CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Programming for Data Analytics

Week8: Visualization

Dr. Haithem Afli

Haithem. afli@cit.ie

@AfliHaithem

2018/2019

Introduction to MatPlotLib

- Matplotlib is a **library** for making **graphs** of arrays in Python.
  - Although matplotlib is written primarily in pure Python, it makes heavy use of NumPy and other extension code to provide good performance even for large arrays.
  - http://matplotlib.org/
- ***matplotlib.pyplot*** is a collection of command style functions.
  - Each pyplot function makes some changes to a figure:
    - Create a figure
    - Create a plotting area in a figure
    - Plot some lines in a plotting area
    - Format the plot with labels
  - http://matplotlib.org/api/pyplot_api.html

Using Pandas with Mat

- Pandas offers visualization capabilities and is tightly integrated with Matplotlib.

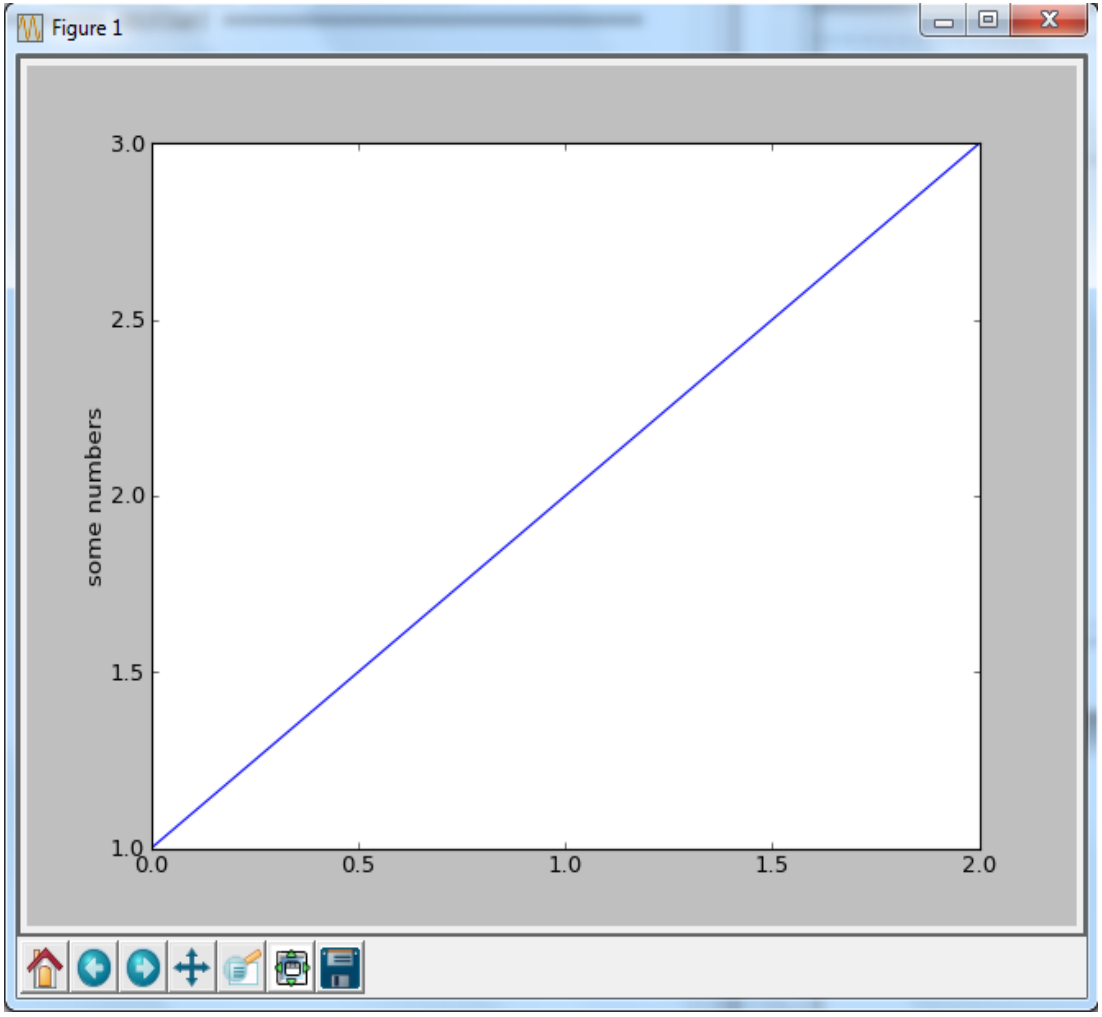
- In the following slides we will show how to generate graphs using both MatPlotLib directly and using Pandas.

Example

```
import matplotlib.pyplot as plt

plt.plot([1,2,3])

plt.show()
```

If the plot method only receives a single list to be plotted then it assumes it is to be plotted on the y axis

The graphical representation is displayed by **show()** function.



Notice the x-axis ranges from 0-2 and the y-axis from 1-3.
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 is [0,1,2].

Plot Command

- plot() is a versatile command, and is capable of accepting an **arbitrary number of arguments**.
- For example, to plot x versus y, you can issue the command:
- **plt.plot([1,2,3,4], [1,4,9,16])**
  - The first list is the x coordinates and the second list are the y coordinates

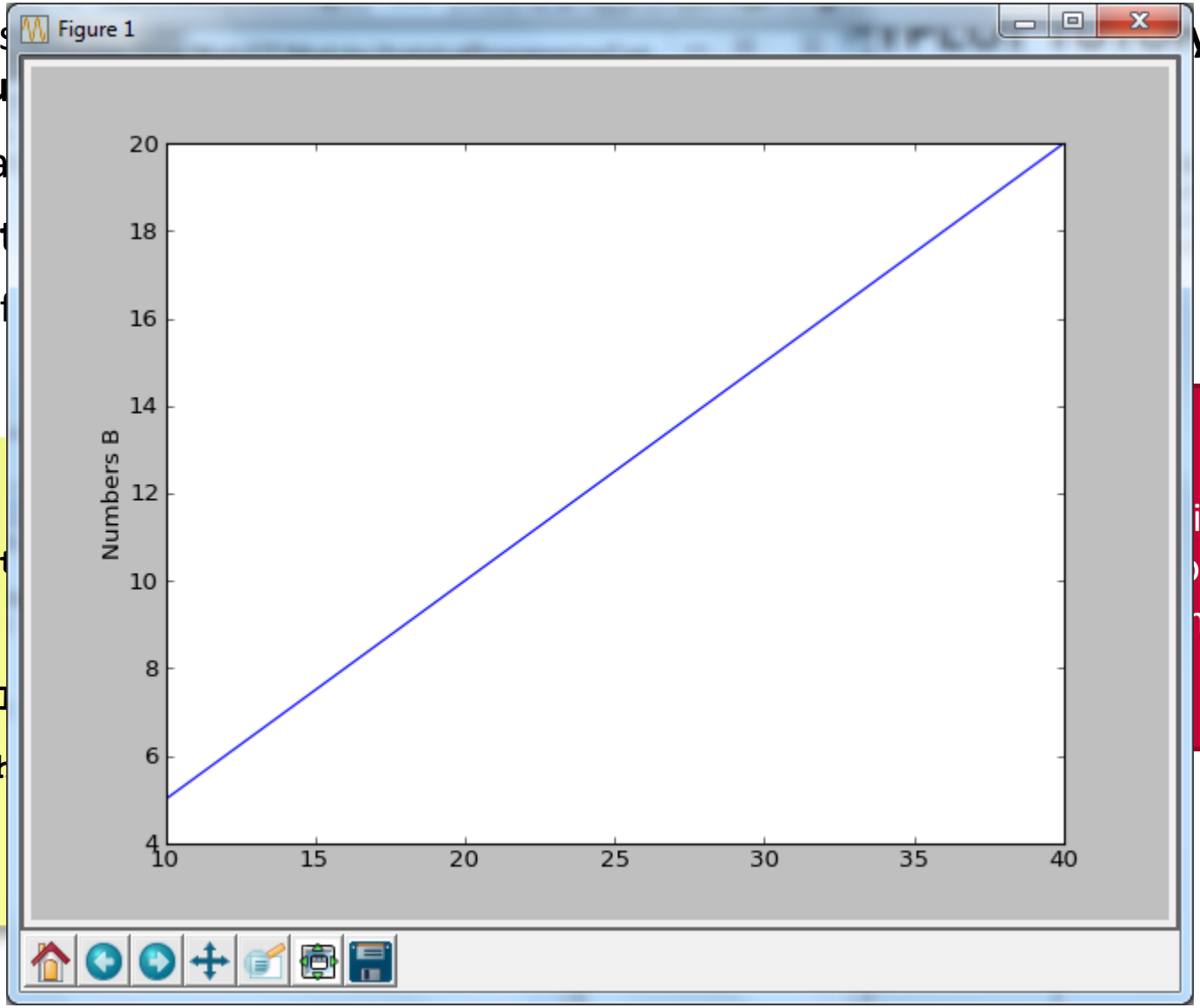
```
import matplotlib.pyplot as plt

plt.plot([10, 20, 30, 40], [5, 10, 15, 20])
plt.show()
```

Plot function uses first list as x coordinates to be mapped

Plot Command

- plot() is **y number of argu**
- For exa
- **plt.plot**
  - The f

```
import

plt.pl
plt.sh
```



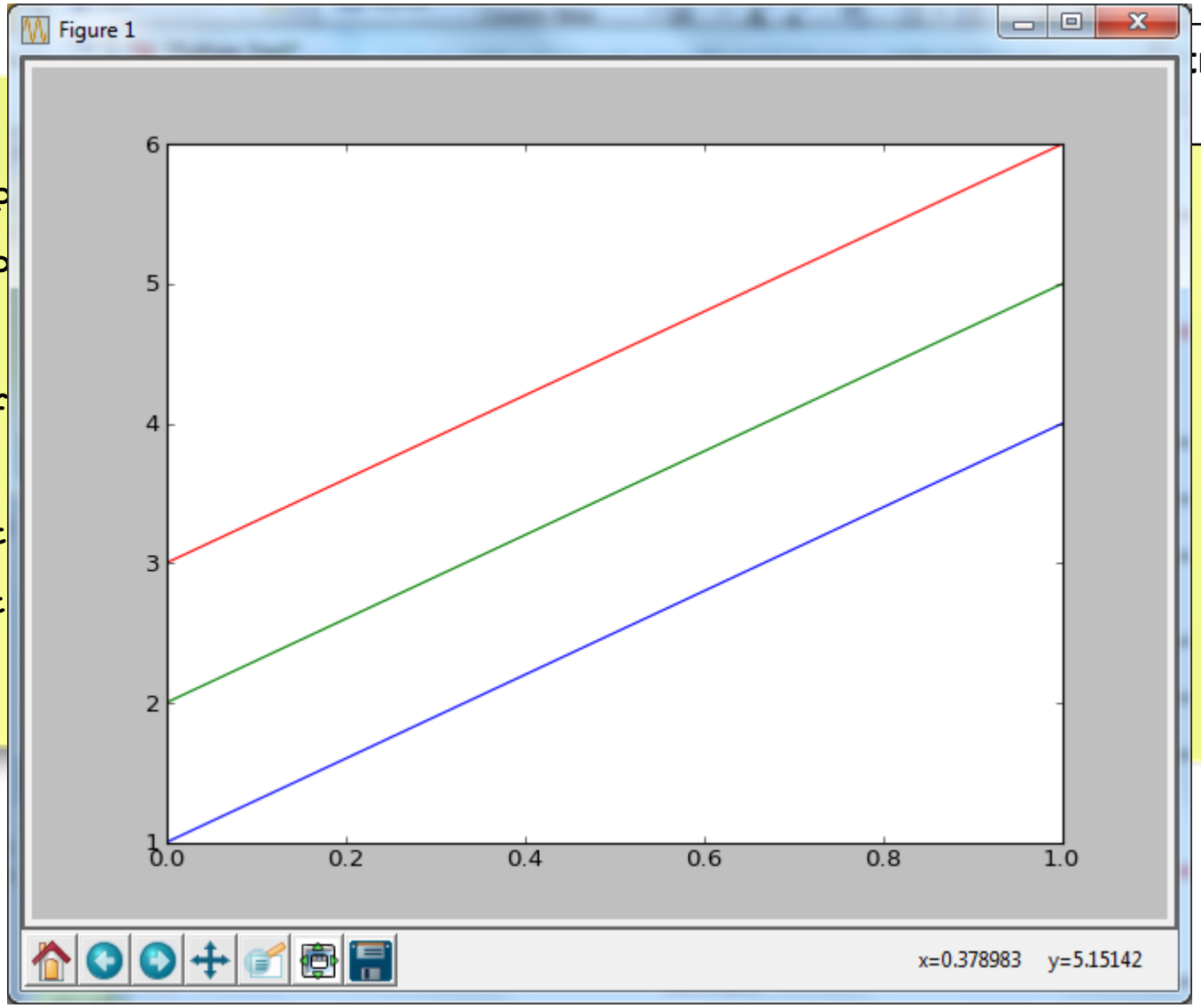
ion uses first
ordinates to
napped

Plot 2D Numpy Arrays

Each **column** in the 2D array is **treated as a separate line**

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

arr = np.array([[1, 2, 3], [4, 5, 6]], float)

plt.plot(arr)
plt.show()
```

Plot 2D Numpy Arrays




```
imp
imp

arr

plt
plt
```

treated as a

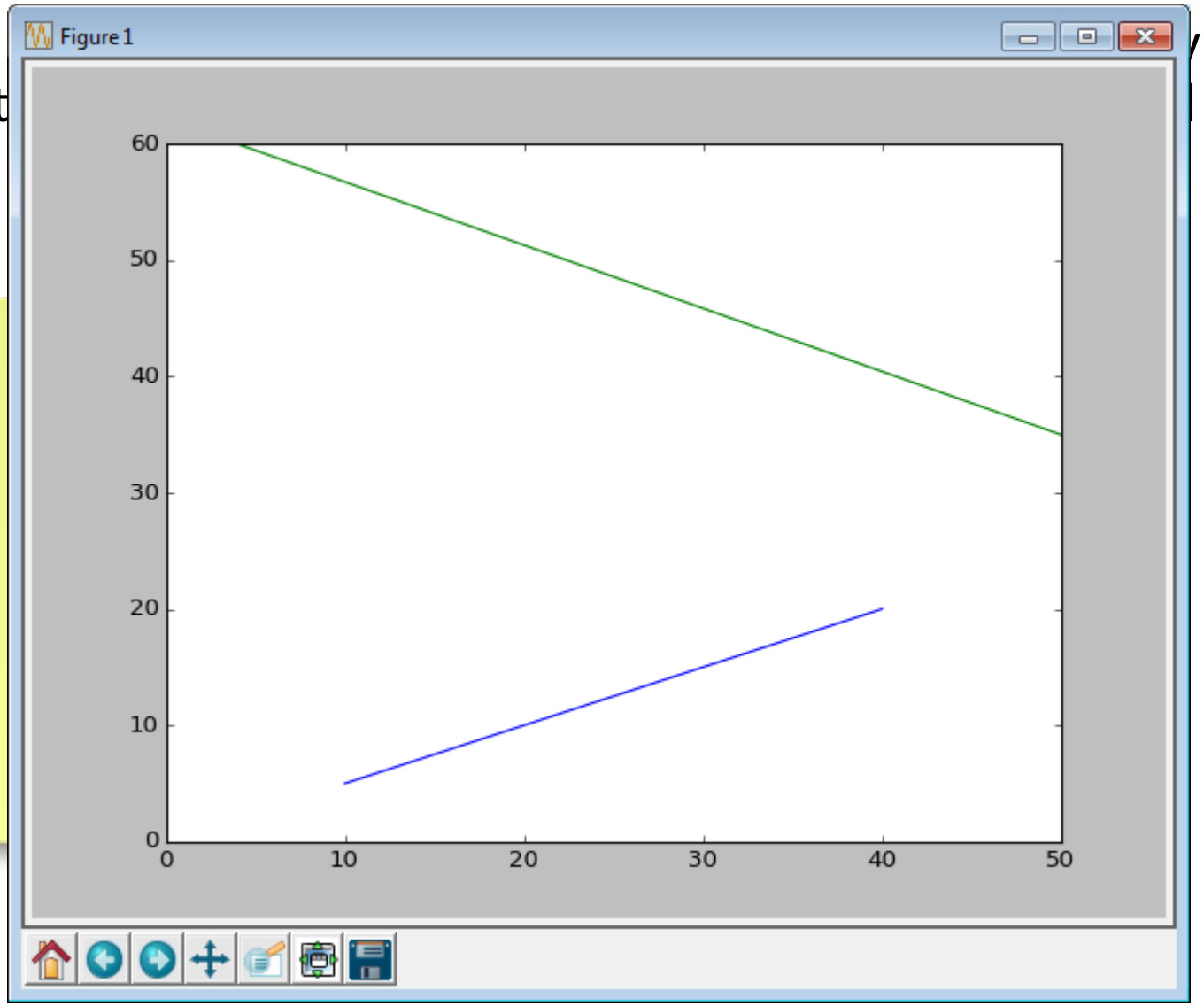
Adding Lines to a Plot

- You can incrementally add additional elements to a plot by using the plot function. Here for example we add a second line to the plot.

```
# plots one line
plt.plot([10, 20, 30, 40], [5, 10, 15, 20])

#plots a second line
plt.plot([4, 50], [60, 35])
plt.show()
```

Adding Lines to a Plot

- You ca
using t
to the



line

Plotting using a Series and DataFrame

- Visualizing a data in pandas is as simple as calling .plot() on a DataFrame or Series object.

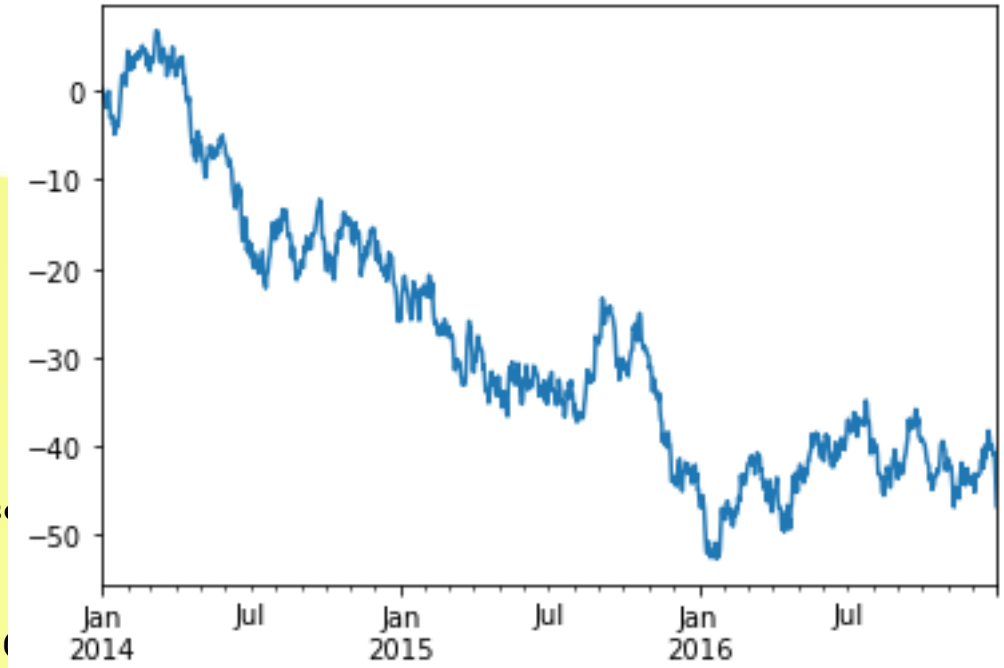
- In the example on the next slide we create a **Series object** that contains some time series data and we then plot it using the plot command.

- Notice, that the:
  - **Index** of the Series becomes the label for the **X axis**
  - **Values** of the Series becomes the **Y values**.

Plotting using a Series

```
import numpy as np
import pandas as pd

# generate a random walk time-series
np.random.seed(19)
s = pd.Series(np.random.randn(1096),
              index=pd.date_range('2014-01-01',
                                  '2016-12-31'))
# the cumsum function will return a Series contains the cumulative sum
# of all values in s
walk_ts = s.cumsum()

walk_ts.plot();
```

Plotting using a DataFrame




```
import numpy as np
import pandas as pd

# generate a random walk time-s
np.random.seed(19)
s = pd.Series(np.random.randn(1
                index=pd.date_range('2014-01-01',
                                    '2016-12-31'))
# the cumsum function will return a Series contains the cumulative sum
# of all values in s
walk_ts = s.cumsum()

walk_ts.plot();
```

Plotting using a DataFrame

- In the code below we create a DataFrame

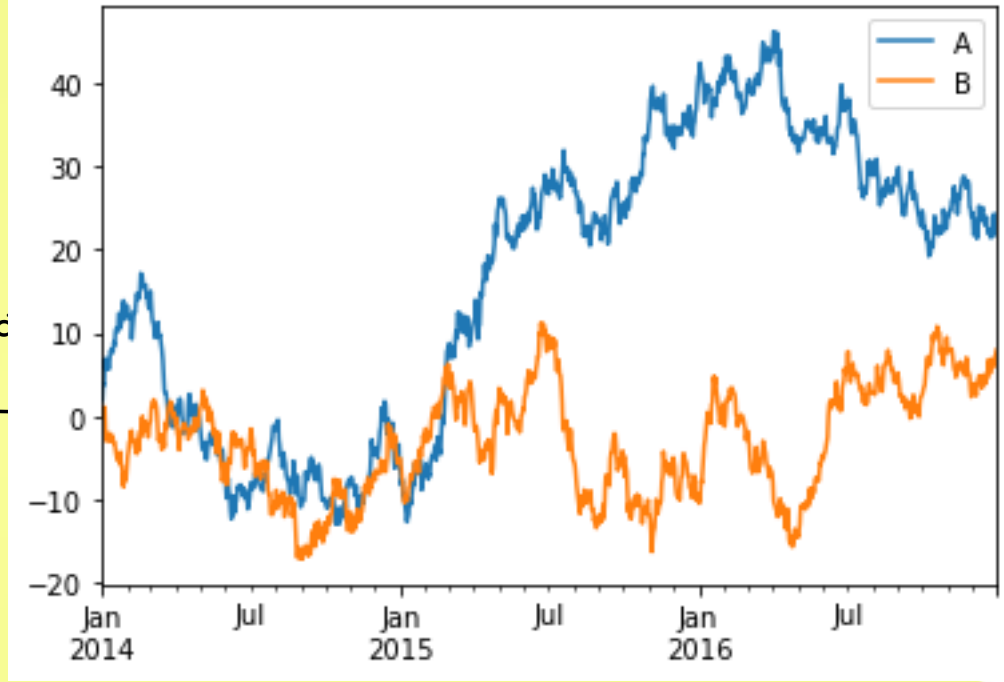
```
df = pd.DataFrame(np.random.randn(1096, 2),
                  index=pd.date_range('2014-01-01',
                                      '2014-16-31'),columns=list('AB'))
walk_df = df.cumsum()

walk_df.head()
```

	A	B
2014-01-01	-0.378286	0.881338
2014-01-02	1.489231	1.170020
2014-01-03	1.590698	0.194133
2014-01-04	1.842982	-0.507354
2014-01-05	2.599857	-1.376844

Plotting using a DataFrame

- In the code below we create a DataFrame

```
df = pd.DataFrame(np.random.rand
                  index=pd.date_

walk_df = df.cumsum()

walk_df.head()

walk_df.plot()
```



Task

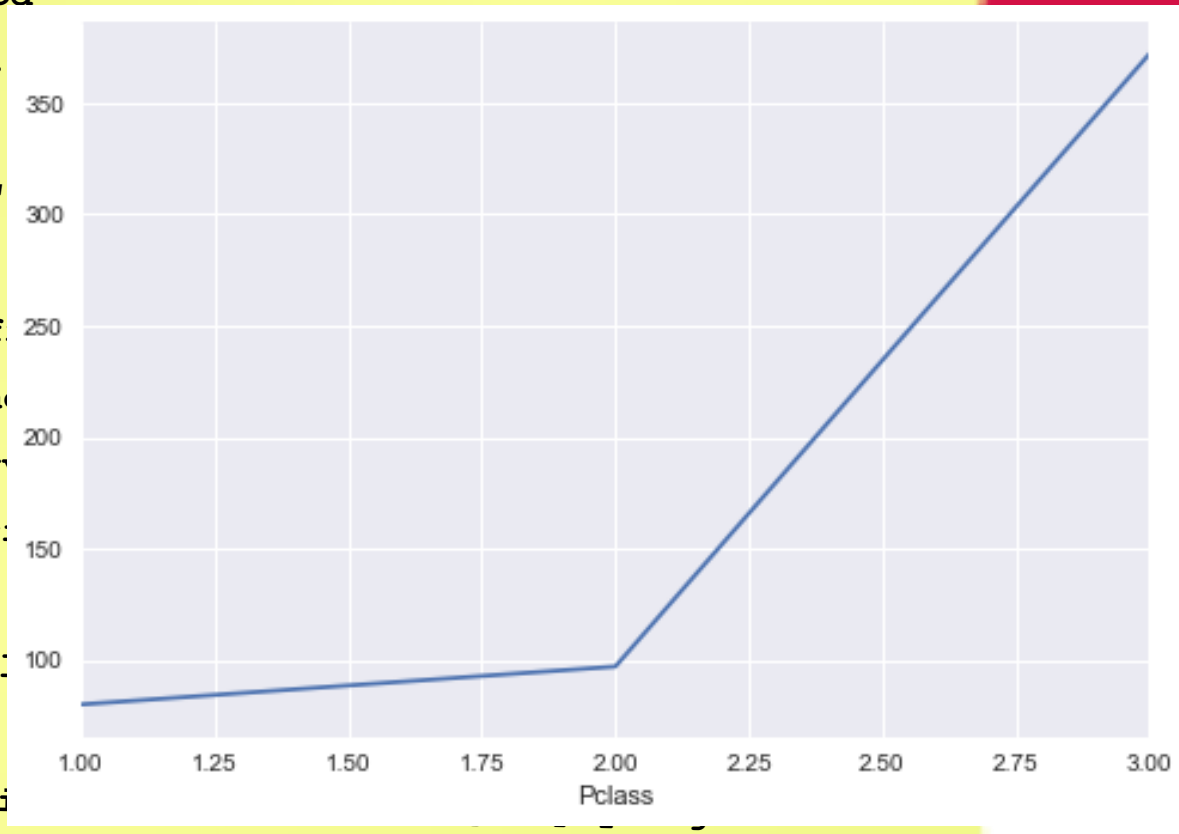
- Returning to our titanic dataset.

- Write a program that will generate a basic graph showing the number of first class, second class and third class passengers that died on the titanic (use groupby functionality in your answer).

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("titanic.csv")

# filter the dataframe to only return rows
# where the passenger died
criteria = df['Survived']==0
fatalities = df[criteria]

pclassGroup = fatalities.groupby("Pclass")

# extract the Survived column from the groupby object
classSurived = pclassGroup['Survived'].count()
print (classSurived)

classSurived.plot()
plt.show()
```

```
Pclass
1     80
2     97
3    372
Name: Survived, dtype: int64
```

```
import pandas as pd
import matplotlib.

df = pd.read_csv("

# filter the dataf
# where the passen
criteria = df['Sur
fatalities = df[cr

pclassGroup = fata

# extract the Survi
classSurived = pclassGroup['Survived'].count()
print (classSurived)

classSurived.plot()
plt.show()
```




```
Pclass
1      80
2      97
3     372
Name: Survived, dtype: int64
```


CIT

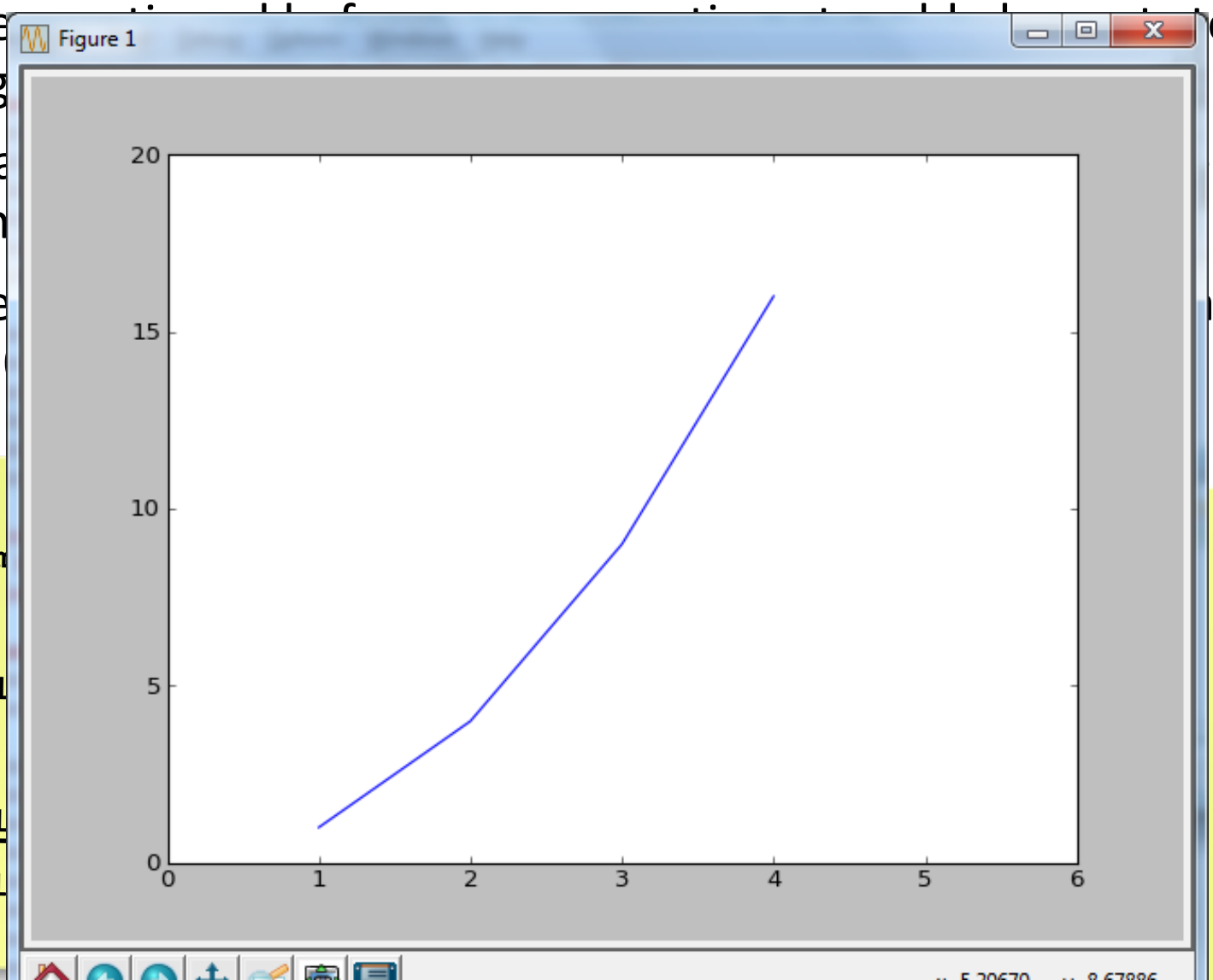
Plot Axis Command

- As we mentioned before we can continue to add elements to the plt figure object and alter aspects of the figure.
- We can use the **axis** method to **set the viewable area** of the graph.
- In the case below we fix it so that the x coordinates shown are from 0 to 6 and the y coordinates from 0 to 20.

```
import matplotlib.pyplot as plt


plt.plot([1,2,3,4], [1,4,9,16])
# Sets viewable area to 0-6 on x axis and 0-20 on y axis
plt.axis([0, 6, 0, 20])
plt.show()
```

Plot Axis Command

- As we
- We ca
- In the



Using the Axis Command with a DataFrame

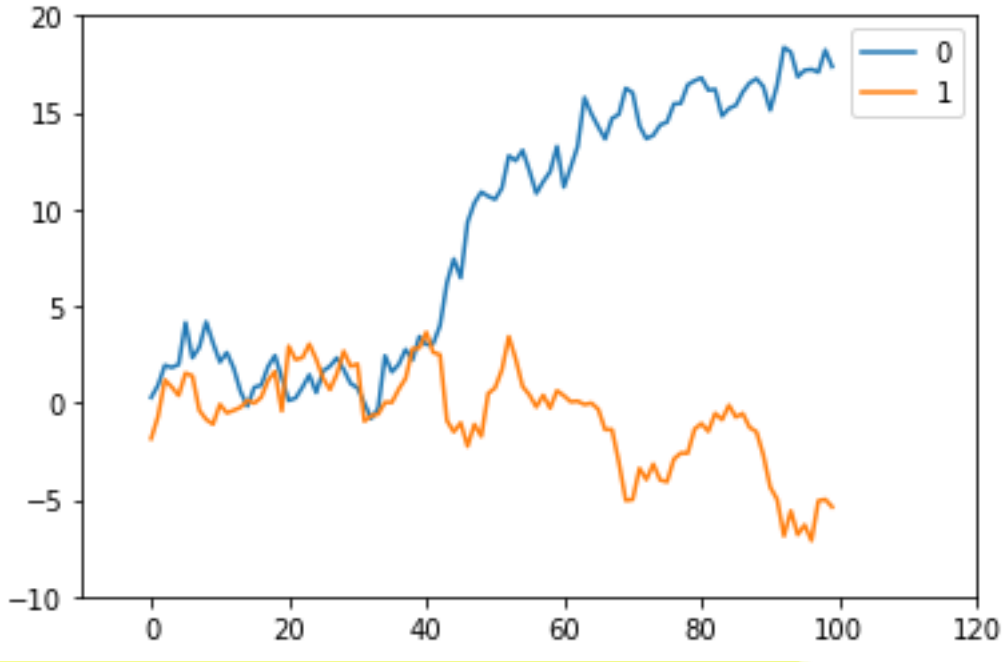
- The DataFrame object is using MatPlotLib when it generates a graph.
- Therefore, the functions we use to alter the plot in MatPlotLib can also be used in conjunction with Pandas.

```
import matplotlib.pyplot as plt
import pandas as pd


np.random.seed(17)
df = pd.DataFrame(np.random.randn(100, 2))


df= df.cumsum()
df.plot()
plt.axis([-10, 120, -10, 20])
plt.show()
```

Using the Axis Command with a DataFrame

- The DataFrame object is using MatPlotLib when it generates a graph.
- Therefore, the functions we use to alter the plot in MatPlotLib can also be used in conjunction with Pandas.

```
import matplotlib.pyplot a
import pandas as pd


np.random.seed(17)
df = pd.DataFrame(np.rando


df= df.cumsum()
df.plot()
plt.axis([-10, 120, -10, 20])
plt.show()
```




CIT

Plot Command and Formatting

- Notice that all information we have plotted to a graph so far has been depicted as lines

- This section will look at adjusting many of the properties of these lines, size, colour, format (dashed)

- plot is a versatile command
  - We can add multiple pairs of x, y coordinates
  - For **every x, y pair** of arguments, there is an **optional third argument** which is the format string that indicates the **colour and line type** of the plot.

  - The letters and symbols of the format string are from MATLAB, and you concatenate a **color** string with a **line style string**.

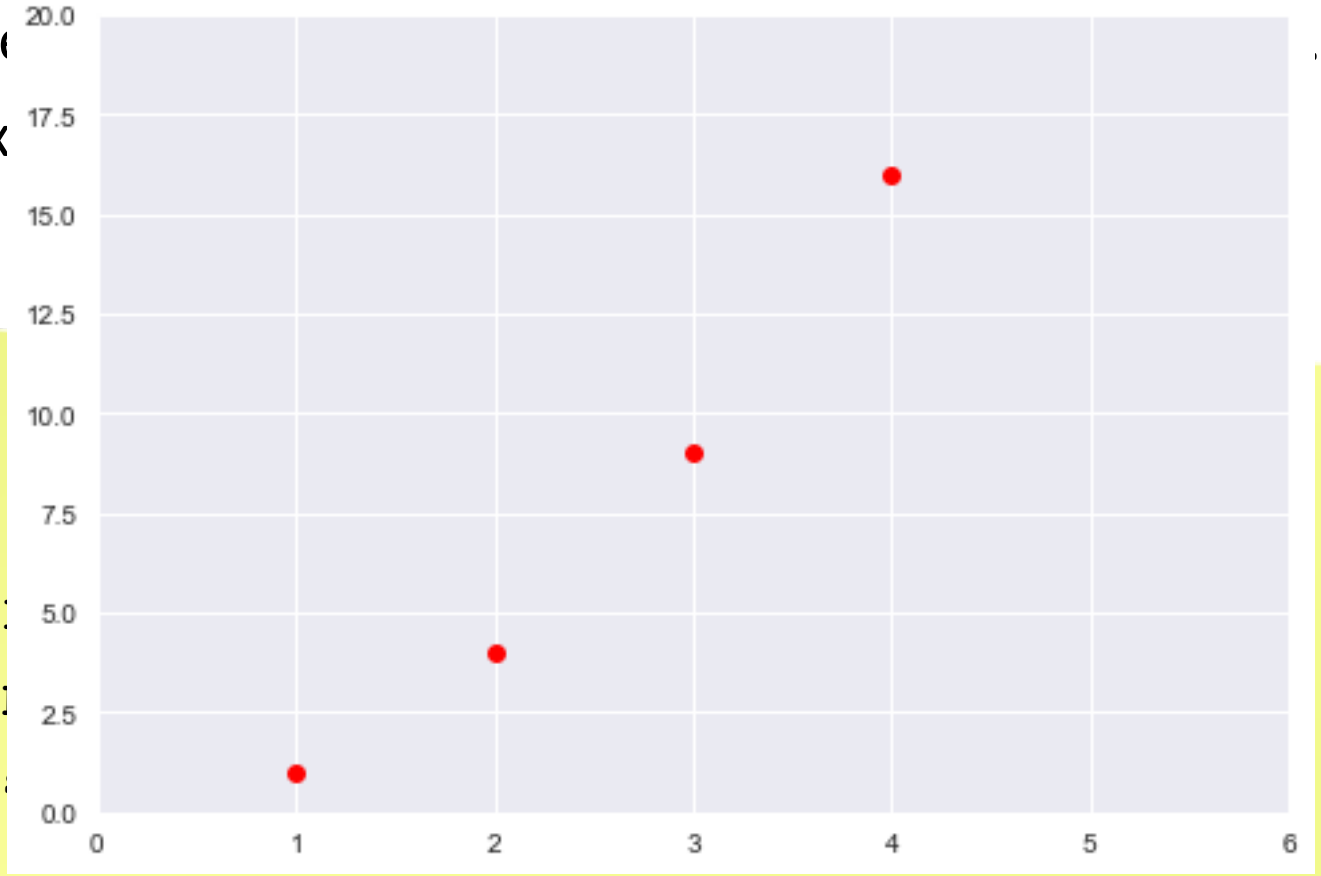
Plot Command and Formatting

- The default format string is ‘b-‘, which is a solid blue line.
- For example, to plot with red circles, you would issue:

```
import matplotlib.pyplot as plt

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

Plot Command and Formatting

- The d
- For ex



Plot Command and Formatting

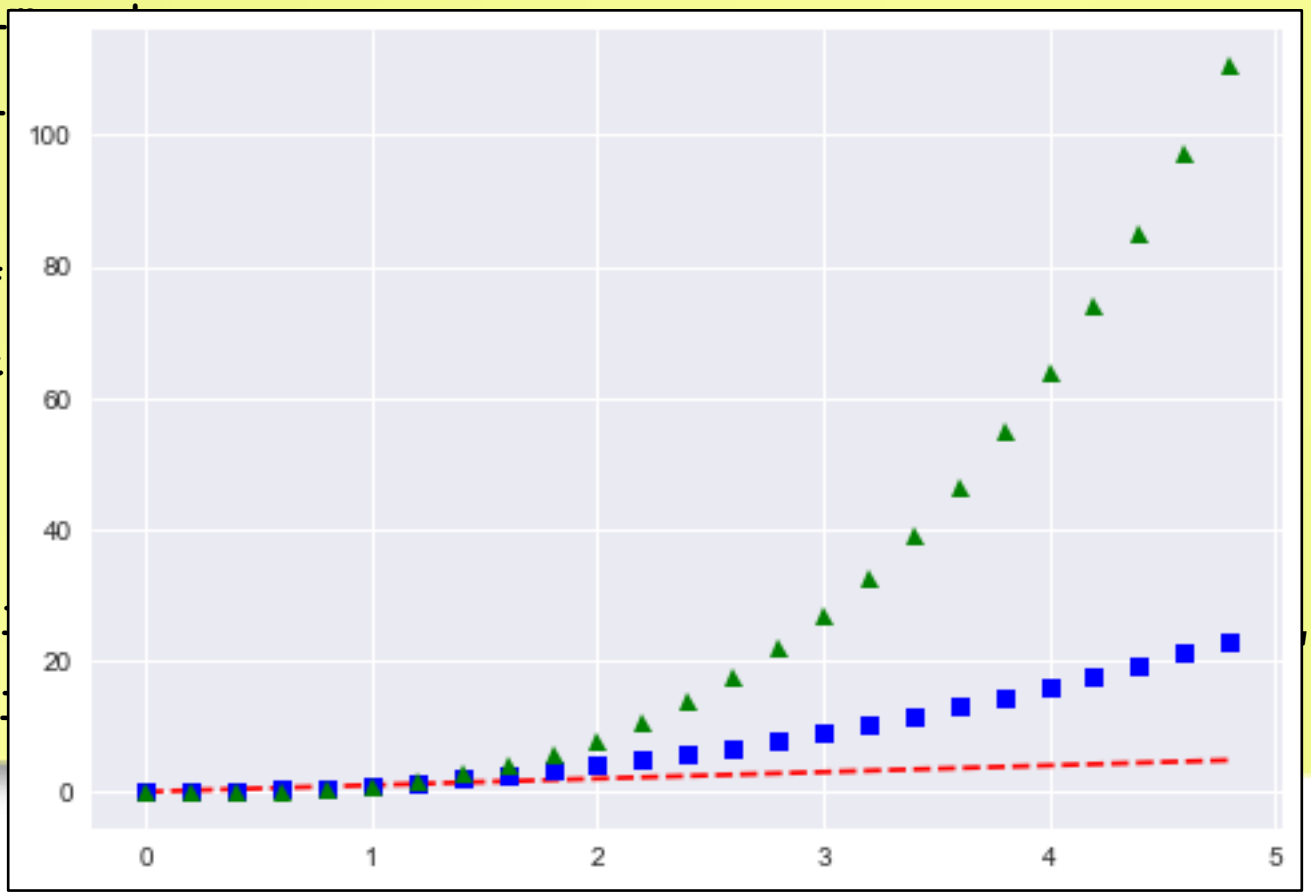
character	color
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

character	description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style
'.'	point marker
','	pixel marker
'o'	circle marker
'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
's'	square marker
'p'	pentagon marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Plotting Multiple Line using Plot Command


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

# evenly sampled time at 200ms intervals
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()
```

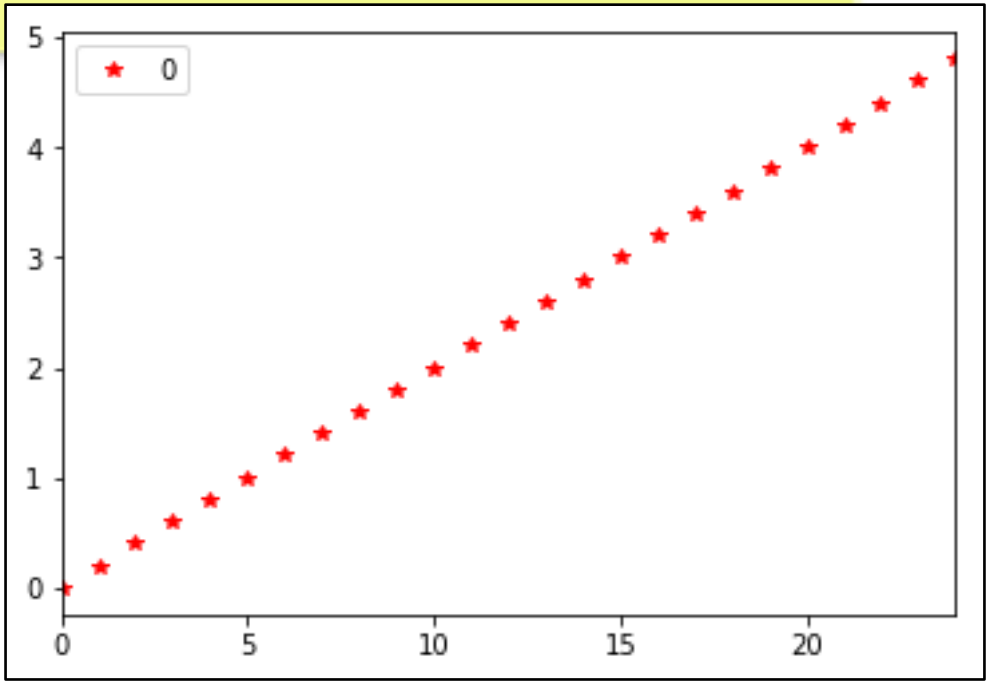
Plotting Multiple Line using Plot Command



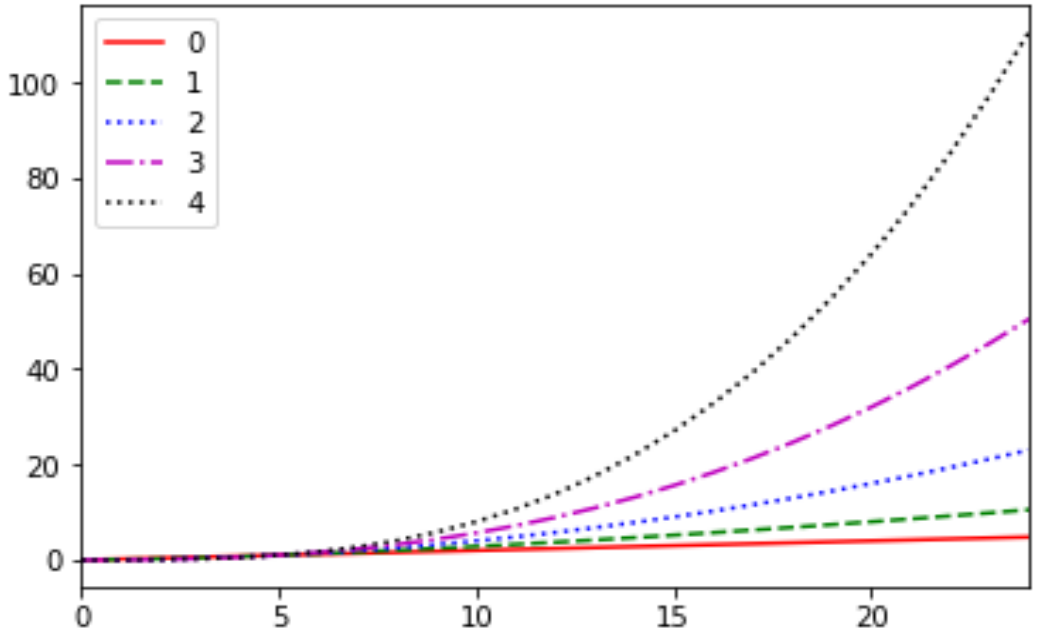
Shortcut Styles in Pandas

```
t = np.arange(0., 5., 0.2)
data = pd.DataFrame({0 : t})
# specifying color and line style for our data
ax = data.plot(style='r*')
```

We can use the style parameter in DataFrames to replicate the shortcut functionality we saw in the previous slides.



```
t = np.arange(0., 5., 0.2)
line_style = pd.DataFrame({0 : t,
                           1 : t**1.5,
                           2 : t**2.0,
                           3 : t**2.5,
                           4 : t**3.0})
ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'])
```



Notice in this example we provide a list of shortcut styles. One for each column in our DataFrame

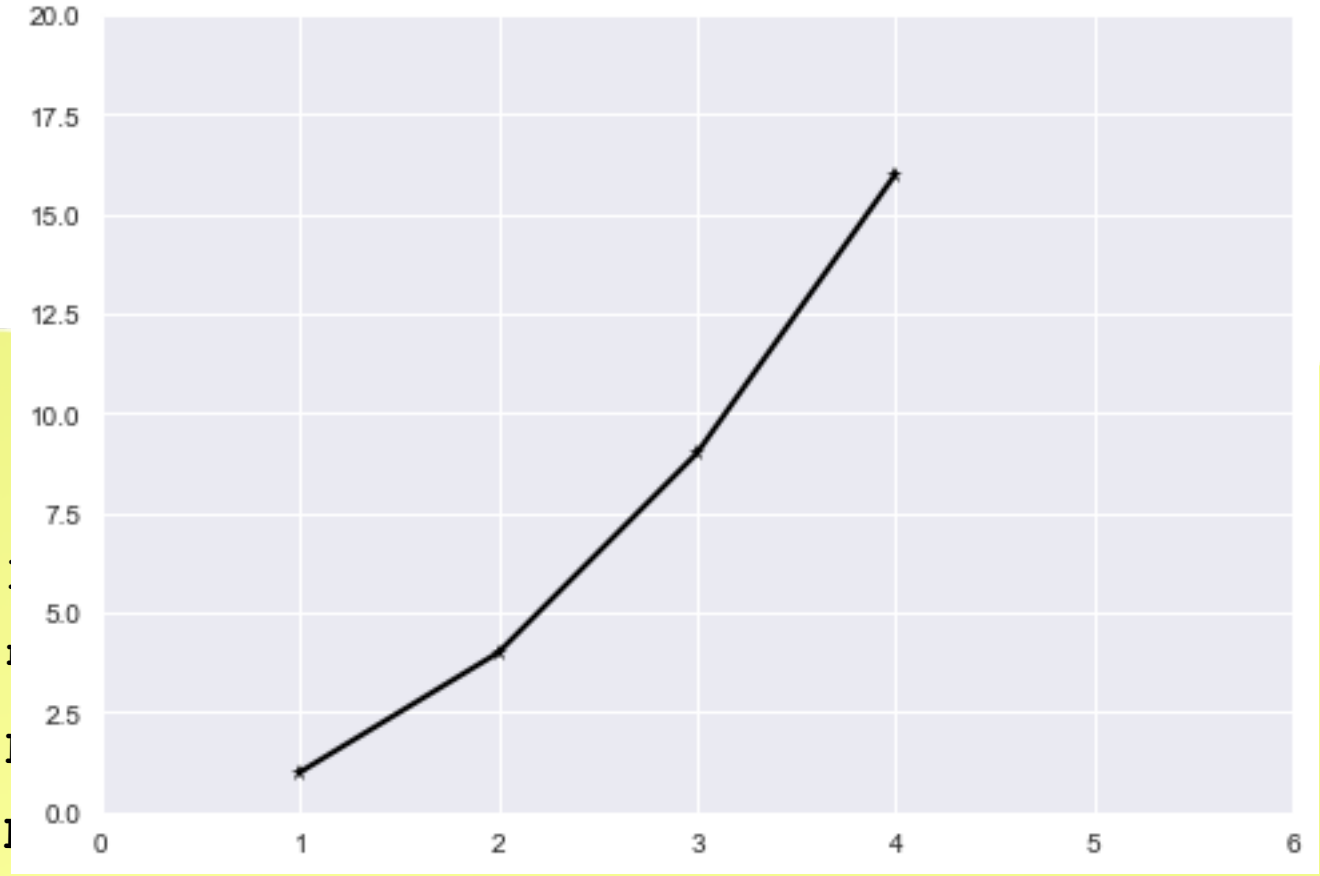
Use keyword arguments when plotting the line

- Lines have many attributes that you can set: linewidth, dash style, etc;
- Aside from the approach in the previous slides we can use keyword arguments when plotting the line
- When using the plot method we can provide additional arguments that control the line
  - plt.plot(x, y, **linewidth**=2.0, **marker** = '*', **linestyle** = '-', **color** = 'black')
  - linewidth, linestyle, color, marker, markersize, markeredgewidth, markeredgecolor, markerfacecolor, etc
  - http://matplotlib.org/api/axes_api.html#matplotlib.axes.Axes.plot

Plot Command and Formatting

```
import matplotlib.pyplot as plt


plt.plot([1,2,3,4], [1,4,9,16], linewidth=2.0,
marker = '*', linestyle = '-', color = 'black')
plt.axis([0, 6, 0, 20])
plt.show()
```

Plot Command and Formatting

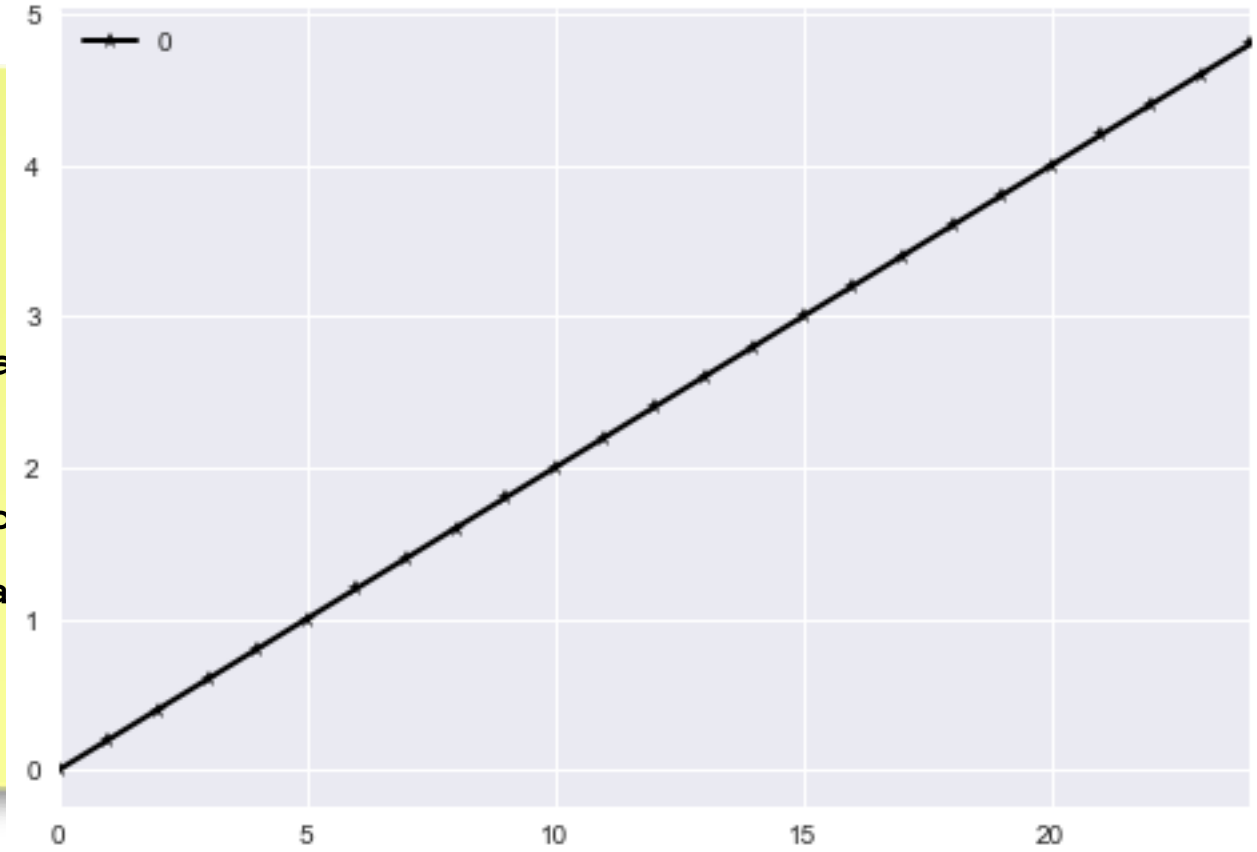


Performing Formatting in Pandas

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

data = pd.DataFrame({0 : t})

# specifying color and line style for our data
data.plot(lw = 2.0, linewidth=2.0, marker = '*',
          linestyle = '-', color = 'black')
```

Performing Formatting in Pandas

```
t =

data

# sp

data
```



Using Text in Figures

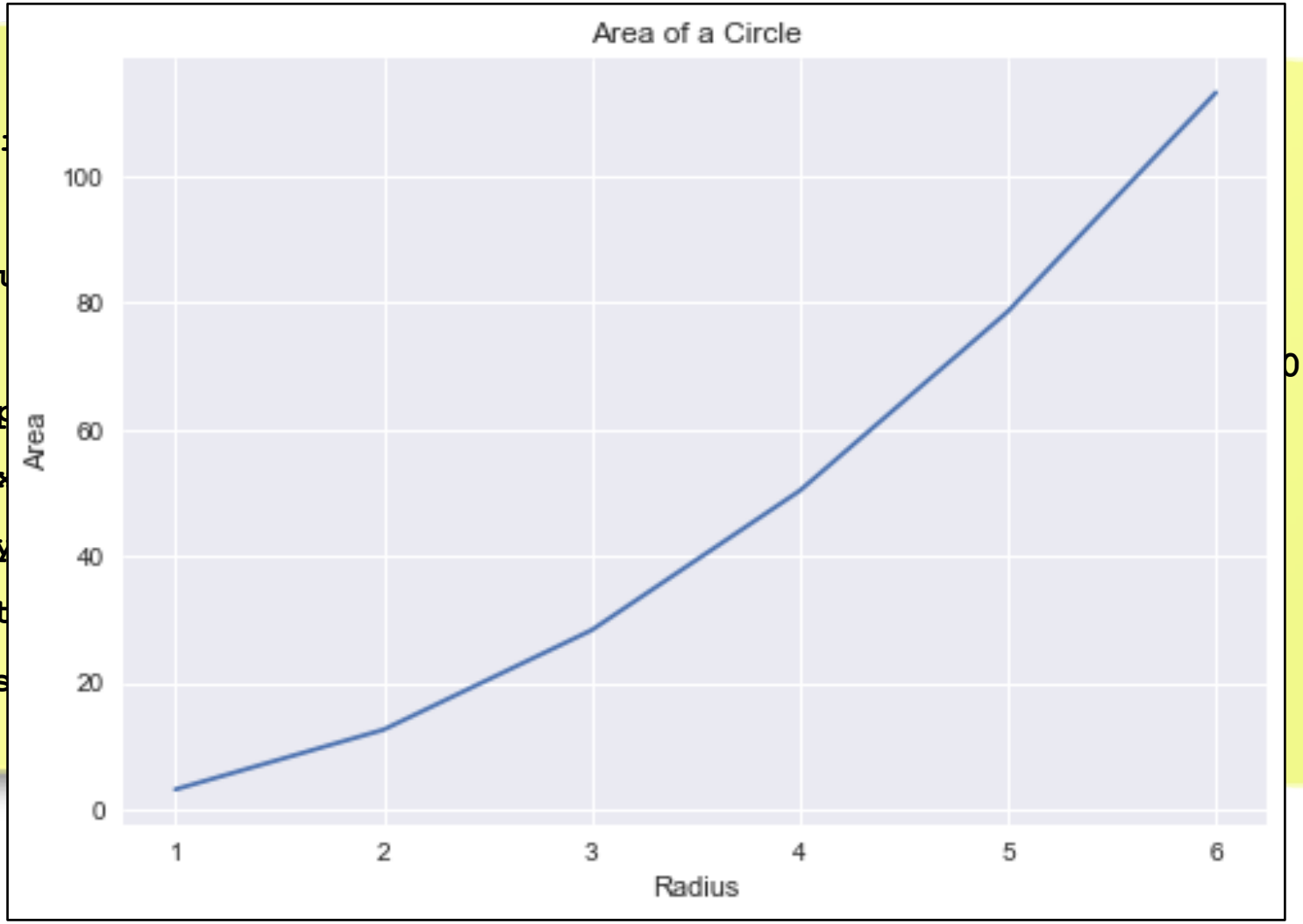
- **xlabel()** - add an axis label to the x-axis;
- **ylabel()** - add an axis label to the y-axis;
- **title()** - add a title to the Axes;
- **text()** - add text at an arbitrary location to the Axes;
- **suptitle()** - add a title to the Figure;

Adding a Title

```
import matplotlib.pyplot as plt


radius = [1.0, 2.0, 3.0, 4.0, 5.0, 6.0]
area = [3.14159, 12.56636, 28.27431, 50.26544, 78.53975, 113.09724]
plt.plot(radius, area)
plt.xlabel('Radius')
plt.ylabel('Area')
plt.title('Area of a Circle')
plt.show()
```

Adding a Title

```
impo

radiu
area
plt.p
plt.x
plt.y
plt.t
plt.s
```




```
09724]
```

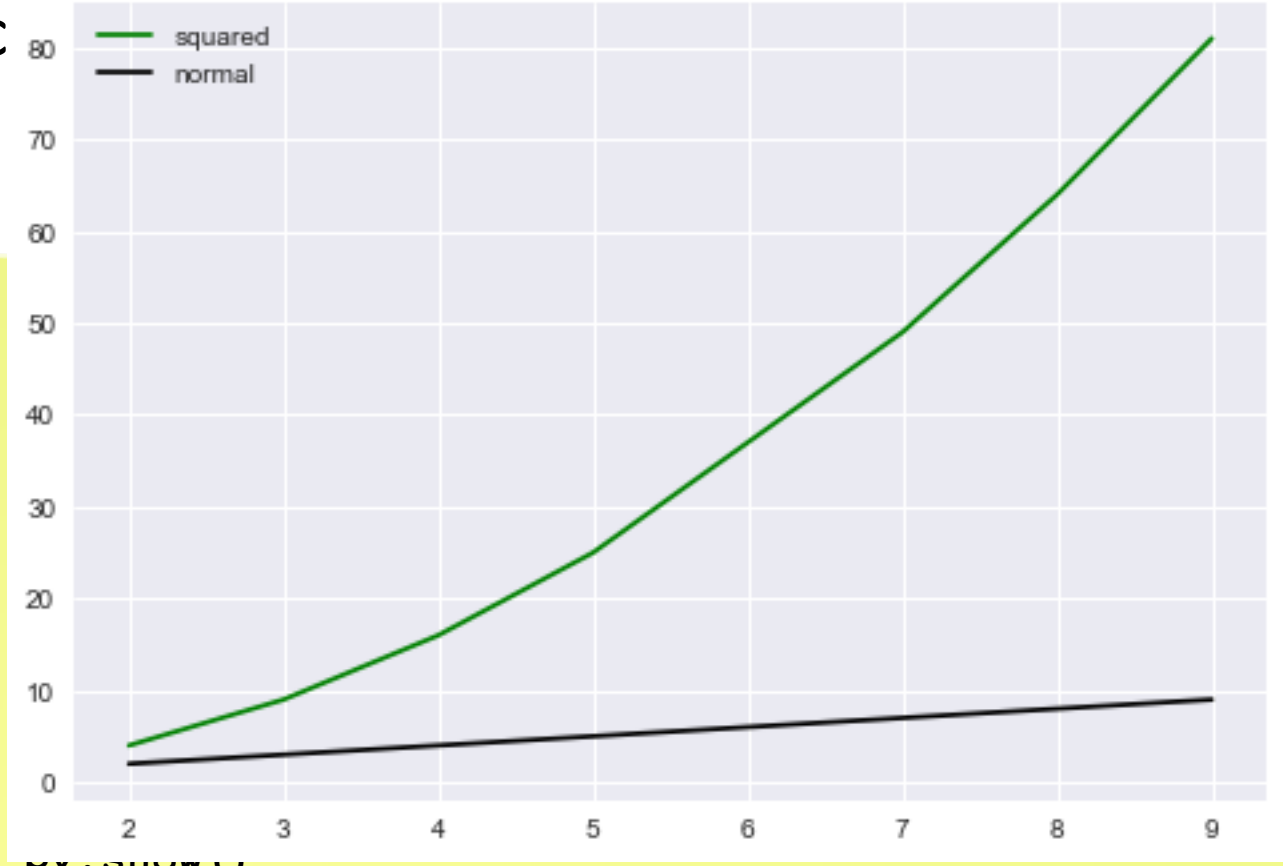
Creating a Legend

- We can use pyplot legend function to create a legend for our graph

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



t = np.array([2, 3, 4, 5, 7, 8, 9], float)
py.plot(t,t**2,'g-', t, t, 'k-')
py.legend(['squared','normal'])
py.show()
```

Creating a Legend

- We c egend for o



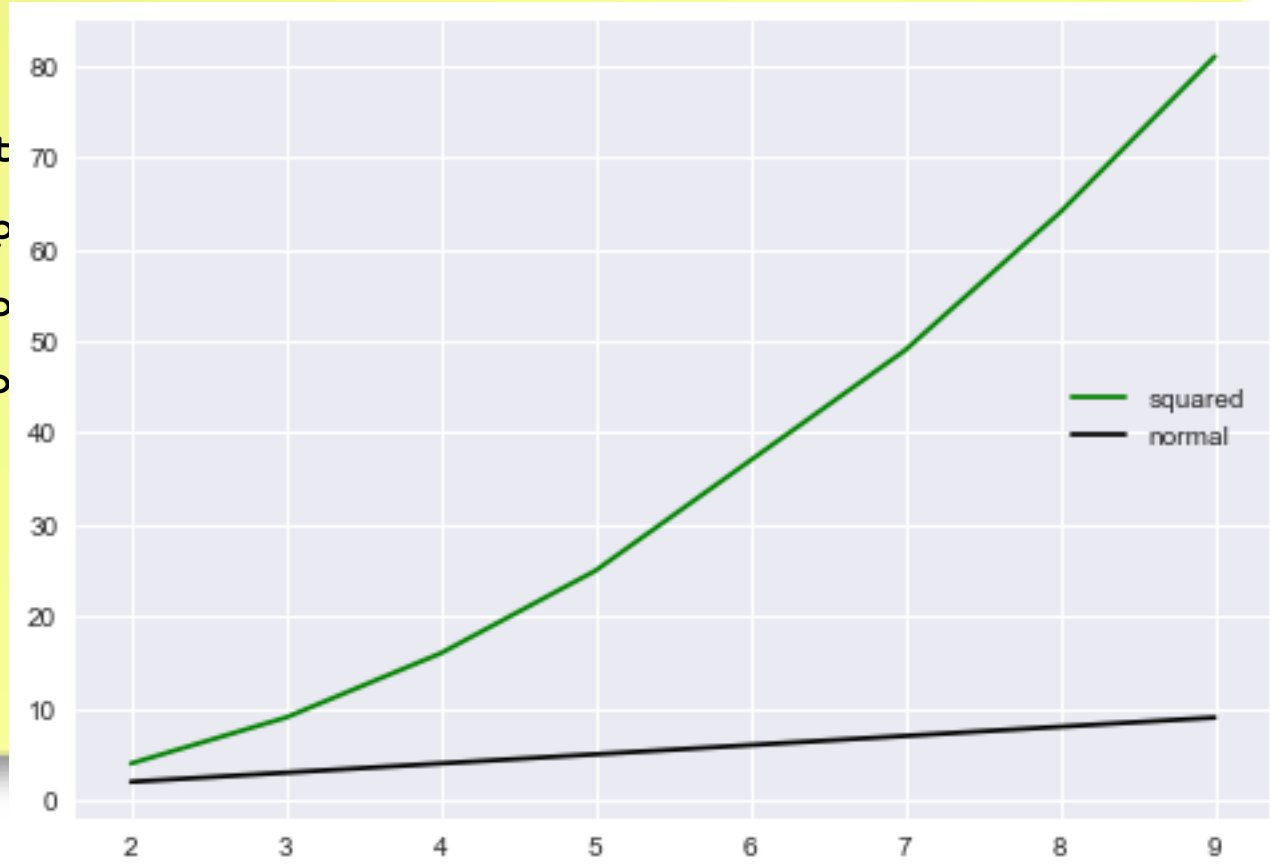
py.show()

Position of a Legend

- We can position a legend within a graph by providing a location argument:
  - legend(['label1', 'label2', 'label3‘], loc='upper left')
  - You can use the integer value or the String value

Location String	Location Code
‘best’	0
‘upper right’	1
‘upper left’	2
‘lower left’	3
‘lower right’	4
‘right’	5
‘center left’	6
‘center right’	7
‘lower center’	8
‘upper center’	9
‘center’	10

```
t = np.array([2, 3, 4, 5, 7, 8, 9], float)
py.plot(t,t**2,'g-', t, t, 'k-')
py.legend(['squared','normal'], loc = 7)
py.show()
```

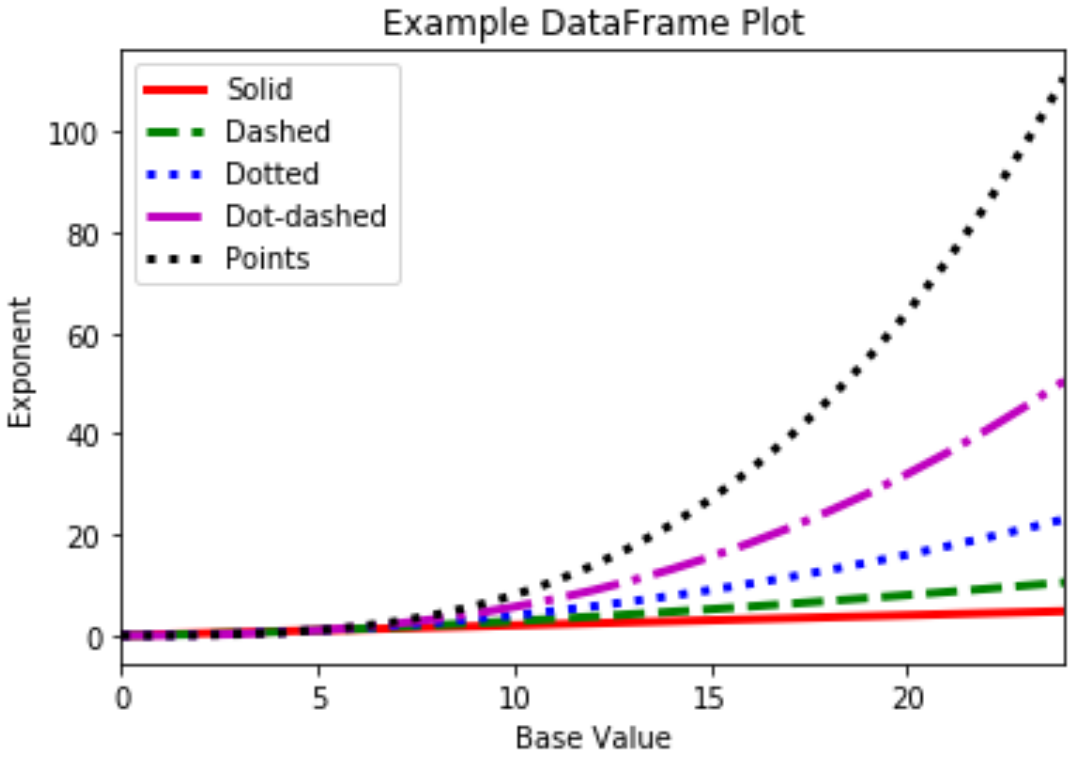

80
70
60
50
40
30
20
10
0
2
3
4
5
6
7
8
9
squared
normal

Legends, Titles, etc with Pandas

```
t = np.arange(0., 5., 0.2)
legend_labels = ['Solid', 'Dashed', 'Dotted', 'Dot-dashed', 'Points']
line_style = pd.DataFrame({0 : t,
                           1 : t**1.5,
                           2 : t**2.0,
                           3 : t**2.5,
                           4 : t**3.0})

ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'], lw=3, )
plt.title("Example DataFrame Plot")
plt.xlabel('Base Value')
plt.ylabel('Exponent');
plt.legend(legend_labels, loc='upper left')
```

Legends, Titles, etc with Pandas

```
t = np.arange(0., 5., 0.2
legend_labels = ['Solid'
line_style = pd.DataFrame
```




```
ax = line_style.plot(style=['r-', 'g--', 'b:', 'm-.', 'k:'], lw=3, )
plt.title("Example DataFrame Plot")
plt.xlabel('Base Value')
plt.ylabel('Exponent');
plt.legend(legend_labels, loc='upper left')
```

Using xticks – Place labels along x axis

- It can also be very useful to place labels along the x axis of a graph.
- The plt.xticks function allows us to **associate specific values** on the x-axis with **descriptive labels**

Discussion

CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computer Science

http://www.cit.ie

Thank you

Haithem. afli@cit.ie

@AfliHaithem