SCIENTIFIC COMPUTING WITH PYTHON

Data Visualization

Course Coordinator: Dr. R. Mariyal Jebasty
Assistant Professor,
Department of Physics
Wavoo Wajeeha College of Arts & Science
Kayalpatnam.

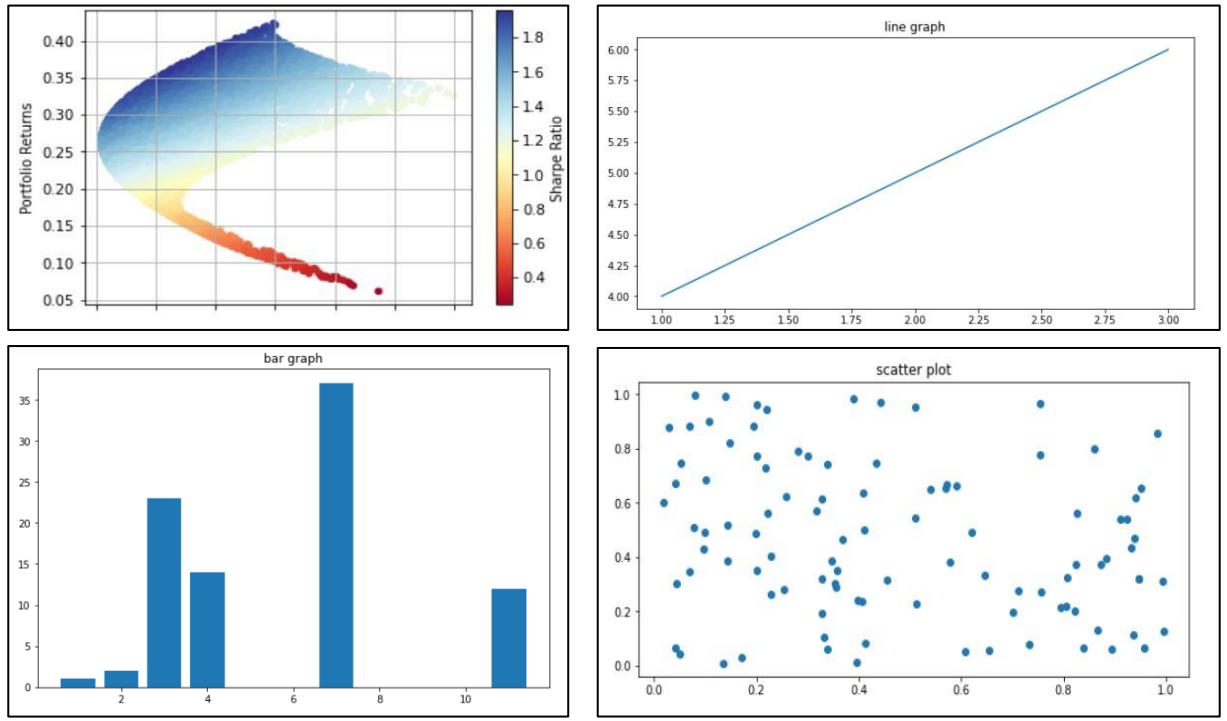
Course Instructors

1. Mrs. Pushpa, Assistant Professor in Physics, Wavoo Wajeeha Women's College of Arts & Science, Kayalpatnam.
2. Dr. S. Usharani , Assistant Professor in Physics, Wavoo Wajeeha Women's College of Arts & Science, Kayalpatnam.

DATA VISUALIZATION USING PYTHON

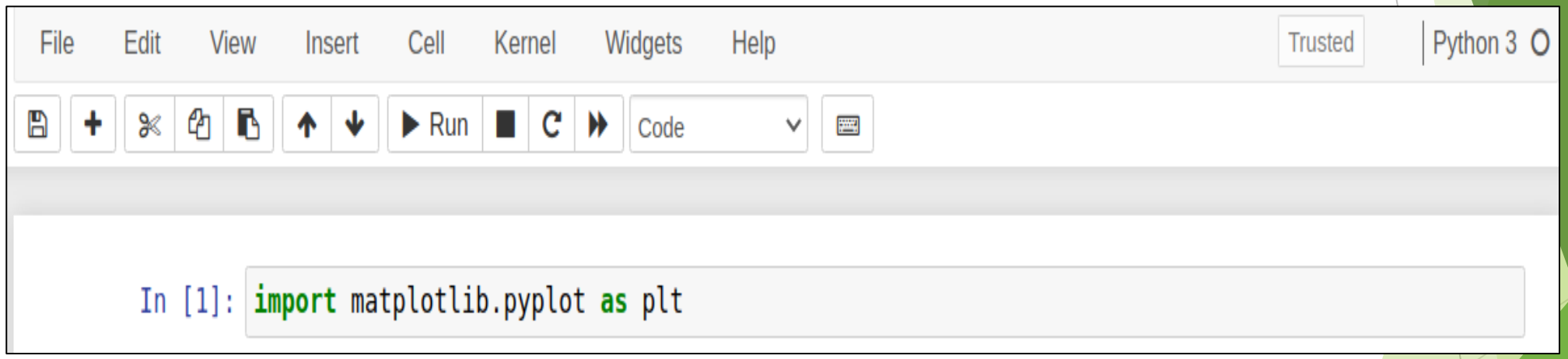
About Matplotlib

- Matplotlib is the most popular python library for plotting different kinds of graphs.
- The Pyplot module inside the Matplotlib makes it work like Matlab.



Importing Matplotlib

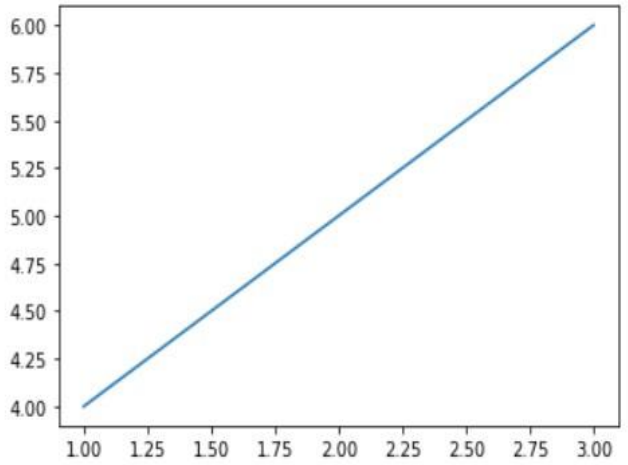
- To import matplotlib.pyplot, type ‘import matplotlib.pyplot’ in Jupyter Notebook and run the cell.
- The common abbreviation used for matplotlib.pyplot is plt.



Plotting Line Plots (1/2)

- We can plot line plots using matplotlib using the .plot() function.
  - The first argument in the .plot() function specifies the x-axis.
  - The second argument in the .plot() function specifies the y-axis.

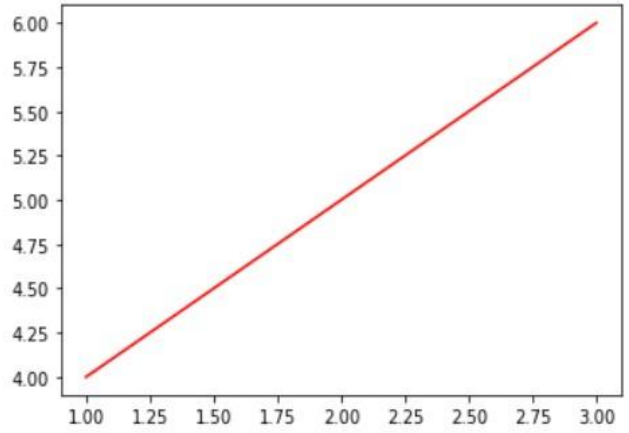
```
In [3]: x_axis = [1,2,3]
        y_axis = [4,5,6]

        plt.plot(x_axis, y_axis)

Out[3]: [<matplotlib.lines.Line2D at 0x7f186e0faf70>]
```



Plotting Line Plots (2/2)

- Changing Color
  - We can also change the color of the line by providing the color as third argument in the
  - plot() function.
  - A list of the color abbreviations can be found at: https://matplotlib.org/2.1.1/api/_as_gen/matplotlib.pyplot.plot.html

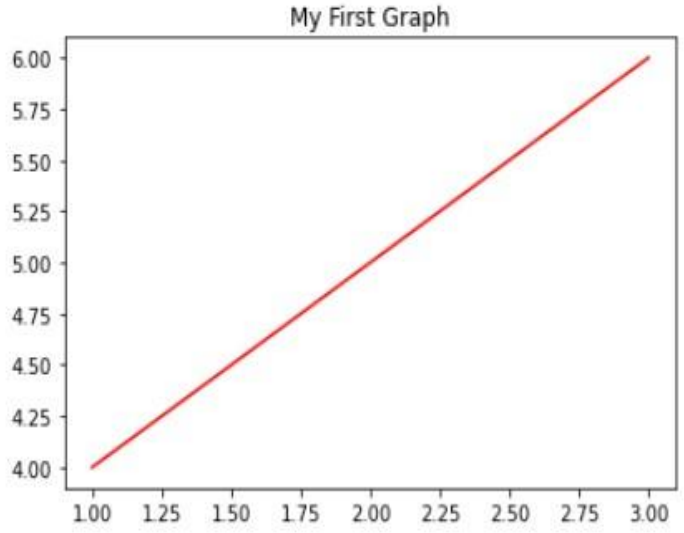
```
In [4]: x_axis = [1,2,3]
        y_axis = [4,5,6]

        plt.plot(x_axis, y_axis, 'r')

Out[4]: [<matplotlib.lines.Line2D at 0x7f186d85eb80>]
```



Title

- To set the title of the plot, use the .title() function.

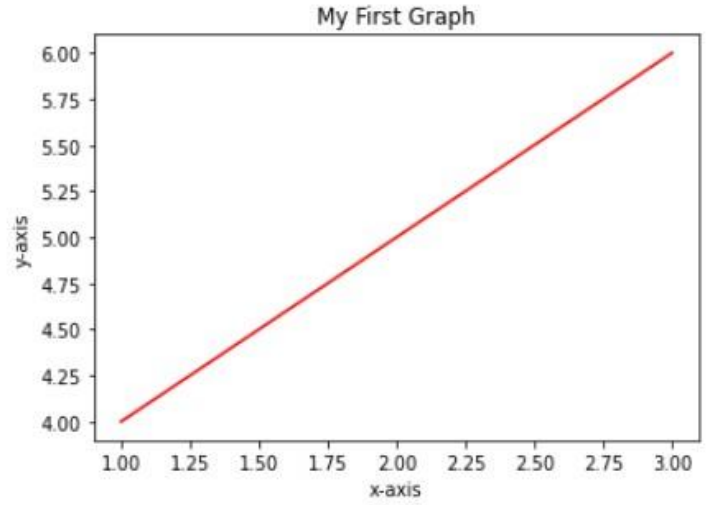
```
In [4]: x_axis = [1,2,3]
        y_axis = [4,5,6]

        plt.title('My First Graph')
        plt.plot(x_axis, y_axis, 'r')

Out[4]: [<matplotlib.lines.Line2D at 0x7efef8336700>]
```



Labels

- To assign labels to x and y-axis, use .xlabel() and .ylabel() respectively.

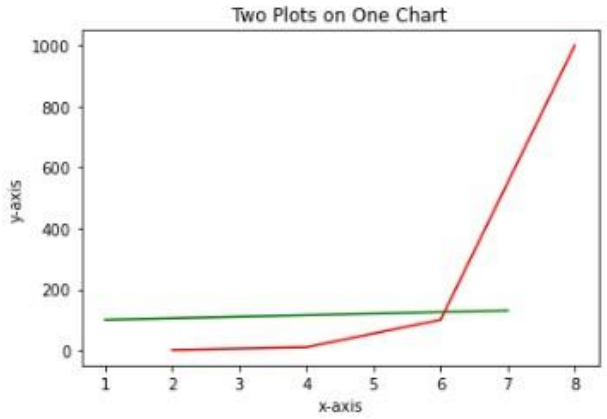
```
In [5]: x_axis = [1,2,3]
        y_axis = [4,5,6]

        plt.title('My First Graph')
        plt.xlabel('x-axis')
        plt.ylabel('y-axis')
        plt.plot(x_axis, y_axis, 'r')

Out[5]: [<matplotlib.lines.Line2D at 0x7efef82a4340>]
```



Legend (1/2)

- We can plot multiple plots on the same chart simply by plotting them one by one as in the given example.

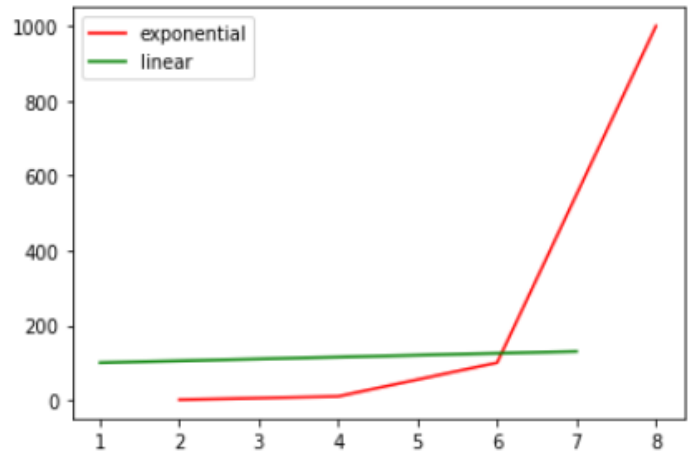
```
In [10]: x1_axis = [2, 4, 6, 8]
         y1_axis = [1, 10, 100, 1000]
         x2_axis = [1, 3, 5, 7]
         y2_axis = [100, 110, 120, 130]

         plt.title('Two Plots on One Chart')
         plt.xlabel('x-axis')
         plt.ylabel('y-axis')

         plt.plot(x1_axis, y1_axis, 'r')
         plt.plot(x2_axis, y2_axis, 'g')

Out[10]: [<matplotlib.lines.Line2D at 0x7efef81c7850>]
```



Legend (2/2)

- If plotting more than one plots, it is a good idea to add a legend in your figure using the .legend() function.

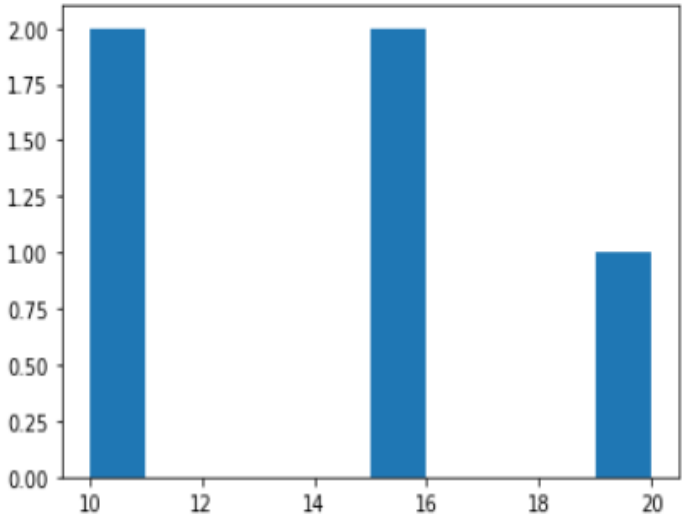
```
In [12]: x1_axis = [2, 4, 6, 8]
         y1_axis = [1, 10, 100, 1000]
         x2_axis = [1, 3, 5, 7]
         y2_axis = [100, 110, 120, 130]

         plt.plot(x1_axis, y1_axis, 'r')
         plt.plot(x2_axis, y2_axis, 'g')

         plt.legend(['exponential', 'linear'])

Out[12]: <matplotlib.legend.Legend at 0x7efef81037c0>
```



Plotting Histograms (1/3)

- We can also plot histograms using .hist() function.
- A histogram is generally used to plot frequency which helps identify distribution of data.

```
In [20]: values = [10, 15, 20, 10, 15]
         plt.hist(values)

Out[20]: (array([2., 0., 0., 0., 0., 2., 0., 0., 0., 1.]),
          array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
          <BarContainer object of 10 artists>)
```

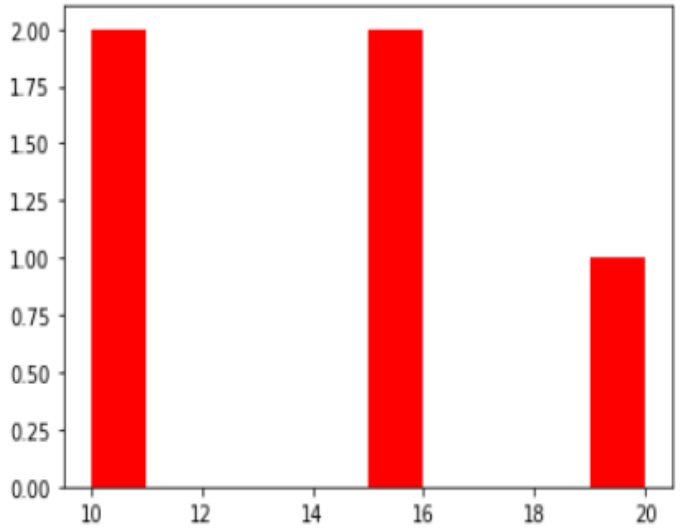


Plotting Histograms (2/3)

Changing Color

- We can also change color of the bars using the ‘color’ parameter inside the hist() function.

```
In [21]: values = [10, 15, 20, 10, 15]
         plt.hist(values, color='r')

Out[21]: (array([2., 0., 0., 0., 0., 2., 0., 0., 0., 1.]),
          array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
          <BarContainer object of 10 artists>)
```

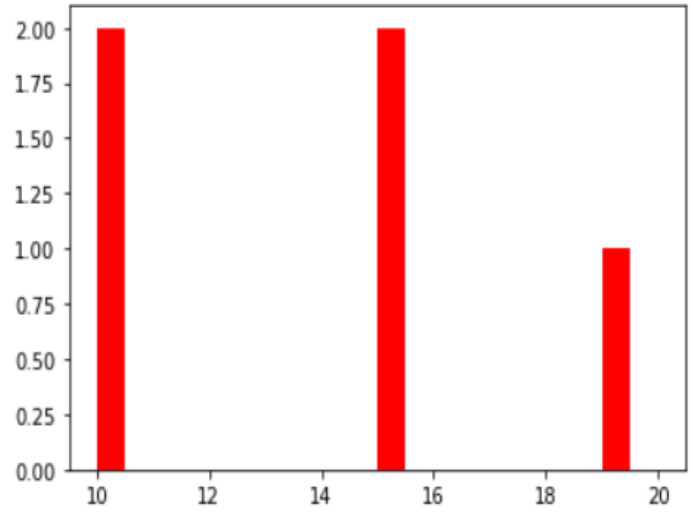


Plotting Histograms (3/3)

Changing Width

- We can also change width of the bars using the ‘width’ parameter inside the hist() function.

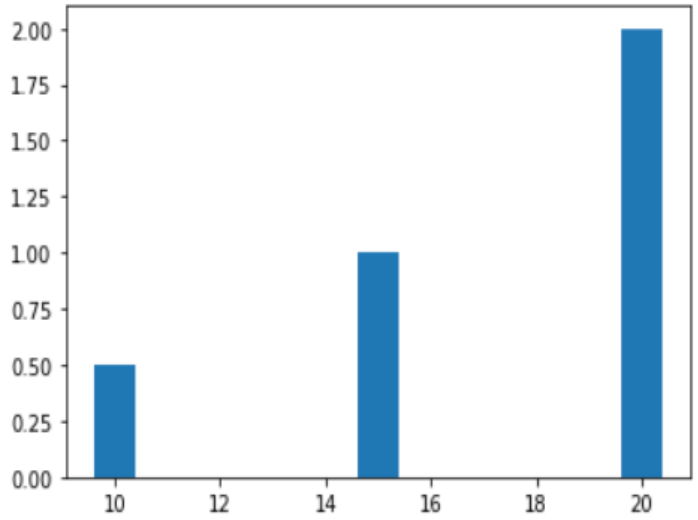
```
In [26]: values = [10, 15, 20, 10, 15]
         plt.hist(values, color='r', width=0.5)

Out[26]: (array([2., 0., 0., 0., 0., 2., 0., 0., 0., 1.]),
          array([10., 11., 12., 13., 14., 15., 16., 17., 18., 19., 20.]),
          <BarContainer object of 10 artists>)
```



Plotting Bar Charts (1/2)

- To plot a bar graph, use the .bar() function of the matplotlib.pyplot.
  - First argument in the bar() function is the x-label.
  - Second argument in the bar() function is the height of each bar, which can be a list of values or a single value.

```
In [33]: values = [10, 15, 20]
         plt.bar(values, [0.5, 1, 2])

Out[33]: <BarContainer object of 3 artists>
```

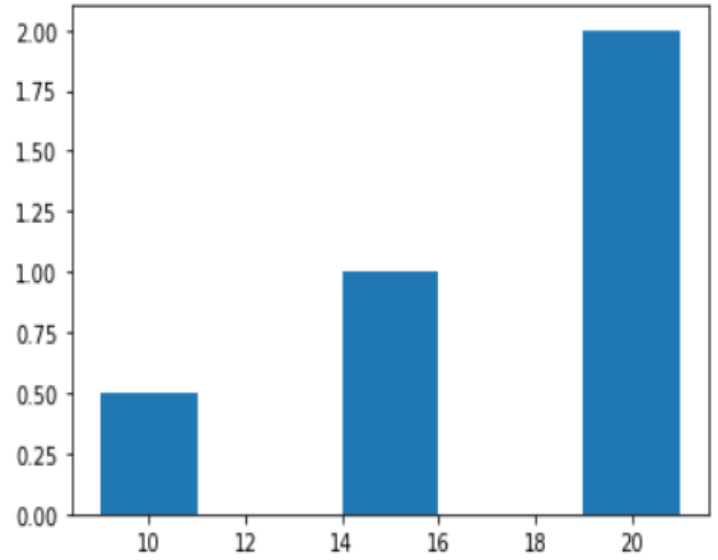


Plotting Bar Charts (2/2)

Changing Width

- We can also change the width of bars in the bar plot using the ‘width’ parameter.

```
In [34]: values = [10, 15, 20]
         plt.bar(values, [0.5, 1, 2], width=2)

Out[34]: <BarContainer object of 3 artists>
```



Plotting Pie Charts (1/4)

- .pie() function is used to create a pie chart in matplotlib.pyplot.

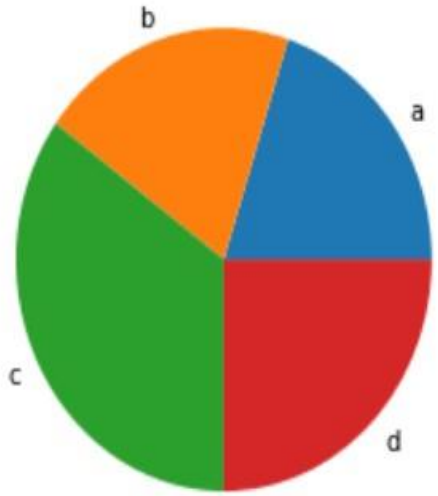
```
In [42]: values = [20, 20, 35, 25]
         plt.pie(values)
```

Plotting Pie Charts (2/4)

Labels

- To add labels in your pie chart, use the ‘labels’ parameter which takes a list of labels.

```
In [43]: values = [20, 20, 35, 25]
         plt.pie(values, labels=['a', 'b', 'c', 'd'])
```



Plotting Pie Charts (3/4)

Explode

- If you want one or more wedges of your pie chart to stand out, you can use the 'explode' parameter.
  - Provide a list containing distance of each wedge from the center to the 'explode' parameter.

```
In [44]: values = [20, 20, 35, 25]
         plt.pie(values, labels=['a', 'b', 'c', 'd'], explode=[0, 0.2, 0, 0])
```

Plotting Pie Charts (4/4)

Colors

- We can also change the colors of wedges by providing a list of colors to the ‘colors’ parameter.

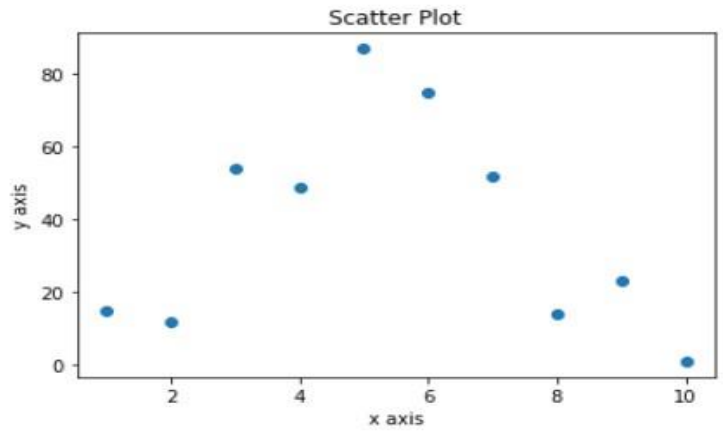
```
In [45]: values = [20, 20, 35, 25]
         plt.pie(values, labels=['a', 'b', 'c', 'd'], colors=['r', 'g', 'b', 'y'])
```

Plotting Scatter Plot (1/5)

- We can also plot scatter plots using the .scatter() function inside matplotlib.pyplot.
- It takes two lists as arguments;
  - First list specifies the values for the x-axis
  - Second list specifies the values for the y-axis

```
In [48]: x_axis = [1,2,3,4,5,6,7,8,9,10]
         y_axis = [15,12,54,49,87,75,52,14,23,1]

         plt.title('Scatter Plot')
         plt.xlabel('x axis')
         plt.ylabel('y axis')

         plt.scatter(x_axis, y_axis)
```

```
Out[48]: <matplotlib.collections.PathCollection at 0x7efef1670a00>
```

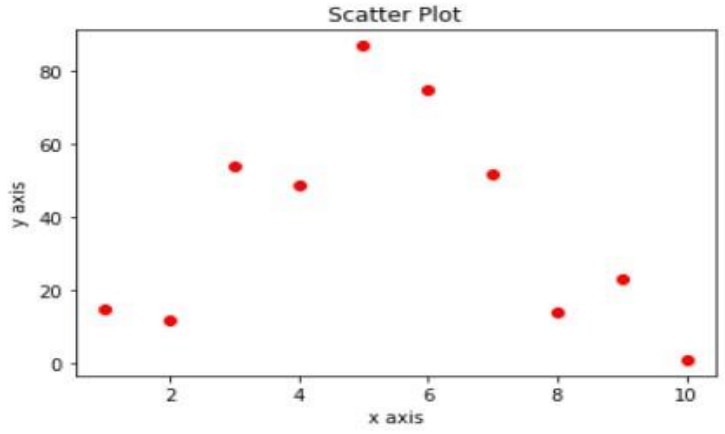


Plotting Scatter Plot (2/5)

Colors

- We can also change color of the dots using the ‘color’ parameter.

```
In [49]: x_axis = [1,2,3,4,5,6,7,8,9,10]
         y_axis = [15,12,54,49,87,75,52,14,23,1]

         plt.title('Scatter Plot')
         plt.xlabel('x axis')
         plt.ylabel('y axis')

         plt.scatter(x_axis, y_axis, color='r')

Out[49]: <matplotlib.collections.PathCollection at 0x7efef1b2dd00>
```

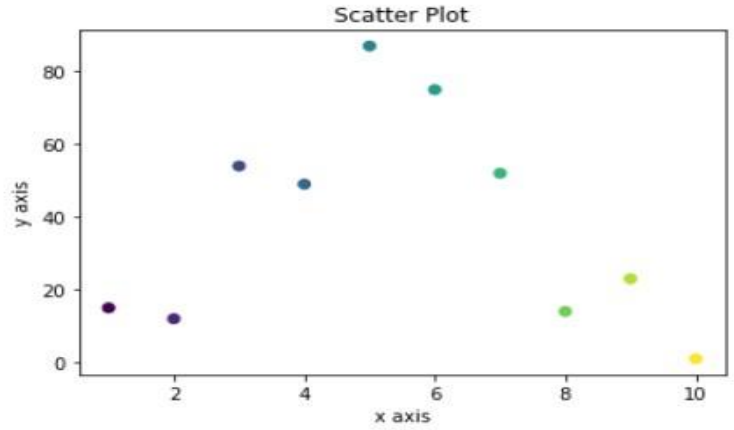


Plotting Scatter Plot (3/5)

Colormap

- If you would like to give each dot a different color, you can provide a list of colors or integers to the ‘c’ parameter.

```
In [52]: x_axis = [1,2,3,4,5,6,7,8,9,10]
         y_axis = [15,12,54,49,87,75,52,14,23,1]

         plt.title('Scatter Plot')
         plt.xlabel('x axis')
         plt.ylabel('y axis')

         plt.scatter(x_axis, y_axis, c=[1,2,3,4,5,6,7,8,9,10])

Out[52]: <matplotlib.collections.PathCollection at 0x7efef19bf610>
```

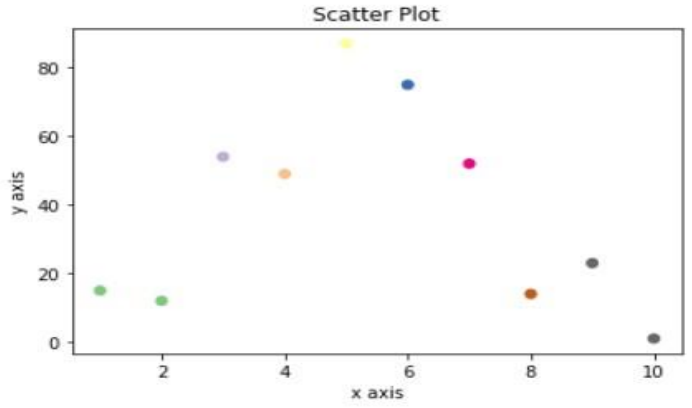


Plotting Scatter Plot (4/5)

Colormap

- You can also set the colormap using the ‘cmap’ parameter.
- For this, you will need to provide a list of integers that will be mapped to colors.
- We have used ‘Accent’ colormap, which is one of the many built-in colormaps in matplotlib.

```
In [53]: x_axis = [1,2,3,4,5,6,7,8,9,10]
         y_axis = [15,12,54,49,87,75,52,14,23,1]

         plt.title('Scatter Plot')
         plt.xlabel('x axis')
         plt.ylabel('y axis')

         plt.scatter(x_axis, y_axis, c=[1,2,3,4,5,6,7,8,9,10], cmap='Accent')

Out[53]: <matplotlib.collections.PathCollection at 0x7efef188d190>
```



Plotting Scatter Plot (5/5)

Colormap

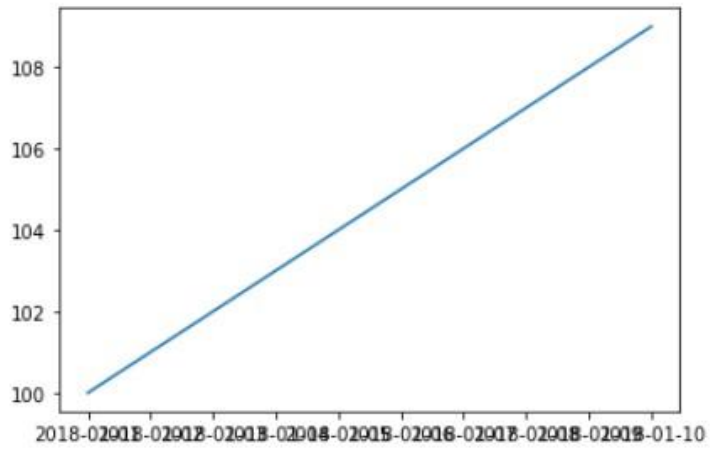
- To see the available colormaps in matplotlib, import ‘cm’ module from matplotlib.
- Give the ‘dir(cm)’ command and run the cell.
- A list of all the available colormaps will be displayed.

```
In [54]: from matplotlib import cm

In [55]: dir(cm)
            '__builtins__',
            '__cached__',
            '__doc__',
            '__file__',
            '__loader__',
            '__name__',
            '__package__',
            '__spec__',
            '_cmap_registry',
            '_gen_cmap_registry',
            '_reverser',
            'afmhot',
            'afmhot_r',
            'autumn',
            'autumn_r',
            'binary',
            'binary_r',
            'bone',
            'bone_r',
```

Handling Dates (1/2)

- Sometimes, there are too many values on the x-axis and it becomes difficult to distinguish them in the graph.
- This also happens when you have dates on the x-axis and they overlap as shown in the figure.

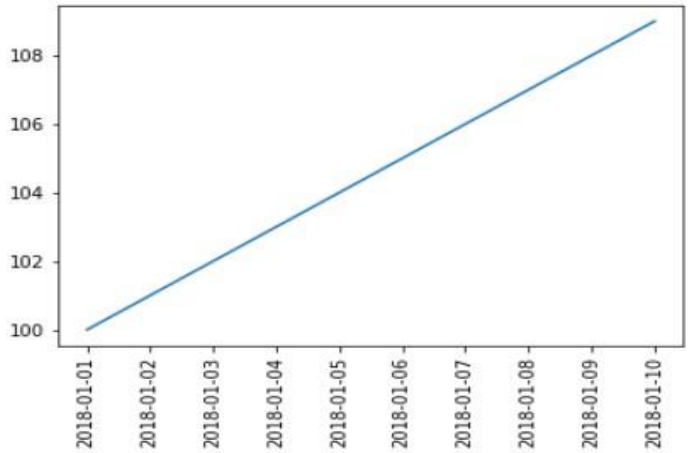
```
In [94]: dates = ['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04', '2018-01-05',
                  '2018-01-06', '2018-01-07', '2018-01-08', '2018-01-09', '2018-01-10']
         values = [100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

         plt.plot(dates, values)

Out[94]: [<matplotlib.lines.Line2D at 0x7efef13c0d60>]
```



Handling Dates (2/2)

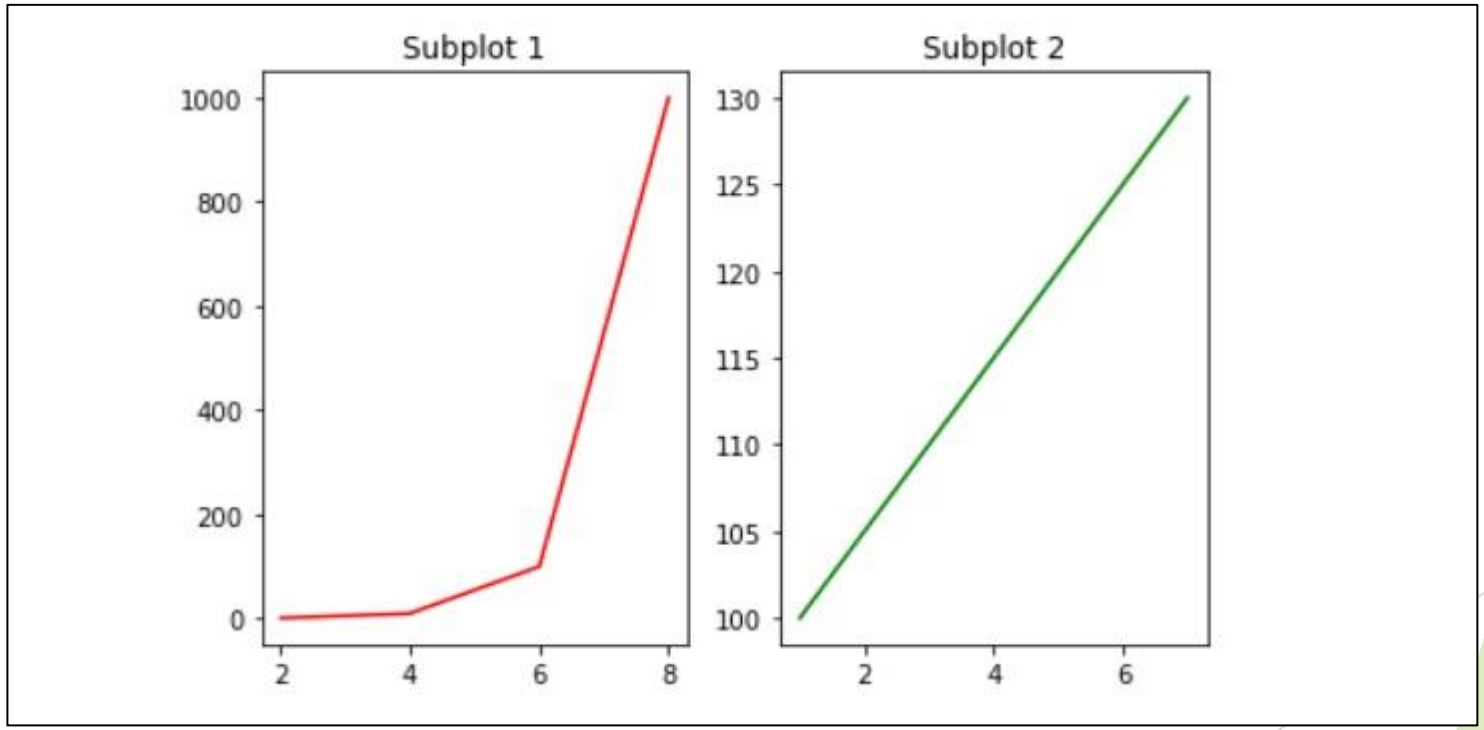
- We can avoid this by changing the orientation of the values on the x-axis using the .xticks() function.
  - .xticks() has a rotation parameter which can be used to rotate the values on the x-axis by a suitable angle.

```
In [96]: dates = ['2018-01-01', '2018-01-02', '2018-01-03', '2018-01-04', '2018-01-05',
                  '2018-01-06', '2018-01-07', '2018-01-08', '2018-01-09', '2018-01-10']
         values = [100, 101, 102, 103, 104, 105, 106, 107, 108, 109]

         plt.xticks(rotation=90)
         plt.plot(dates, values)

Out[96]: [<matplotlib.lines.Line2D at 0x7efef0dae430>]
```



Creating Multiple Subplots in One Figure (1/3)

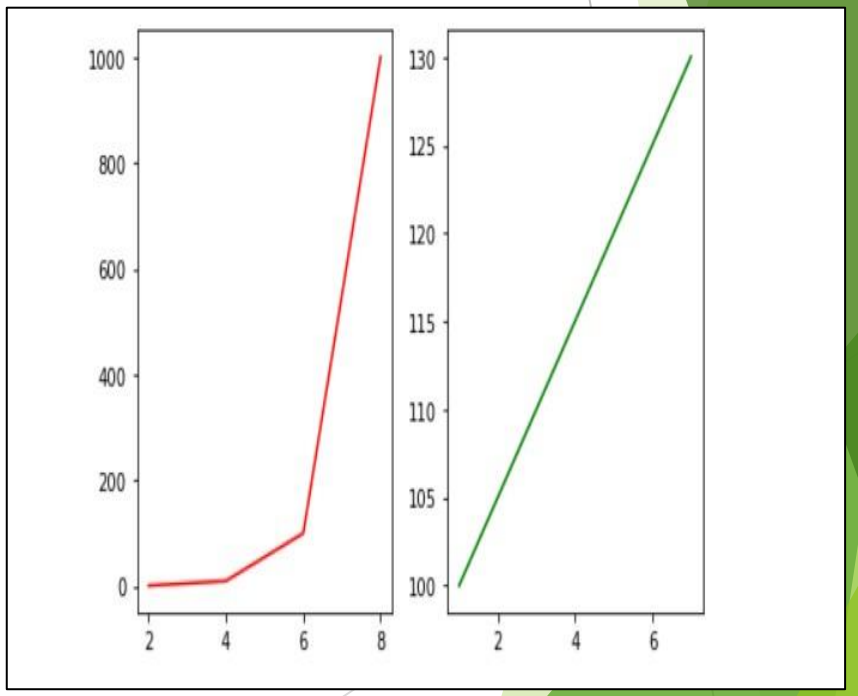
- We saw how we can plot multiple plots on the same chart, but sometimes we would like to have different charts for each plot.
- What we want to have actually is multiple subplots in the same figure, as shown in the given figure.



Creating Multiple Subplots in One Figure (2/3)

- To create subplots, we use the .subplot() function before plotting each graph.
- .subplot() takes 3 parameters;
  - First parameter is the number of rows you want to have in your figure.
  - Second parameter is the number of columns you want to have in your figure.
  - Third parameter is the id/position of the plot in the figure.

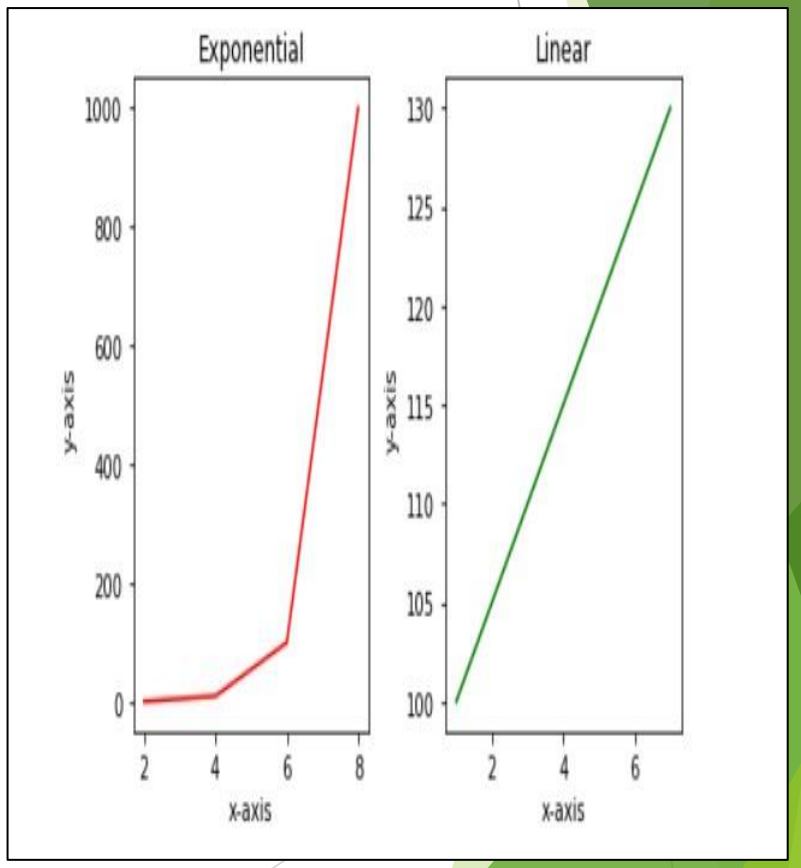
```
In [101]: x1_axis = [2, 4, 6, 8]
          y1_axis = [1, 10, 100, 1000]
          x2_axis = [1, 3, 5, 7]
          y2_axis = [100, 110, 120, 130]

          plt.subplot(1, 2, 1)
          plt.plot(x1_axis, y1_axis, 'r')

          plt.subplot(1, 2, 2)
          plt.plot(x2_axis, y2_axis, 'g')

          plt.tight_layout()
```



Creating Multiple Subplots in One Figure (3/3)

- We can have separate x and y labels as well as title for each subplot in the figure.

```
In [103]: x1_axis = [2, 4, 6, 8]
          y1_axis = [1, 10, 100, 1000]
          x2_axis = [1, 3, 5, 7]
          y2_axis = [100, 110, 120, 130]

          plt.subplot(1, 2, 1)
          plt.title('Exponential')
          plt.xlabel('x-axis')
          plt.ylabel('y-axis')
          plt.plot(x1_axis, y1_axis, 'r')

          plt.subplot(1, 2, 2)
          plt.title('Linear')
          plt.xlabel('x-axis')
          plt.ylabel('y-axis')
          plt.plot(x2_axis, y2_axis, 'g')

          plt.tight_layout()
```



Charting in Colaboratory

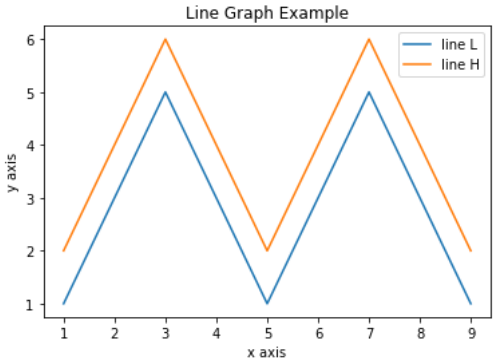
A common use for notebooks is data visualization using charts. Colaboratory makes this easy with several charting tools available as Python imports.

Matplotlib

Matplotlib is the most common charting package, see its documentation for details, and its examples for inspiration.

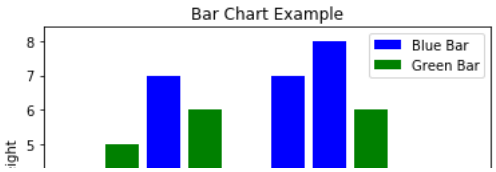
Line Plots

```
import matplotlib.pyplot as plt

x  = [1, 2, 3, 4, 5, 6, 7, 8, 9]
y1 = [1, 3, 5, 3, 1, 3, 5, 3, 1]
y2 = [2, 4, 6, 4, 2, 4, 6, 4, 2]
plt.plot(x, y1, label="line L")
plt.plot(x, y2, label="line H")
plt.plot()

plt.xlabel("x axis")
plt.ylabel("y axis")
plt.title("Line Graph Example")
plt.legend()
plt.show()
```



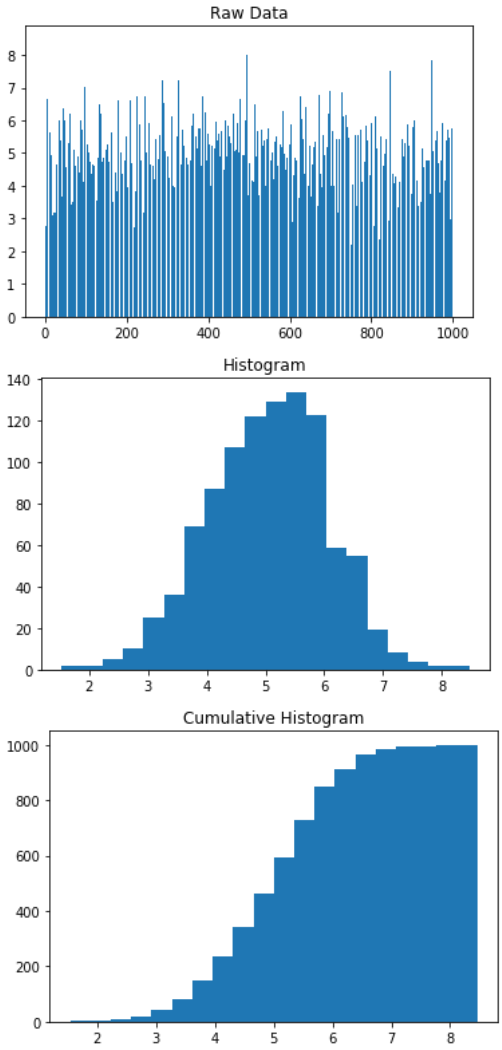
Bar Plots

```
import matplotlib.pyplot as plt

# Look at index 4 and 6, which demonstrate overlapping cases.
x1 = [1, 3, 4, 5, 6, 7, 9]
y1 = [4, 7, 2, 4, 7, 8, 3]

x2 = [2, 4, 6, 8, 10]
y2 = [5, 6, 2, 6, 2]

# Colors: https://matplotlib.org/api/colors_api.html

plt.bar(x1, y1, label="Blue Bar", color='b')
plt.bar(x2, y2, label="Green Bar", color='g')
plt.plot()

plt.xlabel("bar number")
plt.ylabel("bar height")
plt.title("Bar Chart Example")
plt.legend()
plt.show()
```



Histograms

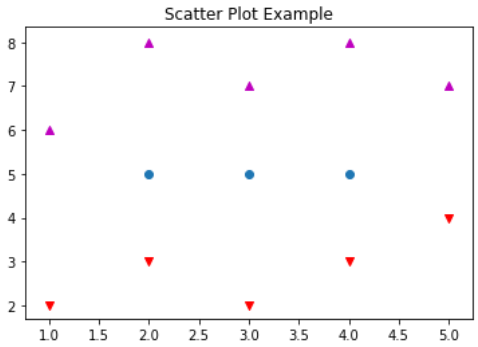
```
import matplotlib.pyplot as plt
import numpy as np

# Use numpy to generate a bunch of random data in a bell curve around 5.
n = 5 + np.random.randn(1000)

m = [m for m in range(len(n))]
plt.bar(m, n)
plt.title("Raw Data")
plt.show()

plt.hist(n, bins=20)
plt.title("Histogram")
plt.show()

plt.hist(n, cumulative=True, bins=20)
plt.title("Cumulative Histogram")
plt.show()
```



Scatter Plots

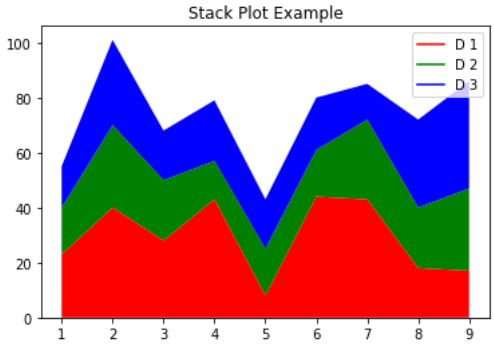
```
import matplotlib.pyplot as plt

x1 = [2, 3, 4]
y1 = [5, 5, 5]
```

```
x2 = [1, 2, 3, 4, 5]
y2 = [2, 3, 2, 3, 4]
y3 = [6, 8, 7, 8, 7]

# Markers: https://matplotlib.org/api/markers_api.html

plt.scatter(x1, y1)
plt.scatter(x2, y2, marker='v', color='r')
plt.scatter(x2, y3, marker='^', color='m')
plt.title('Scatter Plot Example')
plt.show()
```



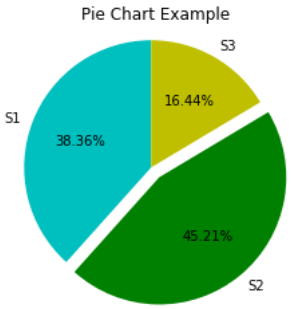
Stack Plots

```
import matplotlib.pyplot as plt

idxes = [ 1,  2,  3,  4,  5,  6,  7,  8,  9]
arr1  = [23, 40, 28, 43,  8, 44, 43, 18, 17]
arr2  = [17, 30, 22, 14, 17, 17, 29, 22, 30]
arr3  = [15, 31, 18, 22, 18, 19, 13, 32, 39]

# Adding legend for stack plots is tricky.
plt.plot([], [], color='r', label = 'D 1')
plt.plot([], [], color='g', label = 'D 2')
plt.plot([], [], color='b', label = 'D 3')

plt.stackplot(idxes, arr1, arr2, arr3, colors= ['r', 'g', 'b'])
plt.title('Stack Plot Example')
plt.legend()
plt.show()
```



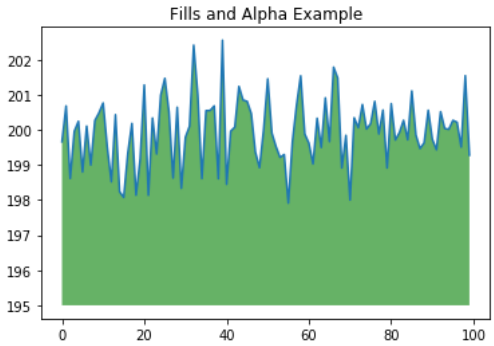
Pie Charts

```
import matplotlib.pyplot as plt

labels = 'S1', 'S2', 'S3'
sections = [56, 66, 24]
colors = ['c', 'g', 'y']

plt.pie(sections, labels=labels, colors=colors,
        startangle=90,
        explode = (0, 0.1, 0),
        autopct = '%1.2f%%')

plt.axis('equal') # Try commenting this out.
plt.title('Pie Chart Example')
plt.show()
```



fill_between and alpha

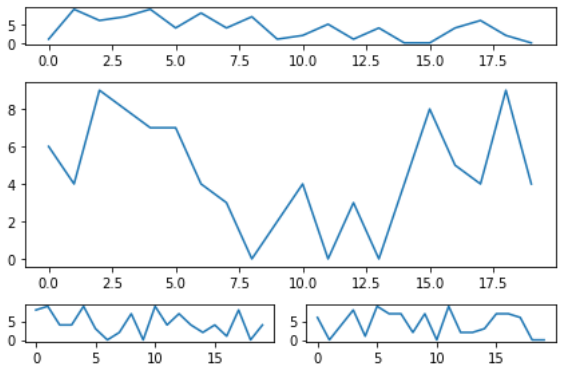
```
import matplotlib.pyplot as plt
import numpy as np

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)

plt.title("Fills and Alpha Example")
plt.show()
```



Subplotting using Subplot2grid

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

def random_plots():
  xs = []
  ys = []

  for i in range(20):
    x = i
    y = np.random.randint(10)

    xs.append(x)
    ys.append(y)

  return xs, ys

fig = plt.figure()
ax1 = plt.subplot2grid((5, 2), (0, 0), rowspan=1, colspan=2)
ax2 = plt.subplot2grid((5, 2), (1, 0), rowspan=3, colspan=2)
ax3 = plt.subplot2grid((5, 2), (4, 0), rowspan=1, colspan=1)
ax4 = plt.subplot2grid((5, 2), (4, 1), rowspan=1, colspan=1)

x, y = random_plots()
ax1.plot(x, y)

x, y = random_plots()
ax2.plot(x, y)

x, y = random_plots()
ax3.plot(x, y)

x, y = random_plots()
ax4.plot(x, y)
```

```
plt.tight_layout()
plt.show()
```



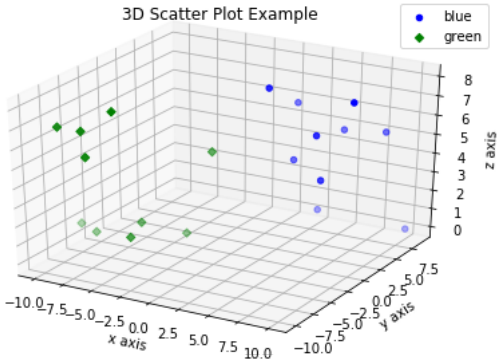
Plot styles

Colaboratory charts use Seaborn's custom styling by default. To customize styling further please see the matplotlib docs.

3D Graphs

3D Scatter Plots

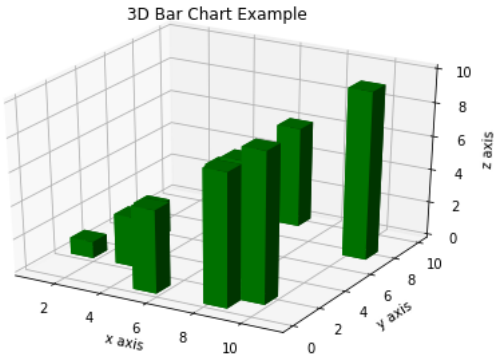
```
import matplotlib.pyplot as plt
import numpy as np
from mpl_toolkits.mplot3d import axes3d

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y1 = np.random.randint(10, size=10)
z1 = np.random.randint(10, size=10)

x2 = [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
y2 = np.random.randint(-10, 0, size=10)
z2 = np.random.randint(10, size=10)

ax.scatter(x1, y1, z1, c='b', marker='o', label='blue')
ax.scatter(x2, y2, z2, c='g', marker='D', label='green')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Scatter Plot Example")
plt.legend()
plt.tight_layout()
plt.show()
```



3D Bar Plots

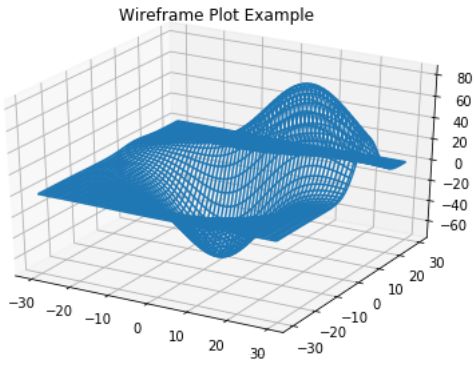
```
import matplotlib.pyplot as plt
import numpy as np
```

```
fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
y = np.random.randint(10, size=10)
z = np.zeros(10)

dx = np.ones(10)
dy = np.ones(10)
dz = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

ax.bar3d(x, y, z, dx, dy, dz, color='g')

ax.set_xlabel('x axis')
ax.set_ylabel('y axis')
ax.set_zlabel('z axis')
plt.title("3D Bar Chart Example")
plt.tight_layout()
plt.show()
```



Wireframe Plots

```
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection = '3d')

x, y, z = axes3d.get_test_data()

ax.plot_wireframe(x, y, z, rstride = 2, cstride = 2)

plt.title("Wireframe Plot Example")
plt.tight_layout()
plt.show()
```

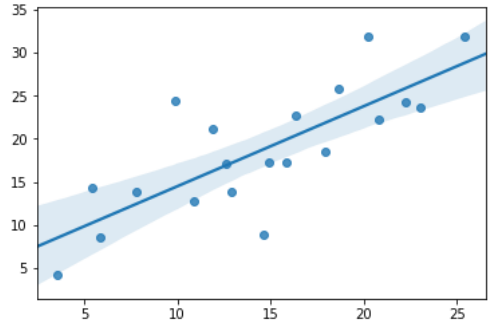


Seaborn

There are several libraries layered on top of Matplotlib that you can use in Colab. One that is worth highlighting is Seaborn:

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

# Generate some random data
num_points = 20
# x will be 5, 6, 7... but also twiddled randomly
x = 5 + np.arange(num_points) + np.random.randn(num_points)
```

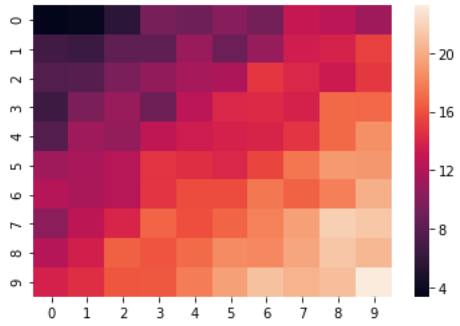
```
# y will be 10, 11, 12... but twiddled even more randomly
y = 10 + np.arange(num_points) + 5 * np.random.randn(num_points)
sns.regplot(x, y)
plt.show()
```



That's a simple scatterplot with a nice regression line fit to it, all with just one call to Seaborn's regplot.

Here's a Seaborn heatmap:

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

# Make a 10 x 10 heatmap of some random data
side_length = 10
# Start with a 10 x 10 matrix with values randomized around 5
data = 5 + np.random.randn(side_length, side_length)
# The next two lines make the values larger as we get closer to (9, 9)
data += np.arange(side_length)
data += np.reshape(np.arange(side_length), (side_length, 1))
# Generate the heatmap
sns.heatmap(data)
plt.show()
```

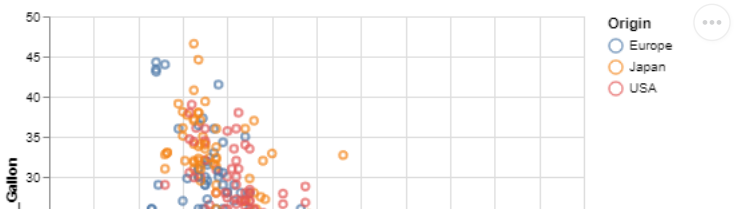


Altair

Altair is a declarative visualization library for creating interactive visualizations in Python, and is installed and enabled in Colab by default.

For example, here is an interactive scatter plot:

```
import altair as alt
from vega_datasets import data
cars = data.cars()

alt.Chart(cars).mark_point().encode(
    x='Horsepower',
    y='Miles_per_Gallon',
    color='Origin',
).interactive()
```

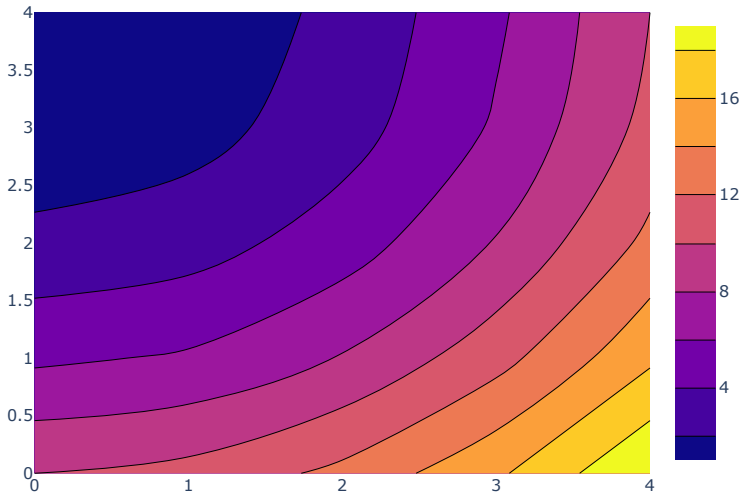


For more examples of Altair plots, see the Altair snippets notebook or the external Altair Example Gallery.

Plotly

Sample

```
from plotly.offline import iplot
import plotly.graph_objs as go

data = [
    go.Contour(
        z=[[10, 10.625, 12.5, 15.625, 20],
           [5.625, 6.25, 8.125, 11.25, 15.625],
           [2.5, 3.125, 5., 8.125, 12.5],
           [0.625, 1.25, 3.125, 6.25, 10.625],
           [0, 0.625, 2.5, 5.625, 10]]
    )
]
iplot(data)
```

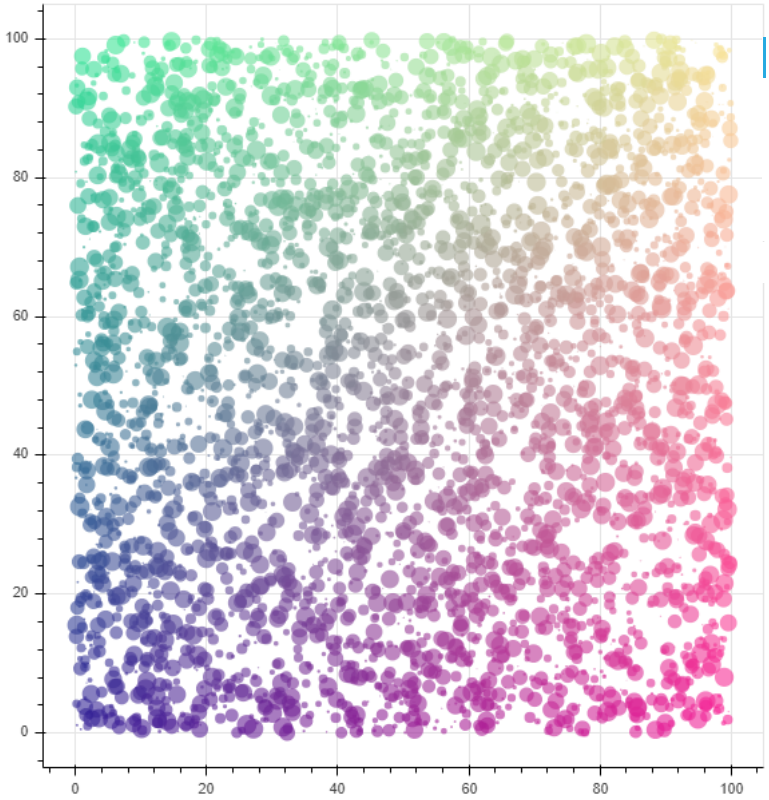


Bokeh

Sample

```
import numpy as np
from bokeh.plotting import figure, show
from bokeh.io import output_notebook

# Call once to configure Bokeh to display plots inline in the notebook.
output_notebook()


N = 4000
x = np.random.random(size=N) * 100
y = np.random.random(size=N) * 100
radii = np.random.random(size=N) * 1.5
```

```
colors = ["#%02x%02x%02x" % (r, g, 150) for r, g in zip(np.floor(50+2*x).astype(int), np.floor(30+2*y).astype(int))]

p = figure()
p.circle(x, y, radius=radii, fill_color=colors, fill_alpha=0.6, line_color=None)
show(p)
```



For More Details

https://data-flair.training/blogs/python-libraries/

Thank You