Interactive Visualization of Geographical Data

Learning Objectives

- Use choropleth maps to represent data across geographical regions
- Generate interactive choropleth maps, including choropleth maps depicting countries in the world and maps depicting states in the US, making layout changes to add functionality/aesthetic appeal, and adding animation
- Generate interactive scatter plots on maps, including scatter plots indicating geolocations of places of interest and interactive bubble plots on maps
- Generate interactive line plots on maps, including line plots indicating trajectories on a map

Introduction

Most datasets generated in today's world involve some features depicting spatial or geographical aspects. For example, users of social media platforms are characterized by the different parts of the world they reside in, world development metrics are calculated for different countries in the world, transportation routes span many different locations across the globe, and so on. Therefore, it is essential to learn systematic ways to understand and present such information in a digestible yet insightful manner.

While **altair** and **geopandas** provide exciting possibilities in visualizing geographical data, **plotly** is especially great for generating a variety of geographical plots that are easy to build, debug, and customize. Therefore, we will be using **plotly** to demonstrate generating different classes of geographical plots with multiple publicly available datasets from a variety of contexts.

Choropleth Maps

A **choropleth map** is a map of a region with different divisions coloured to indicate the value of a specific feature in that division. This *division* may be a country, state, district, or any other well-documented area.

For example, you can visualize country-wise populations using a world map, state-wise populations on a country map, or the percentage of a population with access to a certain technology with a choropleth map.

Let's start exploring the different types of choropleth maps.

Worldwide Choropleth Maps

In the first visualization, we are going to use the *internet usage statistics* published on **Our World in Data** (https://ourworldindata.org/internet) and present the percentage of the population using the internet in each country from **1990** to **2017**. The dataset is available on Moodle

You can view the dataset using the code that follows:

```
import pandas as pd

internet_usage_df = pd.read_csv("share-of-individuals-using-the-internet.csv")

internet_usage_df.head()
```

The output is as follows:

	Country	Code	Year	Individuals using the Internet (% of population)
0	Afghanistan	AFG	1990	0.000000
1	Afghanistan	AFG	2001	0.004723
2	Afghanistan	AFG	2002	0.004561
3	Afghanistan	AFG	2003	0.087891
4	Afghanistan	AFG	2004	0.105809

Figure 6.1: The Our World in Data dataset

Did you notice the feature called Code in the dataset? This refers to a code assigned to each country by a standard called ISO 3166-1. It is widely used so developers across the world have a common way to refer to and access country names in any data. You can learn more about the standard here: https://en.wikipedia.org/wiki/ISO_3166-1.The **Code** feature is also used by **plotly** to map data to the appropriate locations on the world map, as we will see soon.

Let's go ahead and generate our first world-wide choropleth map through an exercise.

Creating a Worldwide Choropleth Map

In this exercise, we'll generate a world-wide choropleth map using the *Our World in Data* dataset "share-of-individuals-using-the-internet.csv". Since the DataFrame contains records from multiple years, let's first subset the data to one specific year, say, **2016**. We'll then use this subset to generate a world-wide map. To do so, let's go through the following steps:

1. Import the Python modules:

   import pandas as pd

2. Read the data from the **.csv** file:

   internet_usage_df = pd.read_csv("data/share-of-individuals-using-the-internet.csv")

3. Subset the data to one specific year since the DataFrame contains records from multiple years:

   internet_usage_2016 = internet_usage_df.query("Year==2016")

   internet_usage_2016.head()

   The output is as follows:

	Country	Code	Year	Individuals using the Internet (% of population)
16	Afghanistan	AFG	2016	10.595726
39	Albania	ALB	2016	66.363445
63	Algeria	DZA	2016	42.945527
85	Andorra	AND	2016	97.930637
107	Angola	AGO	2016	13.000000

Figure 6.2: Subset of the Our World in Data dataset

   For the next steps, we're going to use the *express module* (for its simplicity) from **plotly** and use the **choropleth** function from the module. The first argument passed to this function is the DataFrame that we want to visualize. The following parameters are set:

   - *locations*: This is set to the name of the column in the DataFrame that contains the ISO 3166 country codes.
   - *color*: This is set to the name of the column that contains the numerical feature using which the map is to be color-coded.
   - *hover_name*: This is set to the name of the column that contains the feature to be displayed while hovering over the map.
   - *color_continuous_scale*: This is set to a color scheme, such as *Blues* | *Reds* | *Greens* | *px.colors.sequential.Plasma*.

     Note

     For more options, see the **plotly** express documentation (https://www.plotly.express/plotly_express/colors/index.html).

4. Generate an interactive world-wide choropleth map using **choropleth** function of **plotly** library:

```
import plotly.express as px
fig = px.choropleth(internet_usage_2016,
                    locations="Code", # colunm containing ISO 3166 country codes
                    color="Individuals using the Internet (% of population)",
# column by which to color-code
                    hover_name="Country", # column to display in hover information
                color_continuous_scale=px.colors.sequential.Plasma)
fig.show()
```

The output is as follows:

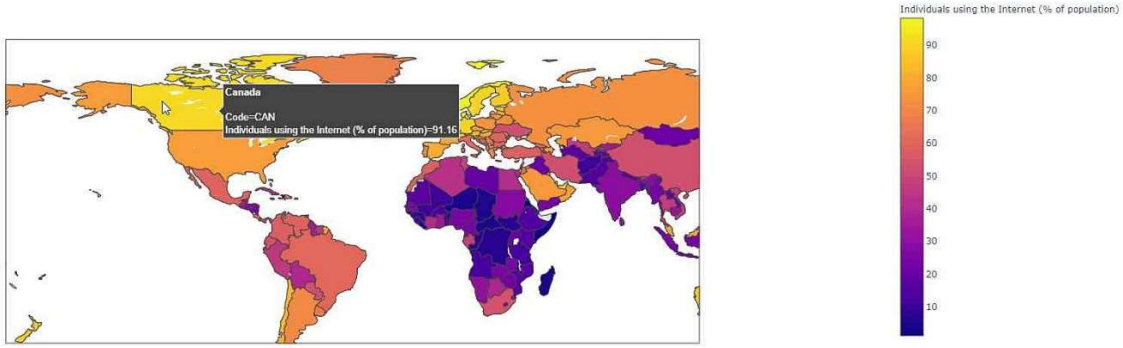

Figure 6.3a: World-wide choropleth map showing data for region=Canada

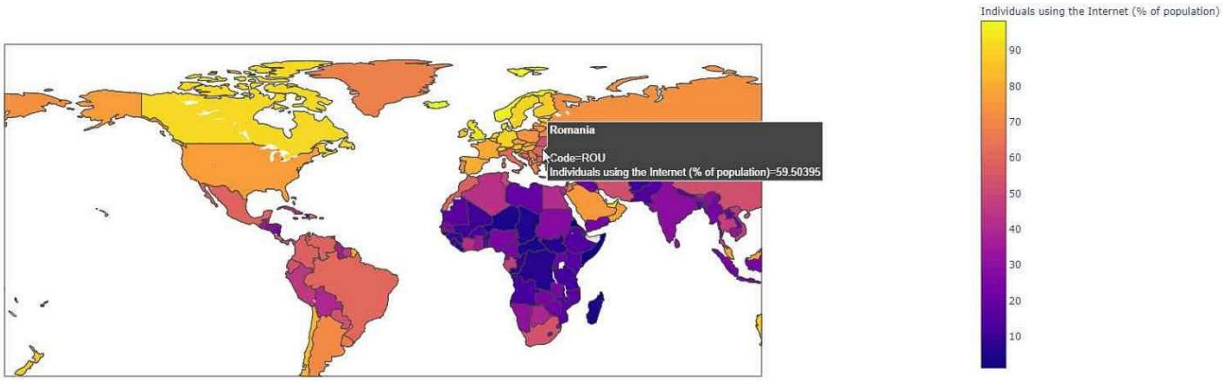

Figure 6.3b: World-wide choropleth map showing data for region=Romania

That was a quick way to get a beautiful plot!

Let's look at the plot carefully and see whether the observations match with our general knowledge. As you would expect, internet usage in the western world is higher than in the east.

Hover over the map a bit more. It is interesting to see, from Figure 6.3a and Figure 6.3b, that a higher percentage of the population (**~91.6**) in **Australia** and **Canada** have access to the internet than in the **US** and most **European** countries (**~59.5**).

What else does the plot show? Did you look at the sidebar at the top right of the plot? There you will see options for *selection types*, *zooming in and out*, *resetting the plot*, and even taking a *snapshot* of the plot in your choice of configuration.

It's worth playing around with the options a bit. Let's explore the interactivity of a choropleth map through the following exercise.

Tweaking a Worldwide Choropleth Map

In this exercise, we will make some simple changes to the layout of the choropleth map, such as changing the map projection from **flat** to **natural earth**, zooming into a specific region, adding text to the map using the **update_layout** function, and adding a **rotation** feature. The following code demonstrates how to add these functionalities to the map. We'll use the same dataset:

To do so, let's look at the following steps:

1. Import the Python modules:

   ```
   import pandas as pd
   ```

2. Read the data from the **.csv** file:

   ```
   internet_usage_df = pd.read_csv("data/share-of-individuals-using-the-internet.csv")
   ```

3. Subset the data to one specific year since the DataFrame contains records from multiple years:

   ```
   internet_usage_2016 = internet_usage_df.query("Year==2016")
   ```

4. Add title text to the choropleth map setting the **title_text** parameter:

   ```
   import plotly.express as px

   fig = px.choropleth(internet_usage_2016,

                       locations="Code",
   ```

```
                color="Individuals using the Internet (% of population)",
# column by which to color-code
                hover_name="Country",
# column to display in hover information
color_continuous_scale=px.colors.sequential.Plasma
)
fig.update_layout(
   # add a title text for the plot
   title_text = 'Internet usage across the world (% population) - 2016'
)
fig.show()
```

The output is as follows:

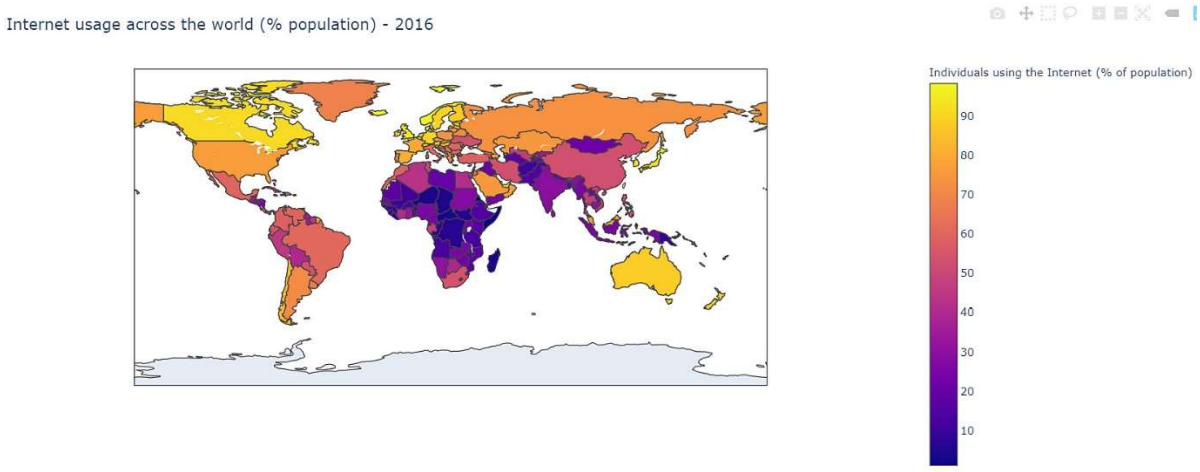

Figure 6.4: Adding text to the choropleth map

That's nice. But let's say, we are only interested in seeing internet usage across the continent of Asia.

5. Set **geo_scope** to **asia** in the **update_layout** function to zoom into the **asia** region. We can quickly do so with the following code:

```
import plotly.express as px

fig = px.choropleth(internet_usage_2016,

                    locations="Code",

                    color="Individuals using the Internet (% of population)", # column by which to color-code

                    hover_name="Country", # column to display in hover information

                    color_continuous_scale=px.colors.sequential.Plasma)

fig.update_layout(

   # add a title text for the plot

   title_text = 'Internet usage across the Asian Continent (% population) - 2016',

   geo_scope = 'asia'

# can be set to north america | south america | africa | asia | europe | usa

)

fig.show()
```

The output is as follows:

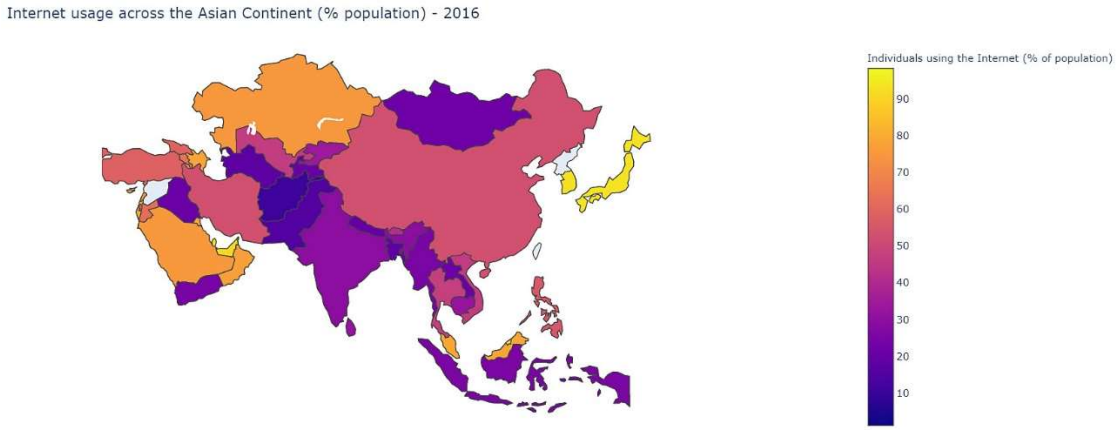

Figure 6.5: Choropleth map displaying the Asia region

Did you try dragging the plot and notice that it can move up and down or left and right? Wouldn't it be nice if the plot could rotate like a real globe? Well, that's easily possible too. All you need to do is to change the projection style of the map.

6. Set **projection type** to **natural earth**:

```
import plotly.express as px

fig = px.choropleth(internet_usage_2016,

                    locations="Code",

                    color="Individuals using the Internet (% of population)", # column by which to color-code

                    hover_name="Country", # column to display in hover information

                    color_continuous_scale=px.colors.sequential.Plasma)

fig.update_layout(

   # add a title text for the plot

   title_text = 'Internet usage across the world (% population) - 2016',

   # set projection style for the plot

   geo = dict(projection={'type':'natural earth'}) # by default, projection type is set to 'equirectangular'

)

fig.show()
```

The output is as follows:

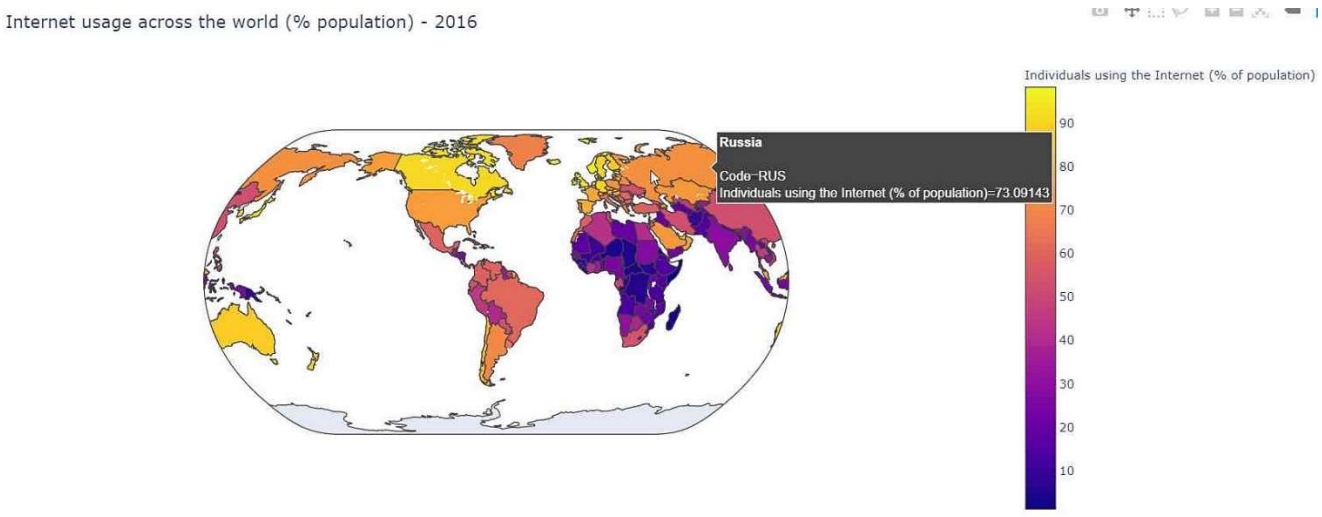

Figure 6.6: Choropleth map with projection type=natural earth

Try dragging the map now. The rotation gives the plot a much more realistic touch! **plotly** offers many such options to tweak visualizations. To experiment with other projection styles apart from the ones we have seen in our examples, visit the official **plotly** documentation here: https://plot.ly/python/reference/#layout-geo-projection.

It's now time to up the game! So far, we have been generating all our plots for the records in a single year, **2016**. *What about all the other timepoints?* While it is definitely possible to generate plots individually for each year we are interested in, that is not the most optimal use of a developer's time.

We'll see how to use animation, in such cases, on a choropleth map in the next section.

Animation in **plotly** choropleth maps is surprisingly easy. We simply need to set a parameter called **animation_frame** to the name of the column whose values we wish to animate our visualization for. Let's go through an exercise to understand how animation works on a choropleth map.

Adding Animation to a Choropleth Map

In this exercise, we'll animate a world-wide choropleth map. First, we'll choose a column. We'll then go ahead and add a slider component to the map to view records at different timepoints. We'll be using the same dataset on the share of populations using the internet. Let's go through the following steps:

1. Import the Python modules:

   ```
   import pandas as pd
   ```

2. Read the data from the **.csv** file:

   ```
   internet_usage_df = pd.read_csv("data/share-of-individuals-using-the-internet.csv")
   ```

3. Add an animation to the **year** column using **animation_frame=year**:

   ```
   import plotly.express as px

   fig = px.choropleth(internet_usage_df, locations="Code",
                       color="Individuals using the Internet (% of population)",
                       hover_name="Country", # column to add to hover information
                       animation_frame="Year", # column on which to animate
                       color_continuous_scale=px.colors.sequential.Plasma)
   ```

```
fig.update_layout(
    # add a title text for the plot
    title_text = 'Internet usage across the world (% population)',
    # set projection style for the plot
    geo = dict(projection={'type':'natural earth'}) # by default, projection type is set to 'equirectangular'
)
fig.show()
```

The output is as follows:

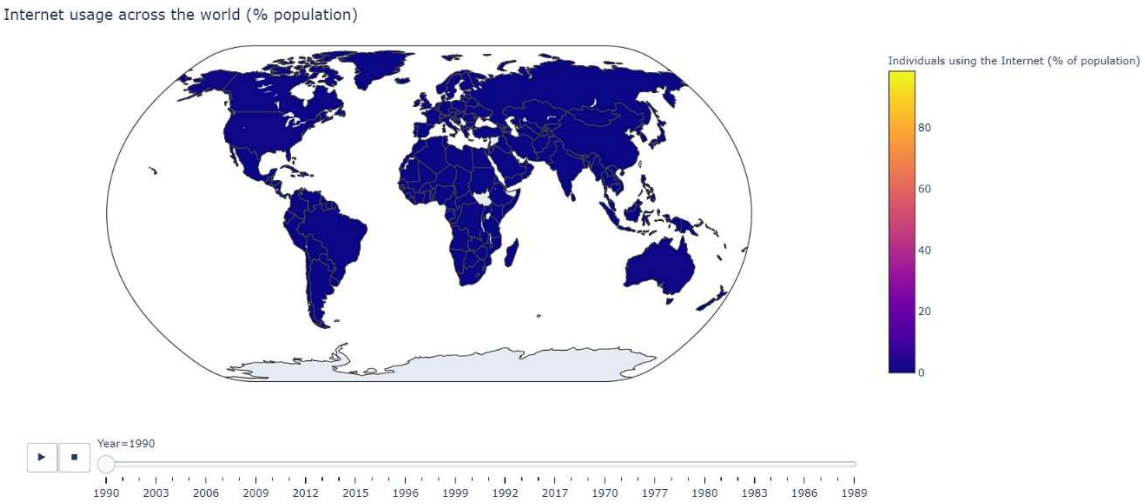

Figure 6.7: Choropleth map with a slider on the year column

Notice that the first argument to our **choropleth** function is the **internet_usage_df** DataFrame, which contains records for all the years between **1970-2017**, and not **internet_usage_2016**, which we had been using until now. If we used the **internet_usage_2016** DataFrame, we would get a *static plot* with no slider, since there would be nothing to animate with records only for a single year.

The animation functionality is really cool and the slider is a simple way to get a quick view of how internet usage has grown in different countries of the world over the years. However, something about the slider is funny! The years on the slider are not in the right order – it starts with **1990**, then goes all the way up to **2015**, and then goes back to **1970** and so on. The easiest way to fix this issue is to sort the DataFrame by time (the **Year** feature).

4. Sort the dataset by **Year** using the following code:

   internet_usage_df.sort_values(by=["Year"],inplace=True)

   internet_usage_df.head()

   The output is as follows:

	Country	Code	Year	Individuals using the Internet (% of population)
5347	Syrian Arab Republic	NaN	1960	0.0
718	Burundi	BDI	1960	0.0
5493	Togo	TGO	1960	0.0
572	Botswana	BWA	1960	0.0
3414	Maldives	MDV	1960	0.0

Figure 6.8: Sorted internet usage dataset

5. Generate the animated plot again now that the sorting is done:

   ```
   import plotly.express as px
   fig = px.choropleth(internet_usage_df, locations="Code",
                       color="Individuals using the Internet (% of population)",
                       hover_name="Country", # column to add to hover information
                       animation_frame="Year", # column on which to animate
                       color_continuous_scale=px.colors.sequential.Plasma)


   fig.update_layout(
      # add a title text for the plot
      title_text = 'Internet usage across the world (% population)',
      # set projection style for the plot
      geo = dict(projection={'type':'natural earth'}) # by default, projection type is set to 'equirectangular'
   )
   ```

fig.show()

The output is as follows:

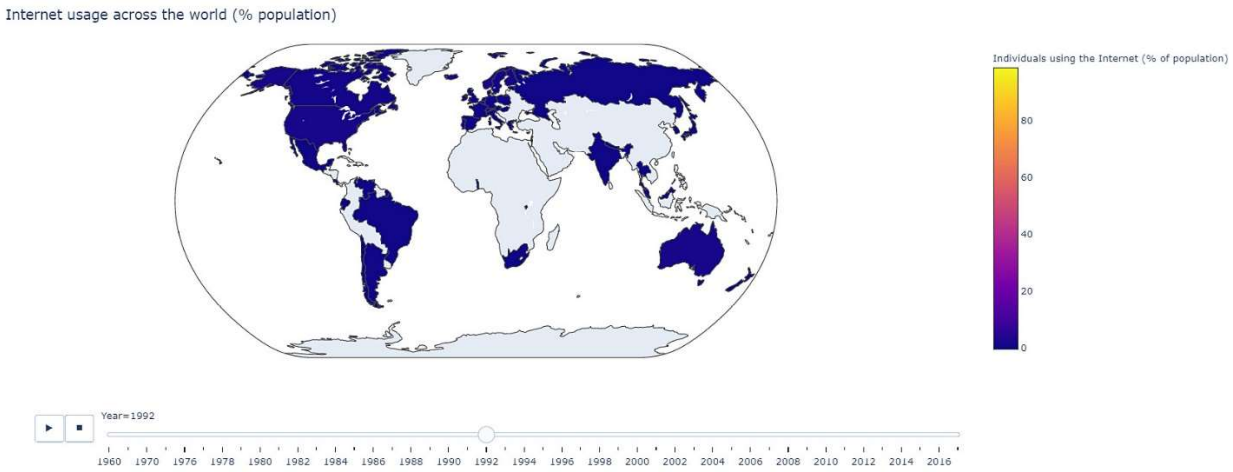

Figure 6.9a: First plot – choropleth map for the year 1992

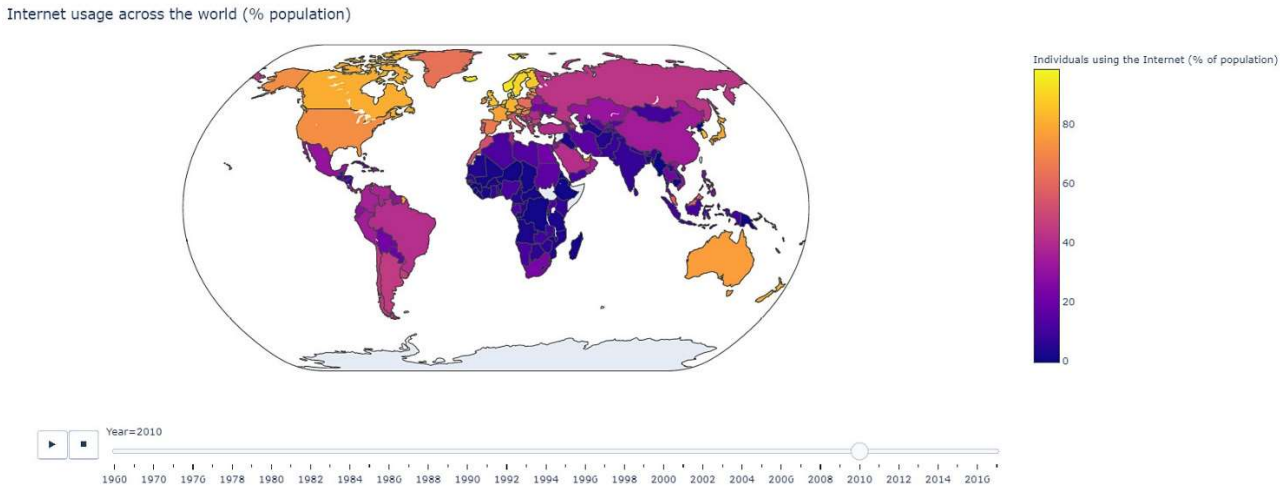

Figure 6.9b: Second plot – choropleth map for the year 2010

And this time, it's right! First plot shows internet usage across the world in the year **1992** while second plot shows the results for the year **2010**. We can see there was definitely an increase in internet usage between **1992** and **2010**.

There is one more point that needs to addressed before we close our discussion on worldwide choropleth maps. In your work, you may come across datasets that would be interesting to visualize on a geographical map but do not have a column that indicates their ISO 3166-1 code. In such cases, you can download the country codes from the official ISO website: https://www.iban.com/country-codes. For easy access, I have also uploaded these country codes to Moodle.

You'll be able to view the **country codes** dataset using the following code:

```
# get the country codes data

import pandas as pd

country_codes = pd.read_csv("data/country_codes.tsv", sep='\t')

country_codes.head()
```

The output is as follows:

	Country	Alpha-2 code	Alpha-3 code	Numeric
0	Afghanistan	AF	AFG	4
1	Albania	AL	ALB	8
2	Algeria	DZ	DZA	12
3	American Samoa	AS	ASM	16
4	Andorra	AD	AND	20

Figure 6.10: Country codes dataset

USA State Maps

While the goal of many visualizations is to compare and contrast specific features across countries, there are often also contexts in which we need to analyse features across smaller regions – such as states within a country. To generate choropleth maps for states in the US, we will be using the state-wise population data made available on the US census website: https://www.census.gov/newsroom/press-kits/2018/pop-estimates-national-state.html. I have also made this available on Moodle: “us_state_population.tsv”

Creating a USA State Choropleth Map

In this exercise, we'll be using the **USA state population** dataset. We'll tweak the dataset and use it to plot a state-wide choropleth map. Then, we'll change the layout of this map to show the US population across states. Let's go through the following steps to do so:

1. Import the Python module:

   ```
   import pandas as pd
   ```

2. Read the dataset from the URL:

```
df = pd.read_csv(“data/us_state_population.tsv”, sep='\t')
```

df.head()

The output is as follows:

	State	Code	2010	2011	2012	2013	2014	2015	2016	2017	2018
0	Alabama	AL	4785448	4798834	4815564	4830460	4842481	4853160	4864745	4875120	4887871
1	Alaska	AK	713906	722038	730399	737045	736307	737547	741504	739786	737438
2	Arizona	AZ	6407774	6473497	6556629	6634999	6733840	6833596	6945452	7048876	7171646
3	Arkansas	AR	2921978	2940407	2952109	2959549	2967726	2978407	2990410	3002997	3013825
4	California	CA	37320903	37641823	37960782	38280824	38625139	38953142	39209127	39399349	39557045

Figure 6.11:USA state population dataset

It is nice that this dataset also has the state codes available in the **Code** feature. However, the data is not in the format we would want it to be – it's in the wide format, and we need it to be long.

3. Use the **melt** function to convert the data to the desired format:

df = pd.melt(df, id_vars=['State', 'Code'], var_name="Year", value_name="Population")

df.head()

The output is as follows:

	State	Code	Year	Population
0	Alabama	AL	2010	4785448
1	Alaska	AK	2010	713906
2	Arizona	AZ	2010	6407774
3	Arkansas	AR	2010	2921978
4	California	CA	2010	37320903

Figure 6.12: Dataset after using the melt function

Once you know how to generate a choropleth map for countries in the world, a choropleth map of US states is quite straight-forward. Unlike the case of generating a worldwide choropleth map where we used the **plotly express** module, we'll use the **graph_objects** module to generate the choropleth map for states in the US. There are a few simple steps involved in drawing the US choropleth:

4. Import the **graph_objects** module:

import plotly.graph_objects as go

5. Initialize the figure with the **Figure** function in **graph_objects**. Specifically, the **data** argument needs to be an instance of the **Choropleth** class with the following parameters:
    - **locations**: This is set to the column of the DataFrame that contains the state name codes.
    - **z**: This is set to the column containing the numerical feature using which the map is to be color-coded.
    - **locationmode**: This is set to **USA-states**.
    - **colorscale**: This is set to a color scheme, such as **Blues | Reds | Greens**. For more options, see the **plotly** official documentation: https://plot.ly/python/reference/.
    - **colorbar_title**:This is set to the title of the color bar on the right, indicating the correspondence of color and feature values. Refer to the following code:

```
# initialize the figure

fig = go.Figure(

  data=go.Choropleth(

    locations=df['Code'], # Code for US states

    z = df['Population'].astype(int), # Data to be color-coded

    locationmode = 'USA-states', # set of locations match entries in 'locations'

    colorscale = 'Blues',

    colorbar_title = "Population",

  )

)
```

6. Make changes to the layout with **update_layout()** – set **title_text** and **geo_scope**:

```
# update layout

fig.update_layout(

  title_text = 'US Population across states',

  geo_scope='usa', # limit map scope to USA

)
```

fig.show()

The output is as follows:

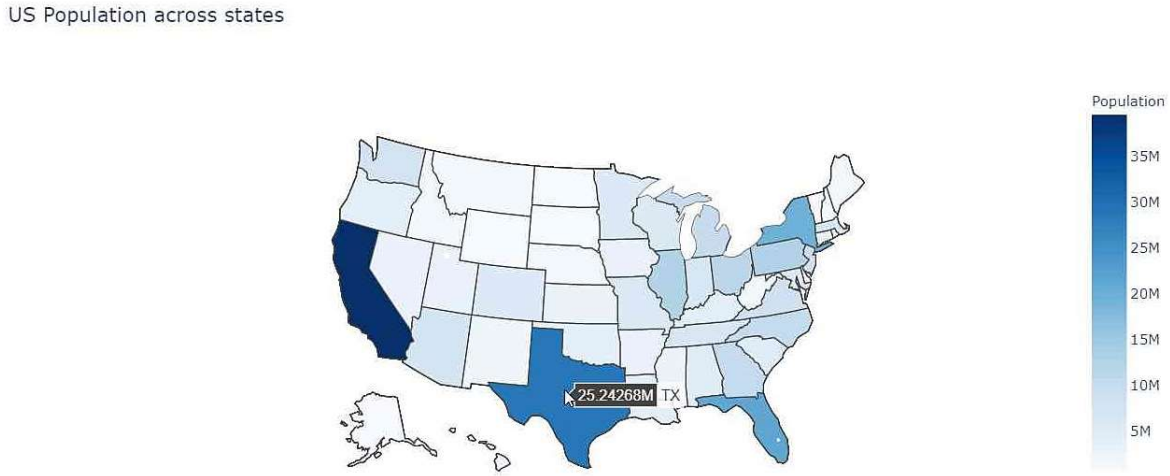

Figure 6.13: State map with updated layout

Choropleth maps are an effective way to visualize aggregate statistics across divisions of a geographical region. Two modules from **plotly express** and **graph_objects** – can be used to generate interactive choropleth maps. The modules map records of divisions such as countries and states to locations on geographical maps using a system of standardized country and state code names.

In the next section, we'll explore how to create scatter plots and bubble plots on geographical maps.

Plots on Geographical Maps

While the previous plots were great for visualizing more global trends – such as countries or states – what if we want to represent features in smaller regions, say within individual states? In this section, you will learn how to draw scatter plots and bubble plots on maps. The most intuitive plot of this type is one that simply pinpoints certain locations of interest on the map.

Scatter Plots

We will be plotting the locations of Walmart stores on a map of the US. This dataset is publicly available at: https://github.com/plotly/datasets/ on the **plotly** website, its also available on Moodle

Creating a Scatter Plot on a Geographical Map

To create a scatter plot from this dataset, we'll be using the **graph_objects** module. We'll find a location of interest on the map and we'll assign longitudes and latitudes on that map and

find out the number of Walmart store openings for different parts of the US. To do so, let's go through the following steps:

1. Import Python modules:

   import pandas as pd

2. Read the data from the URL:

   walmart_loc_df = pd.read_csv("data/1962_2006_walmart_store_openings.csv")

   walmart_loc_df.head()

   The output is as follows:

	storenum	OPENDATE	date_super	conversion	st	county	STREETADDR	STRCITY	STRSTATE	ZIPCODE	type_store	LAT	LON	MONTH	DAY	YEAR
0	1	7/1/62	3/1/97	1.0	5	7	2110 WEST WALNUT	Rogers	AR	72756	Supercenter	36.342235	-94.07141	7	1	1962
1	2	8/1/64	3/1/96	1.0	5	9	1417 HWY 62/65 N	Harrison	AR	72601	Supercenter	36.236984	-93.09345	8	1	1964
2	4	8/1/65	3/1/02	1.0	5	7	2901 HWY 412 EAST	Siloam Springs	AR	72761	Supercenter	36.179905	-94.50208	8	1	1965
3	8	10/1/67	3/1/93	1.0	5	29	1621 NORTH BUSINESS 9	Morrilton	AR	72110	Supercenter	35.156491	-92.75858	10	1	1967
4	7	10/1/67	NaN	NaN	5	119	3801 CAMP ROBINSON RD.	North Little Rock	AR	72118	Wal-Mart	34.813269	-92.30229	10	1	1967

Figure 6.14: Walmart store opening dataset showing data from 1962-2006

   We will again be using the **graph_objects** module to generate our scatter plot on the US map. As for the choropleth map, we will use the **Figure** function from **graph_objects** and the **update_layout()** function. However, this time, we will be assigning an instance of the **Scattergeo** class as the argument to **Figure()**. We will be passing the longitudes and latitudes of our locations of interest using the **lon** and **lat** parameters.

3. Plot the scatter plot using the **update_layout** function:

   import plotly.graph_objects as go

   fig = go.Figure(data=go.Scattergeo(

   lon = walmart_loc_df['LON'],

   # column containing longitude information of the locations to plot

   lat = walmart_loc_df['LAT'],

   # column containing latitude information of the locations to plot

```
        text = walmart_loc_df['STREETADDR'],
    # column containing value to be displayed on hovering over the map
        mode = 'markers' # a marker for each location
        ))
    fig.update_layout(
        title = 'Walmart stores across world',
        geo_scope='usa',
      )
    fig.show()
```

The output is as follows:

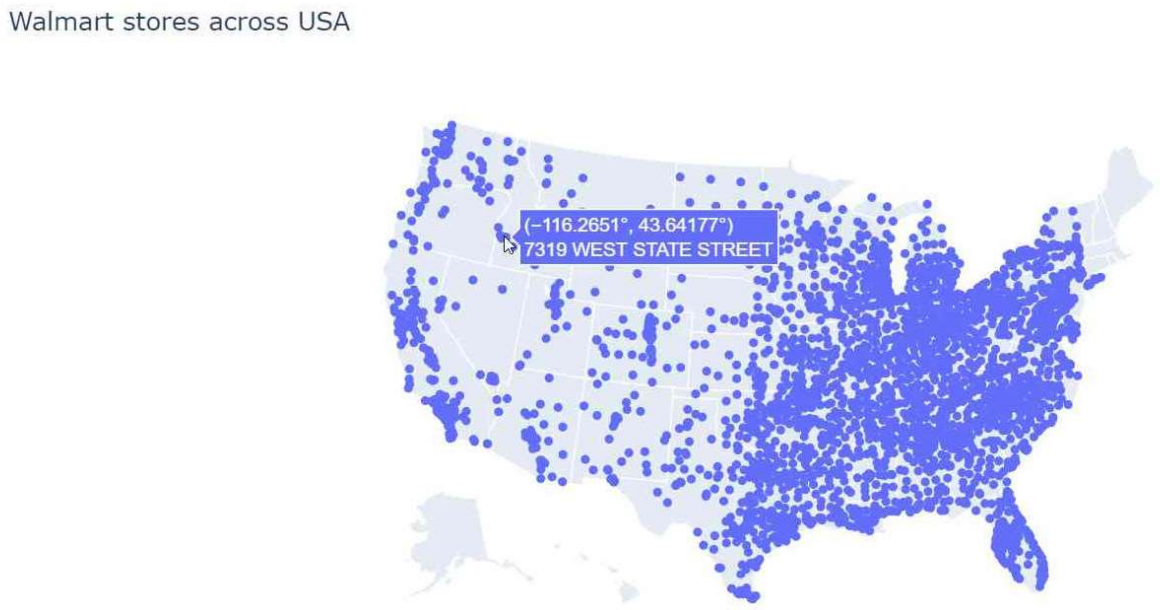

Figure 6.15: Scatter plot for Walmart stores across the US

And that's it – a scatter plot on a map. A striking observation is that Walmart is much more prominent in the east of the US than the west of the US.

Let's go ahead a look at bubble plots on geographical maps.

Bubble Plots

Since the eastern side of the map of the USA appears very densely populated with Walmart stores, it might be a good idea to show an aggregate feature, such as the count of Walmart stores across the different states. **Bubble plots** are designed for exactly this kind of visualization. In the current context of visualizing geographical data, bubble plots are plots with as many bubbles as regions of interest, where the bubble sizes depend on the value they are indicating – the bigger the value, the bigger the bubble.

Creating a Bubble Plot on a Geographical Map

In this exercise, we'll use the Walmart store openings dataset from **1962-2006** again and generate a bubble plot to see the number of Walmart stores across different states in the USA. Then, we'll look at another context and generate a bubble plot using the **internet_usage** dataset to find out the number of internet users across the world. We'll also animate the bubble plot to show the increase in the number of internet users across the world. To do so, let's go through the following steps:

1. Import the Python modules:

   import pandas as pd

2. Read the data from the URL:

   walmart_loc_df = pd.read_csv("data/1962_2006_walmart_store_openings.csv")

   walmart_loc_df.head()

   #The output is as follows:

	storenum	OPENDATE	date_super	conversion	st	county	STREETADDR	STRCITY	STRSTATE	ZIPCODE	type_store	LAT	LON	MONTH	DAY	YEAR
0	1	7/1/62	3/1/97	1.0	5	7	2110 WEST WALNUT	Rogers	AR	72756	Supercenter	36.342235	-94.07141	7	1	1962
1	2	8/1/64	3/1/96	1.0	5	9	1417 HWY 62/65 N	Harrison	AR	72601	Supercenter	36.236984	-93.09345	8	1	1964
2	4	8/1/65	3/1/02	1.0	5	7	2901 HWY 412 EAST	Siloam Springs	AR	72761	Supercenter	36.179905	-94.50208	8	1	1965
3	8	10/1/67	3/1/93	1.0	5	29	1621 NORTH BUSINESS 9	Morrilton	AR	72110	Supercenter	35.156491	-92.75858	10	1	1967
4	7	10/1/67	NaN	NaN	5	119	3801 CAMP ROBINSON RD.	North Little Rock	AR	72118	Wal-Mart	34.813269	-92.30229	10	1	1967

Figure 6.16: Walmart store opening dataset

3. Use the **groupby** function to compute the number of Walmart stores per state. If you don't remember how to do this, it might be a good idea to revise the relevant concepts from the first chapter:

```
walmart_stores_by_state =
walmart_loc_df.groupby('STRSTATE').count()['storenum'].reset_index().rename(colu
mns={'storenum':'NUM_STORES'})

walmart_stores_by_state.head()
```

The output is as follows:

	STRSTATE	NUM_STORES
0	AL	90
1	AR	81
2	AZ	55
3	CA	159
4	CO	56

Figure 6.17: Truncated Walmart store openings dataset

4. To generate the bubble plots, we will use the **plotly express** module and the **scatter_geo** function. Notice how the **locations** parameter is set to the name of the column that contains state codes, and the **size** parameter is set to the **NUM_STORES** feature:

```
import plotly.express as px

fig = px.scatter_geo(walmart_stores_by_state,

                     locations="STRSTATE", # name of column which contains state codes

                     size="NUM_STORES", # name of column which contains aggregate
value to visualize

                     locationmode = 'USA-states',

                     hover_name="STRSTATE",

                     size_max=45)



fig.update_layout(

   # add a title text for the plot

   title_text = 'Walmart stores across states in the US',

   # limit plot scope to USA
```

```
    geo_scope='usa'
)
fig.show()
```

The output is as follows:

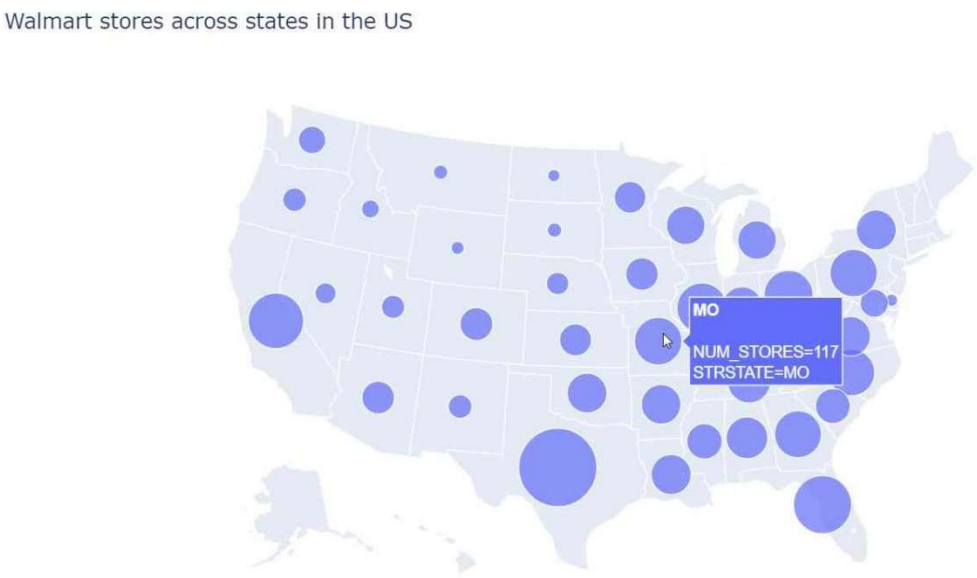

Figure 6.18: Bubble plot

Can you think of any other contexts where a bubble plot may be useful for visualization? How about revisiting the internet usage data (on the percentages of the population using the internet in each country) to generate a world-wide bubble plot? However, bubble plots are more suitable and intuitive for presenting counts/numbers, rather than percentages in individual regions.

It turns out that the count of individuals using the internet in each country is also available from the same resource (Our World in Data: https://ourworldindata.org/internet) that we used to collect our previous data.

Use the following code to read data from the **internet users by country** dataset:

```
import pandas as pd
internet_users_df = pd.read_csv("data/share-of-individuals-using-the-internet.csv")
internet_users_df.head()
```

The output is as follows:

	Country	Code	Year	Number of internet users (users)
0	Afghanistan	AFG	1990	0
1	Afghanistan	AFG	2001	990
2	Afghanistan	AFG	2002	1003
3	Afghanistan	AFG	2003	20272
4	Afghanistan	AFG	2004	25520

Figure 6.19: Internet users dataset

5. Sort the DataFrame by the **Year** feature:

   internet_users_df.sort_values(by=['Year'],inplace=True)

   internet_users_df.head()

   The output is as follows:

	Country	Code	Year	Number of internet users (users)
0	Afghanistan	AFG	1990	0
1257	Eritrea	ERI	1990	0
1236	Equatorial Guinea	GNQ	1990	0
4016	Timor	TLS	1990	0
1214	El Salvador	SLV	1990	0

Figure 6.20: Internet users dataset after sorting by year

6. Plot the number of users using the internet across the world in **2016**:

   import plotly.express as px

   fig = px.scatter_geo(internet_users_df.query("Year==2016"),

   locations="Code", # name of column indicating country-codes

   size=" Individuals using the Internet (% of population)",

   # name of column by which to size the bubble

   hover_name="Country",

   # name of column to be displayed while hovering over the map

   size_max=20,

   # parameter to scale all bubble sizes

color_continuous_scale=px.colors.sequential.Plasma)

```
fig.update_layout(
    # add a title text for the plot
    title_text = 'Internet users across the world - 2016',
    # set projection style for the plot
    geo = dict(projection={'type':'natural earth'}) # by default, projection type is set to 'equirectangular'
)
fig.show()
```

The output is as follows:

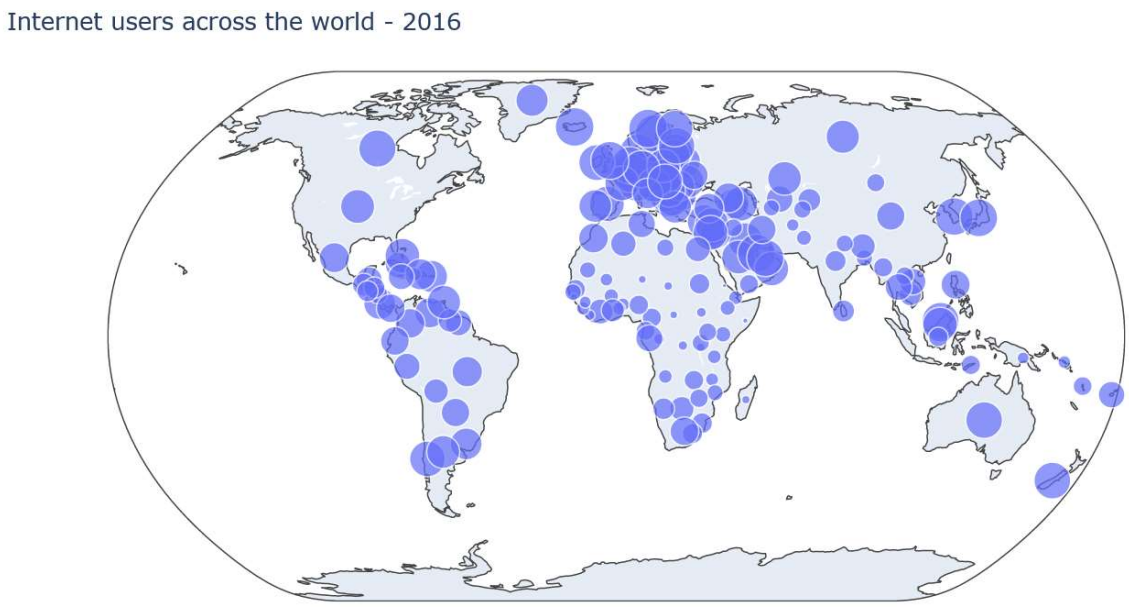

Figure 6.21: Bubble plot to see the number of internet users across the world

7. Animate the bubble plot to show the increase in the number of internet users over the years by using the **animation_frame** parameter:

```
import plotly.express as px
fig = px.scatter_geo(internet_users_df,
```

```
                locations="Code",
 # name of column indicating country-codes
                size="Individuals using the Internet (% of population)",
 # name of column by which to size the bubble
                hover_name="Country",
# name of column to be displayed while hovering over the map
                size_max=20, # parameter to scale all bubble size
                animation_frame="Year",
                )


fig.update_layout(
    # add a title text for the plot
    title_text = 'Internet users across the world',
    # set projection style for the plot
    geo = dict(projection={'type':'natural earth'}) # by default, projection type is set to
'equirectangular'
)
fig.show()
```

The output is as follows:

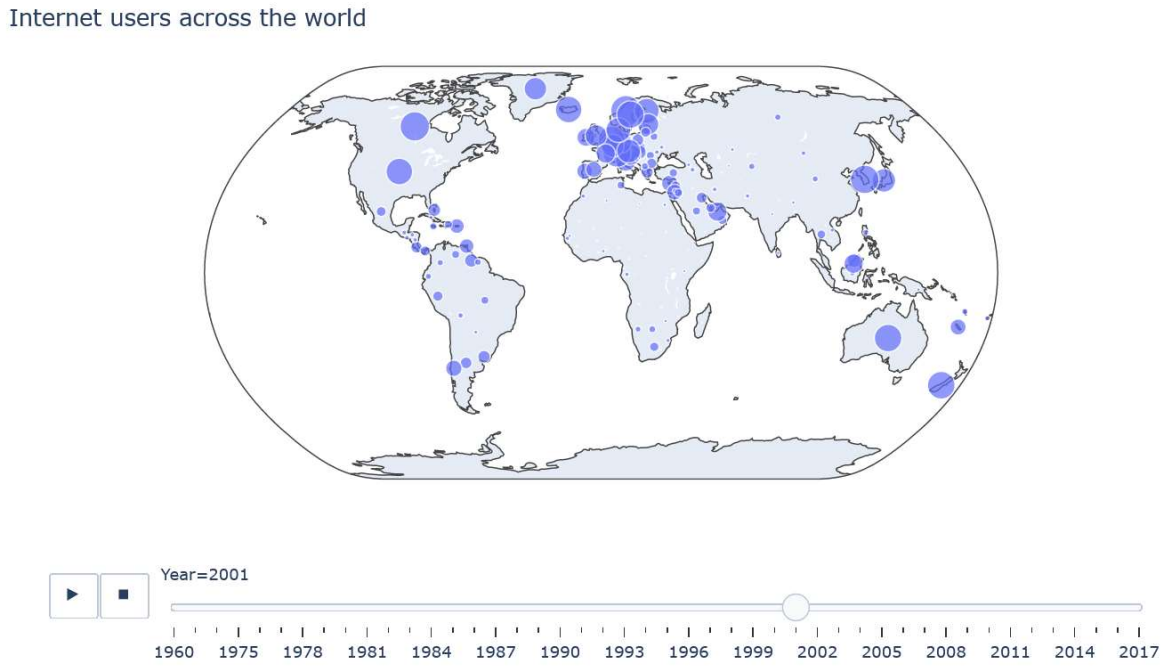

Figure 6.22a: Animated bubble plot for the year 2001

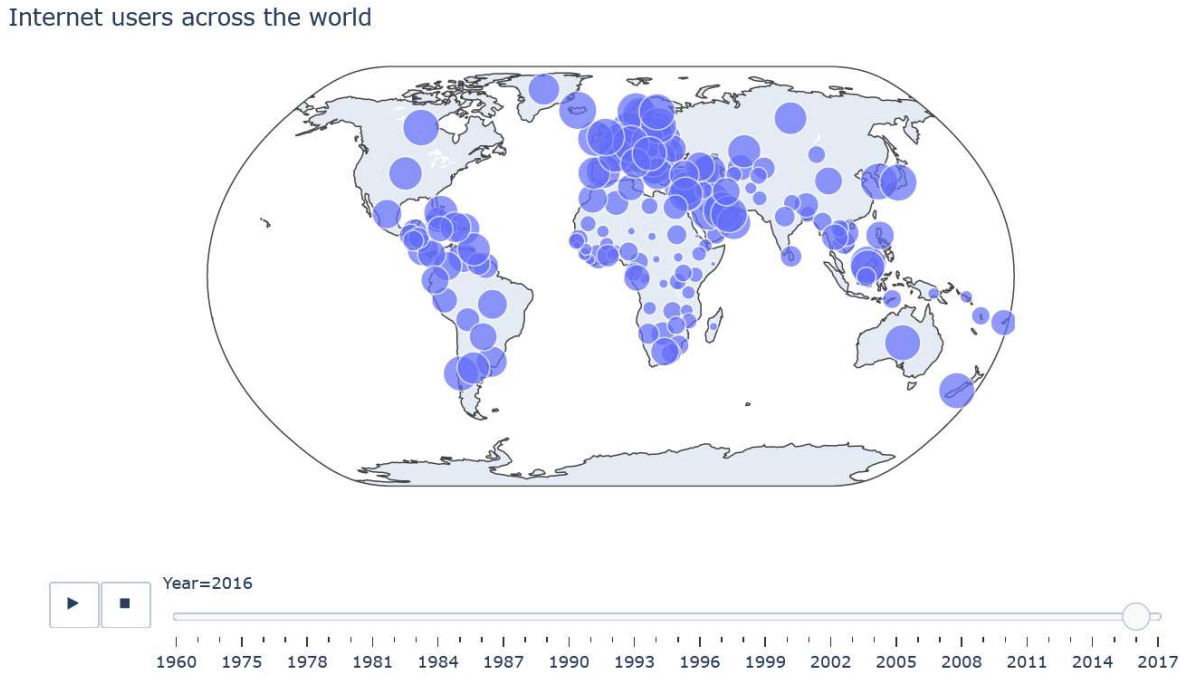

Figure 6.22b: Animated bubble plot for the year 2016

We can see from the preceding two plots how the number of internet users increased between **2001** and **2016**.

Scatter plots on maps can be used to show specific locations of interest on geographical maps, whereas bubble maps are a nice way to present count data across different divisions of a geographical region. The **Scattergeo** function from **plotly graph_objects** and the **scatter_geo**

function from **plotly express** are generally used to generate interactive scatter plots and bubble plots on maps.

In the next section, we'll look at a few line plots on geographical maps.

Line Plots on Geographical Maps

Line plots rendered on maps are another important class of visualization for geographical data.

For this section, we will be using the airport and flight data from the **2015** Flight Delays and Cancellations dataset released by the **U.S. Department of Transportation**'s (**DOT**) Bureau of Transportation Statistics. Since the dataset is huge, we will only include the data for all flights with airline delays on **Jan 1,2015**. This reduced dataset contains the records of **1,820** flights and is made available on Moodle as two files:

airports.csv: Contains location attributes such as latitude and longitude information for all airports

new_year_day_2015_delayed_flights.csv: Contains flight details such as flight numbers, origin, and destination airports for all flights in the selected subset.

Creating Line Plots on a Geographical Map

1. In this exercise, we'll use the **airports** dataset and first generate a scatter plot to find out the locations of all airports in the US. We'll then merge the two DataFrames (**flights** and **airport_record**) together to obtain longitude and latitudes for the origin airports of all flights and draw line plots from the origin airport to the destination airport for each flight using this merged dataset. Let's go through the following steps:
2. Load the **airports** dataset first:

   ```
   import pandas as pd

   us_airports_url = "data/airports.csv"

   us_airports_df = pd.read_csv(us_airports_url)

   us_airports_df.head()
   ```

   The output is as follows:

	IATA_CODE	AIRPORT	CITY	STATE	COUNTRY	LATITUDE	LONGITUDE
0	ABE	Lehigh Valley International Airport	Allentown	PA	USA	40.65236	-75.44040
1	ABI	Abilene Regional Airport	Abilene	TX	USA	32.41132	-99.68190
2	ABQ	Albuquerque International Sunport	Albuquerque	NM	USA	35.04022	-106.60919
3	ABR	Aberdeen Regional Airport	Aberdeen	SD	USA	45.44906	-98.42183
4	ABY	Southwest Georgia Regional Airport	Albany	GA	USA	31.53552	-84.19447

Figure 6.23:Airports dataset

3. Generate a scatter plot on the US map to indicate the locations of all airports in our dataset, using the **graph_objects** module:

```
import plotly.graph_objects as go

fig = go.Figure()

fig.add_trace(go.Scattergeo(

    locationmode = 'USA-states',

    lon = us_airports_df['LONGITUDE'],

    lat = us_airports_df['LATITUDE'],

    hoverinfo = 'text',

    text = us_airports_df['AIRPORT'],

    mode = 'markers',

    marker = dict(size = 5,color = 'black')))

fig.update_layout(

    title_text = 'Airports in the USA',

    showlegend = False,

    geo = go.layout.Geo(

        scope = 'usa'

    ),

)
```

fig.show()

The output is as follows:

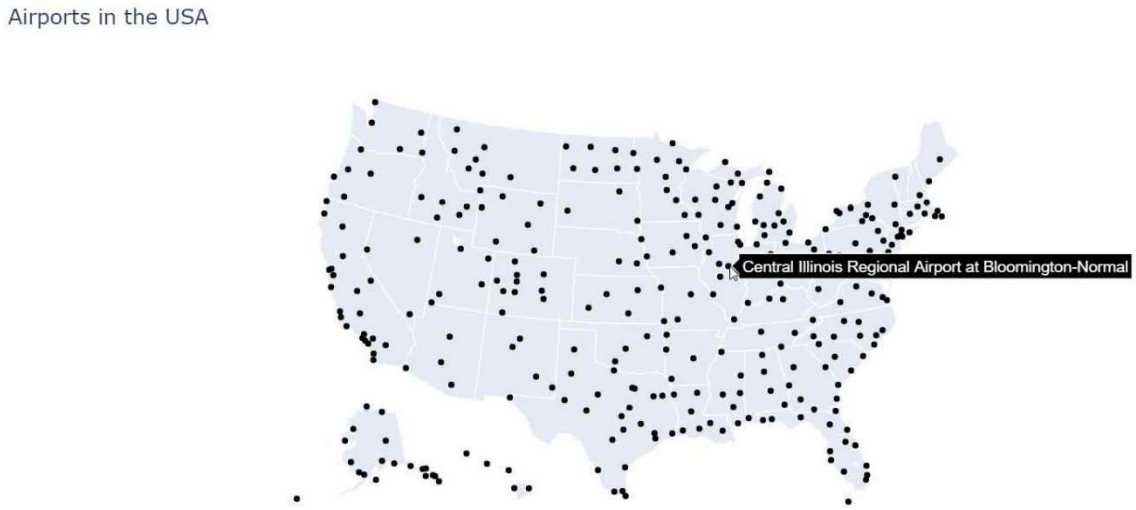

Figure 6.24: Number of airports in the US

That is neat! When you hover over a datapoint, you'll get the name of the US airport. The preceding plot shows **Central Illinois Regional Airport at Bloomington-Normal**.

Did you notice that there is an **add_trace()** function in addition to the usual instance creation of the **Scattergeo** class? The **add_trace** function is used because we are about to superimpose our flight data in the form of lines on top of this scatter plot on the map. The **add_ trace** allows **plotly** to treat the scatter plot and the line plots as multiple layers on the map.

4. Load the file containing the flight records:

new_year_2015_flights_url = "data/new_year_day_2015_delayed_flights.csv"

new_year_2015_flights_df = pd.read_csv(new_year_2015_flights_url)

new_year_2015_flights_df.head()

The output is as follows:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT
0	2015	1	1	4	HA	17	N389HA	LAS	HNL
1	2015	1	1	4	B6	2134	N307JB	SJU	MCO
2	2015	1	1	4	B6	2276	N646JB	SJU	BDL
3	2015	1	1	4	US	425	N174US	PDX	PHX
4	2015	1	1	4	AA	89	N3KVAA	IAH	MIA

Figure 6.25: Dataset with flight records

5. Along with the origin and destination airports for each flight, we need to have the longitude and latitude information of the corresponding airports. To do this, we need to merge the DataFrames containing the airport and flight data. Let's first merge the two datasets to obtain the longitudes and latitudes for the origin airports of all flights:

```
# merge the DataFrames on origin airport codes

new_year_2015_flights_df =
new_year_2015_flights_df.merge(us_airports_df[['IATA_CODE','LATITUDE','LON
GITUDE']], \

                    left_on='ORIGIN_AIRPORT', \

                    right_on='IATA_CODE', \

                    how='inner')

# drop the duplicate column containing airport code

new_year_2015_flights_df.drop(columns=['IATA_CODE'],inplace=True)

# rename the latitude and longitude columns to reflect that they correspond to the
origin airport

new_year_2015_flights_df.rename(columns={"LATITUDE":"ORIGIN_AIRPORT_L
ATITUDE", "LONGITUDE":"ORIGIN_AIRPORT_LONGITUDE"},inplace=True)

new_year_2015_flights_df.head()
```

The output is as follows:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...
0	2015	1	1	4	HA	17	N389HA	LAS	HNL	145	...
1	2015	1	1	4	HA	7	N395HA	LAS	HNL	900	...
2	2015	1	1	4	AA	1623	N438AA	LAS	DFW	905	...
3	2015	1	1	4	DL	1530	N954DN	LAS	MSP	920	...
4	2015	1	1	4	WN	1170	N902WN	LAS	ELP	950	...

Figure 6.26: Dataset with flight records

6. Now, we will perform a similar merging to get the latitude and longitude data for the destination airports of all flights:

```
# merge the DataFrames on destination airport codes
```

```
new_year_2015_flights_df =
new_year_2015_flights_df.merge(us_airports_df[['IATA_CODE','LATITUDE','LON
GITUDE']], \
                    left_on='DESTINATION_AIRPORT', \
                    right_on='IATA_CODE', \
                    how='inner')

# drop the duplicate column containing airport code

new_year_2015_flights_df.drop(columns=['IATA_CODE'],inplace=True)

# rename the latitude and longitude columns to reflect that they correspond to the
destination airport

new_year_2015_flights_df.rename(columns={'LATITUDE':'DESTINATION_AIRP
ORT_LATITUDE',
'LONGITUDE':'DESTINATION_AIRPORT_LONGITUDE'},inplace=True)

new_year_2015_flights_df.head()
```

The output is as follows:

	YEAR	MONTH	DAY	DAY_OF_WEEK	AIRLINE	FLIGHT_NUMBER	TAIL_NUMBER	ORIGIN_AIRPORT	DESTINATION_AIRPORT	SCHEDULED_DEPARTURE	...
0	2015	1	1	4	HA	17	N389HA	LAS	HNL	145	...
1	2015	1	1	4	HA	7	N395HA	LAS	HNL	900	...
2	2015	1	1	4	UA	253	N768UA	IAH	HNL	920	...
3	2015	1	1	4	UA	328	N210UA	DEN	HNL	1130	...
4	2015	1	1	4	UA	1173	N56859	SFO	HNL	805	...

5 rows × 35 columns

Figure 6.27: Merged flights dataset

7. Now, we will draw our line plots. For each flight, we need to draw a line between the origin airport and the destination airport. This is done by providing the latitude and longitude values of the destination and origin airports to the **lon** and **lat** parameters of **Scattergeo** and setting **mode** to *lines* instead of *markers*. Also, notice that we are using another **add_trace** function here. It may take a few minutes for the plot to show the flight routes:

```
for i in range(len(new_year_2015_flights_df)):
    fig.add_trace(
        go.Scattergeo(
```

```
            locationmode = 'USA-states',

            lon = [new_year_2015_flights_df['ORIGIN_AIRPORT_LONGITUDE'][i],
new_year_2015_flights_df['DESTINATION_AIRPORT_LONGITUDE'][i]],

            lat = [new_year_2015_flights_df['ORIGIN_AIRPORT_LATITUDE'][i],
new_year_2015_flights_df['DESTINATION_AIRPORT_LATITUDE'][i]],

            mode = 'lines',

            line = dict(width = 1,color = 'red')

        )

    )

fig.update_layout(

    title_text = 'Delayed flight on Jan 1, 2015 in USA',

    showlegend = False,

    geo = go.layout.Geo(

        scope = 'usa'

    ),

)

fig.show()
```

The output is as follows:

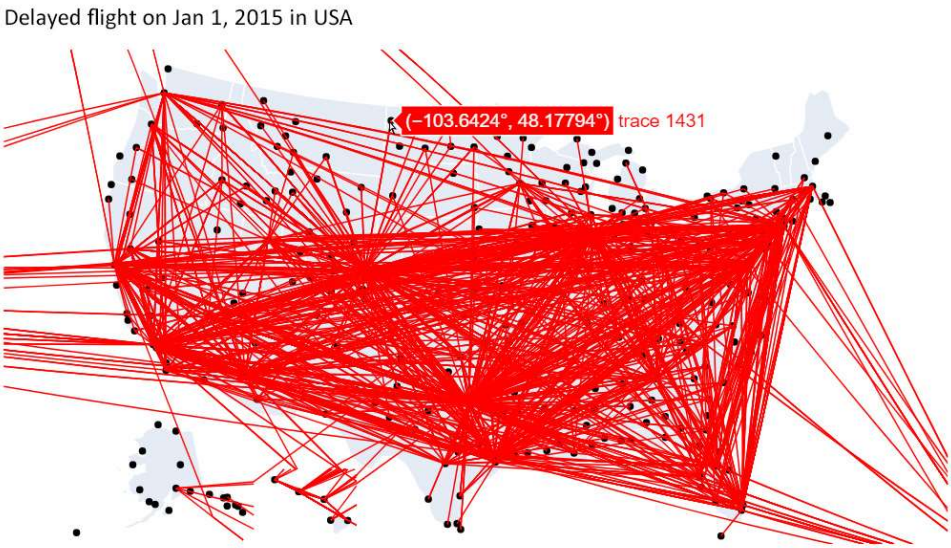

Figure 6.28: Line plot for all the delayed flights

Line plots on geographical maps can be generated using the **graph_objects** module from **plotly**. Generally, a layering technique is used, with the help of the **add_trace()** function to superimpose two plots together on the map – the locations being connected as a scatter plot, and the routes connecting various locations as line plots.