```matlab
%% MACHINE LEARNING PROJECT
% ALGORITHMS PERFORMING THE PREDICTION OF PRICES OF USED CARS (REGRESSION TASK)

% The two algorithms to be performed and compared are :
 % 1. Random Forrest Regressor
 % 2. Decision Tree for Regression

 %% About the dataset

% The data set contains 17,965 rows with 9 columns. The columns and their descriptions are as follows:

%- model --> Model Name.
%- year --> The year it was bought.
%- price --> The price at which the used car will be sold.
%- transmission --> The transmission type i.e. manual, automatic or semi-automatic.
%- mileage --> The miles that used car has driven.
%- fuelType --> The fuel type of the car i.e. petrol, diesel, hybrid, electric or other
%- tax --> The tax that will be applied on the selling price of that used car.
%- mpg --> The miles per gallon ratio telling us how many mies it can drive per gallon of fuel.
%- engineSize --> The engine size of the used car.

%% Uploading the ford dataset from the 100,000 UK Used Car Data set from kaggle
clc
% reading my ford csv file & converting it into a table
fordcar = readtable("C:\Users\FIDVI FATEMA\OneDrive\Desktop\machine learning\10k used car dataset\ford.csv");
%  displaying the head of the dataset
head(fordcar);

%% EXPLORATORY DATA ANALYSIS

%% 1. Checking Missing Values in my dataset
clc
missing_values = sum(ismissing(fordcar));

% Displaying the number of missing values for each variable
disp('Number of missing values for each variable:');
disp(missing_values);

%there are no missing values in my dataset

%% 2. Checking if there are any duplicates in my dataset
clc
[~, unique_indices, ~] = unique(fordcar);

% Getting the duplicate rows and removing them
duplicate_rows = fordcar(setdiff(1:height(fordcar), unique_indices), :);
fordcar_noduplicates = fordcar(unique_indices, :);
```

```
% There were  154 duplicate rows

% Looking at the dimensions of the new fordcar_noduplicates dataset after the↙
removal of the duplilcates
dimensions = size(fordcar_noduplicates);
disp(['Number of rows: ', num2str(dimensions(1))]);
disp(['Number of columns: ', num2str(dimensions(2))]);

% After the removal of the duplicates, the shape of fordcar_final is (17811 x 9)

%% 3. Descriptive Summary of the variables in my dataset
clc
summary(fordcar_noduplicates)

% By doing this, we can see the minimum, maximum and the median values of
% each of the clumns in the dataset.

%% 4. Visually analyzing the distribution for price column

% Extracting the 'price' column
price = fordcar_noduplicates.price;
% Creating a histogram
figure;
histogram(price, 'BinWidth', 5000);
title('Distribution of Price');
xlabel('Price');
ylabel('Frequency');

% from this distribution graph, it is clear that  the overwhelming majority
% of price falls in the 0-25,000 range.

%% 5. Understanding the distribution of categorical columns
clc
% Extracting the 'fuelType' column
fuelType = fordcar_noduplicates.fuelType;
% Using the tabulate function to get fuelType counts
fuelType_counts = tabulate(fuelType);
% Displaying the results
disp(fuelType_counts);

% Performing the same for 'model' and 'transmission' columns
model = fordcar_noduplicates.model;
model_counts = tabulate(model);
disp(model_counts);

transmission = fordcar_noduplicates.transmission;
transmission_counts = tabulate(transmission);
disp(transmission_counts);

%% 6. Checking if there are outliers using boxplots

```

```
% creating a list for numerical columns
num_list = {'price', 'mileage', 'tax', 'mpg'};

% Creating Boxplots for them
figure('Position', [100, 100, 900, 600]);
sgtitle('Distribution of Outliers', 'FontSize', 16);
rows = 2;
cols = 2;

% Iterating through numerical columns
for k = 1:numel(num_list)
    % Creating subplot
    subplot(rows, cols, k);

    % Plot the boxplot
    boxplot(fordcar_noduplicates.(num_list{k}));
    title(num_list{k}, 'FontSize', 14);
    if k > numel(num_list) - cols
        xlabel('Variable');
    end

    if mod(k, cols) == 1
        ylabel('Value');
    end
end

%Even though there are a lot of outliers seen in the boxplots, removing them would↙
not be the best option.
% For example, for price column, it is very much likely to have varied set of values↙
ranging from a very high to very low values.
% Similarly, we can conclude for tax or mileage or mpg (mileage per gallon). If i↙
remove them, it may break the flow of the dataset.
% Even though some of the data points can be actual outliers, I would conduct my↙
research without removing them.

%% 7. Looking at the correlation between year and the sale price of the used cars

figure;
scatter(fordcar_noduplicates.year, fordcar_noduplicates.price, 'filled');
title('Ford cars sale price by year');
xlabel('Year');
ylabel('Price');

% From this, we can see that, the more recent the car is, the more pricier it is.
% There is one point that we can see as an outlier which shows the price of the car↙
in 2060.
% This seems like an obvious outlier and should be removed from my dataset.

% removing the obvious outlier
logical_index = (fordcar_noduplicates.year ~= 2060);
fordcar_final = fordcar_noduplicates(logical_index, :);
```

```matlab

% Again looking at the distribution after removal of the outlier
figure;
scatter(fordcar_final.year, fordcar_final.price, 'filled');
title('Ford used car sale price by year');
xlabel('Year');
ylabel('Price');
%After removing this outlier, we can see the distribution of the price with respect to years much more clearly.
% In this, we can see that, the more recent the car is i.e, the closer the year of buying the car, the higher is the selling price of the car.
% That is, the lesser the age of the car, the higher is the price.

%% 8. Correlation matrix between my numeric variables

% Extracting the columns of interest
selected_columns_final = fordcar_final(:, {'price', 'year', 'mileage', 'tax', 'mpg', 'engineSize'});

% Converting the table to a matrix
selected_matrix = table2array(selected_columns_final);

% Calculating the correlation matrix
correlation_matrix_final = corrcoef(selected_matrix, 'rows', 'pairwise');

% Creating a new figure and heatmap
figure;
heatmap(selected_columns_final.Properties.VariableNames, selected_columns_final.Properties.VariableNames, correlation_matrix_final, 'Colormap',bone, 'ColorLimits', [-1 1], 'FontSize', 10, 'CellLabelColor','k', 'CellLabelFormat', '%.2f');
title('Correlation Matrix (Pearson) for Numeric Values');
xlabel('Variables');
ylabel('Variables');

% This heatmap tells us a lot about our dataset and how our variables are correlated to each other as well as with our target column i.e
% the price. The highest positive correlation with price column is 0.65 and it is with 'year' column, while the strongest negative
% correlation with price column is -0.53 and it is with 'mileage' column. The correlations of all numeric variables with my
% target column are quite high which is a good thing. The heatmap obtained from this code only has the numerical columns and
% not the categorical columns of 'model', 'fuelType' and 'transmission'as they are not label encoded yet.

%% 9. Distribution of the average price of the cars with the year column
clc
groupedData = groupsummary(fordcar_final, 'year', 'mean', 'price');
disp(groupedData);

%from this, we can see that the prices of the older cars i.e the cars from 1996-1998
```

```matlab
are sold at a higher rates and then there is
%decline in the prices after those years as cars from after 1998 are not considered as antiques but just older and non-functional.
%This might justify why we see a positice correlation between 'price' and 'year' column.

%% 10. Distribution of the models of ford car in the dataset
clc
model_distribution = tabulate(fordcar_final.model);
disp(model_distribution);

%% 11. Correlation between model of the Ford car and the price
clc
% Extracting relevant columns
models = fordcar_final.model;
prices = fordcar_final.price;

% Creating a table with unique models and their corresponding total prices
unique_models = unique(models);
total_prices = zeros(size(unique_models));

for i = 1:length(unique_models)
    total_prices(i) = sum(prices(strcmp(models, unique_models{i})));
end

% Creating a bar plot
figure;
bar(unique_models, total_prices);
title('Sum of Model Prices for Ford Cars');
xlabel('Ford Models');
ylabel('Total Price');
xtickangle(90);

% from this bar graph, it is very clear that there are 3 specific models of the ford company car namely 'Fiesta', 'Focus'
% & 'Kuga' which have a huge sum of prices i.e the sale price of these models might be higher than the others.
% the other reason might be that the numbers of these models in our datset is greater than the other models and so
% the total prices of these models exceed the others. The latter seems more accurate looking at the distribution of the model columns above.

%% Feature Engineering (Label Encoding)

%'Label Encoding is a technique that is used to convert categorical columns into numerical ones so that they can be fitted by machine
% learning models which only takes numerical data.
% for my dataset, we have 3 categorical columns namely 'transmission', 'fuelType' and 'model' and one 'year' column.

%% a) One-Hot Encoding :
```

```
clc
% code help was taken from 'https://uk.mathworks.com/help/deeplearning/ref/onehotencode.html'
% The 'transmission' column has 3 unique features while 'fuelType' column has 5 unique features which are not ordinal.
% So, the decision was made to apply one-hot encoding on these two categorical columns.


transmission = fordcar_final.transmission;
fuelType = fordcar_final.fuelType;

% Converting categorical variables to numerical indices
transmission_indices = grp2idx(categorical(fordcar_final.transmission));
fuelType_indices = grp2idx(categorical(fordcar_final.fuelType));

% Performing one-hot encoding
transmission_encoded = full(sparse(1:numel(transmission_indices), transmission_indices, 1));
fuelType_encoded = full(sparse(1:numel(fuelType_indices), fuelType_indices, 1));

% Concatenating the encoded variables to the original table
fordcar_final_encoded = [fordcar_final, array2table(transmission_encoded, 'VariableNames', strcat('transmission_encoded_', cellstr(unique(fordcar_final.transmission))))];
fordcar_final_encoded = [fordcar_final_encoded, array2table(fuelType_encoded, 'VariableNames', strcat('fuelType_encoded_', cellstr(unique(fordcar_final.fuelType))))];

% Displaying the head of the table
head(fordcar_final_encoded);

%% b) Integer Label Encoding

% This type of label encoding is the basic one as you can just assign an integer to the specific categorical value in the column.
% The disadvantage of using this type of labelEncoder is that the model assumes that the higher integer has more value than the
% lower one which is not true if the categorical variables in our column are not ordinal.
% using this labelEncoding method for my 'model' column is more practical as the model column has 23 unique categorical
% values and using one-hot encoding will just increase the dimensionality and make my dataset more sparsed.

% Using integer label encoding on my 'model' column and assigning higher integer to the more frequent models in my dataset.


% Getting the unique models and their counts
[uniqueModels, ~, modelIdx] = unique(fordcar_final_encoded.model);
% Getting the frequency of each model
```

```
modelCounts = histcounts(modelIdx, 1:numel(uniqueModels)+1);
% Getting the rank of each model based on frequency
[~, rankedModels] = sort(modelCounts, 'descend');
% Assigning the ranked models as 'model_encoded'
fordcar_final_encoded.model_encoded = zeros(size(fordcar_final_encoded, 1), 1);

% Loop through uniqueModels and assigning corresponding rank
for i = 1:numel(uniqueModels)
    modelIndices = ismember(fordcar_final_encoded.model, uniqueModels{i});
    fordcar_final_encoded.model_encoded(modelIndices) = rankedModels(i);
end

% We can later see whether the results of the algorithms vary if we just
% assign the variables in 'model' columns with integers without taking into
% account the frequency of the model in our dataset.

%% c) Ordinal label encoding for my 'year' Column
% code help taken from 'https://uk.mathworks.com/help/matlab/matlab_prog/ordinal-↙
categorical-arrays.html'
% Performing Ordinal encoding for my year column as the recent the year is, the↙
higher the integer value it will be assigned.
fordcar_final_encoded.year = categorical(fordcar_final_encoded.year);
fordcar_final_encoded.year_encoded = grp2idx(fordcar_final_encoded.year);

% Dropping the original categorical columns
categorical_columns_to_remove = {'transmission', 'fuelType','model','year'};
fordcar_final_encoded = removevars(fordcar_final_encoded,↙
categorical_columns_to_remove);

% Displaying the head of the updated table
head(fordcar_final_encoded)

%  Dimensions of the dataset after encoding
dim = size(fordcar_final_encoded);
disp(['Number of rows: ', num2str(dim(1))]);
disp(['Number of columns: ', num2str(dim(2))]);

%%  Splitting the data into training, testing and validation (70:20:10 ratio)

% 70 is the training set ratio
% 20 is the testing set ratio
% 10 is the validation set ratio which we will use for performing Grid
% Search for Hyperparameter tuning for our models
clc
% Converting table to matrix
data = fordcar_final_encoded{:,:};

% Setting the random seed for reproducibility i.e each time we run this code, we get↙
the same results.
rng(43);

```

```
% Create a partition for 70% training and 30% testing
cv = cvpartition(size(data, 1), 'HoldOut', 0.3);

% Training data
training_data = data(cv.training, :);

% Hold-out data (combined testing and validation i.e 20:10)
Holdoutdata = data(cv.test, :);

% Further splitting the hold-out data into testing (20%) and validation (10%)
cv2 = cvpartition(size(Holdoutdata, 1), 'HoldOut', 0.3333);

% Testing data
testing_data = Holdoutdata(cv2.training, :);

% Validation data
validation_data = Holdoutdata(cv2.test, :);

% Display the sizes of the datasets
disp(['Training Data Size: ' num2str(size(training_data, 1))]);      %70
disp(['Testing Data Size: ' num2str(size(testing_data, 1))]);        %20
disp(['Validation Data Size: ' num2str(size(validation_data, 1))]);  %10

%% Extracting the features and target variable for training, testing and validation

% as price is our second column
% Extracting features and target variable from validation_data
X_validation = validation_data(:, (2:end));
y_validation = validation_data(:, 1);

% Extracting features and target variable from training_data
X_train = training_data(:, (2:end));
y_train = training_data(:, 1);

% Extracting features and target variable from testing_data
X_test = testing_data(:, (2:end));
y_test = testing_data(:, 1);

%--------------------------------------------------------------------------
%% RANDOM FORREST REGRESSION ALGORITHM

% Statistics and Machine Learning Toolbox installed

%% 1. Grid Search for finding the best Hyperparameters for Random Forest model

% Code help taken from 'https://uk.mathworks.com/help/stats/regression-tree-↙
ensembles.html'
clc
% setting random seed for reproducibility
rng(43);

```

```matlab
% Defining the range of hyperparameters to search over
numTreesRange = 10:50:300;
maxNumSplitsRange = 10:10:150;

% Initializing variables to store results
bestNumTrees = 0;
bestMaxNumSplits = 0;
rfbestR2 = -Inf;
rfbestMSE = Inf;

% Performing grid search
for numTrees = numTreesRange
    for maxNumSplits = maxNumSplitsRange

        % Creating Random Forest model for regression for grid search
        rfmodel_gridsearch = TreeBagger(numTrees, X_train, y_train, 'Method', 'regression', 'MaxNumSplits', maxNumSplits);

        % Evaluating random forest on the validation set
        y_predrf_validation = predict(rfmodel_gridsearch, X_validation);

        % Calculating R-squared value
        rfR2_validation = corr(y_predrf_validation, y_validation)^2;
        % Calculating Mean Squared Error (MSE)
        rfMSE_validation = mean((y_predrf_validation - y_validation).^2);

 % Checking if the current hyperparameters result in better R-squared value and lower MSE
        if rfR2_validation > rfbestR2 && rfMSE_validation < rfbestMSE
            bestNumTrees = numTrees;
            bestMaxNumSplits = maxNumSplits;
            rfbestR2 = rfR2_validation;
            rfbestMSE = rfMSE_validation;
        end
    end
end

% Displaying the best hyperparameters, R-squared and MSE value
disp(['Best Number of Trees for random forest: ' num2str(bestNumTrees)]);
disp(['Best MaxNumSplits for random forest: ' num2str(bestMaxNumSplits)]);
disp(['Best R-squared Value for random forest after gridsearch: ' num2str(rfbestR2)]);
disp(['Best MSE Value for random forest after gridsearch: '  num2str(rfbestMSE)]);

% The computational time taken for this step is more than anticipated.
% The result of this grid search gives us the hyperparameters to get the best results for our prediction.

%% 2. Performing Random Forest Regression on my training data using the best hyperparameters from GridSearch
clc
```

```
% using 60 no. of decision trees and 150 splits from grid search to get best
% results from training the model on training set.

% setting random seed for reproducibility
rng(43);

% Best hyperparameters from grid search
bestNumTrees = 60;
bestMaxNumSplits = 150;

% Training my random forest model on training set
randomforestmodel = TreeBagger(bestNumTrees, X_train, y_train, 'Method',↙
'regression', 'MaxNumSplits', bestMaxNumSplits, 'OOBPredictorImportance', 'on');

% save the best model for random forest
save("bestmodel for random forest.mat", 'randomforestmodel');

% Making predictions on the training set
y_predrf_train = predict(randomforestmodel, X_train);

% Calculating R-squared value and MSE value for my training model
rfMSE_train = mean((y_train - y_predrf_train).^2);
rfR2_train = 1 - sum((y_train - y_predrf_train).^2) / sum((y_train - mean(y_train)).↙
^2);
disp(['MSE Value for Random Forest Model : ' num2str(rfMSE_train)]);
disp(['R-squared Value for Random Forest Model: ' num2str(rfR2_train)]);

% Creating a table containing original and predicted prices
trainingprediction_rf = table(y_train, y_predrf_train);
trainingprediction_rf.Properties.VariableNames = {'Original Price', 'Predicted↙
Price'};

% Displaying the top 10 rows of the table
display(trainingprediction_rf(1:10, :));

% Using the bestNumTrees as 60 and bestMaxNumSplits as 150 from gridsearch, the↙
baseline model gives
% a high r-squared value of 0.9215 which means that 92.15% variance in our predicted↙
values are explained by the model.
% Although MSE value of 1751473.18 is huge but it depends on the scale of our data↙
which is varied.

%% 3. Visualizing the difference between Original price and the predicted price in↙
training data

% Creating a line plot for predictive price and actual price for training set
figure('Position', [100, 100, 800, 500]);
plot(1:length(y_train), y_train, 'r-', 'LineWidth', 0.2);
hold on;
plot(1:length(y_predrf_train), y_predrf_train, 'b-', 'LineWidth', 0.2);
grid on;
```

```
xlabel('Data Point Index');
ylabel('Price');
legend('Actual Price', 'Predicted Price');
title('Actual vs. Predicted Prices for training set for Random Forest');

%% 4. Feature Importance selection for Random Forest
% code help taken from 'https://uk.mathworks.com/help/stats/select-predictors-for-↙
random-forests.html'
rng(43);
% Extract feature importance scores
importance_scores = randomforestmodel.OOBPermutedVarDeltaError;
% Specify the number of top features to select
num_top_features = 5;

% Find indices of top N features
[~, top_feature_indices] = maxk(importance_scores, num_top_features);
% Filter datasets to include only top features
X_train_selected = X_train(:, top_feature_indices);
X_validation_selected = X_validation(:, top_feature_indices);
X_test_selected = X_test(:, top_feature_indices);

% Retrain Random Forest on the selected features
randomforestmodel_selected = TreeBagger(bestNumTrees, X_train_selected, y_train,↙
'Method', 'regression', 'MaxNumSplits', bestMaxNumSplits);

% Make predictions from the selected columns on the training set
y_predrf_train_selected = predict(randomforestmodel_selected, X_train_selected);

% Calculate R-squared value and MSE value for my training model
rfMSE_train_selected = mean((y_train - y_predrf_train_selected).^2);
rfR2_train_selected = 1 - sum((y_train - y_predrf_train_selected).^2) / sum((y_train↙
- mean(y_train)).^2);

% Display R-squared value
disp(['MSE Value for Random Forest Model for selected columns : ' num2str↙
(rfMSE_train_selected)]);
disp(['R-squared Value for Random Forest Model for selected models: ' num2str↙
(rfR2_train_selected)]);

% While Feature selection does seem like an important step for predictive
% modeling, its application to my model yielded minimal impact on performance.
% Notably, the R-squared (r2) value remained consistent, and though there
% was a modest reduction in Mean Squared Error (MSE), it was not substantial
% enough to alter the overall model outcomes. This suggests that the initially
% chosen features already provided relevant information for prediction,
% and the removal of some features did not yield a significant improvement.
% I prefer having a model with all features for interpretability, even if it
% comes at the cost of a slightly more complex model.

%% 5. Evaluating the baseline model on the test data (unseen data)
clc
```

```matlab
% evaluating the performance of random forest model on the unseen test data without the feature selection.
% setting random seed for reproducibility
rng(43);

load("bestmodel for random forest.mat");

% Evaluating the baseline model on the testing set
y_predrf_test = predict(randomforestmodel, X_test);

% Calculating MSE and R-squared values on the testing set
rfMSE_test = mean((y_test - y_predrf_test).^2);
rfR2_test = 1 - sum((y_test - y_predrf_test).^2) / sum((y_test - mean(y_test)).^2);
disp(['MSE on Testing Set for Random Forest Model: ' num2str(rfMSE_test)]);
disp(['R-squared Value on Testing Set for Random Forest Model: ' num2str(rfR2_test)]);

% R2 value for my test set or unseen data is 0.91799 which is considered outstanding but the MSE value
% of this model on my test set is 1828169.73. The lower the MSE value, the better the result is, but our
% MSE value depends on the variability of my data and I want to check whether this high mse value is for
% the entire data or just the test set via k-fold cross validation.

%% 6. Visualizing the difference between Original price and the predicted price in testing set

% Creating a line plot for predictive price vs. actual price on test set
figure('Position', [100, 100, 800, 500]);
plot(1:length(y_test), y_test, 'g-', 'LineWidth', 0.2);
hold on;
plot(1:length(y_predrf_test), y_predrf_test, 'm-', 'LineWidth', 0.2);
grid on;
xlabel('Data Point Index');
ylabel('Price');
legend('Actual Price', 'Predicted Price');
title('Actual vs. Predicted Prices for test set for Random Forest');

%% 7. Visualizing the predictive and actual prices for the training and testing data

% Creating a line plot for collectively seeing the performance of the model on training as well as testing set
figure('Position', [100, 200, 1100, 500]);
plot(y_train, 'r', 'LineWidth', 0.2, 'DisplayName', 'Actual Training');
hold on;
plot(y_predrf_train, 'b', 'LineWidth', 0.2, 'DisplayName', 'Fitted Training');
plot(length(y_train) + (1:length(y_test)), y_test, 'g', 'LineWidth', 0.2, 'DisplayName', 'Actual Testing');
plot(length(y_train) + (1:length(y_test)), y_predrf_test, 'm', 'LineWidth', 0.2, 'DisplayName', 'Fitted Testing');
```

```
grid on;

title('RANDOM FOREST MODEL PREDICTIONS (on training and test set)');
xlabel('Observation');
ylabel('Price');
legend('show');

%% 8. Performing K-fold Cross Validation on my entire dataset for Random Forest
clc
% Performing K-fold Cross Validation after running my model on the training
% as well as testing data is crucial for robust model evaluation.
% It provides a more reliable estimate of the model's performance,
% reducing the risk of overfitting or underfitting observed with a single train-test↙
split.

% setting random seed for reproducibility
rng(43);
% Combine the training, validation, and testing sets
combined_data = [training_data; validation_data; testing_data];

%extract the features to put in X and Y
X = combined_data(:, (2:end));
y = combined_data(:,1);

% Define k-fold partition
cvpart = cvpartition(size(X, 1), 'kfold', 10);

% Performing k-fold cross-validation
mse_scores_rf = [];
r2_scores_rf = [];
for i = 1:cvpart.NumTestSets
    % Extract training and testing data
    trainIdx = cvpart.training(i);
    testIdx = cvpart.test(i);
    X_train = X(trainIdx, :);
    X_test = X(testIdx, :);
    y_train = y(trainIdx);
    y_test = y(testIdx);

   % Getting my random forest model
   randomforestmodel = TreeBagger(bestNumTrees, X_train, y_train, 'Method',↙
'regression', 'MaxNumSplits', bestMaxNumSplits);

   % Evaluating the model
    y_predrf = predict(randomforestmodel, X_test);

    % Calculating MSE
    kfMSE_rf = mean((y_predrf - y_test).^2);
    mse_scores_rf = [mse_scores_rf; kfMSE_rf];

    % Calculating R-squared value
```

```matlab
    kfR2_rf = corr(y_predrf, y_test)^2;
    r2_scores_rf = [r2_scores_rf; kfR2_rf];

    % Displaying results for each fold
    disp(['Fold ' num2str(i) ' - MSE: ' num2str(kfMSE_rf) ', R-squared: ' num2str(kfR2_rf)]);
end

% Analyzing overall results
average_mse_rf = mean(mse_scores_rf);
average_r2_rf = mean(r2_scores_rf);
disp(['Average MSE for k-fold: ' num2str(average_mse_rf)]);
disp(['Average R-squared for k-fold: ' num2str(average_r2_rf)]);

%The results obtained from k-fold cross-validation for our Random Forest model show promising performance.
% The model effectively captures a substantial portion of the variability in the target variable.
% The consistent MSE values indicate that overfitting is not a concern; instead, the observed higher
% MSE values are suggestive of the inherent complexity or dispersion within the dataset.
% It is important to recognize that these elevated MSE values do not necessarily reflect poor model
% performance but rather highlight the nuanced characteristics of the data, signaling potential complexity or variability.

%--------------------------------------------------------------------------
%%  DECISION TREE REGRESSION ALGORITHM

% For decision tree, I will be using the split data and the extracted features and target variables
% for training, testing and validation that I used earlier for random forest regression model.

%% 1. Grid Search to find the best Hyperparameters for Decision Tree Model

% code help taken from 'https://uk.mathworks.com/help/stats/decision-trees.html#bsw6p3v'
% Performing Grid Search method for getting the best Hyperparameters for Decision Tree Model.
clc
% Set the random seed for reproducibility
rng(43);

% Defining the range of hyperparameters to search over
maxsplitsrange = 5:5:100;
minLeafSizeRange = 5:5:50;

% Initializing variables to store results
bestMaxSplits = 0;
```

```
bestMinLeafSize = 0;
dtbestR2 = -Inf;
dtbestMSE = Inf;

% Performing grid search on validation set
for maxnumsplits = maxsplitsrange
    for minLeafSize = minLeafSizeRange

        % Creating decision tree model for regression for gridsearch
        dtreemodel_gridsearch = fitrtree(X_train, y_train, 'MinLeafSize', minLeafSize, 'MaxNumSplits', maxnumsplits);

        % Evaluating the decision tree on the validation set
        y_preddt_validation = predict(dtreemodel_gridsearch, X_validation);

        % Calculating R-squared value
        dtR2_validation = corr(y_preddt_validation, y_validation)^2;
        % Calculating Mean Squared Error (MSE)
        dtMSE_validation = mean((y_preddt_validation - y_validation).^2);

        % Checking if the current hyperparameters result in better R-squared value and lower MSE
        if dtR2_validation > dtbestR2 && dtMSE_validation < dtbestMSE
            bestMaxSplits = maxnumsplits;
            bestMinLeafSize = minLeafSize;
            dtbestR2 = dtR2_validation;
            dtbestMSE = dtMSE_validation;
         end
    end
end

% Displaying the best hyperparameters and performance metrics
disp(['Best MaxNumber of splits: ' num2str(bestMaxSplits)]);
disp(['Best MinLeafSize: ' num2str(bestMinLeafSize)]);
disp(['Best R-squared Value for Decision Tree Model: ' num2str(dtbestR2)]);
disp(['Best MSE Value for Decision Tree Model: ' num2str(dtbestMSE)]);

% Computational time taken for gridsearch for decision tree is way lesser than that taken for random forest.
% The result of this hyperparameter tuning gives us the best number of splits and the minimum leaf size so as
% to obtain the best predictive model by using these hyperparameters.

%% 2. Performing  Decision Tree Model on my training set using the best hyperparameters from GridSearch
clc
% to get the best results for our decision tree's model on our dataset, we'll be using the hyperparameters
% we got from the grid search for our Decision Trees model

% setting random seed for reproducibility
```

```matlab
rng(43);

% Best hyperparameters from grid search
bestMaxSplits = 100;
bestMinLeafSize = 20;

% Training decision tree model for regression on training set
dtreemodel = fitrtree(X_train, y_train, 'MinLeafSize', bestMinLeafSize, 'MaxNumSplits', bestMaxSplits);

%save the best model for decision tree
save("bestmodel for decision tree.mat", "dtreemodel");

% Make predictions on the training set
y_preddt_train = predict(dtreemodel, X_train);

% Calculate MSE and R2 values for the decision tree model
dtMSE_train = mean((y_train - y_preddt_train).^2);
dtR2_train = 1 - sum((y_train - y_preddt_train).^2) / sum((y_train - mean(y_train)).^2);
disp(['R-squared Value for Decision Tree Model on training set: ' num2str(dtR2_train)]);
disp(['MSE Value for Decision Tree Model on training set: ' num2str(dtMSE_train)]);

% Creating a table containing original and predicted prices for the decision tree model
trainingprediction_dt = table(y_train, y_preddt_train);
trainingprediction_dt.Properties.VariableNames = {'Original Price', 'Predicted Price'};

% Displaying the top 10 rows of the table for the decision tree model
display(trainingprediction_dt(1:10, :));

%% 3. Visualizing the difference between Original price and the predicted price in training data for Decision Tree

% Creating a line plot for predictive price and actual price for training set for the decision tree model
figure('Position', [100, 100, 800, 500]);
plot(1:length(y_train), y_train, 'r-', 'LineWidth', 0.2);
hold on;
plot(1:length(y_preddt_train), y_preddt_train, 'b-', 'LineWidth', 0.2);
grid on;
xlabel('Data Point Index');
ylabel('Price');
legend('Actual Price', 'Predicted Price');
title('Actual vs. Predicted Prices for training set for Decision Tree');

%% 4. Evaluating the baseline model on the test data (unseen data) for Decision Tree
clc
% setting random seed for reproducibility
```

```matlab
rng(43);

load("bestmodel for decision tree.mat");

% Evaluating the decision tree model on the testing set
y_preddt_test = predict(dtreemodel, X_test);

% Calculating MSE and R-squared values on the testing set
dtMSE_test = mean((y_test - y_preddt_test).^2);
dtR2_test = 1 - sum((y_test - y_preddt_test).^2) / sum((y_test - mean(y_test)).^2);

% Displaying the MSE and R-squared values on the testing set for the decision tree model
disp(['MSE on Testing Set (Decision Tree Model): ' num2str(dtMSE_test)]);
disp(['R-squared Value on Testing Set (Decision Tree Model): ' num2str(dtR2_test)]);

% It is performing as good on the test set as it did on the training set.

%% 5. Visualizing the difference between Original price and the predicted price in testing set for Decision Tree

% Creating a line plot for predictive price vs. actual price on test set for decision tree
figure('Position', [100, 100, 800, 500]);
plot(1:length(y_test), y_test, 'g-', 'LineWidth', 0.2);
hold on;
plot(1:length(y_preddt_test), y_preddt_test, 'm-', 'LineWidth', 0.2);
grid on;
xlabel('Data Point Index');
ylabel('Price');
legend('Actual Price', 'Predicted Price (Decision Tree)');
title('Actual vs predicted price for test set for Decision Tree Model');

%% 6. Visualizing the predictive and actual prices for the training and testing data for decision tree model

% Creating a line plot for collectively seeing the performance of the model on training as well as testing set
figure('Position', [100, 200, 1100, 500]);
plot(y_train, 'r', 'LineWidth', 0.2, 'DisplayName', 'Actual Training');
hold on;
plot(y_preddt_train, 'b', 'LineWidth', 0.2, 'DisplayName', 'Fitted Training');
plot(length(y_train) + (1:length(y_test)), y_test, 'g', 'LineWidth', 0.2, 'DisplayName', 'Actual Testing');
plot(length(y_train) + (1:length(y_test)), y_preddt_test, 'm', 'LineWidth', 0.2, 'DisplayName', 'Fitted Testing');
grid on;

title('DECISION TREE MODEL PREDICTIONS (on training and testing set)');
xlabel('Observation');
ylabel('Price');
```

```
legend('show');

  %% 7. Performing K-fold Cross Validation on my entire dataset

% K-fold cross validation is done to make sure that my model is not overfitting the training data.
% It has the same motive of providing a more reliable estimate of the model's generalization performance,
% helping identify potential overfitting or underfitting issues and ensuring the model's consistency across diverse data subset.
clc
% setting random seed for reproducibility
rng(43);

% Using the combined data and X and y features that we created above when performing the K-fold cross validation on random forest model.

% Defining k-fold partition
cvpart = cvpartition(size(X, 1), 'kfold', 10);

% Perform k-fold cross-validation on decision tree model (following the same procedure as for random forest)
mse_scores_dt = [];
r2_scores_dt = [];
for i = 1:cvpart.NumTestSets
    % Extracting training and testing data
    trainIdx = cvpart.training(i);
    testIdx = cvpart.test(i);
    X_train = X(trainIdx, :);
    X_test = X(testIdx, :);
    y_train = y(trainIdx);
    y_test = y(testIdx);

  % Getting my decision tree model
  dtreemodel = fitrtree(X_train, y_train, 'MinLeafSize', bestMinLeafSize, 'MaxNumSplits', bestMaxSplits);

    % Evaluating the model
    y_preddt = predict(dtreemodel, X_test);

    % Calculating MSE
    kfMSE_dt = mean((y_preddt - y_test).^2);
    mse_scores_dt = [mse_scores_dt; kfMSE_dt];

    % Calculating R-squared value
    kfR2_dt = corr(y_preddt, y_test)^2;
    r2_scores_dt = [r2_scores_dt; kfR2_dt];

    % Displaying results for each fold
    disp(['Fold ' num2str(i) ' - MSE: ' num2str(kfMSE_dt) ', R-squared: ' num2str(kfR2_dt)]);
```

```
end

% Analyzing overall results
average_mse_dt = mean(mse_scores_dt);
average_r2_dt = mean(r2_scores_dt);
disp(['Average MSE for k-fold for Decision Tree model: ' num2str(average_mse_dt)]);
disp(['Average R-squared for k-fold for Decision Tree model: ' num2str↙
(average_r2_dt)]);

%-----------------------------------------------------------------------
%% Q. Does scaling the data before applying Random Forest and Decision Tree models↙
yield comparable results to unscaled data, or does it significantly impact model↙
performance?
%(extra analytical question)

% - Scaling of my variables (Through minmax scaling)

% Before scaling
figure;
subplot(2, 1, 1);
histogram(X_train(:, 1), 'BinEdges', linspace(min(X_train(:, 1)), max(X_train(:,↙
1)), 20));
title('Before Scaling - Feature 1');

% Calculate min and max values for scaling
min_values_train = min(X_train, [], 1);
max_values_train = max(X_train, [], 1);

% Scale training data
for feature_index = 1:size(X_train, 2)
    X_train(:, feature_index) = (X_train(:, feature_index) - min_values_train↙
(feature_index)) / (max_values_train(feature_index) - min_values_train(feature_index));
end

% Scale validation data using the same min and max values
for feature_index = 1:size(X_validation, 2)
    X_validation(:, feature_index) = (X_validation(:, feature_index) -↙
min_values_train(feature_index)) / (max_values_train(feature_index) - min_values_train↙
(feature_index));
end

% Scale test data using the same min and max values
for feature_index = 1:size(X_test, 2)
    X_test(:, feature_index) = (X_test(:, feature_index) - min_values_train↙
(feature_index)) / (max_values_train(feature_index) - min_values_train(feature_index));
end

% After scaling
subplot(2, 1, 2);
histogram(X_train(:, 1), 'BinEdges', linspace(0, 1, 20));
title('After Scaling - Feature 1');
```

```

% The subplots are just to check whether my scaling worked or not.

%%  Random Forest with scaled data
% 1.Grid Search
% setting random seed for reproducibility
rng(43);

% Defining the range of hyperparameters to search over
numTreesRange = 10:50:300;
maxNumSplitsRange = 10:10:150;

% Initializing variables to store results
bestNumTrees = 0;
bestMaxNumSplits = 0;
rfbestR2 = -Inf;
rfbestMSE = Inf;

% Performing grid search
for numTrees = numTreesRange
    for maxNumSplits = maxNumSplitsRange

        % Creating Random Forest model for regression
        randomforestmodel = TreeBagger(numTrees, X_train, y_train, 'Method',↙
'regression', 'MaxNumSplits', maxNumSplits);

        % Evaluating random forest on the validation set
        y_predrf = predict(randomforestmodel, X_validation);

        % Calculating R-squared value
        rfR2 = corr(y_predrf, y_validation)^2;
         % Calculating Mean Squared Error (MSE)
        rfMSE = mean((y_predrf - y_validation).^2);

 % Checking if the current hyperparameters result in better R-squared value and↙
lower MSE
        if rfR2 > rfbestR2 && rfMSE < rfbestMSE
            bestNumTrees = numTrees;
            bestMaxNumSplits = maxNumSplits;
            rfbestR2 = rfR2;
            rfbestMSE = rfMSE;
        end
    end
end

% Display the best hyperparameters and R-squared value
disp(['Best Number of Trees: ' num2str(bestNumTrees)]);
disp(['Best MaxNumSplits: ' num2str(bestMaxNumSplits)]);
disp(['Best R-squared Value for random forest: ' num2str(rfbestR2)]);
disp(['Best MSE Value for random forest:  '  num2str(rfbestMSE)]);

```

```matlab
% 2. Running the model on training set for Random Forest (Scaled Data)

% setting random seed for reproducibility
rng(43);

% Best hyperparameters from grid search
bestNumTrees = 60;
bestMaxNumSplits = 140;

% Training my random forest model on training set
randomforestmodel = TreeBagger(bestNumTrees, X_train, y_train, 'Method', 'regression', 'MaxNumSplits', bestMaxNumSplits);

% Make predictions on the training set
y_predrf_train = predict(randomforestmodel, X_train);

% Calculate R-squared value and MSE value for my training model
rfMSE_train = mean((y_train - y_predrf_train).^2);
rfR2_train = 1 - sum((y_train - y_predrf_train).^2) / sum((y_train - mean(y_train)).^2);

% Display R-squared value
disp(['MSE Value for Random Forest Model : ' num2str(rfMSE_train)]);
disp(['R-squared Value for Random Forest Model: ' num2str(rfR2_train)]);

% There isn't much of a difference between these results and the ones
% obtained without scaling

%% Decision Tree with scaled data
% 1. Grid Search
% Set the random seed for reproducibility
rng(43)

% Define the range of hyperparameters to search over
maxsplitsrange = 5:5:250;
minLeafSizeRange = 5:5:50;

% Initialize variables to store results
bestMaxSplits = 0;
bestMinLeafSize = 0;
dtbestR2 = -Inf;
dtbestMSE = Inf;

% Performing grid search on validation set
for maxnumsplits = maxsplitsrange
    for minLeafSize = minLeafSizeRange

        % Creating decision tree model for regression
        dtreemodel = fitrtree(X_train, y_train, 'MinLeafSize', minLeafSize, 'MaxNumSplits', maxnumsplits);

        % Evaluating the decision tree on the validation set
        y_preddt = predict(dtreemodel, X_validation);

        % Calculating R-squared value
        dtR2 = corr(y_preddt, y_validation)^2;

        % Calculating Mean Squared Error (MSE)
        dtMSE = mean((y_preddt - y_validation).^2);

        % Checking if the current hyperparameters result in better R-squared value and lower MSE
        if dtR2 > dtbestR2 && dtMSE < dtbestMSE
            bestMaxSplits = maxnumsplits;
            bestMinLeafSize = minLeafSize;
            dtbestR2 = dtR2;
            dtbestMSE = dtMSE;
         end
    end
end

% Display the best hyperparameters and performance metrics
disp(['Best MaxNumber of splits: ' num2str(bestMaxSplits)]);
disp(['Best MinLeafSize: ' num2str(bestMinLeafSize)]);
disp(['Best R-squared Value for Decision Tree Model: ' num2str(dtbestR2)]);
disp(['Best MSE Value for Decision Tree Model: ' num2str(dtbestMSE)]);

% 2. Running the model on training set for Decision Tree (Scaled Data)

% setting random seed for reproducibility
rng(43);

% Best hyperparameters from grid search
bestMaxSplits = 100;
bestMinLeafSize = 20;

% Creating decision tree model for regression
dtreemodel = fitrtree(X_train, y_train, 'MinLeafSize', bestMinLeafSize, 'MaxNumSplits', bestMaxSplits);

% Make predictions on the training set
y_preddt_train = predict(dtreemodel, X_train);

% Calculate MSE value for the decision tree model
dtMSE_train = mean((y_train - y_preddt_train).^2);

% Calculate R-squared value for the decision tree model
dtR2_train = 1 - sum((y_train - y_preddt_train).^2) / sum((y_train - mean(y_train)).^2);

% Display the calculated values
disp(['R-squared Value for Decision Tree Model: ' num2str(dtR2_train)]);
disp(['MSE Value for Decision Tree Model: ' num2str(dtMSE_train)]);

% I am getting the same values as I got before scaling for grid research for decision tree model.

% Scaling the data had minimal impact on the results of Random Forest, while having no impact on
% Decision Tree Model's performance . Consequently, it can be inferred that both Random Forest and Decision Tree algorithms
% exhibit robustness to non-scaled data, performing admirably without the need for normalization or scaling.
%---------------------------------------------------------------------------
```