

Lecture 11 | Geospatial Data I

Max Pellert (<https://mpellert.at>)

IS 616: Large Scale Data Analysis and Visualization

Why do we want to visualize geospatial data?

Many datasets contain information linked to locations in the physical world.

For example, in an ecological study, a dataset may list where specific plants or animals have been found. Similarly, in a socioeconomic or political context, a dataset may contain information about where people with specific attributes (such as income, age, or educational attainment) live and how they vote, or where man-made objects (e.g., bridges, roads, buildings) have been constructed.

In all these cases, it can be helpful to visualize the data in their proper geospatial context, i.e. to show the data on a realistic map or alternatively as a map-like diagram.

Maps tend to be intuitive to readers but they can be challenging to design.

We need to think about concepts such as map projections and whether for our specific application the accurate representation of angles or areas is more critical.

We will also talk about **choropleth maps**, that represent data values as differently colored spatial areas. Choropleth maps can at times be very useful and at other times highly misleading.

As an alternative, we can construct map-like diagrams called **cartograms**, which may purposefully distort map areas or represent them in stylized form, for example as equal-sized squares.



north pole

meridians

parallels

equator

The earth is approximately a sphere. The two locations where the axis of rotation intersects with the spheroid are called the poles (**north pole** and **south pole**). We can separate the spheroid into two hemispheres, the **northern and the southern hemisphere**, by drawing a line equidistant to both poles around the spheroid. This line is called the **equator**.

The lines emanating from the north pole and running south are called **meridians**, and the lines running orthogonal to the meridians are called **parallels**. All meridians have the same length but parallels become shorter the closer we are to either pole.

To uniquely specify a location on the earth, we need three pieces of information: where we are located along the direction of the equator (the **longitude**), how close we are to either pole when moving in 90° to the equator (the **latitude**) and how far we are from the earth's center (the **altitude**).



Longitude and latitude

Longitude, latitude, and altitude are specified relative to a reference system called the **datum**.

The datum specifies properties such as the shape and size of the earth and the location of zero longitude, latitude, and altitude. One widely used datum is the “World Geodetic System (WGS) 84”, which is used by the Global Positioning System (GPS).

For now, we will forget about altitude and just concern us with longitude and latitude.

Longitude and latitude

Both longitude and latitude are angles, expressed in degrees.

Degrees longitude measure how far east or west a location lies. We can refer to meridians as lines of equal longitude and all meridians terminate at the two poles. The prime meridian, corresponding to 0° longitude, runs through the village of Greenwich in the United Kingdom.

The meridian opposite to the prime meridian lies at 180° longitude (also referred to as 180°E), which is equivalent to -180° longitude (also referred to as 180°W), near the international date line.

Longitude and latitude

Degrees latitude measure how far north or south a location lies.

The equator corresponds to 0° latitude, the north pole corresponds to 90° latitude (also referred to as 90°N), and the south pole corresponds to -90° latitude (also 90°S). Lines of equal latitude are referred to as parallels, since they run parallel to the equator.

All meridians have the same length, corresponding to half of a great circle around the globe. The length of parallels depends on their latitude: The longest parallel is the equator, at 0° latitude, and the shortest parallels lie at the north and south poles, 90°N and 90°S , with length zero.

Projection

The challenge in map-making is that we need to take the spherical surface of the earth and flatten it out so we can display it on a map.

This process, called **projection**, necessarily introduces distortions, because a curved surface cannot be projected exactly onto a flat surface.

Specifically, the projection can preserve either angles or areas but not both. A projection that does the former is called **conformal** and a projection that does the latter is called **equal-area**.

Projection

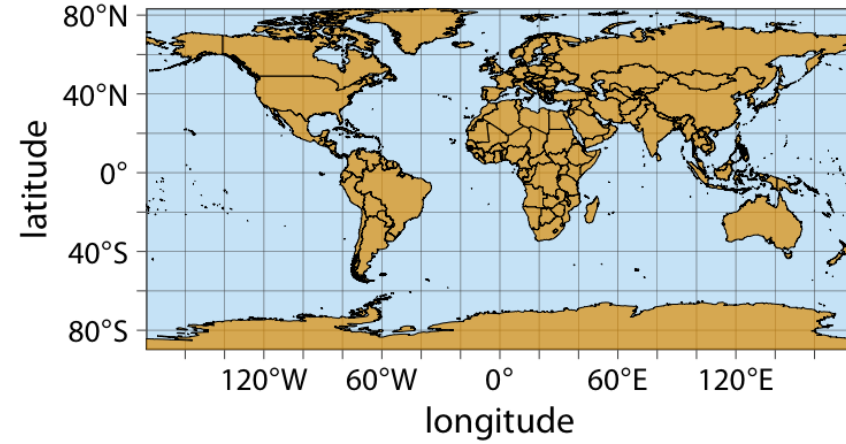
Other projections may preserve neither angles nor areas but instead preserve other quantities of interest, such as distances to some reference point or line. Finally, some projections attempt to strike a compromise between preserving angles and areas.

These compromise projections are frequently used to display the entire world in an aesthetically pleasing manner, and they accept some amount of both angular and area distortion.

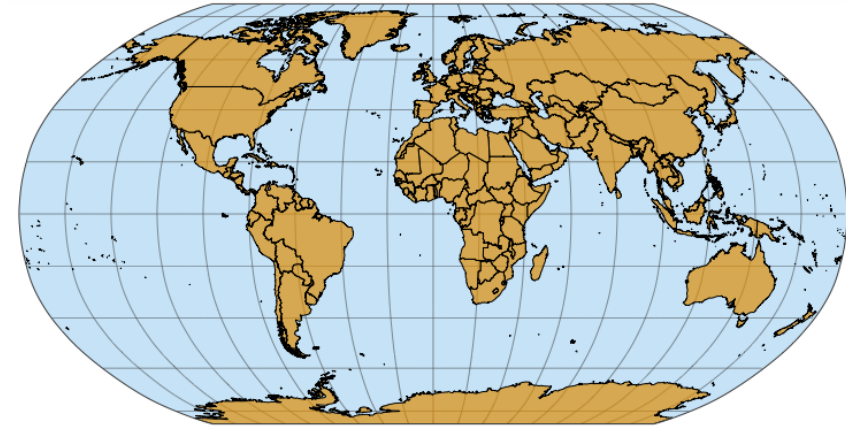
To systematize and keep track of different ways of projecting parts or all of the earth for specific maps, various standards bodies and organizations maintain registries of projections.

Many ways to project onto a 2D plane

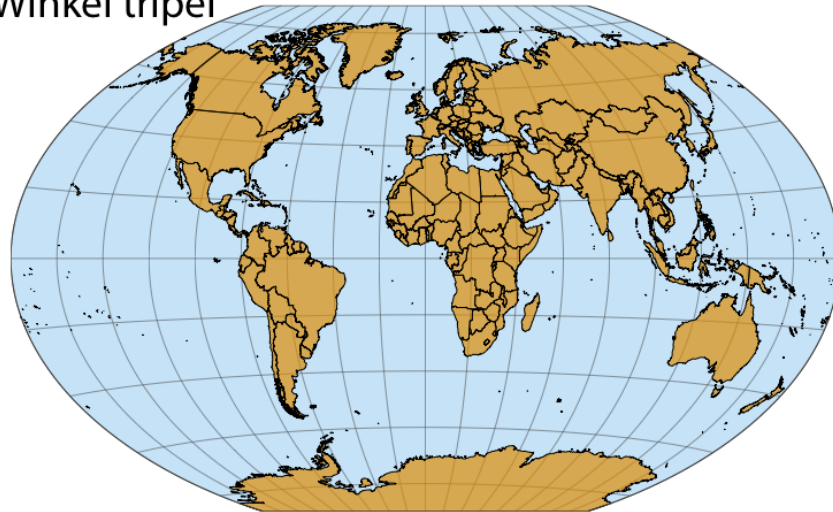
Cartesian longitude and latitude



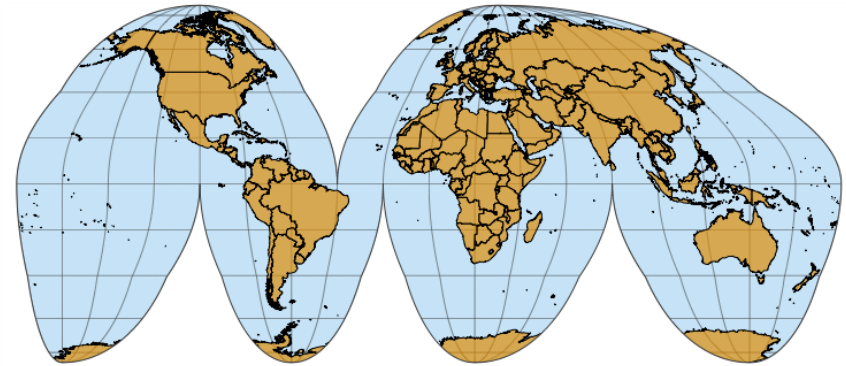
Robinson

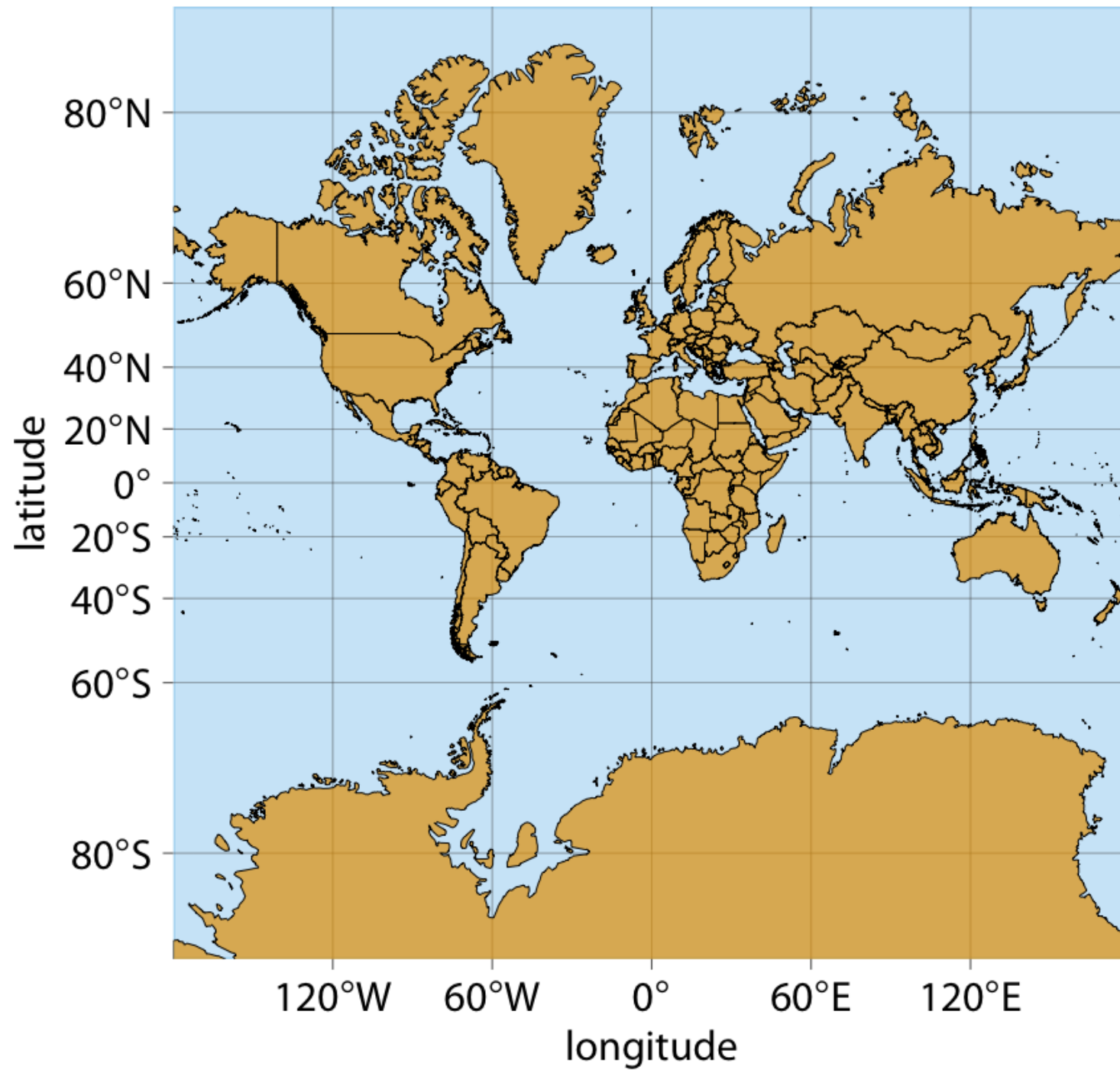


Winkel tripel



Interrupted Goode homolosine

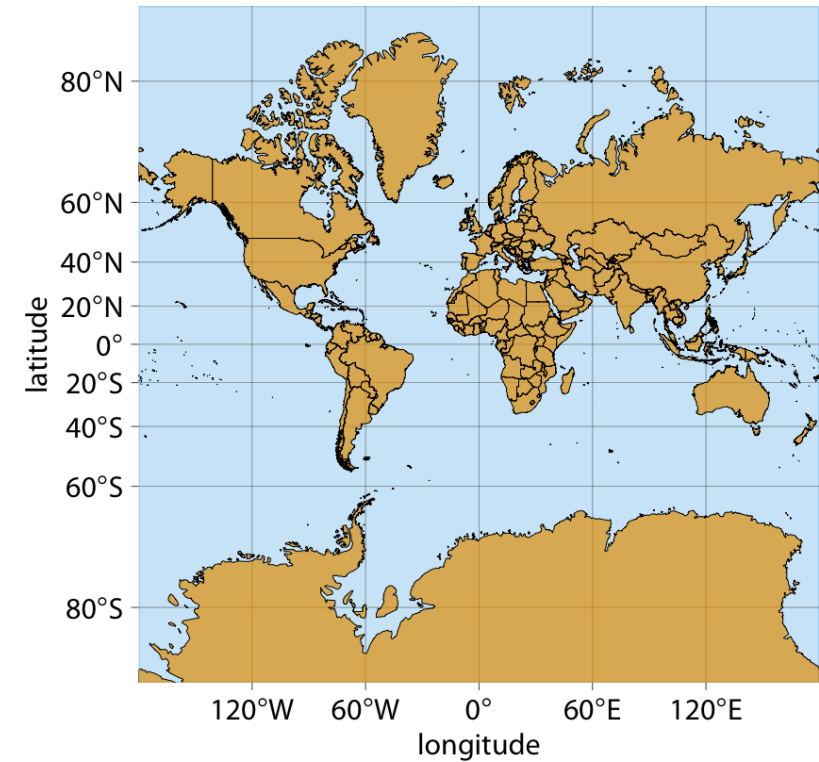




One of the earliest map projections in use, the **Mercator projection**, was developed in the 16th century for nautical navigation.

The Mercator projection maps the globe onto a cylinder and then unrolls the cylinder to arrive at a rectangular map.

It is a conformal projection that accurately represents shapes but introduces severe area distortions near the poles.



Meridians in this projection are evenly spaced vertical lines, whereas parallels are horizontal lines whose spacing increases the further we move away from the equator.

The spacing between parallels increases in proportion to the extent to which they have to be stretched closer to the poles to keep meridians perfectly vertical.

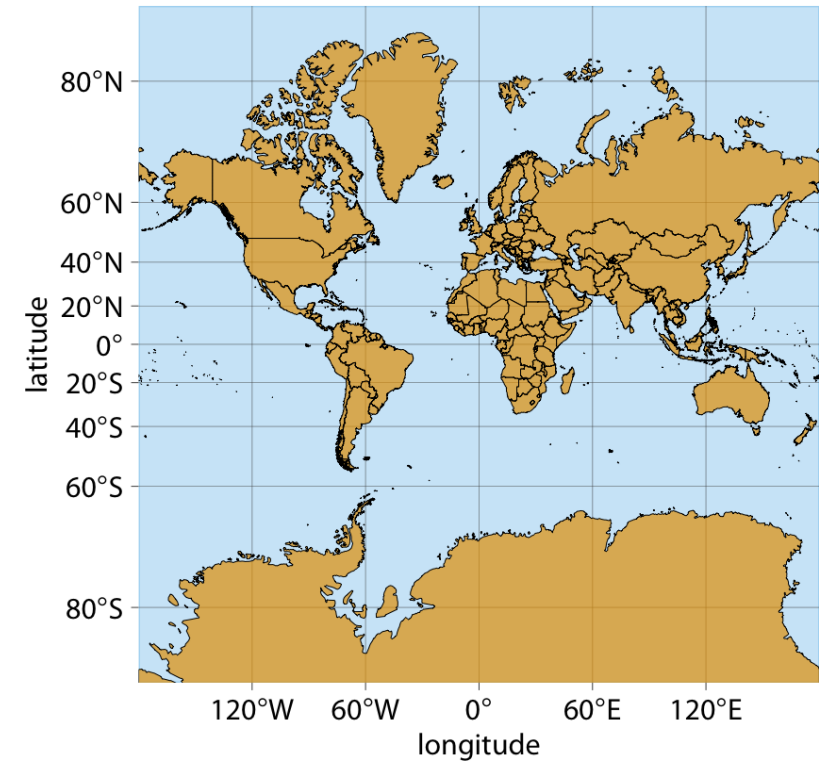
Because of the severe area distortions it produces, the Mercator projection has fallen out of favor for maps of the entire world.

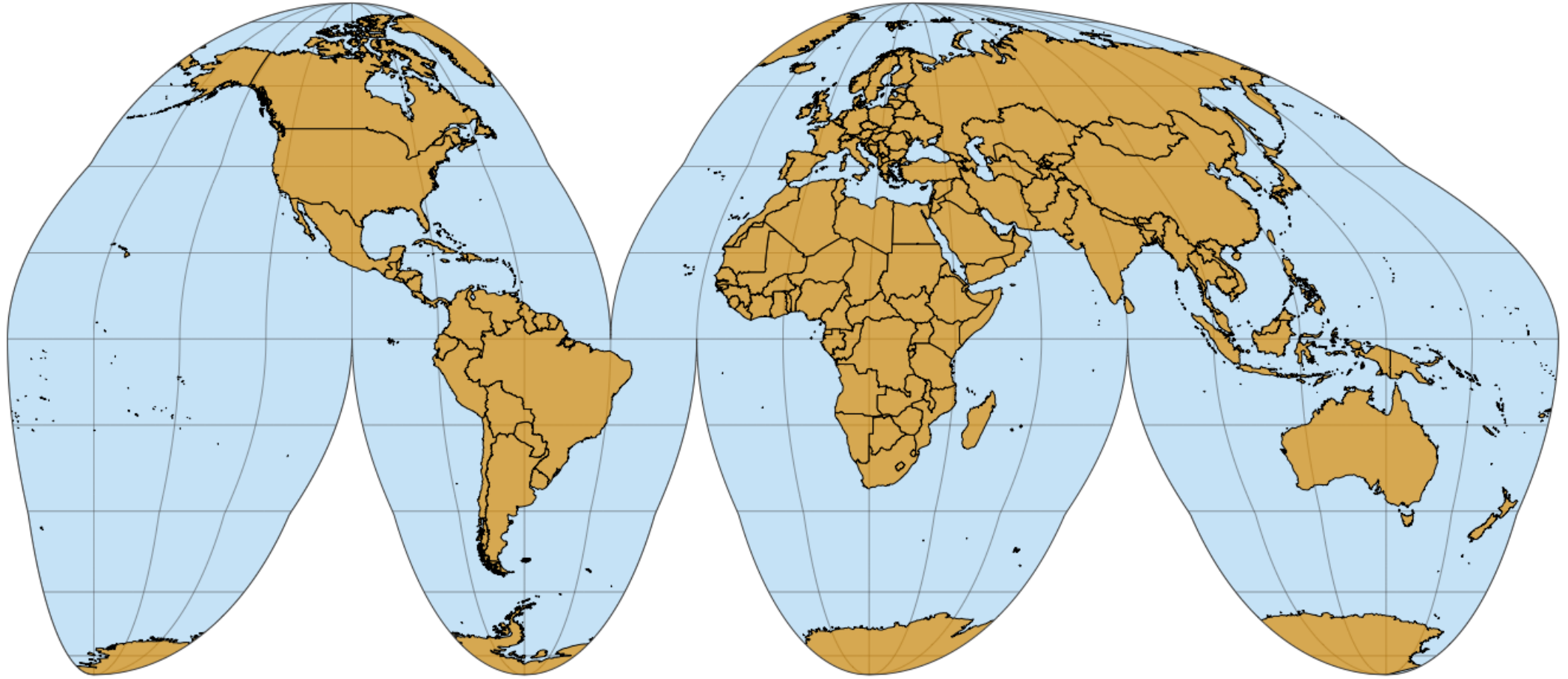
However, variants of this projection continue to live on.

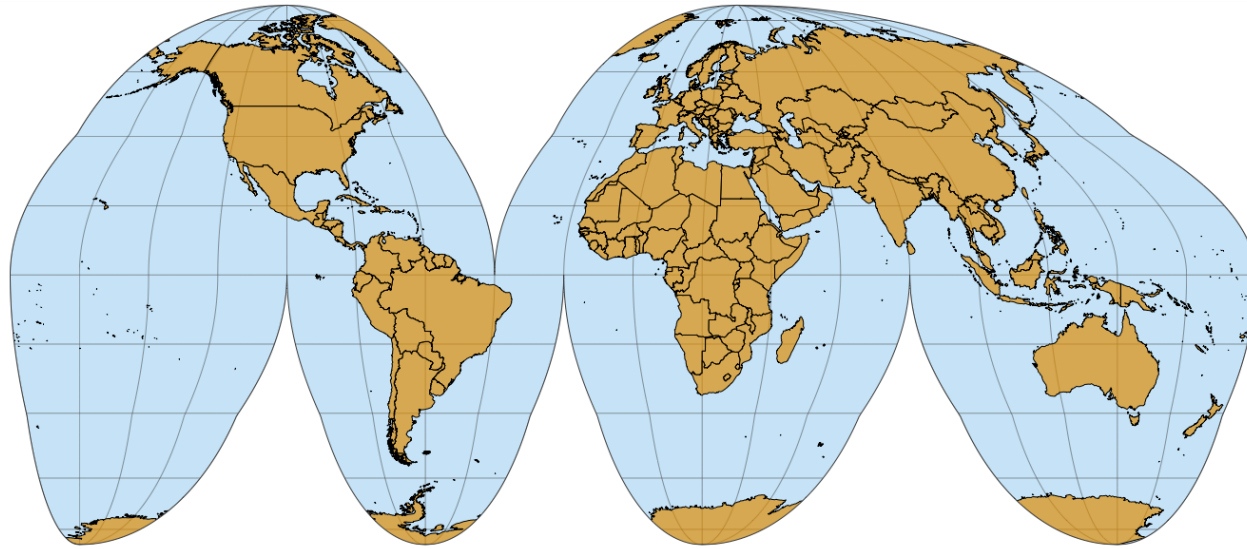
For example, the transverse Mercator projection is routinely used for large-scale maps that show moderately small areas (spanning less than a few degrees in longitude) at large magnification.

Another variant, the web Mercator projection, was introduced by Google for Google Maps and is used by several online mapping applications.

What about another whole-world projection that is, in contrast, perfectly area-preserving?







This projection accurately preserves areas while minimizing angular distortions, at the cost of showing oceans and some land masses (Greenland, Antarctica) in a non-contiguous way.

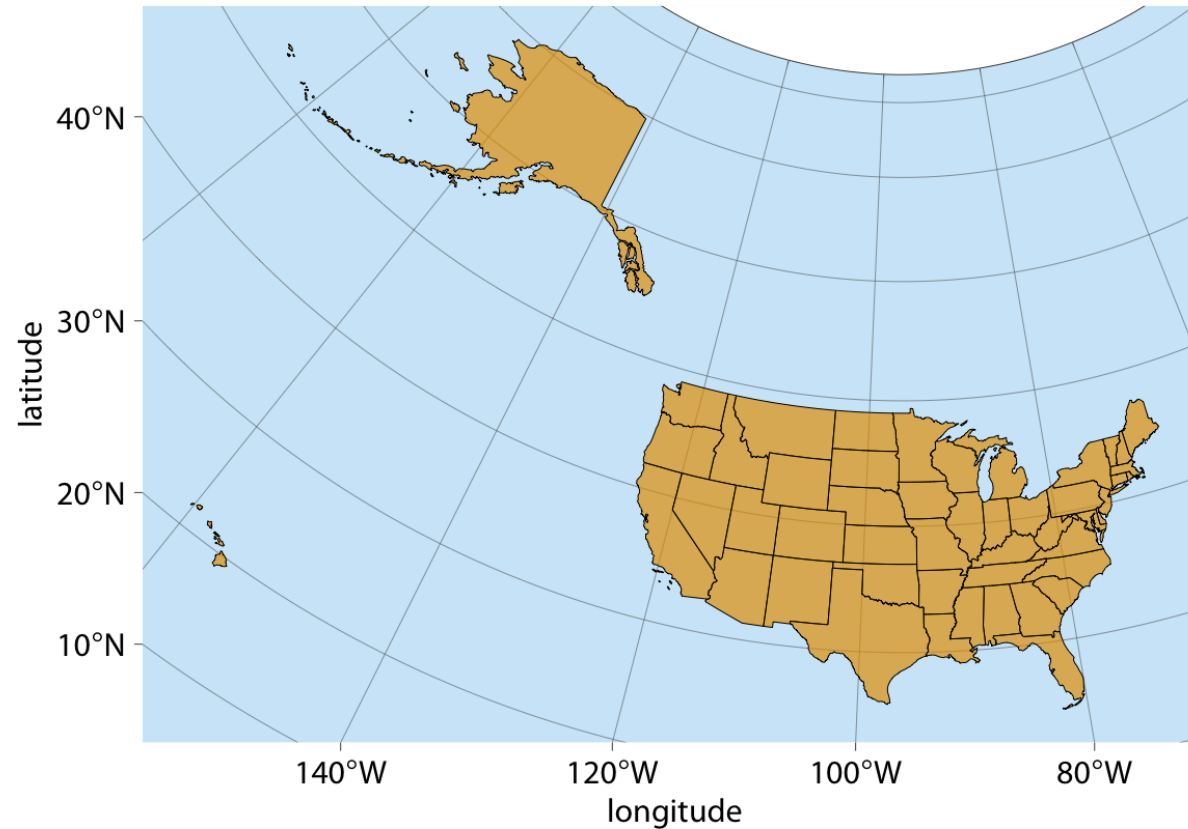
It has one cut in the northern hemisphere and three cuts in the southern hemisphere, carefully chosen so they don't interrupt major land masses. The cuts allow the projection to both preserve areas and approximately preserve angles, at the cost of non-contiguous oceans, a cut through the middle of Greenland, and several cuts through Antarctica. While the interrupted **Goode homolosine** has an unusual aesthetic and a strange name, it is a good choice for mapping applications that require accurate reproduction of areas on a global scale.

Projecting the US



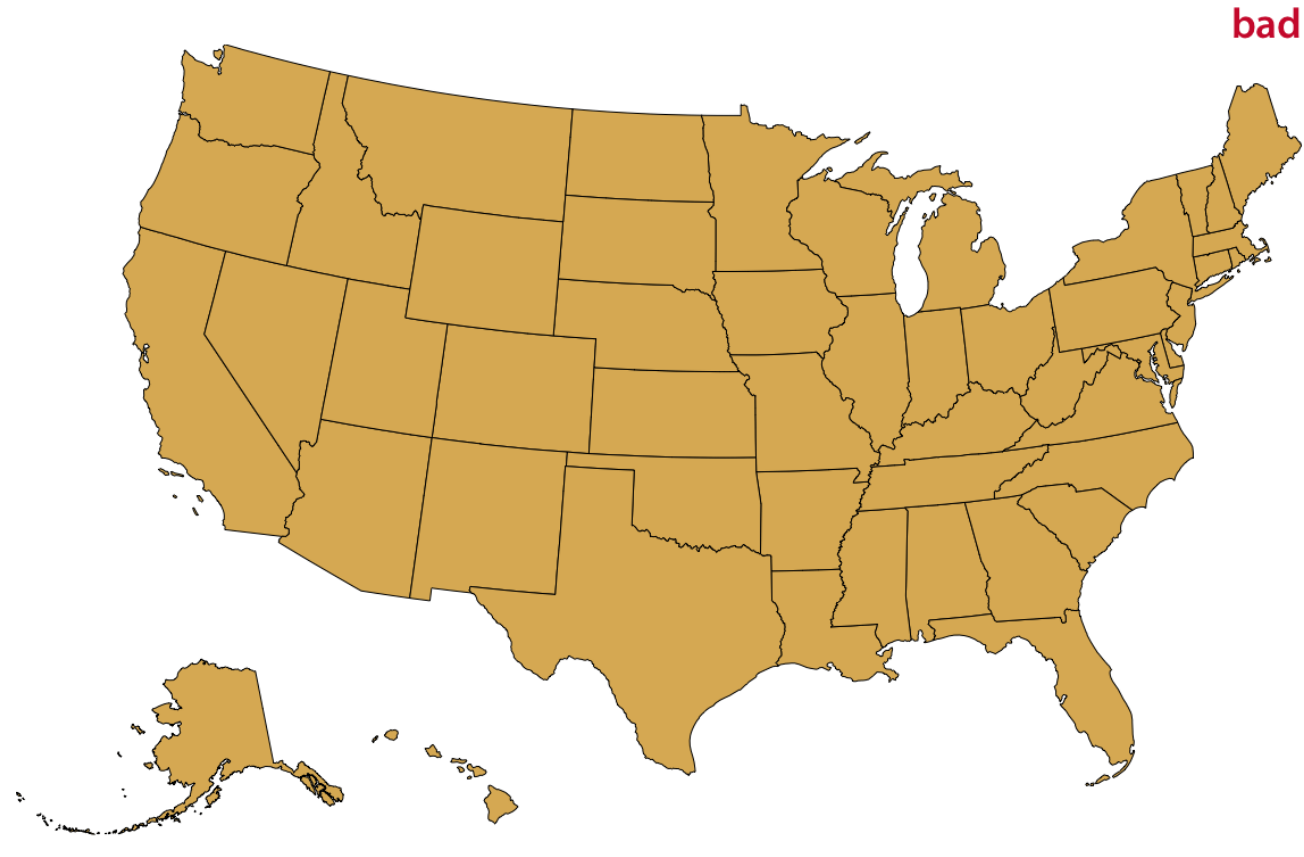
Alaska, Hawaii, and the lower 48 are far apart; difficult to show on one map

Projecting the US



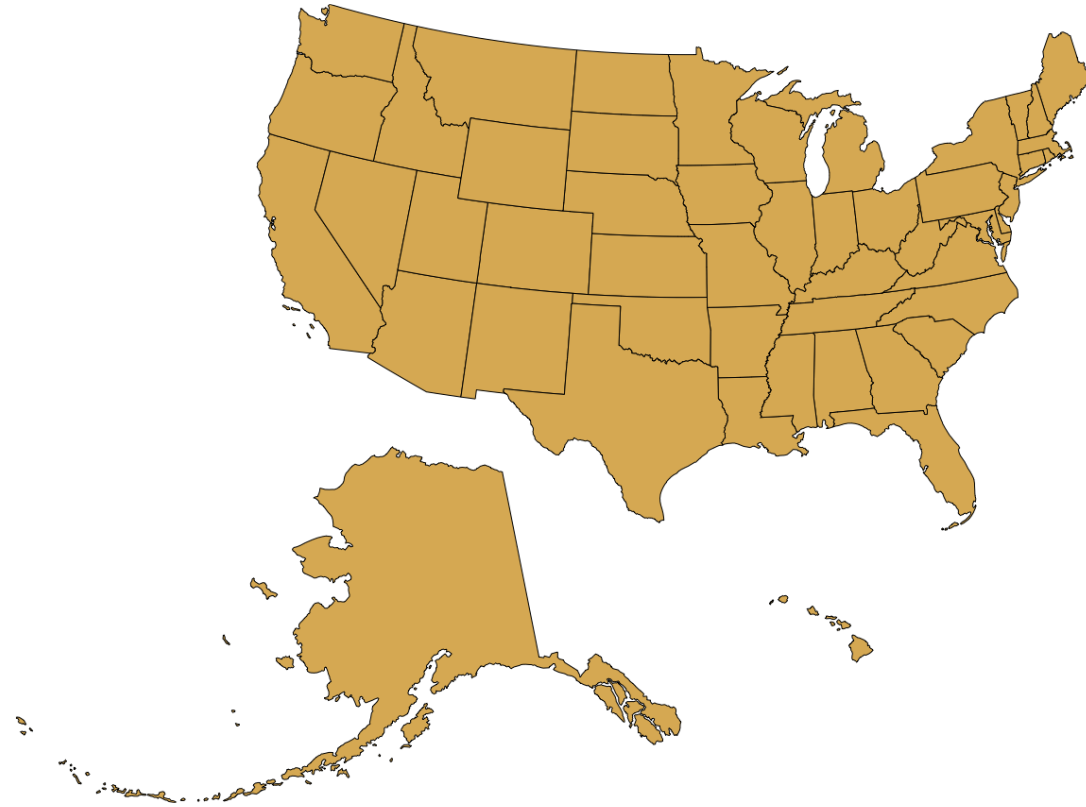
A fair, equal-area Albers projection, but Alaska is a bit stretched and there is so much ocean/empty space?

A common visualization. Alaska?



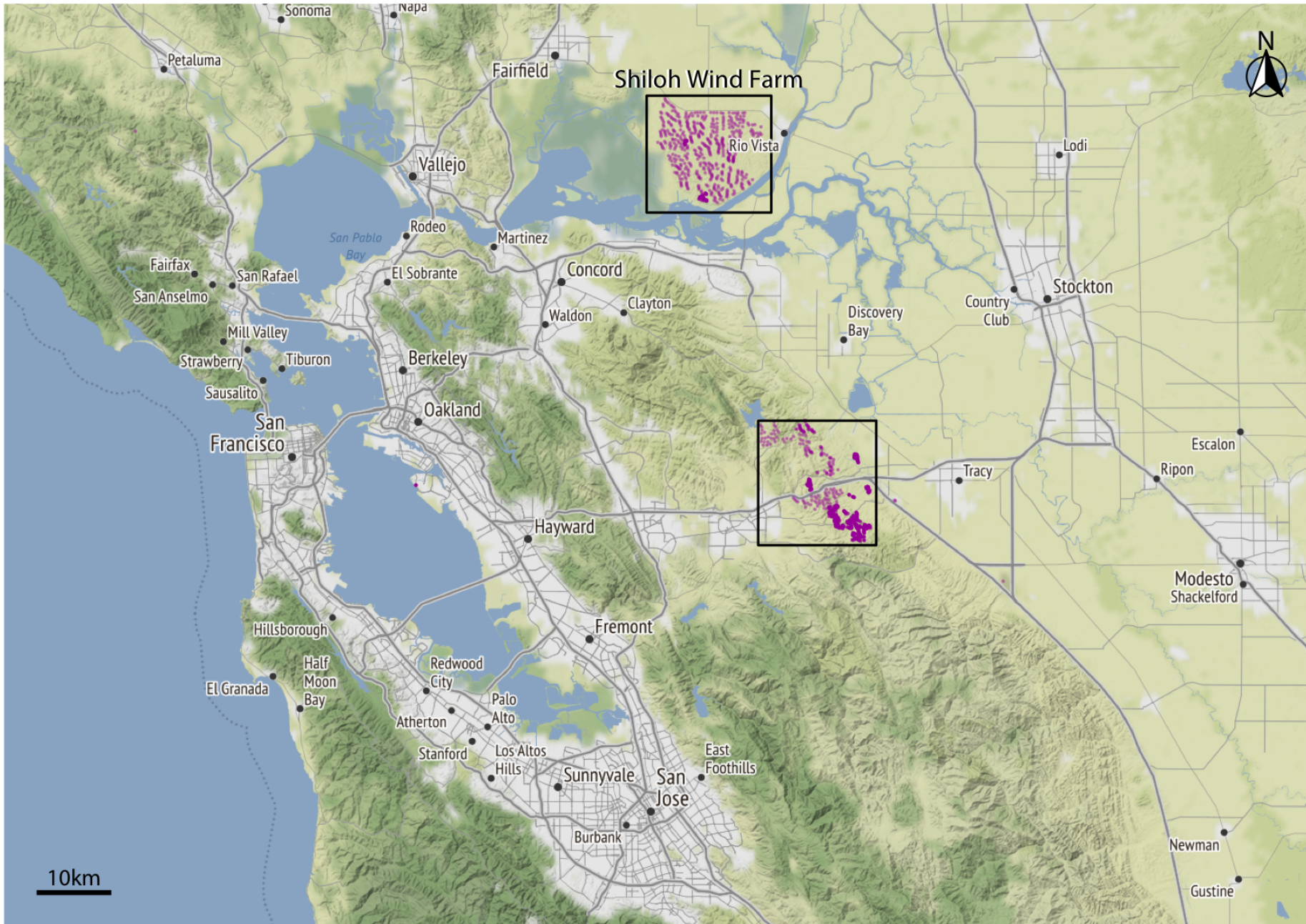
Project Alaska and Hawaii separately and move them closer; **but:**
Alaska was also reduced in size!

A fair visualization of the 50 states

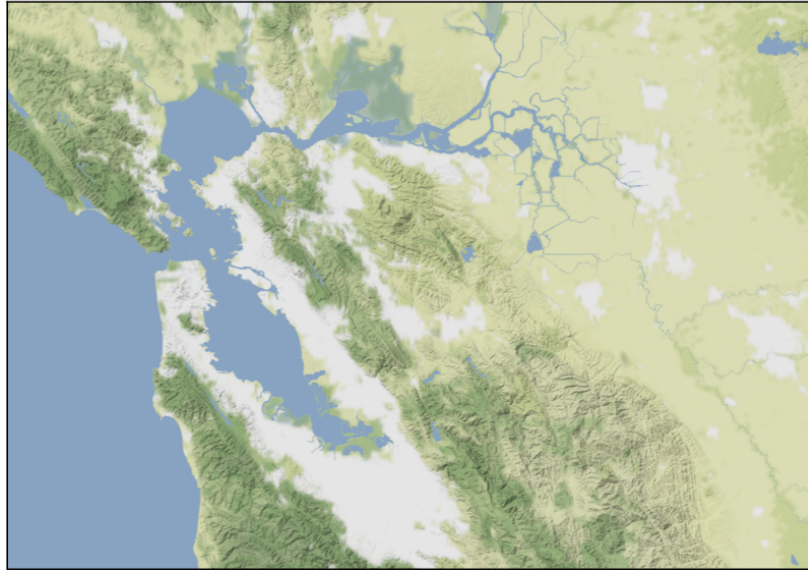


Alaska is actually the largest state; 2.2 times the size of Texas, just because of the scaling it looked similar in size to some other states

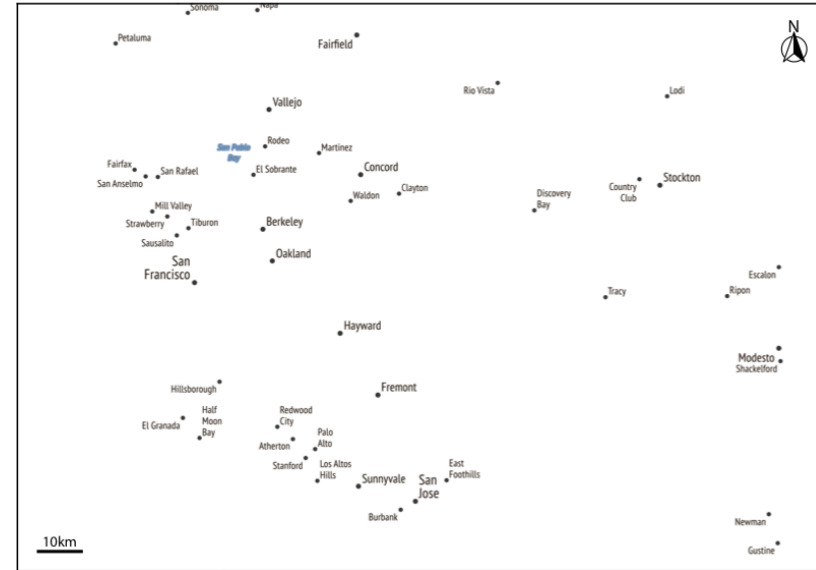
**We usually create maps
consisting of multiple layers
showing different types of
information**



terrain



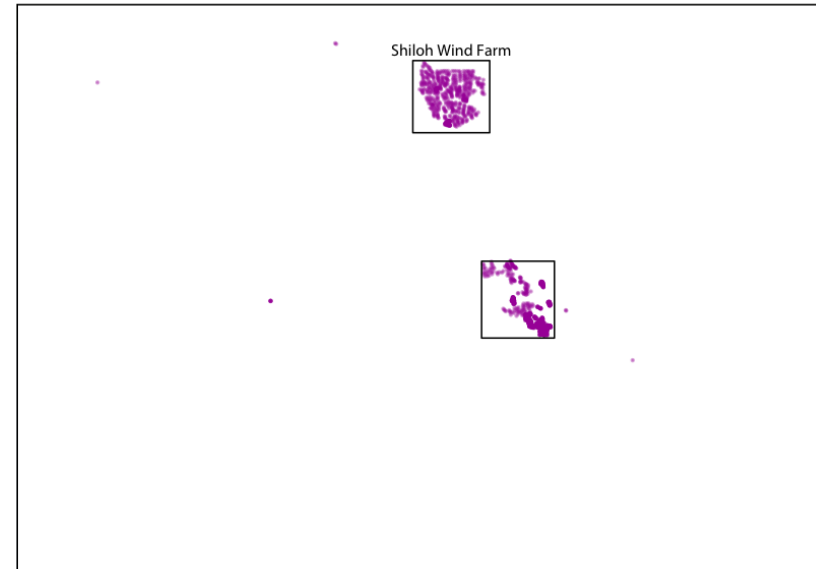
city labels, scale bar



roads



wind turbines



All the concepts discussed of mapping data onto aesthetics carry over to maps.

We can place data points into their geographic context and show other data dimensions via aesthetics such as color or shape.

Let's take a look at a typical workflow in R

```
1 library(forcats)
2 library(ggmap)
3 library(statebins)
4 library(sf)
5 library(lubridate)
6 library(geofacet)
7 library(dplyr)
8 library(cowplot)
9 library(colorspace)
```

Credits: <https://clauswilke.com/dataviz/geospatial-data.html>

```
1 # From http://www.csgnetwork.com/degreenllavcalc.html
2 # Length Of A Degree Of Longitude In Meters at 38deg lat
3 m_per_deg <- 87832.42967867786
4
5 shiloh_bbox <- c(left = -121.9, bottom = 38.06, right = -121.71, top = 38.20)
6 tracy_bbox <- c(left = -121.73, bottom = 37.66, right = -121.55, top = 37.81)
7
8 shiloh_scale = data.frame(
9   x = -121.735,
10  xend = -121.735 + 2000/m_per_deg,
11  y = 38.064,
12  yend = 38.064,
13  label = "2000m"
14 )
15
16 # download.file("https://github.com/clauswilke/dviz.supp/raw/master/data/wind_turbines.rda", "wi
17
18 load("wind_turbines.rda")
19
20 # download.file("https://github.com/clauswilke/dviz.supp/raw/master/data/sfbay_maps.rda", "sfbay
21
22 load("sfbay_maps.rda")
23
```

```
1 #' Locations of wind turbines in the U.S.
2 #'
3 #' Data were obtained from the United States Wind Turbine Database.
4 #'
5 #' @source
6 #' United States Wind Turbine Database (USWTDB)
7 #' \url{https://eerscmap.usgs.gov/uswtodb/data/}
8 #' @examples
9 #' library(sf)
10 #'
11 #' m <- cbind(wind_turbines$xlong, wind_turbines$ylat)
12 #' mproj <- rgdal::project(m, proj = "+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96
13 #' wind_turbines$x <- mproj[,1]
14 #' wind_turbines$y <- mproj[,2]
15 #'
16 #' lower48 <- us_states_geoms$lower48
17 #' ggplot(lower48) +
18 #'   geom_sf(fill = "#56B4E9", color = "grey30", size = 0.3, alpha = 0.5) +
19 #'   geom_point(
20 #'     data = filter(wind_turbines, xlong > -130, ylat > 20),
21 #'     aes(x = x, y = y),
22 #'     color = "black",
23 #'     size = 0.1
```

```
1 head(wind_turbines)
```

```
# A tibble: 6 × 24
```

```
  case_id faa_ors faa_asn usgs_pr_id t_state t_county t_fips p_name p_year
  <int> <chr> <chr> <int> <chr> <chr> <chr> <chr> <int>
1 3073429 missing missing 4960 CA Kern County 06029 251 Wind 1987
2 3071522 missing missing 4997 CA Kern County 06029 251 Wind 1987
3 3073425 missing missing 4957 CA Kern County 06029 251 Wind 1987
4 3071569 missing missing 5023 CA Kern County 06029 251 Wind 1987
5 3005252 missing missing 5768 CA Kern County 06029 251 Wind 1987
6 3003862 missing missing 5836 CA Kern County 06029 251 Wind 1987
```

```
# i 15 more variables: p_tnum <int>, p_cap <dbl>, t_manu <chr>, t_model <chr>,
# t_cap <int>, t_hh <dbl>, t_rd <dbl>, t_rsa <dbl>, t_ttlh <dbl>,
# t_conf_atr <int>, t_conf_loc <int>, t_img_date <chr>, t_img_srce <chr>,
# xlong <dbl>, ylat <dbl>
```

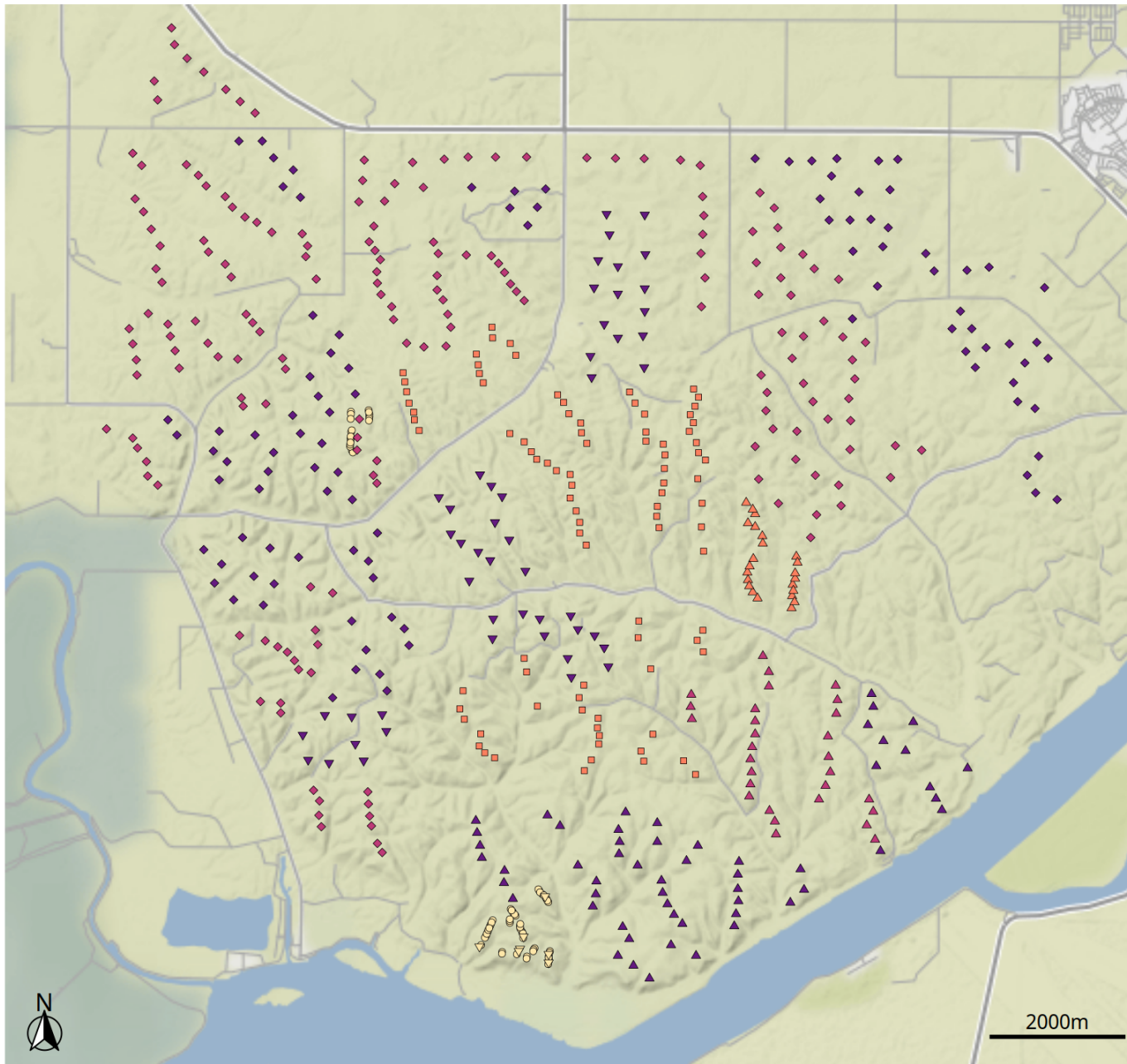
```
1 #' Maps of the bay area in California
2 #'
3 #' Map tiles obtained from maps.stamen.com and from openstreetmap.org.
4 #' @examples
5 #' library(ggmap)
6 #'
7 #' ggmap(sfbay_maps$sfbay_bg) + inset_ggmap(sfbay_maps$sfbay_lines)
8 #' ggmap(sfbay_maps$shiloh_osm)
9 "sfbay_maps"
```

```

1 wind_shiloh <- wind_turbines %>%
2   filter(
3     xlong < shiloh_bbox["right"],
4     xlong > shiloh_bbox["left"],
5     ylat > shiloh_bbox["bottom"],
6     ylat < shiloh_bbox["top"]
7   ) %>%
8   mutate(
9     name = fct_relevel(fct_collapse(p_name,
10      `EDF Renewables` = "EDF Renewable V",
11      `High Winds` = "High Winds",
12      `Shiloh` = c("Shiloh Wind Project", "Shiloh II", "Shiloh III", "Shiloh IV"),
13      `Solano` = c("Solano Phase 3", "Solano Phase IIA", "Solano Wind Project", "Solano Wind Pr
14      `other` = c("Montezuma", "Montezuma Winds II", "unknown Solano County")
15    ), "EDF Renewables", "High Winds", "Shiloh", "Solano", "other"),
16    year_range = cut(
17      p_year,
18      breaks = c(1980, 2000, 2005, 2010, 2015),
19      labels = c("before 2000", "2000 to 2004", "2005 to 2009", "2010 to 2014"),
20      right = FALSE
21    )
22  )

```

```
1 p2 <- ggmap(sf_bay_maps$shiloh_terrain) +
2   geom_point(
3     data = wind_shiloh,
4     aes(x = xlong, y = ylat, fill = year_range, shape = name),
5     size = 1.5,
6     color = "black", stroke = 0.2
7   ) +
8   geom_segment(
9     data = shiloh_scale,
10    aes(x, y, xend = xend, yend = yend),
11    size = 1
12  ) +
13  geom_text(
14    data = shiloh_scale,
15    aes(0.5*(x+xend), y, label = label),
16    hjust = 0.5,
17    vjust = -0.5,
18    family = dviz_font_family,
19    size = 10/.pt
20  ) +
21  ggspatial::annotation_north_arrow(
22    width = grid::unit(1, "cm"),
23    height = grid::unit(1, "cm"),
```



project name

- EDF Renewables
- High Winds
- ◆ Shiloh
- ▲ Solano
- ▼ other

year built

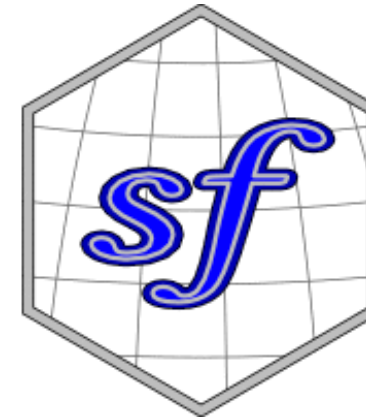
- before 2000
- 2000 to 2004
- 2005 to 2009
- 2010 to 2014



2000m

How to make geospatial visualizations in R?

The sf package: Simple Features in R



Picture credits to
Allison Horst

GeoJSON

GeoJSON is a plain text format for representing geographic data structures using the JavaScript Object Notation (JSON).

Geometry objects are used to represent the seven most common simple feature types.

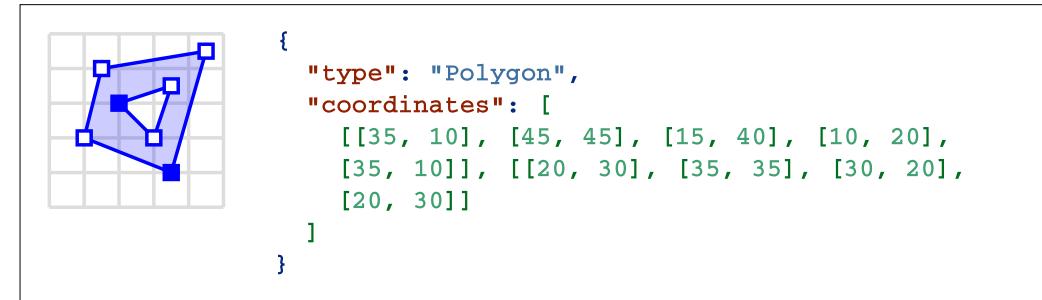
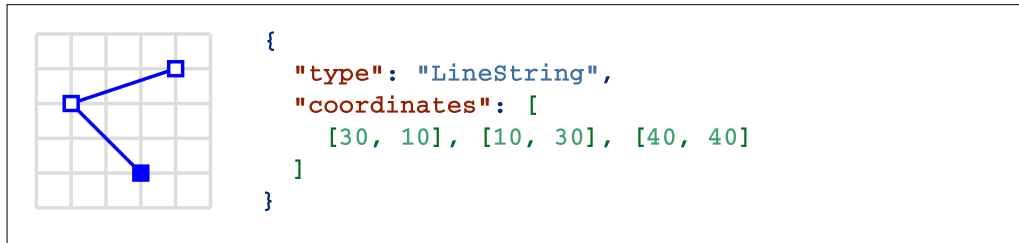
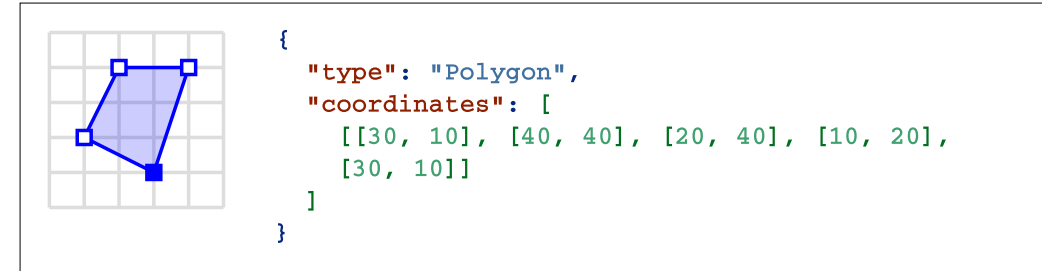
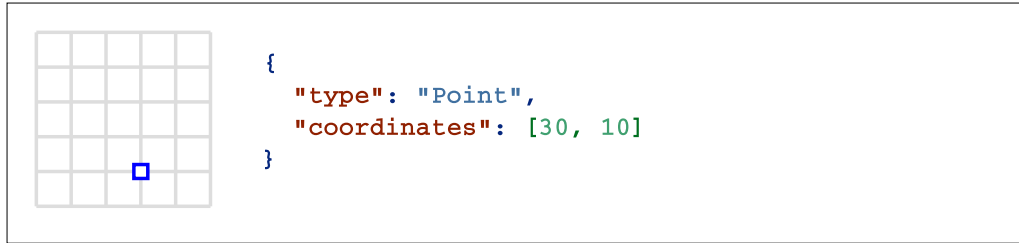
Feature objects are Geometry objects along with their non-spatial attributes.

FeatureCollection objects are collections of Feature objects.

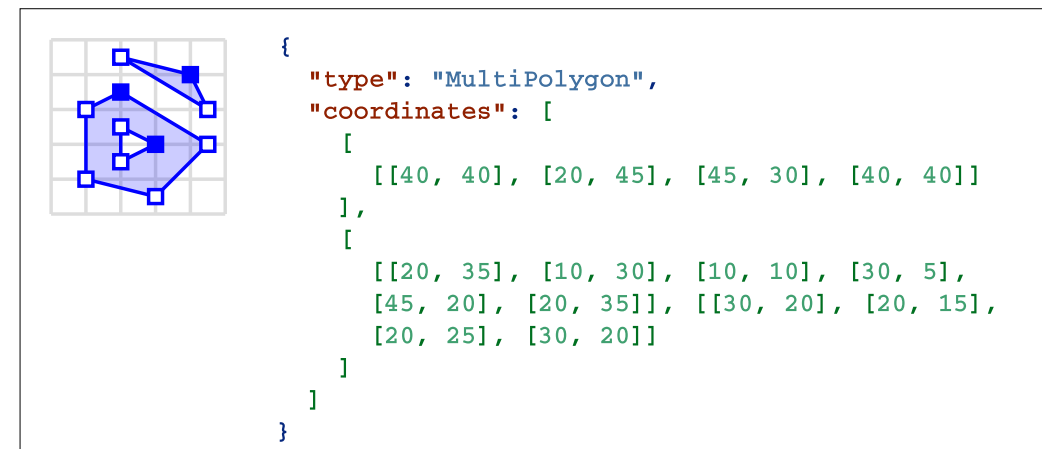
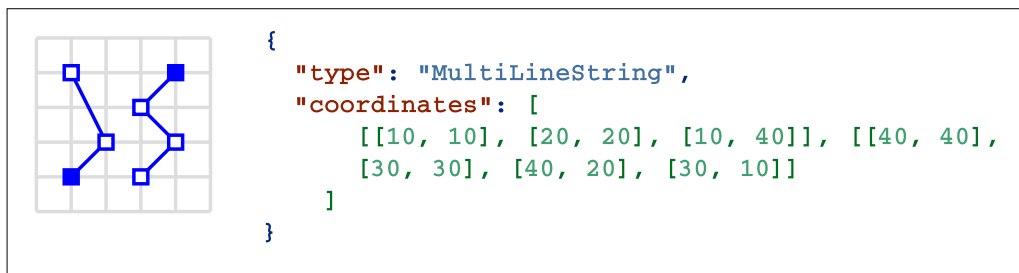
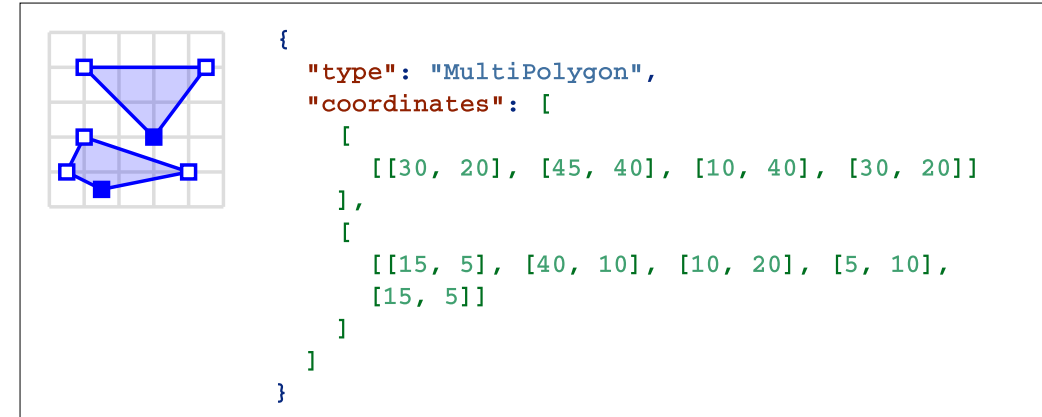
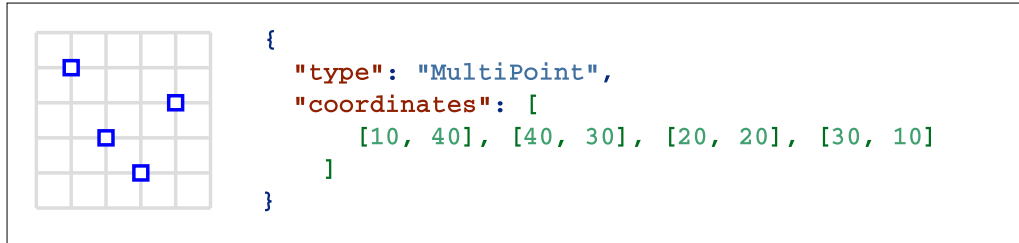
<https://geojson.org/>

<https://en.wikipedia.org/wiki/GeoJSON>

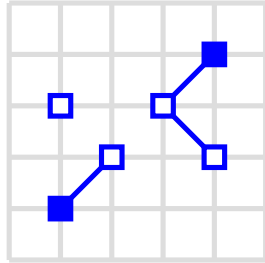
Geometry objects



Geometry objects

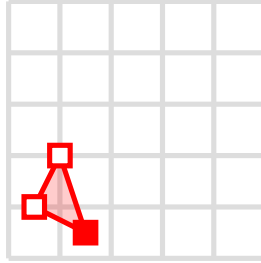


GeometryCollection objects



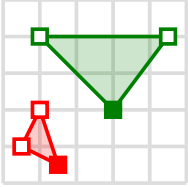
```
{  
  "type": "GeometryCollection",  
  "geometries": [  
    {  
      "type": "Point",  
      "coordinates": [10, 30]  
    },  
    {  
      "type": "MultiLineString",  
      "coordinates": [  
        [[10, 10], [20, 20]], [[40, 40],  
        [30, 30], [40, 20]]  
      ]  
    }  
  ]  
}
```

Feature objects



```
{  
  "type": "Feature",  
  "geometry": {  
    "type": "Polygon",  
    "coordinates": [  
      [[15, 5], [10, 20], [5, 10], [15, 5]]  
    ]  
  },  
  "properties": {  
    "color": "red",  
    "area": 763727.2  
  }  
}
```

FeatureCollection objects



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [[30, 20], [45, 40], [10, 40],
            [30, 20]]
        ]
      },
      "properties": {
        "color": "green",
        "area": 3565747.2
      }
    },
    {
      "type": "Feature",
      "geometry": {
        "type": "Polygon",
        "coordinates": [
          [[15, 5], [10, 20], [5, 10], [15, 5]]
        ]
      },
      "properties": {
        "color": "red",
        "area": 763727.2
      }
    }
  ]
}
```



```
1 # download.file("https://github.com/wilkelab/SDS375/raw/master/datasets/Texas_income.rds", "Texas_income.rds")
2 texas_income <- readRDS("Texas_income.rds")
3 texas_income
```

Simple feature collection with 254 features and 4 fields

Geometry type: MULTIPOLYGON

Dimension: XY

Bounding box: xmin: -106.6456 ymin: 25.83738 xmax: -93.50829 ymax: 36.5007

Geodetic CRS: NAD83

First 10 features:

	FIPS	county	median_income	moe	geometry
1	48001	Anderson	41327	1842	MULTIPOLYGON (((-96.0648 31...
2	48003	Andrews	70423	6038	MULTIPOLYGON (((-103.0647 3...
3	48005	Angelina	44223	1611	MULTIPOLYGON (((-95.00488 3...
4	48007	Aransas	41690	3678	MULTIPOLYGON (((-96.8229 28...
5	48009	Archer	60275	5182	MULTIPOLYGON (((-98.95382 3...
6	48011	Armstrong	59737	4968	MULTIPOLYGON (((-101.6294 3...
7	48013	Atascosa	52192	3005	MULTIPOLYGON (((-98.80479 2...
8	48015	Austin	53687	3810	MULTIPOLYGON (((-96.62085 3...
9	48017	Bailey	37397	8652	MULTIPOLYGON (((-103.0469 3...
10	48019	Bandera	49863	7193	MULTIPOLYGON (((-99.60332 2...

```
1 texas_income$geometry
```

Geometry set for 254 features

Geometry type: MULTIPOLYGON

Dimension: XY

Bounding box: xmin: -106.6456 ymin: 25.83738 xmax: -93.50829 ymax: 36.5007

Geodetic CRS: NAD83

First 5 geometries:

MULTIPOLYGON (((-96.0648 31.98066, -96.06305 31...

MULTIPOLYGON (((-103.0647 32.52219, -103.0005 3...

MULTIPOLYGON (((-95.00488 31.42396, -95.00334 3...

MULTIPOLYGON (((-96.8229 28.16743, -96.82127 28...

MULTIPOLYGON (((-98.95382 33.49637, -98.95377 3...

```
1 texas_income %>%
2   filter(county == "Travis")
```

Simple feature collection with 1 feature and 4 fields

Geometry type: MULTIPOLYGON

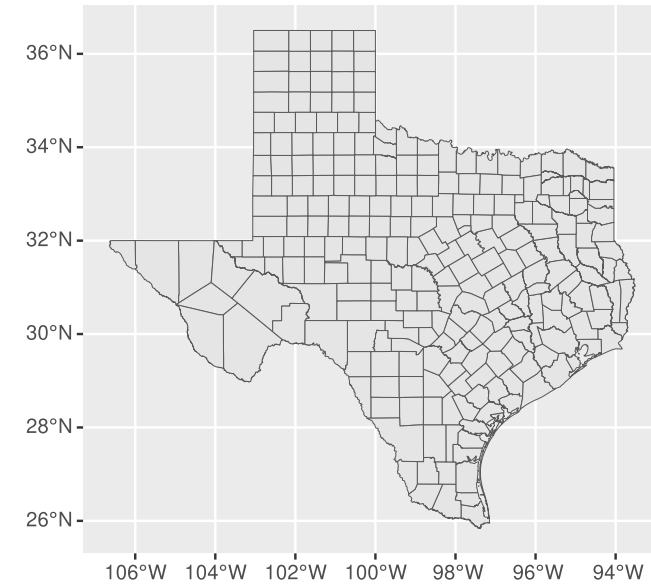
Dimension: XY

Bounding box: xmin: -98.17298 ymin: 30.0245 xmax: -97.36954 ymax: 30.62825

Geodetic CRS: NAD83

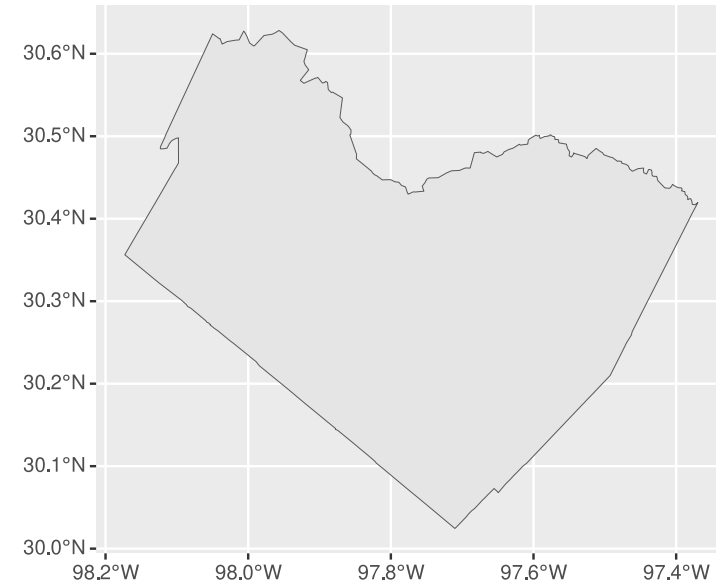
```
   FIPS county median_income moe geometry
1 48453 Travis      61451 591 MULTIPOLYGON (((-98.15927 3...
```

```
1 # plot all of Texas
2 ggplot(texas_income) +
3   geom_sf()
```



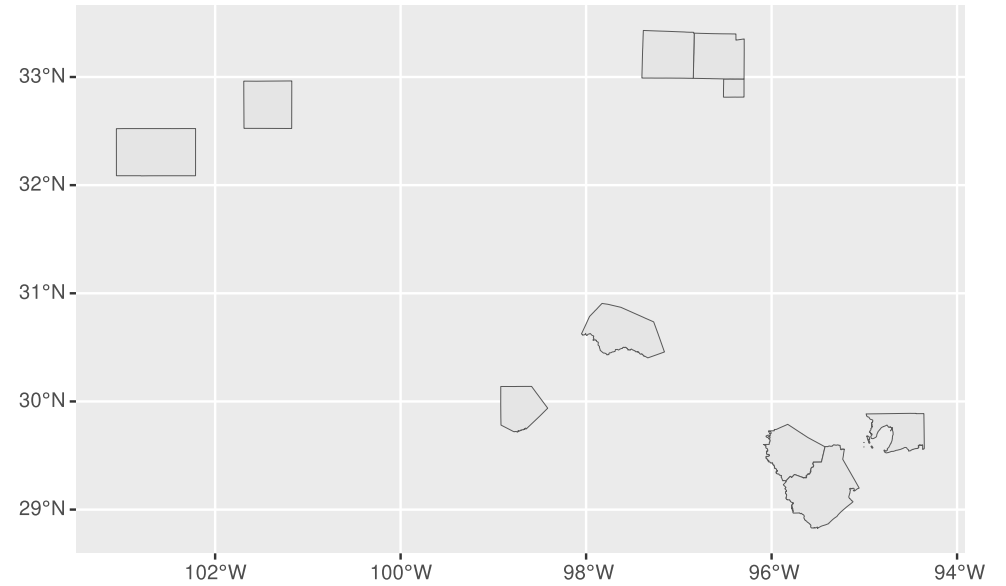
ggplot supports simple features with `geom_sf()`

```
1 # plot only Travis County
2 texas_income %>%
3   filter(county == "Travis") %>%
4   ggplot() +
5   geom_sf()
```

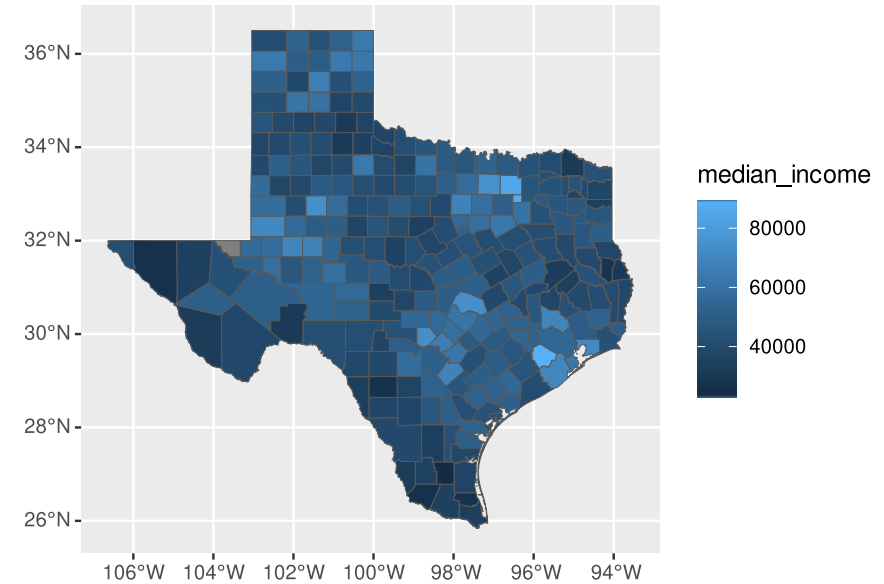


We can do any kind of data operation, the same way as with all the other types of data that we encountered in the course so far

```
1 # plot the ten richest counties
2 texas_income %>%
3   slice_max(median_income, n = 10) %>%
4   ggplot() +
5   geom_sf()
```



```
1 # color counties by median income
2 texas_income %>%
3   ggplot(aes(fill = median_income)) +
4   geom_sf()
```

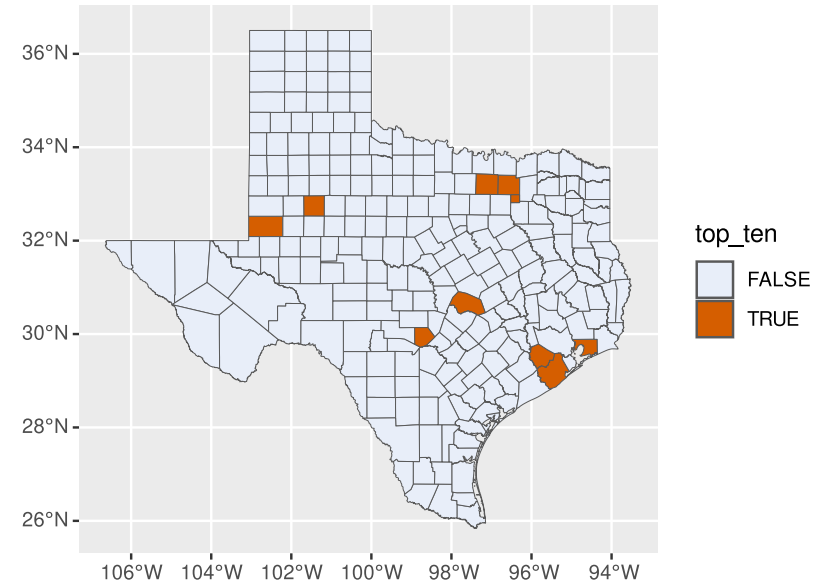


First example of **choropleth** mapping: Coloring areas by data values

```

1 # highlight the ten richest counties
2 texas_income %>%
3   mutate(
4     top_ten =
5       rank(desc(median_income)) <= 10
6   ) %>%
7   ggplot(aes(fill = top_ten)) +
8   geom_sf() +
9   scale_fill_manual(
10    values = c(
11      `TRUE` = "#D55E00",
12      `FALSE` = "#E8EEF9"
13    )
14  )

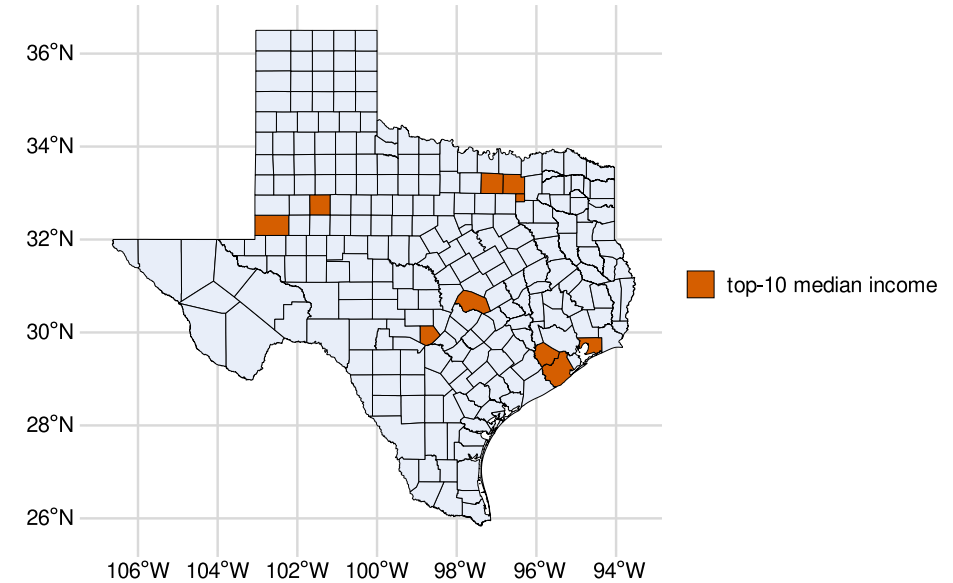
```




```

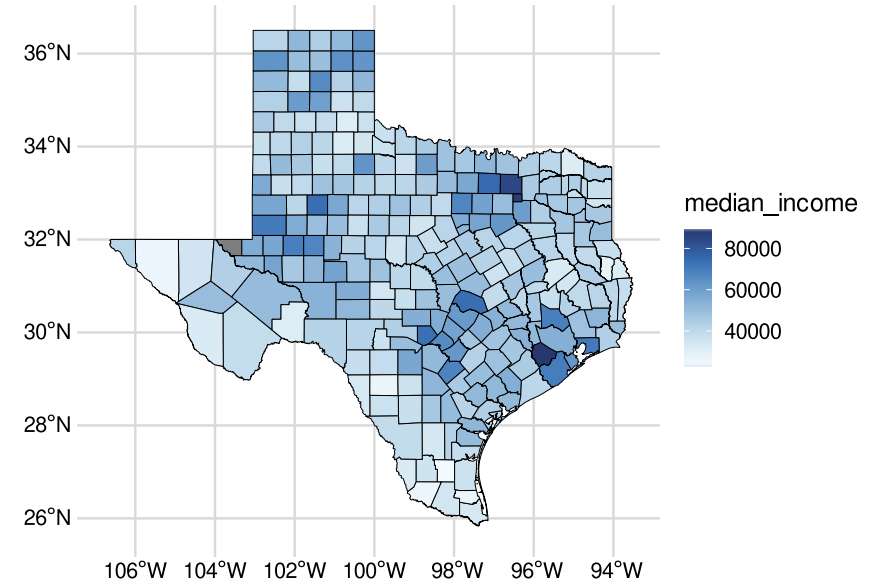
1 # highlight the ten richest counties
2 texas_income %>%
3   mutate(
4     top_ten =
5       rank(desc(median_income)) <= 10
6     ) %>%
7   ggplot(aes(fill = top_ten)) +
8   geom_sf(color = "black",
9           linewidth = 0.1) +
10  scale_fill_manual(
11    name = NULL,
12    values = c(
13      `TRUE` = "#D55E00",
14      `FALSE` = "#E8EEF9"
15    ),
16    breaks = c(TRUE),
17    labels = "top-10 median income"
18  ) +
19  cowplot::theme_minimal_grid(11)

```



We apply styling as usual

```
1 ggplot(texas_income) +  
2   geom_sf(  
3     aes(fill = median_income),  
4     color = "black", linewidth = 0.1  
5   ) +  
6   scale_fill_continuous_sequential(  
7     palette = "Blues", rev = TRUE  
8   ) +  
9   coord_sf() +  
10  theme_minimal_grid(11)
```

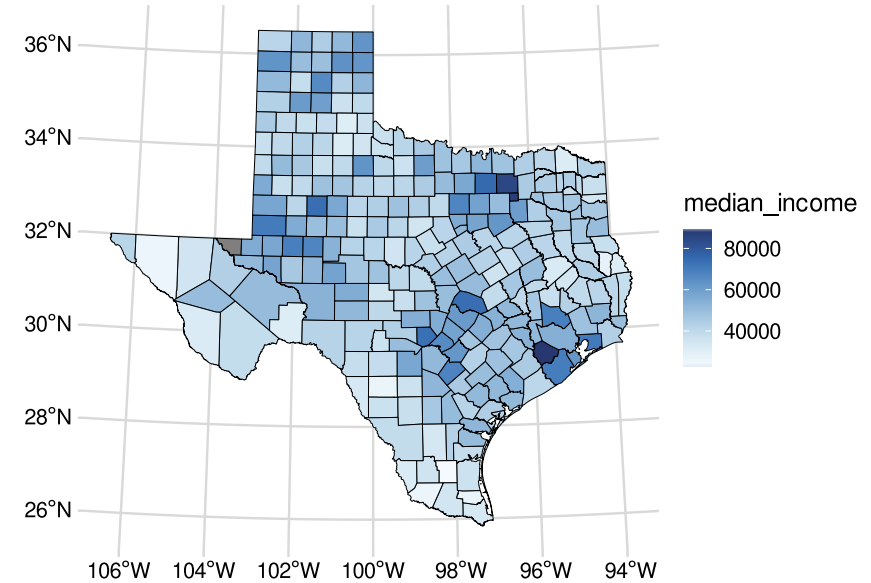


We can customize the projection with `coord_sf()`

```

1 ggplot(texas_income) +
2   geom_sf(
3     aes(fill = median_income),
4     color = "black", linewidth = 0.1
5   ) +
6   scale_fill_continuous_sequential(
7     palette = "Blues", rev = TRUE
8   ) +
9   coord_sf(
10    # Texas Centric Albers Equal Area
11    crs = 3083
12  ) +
13  theme_minimal_grid(11)

```

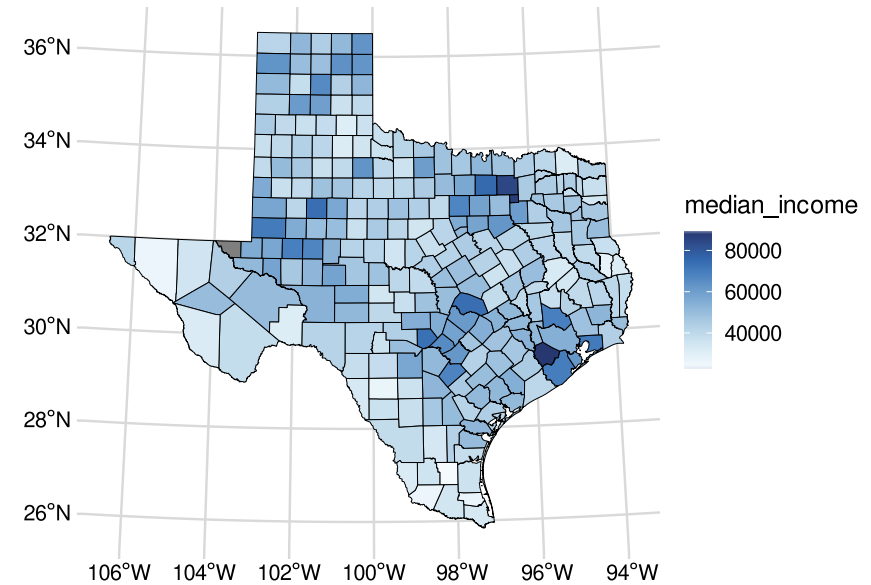


Reference: <https://spatialreference.org/ref/epsg/3083/>

```

1 ggplot(texas_income) +
2   geom_sf(
3     aes(fill = median_income),
4     color = "black", linewidth = 0.1
5   ) +
6   scale_fill_continuous_sequential(
7     palette = "Blues", rev = TRUE
8   ) +
9   coord_sf(
10    # Texas Centric
11    # Lambert Conformal Conic
12    crs = 32139
13  ) +
14  theme_minimal_grid(11)

```

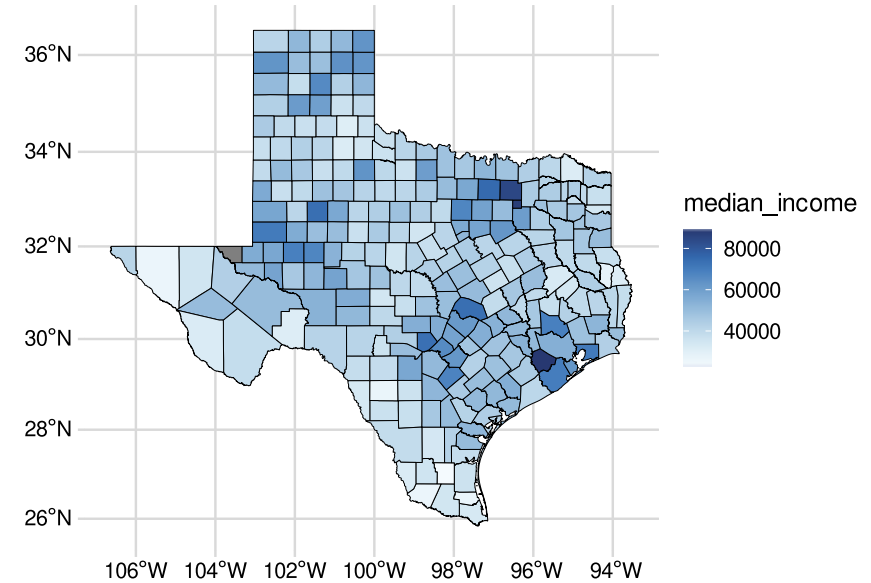


Reference: <https://spatialreference.org/ref/epsg/32139/>

```

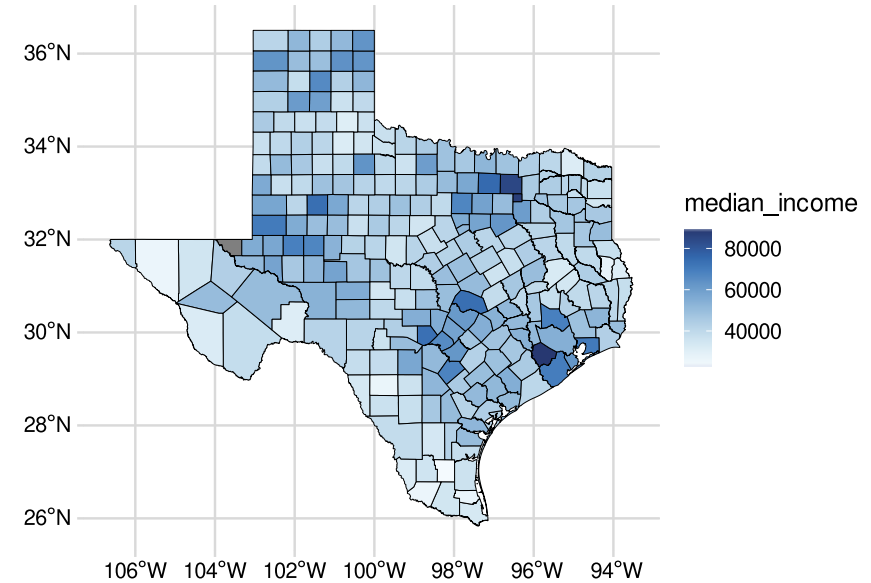
1 ggplot(texas_income) +
2   geom_sf(
3     aes(fill = median_income),
4     color = "black", linewidth = 0.1
5   ) +
6   scale_fill_continuous_sequential(
7     palette = "Blues", rev = TRUE
8   ) +
9   coord_sf(
10    # Web Mercator (Google Maps)
11    crs = 3857
12  ) +
13  theme_minimal_grid(11)

```



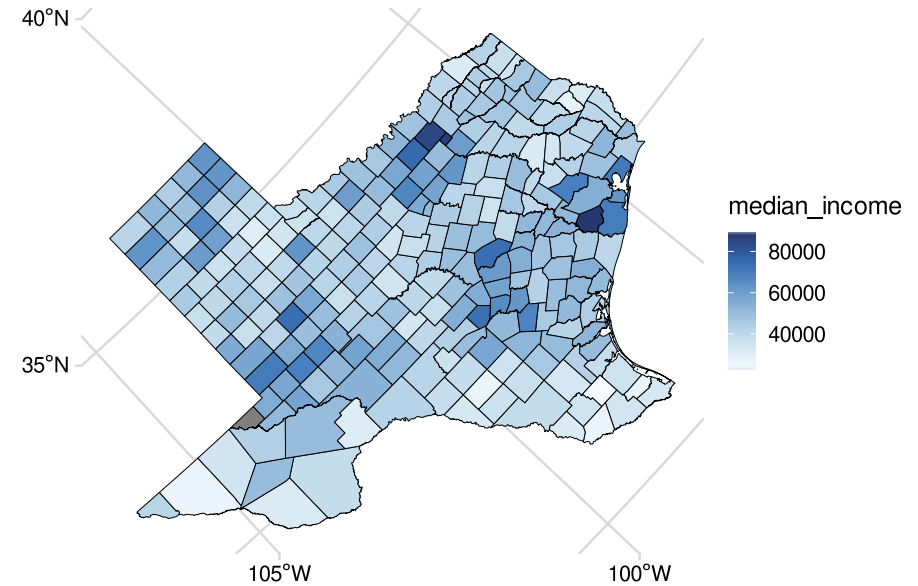
Reference: <https://spatialreference.org/ref/epsg/3857/>

```
1 ggplot(texas_income) +
2   geom_sf(
3     aes(fill = median_income),
4     color = "black", linewidth = 0.1
5   ) +
6   scale_fill_continuous_sequential(
7     palette = "Blues", rev = TRUE
8   ) +
9   coord_sf(
10    # Longitude-Latitude WGS84 (GPS)
11    crs = 4326
12  ) +
13  theme_minimal_grid(11)
```



Reference: <https://spatialreference.org/ref/epsg/4326/>

```
1 ggplot(texas_income) +
2   geom_sf(
3     aes(fill = median_income),
4     color = "black", linewidth = 0.1
5   ) +
6   scale_fill_continuous_sequential(
7     palette = "Blues", rev = TRUE
8   ) +
9   coord_sf(
10    # Alaska Albers equal area
11    crs = 3338
12  ) +
13  theme_minimal_grid(11)
```



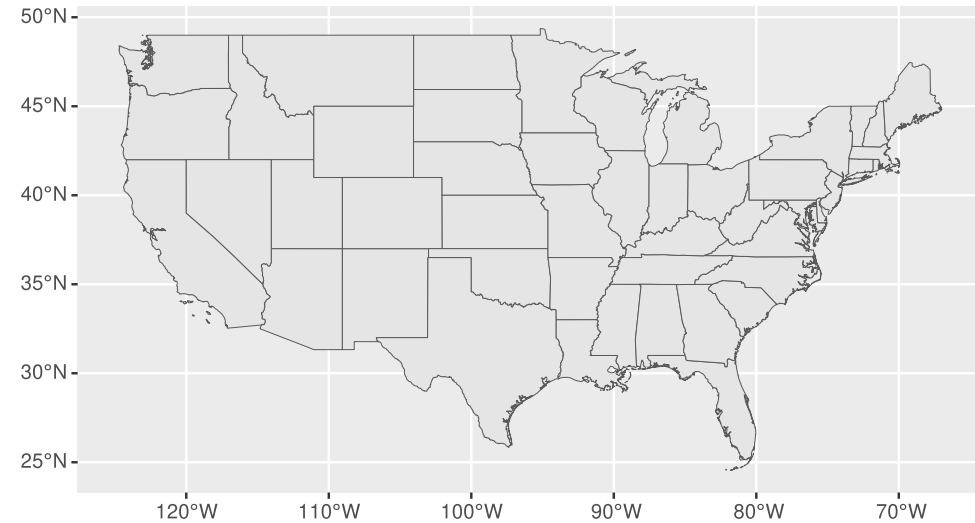
Reference: <https://spatialreference.org/ref/epsg/3338/>

```
1 library(rnaturalearth)
2
3 sf_world <-
4   ne_countries(returnclass='sf')
5
6 ggplot(sf_world) + geom_sf()
```



We can get map data from the `rnaturalearth` package


```
1 sf_us <- ne_states(  
2   country="United States of America",  
3   returnclass='sf'  
4 )  
5  
6 sf_us %>%  
7   # exclude Alaska (US02),  
8   # Hawaii (US15)  
9   filter(!code_local %in%  
10         c("US02", "US15")) %>%  
11   ggplot() + geom_sf()
```



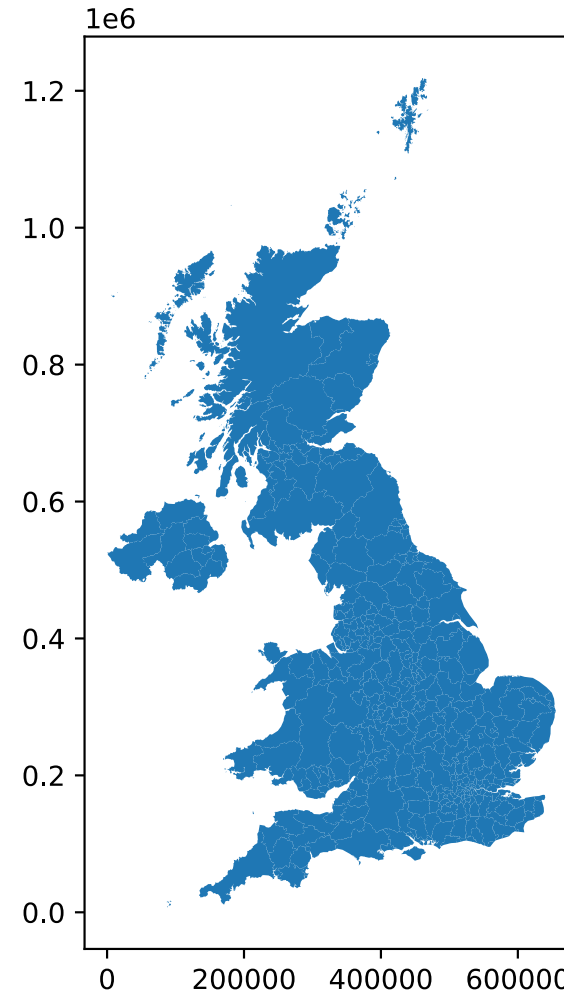
Another example: a map of the lower 48 states of the US (without Alaska and Hawaii)

Geospatial data visualization workflows in Python

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 import os
4 import numpy as np
5 import geopandas as gpd
6 from pathlib import Path
7 import warnings
```

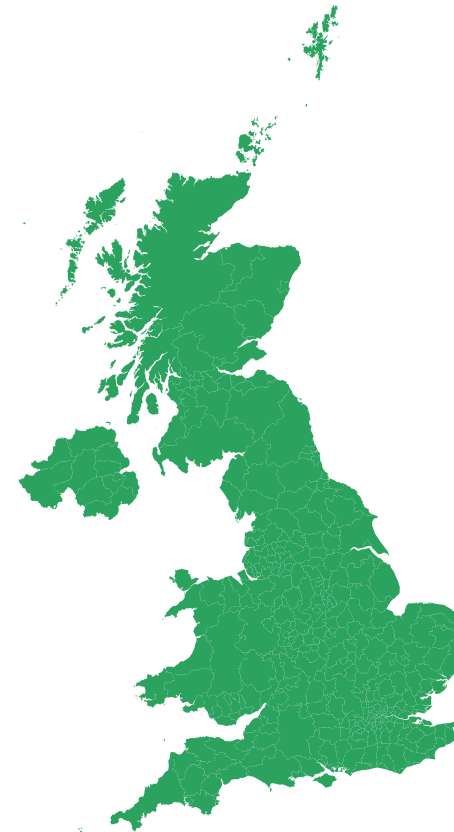
Credits: <https://aeturrell.github.io/coding-for-economists/geo-vis.html>

```
1 # To run this example, you will need to dow
2 # wget https://github.com/aeturrell/coding-
3 # and save them to your project folder
4 df = gpd.read_file(
5     "Local_Authority_Districts__May_2020__U
6 )
7
8 df.plot(figsize=(4,6))
```



Load a shapefile and use the `.plot` method in geopandas

```
1 ax = df.plot(color="#2ca25f",  
2 figsize=(4,6))  
3 _ = ax.axis("off")  
4 plt.show()
```



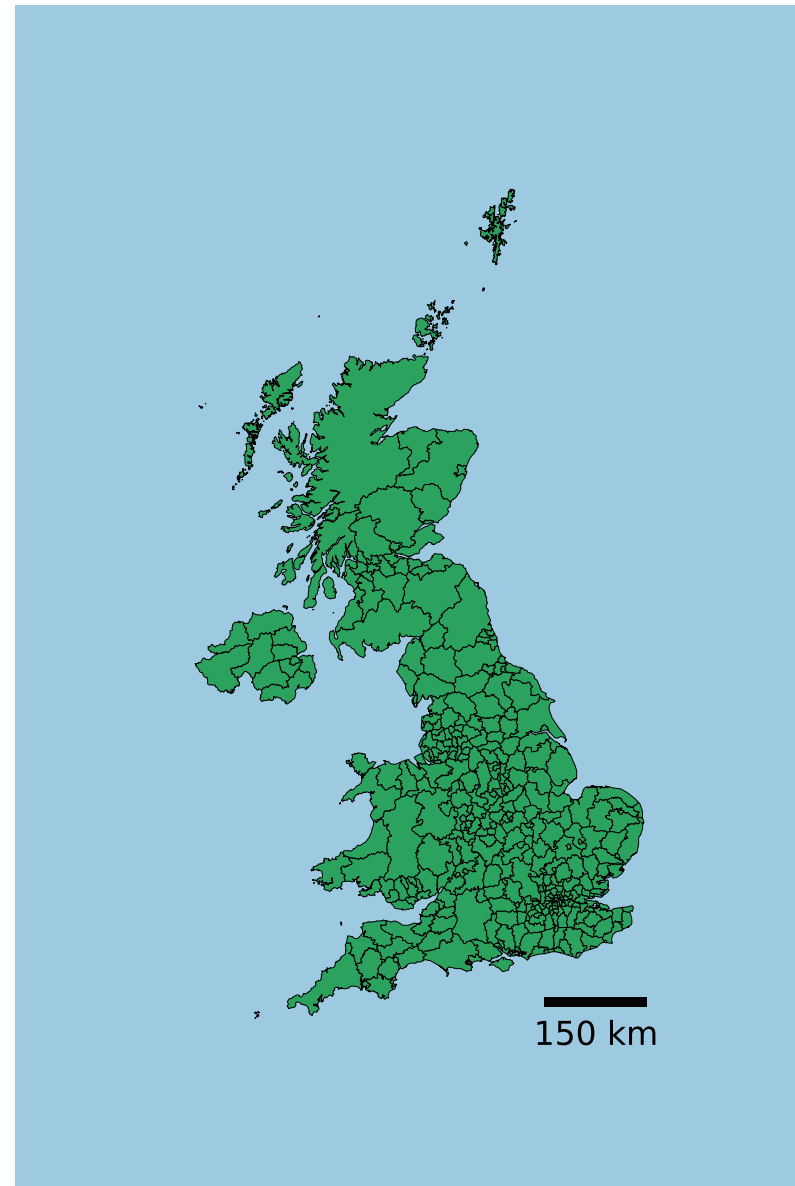
Options for customization: `plot` returns an `axis` object

```

1 from matplotlib_scalebar.scalebar import ScaleBar
2
3 fig, ax = plt.subplots()
4 fig.set_size_inches(4,6)
5 df.plot(color="#2ca25f",
6         edgecolor="k", linewidth=0.2,
7         facecolor="blue", ax=ax)
8 _ = ax.axis("off")
9 fig.patch.set_facecolor("#9ecae1")
10 # Create scale bar
11 scalebar = ScaleBar(
12     1,
13     box_alpha=0,
14     location="lower right",
15     length_fraction=0.25,
16     font_properties={"size": 12},
17 )
18 _ = ax.add_artist(scalebar)

```

Add a scale and format division lines
of the different districts



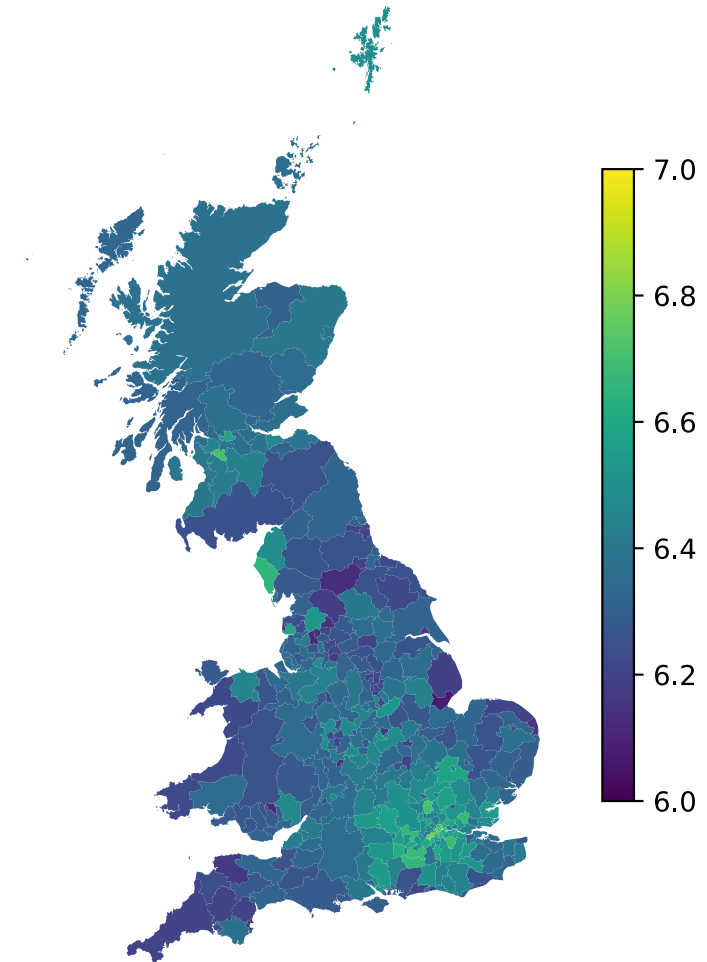
Choropleth mapping: Coloring areas by data values

```

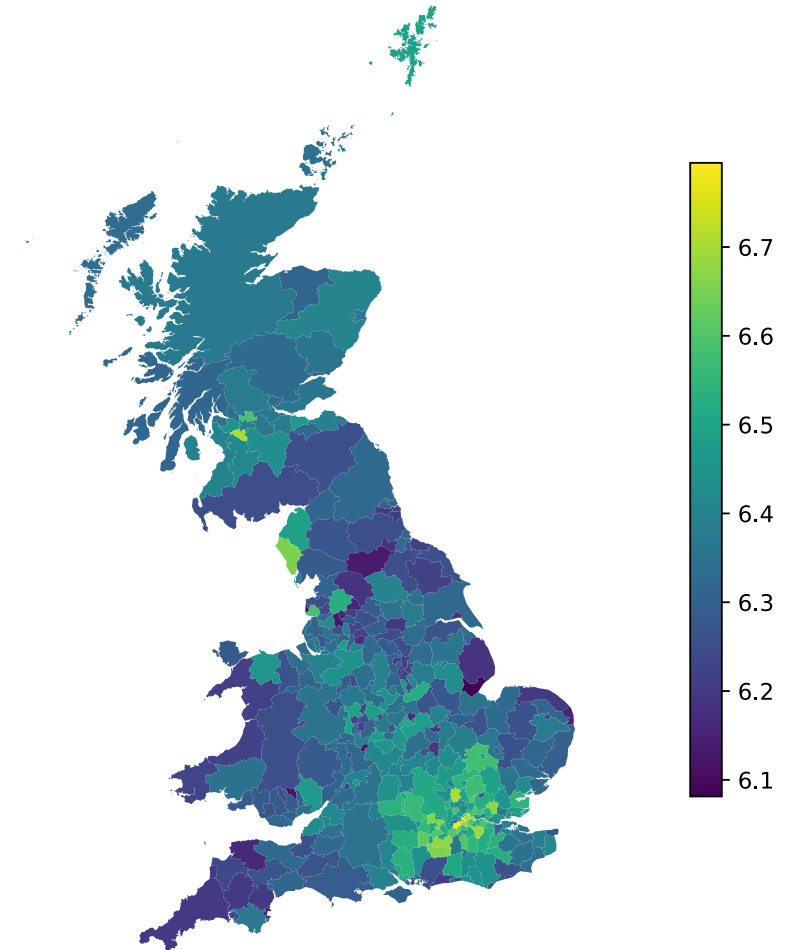
1 pay = pd.read_csv(
2     "https://github.com/aeturrell/coding-fo
3 )
4 pay = pay.rename(columns={"lad": "LAD20CD"})
5 df = df.merge(pay, on=["LAD20CD"], how="inn
6 col = "Log median weekly pay (2020 GBP)"
7 df[col] = np.log(df["Median weekly pay (202
8
9 fig, ax = plt.subplots()
10 fig.set_size_inches(4,6)
11 ax.set_title(col, loc="left")
12 df.plot(
13     ax=ax,
14     column=col,
15     legend=True,
16     legend_kwds={"label": "", "shrink": 0.6
17     vmin=round(df[col].min()),
18     vmax=round(df[col].max()),
19 )
20 _ = ax.axis("off")
21 plt.tight_layout()
22 plt.show()

```

Log median weekly pay (2020 GBP)



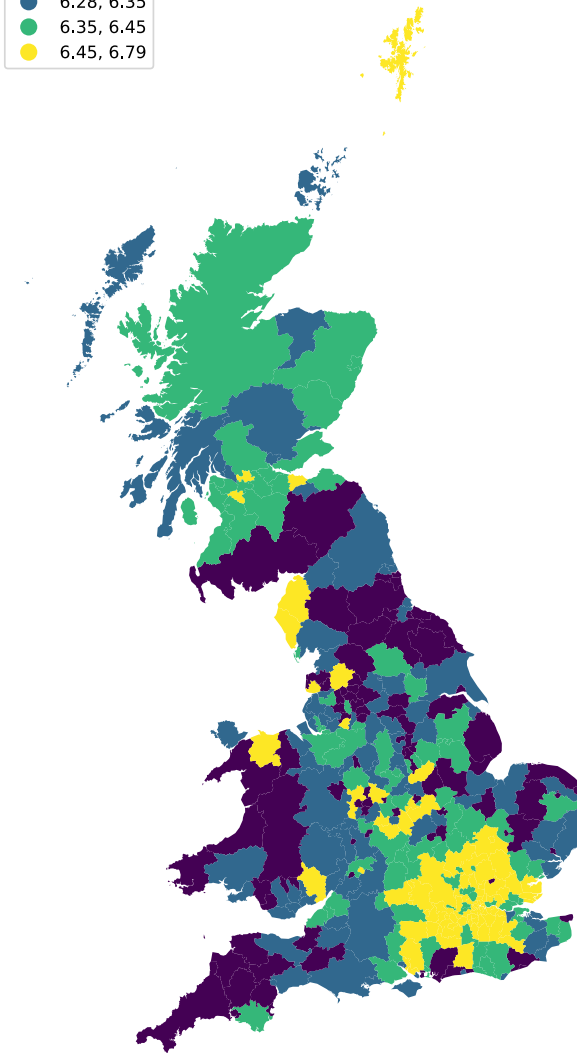
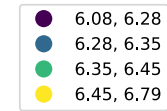
```
1 import geoplot as gplt
2 import geoplot.crs as gcrs
3
4 fig, ax = plt.subplots(
5     figsize=(6,8),
6     subplot_kw={'projection': gcrs.AlbersEqual_Area}
7 )
8
9 gplt.choropleth(
10     df.to_crs("EPSG:4326"),
11     ax=ax,
12     hue=col,
13     cmap="viridis",
14     legend=True,
15     legend_kwargs={'shrink': 0.5}
16 )
17
18 plt.tight_layout()
19 plt.show()
```



We can do the same with geoplot


```
1 fig, ax = plt.subplots(
2     figsize=(8, 10))
3 ax.set_title(col, loc="left")
4 ax.axis("off")
5 df.plot(
6     ax=ax, column=col, legend=True,
7     scheme="Quantiles", k=4,
8     legend_kwds={"loc": 2})
9
10 plt.tight_layout()
11 plt.show()
```

Log median weekly pay (2020 GBP)



We can split the variable
into a distinct number of ranges

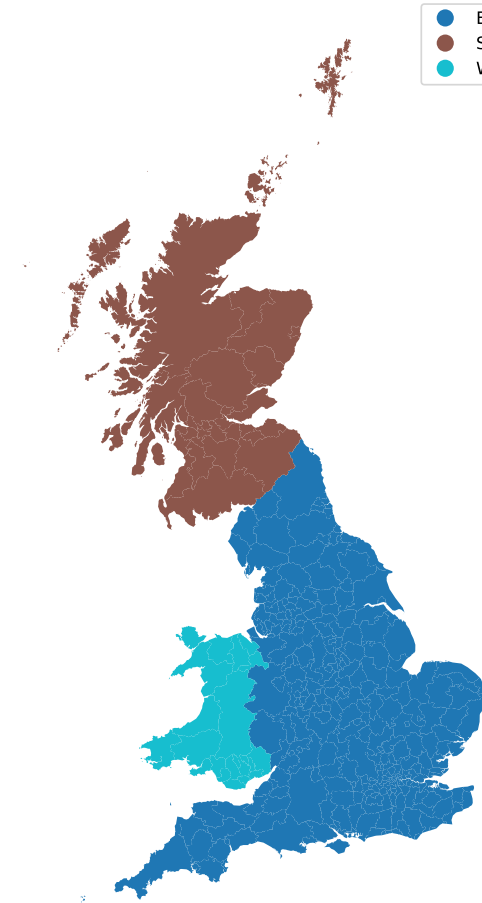
	OBJECTID	LAD20CD	LAD20NM	LAD20NMW	BNG_E	BNG_N	\
0	1	E06000001	Hartlepool	None	447160	531474	
1	2	E06000002	Middlesbrough	None	451141	516887	
2	3	E06000003	Redcar and Cleveland	None	464361	519597	
3	4	E06000004	Stockton-on-Tees	None	444940	518183	
4	5	E06000005	Darlington	None	428029	515648	

	LONG	LAT	Shape__Are	Shape__Len	\
0	-1.27018	54.6761	9.660727e+07	50737.909874	
1	-1.21099	54.5447	5.523093e+07	35500.289988	
2	-1.00608	54.5675	2.483255e+08	85085.268776	
3	-1.30664	54.5569	2.052160e+08	88846.876387	
4	-1.56835	54.5353	1.988128e+08	91926.839545	

	geometry	name	\
0	POLYGON ((448973.593 536745.277, 448986.025 53...	Hartlepool	
1	POLYGON ((451894.299 521145.303, 453997.697 51...	Middlesbrough	
2	POLYGON ((478232.599 518788.828, 477689.303 51...	Redcar and Cleveland	
3	POLYGON ((452243.536 526335.188, 451711.300 52...	Stockton-on-Tees	
4	POLYGON ((436388.002 522354.197, 437351.702 52...	Darlington	

	Median weekly pay (2020 GBP)	Conf %	Log median weekly pay (2020 GBP)
0	550.1	6.5	6.310100
1	517.5	7.5	6.249010
2	502.9	8.6	6.220391
3	558.6	6.3	6.325434
4	488.9	7.7	6.192158

```
1 df["cat_col"] = df["LAD20CD"].apply(
2     lambda x: x[0])
3 df.iloc[:5, -3:]
4
5 fig, ax = plt.subplots(figsize=(8, 10))
6 df.plot(
7     column="cat_col",
8     categorical=True,
9     ax=ax,
10    legend=True,
11    legend_kwds={
12        "loc": 1, "frameon": True},
13 )
14 ax.set_axis_off()
15 plt.show()
```



We can aggregate by any category

Problems with choropleth maps

Choropleths work best when the coloring represents a density (i.e., some quantity divided by surface area, as for example population density).

We perceive larger areas as corresponding to larger amounts than smaller areas and shading by density corrects for this effect.

However, in practice, we often see choropleths colored according to some quantity that is not a density.

For example, we made choropleth maps of median annual income.

Problems with choropleth maps

Such choropleth maps can be appropriate when they are prepared with caution.

There are two conditions under which we can color-map quantities that are not densities:

First, if all the individual areas we color have approximately the same size and shape, then we don't have to worry about some areas drawing disproportionate attention solely due to their size.

Problems with choropleth maps

Second, if the individual areas we color are relatively small compared to the overall size of the map and if the quantity that color represents changes on a scale larger than the individual colored areas, then again we don't have to worry about some areas drawing disproportionate attention solely due to their size.

Both of these conditions are approximately met with the Texan data, but not so much with the English data.

Problems with choropleth maps

It is also important to consider the effect of continuous versus discrete color scales in choropleth mapping.

While continuous color scales tend to look visually appealing, they can be difficult to read.

We are not very good at recognizing a specific color value and matching it against a continuous scale. Our smallest perceivable difference of color actually has to be quite large.

Problems with choropleth maps

Therefore, it is often appropriate to bin the data values into discrete groups that are represented with distinct colors. On the order of four to six bins is a good choice. The binning sacrifices some information, but on the flip side the binned colors can be uniquely recognized.

When comparing choropleths, we always have to keep in mind that with looking at the same geographical area we have a stable frame of reference. Sometimes this similarity of the underlying geometries can mislead us when we are comparing data that is actually quite dissimilar.

Cartograms in R

Not every map-like visualization has to be geographically accurate to be useful.

For example, in the US some states take up a comparatively large area but are sparsely populated while others take up a small area yet have a large number of inhabitants.

What if we deformed the states so their size was proportional to their number of inhabitants?

Such a modified map is called a **cartogram**.

```
1 # download.file("https://github.com/clauswilke/dviz.supp/raw/master/data/US_income.rda", "US_inco
2
3 load("US_income.rda")
4
5 # download.file("https://github.com/clauswilke/dviz.supp/raw/master/data/US_income_cartogram.rda", "US_inco
6
7 load("US_income_cartogram.rda")
8
9 US_income <- mutate(
10   US_income,
11   income_bins = cut(
12     ifelse(is.na(median_income), 25000, median_income), # hide missing value
13     breaks = c(0, 40000, 50000, 60000, 70000, 80000),
14     labels = c("< $40k", "$40k to $50k", "$50k to $60k", "$60k to $70k", "> $70k"),
15     right = FALSE
16   )
17 )
```

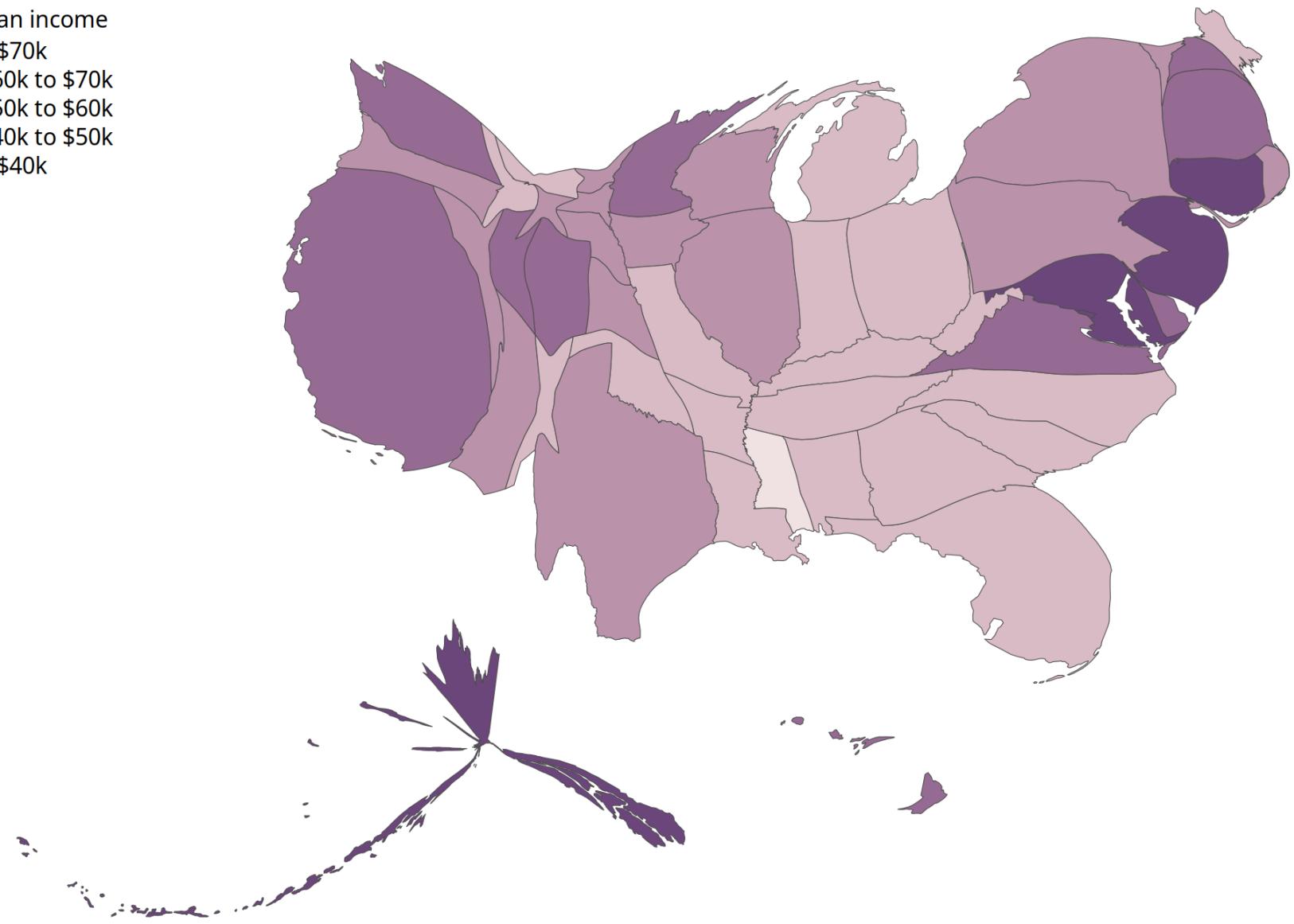
```

1 # copy binned data over, order of both data frames is the same
2 US_income_cartogram$income_bins <- US_income$income_bins
3
4 p <- ggplot(US_income_cartogram, aes(fill = income_bins)) +
5   geom_sf(color = "grey30", size = 0.2) + coord_sf(datum = NA, expand = FALSE) +
6   scale_x_continuous(limits = c(-3900000, 2500000)) +
7   scale_fill_discrete_sequential(
8     h1 = -83, h2 = 20, c1 = 30, cmax = 40, c2 = 0, l1 = 20, l2 = 100, p1 = 1, p2 = 1.2, rev = T
9     name = "median income",
10    nmax = 7,
11    order = 2:6,
12    guide = guide_legend(
13      override.aes = list(colour = "white", size = 1),
14      reverse = TRUE
15    )
16  ) +
17  theme_dviz_map(12, rel_small = 1) +
18  theme(
19    #plot.background = element_rect(fill = "cornsilk"),
20    legend.position = c(0, 1),
21    legend.justification = c(0, 1),
22    legend.spacing.x = grid::unit(3, "pt"),
23    legend.spacing.y = grid::unit(3, "pt"),

```

median income

- > \$70k
- \$60k to \$70k
- \$50k to \$60k
- \$40k to \$50k
- < \$40k



As an alternative to a cartogram with distorted shapes, we can also draw a much simpler cartogram heatmap, where each state is represented by a colored square.

While this representation does not correct for the population number in each state, and thus underrepresents more populous states and overrepresents less populous states, at least it treats all states equally and doesn't weigh them arbitrarily by their shape or size.

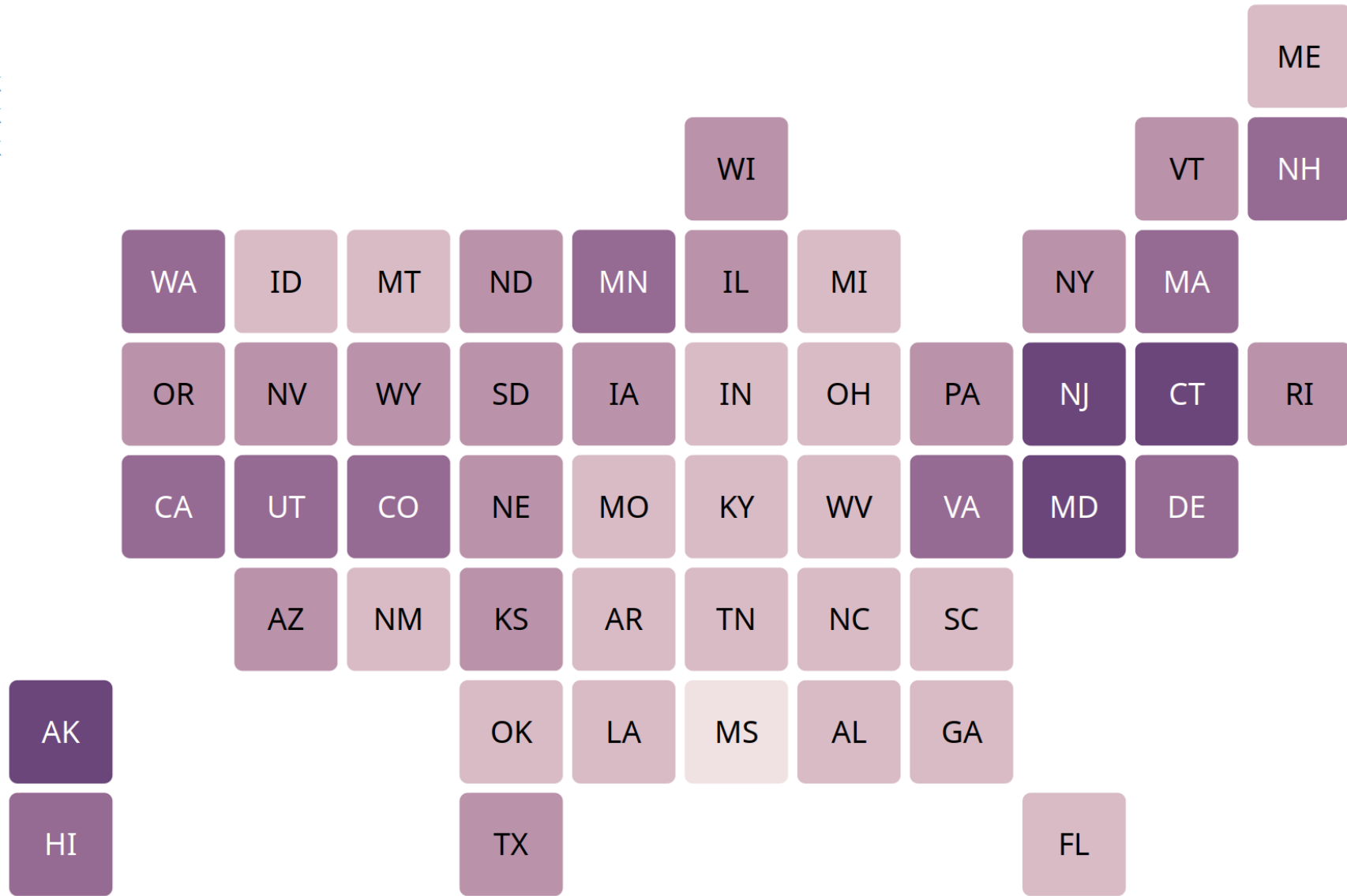
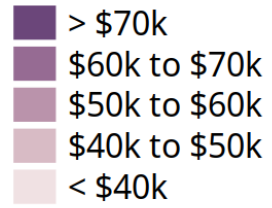
Lastly, we will have an outlook at more complex cartograms that put individual plots at the location of each state.

```

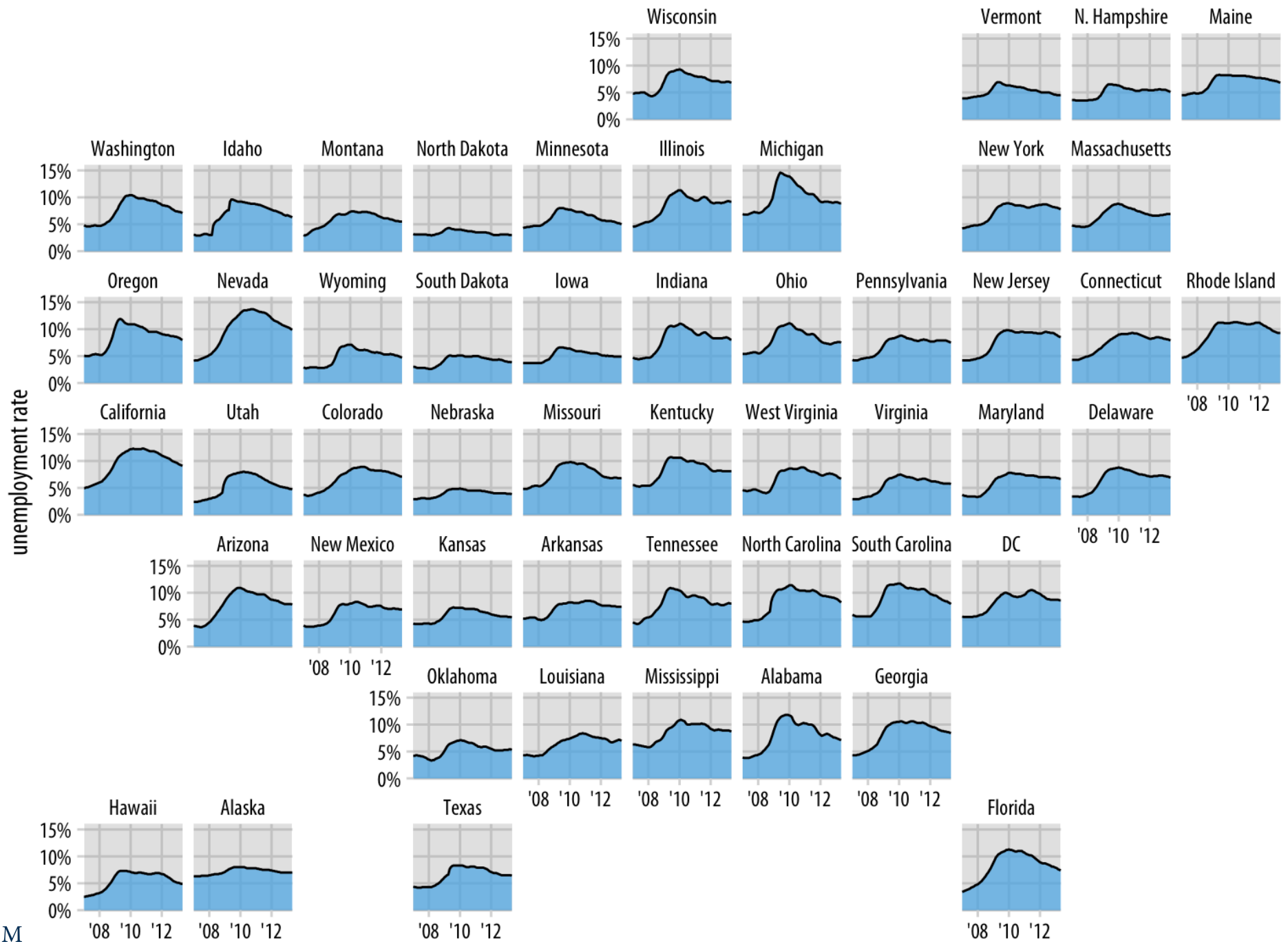
1 filter(US_income, name != "Puerto Rico", GEOID != "11") %>% # remove Puerto Rico and DC
2   ggplot(aes(state = name, fill = income_bins)) +
3   geom_statebins(family = "Myriad Pro",
4                 lbl_size = 14/.pt) +
5   expand_limits(x = -1.3) + # make space for legend
6   coord_equal(expand = FALSE) +
7   scale_fill_discrete_sequential(
8     h1 = -83, h2 = 20, c1 = 30, cmax = 40, c2 = 0, l1 = 20, l2 = 100, p1 = 1, p2 = 1.2, rev = T
9     name = "median income",
10    nmax = 7,
11    order = 2:6,
12    guide = guide_legend(
13      override.aes = list(colour = "white", size = 1),
14      reverse = TRUE
15    )
16  ) +
17  theme_dviz_map(12, rel_small = 1) +
18  theme(
19    #plot.background = element_rect(fill = "cornsilk"),
20    legend.background = element_blank(),
21    legend.position = c(0, 1),
22    legend.justification = c(0, 1),
23    legend.spacing.x = grid::unit(3, "pt"),

```

median income



For next time...



Acknowledgments

<https://wilkelab.org/SDS375/slides/geospatial-data.html>

<https://clauswilke.com/dataviz/geospatial-data.html>

<https://github.com/clauswilke/dataviz>

<https://en.wikipedia.org/wiki/GeoJSON>

<https://geojson.io/>

<https://aeturrell.github.io/coding-for-economists/geo-vis.html>

<https://www.datacamp.com/tutorial/making-map-in-python-using-plotly-library-guide>

<https://matplotlib.org/basemap/users/examples.html>