Assignment -2

Executive Post Graduate Certification in Electric Vehicle Design

Submitted by:

Bolli Sahithya

Problem Statement:

Advanced functions and data visualization capabilities of MATLAB are essential for analyzing and interpreting complex data. This assignment will help you understand how to create and use functions, work with scripts, and visualize data effectively.

Objective:

To develop skills in writing MATLAB functions and scripts.
To explore advanced data visualization techniques.

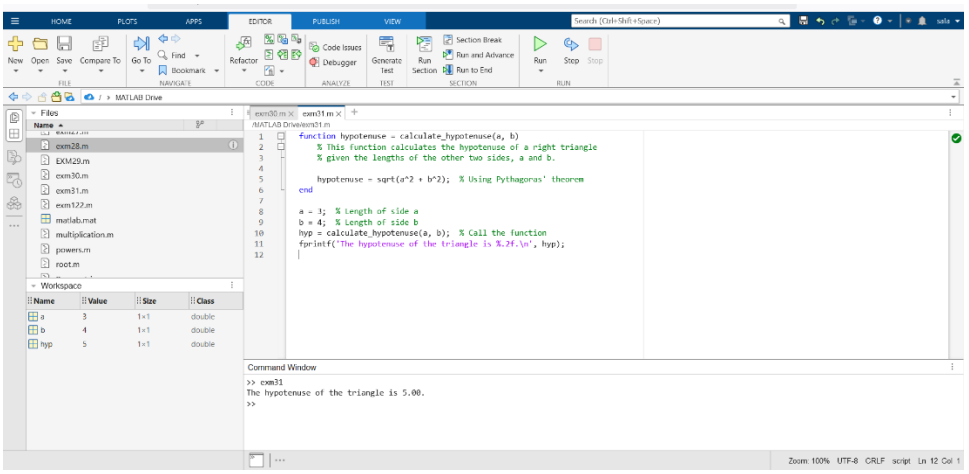
Tasks to be Performed:

1.Creating and Using Functions:

- Write a simple function that takes input arguments and returns outputs.
- Use built-in functions and create custom functions
- Understand the scope of variables within functions

° Write a simple function that takes input arguments and returns outputs:

```
function hypotenuse = calculate_hypotenuse (a, b)

    % This function calculates the hypotenuse of a right triangle
    % given the lengths of the other two sides, a and b

    hypotenuse = sqrt (a^2 + b^2);  % Using Pythagoras' theorem
end

a = 3;  % Length of side a
b = 4;  % Length of side b
hyp = calculate_hypotenuse (a, b);  % Call the function
fprintf ('The hypotenuse of the triangle is %.2f.\n', hyp);
```

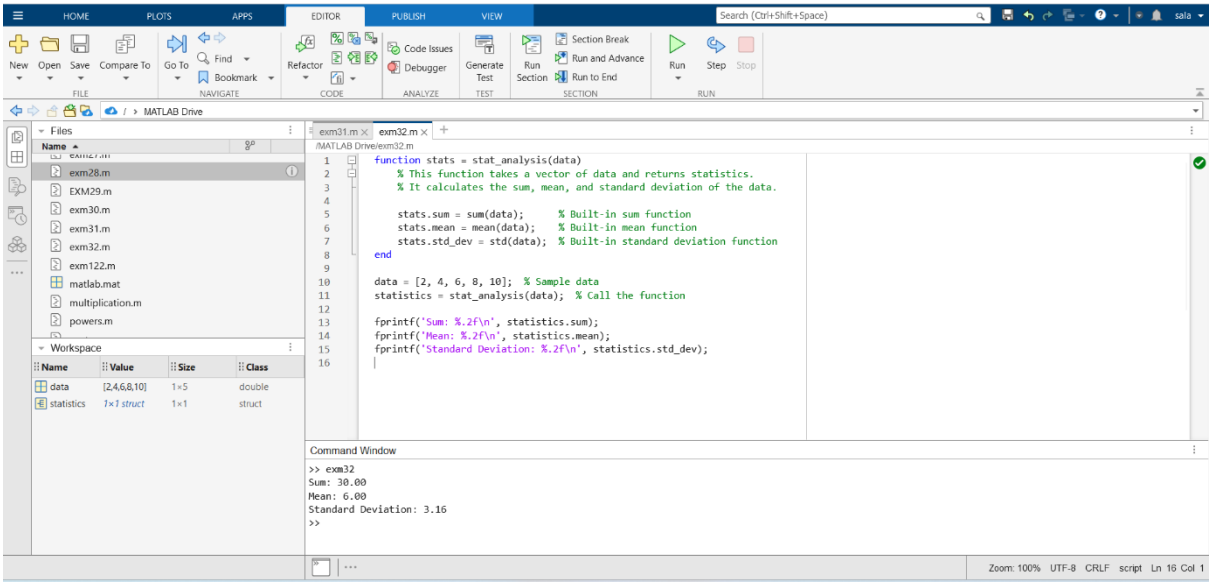


Explanation:

- The function calculate_hypotenuse takes two inputs (a and b), which represent the lengths of the two legs of the triangle.
- It calculates the hypotenuse using the formula sqrt (a^2 + b^2) and returns the result.

° Use built-in functions and create custom functions:

```
function stats = stat_analysis(data)
    % This function takes a vector of data and returns statistics
    % It calculates the sum, mean, and standard deviation of the data

    stats.sum = sum(data);   % Built-in sum function
    stats.mean = mean(data); % Built-in mean function
    stats.std_dev = std(data); % Built-in standard deviation function
end

data = [2, 4, 6, 8, 10];  % Sample data
statistics = stat_analysis(data);  % Call the function

fprintf ('Sum: %.2f\n', statistics.sum);
fprintf ('Mean: %.2f\n', statistics.mean);
fprintf ('Standard Deviation: %.2f\n', statistics.std_dev);
```

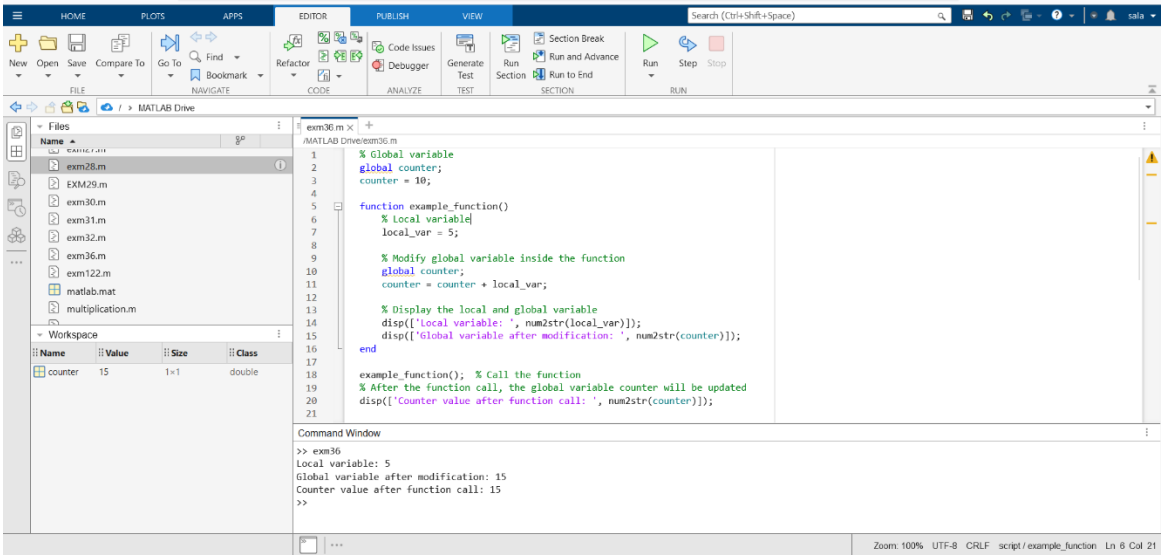


Explanation:

- The function stat_analysis accepts a data vector as input and uses MATLAB's built-in functions sum, mean, and std to calculate the sum, mean, and standard deviation, respectively.
- It then returns these values inside a structure, which can be easily accessed using statistics.sum, statistics. mean, and statistics.std_dev.

° Understand the scope of variables within functions:

```
% Global variable
global counter;
counter = 10;

function example_function()
    % Local variable
    local_var = 5;

    % Modify global variable inside the function
    global counter;
    counter = counter + local_var;

    % Display the local and global variable
    Disp (['Local variable: ', num2str(local_var)]);
    disp (['Global variable after modification: ', num2str(counter)]);
end

example_function();  % Call the function
% After the function call, the global variable counter will be updated
disp (['Counter value after function call: ', num2str(counter)]);
```



Explanation:

- The variable counter is **global**, meaning it can be accessed and modified inside any function as long as it is declared as global inside the function.
- The variable local_var is **local** to example_function, and it cannot be accessed outside this function.
- We modify the global variable inside the function and display both the local and global variables inside the function.

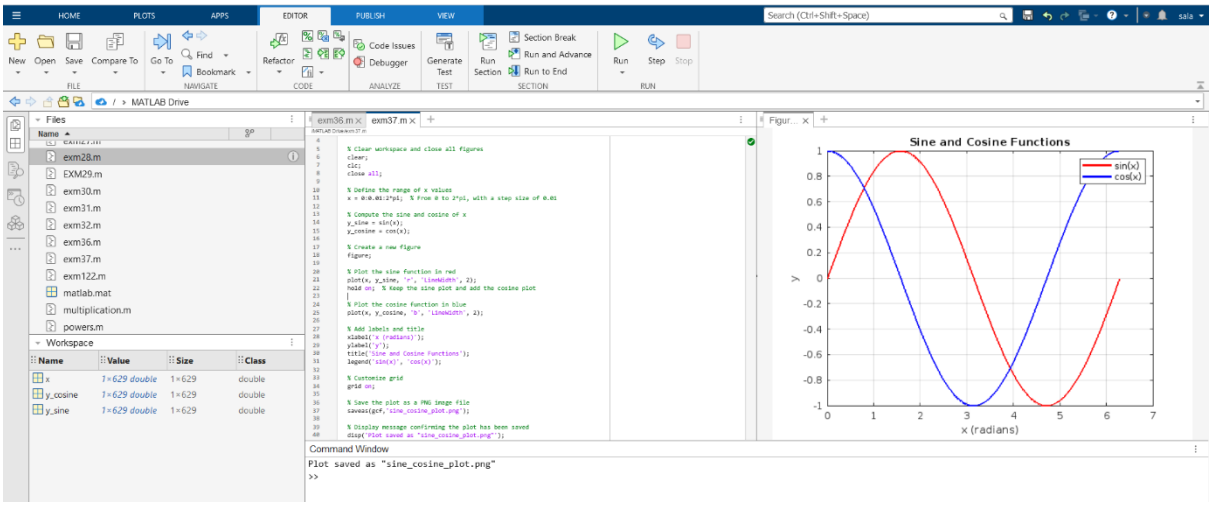
After calling example_function, the global variable counter will be updated to 15 (its original value of 10 plus the local variable value 5).

2. Working with Scripts:

- Create a script file to automate a sequence of commands.
- Use comments to document the script.
- Save and run the script from the command window.

° Create a script file to automate a sequence of commands:

```
% plot_sine_cosine.m
% This script automates the process of plotting the sine and cosine functions
% and customizing the plot.

% Clear workspace and close all figures
clear;
clc;
close all;

% Define the range of x values
x = 0:0.01:2*pi;  % From 0 to 2*pi, with a step size of 0.01

% Compute the sine and cosine of x
y_sine = sin(x);
y_cosine = cos(x);

% Create a new figure
figure;

% Plot the sine function in red
plot(x, y_sine, 'r', 'LineWidth', 2);
hold on;  % Keep the sine plot and add the cosine plot

% Plot the cosine function in blue
Plot (x, y_cosine, 'b', 'LineWidth', 2);

% Add labels and title
Xlabel ('x (radians)');
Ylabel ('y');
title ('Sine and Cosine Functions');
legend('sin(x)', 'cos(x)');

% Customize grid
grid on;

% Save the plot as a PNG image file
Saveas (gcf,'sine_cosine_plot.png');

% Display message confirming the plot has been saved
Disp ('Plot saved as "sine_cosine_plot.png"');
```

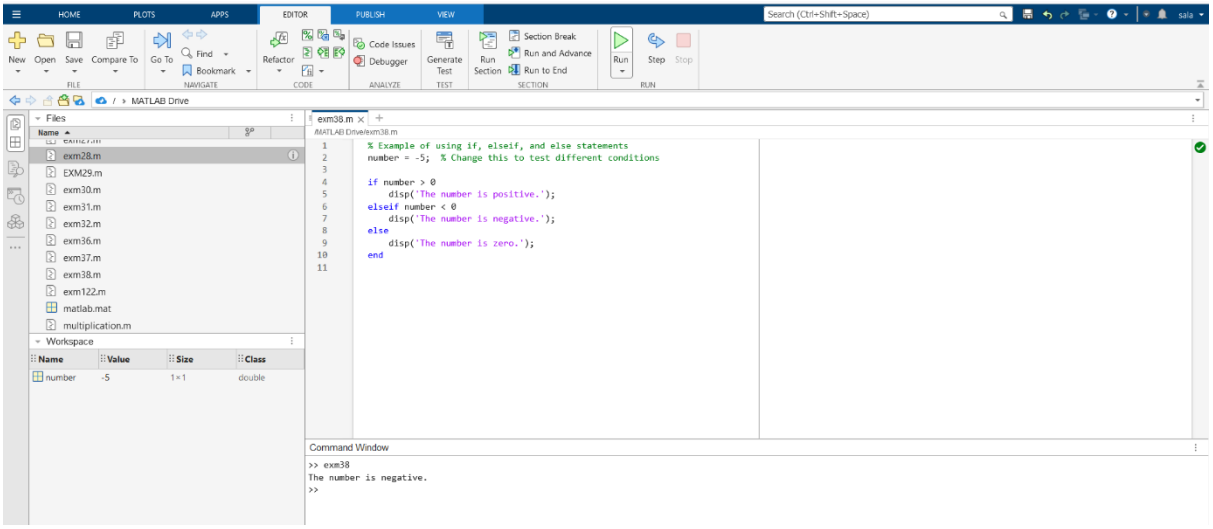


3. Control Flow:

- Use if, else, and elseif statements to control the flow of execution.
- Implement for and while loops for repetitive tasks.
- Use break and continue statements within loops

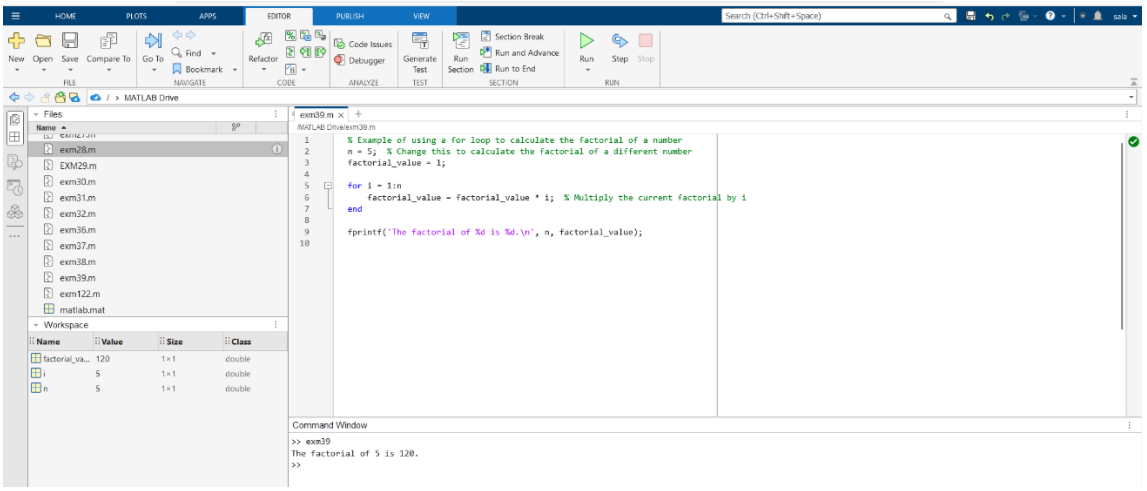
° Use if, else, and elseif statements to control the flow of execution:

```
% Example of using if, elseif, and else statements
number = -5;  % Change this to test different conditions

if number > 0
    disp ('The number is positive.');
elseif number < 0
    disp ('The number is negative.');
else
    disp ('The number is zero.');
end
```



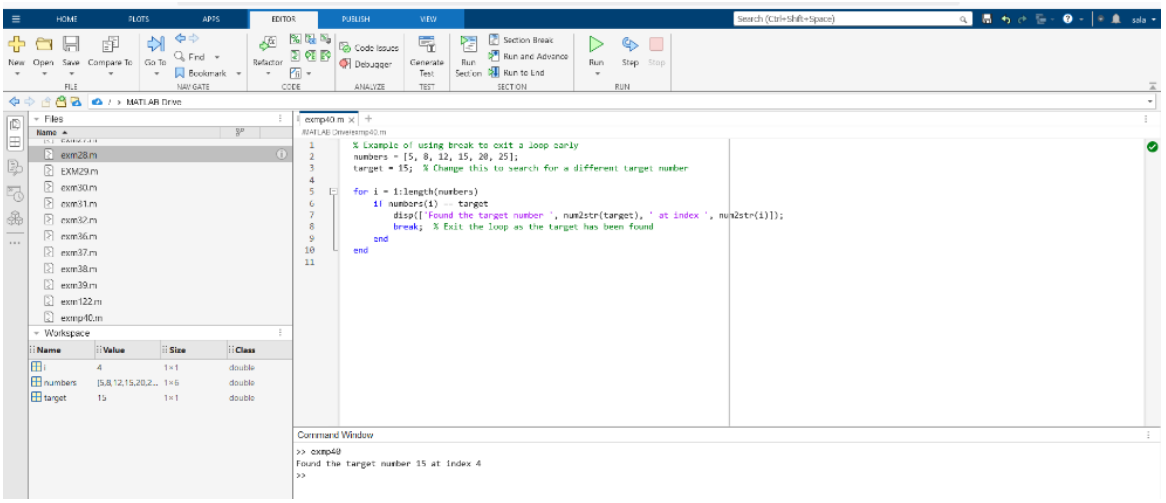
° Implement for and while loops for repetitive tasks:

```
% Example of using a for loop to calculate the factorial of a number
n = 5;  % Change this to calculate the factorial of a different number
factorial_value = 1;

for i = 1:n
    factorial_value = factorial_value * i;  % Multiply the current factorial by i
end

fprintf ('The factorial of %d is %d.\n', n, factorial_value);
```



° Use break and continue statements within loops:

```
% Example of using break to exit a loop early
numbers = [5, 8, 12, 15, 20, 25];
target = 15;  % Change this to search for a different target number

for i = 1:length (numbers)
    if numbers(i) == target
        disp(['Found the target number ', num2str(target), ' at index ', num2str(i)]);
        break;  % Exit the loop as the target has been found
    end
end
```



4. Data Import and Export:

- Import data from external files (e.g., CSV, Excel).
- Export data to external files.
- Use the readmatrix and writematrix functions for data handling

° Import data from external files (e.g., CSV, Excel):

Import CSV file:

```
data = readmatrix('data.csv');
```

Import CSV file:

```
data = readmatrix('data.xlsx');
```

Specify a range or sheet in an Excel file:

```
data = readmatrix('data.xlsx', 'Sheet', 'Sheet1', 'Range', 'A1:C10');
```

° Export data to external files:

Export data to a CSV file:

```
data = [1, 2, 3; 4, 5, 6; 7, 8, 9];
writematrix(data, 'output.csv');
```

Export data to an Excel file:

```
writematrix(data, 'output.xlsx');
```

Specify a sheet and range when exporting to Excel:

```
writematrix(data, 'output.xlsx', 'Sheet', 'Results', 'Range', 'B2');
```

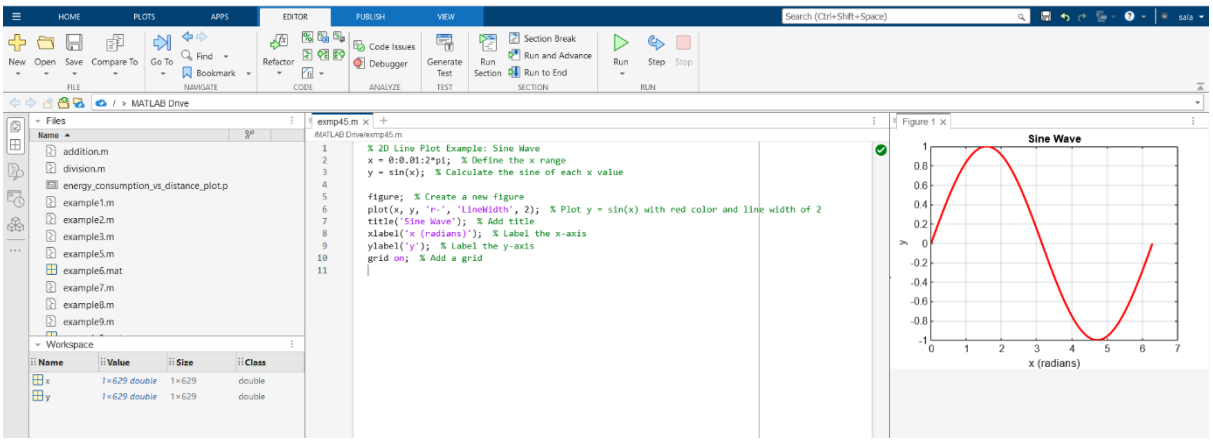
5. Advanced Data Visualization:

- Create 2D and 3D plots (e.g., surface plots, contour plots).
- Use the subplot function to create multiple plots in a single figure.
- Customize plots with different colors, markers, and annotations.

°Create 2D and 3D plots (e.g., surface plots, contour plots):

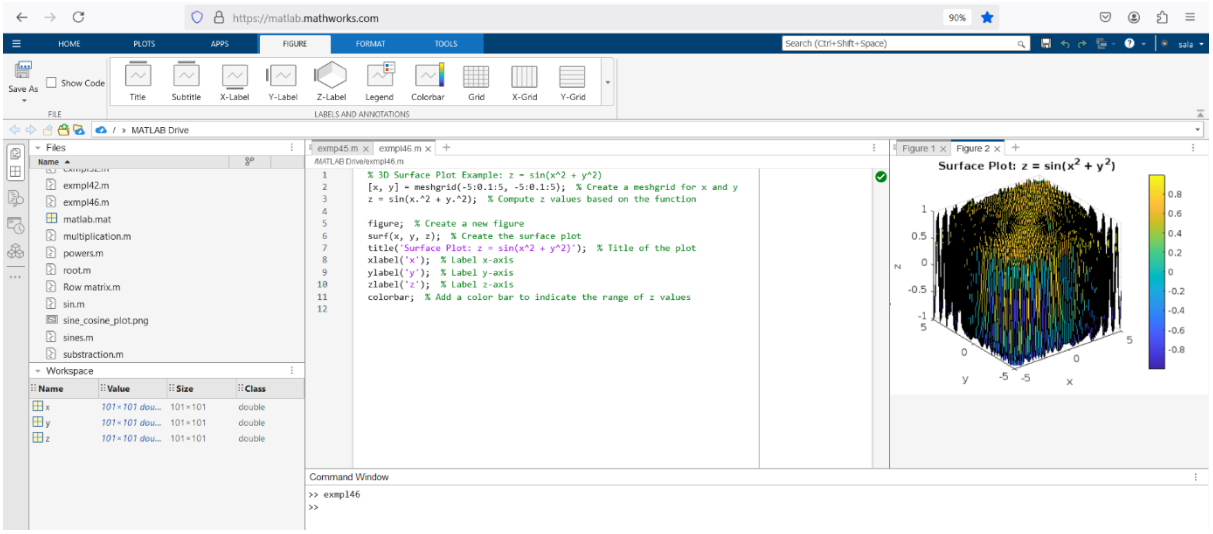
2D Plot (Line plot):

```
% 2D Line Plot Example: Sine Wave
x = 0:0.01:2*pi;  % Define the x range
y = sin(x);  % Calculate the sine of each x value

figure;  % Create a new figure
plot(x, y, 'r-', 'LineWidth', 2);  % Plot y = sin(x) with red color and line width of 2
title('Sine Wave');  % Add title
xlabel('x (radians)');  % Label the x-axis
ylabel('y');  % Label the y-axis
grid on;  % Add a grid
```



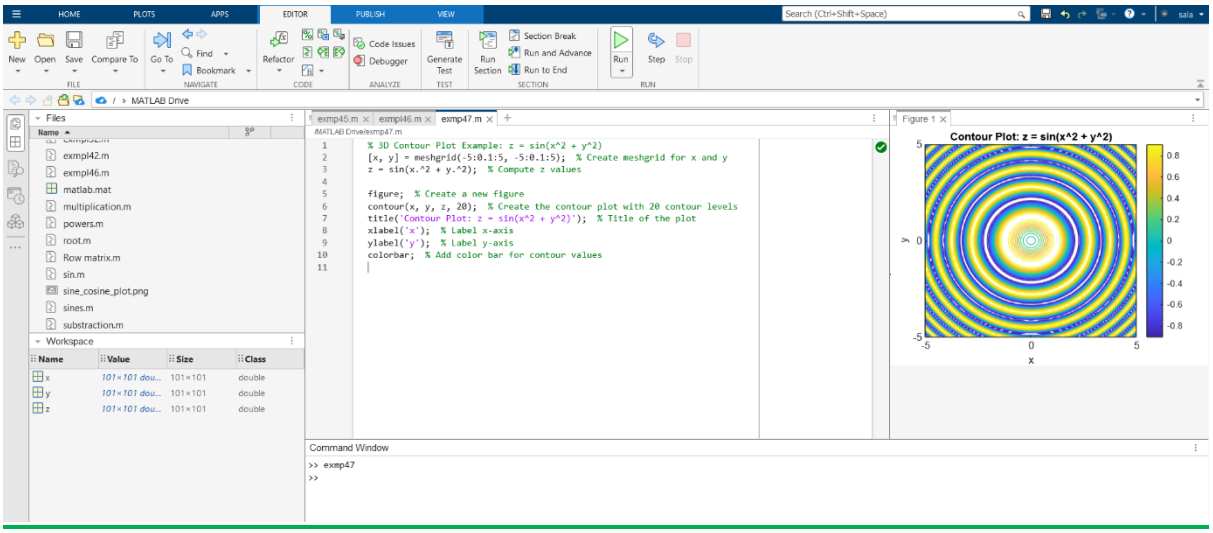
3D Surface Plot:

```
% 3D Surface Plot Example: z = sin(x^2 + y^2)
[x, y] = meshgrid (-5:0.1:5, -5:0.1:5);  % Create a meshgrid for x and y
z = sin (x.^2 + y.^2);  % Compute z values based on the function

figure;  % Create a new figure
surf (x, y, z);  % Create the surface plot
title ('Surface Plot: z = sin (x^2 + y^2)');  % Title of the plot
xlabel('x');  % Label x-axis
ylabel('y');  % Label y-axis
zlabel('z');  % Label z-axis
colorbar;  % Add a color bar to indicate the range of z values
```



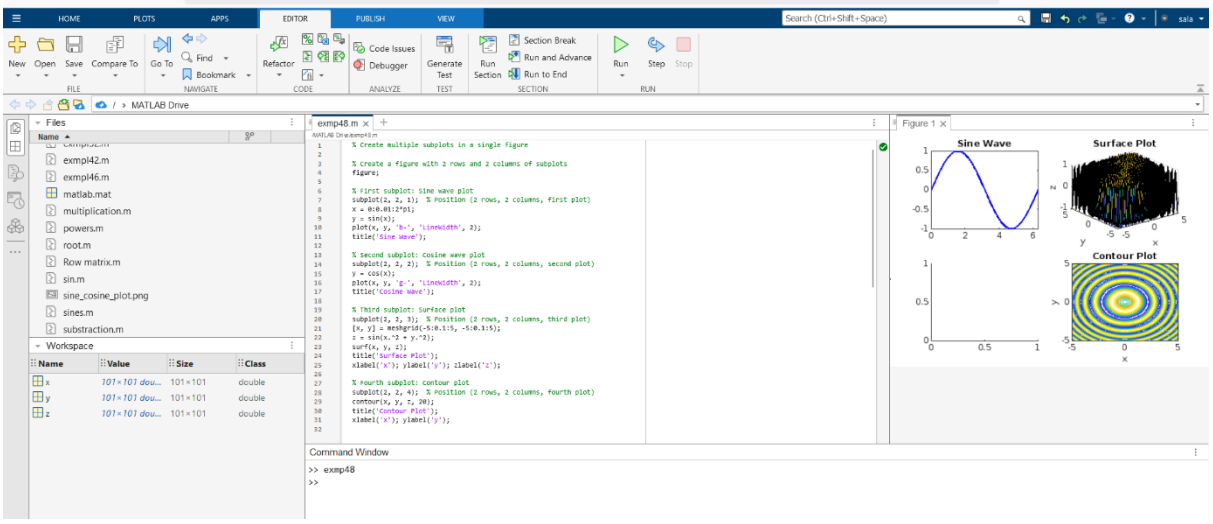
3D Plot: Contour Plot

```
% 3D Contour Plot Example: z = sin (x^2 + y^2)
[x, y] = meshgrid (-5:0.1:5, -5:0.1:5);  % Create meshgrid for x and y
z = sin (x.^2 + y.^2);  % Compute z values

figure;  % Create a new figure
contour (x, y, z, 20);  % Create the contour plot with 20 contour levels
title ('Contour Plot: z = sin(x^2 + y^2)');  % Title of the plot
xlabel('x');  % Label x-axis
ylabel('y');  % Label y-axis
colorbar;  % Add color bar for contour values
```



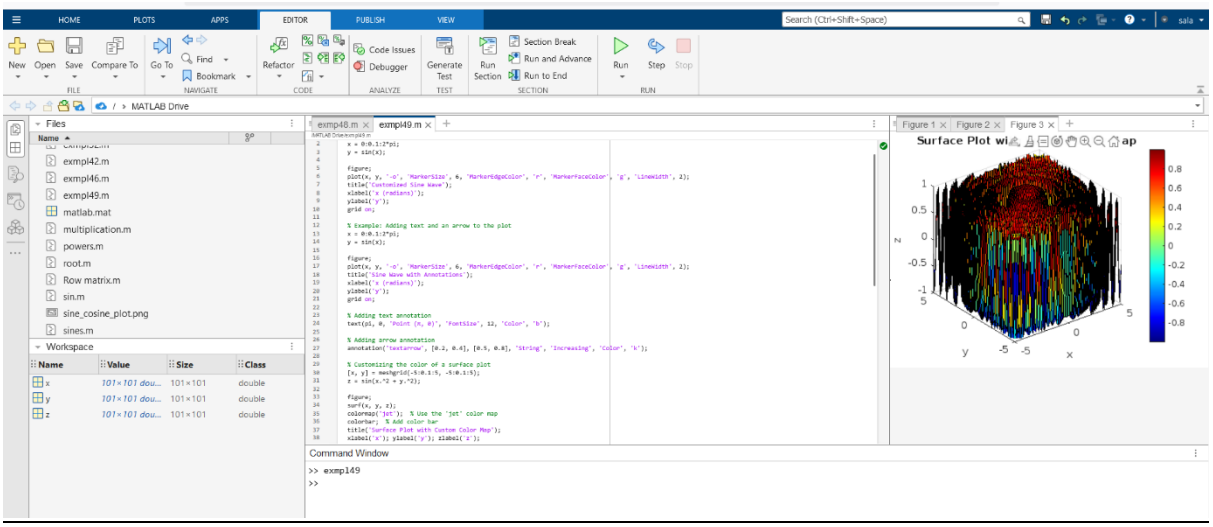
° Use the subplot function to create multiple plots in a single figure:

```
% Create multiple subplots in a single figure

% Create a figure with 2 rows and 2 columns of subplots
figure;

% First subplot: Sine wave plot
subplot (2, 2, 1);  % Position (2 rows, 2 columns, first plot)
x = 0:0.01:2*pi;
y = sin(x);
plot (x, y, 'b-', 'LineWidth', 2);
title ('Sine Wave');

% Second subplot: Cosine wave plot
subplot (2, 2, 2);  % Position (2 rows, 2 columns, second plot)
y = cos(x);
plot (x, y, 'g-', 'LineWidth', 2);
title ('Cosine Wave');

% Third subplot: Surface plot
subplot (2, 2, 3);  % Position (2 rows, 2 columns, third plot)
[x, y] = meshgrid (-5:0.1:5, -5:0.1:5);
z = sin (x.^2 + y.^2);
surf (x, y, z);
title ('Surface Plot');
xlabel('x'); ylabel('y'); zlabel('z');

% Fourth subplot: Contour plot
Subplot (2, 2, 4);  % Position (2 rows, 2 columns, fourth plot)
Contour (x, y, z, 20);
title ('Contour Plot');
xlabel('x'); ylabel('y');
```



° Customize plots with different colors, markers, and annotations:

```
% Example: Line plot with customized markers and colors
x = 0:0.1:2*pi;
y = sin(x);

figure;
plot (x, y, '-o', 'MarkerSize', 6, 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'g', 'LineWidth', 2);
title ('Customized Sine Wave');
xlabel ('x (radians)');
ylabel('y');
grid on;

% Example: Adding text and an arrow to the plot
x = 0:0.1:2*pi;
y = sin(x);

figure;
plot (x, y, '-o', 'MarkerSize', 6, 'MarkerEdgeColor', 'r', 'MarkerFaceColor', 'g', 'LineWidth', 2);
title ('Sine Wave with Annotations');
xlabel ('x (radians)');
ylabel('y');
grid on;

% Adding text annotation
text (pi, 0, 'Point (π, 0)', 'FontSize', 12, 'Color', 'b');

% Adding arrow annotation
Annotation ('textarrow', [0.2, 0.4], [0.5, 0.8], 'String', 'Increasing', 'Color', 'k');

% Customizing the color of a surface plot
[x, y] = meshgrid (-5:0.1:5, -5:0.1:5);
z = sin (x.^2 + y.^2);

figure;
surf (x, y, z);
colormap('jet');  % Use the 'jet' color map
colorbar;  % Add color bar
title ('Surface Plot with Custom Color Map');
xlabel('x'); ylabel('y'); zlabel('z');
```



6. Statistical Analysis:

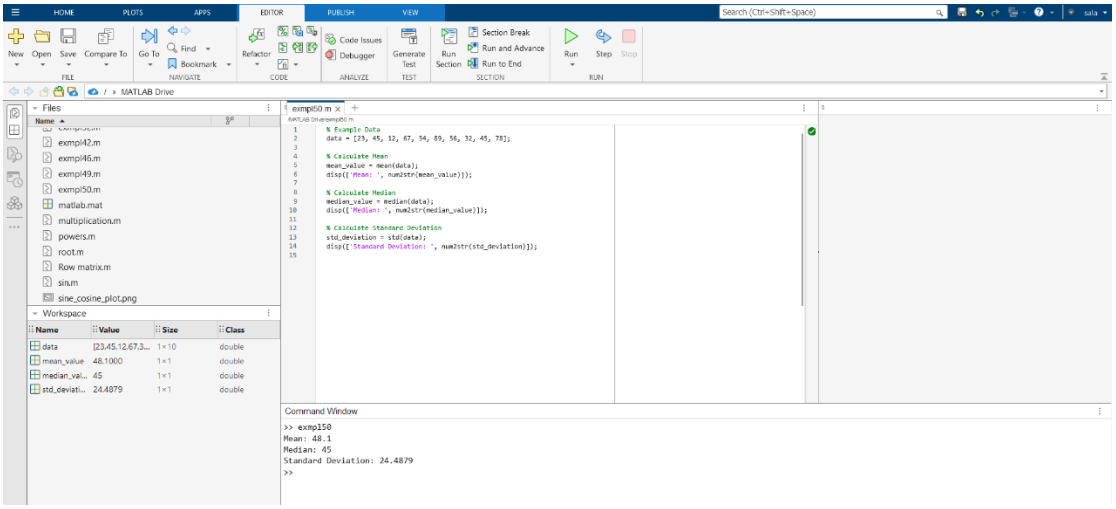
- Perform basic statistical analysis on data (mean, median, standard deviation).
- Use the histogram function to visualize data distribution.
- Implement linear regression and plot the results

° Perform basic statistical analysis on data (mean, median, standard deviation):

```
% Example Data
data = [23, 45, 12, 67, 34, 89, 56, 32, 45, 78];

% Calculate Mean
mean_value = mean(data);
disp (['Mean: ', num2str(mean_value)]);

% Calculate Median
median_value = median(data);
disp (['Median: ', num2str(median_value)]);

% Calculate Standard Deviation
std_deviation = std(data);
disp (['Standard Deviation: ', num2str(std_deviation)]);
```

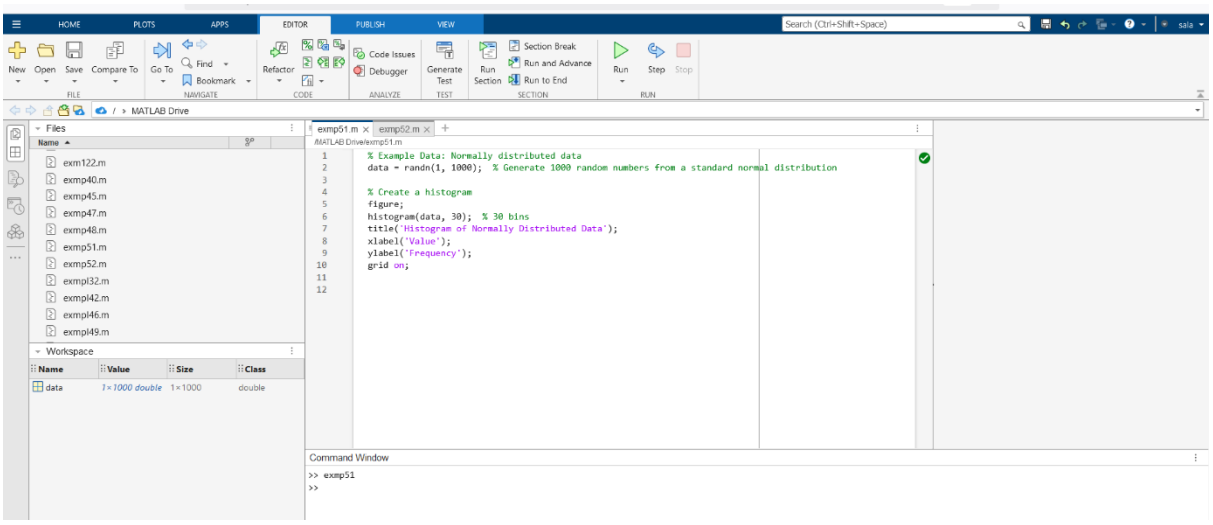
Output:

Mean: 48.1
Median: 45
Standard Deviation: 24.4879



° **Use the histogram function to visualize data distribution:**

```
% Example Data: Normally distributed data
data = randn (1, 1000);  % Generate 1000 random numbers from a standard normal distribution

% Create a histogram
figure;
histogram (data, 30);  % 30 bins
title ('Histogram of Normally Distributed Data');
xlabel('Value');
ylabel('Frequency');
grid on;
```



° **Implement linear regression and plot the results:**

```
% Example Data: Dependent variable (y) and independent variable (x)
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];  % Independent variable
y = [2.3, 4.5, 5.8, 7.2, 9.1, 10.5, 11.9, 14.0, 15.2, 16.1];  % Dependent variable

% Perform linear regression (fit a line)
p = polyfit (x, y, 1);  % p (1) is the slope, p (2) is the intercept

% Generate fitted values (predicted y values)
y_fit = polyval (p, x);  % Compute the fitted values using the polynomial coefficients

% Plot the data points and the linear fit
figure;
scatter (x, y, 'b', 'DisplayName', 'Data Points');  % Scatter plot of the data
hold on;
plot (x, y_fit, 'r-', 'LineWidth', 2, 'DisplayName', 'Linear Fit');  % Plot the linear fit
title ('Linear Regression: y = mx + b');
xlabel('x');
ylabel('y');
legend;
grid on;
```