Autumn 2023

TNM093
Information Visualization

Mini-Project

By Elias Elmquist and Peilin Yu. Based on a previous lab assignment by Kahin Akram.

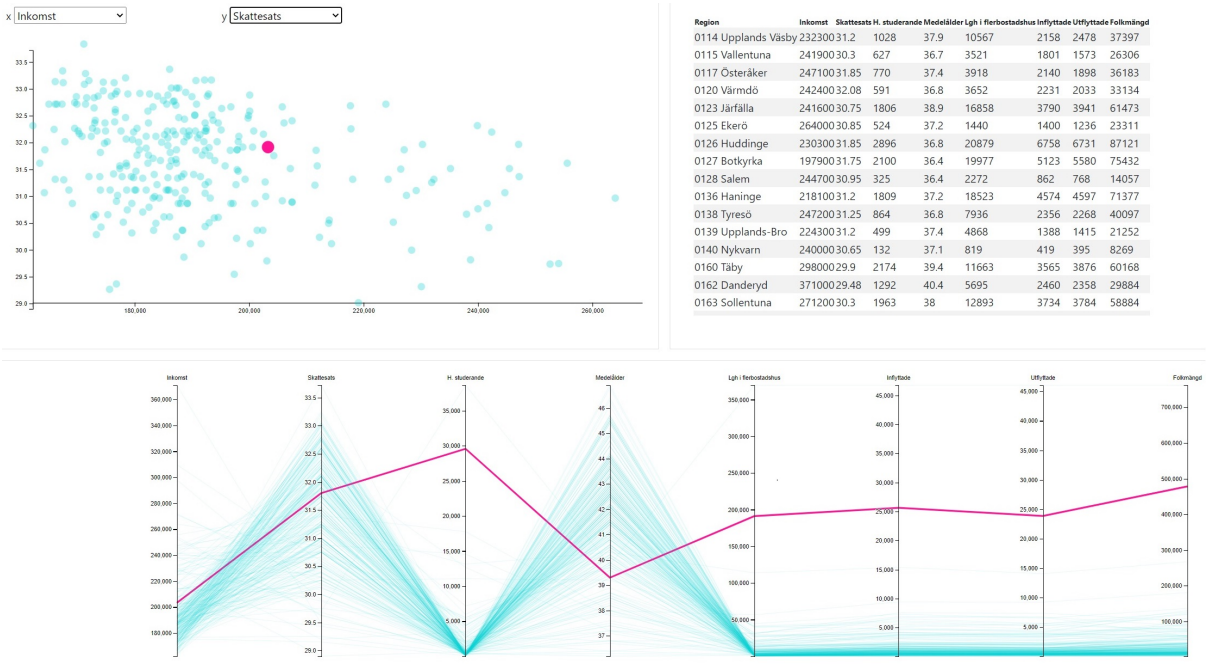

Figure 1: Final view of the assignment of Part 1.

1 Intended Learning Outcomes

The student shall, after finishing this computer exercise, be able to:

- Present a basic understanding of the importance of visual analytics;
- Have a basic understanding of the web library D3.js and be able to implement commonly used visual representations, such as parallel coordinates and scatter plots;
- Have a basic understanding of the clustering technique K-means.

Remember: You must work on this project outside the scheduled time too.

2 The Setup

The JavaScript library D3.js (Data-Driven Documents) will be used in this mini-project. It is a library for visualizing data using web standards. It combines powerful visualization and interaction techniques

with a data-driven approach to DOM manipulation, giving you the full capabilities of modern browsers and the freedom to design the right visual interface for your data.

Remember that D3 uses chain syntax, meaning that functions can be chained together as in JQuery and pure JavaScript!

All the necessary files and libraries are included in the zip file for this mini-project. Choose a preferred editor (Visual Studio Code[1] is recommended) and make sure to comment on the code you implement, this comes in handy when presenting the code to the assistant.

Please look through the following tutorials and examples to prepare for the lab. The links for the API and Wiki will be especially helpful for the tasks in this mini-project.

- Explained - `https://www.youtube.com/watch?v=IdDuWwquSwA`
- Short Tutorial - `https://d3-graph-gallery.com/intro_d3js.html`
- Longer Tutorial - `https://www.tutorialsteacher.com/d3js`
- Examples - `https://observablehq.com/@d3`
- API - `https://github.com/d3/d3/blob/main/API.md`
- Wiki - `https://github.com/d3/d3/wiki`

2.1 Hosting the Webpage

If you are on a lab computer at the Campus, you can use the W hard drive to host your HTML file. If you use your own computer, we would recommend using the Live Server plug-in[2] for Visual Studio Code to host your web page. Otherwise, you can use Firefox by enabling local files by typing "about:config" in the URL bar, search on "privacyfile_unique_origin" and set it to *false*.

3 The Data

This mini-project uses the Swedish population dataset with 290 entries and 9 features. These features are: **Region**, **Medelvärde förvärvsinkomst**, **Kommunal skattesats**, **Antal högskolestuderande**, **Medelålder**, **Antal lägenheter i flerbostadshus**, **Antal inflyttade**, **Antal utflyttade**, and **Folkmängd**.

In English: Region, Average earned income, Municipal tax rate, Number of university students, Average age, Number of flats in apartment buildings, Number of people moved in, Number of people moved out, Population.

4 Finding *Interesting* Research Questions

Task 1:
Run the lab code by hosting it through any of the methods listed in section 2.1. If done correctly you will see a table on the right side showing the data. Your task now is to investigate the data and try to understand it by just using the table. Please, from this stage and on take notes of your findings!

Task 2:
By now you might have understood that it is nearly impossible to find any interesting patterns by just investigating the table. Go back to the variable names and try to come up with **1-2 research questions** about the correlations of the variables. You will then, when all the code is implemented, try to answer these by visually analyzing the data (this is easier if you have a good understanding of the variables).

[1] Visual Studio Code `https://code.visualstudio.com`
[2] Live Server `https://marketplace.visualstudio.com/items?itemName=ritwickdey.LiveServer`

Please, be specific when coming up with the question(s). Do not use questions such as *"Which county/region has the highest average income?"* or similar simple questions that can be answered by doing a sort on the table! A more interesting question could for example be *"Is income related to the level of education?"*.

Visualization helps in identifying complex patterns that are difficult to see in raw data. Therefore, the goal of this mini-project is not to provide us with an answer to the questions you have nor is it to write good code. The goal is to learn how to visually analyze data. When doing this you might come across other interesting findings and questions. The goal is to learn to explore and think about data visualization.

5 Part I: Implementing Visual Representations

Now that you have an understanding of the dataset and your research questions are ready, it is time to implement some code.

5.1 Scatter Plot

A scatter plot (or scatter chart, scatter graph) uses dots to represent values for two different numeric variables. The position of each dot on the horizontal and vertical axis indicates values for an individual data point. Scatter plots are used to observe relationships between at least two variables but can handle more dimensions. However, the overall relationship might then not be the main focus.

The code for the scatter plot will be implemented in the *labcode* function in the *sp.js* file.

5.1.1 Create the x-axis

We will begin by implementing the two axes used to span the scatter plot. Let us begin with the x-axis. Begin by creating the x-axis for our scatter plot and name this variable x. The axis should use a linear scale, and use the minimum and maximum values of the two selected variables from the data set as domain (see the *sp* function further down in the file). Then set the range from 0 to the width (the *width* variable is useful here). **Note: use the + sign before the return in the function for making sure the returned value is not a string value. Also, the methods should be chained!**

Use following methods:
d3.scaleLinear() - Constructs a new continuous scale with the specified domain and range.
.domain() - The Domain denotes the minimum and maximum values of your input data.
d3.min() - Compute the minimum value in an array.
d3.max() - Compute the maximum value in an array.
.range() - The Range is the output range, which we would like the input values to map to.

At this point, you will not see anything new added to the web page, yet.

5.1.2 Append the axis to svg

Now, we will append (draw) the axis variable x on the *sp_svg* window as the variable *xAxis*. This is done by appending a "g" tag to the svg window and translating it to a desired position (the *height* variable is useful here). Then call the x axis and put it at the bottom.

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.attr() - Gets or sets an attribute on the selected element. If you are not familiar with svg transformation, have a look at this tutorial[a] or Google it to find more information.
.call() - Call a method.
d3.axisBottom() - Constructs a new bottom-oriented axis generator for the given scale.

[a] D3.js - SVG Transformation `https://www.tutorialspoint.com/d3js/d3js_svg_transformation.htm`

If you have done all the steps correctly, you should now see a horizontal axis for the scatter plot.

5.1.3 Create y-axis

The y-axis is implemented in the same way as the x-axis. Use the same methods but remember that the y-axis is inverted, we do not need to transform it, and we now use the height instead.

Use following methods:
d3.scaleLinear() - Constructs a new continuous scale with the specified domain and range.
.domain() - The Domain denotes the minimum and maximum values of your input data.
d3.min() - Compute the minimum value in an array.
d3.max() - Compute the maximum value in an array.
.range() - The Range is the output range, which we would like the input values to map to.

At this point, you will not see anything new added to the web page, yet.

5.1.4 Append the axis to svg

As done with the x-axis, we will append the y-axis to our *sp_svg* window as variable *yAxis* by first creating a "g" tag, and then calling the *y* variable we created above and put it on the left.

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.call() - Call a method
d3.axisLeft() - Constructs a new left-oriented axis generator for the given scale.

You should now see a vertical axis for each of the scatter plots.

5.1.5 Append circles to svg

Now that we have the two axes it is time to add the circles. Append a "g" tag to our *sp_svg* window and assign it to a new variable called *myCircles*. Then select all "circle" and use *data* to know how many to create. After that, *enter* the selection, and append the "circle".

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.selectAll() - Selects all elements that match the specified selector string.
.data() - Binds the specified array of data with the selected elements.
.enter() - Returns the enter selection: placeholder nodes for each data item that had no corresponding DOM element in the selection.

At this point, you will not see anything new added to the web page, yet. However, if you Inspect Element (Right-click on the scatter plot) to check that the circle tags are added to the scatter plot (see Figure 2). Notice that they are empty and need to be positioned and filled, which we will do next.

5.1.6 Add attributes to the circles

We now have the placeholders for the circles but as you see, we are missing the information. This is because we have not added the position, radius and color for each circle yet. We will do this now.

Use the methods below to add the "cx" and "cy" positions for each circle created from the two variables we want to show. Then add a radius "r" of *6*, and style the circle by setting "fill" with "darkturquoise" color. Finally, add an opacity of 0.3 to the circles. **Note: This code should be chained on *myCircles*! Use the correct variables for "cx" and "cy" to visualize. Remember to add the + sign before returning the value of the variables. For styling the circles, have a look at this tutorial[3] or Google *D3 scatter plot* to find more information.**

Use following methods:
.attr() - Gets or sets an attribute on the selected element.
.style() - Get or set a style property.

[3]Basic scatterplot in d3.js `https://d3-graph-gallery.com/graph/scatter_basic.html`

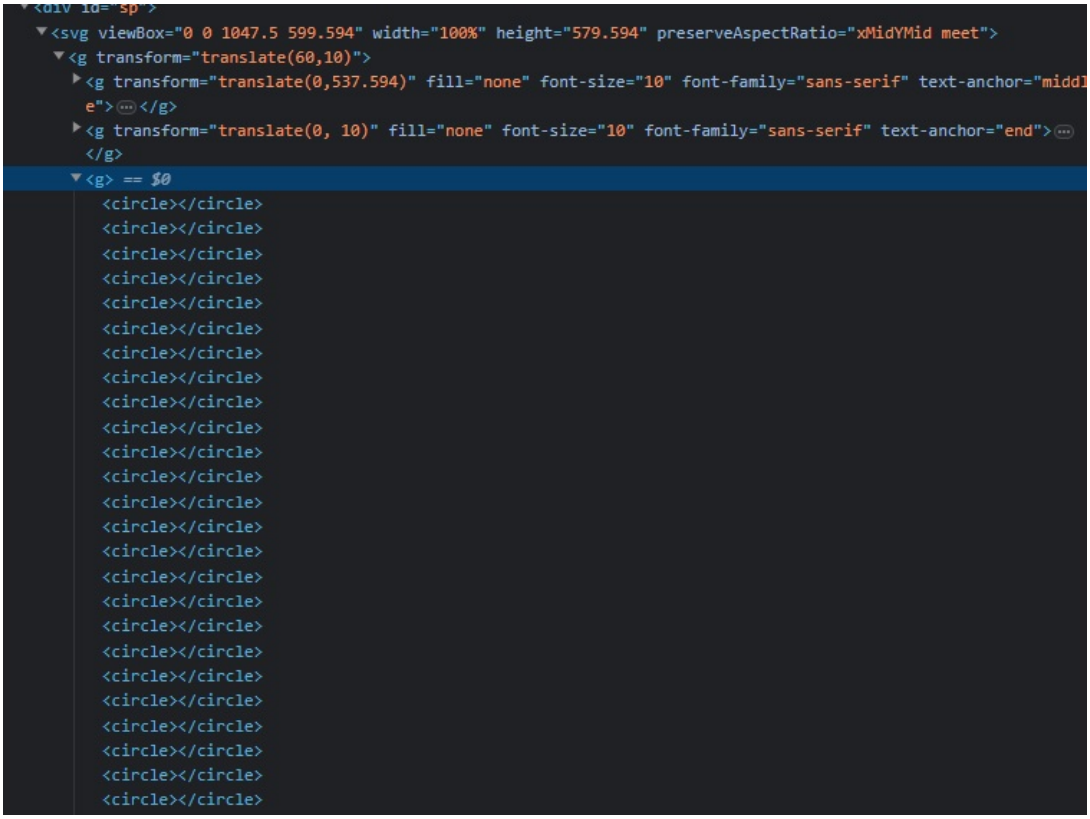

Figure 2: The Circle tags should be added at this point.

You should now see the circles on the scatter plot. You can now also change the variables for the x- and y-axis by changing the variable in the drop-down menu for each scatter plot.

5.1.7 Adding hovering

As a last touch, call our self-implemented function *hovering(myCircles, tooltip);* (this function is not to be chained). With this function, more detailed information is shown to the user when hovering over the circles. Now, we move on to the next visualization.

5.2 Parallel Coordinates

A Parallel Coordinates Plot (PCP) is a visualization technique used to analyze multivariate numerical data (see Figure 1). Each axis has its own unit of measurement and multiple axes are placed parallel to each other. The idea is to find patterns, similarities, clusters, and positive, negative, or no particular relationships in multidimensional datasets.

The code for the parallel coordinates plot will be implemented in the *pc.js* file. Initially, you will only see a bunch of gray lines.

5.2.1 Drawing the Lines

We need to draw one line (also known as a *path*) for each row in the dataset resulting in one poly-line connecting all variables (see Figure 1). For this, we are creating a variable called *foreground*. As usual, append a “g” tag to our *pc_svg* window. Chain the “class” with “foreground”, and select all “path”, before adding the *data*. After that, *enter* the selection, and append *path*. Finally, set the attribute called “d” where it takes the *drawPath* as a parameter.

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.attr() - Gets or sets an attribute on the selected element.
.selectAll() - Selects all elements that match the specified selector string.
.data() - Binds the specified array of data with the selected elements.
.enter() - Returns the enter selection: placeholder nodes for each datum that had no corresponding DOM element in the selection.

If implemented correctly you should see the lines (in cyan color) drawn at the bottom of the web page.

5.2.2 Drawing Axes

As the name Parallel Coordinates suggests, the axes for this representation are drawn parallel. For each variable in the data set, we will create one vertical axis. This can be done by using some of the previous D3 methods we have used and some new ones.

Start by selecting all ".dimension" from the *pc_svg* window, and assign it to a variable called *axes*. Then chain *dimensions* as *data*, *enter*, and append a new "g" tag. You should now see the placeholder tags for each axis when you Inspect Elements (see Figure 3). As you can see they are empty.

Our next task is to fill them with the variables and translate them. Add "dimension axis" to the "class" attribute. Adjust the "transform" attribute by using a function to translate them horizontally, i.e., *x(d)*. Finally, using *.each* for each selected element, use a function to draw the y-axes, i.e., *d3.select(this).call(yAxis.scale(y[]))*.

Use following methods:
.selectAll() - Selects all elements that match the specified selector string.
.data() - Binds the specified array of data with the selected elements.
.enter() - Returns the enter selection: placeholder nodes for each datum that had no corresponding DOM element in the selection.
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.attr() - Gets or sets an attribute on the selected element.
.each() - Invokes the specified function for each selected element.
d3.select() - Selects the root element.
.call() - Call a method
axis.scale() - If scale is specified, sets the scale and returns the axis.

You should now see the axes with the different variables on them.

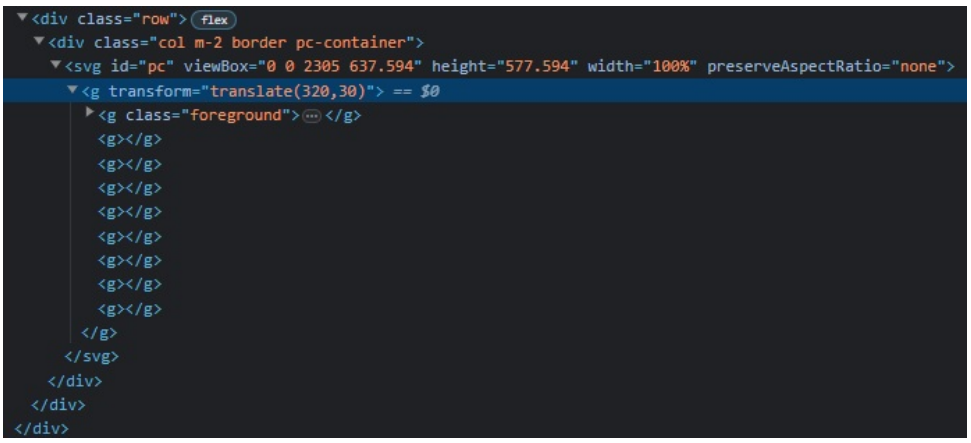

Figure 3: The g tags for the axes.

5.2.3 Appending Axes Titles

However, we are missing axes titles. Start by appending a "text" tag on the existing variable *axes*. Set the "text-anchor" attribute to "middle". Move the text upward slightly by setting the "y" attribute to, for example *-9*. Set the "fill" style to "black". Finally, add the text using a function, i.e., *function (d) return d;* .

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.attr() - Gets or sets an attribute on the selected element.
.style() - Get or set a style property.
.text() - Get or set the text content.

5.2.4 Brushing the axes

We also want to have the alternative to highlight a subset of the data for better focus. This can be done by implementing brushing (selecting a subset of the data items with an input device (mouse)) on each axis. We will now implement this functionality.

As before, append a "g" tag on the *axes* variable. Set the "class" attribute to "brush". Then, using *.each* for each axis, call the *perAxisBrush* function, e.g., *function (d) {d3.select(this).call(perAxisBrush(d));}*. Then select all "rect", set the "x" attribute to, for example, *-8*, and set the "width" attribute to, for example, *10*.

Use following methods:
.append() - Creates a new DOM element and adds it at the end of the selected DOM element.
.attr() - Gets or sets an attribute on the selected element.
.each() - Invokes the specified function for each selected element.
.call() - Call a method.
.selectAll() - Selects all elements that match the specified selector string.

5.2.5 Dragging the Axes

Finding correlations between the variables is an important task with parallel coordinates. However, having static axes restricts this as we only can compare variables on adjacent axes. Therefore, we need to implement movable axes.

Call the *d3.drag()* on the *axes* variable to set the dragging functionality on the axes. This is done by using *.subject()* and calling a function e.g., *function (d) { return { x: x(d) };}*, it takes a function and returns a new object *x*, and the value is set to *d*, passed through *x* axis. Finally, *on* the "start", "drag", and "end", call the existing *startDrag*, *drag*, *endDrag* functions.

Use following methods:
.call() - Call a method.
d3.drag() - Create a drag behavior.
.subject() - Set the thing being dragged.
.on() - Listen for drag events on the current gesture.

If everything has been implemented correctly, you should be able to move the axes now. Click the axis title and move it to the left or right.

5.3 Answer your Research Questions

One thing left to do is to call our self-implemented functions selectValues() and resetSelectedValues() on the table rows (cells variable in info.js). Use .on() and 'mouseover, mouseout' to implement this.

Now, analyze the data and contemplate over your research questions by using the visualizations that you now have implemented. Also, think of what else you could find that you had not thought of before visually analyzing the data. Prepare some notes (which can later be part of your reflection document) that you can use when presenting the answers to your research questions to a lab assistant.

After that, we will look into how clustering techniques can help to analyze your data. We are going to learn about the K-means clustering specifically and will see it in action in the next part.

6 Part II: K-means

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar to each other than to those in other groups (clusters). It is a main task of exploratory data mining, and a common technique for statistical data analysis, used in many fields, including machine learning, pattern recognition, image analysis, information retrieval, and bio-informatics. For this last assignment, we will be using the clustering technique called K-means.

6.1 The Algorithm

K-means is one of the simplest unsupervised learning (unlabeled) algorithms that solves the well-known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters fixed a priori. The main steps of the algorithm are:

1. Randomly place K points into the space represented by the items that are being clustered. These points represent the initial cluster of centroids.

2. Assign each item to the cluster that has the closest centroid. One way to do this is using the Euclidean distance:

$$\sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{1}$$

3. When all objects have been assigned, recalculate the positions of the K centroids to be in the center of the cluster. This is achieved by calculating the average values in all dimensions.

4. Check the quality of the cluster. Use the sum of the squared distances within each cluster as your measure of quality. The objective is to minimize the sum of squared errors within each cluster:

$$min \sum_{i=1}^{k} \sum_{x_j \in S_i} ||x_j - \mu_i||^2 \tag{2}$$

where μ_i is the centroid (mean of the points) of each cluster Si_i. Iterate (repeat steps 2-4) as long as the quality is improving, once there is no or minimal (less than a threshold) improvement, break the algorithm.

Understanding the algorithm:
Write down in pseudo-code how you would implement each step listed above to create the K-means algorithm. See this link for a visual explanation of K-means[a]. Use for-loops and the equations listed above for your pseudo-code.

[a]Visual explanation of K-means `https://en.wikipedia.org/wiki/K-means_clustering#Initialization_methods`

6.2 Analyze the clustering results

Unfortunately, there is no general theoretical solution to find the optimal number of clusters for any given data set. A simple approach is to compare the results of multiple runs with different k classes. In addition, the random initialization of the centroids can influence the result.

Analyze the results:
Display the index2.html file included in the zip file on a web page. It should look like it does in Figure 4.

On this web page, you will see two parallel coordinate plots showing a synthetic dataset, one without the K-means algorithm (top of the screen) and one with the K-means algorithm (bottom of the screen). Furthest down you will find a slider that determines the number of clusters for the K-means algorithm. Change the number of clusters with this slider and study the results.

What do you think is a good number of clusters for this specific data set? What are the shapes of the clusters? Take notes to later discuss your results with the lab assistant.

You are now ready to present your mini-project to the lab assistant.

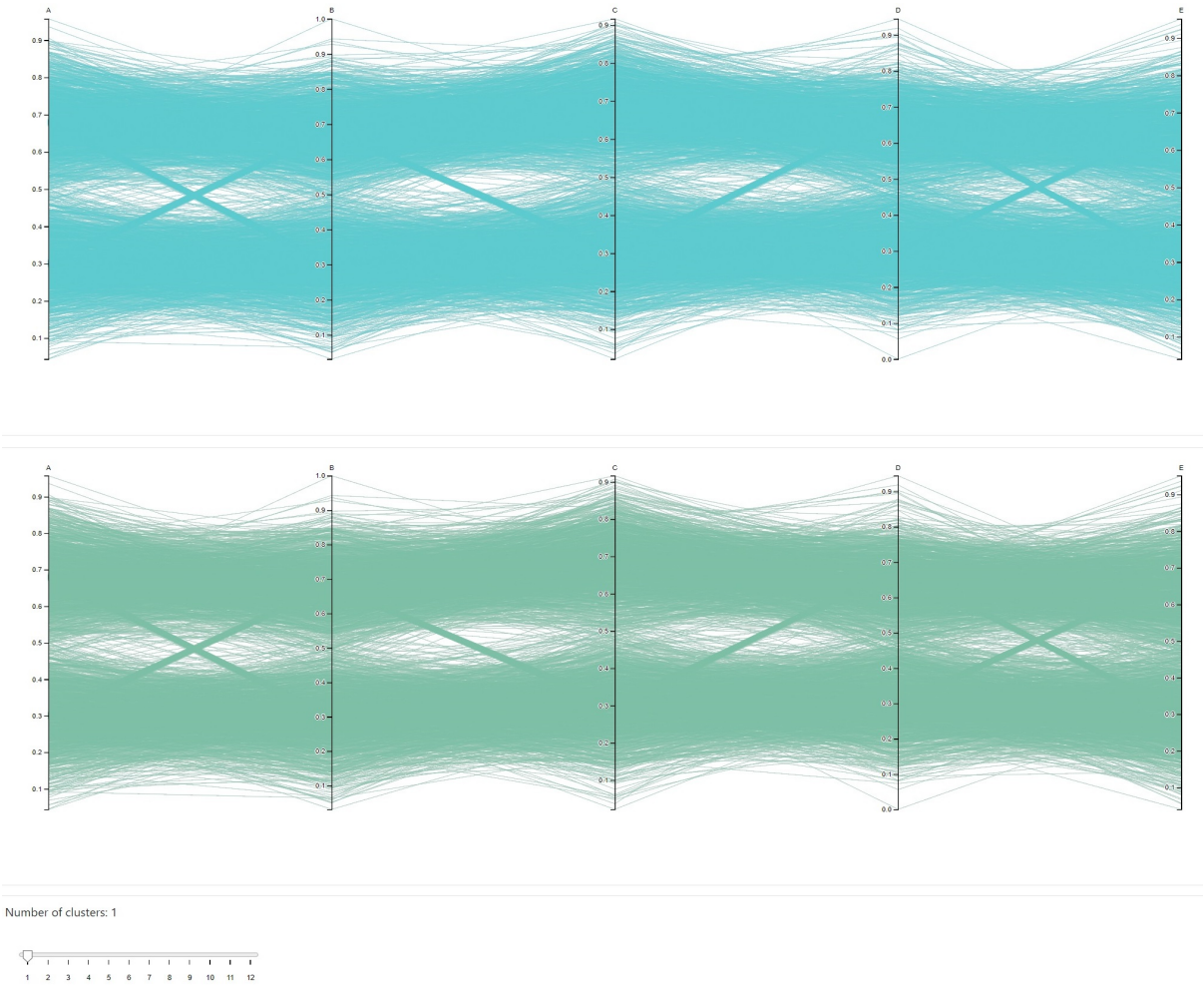

Figure 4: Initial view of the assignment of Part 2.

7 Reflection Document

After presenting the mini-project to a lab assistant, use your notes from the lab to write your one-page reflection document. The reflection document should contain your research questions for the dataset, the answers to them, and how you found out your answers. It should also contain a reflection on the K-means result, including the number of clusters and what shapes were seen in the visualization.