# Matplotlib

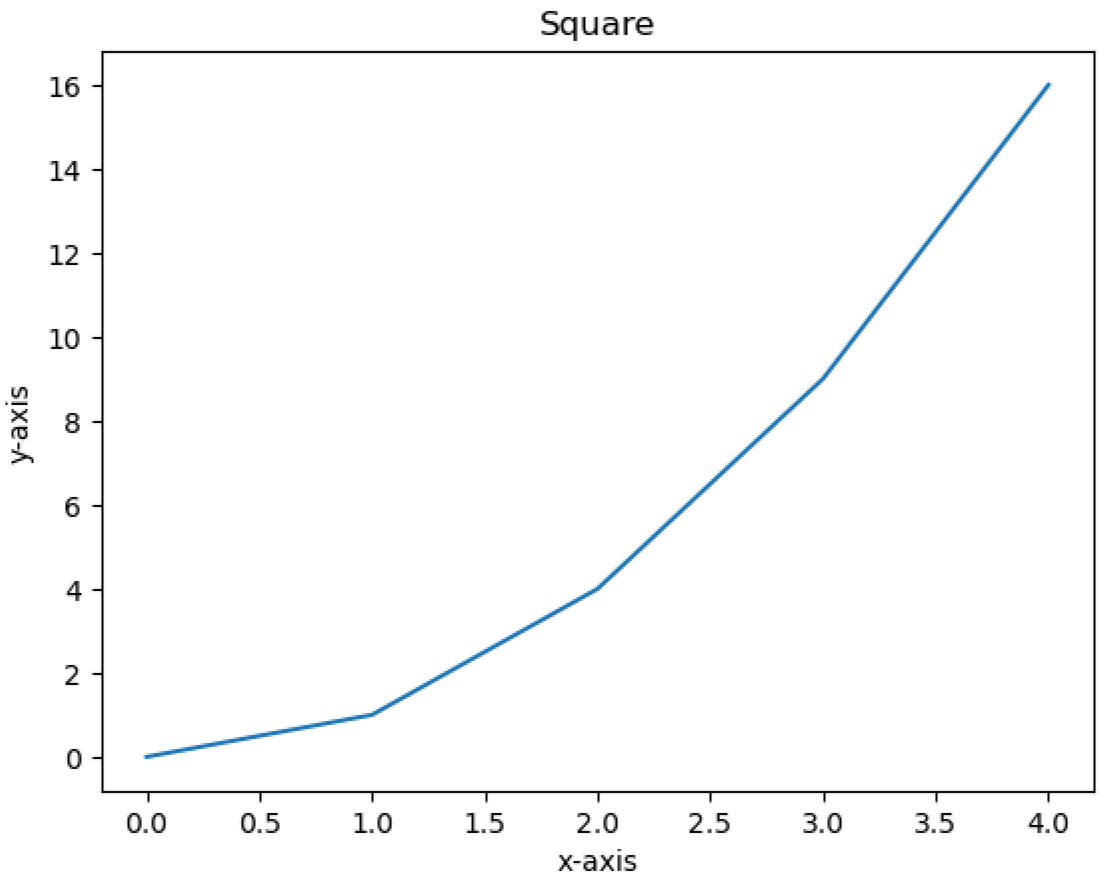
In [4]:
```
# Importing the NumPy library
# NumPy is used for numerical computations — it provides powerful tools for working w
import numpy as np

# Importing the Pandas library
# Pandas is mainly used for data manipulation and analysis — it allows you to work wi
import pandas as pd

# Importing the Matplotlib library (pyplot module)
# Matplotlib is used for creating visualizations — pyplot helps in plotting graphs li
import matplotlib.pyplot as plt
```

In [2]:
```
# Creating an array 'x' using NumPy's arange() function
# np.arange(start, stop) generates values starting from 0 up to (but not including) 5
# So, this will create an array: [0, 1, 2, 3, 4]
x = np.arange(0, 5)
x   # Display the array 'x' to verify its values

# Creating a new array 'y' which stores the square of each element in 'x'
# The expression x ** 2 means: square each element of the array 'x'
# So, y = [0², 1², 2², 3², 4²] → [0, 1, 4, 9, 16]
y = x ** 2
y   # Display the array 'y' to verify the squared values
```

Out[2]:
```
array([ 0,  1,  4,  9, 16])
```

In [3]:
```
# Plotting the graph of x vs y
# plt.plot(x, y) draws a line graph where 'x' values are on the x-axis and 'y' values
plt.plot(x, y)

# Adding a title to the graph
# The title "Square" describes what the graph represents
plt.title("Square")

# Labeling the x-axis
# This helps the viewer understand what values are shown on the horizontal axis
plt.xlabel('x-axis')

# Labeling the y-axis
# This helps the viewer understand what values are shown on the vertical axis
plt.ylabel('y-axis')
```

Out[3]:
```
Text(0, 0.5, 'y-axis')
```

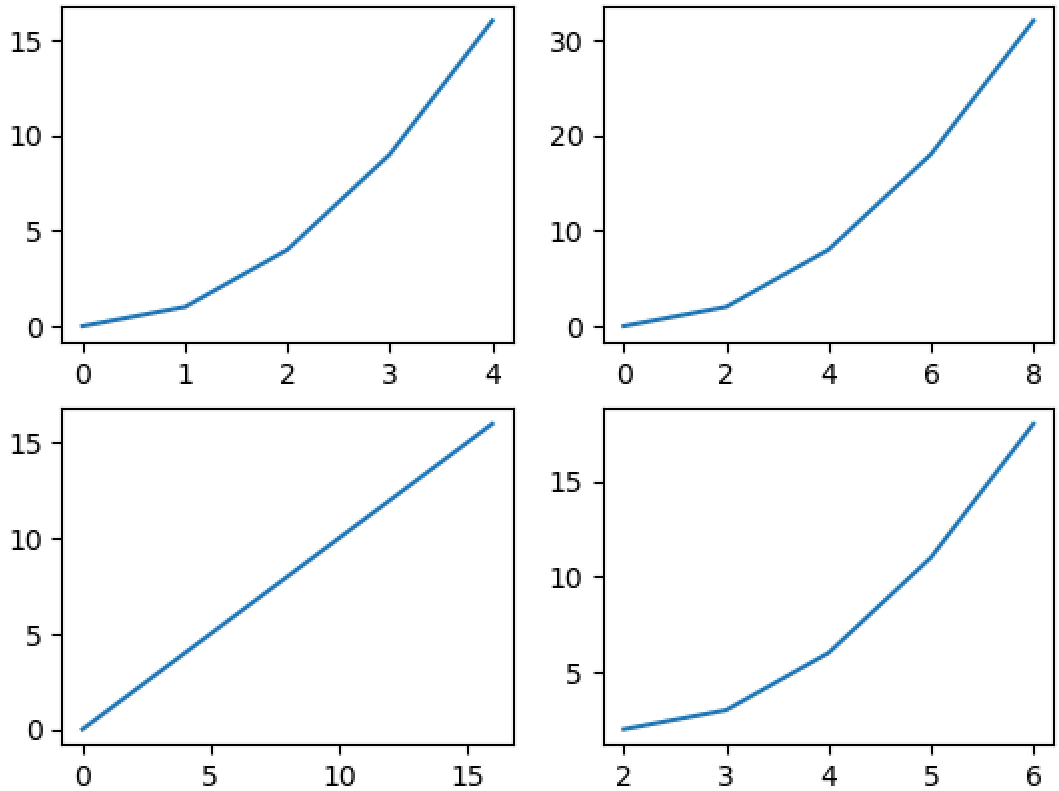



In [5]:

```python
# Creating multiple plots (subplots) in a single figure

# plt.subplot(nrows, ncols, index)
# nrows = number of rows in the grid
# ncols = number of columns in the grid
# index = position number of the current plot (starts from 1)

# 1️⃣ Create a 2x2 grid of plots and select position 1
plt.subplot(2, 2, 1)  # This means: 2 rows, 2 columns, plot in position 1 (top-left)
plt.plot(x, y)         # Plot the original graph (x vs y)

# 2️⃣ Create a subplot at position 3 (bottom-left)
plt.subplot(2, 2, 3)
plt.plot(x**2, x**2)   # Here both axes have squared values — it will be a straight l

# 3️⃣ Create a subplot at position 4 (bottom-right)
plt.subplot(2, 2, 4)
plt.plot(x + 2, y + 2) # Shifts both x and y values by +2 — moves the curve upward ar

# 4️⃣ Create a subplot at position 2 (top-right)
plt.subplot(2, 2, 2)
plt.plot(x * 2, y * 2) # Multiplies both x and y by 2 — stretches the curve

# Note: The figure will show 4 small graphs (subplots) arranged in a 2x2 grid.
```

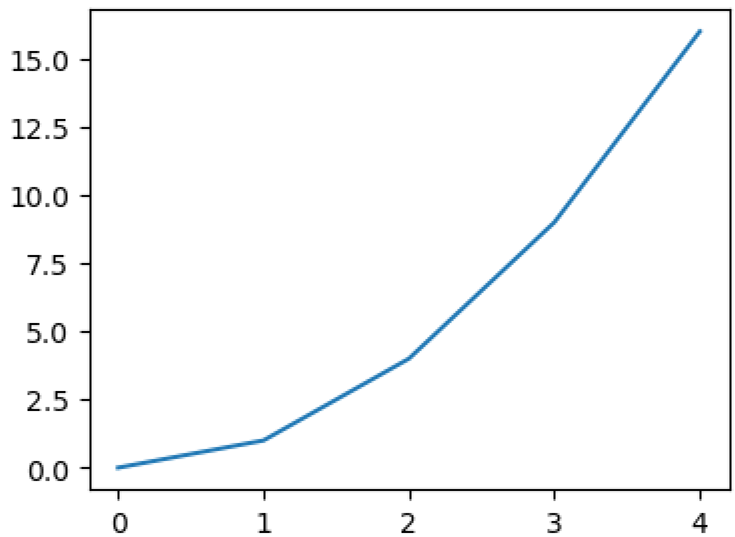
Out[5]:

```
[<matplotlib.lines.Line2D at 0x1b1095ae850>]
```




In [7]:

```
# Creating a new figure object
# A figure is like a blank canvas where we can draw one or more plots (axes)
fig = plt.figure()

# Adding axes to the figure manually
# fig.add_axes([left, bottom, width, height])
# These values are in fraction of the figure size (0 to 1)
# left = distance from left edge, bottom = distance from bottom edge
# width = width of the plot, height = height of the plot
axis1 = fig.add_axes([0.1, 0.1, 0.5, 0.5])  # Creates an axes inside the figure

# Plotting x vs y on the manually created axes
# Using axis1.plot() instead of plt.plot() — this is necessary when using add_axes
axis1.plot(x, y)

plt.show()

# Note: This allows precise control of where the plot appears in the figure.
```

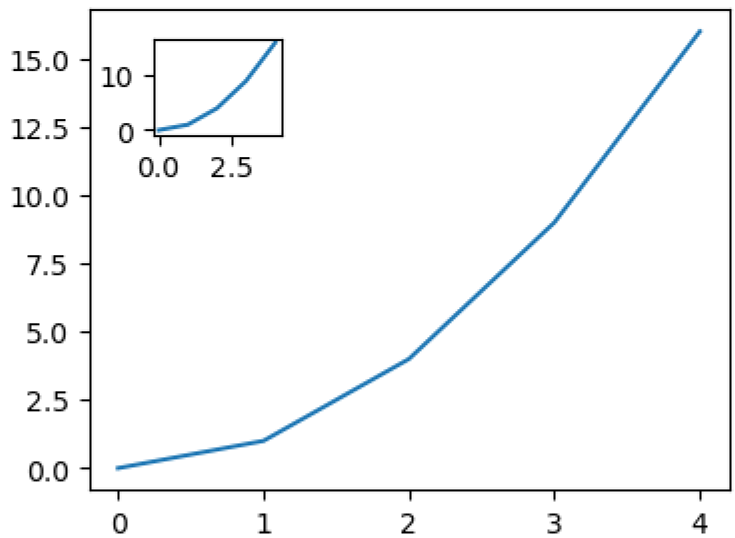



In [8]:

```
# Creating a new figure (blank canvas)
fig = plt.figure()

# Adding the first axes (main plot)
# fig.add_axes([left, bottom, width, height])
# Values are in fraction of the figure size (0 to 1)
# axis1 will be the larger/main plot
axis1 = fig.add_axes([0.1, 0.01, 0.5, 0.5])  # [x-pos, y-pos, width, height]

# Adding the second axes (small inset plot)
# This will appear as a smaller plot inside the same figure
axis2 = fig.add_axes([0.15, 0.38, 0.1, 0.1])  # Smaller plot over the main plot

# Plotting on the first axes (main plot)
axis1.plot(x, y)  # Main line graph

# Plotting on the second axes (small inset plot)
axis2.plot(x, y)  # Smaller version of the same line graph

# ✅ Output meaning:
# - You will see one big plot (axis1) and a smaller inset plot (axis2)
#   inside the same figure window.
# - This is useful for highlighting a specific part of your data or comparison.
```

Out[8]:

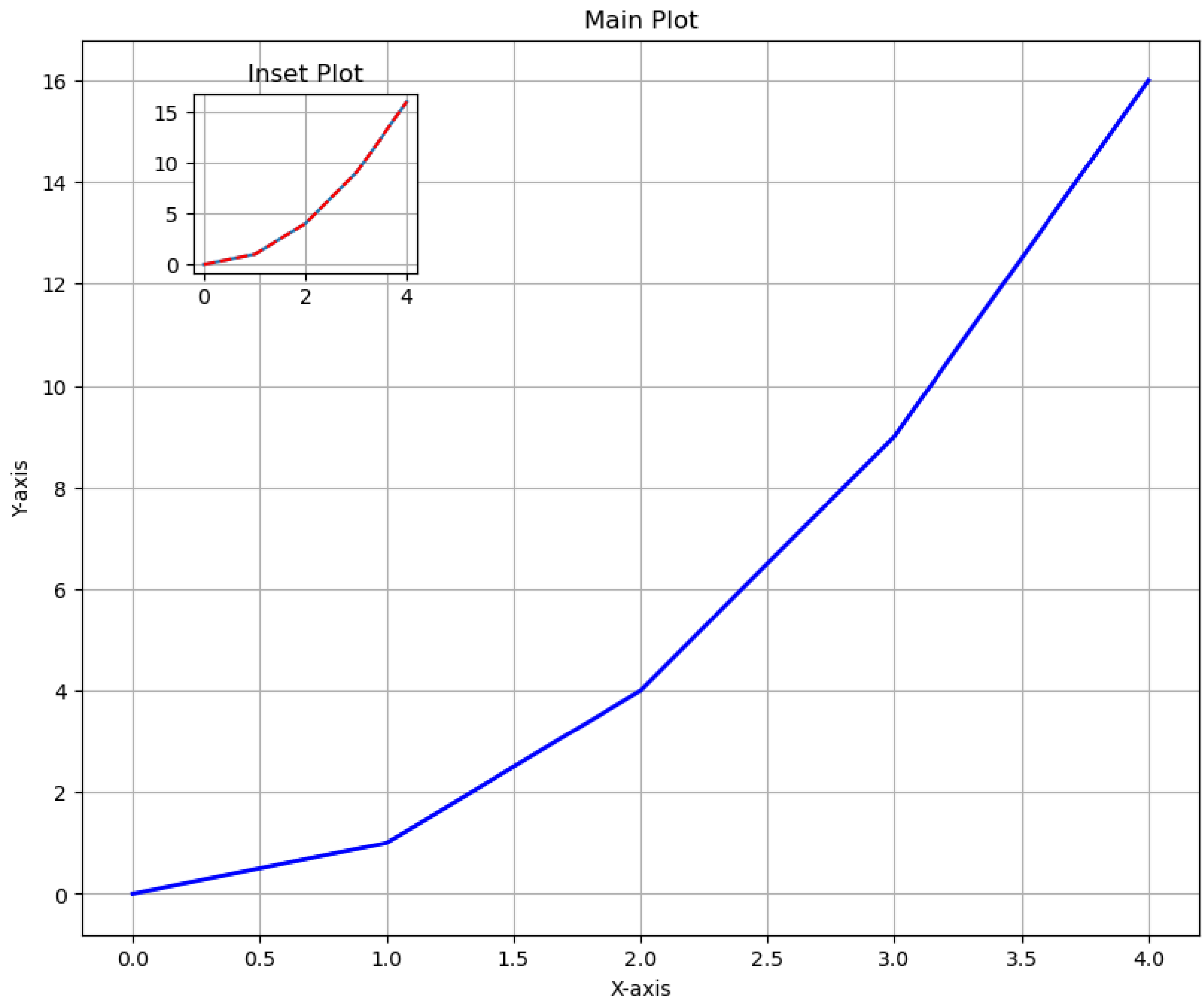
```
[<matplotlib.lines.Line2D at 0x1b1097fccd0>]
```




In [14]:

```
# Creating a new figure with custom size
# figsize = (width, height) in inches
# This makes the figure bigger or smaller depending on your preference
fig = plt.figure(figsize=(15, 12))  # Width = 15, Height = 12

# Adding the first axes (main plot)
# [left, bottom, width, height] — values are fractions of the figure
axis1 = fig.add_axes([0.1, 0.01, 0.5, 0.5])  # Main larger plot

# Adding the second axes (smaller inset plot)
axis2 = fig.add_axes([0.15, 0.38, 0.1, 0.1])  # Smaller plot inside the figure

# Plotting on the first axes (main plot)
axis1.plot(x, y)  # Main graph showing x vs y

# Plotting on the second axes (inset plot)
axis2.plot(x, y)  # Smaller inset graph showing the same data

# ✅ Notes:
# - `figsize` allows you to make the overall figure bigger, which is useful for prese
# - The positions of axes (axis1 and axis2) remain proportional to the figure size.
# - You can further customize each axis with labels, titles, colors, or grids.



axis1.set_title("Main Plot")
axis2.set_title("Inset Plot")
axis1.set_xlabel("X-axis")
axis1.set_ylabel("Y-axis")
axis1.grid(True)
axis2.grid(True)
axis1.plot(x, y, color='blue', linewidth=2)
axis2.plot(x, y, color='red', linestyle='--')
```

Out[14]: `[<matplotlib.lines.Line2D at 0x1b109813d90>]`

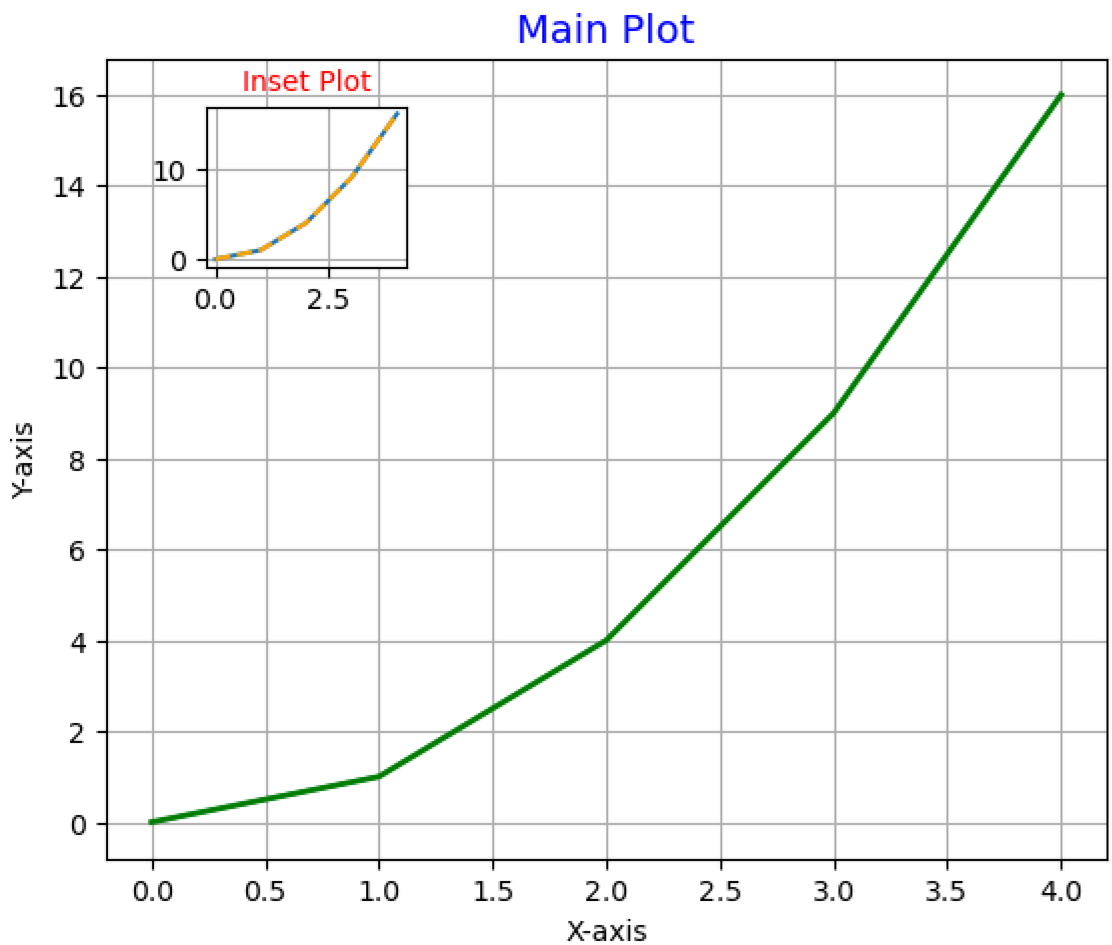



In [16]:

```
# Creating a new figure with custom size and resolution
# figsize = (width, height) in inches → changes the size of the figure
# dpi = dots per inch → controls the resolution/clarity of the figure
fig = plt.figure(figsize=(10, 8), dpi=100)  # Width = 10, Height = 8, Resolution = 10

# Adding the first axes (main plot)
# [left, bottom, width, height] — values are fractions of the figure
axis1 = fig.add_axes([0.1, 0.01, 0.5, 0.5])  # Main larger plot

# Adding the second axes (smaller inset plot)
axis2 = fig.add_axes([0.15, 0.38, 0.1, 0.1])  # Smaller plot inside the figure

# Plotting on the first axes (main plot)
axis1.plot(x, y)  # Main line graph showing x vs y

# Plotting on the second axes (inset plot)
axis2.plot(x, y)  # Smaller inset graph showing the same data

# ✅ Notes:
# 1. `figsize` lets you make the figure bigger/smaller (good for presentations).
# 2. `dpi` increases the sharpness/resolution of the figure — higher dpi = clearer in
# 3. `add_axes()` lets you manually control the exact placement and size of plots.
# 4. Each axis can be customized individually with titles, labels, colors, and grids.
```

```python
axis1.set_title("Main Plot", fontsize=14, color='blue')
axis2.set_title("Inset Plot", fontsize=10, color='red')
axis1.set_xlabel("X-axis")
axis1.set_ylabel("Y-axis")
axis1.grid(True)
axis2.grid(True)
axis1.plot(x, y, color='green', linewidth=2)
axis2.plot(x, y, color='orange', linestyle='--')
```

Out[16]: [<matplotlib.lines.Line2D at 0x1b109b64550>]



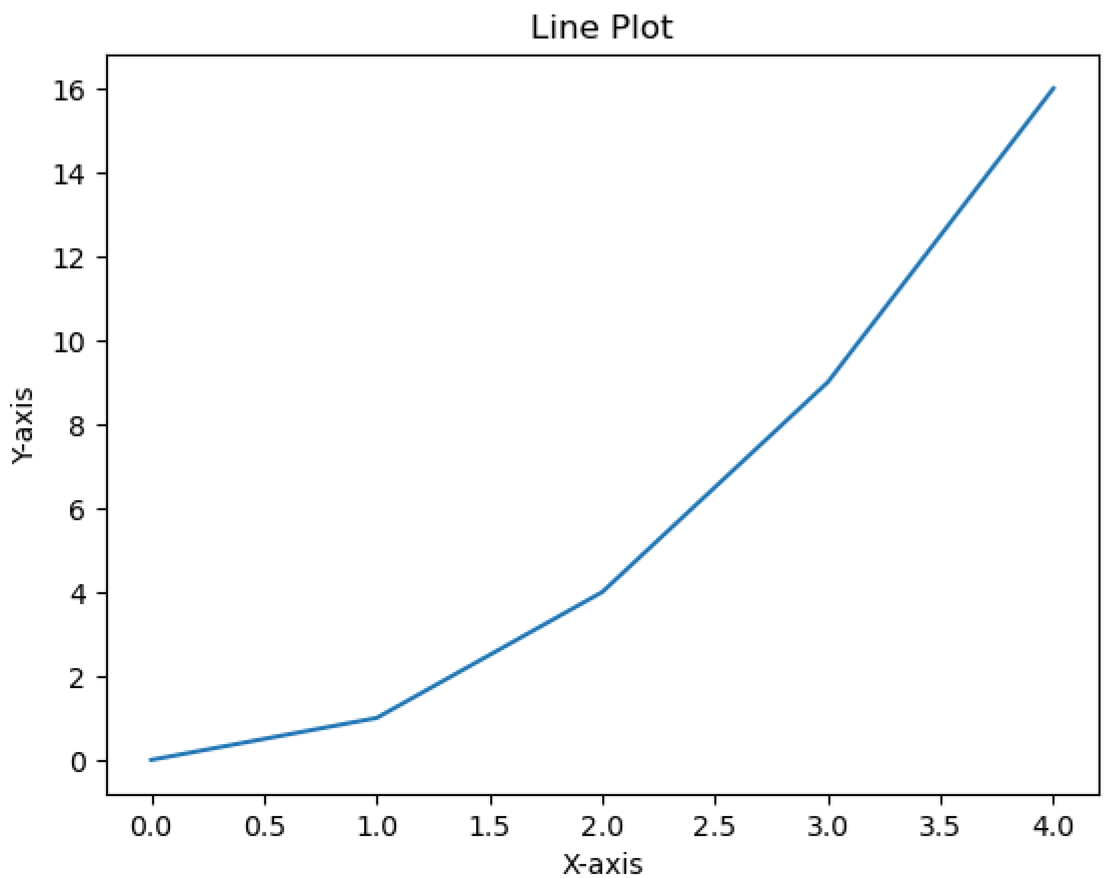

# Type of Plots

## 1️⃣ Line Plot

- Shows the relationship between x and y as a line.
- Useful for trends over time or continuous data.

In [17]:
```python
plt.plot(x, y)
plt.title("Line Plot")
plt.xlabel("X-axis")
```

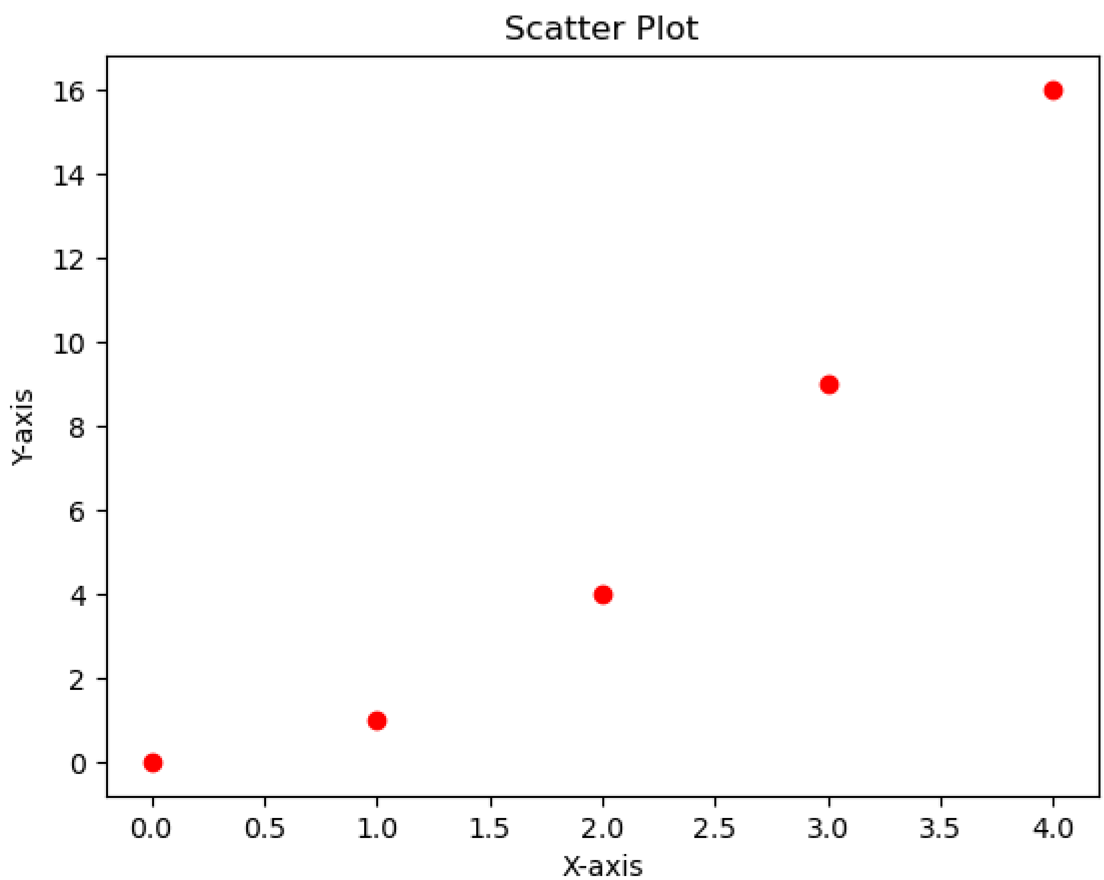
```
plt.ylabel("Y-axis")
plt.show()
```




## 2️⃣ Scatter Plot

- Shows individual data points.
- Useful for visualizing correlation between two variables.

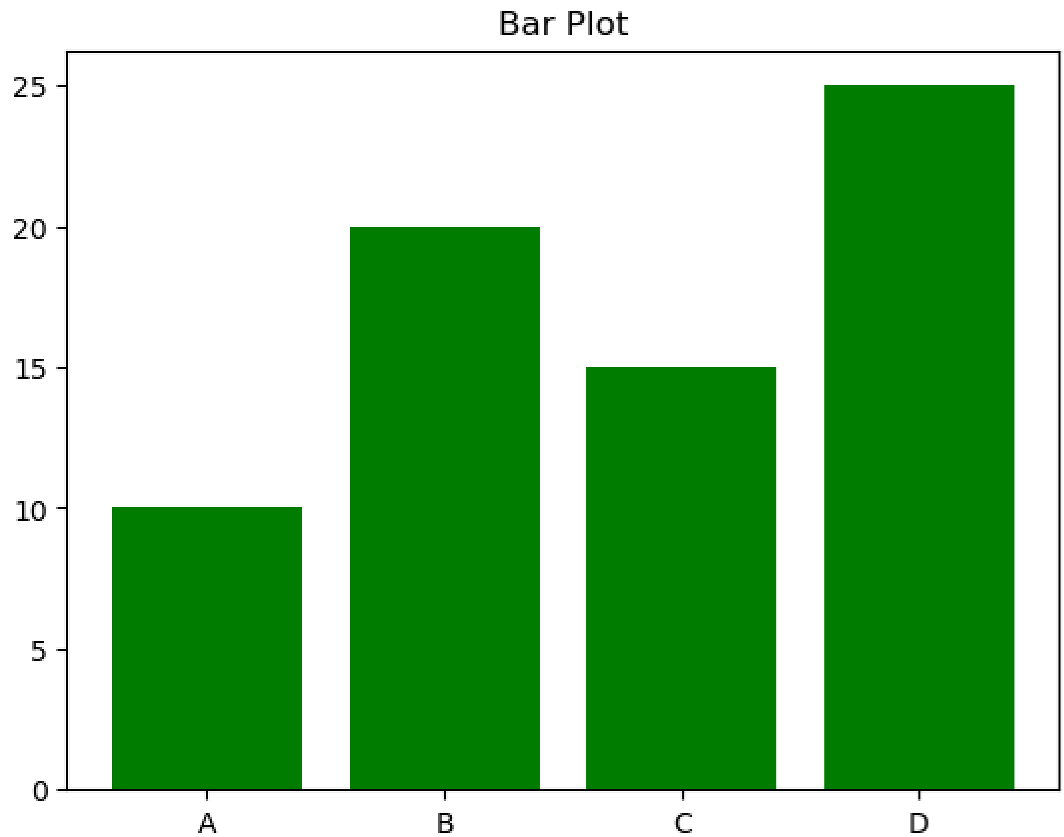
In [18]:
```
plt.scatter(x, y, color='red', marker='o')
plt.title("Scatter Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

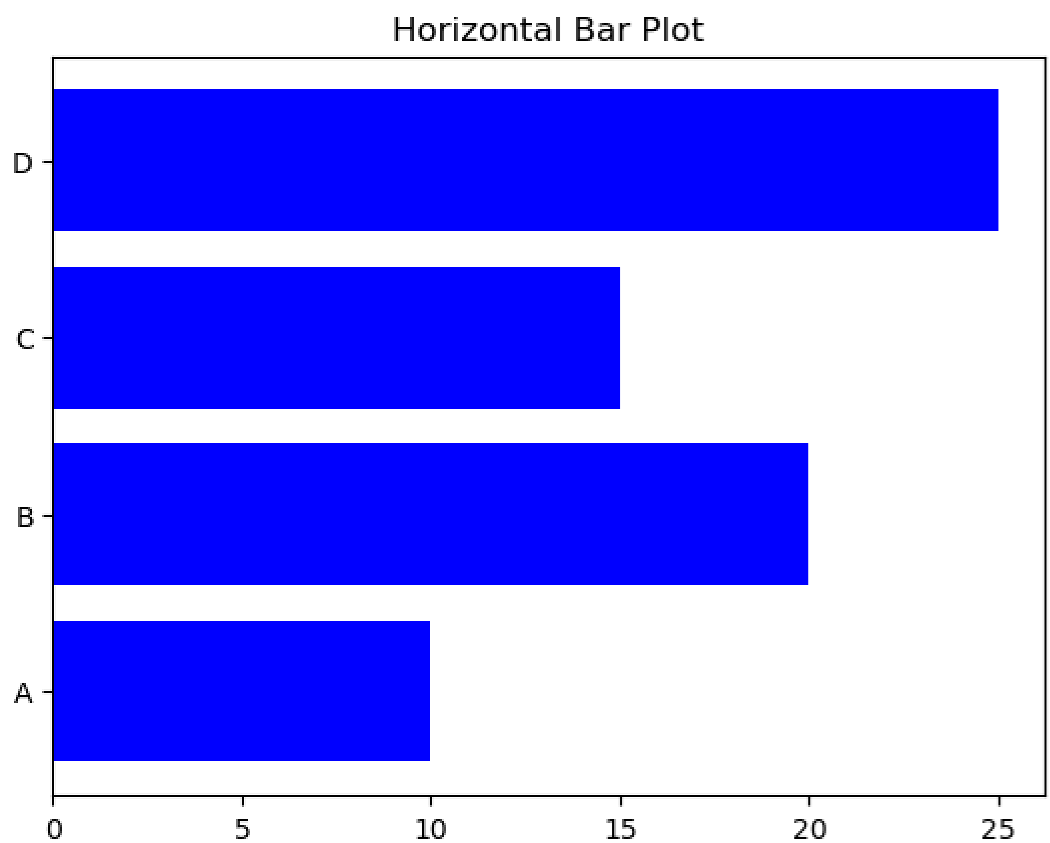



# 3️⃣ Bar Plot

- Shows comparison between categories.
- Can be vertical (bar) or horizontal (barh).

In [20]:
```
categories = ['A', 'B', 'C', 'D']
values = [10, 20, 15, 25]

# Vertical bar plot
plt.bar(categories, values, color='green')
plt.title("Bar Plot")
plt.show()

# Horizontal bar plot
plt.barh(categories, values, color='blue')
plt.title("Horizontal Bar Plot")
plt.show()
```

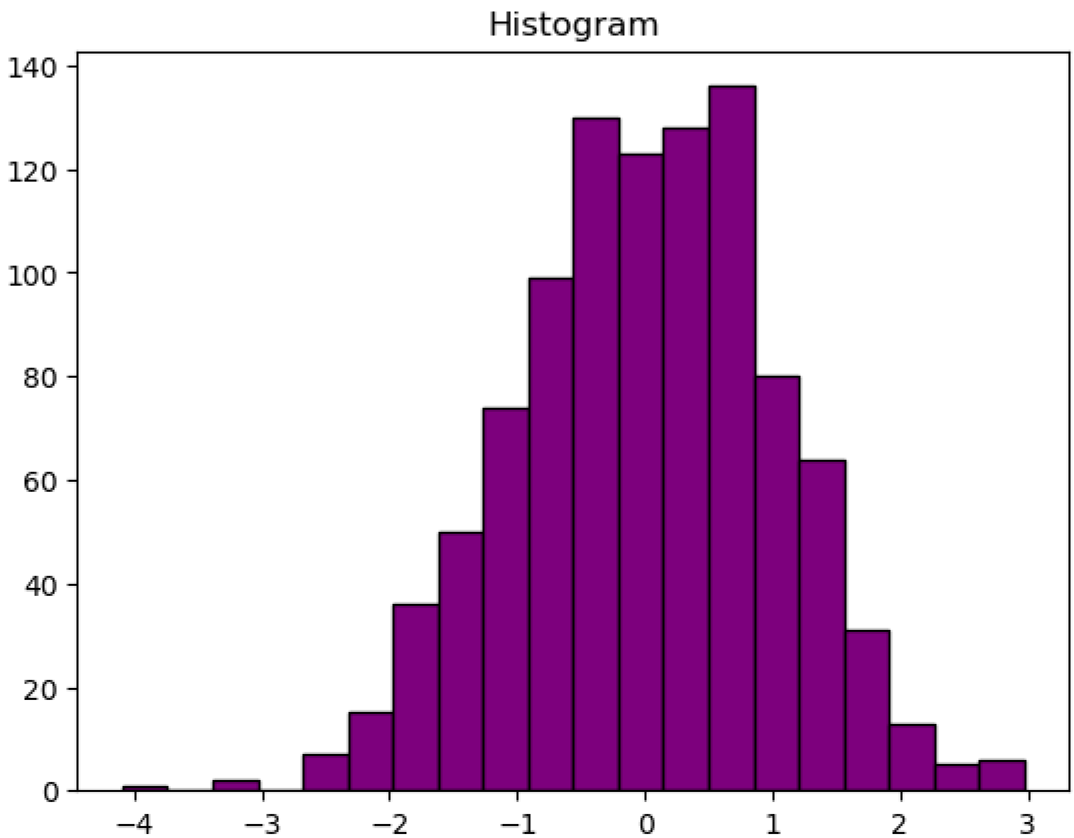

Bar Plot
25
20
15
10
5
0
A
B
C
D


Horizontal Bar Plot
D
C
B
A
0
5
10
15
20
25

## 4 Histogram

- Shows frequency distribution of numerical data.
- Useful for understanding data distribution.

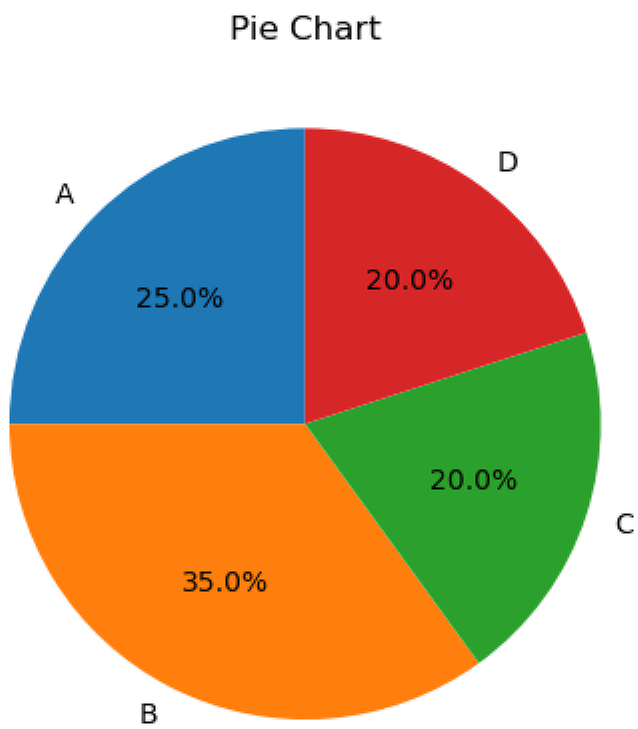
In [21]:
```
data = np.random.randn(1000)  # 1000 random numbers
plt.hist(data, bins=20, color='purple', edgecolor='black')
plt.title("Histogram")
plt.show()
```




# 5️⃣ Pie Chart

- Shows percentages/proportions of categories.

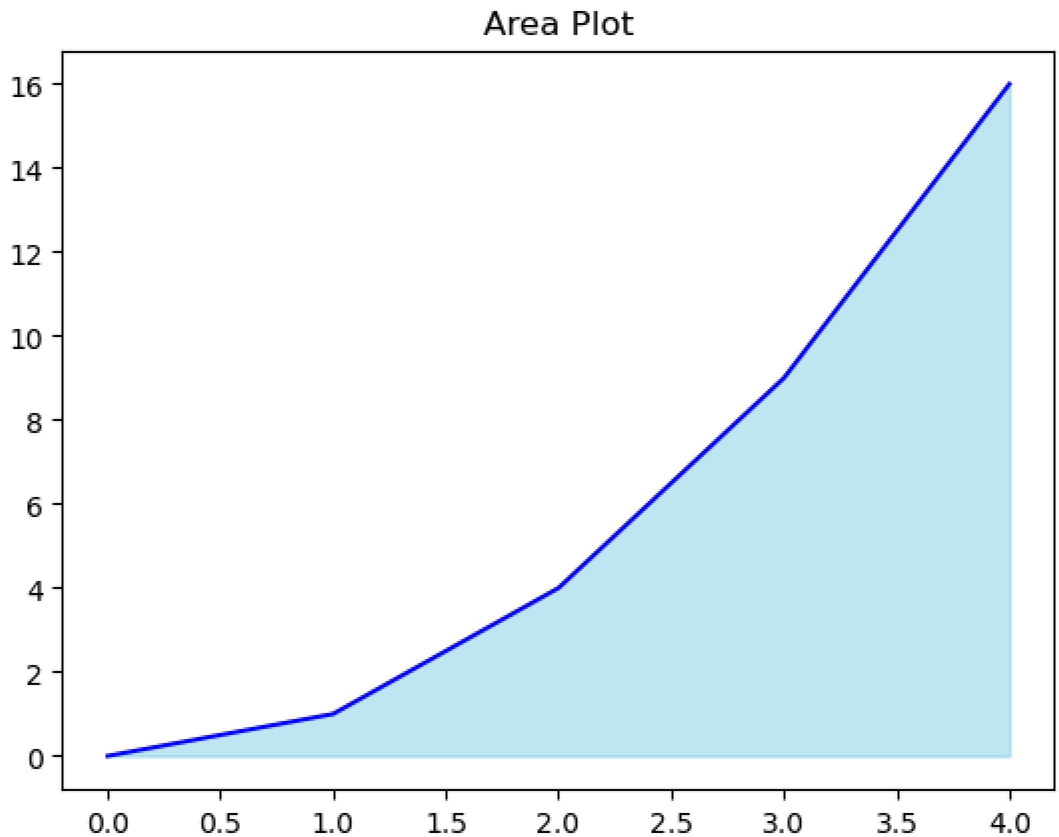
In [22]:
```
sizes = [25, 35, 20, 20]
labels = ['A', 'B', 'C', 'D']
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title("Pie Chart")
plt.show()
```




# 6 Area Plot

- Similar to a line plot but the area under the line is filled.

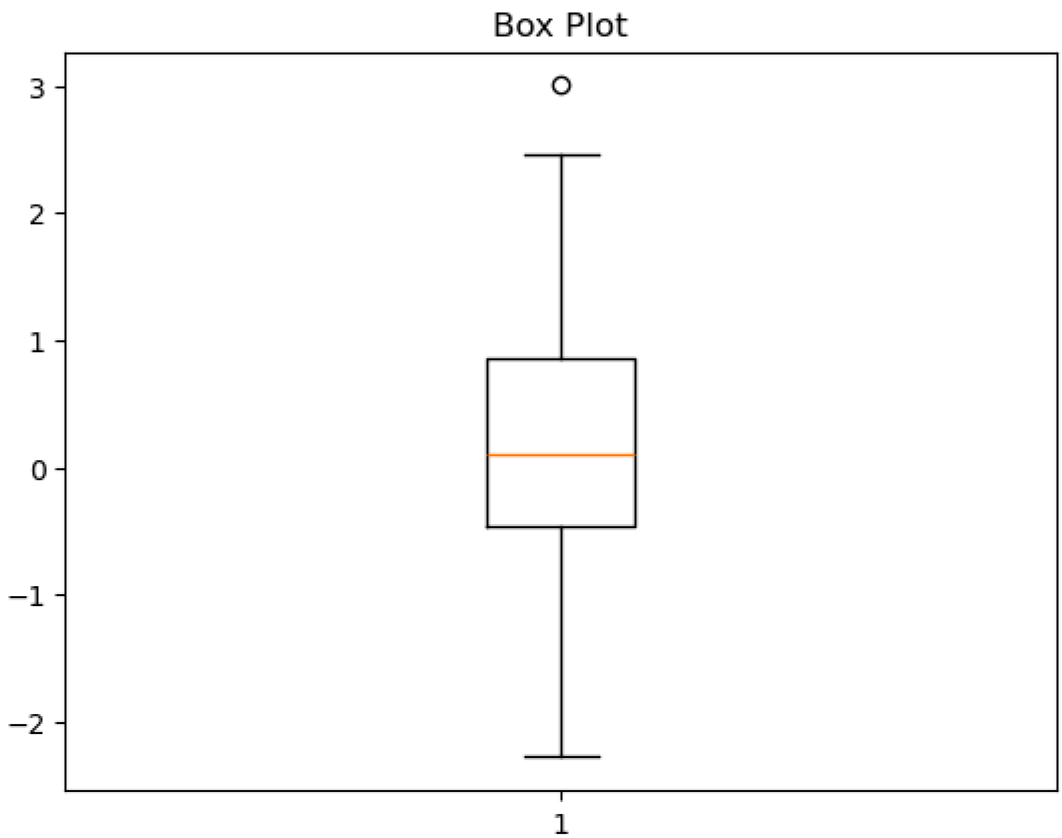
In [23]:
```
plt.fill_between(x, y, color='skyblue', alpha=0.5)
plt.plot(x, y, color='blue')
plt.title("Area Plot")
plt.show()
```




## 7️⃣ Box Plot

- Shows data distribution with median, quartiles, and outliers.

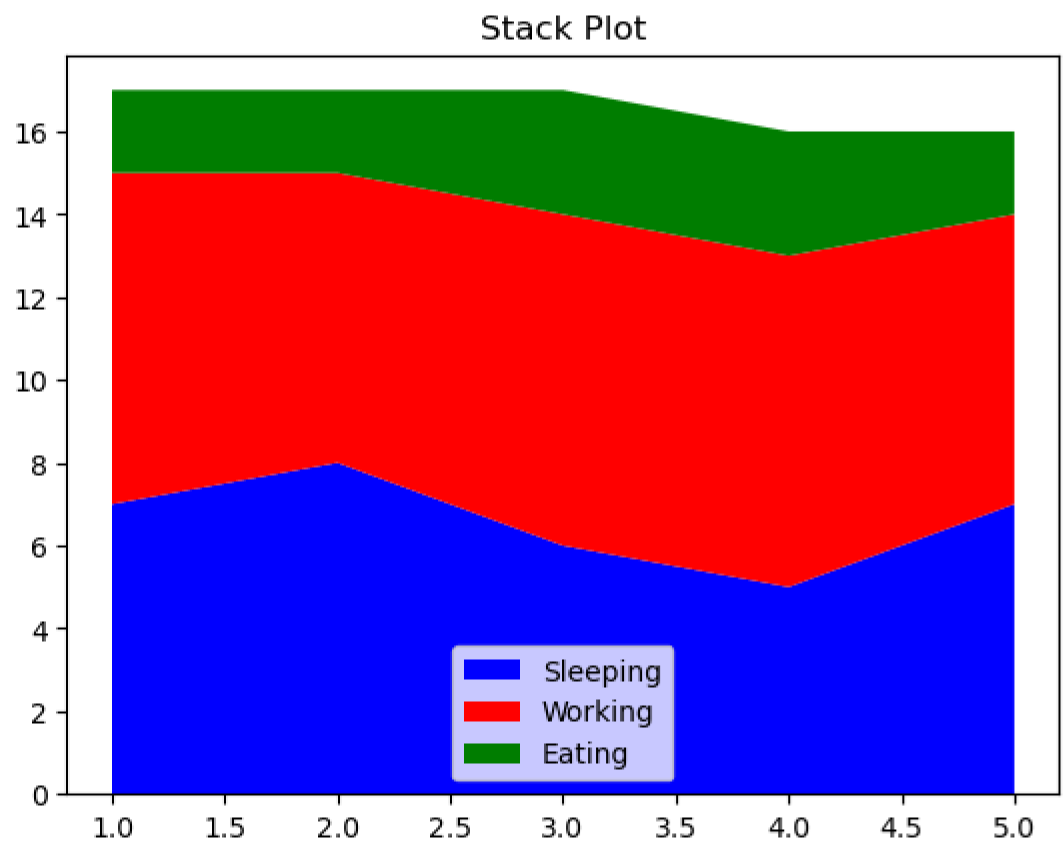
In [25]:
```
data = np.random.randn(100)
plt.boxplot(data)
plt.title("Box Plot")
plt.show()
```




# 8️⃣ Stack Plot

- Shows cumulative data over time (like layers).

In [26]:
```
days = [1, 2, 3, 4, 5]
sleeping = [7, 8, 6, 5, 7]
working = [8, 7, 8, 8, 7]
eating = [2, 2, 3, 3, 2]

plt.stackplot(days, sleeping, working, eating, labels=['Sleeping','Working','Eating']
plt.legend()
plt.title("Stack Plot")
plt.show()
```



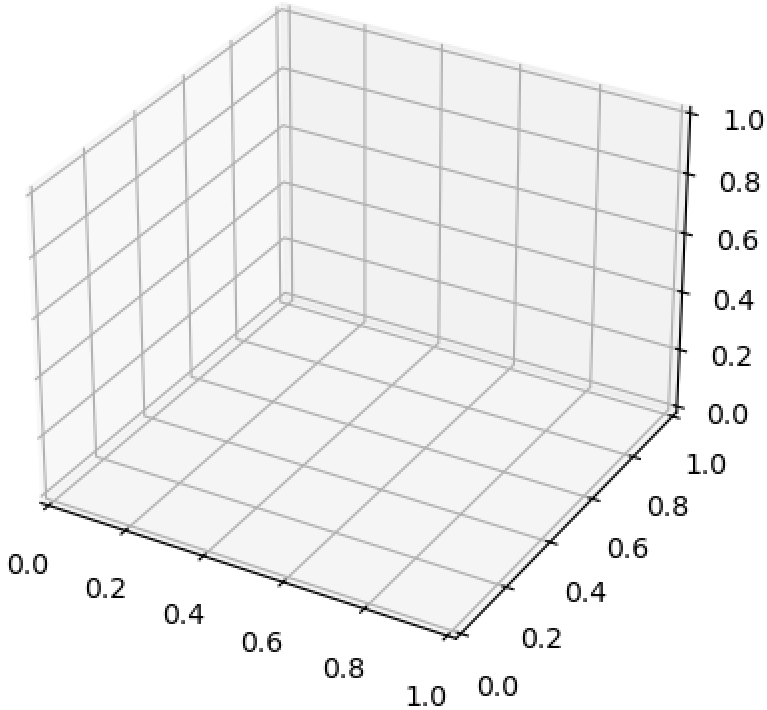

# 3D Plots:

## 1️⃣ Import 3D toolkit

In [27]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # This is required for 3D plotting
import numpy as np
```

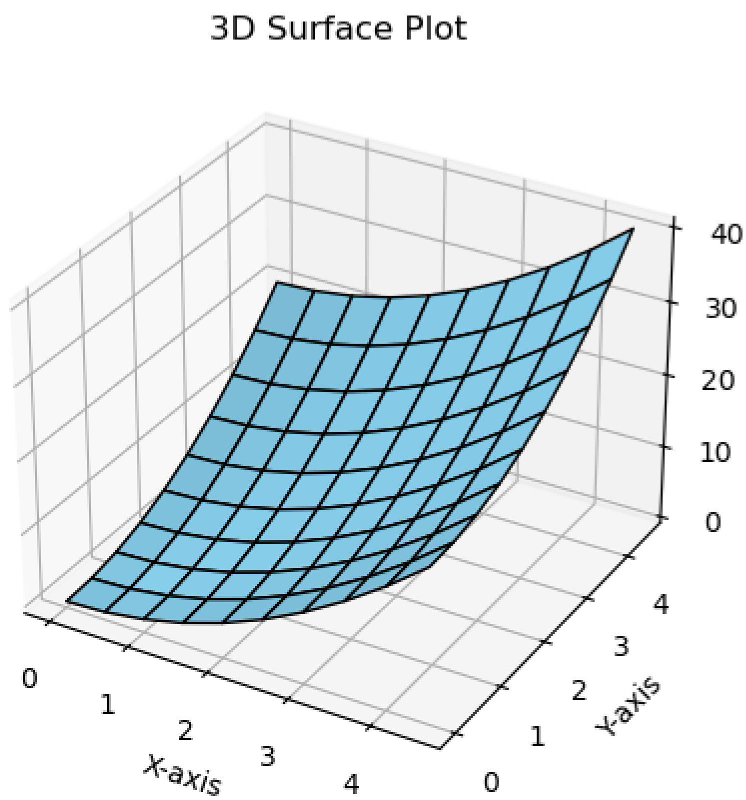
## 2️⃣ Create a 3D figure and axes

In [29]:
```
fig = plt.figure()                 # Create a new figure
ax = fig.add_subplot(111, projection='3d')  # Add a 3D subplot


# projection='3d' tells Matplotlib to create a 3D axes.
# 111 means 1 row, 1 column, plot number 1 (standard subplot notation).
```



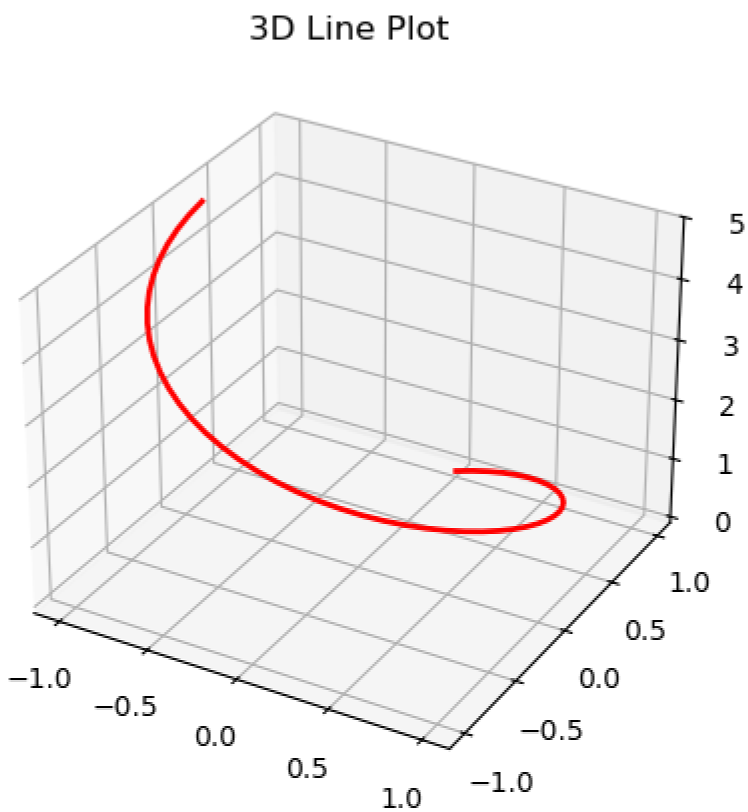

# 4️⃣ Plot 3D surface

In [34]:
```
# Step 1: Import required libraries
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D  # For 3D plotting
import numpy as np

# Step 2: Create x, y, z data
x = np.arange(0, 5, 0.5)
y = np.arange(0, 5, 0.5)
X, Y = np.meshgrid(x, y)  # Create a grid of x and y values
Z = X**2 + Y**2           # Example: z = x^2 + y^2

# Step 3: Create a 3D figure and axes
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')  # 3D axes

# Step 4: Plot the 3D surface
ax.plot_surface(X, Y, Z, color='skyblue', edgecolor='black')

# Step 5: Customize the plot
ax.set_title("3D Surface Plot")
ax.set_xlabel("X-axis")
ax.set_ylabel("Y-axis")
ax.set_zlabel("Z-axis")

# Step 6: Show the plot
plt.show()
```




## 5️⃣ Other 3D plot types

In [32]:
```
z = np.arange(0, 5, 0.1)
x = np.sin(z)
y = np.cos(z)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot(x, y, z, color='red', linewidth=2)
ax.set_title("3D Line Plot")
plt.show()
```




In [33]:
```
x = np.random.rand(50)
y = np.random.rand(50)
z = np.random.rand(50)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x, y, z, color='green', s=50)  # s = marker size
ax.set_title("3D Scatter Plot")
plt.show()
```

3D Scatter Plot