Geo-Spatial Data Visualizations

Focus: Geospatial Data Visualization

Exercises: GIS files, Geopandas, Geopands,plotly, Map classify

The learning curve for working with geospatial data goes through lots of terms, some of them are as:

- **shapefile:** data file format or information document design that can be used to show items on a map.
- **point:** (a specific location) A point is the smallest and most basic visual unit used to represent a single data record or location in a visualization. It marks a specific position in space — usually defined by one or more coordinate values (e.g., x, y or latitude, longitude). In data visualization, a point visually encodes an observation's position (and sometimes magnitude or category) within a defined coordinate system.
- **polygon**: In GIS and spatial data visualization, polygons are used to represent areas or regions on a map. Polygons form the building blocks of many charts and spatial visualizations:
  - -Choropleth maps (colored polygons = regions)
  - -Voronoi diagrams (each region is a polygon)
  - -Heatmap cells (often polygonal)

  They are filled with colors, gradients, or textures to represent quantitative attributes (e.g., population density, pollution level).
- **geometry:** a vector used to represent points, polygons, and other geometric shapes or locations.
- **basemap:** the background setting for a map
- **projection:** since the Earth is a 3D spheroid, chose a method for how an area gets flattened into a 2D map.
- **colormap:** the decision of a color shading palette for rendering data, chosen with the cmap parameter.
- **choropleth:** A choropleth map visualizes quantitative data (e.g., rates, ratios, counts) by color-shading geographic areas (e.g., countries, districts, census tracts). Each region's color intensity or hue represents a numerical value — for example:
  - -Population density
  - -Disease incidence rate
  - -Average income
  - -Air quality index
- **kernel density estimation:** a data smoothing procedure that generates a curve of shading to show different data levels.
- **cartogram:** a cartogram is a type of map visualization in which the geometry or size of geographic regions is distorted to represent a statistical variable, such as population, GDP, or disease cases — rather than showing true land area.
- **quantiles:** In data visualization, quantiles are threshold values that divide a dataset into equal-sized groups, based on the ranked order of a variable. They are used to group continuous data into categories so that each category (or color bin) contains roughly the same number of observations. Quantiles help assign color or symbol categories evenly across data values — making visual patterns clearer and more balanced.

Visualization Type	How Quantiles Are Applied	Purpose
Choropleth map	Divide numeric values into equal-count bins for coloring regions	Balanced color representation
Box plot	Defines quartiles (Q1, Q2/median, Q3) to show data spread	Summarize distribution visually
Histogram	Used as bin boundaries to represent percentiles	Compare relative frequencies
Violin / Density plots	Quantiles shown as markers	Indicate key distribution thresholds

- **Voronoi diagram**: Voronoi diagrams partition space into regions of influence, proximity, or dominance — a perfect way to visualize who's closest to what, which facility serves which area, or how space is organized around key points. They're a bridge between spatial analysis and visual storytelling — ideal for digital twins, healthcare access, smart city, and environmental mapping.

  A Voronoi diagram (or Thiessen polygon map) divides a space into regions based on distance to a set of points. Each region contains all locations that are closest to one particular point compared to any other point. So, if you drop several points (like hospitals or weather stations) on a map, the Voronoi diagram will show the zone of influence of each — i.e., which areas are closest to each station.

Practice Excercises

This exercise is based on US Census Beaurue's database is very popular in the literature and there are several online courses and blogs that help beginners to understand the basic concepts of the geospatial data visualization techniques.

Data Preparation

The map database is named "us_state.zip" which contains shapefiles accordingly.
This Dataset is consisting of Cartographic Boundary Files.

- These cartographic boundary files are simplified/disentangled portrayals of chose geographic zones from the Census Bureau's geographic database.
- These boundary files are purposely designed for small scale thematic mapping.

Types of Files

The cartographic boundary files are accessible in both shapefile and KML format. Each zipped file package contains either a shapefile or a KML file along with file-based metadata in XML format.

Advantages

- Simplified shapes improve the presence of geographic territories when shown at little scales.
- These boundary files occupy less circle room than their ungeneralized reciprocals.
- Cartographic boundary files set aside less effort and take less time to render on screen.

Limitations

Geographic areas may not line up with similar regions from another year. Some geographic areas are excluded from these files.

These files should not be used for:

- geographic analysis including area or perimeter calculation.
- geocoding addresses.
- determining precise geographic area relationships.

Scale

The cartographic boundary files are accessible at three target map scales i.e. not all geographic zone or area limits are available at all target scales:

- 1:500,000
- 1:5,000,000
- 1:20,000,000

We also need some data to plot on the map. We are going to use the population estimate of the US and are going to place it in a subdirectory called “data”.

- The U.S. Census Bureau is the main wellspring of statistical information about the nation’s people.
- Their population statistics come from decennial censuses, which check the whole U.S. populace at regular intervals, alongside a few different other surveys.

1. We will first import the necessary packages

---------- Code ----------

```
import geoplot as gplt
import geopandas as gpd
import geoplot.crs as gcrs
import imageio
import pandas as pd
import pathlib
import matplotlib.animation as animation
import matplotlib.pyplot as plt
import mapclassify as mc
import numpy as np
import pycountry
import plotly.express as px
```

2. We are going to load a shapefile and view parts of it. Note that the geometry column stipulates the polygon shapes.

Code

```
usa = gpd.read_file("maps/us_state.shp")
print(usa.head())
```

Output

	STATEFP	STATENS	AFFGEOID	GEOID	STUSPS	NAME	LSAD	ALAND	AWATER	geometry
0	24	01714934	0400000US24	24	MD	Maryland	00	25151100280	6979966958	MULTIPOLYGON (((-76.04621 38.02553, -76.00734 ...
1	19	01779785	0400000US19	19	IA	Iowa	00	144661267977	1084180812	POLYGON ((-96.62187 42.77925, -96.57794 42.827...
2	10	01779781	0400000US10	10	DE	Delaware	00	5045925646	1399985648	POLYGON ((-75.77379 39.72220, -75.75323 39.757...
3	39	01085497	0400000US39	39	OH	Ohio	00	105828882568	10268850702	MULTIPOLYGON (((-82.86334 41.69369, -82.82572 ...
4	42	01779798	0400000US42	42	PA	Pennsylvania	00	115884442321	3394589990	POLYGON ((-80.51989 40.90666, -80.51964 40.987...

3. We are going to load the US Census data as a pandas DataFrame and view a portion of it

Code

```
state_pop = pd.read_csv("data/us_state_est_population.csv")
print(state_pop.head())
```

Output

	SUMLEV	REGION	DIVISION	STATE	NAME	CENSUS2010POP	ESTIMATESBASE2010	POPESTIMATE2010	POPESTIMATE2011	POPESTIMATE2012
0	10	0	0	0	United States	308745538	308758105	309326085	311580009	313874218
1	20	1	0	0	Northeast Region	55317240	55318430	55380645	55600532	55776729
2	20	2	0	0	Midwest Region	66927001	66929743	66974749	67152631	67336937
3	20	3	0	0	South Region	114555744	114563045	114867066	116039399	117271075
4	20	4	0	0	West Region	71945553	71946887	72103625	72787447	73489477

4. Then merge the shapefile with population data, joining on the state names

Code

```
pop_states = usa.merge(state_pop, left_on="NAME", right_on="NAME")
pop_states.head()
```

Output

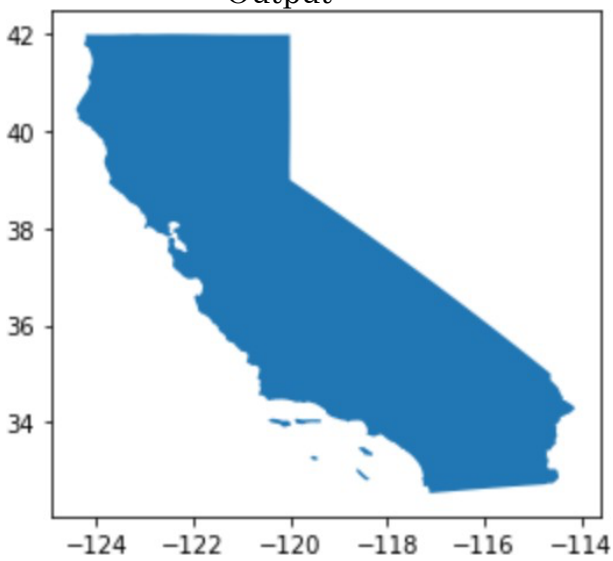
	STATEFP	STATENS	AFFGEOID	GEOID	STUSPS	NAME	LSAD	ALAND	AWATER	geometry	...	RDOMESTICMIG2017
0	24	01714934	0400000US24	24	MD	Maryland	00	25151100280	6979966958	MULTIPOLYGON (((-76.04621 38.02553, -76.00734 ...	...	-3.991992
1	19	01779785	0400000US19	19	IA	Iowa	00	144661267977	1084180812	POLYGON ((-96.62187 42.77925, -96.57794 42.827...	...	-1.278002
2	10	01779781	0400000US10	10	DE	Delaware	00	5045925646	1399985648	POLYGON ((-75.77379 39.72220, -75.75323 39.757...	...	4.689728
3	39	01085497	0400000US39	39	OH	Ohio	00	105828882568	10268850702	MULTIPOLYGON (((-82.86334 41.69369, -82.82572 ...	...	-0.698138
4	42	01779798	0400000US42	42	PA	Pennsylvania	00	115884442321	3394589990	POLYGON ((-80.51989 40.90666, -80.51964 40.987...	...	-2.144836

5. Now that data is ready to plot a shape. We'll specify "California" by name.

Code

```
pop_states[pop_states.NAME=="California"].plot()
```

Output



6. In its place, we can create a GeoDataFrame (a DataFrame with geospatial data) by loading one of the sample datasets from geoplot, in this case, the polygons for state boundaries

Code

```
path = gplt.datasets.get_path("contiguous_usa")
contiguous_usa = gpd.read_file(path)
contiguous_usa.head()
```

Output

	state	adm1_code	population	geometry
0	Minnesota	USA-3514	5303925	POLYGON ((-89.59941 48.01027, -89.48888 48.013...
1	Montana	USA-3515	989415	POLYGON ((-111.19419 44.56116, -111.29155 44.7...
2	North Dakota	USA-3516	672591	POLYGON ((-96.60136 46.35136, -96.53891 46.199...
3	Idaho	USA-3518	1567582	POLYGON ((-111.04973 44.48816, -111.05025 42.0...
4	Washington	USA-3519	6724540	POLYGON ((-116.99807 46.33017, -116.90653 46.1...

7. Then plot the map of the US map boundary,

Code

```
gplt.polyplot(contiguous_usa)
```

Output

8. Let's load another sample dataset, in this case for US cities,

Code

```
path = gplt.datasets.get_path("usa_cities")
usa_cities = gpd.read_file(path)
usa_cities.head()
```

Output

	id	POP_2010	ELEV_IN_FT	STATE	geometry
0	53	40888.0	1611.0	ND	POINT (-101.29627 48.23251)
1	101	52838.0	830.0	ND	POINT (-97.03285 47.92526)
2	153	15427.0	1407.0	ND	POINT (-98.70844 46.91054)
3	177	105549.0	902.0	ND	POINT (-96.78980 46.87719)
4	192	17787.0	2411.0	ND	POINT (-102.78962 46.87918)

9. Then plot the locations of each city in the continental US as points,

Code

```
continental_usa_cities = usa_cities.query('STATE not in ["HI", "AK", "PR"]')
gplt.pointplot(continental_usa_cities)
```

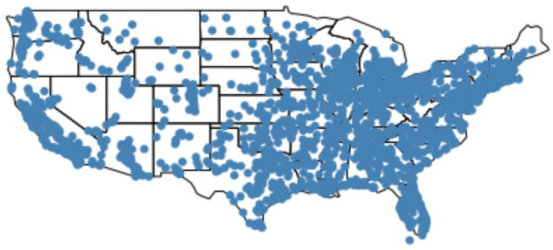
Output

10. Combining those two, we'll use overplotting to show the cities and states in the continental US. Note how the ax variable for the state polygons postulates an axis on which we are going to plot the cities accordingly,

Code

```
ax = gplt.polyplot(contiguous_usa)
gplt.pointplot(continental_usa_cities, ax=ax)
```

Output

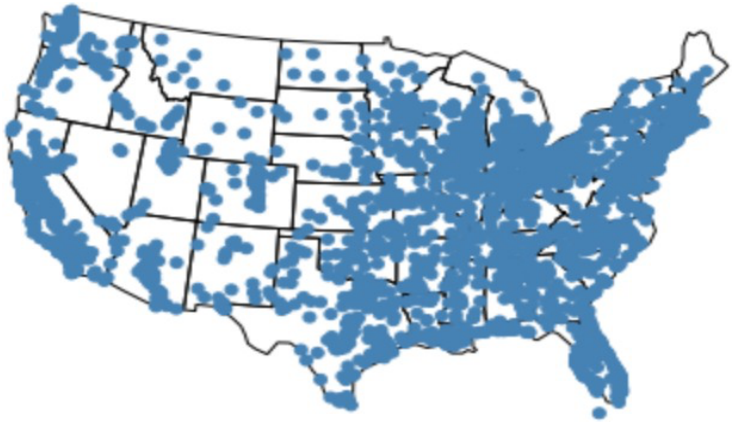


11. That looks a bit stretched, so let's adjust the projection to use an Albers equal-area conic projection,

Code

```
ax = gplt.polyplot(contiguous_usa, projection=gcrs.AlbersEqualArea())
gplt.pointplot(continental_usa_cities, ax=ax)
```

Output



12. let's compare several different ways to visualize geospatial data.
    - First, we'll change the color of a city's plotted point based on that city's elevation add a legend for people to decode the meaning of the different colors
    - The parameter lists start to get long-ish, so we'll specify parameters on different lines,

Code

```
ax = gplt.polyplot(contiguous_usa, projection=gcrs.AlbersEqualArea())

gplt.pointplot(
    continental_usa_cities,
    ax=ax,
    hue="ELEV_IN_FT",
    legend=True
)
```

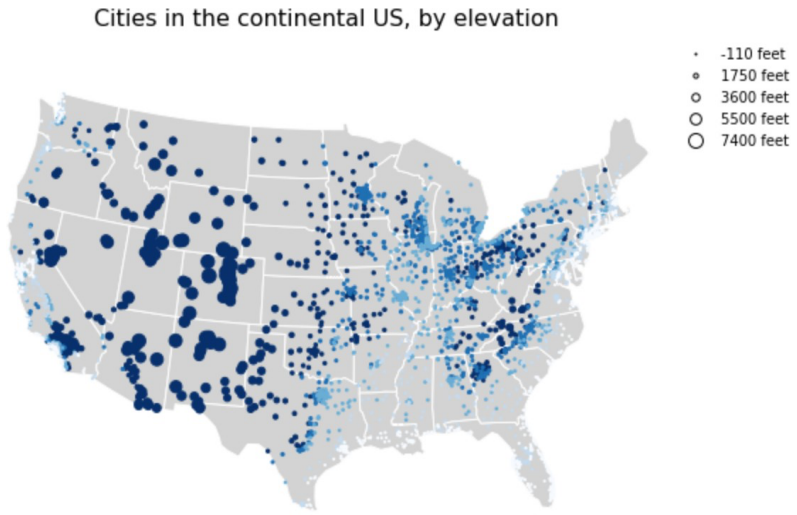
Output

13. We can also use the scale of each plotted point to represent another dimension.
    - In this case, the scale of the city points is based on their elevation,

Code

```
ax = gplt.polyplot(
    contiguous_usa,
    edgecolor="white",
    facecolor="lightgray",
    figsize=(12, 8),
    projection=gcrs.AlbersEqualArea()
)

gplt.pointplot(
    continental_usa_cities,
    ax=ax,
    hue="ELEV_IN_FT",
    cmap="Blues",
    scheme="quantiles",
    scale="ELEV_IN_FT",
    limits=(1, 10),
    legend=True,
    legend_var="scale",
    legend_kwargs={"frameon": False},
    legend_values=[-110, 1750, 3600, 5500, 7400],
    legend_labels=["-110 feet", "1750 feet", "3600 feet", "5500 feet", "7400 feet"]
)

ax.set_title("Cities in the continental US, by elevation", fontsize=16)
```

Output

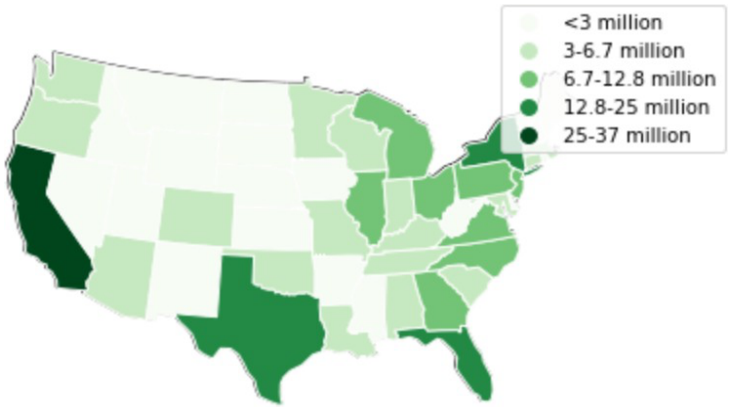


14. With a choropleth we use different hues or colors to shade polygons, to represent a dimension of data.

Code

```
ax = gplt.polyplot(contiguous_usa, projection=gcrs.AlbersEqualArea())

gplt.choropleth(
    contiguous_usa,
    hue="population",
    edgecolor="white",
    linewidth=1,
    cmap="Greens",
    legend=True,
    scheme="FisherJenks",
    legend_labels=[
        "<3 million", "3-6.7 million", "6.7-12.8 million",
        "12.8-25 million", "25-37 million"
    ],
    projection=gcrs.AlbersEqualArea(),
    ax=ax
)
```

Output



15. Next, we are going to show how to work with data associated with areas of polygons.
- We'll load a dataset about obesity rates by US state,

Code

```
obesity = pd.read_csv(gplt.datasets.get_path("obesity_by_state"), sep="\t")
obesity.head()
```

Output

	State	Percent
0	Alabama	32.4
1	Missouri	30.4
2	Alaska	28.4
3	Montana	24.6
4	Arizona	26.8

16. Convert that into a GeoDataFrame using a join.
-this will add the required "geometry" column,

Code

```
geo_obesity = contiguous_usa.set_index("state").join(obesity.set_index("State"))
geo_obesity.head()
```

Output

state	adm1_code	population	geometry	Percent
Minnesota	USA-3514	5303925	POLYGON ((-89.59941 48.01027, -89.48888 48.013...	25.5
Montana	USA-3515	989415	POLYGON ((-111.19419 44.56116, -111.29155 44.7...	24.6
North Dakota	USA-3516	672591	POLYGON ((-96.60136 46.35136, -96.53891 46.199...	31.0
Idaho	USA-3518	1567582	POLYGON ((-111.04973 44.48816, -111.05025 42.0...	29.6
Washington	USA-3519	6724540	POLYGON ((-116.99807 46.33017, -116.90653 46.1...	27.2

17. Now we can use this data to plot a cartogram, which grows or shrinks polygons to represent a dimension of data – i.e. the obesity rates per state,

Code

```
gplt.cartogram(
    geo_obesity,
    scale="Percent",
    projection=gcrs.AlbersEqualArea()
)
```

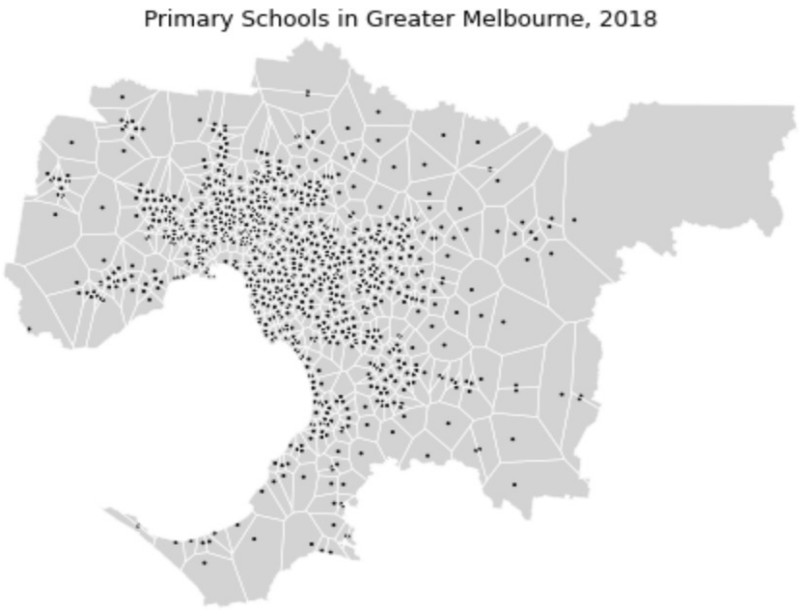
Output

18. Next, we are going to create a Voronoi diagram that calculates polygon areas based on a dimension of the data. Each polygon is positioned based on a creating point, with the end goal that each area in the polygon is nearer to its creating point than to some other.
    For example, we'll plot the locations of primary schools in Melbourne, Australia, and use a Voronoi diagram to demonstrate where they are concentrated,

Code

```
melbourne = gpd.read_file(gplt.datasets.get_path("melbourne"))
df = gpd.read_file(gplt.datasets.get_path("melbourne_schools"))
melbourne_primary_schools = df.query('School_Type == "Primary"')

ax = gplt.voronoi(
    melbourne_primary_schools,
    clip=melbourne,
    linewidth=0.5,
    edgecolor="white",
    projection=gcrs.Mercator()
)

gplt.polyplot(
    melbourne,
    edgecolor="None",
    facecolor="lightgray",
    ax=ax
)

gplt.pointplot(
    melbourne_primary_schools,
    color="black",
    ax=ax,
    s=1,
    extent=melbourne.total_bounds
)

plt.title("Primary Schools in Greater Melbourne, 2018")
```

Output



Covid-19 Visualization Exercise

The GIS technologies have played an important role in many aspects of the fight against COVID-19, including the data incorporation, and geospatial visualization of epidemic information, spatial tracking of confirmed cases, prediction of regional transmission, and many more. These provide support information for government sectors to fight against the COVID-19 spreading.
We are going to create a dynamic map visualizing the spreading of COVID-19 overtime using the COVID-19 dataset.

Used Python Libraries

For this exercise, we are going to use Pandas, Pycountry, and Plotly Express libraries.

Load Dataset

The dataset we are going to use here will be provided with the assignment accordingly. The dataset contains information about the total confirmed cases of covid19 throughout the whole world from January to October 2020 as a time-series.

-Load the data to the DataFrame,

Code

```
import pandas as pd
import pycountry
import plotly.express as px
```

```
df_confirmedGlobal = pd.read_csv("time_series_covid19_confirmed_global.csv")
print(df_confirmedGlobal.head())
```

Output

```
  Province/State Country/Region       Lat       Long  1/22/20  1/23/20  \
0            NaN    Afghanistan  33.93911  67.709953        0        0
1            NaN        Albania  41.15330  20.168300        0        0
2            NaN        Algeria  28.03390   1.659600        0        0
3            NaN        Andorra  42.50630   1.521800        0        0
4            NaN         Angola -11.20270  17.873900        0        0
```

Clean the Dataset

To visualize the time series dataset dynamically with Plotly Express, we are going to make the following changes to our DataFrame.

- **Aggregate the dataset** into the degree of the country.
- **Get the three-letter country codes** for each country; for example, AFG for Afghanistan.
    - we can do this by using PyCountry or any other method.
- **Transform the dataset in a long format** in which the date value is representing in a single column
    - we can do this by loading the dataset into the Pandas DataFrame and then using **melt** in Pandas.

Code

```
df_confirmedGlobal = df_confirmedGlobal.drop(columns=['Province/State', 'Lat', 'Long'])
df_confirmedGlobal = df_confirmedGlobal.groupby('Country/Region').agg('sum')
date_list = list(df_confirmedGlobal.columns)


# Get the three-letter country codes for each country
def get_country_code(name):
    try:
        return pycountry.countries.lookup(name).alpha_3
    except:
        return None


df_confirmedGlobal['country'] = df_confirmedGlobal.index
df_confirmedGlobal['iso_alpha_3'] = df_confirmedGlobal['country'].apply(get_country_code)

# Transform the dataset in a long format
df_long = pd.melt(df_confirmedGlobal, id_vars=['country', 'iso_alpha_3'], value_vars=date_list)
print(df_long)
```

Output

```
                      country iso_alpha_3 variable  value
0             Afghanistan         AFG  1/22/20      0
1                 Albania         ALB  1/22/20      0
2                 Algeria         DZA  1/22/20      0
3                 Andorra         AND  1/22/20      0
4                  Angola         AGO  1/22/20      0
...                   ...         ...      ...    ...
55665  West Bank and Gaza        None  11/9/20  58838
55666      Western Sahara         ESH  11/9/20     10
55667               Yemen         YEM  11/9/20   2071
55668              Zambia         ZMB  11/9/20  16971
55669            Zimbabwe         ZWE  11/9/20   8561
```

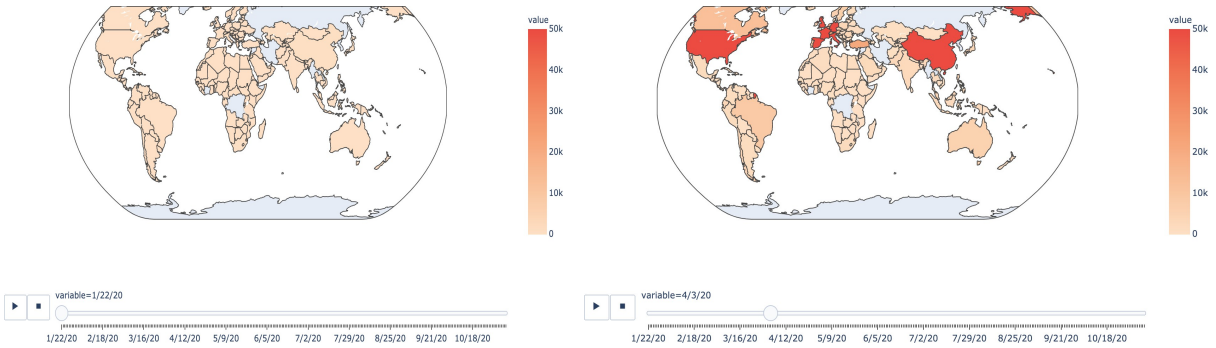
Create Map with Plotly Express

After the dataset is ready, we can easily create a map animation with Plotly Express. For example, we can create an animated choropleth map from the DataFrame from the above step as below (you may adjust each parameter as you like),

Code

```
fig = px.choropleth(df_long,
                    locations="iso_alpha_3",
                    color="value",
                    hover_name="country",
                    animation_frame="variable",
                    projection="natural earth",
                    color_continuous_scale='Peach',
                    range_color=[0, 50000]
                    )
fig.show()
fig.write_html("Covid19_map.html")  # write it to HTML file
```

Output



Useful Links for Geo-Spatial Data visualization

https://geohub-who.hub.arcgis.com

https://www.worldpop.org

https://cds.climate.copernicus.eu

https://gadm.org

https://docs.qgis.org/latest/en/docs/user_manual/processing_algs/qgis/vectorgeometry.html#voronoi-polygons