Plotting and Visualization using Matplotlib

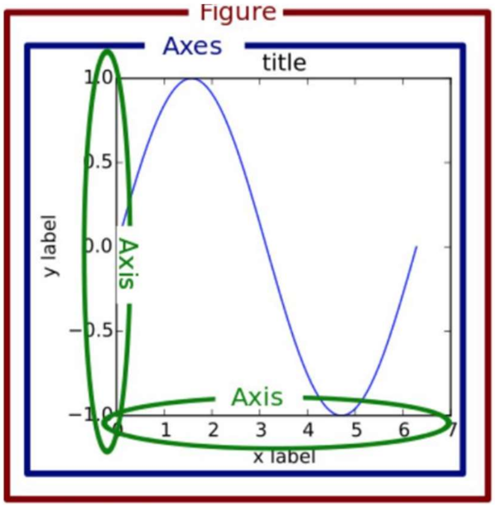
- Matplotlib is a Python library for publication-quality 2D and 3D graphics, with support for a variety of different output formats.
- It is built on NumPy arrays and designed to work with the broader SciPy stack and consists of several plots like line, bar, scatter, histogram, etc.

Install Matplotlib on Windows

- Users who prefer to use pip can use the below command to install Matplotlib:

```
pip install matplotlib
```

Matplotlib - Object Hierarchy



- Figure: Outermost container for a Matplotlib graphic. Can contain multiple Axes objects.
- Axes: Actual plots. Contain smaller objects (tick marks, individual lines, etc.)
- Artist: Everything that is seen on the figure is an artist.

Creating Different Types of Plot

- In data visualization, creating various types of plots is essential for effectively conveying insights from data. Below, we'll explore how to create different types of plots using Matplotlib, a powerful plotting library in Python.

Different Types of Charts

Matplotlib supports a variety of charts, each suitable for different data visualization needs:

- **Line Charts**: Ideal for showing trends over time. A line chart can be created using the `plot()` function.
- **Bar Charts**: Useful for comparing quantities corresponding to different groups. Bar charts can be vertical or horizontal and are generated using the `bar()` or `barh()` functions.
- **Histograms**: Great for showing the frequency distribution of a dataset. Histograms are created using the `hist()` function.
- **Box Plots**: Used to show the distribution of a dataset. It can be created with the `boxplot()` function and is useful for detecting outliers.
- **Scatter Plots**: Perfect for showing the relationship between two variables. Created using the `scatter()` function.
- **Pie Charts**: Best for showing the proportional makeup of a dataset. Pie charts are generated using the `pie()` function.
- **Area Charts**: Similar to line charts but with the area below the line filled in. Created using the `stackplot()` function.
- **Density Plots**: Useful for visualizing the distribution of a dataset. Density plots can be created using the `kde()` function.
- **Hexbin Plots**: Ideal for representing the intensity of data points in two dimensions. Created using the `hexbin()` function.

Markers

character	description
'.'	point marker
','	pixel marker
'o'	circle marker
'v'	triangle_down marker
'^'	triangle_up marker
'<'	triangle_left marker
'>'	triangle_right marker
'1'	tri_down marker
'2'	tri_up marker
'3'	tri_left marker
'4'	tri_right marker
'8'	octagon marker
's'	square marker
'p'	pentagon marker
'P'	plus (filled) marker
'*'	star marker
'h'	hexagon1 marker
'H'	hexagon2 marker
'+'	plus marker
'x'	x marker
'X'	x (filled) marker
'D'	diamond marker
'd'	thin_diamond marker
'|'	vline marker
'_'	hline marker

Line Styles

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

Colors

character	color
'b'	**blue**
'g'	**green**
'r'	**red**
'c'	**cyan**
'm'	**magenta**
'y'	**yellow**
'k'	**black**
'w'	white

Subplot Placement

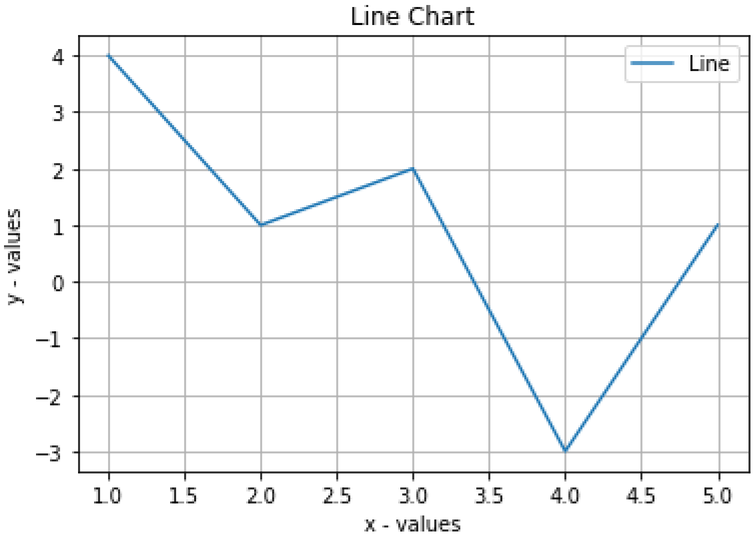
221	222
223	224

321	322	323
324	325	326

A Basic Line Plot

In [1]:

```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4, 5]
y = [4, 1, 2, -3, 1]

# This will plot a simple line chart
# with elements of x as x axis and y
# as y axis
plt.plot(x, y)
plt.grid()
plt.title("Line Chart")
plt.xlabel('x - values')
plt.ylabel('y - values')

# Adding the legends
plt.legend(["Line"])
plt.show()
```

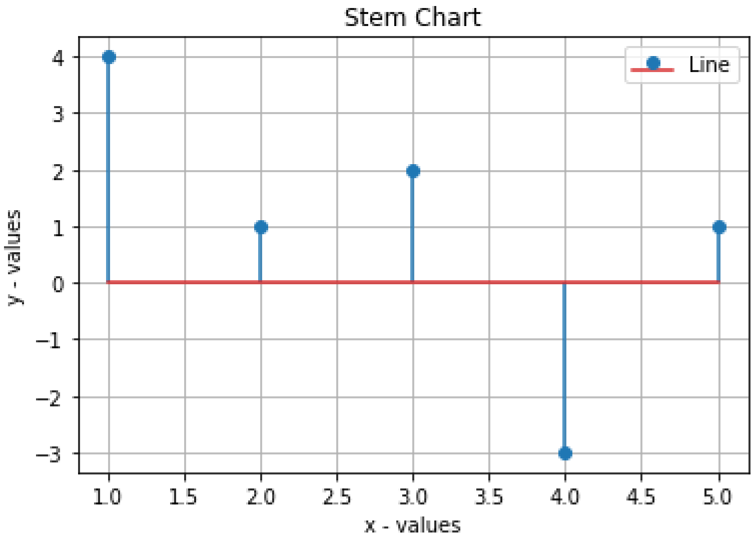


A Stem Plot

In [2]:

```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4, 5]
y = [4, 1, 2, -3, 1]

# This will plot a simple line chart
# with elements of x as x axis and y
# as y axis
plt.stem(x, y)
plt.grid()
plt.title("Stem Chart")
plt.xlabel('x - values')
plt.ylabel('y - values')
```

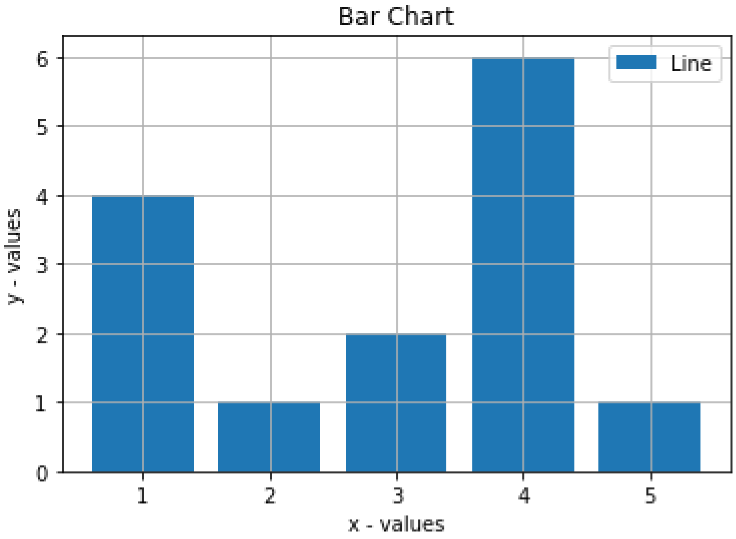
```
# Adding the legends
plt.legend(["Line"])
plt.show()
```



A Bar Plot

In [3]:

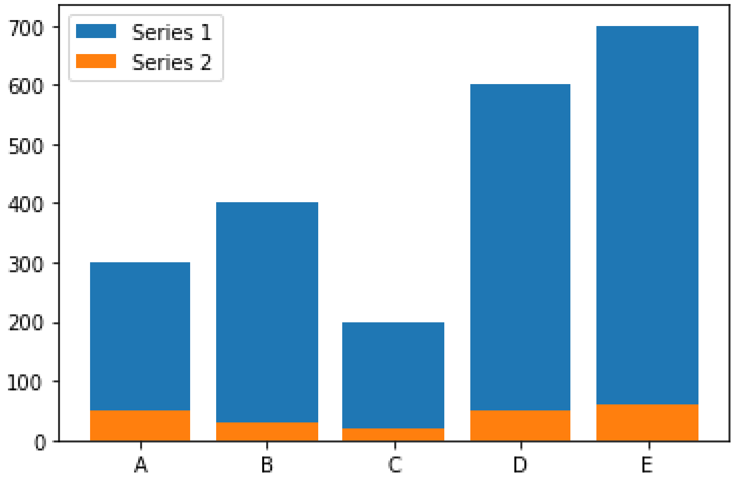
```
import matplotlib.pyplot as plt
# data to display on plots
x = [1, 2, 3, 4, 5]
y = [4, 1, 2, 6, 1]

# This will plot a simple line chart
# with elements of x as x axis and y
# as y axis
plt.bar(x, y)
plt.grid()
plt.title("Bar Chart")
plt.xlabel('x - values')
plt.ylabel('y - values')

# Adding the legends
plt.legend(["Line"])
plt.show()
```



Multiple Data on Bar Plot (The incorrect method)

In [4]:

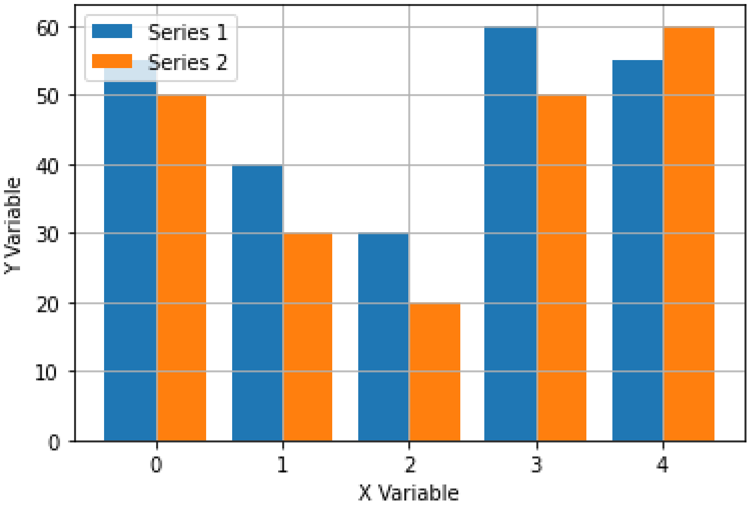
```
import matplotlib.pyplot as plt
import numpy as np

x = ['A','B','C','D','E']
y1 = [300,400,200,600,700]
y2 = [50,30,20,50,60]

plt.bar(x,y1)  # Plot some data on the axes.
plt.bar(x,y2)  # Plot some data on the axes.
plt.legend(['Series 1', 'Series 2'])
plt.show()
```



Multiple Data on Bar Plot (The correct method)

In [5]:

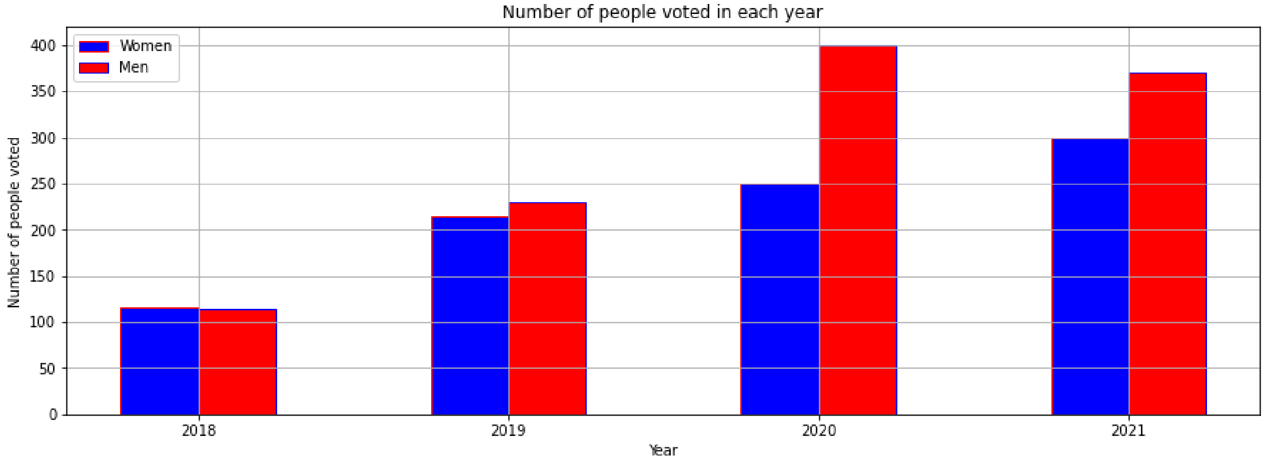
```
import matplotlib.pyplot as plt
import numpy as np


x = ['A','B','C','D','E']
y1 = [55,40,30,60,55]
y2 = [50,30,20,50,60]

x_axis = np.arange(len(x))

# plt.bar(x_axis-0.2,y1,0.4)
# plt.bar(x_axis+0.2,y2,0.4)

bar_width = 0.4
plt.bar(x_axis-bar_width/2,y1,bar_width)  # Plot some data on the axes.
plt.bar(x_axis+bar_width/2,y2,bar_width)  # Plot some data on the axes.


plt.legend(['Series 1', 'Series 2'])
plt.grid()

plt.xlabel('X Variable')
plt.ylabel('Y Variable')
plt.show()
```



In [6]:

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

Women = [115, 215, 250, 300]
Men = [114, 230, 400, 370]

x_axis = np.arange(len(Men))
width = 0.25

plt.figure(figsize=(15, 5))

plt.bar(x_axis, Women, color = 'b', width = width,
        edgecolor = 'red', label='Women')
plt.bar(x_axis + width, Men, color = 'r', width = width,
        edgecolor = 'blue', label='Men')
```

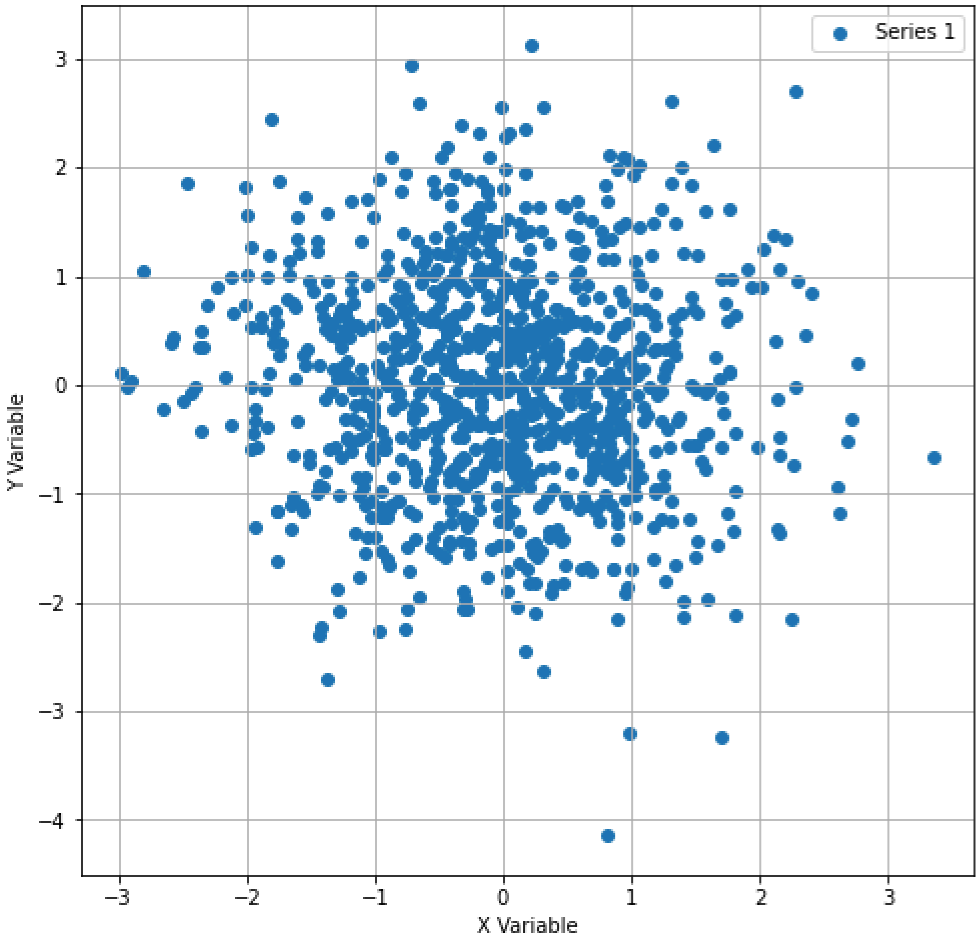
```python
plt.xlabel("Year")
plt.ylabel("Number of people voted")
plt.title("Number of people voted in each year")

# plt.grid(linestyle='--')
plt.xticks(x_axis + width/2,['2018','2019','2020','2021'])
plt.legend()
plt.grid()

plt.show()
```



Scatter Plot

In [7]:

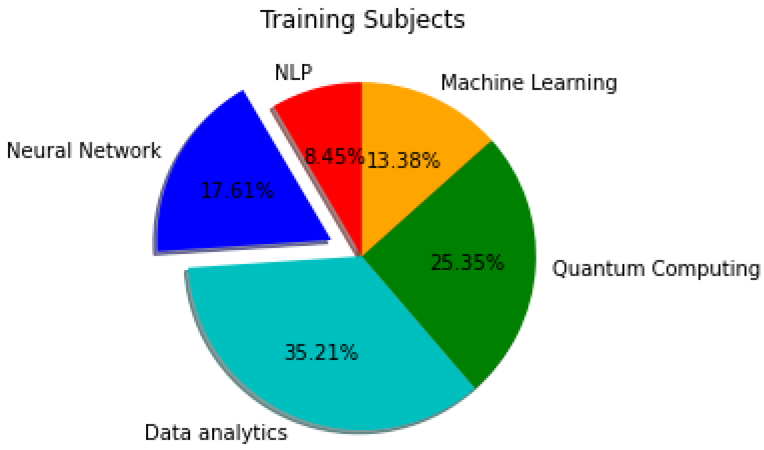
```python
import numpy as np
import matplotlib.pyplot as plt

x = np.random.randn(1, 1000)
y = np.random.randn(1, 1000)

plt.figure(figsize=(8, 8))
plt.scatter(x,y)  # Plot some data on the axes.
plt.legend(['Series 1', 'Series 2'])

plt.xlabel('X Variable')
plt.ylabel('Y Variable')
plt.grid()
plt.show()
```



A Pie Chart

In [8]:

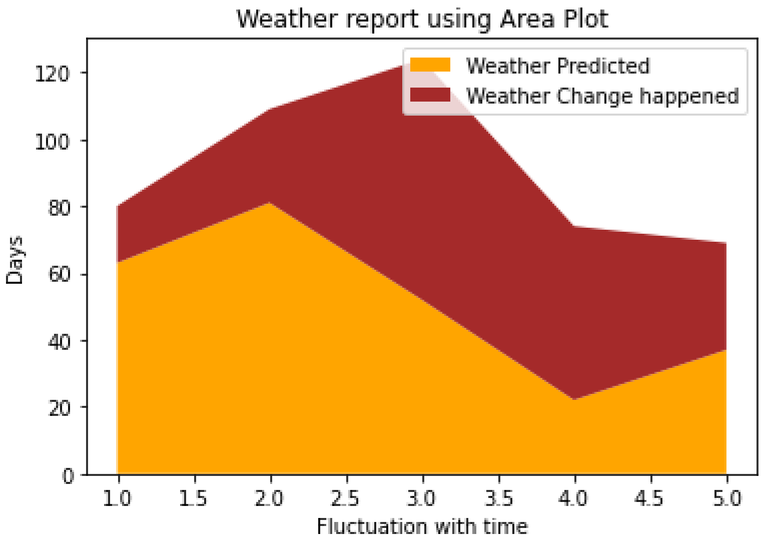
```
import matplotlib.pyplot as plt

data = [12, 25, 50, 36, 19]
activities = ['NLP','Neural Network', 'Data analytics', 'Quantum Computing', 'Machine L
cols = ['r','b','c','g', 'orange']
# plt.pie(data)
plt.pie(data,labels = activities, colors = cols, startangle = 90,
        shadow = True, explode =(0,0.2,0,0,0), autopct ='%1.2f%%')
plt.title('Training Subjects')

# Print the chart
plt.show()
```



Area Plot

In [9]:

```
import matplotlib.pyplot as plt

days = [1,2,3,4,5]
age = [63, 81, 52, 22, 37]
weight = [17, 28, 72, 52, 32]

plt.stackplot(days, age, weight, colors = ['orange', 'brown'])
plt.xlabel('Fluctuation with time')
plt.ylabel('Days')
plt.title('Weather report using Area Plot')
plt.legend(['Weather Predicted', 'Weather Change happened'])

# Print the chart
plt.show()
```

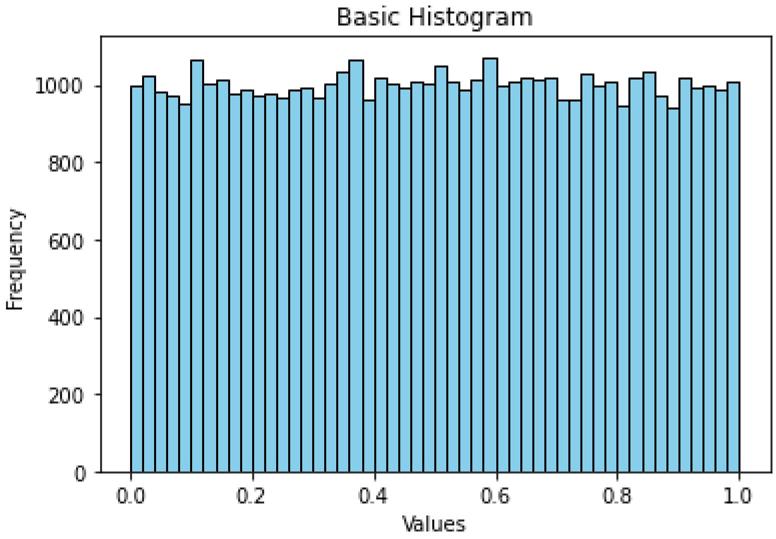


Histogram Plot

In [10]:

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

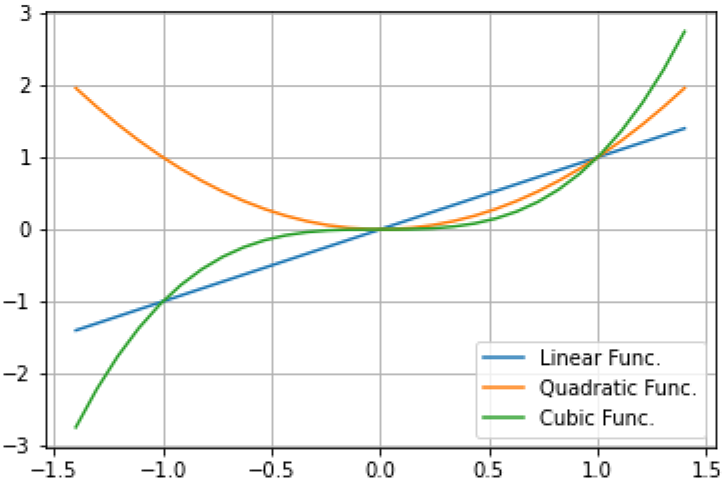
```
# Generate random data for the histogram
data = np.random.rand(50000)

# Plotting a basic histogram
plt.hist(data, bins=50, color='skyblue', edgecolor='black')

# Adding labels and title
plt.xlabel('Values')
plt.ylabel('Frequency')
plt.title('Basic Histogram')

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



Line Styles, Colours

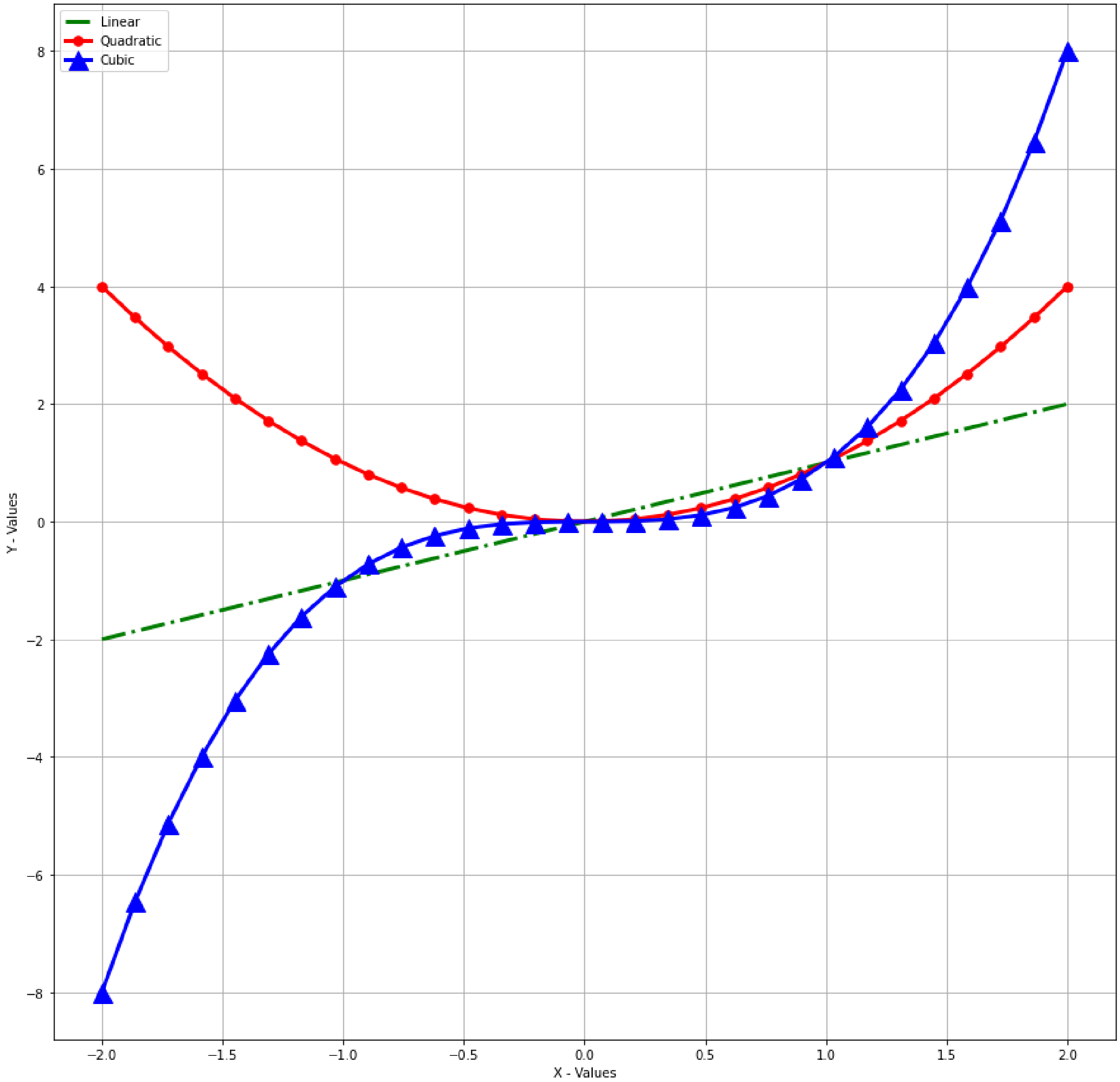
In [19]:

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

x = np.linspace(-1.4, 1.4, 30)
plt.plot(x, x)
plt.plot(x, x**2)
plt.plot(x, x**3)
plt.legend(['Linear Func.', 'Quadratic Func.', 'Cubic Func.'])
plt.grid()
plt.show()
```



In [12]:

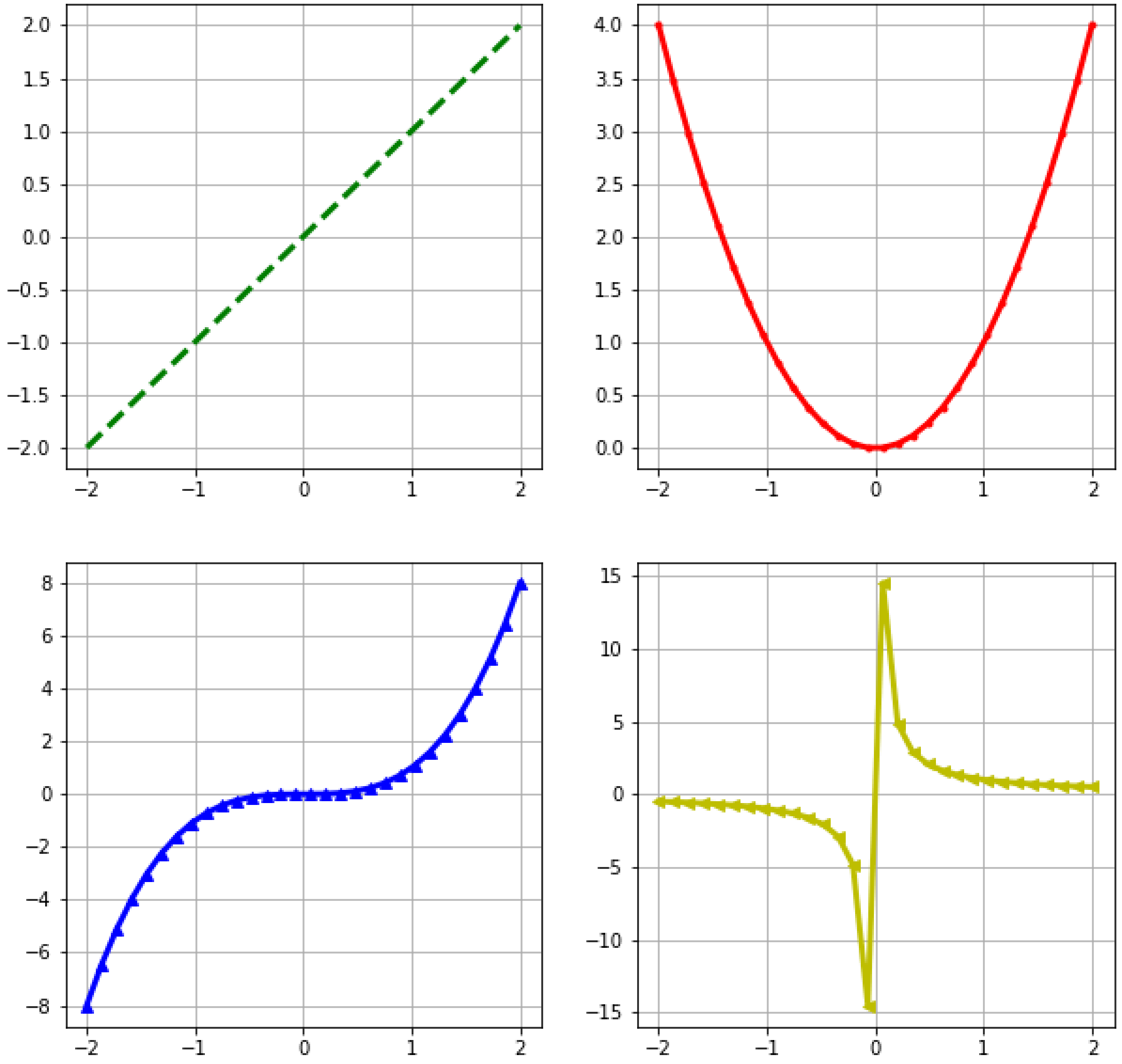
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-2, 2, 30)
plt.figure(figsize=(15, 15))
plt.plot(x, x, 'g-.', label = 'Linear', linewidth = 3, markersize=15)
plt.plot(x, x**2, 'r.-', label = 'Quadratic', linewidth = 3, markersize=15)
plt.plot(x, x**3, 'b^-', label = 'Cubic', linewidth = 3, markersize=15)
plt.xlabel('X - Values')
plt.ylabel('Y - Values')
plt.grid()
plt.legend()
plt.show()
```



Subplots

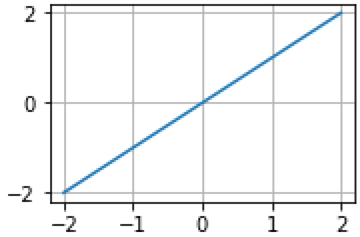
In [13]:

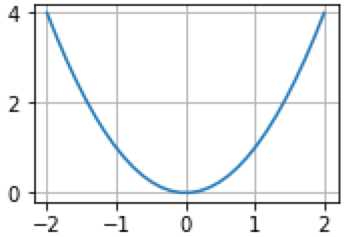
```
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(-2, 2, 30)

plt.figure(figsize=(10, 10))
plt.subplot(221)
plt.plot(x, x, 'g--', label = 'Linear', linewidth = 3)
plt.grid()

plt.subplot(2, 2, 2)
plt.plot(x, x**2, 'r.-', label = 'Quadratic', linewidth = 3)
plt.grid()

plt.subplot(2, 2, 3)
plt.plot(x, x**3, 'b^-', label = 'Cubic', linewidth = 3)
plt.grid()
plt.subplot(2, 2, 4)
plt.plot(x, x**-1, 'y<-', label = 'Reciprocal', linewidth = 3)
plt.grid()


plt.show()
```

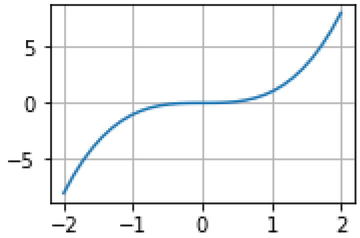


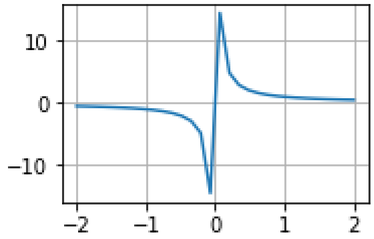
In [14]:

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


def plotCustom(x,y, plotNum, style, plotLabel):
    plt.subplot(2, 2, plotNum)
    plt.plot(x,y, 1, style, label = plotLabel)
    plt.grid()
    plt.show()

x = np.linspace(-2, 2, 30)

plotCustom(x, x, 1, 'g--', 'Linear')
plotCustom(x, x**2, 2, 'g--', 'Quadratic')
plotCustom(x, x**3, 3, 'g--', 'Cubic')
plotCustom(x, x**-1, 4, 'g--', 'Inverse')
```





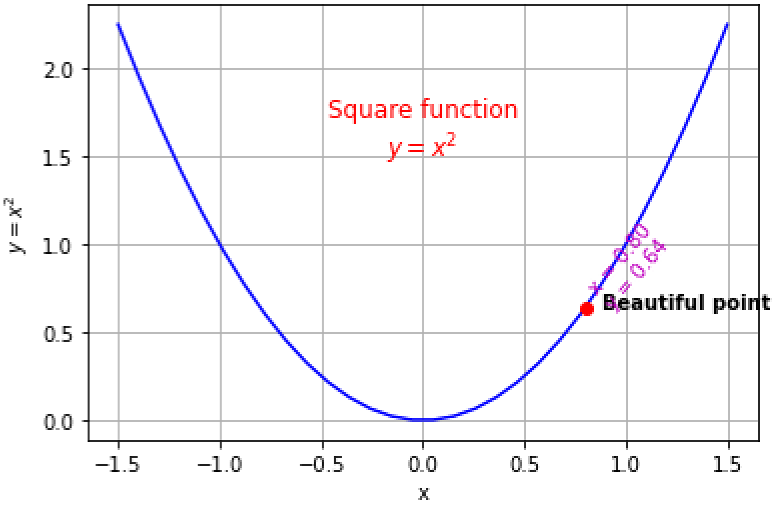




More Customizations

In [15]:

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

x = np.linspace(-1.5, 1.5, 30)
px = 0.8
py = px**2

plt.plot(x, x**2, "b-", px, py, "ro")

plt.text(0, 1.5, "Square function\n$y = x^2$", fontsize=12, color='r',
         ha="center")
plt.text(px + 0.08, py, "Beautiful point", ha="left", weight="heavy")
plt.text(px, py, "x = %0.2f\ny = %0.2f"%(px, py), rotation=50, color='m')
plt.grid()
plt.xlabel('x')
plt.ylabel('$y = x^2$')

plt.show()
```


Square function
$y = x^2$
x = 0.80
y = 0.64
Beautiful point
2.0
1.5
1.0
0.5
0.0
$y = x^2$
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x