Topic 3

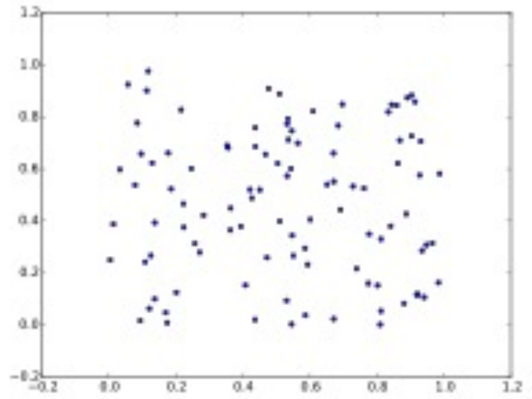
Data Visualization Using Matplotlib

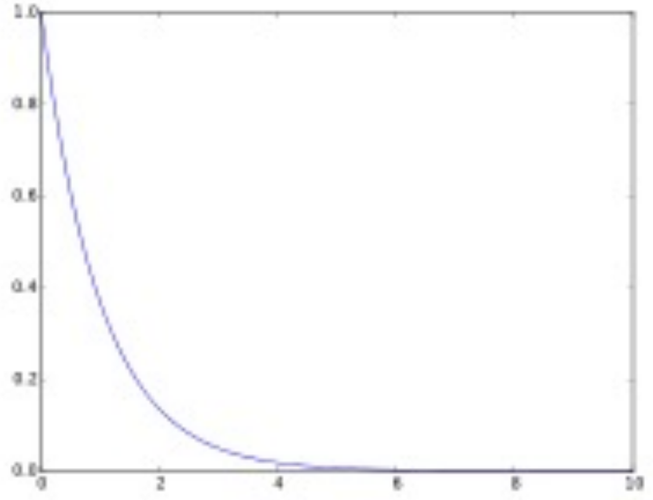
Contents

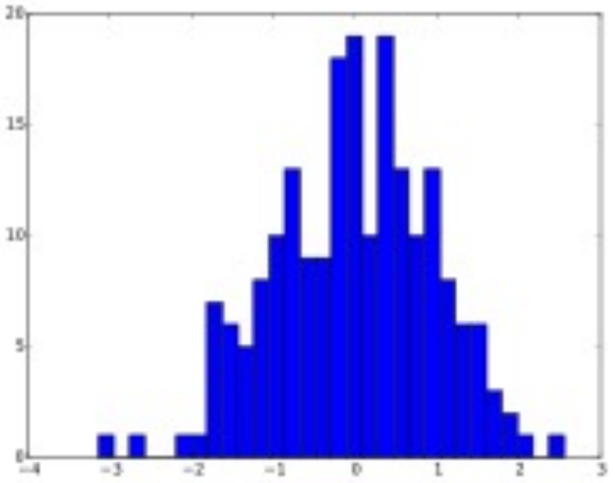
Intro to Matplotlib

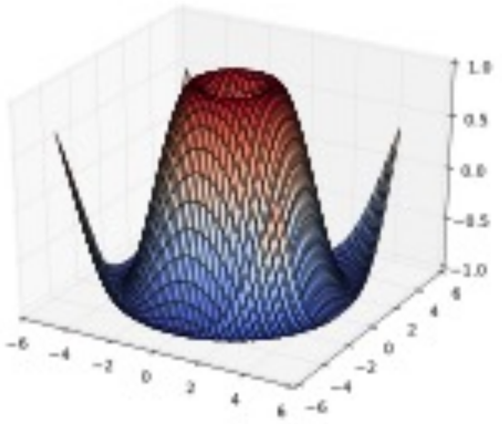
What is Matplotlib?

- Matplotlib is a library for making 2D plots of arrays in Python
- The latest version of Matplotlib (as of Sep 2020) is 3.3.2









Matplotlib is both easy yet powerful

- Produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms
- Can be used in Python scripts, Jupyter notebook, web application servers, and four graphical user interface toolkits
- For simple plotting, the pyplot module provides a MATLAB-like interface, particularly when combined with IPython
- For the power user, you have full control of line styles, font properties, axes properties, etc, via an object oriented interface or via a set of functions familiar to MATLAB users

Installing Matplotlib

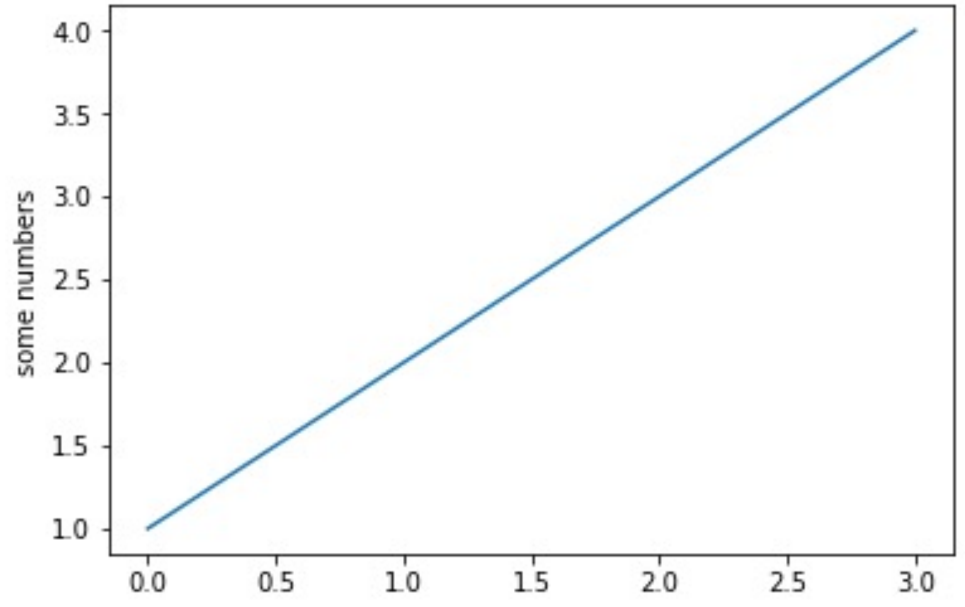
- If you don’t already have Python installed, start off with Anaconda, which has matplotlib and many of its dependencies, plus other useful packages, preinstalled.
- For standard Python installations, install matplotlib using pip:

```
pip install matplotlib
```

A first taste of Matplotlib

Plot a line graph (supply y values only)

```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

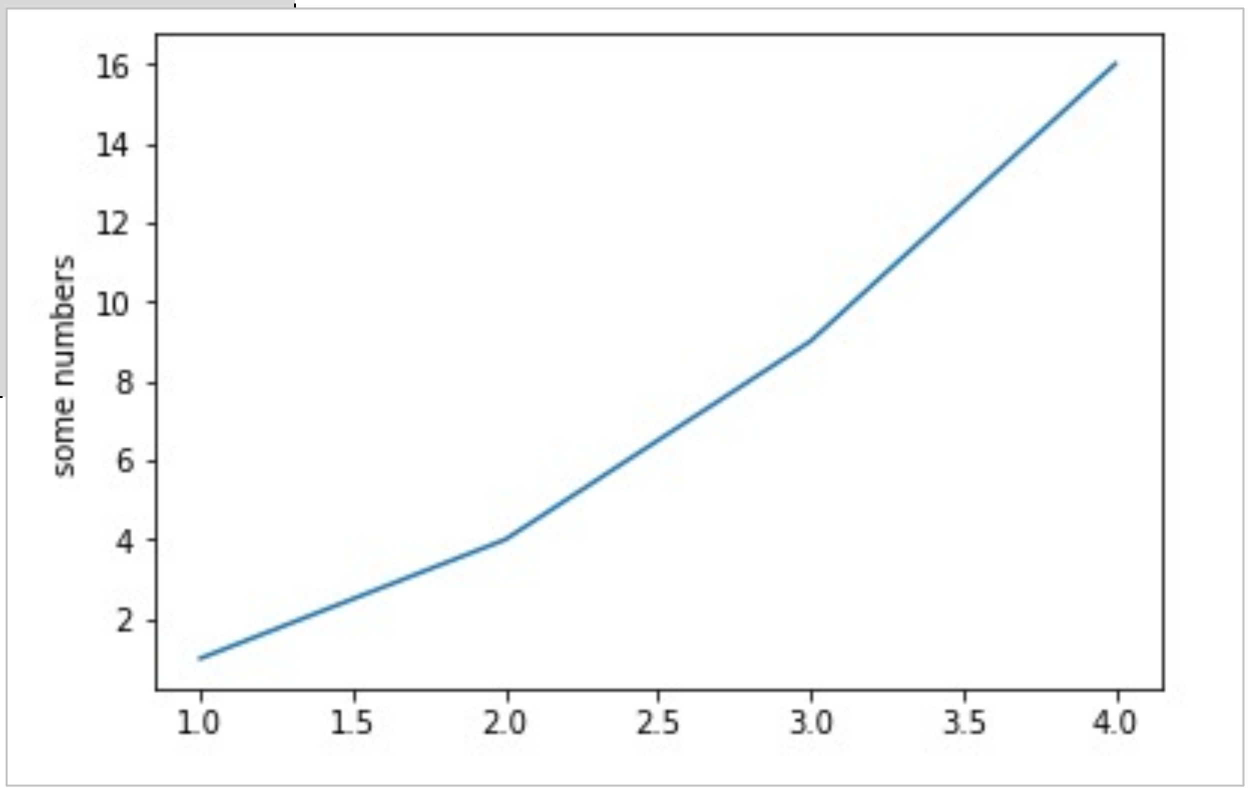


- If you provide a single list or array to the plot() command, matplotlib assumes it is a sequence of y values, and automatically generates the x values for you
- Since python ranges start with 0, the default x vector has the same length as y but starts with 0
- Hence the x data are [0,1,2,3].

Plot a line graph (supply y+x values)

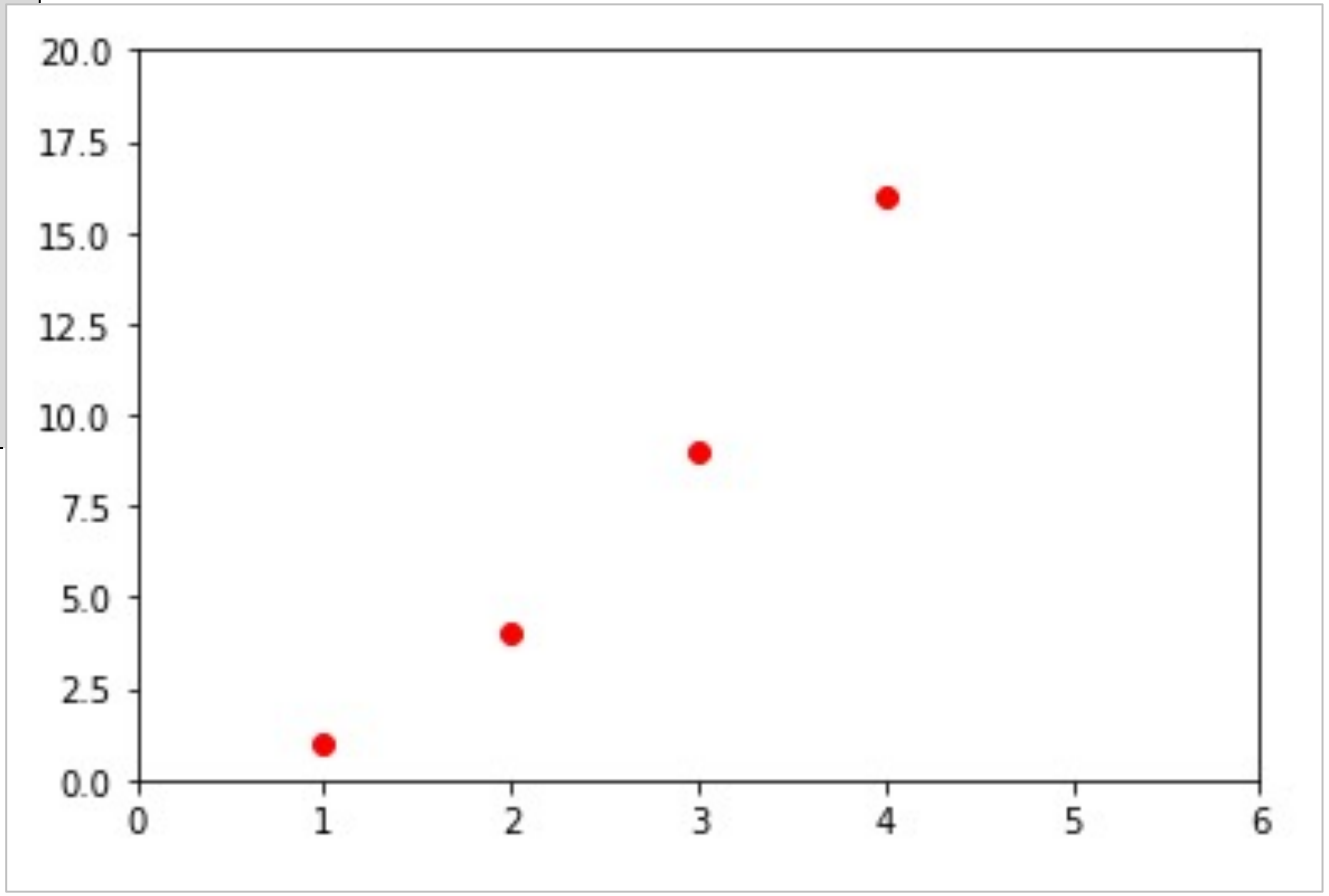
```
import matplotlib.pyplot as plt

plt.plot([1, 2, 3, 4],
         [1, 4, 9, 16])
plt.ylabel('some numbers')
plt.show()
```

- To plot x versus y, you can issue the commands above



Plot a line graph (specify color and pattern)

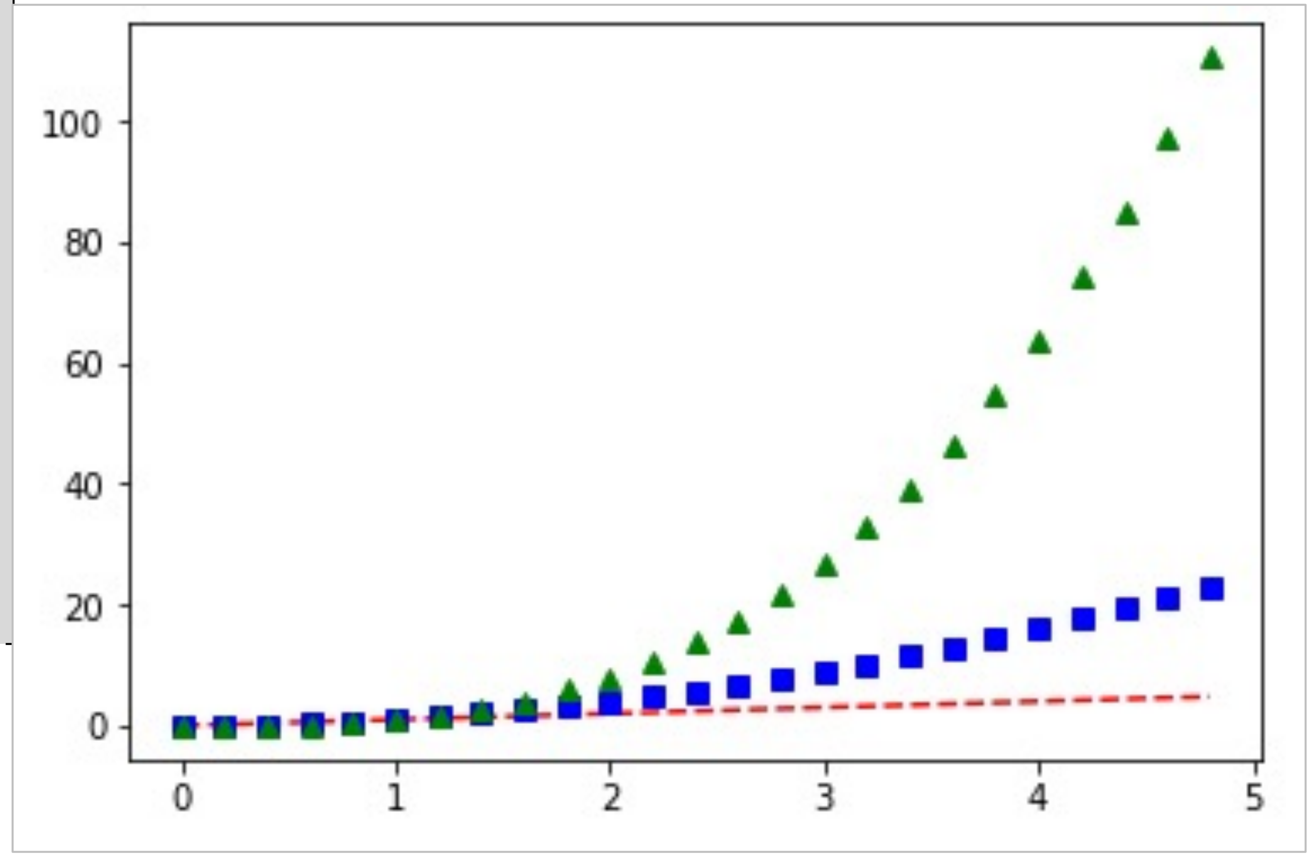
```
import matplotlib.pyplot as plt

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



- You can specify a third argument which is the format string that indicates the color and line type of the plot
- The default format string is 'b-' which is a solid blue line
- To plot the above with red circles, you would use 'ro' instead
- Refer to link below for more marker styles https://matplotlib.org/api/markers_api.html

Plot a line graph (with numpy arrays)

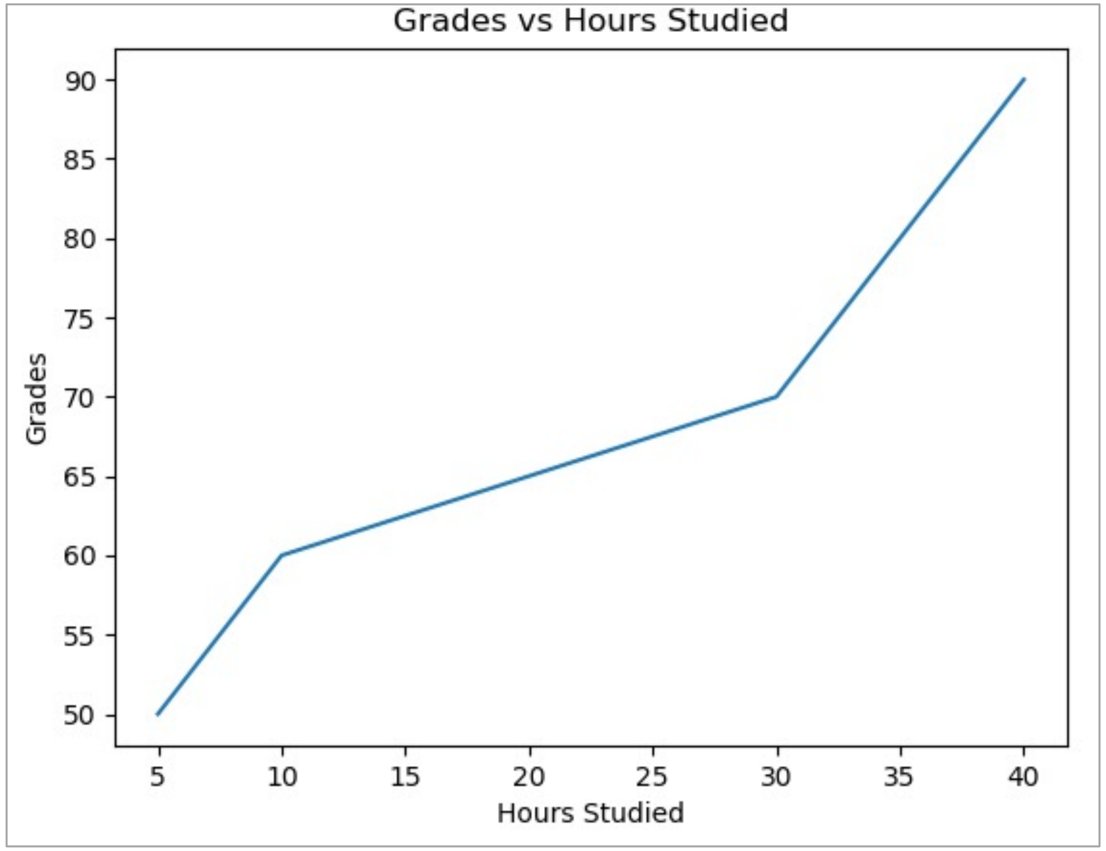
```
import numpy as np
import matplotlib.pyplot as plt

# Create numpy array with values
# between 0 and 5 in steps of 0.2
t = np.arange(0, 5, 0.2)

plt.plot(t, t, 'r--')
plt.plot(t, t**2, 'bs')
plt.plot(t, t**3, 'g^')
plt.show()
```



- Though the previous examples show how matplotlib is used with lists, generally, matplotlib is used with numpy arrays for data processing
- The example above illustrates a plotting several lines with different format styles using numpy arrays.

Add titles and labels

```
import matplotlib.pyplot as plt

plt.title('Grades vs Hours Studied')
plt.xlabel('Hours Studied')
plt.ylabel('Grades')

plt.plot([5,10,30,40], [50,60,70,90])
plt.show()
```



The following format string characters are accepted to control the line style or marker

	description
'-'	**solid line style**
'--'	**dashed line style**
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	**circle marker**
's'	**square marker**
'p'	pentagon marker
'*'	**star marker**
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker

	description
'v'	triangle_down marker
'^'	**triangle_up marker**
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

The following color abbreviations are supported

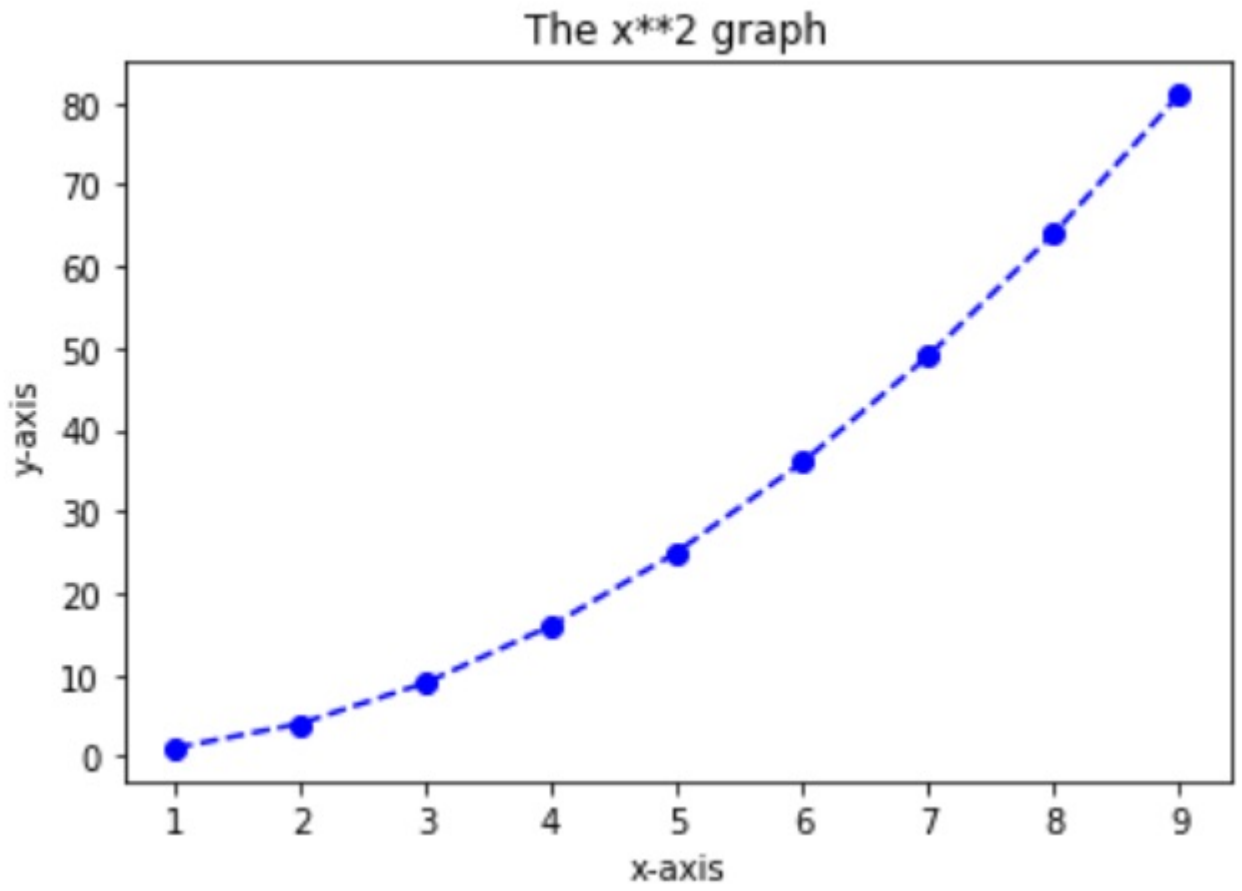
In addition, you can specify colors in other ways

- full names ('green')
- hex strings ('#008000')
- RGB or RGBA tuples ((0,1,0,1))
- grayscale intensities as a string ('0.8')

	color
b’	blue
‘g’	green
‘r’	red
‘c’	cyan
‘m’	magenta
‘y’	yellow
‘k’	black
‘w’	white

Class Exercise

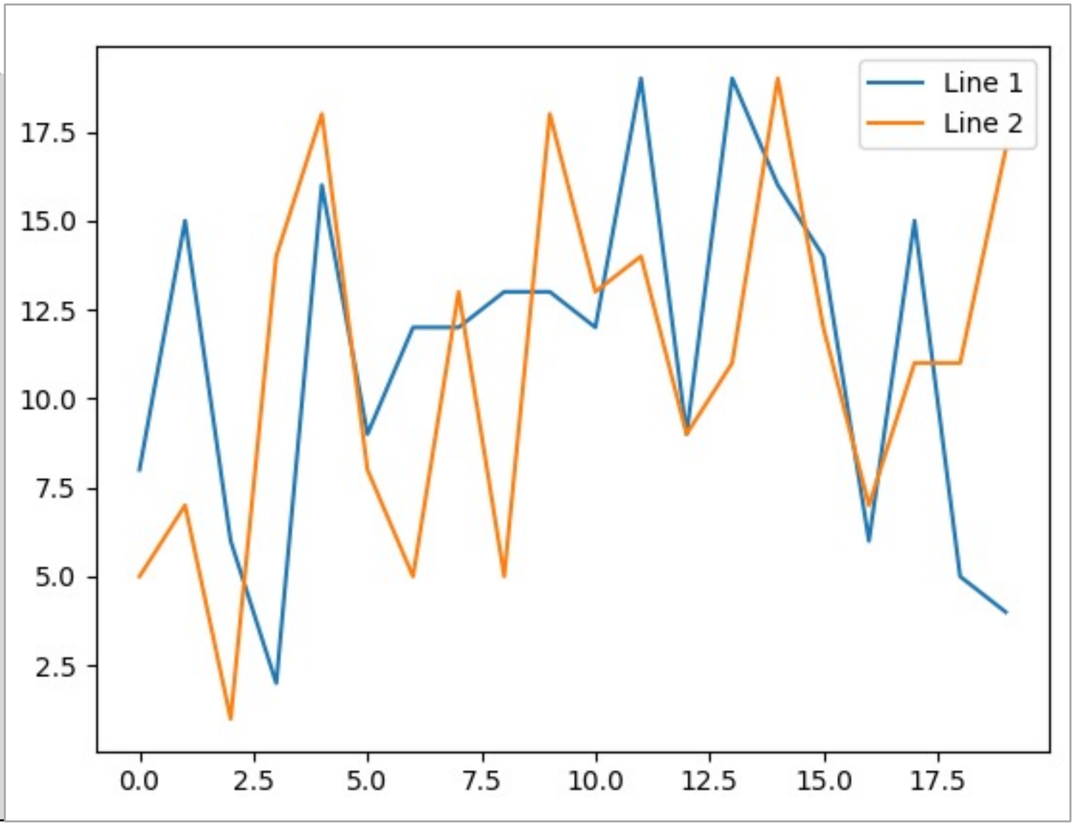
- Draw the following with the help of numpy package



Add legend to a plot (Example 1)

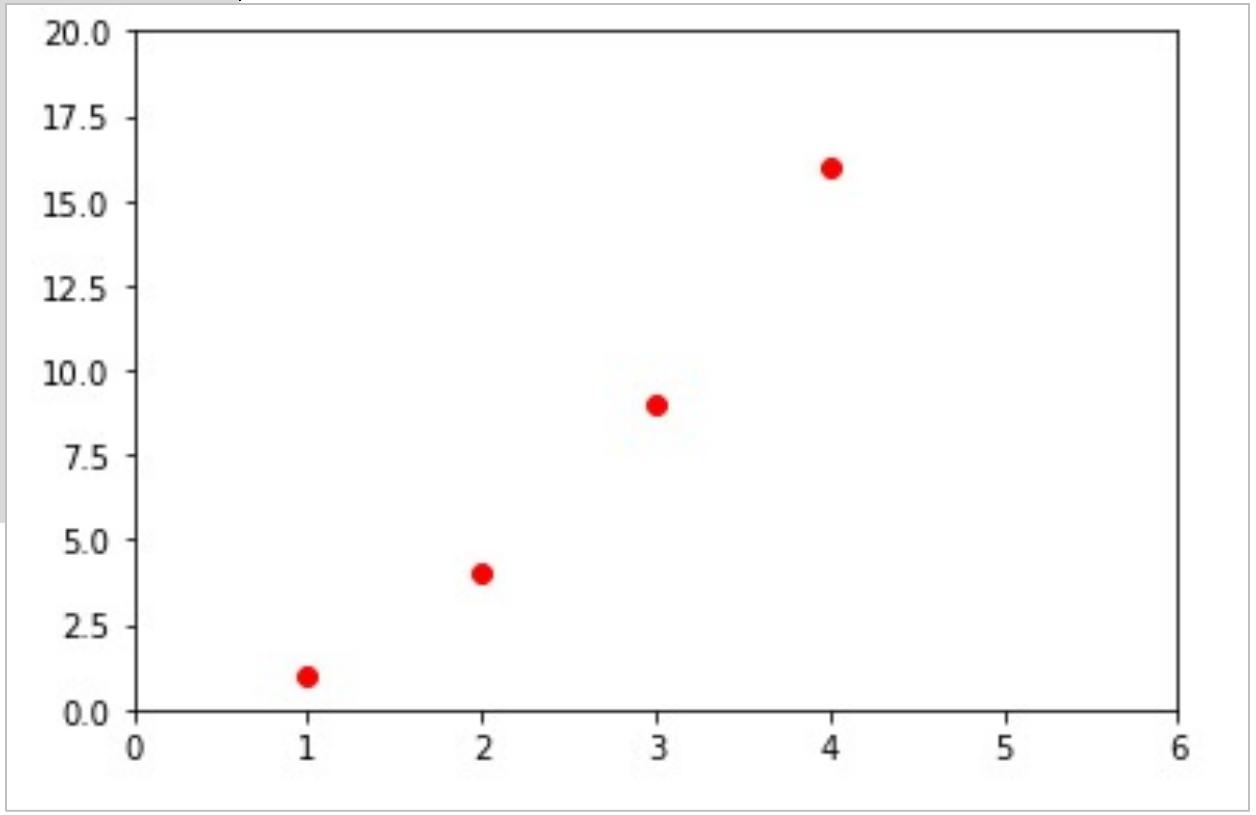
- Call **plt.legend** to add a legend to a plot

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

line_1 = np.random.randint(1,20,20)
line_2 = np.random.randint(1,20,20)
plt.plot(line_1,label="Line 1")
plt.plot(line_2,label="Line 2")

legend = plt.legend()
plt.show()
```



Set axis limits (ylim and xlim method)

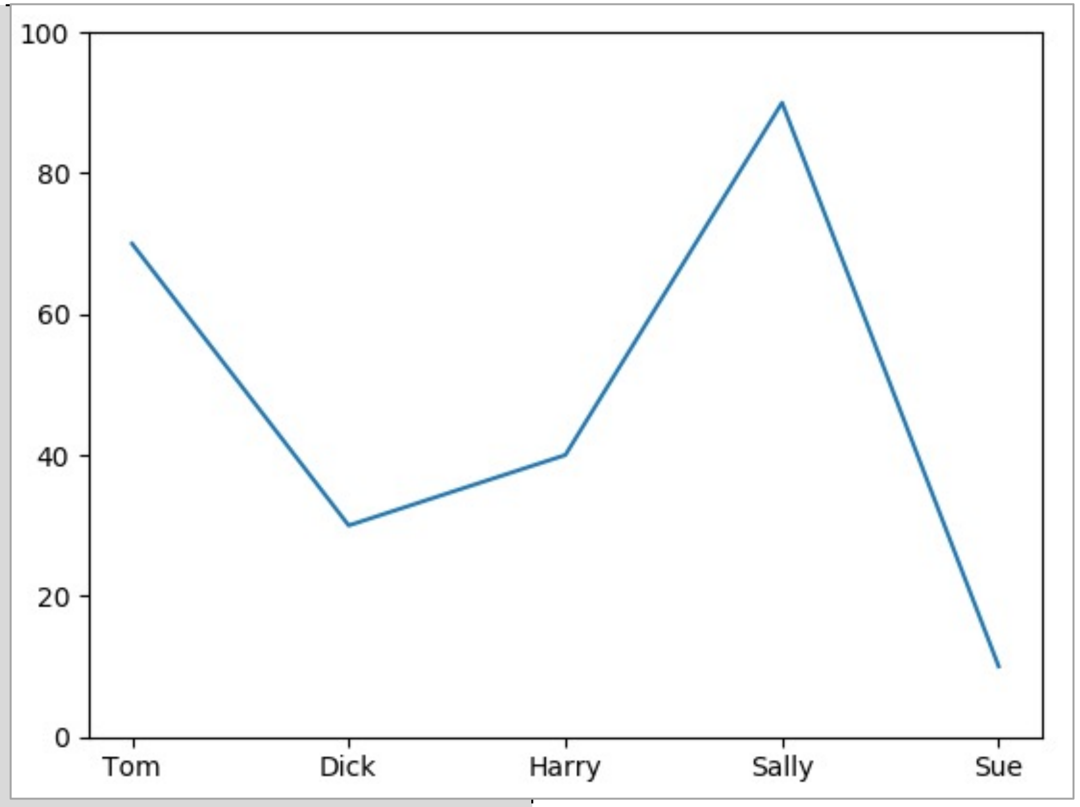
```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4], [1,4,9,16], 'ro')
plt.xlim(0,6)
plt.ylim(0,20)
plt.show()
```



Add x-ticks

- Call **plt.xticks** to add xticks to a plot

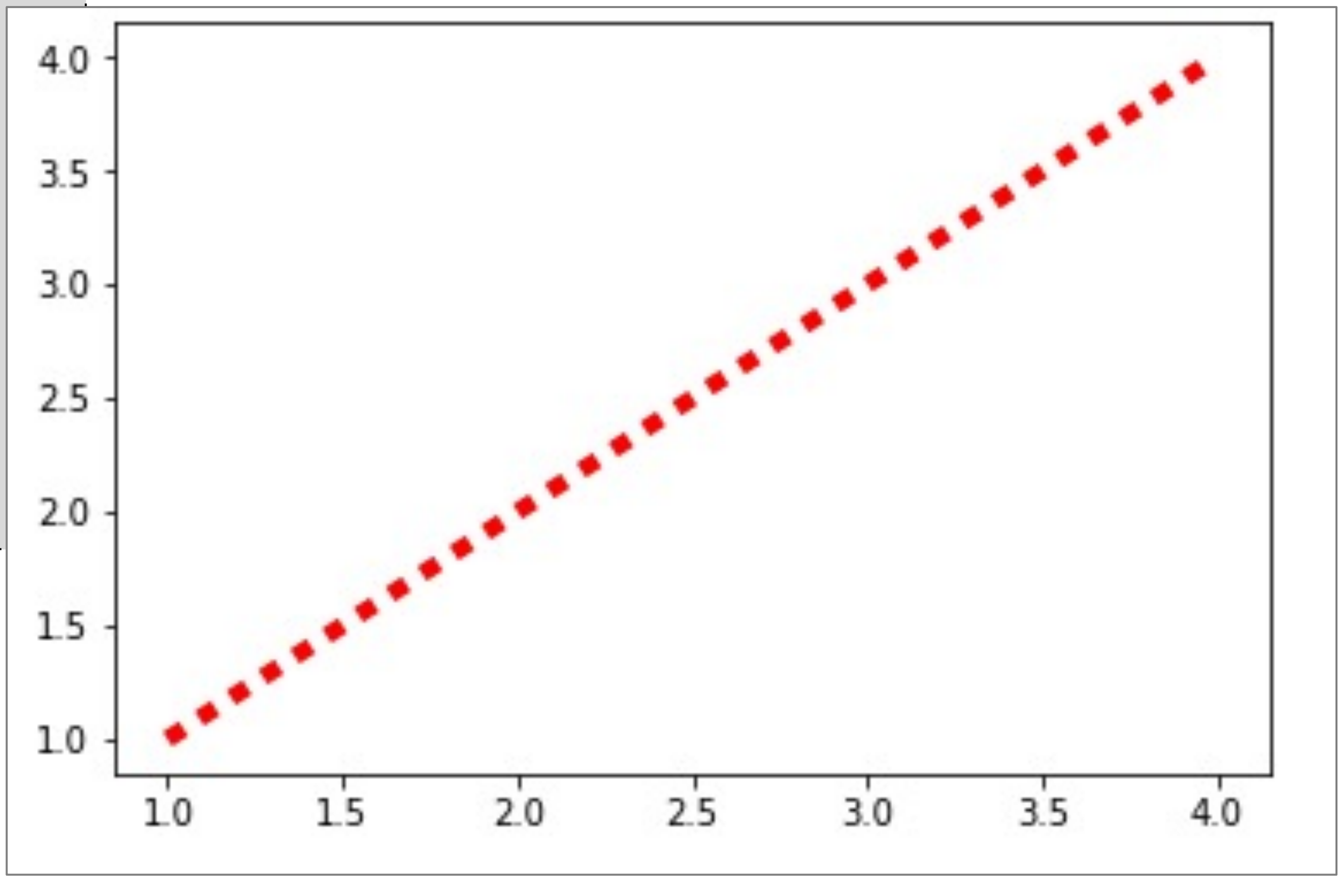
```
import numpy as np
import matplotlib.pyplot as plt

y = [70,30,40,90,10]
xlabels = ('Tom', 'Dick', 'Harry',
           'Sally', 'Sue')
plt.xticks(np.arange(5),xlabels)
plt.ylim(0,100)
plt.plot(y)

plt.show()
```



Controlling Line Properties

Using keyword args with plot method

```
import matplotlib.pyplot as plt
x,y = [1,2,3,4],[1,2,3,4]
plt.plot(x, y,
         linewidth=5,
         linestyle="dotted",
         color="red")

plt.show()
```



- Lines have many attributes that you can set: linewidth, dash style, antialiased
- One easy way to control the line attributes is to add them as keyword args when you call the plot method

Using subplots method

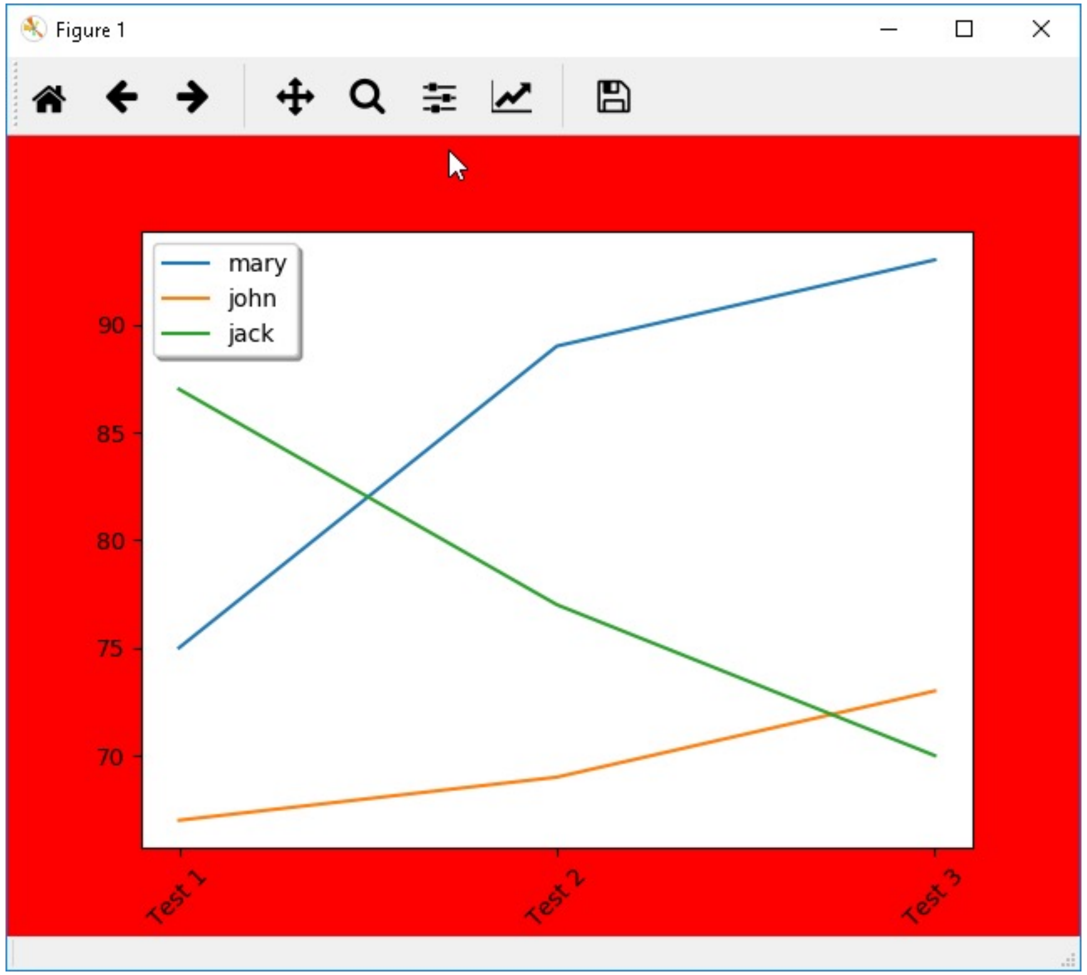
- **plt.subplots()** is a function that returns a tuple containing a **figure** and **axes** object(s)
- Thus when using plt.subplots() you unpack this tuple into the variables **fig** and **ax**

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

- Having a **fig** object is useful if you want to change figure-level attributes or save the figure as an image file later (e.g. with fig.savefig('yourfilename.png')
- Having an **axes** object is useful if you want to change axis-level attributes (e.g. customize ticks, pan/zoom axis etc)

Using fig and ax objects

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

x_labels = ["Test 1", "Test 2", "Test 3"]
mary = [75,89,93]
john = [67,69,73]
jack = [87,77,70]

fig,ax = plt.subplots()
fig.set_facecolor("red")

ax.set_xticks(np.arange(3))
ax.set_xticklabels(x_labels, rotation=45)

ax.plot(mary,label="mary")
ax.plot(john,label="john")
ax.plot(jack,label="jack")

legend = ax.legend(loc='upper left', shadow=True)

plt.show()
```

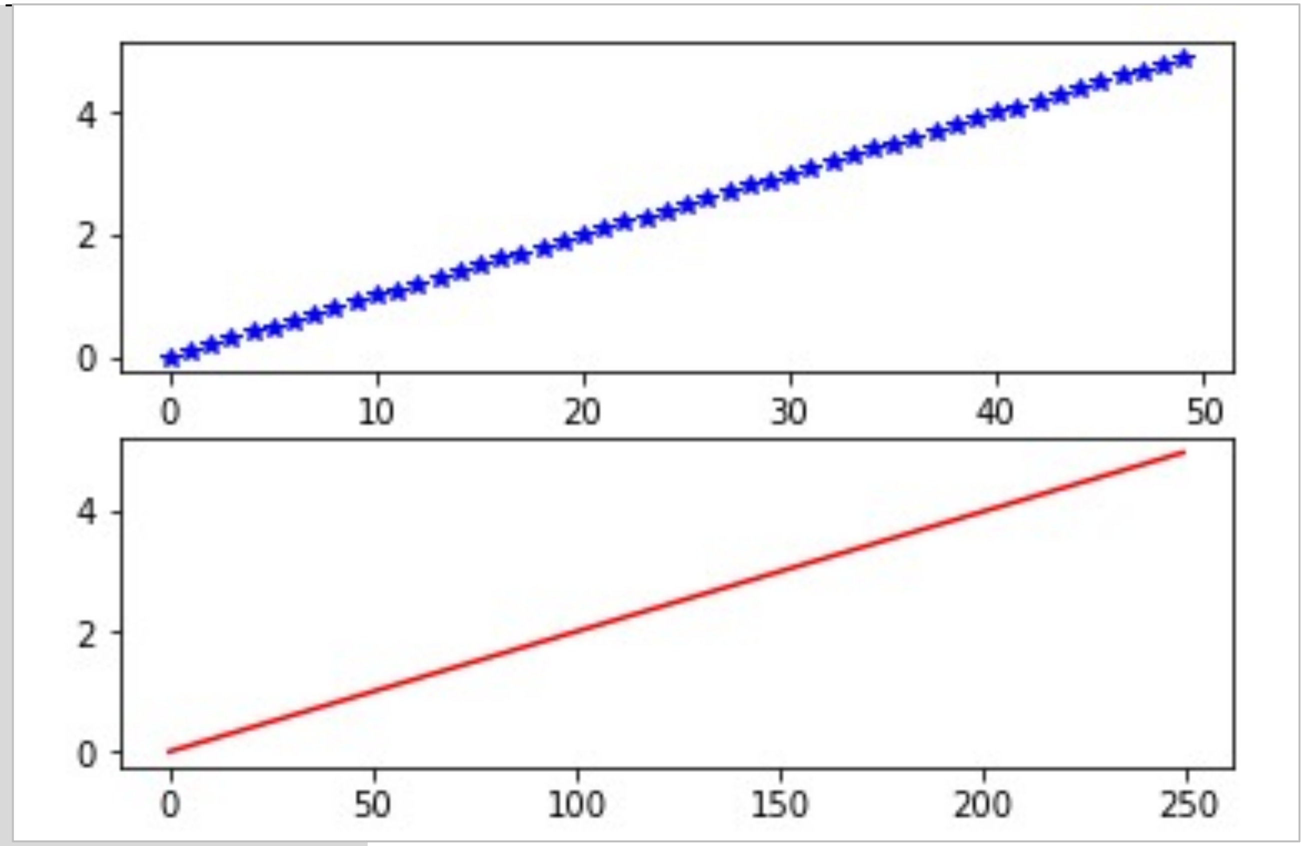


Plotting multiple plots or multiple figures

- Matplotlib allows you to plot multiple plots on a single figure or plot multiple plots on multiple figures using the subplot and figure commands
- The examples that follow in this section illustrate how you can plot multiple subplots on a single figure, as well as plot them on different figures, which allows you to save them separately as individual image files

Two subplots on 1 figure (Example 1)

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

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure(1)
plt.subplot(211)
plt.plot(t1,'b*')

plt.subplot(212)
plt.plot(t2,'r-')
plt.show()
```



- Create a multi-plot by first declaring the figure with a unique reference number, then the subplot method with a unique reference number, followed by the plot method
- subplot(nrows, ncols, plot_number)

How to change size of figure

- plt.figure(figsize=(12,6))
- plt.show()

Working with Text

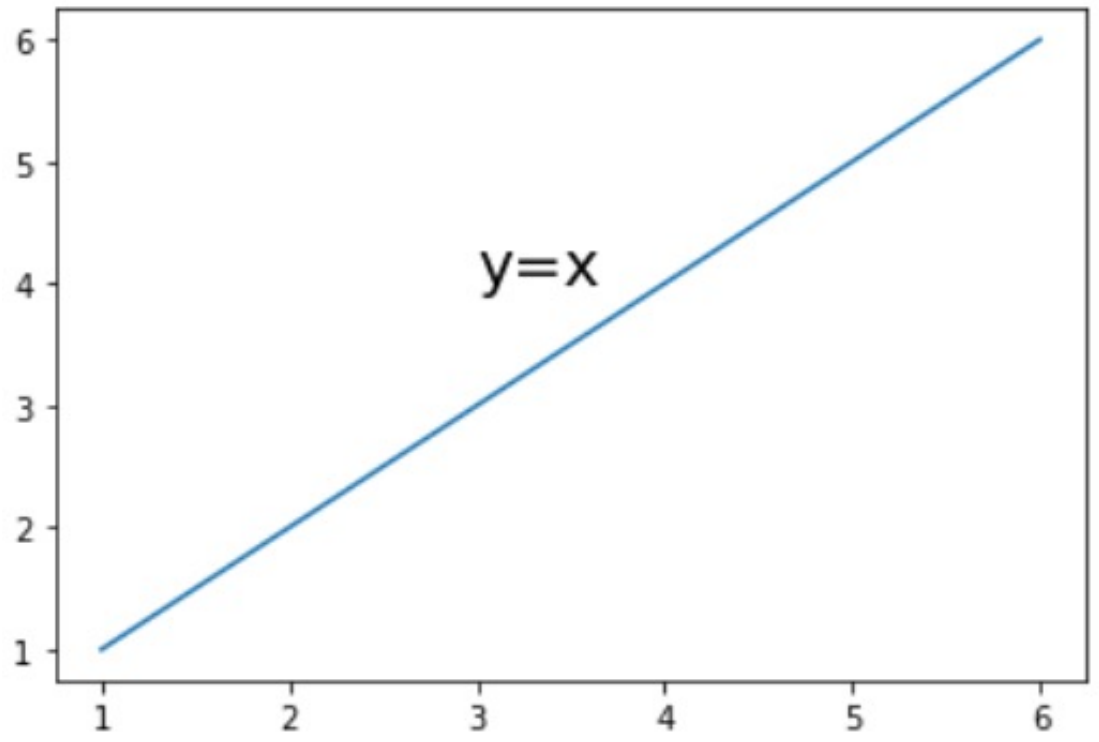
Working with Text

- The following commands are used to create text in the pyplot interface
- The text() command can be used to add text in an arbitrary location
- xlabel(), ylabel() and title() are used to label the axes
- All of these functions create and return a matplotlib.text.Text() instance, which can be configured with a variety of font and other properties

Method	Description
text	add text at an arbitrary location to the Axes
xlabel	add a label to the x-axis
ylabel	add a label to the y-axis
title	add a title to the Axes
figtext	add text at an arbitrary location to the Figure
subtitle	add a title to the Figure
annotate	add an annotation, with optional arrow, to the Axes

Add text at a location

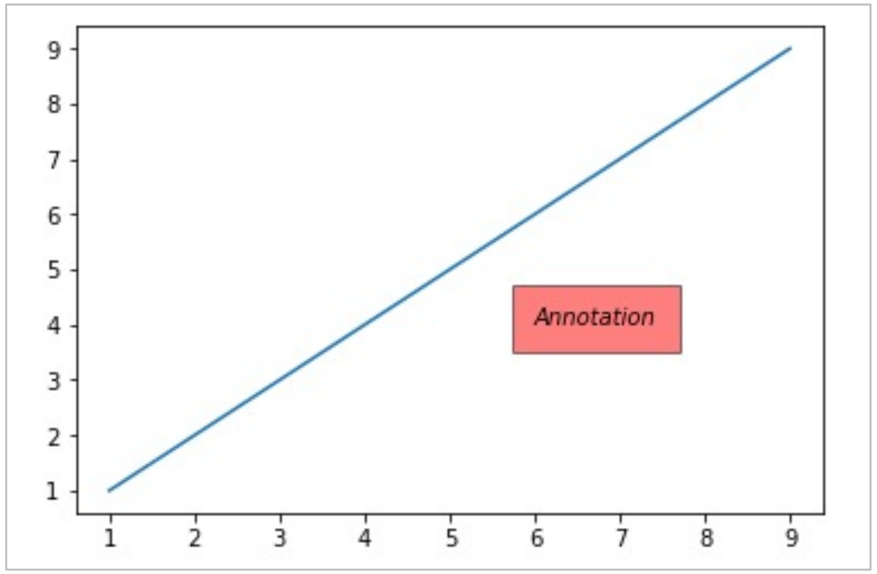
```
import matplotlib.pyplot as plt


plt.plot([1,2,3,4,5,6], [1,2,3,4,5,6])

plt.text(3, 4, 'y=x', fontsize=20)

plt.show()
```



The text() command places text at an arbitrary position

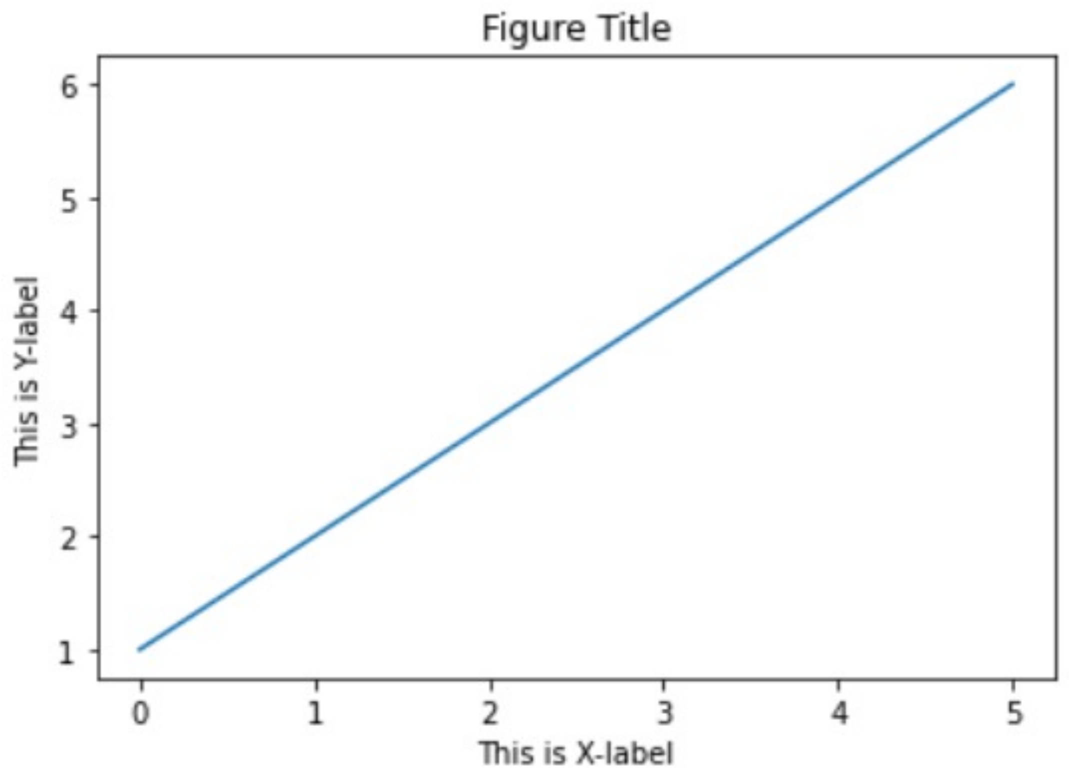
Annotating with Text with Box

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

x1 = np.arange(1,10,2)
y1 = np.arange(1,10,2)

plt.plot(x1,y1)
plt.text(6, 4, 'Annotation', style='italic',
        bbox={'facecolor':'red', 'alpha':0.5,
'pad':10})

plt.show()
```



Add labels and titles

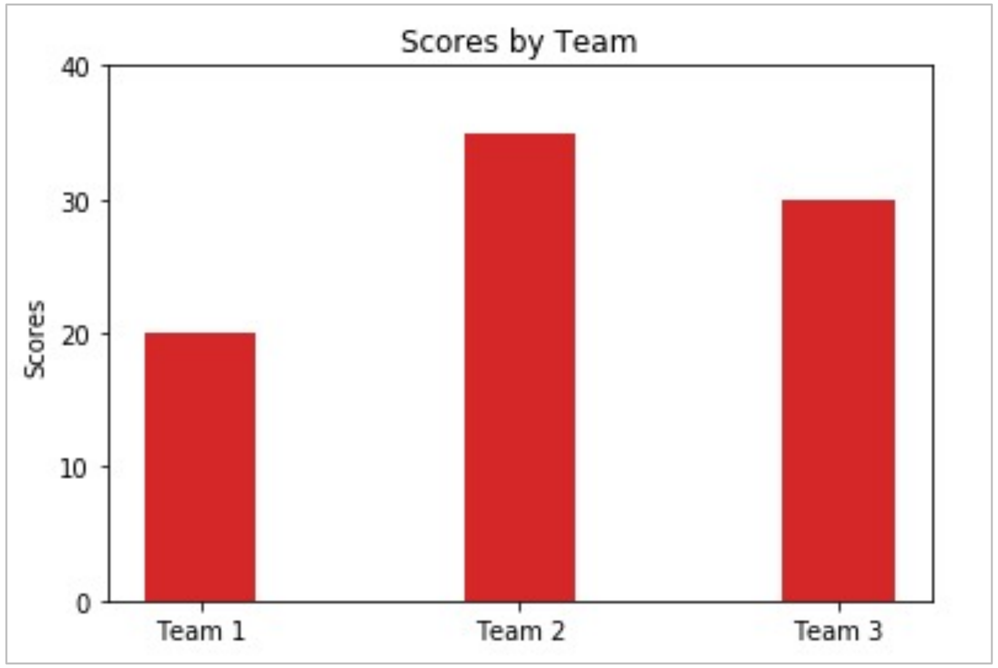
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4,5,6])
plt.title('Figure Title')
plt.xlabel('This is X-label')
plt.ylabel('This is Y-label')
plt.show()
```



Bar charts

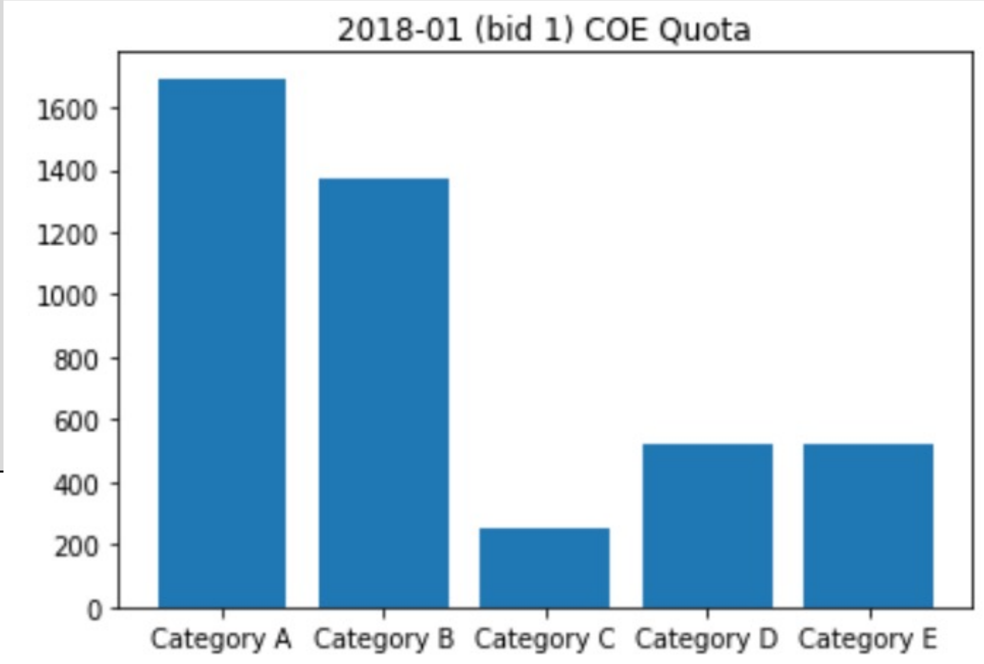
Bar Charts (Example #1)

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

teams = np.arange(3)
scores = (20, 35, 30)
width = 0.35
plt.bar(teams, scores, width,
             color='#d62728')

plt.ylabel('Scores')
plt.title('Scores by Team')

plt.xticks(teams,
           ('Team 1', 'Team 2','Team 3'))
plt.yticks(np.arange(0, 50, 10))

plt.show()
```



Bar Charts (Example #2)

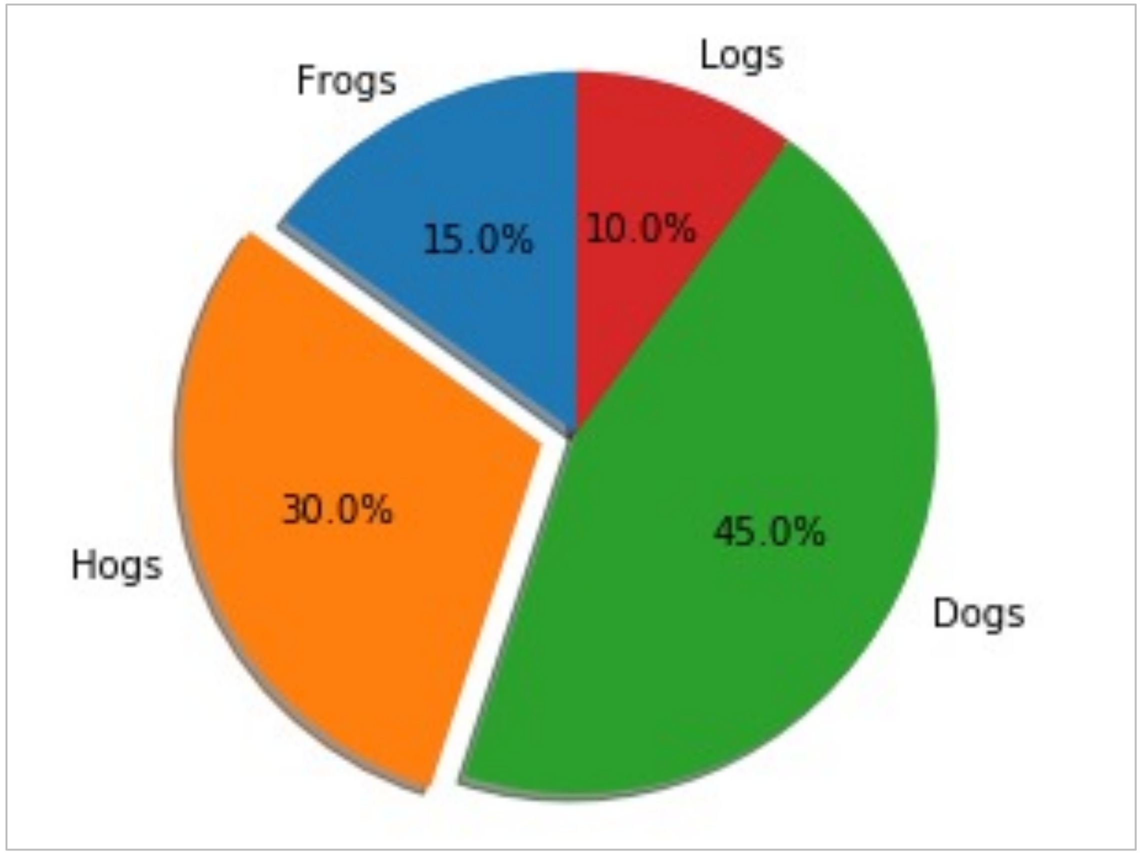
```
import numpy as np
import matplotlib.pyplot as plt

data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
            names=True, dtype=('U7',int,'U10',int,int,int,int))

data2018 = data[(data['month']=='2018-01')]
data2018_1 = data2018[data2018['bidding_no']==1]

plt.title('2018-01 (bid 1) COE Quota')
plt.bar(data2018_1['vehicle_class'],
        data2018_1['quota'])

plt.show()
```



Pie charts

Pie Charts (Example #1)

```
import matplotlib.pyplot as plt

# Slices are ordered, anti-clockwise
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0) # Explode 2nd

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode,
labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)

# Eq aspect ratio draws pie as circle
ax1.axis('equal')
plt.show()
```

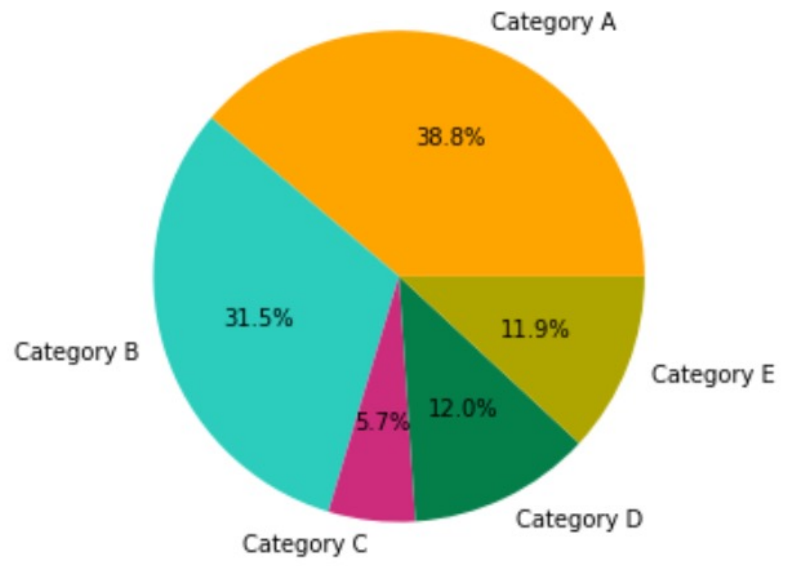


Pie Charts (Example #2)

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

data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
            names=True, dtype=('U7',int,'U10',int,int,int,int))

data2018_01 = data[(data['month']=='2018-01')]
data2018_01_1 = data2018_01[data2018_01['bidding_no']==1]

# Extract rows containing required keywords only
data2018_01_1 = data2018_01_1[np.isin(data2018_01_1['vehicle_class'],
['Category A', 'Category B', 'Category C', 'Category D','Category E'])]
```

Pie Charts (Example #2)

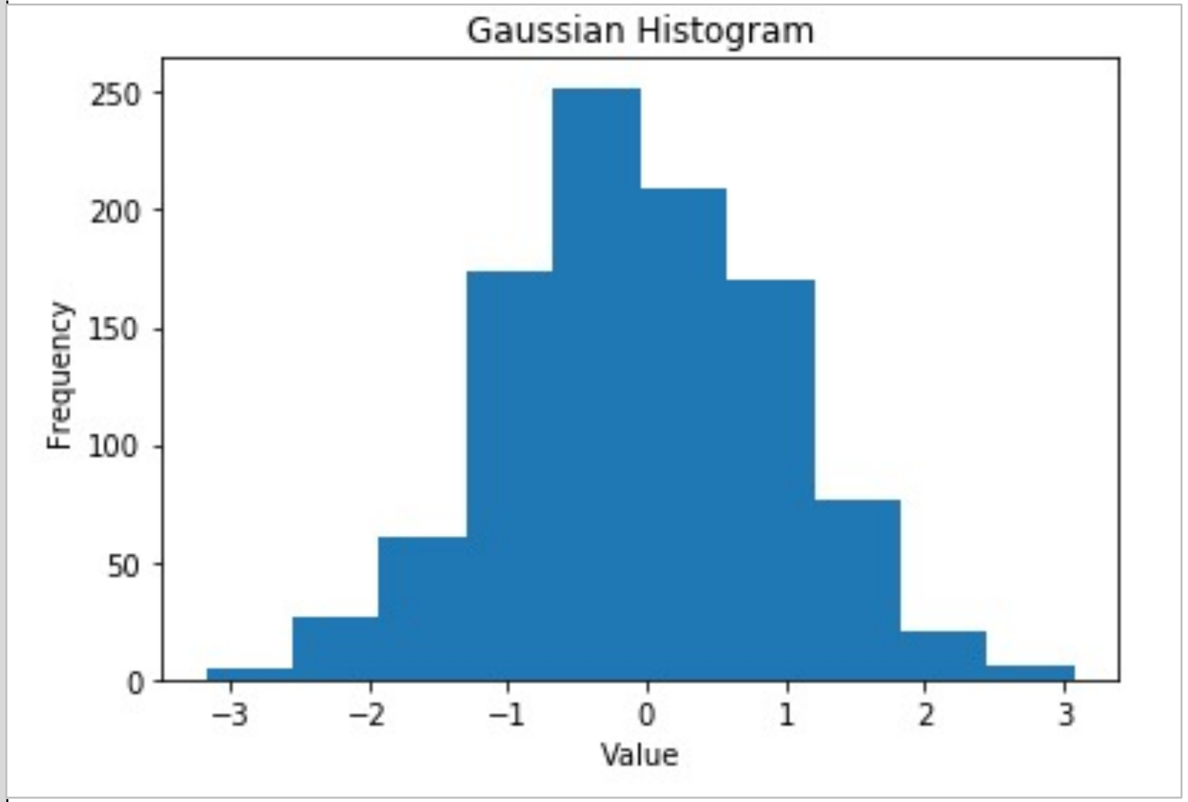
```
labels =
data2018_01_1['vehicle_class']
values = data2018_01_1['quota']

colors = ['#FFA500', '#2CCDBD',
'#CD2C7D', '#057F4A','#AFA500']

explode = (0.1, 0, 0, 0)
plt.figure(figsize=(5,5))
plt.pie(values, labels=labels,
colors=colors, autopct='%1.1f%%')

plt.show()
```



Histograms

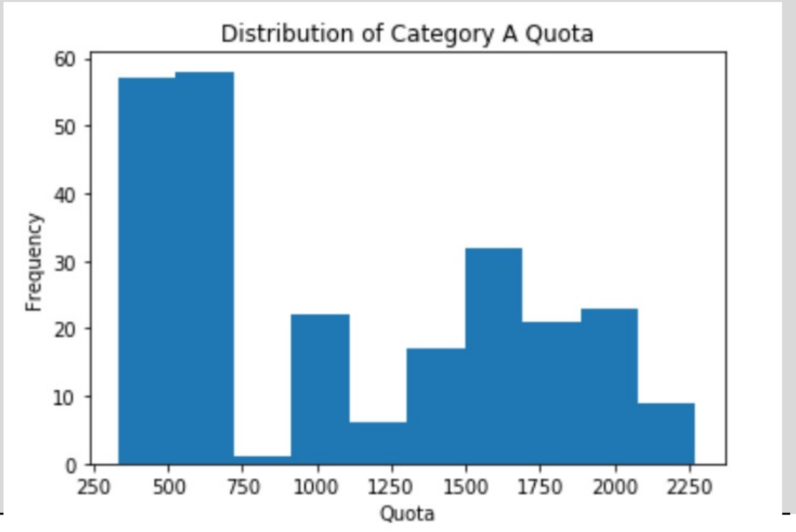
Example #1 - Single dataset

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

gaussian_numbers = np.random.randn(1000)
plt.hist(gaussian_numbers)

plt.title("Gaussian Histogram")
plt.xlabel("Value")
plt.ylabel("Frequency")

plt.show()
```



Example #2

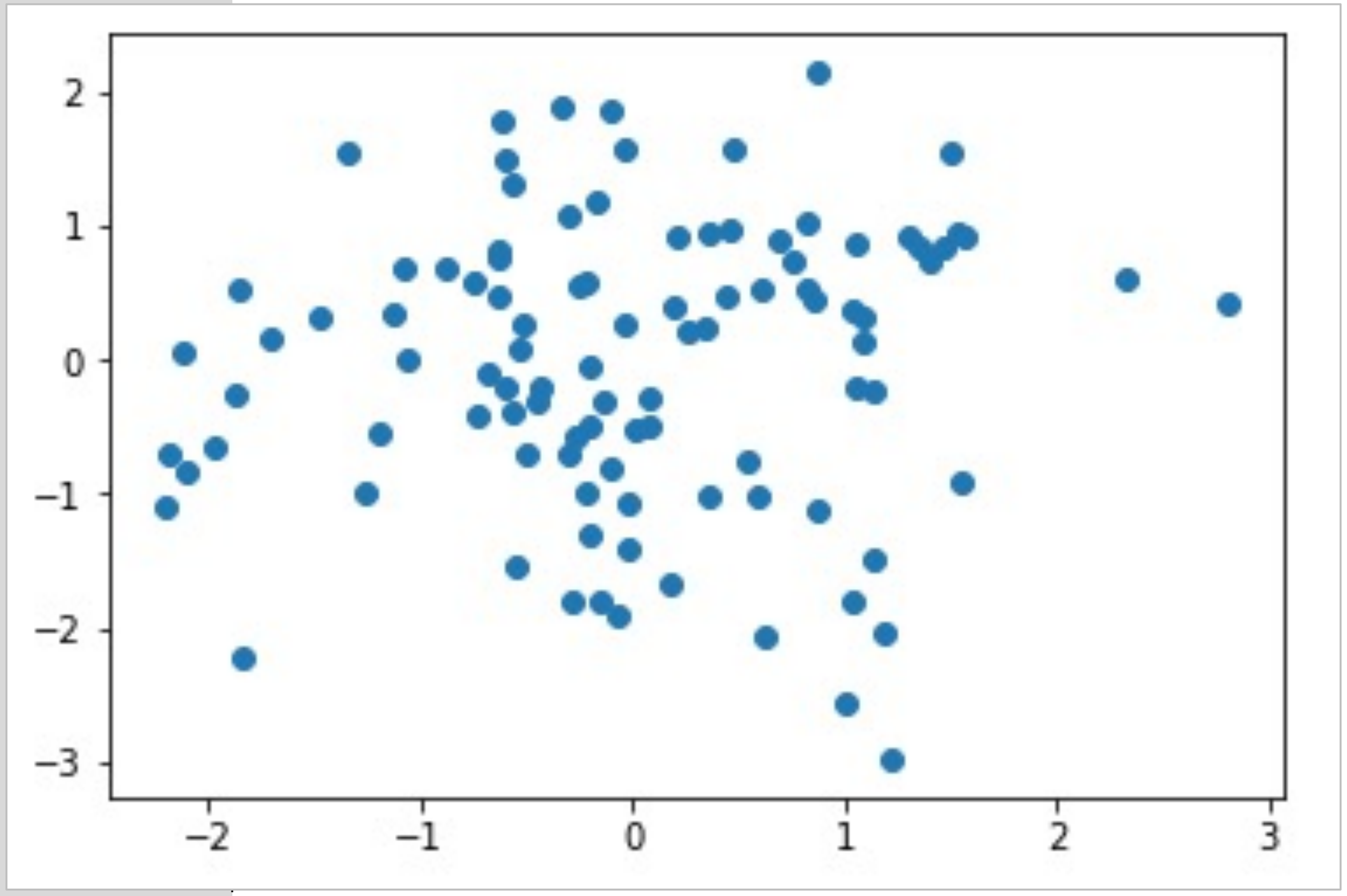
```
import numpy as np
import matplotlib.pyplot as plt

#read in the dataset
data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
            names=True, dtype=('U7',int,'U10',int,int,int,int))

#set up the values
catA = data[data['vehicle_class'] == 'Category A']['quota']

#label the plot
plt.title('Distribution of Category A Quota')
plt.xlabel('Quota')
plt.ylabel('Frequency')

#plot the histogram
plt.hist(catA)
plt.show()
```



Scatter Plots

Scatter Plots (Example #1)

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

x = np.random.randn(100)
y = np.random.randn(100)

plt.scatter(x,y)
plt.show()
```



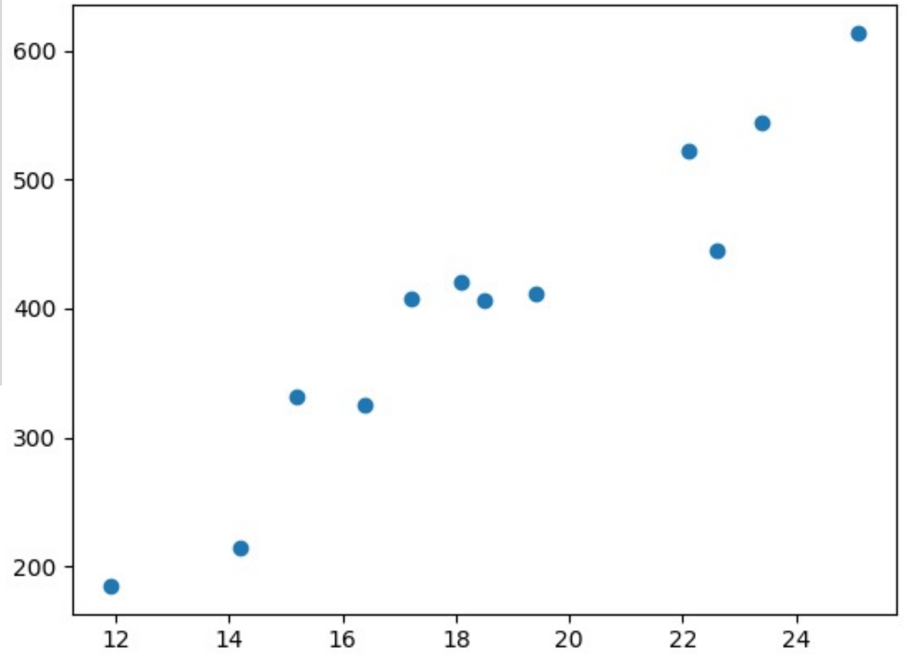
Scatter Plots (Example #2)

- The local ice cream shop keeps track of how much ice cream they sell versus the noon temperature on that day.
- Here are their figures for the last 12 days
- See the next slide for how the same data is plotted as a scatter plot

Temperature	Ice-cream sales
14.2	215
16.4	325
11.9	185
15.2	332
18.5	406
22.1	522
19.4	412
25.1	614
23.4	544
18.1	421
22.6	445
17.2	408

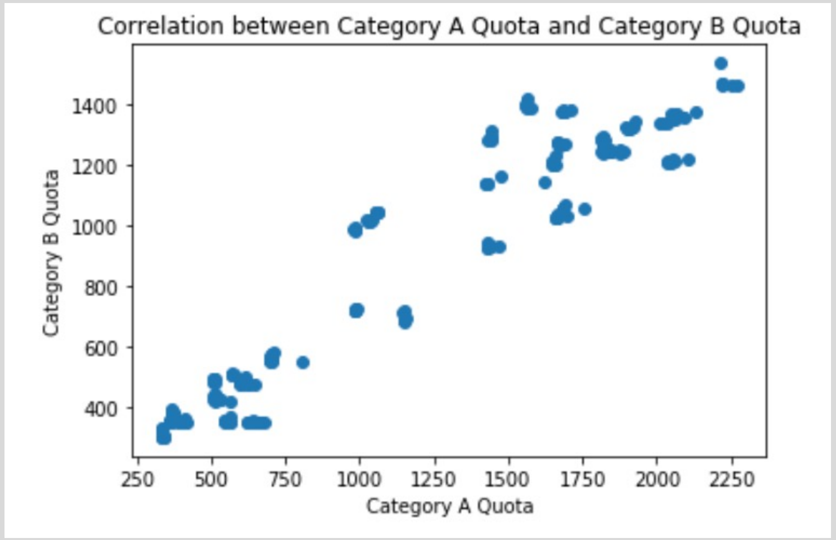
Scatter Plots (Example #2)

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

data = np.loadtxt("icecream.csv",delimiter=",")

# split the data into x and y arrays
x,y = np.split(data, 2, axis=1)

plt.scatter(x,y)
plt.show()
```



Scatter Plots (Example #3)

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

#read in the dataset
data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
            names=True,
            dtype=('U7',int,'U10',int,int,int,int))

#set up the values
catA = data[data['vehicle_class'] == 'Category A']['quota']
catB = data[data['vehicle_class'] == 'Category B']['quota']

#label the plot
plt.title('Correlation between Category A Quota and Category B Quota')
plt.xlabel('Category A Quota')
plt.ylabel('Category B Quota')

#plot the scatter plot
plt.scatter(catA, catB)
plt.show()
```



Box Plots

Box Plots (Example #1)

- This example shows how you can create a box-plot that shows the median of the number of hours 6 students sleep in a week

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

# No.hours 6 students sleep in a week
data = np.array([(8,5,7,3,4,5,9),(6,5,6,8,8,8,6),(4,5,7,6,6,7,9),
               (3,5,7,8,8,7,6),(3,6,7,7,7,6,6),(8,7,6,6,7,7,9)])
labels = np.array(['Sun','Mon', 'Tue','Wed','Thu','Fri','Sat'])

plt.boxplot(data,labels=labels)
plt.show()
```

Box Plots (Example #2)

- This example shows how you can overlay the numeric value of median and outliers in a boxplot

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

#read in the dataset
data = np.genfromtxt("data/coe-results.csv",
            delimiter=',',
            names=True, dtype=('U7',int,'U10',int,int,int,int))

#set up the y values
catA = data[data['vehicle_class'] == 'Category A']['quota']
catB = data[data['vehicle_class'] == 'Category B']['quota']
catC = data[data['vehicle_class'] == 'Category C']['quota']
catD = data[data['vehicle_class'] == 'Category D']['quota']
catE = data[data['vehicle_class'] == 'Category E']['quota']
y_values = np.array([catA, catB, catC, catD, catE])
```

Box Plots (Example #2)

- This example shows how you can overlay the numeric value of median and outliers in a boxplot

```
#set up the labels on the x-axis
x_labels = np.unique(data['vehicle_class'])
x_ticks = np.arange(len(x_labels))
plt.xticks(rotation=90)

plt.boxplot(y_values.transpose(),labels=x_labels, patch_artist=True)
plt.show()
```

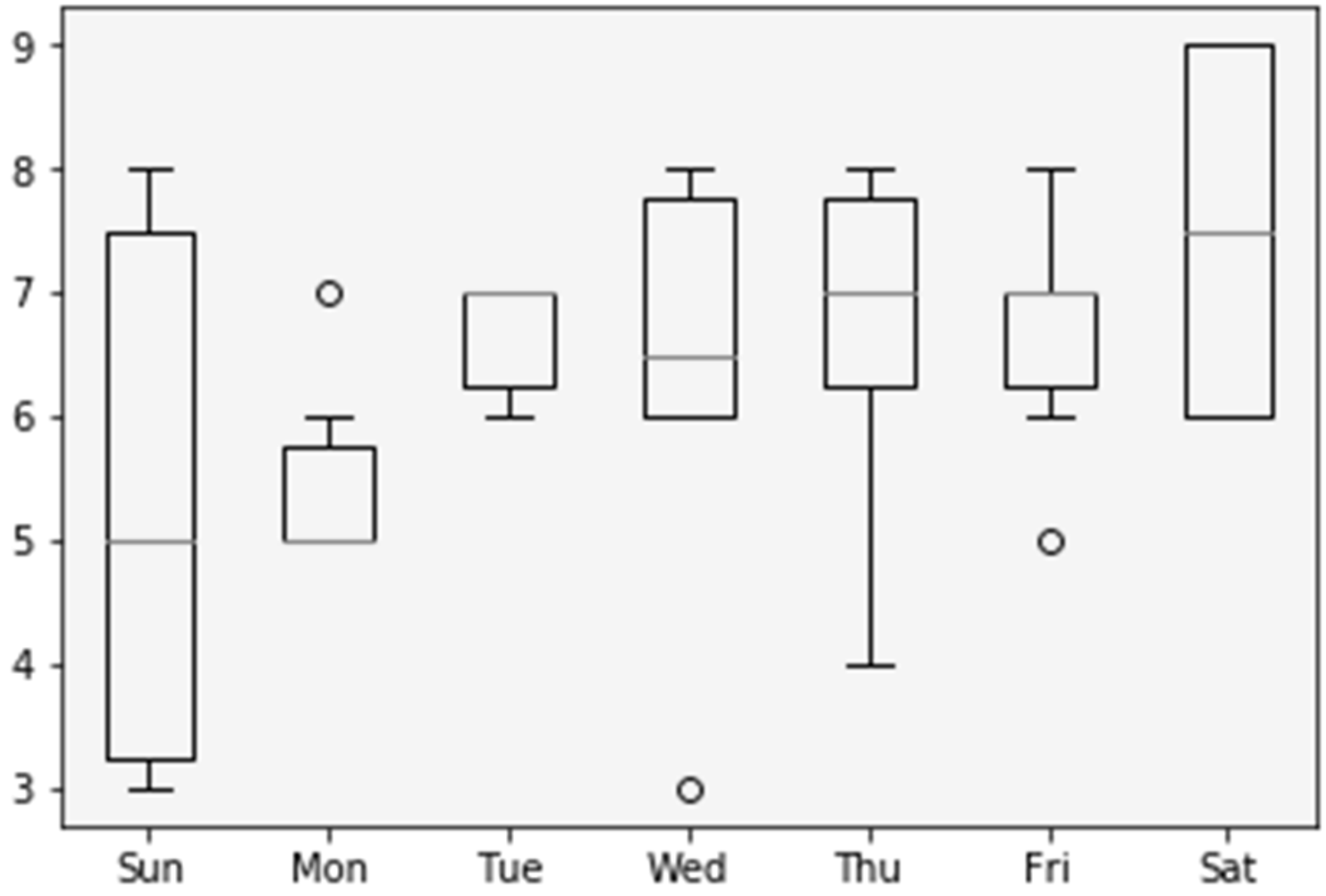
Output from Example 1

From the output, we can tell that most of the students do get more than 6 hours of sleep on most days, and do especially well on Saturdays.

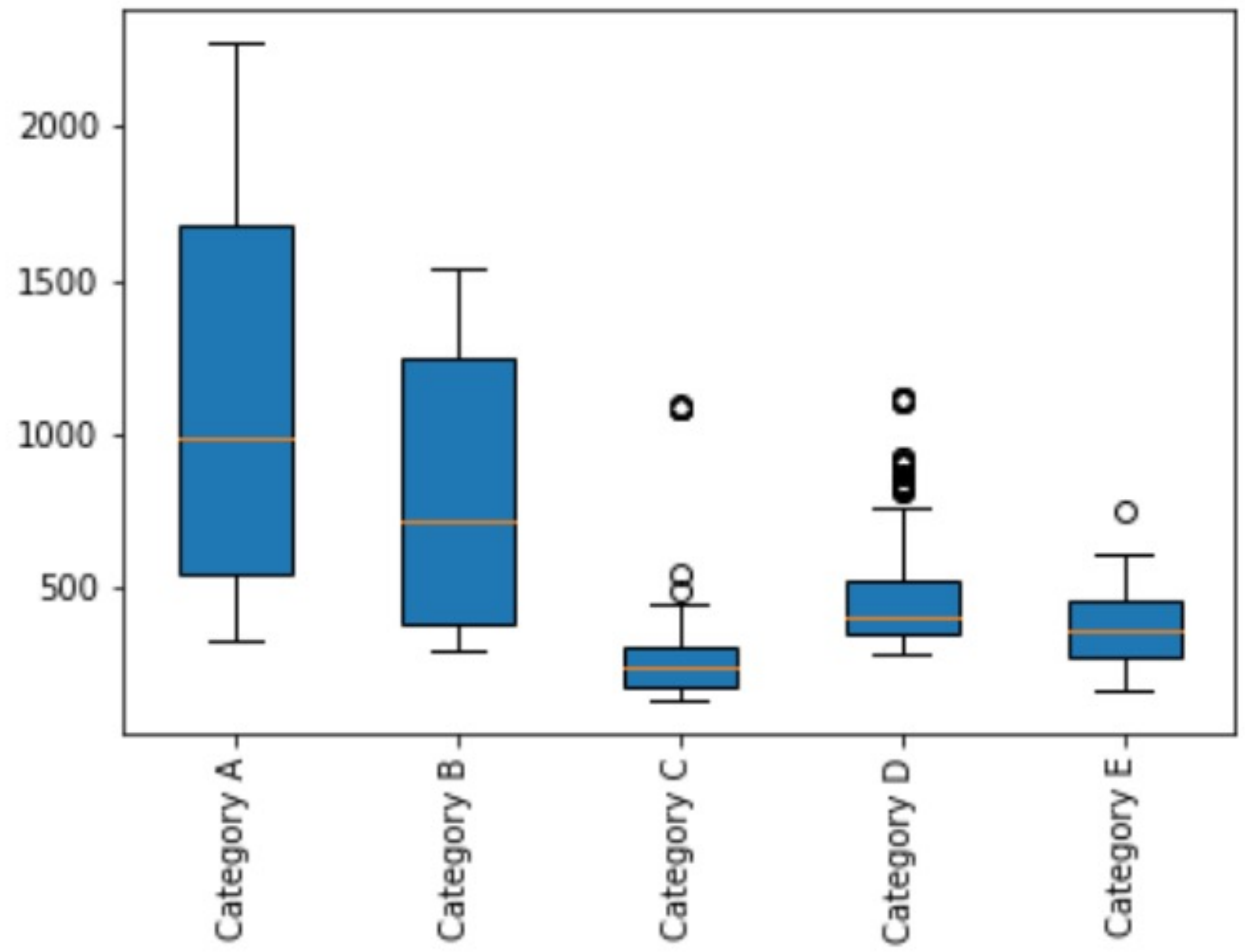
However, there is a huge disparity of sleep patterns on Sunday where some students get much less sleep than others.

Most students sleep very little on Mondays with little difference. The sleeping patterns of Tue, and Fri also have little variation.

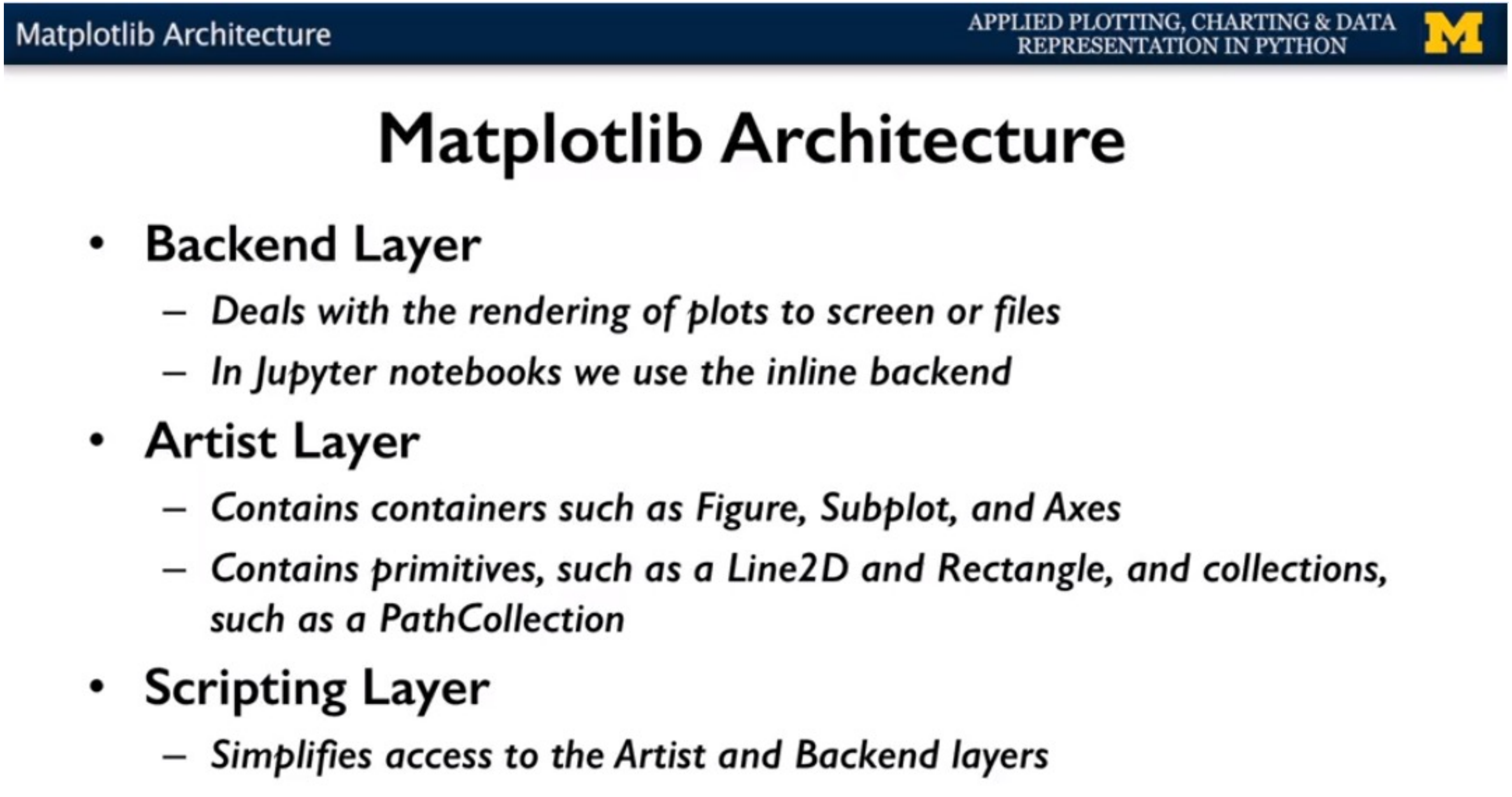
Thu sleep patterns vary by quite a lot though



Output from Example 2



The End



Matplotlib Architecture

APPLIED PLOTTING, CHARTING & DATA REPRESENTATION IN PYTHON

Matplotlib Architecture

- **Backend Layer**
  - *Deals with the rendering of plots to screen or files*
  - *In Jupyter notebooks we use the inline backend*
- **Artist Layer**
  - *Contains containers such as Figure, Subplot, and Axes*
  - *Contains primitives, such as a Line2D and Rectangle, and collections, such as a PathCollection*
- **Scripting Layer**
  - *Simplifies access to the Artist and Backend layers*