MACHINE LEARNING CONCEPTS

CODE 1: (numpy1.py)

CONCEPT USED: NumPy array creation and initialization

Observation: Creates 1D arrays, 2D matrices, zero/ones matrices, range arrays, and linearly spaced arrays with different patterns.

Logic of the code: Uses np.array(), np.zeros(), np.ones(), np.arange(), and np.linspace() functions for various array initialization methods.

Output screenshots:

```
1D Array: [1 2 3 4 5]
2D Array:
 [[1 2 3]
 [4 5 6]]
3x3 Zero Matrix:
 [[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
2x4 Ones Matrix:
 [[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
Range Array: [0 2 4 6 8]
Linearly spaced values: [0.   0.25 0.5  0.75 1.  ]
```

CODE 2: (numpy2.py)

Concept used: NumPy element-wise arithmetic operations

Observation: Performs mathematical operations (addition, subtraction, multiplication, division, square root, exponentiation) between arrays element by element.

Logic of the code: Applies arithmetic operators (+, -, *, /) directly to arrays and uses np.sqrt() and np.power() for mathematical functions.

Output screenshots:

```
Addition: [11 22 33 44]
Subtraction: [ 9 18 27 36]
Multiplication: [ 10  40  90 160]
Division: [10. 10. 10. 10.]
Square root of a: [3.16227766 4.47213595 5.47722558 6.32455532]
a squared: [ 100  400  900 1600]
```

CODE 3: (numpy3.py)

Concept used: NumPy array indexing and slicing

Observation: Access individual elements, slice arrays for subsets, modify elements, and work with 1D/2D array manipulation.

Logic of the code: Uses square brackets for indexing, colon notation for slicing, negative indexing for reverse access, and comma-separated indices for 2D arrays.

Output screenshots:

```
First element: 10
Last element: 60
First 3 elements: [10 20 30]
Every second element: [10 30 50]
Modified array: [10 20 99 40 50 60]
Element at (1,2): 6
First row: [1 2 3]
Second column: [2 5 8]
```

CODE 4: (numpy4.py)

Concept used: NumPy statistical functions

Observation: Calculates max, min, sum, mean, standard deviation, and finds indices of extreme values in arrays.

Logic of the code: Uses built-in functions np.max(), np.min(), np.sum(), np.mean(), np.std(), np.argmax(), and np.argmin() for statistical analysis.

Output screenshots:

```
Max: 9
Min: 2
Sum: 26
Mean: 5.2
Standard Deviation: 2.5612496949731396
Index of Max Value: 3
Index of Min Value: 2
```

CODE 5: (numpy5.py)

Concept used: NumPy random number generation

Observation: Generates random integers, floats, normal distribution matrices, and shuffles arrays.

Logic of the code: Uses np.random.randint(), np.random.rand(), np.random.randn(), and np.random.shuffle() for randomization.

Output screenshots:

```
Random Integers: [4 8 9 4 4]
Random Floats: [0.1522764  0.01860702 0.14331616 0.56335402 0.78844433]
Random Normal Distribution Matrix:
 [[ 0.89413519 -0.68343405 -0.12073987]
 [ 1.19591632 -1.2151296  -0.63111423]
 [ 1.4601782   0.23490927  0.17227189]]
Shuffled Array: [4 1 5 3 2]
```

CODE 6: (numpy6.py)

Concept used: Train-test data splitting

Observation: Splits dataset into training (80%) and testing (20%) portions for machine learning.

Logic of the code: Uses train_test_split() from sklearn with test_size=0.2 and random_state=42 for reproducible splits.

Output screenshots:

```
Training Data:
 [[1]
 [8]
 [3]
 [5]
 [4]
 [7]] [ 2 16  6 10  8 14]
Testing Data:
 [[2]
 [6]] [ 4 12]
```

CODE 7: (numpy7.py)

Concept used: Linear regression

Observation: Creates linear model, trains on y=2x data, makes predictions, and shows learned parameters (slope, intercept).

Logic of the code: Uses LinearRegression() with fit() for training and predict() for making predictions on new data.

Output screenshots:

```
Prediction for 6: [12.]
Slope (Coefficient): [2.]
Intercept: 0.0
```

CODE 8: (numpy8.py)

Concept used: Logistic regression for binary classification

Observation: Predicts pass/fail based on study hours, shows both class predictions and probability estimates.

Logic of the code: Uses LogisticRegression() with predict() for classifications and predict_proba() for probability estimates.

Output screenshots:

```
Prediction for 2.5 hours: [0]
Prediction for 6 hours: [1]
Probabilities for 2.5 hours: [[0.75496813 0.24503187]]
```

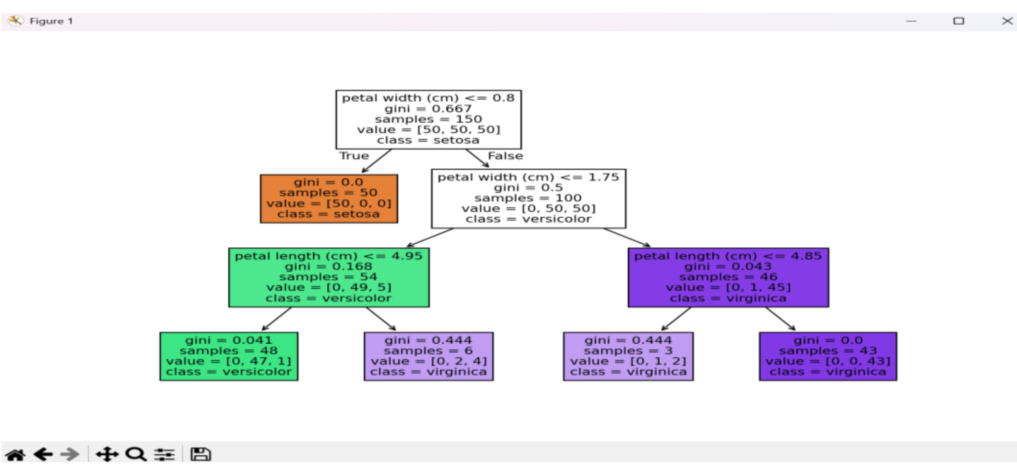
CODE 9: (numpy9.py)

Concept used: Decision tree classification with visualization

Observation: Trains decision tree on Iris dataset, makes predictions, and visualizes the complete tree structure.

Logic of the code: Uses DecisionTreeClassifier() with max_depth=3 and tree.plot_tree() for visualization.

Output screenshots:



CODE 10: (numpy10.py)

Concept used: Data standardization

Observation: Converts data features to have zero mean and unit variance for preprocessing.

Logic of the code: Uses StandardScaler() with fit_transform() to normalize features: (value - mean) / std.

Output screenshots:

```
Original Data:
 [[ 10 100]
 [ 20 200]
 [ 30 300]
 [ 40 400]]
Standardized Data:
 [[-1.34164079 -1.34164079]
 [-0.4472136  -0.4472136 ]
 [ 0.4472136   0.4472136 ]
 [ 1.34164079  1.34164079]]
```

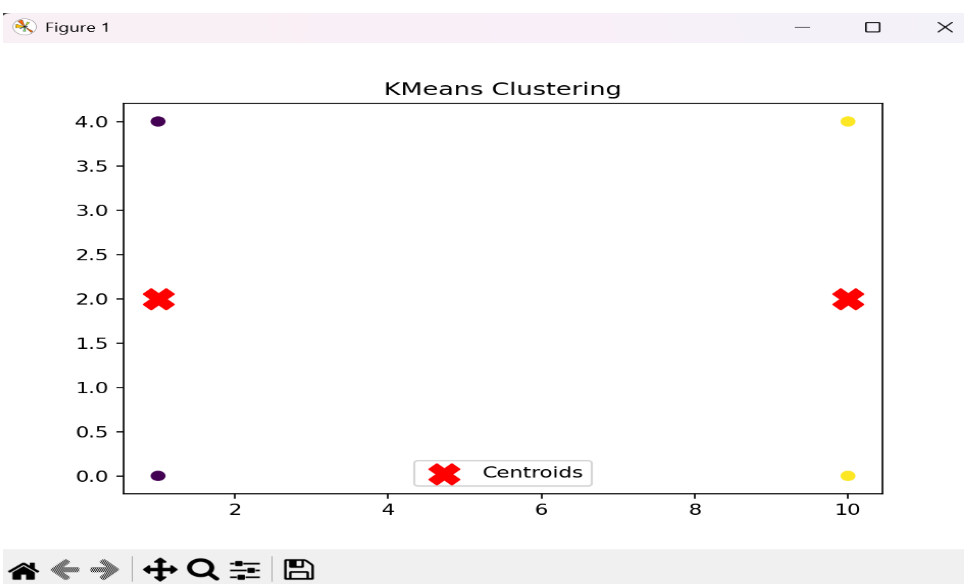
CODE 11: (numpy11.py)

Concept used: K-Means clustering

Observation: Groups 2D points into 2 clusters, identifies centroids, and visualizes clustering results with colors.

Logic of the code: Uses KMeans(n_clusters=2) algorithm and scatter plots with different colors for visualization.

Output screenshots:



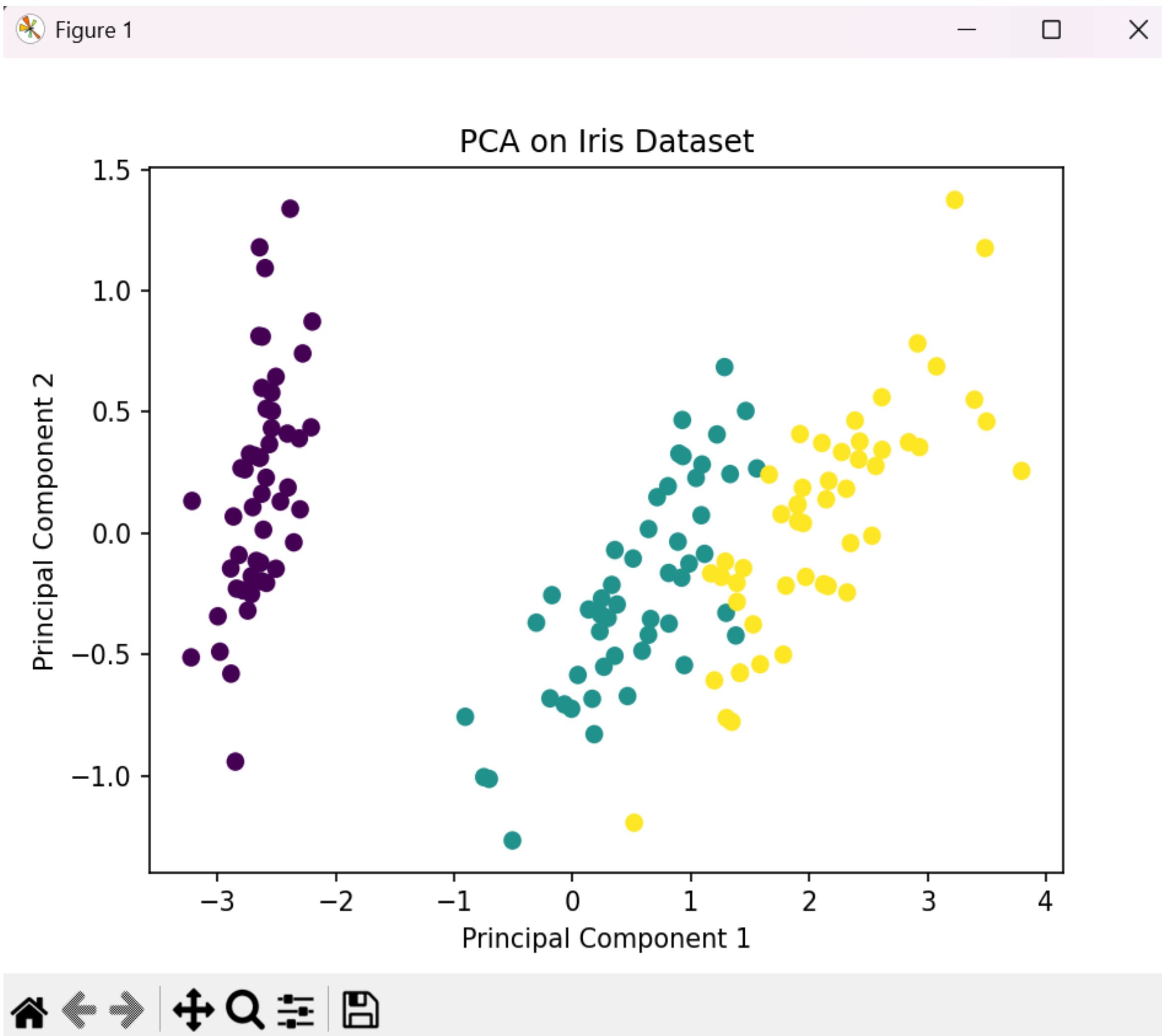
CODE 12: (numpy12.py)

Concept used: Principal Component Analysis (PCA)

Observation: Reduces Iris dataset from 4D to 2D while preserving variance, enables 2D visualization.

Logic of the code: Uses PCA(n_components=2) to project data onto top 2 principal components.

Output screenshots:



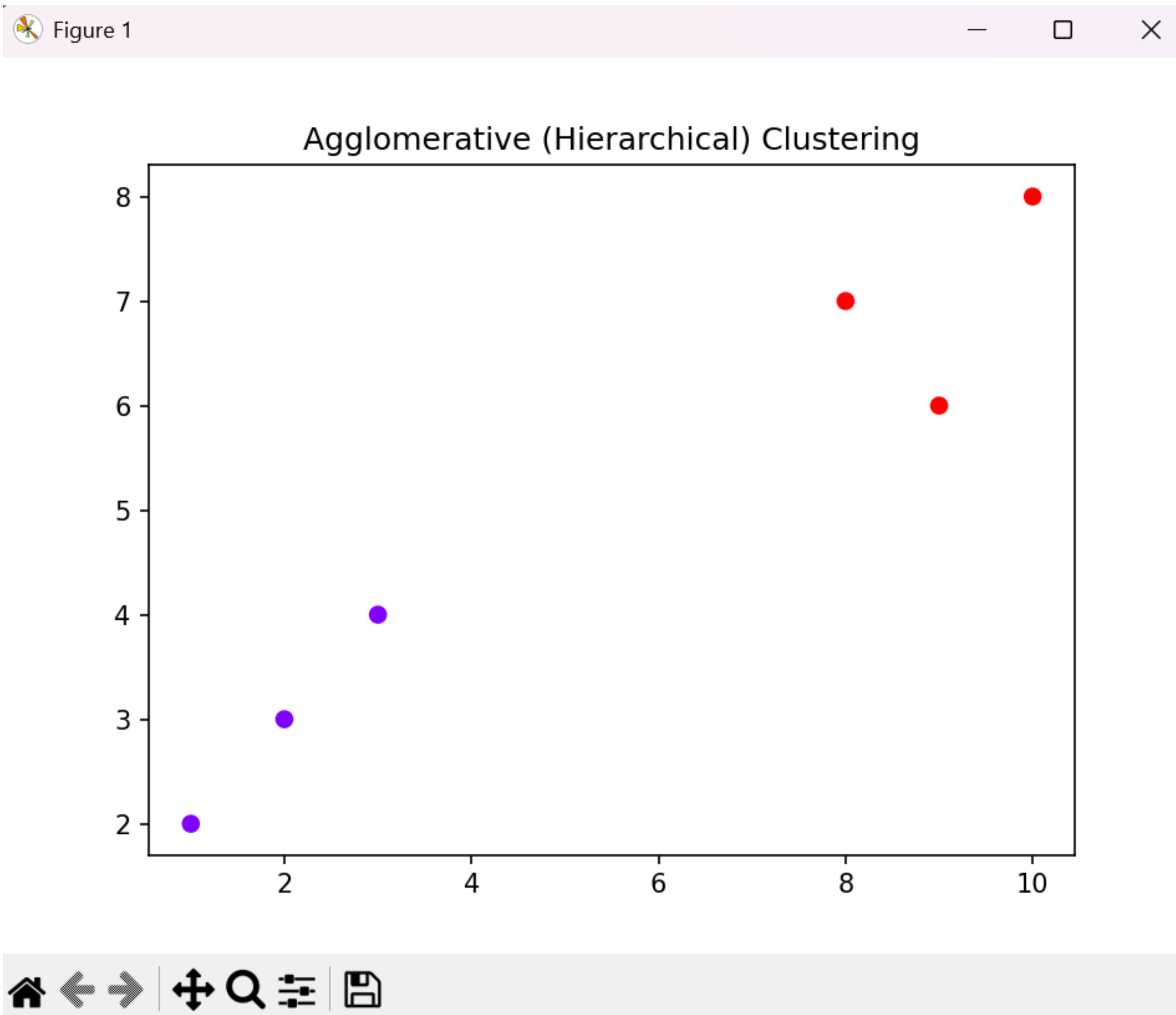
CODE 13: (numpy13.py)

Concept used: Hierarchical clustering

Observation: Performs bottom-up clustering by merging closest points/clusters, visualizes final groupings.

Logic of the code: Uses AgglomerativeClustering(n_clusters=2) to build cluster hierarchy from individual points.

Output screenshots:



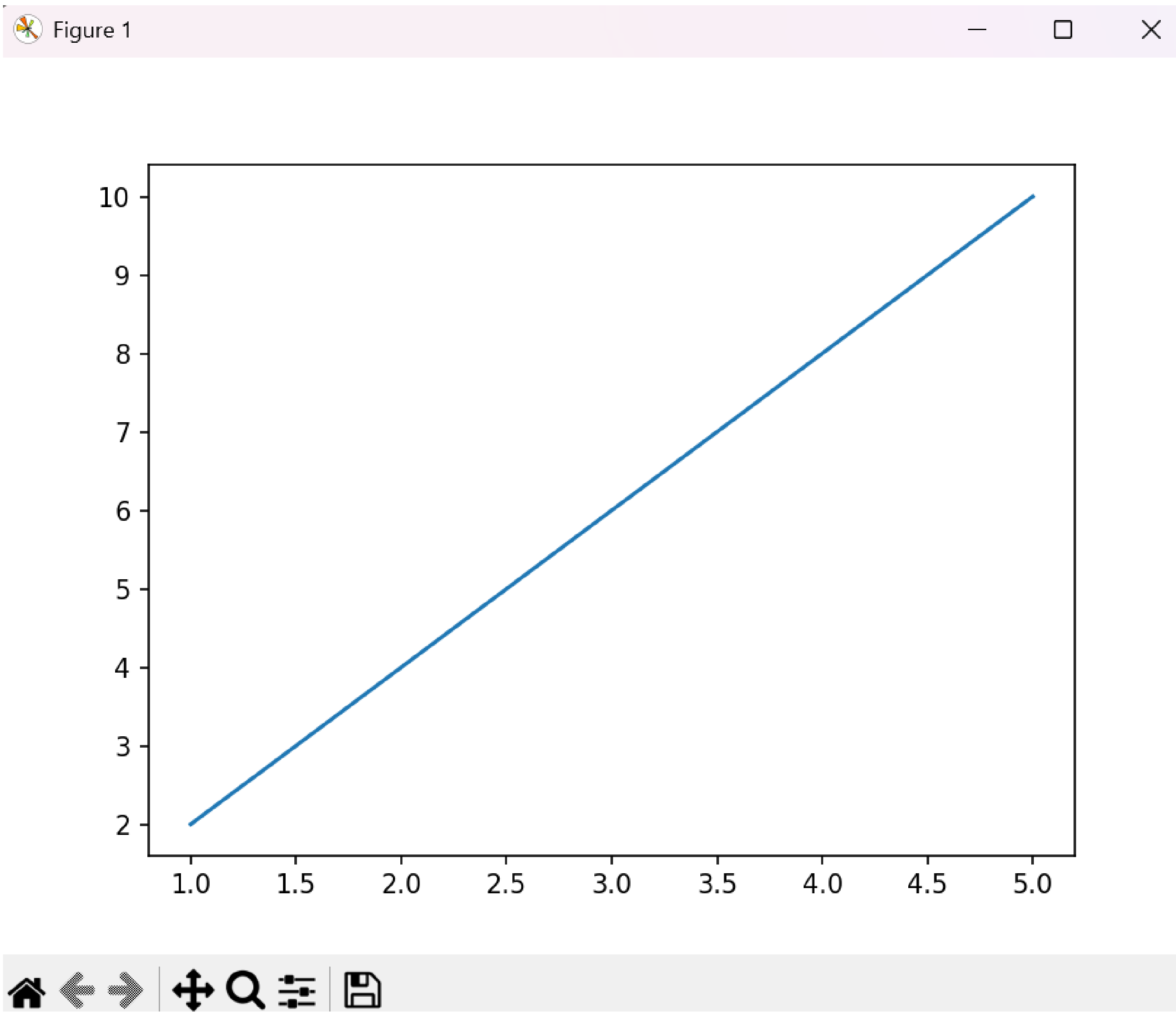
CODE 14: (plot1.py)

Concept used: Basic line plotting

Observation: Creates simple line plot connecting data points showing linear relationship.

Logic of the code: Uses plt.plot() to connect x,y coordinates and plt.show() to display the plot.

Output screenshots:



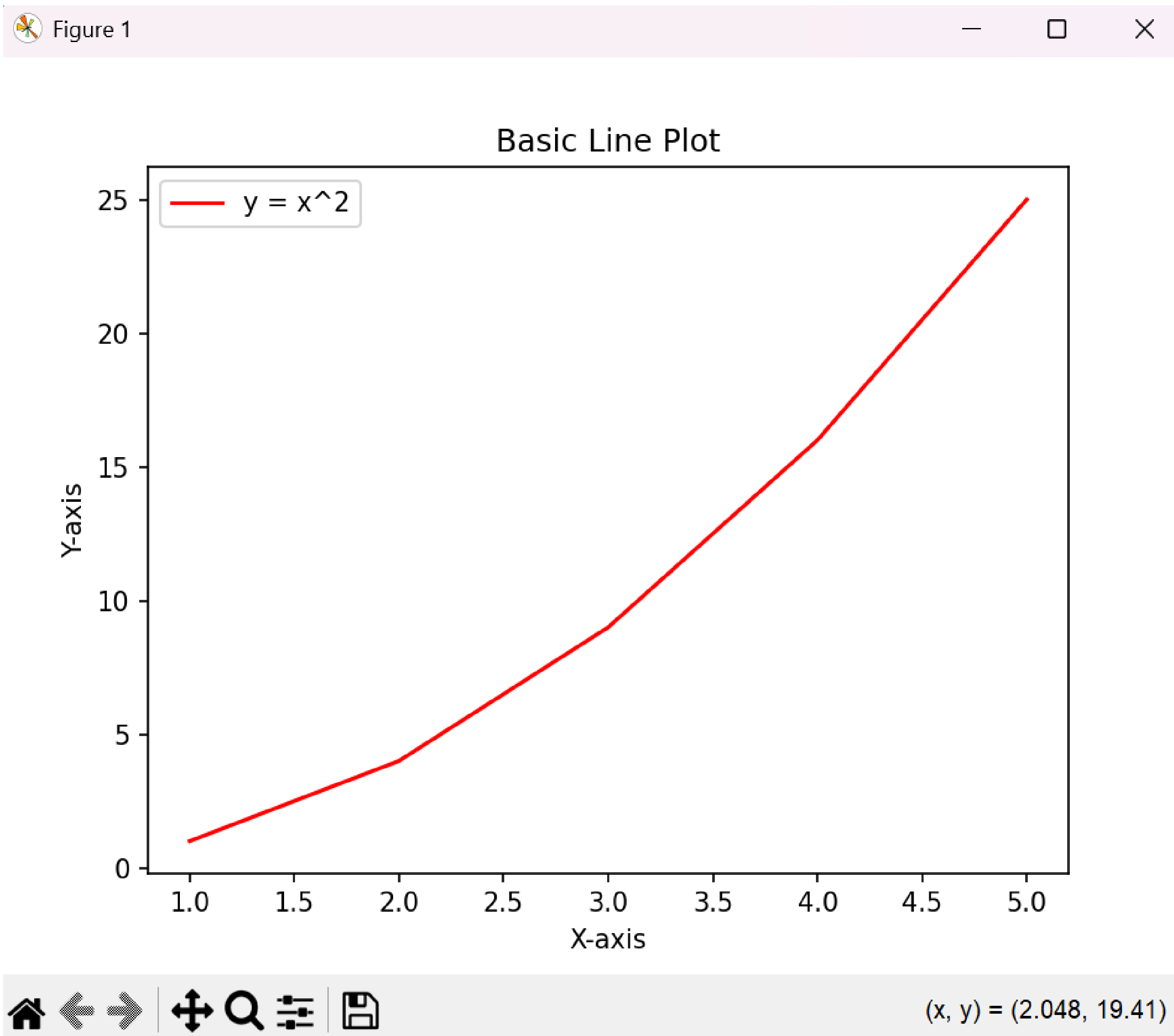
CODE 15: (plot2.py)

Concept used: Enhanced line plotting with formatting

Observation: Creates formatted plot showing quadratic relationship with labels, colors, title, and legend.

Logic of the code: Uses plt.plot() with customization parameters and adds xlabel(), ylabel(), title(), legend() for formatting.

Output screenshots:



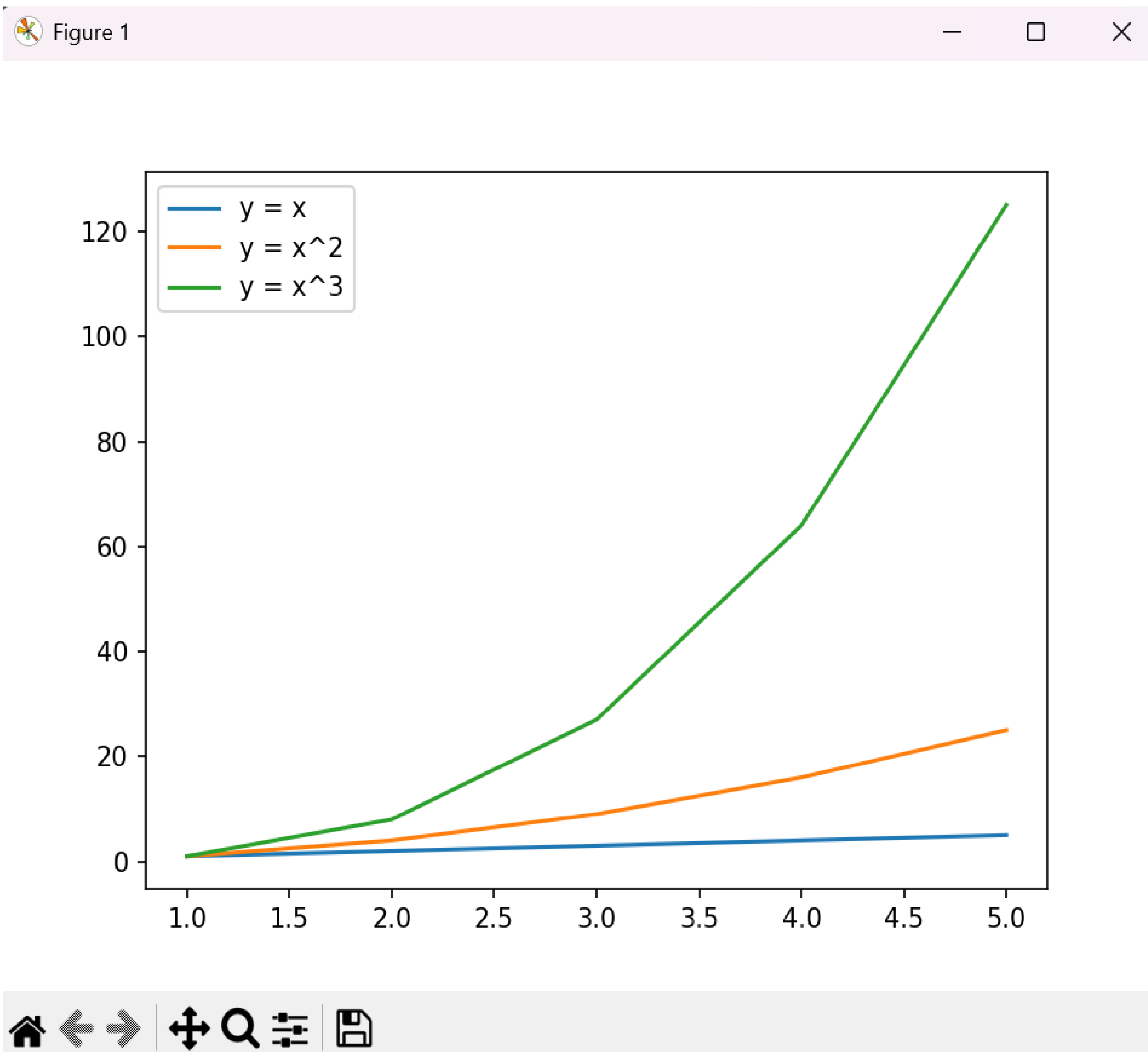
CODE 16: (plot3.py)

Concept used: Multiple line plots

Observation: Displays three mathematical functions (linear, quadratic, cubic) on same plot with different colors and legend.

Logic of the code: Multiple plt.plot() calls with list comprehensions to generate different function values.

Output screenshots:



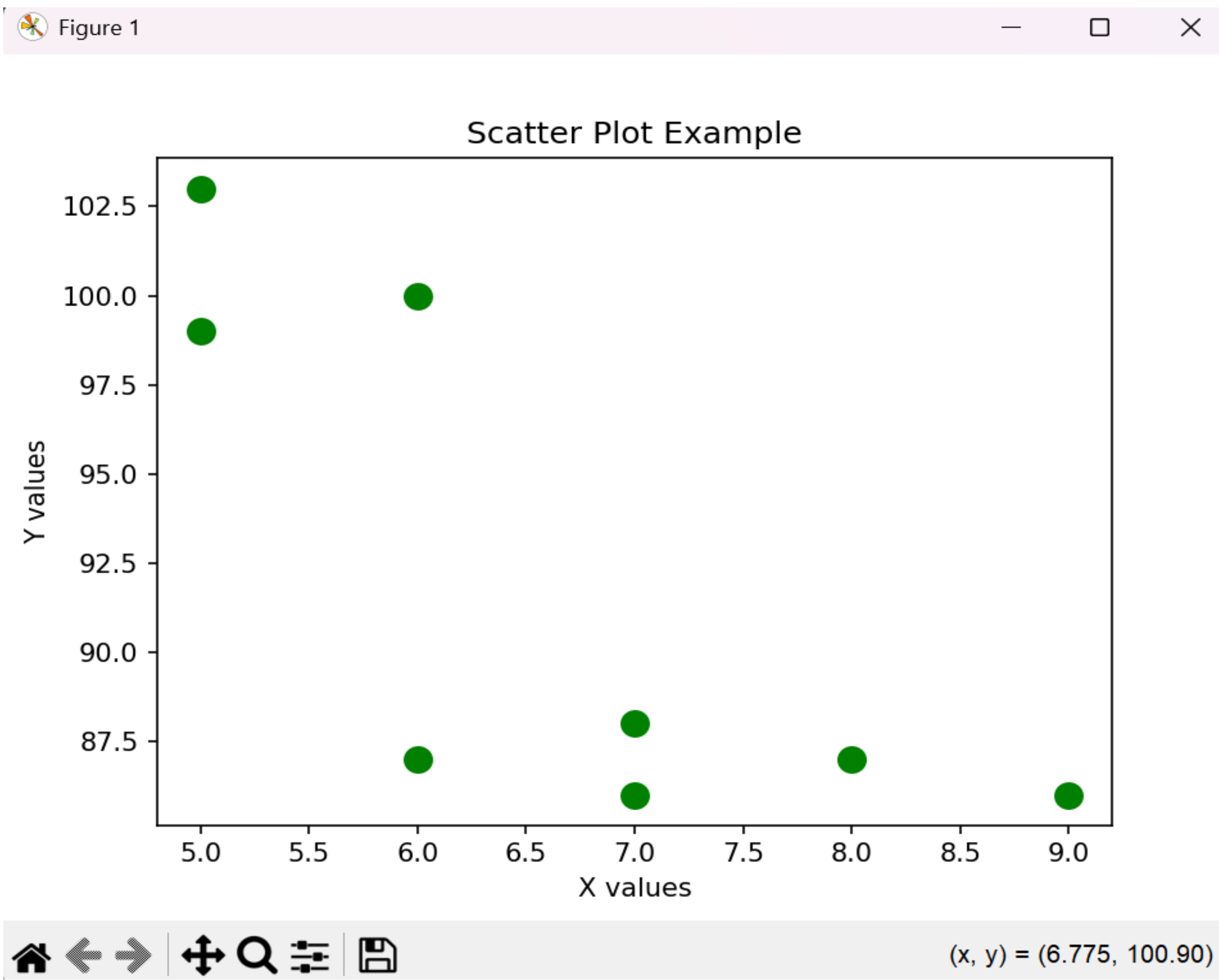
CODE 17: (plot4.py)

Concept used: Scatter plot visualization

Observation: Shows individual data points as separate markers for visualizing relationships and patterns.

Logic of the code: Uses plt.scatter() with parameters for color, marker, and s (size) customization.

Output screenshots:



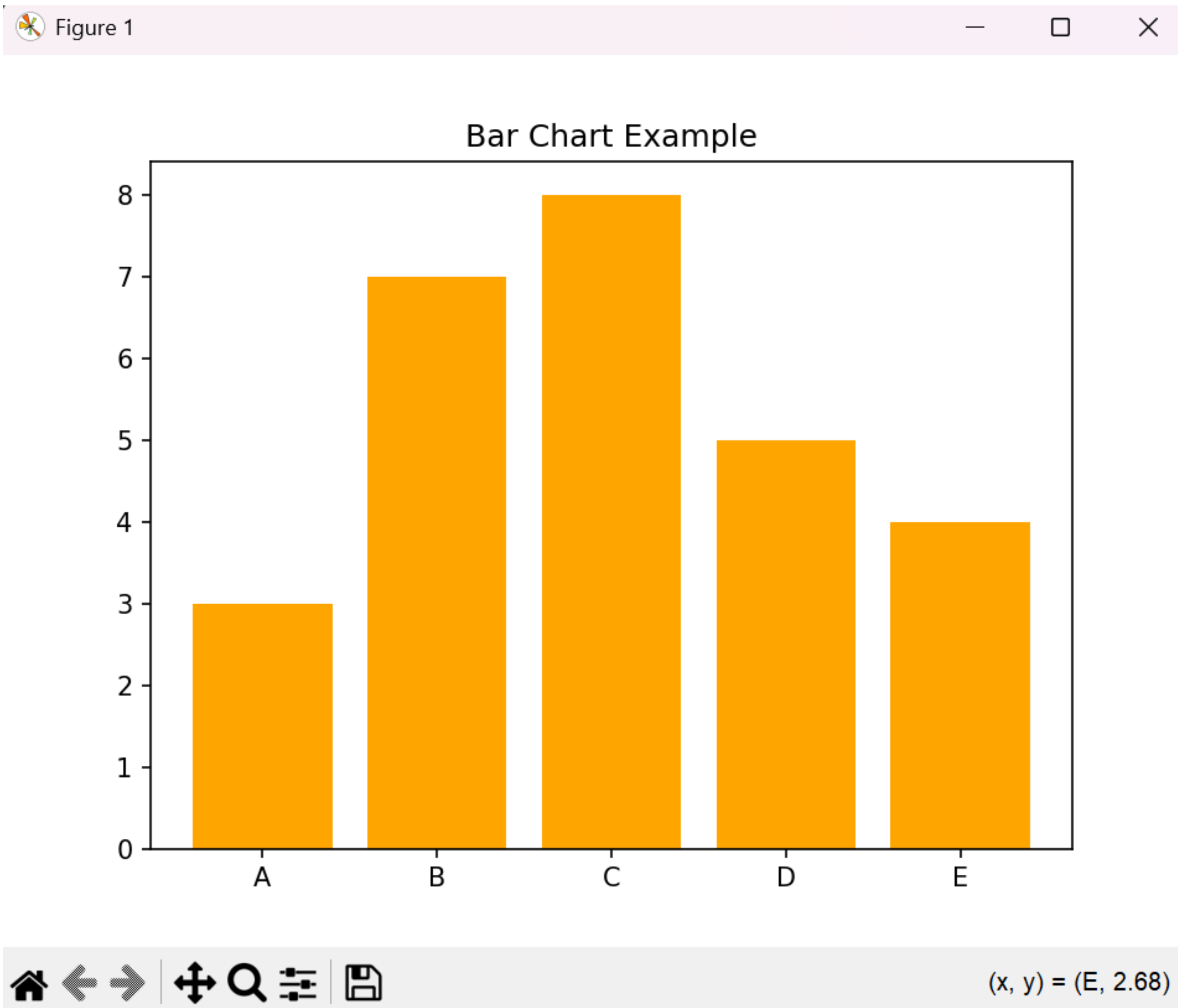
CODE 18: (plot5.py)

Concept used: Bar chart visualization

Observation: Creates vertical bars for categorical data comparison with each category having corresponding value heights.

Logic of the code: Uses plt.bar() with category labels as x-axis and values as y-axis heights.

Output screenshots:



--

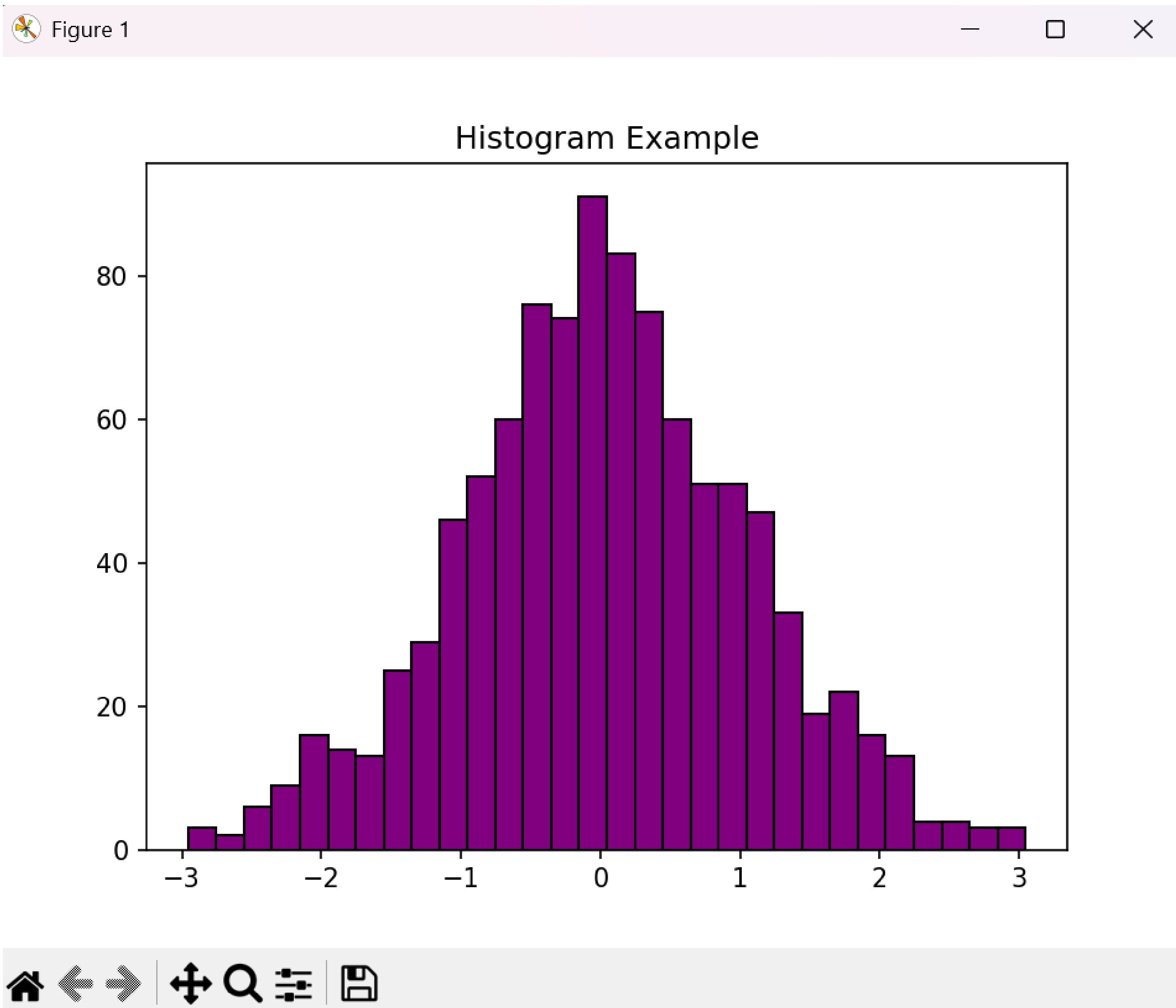
CODE 19: (plot6.py)

Concept used: Histogram visualization

Observation: Creates frequency distribution of 1000 random numbers using 30 bins with purple bars and black edges.

Logic of the code: Uses np.random.randn() for normal distribution data and plt.hist() with bins, color, and edgecolor parameters.

Output screenshots:



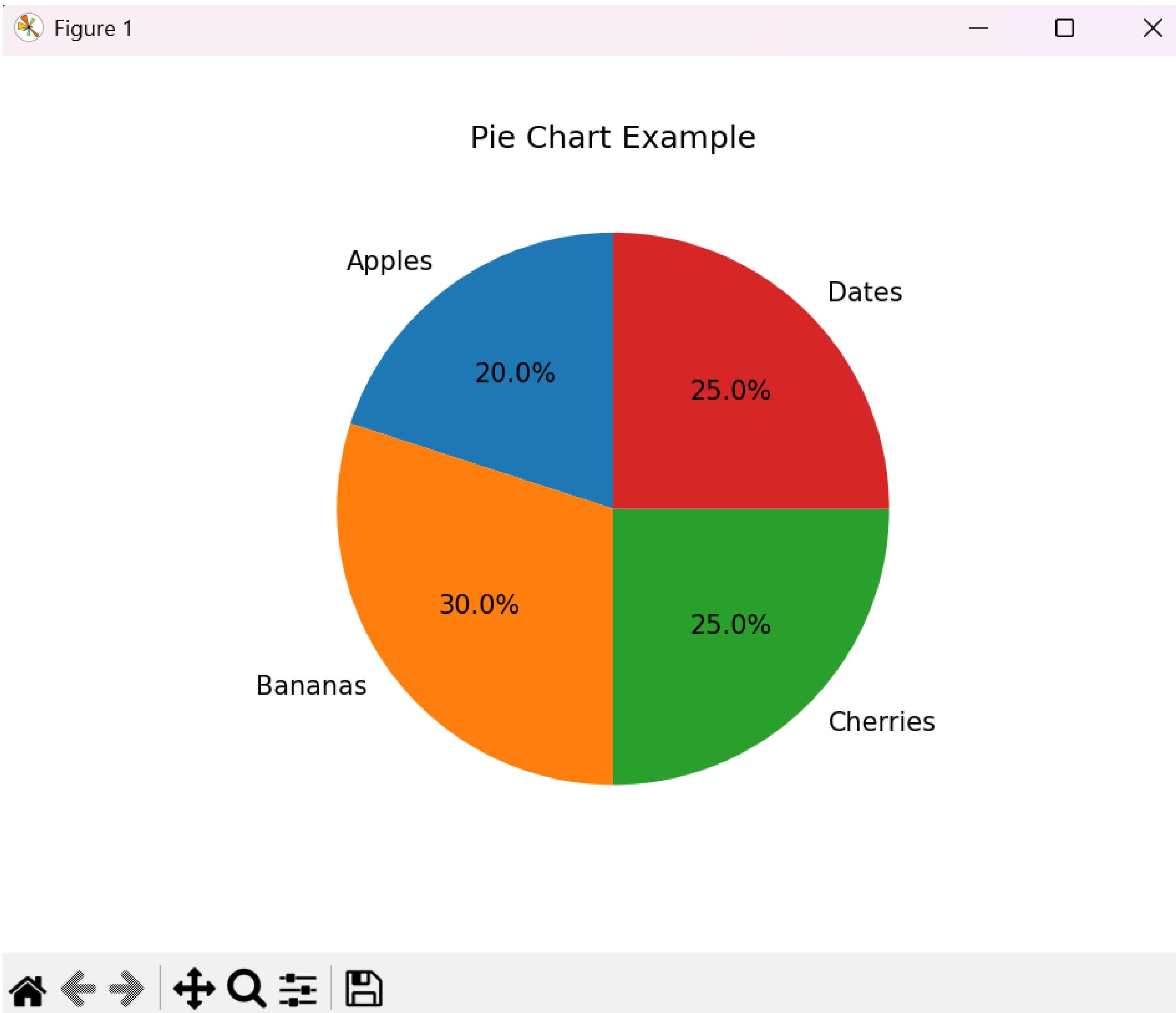
CODE 20: (plot7.py)

Concept used: Pie chart visualization

Observation: Creates circular chart showing percentage distribution of fruit categories with automatic percentage labels.

Logic of the code: Uses plt.pie() with sizes, labels, autopct for percentages, and startangle for rotation.

Output screenshots:



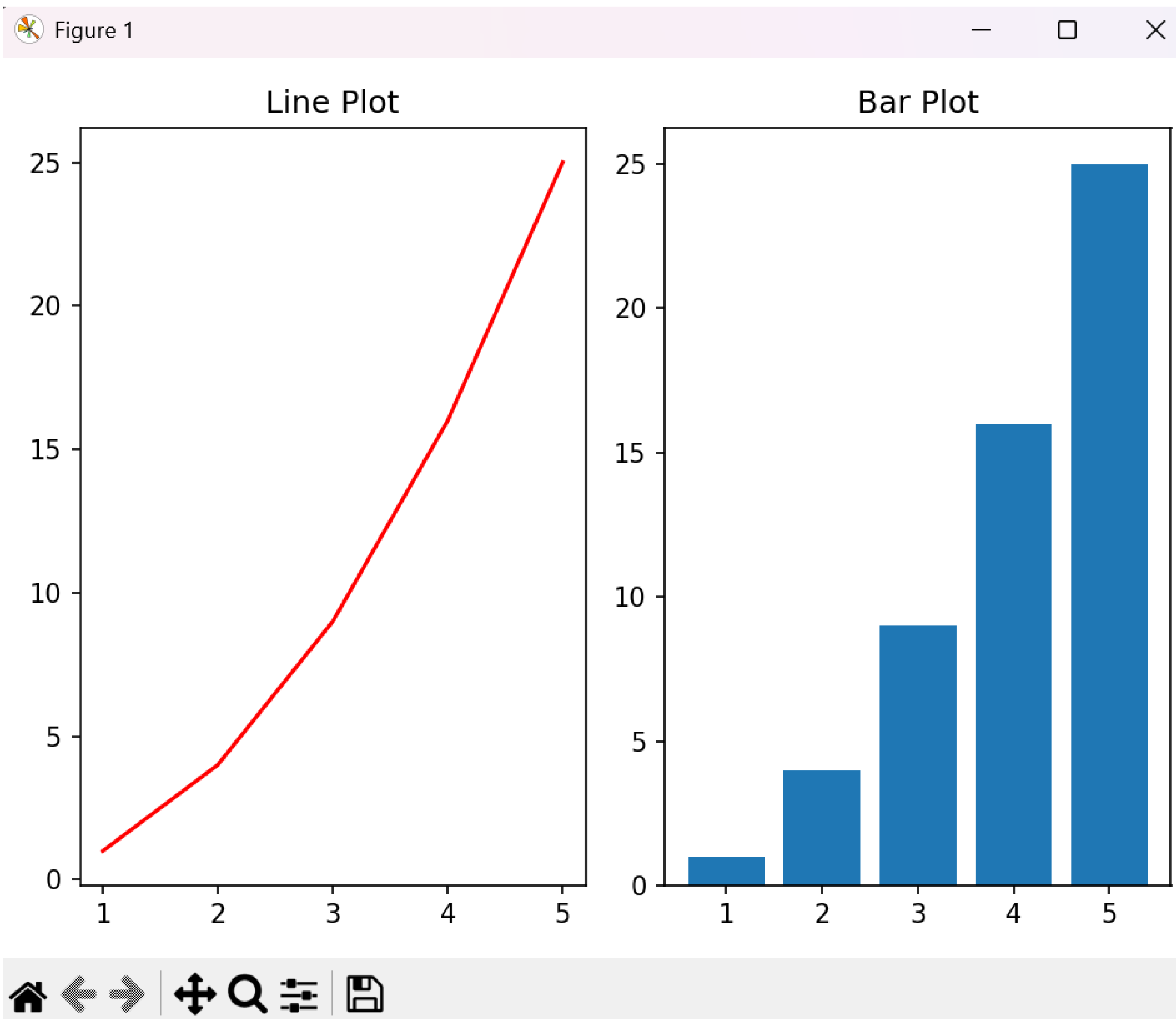
CODE 21: (plot8.py)

Concept used: Subplot visualization

Observation: Creates two side-by-side plots (line and bar) of the same data in a single figure for comparison.

Logic of the code: Uses plt.subplot(1, 2, 1) and plt.subplot(1, 2, 2) for layout, with plt.tight_layout() for spacing.

Output screenshots:



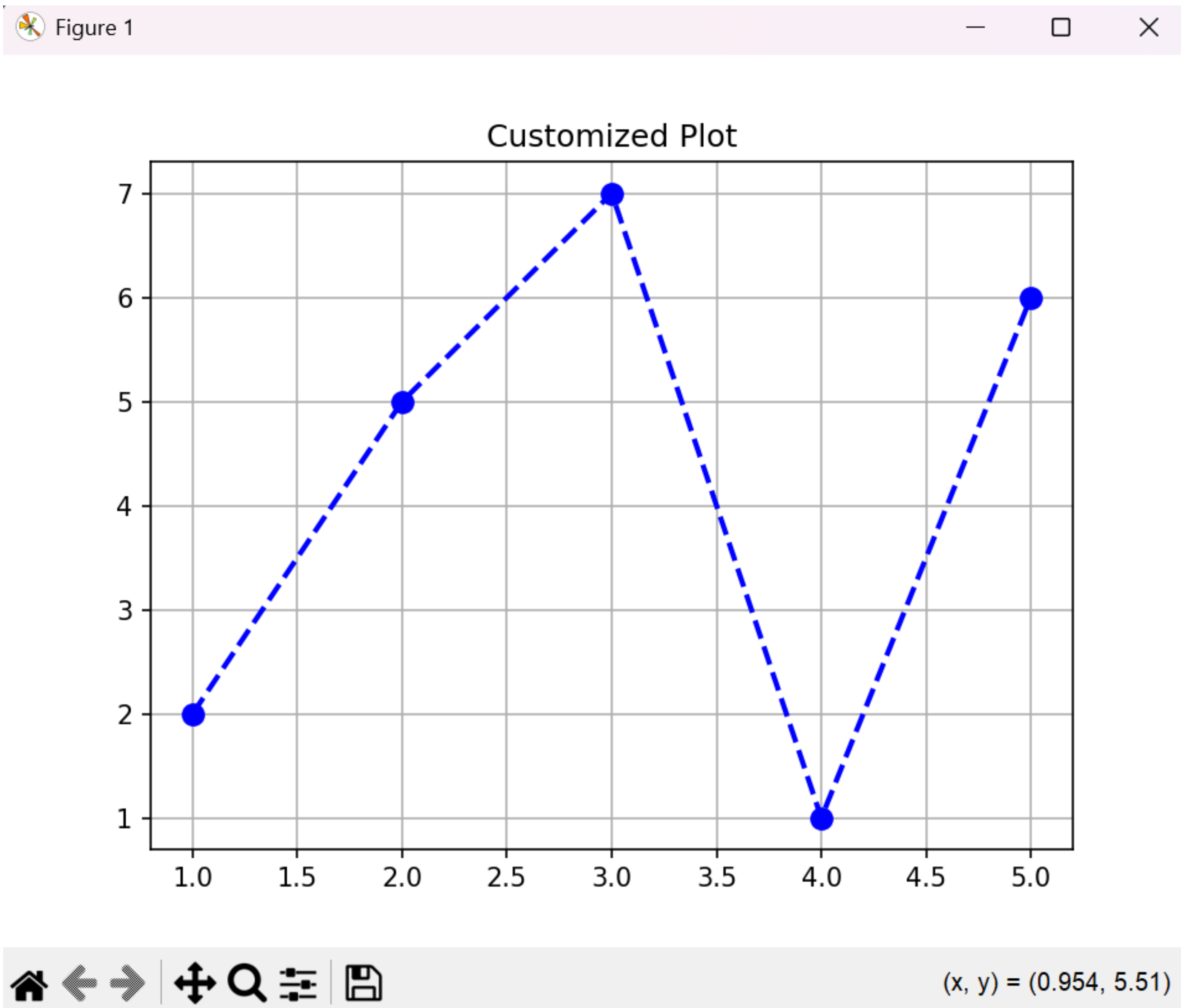
CODE 22: (plot9.py)

Concept used: Customized line plot styling

Observation: Creates styled line plot with custom color, dashed lines, circular markers, grid, and specified line/marker sizes.

Logic of the code: Uses plt.plot() with color, linestyle, marker, linewidth, markersize parameters and plt.grid(True).

Output screenshots:



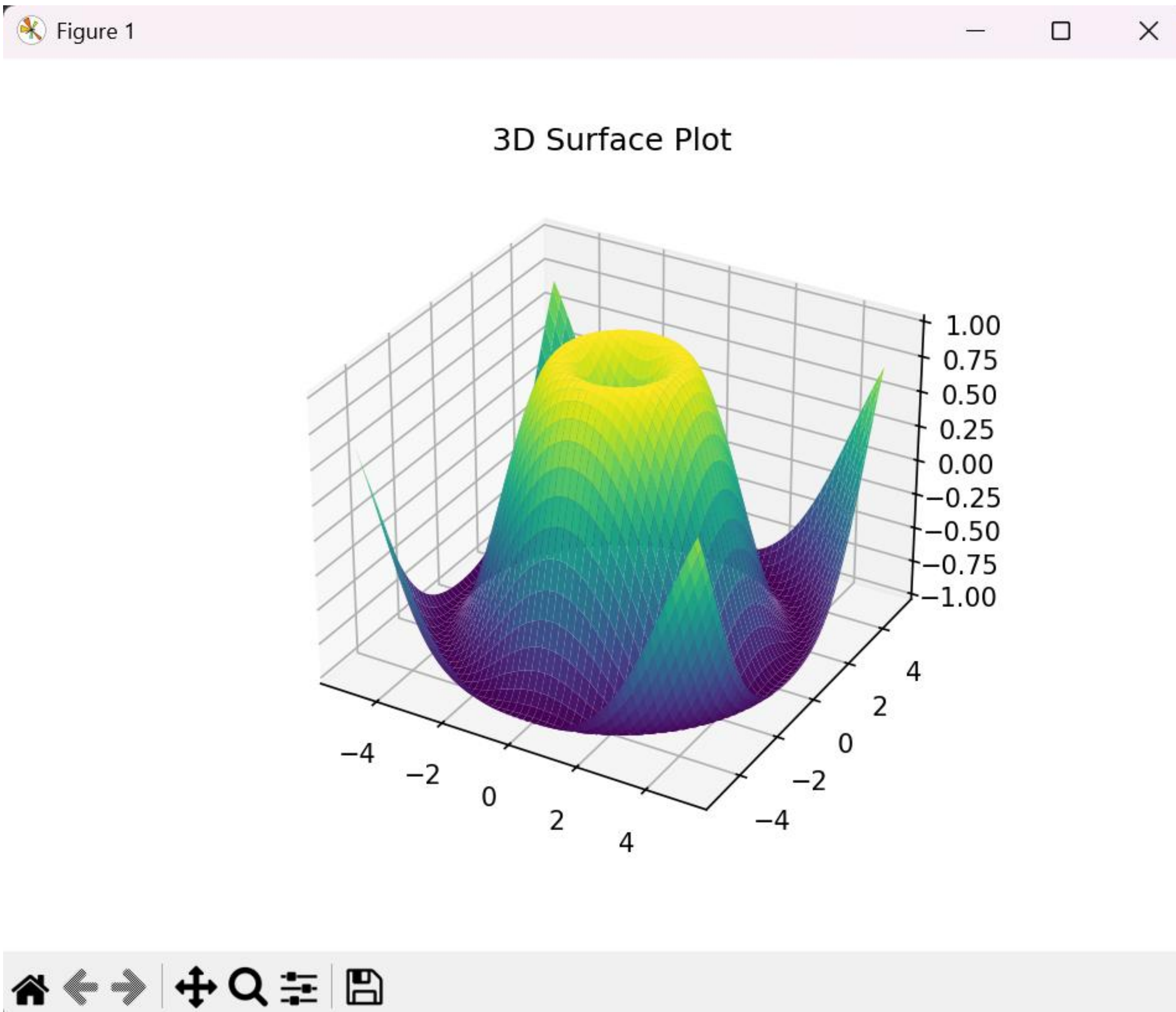
CODE 23: (plot10.py)

Concept used: 3D surface plot visualization

Observation: Creates 3D surface plot of mathematical function $Z = \sin(\sqrt{X^2 + Y^2})$ with color mapping.

Logic of the code: Uses meshgrid() for 3D coordinates, fig.add_subplot(111, projection='3d') for 3D axes, and plot_surface() with viridis colormap.

Output screenshots:



CODE 24: (scikitlearn_linear_regression.py)

Concept used: Complete linear regression workflow with visualization

Observation: Generates synthetic data, trains linear model, displays learned parameters, makes predictions, and plots training data with regression line.

Logic of the code: Creates noisy linear data, uses LinearRegression() for training, extracts coefficients with model.coef_ and model.intercept_, and visualizes with scatter plot and prediction line.

Output screenshots: