Answers to Exercises in Chapter 9: Visualisation tools and workflows

1. Making space; Creating three separate charts iso 1 double-column and 1 single

```
# Create a matplotlib figure with one column
# fig, ax = plt.subplots(1, 2)
fig, ax = plt.subplots()
          [... matplotlib code]
          [ ..replace all ax[0] by ax ...]
plt.show()
```

2. The population size per municipality: ordering more sensibly

Note: Sorting does not allow you to re-order the dictionary in-place. We are writing the ordered pairs in a new, empty dictionary.

```
# a new Dict:
municipality_population_by_value = {}
# sort the Dict keys by their values, reverse=True to get from high-low:
sorted_keys = sorted(municipality_population, key=municipality_population.get, reverse=True)
# fill new Dict with sorted tuples from old one:
for i in sorted_keys:
    municipality_population_by_value[i] = municipality_population[i]

# creating labels and the valuelist for the municipalities
labels = [key for key in municipality_population_by_value] # no need sorting here any more!
valuelist = [municipality_population_by_value[key] for key in
municipality_population_by_value]
```

3. Changing to a horizontal bar chart

```
indices = range(len(labels))
bar1 = ax.barh(indices, valuelist, label="Population")  # ax.barh i.s.o ax.bar
ax.set_yticks(indices)                                  # yticks i.s.o. xticks
ax.set_yticklabels(labels)                              # yticks i.s.o. xticks
ax.tick_params(axis='y', rotation=0)                    # y i.s.o. x, no rotation
ax.set_xlabel('Population size')                       # switch y- and xlabel
ax.set_ylabel('Municipality')
ax.set_title('Population size per Municipality')
```

4. Remove ‘chart junk’ and change the formatting

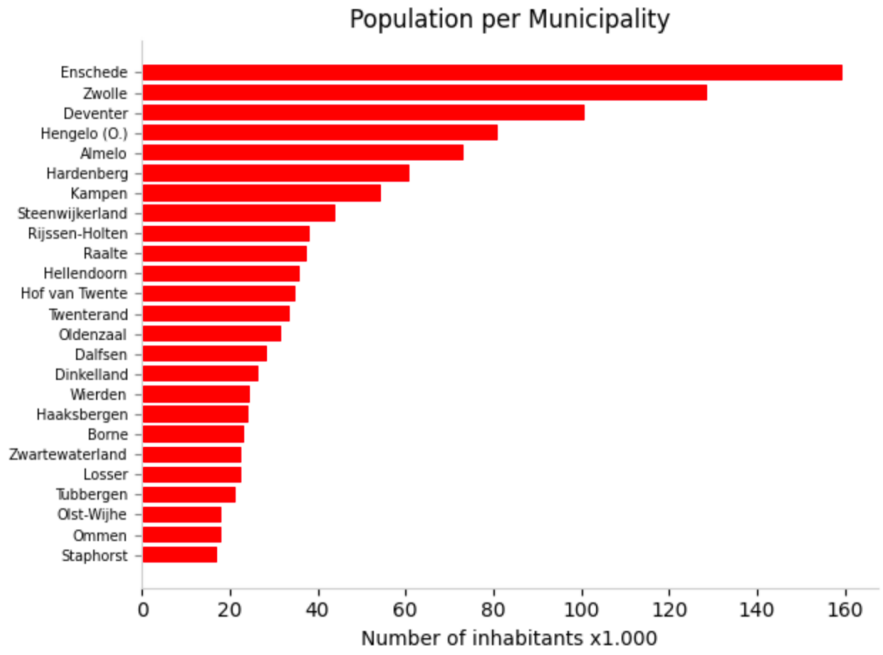
```
# a new Dict:
municipality_population_by_value = {}
# sort the Dict keys by their values, reverse=True to get from high-low:
sorted_keys = sorted(municipality_population, key=municipality_population.get, reverse=False)
# fill new Dict with sorted tuples from old one:
for i in sorted_keys:
    municipality_population_by_value[i] = municipality_population[i]

# creating labels for the municipalities
labels = [key for key in municipality_population_by_value] # no need for sorting here any
more!
# divide values by 1000:
valuelist = [ (municipality_population_by_value[key]/1000) for key in
municipality_population_by_value]
```

```
# Create a matplotlib figure with one column
fig, ax = plt.subplots()

indices = range(len(labels))
bar1 = ax.barh(indices, valuelist, label="Population", color="red")
ax.spines["bottom"].set_color("#cccccc")
ax.spines["top"].set_color("white")
ax.spines["right"].set_color("white")
ax.spines["left"].set_color("#cccccc")
ax.set_yticks(indices)
ax.set_yticklabels(labels, fontsize=7)
ax.tick_params(axis='y', rotation=0, color="grey")
ax.set_xlabel('Number of inhabitants x1.000')
ax.set_ylabel('')
ax.set_title('Population per Municipality')

plt.show()
```



5. Loading GeoJSON in Python

```
'''
Load JSON object into python, and then print into the console:
   For the first feature, print its properties description,
   The coordinates of the first feature,
   The geometry type of the second feature.
'''
##
import json

ITCjson = '''
{
  "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
        ]
      }
    }
  ]
}
'''

ITCData = json.loads(ITCjson)
print(ITCData["features"][0]["properties"]["description"])
print(ITCData["features"][0]["geometry"]["coordinates"])
print(ITCData["features"][1]["geometry"]["type"])
##
```

6. Creating a MAP plot

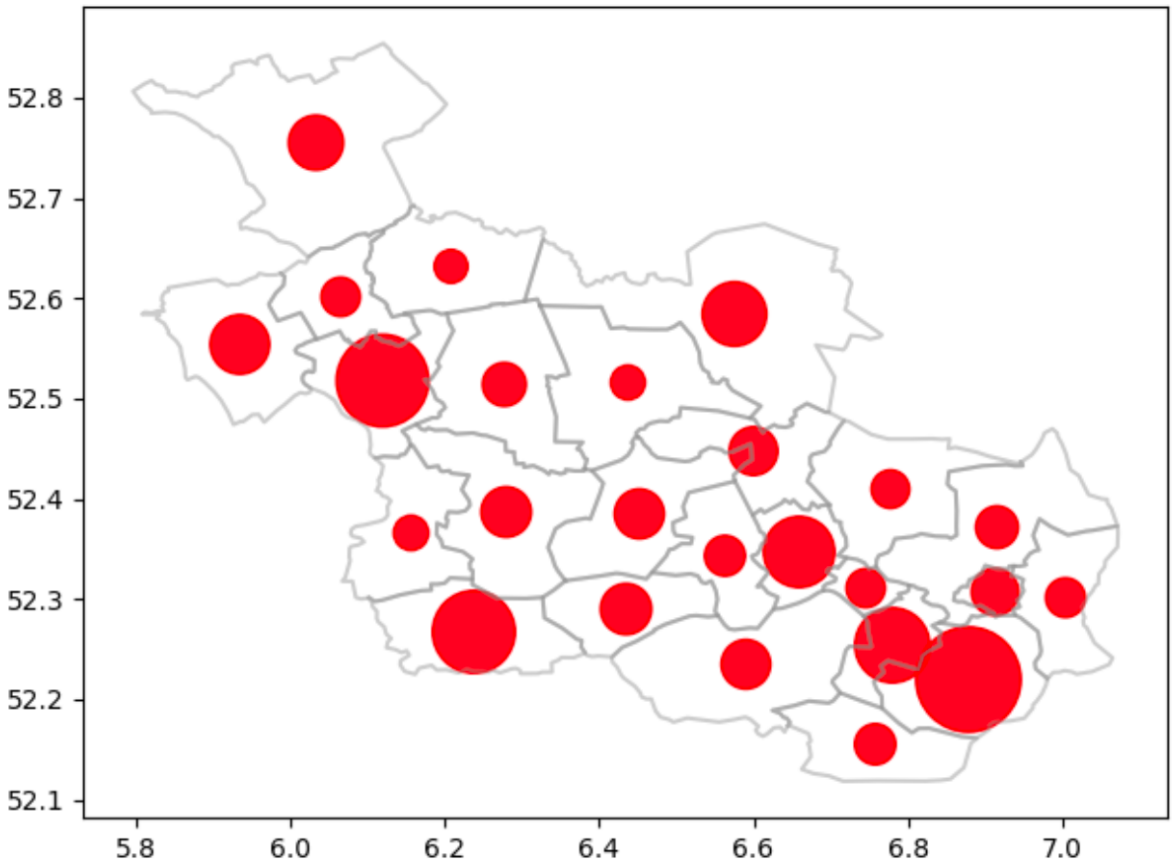
```
import matplotlib.pyplot as plt
import json
import math

with open("overijssel_municipalities.geo.json") as json_file:
    json_data = json.load(json_file)

#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"])

# for each feature (= a municipality) in the feature set:
for municipality in json_data["features"]:
    #get the coordinates arrays out of the geomtery object
    coords = municipality["geometry"]["coordinates"]
    x = [i for i,j in coords[0]]
    y = [j for i,j in coords[0]]
    plt.plot(x,y, c="grey", alpha=0.4)
    # get the
    centroid_x = municipality["properties"]["centroid_lon"]
    centroid_y = municipality["properties"]["centroid_lat"]
    inhabitants = municipality["properties"]["no_inhabitants"]
    # s = The marker size in points**2, thus we need to scale it down:
    symbolsize = inhabitants/100
    plt.scatter(centroid_x,centroid_y, c="red", alpha=0.8, s=symbolsize)

plt.show()
```


52.8
52.7
52.6
52.5
52.4
52.3
52.2
52.1
5.8
6.0
6.2
6.4
6.6
6.8
7.0