Problem Set 4

By Steven Cao *(written using MATLAB R2019b)*

Dependencies: lhist.m

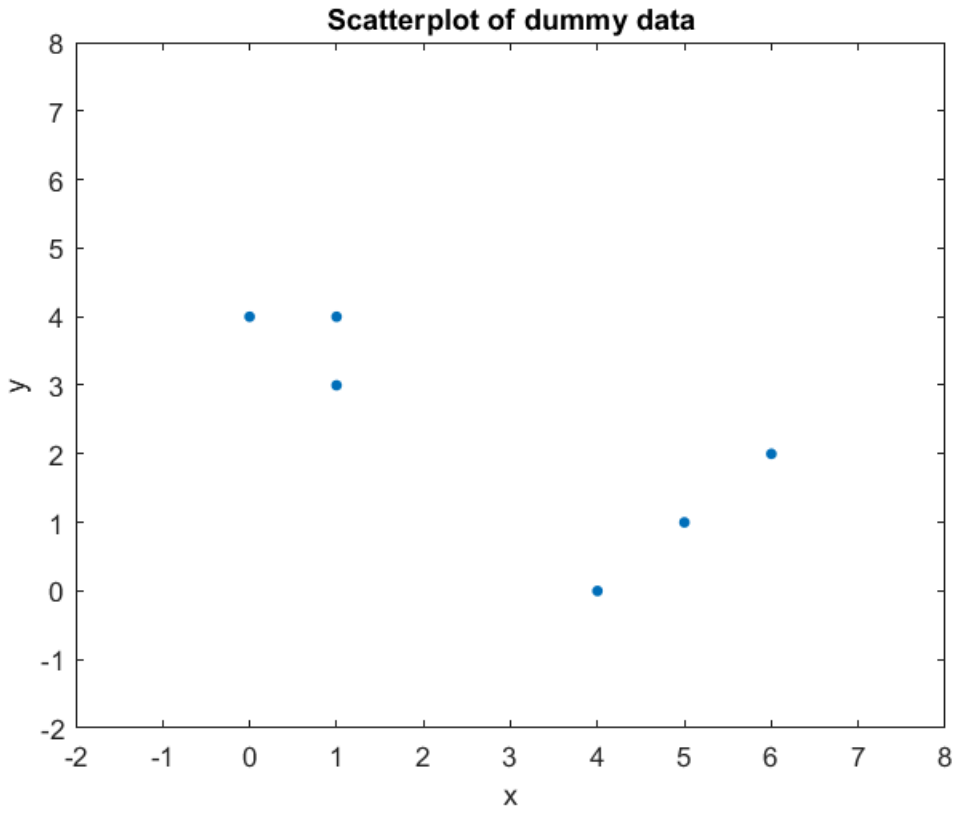
Part 1 - Performing k-means by hand

Part 1, Problem 1 - Create dummy data and plot it

```
% Create dummy data.
dummy = [ [1;1;0;5;6;4] , [4;3;4;1;2;0] ]
```

```
dummy = 6×2
     1     4
     1     3
     0     4
     5     1
     6     2
     4     0
```

```
% Create some helper variables for convenience.
[numObservations, numFeatures] = size(dummy);
x = dummy(:,1);
y = dummy(:,2);

% Make a scatterplot of our data.
figure;
plot(x, y, '.', 'MarkerSize', 12);
title('Scatterplot of dummy data');
xlabel('x');
ylabel('y');
xlim([-2,8]);
ylim([-2,8]);
```



Part 1, Problem 2 - Give random assignments to data and plot it

```
% Set seed first.
s = RandStream('mt19937ar','Seed',123);
RandStream.setGlobalStream(s);

% Set up the different classes that we'll be using (i.e. an observation can be blue or red).
class_assignments = ["blue", "red"];
class_assignments_index = [1, 2]; % i.e. blue==1, red==2
numClasses = length(class_assignments); % also the number of centroids

% Randomly assign a class to our observations.
random_integers = randi( class_assignments_index , numObservations , 1 );
dummy_assignments = class_assignments(random_integers)'
```

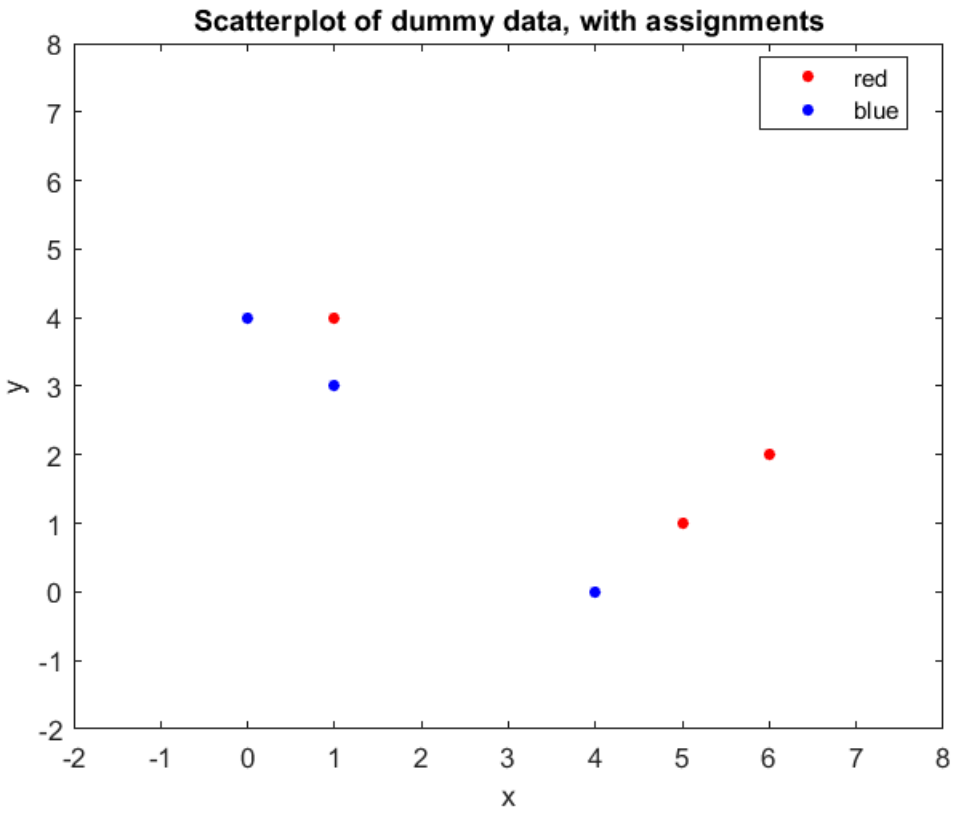
```
dummy_assignments = 6×1 string array
"red"
"blue"
"blue"
"red"
"red"
"blue"
```

```
% This is just to list what's going on (we're not storing this as a variable).
table(x, y, dummy_assignments)
```

```
ans = 6×3 table
```

	x	y	dummy_assignments
1	1	4	"red"
2	1	3	"blue"
3	0	4	"blue"
4	5	1	"red"
5	6	2	"red"
6	4	0	"blue"

```
% Now plot our assigned data with a group-based scatterplot.
figure;
h = gscatter(x, y, dummy_assignments); % h == handle to the graphical object
title('Scatterplot of dummy data, with assignments');
xlabel('x');
ylabel('y');
xlim([-2,8]);
ylim([-2,8]);
% (extra tweaking to *guarantee* that the group colours will match the names of the group label
clusterName1 = h(1).DisplayName;
clusterName2 = h(2).DisplayName;
h(1).Color = clusterName1;
h(2).Color = clusterName2;
```



Part 1, Problem 3 - Compute the centroid for each cluster

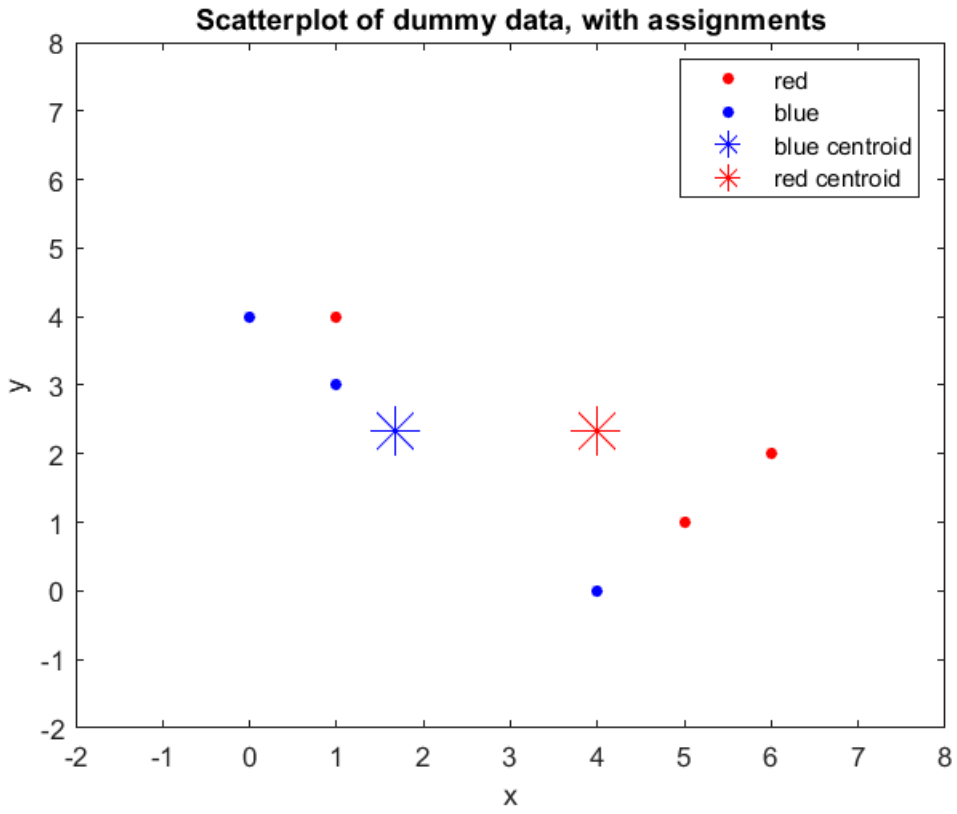
```
% Note: I'm initialising the centroids a bit differently here from other implementations.
% Instead of choosing a random observation to be the centroid, I'm initialising the centroids
% to be the averages of their respective (and randomly-assigned) observations.

% Get the x-values and y-values for each cluster (blue and red).
whichObservations_areBlue = contains(dummy_assignments, "blue");
whichObservations_areRed  = contains(dummy_assignments, "red");
x_blueOnly = x(whichObservations_areBlue);
y_blueOnly = y(whichObservations_areBlue);
x_redOnly  = x(whichObservations_areRed);
y_redOnly  = y(whichObservations_areRed);

% Get the x-average and y-average for each cluster.
x_blueAverage = mean(x_blueOnly);
y_blueAverage = mean(y_blueOnly);
x_redAverage  = mean(x_redOnly);
y_redAverage  = mean(y_redOnly);

% The x-average and y-average form the x- and y-coordinates of our centroids.
centroidCoordinates_blue = [x_blueAverage, y_blueAverage]
```

```
centroidCoordinates_blue = 1×2
    1.6667    2.3333
```

```
centroidCoordinates_red = [x_redAverage, y_redAverage]
```

```
centroidCoordinates_red = 1×2
    4.0000    2.3333
```

```
% (store both centroid coordinates in one variable which we'll use later on)
centroidCoordinates = [centroidCoordinates_blue; centroidCoordinates_red];

% Some extra visualisation, just to see if our calculations worked out.
hold on;
plot(x_blueAverage, y_blueAverage, 'b*', 'MarkerSize', 18, 'DisplayName', 'blue centroid');
plot(x_redAverage , y_redAverage, 'r*' , 'MarkerSize', 18, 'DisplayName', 'red centroid' );
hold off;
```




```
% The centroids seem to obey "center of mass", so looks like it did work out.
```

Part 1, Problem 4 - Reassign observations based on current centroid coordinates

```
% Preallocate a new vector for our new assignments.
dummy_assignments2 = strings(numObservations,1); % strings() generates a string array of empty

% For each observation, compute its distance from all centroids.
% Then reassign based on closest centroid.
for i = 1:numObservations

    currentObservation = dummy(i,:);

    distanceFromEachCentroid = nan(1,numClasses); % initialise helper array
    % Calculate distance of current observation from each centroid (i.e. each class).
    for j = 1:numClasses
        xDiff = currentObservation(1) - centroidCoordinates(j,1);
        yDiff = currentObservation(2) - centroidCoordinates(j,2);

        distance = sqrt(xDiff^2 + yDiff^2); % Euclidean

        distanceFromEachCentroid(j) = distance;
    end

    % Get the closest centroid's class name ("blue" or "red").
    [~,indexOfClosestCentroid] = min(distanceFromEachCentroid);
    classOf_closestCentroid = class_assignments(indexOfClosestCentroid);
```

```
    % Assign the current observation to that class.
    dummy_assignments2(i) = classOf_closestCentroid;

end

% List our new assignments.
dummy_assignments2
```

```
dummy_assignments2 = 6×1 string array
"blue"
"blue"
"blue"
"red"
"red"
"red"
```

```
% Make a scatterplot of our new assignments.
figure;
hold on;
h = gscatter(x, y, dummy_assignments2);
title('Scatterplot of dummy data, with assignments, again');
xlabel('x');
ylabel('y');
xlim([-2,8]);
ylim([-2,8]);
% (extra tweaking to *guarantee* that the group colours will match the names of the group label
clusterName1 = h(1).DisplayName;
clusterName2 = h(2).DisplayName;
h(1).Color = clusterName1;
h(2).Color = clusterName2;

% Also calculate and plot the centroids while we're at it.
whichObservations_areBlue2 = contains(dummy_assignments2, "blue");
whichObservations_areRed2 = contains(dummy_assignments2, "red");
x_blueOnly2 = x(whichObservations_areBlue2);
y_blueOnly2 = y(whichObservations_areBlue2);
x_redOnly2 = x(whichObservations_areRed2);
y_redOnly2 = y(whichObservations_areRed2);
x_blueAverage2 = mean(x_blueOnly2);
y_blueAverage2 = mean(y_blueOnly2);
x_redAverage2 = mean(x_redOnly2);
y_redAverage2 = mean(y_redOnly2);
centroidCoordinates_blue2 = [x_blueAverage2, y_blueAverage2]
```

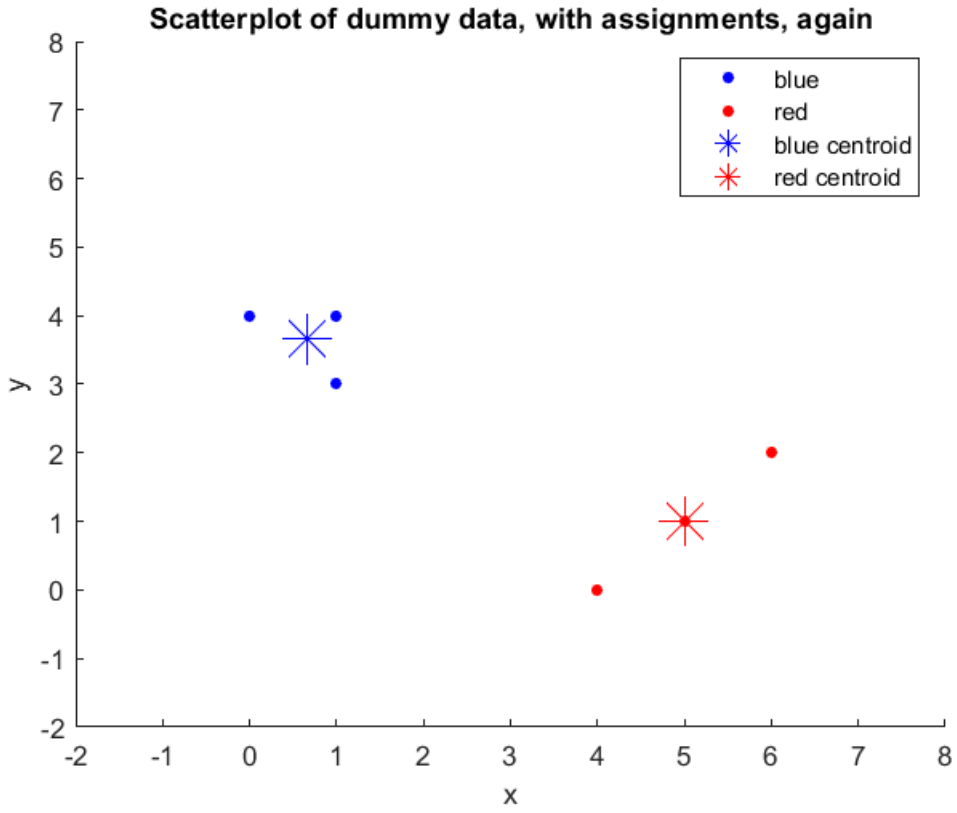
```
centroidCoordinates_blue2 = 1×2
    0.6667    3.6667
```

```
centroidCoordinates_red2 = [x_redAverage2, y_redAverage2]
```

```
centroidCoordinates_red2 = 1×2
     5     1
```

```
centroidCoordinates2 = [centroidCoordinates_blue2; centroidCoordinates_red2];

% Plot centroids.
```

```
plot(x_blueAverage2, y_blueAverage2, 'b*', 'MarkerSize', 18, 'DisplayName', 'blue centroid');
plot(x_redAverage2, y_redAverage2, 'r*', 'MarkerSize', 18, 'DisplayName', 'red centroid');

hold off;
```



Part 1, Problem 5 - Repeat iterations of k-means until locally-optimal solution is reached

Well... looking at the plot, it really only took one additional iteration (after initialising the way I did) to get our centroids to the optimal spot. So no further iterations are needed.

Now, if we had to judge this without resorting to plots or visual assessments in the first place, then the way to go about this would be to look at dummy_assignments and see if the list of assignments is changing after a certain number of iterations (e.g. one of the rows changes from "red" to blue"). If we go one iteration without any assignments changing, then we've hit a locally-optimal solution.

Part 1, Problem 6 - Plot the converged solution

Already done, see the plot above.

Part 2 - Clustering state legslative professionalism

Part 2, Problem 1 - Load the data

```
% Boom, done.
legprof_dataset = readtable('legprof.csv');
```

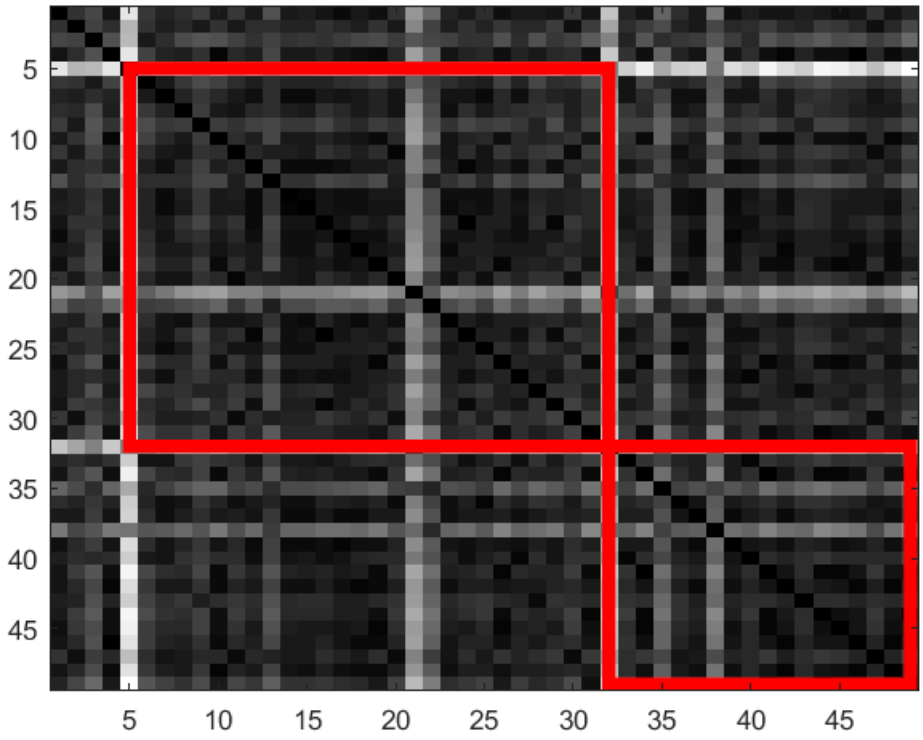
Part 2, Problem 2 - Data strangling

```
% Generate some helper variables.
labelVar = {'stateabv'};
filterVar = {'sessid'};
inputVar = {'t_slength', 'slength', 'salary_real', 'expend'};

% Generate a list of our labels.
%legprof_labels = unique(legprof_dataset{:,labelVar}); % doesn't work because it alphabetises b
legprof_labels = legprof_dataset{:,labelVar}; % contains duplicate entries, will need to deal w

% Generate subsets of our data (one containing only session ID, the other containing the 4 feat
legprof_filter = legprof_dataset{:,filterVar};
legprof_subset = legprof_dataset(:,inputVar);

% Filter our subset of features using the session ID feature.
validRowIndex = contains(legprof_filter, '2009/10');
legprof_subsetFiltered = legprof_subset(validRowIndex,:);

% While we're at it, conveniently get rid of duplicate state labels.
legprof_labels = legprof_labels(validRowIndex,:);

% Look for those rows that had NAs (encoded as the string, "NA", which complicates things).
[row_NAs,~] = find(contains( string(table2cell(legprof_subsetFiltered)) , "NA"));
row_NAs = unique(row_NAs);
% If the row has at least one NA, then remove that row...
legprof_subsetFiltered(row_NAs,:) = [];
% ...and *also* generate its corresponding array of labels.
legprof_labelsFiltered = legprof_labels;
legprof_labelsFiltered(row_NAs) = [];

% Do some object type conversion on the data subset we will be working with.
% (Because the table is almost entirely composed of strings rather than numbers.)
legprof_subsetFiltered = str2double(table2array(legprof_subsetFiltered));
% z-score it to standardise units.
legprof_subsetFiltered = zscore(legprof_subsetFiltered);
```

Part 2, Problem 3 - Diagnose clusterability via ODI

```
% Get pairwise distance values.
legprof_distanceMatrix = pdist(legprof_subsetFiltered);
% Turn it into a dissimilarity matrix.
legprof_distanceMatrix = squareform(legprof_distanceMatrix);

% Plot it.
figure;
imagesc(legprof_distanceMatrix);
colormap(gray);

% Also overlay some red rectangles to specify where I think clusters are indicated.
hold on;
rectangle('Position', [5,5   27 27], 'EdgeColor', 'r', 'LineWidth', 5);
rectangle('Position', [32,32 17 17], 'EdgeColor', 'r', 'LineWidth', 5);
```

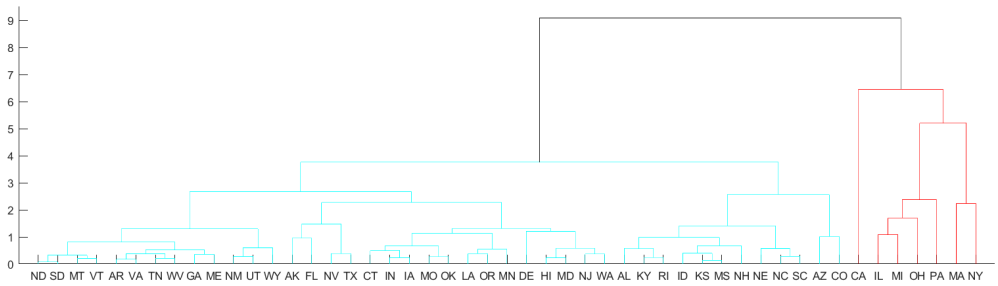
```
hold off;
```



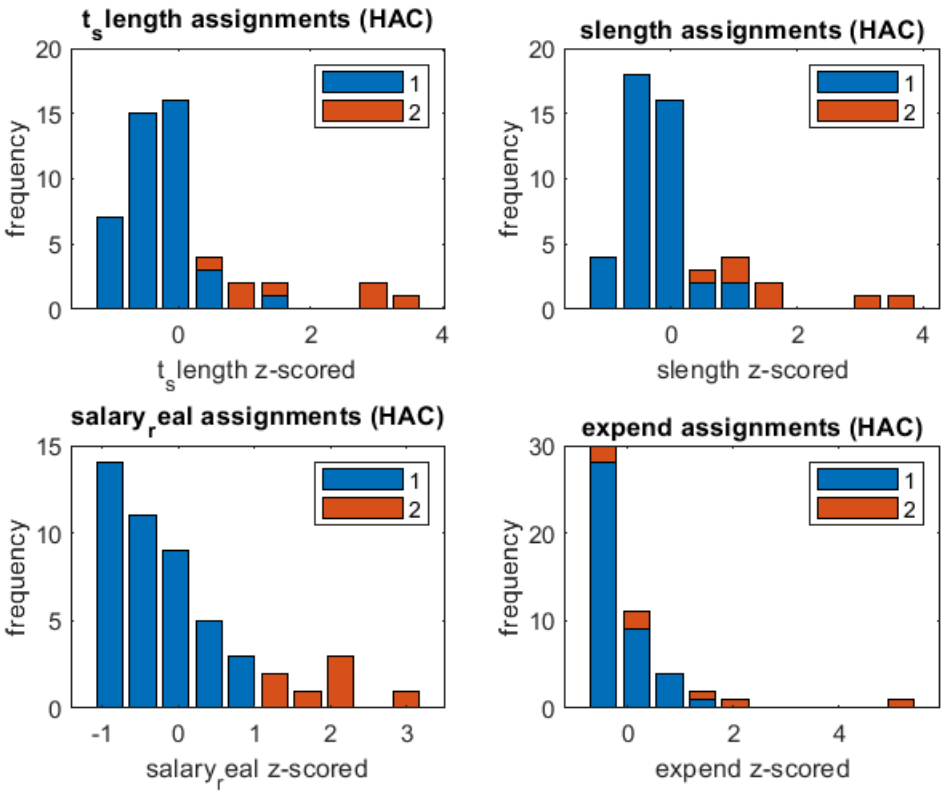
Overall, the clusterability does not look too good - there is nothing clear-cut. But maybe two clusters would be able to work here.

Part 2, Problem 4 - HAC

```
% Make model.
model_HAC = linkage(legprof_subsetFiltered,'Complete','Euclidean');

% Plot dendrogram.
chopAtHeight = 7.0;
figure('Position', [0 250 1600 400]);
dendrogram(model_HAC, 0, 'ColorThreshold', chopAtHeight); % the second argument == cap on leaf
% (replace x-axis labels with appropriate state names)
currentGraph = gca; % gca is a hardcoded variable native to MATLAB which serves as a pointer to
currentGraph.XTickLabel = legprof_labelsFiltered(str2num(currentGraph.XTickLabel));
```

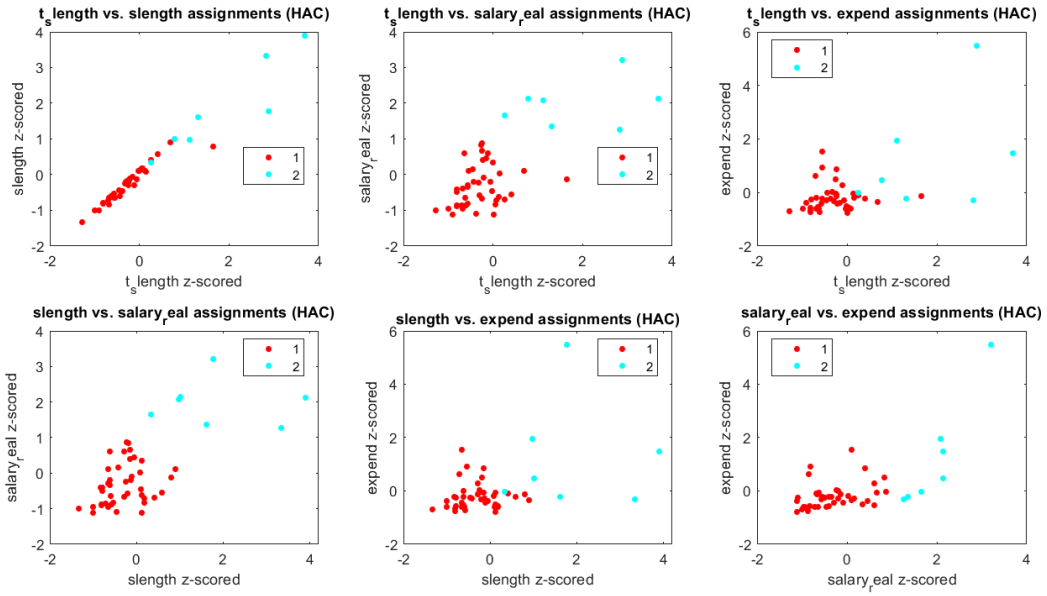


```
% Get the class assignments.
assignments_HAC = cluster(model_HAC,'Maxclust',2);
% (cast assignments from integers to strings, for convenience)
assignments_HAC = num2str(assignments_HAC);

% Lattice of histograms.
figure;
for i=1:length(inputVar)
    subplot(2,2,i);
    lhist( legprof_subsetFiltered(:,i) , 10 , assignments_HAC ); % the '10' refers to the numbe
    title(  [inputVar{i}, ' assignments (HAC)'] );
    xlabel( [inputVar{i}, ' z-scored'] );
    ylabel( 'frequency' );
end
```




```
% Lattice of scatterplots.
allGraphCombinations = nchoosek(1:4,2);
numGraphCombinations = 6;
figure('Position', [10,10,1200,600]);
for i=1:numGraphCombinations
    subplot(2,3,i);
    variableX = legprof_subsetFiltered( : , allGraphCombinations(i,1) );
    variableY = legprof_subsetFiltered( : , allGraphCombinations(i,2) );
    variableXName = inputVar{ allGraphCombinations(i,1) };
    variableYName = inputVar{ allGraphCombinations(i,2) };
    gscatter( variableX , variableY , assignments_HAC );
    title(  [variableXName, ' vs. ', variableYName ' assignments (HAC)'] );
    xlabel( [variableXName, ' z-scored'] );
```

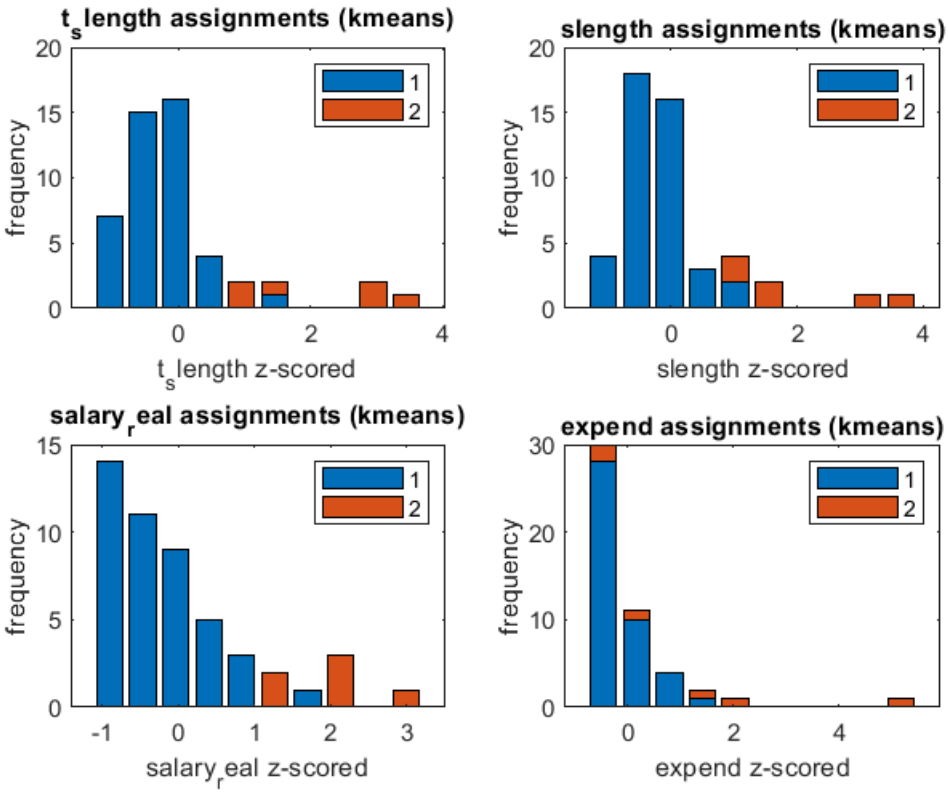
```
    ylabel( [variableYName, ' z-scored'] );
end
```



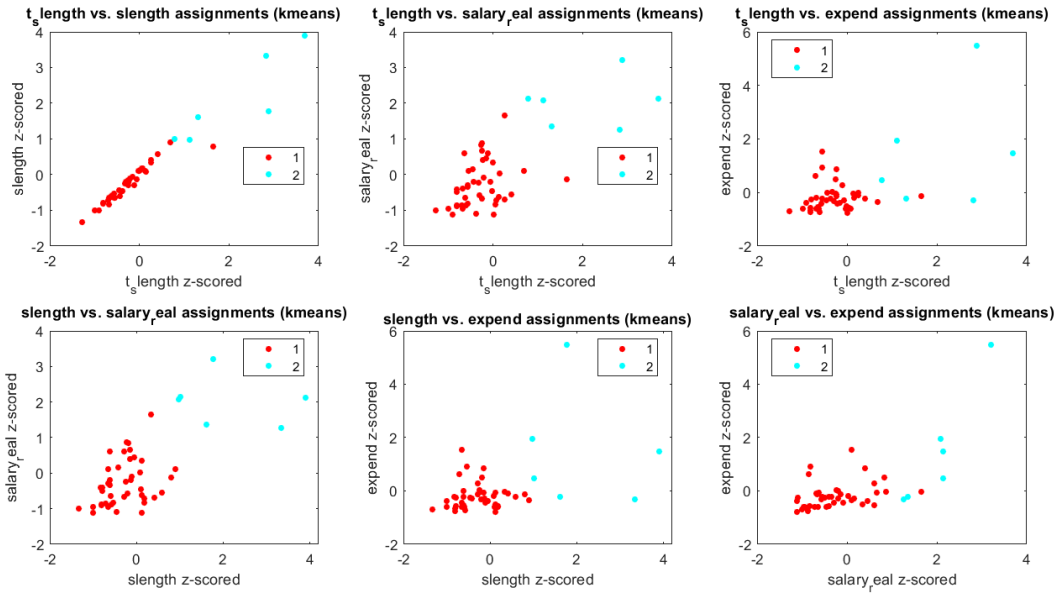
Generally, there are two clusters, with one cluster having far fewer observations than the other. It seems as though the cluster with fewer observations is capturing all of the "outliers" (outliers regarding all four feature variables). Additionally, the states in the smaller cluster (e.g. California, New York, Illinois) tend to have cities, and bigger ones at that, which also implies that they have money to throw around (i.e. higher expenditures and salaries). Also, t_slength and s_length happen to be highly correlated.

Part 2, Problem 5 - k-means

```
% Get the class assignments.
assignments_kmeans = kmeans(legprof_subsetFiltered,2);
assignments_kmeans = num2str(assignments_kmeans);

% Lattice of histograms.
figure;
for i=1:length(inputVar)
    subplot(2,2,i);
    lhist( legprof_subsetFiltered(:,i) , 10 , assignments_kmeans ); % the '10' refers to the nu
    title(  [inputVar{i}, ' assignments (kmeans)'] );
    xlabel( [inputVar{i}, ' z-scored'] );
    ylabel( 'frequency' );
end
```




```
% Lattice of scatterplots.
allGraphCombinations = nchoosek(1:4,2);
numGraphCombinations = 6;
figure('Position', [10,10,1200,600]);
for i=1:numGraphCombinations
    subplot(2,3,i);
    variableX = legprof_subsetFiltered( : , allGraphCombinations(i,1) );
    variableY = legprof_subsetFiltered( : , allGraphCombinations(i,2) );
    variableXName = inputVar{ allGraphCombinations(i,1) };
    variableYName = inputVar{ allGraphCombinations(i,2) };
    gscatter( variableX , variableY , assignments_kmeans );
    title(  [variableXName, ' vs. ', variableYName ' assignments (kmeans)'] );
    xlabel( [variableXName, ' z-scored'] );
    ylabel( [variableYName, ' z-scored'] );
end
```

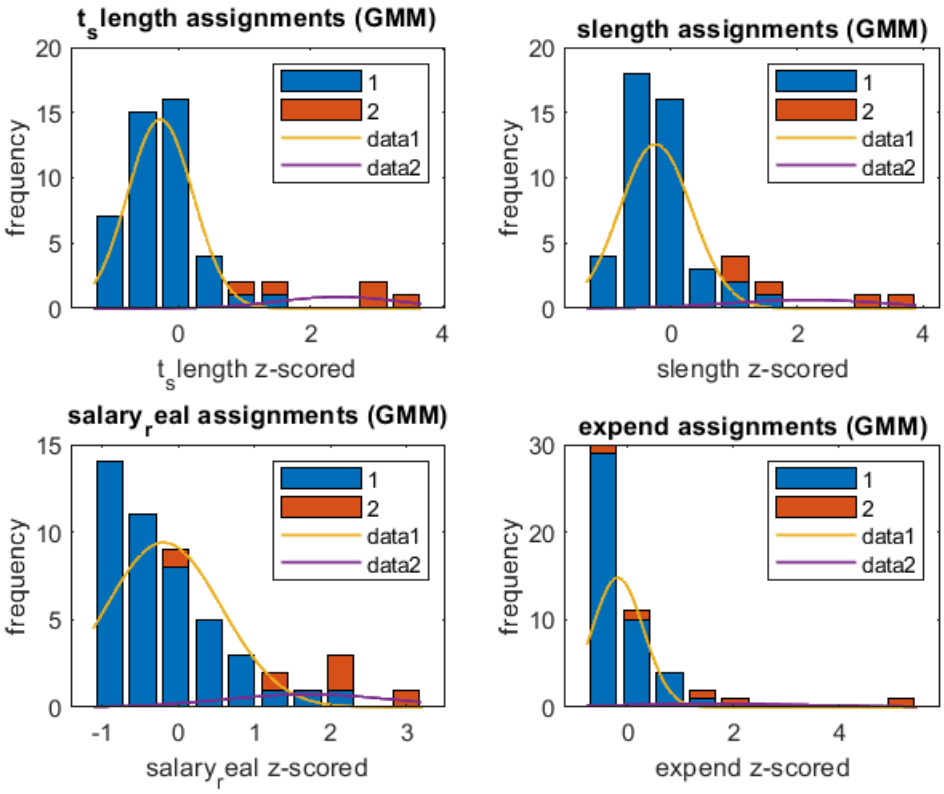


This output generally corroborates the output of the HAC algorithm. One notable thing, however, is that for all scatterplots *except* for the two on the right (expend vs salary_real & expend vs t_slength), k-means seems to have very cleanly cut through the data to make its classifications.

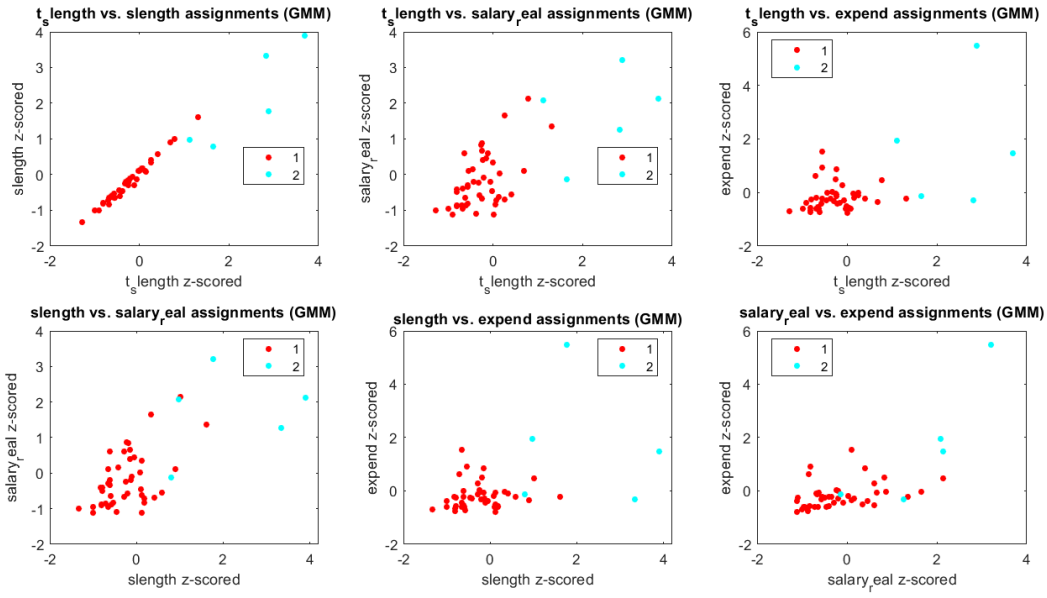
Part 2, Problem 6 - GMM

```
% Make model.
model_GMM = fitgmdist(legprof_subsetFiltered,2); % yes, this uses the EM algorithm

% Get the class assignments.
% (Based on whichever cluster has the higher posterior probability for that observation.)
assignments_GMM = cluster( model_GMM, legprof_subsetFiltered );
assignments_GMM = num2str(assignments_GMM);

% Lattice of histograms.
figure;
for i=1:length(inputVar)
    subplot(2,2,i);
    lhist( legprof_subsetFiltered(:,i) , 10 , assignments_GMM ); % the '10' refers to the numbe
    title(  [inputVar{i}, ' assignments (GMM)'] );
    xlabel( [inputVar{i}, ' z-scored'] );
    ylabel( 'frequency' );
    % (plot curves of each component too)
    hold on;
    minVal=min(legprof_subsetFiltered(:,i));
    maxVal=max(legprof_subsetFiltered(:,i));
    xSpace = minVal:0.01:maxVal;
    rescaleFactor = 20;
    component1 = pdf('Normal', xSpace, model_GMM.mu(1,i), sqrt(model_GMM.Sigma(i,i,1)));
    component2 = pdf('Normal', xSpace, model_GMM.mu(2,i), sqrt(model_GMM.Sigma(i,i,2)));
    plot(xSpace, component1*model_GMM.ComponentProportion(1)*rescaleFactor, 'LineWidth', 1);
    plot(xSpace, component2*model_GMM.ComponentProportion(2)*rescaleFactor, 'LineWidth', 1);
    hold off;
```

```
end
```




```
% Lattice of scatterplots.
allGraphCombinations = nchoosek(1:4,2);
numGraphCombinations = 6;
figure('Position', [10,10,1200,600]);
for i=1:numGraphCombinations
    subplot(2,3,i);
    variableX = legprof_subsetFiltered( : , allGraphCombinations(i,1) );
    variableY = legprof_subsetFiltered( : , allGraphCombinations(i,2) );
    variableXName = inputVar{ allGraphCombinations(i,1) };
    variableYName = inputVar{ allGraphCombinations(i,2) };
    gscatter( variableX , variableY , assignments_GMM );
    title(  [variableXName, ' vs. ', variableYName ' assignments (GMM)'] );
    xlabel( [variableXName, ' z-scored'] );
    ylabel( [variableYName, ' z-scored'] );
%   (contour-plotting is broken)
%     hold on
%     gmPDF = @(x,y)reshape(pdf(model_GMM,[x(:) y(:)]),size(x));
%     fcontour(gmPDF)
%     hold off
end
```



Once again, the output of applying GMM corroborates prior outputs. Of note, however, is that this model has allowed for much more overlap in the clusters (as evidenced by the scatterplots) than the previous algorithms had. Of another note is that one of the components (as seen in the histograms) is very wide but has a low height, suggesting that that component is attempting to capture all of the outliers, demonstrating similar clustering behaviour to the previous algorithms.

Part 2, Problem 7 - Compare outputs of the different clustering algorithms

Cf. the plots rendered above. What they all have in common is one cluster capturing the majority of the states, with the other cluster capturing those states which are "outliers" (i.e. have a relatively high value in all of our features). While there are some minor differences in the output, as expected, none seem to be considerable enough to warrant further comment beyond what has already been said.

Part 2, Problem 8 - Internal validation

```
% Set up some helper variables.
klist = 2:10; % range of k to test
name_HAC = 'linkage';
name_kmeans = 'kmeans';
name_GMM = 'gmdistribution';

% I know the problem asks for just one index, but I threw two additional indices
% (Gap and Davies-Bouldin) just to see what would happen.
% For the purposes of the next problem, I'm only going to use the silhouette index.

% Silhouette
validateSilhouette_HAC    = evalclusters(legprof_subsetFiltered, name_HAC    , 'silhouette', 'KL
```

```
validateSilhouette_HAC =
  SilhouetteEvaluation with properties:

    NumObservations: 49
```

```
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.7973 0.7992 0.7440 0.3686 0.4166 0.4402 0.4614 0.4516 0.4680]
       OptimalK: 3
```

```
validateSilhouette_kmeans = evalclusters(legprof_subsetFiltered, name_kmeans, 'silhouette', 'KL
```

```
validateSilhouette_kmeans =
  SilhouetteEvaluation with properties:

    NumObservations: 49
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.8241 0.8254 0.7183 0.4561 0.5309 0.5454 0.5656 0.5461 0.4601]
       OptimalK: 3
```

```
validateSilhouette_GMM    = evalclusters(legprof_subsetFiltered, name_GMM   , 'silhouette', 'KL
```

```
Warning: Failed to converge in 100 iterations during replicate 5 for gmdistribution with 9 components
validateSilhouette_GMM =
  SilhouetteEvaluation with properties:

    NumObservations: 49
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.8107 0.7815 0.5189 0.4756 0.3231 0.3270 0.1023 0.1979 0.3583]
       OptimalK: 2
```

```
% Gap
validateGap_HAC    = evalclusters(legprof_subsetFiltered, name_HAC   , 'gap', 'KList', klist)
```

```
validateGap_HAC =
  GapEvaluation with properties:

    NumObservations: 49
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.8926 0.8595 0.8390 0.8628 0.8964 0.9335 0.9683 1.0061 1.0575]
       OptimalK: 9
```

```
validateGap_kmeans = evalclusters(legprof_subsetFiltered, name_kmeans, 'gap', 'KList', klist)
```

```
validateGap_kmeans =
  GapEvaluation with properties:

    NumObservations: 49
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.8951 0.8210 0.7737 0.8755 0.8613 1.0308 1.0986 1.1093 1.1864]
       OptimalK: 8
```

```
validateGap_GMM    = evalclusters(legprof_subsetFiltered, name_GMM   , 'gap', 'KList', klist)
```

```
validateGap_GMM =
  GapEvaluation with properties:

    NumObservations: 49
     InspectedK: [2 3 4 5 6 7 8 9 10]
CriterionValues: [0.6049 0.9044 0.7484 0.9670 1.1000 1.2239 0.6694 0.8675 1.2503]
       OptimalK: 6
```

```
% Davies-Bouldin
validateDaviesBouldin_HAC    = evalclusters(legprof_subsetFiltered, name_HAC   , 'DaviesBouldin
```

```
validateDaviesBouldin_HAC =
  DaviesBouldinEvaluation with properties:

    NumObservations: 49
```

```
        InspectedK: [2 3 4 5 6 7 8 9 10]
    CriterionValues: [0.8191 0.6298 0.5399 0.8489 0.8898 0.7663 0.7343 0.7620 0.7113]
           OptimalK: 4
```

```
validateDaviesBouldin_kmeans = evalclusters(legprof_subsetFiltered, name_kmeans, 'DaviesBouldir
```

```
validateDaviesBouldin_kmeans =
  DaviesBouldinEvaluation with properties:

    NumObservations: 49
        InspectedK: [2 3 4 5 6 7 8 9 10]
    CriterionValues: [0.7686 0.8705 0.6580 0.8428 0.7537 0.7201 0.6973 0.7706 0.7272]
           OptimalK: 4
```

```
validateDaviesBouldin_GMM    = evalclusters(legprof_subsetFiltered, name_GMM   , 'DaviesBouldir
```

```
Warning: Failed to converge in 100 iterations during replicate 2 for gmdistribution with 9 components
Warning: Failed to converge in 100 iterations during replicate 3 for gmdistribution with 9 components
validateDaviesBouldin_GMM =
  DaviesBouldinEvaluation with properties:

    NumObservations: 49
        InspectedK: [2 3 4 5 6 7 8 9 10]
    CriterionValues: [0.8316 0.6646 0.6377 0.9072 0.7478 1.1029 0.9321 0.6004 1.0598]
           OptimalK: 9
```

Part 2, Problem 9 - Discuss internal validation

Going by the silhouette index (the other two seemed too crazy/unhelpful), the optimal number of clusters was k=3 for HAC and k-means, and k=2 for GMM. But looking at the CriterionValues for HAC and k-means, we can see that the scores between k=2 and k=3 were actually very close to each other. No other values of k came anywhere near the values at k=2 and k=3, so we can say that we made a pretty good choice in choosing k=2 for all three algorithms at the beginning. That k=2 produces the best, most well-defined clusters is not surprising, given the ODI plot when visualising clusterability. The fact that k=3 was also suggested might have something to do with the tiny "cluster" at the very top-left of that plot.

Now, there are definitely reasons for choosing a "sub-optimal" strategy when clustering. While each algorithm has the same goal of "clustering", in reality, this means conceiving of some kind of possible structure in the data - but there are different ways of conceiving different kinds of structure, meaning that different algorithms will look for different things (e.g. DBSCAN will look for density, etc.). Sometimes hard partitioning via k-means is good because its output is simple to read, but sometimes there's a really fuzzy boundary that should be respected where the information-loss is bad - in that case, GMM would be more suitable. The choice of algorithm really boils down to what kind of information/structure one wants to extract and preserve, which can be informed by domain expertise.