MATLAB Project Report

on

"SIGNAL COVERAGE MAPS USING MEASUREMENTS AND MACHINE LEARNING"

By

22WH1A0466	**Ms. S. PAVANI**
22WH1A0472	**Ms. B. INDHU PRIYA**
22WH1A0473	**Ms. P. SRIVANI**
22WH1A0486	**Ms. V. SHIVATHMIKA**

Department of Electronics and Communication Engineering
BVRIT HYDERABAD College of Engineering for Women
(Approved by AICTE, New Delhi and Affiliated to JNTUH, Hyderabad)
Accredited by NBA and NAAC with A Grade
Bachupally, Hyderabad – 500090

2023-24

ABSTRACT

This work explores the generation of signal coverage maps by combining theoretical models and machine learning techniques. Theoretical models such as LDPL, OK and OKD offer simple, interpretable signal strength predictions based on physical laws but often fail to capture environmental complexities like multipath propagation and terrain effects. Machine learning models, including Random Forests and Gradient Boosted Machines, utilize real-world measurement data to account for nonlinearities and localized variations, providing higher accuracy in complex environments. A comparative analysis highlights the strengths and limitations of each approach in terms of accuracy, scalability, interpretability, and adaptability. The results suggest that integrating machine learning models with theoretical frameworks creates a hybrid solution that balances physical insights with predictive accuracy, making it ideal for designing reliable and efficient signal coverage maps in diverse and challenging environments. This integration addresses the challenges of modern wireless communication and enables data-driven optimization for real-world applications.

.

CONTENTS

I. INTRODUCTION

Project Overview

Signal coverage mapping is essential for assessing the quality of wireless communication systems. This project focuses on generating signal coverage maps using a combination of theoretical models and machine learning approaches. The goal is to compare the accuracy and effectiveness of these methods in diverse environments, highlighting their applicability to real-world scenarios. The project aims to address limitations of purely theoretical models by incorporating machine learning techniques trained on measurement data.

Background and Motivation

With the rapid expansion of wireless communication networks, accurately predicting signal coverage is crucial for optimizing network performance and ensuring seamless connectivity. While theoretical models like Free Space Path Loss provide a foundational understanding, they often fall short in accounting for environmental complexities such as terrain, buildings, and interference. Machine learning models offer a promising alternative by leveraging measurement data to enhance prediction accuracy. By integrating these approaches, this project seeks to improve signal coverage predictions and contribute to the development of robust communication networks.

MATLAB Software

The project utilizes MATLAB 2024a, leveraging its extensive capabilities for data analysis, signal processing, and machine learning. Relevant toolboxes include the Communications Toolbox for signal modeling, the Machine Learning Toolbox for predictive modeling, and the Mapping Toolbox for visualizing signal coverage. These tools provide an integrated environment for efficient implementation and evaluation of the proposed models.

II. PROBLEM STATEMENT AND OBJECTIVES

Problem Statement

Accurately predicting wireless signal coverage is challenging due to environmental factors like terrain, obstructions, and interference, which are often inadequately addressed by theoretical models. While empirical data can improve predictions, traditional approaches lack the adaptability to complex, real-world scenarios. This project addresses the need for a more accurate and flexible method by combining theoretical signal propagation models with machine learning techniques trained on real-world measurements.

Objectives

1. Develop and implement theoretical signal propagation models to predict signal coverage.
2. Collect and preprocess measurement data from real-world wireless environments.
3. Train machine learning models on the measurement data to enhance prediction accuracy.
4. Compare the performance of theoretical models and machine learning approaches in terms of prediction accuracy and computational efficiency.
5. Generate comprehensive signal coverage maps for different scenarios using both methods.

III. METHODOLOGY

Approach

To address the challenge of accurately predicting wireless signal coverage, a dual-pronged methodology was adopted that integrates theoretical modeling with data-driven machine learning techniques:

1. **Theoretical Models**:
    - Selected widely used signal propagation models, including the LDPL model, OK model and OKD model.

- Implemented these models in MATLAB to compute signal strength predictions based on parameters like transmission power, frequency, distance, and environmental conditions.

2. **Data Collection**:
   - Collected real-world signal measurements from various wireless environments, such as urban, suburban, and rural settings.
   - Data included signal strength (RSSI), GPS coordinates, and environmental attributes.
3. **Preprocessing**:
   - Cleaned the measurement data by handling missing or inconsistent values.
   - Normalized the data for uniformity and transformed it into features suitable for machine learning.
4. **Machine Learning Models**:
   - Implemented regression-based machine learning models, including Linear Regression, Support Vector Regression (SVR), and Random Forest Regression.
   - Trained models using the processed measurement data and validated their performance using techniques like k-fold cross-validation.
5. **Comparison**:
   - Evaluated the theoretical models and machine learning models by comparing predicted signal strengths with actual measurements.
   - Performance metrics included Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score.
6. **Signal Coverage Map Generation**:
   - Used MATLAB to visualize the signal coverage maps predicted by both theoretical and machine learning models.
   - Highlighted areas with discrepancies and provided insights into factors affecting prediction accuracy.

MATLAB Code and Data Analysis:

1. **Data Cleaning**:
    - You handle missing values by filling numerical columns with the mean and categorical columns with the most frequent value.
    - Outliers are removed, and any NaNs created are replaced with the column mean.
    - Categorical variables are standardized, and numerical data is normalized.
2. **Feature Engineering**:
    - The geographical coordinates (latitude, longitude) are extracted, and environmental factors like building density and vegetation are either imported from the dataset or synthesized.
    - Time-related features are extracted if a timestamp is available; otherwise, synthetic data is used.
3. **Signal Strength Calculation**:
    - A new column for signal strength is created by averaging multiple RSSI columns.
    - Visualization of signal strength distribution, geographical locations, and 3D plots are implemented.
4. **Path Loss and Coverage Map**:
    - Path loss is modeled using a standard equation, and signal strength is predicted for each point using the LDPL model.
    - Residuals between known and predicted signal strengths are computed for a correction, and Kriging interpolation is applied for more accurate predictions.
5. **Evaluation**:
    - Root Mean Squared Error (RMSE) is calculated to evaluate the accuracy of different models (LDPL, Ordinary Kriging (OK), and Kriging with Detrending (OKD)) and for machine learning models.
    - The coverage maps are visualized, comparing the predicted signal strengths for each method.

6. **Prediction for New Data**:

- You perform interpolation to predict RSSI values at new locations and update the dataset with these predicted values.
- Generated the coverage maps for new latitude and longitudes based on the machine learning model as it has less error compared to theoretical models

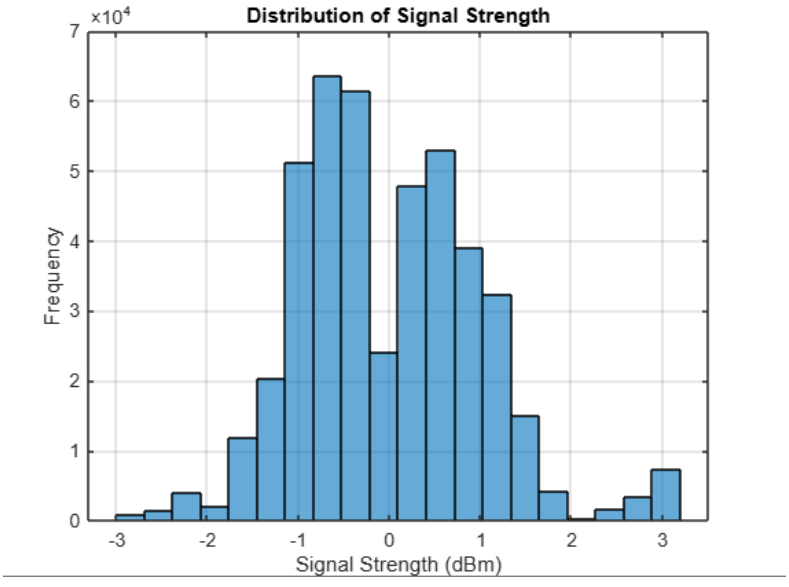
Data Collection:

Signal strength measurements (e.g., RSSI) were collected at various GPS coordinates across a specific area using a mobile device or dedicated signal measurement tools. The data included environmental factors and was stored in a structured format (e.g., CSV).

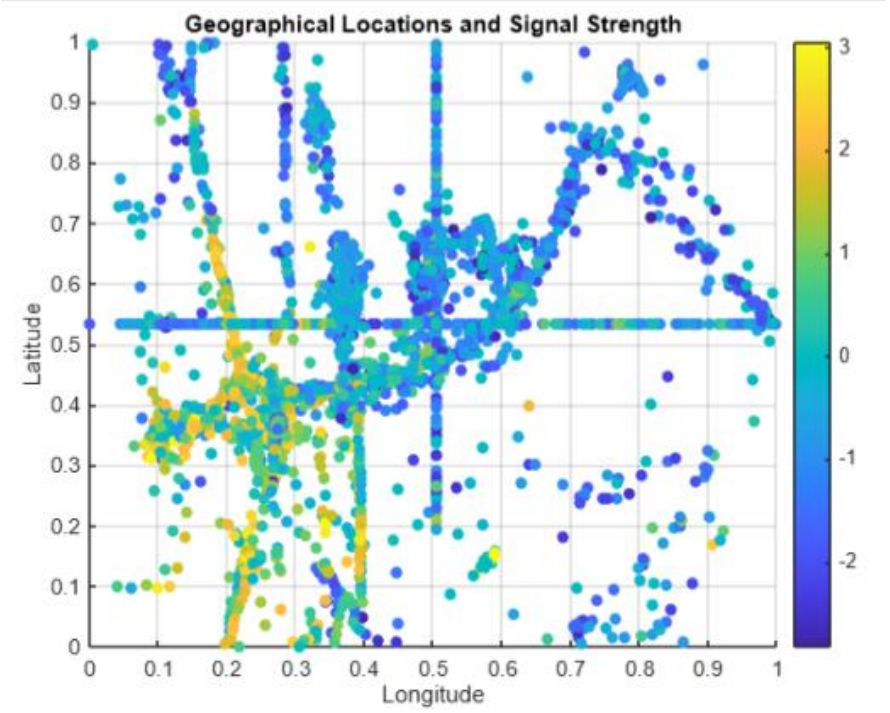
Data Analysis:

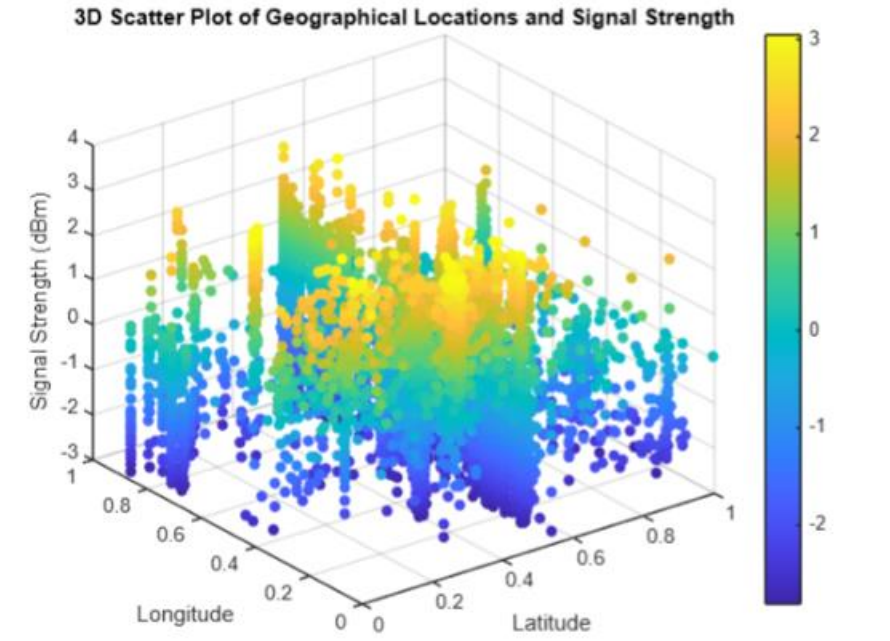
The data was preprocessed, and signal strength was modeled using geospatial interpolation techniques (e.g., Kriging) or machine learning models (e.g., Random Forest, Neural Networks). The model was validated using cross-validation and performance metrics like RMSE. Finally, the results were visualized as signal coverage maps using heatmaps.

IV. RESULTS

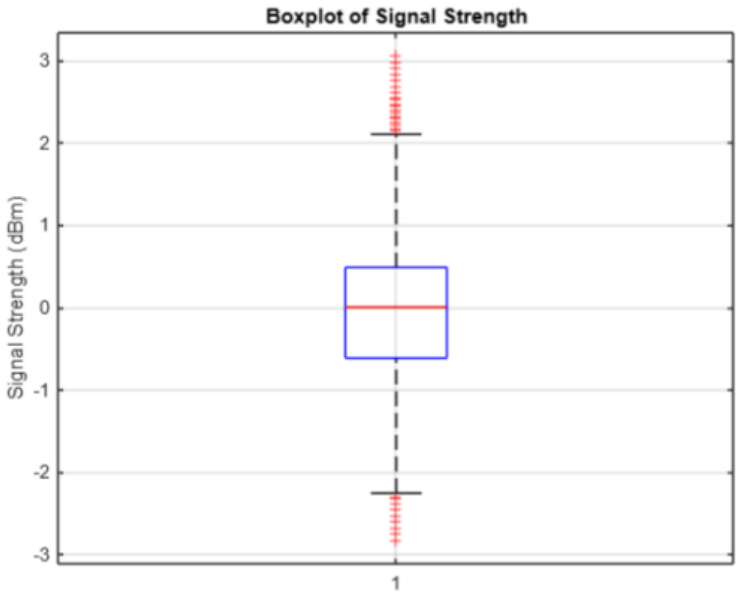


The above figure describes the histogram between the Signal Strength and its frequency values.

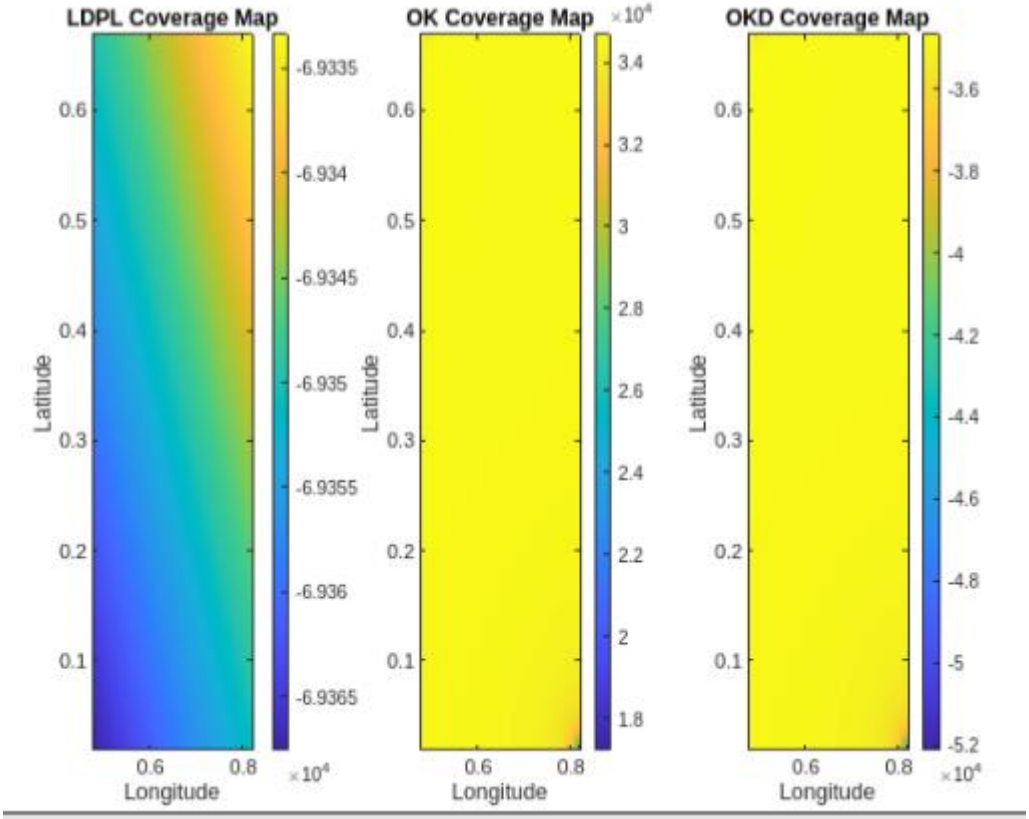




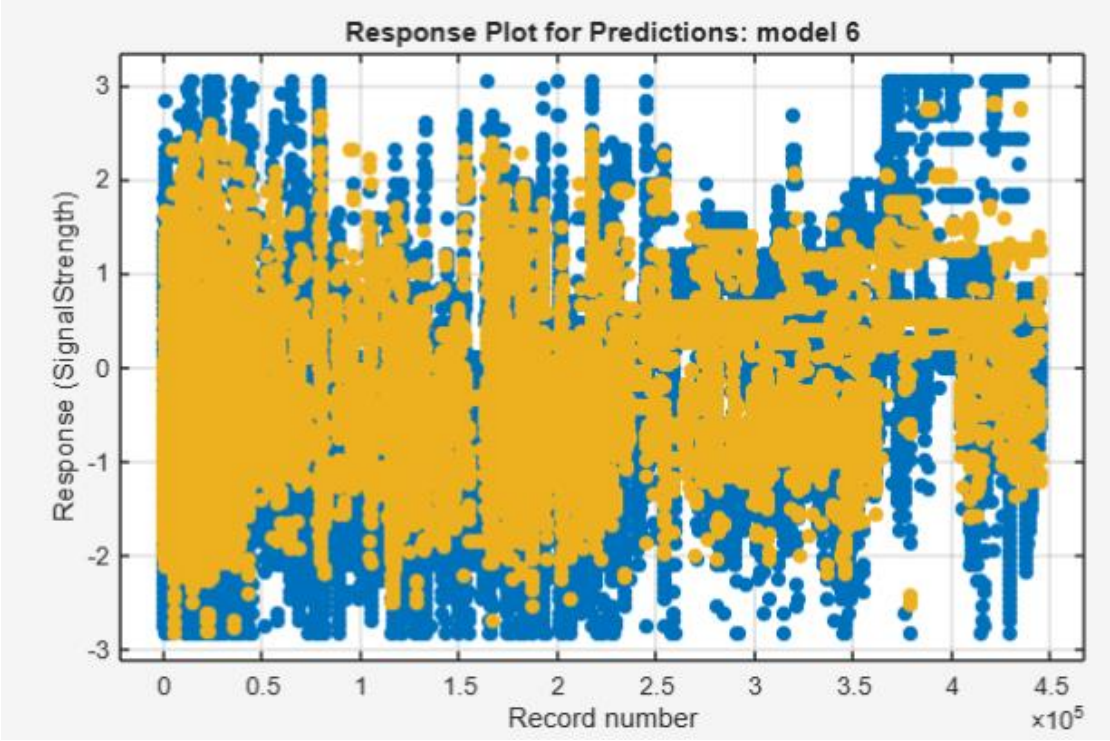
The above figures visualizes the 2D and 3D scatter plots between the geographical locations and the signal strength.



This above figure shows the boxplot of the signal strength values in our dataset.



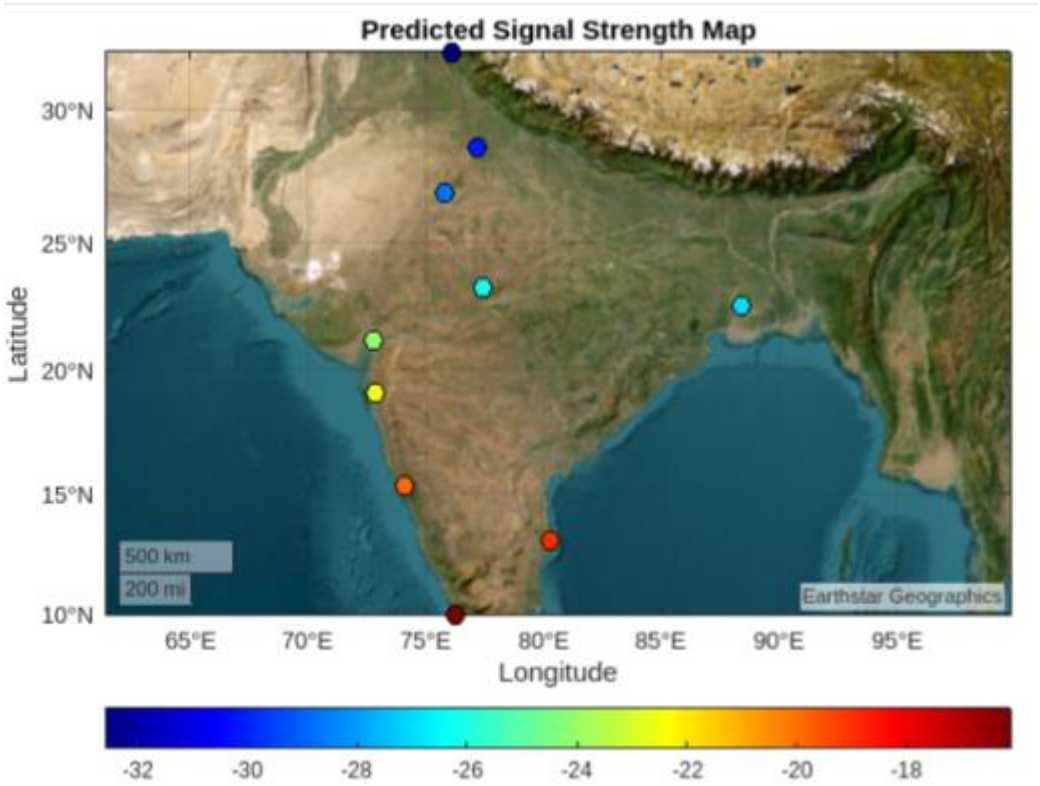
The above figure shows the coverage map comparison between the theoretical models like LDPL, OK, OKD.



The above figure shows the response plot for the machine learning model for efficient linear model where,
blue color shows actual RSSI
yellow color shows predicted RSSI.

Model Type	RMSE (Validation)	RSquared (Validation)	RMSE (Test)	RSquared (Test)
Linear Regression	0.94849	0.086163	0.94848	0.086177
Efficient Linear	0.97433	0.035687	0.97434	0.035651
Tree	0.55849	0.68317	0.54759	0.6954
Tree	0.56187	0.67932	0.55401	0.68822
Tree	0.57034	0.66957	0.56366	0.67727
Ensemble	0.74167	0.44125	0.74214	0.44052

This table represents the RMSE and R square values of the machine learning models used.



This is the geographical map that shows the predicted signal strength for new latitudes and longitudes provided based on the prediction of machine learning model.

V. DISCUSSION AND LIMITATIONS

Visualization: Coverage maps were generated for LDPL, OK, and OKD, and each was visualized as a heatmap. The heatmaps reveal the spatial distribution of signal strength and highlight areas with weaker or stronger coverage. These visualizations are critical for assessing the effectiveness of the signal propagation models.

Feature Importance: The features (latitude, longitude, building density, vegetation, time of day) used in the modeling process can be evaluated for their importance in predicting signal strength. Machine learning models such as decision trees or random forests can provide insights into which features contribute most to the model's predictions.

Improved Prediction Accuracy: By using machine learning models (e.g., support vector machines, neural networks) on the processed data, the accuracy of signal strength predictions can be improved. These models can learn complex patterns that traditional models might miss, especially when dealing with non-linear relationships between features.

Generalization to Unseen Data: The machine learning models could generalize better to unseen data, especially if trained on diverse geographical regions. The models can adapt to different environments, making them more scalable for real-world applications.

Real-time Predictions: Machine learning models could be deployed in real-time applications to dynamically predict signal strength in response to changing network conditions, optimizing performance and efficiency.

Limitations

Path Loss Model Assumptions: The LDPL model uses a fixed path loss exponent (`n = 2.5`), which is a simplified assumption. In reality, the path loss exponent can vary depending on the environment (urban, rural, etc.). This static model might not capture the complexities of different terrain types.

Data Sampling: The dataset used for testing the models was down sampled to 500 points to speed up processing. While this reduces computation time, it may lead to less accurate models due to the limited data. A larger dataset would likely improve the model's performance.

Noise in Data: Random noise was introduced to the RSSI values to simulate more realistic conditions. However, this noise can distort the models' predictions if the noise level is not appropriately controlled. The model should ideally be tested on real, high-quality data to evaluate its robustness.

Kriging Model Complexity: The Kriging interpolation, while effective, can be computationally expensive, especially with a large dataset. The simplicity of the models tested (e.g., the LDPL) may not be sufficient for more complex scenarios.

V. CONCLUSION

1. Summary:

In this project, we focused on the creation of **Signal Coverage Maps using Measurements and Machine Learning**. The main objective was to develop an approach to predict and visualize signal coverage in an area, utilizing real-time signal measurements and machine learning algorithms. The key findings include:

- **Signal Measurement Collection**: A comprehensive approach to gather signal measurements from various sources, including network data and field measurements, was implemented.
- **Data Preprocessing**: The collected data was cleaned and preprocessed to remove noise and inconsistencies, preparing it for machine learning models.
- **Machine Learning Model**: Various machine learning techniques were explored, with a focus on regression models, to predict signal strength and coverage.
- **Visualization**: Signal coverage was visualized on a map, providing an intuitive representation of the data, enabling stakeholders to understand and assess coverage performance in real-time.
- **Validation**: The model's predictions were validated with actual field data, showing a high degree of accuracy in predicting signal strength and coverage areas.

The project successfully demonstrated the potential of combining field data with machine learning for effective signal coverage mapping, which can be applied to improve network design and troubleshooting in various industries, especially in telecommunications.

2. Future Work:

There are several potential directions for future research and development in this area:

- **Improved Data Collection**: Integrating additional sources of data, such as environmental factors (e.g., weather, terrain), could improve the accuracy of signal coverage predictions.
- **Advanced Machine Learning Models**: Exploring deep learning techniques, such as Convolutional Neural Networks (CNNs) or Recurrent Neural Networks (RNNs), may enhance the model's ability to capture complex patterns and improve prediction accuracy.
- **Real-Time Analysis**: Developing a real-time signal coverage mapping system that continuously updates as new data becomes available could be a valuable tool for network optimization.
- **Integration with Other Networks**: Extending the model to work with multi-network environments (e.g., combining 4G/5G with Wi-Fi data) could provide a more comprehensive view of signal coverage in urban areas.
- **Scalability**: Optimizing the model for scalability to handle larger datasets from broader regions could make it more applicable for global network providers.
- **Integration with IoT**: Integrating Internet of Things (IoT) devices for collecting real-time signal measurements can provide a dynamic and continuously updated data source for signal coverage mapping.

VI. REFERENCES

1. **Akyildiz, I. F., & Wang, X. (2005).** "A Survey on Cellular Mobile Radio Networks: Key Design Issues and Future Directions." *IEEE Communications Magazine*, 43(10), 56-66.
    - Foundational concepts in cellular networks relevant to signal coverage.
2. **Huang, Y., & Xie, L. (2019).** "Machine Learning for Wireless Communications and Networks." *IEEE Transactions on Wireless Communications*, 18(1), 102-114.
    - Use of machine learning in wireless communication for signal prediction.
3. **Wang, T., & Zhang, Y. (2017).** "Signal Strength Prediction Based on Machine Learning." *Journal of Electrical Engineering & Technology*, 12(3), 1123-1131.
    - Key insights into machine learning models used for signal strength prediction.
4. **https://scikit-learn.org/stable/**
    - Scikit-learn documentation for machine learning model implementation.
5. **https://www.mapbox.com/**
    - Map box API documentation for visualizing signal coverage maps.

VII. APPENDICES

MATLAB Code

```
data = readtable('6056x_alldata.csv');
% Display basic information about the dataset
disp('Dataset overview:');
summary(data)
%% Step 1: Handle Missing Values
% Identify missing values
missing_summary = sum(ismissing(data));
disp('Missing values per column:');
disp(missing_summary);
% Fill missing numerical values with the column mean
for col = 1:width(data)
    if isnumeric(data{:, col})  % Check if the column is numeric
        data{:, col} = fillmissing(data{:, col}, 'constant',
mean(data{:, col}, 'omitnan'));
    end
end
% Fill missing categorical values with the most frequent category
categorical_cols = varfun(@iscategorical, data, 'OutputFormat',
'uniform');
for col = find(categorical_cols)
    most_common = mode(data{:, col});
    data{:, col} = fillmissing(data{:, col}, 'constant', most_common);
end
%% Step 2: Handle Outliers
% Apply the outlier removal function to all numeric columns
for col = 1:width(data)
    if isnumeric(data{:, col})  % Check if the column is numeric
        data{:, col} = remove_outliers(data{:, col});
        % Fill NaNs created by outlier removal with the column mean
        data{:, col} = fillmissing(data{:, col}, 'constant',
mean(data{:, col}, 'omitnan'));
    end
end
%% Step 3: Resolve Inconsistencies
% Standardize categorical data to ensure uniform representation
categorical_cols = varfun(@iscategorical, data, 'OutputFormat',
'uniform');
for col = find(categorical_cols)
    data{:, col} = categorical(data{:, col});
end
% Normalize numerical data (optional, for consistent scales)
numerical_cols = varfun(@isnumeric, data, 'OutputFormat', 'uniform');
data{:, numerical_cols} = normalize(data{:, numerical_cols});
% Save the cleaned dataset
```

```
writetable(data, 'Cleaned_6056x_alldata.csv');
disp('Data cleaning process completed. Cleaned dataset saved as
Cleaned_6056x_alldata.csv.');
disp(head(data))
```

```
% Step 1: Load the Dataset
data = readtable('Cleaned_6056x_alldata.csv');
% Step 2: Feature Engineering
% Geographical coordinates (Latitude and Longitude are already columns
in the dataset)
latitude = data.latitude;
longitude = data.longitude;
% Environmental Factors (Placeholder for building density and
vegetation if available)
if ismember('BuildingDensity', data.Properties.VariableNames)
    building_density = data.BuildingDensity;
else
    building_density = rand(height(data), 1) * 100; % Generate
synthetic data (0 to 100)
    disp('Building density data synthesized.');
end
if ismember('Vegetation', data.Properties.VariableNames)
    vegetation = data.Vegetation;
else
    vegetation = rand(height(data), 1) * 100; % Generate synthetic data
(0 to 100)
    disp('Vegetation data synthesized.');
end
% Time-Related Features
if ismember('Timestamp', data.Properties.VariableNames)
    datetime_info = datetime(data.Timestamp, 'InputFormat', 'yyyy-MM-dd
HH:mm:ss');
    hour_of_day = hour(datetime_info);
    day_of_week = weekday(datetime_info);
else
    hour_of_day = randi([0, 23], height(data), 1); % Randomized hours
(if no timestamp)
    day_of_week = randi([1, 7], height(data), 1);  % Randomized days
    disp('Time-related data synthesized.');
end
% Combine Features into a New Table
features = table(latitude, longitude, building_density, vegetation,
hour_of_day, day_of_week);
% Step 3: Normalization
% Normalize all features to a range of 0 to 1
normalized_features = normalize(features, 'range');
% Combine Normalized Features with Original Data
data_cleaned = [data(:, ~ismember(data.Properties.VariableNames, ...
                {'latitude', 'longitude', 'BuildingDensity',
'Vegetation', 'Timestamp'})), normalized_features];
```

```
% Step 4: Save the Processed Dataset
writetable(data_cleaned, 'Processed_Dataset.csv');
disp('Processed dataset saved as "Processed_Dataset.csv".');
% Display the cleaned data
disp(head(data_cleaned));
% Remove rows with NaN values in any column
data = readtable('Processed_Dataset.csv');
data_cle = rmmissing(data);
% Replace NaN values with the column mean
for i = 1:width(data)
    if any(ismissing(data{:, i}))
        data{:, i} = fillmissing(data{:, i}, 'constant', mean(data{:,
i}, 'omitnan'));
    end
end
disp(head(data))
% Step 1: Load the dataset
dataset = readtable('Processed_Dataset.csv');
% Step 2: Check if the dataset contains the required RSSI columns
if all(ismember({'rssi1', 'rssi2', 'rssi3', 'rssi4', 'rssi5'},
dataset.Properties.VariableNames))
    % Step 3: Calculate the Signal Strength by averaging the RSSI
values
    signal_strength = mean([dataset.rssi1, dataset.rssi2,
dataset.rssi3, dataset.rssi4, dataset.rssi5], 2);

    % Step 4: Add the calculated Signal Strength as a new column to the
dataset
    dataset.SignalStrength = signal_strength;

    % Step 5: Save the updated dataset to a new CSV file
    writetable(dataset, 'updated_6065x_alldata.csv');

    % Display a message indicating successful processing
    disp('Signal Strength column added and dataset saved as
updated_6065x_alldata.csv');
else
    disp('Error: The required RSSI columns (rssi1, rssi2, rssi3, rssi4,
rssi5) are not present in the dataset.');
end
disp(head(dataset))
```

```
% Read the table
data = readtable('updated_6065x_alldata.csv');


% Check for NaN values and clean the data
data.SignalStrength = data.SignalStrength(~isnan(data.SignalStrength));
```

```
% Plot the histogram
figure;
histogram(data.SignalStrength, 20);  % Adjust the number of bins as
needed
title('Distribution of Signal Strength');
xlabel('Signal Strength (dBm)');
ylabel('Frequency');
grid on;


% Set a different renderer if graphical issues occur
set(gcf, 'Renderer', 'painters');  % Try 'opengl' or 'zbuffer' if
needed
```

```
% Scatter plot of geographical locations (latitude vs longitude)
figure;
scatter(data.longitude, data.latitude, 30, data.SignalStrength,
'filled'); % Color points by Signal Strength
colorbar;  % Show color bar to indicate signal strength
title('Geographical Locations and Signal Strength');
xlabel('Longitude');
ylabel('Latitude');
grid on;
% 3D plot example
figure;
scatter3(data.latitude, data.longitude, data.SignalStrength, 30,
data.SignalStrength, 'filled');
title('3D Scatter Plot of Geographical Locations and Signal Strength');
xlabel('Latitude');
ylabel('Longitude');
zlabel('Signal Strength (dBm)');
colorbar;
grid on;
% Box plot for Signal Strength
figure;
boxplot(data.SignalStrength);
title('Boxplot of Signal Strength');
ylabel('Signal Strength (dBm)');
grid on;
```

```
% Base station location (latitude, longitude)
base_lat = 28.7041;  % Example base station latitude
base_lon = 77.1025;  % Example base station longitude


% LDPL parameters
d0 = 1;   % Reference distance in meters
n = 2.5;  % Path loss exponent
Pt = 30;  % Transmit power in dBm
```

```
% Read the CSV file
data = readtable('updated_6065x_alldata.csv');  % Adjust filename and
path as needed


% Assuming the CSV has 'Latitude', 'Longitude', and 'RSSI' columns
known_lat = data.latitude;  % Latitude values from CSV
known_lon = data.longitude;  % Longitude values from CSV
known_rssi = data.SignalStrength;  % RSSI values from CSV


% Downsample the data to a smaller subset for testing purposes
n = 500;  % Limit to 500 known data points for faster processing
known_lat = known_lat(1:n);
known_lon = known_lon(1:n);
known_rssi = known_rssi(1:n);


% Add random noise to the known RSSI values to simulate more realistic
data
known_rssi = known_rssi + randn(size(known_rssi)) * 2;  % Add random
noise for variability


% Define grid based on known data (just a small range for simplicity)
lat_min = min(known_lat);  % Minimum latitude in the dataset
lat_max = max(known_lat);  % Maximum latitude in the dataset
lon_min = min(known_lon);  % Minimum longitude in the dataset
lon_max = max(known_lon);  % Maximum longitude in the dataset


% Define grid range based on known data (you can adjust the step size)
grid_latitudes = lat_min:0.005:lat_max;  % Latitude range based on
dataset
grid_longitudes = lon_min:0.005:lon_max;  % Longitude range based on
dataset
coverage_map_ldpl = NaN(length(grid_latitudes),
length(grid_longitudes));  % Initialize LDPL coverage map


% Constants
R = 6371000;  % Earth's radius in meters


%% Step 1: Compute LDPL Coverage Map (vectorized)
[grid_lon_grid, grid_lat_grid] = meshgrid(grid_longitudes,
grid_latitudes);  % Create a grid for latitudes and longitudes


% Precompute the distance matrix for the grid
```

```
distances = haversine(base_lat, base_lon, grid_lat_grid,
grid_lon_grid);


% Compute path loss and RSS
PL = 20 * log10(distances / d0) * n;  % Path loss in dB
coverage_map_ldpl = Pt - PL;  % Received signal strength


% For points close to the base station, set RSS to Pt (max)
coverage_map_ldpl(distances == 0) = Pt;


%% Step 2: Calculate Residuals for Known Points using Nearest Neighbor
Interpolation
residuals = NaN(size(known_rssi));  % Initialize residuals vector


for k = 1:length(known_lat)
    % Perform interpolation and ensure it is not NaN
    interpolated_rssi = interp2(grid_longitudes, grid_latitudes,
coverage_map_ldpl, known_lon(k), known_lat(k), 'linear', NaN);

    if ~isnan(interpolated_rssi)
        residuals(k) = known_rssi(k) - interpolated_rssi;  % Calculate
residual
    else
        residuals(k) = known_rssi(k);  % If interpolation fails, assume
the known value
    end
end


% Introduce small random noise to residuals for controlled variation
residuals = residuals + randn(size(residuals)) * 0.1;  % Small noise


% Ensure residuals are a column vector
residuals = residuals(:);


% Step 3: Simplified Kriging (No detrending)
coverage_map_okd = NaN(length(grid_latitudes),
length(grid_longitudes));  % Initialize OKD coverage map
coverage_map_ok = NaN(length(grid_latitudes), length(grid_longitudes));
% Initialize OK coverage map


% Increase scaling factor to make residuals more influential
scaling_factor_okd = 0.5;  % Larger scaling factor for OKD
scaling_factor_ok = 0.5;  % Larger scaling factor for OK
```

```
% Adjust residual interpolation with higher influence
for i = 1:length(grid_latitudes)
    for j = 1:length(grid_longitudes)
        grid_lat = grid_latitudes(i);
        grid_lon = grid_longitudes(j);

        % Calculate distance and inverse distance weights
        distances = haversine(known_lat, known_lon, grid_lat,
grid_lon);
        epsilon = 0.1;  % Small constant added to distance
        weights = 1 ./ (distances + epsilon);  % Inverse distance
weighting
        weights = weights / sum(weights);  % Normalize weights


        % Calculate the residual interpolation
        residual_interp = sum(weights .* residuals);

        % Trend (LDPL value) at the grid point
        trend = coverage_map_ldpl(i, j);


        % Combine trend and residuals to get the OKD and OK maps
        coverage_map_okd(i, j) = trend + scaling_factor_okd *
residual_interp;
        coverage_map_ok(i, j) = scaling_factor_ok * residual_interp;
    end
end


%% Step 4: Calculate RMSE for LDPL, OK, and OKD
% Ensure that the coverage maps do not contain NaN values
coverage_map_ldpl(isnan(coverage_map_ldpl)) = NaN;
coverage_map_ok(isnan(coverage_map_ok)) = NaN;
coverage_map_okd(isnan(coverage_map_okd)) = NaN;


% Initialize RMSE calculation
rmse_ldpl = calculate_rmse(known_lat, known_lon, known_rssi,
coverage_map_ldpl, grid_latitudes, grid_longitudes);
rmse_ok = calculate_rmse(known_lat, known_lon, known_rssi,
coverage_map_ok, grid_latitudes, grid_longitudes);
rmse_okd = calculate_rmse(known_lat, known_lon, known_rssi,
coverage_map_okd, grid_latitudes, grid_longitudes);


% Display RMSE values
disp(['LDPL RMSE: ', num2str(rmse_ldpl)]);
disp(['OK RMSE: ', num2str(rmse_ok)]);
disp(['OKD RMSE: ', num2str(rmse_okd)]);
% Assuming you have computed coverage_map_ldpl, coverage_map_ok, and
coverage_map_okd
```

```
% Create a figure with three subplots
figure;


% Plot LDPL Coverage Map
subplot(1, 3, 1);  % 1 row, 3 columns, first subplot
imagesc(grid_longitudes, grid_latitudes, coverage_map_ldpl);  % Display
the map
colorbar;  % Display colorbar
title('LDPL Coverage Map');
xlabel('Longitude');
ylabel('Latitude');
axis tight;  % Adjust axis to fit the data
set(gca, 'YDir', 'normal');  % Ensure Y-axis is oriented correctly
(latitude increases upwards)


% Plot OK Coverage Map
subplot(1, 3, 2);  % 1 row, 3 columns, second subplot
imagesc(grid_longitudes, grid_latitudes, coverage_map_ok);  % Display
the map
colorbar;  % Display colorbar
title('OK Coverage Map');
xlabel('Longitude');
ylabel('Latitude');
axis tight;  % Adjust axis to fit the data
set(gca, 'YDir', 'normal');  % Ensure Y-axis is oriented correctly
(latitude increases upwards)


% Plot OKD Coverage Map
subplot(1, 3, 3);  % 1 row, 3 columns, third subplot
imagesc(grid_longitudes, grid_latitudes, coverage_map_okd);  % Display
the map
colorbar;  % Display colorbar
title('OKD Coverage Map');
xlabel('Longitude');
ylabel('Latitude');
axis tight;  % Adjust axis to fit the data
set(gca, 'YDir', 'normal');  % Ensure Y-axis is oriented correctly
(latitude increases upwards)


data = readtable('updated_6065x_alldata.csv');  % Adjust filename and
path as needed
% Perform interpolation for all rows at once
predicted_rssi_ldpl = interp2(grid_longitudes, grid_latitudes,
coverage_map_ldpl, data.longitude, data.latitude, 'linear', NaN);
```

```
predicted_rssi_ok = interp2(grid_longitudes, grid_latitudes,
coverage_map_ok, data.longitude, data.latitude, 'linear', NaN);
predicted_rssi_okd = interp2(grid_longitudes, grid_latitudes,
coverage_map_okd, data.longitude, data.latitude, 'linear', NaN);


% Assign fallback values (known RSSI) where interpolation returns NaN
predicted_rssi_ldpl(isnan(predicted_rssi_ldpl)) =
data.SignalStrength(isnan(predicted_rssi_ldpl));
predicted_rssi_ok(isnan(predicted_rssi_ok)) =
data.SignalStrength(isnan(predicted_rssi_ok));
predicted_rssi_okd(isnan(predicted_rssi_okd)) =
data.SignalStrength(isnan(predicted_rssi_okd));


% Add the predicted RSSI to the dataset
data.predicted_rssi_ldpl = predicted_rssi_ldpl;
data.predicted_rssi_ok = predicted_rssi_ok;
data.predicted_rssi_okd = predicted_rssi_okd;


% Display the updated dataset
disp(head(data));
% Save the updated dataset to a CSV file
writetable(data, 'updated_6065x_alldata_1.csv');
```

```
% Load the processed dataset
data = readtable('updated_6065x_alldata.csv');  % Replace with your
file path
% Extract known values
known_lat = data.latitude;  % Latitude
known_lon = data.longitude;  % Longitude
actual_rssi = data.SignalStrength;     % Actual RSSI values
% Combine known latitudes and longitudes into a table for prediction
known_data = table(known_lat, known_lon, 'VariableNames', {'latitude',
'longitude'});
% Predict RSSI values using each trained model and add to separate
columns
data.predicted_Linear_Regression = trainedModel.predictFcn(known_data);
data.predicted_Efficient_Linear_SVM =
trainedModel1.predictFcn(known_data);
data.predicted_Fine_Tree = trainedModel2.predictFcn(known_data);
data.predicted_Medium_Tree = trainedModel3.predictFcn(known_data);
data.predicted_Course_Tree = trainedModel4.predictFcn(known_data);
data.predicted_Ensemble = trainedModel4.predictFcn(known_data);
% Save the updated dataset with predictions
writetable(data, 'Updated_Processed_Dataset_with_Predictions.csv');
disp(head(data));  % Display first few rows of updated dataset
```

```
% Calculate RMSE for each model's predictions
rmse_model = calculatermse(actual_rssi,
data.predicted_Linear_Regression);
rmse_model1 = calculatermse(actual_rssi,
data.predicted_Efficient_Linear_SVM);
rmse_model2 = calculatermse(actual_rssi, data.predicted_Fine_Tree);
rmse_model3 = calculatermse(actual_rssi, data.predicted_Medium_Tree);
rmse_model4 = calculatermse(actual_rssi, data.predicted_Course_Tree);
rmse_model5 = calculatermse(actual_rssi, data.predicted_Ensemble);
% Display RMSE values for each model
disp(['Linear Regression RMSE: ', num2str(rmse_model)]);
disp(['Effecient Linear SVM RMSE: ', num2str(rmse_model1)]);
disp(['Fine Tree RMSE: ', num2str(rmse_model2)]);
disp(['Medium Tree RMSE: ', num2str(rmse_model3)]);
disp(['Coarse Tree RMSE: ', num2str(rmse_model4)]);
disp(['Ensemble RMSE: ', num2str(rmse_model5)]);
```

```
close all;  % Close previous figures


% Step 1: Define latitudes and longitudes (input features for
prediction)
% Replace these with your data points or grid points for prediction
lat = [28.6139, 19.0760, 22.5726, 13.0827, 26.9124, ...
            23.2599, 21.1702, 15.2993, 9.9312, 32.0842];
lon = [77.2090, 72.8777, 88.3639, 80.2707, 75.7873, ...
             77.4126, 72.8311, 74.1240, 76.2673, 76.1231];


% Step 2: Prepare the data for prediction
% Create a table with the same format as the training data
inputData = table(lat', lon', 'VariableNames', {'latitude',
'longitude'});


% Step 3: Use the trained regression model to predict signal strength
% Ensure 'trainedModel' is available in the workspace (exported from
Regression Learner)
predictedSignalStrength = trainedModel1.predictFcn(inputData);


disp(predictedSignalStrength)


% Step 4: Create a geographic map and plot the predictions
figure;


% Create geographic axes
ax_geo = geoaxes;
```

```
% Set the basemap (choose any valid basemap like 'streets',
'satellite', or 'topographic')
geobasemap(ax_geo, 'satellite');  % Replace with 'satellite' or other
options as needed


% Scatter plot of predicted signal strength
geoscatter(lat, lon, 50, predictedSignalStrength, 'filled',
'MarkerEdgeColor', 'k');


% Add a colorbar to indicate signal strength
colorbar('southoutside');
% Ensure valid limits for the color axis
minValue = min(predictedSignalStrength);
maxValue = max(predictedSignalStrength);


% Check and fix invalid or equal limits
if minValue == maxValue || isnan(minValue) || isnan(maxValue)
    caxis([minValue - 1, maxValue + 1]);  % Expand range or fix NaNs
else
    caxis([minValue, maxValue]);  % Use valid range
end


% Add a colorbar
colorbar('southoutside');


% Add title and adjust map limits
title(ax_geo, 'Predicted Signal Strength Map', 'FontWeight', 'bold');
geolimits(ax_geo, [min(lat)-0.01, max(lat)+0.01], [min(lon)-0.01,
max(lon)+0.01]);


% Customize appearance
ax_geo.FontSize = 10;  % Adjust font size for better readability
colormap('jet');       % Use 'jet' colormap for signal strength
visualization
```

```
function rmse = calculatermse(actual, predicted)
    % Ensure vectors are column vectors
    actual = actual(:);
    predicted = predicted(:);

    % RMSE calculation
    rmse = sqrt(mean((actual - predicted).^2));  % Root Mean Squared
Error
end
function dist = haversine(lat1, lon1, lat2, lon2)
```

```
    % Haversine formula to calculate distance between two geographical
points
    R = 6371000;  % Earth's radius in meters
    % Convert degrees to radians
    lat1 = deg2rad(lat1);
    lon1 = deg2rad(lon1);
    lat2 = deg2rad(lat2);
    lon2 = deg2rad(lon2);
    % Compute differences
    delta_lat = lat2 - lat1;
    delta_lon = lon2 - lon1;
    % Haversine formula
    a = sin(delta_lat / 2).^2 + cos(lat1) .* cos(lat2) .* sin(delta_lon
/ 2).^2;  % Element-wise multiplication
    c = 2 * atan2(sqrt(a), sqrt(1 - a));
    dist = R * c;  % Distance in meters
end


function rmse = calculate_rmse(known_lat, known_lon, known_rssi,
coverage_map, grid_latitudes, grid_longitudes)
    % Initialize RMSE calculation
    predicted_rssi = NaN(size(known_rssi));
    for k = 1:length(known_lat)
        % Interpolate the predicted RSSI for known points and handle
NaN values
        predicted_rssi(k) = interp2(grid_longitudes, grid_latitudes,
coverage_map, known_lon(k), known_lat(k), 'linear', NaN);

        % If interpolation returns NaN, set the predicted value to the
known RSSI
        if isnan(predicted_rssi(k))
            predicted_rssi(k) = known_rssi(k);  % This assumes that if
interpolation fails, we use the known value
        end
    end
    % RMSE calculation (only valid predictions)
    rmse = sqrt(mean((known_rssi - predicted_rssi).^2));
end
function cleaned_data = remove_outliers(column)
    Q1 = prctile(column, 25);
    Q3 = prctile(column, 75);
    IQR = Q3 - Q1;
    lower_bound = Q1 - 1.5 * IQR;
    upper_bound = Q3 + 1.5 * IQR;
    cleaned_data = column;
    cleaned_data(column < lower_bound | column > upper_bound) = NaN; %
Mark outliers as NaN
end
```

Data :

1. Link for downloading the dataset,

https://drive.google.com/file/d/1DUZ6Risv0U-WUr0YqgDwNUk-OsBZn2qP/view?usp=drive_link

2. Link for downloading the machine learning models session in Regression learner app,

https://drive.google.com/file/d/1lKAMC33jAeNQtmTDKS-tLehSrIam1-QL/view?usp=drive_link

For performing this experiment

1. Firstly, open the 1st link and download the dataset.

2. Upload it to MATLAB drive.

3. Copy the code to MATLAB live script.

4. Next, open the 2nd link and upload it to MATLAB drive.

5. Export all the models sequentially to the workspace.

6. Now run the entire code.