

---

# **Matplotlib**

# **Notes by Durga Sir**

---

---

## Table of Contents

<b>Chapter-1 .....</b>	<b>10</b>
<b>Introduction to Matplotlib.....</b>	<b>10</b>
Matplotlib Introduction .....	10
Need of Data visualization.....	10
Basic Introduction to Matplotlib.....	10
Installing Matplotlib .....	11
<b>Chapter-2.....</b>	<b>12</b>
<b>Line Plot- Basics.....</b>	<b>12</b>
Types of Plots .....	12
Matplotlib Anatomy .....	13
Line plots .....	13
Creation of line plot by passing 2 nd-arrays.....	15
What is figure?.....	17
How to add title to the line plot.....	17
How to add xlabel and ylabel to the line plot.....	17
<b>Chapter -3.....</b>	<b>19</b>
<b>Line Plots-Advanced .....</b>	<b>19</b>
Line properties .....	19
Marker property.....	19
Linestyle Property.....	20
color property .....	25
default color .....	27
Shortcut way to set color, marker and line style.....	28
alpha property .....	33
linewidth and marker size.....	34
Components of Line plot .....	37
Sequence of Activities of plot() function.....	37
How to customize the size of the figure .....	37

---

---

To get all default settings in Matplotlib.....	38
How to save line plot to a file .....	39
To know the supported files types for saving figure object .....	40
Creation of line plot by passing a single ndarray.....	41
Multiple lines on the same Plot .....	42
Shortcut way .....	43
How to customize title properties.....	45
Customization of xlabel and ylabel.....	50
<b>Chapter-4</b> .....	53
<b>How to add grid lines to plot.</b> .....	53
How to add grid lines to plot .....	53
Various cases of grid() function usage.....	53
which property .....	57
Difference between major and minor grid lines: .....	60
axis property.....	60
Passing other keyword arguments .....	62
<b>Chapter-5</b> .....	64
<b>Adding Legend.</b> .....	64
Legend.....	64
legend().....	64
legend(labels) .....	65
legend(handles, labels): .....	66
How to adjust legend location .....	69
How to specify number of columns in the legend.....	70
Adding title to legend.....	71
How to add legend outside of the plot.....	72
<b>Chapter-6</b> .....	74
<b>Customization of Tick Location and Labels</b> .....	74
Customization of tick location and labels .....	74

---



---

Without customizing xticks() and yticks() → default values are based on the input.....	75
customizing xticks() .....	77
customizing both xticks() and yticks().....	79
disabling xticks() or yticks() .....	82
<b>Chapter-7 .....</b>	<b>84</b>
<b>How to set limit range of values on x-axis and y-axis by using xlim() and ylim() functions.....</b>	<b>84</b>
How to set limit range of values on x-axis and y-axis .....	84
xlim() .....	85
ylim() .....	91
<b>Chpater-8 .....</b>	<b>92</b>
<b>How to set scale of x-axis and y-axis?.....</b>	<b>92</b>
Scaling: How to set scale for x-axis and y-axis: .....	92
Linear scaling.....	93
Logarithmic scaling.....	94
How to customize base value in logarithmic scaling:.....	95
<b>Chapter-9 .....</b>	<b>97</b>
<b>Plotting Styles.....</b>	<b>97</b>
Plotting styles.....	97
To know the plotting styles available in matplotlib.....	97
How to use the style.....	97
Default styling.....	100
<b>Chapter-10 .....</b>	<b>101</b>
<b>Functional/Procedural Oriented Vs Object Oriented Approached of plotting .....</b>	<b>101</b>
Approaches to create Plot.....	101
Procedural or Functional Oriented Approach.....	101
OOP Approach.....	103
Setp1: Creation of Figure object: .....	104

---

---

Step2: Creation of Axes object: .....	104
Step3: Plot the graph:.....	106
Step 4: Set the properties of the axes.....	106
<b>Chapter-11 .....</b>	<b>109</b>
<b>Bar Chart/ Bar Graph/ Bar Plot .....</b>	<b>109</b>
Bar Chart/Bar Graph/Bar Plot.....	109
Simple bar chart/vertical bar chart:.....	109
Represent the number of movies of each hero by using bar chart .....	110
Mobile Sales of Nokia Company from 2011 to 2020 .....	121
How to add labels to the bar .....	122
pyplot.text().....	122
Adding Labels for the data points of lineplot: .....	122
pyplot.annotate() .....	124
How to add labels to the bar chart: .....	126
With more readable labels.....	127
Plotting bar chart with data from csv file .....	129
Horizontal bar chart: .....	132
Stacked Bar chart: .....	136
Clustered Bar chart/Grouped Bar Chart/Multiple Bar Chart:.....	142
<b>Chapter-12 .....</b>	<b>148</b>
<b>Pie Chart .....</b>	<b>148</b>
Pie Chart .....	148
Adding Labels ==> labels argument.....	149
autopct .....	150
explode.....	154
shadow .....	155
colors .....	156
VIBGYOR colors.....	157
startangle.....	159

---

---

counterclock .....	160
Adding legend .....	161
wedgeprops.....	162
edgecolor.....	162
edgecolor and linestyle .....	163
edgecolor, linestyle and linewidth.....	165
<b>Chapter-13 .....</b>	<b>166</b>
<b>Histogram .....</b>	<b>166</b>
Histograms.....	166
How to change color of bars in histogram .....	172
How to change color of the each bars in histogram .....	174
To change color of first bar .....	177
To change color of each bar .....	178
How to get the real data sets.....	179
How many movies and TV shows are released on Netflix year-wise .....	180
Number of movies and TV shows are released on Netflix from 2000 to 2021 .....	182
Number of movies and TV shows are released on Netflix from 2000 to 2021 only from 'India' .....	183
Number of movies and TV shows released in past decade .....	184
<b>Chapter-14 .....</b>	<b>186</b>
<b>Scatter Plots .....</b>	<b>186</b>
Scatter plot .....	186
Changing the color and size of the markers .....	189
How to change color of the marker .....	189
Different colors for each markers.....	190
Changing the size of the marker.....	191
Different sizes for the markers.....	192
cmap ==> colormap .....	192

---



---

colormap ==> viridis.....	194
colormap ==> summer.....	195
colormap ==> winter.....	196
colormap ==> cool.....	197
colormap ==> hot.....	198
colormap ==> rainbow.....	199
colormap ==> rainbow_r ==> reverse of rainbow.....	200
colormap ==> prism.....	201
colormap ==> flag.....	202
usage of keyword argument c with cmap.....	203
usage of keyword argument c without cmap.....	204
Line plot vs scatter plot: .....	206
How to add labels to scatter plot data points:.....	206
using plt.text(x,y,text).....	207
using plt.annotate(text,(x,y)).....	208
How to add color legend to the scatter plot.....	209
Kaggle Case Study : Latest Covid-19 India Statewise Data.....	212
<b>Chapter-15 .....</b>	<b>216</b>
<b>Subplots.....</b>	<b>216</b>
Subplots.....	216
Need of subplots.....	220
How to create subplots.....	220
pyplot.subplot() function .....	220
pyplot.subplots() function.....	232
<b>Chapter-16 .....</b>	<b>243</b>
<b>Plotting Geographic Data with Basemap .....</b>	<b>243</b>
Plotting Geographic data with Basemap:.....	243
How to install basemap.....	243
Important Theoretical Terminology: .....	248

---



---

1. projection .....	248
2. Resolution: .....	249
3. Latitude and Longitude:.....	249
Important methods of Basemap.....	251
1. Methods for physical boundaries and bodies of water:.....	251
2. Methods for political boundaries:.....	251
3. Methods for Map features: .....	251
4. Methods for whole-globe images:.....	251
How to select particular area of the map.....	269
selection of full world map.....	270
selection of upper half of worldmap .....	272
selection of lower half of worldmap.....	273
selection of left half of worldmap.....	274
selection of right half of worldmap .....	276
How to select only India.....	278
Locate Delhi on India Map: .....	280
Top-5 Tourist Places in India: .....	283
Top-5 Tourist Places in India: with colormap.....	284
Top-5 Tourist Places in India: with color legend .....	286
Top-5 IITs in India.....	287
Top-5 IITs in India: with colormap.....	289
Top 10 states with the highest total covid-19 cases in India.....	291
Top 10 states with the highest total covid-19 cases in India ➔ Reading data from csv file .....	293
Top 5 cities with more covid cases in india: .....	295
<b>Chapter-17 .....</b>	<b>297</b>
<b>Three-Dimensional(3-D) Plotting in Matplotlib.....</b>	<b>297</b>
Three-Dimesional plotting(3-D) in Matplotlib: .....	297
Creating of 3-D axes object .....	297

---

---

Creation of 3-D line plot.....	298
Creation of 3-D Scatter plot: .....	301
Creation of 3-D Bar charts:.....	302
Wireframes and surface plots.....	305
Surface plots.....	307
<b>Chapter-18 .....</b>	<b>309</b>
<b>Animations .....</b>	<b>309</b>
Animations.....	309
Reuse the same line object with different data sets.....	316
Demo program for falling line animation.....	317
Line falling down and up animation:.....	318
Demo Program for Growing Line Animation.....	319
Demo Program for Burning Coil Animation.....	320
How to save animations to a file.....	321
Rain Simulation from matplotlib documentation .....	322

---

---

## Chapter-1

# Introduction to Matplotlib

### Matplotlib Introduction

- ✓ **Numpy** ---> Data Analysis Library
- ✓ **Pandas** ---> Data Analysis Library/Visualization library
- ✓ **Matplotlib/Seaborn/Plotly** ---> Data Visualization Libraries

### Need of Data visualization

- ✓ Data can be represented either in text form or in graphical form
- ✓ Data visualization is the representation of data in visual format.

### Advantages

- ✓ We can compare very easily.
- ✓ We can identify relationships very easily.
- ✓ We can identify symmetry and patterns between data.
- ✓ We can analyze very easily. etc

### There are multiple python based data visualization libraries:

- ✓ **Matplotlib**
- ✓ **Seaborn**
- ✓ **Plotly**

### Basic Introduction to Matplotlib

- ✓ Most popular and oldest data visualization library. Python's alternative to MatLab
  - ✓ It is open source and freeware whereas Matlab is not open source (closed source) and not freeware.
  - ✓ By using this library we can plot data in graphical form very easily. That graphical form can be either 2-D or 3-D.
  - ✓ It is comprehensive library for creating static, animated, and interactive visualizations in python.
  - ✓ **John Hunter** developed matplotlib on top of Numpy and SciPy libraries.
  - ✓ It has very large community support. Every data scientist used this library atleast once in his life.
  - ✓ Advanced libraries like seaborn, plotly are developed on top of matplotlib.
-

- 
- ✓ The official website: <https://matplotlib.org> → Examples tab

## Installing Matplotlib

**There are 2 ways**

- ✓ With Anaconda distribution, this library will be available automatically.  
**conda install matplotlib**
- ✓ In our system, if python is already available, then we can install by using python package manager(pip)  
**pip install matplotlib**

In [1]:

```
# How to check installation
```

```
import matplotlib
print(matplotlib.__version__)
```

3.3.4

### Note

- ✓ we can check the matplotlib installation using  
**pip list**  
**pip freeze**

---

## Chapter-2

### Line Plot- Basics

#### Types of Plots

There are multiple types are available to represent our data in graphical form.

#### The important are:

1. Line Plots
2. Bar charts
3. Pie charts
4. Histogram
5. Scatter plots

Based on input data and requirement, we can choose the corresponding plot.

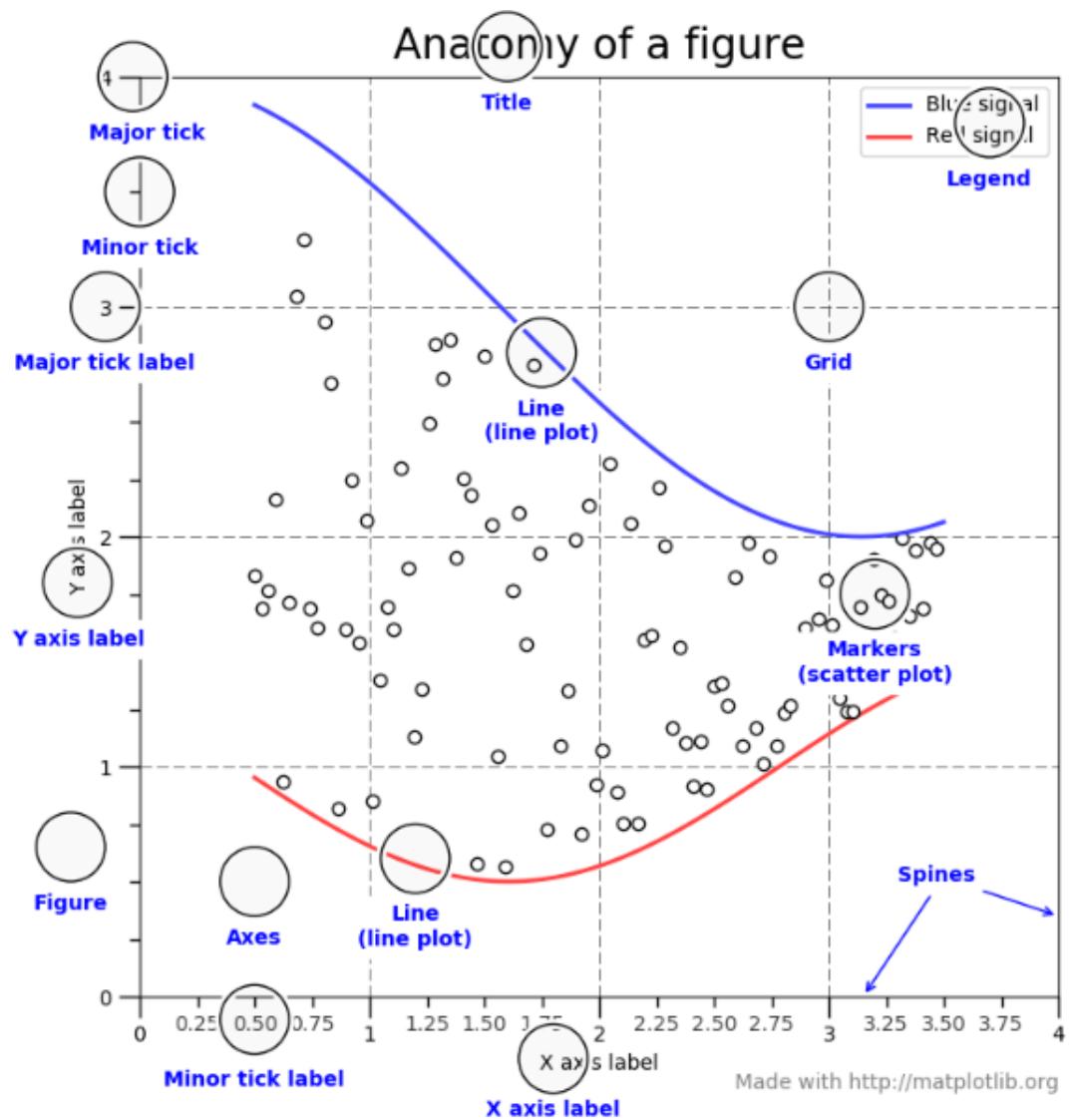
#### Note

- ✓ **matplotlib** ==> package/library
- ✓ **pyplot** ==> module name
- ✓ pyplot module defines several functions to create plots
  - a. **plot()**
  - b. **bar()**
  - c. **pie()**
  - d. **hist()**
  - e. **scatter()**

We can create plots in 2 approaches.

1. **Functional oriented approach (For small data sets)**
2. **Object oriented approach (For larger data sets)**

## Matplotlib Anatomy



### Line plots

- ✓ We can mark data points from the input data and we can connect these data points with lines. Such type of plots are called line plots.
- ✓ We can use line plots to determine the relationship between two data sets.

- 
- ✓ Data set is a collection of values like ndarray,python's list etc  
wickets = [1,2,3,4,5,6,7,8,9,10]  
overs = [1,4,5,...20]
  - ✓ The values from each data set will be plotted along an axis.(x-axis,y-axis)

### **matplotlib.pyplot.plot()**

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

- ✓ **plt.plot()** → To create line plot
- ✓ **plt.bar()** → To create bar chart
- ✓ **plt.pie()** → To create pie chart
- ✓ **plt.hist()** → To create Histograms
- ✓ **plt.scatter()** → To created Scatter plots

```
In [2]:
```

```
import matplotlib.pyplot as plt
help(plt.plot)
```

Help on function plot in module matplotlib.pyplot:

```
plot(*args, scalex=True, scaley=True, data=None, **kwargs)
      Plot y versus x as lines and/or markers.
```

#### **Call signatures::**

```
plot([x], y, [fmt], *, data=None, **kwargs)
plot([x], y, [fmt], [x2], y2, [fmt2], ..., **kwargs)
```

- ✓ **\*args** → any collection of values.This collection of values becomes tuple
- ✓ **\*\*kwargs** → Any collection of key-value pairs. This collection of values will become dict

```
In [3]:
```

```
def function1(*args):
    print(type(args))
function1()
```

```
<class 'tuple'>
```

---

```
In [4]:
```

```
def function2(**kwargs):
    print(type(kwargs))
function2()
```

```
<class 'dict'>
```

## Creation of line plot by passing 2 nd-arrays

### **plt.plot(x,y)**

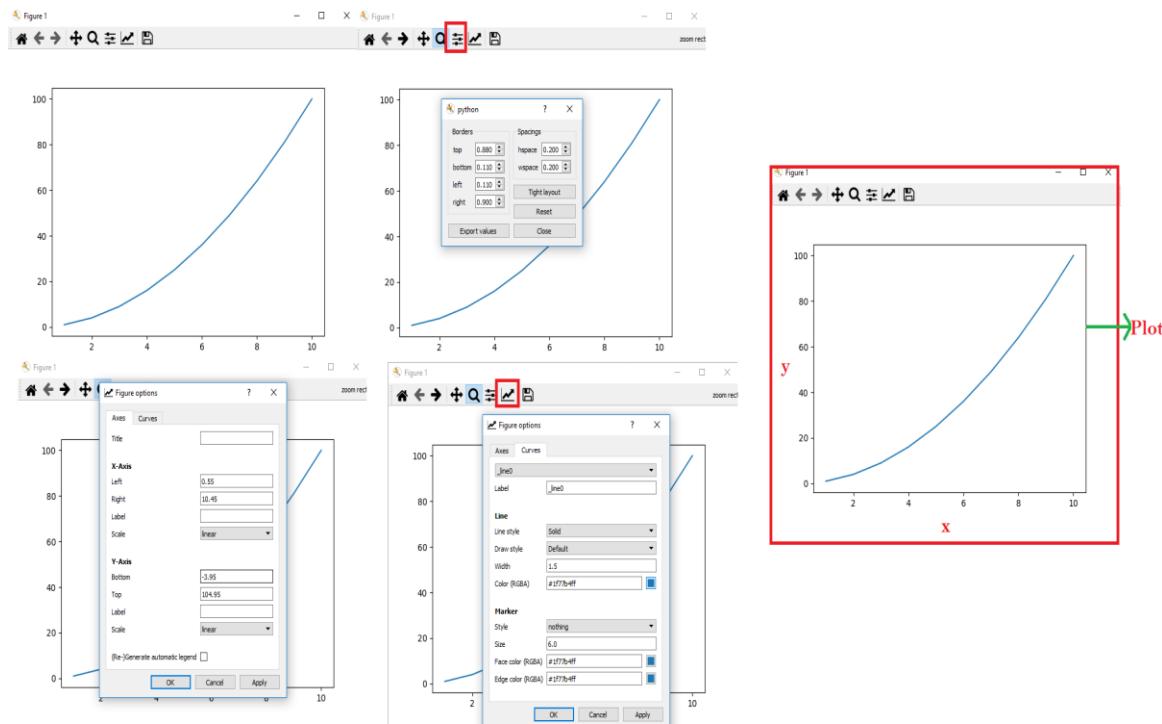
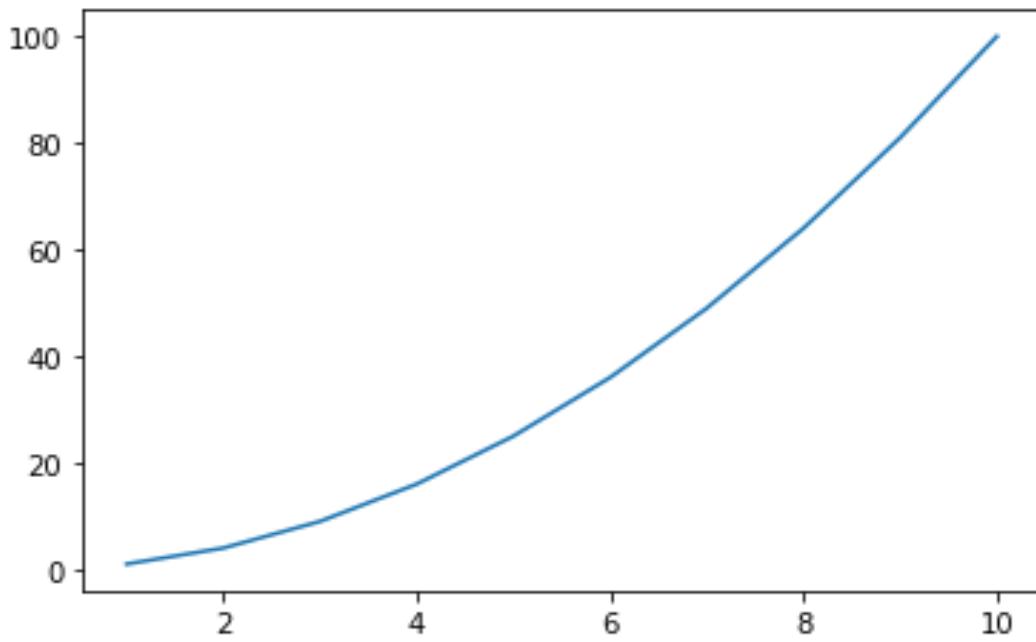
- ✓ The data points will be considered from x and y values.
- ✓ x=[10,20,30]
- ✓ y=[1,2,3]
- ✓ Data points: (10,1), (20,2),(30,3)

```
In [5]:
```

```
# Creation of line plot by passing 2 nd-arrays
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y) #(1,1),(2,4),(3,9)...
plt.show()
```

```
Out[5]:
```

```
[<matplotlib.lines.Line2D at 0x2032d41c8b0>]
```



---

## What is figure?

- ✓ Figure is an individual window on the screen, in which matplotlib displays the graphs i.e., it is the container for the graphical output.
- ✓ **plot()** function is responsible to create this figure object.
- ✓ **figure** → Container for plot
- ✓ figure object contains multiple plots also

## How to add title to the line plot

- ✓ By using **plt.title()** function we can add title to the line plot
- ✓ **plt.title('Square function line plot')**

In [6]:

```
import matplotlib.pyplot as plt
help(plt.title)
```

Help on function title in module matplotlib.pyplot:

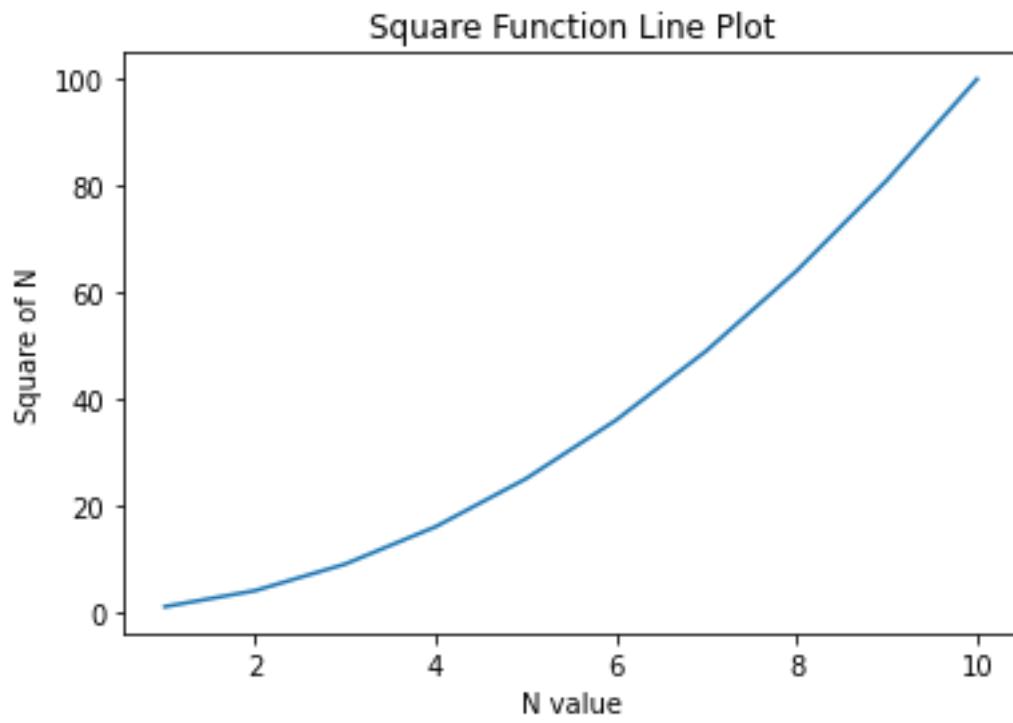
```
title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)
    Set a title for the axes.
```

## How to add xlabel and ylabel to the line plot

- ✓ **plt.xlabel()** function describes information about x-axis data.
- ✓ **plt.xlabel('N value')**
- ✓ **plt.ylabel()** function describes information about y-axis data.
- ✓ **plt.ylabel('Square of N')**

In [7]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y) # (1,1), (2,4), (3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```



#### Note

- ✓ **plt.plot(x,y)** ==> To draw line plot
  - ✓ **plt.title()** ==> To provide title to the line plot
  - ✓ **plt.xlabel()** ==> Describes information about x-axis data
  - ✓ **plt.ylabel()** ==> Describes information about y-axis data
  - ✓ **plt.show()** ==> To show the line plot
-

---

## Chapter -3

### Line Plots-Advanced

#### Line properties

- ✓ A line drawn on the graph has several properties like color, style, width of the line, transparency etc. We can customize these based on our requirement.

#### Marker property

- ✓ We can use marker property to highlight data points on the line plot.
- ✓ We have to use **marker** keyword argument.
- ✓ **plt.plot(a,b,marker='o')** ==> o means circle

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 marker

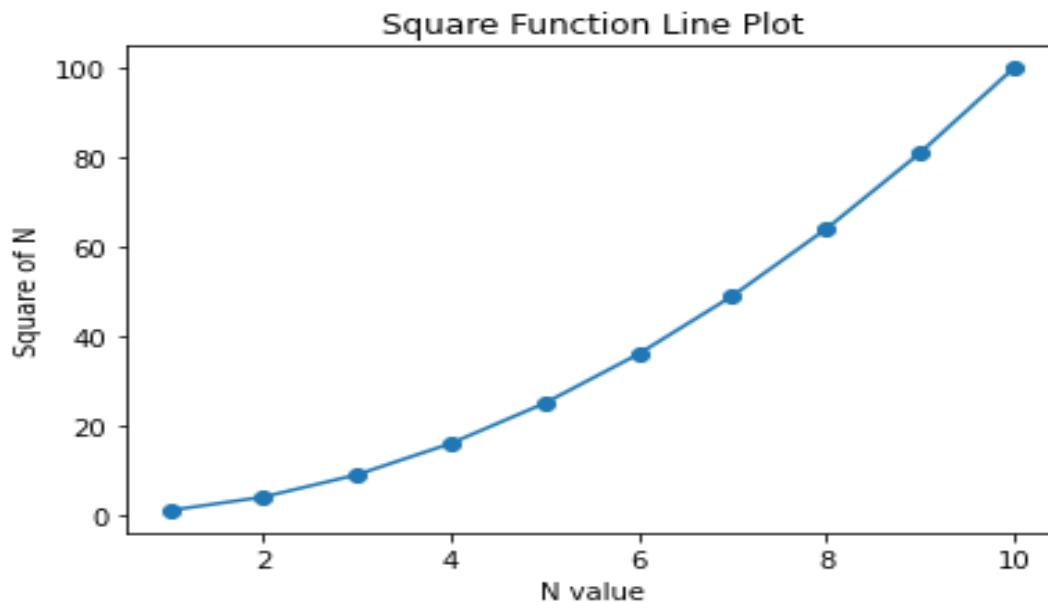
---

In [8]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o') # (1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[8]:

Text(0, 0.5, 'Square of N')



### Linestyle Property

- ✓ Specifies the line style ==> **solid, dotted, dashed**
  - ✓ we can use by using **linestyle** keyword argument  
**plt.plot(a,b,marker='o',linestyle='--')**
-

---

**\*\*Line Styles\*\***

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

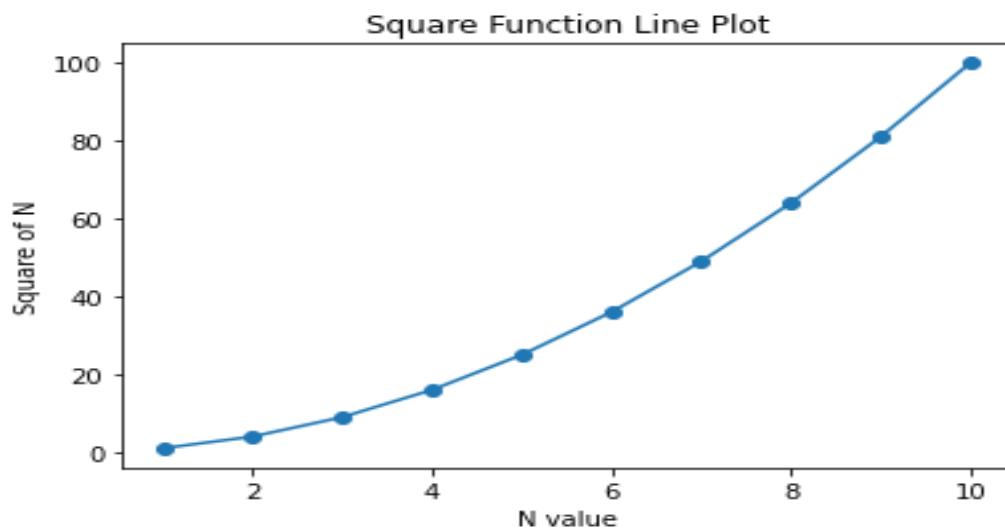
---

In [9]:

```
# Solid line style ==> default
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle='-' ) # (1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[9]:

Text(0, 0.5, 'Square of N')



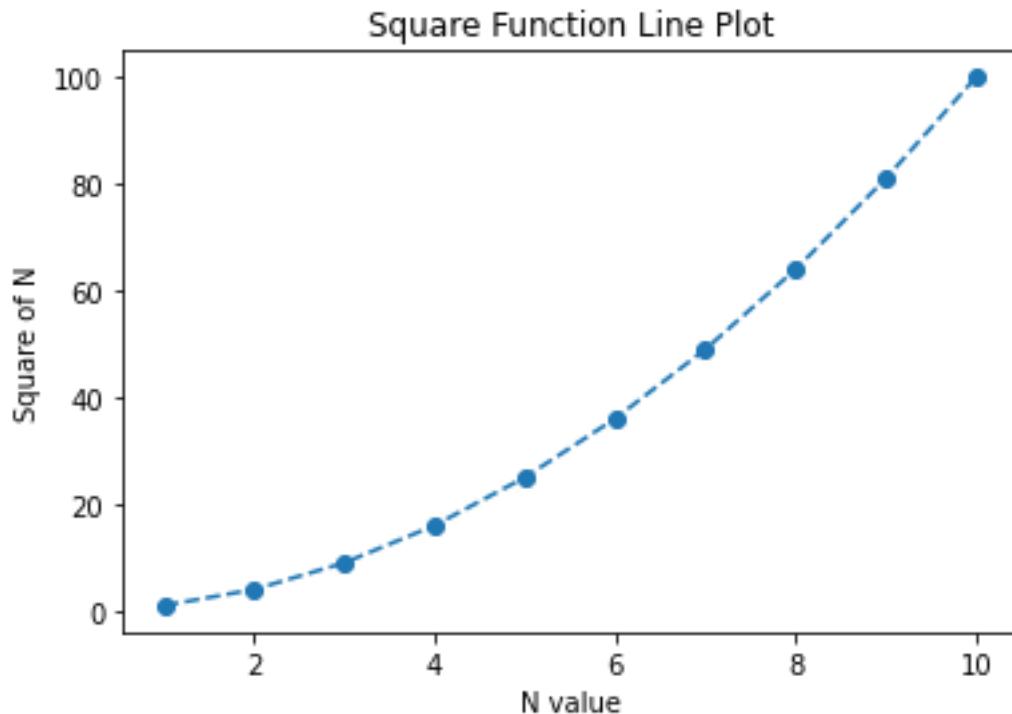
---

In [10]:

```
# dashed line style
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle='--') # (1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[10]:

Text(0, 0.5, 'Square of N')



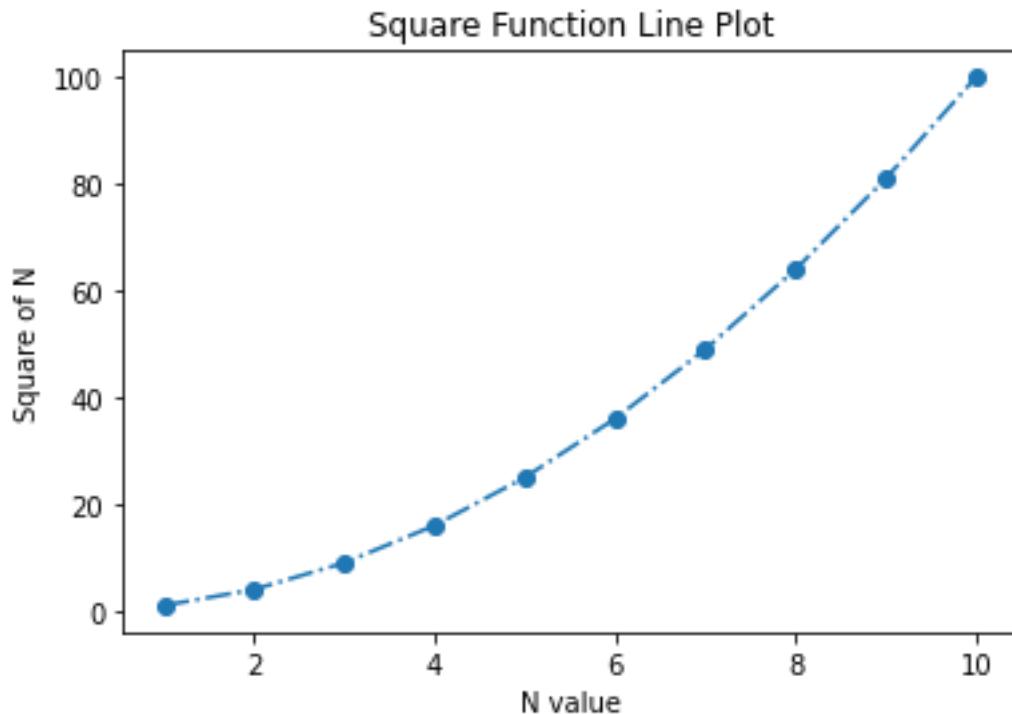
---

In [11]:

```
# dash-dot line style
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle='-.') # (1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[11]:

Text(0, 0.5, 'Square of N')



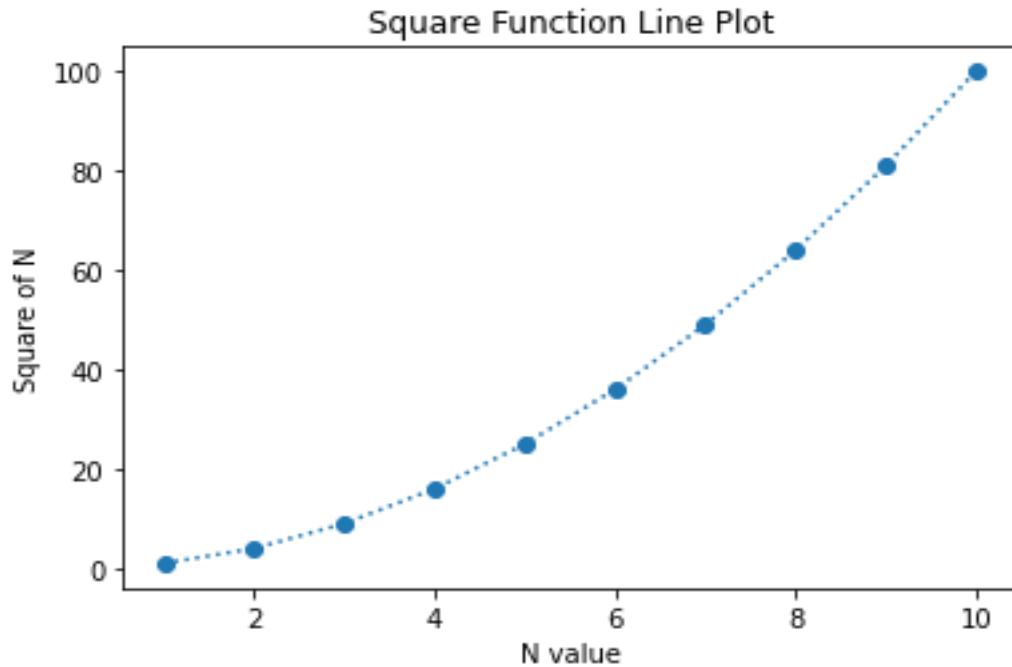
---

In [12]:

```
# dotted line style
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle=':') # (1,1), (2,4), (3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[12]:

Text(0, 0.5, 'Square of N')



---

### color property

- ✓ By using **color** keyword argument we can provide colors to our plot
- ✓ We can specify our required color for the line plot
- ✓ We can use any color even hexa code also.

### matplotlib defines some short codes for commonly used colors

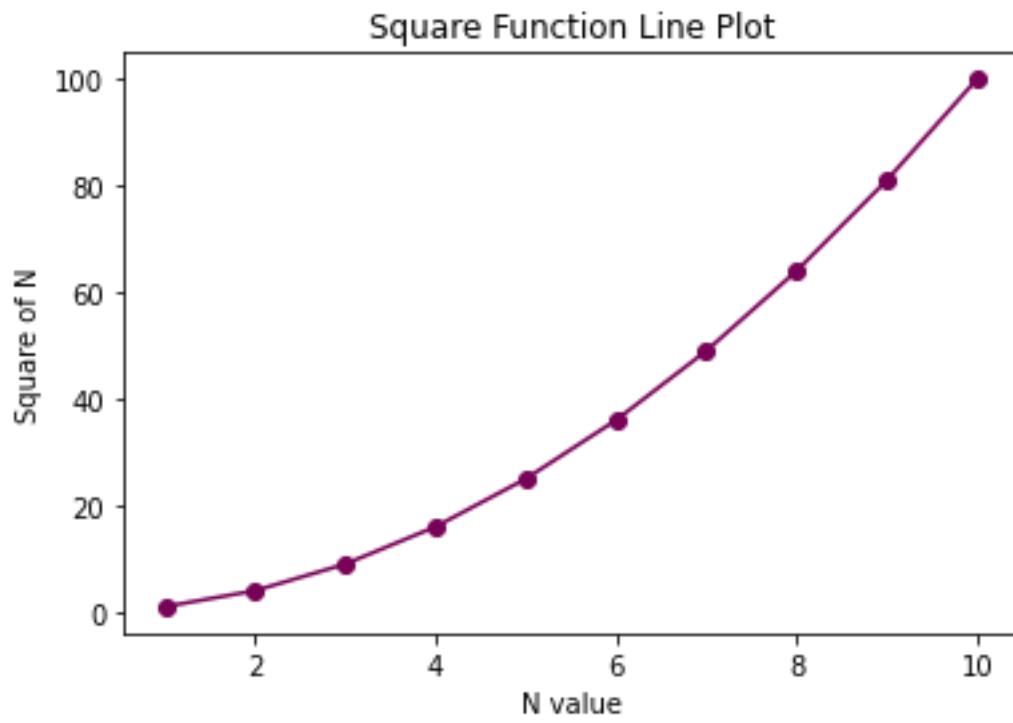
- ✓ '**b**' ==> blue
- ✓ '**g**' ==> green
- ✓ '**r**' ==> red
- ✓ '**c**' ==> cyan
- ✓ '**m**' ==> magento
- ✓ '**y**' ==> yellow
- ✓ '**k**' ==> black
- ✓ '**w**' ==> white

In [13]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle='-',color='#780257') #((1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[13]:

Text(0, 0.5, 'Square of N')



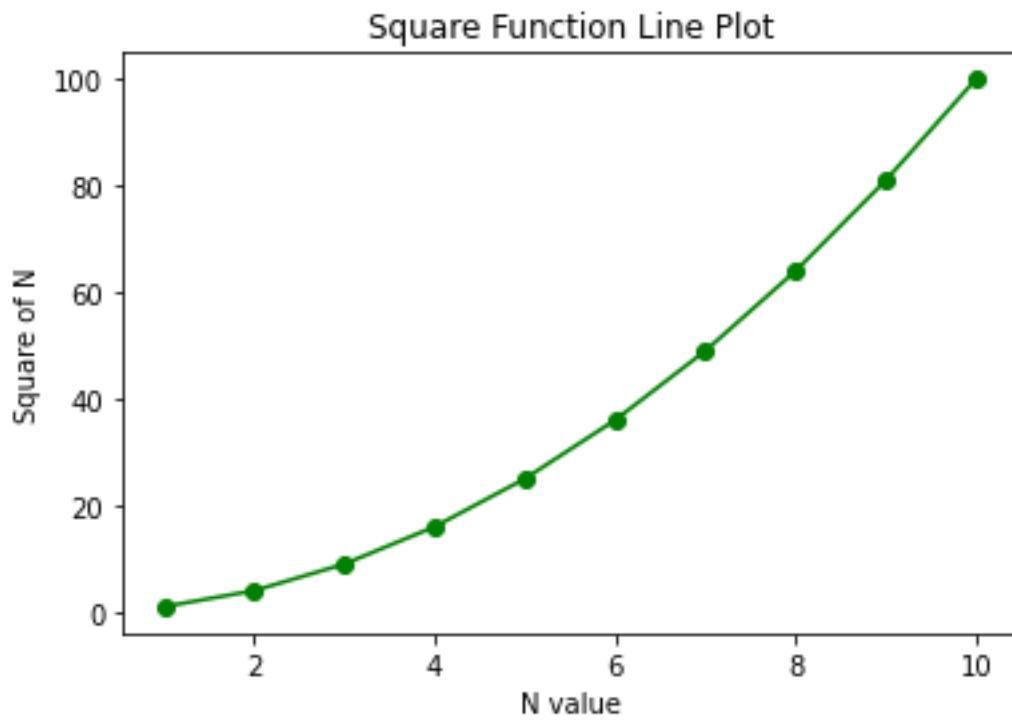
In [14]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,marker='o',linestyle='-',color='green') #((1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[14]:

Text(0, 0.5, 'Square of N')

---



#### default color

- ✓ If we are not specify color then default color will be selected from the style circle
- ✓ To find the default color  
`plt.rcParams['axes.prop_cycle'].by_key()`  
**blue** → first default  
**orange** → second default  
**green** → third default  
**red** → fourth default

In [15]:

```
# default color
import matplotlib.pyplot as plt
print(plt.rcParams['axes.prop_cycle'].by_key())

{'color': ['#1f77b4', '#ff7f0e', '#2ca02c', '#d62728', '#9467bd', '#8c564b',
 '#e377c2', '#7f7f7f', '#bcbd22', '#17becf']}
```

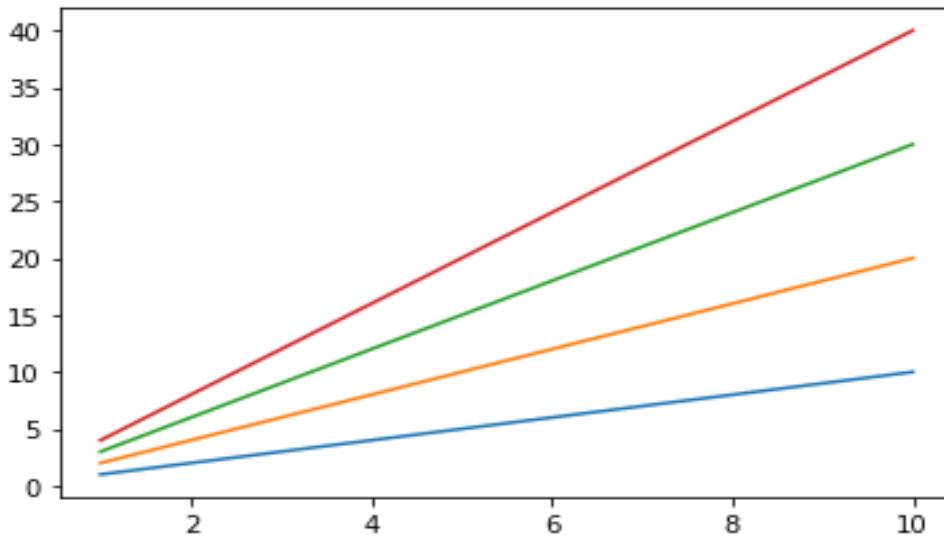
---

In [16]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,11)
plt.plot(x,x) # blue
plt.plot(x,x*2) # orange
plt.plot(x,x*3) # green
plt.plot(x,x*4) # red
plt.show()
```

Out[16]:

```
[<matplotlib.lines.Line2D at 0x2032de623d0>]
```



### Shortcut way to set color, marker and line style

- ✓ We can specify the shortcut notation either **mfc** or **clm**
  - c** → color
  - l** → linestyle
  - m** → marker

### Note

- ✓ In this shortcut way we should use short code for color i.e., **b,g,y,k,c** etc.
  - ✓ The values red,yellow not allowed in shortcut way
-

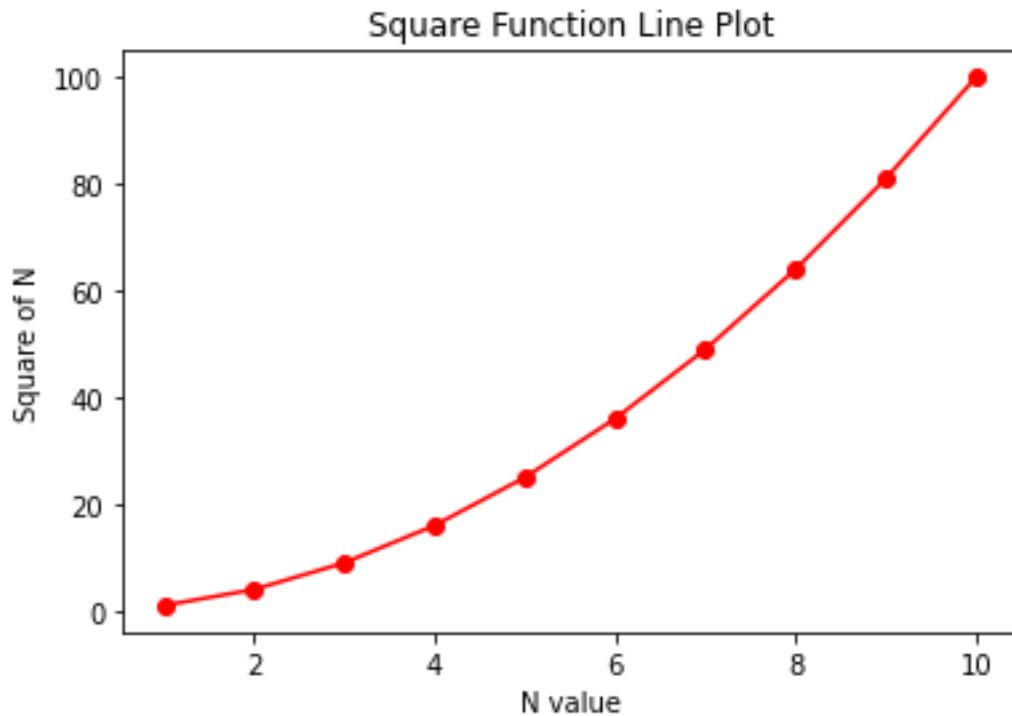
---

In [17]:

```
# mlc form
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,'o-r') # (1,1), (2,4), (3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[17]:

Text(0, 0.5, 'Square of N')



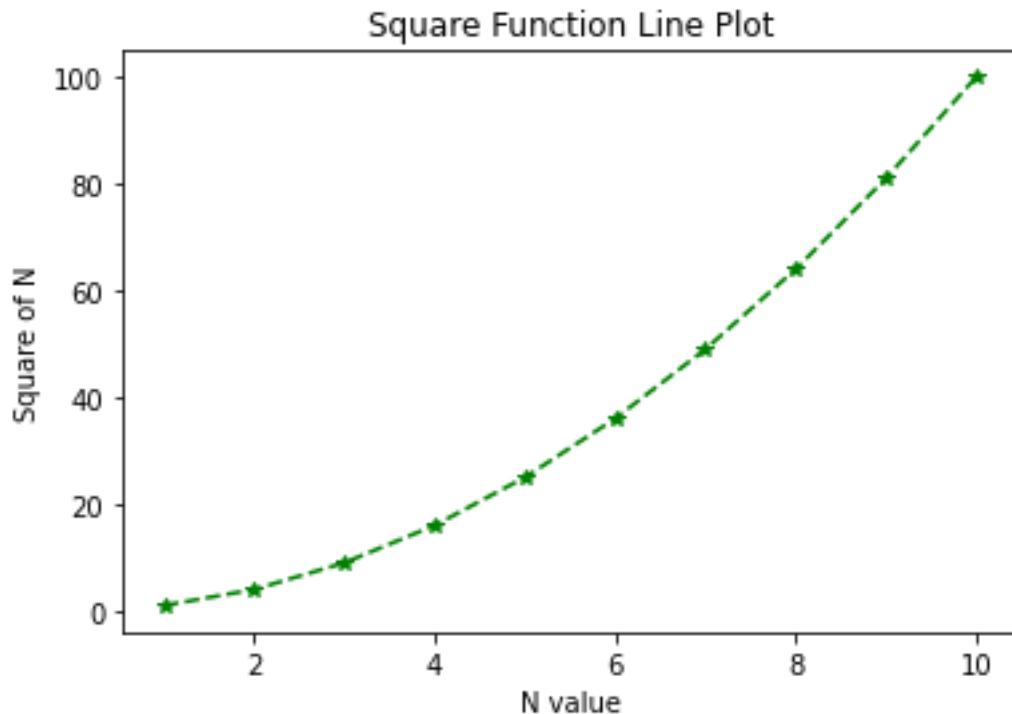
---

In [18]:

```
# clm form
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,'g--*') # (1,1),(2,4),(3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[18]:

Text(0, 0.5, 'Square of N')



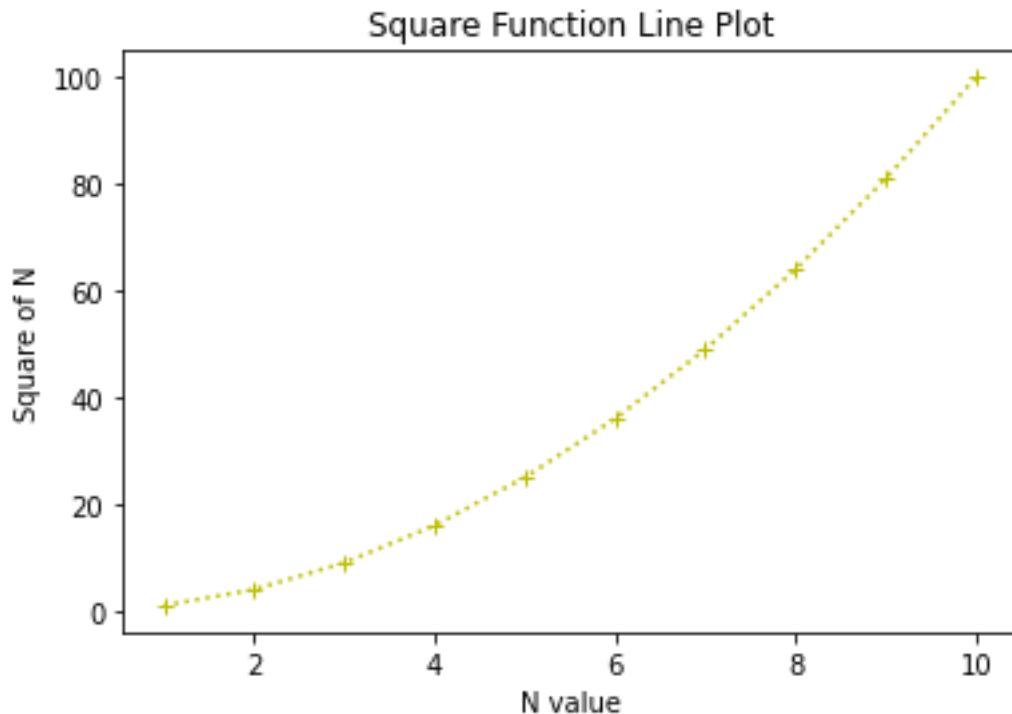
---

In [19]:

```
# mlc form
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,'+y') # (1,1), (2,4), (3,9)...
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[19]:

Text(0, 0.5, 'Square of N')



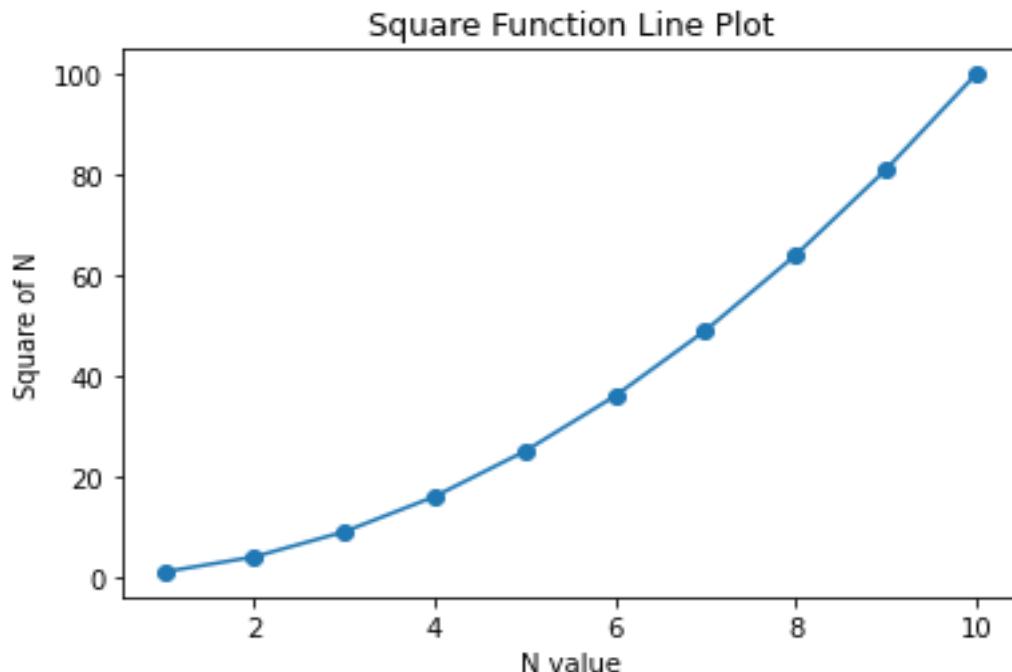
---

In [20]:

```
# mlc form
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
y = x**2
plt.plot(x,y,'o-') # default blue color
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[20]:

Text(0, 0.5, 'Square of N')



<https://rgbacolorpicker.com/>

[https://matplotlib.org/2.0.2/api/lines\\_api.html](https://matplotlib.org/2.0.2/api/lines_api.html)

---

---

### alpha property

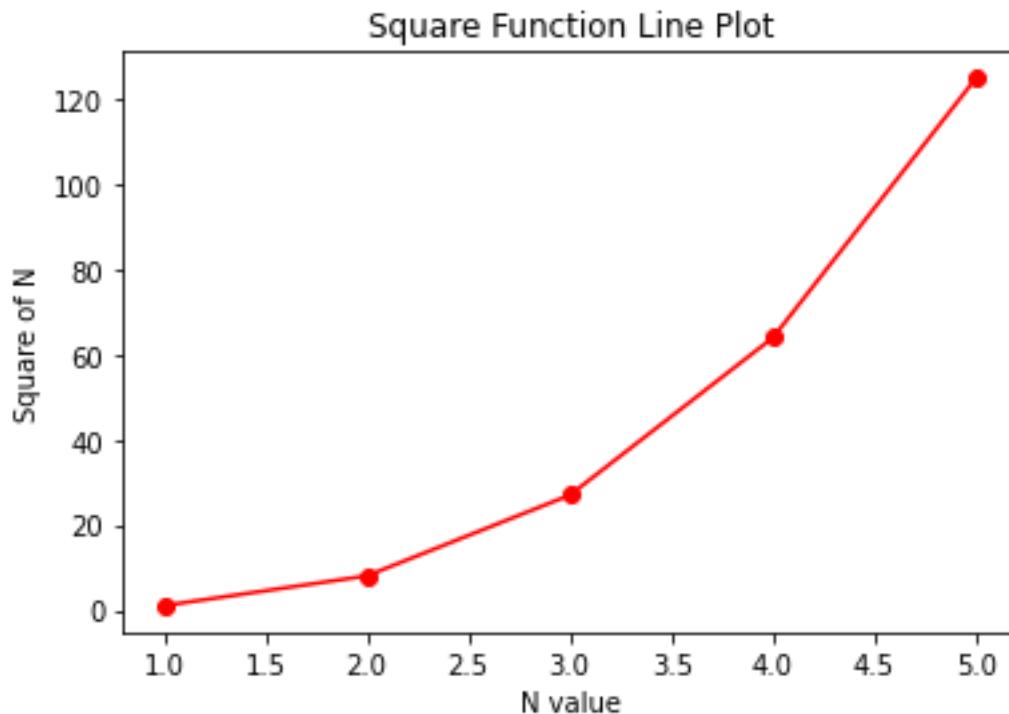
- ✓ denotes **opaque** or **transparency** of the color
- ✓ value lies between **0.0 to 1.0**

In [21]:

```
# alpha property ==> denotes opaque or transparency of the color
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,6)
plt.plot(x,x**3,'o-r',alpha=1.0)
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[21]:

Text(0, 0.5, 'Square of N')



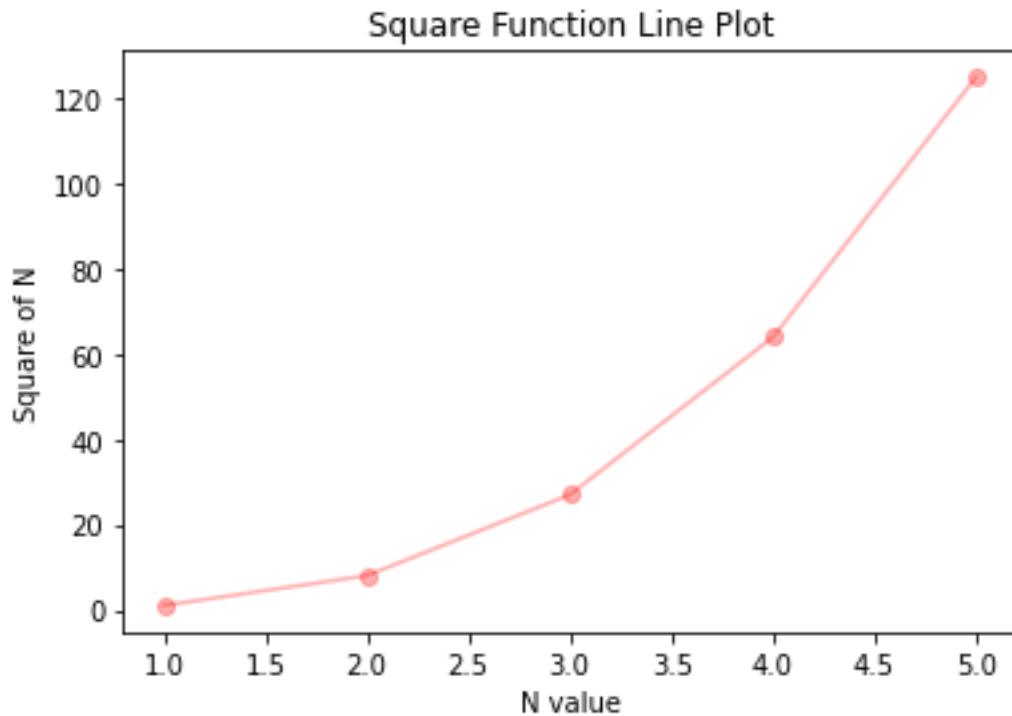
---

In [22]:

```
# alpha property ==> denotes opaque or transparency of the color
# value lies between 0.0 to 1.0
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,6)
plt.plot(x,x**3,'o-r',alpha=0.3)
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[22]:

Text(0, 0.5, 'Square of N')



#### linewidth and marker size

- ✓ **linewidth** denotes the width of the line by using **lw** property
  - ✓ **markersize** denotes the size of the marker by using **ms** property
-

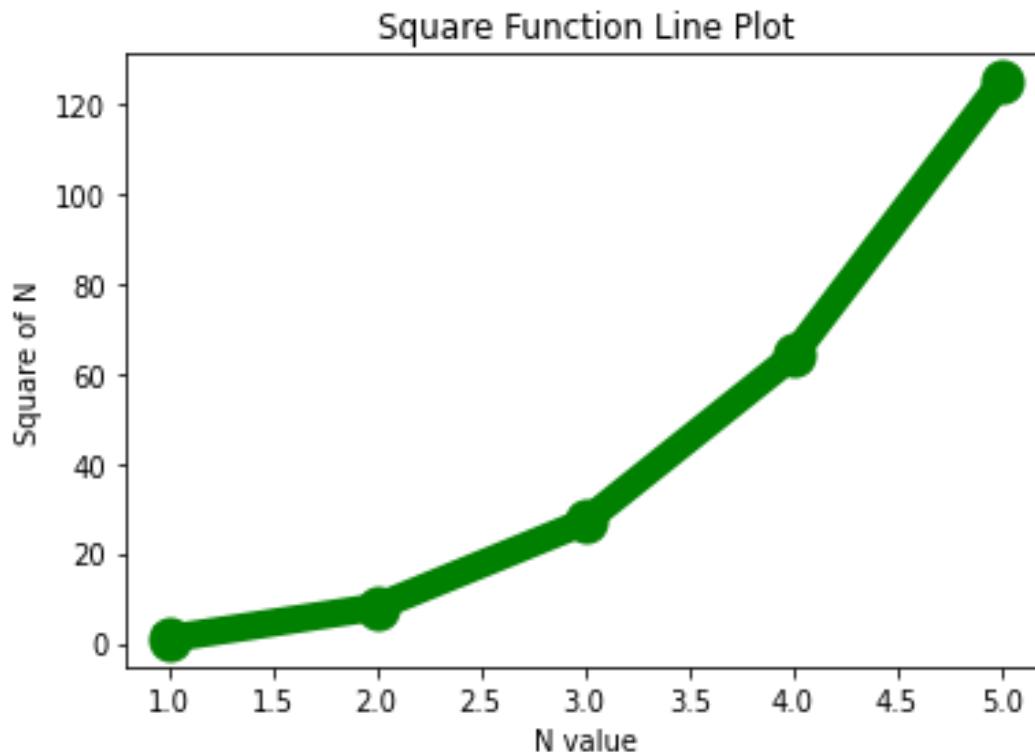
---

In [23]:

```
# linewidth and marker size can be represented by using 'lw' and 'ms'  
import matplotlib.pyplot as plt  
import numpy as np  
x = np.arange(1,6)  
plt.plot(x,x**3,'o-g',lw=10,ms=15) # default blue color  
plt.title('Square Function Line Plot')  
plt.xlabel('N value')  
plt.ylabel('Square of N')  
plt.show()
```

Out[23]:

Text(0, 0.5, 'Square of N')



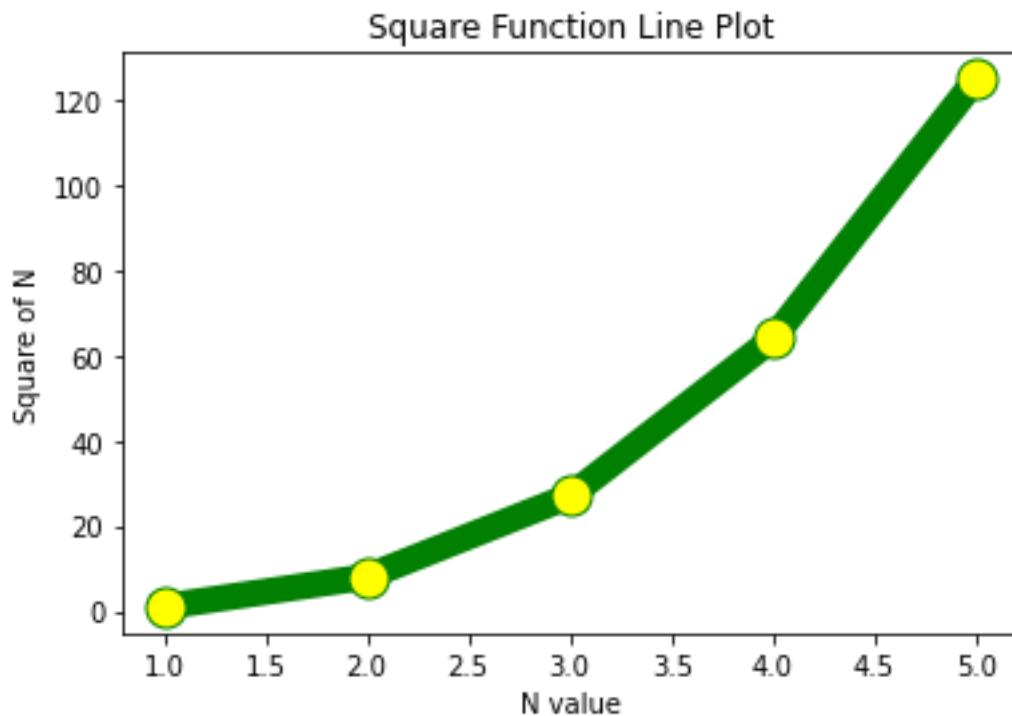
---

In [24]:

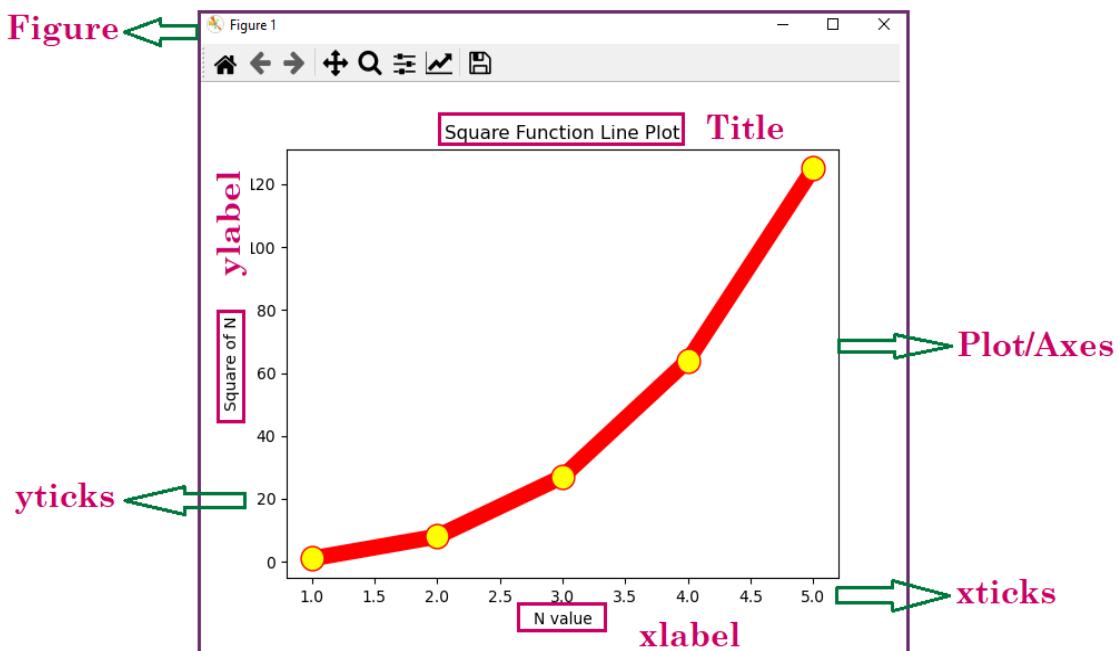
```
# linewidth and marker size can be represented by using 'lw' and 'ms'.
markerfacecolor ==> mfc
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,6)
plt.plot(x,x**3,'o-g',lw=10,ms=15,mfc='yellow') # default blue color
plt.title('Square Function Line Plot')
plt.xlabel('N value')
plt.ylabel('Square of N')
plt.show()
```

Out[24]:

Text(0, 0.5, 'Square of N')



## Components of Line plot



## Sequence of Activities of `plot()` function

- ✓ Creation of **figure** object
- ✓ Creation of **plot/axes** object
- ✓ Draw x & y axis
- ✓ Mark evenly spaced values on x-axis and y-axis (**xticks and yticks**)
- ✓ Plot the data points
- ✓ connect these data points with line
- ✓ Add **title, xlabel, ylabel**

## How to customize the size of the figure

- ✓ The default size of the figure: 8 inches width and 6 inches height.
- ✓ But we can customize based on our requirement. For this we have to use **figure()** function.

---

```
In [25]:
```

```
import matplotlib.pyplot as plt
help(plt.figure)
```

Help on function figure in module matplotlib.pyplot:

```
figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None,
frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False,
**kwargs)
```

Create a new figure, or activate an existing figure.

- **num** ➔ A unique identifier for the figure. Either int or str
- **figsize** ➔ (float, float), default: :rc:figure.figsize Width, height in inches.

### To get all default settings in Matplotlib

- ✓ `print(plt.rcParams)` # it displays all default setting in Matplotlib
- ✓ `print(plt.rcParams.get('figure.figsize'))`

```
In [26]:
```

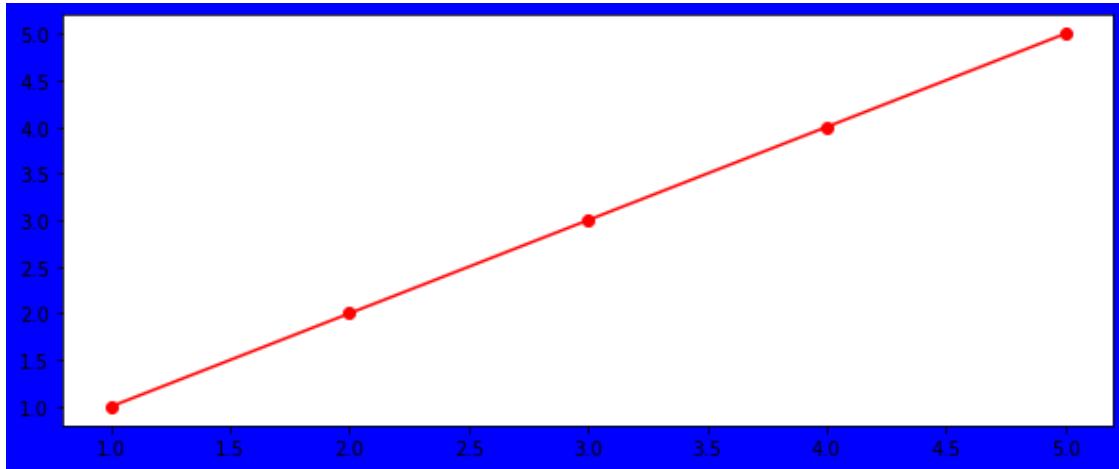
```
# To get the default figure size
import matplotlib.pyplot as plt
print(plt.rcParams.get('figure.figsize'))
# print(plt.rcParams['figure.figsize'])
```

[6.0, 4.0]

```
In [27]:
```

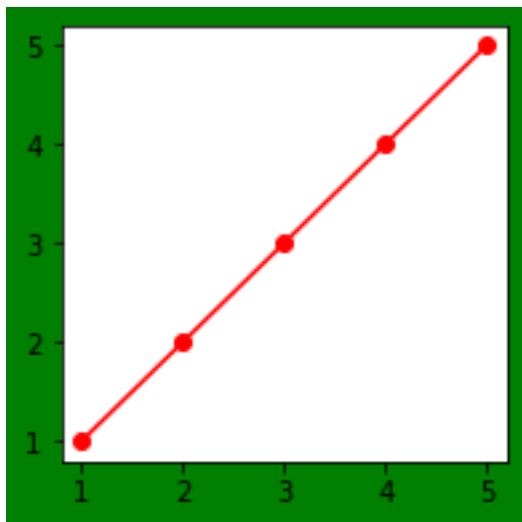
```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(num=1,figsize=(10,4),facecolor='blue')
a = np.arange(1,6)
plt.plot(a,a,'o-r')
plt.show()
```

# matplotlib



In [28]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(3,3),facecolor='g')
a = np.arange(1,6)
plt.plot(a,a,'o-r')
plt.show()
```



## How to save line plot to a file

- ✓ We can save line plot to a file instead of displaying on the screen.
- ✓ We have to use **savefig()** function.

- 
- ✓ By default this figure will be saved in the current working directory. But we can provide any location based on our requirement.

```
plt.savefig('C:\\\\Users\\\\Gopi\\\\Desktop\\\\identitylineplot.jpeg')
```

In [29]:

```
import matplotlib.pyplot as plt  
help(plt.savefig)
```

Help on function savefig in module matplotlib.pyplot:

```
savefig(*args, **kwargs)  
    Save the current figure.
```

Call signature::

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',  
        orientation='portrait', papertype=None, format=None,  
        transparent=False, bbox_inches=None, pad_inches=0.1,  
        frameon=None, metadata=None)
```

## To know the supported files types for saving figure object

In [30]:

```
import matplotlib.pyplot as plt  
fig = plt.figure()  
fig.canvas.get_supported_filetypes()
```

Out[30]:

```
{'eps': 'Encapsulated Postscript',  
'jpg': 'Joint Photographic Experts Group',  
'jpeg': 'Joint Photographic Experts Group',  
'pdf': 'Portable Document Format',  
'pgf': 'PGF code for LaTeX',  
'png': 'Portable Network Graphics',  
'ps': 'Postscript',  
'raw': 'Raw RGBA bitmap',  
'rgba': 'Raw RGBA bitmap',  
'svg': 'Scalable Vector Graphics',  
'svgz': 'Scalable Vector Graphics',
```

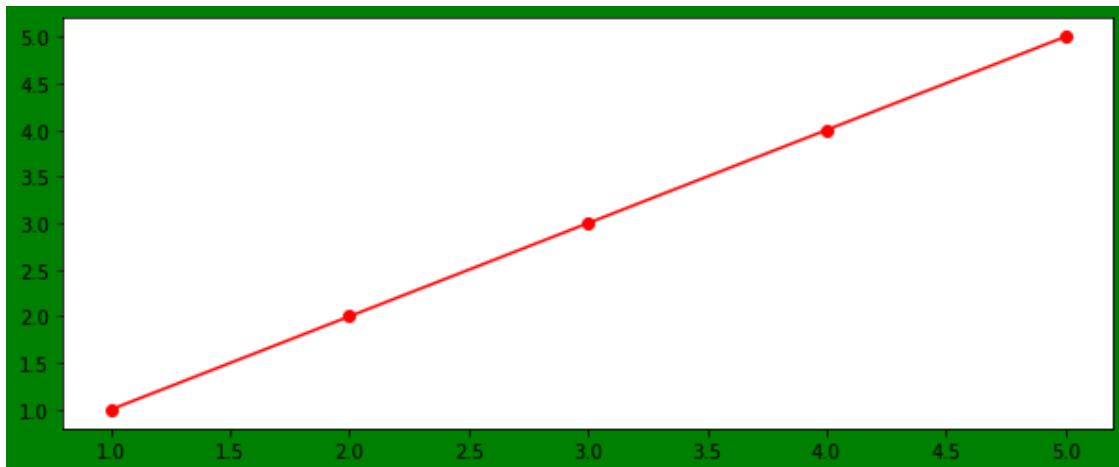
---

```
'tif': 'Tagged Image File Format',
'tiff': 'Tagged Image File Format'}
```

```
<Figure size 432x288 with 0 Axes>
```

In [31]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(num=1,figsize=(10,4),facecolor='green')
a = np.arange(1,6)
plt.plot(a,a,'o-r')
plt.savefig('identitylineplot.png')
```



### Creation of line plot by passing a single ndarray

- ✓ **plt.plot(a,b)** ➔ a for x-axis and b for y-axis.
- ✓ **plt.plot(a)** ➔ a is for y-axis and x-axis values will be generated automatically by matplotlib from 0 to N-1. Where N is size of the datapoints of a

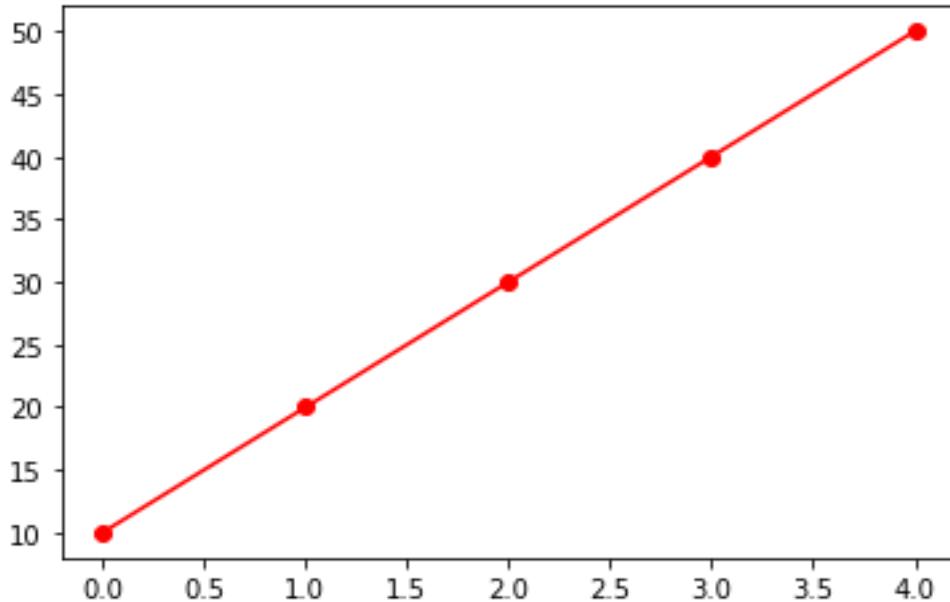
**Eg:**

```
a = np.array([10,20,30,40,50])
plt.plot(a)
0 to 4 will be considered for x-axis.
a values are considered for y-axis
Now the data points are: (0,10),(1,20),(2,30),(3,40),(4,50)
```

---

In [32]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,'o-r')
plt.show()
```

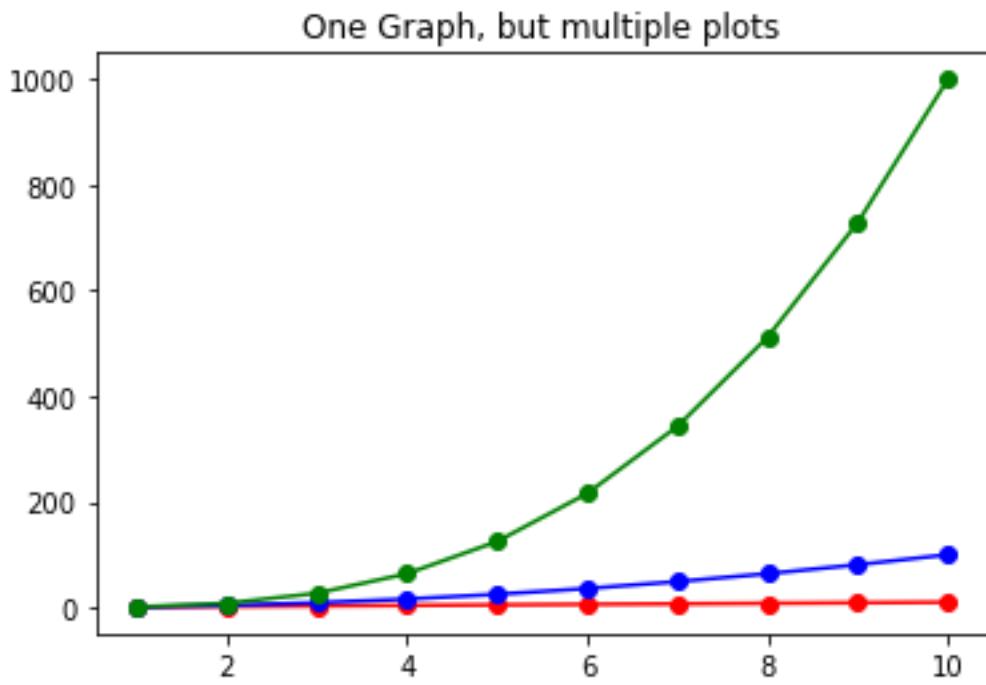


## Multiple lines on the same Plot

In [33]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
i = x
s = x**2
c = x**3
plt.plot(x,i,'o-r')
plt.plot(x,s,'o-b')
plt.plot(x,c,'o-g')
plt.title('One Graph, but multiple plots')
plt.show()
```

---

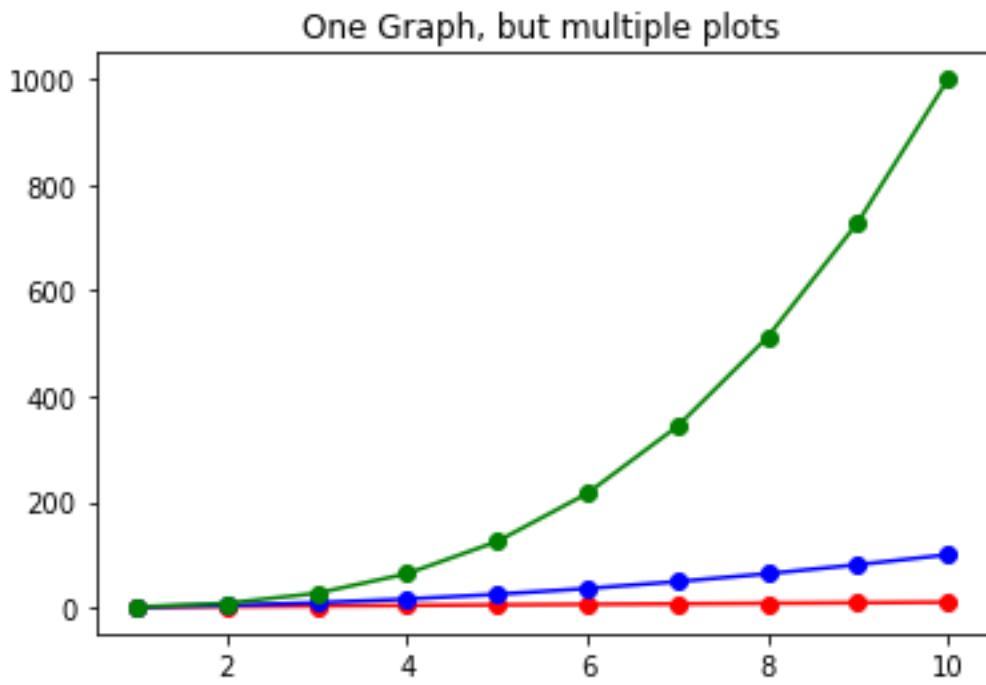


### Shortcut way

We can also use single plot() function for all 3 lines.

In [34]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
i = x
s = x**2
c = x**3
plt.plot(x,i,'o-r',x,s,'o-b',x,c,'o-g')
plt.title('One Graph, but multiple plots')
plt.show()
```

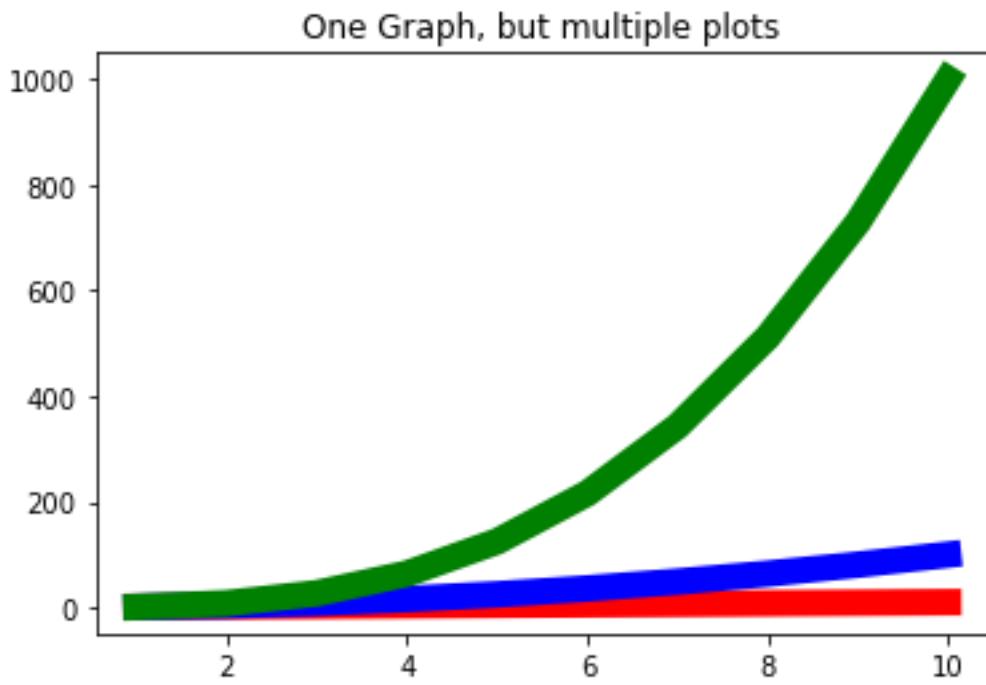


**Note:**

- ✓ `plt.plot(x,i,'o-r',x,s,'o-b',x,c,'o-g',lw=10)`
- ✓ For all the lines the linewidth(`lw`) will be applicable
- ✓ For the **first line**: `x,i,'o-r'`
- ✓ For the **second line** : `x,s,'o-b'`
- ✓ For the **third line** : `x,c,'o-g'`
- ✓ linewidth property is applicable for all 3 lines.

In [35]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
i = x
s = x**2
c = x**3
plt.plot(x,i,'o-r',x,s,'o-b',x,c,'o-g',lw=10)
plt.title('One Graph, but multiple plots')
plt.show()
```



### Note

In above program **lw=10** is common for all the three graphs

### How to customize title properties

In [36]:

```
import matplotlib.pyplot as plt  
help(plt.title)
```

Help on function title in module matplotlib.pyplot:

```
title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs)  
    Set a title for the axes.
```

---

[https://matplotlib.org/stable/tutorials/text/text\\_props.html](https://matplotlib.org/stable/tutorials/text/text_props.html)

**fontdict : dict** ➔ A dictionary controlling the appearance of the title text

- ✓ **family** ➔ [ 'serif' | 'sans-serif' | 'cursive' | 'fantasy' | 'monospace' ]
- ✓ **style** or **fontstyle** ➔ [ 'normal' | 'italic' | 'oblique' ]
- ✓ **size** or **fontsize** ➔ [ size in points | relative size, e.g., 'smaller', 'x-large' ]
- ✓ **weight** or **fontweight** ➔ [ 'normal' | 'bold' | 'heavy' | 'light' | 'ultrabold' | 'ultralight' ]
- ✓ **name** or **fontname** ➔ string e.g., ['Sans' | 'Courier' | 'Helvetica' ...]

**loc : {'center', 'left', 'right'}**

**pad : float, default: :rc:axes.titlepad** ➔ The offset of the title from the top of the Axes, in points.

**\*\*kwargs** ➔ To customize font properties

### Note

a. We can use dictionary properties as well as keyword arguments.

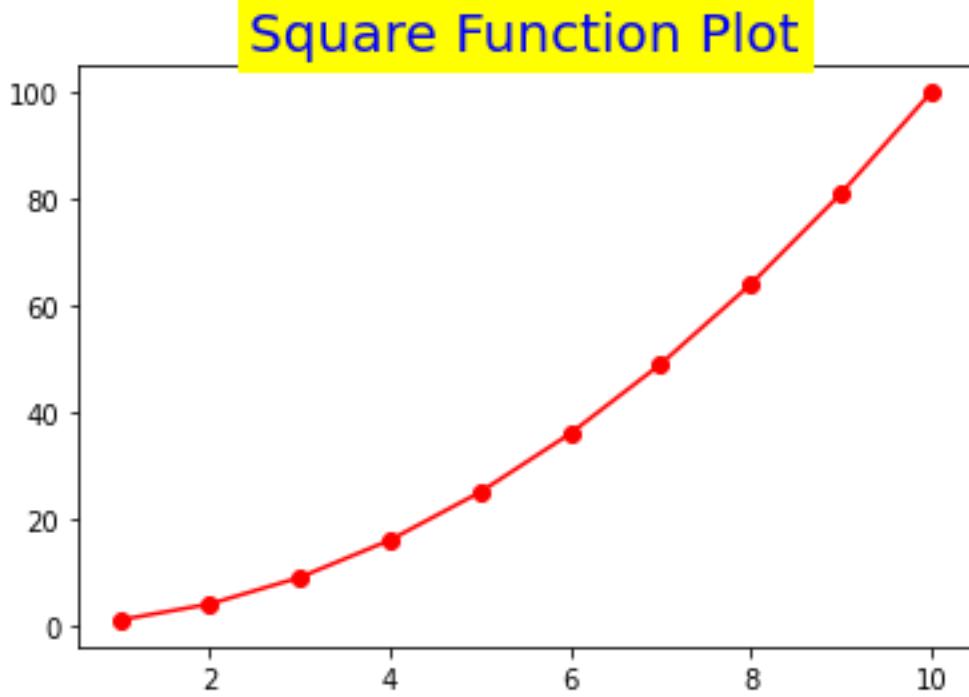
b. If both are provided then keyword arguments are preferred first

plt.title('Square Function Plot',{'color':'b'}) ➔ Title color will be **blue**

plt.title('Square Function Plot',{'color':'b'},color='g') ➔ dictionary and keyword arguments for color are present. So here keyword arguments are taken as preference. So the title will be in **Green color**

In [37]:

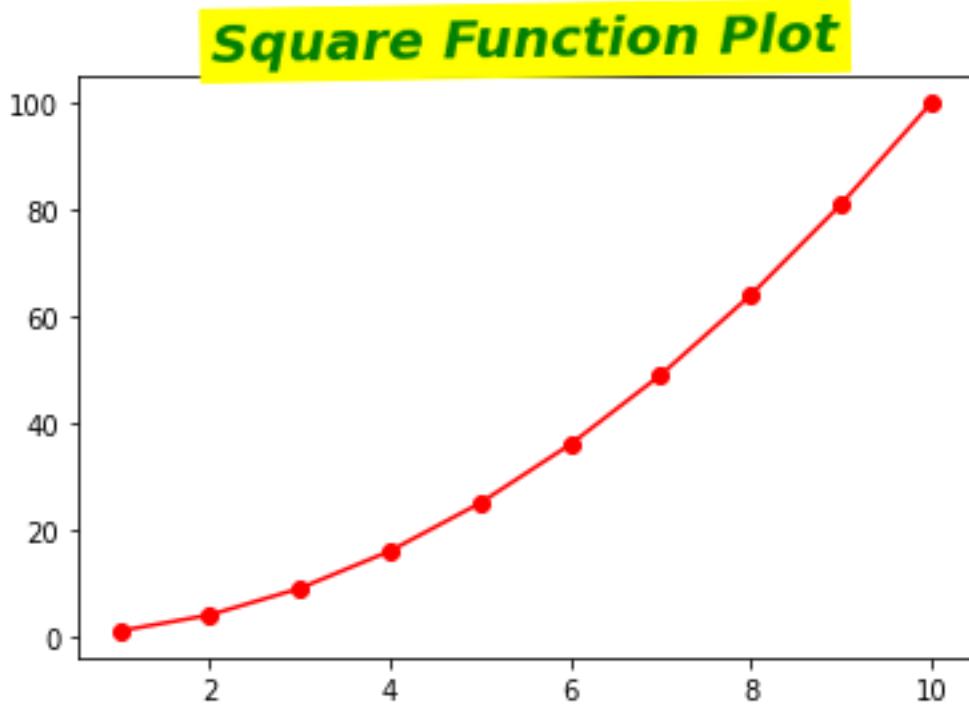
```
# font color,size and backgroundcolor
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
s = x**2
plt.plot(x,s,'o-r')
plt.title('Square Function Plot',{'color':'b','size':20,'backgroundcolor':'yellow'})
plt.show()
```



In [38]:

```
# font color,size,backgroundcolor,fontstyle,family,weight and rotation
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
s = x**2
plt.plot(x,s,'o-r')
plt.title('Square Function
Plot',{'color':'g','size':20,'backgroundcolor':'yellow','alpha':1,
         'fontstyle':'italic','family':'cursive','weight':1000,
         'rotation':1})
plt.show()
```

findfont: Font family ['cursive'] not found. Falling back to DejaVu Sans.

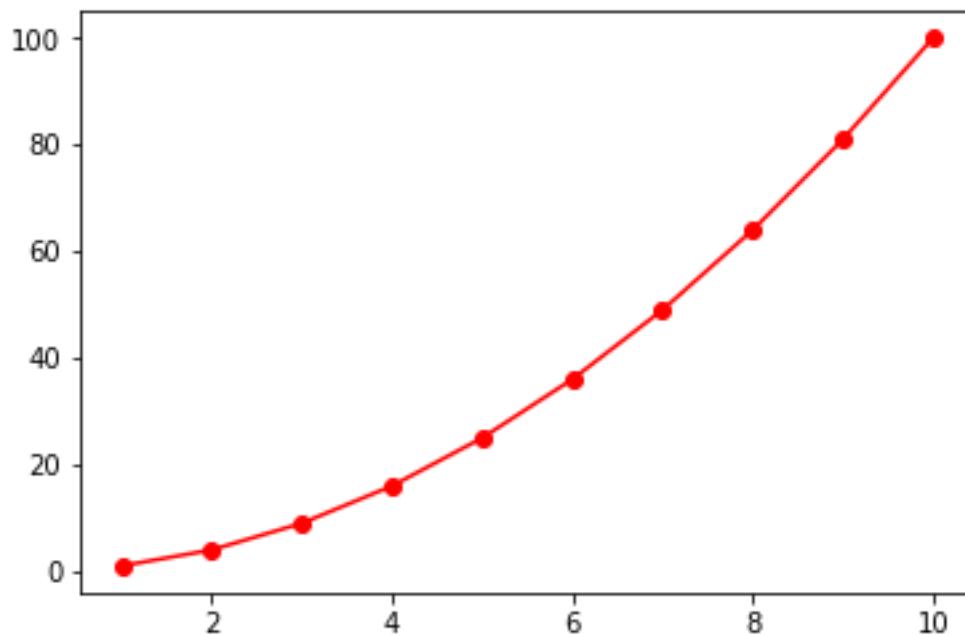


In [39]:

```
# font color,size,backgroundcolor,fontstyle,family,weight and rotation
# loc and pad
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
s = x**2
plt.plot(x,s,'o-r')
plt.title('Square Function
Plot',{'color':'g','size':20,'backgroundcolor':'yellow','alpha':1,
'fontstyle':'italic','family':'fantasy','weight':1000,
'rotation':1}, loc='left',pad=25)
plt.show()
```

---

## *Square Function Plot*

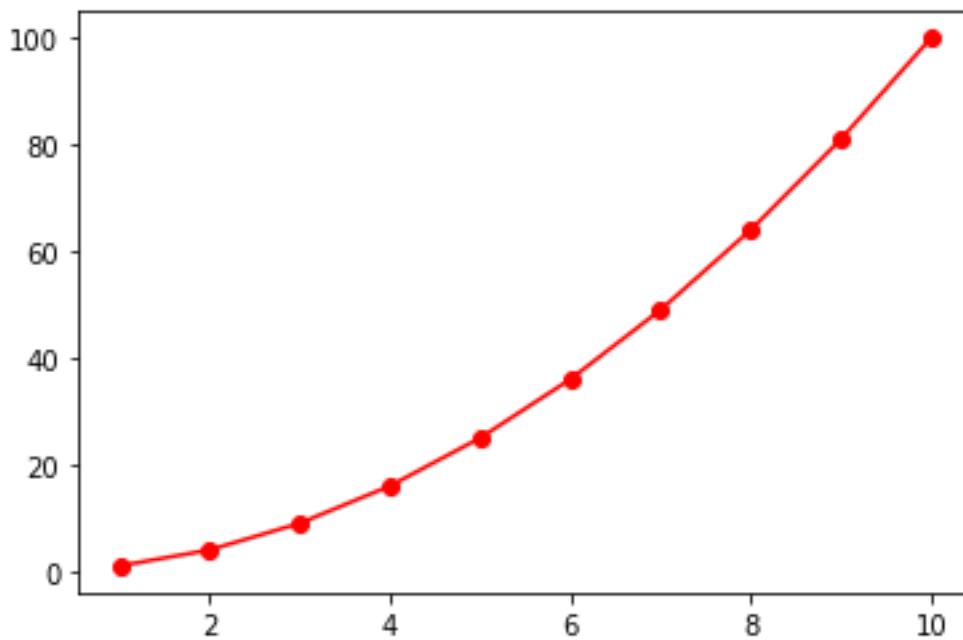


In [40]:

```
# color argument is given in dict as well as keyword argument.
# Here keyword argument will be taken as preference i.e., red
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,11)
s = x**2
plt.plot(x,s,'o-r')
plt.title('Square Function
Plot',{'color':'g','size':20,'backgroundcolor':'yellow','alpha':1,
        'fontstyle':'italic','family':'cursive','weight':1000,
        'rotation':1},loc='left',pad=25,color='red')
plt.show()
```

---

## Square Function Plot



### Customization of xlabel and ylabel

- ✓ exactly same as title customization.
- ✓ `xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)`  
→ Set the label for the x-axis.
- ✓ `ylabel(ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)`  
→ Set the label for the y-axis.

In [41]:

```
import matplotlib.pyplot as plt
help(plt.xlabel)
```

Help on function xlabel in module matplotlib.pyplot:

```
xlabel(xlabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
Set the label for the x-axis.
```

---

---

```
In [42]:
```

```
help(plt.ylabel)
```

Help on function ylabel in module matplotlib.pyplot:

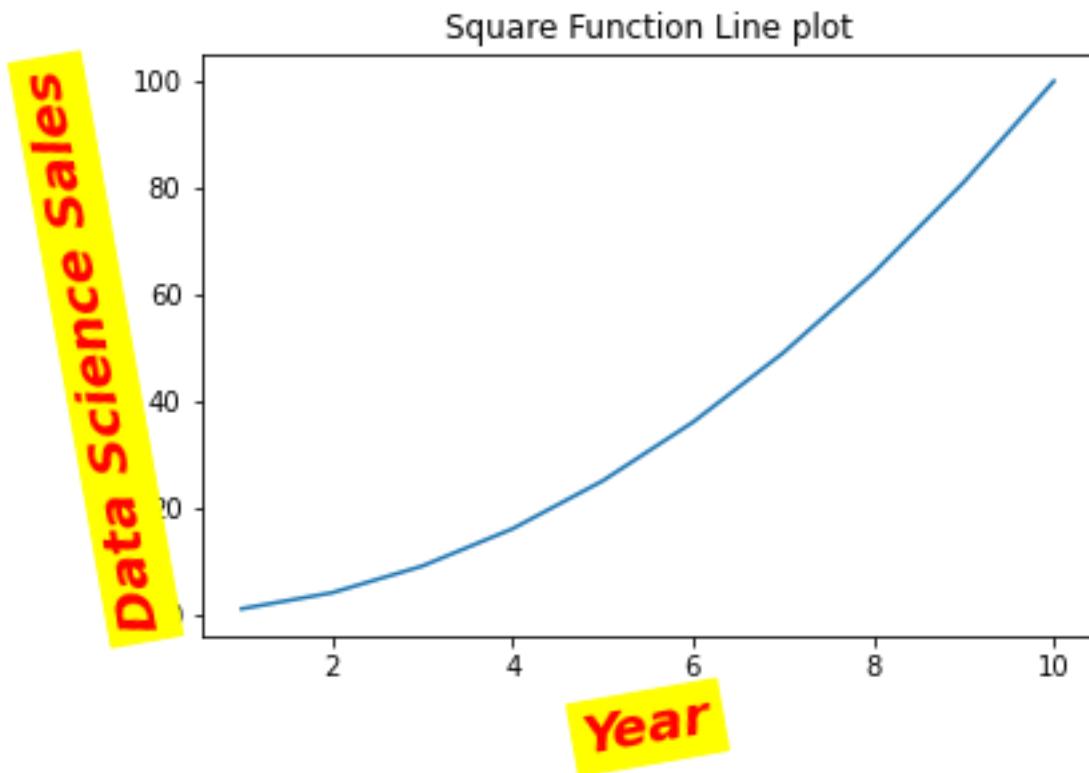
```
ylabel(ylabel, fontdict=None, labelpad=None, *, loc=None, **kwargs)
      Set the label for the y-axis.
```

```
In [43]:
```

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,11)
b = a**2
plt.plot(a,b)
plt.title('Square Function Line plot')
plt.xlabel('Year',
           {'color':'r','size':20,'backgroundcolor':'yellow','rotation':10,'alpha':1,
            'fontstyle':'italic','family':'cursive','weight':1000})
plt.ylabel('Data Science Sales',
           color='r',size=20,backgroundcolor='yellow',rotation=100,
           alpha=1,fontstyle='italic',family='cursive',weight=1000)
# plt.show()
```

Out[43]:

```
Text(0, 0.5, 'Data Science Sales')
```



#### Note

- ✓ `{'color':'r'}` and `color='b'`  
In the case of conflict, **keyword arguments** will get more priority.
- ✓ This is because keyword arguments are provided at last when we are calling a function.
- ✓ Generally latest values are to be considered by the PVM.
- ✓ Here keyword arguments are the latest values
- ✓ **fontdict** properties are **same** for **title, xlabel and ylabel**. These values can be passed as **keyword arguments also**. In the case of conflict, keyword arguments will get more priority

---

## Chapter-4

### How to add grid lines to plot

#### How to add grid lines to plot

- ✓ We can add grid lines to the plot. For this we have to use grid() function.

**plt.grid()**

In [44]:

```
import matplotlib.pyplot as plt
help(plt.grid)
```

Help on function grid in module matplotlib.pyplot:

```
grid(b=None, which='major', axis='both', **kwargs)
Configure the grid lines.
```

- ✓ **plt.grid()** ==> on ==> grid lines are visible
- ✓ **plt.grid()** ==> off ==> grid lines are invisible
- ✓ **plt.grid()** ==> on ==> grid lines are visible
- ✓ **plt.grid()** ==> off ==> grid lines are invisible

#### Note

- ✓ If **b** is **None** and there are no **kwargs**, this toggles the visibility of the lines.
- ✓ default value for b is **None**

#### Various cases of grid() function usage

##### case-1:

- ✓ plt.grid() ==> In this case grid will be visible.

##### case-2:

- ✓ plt.grid()
- ✓ plt.grid()

**grid lines won't be visible**

---

---

### **case-3:**

- ✓ plt.grid()
- ✓ plt.grid(color='g')

**grid lines are visible** ==> because keyword arguments are given

### **case-4:**

- ✓ plt.grid(b=True)

**grid lines are visible**

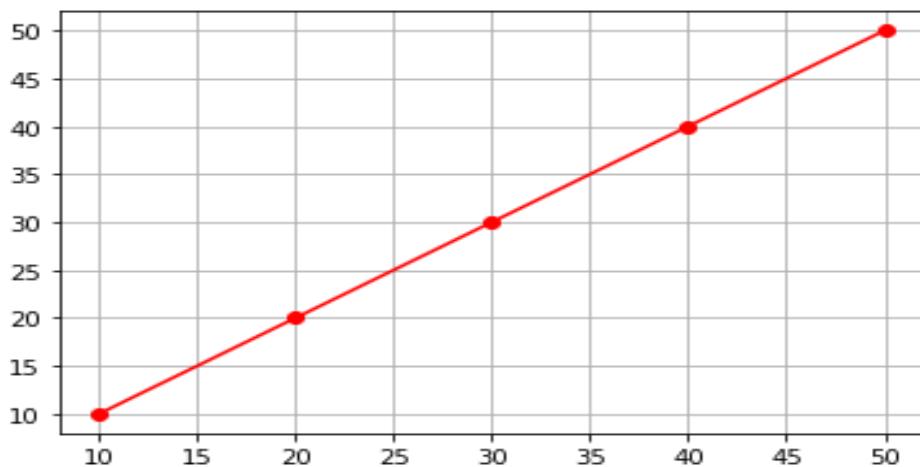
### **case-5:**

- ✓ plt.grid(b=False)

**grid lines are invisible**

In [45]:

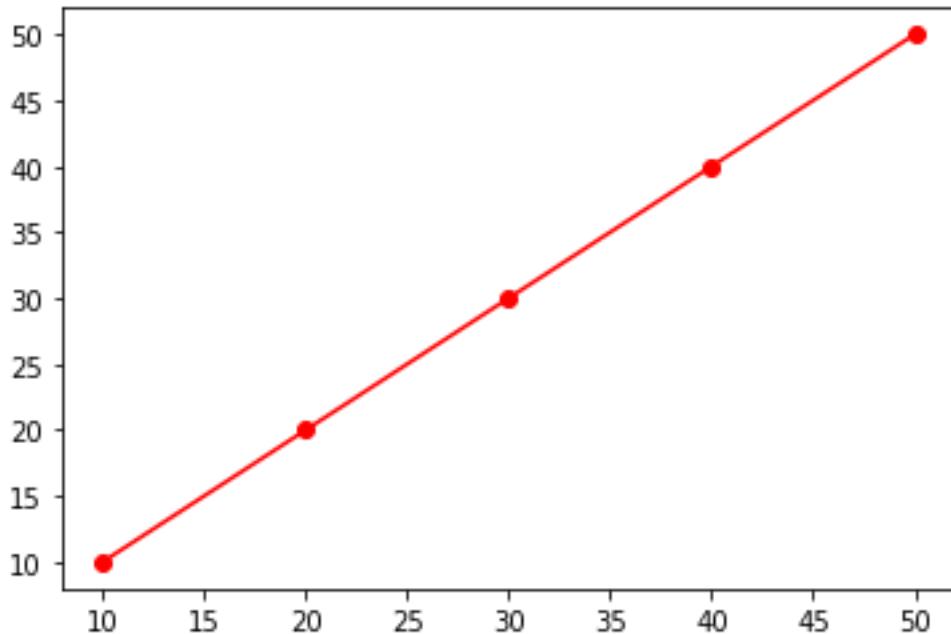
```
# case-1: grid lines are visible
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid()
plt.show()
```



---

In [46]:

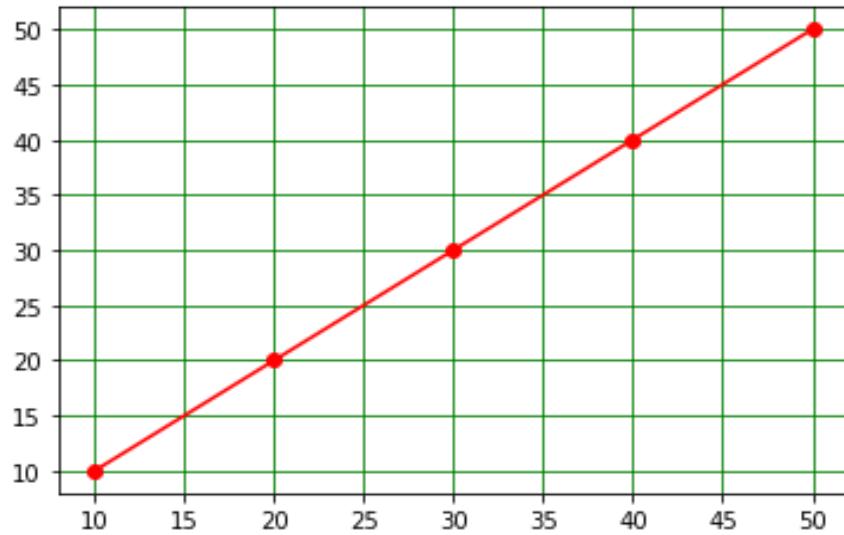
```
# case-2: grid lines are not visible
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid()
plt.grid()
plt.show()
```



In [47]:

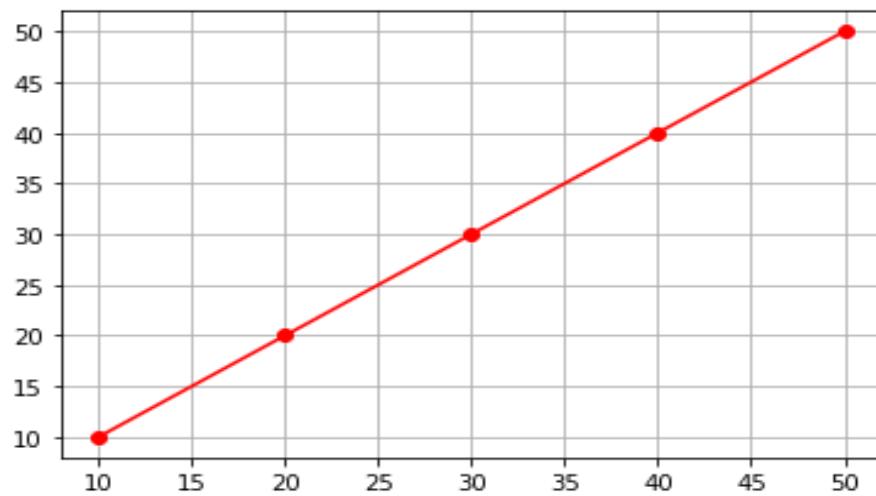
```
#case-3: if keyword arguments are given then grid lines are visible
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid(color='g')
plt.show()
```

---



In [48]:

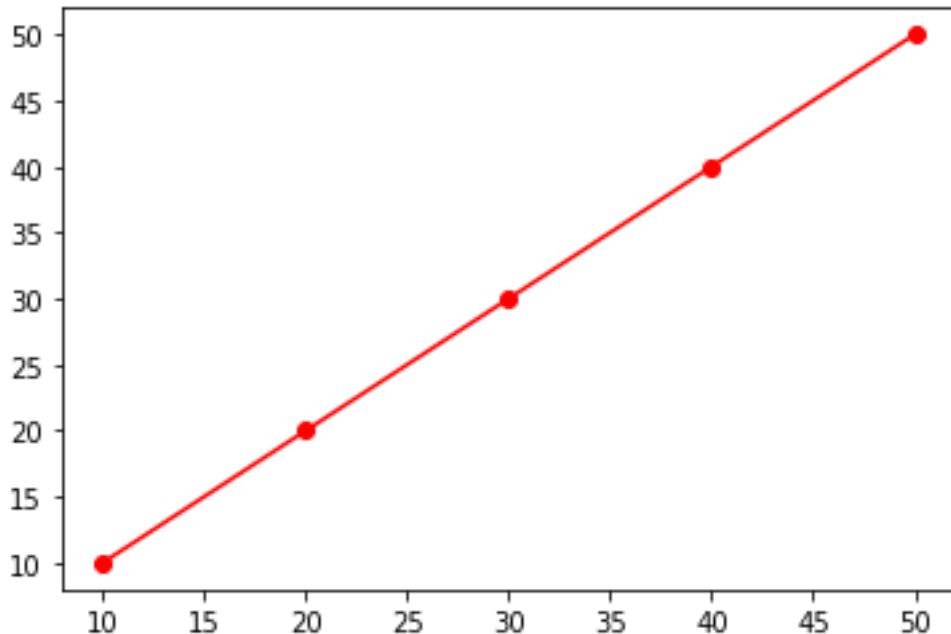
```
# case-4: plt.grid(b=True) ==> gridlines are visible
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid(b=True)
plt.show()
```



---

In [49]:

```
# case-5: plt.grid(b=False) ==> gridlines are not visible
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid(b=False)
plt.show()
```



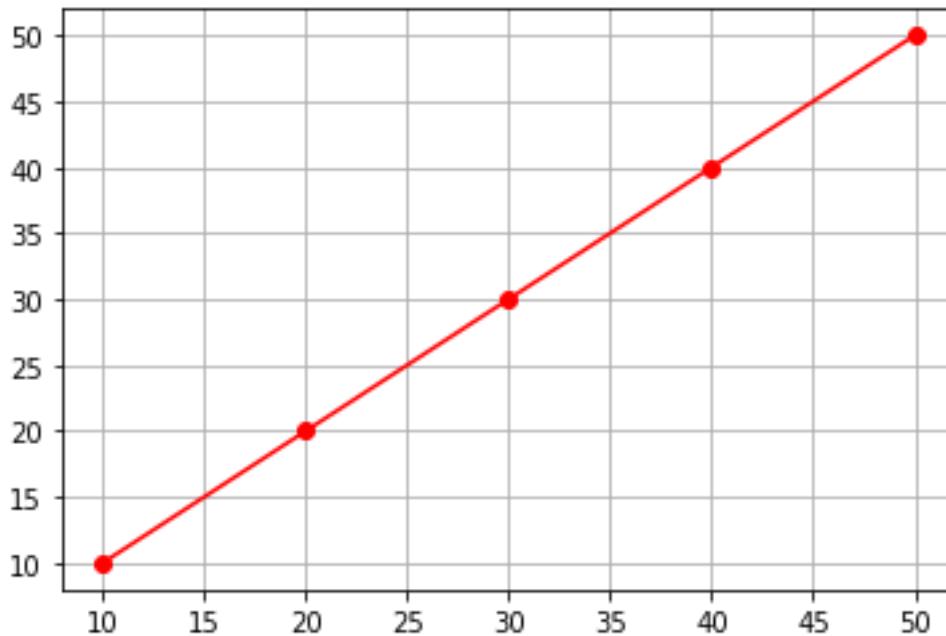
### which property

- ✓ default is **major**
  - ✓ There are **major grid lines** and **minor grid lines** are present
  - ✓ **which** property will decides which grid lines are going to be displayed
  - ✓ The allowed values are  
**which : {'major', 'minor', 'both'}** ==> The **default** value is **major**.
-

---

```
In [50]:
```

```
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.grid(which='both')
plt.show()
```



### Note

- ✓ To show the minor gridlines we have activate the minorticks\_on.
- ✓ Displaying minor ticks may reduce performance; you may turn them off using minorticks\_off() if drawing speed is a problem.
- ✓ **minorticks\_on()** → Display minor ticks on the axes.

```
In [51]:
```

```
help(plt.minorticks_on)
```

Help on function minorticks\_on in module matplotlib.pyplot:

```
minorticks_on()
```

---

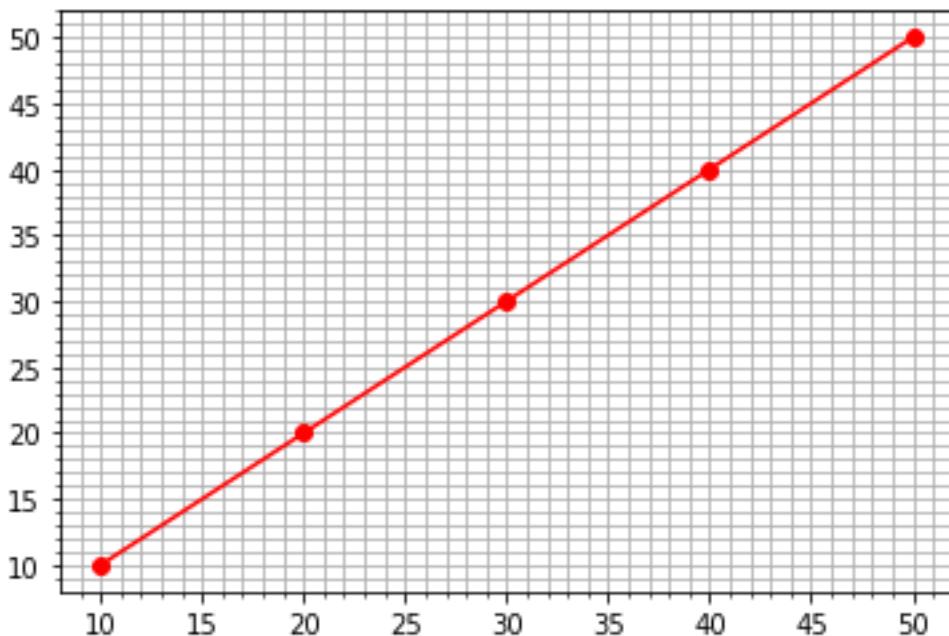
---

Display minor ticks on the axes.

Displaying minor ticks may reduce performance; you may turn them off using `minorticks\_off()` if drawing speed is a problem.

In [52]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r')
plt.minorticks_on()
plt.grid(which='both')
plt.show()
```

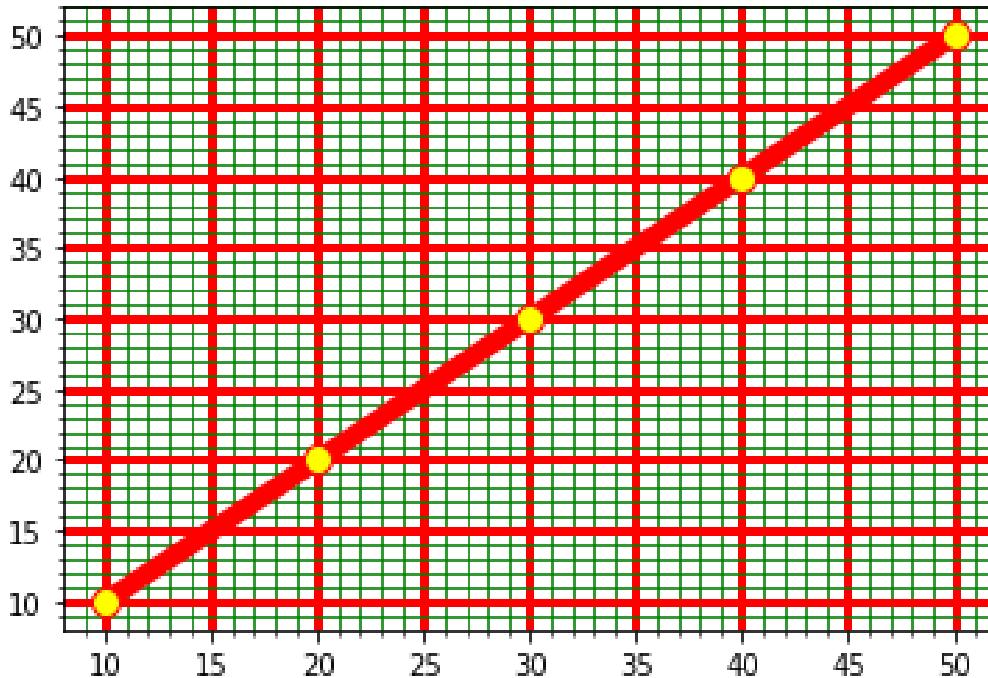


---

## Difference between major and minor grid lines:

In [53]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r',lw=7,markersize=10,mfc='yellow')
plt.grid(color='red',lw=3)
plt.minorticks_on()
plt.grid(which='minor',color='g')
plt.show()
```



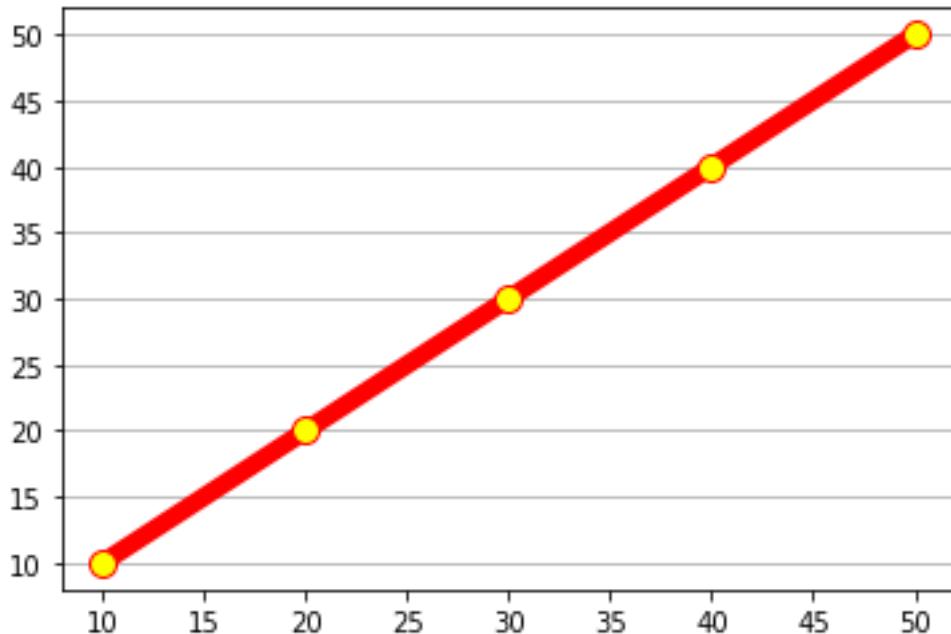
## axis property

- ✓ Along which axis, grid lines have to display  
**axis : {'both', 'x', 'y'}** ==> **default value: both**
-

---

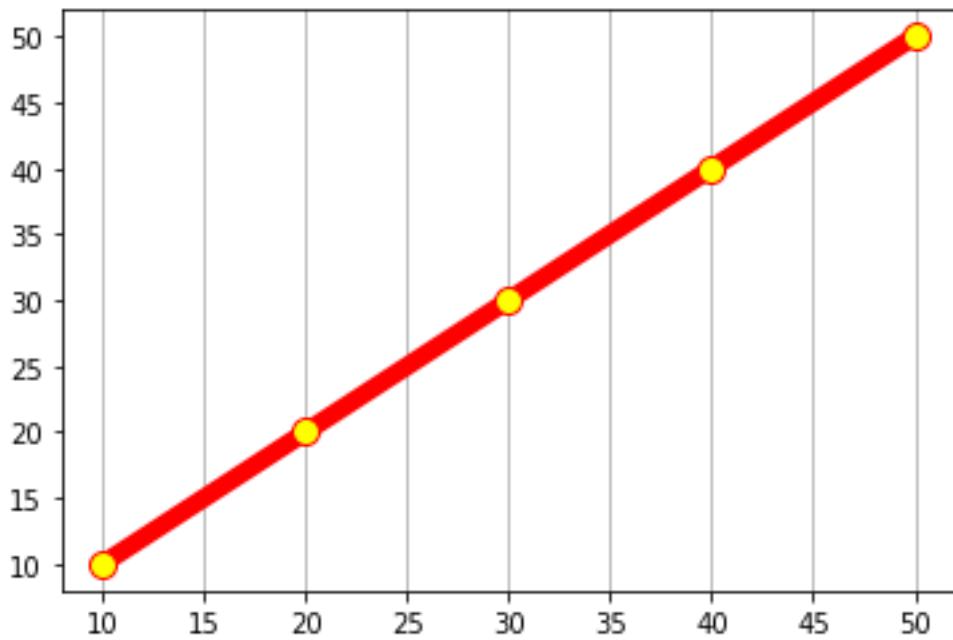
In [54]:

```
# y-axis
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r',lw=7,markersize=10,mfc='yellow')
plt.grid(axis='y')
plt.show()
```



In [55]:

```
# x-axis
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r',lw=7,markersize=10,mfc='yellow')
plt.grid(axis='x')
plt.show()
```



## Passing other keyword arguments

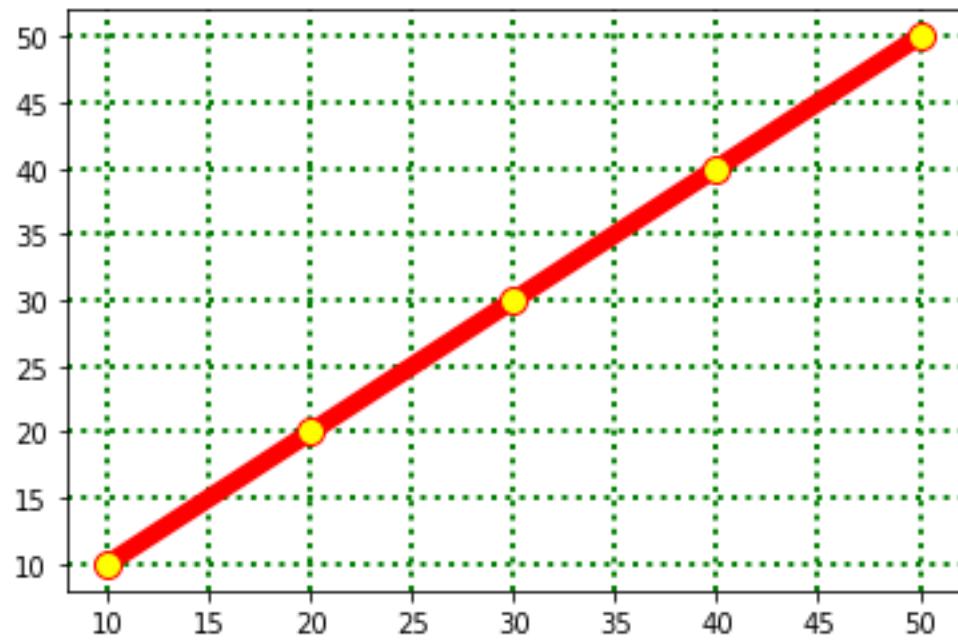
### Note:

- ✓ We can use several keyword arguments also.

```
plt.grid(color='g',lw=2,linestyle=':')
```

In [56]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.array([10,20,30,40,50])
plt.plot(a,a,'o-r',lw=7,markersize=10,mfc='yellow')
plt.grid(color='g',lw=2,linestyle=':')
plt.show()
```



---

## Chapter-5

### Adding Legend

#### Legend

- ✓ If multiple lines present then it is difficult to identify which line represents which dataset/function.
- ✓ To overcome this problem we can add legend.

In [57]:

```
import matplotlib.pyplot as plt
help(plt.legend)
```

Help on function legend in module matplotlib.pyplot:

```
legend(*args, **kwargs)
Place a legend on the axes.
```

#### Call signatures::

```
legend()
legend(labels)
legend(handles, labels)
```

#### legend()

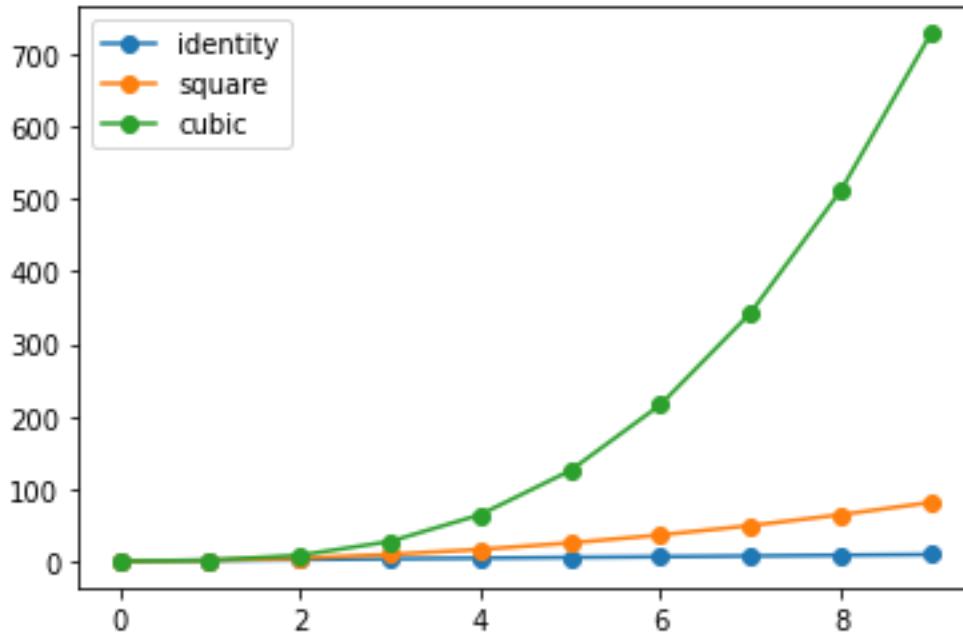
In [58]:

```
# legend() ==> without handles and labels
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
plt.plot(a,a,marker='o',label='identity')
plt.plot(a,a**2,marker='o',label='square')
plt.plot(a,a**3,marker='o',label='cubic')
plt.legend()
```

---

Out[58]:

<matplotlib.legend.Legend at 0x2032f636490>

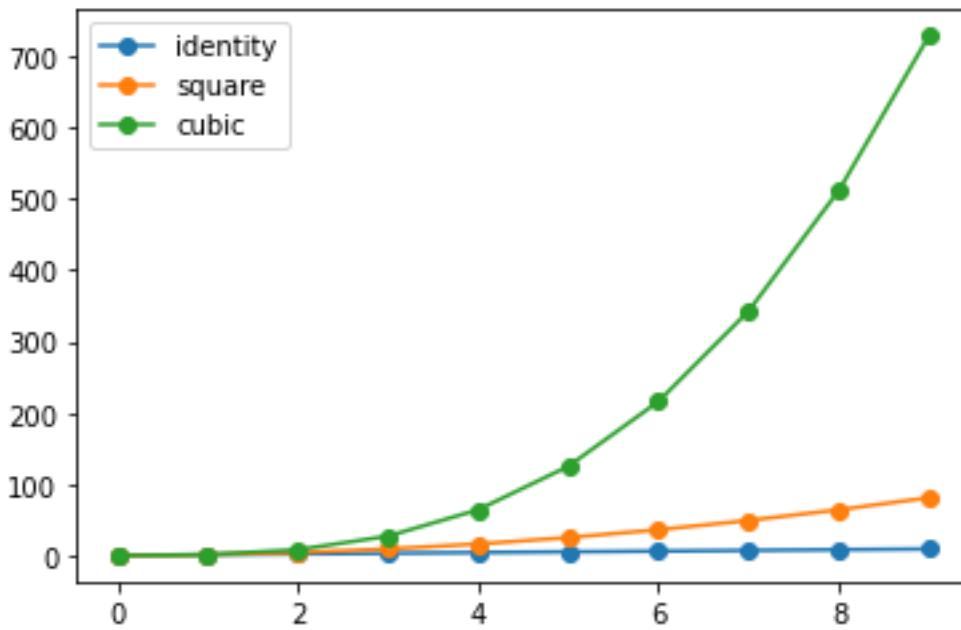


### legend(labels)

- ✓ The argument is list of strings.
  - ✓ Each string is considered as a label for the plots, in the order they created.
- plt.legend(['label-1','label-2','label-3'])**
- ✓ This approach is best suitable for adding legend for already existing plots.

In [59]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
plt.plot(a,a,marker='o')
plt.plot(a,a**2,marker='o')
plt.plot(a,a**3,marker='o')
plt.legend(['identity','square','cubic'])
plt.show()
```



**Note:**

- ✓ This approach is not recommended to use because we should aware the order in which plots were created.

**legend(handles, labels):**

- ✓ We can define explicitly lines and labels in the legend() function itself.
- ✓ It is recommended approach as we have complete control.

**plt.legend([line1, line2, line3], ['label-1', 'label-2', 'label-3'])**

**observation:**

```
1 = [10]
a = 1
print(a) #[10]
a, = 1 #unpack list elements and then assign values to provided variables
print(a) #10
```

---

```
In [60]:
```

```
l = [10]
a = 1
print(f"After initializing a = 1, a value is {a}") #[10]
```

```
a, = 1 #unpack list elements and then assign values ot provided variables
print(f"After unpacking the value of a, is {a}") #10
```

After initializing a = 1, a value is [10]

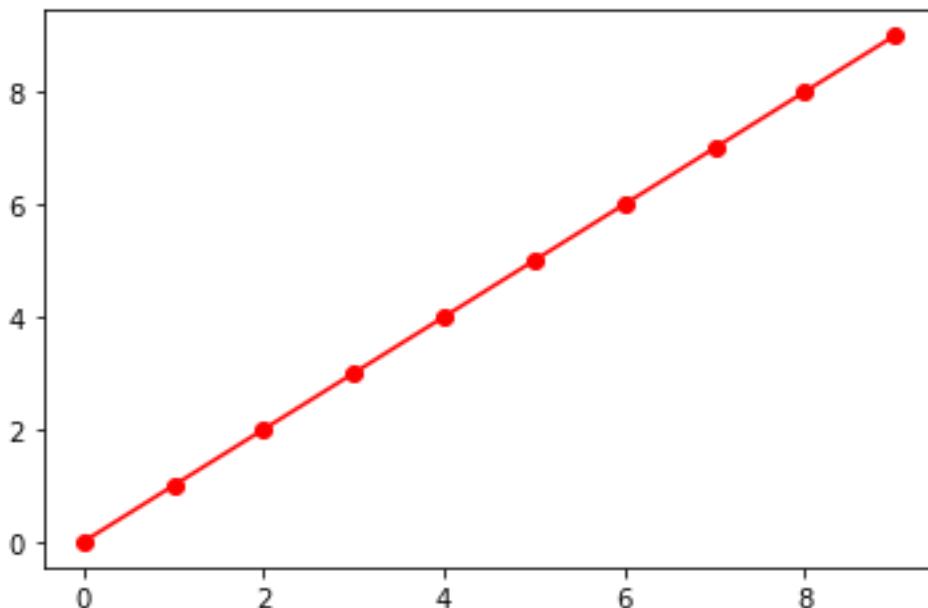
After unpacking the value of a, is 10

```
In [61]:
```

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r')
print(f"Type of lines : {type(lines)} ")
print(f"Lines object : ==> {lines}")
```

Type of lines : <class 'list'>

Lines object : ==> [<matplotlib.lines.Line2D object at 0x000002032E08DA60>]

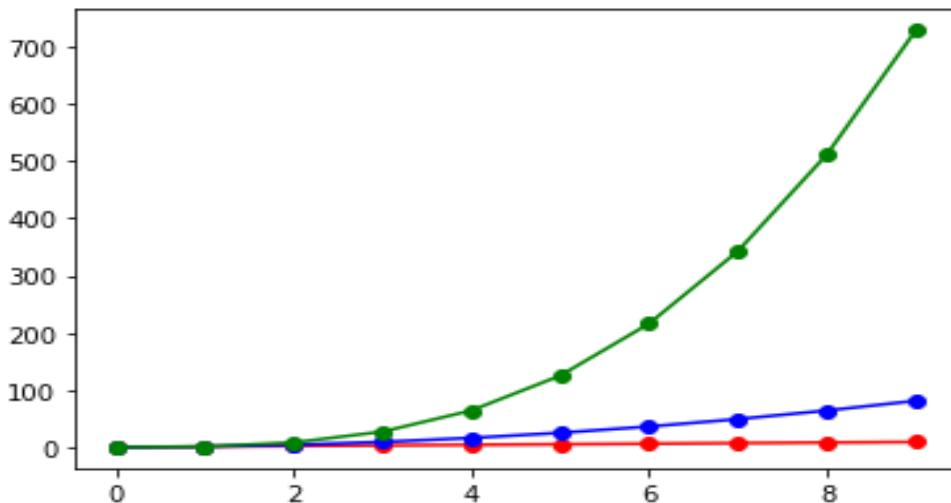


---

In [62]:

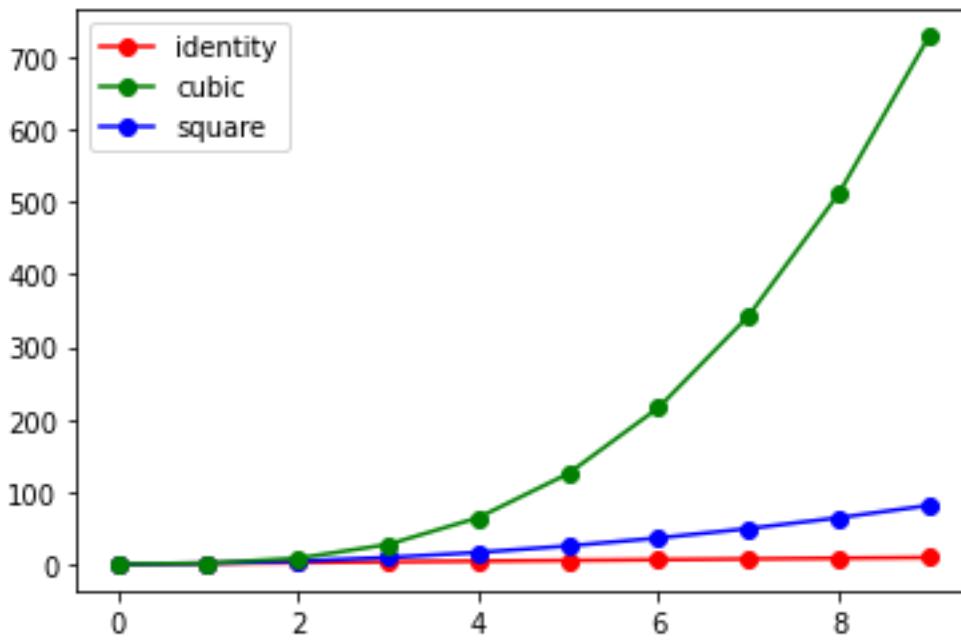
```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r',a,a**2,'o-b',a,a**3,'o-g')
print(f"Type of lines : {type(lines)} ")
print(f"Lines object : ==> {lines}")
```

Type of lines : <class 'list'>  
Lines object : ==> [<matplotlib.lines.Line2D object at 0x000002032DBC6AC0>, <matplotlib.lines.Line2D object at 0x000002032DBC6D30>, <matplotlib.lines.Line2D object at 0x000002032DBC64F0>]



In [63]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
line1, = plt.plot(a,a,'o-r')
line2, = plt.plot(a,a**2,'o-b')
line3, = plt.plot(a,a**3,'o-g')
plt.legend([line1,line3,line2],['identity','cubic','square'])
plt.show()
```



## How to adjust legend location

- ✓ Based on our requirement we can decide legend location in the plot.
- loc** argument  
loc → location

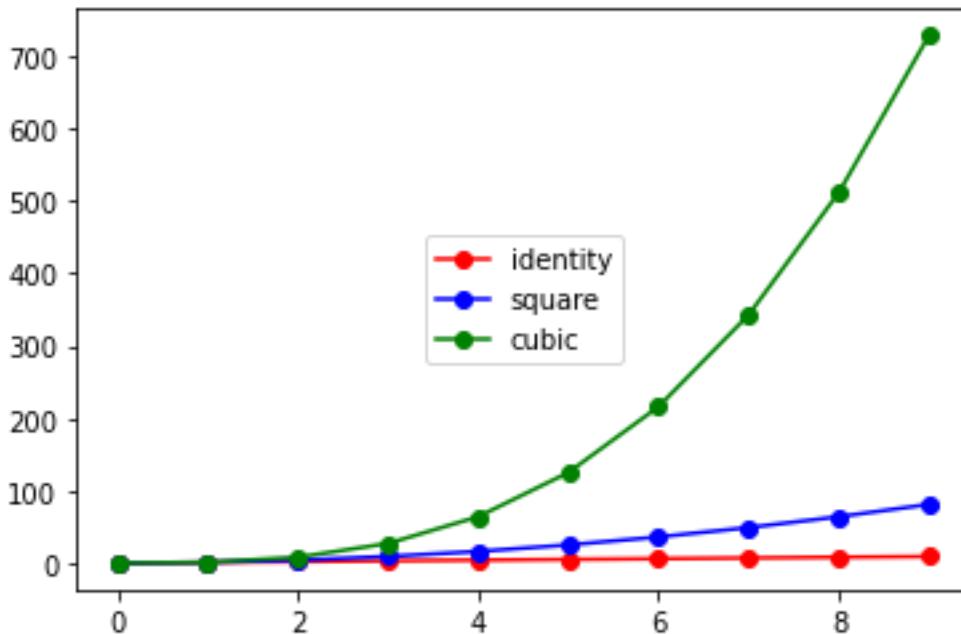
The possible values for the location are:

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

---

In [64]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r',a,a**2,'o-b',a,a**3,'o-g')
plt.legend(lines,['identity','square','cubic'],loc = 10)
plt.show()
```

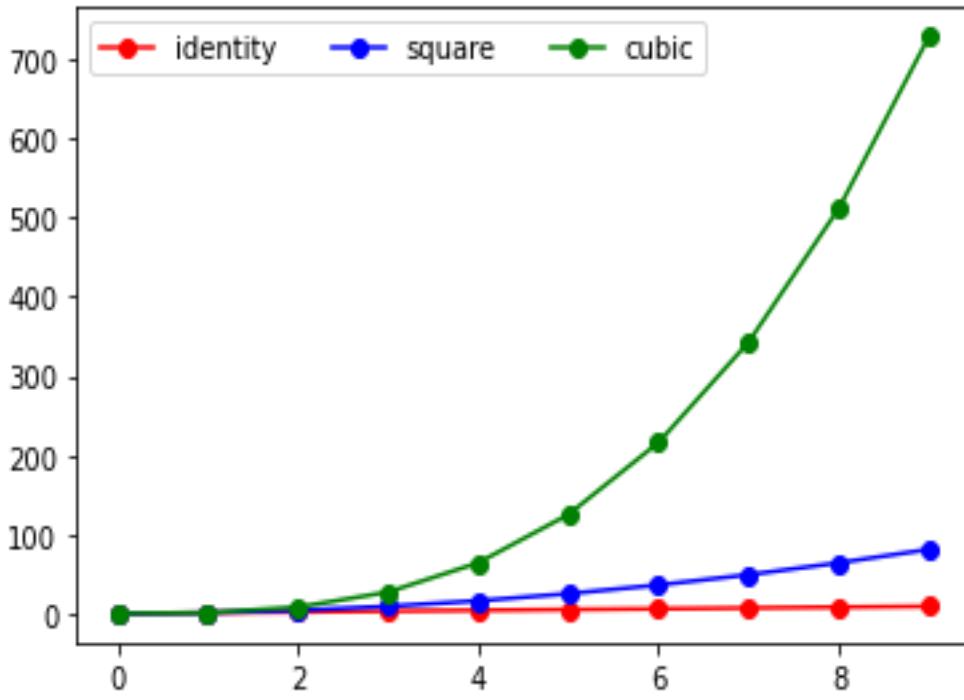


### How to specify number of columns in the legend

- ✓ By default the number of columns: 1
- ✓ But we can customize by using **ncol** argument.

In [65]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r',a,a**2,'o-b',a,a**3,'o-g')
plt.legend(lines,['identity','square','cubic'],ncol=3)
plt.show()
```



We can do more customization for the legend like

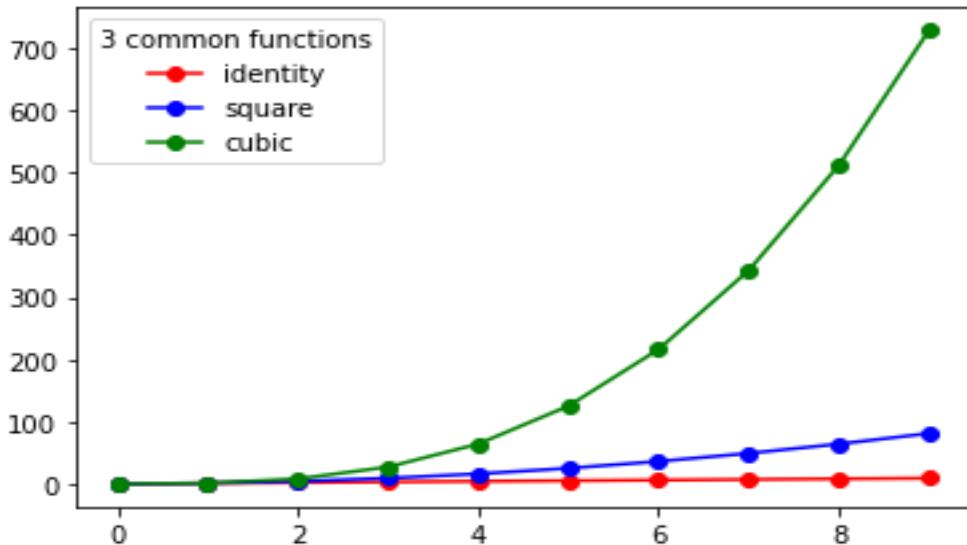
- ✓ We can add title to the legend.
- ✓ We can change look and feel
- ✓ We can change fontsize and color
- ✓ We can place legend outside of the plot etc

### Adding title to legend

- ✓ We can title for the legend explicitly. For this we have to use **title** keyword argument.

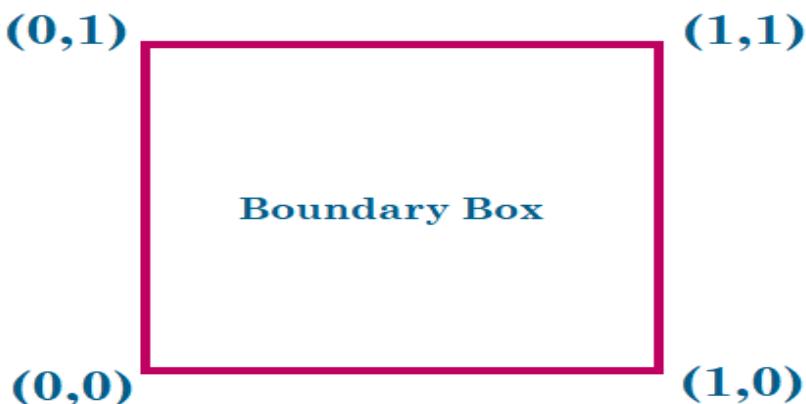
In [66]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r',a,a**2,'o-b',a,a**3,'o-g')
plt.legend(lines,['identity','square','cubic'],title='3 common functions')
plt.show()
```



### How to add legend outside of the plot

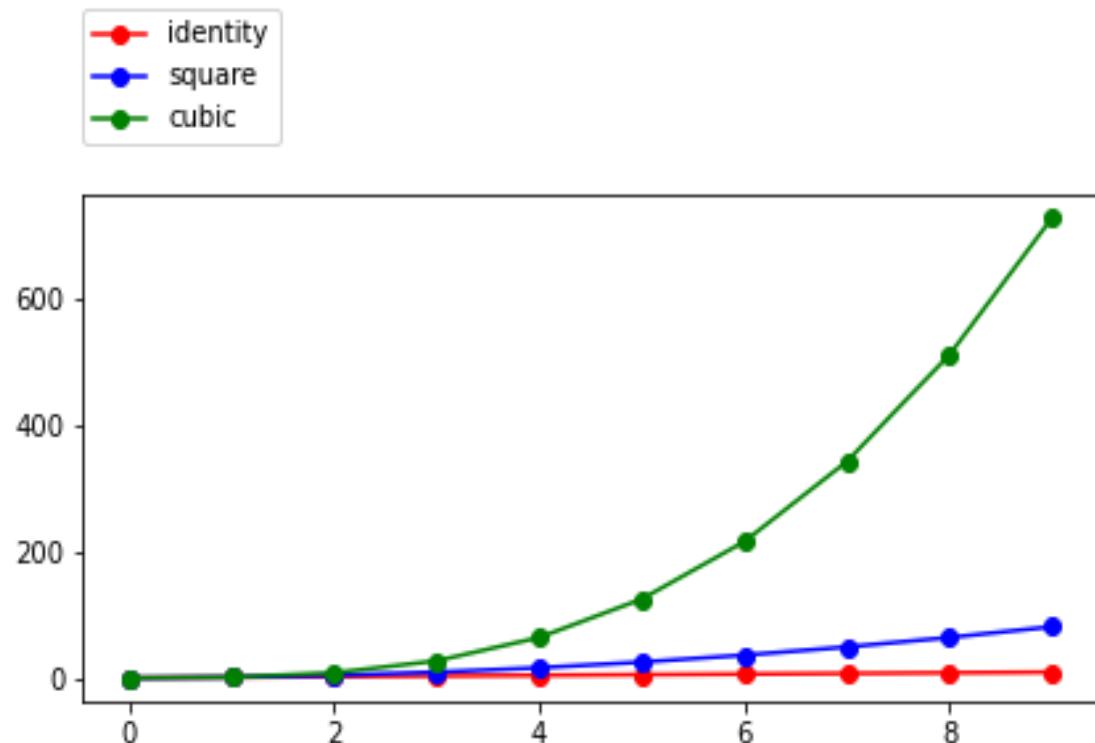
- ✓ We can add legend outside of the plot also.
- ✓ For this we have to use **loc** keyword argument.  
 $\text{loc} = (\text{x}, \text{y})$
- ✓ **loc** keyword can take three types of values  
 loc → location string  
 loc → location code  
 loc → tuple of two float values ( this is to add legend to outside of the plot)
- ✓  $\text{loc}=(\text{v1}, \text{v2})$  → lowest corner of Legend box



---

In [67]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
lines = plt.plot(a,a,'o-r',a,a**2,'o-b',a,a**3,'o-g')
plt.legend(lines,['identity','square','cubic'],loc=(0,1.1))
plt.tight_layout() # to fix the legend at fixed position
plt.show()
```



---

## Chapter-6

### Customization of Tick Location and Labels

#### Customization of tick location and labels

- ✓ Ticks are the markers to represent specific value on the axis.
- ✓ Ticks are very helpful to locate data points on the plot very easily.
- ✓ Based on our input, matplotlib decides tick values automatically.
- ✓ Based on our requirement, we can customize tick location and labels.
- ✓ For this we have to use **xticks()** and **yticks()**  
    `xticks(ticks=None, labels=None, **kwargs)`  
        **ticks** → arrays like ticks location(array like)  
        **labels** → label for ticks location(array like)  
        **kwargs** → to change the text properties of the label
- ✓ **ticks location** → where we want to place the ticks
- ✓ **ticks label** → name given to the tick location
- ✓ calling `xticks()` **without any argument** is nothing but the **getter** method
- ✓ calling `xticks()` **with arguments** is nothing but the **setter** method

#### Note

- ✓ `plt.xticks([])` ==> disable the xticks
- ✓ `plt.yticks([])` ==> disable the yticks

In [68]:

```
import matplotlib.pyplot as plt
help(plt.xticks)
```

Help on function `xticks` in module `matplotlib.pyplot`:

```
xticks(ticks=None, labels=None, **kwargs)
    Get or set the current tick locations and labels of the x-axis.
```

Pass no arguments to return the current values without modifying them.

---

In [69]:

```
help(plt.yticks)
```

Help on function yticks in module matplotlib.pyplot:

```
yticks(ticks=None, labels=None, **kwargs)
```

Get or set the current tick locations and labels of the y-axis.

Pass no arguments to return the current values without modifying them.

## Without customizing xticks() and yticks() → default values are based on the input

In [70]:

```
# To get the default values of the xticks
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
print("Default values of xticks generated by Matplotlib based on our input values")
print(f"plt.xticks() ==> {plt.xticks()}")
print('*'*90)
print("Default values of yticks generated by Matplotlib based on our input values")
print(f"plt.yticks() ==> {plt.yticks()}")
print('*'*90)
plt.show()
```

Default values of xticks generated by Matplotlib based on our input values  
plt.xticks() ==> (array([-2., 0., 2., 4., 6., 8., 10., 12.]), [Text(0, 0, ''), Text(0, 0, '')])

---

```
*****
```

Default values of yticks generated by Matplotlib based on our input values  
plt.yticks() ==> (array([-200., 0., 200., 400., 600., 800., 1000., 1200.]),  
[Text(0, 0, ""), Text(0, 0, ""), Text(0, 0, ""), Text(0, 0, ""), Text(0, 0, ""),  
Text(0, 0, ""), Text(0, 0, "")])

```
*****
```



- ✓ [Text(0, 0, ""), Text(0, 0, "")] ➔ ticks labels
  - ✓ It is the Text instance
  - ✓ first argument denotes x(0), second argument denotes y(0) and third argument denotes the text("")
- help(plt.Text)**

---

## customizing xticks()

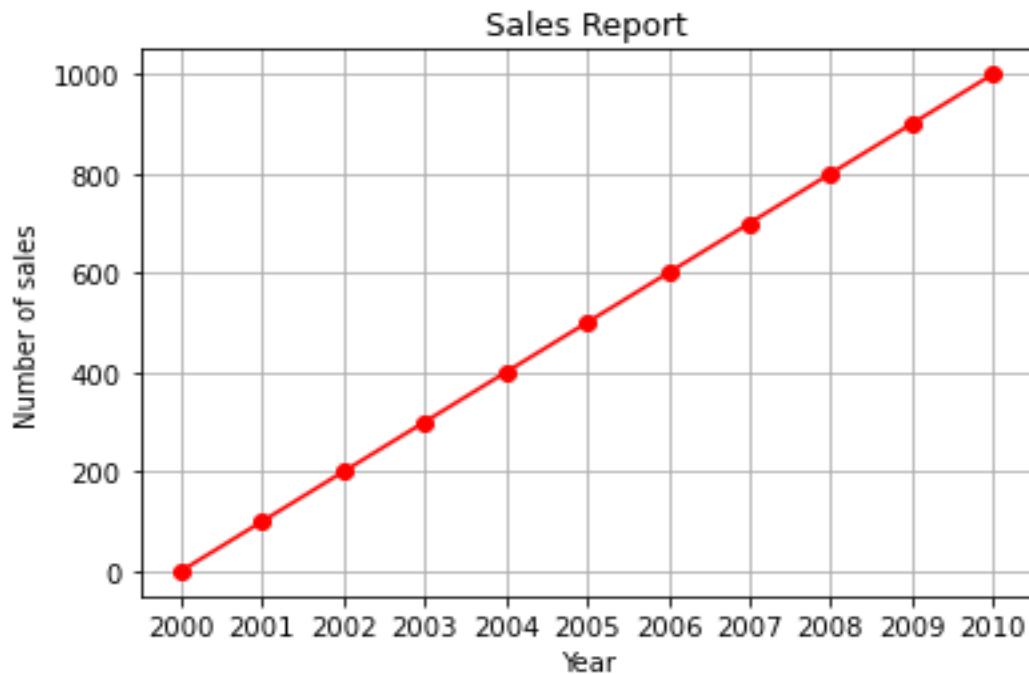
- ✓ plt.xticks(ticks=[0,1,2,3,4,5,6,7,8,9,10]) #to place our own xtick values
- ✓ For these xticks we can add labels also
- ✓ plt.xticks(ticks=[0,1,2,3,4,5,6,7,8,9,10],  
labels = ['2000','2001','2002','2003','2004','2005',  
'2006','2007','2008','2009','2010'])

In [71]:

```
# customizing the xticks values
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
plt.xticks(ticks= [0,1,2,3,4,5,6,7,8,9,10],
           labels= ['2000','2001','2002','2003','2004','2005',
           '2006','2007','2008','2009','2010']) #to place our own xtick values
print("Values of xticks after customization ")
print(f"plt.xticks() ==> {plt.xticks()}")
plt.show()
```

Values of xticks after customization

```
plt.xticks() ==> (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]), [Text(0, 0, '2000'),
Text(1, 0, '2001'), Text(2, 0, '2002'), Text(3, 0, '2003'), Text(4, 0, '2004'), Text(5,
0, '2005'), Text(6, 0, '2006'), Text(7, 0, '2007'), Text(8, 0, '2008'), Text(9, 0,
'2009'), Text(10, 0, '2010')])
```



### Note

We can customize **label properties** by using keyword arguments like **color, font, size** etc

In [72]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
plt.xticks(ticks= [0,1,2,3,4,5,6,7,8,9,10],
           labels= ['2000','2001','2002','2003','2004','2005',
                    '2006','2007','2008','2009','2010'],
           color='blue',size=15,rotation=30,family='fantasy')
plt.show()
```



### customizing both xticks() and yticks()

In [73]:

```
# customizing the xticks and yticks values
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
plt.xticks(ticks= [0,1,2,3,4,5,6,7,8,9,10],
           labels= ['2000','2001','2002','2003','2004','2005',
                    '2006','2007','2008','2009','2010'],
           color='blue',size=15,rotation=30,family='fantasy')
```

```

print("Values of xticks after customization ")
print(f"plt.xticks() ==> {plt.xticks()}")
print("*"*90)
plt.xticks([0,500,1000])# we have given locations only. labels are not given
print("Values of yticks after customization ")
print(f"plt.yticks() ==> {plt.yticks()}")
print("*"*90)
plt.show()
*****  

Values of xticks after customization  

plt.xticks() ==> (array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10]), [Text(0, 0, '2000'),  

Text(1, 0, '2001'), Text(2, 0, '2002'), Text(3, 0, '2003'), Text(4, 0, '2004'), Text(5,  

0, '2005'), Text(6, 0, '2006'), Text(7, 0, '2007'), Text(8, 0, '2008'), Text(9, 0,  

'2009'), Text(10, 0, '2010')])  

*****  

Values of yticks after customization  

plt.yticks() ==> (array([ 0,  500, 1000]), [Text(0, 0, ''), Text(0, 0, ''), Text(0, 0, '')])  

*****

```



---

**Note:**

Without providing **ticks** values we cannot provide labels, otherwise we will get **TypeError**

In [74]:

```
# Following code will gives TypeError because here without ticks values we
have provided
# labels values
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
plt.xticks(labels=['2000','2001','2002','2003','2004','2005',
                   '2006','2007','2008','2009','2010'],
           color='blue',size=15,rotation=30)
plt.yticks([0,500,1000])
plt.show()
```

**TypeError: xticks(): Parameter 'labels' can't be set without setting 'ticks'**



### disabling xticks() or yticks()

- ✓ If we pass empty list to xtick() or yticks() then corresponding ticks will be disabled on the plot  
**plt.xticks([])** ==> xticks will be disabled  
**plt.yticks([])** ==> yticks will be disabled

In [75]:

```
# to disable the yticks we will use plt.yticks([]) function
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(11)
b = a*100
plt.plot(a,b,'o-r')
plt.grid()
plt.title('Sales Report')
plt.xlabel('Year')
plt.ylabel('Number of sales')
plt.xticks([0,1,2,3,4,5,6,7,8,9,10],
           ['2000','2001','2002','2003','2004','2005','2006','2007','2008','2009','2010'],
           color='blue',size=15,rotation=30) #to place our own xtick values
plt.yticks([]) # disable the yticks
plt.show()
```



---

## Chapter-7

### How to set limit range of values on x-axis and y-axis by using xlim() and ylim() functions

#### How to set limit range of values on x-axis and y-axis

- ✓ By using **xlim()** and **ylim()** functions we can set limit range on axis
  - For **x-axis**: → left and right
  - For **y-axis**: → bottom and top

In [76]:

```
import matplotlib.pyplot as plt
help(plt.xlim)
```

Help on function xlim in module matplotlib.pyplot:

```
xlim(*args, **kwargs)
Get or set the x limits of the current axes.
```

Call signatures::

```
left, right = xlim() # return the current xlim
xlim((left, right)) # set the xlim to left, right
xlim(left, right) # set the xlim to left, right
```

In [77]:

```
import matplotlib.pyplot as plt
help(plt.ylim)
```

Help on function ylim in module matplotlib.pyplot:

```
ylim(*args, **kwargs)
Get or set the y-limits of the current axes.
```

Call signatures::

```
bottom, top = ylim() # return the current ylim
ylim((bottom, top)) # set the ylim to bottom, top
ylim(bottom, top) # set the ylim to bottom, top
```

---

## xlim()

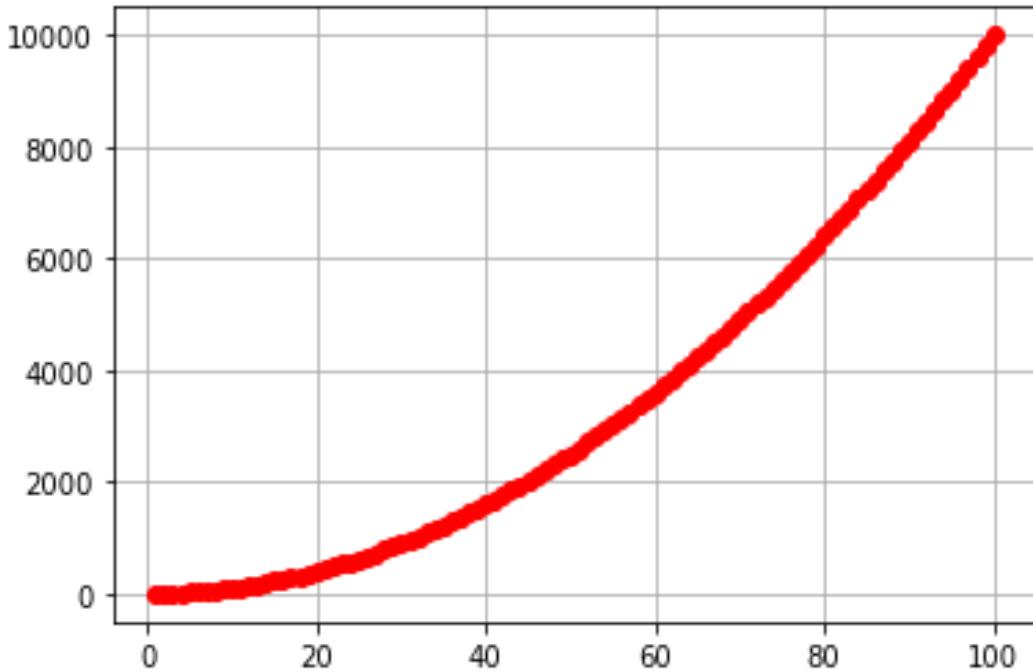
### Call signatures::

- ✓ left, right = xlim() # return the current xlim
- ✓ xlim((left, right)) # set the xlim to left, right
- ✓ xlim(left, right) # set the xlim to left, right
- ✓ xlim(right=3) # adjust the right leaving left unchanged
- ✓ xlim(left=1) # adjust the left leaving right unchanged
- ✓ If we are **not passing any argument** to xlim() function then it acts as **getter** function.
- ✓ If we are **passing any argument** then it acts as **setter** function.

In [78]:

```
# Getting left and right limits on the x-axis:  
import matplotlib.pyplot as plt  
import numpy as np  
a = np.arange(1,101)  
b = a**2  
plt.plot(a,b,'o-r')  
plt.grid()  
left,right = plt.xlim()  
bottom,top = plt.ylim()  
print('Left limit on the x-axis:',left)  
print('Right limit on the x-axis:',right)  
print('Bottom limit on the y-axis:',bottom)  
print('Top limit on the y-axis:',top)  
plt.show()
```

Left limit on the x-axis: -3.95  
Right limit on the x-axis: 104.95  
Bottom limit on the y-axis: -498.95000000000005  
Top limit on the y-axis: 10499.95



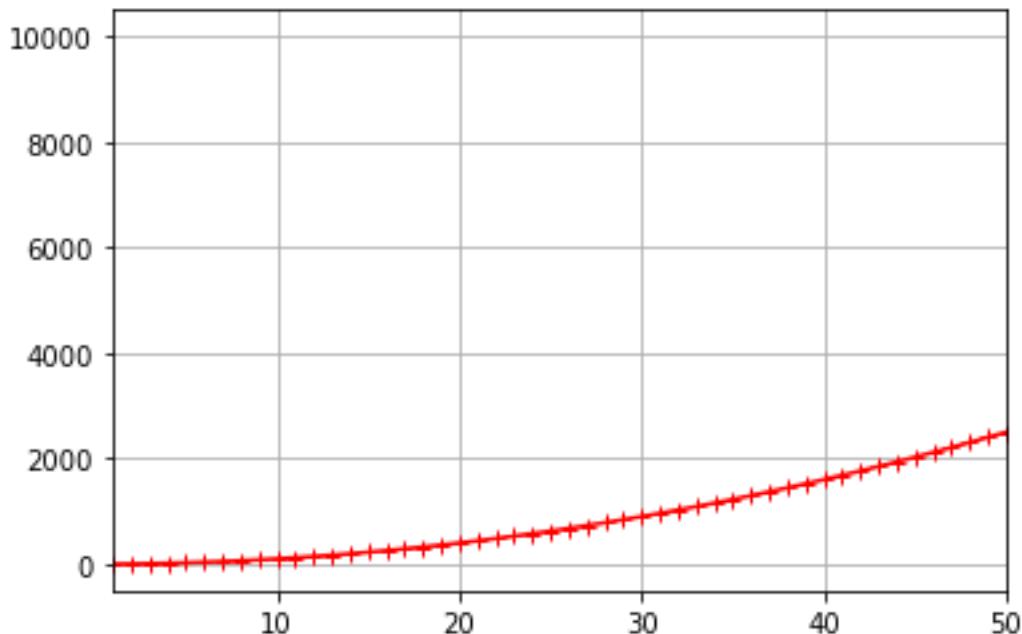
#### To set left and right limits on x-axis:

- ✓ plt.xlim(left,right)
- ✓ plt.xlim((left,right))
- ✓ plt.xlim(right=3) left will be generated by matplotlib
- ✓ plt.xlim(left=3) right will be generated by matplotlib
- ✓ plt.xlim(3)----->3 is for left and default for right

---

In [79]:

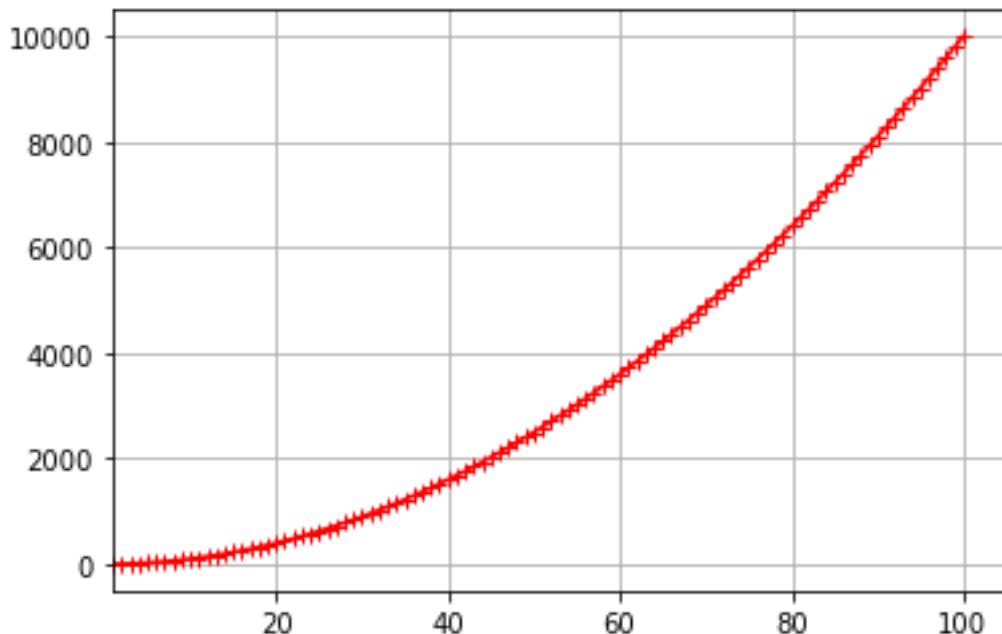
```
# left is 1 and right is 50
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,101)
b = a**2
plt.plot(a,b,'+-r')
plt.grid()
plt.xlim(1,50) # left is 1 and right is 50
plt.show()
```



---

In [80]:

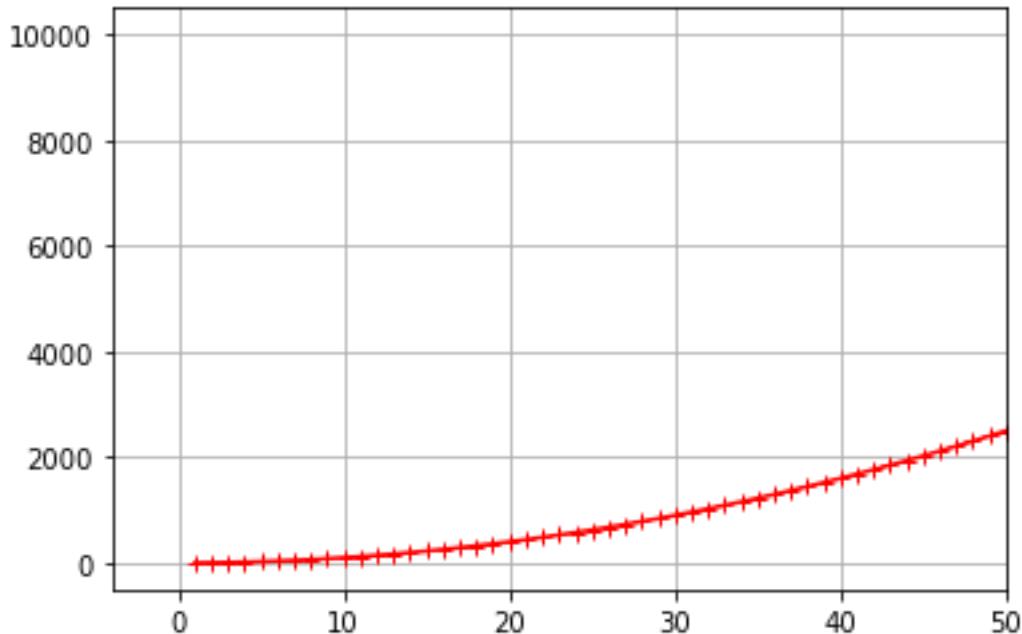
```
#left is 1 and for right default value
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,101)
b = a**2
plt.plot(a,b,'+-r')
plt.grid()
plt.xlim(left=1) #left is 1 and for right default value
plt.show()
```



---

In [81]:

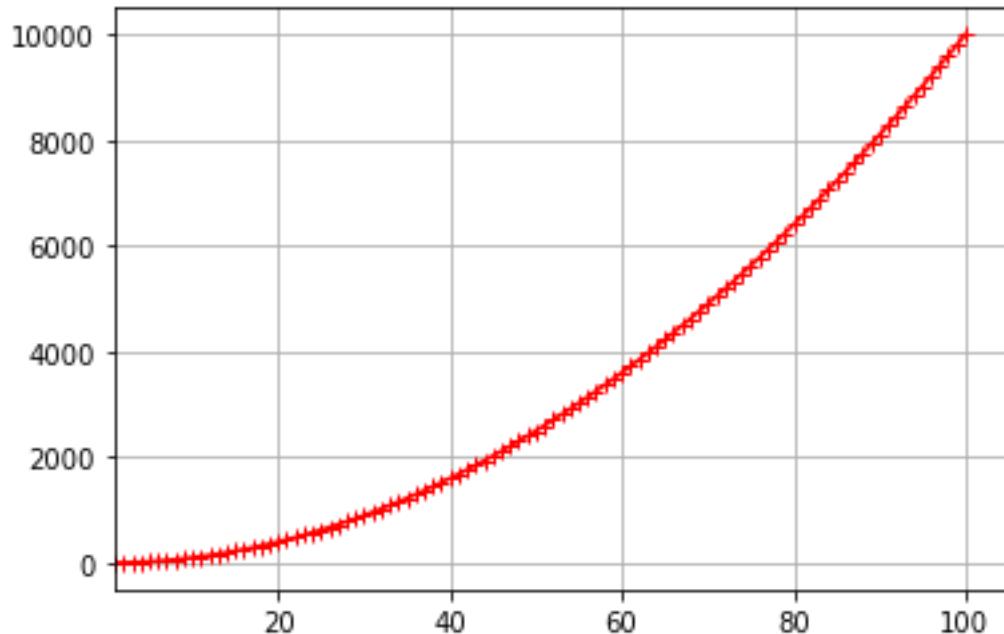
```
#left is default and for right 50
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,101)
b = a**2
plt.plot(a,b,'+-r')
plt.grid()
plt.xlim(right=50) #left is default and for right 50
plt.show()
```



---

In [82]:

```
#1 is for left and default for right
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,101)
b = a**2
plt.plot(a,b,'+-r')
plt.grid()
plt.xlim(1)#1 is for left and default for right
plt.show()
```



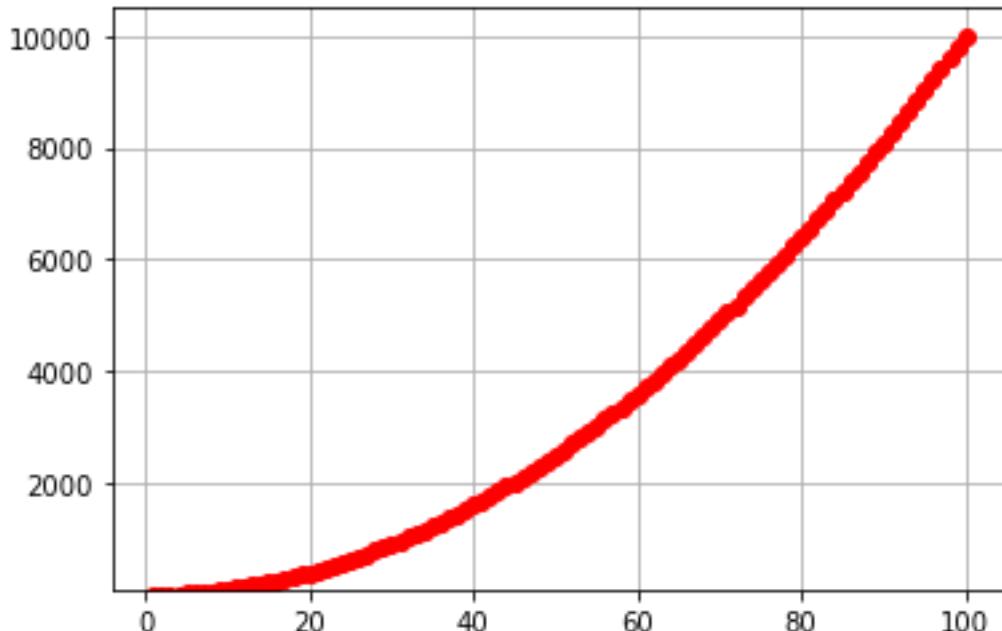
---

## ylim()

In [83]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,101)
b = a**2
plt.plot(a,b,'o-r')
plt.grid()
plt.ylim(bottom=100)
print(plt.ylim())
plt.show()
```

(100.0, 10499.95)



---

## Chpater-8

### How to set scale of x-axis and y-axis?

#### Scaling: How to set scale for x-axis and y-axis:

- ✓ The difference between any two consecutive points on any axis is called scaling.
- ✓ The most commonly used scales are:
  - **Linear scaling**
  - **Logarithmic Scaling**

#### Linear scaling

- ✓ The difference between any two consecutive points on the given axis is always fixed, such type of scaling is called linear scaling.
- ✓ Default scaling in matplotlib is linear scaling.
- ✓ If the data set values are spreaded over small range, then linear scaling is the best choice.

#### Logarithmic Scaling

- ✓ The difference between any two consecutive points on the given axis is not fixed and it is multiples of 10, such type of scaling is called logarithmic scaling.
- ✓ If the data set values are spreaded over big range, then logarithmic scaling is the best choice.
- ✓ Scaling purpose we can user **plt.xscale()** and **plt.yscale()**

In [84]:

```
import matplotlib.pyplot as plt
help(plt.xscale)
```

Help on function xscale in module matplotlib.pyplot:

**xscale(value, \*\*kwargs)**

Set the x-axis scale.

Parameters

value : {"linear", "log", "symlog", "logit", ...}

The axis scale type to apply.

---

In [85]:

```
import matplotlib.pyplot as plt  
help(plt.yscale)
```

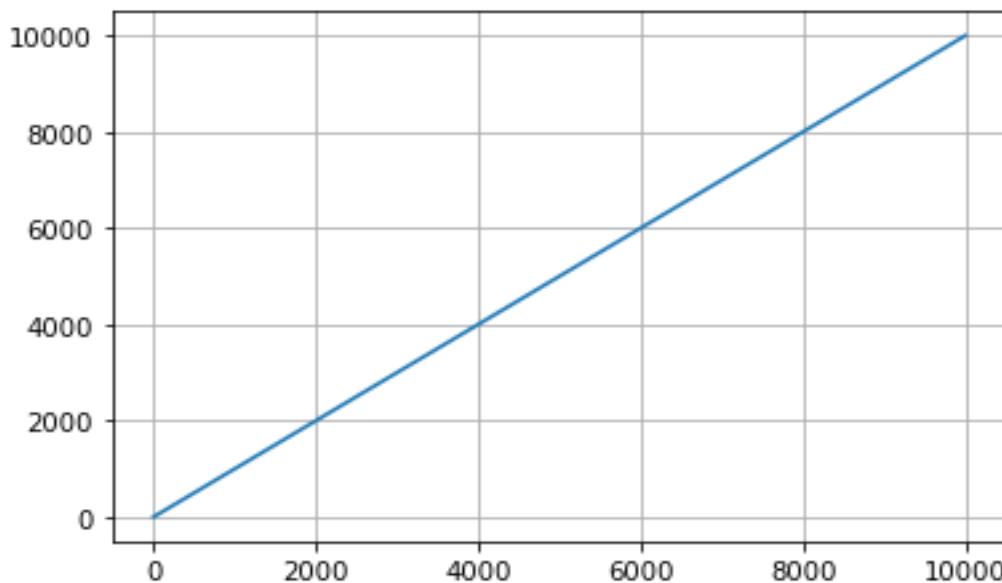
Help on function `yscale` in module `matplotlib.pyplot`:

```
yscale(value, **kwargs)  
    Set the y-axis scale.
```

## Linear scaling

In [86]:

```
# Linear scaling  
import matplotlib.pyplot as plt  
import numpy as np  
a = np.arange(10000)  
b = np.arange(10000)  
plt.plot(a,b)  
plt.grid()  
plt.xscale('linear')  
plt.show()
```

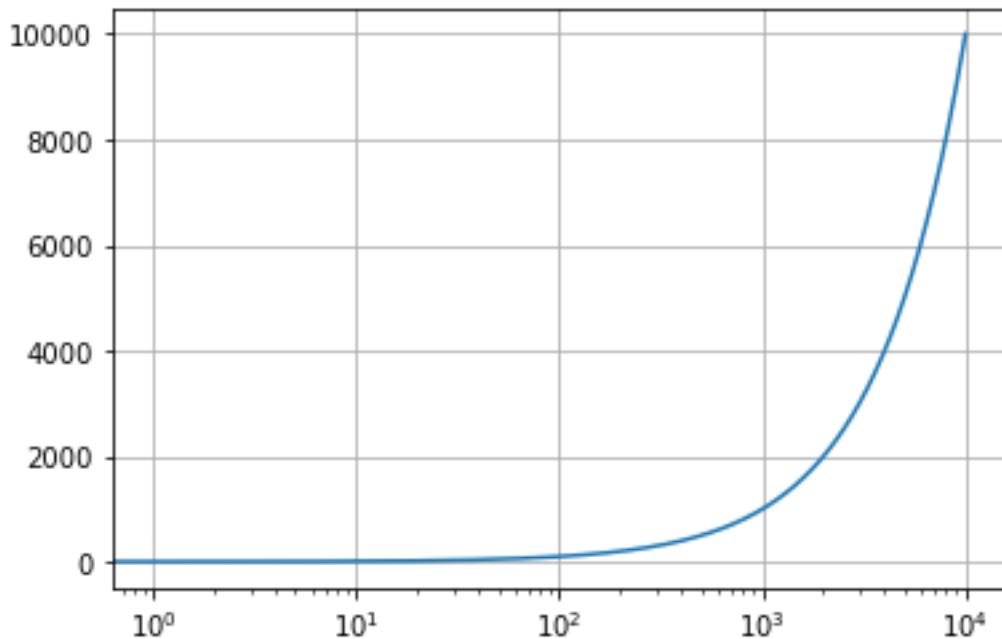


---

## Logarithmic scaling

In [87]:

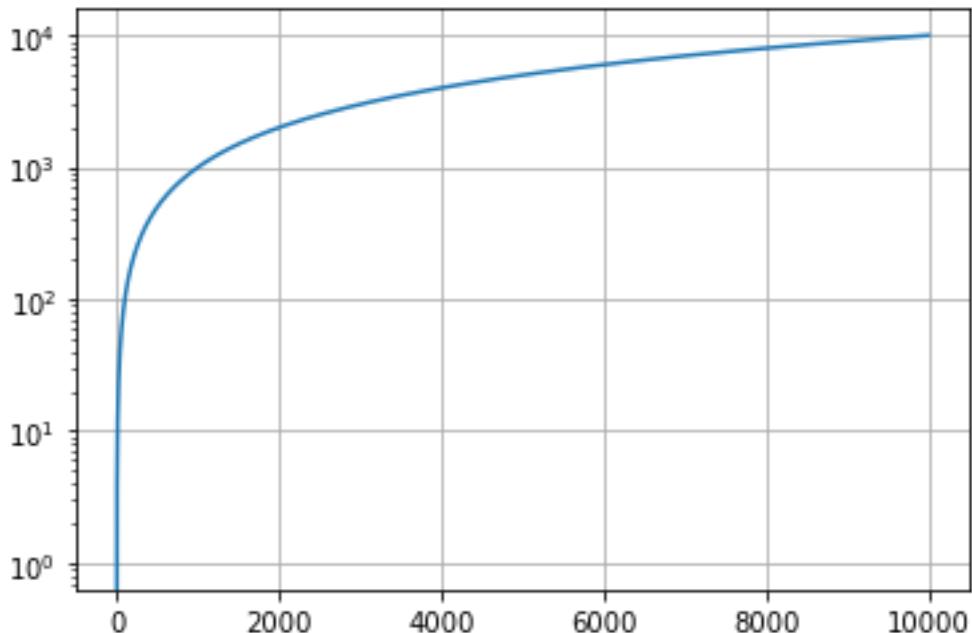
```
# Logarithmic scaling
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.plot(a,b)
plt.grid()
plt.xscale('log')
plt.show()
```



---

In [88]:

```
# Logarithmic scaling on y-axis and Linear scaling on x-axis
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.plot(a,b)
plt.grid()
plt.xscale('linear')
plt.yscale('log')
plt.show()
```



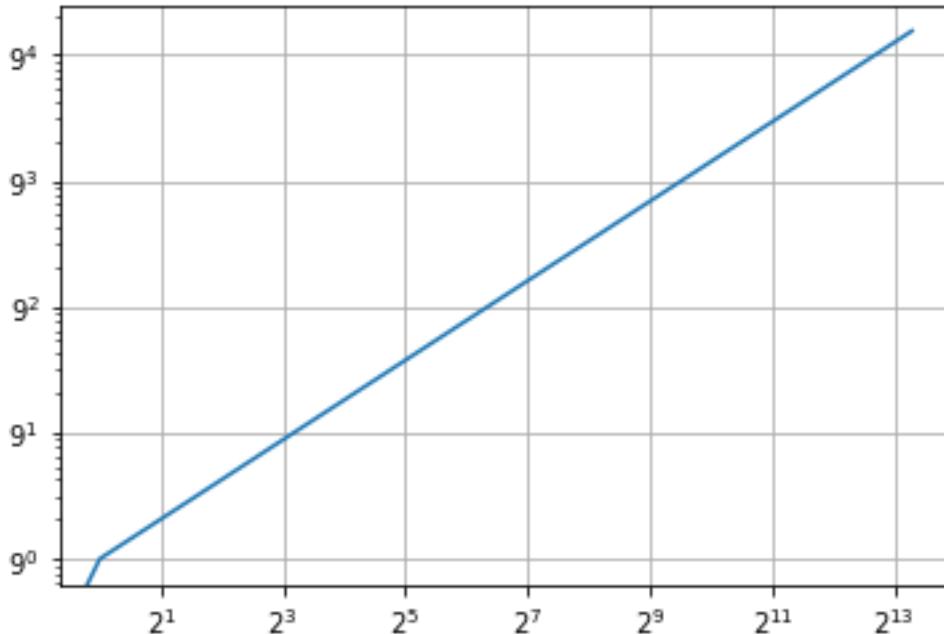
### How to customize base value in logarithmic scaling:

- ✓ By default '**10**' is the **base** parameter value.
  - ✓ We have to use keyword argument **base** to customize the logarithmic scaling
-

---

In [89]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.plot(a,b)
plt.grid()
plt.xscale('log',base=2) #logarithmic scaling
plt.yscale('log',base=9) #logarithmic scaling
plt.show()
```



---

## Chapter-9

### Plotting Styles

#### Plotting styles

- ✓ We can customize look and feel of hte plot by using style library.
- ✓ There are multiple predefined styles are available.

#### To know the plotting styles available in matplotlib

In [90]:

```
import matplotlib.pyplot as plt  
print(plt.style.available)
```

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

- ✓ **matplotlib** → visuliazation library
- ✓ **seaborn** → another visuliazation library

#### Note:

- ✓ **ggplot** → To emulate the most powerful ggplot style of R language.
- ✓ **seaborn** → To emulate seaborn style
- ✓ **fivethirtyeight** → The most commonly used style in real time. etc

#### How to use the style

- ✓ **plt.style.use('style\_name')**  
eg: **plt.style.use('ggplot')**

#### Note

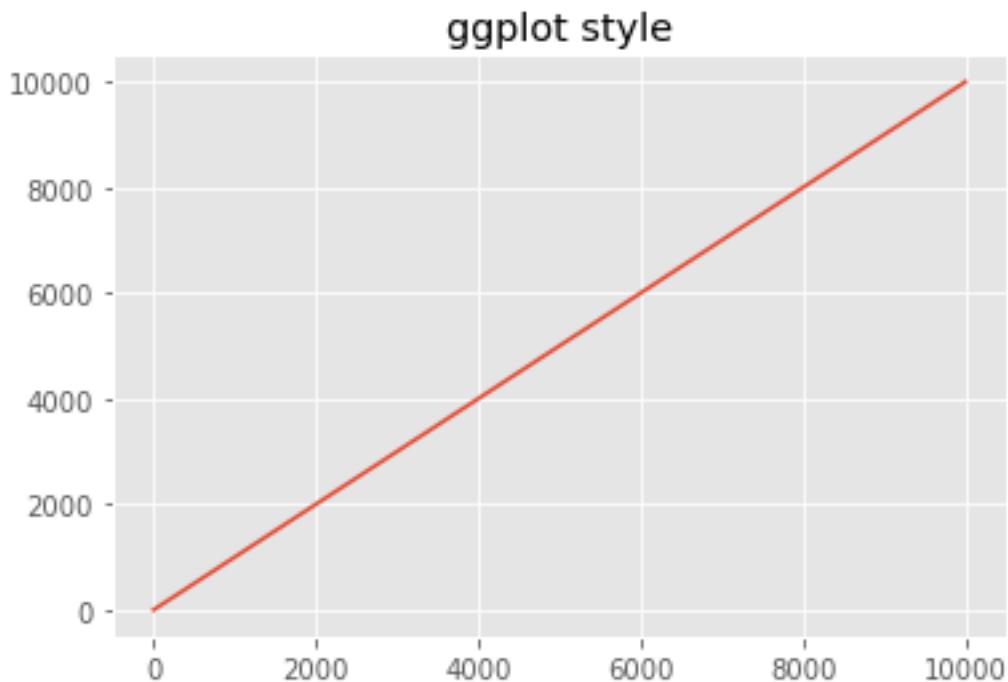
- ✓ Before calling the **plot()** function we have to set the style  
**plt.style.use('style\_name')**
- ✓ But this will change the style for the rest of the session!
- ✓ To revert back to default values run the following command

---

```
import matplotlib.pyplot as plt
plt.rcParams.update(plt.rcParamsDefault)
```

In [91]:

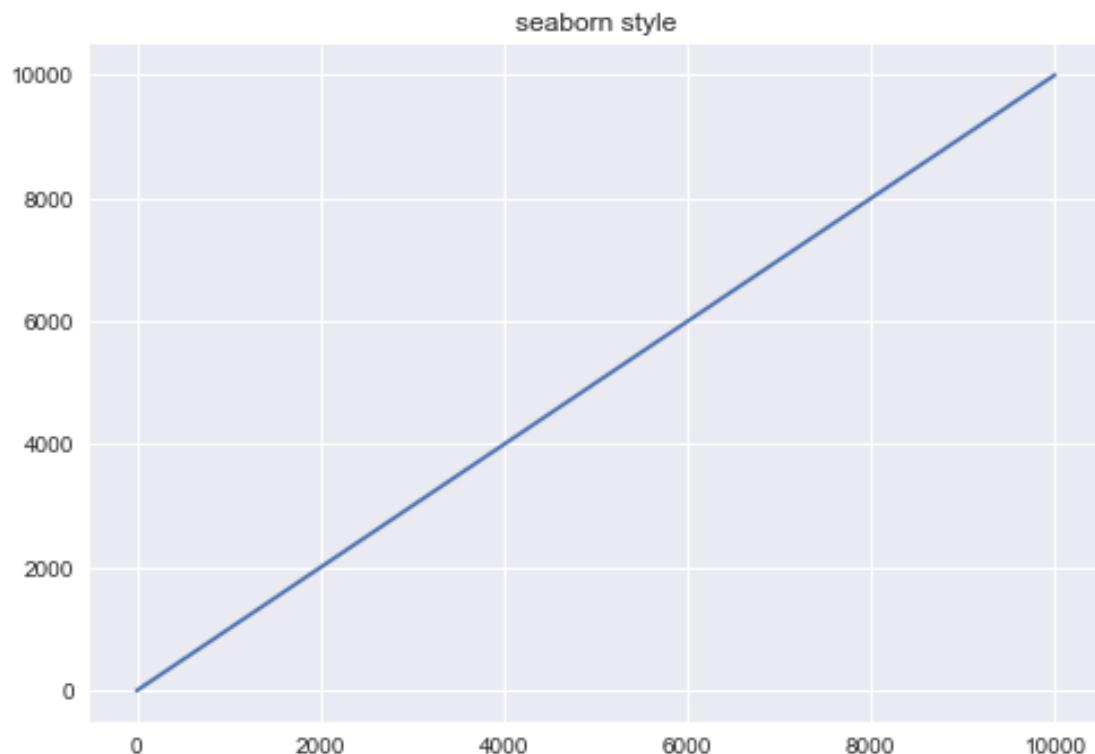
```
# ggplot style
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.style.use('ggplot')
plt.plot(a,b)
plt.title('ggplot style')
plt.show()
```



---

In [92]:

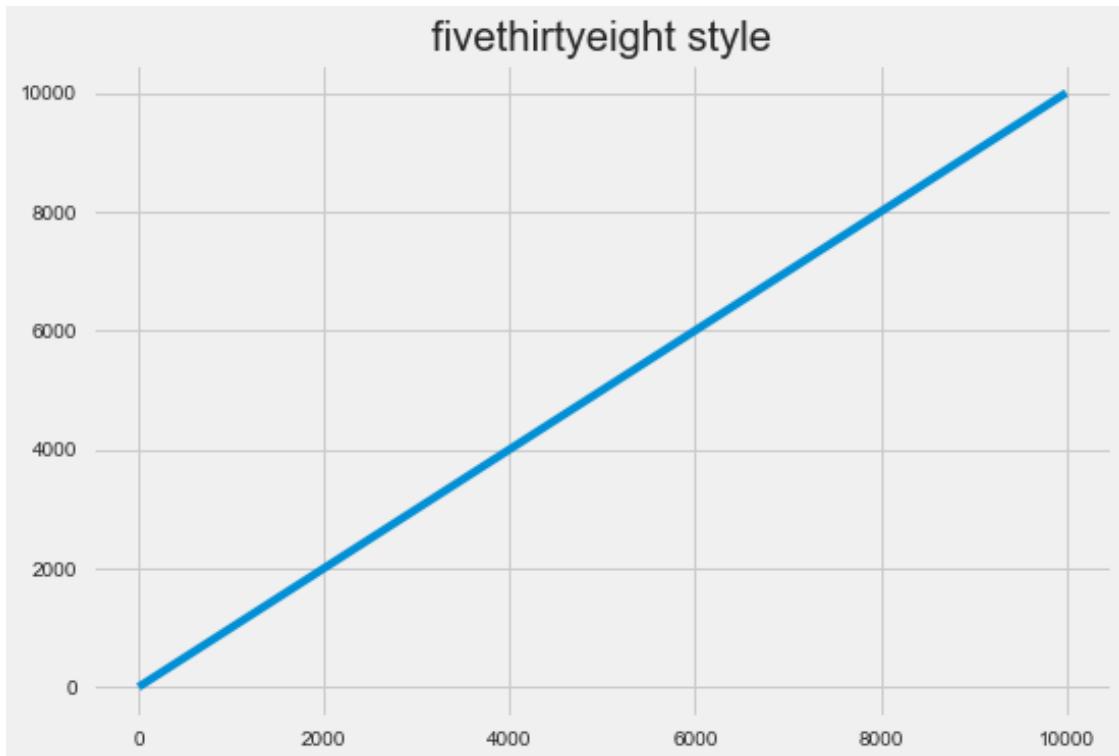
```
# seaborn style
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.style.use('seaborn')
plt.plot(a,b)
plt.title('seaborn style')
plt.show()
```



---

In [93]:

```
# fivethirtyeight style
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10000)
b = np.arange(10000)
plt.style.use('fivethirtyeight')
plt.plot(a,b)
plt.title('fivethirtyeight style')
plt.show()
```



## Default styling

To revert back to default values run the following command

In [94]:

```
import matplotlib.pyplot as plt
plt.rcParams.update(plt.rcParamsDefault)
```

---

## Chapter-10

# Functional/Procedural Oriented Vs Object Oriented Approached of plotting

### Approaches to create Plot

1. Procedural/Functional oriented approach
2. OOP approach

### Procedural or Functional Oriented Approach

In [95]:

```
def f1():
    print('f1 function')

def f2():
    print('f2 function')

def f3():
    print('f3 function')

def f4():
    print('f4 function')

f1()
f2()
f3()
f4()
```

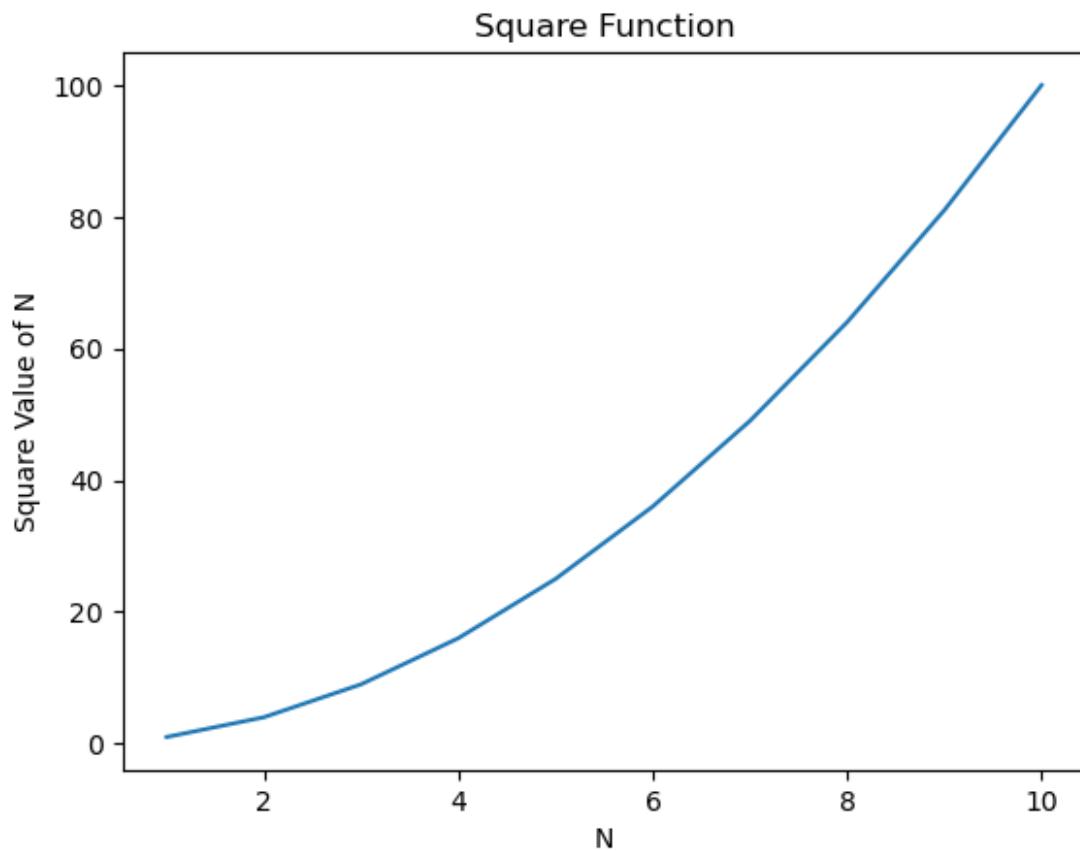
f1 function  
f2 function  
f3 function  
f4 function

- ✓ Up to now whatever we have discussed are belongs to Functional Oriented Approach
- ✓ We can create plots with the help of multiple functions from pyplot module.

---

In [96]:

```
#Creation of line plot to represent square functionality from 1 to 10.
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,11)
b = a**2
plt.plot(a,b)
plt.xlabel('N')
plt.ylabel('Square Value of N')
plt.title('Square Function')
plt.show()
```



---

The **functions** of **matplotlib.pyplot** module

- ✓ **plot()**
- ✓ **xlabel()**
- ✓ **ylabel()**
- ✓ **title()**
- ✓ **show()**

## OOP Approach

In [97]:

```
class Test:  
    def m1(self):  
        print('m1 method')  
    def m2(self):  
        print('m2 method')  
    def m3(self):  
        print('m3 method')  
    def m4(self):  
        print('m4 method')  
  
t = Test()  
t.m1()  
t.m2()  
t.m3()  
t.m4()
```

m1 method  
m2 method  
m3 method  
m4 method

- ✓ In this approach, we have to create objects and on those objects we have to call corresponding methods to create a plot.

Step 1: **Creation of Figure object**

Step 2: **Creation of Axes object**

Step 3: **Plot the graph**

Step 4: **Set the properties of the axes.**

---

### **Step1: Creation of Figure object:**

- ✓ fig = plt.figure()

In [98]:

```
import matplotlib.pyplot as plt
help(plt.figure)
```

Help on function figure in module matplotlib.pyplot:

```
figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None,
frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False,
**kwargs)
```

Create a new figure, or activate an existing figure.

In [99]:

```
import matplotlib.pyplot as plt

fig = plt.figure()
plt.show()
```

<Figure size 640x480 with 0 Axes>

### **Step2: Creation of Axes object:**

- ✓ Once figure object is ready, then we have to add axes to that object.
- ✓ For this we have to use **add\_axes()** method of Figure class.
- ✓ This method returns Axes object.

In [100]:

```
import matplotlib
help(matplotlib.figure.Figure.add_axes)
```

Help on function add\_axes in module matplotlib.figure:

```
add_axes(self, *args, **kwargs)
Add an axes to the figure.
```

---

Call signatures::

```
add_axes(rect, projection=None, polar=False, **kwargs)
add_axes(ax)
```

Parameters

-----  
rect : sequence of float

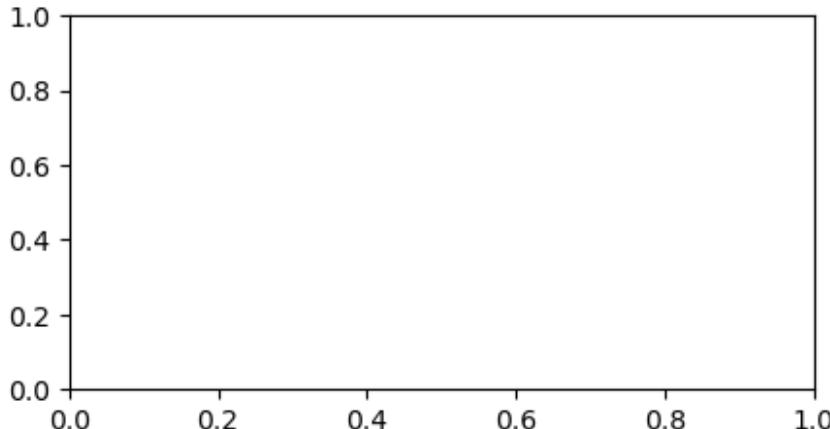
The dimensions [left, bottom, width, height] of the new axes. All quantities are in fractions of figure width and height.

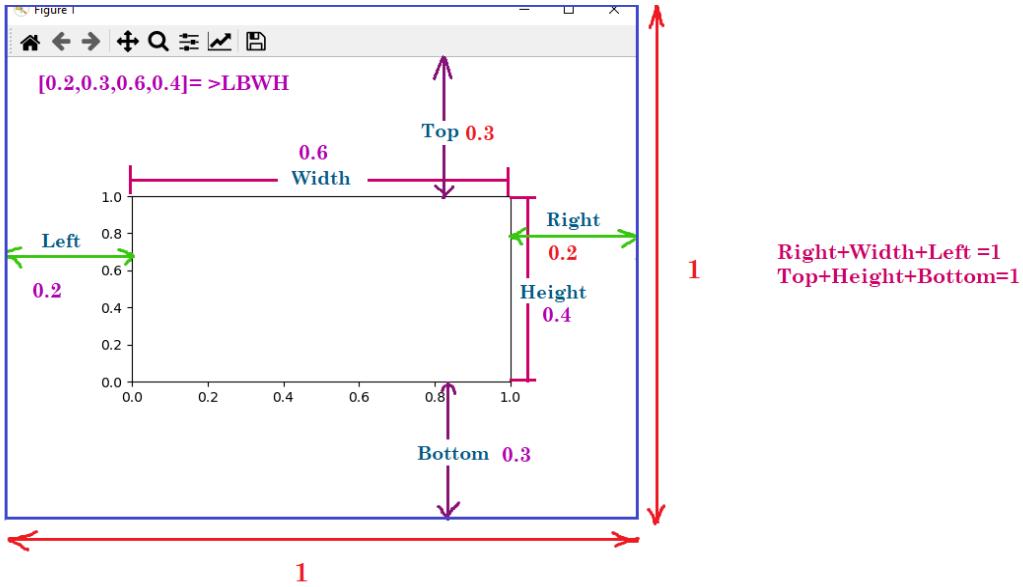
**LBWH**

In [101]:

```
import matplotlib.pyplot as plt

fig = plt.figure()
axes = fig.add_axes([0.2,0.3,0.6,0.4])
plt.show()
```





### Step3: Plot the graph:

- Once Axes object is ready, then we can use the following methods.  
axes.plot(a,b)

### Step 4: Set the properties of the axes.

```
axes.set_xlabel('xlabel')
axes.set_ylabel('ylabel')
axes.set_title('title')
plt.show()
```

In [102]:

```
#Creation of line plot to represent square functionality from 1 to 10.
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,11)
b = a**2

# Step 1: Creation of figure object
fig = plt.figure()

# Step 2: Creation of Axes object
axes = fig.add_axes([0.2,0.3,0.6,0.4]) #[left,bottom,width,height] lbwh
```

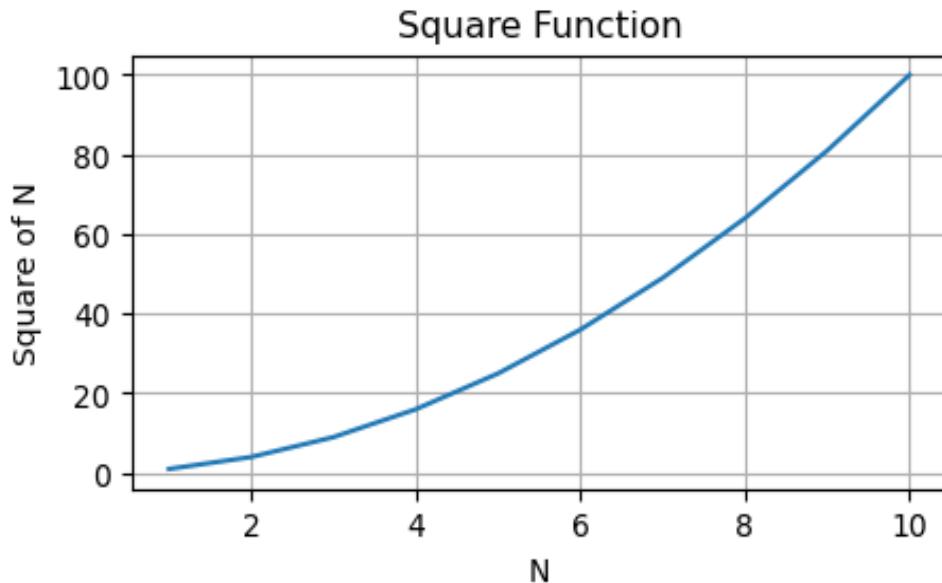
---

```

# Step 3: plotting the graph
axes.plot(a,b)

# Step 4: Setting axes properties
axes.set_xlabel('N')
axes.set_ylabel('Square of N')
axes.set_title('Square Function')
axes.grid()
plt.show()

```



**Note:**

- ✓ We can use single **set()** method to set all axes properties like title,xlabel,ylabel,xlim,ylim etc

**eg:** `axes.set(xlabel='N',  
 ylabel='Square of N',  
 title='Square Function',  
 xlim=(1,5),  
 ylim=(1,25))`

---

In [103]:

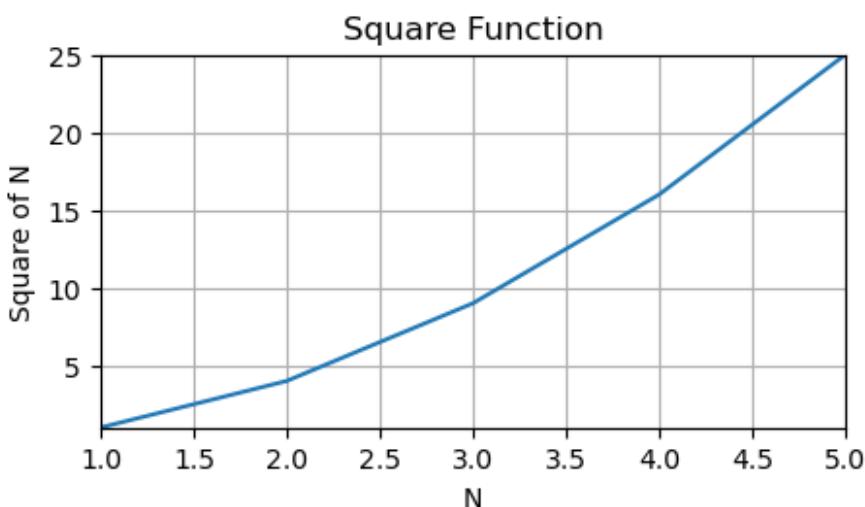
```
#Creation of line plot to represent square functionality from 1 to 10.
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(1,11)
b = a**2

# Step 1: Creation of figure object
fig = plt.figure()

# Step 2: Creation of Axes object
axes = fig.add_axes([0.2,0.3,0.6,0.4]) #left,bottom,width,height l b w h

# Step 3: plotting the graph
axes.plot(a,b)

# Step 4: Setting axes properties ==> with single set() method
axes.set(xlabel='N',
          ylabel='Square of N',
          title='Square Function',
          xlim=(1,5),
          ylim=(1,25))
axes.grid()
plt.show()
```



---

## Chapter-11

# Bar Chart/ Bar Graph/ Bar Plot

### Bar Chart/Bar Graph/Bar Plot

- ✓ In a line plot, the data points will be marked and these markers will be connected by line.
- ✓ But in bar chart, data will be represented in the form of bars.

#### 4 types of bar charts

1. Simple bar chart/vertical bar chart
2. Horizontal bar chart
3. Stacked Bar chart
4. Clustered Bar Chart/Grouped Bar Chart

#### Simple bar chart/vertical bar chart:

- ✓ The data will be represented in the form of vertical bars.
- ✓ Each vertical bar represents an individual category.
- ✓ The height/length of the bar is based on value it represents.
- ✓ Most of the times the width of the bar is fixed, but we can customize.
- ✓ The **default width: 0.8**
- ✓ By using **bar()** function we can create bar chart

In [104]:

```
import matplotlib.pyplot as plt
help(plt.bar)
```

Help on function bar in module matplotlib.pyplot:

```
bar(x, height, width=0.8, bottom=None, *, align='center', data=None, **kwargs)
    Make a bar plot.
```

**x** → values of x-axis, category names  
**height** → values for y-axis, height of the bars.  
**width** → width of each bar,default is 0.8  
**bottom** → From where bar has to start.  
**align** → alignment of the bars on the x-axis.

---

`align : {'center', 'edge'}, default: 'center' ➔ Alignment of the bars to the x coordinates:`

`'center': Center the base on the x positions.`

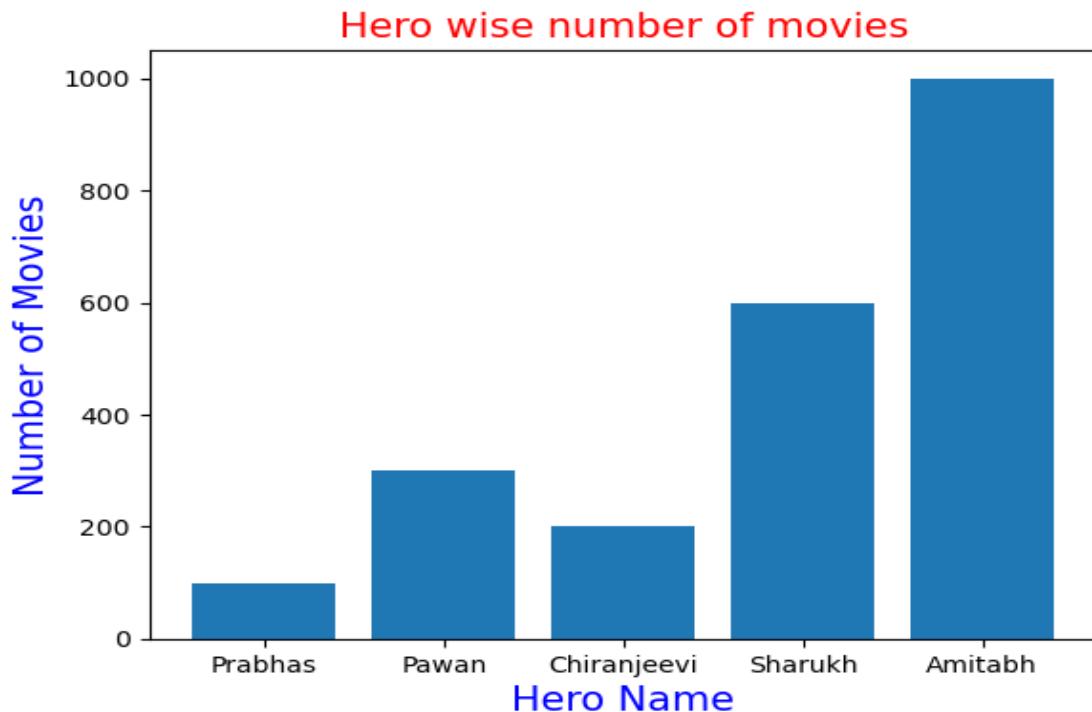
`'edge': Align the left edges of the bars with the x positions.`

`To align the bars on the right edge pass a negative width and align='edge'.`

### Represent the number of movies of each hero by using bar chart

In [105]:

```
import matplotlib.pyplot as plt
heroes = ['Prabhas','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies)
plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.show()
```



---

## We can customize several things like

- ✓ changing color of each bar
- ✓ changing width of each bar
- ✓ changing bottom of each bar
- ✓ changing alignments etc

### Observations

a) plt.bar(heroes,movies,color='r') ==> Now all bars with RED color

b) Separate color for each bar

```
c = ['r','b','k','g','orange']  
plt.bar(heroes,movies,color=c)
```

c) The width of each bar should be 0.5( default is 0.8)

```
plt.bar(heroes,movies,width=0.5)
```

d) Different widths for bars

```
w = [0.8,0.6,0.7,0.9,0.5]  
plt.bar(heroes,movies,width=w)
```

e) bottom should be 50 instead of 0

```
plt.bar(heroes,movies,bottom=50)
```

f) Different bottom values for bar?

```
b=[0,10,30,50,70]  
plt.bar(heroes,movies,bottom=b)
```

g) alignment: center

for **left alignment**: plt.bar(heroes,movies,align='edge')

for **right alignment**: plt.bar(heroes,movies,align=-0.8,align='edge')

---

### All bars with RED color

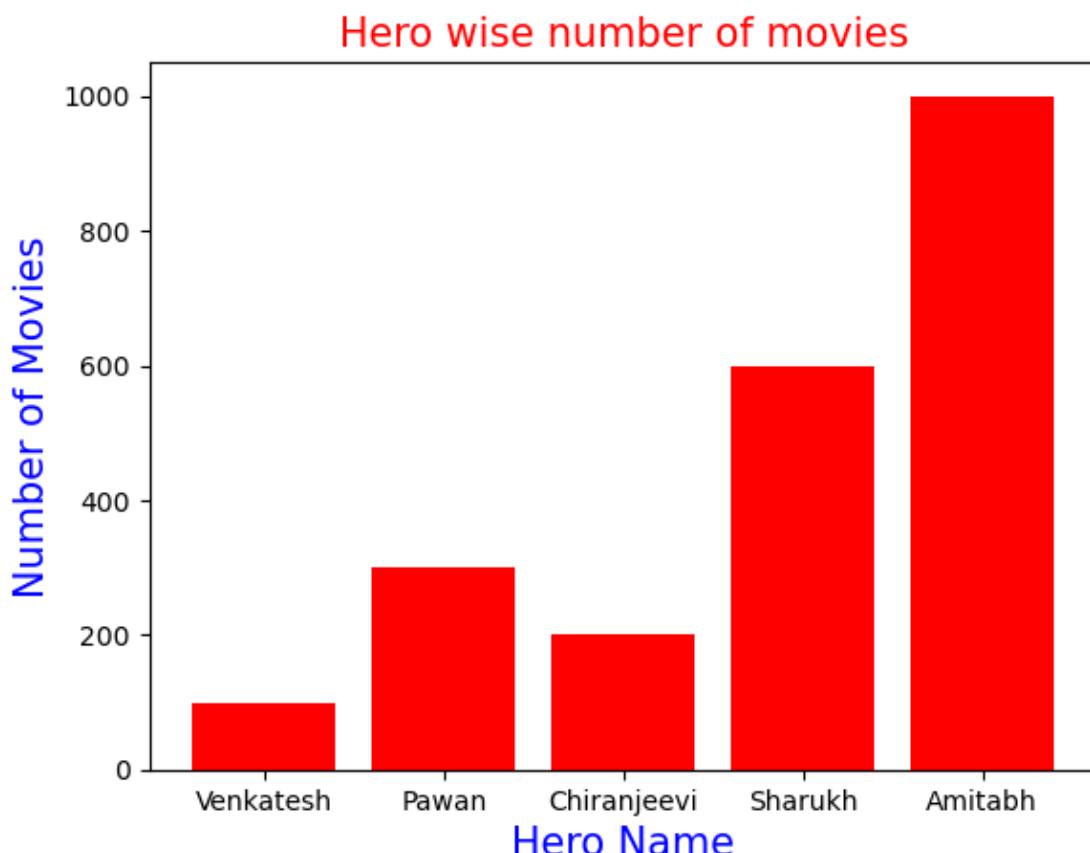
```
plt.bar(heroes,movies,color='r')
```

In [106]:

```
import matplotlib.pyplot as plt

heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies,color='r')

plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.show()
```



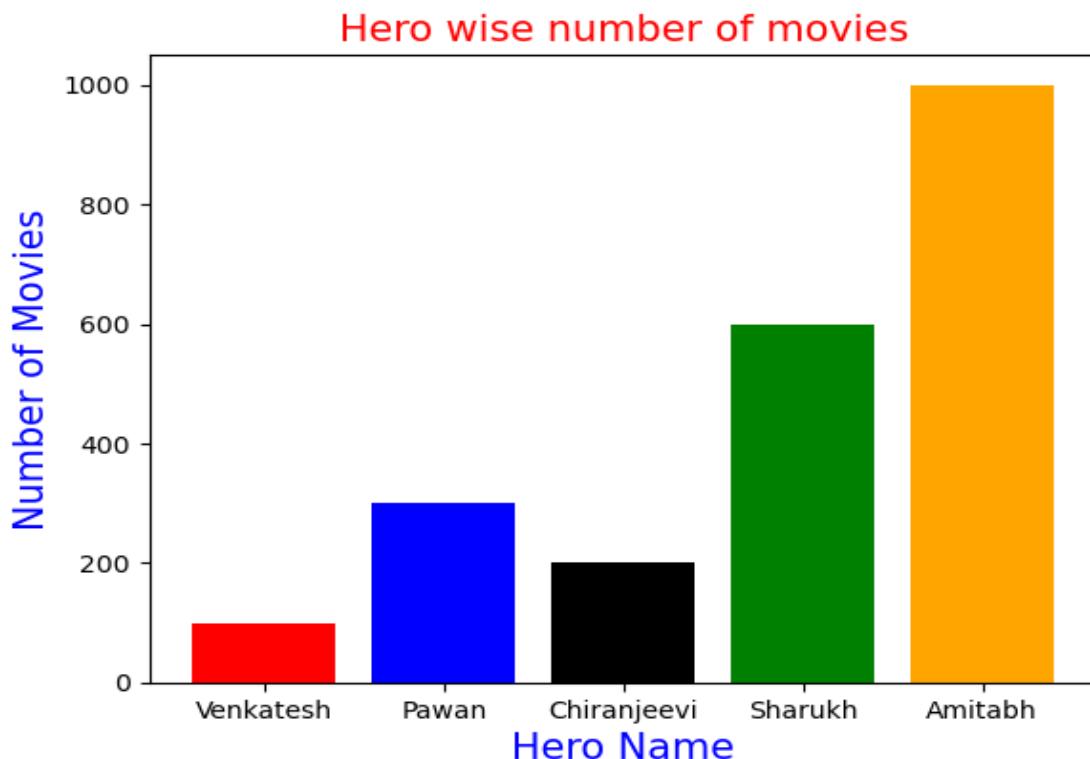
---

### Separate color for each bar

```
c = ['r','b','k','g','orange']  
plt.bar(heroes,movies,color=c)
```

In [107]:

```
import matplotlib.pyplot as plt  
  
heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values  
movies = [100,300,200,600,1000] #height of bars, values for y-axis  
c = ['r','b','k','g','orange']  
plt.bar(heroes,movies,color=c)  
  
plt.xlabel('Hero Name',color='b',fontsize=15)  
plt.ylabel('Number of Movies',color='b',fontsize=15)  
plt.title('Hero wise number of movies',color='r',fontsize=15)  
plt.show()
```



**The width of each bar should be 0.5( default is 0.8)**

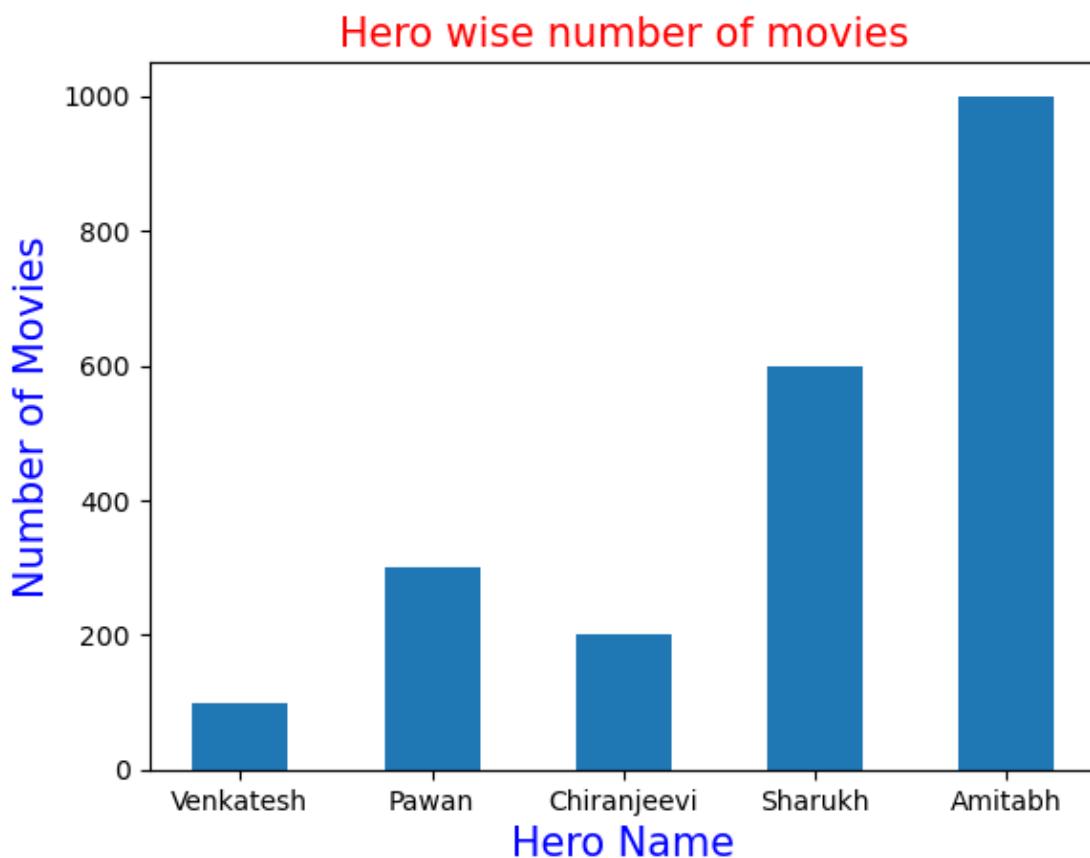
```
plt.bar(heroes,movies,width=0.5 )
```

In [108]:

```
import matplotlib.pyplot as plt

heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies,width=0.5 )

plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.show()
```



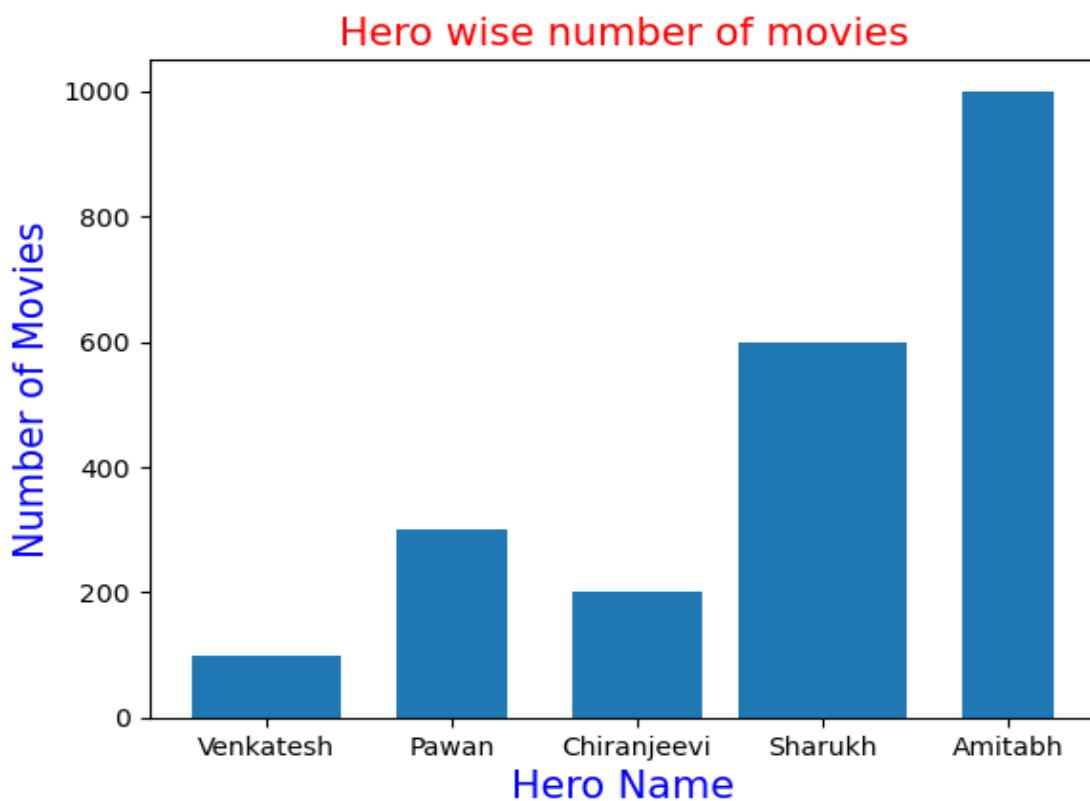
---

### Different widths for bars

```
w = [0.8,0.6,0.7,0.9,0.5]  
plt.bar(heroes,movies,width=w)
```

In [109]:

```
import matplotlib.pyplot as plt  
  
heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values  
movies = [100,300,200,600,1000] #height of bars, values for y-axis  
w = [0.8,0.6,0.7,0.9,0.5]  
plt.bar(heroes,movies,width=w)  
  
plt.xlabel('Hero Name',color='b',fontsize=15)  
plt.ylabel('Number of Movies',color='b',fontsize=15)  
plt.title('Hero wise number of movies',color='r',fontsize=15)  
plt.show()
```



**bottom should be 50 instead of 0**

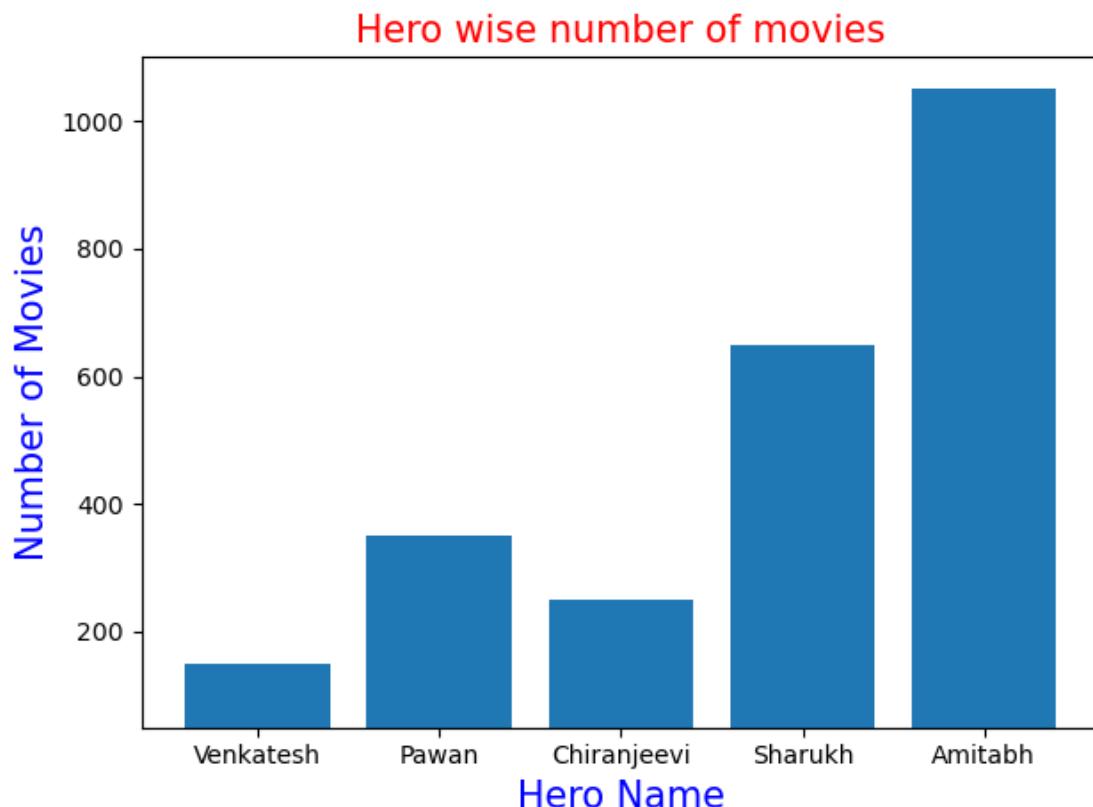
```
plt.bar(heroes,movies,bottom=50)
```

In [110]:

```
import matplotlib.pyplot as plt

heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies,bottom=50)

plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.tight_layout()
plt.show()
```



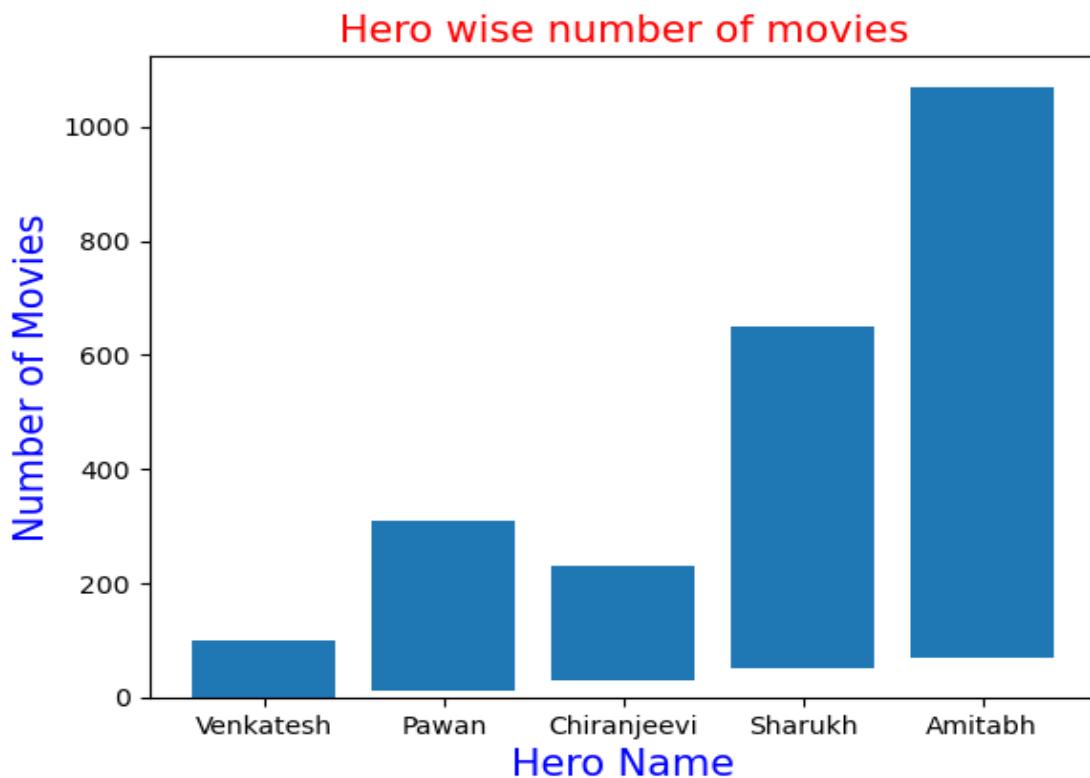
---

### Different bottom values for bar

```
b=[0,10,30,50,70]  
plt.bar(heroes,movies,bottom=b)
```

```
In [111]:
```

```
import matplotlib.pyplot as plt  
  
heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values  
movies = [100,300,200,600,1000] #height of bars, values for y-axis  
b=[0,10,30,50,70]  
plt.bar(heroes,movies,bottom=b)  
  
plt.xlabel('Hero Name',color='b',fontsize=15)  
plt.ylabel('Number of Movies',color='b',fontsize=15)  
plt.title('Hero wise number of movies',color='r',fontsize=15)  
plt.show()
```



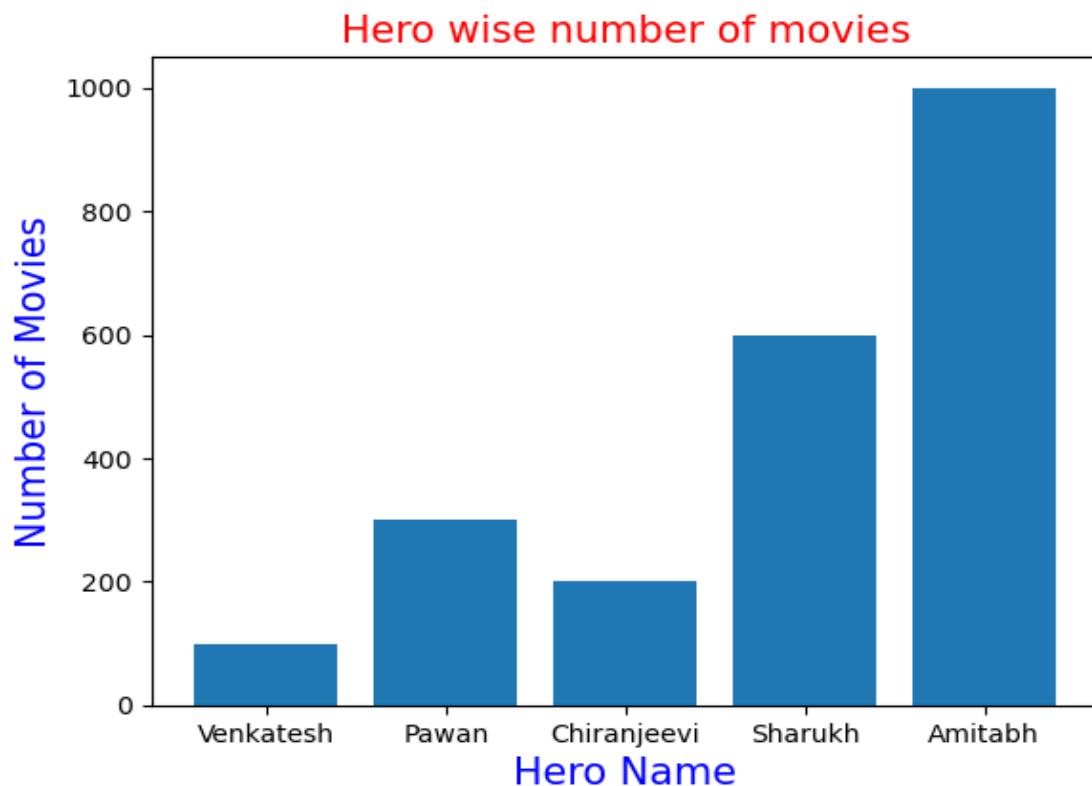
```
alignment: center
plt.bar(heroes,movies,align='center')
plt.bar(heroes,movies) # By default the alignment is center
```

In [112]:

```
import matplotlib.pyplot as plt

heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies,align='center')

plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.show()
```



```
alignment: left
```

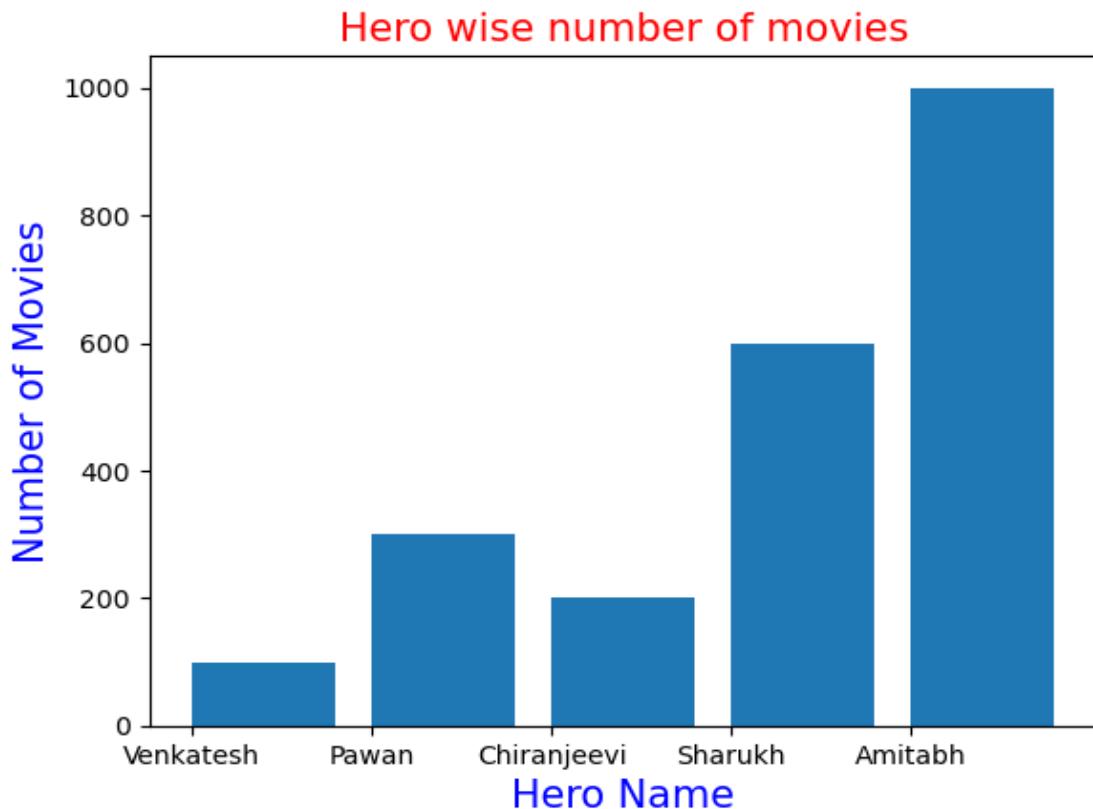
```
plt.bar(heroes,movies,align='edge')
```

```
In [113]:
```

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

```
heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values  
movies = [100,300,200,600,1000] #height of bars, values for y-axis  
plt.bar(heroes,movies,align='edge')
```

```
plt.xlabel('Hero Name',color='b',fontsize=15)  
plt.ylabel('Number of Movies',color='b',fontsize=15)  
plt.title('Hero wise number of movies',color='r',fontsize=15)  
plt.show()
```



```
alignment: right
```

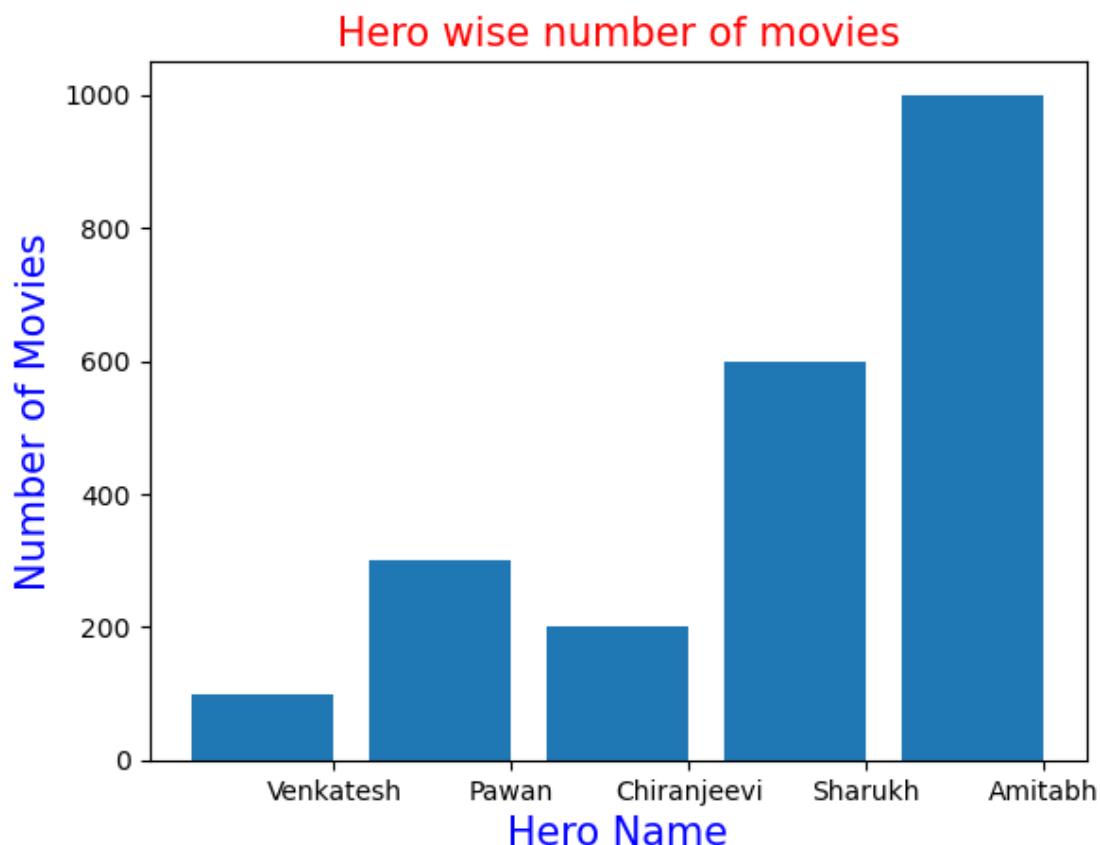
```
plt.bar(heroes,movies,width=-0.8,align='edge')
```

```
In [114]:
```

```
import matplotlib.pyplot as plt

heroes = ['Venkatesh','Pawan','Chiranjeevi','Sharukh','Amitabh'] # x-axis values
movies = [100,300,200,600,1000] #height of bars, values for y-axis
plt.bar(heroes,movies,width=-0.8,align='edge')

plt.xlabel('Hero Name',color='b',fontsize=15)
plt.ylabel('Number of Movies',color='b',fontsize=15)
plt.title('Hero wise number of movies',color='r',fontsize=15)
plt.show()
```



---

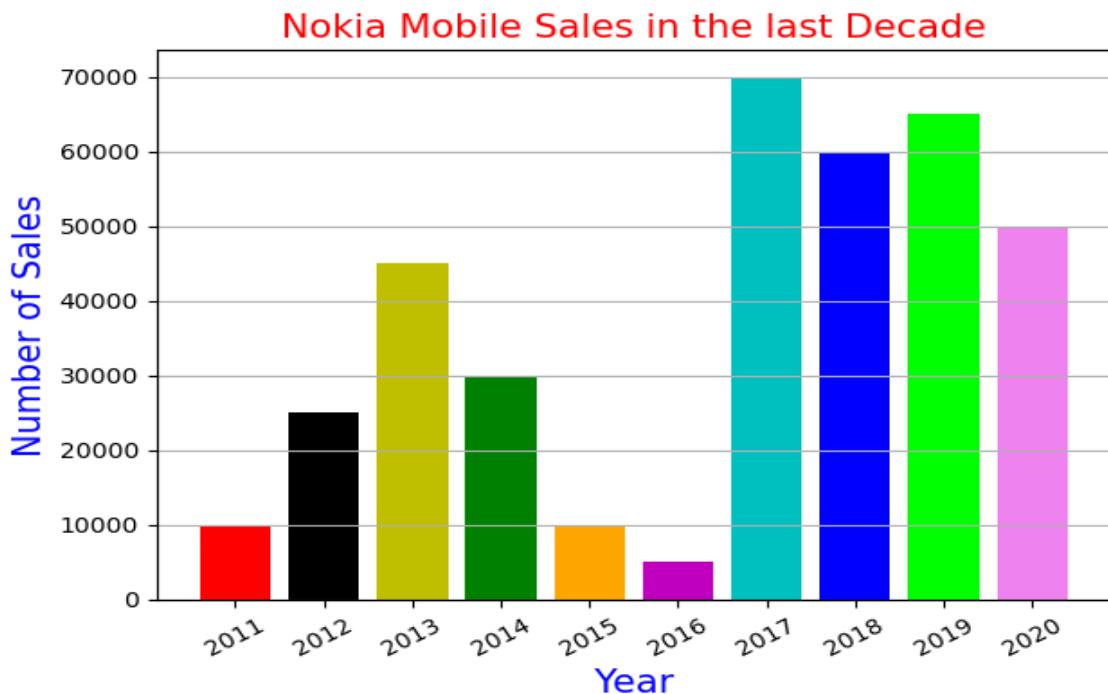
## Mobile Sales of Nokia Company from 2011 to 2020

In [115]:

```
import matplotlib.pyplot as plt

years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
sales = [10000, 25000, 45000, 30000, 10000,
         5000, 70000, 60000, 65000, 50000]
c = ['r','k','y','g','orange','m','c','b','lime','violet']
plt.bar(years,sales,color=c)

plt.xlabel('Year',color='b',fontsize=15)
plt.ylabel('Number of Sales',color='b',fontsize=15)
plt.title('Nokia Mobile Sales in the last Decade',color='r',fontsize=15)
plt.xticks(years,rotation=30)
plt.tight_layout()
plt.grid(axis='y')
plt.show()
```



---

## How to add labels to the bar

We can add labels to any plot by using the following 2 functions

1. pyplot.text()
2. pyplot.annotate()

### pyplot.text()

In [116]:

```
import matplotlib.pyplot as plt
help(plt.text)
```

Help on function text in module matplotlib.pyplot:

```
text(x, y, s, fontdict=None, **kwargs)
    Add text to the axes.
```

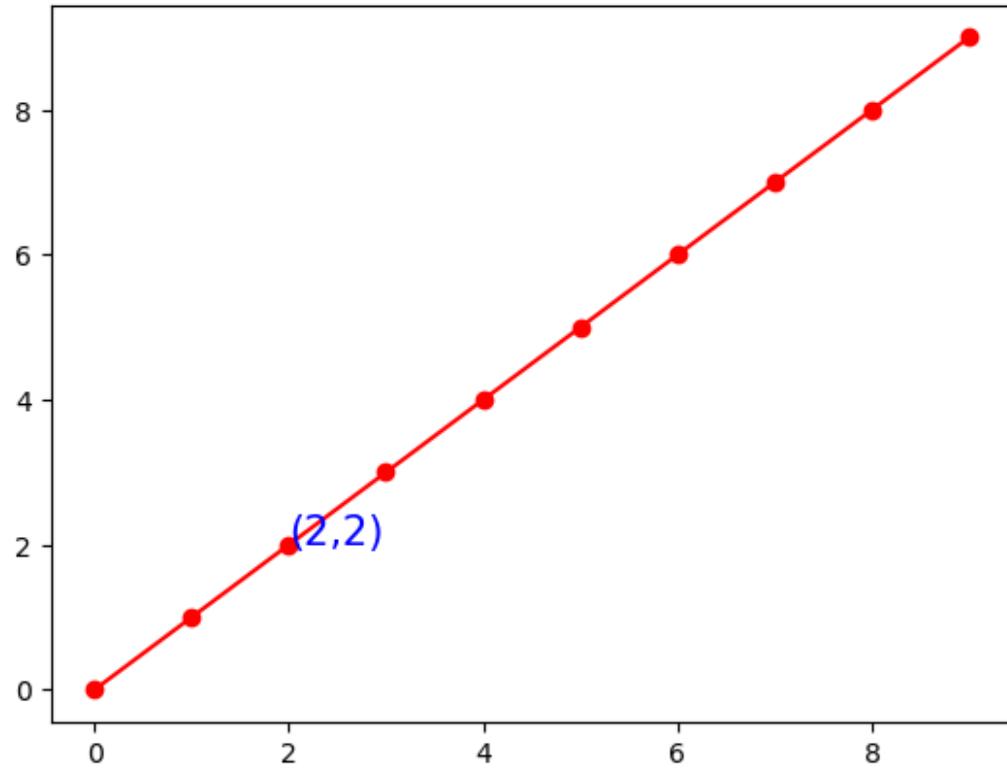
Add the text \*s\* to the axes at location \*x\*, \*y\* in data coordinates.

### Adding Labels for the data points of lineplot:

In [117]:

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

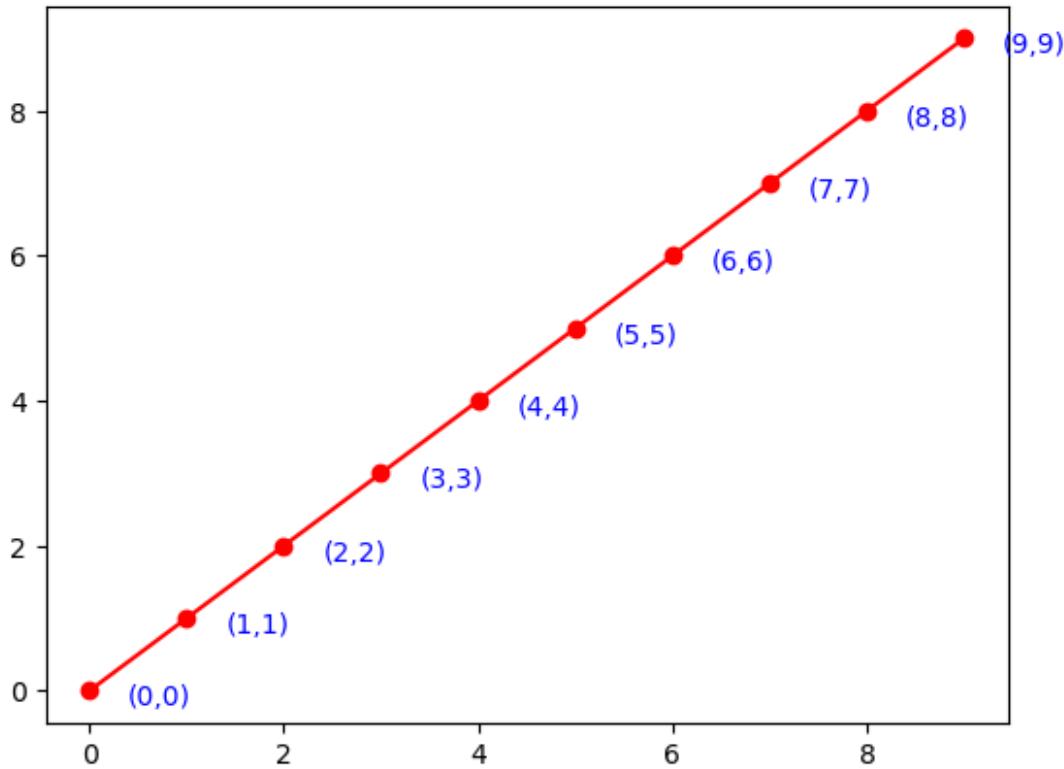
a = np.arange(10)
plt.plot(a,a,'r-o')
plt.text(2,2,(2,2),color='b',size=15)
plt.show()
```



In [118]:

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

a = np.arange(10)
plt.plot(a,a,'o-r')
for i in range(a.size): # 0 to 9
    plt.text(a[i]+0.4,a[i]-0.2,f'({a[i]},{a[i]})',color='b')
plt.show()
```



```
pyplot.annotate()
```

```
In [119]:
```

```
import matplotlib.pyplot as plt  
help(plt.annotate)
```

Help on function `annotate` in module `matplotlib.pyplot`:

```
annotate(text, xy, *args, **kwargs)  
        Annotate the point *xy* with text *text*.
```

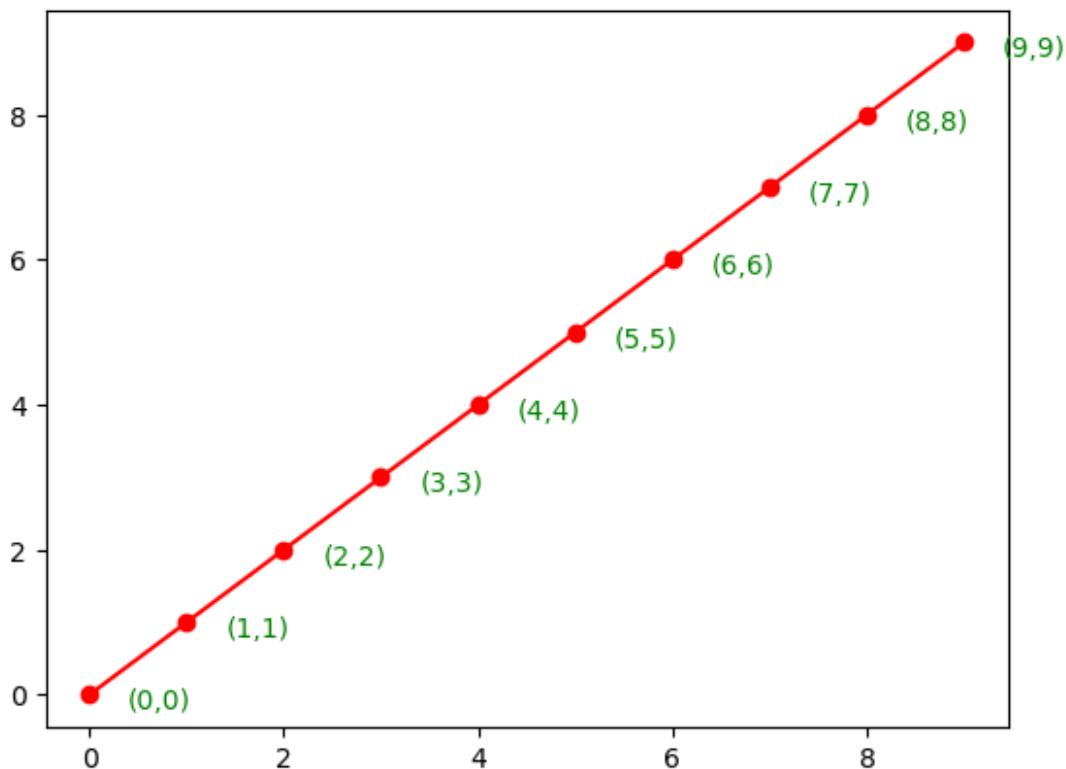
In the simplest form, the text is placed at \*xy\*.

---

In [120]:

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

a = np.arange(10)
plt.plot(a,a,'o-r')
for i in range(a.size): # 0 to 9
    #plt.text(a[i]+0.4,a[i]-0.2,f'({a[i]},{a[i]})',color='b')
    plt.annotate(f'({a[i]},{a[i]})',(a[i]+0.4,a[i]-0.2),color='g')
plt.show()
```



---

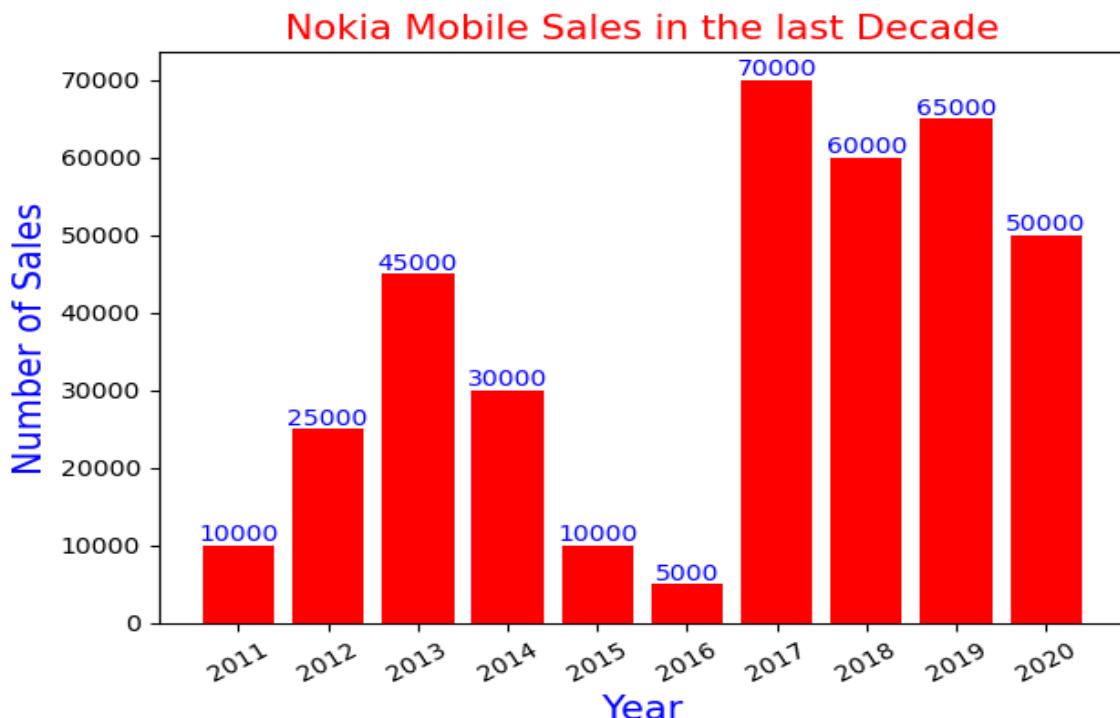
## How to add labels to the bar chart:

In [121]:

```
import matplotlib.pyplot as plt

years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
sales = [10000, 25000, 45000, 30000, 10000,
         5000,70000,60000,65000,50000]

plt.bar(years,sales,color='r')
plt.xlabel('Year',color='b',fontsize=15)
plt.ylabel('Number of Sales',color='b',fontsize=15)
plt.title('Nokia Mobile Sales in the last Decade',color='r',fontsize=15)
plt.xticks(years,rotation=30)
plt.tight_layout()
for i in range(len(years)): # 0 to 9
    plt.text(years[i],sales[i]+500,sales[i],ha='center',color='b')
plt.show()
```



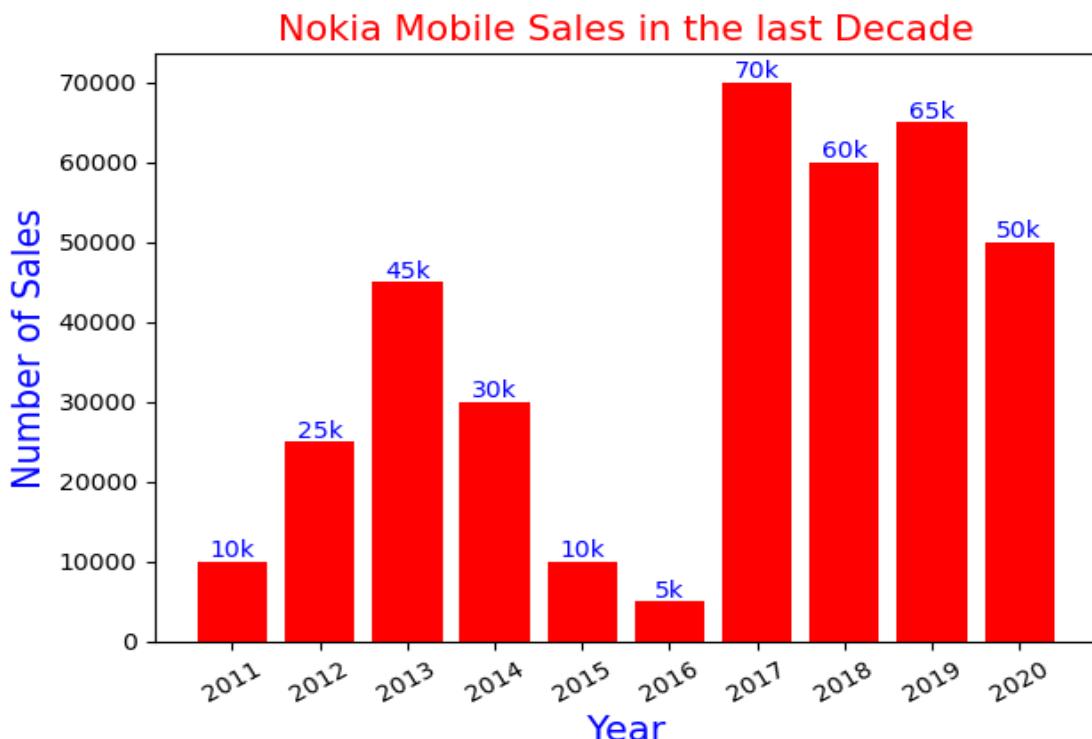
---

## With more readable labels

In [122]:

```
# using text() function
import matplotlib.pyplot as plt

years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
sales = [10000, 25000, 45000, 30000, 10000,
         5000,70000,60000,65000,50000]
plt.bar(years,sales,color='r')
plt.xlabel('Year',color='b',fontsize=15)
plt.ylabel('Number of Sales',color='b',fontsize=15)
plt.title('Nokia Mobile Sales in the last Decade',color='r',fontsize=15)
plt.xticks(years,rotation=30)
plt.tight_layout()
for i in range(len(years)): # 0 to 9
    plt.text(years[i],sales[i]+500,str(sales[i]//1000) + 'k',ha='center',color='b')
plt.show()
```

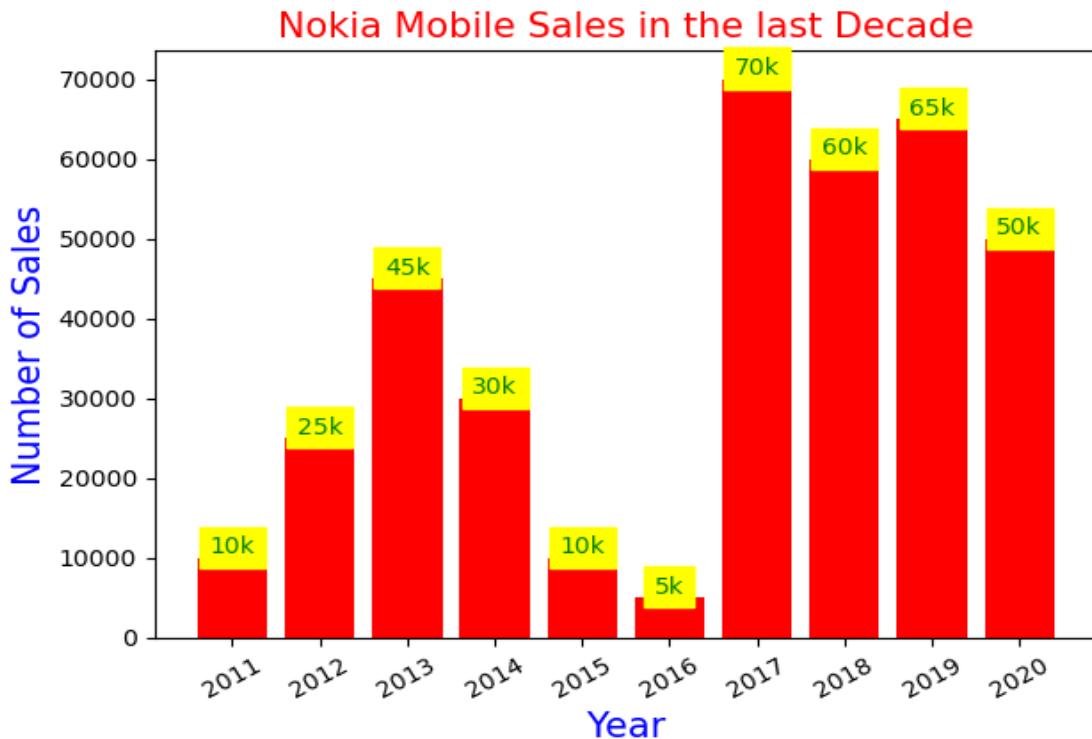


---

In [123]:

```
# using annotate() function
import matplotlib.pyplot as plt

years = [2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020]
sales = [10000, 25000, 45000, 30000, 10000,
         5000, 70000, 60000, 65000, 50000]
plt.bar(years,sales,color='r')
plt.xlabel('Year',color='b',fontsize=15)
plt.ylabel('Number of Sales',color='b',fontsize=15)
plt.title('Nokia Mobile Sales in the last Decade',color='r',fontsize=15)
plt.xticks(years,rotation=30)
plt.tight_layout()
for i in range(len(years)): # 0 to 9
    plt.annotate(str(sales[i]/1000) + 'k',(years[i],sales[i]+500),
                ha='center',color='g',backgroundcolor='yellow')
plt.show()
```



---

## Plotting bar chart with data from csv file

In [124]:

```
# Assume that data is available in students.csv file,
# # which is present in current working directory.
# students.csv
# -----
# Name of,Student Marks
# Sunny,100
# Bunny,200
# Chinny,300
# Vinny,200
# Pinny,400
# Zinny,300
# Kinny,500
# Minny,600
# Dinny,400
# Ginny,700
# Sachin,300
# Dravid,900
# Kohli,1000
# Rahul,800
# Ameer,600
# Sharukh,500
# Salman,700
# Ranveer,600
# Katrtina,300
# Kareena,400

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

names = np.array([],dtype='str')
marks = np.array([],dtype='int')
```

```

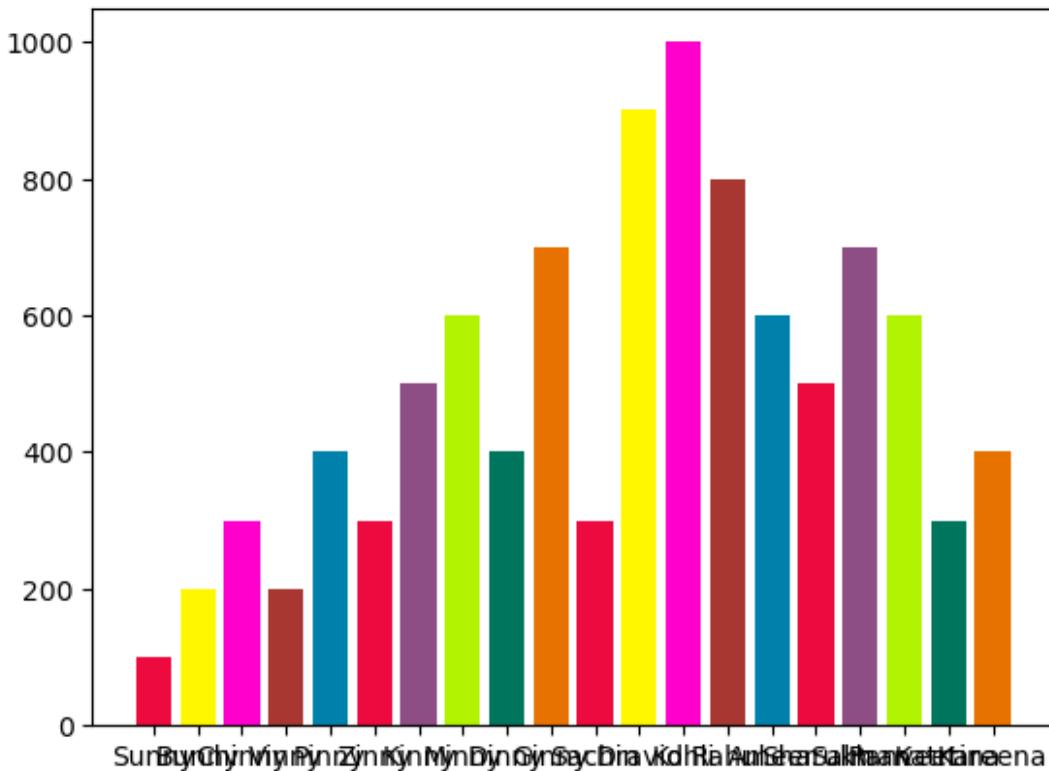
f = open('students.csv','r')

# Returns csvreader object
r = csv.reader(f)

#to read header and ignore
h = next(r)
for row in r:
    names = np.append(names,row[0])
    marks = np.append(marks,int(row[1]))

c = ['#ED0A3F','#FFF700', '#FF00CC','#A83731','#0081AB',
      '#ED0A3F','#8D4E85','#B2F302','#00755E','#E77200']
plt.bar(names,marks,color=c)
plt.show()

```



---

In [125]:

```
# Assume that data is available in students.csv file,
# which is present in current working directory.

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

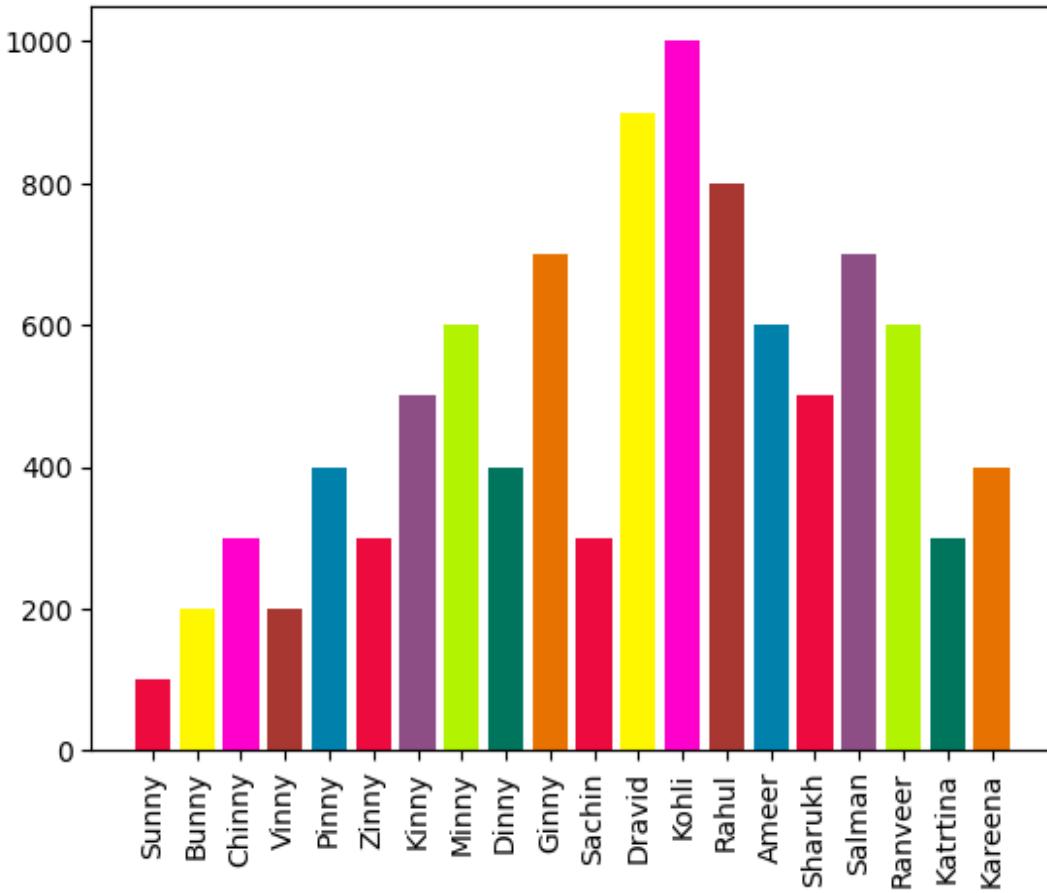
names = np.array([],dtype='str')
marks = np.array([],dtype='int')

f = open('students.csv','r')

# Returns csvreader object
r = csv.reader(f)

#to read header and ignore
h = next(r)
for row in r:
    names = np.append(names,row[0])
    marks = np.append(marks,int(row[1]))

c = ['#ED0A3F','#FFF700', '#FF00CC','#A83731','#0081AB',
      '#ED0A3F','#8D4E85','#B2F302','#00755E','#E77200']
plt.bar(names,marks,color=c)
plt.xticks(names,rotation=90)
plt.show()
```



### Horizontal bar chart:

- ✓ If the labels are too long or too many values to represent then we should go for horizontal bar chart instead of vertical bar chart.
- ✓ we will use **barh()** function
- ✓ Here the data will be represented in the form of horizontal bars.
- ✓ Each bar represents an individual category.
- ✓ The categories will be plotted on y-axis and data values will be plotted on x-axis.
- ✓ width/length of the bar is proportional to the value it represents.
- ✓ The default height is 0.8, but we can customize this value.

---

```
In [126]:
```

```
import matplotlib.pyplot as plt
help(plt.barh)
```

Help on function barh in module matplotlib.pyplot:

```
barh(y, width, height=0.8, left=None, *, align='center', **kwargs)
    Make a horizontal bar plot.
```

```
In [127]:
```

```
# Assume that data is available in students.csv file,
# which is present in current working directory.
```

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

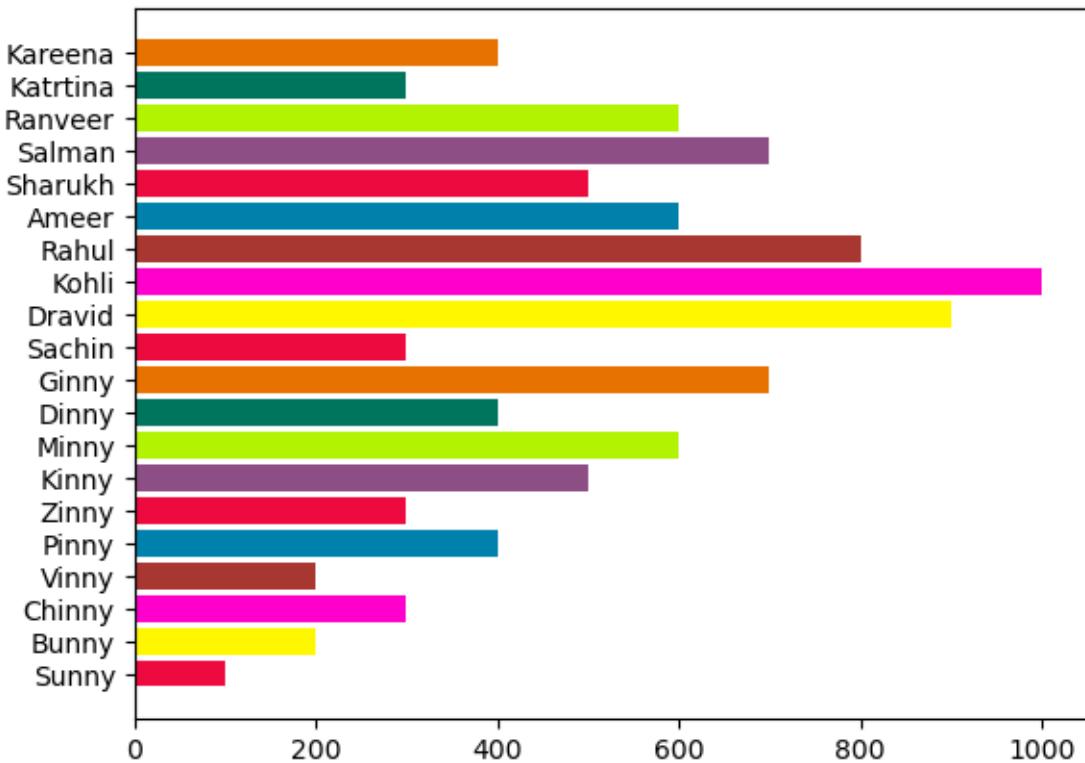
names = np.array([],dtype='str')
marks = np.array([],dtype='int')

f = open('students.csv','r')

# Returns csvreader object
r = csv.reader(f)

#to read header and ignore
h = next(r)
for row in r:
    names = np.append(names,row[0])
    marks = np.append(marks,int(row[1]))

c = ['#ED0A3F','#FFF700', '#FF00CC','#A83731','#0081AB',
      '#ED0A3F','#8D4E85','#B2F302','#00755E','#E77200']
plt.barh(names,marks,color=c)
plt.show()
```



### vertical vs horizontal

- ✓ height ---> width
- ✓ width ---> height
- ✓ bottom --> left
- ✓ bar() --> barh()

In [128]:

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

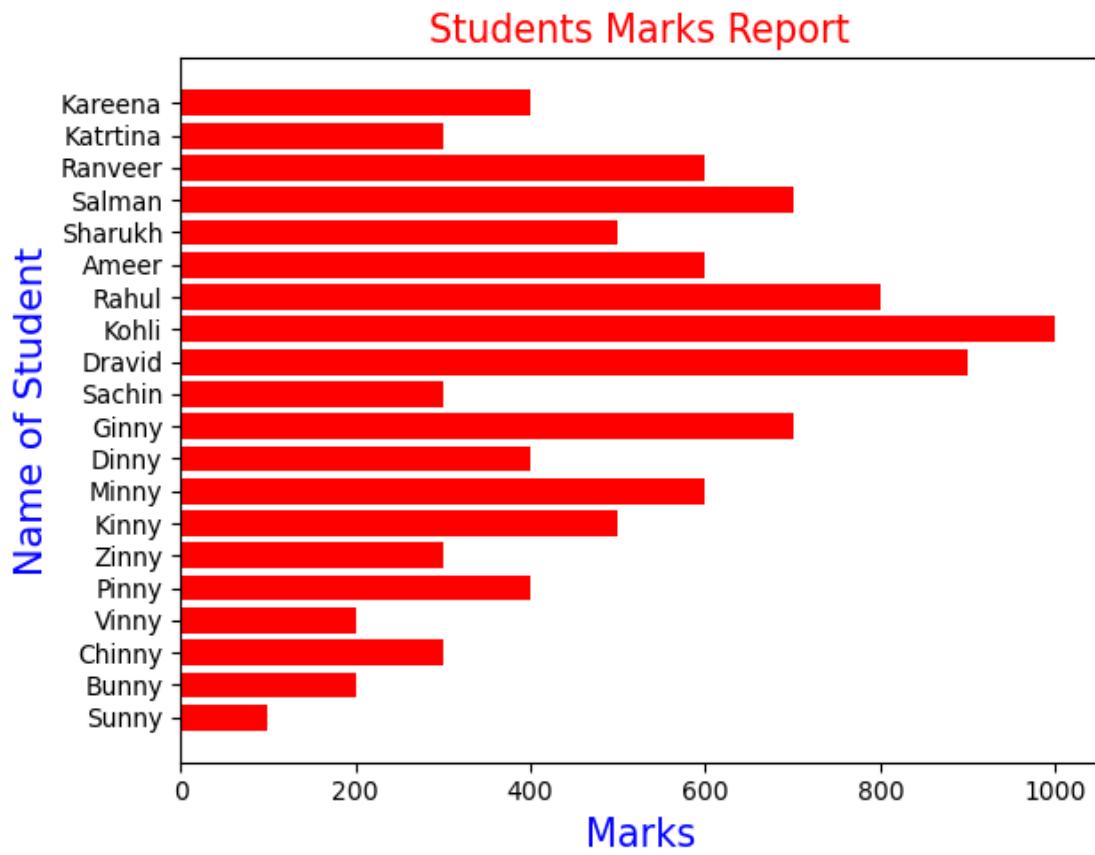
names = np.array([],dtype='str')
marks = np.array([],dtype='int')

f = open('students.csv','r')
r = csv.reader(f) # Returns csvreader object
```

```

h = next(r) #to read header and ignore
for row in r:
    names = np.append(names,row[0])
    marks = np.append(marks,int(row[1]))
plt.barh(names,marks,color='r')
plt.xlabel('Marks',fontsize=15,color='b')
plt.ylabel('Name of Student',fontsize=15,color='b')
plt.title('Students Marks Report',fontsize=15,color='r')
plt.tight_layout()
plt.show()

```



---

## Stacked Bar chart:

- ✓ If each category contains multiple subcategories then we should go for stacked bar chart.
- ✓ Here each subcategory will be plotted on top of other subcategory.

**Eg-1:** Country wise total population we have to represent. But in that population we have to plot separately men and women.

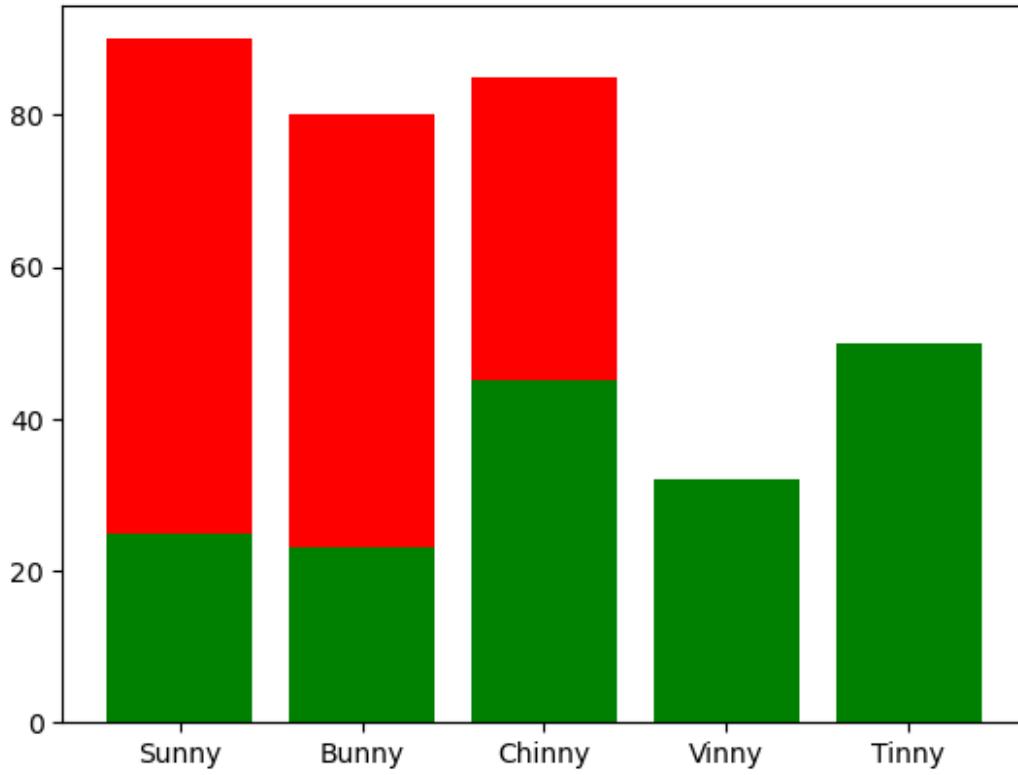
**Eg-2:** Country wise medals we have to represent. But in that total number of medals, we have to represent gold, silver and bronze medals separately.

- ✓ **verical bar chart** ==> bar()
- ✓ **horizontal bar chart** ==> barh()
- ✓ **stacked bar chart** ==> either bar() or barh()
- ✓ The stacked bar chart can be either vertical or horizontal

In [129]:

```
# demo program-1: marks wise student data
import matplotlib.pyplot as plt

names = ['Sunny','Bunny','Chinny','Vinny','Tinny']
english_marks = [90,80,85,25,50]
maths_marks = [25,23,45,32,50]
plt.bar(names,english_marks,color='r')
plt.bar(names, maths_marks, color='g')
plt.show()
```



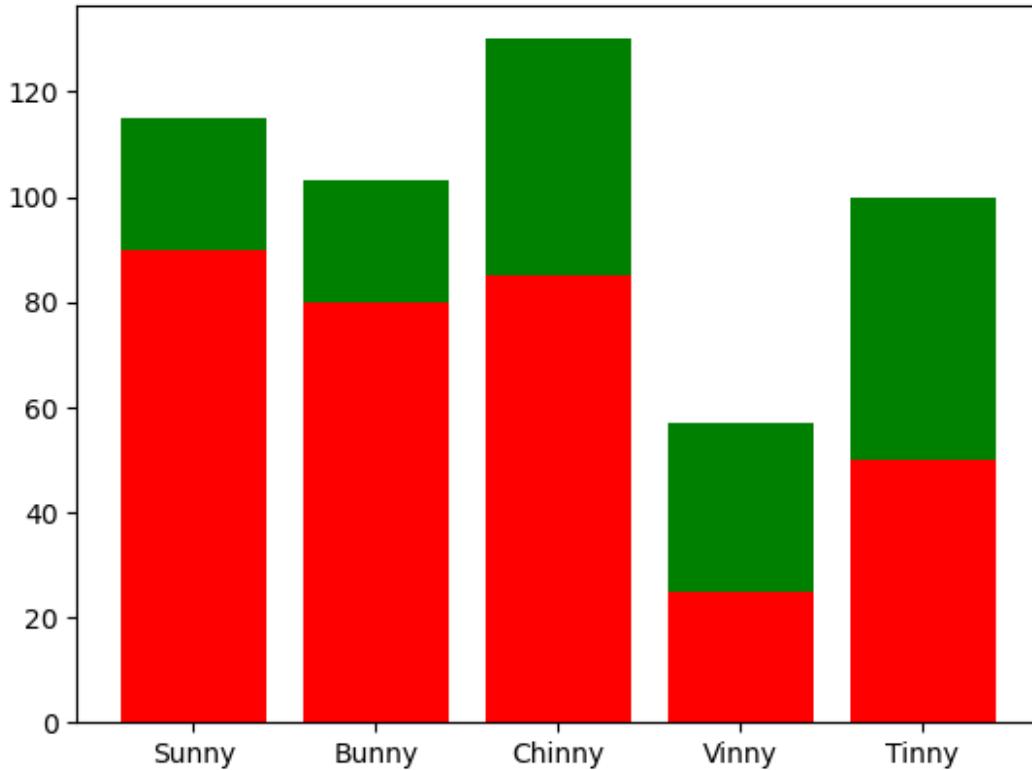
### Note

- ✓ In the above program overlapping will happen because both bars are started from 0 itself
- ✓ to solve this issue we will use **bottom** property from which position the second bar should start

In [130]:

```
# by using bottom property
import matplotlib.pyplot as plt

names = ['Sunny','Bunny','Chinny','Vinny','Tinny']
english_marks = [90,80,85,25,50]
maths_marks = [25,23,45,32,50]
plt.bar(names,english_marks,color='r')
plt.bar(names, maths_marks, bottom=english_marks, color='green')
plt.show()
```



```
In [131]:
```

```
# with text labels on the bar

import matplotlib.pyplot as plt
import numpy as np
names = ['Sunny','Bunny','Chinny','Vinny','Tinny']
english_marks = np.array([90,80,85,25,50])
math_marks = np.array([25,23,45,32,25])
total_marks = english_marks+math_marks

plt.bar(names,english_marks,color="#09695c",label='English')
plt.bar(names,math_marks,bottom=english_marks,color="#9c0c8b",label="Math
s")

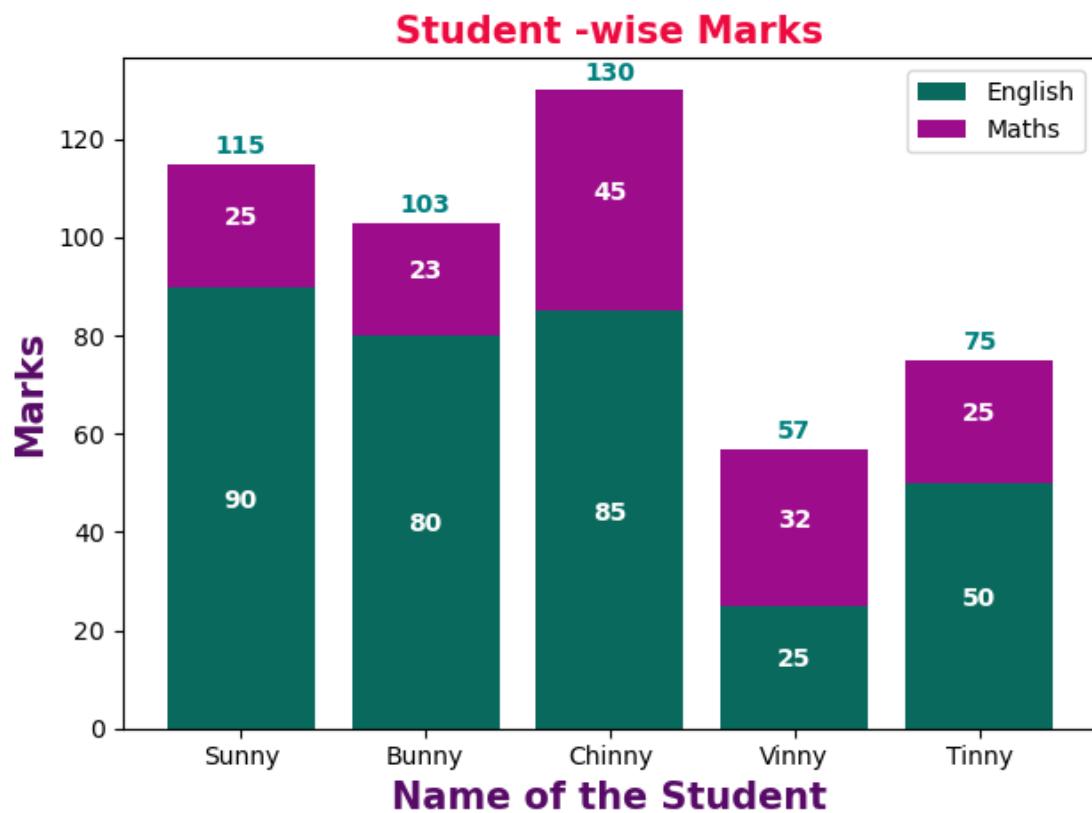
for i in range(len(names)):
    plt.text(names[i],(english_marks[i]/2),str(english_marks[i]),
```

```

ha='center',color='white',weight=1000)
plt.text(names[i],(english_marks[i]+math_marks[i]/2),str(math_marks[i]),
         ha='center',color='white',weight=1000)
plt.text(names[i],(total_marks[i]+2), str(total_marks[i])),
         ha='center',color='#008080',weight=1000)

plt.xlabel("Name of the Student",color="#570b66", fontsize=15,weight=1000)
plt.ylabel("Marks",color="#570b66", fontsize=15,weight=1000)
plt.title("Student -wise Marks",color="#ED0A3F", fontsize=15,weight=1000)
plt.legend()
plt.tight_layout()
plt.show()

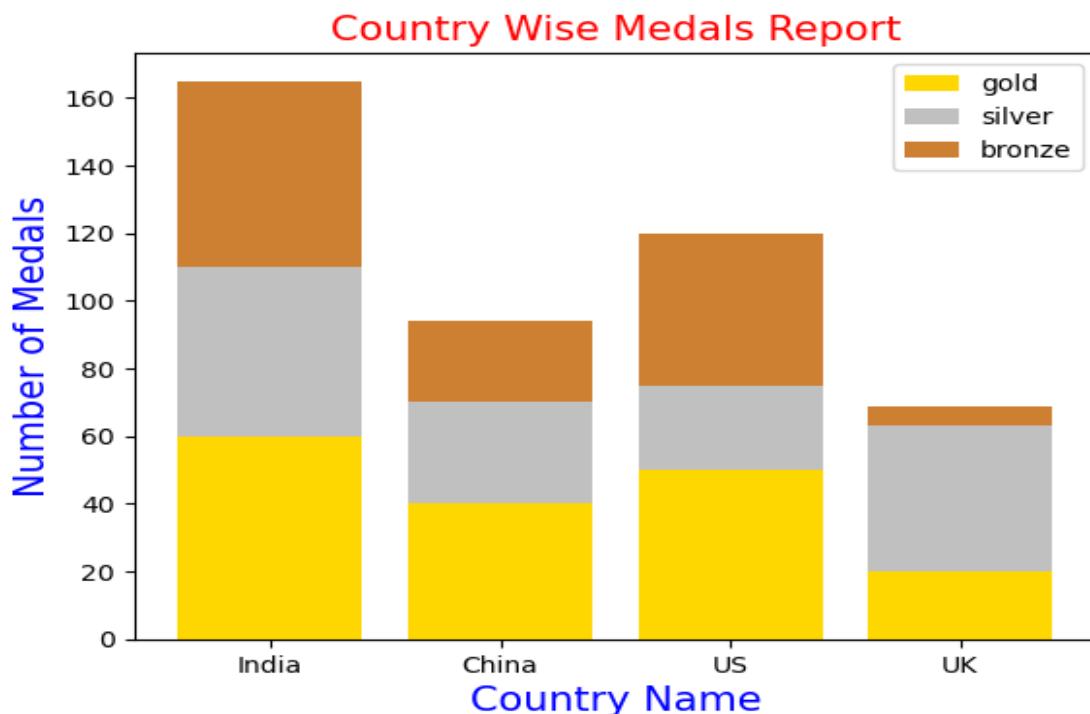
```



---

In [132]:

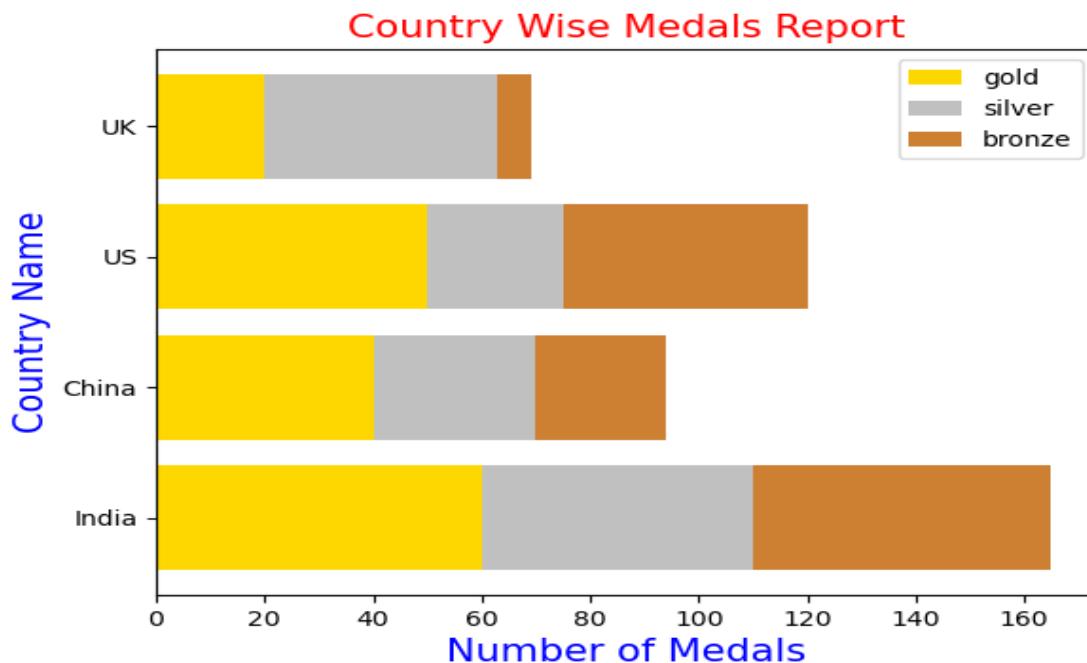
```
#eg-2: country wise medals but sub categories
import matplotlib.pyplot as plt
import numpy as np
country_name = ['India','China','US','UK']
gold_medals = np.array([60,40,50,20])
silver_medals = np.array([50,30,25,43])
bronze_medals = np.array([55,24,45,6])
plt.bar(country_name,gold_medals,color='#FFD700',label='gold')
plt.bar(country_name,silver_medals,
        bottom = gold_medals,color='#C0C0C0',label='silver')
plt.bar(country_name,bronze_medals,
        bottom = gold_medals+silver_medals ,color='#CD7F32',label='bronze')
plt.xlabel('Country Name',color='b',fontsize=15)
plt.ylabel('Number of Medals',color='b',fontsize=15)
plt.title('Country Wise Medals Report',color='r',fontsize=15)
plt.legend()
plt.show()
```



---

In [133]:

```
# eg-3: Stacked Horizontal Bar Chart:  
import matplotlib.pyplot as plt  
import numpy as np  
country_name = ['India','China','US','UK']  
gold_medals = np.array([60,40,50,20])  
silver_medals = np.array([50,30,25,43])  
bronze_medals = np.array([55,24,45,6])  
plt.barh(country_name, gold_medals, color='#FFD700', label='gold')  
plt.barh(country_name, silver_medals,  
         left = gold_medals, color='#C0C0C0', label='silver')  
plt.barh(country_name, bronze_medals,  
         left = gold_medals+silver_medals ,color='#CD7F32', label='bronze')  
plt.ylabel('Country Name',color='b',fontsize=15)  
plt.xlabel('Number of Medals',color='b',fontsize=15)  
plt.title('Country Wise Medals Report',color='r',fontsize=15)  
plt.legend()  
plt.show()
```



---

## Note

### To change the vertical bar to horizontal bar

- ✓ bar() → barh()
- ✓ bottom → left
- ✓ xlabel and ylabel are interchanged

### Clustered Bar chart/Grouped Bar Chart/Multiple Bar Chart:

- ✓ If each category contains multiple sub categories and if we want to represent all these sub categories side by side then we should go for **Clustered Bar Chart**.

**eg-1:** Country wise total population we have to represent. But in that population we have to plot separately men and women side by side.

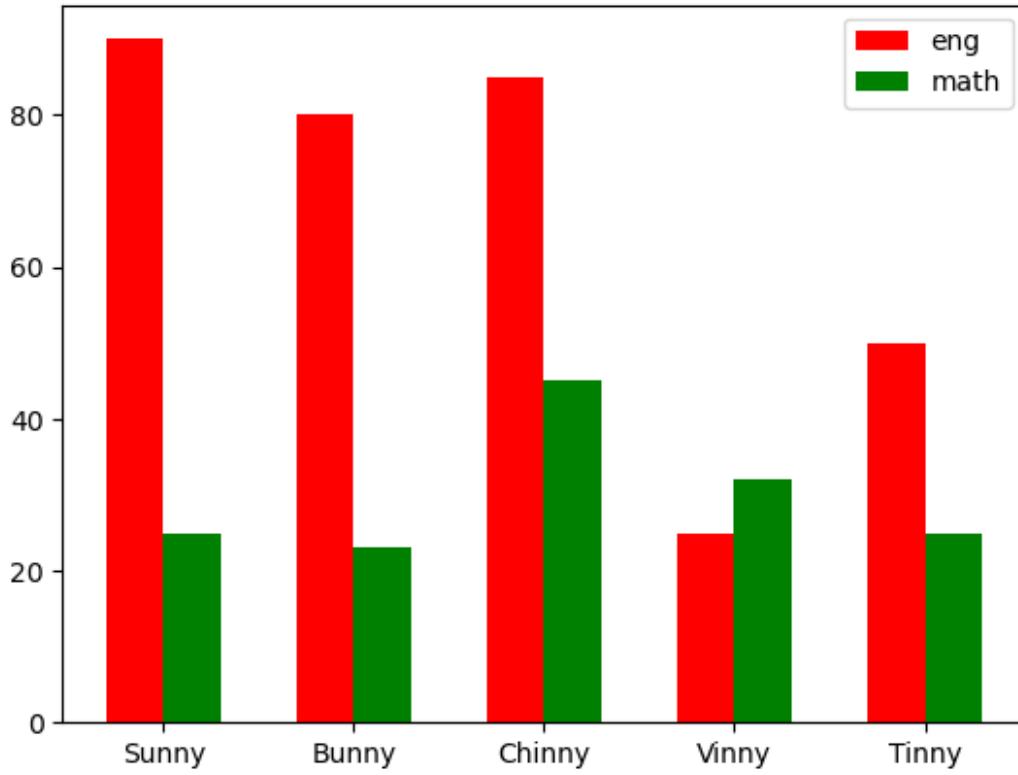
**eg-2:** Country wise medals we have to represent. But in that total number of medals, we have to represent gold, silver and bronze medals separately side by side.

- ✓ We can create clustered bar chart by using either bar() or barh() functions.

In [134]:

```
# Demo program

import matplotlib.pyplot as plt
import numpy as np
names = ['Sunny', 'Bunny', 'Chinny', 'Vinny', 'Tinny']
english_marks = np.array([90, 80, 85, 25, 50])
math_marks = np.array([25, 23, 45, 32, 25])
xpos = np.arange(len(names)) #[0,1,2,3,4]
w = 0.3
plt.bar(xpos, english_marks, color='r', width=w)
plt.bar(xpos+w, math_marks, color='g', width=w)
# plt.xticks(xpos+0.15, names)
plt.xticks(xpos+w/2, names)
plt.legend(['eng', 'math'])
plt.show()
```



In [135]:

```
# Demo program-2:  
import matplotlib.pyplot as plt  
import numpy as np  
country_name = ['India','China','US','UK']  
gold_medals = np.array([60,40,50,20])  
silver_medals = np.array([50,30,25,43])  
bronze_medals = np.array([55,24,45,6])  
xpos = np.arange(len(country_name)) #[0,1,2,3]  
w = 0.2  
  
plt.bar(xpos, gold_medals, color='#FFD700', width=w)  
plt.bar(xpos+w, silver_medals, color='#C0C0C0', width=w)  
plt.bar(xpos+2*w, bronze_medals, color='#CD7F32', width=w)  
  
plt.xticks(xpos+w, country_name)
```

```

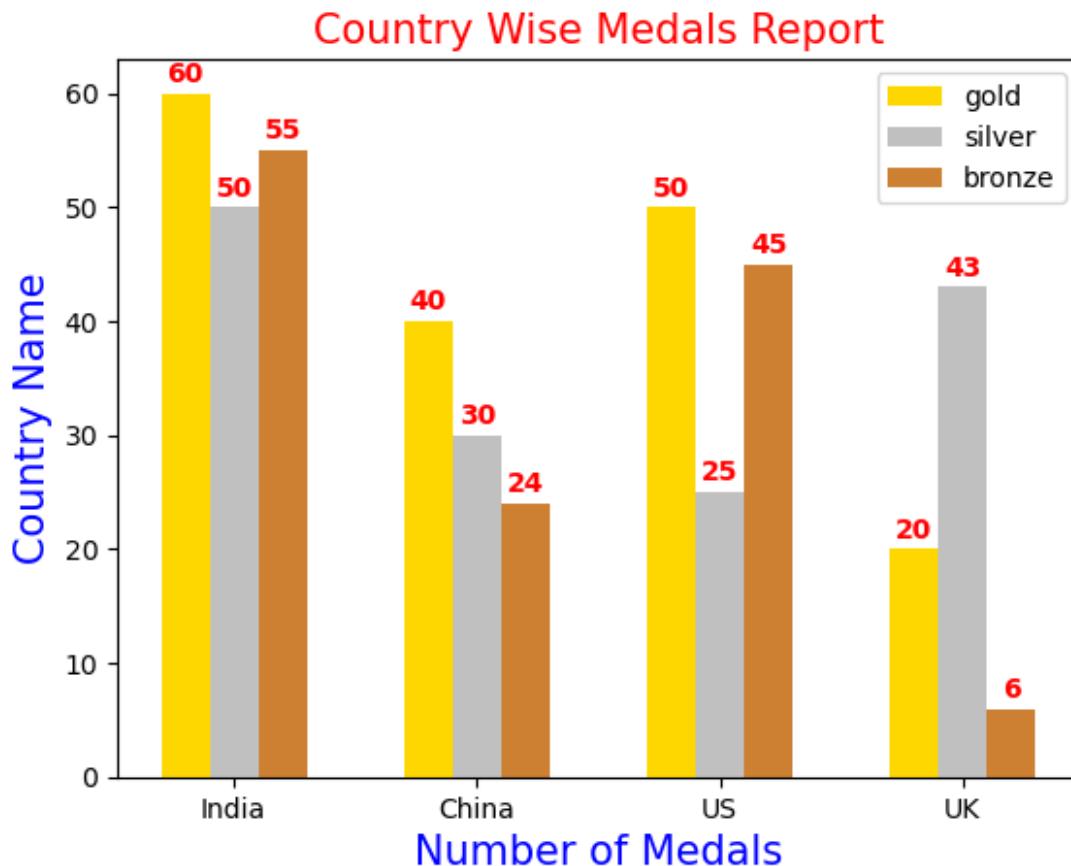
plt.ylabel('Country Name',color='b',fontsize=15)
plt.xlabel('Number of Medals',color='b',fontsize=15)
plt.title('Country Wise Medals Report',color='r',fontsize=15)
plt.legend(['gold','silver','bronze'])

for i in range(len(country_name)):

    plt.text(xpos[i],gold_medals[i]+1,gold_medals[i],
              ha='center',color='r',weight=1000)
    plt.text(xpos[i]+w,silver_medals[i]+1,silver_medals[i],
              ha='center',color='r',weight=1000)
    plt.text(xpos[i]+2*w,bronze_medals[i]+1,bronze_medals[i],
              ha='center',color='r',weight=1000)

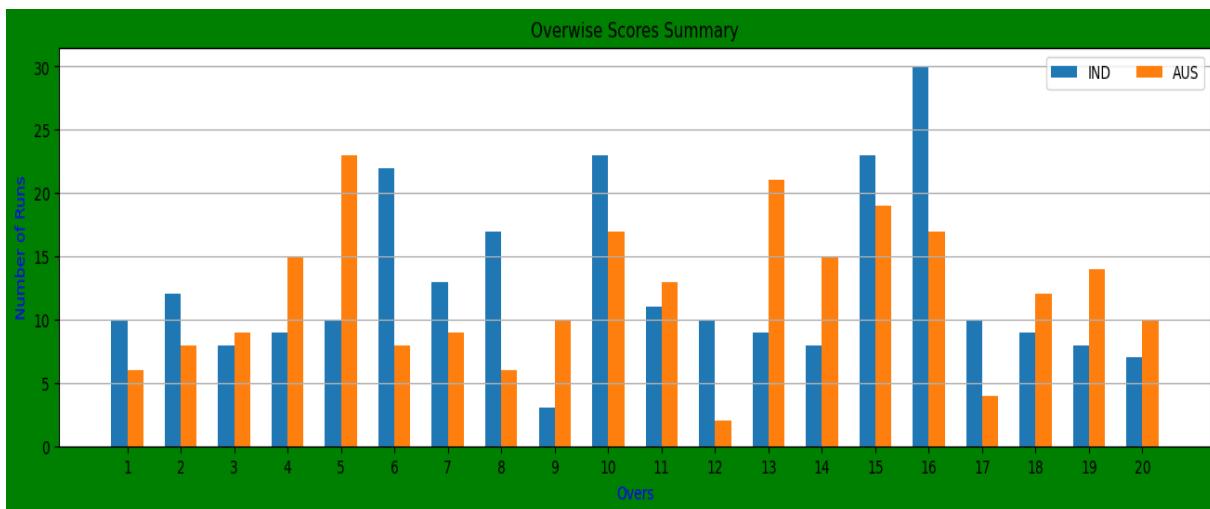
plt.show()

```



In [136]:

```
#eg-2A: India and Australia 20-20 overwise scores required to represent by
using
# clustered bar chart
import matplotlib.pyplot as plt
import numpy as np
overs = np.arange(1,21)
xpos=np.arange(overs.size)
ind_score = [10,12,8,9,10,22,13,17,3,23,11,10,9,8,23,30,10,9,8,7]
aus_score = [6,8,9,15,23,8,9,6,10,17,13,2,21,15,19,17,4,12,14,10]
w=0.3
plt.figure(num=1,figsize=(16,4),facecolor='g')
plt.bar(xpos,ind_score,width=w)
plt.bar(xpos+w,aus_score,width=w)
plt.xticks(xpos+(w/2),labels=overs)
plt.xlabel('Overs',color='b')
plt.ylabel('Number of Runs',color='b')
plt.title('Overwise Scores Summary')
plt.legend(['IND','AUS'],ncol=2)
plt.grid(axis='y')
plt.show()
```

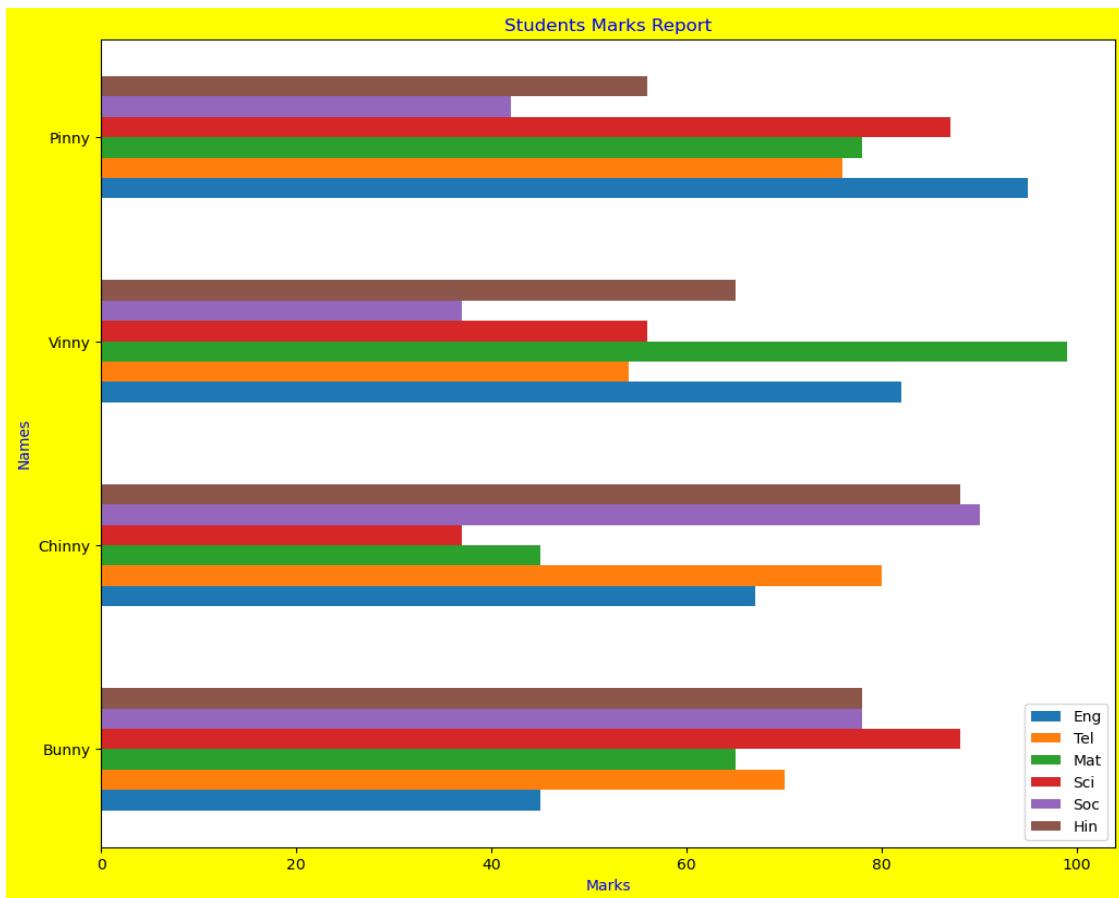


---

In [137]:

```
# horizontal clustered bar chart:
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(num=1,figsize=(12,10),facecolor='yellow')
names = ['Bunny','Chinny','Vinny','Pinny']
ypos = np.arange(len(names)) #[0,1,2,3]
h=0.1
english=np.array([45,67,82,95])
telugu=np.array([70,80,54,76])
maths=np.array([65,45,99,78])
science=np.array([88,37,56,87])
social=np.array([78,90,37,42])
hindi=np.array([78,88,65,56])
plt.barh(ypos,english,height=h)
plt.barh(ypos+h,telugu,height=h)
plt.barh(ypos+2*h,maths,height=h)
plt.barh(ypos+3*h,science,height=h)
plt.barh(ypos+4*h,social,height=h)
plt.barh(ypos+5*h,hindi,height=h)
plt.yticks(ypos+2.5*h,names)
plt.xlabel('Marks',color='b')
plt.ylabel('Names',color='b')
plt.title('Students Marks Report',color='b')
plt.legend(['Eng','Tel','Mat','Sci','Soc','Hin'])
plt.show()
```



## Summary

- ✓ If we want to compare different categories of values then we should go for bar chart. ie **vertical bar chart** and we can create by using **bar()** function.
- ✓ If we want to compare different categories of values and the **labels are too long or multiple values** to represent then we should go for **horizontal bar chart**. We can create by using **barh()** function.
- ✓ If we want to compare different categories of values and each category contains **multiple sub categories** and if we want to represent values **on top of other** then we should go for **stacked bar chart**. It can be **either vertical or horizontal**.
- ✓ If we want to compare different categories of values and each category contains **multiple sub categories** and if we want to represent values **side by side** then we should go for **clustered bar chart**. It can be **either vertical or horizontal**.

---

## Chapter-12

### Pie Chart

#### Pie Chart

- ✓ **Pie chart** is a circular chart divided into segments. These segments are called **wedges**.
- ✓ Each wedge represents an individual category. The area of the wedge is proportional to value of that category.
- ✓ Pie chart is very helpful for comparison of categories.
- ✓ The number of categories are less mostly <=5.  
Eg: 20 overs, overwise scores--->bar chart but not pie chart
- ✓ The chance of winning match-->pie chart
- ✓ By using **pie() function of pyplot → pie chart**

```
In [138]:
```

```
import matplotlib.pyplot as plt  
help(plt.pie)
```

Help on function pie in module matplotlib.pyplot:

```
pie(x, explode=None, labels=None, colors=None, autopct=None,  
pctdistance=0.6, shadow=False, labeldistance=1.1, startangle=0, radius=1,  
counterclock=True, wedgeprops=None, textprops=None, center=(0, 0),  
frame=False, rotatelabels=False, *, normalize=None, data=None)
```

Plot a pie chart.

---

```
In [139]:
```

```
# eg-1:
```

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
plt.pie(marks)
plt.show()
```



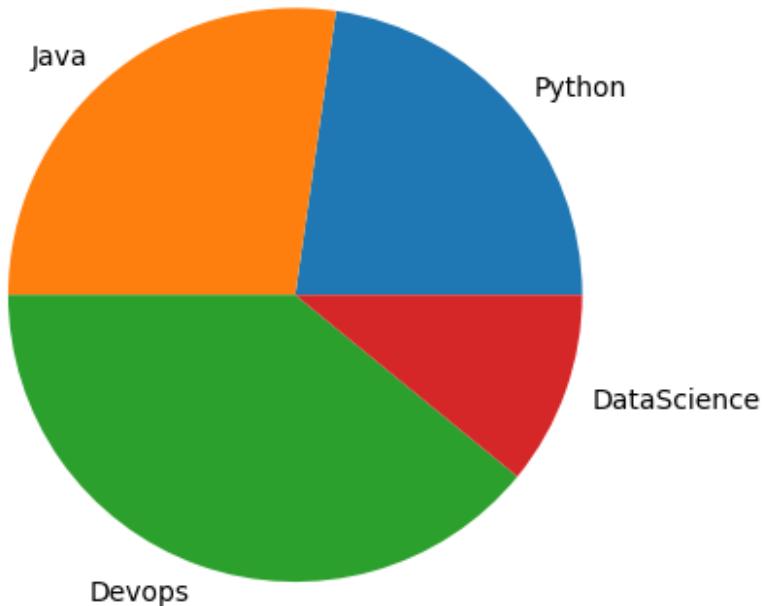
### Adding Labels ==> **labels argument**

- ✓ We can add labels to the wedges. For this we have to use **labels argument**.
  - ✓ It should be any sequence of strings.
-

---

In [140]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
plt.pie(marks,labels=mylabels)
plt.show()
```



### autopct

- ✓ **autopct** ==> auto percentage
- ✓ To label wedges with their numeric percentage(%)
- ✓ We can specify its value by using formatted string.

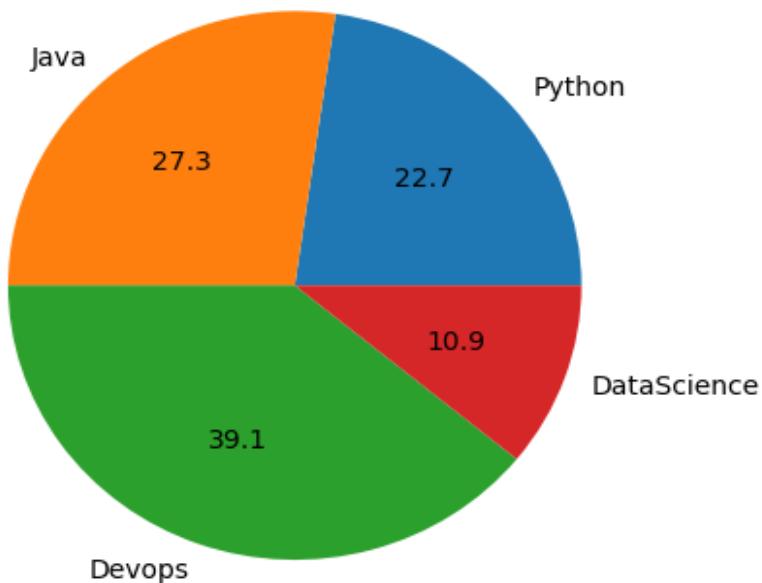
**After decimal point, if we want to consider only one digit**

```
autopct = '%.1f'
plt.pie(marks,labels=mylabels,autopct = '%.1f')
```

---

In [141]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
plt.pie(marks,labels=mylabels,autopct = '%.1f')
plt.show()
```



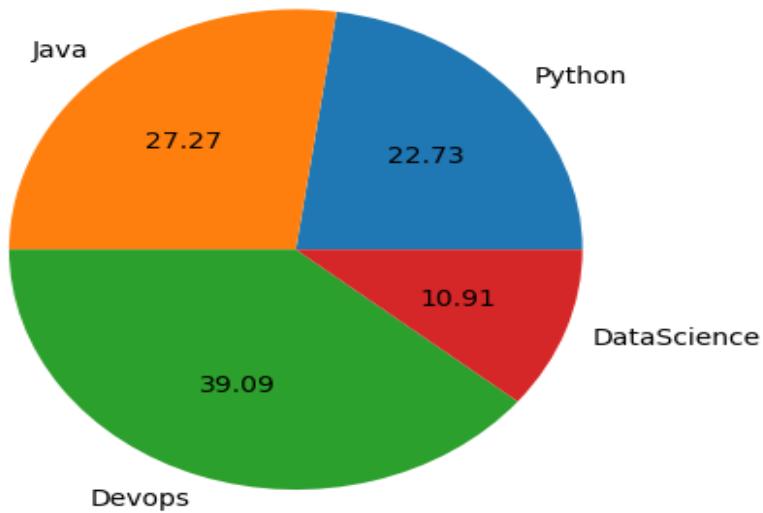
**After decimal point, if we want to consider two digits**

```
autopct = '%.2f'
plt.pie(marks,labels=mylabels,autopct = '%.2f')
```

In [142]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
```

```
mylabels = ['Python','Java','Devops','DataScience']
plt.pie(marks,labels=mylabels,autopct = '%.2f')
plt.show()
```



### To add % symbol also

```
autopct = '%.2f%'  
ValueError: incomplete format
```

In [143]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
plt.pie(marks,labels=mylabels,autopct = '%.2f%')
plt.show()
```

**ValueError: incomplete format**

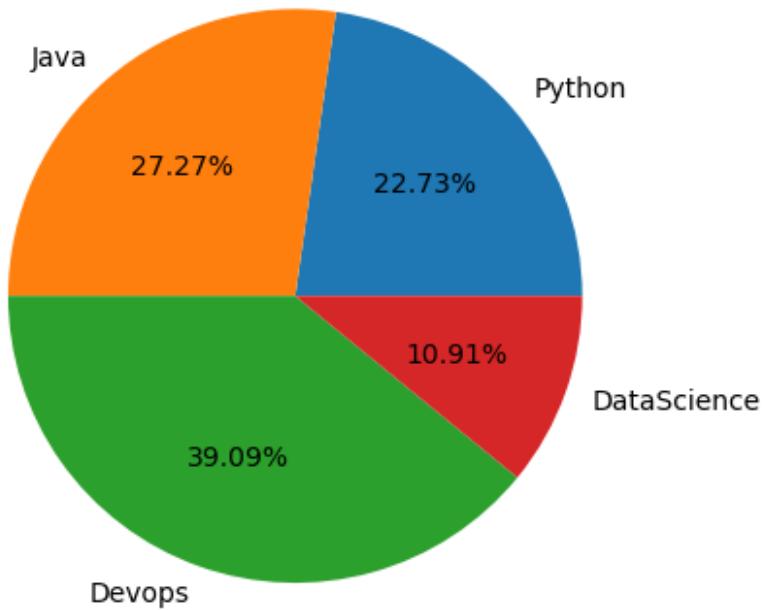
---

**To add % symbol also ==> use wild character**

```
autopct = '%.2f%%'  
plt.pie(marks,labels=mylabels,autopct = '%.2f%%')
```

In [144]:

```
import matplotlib.pyplot as plt  
import numpy as np  
marks = np.array([25,30,43,12])  
mylabels = ['Python','Java','Devops','DataScience']  
plt.pie(marks,labels=mylabels,autopct = '%.2f%%')  
plt.show()
```



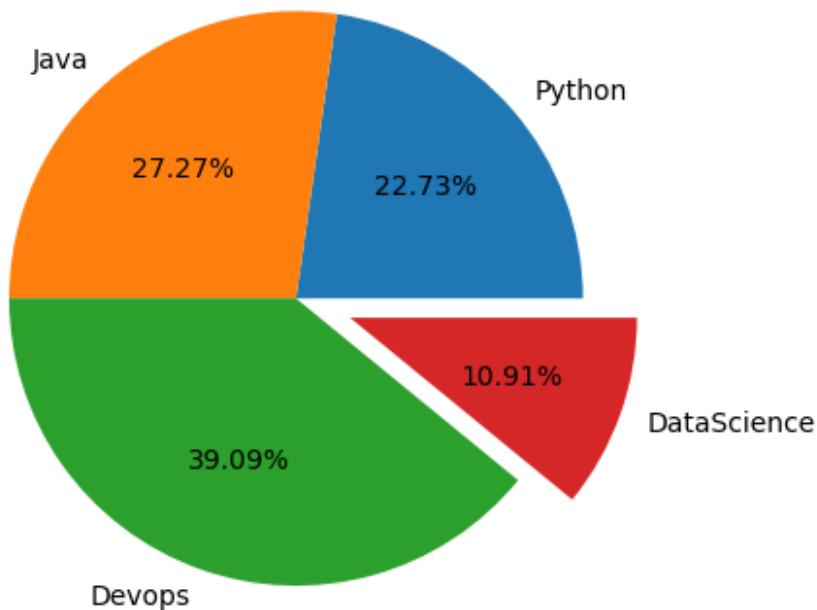
---

## explode

If we want to explode/highlight a particular category then we should use explode argument.

In [145]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
plt.pie(marks,labels=mylabels,autopct = '%.2f%%',explode=myexplode)
plt.show()
```



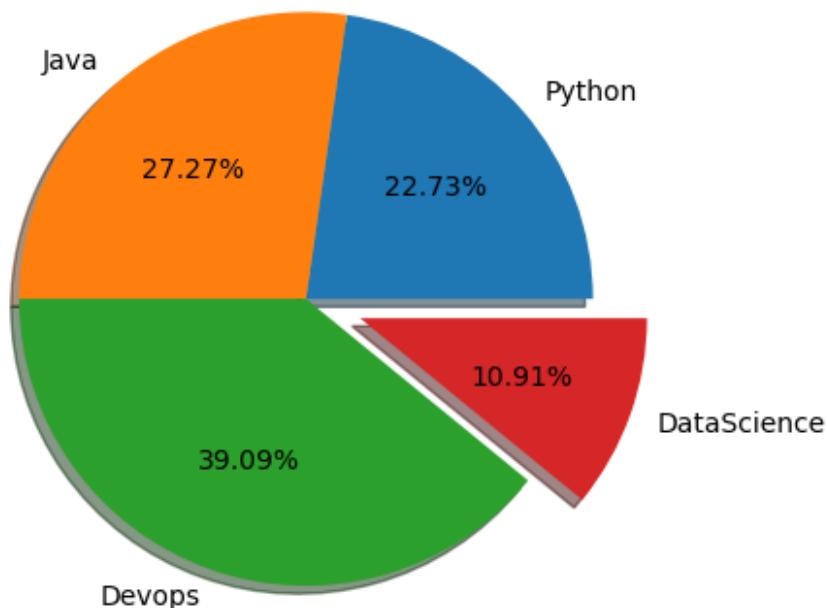
## shadow

We can add shadow effect to the pie chart by using **shadow parameter**.

The **default value** is **False**. By setting to **True**, we can see **shadow effect**.

In [146]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
plt.pie(marks,
         labels=mylabels,
         autopct = '%.2f%%',
         explode=myexplode,
         shadow=True)
plt.show()
```



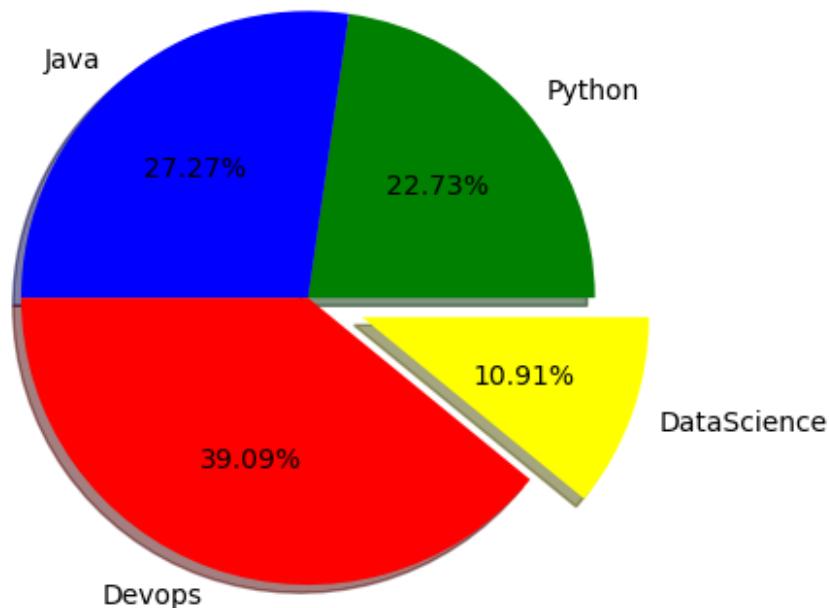
## colors

We can specify our own colors for the wedges. It can be **list or array**.

We can specify **color name** or **short color** code or even **hexa code** also.

In [147]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
         labels=mylabels,
         autopct = '%.2f%%',
         explode=myexplode,
         shadow=True,
         colors=mycolors)
plt.show()
```

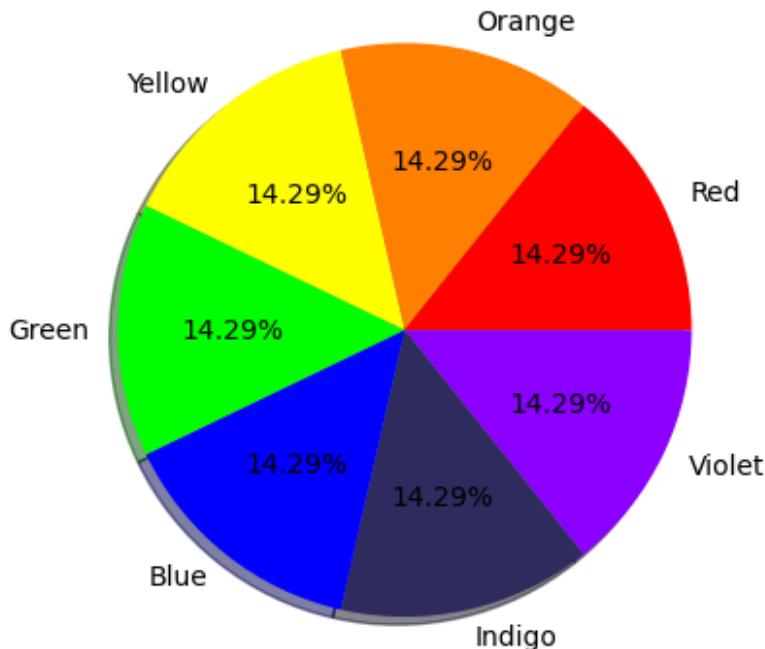


---

## VIBGYOR colors

In [148]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([1,1,1,1,1,1])
my_labels = ['Violet','Indigo','Blue','Green','Yellow','Orange','Red']
# my_colors = ['violet','indigo','blue','green','yellow','orange','red']
my_colors =
['#8B00FF','#2E2B5F','#0000FF','#00FF00','#FFFF00','#FF7FFF','#FF0000']
plt.pie(marks,
    labels=my_labels,
    autopct = '%.2f%%',
    shadow=True,
    colors = my_colors,
    counterclock=False)
plt.show()
```



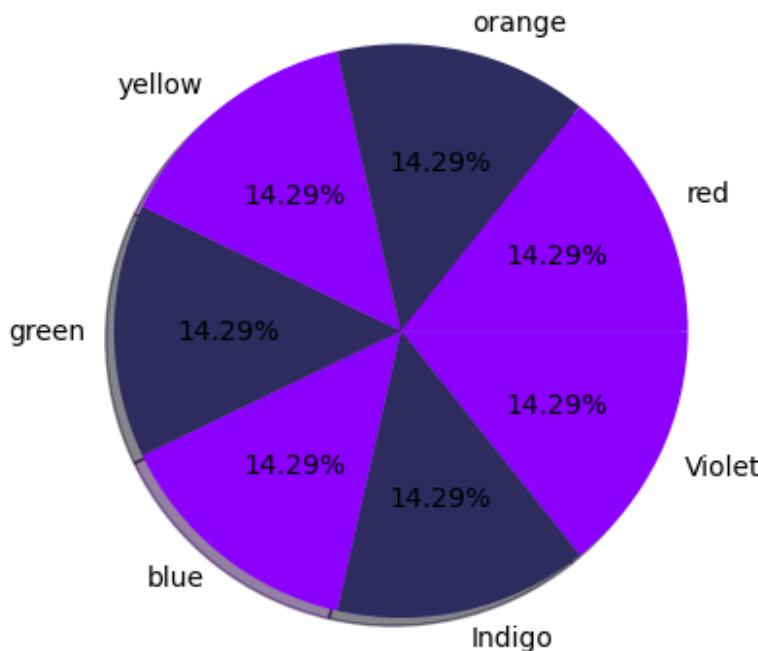
---

## Note

If the number of colors is less than the number of wedges, then the colors will be reused.

In [149]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([1,1,1,1,1,1,1])
my_labels = ['Violet','Indigo','blue','green','yellow','orange','red']
my_colors = ['#8B00FF','#2E2B5F']
plt.pie(marks,
        labels=my_labels,
        autopct = '%.2f%%',
        shadow=True,
        colors = my_colors,
        counterclock=False)
plt.show()
```



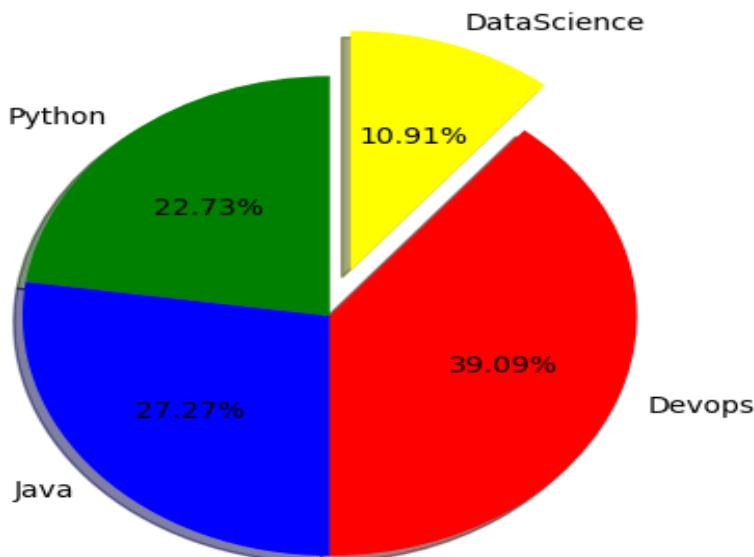
## **startangle**

**startangle** represents from where **first wedge has to start**.

**Bydefault** it starts from **zero from x-axis** and move in **counter clockwise direction**.

In [150]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
        labels=mylabels,
        autopct = '%.2f%%',
        explode=myexplode,
        shadow=True,
        colors=mycolors,
        startangle=90)
plt.show()
```



## counterclock

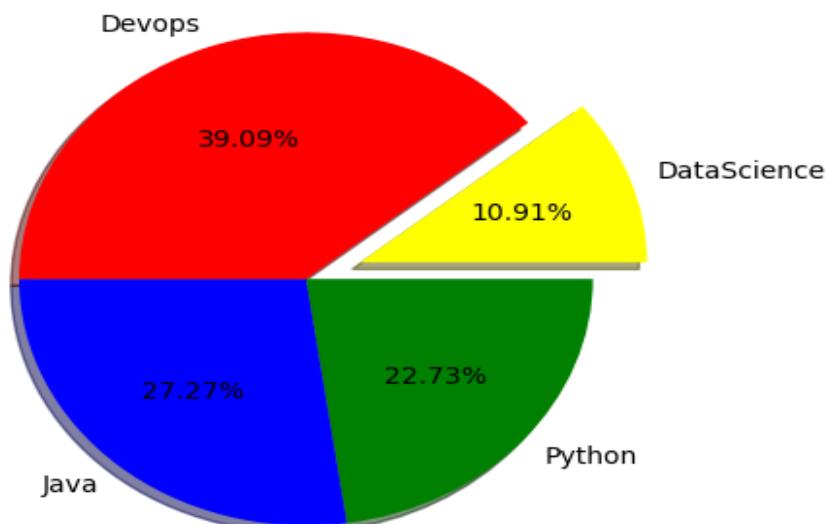
By default the wedges will be considered in counter clockwise direction.

If we want clock wise direction we have to use **counterclock** argument.

**The defualt value is True**

In [151]:

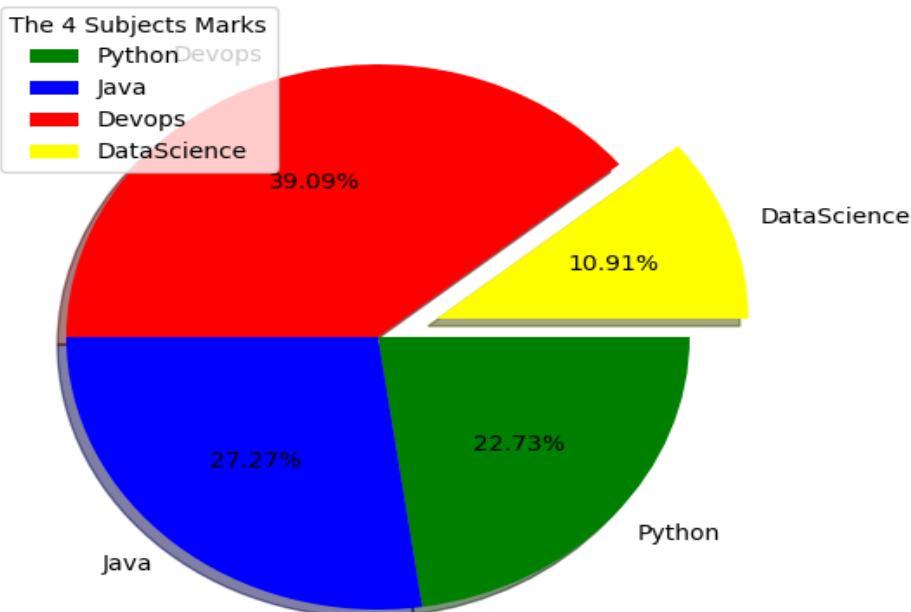
```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
        labels=mylabels,
        autopct = '%.2f%%',
        explode=myexplode,
        shadow=True,
        colors=mycolors,
        counterclock=False)
plt.show()
```



## Adding legend

In [152]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
         labels=mylabels,
         autopct = '%.2f%%',
         explode=myexplode,
         shadow=True,
         colors=mycolors,
         counterclock=False)
plt.legend(title='The 4 Subjects Marks')
plt.tight_layout()
plt.show()
```



---

## wedgeprops

The **wedges** of pie chart can be **customized** using the **wedgeprops parameter**.

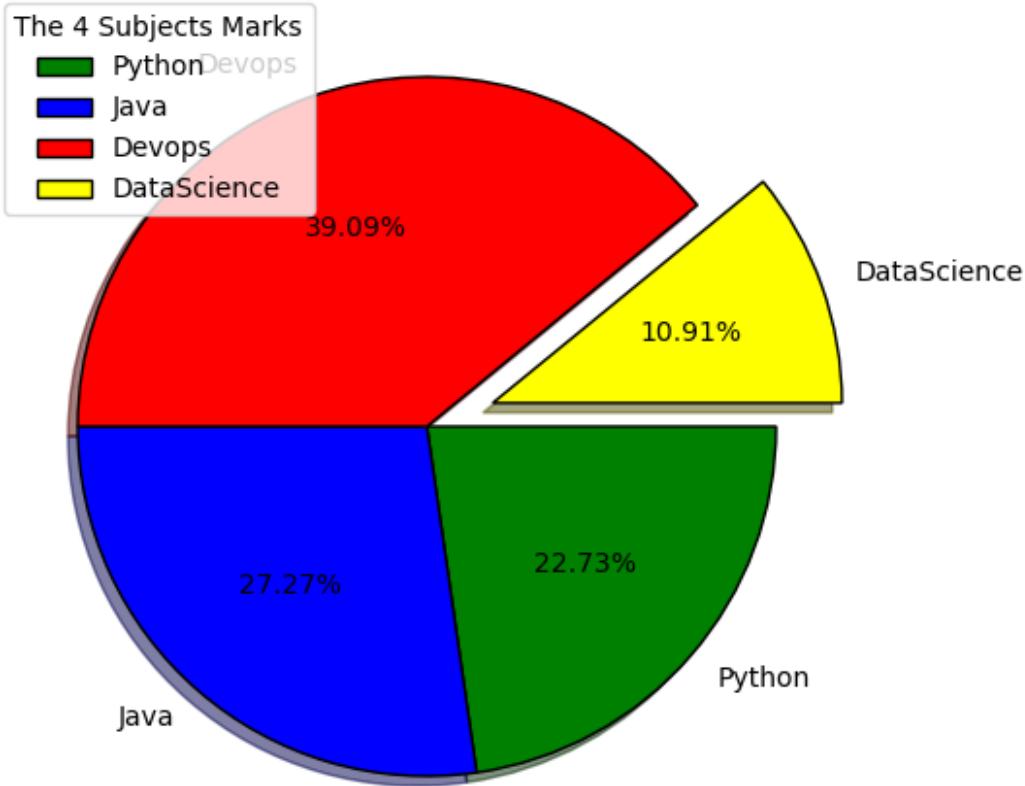
It is a **dictionary of key-value pairs**

The **keys** can be **edgecolor, linestyle, linewidth etc**

## edgecolor

In [153]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
        labels=mylabels,
        autopct = '%.2f%%',
        explode=myexplode,
        shadow=True,
        colors=mycolors,
        counterclock=False,
        wedgeprops={'edgecolor':'k'})
plt.legend(title='The 4 Subjects Marks')
plt.tight_layout()
plt.show()
```

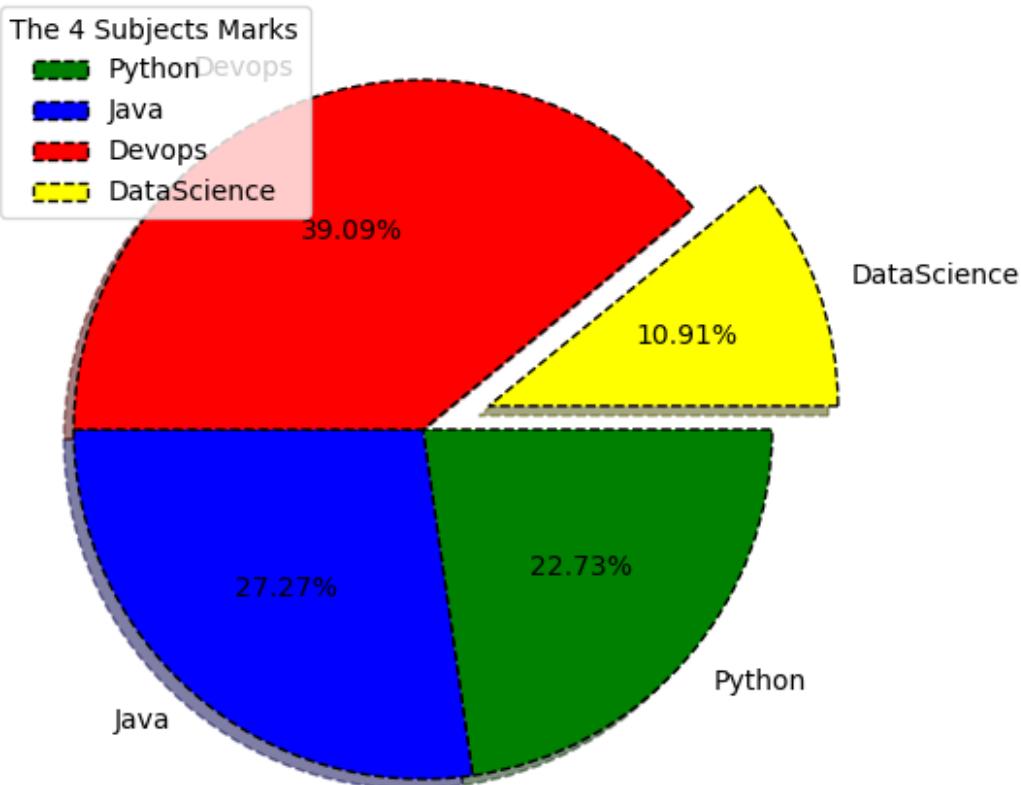


## edgecolor and linestyle

In [154]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
        labels=mylabels,
        autopct = '%.2f%%',
```

```
explode=myexplode,
shadow=True,
colors=mycolors,
counterclock=False,
wedgeprops={'edgecolor':'k','linestyle': '--'})
plt.legend(title='The 4 Subjects Marks')
plt.tight_layout()
plt.show()
```

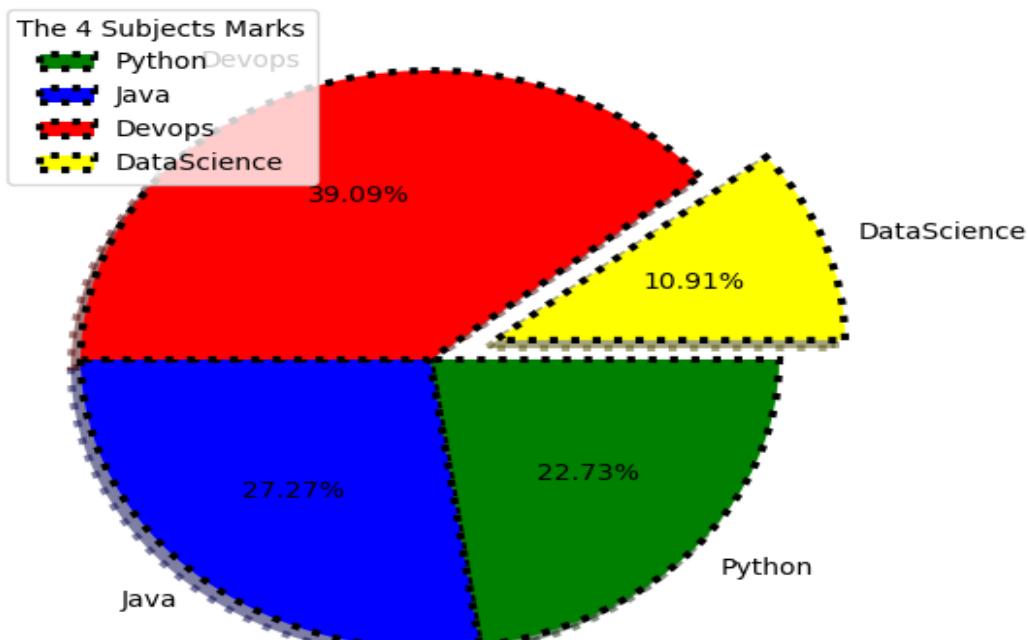


---

## edgecolor, linestyle and linewidth

In [155]:

```
import matplotlib.pyplot as plt
import numpy as np
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
myexplode = [0.0,0.0,0.0,0.2]
mycolors = ['g','b','r','yellow']
plt.pie(marks,
        labels=mylabels, autopct = '%.2f%%',
        explode=myexplode, shadow=True,
        colors=mycolors, counterclock=False,
        wedgeprops={'edgecolor':'k','linestyle':'-', 'linewidth':3})
plt.legend(title='The 4 Subjects Marks')
plt.tight_layout()
plt.show()
```



---

## Chapter-13 Histogram

### Histograms

- ✓ **Frequency Distribution** ==> It is nothing but number of observations in the given interval.
- ✓ To represent such type of frequency distributions, we should go for histogram.

### Eg-1

#### 300 students (marks : 0 to 100)

- ✓ 23 students got marks in the range: 0 to 34
- ✓ 120 students got marks in the range: 35 to 49
- ✓ 47 students got marks in the range: 50 to 59
- ✓ 80 students got marks in the range: 60 to 79
- ✓ 30 students got marks in the range: 80 to 100

We distributed 300 values in 5 intervals.

### Eg-2

- ✓ We are conducting an experiment, where we are trying to roll 3 dices one lakh times.
- ✓ Every time the sum of outcome of 3 dices will be appended to the list.
- ✓ To analyze these 1 lakh values from the list, it is very difficult.
- ✓ If we convert this into visualization form then it is very easy.
- ✓ For every dice throw, the minimum value: 1 and maximum value: 6
- ✓ The minimum outcome is 3 and maximum outcome is 18.
- ✓ Total possible outcomes: 3 to 18 means 16 possible outcomes are there.
- ✓ total outcome: 18,15,12,13,16,13,5,9,4,7,6
- ✓ 1 lakh values should be distributed into 16 intervals
- ✓ most likely possible values are lies **9 to 12**

---

In [156]:

```
import numpy as np
import time
while True:
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
    print(f'{d1} + {d2} + {d3} = {d1+d2+d3}')
    time.sleep(3)
```

```
5 + 4 + 5 = 14
2 + 5 + 4 = 11
1 + 6 + 2 = 9
1 + 5 + 3 = 9
6 + 2 + 4 = 12
2 + 1 + 5 = 8
4 + 2 + 1 = 7
5 + 6 + 3 = 14
1 + 4 + 3 = 8
5 + 5 + 2 = 12
5 + 6 + 5 = 16
6 + 4 + 3 = 13
1 + 6 + 5 = 12
5 + 1 + 1 = 7
6 + 1 + 2 = 9
4 + 1 + 2 = 7
3 + 5 + 5 = 13
5 + 4 + 2 = 11
1 + 4 + 6 = 11
4 + 6 + 3 = 13
```

---

```
KeyboardInterrupt                               Traceback (most recent call last)
<ipython-input-156-e95bde62813f> in <module>
      6     d3 = np.random.randint(1,7)
      7     print(f'{d1} + {d2} + {d3} = {d1+d2+d3}')
----> 8     time.sleep(3)
```

KeyboardInterrupt:

---

- 
- ✓ **Histograms** are very helpful to **analyze large data sets**.
  - ✓ To plot histograms, we have to divide total input values into equal sized groups or **bins**.
  - ✓ A bar is drawn for each bin. The height of each bar is proportional to the number of values related to that bar(bin or interval)
  - ✓ By using **hist() function**, we can create histogram.

```
In [157]:
```

```
import matplotlib.pyplot as plt  
help(plt.hist)
```

Help on function hist in module matplotlib.pyplot:

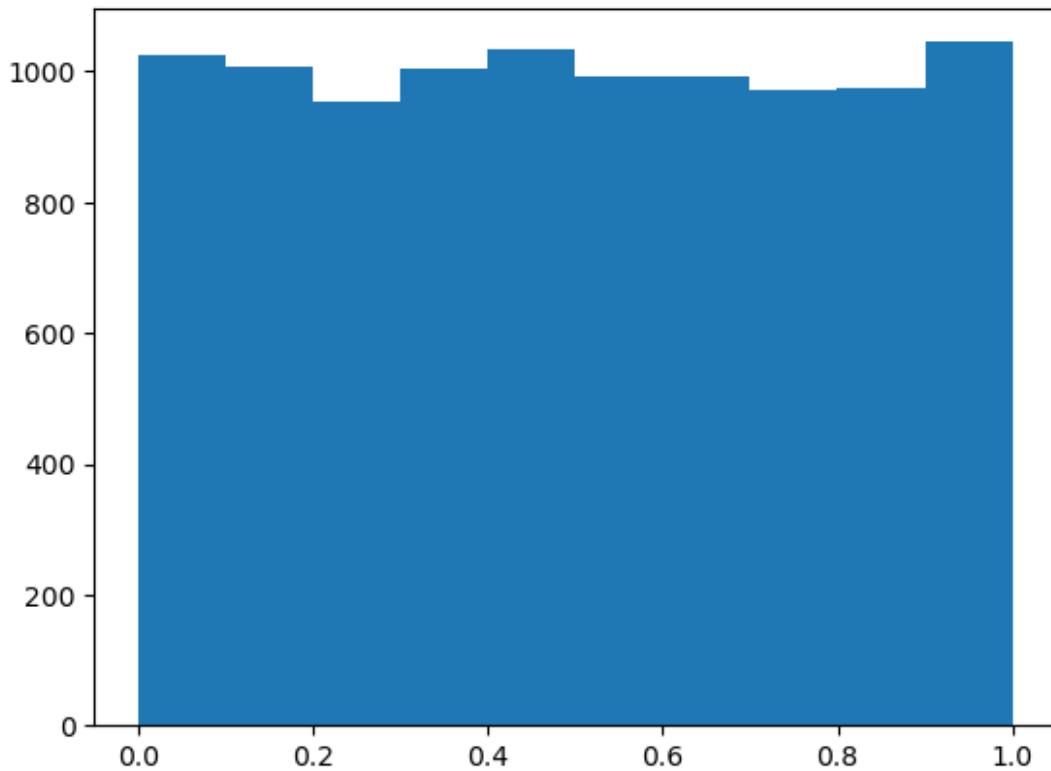
```
hist(x, bins=None, range=None, density=False, weights=None,  
cumulative=False, bottom=None, histtype='bar', align='mid',  
orientation='vertical', rwidth=None, log=False, color=None, label=None,  
stacked=False, *, data=None, **kwargs)
```

Plot a histogram.

**eg-1: To create histogram with 10000 samples from uniform distribution in the interval [0,1]**

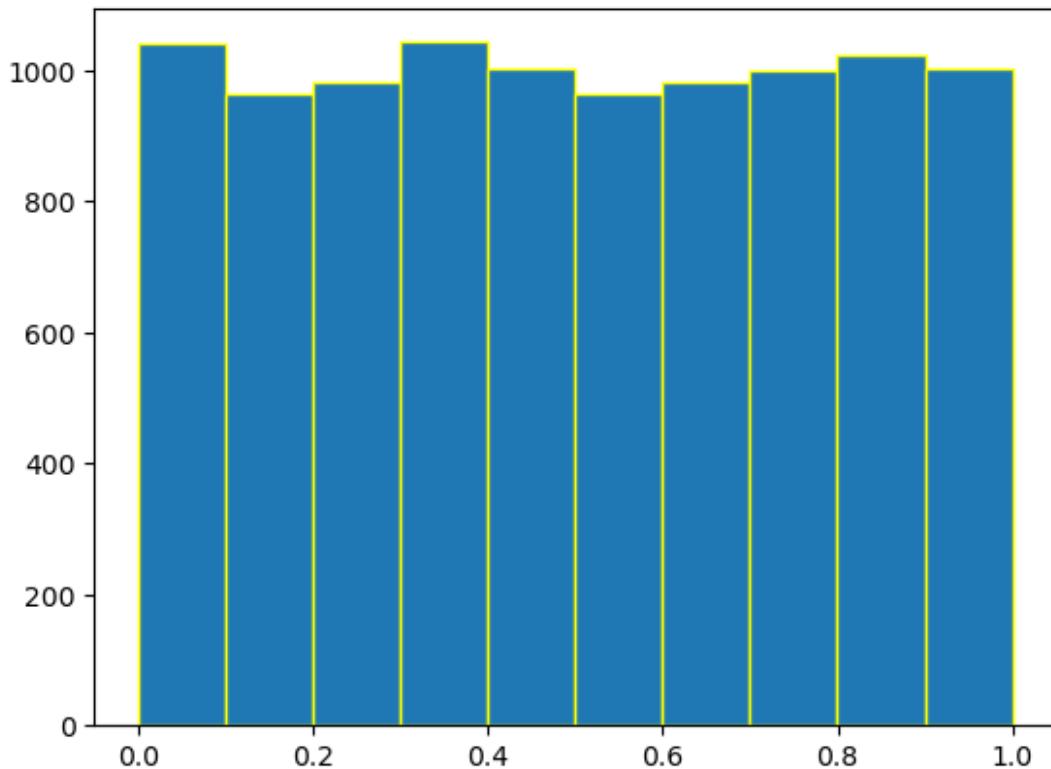
```
In [158]:
```

```
import matplotlib.pyplot as plt  
import numpy as np  
x = np.random.rand(10000)  
plt.hist(x)  
plt.show()
```



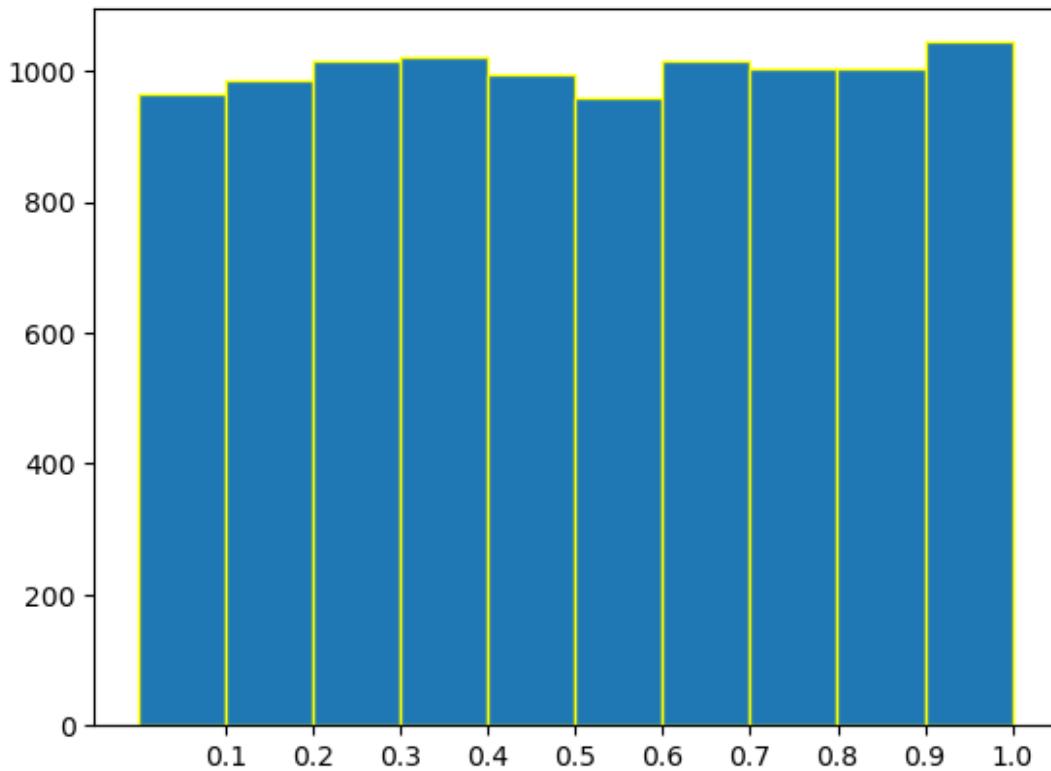
In [159]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.rand(10000)
plt.hist(x, ec='yellow')
plt.show()
```



In [160]:

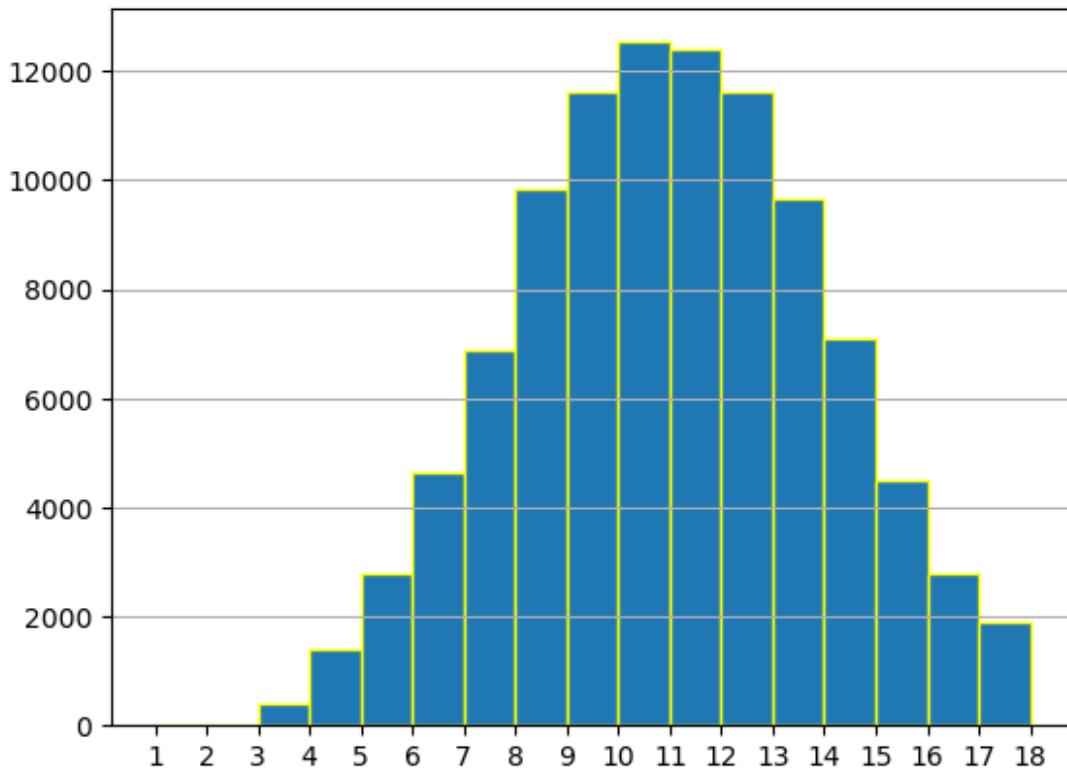
```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.rand(10000)
xticks_1=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0]
plt.hist(x,bins=10,ec='yellow')
plt.xticks(xticks_1)
plt.show()
```



### eg-2: Rolling 3 dice experiment

In [161]:

```
import matplotlib.pyplot as plt
import numpy as np
l = []
x = np.arange(1,19) #[1,2,3,...,18]
for i in range(100000):
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
    l.append(d1+d2+d3)
plt.hist(l,bins=x,ec='yellow')
plt.xticks(x)
plt.grid(axis='y')
plt.show()
```



## Histograms

- ✓ Frequency Distributions
- ✓ Large data sets
- ✓ Each interval is nothing but bin
- ✓ `hist()` function is used to plot histograms
- ✓ The default number of bins: 10

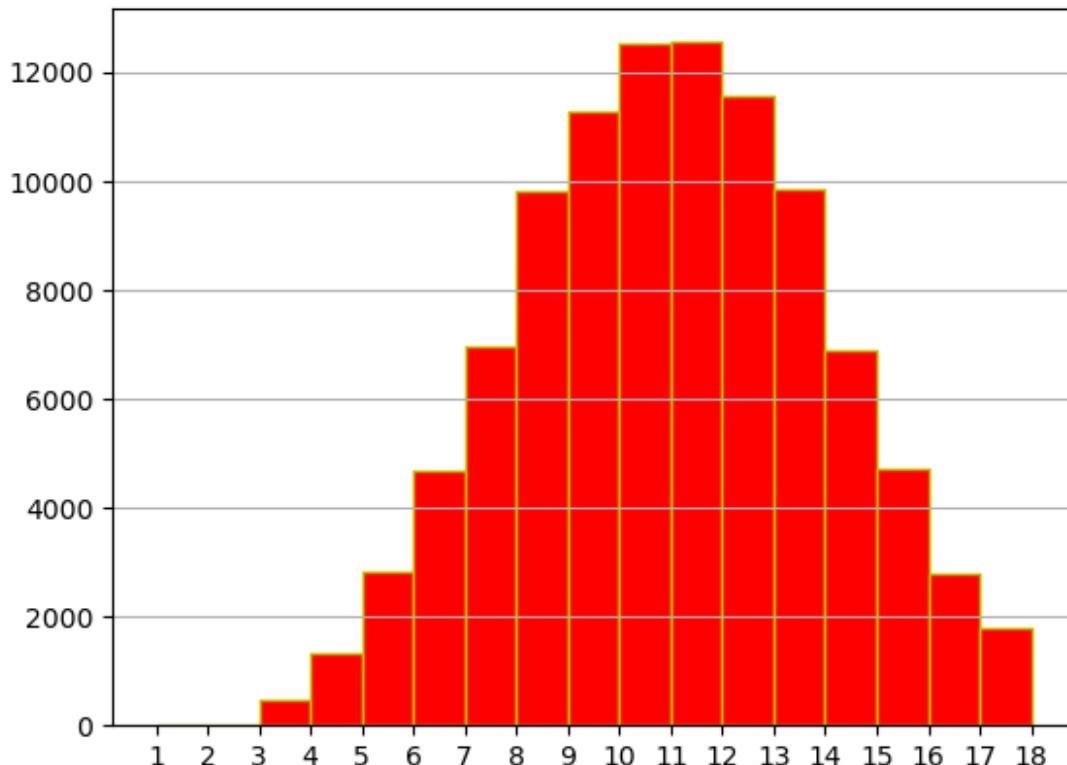
## How to change color of bars in histogram

We can specify our own customized color for the bars by using **color argument**.

---

In [162]:

```
import matplotlib.pyplot as plt
import numpy as np
l = []
x = np.arange(1,19) #[1,2,3,...,18]
for i in range(100000):
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
    l.append(d1+d2+d3)
plt.hist(l,bins=x,color='r',ec='y')
plt.xticks(x)
plt.grid(axis='y')
plt.show()
```



---

## How to change color of the each bars in histogram

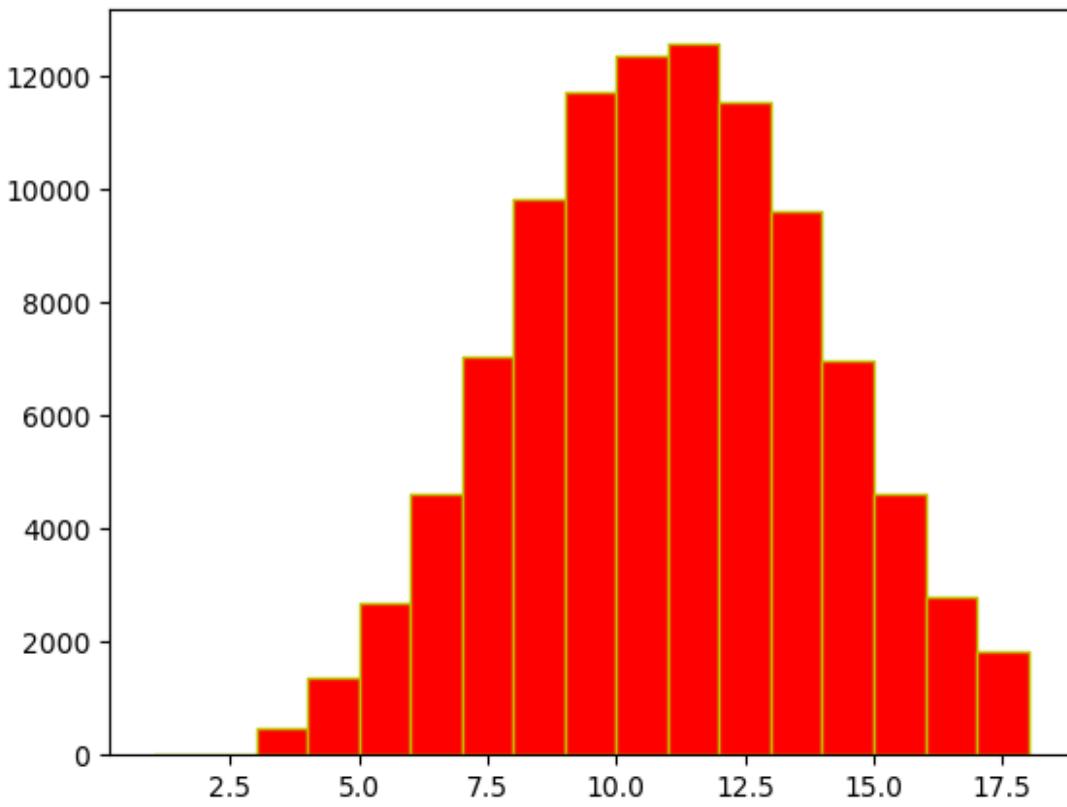
The **hist()** function returns the tuple of the following 3 values.

- a) **n** → The number of values present inside each bin
- b) **bins** → The edges of bins
- c) **patches** → list of objects. BarContainer of individual artists used to create the histogram.

In [163]:

```
import matplotlib.pyplot as plt
import numpy as np
l = []
x = np.arange(1,19) #[1,2,3,...,18]
for i in range(100000):
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
    l.append(d1+d2+d3)
x = plt.hist(l,bins=x,color='r',ec='y')
print(f'type of x ==> {type(x)}')
print(f'size of the x ==> {len(x)}')
print(x)
```

type of x ==> <class 'tuple'>  
size of the x ==> 3  
(array([ 0., 0., 462., 1365., 2680., 4621., 7053., 9814.,  
11706., 12370., 12571., 11530., 9620., 6968., 4600., 2804.,  
1836.]), array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17,  
18]), <BarContainer object of 17 artists>)



### Note

**n** ➔ array([ 0., 0., 462., 1365., 2680., 4621., 7053., 9814., 11706., 12370., 12571., 11530., 9620., 6968., 4600., 2804., 1836.])

**bins** ➔ array([ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18]),

**patches** ➔ <BarContainer object of 17 artists>

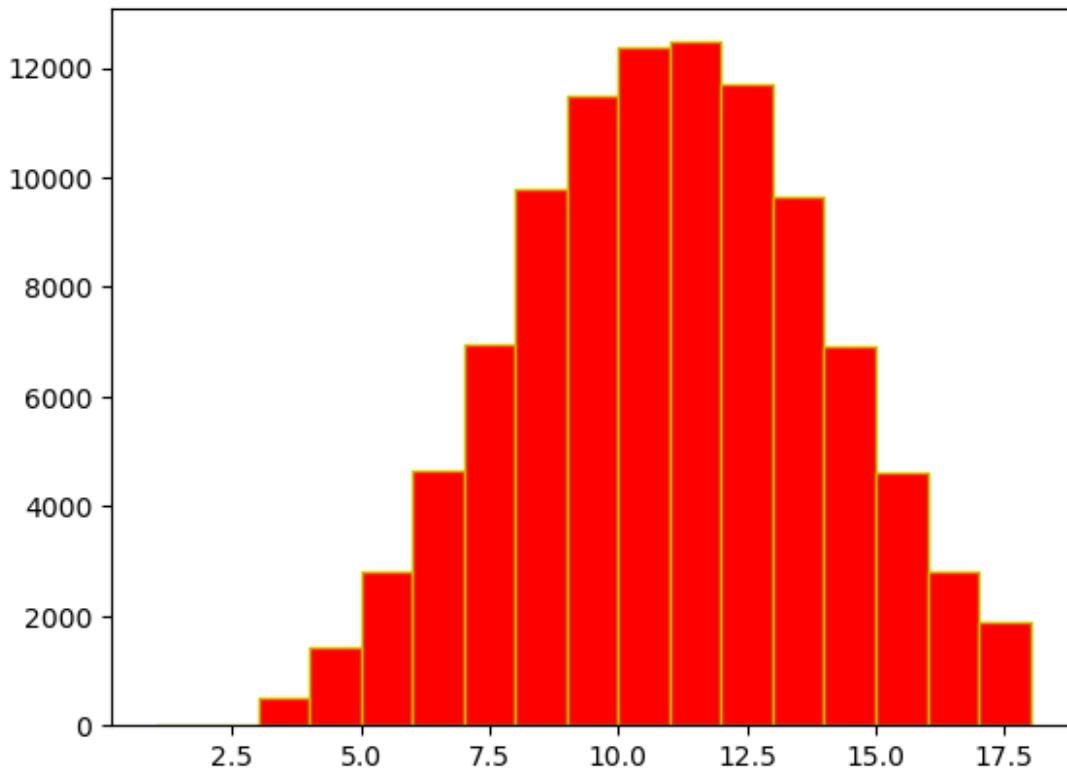
---

In [164]:

```
import matplotlib.pyplot as plt
import numpy as np
l = []
x = np.arange(1,19) #[1,2,3,...,18]
for i in range(100000):
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
    l.append(d1+d2+d3)
n,bins,patches = plt.hist(l,bins=x,color='r',ec='y')
print('The number of values in each bin : ',n)
print('The edges of bins : ',bins)
print('The Patches-Container of artists : ',patches)
```

The number of values in each bin : [ 0. 0. 515. 1411. 2799. 4645.  
6936. 9784. 11488. 12383. 12470. 11705. 9658. 6919. 4618. 2798.  
1871.]

The edges of bins : [ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18]  
The Patches-Container of artists : <BarContainer object of 17 artists>



To change color of first bar

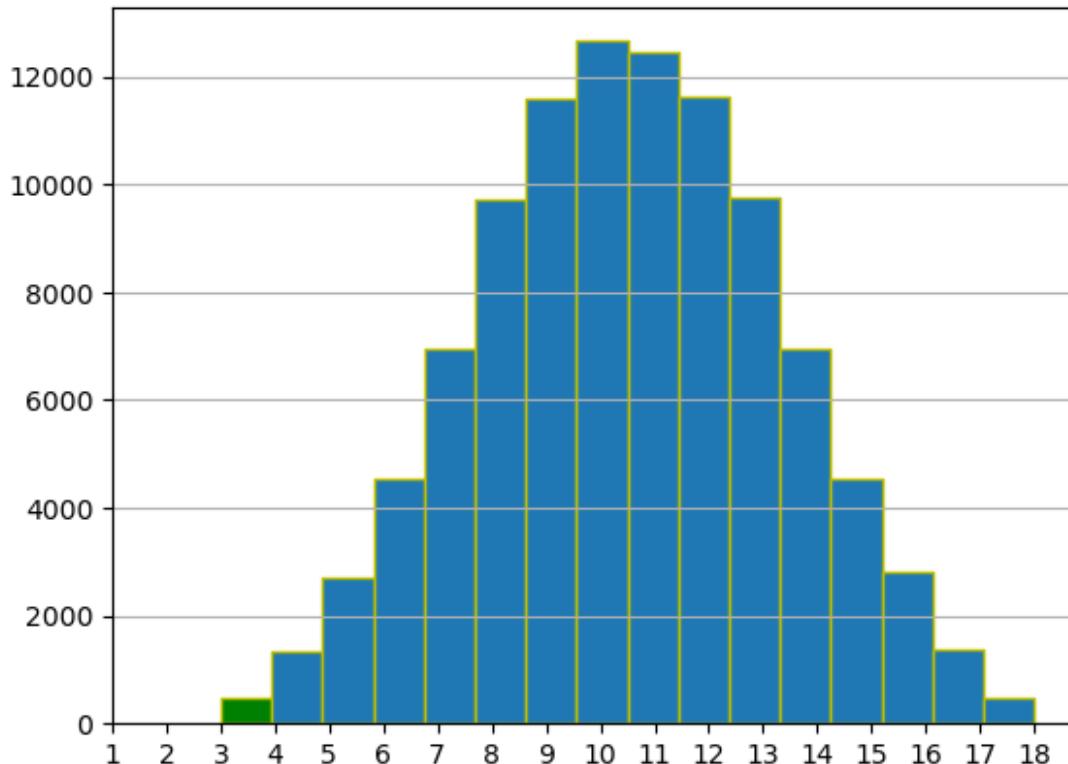
We can use any of the following statements to change the color of the bar

```
patches[0].set_facecolor('green')  
patches[0].set_fc('green')
```

In [165]:

```
import matplotlib.pyplot as plt  
import numpy as np  
l = []  
x = np.arange(1,19) #[1,2,3,...,18]  
for i in range(100000):  
    d1 = np.random.randint(1,7)  
    d2 = np.random.randint(1,7)  
    d3 = np.random.randint(1,7)  
    l.append(d1+d2+d3)
```

```
n,bins,patches = plt.hist(l,bins=16,ec='y')
plt.xticks(x)
plt.grid(axis='y')
#patches[0].set_facecolor('green')
patches[0].set_fc('green')
```



### To change color of each bar

In [166]:

```
import matplotlib.pyplot as plt
import numpy as np
l = []
x = np.arange(1,19) #[1,2,3,...,18]
for i in range(100000):
    d1 = np.random.randint(1,7)
    d2 = np.random.randint(1,7)
    d3 = np.random.randint(1,7)
```

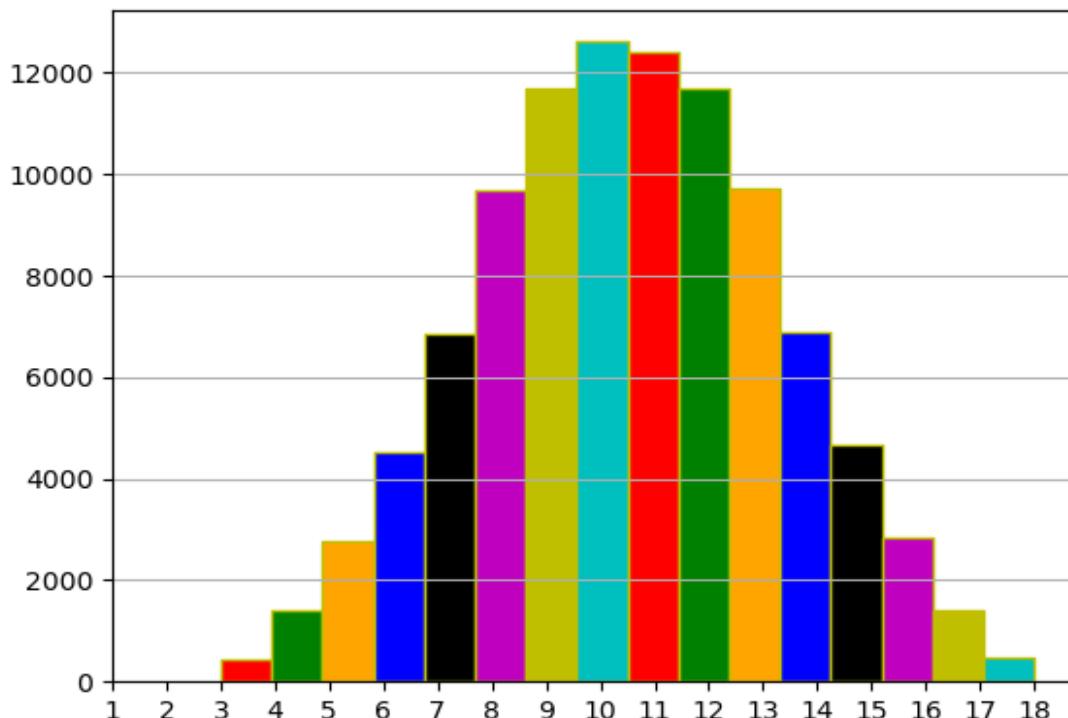
```

l.append(d1+d2+d3)
n,bins,patches = plt.hist(l,bins=16,ec='y')
print(f'Values of histogram bins ==> {n.size}')
print(f'Edges of the bins ==> {bins.size}')
plt.xticks(x)
plt.grid(axis='y')
colors = ['r','g','orange','b','k','m','y','c','r','g','orange','b','k','m','y','c']
for i in range(n.size):
    patches[i].set_facecolor(colors[i])
plt.show()

```

Values of histogram bins ==> 16

Edges of the bins ==> 17



### How to get the real data sets

<https://www.kaggle.com/>

TV Shows and Movies Listed on Netflix: ➔

<https://www.kaggle.com/shivamb/netflix-shows>

---

## How many movies and TV shows are released on Netflix year-wise

In [167]:

```
# How many movies and TV shows are released on Netflix year-wise
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    release_year_list.append(int(row[7]))
print(release_year_list)
```

UnicodeDecodeError: 'charmap' codec can't decode byte 0x81 in position 6032:  
character maps to <undefined>

### Note

While reading the csv data from the data we must use the **encoding** parameter  
as **utf-8**

In [168]:

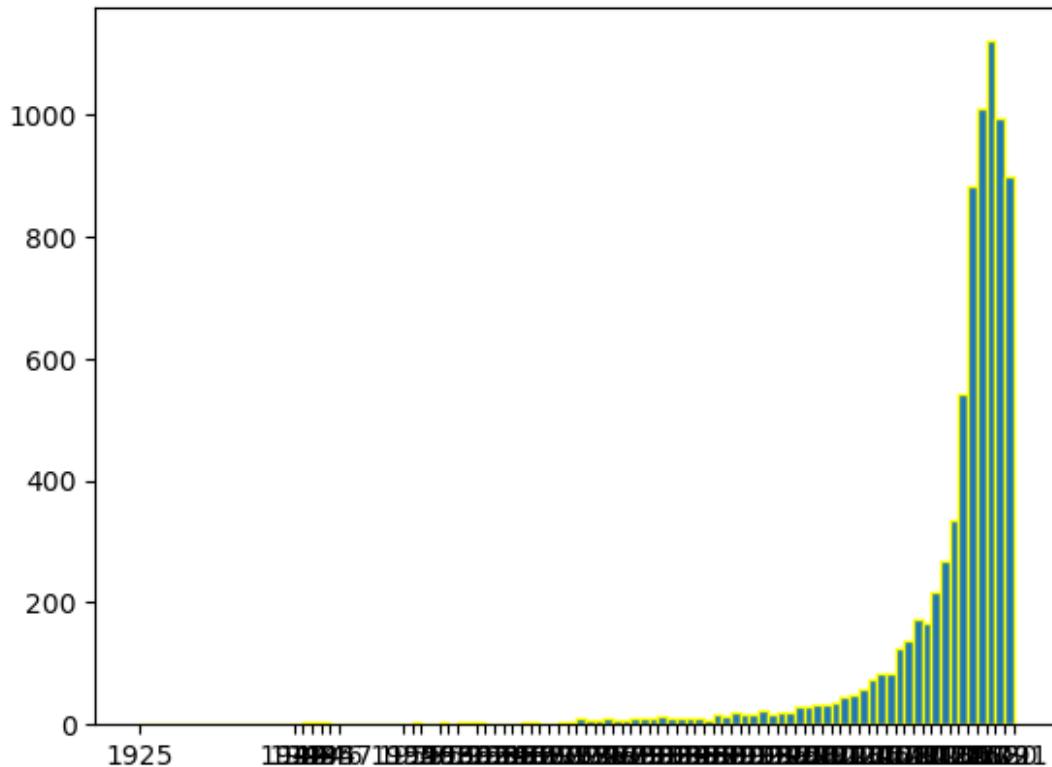
```
# How many movies and TV shows are released on Netflix year-wise
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    release_year_list.append(int(row[7])) # row[7] denotes the release_year in the
data set
print(release_year_list)
```

[2020, 2016, .....2019, 2019, 2019]

---

In [169]:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    release_year_list.append(int(row[7]))
unique_values = np.unique(release_year_list) # to get the unique years
plt.hist(release_year_list,bins = unique_values,ec='yellow')
plt.xticks(unique_values)
plt.show()
```

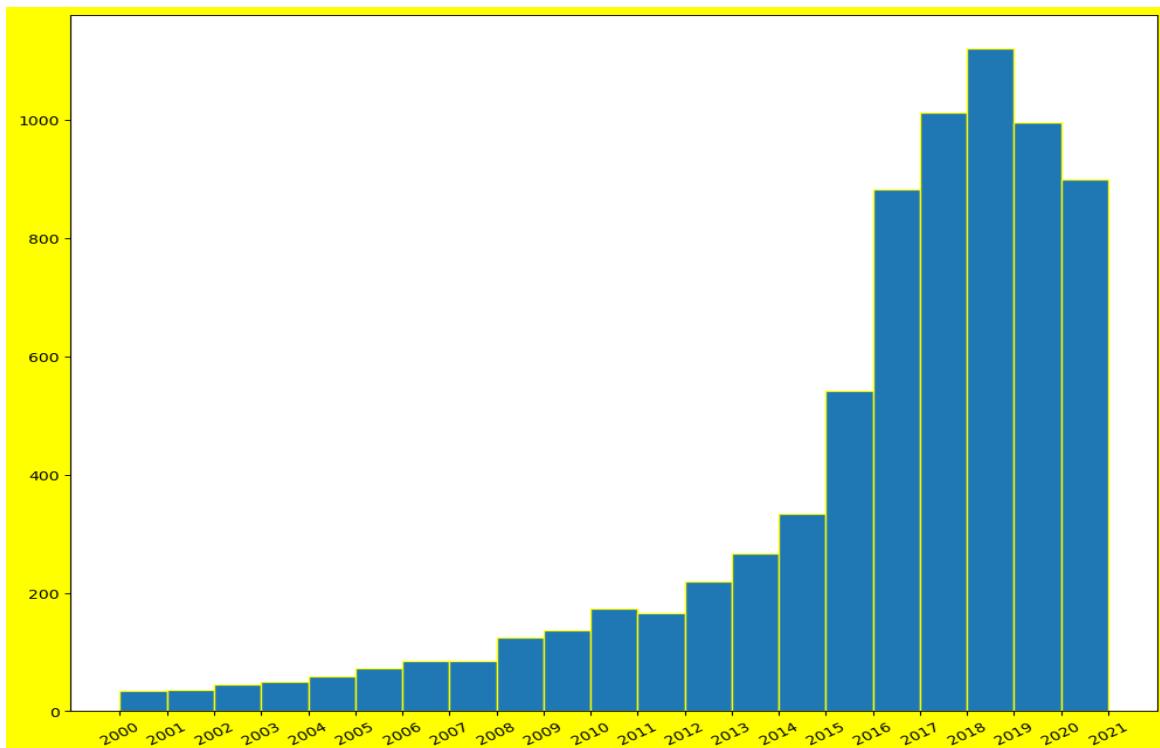


---

## Number of movies and TV shows are released on Netflix from 2000 to 2021

In [170]:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    release_year_list.append(int(row[7]))
required_years = np.arange(2000,2022)
plt.figure(num=1,figsize=(12,10),facecolor='yellow')
plt.hist(release_year_list,bins = required_years,ec='yellow')
plt.xticks(required_years,rotation=30)
plt.show()
```



---

**Number of movies and TV shows are released on Netflix from 2000 to 2021 only from 'India'**

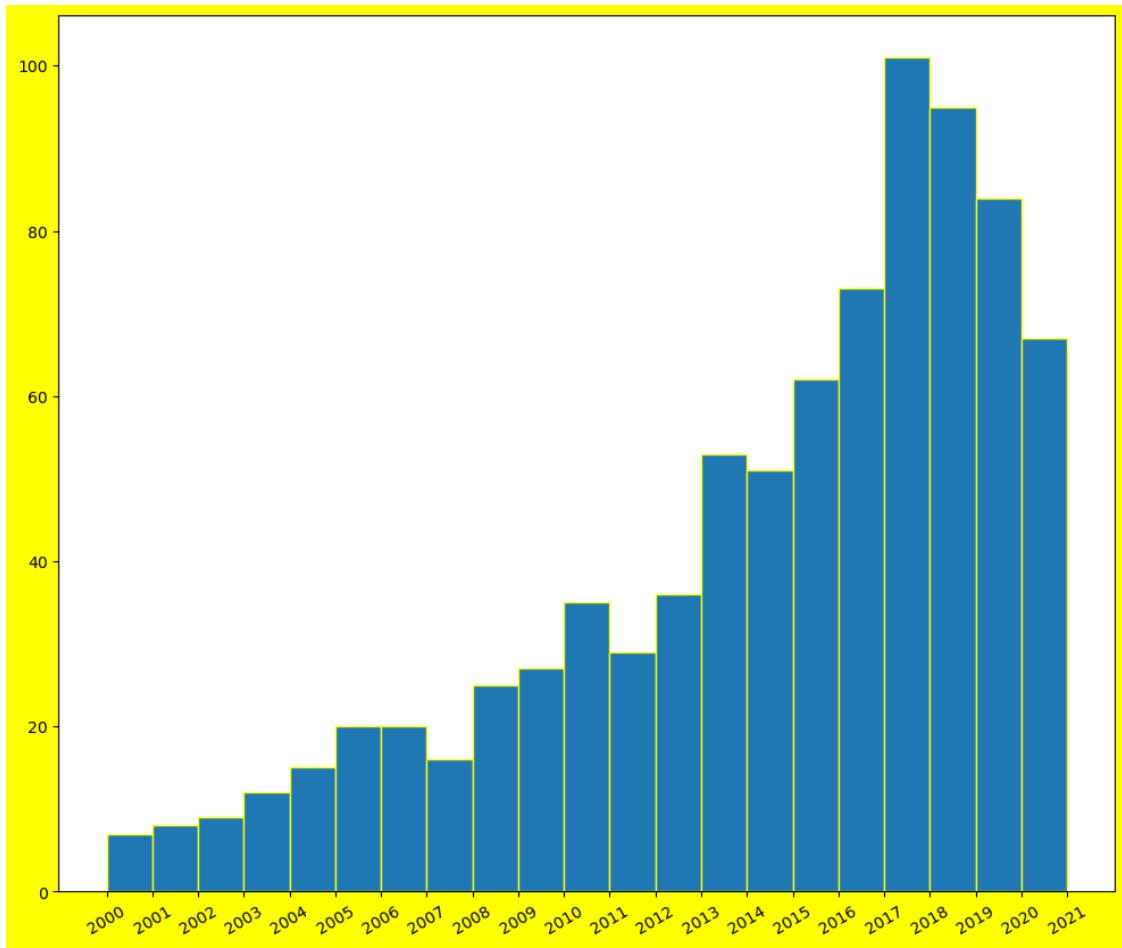
**row[5]** contains **country** information

### **Columns in the dataset**

- ✓ show\_id
- ✓ type
- ✓ title
- ✓ director
- ✓ cast
- ✓ country
- ✓ date\_added
- ✓ release\_year
- ✓ rating
- ✓ duration
- ✓ listed\_in
- ✓ description

In [171]:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    if row[5] == 'India':
        release_year_list.append(int(row[7]))
required_years = np.arange(2000,2022)
plt.figure(num=1,figsize=(12,10),facecolor='yellow')
plt.hist(release_year_list,bins = required_years,ec='yellow')
plt.xticks(required_years,rotation=30)
plt.show()
```



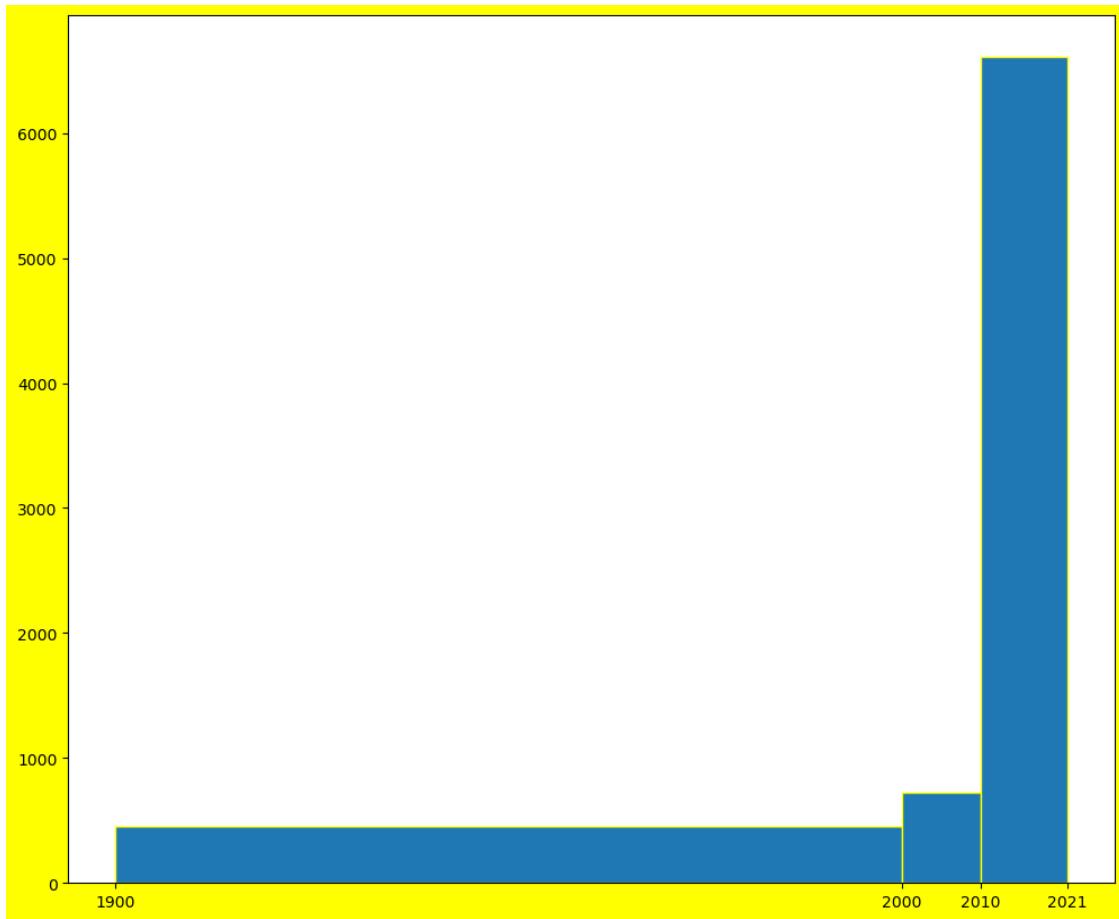
**Number of movies and TV shows released in past decade**

In [172]:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
release_year_list=[]
f = open('netflix_titles.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
header = next(r) #read header and ignore
for row in r:
    release_year_list.append(int(row[7]))
required_bins = [1900,2000,2010,2021]
```

---

```
plt.figure(num=1,figsize=(12,10),facecolor='yellow')
plt.hist(release_year_list,bins = required_bins,ec='yellow')
plt.xticks(required_bins)
plt.show()
```



---

## Chapter-14 Scatter Plots

### Scatter plot

- ✓ Scatter plots are glorious. Of all the major chart types the most powerful and most commonly used charts are scatter plots.
- ✓ These are very similar to line plots. The terminology like x-axis,y-axis,data points etc are exactly same.
- ✓ Scatter plots are very helpful to represent the relation between two variables
- ✓ **scatter()** function is used to draw **scatter plot**

### Eg

Location and covid cases

In [173]:

```
import matplotlib.pyplot as plt
help(plt.scatter)
```

Help on function scatter in module matplotlib.pyplot:

```
scatter(x, y, s=None, c=None, marker=None, cmap=None, norm=None,
vmin=None, vmax=None, alpha=None, linewidths=None, verts=<deprecated
parameter>, edgecolors=None, *, plotnonfinite=False, data=None, **kwargs)
A scatter plot of *y* vs. *x* with varying marker size and/or color.
```

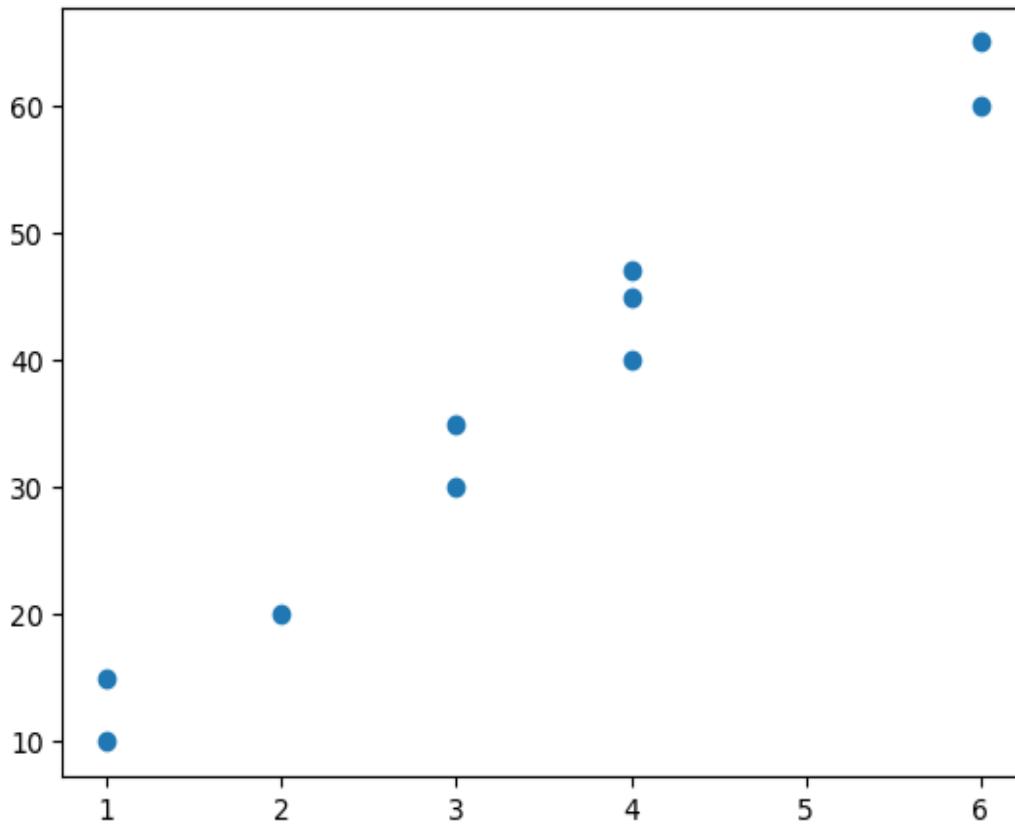
- ✓ **s** → size of the each marker
- ✓ **c** → color of each marker
- ✓ **marker** → marker style
- ✓ **cmap** → colormap

---

In [174]:

```
## Basic understanding of scatter plot

import matplotlib.pyplot as plt
import numpy as np
x = [1,4,3,2,6,1,3,4,4,6]
y = [10,40,30,20,60,15,35,45,47,65]
plt.scatter(x,y) #data points:(1,10),(4,40),(3,30),(2,20),(6,60),(1,15),(3,35)...
#xlabel,label,title,grid,xticks,yticks,xlim,ylim,legend...
plt.show()
```

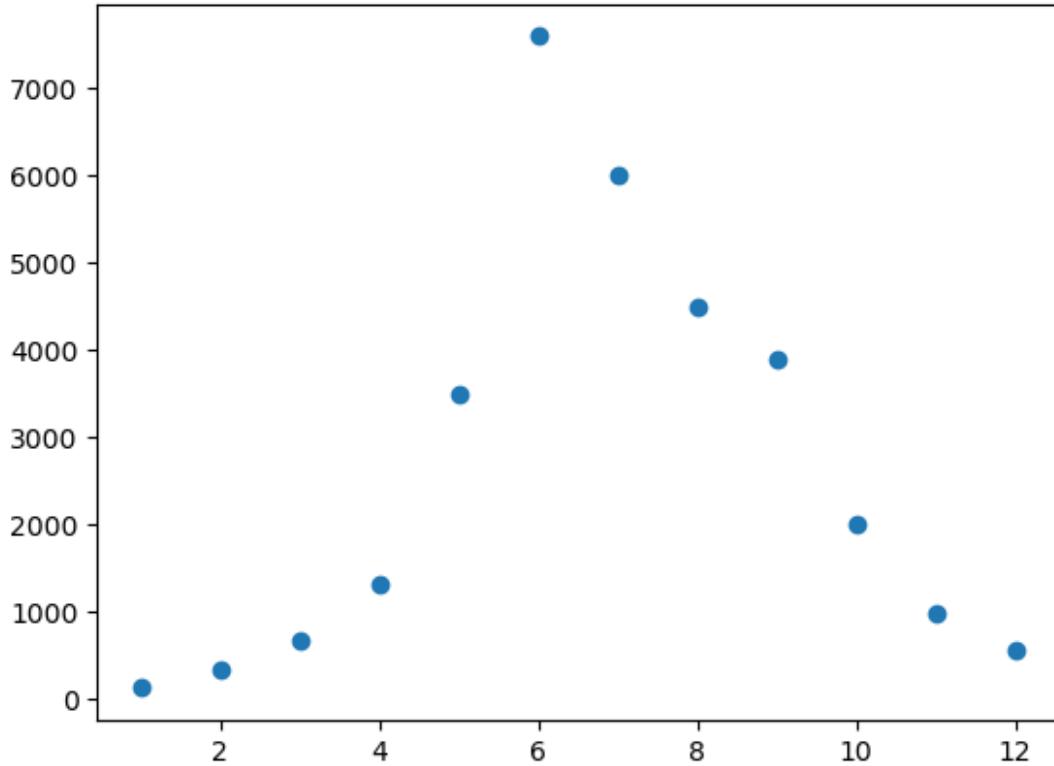


---

In [175]:

```
## Month wise covid cases in a particular state

import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,13)
y = np.array([120,340,670,1300,3500,7600,6000,4500,3890,1990,980,545])
plt.scatter(x,y)
plt.show()
```



---

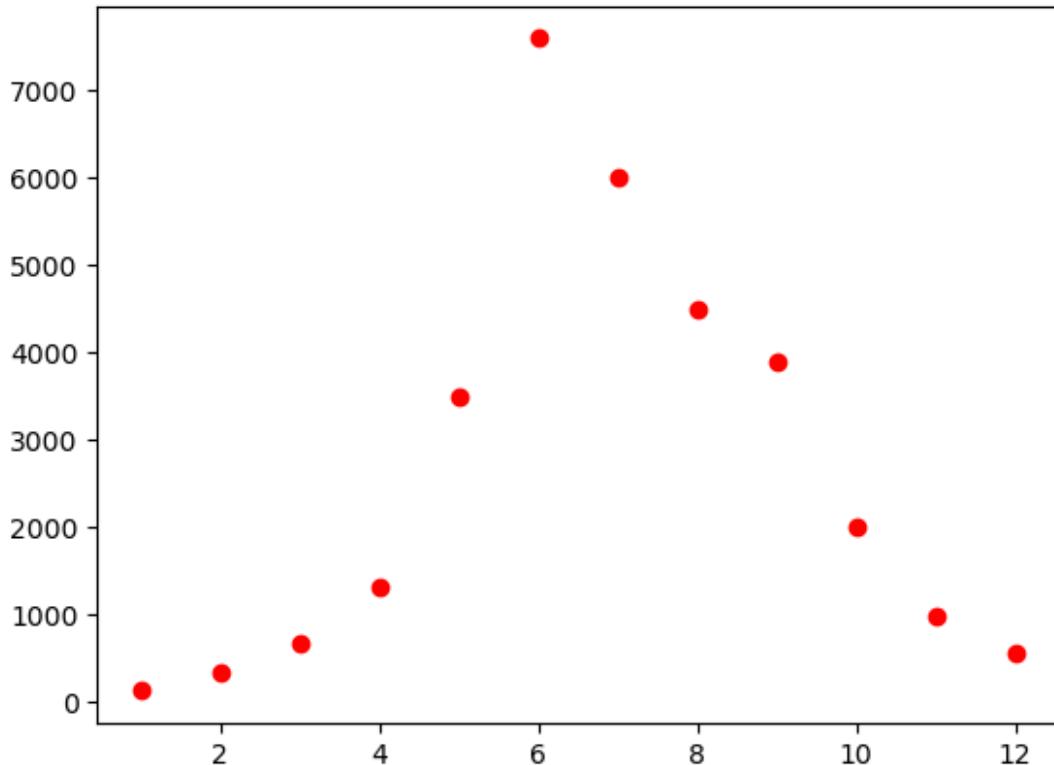
## Changing the color and size of the markers

### How to change color of the marker

By using **color** keyword argument we can change the color of the marker

In [176]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,13)
y = np.array([120,340,670,1300,3500,7600,6000,4500,3890,1990,980,545])
plt.scatter(x,y,color='red')
plt.show()
```

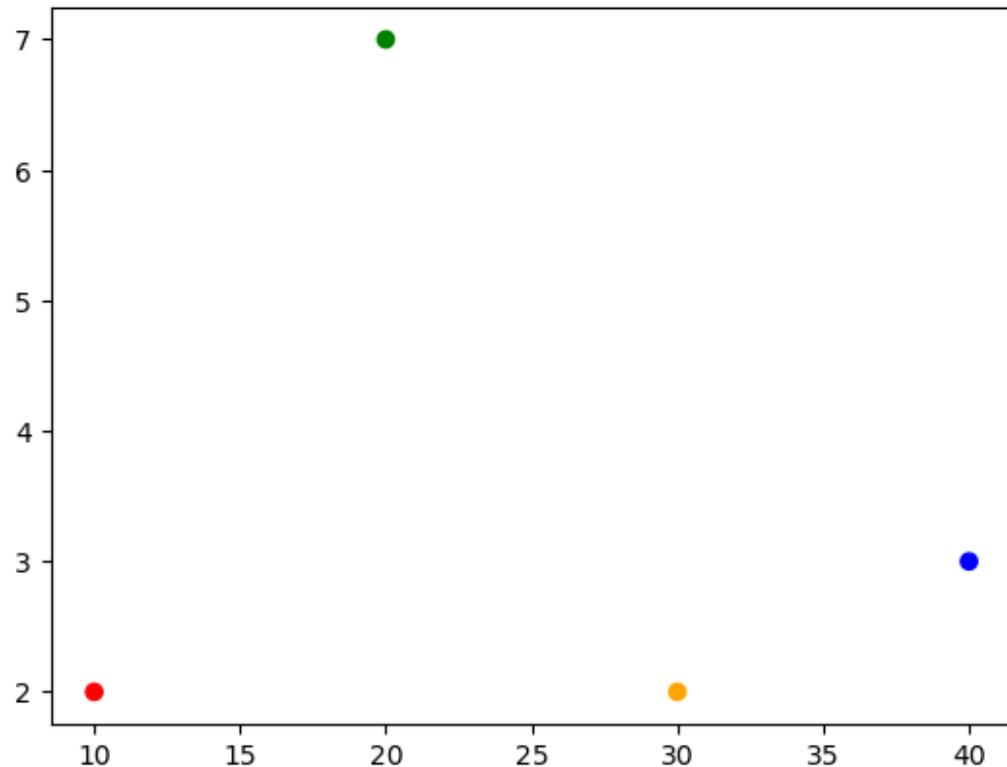


---

## Different colors for each markers

In [177]:

```
import matplotlib.pyplot as plt
import numpy as np
x = [10,40,30,20]
y = [2,3,2,7]
colors = ['red','blue','orange','green']
plt.scatter(x,y,color=colors)
plt.show()
```



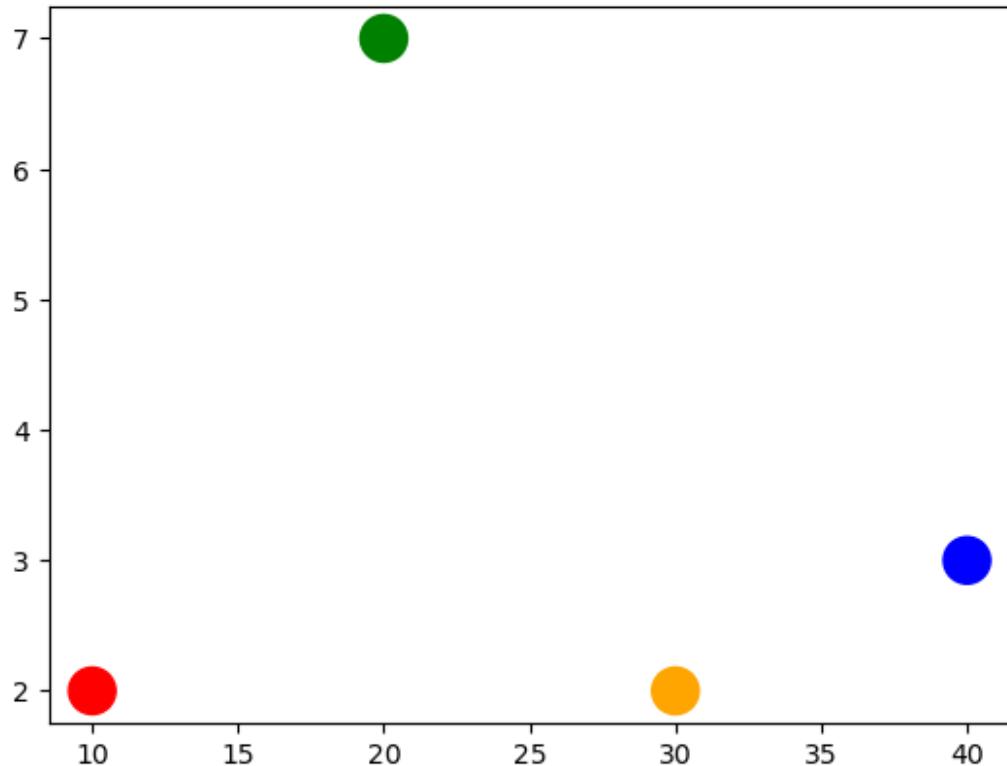
---

## Changing the size of the marker

we have to use **s** argument to change the size of the marker

In [178]:

```
import matplotlib.pyplot as plt
import numpy as np
x = [10,40,30,20]
y = [2,3,2,7]
colors = ['red','blue','orange','green']
plt.scatter(x,y,color=colors,s=300)
plt.show()
```

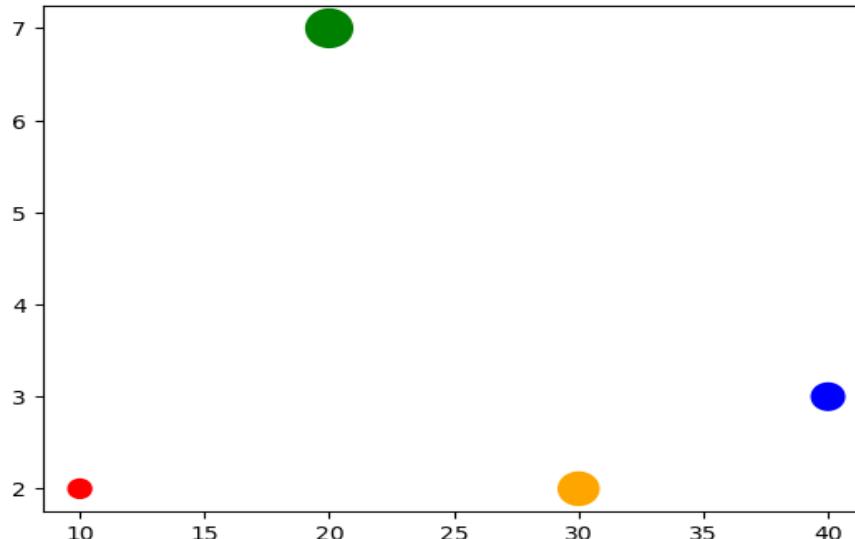


---

## Different sizes for the markers

In [179]:

```
import matplotlib.pyplot as plt
import numpy as np
x = [10,40,30,20]
y = [2,3,2,7]
colors = ['red','blue','orange','green']
sizes = [100,200,300,400]
plt.scatter(x,y,color=colors,s=sizes)
plt.show()
```



## cmap ==> colormap

- ✓ colormap maps **colors to numbers**.
- ✓ If **huge number of colors** are required, then we should go for **colormap**.
- ✓ matplotlib defines several predefined colormaps.
- ✓ The **default colormap is: 'viridis'**, where **0** represents **purple color** and **100** represents **yellow color**.
- ✓ To use **colormap** we have to use **cmap** argument.
- ✓ Whenever we are using **cmap** argument to represents the colors we have to use **c** argument.

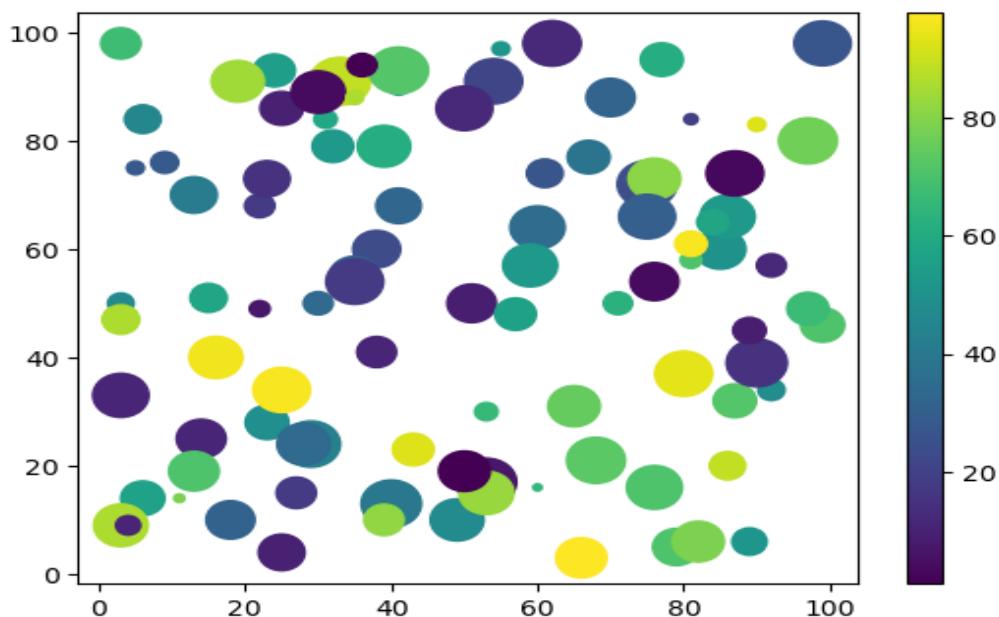
- 
- ✓ To display the color bar of the corresponding colormap we have to use **plt.colorbar()**

#### Note

- ✓ if we use **viridis\_r** then the values are reversed. **0** represents **yellow color** and **100** represents **purple color**
- ✓ Documentation link → <https://matplotlib.org/stable/tutorials/colors/colormaps.html>

In [180]:

```
# default colormap is 'viridis'
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors)
plt.colorbar()
plt.show()
```

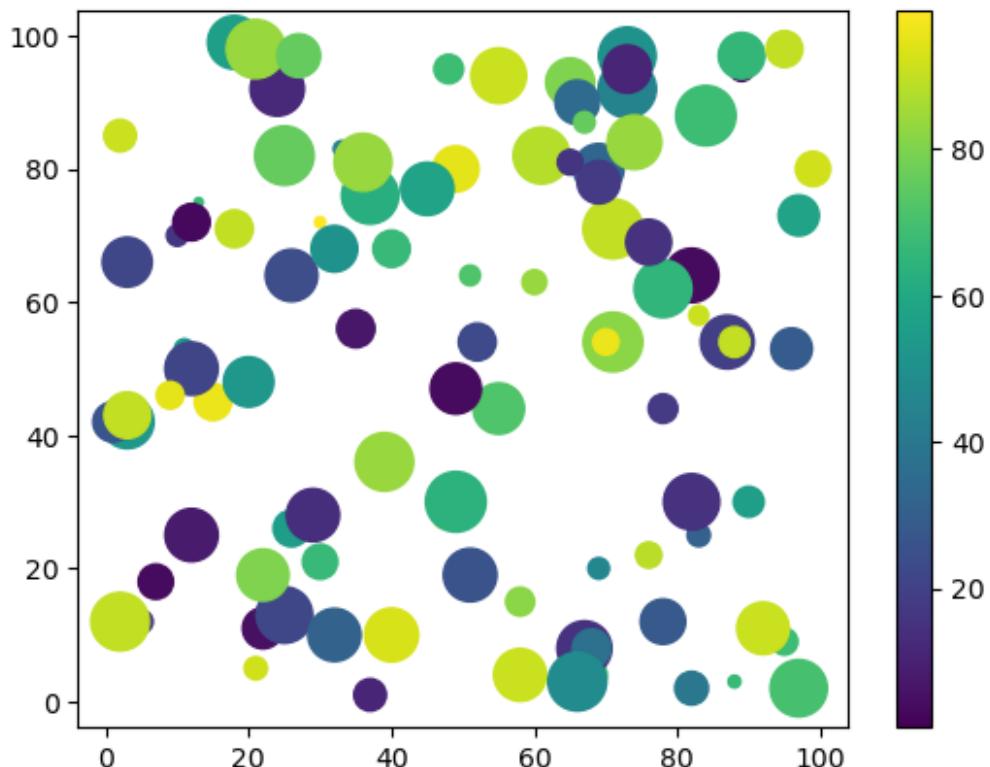


---

## colormap ==> viridis

In [181]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='viridis')
plt.colorbar()
plt.show()
```

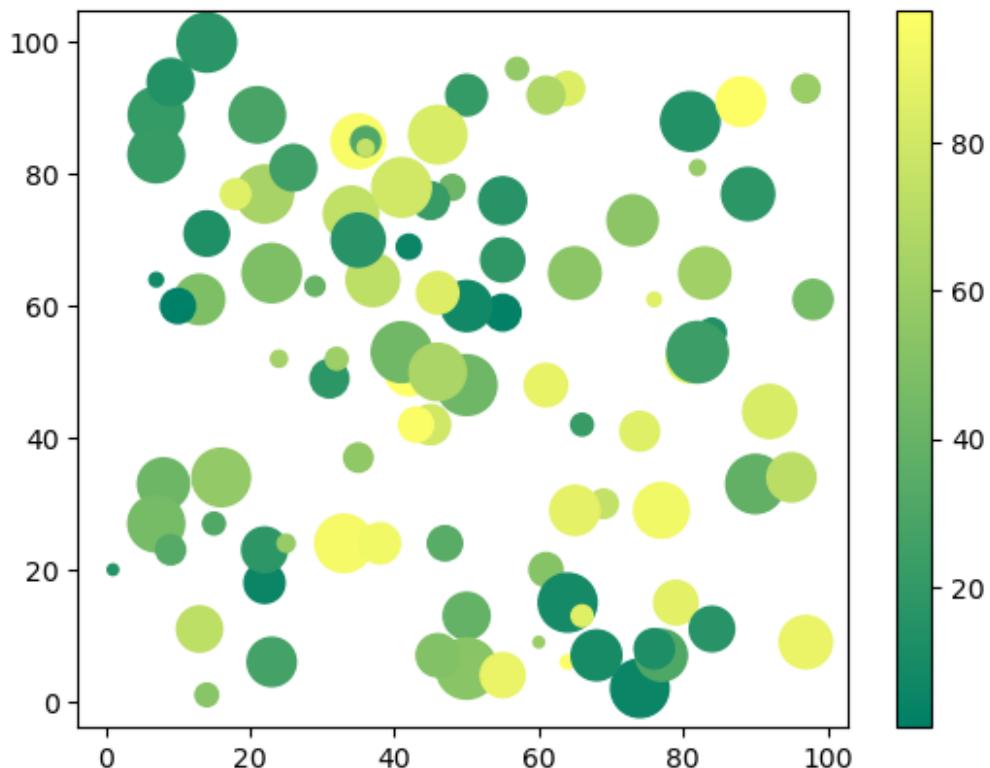


---

## colormap ==> summer

In [182]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='summer')
plt.colorbar()
plt.show()
```

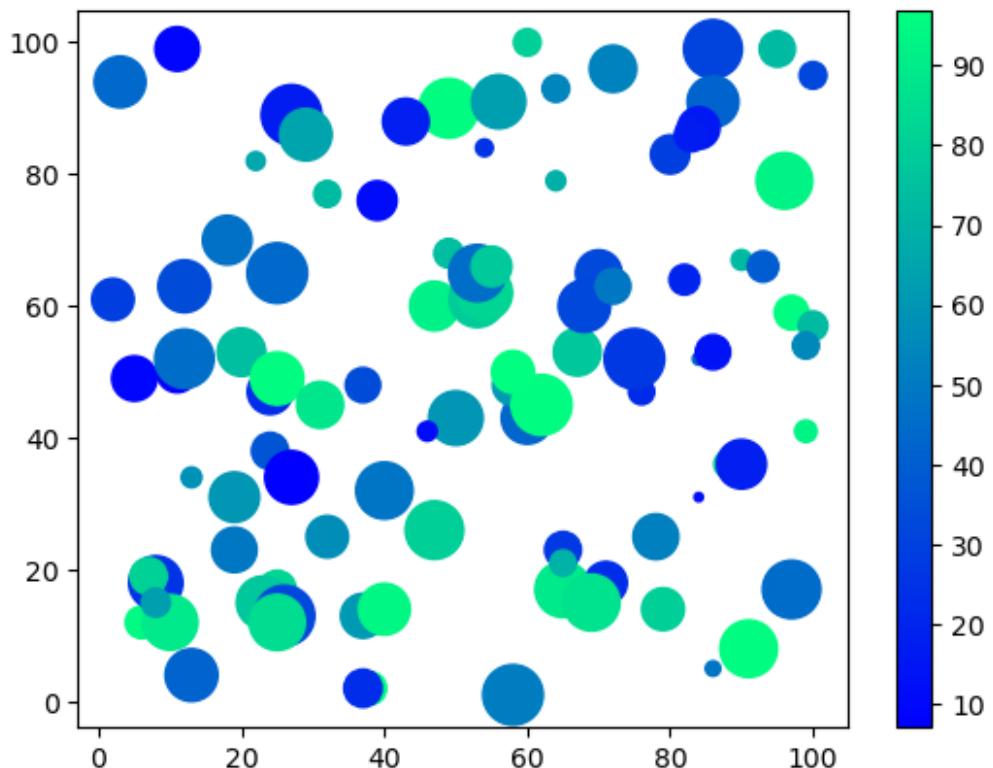


---

## colormap ==> winter

In [183]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='winter')
plt.colorbar()
plt.show()
```

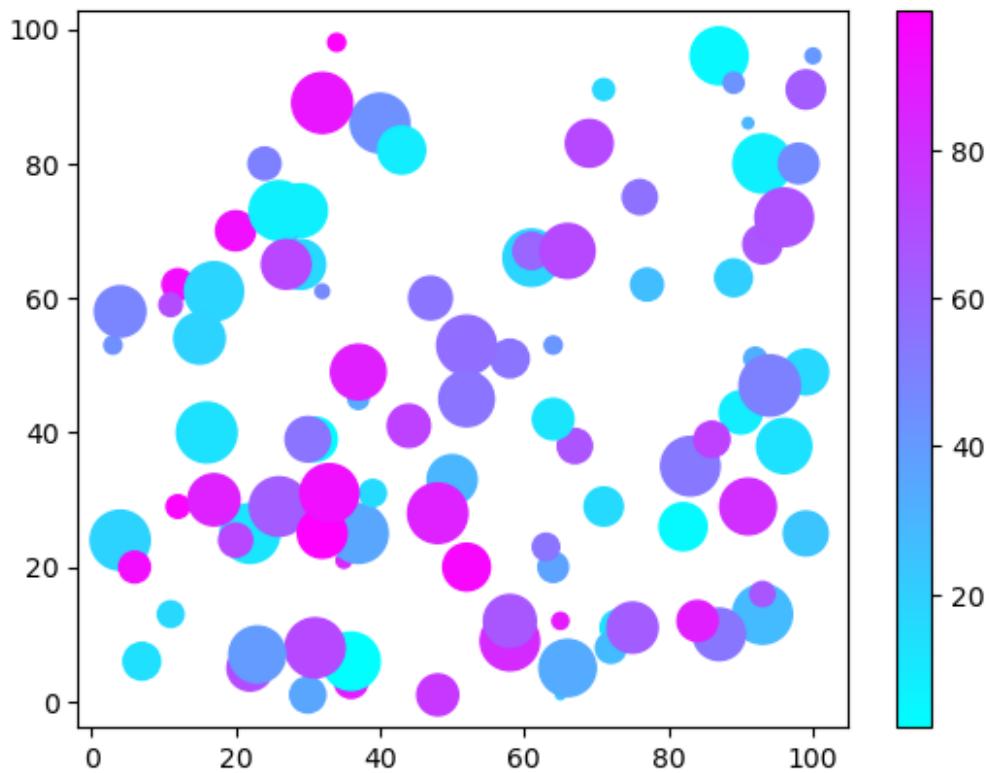


---

## colormap ==> cool

In [184]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='cool')
plt.colorbar()
plt.show()
```

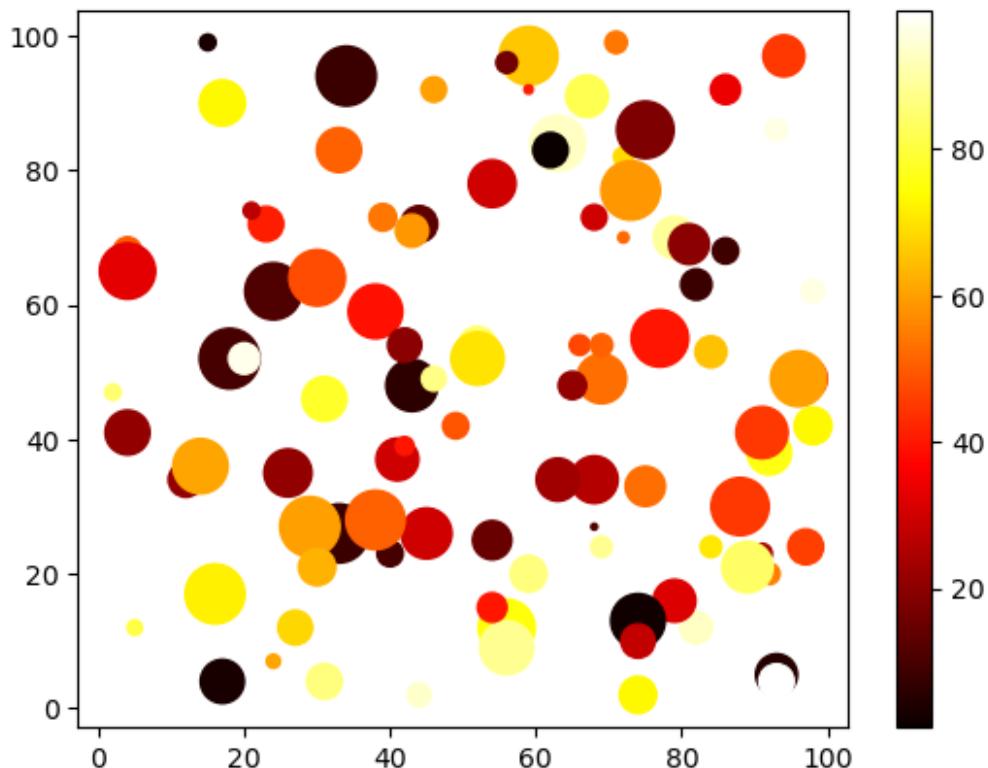


---

## colormap ==> hot

In [185]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='hot')
plt.colorbar()
plt.show()
```

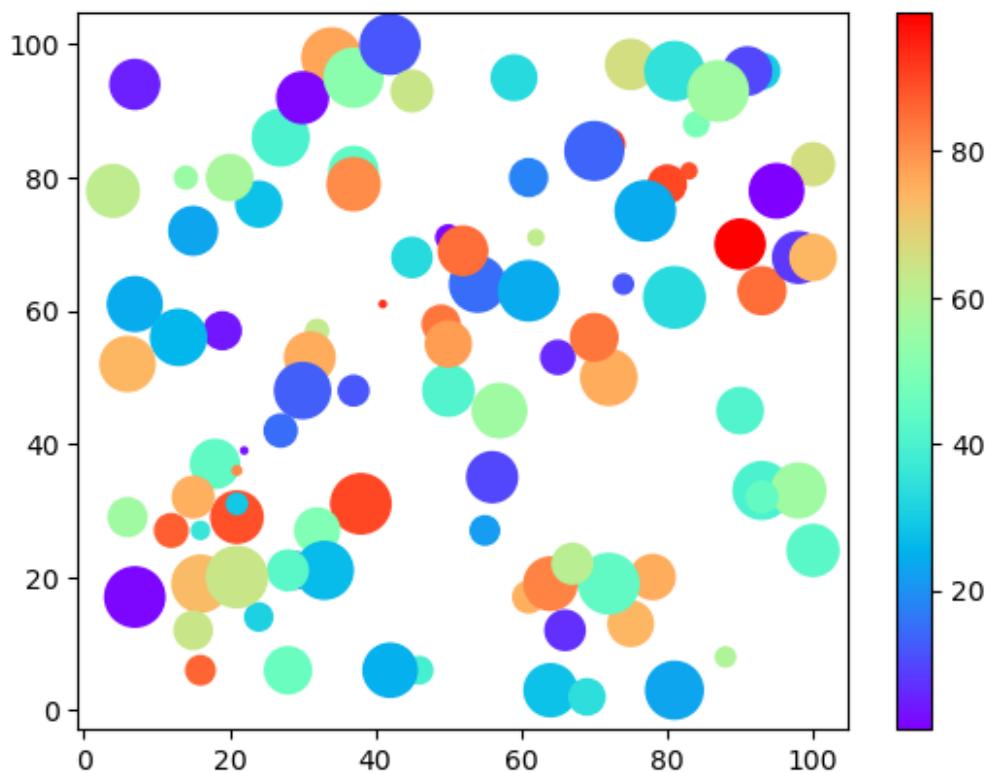


---

## colormap ==> rainbow

In [186]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='rainbow')
plt.colorbar()
plt.show()
```

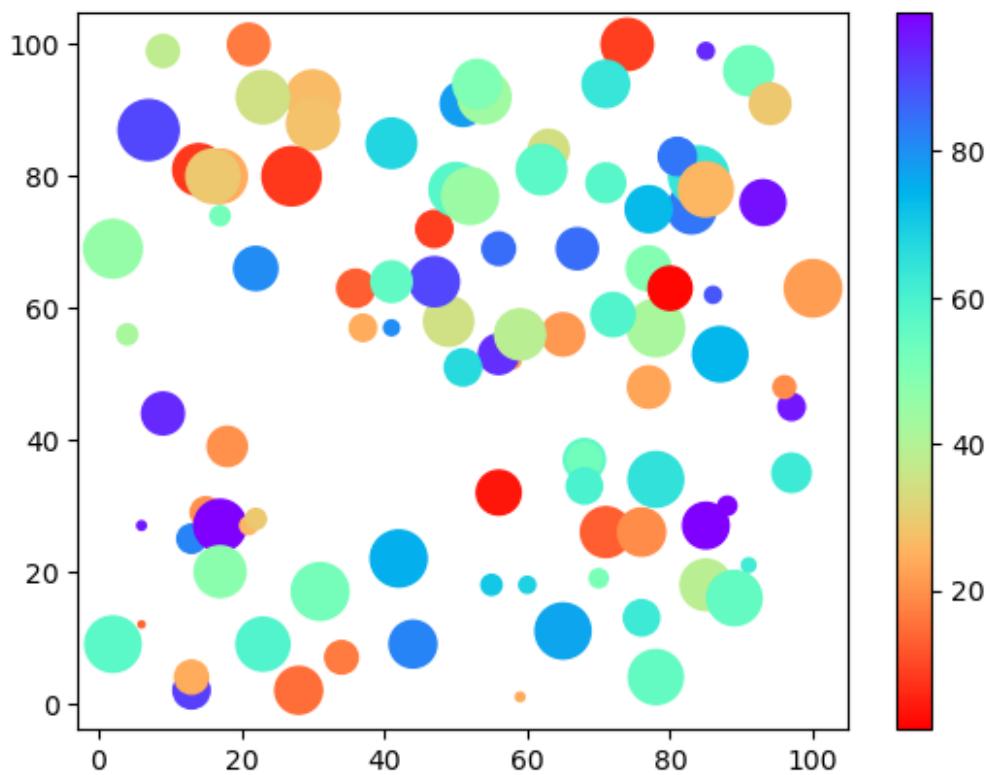


---

**colormap ==> rainbow\_r ==> reverse of rainbow**

In [187]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='rainbow_r')
plt.colorbar()
plt.show()
```

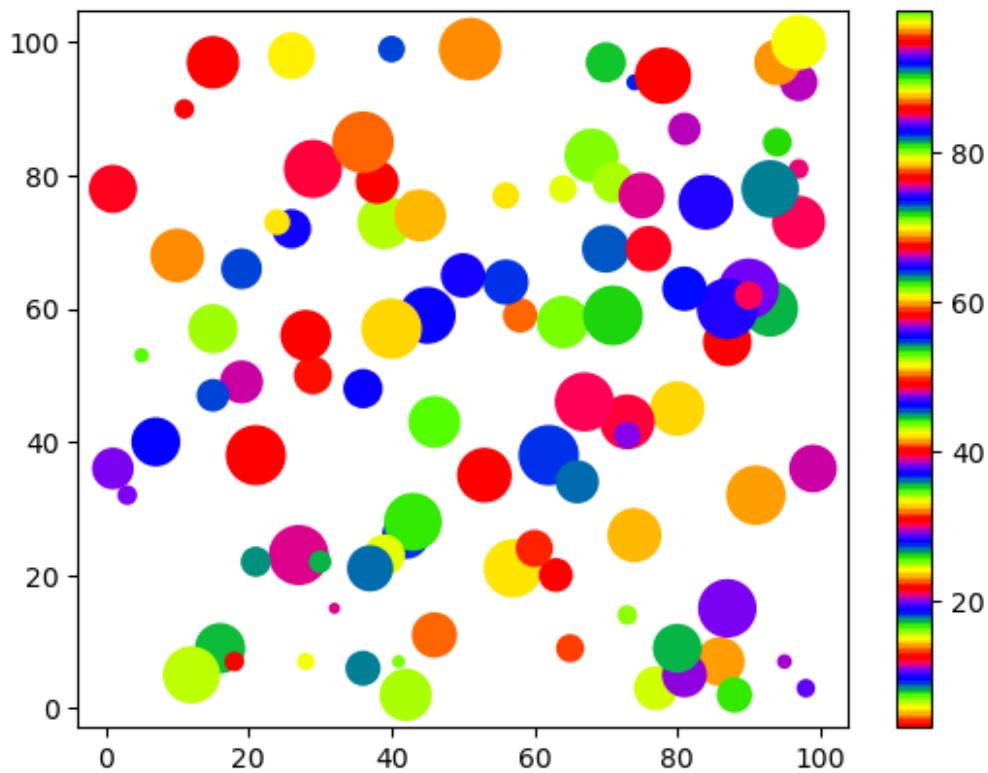


---

## colormap ==> prism

In [188]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='prism')
plt.colorbar()
plt.show()
```

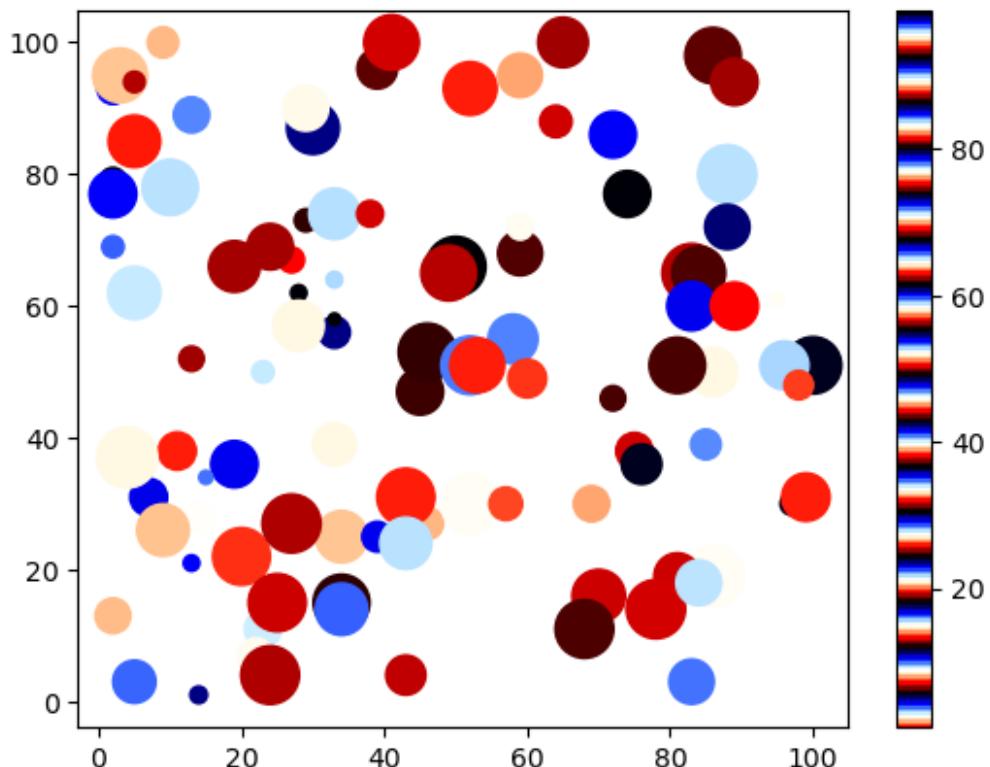


---

## colormap ==> flag

In [189]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='flag')
plt.colorbar()
plt.show()
```



---

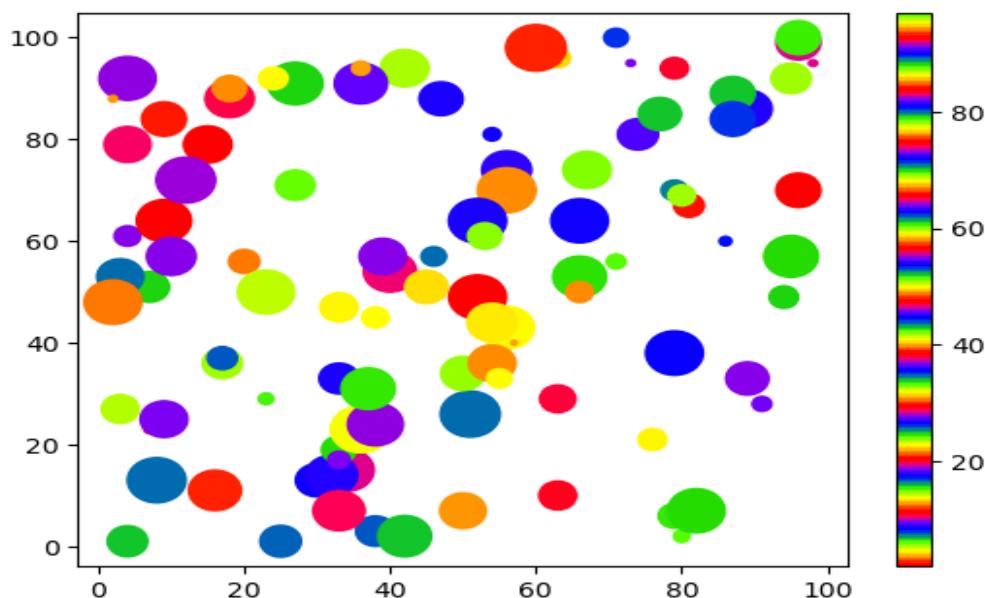
## Note

- ✓ Whenever we are using **cmap** then to specify colors we have to **use keyword argument c but not color**.
- ✓ **eg-1** ==> plt.scatter(x,y,s=sizes,color=colors,cmap='prism') ➔ invalid ValueError: 'color' kwarg must be an color or sequence of color specs. For a sequence of values to be color-mapped, use the 'c' argument instead.
- ✓ **eg-2** ==> plt.scatter(x,y,s=sizes,c=colors,cmap='prism') ➔ valid
- ✓ **eg-3** ==> plt.scatter(x,y,s=sizes,c='r') ➔ valid

### usage of keyword argument c with cmap

In [190]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c=colors,cmap='prism')
plt.colorbar()
plt.show()
```

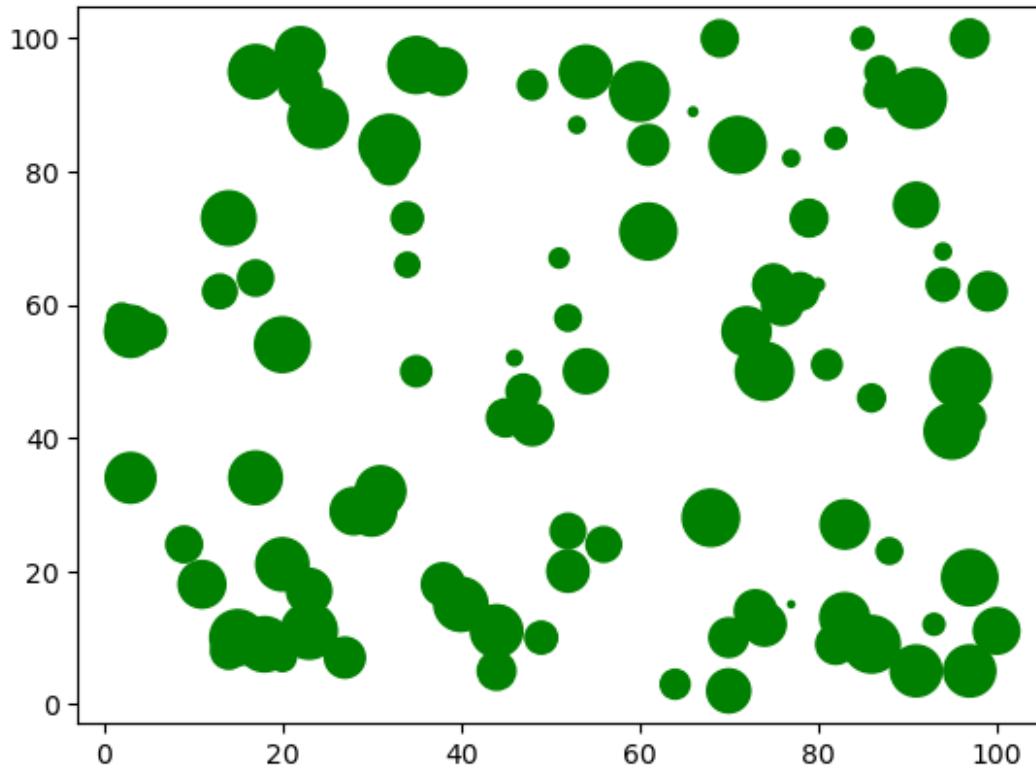


---

### usage of keyword argument c without cmap

In [191]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,c='g')
plt.show()
```



---

## Note

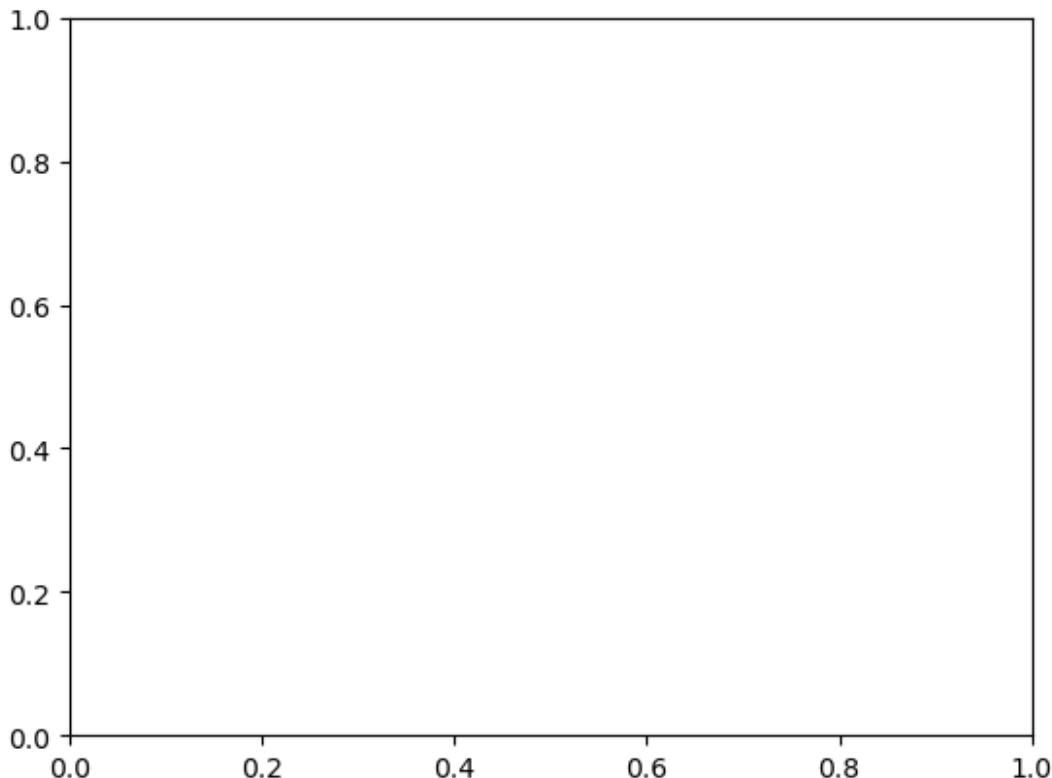
- ✓ To specify colors for the markers we can use either keyword argument **c** and **color**.
- ✓ But when we are using **cmap** keyword argument we should use **c** but not **color**  
plt.scatter(x,y,s=sizes,color=colors,cmap='prism') → invalid

In [192]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.random.randint(1,101,size=(100))
y = np.random.randint(1,101,size=(100))
sizes = 5*np.random.randint(1,101,size=(100))
colors = np.random.randint(1,100,100)
plt.scatter(x,y,s=sizes,color=colors,cmap='prism')
plt.colorbar()
plt.show()
```

---

ValueError: 'color' kwarg must be an color or sequence of color specs. For a sequence of values to be color-mapped, use the 'c' argument instead.



### Line plot vs scatter plot:

- ✓ In the case of lineplot, all markers should have same size and same color. But in scatter plot, markers can have different sizes and different colors also.
- ✓ line plot is used to represent continuous trends, scatter plot is for just marking multiple values and values need not be continuous.

### How to add labels to scatter plot data points:

We can add labels to the scatter plot data points in 2 ways

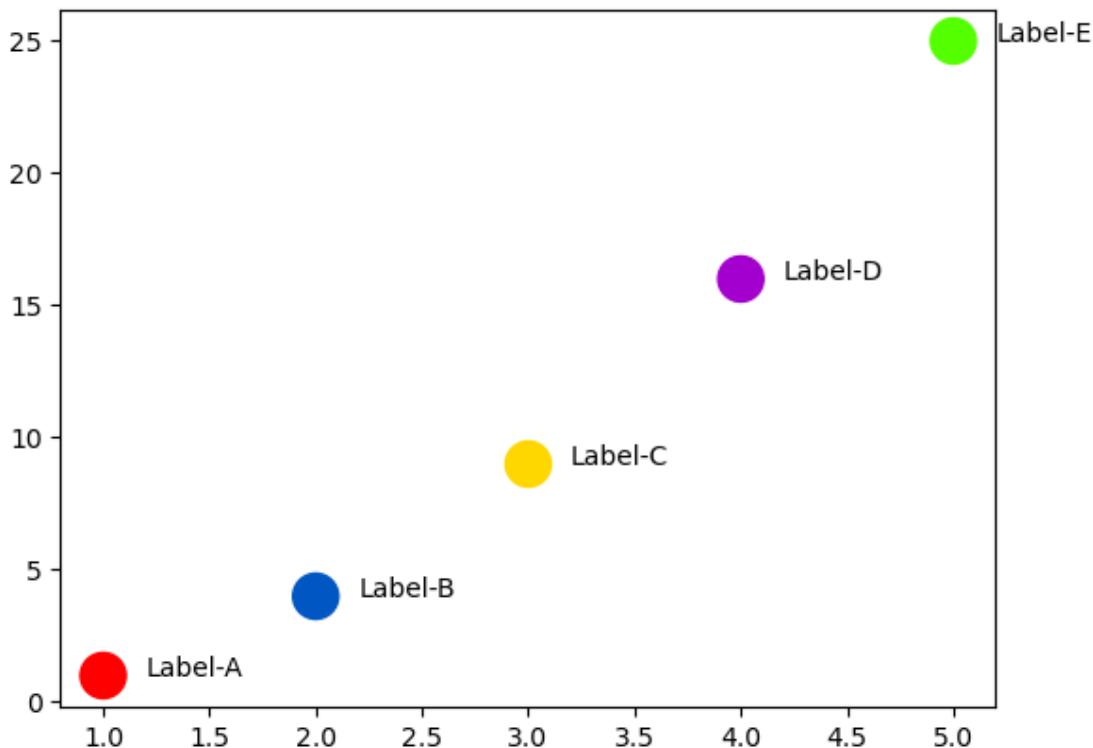
- ✓ plt.text(x,y,text)
- ✓ plt.annotate(text,(x,y))

---

**using plt.text(x,y,text)**

In [193]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,6) #1 to 5
y = x ** 2 #1,4,9,16,25
labels = ['Label-A','Label-B','Label-C','Label-D','Label-E']
plt.scatter(x,y,s=300,c=[0,100,200,300,400],cmap='prism')
for i, label in enumerate(labels):
    plt.text(x[i]+0.2,y[i],label)
plt.show()
```

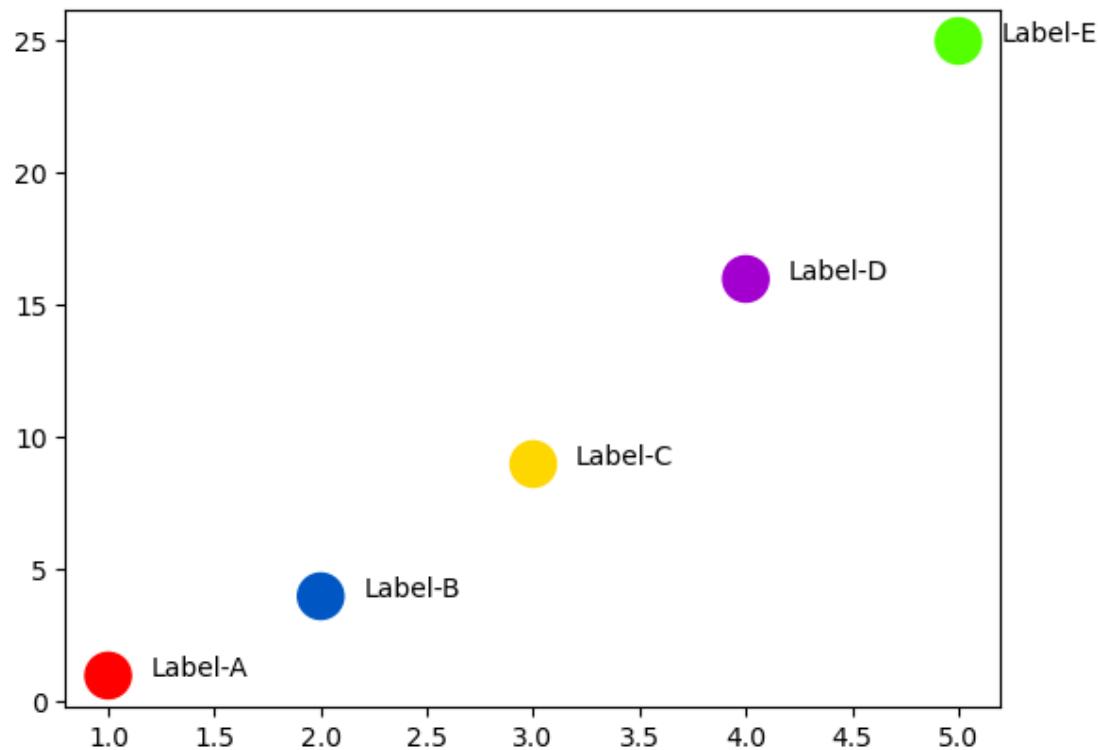


---

**using plt.annotate(text,(x,y))**

In [194]:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.arange(1,6) #1 to 5
y = x ** 2 #1,4,9,16,25
labels = ['Label-A','Label-B','Label-C','Label-D','Label-E']
plt.scatter(x,y,s=300,c=[0,100,200,300,400],cmap='prism')
for i, label in enumerate(labels):
    plt.annotate(label,(x[i]+0.2,y[i]))
plt.show()
```



---

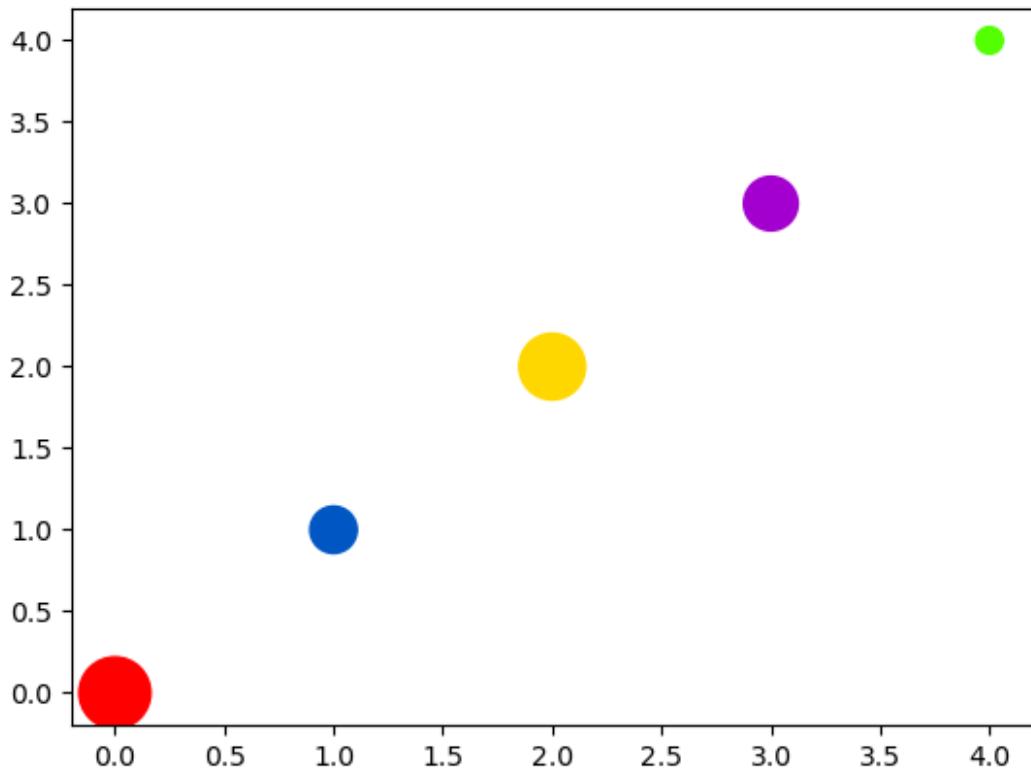
## How to add color legend to the scatter plot

- ✓ `plt.scatter()` returns **matplotlib.collections.PathCollection object**
- ✓ To get markers of scatter plot, we have to use `legend_elements()[0]` on the PathCollection object.
- ✓ we can use **legend()** function to add color legend to the scatter plot
- ✓ while calling the **legend()** function for **handles** argument we have to pass the **PathCollection** object.

In [195]:

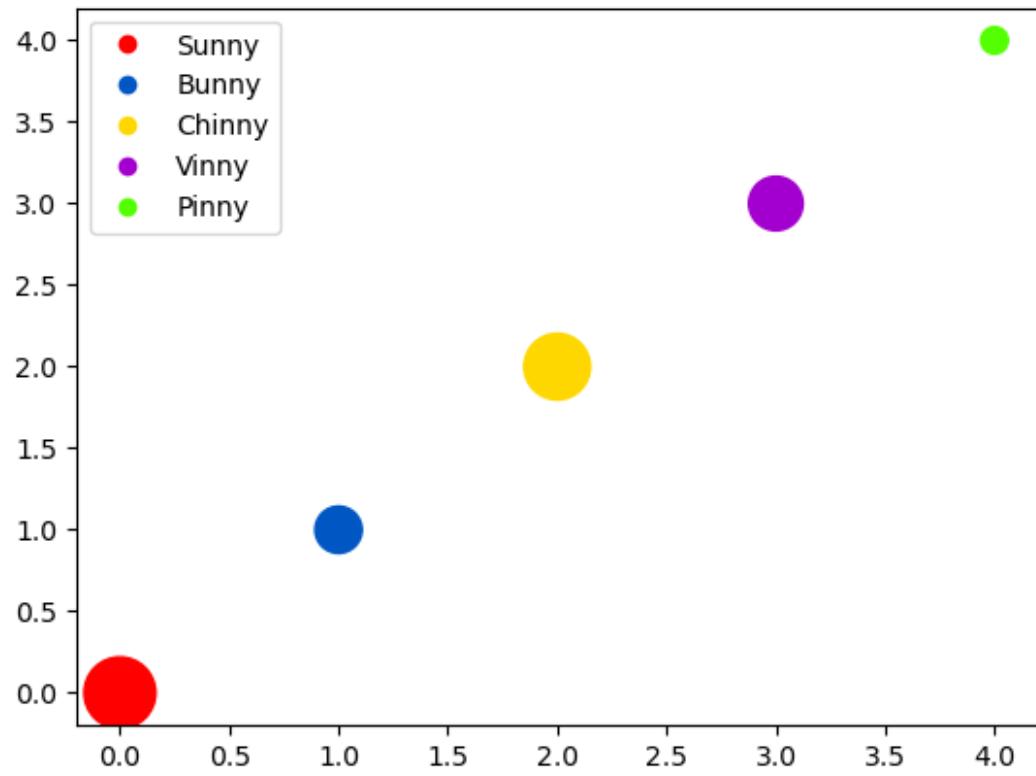
```
import matplotlib.pyplot as plt
import numpy as np
names = ['Sunny','Bunny','Chinny','Vinny','Pinny']
marks = [700,300,600,400,100]
x = np.arange(len(names)) #[0,1,2,3,4]
y = x
scat = plt.scatter(x,y,s = marks,c=x,cmap='prism')
print(fReturn type of scatter(): => {type(scat)}')
print(fvalues returned by the scatter() => {scat}'')
print(fObjects of PathCollection : {scat.legend_elements()[0]}')
```

Return type of scatter(): => <class 'matplotlib.collections.PathCollection'>  
values returned by the scatter() => <matplotlib.collections.PathCollection object  
at 0x0000020330DF6550>  
Objects of PathCollection : [<matplotlib.lines.Line2D object at  
0x0000020330DF6850>, <matplotlib.lines.Line2D object at  
0x0000020330DF6970>, <matplotlib.lines.Line2D object at  
0x000002032F17E7F0>, <matplotlib.lines.Line2D object at  
0x0000020330DF6A00>, <matplotlib.lines.Line2D object at  
0x0000020330DF6B50>]



In [196]:

```
# Adding color legend to the scatter plot
import matplotlib.pyplot as plt
import numpy as np
names = ['Sunny','Bunny','Chinny','Vinny','Pinny']
marks = [700,300,600,400,100]
x = np.arange(len(names)) #[0,1,2,3,4]
y = x
scat = plt.scatter(x,y,s = marks,c=x,cmap='prism')
plt.legend(handles = scat.legend_elements()[0],labels = names)
plt.show()
```



---

## Kaggle Case Study : Latest Covid-19 India Statewise Data

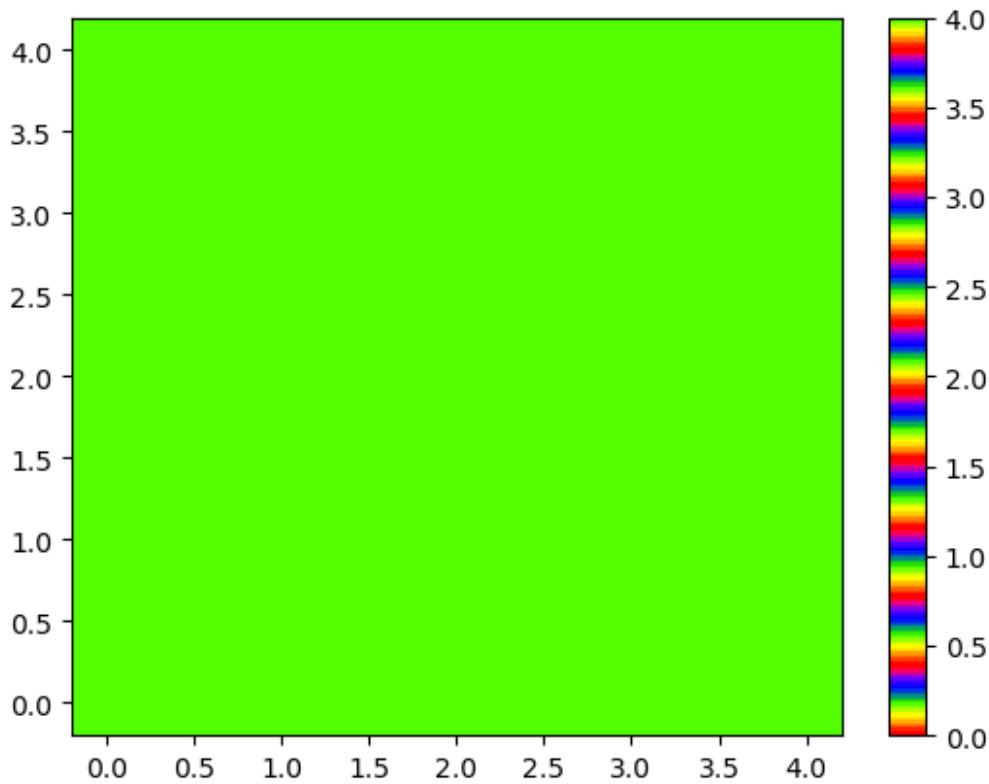
Dataset →

<https://www.kaggle.com/anandhuh/latest-covid19-india-statewise-data>

scatter plot statewise total number of cases and labels also must be required.

In [197]:

```
# Display the top 5 states data on the plot
import matplotlib.pyplot as plt
import numpy as np
import csv
state_names = []
total_cases = np.array([],dtype=int)
f = open('Latest Covid-19 India Status.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
h = next(r) # read header column and ignore
for row in r:
    state_names.append(row[0])
    total_cases = np.append(total_cases,int(row[1]))
x = np.arange(5)
plt.scatter(x,x,s = total_cases[:5],c=x,cmap='prism')
plt.colorbar()
plt.show()
```



### Note

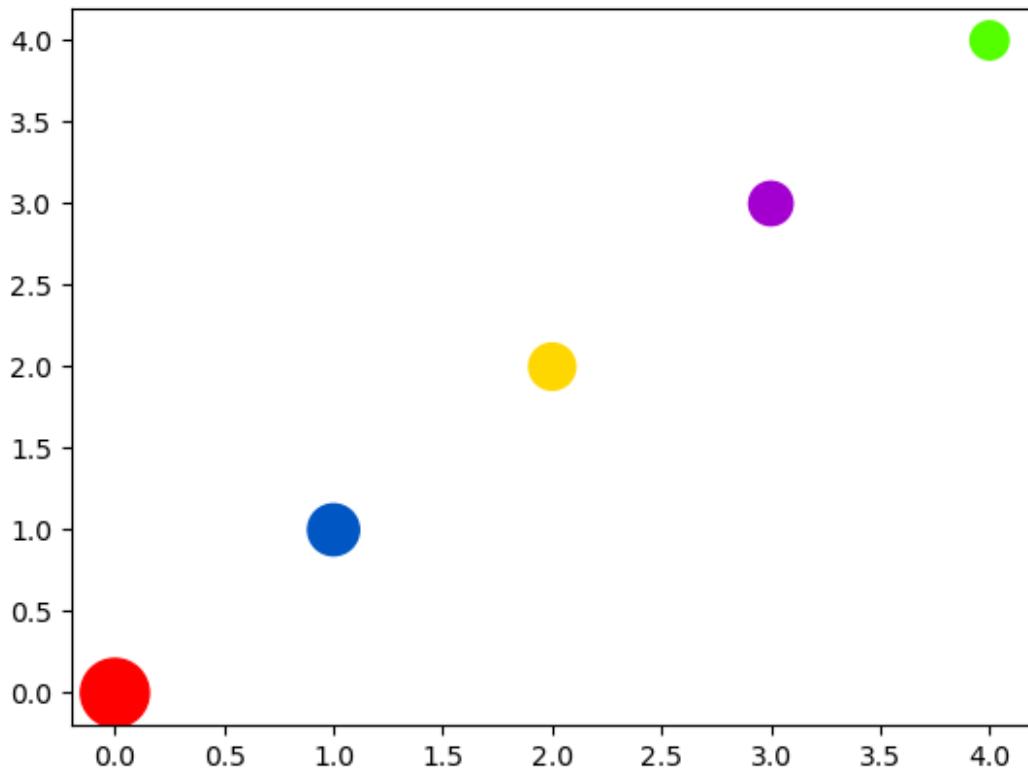
- ✓ The problem in above plot is total\_cases are having very large values in lakhs
- ✓ To resolve this we have to use **total\_cases[:5]/10000**

In [198]:

```
# Display the first 5 states data on the plot
import matplotlib.pyplot as plt
import numpy as np
import csv
state_names = []
total_cases = np.array([],dtype=int)
f = open('Latest Covid-19 India Status.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
h = next(r) # read header column and ignore
for row in r:
```

```
state_names.append(row[0])
total_cases = np.append(total_cases,int(row[1]))
x = np.arange(5)
plt.scatter(x,x,s = total_cases[:5]/10000,c=x,cmap='prism')

plt.show()
```



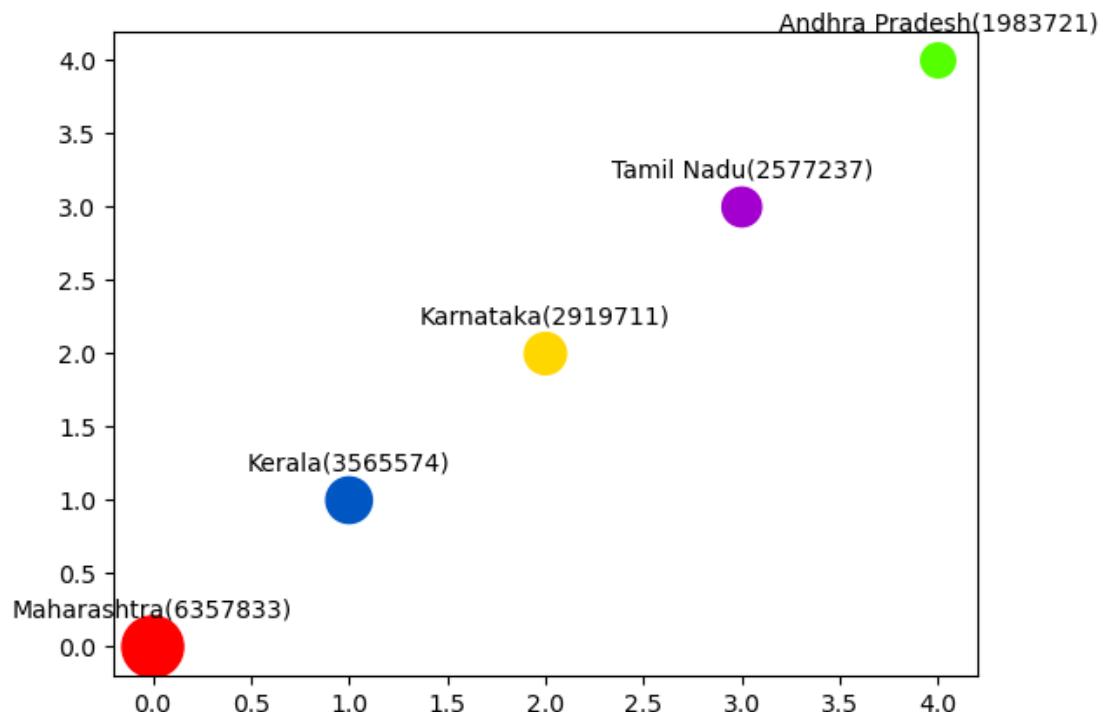
In [199]:

```
# scatter plot statewise total number of cases and labels.
import matplotlib.pyplot as plt
import numpy as np
import csv
state_names = []
total_cases = np.array([],dtype=int)
f = open('Latest Covid-19 India Status.csv','r',encoding='utf-8')
r = csv.reader(f) #returns csvreader object
h = next(r) # read header column and ignore
```

```

for row in r:
    state_names.append(row[0])
    total_cases = np.append(total_cases,int(row[1]))
x = np.arange(5)
y=x
plt.scatter(x,y,s = total_cases[:5]/10000,c=x,cmap='prism')
for i,label in enumerate(state_names[:5]):
    plt.text(x[i],y[i]+0.20,f'{label}({total_cases[i]})',ha='center')

```



---

## Chapter-15 Subplots

### Subplots

The two important terms

- ✓ Figure
- ✓ Axes

#### 1. Figure:

Figure is an individual window on the screen, in which matplotlib displays the graphs. i.e it is a container for graphical output.

#### 2. Axes:

- ✓ The axes is the plotting area, contained within the figure object.
- ✓ For every graph, we can take separate axes.
- ✓ Inside axes only, we can take x-axis,y-axis,grid,legend,bars,data points etc
- ✓ Per Figure, we can take any number of axes objects.
- ✓ Per axes, we can take only one graph.

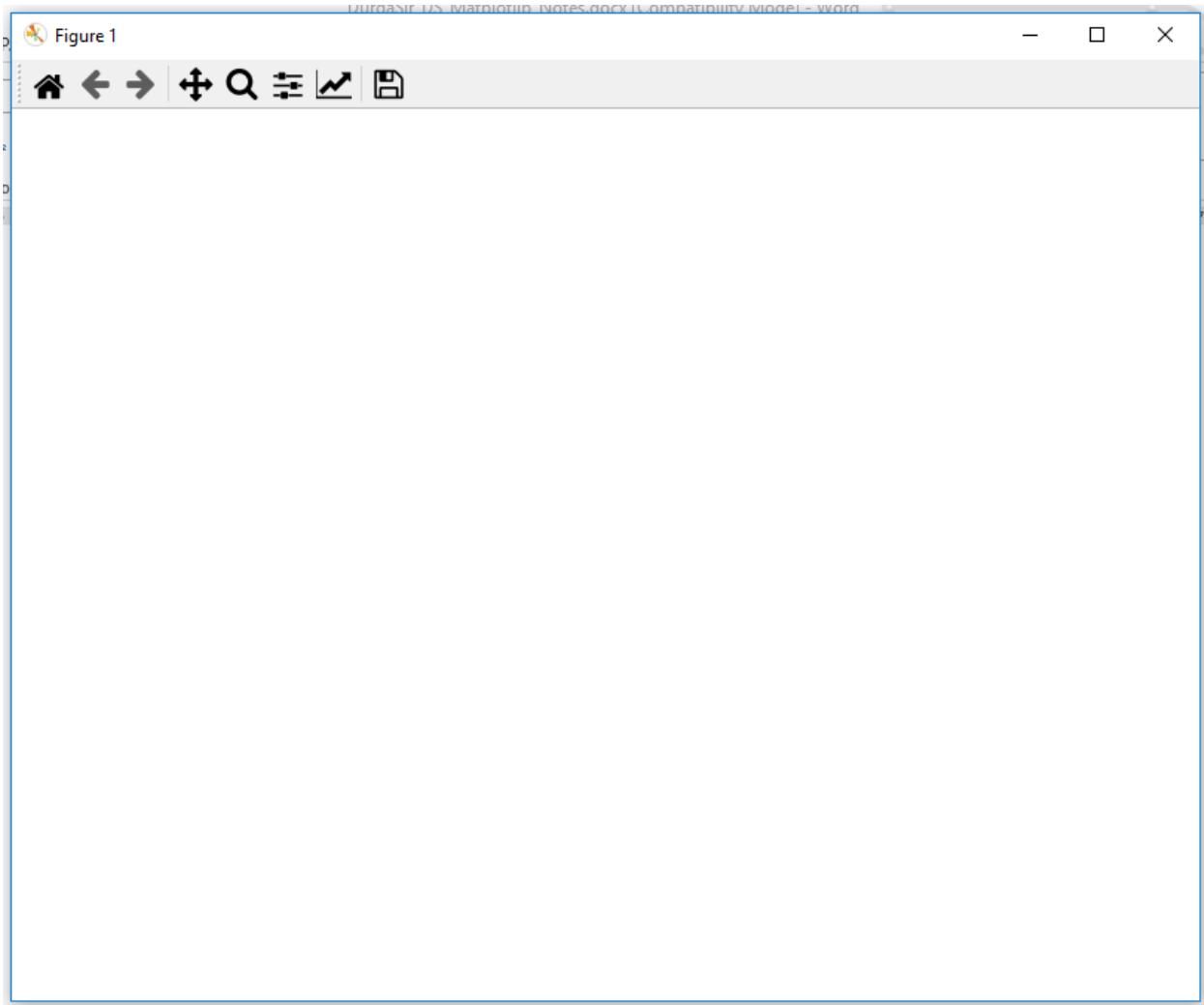
In [200]:

```
# Creating the figure object
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)

plt.show()
```

<Figure size 800x600 with 0 Axes>

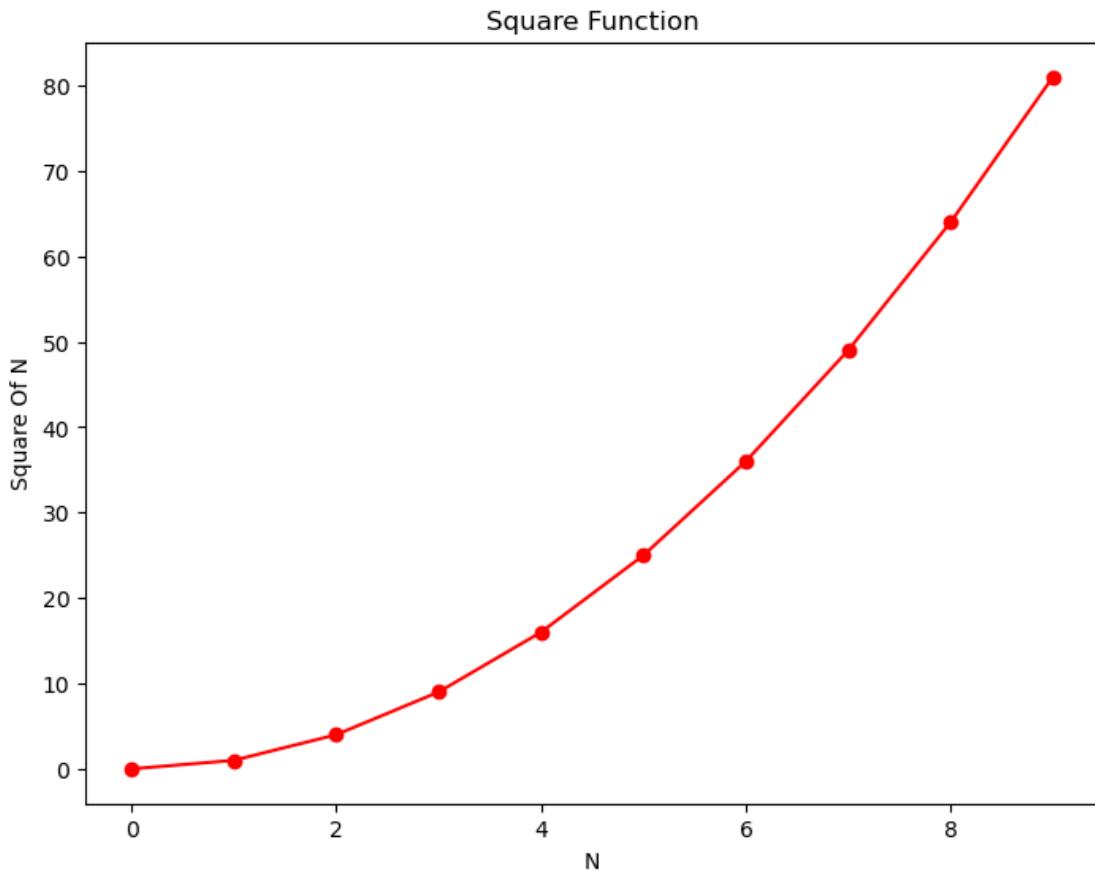


In [201]:

```
# creating axes object
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = fig.add_axes([0.1,0.1,0.8,0.8]) #[l,b,w,h]
```

```
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')
plt.show()
```



In [202]:

```
# creating subplot

import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
```

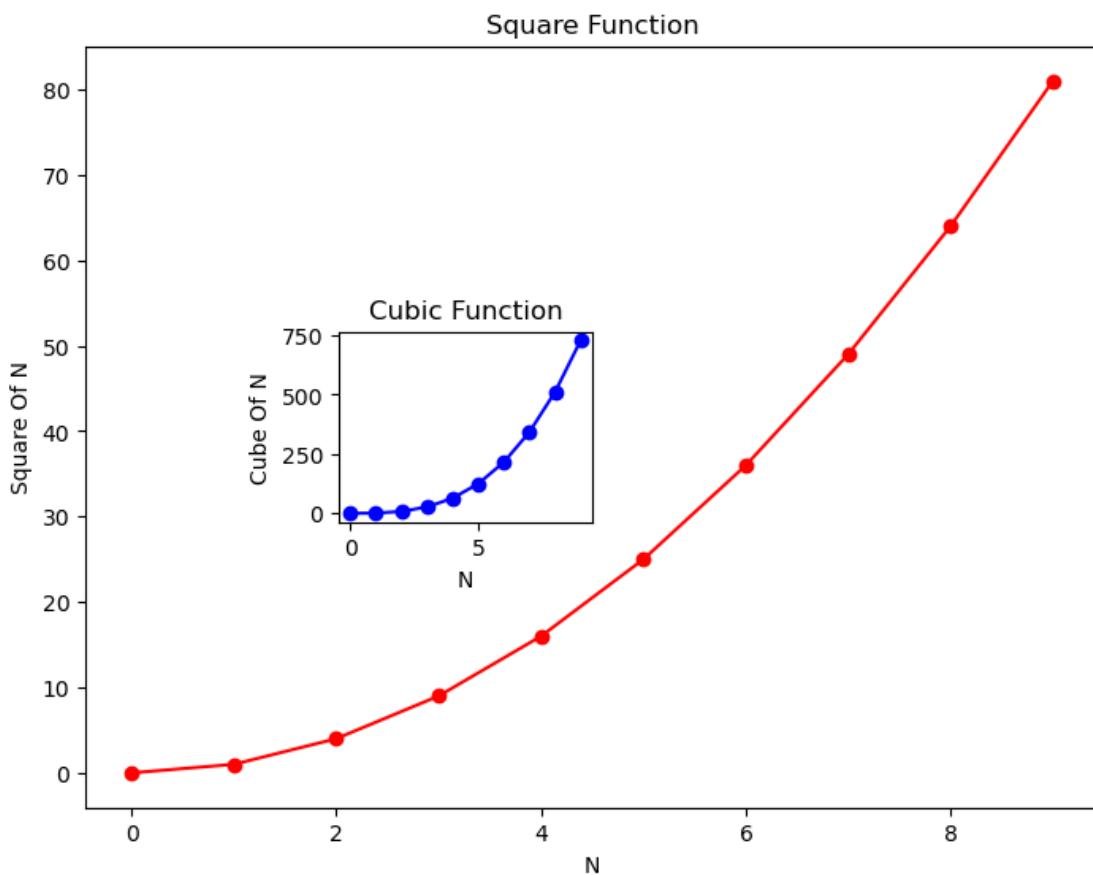
```

ax1 = fig.add_axes([0.1,0.1,0.8,0.8]) #[l,b,w,h]
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax2 = fig.add_axes([0.3,0.4,0.2,0.2]) #[l,b,w,h]
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.show()

```



---

## Need of subplots

- ✓ in the above resultant diagram, one plot placed inside another plot(Nested plot).
- ✓ If we want to place plots side by side and one plot on top of another, in well organized way, then we should go for subplots concept.

## How to create subplots

We can create subplots by using the following functions

1. pyplot.subplot() function
2. pyplot.subplots() function

### pyplot.subplot() function

In [203]:

```
import matplotlib.pyplot as plt
help(plt.subplot)
```

Help on function subplot in module matplotlib.pyplot:

```
subplot(*args, **kwargs)
    Add a subplot to the current figure.
```

### Call signatures::

```
subplot(nrows, ncols, index, **kwargs)
subplot(pos, **kwargs)
subplot(**kwargs)
subplot(ax)
```

### Note

- ✓ Here **index** starts from 1
- ✓ plt.subplot(1,1,1)

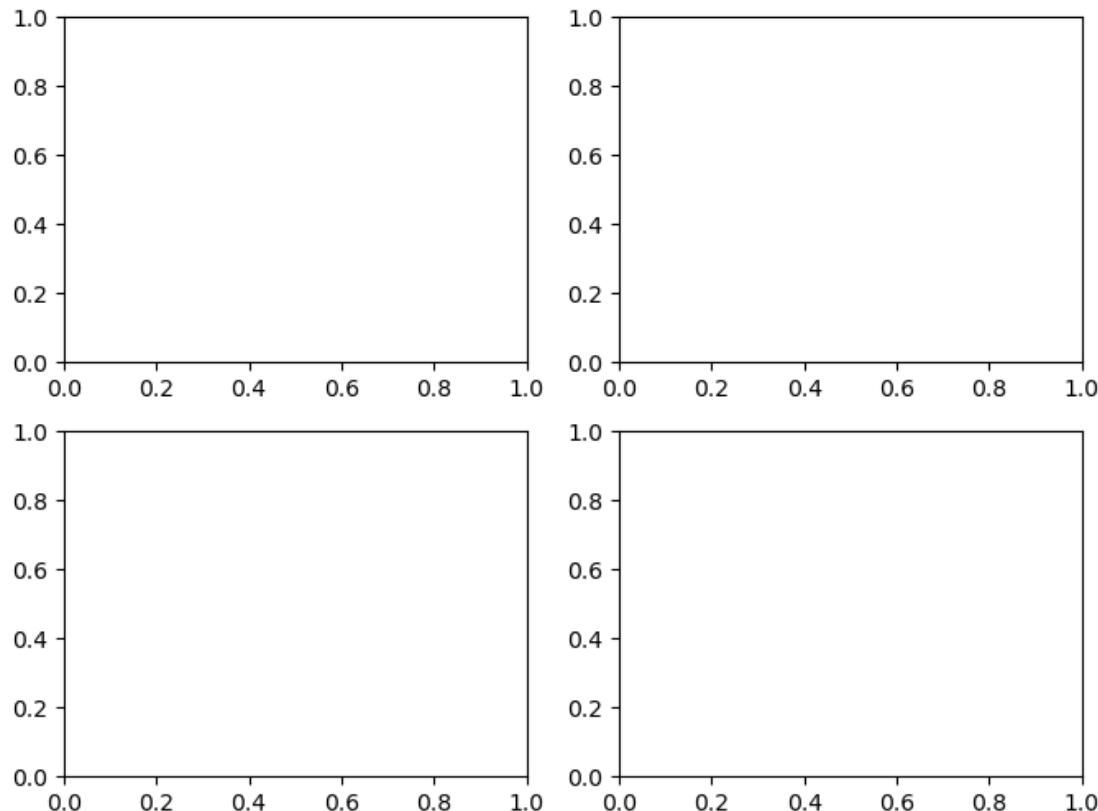
---

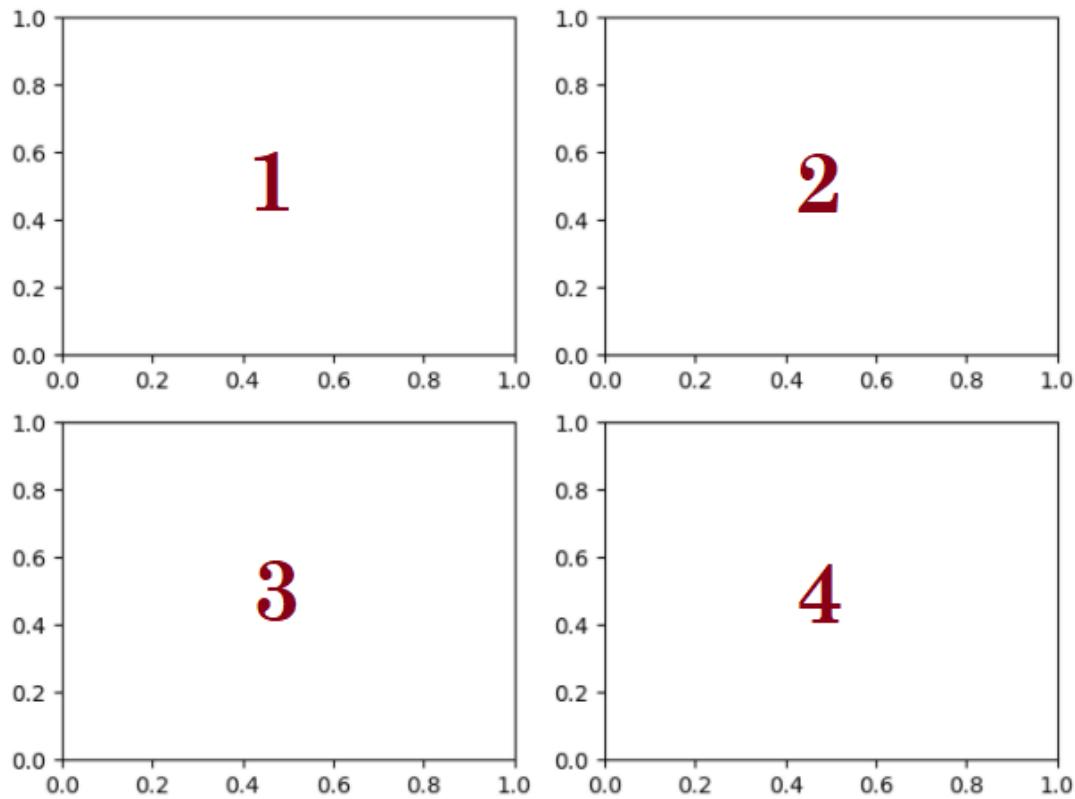
In [204]:

```
# Demo program ==> subplot(nrows, ncols, index, **kwargs)
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(2,2,1)
ax2 = plt.subplot(2,2,2)
ax3 = plt.subplot(2,2,3)
ax4 = plt.subplot(2,2,4)

plt.show()
```





In [205]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(2,2,1)
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax2 = plt.subplot(2,2,2)
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')
```

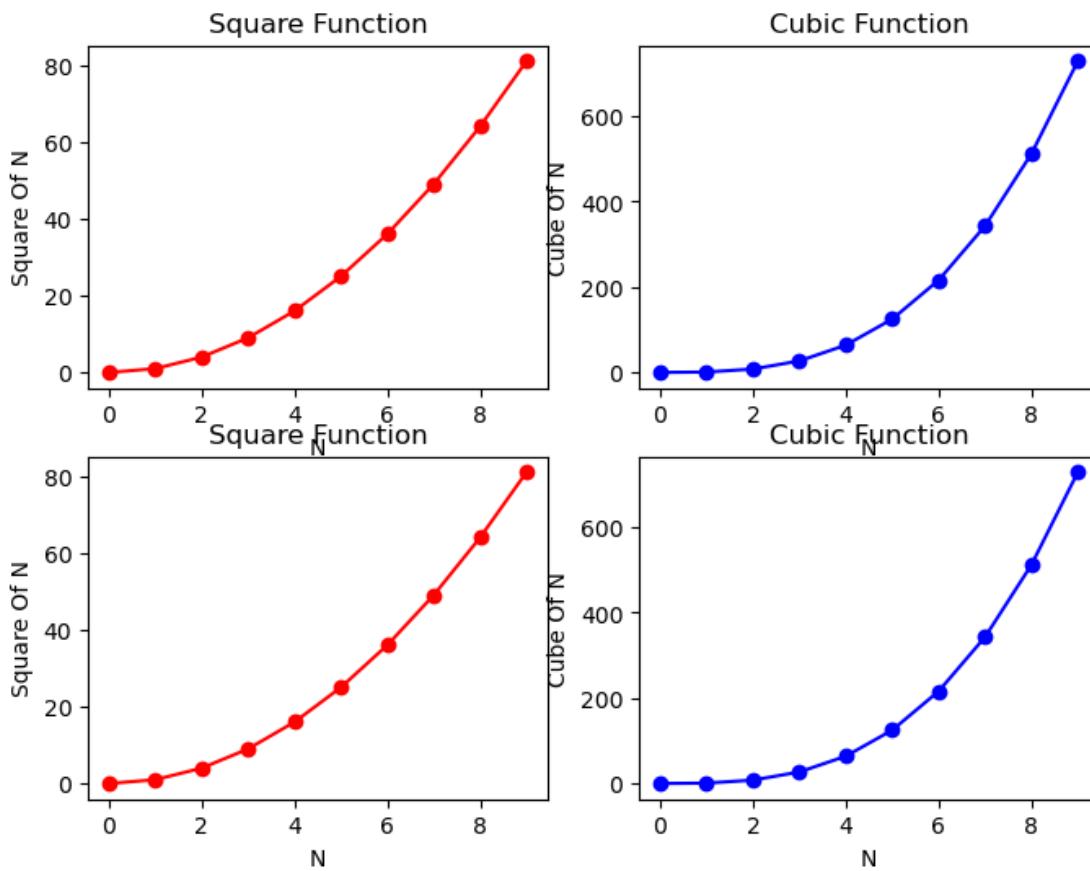
```

ax3 = plt.subplot(2,2,3)
ax3.plot(a,b,color='r',marker='o')
ax3.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax4 = plt.subplot(2,2,4)
ax4.plot(a,c,color='b',marker='o')
ax4.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.show()

```



### Note

- ✓ In the above figure the title and xlabel are overlapping.
- ✓ To overcome this type of problem we can use **plt.tight\_layout()**

---

In [206]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

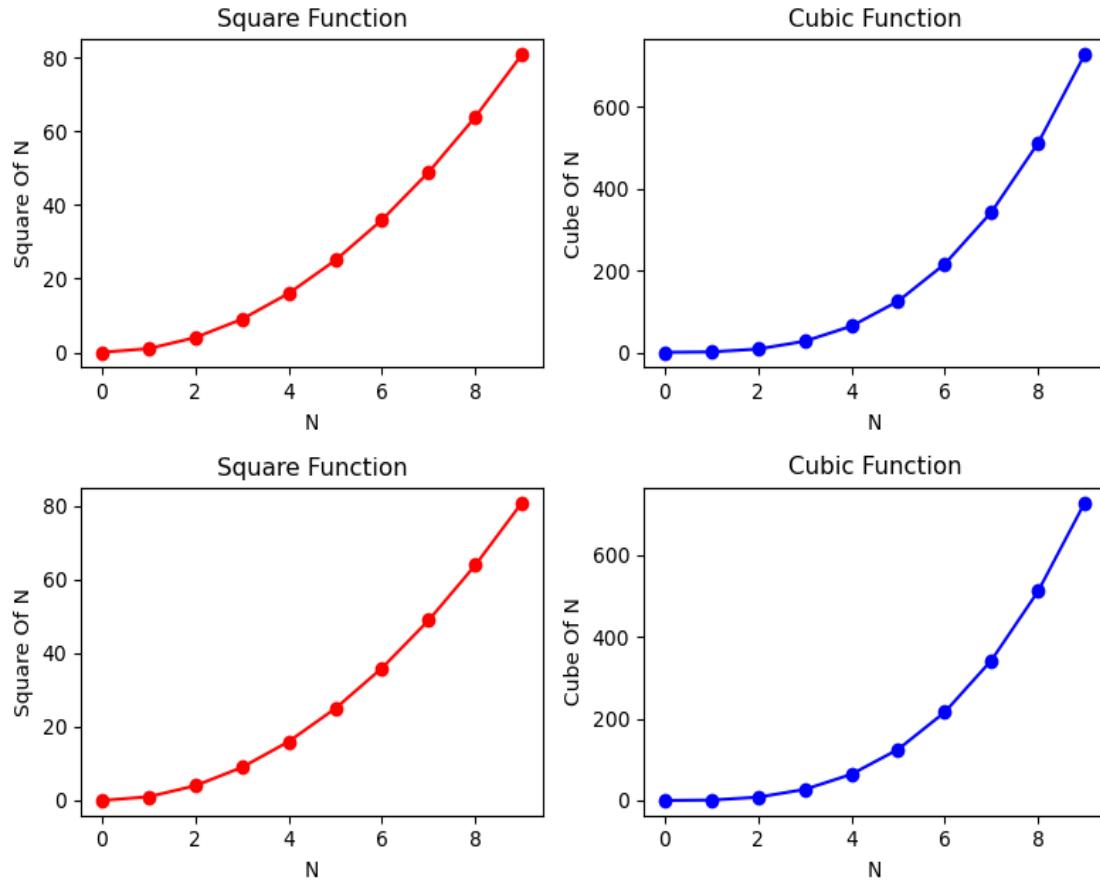
fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(2,2,1)
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax2 = plt.subplot(2,2,2)
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

ax3 = plt.subplot(2,2,3)
ax3.plot(a,b,color='r',marker='o')
ax3.set(xlabel='N',ylabel='Square Of N',title='Square Function')

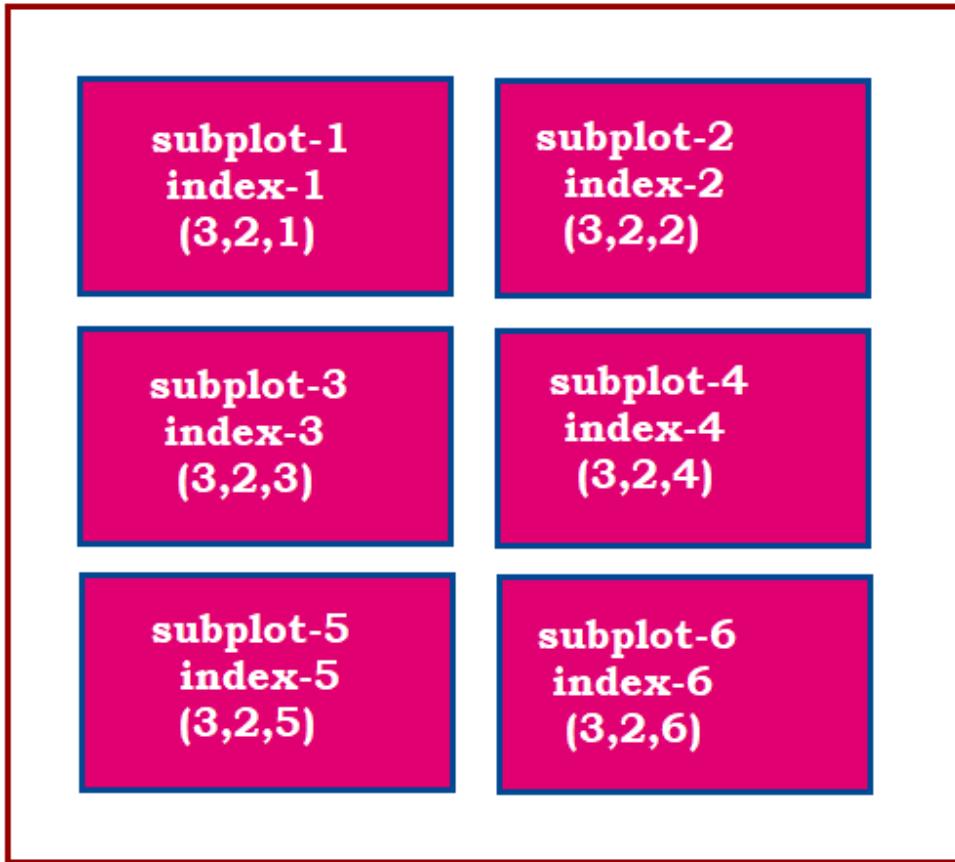
ax4 = plt.subplot(2,2,4)
ax4.plot(a,c,color='b',marker='o')
ax4.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.tight_layout()
plt.show()
```



---

**create 6 subplots and as shown in the figure**



In [207]:

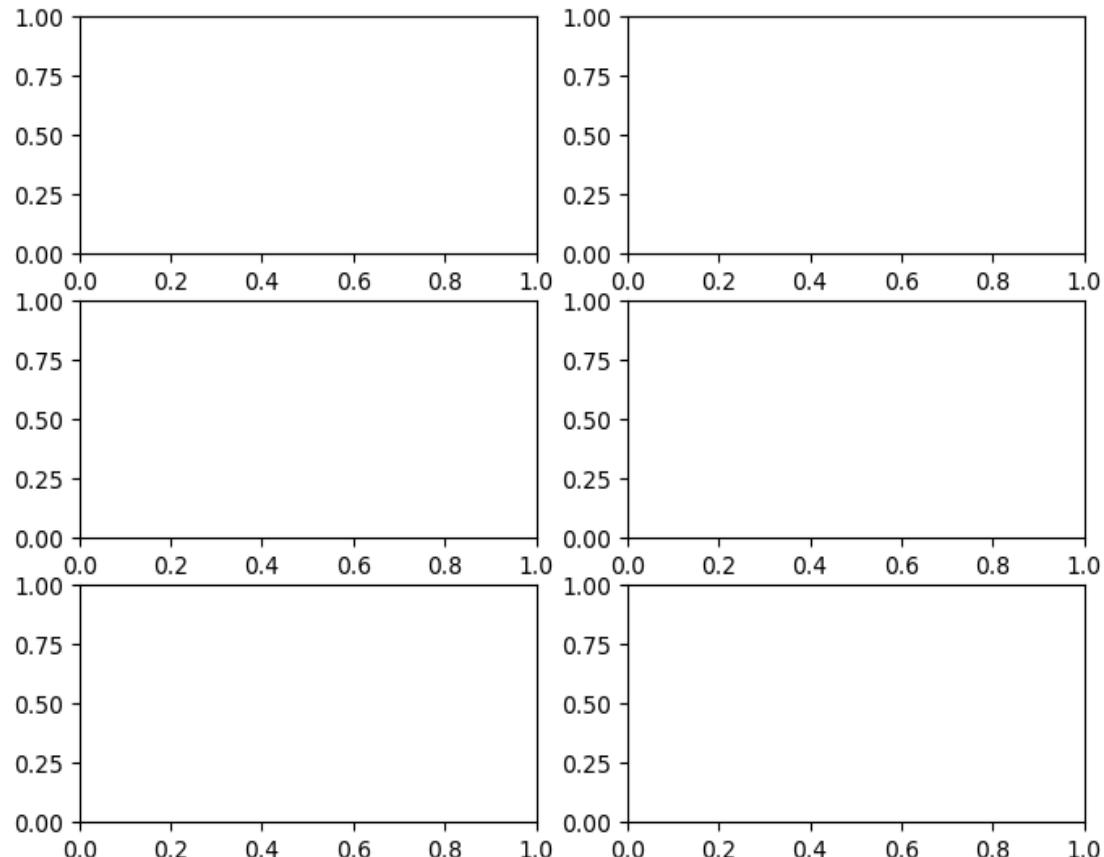
```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(3,2,1)
ax2 = plt.subplot(3,2,2)
```

---

```
ax3 = plt.subplot(3,2,3)
ax4 = plt.subplot(3,2,4)
ax5 = plt.subplot(3,2,5)
ax6 = plt.subplot(3,2,6)

plt.show()
```



---

In [208]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(3,2,1)
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax2 = plt.subplot(3,2,2)
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

ax3 = plt.subplot(3,2,3)
ax3.plot(a,b,color='r',marker='o')
ax3.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax4 = plt.subplot(3,2,4)
ax4.plot(a,c,color='b',marker='o')
ax4.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

ax5 = plt.subplot(3,2,5)
ax5.plot(a,b,color='r',marker='o')
ax5.set(xlabel='N',ylabel='Square Of N',title='Square Function')

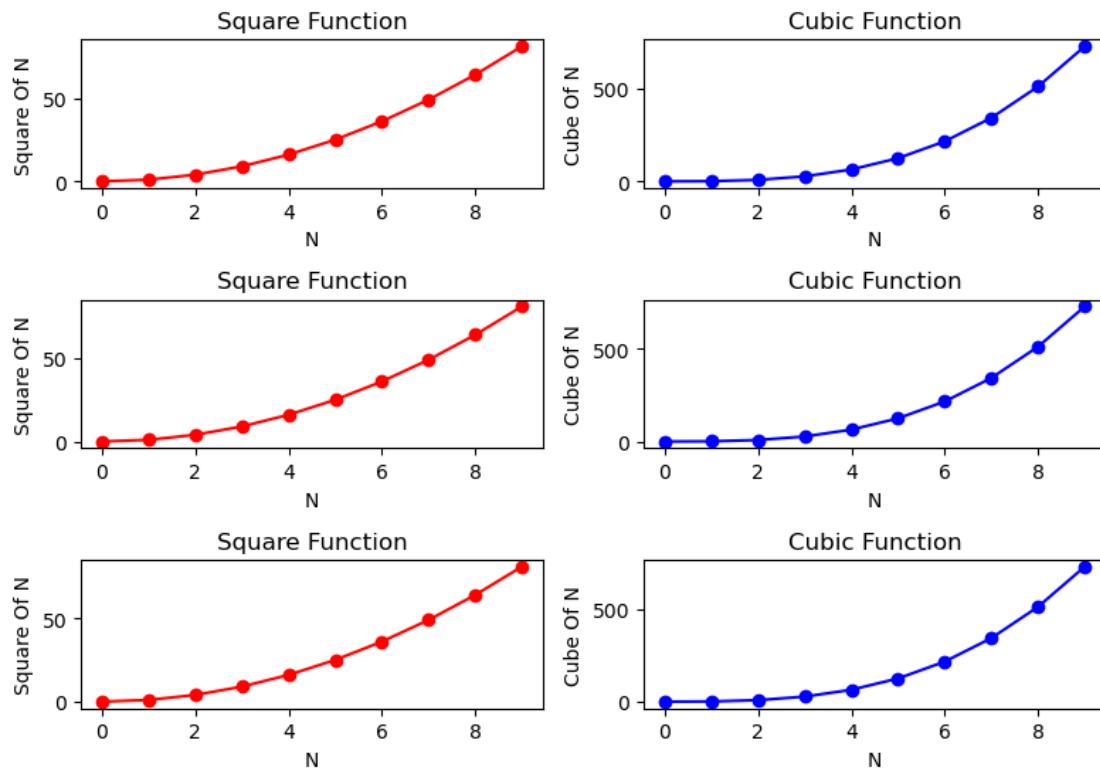
ax6 = plt.subplot(3,2,6)
ax6.plot(a,c,color='b',marker='o')
ax6.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.suptitle('One Figure But six plots',color='r',size=15)
plt.tight_layout()

plt.show()
```

---

## One Figure But six plots



### Note

**1. plt.suptitle('One Figure But six plots',color='r',size=15)**  
To add a centered super title for the figure

**2. plt.tight\_layout()**  
If we are not using this, may be xlabel and title are overlapping.

**3. ax1 = plt.subplot(3,2,1)**  
Instead of this, we can also take like  
ax1 = plt.subplot(321) ➔ means we can remove comma.  
ax1 = plt.subplot(111) #default value

---

## **PROGRAM WITH More realistic Titles:**

In [209]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig = plt.figure(figsize=(8,6),num=1)
ax1 = plt.subplot(3,2,1)
ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Enquiry Report')

ax2 = plt.subplot(3,2,2)
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Sales Report')

ax3 = plt.subplot(3,2,3)
ax3.plot(a,b,color='r',marker='o')
ax3.set(xlabel='N',ylabel='Square Of N',title='Faculty Report')

ax4 = plt.subplot(3,2,4)
ax4.plot(a,c,color='b',marker='o')
ax4.set(xlabel='N',ylabel='Cube Of N',title='Fees Report')

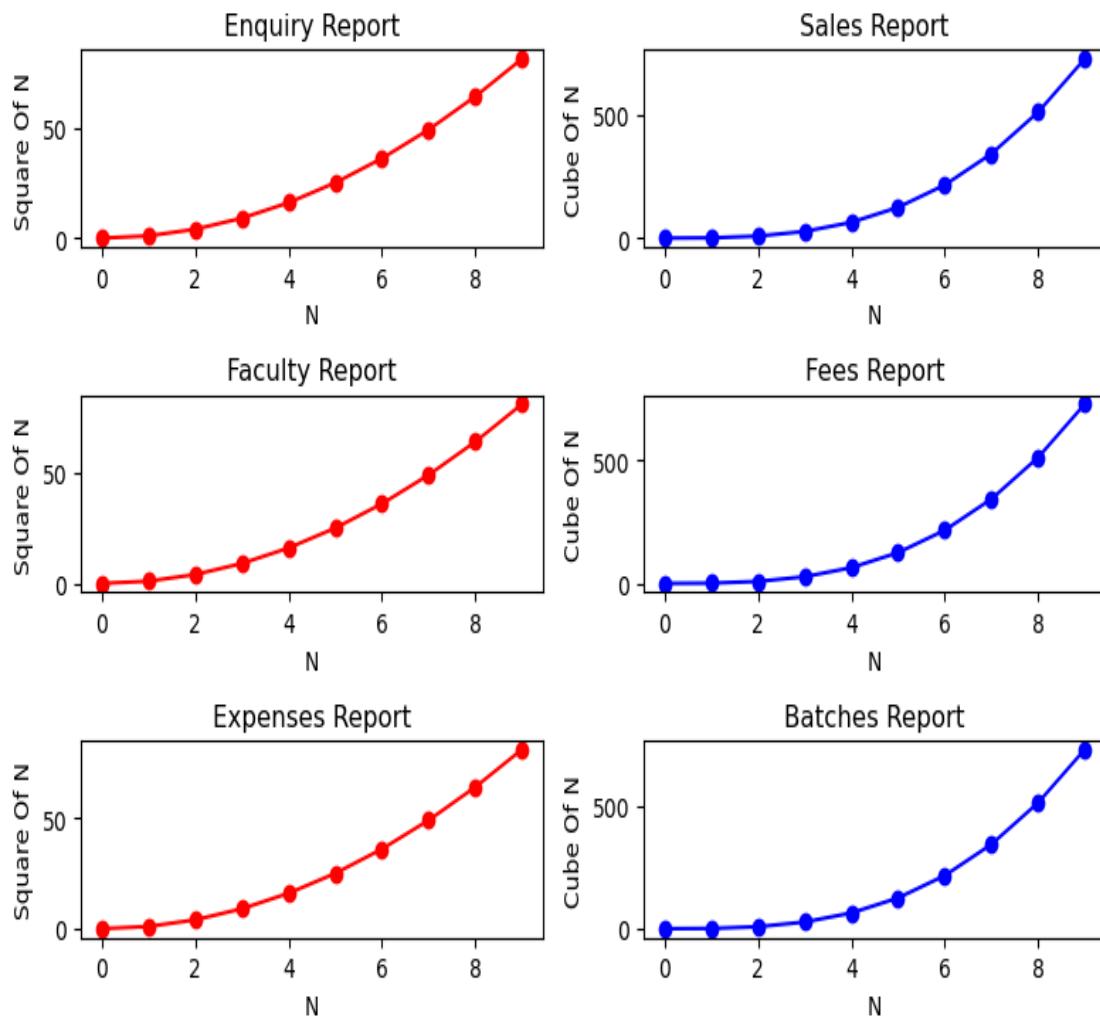
ax5 = plt.subplot(3,2,5)
ax5.plot(a,b,color='r',marker='o')
ax5.set(xlabel='N',ylabel='Square Of N',title='Expenses Report')

ax6 = plt.subplot(3,2,6)
ax6.plot(a,c,color='b',marker='o')
ax6.set(xlabel='N',ylabel='Cube Of N',title='Batches Report')

plt.suptitle('DURGASOFT REPORTS',color='r',size=15)
plt.tight_layout()

plt.show()
```

## DURGASOFT REPORTS



### Problem with subplot() approach:

- ✓ For every subplot we have to call subplot() function separately.
- ✓ If we want to create large number of subplots, then it will become difficult.
- ✓ To overcome this problem we should go for **subplots() function**, which returns all axes objects at a time.

---

## **pyplot.subplots() function**

In [210]:

```
import matplotlib.pyplot as plt  
help(plt.subplots)
```

Help on function subplots in module matplotlib.pyplot:

```
subplots(nrows=1, ncols=1, *, sharex=False, sharey=False, squeeze=True,  
subplot_kw=None, gridspec_kw=None, **fig_kw)
```

Create a figure and a set of subplots.

### **Usage**

#### **1. fig,ax = plt.subplots()**

It returns one figure and one axes object

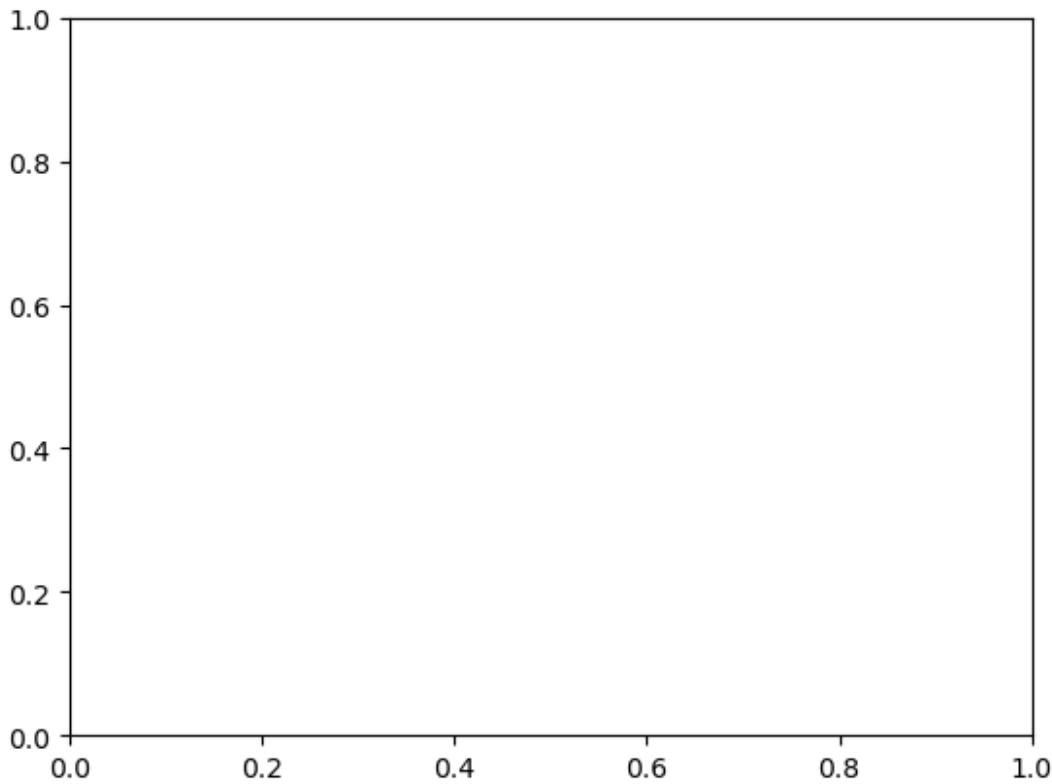
#### **2. fig,axs = plt.subplots(2,2)**

It returns one figure object and an ndarray of 4 axes objects.

In [211]:

```
# one figure object and one axes object  
import matplotlib.pyplot as plt  
import numpy as np  
a = np.arange(10)  
b = a**2  
c = a**3  
  
fig,ax = plt.subplots()  
print(fig)  
print(ax)
```

Figure(640x480)  
AxesSubplot(0.125,0.11;0.775x0.77)



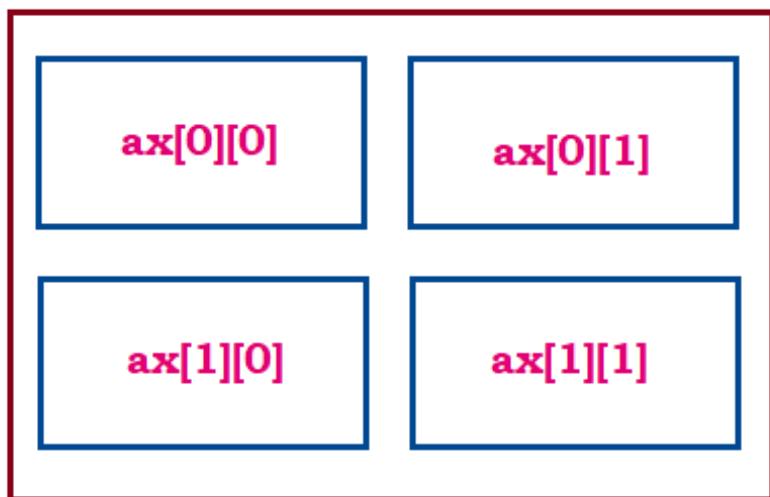
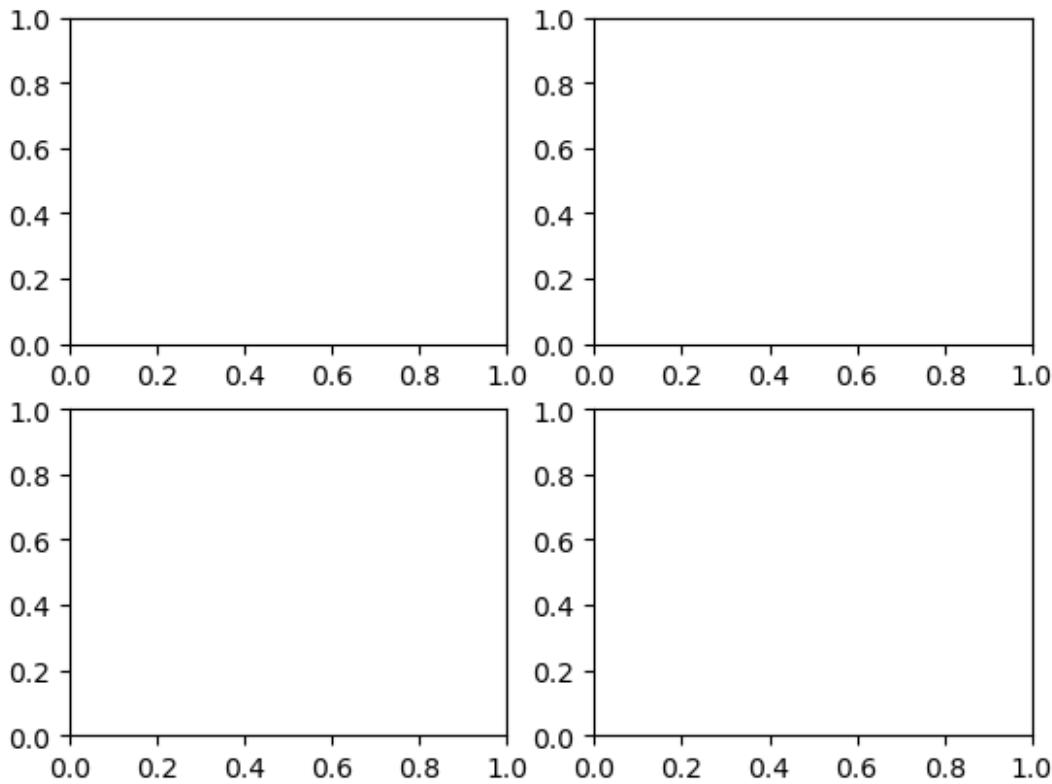
```
In [212]:
```

```
# one figure object and multiple axes objects
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig,axs = plt.subplots(2,2)
print(fig)
print(axs)
```

Figure(640x480)

```
[[<AxesSubplot:> <AxesSubplot:>]
 [<AxesSubplot:> <AxesSubplot:>]]
```



---

In [213]:

```
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig,ax = plt.subplots(3,2)

ax[0][0].plot(a,b,color='r',marker='o')
ax[0][0].set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax[0][1].plot(a,c,color='b',marker='o')
ax[0][1].set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

ax[1][0].plot(a,b,color='r',marker='o')
ax[1][0].set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax[1][1].plot(a,c,color='b',marker='o')
ax[1][1].set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

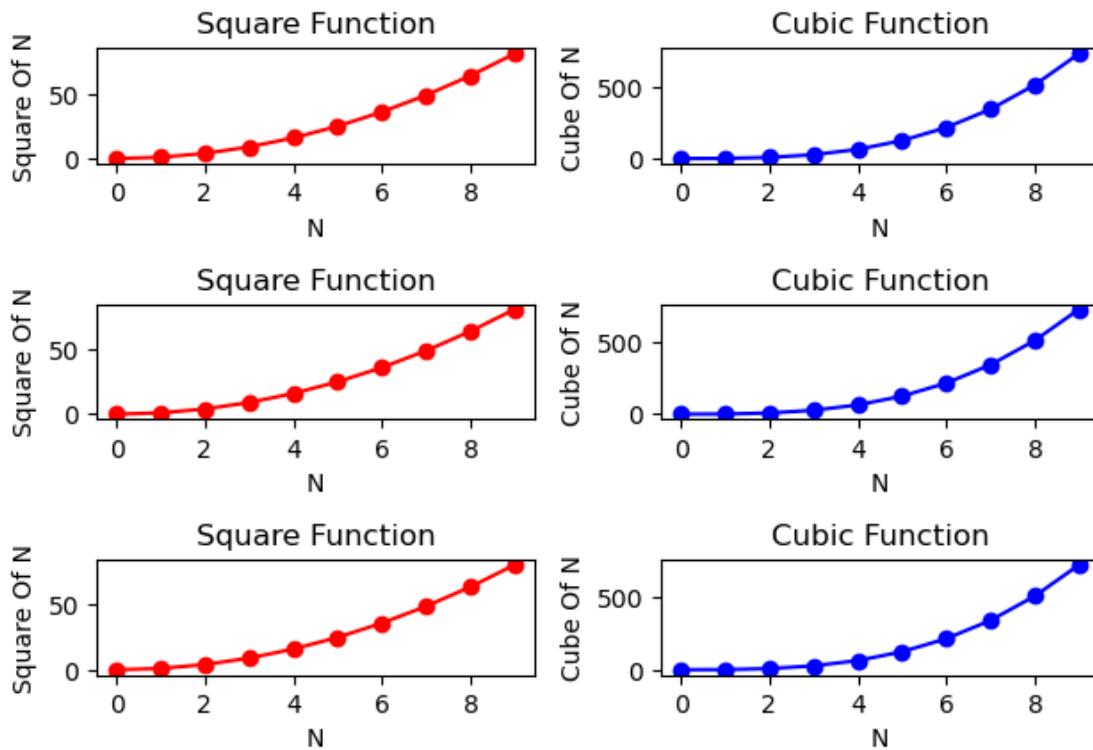
ax[2][0].plot(a,b,color='r',marker='o')
ax[2][0].set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax[2][1].plot(a,c,color='b',marker='o')
ax[2][1].set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.suptitle('One Figure But six plots',color='r',size=15)
plt.tight_layout()

plt.show()
```

## One Figure But six plots



### Note

Here we have to call **subplots()** function only once

In [214]:

```
# short-cut way
import matplotlib.pyplot as plt
import numpy as np
a = np.arange(10)
b = a**2
c = a**3

fig,((ax1,ax2),(ax3,ax4),(ax5,ax6)) = plt.subplots(3,2)

ax1.plot(a,b,color='r',marker='o')
ax1.set(xlabel='N',ylabel='Square Of N',title='Square Function')
```

```
ax2.plot(a,c,color='b',marker='o')
ax2.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

ax3.plot(a,b,color='r',marker='o')
ax3.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax4.plot(a,c,color='b',marker='o')
ax4.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

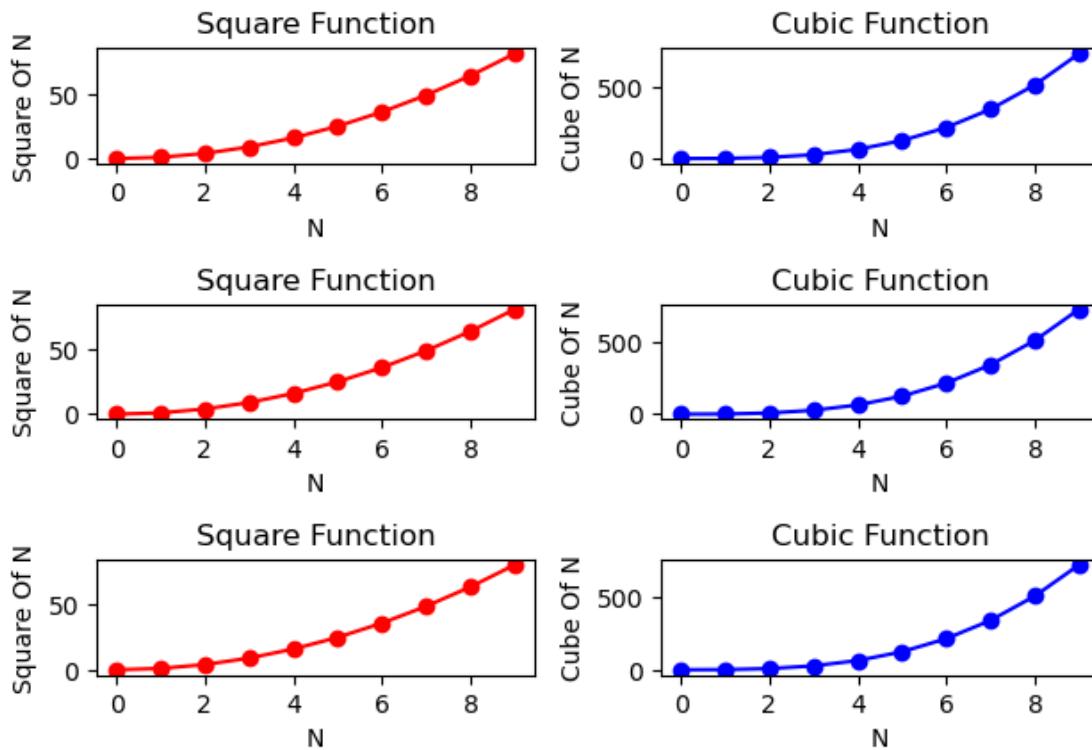
ax5.plot(a,b,color='r',marker='o')
ax5.set(xlabel='N',ylabel='Square Of N',title='Square Function')

ax6.plot(a,c,color='b',marker='o')
ax6.set(xlabel='N',ylabel='Cube Of N',title='Cubic Function')

plt.suptitle('One Figure But six plots',color='r',size=15)
plt.tight_layout()

plt.show()
```

## One Figure But six plots



### Note

#### Shortcut way of calling subplots

```
fig,((ax1,ax2),(ax3,ax4),(ax5,ax6)) = plt.subplots(3,2)
```

In [215]:

```
# Demo program for different types of subplots:  
  
import matplotlib.pyplot as plt  
import numpy as np  
fig,((ax1,ax2),(ax3,ax4)) = plt.subplots(2,2)  
  
# setting the figure size  
fig.set_size_inches(10,6)
```

---

```

#line plot creation
x = np.arange(5)
y = x**2
ax1.plot(x,y,'ro-')
ax1.set(xlabel='N Value',ylabel='Square Of N',title='Line Plot')

#scatter plot creation
x = np.arange(5)
y = x**2
ax2.scatter(x,y,s=300,c=[0,25,50,75,100],cmap='prism')
ax2.set(xlabel='N Value',ylabel='Square Of N',title='Scatter Plot')

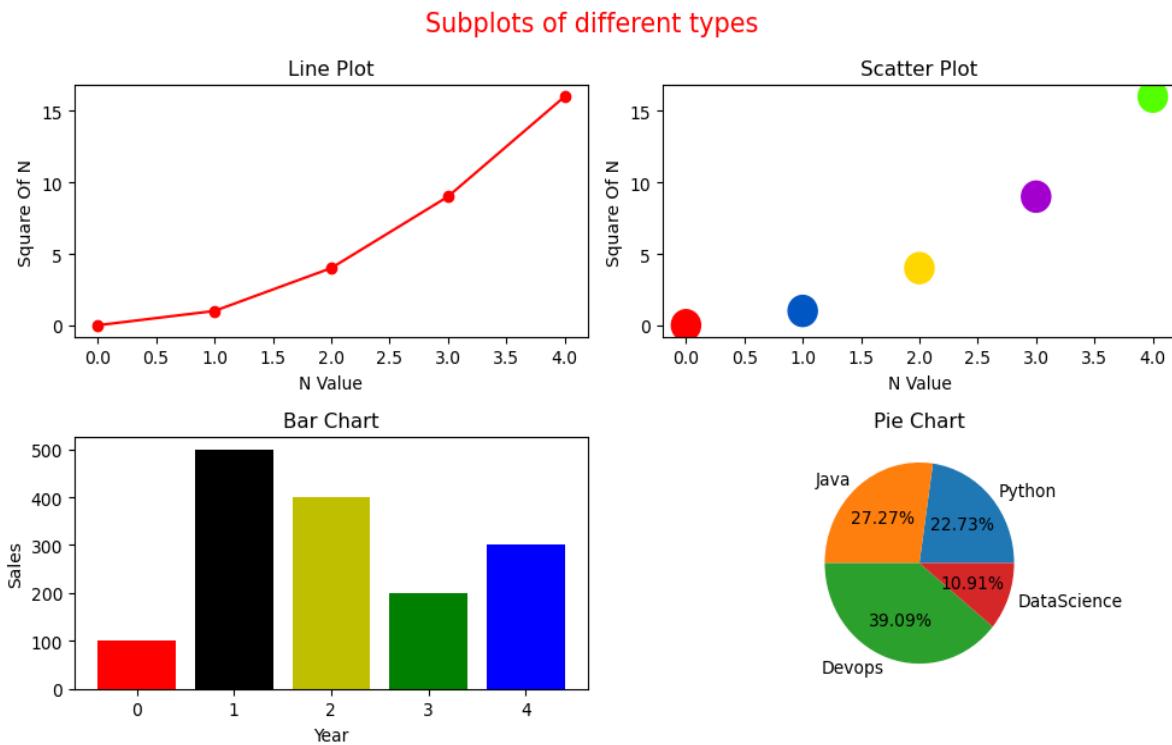
#Bar Chart Creation
x = np.arange(5)
y = [100,500,400,200,300]
ax3.bar(x,y,color=['r','k','y','g','b'])
ax3.set(xlabel='Year',ylabel='Sales',title='Bar Chart')

#Pie Chart Creation
marks = np.array([25,30,43,12])
mylabels = ['Python','Java','Devops','DataScience']
ax4.pie(marks,labels = mylabels,autopct='%.2f%%')
ax4.set(title='Pie Chart')

plt.suptitle('Subplots of different types',color='r',size=15)
plt.tight_layout()

plt.show()

```



## Note

Setting the figure size using

### 1. By specifying figsize parameter while calling the figure() function

- ✓ plt.figure(figsize=(8,6),num=1) ==> width:8 height:6 in inches
- ✓ help(matplotlib.pyplot.figure)

```
figure(num=None, figsize=None, dpi=None, facecolor=None, edgecolor=None,
frameon=True, FigureClass=<class 'matplotlib.figure.Figure'>, clear=False,
**kwargs)
```

Create a new figure, or activate an existing figure.

### 2. calling set\_size\_inches method on the figure object

- ✓ fig,ax = plt.subplots()
- ✓ By this we can get the figure and axes objects  
fig.set\_size\_inches(8,6) ➔ sets the size of the figure

---

```
✓ help(matplotlib.figure.Figure.set_size_inches)
    set_size_inches(self, w, h=None, forward=True)
        Set the figure size in inches.
```

Call signatures::

```
fig.set_size_inches(w, h) # OR
fig.set_size_inches((w, h))
```

In [216]:

**# Another example from matplotlib documentation:**

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

n_bins = 10
x = np.random.randn(1000, 3)

fig, ((ax0, ax1), (ax2, ax3)) = plt.subplots(nrows=2, ncols=2)

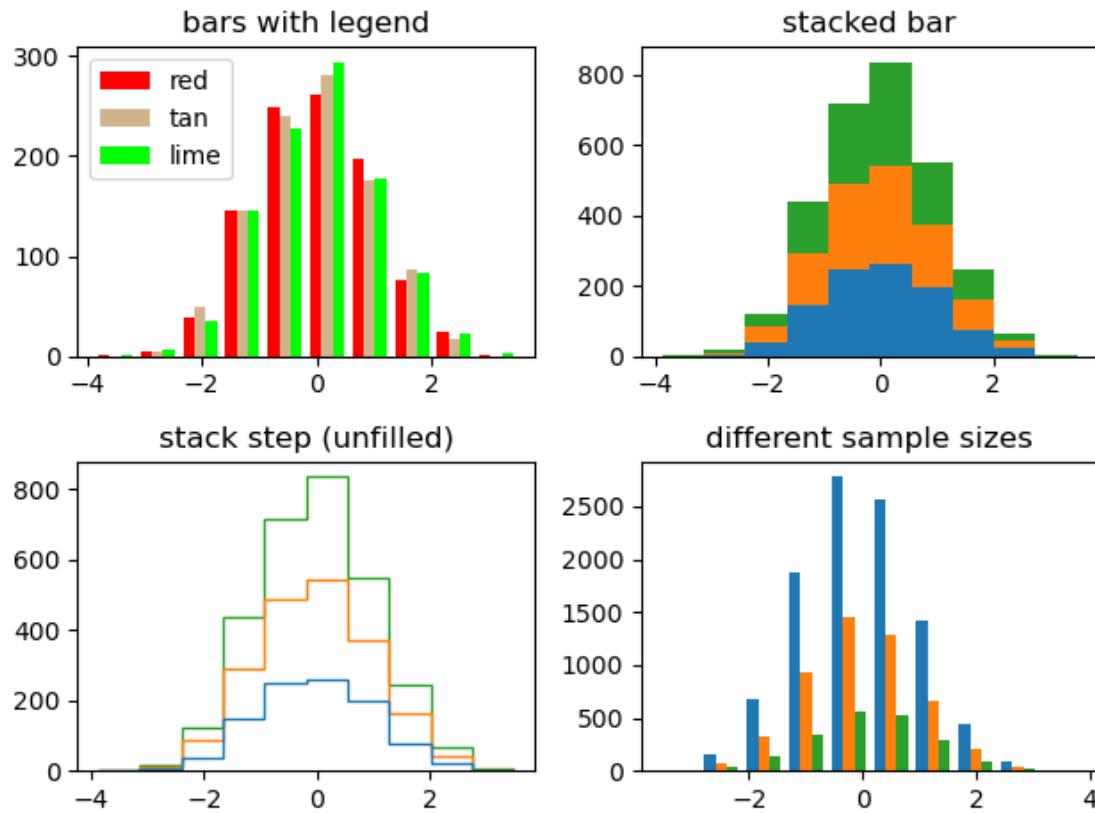
colors = ['red', 'tan', 'lime']
ax0.hist(x, n_bins, histtype='bar', color=colors, label=colors)
ax0.legend(prop={'size': 10})
ax0.set_title('bars with legend')

ax1.hist(x, n_bins, histtype='bar', stacked=True)
ax1.set_title('stacked bar')

ax2.hist(x, n_bins, histtype='step', stacked=True, fill=False)
ax2.set_title('stack step (unfilled)')

# Make a multiple-histogram of data-sets with different length.
x_multi = [np.random.randn(n) for n in [10000, 5000, 2000]]
ax3.hist(x_multi, n_bins, histtype='bar')
ax3.set_title('different sample sizes')

fig.tight_layout()
plt.show()
```



### Note

We can create subplots in 3 ways

- ✓ Manually by adding axes → `fig.add_axes([0.1,0.1,0.8,0.8]) #[l,b,w,h]`
- ✓ By using `plt.subplot()`
- ✓ By using `plt.subplots()`

---

## Chapter-16

### Plotting Geographic Data with Basemap

#### **Plotting Geographic data with Basemap:**

- ✓ If we want to plot geographic locations like world map or india map etc, then we should go for Basemap.
- ✓ Basemap is matplotlib extension and it is not available bydefault with matplotlib, we have to install separately.

#### **How to find python's Home directory?**

##### **1<sup>st</sup> way:**

- ✓ From the command prompt execute the following command  
D:\durgaclasses>where python  
C:\Python38

##### **2<sup>nd</sup> way:**

- ✓ By using system's environment variable :PATH  
This PC--->right click-->Properties--->Advanced System Settings-->Environment Variables-->Path-->Edit  
C:\Python38\

#### How to check python's version?

D:\durgaclasses>python --version or D:\durgaclasses>python -V  
Python 3.8.6

#### **How to install basemap**

- ✓ There are multiple ways are there for installation
- ✓ official website for basemap → <https://matplotlib.org/basemap/>

#### **Step-1: Installation of pyproj library:**

- ✓ Without pyproj, basemap won't work. Hence before installing basemap we have to install pyproj. i.e., pyproj is the dependent library for basemap.
- ✓ Download pyproj installation wheel from the link → <https://www.lfd.uci.edu/~gohlke/pythonlibs/#pyproj>

- ✓ While downloading we have to choose proper wheel based on your platform and python version.
- ✓ We have to download the following wheel( windows 64, python 3.8) → **pyproj-3.1.0-cp38-cp38-win\_amd64.whl**
- ✓ Copy this wheel in python's home directory(C:\Python38)
- ✓ In the command prompt, from this location execute the following command→

**C:\Python38>pip install pyproj-3.1.0-cp38-cp38-win\_amd64.whl**

Processing c:\python38\pyproj-3.1.0-cp38-cp38-win\_amd64.whl

Requirement already satisfied: certifi in c:\python38\lib\site-packages (from pyproj==3.1.0) (2020.12.5) pyproj is already installed with the same version as the provided wheel. Use --force-reinstall to force an installation of the wheel.

← → C ⌂ lfd.uci.edu/~gohlke/pythonlibs/#pyproj

**Pyproj:** an interface to the PROJ library for cartographic transformations.

[pyproj-3.1.0-pp37-pypy37\\_pp73-win\\_amd64.whl](#)  
[pyproj-3.1.0-cp310-cp310-win\\_amd64.whl](#)  
[pyproj-3.1.0-cp310-cp310-win32.whl](#)  
[pyproj-3.1.0-cp39-cp39-win\\_amd64.whl](#)  
[pyproj-3.1.0-cp39-cp39-win32.whl](#)  
**[pyproj-3.1.0-cp38-cp38-win\\_amd64.whl](#)**  
[pyproj-3.1.0-cp38-cp38-win32.whl](#)  
[pyproj-3.1.0-cp37-cp37m-win\\_amd64.whl](#)  
[pyproj-3.1.0-cp37-cp37m-win32.whl](#)  
[pyproj-3.0.1-cp39-cp39-win\\_amd64.whl](#)  
[pyproj-3.0.1-cp39-cp39-win32.whl](#)  
[pyproj-3.0.1-cp38-cp38-win\\_amd64.whl](#)  
[pyproj-3.0.1-cp38-cp38-win32.whl](#)  
[pyproj-3.0.1-cp37-cp37m-win\\_amd64.whl](#)  
[pyproj-3.0.1-cp37-cp37m-win32.whl](#)  
[pyproj-3.0.1-cp36-cp36m-win\\_amd64.whl](#)  
[pyproj-3.0.1-cp36-cp36m-win32.whl](#)

---

## **Step-2: Installation of basemap library**

- ✓ It is exactly same as step-1 except that wheel name will be changed.

**<https://www.lfd.uci.edu/~gohlke/pythonlibs/#basemap>**

- ✓ basemap-1.2.2-cp38-cp38-win\_amd64.whl
- ✓ Copy this wheel to python's home directory and execute the following command from that location→

**C:\Python38>pip install basemap-1.2.2-cp38-cp38-win\_amd64.whl**

```
C:\Python38>pip install basemap-1.2.2-cp38-cp38-win_amd64.whl
Processing c:\python38\basemap-1.2.2-cp38-cp38-win_amd64.whl
Requirement already satisfied: pyproj>=1.9.3 in c:\python38\lib\site-packages
(from basemap==1.2.2) (3.1.0) Requirement already satisfied:
matplotlib!=3.0.1,>=1.0.0 in c:\python38\lib\site-packages (from
basemap==1.2.2) (3.4.2) Requirement already satisfied: numpy>=1.2.1 in
c:\python38\lib\site-packages (from basemap==1.2.2) (1.20.2) Requirement
already satisfied: pyshp>=1.2.0 in c:\python38\lib\site-packages (from
basemap==1.2.2) (2.1.3)
```



lfd.uci.edu/~gohlke/pythonlibs/#basemap

**Basemap:** a matplotlib toolkit for plotting 2D data on maps based on GEOS.  
Requires `pyproj`.

[basemap-1.2.2-pp37-pypy37\\_pp73-win\\_amd64.whl](#)  
[basemap-1.2.2-cp310-cp310-win\\_amd64.whl](#)  
[basemap-1.2.2-cp310-cp310-win32.whl](#)  
[basemap-1.2.2-cp39-cp39-win\\_amd64.whl](#)  
[basemap-1.2.2-cp39-cp39-win32.whl](#)  
[basemap-1.2.2-cp38-cp38-win\\_amd64.whl](#)  
[basemap-1.2.2-cp38-cp38-win32.whl](#)  
[basemap-1.2.2-cp37-cp37m-win\\_amd64.whl](#)  
[basemap-1.2.2-cp37-cp37m-win32.whl](#)  
[basemap-1.2.2-cp36-cp36m-win\\_amd64.whl](#)  
[basemap-1.2.2-cp36-cp36m-win32.whl](#)  
[basemap-1.2.1-cp35-cp35m-win\\_amd64.whl](#)  
[basemap-1.2.1-cp35-cp35m-win32.whl](#)  
[basemap-1.2.1-cp27-cp27m-win\\_amd64.whl](#)  
[basemap-1.2.1-cp27-cp27m-win32.whl](#)  
[basemap-1.2.0-cp34-cp34m-win\\_amd64.whl](#)  
[basemap-1.2.0-cp34-cp34m-win32.whl](#)

### Step-3: How to check installation:

D:\durgaclasses>py

Python 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v.1927 64 bit (AMD64)] on win32

Type "help", "copyright", "credits" or "license" for more information.

>>> from mpl\_toolkits.basemap import Basemap

In [217]:

```
from mpl_toolkits import basemap  
print(basemap.__version__)
```

1.2.2+dev

---

In [218]:

```
# Demo program to get basic idea:
```

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
m = Basemap(projection='mill')
m.drawcoastlines()
plt.show()
```



---

## Important Theoretical Terminology:

### 1. projection

- ✓ To represent the curved surface of the earth on a two-dimensional map, a map projection is required.
- ✓ Basemap package provides 24 different map projections. Some are global and some are specific to a particular portion of the globe.
- ✓ The supported projections are →  
<https://matplotlib.org/basemap/users/index.html>

Azimuthal Equidistant Projection  
Gnomonic Projection  
Orthographic Projection  
Geostationary Projection  
Near-Sided Perspective Projection  
Mollweide Projection  
Hammer Projection  
Robinson Projection  
Eckert IV Projection  
Kavrayskiy VII Projection  
McBryde-Thomas Flat Polar Quartic  
Sinusoidal Projection  
Equidistant Cylindrical Projection  
Cassini Projection  
Mercator Projection  
Transverse Mercator Projection  
Oblique Mercator Projection  
Polyconic Projection  
Miller Cylindrical Projection  
Gall Stereographic Projection  
Cylindrical Equal-Area Projection  
Lambert Conformal Projection  
Lambert Azimuthal Equal Area Projection

---

Stereographic Projection  
Equidistant Conic Projection  
Albers Equal Area Projection  
Polar Stereographic Projection  
Polar Lambert Azimuthal Projection  
Polar Azimuthal Equidistant Projection  
van der Grinten Projection

## 2. Resolution:

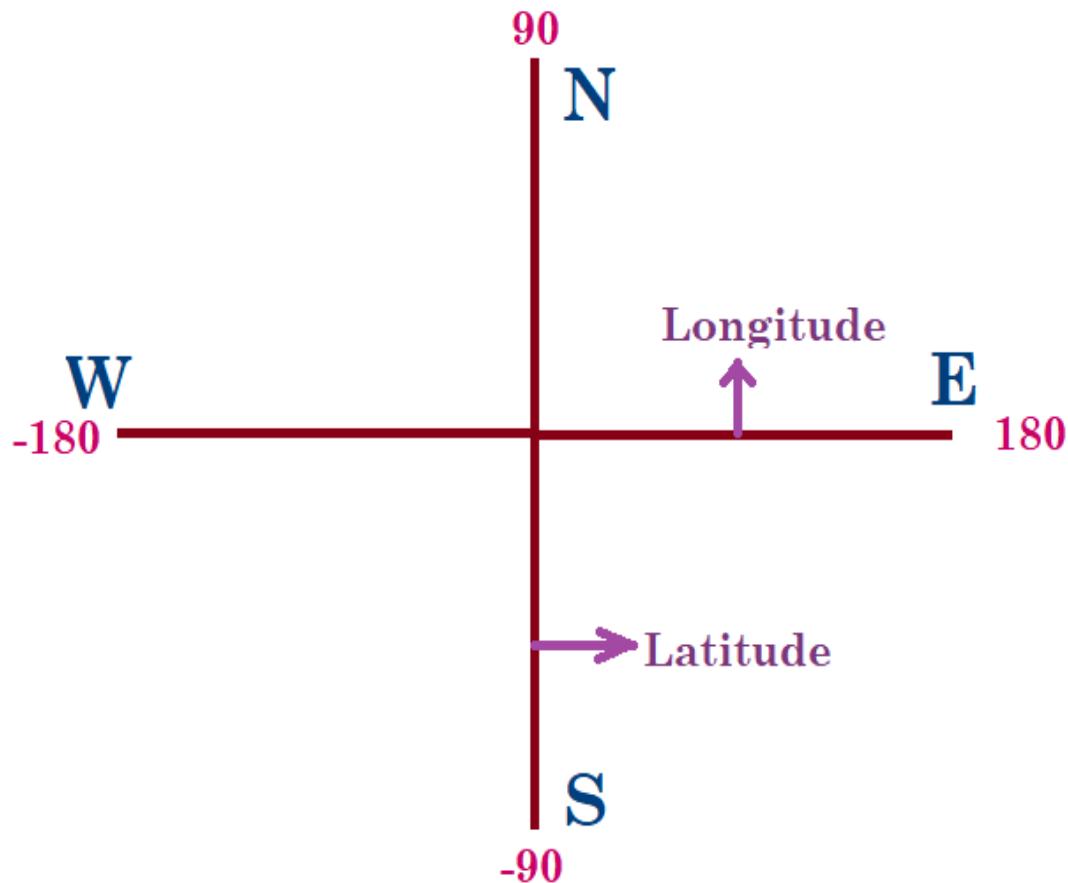
- ✓ Resolution is the number of pixels to represent the graph. More resolution means more clarity.
- ✓ Basemap supports the following resolutions:
  1. c → crude
  2. l → low
  3. i → intermediate
  4. h → high
  5. f → full

**Note:** It is not recommended to use higher level resolutions as it takes more time to load graph

## 3. Latitude and Longitude:

- ✓ Every location on the earth has a global address. This **global address** is represented by two coordinates, which are nothing but latitude and longitude.
- ✓ By using latitude and longitude, we can identify any geographic location on the globe.
- ✓ **Latitude** is the **angular distance of a point north or south of the equator**. Lines of latitude are called **parallels**. The **range** of latitude is **from -90 to 90**.  
-90 represents **south** where as  
90 represents **north**.

- 
- ✓ **Longitude** is the **angular distance of a point east or west of the prime meridian(Greenwich Meridian)**. Lines of longitude are called **meridians**. The **range** of longitude is from **-180 to 180**.  
**-180 represents west where as  
180 represents east.**
  - ✓ Very easily we can identify longitude and latitude values.



---

## Important methods of Basemap

### 1. Methods for physical boundaries and bodies of water:

- ✓ **m.drawcoastlines()** ➔ To draw continental coast lines
- ✓ **m.drawmapboundary()** ➔ To draw map boundary
- ✓ **m.drawrivers()** ➔ To draw rivers on the map
- ✓ **m.fillcontinents()** ➔ To fill continents with a given color
- ✓ **m.drawlsmask()** ➔ To draw a mask between the land and sea

### 2. Methods for political boundaries:

- ✓ **m.drawcountries()** ➔ Draw country boundaries
- ✓ **m.drawstates()** ➔ Draw US State boundaries

### 3. Methods for Map features:

- ✓ **m.drawparallels()** ➔ To draw lines of latitude
- ✓ **m.drawmeridians()** ➔ To draw lines of longitude

### 4. Methods for whole-globe images:

- ✓ **m.bluemarble()** ➔ To project NASA's blue marble image onto the map.
- ✓ **m.etopo()** ➔ To project an etopo relief image onto the map
- ✓ **m.shadedrelief()** ➔ To project a shaded relief image on to the map.

---

In [219]:

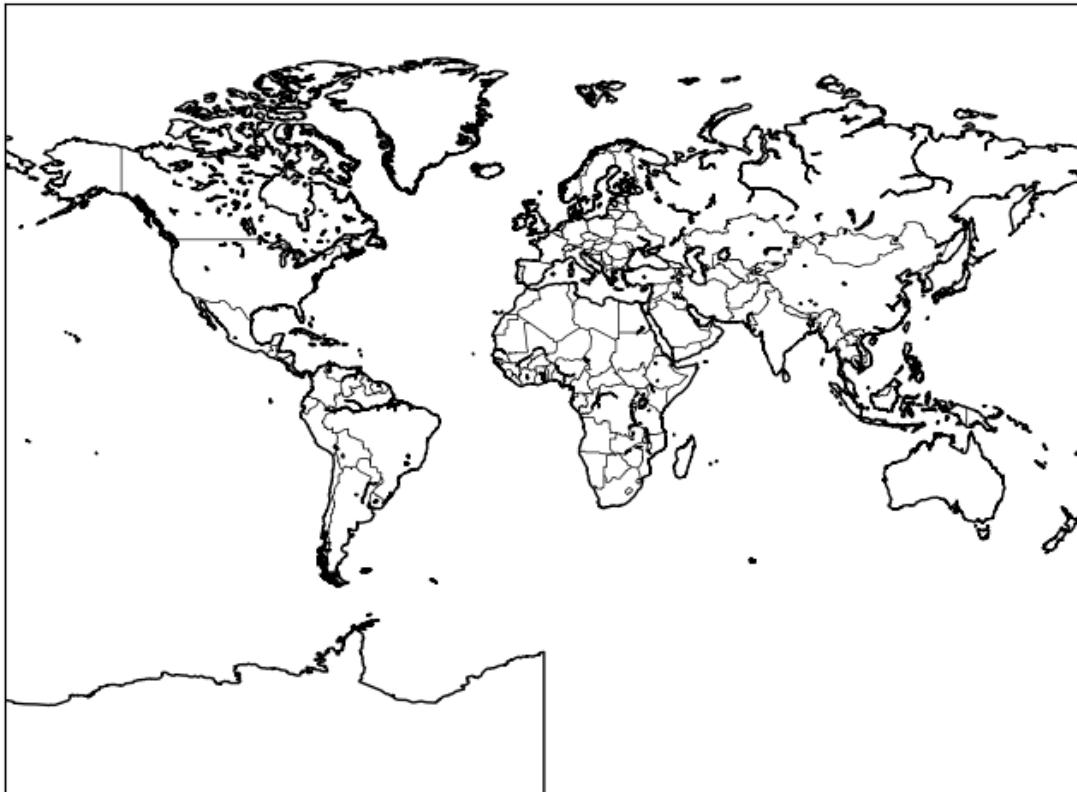
```
# Draw countries. resolution 'crude'  
import matplotlib.pyplot as plt  
from mpl_toolkits.basemap import Basemap  
fig = plt.figure(figsize=(12,6))  
m = Basemap(projection='mill',resolution='c')  
m.drawcoastlines() # to draw continental coast lines  
m.drawcountries()  
plt.show()
```



---

In [220]:

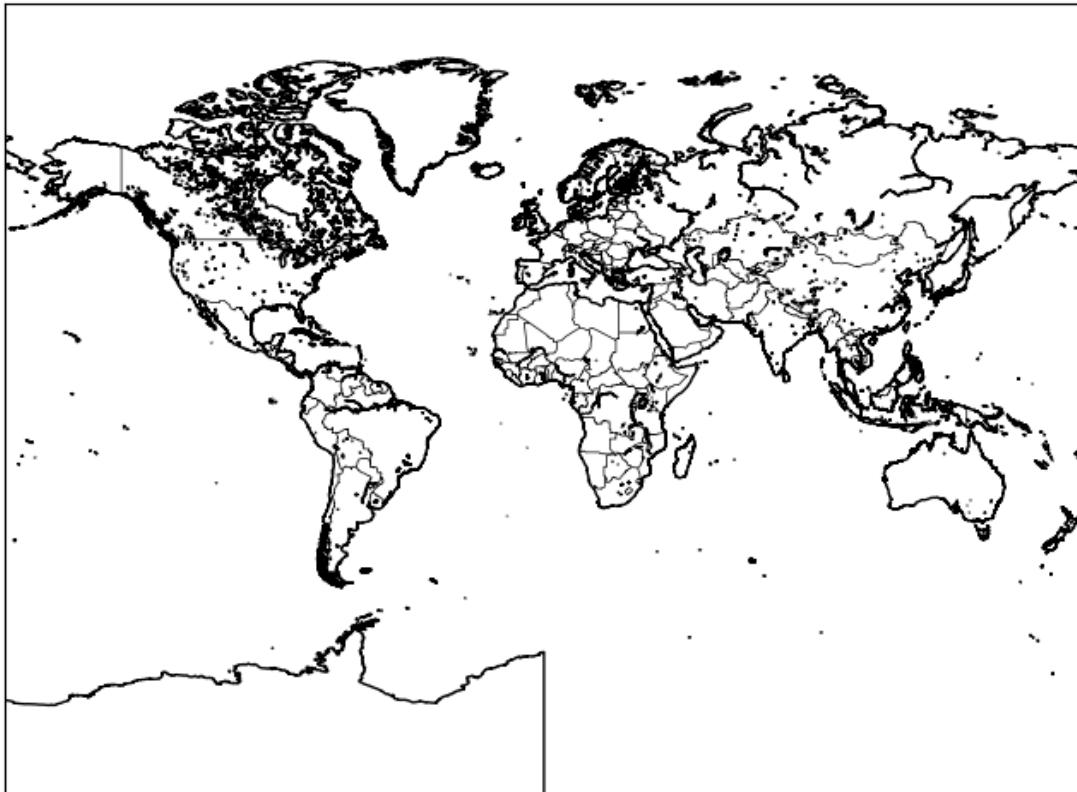
```
# Draw countries. resolution 'low'  
import matplotlib.pyplot as plt  
from mpl_toolkits.basemap import Basemap  
fig = plt.figure(figsize=(12,6))  
m = Basemap(projection='mill',resolution='l')  
m.drawcoastlines() # to draw continental coast lines  
m.drawcountries()  
plt.show()
```



---

In [221]:

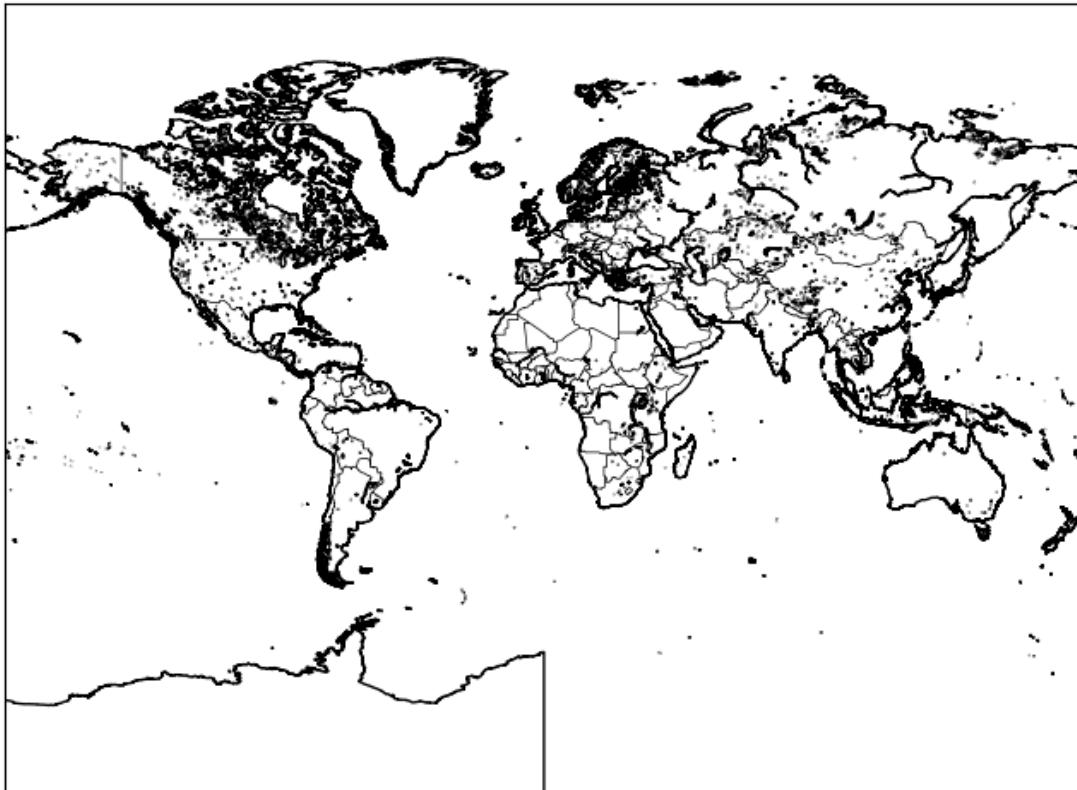
```
# Draw countries. resolution 'intermediate'  
import matplotlib.pyplot as plt  
from mpl_toolkits.basemap import Basemap  
fig = plt.figure(figsize=(12,6))  
m = Basemap(projection='mill',resolution='i')  
m.drawcoastlines() # to draw continental coast lines  
m.drawcountries()  
plt.show()
```



---

In [222]:

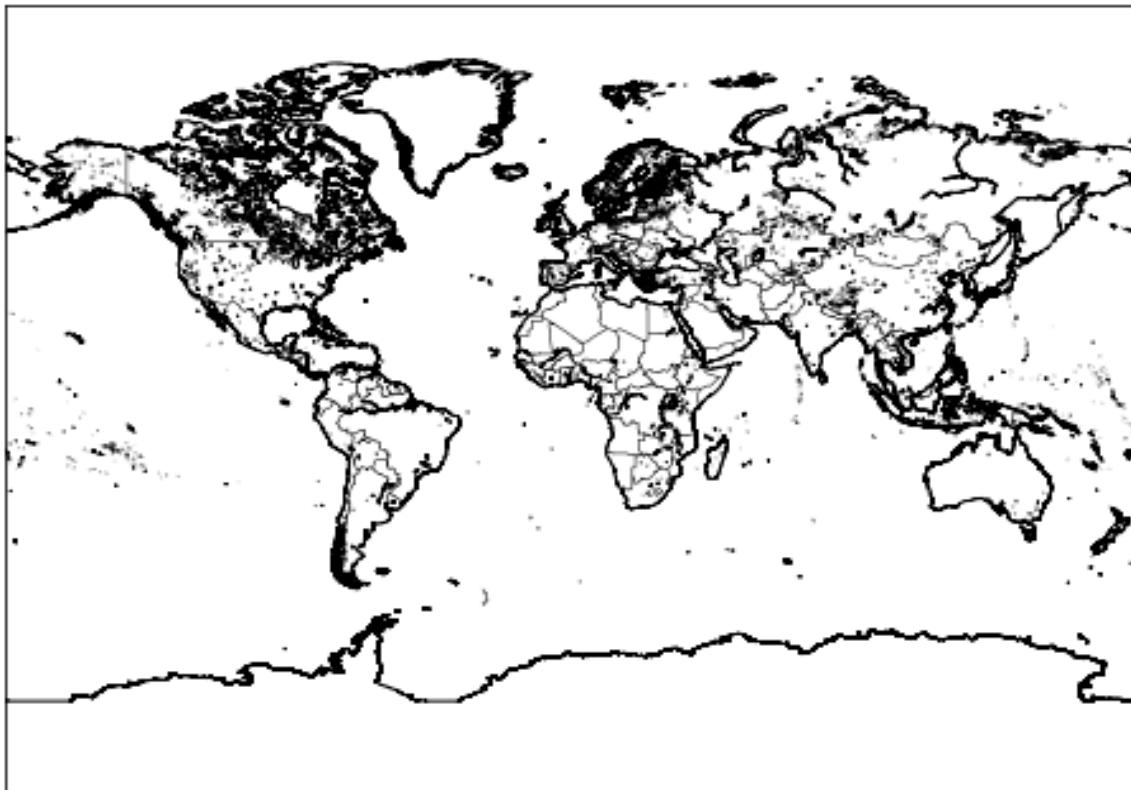
```
# Draw countries. resolution 'high'  
import matplotlib.pyplot as plt  
from mpl_toolkits.basemap import Basemap  
fig = plt.figure(figsize=(12,6))  
m = Basemap(projection='mill',resolution='h')  
m.drawcoastlines() # to draw continental coast lines  
m.drawcountries()  
plt.show()
```



---

In [223]:

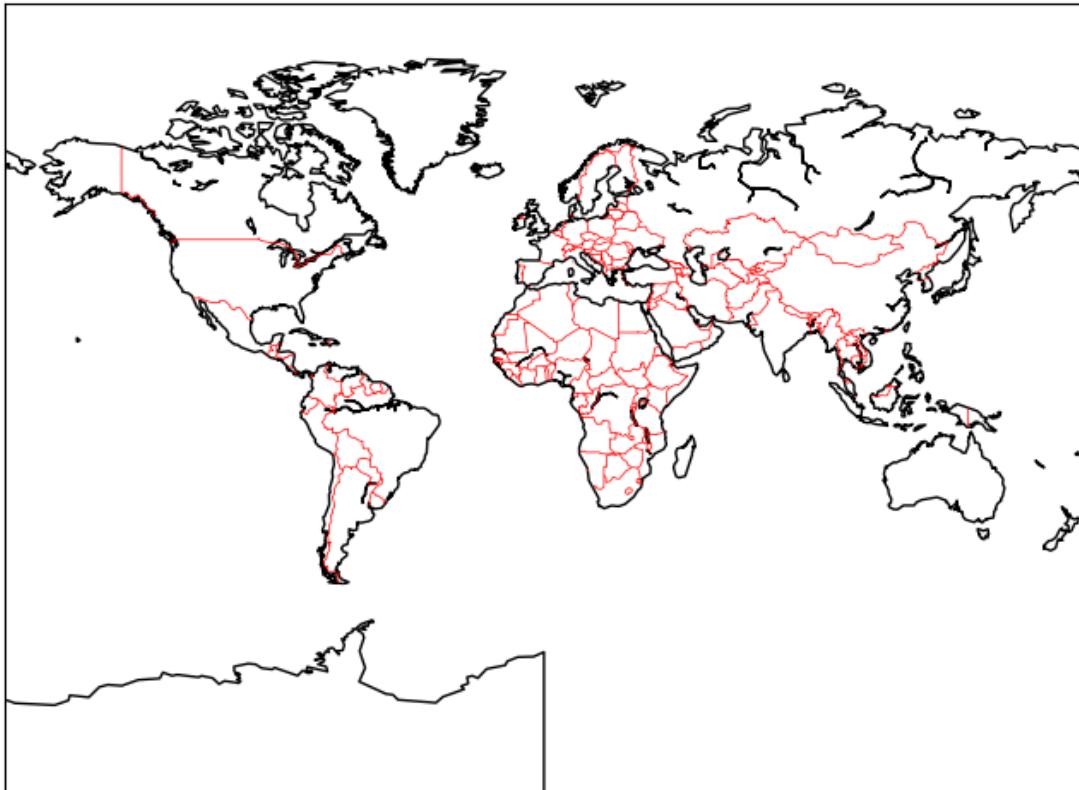
```
# full resolution
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='f')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries()
plt.show()
```



---

In [224]:

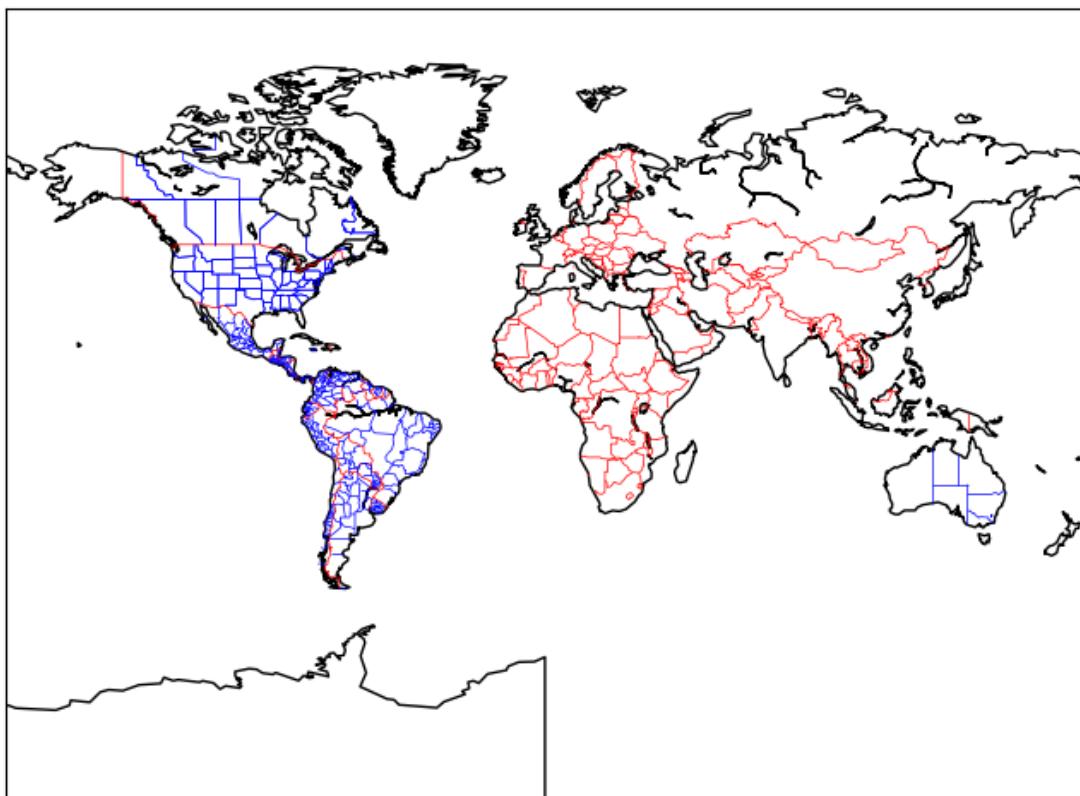
```
# Draw countries with color
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
plt.show()
```



---

In [225]:

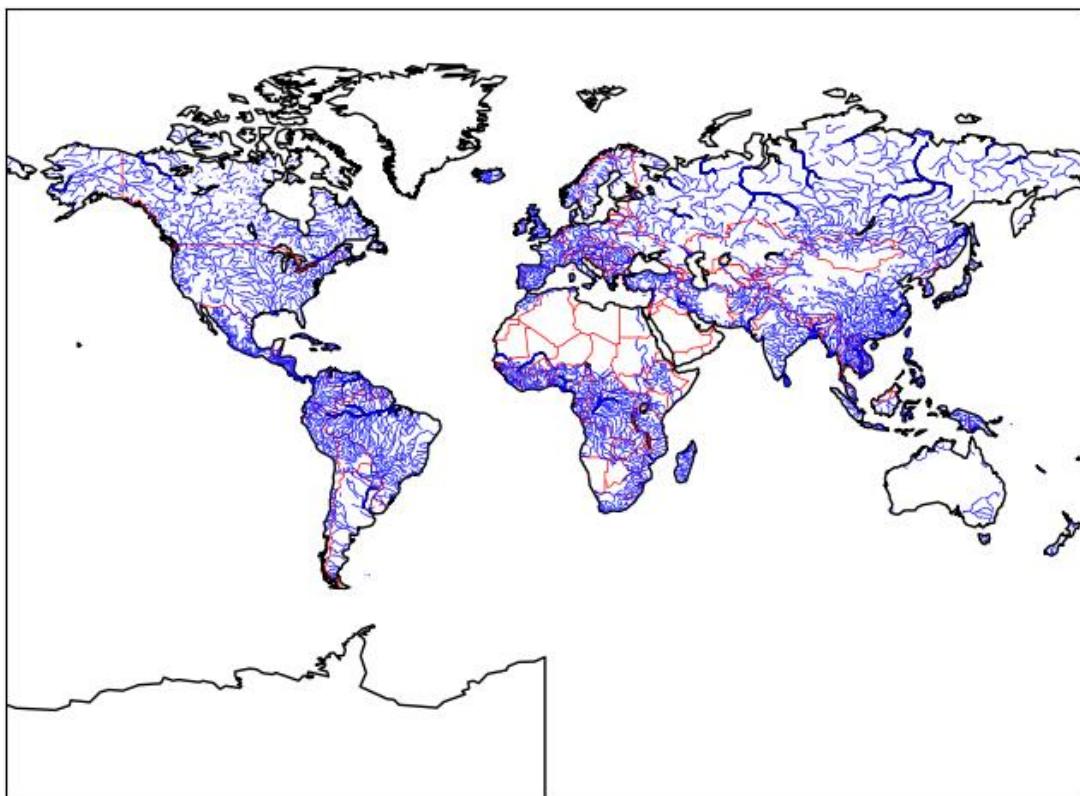
```
# Draw countries and states with color
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawstates(color='b')
plt.show()
```



---

In [226]:

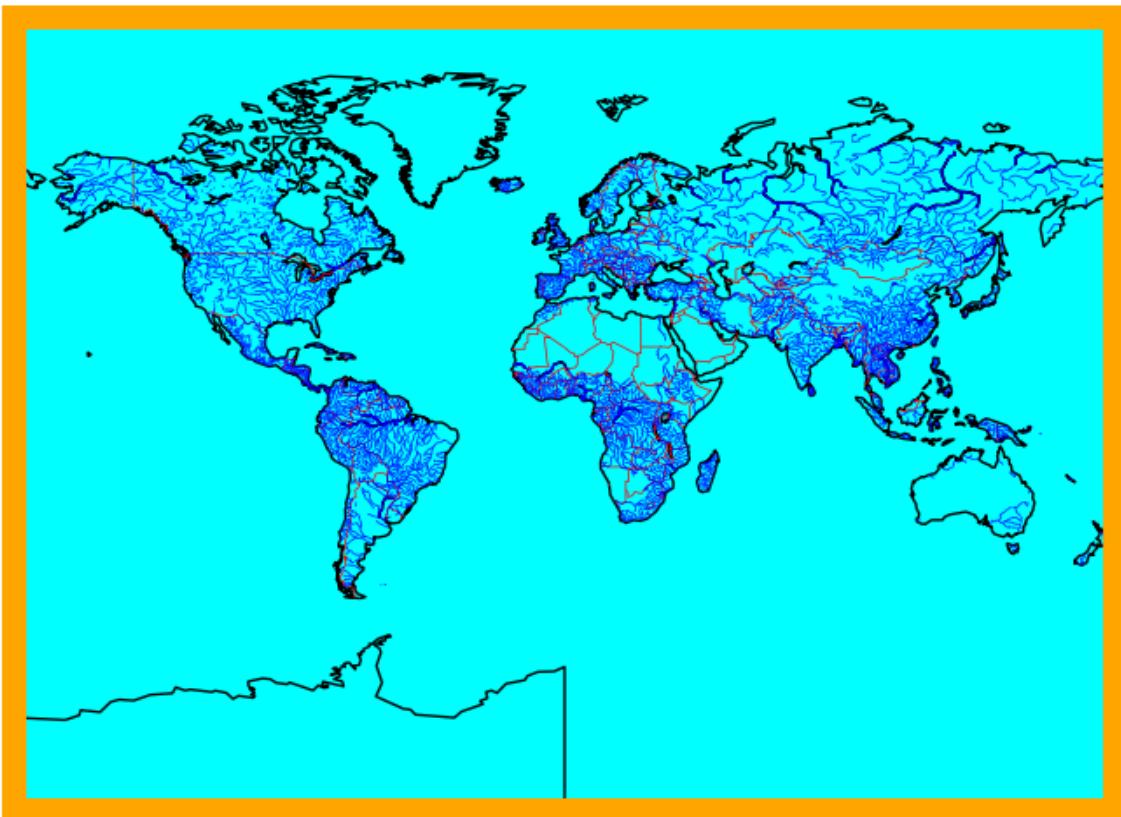
```
# Draw countries and rivers with color
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawrivers(color='b')
plt.show()
```



---

In [227]:

```
# Draw countries and rivers with color and mapboundary
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawrivers(color='b')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
plt.show()
```

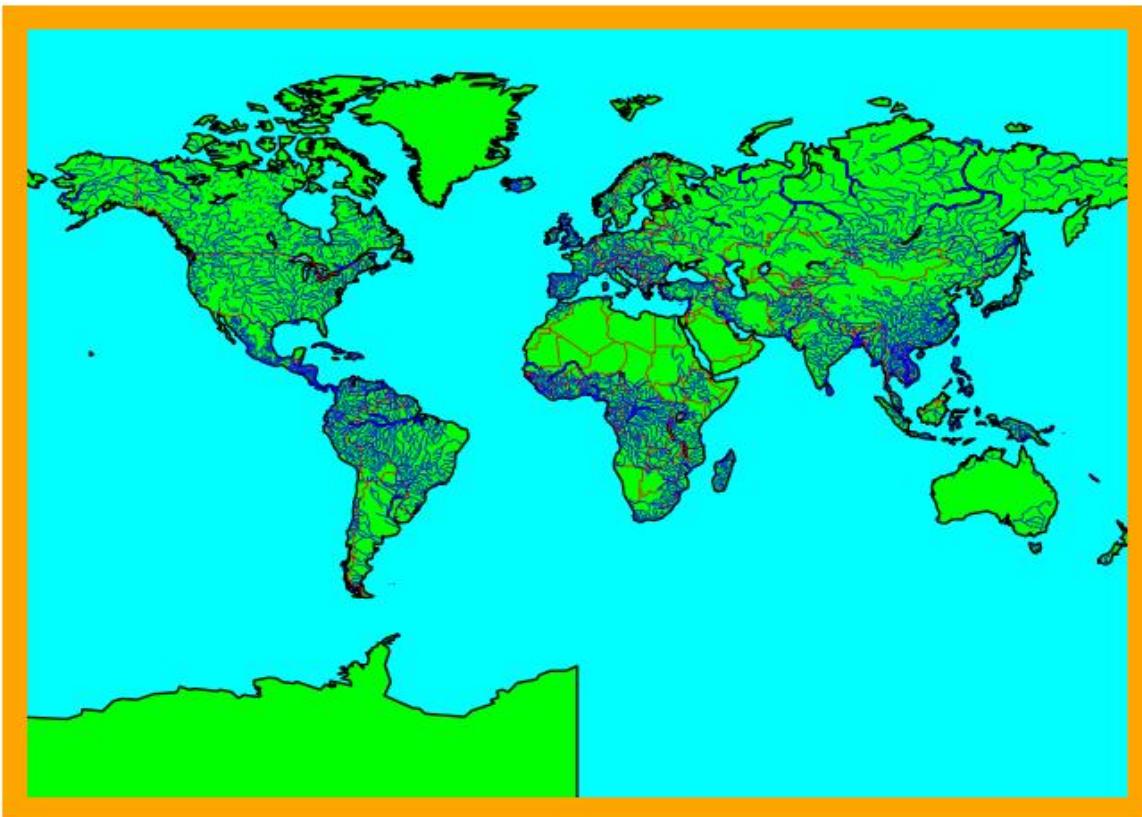


---

In [228]:

**# fillcontinents**

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawrivers(color='b')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
plt.show()
```

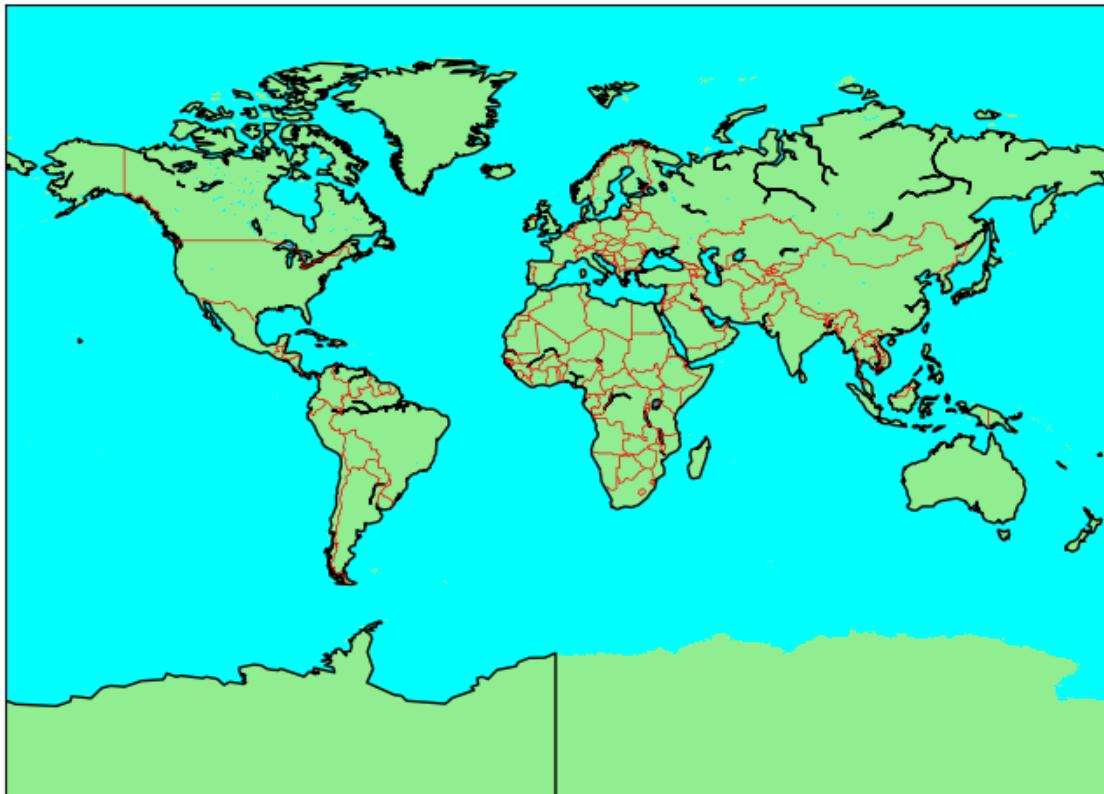


---

In [229]:

**#drawlsmask**

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawlsmask(land_color='lightgreen',ocean_color='aqua',lakes=True)
plt.show()
```

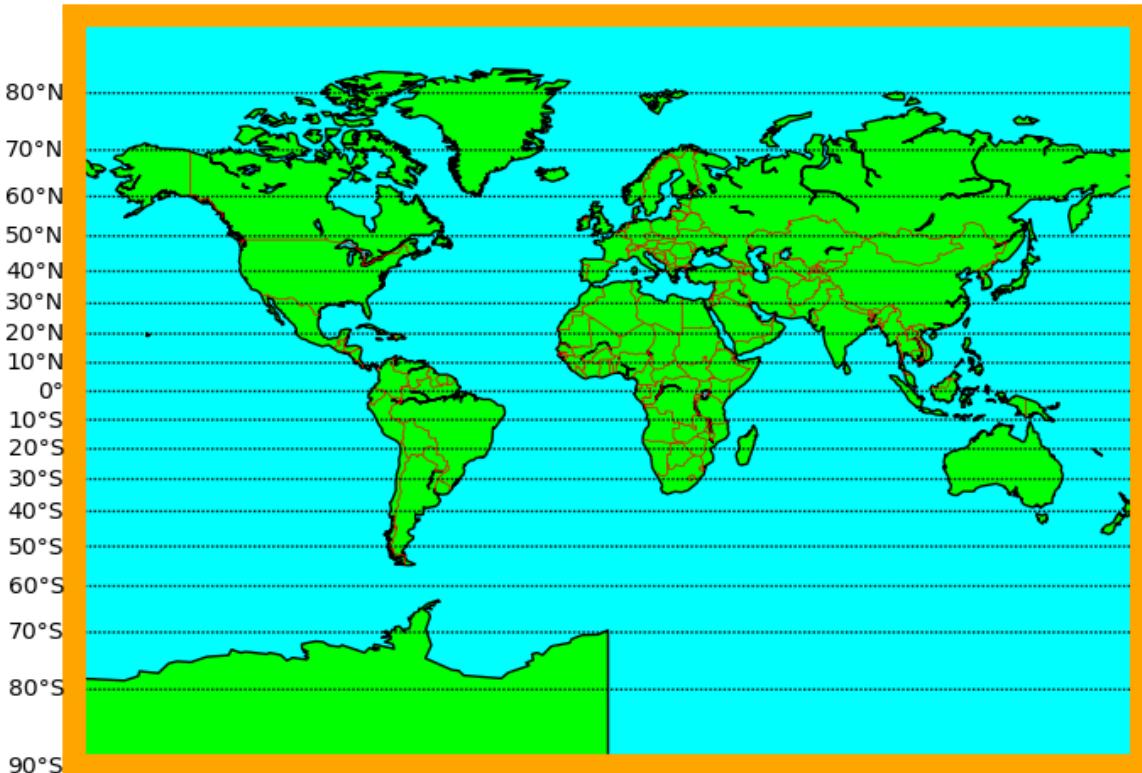


---

In [230]:

**# drawparallels**

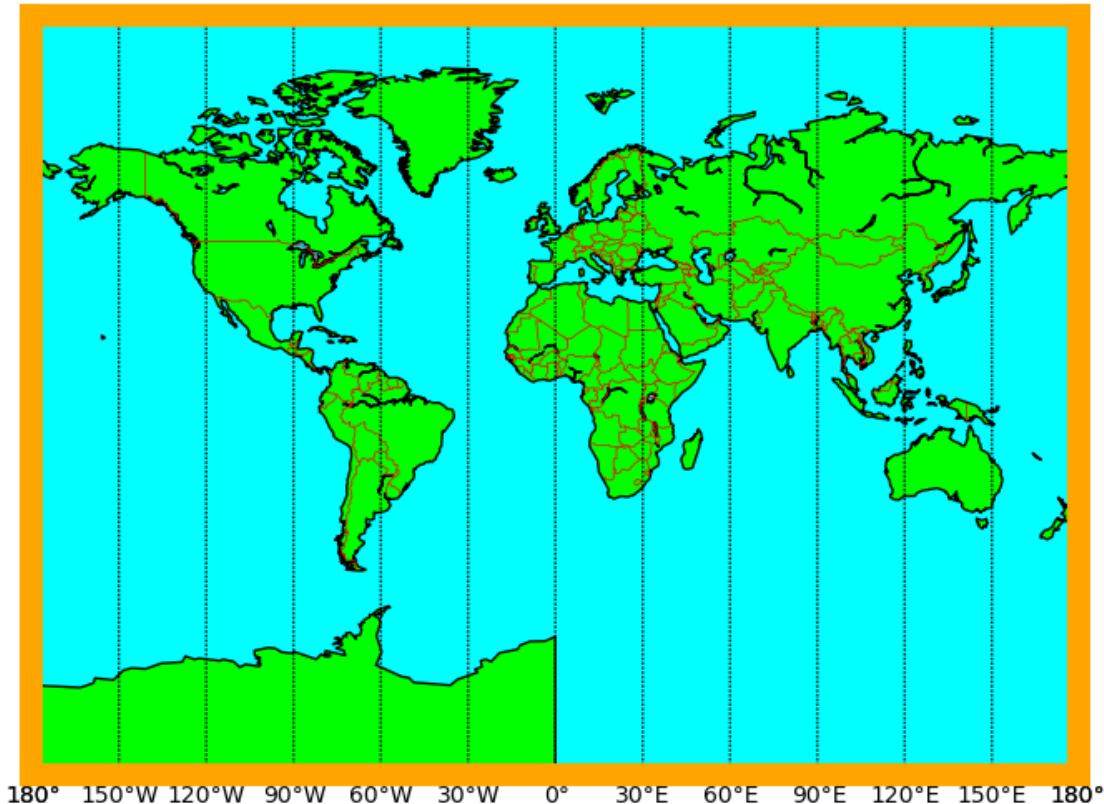
```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
plt.show()
```



---

In [231]:

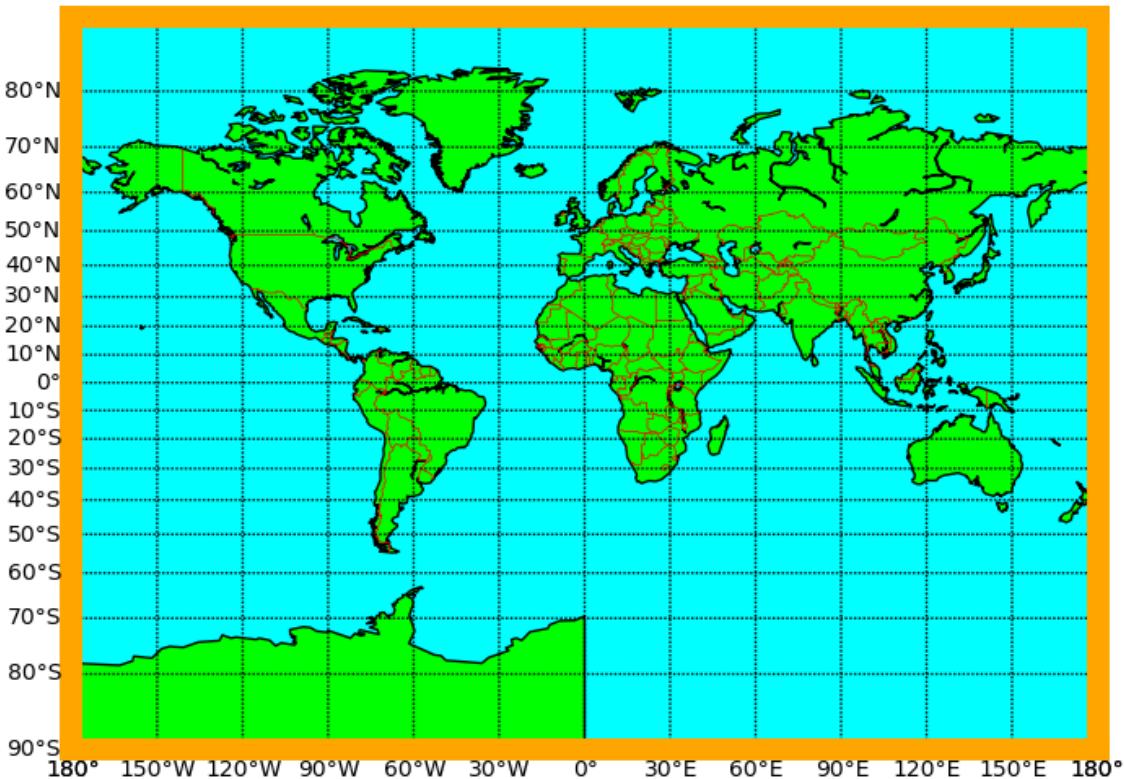
```
# drawmeridians
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
#labels=[left,rigth,top,bottom]
plt.show()
```



---

In [232]:

```
# drawparallels and drawmeridians
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
plt.show()
```



---

In [233]:

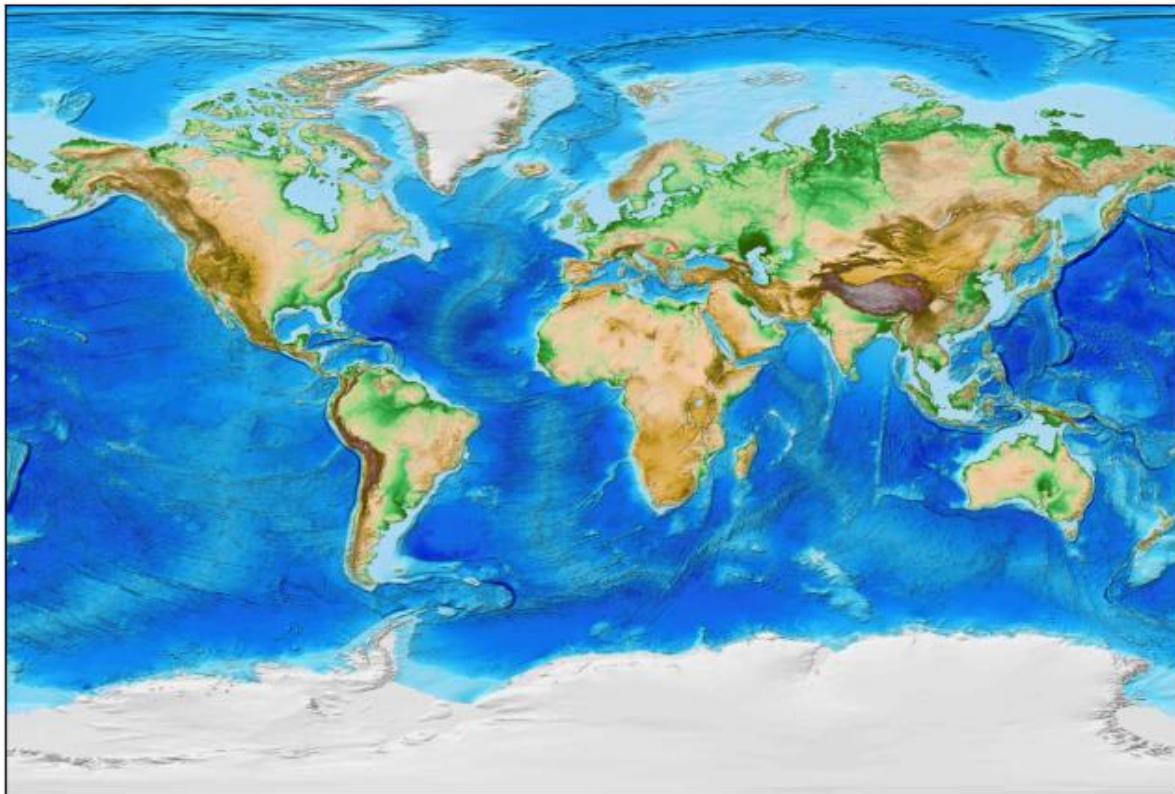
```
# bluemarble
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.bluemarble()
plt.show()
```



---

In [234]:

```
# etopo
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill',resolution='c')
m.etopo()
plt.show()
```



---

In [235]:

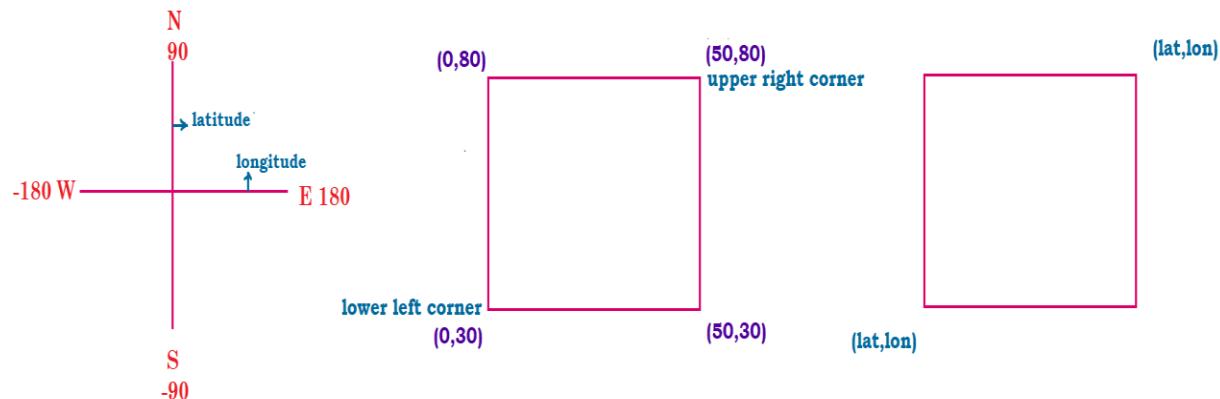
```
# shadedrelief
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
fig = plt.figure(figsize=(12,6))
m = Basemap(projection='mill', resolution='c')
m.shadedrelief()
plt.show()
```



## How to select particular area of the map

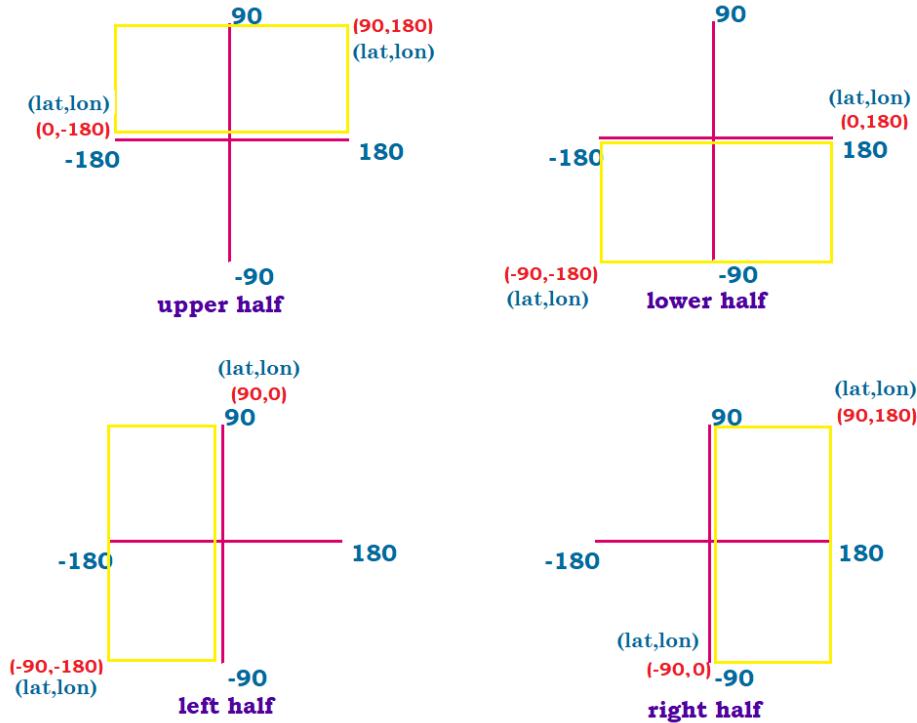
We can select/project any area on the map by using the following 4 values:

- ✓ lower left corner latitude(llcrnrlat)
  - ✓ lower left corner longitude(llcrnrlon)
  - ✓ upper right corner latitude(urcrnrlat)
  - ✓ upper right corner longitude(urcrnrlon)
- latitude: -90 to 90  
longitude: -180 to 180



**Latitude** is the angular distance of a point north or south of the equator. Lines of latitude are called parallels. The range of latitude is from -90 to 90.  
-90 represents south where as 90 represents north.

**Longitude** is the angular distance of a point east or west of the prime meridian(Greenwich Meridian). Lines of longitude are called meridians. The range of longitude is from -180 to 180.  
-180 represents west where as 180 represents east.



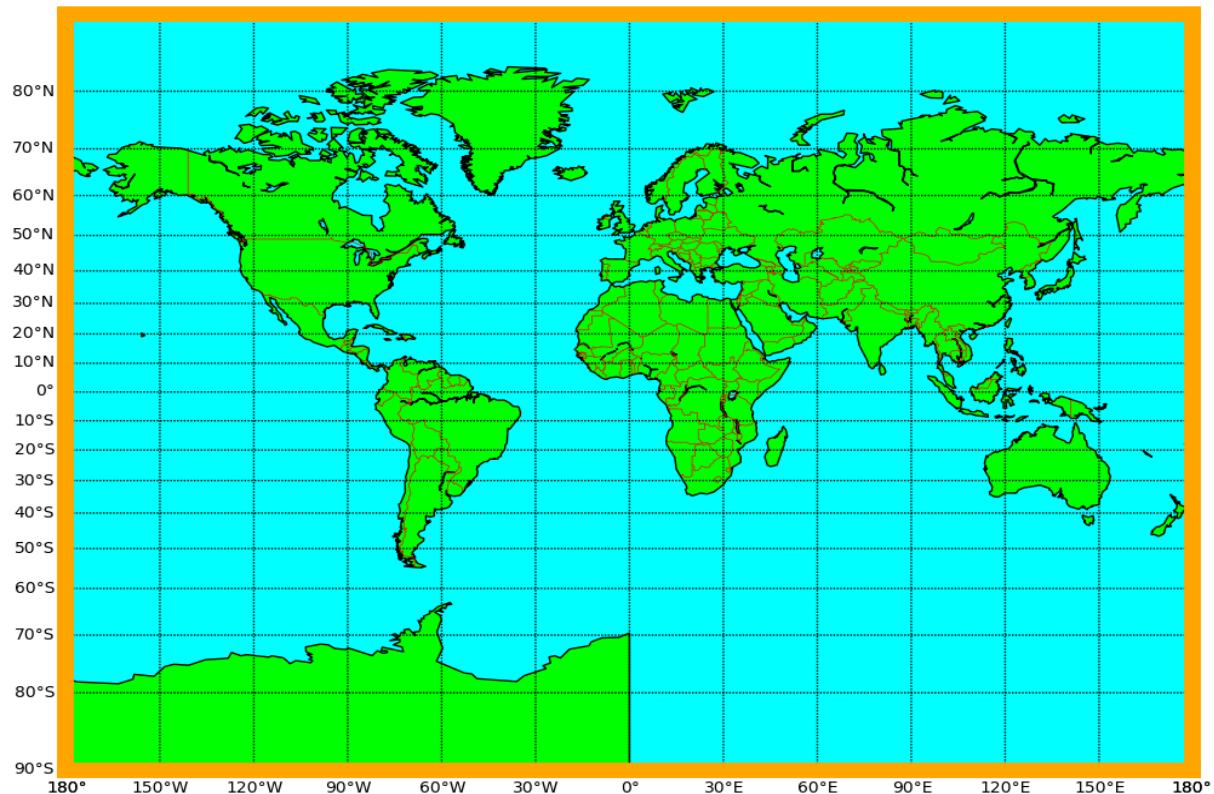
## selection of full world map

In [236]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=-90,
    llcrnrlon=-180,
    urcrnrlat=90,
    urcrnrlon=180)
m.drawcoastlines() #to draw continental coast lines
m.drawcountries(color='r')
```

```
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
plt.show()
```



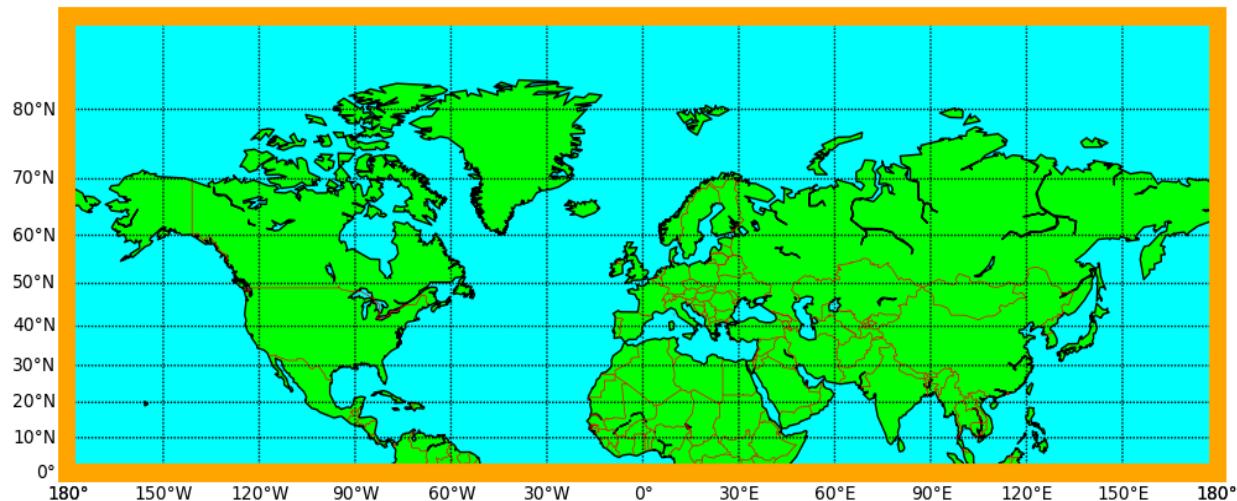
---

## selection of upper half of worldmap

In [237]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat= 0,
    llcrnrlon=-180,
    urcrnrlat=90,
    urcrnrlon=180)
m.drawcoastlines() #to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
plt.show()
```



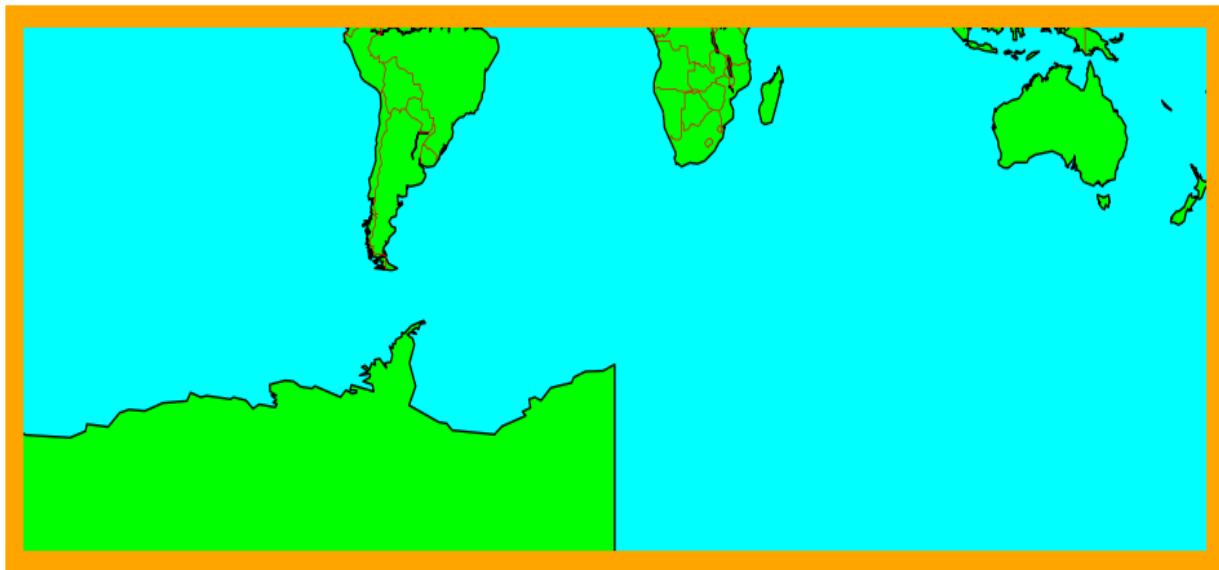
---

## selection of lower half of worldmap

In [238]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt

fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=-90,
    llcrnrlon=-180,
    urcrnrlat=0,
    urcrnrlon=180)
m.drawcoastlines() #to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
plt.show()
```

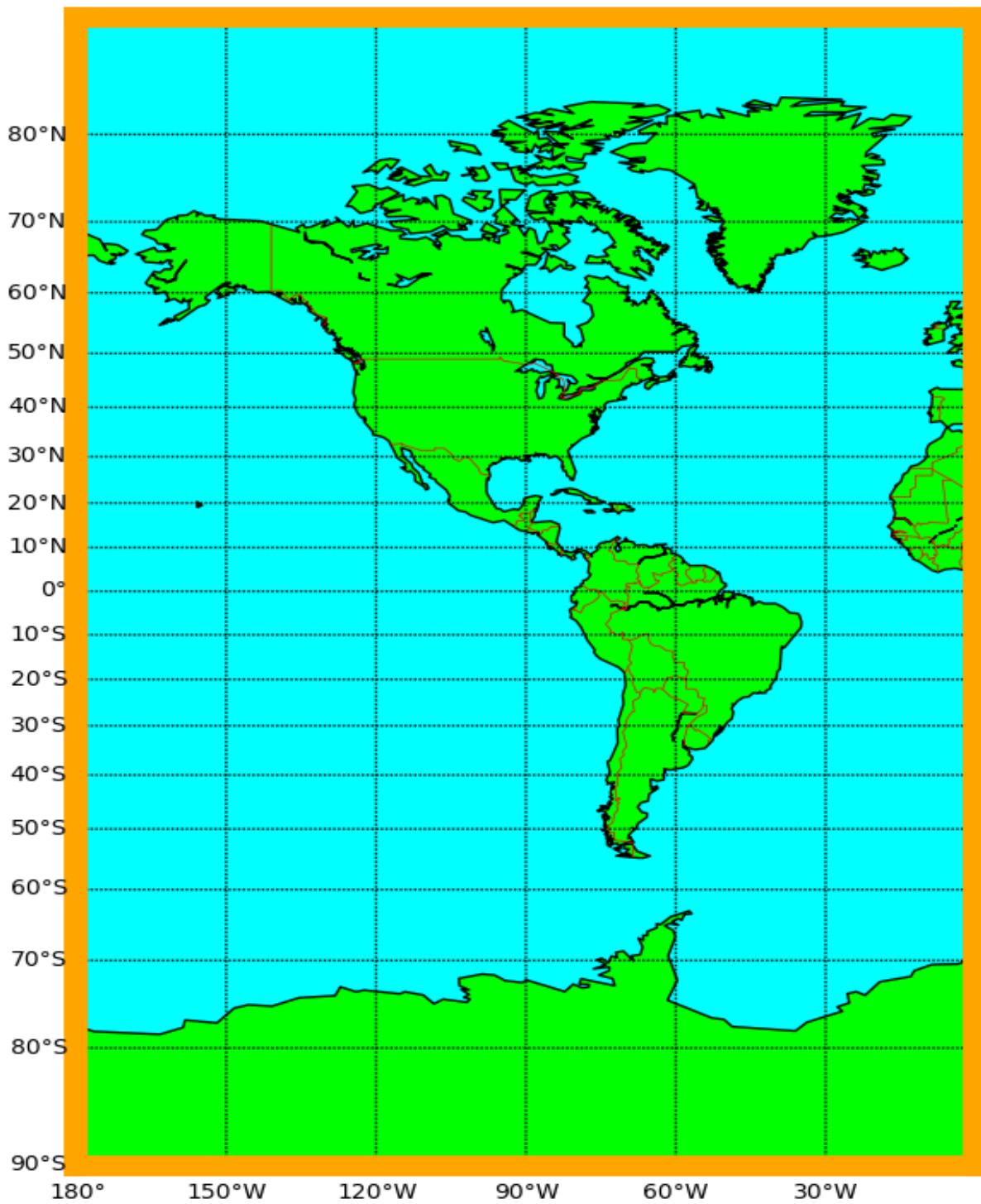


---

## selection of left half of worldmap

In [239]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=-90,
    llcrnrlon=-180,
    urcrnrlat=90,
    urcrnrlon=0)
m.drawcoastlines() #to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
plt.show()
```



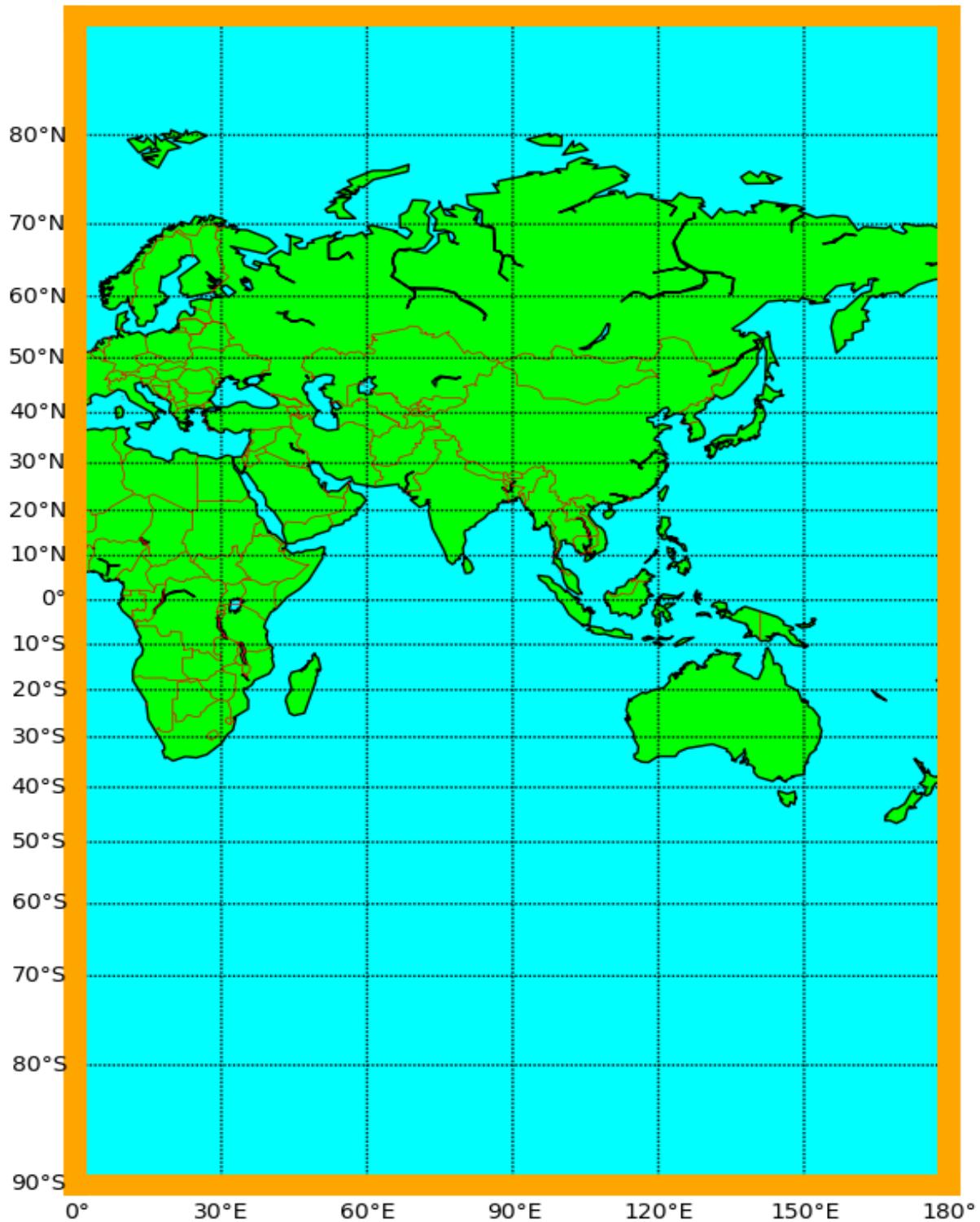
---

## selection of right half of worldmap

In [240]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np

fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=-90,
    llcrnrlon=0,
    urcrnrlat=90,
    urcrnrlon=180)
m.drawcoastlines() #to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange',linewidth=10)
m.fillcontinents(color='lime',lake_color='aqua')
m.drawparallels(np.arange(-90,90,10),labels=[True,False,False,False])
#labels=[left,rigth,top,bottom]
m.drawmeridians(np.arange(-180,180,30),labels=[False,False,False,True])
plt.show()
```



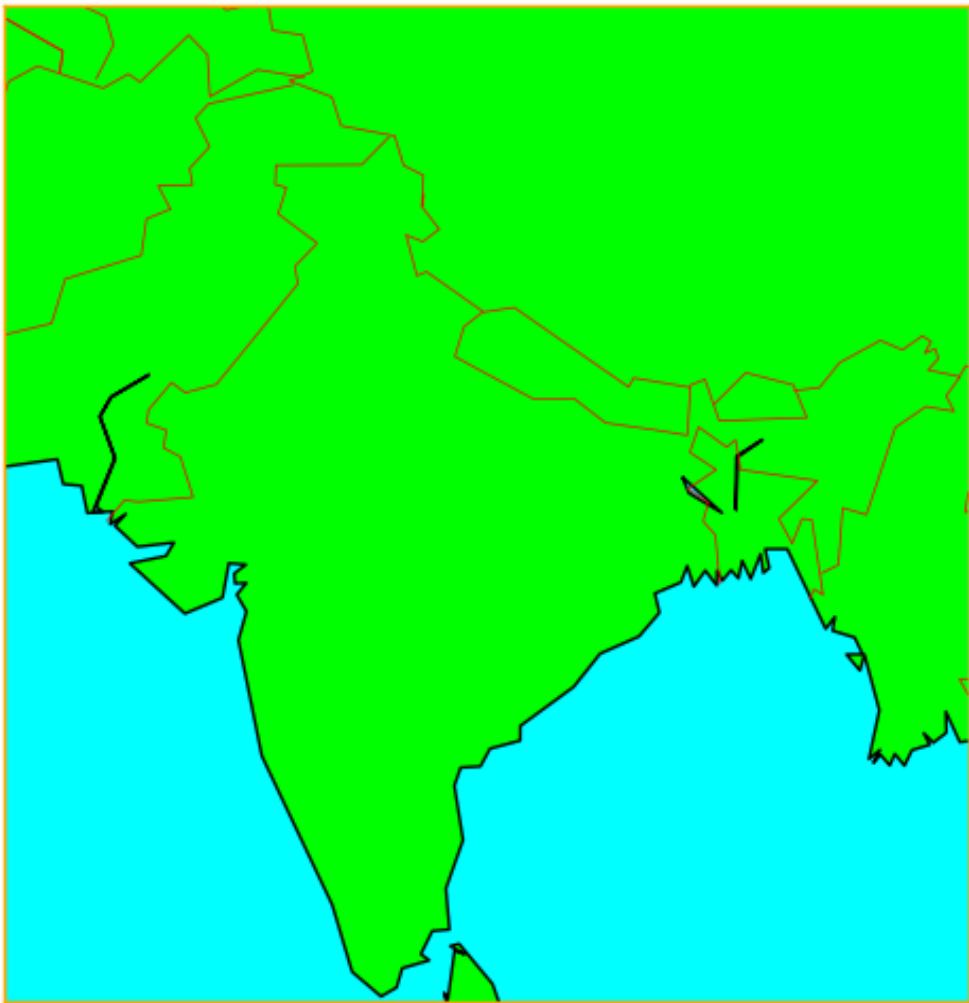
---

## How to select only India

- ✓ Open <https://www.google.com/maps/> and take the latitude and longitude of lower left corner and upper right corner  
7.890072140756692,64.6128442670092 → lower left corner(lat,lon)  
39.31741853835297,97.64121300133036 → upper right corner(lat,lon)

In [241]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
    urcrnrlat=39.31741853835297,
    urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(fill_color='aqua',color='orange')
m.fillcontinents(color='lime',lake_color='aqua')
plt.show()
```



---

## Locate Delhi on India Map:

From google take the latitude and longitude of Delhi  
28.7041° N, 77.1025° E

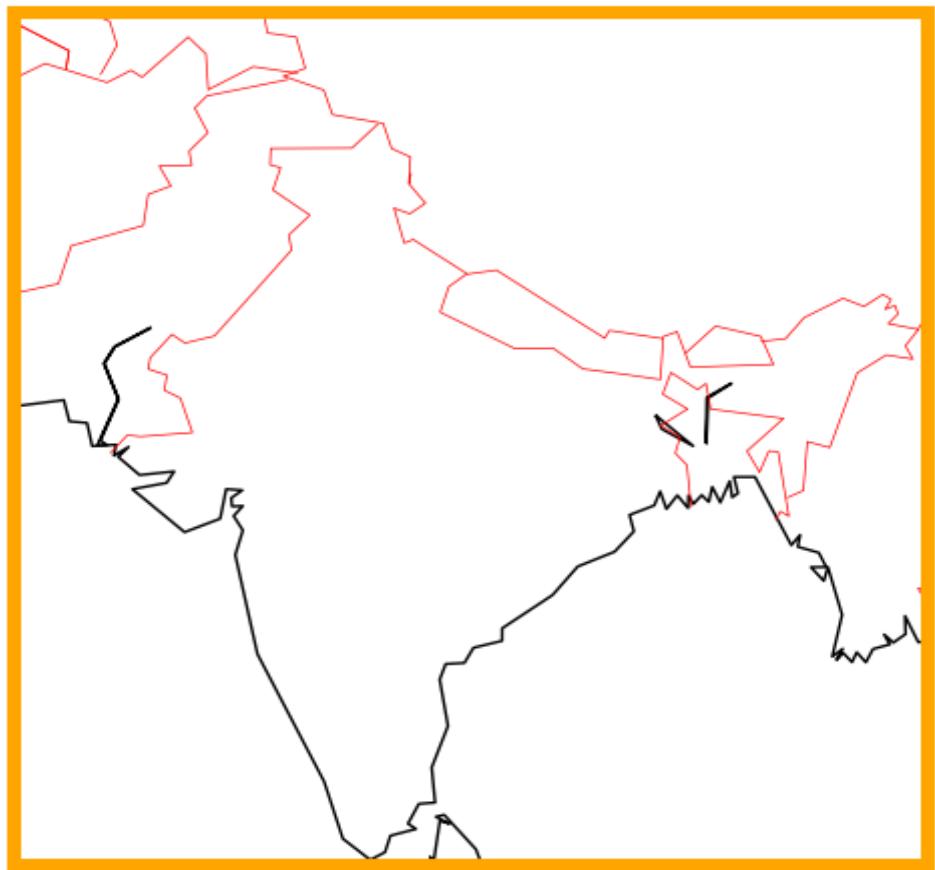
In [242]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
    urcrnrlat=39.31741853835297,
    urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=5)
#x,y = m(lon,lat)
x,y = m(77.1025,28.7041)
print("Converting the longitude and lattitude values to x and y")
print(f'x value ==> {x}')
print(f'y value ==> {y}')
```

Converting the longitude and lattitude values to x and y

x value ==> 1388785.699088861

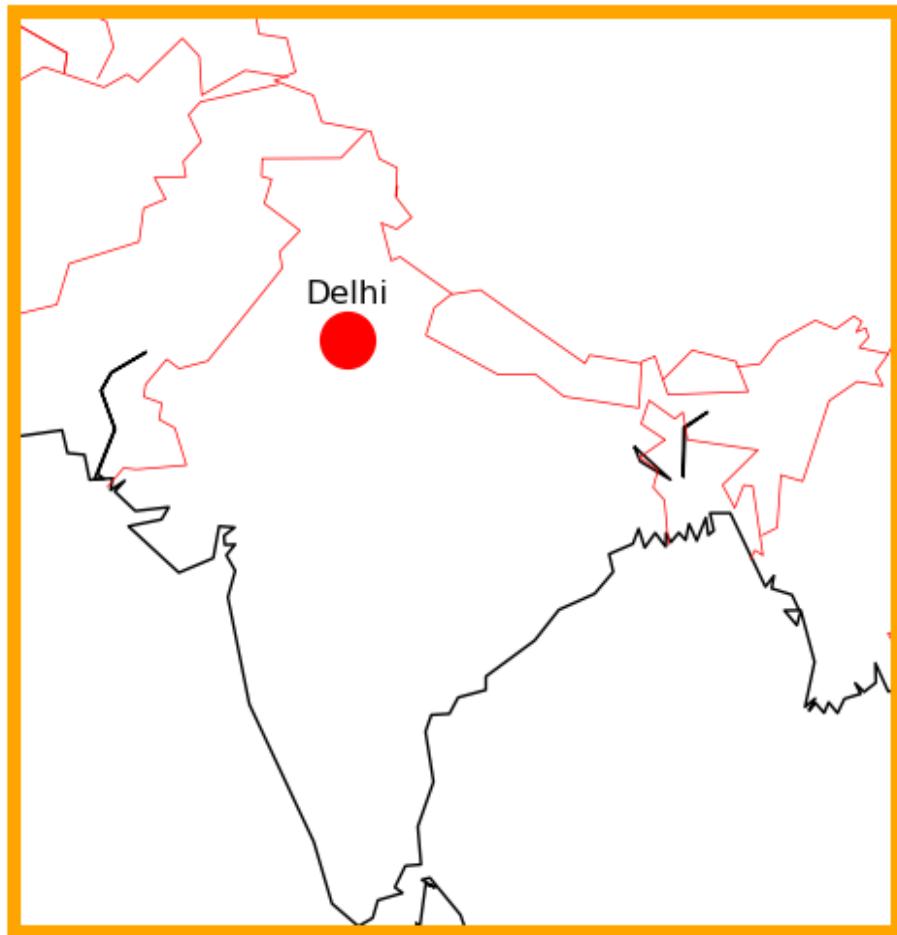
y value ==> 2401680.7952138484



In [243]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
    urcrnrlat=39.31741853835297,
    urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
```

```
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=5)
#x,y = m(lon,lat)
x,y = m(77.1025,28.7041)
plt.plot(x,y,'ro-', markersize=20)
plt.text(x,y+150000,'Delhi', fontsize=12, ha='center')
plt.show()
```



---

## Top-5 Tourist Places in India:

Tajmahal, Charminar, Red Fort, Tirumala, Shirdi

Tajmahal → 27.1751° N, 78.0421° E

Charminar → 17.3616° N, 78.4747° E

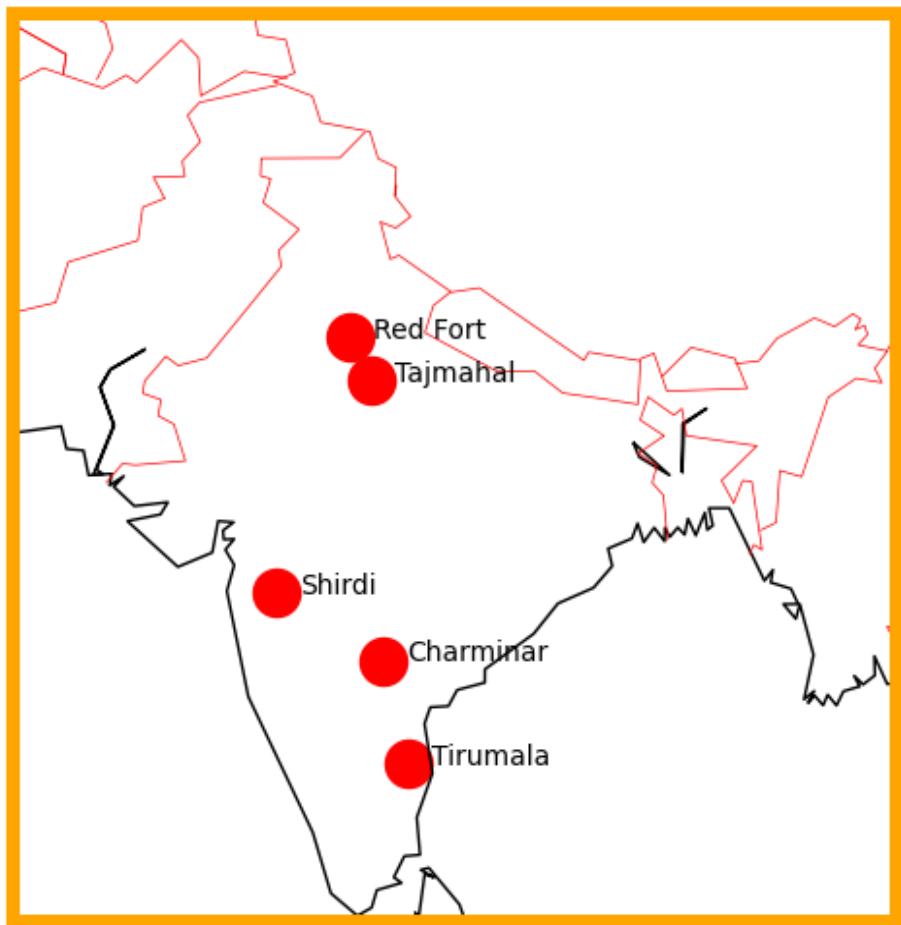
Red Fort → 28.6562° N, 77.2410° E

Tirumala → 13.6288° N, 79.4192° E

Shiridi → 19.7645° N, 74.4762° E

In [244]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
    urcrnrlat=39.31741853835297,
    urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=5)
top_5_places = ['Tajmahal','Charminar','Red Fort','Tirumala','Shirdi']
longitudes = [78.0421,78.4747,77.2410,79.4192,74.4762]
latitudes = [27.1751, 17.3616, 28.6562, 13.6288, 19.7645]
x,y = m(longitudes,latitudes)
plt.scatter(x,y,s=300,color='red')
for i,label in enumerate(top_5_places):
    plt.text(x[i]+100000,y[i],label)
plt.show()
```



## Top-5 Tourist Places in India: with colormap

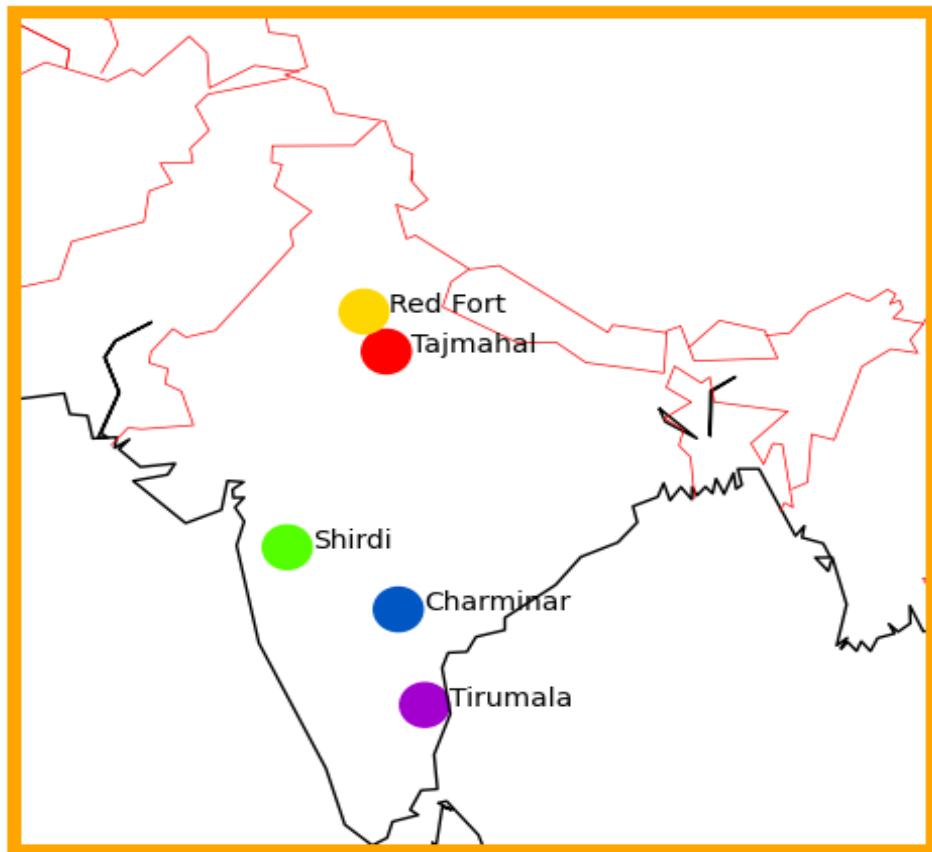
In [245]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
```

```

urcrnrlat=39.31741853835297,
urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=5)
top_5_places = ['Tajmahal','Charminar','Red Fort','Tirumala','Shirdi']
longitudes = [78.0421,78.4747,77.2410,79.4192,74.4762]
latitudes = [27.1751, 17.3616, 28.6562, 13.6288, 19.7645]
x,y = m(longitudes,latitudes)
plt.scatter(x,y,s=300,c=[0,20,40,60,80],cmap='prism')
for i,label in enumerate(top_5_places):
    plt.text(x[i]+100000,y[i],label)
plt.show()

```

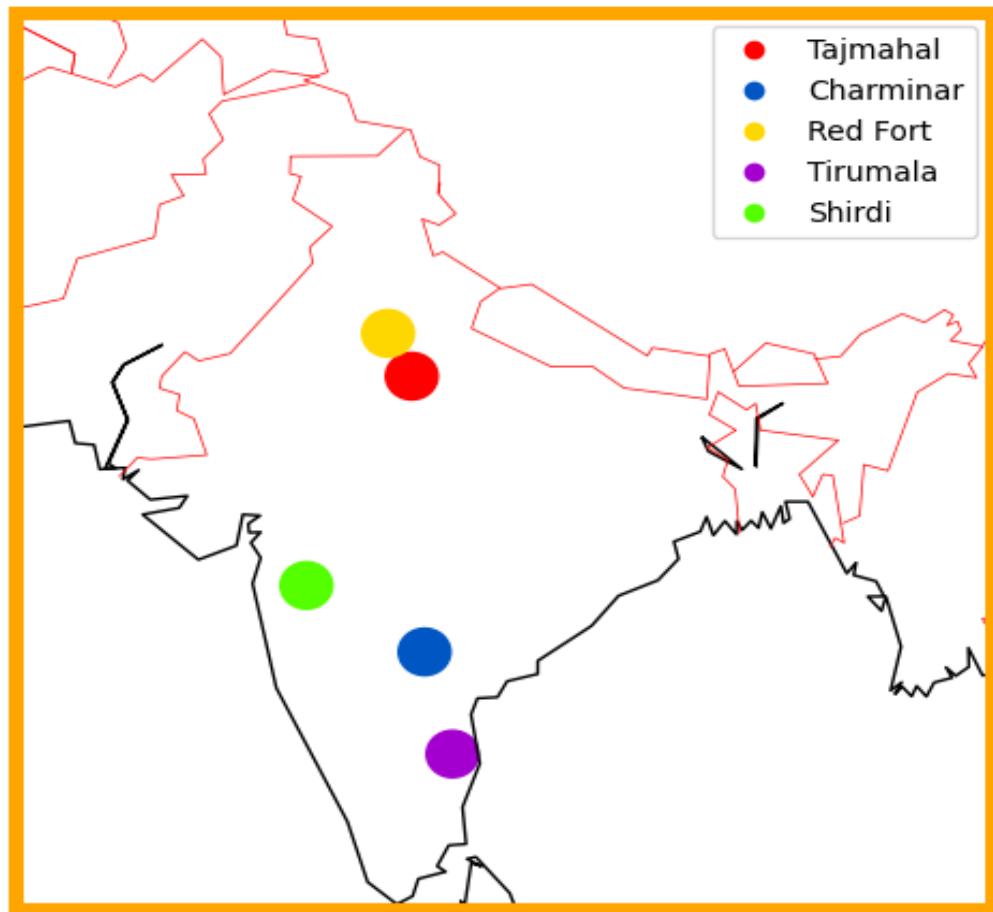


---

## Top-5 Tourist Places in India: with color legend

In [246]:

```
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap
import numpy as np
fig = plt.figure(figsize=(12,6))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.890072140756692,
    llcrnrlon=64.6128442670092,
    urcrnrlat=39.31741853835297,
    urcrnrlon=97.64121300133036
)
m.drawcoastlines() # to draw continental coast lines
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=5)
top_5_places = ['Tajmahal','Charminar','Red Fort','Tirumala','Shirdi']
longitudes = [78.0421,78.4747,77.2410,79.4192,74.4762]
latitudes = [27.1751, 17.3616, 28.6562, 13.6288, 19.7645]
x,y = m(longitudes,latitudes)
scat = plt.scatter(x,y,s=300,c=[0,20,40,60,80],cmap='prism')
plt.legend(handles=scat.legend_elements()[0],labels=top_5_places)
plt.show()
```



### Top-5 IITs in India

IIT Madras, Chennai → Latitude: 12.9915° N, Longitude: 80.2337° E

IIT Delhi, New Delhi → Latitude: 28.5457° N, Longitude: 77.1928° E

IIT Bombay, Mumbai → Latitude: 19.1335 Longitude: 72.9092.

IIT Kanpur → Latitude: 26.5123° N, Longitude 80.2329° E

IIT Kharagpur → Latitude: 22.3185 Longitude: 87.3060.

latitudes = [12.9915,28.5457,19.1335, 26.5123,22.3185]

longitudes= [80.2337,77.1928,72.9092,80.2329,87.3060]

```
top_5_iits = ['IIT Madras, Chennai', 'IIT Delhi, New Delhi', 'IIT Bombay,
Mumbai', 'IIT Kanpur','IIT Kharagpur']
```

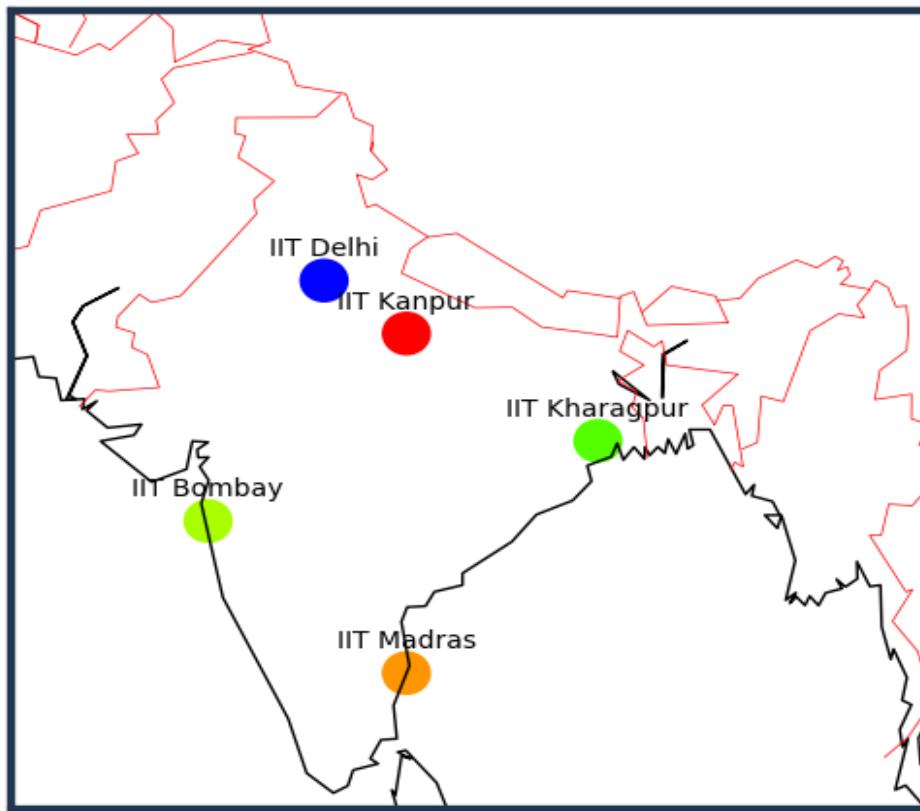
---

In [247]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

fig = plt.figure(figsize=(12,6))
# 7.4042598501668655, 65.6225880268181
# 38.51237702578474, 99.46047644125078
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=7.4042598501668655,
    llcrnrlon=65.6225880268181,
    urcrnrlat=38.51237702578474,
    urcrnrlon=99.46047644125078
)
m.drawcoastlines()
m.drawcountries(color='r')
m.drawmapboundary(color='#20354e', linewidth=3)
latitudes = [12.9915, 28.5457, 19.1335, 26.5123, 22.3185]
longitudes = [80.2337, 77.1928, 72.9092, 80.2329, 87.3060]
top_5_iits = ['IIT Madras', 'IIT Delhi', 'IIT Bombay', 'IIT Kanpur', 'IIT Kharagpur']
x, y = m(longitudes, latitudes)
plt.scatter(x, y, s=300, c=[45, 20, 68, 15, 90], cmap='prism')
for i, label in enumerate(top_5_iits):
    plt.text(x[i], y[i]+120000, label, ha='center')

plt.show()
```



## Top-5 IITs in India: with colormap

In [248]:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.basemap import Basemap

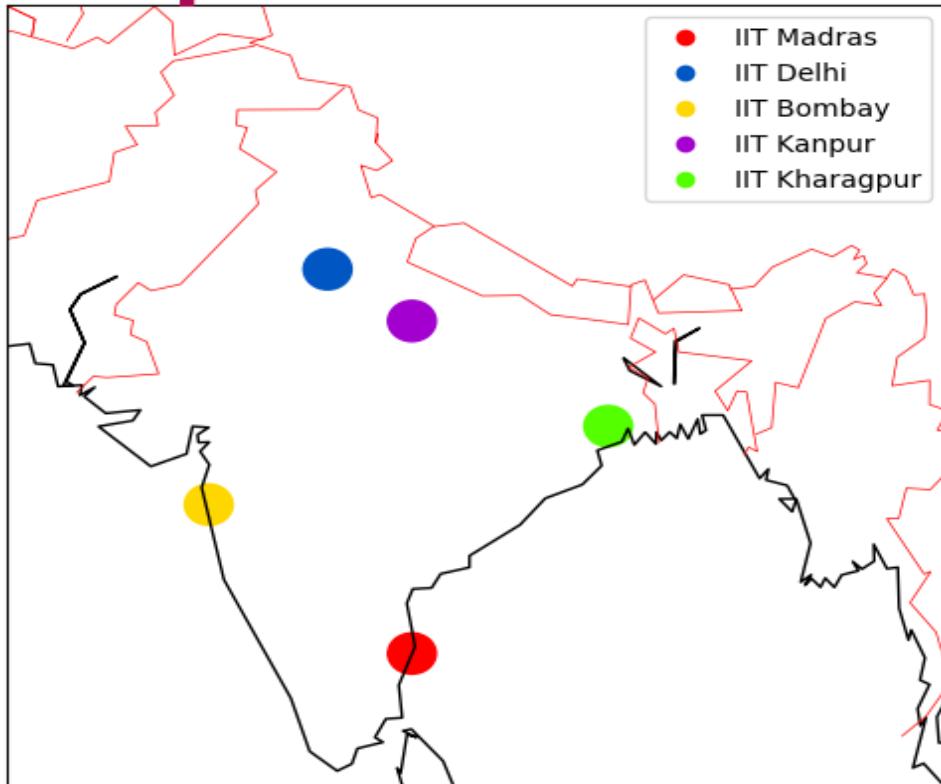
fig = plt.figure(figsize=(12,6))
# 7.4042598501668655, 65.6225880268181
# 38.51237702578474, 99.46047644125078
m = Basemap(projection='mill', resolution='c',
            llcrnrlat=7.4042598501668655,
            llcrnrlon=65.6225880268181,
            urcrnrlat=38.51237702578474,
            urcrnrlon=99.46047644125078
)
```

```

m.drawcoastlines()
m.drawcountries(color='r')
latitudes = [12.9915,28.5457,19.1335, 26.5123,22.3185]
longitudes= [80.2337,77.1928,72.9092,80.2329,87.3060]
top_5_iits = ['IIT Madras', 'IIT Delhi','IIT Bombay', 'IIT Kanpur','IIT Kharagpur']
x,y = m(longitudes,latitudes)
scat = plt.scatter(x,y,s=300,c=[10,30,50,70,90],cmap='prism')
plt.legend(handles=scat.legend_elements()[0],labels=top_5_iits)
plt.title('Top 5 IITs in India', color='#A80D53',weight=1000,size=30)
plt.show()

```

## Top 5 IITs in India



---

## Top 10 states with the highest total covid-19 cases in India

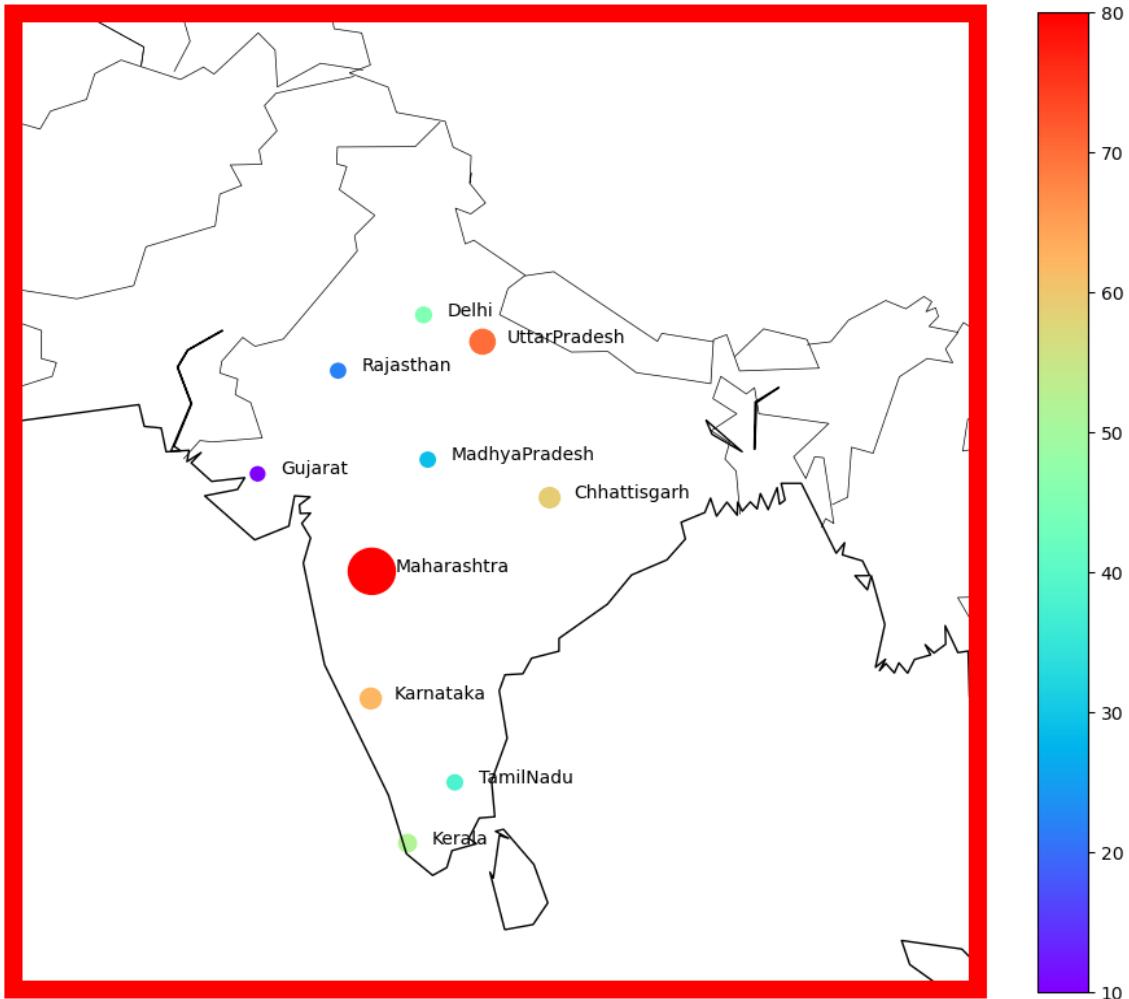
In [249]:

```
from mpl_toolkits.basemap import Basemap
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=3.7057276691051433,
    llcrnrlon=61.71238525260492,
    urcrnrlat=39.10908305333747,
    urcrnrlon=98.09910379996306)
m.drawcoastlines()
m.drawcountries(color='k')
m.drawmapboundary(color='red', linewidth=10)
totalcases = np.array([672037,191457,133562,128019,94009,
                      74941,70391,68576,67135,61647])
places=['Maharashtra','UttarPradesh','Karnataka','Chhattisgarh','Kerala',
        'Delhi','TamilNadu','MadhyaPradesh','Rajasthan','Gujarat']
lon= [75.24097269416545,79.41734049534212,75.20035962294948,
      81.9508271693893, 76.58860906089518, 77.19485012668703,
      78.37206795299163, 77.35150294839225,
      73.96775870899172, 70.93096200054012]
lat= [19.52952488935126,27.8688783819858,14.78193601460986,
      22.24028511499017, 9.301909791605237, 28.821084470949717,
      11.61537886387894,23.62693177315555,
      26.834827566624202,23.11050796063762]

x,y = m(lon,lat)
plt.scatter(x,y,s=totalcases/1000,
            c=[80,70,62,59,52,45,38,29,22,10],cmap='rainbow')
plt.colorbar()
for i,label in enumerate(places):
    plt.text(x[i]+100000,y[i],label )
plt.title('Top 10 states with the highest Total Covid-19 cases in INDIA',
          color='#A80D53',weight=1000,size=15,pad=20)
plt.show()
```

---

### Top 10 states with the highest Total Covid-19 cases in INDIA



---

## Top 10 states with the highest total covid-19 cases in India → Reading data from csv file

In [250 ]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
import numpy as np
import csv
fig = plt.figure(figsize=(12,10))
m = Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=3.7057276691051433,
    llcrnrlon=61.71238525260492,
    urcrnrlat=39.10908305333747,
    urcrnrlon=98.09910379996306)
m.drawcoastlines()
m.drawcountries(color='r')
m.drawmapboundary(color='orange', linewidth=4)
state_names = []
total_cases = np.array([], dtype=int)
f = open('Latest Covid-19 India Status.csv', 'r', encoding='utf-8')
r = csv.reader(f) #returns csvreader object
h = next(r) # read header column and ignore

lon= [75.24097269416545,76.83793537961354,75.74901207037114,
      78.81135687436876, 78.91193607106908, 79.87816782390769,
      87.17404280995771, 77.22087515091808,
      82.08202682134386, 84.43170304176574]
lat= [19.52952488935126,9.364025318933216,14.802952447940608,
      11.550763028251833, 5.003889071936557, 27.868512459396865,
      23.05855832624224, 28.708931193032743,
      2.23998411698696, 20.537195928132455]

for row in r:
    state_names.append(row[0])
    total_cases = np.append(total_cases,int(row[1]))
```

```

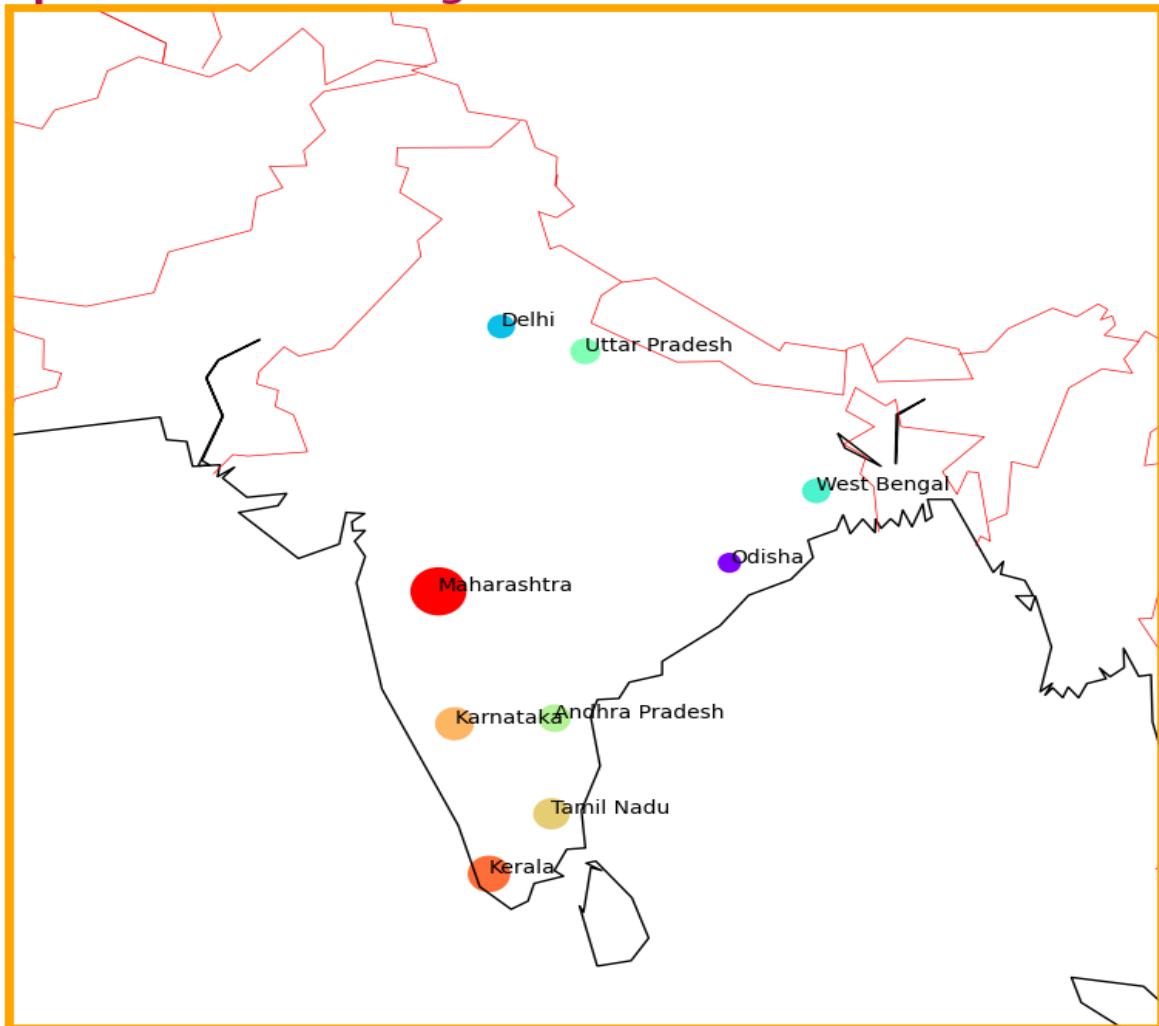
x,y = m(lon,lat)
plt.scatter(x,y,s=total_cases[:10]/10000,
            c=[80,70,62,59,52,45,38,29,22,10],cmap='rainbow')

for i,label in enumerate(state_names[:10]):
    plt.text(x[i],y[i]+1000,f'{label}',fontsize=10,weight=500 )

plt.title('Top 10 states with highest total covid-19 cases in India',
          fontsize=18,color='#A80D53',weight=1000)
plt.show()

```

### Top 10 states with highest total covid-19 cases in India



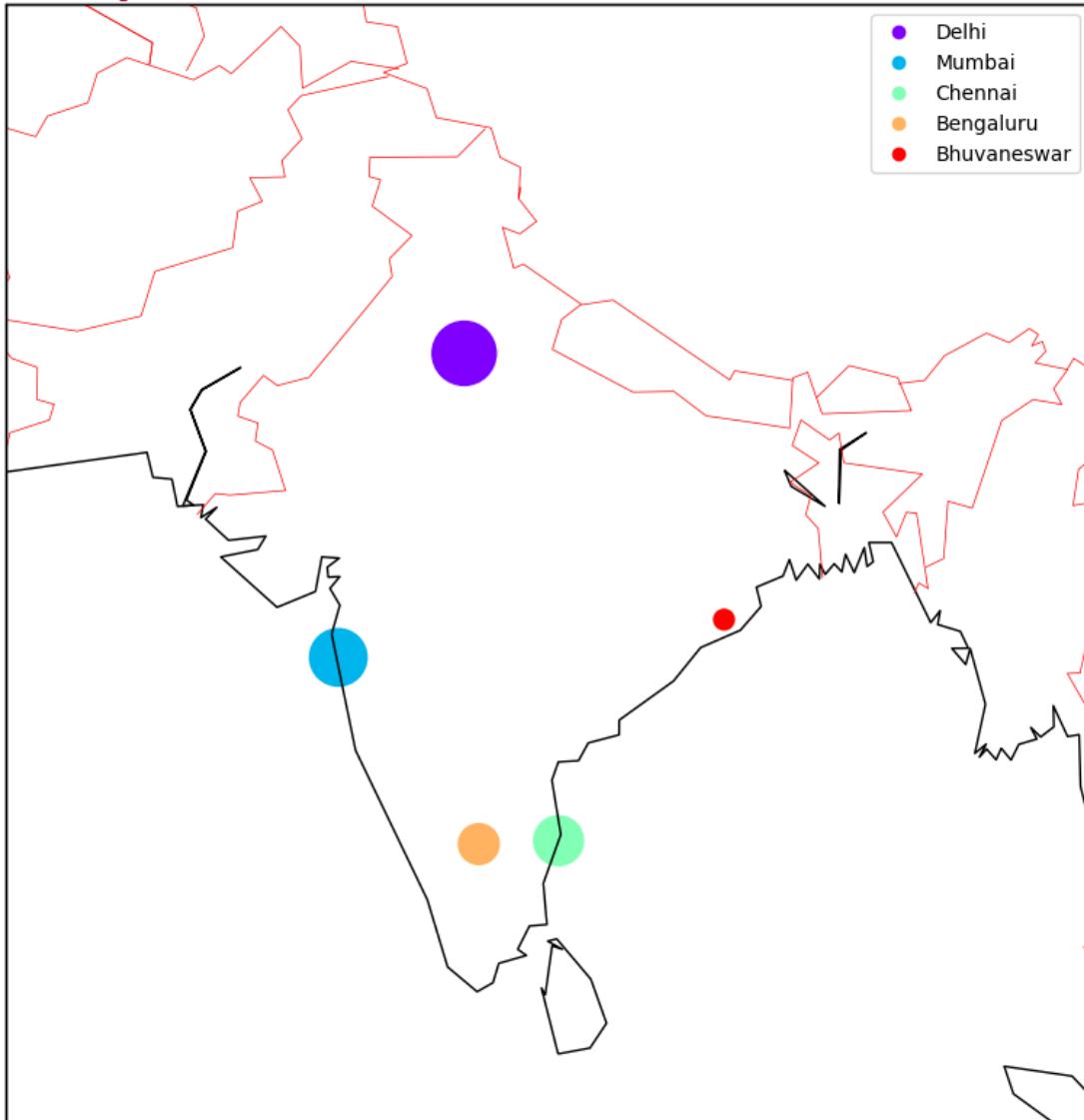
---

## Top 5 cities with more covid cases in india:

In [251]:

```
from mpl_toolkits.basemap import Basemap
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(12,10))
m=Basemap(
    projection='mill',
    resolution='c',
    llcrnrlat=3.7057276691051433,
    llcrnrlon=61.71238525260492,
    urcrnrlat=39.10908305333747,
    urcrnrlon=98.09910379996306)
m.drawcoastlines()
m.drawcountries(color='red')
m.drawmapboundary()
top_5_covidcases_cities=['Delhi','Mumbai','Chennai','Bengaluru','Bhuvaneswar']
longitudes=[77.1025,72.8777,80.2707,77.5946,85.8245]
latitudes=[28.7041,19.0760,13.0827,12.9716,20.2961]
x,y=m(longitudes,latitudes)
sizes=[1000,800,600,400,100]
#colors=['red','blue','black','orange','green']
scatter=plt.scatter(x,y,ss=sizes,c=[0,20,40,60,80],cmap='rainbow')
plt.legend(handles=scatter.legend_elements()[0],labels=top_5_covidcases_cities)
plt.title('Top 5 cities with more covid cases in India',
          color='#A80D53',weight=1000,size=20)
plt.show()
```

## Top 5 cities with more covid cases in India



---

## Chapter-17

# Three-Dimensional(3-D) Plotting in Matplotlib

### Three-Dimesional plotting(3-D) in Matplotlib:

- ✓ The original version of matplotlib supports only 2-D plotting.
- ✓ But later versions supports 3-D plotting also. For this we have to use 'mplot3d' toolkit.
- ✓ We can import this toolkit as follows:

```
from mpl_toolkits import mplot3d
```

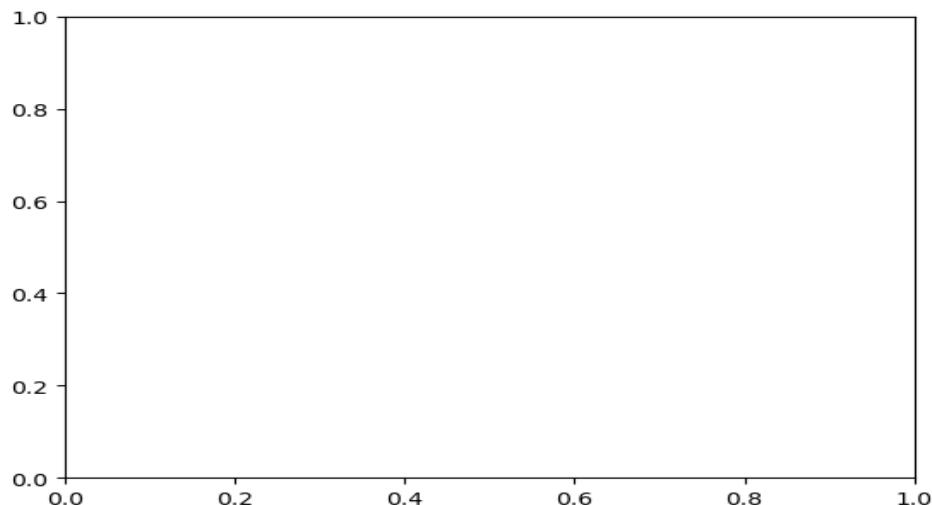
- ✓ It is bydefault available with matplotlib and we are not required to install separately.

### Creating of 3-D axes object

We can create by using keyword argument: **projection='3d'**

In [252]:

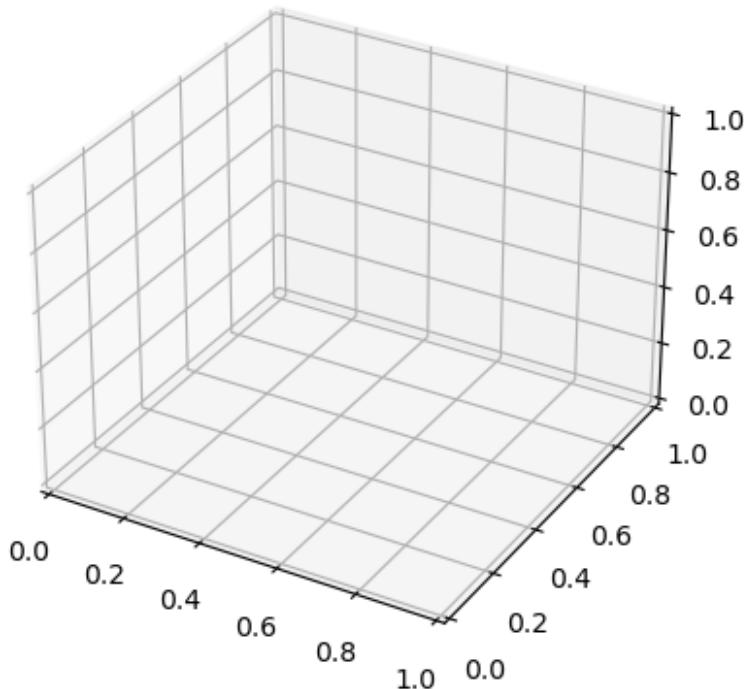
```
# 2-D axes object creation
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes()
plt.show()
```



---

In [253]:

```
# 3-D axes object creation
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection='3d')
plt.show()
```



### Creation of 3-D line plot

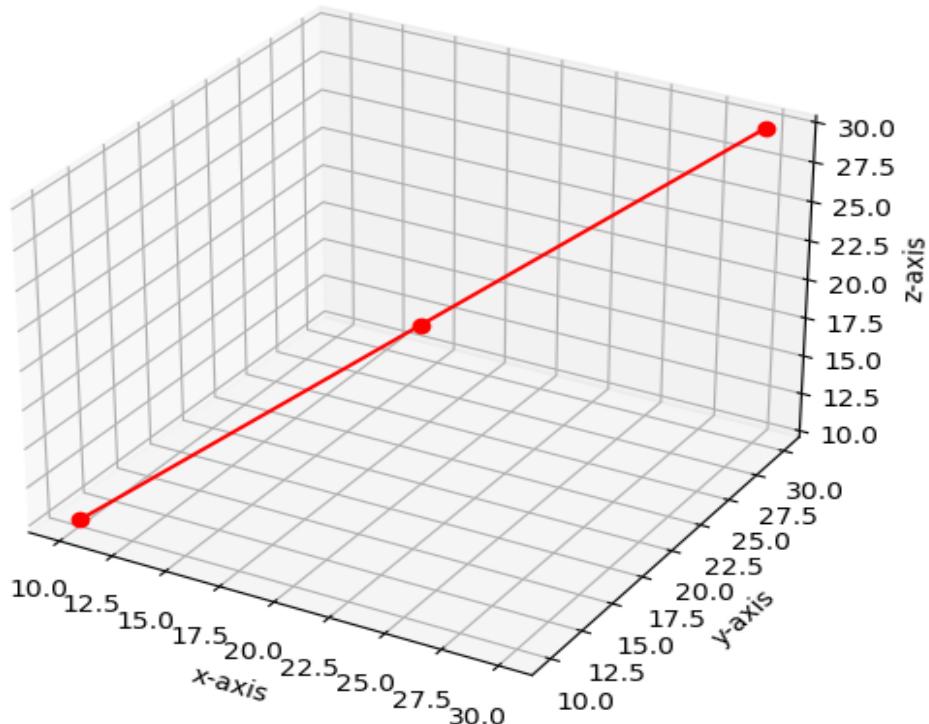
- ✓ We have to use **plot3D()** method.
  - ✓ To create 3-D plot, each data point should contains 3 coordinates(x,y,z)  
**ax.plot()** → To create 2-D plot  
**ax.plot3D()** → To create 3-D plot
-

---

In [254]:

```
import matplotlib.pyplot as plt
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = [10,20,30]
y = [10,20,30]
z = [10,20,30]
ax.plot3D(x,y,z,'ro-') #Data points are: (10,10,10),(20,20,20),(30,30,30)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Line Plot')
plt.show()
```

3-D Line Plot

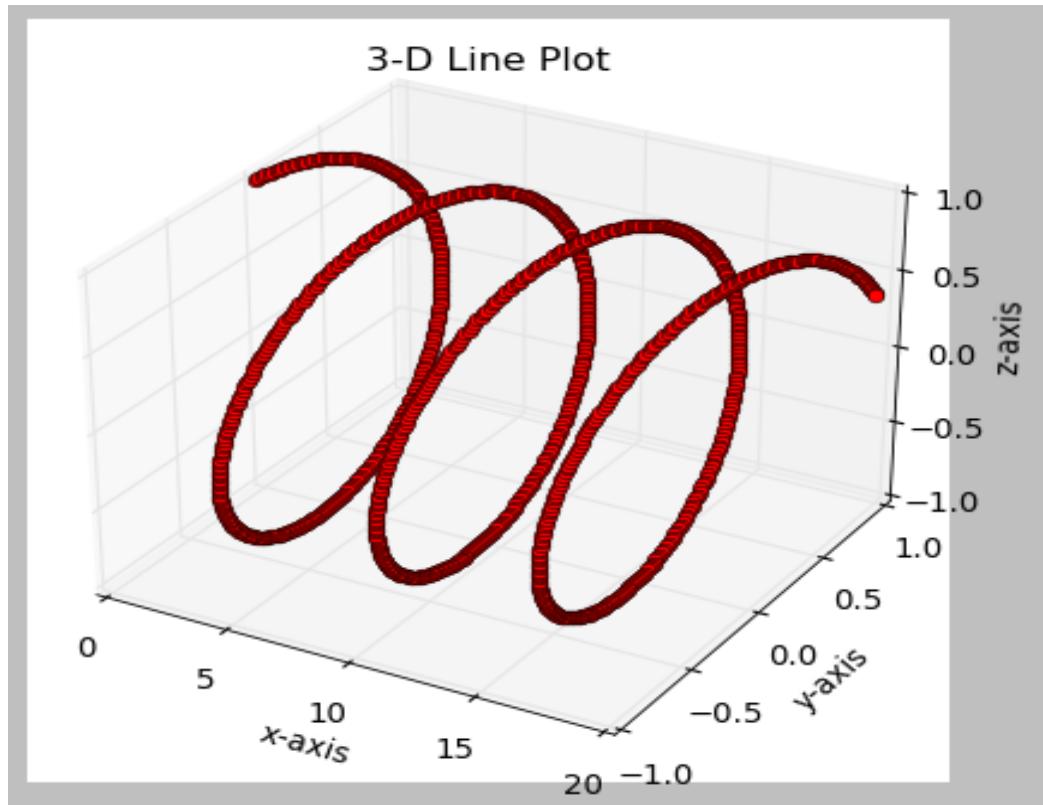


---

In [255]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('classic')

fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = np.linspace(0,20,1000)
y = np.sin(x)
z = np.cos(x)
ax.plot3D(x,y,z,'ro-') #Data points are: (10,10,10),(20,20,20),(30,30,30)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Line Plot')
plt.show()
```



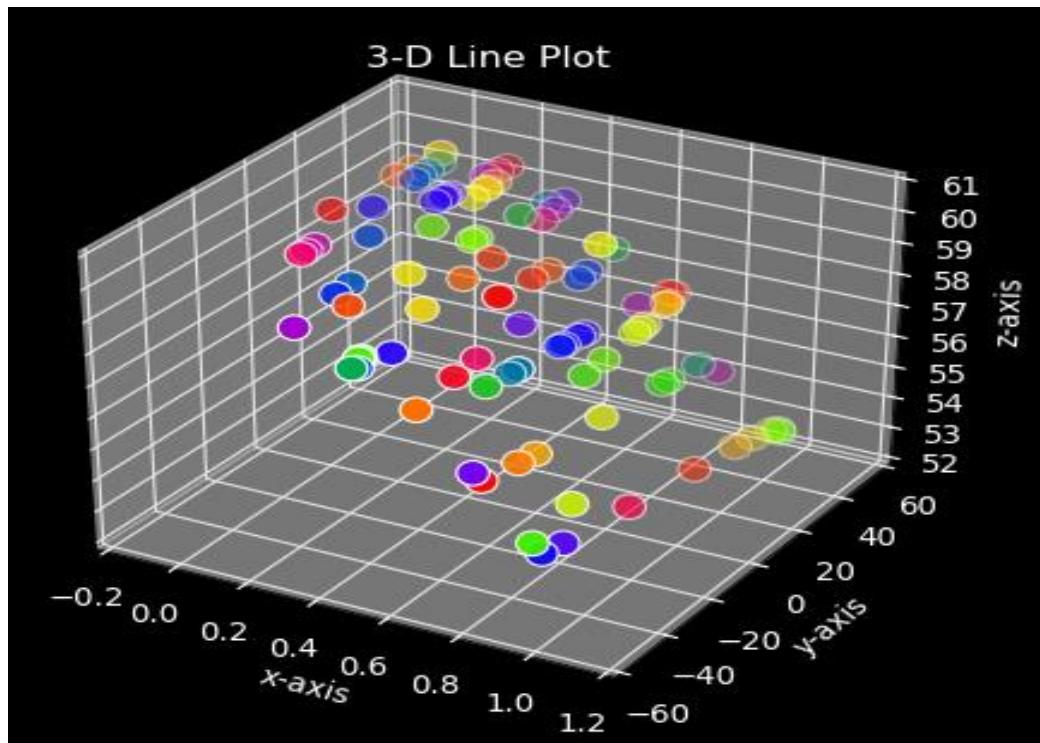
---

## Creation of 3-D Scatter plot:

We have to use **ax.scatter3D()** method.

In [256]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = np.random.rand(100)
y = 50*np.sin(40*x)
z = 60*np.cos(x/2)
ax.scatter3D(x,y,z,s=150,c=x,cmap='prism')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Line Plot')
plt.show()
```

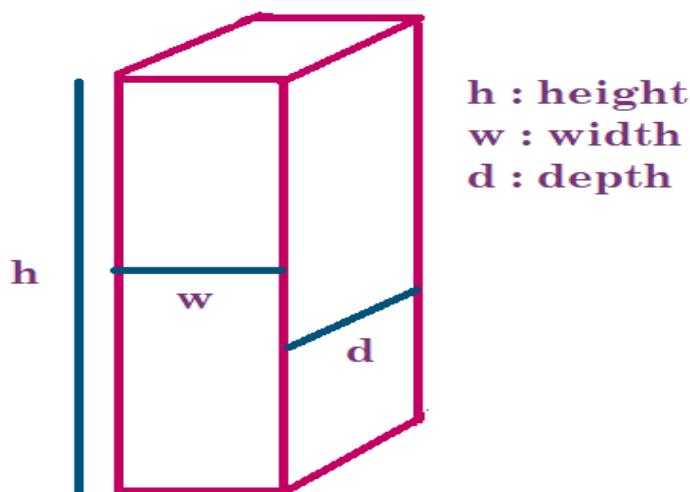


## **Creation of 3-D Bar charts:**

- ✓ 3-D Line Plot- → **ax.plot3D()**
- ✓ 3-D Scatter Plot → **ax.scatter3D()**
- ✓ 3-D Bar Chart → **ax.bar3d()**

### **6 arguments are required**

- ✓ x,y,z → position of bar
- ✓ dx → width of the bar
- ✓ dy → depth of the bar
- ✓ dz → height of the bar



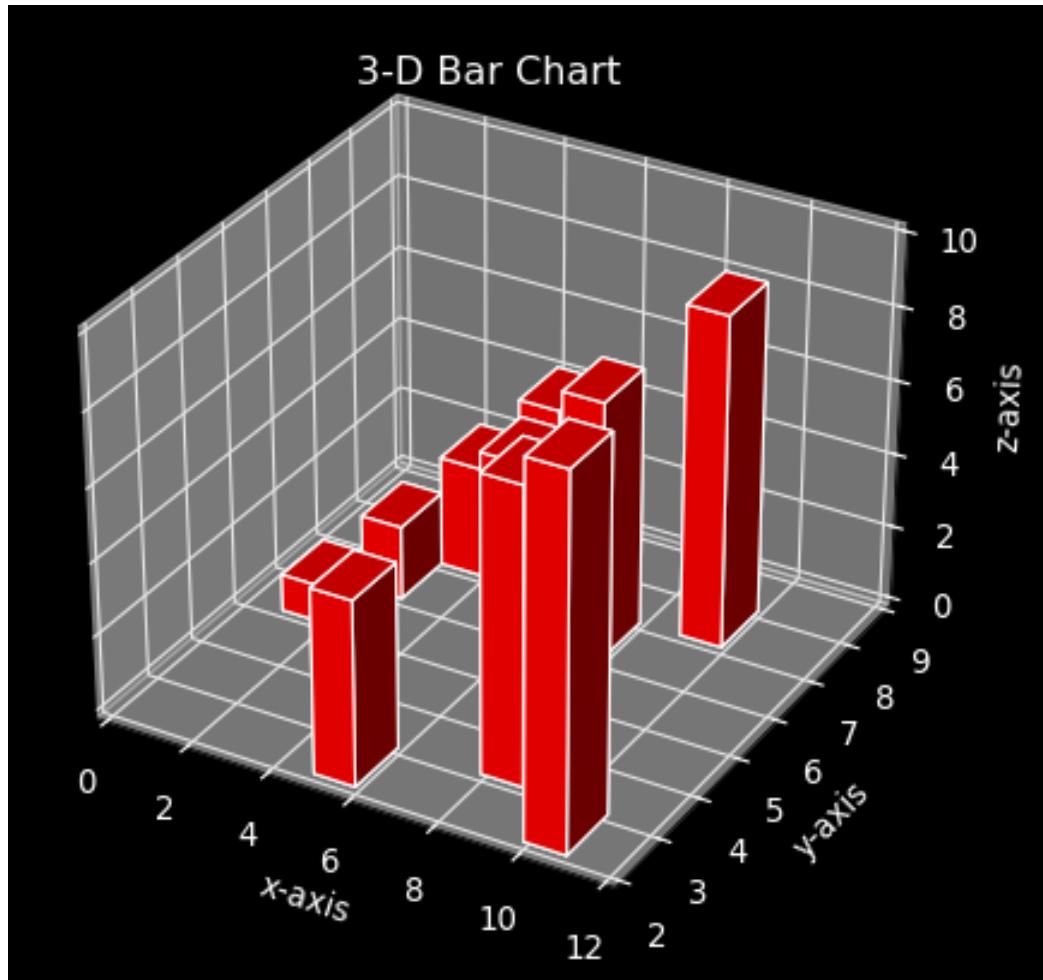
In [257]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = [1,2,3,4,5,6,7,8,9,10]
y = [5,6,7,8,2,5,6,3,7,2]
z = np.zeros(10) #The position of the first bar:(1,5,0)

dx = np.ones(10)
```

```
dy = np.ones(10)
dz = [1,2,3,4,5,6,7,8,9,10]

ax.bar3d(x,y,z,dx,dy,dz,color='red')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Bar Chart')
plt.show()
```



---

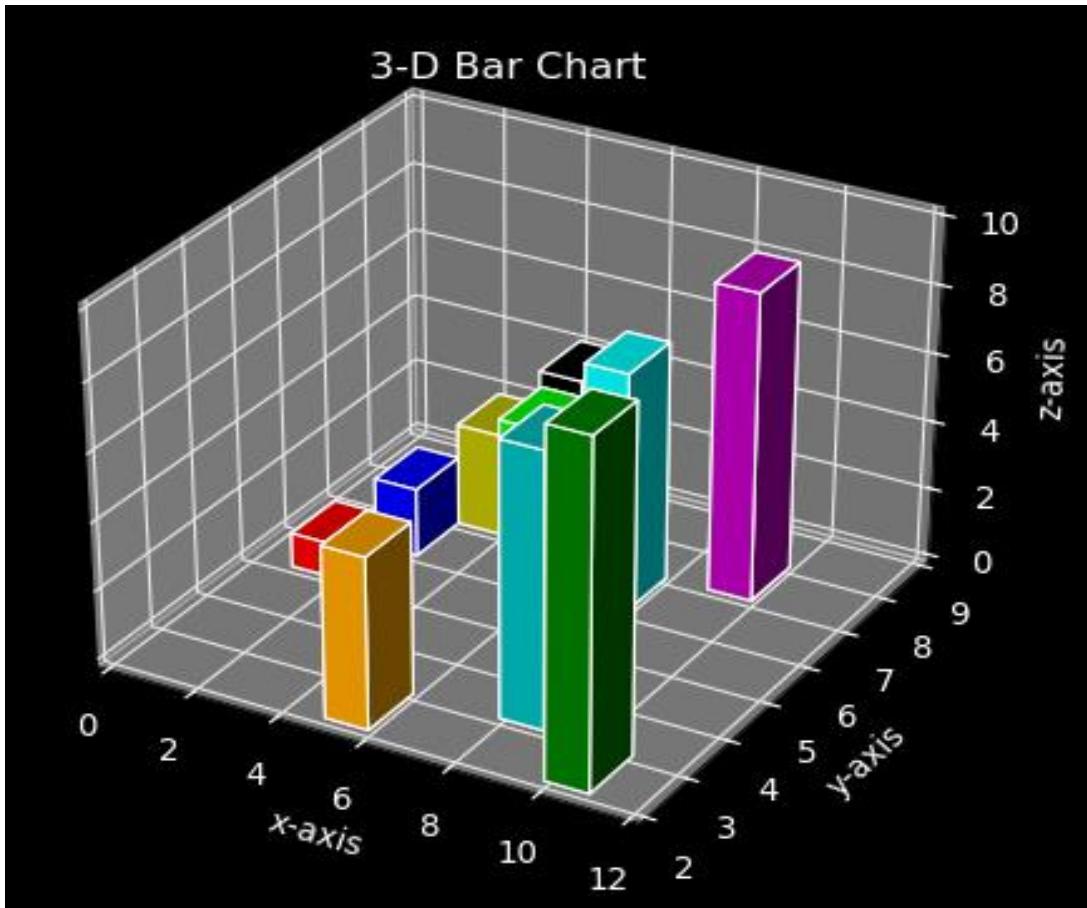
In [258]:

```
# 3-D bars with different colors

import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = [1,2,3,4,5,6,7,8,9,10]
y = [5,6,7,8,2,5,6,3,7,2]
z = np.zeros(10) #The position of the first bar:(1,5,0)

dx = np.ones(10)
dy = np.ones(10)
dz = [1,2,3,4,5,6,7,8,9,10]
colors = ['r','b','y','k','orange','lime','aqua','c','m','g']

ax.bar3d(x,y,z,dx,dy,dz,color=colors)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Bar Chart')
plt.show()
```



## Wireframes and surface plots

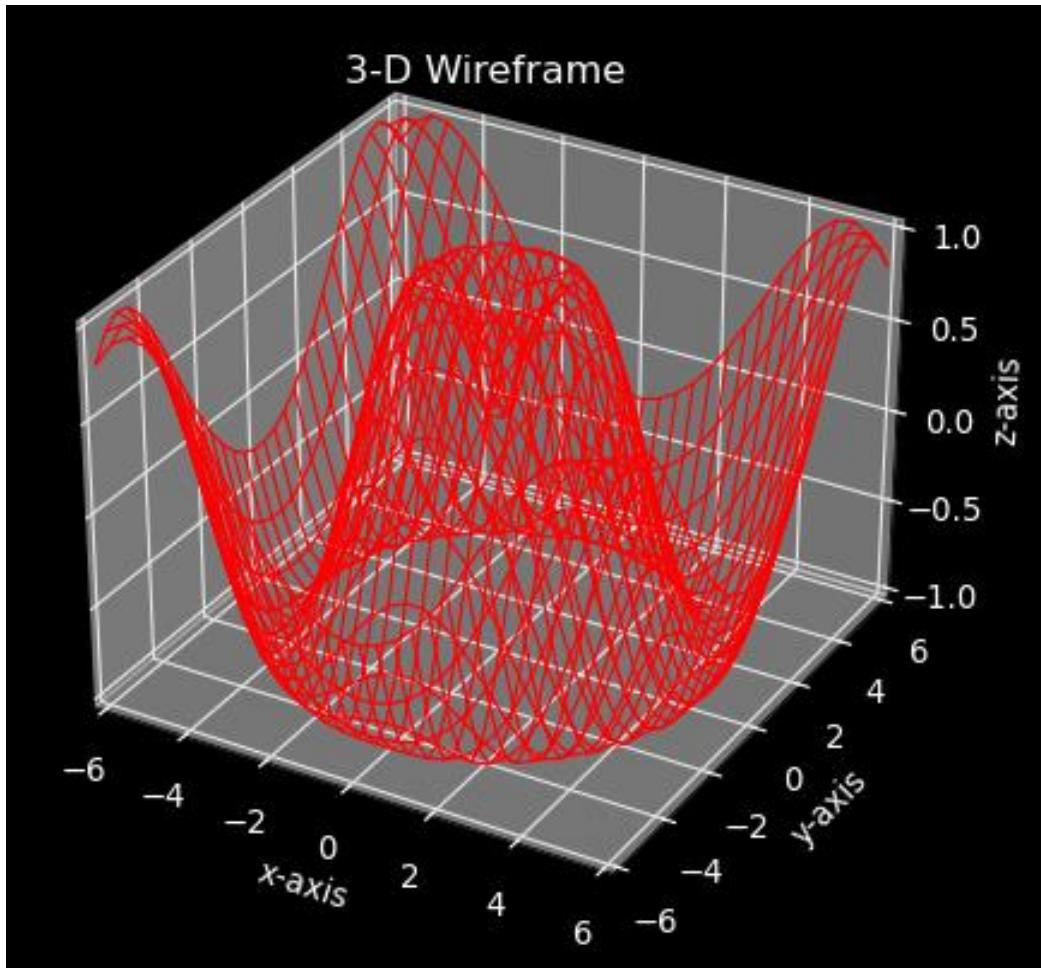
- ✓ These are the most commonly used 3-D plots that work on gridded data.
- ✓ These plots take a grid of values and project it onto the 3-D surface and can make resulting 3-D forms quite easy to visualize.

---

In [259]:

```
# wireframes
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')
x = np.linspace(-6,6,30)
y = np.linspace(-6,6,30)
X,Y = np.meshgrid(x,y)
Z = np.sin(np.sqrt(X**2+Y**2))

ax.plot_wireframe(X,Y,Z,color='r')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Wireframe')
plt.show()
```



## Surface plots

In [260]:

```
import matplotlib.pyplot as plt
import numpy as np
plt.style.use('dark_background')
fig = plt.figure(figsize=(12,6))
ax = plt.axes(projection='3d')

r = np.linspace(0,6,20)
theta = np.linspace(-0.9*np.pi,0.8*np.pi,40)
```

---

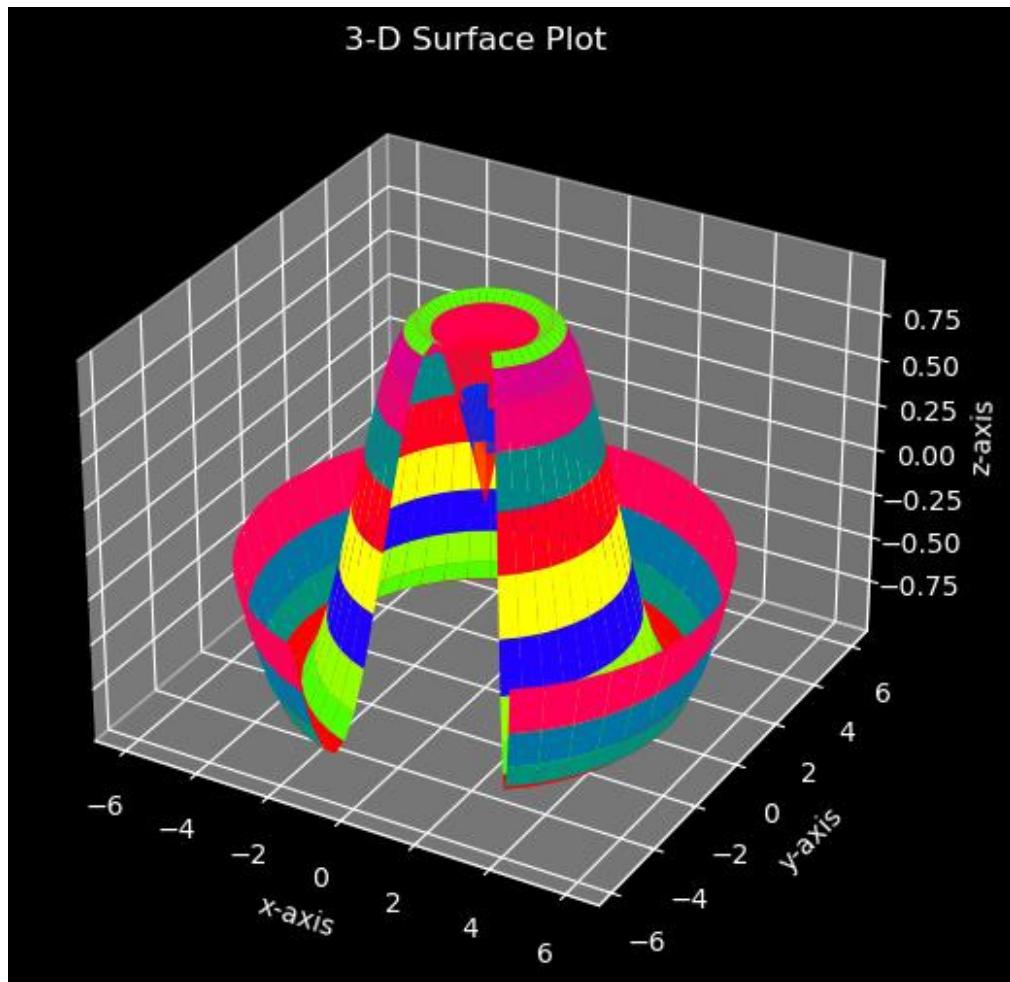
```

r,theta = np.meshgrid(r,theta)

X = r*np.sin(theta)
Y = r*np.cos(theta)
Z = np.sin(np.sqrt(X**2+Y**2))

ax.plot_surface(X,Y,Z,cmap='prism')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_title('3-D Surface Plot')
plt.show()

```



---

## Chapter-18 Animations

### Animations

- ✓ Animation is the technique of displaying images rapidly to generate illusion of movement. Each image is called a frame. If we display 24 frames per second, human eye perceives the animation as motion.
- ✓ **24 frames** is the most common word in movie world.
- ✓ We can create animations by using matplotlib. For this we have to use **animation module**.
- ✓ animation module contains **Animation class**, which acts as base class(parent class) to create animations.
- ✓ Animation contains 3 child classes:
  1. FuncAnimation
  2. TimedAnimation
  3. ArtistAnimation
- ✓ Most of the times we will use FuncAnimation class, where we can create animations by repeatedly calling a function.

In [261]:

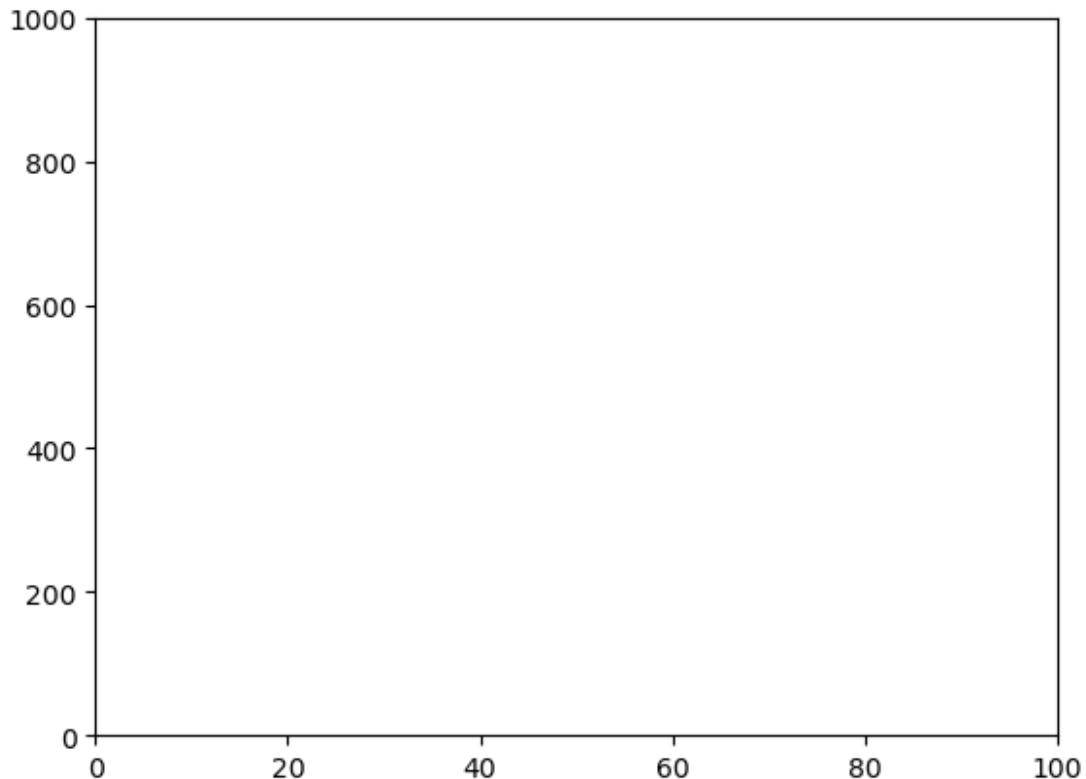
```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
print('x values ==> {x}')
print('y values ==> {y}')
plt.show()
```

```
x values ==> [ 0  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 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69
```

---

70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93  
94 95 96 97 98 99]

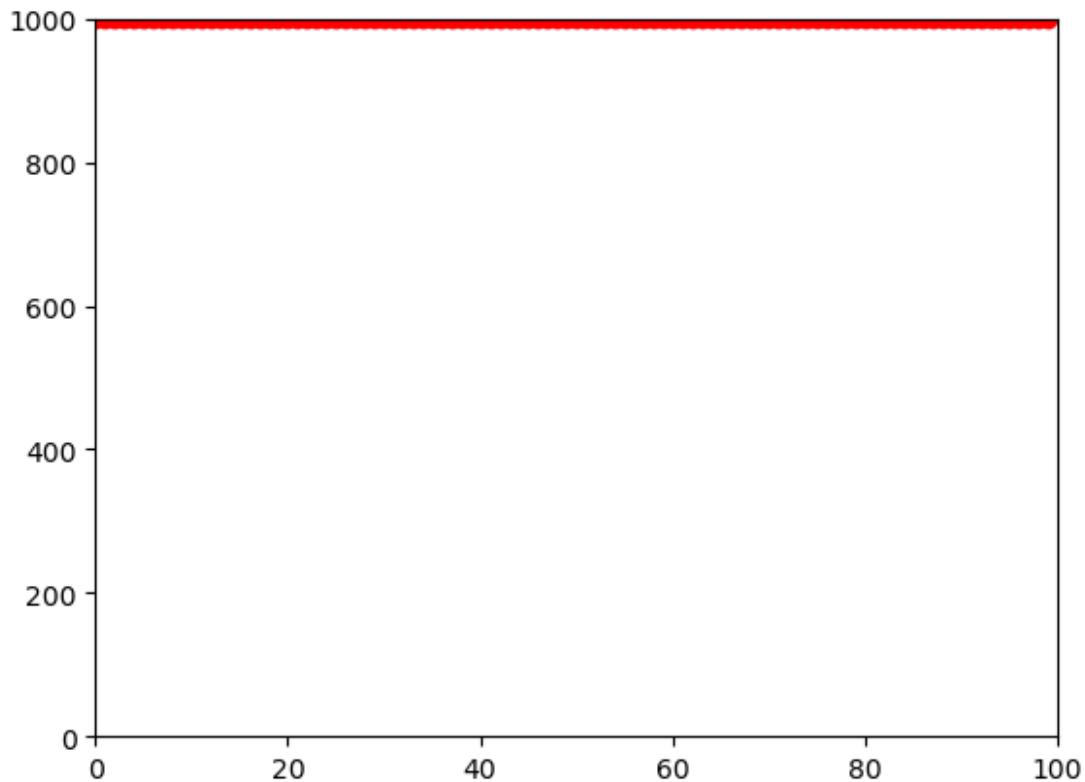
y values ==> [1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.  
1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000. 1000.]



---

In [262]:

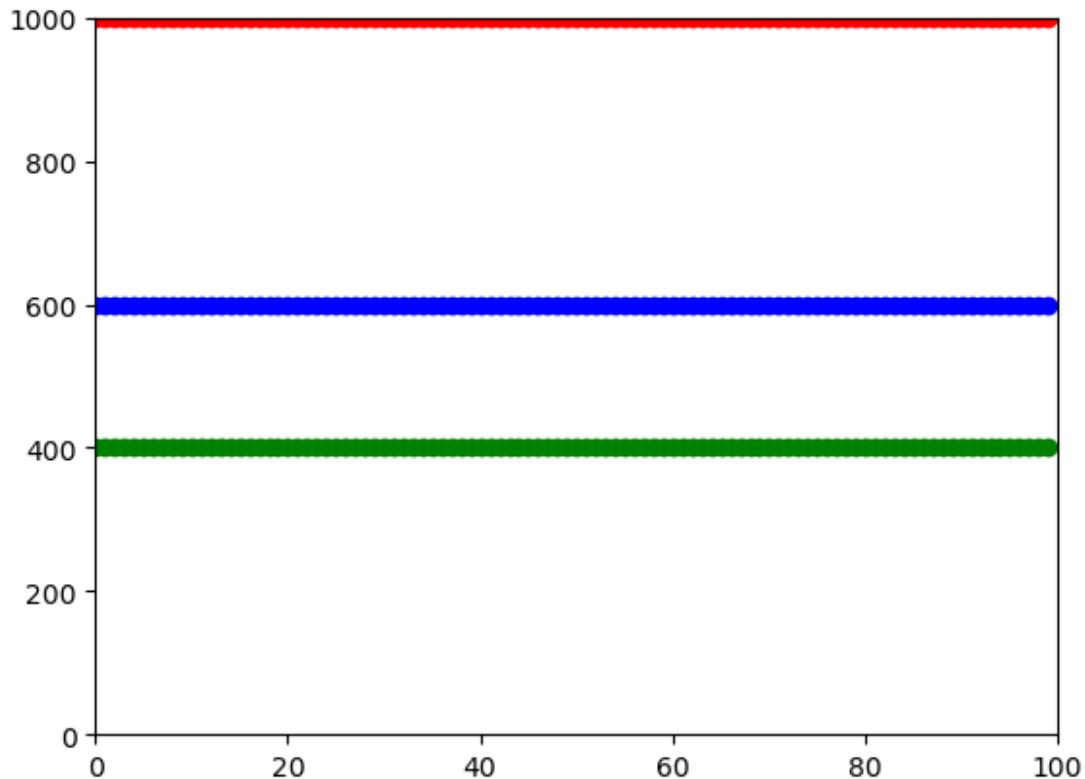
```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
ax.plot(x,y,'ro-')
plt.show()
```



---

In [263]:

```
# The following code generates 3 lines
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
ax.plot(x,y,'ro-')
ax.plot(x,y-400,'bo-')
ax.plot(x,y-600,'go-')
plt.show()
```



---

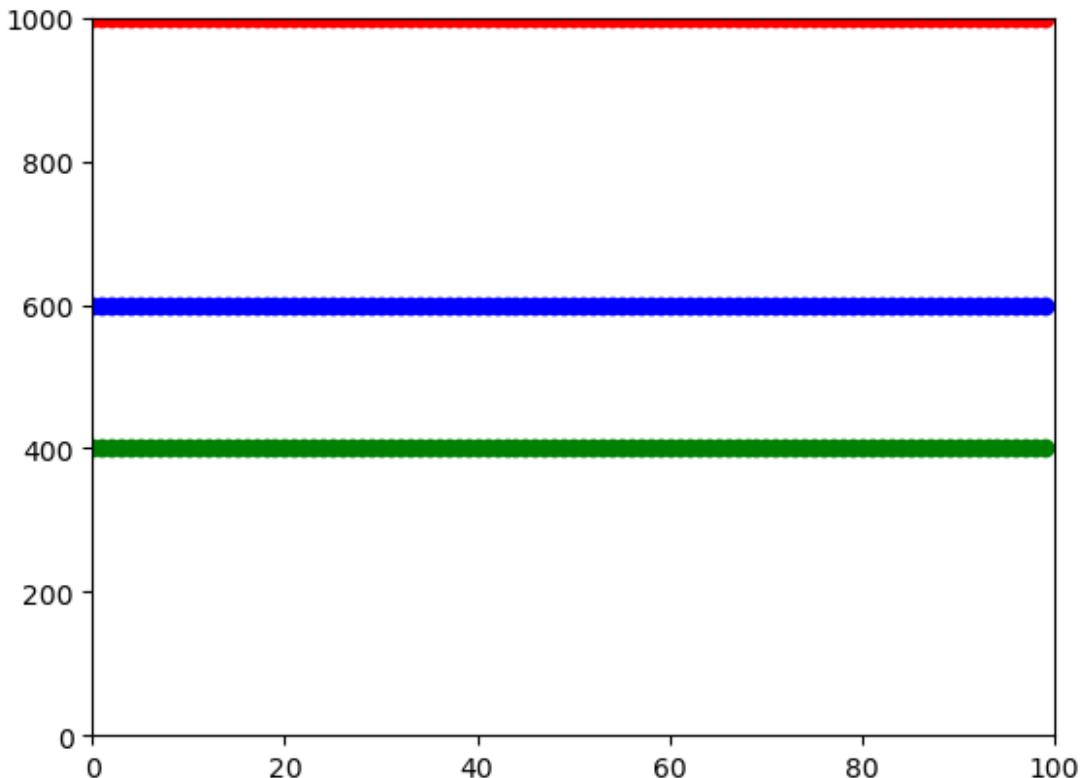
**Note:**

The return type of **plot() function** is **list object**

In [264]:

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
l = ax.plot(x,y,'ro-',x,y-400,'bo-',x,y-600,'go-')
print(f'Return type of plot() function is :{type(l)}')
print(l)
plt.show()
```

Return type of plot() function is :<class 'list'>  
[<matplotlib.lines.Line2D object at 0x000002032F29E910>,  
<matplotlib.lines.Line2D object at 0x000002032F29EA60>,  
<matplotlib.lines.Line2D object at 0x000002032F29EF70>]



In [265]:

```
l = [10]
print(l) # list object
11, = [10] # unpacking of list
print(11) # unpack list and assign first value of list to 11
```

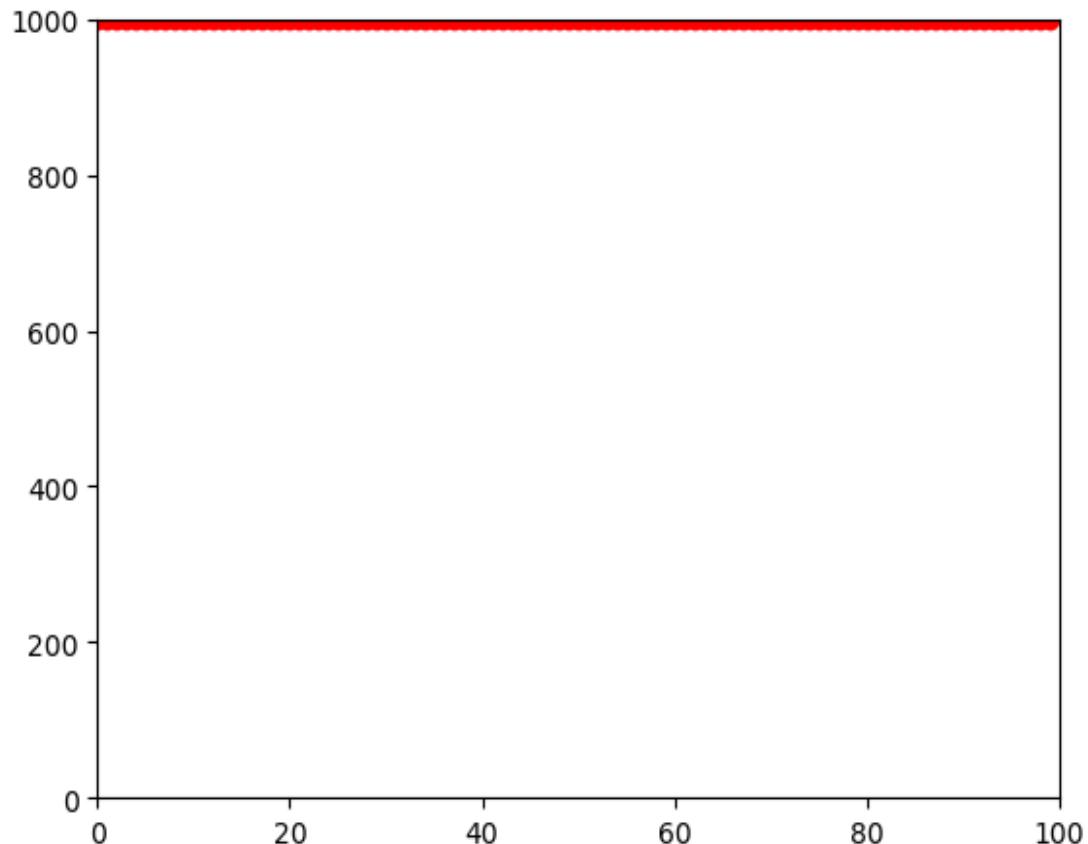
```
[10]
10
```

In [266]:

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
```

```
x = np.arange(100)
y = np.ones(100)*1000
line1, = ax.plot(x,y,'ro-') #unpack list and assign the first value to line1
print(line1)
```

Line2D(\_line0)

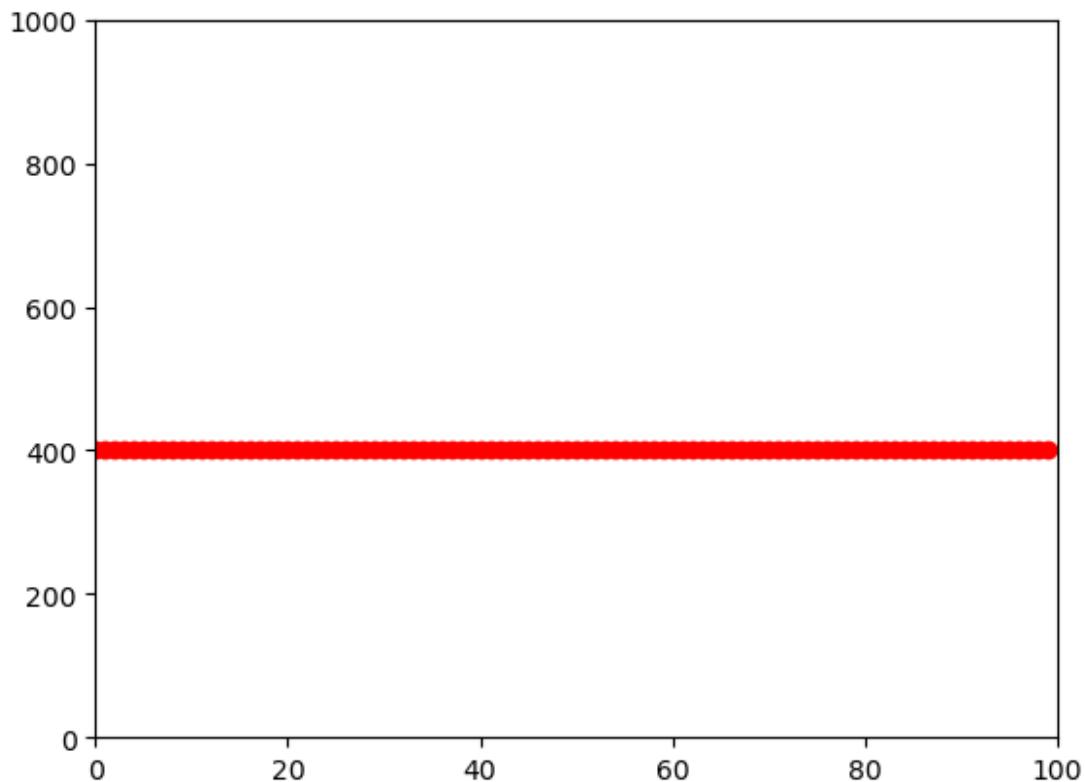


---

## Reuse the same line object with different data sets

In [267]:

```
import matplotlib.pyplot as plt
import numpy as np
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
line1, = ax.plot([],[],'ro-')
line1.set_data(x,y)
line1.set_data(x,y-400)
line1.set_data(x,y-600)
plt.show()
```



---

## Note

In the above program at a time only one line will be displayed and we are reusing the same line object.

## Demo program for falling line animation

In [268]:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.animation import FuncAnimation
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100)
y = np.ones(100)*1000
line1, = ax.plot([],[],'ro-')
def init():
    print('init function called...')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Falling Line Animation')
    global y
    y = np.ones(100)*1000
    line1.set_data([],[])
    return line1
def animate(i): # 0 to 999
    print('animate function called with i value:',i)
    global y
    y = y-1
    line1.set_data(x,y)
    return line1

anm = FuncAnimation(fig,animate,
                    init_func=init,
                    frames=1000,
                    interval=10,repeat=True)
plt.show()
```

---

## Line falling down and up animation:

In [269]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,100)
ax.set_ylim(0,1000)
x = np.arange(100) #[0 1 ... 99]
y = np.ones(100)*1000 #[1000 1000 1000 ... 1000]
line1, = ax.plot([],[],'ro-')
forward=True
def init():
    print('init function called')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Falling Line')
    line1.set_data([],[])
    return line1

def animate(i): # i values are from 0 to 999
    print('animate function called with i value:',i)
    global y
    global forward
    if y[0] in range(0,1001) and forward == True:
        y = y-1
        if y[0] <0:
            forward = False
    elif y[0] in range(-1,1001) and forward == False:
        y = y+1
        if y[0] == 1000:
            forward=True
    line1.set_data(x,y)
    return line1
anm =
FuncAnimation(fig,animate,init_func=init,frames=1000,interval=1,repeat=True)
plt.show()
```

---

## Demo Program for Growing Line Animation

In [270 ]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
plt.style.use('dark_background')
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(0,1000)
ax.set_ylim(0,1000)
line1, = ax.plot([],[],'yo-')
xdata,ydata=[],[]
def init():
    global xdata,ydata
    xdata,ydata=[],[]
    print('init function called')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Increasing Line Animation')
    line1.set_data([],[])
    return line1

def animate(i): # i values are from 0 to 999
    print('animate function called with i value:',i)
    xdata.append(i)
    ydata.append(i)
    line1.set_data(xdata,ydata)
    return line1

anm =
FuncAnimation(fig,animate,init_func=init,frames=1000,interval=1,repeat=True)
plt.show()
```

---

## Demo Program for Burning Coil Animation

In [271]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
plt.style.use('dark_background')
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(-100,100)
ax.set_ylim(-100,100)
line1, = ax.plot([],[],color='y',lw=2)
xdata,ydata=[],[]
def init():
    global xdata,ydata
    xdata,ydata=[],[]
    print('init function called')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Burning Coil Animation')
    line1.set_data([],[])
    return line1

def animate(i): # i values are from 0 to 999
    print('animate function called with i value:',i)
    t = 0.1*i
    x = t*np.sin(t)
    y = t*np.cos(t)
    xdata.append(x)
    ydata.append(y)
    line1.set_data(xdata,ydata)
    return line1

anm =
FuncAnimation(fig,animate,init_func=init,frames=1000,interval=1,repeat=True)
plt.show()
```

---

## How to save animations to a file

We have to use **save()** method

In [272]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
plt.style.use('dark_background')
fig = plt.figure()
ax = plt.subplot(111)
ax.set_xlim(-100,100)
ax.set_ylim(-100,100)
line1, = ax.plot([],[],color='y',lw=2)
xdata,ydata=[],[]
def init():
    global xdata,ydata
    xdata,ydata=[],[]
    print('init function called')
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_title('Burning Coil Animation')
    line1.set_data([],[])
    return line1

def animate(i): # i values are from 0 to 999
    print('animate function called with i value:',i)
    t = 0.1*i
    x = t*np.sin(t)
    y = t*np.cos(t)
    xdata.append(x)
    ydata.append(y)
    line1.set_data(xdata,ydata)
    return line1

anm =
FuncAnimation(fig,animate,init_func=init,frames=500,interval=10,repeat=False)
anm.save('burningcoil.gif')
```

---

## Note

The specified file will be created in the current working directory. But we can specify other location by using absolute path.

## Rain Simulation from matplotlib documentation

In [273]:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Create new Figure and an Axes which fills it.
fig = plt.figure(figsize=(7, 7))
ax = fig.add_axes([0, 0, 1, 1], frameon=False)
ax.set_xlim(0, 1), ax.set_xticks([])
ax.set_ylim(0, 1), ax.set_yticks([])

# Create rain data
n_drops = 50
rain_drops = np.zeros(n_drops, dtype=[('position', float, 2),
                                      ('size',    float, 1),
                                      ('growth',  float, 1),
                                      ('color',   float, 4)])

# Initialize the raindrops in random positions and with
# random growth rates.
rain_drops['position'] = np.random.uniform(0, 1, (n_drops, 2))
rain_drops['growth'] = np.random.uniform(50, 200, n_drops)

# Construct the scatter which we will update during animation
# as the raindrops develop.
scat = ax.scatter(rain_drops['position'][:, 0], rain_drops['position'][:, 1],
                  s=rain_drops['size'], lw=0.5, edgecolors=rain_drops['color'],
                  facecolors='none')

def update(frame_number):
```

---

```
# Get an index which we can use to re-spawn the oldest raindrop.
current_index = frame_number % n_drops

# Make all colors more transparent as time progresses.
rain_drops['color'][ :, 3] -= 1.0/len(rain_drops)
rain_drops['color'][ :, 3] = np.clip(rain_drops['color'][ :, 3], 0, 1)

# Make all circles bigger.
rain_drops['size'] += rain_drops['growth']

# Pick a new position for oldest rain drop, resetting its size,
# color and growth factor.
rain_drops['position'][current_index] = np.random.uniform(0, 1, 2)
rain_drops['size'][current_index] = 5
rain_drops['color'][current_index] = (0, 0, 0, 1)
rain_drops['growth'][current_index] = np.random.uniform(50, 200)

# Update the scatter collection, with the new colors, sizes and positions.
scat.set_edgecolors(rain_drops['color'])
scat.set_sizes(rain_drops['size'])
scat.set_offsets(rain_drops['position'])

# Construct the animation, using the update function as the animation
# director.
animation = FuncAnimation(fig, update, interval=10)
plt.show()
```