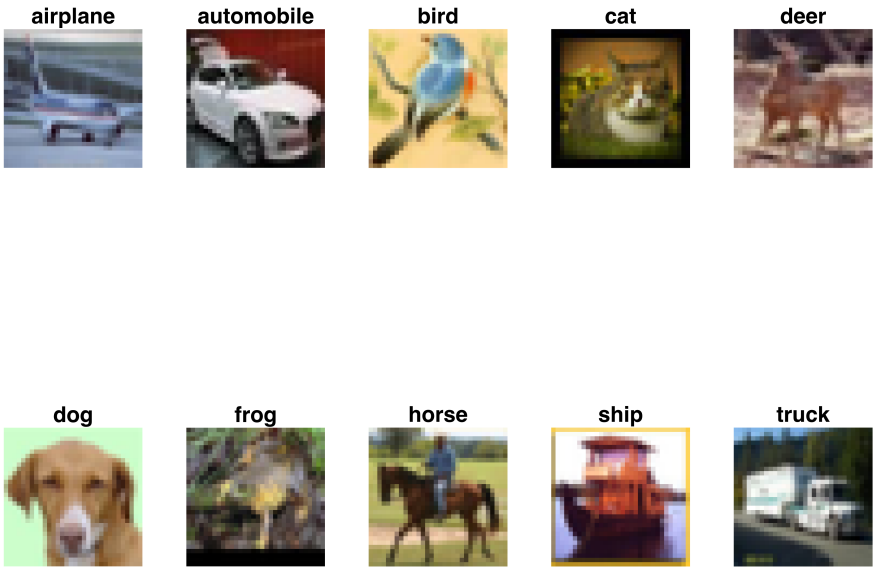
```
clc;
close all;
clear all;

% Load CIFAR-10 dataset
load('cifar10.mat');

% Print one image of each class
classes = {'airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog',
'horse', 'ship', 'truck'};
num_classes = length(classes);

figure;
for i = 1:num_classes
    % Find index of the first image in the current class
    class_idx = find(all_labels == (i - 1), 1);
    % Reshape image data to 32x32x3
    img = reshape(all_images(class_idx, :), [32, 32, 3]);
    % Rotate image right by 90 degrees
    img_rotated = imrotate(img, -90);
    % Plot the rotated image
    subplot(2, 5, i);
    imshow(uint8(img_rotated));
    title(classes{i});
end
```



```
% Display shape of features and number of samples per class
fprintf('Shape of features: %dx%d\n', size(all_images, 1), size(all_images,
2));
```

```
Shape of features: 60000x3072
```

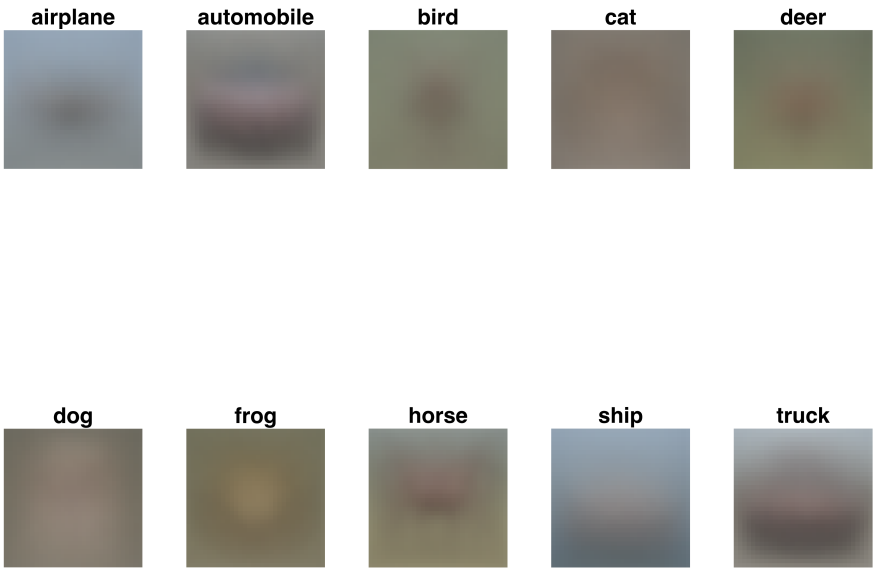
```
fprintf('Number of samples per class:\n');
```

```
Number of samples per class:
```

```
for i = 1:num_classes
    num_samples = sum(all_labels == (i - 1));
    fprintf('%s: %d\n', classes{i}, num_samples);
end
```

```
airplane: 6000
automobile: 6000
bird: 6000
cat: 6000
deer: 6000
dog: 6000
frog: 6000
horse: 6000
ship: 6000
truck: 6000
```

```
% Calculate mean for each class
mean_per_class_original = zeros(num_classes, size(all_images, 2));
for i = 1:num_classes
    class_indices = find(all_labels == (i - 1)); % Indices of samples
belonging to the current class
    mean_per_class_original(i, :) = mean(all_images(class_indices, :), 1);
end

% Plot the mean image for each class
figure;
for i = 1:num_classes
    % Reshape mean image data to 32x32x3
    mean_img_original = reshape(mean_per_class_original(i, :), [32, 32, 3]);
    % Rotate mean image right by 90 degrees
    mean_img_rotated_original = imrotate(mean_img_original, -90);
    % Plot the rotated mean image
    subplot(2, 5, i);
    imshow(uint8(mean_img_rotated_original));
    title(classes{i});
end
```




```
% Subset of the original data
rng(123); % Set a fixed random seed for reproducibility

% Subsample the data
subsample_size = 2000; % Choose the desired size of the subsample
subsample_indices = [];
for i = 1:num_classes
    class_indices = find(all_labels == (i - 1)); % Indices of samples
belonging to the current class
    subsample_indices = [subsample_indices; datasample(class_indices,
subsample_size, 'Replace', false)];
end

% Extract subsampled data and labels
subsample_data = all_images(subsample_indices, :);
subsample_labels = all_labels(subsample_indices);

% Shuffle the subsampled data and labels
shuffled_indices = randperm(size(subsample_data, 1));
subsample_data_shuffled = subsample_data(shuffled_indices, :);
subsample_labels_shuffled = subsample_labels(shuffled_indices);

% Print one image of each class from the subsampled data
figure;
for i = 1:num_classes
```

```
    % Find index of the first image in the current class
    class_idx = find(subsample_labels == (i - 1), 1);
    % Reshape image data to 32x32x3
    img = reshape(subsample_data(class_idx, :), [32, 32, 3]);
    % Rotate image right by 90 degrees
    img_rotated = imrotate(img, -90);
    % Plot the rotated image
    subplot(2, 5, i);
    imshow(uint8(img_rotated));
    title(classes{i});
end
```


```
% Display shape of subsampled features and number of samples per class
fprintf('Shape of subsampled features: %dx%d\n', size(subsample_data, 1),
size(subsample_data, 2));
```

```
Shape of subsampled features: 20000x3072
```

```
fprintf('Number of samples per class in subsampled data:\n');
```

```
Number of samples per class in subsampled data:
```

```
for i = 1:num_classes
    num_samples = sum(subsample_labels == (i - 1));
    fprintf('%s: %d\n', classes{i}, num_samples);
end
```

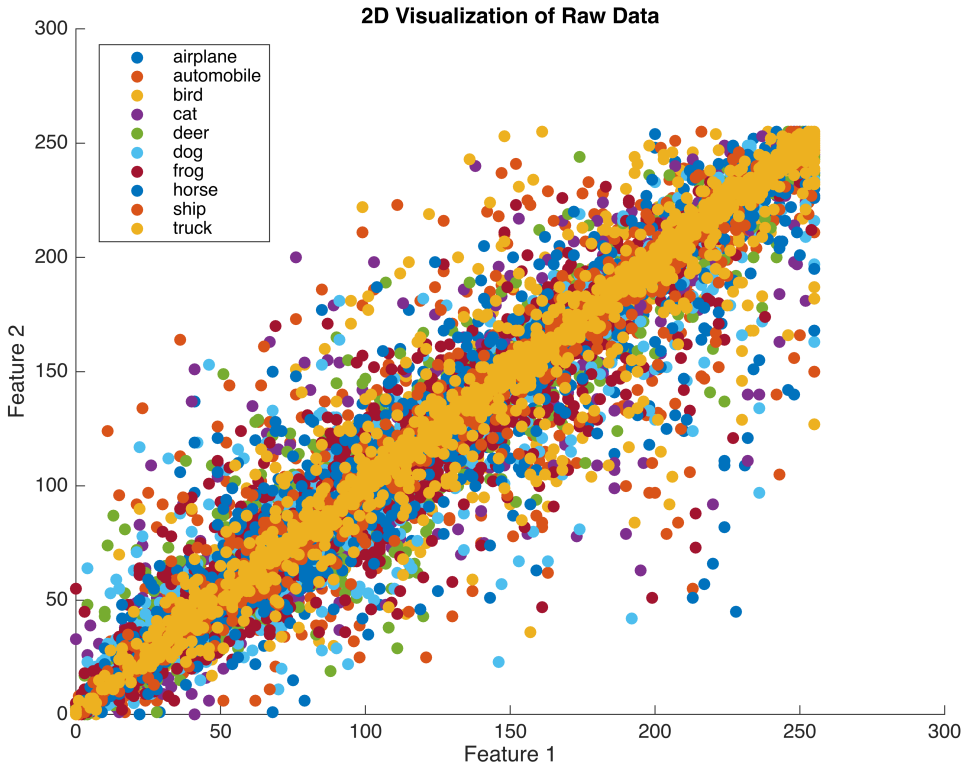
```
airplane: 2000
automobile: 2000
bird: 2000
cat: 2000
deer: 2000
dog: 2000
frog: 2000
horse: 2000
ship: 2000
truck: 2000
```

```
% Calculate mean for each class in the subsampled data
mean_per_class = zeros(num_classes, size(subsample_data, 2));
for i = 1:num_classes
    class_indices = find(subsample_labels == (i - 1)); % Indices of samples
belonging to the current class
    mean_per_class(i, :) = mean(subsample_data(class_indices, :), 1);
end

% Plot the mean image for each class in the subsampled data
figure;
for i = 1:num_classes
    % Reshape mean image data to 32x32x3
    mean_img = reshape(mean_per_class(i, :), [32, 32, 3]);
    % Rotate mean image right by 90 degrees
    mean_img_rotated = imrotate(mean_img, -90);
    % Plot the rotated mean image
    subplot(2, 5, i);
    imshow(uint8(mean_img_rotated));
    title(classes{i});
end
```


```
% Convert subsampled data to double precision
subsample_data_shuffled = double(subsample_data_shuffled);
subsample_labels_shuffled = double(subsample_labels_shuffled);

% Split the subsampled data into train, validate, and test sets (80-10-10
split)
num_samples = size(subsample_data_shuffled, 1);
train_size = round(0.8 * num_samples);
val_size = round(0.1 * num_samples);
test_size = num_samples - train_size - val_size;

train_data = subsample_data_shuffled(1:train_size, :);
train_labels = subsample_labels_shuffled(1:train_size);
val_data = subsample_data_shuffled(train_size+1:train_size+val_size, :);
val_labels = subsample_labels_shuffled(train_size+1:train_size+val_size);
test_data = subsample_data_shuffled(train_size+val_size+1:end, :);
test_labels = subsample_labels_shuffled(train_size+val_size+1:end);

% 1. Raw Data
% Train kNN classifier on raw data
tic;
knn_model_raw = fitcknn(train_data, train_labels, 'NumNeighbors', 20);
train_time_raw = toc;

% Validate kNN classifier on raw data
```

```
tic;
pred_labels_raw_val = predict(knn_model_raw, val_data);
val_time_raw = toc;

% Calculate accuracy and F1-score for validation set
accuracy_raw_val = sum(pred_labels_raw_val == val_labels) /
numel(val_labels);
% Calculate confusion matrix for F1-score calculation
conf_mat_raw_val = confusionmat(val_labels, pred_labels_raw_val);
tp_raw_val = diag(conf_mat_raw_val);
fp_raw_val = sum(conf_mat_raw_val, 1)' - tp_raw_val;
fn_raw_val = sum(conf_mat_raw_val, 2) - tp_raw_val;
tn_raw_val = sum(conf_mat_raw_val(:)) - (tp_raw_val + fp_raw_val +
fn_raw_val);
precision_raw_val = tp_raw_val ./ (tp_raw_val + fp_raw_val);
recall_raw_val = tp_raw_val ./ (tp_raw_val + fn_raw_val);
f1_score_raw_val = 2 * (precision_raw_val .* recall_raw_val) ./
(precision_raw_val + recall_raw_val);

% Test kNN classifier on raw data
tic;
pred_labels_raw_test = predict(knn_model_raw, test_data);
test_time_raw = toc;

% Calculate accuracy and F1-score for test set
accuracy_raw_test = sum(pred_labels_raw_test == test_labels) /
numel(test_labels);
% Calculate confusion matrix for F1-score calculation
conf_mat_raw_test = confusionmat(test_labels, pred_labels_raw_test);
tp_raw_test = diag(conf_mat_raw_test);
fp_raw_test = sum(conf_mat_raw_test, 1)' - tp_raw_test;
fn_raw_test = sum(conf_mat_raw_test, 2) - tp_raw_test;
tn_raw_test = sum(conf_mat_raw_test(:)) - (tp_raw_test + fp_raw_test +
fn_raw_test);
precision_raw_test = tp_raw_test ./ (tp_raw_test + fp_raw_test);
recall_raw_test = tp_raw_test ./ (tp_raw_test + fn_raw_test);
f1_score_raw_test = 2 * (precision_raw_test .* recall_raw_test) ./
(precision_raw_test + recall_raw_test);

% Individual class accuracies for raw data
class_accuracy_raw_val = diag(conf_mat_raw_val) ./ sum(conf_mat_raw_val, 2);
class_accuracy_raw_test = diag(conf_mat_raw_test) ./ sum(conf_mat_raw_test,
2);

% Plot 2D visualization of raw data
figure;
color_map = lines(num_classes);  % Define a color map
for i = 1:num_classes
    class_indices = find(train_labels == (i - 1));
```

```
    scatter(train_data(class_indices, 1), train_data(class_indices, 2), [],
color_map(i, :), 'filled');
    hold on;
end
title('2D Visualization of Raw Data');
xlabel('Feature 1');
ylabel('Feature 2');

% Customize legend entries with class names
class_names_legend = cell(num_classes, 1);
for i = 1:num_classes
    class_names_legend{i} = strcat(classes{i});
end
legend(class_names_legend, 'Location', 'Best');
hold off;
```




```
fprintf('Raw Data:\n');
```

```
Raw Data:
```

```
fprintf('Accuracy - Validation: %.2f%%\n', accuracy_raw_val * 100);
```

```
Accuracy - Validation: 28.10%
```

```
fprintf('F1-score - Validation: %.2f\n', mean(f1_score_raw_val));
```

```
F1-score - Validation: 0.26
```

```
fprintf('Accuracy - Test: %.2f%%\n', accuracy_raw_test * 100);
```

```
Accuracy - Test: 29.80%
```

```
fprintf('F1-score - Test: %.2f\n', mean(f1_score_raw_test));
```

```
F1-score - Test: 0.28
```

```
fprintf('Training Time: %.4f seconds\n', train_time_raw);
```

```
Training Time: 0.1057 seconds
```

```
fprintf('Validation Time: %.4f seconds\n', val_time_raw);
```

```
Validation Time: 10.7433 seconds
```

```
fprintf('Testing Time: %.4f seconds\n\n', test_time_raw);
```

```
Testing Time: 10.2363 seconds
```

```
fprintf('Individual Class Accuracies - Validation:\n');
```

```
Individual Class Accuracies - Validation:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_raw_val(i) * 100);
end
```

```
airplane: 50.24%
automobile: 9.66%
bird: 32.32%
cat: 13.81%
deer: 50.75%
dog: 18.23%
frog: 19.62%
horse: 13.78%
ship: 64.17%
truck: 11.17%
```

```
fprintf('Individual Class Accuracies - Test:\n');
```

```
Individual Class Accuracies - Test:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_raw_test(i) * 100);
end
```

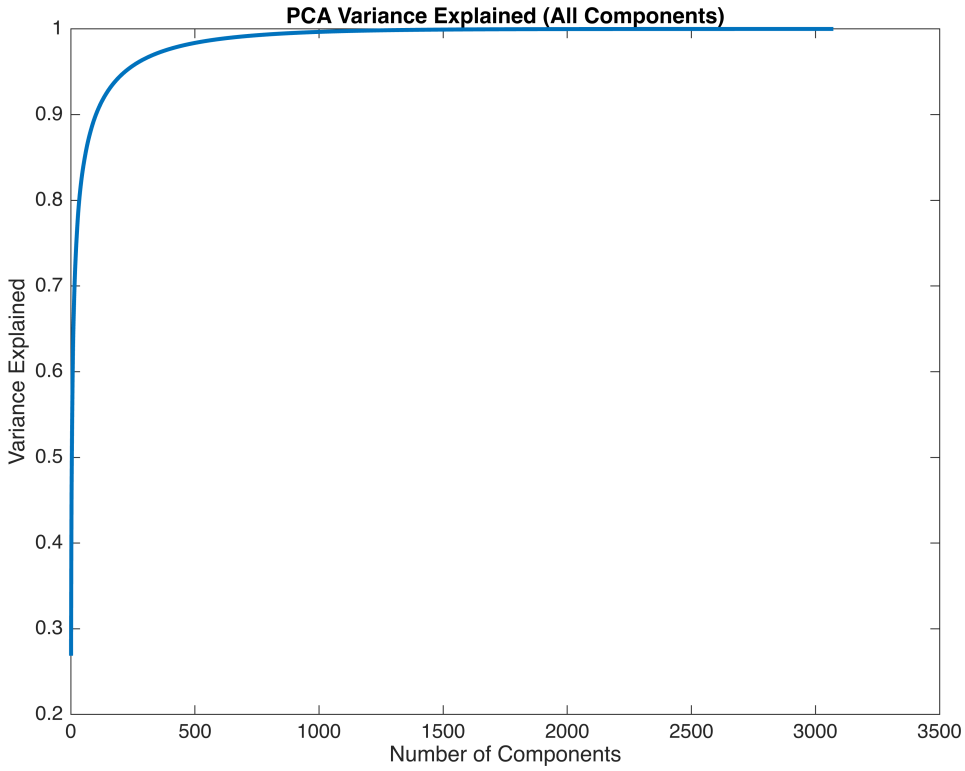
```
airplane: 43.01%
automobile: 13.33%
bird: 34.95%
cat: 10.55%
deer: 53.39%
dog: 16.20%
frog: 24.88%
horse: 19.90%
ship: 70.16%
truck: 10.88%
```

```
```

```
% Convert subsampled data to double precision
subsample_data = double(subsample_data);
subsample_labels = double(subsample_labels);

% Compute the mean of all images
mean_image = mean(subsample_data, 1);
mean_image = cast(mean_image, class(subsample_data)); % Cast mean_image to
the same class as subsample_data

% Subtract the mean from each image
zero_centered_data = subsample_data - mean_image;

% Convert zero_centered_data to single or double precision
zero_centered_data = double(zero_centered_data);

% Calculate the standard deviation of each pixel
std_dev = std(zero_centered_data, 0, 1);

% Divide each pixel by its standard deviation to obtain z-scores
z_scores_data = zero_centered_data ./ std_dev;

% Apply PCA to obtain coefficients
tic;
[data_pca, mapping_pca] = compute_mapping(z_scores_data,
'PCA',size(z_scores_data,2));
```

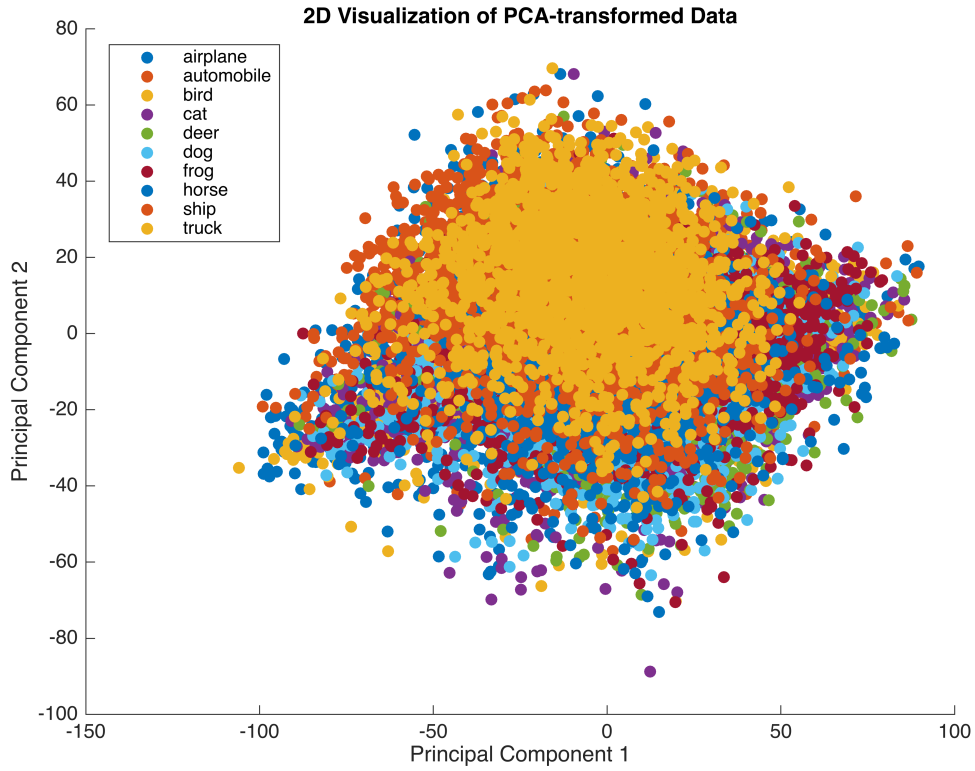
```
Welcome to the Matlab Toolbox for Dimensionality Reduction, version 0.8b (18-April-2012).
   You are free to modify or redistribute this code (for non-commercial purposes), as long as a referer
   to the original author (Laurens van der Maaten, Delft University of Technology) is retained.
   For more information, please visit http://homepage.tudelft.nl/19j49
```

```
pca_time = toc;

% Get explained variance for all components
explained_variance = cumsum(mapping_pca.lambda / sum(mapping_pca.lambda));

% Plot PCA variance for all components
figure;
plot(explained_variance, 'LineWidth', 2);
xlabel('Number of Components');
ylabel('Variance Explained');
title('PCA Variance Explained (All Components)');
```




```
% Consider only the first 100 components
data_pca_100 = data_pca(:, 1:400);

% Split the data into training, validation, and test sets (80-10-10 split)
num_samples = size(data_pca_100, 1);
num_train = round(0.8 * num_samples);  % 80% for training
num_val = round(0.1 * num_samples);    % 10% for validation
num_test = num_samples - num_train - num_val;  % Remaining for testing

% Randomly shuffle the indices
indices = randperm(num_samples);

% Split the indices
train_indices = indices(1:num_train);
val_indices = indices(num_train+1:num_train+num_val);
test_indices = indices(num_train+num_val+1:end);

% Split the data
train_data_pca = data_pca_100(train_indices, :);
train_labels_pca = subsample_labels(train_indices);
val_data_pca = data_pca_100(val_indices, :);
val_labels_pca = subsample_labels(val_indices);
test_data_pca = data_pca_100(test_indices, :);
test_labels_pca = subsample_labels(test_indices);
```

```
% Build the kNN model for PCA
k = 20;
mdl_pca = fitcknn(train_data_pca, train_labels_pca, 'NumNeighbors', k);

% Predict labels for validation and test data
val_predicted_labels_pca = predict(mdl_pca, val_data_pca);
test_predicted_labels_pca = predict(mdl_pca, test_data_pca);

% Calculate accuracy for validation and test data
val_accuracy_pca = sum(val_predicted_labels_pca == val_labels_pca) /
numel(val_labels_pca);
test_accuracy_pca = sum(test_predicted_labels_pca == test_labels_pca) /
numel(test_labels_pca);

% Calculate F1 score for validation and test data
val_f1_score_pca = f1_score(val_labels_pca, val_predicted_labels_pca);
test_f1_score_pca = f1_score(test_labels_pca, test_predicted_labels_pca);

% Individual class accuracies for PCA
conf_mat_pca_val = confusionmat(val_labels_pca, val_predicted_labels_pca);
conf_mat_pca_test = confusionmat(test_labels_pca,
test_predicted_labels_pca);
class_accuracy_pca_val = diag(conf_mat_pca_val) ./ sum(conf_mat_pca_val, 2);
class_accuracy_pca_test = diag(conf_mat_pca_test) ./ sum(conf_mat_pca_test,
2);

% Define a colormap with distinct colors for each class
class_colors = lines(num_classes);

% Plot 2D visualization of PCA-transformed data
figure;
for i = 1:num_classes
    class_indices = find(subsample_labels == (i - 1));
    scatter(data_pca(class_indices, 1), data_pca(class_indices, 2), [],
class_colors(i, :), 'filled');
    hold on;
end
title('2D Visualization of PCA-transformed Data');
xlabel('Principal Component 1');
ylabel('Principal Component 2');
legend(classes, 'Location', 'Best');
hold off;
```




```
fprintf('PCA:\n');
```

```
PCA:
```

```
fprintf('Validation Accuracy: %.2f%%\n', val_accuracy_pca * 100);
```

```
Validation Accuracy: 30.80%
```

```
fprintf('Validation F1 Score: %.2f\n', val_f1_score_pca);
```

```
Validation F1 Score: 0.28
```

```
fprintf('Test Accuracy: %.2f%%\n', test_accuracy_pca * 100);
```

```
Test Accuracy: 31.05%
```

```
fprintf('Test F1 Score: %.2f\n', test_f1_score_pca);
```

```
Test F1 Score: 0.28
```

```
fprintf('PCA Transformation Time: %.4f seconds\n', pca_time);
```

```
PCA Transformation Time: 4.2570 seconds
```

```
fprintf('Individual Class Accuracies - Validation:\n');
```

```
Individual Class Accuracies - Validation:
```

```
for i = 1:num_classes
```

```
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_pca_val(i) * 100);
end
```

```
airplane: 46.15%
automobile: 16.40%
bird: 32.27%
cat: 18.26%
deer: 49.75%
dog: 15.84%
frog: 35.15%
horse: 18.91%
ship: 68.72%
truck: 8.29%
```

```
fprintf('Individual Class Accuracies - Test:\n');
```

```
Individual Class Accuracies - Test:
```

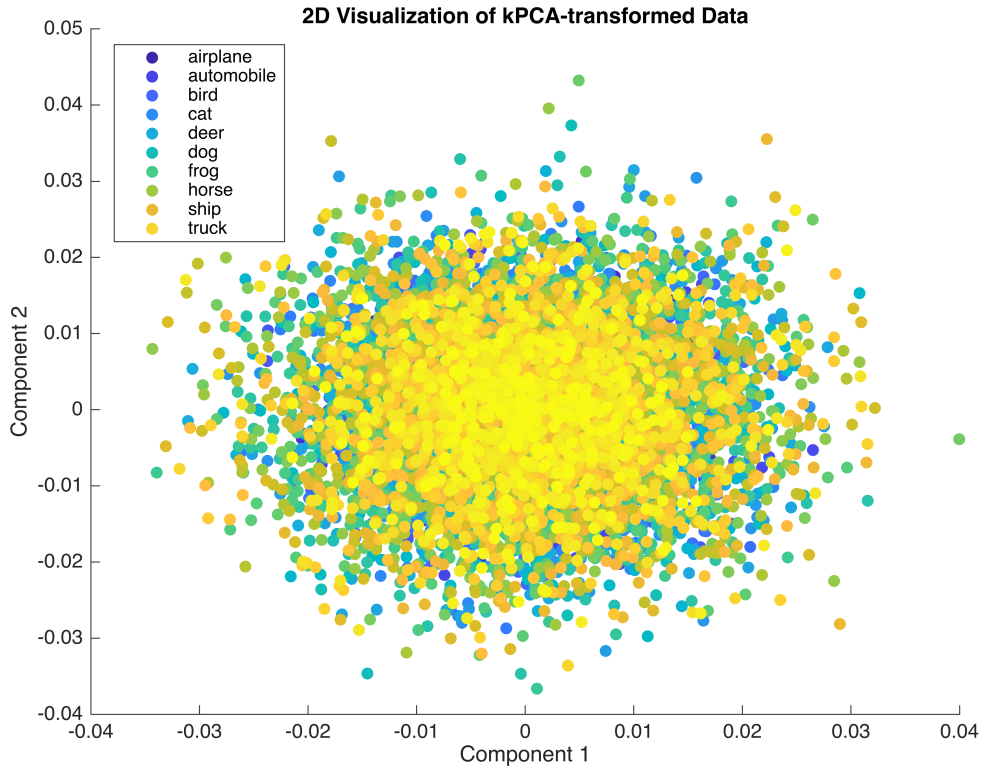
```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_pca_test(i) * 100);
end
```

```
airplane: 42.29%
automobile: 14.43%
bird: 42.25%
cat: 17.00%
deer: 57.65%
dog: 12.15%
frog: 25.79%
horse: 18.81%
ship: 66.35%
truck: 14.15%
```

```


% Apply kernel PCA to obtain coefficients
tic;
[data_kpca, mapping_kpca] = compute_mapping(z_scores_data, 'KernelPCA',400);
```

```
Computing kernel matrix...
Eigenanalysis of kernel matrix...
Computing final embedding...
```

```
kpca_time = toc;

% Plot 2D visualization of kPCA-transformed data
figure;
for i = 1:num_classes
    class_indices = find(subsample_labels == (i - 1));
    scatter(data_kpca(class_indices, 1), data_kpca(class_indices, 2), [],
class_indices, 'filled');
    hold on;
end
title('2D Visualization of kPCA-transformed Data');
xlabel('Component 1');
ylabel('Component 2');
legend(classes, 'Location', 'Best');
hold off;
```




```
% Split the data into training, validation, and test sets (80-10-10 split)
num_samples = size(data_kpca, 1);
num_train = round(0.8 * num_samples);  % 80% for training
num_val = round(0.1 * num_samples);    % 10% for validation
num_test = num_samples - num_train - num_val;  % Remaining for testing

% Randomly shuffle the indices
indices = randperm(num_samples);

% Split the indices
train_indices = indices(1:num_train);
val_indices = indices(num_train+1:num_train+num_val);
test_indices = indices(num_train+num_val+1:end);

% Split the data
train_data_kpca = data_kpca(train_indices, :);
train_labels_kpca = subsample_labels(train_indices);
val_data_kpca = data_kpca(val_indices, :);
val_labels_kpca = subsample_labels(val_indices);
test_data_kpca = data_kpca(test_indices, :);
test_labels_kpca = subsample_labels(test_indices);

% Build the kNN model for kPCA
k = 20;
mdl_kpca = fitcknn(train_data_kpca, train_labels_kpca, 'NumNeighbors', k);
```

```
% Predict labels for validation and test data
val_predicted_labels_kpca = predict(mdl_kpca, val_data_kpca);
test_predicted_labels_kpca = predict(mdl_kpca, test_data_kpca);

% Calculate accuracy for validation and test data
val_accuracy_kpca = sum(val_predicted_labels_kpca == val_labels_kpca) /
numel(val_labels_kpca);
test_accuracy_kpca = sum(test_predicted_labels_kpca == test_labels_kpca) /
numel(test_labels_kpca);

% Calculate F1 score for validation and test data
val_f1_score_kpca = f1_score(val_labels_kpca, val_predicted_labels_kpca);
test_f1_score_kpca = f1_score(test_labels_kpca, test_predicted_labels_kpca);

% Replace NaN values with zeros in F1-score
val_f1_score_kpca(isnan(val_f1_score_kpca)) = 0;
test_f1_score_kpca(isnan(test_f1_score_kpca)) = 0;

% Individual class accuracies for kPCA
conf_mat_kpca_val = confusionmat(val_labels_kpca,
val_predicted_labels_kpca);
conf_mat_kpca_test = confusionmat(test_labels_kpca,
test_predicted_labels_kpca);
class_accuracy_kpca_val = diag(conf_mat_kpca_val) ./ sum(conf_mat_kpca_val,
2);
class_accuracy_kpca_test = diag(conf_mat_kpca_test) ./
sum(conf_mat_kpca_test, 2);

fprintf('kPCA:\n');
```

```
kPCA:
```

```
fprintf('Validation Accuracy: %.2f%%\n', val_accuracy_kpca * 100);
```

```
Validation Accuracy: 14.75%
```

```
fprintf('Validation F1 Score: %.2f\n', val_f1_score_kpca);
```

```
Validation F1 Score: 0.10
```

```
fprintf('Test Accuracy: %.2f%%\n', test_accuracy_kpca * 100);
```

```
Test Accuracy: 13.60%
```

```
fprintf('Test F1 Score: %.2f\n', test_f1_score_kpca);
```

```
Test F1 Score: 0.08
```

```
fprintf('kPCA Transformation Time: %.4f seconds\n', kpca_time);
```

```
kPCA Transformation Time: 233.4558 seconds
```

```
fprintf('Individual Class Accuracies - Validation:\n');
```

```
Individual Class Accuracies - Validation:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_kpca_val(i) * 100);
end
```

```
airplane: 79.90%
automobile: 9.65%
bird: 3.14%
cat: 9.09%
deer: 2.44%
dog: 6.57%
frog: 1.98%
horse: 3.48%
ship: 11.86%
truck: 20.53%
```

```
fprintf('Individual Class Accuracies - Test:\n');
```

```
Individual Class Accuracies - Test:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_kpca_test(i) * 100);
end
```

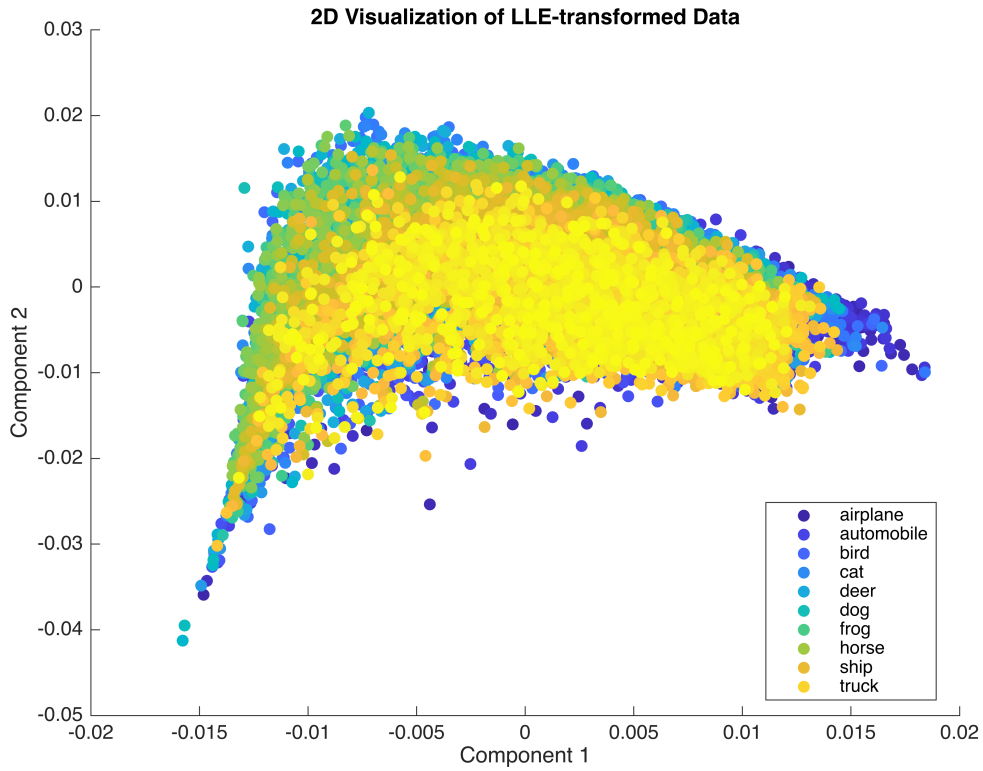
```
airplane: 77.84%
automobile: 6.50%
bird: 4.04%
cat: 3.43%
deer: 2.86%
dog: 2.94%
frog: 2.50%
horse: 5.29%
ship: 11.33%
truck: 20.09%
```

```

% Apply LLE to obtain coefficients
tic;
[data_lle, mapping_lle] = compute_mapping(z_scores_data, 'LLE',15);
```

```
Finding nearest neighbors...
Compute reconstruction weights...
Compute embedding (solve eigenproblem)...
Warning: Ignoring issym field in the options structure since the first input is not a function
handle.
Warning: The first input matrix, shifted by sigma, is close to singular or badly scaled (RCOND =
2.411964e-19) and results may be inaccurate. Consider specifying a perturbed numeric sigma value to
improve the condition of the matrix.
```

```
lle_time = toc;

% Plot 2D visualization of LLE-transformed data
figure;
for i = 1:num_classes
    class_indices = find(subsample_labels == (i - 1));
    scatter(data_lle(class_indices, 1), data_lle(class_indices, 2), [],
class_indices, 'filled');
```

```
    hold on;
end
title('2D Visualization of LLE-transformed Data');
xlabel('Component 1');
ylabel('Component 2');
legend(classes, 'Location', 'Best');
hold off;
```




```
% Split the data into training, validation, and test sets (80-10-10 split)
num_samples = size(data_lle, 1);
num_train = round(0.8 * num_samples);  % 80% for training
num_val = round(0.1 * num_samples);    % 10% for validation
num_test = num_samples - num_train - num_val;  % Remaining for testing

% Randomly shuffle the indices
indices = randperm(num_samples);

% Split the indices
train_indices = indices(1:num_train);
val_indices = indices(num_train+1:num_train+num_val);
test_indices = indices(num_train+num_val+1:end);

% Split the data
train_data_lle = data_lle(train_indices, :);
train_labels_lle = subsample_labels(train_indices);
val_data_lle = data_lle(val_indices, :);
```

```
val_labels_lle = subsample_labels(val_indices);
test_data_lle = data_lle(test_indices, :);
test_labels_lle = subsample_labels(test_indices);

% Build the kNN model for LLE
k = 20;
mdl_lle = fitcknn(train_data_lle, train_labels_lle, 'NumNeighbors', k);

% Predict labels for validation and test data
val_predicted_labels_lle = predict(mdl_lle, val_data_lle);
test_predicted_labels_lle = predict(mdl_lle, test_data_lle);

% Calculate accuracy for validation and test data
val_accuracy_lle = sum(val_predicted_labels_lle == val_labels_lle) /
numel(val_labels_lle);
test_accuracy_lle = sum(test_predicted_labels_lle == test_labels_lle) /
numel(test_labels_lle);

% Calculate F1 score for validation and test data
val_f1_score_lle = f1_score(val_labels_lle, val_predicted_labels_lle);
test_f1_score_lle = f1_score(test_labels_lle, test_predicted_labels_lle);

% Individual class accuracies for LLE
conf_mat_lle_val = confusionmat(val_labels_lle, val_predicted_labels_lle);
conf_mat_lle_test = confusionmat(test_labels_lle,
test_predicted_labels_lle);
class_accuracy_lle_val = diag(conf_mat_lle_val) ./ sum(conf_mat_lle_val, 2);
class_accuracy_lle_test = diag(conf_mat_lle_test) ./ sum(conf_mat_lle_test,
2);

fprintf('LLE:\n');
```

```
LLE:
```

```
fprintf('Validation Accuracy: %.2f%%\n', val_accuracy_lle * 100);
```

```
Validation Accuracy: 29.35%
```

```
fprintf('Validation F1 Score: %.2f\n', val_f1_score_lle);
```

```
Validation F1 Score: 0.28
```

```
fprintf('Test Accuracy: %.2f%%\n', test_accuracy_lle * 100);
```

```
Test Accuracy: 26.80%
```

```
fprintf('Test F1 Score: %.2f\n', test_f1_score_lle);
```

```
Test F1 Score: 0.25
```

```
fprintf('LLE Transformation Time: %.4f seconds\n', lle_time);
```

```
LLE Transformation Time: 65.5507 seconds
```

```
fprintf('Individual Class Accuracies - Validation:\n');
```

```
Individual Class Accuracies - Validation:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_lle_val(i) * 100);
end
```

```
airplane: 37.31%
automobile: 26.79%
bird: 19.18%
cat: 20.11%
deer: 18.72%
dog: 26.52%
frog: 37.29%
horse: 28.28%
ship: 45.27%
truck: 35.45%
```

```
fprintf('Individual Class Accuracies - Test:\n');
```

```
Individual Class Accuracies - Test:
```

```
for i = 1:num_classes
    fprintf('%s: %.2f%%\n', classes{i}, class_accuracy_lle_test(i) * 100);
end
```

```
airplane: 39.13%
automobile: 23.65%
bird: 14.95%
cat: 16.04%
deer: 23.00%
dog: 25.12%
frog: 31.48%
horse: 22.05%
ship: 38.38%
truck: 32.64%
```

```

% Function to calculate F1 score
```

```
function f1 = f1_score(labels_true, labels_pred)
    num_classes = max(labels_true);
    f1 = zeros(num_classes, 1);
    for i = 1:num_classes
        tp = sum(labels_true == i & labels_pred == i);
        fp = sum(labels_true ~= i & labels_pred == i);
        fn = sum(labels_true == i & labels_pred ~= i);
        precision = tp / (tp + fp);
        recall = tp / (tp + fn);
        f1(i) = 2 * (precision * recall) / (precision + recall);
    end
    f1 = mean(f1);
end
```