DATA SOCIETY®

Interactive visualization with R - Part 2

"One should look for what is and not what he thinks should be."
-Albert Einstein.

Warm up

Before we start, check out a couple of examples from this list of interactive maps:

https://carto.com/blog/eighty-data-visualizations-examples-using-location-data-maps/

Welcome back!

- In the last module, we covered `highcharter`'s basic capabilities for creating interactive visualizations
- Today, we will put together different types of charts in a **layered** visualization
- We will also use `highcharter` to create interactive **maps** and **visualize changes over time**

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	
Create interactive maps utilizing JSON files	
Add motion to maps to display spatial data over time	
Discuss best practices for highcharter maps	

Recap: series

- Just like `ggplot2`, the `highcharts` library has its own vocabulary
- Each new data / graphic layer in `highcharts` is called a `series`
- Each series can be a different `type`. Here are some widely-used ones:

Highcharts series type	Plot type
scatter	scatterplot
line	line graph
boxplot	boxplot
column	bar plot
bar	horizontal bar plot
histogram	histogram
area	density

Compound plots: highchart with layers

- The charts we covered in the previous module can be layered to bring out insights about the interaction of different variables.
- It's all just a big layered cake!

```
highchart() %>%             #<- main plot
  hc_chart( ... )  %>%      #<- global chart options to apply to all layers
  hc_add_series( ... )  %>% #<- plot an independent layer of data
  hc_add_series( ... )  %>% #<- plot another independent layer of data
  ...
  hc_xAxis( ... ) %>%       #<- adjust x-axis options (if necessary)
  hc_yAxis( ... ) %>%       #<- adjust y-axis options (if necessary)
  hc_tooltip( ... ) %>%     #<- adjust tooltip (if necessary)
  hc_plotOptions( ... ) %>% #<- adjust other plot options (if necessary)
  hc_legend( ... )  %>%     #<- adjust legend (if necessary)
  hc_title( ... )           #<- add/edit title (if necessary)
```

Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into variables
- Let the `main_dir` be the variable corresponding to your `skillsoft` folder on your `Desktop`

```r
# Set `main_dir` to the location of your `skillsoft` folder (for Mac/Linux).
main_dir = "~/Desktop/skillsoft"

# Set `main_dir` to the location of your `skillsoft` folder (for Windows).
main_dir = "C:/Users/[username]/Desktop/skillsoft"

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

# Do the same for your 'plot_dir' which is where your interactive plots will be stored.
plot_dir = paste0(main_dir, "/plots")
```

Loading packages

- Loading the packages we will need to use today

```r
library(htmlwidgets)
library(tidyverse)
library(highcharter)
library(broom)
library(dplyr)
library(visNetwork)
```

Set up: load & prepare data

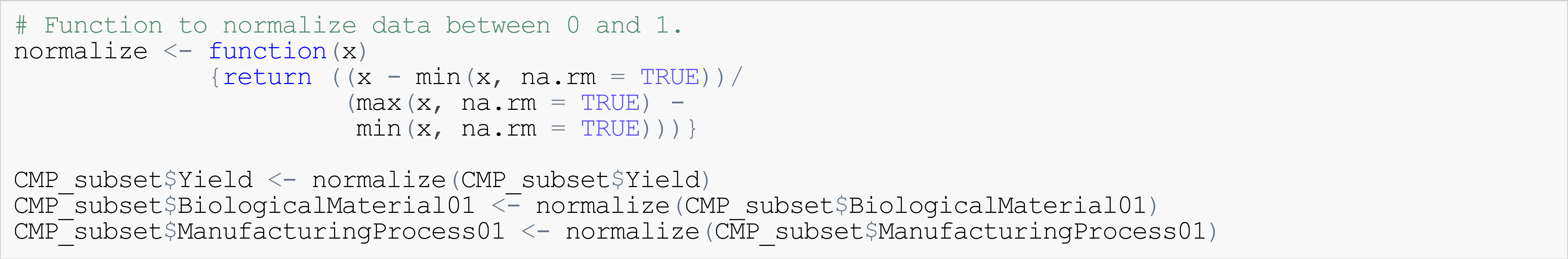
- Let's load the long CMP dataset from the previous module

```
# Set working directory to where we store data.
setwd(data_dir)

# Read CSV file
CMP = read.csv("ChemicalManufacturingProcess.csv",
               header = TRUE)

# Select only few variables from CMP data to plot
CMP_subset = select(CMP,
                    Yield,
                    BiologicalMaterial01,
                    ManufacturingProcess01)

head(CMP_subset)
```

```
  Yield BiologicalMaterial01 ManufacturingProcess01
1 38.00                 6.25                     NA
2 42.44                 8.01                    0.0
3 42.03                 8.01                    0.0
4 41.42                 8.01                    0.0
5 42.49                 7.47                   10.7
6 43.57                 6.12                   12.0
```

Compound plots: highchart with layers

- Before we build the layered plot of `Yield`, `BiologicalMaterial01`, and `ManufacturingProcess01`, we want to avoid having **different scales of data**
- We will **normalize** the variables between 0 and 1 so that their density plots can be layered on top of each other and compared meaningfully



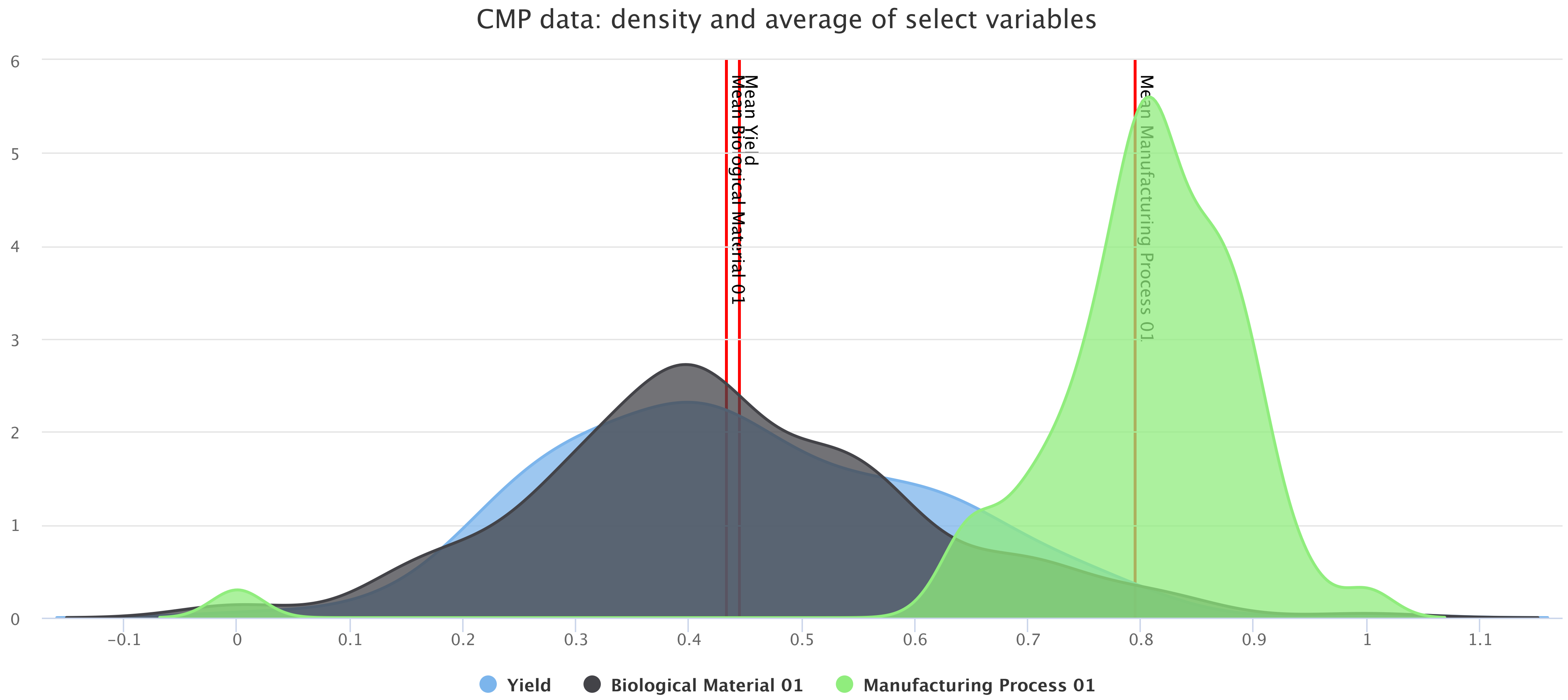
```
# Function to normalize data between 0 and 1.
normalize <- function(x)
             {return ((x - min(x, na.rm = TRUE))/
                      (max(x, na.rm = TRUE) -
                       min(x, na.rm = TRUE)))}

CMP_subset$Yield <- normalize(CMP_subset$Yield)
CMP_subset$BiologicalMaterial01 <- normalize(CMP_subset$BiologicalMaterial01)
CMP_subset$ManufacturingProcess01 <- normalize(CMP_subset$ManufacturingProcess01)
```

Compound plots: density + lines example

```
layered_density_interactive = highchart() %>%
  hc_chart(type = "area") %>%
  hc_add_series(data = density(CMP_subset$Yield, na.rm = TRUE),
                name = "Yield") %>%
  hc_add_series(data = density(CMP_subset$BiologicalMaterial01, na.rm = TRUE),
                name = "Biological Material 01") %>%
  hc_add_series(data = density(CMP_subset$ManufacturingProcess01, na.rm = TRUE),
                name = "Manufacturing Process 01") %>%
  hc_xAxis(plotLines = list(
            list(label = list(text = "Mean Yield"),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$Yield)),
            list(label = list(text = "Mean Biological Material 01"),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$BiologicalMaterial01)),
            list(label = list(text = "Mean Manufacturing Process 01"),
                  width = 2,
                  color = "red",
                  value = mean(CMP_subset$ManufacturingProcess01, na.rm = TRUE)))) %>%
  hc_tooltip(crosshairs = TRUE) %>%
  hc_title(text = "CMP data: density and average of select variables")
```

Compound plots: highchart with layers

```
layered_density_interactive
```



CMP data: density and average of select variables

Compound plots: highchart with layers

- The values of Yield and Biological Material 01 seem to have a similar distribution
- This could be useful in helping us predict Yield!
- Layering different charts and different variables is an extremely useful tool to uncover variable interactions during exploratory data analysis
- But what if we wanted to explore the interaction between Manufacturing Yield and the location of the manufacturing plant?
- For this kind of analysis, we would prefer an **interactive map**, which is what we will learn to create next!

Exercise 1

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	
Add motion to maps to display spatial data over time	
Discuss best practices for highcharter maps	

Why interactive maps?

- Spatial data is a prevalent data type, especially in the social sciences
- Visualizing spatial data can uncover interaction between variables and geographic locations
- **Interactive maps** are useful to visualize **spatial data** due to:
  - 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. Vis. effects
2. Interactive effects
3. Legend

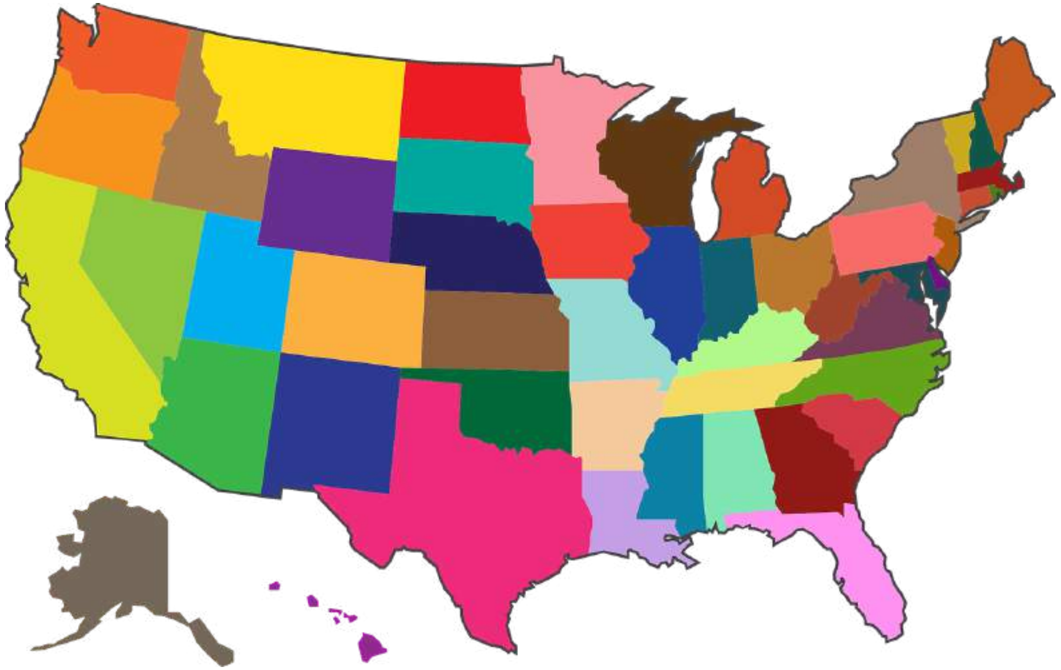
Polish

1. Customize theme
2. Map layers
3. Text

Set up: state.x77 data

For our non-spatial attributes, we are going to use a dataset that contains facts and figures about U.S. states in 1975, which can be found in R `datasets`.

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

# Set the working directory to the data
directory.
setwd(data_dir)

# Load the dataset.
state_df = read.csv("state_data.csv",
                     header = TRUE,
                     stringsAsFactors = FALSE)
```



Set up: state.x77 data

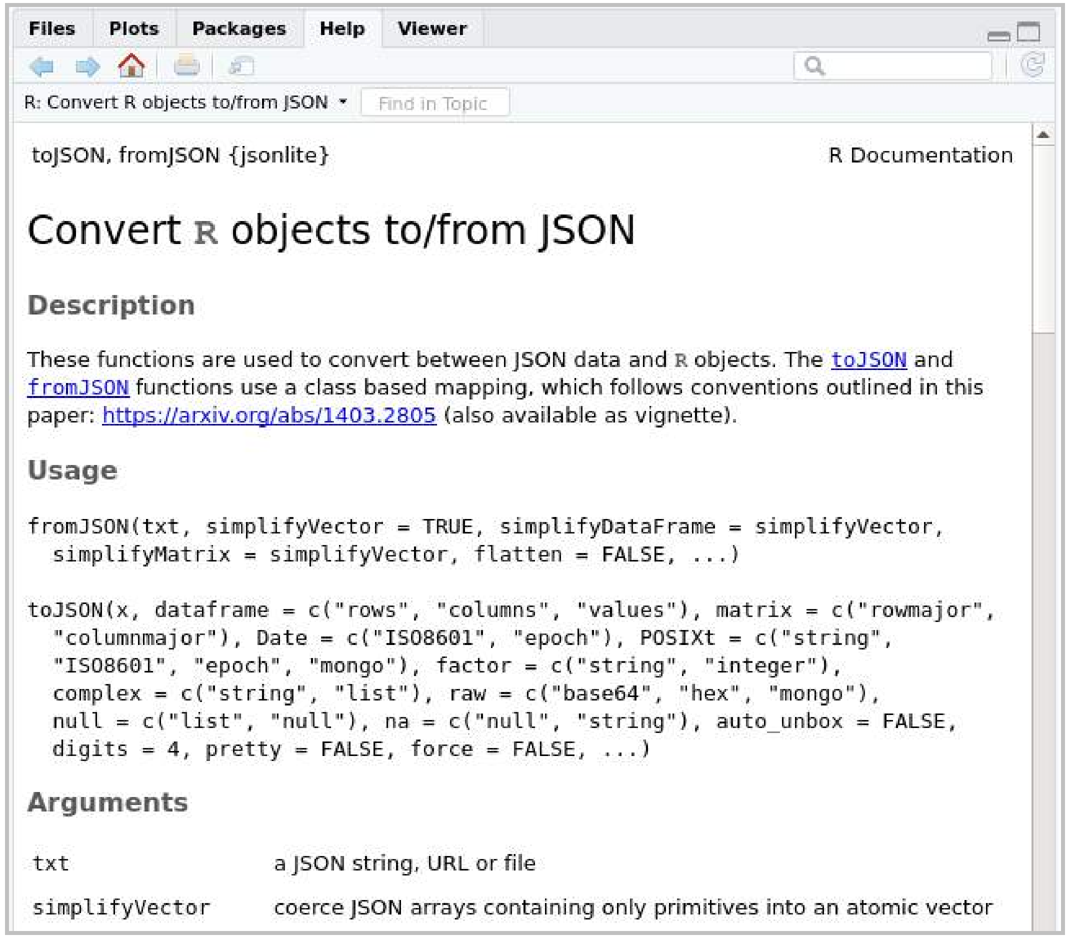
```
# View dataset.
str(state_df)
```

```
'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 for representing simple geographical features, along with their non-spatial attributes
- `Highcharts` has a ***collection of geoJSON files*** covering most areas of the world
- You can either reference these by link or download and load them using the `jsonlite` package

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

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

?fromJSON
```



toJSON, fromJSON {jsonlite} R Documentation

Convert R objects to/from JSON

Description

These functions are used to convert between JSON data and R objects. The toJSON and fromJSON functions use a class based mapping, which follows conventions outlined in this paper: https://arxiv.org/abs/1403.2805 (also available as vignette).

Usage

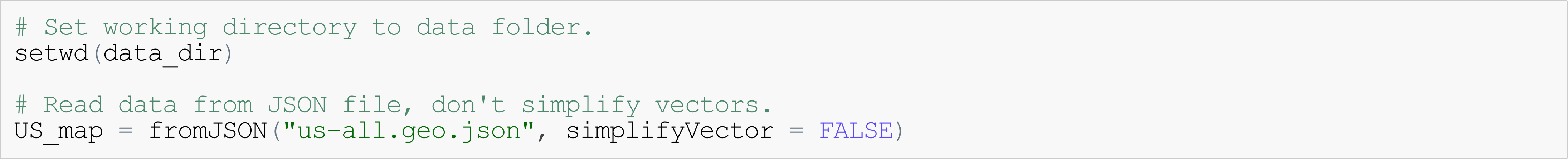
```
fromJSON(txt, simplifyVector = TRUE, simplifyDataFrame = simplifyVector,
  simplifyMatrix = simplifyVector, flatten = FALSE, ...)

toJSON(x, dataframe = c("rows", "columns", "values"), matrix = c("rowmajor",
  "columnmajor"), Date = c("ISO8601", "epoch"), POSIXt = c("string",
  "ISO8601", "epoch", "mongo"), factor = c("string", "integer"),
  complex = c("string", "list"), raw = c("base64", "hex", "mongo"),
  null = c("list", "null"), na = c("null", "string"), auto_unbox = FALSE,
  digits = 4, pretty = FALSE, force = FALSE, ...)
```

Arguments

txt	a JSON string, URL or file
simplifyVector	coerce JSON arrays containing only primitives into an atomic vector

Set up: working with GEO data and JSON files

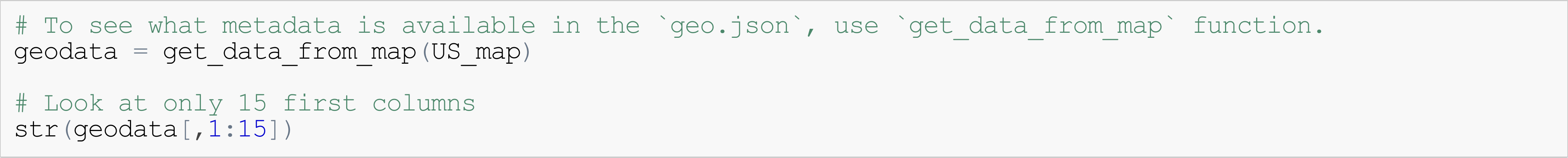
- We need to give the function a geoJSON file with the spatial data as the first argument
- We need to tell the function not to simplify any vectors (this setting is necessary for correct object translation into the map data)



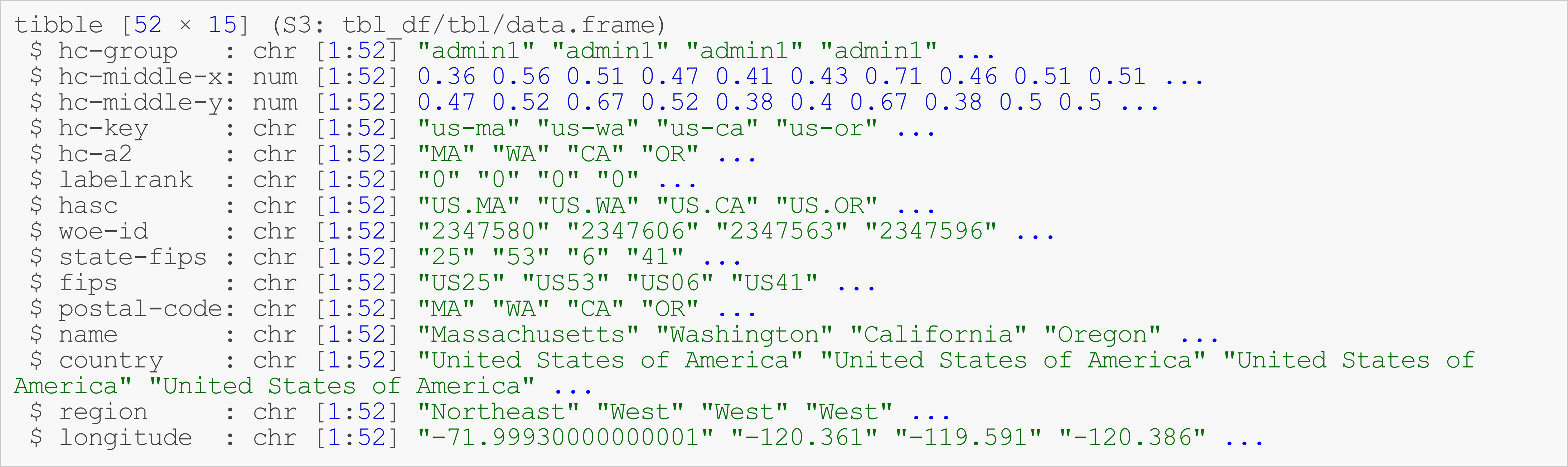
```
# Set working directory to data folder.
setwd(data_dir)

# 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])
```

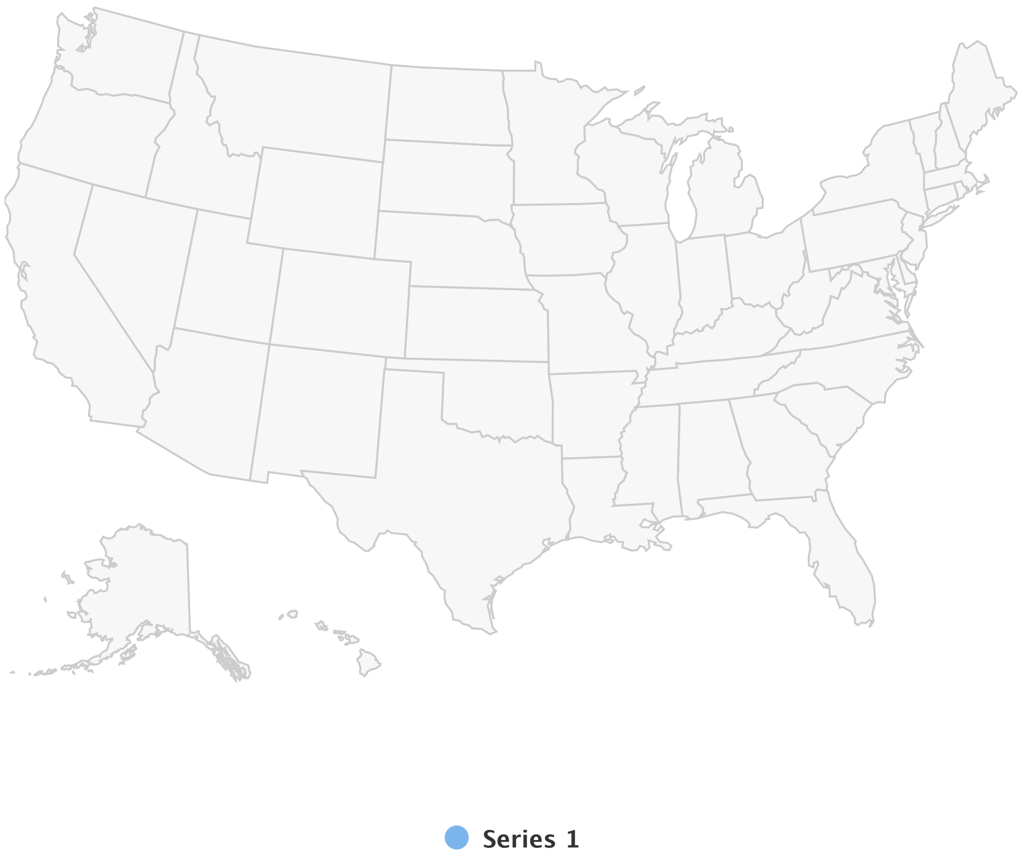


```
tibble [52 × 15] (S3: tbl_df/tbl/data.frame)
 $ hc-group   : chr [1:52] "admin1" "admin1" "admin1" "admin1" ...
 $ hc-middle-x: num [1:52] 0.36 0.56 0.51 0.47 0.41 0.43 0.71 0.46 0.51 0.51 ...
 $ hc-middle-y: num [1:52] 0.47 0.52 0.67 0.52 0.38 0.4 0.67 0.38 0.5 0.5 ...
 $ hc-key     : chr [1:52] "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

```
# 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 need to join the data in `US_map` with the data in `state_df` so that the plotting function knows what value to assign to what shape
- We will use **highchart's joinBy argument**, which is a vector of 2 variable names, each of which should correspond to **variables having 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

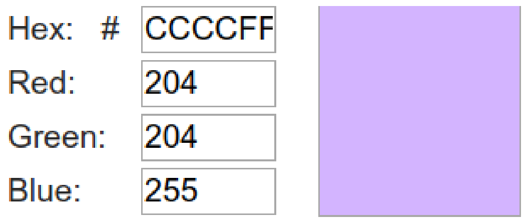
Set up: preparing map data - cont'd

```
# 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)
```

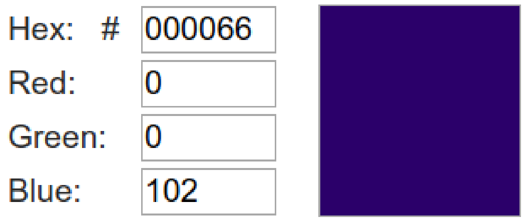
```
   value      State
1  3.615    Alabama
2  0.365     Alaska
3  2.212    Arizona
4  2.110   Arkansas
5 21.198 California
6  2.541   Colorado
```

Set up: color palettes in maps

- **Continuous color palettes** are used to indicate **ranges of values from small to large**
- We can use a **range of colors** generated **based on the state population**
  - The state with smallest population will be the **lightest**
  - The state with highest population will be the **darkest**
- Color codes (both `RGB` and `hex`) can be found on ***https://www.rapidtables.com/***
- Color palettes can also be created using the `color_stops()` function



Hex: # CCCCFF
Red: 204
Green: 204
Blue: 255



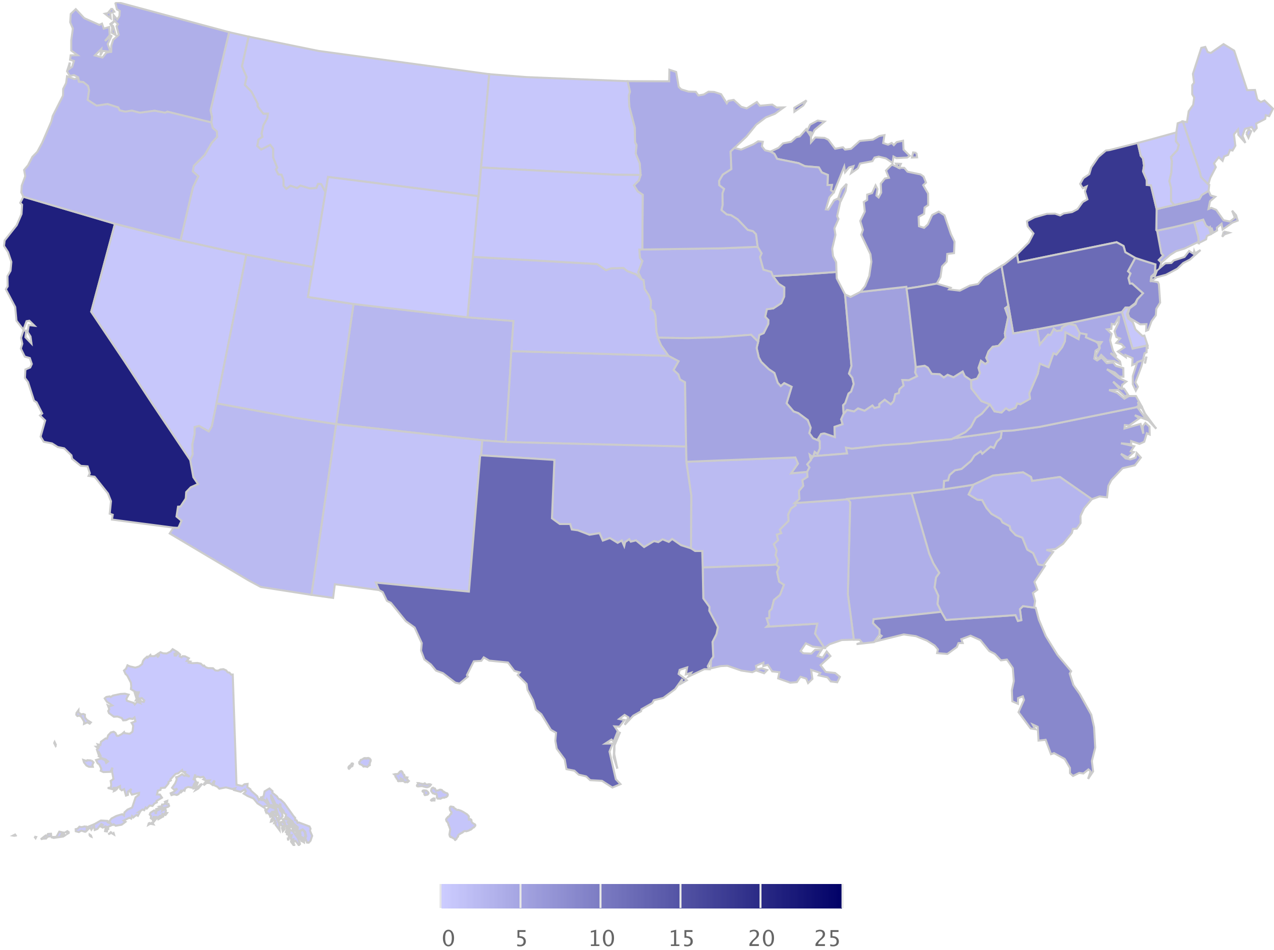
Hex: # 000066
Red: 0
Green: 0
Blue: 102

Set up: creating an interactive population map

```
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
```



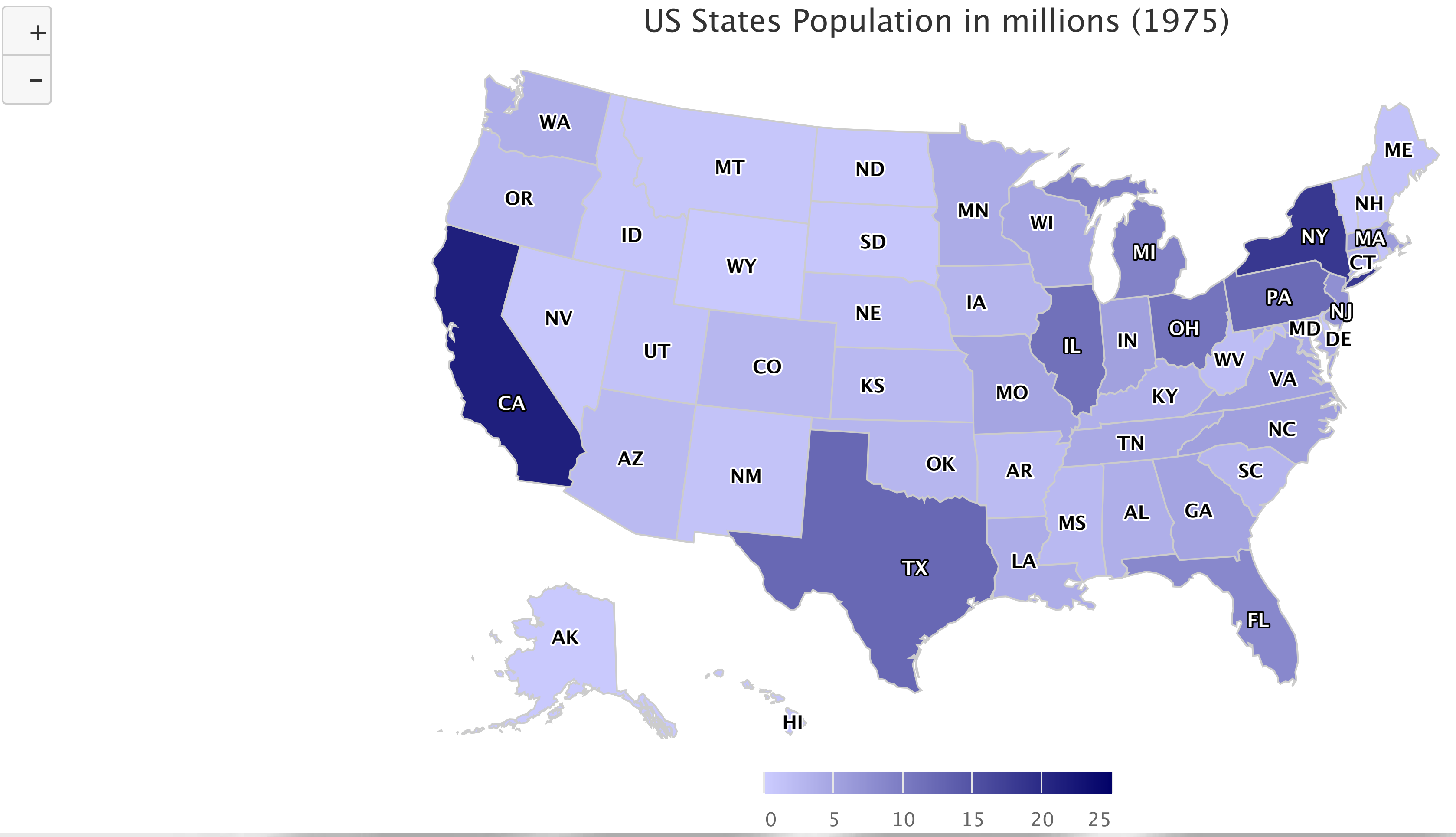
Adjust: creating an interactive population map

- Let's add some details to the previous map

```
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
```

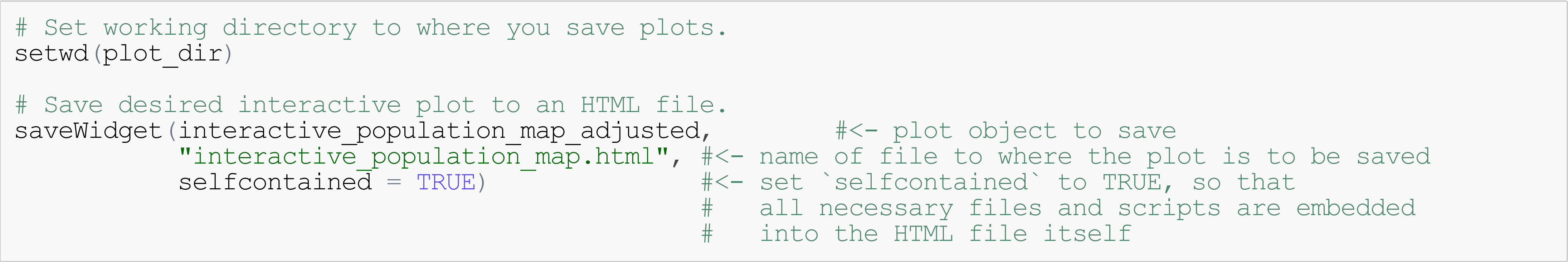
Adjust: creating an interactive population map

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



US States Population in millions (1975)

Save interactive plots: htmlwidgets



```
# 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
```

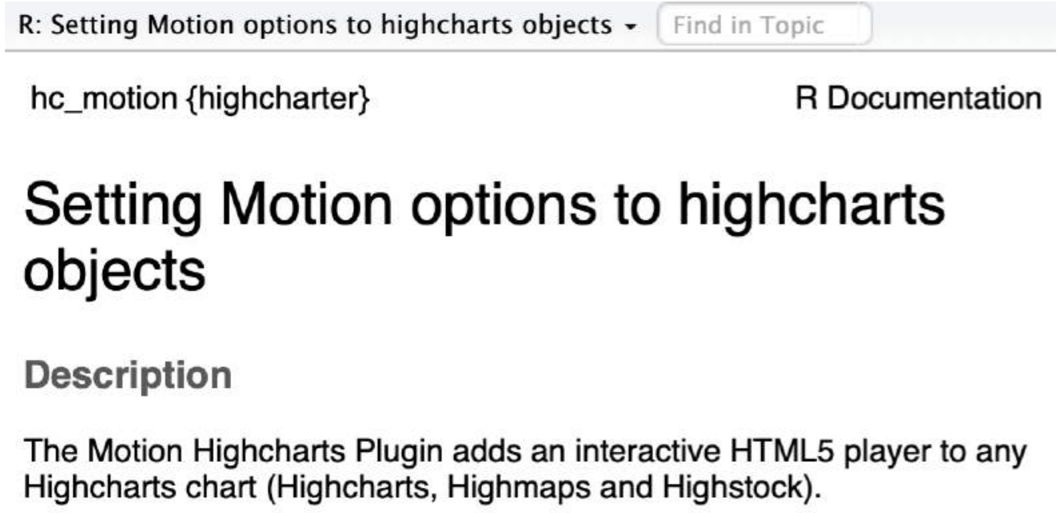
Knowledge check 1

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	✔
Add motion to maps to display spatial data over time	
Discuss best practices for highcharter maps	

Adding motion to maps

- Motion in maps can be useful to visualize patterns in data over time and space
- `Highcharts` has ***a plug-in*** for adding motion to charts and maps
- This plug-in can be used in `highcharter` with the `hc_motion` function



hc_motion {highcharter} R Documentation

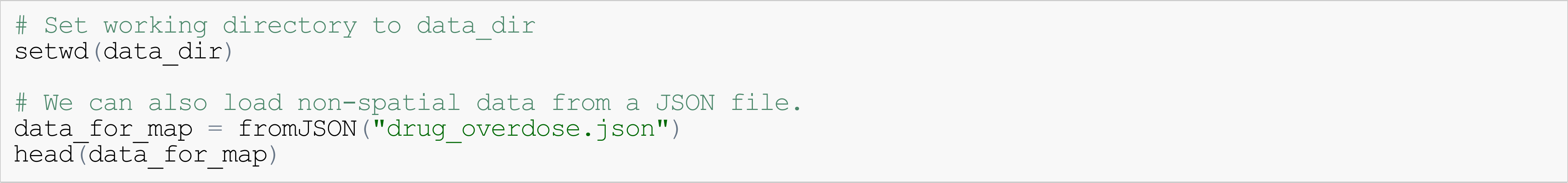
Setting Motion options to highcharts objects

Description

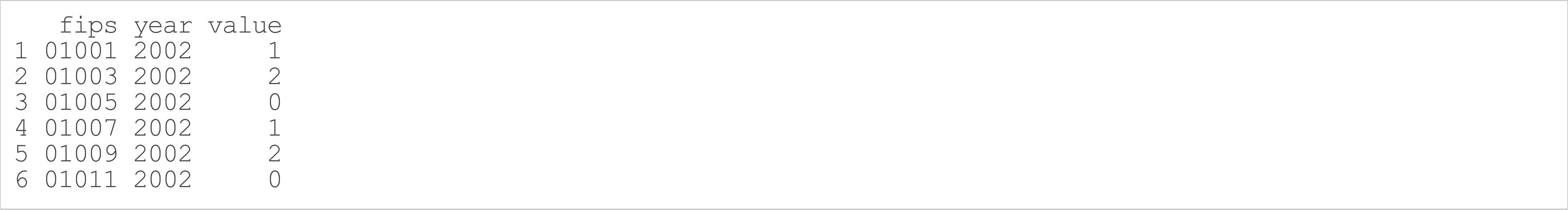
The Motion Highcharts Plugin adds an interactive HTML5 player to any Highcharts chart (Highcharts, Highmaps and Highstock).

Preparing the data: adding motion to maps

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



```r
# 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 U.S. county
- **year:** year for which data was collected
- **value:** number of deaths due to drug overdose in a particular year in a particular county

Preparing the data: adding motion to maps

- We will create a list containing a data series for each year containing three things:
  - **FIPS:** this will be used 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

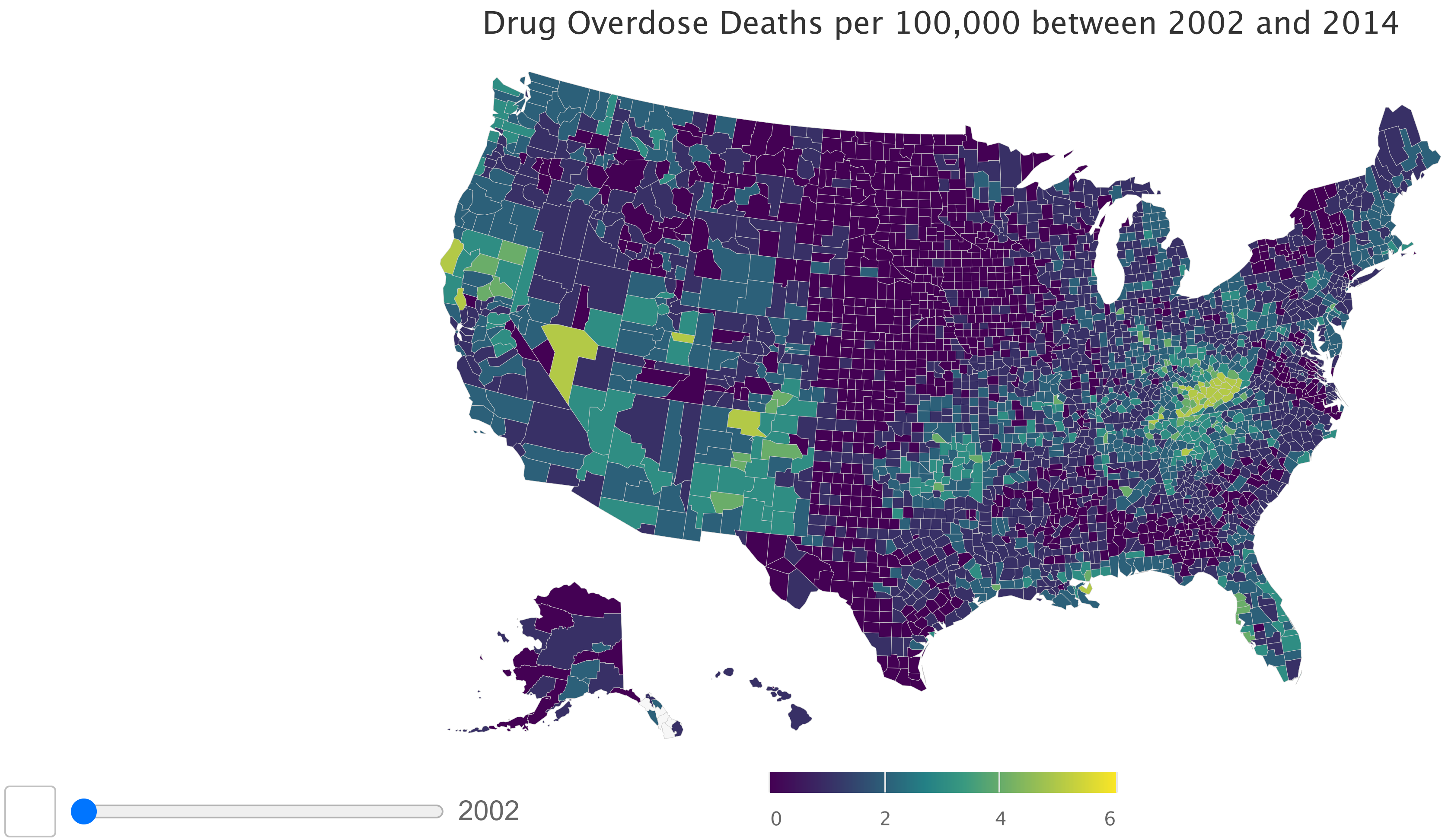
```
ds <- data_for_map %>%
      group_by(fips) %>%                         #<- group by state
      do(state_deaths = list(                    #<- go through each state
         fips = first(.$fips),                   #<- select the FIPS for that state
         sequence = .$value,                     #<- 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

```r
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
  )
```

Set up: adding motion to maps

```
interactive_map_motion
```



Drug Overdose Deaths per 100,000 between 2002 and 2014

Save interactive plots: htmlwidgets

```r
# 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 visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	✔
Add motion to maps to display spatial data over time	✔
Discuss best practices for highcharter maps	

Tips for map visualization

- Aggregated data at the correct level for your analysis (country vs state vs city)
- Maps can easily look cluttered, so:
  - eliminate or lighten unnecessary boundaries
  - use abbreviations wherever possible–as long as they are recognizable
  - use color with transparency
- Use gradient color scales on the maps when plotting ordinal values
- Layer different data on the same map to get interesting variable interactions
- Use motion to add time-series spatial data to your maps

Knowledge check 2

Exercise 2

Module completion checklist

Objective	Complete
Create interactive visualizations with transformed summary data	✔
Create interactive maps utilizing JSON files	✔
Add motion to maps to display spatial data over time	✔
Discuss best practices for highcharter maps	✔

Summary

- Today we learned how to create layered charts and maps using `highcharter`
- However, there are several other R packages that create R bindings to JavaScript libraries and open up access to a whole new catalog of visualization types!
- Next time, we will create a **network visualization** using a new package called `visNetwork`

This completes our module

Congratulations!