Chapter 9

Visualisation tools and workflows

In which we start with a charting result from your Assignment 1, and take it a bit further...

9.1 Improving the population chart

9.1.1 Creating a single, wider chart

In assignment 1 you have been creating 3 charts. The specific assignment concentrated on wrangling the data and creating basic plots, not so much on the communication qualities of the charts. Let's concentrate on the population chart and see if we can improve that.

The part of the assignment code that creates the first chart, the municipal population chart, should look something like this:

```
############# Block 2 ################
# creating labels for the municipalities from the first
# three characters of the municipality name
labels = [key[0:3].upper() for key in  \
    sorted(municipality_population.keys())]
valuelist = [ municipality_population[key] for key in \
    sorted(municipality_population.keys())]

# Create a matplotlib figure with two columns
fig, ax = plt.subplots(1, 2)
width = 0.35

indices = range(len(labels))
bar1 = ax[0].bar(indices, valuelist, label="Population")
ax[0].set_xticks(indices)
```

```
ax[0].set_xticklabels(labels)
ax[0].tick_params(axis='x', rotation=90)
ax[0].set_ylabel('Population␣size')
ax[0].set_xlabel('Municipality')
ax[0].set_title('Population␣size␣per␣Municipality')
```

It should have resulted in a chart such as in figure 9.1.

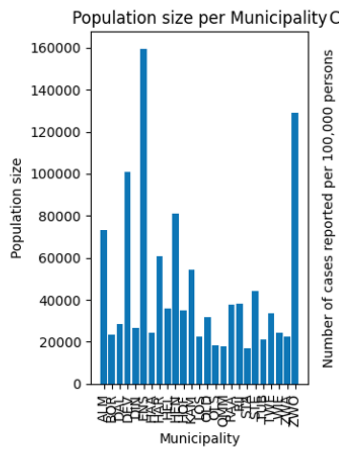

Figure 9.1: The original code created this chart

Ex 9.1

Since we will work only on this figure, you can delete (or comment out) any code after this. Then change the code to in block 2 to only create the one plot, and have it use the full plot width. The result should look like figure 9.2. So instead of two subplots, only use the one plot, and of course you also have to get rid of the indexed axes, so change the ax[0] elements. Also, do not forget to actually plot the figure. . . ! [as always, check the Answers chapter for solutions].

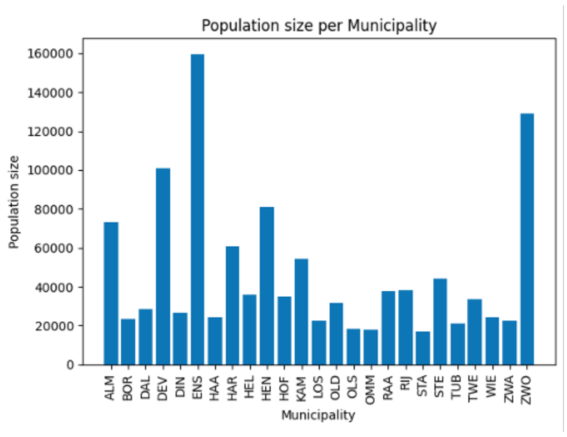

Figure 9.2: The population chart using the full width

9.1.2 Ordering by population size

Secondly, we want to make the chart more readable by ordering the bars by the population size, which is a more sensible ordering parameter than the municipality name. Carefully study the part of the code that creates the `valuelist` at the moment:

```
labels = [key[0:3].upper() for key in \
    sorted(municipality_population.keys())]
valuelist = [ municipality_population[key] for key in \
    sorted(municipality_population.keys())]
```

Ex 9.2

Change this code in such a way that:

- the `valuelist` (and thus the bars in the chart) are ordered by the value of the `municipality_population` property from high to low;
- the labels are no longer clipped at 3 characters, and also not turned into uppercase.

You should know by now how to sort things, and hopefully have realised that sorting works subtly differently for arrays, `lists`, `tuples` and `dicts`. A good overview for sorting `dicts` can be found at: `https://www.pythoncentral.io/how-to-sort-python-dictionaries-by-key-or-value/`. It is important to know that *sorting does not allow you to re-order a dictionary in-place.* You should be writing the ordered pairs into a new, empty dictionary. The result should look like figure 9.3.

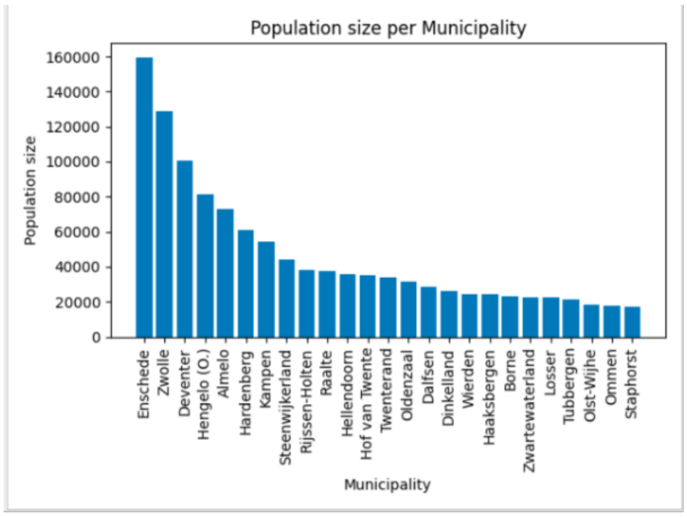

Figure 9.3: Chart ordered by population size

9.1.3 Changing to a horizontal bar chart

Bar graphs such as the ordered chart you created before work much better in understanding the data. However, making the connection between the bar height (the population values) and their label (the municipality name) is a bit difficult. In most cultures we are used to reading thing horizontally, so let's change our chart to offer its information that way.

Ex 9.3

Change the standard bar chart to a *horizontal* bar chart as shown in figure 9.4. `Matplotlib` considers the horizontal bar chart as a separate type of chart (not a simple rotation of a normal bar chart), so try to find out how it works from the `matplotlib` documentation...

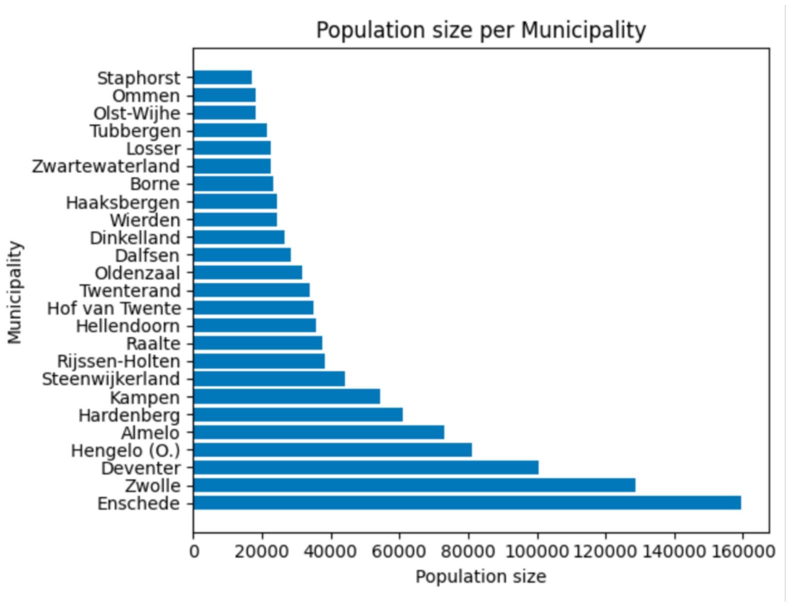

Figure 9.4: A horizontal Bar Chart

9.1.4 Remove 'chart junk' and change the formatting

Note that the ordering in horizontal bar charts works different from the ordering in normal bar charts: `Matplotlib` orders now from bottom to top. Also, this chart has what Tufte called 'chart junk' (in his 1983 book The Visual Display of Quantitative Information). *Chartjunk* refers to all visual elements in charts and graphs that are not necessary to comprehend the information represented on the graph, or that distract the viewer from this information.

Ex 9.4

As the last improvement for this chart, make some final changes:

- Reverse the ordering to having the largest values at the top;

- Remove the superfluous label on the label axis;
- Try to think of a useful way to simplify the value axis, which now has all these large numbers with many zero's;
- Change any other formatting you think will help the readability of he chart...

In the Answers document you can find *one* way of doing this, but try to be creative yourself and experiment.

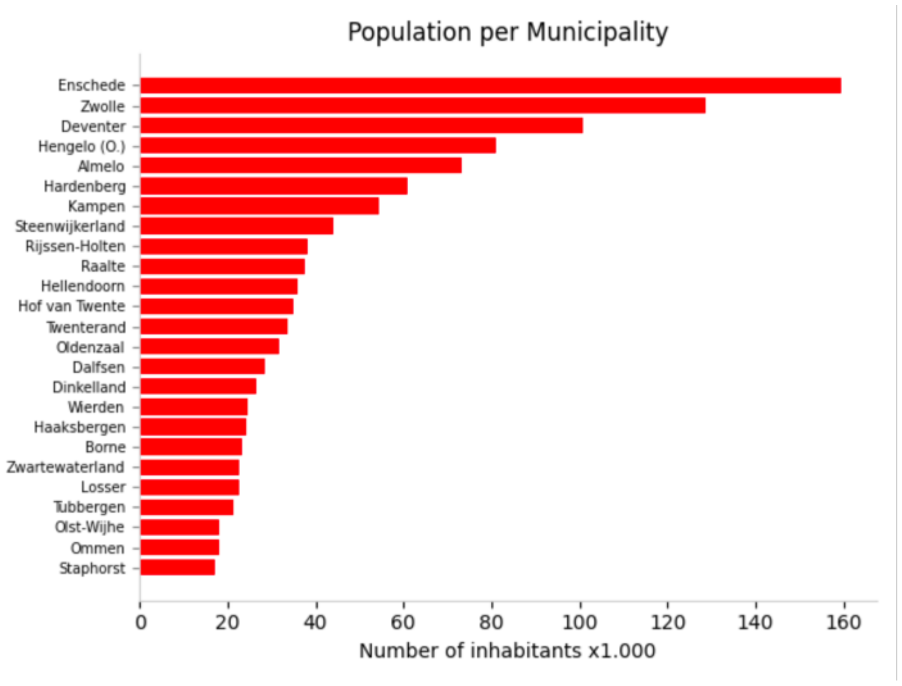

Figure 9.5: A possible solution of exercise 4

9.2 Mapping the data

There are many ways to have Python create *geographic charts*, otherwise known as *maps*. You will learn to use specific python modules later that enable you to do that in data-oriented ways, using GIS-like functionality. Bit even the simple `matplotlib` charting let's you do this, because you can plot X- and Y (or Latitude and Longitude) values directly on the two axes of a chart.

For the next exercises, you will need data. It is available on Canvas as file `overijssel_municipalities.geo.json` It is a single geoJSON file that holds the municipalities of Overijssel and their properties. GeoJSON is a version of normal

JSON that has geographic coordinates that express spatial features. To Python, any (Geo)JSON is essentially a `dict`.

The following GeoJSON object has two features (ITC and ITC Hotel).

```
{
   "type": "FeatureCollection",
   "features":
     [
       {
           "type": "Feature",
           "properties": {
              "description": "ITC"
                         },
           "geometry": {
              "type": "Point",
              "coordinates":[6.885885000228882,52.223790522038215]
                       }
        },
        {
             "type": "Feature",
             "properties": {
                 "description": "ITC␣Hotel"
                           },
             "geometry": {
                 "type": "Point",
                 "coordinates":[6.89068078994751,52.21800655772852]
                         }
         }
    ]
}
```

You can use the website http://geojson.io to see this GeoJSON points on top of a webmap.

Ex 9.5

Create Python code that loads the above JSON as a Python variable, and then print the following:

- For the first feature, print its properties description,
- The coordinates of the first feature,
- The geometry type of the second feature.

Ex 9.6

You can of course also load the GeoJSON from a file. As a final challenge, we ask you to load the file `overijssel_municipalities.geo.json` and plot it as a chart. Any type of chart you manage to create using ONLY MATPLOTLIB (!) is OK. You *can* reach the result shown in figure 9.6, but any plot is fine. We offer you the following tips:

- The geometry objects are arrays of arrays of longitude-latitude pairs. Such arrays can be plotted on the X,Y axis straight away with the `plt.plot(x,y)` chart type.
- The properties of the municipalities include a centroid Latitude and Longitude. These can be used also to plot along the X and Y axes, e.g. using the `plt.scatter(x,y)` scatterplot chart type.
- Picking apart the complicated structure of the GeoJSON requires you to understand its structure. Carefully study it, read about it and use some explorative `print` statement to see how you can get hold of the separate parts. See the listing below for some useful experiments.

```
#pick apart the structure of the geojson:
print (json_data.keys())
print (json_data["features"][0].keys())
print (json_data["features"][0]["geometry"].keys())
print (json_data["features"][0]["geometry"]["coordinates"])
print (json_data["features"][0]["properties"]["name"])
print (json_data["features"][0]["properties"]["no_inhabitants"])
print (json_data["features"][0]["properties"]["centroid_lon"])
```

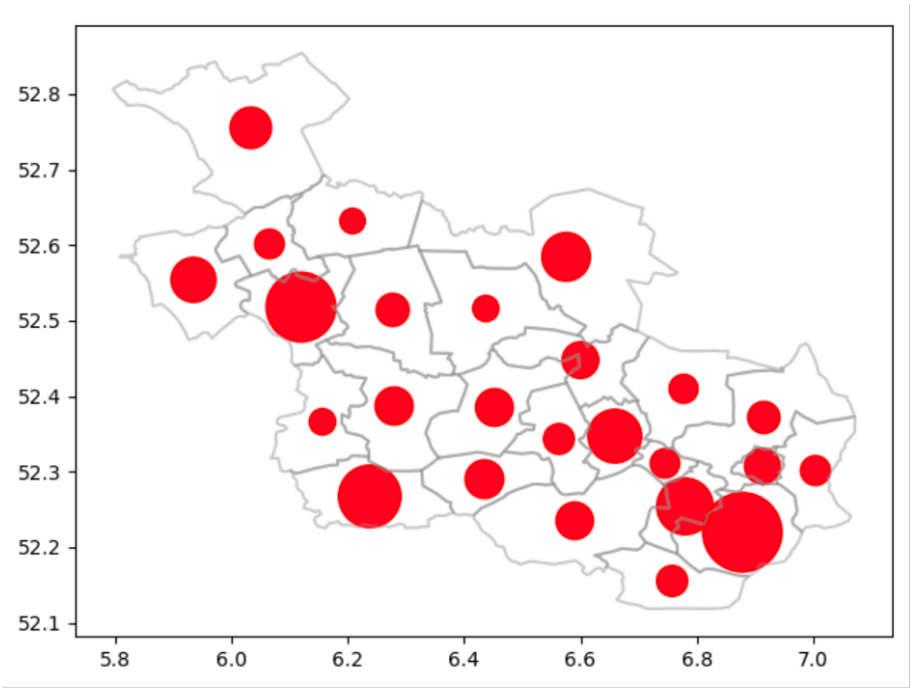

Figure 9.6: A possible solution of exercise 4