Lab 1:

WAP a program in python to represent line graph where x & y will be user inputs

lab 2:

Write a code in python to demonstrate python graph where the no of vertices is user inputs

lab 3 :

Write a code in python to demonstrate the concept of coloring of a graph where the no of vertices is user input

Lab 4: Bar chart,pie chart ,line graph ,area graph ,scatter plot in excel.

Lab 5: Bar chart,pie chart ,line graph ,area graph ,scatter plot using excel in jupyter notebook.

Lab 6:

Lab 7: Linear regression

Lab 8:Multiple linear regression

Lab 9: Program to create and handle frequency table using python

Lab 10: Program to demonstrate sampling distribution using python.

Assignment 1

Write a program in python to represent a line graph where x and y value will be user inputs.

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

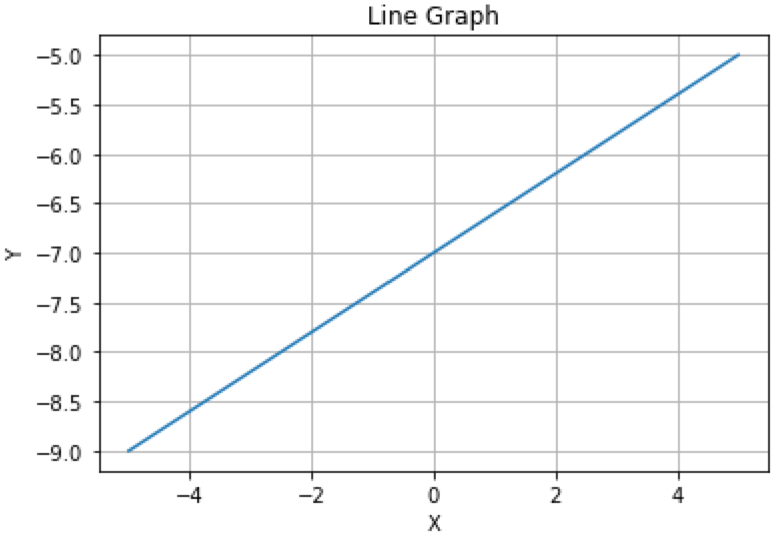
In [2]:
```
x_input = input("Enter a list of x values separated by spaces: ")
# Remove commas and split the string into a list of values
x_values = [float(val) for val in x_input.replace(',', '').split()]
```

```
Enter a list of x values separated by spaces: -5 , 0 , 5
```

In [3]:
```
y_input = input("Enter a list of y values separated by spaces: ")
# Remove commas and split the string into a list of values
y_values = [float(val) for val in y_input.replace(',', '').split()]
```

```
Enter a list of y values separated by spaces: -9 , -7 , -5
```

In [5]:
```
plt.plot(x_values, y_values)
plt.xlabel('X')
plt.ylabel('Y')
plt.title('Line Graph')
plt.grid(True)
plt.show()
```



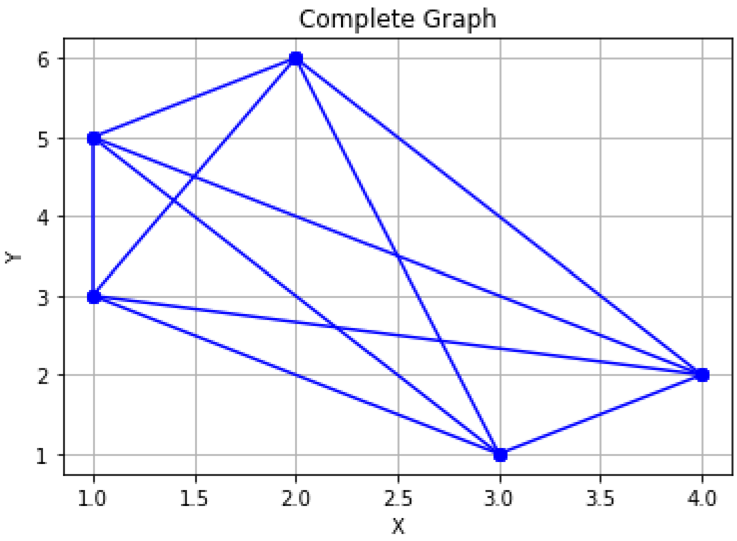
Assignment 2

Write a program in python to represent a complete graph by taking user inputs.

In [17]:

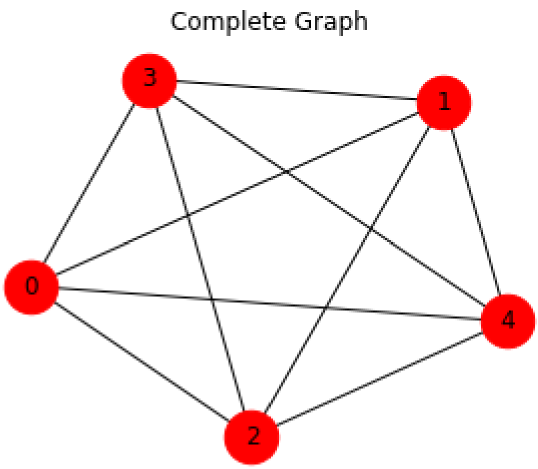
```
import matplotlib.pyplot as plt

# Take user input for number of vertices
num_vertices = int(input("Enter the number of vertices: "))

# Take user input for coordinates of vertices
vertices = [tuple(map(float, input(f"Enter coordinates for vertex {i+1} (se

# Plot the complete graph
plt.figure()
for i, (x1, y1) in enumerate(vertices):
    for x2, y2 in vertices[i+1:]:
        plt.plot([x1, x2], [y1, y2], 'bo-')

plt.title('Complete Graph')
plt.xlabel('X')
plt.ylabel('Y')
plt.grid(True)
plt.show()
```

```
Enter the number of vertices: 5
Enter coordinates for vertex 1 (separated by space): 2 6
Enter coordinates for vertex 2 (separated by space): 3 1
Enter coordinates for vertex 3 (separated by space): 1 3
Enter coordinates for vertex 4 (separated by space): 1 5
Enter coordinates for vertex 5 (separated by space): 4 2
```



```
In [20]: import networkx as nx

         # Take user input for the number of nodes
         num_nodes = int(input("Enter the number of nodes: "))

         # Create the complete graph
         G = nx.complete_graph(num_nodes)

         # Draw the graph with grid background
         plt.figure(figsize=(4, 3))
         pos = nx.spring_layout(G)  # Positions for all nodes
         nx.draw(G, pos, with_labels=True, node_color='red', node_size=700)

         plt.title('Complete Graph')
         plt.show()
```

```
Enter the number of nodes: 5
```



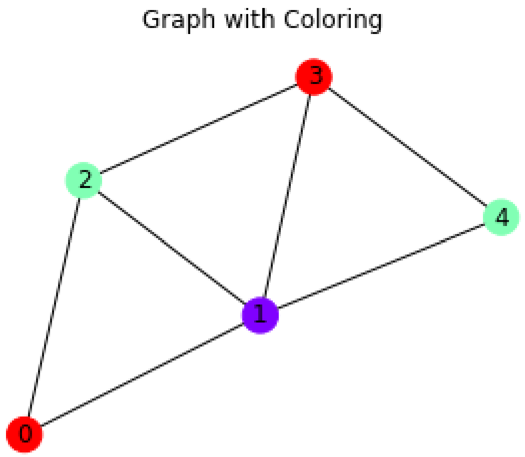
Assignment 3

Write a python program to demonstrate coloring of graph (Such that no two adjacent vertices are same color).

In [17]:

```
import networkx as nx
import matplotlib.pyplot as plt

# Define the graph
G = nx.Graph()
G.add_edges_from([(0, 1), (0, 2), (1, 2), (1, 3), (2, 3), (3, 4), (4, 1)])

# Find a coloring
coloring = nx.greedy_color(G, strategy="largest_first")

# Print the coloring
print("Vertex colors:")
for node, color in coloring.items():
    print(f"Vertex {node}: Color {color}")

# Draw the graph with vertex colors
plt.figure(figsize=(4, 3))
pos = nx.spring_layout(G)  # Positions for all nodes
node_colors = [coloring[node] for node in G.nodes()]  # Get colors of nodes
nx.draw(G, pos, with_labels=True, node_color=node_colors, cmap=plt.cm.rainb
plt.title('Graph with Coloring')
plt.show()
```

```
Vertex colors:
Vertex 1: Color 0
Vertex 2: Color 1
Vertex 3: Color 2
Vertex 0: Color 2
Vertex 4: Color 1
```



In [1]:
```
import openpyxl
import matplotlib.pyplot as plt
```

In [4]:
```
excel_file_path = r'C:\Users\lab\Desktop\Book1.xlsx'
```

In [5]:
```
workbook = openpyxl.load_workbook(excel_file_path)
```

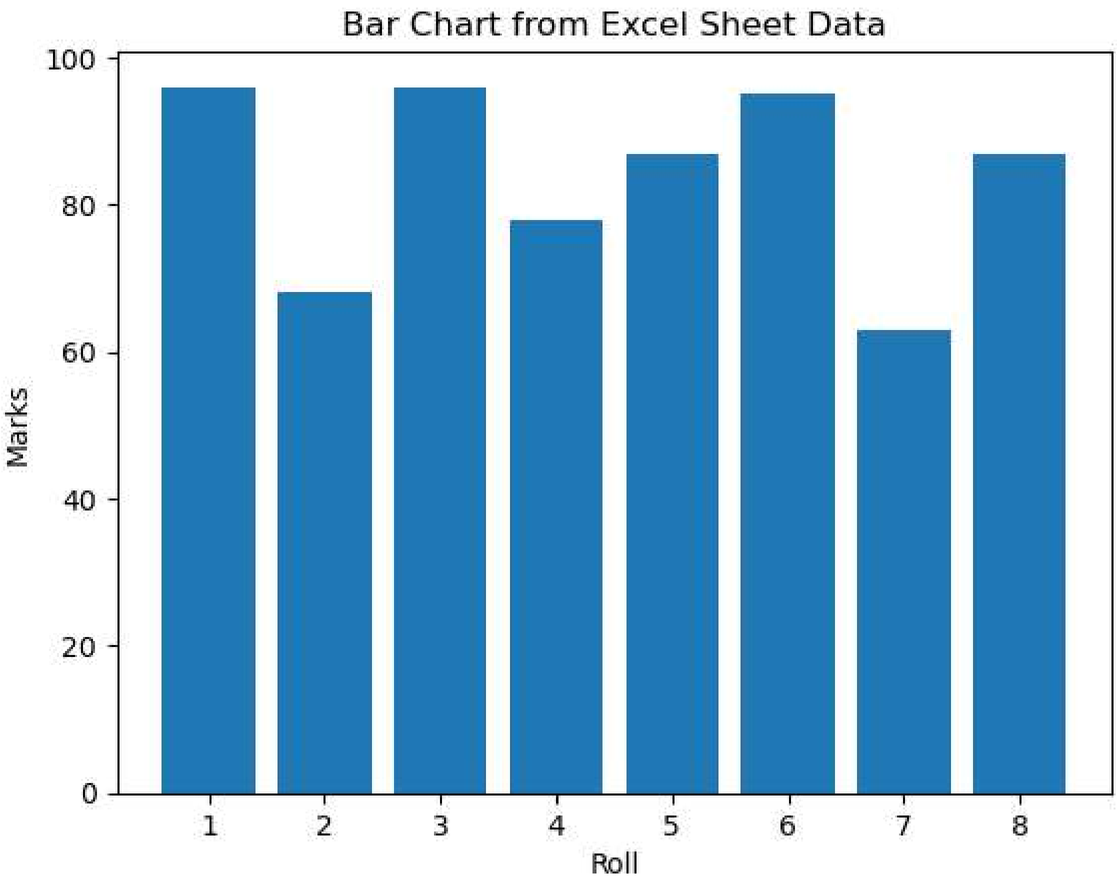
In [8]:
```
sheet = workbook['Sheet1']
```

In [9]:
```
data = {'Roll': [], 'Marks': []}
```

In [12]:
```
for row in sheet.iter_rows(min_row=2, values_only=True):  # Assuming data start
    roll, marks = row
    data['Roll'].append(roll)
    data['Marks'].append(marks)
```

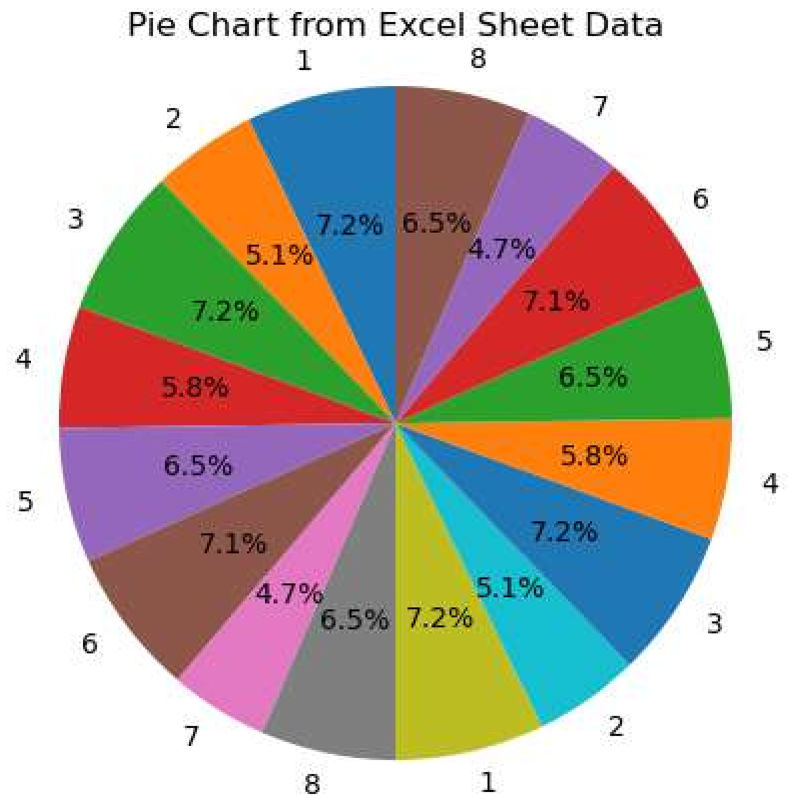
In [13]:
```
plt.bar(data['Roll'], data['Marks'])
plt.xlabel('Roll')
plt.ylabel('Marks')
plt.title('Bar Chart from Excel Sheet Data')
```

Out[13]:
```
Text(0.5, 1.0, 'Bar Chart from Excel Sheet Data')
```



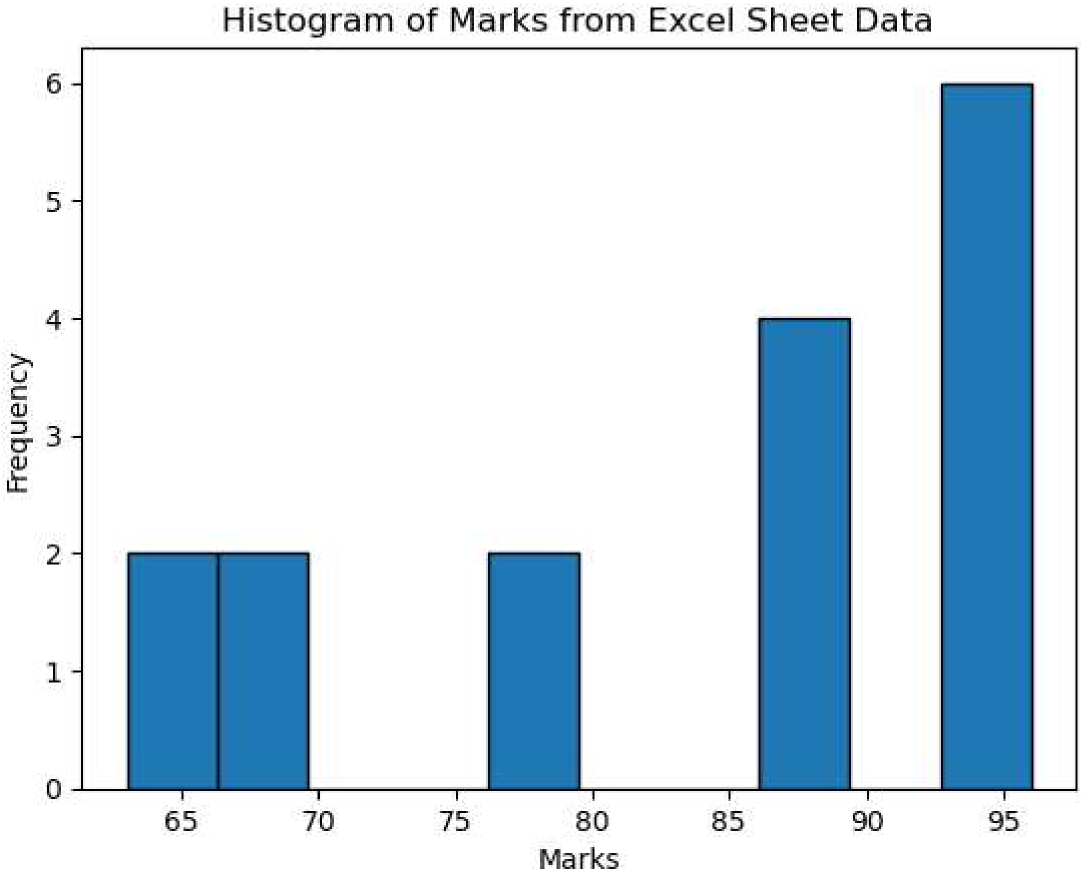
```
In [14]: plt.pie(data['Marks'], labels=data['Roll'], autopct='%1.1f%%', startangle=90)
         plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
         plt.title('Pie Chart from Excel Sheet Data')
```

```
Out[14]: Text(0.5, 1.0, 'Pie Chart from Excel Sheet Data')
```



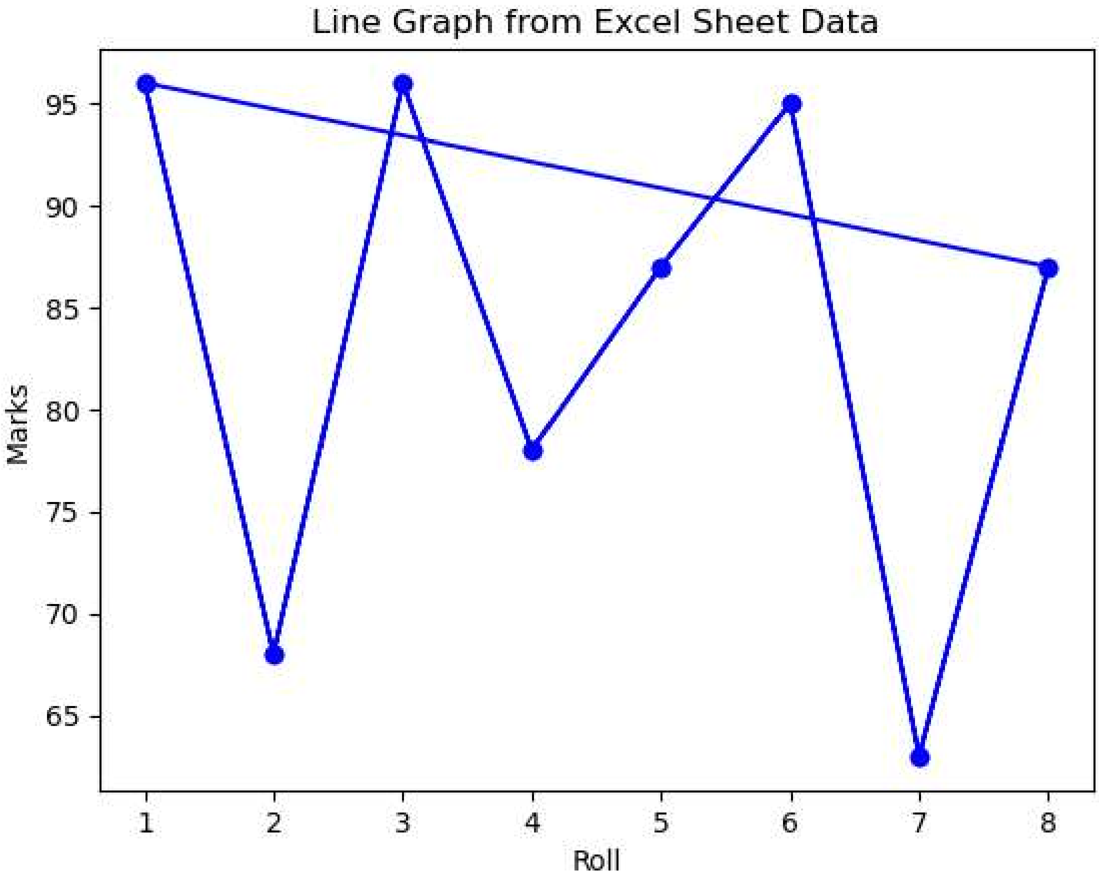
```
In [16]: plt.hist(data['Marks'], bins=10, edgecolor='black')  # Adjust the number of bir
         plt.xlabel('Marks')
         plt.ylabel('Frequency')
         plt.title('Histogram of Marks from Excel Sheet Data')
```

```
Out[16]: Text(0.5, 1.0, 'Histogram of Marks from Excel Sheet Data')
```



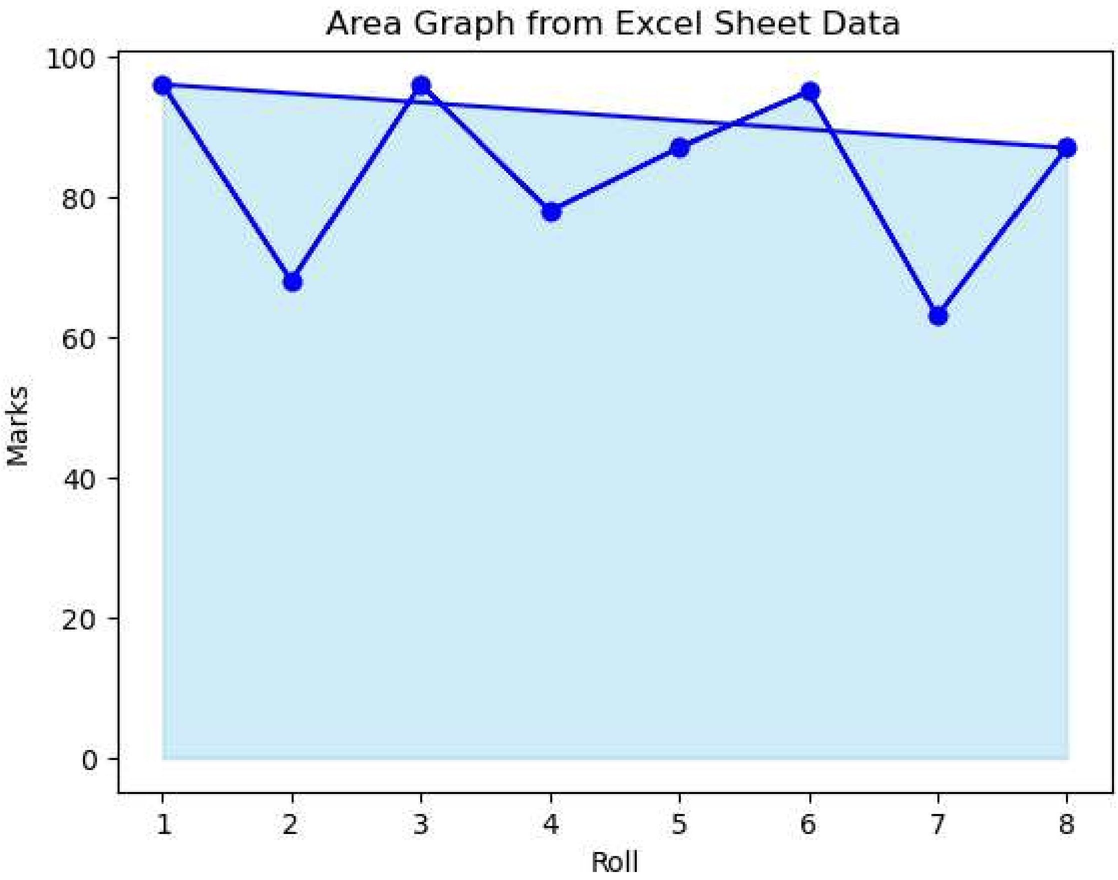
```
In [17]: plt.plot(data['Roll'], data['Marks'], marker='o', linestyle='-', color='b')
         plt.xlabel('Roll')
         plt.ylabel('Marks')
         plt.title('Line Graph from Excel Sheet Data')
```

```
Out[17]: Text(0.5, 1.0, 'Line Graph from Excel Sheet Data')
```



In [18]:
```
plt.fill_between(data['Roll'], data['Marks'], color='skyblue', alpha=0.4)
plt.plot(data['Roll'], data['Marks'], marker='o', linestyle='-', color='b')
plt.xlabel('Roll')
plt.ylabel('Marks')
plt.title('Area Graph from Excel Sheet Data')
```

Out[18]:
```
Text(0.5, 1.0, 'Area Graph from Excel Sheet Data')
```

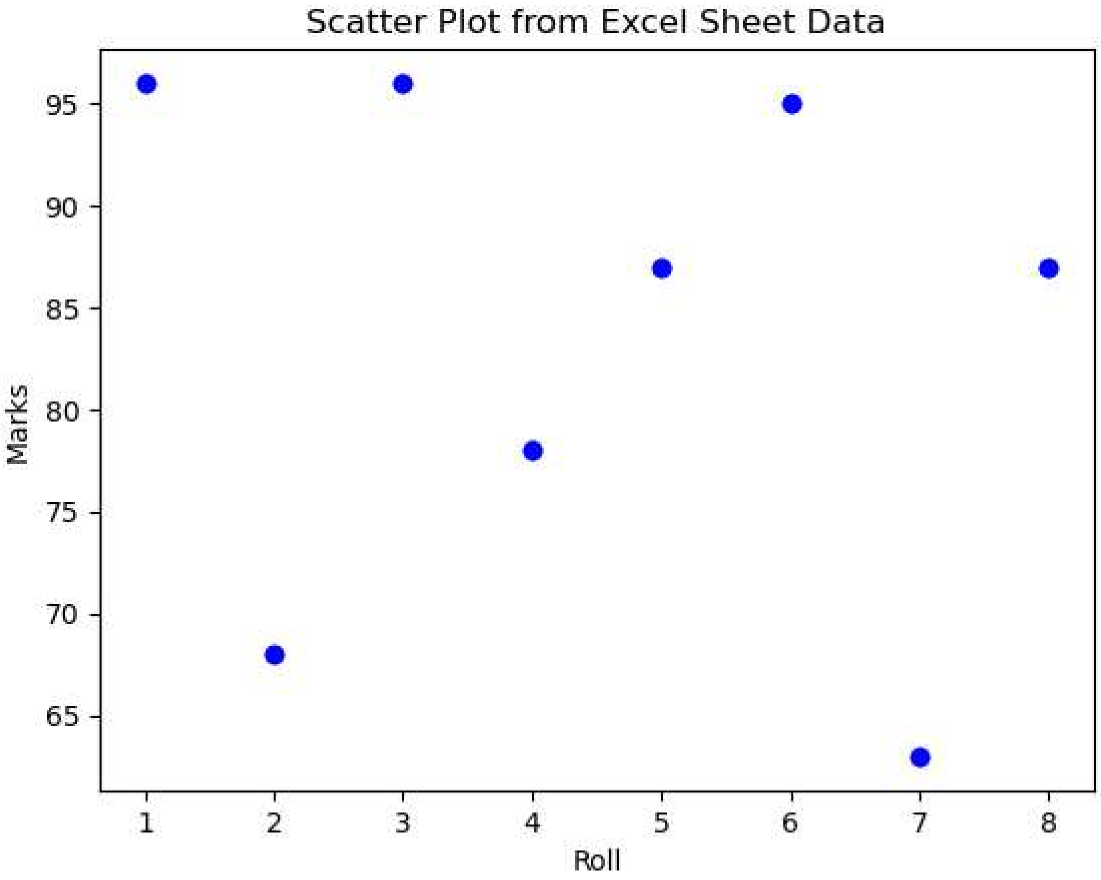


In [19]:

```
plt.scatter(data['Roll'], data['Marks'], color='b', marker='o')
plt.xlabel('Roll')
plt.ylabel('Marks')
plt.title('Scatter Plot from Excel Sheet Data')
```

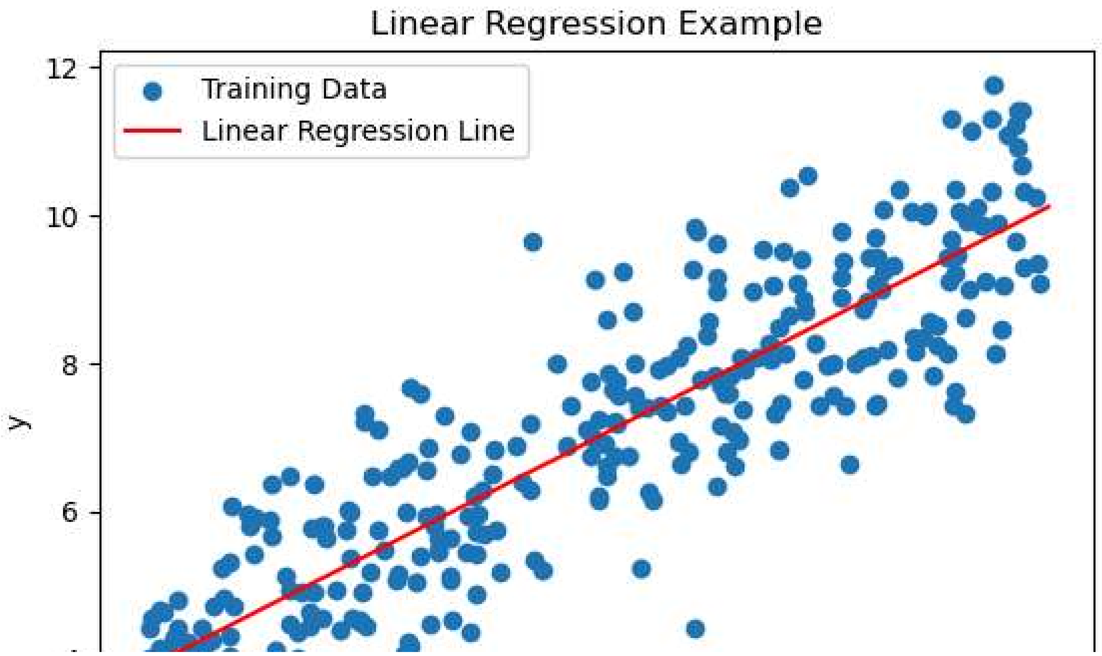
Out[19]:

```
Text(0.5, 1.0, 'Scatter Plot from Excel Sheet Data')
```



In [11]:

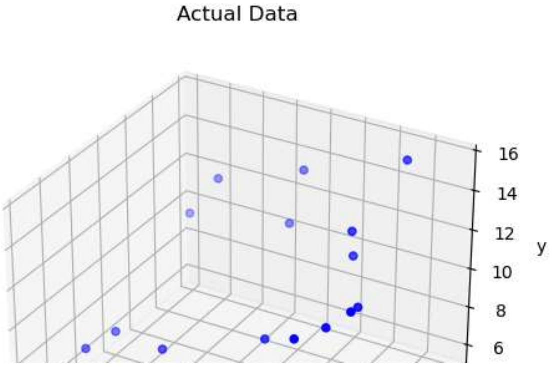
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Generating some sample data
np.random.seed(42)
X = 2 * np.random.rand(300, 1)
y = 4 + 3 * X + np.random.randn(300, 1)

# Creating a Linear Regression model
model = LinearRegression()

# Training the model
model.fit(X, y)

# Making predictions
X_new = np.array([[0], [2]])
y_pred = model.predict(X_new)

# Plotting the training data and the linear regression line
plt.scatter(X, y, label='Training Data')
plt.plot(X_new, y_pred, 'r-', label='Linear Regression Line')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.title('Linear Regression Example')
plt.show()
```

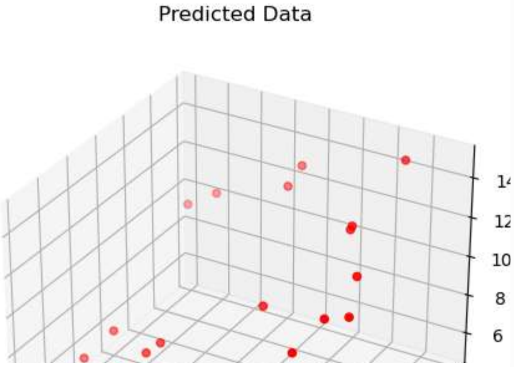


In [12]:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Generating some sample data with two independent variables
np.random.seed(42)
X1 = 2 * np.random.rand(100, 1)
X2 = 3 * np.random.rand(100, 1)
X = np.hstack([X1, X2])
y = 4 + 3 * X1 + 2 * X2 + np.random.randn(100, 1)

# Splitting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random

# Creating a Multiple Linear Regression model
model = LinearRegression()

# Training the model
model.fit(X_train, y_train)

# Making predictions on the test set
y_pred = model.predict(X_test)

# Evaluating the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')

# Visualizing the predictions
fig = plt.figure(figsize=(12, 6))

ax1 = fig.add_subplot(121, projection='3d')
ax1.scatter(X_test[:, 0], X_test[:, 1], y_test, color='blue', label='Actual Dat
ax1.set_xlabel('X1')
ax1.set_ylabel('X2')
ax1.set_zlabel('y')
ax1.set_title('Actual Data')

ax2 = fig.add_subplot(122, projection='3d')
ax2.scatter(X_test[:, 0], X_test[:, 1], y_pred, color='red', label='Predicted D
ax2.set_xlabel('X1')
ax2.set_ylabel('X2')
ax2.set_zlabel('y')
ax2.set_title('Predicted Data')

plt.show()
```

```
Mean Squared Error: 0.6664320988651887
```


Actual Data


Predicted Data

Program to create and handle frequency tables using python

In [2]:

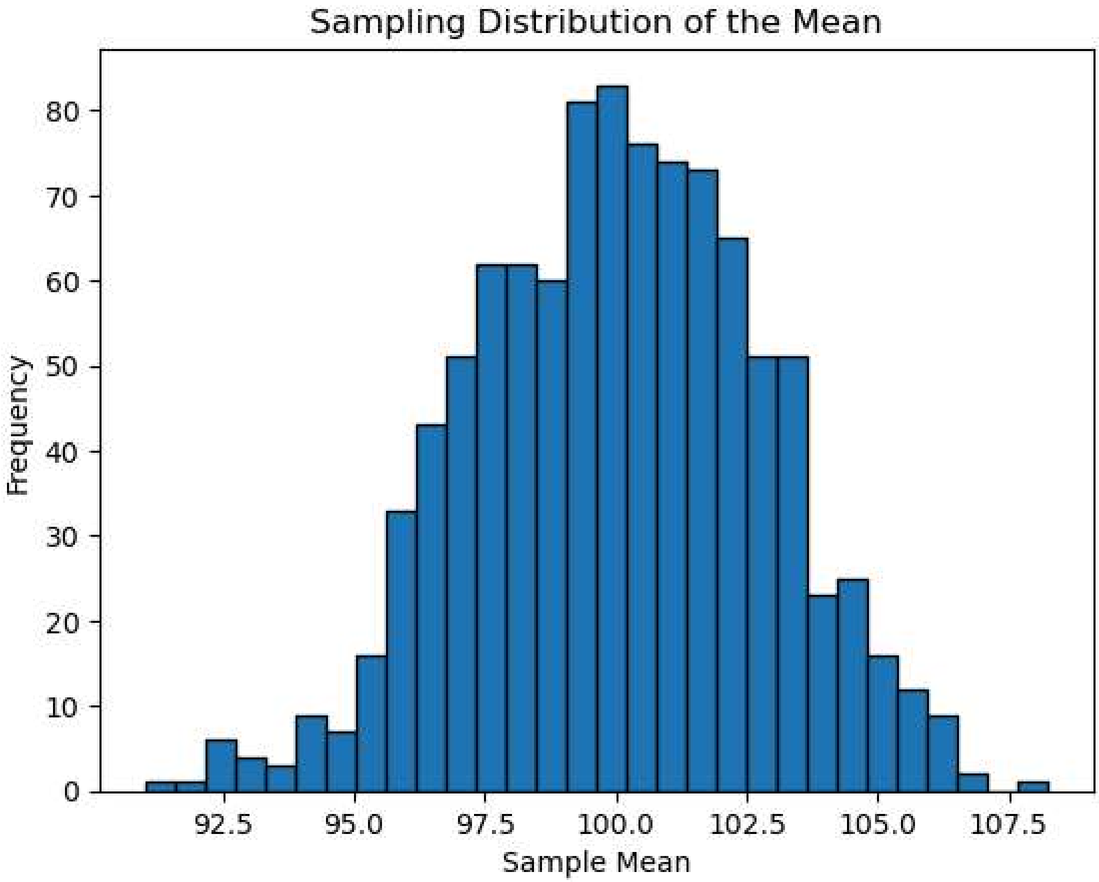
```python
from collections import Counter
import pandas as pd

def create_frequency_table(data):
    # Using collections.Counter
    frequency_table_counter = Counter(data)

    # Using Pandas DataFrame
    df = pd.DataFrame(data, columns=['Values'])
    frequency_table_pandas = df['Values'].value_counts().reset_index()
    frequency_table_pandas.columns = ['Value', 'Frequency']

    return frequency_table_counter, frequency_table_pandas

def print_frequency_table(frequency_table):
    for item, count in frequency_table.items():
        print(f"{item}: {count} times")
    print("\n")

    print("Pandas Frequency Table:")
    print(frequency_table_pandas)

if __name__ == "__main__":
    # Sample data
    data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5]

    # Create and print frequency tables
    frequency_table_counter, frequency_table_pandas = create_frequency_table(da

    print("Counter Frequency Table:")
    print_frequency_table(frequency_table_counter)
```

```
Counter Frequency Table:
1: 1 times
2: 2 times
3: 3 times
4: 4 times
5: 5 times


Pandas Frequency Table:
   Value  Frequency
0      5          5
1      4          4
2      3          3
3      2          2
4      1          1
```

Demonstrate sampling distribution concept using python.

In [1]:

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

# Set a random seed for reproducibility
np.random.seed(42)

# Generate a population with a normal distribution
population_mean = 100
population_std = 15
population_size = 10000
population = np.random.normal(population_mean, population_std, population_size)

# Number of samples to take
num_samples = 1000
sample_size = 30

# Initialize an array to store sample means
sample_means = np.zeros(num_samples)

# Take multiple samples and calculate means
for i in range(num_samples):
    sample = np.random.choice(population, size=sample_size)
    sample_means[i] = np.mean(sample)

# Plot the histogram of sample means
plt.hist(sample_means, bins=30, edgecolor='black')
plt.title('Sampling Distribution of the Mean')
plt.xlabel('Sample Mean')
plt.ylabel('Frequency')
plt.show()
```


Sampling Distribution of the Mean
Frequency
0
10
20
30
40
50
60
70
80
92.5
95.0
97.5
100.0
102.5
105.0
107.5
Sample Mean