
data2 = [5. , 25. , 50. , 20. ]
plt.bar(range(len(data2)), data2 , color = "green")
plt.show()
```

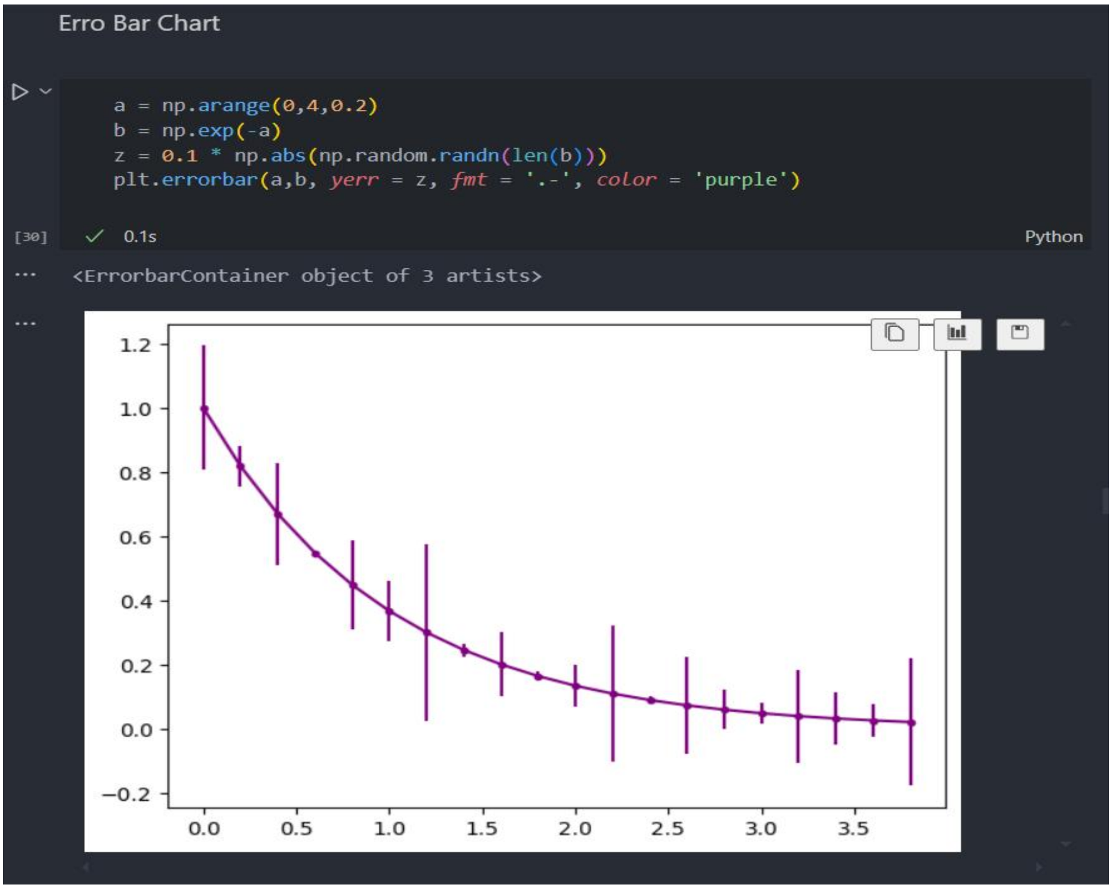
[26] ✓ 0.1s Python



Horizantal Bar Chart

```
data2 = [5. , 25. , 50. , 20. ]
plt.barh(range(len(data2)), data2 , color = "red")
plt.show()
```

[27] ✓ 0.1s Python

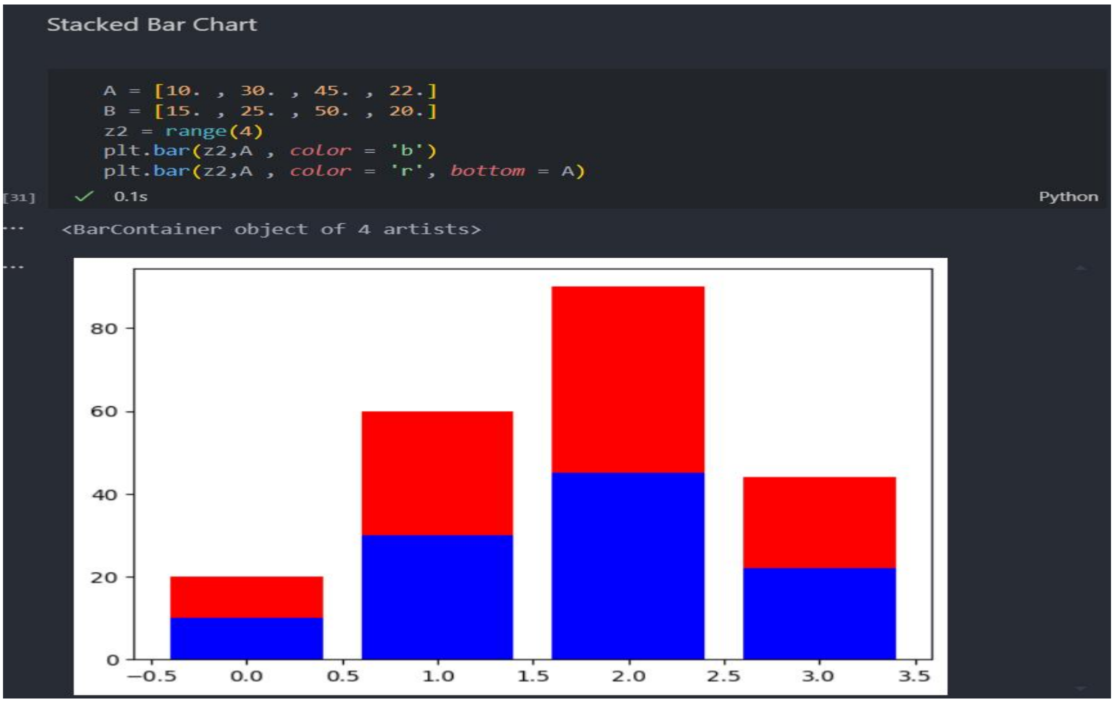


Erro Bar Chart

```
a = np.arange(0,4,0.2)
b = np.exp(-a)
z = 0.1 * np.abs(np.random.randn(len(b)))
plt.errorbar(a,b, yerr = z, fmt = '.-', color = 'purple')
```

[30] ✓ 0.1s Python

```
<ErrorbarContainer object of 3 artists>
```

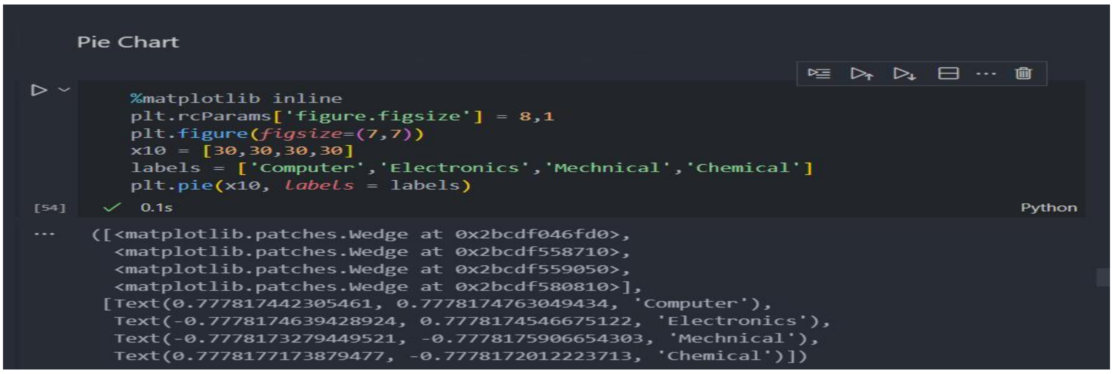


Stacked Bar Chart

```
A = [10. , 30. , 45. , 22.]
B = [15. , 25. , 50. , 20.]
z2 = range(4)
plt.bar(z2,A , color = 'b')
plt.bar(z2,A , color = 'r', bottom = A)
```

[31] ✓ 0.1s Python

```
<BarContainer object of 4 artists>
```

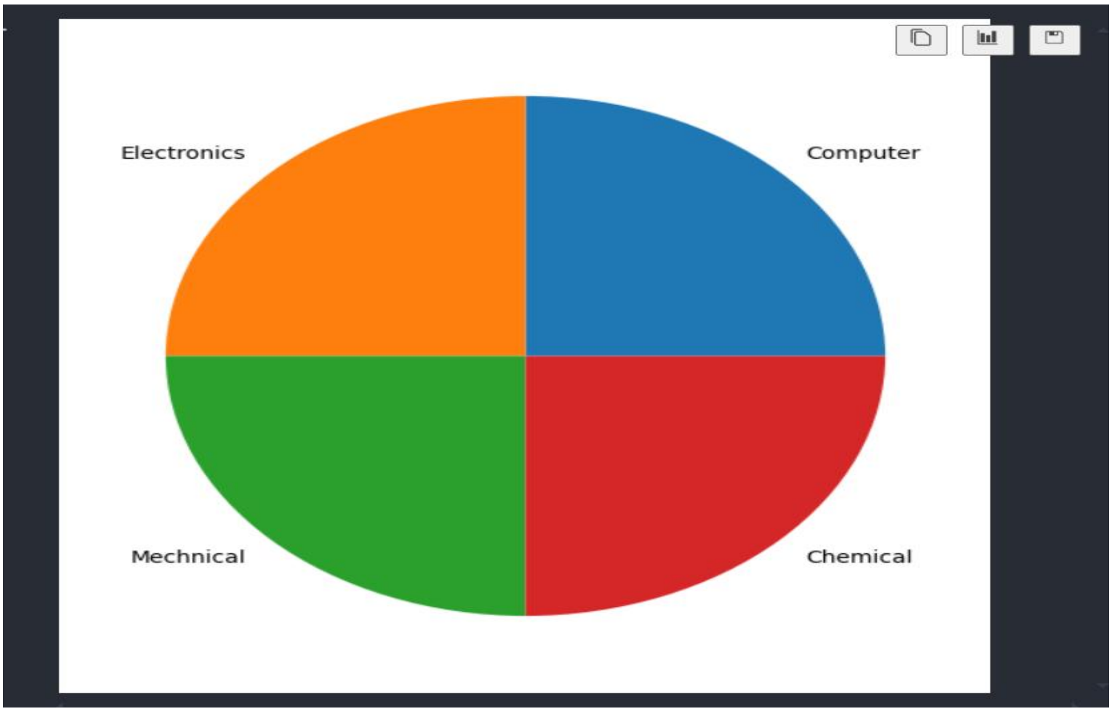


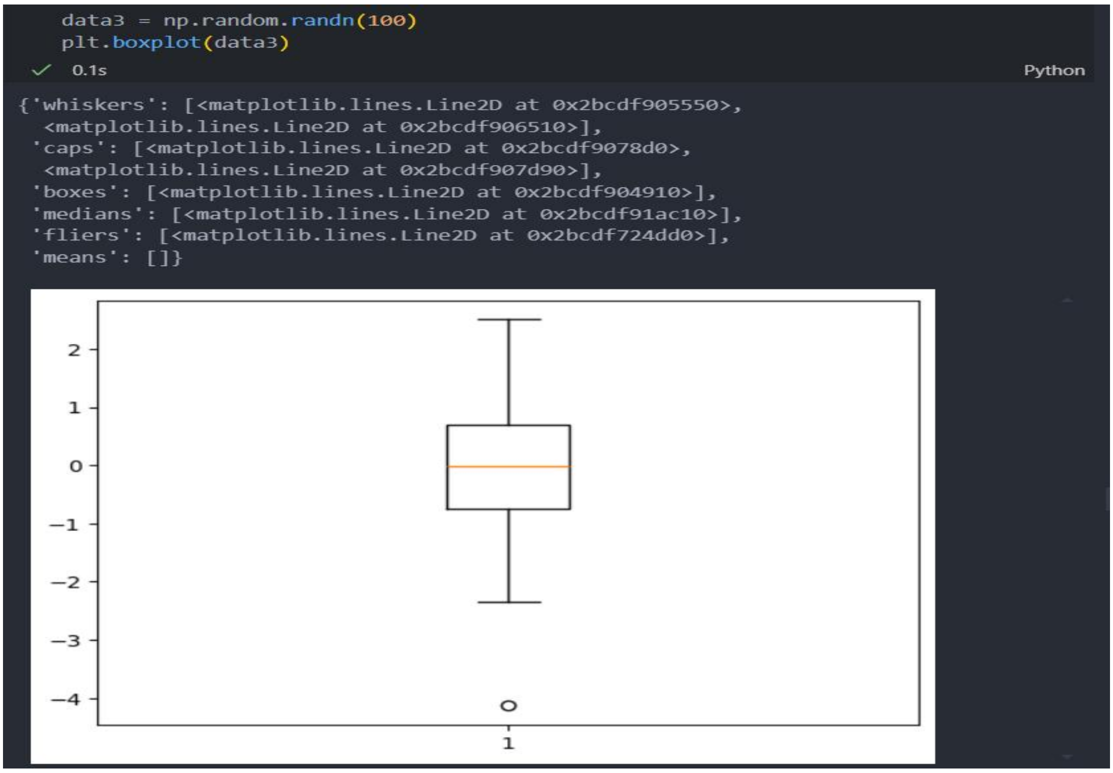
Pie Chart

```
%matplotlib inline
plt.rcParams['figure.figsize'] = 8,1
plt.figure(figsize=(7,7))
x10 = [30,30,30,30]
labels = ['Computer','Electronics','Mechnical','Chemical']
plt.pie(x10, labels = labels)
```

[54] ✓ 0.1s Python

```
([<matplotlib.patches.Wedge at 0x2bcdf046fd0>,
  <matplotlib.patches.Wedge at 0x2bcdf558710>,
  <matplotlib.patches.Wedge at 0x2bcdf559050>,
  <matplotlib.patches.Wedge at 0x2bcdf580810>],
 [Text(0.777817442305461, 0.7778174763049434, 'Computer'),
  Text(-0.7778174639428924, 0.7778174546675122, 'Electronics'),
  Text(-0.7778173279449521, -0.7778175906654303, 'Mechnical'),
  Text(0.7778177173879477, -0.7778172012223713, 'Chemical')])
```



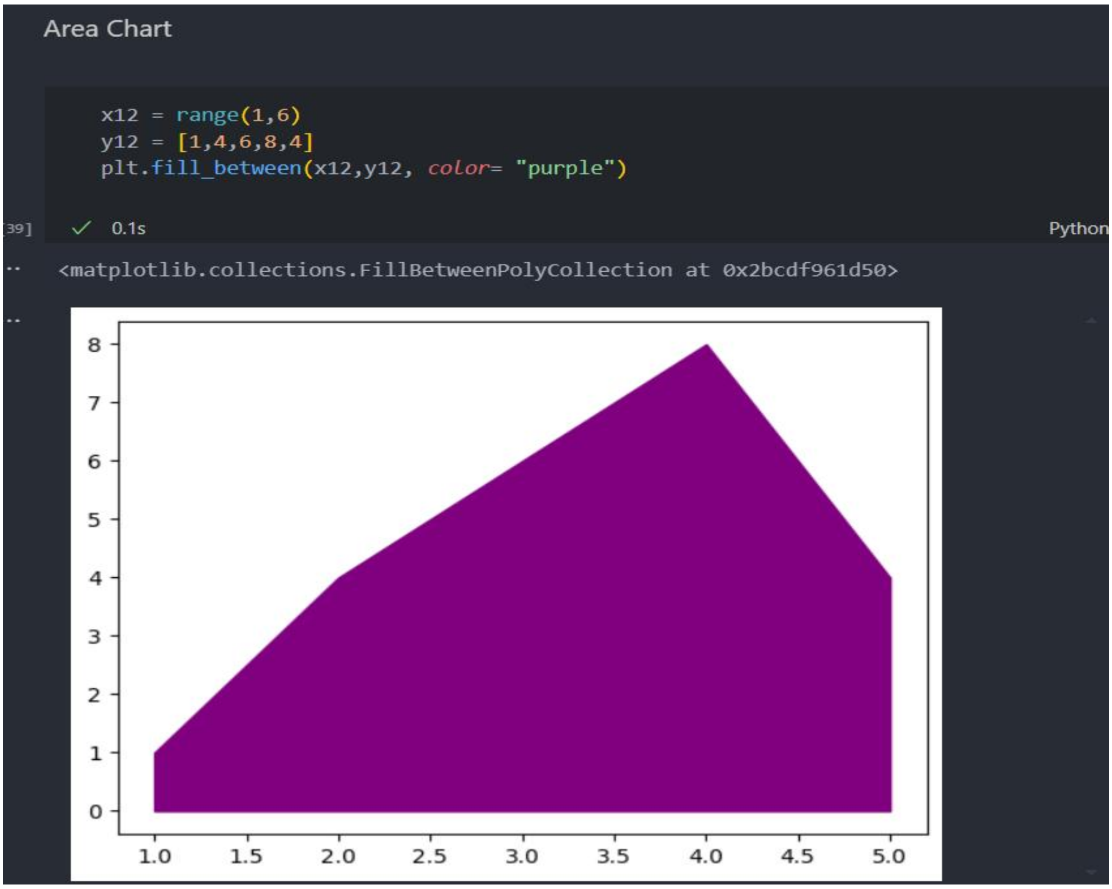


```
data3 = np.random.randn(100)
plt.boxplot(data3)
```

✓ 0.1s

Python

```
{'whiskers': [<matplotlib.lines.Line2D at 0x2bcdf905550>,
  <matplotlib.lines.Line2D at 0x2bcdf906510>],
 'caps': [<matplotlib.lines.Line2D at 0x2bcdf9078d0>,
  <matplotlib.lines.Line2D at 0x2bcdf907d90>],
 'boxes': [<matplotlib.lines.Line2D at 0x2bcdf904910>],
 'medians': [<matplotlib.lines.Line2D at 0x2bcdf91ac10>],
 'fliers': [<matplotlib.lines.Line2D at 0x2bcdf724dd0>],
 'means': []}
```



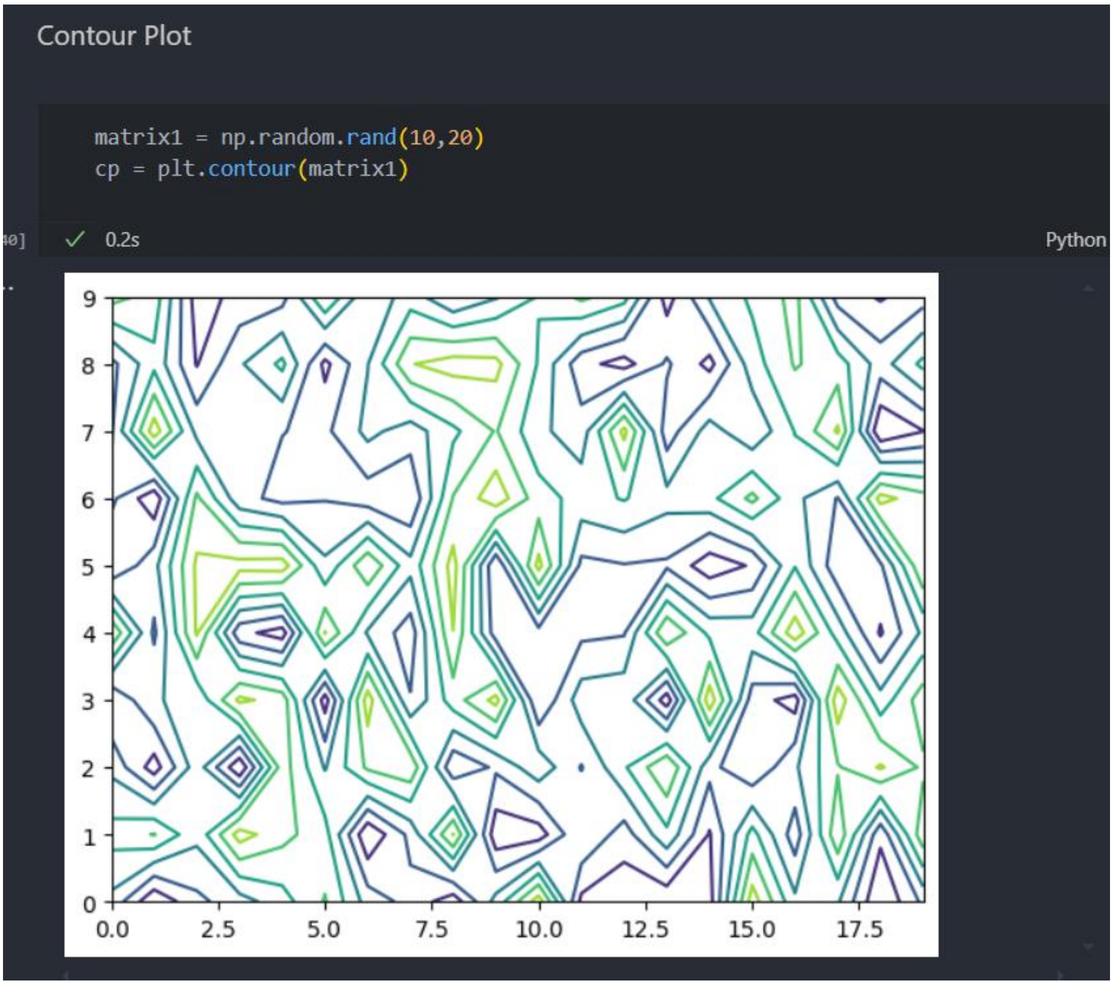
Area Chart

```
x12 = range(1,6)
y12 = [1,4,6,8,4]
plt.fill_between(x12,y12, color= "purple")
```

[39] ✓ 0.1s

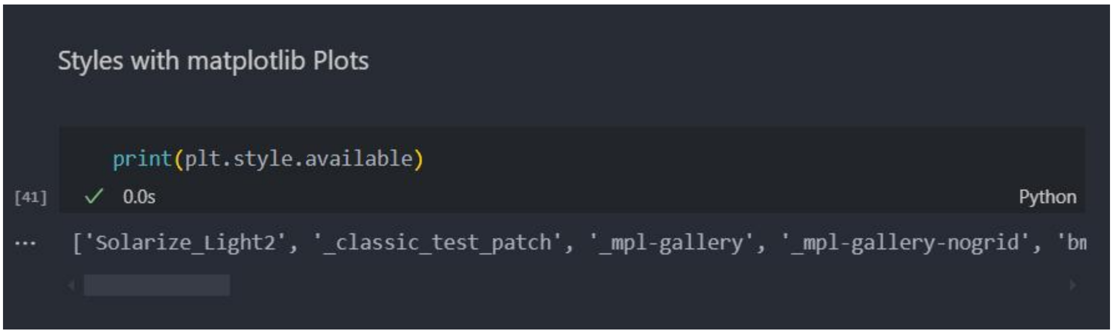
Python

```
<matplotlib.collections.FillBetweenPolyCollection at 0x2bcdf961d50>
```



Contour Plot

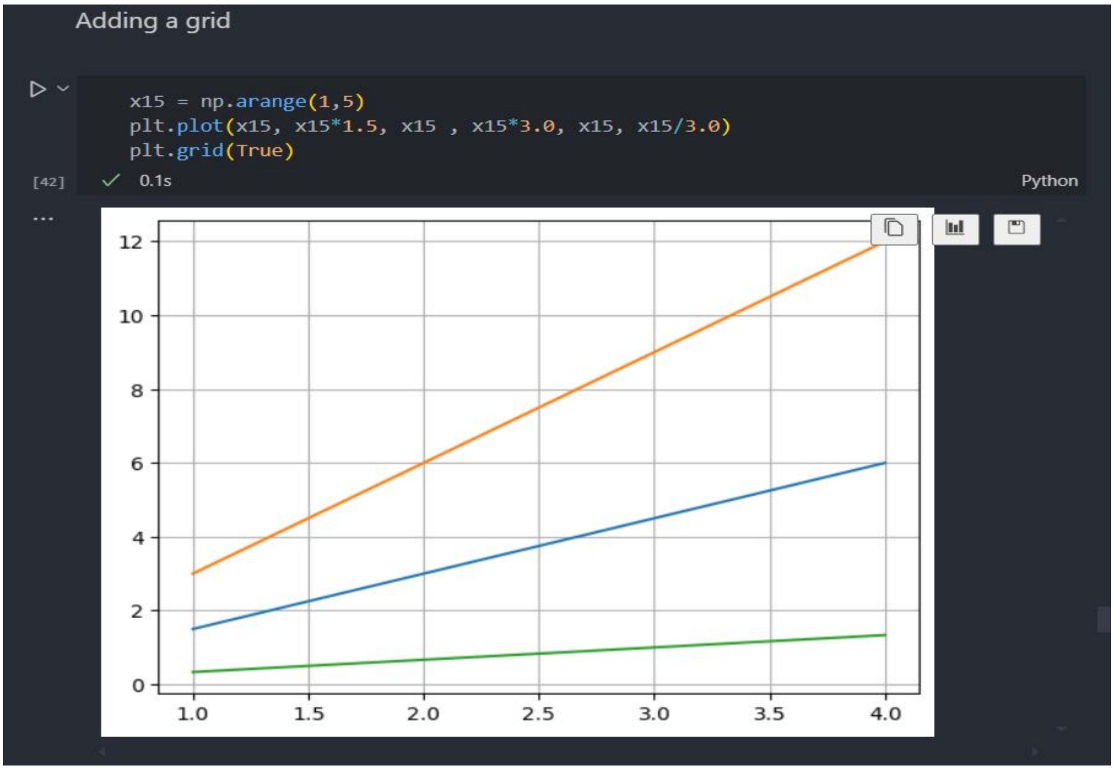
```
matrix1 = np.random.rand(10,20)
cp = plt.contour(matrix1)
```



Styles with matplotlib Plots

```
print(plt.style.available)
```

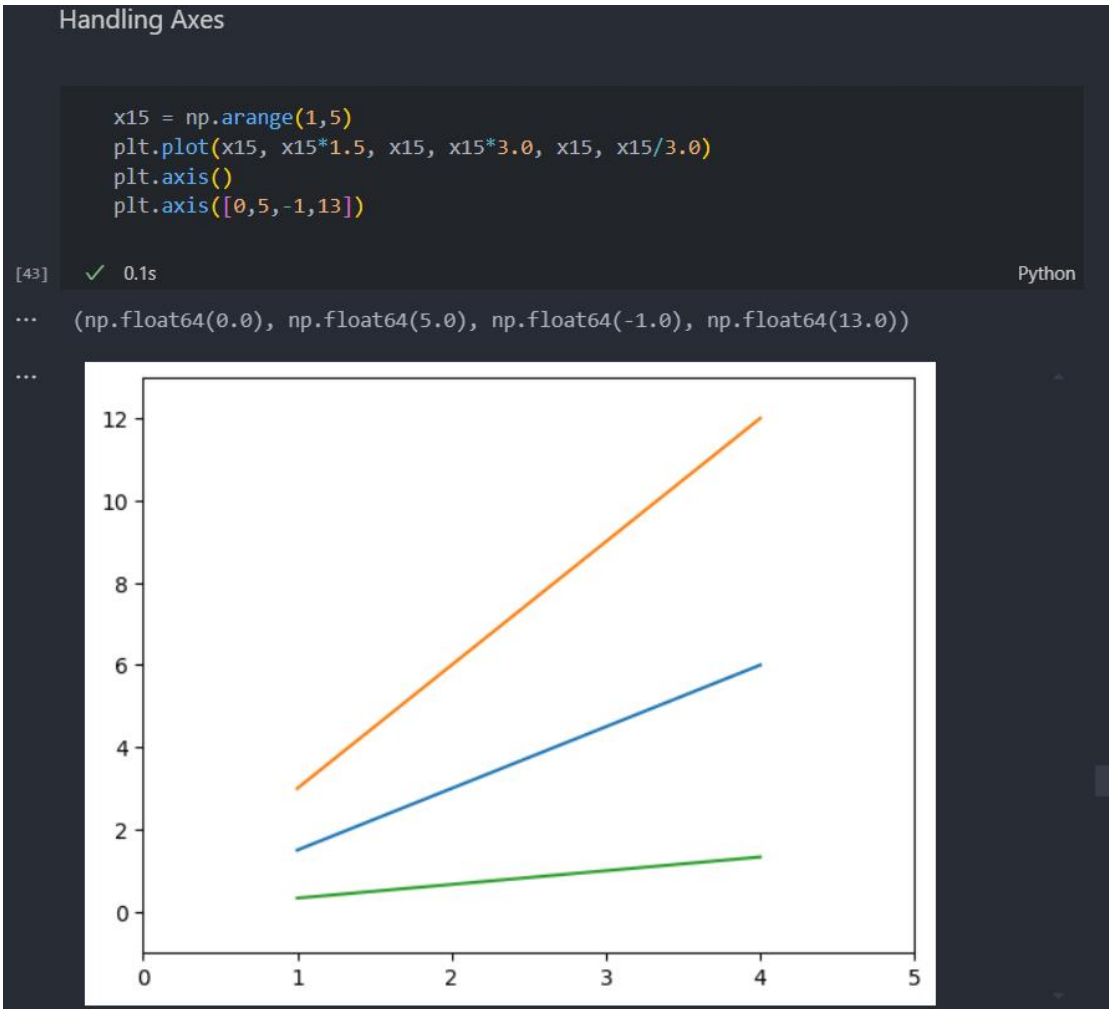
```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'bm
```



Adding a grid

```
x15 = np.arange(1,5)
plt.plot(x15, x15*1.5, x15 , x15*3.0, x15, x15/3.0)
plt.grid(True)
```

[42] ✓ 0.1s Python

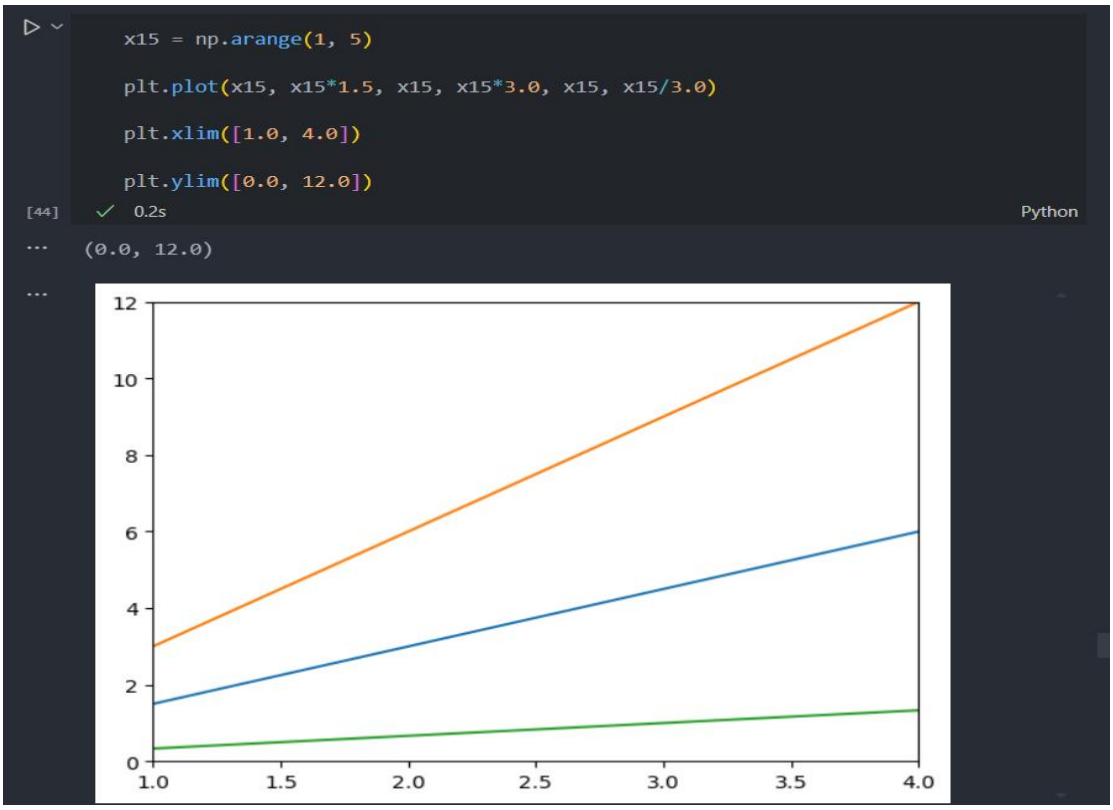


Handling Axes

```
x15 = np.arange(1,5)
plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0)
plt.axis()
plt.axis([0,5,-1,13])
```

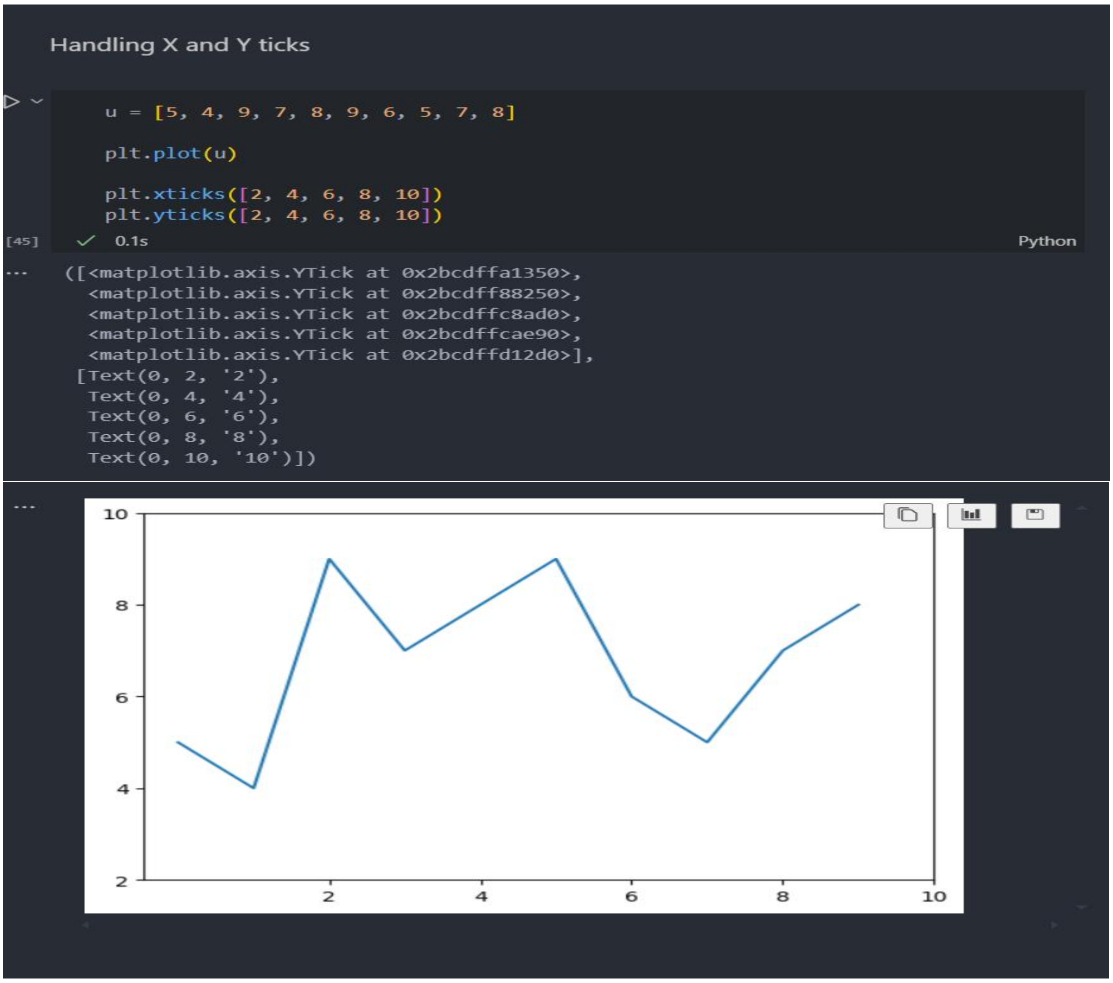
[43] ✓ 0.1s Python

```
(np.float64(0.0), np.float64(5.0), np.float64(-1.0), np.float64(13.0))
```



```
x15 = np.arange(1, 5)

plt.plot(x15, x15*1.5, x15, x15*3.0, x15, x15/3.0)

plt.xlim([1.0, 4.0])

plt.ylim([0.0, 12.0])
```

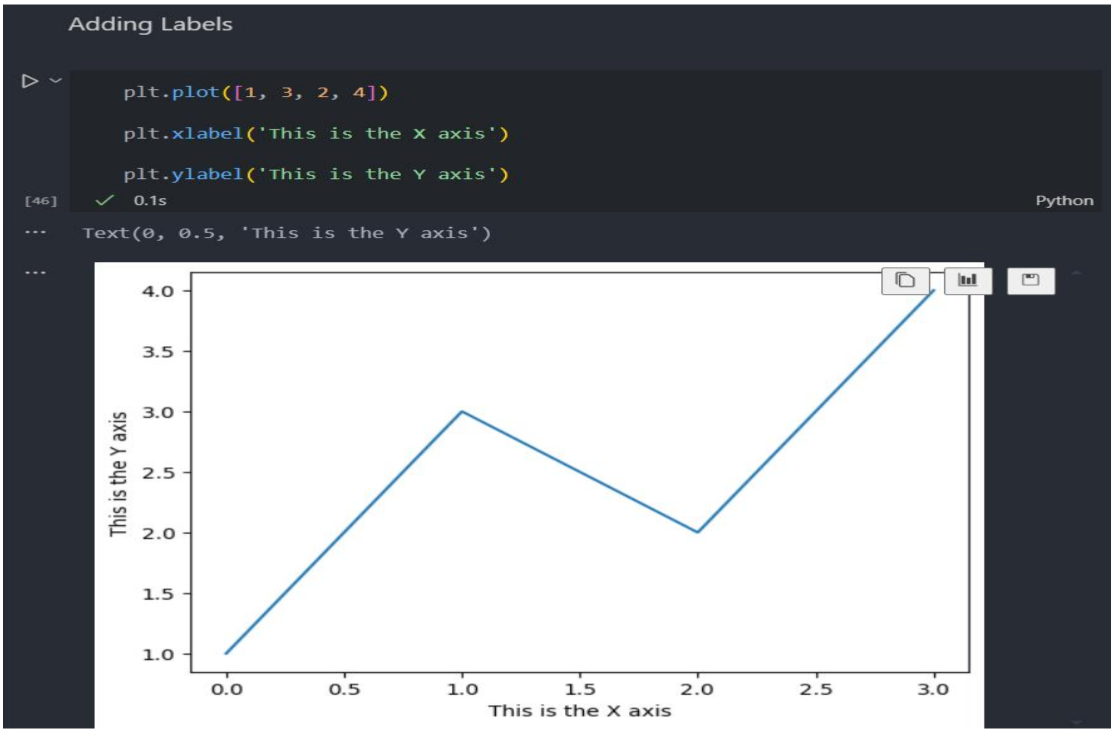
```
(0.0, 12.0)
```



Handling X and Y ticks

```
u = [5, 4, 9, 7, 8, 9, 6, 5, 7, 8]

plt.plot(u)

plt.xticks([2, 4, 6, 8, 10])
plt.yticks([2, 4, 6, 8, 10])
```

```
([<matplotlib.axis.YTick at 0x2bcdffa1350>,
  <matplotlib.axis.YTick at 0x2bcdff88250>,
  <matplotlib.axis.YTick at 0x2bcdffc8ad0>,
  <matplotlib.axis.YTick at 0x2bcdffcae90>,
  <matplotlib.axis.YTick at 0x2bcdffd12d0>],
 [Text(0, 2, '2'),
  Text(0, 4, '4'),
  Text(0, 6, '6'),
  Text(0, 8, '8'),
  Text(0, 10, '10')])
```

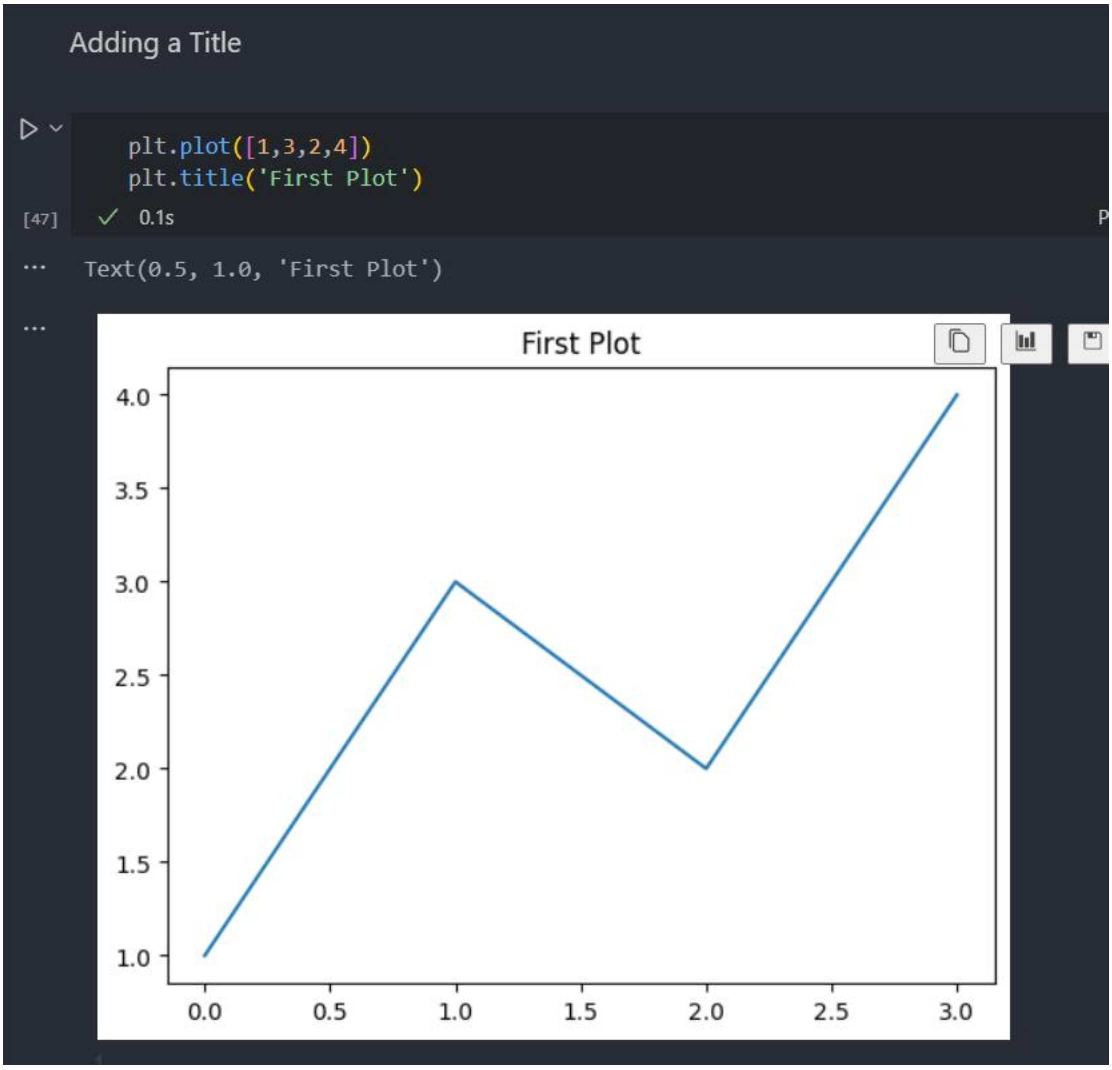


Adding Labels

```
plt.plot([1, 3, 2, 4])
plt.xlabel('This is the X axis')
plt.ylabel('This is the Y axis')
```

[46] ✓ 0.1s Python

```
Text(0, 0.5, 'This is the Y axis')
```

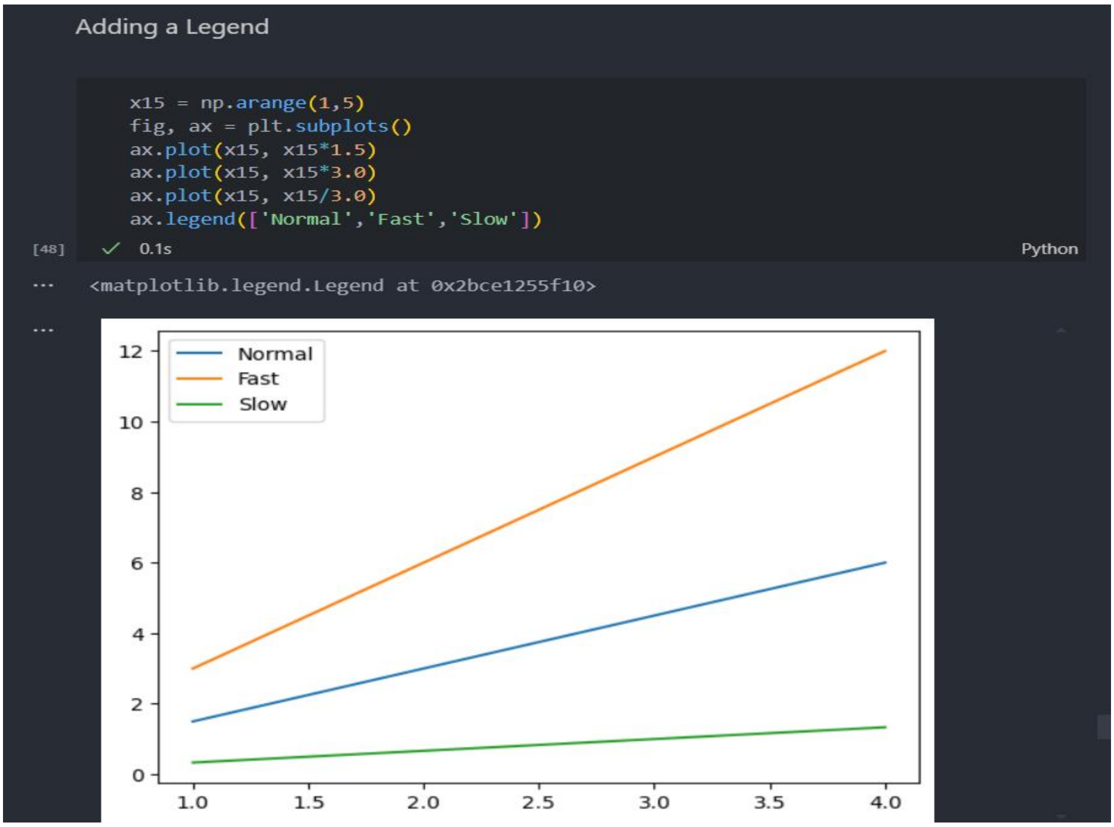


Adding a Title

```
plt.plot([1,3,2,4])
plt.title('First Plot')
```

[47] ✓ 0.1s

```
Text(0.5, 1.0, 'First Plot')
```

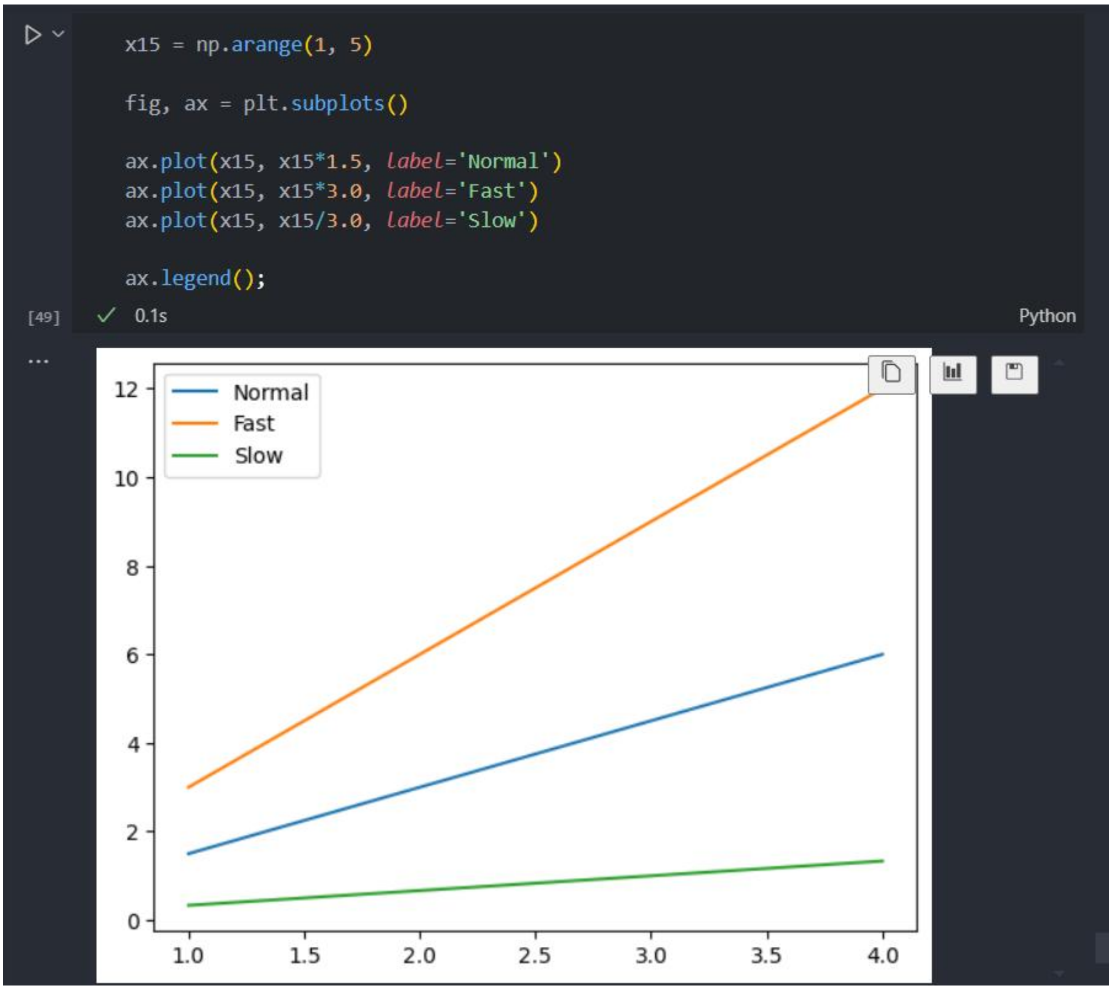


Adding a Legend

```
x15 = np.arange(1,5)
fig, ax = plt.subplots()
ax.plot(x15, x15*1.5)
ax.plot(x15, x15*3.0)
ax.plot(x15, x15/3.0)
ax.legend(['Normal','Fast','Slow'])
```

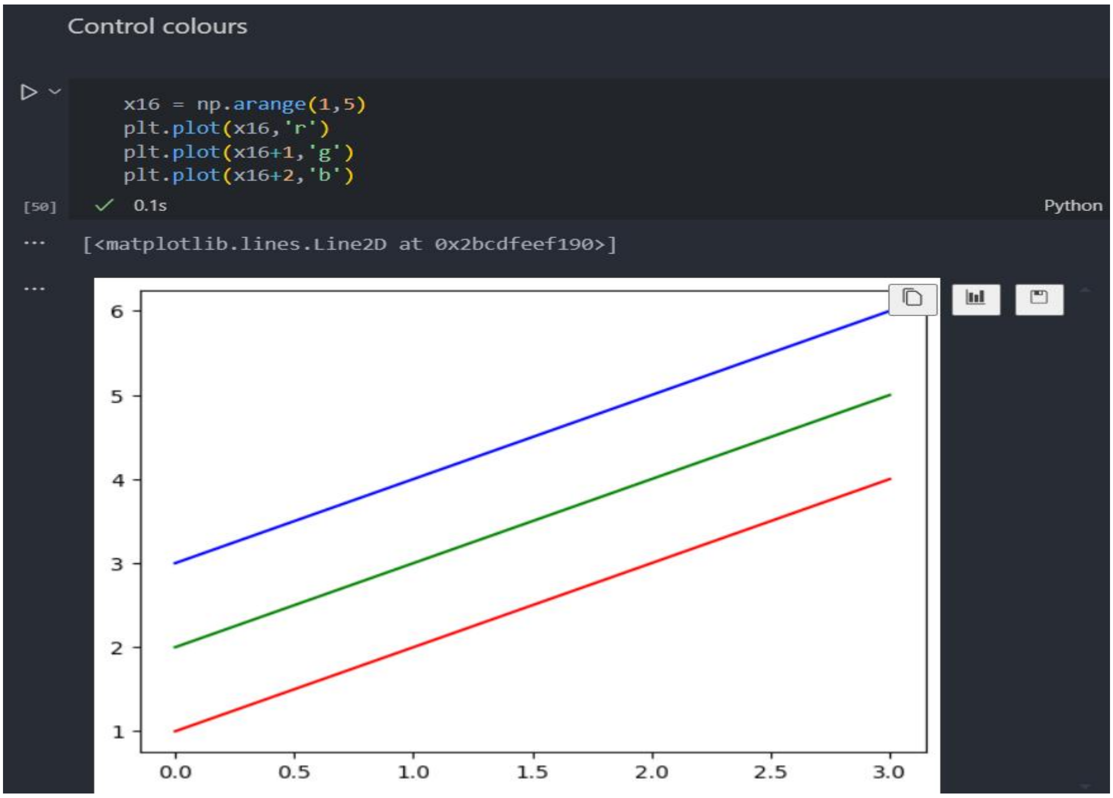
[48] ✓ 0.1s Python

```
<matplotlib.legend.Legend at 0x2bce1255f10>
```



```
x15 = np.arange(1, 5)

fig, ax = plt.subplots()

ax.plot(x15, x15*1.5, label='Normal')
ax.plot(x15, x15*3.0, label='Fast')
ax.plot(x15, x15/3.0, label='Slow')

ax.legend();
```

[49] ✓ 0.1s Python

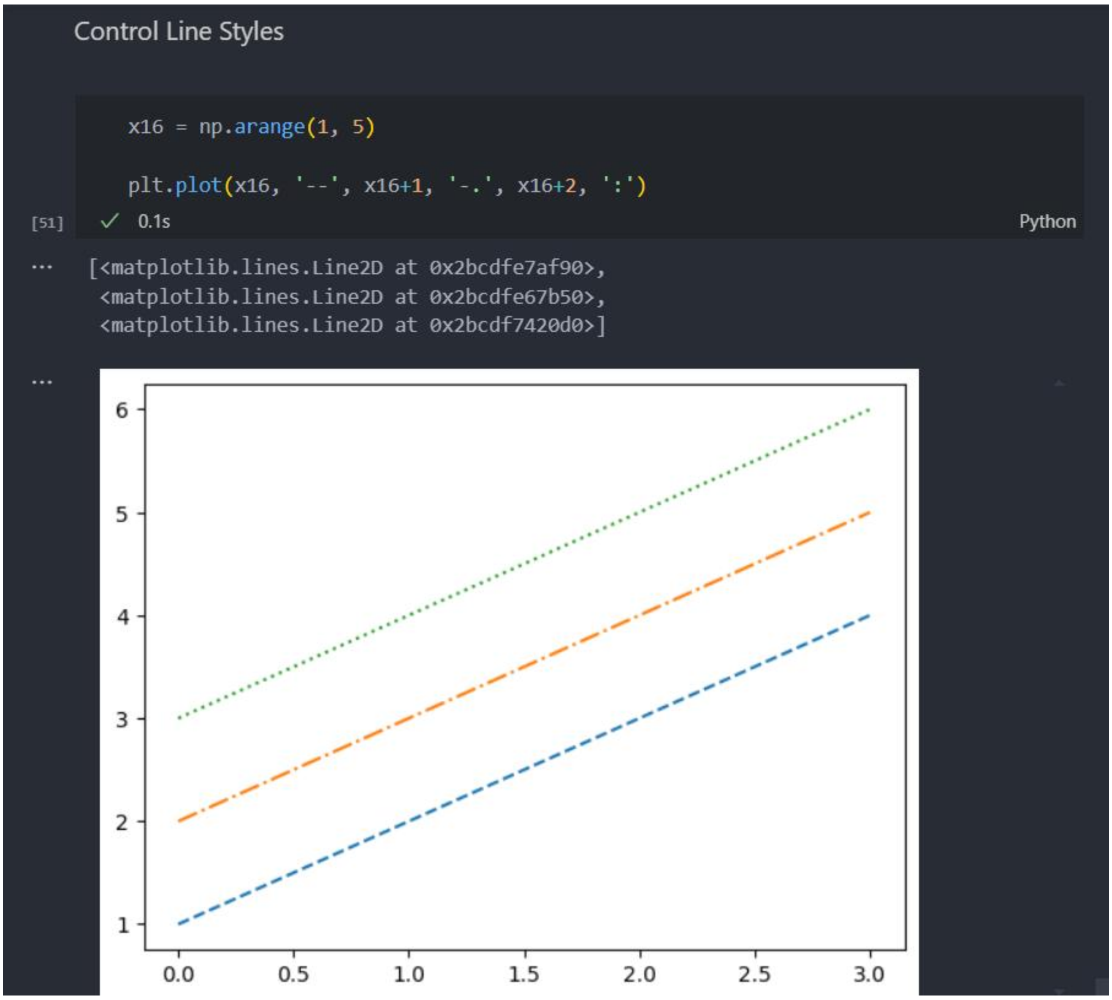


Control colours

```
x16 = np.arange(1,5)
plt.plot(x16,'r')
plt.plot(x16+1,'g')
plt.plot(x16+2,'b')
```

[50] ✓ 0.1s Python

```
[<matplotlib.lines.Line2D at 0x2bcdfeef190>]
```



Control Line Styles

```
x16 = np.arange(1, 5)

plt.plot(x16, '--', x16+1, '-.', x16+2, ':')
```

[51] ✓ 0.1s Python

```
[<matplotlib.lines.Line2D at 0x2bcdfe7af90>,
 <matplotlib.lines.Line2D at 0x2bcdfe67b50>,
 <matplotlib.lines.Line2D at 0x2bcdf7420d0>]
```