12_Matplotlib

March 30, 2020

0.1 Data Visualization Using Matplotlib

Matplotlib is the most popular data visualization library in Python. It allows us to create figures and plots, and makes it very easy to produce static raster or vector files without the need for any GUIs.

It took inspiration from MATLAB programming language and provides a similar MATLAB like interface for graphics. The beauty of this library is that it integrates well with pandas package which is used for data manipulation. With the combination of these two libraries, you can easily perform data wrangling along with visualization and get valuable insights out of data.

Matplotlib is a 2D and 3D graphics library. Figures are controlled *programmatically*, i.e. you can script it, ensure reproducibility and re-use.

Advantages

- Easy to get started
- Support for $\LaTeX$ formatted labels and texts
- Great control of every element in a figure, including figure size and DPI.
- High-quality output in many formats, including PNG, PDF, SVG, EPS.
- GUI for interactively exploring figures *and* support for headless generation of figure files (useful for batch jobs).

0.1.1 Install library

If you have Anaconda, you can simply install Matplotlib from your terminal or command prompt using:

```
conda install matplotlib
```

0.1.2 Import / Load Library

We will import Matplotlib's Pyplot module and used alias or short-form as plt

[1]:
```
import matplotlib.pyplot as plt
```

[2]:
```
#to display or show plots
%matplotlib inline
```

[3]:
```
from pylab import * #  include the symbols in the pylab module
```

0.2 Anatomy of a Plot

There are two key components in a Plot; namely, **Figure** and **Axes**.

The Figure is the top-level container that acts as the window or page on which everything is drawn. It can contain multiple independent figures, multiple Axes, a subtitle (which is a centered title for the figure), a legend, a color bar, etc.

The Axes is the area on which we plot our data and any labels/ticks associated with it. Each Axes has an X-Axis and a Y-Axis.

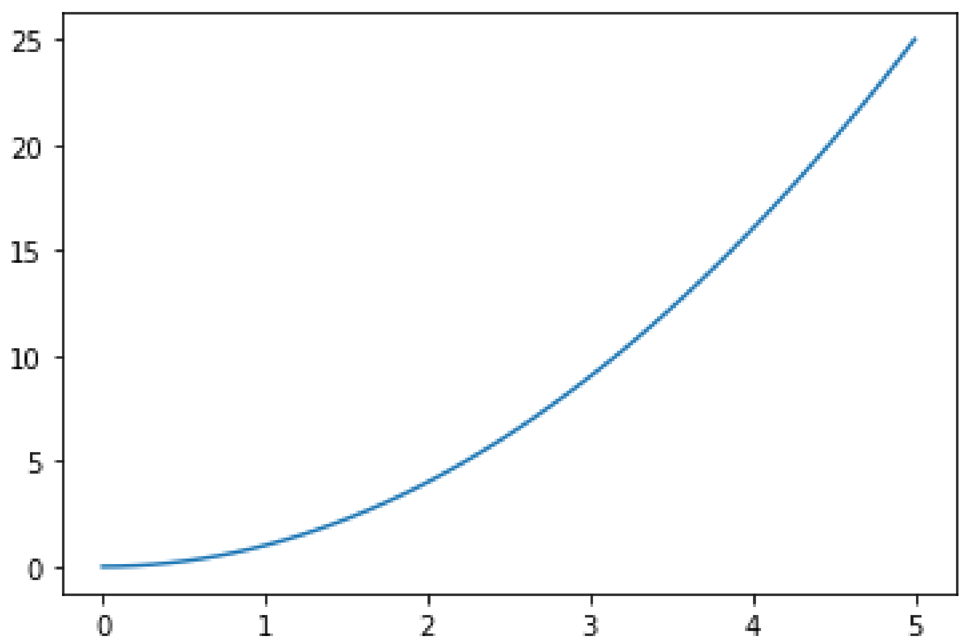
0.3 Two Approaches for creating Plots

0.3.1 1. Functional Approach:

Using the basic matplotlib command, we can easily create a plot. Let's plot an example using two Numpy arrays x and y :

[4]:
```
import numpy as np

# the function to plot
x = linspace(0, 5) # create a numpy array going from 0 to 5
y = x ** 2
plt.plot(x,y)
```

[4]:
```
[<matplotlib.lines.Line2D at 0x14508080d48>]
```

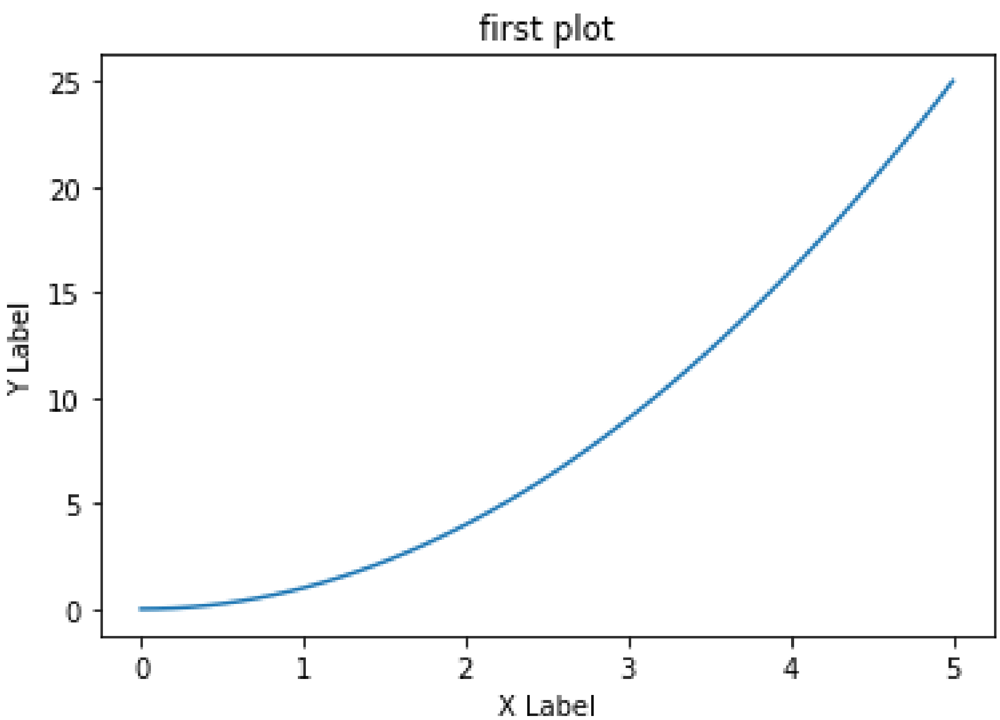


[5]:
```
linspace?
```

Now that we have a plot, let's go on to name the x-axis, y-axis, and add a title using .xlabel(), .ylabel() and .title() using:

[6]:
```
plt.plot(x,y)
plt.title("first plot")
plt.xlabel("X Label")
plt.ylabel("Y Label")
```

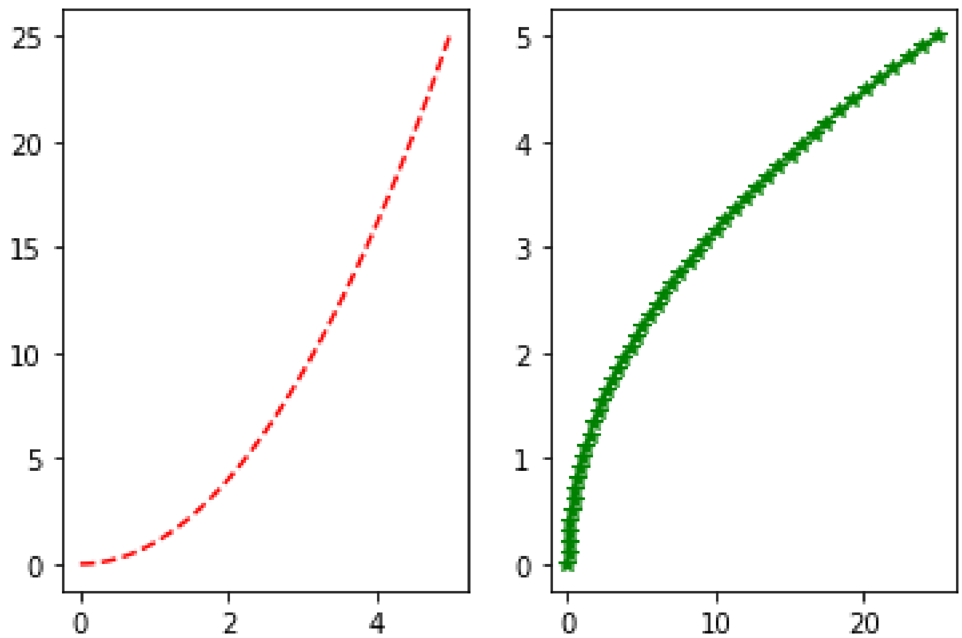
[6]:
```
Text(0, 0.5, 'Y Label')
```



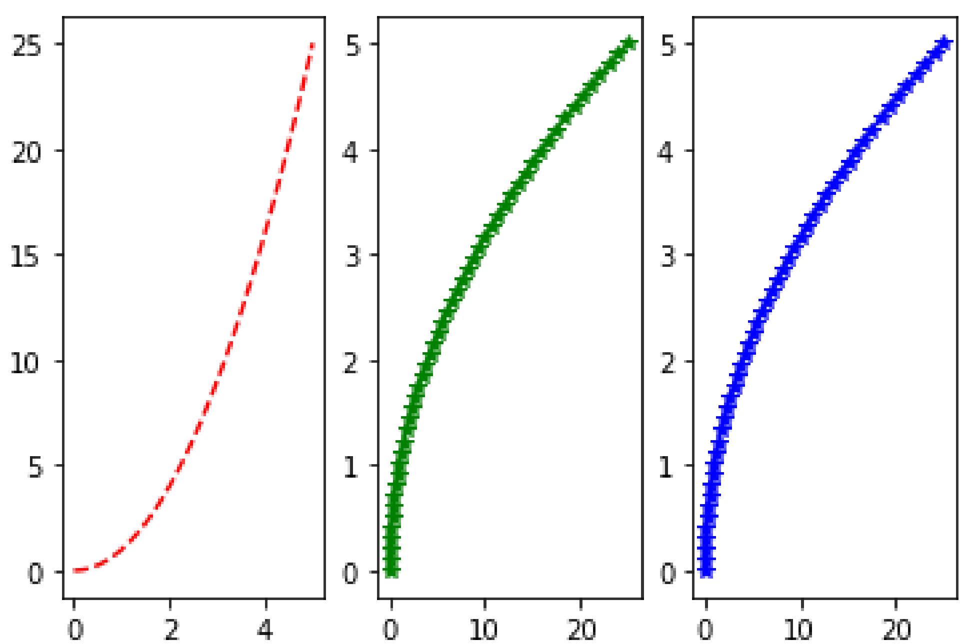
Imagine we needed more than one plot on that canvas. Matplotlib allows us easily create multi-plots on the same figure using the subplot() method. This subplot() method takes in three parameters, namely:

- `subplot(nrows, ncols, plot_number)`
  - `nrows` - number of rows in the plot figure
  - `ncols` - number of cols in the plot figure
  - `plot_number`- the plot which should be activated
    - *plot_number* starts at 1, increments across rows first an has a maximum of `nrows * ncols`.

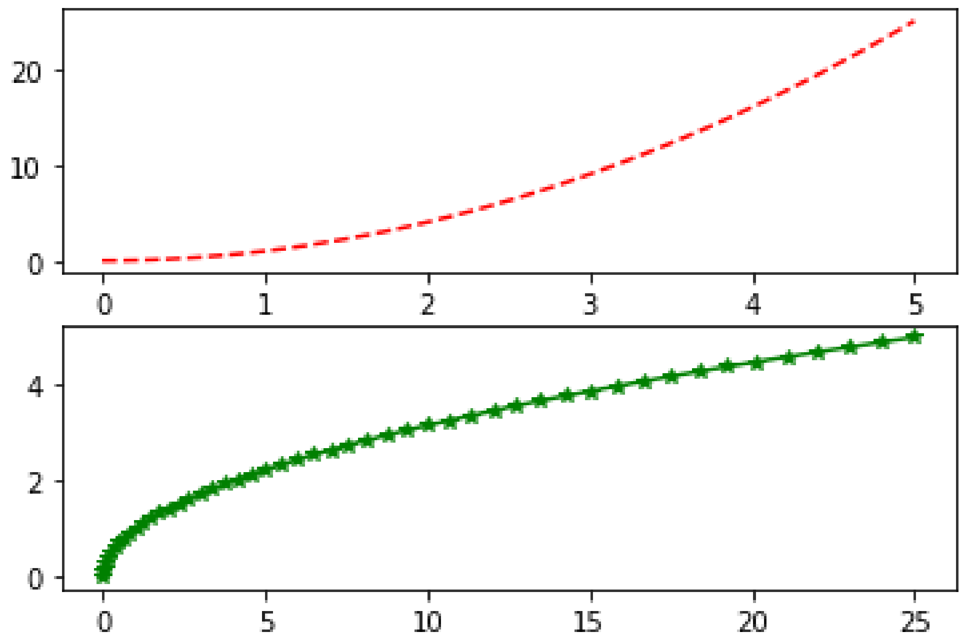
[7]:
```
subplot(1,2,1) # 1 row, 2 columns. select first subplot
plot(x, y, 'r--') #plot in red dashed lines
subplot(1,2,2) # 1 row, 2 columns. select second subplot
plot(y, x, 'g*-'); #plot in green line with marking points
```



[8]:
```
subplot(1,3,1) # 1 row, 2 columns. select first subplot
plot(x, y, 'r--') #plot in red dashed lines
subplot(1,3,2) # 1 row, 2 columns. select second subplot
plot(y, x, 'g*-'); #plot in green line with marking points
subplot(1,3,3) # 1 row, 2 columns. select second subplot
plot(y, x, 'b*-'); #plot in green line with marking points
```



[9]:
```
subplot(2,1,1) # 2 rows, 1 column. select first subplot
plot(x, y, 'r--') #plot in red dashed lines
subplot(2,1,2) # 2 rows, 1 column. select second subplot
plot(y, x, 'g*-');#plot in green line with marking points
```



0.3.2 2. Object oriented Interface:

This is the best way to create plots. The idea here is to create Figure objects and call methods off it. Let's create a blank Figure using the .figure() method.

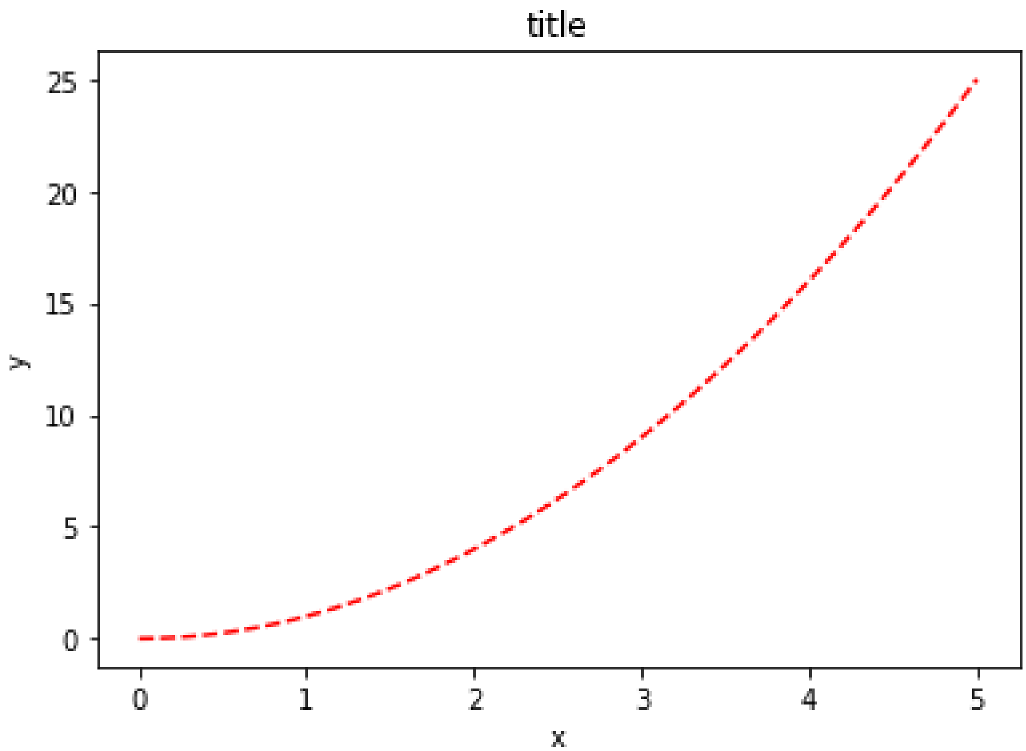
Approach * You start by creating a figure object (instance of `Figure` class) * A figure consists of `axes`, where new axes can be added via the `add_axes` method in the `Figure` class

fig = plt.figure()

The add_axes() method takes in a list of four arguments (left, bottom, width, and height—which are the positions where the axes should be placed) ranging from 0 to 1

[10]:
```
fig = plt.figure() #create new figure object

axes = fig.add_axes([0.1, 0.1, 0.8, 0.8])
# left, bottom, width, height (range 0 to 1)

axes.plot(x, y, 'r--') # plot red line
```

```
axes.set_xlabel('x') # set xlabel
axes.set_ylabel('y') # set ylabel
axes.set_title('title'); # set title
```



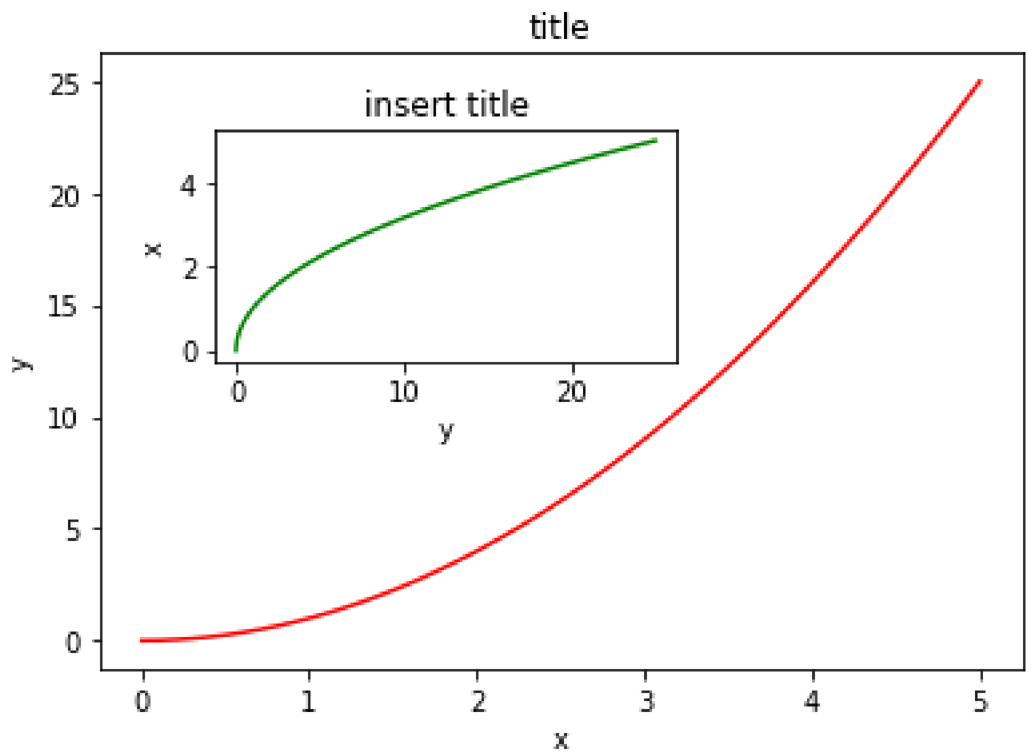
Adding axes

You can add axes (=sub plots) anywhere by specifying their bounding box `[left, bottom, width, height]`

[11]:

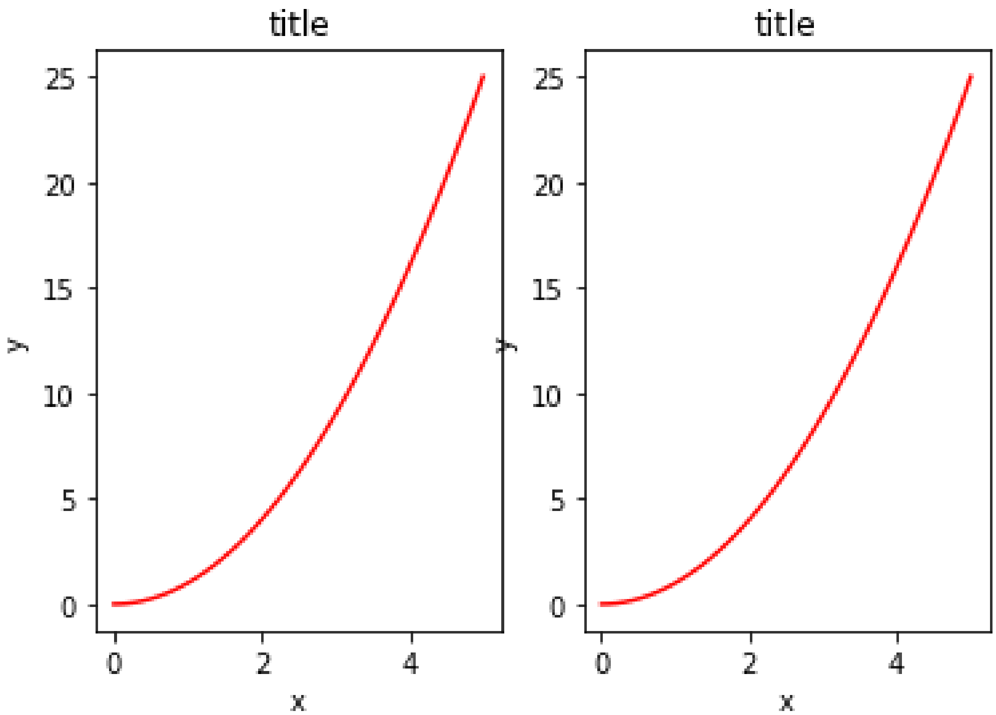
```
fig = plt.figure()
axes1 = fig.add_axes([0.1, 0.1, 0.8, 0.8]) # main axes
axes2 = fig.add_axes([0.2, 0.5, 0.4, 0.3]) # insert axes

# main figure
axes1.plot(x, y, 'r')
axes1.set_xlabel('x')
axes1.set_ylabel('y')
axes1.set_title('title')
# insert
axes2.plot(y, x, 'g')
axes2.set_xlabel('y')
axes2.set_ylabel('x')
axes2.set_title('insert title')
```

[11]: Text(0.5, 1.0, 'insert title')



Creating Subplots

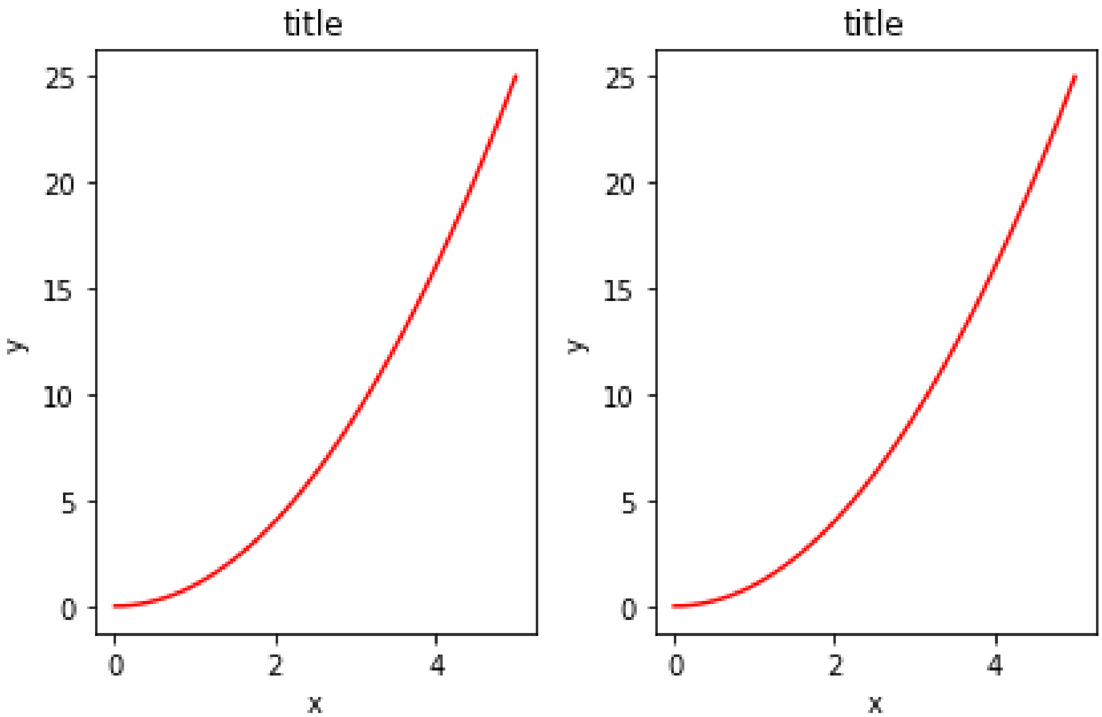
[12]:
```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title');
```



Remove Overlapping

- Use the `fig.tight_layout` method, to automatically adjusts the positions of the axes on the figure canvas so that there is no overlapping content:

[13]:
```
fig, axes = plt.subplots(nrows=1, ncols=2)

for ax in axes:
    ax.plot(x, y, 'r')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('title')

fig.tight_layout()
```



0.3.3 Layout and Formatting

Figure size, aspect ratio and DPI

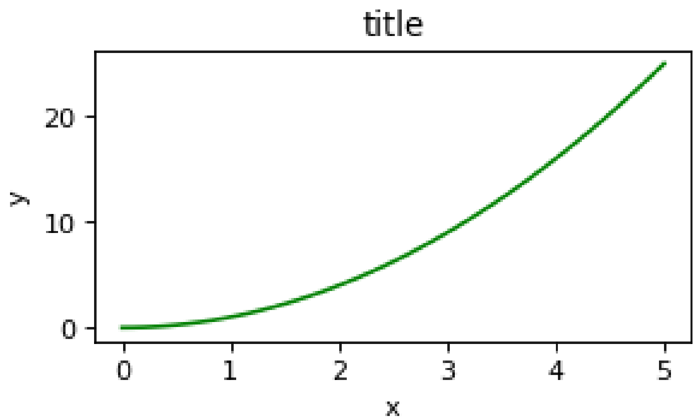
- Figures can have different aspect ratios and dots-per-inch (DPI)
- Set when creating `Figure` object using the `figsize` and `dpi` keyword arguments
- `figsize` is a tuple with width and height of the figure in inches,
- `dpi` is the dot-per-inch (pixel per inch). To create a figure with size 800 by 400 pixels we can do:

[14]:
```
fig = plt.figure(figsize=(8,4), dpi=50)
```

```
<Figure size 400x200 with 0 Axes>
```

The same arguments can also be passed to layout managers, such as the `subplots` function.

[15]:
```
fig, axes = plt.subplots(figsize=(4,2), dpi=80)

axes.plot(x, y, 'green')
axes.set_xlabel('x')
axes.set_ylabel('y')
axes.set_title('title');
```



Saving figures

To save a figure, a file we can use the `savefig` method in the `Figure` class.

[16]:
```
fig.savefig("filename1.png")
```

Here we can also optionally specify the DPI, and chose between different output formats.

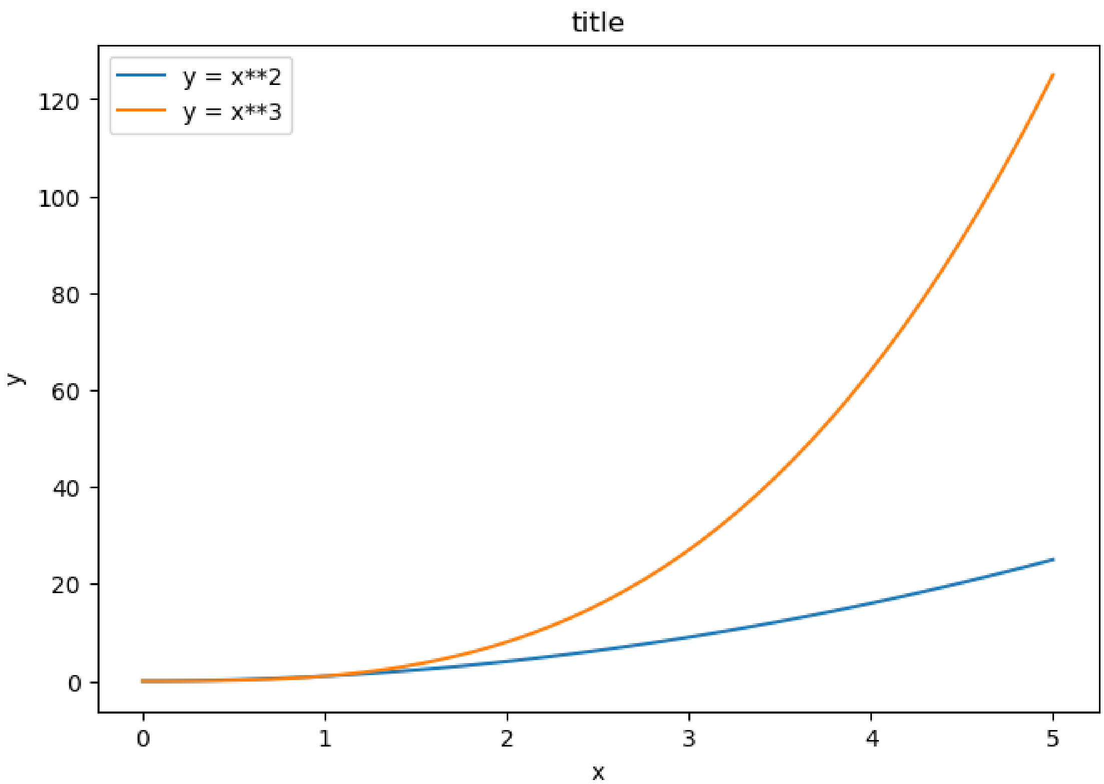
[17]:
```
fig.savefig("filename.png", dpi=200)
```

[18]:
```
fig.savefig("filename.svg")
```

0.3.4 Legends

Legends allows us to distinguish between plots. With Legends, you can use label texts to identify or differentiate one plot from another. For example, say we have a figure having two plots like below:

[19]:
```
fig = plt.figure(figsize=(6,4), dpi=100)
ax=fig.add_axes([0,0,1,1])

ax.plot(x, x**2, label="y = x**2")
ax.plot(x, x**3, label="y = x**3")
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_title('title')
ax.legend(loc=2); # upper left corner
```



The legend function takes and optional keywork argument loc that can be used to specify where in the figure the legend is to be drawn. The allowed values of loc are numerical codes for the various places the legend can be drawn. Some most common alternatives are:

[20]:
```
ax.legend(loc=0) # let matplotlib decide the optimal location
ax.legend(loc=1) # upper right corner
ax.legend(loc=2) # upper left corner
ax.legend(loc=3) # lower left corner
ax.legend(loc=4) # lower right corner
```

[20]:
```
<matplotlib.legend.Legend at 0x145085ca408>
```

0.3.5 Formatting text: LaTeX, fontsize, font family

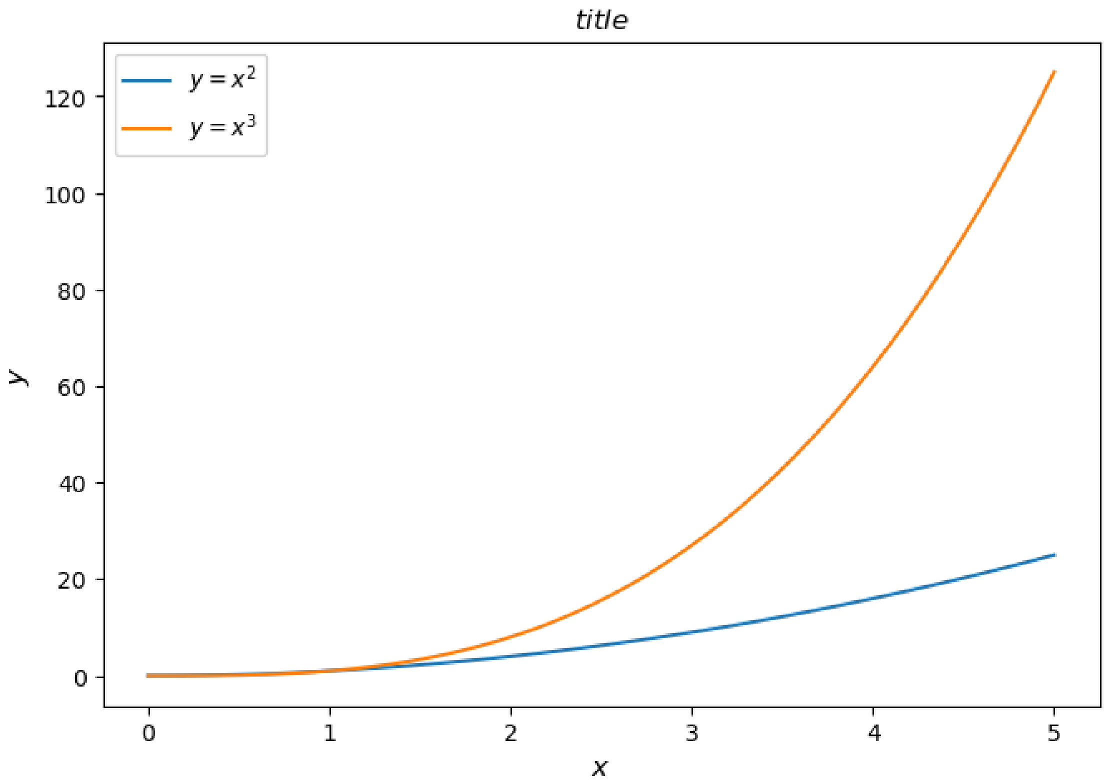
- Improve text to improve readability.
- We can use LaTeX formatted text and adjust font properties (size, font familiy, bold etc.)

Latex Support

- use dollar signs encapsulate LaTeX in any text (legend, title, label, etc.). For example, `"$y=x^3$"`.
- However, we need to escape \ for Latex commands
- Solution: use raw text strings

- r"String"
- e.g. r"\alpha" or r'\alpha' instead of "\alpha" or '\alpha'.

[21]:
```
fig = plt.figure(figsize=(6,4), dpi=100)
ax=fig.add_axes([0,0,1,1])
ax.plot(x, x**2, label=r"$y = x^2$") # use latex equations as raw strings
ax.plot(x, x**3, label=r"$y = x^3$") # use latex equations as raw strings
ax.set_xlabel(r'$x$', fontsize=12)
ax.set_ylabel(r'$y$', fontsize=12)
ax.set_title(r'$title$')
ax.legend(loc=2); # upper left corner
```



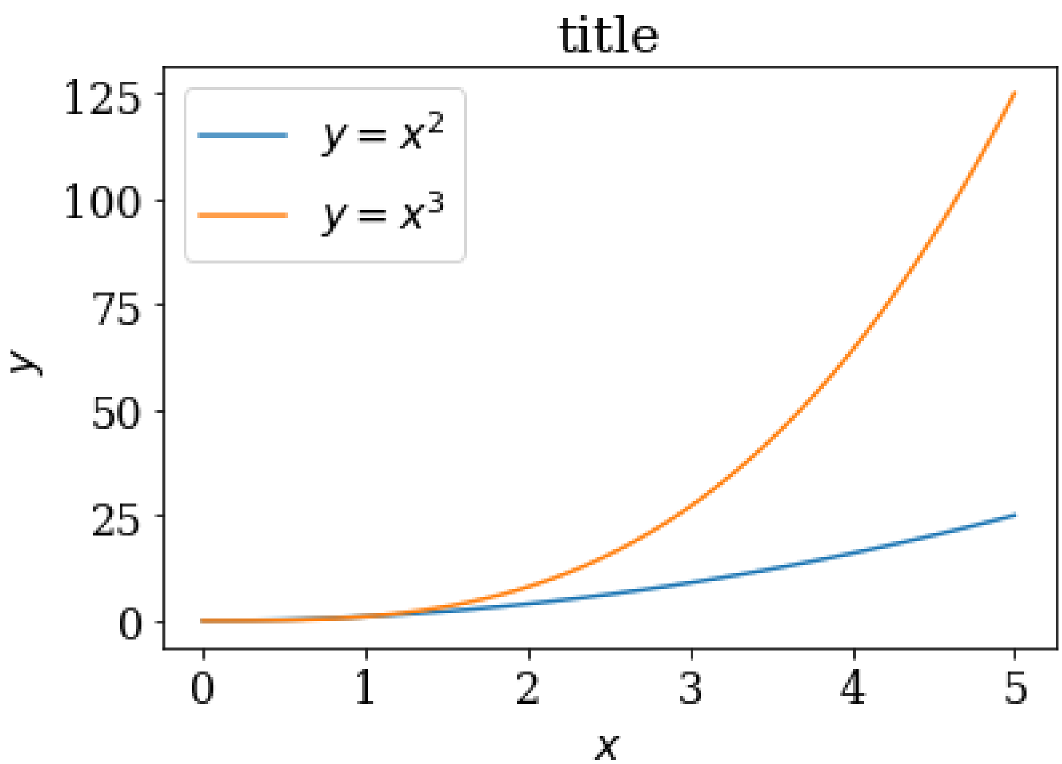
Updating Font Size

We can also change the global font size and font family, which applies to all text elements in a figure (tick labels, axis labels and titles, legends, etc.):

[22]:
```
# Update the matplotlib configuration parameters:
matplotlib.rcParams.update({'font.size': 16,
                            'font.family': 'serif'})
```

[23]:
```
fig, ax = plt.subplots()
```

```
ax.plot(x, x**2, label=r"$y = x^2$")
ax.plot(x, x**3, label=r"$y = x^3$")
ax.set_xlabel(r'$x$')
ax.set_ylabel(r'$y$')
ax.set_title('title')
ax.legend(loc=2); # upper left corner
```



[24]:
```
# restore
matplotlib.rcParams.update({'font.size': 12,
                            'font.family': 'sans'})
```

0.3.6 Setting colors, linewidths, linetypes

Colors

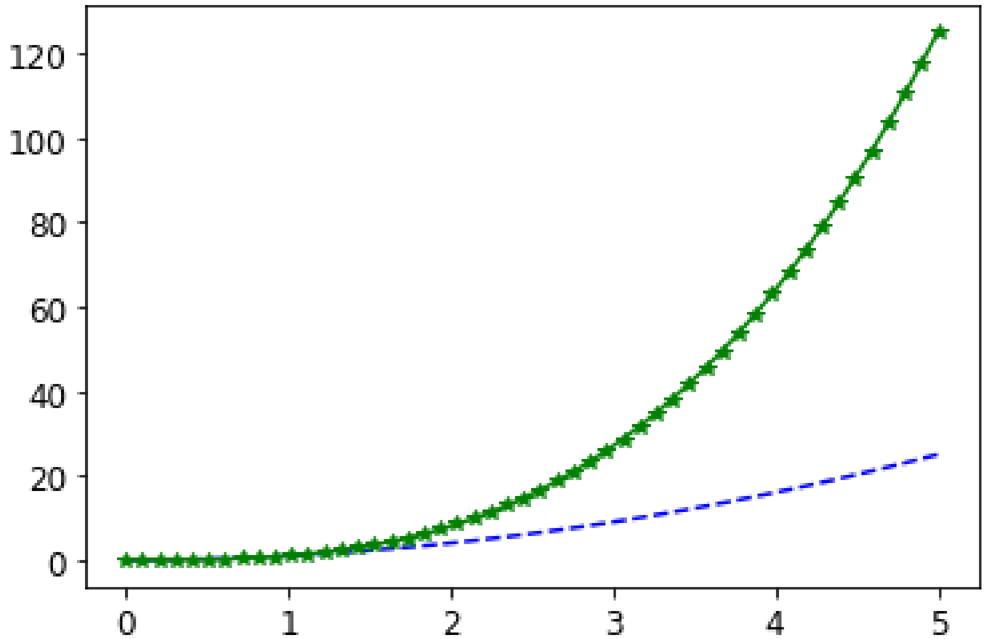
Colors of lines and other graphical elements can be defined in a number of way.

For example, we can use the MATLAB-like syntax where `'b'` means blue, `'g'` means green, etc. The MATLAB API for selecting line styles are also supported: where for example 'b.-' mean a blue line with dots.

[25]:
```
# MATLAB style line color and style
fig, ax = plt.subplots()
```

```
ax.plot(x, x**2, 'b--') # blue line with dots
ax.plot(x, x**3, 'g*-') # green dashed line
```

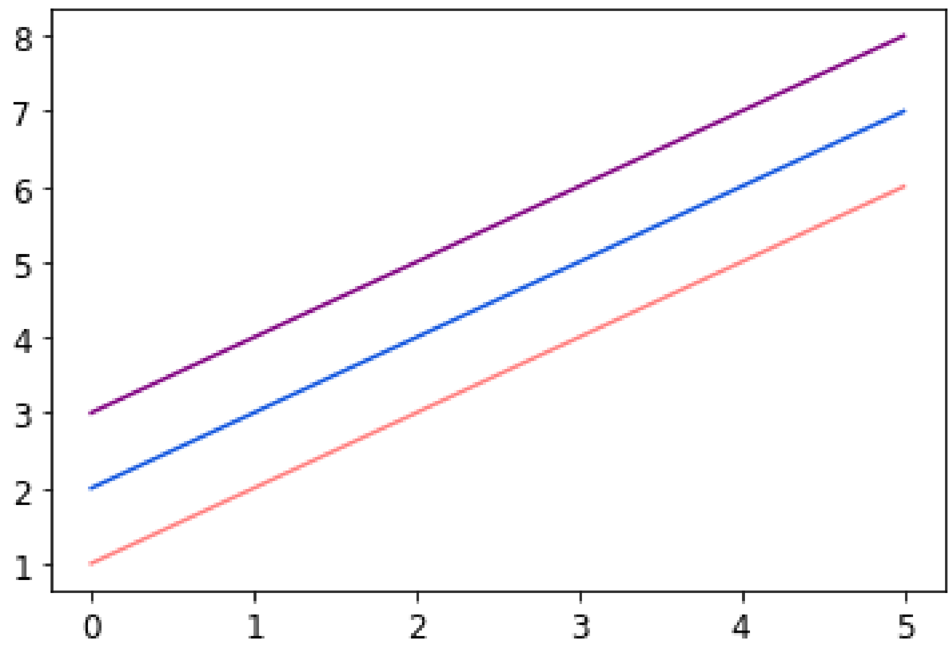
[25]: [<matplotlib.lines.Line2D at 0x145083c8848>]



In matplotlib we can also define colors by their name or RGB hex codes, and optionally provide an alpha value, using the `color` and `alpha` keyword arguments:

[26]:
```
fig, ax = plt.subplots()

ax.plot(x, x+1, color="red", alpha=0.5) # half-transparant red
ax.plot(x, x+2, color="#1155dd")        # RGB hex code for a bluish color
ax.plot(x, x+3, color="purple")        # RGB hex code for a greenish color
```

[26]: [<matplotlib.lines.Line2D at 0x145081e5488>]



0.4 Basic Plotting: using `plot`

The plot method on Series and DataFrame is just a simple wrapper around `plt.plot()`from the mathplotlib library

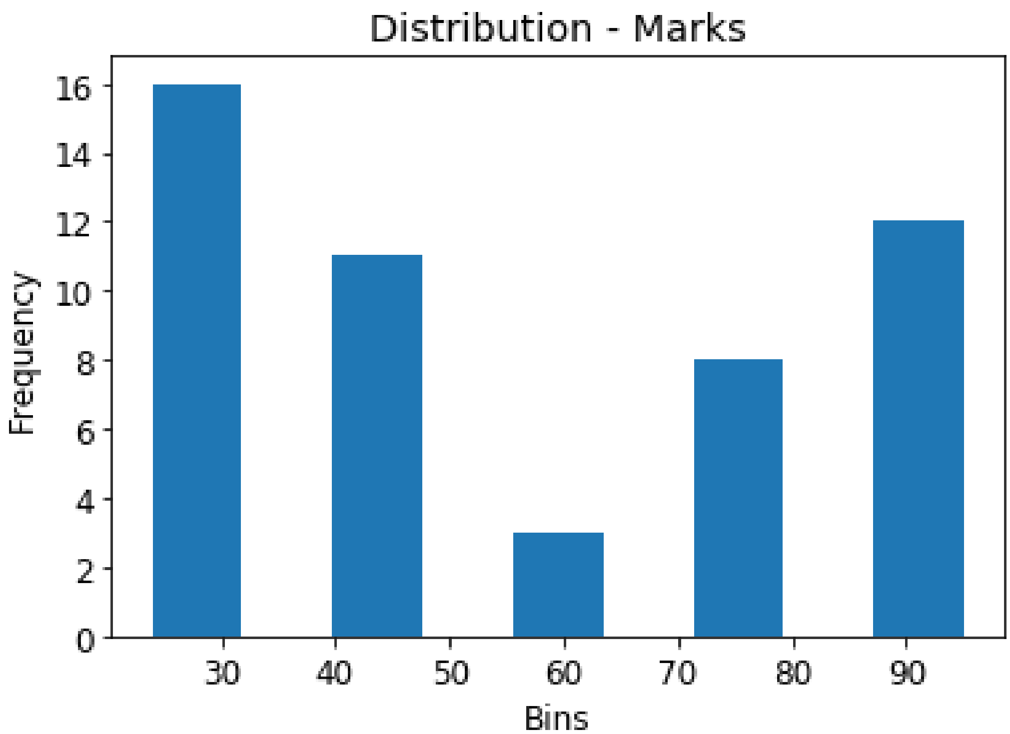
Matplotlib allows us create different kinds of plots ranging from histograms and scatter plots to bar graphs and bar charts. The key to knowing which plot to use depends on the purpose of the visualization. You may be trying to compare two quantitative variables to each other, or you might want to check for differences between groups, or you may be interested in knowing the distribution of a variable. Each of these goals is best served by different plots, and using the wrong one could distort the interpretation of the data.

Histograms: help us understand the distribution of a numeric value in a way that you cannot with mean or median alone. Using .hist() method, we can create a simple histogram:

Example 1:

[27]:
```
# Creating random data
import numpy as np
import pandas as pd
np.random.seed(1)
mydf = pd.DataFrame({"marks" :np.random.randint(low=20, high=100, size=50)})

# Histogram
ax = mydf.plot(bins= 5, kind="hist",
               rwidth = 0.5,
               title = 'Distribution - Marks',
```

```
                legend=False)
ax.set(xlabel="Bins")
plt.show()
```

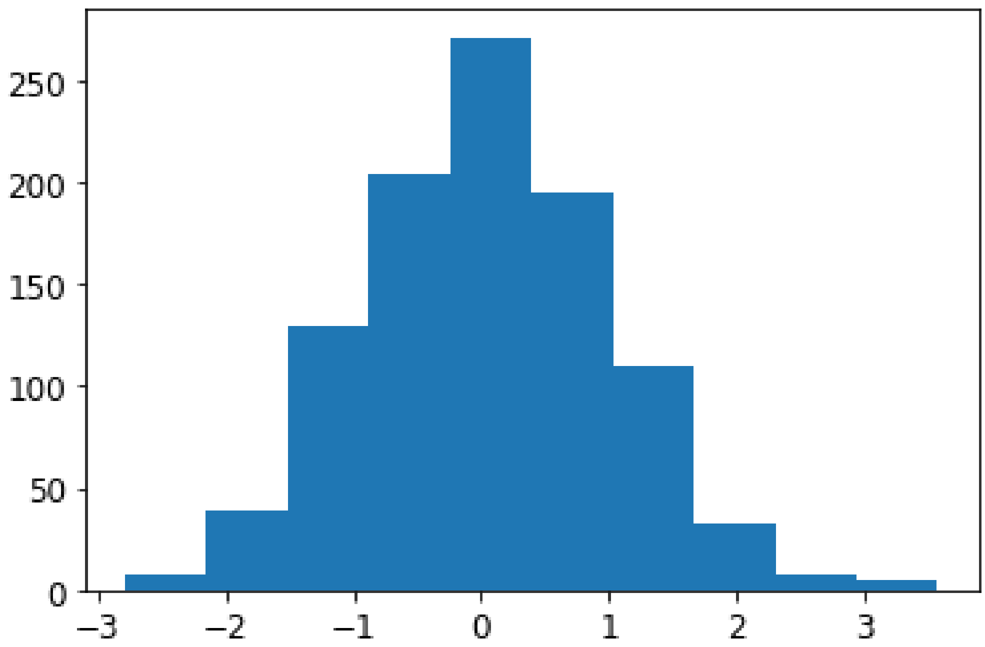


Example 2:

[28]:
```
x=np.random.randn(1000)
```

[29]:
```
plt.hist(x)
```

[29]:
```
(array([  7.,  39., 129., 204., 271., 195., 110.,  32.,   8.,   5.]),
 array([-2.78929263, -2.15302746, -1.51676229, -0.88049712, -0.24423195,
         0.39203322,  1.02829839,  1.66456356,  2.30082873,  2.9370939 ,
         3.57335907]),
 <a list of 10 Patch objects>)
```

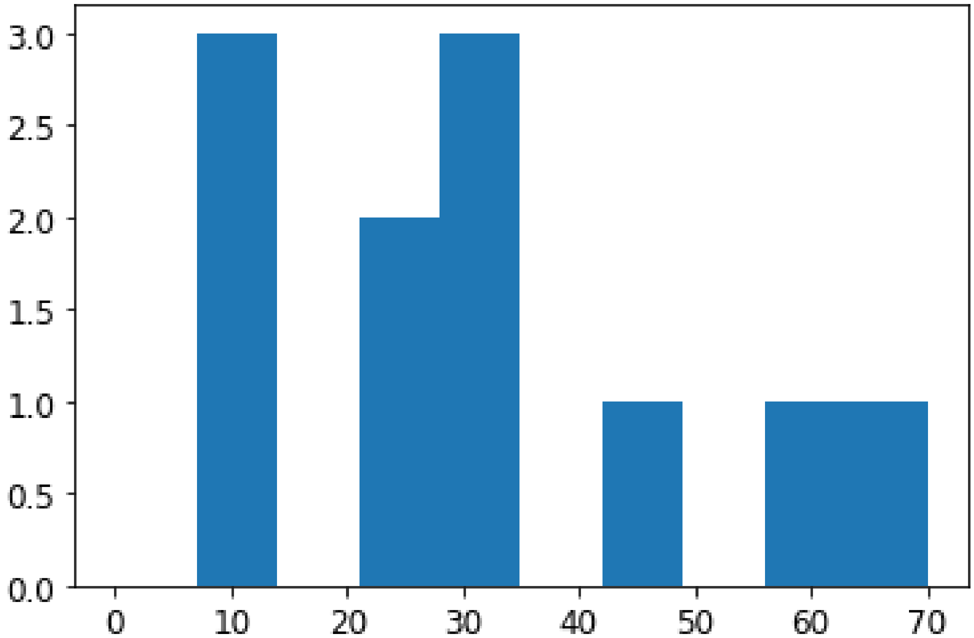


[30]:
```
hist?
```

Example 3:

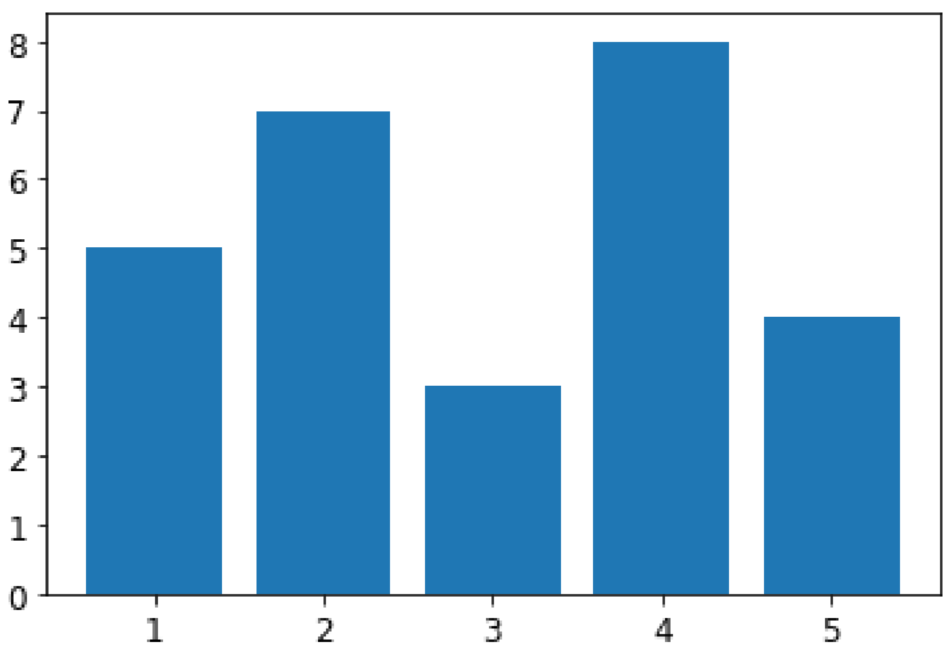
[31]:
```
age = [12,12,30,25,25,30,45,30,65,60,12]
plt.hist(age, range=(0,70))
```

[31]:
```
(array([0., 3., 0., 2., 3., 0., 1., 0., 1., 1.]),
 array([ 0.,  7., 14., 21., 28., 35., 42., 49., 56., 63., 70.]),
 <a list of 10 Patch objects>)
```



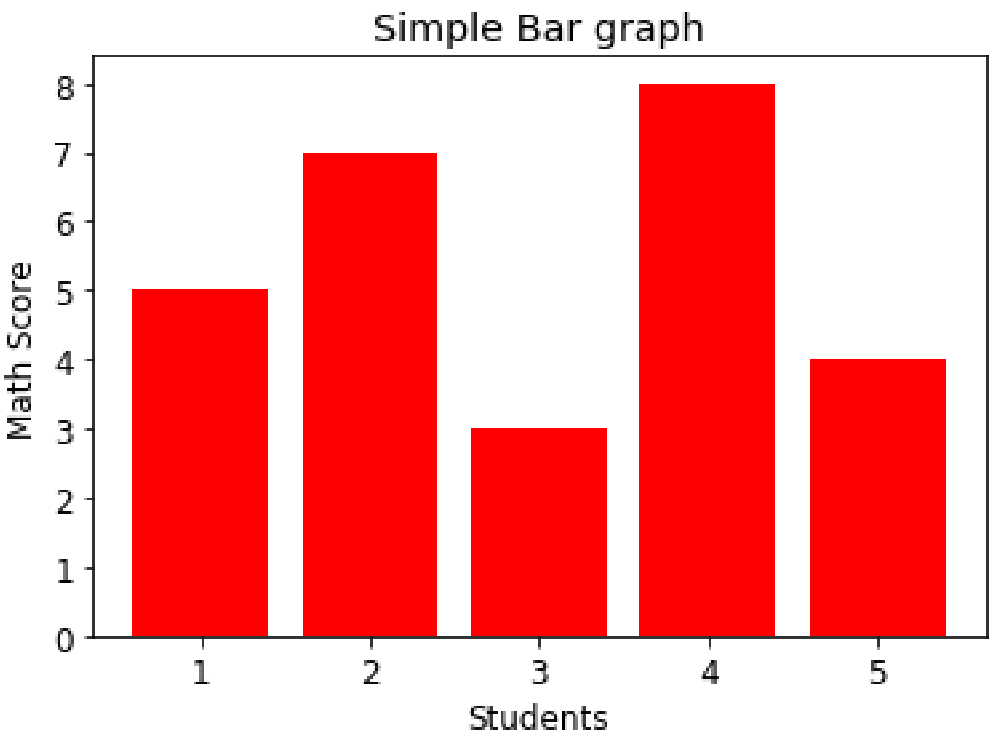
Bar graphs: are convenient for comparing numeric values of several groups. Using .bar() method, we can create a bar graph:

[32]:
```
x = [1, 2, 3, 4, 5]
y = [5, 7, 3, 8, 4]
plt.bar(x,y)
plt.show()
```



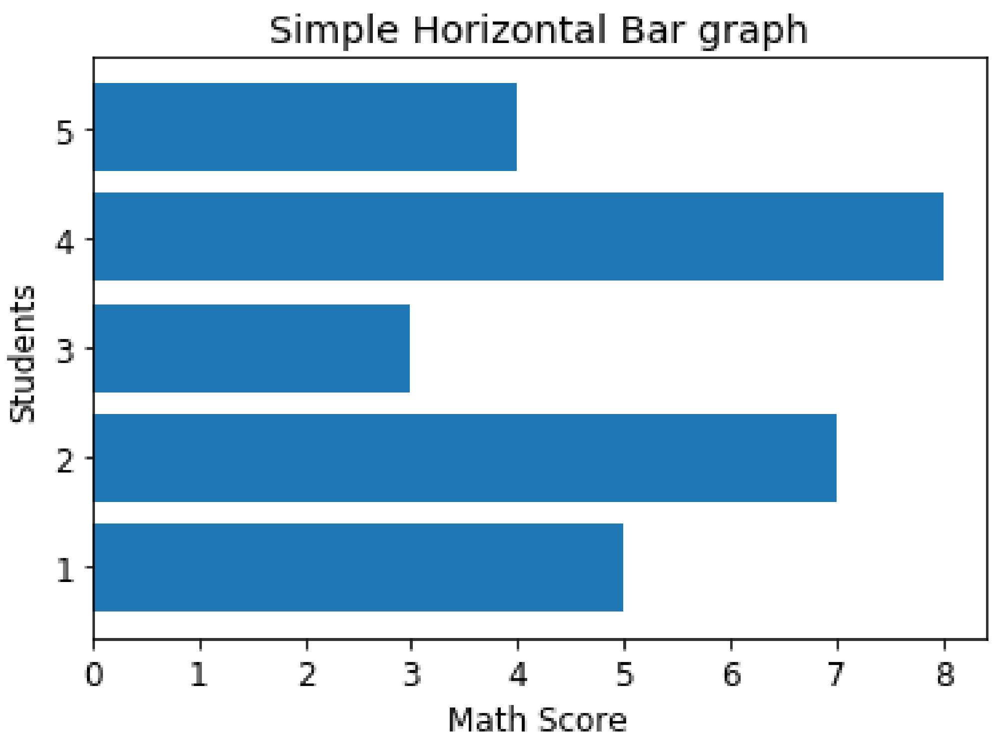
[33]:
```
plt.bar?
```

[34]:
```
plt.title("Simple Bar graph") # Name title of the graph
plt.xlabel('Students') # Assign the name of the x axis
plt.ylabel("Math Score") # Assign the name of the y axis
plt.bar(x, y, color='red') # Change bar color
plt.show()
```



[35]:
```
plt.barh(x,y)
plt.title("Simple Horizontal Bar graph")
plt.xlabel("Math Score")
plt.ylabel('Students')
```

[35]: Text(0, 0.5, 'Students')



0.4.1 Use Professional Themes / Styles for Graphs

There are many themes available in pyplot module. See the list of built-in themes which you can leverage to make your graph more graceful.

[36]:
```
print(plt.style.available)
```

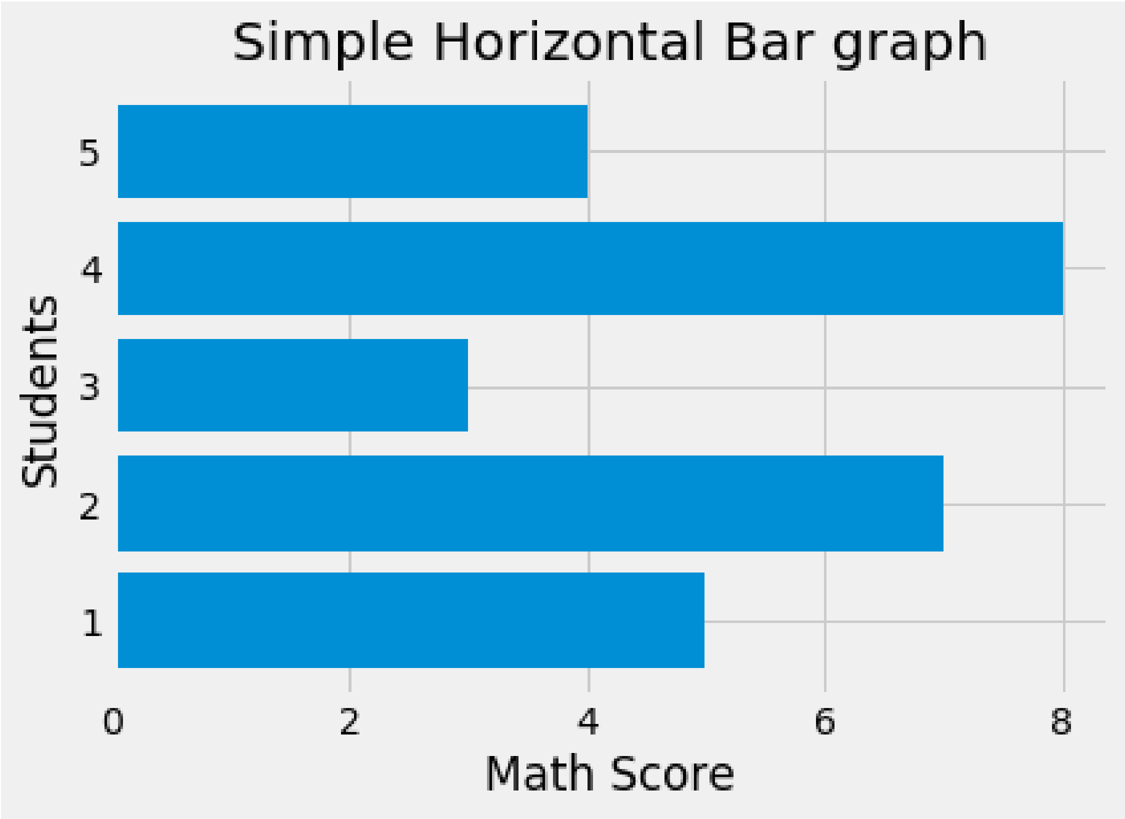
```
['bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot',
'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette',
'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-
notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk',
'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn',
'Solarize_Light2', 'tableau-colorblind10', '_classic_test']
```

Let us use fivethirtyeight theme.

[37]:
```
plt.style.use('fivethirtyeight')
```

[38]:
```
plt.barh(x,y)
plt.title("Simple Horizontal Bar graph")
plt.xlabel("Math Score")
plt.ylabel('Students')
```

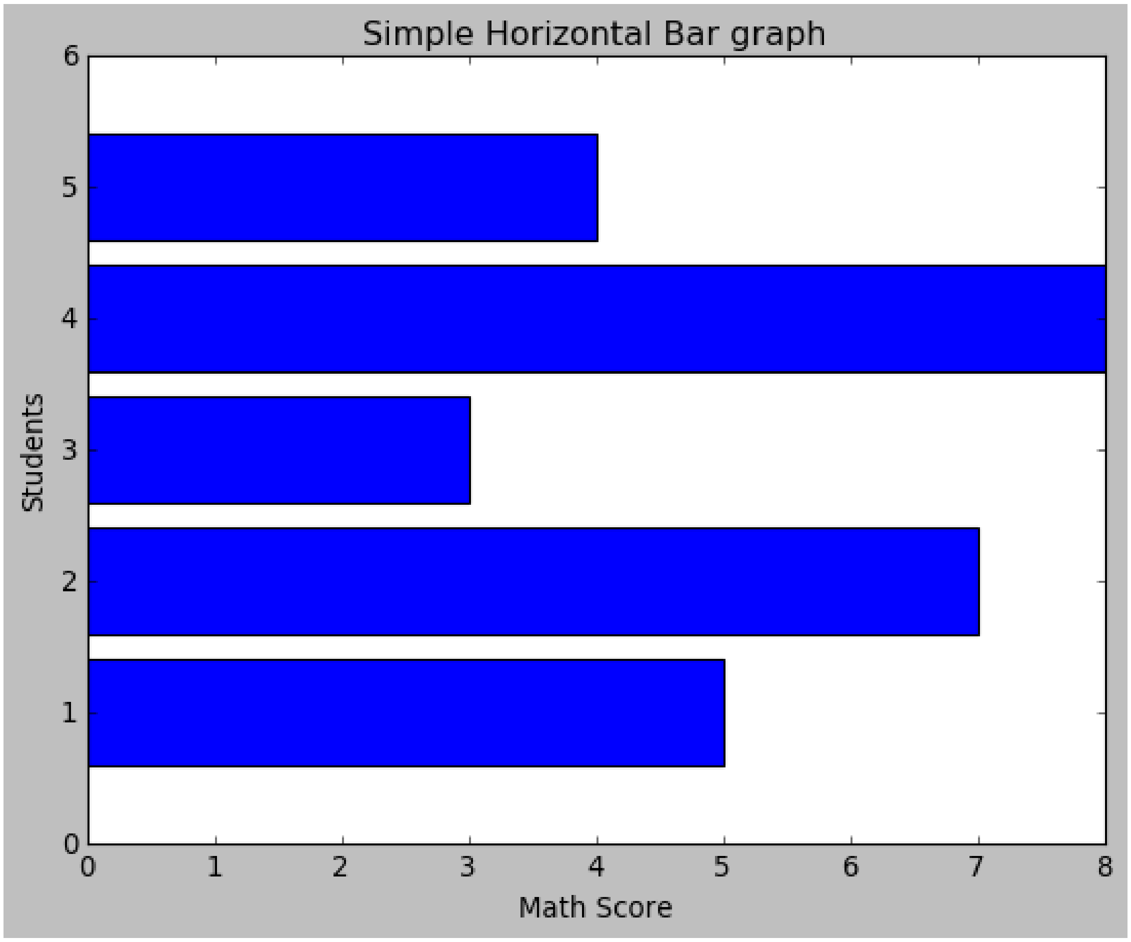
[38]:
```
Text(0, 0.5, 'Students')
```



[39]:
```
plt.style.use('classic')
```

[40]:
```
plt.barh(x,y)
plt.title("Simple Horizontal Bar graph")
plt.xlabel("Math Score")
plt.ylabel('Students')
```

[40]:
```
Text(0, 0.5, 'Students')
```

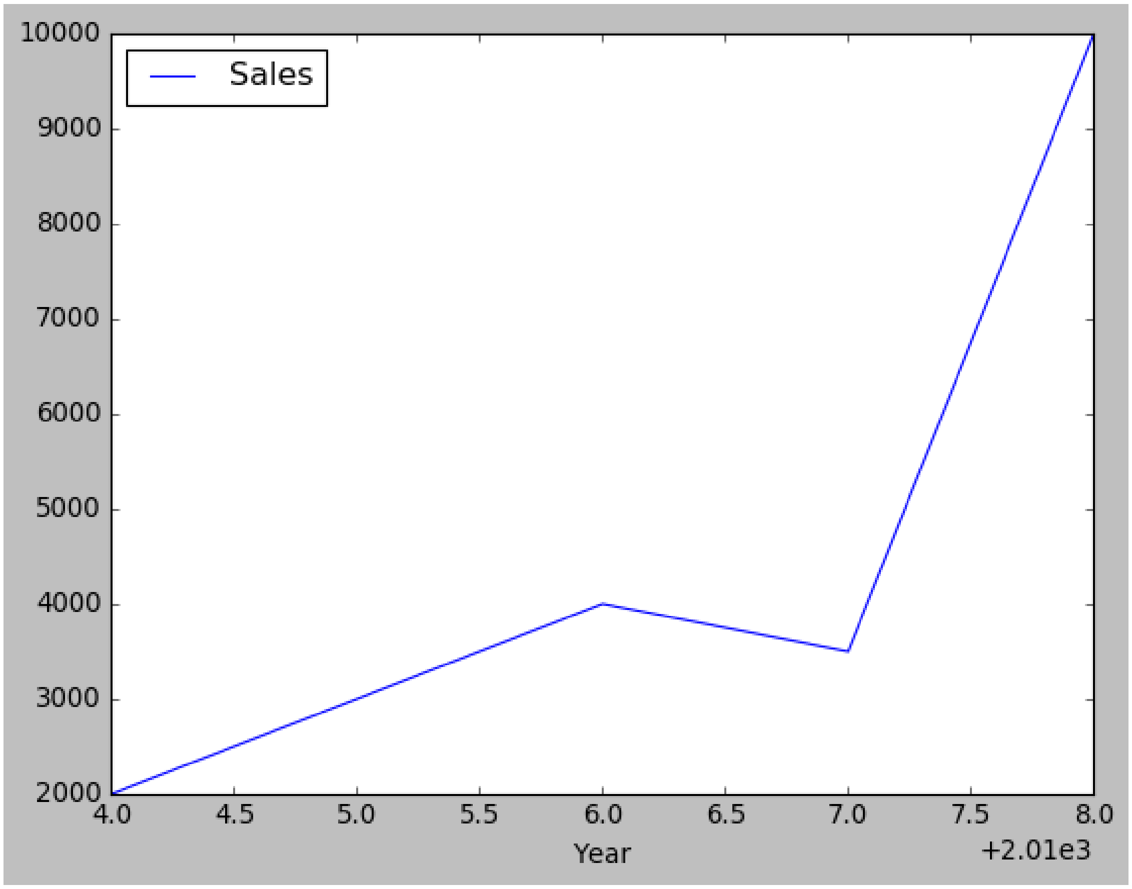


Pandas can make graphs by calling plot directly from the data frame. Plot can be called by defining plot type in kind= option. Syntax of Plot in Pandas - 'line' for line plot (Default) - 'bar' for vertical bar plots - 'barh' for horizontal bar plots - 'hist' for histogram - 'pie' for pie plots - 'box' for boxplot - 'kde' for density plots - 'area' for area plots - 'scatter' for scatter plots - 'hexbin' for hexagonal bin plots

[41]:
```
import pandas as pd

df = pd.DataFrame({"Year" : [2014,2015,2016,2017,2018],
                   "Sales" : [2000, 3000, 4000, 3500, 10000]})
```

[42]:
```
df.plot(x="Year", y="Sales", kind="line")
```

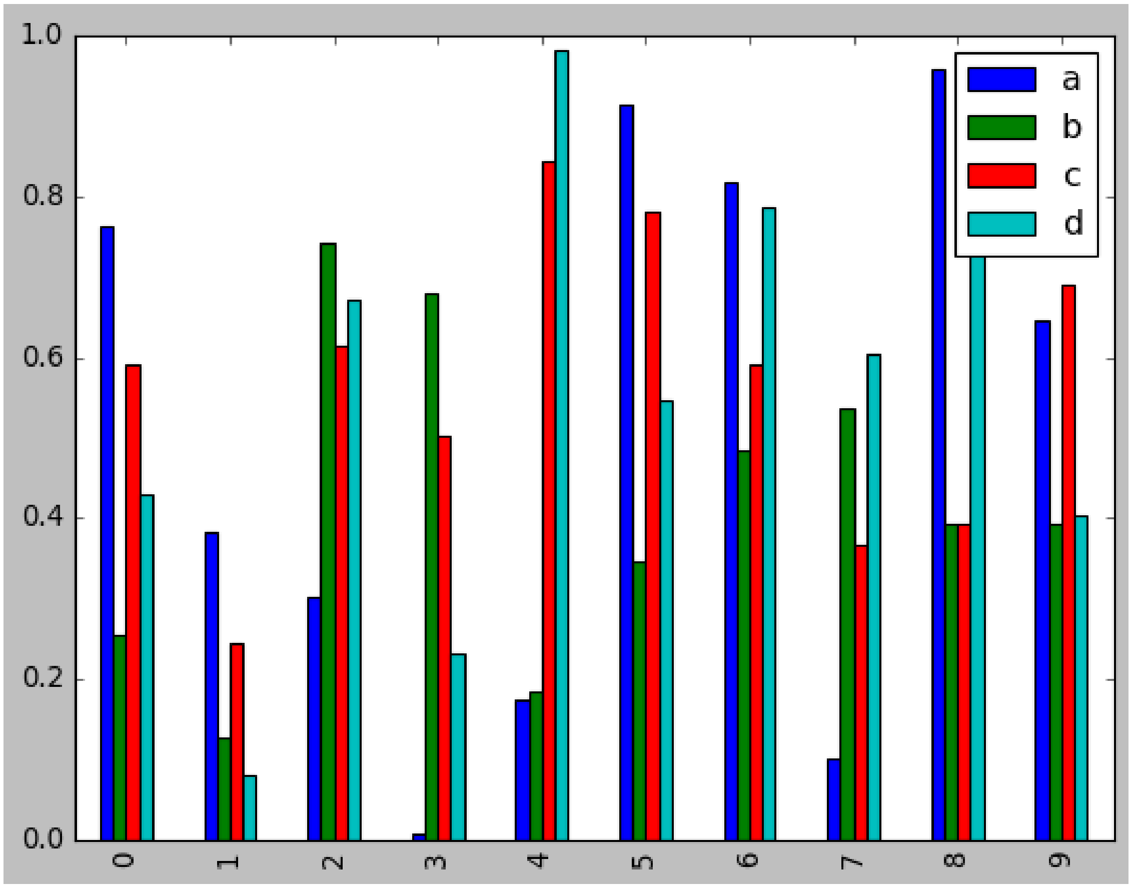
[42]: <matplotlib.axes._subplots.AxesSubplot at 0x1450a998988>



[43]:
```
#data frame
df2 = pd.DataFrame(np.random.rand(10, 4),
                   columns=['a', 'b', 'c', 'd'])
print(df2)
df2.plot(kind='bar')
```

```
          a         b         c         d
0  0.763873  0.253889  0.591387  0.428795
1  0.382101  0.126713  0.244267  0.079342
2  0.302858  0.741705  0.615016  0.672411
3  0.008383  0.678377  0.503181  0.230908
4  0.174897  0.184983  0.844260  0.982590
5  0.913740  0.347369  0.780225  0.546455
6  0.816442  0.483727  0.589951  0.787263
7  0.100235  0.537007  0.365983  0.604307
8  0.958803  0.392823  0.392973  0.790637
9  0.645078  0.393884  0.688955  0.404002
```

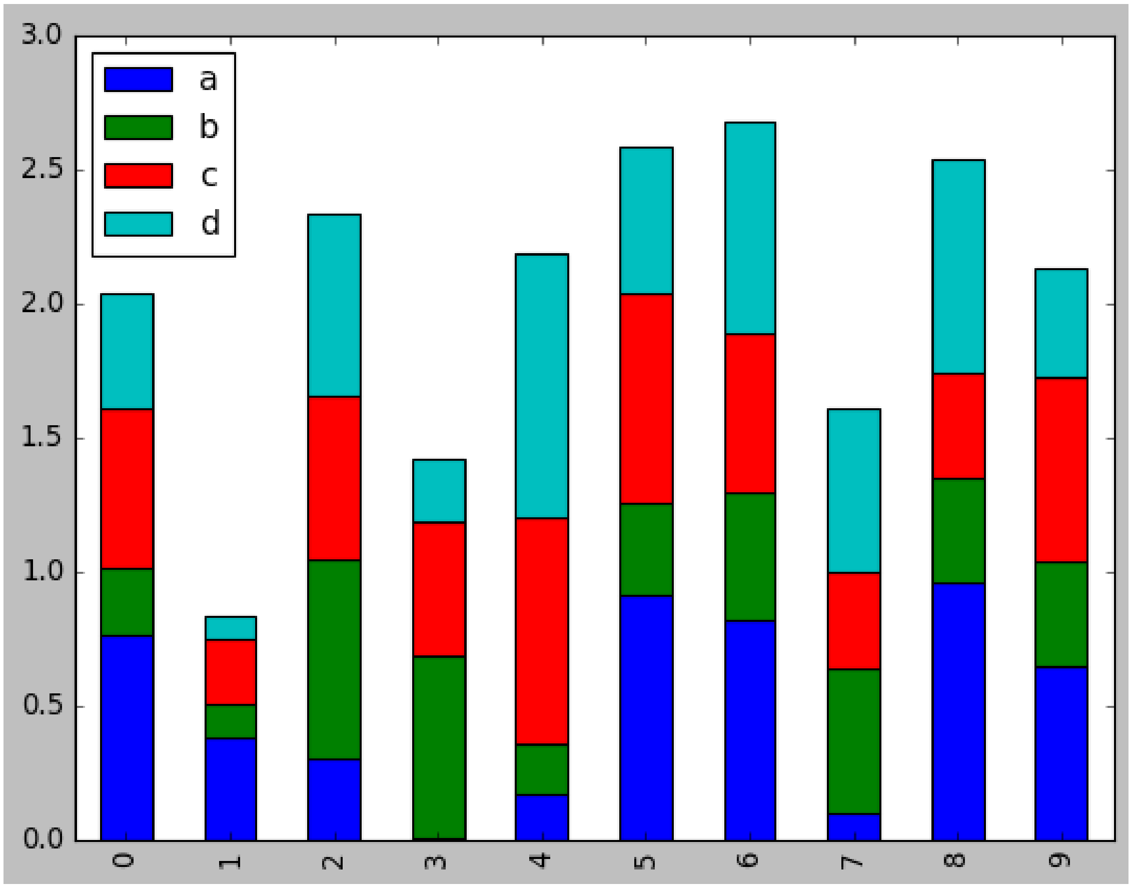
[43]: <matplotlib.axes._subplots.AxesSubplot at 0x1450a88cfc8>



[44]:
```
rand?
```

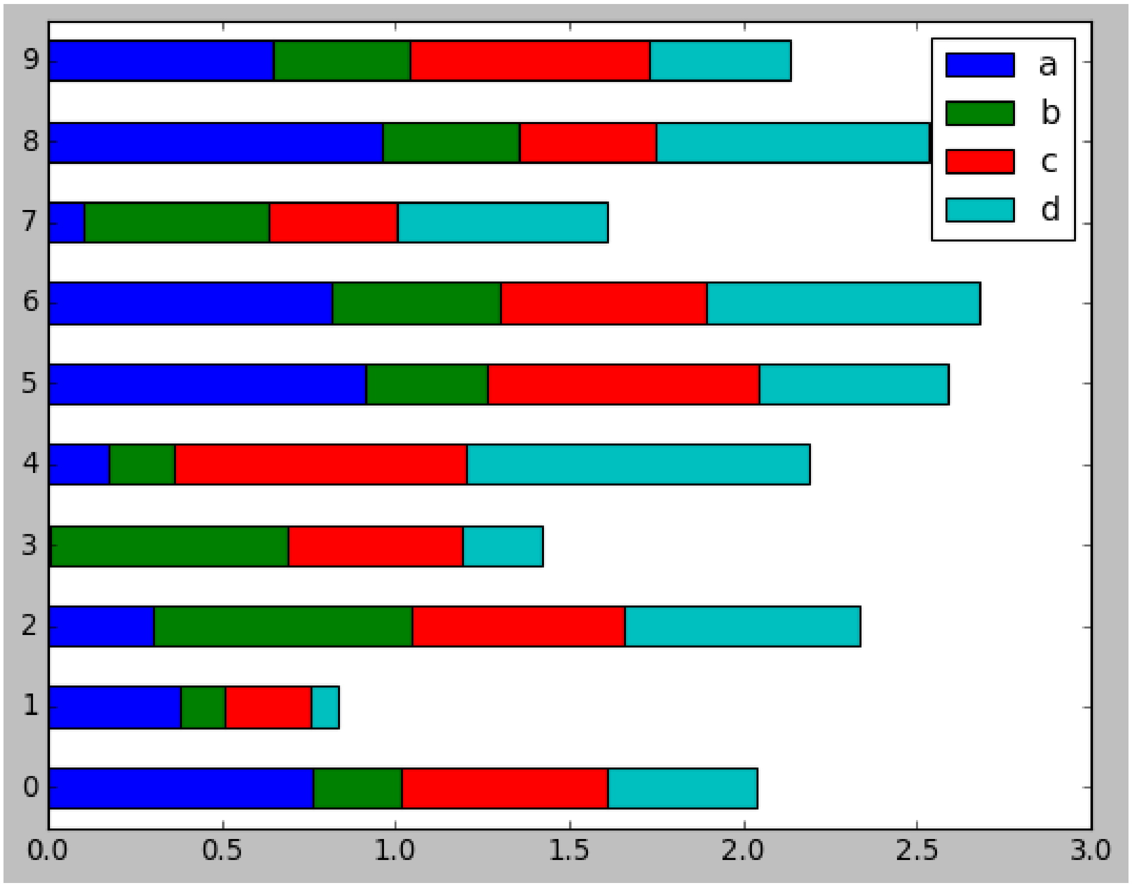
[45]:
```
#stacked
df2.plot(kind='bar',stacked=True, figsize=(8,6))
```

[45]: <matplotlib.axes._subplots.AxesSubplot at 0x1450aaa6748>



[46]:
```
#stacked horizontal
df2.plot(kind='barh',stacked=True)
```

[46]: <matplotlib.axes._subplots.AxesSubplot at 0x1450ad132c8>



Line Graph

Line graphs are used to show value of some items over time. Suppose you need to show pass percentage of students of government schools in the last 5 years. Another example - how sales have changed in the past five years? Let's create a pandas dataframe for the same.

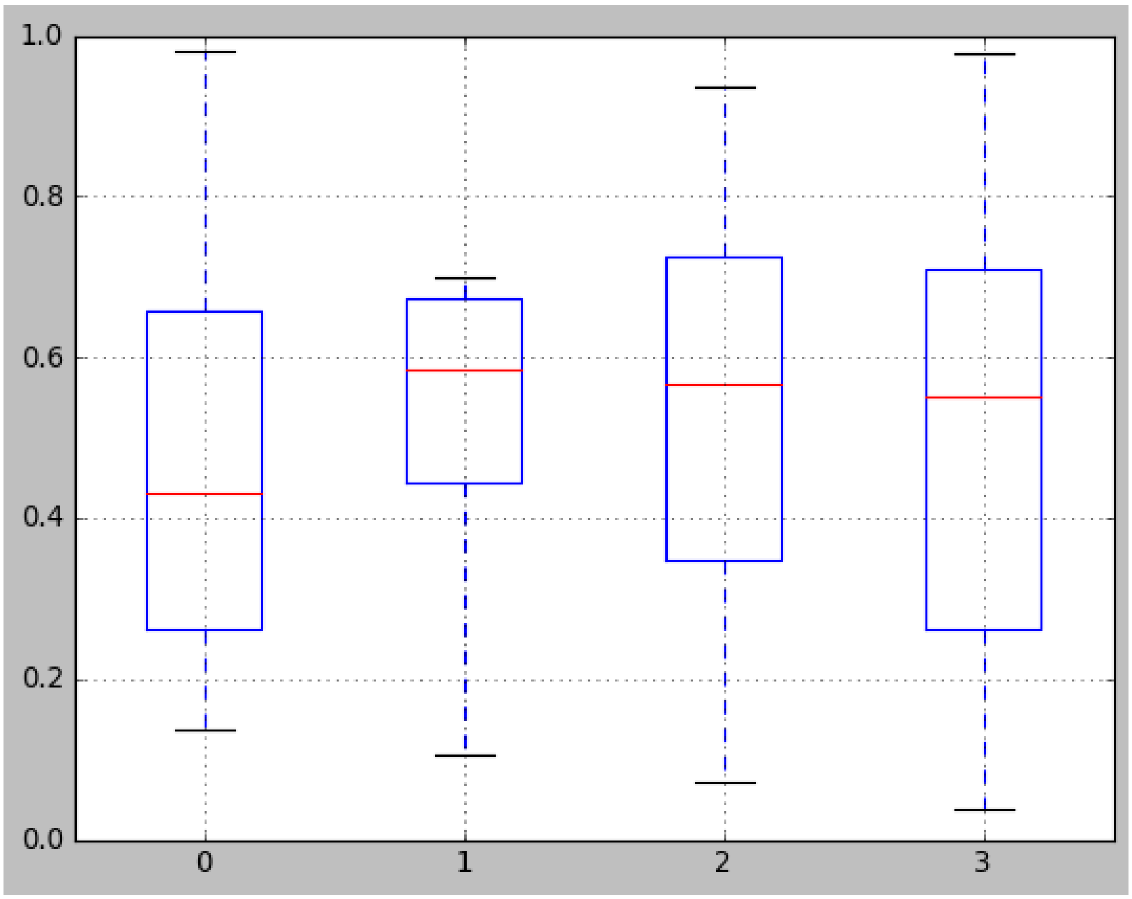
Box Plots

[47]:
```
import pandas as pd
df = pd.DataFrame(rand(10,4))
print (df)
df.boxplot()
```

```
          0         1         2         3
0  0.136113  0.450643  0.334419  0.217969
1  0.925775  0.689639  0.598821  0.746657
2  0.541971  0.700533  0.930560  0.862617
3  0.167442  0.263139  0.071610  0.978673
4  0.236384  0.660598  0.089631  0.210771
5  0.502824  0.104954  0.388873  0.537988
6  0.339204  0.555900  0.536768  0.390610
```

```
7  0.696270  0.678482  0.687303  0.561491
8  0.358080  0.613112  0.935736  0.596889
9  0.979590  0.441007  0.739612  0.037604
```

[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1450aa6f588>



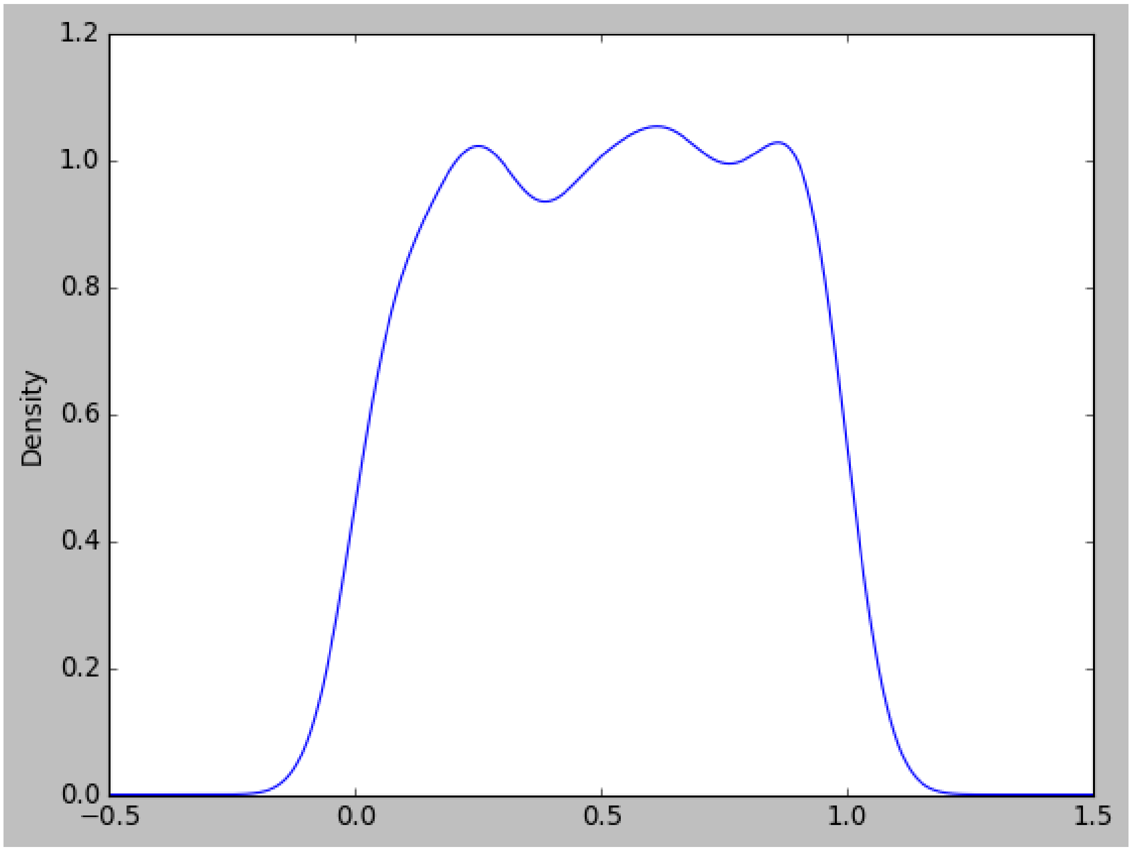
[48]:
```
rand?
```

[49]:
```
boxplot?
```

Density Plots Plot an estimated probability density function (PDE)

[50]:
```
#import pandas as pd
#import numpy as np

ser = pd.Series(np.random.rand(1000))
ser.plot(kind='kde')
```

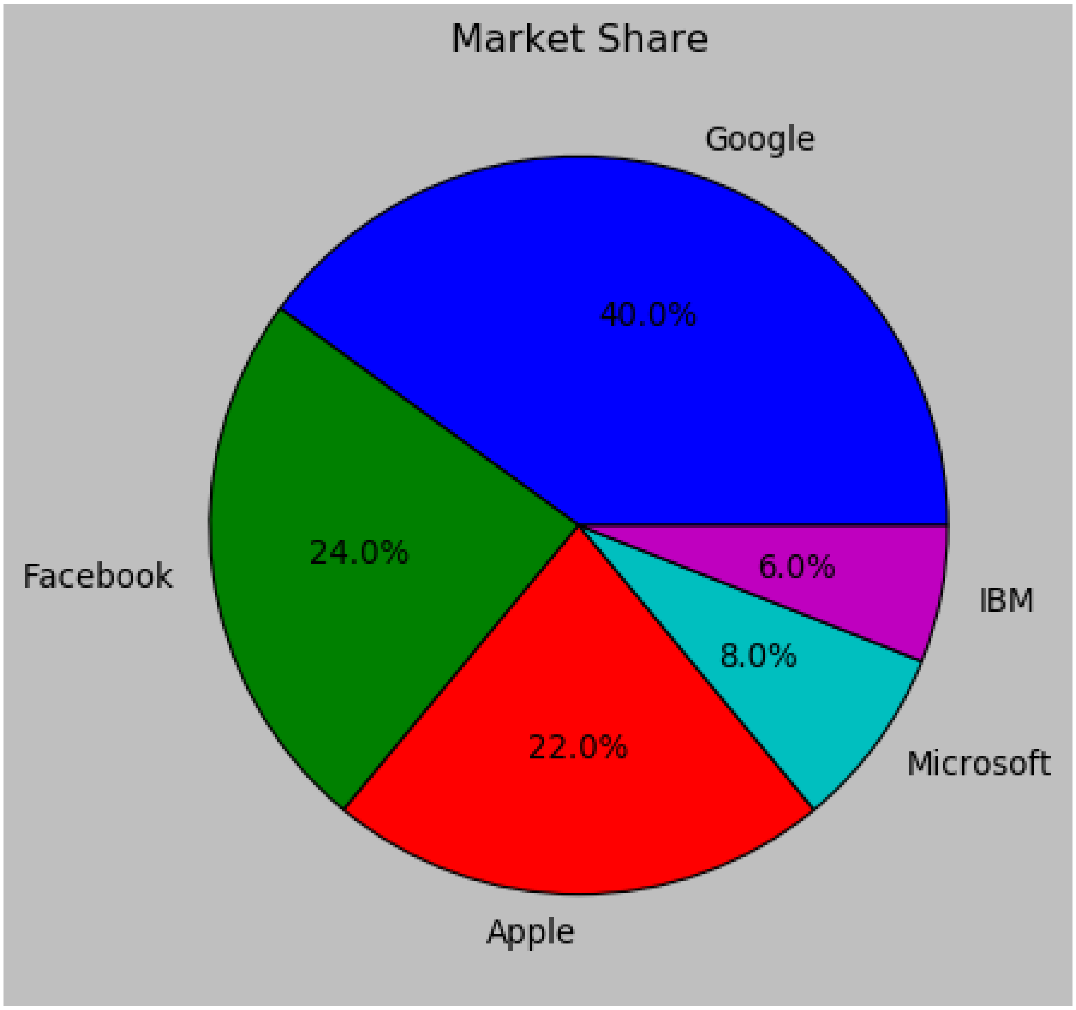
[50]: <matplotlib.axes._subplots.AxesSubplot at 0x1450ad89c88>



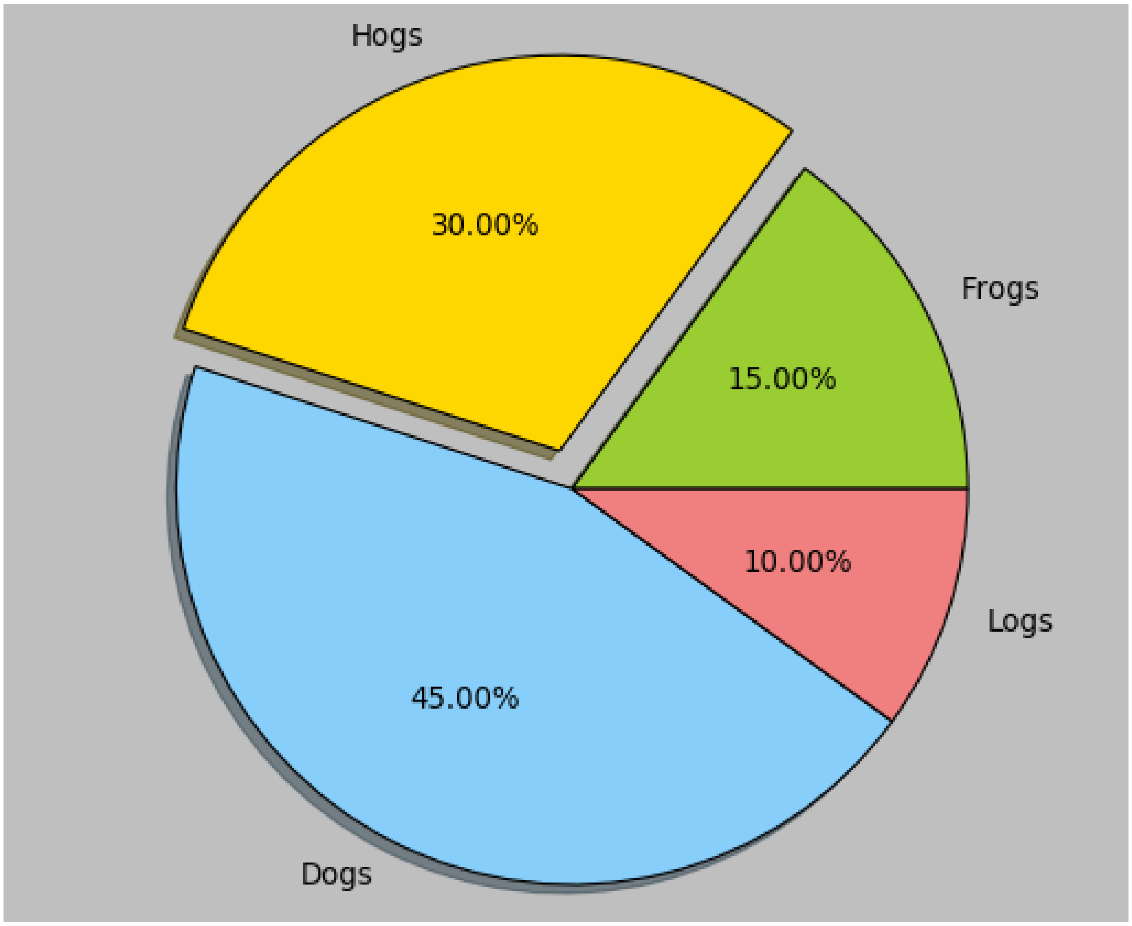
Pie Charts

```
[51]: share = [20, 12, 11, 4, 3]
companies = ['Google', 'Facebook', 'Apple', 'Microsoft', 'IBM']
comp = pd.DataFrame({"share" : share, "companies" : companies})
ax = comp.plot(y="share", kind="pie",
               labels = comp["companies"],
               autopct = '%1.1f%%', legend=False,
               title='Market Share')

# Hide y-axis label
ax.set(ylabel='')
```

```
[51]: [Text(0, 0.5, '')]
```



[52]:
```
# The slices will be ordered and plotted counter-clockwise.
labels = ['Frogs', 'Hogs', 'Dogs', 'Logs']
sizes = [15, 30, 45, 10]
colors = ['yellowgreen', 'gold', 'lightskyblue', 'lightcoral']
explode = [0, 0.1, 0, 0] # only "explode" the 2nd slice (i.e. 'Hogs')

plt.pie(sizes, explode=explode, labels=labels,
        colors=colors, autopct='%2.2f%%',
        shadow=True)
# Set aspect ratio to be equal so that pie is drawn as a circle.
plt.axis('equal')
plt.show()
```

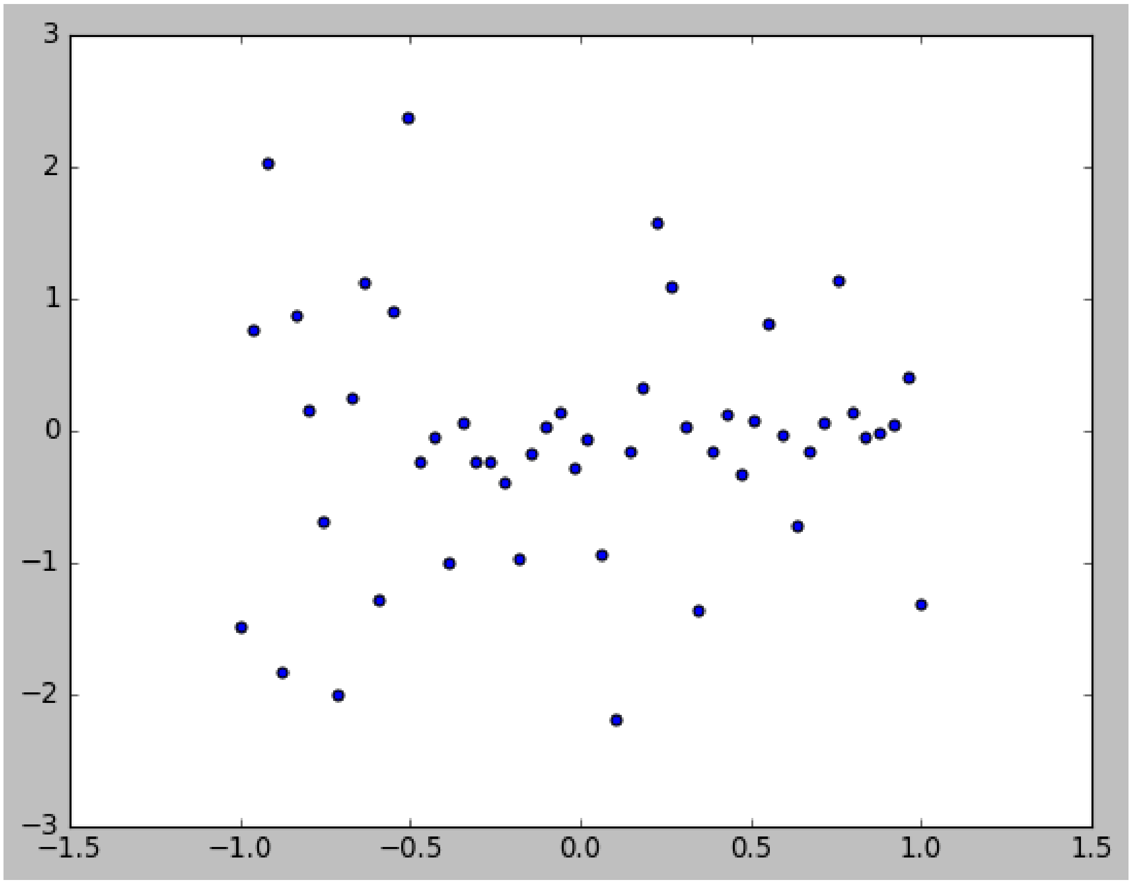


[53]:
```
pie?
```

Scatter plots Scatter plots offer a convenient way to visualize how two numeric values are related in your data. It helps in understanding relationships between multiple variables. Using .scatter() method, we can create a scatter plot:

[54]:
```
fig, ax = plt.subplots()
x = np.linspace(-1, 1, 50)
y = np.random.randn(50)
ax.scatter(x,y)
```

[54]:
```
<matplotlib.collections.PathCollection at 0x1450cb8ed88>
```


3
2
1
0
−1
−2
−3
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5

[ ]: