15.2_Matplotlib

April 26, 2023

1 Introduction to Python for Open Source Geocomputation

- Instructor: Dr. Wei Kang
- Class Location and Time: ENV 336, Mon & Wed 12:30 pm - 1:50 pm

Content:

- What is Matplotlib?
- Options with Matplotlib
  - Different plot types
  - Styling
  - multiple plots
- Saving a plotted figure

2 What is Matplotlib?

- One of the most popular packages for quickly creating figures in python
- Many packages are built on top of and extending Matplotlib
  - Examples
    - pandas: Tabular data analysis and manipulation tool providing a .plot() API for Matplotlib plotting.
    - animatplot: Interactive animated plots.
    - seaborn: High-level interface for drawing attractive statistical graphics.
    - GeoPandas: Pandas extended to support geographical data, algorithms, and visualization
  - See a fuller list here

2.1 Installation of Matplotlib

From a terminal:

`pip install matplotlib`

or

`conda install matplotlib`

`matplotlib` is included in conda installation, so our working environment should already have `matplotlib` installed.

2.2 Support Many Different Types of Plots

- Basic plots: lines, scatter plot, bar plot
- image plots
- statistical plots: histogram, boxplot, pie
- 3D plots

[1]:
```
from IPython.display import IFrame
IFrame(src="https://matplotlib.org/stable/plot_types/index.html", width=1000,␣
↪height=550)
```

[1]:
```
<IPython.lib.display.IFrame at 0x7f9ec97f99a0>
```

2.3 Creating a simple figure in Matplotlib

Two interfaces for plotting with Matplotlib

- implicit interface:
  - procedural approach
  - inspired by and modeled on MATLAB
  - uses an global state-based interface which is is encapsulated in the `pyplot` module to plot to the "current Axes"
  - use `functions`
  - See the pyplot tutorials for a more in-depth look at the pyplot interface.
- explicit interface:
  - object-oriented (OO) interface
  - directly utilize instances of `axes.Axes` to build up the visualization in an instance of `figure.Figure`
  - use `methods` associated with instances of classes `axes.Axes` and `figure.Figure`

[2]:
```
import matplotlib.pyplot as plt #import the pyplot module from matplotlib
```

[3]:
```
dir(plt)
```

[3]:
```
['Annotation',
 'Arrow',
 'Artist',
 'AutoLocator',
```

```
'Axes',
'Button',
'Circle',
'Enum',
'ExitStack',
'Figure',
'FigureBase',
'FigureCanvasBase',
'FigureManagerBase',
'FixedFormatter',
'FixedLocator',
'FormatStrFormatter',
'Formatter',
'FuncFormatter',
'GridSpec',
'IndexLocator',
'Line2D',
'LinearLocator',
'Locator',
'LogFormatter',
'LogFormatterExponent',
'LogFormatterMathtext',
'LogLocator',
'MaxNLocator',
'MouseButton',
'MultipleLocator',
'Normalize',
'NullFormatter',
'NullLocator',
'Number',
'PolarAxes',
'Polygon',
'Rectangle',
'ScalarFormatter',
'Slider',
'Subplot',
'SubplotSpec',
'Text',
'TickHelper',
'Widget',
'_REPL_DISPLAYHOOK',
'_ReplDisplayHook',
'__builtins__',
'__cached__',
'__doc__',
'__file__',
'__loader__',
```

```
'__name__',
'__package__',
'__spec__',
'_api',
'_auto_draw_if_interactive',
'_backend_mod',
'_copy_docstring_and_deprecators',
'_docstring',
'_draw_all_if_interactive',
'_get_backend_mod',
'_get_pyplot_commands',
'_get_required_interactive_framework',
'_interactive_bk',
'_log',
'_pylab_helpers',
'_warn_if_gui_out_of_main_thread',
'acorr',
'angle_spectrum',
'annotate',
'arrow',
'autoscale',
'autumn',
'axes',
'axhline',
'axhspan',
'axis',
'axline',
'axvline',
'axvspan',
'bar',
'bar_label',
'barbs',
'barh',
'bone',
'box',
'boxplot',
'broken_barh',
'cbook',
'cla',
'clabel',
'clf',
'clim',
'close',
'cm',
'cohere',
'color_sequences',
'colorbar',
```

```
'colormaps',
'connect',
'contour',
'contourf',
'cool',
'copper',
'csd',
'cycler',
'delaxes',
'disconnect',
'draw',
'draw_all',
'draw_if_interactive',
'errorbar',
'eventplot',
'figaspect',
'figimage',
'figlegend',
'fignum_exists',
'figtext',
'figure',
'fill',
'fill_between',
'fill_betweenx',
'findobj',
'flag',
'functools',
'gca',
'gcf',
'gci',
'get',
'get_backend',
'get_cmap',
'get_current_fig_manager',
'get_figlabels',
'get_fignums',
'get_plot_commands',
'get_scale_names',
'getp',
'ginput',
'gray',
'grid',
'hexbin',
'hist',
'hist2d',
'hlines',
'hot',
```

```
'hsv',
'importlib',
'imread',
'imsave',
'imshow',
'inferno',
'inspect',
'install_repl_displayhook',
'interactive',
'ioff',
'ion',
'isinteractive',
'jet',
'legend',
'locator_params',
'logging',
'loglog',
'magma',
'magnitude_spectrum',
'margins',
'matplotlib',
'matshow',
'minorticks_off',
'minorticks_on',
'mlab',
'new_figure_manager',
'nipy_spectral',
'np',
'pause',
'pcolor',
'pcolormesh',
'phase_spectrum',
'pie',
'pink',
'plasma',
'plot',
'plot_date',
'polar',
'prism',
'psd',
'quiver',
'quiverkey',
'rc',
'rcParams',
'rcParamsDefault',
'rcParamsOrig',
'rc_context',
```

```
'rcdefaults',
'rcsetup',
're',
'register_cmap',
'rgrids',
'savefig',
'sca',
'scatter',
'sci',
'semilogx',
'semilogy',
'set_cmap',
'set_loglevel',
'setp',
'show',
'specgram',
'spring',
'spy',
'stackplot',
'stairs',
'stem',
'step',
'streamplot',
'style',
'subplot',
'subplot2grid',
'subplot_mosaic',
'subplot_tool',
'subplots',
'subplots_adjust',
'summer',
'suptitle',
'switch_backend',
'sys',
'table',
'text',
'thetagrids',
'threading',
'tick_params',
'ticklabel_format',
'tight_layout',
'time',
'title',
'tricontour',
'tricontourf',
'tripcolor',
'triplot',
```

```
 'twinx',
 'twiny',
 'uninstall_repl_displayhook',
 'violinplot',
 'viridis',
 'vlines',
 'waitforbuttonpress',
 'winter',
 'xcorr',
 'xkcd',
 'xlabel',
 'xlim',
 'xscale',
 'xticks',
 'ylabel',
 'ylim',
 'yscale',
 'yticks']
```

[4]:
```
import numpy as np
```

[5]:
```
x = np.arange(10)
y = x * 2
```

[6]:
```
x
```

[6]:
```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

[7]:
```
y
```

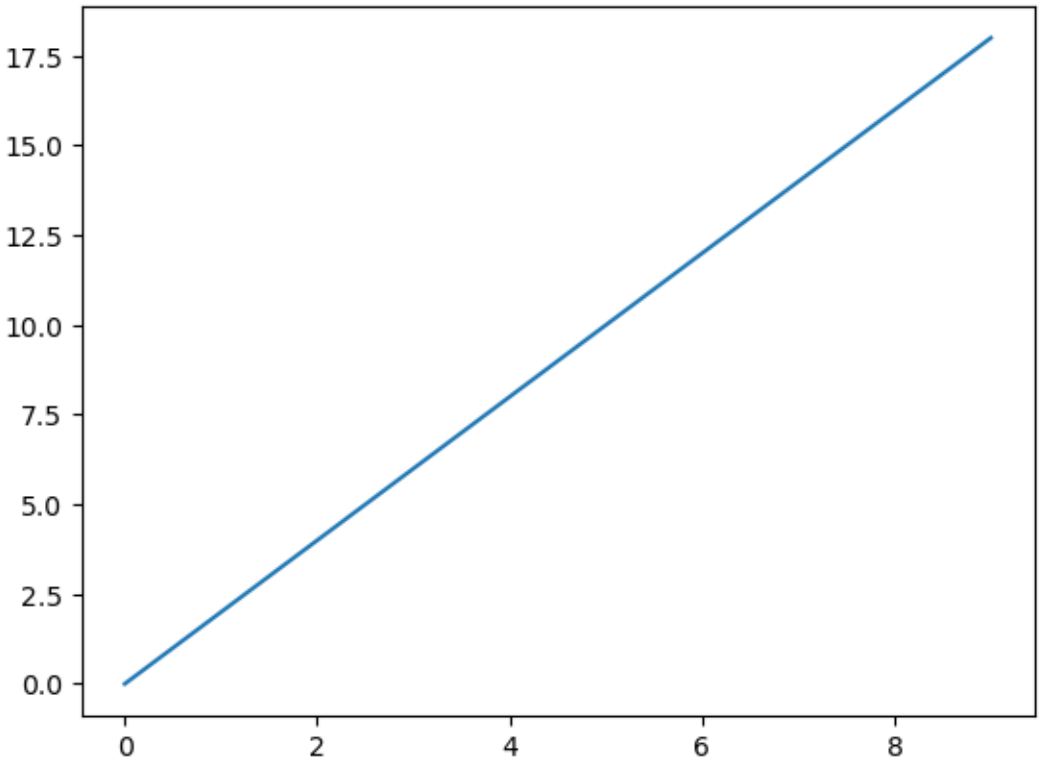
[7]:
```
array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

Explicit interface This interface works by

- instantiating an instance of a Figure class (fig below)
- using a method subplots method (or similar) on that object to create one or more Axes objects (ax below)
- then calling drawing methods on the Axes (plot in this example)

[8]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x, y)  # Plot some data on the axes.
```

[8]:
```
[<matplotlib.lines.Line2D at 0x7f9eb81b8280>]
```

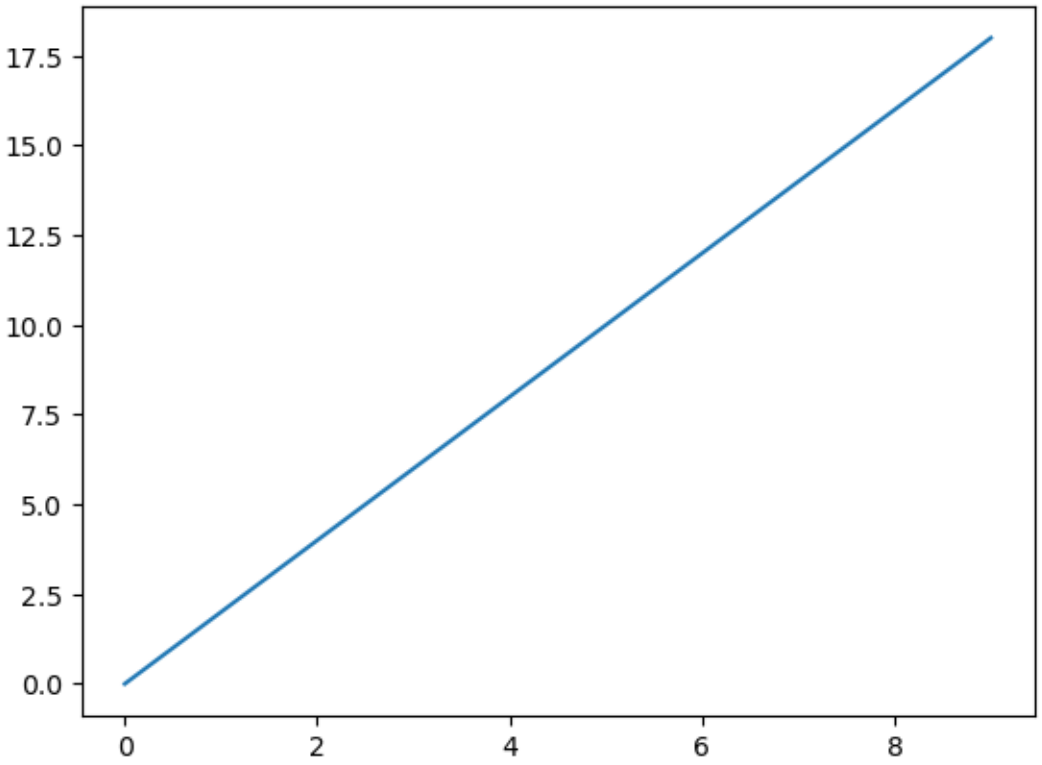


implicit interface

- shadows most of the Axes plotting methods to give the equivalent of the above
- the creation of the Figure and Axes is done for the user

[9]:
```
plt.plot(x,y)
```

[9]:
```
[<matplotlib.lines.Line2D at 0x7f9e881b3d00>]
```



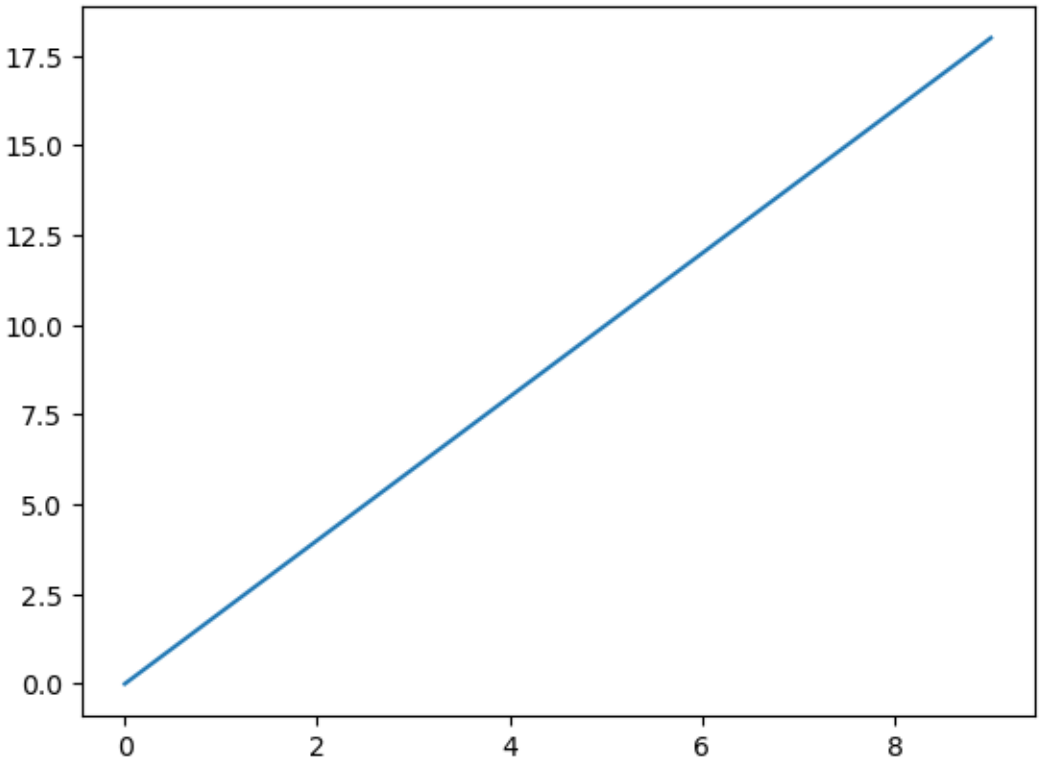
2.4 Anatamy of a Matplotlib Figure

2.4.1 `Figure` and `Axes`

- `Axes`:
  - represent an individual plot
  - differnt from "axis", which refers to the x/y axis of a plot
- `Figure`: the final image that may contain 1 or more `Axes`
- Syntax for creating `Figure` and `Axes`
  - `fig = plt.figure()  # an empty figure with no Axes`
  - `fig, ax = plt.subplots()  # a figure with a single Axes`
  - `fig, axs = plt.subplots(2, 2)  # a figure with a 2x2 grid of Axes`

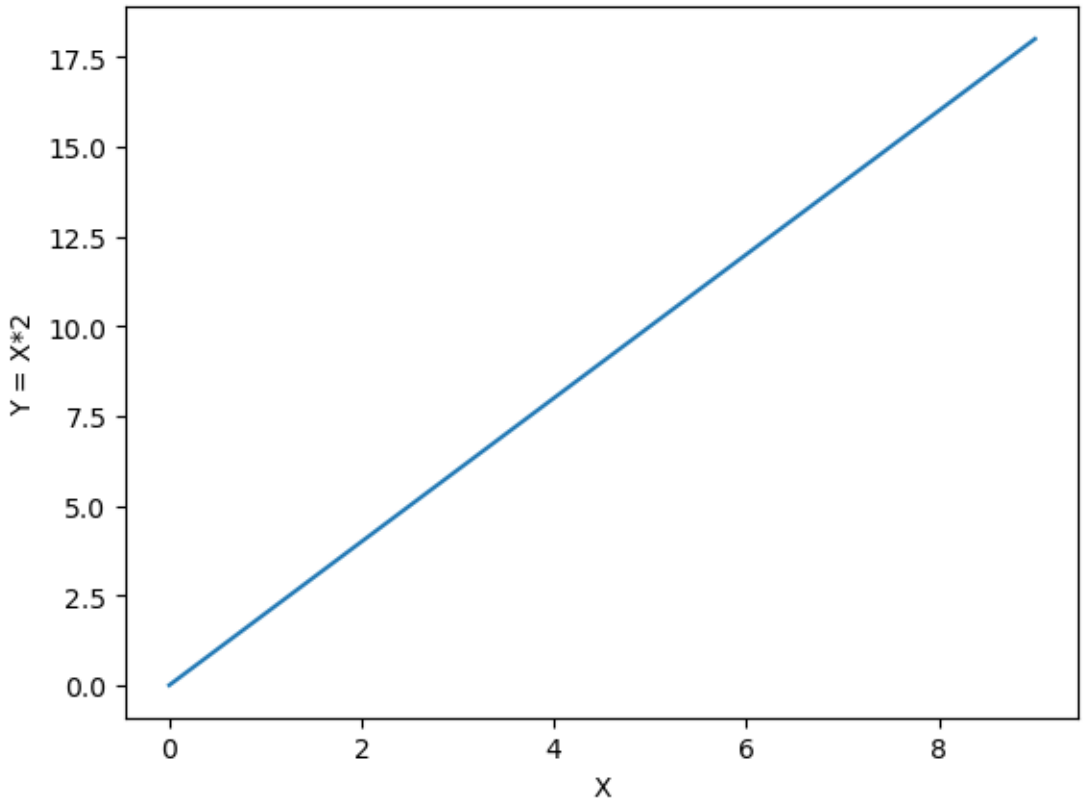
[10]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x, y)  # Plot some data on the axes.
```

[10]:
```
[<matplotlib.lines.Line2D at 0x7f9e98c500a0>]
```



[11]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x, y)  # Plot some data on the axes.
ax.set_xlabel("X")
ax.set_ylabel("Y = X*2")
```

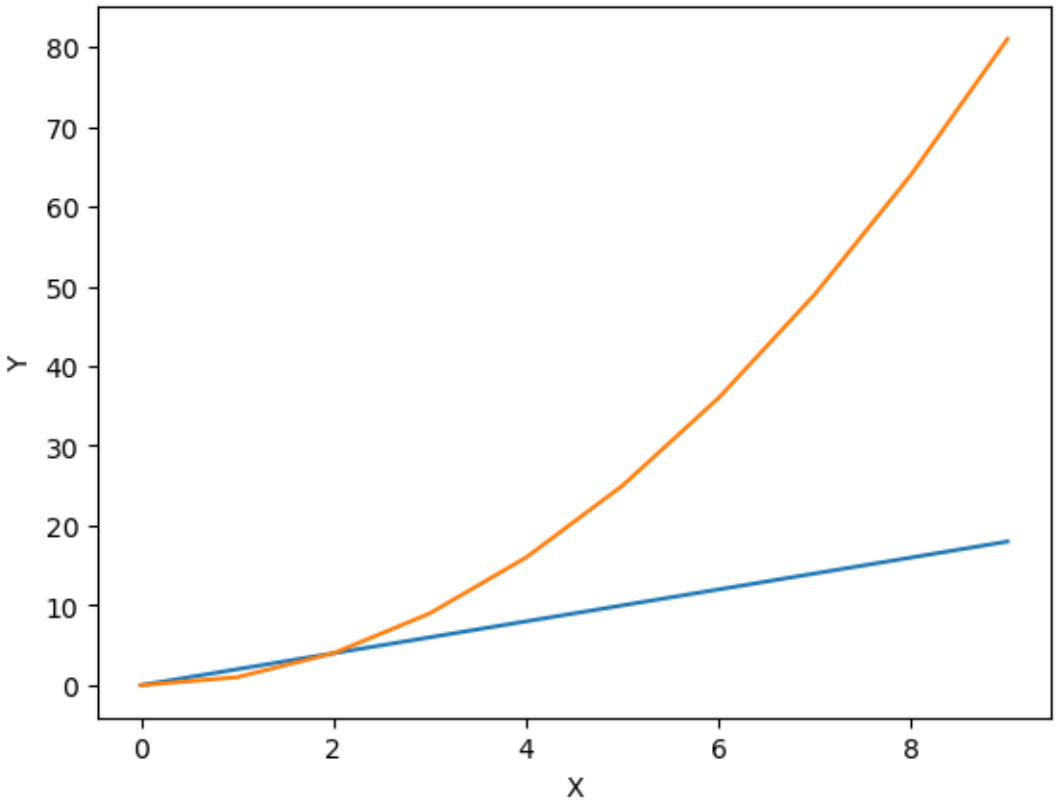
[11]:
```
Text(0, 0.5, 'Y = X*2')
```



What it we want to add another line to the current plot?

[12]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot(x, y)  # Plot some data on the axes.


y2 = x ** 2
ax.plot(x, y2)  # Plot some data on the axes.

ax.set_xlabel("X")
ax.set_ylabel("Y")
```

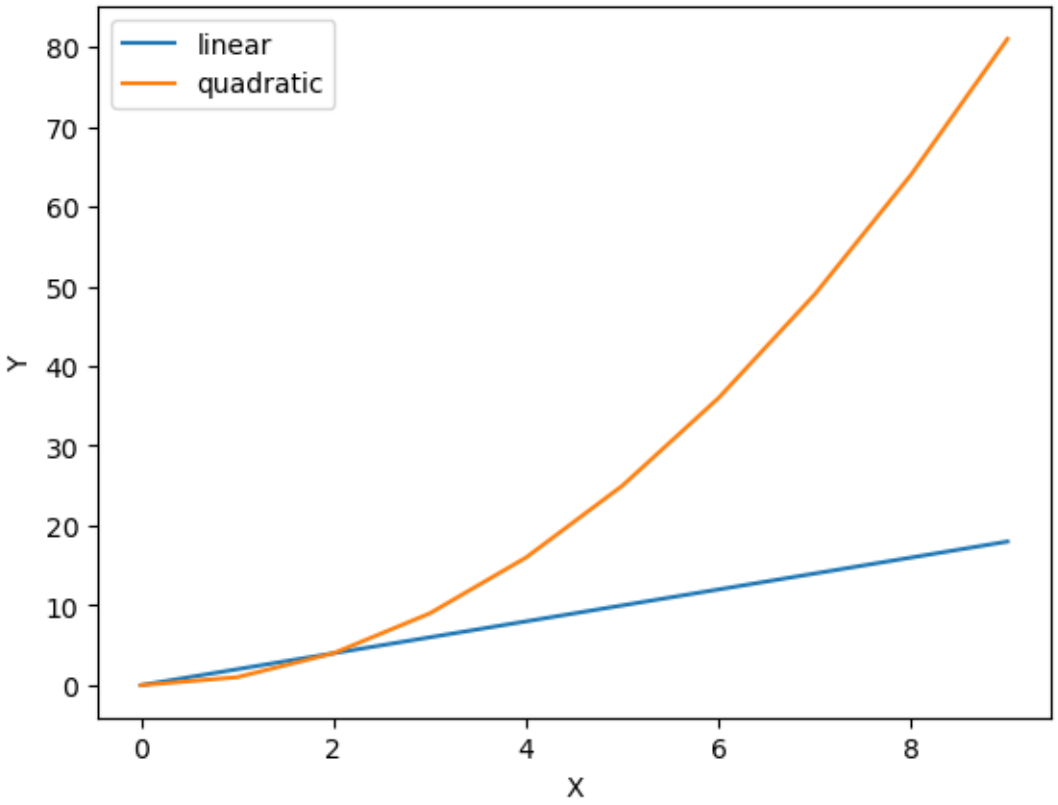
[12]:
```
Text(0, 0.5, 'Y')
```



add legend to different different lines

[13]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
x = np.arange(10)
y = x * 2
ax.plot(x, y, label='linear')

y2 = x ** 2
ax.plot(x, y2, label='quadratic')  # Plot some data on the axes.

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

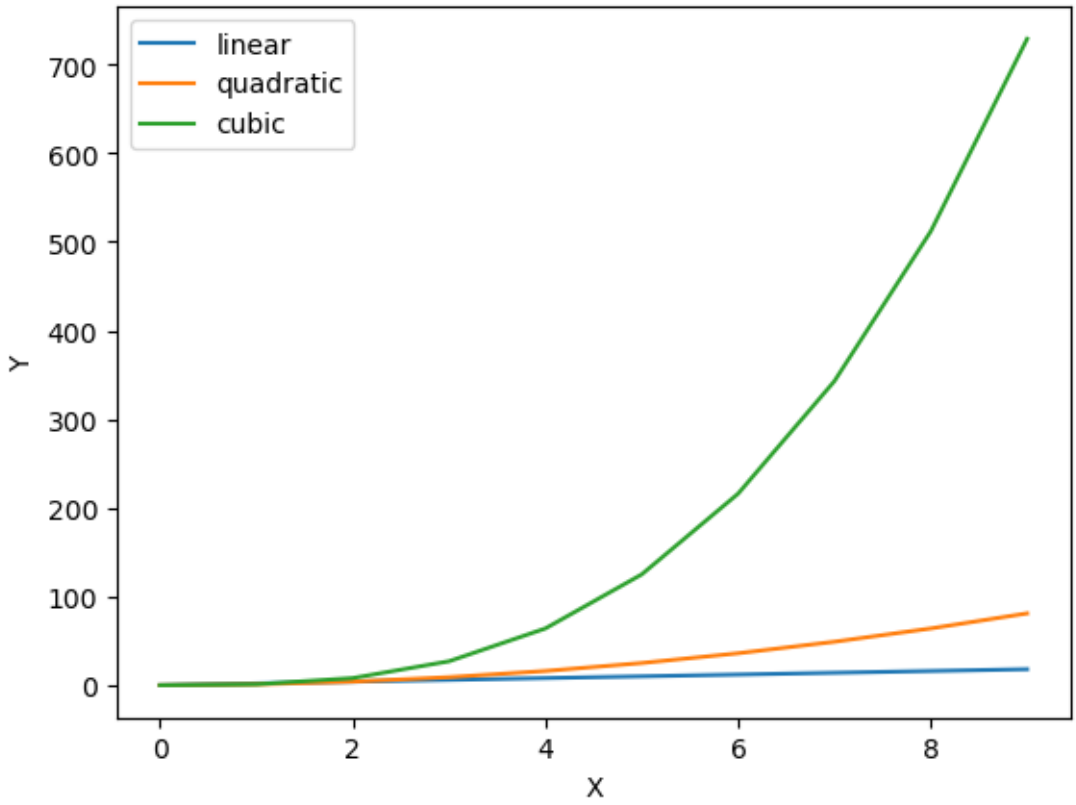
[13]: <matplotlib.legend.Legend at 0x7f9e88221400>



add a third line

[14]:
```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
x = np.arange(10)
y = x * 2
ax.plot(x, y, label='linear')

y2 = x ** 2
ax.plot(x, y2, label='quadratic')  # Plot some data on the axes.

y3 = x ** 3
ax.plot(x, y3, label='cubic')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

[14]:
```
<matplotlib.legend.Legend at 0x7f9e98e21790>
```



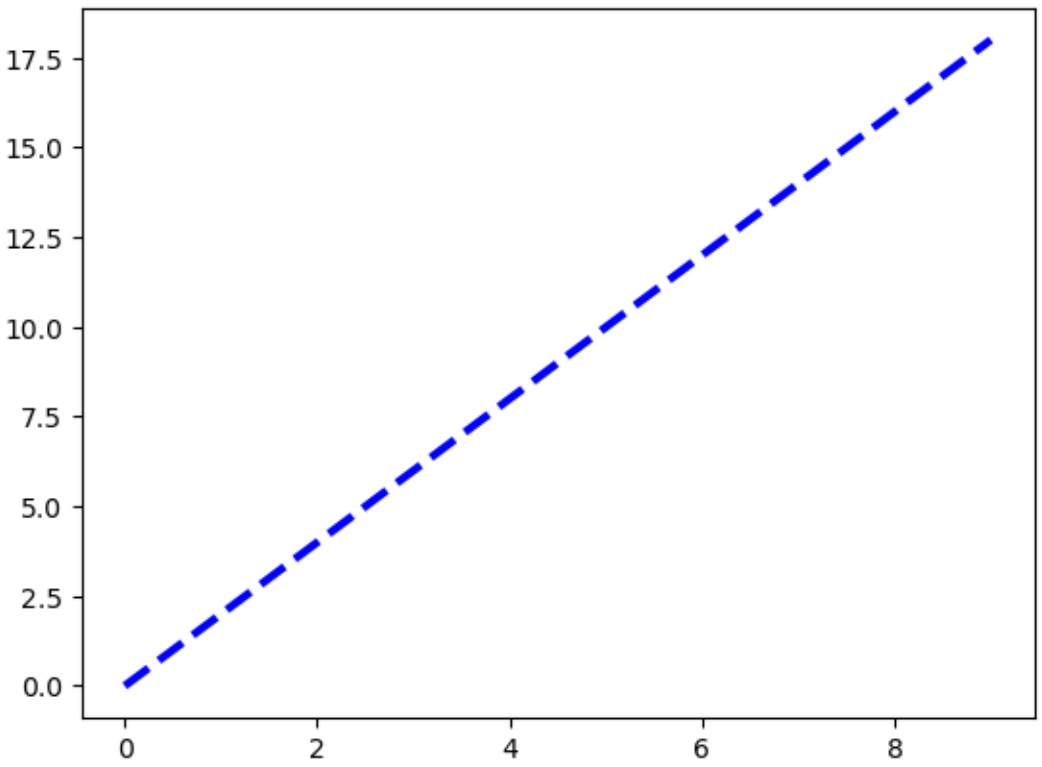
2.4.2 Styling Artists

`Axes`'s plotting methods have styling options for:

- color
- linewidth
- linestyle

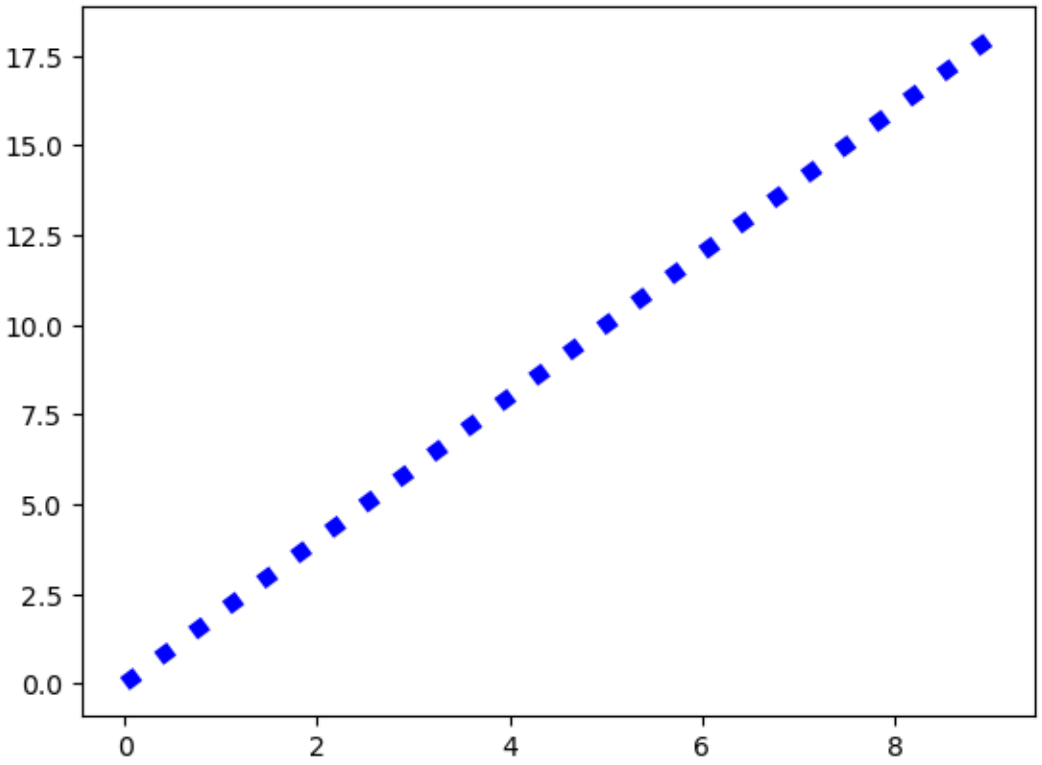
[15]:
```
fig, ax = plt.subplots()
ax.plot(x, y, color='blue', linewidth=3, linestyle='--')
```

[15]:
```
[<matplotlib.lines.Line2D at 0x7f9ec9dc6580>]
```



[16]:
```
fig, ax = plt.subplots()
ax.plot(x, y, color='blue', linewidth=6, linestyle=':')
```

[16]: [<matplotlib.lines.Line2D at 0x7f9e991742e0>]



2.4.3 Group exercise

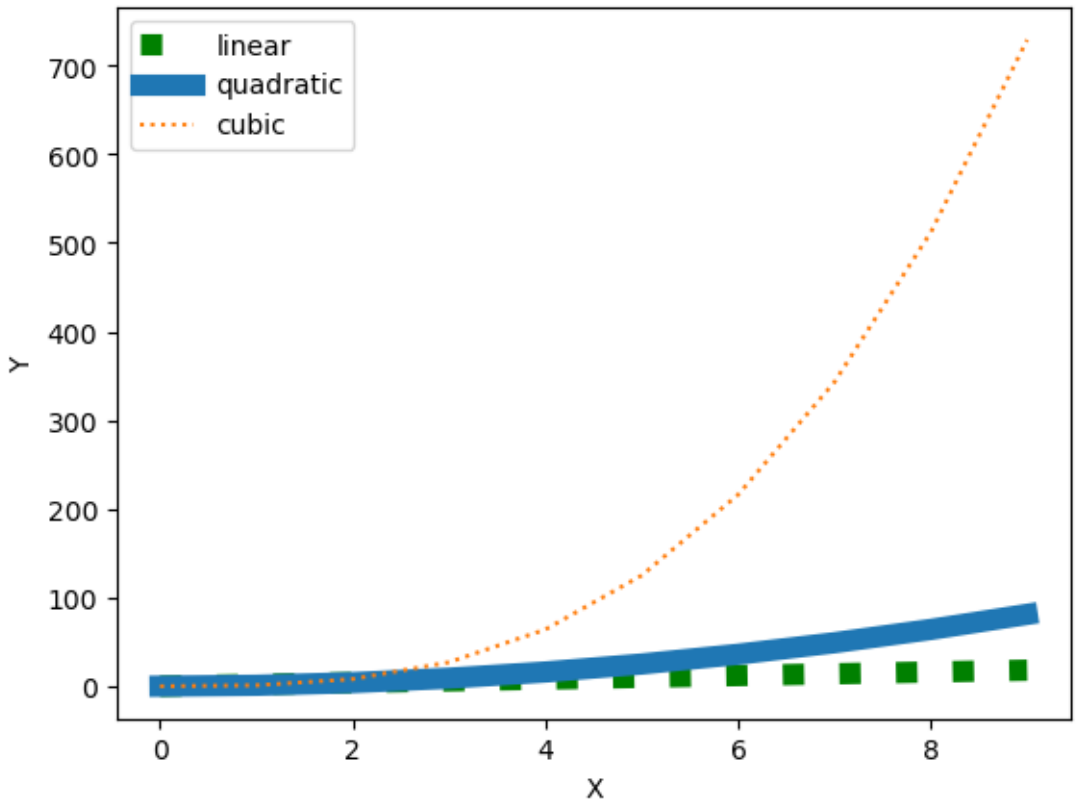
Edit the following python program to change the style (color, linewidth, linestyle) of the three plotted lines

```
fig, ax = plt.subplots()
x = np.arange(10)
y = x * 2
ax.plot(x, y, label='linear')

y2 = x ** 2
ax.plot(x, y2, label='quadratic')

y3 = x ** 3
ax.plot(x, y3, label='cubic')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

when you are done, raise your hand!

[17]:
```
fig, ax = plt.subplots()
x = np.arange(10)
y = x * 2
ax.plot(x, y, label='linear', color="green", linewidth = 8, linestyle =␣
 ↪"dotted")

y2 = x ** 2
ax.plot(x, y2, label='quadratic', linewidth = 8)

y3 = x ** 3
ax.plot(x, y3, label='cubic', linestyle = "dotted")

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

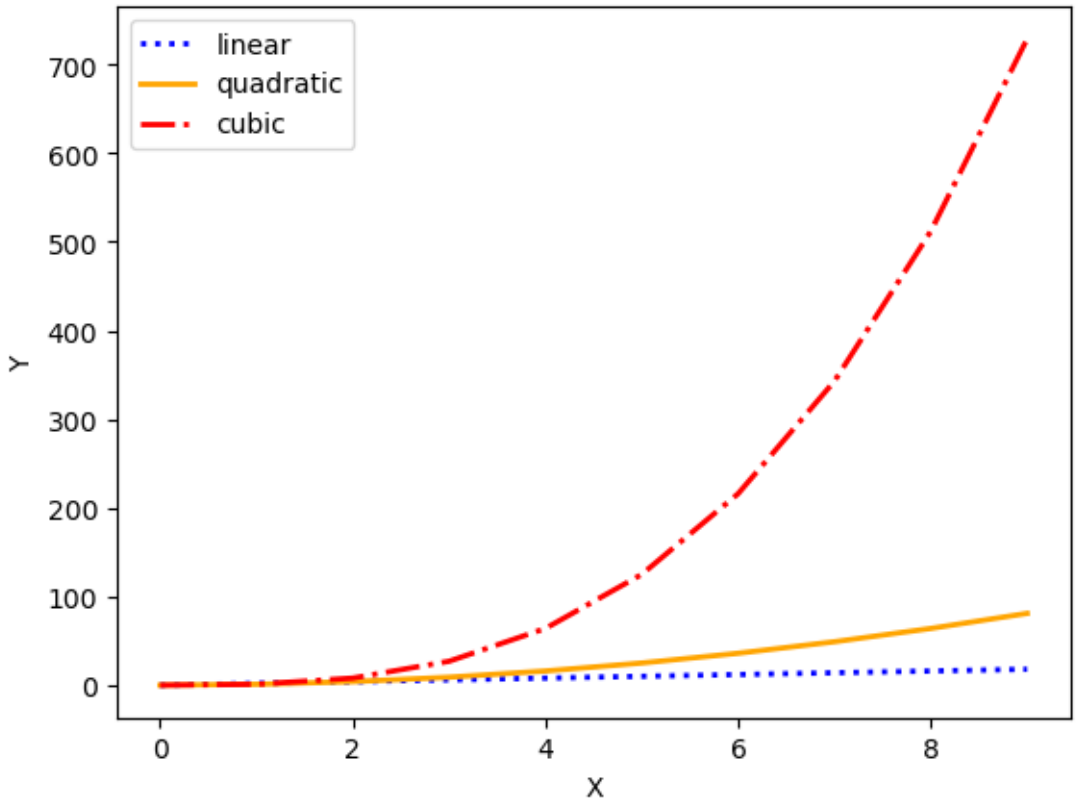
[17]:
```
<matplotlib.legend.Legend at 0x7f9ec9f10be0>
```



[18]:
```
fig, ax = plt.subplots()
x = np.arange(10)
y = x * 2
```

```
ax.plot(x, y, label='linear', color='blue', linewidth=2, linestyle=':')

y2 = x ** 2
ax.plot(x, y2, label='quadratic',color='orange', linewidth=2, linestyle='-')

y3 = x ** 3
ax.plot(x, y3, label='cubic',color='red', linewidth=2, linestyle='dashdot')

ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

[18]: <matplotlib.legend.Legend at 0x7f9ec9f5c7c0>



2.5 multiple plots (Axes) in one figure

When to use multiple plots instead of one

- the two plots are on different scales
  - linear: range [0,9]
  - cubic: range [0, 800]

How?

- Create a figure containing two axes.

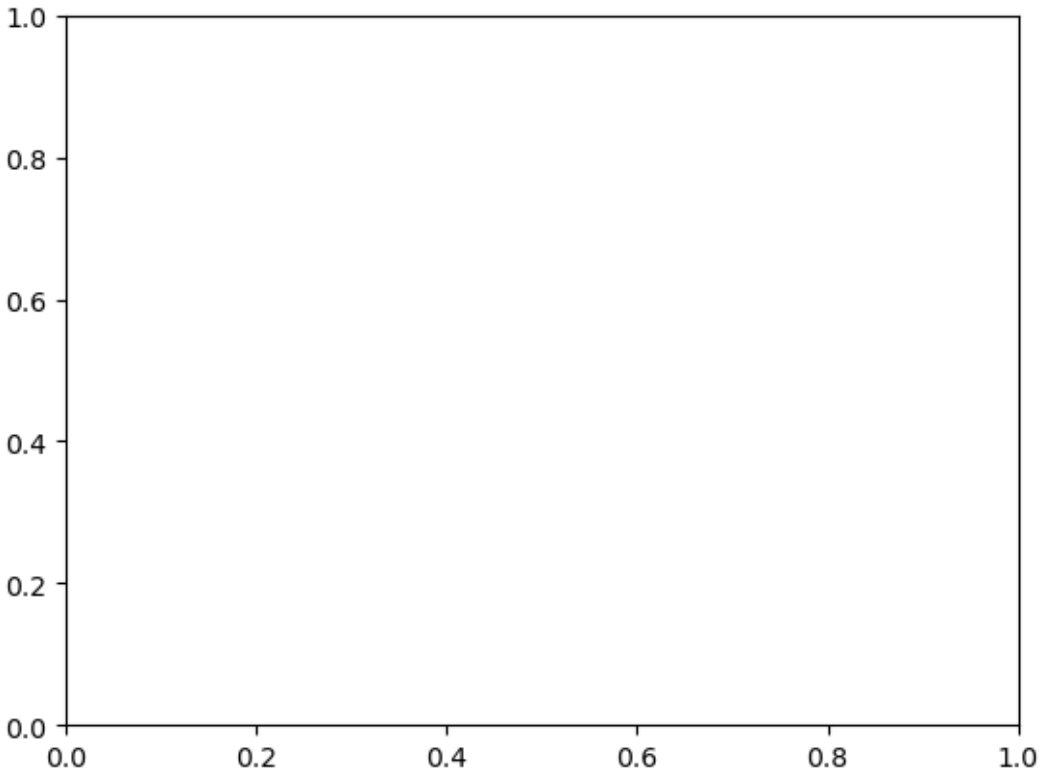
`fig, axes = plt.subplots(1,2)`

- Create a figure containing four axes.

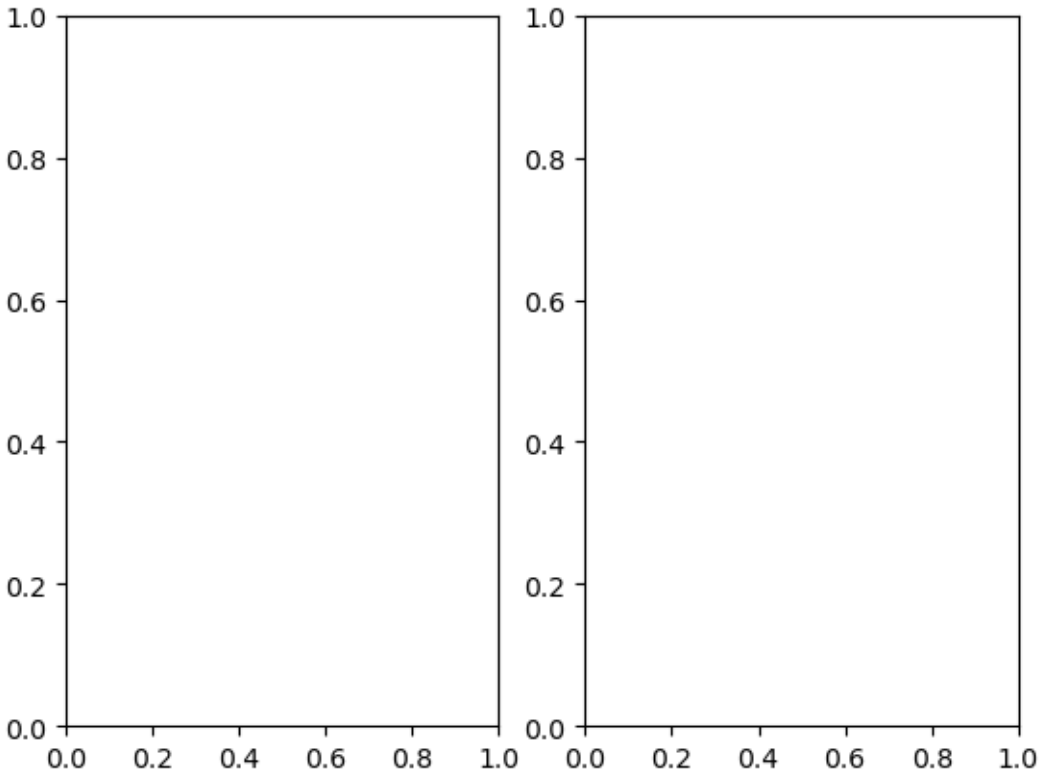
`fig, axes = plt.subplots(2,2)` (2 rows and 2 columns)

`fig, axes = plt.subplots(1,4)` (1 row and 4 columns)

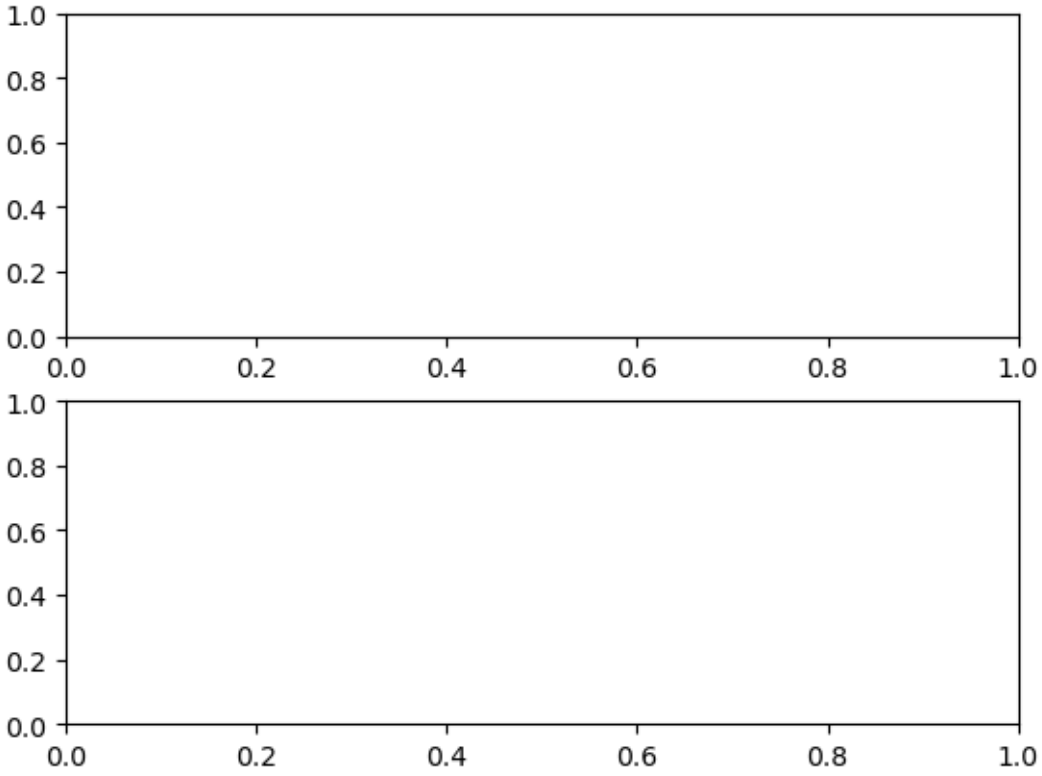
[19]:
```
fig, axes = plt.subplots()
```



[20]:
```
fig, axes = plt.subplots(1,2)  # Create a figure containing two axes.
```

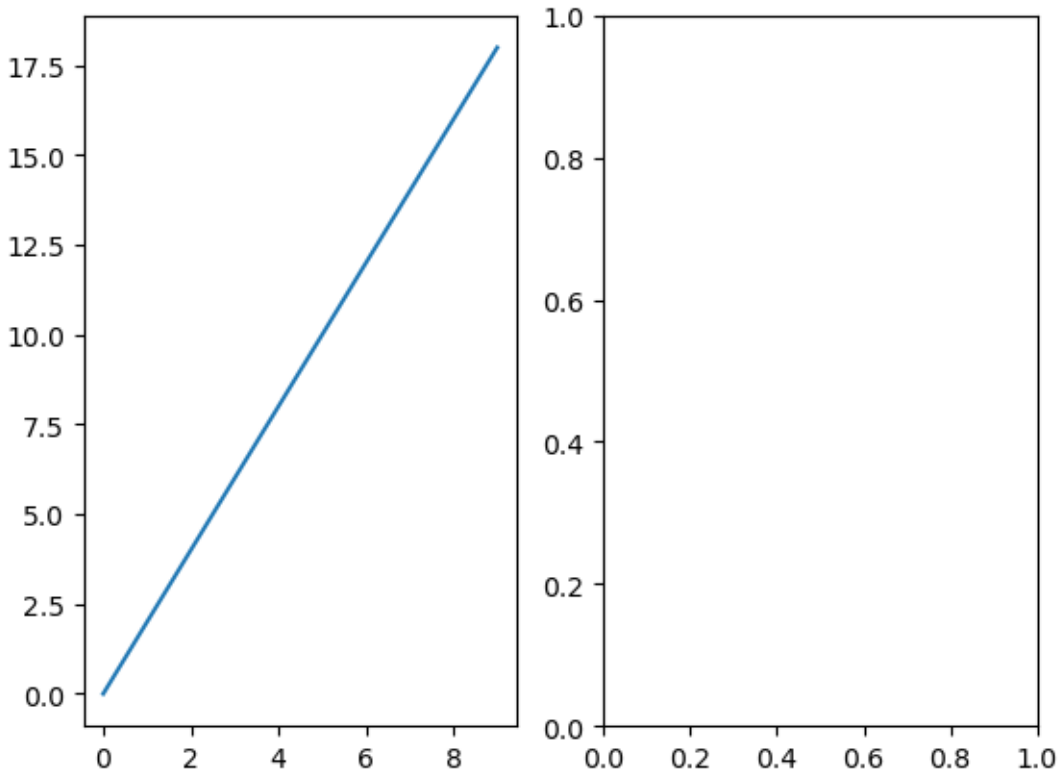


[21]:
```
fig, axes = plt.subplots(2,1)
```



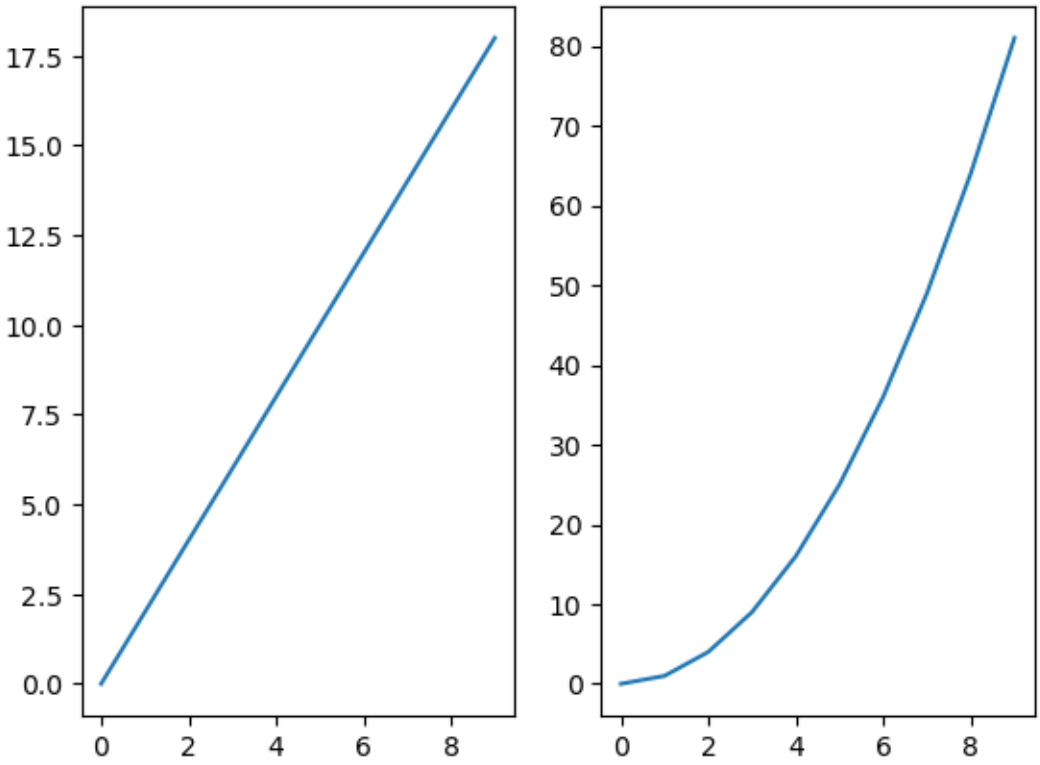
[22]:
```
fig, axes = plt.subplots(1,2)  # Create a figure containing two axes.
ax1 = axes[0]
ax1.plot(x,y)
```

[22]: [<matplotlib.lines.Line2D at 0x7f9ec9ffb4c0>]



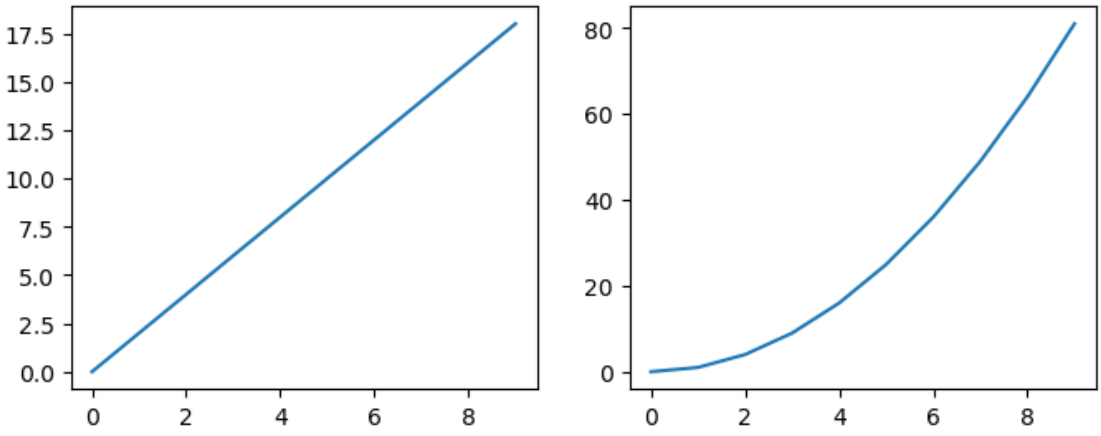
[23]:
```
fig, axes = plt.subplots(1,2)  # Create a figure containing two axes.
ax1 = axes[0]
ax1.plot(x,y)

ax2 = axes[1]
ax2.plot(x,x**2)
```

[23]: [<matplotlib.lines.Line2D at 0x7f9e993b9040>]



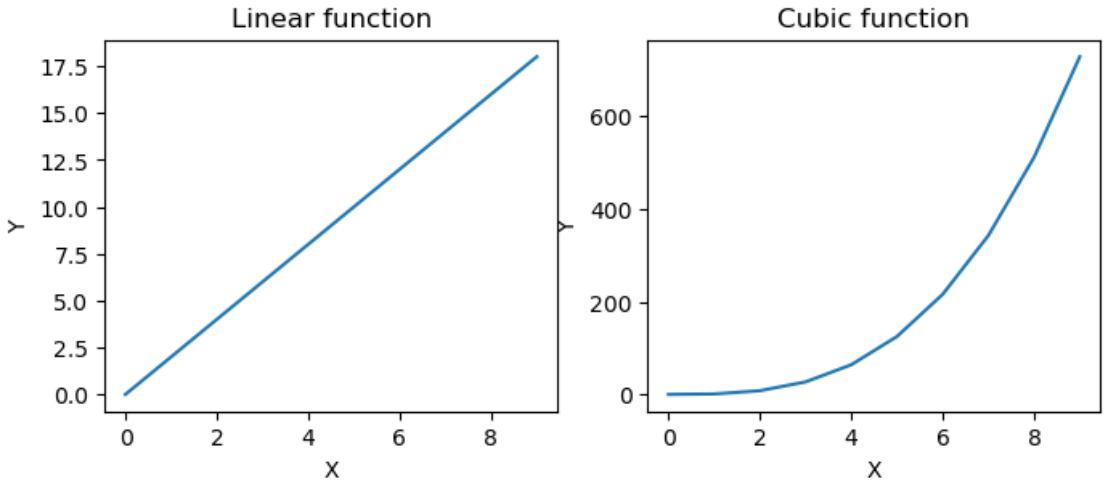
[24]:
```
fig, axes = plt.subplots(1,2,figsize=(8,3))  # Create a figure containing two␣
↪axes.
ax1 = axes[0]
ax1.plot(x,y)

ax2 = axes[1]
ax2.plot(x,x**2)
```

[24]:
```
[<matplotlib.lines.Line2D at 0x7f9ea9794f70>]
```



[25]:
```
fig, axes = plt.subplots(1,2,figsize=(8,3))  # Create a figure containing two␣
↪axes.
ax1 = axes[0]
ax1.plot(x,y)
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_title("Linear function")

ax2 = axes[1]
ax2.plot(x,x**3)
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.set_title("Cubic function")
```

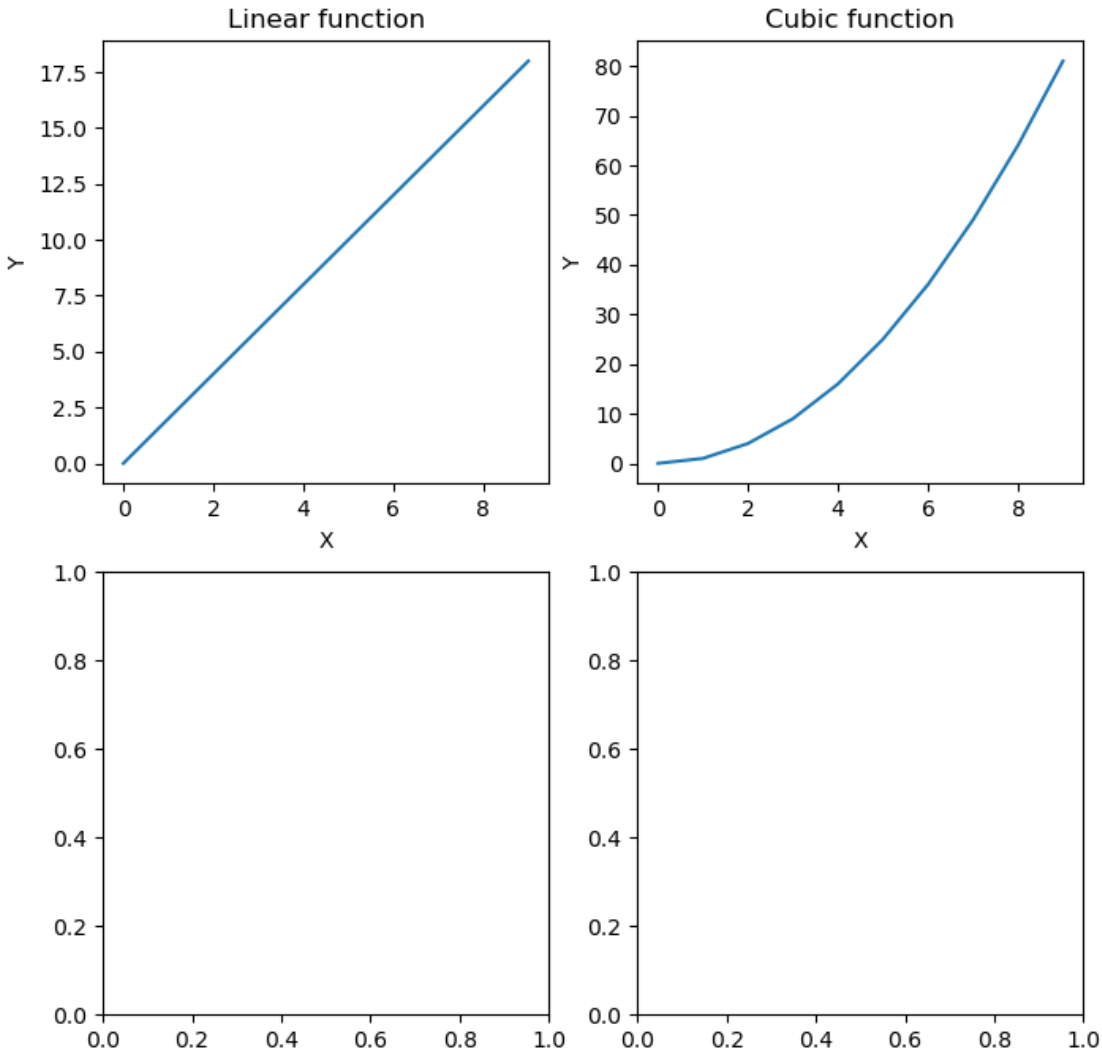
[25]:
```
Text(0.5, 1.0, 'Cubic function')
```



[26]:
```
fig, axes = plt.subplots(2,2,figsize=(8,8))  # Create a figure containing two␣
↪axes.
ax1 = axes[0,0]
ax1.plot(x,y)
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_title("Linear function")

ax2 = axes[0,1]
ax2.plot(x,x**2)
ax2.set_xlabel("X")
```

```
ax2.set_ylabel("Y")
ax2.set_title("Cubic function")
```

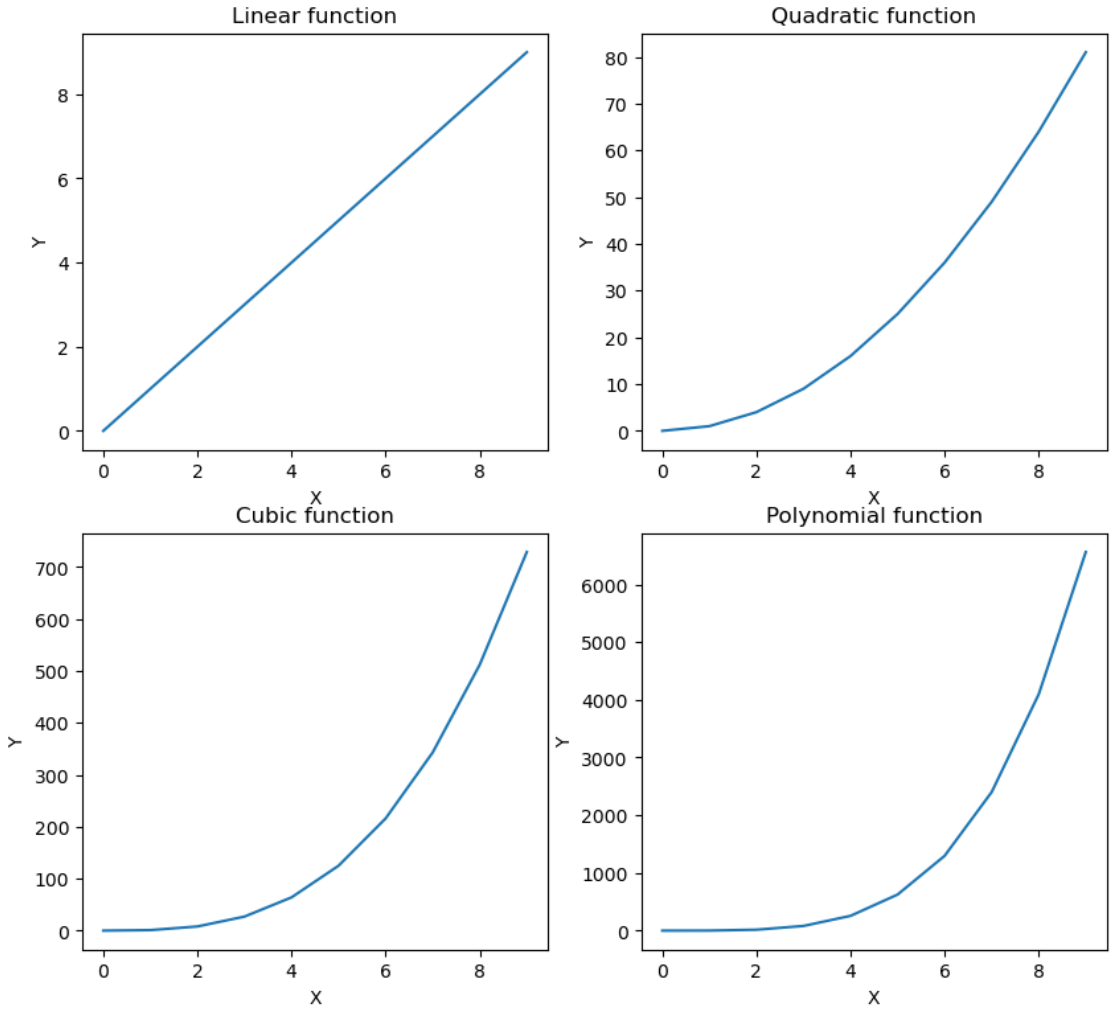
[26]: `Text(0.5, 1.0, 'Cubic function')`



[27]:
```
x = np.arange(10)

fig, axes =plt.subplots(2,2 ,figsize=(10,9))  # Create a figure containing four␣
↪axes.
ax1 = axes[0,0]
ax1.plot(x,x)
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_title("Linear function")
```

```
ax2 = axes[0,1]
ax2.plot(x,x**2)
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.set_title("Quadratic function")

ax3 = axes[1,0]
ax3.plot(x,x**3)
ax3.set_xlabel("X")
ax3.set_ylabel("Y")
ax3.set_title("Cubic function")

ax4 = axes[1,1]
ax4.plot(x,x**4)
ax4.set_xlabel("X")
ax4.set_ylabel("Y")
ax4.set_title("Polynomial function")
```

[27]: `Text(0.5, 1.0, 'Polynomial function')`



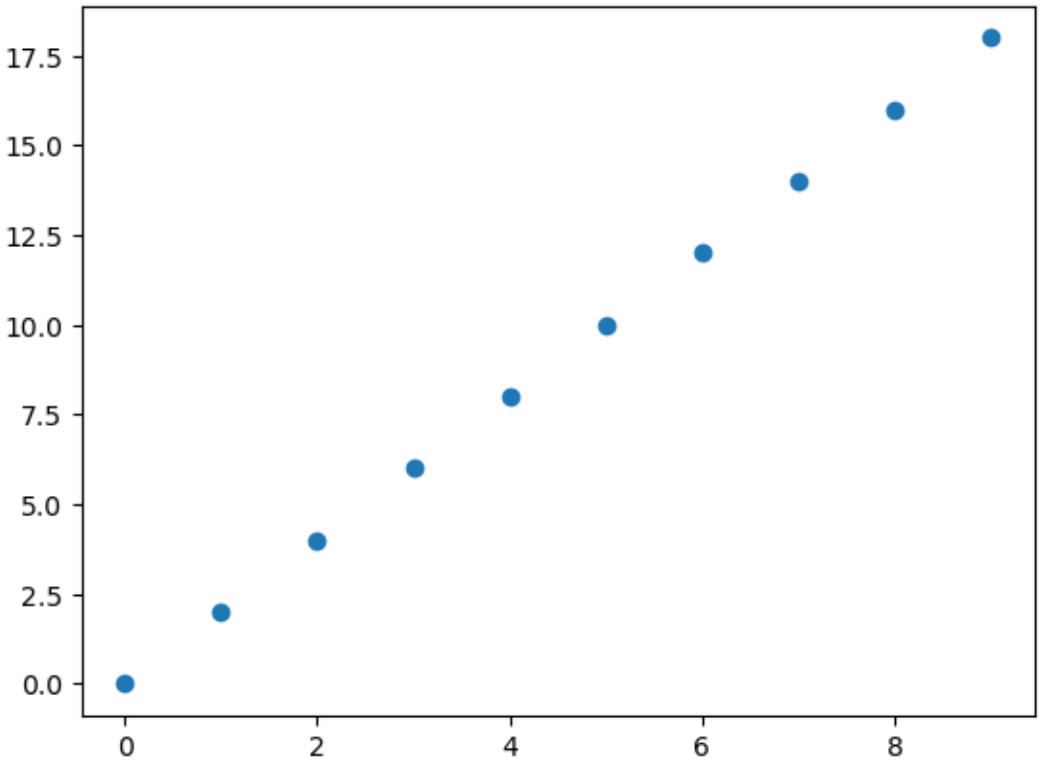
2.6 Scatter plot

`axes.scatter()`

- marker size `s=None`
- marker colors `c=None`
- marker style `marker=None`

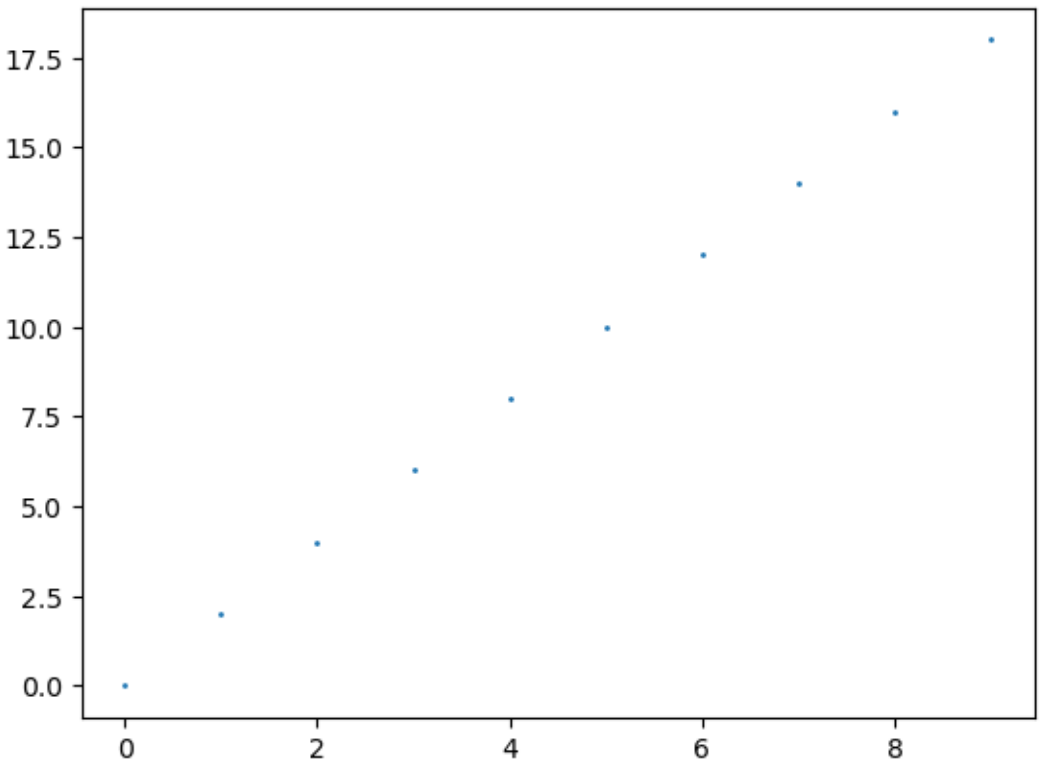
[28]:
```
fig, ax = plt.subplots()
ax.scatter(x,y)
```

[28]:
```
<matplotlib.collections.PathCollection at 0x7f9ea98d5fa0>
```



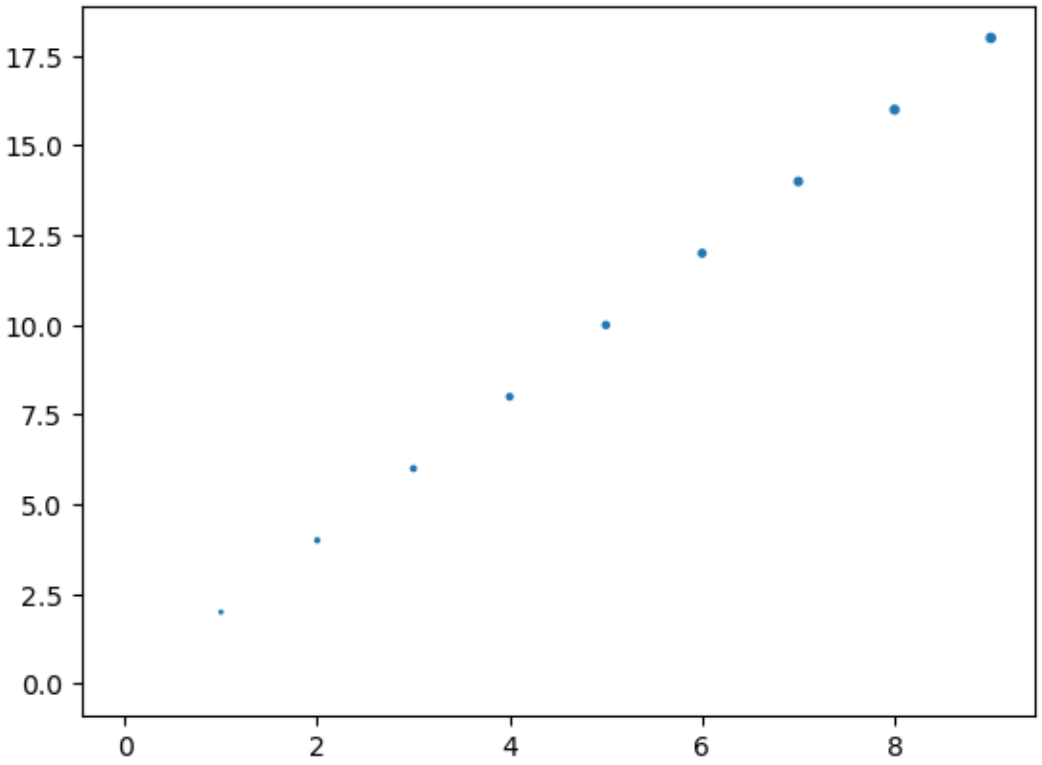
[29]:
```
fig, ax = plt.subplots()
ax.scatter(x,y, s=1)
```

[29]: <matplotlib.collections.PathCollection at 0x7f9eca4395b0>



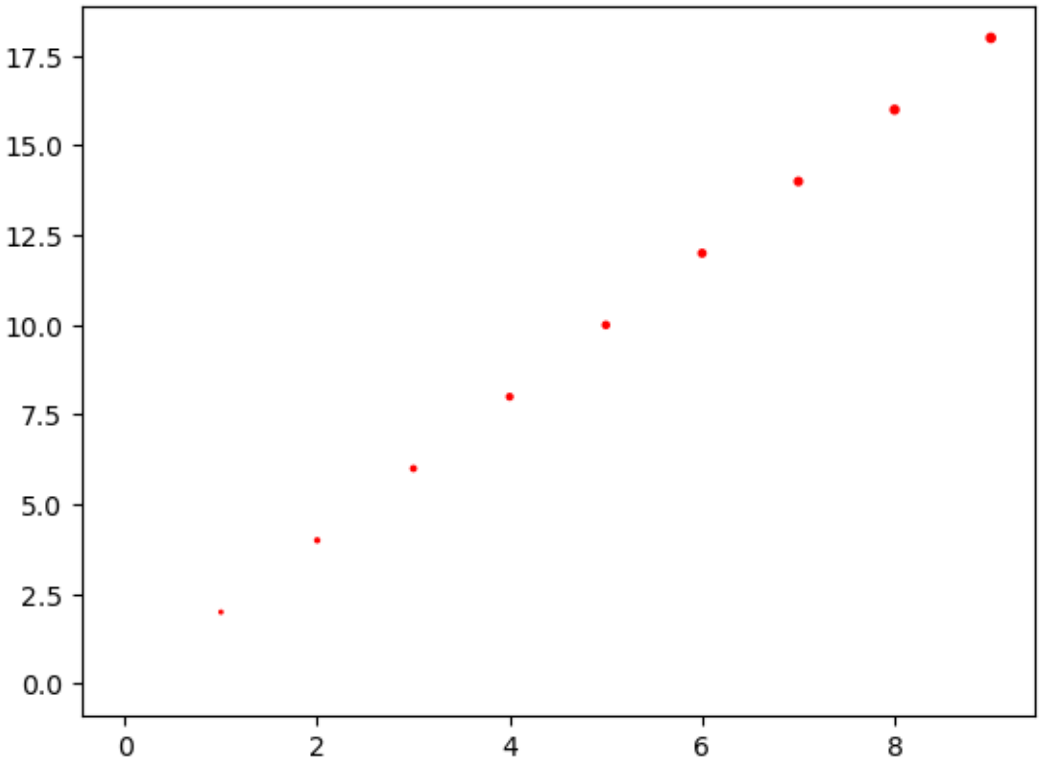
[30]:
```
fig, ax = plt.subplots()
ax.scatter(x,y, s=np.arange(10))
```

[30]:
```
<matplotlib.collections.PathCollection at 0x7f9eb84faa60>
```



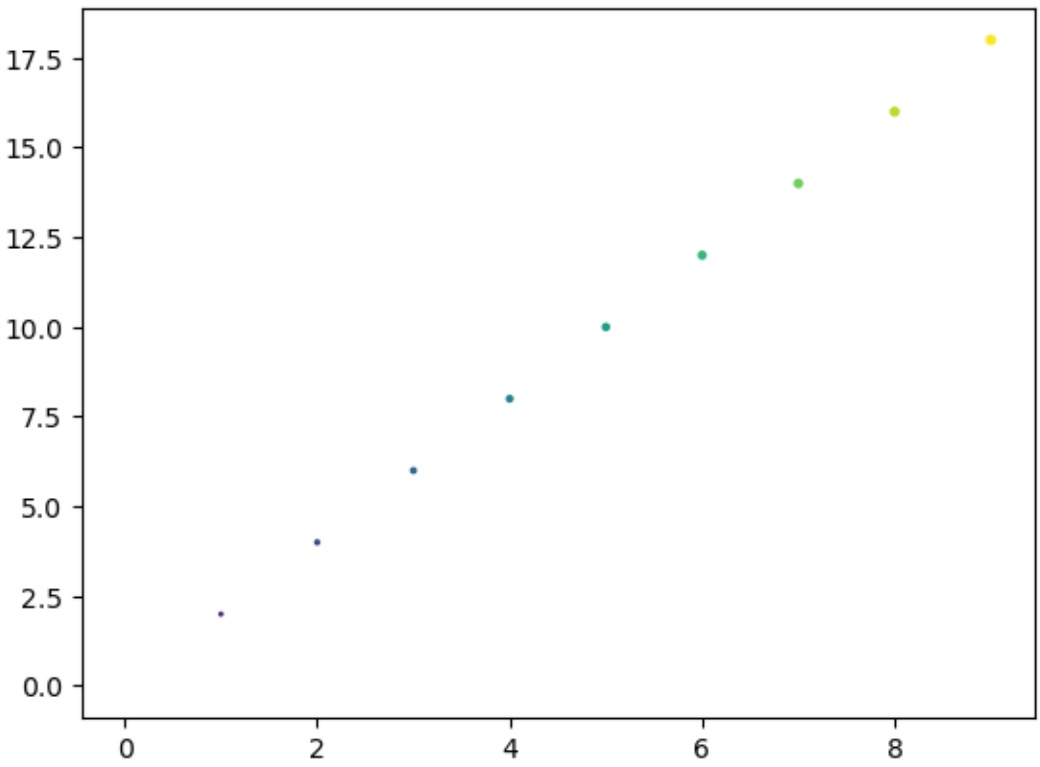
[31]:
```
fig, ax = plt.subplots()
ax.scatter(x,y, s=np.arange(10),c="red")
```

[31]: <matplotlib.collections.PathCollection at 0x7f9e7830fb20>



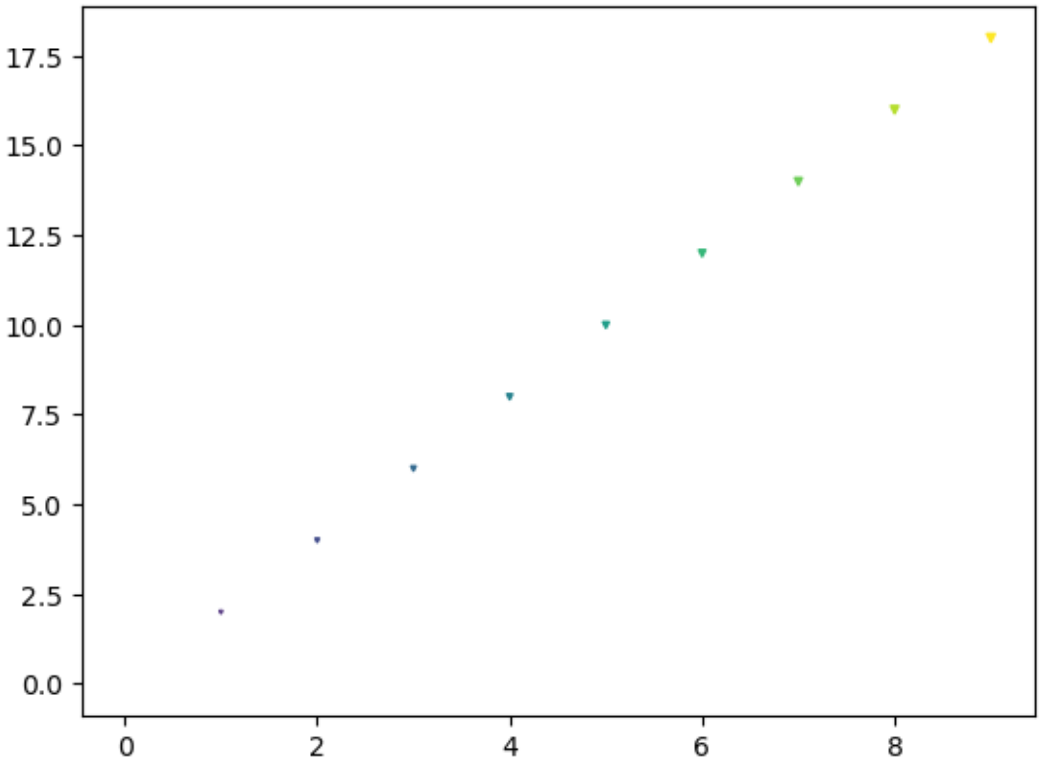
[32]:
```
fig, ax = plt.subplots()
ax.scatter(x,y, s=np.arange(10),c=np.arange(10))
```

[32]: <matplotlib.collections.PathCollection at 0x7f9e98eb5790>



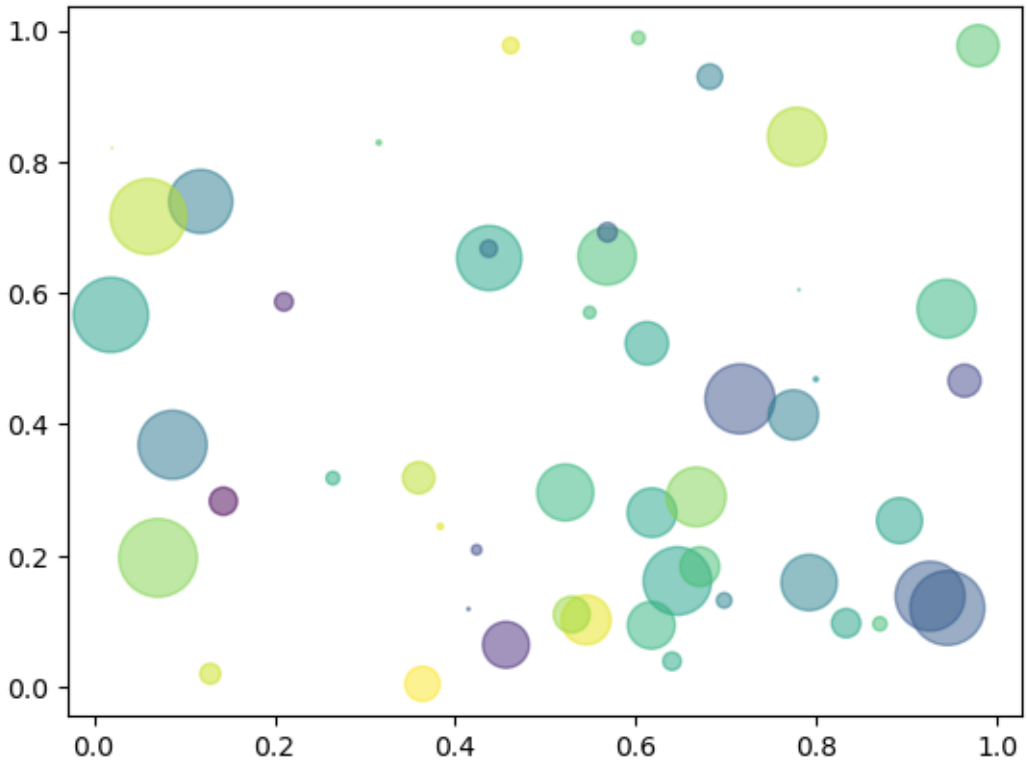
[33]:
```
fig, ax = plt.subplots()
ax.scatter(x,y, s=np.arange(10),c=np.arange(10), marker="v")
```

[33]: <matplotlib.collections.PathCollection at 0x7f9ea9ab0fd0>



[34]:
```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(0)


N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = (30 * np.random.rand(N))**2  # 0 to 15 point radii
fig, ax = plt.subplots()
ax.scatter(x, y, s=area, c=colors, alpha=0.5)
```

[34]:
```
<matplotlib.collections.PathCollection at 0x7f9e884fd280>
```



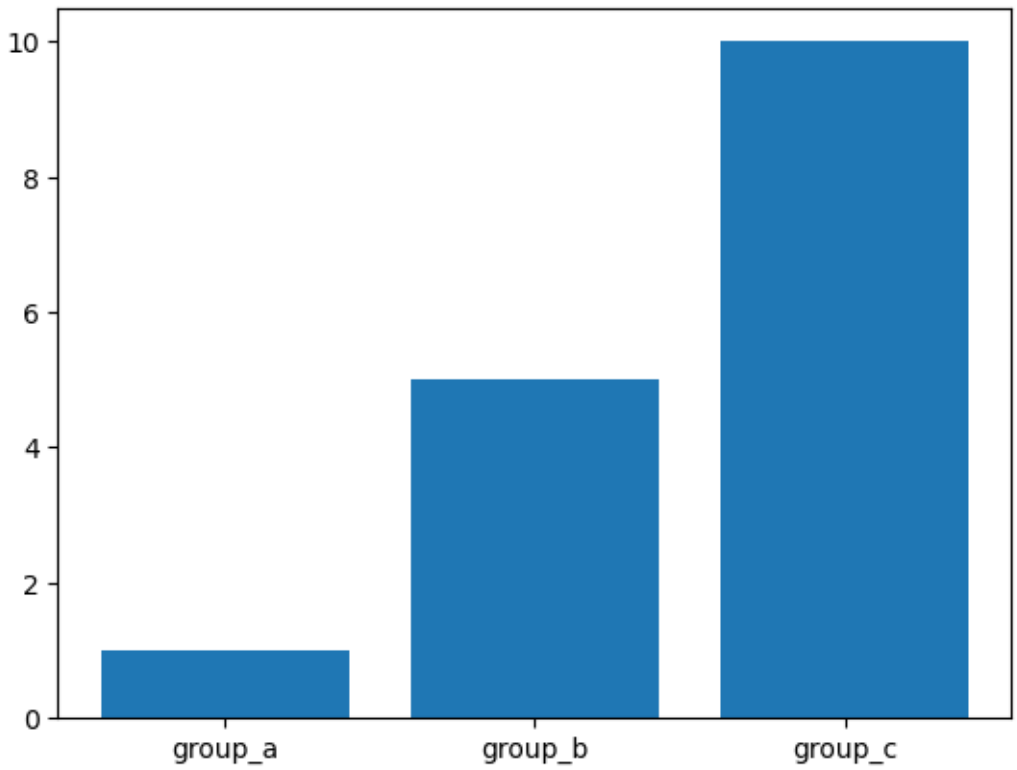
2.7 bar plot for categorical variables

`axes.bar()`: a vertical bar plot.

`axes.barh()`: a horizontal bar plot.

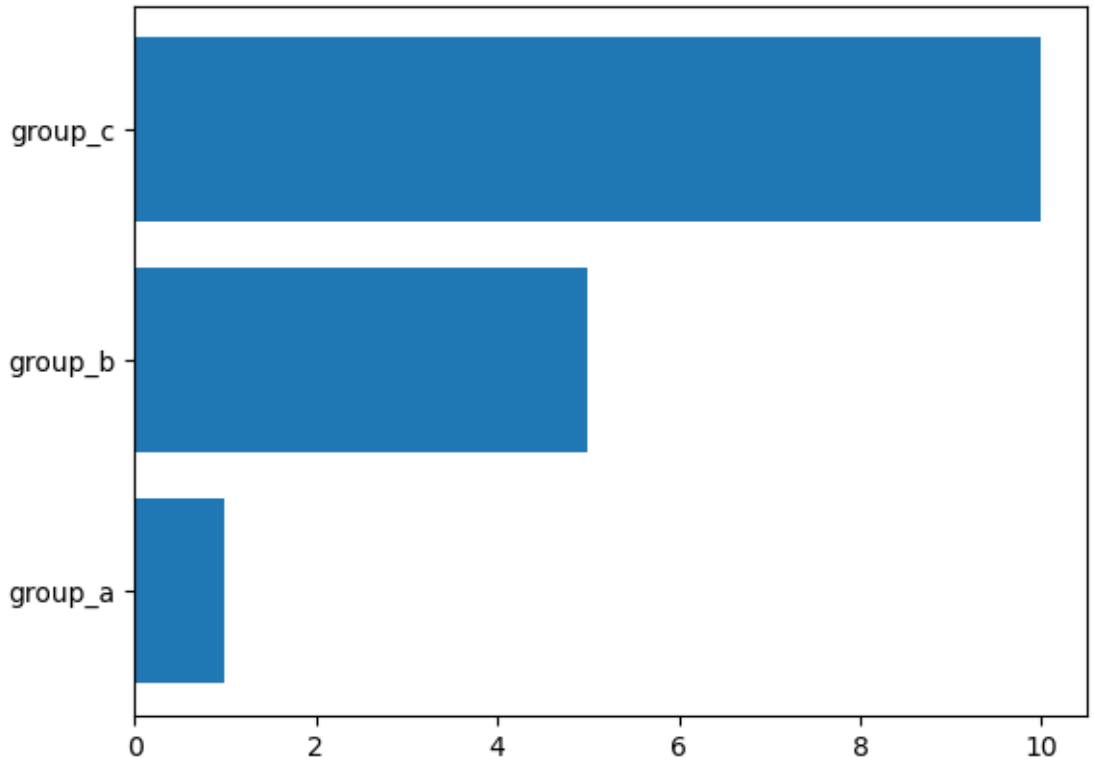
[35]:
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 5, 10]
fig, ax = plt.subplots()
ax.bar(names, values)
```

[35]:
```
<BarContainer object of 3 artists>
```



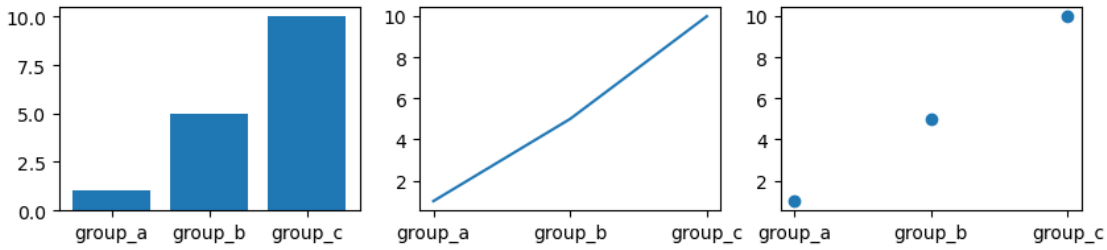
[36]:
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 5, 10]
fig, ax = plt.subplots()
ax.barh(names, values)
```

[36]: <BarContainer object of 3 artists>



[37]:
```
names = ['group_a', 'group_b', 'group_c']
values = [1, 5, 10]
fig, axes = plt.subplots(1,3, figsize=(10,2))
axes[0].bar(names, values)
axes[1].plot(names, values)
axes[2].scatter(names, values)
```

[37]: <matplotlib.collections.PathCollection at 0x7f9e8857ffd0>



2.8 Distributions and Densities

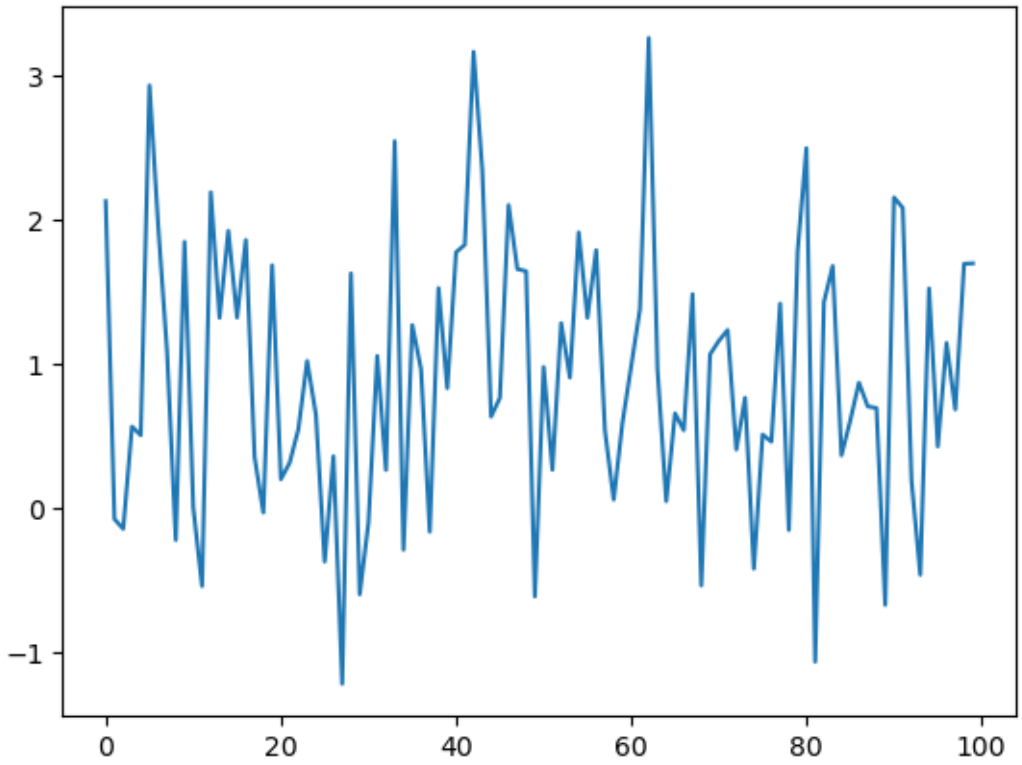
[38]:
```
import numpy as np

x = np.random.normal(1,1,100)
```

[39]:
```
x
```

[39]:
```
array([ 2.12663592e+00, -7.99315084e-02, -1.47468652e-01,  5.62179955e-01,
        5.01967549e-01,  2.92953205e+00,  1.94942081e+00,  1.08755124e+00,
       -2.25435519e-01,  1.84436298e+00, -2.15347390e-04, -5.44771097e-01,
        2.18802979e+00,  1.31694261e+00,  1.92085882e+00,  1.31872765e+00,
        1.85683061e+00,  3.48974407e-01, -3.42428418e-02,  1.68159452e+00,
        1.96590336e-01,  3.10450222e-01,  5.44467496e-01,  1.01747916e+00,
        6.46006089e-01, -3.74951293e-01,  3.56381597e-01, -1.22340315e+00,
        1.62523145e+00, -6.02057656e-01, -1.04383339e-01,  1.05216508e+00,
        2.60437004e-01,  2.54301460e+00, -2.92856910e-01,  1.26705087e+00,
        9.60717182e-01, -1.68093498e-01,  1.52327666e+00,  8.28453669e-01,
        1.77179055e+00,  1.82350415e+00,  3.16323595e+00,  2.33652795e+00,
        6.30818162e-01,  7.60620822e-01,  2.09965960e+00,  1.65526373e+00,
        1.64013153e+00, -6.16956044e-01,  9.75673876e-01,  2.61969091e-01,
        1.27992460e+00,  9.01849610e-01,  1.91017891e+00,  1.31721822e+00,
        1.78632796e+00,  5.33580903e-01,  5.55537441e-02,  5.89950307e-01,
        9.82979586e-01,  1.37915174e+00,  3.25930895e+00,  9.57742848e-01,
        4.40549995e-02,  6.54018224e-01,  5.36404025e-01,  1.48148147e+00,
       -5.40797014e-01,  1.06326199e+00,  1.15650654e+00,  1.23218104e+00,
        4.02683931e-01,  7.62078270e-01, -4.24060909e-01,  5.06680117e-01,
        4.57138524e-01,  1.41605005e+00, -1.56182432e-01,  1.78119810e+00,
        2.49448454e+00, -1.06998503e+00,  1.42625873e+00,  1.67690804e+00,
        3.62562974e-01,  6.02728186e-01,  8.67119422e-01,  7.02209121e-01,
        6.90987031e-01, -6.76003806e-01,  2.15233156e+00,  2.07961859e+00,
        1.86635741e-01, -4.66424328e-01,  1.52106488e+00,  4.24212030e-01,
        1.14195316e+00,  6.80671583e-01,  1.69153875e+00,  1.69474914e+00])
```

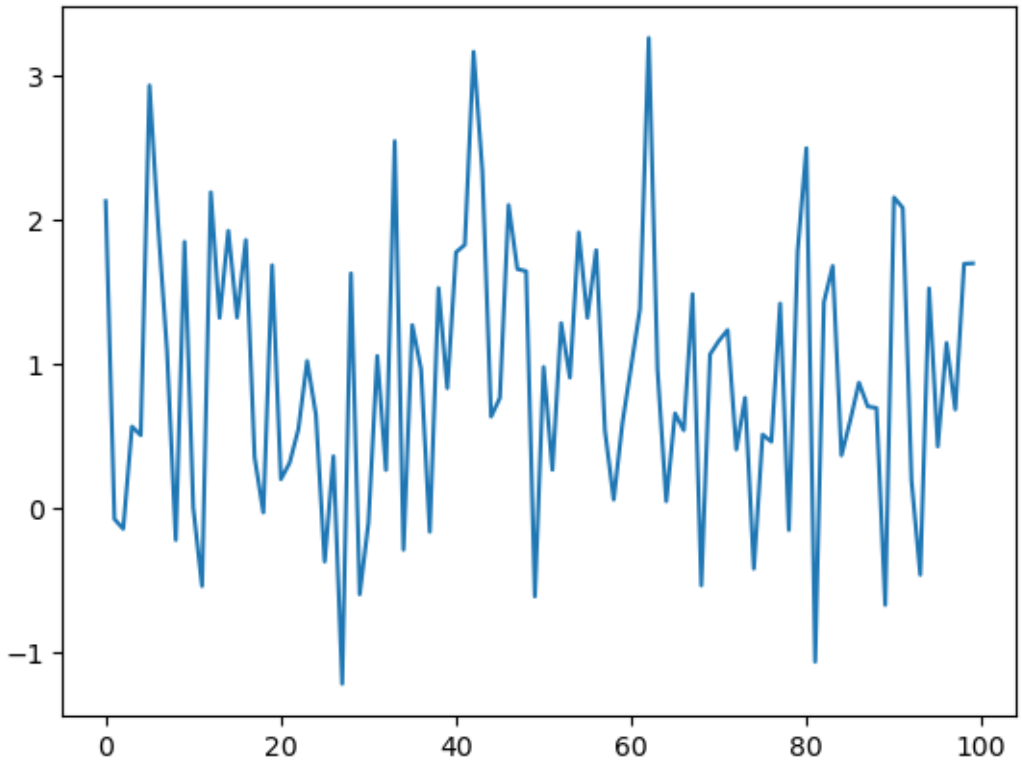
[40]:
```
fig, ax = plt.subplots()
ax.plot(x)
```

[40]:
```
[<matplotlib.lines.Line2D at 0x7f9e99354ca0>]
```



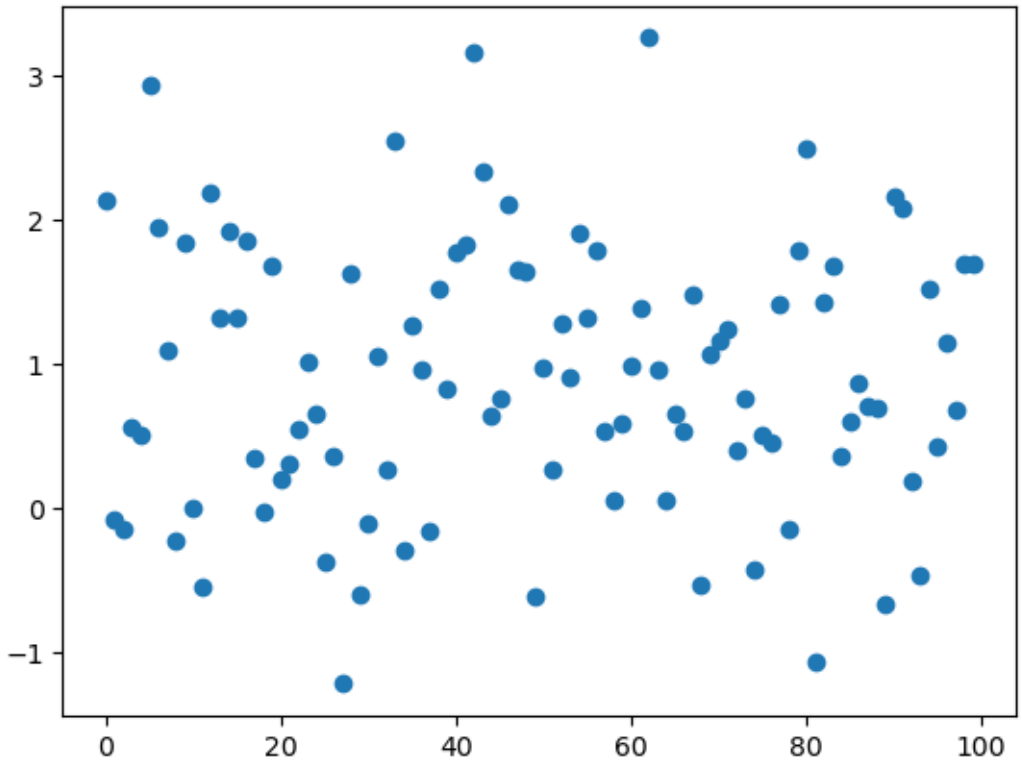
[41]:
```
fig, ax = plt.subplots()
ax.plot(np.arange(100),x)
```

[41]: [<matplotlib.lines.Line2D at 0x7f9eb831d9d0>]



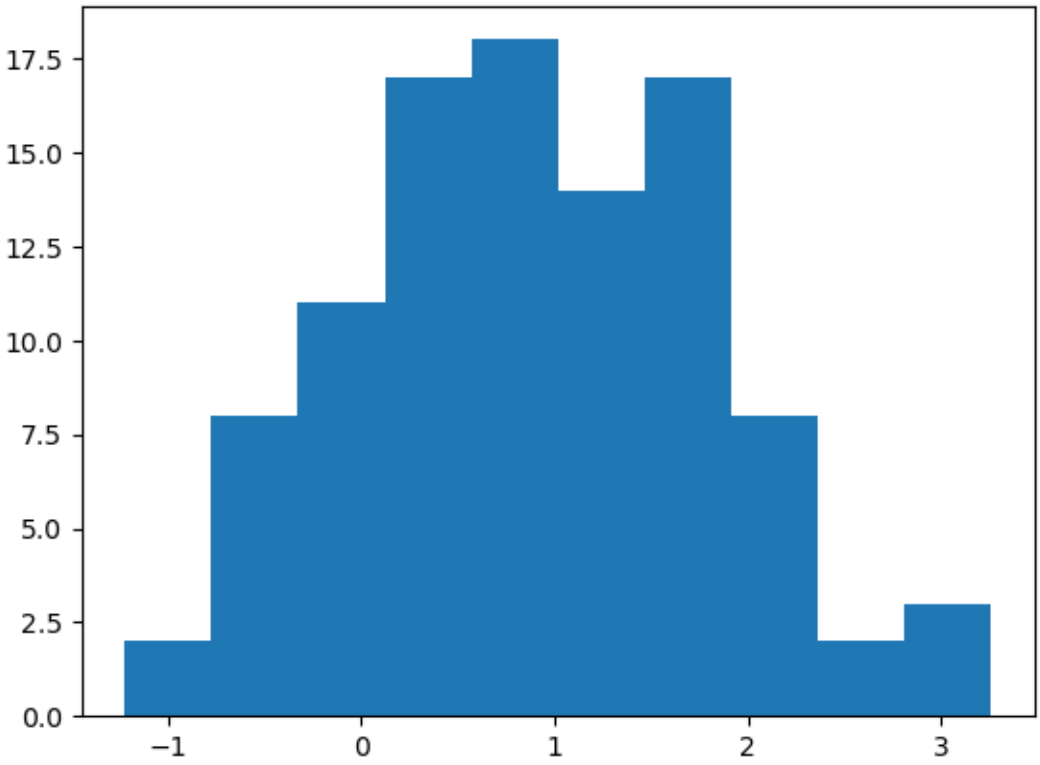
[42]:
```
fig, ax = plt.subplots()
ax.scatter(np.arange(100),x)
```

[42]:
```
<matplotlib.collections.PathCollection at 0x7f9eca38d9a0>
```



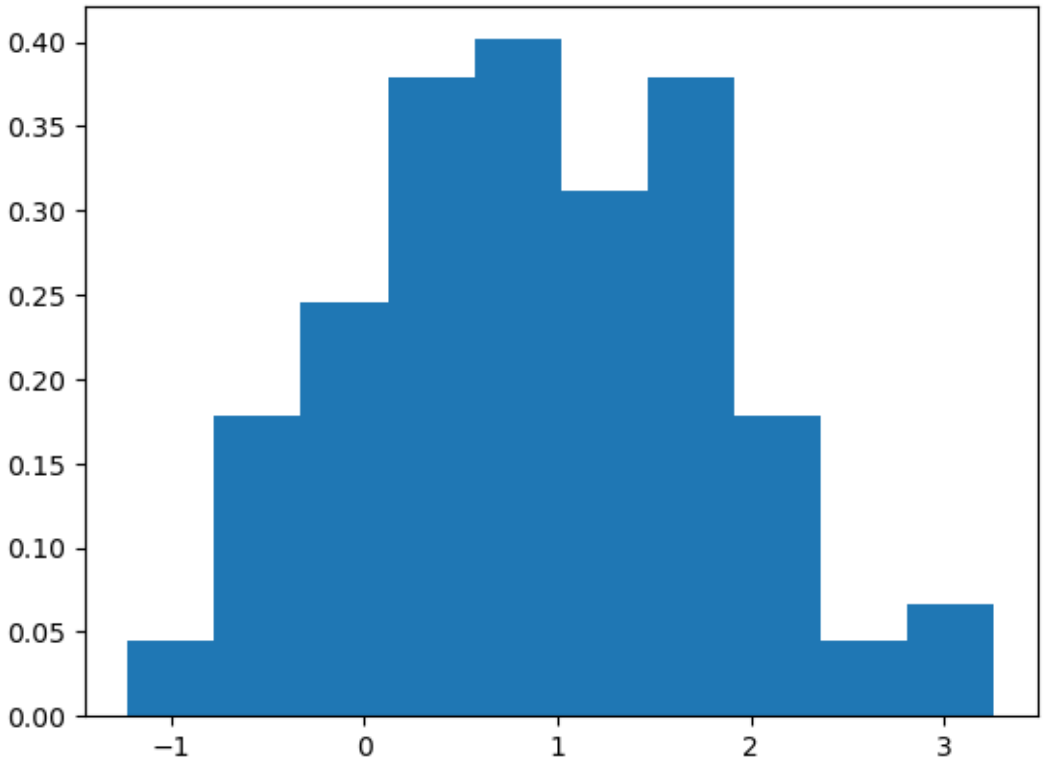
[43]:
```
fig, ax = plt.subplots()
ax.hist(x)
```

[43]:
```
(array([ 2.,  8., 11., 17., 18., 14., 17.,  8.,  2.,  3.]),
 array([-1.22340315, -0.77513194, -0.32686073,  0.12141048,  0.56968169,
         1.0179529 ,  1.46622411,  1.91449532,  2.36276653,  2.81103774,
         3.25930895]),
 <BarContainer object of 10 artists>)
```



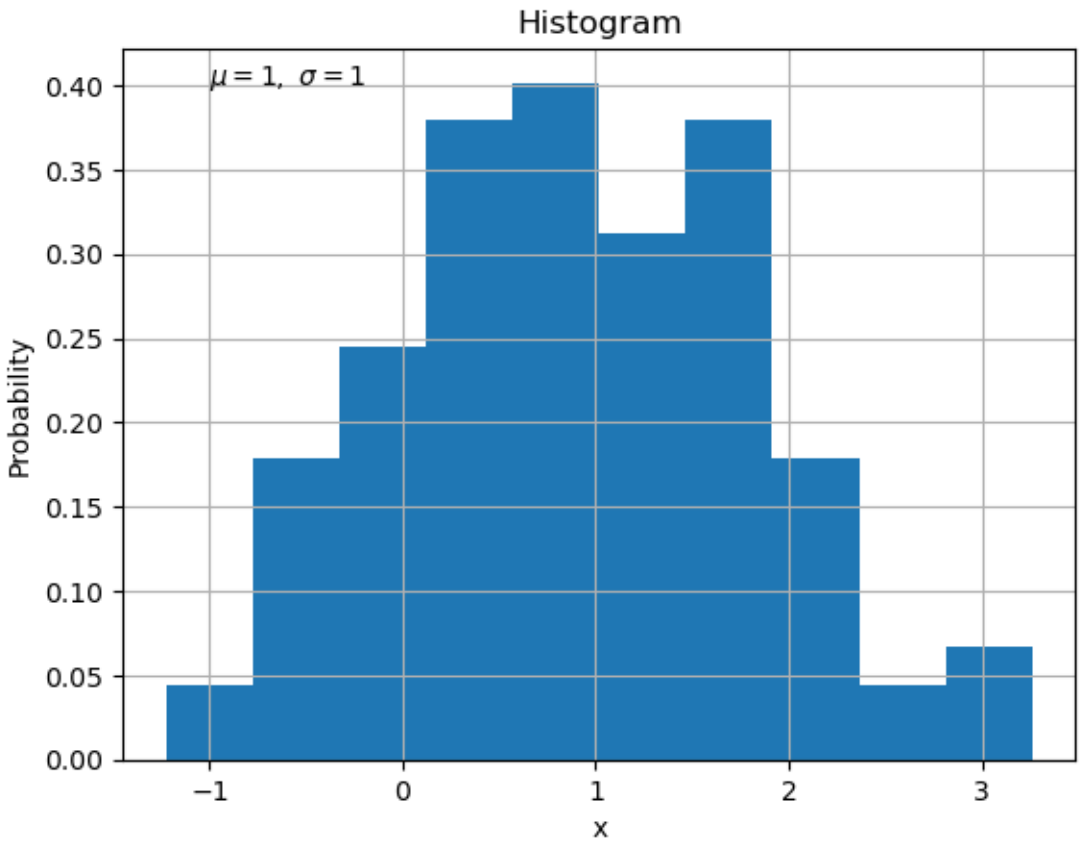
[44]:
```
fig, ax = plt.subplots()
ax.hist(x, density=True)
```

[44]:
```
(array([0.04461585, 0.17846339, 0.24538716, 0.3792347 , 0.40154263,
        0.31231093, 0.3792347 , 0.17846339, 0.04461585, 0.06692377]),
 array([-1.22340315, -0.77513194, -0.32686073,  0.12141048,  0.56968169,
         1.0179529 ,  1.46622411,  1.91449532,  2.36276653,  2.81103774,
         3.25930895]),
 <BarContainer object of 10 artists>)
```



[45]:
```
ax.hist?
```

[46]:
```
fig, ax = plt.subplots()
ax.hist(x, density=True)
ax.set_xlabel('x')
ax.set_ylabel('Probability')
ax.set_title('Histogram')
ax.text(-1, .4, r'$\mu=1,\ \sigma=1$')
ax.grid(True)
```



2.9 Image plotting

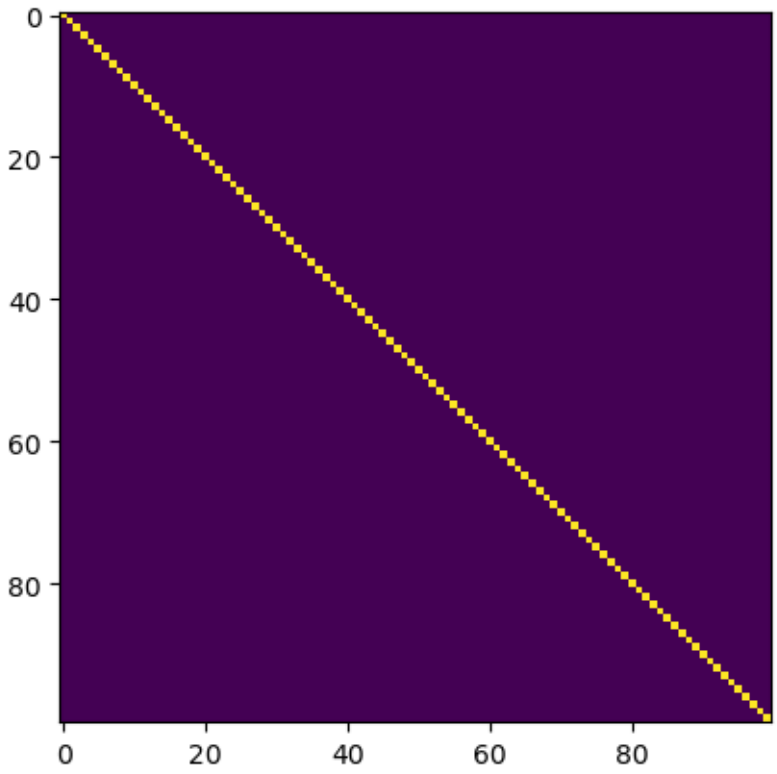
ax.imshow

[47]:
```
image = np.eye(100)
image
```

[47]:
```
array([[1., 0., 0., …, 0., 0., 0.],
       [0., 1., 0., …, 0., 0., 0.],
       [0., 0., 1., …, 0., 0., 0.],
       …,
       [0., 0., 0., …, 1., 0., 0.],
       [0., 0., 0., …, 0., 1., 0.],
       [0., 0., 0., …, 0., 0., 1.]])
```

[48]:
```
fig, ax = plt.subplots()
ax.imshow(image)
```

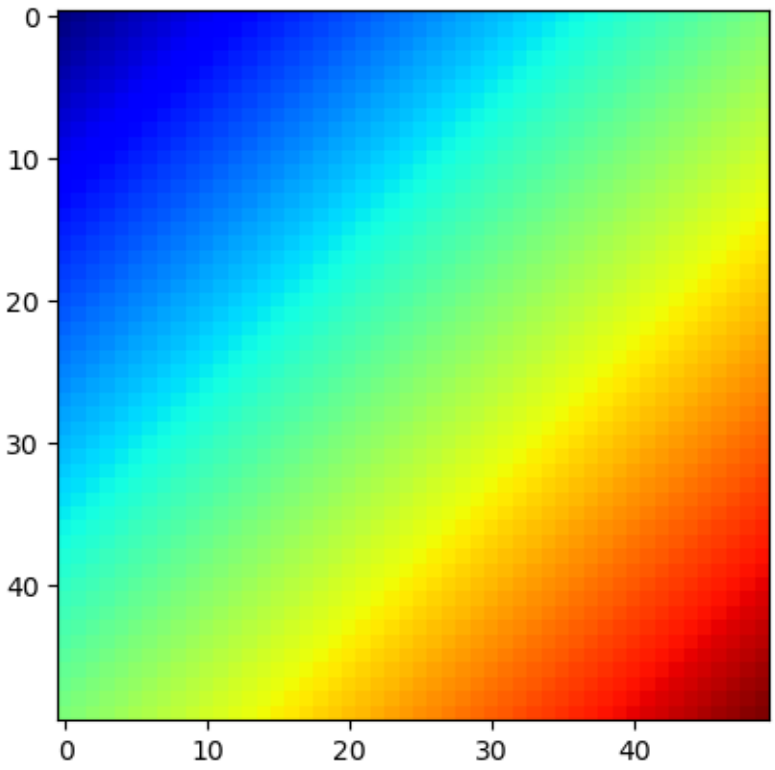
[48]:
```
<matplotlib.image.AxesImage at 0x7f9eaa1acac0>
```



[49]:
```
dim = 50
x_coords, y_coords = np.meshgrid(range(dim), range(dim), indexing='ij')
beta_low = np.zeros((dim,dim))
for i in range(x_coords.shape[0]):
    for j in range(x_coords.shape[1]):
        x = x_coords[i,j]
        y = y_coords[i,j]
        beta_low[i,j] = 1 + 1/24*(i+j)
```

[50]:
```
fig, ax = plt.subplots()
ax.imshow(beta_low,cmap="jet")
```

[50]: <matplotlib.image.AxesImage at 0x7f9e8867e220>

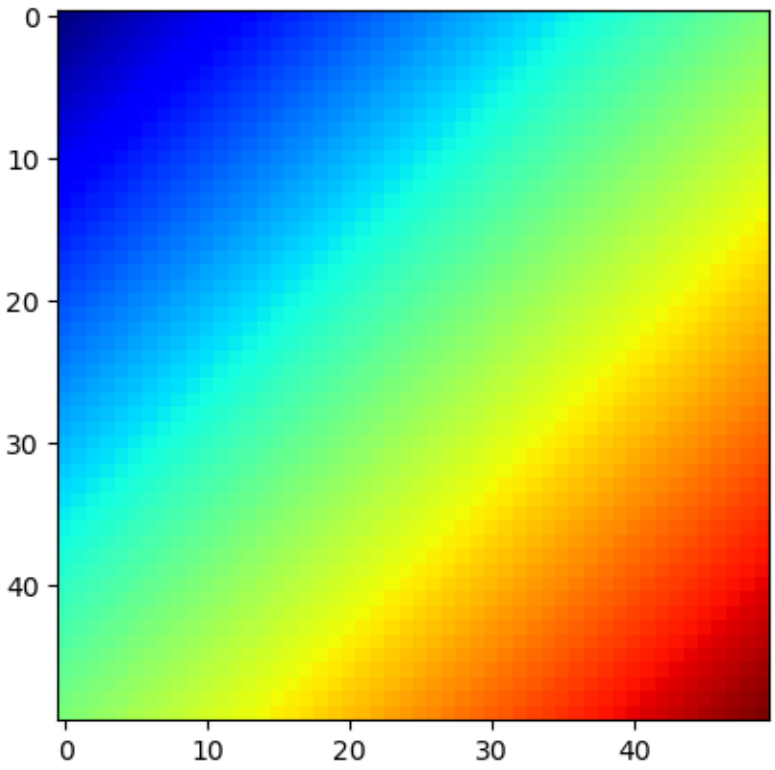


2.10 Saving a figure

`plt.savefig()`

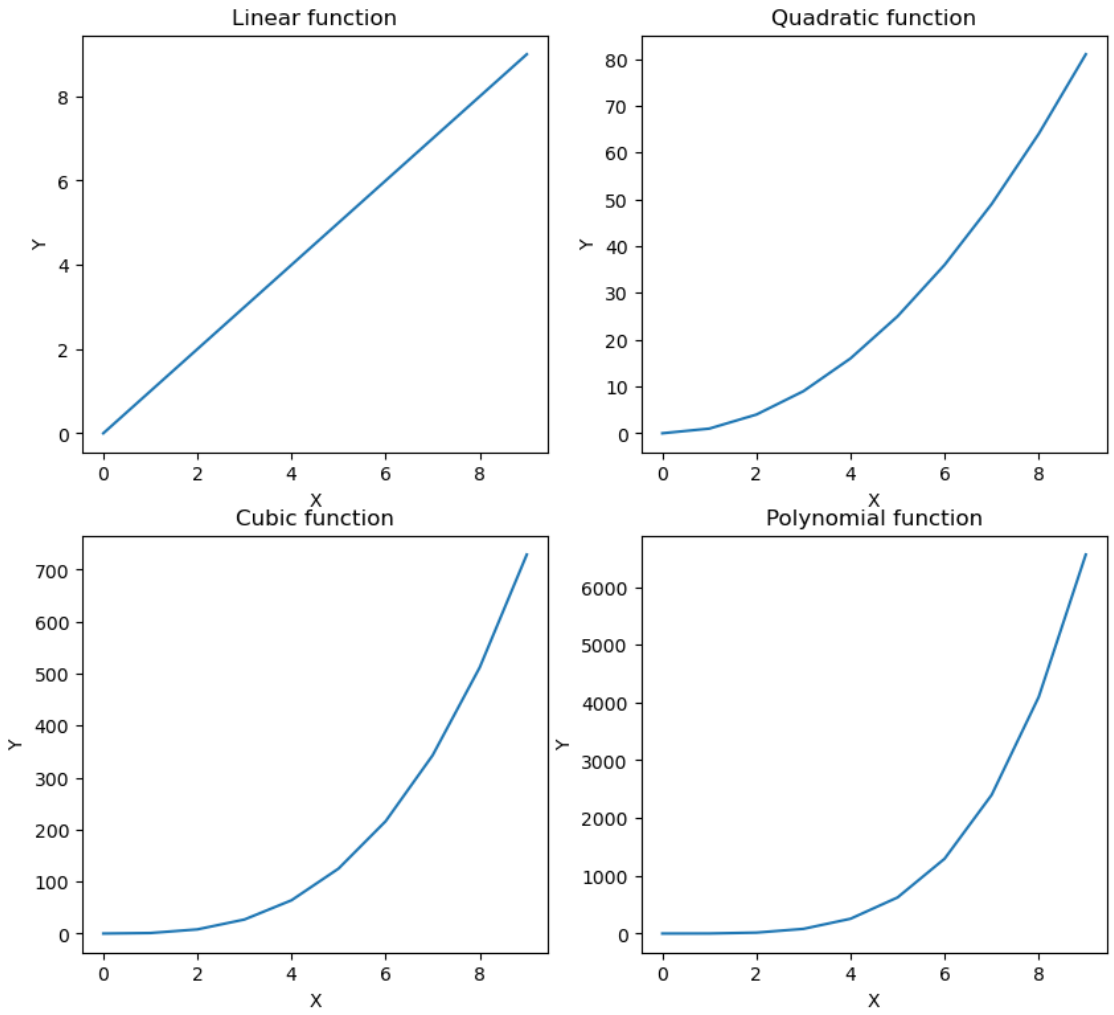
Save the current figure.

[51]:
```
fig, ax = plt.subplots()
ax.imshow(beta_low,cmap='jet')
plt.savefig("image.png")
```



[52]:
```
x = np.arange(10)

fig, axes = plt.subplots(2,2,figsize=(10,9))  # Create a figure containing four␣
↪axes.
ax1 = axes[0,0]
ax1.plot(x,x)
ax1.set_xlabel("X")
ax1.set_ylabel("Y")
ax1.set_title("Linear function")

ax2 = axes[0,1]
ax2.plot(x,x**2)
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.set_title("Quadratic function")

ax3 = axes[1,0]
ax3.plot(x,x**3)
ax3.set_xlabel("X")
ax3.set_ylabel("Y")
ax3.set_title("Cubic function")
```

```
ax4 = axes[1,1]
ax4.plot(x,x**4)
ax4.set_xlabel("X")
ax4.set_ylabel("Y")
ax4.set_title("Polynomial function")

plt.savefig("functions_4.png",dpi=500)
```



There is a lot more to matplotlib. One can visit the gallery and pull examples in to get a sense of what is possible, and how to adapt examples for your own purposes.