Unit -1

Basics of Machine Learning and Python

Review of Linear Algebra:

Linear Algebra is an essential field of mathematics, which defines the study of vectors, matrices, planes, mapping, and lines required for linear transformation.

The term Linear Algebra was initially introduced in the early 18th century to find out the unknowns in Linear equations and solve the equation easily; hence it is an important branch of mathematics that helps study data.

Linear algebra plays a vital role and key foundation in machine learning***, and it enables ML algorithms to run on a huge number of datasets***.

Below are some benefits of learning Linear Algebra before Machine learning:

- **Better Graphic experience**
- **Improved Statistics**
- **Creating better Machine Learning algorithms**
- **Estimating the forecast of Machine Learning**
- **Easy to Learn**

Few supervised learning algorithms can be created using Linear Algebra, which is as follows:

- **Logistic Regression**
- **Linear Regression**
- **Decision Trees**
- **Support Vector Machines (SVM)**

There are some unsupervised learning algorithms listed that can also be created with the help of linear algebra as follows:

- **Single Value Decomposition (SVD)**
- **Clustering**
- **Components Analysis**

Machine Learning:

Machine learning is a growing technology which enables computers to learn automatically from past data. Machine learning uses various algorithms for **building mathematical models and making predictions using historical data or information**.

The term machine learning was first introduced by **Arthur Samuel** in **1959**.

Machine learning enables a machine to automatically learn from data, improve performance from experiences, and predict things without being explicitly programmed.

Features of Machine Learning:

- Machine learning uses data to detect various patterns in a given dataset.
- It can learn from past data and improve automatically.
- It is a data-driven technology.
- Machine learning is much similar to data mining as it also deals with the huge amount of the data.

Machine learning allows the user to feed a computer algorithm an immense amount of data and have the computer analyze and make data-driven recommendations and decisions based on only the input data. If any corrections are identified, the algorithm can incorporate that information to improve its future decision making.

Goal of Machine Learning:

The major goal of machine learning research is to develop general-purpose algorithms to gain practical insights. These kinds of algorithms are efficient. Like always, as computer scientists, we take care of certain things like time and space efficiency. We also pay attention towards the great learning. We also deal with the amount of data we need for algorithms.

We always thrive for very simple algorithms to learn and solve problems. Our main objective is to predict the outcome of learning, to be as accurate as possible in the predictions it makes. Very rarely, we may also be interested in the interpretation of the rules of assessment produced by learning.

Basic Difference in ML and Traditional Programming:

- **Traditional Programming:** We feed in DATA (Input) + PROGRAM (logic), run it on the machine, and get the output.
- **Machine Learning:** We feed in DATA(Input) + Output, run it on the machine during training and the machine creates its own program(logic), which can be evaluated while testing.

Some Massive Usages of Machine Learning:

1. **Amazon's Product Recommendations:** Ever wondered how Amazon always has a recommendation that just tempts you to lighten your wallet. Well, that's a Machine Learning Algorithm(s) called "Recommender Systems" working in the backdrop. It learns every user's personal preferences and makes recommendations according to that.
2. **YouTube/Netflix:** They work just as above!
3. **Data Mining / Big Data:** This might not be so much of a shock to many. But Data Mining and Big Data are just manifestations of studying and learning from data at a larger scale. And wherever there's the objective of extracting information from data, you'll find Machine Learning lurking nearby.
4. **Stock Market/Housing Finance/Real Estate:** All of these fields, incorporate a lot of Machine Learning systems in order to better assess the market, namely "Regression Techniques", for things as mediocre as predicting the price of a House, to predicting and analyzing stock market trends.

INFORMATION: Data that has been interpreted and manipulated and has now some meaningful inference for the users.

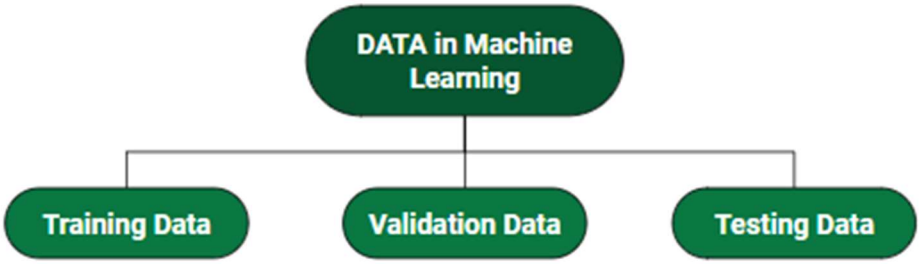
KNOWLEDGE: Combination of inferred information, experiences, learning, and insights. Results in awareness or concept building for an individual or organization.

How we split data in Machine Learning?

- **Training Data:** The part of data we use to train our model. This is the data that your model actually sees(both input and output) and learns from.

- **Validation Data:** The part of data that is used to do a frequent evaluation of the model, fit on the training dataset along with improving involved hyper parameters (initially set parameters before the model begins learning). This data plays its part when the model is actually training.

- **Testing Data:** Once our model is completely trained, testing data provides an unbiased evaluation. When we feed in the inputs of testing data, our model will predict some values (without seeing actual output). After prediction, we evaluate our model by comparing it with the actual output present in the testing data. This is how we evaluate and see how much our model has learned from the experiences feed in as training data, set at the time of training.



Different Forms of Data

- **Numeric Data**: If a feature represents a characteristic measured in number, it is called a numeric feature.

- **Categorical Data:** A categorical feature is an attribute that can take on one of the limited, and usually fixed number of possible values on the basis of some qualitative property. A categorical feature is also called a nominal feature.

Properties of Data –

1. **Volume:** Scale of Data. With the growing world population and technology at exposure, huge data is being generated each and every millisecond.

2. **Variety:** Different forms of data – healthcare, images, videos, audio clippings.

3. **Velocity:** Rate of data streaming and generation.

4. **Value:** Meaningfulness of data in terms of information that researchers can infer from it.

5. **Veracity:** Certainty and correctness in data we are working on.

Unit-1

Introduction to Numpy

Numpy is one of the most important foundational packages for Numerical computing in python

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

NumPy is a Python library and is written partially in Python, but most of the parts that require fast computation are written in C or C++.

Advantages of Numpy:

1. NumPy performs array-oriented computing.
2. It efficiently implements the multidimensional arrays.
3. It performs scientific computations.
4. It is capable of performing Fourier Transform and reshaping the data stored in multidimensional arrays.
5. NumPy provides the in-built functions for linear algebra and random number generation

Advantages of using Numpy Arrays Over Python Lists:

1. consumes less memory.
2. fast as compared to the python List.
3. convenient to use.

Installation of Numpy

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-packages (1.19
Note: you may need to restart the kernel to use updated packages.
```

Import Numpy and check the version?

```
import numpy as np
print(np.__version__)
```

```
1.19.2
```

Data Types in Numpy

NumPy Built-in Data Types

We can reference the built-in data types in NumPy by particular character code.

i – integer

b – boolean

u – unsigned

f – float

c – complex float

m – timedelta

M – datetime

O – object

S – string

U – unicode string

V – void

Numpy Scalar Data Types

1. bool_ – It is used to return Boolean true or false values.
2. int_ – It is the default integer type (int64 or int32)
3. intc – It is identical to the integer in C (int32 or int64)
4. intp – It is the integer value used for indexing
5. int8 – It is for assigning 8-bit integer value (-128 to 127)
6. int16 – It is for assigning 16-bit integer value (-32768 to 32767)
7. int32 – It is for assigning 32-bit integer value (-2147483648 to 2147483647)
8. int32 – It is for assigning 64-bit integer value (-9223372036854775808 to 9223372036854775807)
9. uint8 – It is for assigning unsigned 8-bit integer value (0 to 255)
10. uint16 – It is for assigning unsigned 16-bit integer value (0 to 65535)
11. uint32 – It is for assigning unsigned 32-bit integer value (0 to 4294967295)
12. uint64 – It is for assigning unsigned 64-bit integer value (0 to 18446744073709551615)
13. float_ – It is to assign float values.
14. float16 – It is for half precision float values.
15. float32 – It is for single-precision float values.
16. float64 – It is for double-precision float values.
17. complex_ – It is to assign complex values.
18. complex64 – It is to represent two 32-bit float complex values (real and imaginary)
19. complex128 – It is to represent two 64-bit float complex values (real and imaginary)

Casting

You can convert or cast an array from one data type to another

```
arra1 = np.array([1, 2, 3, 4, 5])
print(arra1.astype(np.float64))   #Change Data Int to float.
print(arra1.dtype)
```

```
[1. 2. 3. 4. 5.]
int32
```

```
# One Dimentional Arrays

import numpy as np
a = np.array([1,2,3])
print(a)
```

```
[1 2 3]
```

```
# more than one dimensions

import numpy as np
a = np.array([[1, 2], [3, 4]])
print(a)
```

```
[[1 2]
 [3 4]]
```

```
# minimum dimensions
import numpy as np
a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a)
```

```
[[1 2 3 4 5]]
```

```
# dtype parameter
import numpy as np
a = np.array([1, 2, 3], dtype = complex)
print(a)
```

```
[1.+0.j 2.+0.j 3.+0.j]
```

```
print(a.shape)
```

```
(3,)
```

```
# Reshape Function

import numpy as np
a = np.array([[1,2,3],[4,5,6]])
b = a.reshape(3,2)
print(b)
```

```
[[1 2]
 [3 4]
 [5 6]]
```

How to create a boolean array?

```
np.full((3, 3), True, dtype=bool)
```

```
array([[ True,  True,  True],
       [ True,  True,  True],
       [ True,  True,  True]])
```

How to extract items that are Odd Numbers from 1D array?

```
arr = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

arr[arr % 2 == 1]
```

```
array([1, 3, 5, 7, 9])
```

Generate Random Numbers

```
# Random Numbers from 0 to 100 and total 10 Numbers.

print(np.random.randint(0,100, size=10,dtype=int))
```

```
[ 1 39 49 45 42 68 18 35 56 51]
```

Other ways to create new arrays

↳ *1 cell hidden*

Zero

`zero()` creates an array filled with zero values. arguement is an array or tulip with size of each dimension of the array

```
np.zeros(10)
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
np.zeros((3, 6))
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

Ones

`ones()` - filled with one values.

```
np.ones((2,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

Empty

`empty()` function create an array of specified shape. Arguement is an rray or tulip with lenght of each dimension of array
Value created are random and empty array needs to be assigned before use

```
np.empty((3, 3, 2))
```

```
array([[[0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.]],

       [[0., 0.],
        [0., 0.],
        [0., 0.]]])
```

`eye`, `identity` Create a square N × N identity matrix (1s on the diagonal and 0s elsewhere)

```
np.eye(2)
```

```
array([[1., 0.],
       [0., 1.]])
```

```

```

```

```

Numpy ndarray - Multidimensional Array Object

`ndarray` is one of the most important feature of Numpy
`ndarray` is multidimensional container of homogeneous data

All the elements are of same type

```
import numpy as np
data = np.random.randn(2, 3)  # Random Numbers generated between 2 to 3.
print(data)
```

```
[[ 1.34092554 -0.53189297 -1.03180517]
 [ 0.76183449  0.10177404  0.69100574]]
```

```
an_arr = np.arange(1000)
print(type(an_arr))         #Type of Array.
```

```
<class 'numpy.ndarray'>
```

Combining ndarray

New ndarrary can be created.

`vstack()` and `hstack` are two popular methods

`vstack()` - Vertical Stack

`hstack()` - Horizontal Stack

Vertical Stack

Two or more array can be stacked vertically

```
arr1 = np.array([[1,2,3],[4,5,6]])         #2D Array
print(arr1)
arr2 = np.array([[7,8,9],[10,11,12]])       #3D Array
print(arr2)
arr3 = np.vstack((arr1, arr2))         #Combining Array in Vertical Stack.
print(arr3)
print(arr3.shape)
```

```
[[1 2 3]
 [4 5 6]]
[[ 7  8  9]
 [10 11 12]]
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
(4, 3)
```

Horizontal Stack

Two or more ndarray can be stacked horizontally

```
arr3 = np.hstack((arr1, arr2))
print(arr3)
print(arr3.shape)
```

```
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
(2, 6)
```

Concatenating ndarray

`concatenate()` take a sequence (tulip, list etc) and join them along input axis

```
arr3 = np.concatenate([arr1, arr2], axis=0)
print(arr3)
print(arr3.shape)
```

```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
(4, 3)
```

```
arr3 = np.concatenate([arr1, arr2], axis=1)
print(arr3)
print(arr3.shape)
```

```
[[ 1  2  3  7  8  9]
 [ 4  5  6 10 11 12]]
(2, 6)
```

Splitting Array

divides the array into subarrays along a specified axis

Syntax: numpy.split(ary, indices_or_sections, axis)

```
a = np.arange(9)
print('first array')
print(a)
print('\n')
print('splitting the array in 3 equal size arrays')
b = np.split(a,3)
print(b)
print('\n')
print('splitting the array indicated first at 4, second 5- 7 remaining in 3rd')
```

```
c=np.split(a,[2,5,7])
print(c)
print('\n')
```

```
first array
[0 1 2 3 4 5 6 7 8]


splitting the array in 3 equal size arrays
[array([0, 1, 2]), array([3, 4, 5]), array([6, 7, 8])]


splitting the array indicated first at 4, second 5- 7 remaining in 3rd
[array([0, 1]), array([2, 3, 4]), array([5, 6]), array([7, 8])]
```

Arithmetic with Numpy Array

Numpy support the batch operations on data without `for` loops. This is call as *vectorization*

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

```
array([[1, 2, 3],
       [4, 5, 6]])
```

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

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
arr+arr
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

```
arr-arr
```

```
array([[0, 0, 0],
       [0, 0, 0]])
```

```
arr*arr
```

```
array([[ 1,  4,  9],
       [16, 25, 36]])
```

```
arr%arr
```

```
array([[0, 0, 0],
       [0, 0, 0]], dtype=int32)
```

```
arr/arr
```

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

```
1/arr
```

```
array([[1.        , 0.5       , 0.33333333],
       [0.25      , 0.2       , 0.16666667]])
```

```
arr**0.5
```

```
array([[1.        , 1.41421356, 1.73205081],
       [2.        , 2.23606798, 2.44948974]])
```

Comparision of array

```
arr2 = np.array([[3,4,5],[6,7,8]])
arr2
```

```
array([[3, 4, 5],
       [6, 7, 8]])
```

```
arr2 = np.array([[1,2,6],[4,5,6]])
arr2
```

```
array([[1, 2, 6],
       [4, 5, 6]])
```

```
arr2>arr1    #Show the Boolean Result
```

```
array([[False, False,  True],
       [False, False, False]])
```

Array Indexing

You can access data in Numpy using Indexing

One Dimensional Indexing

You can access One Dimensional Indexing using bracket operator [] by specifying the zero offset value

```
arr = np.array([10, 20, 30, 40, 50])
```

```
arr[0]
```

```
10
```

```
10+arr[0]
```

```
20
```

```
arr[-1]
```

```
50
```

Two Dimensional Index

use comma to separate the index of each dimension

```
arr2d=np.array([[10, 20, 30],[40, 50, 60]])
arr2d
```

```
array([[10, 20, 30],
       [40, 50, 60]])
```

```
print(arr2d[0,0])
print(arr2d[1,2])
print(arr2d[1,0])
```

```
10
60
40
```

```
arr2d[1,]
```

```
array([40, 50, 60])
```

Array Slicing

slicing is specified using a colon operator `:` with *from* and *to* arguements

`data[from:to]`

One Dimensional Slicing

You can index all data using slice : with no index

[ ] ↳ *1 cell hidden*

Array Broadcasting

Array Arithmetic Limitations

Array with different shape and size cannot be added or subtracted in Arithmetic
A one dimensional array with length 8 can be added or subtracted from another array of length 8 only
This limittion in arithmetic is limiting
Python has buildin functionlity **broadcasting** to address this

```
a = np.array([1, 2, 3])
print(a)
b = 2
print(b)
c = a + b
print(c)
```

```
[1 2 3]
2
[3 4 5]
```

2D array and Scalar

```
a = np.array([[1, 2, 3],[4, 5, 6]])
print(a)
b = 4
print(b)
c = a + b # scalar b will duplicate itself 5 more times to broadcast
print(c)
```

```
[[1 2 3]
 [4 5 6]]
4
[[ 5  6  7]
 [ 8  9 10]]
```

1D and 2D Array Broadcasting

```
a = np.array([[1, 2, 3],[4, 5, 6]])
print(a)
b = np.array([7, 8, 9])
print(b)
c = a+b  # 1D array b is broadcast across each row of 2D by creating a copy of itself
print(c)
```

```
[[1 2 3]
 [4 5 6]]
[7 8 9]
[[ 8 10 12]
 [11 13 15]]
```

Limitations of Broadcasting

Broadcasting can be performed when

1. the shape of each dimension in the array are equal

   or
2. one has dimension size 1

```
import numpy as np

arr = np.array([[1, 4, 3], [11, 2, 10]])

print ("Input array : ", arr)

sum1 = np.cumsum(arr)    #Addition of all Elements to the Previous elements.
print(sum1)
```

```
Input array :  [[ 1  4  3]
 [11  2 10]]
[ 1  5  8 19 21 31]
```

What is Matplotlib

Matplotlib is a low level graph plotting library in python that serves as a visualization utility.

Matplotlib was created by John D. Hunter.

Matplotlib is open source and we can use it freely.

Matplotlib is mostly written in python, a few segments are written in C, Objective-C and Javascript for Platform compatibility.

Installation of Matplotlib

In [1]:
```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\programdata\anaconda3\lib\site-
packages (3.3.2)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\p
rogramdata\anaconda3\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: numpy>=1.15 in c:\programdata\anaconda3\lib\site
-packages (from matplotlib) (1.19.2)
Requirement already satisfied: certifi>=2020.06.20 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib) (2020.6.20)
Requirement already satisfied: python-dateutil>=2.1 in c:\programdata\anaconda3
\lib\site-packages (from matplotlib) (2.8.1)Note: you may need to restart the k
ernel to use updated packages.
Requirement already satisfied: cycler>=0.10 in c:\programdata\anaconda3\lib\sit
e-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pillow>=6.2.0 in c:\programdata\anaconda3\lib\si
te-packages (from matplotlib) (8.0.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\programdata\anaconda3\li
b\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-pa
ckages (from python-dateutil>=2.1->matplotlib) (1.15.0)
```

Import Matplotlib

Once Matplotlib is installed, import it in your applications by adding the import module statement:

Most of the Matplotlib utilities lies under the pyplot submodule, and are usually imported under the plt alias:

In [2]:
```
import matplotlib.pyplot as plt
```

Checking Matplotlib Version

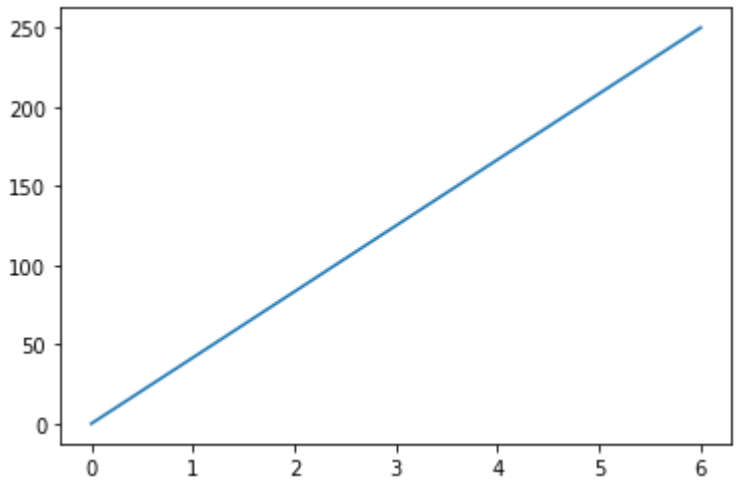
The version string is stored under **version** attribute.

In [3]:
```
import matplotlib
print(matplotlib.__version__)
```

```
3.3.2
```

Draw a line in a diagram from position (0,0) to position (6,250):

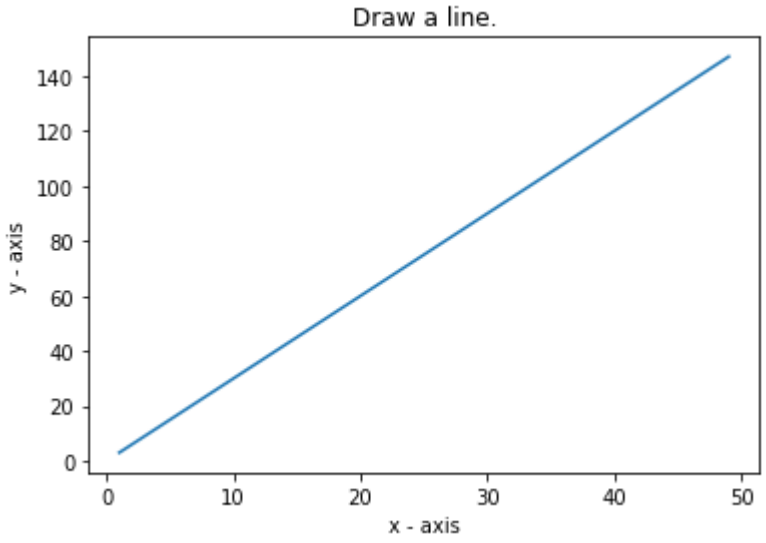
In [5]:
```
import matplotlib.pyplot as plt
import numpy as np

xpoints = np.array([0, 6])
ypoints = np.array([0, 250])

plt.plot(xpoints, ypoints)
plt.show()
```



1. Write a Python program to draw a line with suitable label in the x axis, y axis and a title.

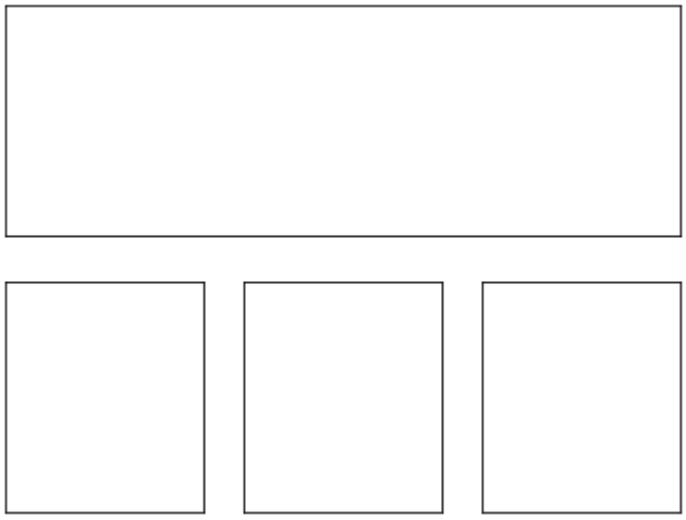
In [6]:
```
import matplotlib.pyplot as plt
X = range(1, 50)
Y = [value * 3 for value in X]
print("Values of X:")
print(*range(1,50))
print("Values of Y (thrice of X):")
print(Y)
# Plot lines and/or markers to the Axes.
plt.plot(X, Y)
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Draw a line.')
# Display the figure.
plt.show()
```

```
Values of X:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 3
0 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
Values of Y (thrice of X):
[3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57, 60, 6
3, 66, 69, 72, 75, 78, 81, 84, 87, 90, 93, 96, 99, 102, 105, 108, 111, 114, 11
7, 120, 123, 126, 129, 132, 135, 138, 141, 144, 147]
```



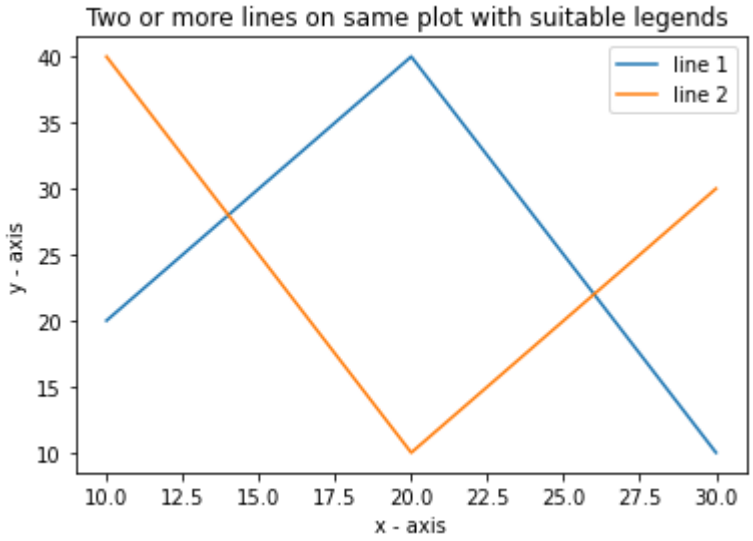
Write a Python program to create multiple plots.

In [7]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
fig.subplots_adjust(bottom=0.020, left=0.020, top = 0.900, right=0.800)

plt.subplot(2, 1, 1)
plt.xticks(()), plt.yticks(())

plt.subplot(2, 3, 4)
plt.xticks(())
plt.yticks(())

plt.subplot(2, 3, 5)
plt.xticks(())
plt.yticks(())

plt.subplot(2, 3, 6)
plt.xticks(())
plt.yticks(())

plt.show()
```



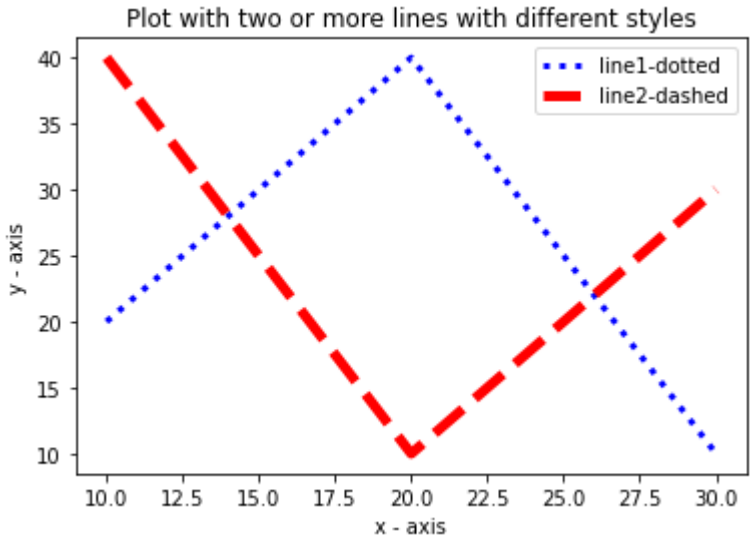
Write a Python program to plot two or more lines on same plot with suitable legends of each line.

In [8]:
```
import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# plotting the line 1 points
plt.plot(x1, y1, label = "line 1")
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# plotting the line 2 points
plt.plot(x2, y2, label = "line 2")
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title of the current axes.
plt.title('Two or more lines on same plot with suitable legends ')
# show a legend on the plot
plt.legend()
# Display a figure.
plt.show()
```



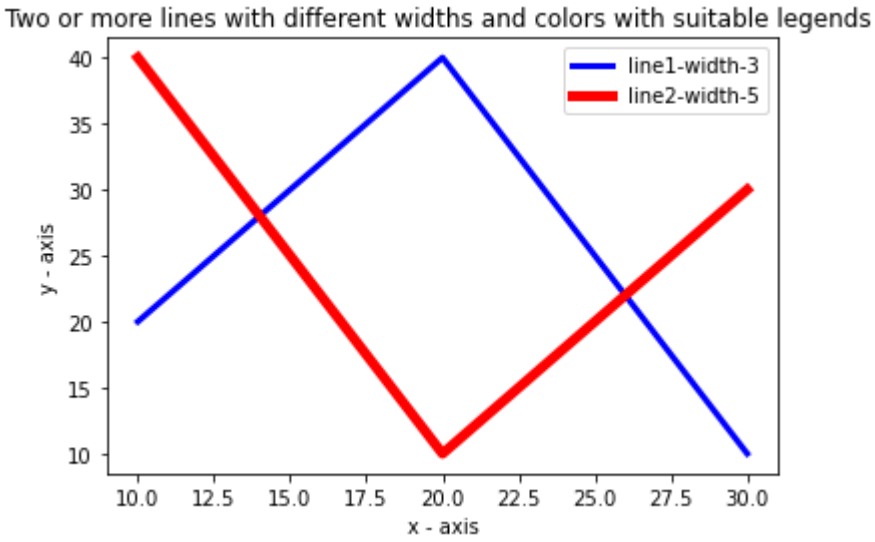
Write a Python program to plot two or more lines with different styles.

In [9]:
```
import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Plot lines and/or markers to the Axes.
plt.plot(x1,y1, color='blue', linewidth = 3,  label = 'line1-dotted',linestyle='d
plt.plot(x2,y2, color='red', linewidth = 5,  label = 'line2-dashed', linestyle='d
# Set a title
plt.title("Plot with two or more lines with different styles")
# show a legend on the plot
plt.legend()
# function to show the plot
plt.show()
```



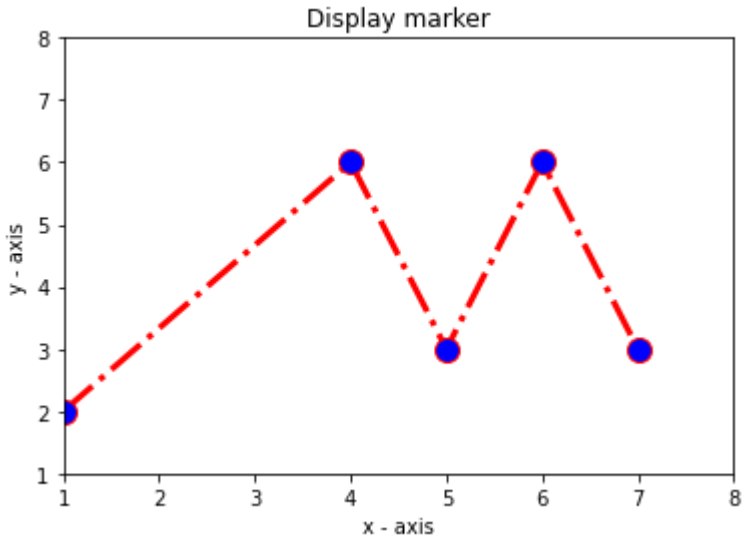
Write a Python program to plot two or more lines with legends, different widths and colors.

In [10]:
```
import matplotlib.pyplot as plt
# line 1 points
x1 = [10,20,30]
y1 = [20,40,10]
# line 2 points
x2 = [10,20,30]
y2 = [40,10,30]
# Set the x axis label of the current axis.
plt.xlabel('x - axis')
# Set the y axis label of the current axis.
plt.ylabel('y - axis')
# Set a title
plt.title('Two or more lines with different widths and colors with suitable leger
# Display the figure.
plt.plot(x1,y1, color='blue', linewidth = 3,  label = 'line1-width-3')
plt.plot(x2,y2, color='red', linewidth = 5,  label = 'line2-width-5')
# show a legend on the plot
plt.legend()
plt.show()
```



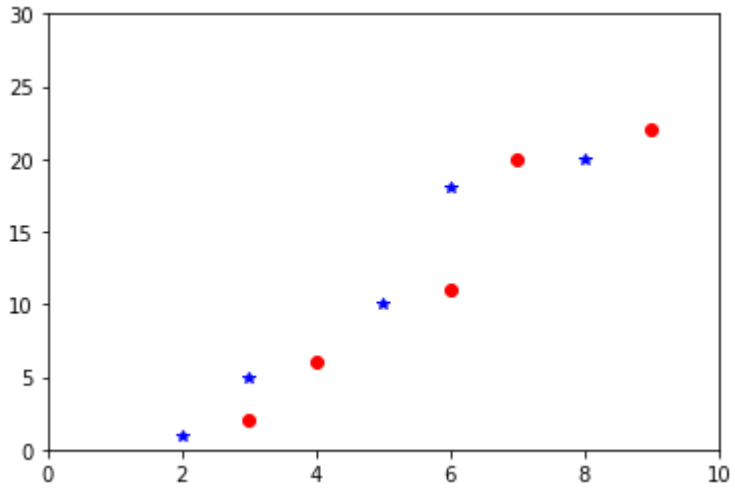
Write a Python program to plot two or more lines and set the line markers.

In [11]:
```
import matplotlib.pyplot as plt
# x axis values
x = [1,4,5,6,7]
# y axis values
y = [2,6,3,6,3]

# plotting the points
plt.plot(x, y, color='red', linestyle='dashdot', linewidth = 3,
         marker='o', markerfacecolor='blue', markersize=12)
#Set the y-limits of the current axes.
plt.ylim(1,8)
#Set the x-limits of the current axes.
plt.xlim(1,8)

# naming the x axis
plt.xlabel('x - axis')
# naming the y axis
plt.ylabel('y - axis')

# giving a title to my graph
plt.title('Display marker')
# function to show the plot
plt.show()
```



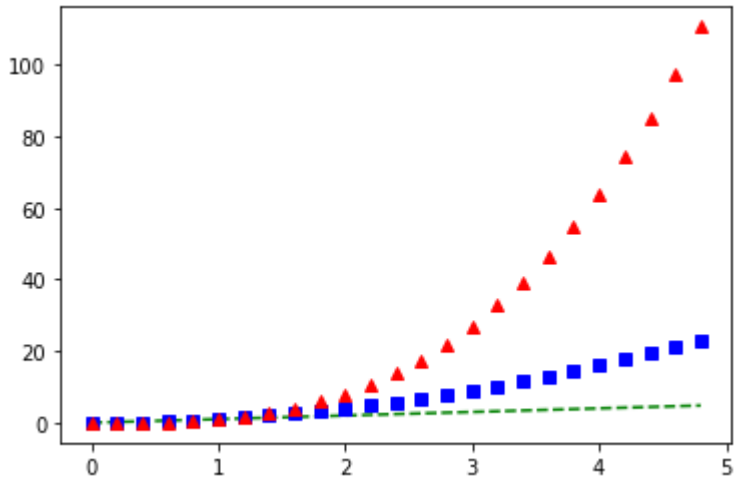
Write a Python program to plot quantities which have an x and y position.

In [12]:
```
import numpy as np
import pylab as pl
# Make an array of x values
x1 = [2, 3, 5, 6, 8]
# Make an array of y values for each x value
y1 = [1, 5, 10, 18, 20]
# Make an array of x values
x2 = [3, 4, 6, 7, 9]
# Make an array of y values for each x value
y2 = [2, 6, 11, 20, 22]
# set new axes limits
pl.axis([0, 10, 0, 30])
# use pylab to plot x and y as red circles
pl.plot(x1, y1,'b*', x2, y2, 'ro')
# show the plot on the screen
pl.show()
```



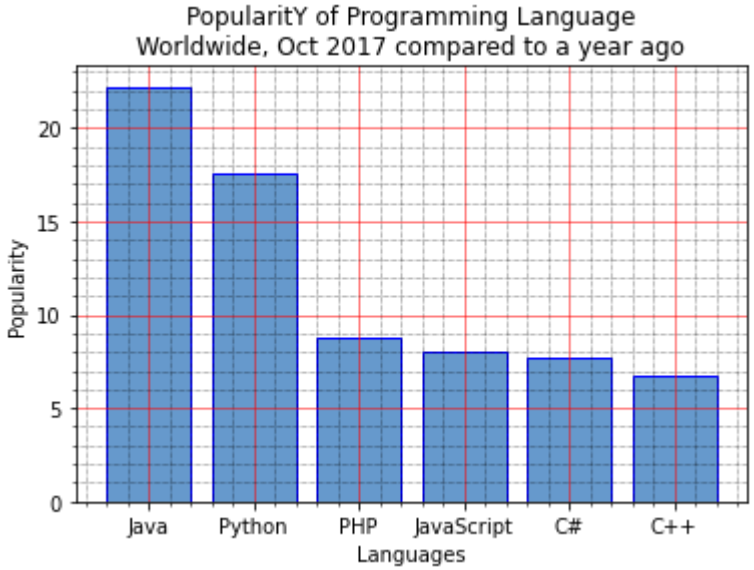
Write a Python program to plot several lines with different format styles in one command using arrays.

In [13]:
```
import numpy as np
import matplotlib.pyplot as plt

# Sampled time at 200ms intervals
t = np.arange(0., 5., 0.2)

# green dashes, blue squares and red triangles
plt.plot(t, t, 'g--', t, t**2, 'bs', t, t**3, 'r^')
plt.show()
```



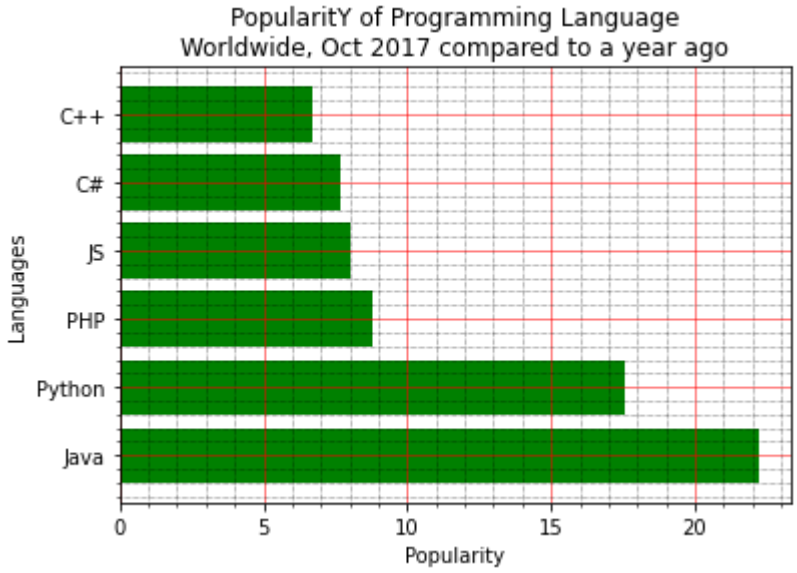
Write a Python programming to display a bar chart of the popularity of programming Languages.

In [14]:
```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, popularity, color=(0.4, 0.6, 0.8, 1.0), edgecolor='blue')

plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



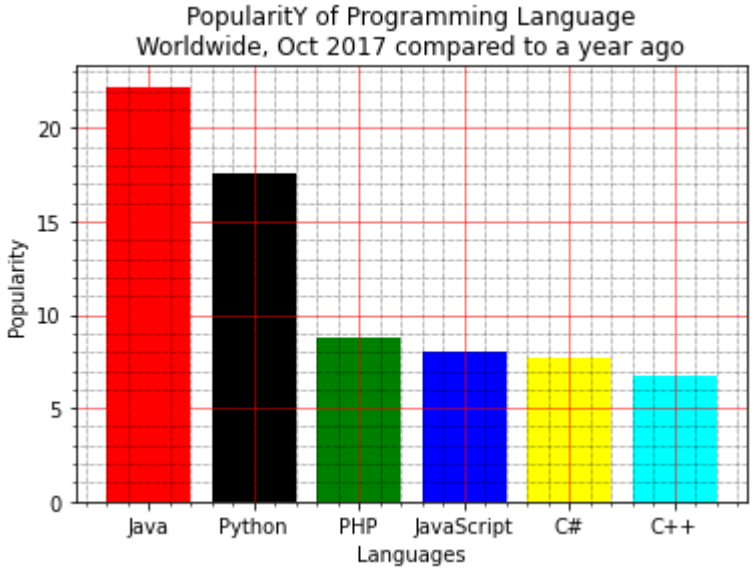
Write a Python programming to display a horizontal bar chart of the popularity of programming Languages.

In [15]:
```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JS', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]
plt.barh(x_pos, popularity, color='green')
plt.xlabel("Popularity")
plt.ylabel("Languages")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared
plt.yticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



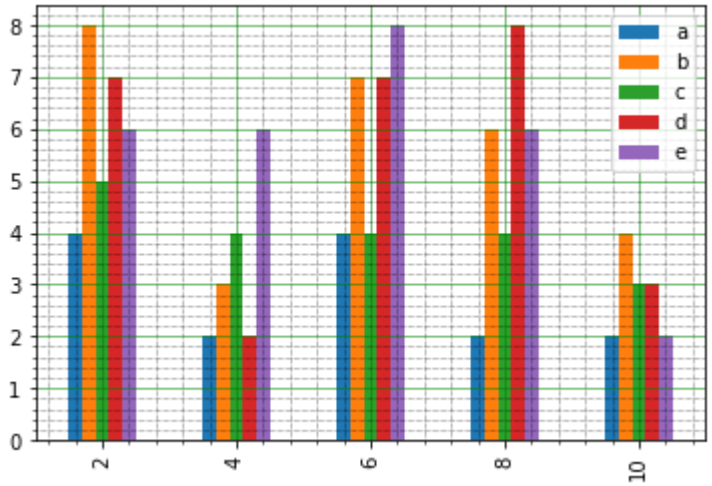
Write a Python programming to display a bar chart of the popularity of programming Languages. Use different color for each bar

In [16]:
```
import matplotlib.pyplot as plt
x = ['Java', 'Python', 'PHP', 'JavaScript', 'C#', 'C++']
popularity = [22.2, 17.6, 8.8, 8, 7.7, 6.7]
x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, popularity, color=['red', 'black', 'green', 'blue', 'yellow', 'cya

plt.xlabel("Languages")
plt.ylabel("Popularity")
plt.title("PopularitY of Programming Language\n" + "Worldwide, Oct 2017 compared
plt.xticks(x_pos, x)
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='red')
# Customize the minor grid
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')
plt.show()
```



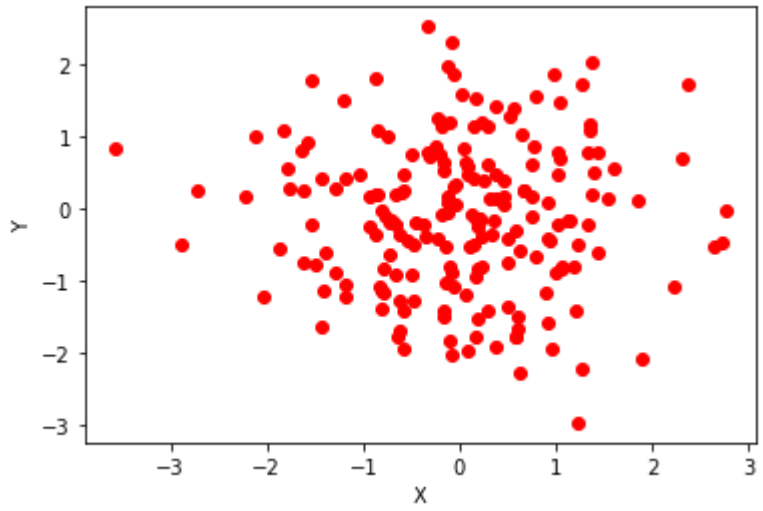
Write a Python program to create bar plot from a DataFrame.

In [17]:
```
from pandas import DataFrame
import matplotlib.pyplot as plt
import numpy as np

a=np.array([[4,8,5,7,6],[2,3,4,2,6],[4,7,4,7,8],[2,6,4,8,6],[2,4,3,3,2]])
df=DataFrame(a, columns=['a','b','c','d','e'], index=[2,4,6,8,10])

df.plot(kind='bar')
# Turn on the grid
plt.minorticks_on()
plt.grid(which='major', linestyle='-', linewidth='0.5', color='green')
plt.grid(which='minor', linestyle=':', linewidth='0.5', color='black')

plt.show()
```



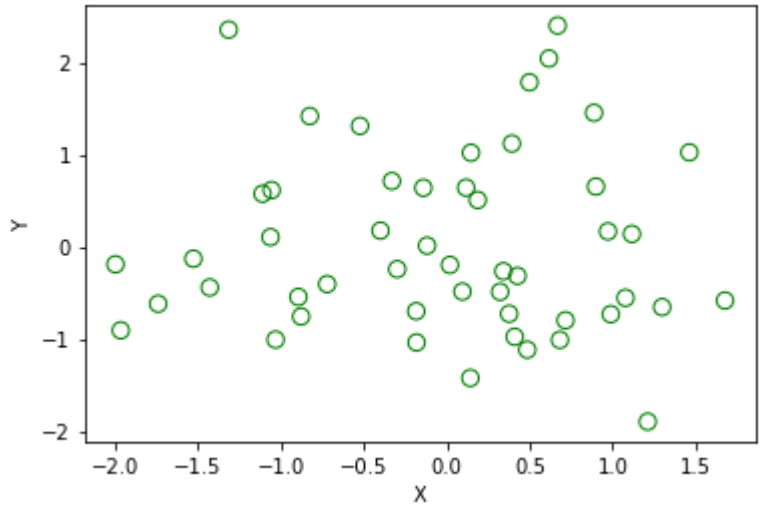
Write a Python program to draw a scatter graph taking a random distribution in X and Y and plotted against each other.

In [19]:
```
import matplotlib.pyplot as plt
from pylab import randn
X = randn(200)
Y = randn(200)
plt.scatter(X,Y, color='r')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



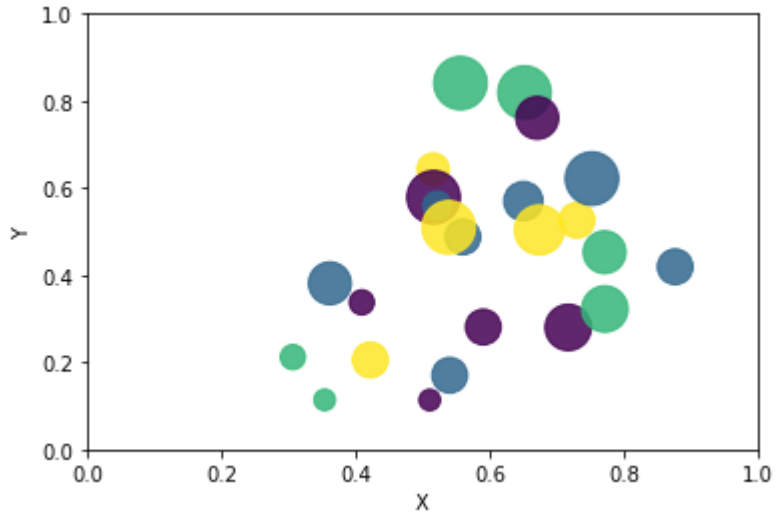
Write a Python program to draw a scatter plot with empty circles taking a random distribution in X and Y and plotted against each other.

In [20]:
```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randn(50)
y = np.random.randn(50)
plt.scatter(x, y, s=70, facecolors='none', edgecolors='g')
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



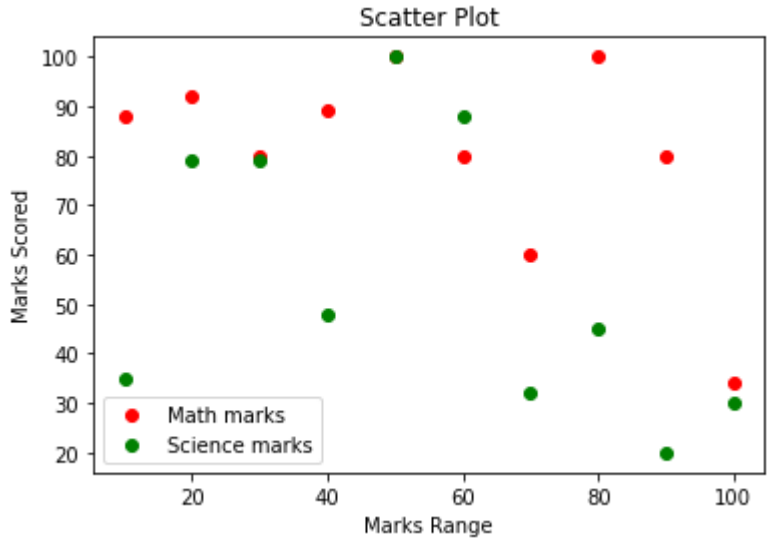
Write a Python program to draw a scatter plot using random distributions to generate balls of different sizes.

In [21]:
```
import math
import random
import matplotlib.pyplot as plt
# create random data
no_of_balls = 25
x = [random.triangular() for i in range(no_of_balls)]
y = [random.gauss(0.5, 0.25) for i in range(no_of_balls)]
colors = [random.randint(1, 4) for i in range(no_of_balls)]
areas = [math.pi * random.randint(5, 15)**2 for i in range(no_of_balls)]
# draw the plot
plt.figure()
plt.scatter(x, y, s=areas, c=colors, alpha=0.85)
plt.axis([0.0, 1.0, 0.0, 1.0])
plt.xlabel("X")
plt.ylabel("Y")
plt.show()
```



Write a Python program to draw a scatter plot comparing two subject

In [22]:
```
import matplotlib.pyplot as plt
import pandas as pd
math_marks = [88, 92, 80, 89, 100, 80, 60, 100, 80, 34]
science_marks = [35, 79, 79, 48, 100, 88, 32, 45, 20, 30]
marks_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(marks_range, math_marks, label='Math marks', color='r')
plt.scatter(marks_range, science_marks, label='Science marks', color='g')
plt.title('Scatter Plot')
plt.xlabel('Marks Range')
plt.ylabel('Marks Scored')
plt.legend()
plt.show()
```



In [ ]:

In [ ]:

In [ ]: