HWK6 Part 2: Geovisualization

Name: Dyab Asdi

UT EID: da32435

Name:

UT EID:

Date: 3/10/2025

In this portion of the homework we will be analyzing and visualizing geographic data; i.e., data that refers to geospatial entities. Geospatial entities can, for example, be particular places such as schools and libraries or political boundaries of cities or countries. Of course, this tutorial only scratches the surface. Consider this as a teaser into geovisualization, which in itself has become a branch of research and practice at the intersection of geography and visualization.

In [67]:
```
import altair as alt
import pandas as pd
```

Install packages

For this tutorial we will continue to rely on Altair and Pandas, but add **GeoPandas**, which will help us to work with DataFrames that contain spatial entities to carry out geometric analysis on them. As before, the pip install command is carried out via the shell, which is indicated by the exclamation mark at the beginning of the line:

In [68]:
```
!pip install geopandas
import geopandas as gpd
```

```
Requirement already satisfied: geopandas in c:\users\dyaba\desktop\school\spring 2025\data
visualization\.venv\lib\site-packages (1.0.1)
Requirement already satisfied: numpy>=1.22 in c:\users\dyaba\desktop\school\spring 2025\da
ta visualization\.venv\lib\site-packages (from geopandas) (2.2.1)
Requirement already satisfied: pyogrio>=0.7.2 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from geopandas) (0.10.0)
Requirement already satisfied: packaging in c:\users\dyaba\desktop\school\spring 2025\data
visualization\.venv\lib\site-packages (from geopandas) (24.2)
Requirement already satisfied: pandas>=1.4.0 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from geopandas) (2.2.3)
Requirement already satisfied: pyproj>=3.3.0 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from geopandas) (3.7.1)
Requirement already satisfied: shapely>=2.0.0 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from geopandas) (2.0.7)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\dyaba\desktop\school\spr
ing 2025\data visualization\.venv\lib\site-packages (from pandas>=1.4.0->geopandas) (2.9.
0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dyaba\desktop\school\spring 2025\d
ata visualization\.venv\lib\site-packages (from pandas>=1.4.0->geopandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from pandas>=1.4.0->geopandas) (2024.2)
Requirement already satisfied: certifi in c:\users\dyaba\desktop\school\spring 2025\data v
isualization\.venv\lib\site-packages (from pyogrio>=0.7.2->geopandas) (2025.1.31)
Requirement already satisfied: six>=1.5 in c:\users\dyaba\desktop\school\spring 2025\data
visualization\.venv\lib\site-packages (from python-dateutil>=2.8.2->pandas>=1.4.0->geopand
as) (1.17.0)
```

To access the data of the OpenStreetMap, we will install the handy package **OSMPythonTools**:

```
In [69]: !pip install --upgrade OSMPythonTools
```

```
Requirement already satisfied: OSMPythonTools in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (0.3.5)
Requirement already satisfied: beautifulsoup4 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from OSMPythonTools) (4.12.3)
Requirement already satisfied: geojson in c:\users\dyaba\desktop\school\spring 2025\data v
isualization\.venv\lib\site-packages (from OSMPythonTools) (3.2.0)
Requirement already satisfied: lxml in c:\users\dyaba\desktop\school\spring 2025\data visu
alization\.venv\lib\site-packages (from OSMPythonTools) (5.3.1)
Requirement already satisfied: matplotlib in c:\users\dyaba\desktop\school\spring 2025\dat
a visualization\.venv\lib\site-packages (from OSMPythonTools) (3.10.0)
Requirement already satisfied: numpy in c:\users\dyaba\desktop\school\spring 2025\data vis
ualization\.venv\lib\site-packages (from OSMPythonTools) (2.2.1)
Requirement already satisfied: pandas in c:\users\dyaba\desktop\school\spring 2025\data vi
sualization\.venv\lib\site-packages (from OSMPythonTools) (2.2.3)
Requirement already satisfied: ujson in c:\users\dyaba\desktop\school\spring 2025\data vis
ualization\.venv\lib\site-packages (from OSMPythonTools) (5.10.0)
Requirement already satisfied: xarray in c:\users\dyaba\desktop\school\spring 2025\data vi
sualization\.venv\lib\site-packages (from OSMPythonTools) (2025.1.2)
Requirement already satisfied: soupsieve>1.2 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from beautifulsoup4->OSMPythonTools) (2.6)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\dyaba\desktop\school\spring 20
25\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\dyaba\desktop\school\spring 2025\d
ata visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\dyaba\desktop\school\spring 2
025\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (4.55.3)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\dyaba\desktop\school\spring 2
025\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (1.4.8)
Requirement already satisfied: packaging>=20.0 in c:\users\dyaba\desktop\school\spring 202
5\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (24.2)
Requirement already satisfied: pillow>=8 in c:\users\dyaba\desktop\school\spring 2025\data
visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (11.1.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\dyaba\desktop\school\spring 20
25\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (3.2.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\dyaba\desktop\school\sprin
g 2025\data visualization\.venv\lib\site-packages (from matplotlib->OSMPythonTools) (2.9.
0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\dyaba\desktop\school\spring 2025\d
ata visualization\.venv\lib\site-packages (from pandas->OSMPythonTools) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\dyaba\desktop\school\spring 2025
\data visualization\.venv\lib\site-packages (from pandas->OSMPythonTools) (2024.2)
Requirement already satisfied: six>=1.5 in c:\users\dyaba\desktop\school\spring 2025\data
visualization\.venv\lib\site-packages (from python-dateutil>=2.7->matplotlib->OSMPythonToo
ls) (1.17.0)
```

Once we have the tools assembled, we can get started working with geospatial data. There are actually plenty of formats used to record geospatial data, but GeoJSON has become an important standard for exchanging geospatial data on the web. However, please note that GeoPandas can actually load many other vector-based data formats used in digital cartography, such as shapefiles and GeoPackage.

Import GeoJSON

Suppose we would like to get the geographic boundaries of Austin's neighborhoods. Akin to how we would read a JSON file with Pandas, we can also use `read_file()` provided by GeoPandas simply by passing the geojson filename and get a geographic DataFrame back:

In [70]:
```
neighborhoods = gpd.read_file("https://gist.githubusercontent.com/TieJean/40e9ccc69f0cc65
```

The main difference between a regular Pandas DataFrame is that a GeoDataFrame features a `geometry` column, which is a geoseries containing the points, paths, and polygons for each row. For example, if each row represents one neighborhood, the respective geometries would probably contain the geospatial boundaries...

💡 *Are you curious what the neighborhoods dataframe actually looks like? Take a look at it with the methods you know by now:*

In [71]:
```
neighborhoods.head()
```

Out[71]:

	name	cartodb_id	created_at	updated_at	geometry
0	Blackland	1	2013-02-17 09:28:09.692000+00:00	2013-02-17 09:28:09.956000+00:00	MULTIPOLYGON (((-97.72409 30.27926, -97.72514 ...
1	Bouldin Creek	2	2013-02-17 09:28:09.692000+00:00	2013-02-17 09:28:09.956000+00:00	MULTIPOLYGON (((-97.75962 30.24211, -97.76031 ...
2	Brentwood	3	2013-02-17 09:28:09.692000+00:00	2013-02-17 09:28:09.956000+00:00	MULTIPOLYGON (((-97.72354 30.33038, -97.72371 ...
3	Cherrywood	4	2013-02-17 09:28:09.692000+00:00	2013-02-17 09:28:09.956000+00:00	MULTIPOLYGON (((-97.70711 30.2892, -97.707 30....
4	Chestnut	5	2013-02-17 09:28:09.692000+00:00	2013-02-17 09:28:09.956000+00:00	MULTIPOLYGON (((-97.71991 30.27379, -97.7201 3...

Geographically speaking, the districts are defined by their geographic shapes, which are represented as polygons, each of which is a list of tuples of geographic coordinates. Next we add information about schools in Austin:

In [72]:
```
schools = gpd.read_file("https://gist.githubusercontent.com/TieJean/4e9f595eb3cb5dc60a960
schools.head()
```

Out[72]:

	FID	NCESSCH	LEAID	NAME	OPSTFIPS	STREET	CITY	STATE	ZIP	STI
0	81891	480000811280	4800008	ROOSTER SPRINGS EL	48	1001 BELTERRA DR	AUSTIN	TX	78737	
1	81892	480000813086	4800008	SYCAMORE SPRINGS EL	48	14451 SAWYER RANCH RD	AUSTIN	TX	78737	
2	81893	480000813151	4800008	SYCAMORE SPRINGS MIDDLE	48	14451 SAWYER RANCH RD	AUSTIN	TX	78737	
3	81942	480001609410	4800016	AUSTIN CAN ACADEMY	48	2406 ROSEWOOD AVE	AUSTIN	TX	78702	
4	82016	480004408055	4800044	WAYSIDE EDEN PARK ACADEMY	48	6215 MANCHACA RD	AUSTIN	TX	78745	

5 rows × 27 columns

💡 *Have a look at the schools as well, and compare the contents of the* `geometry` *columns in schools and districts. Do you notice anything?*

Query OpenStreetMap

OpenStreetMap (OSM) is "a collaborative project to create a free editable map of the world". As such it has millions of contributing users who have been collecting, updating and refining map data for over 15 years, which has generated a vastly comprehensive source of geographic data. It is by no means complete—whatever this would mean—but it is an impressively large geographic database and, of course, a map in itself, too.

OpenStreetMap has its own kind of query language, which is quite compact and can also be a source for errors. To make query formulation easier, you can either use the web interface overpass turbo or the `overpassQueryBuilder`, which provides access to the main parameters:

In [73]:
```python
from OSMPythonTools.overpass import overpassQueryBuilder

AustinAreaID = 3600000000 + 113314
# The area ID of a city is found by adding 3600000000 to the city's relation ID
# You can look up the relation ID of any city by searching on the OSM website; for example

library_query = overpassQueryBuilder(
    area=AustinAreaID, # the query can be constrained by an area of an item
    elementType='node', # which are points (OSM also has ways and relations)
    # the selector in the next line is really the heart of the query:
    selector='"amenity"="library"', # we're looking for libraries
    out='body', # body indicates that we want the data, not just the count
    includeGeometry=True # and we want the geometric information, too
)
```

```
library_query
```

Out[73]:
```
'area(3600113314)->.searchArea;(node["amenity"="library"](area.searchArea););out body ge
om;'
```

The output of above cell is the compact version of the query, which is carried out in the next step:

In [74]:
```
from OSMPythonTools.overpass import Overpass
overpass = Overpass()

lib_data = overpass.query(library_query)
```

The variable `lib_data` now already contains the result from the query against OSM. Let's have a look at it. With `nodes()` we can access the retrieved points. Let's take a look at the first entry:

In [75]:
```
lib_data.nodes()[0].tags()
```

Out[75]:
```
{'addr:state': 'TX',
 'amenity': 'library',
 'ele': '163',
 'gnis:feature_id': '2360810',
 'name': 'Texas State Law Library',
 'source': 'USGS Geonames'}
```

Similarly, we can also access the geometry, which in this case is just a point:

In [76]:
```
lib_data.nodes()[0].geometry()
```

Out[76]:
```
{"coordinates": [-97.741983, 30.276236], "type": "Point"}
```

Next, we use the compact form of a list comprehension to extract the libraries' names and coordinates:

In [77]:
```
libraries = [ (lib.tag("name"), lib.geometry() ) for lib in lib_data.nodes()]
```

... which we turn into a GeoDataFrame. By naming the second column `geometry` we indicate towards GeoPandas to interpret the coordinates as geographic locations:

In [78]:
```
libraries = gpd.GeoDataFrame(libraries, columns = ['name', 'geometry'])
libraries.head()
```

Out[78]:

	name	geometry
0	Texas State Law Library	POINT (-97.74198 30.27624)
1	Lantana Free Tiny Library	POINT (-97.87417 30.24811)
2	Austin Public Library - St. John Branch	POINT (-97.69372 30.33205)

🗺️ Present

When we have geospatial data readily available as GeoDataFrames, we can now map them with Altair.

(There are other mapping libraries for Python, such as Folium, that provide additional functionalities. Altair's geovis features are basic, but do provide some variety of techniques and have the benefit to work consistently with the other chart types we covered.)

Markers on maps

A simple start is placing locations on a base map and adding a bit of further information via tooltips. Let's do this with Austin's schools! First, we can have another look at the attributes:

In [79]:
```
schools.head()
```

Out[79]:

	FID	NCESSCH	LEAID	NAME	OPSTFIPS	STREET	CITY	STATE	ZIP	STF
0	81891	480000811280	4800008	ROOSTER SPRINGS EL	48	1001 BELTERRA DR	AUSTIN	TX	78737	
1	81892	480000813086	4800008	SYCAMORE SPRINGS EL	48	14451 SAWYER RANCH RD	AUSTIN	TX	78737	
2	81893	480000813151	4800008	SYCAMORE SPRINGS MIDDLE	48	14451 SAWYER RANCH RD	AUSTIN	TX	78737	
3	81942	480001609410	4800016	AUSTIN CAN ACADEMY	48	2406 ROSEWOOD AVE	AUSTIN	TX	78702	
4	82016	480004408055	4800044	WAYSIDE EDEN PARK ACADEMY	48	6215 MANCHACA RD	AUSTIN	TX	78745	

5 rows × 27 columns

We will now create a simple map with markers in the form of an Altair chart consisting of two layers:

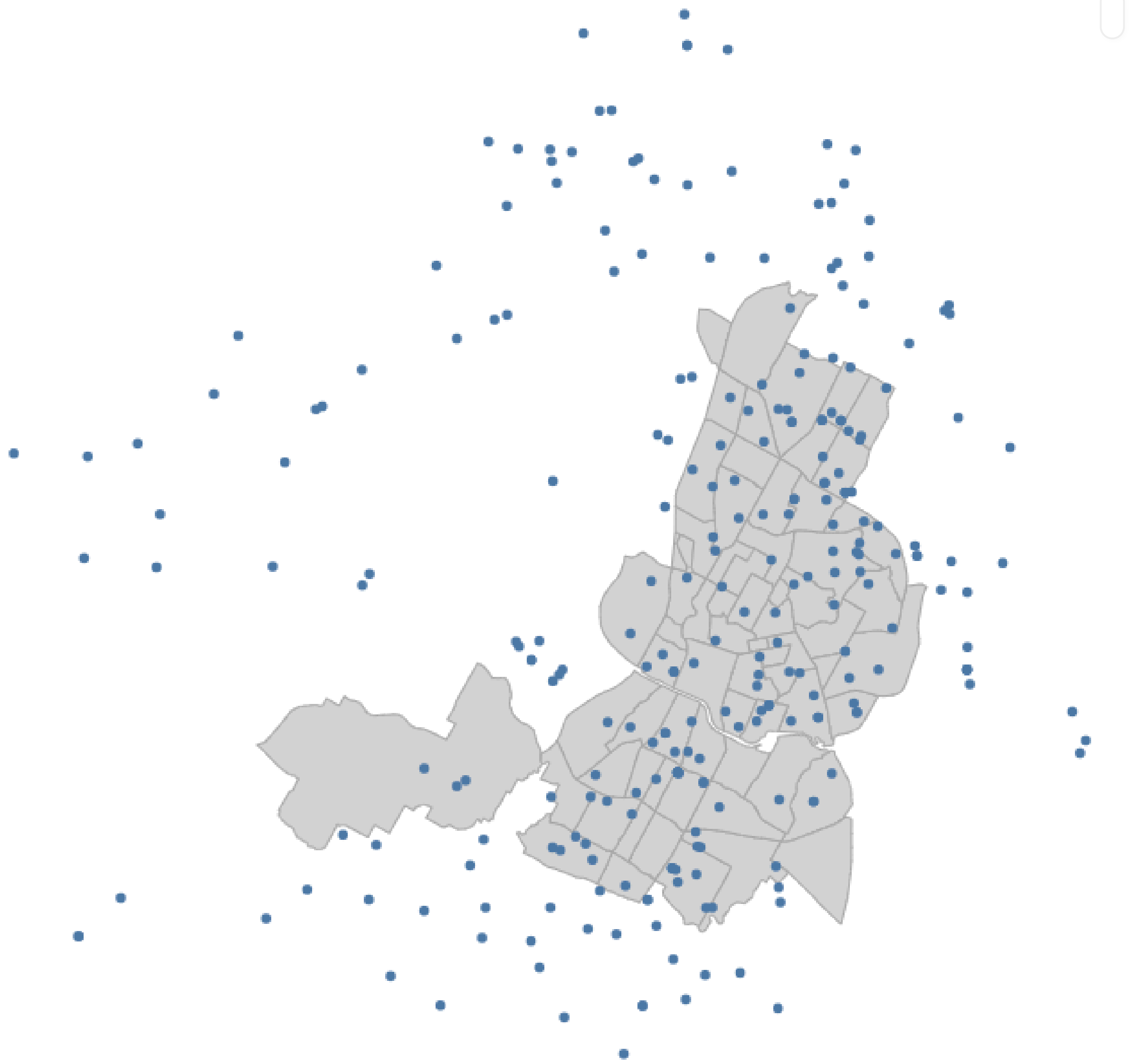
1. The `neighborhoods` form the lower layer representing their boundaries and the overall geographic shape of Austin
2. The `schools` are the points of interests that are displayed on top

When putting the two layers together they should actually refer to the same geographic location to make sense. Here the neighborhoods and schools both refer to Austin. Also note that the order when the charts are added together determines the vertical order: first the basemap and then markers on top.

In [80]:
```
# 1.  mark_geoshape transparently uses the geometry column
basemap = alt.Chart(neighborhoods).mark_geoshape(
    # add some styling to reduce the salience of the basemap
    fill="lightgray", stroke="darkgray",
).encode(
    tooltip = ['name'],
).properties(width=600, height=600)
```

```
# 2.  we use mark_circle to have more control over visual variables
markers = alt.Chart(schools).mark_circle(opacity=1).encode(
    # point latitude & longitude to coordinates in geometry column
    longitude='geometry.coordinates[0]:Q',
    latitude='geometry.coordinates[1]:Q',
    tooltip=['NAME', 'STREET', 'ZIP'],
)

# combine the two layers
basemap + markers
```

Out[80]:



Dot density maps

Let's use the open maps data set again, and plot New York's trees. Note, we will have to disable the max rows since there are more than 5,000 trees returned from the query.

In [81]:

```
NewYorkCityAreaID = 3600000000 + 175905

# 1. prepare query (and directly include the location lookup)
tree_query = overpassQueryBuilder(
    area=NewYorkCityAreaID,
    elementType='node',
    selector='"natural"="tree"',
    out='body',
    includeGeometry=True
)

# 2. execute query (and give it a bit more time to finish)
tree_data = overpass.query(tree_query, timeout=60)

# 3. get ids and coordinates of trees
tree_locs = [ (tree.id(), tree.geometry()) for tree in tree_data.nodes()]

# 4. create GeoDataFrame
trees = gpd.GeoDataFrame(tree_locs, columns=["id", "geometry"])

trees.head()
```

Out[81]:

	id	geometry
0	207694783	POINT (-73.96385 40.66462)
1	1201708558	POINT (-73.93184 40.8551)
2	1201708559	POINT (-73.93239 40.85584)
3	1201708560	POINT (-73.93235 40.85594)
4	1201708561	POINT (-73.93268 40.8559)

In [82]:

```
alt.data_transformers.disable_max_rows()
treemap = alt.Chart(trees).mark_circle(
    # reduce the visual presence of each element
    size=5,
    # with a low dot opacity we can use overplotting to indicate densities
    opacity=.25,
    # a natural choice
    color="green"
).encode(
    longitude='geometry.coordinates[0]:Q',
    latitude='geometry.coordinates[1]:Q'
).properties(width=600, height=600)

treemap
```

Out[82]:

Choropleth maps

Finally, let's create the geovisualization that uses the fill color of geospatial shapes to encode quantitative data. To illustrate this, we will visualize the population densities around the world. We will use area and population information from GeoNames and get the geographic shapes of countries from a geojson file.

In [83]:
```
# load country data from geonames CSV
geonames = pd.read_csv("./countryInfoCSV.csv", sep='\t')
# select four columns
geonames = geonames[['name', 'iso alpha3', 'areaInSqKm', 'population']]
# set index to country code
geonames = geonames.set_index("iso alpha3")

geonames.head()
```

Out[83]:

iso alpha3	name	areaInSqKm	population
AND	Andorra	468.0	77006
ARE	United Arab Emirates	82880.0	9630959
AFG	Afghanistan	647500.0	37172386
ATG	Antigua and Barbuda	443.0	96286
AIA	Anguilla	102.0	13254

Next we collect the geographic boundaries and `simplify` them a bit, as they have more detail than what we need here:

In [84]:
```
# load country's polygons from datahub
polygons = gpd.read_file("https://gist.githubusercontent.com/TieJean/f739f67075108868059b
# remove country names, as we have them already
polygons = polygons.drop(columns=["name"])
# set index to country code
polygons = polygons.set_index("id")
# reduce the complexity of the shapes
polygons.geometry = polygons.geometry.simplify(.1)

polygons.head(20)
```

Out[84]:

id	geometry
AFG	POLYGON ((61.21082 35.65007, 62.23065 35.27066...
AGO	MULTIPOLYGON (((16.32653 -5.87747, 16.86019 -7...
ALB	POLYGON ((20.59025 41.8554, 20.46318 41.51509,...
ARE	POLYGON ((51.57952 24.2455, 51.75744 24.29407,...
ARG	MULTIPOLYGON (((-65.5 -55.2, -66.45 -55.25, -6...
ARM	POLYGON ((43.58275 41.09214, 44.97248 41.24813...
ATA	MULTIPOLYGON (((-59.5721 -80.04018, -60.15966 ...
ATF	POLYGON ((68.935 -48.625, 69.58 -48.94, 70.525...
AUS	MULTIPOLYGON (((145.39798 -40.79255, 146.36412...
AUT	POLYGON ((16.97967 48.1235, 16.90375 47.71487,...
AZE	MULTIPOLYGON (((45.00199 39.74, 45.29814 39.47...
BDI	POLYGON ((29.34 -4.49998, 29.27638 -3.29391, 2...
BEL	POLYGON ((3.31497 51.34578, 4.04707 51.26726, ...
BEN	POLYGON ((2.6917 6.25882, 1.86524 6.14216, 1.6...
BFA	POLYGON ((-2.8275 9.64246, -3.5119 9.90033, -3...
BGD	POLYGON ((92.67272 22.04124, 92.65226 21.32405...
BGR	POLYGON ((22.65715 44.23492, 22.94483 43.82378...
BHS	MULTIPOLYGON (((-77.53466 23.75975, -77.78 23....
BIH	POLYGON ((19.00549 44.86023, 19.36803 44.863, ...
BLR	POLYGON ((23.48413 53.9125, 24.45068 53.9057, ...

As both DataFrames use the three-letter country codes as indices we can join them like this (join uses the index by default, so we don't have to specify what to join on):

In [85]:
```python
# inner means that we keep only those countries
# for which we have geometric and attribute data
countries = polygons.join(geonames, how='inner')

countries.tail()
```

Out[85]:

id	geometry	name	areaInSqKm	population
PSE	POLYGON ((35.54566 32.39399, 35.39756 31.48909...	Palestine	5970.0	4569087
YEM	POLYGON ((53.10857 16.65105, 52.38521 16.38241...	Yemen	527970.0	28498687
ZAF	POLYGON ((31.521 -29.25739, 30.05572 -31.14027...	South Africa	1219912.0	57779622
ZMB	POLYGON ((32.75938 -9.2306, 33.23139 -9.67672,...	Zambia	752614.0	17351822
ZWE	POLYGON ((31.19141 -22.25151, 29.43219 -22.091...	Zimbabwe	390580.0	14439018

Visualizing area or population in a choropleth map, arguably, makes little sense. So let's compute population densities:

In [86]:
```
countries["density"] = countries["population"] / countries["areaInSqKm"]
countries.head()
```

Out[86]:

id	geometry	name	areaInSqKm	population	density
AFG	POLYGON ((61.21082 35.65007, 62.23065 35.27066...	Afghanistan	647500.0	37172386	57.409090
AGO	MULTIPOLYGON (((16.32653 -5.87747, 16.86019 -7...	Angola	1246700.0	30809762	24.713052
ALB	POLYGON ((20.59025 41.8554, 20.46318 41.51509,...	Albania	28748.0	2866376	99.706971
ARE	POLYGON ((51.57952 24.2455, 51.75744 24.29407,...	United Arab Emirates	82880.0	9630959	116.203656
ARG	MULTIPOLYGON (((-65.5 -55.2, -66.45 -55.25, -6...	Argentina	2766890.0	44494502	16.081052

Keep only those countries with valid density value and turn these densities into integers:

In [87]:
```
countries = countries[countries['density'].notna()]
countries.density = countries.density.round(0).astype(int)
```

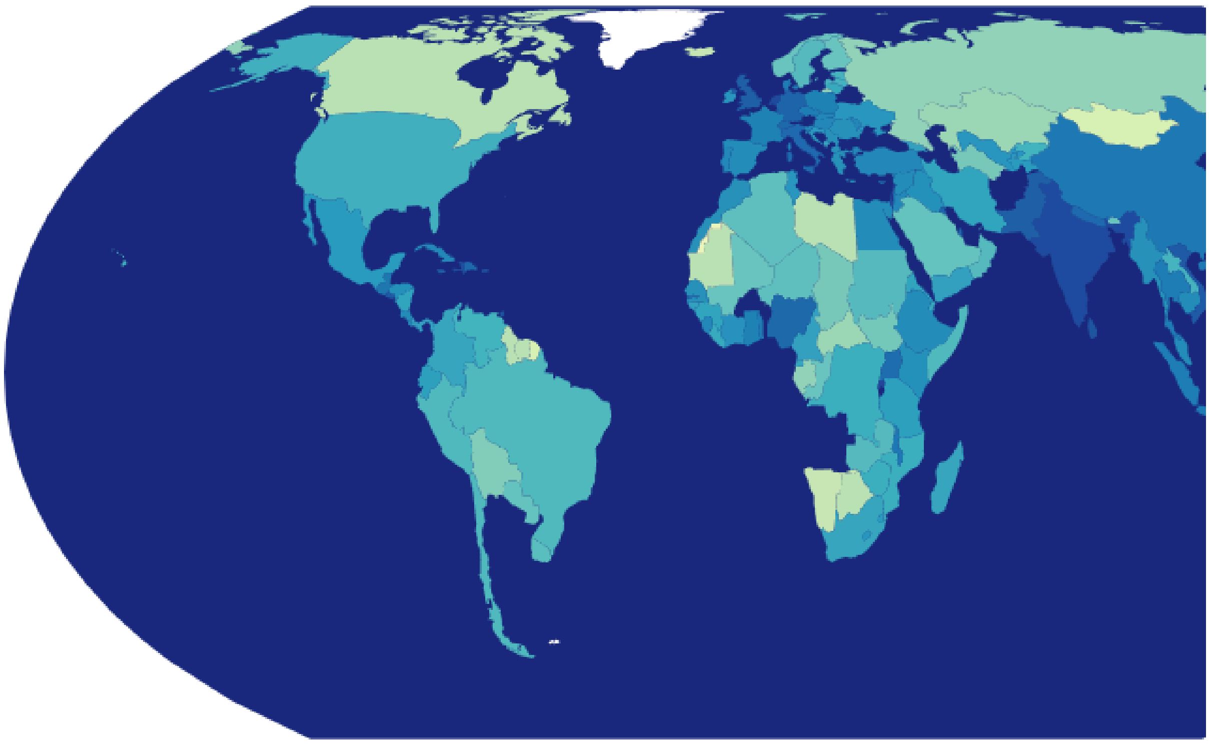
There is one ‘country’ that is not really one, which is Antarctica. We'll remove this from the list here.

In [88]:
```
countries = countries.drop("ATA")
```

Finally, we draw the chart using Altair's `mark_geoshape()` method. The distribution of densities is highly skewed, due to very small countries with relatively high population numbers, such as Monaco. To spread out the low and high density values we use a logarithmic scale and set the domain between 1 and 1000. Note that the domain has to end in a multiple of the base, which is by default 10.

In [89]:
```
alt.Chart(countries).mark_geoshape().encode(
    color=alt.Color('density', scale=alt.Scale(type="log", domain=[1,1000] )),
    tooltip=['name', 'areaInSqKm', 'population', 'density']
).properties(
    width=800,
    height=600
)
```

Out[89]:



The map is shown in the default Mercator projection, which particularly distorts the area sizes of North America, Europe and Russia in contrast to Africa, Southern Asia and parts of South America.

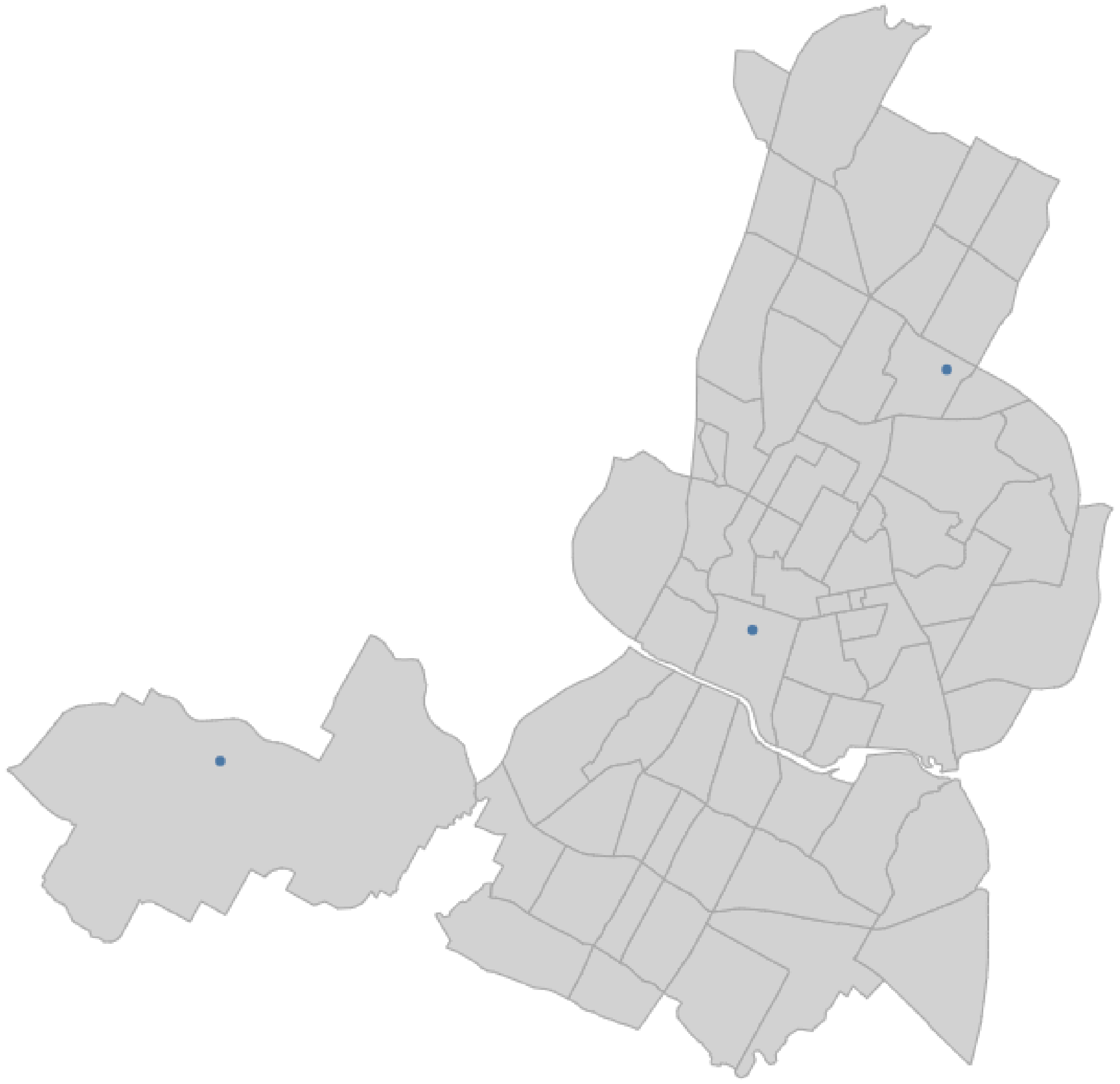
💡 *You can change the projection used above to one that does not distort area sizes as much (see this list for options).*

Your Turn

Q1 (10 points). Create a visualization with Austin's neighborhoods overlayed with Austin's libraries. The tool tip should provide necessary information to identify each neighborhood and library.

In [90]:
```
basemap = alt.Chart(neighborhoods).mark_geoshape(
    fill="lightgray", stroke="darkgray",
).encode(
    tooltip = ['name'],
).properties(width=600, height=600)

markers = alt.Chart(libraries).mark_circle(opacity=1).encode(
    longitude='geometry.coordinates[0]:Q',
    latitude='geometry.coordinates[1]:Q',
    tooltip=['name'],
)

basemap + markers
```

Out[90]:



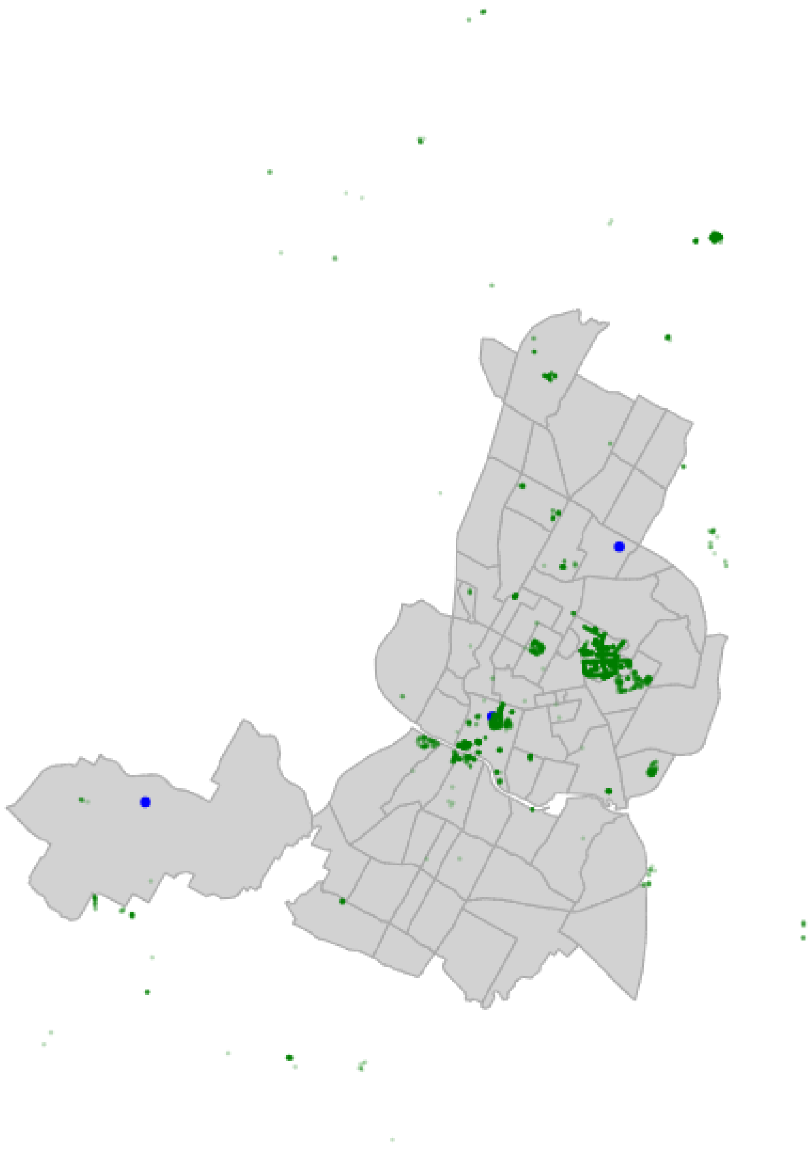
Q2 (5 points). Do you think the open data source for Austin's libraries is reliable? Why or why not? Answer in the Markdown cell below.

The data is not reliable, as there are more than 3 libraries in Austin. The dataset might be outdated or is not accounting for smaller libraries. UT itself has 10 libraries, so this data is not reliable.

Q3 (10 points). Add the location of all the trees in Austin (according to Open Street Map) to the visualization you just created in Q1.

In [91]:
```
AustinAreaID = 3600000000 + 113314
tree_query = overpassQueryBuilder(
    area=AustinAreaID,
    elementType='node',
    selector='"natural"="tree"',
    out='body',
    includeGeometry=True
)
austin_tree_data = overpass.query(tree_query, timeout=60)

austin_tree_locs = [(tree.id(), tree.geometry()) for tree in austin_tree_data.nodes()]

trees = gpd.GeoDataFrame(austin_tree_locs, columns=["id", "geometry"])


basemap = alt.Chart(neighborhoods).mark_geoshape(
    fill="lightgray", stroke="darkgray",
).encode(
    tooltip = ['name'],
).properties(width=600, height=600)

markers = alt.Chart(libraries).mark_circle(opacity=1, color = 'blue').encode(
    longitude='geometry.coordinates[0]:Q',
    latitude='geometry.coordinates[1]:Q',
    tooltip=['name'],
)

alt.data_transformers.disable_max_rows()
treemap = alt.Chart(trees).mark_circle(
    # reduce the visual presence of each element
    size=5,
    # with a low dot opacity we can use overplotting to indicate densities
    opacity=.25,
    # a natural choice
    color="green"
).encode(
    longitude='geometry.coordinates[0]:Q',
    latitude='geometry.coordinates[1]:Q'
).properties(width=600, height=600)

basemap + markers + treemap
```

Out[91]:



Q4 (10 points). Create a global population density choropleth with the Albers map projection.

In [92]:
```
alt.Chart(countries).mark_geoshape().encode(
    color=alt.Color('density', scale=alt.Scale(type="log", domain=[1,1000] )),
    tooltip=['name', 'areaInSqKm', 'population', 'density']
).properties(
    width=800,
    height=600
).project(
    type = "albers"
)
```

Out[92]:

Sources

Tutorials & Documentation

- Specifying Geospatial Data in Altair — Altair 4.1.0 documentation
- GeoPandas
- OSMPythonTools

Data

- OpenStreetMap
- GeoNames
- Data.gov