R Advanced Spatial Lessons

Ben Best

2017-09-27

Contents

Prerequisites

Lessons presented here are a continuation of the Geospatial workshop using R of Data Carpentry described more specifically for the Lawrence Berkeley National Lab: Sep 27-28, 2017.

This content is setup for now using bookdown (using the bookdown-demo) for expediency, and meant to eventually be folded into the Software Carpentry style.

Setup

0.1 Install software

This workshop will require the following software installed on your machine:

- R
- RStudio

Please download the appropriate stable release for your operating system.

0.2 Launch RStudio

RStudio is an integrated development environment (IDE) for editing text in the code editor, running commands in the R console, viewing defined objects in the workspace and past commands in the history, and viewing plots, files and help. Here's a layout of the panes in RStudio, each of which can have several tabs for alternate functionality:

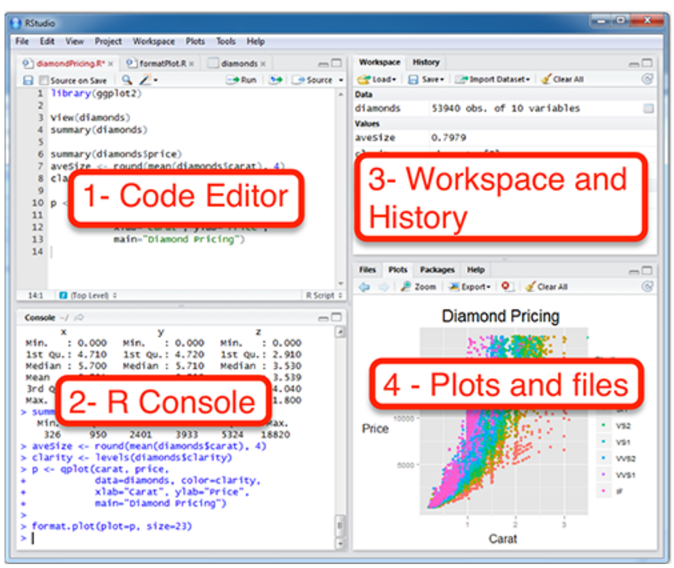


Figure 1:

Check out the Help > Cheatsheets > RStudio IDE Cheat Sheet.

0.3 Create RStudio project

In RStudio, please create an RStudio project (File > New Project… > New Directory > Empty Project) to organize your code and data for this workshop into a single folder. Feel free to organize this folder wherever makes sense on your laptop. Here's what I'm using:

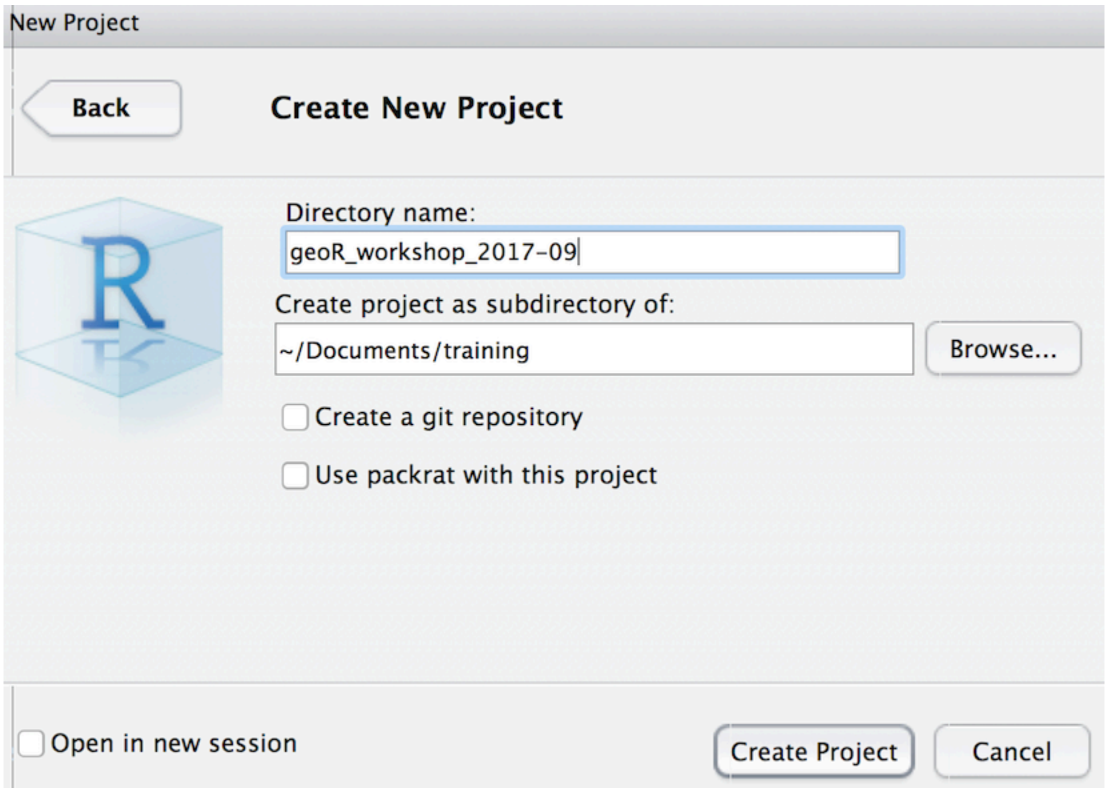


Figure 2:

0.3.1 Create relative path to data folder

In order to work through a common set of code and access data files in a consistent manner, getting properly situated with paths will save many headaches. Paths can be referenced as an "absolute" path starting with your drive letter (eg "C:" on Windows or "/Users" on Mac). Absolute paths are specific to platform (Windows / Linux / Mac) and not portable for use on other machines or other users wishing to organize content anywhere else. A "relative" path, meaning relative to your current working directory, is portable. Try these commands in the console of RStudio:

```
# get working directory
getwd()

# list files in working directory
list.files('.')

# list files one directory up
list.files('..')

# create a directory
```

```r
dir.create('data')

# set working directory
setwd('data')
setwd('..')
```

So you discovered the absolute path to your current working directory, and list files in the current directory (`.`) or one directory up (`..`). Then you created the "data" directory and set the working directory to be inside of it, then back out of it.

The main reason we created the RStudio project file (filename ends in `.Rproj`) is to have the same working directory every time you return to this project by double-clicking the `*.Rproj` file in your file explorer. So paths defined in the code of files there can be based on that same working driectory and consistently work, even if you zip them up and send to a colleague who places them on an arbitrary location on their computer.

0.4 Download data

Please download the following zip files into your newly created "data" folder from above.

- Site layout shapefiles
- Airborne remote sensing data
- Landsat NDVI raster data
- Meteorological data

Then unzip. You should see the following when listing files (and directories) in the data directory.

```r
list.files('data')
```

```
## [1] "NEON-DS-Airborne-Remote-Sensing" "NEON-DS-Landsat-NDVI"
## [3] "NEON-DS-Met-Time-Series"         "NEON-DS-Site-Layout-Files"
## [5] "NEONDSAirborneRemoteSensing.zip" "NEONDSLandsatNDVI.zip"
## [7] "NEONDSMetTimeSeries.zip"         "NEONDSSiteLayoutFiles.zip"
## [9] "states.geojson"
```

0.5 Install R Packages

Here's a bit of code to install packages that we'll use throughout the workshop. Please copy and paste this code into your console.

```r
# concatenate a vector of package names to install
packages = c(
  # general data science
  'tidyverse',
  # dynamic document creation
  'knitr','rmarkdown',
  # handle spatial data
  'rgdal','raster','sp','sf','geojsonio',
  # spatial data
  'maps',
  # spatial analysis
  'rgeos','geosphere',
  # static plotting & mapping
  'RColorBrewer','ggplot2','rasterVis',
```

```
  # interactive plotting & mapping
  'plotly','leaflet','mapview','mapedit')

# loop through packages
for (p in packages){

  # if package not installed
  if (!require(p, character.only=T)){

    # install package
    install.packages(p)
  }

  # load package
  library(p, character.only=T)
}

# report on versions of software & packages
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: OS X El Capitan 10.11.6
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] mapedit_0.3.2       rasterVis_0.41      latticeExtra_0.6-28
##  [4] lattice_0.20-35     RColorBrewer_1.1-2  rgeos_0.3-25
##  [7] maps_3.2.0          geojsonio_0.4.2     rgdal_1.2-11
## [10] rmarkdown_1.6       knitr_1.17          scales_0.5.0
## [13] htmltools_0.3.6     mapview_2.1.4       leaflet_1.1.0.9000
## [16] plotly_4.7.1        raster_2.5-8        units_0.4-6
## [19] geosphere_1.5-5     sp_1.2-5            bindrcpp_0.2
## [22] sf_0.5-4            dplyr_0.7.3         purrr_0.2.3
## [25] readr_1.1.1         tidyr_0.7.1         tibble_1.3.4
## [28] ggplot2_2.2.1.9000  tidyverse_1.1.1
##
## loaded via a namespace (and not attached):
##  [1] nlme_3.1-131      satellite_1.0.0   lubridate_1.6.0
##  [4] webshot_0.4.1     httr_1.3.1        rprojroot_1.2
##  [7] tools_3.4.0       backports_1.1.0   R6_2.2.2
## [10] DBI_0.7           lazyeval_0.2.0    colorspace_1.3-2
## [13] mnormt_1.5-5      curl_2.8.1        compiler_3.4.0
## [16] rvest_0.3.2       xml2_1.1.1        bookdown_0.5
## [19] hexbin_1.27.1     psych_1.7.8       stringr_1.2.0
```

```
## [22] digest_0.6.12      foreign_0.8-69    R.utils_2.5.0
## [25] vembedr_0.1.3      base64enc_0.1-3   pkgconfig_2.0.1
## [28] htmlwidgets_0.9    rlang_0.1.2       readxl_1.0.0
## [31] rstudioapi_0.7     shiny_1.0.5       bindr_0.1
## [34] zoo_1.8-0          jsonlite_1.5      crosstalk_1.0.0
## [37] R.oo_1.21.0        magrittr_1.5      Rcpp_0.12.12
## [40] munsell_0.4.3      R.methodsS3_1.7.1 stringi_1.1.5
## [43] yaml_2.1.14        plyr_1.8.4        grid_3.4.0
## [46] maptools_0.9-2     parallel_3.4.0    forcats_0.2.0
## [49] udunits2_0.13      haven_1.1.0       hms_0.3
## [52] gdalUtils_2.0.1.7 reshape2_1.4.2    codetools_0.2-15
## [55] stats4_3.4.0       glue_1.1.1        evaluate_0.10.1
## [58] V8_1.5             data.table_1.10.4 modelr_0.1.1
## [61] png_0.1-7          httpuv_1.3.5      foreach_1.4.3
## [64] cellranger_1.1.0   gtable_0.2.0      assertthat_0.2.0
## [67] mime_0.5           xtable_1.8-2      broom_0.4.2
## [70] viridisLite_0.2.0 iterators_1.0.8
```

0.6 Create Rmarkdown file

Rmarkdown is a dynamic document format that allows you to knit chunks of R code with formatted text (aka markdown). We recommend that you use this format for keeping a reproducible research document as you work through the lessons To get started, File > New File > Rmarkdown… and go with default HTML document and give any title you like (default "Untitled" or "test" or "First Rmd" is fine).

Check out the Help > Markdown Quick Reference and Help > Cheatsheets > R Markdown Cheat Sheet.

Here's a 1 minute video on the awesomeness of Rmarkdown:

Chapter 1

Tidy Spatial Analysis

1.1 Overview

Questions

- How to elegantly conduct complex spatial analysis by chaining operations?
- What is the percent area of water by region across the United States?

Objectives

- Use the `%>%` operator (aka "then" or "pipe") to pass output from one function into input of the next.
- Calculate metrics on spatial attributes.
- Aggregate spatial data with metrics.
- Display a map of results.

1.2 Prerequisites

R Skill Level: Intermediate - you've got basics of R down.

You will use the `sf` package for vector data along with the `dplyr` package for calculating and manipulating attribute data.

```
# load packages
library(tidyverse)  # load dplyr, tidyr, ggplot2 packages
library(sf)         # vector reading & analysis

# set working directory to data folder
# setwd("pathToDirHere")
```

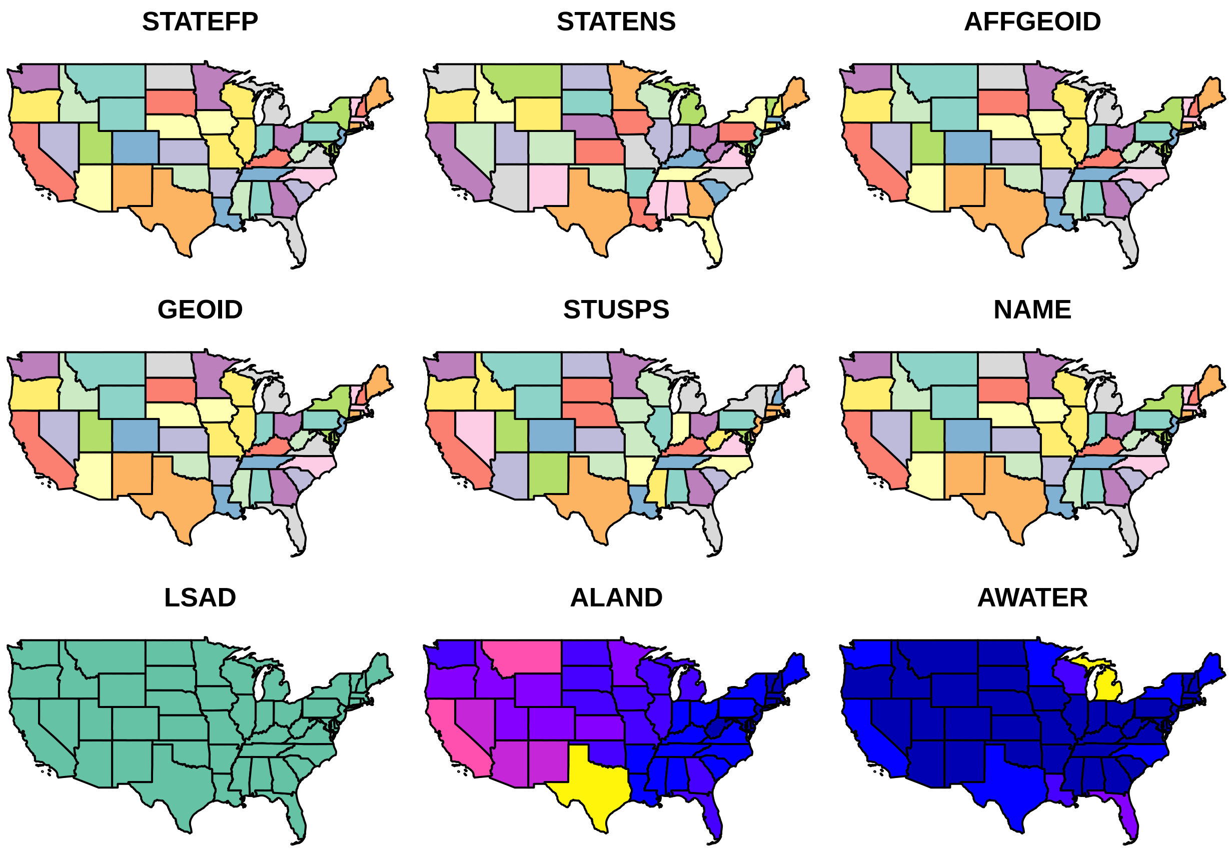
1.3 States: read and plot

Similar to Lesson 9: Handling Spatial Projection & CRS in R, we'll start by reading in a polygon shapefile using the `sf` package. Then use the default `plot()` function to see what it looks like.

```
# read in states
states <- read_sf("data/NEON-DS-Site-Layout-Files/US-Boundary-Layers/US-State-Boundaries-Census-2014.sh
```

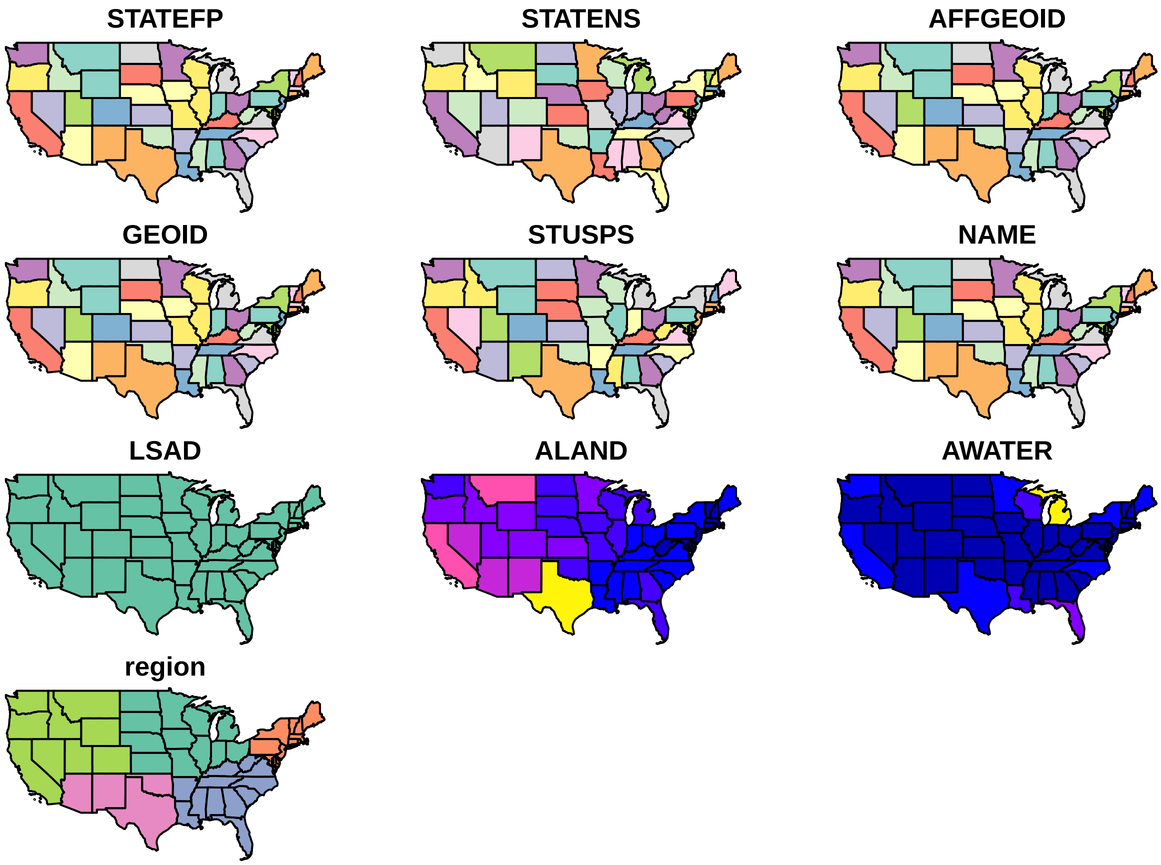
```r
# plot the states
plot(states)
```

```
## Warning: plotting the first 9 out of 10 attributes; use max.plot = 10 to
## plot all
```



Notice the default plot on `sf` objects outputs colorized values of the first 9 of 10 columns. Use the suggestion from the warning to plot the 10th column.

```r
# plot 10th column
plot(states, max.plot = 10)
```




```r
# show columns of the data frame
names(states)
```

```
##  [1] "STATEFP"  "STATENS"  "AFFGEOID" "GEOID"    "STUSPS"   "NAME"
##  [7] "LSAD"     "ALAND"    "AWATER"   "region"   "geometry"
```

```r
# look at table
glimpse(states)
```

```
## Observations: 58
## Variables: 11
## $ STATEFP  <chr> "06", "11", "12", "13", "16", "17", "19", "21", "22",...
## $ STATENS  <chr> "01779778", "01702382", "00294478", "01705317", "0177...
## $ AFFGEOID <chr> "0400000US06", "0400000US11", "0400000US12", "0400000...
## $ GEOID    <chr> "06", "11", "12", "13", "16", "17", "19", "21", "22",...
## $ STUSPS   <chr> "CA", "DC", "FL", "GA", "ID", "IL", "IA", "KY", "LA",...
## $ NAME     <chr> "California", "District of Columbia", "Florida", "Geo...
## $ LSAD     <chr> "00", "00", "00", "00", "00", "00", "00", "00", "00",...
## $ ALAND    <dbl> 403483823181, 158350578, 138903200855, 148963503399, ...
## $ AWATER   <dbl> 20483271881, 18633500, 31407883551, 4947080103, 23977...
## $ region   <chr> "West", "Northeast", "Southeast", "Southeast", "West"...
## $ geometry <simple_feature> MULTIPOLYGONZ(((-118.593969..., MULTIPOLYG...
```

```r
# convert to tibble for nicer printing
as_tibble(states)
```

```
## Simple feature collection with 58 features and 10 fields
## geometry type:  MULTIPOLYGON
```

```
## dimension:      XYZ
## bbox:           xmin: -124.7258 ymin: 24.49813 xmax: -66.9499 ymax: 49.38436
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## # A tibble: 58 x 11
##    STATEFP  STATENS    AFFGEOID GEOID STUSPS                 NAME  LSAD
##      <chr>    <chr>       <chr> <chr>  <chr>                <chr> <chr>
##  1      06 01779778 0400000US06    06     CA           California    00
##  2      11 01702382 0400000US11    11     DC District of Columbia    00
##  3      12 00294478 0400000US12    12     FL              Florida    00
##  4      13 01705317 0400000US13    13     GA              Georgia    00
##  5      16 01779783 0400000US16    16     ID                Idaho    00
##  6      17 01779784 0400000US17    17     IL             Illinois    00
##  7      19 01779785 0400000US19    19     IA                 Iowa    00
##  8      21 01779786 0400000US21    21     KY             Kentucky    00
##  9      22 01629543 0400000US22    22     LA            Louisiana    00
## 10      24 01714934 0400000US24    24     MD             Maryland    00
## # ... with 48 more rows, and 4 more variables: ALAND <dbl>, AWATER <dbl>,
## #   region <chr>, geometry <simple_feature>
```

```r
names(states)
```

```
##  [1] "STATEFP"  "STATENS"  "AFFGEOID" "GEOID"    "STUSPS"   "NAME"
##  [7] "LSAD"     "ALAND"    "AWATER"   "region"   "geometry"
```

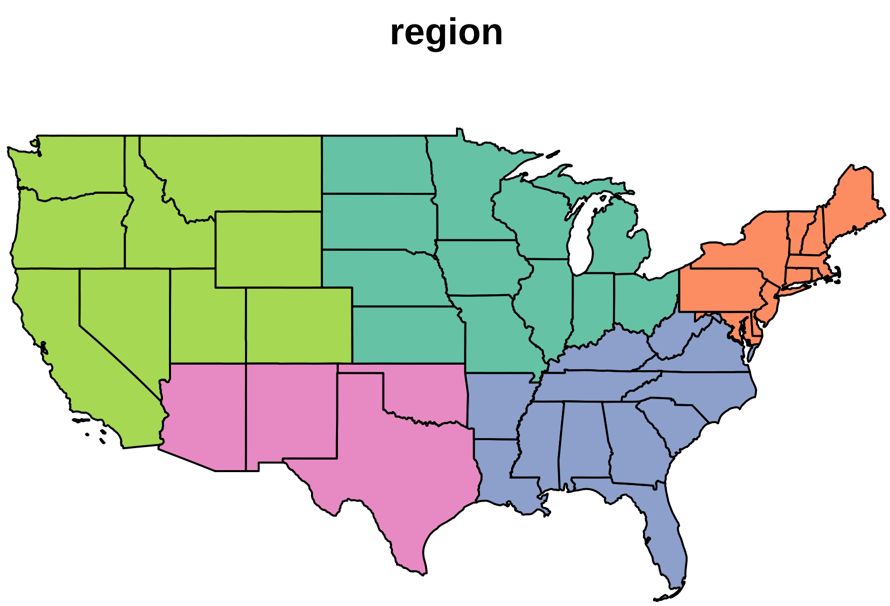
```r
# inspect the class(es) of the states object
class(states)
```

```
## [1] "sf"         "tbl_df"     "tbl"        "data.frame"
```

The class of the `states` object is both a simple feature (`sf`) as well as a data frame, which means the many useful functions available to a data frame (or "tibble") can be applied.

To plot the column of interest, feed the "slice" of that column to the `plot()` function.

```r
plot(states['region'])
```



Question: To motivate the spatial analysis for the rest of this lesson, you will answer this question: "***What***

is the percent water by region?”

1.4 Challenge: analytical steps?

Outline a sequence of analytical steps needed to arrive at the answer.

1.4.1 Answers

1. **Sum** the area of water (`AWATER`) and land (`ALAND`) per region.
2. **Divide** the area of water (`AWATER`) by the area of land (`ALAND`) per region to arrive at percent water.
3. Show **table** of regions sorted by percent water.
4. Show **map** of regions by percent water with a color ramp and legend.

1.5 Regions: calculate % water

- Use the `%>%` operator (aka “then” or “pipe”) to pass output from one function into input of the next.
  - In RStudio, see menu Help > Keyboard Shortcuts Help for a shortcut to the “Insert Pipe Operator”.
- Calculate metrics on spatial attributes.
  - In RStudio, see menu Help > Cheatsheets > Data Manipulation with dplyr, tidyr.
- Aggregate spatial data with metrics.

```
regions = states %>%
  group_by(region) %>%
  summarize(
    water = sum(AWATER),
    land  = sum(ALAND)) %>%
  mutate(
    pct_water = water / land * 100 %>% round(2))

# object
regions
```

```
## Simple feature collection with 5 features and 4 fields
## geometry type:  GEOMETRY
## dimension:      XYZ
## bbox:           xmin: -124.7258 ymin: 24.49813 xmax: -66.9499 ymax: 49.38436
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
## # A tibble: 5 x 5
##      region        water         land pct_water          geometry
##       <chr>        <dbl>        <dbl>     <dbl>  <simple_feature>
## 1   Midwest 184383393833 1.943869e+12  9.485380 <MULTIPOLYGON...>
## 2 Northeast 108922434345 8.690661e+11 12.533273 <MULTIPOLYGON...>
## 3 Southeast 103876652998 1.364632e+12  7.612063 <MULTIPOLYGON...>
## 4 Southwest  24217682268 1.462632e+12  1.655761 <POLYGONZ((-9...>
## 5      West  57568049509 2.432336e+12  2.366780 <MULTIPOLYGON...>
```

Notice the geometry in the column. To remove the geometry column pipe to `st_set_geometry(NULL)`. To arrange in descending order use `arrange(desc(pct_water))`.

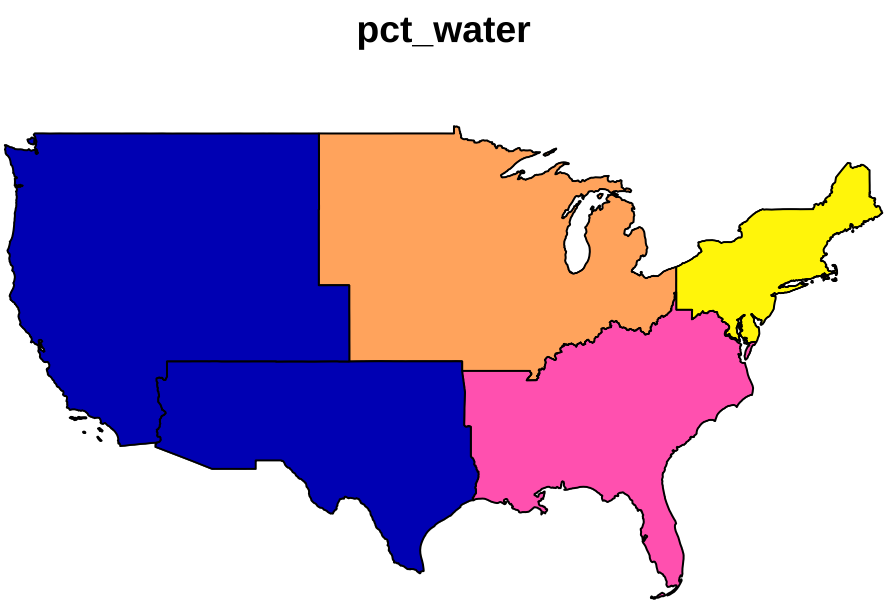
```
# table
regions %>%
  st_set_geometry(NULL) %>%
  arrange(desc(pct_water))
```

```
## # A tibble: 5 x 4
##      region        water         land pct_water
##       <chr>        <dbl>        <dbl>     <dbl>
## 1 Northeast 108922434345 8.690661e+11 12.533273
## 2   Midwest 184383393833 1.943869e+12  9.485380
## 3 Southeast 103876652998 1.364632e+12  7.612063
## 4      West  57568049509 2.432336e+12  2.366780
## 5 Southwest  24217682268 1.462632e+12  1.655761
```

1.6 Regions: plot

Now plot the regions.

```
# plot, default
plot(regions['pct_water'])
```



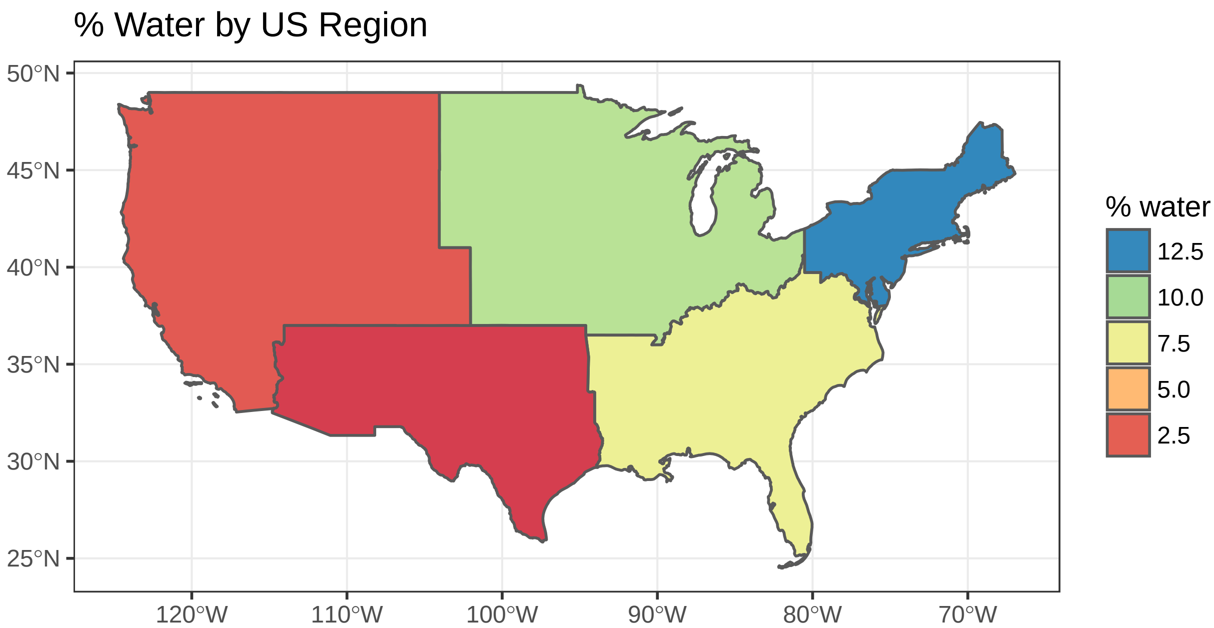
1.7 Regions: ggplot

The `ggplot2` library can visualise sf objects.

- In RStudio, see menu Help > Cheatsheets > Data Visualization with ggplot2.

```
# plot, ggplot
ggplot(regions) +
  geom_sf(aes(fill = pct_water)) +
  scale_fill_distiller(
    "pct_water", palette = "Spectral", direction=1,
    guide = guide_legend(title = "% water", reverse=T)) +
```

```
  theme_bw() +
  ggtitle("% Water by US Region")
```



1.8 Regions: recalculate area

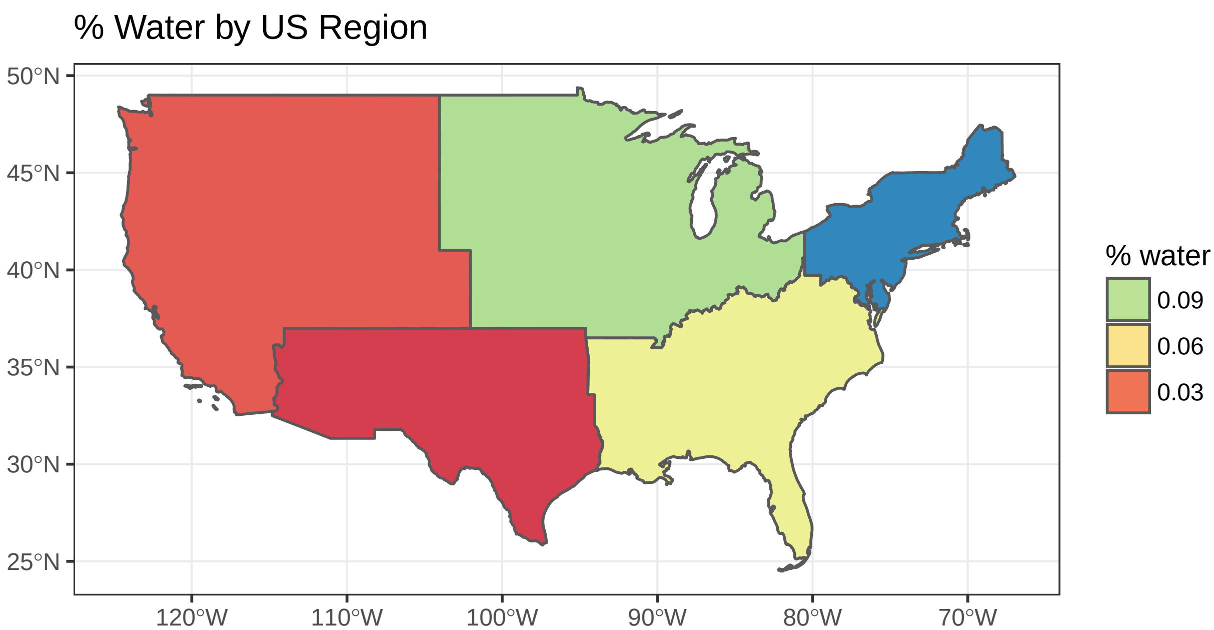
So far you've used the `ALAND` column for area of the state. But what if you were not provided the area and needed to calculate it? Because the `states` are in geographic coordinates, you'll need to either transform to an equal area projection and calculate area, or use geodesic calculations. Thankfully, the `sf` library provides area calculations with the `st_area()` and uses the `geosphere::distGeo()` to perform geodesic calculations (ie trigonometric calculation accounting for the spheroid nature of the earth). Since the `states` data has the unusual aspect of a z dimension, you'll need to first remove that with the `st_zm()` function.

```
library(geosphere)
library(units)

regions = states %>%
  mutate(
    water_m2 = AWATER %>% set_units(m^2),
    land_m2  = geometry %>% st_zm() %>% st_area()) %>%
  group_by(region) %>%
  summarize(
    water_m2 = sum(water_m2),
    land_m2  = sum(land_m2)) %>%
  mutate(
    pct_water = water_m2 / land_m2)

# table
regions %>%
  st_set_geometry(NULL) %>%
  arrange(desc(pct_water))
```

```
## # A tibble: 5 x 4
```

```
##      region         water_m2          land_m2    pct_water
##       <chr>          <units>          <units>      <units>
## 1 Northeast 108922434345 m^2 9.117041e+11 m^2 0.11947126 1
## 2   Midwest 184383393833 m^2 1.987268e+12 m^2 0.09278233 1
## 3 Southeast 103876652998 m^2 1.427079e+12 m^2 0.07278971 1
## 4      West  57568049509 m^2 2.467170e+12 m^2 0.02333363 1
## 5 Southwest  24217682268 m^2 1.483765e+12 m^2 0.01632178 1
```

```
# plot, ggplot
ggplot(regions) +
  geom_sf(aes(fill = as.numeric(pct_water))) +
  scale_fill_distiller(
    "pct_water", palette = "Spectral", direction=1,
    guide = guide_legend(title = "% water", reverse=T)) +
  theme_bw() +
  ggtitle("% Water by US Region")
```



1.9 Challenge: project & recalculate area

Use `st_transform()` with a USA Contiguous Albers Equal Area Conic Projection that minimizes distoration, and then calculate area using the `st_area()` function.

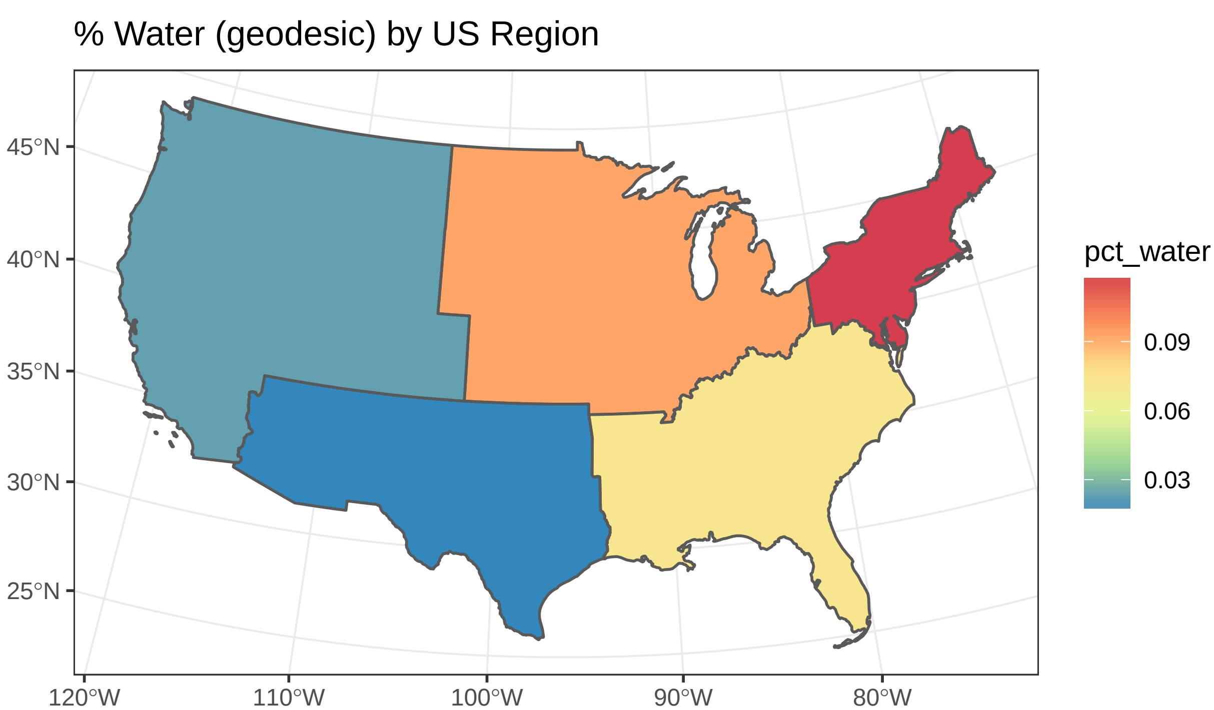
1.9.1 Answers

```
library(geosphere)
library(units)

# Proj4 of http://spatialreference.org/ref/esri/usa-contiguous-albers-equal-area-conic/
crs_usa = '+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 +ellps=GRS80 +datum=N

regions = states %>%
```

```r
  st_transform(crs_usa) %>%
  mutate(
    water_m2 = AWATER %>% set_units(m^2),
    land_m2  = geometry %>% st_zm() %>% st_area()) %>%
  group_by(region) %>%
  summarize(
    water_m2 = sum(water_m2),
    land_m2  = sum(land_m2)) %>%
  mutate(
    pct_water = water_m2 / land_m2)

# table
regions %>%
  st_set_geometry(NULL) %>%
  arrange(desc(pct_water))
```

```
## # A tibble: 5 x 4
##      region         water_m2          land_m2    pct_water
##       <chr>          <units>          <units>      <units>
## 1 Northeast 108922434345 m^2 9.117031e+11 m^2 0.11947138 1
## 2   Midwest 184383393833 m^2 1.987266e+12 m^2 0.09278246 1
## 3 Southeast 103876652998 m^2 1.427078e+12 m^2 0.07278973 1
## 4      West  57568049509 m^2 2.467167e+12 m^2 0.02333367 1
## 5 Southwest  24217682268 m^2 1.483758e+12 m^2 0.01632185 1
```

```r
# plot, ggplot
ggplot(regions) +
  geom_sf(aes(fill = as.numeric(pct_water))) +
  scale_fill_distiller("pct_water", palette = "Spectral") +
  theme_bw() +
  ggtitle("% Water (geodesic) by US Region")
```



1.10 Key Points

- The `sf` package can take advantage of chaining spatial operations using the `%>%` operator.
- Data manipulation functions in `dplyr` such as `group_by()`, `summarize()` and `mutate()` work on `sf` objects.
- Area can be calculated a variety of ways. Geodesic is preferred if starting with geographic coordinates (vs projected).

Chapter 2

Interactive Maps

2.1 Overview

Questions

- How do you generate interactive plots of spatial data to enable pan, zoom and hover/click for more detail?

Objectives

Learn variety of methods for producing interactive spatial output using libraries:

- `plotly`: makes any ggplot2 object interactive
- `mapview`: quick view of any spatial object
- `leaflet`: full control over interactive map

2.2 Things You'll Need to Complete this Tutorial

R Skill Level: Intermediate - you've got basics of R down.

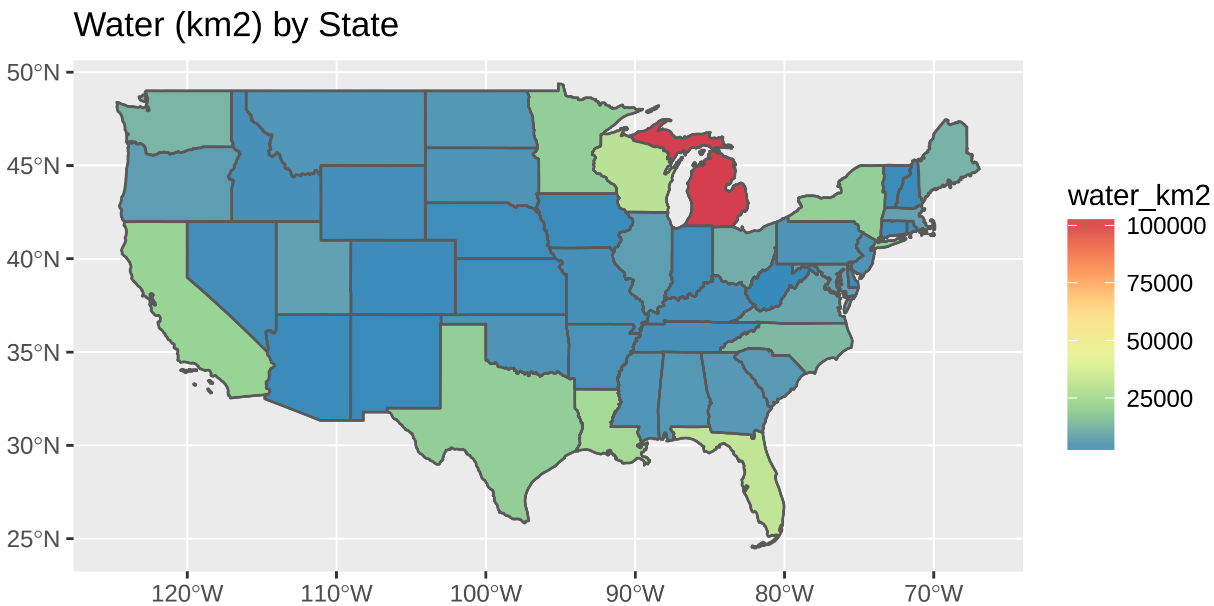
We will continue to use the `sf` and `raster` packages and introduce the `plotly`, `mapview`, and `leaflet` packages in this tutorial.

```r
# load packages
library(tidyverse)  # loads dplyr, tidyr, ggplot2 packages
library(sf)         # simple features package - vector
library(raster)     # raster
library(plotly)     # makes ggplot objects interactive
library(mapview)    # quick interactive viewing of spatial objects
library(leaflet)    # interactive maps

# set working directory to data folder
# setwd("pathToDirHere")
```

2.3 States: ggplot2

Recreate the ggplot object from Lesson 1 and save into a variable for subsequent use with the `plotly` package.

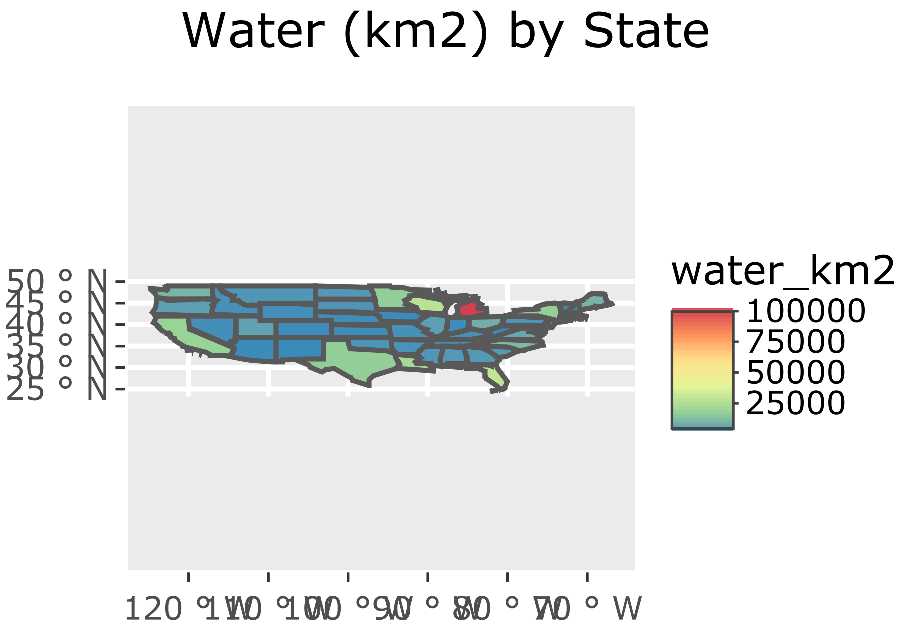
```
# read in states
states <- read_sf("data/NEON-DS-Site-Layout-Files/US-Boundary-Layers/US-State-Boundaries-Census-2014.shp
  st_zm() %>%
  mutate(
    water_km2 = (AWATER / (1000*1000)) %>% round(2))

# plot, ggplot
g = ggplot(states) +
  geom_sf(aes(fill = water_km2)) +
  scale_fill_distiller("water_km2", palette = "Spectral") +
  ggtitle("Water (km2) by State")
g
```



2.4 States: plotly

The `plotly::ggplotly()` function outputs a ggplot into an interactive window capable of pan, zoom and identify.

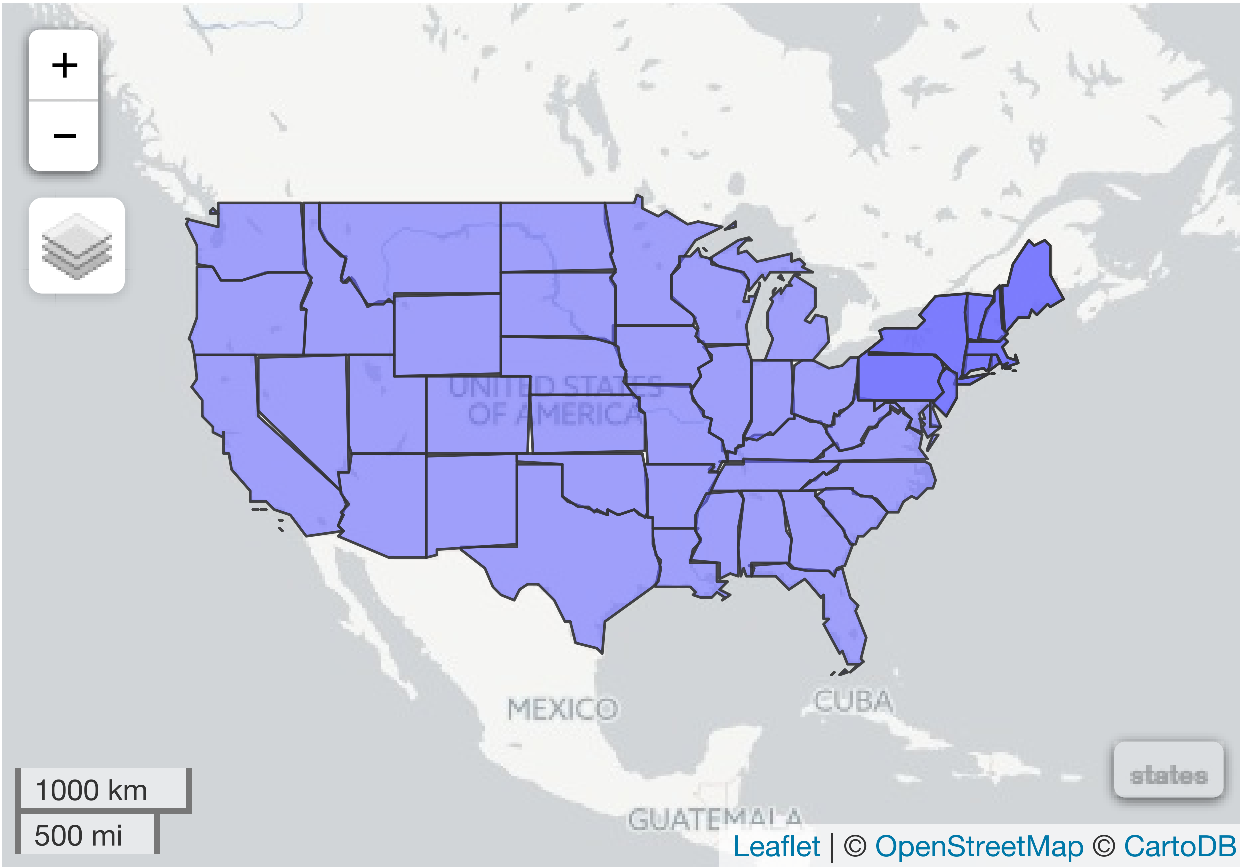
```
library(plotly)

ggplotly(g)
```



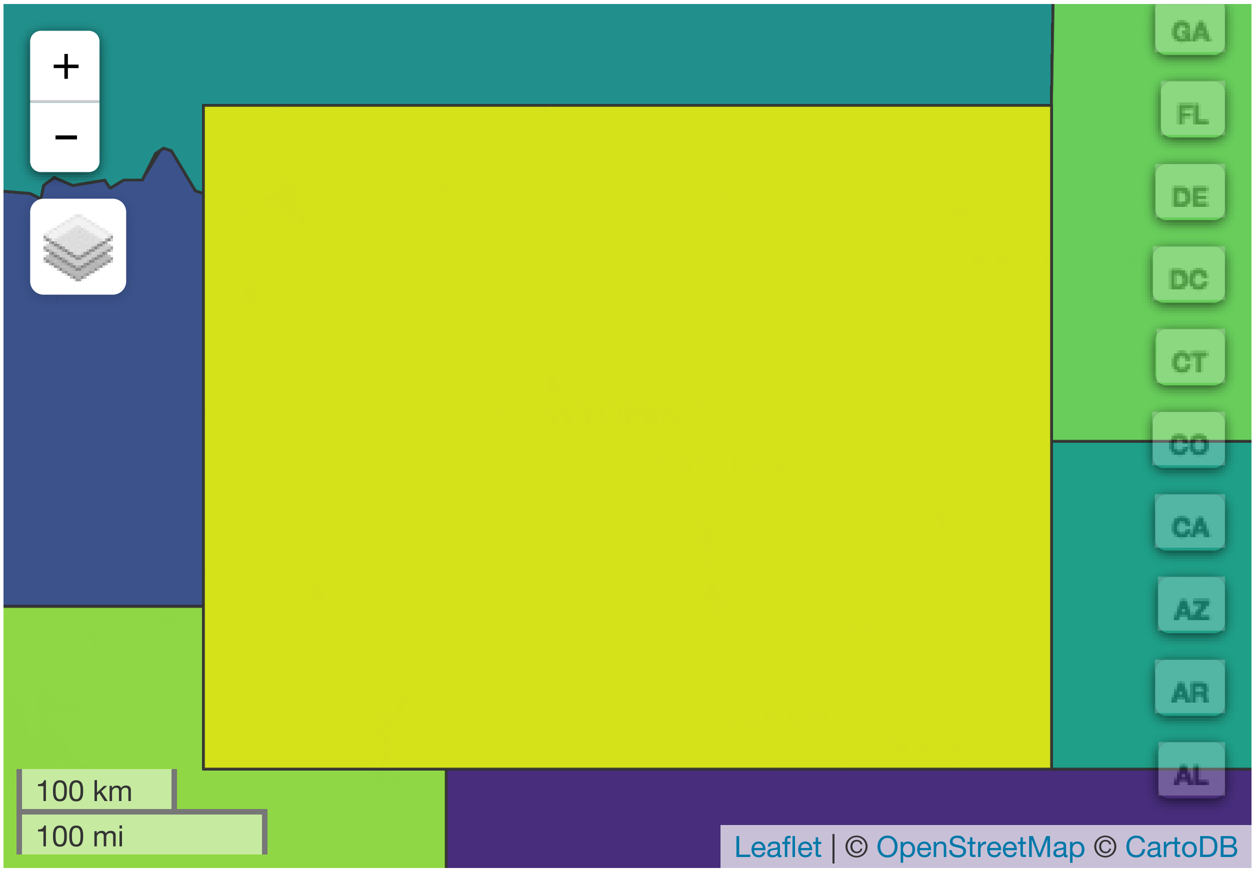
2.5 States: mapview

The `mapview::mapview()` function can work for a quick view of the data, providing choropleths, background maps and attribute popups. Performance varies on the object and customization can be tricky.

```
library(mapview)

# simple view with popups
mapview(states)
```



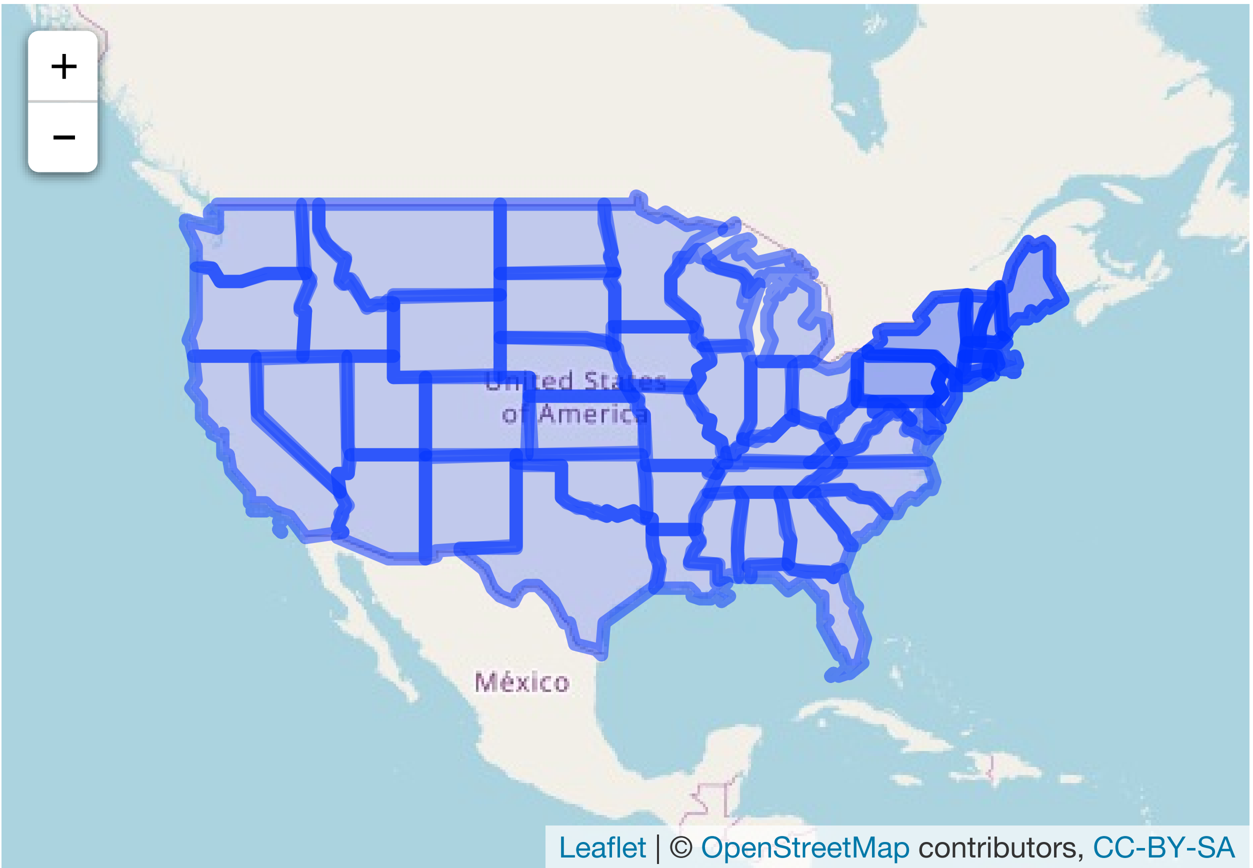

```
# coloring and layering
mapview(states, zcol='water_km2', burst='STUSPS')
```



2.6 States: leaflet

The `leaflet` package offers a robust set of functions for viewing vector and raster data, although requires more explicit functions.

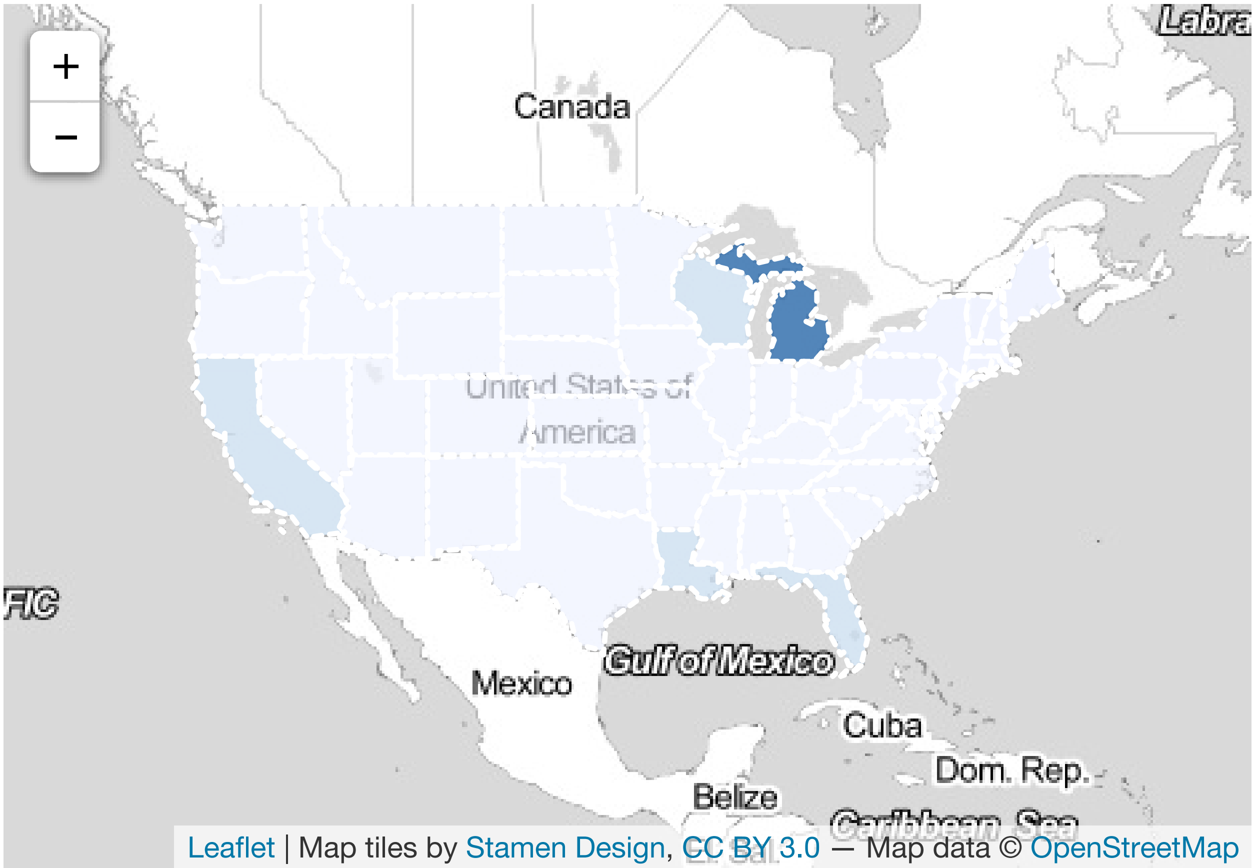
```
library(leaflet)

leaflet(states) %>%
  addTiles() %>%
  addPolygons()
```



2.6.1 Choropleth

Drawing from the documentation from Leaflet for R - Choropleths, we can construct a pretty choropleth.

```
pal <- colorBin("Blues", domain = states$water_km2, bins = 7)


leaflet(states) %>%
  addProviderTiles("Stamen.TonerLite") %>%
  addPolygons(
    # fill
    fillColor   = ~pal(water_km2),
    fillOpacity = 0.7,
    # line
    dashArray   = "3",
    weight      = 2,
    color       = "white",
    opacity     = 1,
    # interaction
    highlight = highlightOptions(
      weight = 5,
      color = "#666",
      dashArray = "",
      fillOpacity = 0.7,
      bringToFront = TRUE))
```

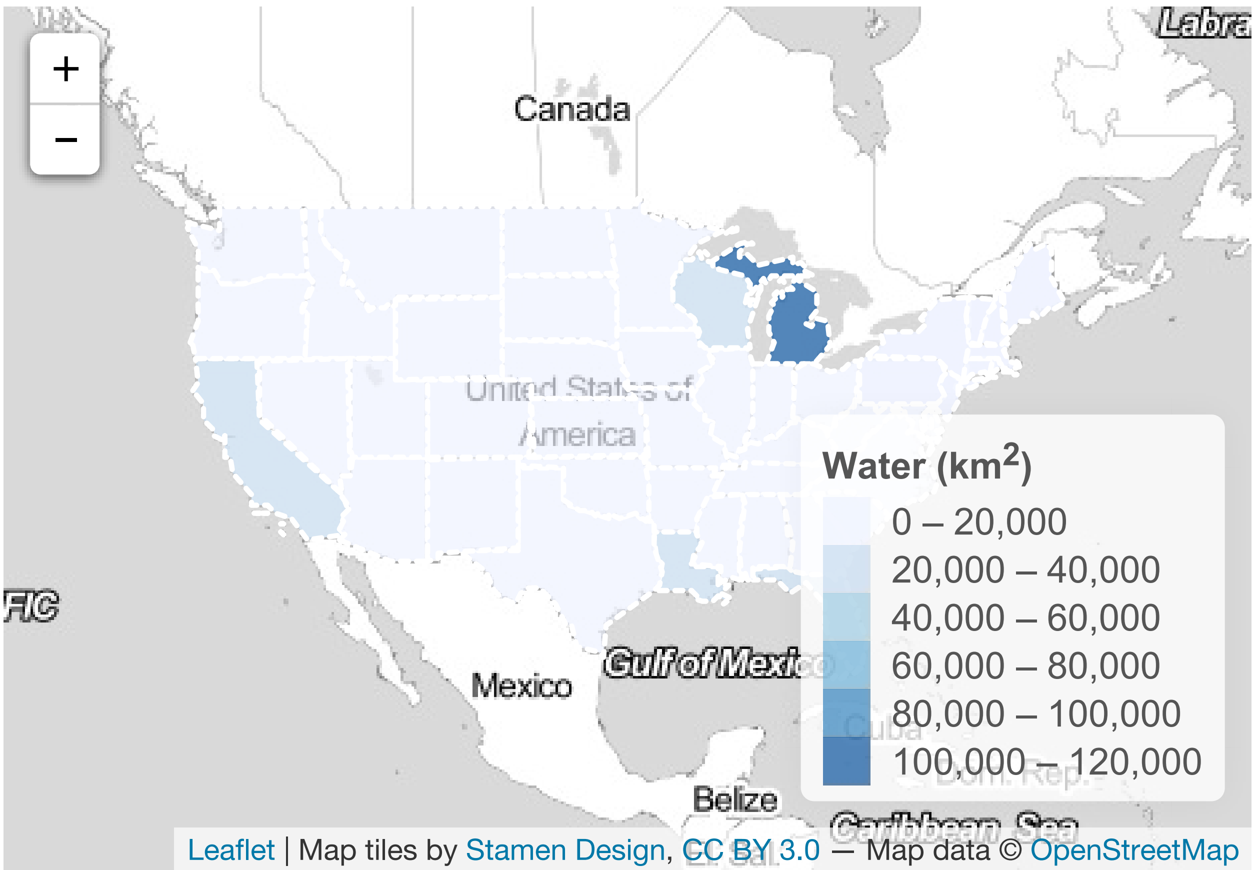


2.6.2 Popups and Legend

Adding a legend and popups requires a bit more work, but achieves a very aesthetically and functionally pleasing visualization.

```
library(htmltools)
library(scales)

labels <- sprintf(
  "<strong>%s</strong><br/> water: %s km<sup>2</sup>",
  states$NAME, comma(states$water_km2)) %>%
  lapply(HTML)

leaflet(states) %>%
  addProviderTiles("Stamen.TonerLite") %>%
  addPolygons(
    # fill
    fillColor   = ~pal(water_km2),
    fillOpacity = 0.7,
    # line
    dashArray   = "3",
    weight      = 2,
    color       = "white",
    opacity     = 1,
    # interaction
```

```
    highlight = highlightOptions(
      weight = 5,
      color = "#666",
      dashArray = "",
      fillOpacity = 0.7,
      bringToFront = TRUE),
  label = labels,
  labelOptions = labelOptions(
    style = list("font-weight" = "normal", padding = "3px 8px"),
    textsize = "15px",
    direction = "auto")) %>%
  addLegend(
    pal = pal, values = ~water_km2, opacity = 0.7, title = HTML("Water (km<sup>2</sup>)"),
    position = "bottomright")
```

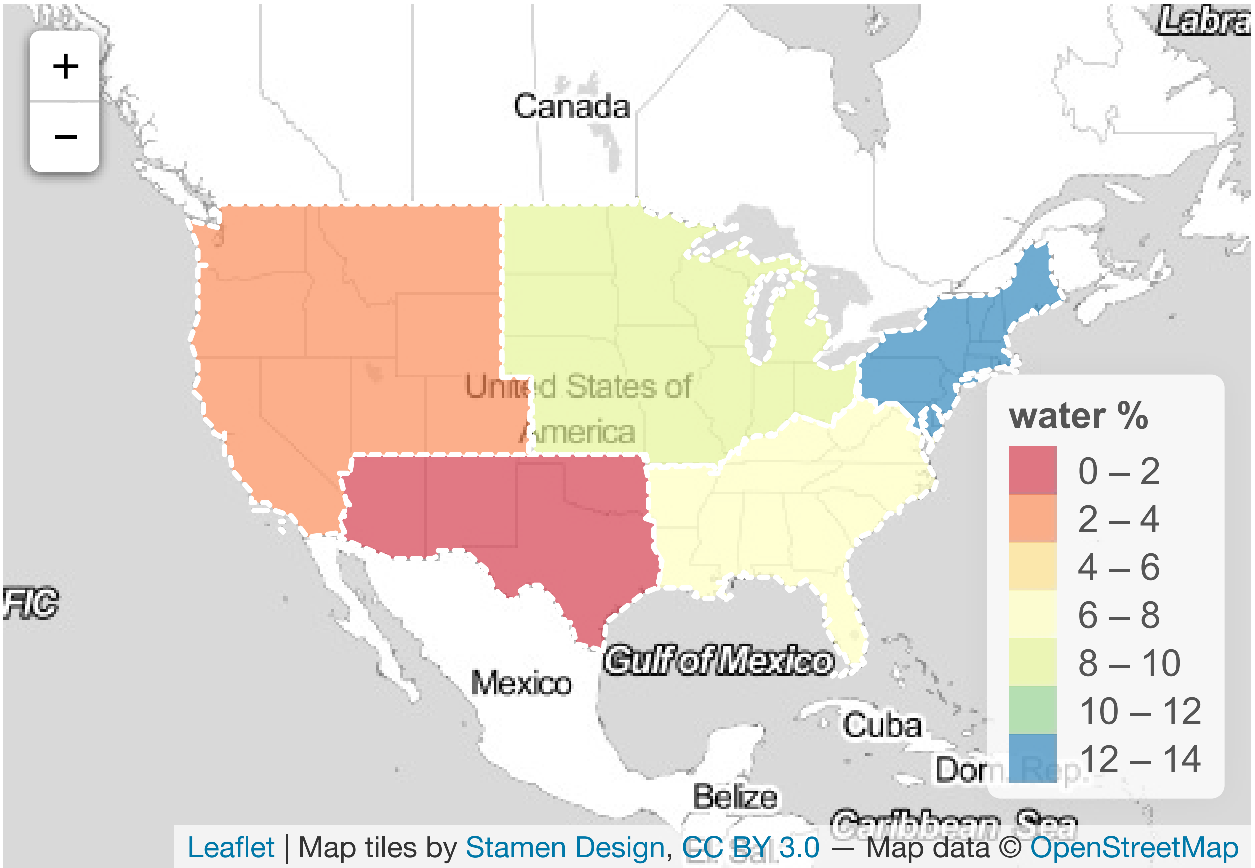


2.7 Challenge: leaflet for regions

Use Lesson 1 final output to create a regional choropleth with legend and popups for percent water by region.

2.7.1 Answers

```
regions = states %>%
  group_by(region) %>%
  summarize(
    water = sum(AWATER),
    land  = sum(ALAND)) %>%
  mutate(
    pct_water = (water / land * 100) %>% round(2))

pal <- colorBin("Spectral", domain = regions$pct_water, bins = 5)

labels <- sprintf(
  "<strong>%s</strong><br/>water: %s%%",
  regions$region, comma(regions$pct_water)) %>%
  lapply(HTML)

leaflet(regions) %>%
  addProviderTiles("Stamen.TonerLite") %>%
  addPolygons(
    # fill
    fillColor   = ~pal(pct_water),
    fillOpacity = 0.7,
    # line
    dashArray   = "3",
    weight      = 2,
    color       = "white",
    opacity     = 1,
    # interaction
    highlight = highlightOptions(
      weight = 5,
      color = "#666",
      dashArray = "",
      fillOpacity = 0.7,
      bringToFront = TRUE),
  label = labels,
  labelOptions = labelOptions(
    style = list("font-weight" = "normal", padding = "3px 8px"),
    textsize = "15px",
    direction = "auto")) %>%
  addLegend(
    pal = pal, values = ~pct_water, opacity = 0.7, title = "water %",
    position = "bottomright")
```



2.8 Raster: leaflet

TODO: show raster overlay using NEON raster dataset example

2.9 Key Points

- Interactive maps provide more detail for visual investigation, including use of background maps, but is only relevant in a web context.
- Several packages exist for providing interactive views of data.
- The `plotly::ggplotly()` function works quickly if you already have a ggplot object, which is best for static output.
- The `mapview::mapview()` function can work for a quick view of the data, providing choropleths, background maps and attribute popups. Performance varies on the object and customization can be tricky.
- The `leaflet` package provides a highly customizable set of functions for rendering of interactive choropleths with background maps, legends, etc.

References

Spatial Analysis in R books

- Geocomputation with R - robinlovelace.net/geocompr
- Spatial Data Analysis and Modeling with R — rspatial.org
- Geospatial Analysis 5th Edition, 2015 - spatialanalysisonline.com

Spatial Analysis in R blogs

- CRAN Task View: Analysis of Spatial Data
- R spatial projects
- R sp graphics example figures
- Maps and Data Visualisations with R – spatial.ly

Spatial Analysis in R courses

- Spatial Analysis in R - datacamp.com
- An Introduction to Spatial Data Analysis and Visualisation in R - CDRC Data
- Introduction to Mapping and Spatial Analysis with R - cengel.github.io/rspatial
- GEOG 4/595: Geographic Data Analysis - geog.uoregon.edu
- Geospatial Data in R and Beyond - maths.lancs.ac.uk
  - R Spatial Cheatsheet

Tidy Spatial Analysis

- Tidy spatial data in R: using dplyr, tidyr, and ggplot2 with sf
- Cheatsheets – RStudio

Interactive Maps

- Visualization in R - 2016-04-15-UCSB workshop
- `leaflet`
- `mapedit`
- `mapview`