Spring 2026

Course Assignments
for

Information Visualization (TNM111)
Spring 2026

Assignment 3

The deadline for this assignment is Feb 16th, 2026 (at 23:59).

Part 1 *Interaction*

- What is the difference between Overview+Details and Focus+Context? Try to describe the pros and cons of each approach.
- What is the difference between the Graphic Fish-eye and Logic Fish-eye views? Can they be combined? If yes, how (give an example)?
- What are Polyfocal Displays? What are they used for? Draw the transformation function for Polyfocal Displays.
- Describe the idea behind Magic Lenses. Can you provide an example of a magic lens not discussed in the lectures together with a short description?

Part 2 *Visual Information-Seeking Mantra (Originally authored by Kahin Akram)*

Introduction

The design process is an important starting point when implementing advanced visual graphics. In this assignment, you will go through a concept called *Visual Information–Seeking Mantra* (*Please check this link*). The idea behind this mantra is to first introduce an overview to the user, then provide zooming and filtering, and lastly details on demand.

The Setup

HTML5, D3 JavaScript library (version 4), and Leaflet.js will be used. All the necessary files and libraries are included in the zip file for the assignment. Choose a preferred editor and make sure to comment the code well, this comes in handy when presenting the code to the assistant.
If you are on a lab computer at Campus, you can use the W hard drive to host your HTML file. Otherwise, you can use Xampp or Python Flask to run a server, or use the live server plug-in for Visual Code.

The Data

Earthquake data will be used in this assignment. The data is packaged as GeoJson (*You can find information on GeoJson here!*) format in order easier plot it on a map. Please, for a better understanding of the data, investigate this file a bit, you can find it under *data/ethqk.geojson*.

1 Visual Information - Seeking Mantra

We will mainly be working with three different files: *fpc.js*, *map.js*, and *plot_points.js*. These can be found in the *js*-folder.

1.1 Step 1 - Overview

The visual representation should first give an overview of the entire data collection for the user to quickly identify points of interest. This view should contain a movable field-of-view box allowing the user to control the content of the detail view. In this assignment, zooming is used. We will use a concept called *focus + context*. *Check this example!*

However, we will implement the context view first in order to follow the mantra. This view will also have brushing implemented which will be connected to the *focus* area as well as the world map.

Task 1:
We will start small, and begin with the parsing of the date. Use *timeParse* in *fpc.js* and assign it to a variable called *parseDate* that we will use later on. The format should be ***Y-m-d***.

Task 2:
In this task you have to define d3 scales and axes for the scatter plot (the focus area), four variables are needed. Call them *xScale*, *yScale*, *xAxis*, *yAxis*. xScale should use scaleTime and have a range between zero and *width*, use scaleLinear for the yScale. Then use axisBottom on xScale and call it xAxis and use axisLeft on yScale and call it yAxis. **Hint: Use width and height.**

Task 3:
Now that you have the axes for the scatter plot, the context area needs some too. Three variables, (*navXScale*, *navYScale*, *navXAxis*) are needed here. Again, use scaleTime, scaleLinear, and axisBottom. **Hint: Use width and height2.**

You will not see anything yet, but if you have implemented everything correctly you will soon.

Task 4:
The brush functionality will be implemented here. Create a variable called brush and assign it to *d3.brushX()*. Now add **.extent** with *width* and *height2* and use **.on** for calling the *brushed()* function.

Just a reminder: Google it!!

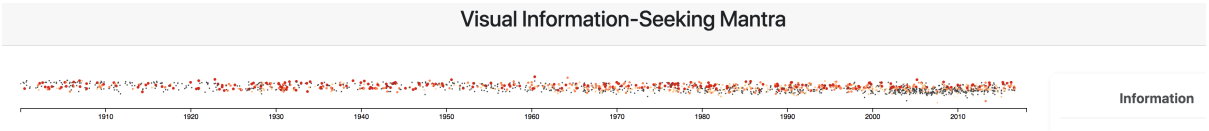
Task 5:
Use **.domain** to set the axes scales for both graphs. Remember to use the scaling variables defined in the file after task 5: (xScale, yScale, navXScale, navYScale).

Task 6:
Use **.call** to call the navigation axis variable (navXAxis) from task 3 on the *context* variable.
You should now see the x-axis on the context graph with a range between 1910-2010.

Now we need to plot the points on the context graph to get the overview.

Task 7:
Use **.data()**, **.enter()**, **.append()**, and **.attr()** on *small_points* to draw the circles.
Use **data.features** for **.data()**, and add *enter()*, append **"circle"**
as well as appending a class **"dotContext"** with **.attr()**.

Task 8:
Call the plot function with three parameters, and uncomment the line at the end of *fpc.js* that assigns to variable *curr_points_view*. **You should now see the small graph, see the figure below.**



1.2 Step 2 - Zooming and Filtering

In this step, the user wants to zoom in on items of interest. (S)he usually has a task in mind, therefore the option of zooming, filtering, ordering, etc. of the data must be there. Zooming will help users preserve their sense of position and context, zooming could be on one dimension at a time which is the case in this assignment but it could also be on multiple dimensions. I will leave that for you to solve in the project, if interested. With filtering, we want to filter out the uninteresting items in the data. Here dynamic queries ought to be applied for faster response to make the visual analysis smoother. Sliders, buttons, or other control widgets could be used for filtering the data and coupling it to rapid display updates which is the goal even if the data is large. Zooming in this exercise will happen in the *focus view* (large graph) while the filtering will be on the *context view* (small graph).

Task 9:
The *brushed()* function in *fpc.js* calls the dots in the scatter plot (large graph), therefore, we have to put it after the scatter plot. Implement the code for the brush just above the *brushed()* function at the bottom of the file. Use **.append()**, **.attr()**, and **.call()** to append a **d3** *g* tag and a class named *brush* to the *context* variable. Then call *brush* and use *brush.move* on *xScale.range()* in another call.

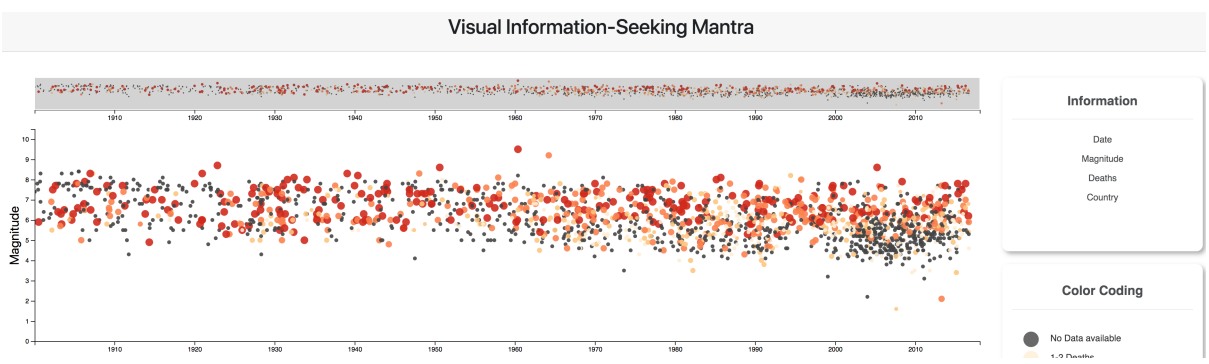
You should now get a brush over the small graph that you can use on the x-axis.

Task 10:
Under task 10 there are two *g* tag append on the *focus* variable, these are for the axes. Your task is to add some attributes. On the first add two attributes (**.attr()**), one for the class *axis axis--x* and one for transforming to *0, " + height + "*). Then call the xAxis. On the other tag, add one **.attr()** for the class *axis axis--y* and call the yAxis.

Task 11:
Use *data.features* together with **.enter()** and append **"circle"** as well as appending a class **"dot"** to *selected_dots*. Finally set the opacity to a desired value.

Task 12:
Call the plot function.

If you have implemented all the steps correctly you should end up with the results shown in the image below.



1.3 Step 3 - Details on Demand

This step will cover details on demand, when the user selects an item or a group of items details of those items should be provided for deeper analysis. This step does not have to be very difficult, it could be as simple as a pop-up or update of information somewhere. In the following steps we will be working with the world map plus some hovering.

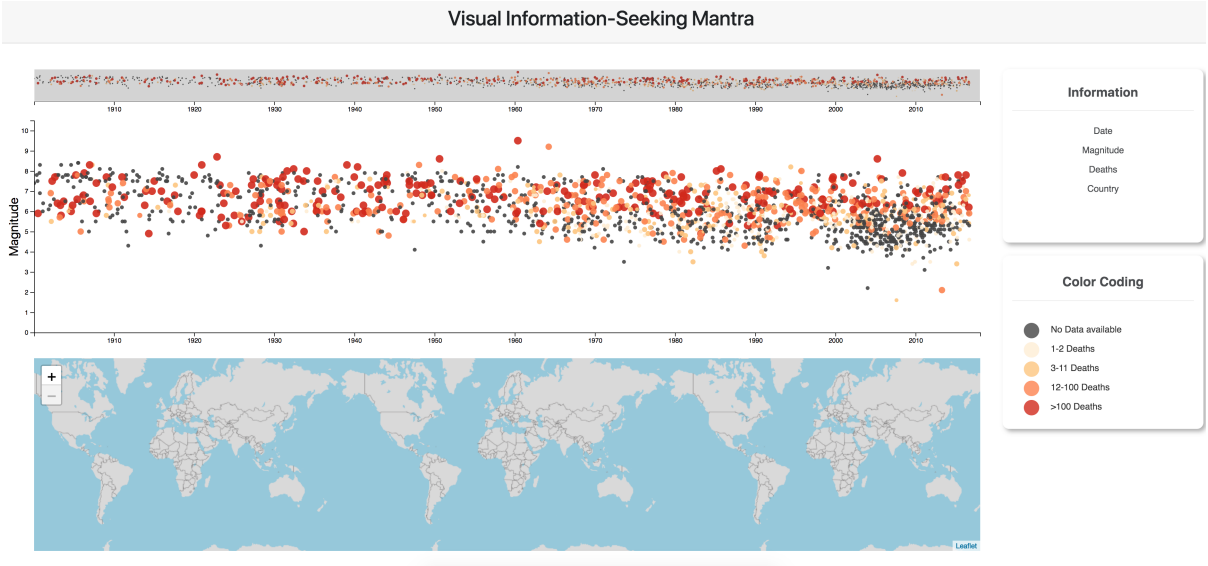
Task 13:
Inside the mouseOver() function in *fpc.js* add **.tooltip()** on the *points* variable with **d** as the parameter.

It's time to implement and work with the map. For this assignment, we will use *Leaflet.js*. Check the link here!

Task 14:
In *map.js* create a leaflet map and put the center to 10, 15 with a zoom scale of 1.

Task 15:
Assign the variable *mapLink* to **.tileLayer** on **L** with **map_link()** as the parameter. Then use **.addTo()** and add *leaflet_map* to *mapLink*.

You should get the results shown in the image below.



Task 16:
Create a variable called *svg_map* and use **d3.select()**, **.getPanes().overlayPane**, and **.append()** to create the svg map. Assign *svg_map* to **d3.select()** with **.getPanes().overlayPane** from *leaflet_map* as the parameter and then append "svg" to the **d3.select()**. Now create another variable called **g** and assign it to *svg_map*, but append also another **"g"** tag with the class *leaflet-zoom-hide*.

Task 17:
Create a function called **projectPointsOnMap** that takes **x, y** points and projects it on the map. Inside the function create a variable called **point** and use **latLngToLayerPoint** on *leaflet_map* to create a new **L.LatLng()**. Now use **this.stream** on the point with the **point.x**, **point.y**.

Task 18:
Now we need to transform all to the specific projection. Create a variable called *transform* and use **d3.geoTransform** with the function above as parameter (point:function). Create another variable named *d3path* to project this transformation to it.
Use: d3.geoTransform(), d3.geoPath() and .projection(). Also don't forget to remove the comment tags in the applyLatLngToLayer() function.

Task 19:
Plot the dots on the map. Create a variable and name it **feature**. Select all circle from **g** tag and use **data.features**. Also, add a class called **mapcircle** and set opacity to a desired value.

Task 20:
Now call the plot function with the feature variable. **Also remove the comment tags on *leaflet_map.on("moveend", reset), reset()* and the *mouseOver/mouseOut* functions.**

Please prepare a report (PDF) with your results for Part 1, and upload it as a submission to Lisam by the given deadline! If you have questions you can contact your lab assistant, Zeyang Huang (zeyang.huang@liu.se) for class A, or Jinyi Wang (jinyi.wang@liu.se) for class B.
You will present your work during the regular lab session in the lab rooms. Please check Lisam for the exact date, time, and room assignment for your group.
Note: Any kind of plagiarism is not acceptable!