```
%% --------MATLAB Final Project - Engine Type Clustering --------%%
%                         By: Ori Algave
%
% This script loads aircraft performance data, cleans it, converts
% ft/in columns to numeric feet, standardizes engine labels,
% partitions data into Test Set 1, Test Set 2, and Training Set,
% normalizes features, applies weights, runs K-means clustering,
% maps clusters to engine types, and evaluates results.
%-------------------------------------------------------------------%
clc; clear; close all;  % Clear command window, workspace variables, and open figures
%% ------------------------------------------------------------------
%  STEP 1 - Load the raw data
% ------------------------------------------------------------------
filename = 'bluebook.csv';  % Name of input CSV file containing aircraft data
rawData = readtable(filename, 'VariableNamingRule','preserve');  % Read CSV as a table, then preserve original column headers
cleanedData = rawData;  % Make a working copy for cleaning so rawData stays unchanged

%% ------------------------------------------------------------------
%  STEP 2 - Remove rows with too many missing values (5+)
% ------------------------------------------------------------------
keepRow = true(height(cleanedData),1);  % Logical vector indicating which rows to keep (start with all true)
for ii = 1:height(cleanedData)  % Loop through each row of entire table
    numMissing = sum(ismissing(cleanedData(ii,:)));  % Count missing entries in current row using ismissing
    if  numMissing > 5  % If more than 5 cells are missing in this row
        keepRow(ii) = false;  % Mark the current row as false so it will be removed
    end
end
cleanedData = cleanedData(keepRow,:);  % Keep only rows marked true, drop rows with too many NaNs

%% ------------------------------------------------------------------
%  STEP 3 - Remove columns with >10% missing
% ------------------------------------------------------------------
numRows = height(cleanedData);  % Store number of remaining rows after row cleaning
keepCol = true(1, width(cleanedData));  % Logical vector for keeping or dropping columns

for jj = 1:width(cleanedData)  % Loop through each column of the table
    numMissingCol = sum(ismissing(cleanedData(:,jj)));  % Count how many entries in this column are missing
    if  numMissingCol > 0.10*numRows  % If more than 10% of entries are missing
        keepCol(jj) = false;  % Mark this column as false to be removed
    end
end

cleanedData = cleanedData(:, keepCol);  % Keep only columns with acceptable levels of missing data

% Delete column with 'Height' because it has months and isn't useful for data
colsToDelete = ["Height ft/in"];
for ii = colsToDelete
     if ismember(ii, cleanedData.Properties.VariableNames)
         cleanedData.(ii) = [];
     end
end

%% ------------------------------------------------------------------
%  STEP 4 - Standardize Engine Type Labels
% ------------------------------------------------------------------
```

```
engineTypeRaw = cleanedData.("Engine Type");  %Extract original engine type column from cleaned table
engineStr = string(engineTypeRaw);  % Convert to string array for easier text operations
engineStr = strtrim(lower(engineStr));  % Convert to lowercase and trim surrounding whitespace

engineTypeLabels = strings(size(engineStr));  % Preallocate array for standardized engine labels

for ii = 1:numel(engineStr)  % Loop through each original engine type string
    et = engineStr(ii);  % Shorter variable name for current engine string
    if contains(et,"jet") && ~contains(et,"prop")  % If contains 'jet' but not 'prop', treat it as pure jet
        engineTypeLabels(ii) = "Jet";  % Standardized label: Jet
    elseif contains(et,"prop")  % If contains 'prop', treat as a turboprop / propjet
        engineTypeLabels(ii) = "Propjet";  % Standardized label Propjet
    elseif contains(et,"pist")  %If matches 'pist' (piston variants / misspellings)
        engineTypeLabels(ii) = "Piston";  % Standardized label: Piston
    else
        engineTypeLabels(ii) = "UNKNOWN";  % Anything not matching the above categorizations is unknown, mark as  UNKNOWN so it could be removed later
    end
end

cleanedData.EngineTypeStandardized = engineTypeLabels;  % Attach standardized engine labels as a column in table

%% ---------------------------------------------------------------
%  STEP 5 - Remove UNKNOWN engine types before clustering
% ---------------------------------------------------------------
unknownMask = (engineTypeLabels == "UNKNOWN");  % Logical mask where standardized label equals UNKNOWN
cleanedData(unknownMask,:) = [];  % Remove UNKNOWN rows from cleaned data
engineTypeLabels(unknownMask) = []; % Remove corresponding labels to keep alignment

%% ---------------------------------------------------------------
%  STEP 6 - Convert ft/in columns
% ---------------------------------------------------------------

% ---------- LENGTH ----------
if ismember("Length ft/in", cleanedData.Properties.VariableNames)  % Check if Length ft/in string column exists
    rawLength = cleanedData.("Length ft/in");  % Original name has a space, so set it equal to a variable
    lengthFeet = zeros(height(cleanedData), 1);  % Preallocate numeric vector for length in feet

    for ii = 1:height(cleanedData)  % Loop through each row to comvert length
        value = rawLength{ii};  % Extract raw ft/in string from table cell

        if contains(value, "/")  % If string has '/', assume its 'feet / inches' format
            parts = split(value, "/");  % split function splits the string into substring parts around '/'

            if numel(parts) == 2  % If there are exactly 2 parts, feet and inches
                feet   = str2double(parts{1});  % str2double converts feet substring to numeric value
                inches = str2double(parts{2});  % Convert inches substring to numeric value
            else
                feet   = str2double(parts{1});  % If more unusual format but still uses '/' then first part is feet
                inches = 0;  % Defaults inches to zero
            end
        else
```

```
            feet   = str2double(value);  % If there is no '/', treat entire value as feet only and convert full string directly to numeric feet
            inches = 0;  % No inches in this case
        end

        totalInches = feet*12 + inches;      % Convert combination of feet and inches into total inches
        lengthFeet(ii) = totalInches / 12;   % Convert inches back to feet
    end

    cleanedData.LengthFeet = lengthFeet;  % Store numeric length in new column LengthFeet
    cleanedData.("Length ft/in") = [];   % Remove original string based length column
end


% ---------- WINGSPAN ----------
if ismember("Wing span ft/in", cleanedData.Properties.VariableNames)  % Check if Wing span ft/in string column exists
    rawWing = cleanedData.("Wing span ft/in");  % Original name has a space, so set it equal to a variable
    wingspanFeet = zeros(height(cleanedData), 1);   % Preallocate numeric vector for wingspan in feet

    for ii = 1:height(cleanedData)  % Loop through each row for wingspan conversion
        value = rawWing{ii};  % Extract raw wingspan ft/in string

       if contains(value, "/")  % If string has '/', assume its 'feet / inches' format
            parts = split(value, "/");  % split function splits the string into substring parts arounf '/'

            if numel(parts) == 2  % If there are exactly 2 parts, feet and inches
                feet   = str2double(parts{1});  % str2double converts feet substring to numeric value
                inches = str2double(parts{2});  % Convert inches substring to numeric value
            else
                feet   = str2double(parts{1});  % If more unusual format but still uses '/' then first part is feet
                inches = 0;  % Defaults inches to zero
            end
        else
            feet   = str2double(value);  % If there is no '/', treat entire value as feet only and convert full string directly to numeric feet
            inches = 0;  % No inches in this case
        end

        totalInches = feet*12 + inches;      % Convert combination of feet and inches into total inches
        wingspanFeet(ii) = totalInches / 12;   % Convert inches back to feet
    end

    cleanedData.WingSpanFeet = wingspanFeet;  % Store numeric wingspan measures
    cleanedData.("Wing span ft/in") = [];  % Remove original wingspan string column
end

%% ------------------------------------------------------------
%  STEP 7 - Build numeric matrix and impute missing values
% ------------------------------------------------------------
numericData = cleanedData(:, vartype("numeric"));  % Select only numeric columns using vartype("numeric")
X = table2array(numericData);  % Convert numeric table to matrix for faster numeric operations

```

```
for col = 1:size(X,2)  % Loop over each feature column in X
    colData = X(:,col);  % Extract one column as a vector
    mu_val = mean(colData,'omitnan');  % Compute mean while ignoring NaNs (omitnan ignores missing data)
    colData(isnan(colData)) = mu_val;  % Replace NaNs in this column with the column mean
    X(:,col) = colData;  % Put cleaned column back into X
end

disp("Remaining NaN count in X:");  % Print label indicating NaN count check
disp(sum(isnan(X),'all'));  % Count all NaNs in X, this value should be zero after imputation

% Overwrite numeric columns in cleanedData with fully imputed values (Extract names of numeric variables to update table columns)
numericVars = numericData.Properties.VariableNames;  % Get variable names so they could overwrite each column in loop
for col = 1:length(numericVars)  % Loop across each numeric variable column name
    cleanedData.(numericVars{col}) = X(:,col);  % Replace original column in cleanedData with fully imputed numeric column
end

%% ---------------------------------------------------------------
%  STEP 8 - Sectioning the data set into test and training sets
%----------------------------------------------------------------

% Calculate 10% of total dataset size
totalRows = height(cleanedData);  % Get the total number of rows in the cleaned dataset
numToPick = round(0.10 * totalRows);  % Get the number of rows that make up 10% of the total data to be extracted

%----------------------------------------------------------------
%          Create the first 10% test set
%----------------------------------------------------------------
test1Indices = [];  % Initialize an empty array to store indices for test set 1
while length(test1Indices) < numToPick  % Loop until there are enough unique indices that makeup less that 10%
    randomIndex = randi(totalRows);  % Randomly generate a random integer from 1 to total rows
    if ~ismember(randomIndex, test1Indices)  % Check if the current index has not been chosen already, ismember checks if its a part of the chosen indices
        test1Indices(end+1) = randomIndex;  % If it's a new index then add after the last index in the list of selected indices
    end
end

%----------------------------------------------------------------
%           Create the second 10% test set
%----------------------------------------------------------------
test2Indices = [];  % Initialize an empty array to store indices for test set 2
while length(test2Indices) < numToPick  % Loop until there are enough unique indices that makeup less that 10%
    randomIndex = randi(totalRows);  % Randomly generate a random integer from 1 to total rows
    if ~ismember(randomIndex, [test1Indices, test2Indices])  % Check if the current index has not been chosen already from set 1 or for set 2, ismember checks if its a part of the chosen indices
        test2Indices(end+1) = randomIndex;  % If it's a new index then add after the last index in the list of selected indices
    end
end

%----------------------------------------------------------------
%            Extract test sets using selected indices
%----------------------------------------------------------------
```

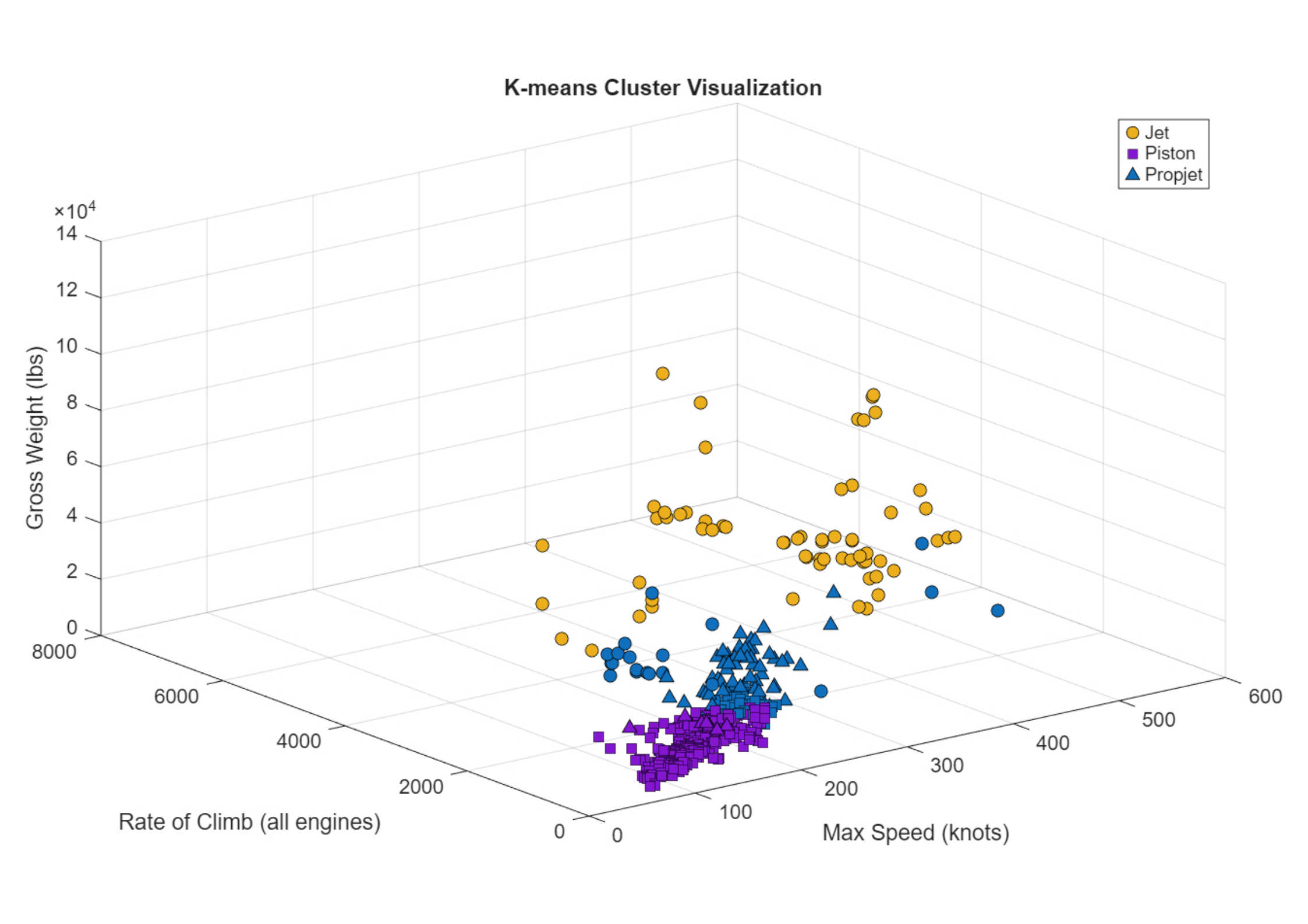
```
testSet1 = cleanedData(test1Indices, :);  % Use the selected indices to extract the first test set from the cleaned data based off of the list of random indices
testSet2 = cleanedData(test2Indices, :);  % Use the selected indices to extract the second test set from the cleaned data based off of the list of random indices

% Extract engine labels for each test set
testLabels1 = engineTypeLabels(test1Indices);  % Engine labels that match test set 1
testLabels2 = engineTypeLabels(test2Indices);  % Engine labels that match test set 2

%-------------------------------------------------------------
%             Build the training set (remaining 80%)
%-------------------------------------------------------------
remainingIndices = setdiff(1:totalRows, [test1Indices test2Indices]);  % Find all indices not selected into test sets
trainingSet = cleanedData(remainingIndices, :);  % Use remaining indices to create the training set
trainingLabels = engineTypeLabels(remainingIndices);  % Use same remaining indices to extract training labels

%-------------------------------------------------------------
%             Save all 3 sets into CSV files
%-------------------------------------------------------------
writetable(testSet1, 'test_set_1.csv');  % Writes the first 10% of indices into its own csv for test 1
writetable(table(testLabels1), 'test_set_1_labels.csv');  % Writes the matching engine labels for test 1

writetable(testSet2, 'test_set_2.csv');  % Writes the second 10% of indices into its own csv for test 2
writetable(table(testLabels2), 'test_set_2_labels.csv');  % Writes the matching engine labels for test 2

writetable(trainingSet, 'training_set.csv');  % Writes the remaining 80% of clean data into its own csv for training
writetable(table(trainingLabels), 'training_labels.csv');  % Writes the matching engine labels for training

%-------------------------------------------------------------
%  Update cleanedData and engineTypeLabels to training data only
%-------------------------------------------------------------
cleanedData = trainingSet;  % Replace cleanedData with training set for future steps
engineTypeLabels = trainingLabels;  % Replace engine labels with training labels for clustering

% Rebuild numeric matrix X for the training pipeline
numericData = cleanedData(:, vartype("numeric"));  % Extract numeric columns for model training
X = table2array(numericData);  % Convert numeric table into matrix for normalization and clustering

%% -------------------------------------------------------------
%  STEP 9 - Z-score normalization
%  Z-score = (x - mu) / sigma
% -------------------------------------------------------------

% Numeric feature names (for saving parameters later)
varNames = numericData.Properties.VariableNames;  % This keeps track of which column corresponds to which feature which allows for correct normalization

% Preallocate mean and std arrays (one per column of X)
numFeatures = size(X, 2);  % Determine how many numeric features there are in total
mu = zeros(1, numFeatures);  % Stores mean for each feature column
```

```
sigma = ones(1, numFeatures);  % Stires standard deviation, which is initialized to all 1 to avoid division problems

% Loop over each numeric feature/column
for ii = 1:numFeatures  % Loop through every numeric feature and calculate z-score parameters
    col = X(:, ii);  % Extract the full column for the current feature
    mu(ii)    = mean(col, 'omitnan');   % Calculate the mean and ignore any missing values, the mean centers the distribution
    sigma(ii) = std(col,  'omitnan');   % Calculate the standard deviation and ignore any missing values, the standard deviation scales the distribution

    % Avoid divide-by-zero for constant columns
    if sigma(ii) == 0
        sigma(ii) = 1;  % Prevents division by zero and preserves the feature shape
    end

    % Perform z-score normalization and overwrite this column with its z-scored version
    X(:, ii) = (col - mu(ii)) / sigma(ii);  % This transformation makes all feautres comparable in scale, which prevents variables with a larger magnitude from having a larger influence on clustering
end

Xnorm = X;  % Final normalized feature matrix stored separately

% Save normalization parameters for future use on test sets
save('normParams_loop.mat', 'mu', 'sigma', 'varNames');  % Save mu and sigma so test sets could use identical scaling

%% ---------------------------------------------------------------
%  STEP 10 - Weight Assigning
% ---------------------------------------------------------------
numFeatures = size(Xnorm,2);  % Count how many normalized features exist
weights = ones(1,numFeatures);  % Initialize all feature weights to 1 to have a neutral influence
numericNames = numericData.Properties.VariableNames;  % Retreieve feature names used to apply targeted weights

for ii = 1:numFeatures  % Loop through all of the features
    lname = lower(numericNames{ii});  % Convert all feature names to lowercase for reliable substring matching
    if contains(lname,"rate") && contains(lname,"climb")  % Rate of climb strongly separates turbine vs piston aircraft
        weights(ii) = 9.0;
    elseif contains(lname,"service") && contains(lname,"ceiling")  % Service ceiling is engine power dependant, so jets highly dominate this region
        weights(ii) = 4.5;
    elseif contains(lname,"max") && contains(lname,"speed")  % Max speed cleanly separates jets from others, propjets fall in between here
        weights(ii) = 2.5;
    elseif contains(lname,"cruise") || contains(lname,"rcmnd")  % Cruise speed correlates with propjet vs piston crossover
        weights(ii) = 4.5;
    elseif contains(lname,"takeoff")  % Takeoff distance reflects the thrust to weight and engine response
        weights(ii) = 4.0;
    elseif contains(lname,"landing")  % Landing distance is somewhat correlated but is largely influences by the drag and flaps used in landing
        weights(ii) = 3.5;
    elseif contains(lname,"weight")  % Gross/ empty weight is a reflection of aircraft category and engine capability
        weights(ii) = 4.5;
```

```
    elseif contains(lname,"stall")  % Stall speed provides some insight into engine categories but is not so decisive
        weights(ii) = 1.5;
    elseif contains(lname,"fuel")  % Fuel capacity varies widely but isn't very string at categorizing engine types
        weights(ii) = 1.5;
    elseif contains(lname,"range")  % Range is influence by a large magnitude of factors, engine type itself isn't very predictive alone
        weights(ii) = 1.0;
    elseif contains(lname,"length") || contains(lname,"height") || contains(lname,"wing")  % The dimensions of the aircraft have some dependancy on engine type
        weights(ii) = 0.5;
    else  % Any remaining features that havent been accounted for get a neutral default weight
        weights(ii) = 1.0;
    end
end

Xweighted = Xnorm .* weights;  % Apply the weights by scaling each feature column

%% ---------------------------------------------------------------
%  STEP 11 - K-means clustering
% ---------------------------------------------------------------
k = 3;  % 3 clusters for the 3 engine types: Jet, Piston, Propjet
rng(1);  % Fixing random seed ensures reproducible clustering results and doesn't rerandomize centroids
[idx, C] = kmeans(Xweighted, k,...  % Perform k-means on the weighted, normalized feature set
    'Replicates', 25,...  % Run the algorithm 25 times to avoid a bad local minima
    'MaxIter', 1000,...  % Allow up to 1000 iterations for convergence per replicate
    'Display','final');  % MATLAB prints summary including best within cluster sum of squared errors
%% ---------------------------------------------------------------
%  STEP 12 - Majority Voting Cluster Assignments
% ---------------------------------------------------------------
engineTypes = ["Jet","Piston","Propjet"];  % Target categories for classifications
clusterToType = strings(k,1);  % Preallocate mapping array

for ii = 1:k
    members = engineTypeLabels(idx==ii);  % Extract engine types that appear inside the given cluster
    if isempty(members)  % If a cluster ended up empty
        clusterToType(ii) = "UNKNOWN";  % Label it as UNKNOWN
        continue;
    end
    numJet = sum(members=="Jet");  % Count the number of each jet engine class in a given cluster
    numPiston = sum(members=="Piston");  % Count the number of each piston engine class in a given cluster
    numPropjet = sum(members=="Propjet");  % Count the number of each propjet engine class in a given cluster
    % This applies a majority voting rule for classifications of clusters
    if numPropjet >= max(numJet,numPiston)*0.63  % If propjets are atleast 63% of the dominant class, classify as propjet
        clusterToType(ii) = "Propjet";  % Propjets have been seemingly the most difficult to separate, so its boosted to improve the mapping of it later on
    elseif numJet >= numPiston  % If jets dominate more than pistons
        clusterToType(ii) = "Jet";  % Assign the cluster type as jet
    else
        clusterToType(ii) = "Piston";  % Otherwise, assign the cluster as piston
    end
end

```

```
disp("Cluster Labels:");
disp(clusterToType);  % This displays the reuslting cluster classification labels

%% ------------------------------------------------------------
%  STEP 13 - Confusion Matrix
% ------------------------------------------------------------
predictedTrain = clusterToType(idx);  % Convert cluster numbers back into engine type predictions

% Convert to categorical
engineTypes = ["Jet","Piston","Propjet"];  % Define category order for a consistent result
orderCats = categorical(engineTypes, engineTypes);  % Force categorical ordering in confusion
matrix

trueCat = categorical(engineTypeLabels, engineTypes);  % Convert true labels to categorical type
predCat = categorical(predictedTrain, engineTypes);  % Convert predicted labels into the same
categorical type

confMat = confusionmat(trueCat, predCat, 'Order', orderCats);  % Compute the confusion matrix
using consistent labeling order

confTable = array2table(confMat,'VariableNames', cellstr(engineTypes),'RowNames',
cellstr(engineTypes));  % Convert the confusion matrix to a readable table

disp("Training Confusion Matrix:");
disp(confTable);  % Print the full confusion matrix

acc = sum(predCat == trueCat) / numel(trueCat);  % Compute the fraction of correctly predicted
aircraft (predicted / true)
fprintf("Training Accuracy: %.2f%%\n", acc * 100);  % Display the accuracy of predicted / true

%% ------------------------------------------------------------
%  STEP 14 - Visualization (Max Speed vs Climb Rate vs Gross Weight)
% ------------------------------------------------------------
maxSpeed = cleanedData.("Max speed Knots");  % Use max speed as x-axis feature for visualization
roc = cleanedData.("All eng rate of climb");  % Use rate of climb as y-axis feature
gross = cleanedData.("Gross weight lbs");  % Use gross weight as z-axis feature
clusterCat = categorical(idx);  % Convert the cluster IDs into categorical values for coloring

figure; hold on; grid on;  % Start a new scatter plot with the grid enabled for visual clarity
markers = {'o','s','^'};  % Marker shapes corresponding to each true engine type

for ii = 1:numel(engineTypes)
    mask = (trueCat == engineTypes(ii));  % Filter the rows belonging to the true class ii
    scatter3(maxSpeed(mask), roc(mask), gross(mask), ...  % Scatter plot showing cluster
coloring but true shaped markers
    40, clusterCat(mask), markers{ii}, 'filled', 'MarkerEdgeColor', 'k', 'LineWidth', 0.4);  %
Filled markers with an outer edge to improve visibility on the clustered and overlapped markers
end

xlabel("Max Speed (knots)");  % Label x-axis for interpretabilty
ylabel("Rate of Climb (all engines)");  % Label y-axis
zlabel("Gross Weight (lbs)");  % Label z-axis
title("K-means Cluster Visualization");  % Plot title describing the purpose of the plot
legend("Jet","Piston","Propjet");  % Includes a legend representing true engine types via the
marker shapes
colormap(lines);  % Use MATLAB's built in parula colormap for good color contrast
view(3);  % 3D perspective in MATLAB
rotate3d on;  % Allows for drag and rotate of plot
hold off;  % Finish the plot
% This type of visualization of the clustering makes it clear to see where there are
misclassifications
```

```
399 % The color shows what k-means thinks the plane is (predicted)
400 % The marker shape shows what the plane really is
401
```


K-means Cluster Visualization
Jet
Piston
Propjet
×10^4
14
12
10
8
6
4
2
0
Gross Weight (lbs)
8000
6000
4000
2000
0
Rate of Climb (all engines)
0
100
200
300
400
500
600
Max Speed (knots)

Command Window

```
Cluster Labels:
    "Propjet"
    "Jet"
    "Piston"

Training Confusion Matrix:
```

	Jet	Piston	Propjet
Jet	65	0	21
Piston	0	372	51
Propjet	0	8	61

```
Training Accuracy: 86.16%
>>
```