Part-IV-A Matplotlib Practice

March 1, 2025

1 Basic Programming with Matplotlib

1.0.1 Matplotlib

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

Orgainzed in hierarchy

top-level matplotlib.pyplot module is present

pyplot is used for few activity such as figure creation,

Through the created figure, one or more axes/subplot object are created

Axes object can further use for doing many plotting actions.

1.1 A few basic utilty methods

Function/Method	Description	Example Usage
`plt.figure()`	Creates a new figure for plotting.	`fig = plt.figure(figsize=(8, 6))`
`plt.show()`	Displays the current figure.	`plt.show()`
`add_subplot()`	Adds an axes (subplot) to the figure.	`ax = fig.add_subplot(1, 1, 1)`
`set()`	Sets multiple properties of an Axes object at once.	`ax.set(title="Figure", xlabel="X", ylabel="Y")`
`set_title()`	Sets the title of the plot.	`ax.set_title("Figure")`
`set_xlabel()`	Sets the label for the x-axis.	`ax.set_xlabel("X-Axis")`
`set_ylabel()`	Sets the label for the y-axis.	`ax.set_ylabel("Y-Axis")`
`set_xlim()`	Sets the limits for the x-axis.	`ax.set_xlim(0, 10)`
`set_ylim()`	Sets the limits for the y-axis.	`ax.set_ylim(0, 20)`

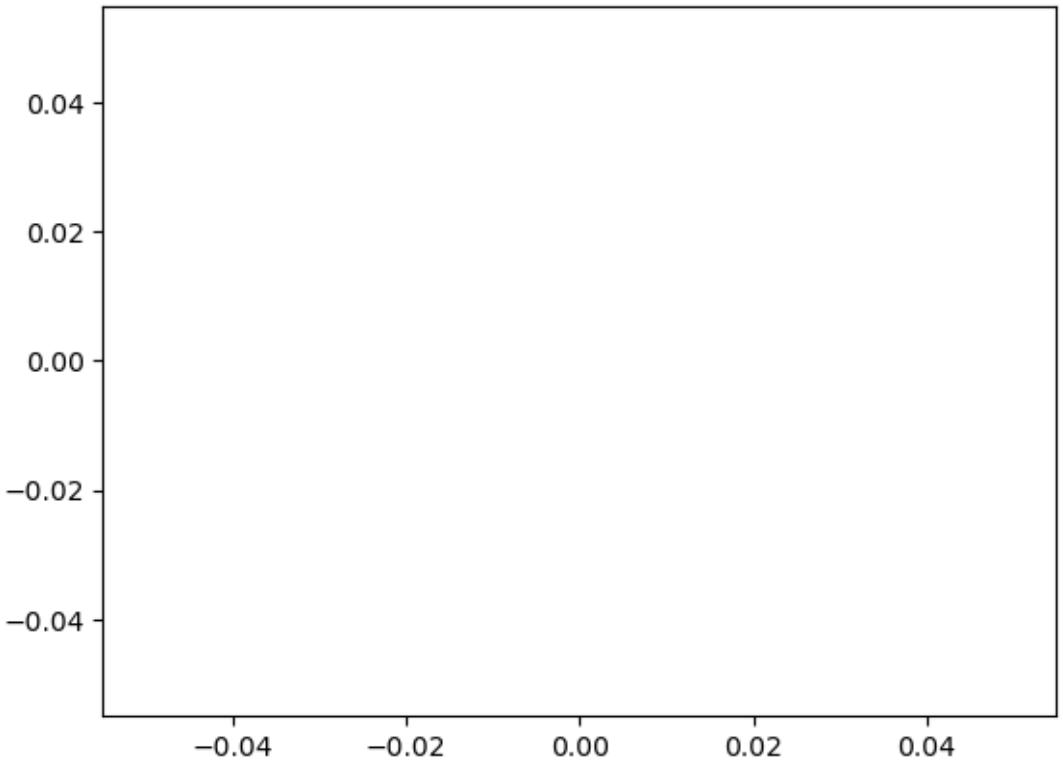
Function/Method	Description	Example Usage
`set_xticks()`	Sets the positions of ticks on the x-axis.	`ax.set_xticks([0, 1, 2, 3, 4])`
`set_xticklabels()`	Sets custom labels for the x-axis ticks.	`ax.set_xticklabels(['A', 'B', 'C', 'D', 'E'])`
`plt.plot()`	Plots a line graph.	`plt.plot(x, y, 'b--', label='Line 1')`
`plt.scatter()`	Creates a scatter plot.	`plt.scatter(x, y, color='blue', label='Group 1')`
`plt.bar()`	Creates a vertical bar plot.	`ax.bar(categories, values, color='skyblue')`
`plt.barh()`	Creates a horizontal bar plot.	`ax.barh(categories, values, color='skyblue')`
`plt.pie()`	Creates a pie chart.	`plt.pie(x, labels=labels, autopct='%1.1f%%')`
`plt.hist()`	Creates a histogram to show data distribution.	`ax.hist(data, bins=5, color='blue')`

1.2 Intilaization of programming environment

[6]:
```
# Import the following as a one-time job...

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

[ ]:
```
## Example 1:
### Drawing a simple plot...Just have nothing!
```

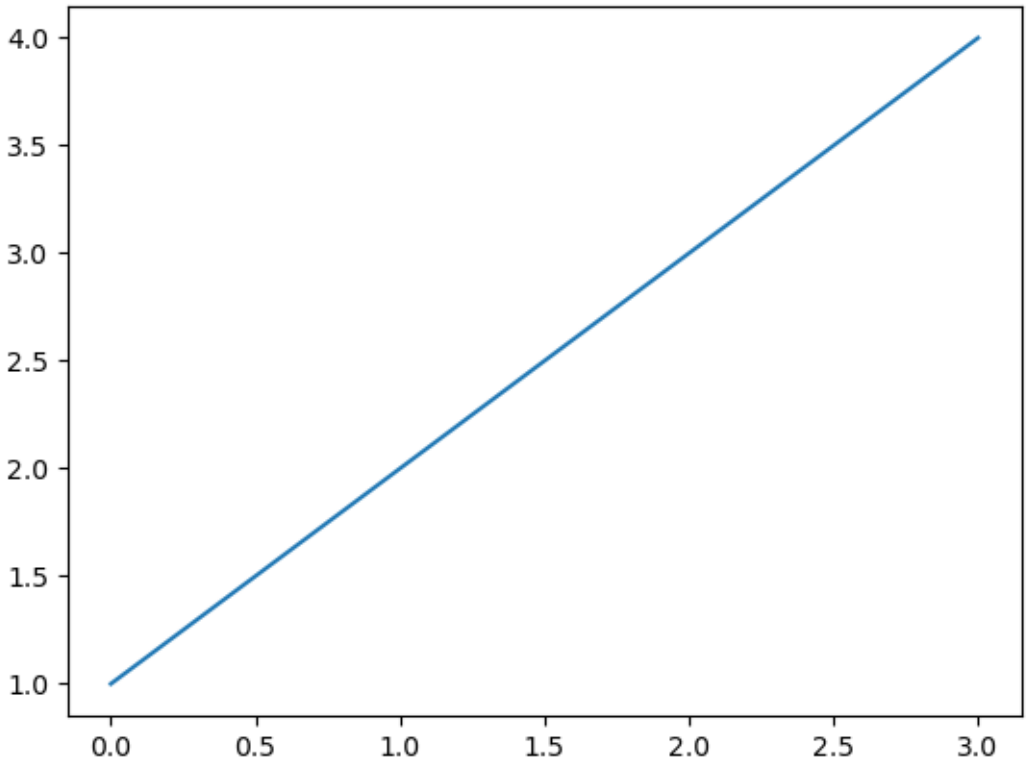
[10]:
```
plt.plot()                    # Call the plot() without any parameter
plt.show()                    # Show the graph
```



1.3 Example 2:

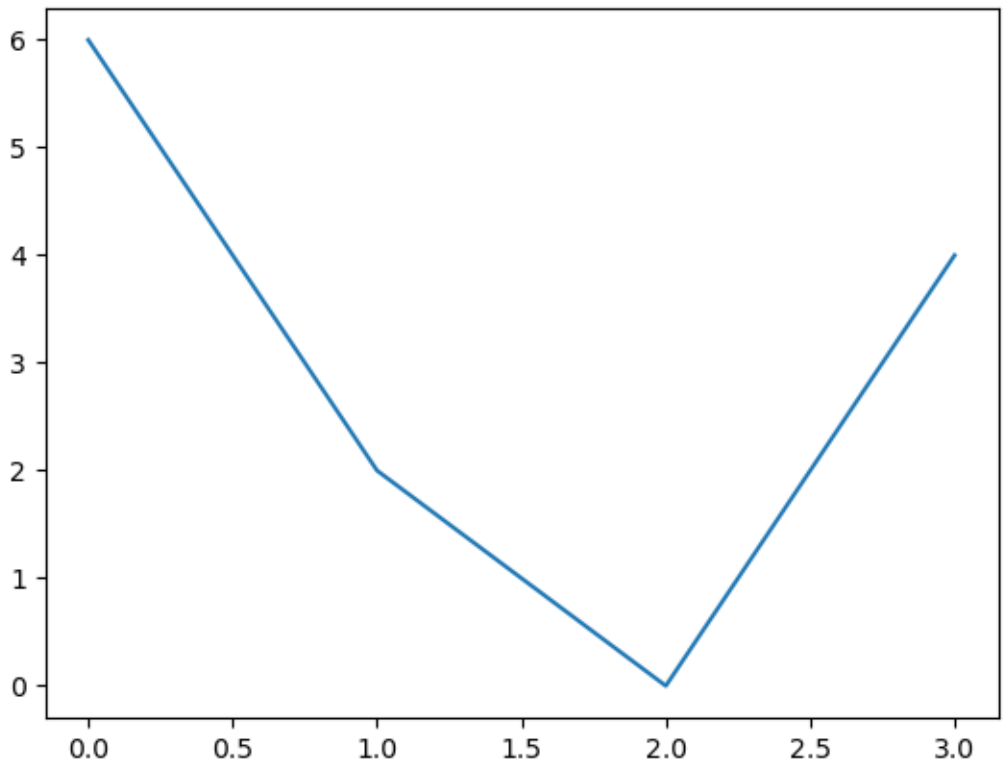
1.3.1 Drawing a simple plot...Given the values of y's while x's are default

[6]:
```
plt.plot([1,2,3,4])              # Default values are the values on y-axis
                                 # x-axis is [0, 1, 2, 3] set automatically with␣
 ↪the same range as y-axis but start with 0
plt.show()
```



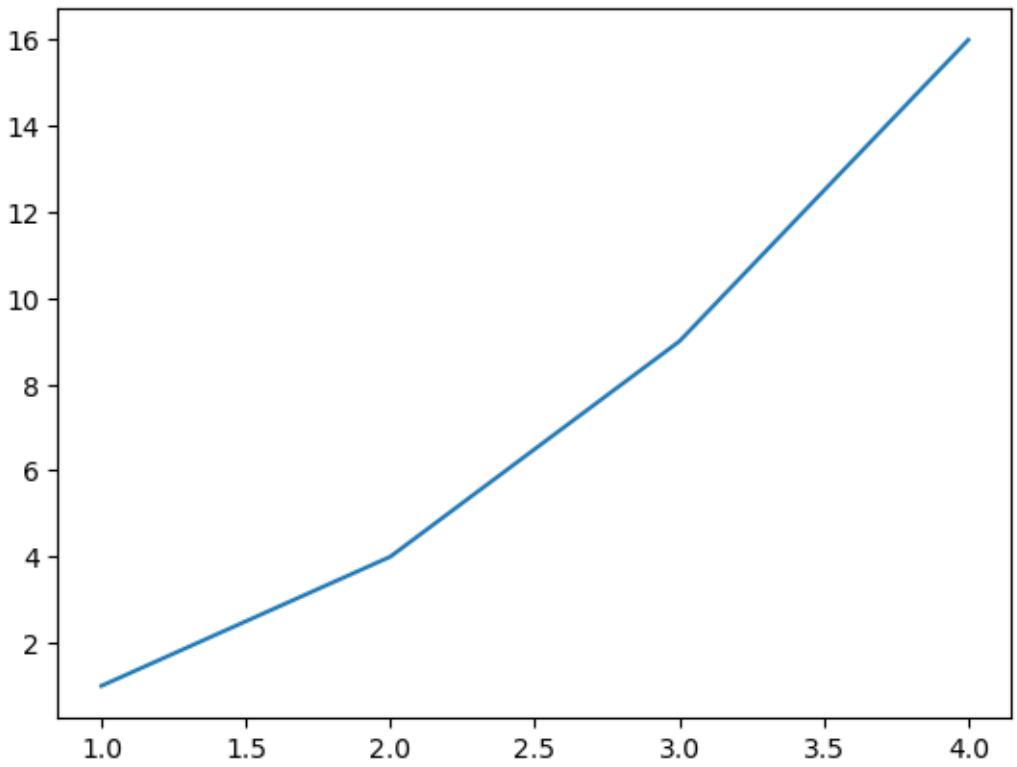
1.4 Drawing a simple plot... Given another set of y-values, but in random order

[4]:
```
import matplotlib.pyplot as plt

plt.plot([6,2,0,4])             # Default values are the values on y-axis,␣
 ↪x-axis is [0, 1, 2, 3]
plt.show()
```



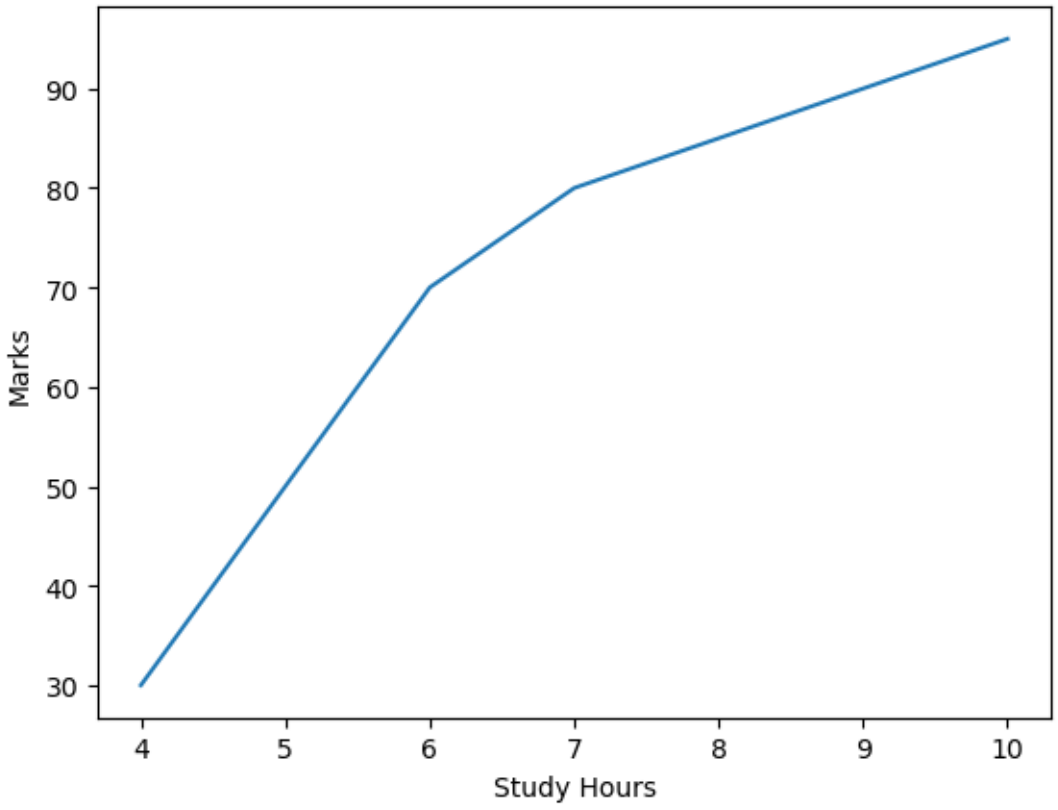
1.5 Draw a simple plot...Given both the y- and x-values: The first argument is x- and second is y-axis values

[11]:
```
import matplotlib.pyplot as plt

plt.plot([1,2,3,4], [1, 4, 9, 16])            # Default values are the values␣
 ↪on y-axis, x-axis is [0, 1, 2, 3]
plt.show()
```



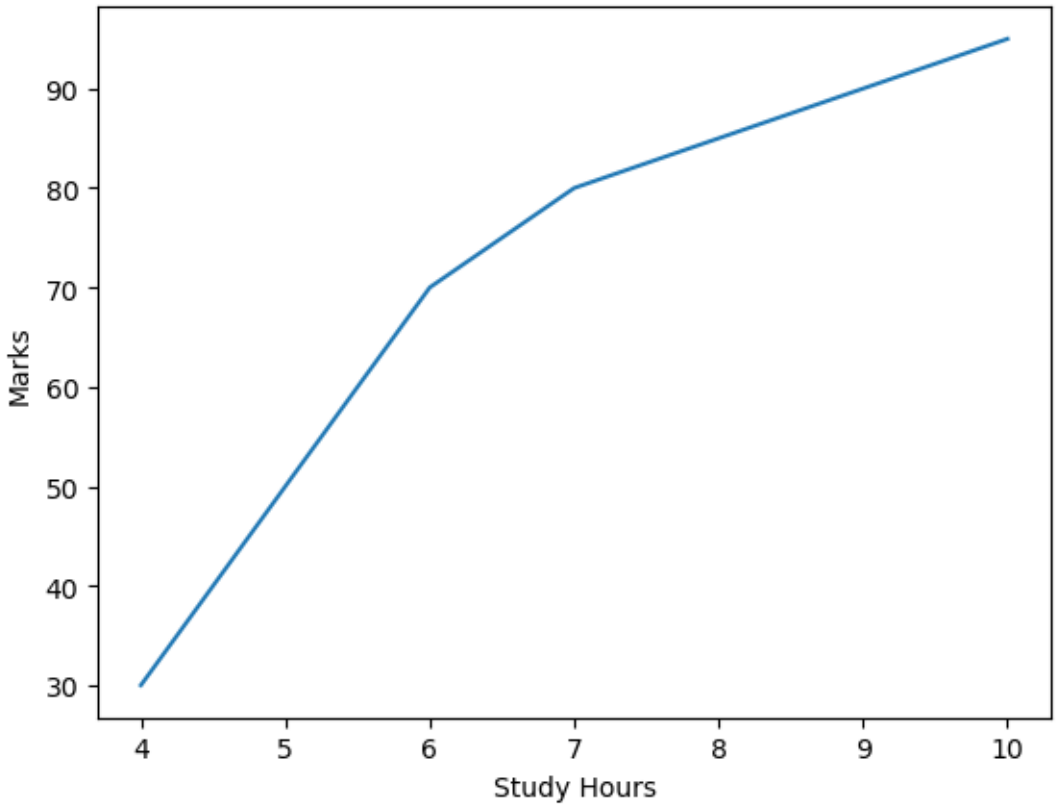
1.6 Draw a simple plot...Setting the labels of x- and y-axis

[23]:
```
import matplotlib.pyplot as plt

plt.plot([4, 5, 6, 7, 8, 9, 10], [30, 50, 70, 80, 85, 90, 95])              #␣
 ↪Default values are the values on y-axis, x-axis is [0, 1, 2, 3]
plt.ylabel('Marks')                                 # You may label your y-axis:␣
 ↪optional
plt.xlabel('Study Hours')                                  # You may label your x-axis:
 ↪ optional
plt.show()
```



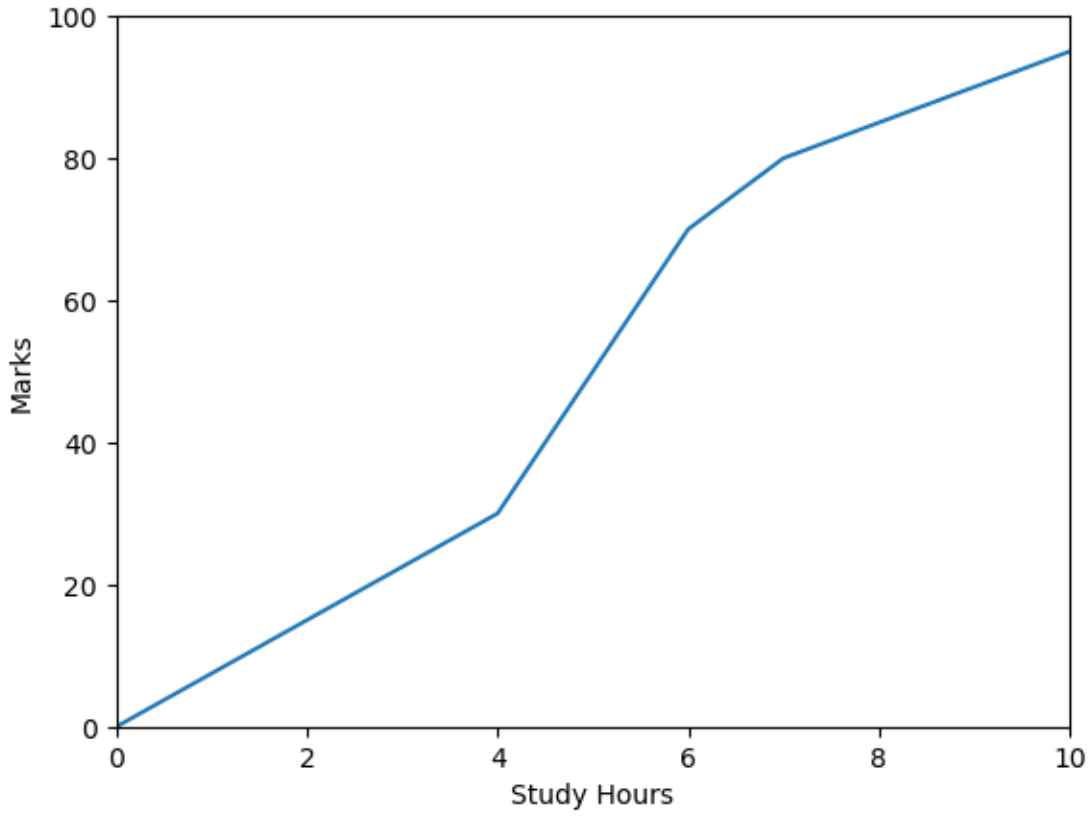
1.7 Draw a simple plot...Giving the values alternatively

[25]:
```
import matplotlib.pyplot as plt

marks = [4, 5, 6, 7, 8, 9, 10]                  # Values for x-axis
studyHours = [30, 50, 70, 80, 85, 90, 95]       # Values for y-axis

plt.plot(marks, studyHours)
plt.ylabel('Marks')
plt.xlabel('Study Hours')
plt.show()
```



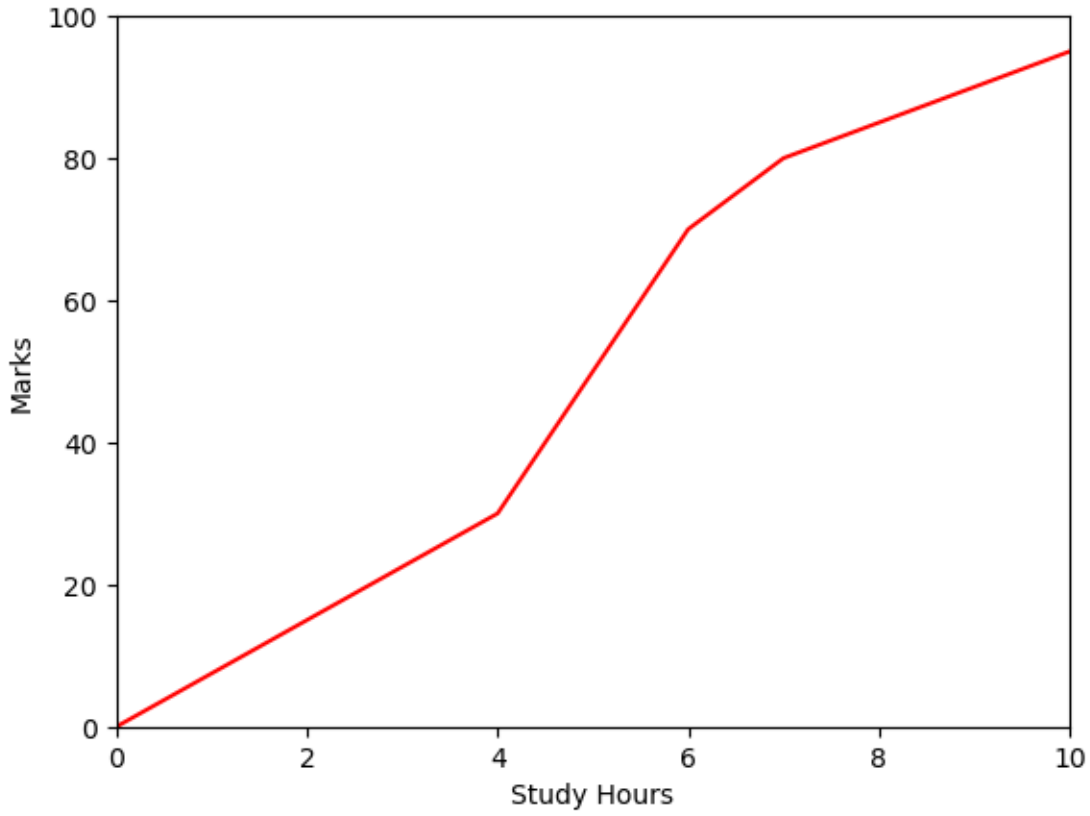
1.8 Draw a simple plot...Giving the values with different ranges of values

[37]:
```
import matplotlib.pyplot as plt

marks = [0, 4, 5, 6, 7, 8, 9, 10]                    # Values for x-axis
studyHours = [0, 30, 50, 70, 80, 85, 90, 95]         # Values for y-axis

plt.plot(marks, studyHours)
plt.axis([0, 10, 0, 100])                            # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.ylabel('Marks')
plt.xlabel('Study Hours')
plt.show()
```



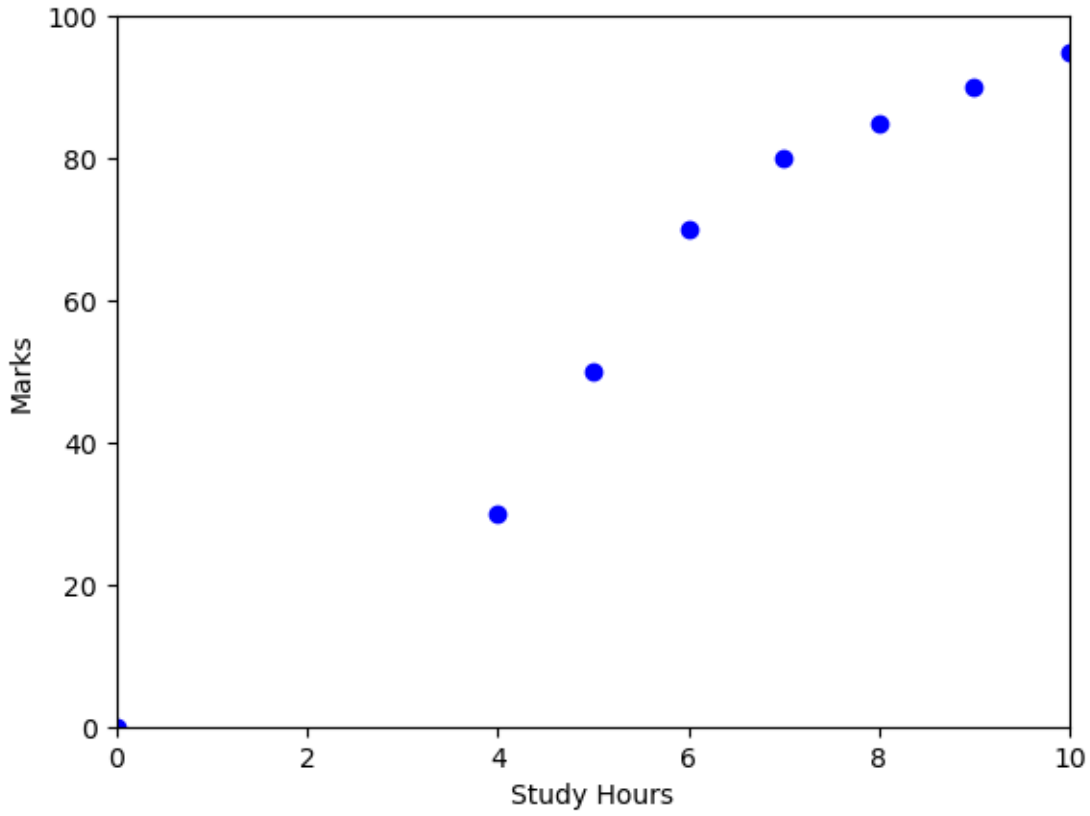
1.9 Drawing simple plot with different colors, line style, etc.

[15]:
```
# The plot() is a versatile command that can take many arguments! Setting Line␣
 ↪color

import matplotlib.pyplot as plt

marks = [0, 4, 5, 6, 7, 8, 9, 10]                    # Values for x-axis
studyHours = [0, 30, 50, 70, 80, 85, 90, 95]         # Values for y-axis

plt.plot(marks, studyHours, 'r-')                    # Here, 'ro' stands for the␣
 ↪red-filled circles, others are: 'bo', 'go', 'r-', for examples
plt.axis([0, 10, 0, 100])                            # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.ylabel('Marks')
plt.xlabel('Study Hours')
plt.show()
```



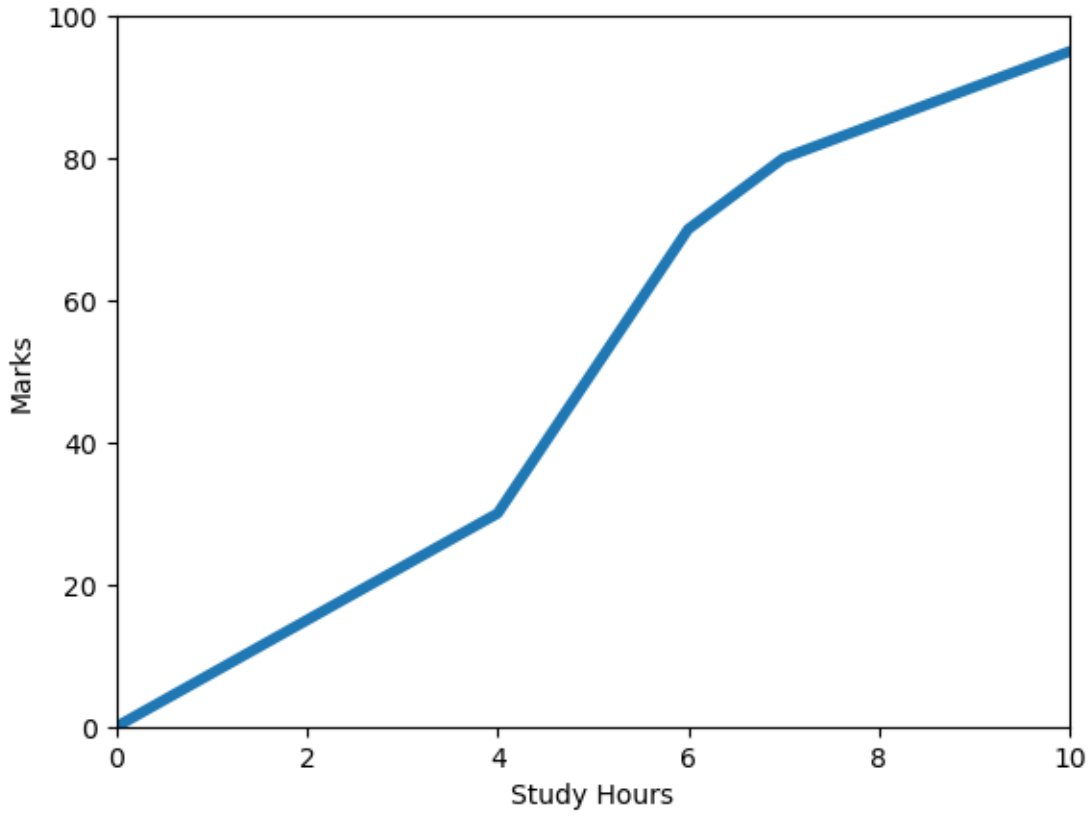
1.10 Setting plot items: Line style is circular dots

[19]:
```
import matplotlib.pyplot as plt

marks = [0, 4, 5, 6, 7, 8, 9, 10]                    # Values for x-axis
studyHours = [0, 30, 50, 70, 80, 85, 90, 95]         # Values for y-axis

plt.plot(marks, studyHours, 'bo')                    # Here, 'ro' stands for the␣
 ↪red-filled circles, others are: 'bo', 'go', 'r-', for examples
plt.axis([0, 10, 0, 100])                            # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.ylabel('Marks')
plt.xlabel('Study Hours')
plt.show()
```



1.11 Setting line width

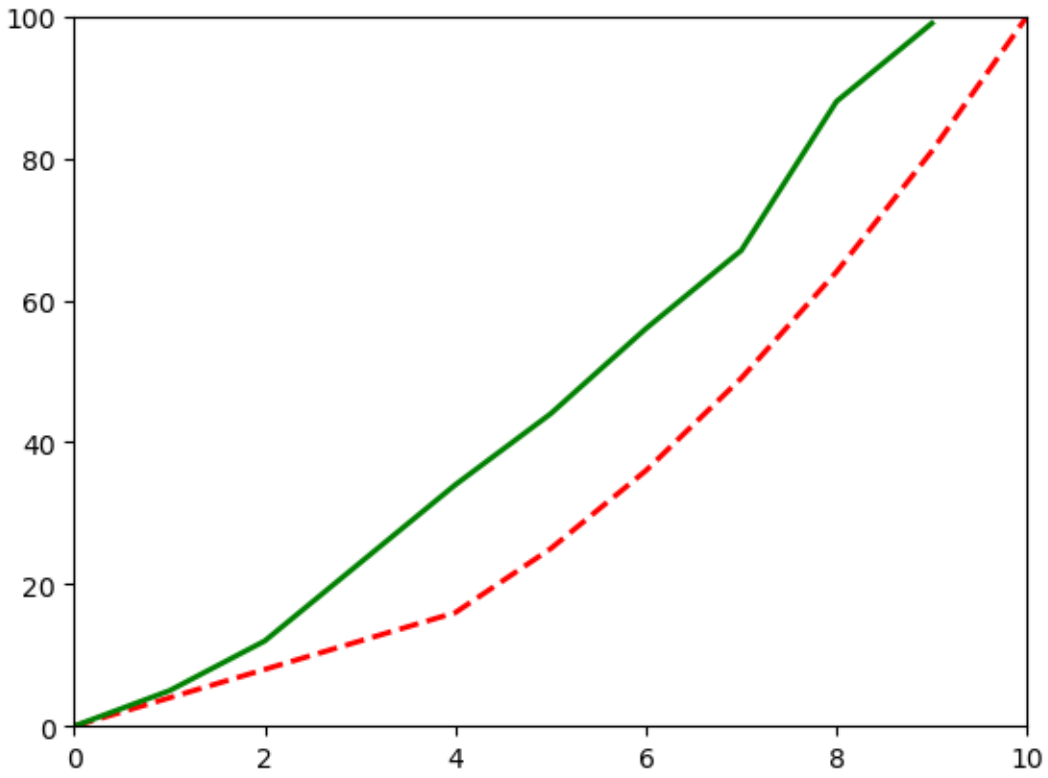
[27]:
```
import matplotlib.pyplot as plt

marks = [0, 4, 5, 6, 7, 8, 9, 10]                    # Values for x-axis
studyHours = [0, 30, 50, 70, 80, 85, 90, 95]         # Values for y-axis

plt.plot(marks, studyHours, linewidth=4.0)                  # Here, 'ro' stands␣
 ↪for the red-filled circles, others are: 'bo', 'go', 'r-', for examples
plt.axis([0, 10, 0, 100])                             # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.ylabel('Marks')
plt.xlabel('Study Hours')
plt.show()
```



1.12 Drawing more than one lines on the same graph

[41]:
```
import matplotlib.pyplot as plt

x1 = [0, 4, 5, 6, 7, 8, 9, 10]                  # x-attributes for line1
y1 = [0, 16, 25, 36, 49, 64, 81, 100]            # y-attribute for line 2
line1 = plt.plot(x1, y1, 'r--')                  # Plot line1 with red line curve

x2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]                    # x-attributes for line1
y2 = [0, 5, 12, 23, 34, 44, 56, 67, 88, 99]              # y-attribute for line 2
line1 = plt.plot(x2, y2, 'g-', linewidth=3.0)                    # Plot line1 with␣
 ↪red line curve

plt.axis([0, 10, 0, 100])                              # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.show()

# Alternatively .... with one plot()
'''
```

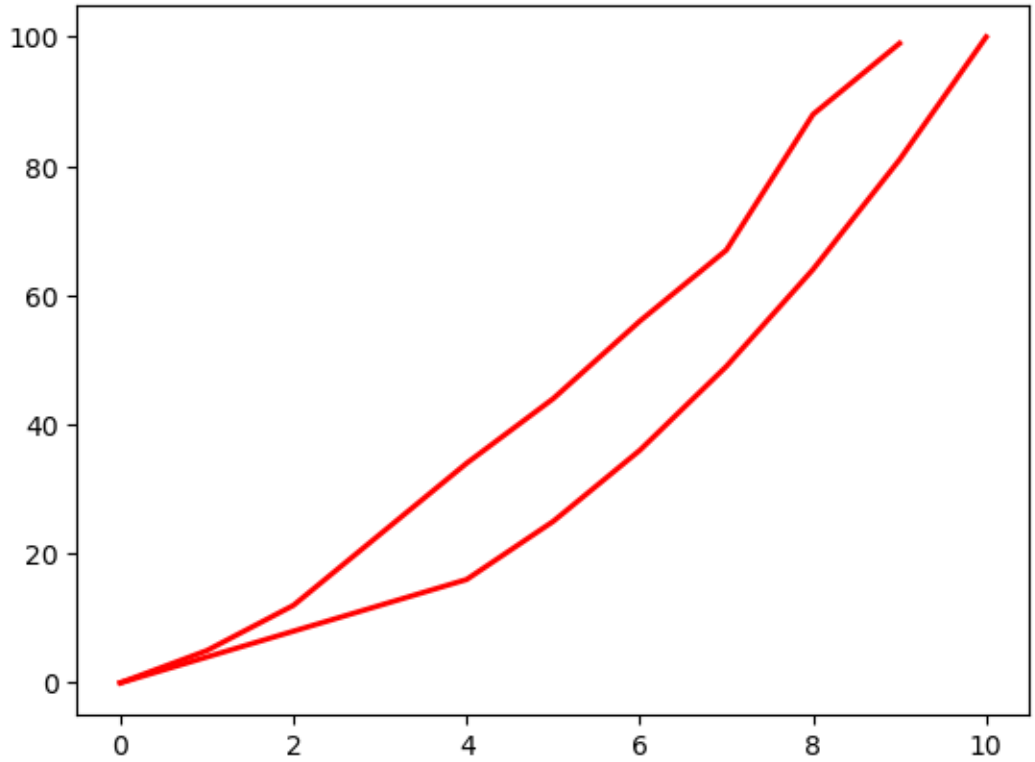
```
import matplotlib.pyplot as plt

x1 = [0, 4, 5, 6, 7, 8, 9, 10]                  # x-attributes for line1
y1 = [0, 16, 25, 36, 49, 64, 81, 100]            # y-attribute for line 2
# line1 = plt.plot(x1, y1, 'r--')                  # Plot line1 with red line␣
 ↪curve

x2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]                 # x-attributes for line1
y2 = [0, 5, 12, 23, 34, 44, 56, 67, 88, 99]           # y-attribute for line 2
# line1 = plt.plot(x2, y2, 'g-', linewidth=3.0)                  # Plot line1␣
 ↪with red line curve

plt.axis([0, 10, 0, 100])                           # The range of values for two␣
 ↪axes (x-min, x-max, y-min, y-max)
plt.plot(x1, y1, 'r--', x2, y2, 'g-', linewidth=2.0)
plt.show()
'''
```



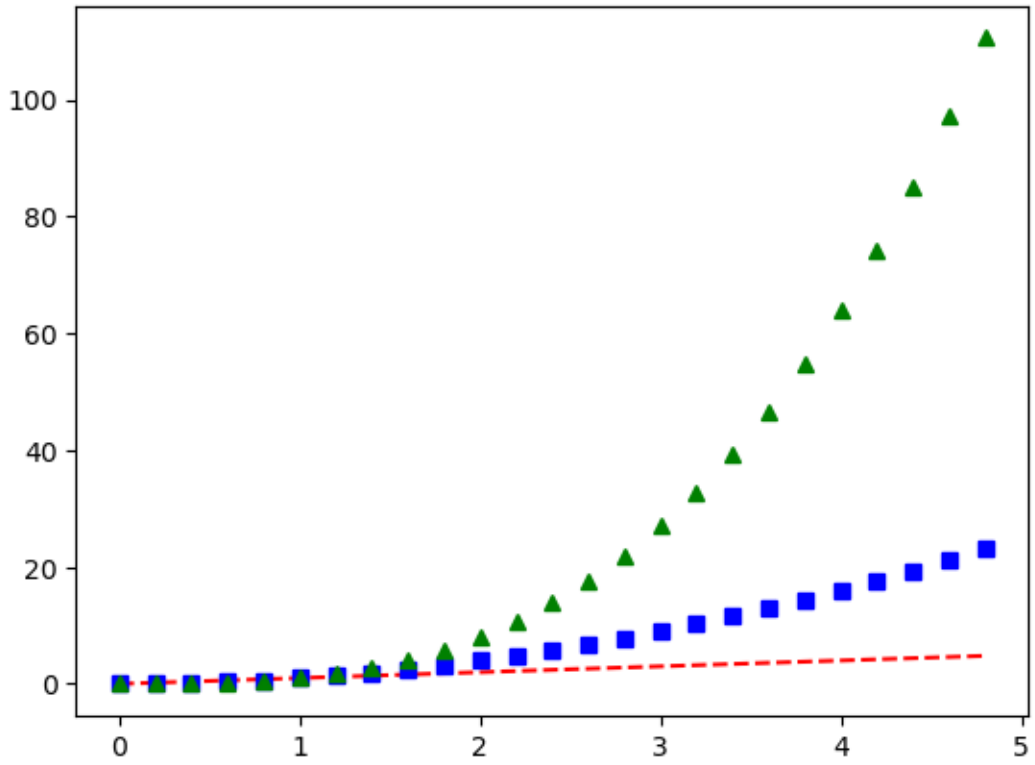
[52]:
```
x1 = [0, 4, 5, 6, 7, 8, 9, 10]                  # x-attributes for line1
y1 = [0, 16, 25, 36, 49, 64, 81, 100]           # y-attribute for line 2
```

```
x2 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]              # x-attributes for line1
y2 = [0, 5, 12, 23, 34, 44, 56, 67, 88, 99]      # y-attribute for line 2

lines = plt.plot(x1, y1, x2, y2)
# use keyword args
plt.setp(lines, color='r', linewidth=2.0)      #To set multiple properties on a␣
 ↪list of lines???
```

[52]: [None, None, None, None]



[49]:
```
# Illustrating a few more arguments for the versatile plot() method

import numpy as np
import matplotlib.pyplot as plt

t = np.arange(0., 5., 0.2)       # Evenly sampled time at 200ms intervals with␣
 ↪a range 0..5
# red dashes, blue squares and green triangles
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



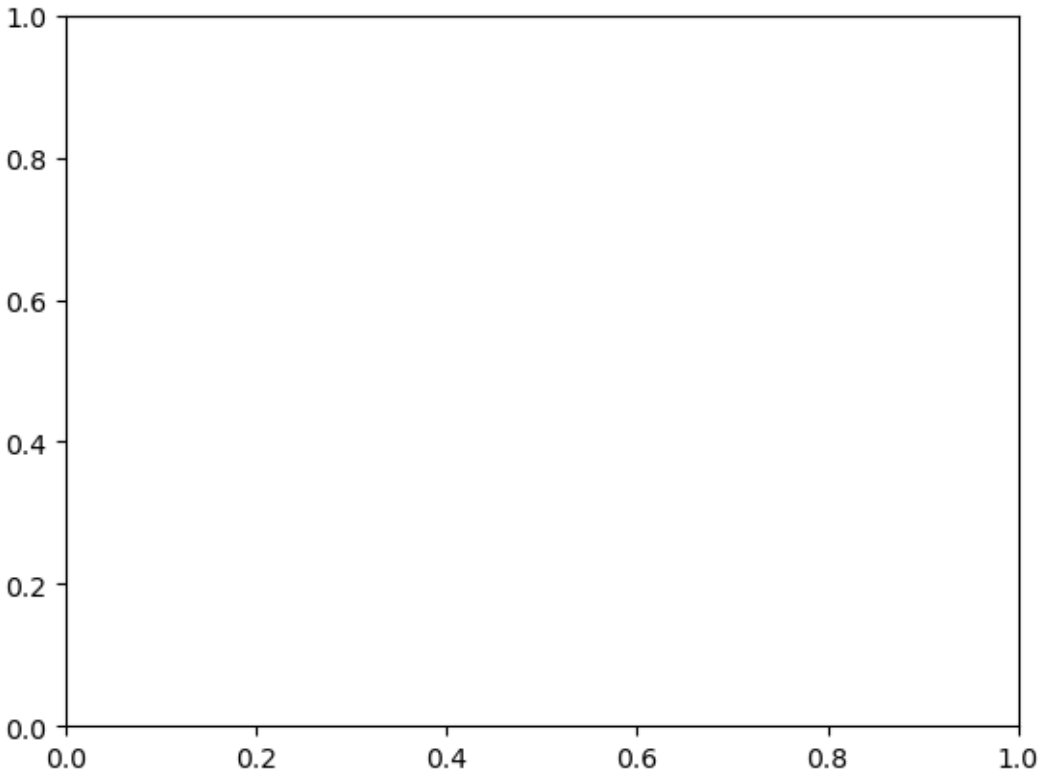
2 Drawing using figure() : Alternative to plot()

[55]:
```
#create figure using plt.figure() return figure object
fig = plt.figure()

# Viewing figure to display figure need to tell explicitly pyplot to display it
# below command will return the object to display figure it required at least␣
 ↪one axes.
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```

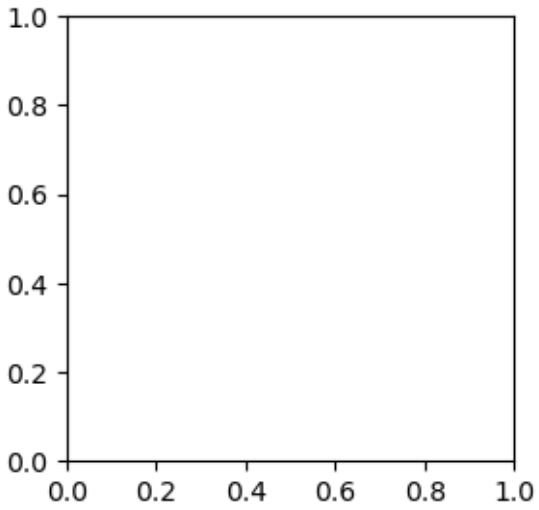
2.1 Adding axes

[58]:
```
#Axes can be added by using methods add_subplot(nrows, ncols, index) return␣
 ↪axes object
fig = plt.figure()
ax = fig.add_subplot()
plt.show()
```



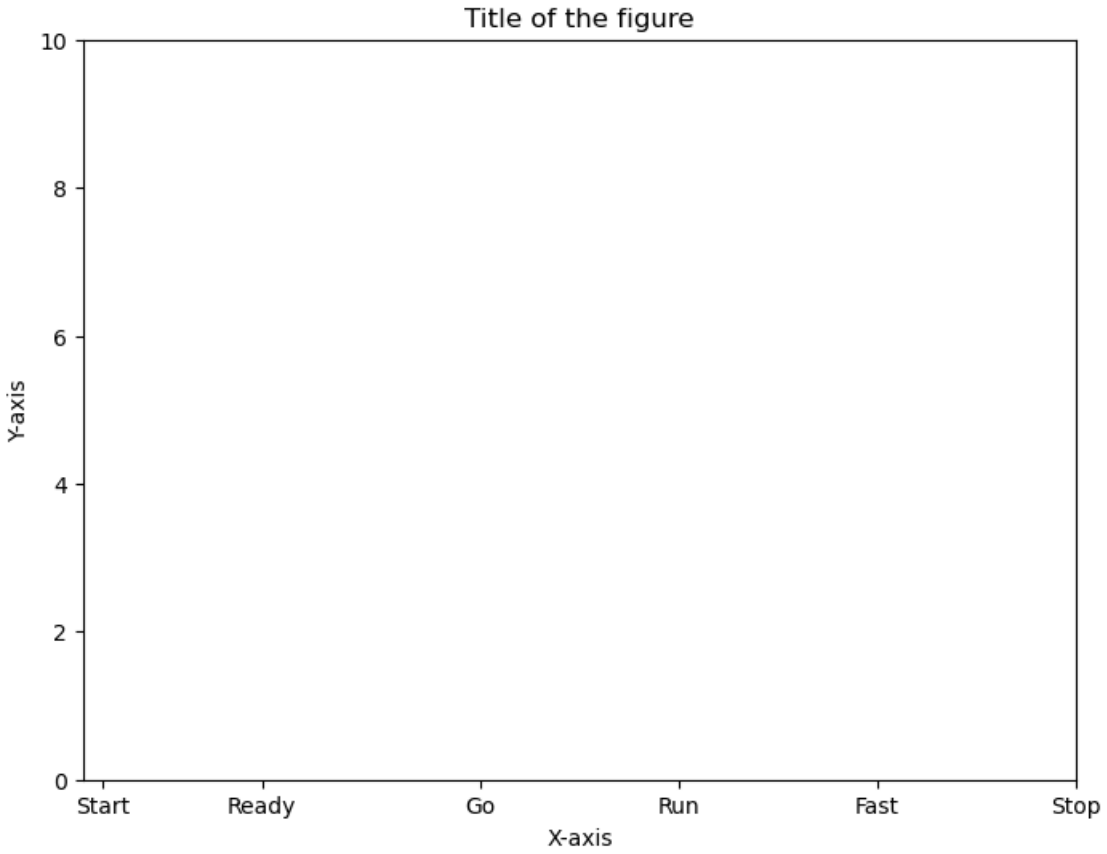
2.2 Adjusting the size of the figure

[61]:
```
#Adjusting Figure Size => plt.figure(figsize=(x, y))
fig = plt.figure(figsize=(3, 3))
ax = fig.add_subplot()
plt.show()
```



2.3 Setting properties for the axes

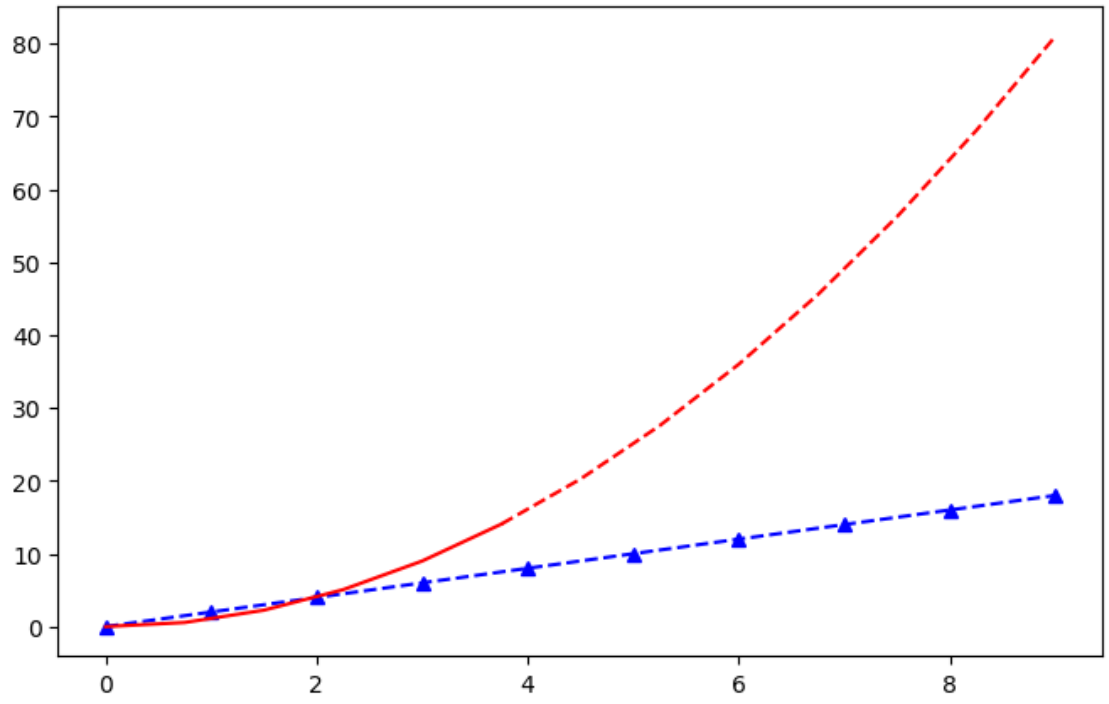
[64]:
```
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot()

# Set properties for the axes
ax.set(
    title="Title of the figure",              # Set the title of the plot
    xlabel='X-axis',                          # Label for the x-axis
    ylabel='Y-axis',                          # Label for the y-axis
    xlim=(0,5),                               # Set the limits for the x-axis␣
 ↪(0 to 5)
    ylim=(0,10),                              # Set the limits for the y-axis␣
 ↪(0 to 10)
    xticks=[0.1, 0.9, 2, 3, 4, 5],            # Specify tick positions on the␣
 ↪x-axis
    xticklabels=['Start', 'Ready', 'Go', 'Run', 'Fast', 'Stop']  # Custom␣
 ↪labels for x-axis ticks
)

plt.show()
```



2.4 A simple example....

[34]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

x1 = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]                          # Range of values␣
 ↪for x-values
y1 = [0,2,4,6,8, 10, 12, 14, 16, 18]                      # Range of values for f1:␣
 ↪y = f1(x)

# Resize your Graph (dpi specifies pixels per inch. When saving probably should␣
 ↪use 300 if possible)
plt.figure(figsize=(8,5), dpi=100)                          # Set the size and␣
 ↪quality of figure

# Line 1: Drawing the line1 on the figure for the function y = 2*x
plt.plot(x1,y1, 'b^--', label='y = f1(x) = 2*x')
```

```
# Line2: Drawing the line2 on the figure for the function y = x^2

# Select the interval we want to plot points at
x2 = np.arange(0, 9.5, 0.75)
#print(x2)
# Plot part of the graph as line
plt.plot(x2[:6], x2[:6]**2, 'r', label='y = f2(x) = X^2')

# Plot remainder of the graph as a dot
plt.plot(x2[5:], x2[5:]**2, 'r--')
plt.show()
```



3 Different types of graph plotting using Matplotlib Pyplot

3.0.1 Types of Plot

1. Line plot
2. Scatter plot
3. Bar plot
4. Pie plot
5. Histogram

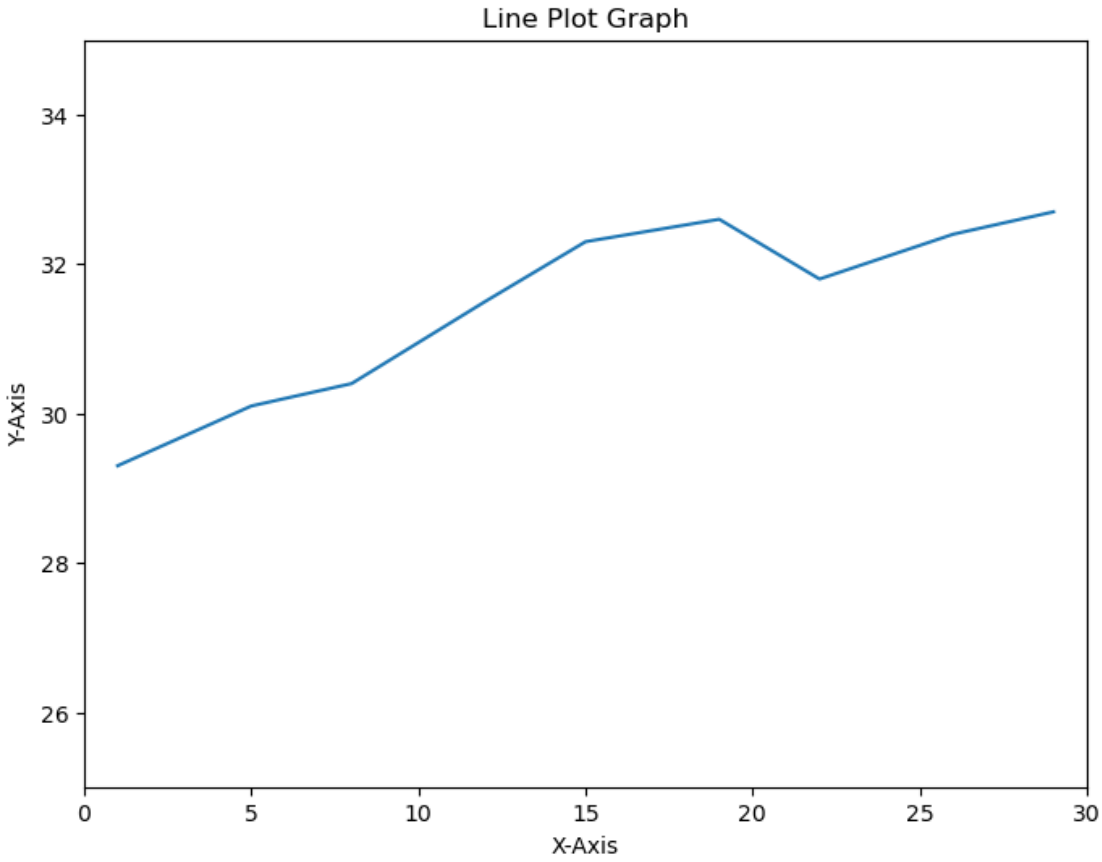
6. Box plot

3.0.2 1. Line Plot

- Line Plot is used to visualize a trend in data.
- Line Plot is also used to compare two variables.
- Line Plots are simple and effective in communicating.
- plot function is used for drawing Line plots.
- Syntax: plt.plot(x,y)

3.0.3 Example of lineplot

[43]:
```
x = [1, 5, 8, 12, 15, 19, 22, 26, 29]
y = [29.3, 30.1, 30.4, 31.5, 32.3, 32.6, 31.8, 32.4, 32.7]

fig = plt.figure(figsize=(8,6))        # Initialiaze figure object
ax = fig.add_subplot(1,1,1)            # Initialize axes object

ax.set(title='Line Plot Graph'         # Some basic setting of axes
       , xlabel='X-Axis'
       , ylabel='Y-Axis'
       , xlim=(0, 30)
       , ylim=(25, 35))

ax.plot(x, y)                  # Plotted with default parameters
plt.show()
```



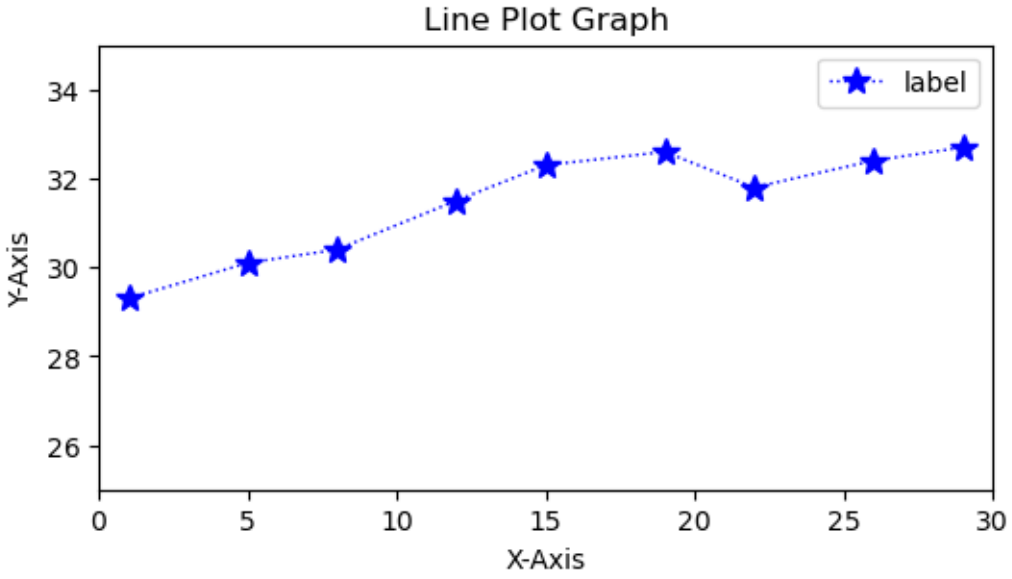
3.0.4 A few parameters for plot() method

- color - Sets the color of the line.
- linestyle - Sets the line style, e.g., solid, dashed, etc.
- linewidth - Sets the thickness of a line.
- marker - Chooses a marker for data points, e.g., circle, triangle, etc.
- markersize - Sets the size of the chosen marker.
- label - Names the line, which will come in legend.

3.0.5 Example of lineplot with a seeting in plot()

[52]:
```
x = [1, 5, 8, 12, 15, 19, 22, 26, 29]
y = [29.3, 30.1, 30.4, 31.5, 32.3, 32.6, 31.8, 32.4, 32.7]


fig = plt.figure(figsize=(6,3))
ax = fig.add_subplot(1,1,1)
```

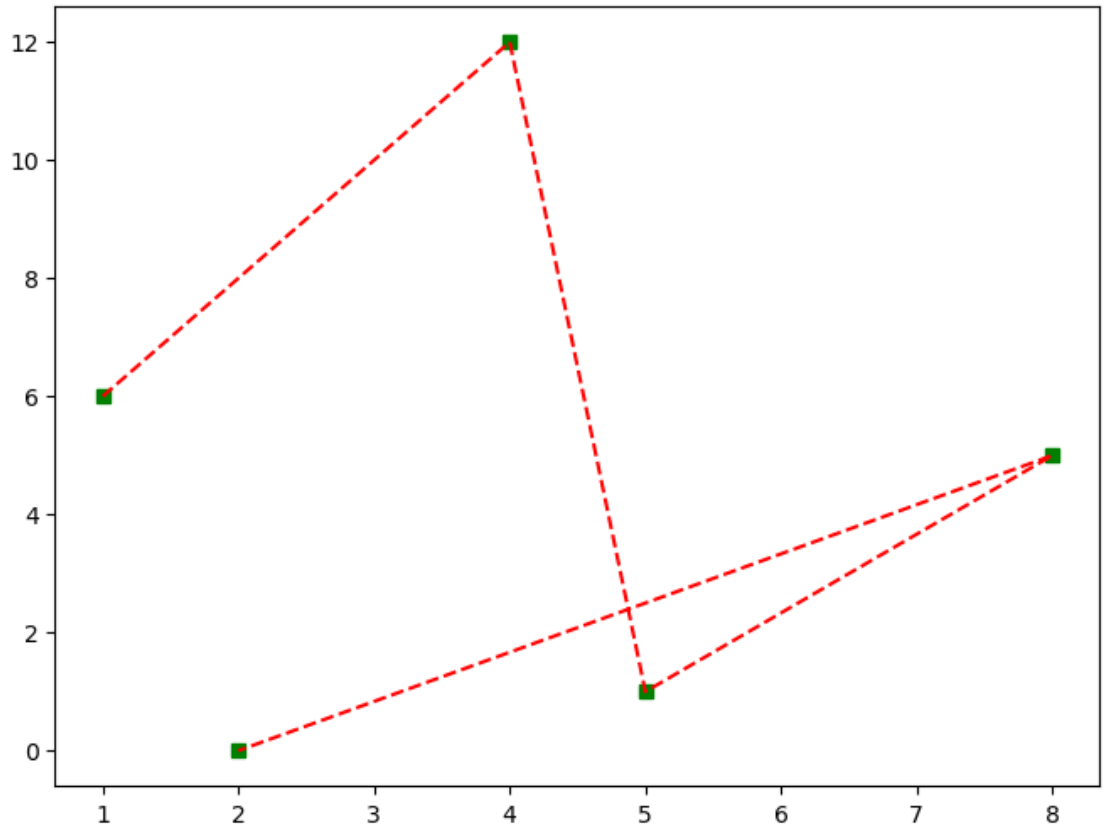
```
ax.set(title='Line Plot Graph'
       , xlabel='X-Axis'
       , ylabel='Y-Axis'
       , xlim=(0, 30)
       , ylim=(25, 35))

ax.plot(x
        , y
        , color='blue'
        , linestyle='dotted'
        , linewidth=1
        , marker='*'                # Other parameter value can be 'o', 's', '*',␣
 ↪'^', etc.
        , markersize=10
        , label='label')
plt.legend()                         # To show the legend
plt.show()
```



3.0.6 Example of muktple lineplot with single plot()

[56]:
```
x = [1, 4, 5, 8, 2]  # x-coordinates for the plot
y = [6, 12, 1, 5, 0]  # y-coordinates for the plot

fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot()
```

```
""" Plot the data with two styles:
'gs': green square markers for the data points
'r--': red dashed line connecting the data points """
ax.plot(x, y, 'gs', x, y, 'r--')

plt.show()
```



3.0.7 2. Scatter Graph

- It similar to line graph
- Used to show how one variable is related to another
- It consist of data point, if it in linear then it higly corelated
- It only mark the data point.
- Syntax: plt.scatter(x,y)

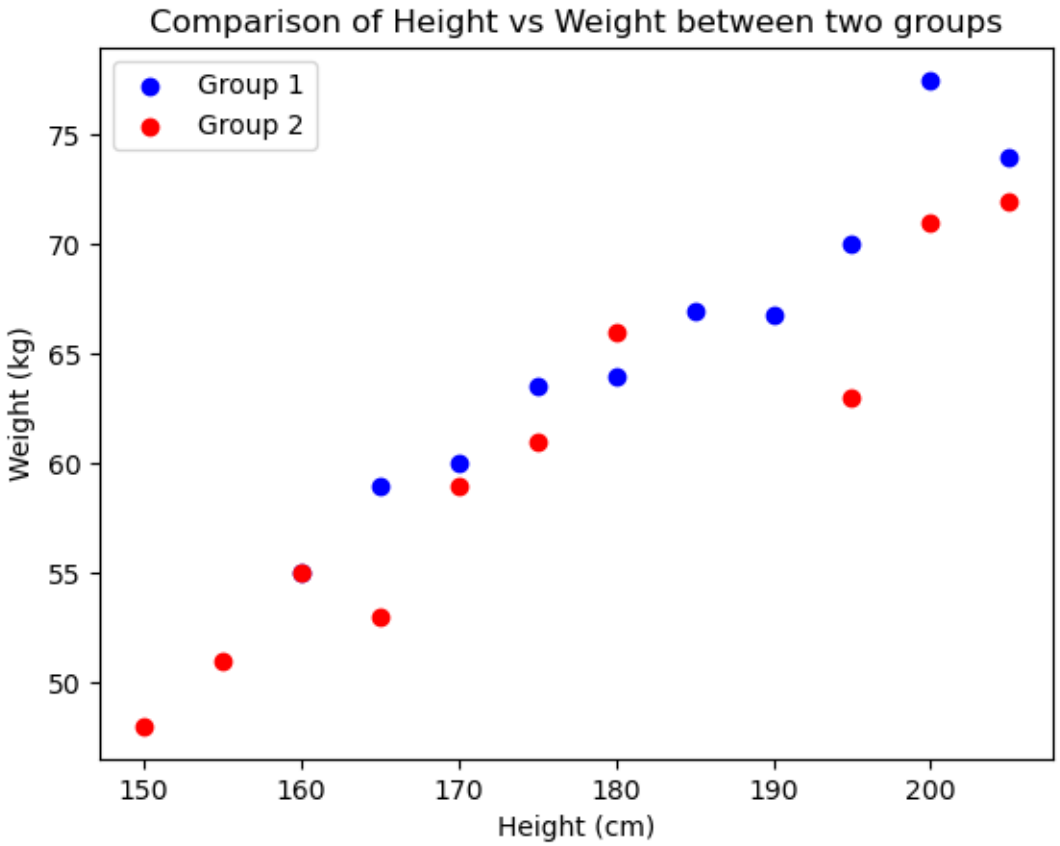
3.0.8 A few basic paramters of Scatter plot

- c: Sets color of markers.
- s: Sets size of markers.

- marker: Selects a marker. e.g: circle, triangle, etc
- edgecolor: Sets the color of lines on edges of markers.

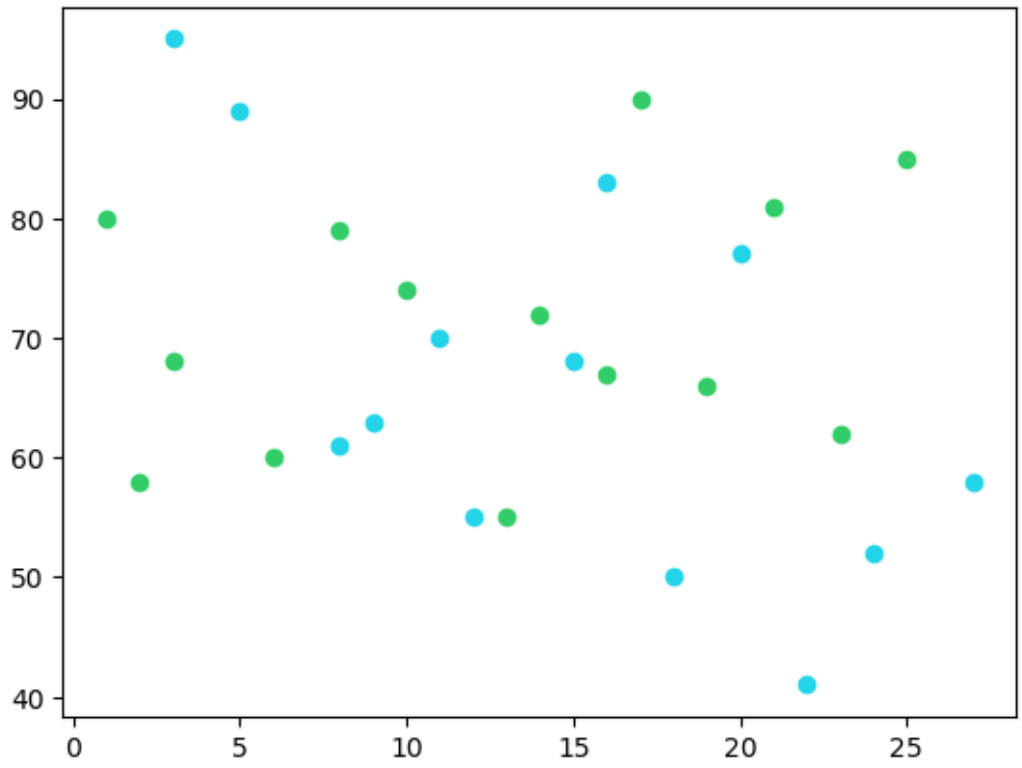
3.0.9 Example 1: A simple scatter plot

[61]:
```
x1 = np.array([160, 165, 170, 175, 180, 185, 190, 195, 200, 205])
y1 = np.array([55, 59, 60, 63.5, 64, 67, 66.8, 70, 77.5, 74])

x2 = np.array([150, 155, 160, 165, 170, 175, 180, 195, 200, 205])
y2 = np.array([48, 51, 55, 53, 59, 61, 66, 63, 71, 72])

plt.scatter(x1, y1, color='blue', label='Group 1')
plt.scatter(x2, y2, color='red', label='Group 2')

plt.xlabel('Height (cm)')
plt.ylabel('Weight (kg)')
plt.title('Comparison of Height vs Weight between two groups')

plt.legend()
plt.show()
```



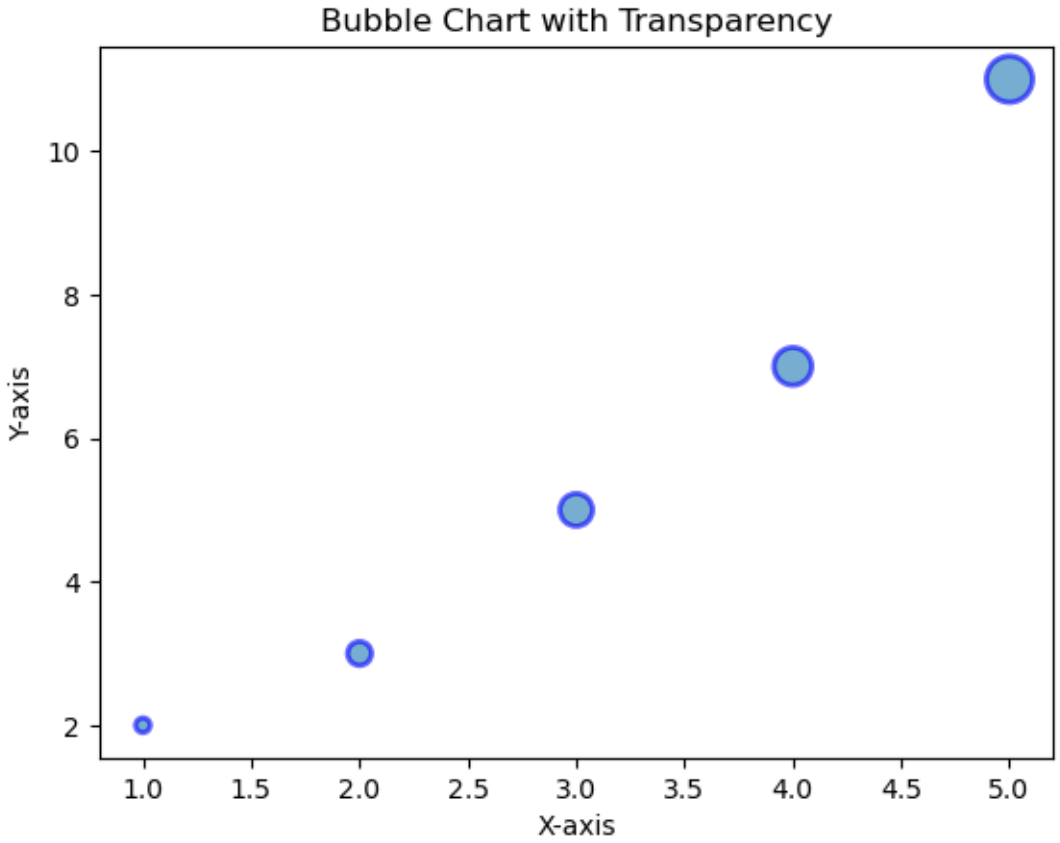
3.0.10 Example 2: Scatter plot with the customization of color and size

[17]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.array([3, 12, 9, 20, 5, 18, 22, 11, 27, 16, 8, 24, 15])
y = np.array([95, 55, 63, 77, 89, 50, 41, 70, 58, 83, 61, 52, 68])
plt.scatter(x, y, color='#23d4e8')

x = np.array([1, 6, 14, 17, 3, 13, 10, 8, 19, 21, 2, 16, 23, 25])
y = np.array([80, 60, 72, 90, 68, 55, 74, 79, 66, 81, 58, 67, 62, 85])
plt.scatter(x, y, color='#32cd68')

plt.show()
```



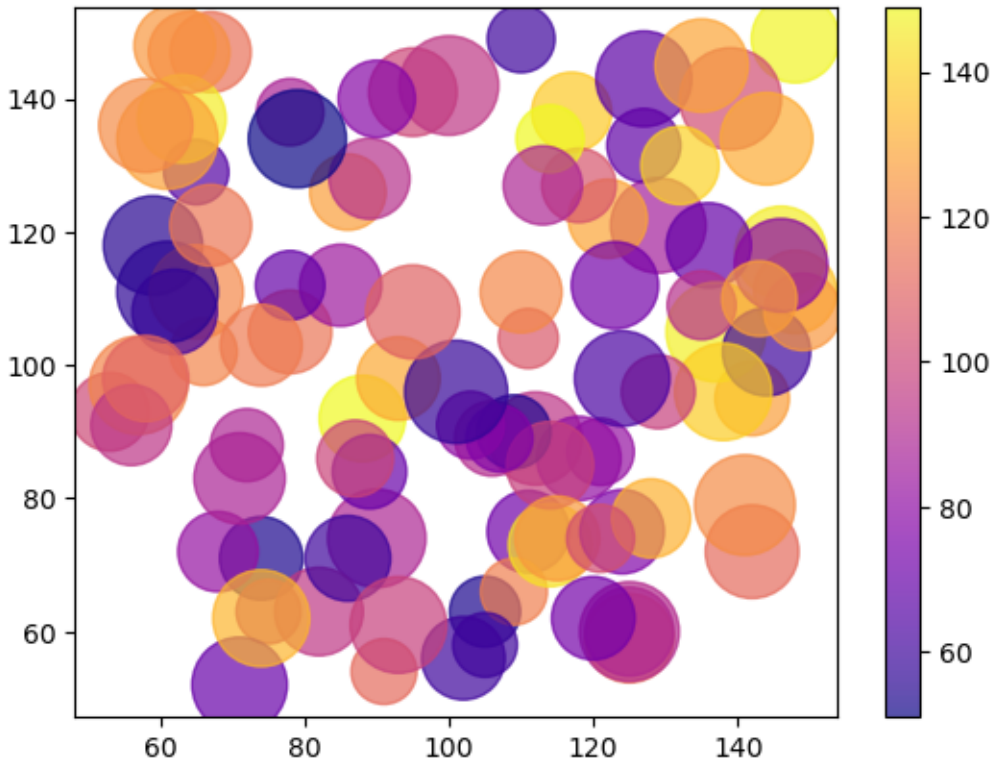
3.0.11 Example 3: Scatter plot with Bubble points

[20]:
```
import matplotlib.pyplot as plt

# Data
x_values = [1, 2, 3, 4, 5]
y_values = [2, 3, 5, 7, 11]
bubble_sizes = [30, 80, 150, 200, 300]

# Create a bubble chart with customization
plt.scatter(x_values, y_values, s=bubble_sizes, alpha=0.6, edgecolors='b',␣
 ↪linewidths=2)

# Add title and axis labels
plt.title("Bubble Chart with Transparency")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")

# Display the plot
plt.show()
```



3.0.12 Example 4: Customization of color plots with colr, size, and transparency

[23]:
```
import matplotlib.pyplot as plt
import numpy as np

x = np.random.randint(50, 150, size=(100))
y = np.random.randint(50, 150, size=(100))
colors = np.random.randint(50, 150, size=(100))
sizes = 10 * np.random.randint(50, 150, size=(100))

plt.scatter(x, y, c=colors, s=sizes, alpha=0.7, cmap='plasma')

plt.colorbar()
plt.show()
```



3.0.13 3. Bar Chart

- It mostly used to compare categories
- bar is used for vertical bar plots
- barh is used for horizontal bar plots
- Syntax: bar(x, height) or barh(y,width)

bar(x,width)

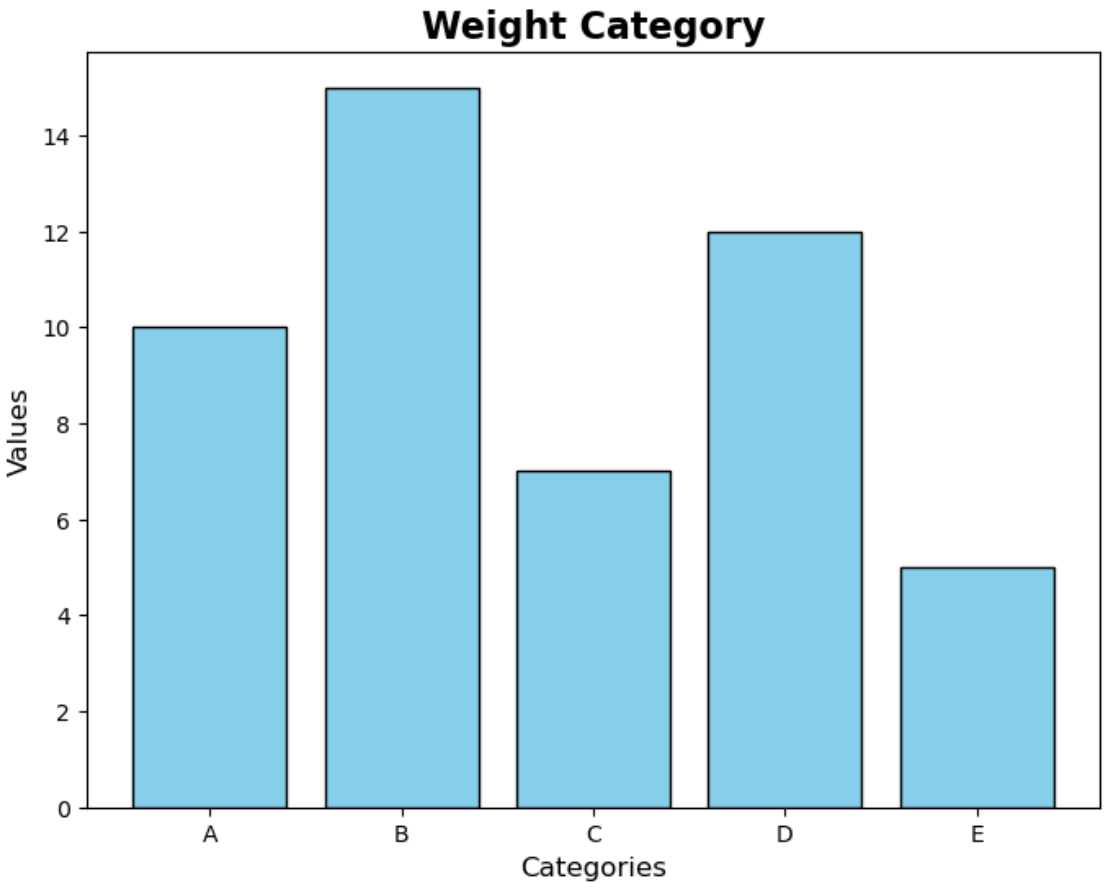
- width: Sets the width of bars

barh(y,height)

- height: Sets the height of bars

3.0.14 Example of bar chart

[70]:
```
categories = ['A', 'B', 'C', 'D', 'E']          # Weigth Categories on the x-axis
values = [10, 15, 7, 12, 5]                     # Number of students in each␣
 ↪weight category

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

#fig, ax = plt.subplots(figsize=(8, 6))

# Create a bar chart
ax.bar(categories, values, color='skyblue', edgecolor='black')

# Set the title and axis labels
ax.set_title('Weight Category', fontsize=16, fontweight='bold')
ax.set_xlabel('Categories', fontsize=12)
ax.set_ylabel('Values', fontsize=12)

plt.show()
```



3.0.15 4. Pie Plot

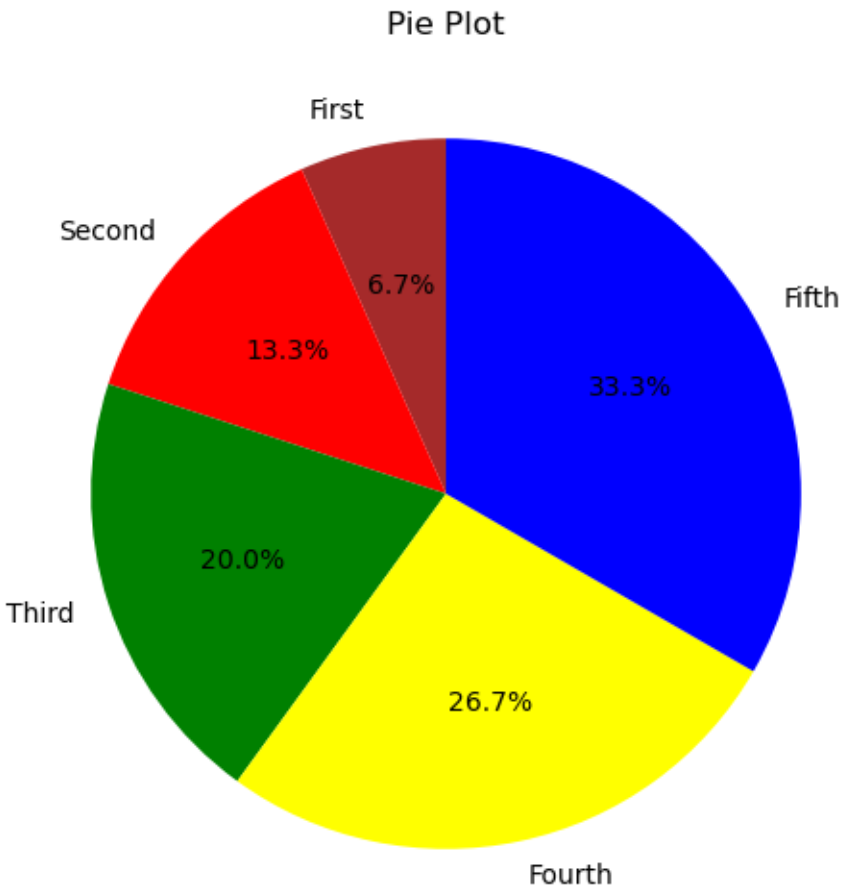
- It is effective in showing the proportion of categories.
- It is best suited for comparing fewer categories.
- It is used to highlight proportion of one or a group of categories.
- Syntax: pie(x), x: size of portions, passed as fraction or number

3.0.16 Parameters of Pie plot

- colors: Sets the colors of portions.
- labels: Sets the labels of portions.
- startangle: Sets the start angle at which portion drawing starts.
- autopct: Sets the percentage display format of an area, covering portions.

3.0.17 Example of pie plot

[76]:
```
x=[1,2,3,4,5]

fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot()
ax.set(title='Pie Plot')

# create pie plot
plt.pie(x
       , colors=['brown', 'red', 'green', 'yellow', 'blue'] # colors for each␣
 ↪slice
       , labels=['First', 'Second', 'Third', 'Fourth', 'Fifth'] # labels for␣
 ↪each slice
       , startangle=90  # Start angle for the first slice
       , autopct='%1.1f%%')  # Show percentage on each slice


plt.show()
```



3.0.18 5. Histogram Chart

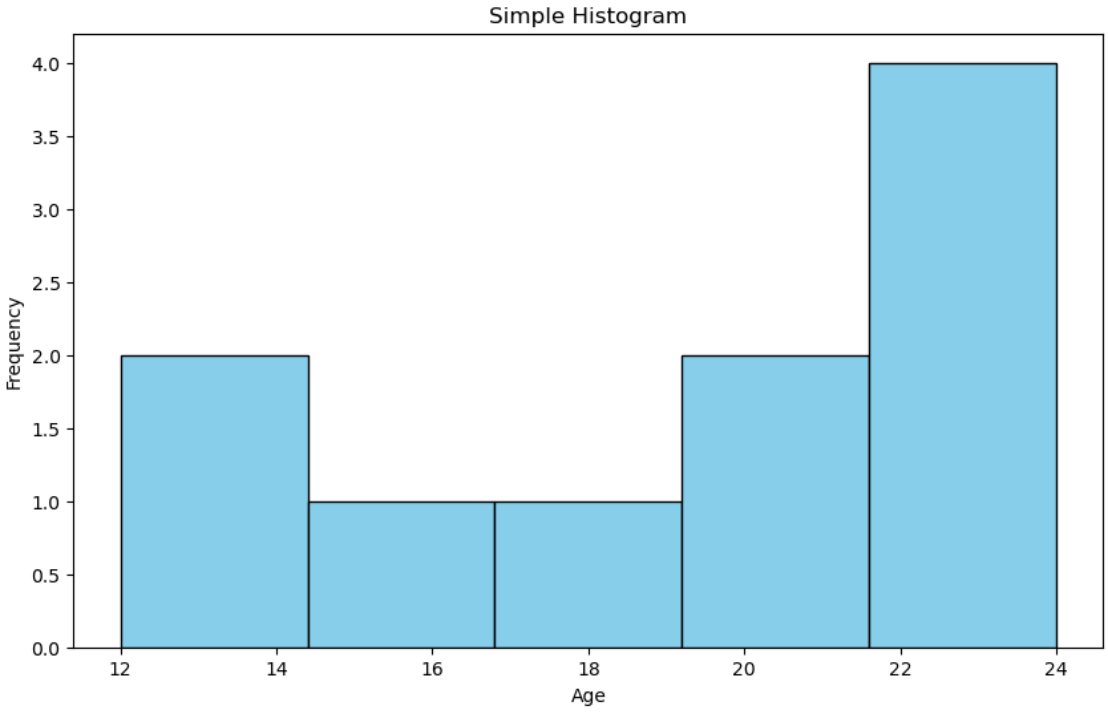
- It used to visualize the spread of data of a distribution
- Syntax: hist(x), x is the data values

3.0.19 Parameter of Histogram

- color: Sets the color of bars.
- bins: Sets the number of bins to be used.
- density: Sets to True where bins display fraction and not the count.

3.0.20 Example of histogram plot

[82]:
```
# Example data (e.g., ages of 10 people)
x = [12, 15, 12, 18, 20, 20, 22, 22, 22, 24]

fig = plt.figure(figsize=(10, 6))
ax = fig.add_subplot()

ax.set(title='Simple Histogram',
       xlabel='Age',
       ylabel='Frequency')

# Create a basic histogram
ax.hist(x,                # Data values
        color='skyblue',   # Color of the bars
        bins=5,            # Number of bins (divisions)
        edgecolor='black') # Color of the edge of bars

plt.show()
```



3.0.21 6. Box plot

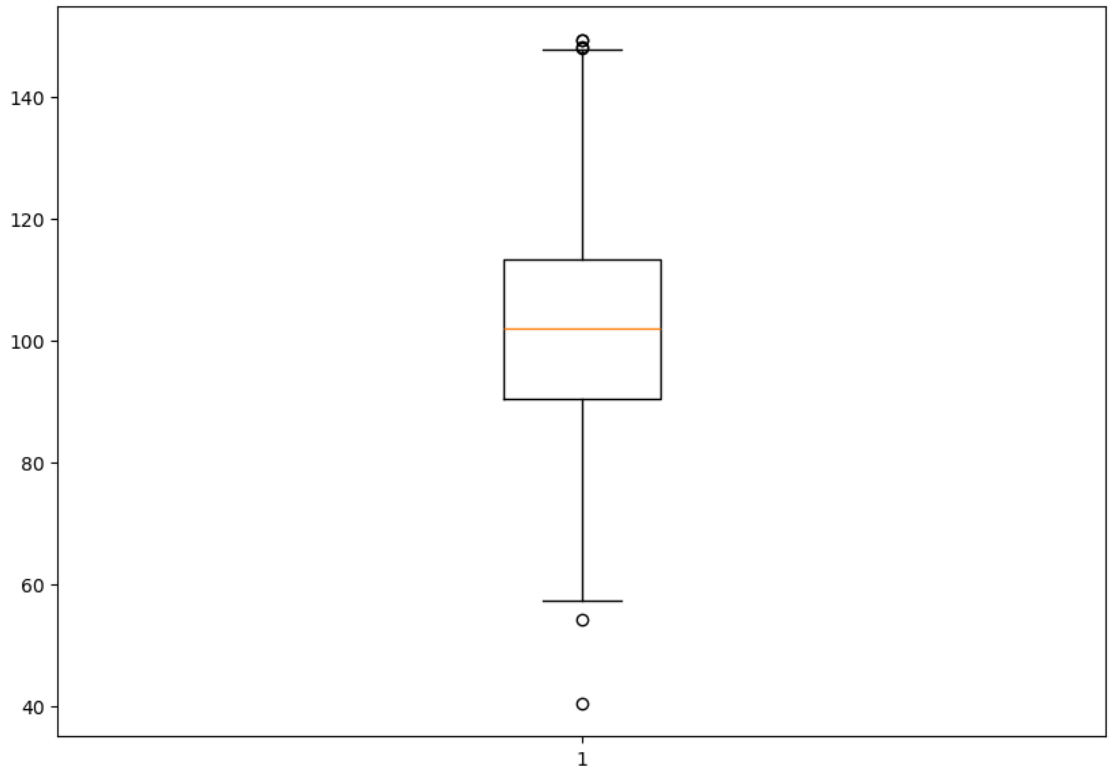
A Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile (Q1 : 25 percentile), median (Q2 : 50 percentile), third quartile(Q3 : 75 percentile) and maximum.

In the box plot, a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here, x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.

3.0.22 Example: A single box plot

[15]:
```
# Creating dataset
np.random.seed(10)          #Initilize the random number generator with a seed␣
 ↪value

#Generate a set of random numbers following a normal distribution with center␣
 ↪(100), std (20), and size (200)
data = np.random.normal(100, 20, 200)
#print(data)
```

```
fig = plt.figure(figsize =(10, 7))

# Creating plot
plt.boxplot(data)

# show plot
plt.show()
```

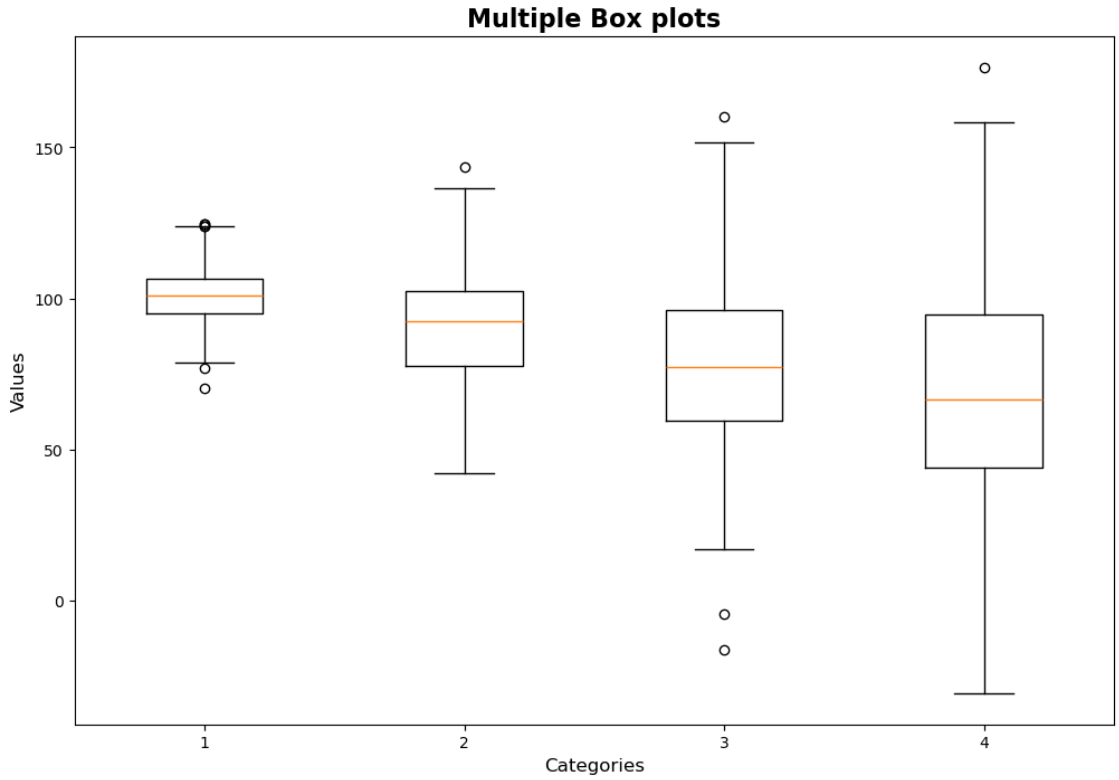


3.0.23 One or more box plots can be drawn to show the variances of different groups of data

3.0.24 Example: Multiple box plots on the same figure

[7]:
```
import numpy as np
import matplotlib.pyplot as plt

# Creating dataset
np.random.seed(10)

group1 = np.random.normal(100, 10, 200)
group2 = np.random.normal(90, 20, 200)
```

```
group3 = np.random.normal(80, 30, 200)
group4 = np.random.normal(70, 40, 200)

data = [group1, group2, group3, group4]

fig = plt.figure(figsize = (12, 8))

# Creating axes instance
ax = fig.add_subplot()
# Set the title and axis labels
ax.set_title('Multiple Box plots', fontsize=16, fontweight='bold')
ax.set_xlabel('Categories', fontsize=12)
ax.set_ylabel('Values', fontsize=12)

ax.set(xticks=[1, 2, 3, 4],            # Specify tick positions on the x-axis
       xticklabels=['Group 1', 'Group 2', 'Group 3', 'Group 4'])  # Custom␣
 ↪labels for x-axis ticks

# Creating plot
bp = ax.boxplot(data)

# show plot
plt.show()
```



3.1 Subplots: Figures with grids

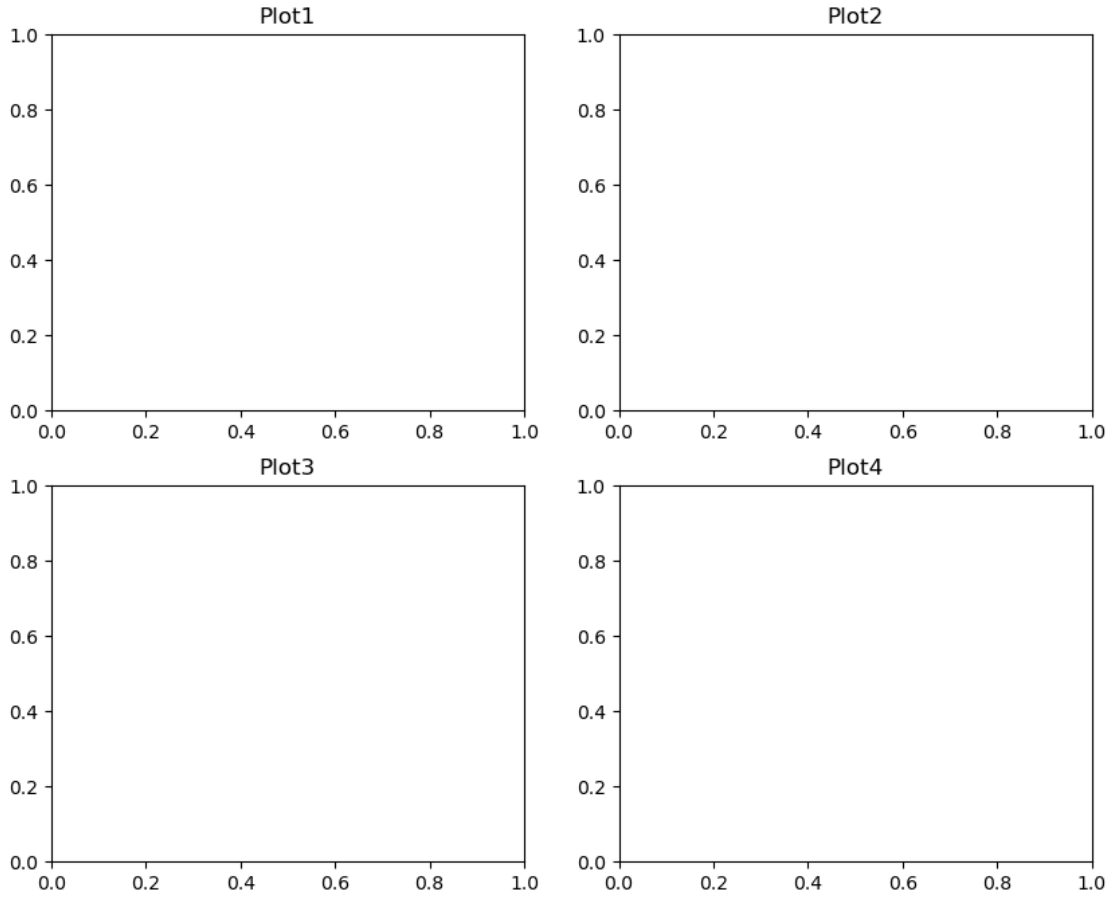
3.1.1 It provides a way to plot multiple plots on a single figure. Given the number of rows and columns, it returns a tuple (fig, ax), giving a single figure fig with an array of axes ax.

3.1.2 subplot creates the Axes object at index position and returns it.

- 'index' is the position in a virtual grid with 'nrows' and 'ncols'
- 'index' number varies from 1 to `nrows*ncols`.
- Syntax: subplot(nrows, ncols, index)

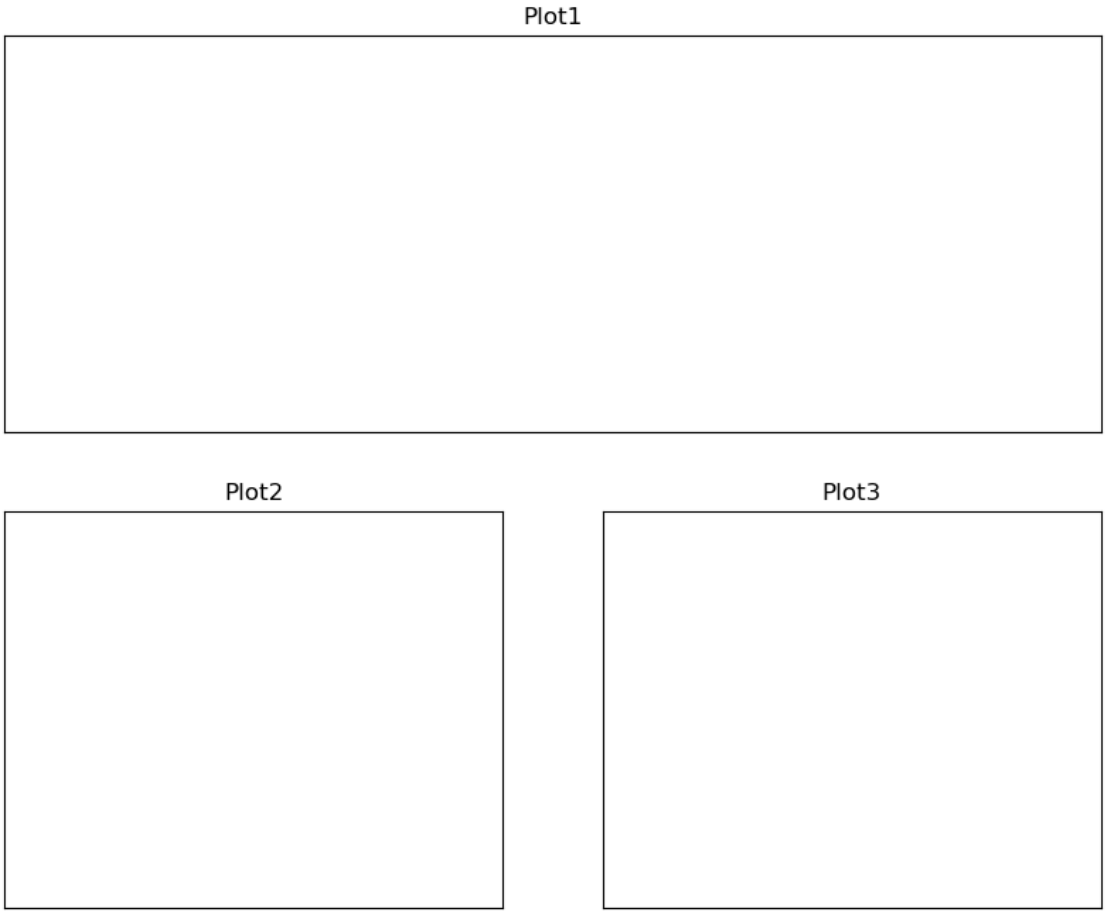
3.1.3 Example 1: Show a grid of four subplots

[51]:
```
fig = plt.figure(figsize=(10,8))
ax1 = plt.subplot(2, 2, 1, title='Plot1')      #subplot(nrows, ncolumn, index),␣
 ↪index 1, 2, 3, ...
ax2 = plt.subplot(2, 2, 2, title='Plot2')
ax3 = plt.subplot(2, 2, 3, title='Plot3')
ax4 = plt.subplot(2, 2, 4, title='Plot4')
plt.show()
```



3.1.4 Example 2: Customisation in grid layout

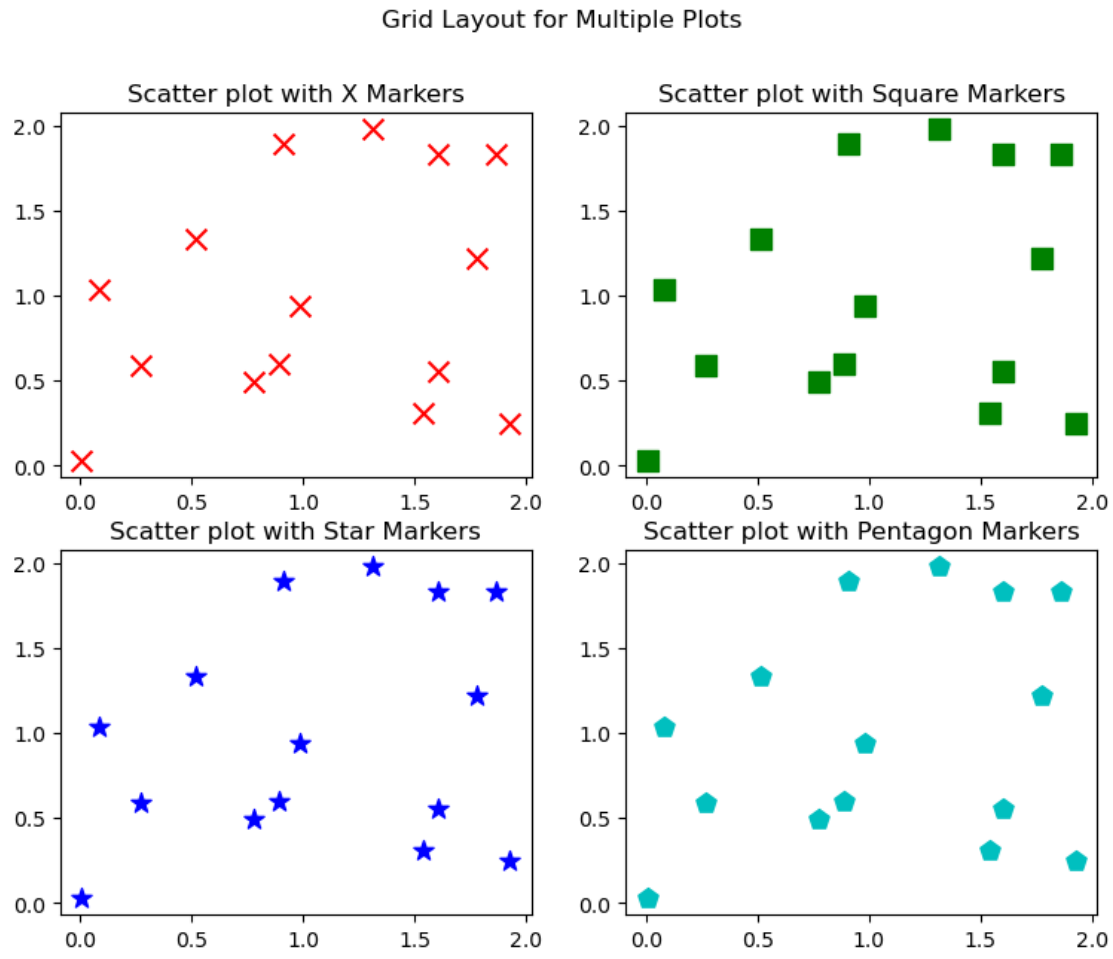
[10]:
```
fig = plt.figure(figsize=(10,8))

ax1 = plt.subplot(2, 2, (1,2), title='Plot1')
ax1.set_xticks([])
ax1.set_yticks([])

ax2 = plt.subplot(2, 2, 3, title='Plot2')
ax2.set_xticks([])
ax2.set_yticks([])

ax3 = plt.subplot(2, 2, 4, title='Plot3')
ax3.set_xticks([])
ax3.set_yticks([])
plt.show()
```



3.1.5 Example 3: Ploting graphs on each axes

[13]:
```
# Generate new random data
np.random.seed(2025)
x = np.random.rand(15) * 2  # Values between 0 and 2
y = np.random.rand(15) * 2  # Values between 0 and 2
z = np.sqrt(x**2 + y**2)    # Calculate the distance from origin

fig = plt.figure(figsize=(9, 7))
fig.suptitle('Grid Layout for Multiple Plots')

# Subplot 1: Scatter plot with "X" Markers
ax1 = plt.subplot(2, 2, 1, title='Scatter plot with X Markers')
ax1.scatter(x, y, s=100, c='r', marker='x')  # "X" markers
ax1.set(xticks=(0.0, 0.5, 1.0, 1.5, 2.0), yticks=(0.0, 0.5, 1.0, 1.5, 2.0))

# Subplot 2: Scatter plot with "Square" Markers
```

```
ax2 = plt.subplot(2, 2, 2, title='Scatter plot with Square Markers')
ax2.scatter(x, y, s=100, c='g', marker='s')  # "Square" markers
ax2.set(xticks=(0.0, 0.5, 1.0, 1.5, 2.0), yticks=(0.0, 0.5, 1.0, 1.5, 2.0))

# Subplot 3: Scatter plot with "Star" Markers
ax3 = plt.subplot(2, 2, 3, title='Scatter plot with Star Markers')
ax3.scatter(x, y, s=100, c='b', marker='*')  # "Star" markers
ax3.set(xticks=(0.0, 0.5, 1.0, 1.5, 2.0), yticks=(0.0, 0.5, 1.0, 1.5, 2.0))

# Subplot 4: Scatter plot with "Pentagon" Markers
ax4 = plt.subplot(2, 2, 4, title='Scatter plot with Pentagon Markers')
ax4.scatter(x, y, s=100, c='c', marker='p')  # "Pentagon" markers
ax4.set(xticks=(0.0, 0.5, 1.0, 1.5, 2.0), yticks=(0.0, 0.5, 1.0, 1.5, 2.0))

plt.show()
```



3.2 Finally you should save your work in a drive.

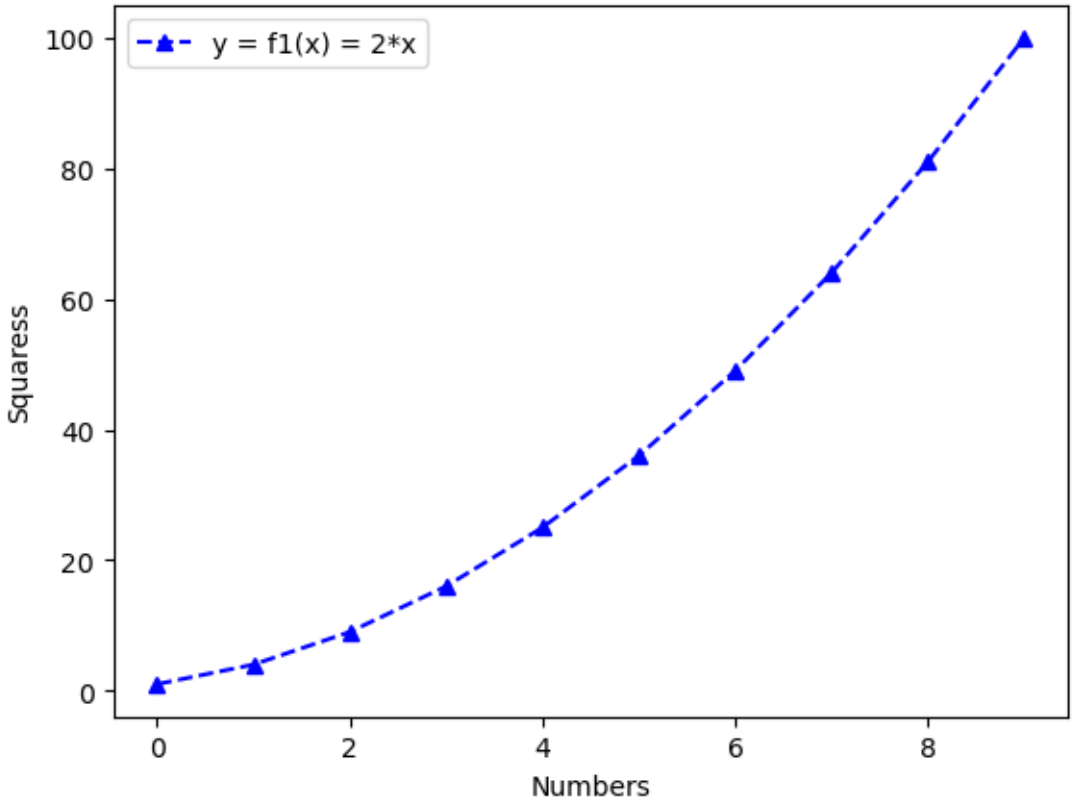
3.2.1 To save a Matplotlib figure, you can use the savefig() method:

For example:

plt.savefig('my_figure.png'):

This line saves the current figure to a file named my_figure.png in the current working directory.

[22]:
```
import matplotlib.pyplot as plt

# Create a figure and plot something
y = [1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
plt.plot(y, 'b^--', label='y = f1(x) = 2*x')
plt.xlabel('Numbers')
plt.ylabel('Squaress')

# Save the figure
plt.legend()
plt.savefig('my_figure.png')
```

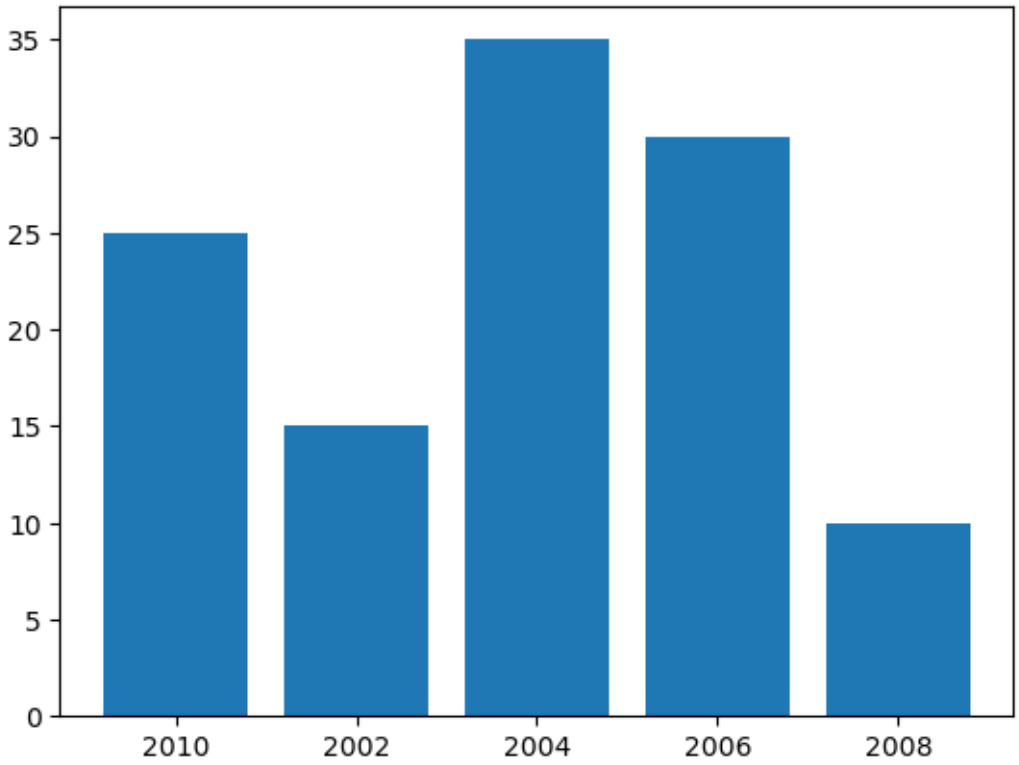


3.2.2 Important arguments:

- You can use the following arguments depending in our requirements:
  fname: The filename (including the path if needed) where you want to save the figure.
  format: The format of the saved figure (e.g., 'png', 'jpg', 'pdf', 'svg'). If not specified, it is inferred from the filename extension.
  dpi: The resolution of the saved figure (in dots per inch, for example, 200).
  bbox_inches: Use bbox_inches='tight' to remove extra whitespace around the figure.
  transparent: Set to True to make the figure background transparent.

3.2.3 Another example:

[26]:
```
# Creating data
year = ['2010', '2002', '2004', '2006', '2008']
production = [25, 15, 35, 30, 10]

# Plotting barchart
plt.bar(year, production)

# Saving the figure.
plt.savefig("output.jpg")

# Saving figure by changing parameter values
plt.savefig("output1", facecolor='y', bbox_inches="tight",
            pad_inches=0.3, transparent=True)
```


35
30
25
20
15
10
5
0
2010
2002
2004
2006
2008