Interactive Visualization with R - Interactive Maps - 1

One should look for what is and not what he thinks should be. (Albert Einstein)

Interactive maps: topic introduction

In this part of the course, we will cover the following concepts:

- Transform and prepare data for maps
- Create maps and display spatial data over time

Warm-Up

- Take a few minutes to craft a “tweet” of about 140 characters that either:
  - poses a question about something we’ve discussed
  - sums up something valuable you’ve learned
  - offers a resource you’ve discovered on your own
  - asks for additional information
- When you’re done, post your tweet in the chat so we can pick out a few to discuss in more detail

Module completion checklist

Objective	Complete
Create interactive maps utilizing JSON files	
Illustrate adding motion to maps to display spatial data over time	
Discuss best practices for highcharter maps	

Why interactive maps?

- Visualizing spatial data can uncover the interaction between variables and geographic locations

- **Interactive maps** are helpful in visualizing spatial data because of the following:
  - The ability to create layers of information
  - Zoom functions and tooltips to show details of a specific point or area
  - Animations to show the effect of time

Process of creating interactive maps

Set up

1. Specify data
2. Link data to visuals
3. Assign shapes

Adjust

1. Visual effects
2. Interactive effects
3. Legend

Polish

1. Customize theme
2. Map layers
3. Text

Directory settings

- In order to maximize the efficiency of your workflow, use the `box` package and encode your directory structure into `variables`
- Let the `main_dir` be the variable corresponding to your materials folder

```
# Set `main_dir` to the location of your materials folder.

path = box::file()
main_dir = dirname(dirname(path))
```

Directory settings (cont'd)

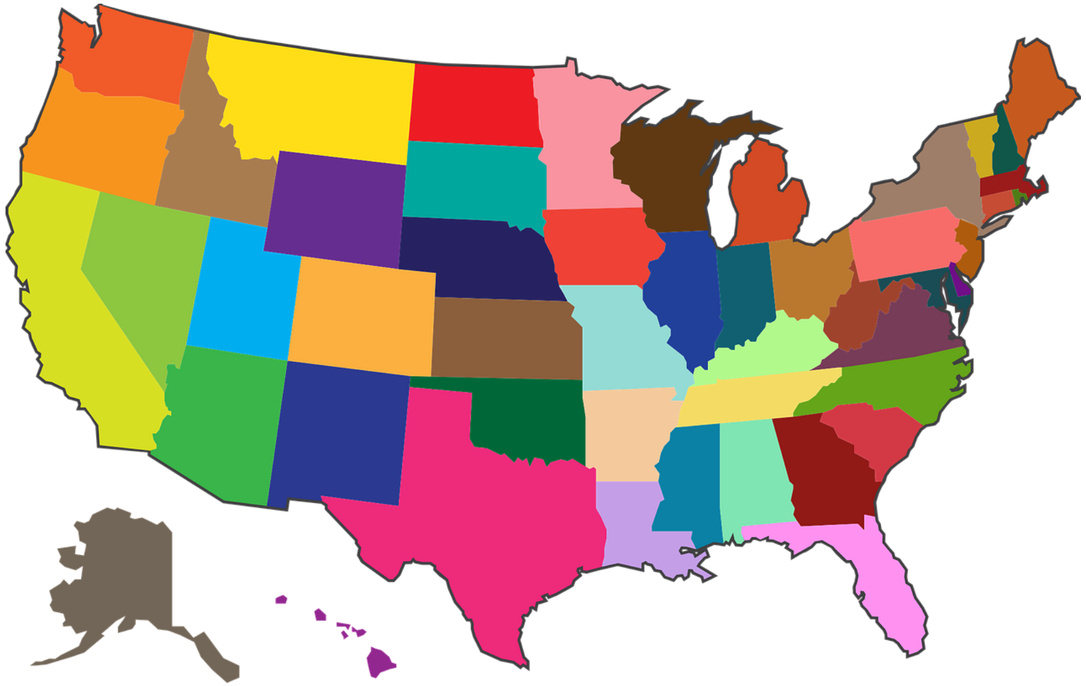
- We will store all datasets in the `data` directory inside the materials folder in your environment; hence we will save their path to a `data_dir` variable
- We will save all the plots in the `plots` directory corresponding to `plot_dir` variable

- To append one string to another, use `paste0` command and pass the strings you would like to paste together

```
# Make `data_dir` from the `main_dir` and
# remainder of the path to data directory.
data_dir = paste0(main_dir, "/data")
# Make `plots_dir` from the `main_dir` and
# remainder of the path to plots directory.
plot_dir = paste0(main_dir, "/plots")
```

Set up: `state.x77` data

- For the non-spatial attributes, we will use a dataset containing facts and figures about **U.S. states in 1975**, which can be found in R `datasets` library
- For ease of usage, we have stored the dataset in the data folder and we will be loading it directly from the data folder



```
# We will load the formatted dataset from a csv
file.

# Load the dataset.
state_df = read.csv(file =
file.path(data_dir,"/state_data.csv"), #<-
provide file path
                header = TRUE,             #<- if
file has header set to TRUE
                stringsAsFactors = FALSE) #<- read
strings as characters, not as factors
```

Set up: `state.x77` data

- Let's see what variables are represented in the data

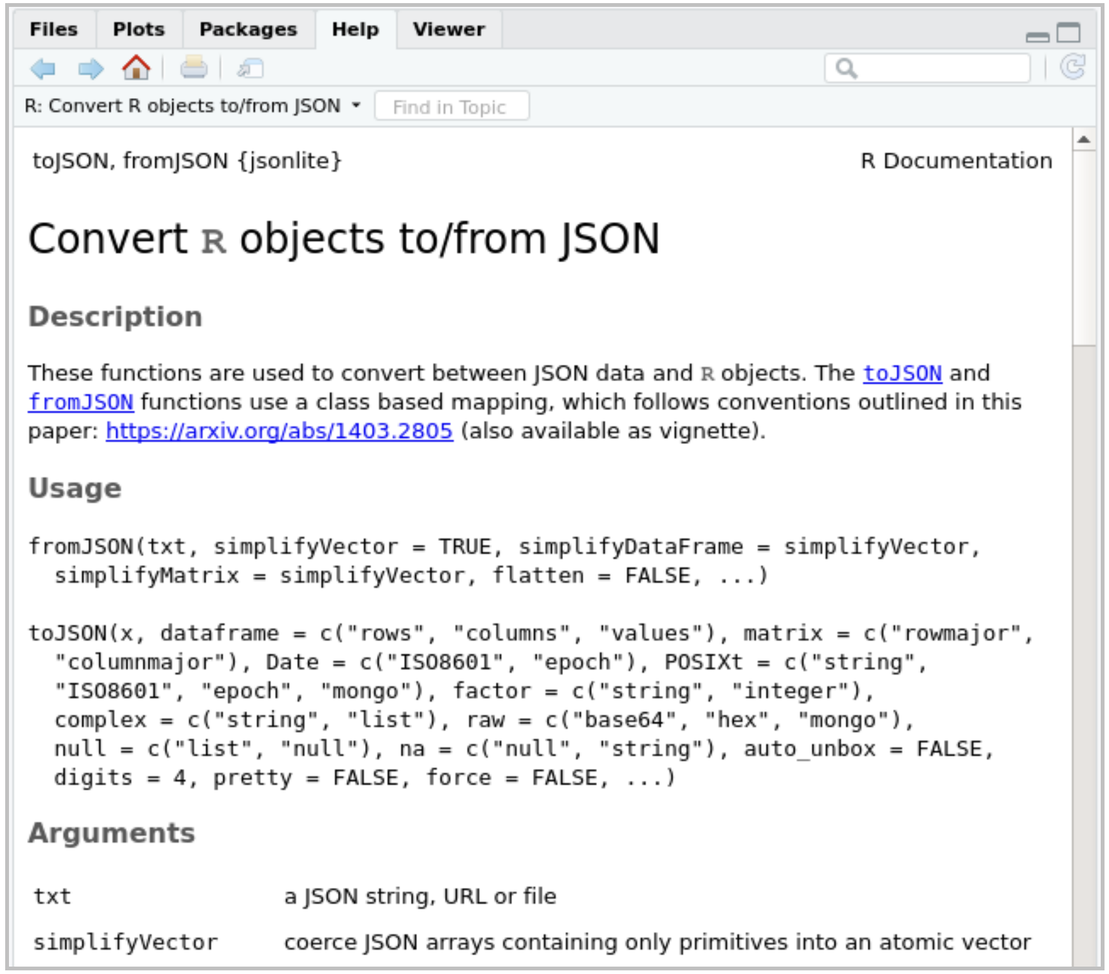
```
# View dataset.
str(state_df)
```

```
'data.frame':   50 obs. of  10 variables:
 $ Population: int  3615 365 2212 2110 21198 2541 3100 579 8277 4931 ...
 $ Income    : int  3624 6315 4530 3378 5114 4884 5348 4809 4815 4091 ...
 $ Illiteracy: num  2.1 1.5 1.8 1.9 1.1 0.7 1.1 0.9 1.3 2 ...
 $ Life.Exp  : num  69 69.3 70.5 70.7 71.7 ...
 $ Murder    : num  15.1 11.3 7.8 10.1 10.3 6.8 3.1 6.2 10.7 13.9 ...
 $ HS.Grad   : num  41.3 66.7 58.1 39.9 62.6 63.9 56 54.6 52.6 40.6 ...
 $ Frost     : int  20 152 15 65 20 166 139 103 11 60 ...
 $ Area      : int  50708 566432 113417 51945 156361 103766 4862 1982 54090 58073 ...
 $ State     : chr  "Alabama" "Alaska" "Arizona" "Arkansas" ...
 $ code      : chr  "AL" "AK" "AZ" "AR" ...
```

Set up: working with GEO data and JSON files

- `geoJSON` is a **JavaScript Object Notation** format representing simple geographical features and non-spatial attributes
- `Highcharts` has a ***viewable collection of geoJSON files*** covering most areas of the world
- You can reference these by link or download and load them using the `jsonlite` package

```
# Load the library.
library(jsonlite)

# View documentation.
library(help = "jsonlite")

?fromJSON
```



Set up: working with GEO data and JSON files

- We need to give the function a geoJSON file with the spatial data as the first argument
- Then we need to tell the function **not** to simplify any vectors
  - This setting is necessary for correct object translation into the map data

```
library(highcharter)
```

```
Registered S3 method overwritten by 'quantmod':
  method            from
  as.zoo.data.frame zoo
```

```
# Set working directory to data folder.
setwd(data_dir)

# Read data from JSON file, don't simplify vectors.
US_map = fromJSON("us-all.geo.json", simplifyVector = FALSE)
```

Set up: working with GEO data and JSON files

- The `hc_middle_x` and `hc_middle-y` coordinates will be used for plotting
- The `name` and `postal-code` will be used to create and label the states in the map

```
# To see what metadata is available in the `geo.json`, use `get_data_from_map` function.
geodata = get_data_from_map(US_map)

# Look at only 15 first columns
str(geodata[,1:15])
```

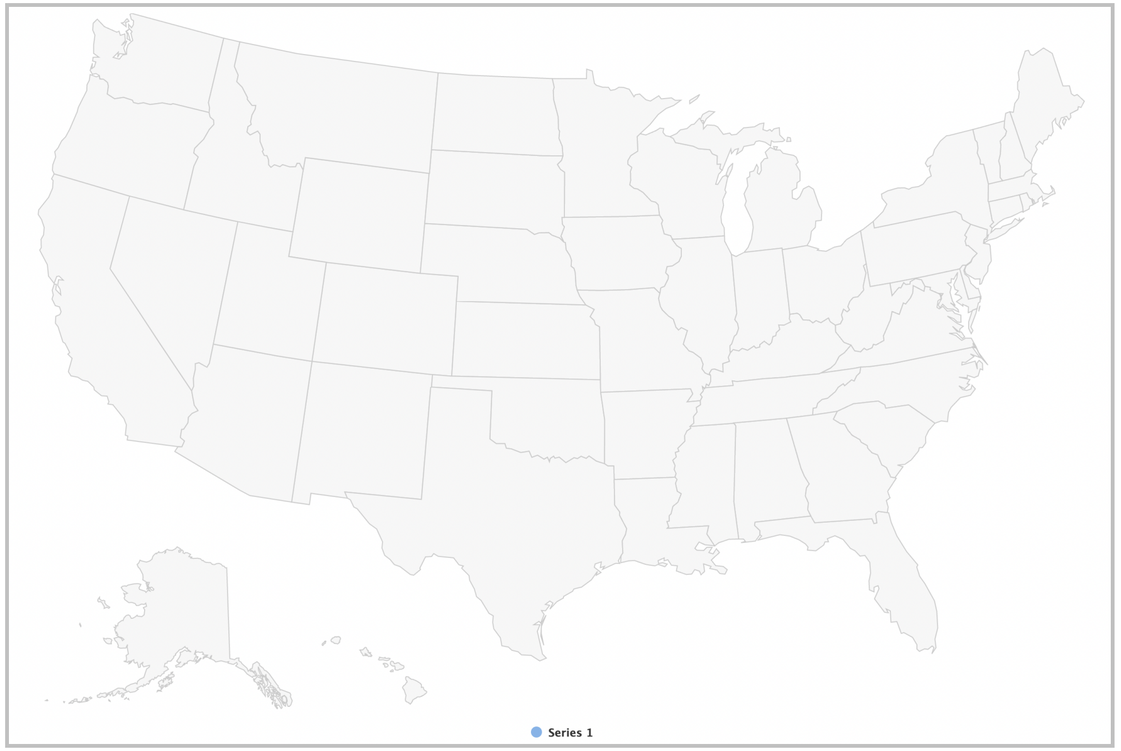
```
tibble [52 × 15] (S3: tbl_df/tbl/data.frame)
 $ hc-group   : chr [1:52] "admin1" "admin1" "admin1" "admin1" ...
 $ hc-middle-x: num [1:52] 0.36 0.56 0.51 0.47 0.41 0.43 0.71 0.46 0.51 0.51 ...
 $ hc-middle-y: num [1:52] 0.47 0.52 0.67 0.52 0.38 0.4 0.67 0.38 0.5 0.5 ...
 $ hc-key     : chr [1:52] "us-ma" "us-wa" "us-ca" "us-or" ...
 $ hc-a2      : chr [1:52] "MA" "WA" "CA" "OR" ...
 $ labelrank  : chr [1:52] "0" "0" "0" "0" ...
 $ hasc       : chr [1:52] "US.MA" "US.WA" "US.CA" "US.OR" ...
 $ woe-id     : chr [1:52] "2347580" "2347606" "2347563" "2347596" ...
 $ state-fips : chr [1:52] "25" "53" "6" "41" ...
 $ fips       : chr [1:52] "US25" "US53" "US06" "US41" ...
 $ postal-code: chr [1:52] "MA" "WA" "CA" "OR" ...
 $ name       : chr [1:52] "Massachusetts" "Washington" "California" "Oregon" ...
 $ country    : chr [1:52] "United States of America" "United States of America" "United
States of America" "United States of America" ...
 $ region     : chr [1:52] "Northeast" "West" "West" "West" ...
 $ longitude  : chr [1:52] "-71.99930000000001" "-120.361" "-119.591" "-120.386" ...
```

Set up: creating a base map

- As always, we first create the empty base, this time of the `map` type
- We use the function `hc_add_series` to plot the base as the US map

```
# Create a base interactive map.
interactive_population_map =
  highchart(type = "map") %>%     #<- base plot
  hc_add_series(mapData = US_map)#<- map series
```

```
# This is just our base plot.
interactive_population_map
```



Set up: preparing map data

- We must join the data in `US_map` with the data in `state_df`
- This way, the plotting function knows what value must be assigned to what shape
- We will use highcharters' **`joinBy` argument**, a vector of 2 variable names, each of which should correspond to variables of the **same values in both datasets**
- In our scenario, let's use variables that identify the name of the state
- These variables are:
  - The property `name` in the `US_map`
  - The column `State` in the `state_df` dataframe

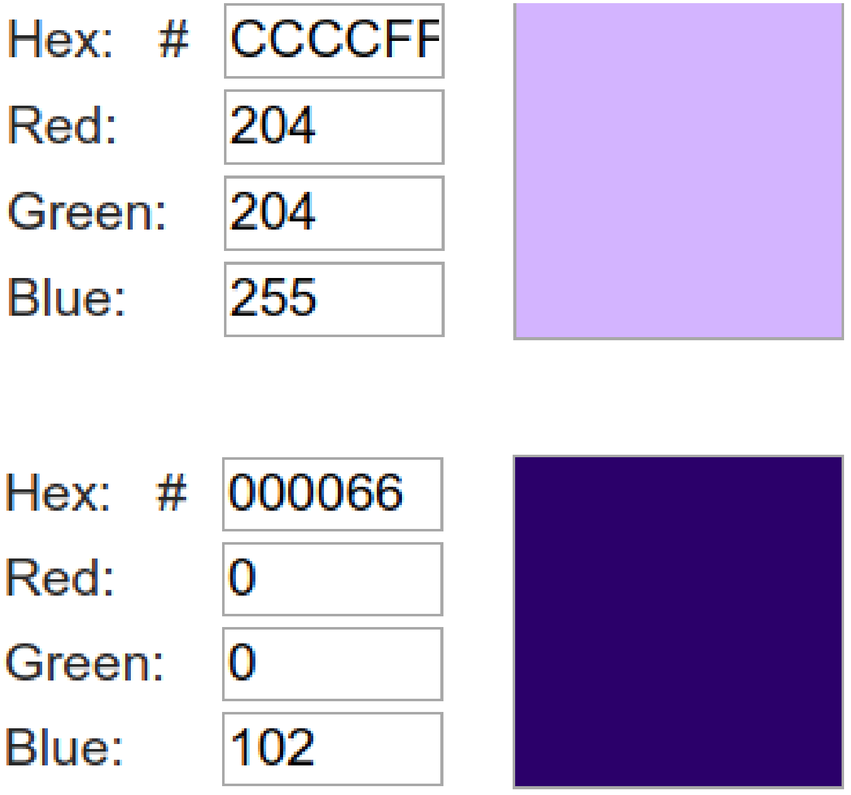
Set up: preparing map data (cont'd)

```
# Select columns to display on map.
data_for_map = select(state_df,
                      Population, #<- select `Population` to display as value on shape
                      State)      #<- select `State` to join this data with map data

# Rename columns: `Population` -> `value`, because highcharts needs a column
# called `value` to attach it to shape.
colnames(data_for_map) = c("value", "State")

# Adjust data (divide population by 1000, to make units in millions).
data_for_map$value = data_for_map$value/1000
head(data_for_map)
```

```
   value       State
1  3.615     Alabama
2  0.365      Alaska
3  2.212     Arizona
4  2.110    Arkansas
5 21.198  California
6  2.541    Colorado
```

Set up: color palettes in maps

- Use **continuous color palettes** to indicate ranges of values from small to large
- For example, we can use a range of colors generated based on the state population
  - The state with the smallest population will be the lightest
  - The state with the highest population will be the **darkest**
- Color codes (both `RGB` and `hex`) can be found on ***this link***
- Color palettes can also be created using the `color_stops()` function



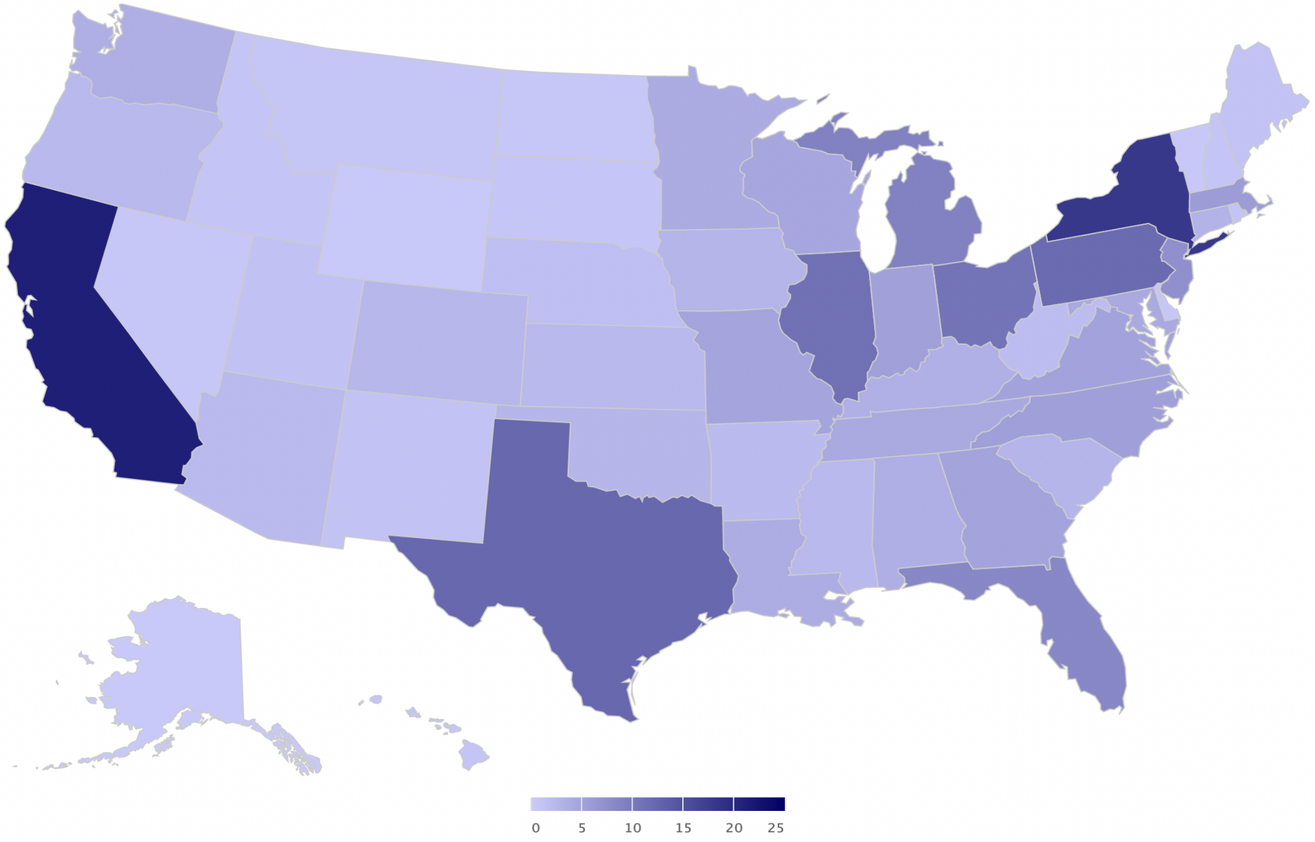
Set up: creating an interactive population map

- Let's combine our input data files and color palette parameters to create an interactive map
- We will use the `data` argument in the `hc_add_series` function to plot the columns from the cleaned data
- We use the function `hc_colorAxis` to add the color gradient for representing the population in different states

```
interactive_population_map =
  highchart(type = "map") %>%
  hc_add_series(mapData = US_map,
                data = data_for_map,                  #<- data to plot on shapes
                name = "Population in 1975",          #<- series name is `Population`
                joinBy = c("name",                    #<- join by `name` property in `mapData`
                           "State") ) %>%             #<- with `State` column in `data`
  hc_colorAxis(min = min(data_for_map$value),         #<- set colors: min value => minimum population
               max = max(data_for_map$value),         #<- max value => maximum population
               minColor = "#CCCCFF",                  #<- min value color
               maxColor = "#000066")                  #<- max value color
```

Set up: creating an interactive population map

```
interactive_population_map
```



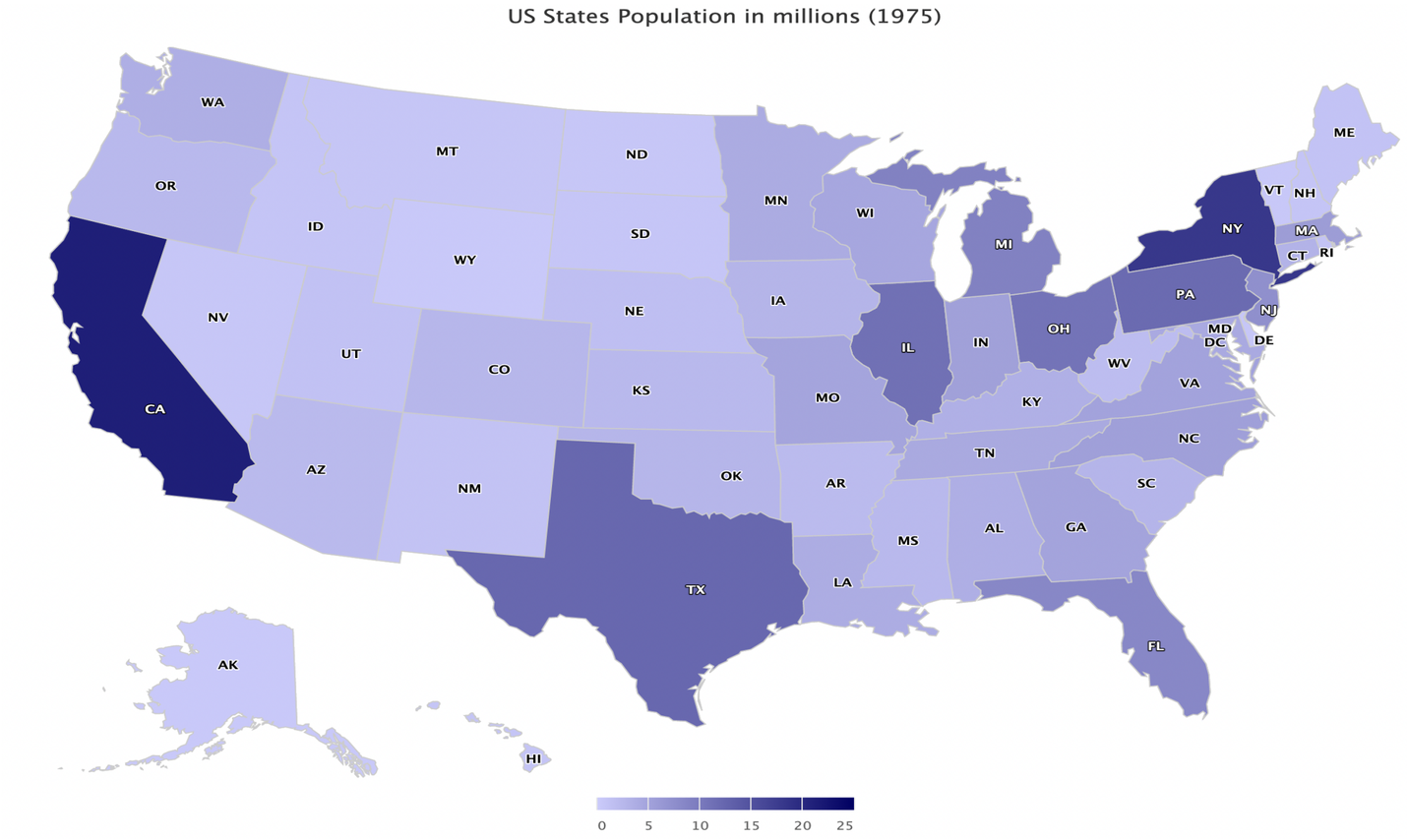
Adjust: creating an interactive population map

- Let’s add some details to the previous map to improve its functionality for end users

```
interactive_population_map_adjusted = interactive_population_map %>%
    hc_add_series(mapData = US_map,
                  data = data_for_map,
                  name = "Population in 1975 ",
                  joinBy = c("name",
                             "State"),
                  dataLabels = list(enabled = TRUE,                    #<- Add labels with the
                                    format =                           #   postal code of the state
                                    '{point.properties.postal-code}')
                                    ) %>%
  hc_tooltip(valueSuffix = " million") %>%                             #<- set value suffix
  hc_mapNavigation(enabled = TRUE) %>%                                 #<- Add zoom feature
  hc_title(text = "US States Population in millions (1975)")           #<- set plot title
```

Adjust: creating an interactive population map

```
# Double click on area or use the `+` button to zoom in on an area.
# Use the `-` button to zoom out.
interactive_population_map_adjusted
```



Save interactive plots: htmlwidgets

- Finally, let's save our plot as a self-contained HTML file

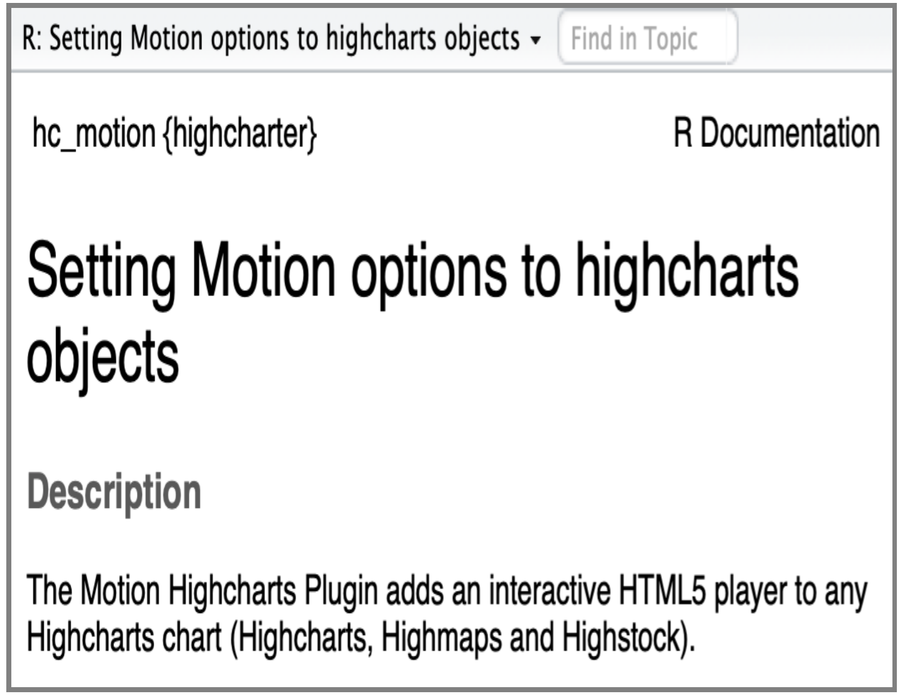
```
# Set working directory to where you save plots.
setwd(plot_dir)

# Save desired interactive plot to an HTML file.
saveWidget(interactive_population_map_adjusted,        #<- plot object to save
           "interactive_population_map.html", #<- name of file to where the plot is to be saved
           selfcontained = TRUE)              #<- set `selfcontained` to TRUE, so that
                                              #   all necessary files and scripts are embedded
                                              #   into the HTML file itself
```

Module completion checklist

Objective	Complete
Create interactive maps utilizing JSON files	✔
Illustrate adding motion to maps to display spatial data over time	
Discuss best practices for highcharter maps	

Adding motion to maps

- Motion in maps can help visualize patterns in data over time and space
- Highcharter has ***a well-documented plugin*** for adding motion to charts and maps
- This plugin can be used in highcharter with the `hc_motion()` function
- Let's practice using this function by creating a visualization based on the change in drug overdose deaths in the United States between 2002 and 2014


R: Setting Motion options to highcharts objects ▾ Find in Topic

hc_motion {highcharter} R Documentation

Setting Motion options to highcharts objects

Description

The Motion Highcharts Plugin adds an interactive HTML5 player to any Highcharts chart (Highcharts, Highmaps and Highstock).

Reading in the CDC dataset

- We will use a dataset from the ***Center for Disease Control and Prevention*** with information on drug overdose deaths from 2002 to 2014 in the U.S.

```
# Load the library.
library(highcharter)
```

```
# Set working directory to data_dir
setwd(data_dir)

# We can also load non-spatial data from a JSON file.
data_for_map = fromJSON("drug_overdose.json")
head(data_for_map)
```

```
   fips year value
1 01001 2002     1
2 01003 2002     2
3 01005 2002     0
4 01007 2002     1
5 01009 2002     2
6 01011 2002     0
```

- **fips:** numeric codes to uniquely identify a county in the U.S.
- **year:** year for which data was collected
- **value:** number of deaths due to drug overdose

Preparing the data

- Let's create a list containing a data series for each year based on three sets of values from the dataset:
  - **fips:** we will use this to join with the geoJSON data to create the map
  - **sequence:** a list of values in the given state for each year
  - **value:** the first value in the list to initialize the map

```
ds <- data_for_map %>%
      group_by(fips) %>%                          #<- group by state
      do(state_deaths = list(                     #<- go through each state
         fips = first(.$fips),                    #<- select the FIPS for that state
         sequence = .$value,                      #<- create a list of values for that state
         value = first(.$value))) %>%             #<- select the first value to initialize the map
      .$state_deaths                              #<- put the three together in a list
`state_deaths`
```

Adding motion to maps

The `hc_motion` function is commonly used to create dynamic and animated charts in highcharter, allowing you to visualize data changes over time or other sequential steps.

- **Parameters**:
  i. `enabled` (boolean): Enables or disables motion in the plot.
  ii. `axisLabel` (string): Sets the label for the motion slider.
  iii. `labels` (vector): Labels the animation steps (e.g., by year or time period).

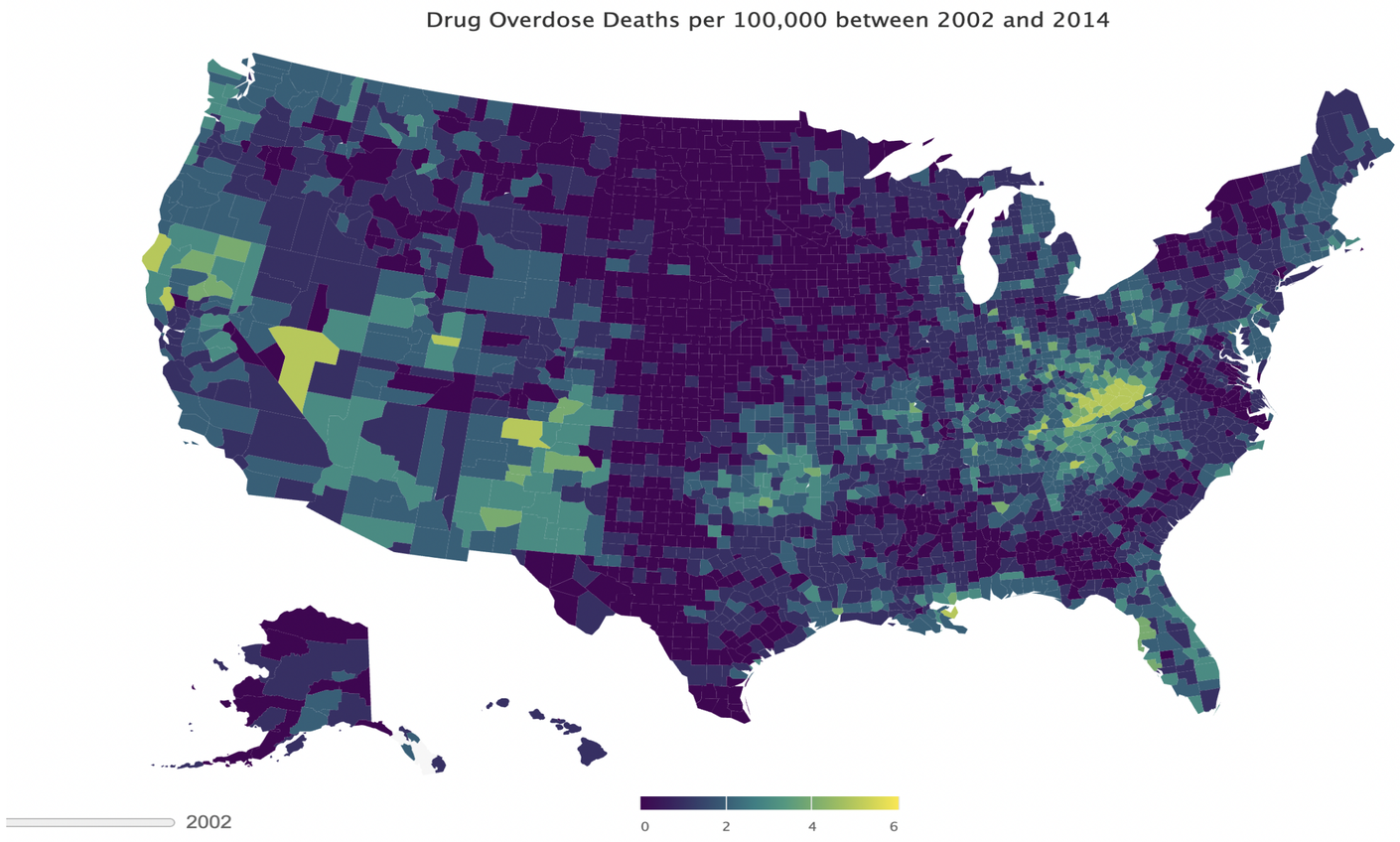
Adding motion to maps (cont'd)

- Now we can add the interactivity and motion elements to the base map plot

```
interactive_map_motion <- highchart(type = "map") %>%
  hc_add_series(data = ds,                                #<- data to plot on shapes
                name = "Drug deaths per 100,000",         #<- series name
                mapData = uscountygeojson,                #<- map data from `highcharter`
                joinBy = "fips",                          #<- join by `fips` field in `data` and
`mapData`
                borderWidth = 0.01) %>%                   #<- adjust border line for each shape
  hc_colorAxis(stops = color_stops()) %>%                 #<- create a color legend
  hc_title(text = "Drug Overdose Deaths per 100,000 between 2002 and 2014") %>%
  hc_motion(                                              #<- add motion
    enabled = TRUE,
    axisLabel = "year",                                   #<- name of motion slider
    labels = sort(unique(data_for_map$year)),             #<- label the animation by year
  )
```

Visualizing the interactive map

```
interactive_map_motion
```



Save interactive plots: htmlwidgets

- Remember to save your work!

```
# Set working directory to where you save plots.
setwd(plot_dir)

# Save desired interactive plot to an HTML file.
saveWidget(interactive_map_motion,         #<- plot object to save
           "interactive_map_motion.html", #<- name of file to where the plot is to be saved
           selfcontained = TRUE)              #<- set `selfcontained` to TRUE, so that
                                              #   all necessary files and scripts are embedded
                                              #   into the HTML file itself
```

Module completion checklist

Objective	Complete
Create interactive maps utilizing JSON files	✔
Illustrate adding motion to maps to display spatial data over time	✔
Discuss best practices for highcharter maps	

Tips for map visualization

- **Aggregate** the data correctly for your analysis, i.e., by country, state, or city
  - Example: Aggregate sales data by summing sales for each city within a state when creating a map showing total sales by state.
- Use **gradient color scales** on maps when plotting continuous or ordinal values
  - Example: Represent population density by state using a gradient color scale, where higher densities are shown with darker shades and lower densities with lighter shades.
- **Layer** different data on the same map to see variable interactions
  - Example: Create a map displaying both population density and median income by state to analyze potential patterns or correlations between income and population distribution.

Tips for map visualization (cont'd)

- Use **motion** to add dynamic time-series data to spatial data
  - Example: Animate a map showing historical temperature data by month for different cities, illustrating temperature patterns throughout the year.
- To **declutter** a map that looks too busy:
  - eliminate or lighten unnecessary boundaries
  - use easily interpreted abbreviations wherever possible
  - use color with some degree of transparency
  - Example: When creating a map of the United States with state boundaries, decrease the opacity of state boundaries, and use common state abbreviations (e.g., "CA" for California) for improved readability.

Knowledge check

Exercise

You are now ready to try tasks 1-6 in the Exercise for this topic

Module completion checklist

Objective	Complete
Create interactive maps utilizing JSON files	✔
Illustrate adding motion to maps to display spatial data over time	✔
Discuss best practices for highcharter maps	✔

Interactive maps: topic summary

In this part of the course, we have covered:

- Transforming and preparing data for maps
- Creating maps and displaying spatial data over time

Congratulations on completing this module!