Visualization Project Report 2: Are we running out of cash? - Analysis of ATM locations provided by Open Street Map

written by Moritz Blum (au640047)
for the course: E19 - Data Visualization [220192U001]

October 21, 2019

1 Changes w.r.t. the last hand-in

The first change is a replacement of the continuous heatmap by a discrete one which can be normalized by the population. Since the problem of heatmaps is that they are sometimes not expressive to show the desired data properties, but instead show a undesired one, i.e. in this application the heatmap is probably extremely biased to the distribution of population. Germany can be completely separated into multiple Landkreise and some Kreisfreie Städte, which is documented by the government, including the number of inhabitants and many other information. This data is publicly accessible in a GEOJSON format, but since this dataset contains every detail, the file is way too large to access and load in a web browser. There are ways to reduce the accuracy of GEOJSON files and therefore I found a simplified version provided by Carto[1].

The second change must be done due to the change to a discrete heatmap. Now clicking on the map and getting a detailed view on the distances and properties of ATMs in a certain region around the clicked location would not be meaningful, because in one region, the user can not visually differentiate different locations, since everything is in the same color. Therefore the user gets a detailed view about the contained ATMs without involving the distances to the clicked position.

[1] `https://lznet.carto.com/tables/landkreise_deutschland_vereinfacht/public/map`

2 Work done so far

Up to now, mainly preprocessing and preparations were made, including the setup of a web server and some initial code. Especially the data wrangling task was very work intensive, because the dataset was entered by human and therefore contained much inconsistency in naming.

2.1 Preprocessing

First of all, in Data Profiling, I figured out how to exactly query the OSM[2] data through Overpass Turbo[3]. The data were queried in steps, because otherwise the server would have thrown a timeout. Therefore a quadratic window was created in which the data of interest were queried. This window was then slid over the whole world. The query asked for nodes in OSM graph contain the tag "ATM". The whole process was done on the 12th September 2019 and took around 3 hours for the whole world. Before loading the data into OpenRefine, the JSON returned by Overpass Turbo had to be joined and reformatted, cause the structure was not read in correctly. One formatting issue was e.g. an array containing the Latitude(lon) and Latitude(lat) which needed to get split up into single entries. Another problem was that some rows were too difficult to use, e.g. different formats about the opening time, these values were too inconsistent to parse, very rare and furthermore there is no simple representation for this values.

The Data Wrangling task took much time, because many regular expressions and column joins were required. I recognized that the data is too large and too inconsistent to clean on my own, e.g. there were so many different bank names in different spellings and acronym, that some background knowledge about the financial sector in each country would be required. So I decided to only consider Germany, approximately in the window (lat:58, lon:3) to (lat:46, lon: 16), I have knowledge about. Now the dataset was around 3 MB large, containing 21644 ATMs and 67 different tags at the nodes. Then I started filtering for interesting tags. This was done manually, because there was inconsistency in the tag naming, too. Some example tags are: "wheelchair", "brand", "opening_hours", "operator", "website", "layer", "surveillance".

Then I started to establish consistent bank names for the bank for which I know that they belong together, e.g. ING or ING - Diba were named to ING.

[2] `https://www.openstreetmap.de/`
[3] `https://overpass-turbo.eu/`

There was also inconsistency across regions, e.g. the Sparkasse in Bielefeld is called Sparkasse Bielefeld and the Sparkasse in Halle in called Kreissparkasse Halle. I tried clustering, but this was not very helpful, since the clusters were not complete and each bank had way too many clusters. The most suitable solution was to search with regular expression for parts of the names and change them to a consistent naming. This needed to be done in multiple columns, since the tags were not consistent too. Then I merged everything together. The banks I differentiate are: Sparkasse, Volksbank, ING, Commerzbank, Deutsche Bank, Sparda-Bank, UBS, Postbank, DKB, Euronet, HypoVereinsbank and "other". For the other columns this was done in a similar way and the result was a table of the form: [ID] INTEGER, [OPERATOR] TEXT, [LAT] REAL, [LON] REAL, [WHEELCHAIR] TEXT, [POSITION_LEVEL] INTEGER, [SURVEILLANCE] TEXT, [BRAND_WIKIDATA] TEXT, [EMAIL] TEXT, [ATM_COUNT] INTEGER, [WEBSITE] TEXT

Data Transformation was not required, because I plan to use only a few columns in my visualization.

2.2 Setup

A SQLite database for fast data access was set up and filled with the preprocessed data exported from OpenRefine. Furthermore, I created an initial Python Flask webserver with a simple interface returning a JSON response to a query containing a SQL command, just for testing the data access from JavaScript. Then a fist MapBox[4] prototype was implemented with the response from the web-server drawn as marks on it to test performance of data loading and MapBox. To test the drawing of the regions, I loaded and displayed the GEOJSON file and colored the different regions arbitrary. I tested D3.js[5] as an alternative for drawing a map of Germany, which worked really straight forward, too. Figure 1 shows my test visualizations.

3 Visualization Options

The different visualization options are split up into two different types for visualizing the geographic locations and three based on-top of the one I consider as

[4] `https://docs.mapbox.com/help/how-mapbox-works/web-apps/`
[5] `https://d3js.org/`

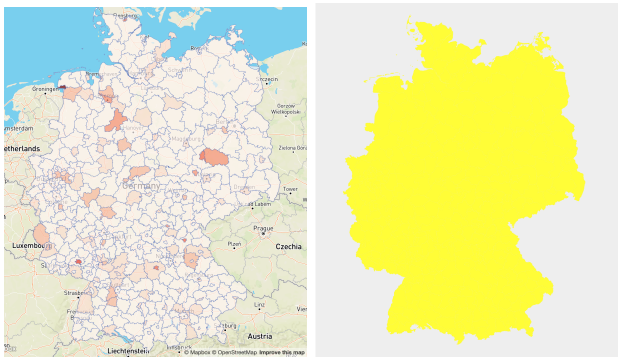

Figure 1: Heatmap visualization options. On the left MapBox, on the right D3js.

the best solution. The argumentation about benefits and drawbacks is mainly based on Mackinlays ranking of perceptual tasks[1] and the desired use case.

For presenting the locations:

1. Simple heatmap showing the continuous distribution of ATMs where the user click on a location on the map to get a detailed view about the region around. The great thing about this solution is the continuous space in which the user could inspect regions in unlimited detail. But its large drawback is the expressiveness because of an included bias, see Section 1.

2. The better solution would be a discretized heatmap, where each region is normalized by population. The drawback is that the user is restricted to the regions when exploring the data in more detail.

I decided to go with the discrete heatmap and for inspecting the regions, here are three different visualizations:

1. The inspector shows one pie chart for each operator where the number of ATMs is mapped to the size of the circle and the wheelchair properties are represented by the separation inside. The benefit of it is the great separation of different banks and the wheelchair situation can be seen directly, but the area of a circle is very bad for representing the quantitative dimension count.

2. An alternative is a mosaic plot where the operator and the count of its AMTs is mapped on the x-axis and on the y-axis the property wheelchair

accessibility is shown, which has only three different values(yes, no, unknown). This results in a compact graphic without to many details, but the plot still uses areas to represent quantitative values. In addition to this, mosaic plots are a little bit more difficult to implement than other more frequently used plots.

3. Another option is a stacked bar chart. The operators will be mapped onto the x-axis and on the y-axis the count with colors about how many of each wheelchair accessibility states is given. The main drawback is that this plot would not look very spectacular and would not attract as much attention as more advanced plots. But the mapping of values to perceptual features is very great. Furthermore the visualization is simple to implement and well known even by non experts.

4 Visualization Design

I chose to use the discrete heatmap combined with a stacked bar chart, due to the described benefits and drawbacks.

By using a heatmap, I can not make use of the best channels for quantitative values, position and length, which is clearly a drawback, but otherwise I would lose the geographic feature. So I will use color saturation and color hue to express the normalized count of ATMs in each region. Therefore, I will probably use colors of the Colorbrewer[6]. I will avoid to use the common used green-red coloration, because it often implies a valuation, which I don't want to influence.

In the stacked bar chart the relative count is mapped onto the height of the bar and the different states of the wheelchair property are shown inside this by different colors. As colors I need diverging colors, because the properties should be easy to distinguish and have no order. The plot is very effective for the abstract task of presenting quantitative variable depending on two different nominal variables.

5 Must-Have Features

My visualization is straight forward and does require multiple different visual features, that can be gathered in two different plots: A discrete heatmap showing the different regions with their associated color and a stacked bar chart acting

[6] `http://colorbrewer2.org`

Figure 2: Dummy filter.

as inspector. The resulting required visual features are self-explanatory and are directly provided by the APIs D3js, MapBox and Vega-Lite[7] I plan to use.

The interactive features are mainly based on filtering data and updating the bar chart on click:

- A filter for different operators e.g. plots should only include AMTs from Volksbank and Sparkasse or all except these from Deutsche Bank. Figure 2 gives an intention about how this could look.
- Similar for the wheelchair accessibility: Only with wheelchair access, only without, only where it is unknown.
- The user can clicks on regions to get the stacked bar chart.
- Hovering over the bars shows the concrete values.
- Zooming on the map must be possible s.t. small regions can be selected easily.

6 Optional Features

- If there is enough performance, updating the chart when hovering over would be a great instead of selecting different regions by a mouse click.
- The user can select multiple regions at once e.g. to get one bar chart containing the data of these regions or to get multiple charts for comparison. It is not trivial to visualize which regions are selected, because simply changing the color would affect the next selection of the user.

[7] `https://vega.github.io/vega-lite/`

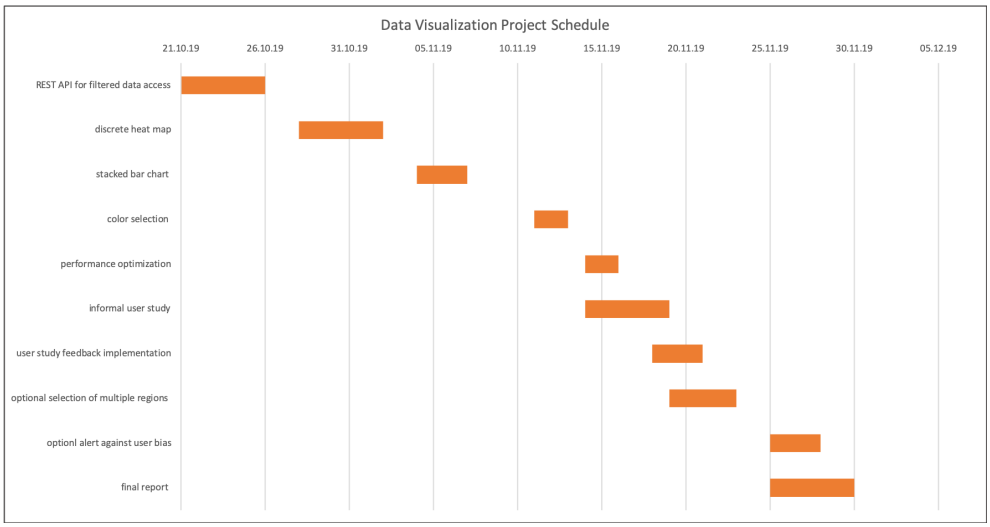

Figure 3: Project Schedule

- Avoiding cognitive bias: If the user only selects certain regions, e.g. only those with a very low number of ATMs or I a certain part of Germany, an alert is shown which informs the user about that or recommending him to try some other regions. This method is described in the work by Wall et al.[2].
- Some statistics are shown directly on the map or next to the bar chart e.g. the mean pairwise distance of the ATMs or the mean distance to a central point.

7 Project Schedule

I am not sure if I go with MapBox or a D3js for visualizing the heatmap. Since both have advantages and disadvantages, I will keep the server side backend as general as possible and try both solutions before I finally decide in the end of October.

The Gantt chart in Figure 3 shows the project schedule for the next 7 weeks, including a week buffer for unexpected problems.

References

[1] Jock Mackinlay. Automating the design of graphical presentations of relational information. *Acm Transactions On Graphics (Tog)*, 5(2):110–141, 1986.

[2] Emily Wall, John Stasko, and Alex Endert. Toward a design space for mitigating cognitive bias in vis.