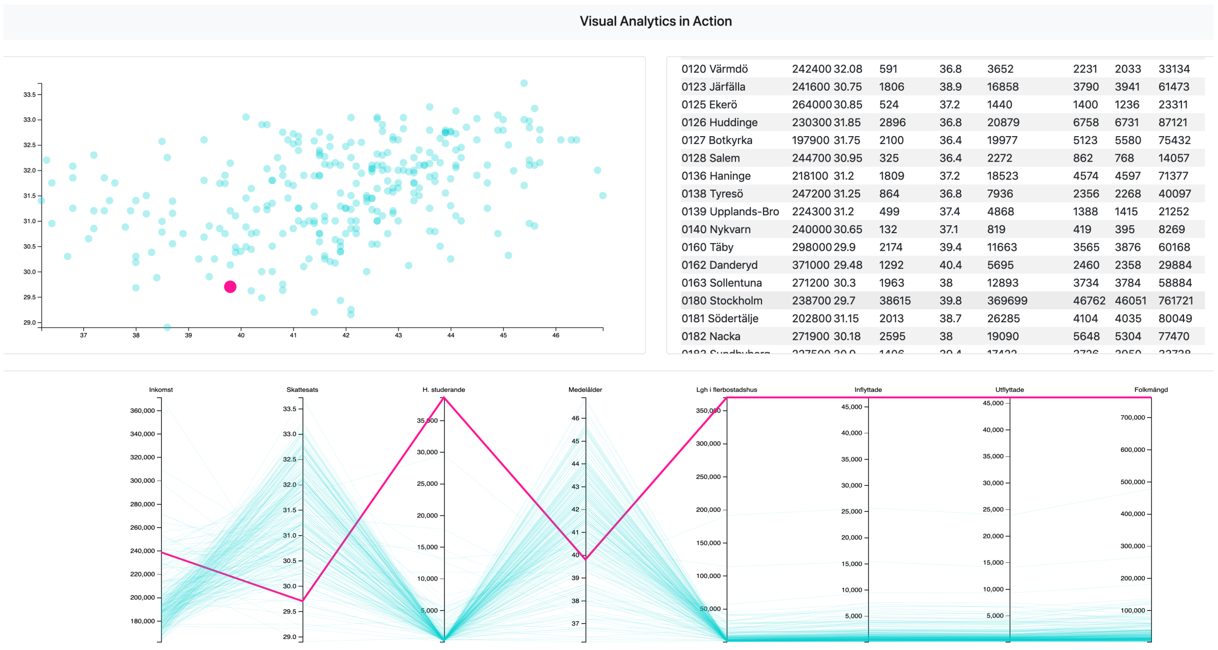
TNM093—INFORMATION VISUALIZATION

COURSE RESPONSIBLE: ALEXANDER BOCK

LAB CREATED BY: Kahin Akram Hassan

LAB ASSISTANTS: Elias Elmquist, Carlo Navarra

elias.elmquist@liu.se
carlo.navarra@liu.se



0120 Värmdö	242400	32.08	591	36.8	3652	2231	2033	33134
0123 Järfälla	241600	30.75	1806	38.9	16858	3790	3941	61473
0125 Ekerö	264000	30.85	524	37.2	1440	1400	1236	23311
0126 Huddinge	230300	31.85	2896	36.8	20879	6758	6731	87121
0127 Botkyrka	197900	31.75	2100	36.4	19977	5123	5580	75432
0128 Salem	244700	30.95	325	36.4	2272	862	768	14057
0136 Haninge	218100	31.2	1809	37.2	18523	4574	4597	71377
0138 Tyresö	247200	31.25	864	36.8	7936	2356	2268	40097
0139 Upplands-Bro	224300	31.2	499	37.4	4868	1388	1415	21252
0140 Nykvarn	240000	30.65	132	37.1	819	419	395	8269
0160 Täby	298000	29.9	2174	39.4	11663	3565	3876	60168
0162 Danderyd	371000	29.48	1292	40.4	5695	2460	2358	29884
0163 Sollentuna	271200	30.3	1963	38	12893	3734	3784	58884
0180 Stockholm	238700	29.7	38615	39.8	369699	46762	46051	761721
0181 Södertälje	202800	31.15	2013	38.7	26285	4104	4035	80049
0182 Nacka	271900	30.18	2595	38	19090	5648	5304	77470

Figure 1: Final view of the assignment

You must work on this project outside scheduled time too

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 v4 and be able to implement commonly used visual representations such as parallel coordinates and scatter plot.

2 The Data

This computer exercise 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, Folkmängd**
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

3 The Environment

The JavaScript library D3.js (Data-Driven Documents or D3.js) version 4 will be used in this assignment. 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.

Please look through the following tutorials and examples to prepare for the lab. Then use the links and the API documentation as help.

- Explained – https://youtu.be/IdDuWwquSwA
- Short Tutorial – https://www.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/master/API.md
- Wiki – https://github.com/d3/d3/wiki

4 Finding *Interesting* Research Questions

Task 1:
Run the code on the **W** drive and open up the web page via ***www.student.itn.liu.se/∼student_id*** 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.** You will then, when all the code is implemented, try to answer these by visually analyzing the data (this becomes easier if you have a good understanding of the variables).

Please, be specific when coming up with the question. Do not use questions such as *"Which county/region has the highest average income?"* or similar simple questions! A more interesting question could be *"Is income related to the level of education?"* Or *"What could be (some of) the attractive factors for immigration?"* You must think of other questions than the examples above. Although, you are, of course, welcome to use them too, but you must have your own questions as well.

Visualization helps in identifying complex patterns that are difficult to see in raw data. Therefore, the goal of this computer exercise 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 assignment is to learn to explore and think around data visualization.

5 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.

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.

5.0.1 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 have a linear scale and use the minimum and maximum values of the two selected variables from the data set as domain (see top of file). Then set the range from 0 to the width. **Note: use + 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 minimum and maximum values of your input data.
.range() – The Range is the output range, which we would like the input values to map to.
d3.min() – Compute the minimum value in an array.
d3.max() – Compute the maximum value in an array.

5.0.2 Append the axis to svg

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

Use following methods:
.append() – Creates a new DOM element and adds it at the end of selected DOM element.
.attr() – Gets or sets an attribute on the selected element.
.call() – Call a method.
d3.axisBottom() – Constructs a new bottom-oriented axis generator for the given scale.

If you have done all the steps correctly, you should see the axis drawn on the top-left square now.

5.0.3 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 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 minimum and maximum values of your input data.
.range() – The Range is the output range, which we would like the input values to map to.
d3.min() – Compute the minimum value in an array.
d3.max() – Compute the maximum value in an array.

5.0.4 Append the axis to svg

As the x-axis we will append the y-axis to our svg window 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 selected DOM element.
d3.axisLeft() – Constructs a new left-oriented axis generator for the given scale.
.call() – Call a method

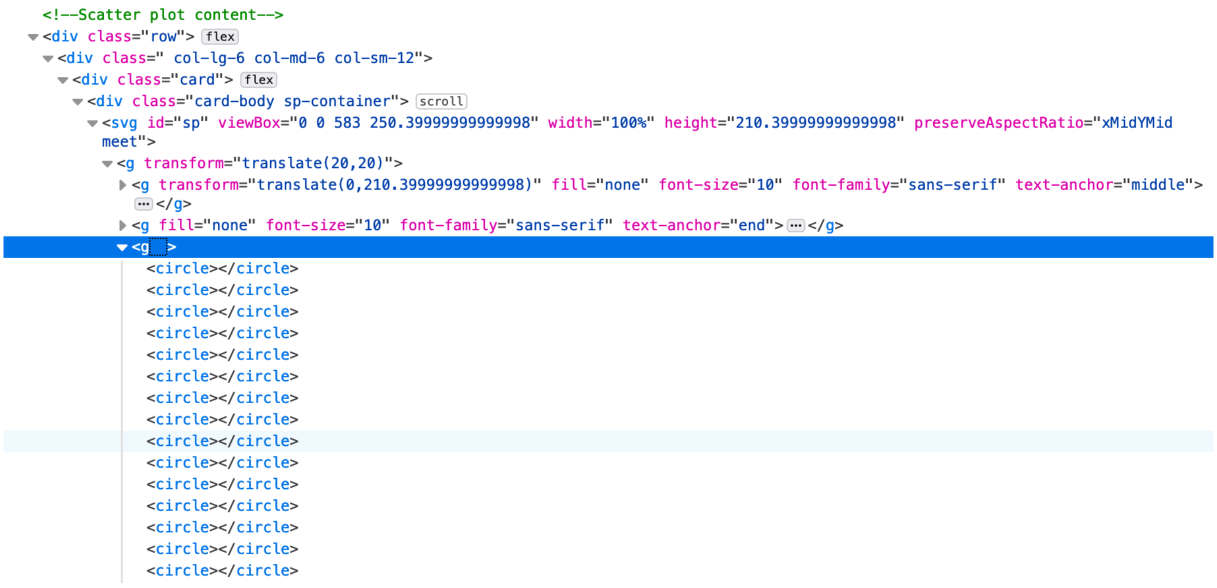
You should now see the two axes drawn on the svg window.

5.0.5 Append circles to svg

Now that we have the two axes it is time to add the circles. Append a ’g’ tag to our 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 entering append ’circle’ to this as well.

Use following methods:
.append() – Creates a new DOM element and adds it at the end of 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.

If you use *Inspect Element* (Right-click the mouse somewhere on the page) 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'll do next.



```
  <!--Scatter plot content-->
▼ <div class="row"> flex
  ▼ <div class=" col-lg-6 col-md-6 col-sm-12">
    ▼ <div class="card"> flex
      ▼ <div class="card-body sp-container"> scroll
        ▼ <svg id="sp" viewBox="0 0 583 250.39999999999998" width="100%" height="210.39999999999998" preserveAspectRatio="xMidYMid
          meet">
          ▼ <g transform="translate(20,20)">
            ▶ <g transform="translate(0,210.39999999999998)" fill="none" font-size="10" font-family="sans-serif" text-anchor="middle">
              … </g>
            ▶ <g fill="none" font-size="10" font-family="sans-serif" text-anchor="end"> … </g>
            ▼ <g >
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
                <circle></circle>
```

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

5.0.6 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' position for each circle created from the two variables we want to show. Then add a radius of 6 and fill the circles with 'darkturquoise' color. Finally add a opacity of 0.3 to the circles. **Note:** This code should be chained on *myCircles*. Use the variables we want to visualize and remember to add a '+' sign before returning the value of the variables. Please Google **D3.js Iris data** to see how to add these circles correctly.

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

You should now see the circles on the scatter plot.

5.0.7 Adding hovering

As a last touch, call our self-implemented function *hovering()* (this function is not to be chained). With this function more detailed information is shown to the user when hovering the circles.

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.

5.0.8 Drawing the Lines

We need to draw one line also known as path for each row in the data set resulting in one poly-line connecting all variables (see Figure 1). As usual append a 'g' tag to our *pc_svg* window and assign it to a new variable called *foreground*. Chain the class called *foreground* and select all *path* before adding the data. Finally, append *path* and an attribute called *d* where it takes the **drawPath** as parameter.

Use following methods:
.append() – Creates a new DOM element and adds it at the end of 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 drawn in the square at the bottom.

5.0.9 Drawing Axes

As the name suggest 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. 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 (see Figure 3). As you can see they are empty. Our next task is to fill them with the variables and translate them. Add two classes (*dimension, axis*) as attributes to these tags and use transform to translate them horizontally on *x*. Then for each use *d3.select(this)* to call *yAxis* and scale it on *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 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 units on them.

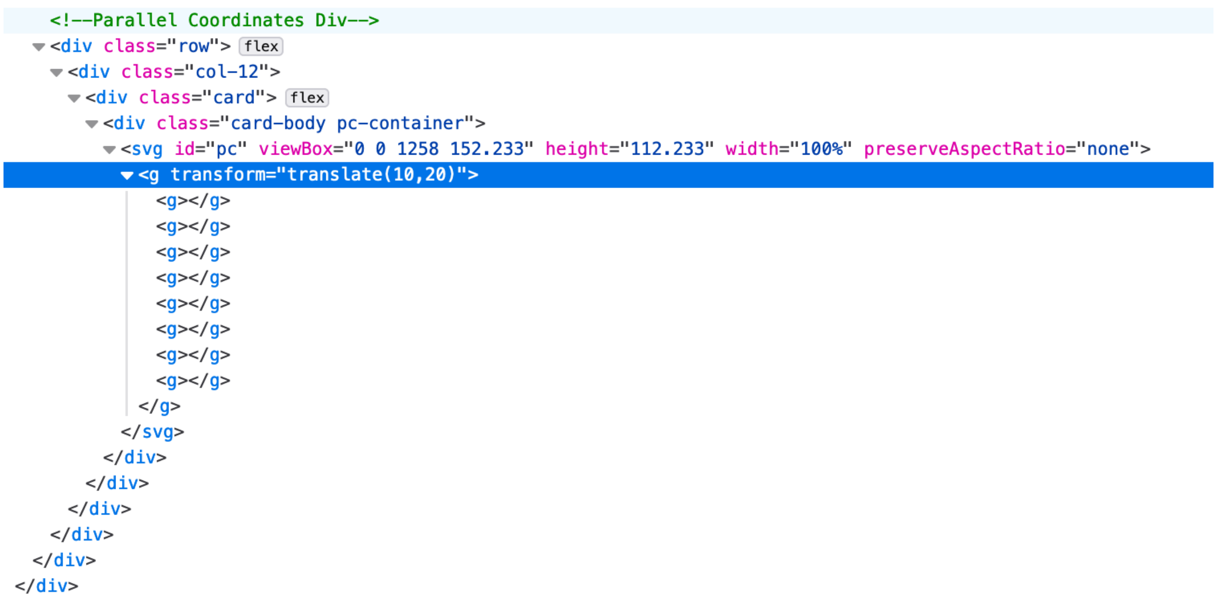
5.0.10 Appending Axes Titles

Though, we are missing axes titles. Use the code below to add titles to the axes. Remember to add the class *title* to each tag.

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

5.0.11 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 axes. We will now implement this functionality. As before, append a 'g' tag on the axes variable and a class *brush*. Then use *.each()* with a function that takes *d3.select(this)* and calls *perAxisBrush(param)* as parameter. Then select all 'rect' and add two attributes an arbitrary 'x' and 'width'.



```
<!--Parallel Coordinates Div-->
▼<div class="row"> flex
  ▼<div class="col-12">
    ▼<div class="card"> flex
      ▼<div class="card-body pc-container">
        ▼<svg id="pc" viewBox="0 0 1258 152.233" height="112.233" width="100%" preserveAspectRatio="none">
          ▼<g transform="translate(10,20)">
              <g></g>
              <g></g>
              <g></g>
              <g></g>
              <g></g>
              <g></g>
              <g></g>
              <g></g>
            </g>
          </svg>
        </div>
      </div>
    </div>
  </div>
 </div>
</div>
```

Figure 3: The g tags for the axes.

Use following methods:
.append() – Creates a new DOM element and adds it at the end of 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.

5.0.12 Dragging the Axes

Finding correlation 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 *d3.drag()* on the *axes* variable and set the dragging functionality on the axes. This is done by using *.subject()*. It takes a function and returns a new object *x:*. The value is set to *d*, passed through *x* axis. After this, **start, drag, and end** the dragging by calling *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.

6 Final Thoughts

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. Also, think of what else your could find that you had not taught of before visually analyzing the data? **Finally, to be prepared for the examination you are required to write a one page summary of what you have found and why you think visualization is important. Use this document to discuss the findings with the lab assistant.**