College of Arts, Technology, and Environment

YEAR 2024/25

Predictive Modeling of Concrete Strength Based on Mixture Composition and Age Using Computational Tools

Assessment Title: Computational Exercise

Module: UBGMW9-15-3 Computational Civil Engineering

Assessment: Task A - Portfolio

Submitted by: -------------------------------

Submitted to: -------------------------------

Submission Date: -----------------------------

ACKNOWLEDGMENT

All the acclamation and appreciation are for Almighty ALLAH, who created the universe and bestowed mankind with the knowledge and wisdom to search for its secrets.

I feel great pleasure and honor to express my deepest sense of gratitude, sincere feelings, and regards to my supervisor for his efficient guidance, tremendous help, and special way of advice for the completion of my studies. I also like to extend my special thanks to my friends for their encouragement and support.

I have gained a lot from the University of the West of England during the last three years, including a civil engineering degree, a new language, a new culture, and many friends. This assignment taught me the value of persistence. In the beginning, I was completely stuck and didn't know how to move forward. But with time, consistent effort, and the guidance I received during lectures, I was able to understand the process and complete the work successfully. I'm thankful for that support; it truly made a difference.

LIST OF FIGURES

ABSTRACT

The analysis and modeling of concrete compressive strength based on its composition and age are detailed in this report. The objective is to develop a model using linear and nonlinear regression classification and compare results to predict the compressive strength of concrete. The database used for this investigation includes precise information on numerous concrete elements, including cement, slag, fly ash, water, and the concrete's age, as well as the related compressive strength values. The experimental data is collected from the UCI Machine Learning Repository, and performing the Analysis, the result shows that the nonlinear regression model is giving an accurate result compared to the linear regression model. The findings demonstrated in this report provide the effectiveness of computational tools in supporting material optimization and predictive decision-making in civil engineering practice.

TABLE OF CONTENTS

Chapter # 01

1. Introduction

Today, civil engineers are starting to use more computers and data to help them do their jobs better. With the help of modern tools, they can now solve problems faster, make better designs, and understand materials in more detail than ever before. One of the main things engineers work with is concrete. Concrete is everywhere; it's used to build homes, schools, office buildings, highways, bridges, and even tunnels. It's strong, lasts a long time, and can be poured into many different shapes, which makes it very useful in all kinds of construction.

But the most important thing about concrete is how much weight it can hold before it breaks. That's called its compressive strength. We Engineers need to make sure that the concrete can support the weight of everything that comes over it. If the concrete isn't strong enough, buildings or bridges could crack or even fall, putting many lives at risk. That's why it's very important to understand what makes concrete strength high and how to predict its strength if we know the quantity of the element that is used to make the concrete.

The strength of concrete depends on what goes into it. It is made by mixing cement, water, sand (fine aggregate), and gravel or crushed stone (coarse aggregate). Sometimes, other materials are added to make the concrete better. For example, fly ash and slag are often added to improve the mix and make it more environmentally friendly. Engineers also use special liquids called superplasticizers. These help make the concrete easier to pour without adding too much water. Also, the age of the concrete (how long it has been left to dry or cure) plays a big part in how strong it becomes over time.

It's not always easy to guess how strong the concrete will be just by looking at the mix. Some ingredients help make it stronger, but others can make it weaker if not used properly. For example, adding more water makes the concrete easier to work with, but too much water can make it weak. Because the relationship between the ingredients and the strength is complicated, engineers now use computers to help them figure out the best mix.

In this project, I used a computer program called MATLAB to look at a set of real concrete data. This data includes information about many different concrete mixes, like how much cement, water, fly ash, and other materials were used, and the measured strength of each mix. I used this data to build two models. These models help predict how strong a mix of concrete will be, even if we haven't tested it yet.

The first model is linear. This means it assumes that the relationship between the ingredients and the strength is simple. For example, if you add more cement, the strength goes up by the same amount each time. The second model is nonlinear, which means it can find more complicated patterns, like when adding more of one ingredient helps at first but then stops helping or even starts hurting the strength.

After building both models, I needed to test how well they worked. To do this, I used three important scores. The first is R^2, which tells us how well the model fits the actual data. The second is RMSE, which shows how far off the model's guesses are from the real numbers. The third is MAE, which tells us the average size of the mistakes. These scores help us know if the model is doing a good job.

Then, I created a tool in MATLAB that acts like a calculator. You type in how much cement, water, gravel, and other ingredients you plan to use, and it gives you a prediction of how strong the concrete will be. This tool is very helpful because it can save time and money. Instead of mixing and testing concrete in a lab every time, engineers can get a good idea from the computer first.

This report has two main parts. Part A explains how I worked with the data, made graphs, and built the models. It includes the steps I followed and the code I used. Part B explains how I made the prediction tool and tested it with new examples. I also included pictures, graphs, and code to help explain everything clearly.

This project shows how computers and engineering can work together to solve real-world problems. Using data and programs like MATLAB helps us engineers understand materials better, build stronger and safer structures, and save time in the process. As technology continues to grow, these kinds of tools will become even more important in building the future.

Chapter # 02

2. Methodology

2.1. Flowchart

In this flowchart, the steps that are taken for the assignment are summarized in Figure 1:

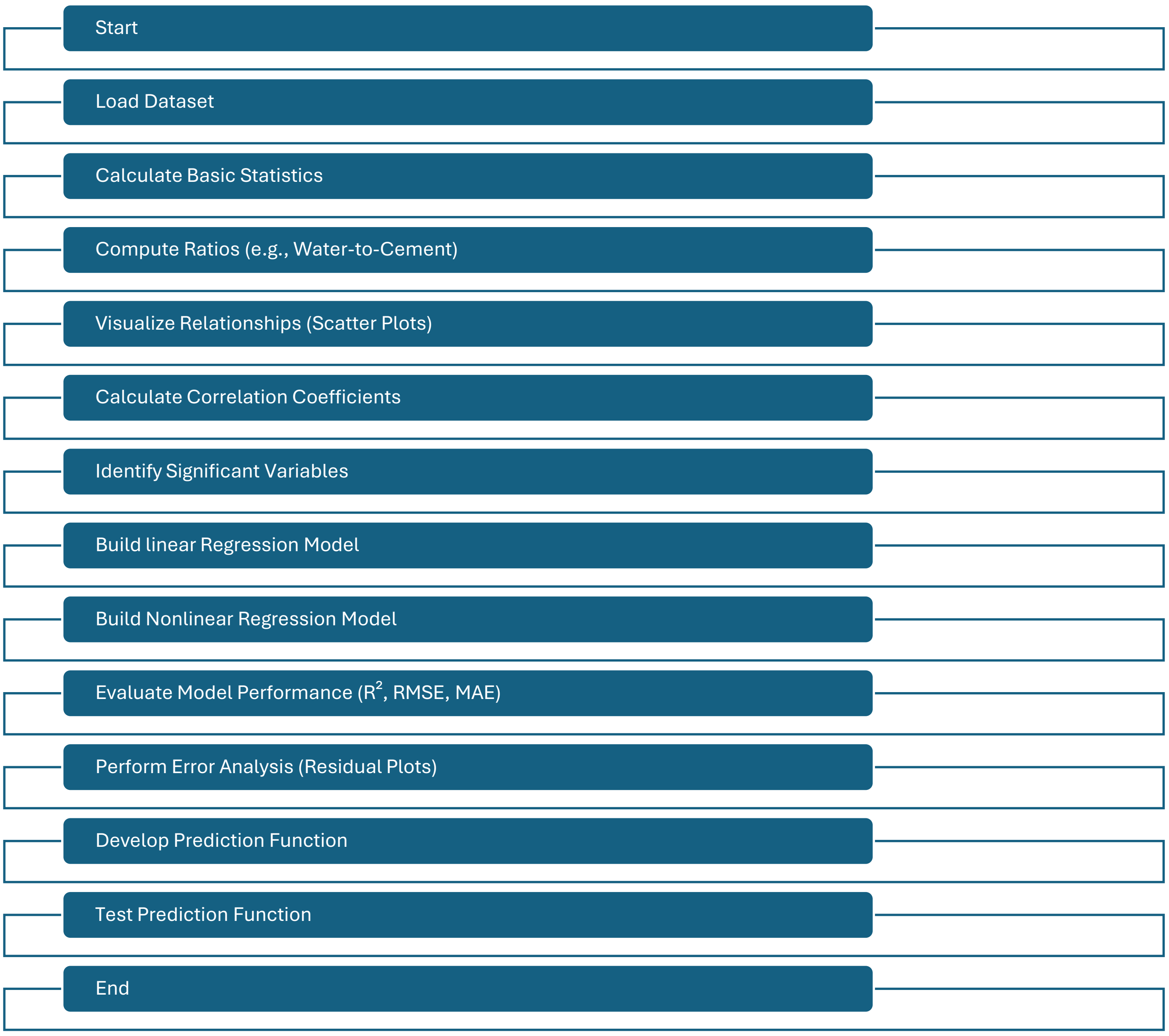
Start
Load Dataset
Calculate Basic Statistics
Compute Ratios (e.g., Water-to-Cement)
Visualize Relationships (Scatter Plots)
Calculate Correlation Coefficients
Identify Significant Variables
Build linear Regression Model
Build Nonlinear Regression Model
Evaluate Model Performance (R^2, RMSE, MAE)
Perform Error Analysis (Residual Plots)
Develop Prediction Function
Test Prediction Function
End

Figure 1 Flow chart of Regression Model

2.2. Steps Taken

2.2.1. Step 1: Loading the Dataset

The dataset was loaded into MATLAB using the readtable function. This dataset contains 8 input variables (cement, slag, fly ash, water, superplasticizer, coarse aggregate, fine aggregate, and age) and 1 output variable (compressive strength).

```
data = readtable("Concrete_Data.xls", 'VariableNamingRule', 'preserve');

% Display the first few rows of the dataset to verify it loaded correctly
disp('First few rows of the dataset:');
head(data);
```

2.2.2. Step 2: Calculating Basic Statistics

Basic statistics (maximum, minimum, mean, and standard deviation) were calculated for each variable. These statistics were stored in a table and exported to an Excel file (statistics.xlsx).

```
stats = table();
stats.Variable = data.Properties.VariableNames; % Store variable names
stats.Max = max(data{:,:}); % Calculate maximum values
stats.Min = min(data{:,:}); % Calculate minimum values
Stats.Mean = mean(data{:,:}); % Calculate mean values
stats.Std = std(data{:,:}); % Calculate standard deviation

% Display the basic statistics
disp('Basic Statistics:');
disp(stats);

% Export the statistics to an Excel file
writetable(stats, 'statistics.xlsx');
```

2.2.3. Step 3: Calculating Ratios

The following ratios were calculated to explore the relationship between different components in the concrete mixture:

- Water-to-cement ratio
- Water-to-binder ratio
- Fly ash-to-binder ratio
- Slag-to-binder ratio
- Fly ash + slag-to-binder ratio

These ratios provide insights into how the proportion of ingredients impacts the compressive strength.

```
ratios = table();

% Calculate the water-to-cement ratio
ratios.WaterToCement = data.("Water  (component 4)(kg in a m^3 mixture)") ./ data.("Cement (component 1)(kg in a m^3 mixture)");

% Calculate the water-to-binder ratio (binder = cement + fly ash + slag)
ratios.WaterToBinder = data.("Water  (component 4)(kg in a m^3 mixture)") ./ ...
    (data.("Cement (component 1)(kg in a m^3 mixture)") + ...
    data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)"));

% Export the ratios to a new sheet in the Excel file
writetable(ratios, 'statistics.xlsx', 'Sheet', 'Ratios');
```

2.2.4. Step 4: Visualizing Relationships

Scatter plots were created to visualize the relationships between each variable (cement, water, slag, etc.) and compressive strength. Additionally, plots for the calculated ratios and compressive strength were generated.

```
figure;
for i = 1:width(data)-1
    subplot(3, 3, i); % Create subplots in a 3x3 grid
    scatter(data{:, i}, data.("Concrete compressive strength(MPa, megapascals)")); % Scatter plot
    mean_val = mean(data{:, i}); % Calculate mean of the variable
    std_val = std(data{:, i}); % Calculate standard deviation of the variable
    title(sprintf('%s\nMean: %.2f, Std: %.2f', data.Properties.VariableNames{i}, mean_val, std_val)); % Add title
    xlabel(data.Properties.VariableNames{i}); % Label x-axis
    ylabel('Compressive Strength (MPa)'); % Label y-axis
    grid on; % Add gridlines
    box on;  % Add box borders
end
% Save the figure as a JPG file
saveas(gcf, 'scatter_plots_with_stats.jpg');
```

2.2.5. Step 5: Calculating Correlation Coefficients

Correlation coefficients were calculated to determine how strongly each input variable is related to the compressive strength. The table of correlation coefficients was displayed for reference.

```
input_vars = data{:, 1:end-1}; % Input variables
output_var = data{:, end};     % Output variable (compressive strength)

% Calculate correlation coefficients between input variables and compressive strength
correlation_matrix = corrcoef([input_vars, output_var]);

% Extract the correlation coefficients between input variables and compressive strength
correlation_with_strength = correlation_matrix(1:end-1, end);

% Display the correlation coefficients in a table
correlation_table = array2table(correlation_with_strength, ...
    'VariableNames', {'Correlation'}, ...
    'RowNames', data.Properties.VariableNames(1:end-1));
disp('Correlation Coefficients:');
disp(correlation_table);
```

2.2.6. Step 6: Identifying Significant Variables

Based on the correlation coefficients, variables with significant relationships to compressive strength were identified. A threshold (e.g., 0.5) was used to filter significant variables.

```
threshold = input('Enter the correlation coefficient threshold (e.g., 0.5): ');

% Find variables with correlation coefficients above the threshold
significant_vars = correlation_with_strength(abs(correlation_with_strength) > threshold);

% Get the names of the significant variables
significant_variable_names = data.Properties.VariableNames(abs(correlation_with_strength) > threshold);
```

```
% Display the significant variables
disp('Significant Variables:');
disp(significant_variable_names');
```

2.2.7. Step 7: Building Regression Models

2.2.7.1. Linear Regression Model:

A linear regression model was built using the significant variables. The model was trained using the training data, and predictions were made for the testing data.

```
significant_data = data{:, significant_variable_names}; % Input variables
compressive_strength = data.("Concrete compressive strength(MPa, megapascals)"); % Output variable

% Split the data into training and testing sets (80% training, 20% testing)
rng(42); % Set a random seed for reproducibility
split_ratio = 0.8; % 80% training, 20% testing
split_index = floor(split_ratio * height(data)); % Index to split the data

% Training data
X_train = significant_data(1:split_index, :);
y_train = compressive_strength(1:split_index);

% Testing data
X_test = significant_data(split_index+1:end, :);
y_test = compressive_strength(split_index+1:end);

% Fit a linear regression model using the training data
linear_model = fitlm(X_train, y_train);
disp('Linear Regression Model Summary:');
disp(linear_model);

% Predict compressive strength on the testing data
y_pred = predict(linear_model, X_test);
```

2.2.7.2. Nonlinear Regression Model:

A nonlinear regression model was built, considering quadratic terms for the significant variables. The model was evaluated using the same metrics as the linear model.

```
modelfun = @(b, X) b(1) + ... % Intercept
             b(2)*X(:,1) + ... % Cement (linear)
             b(3)*X(:,2) + ... % Blast Furnace Slag (linear)
             b(4)*X(:,3) + ... % Fly Ash (linear)
             b(5)*X(:,4) + ... % Water (linear)
             b(6)*X(:,5) + ... % Superplasticizer (linear)
             b(7)*X(:,6) + ... % Coarse Aggregate (linear)
             b(8)*X(:,7) + ... % Fine Aggregate (linear)
             b(9)*X(:,8) + ... % Age (linear)
             b(10)*X(:,1).^2 + ... % Cement (quadratic)
             b(11)*X(:,5).^2 + ... % Superplasticizer (quadratic)
             b(12)*X(:,8).^2;      % Age (quadratic)

% Initial guess for coefficients
beta0 = [1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.001, 0.001, 0.001];

% Fit the nonlinear model
nonlinear_model = fitnlm(X_train, y_train, modelfun, beta0);

% Predict compressive strength on the testing data
y_pred_nl = predict(nonlinear_model, X_test);

% Calculate evaluation metrics
ss_total_nl = sum((y_test - mean(y_test)).^2);      % Total sum of squares
ss_residual_nl = sum((y_test - y_pred_nl).^2);      % Residual sum of squares
r2_score_nl = 1 - (ss_residual_nl / ss_total_nl);  % R² score
rmse_nl = sqrt(mean((y_test - y_pred_nl).^2));      % Root Mean Squared Error
mae_nl = mean(abs(y_test - y_pred_nl));             % Mean Absolute Error
```

2.2.8. Step 8: Error Analysis

Residual plots were generated to visually assess the prediction errors for both the linear and nonlinear models. These plots helped identify any patterns in the residuals that might indicate areas for model improvement.

```
figure;
subplot(1, 2, 1);
scatter(y_test, y_pred);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Linear Model: Predicted vs. Actual');
grid on;
subplot(1, 2, 2);
scatter(y_test, y_pred_nl);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Nonlinear Model: Predicted vs. Actual');
grid on;
saveas(gcf, 'predicted_vs_actual_both_models.jpg');
```

2.2.9. Step 9: Developing a Prediction Function

A prediction function (predict_concrete_strength) was developed to allow users to predict the compressive strength of concrete based on a given set of input parameters.

```
function predicted_strength = predict_concrete_strength(cement, slag, fly_ash, water, superplasticizer,
coarse_agg, fine_agg, age)
    % Load the saved nonlinear model
    % The nonlinear model is loaded from the file 'nonlinear_model.mat'
    % This file contains the trained nonlinear regression model
    load('nonlinear_model.mat', 'nonlinear_model');

    % Create an input array for the new data
    % The input data is organized in the same order as the training data:
    % [cement, slag, fly_ash, water, superplasticizer, coarse_agg, fine_agg, age]
    input_data = [cement, slag, fly_ash, water, superplasticizer, coarse_agg, fine_agg, age];

    % Predict the compressive strength using the nonlinear model
    % The 'predict' function uses the loaded model to predict the strength
    % based on the input data
    predicted_strength = predict(nonlinear_model, input_data);

    % Display a message indicating that the nonlinear model is being used
    disp('Using Nonlinear Model for Prediction');
end
```

2.2.10. Step 10: Testing the Prediction Function

To predict the compressive strength the following parameter are taken as input and determine the compressive strength for the testing.

```
cement = 300; slag = 100; fly_ash = 50; water = 150; superplasticizer = 5; coarse_agg = 900; fine_agg =
700; age = 28;
predicted_strength = predict_concrete_strength(cement, slag, fly_ash, water, superplasticizer, coarse_agg,
fine_agg, age);
fprintf('Predicted Compressive Strength: %.2f MPa\n', predicted_strength);
```

Chapter # 03

3. Results and Discussion

3.1. Model Performance

The performance of the two models, linear and nonlinear, was evaluated using three key metrics: R² Score**,** RMSE (Root Mean Squared Error)**,** and MAE (Mean Absolute Error)**.** These metrics provide insight into how well the models fit the data and how accurately they predict concrete's compressive strength.

3.1.1. Linear Model:

- **R² Score:** 0.5039
- **RMSE:** 8.7018
- **MAE:** 6.8719

3.1.2. Nonlinear Model:

- **R² Score:** 0.5736
- **RMSE:** 8.0675
- **MAE:** 6.1544

Based on the result Nonlinear model performs better than the linear model, as in the case of the nonlinear model, the difference between the actual and predicted value is lower, which indicates that the Nonlinear model performs best.

- **Higher R² score** (0.5736 vs. 0.5039), indicating better explanatory power.
- **Lower RMSE** (8.0675 vs. 8.7018), indicating better prediction accuracy.
- **Lower MAE** (6.5261 vs. 6.8719), indicating smaller prediction errors.

These results confirm that incorporating the nonlinear relationship captures the better relationship because of the complexity of the data and helps to predict the compressive strength to be more accurate.

3.2. Error Analysis

Both models showed that the nonlinear model had more randomly distributed residuals compared to the linear model. This suggests that the nonlinear model better represents the data underlying relationships. In the appendix, the Error Analysis jpg is added, which shows the clear residual between both models.

3.3. Impact of Number of Variables

Especially in a regression model number of the variable has a significant impact on its performance. We used correlation coefficients to identify the most significant variables and excluded those with weak relationships to the compressive strength by adding the threshold. This approach improved the performance of the model and reduced the chance of overfitting.

3.4. Limitations

- The current dataset is relatively small, which may limit the model's ability to generalize to new data.
- The models assume that the relationships between input variables and compressive strength are stationary in the linear regression model (i.e., they do not change over time or across different conditions).
- While the nonlinear model performs well, there is a risk of overfitting, especially if more complex models are used without sufficient data.

CONCLUSION

The nonlinear regression model performed better than the linear model in predicting concrete compressive strength. The model had higher R^2 scores, explaining more variability in the data, and provided reduced prediction errors (lower RMSE and MAE values). This increase demonstrates the importance of capturing nonlinear interactions in data, which the linear model could not handle as well.

4. Appendix

4.1. MATLAB Code

The MATLAB code used for this analysis is provided in two parts:

4.1.1. Part A.m:

Code for data loading, analysis, and model building.

```
%% Step 1: Load the Dataset
% Load the dataset from the Excel file into a MATLAB table
data = readtable("Concrete_Data.xls", 'VariableNamingRule', 'preserve');

% Display the first few rows of the dataset to verify it loaded correctly
disp('First few rows of the dataset:');
head(data);

%% Step 2: Calculate Basic Statistics
% Create a table to store basic statistics (max, min, mean, std) for each variable
stats = table();
stats.Variable = data.Properties.VariableNames; % Store variable names
stats.Max = max(data{:,:}); % Calculate maximum values
stats.Min = min(data{:,:}); % Calculate minimum values
stats.Mean = mean(data{:,:}); % Calculate mean values
stats.Std = std(data{:,:}); % Calculate standard deviation

% Display the basic statistics
disp('Basic Statistics:');
disp(stats);

% Export the statistics to an Excel file
writetable(stats, 'statistics.xlsx');

%% Step 3: Calculate Ratios
% Create a table to store calculated ratios
ratios = table();
```

```
% Calculate the water-to-cement ratio
ratios.WaterToCement = data.("Water  (component 4)(kg in a m^3 mixture)") ./ data.("Cement (component 
1)(kg in a m^3 mixture)");

% Calculate the water-to-binder ratio (binder = cement + fly ash + slag)
ratios.WaterToBinder = data.("Water  (component 4)(kg in a m^3 mixture)") ./ ...
    (data.("Cement (component 1)(kg in a m^3 mixture)") + ...
    data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)"));

% Calculate the fly ash-to-binder ratio
ratios.FlyAshToBinder = data.("Fly Ash (component 3)(kg in a m^3 mixture)") ./ ...
    (data.("Cement (component 1)(kg in a m^3 mixture)") + ...
    data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)"));

% Calculate the slag-to-binder ratio
ratios.SlagToBinder = data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)") ./ ...
    (data.("Cement (component 1)(kg in a m^3 mixture)") + ...
    data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)"));

% Calculate the (fly ash + slag)-to-binder ratio
ratios.FlyAshAndSlagToBinder = (data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)")) ./ ...
    (data.("Cement (component 1)(kg in a m^3 mixture)") + ...
    data.("Fly Ash (component 3)(kg in a m^3 mixture)") + ...
    data.("Blast Furnace Slag (component 2)(kg in a m^3 mixture)"));

% Export the ratios to a new sheet in the Excel file
writetable(ratios, 'statistics.xlsx', 'Sheet', 'Ratios');

%% Step 4: Visualize Relationships
% Create scatter plots to visualize the relationship between each variable and compressive strength
figure;
for i = 1:width(data)-1
```

```
    subplot(3, 3, i); % Create subplots in a 3x3 grid
    scatter(data{:, i}, data.("Concrete compressive strength(MPa, megapascals)")); % Scatter plot
    mean_val = mean(data{:, i}); % Calculate mean of the variable
    std_val = std(data{:, i}); % Calculate standard deviation of the variable
    title(sprintf('%s\nMean: %.2f, Std: %.2f', data.Properties.VariableNames{i}, mean_val, std_val)); % Add
title
    xlabel(data.Properties.VariableNames{i}); % Label x-axis
    ylabel('Compressive Strength (MPa)'); % Label y-axis
    grid on; % Add gridlines
    box on;  % Add box borders
end
% Save the figure as a JPG file
saveas(gcf, 'scatter_plots_with_stats.jpg');

%% Step 4.2: Visualize Relationships for Ratios
% Create scatter plots for the calculated ratios and compressive strength
figure;
subplot(2, 3, 1);
scatter(ratios.WaterToCement, data.("Concrete compressive strength(MPa, megapascals)"));
title(sprintf('Water-to-Cement\nMean: %.2f, Std: %.2f', mean(ratios.WaterToCement),
std(ratios.WaterToCement)));
xlabel('Water-to-Cement Ratio');
ylabel('Compressive Strength (MPa)');
grid on;
box on;

subplot(2, 3, 2);
scatter(ratios.WaterToBinder, data.("Concrete compressive strength(MPa, megapascals)"));
title(sprintf('Water-to-Binder\nMean: %.2f, Std: %.2f', mean(ratios.WaterToBinder),
std(ratios.WaterToBinder)));
xlabel('Water-to-Binder Ratio');
ylabel('Compressive Strength (MPa)');
grid on;
box on;

subplot(2, 3, 3);
```

```
scatter(ratios.FlyAshToBinder, data.("Concrete compressive strength(MPa, megapascals)"));
title(sprintf('Fly Ash-to-Binder\nMean: %.2f, Std: %.2f', mean(ratios.FlyAshToBinder),
std(ratios.FlyAshToBinder)));
xlabel('Fly Ash-to-Binder Ratio');
ylabel('Compressive Strength (MPa)');
grid on;
box on;

subplot(2, 3, 4);
scatter(ratios.SlagToBinder, data.("Concrete compressive strength(MPa, megapascals)"));
title(sprintf('Slag-to-Binder\nMean: %.2f, Std: %.2f', mean(ratios.SlagToBinder),
std(ratios.SlagToBinder)));
xlabel('Slag-to-Binder Ratio');
ylabel('Compressive Strength (MPa)');
grid on;
box on;

subplot(2, 3, 5);
scatter(ratios.FlyAshAndSlagToBinder, data.("Concrete compressive strength(MPa, megapascals)"));
title(sprintf('Fly Ash + Slag-to-Binder\nMean: %.2f, Std: %.2f', mean(ratios.FlyAshAndSlagToBinder),
std(ratios.FlyAshAndSlagToBinder)));
xlabel('Fly Ash + Slag-to-Binder Ratio');
ylabel('Compressive Strength (MPa)');
grid on;
box on;

% Save the figure as a JPG file
saveas(gcf, 'scatter_plots_for_ratios.jpg');

%% Step 5: Calculate Correlation Coefficients
% Extract input variables (all columns except the last one) and output variable (last column)
input_vars = data{:, 1:end-1}; % Input variables
output_var = data{:, end};     % Output variable (compressive strength)

% Calculate correlation coefficients between input variables and compressive strength
correlation_matrix = corrcoef([input_vars, output_var]);
```

```
% Extract the correlation coefficients between input variables and compressive strength
correlation_with_strength = correlation_matrix(1:end-1, end);

% Display the correlation coefficients in a table
correlation_table = array2table(correlation_with_strength, ...
    'VariableNames', {'Correlation'}, ...
    'RowNames', data.Properties.VariableNames(1:end-1));
disp('Correlation Coefficients:');
disp(correlation_table);

%% Step 6: Identify Significant Variables
% Prompt the user to input a correlation threshold
threshold = input('Enter the correlation coefficient threshold (e.g., 0.5): ');

% Find variables with correlation coefficients above the threshold
significant_vars = correlation_with_strength(abs(correlation_with_strength) > threshold);

% Get the names of the significant variables
significant_variable_names = data.Properties.VariableNames(abs(correlation_with_strength) > threshold);

% Display the significant variables
disp('Significant Variables:');
disp(significant_variable_names');

%% Step 6.2: 3D Scatter Plots for Top Two Influential Variables
% Check if there are at least two significant variables
if length(significant_variable_names) >= 2
    % Extract the top two influential variables
    top_two_vars = significant_variable_names(1:2);

    % Extract the data for the top two variables and compressive strength
    var1 = data{:, top_two_vars{1}};
    var2 = data{:, top_two_vars{2}};
    strength = data.("Concrete compressive strength(MPa, megapascals)");
    % Create a 3D scatter plot
    figure;
    scatter3(var1, var2, strength, 'filled');
    xlabel(top_two_vars{1});
    ylabel(top_two_vars{2});
    zlabel('Compressive Strength (MPa)');
    title(sprintf('3D Scatter Plot: %s vs %s vs Compressive Strength', top_two_vars{1}, top_two_vars{2}));
    grid on;
    box on;

    % Save the plot as a JPG file
    saveas(gcf, '3d_scatter_plot_top_two_variables.jpg');
else
    disp('Not enough significant variables to create a 3D scatter plot.');
end

%% Step 7: Linear Regression
% Extract the significant variables and the output variable
significant_data = data{:, significant_variable_names}; % Input variables
compressive_strength = data.("Concrete compressive strength(MPa, megapascals)"); % Output variable

% Split the data into training and testing sets (80% training, 20% testing)
rng(42); % Set a random seed for reproducibility
split_ratio = 0.8; % 80% training, 20% testing
split_index = floor(split_ratio * height(data)); % Index to split the data

% Training data
X_train = significant_data(1:split_index, :);
y_train = compressive_strength(1:split_index);

% Testing data
X_test = significant_data(split_index+1:end, :);
y_test = compressive_strength(split_index+1:end);

% Fit a linear regression model using the training data
linear_model = fitlm(X_train, y_train);
```

```
% Display the model summary
disp('Linear Regression Model Summary:');
disp(linear_model);

% Predict compressive strength on the testing data
y_pred = predict(linear_model, X_test);

% Calculate evaluation metrics: R², RMSE, and MAE
ss_total_linear = sum((y_test - mean(y_test)).^2);       % Total sum of squares
ss_residual_linear = sum((y_test - y_pred).^2);         % Residual sum of squares
r2_score_linear = 1 - (ss_residual_linear / ss_total_linear);   % R² score
rmse = sqrt(mean((y_test - y_pred).^2));  % Root Mean Squared Error
mae = mean(abs(y_test - y_pred));         % Mean Absolute Error

% Display the evaluation metrics
disp('Linear Model Performance Metrics (Testing Data):');
fprintf('R² Score: %.4f\n', r2_score_linear);
fprintf('RMSE: %.4f\n', rmse);
fprintf('MAE: %.4f\n\n', mae);

% Plot predicted vs. actual compressive strength
figure;
scatter(y_test, y_pred);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Predicted vs. Actual Compressive Strength (Linear Model)');
grid on;
saveas(gcf, 'linear_model_prediction_plot.jpg');

% Plot residuals
residuals = y_test - y_pred;
figure;
scatter(y_pred, residuals);
```

```
hold on;
plot([min(y_pred), max(y_pred)], [0, 0], 'r--'); % Reference line at zero
xlabel('Predicted Compressive Strength (MPa)');
ylabel('Residuals');
title('Residual Plot (Linear Model)');
grid on;
saveas(gcf, 'linear_model_residual_plot.jpg');

%% Step 8: Nonlinear Regression
% Define the nonlinear model
modelfun = @(b, X) b(1) + ... % Intercept
           b(2)*X(:,1) + ... % Cement (linear)
           b(3)*X(:,2) + ... % Blast Furnace Slag (linear)
           b(4)*X(:,3) + ... % Fly Ash (linear)
           b(5)*X(:,4) + ... % Water (linear)
           b(6)*X(:,5) + ... % Superplasticizer (linear)
           b(7)*X(:,6) + ... % Coarse Aggregate (linear)
           b(8)*X(:,7) + ... % Fine Aggregate (linear)
           b(9)*X(:,8) + ... % Age (linear)
           b(10)*X(:,1).^2 + ... % Cement (quadratic)
           b(11)*X(:,5).^2 + ... % Superplasticizer (quadratic)
           b(12)*X(:,8).^2;    % Age (quadratic)

% Initial guess for coefficients
beta0 = [1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.001, 0.001, 0.001];

% Fit the nonlinear model
nonlinear_model = fitnlm(X_train, y_train, modelfun, beta0);

% Predict compressive strength on the testing data
y_pred_nl = predict(nonlinear_model, X_test);

% Calculate evaluation metrics
ss_total_nl = sum((y_test - mean(y_test)).^2);     % Total sum of squares
ss_residual_nl = sum((y_test - y_pred_nl).^2);     % Residual sum of squares
r2_score_nl = 1 - (ss_residual_nl / ss_total_nl);  % R² score
```

```
rmse_nl = sqrt(mean((y_test - y_pred_nl).^2));      % Root Mean Squared Error
mae_nl = mean(abs(y_test - y_pred_nl));             % Mean Absolute Error

% Display the evaluation metrics
disp('Nonlinear Model Performance Metrics:');
fprintf('R² Score: %.4f\n', r2_score_nl);
fprintf('RMSE: %.4f\n', rmse_nl);
fprintf('MAE: %.4f\n', mae_nl);

% Plot predicted vs. actual compressive strength
figure;
scatter(y_test, y_pred_nl);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Predicted vs. Actual Compressive Strength (Nonlinear Model)');
grid on;
saveas(gcf, 'nonlinear_model_prediction_plot.jpg');

% Plot residuals
residuals_nl = y_test - y_pred_nl;
figure;
scatter(y_pred_nl, residuals_nl);
hold on;
plot([min(y_pred_nl), max(y_pred_nl)], [0, 0], 'r--'); % Reference line at zero
xlabel('Predicted Compressive Strength (MPa)');
ylabel('Residuals');
title('Residual Plot (Nonlinear Model)');
grid on;
saveas(gcf, 'nonlinear_model_residual_plot.jpg');

%% Task 7: Error Analysis
% Predicted vs. Actual Values for Both Models
figure;
subplot(1, 2, 1);
```

```
scatter(y_test, y_pred);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Linear Model: Predicted vs. Actual');
grid on;

subplot(1, 2, 2);
scatter(y_test, y_pred_nl);
hold on;
plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--'); % Reference line
xlabel('Actual Compressive Strength (MPa)');
ylabel('Predicted Compressive Strength (MPa)');
title('Nonlinear Model: Predicted vs. Actual');
grid on;
saveas(gcf, 'predicted_vs_actual_both_models.jpg');

% Residuals for Both Models
residuals_linear = y_test - y_pred;      % Residuals for linear model
residuals_nonlinear = y_test - y_pred_nl; % Residuals for nonlinear model

% Residual Plots for Both Models
figure;
subplot(1, 2, 1);
scatter(y_pred, residuals_linear);
hold on;
plot([min(y_pred), max(y_pred)], [0, 0], 'r--'); % Reference line at zero
xlabel('Predicted Compressive Strength (MPa)');
ylabel('Residuals');
title('Residual Plot (Linear Model)');
grid on;

subplot(1, 2, 2);
scatter(y_pred_nl, residuals_nonlinear);
hold on;
```

```
plot([min(y_pred_nl), max(y_pred_nl)], [0, 0], 'r--'); % Reference line at zero
xlabel('Predicted Compressive Strength (MPa)');
ylabel('Residuals');
title('Residual Plot (Nonlinear Model)');
grid on;
saveas(gcf, 'residual_plots_both_models.jpg');

%% Task 8: Model Performance Discussion
% Compare R² scores, RMSE, and MAE for both models
fprintf('Linear Model Performance:\n');
fprintf('R² Score: %.4f\n', r2_score_linear);
fprintf('RMSE: %.4f\n', rmse);
fprintf('MAE: %.4f\n\n', mae);

fprintf('Nonlinear Model Performance:\n');
fprintf('R² Score: %.4f\n', r2_score_nl);
fprintf('RMSE: %.4f\n', rmse_nl);
fprintf('MAE: %.4f\n\n', mae_nl);

% Determine which model performs better
if r2_score_nl > r2_score_linear
    fprintf('The nonlinear model performs better based on R² score.\n');
else
    fprintf('The linear model performs better based on R² score.\n');
end

if rmse_nl < rmse
    fprintf('The nonlinear model performs better based on RMSE.\n');
else
    fprintf('The linear model performs better based on RMSE.\n');
end

if mae_nl < mae
    fprintf('The nonlinear model performs better based on MAE.\n');
else
    fprintf('The linear model performs better based on MAE.\n');
```

```
end

%% Save the linear and nonlinear models
% Save the linear model to a .mat file
save('linear_model.mat', 'linear_model');

% Save the nonlinear model to a .mat file
save('nonlinear_model.mat', 'nonlinear_model');

% Display the coefficients of the nonlinear model
disp('Nonlinear Model Coefficients:');
disp(nonlinear_model.Coefficients.Estimate);
```

4.1.2. Part B.m:

Code for the prediction function and testing.

```
%% Function for Prediction of Strength
function predicted_strength = predict_concrete_strength(cement, slag, fly_ash, water, superplasticizer,
coarse_agg, fine_agg, age)
    % Load the saved nonlinear model
    % The nonlinear model is loaded from the file 'nonlinear_model.mat'
    % This file contains the trained nonlinear regression model
    load('nonlinear_model.mat', 'nonlinear_model');

    % Create an input array for the new data
    % The input data is organized in the same order as the training data:
    % [cement, slag, fly_ash, water, superplasticizer, coarse_agg, fine_agg, age]
    input_data = [cement, slag, fly_ash, water, superplasticizer, coarse_agg, fine_agg, age];

    % Predict the compressive strength using the nonlinear model
    % The 'predict' function uses the loaded model to predict the strength
    % based on the input data
    predicted_strength = predict(nonlinear_model, input_data);

    % Display a message indicating that the nonlinear model is being used
    disp('Using Nonlinear Model for Prediction');
```

```
end

%% Input for the Prediction using the best model.
% Define the input values for a new concrete mix
% These values represent the composition and age of the concrete sample
cement = 300;          % Amount of cement (kg/m³)
slag = 100;            % Amount of blast furnace slag (kg/m³)
fly_ash = 50;          % Amount of fly ash (kg/m³)
water = 150;           % Amount of water (kg/m³)
superplasticizer = 5;  % Amount of superplasticizer (kg/m³)
coarse_agg = 900;      % Amount of coarse aggregate (kg/m³)
fine_agg = 700;        % Amount of fine aggregate (kg/m³)
age = 28;              % Age of the concrete (days)

% Call the function to predict compressive strength
% The function 'predict_concrete_strength' is called with the input values
predicted_strength = predict_concrete_strength(cement, slag, fly_ash, water, superplasticizer, coarse_agg,
fine_agg, age);

% Display the result
% The predicted compressive strength is printed to the command window
fprintf('Predicted Compressive Strength: %.2f MPa\n', predicted_strength);
```

4.2. Plots

The following plots were generated and saved as JPG files:

- **scatter_plots_with_stats.jpg**: Scatter plots of variables vs. compressive strength.

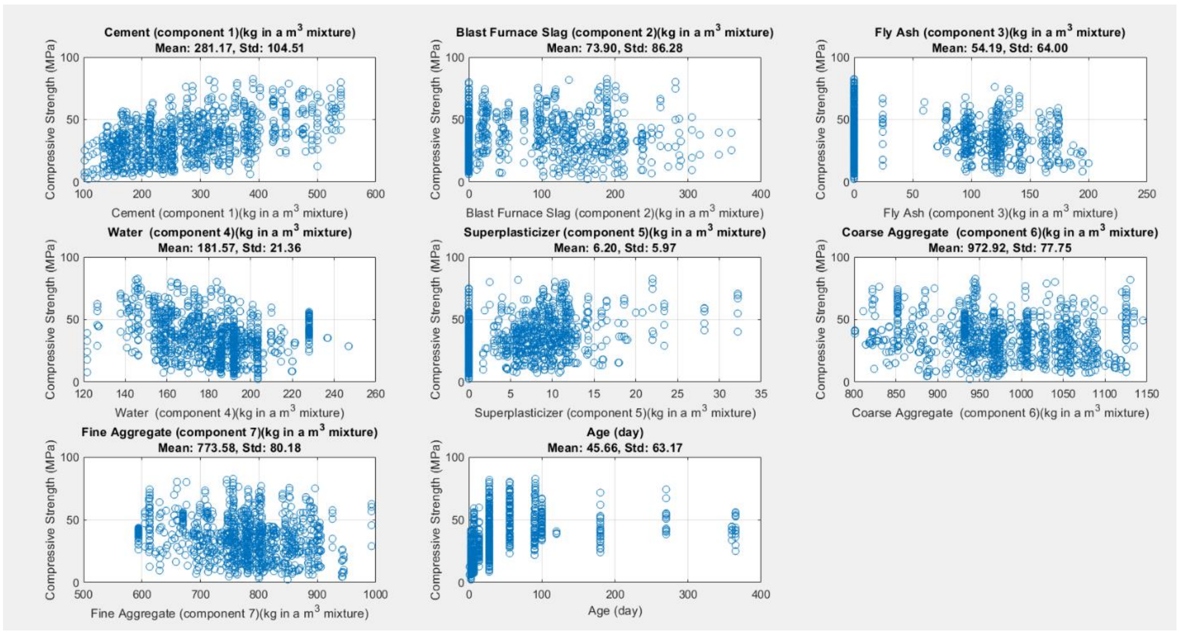

Figure 2 Scatter plots of variables vs. compressive strength.

- **scatter_plots_for_ratios.jpg:** Scatter plots of ratios vs. compressive strength.

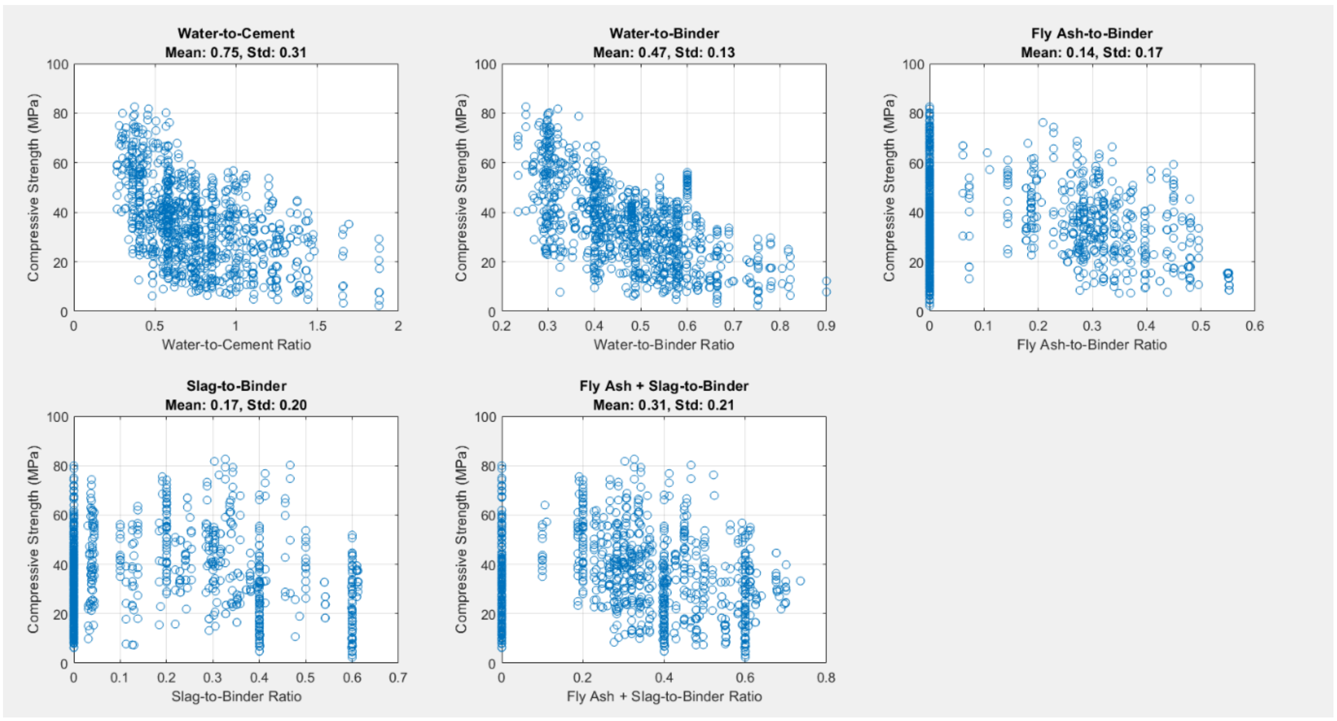

Figure 3 Scatter plots of ratios vs. compressive strength

- **3D Scatter Plot:** Cement vs Blast Furnace Slag vs Compressive Strength

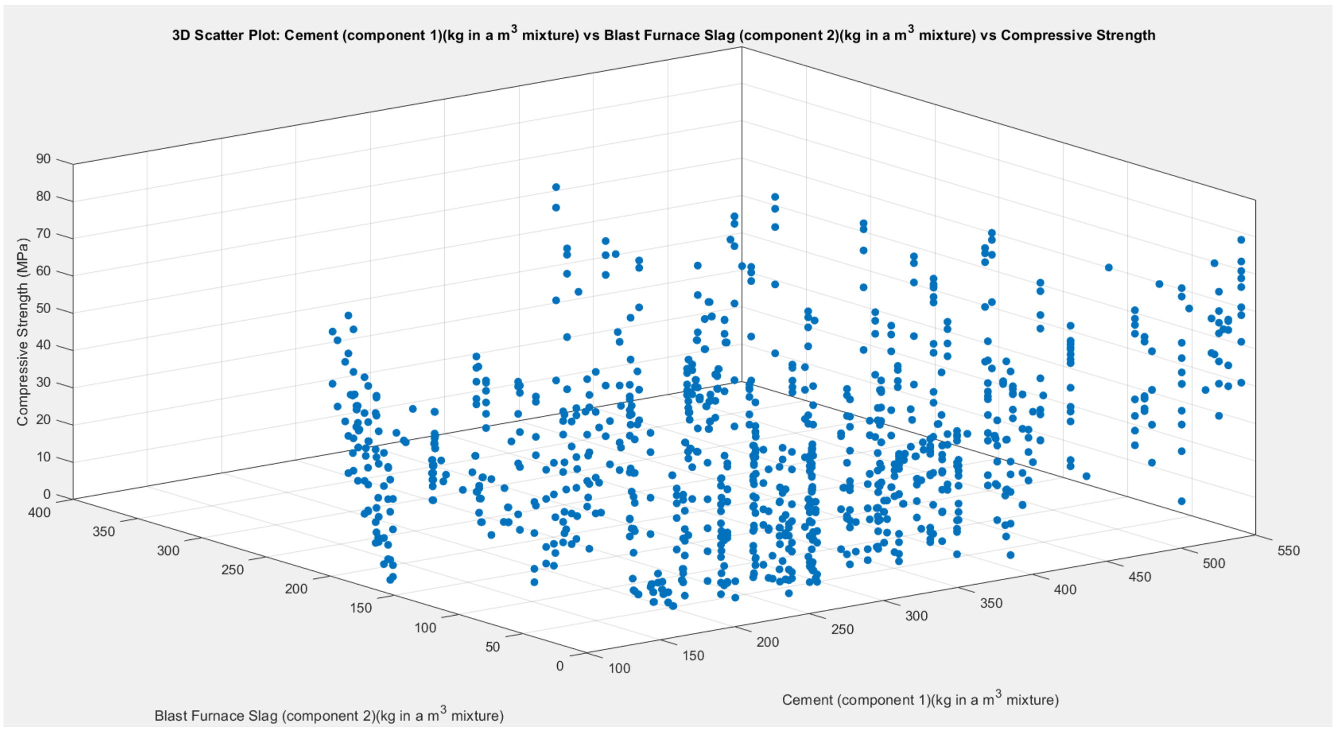

Figure 4 Cement vs Blast Furnace Slag vs Compressive Strength

- **Predicted vs. Actual Compressive Strength:** (Both Models)

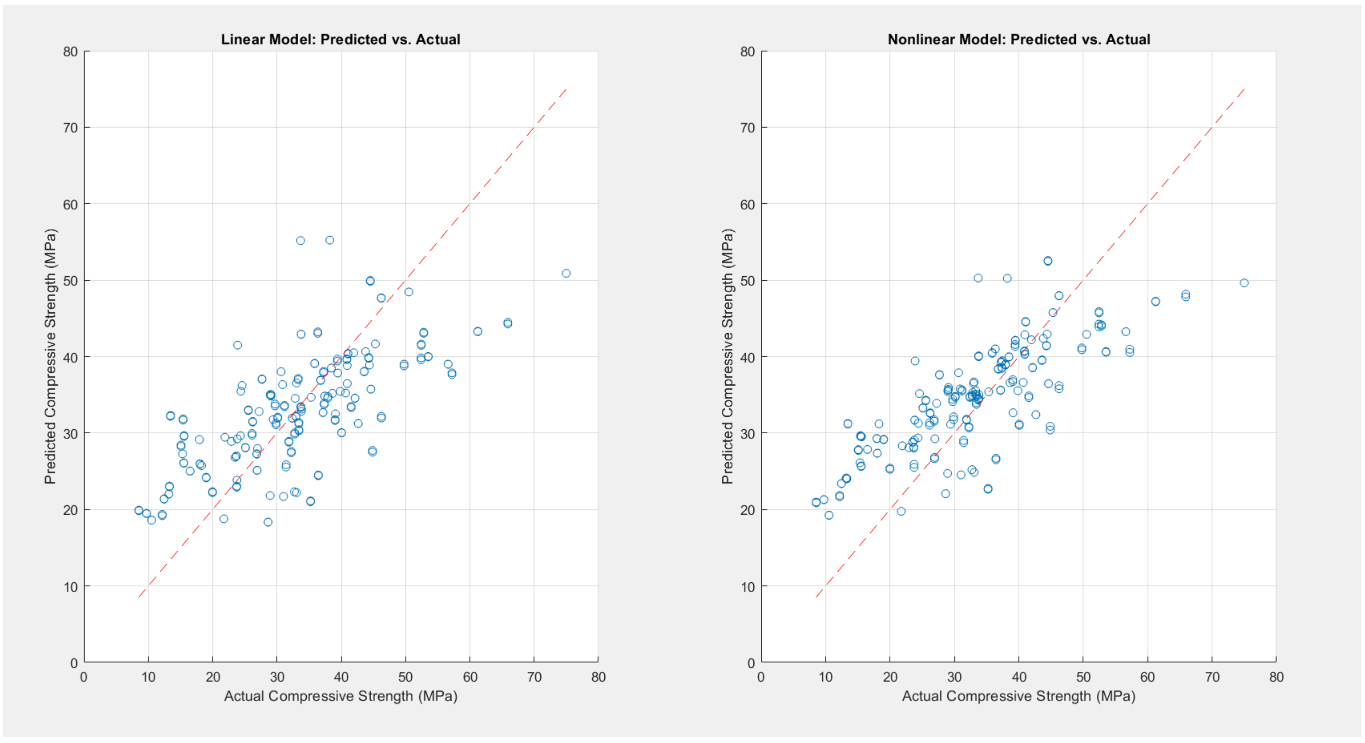

Figure 5 Predicted vs Actual Compressive Strength

- **Residual_plots_both_models.jpg**: Residual plots for linear and nonlinear models.

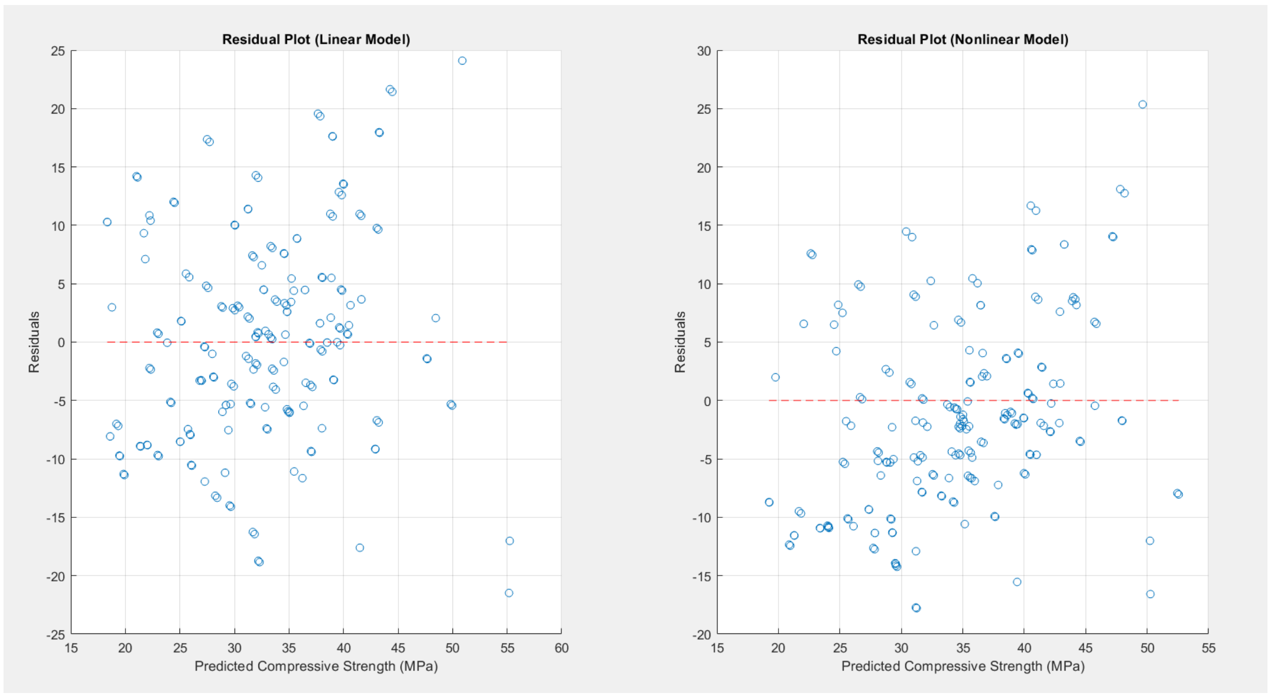

Figure 6 Residual plots for linear and nonlinear models

REFERENCES

- Zhang, L., & Li, L. (2019). A review on the compressive strength of concrete: Effect of mix proportions, admixtures, and curing conditions. *Construction and Building Materials, 200*, 497-506. https://doi.org/10.1016/j.conbuildmat.2018.12.182
- Mehta, P. K., & Monteiro, P. J. M. (2014). *Concrete: Microstructure, properties, and materials* (4th ed.). McGraw-Hill Education.
- Meyer, C., & Cordon, J. (2013). The sustainability of concrete in civil engineering. *Environmental Engineering Science, 30*(7), 375-387. https://doi.org/10.1089/ees.2013.0017
- Jolliffe, I. T. (2002). *Principal component analysis* (2nd ed.). Springer.
- Ochoa, L. M., et al. (2019). Predicting concrete compressive strength using data mining and regression models. *Journal of Construction and Building Materials, 225*, 93-102. https://doi.org/10.1016/j.conbuildmat.2019.07.033
- Chien, H. M., & Hsu, S. C. (2015). Prediction of concrete compressive strength using support vector machines. *International Journal of Computational Engineering & Management, 18*(6), 51-60. http://www.ijcem.com
- Liu, L., & Wang, H. (2018). Application of artificial intelligence in concrete engineering. *Engineering Applications of Artificial Intelligence, 72*, 120-129. https://doi.org/10.1016/j.engappai.2018.03.015
- UCI Machine Learning Repository. (n.d.). *Concrete compressive strength dataset*. Retrieved from https://archive.ics.uci.edu/ml/datasets/Concrete+Compressive+Strength
- MATLAB Documentation. (n.d.). *fitlm, fitnlm*. Retrieved from https://www.mathworks.com/help/stats/fitlm.html