Lesson3 Plotting with Matplot Last Update Mar 28, 2021

Matplot lets you plot data on a chart. Matplot has many types of charts available:

- line chart
- scatter charts
- bar charts
- histograms

Plotting a simple line chart with Matplot

We will first make a simple line chart. A chart has a x and y axes with x and y values. To use matplot you need to import the matplotlib.pyplot library:

```
import matplotlib.pyplot as plt
```

You may also need to add the **matplotlib** module to your python using the shell:

```
pip install matplot
```

Next you need a set of x and y points. We make 2 lists of x and y points.

```
x = [0,1,2,3,4,5,6,7,8,9]
y = [0,1,2,3,4,5,6,7,8,9]
```

We then add some x and y axes labels, all charts need axes labels.

```
plt.xlabel('x')
plt.ylabel('y')
```

All charts also need a title:

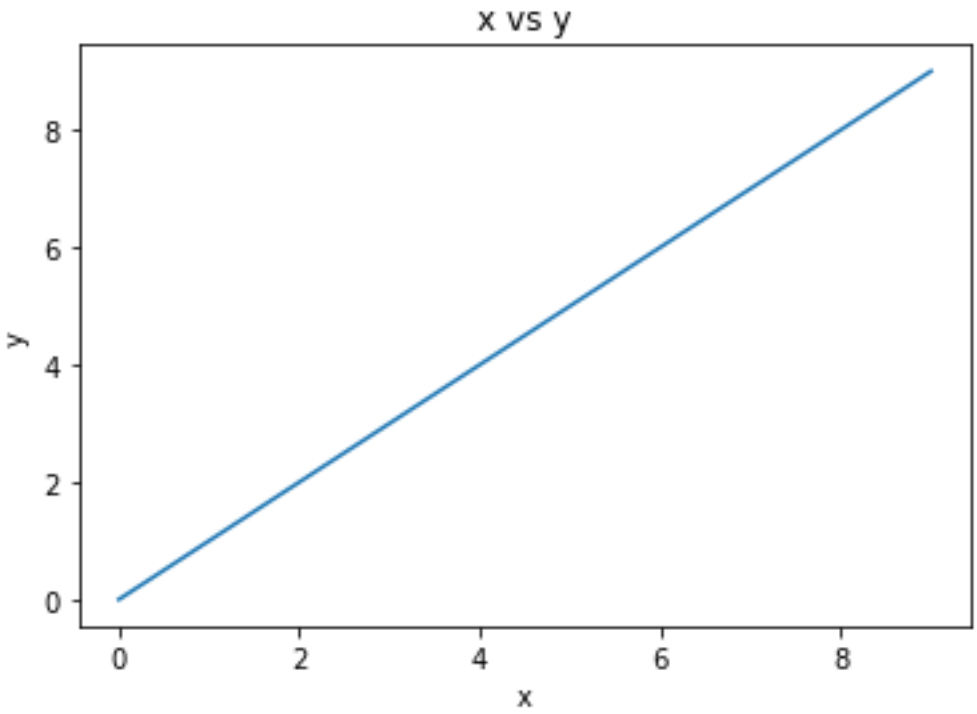
```
plt.title('x vs y')
```

Finally we call the **plot** function, the default chart is a line chart

```
plt.plot(x,y)
```

To show the plot we must call the show function

plt.show()



Here is the complete program:

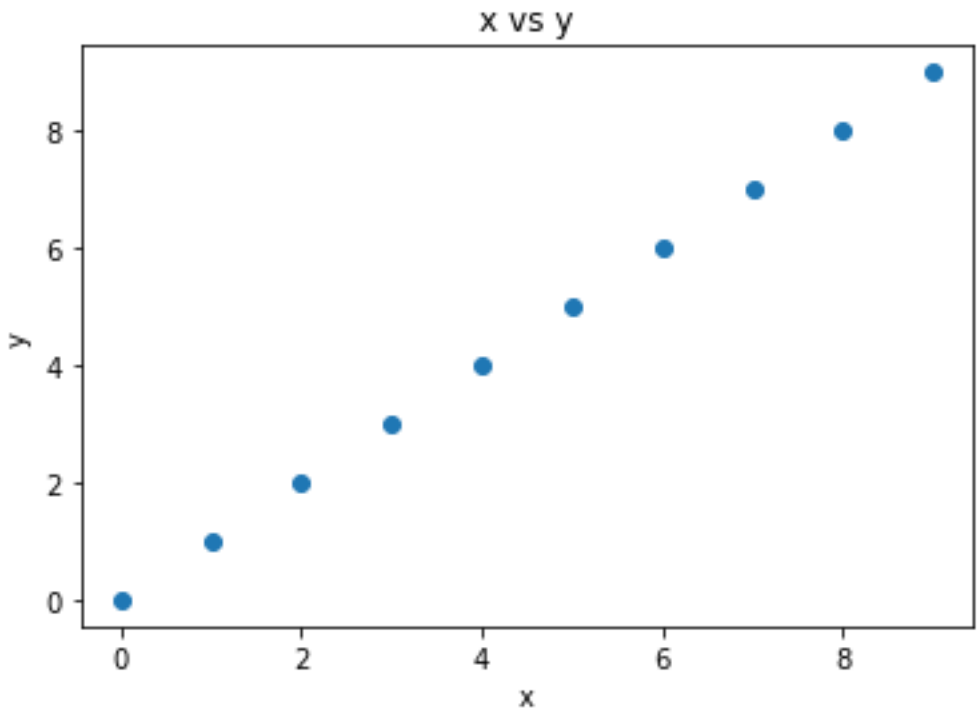
```
# line plot
import matplotlib.pyplot as plt

# make x points
x = [0,1,2,3,4,5,6,7,8,9]

# make y points
y = [0,1,2,3,4,5,6,7,8,9]

# make x-axis labels
plt.xlabel('x')
# make y-axis labels
plt.ylabel('y')
```

```
# add a title
plt.title('x vs y')

# plot line chart
plt.plot(x,y)

# show plot
plt.show()
```

We can plot dots instead of lines using a third **fmt** parameter with a value of 'o'

Here is the complete program:

```
# dot plot
import matplotlib.pyplot as plt

x = [0,1,2,3,4,5,6,7,8,9]
y = [0,1,2,3,4,5,6,7,8,9]

plt.xlabel('x')
plt.ylabel('y')
plt.title('x vs y')
plt.plot(x,y,'o')
plt.show()
```

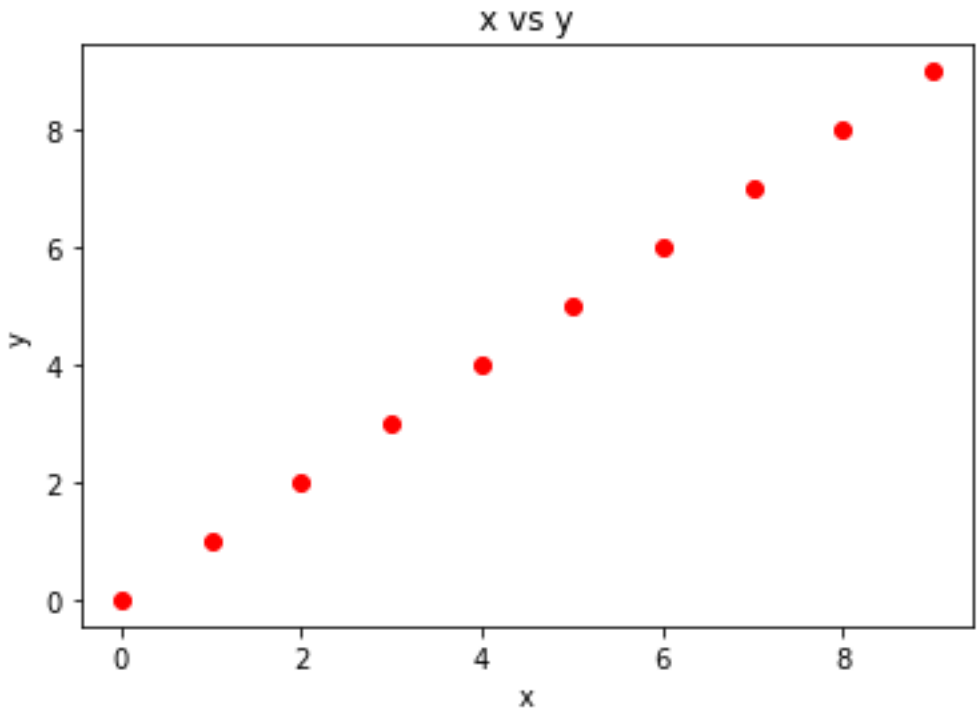


You can also change the color of the dots by specifying a color letter b,g,r,c,m,y,k or u the default color is b (blue). Each letter specifies a common color.

We specify red dots **'ro'** where r = 'red' o = 'dots

plt.plot(x,y,'ro')

We can now have a chart with red dots.



To do:

Change the chart to green dots or try some of the other colours.

Changing the length and width of a chart

You can change the length and width of a chart using

plt.figure(figsize=(10,5))

where the first number is width in inches and the second number is height in inches. The above sets a plot figure of 10 width by 5 height inches.

Here is the complete program:

```
# large dot plot
import matplotlib.pyplot as plt
x = [0,1,2,3,4,5,6,7,8,9]
y = [0,1,2,3,4,5,6,7,8,9]

# change plot size 10 wide by 5 inches high
plt.figure(figsize=(10,5))
plt.xlabel('x')
plt.ylabel('y')
plt.title('x vs y')
plt.plot(x,y,'ro')
plt.show()
```

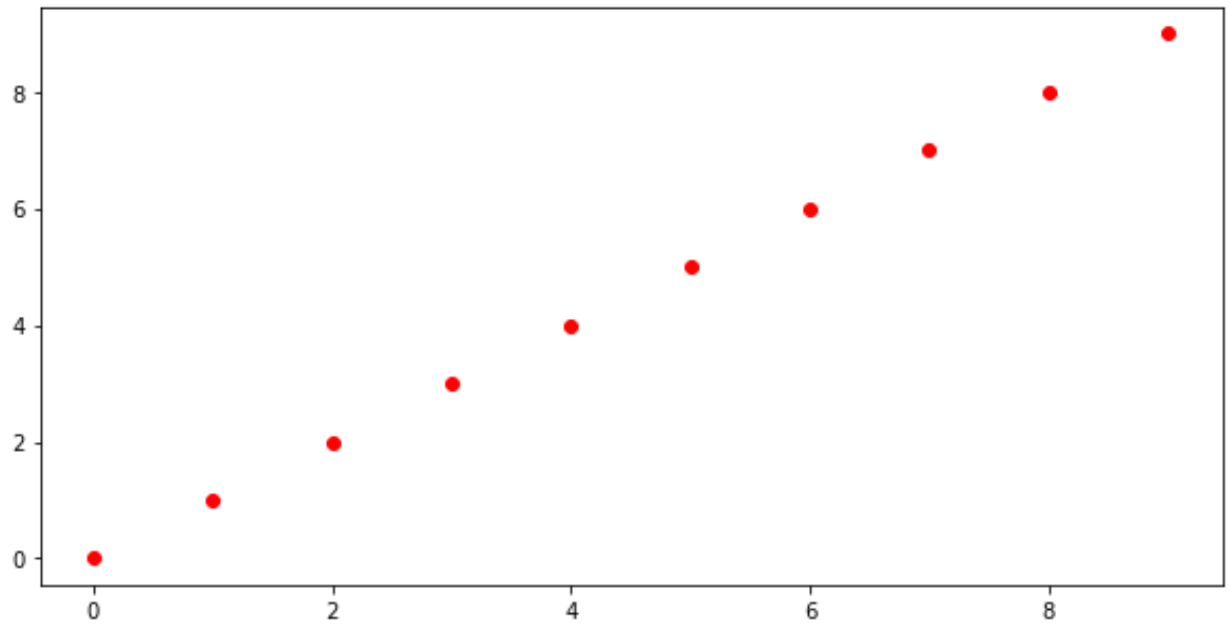


chart fmt specifiers:

The fmt specifier lets you specify a chart with a **marker**, **type** of line and line **color**. The **markers** can be dots, circles triangles etc. The **line style** may solid, dashes or dots. The **colors** are one of b,g,r,c,m,y,k or u.

fmt = '[marker][line][color]'

Here are all of all markers, line styles and colors for your reference:

Line Styles

character	Description
'-'	solid line style
'--'	dashed line style
'-.'	dash-dot line style
':'	dotted line style

Colors

character	Color
'b'	Blue
'g'	Green
'r'	Red
'c'	Cyan
'm'	Magenta
'y'	Yellow
'k'	Black
'w'	White

Markers

character	Description
'.'	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 marke

Example format styles:

Format	Example
'b'	blue markers with default shape
'or'	red circles
'-g'	green solid line
'--'	dashed line with default color
'^k:'	black triangle_up markers connected by a dotted line

Plotting more than one line on a chart

We can plot more than 1 line on a chart with a succession of plot calls. We will plot the following mathematical functions on a chart:

f(x) = x
f(x) = log (x)
f(x) = x ^2
f(x) = x log (x)

where y = f(x)

We use the numpy **arrange** function to make values for the x axes where as the y values would be one of our f(x) functions.

The numpy **arrange** function returns evenly spaced stop and start values within a given interval.

numpy.arange([start,]stop, [step,]dtype=None)
x = np.arange(.1, 10, 0.1)

We set the y-axis values to the result of each function

```
# y axes formulas
y1 = x
y2 = [math.log(x) for x in x]
y3 = x**2
y4 = [x * math.log(x) for x in x]
```

We can add labels to our chart using the **label** parameter and the function **title.**

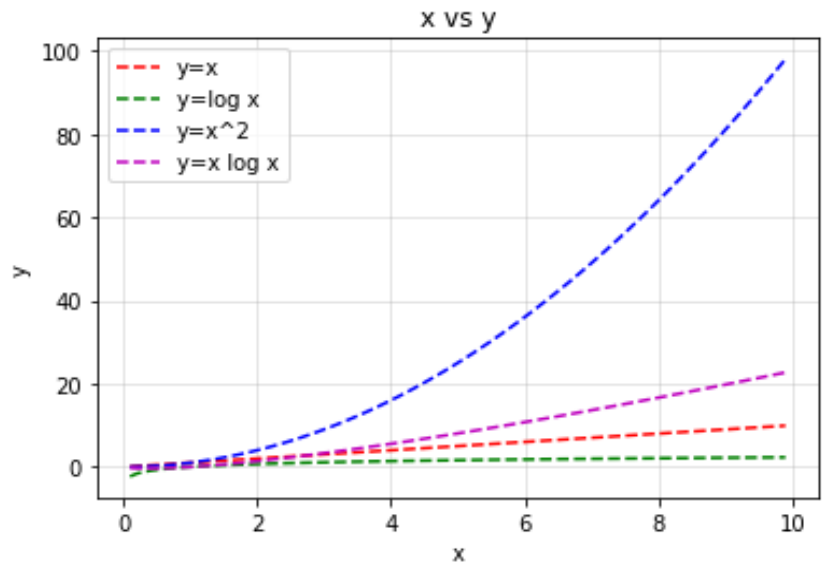
```
# add label
plt.plot(x, y1,'r--', label='y=x')
plt.plot(x, y2,'g--', label='y=log x')
plt.plot(x, y3,'b--', label='y=x^2')
plt.plot(x, y4,'m--', label='y=x log x')
```

We use the grid function to show a grid with a specified grid alpha level and line style**.** The alpha level specifies the grid brightness level. We want our grid lines to be solid and dim.

```
# add a grid
plt.grid(alpha=.4,linestyle='-')
```

We use the **legend** function to show the legend accordingly to our specified function labels

```
# add a Legend
plt.legend()

# show chart
plt.show()
```



Here is the complete program:

```
# multi-line plot
import matplotlib.pyplot as plt
import math
import numpy as np

# x-axes
# .1 to 10 by .1 intervals
x = np.arange(.1, 10, 0.1)

# y axes formulas
y1 = x
y2 = [math.log(x) for x in x]
y3 = x**2
y4 = [x * math.log(x) for x in x]

# add label
plt.plot(x, y1,'r--', label='y=x')
plt.plot(x, y2,'g--', label='y=log x')
plt.plot(x, y3,'b--', label='y=x^2')
plt.plot(x, y4,'m--', label='y=x log x')
```

```
# add a grid
plt.grid(alpha=.4,linestyle='-')

# add a Legend
plt.legend()

# show chart
plt.show()
```

Scatter plots

Each dot in the scatter plot represents a x, y coordinate. For this example we use the **numpy.random.normal** function to generate random x and y points.

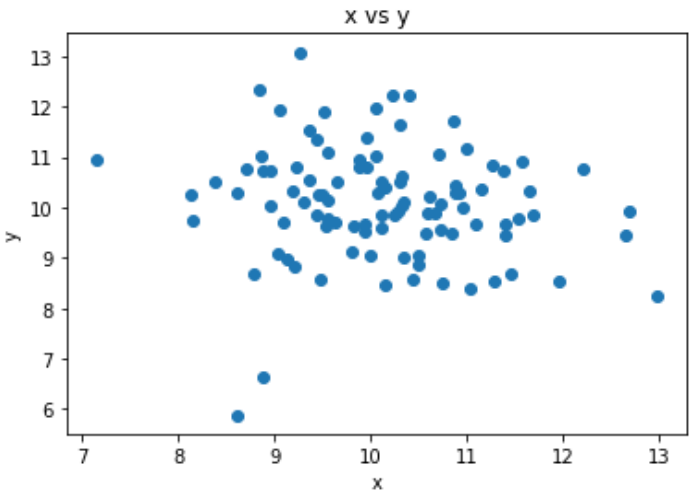
numpy.random.normal(mean, standard deviation, size)

numpy.random.normal(10, 1.0, 100)

We generate 100 numbers having a center of 10 and deviation of 1. Deviation means the x and y random values will deviate between center value by 1

Here is the complete program:

```
# scatter plot
import numpy
import matplotlib.pyplot as plt

# make x,y points
x = numpy.random.normal(10.0, 1.0, 100)
y = numpy.random.normal(10.0, 1.0, 100)

plt.scatter(x, y,c='g',marker='o')
plt.title('scatter plot')
plt.show()
```

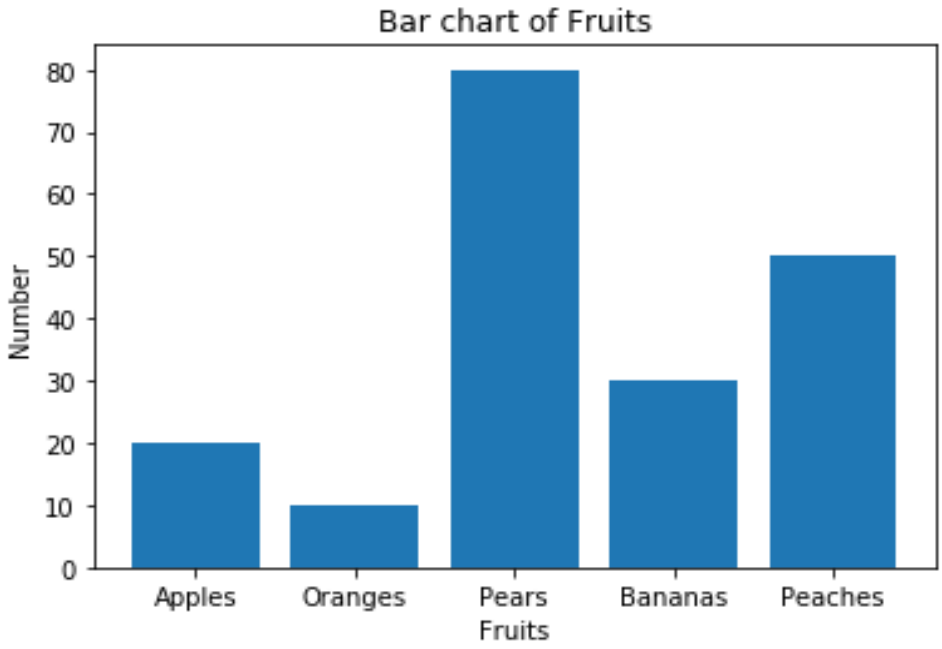


Bar chart

A bar chart display values as vertical or horizontal bars. 2 arrays are needed 1 for the bar name's and the other one for the bar values. You make a bar chart with the matplot **bar** function.

Here is the bar chart complete program:

```
# bar chart of fruits
import matplotlib.pyplot as plt
names = ['Apples', 'Oranges', 'Pears','Bananas', 'Peaches']
values = [20, 10, 80, 30, 50]
plt.bar(names, values)
plt.title('Bar chart of fruits')
plt.xlabel('names')
plt.ylabel('values')
plt.show()
```



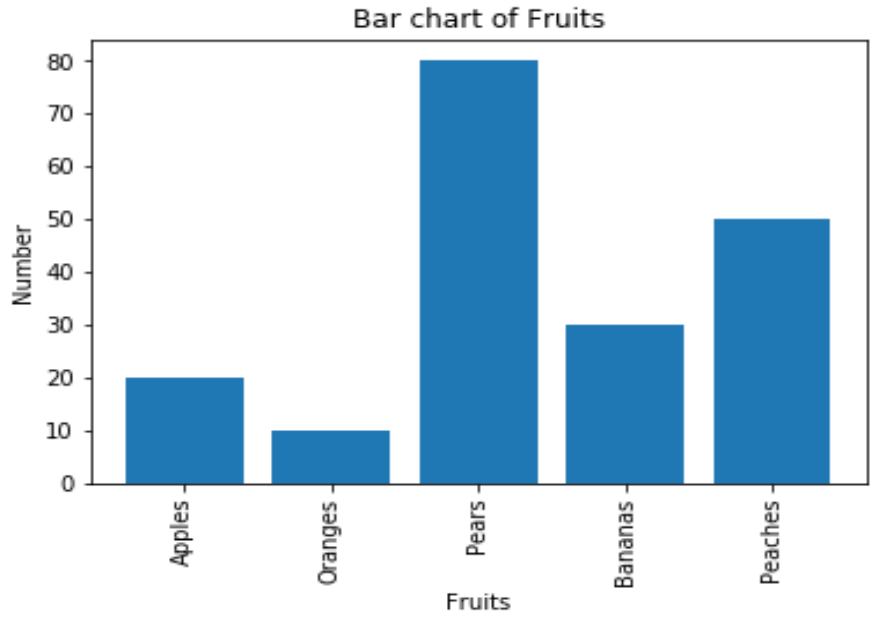
Printing label names vertically

You can use the **xticks** function to rotate the names label by a certain angle like 90 degrees.

```
plt.xticks(rotation=90)
```

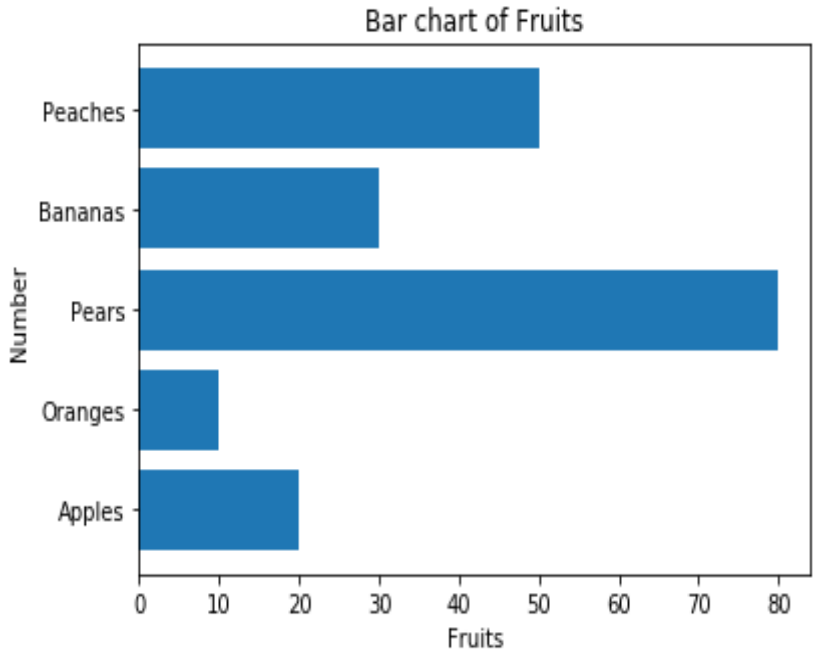
Here is the complete program with rotated xticks:

```
# bar chart of fruits
import matplotlib.pyplot as plt

names = ['Apples', 'Oranges', 'Pears','Bananas', 'Peaches']
values = [20, 10, 80, 30, 50]
plt.bar(names, values)
# rotate xticks 90 degrees
plt.xticks(rotation=90)
plt.title('Bar chart of fruits')
plt.xlabel('names')
plt.ylabel('values')
plt.show()
```



Horizontal Bar Chart

You can use **plt.barh** to plot a horizontal bar chart



Potting a Bell Curve

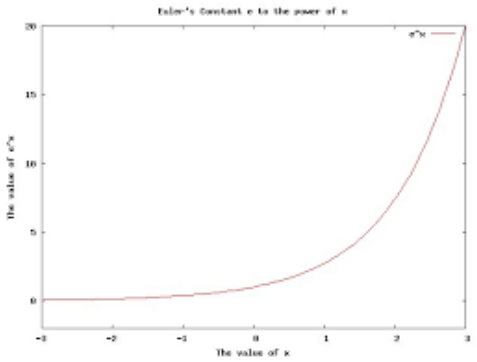
The formula for normal distribution is as follows:

$$y = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$\mu =$ Mean
$\sigma =$ Standard Deviation
$\pi \approx 3.14159\cdots$
$e \approx 2.71828\cdots$

The **μ mean** is the center where the **σ standard deviation** specifies the spread from the center, ***π*** **pi** is defined as the ratio between a circle's circumferencen to its diameter and the letter ***e*** represents the mathematical constant **Euler's** number which is approximately 2.71828 calculated from the sum of $(1 + 1/n)^n$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{1} + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \cdots$$



A particular normal distribution is completely determined by the **mean** and **standard deviation**. The **variance** is the square of standard deviation.

The above formula is used for calculating probabilities that are related to a normal distribution. (**probability density function** pdf). We can calculate the normal distributions for mean and standard deviation and plot on a chart as follows:

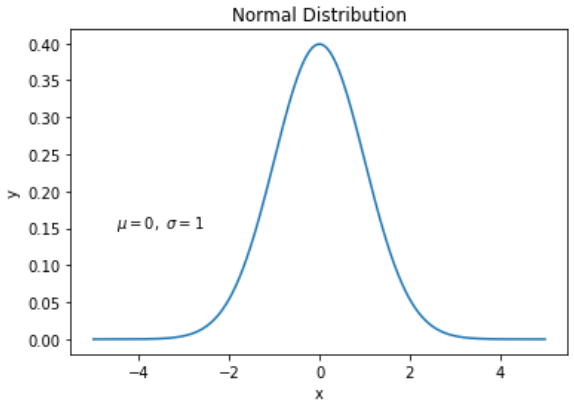
```
# plotting bell curve
import matplotlib.pyplot as plt
mean = 0
std = 1
variance = np.square(std)
x = np.arange(-5,5,.01)
y = np.exp(-np.square(x-mean)/2*variance)/(np.sqrt(2*np.pi*variance))
plt.plot(x,y)
plt.xlabel('x')
plt.ylabel('gaussian distribution')
plt.title('Bell Curve')
plt.show()
```

We can also write text on chart using matplot **text** function at specified x and y coordinates

```
plt.text(-4.5, .15, r'$\mu=0,\ \sigma=1$')
```

-4.5 is the x coordinate .15 is the y coordinate to position the text and **'\mu'** is the mean character μ and **'\sigma'** is the std character σ and **'\ '** is a space

The while expression is enclosed in a **r'** that signifies that the string is a *raw* string. In this case the backslashes are converted to matplot back slashes for plotting special characters μ (mean) and σ (std deviation) on the chart.



to do: try different values of mean and std

Histogram

A histogram is similar to a bar chart but displays the y values sequentially as filled in bars. Each bar on a histogram displays the number or count of samples that have the same values. The histogram displays frequencies rather than values.

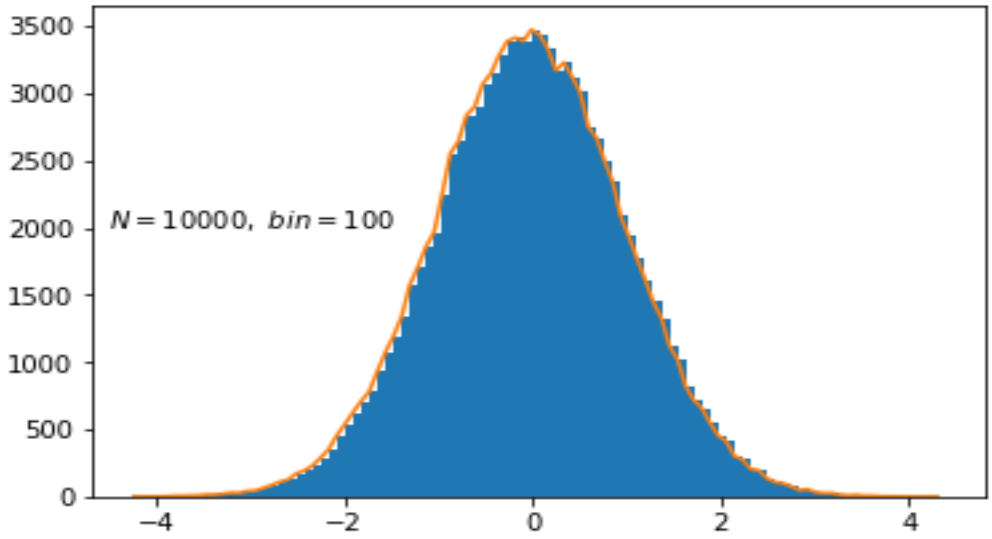
Our histogram example plots the normal distribution of N random points for a specified number of bins. Our normal distribution is a sequence of random values. We use the numpy **randn** function to Return a sample samples from the standard normal distribution.

```
x = np.random.randn(N)
```

We then call the matplot **hist** function to plot the histogram. The **hist** function also returns the calculated y values and x bin values so we can plot a curve around the histogram. The returned y value length are n-1 of the bin values

Here is the complete program:

```
# plotting histogram
import matplotlib.pyplot as plt

N = 100000
bins = 100
x = np.random.randn(N)
y,x,p= plt.hist(x, bins) # plot histogram
plt.title('Histogram')
plt.plot(x[:-1],y) # plot a orange curve around the histogram
plt.show()
```



The peak of the histogram is at the center because the majority of the random numbers are centered around 0. The mean is at 0 and the standard deviation (spread) is 1.

To do:

Try different N and bin values.

Plot a Histogram with a Normal Distribution curve

We will use the scipy module , python scientific programming module.

pip install scipy

We first need to import the **norm** function from scipy.stats

```
from scipy.stats import norm
import matplotlib.pyplot as plt
```

We then generate 1000 samples of center 50 with deviation of 5

```
sample = np.random.normal(loc=50, scale=5, size=1000)
```

We then calculate the mean and std of the sample

```
sample_mean = np.mean(sample)
sample_std = np.std(sample)
```

From the **sample mean** and **std** we use scipy **stats** function to calculate the normal distribution

```
dist = norm(sample_mean, sample_std)
```

Using list compression we make a list of all values between 30 and 70

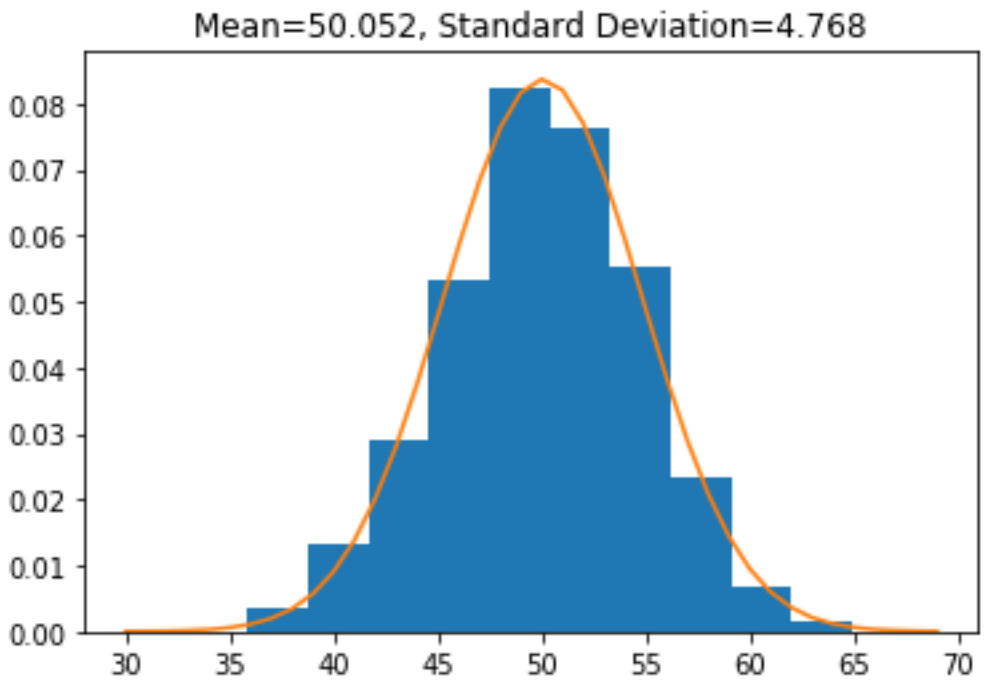
```
values = [value for value in range(30, 70)]
```

Using the distribution we calculate the **probability density function (pdf)** for each value in values. The probability density function (pdf) represents a continuous probability distribution.

```
probabilities = [dist.pdf(value) for value in values]
```

We can now plot the pdf probabilities:

```
plt.hist(sample, bins=10, density=True)
plt.plot(values, probabilities)
plt.title('Mean=%.3f, Standard Deviation=%.3f' % (sample_mean, sample_std))
plt.show()
```



Here is the complete program:

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt

# generate 1000 samples of center 50 with deviation of 5
sample = np.random.normal(loc=50, scale=5, size=1000)

#calculate the mean and std  of the sample
sample_mean = np.mean(sample)
sample_std = np.std(sample)

#calculate the normal distribution
dist = norm(sample_mean, sample_std)

#make a list of all values between 30 and 70
values = [value for value in range(30, 70)]

# calculate probabilities of the pdf using dist distribution
probabilities = [dist.pdf(value) for value in values]
```

```
# plot the pdf probabilities:
plt.hist(sample, bins=10, density=True)
plt.plot(values, probabilities)
plt.title('Mean=%.3f, Standard Deviation=%.3f' % (sample_mean, sample_std))
plt.show()
```

Todo: change the sample parameters and see what happens.

Difference between Bar Chart and Histogram

Histograms are used to show frequency distributions of variables while **bar charts** are used to show values for comparisons. **Histograms plot** quantitative data with ranges of the data grouped into bins or intervals while **bar charts plot** categorical data.

A good example is the value of each coin and amount of coins. Each coin has a value 1 cent, 5 cent, 10 cents, 25 cents and 1 dollar. Where you may have 26 pennies, 14 nickels, 23 dimes, 34 quarters, 45 half dollars and 28 dollar coins.

The Coin Bar Chart will show the amounts of the coins in the y axis and the number of coins in the x axis. In the Coin Histogram the x axis will show the amounts and the y axis will shows the frequencies.

Bar Chart of Coins

Coins by Amount

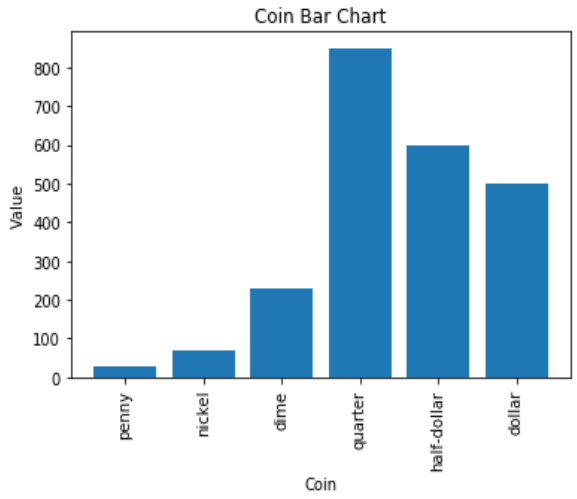
Coin	Number Coins	Total Amount
penny	26	26
nickel	14	70
dime	23	230
quarter	34	850
Half Dollar	12	600
Dollar	5	500

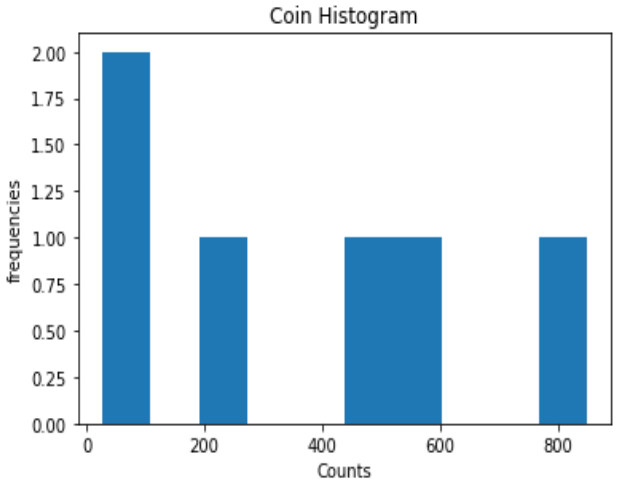
We can now group by frequencies

Histogram of Coins

Range	Number Coins	Coins
0-100	2	Penny, Nickel
100-200	1	
200-300	1	Dime
300-400	0	
400-500	0	
500-600	1	Dollar
600-700	1	Half Dollar
700-800	0	
800-900	1	Quarter
900-1000	0	

Here are the Charts:





Here is the codes:

```
# bar chart of coin values
names = ['penny', 'nickel', 'dime','quarter', 'half-dollar','dollar']
values = [26,14*5,23*10,34*25,12*50, 5*100]

plt.bar(names, values)
plt.title('Coin Bar Chart')
plt.xlabel('Coin')
plt.ylabel('Value')
plt.xticks(rotation=90)
plt.show()

# plot histogram of coin values
x = [26,14*5,23*10,34*25,12*50, 5*100]

plt.hist(x)
plt.title('Coin Histogram')
plt.xlabel('Counts')
plt.ylabel('frequencies')

plt.show()
```

SUBPLOTS

Figures contain axes and subplots

Each figure can contain many axes and subplots:

The subplot() command specifies number rows, number columns and plot_number where plot_numbers range from 1 to number rows*number columns. The commas in the subplot command are optional if number rows * number columns < 10. So subplot(211) is identical to subplot(2, 1, 1).

211 = 2 rows, 1 column a plot number 1

Subplots can be vertical, horizontal or grid formations

Horizontal sub plots

Horizontal subplots have row number equal to 1

First we make x and y axes data

```
# horizontal subplots
x = range(10)
y = range(10)
```

then we make a figure

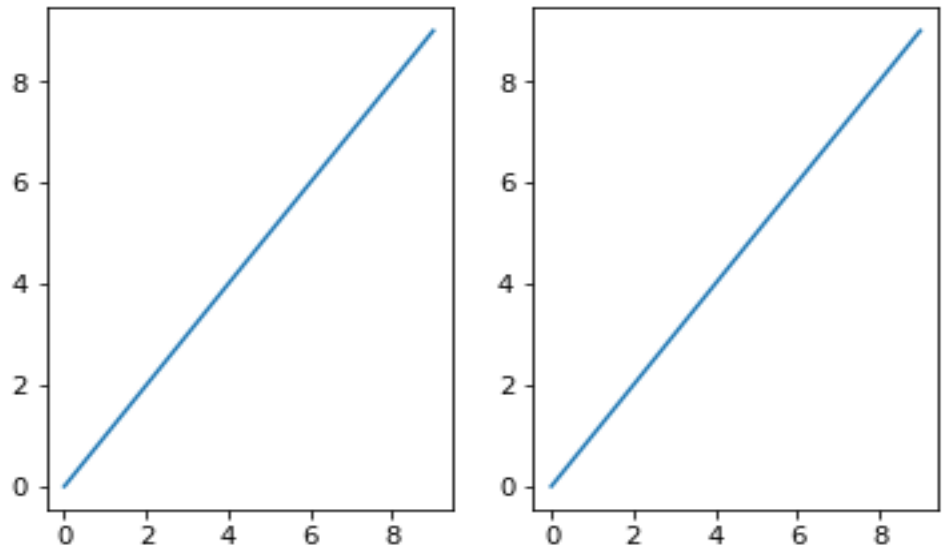
```
plt.figure()
```

we then specify 1 row 2 columns and plot #1 (121)

```
plt.subplot(121)
plt.plot(x,y)
```

we then specify 1 row 2 columns and plot #2 (122)

```
plt.subplot(122)
plt.plot(x,y)
plt.show()
```



Vertical sub plots

Vertical subplots have column number equal to 1

First make x and y axes data

```
# vertical subplots
x = range(10)
y = range(10)
```

then we make a figure

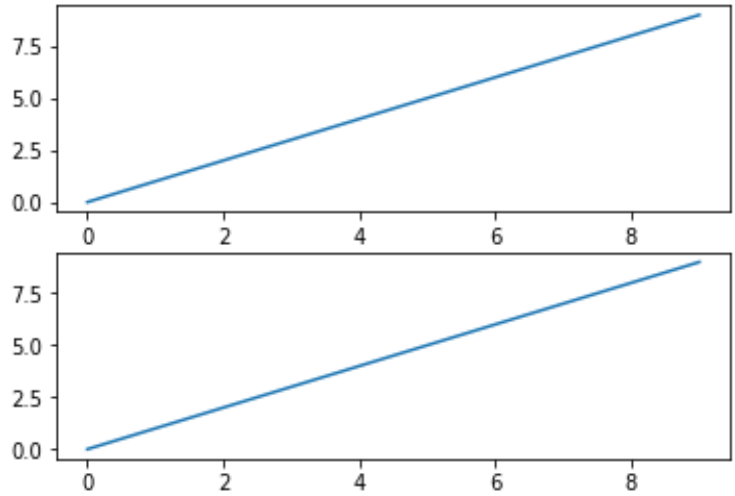
```
plt.figure()
```

we then specify 2 rows and 1 columns and plot #1 (211)

```
plt.subplot(211)
plt.plot(x,y)
```

we then specify 2 rows 1 columns and plot #2 (212)

```
plt.subplot(212)
plt.plot(x,y)
plt.show()
```

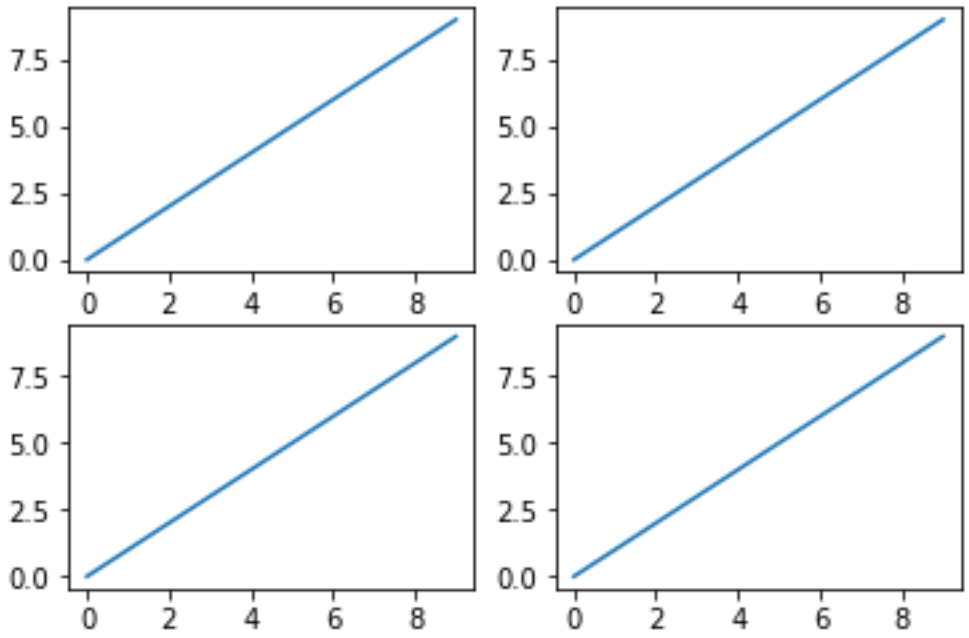


Grid subplots

Grid subplots have equal number or rows and columns

Here is the Grid plot sample program: (2 rows and 2 columns)

```
# grid plot
x = range(10)
y = range(10)
plt.figure()
plt.subplot(221)
plt.plot(x,y)
plt.subplot(222)
plt.plot(x,y)
plt.subplot(223)
plt.plot(x,y)
plt.subplot(224)
plt.plot(x,y)
plt.show()
```

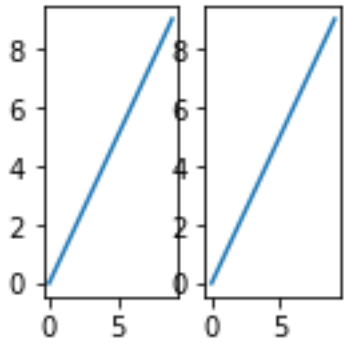


Using axes

The **axes** object contains the methods for plotting, as well as most customization options, while the **figure** object stores all of the figure-level attributes and allow the plot to output as an image. It is good to know how to use the **axes** object.

The **subplot** function returns the **figure** and the **axes** list. From the **axes** list we can do our plots. We can also use the **figure** object to set the size of the plot using the **set_size_inches** function.

```
# using axes
x = range(10)
y = range(10)

fig, ax = plt.subplots(nrows=1, ncols=2)
fig.set_size_inches(2, 2)

ax[0].plot(x,y)
ax[1].plot(x,y)
plt.show()
```



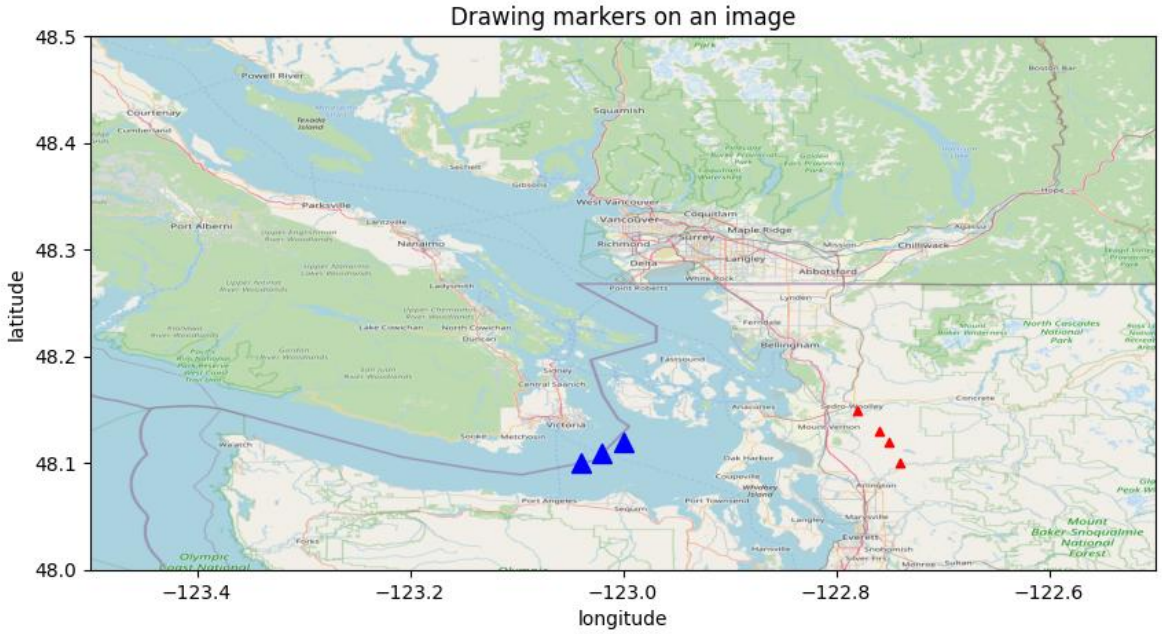
Drawing markers on an Image

You may want to draw markers on an image or even a scatter plot or regression line. You first load the image then draw on it. You can then save the image back to the image file using **plt.savefig(filename).** We set the plot coordinates to the latitude and longitude of the map using the **set_xlim** and **set_ylim** plot functions.

```
# drawing markers on an image

# load image small map
filename = "map1.png"
image = plt.imread(filename)

#make subplots
fig, ax = plt.subplots(figsize = (10,10))

#set map bounds
min_long = -123.5
max_long = -122.5
min_lat = 48
max_lat = 48.5
#put map bounds in rectangle (bounding box)
BBox = (min_long,max_long,min_lat, max_lat)

# draw some markers
ax.plot(-123.0, 48.12, 'b^',markersize=10)
ax.plot(-123.02, 48.11, 'b^',markersize=10)
ax.plot(-123.04, 48.1, 'b^',markersize=10)


# plot markers as a scatter plot
lats = [48.1,48.12, 48.13,481.4, 48.15]
longs =[-122.74,-122.75,-122.76,-122.77,-122.78]
ax.scatter(longs,lats,c='r',marker='^', s=20)

# we use set_xlim to set the min and max longitude coordinates  on the map
# we use set_ylim to set the min and max latitude coordinates  on the map
ax.set_xlim(BBox[0],BBox[1])
ax.set_ylim(BBox[2],BBox[3])

#set labels, and title
ax.set_title('Markers on a Map')
ax.set_xlabel("longitude")
ax.set_ylabel("latitude")
```

 # show and plot image
ax.imshow(image, zorder=0, extent = BBox, aspect= 'equal')
 # save image back to file
plt.savefig("new"+filename)
plt.show()

Our map image with some markers and scatter plot drawn on it:



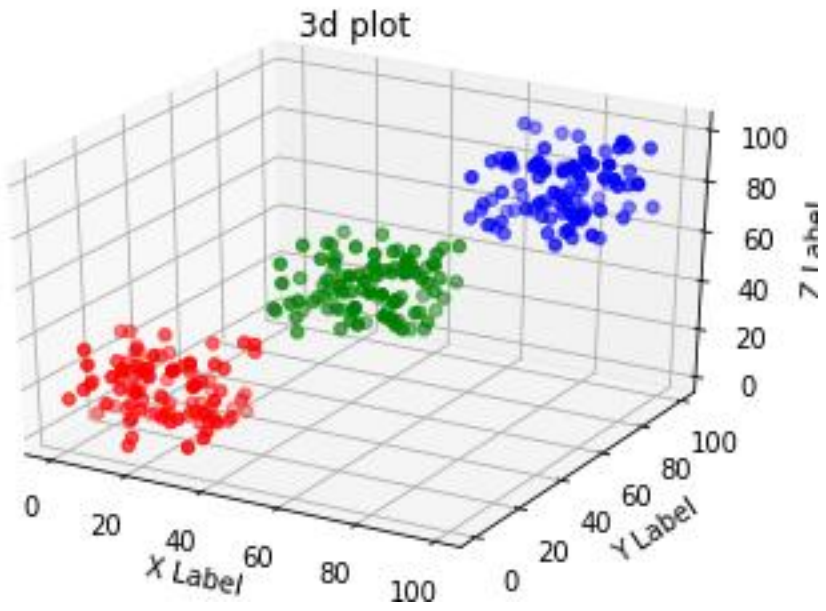
3D Scatter Plot

With matplot we can also plot 3D Scatter plot. We must first import the axes3d library

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

 # make a figure and subplot
fig = plt.figure()

```
# retrieve the axes from the subplot to do the 3d plot.
ax = fig.add_subplot(111, projection='3d')

# plot 3 color groups each having 100 points each:
n = 100


# assign color , and low high values for 3 groups
for c,low,high in [('r', 0, 33), ('g', 33, 66), ('b',66,100)]:
   # calculate x random value between low and high
   x = (high-low)*np.random.rand(n)+low
   # calculate y random value between low and high
   y = (high-low)*np.random.rand(n)+low

   # calculate random z value between low and high
   z = (high-low)*np.random.rand(n)+low

   # plot point x,y,z
   ax.scatter(x, y, z, c=c, marker='o')

# plot lables and title
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
ax.set_title('3d plot')

# plot 2d plot
plt.show()
```



HOMEWORK

Question 1

Make a 2x2 Grid plot, put a scatter plot in the first grid, a line chart on the second grid, a bar chart on the third grid and a histograms in the last grid.

You can use random or fixed data. In one chart use numpy **arrange** function for the x axis in another chart use numpy **linspace** function for the x axis.

Use excerpts from out numpy lesson:

arrange function

#create an array with a sequence of 0 to 18 by step of 2
x = np.arange(0, 20, 2)

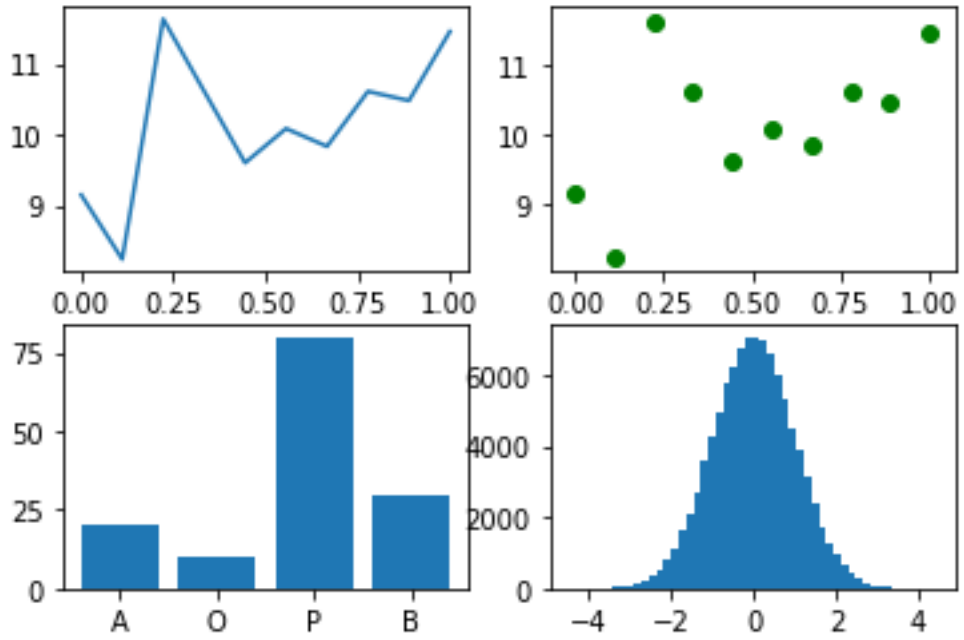
[0, 2, 4, 6, 8,10,12,14,16,18]

linspace function

#create an array of even space between a given range of values
 # (divides 0 to 1 evenly by 5)
x = np.linspace(0, 1, 5)

```
([ 0., 0.25, 0.5 , 0.75, 1.])
```

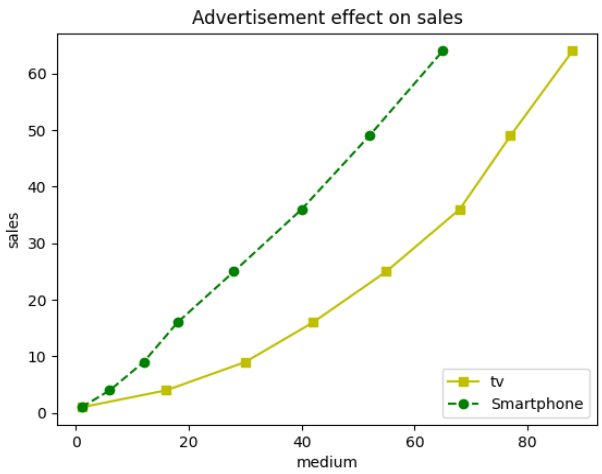
You should get something like this:



Question 2

Make a line plot that plots two type of advertising mediums that effects costs against sales. Sales can range between-0 and 60 (million)and advertising costs can range between 0 and 80 (thousand). One type of advertising can be TV another one could be a smart phone. Hard code the sales values and adverting cost values in arrays. Use circle and square markers for the plots. Make a legend.

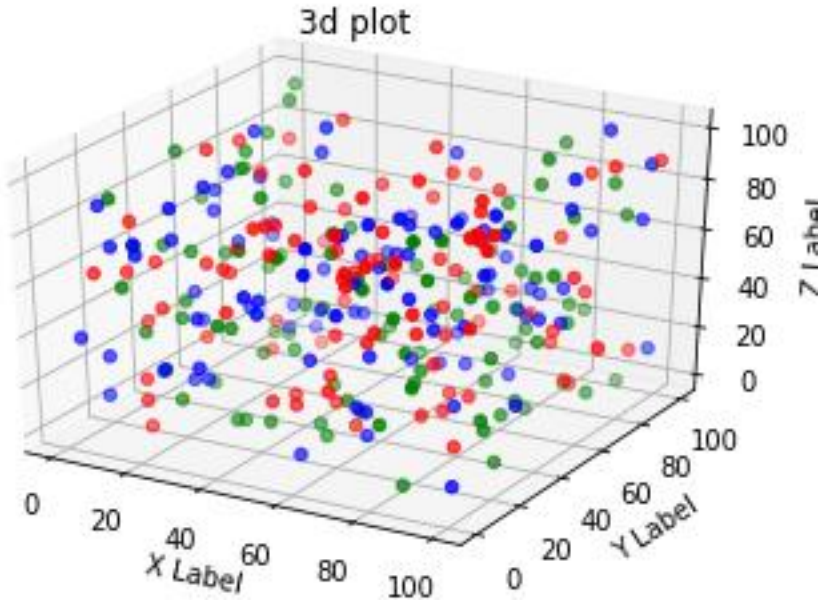
You should get something like this:



Question 3

Change the 3d plot so that all the color groups are mixed together. (maybe add some more colors)

You should get something like this:



PDF, CDF and KDE

The **probability density function** (PDF) and **cumulative distribution function** (CDF), both give a complete description of the probability distribution of a random variable.

The PDF is the probability that a random sample X will be near the given value x) for a *given* mean (mu) *and* standard deviation (sigma).

The CDF is the probability that a random sample X will be less than or equal to x.

Whereas Probability is the chance that the variable has a specific value.

The **Kernel density estimation** (KDE) is a way to estimate the probability density function (PDF) of a random variable.

We denote:

- The probability density function, pdf, as f(x).
- The cumulative distribution function, cdf, as F(x).

The mathematical relationship between the pdf and cdf is given by:

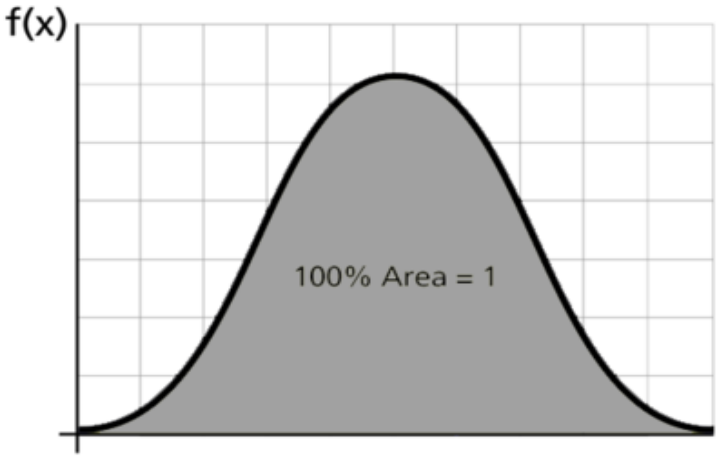
$$F(x)=\int x0 f(s)ds$$

where s is a dummy integration variable.

Conversely:

$$f(x)=d(F(x))dx$$

The following figure illustrates a pdf.



The **cdf** is the area under the probability density function up to a value of *x*. The total area under the **pdf** is always equal to 1

The well-known normal (or Gaussian) distribution is an example of a probability density function. The *pdf* for this distribution is given by:

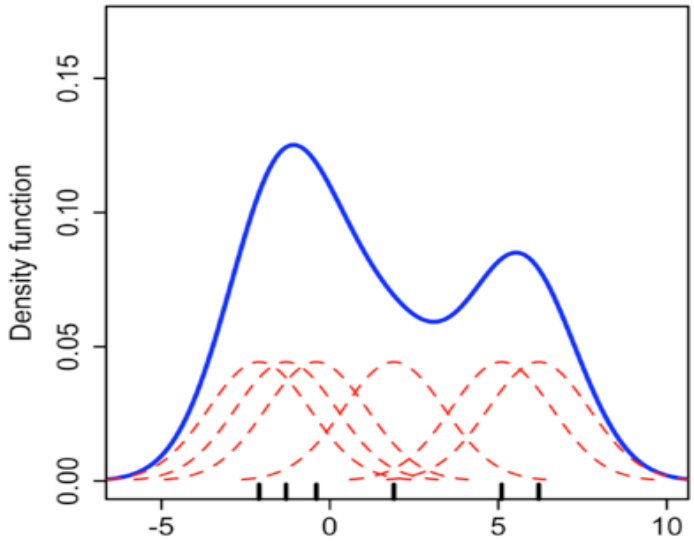
$$f(t) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$

where μ is the mean and σ is the standard deviation.

Kernel Density Estimation KDE

A density plot is a smoothed, continuous version of a histogram estimated from the data. The most common form of estimation is known as **kernel density estimation (KDE)**. The kernel density estimation (KDE) is used to smooth the histogram which filters out the noise.

In this method, a continuous curve known as the **kernel** is drawn at every individual data point and then all of these curves are then added together to make a single smooth density estimation. The density plot is calculated from the sum of theses curves. The kernel most often used is a Gaussian which produces a Gaussian bell curve at each data point.



Plotting PDF, CDF and KDE density

To plot the pdf, cdf and KDE density we first need to import numpy, scipy and the matplot libraries.

```
import numpy as np
from numpy.random import normal
from numpy import mean
from numpy import std
from scipy import stats
import scipy
import matplotlib.pyplot as plt
```

We first make 100 samples with mean 50 and std deviation of 10 using the numpy **random normal** function

```
# generate 100 samples for mean and std
# where loc = mean std = scale
sample = np.random.normal(loc=50, scale=10, size=100)
```

From the sample we calculate mean and std using the numpy mean and std functions

```
# calculate parameters
sample_mean = np.mean(sample)
sample_std = np.std(sample)
print(sample_mean,sample_std)
```

We then use the scipy **norm** function to store the norm distribution mean and std. The **scipy.stats.norm()** represents a normal continuous random variable. A **continuous random variable** whose probabilities are described by the **normal** distribution with mean μ and standard deviation σ is called a **normally** distributed **random variable**, or a **normal random variable** for short, with mean μ and standard deviation σ.

The norm distribution is scaled with the mean (loc) and the std (scale) when mean is not 0 and std not equal to 1.

y = (x - loc) / scale.

We use the scipy stats.norm function to obtain the normal mean of 0 and the std of 1 or to store our own **mean** and **std** value to calculate our distribution values.

```
# store the norm distribution mean and std
dist = scipy.stats.norm(sample_mean, sample_std)
```

The dist just store's the sample_mean and sample std. The returned result from dist = scipy.stats.norm(sample_mean, sample_std) is called a **rv_frozen object** once set it cannot be changed. You can obtain the values from the dist. We can print out the dist and the stored mean and std.

```
print(dist);
print("mean:",dist.mean());
print("std:".dist.std());
```

The print out would be:

```
norm distribution:  <scipy.stats._distn_infrastructure.rv_frozen object at 0x03BDFC10>
mean: 50.26921089112706
std: 10.15038305498074
```

whereas:

```
dist = scipy.stats.norm()
print("mean:",dist.mean());
print("std:".dist.std());
```

would print out:

mean: 0.0
std: 1.0

Since these are the normal distribution defaults.

From the norm distribution we can calculate the pdf probabilities for each 100 values. We calculate pdf's using the norm distribution **mean** and **std** stored in **dist**.

```
# x values 0 to 99
values = [value for value in range(0, 100)]

# calculate pdf's  using normal distribution for x values
pdfs = [dist.pdf(value) for value in values]

# calculate cdf's  using normal distribution for x values
cdfs = [dist.cdf(value) for value in values]
```

Note:

```
dist.pdf(value)
dist.cdf(value)
```

really means

```
stats.norm.pdf(value,dist.mean(), dist.std())
stats.norm.cdf(value,dist.mean(), dist.std())
```

The **dist** object is just used for convenience

In out lot we need to make a second axes for CDF because CDF's have a range from 0 to 1. Whereas histograms, pdf and cdf have a smaller range 0 to .05. Without a second axes the histogram, pdf and density would be too small to view.

```
# plot histogram, pdf and density on 1 subplot
fig, ax1 = plt.subplots()
```

next we assign a title

```
# assign plot title
plt.title('Mean=%.3f, Standard Deviation=%.3f' % (sample_mean, sample_std))
```

We can now plot the histogram pdf, cdf and density on the first axes.

```
# plot the histogram and pdf
ax1.hist(sample, bins=10, density=True)
ax1.plot(values, pdfs,c='r',label="pdf")

# set x and y labels for axes 1
ax1.set_ylabel('pdf density')
ax1.set_xlabel('x values')
```

we make a second axes to plot cdf's

```
# second axes for cdf
ax2 = ax1.twinx()
```

We use the second axes to plot cdf

```
ax2.plot(values, cdfs,c='m',label="cdf")
ax2.set_ylabel('cdf')
```

We use the **scipy.stats.kde.gaussian_kde** function to calculate the KDE from the sample data. The **scipy.stats.kde.gaussian_kde** function is a representation of a kernel-density estimate using Gaussian kernels. Whereas kernel density estimation is a way to estimate the probability density function (PDF) of a random variable in a non-parametric way

```
# calculate kde density
density = scipy.stats.kde.gaussian_kde(sample)
```

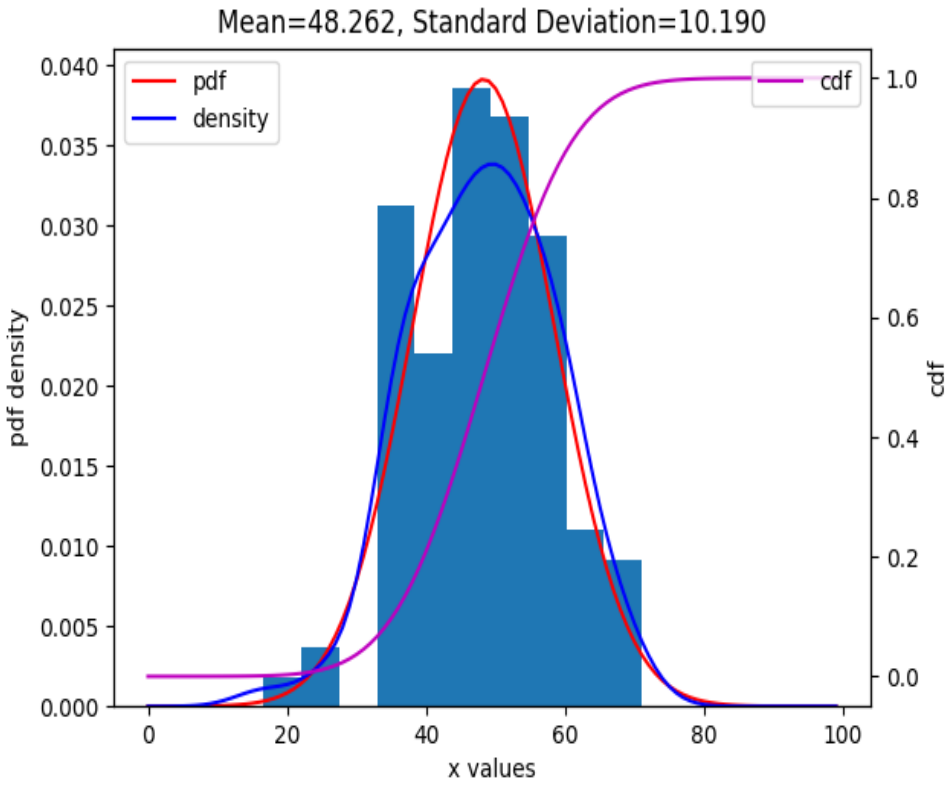
We then plot the density on axes 1 for our 100 values. We are using the same sample and values as above used for calculating and plotting PDF and CDF

```
# plot kde density
ax1.plot(density(values),c='b',label="density")
```

We add legend's for PDF, density and CDF

```
# display legend's
ax2.legend(loc=1)
ax1.legend(loc=2)

plt.show()
```

Our plot is as follows:



The density is calculated from the kernel-density estimate using Gaussian kernels where as the pdf and cdf's are calculated from the sample mean and std of the histogram.

Here is the complete program:

```
# plot histogram, pdf, density and cdf
import numpy as np
from numpy.random import normal
from numpy import mean
from numpy import std
from scipy import stats
import scipy
import matplotlib.pyplot as plt
```

```
# generate 100 samples for mean and std
# loc = mean std = scale
sample = np.random.normal(loc=50, scale=10, size=100)

# calculate sample mean and std
sample_mean = np.mean(sample)
sample_std = np.std(sample)

print(sample_mean,sample_std)

# define the norm distribution
dist = scipy.stats.norm(sample_mean, sample_std)

print("norm distribution: ",dist);
print("mean:",dist.mean());
print("std:",dist.std());

# x values 0 to 99
values = [value for value in range(0, 100)]

# calculate pdf's  using normal distribution for x values
pdfs = [dist.pdf(value) for value in values]

# calculate cdf's  using normal distribution for x values
cdfs = [dist.cdf(value) for value in values]

# plot histogram, pdf and density on 1 subplot
fig, ax1 = plt.subplots()

# assign plot title
plt.title('Mean=%.3f, Standard Deviation=%.3f' % (sample_mean, sample_std))

# plot the histogram and pdf
ax1.hist(sample, bins=10, density=True)
ax1.plot(values, pdfs,c='r',label="pdf")
```

```
# set x and y labels for axes 1
ax1.set_ylabel('pdf density')
ax1.set_xlabel('x values')

# second axes for cdf
ax2 = ax1.twinx()

# plot cdf
ax2.plot(values, cdfs,c='m',label="cdf")
ax2.set_ylabel('cdf')

# plot kde density
density = scipy.stats.kde.gaussian_kde(sample)
ax1.plot(density(values),c='b',label="density")

# display legend's
ax2.legend(loc=1)
ax1.legend(loc=2)

plt.show()
```

Example calculating the KDE by summing Gaussian Kernels

In this example we calculate each Gaussian kernel of the histogram bin's using the Gaussian pdf. We obtain the bin centers from the **plt.hist** function,.

```
(n, bins, patches) = plt.hist(sample, bins=10, density=True)
```

We obtain each Gaussian pdf using the **scipy.stats.norm** function**. The scipy.stats.norm** takes in a **mean** and a **std**. We obtain the **mean** from the histogram **bin** and the **std** is the histogram bandwidth THE number of bins. The **scipy.stats.norm function** using the mean and std calculate the **pdf** for the data points using the pdf function.

```
kernel = scipy.stats.norm(bin_i, bandwidth).pdf(values)
```

Next we sum up the kernels to calculate the kde.

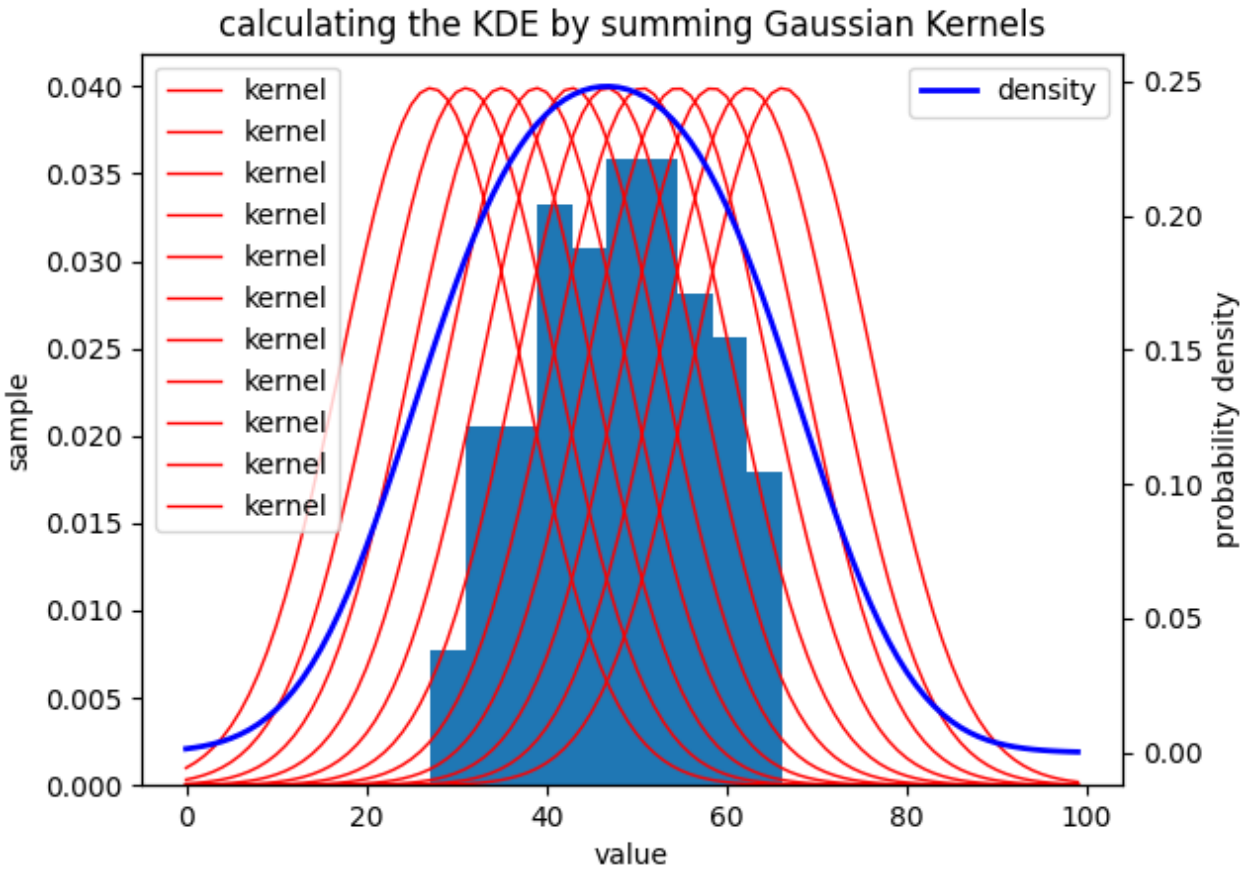
```
density = np.sum(kernels, axis=0)
```

Here is the complete program that uses the same sample data from the previous program

```
# plot Gaussian kernel
# calculate and plot KDE
import numpy as np
import scipy
from scipy import stats
import matplotlib.pyplot as plt

# generate 100 samples for mean and std
# where loc = mean std = scale
sample = np.random.normal(loc=50, scale=10, size=100)

# x values 0 to 99
values = [value for value in range(0, 100)]

# plot histogram and kernel on 1 subplot
fig, ax1 = plt.subplots()

# plot histogram obtain bin centers
(n, bins, patches) = ax1.hist(sample, bins=10, density=True)

# array of kernels
kernels = []

# bandwidth is the std dev of Gaussian kernels
bandwidth = 10
```

```
# for each force
i = 1
for bin_i in bins:

   # obtain kernel on data point
   kernel = scipy.stats.norm(bin_i, bandwidth).pdf(values)

   # add to the kernel list
   kernels.append(kernel)

   # plot each kernal
   ax1.plot(values, kernel, lw=1, color="r",label="kernel" + str(i))
   i+=1

# sum up kernels along rows
density = np.sum(kernels, axis=0)

# second axes for cdf
ax2 = ax1.twinx()

# plot kde
plt.title("calculating the KDE by summing Gaussian Kernels")
ax1.set_xlabel('value')
ax1.set_ylabel("sample")
ax2.set_ylabel('probability density')
ax2.plot(values, density, lw=2, c='b',label="density")

# display legend's
ax2.legend(loc=1)
ax1.legend(loc=2)
plt.show()
```

When you run this program you will get something like this:

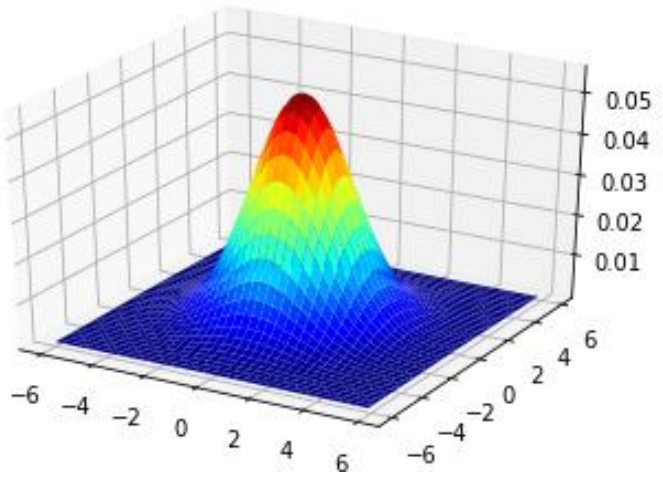


3D Gaussian Plot

Using the Gaussian kernel formula we can plot3d using a color map of many colors.

```
# 3d Gaussian plot
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm # color map
import matplotlib.pyplot as plt
```

```
fwhm = 4
sigma = fwhm / np.sqrt(8 * np.log(2))

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
dx = 0.1
dy = 0.1
x = np.arange(-6, 6, dx)
y = np.arange(-6, 6, dy)
x2d, y2d = np.meshgrid(x, y)
kernel_2d = np.exp(-(x2d ** 2 + y2d ** 2) / (2 * sigma ** 2))
kernel_2d = kernel_2d / (2 * np.pi * sigma ** 2) # unit integral
ax.plot_surface(x2d, y2d, kernel_2d,cmap=cm.jet)
plt.show()
```



Homework

Question 4

Make a histogram of some data, then plot the pdf, cdf and density using the KDE. You can make some data like this:

data = [1.5]*7 + [2.5]*2 + [3.5]*8 + [4.5]*3 + [5.5]*1 + [6.5]*8

From the data get the mead and standard deviation.

data_mean = np.mean(data)
data_std = np.std(data)

Using the mean and std deviation you can calculate the pdf and cdf from the scipy.stats.norm module functions pdf and cdf.

pdf = scipy.stats.norm.pdf(xs,data_mean,data_std)
cdf = scipy.stats.norm.cdf(xs,data_mean,data_std)

From the data sample you can calculate density

density = scipy.stats.kde.gaussian_kde(data)

Next use **linspace** to evenly divide the data for the x axis

xs = np.linspace(0,8,100)

You can now plot the pdf, cdf and density using a legend.

Finally calculate the KDE by plotting the Gaussian Kernels.

Use

fig, ax1 = plt.subplots()

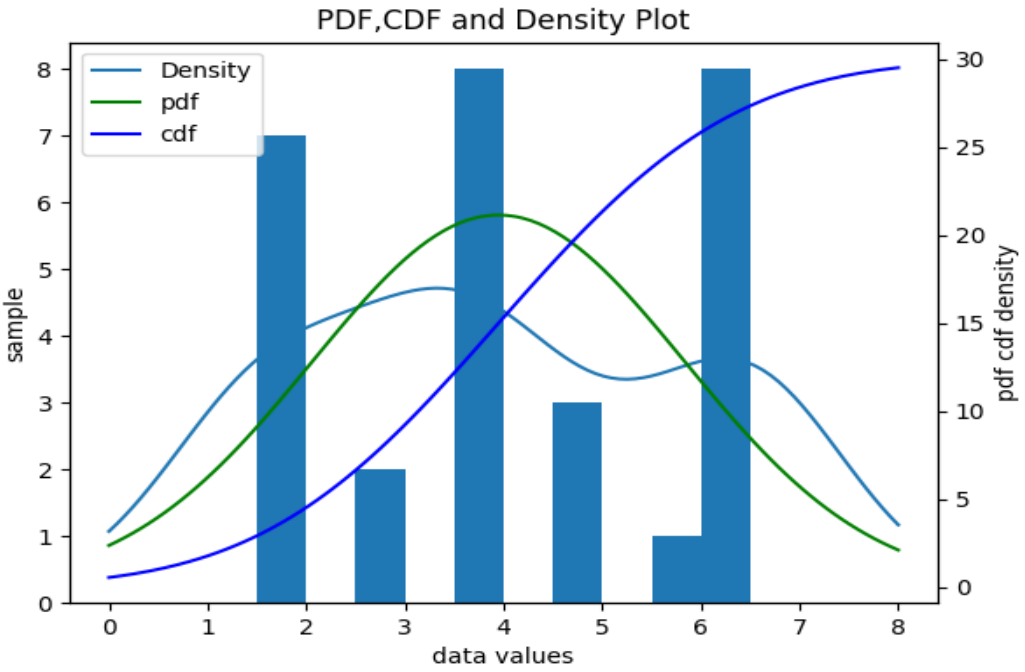
and

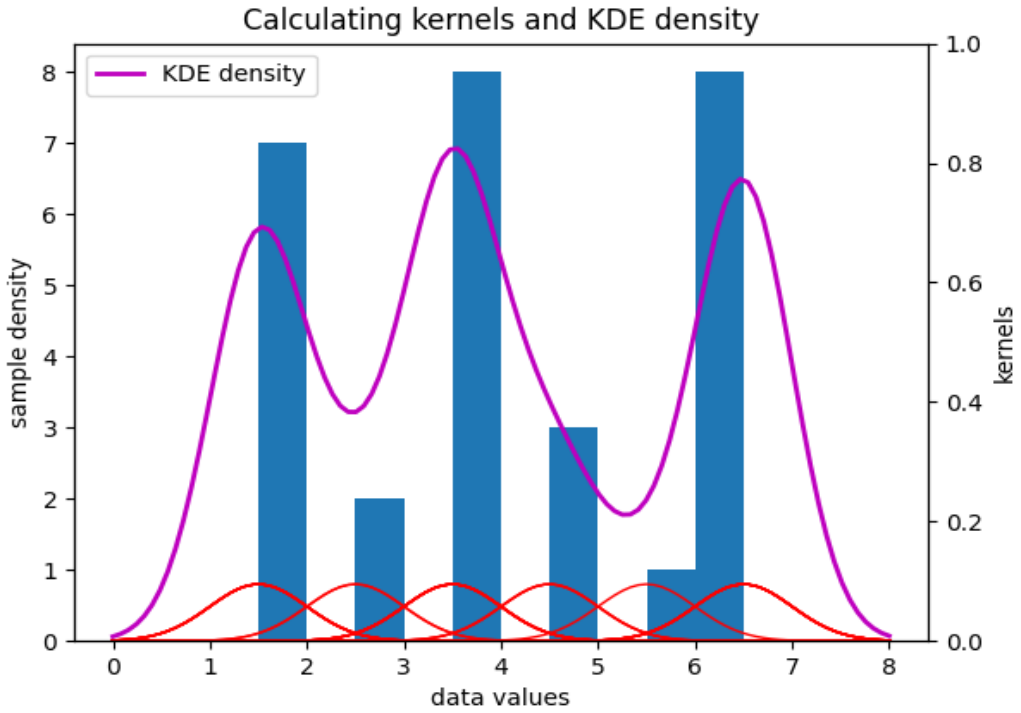
ax2 = ax1.twinx()

To make a second axis for the cdf.

You may put every thing in 1 plot or 2 plots

You should get something like this:





END