Reproducible Reporting with R (R^3) for marine ecosystem indicators

Contents

Overview

These materials are for a 1-day workshop on Reproducible Reporting with R for the Working Group on the Northwest Atlantic Regional Sea (WGNARS) at Dartmouth SMAST on March 16, 2020.

- ***Location***: School for Marine Science & Technology (SMAST), UMass Dartmouth 706 S Rodney French Blvd, New Bedford, MA 02744
- ***Date***: Monday, March 16, 2020

Marine ecosystem indicators are ideally developed in an efficient, transparent, time sensitive manner so as to consistently and effectively support decision-making by managers and inform on ecosystem status for the general public. In this workshop we will demonstrate the use of the R scripting language for automating the ingestion and wrangling of data, calculation of indicators, visualization of maps or time series and online communication with interactive or static reports. We will also use Github to version and collaboratively develop code, communicate technically and host free websites. Finally, development of interactive user applications using Shiny and interactive infographics using a custom library will be briefly reviewed. Please check back here for the latest online materials.

This is the landing site for our workshop materials. Google Drive (folders, docs, slides, sheets, etc.) allows us to be agile with adapting content on the fly. The rest of the content in this site is generated using the techniques taught: R + markdown = Rmarkdown, git, Github, etc.

Github

Here is the Github repository of the source files for this website:

- github.com/ecoquants/r3-workshop

Google Drive

- **workshop/**
  - **agenda**
  - **notes**
  - **presentations/**
  - **data/**

Resources

- R for Data Science
- Spatial Data Analysis and Modeling with R
- Introduction to GIS - slides

Chapter 1

Collaborate: git, github

This section is paired with:

- Data Wrangling in the R Tidyverse - Google Slides

The two main tools you'll learn about to start are:

- **Git** is a version control system that lets you track changes to files over time. These files can be any kind of file (eg doc, pdf, xls), but free text differences are most easily visible (eg txt, csv, md). You can rollback changes made by you, or others. This facilitates a playground for collaboration, without fear of experimentation (you can always rollback changes).

- **Github** is a website for storing your git versioned files remotely. It has many nice features to be able visualize differences between images, rendering & diffing map data files, render text data files, and track changes in text.

Steps:

1. Create Github login
2. Create project website with Github Pages
3. Edit README.md in Markdown
4. Create HTML website content with R Markdown

1.1 Setup Github & Git

1. Create **Github** account at http://github.com, if you don't already have one. For username, I recommend all lower-case letters, short as you can. If you use an email ending in `.edu`, you can request free private repositories via GitHub Education discount.

2. Configure **git** with global commands. Open up the Bash version of Git and type the following:

```
# display your version of git
git --version

# replace USER with your Github user account
git config --global user.name USER

# replace USER@SOMEWHERE.EDU with the email you used to register with Github
git config --global user.email USER@SOMEWHERE.EDU

# list your config to confirm user.* variables set
git config --list
```

1.2 Github Workflows

The two most common workflow models for working Github repositories is based on your permissions:

1. **writable**: Push & Pull (simplest)
2. **read only**: Fork & Pull Request (extra steps)

We will only go over the first writable mode. For more on the second mode, see Forking Projects · GitHub Guides.

1.2.1 Push & Pull

repo location	initialize	edit	update
`github.com/OWNER/REPO`	**create**		
`~/github/REPO`	**clone**	**commit** , **push**	**pull**

Note that OWNER could be either an individual USER or group ORGANIZATION, which has member USERs.

1.3 Create Repository `nps-demo`

Now you will create a Github repository for a project.

1. Create a repository called `my-project`.

Please be sure to tick the box to **Initialize this repository with a README**. Otherwise defaults are fine.

2. Create a branch called `gh-pages`.

   Per pages.github.com, since this will be a project site only web files in the `gh-pages` branch will show up at `http://USER.github.io/REPO`. For a user (or organization) site, the REPO must be named `USER.github.io` (or `ORG.github.io`) and then the default `master` branch will contain the web files for the website `http://USER.github.io` (or `http://ORG.github.io`). See also User, Organization, and Project Pages - Github Help.

3. Set the default branch to `gh-pages`, NOT the default `master`.

4. Delete the branch `master`, which will not be used.

1.3.1 Edit README.md in Markdown

Commit your first change by editing the `README.md` which is in **markdown**, simple syntax for conversion to HTML. Now update the contents of the `README.md` with the following, having a link and a numbered list:

```
# nps-demo

Wrangling data with R.

## Introduction

This repository demonstrates **software** and _formats_:

1. **Git**
1. **Github**
1. _Markdown_
1. _Rmarkdown_

## Conclusion

![](https://octodex.github.com/images/labtocat.png)
```

Now click on the Preview changes to see the markdown rendered as HTML:

Notice the syntax for:

- **numbered list** gets automatically sequenced: `1.`, `1.`
- **headers** get rendered at multiple levels: `#`, `##`
- **link**: `[](http://...)`
- **image**: `![](http://...)`
- *italics*: `_word_`

- **bold**: `**word**`

See Mastering Markdown · GitHub Guides and add some more personalized content to the README of your own, like a bulleted list or blockquote.

1.4 Create `index.html`

By default `index.html` is served up. Go ahead and create a new file named `index.html` with the following basic HTML:

```
<!DOCTYPE html>
<html>
<body>

<h1>My First Heading</h1>

<p>My first paragraph.</p>

</body>
</html>
```

You'll be prompted to clone this repository into a folder on your local machine.

See GitHub Desktop User Guides for more. You could also do this from the Bash Shell for Git with the command `git clone https://github.com/USER/REPO.git`, replacing USER with your Github username and REPO with my_project. Or you can use the Github Desktop App menu File -> Clone Repository…

1.5 Create RStudio Project with Git Repository

Next, you will clone the repository onto your local machine using RStudio. I recommend creating it in a folder `github` under your user or Documents folder.

Open RStudio and under the menu File -> New Project… -> Version Control -> git and enter the URL with the `.git` extension (also available from the repository's Clone button):

If it all works correctly then you should see the files downloaded and showing up in the Files pane of RStudio. If RStudio is configured correctly to work with Git, then you should also see a Git pane.

1.6 Create `index.Rmd` in Rmarkdown

Back in RStudio, let's create a new Rmarkdown file, which allows us to weave markdown text with chunks of R code to be evaluated and output content like tables and plots.

File -> New File -> Rmarkdown… -> Document of output format HTML, OK.

You can give it a Title of "My Project". After you click OK, most importantly File -> Save as `index` (which will get named with the filename extension `index.Rmd`).

Some initial text is already provided for you. Let's go ahead and "Knit HTML".

Notice how the markdown is rendered similar to as before + **R code chunks** are surrounded by 3 backticks and `{r LABEL}`. These are evaluated and return the output text in the case of `summary(cars)` and the output plot in the case of `plot(pressure)`.

Notice how the code `plot(pressure)` is not shown in the HTML output because of the R code chunk option `echo=FALSE`.

Before we continue exploring Rmarkdown, visit the Git pane, check all modified (M) or untracked (?) files, click Commit, enter a message like "added index" and click the "Commit" button. Then Push (up green arrow) to push the locally committed changes on your lapto up to the Github repository online. This will update https://github.com/USER/nps-demo, and now you can also see your project website with a default `index.html` viewable at http://USER.github.io/nps-demo

For more on Rmarkdown:

- rmarkdown cheatsheet.pdf
- rmarkdown.rstudio.com
- knitr in a knutshell - Karl Broman

A more advanced topic worth mentioning is dealing merge conflicts

1.7 Exercise: Intertidal Sites Dataset

Gil Rilov shared the following dataset for us to play with:

- Israel sites fall 2015-16.xlsx

Please download and open this dataset. Your task is to investigate this dataset and prepare it for submission to OBIS.

1.7.1 Task: Provide Excel cell ranges for how you would divide data into tables?

For reading and wrangling data in R, please see cheat sheets and resources mentioned in:

- Data Wrangling in the R Tidyverse - Google Slides

Chapter 2

Manipulate: `tidyr`, `dplyr`

> Data scientists, according to interviews and expert estimates, spend from 50 percent to 80 percent of their time mired in the mundane labor of collecting and preparing data, before it can be explored for useful information. - NYTimes (2014)

Today we're going to learn about a package by Hadley Wickham called `dplyr` and how it will help you with simple data exploration, and how you can use it in combination with the `%>%` operator for more complex wrangling (including a lot of the things you would use for loops for.

And we're going to do this in Rmarkdown in the `my-project` repository we created this morning.

Here are the steps:

1. Open RStudio
2. Make sure you're in your `my-project` repo (and if not, get there)
3. New > Rmarkdown… (defaults are fine)
4. Save as `gapminder-dplyr.rmd`
5. Our workflow together will be to write some description of our analysis in Markdown for humans to read, and we will write all of our R code in the 'chunks'. Get ready for the awesomeness, here we go…

Today's materials are again borrowing from some excellent sources, including

- Dr. Jenny Bryan's lectures from STAT545 at UBC: Introduction to dplyr
- Software Carpentry's R for reproducible scientific analysis materials: Dataframe manipulation with dplyr

2.1 install our first package: dplyr

Packages are bundles of functions, along with help pages and other goodies that make them easier for others to use, (ie. vignettes).

So far we've been using packages included in 'base R'; they are 'out-of-the-box' functions. You can also install packages from online. The most traditional is CRAN, the Comprehensive R Archive Network. This is where you went to download R originally, and will go again to look for updates.

You don't need to go to CRAN's website to install packages, we can do it from within R with the command `install.packages("package-name-in-quotes")`.

```
## from CRAN:
#install.packages("dplyr") ## do this once only to install the package on your compute

library(dplyr) ## do this every time you restart R and need it
select <- dplyr::select # overwrite raster::select
```

What's the difference between `install.packages()` and `library()`? Here's my analogy:

- `install.packages()` is setting up electricity for your house. Just need to do this once (let's ignore monthly bills).
- `library()` is turning on the lights. You only turn them on when you need them, otherwise it wouldn't be efficient. And when you quit R, and come back, you'll have to turn them on again with `library()`, but you already have your electricity set up.

2.2 Use dplyr::filter() to subset data row-wise.

First let's read in the gapminder data.

```
# install.packages('gapminder') # instead of reading in the csv
library(gapminder) # this is the package name
str(gapminder) # and it's also the data.frame name, just like yesterday
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    1704 obs. of  6 variables:
##  $ country  : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...
##  $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
##  $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
##  $ pop      : int  8425333 9240934 10267083 11537966 13079460 14880372 12881816 138
##  $ gdpPercap: num  779 821 853 836 740 ...
```

`filter()` takes logical expressions and returns the rows for which all are `TRUE`. Visually, we are doing this (thanks RStudio for your cheatsheet):


```
filter(gapminder, lifeExp < 29)
filter(gapminder, country == "Rwanda")
filter(gapminder, country %in% c("Rwanda", "Afghanistan"))
```

Compare with some base R code to accomplish the same things

```
gapminder[gapminder$lifeExp < 29, ] ## repeat `gapminder`, [i, j] indexing is distracting
subset(gapminder, country == "Rwanda") ## almost same as filter ... but wait ...
```

2.3 Meet the new pipe operator

Before we go any further, we should exploit the new pipe operator that `dplyr` imports from the `magrittr` package by Stefan Bache. **This is going to change your data analytical life**. You no longer need to enact multi-operation commands by nesting them inside each other. This new syntax leads to code that is much easier to write and to read.

Here's what it looks like: `%>%`. The RStudio keyboard shortcut: Ctrl + Shift + M (Windows), Cmd + Shift + M (Mac).

Let's demo then I'll explain:

```
gapminder %>% head
```

This is equivalent to `head(gapminder)`. This pipe operator takes the thing on the left-hand-side and **pipes** it into the function call on the right-hand-side – literally, drops it in as the first argument.

Never fear, you can still specify other arguments to this function! To see the first 3 rows of Gapminder, we could say `head(gapminder, 3)` or this:

```
gapminder %>% head(3)
```

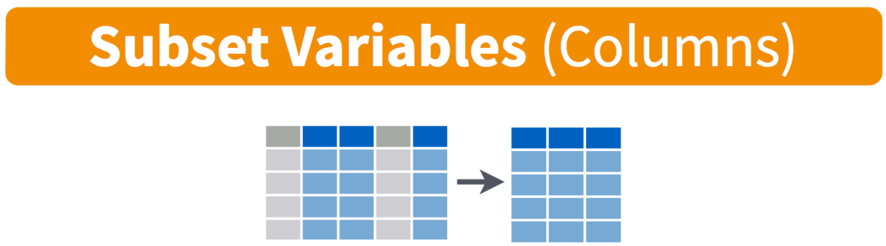
I've advised you to think "gets" whenever you see the assignment operator, `<-`. Similary, you should think "then" whenever you see the pipe operator, `%>%`.

You are probably not impressed yet, but the magic will soon happen.

2.4 Use dplyr::select() to subset the data on variables or columns.

Back to dplyr ...

Use select() to subset the data on variables or columns. Visually, we are doing this (thanks RStudio for your cheatsheet):



Here's a conventional call:

```
select(gapminder, year, lifeExp)
```

But using what we just learned, with a pipe, we can do this:

```
gapminder %>% select(year, lifeExp)
```

Let's write it again but using multiple lines so it's nicer to read. And let's add a second pipe operator to pipe through head:

```
gapminder %>%
  select(year, lifeExp) %>%
  head(4)
```

```
## # A tibble: 4 x 2
##    year lifeExp
##   <int>   <dbl>
## 1  1952    28.8
## 2  1957    30.3
## 3  1962    32.0
## 4  1967    34.0
```

Think: "Take gapminder, then select the variables year and lifeExp, then show the first 4 rows."

2.5 Revel in the convenience

Let's do a little analysis where we calculate the mean gdp for Cambodia.

Here's the gapminder data for Cambodia, but only certain variables:

```
gapminder %>%
  filter(country == "Cambodia") %>%
  # select(country, year, pop, gdpPercap) ## entering 4 of the 6 columns is tedious
  select(-continent, -lifeExp) # you can use - to deselect columns
```

and what a typical base R call would look like:

```
gapminder[gapminder$country == "Cambodia", c("country", "year", "pop", "gdpPercap")]
```

```
## # A tibble: 12 x 4
##    country   year      pop gdpPercap
##    <fct>    <int>    <int>     <dbl>
##  1 Cambodia  1952  4693836      368.
##  2 Cambodia  1957  5322536      434.
##  3 Cambodia  1962  6083619      497.
##  4 Cambodia  1967  6960067      523.
##  5 Cambodia  1972  7450606      422.
##  6 Cambodia  1977  6978607      525.
##  7 Cambodia  1982  7272485      624.
##  8 Cambodia  1987  8371791      684.
##  9 Cambodia  1992 10150094      682.
## 10 Cambodia  1997 11782962      734.
## 11 Cambodia  2002 12926707      896.
## 12 Cambodia  2007 14131858     1714.
```

or, possibly?, a nicer look using base R's `subset()` function:

```
subset(gapminder, country == "Cambodia", select = c(country, year, pop, gdpPercap))
```

```
## # A tibble: 12 x 4
##    country   year      pop gdpPercap
##    <fct>    <int>    <int>     <dbl>
##  1 Cambodia  1952  4693836      368.
##  2 Cambodia  1957  5322536      434.
##  3 Cambodia  1962  6083619      497.
##  4 Cambodia  1967  6960067      523.
##  5 Cambodia  1972  7450606      422.
##  6 Cambodia  1977  6978607      525.
##  7 Cambodia  1982  7272485      624.
##  8 Cambodia  1987  8371791      684.
##  9 Cambodia  1992 10150094      682.
## 10 Cambodia  1997 11782962      734.
```

```
## 11 Cambodia  2002 12926707      896.
## 12 Cambodia  2007 14131858     1714.
```

2.6 Use mutate() to add new variables

Imagine we wanted to recover each country's GDP. After all, the Gapminder data has a variable for population and GDP per capita. Let's add a new column and multiply them together.

Visually, we are doing this (thanks RStudio for your cheatsheet):


```
gapminder %>%
  mutate(gdp = pop * gdpPercap)
```

> Exercise: how would you add that to the previous `filter` and `select` commands we did with Cambodia:

```
gapminder %>%
  filter(country == "Cambodia") %>%
  select(-continent, -lifeExp)
```

Answer:

```
gapminder %>%
  filter(country == "Cambodia") %>%
  select(-continent, -lifeExp) %>%
  mutate(gdp = pop * gdpPercap)
```

2.7 group_by and summarize

Great! And now we want to calculate the mean gdp across all years (Let's pretend that's a good idea statistically)

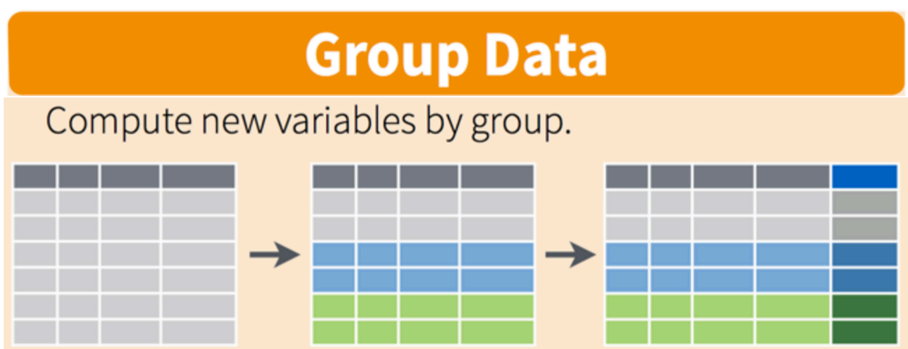
Visually, we are doing this (thanks RStudio for your cheatsheet):


```
gapminder %>%
  filter(country == "Cambodia") %>%
  select(-continent, -lifeExp) %>%
  mutate(gdp = pop * gdpPercap) %>%
  group_by(country) %>%
  summarize(mean_gdp = mean(gdp)) %>%
  ungroup() # if you use group_by, also use ungroup() to save heartache later
```

2.8 Remember our for loop?

And how would you then do this for every country, not just Cambodia? Well, yesterday we would have been thinking about putting this whole analysis inside a for loop, replacing "Cambodia" with a new name each time we iterated through the loop. But today, we have it already, just need to *delete* one line from our analysis–we don't need to filter out Cambodia anymore!!

Visually, we are doing this (thanks RStudio for your cheatsheet):



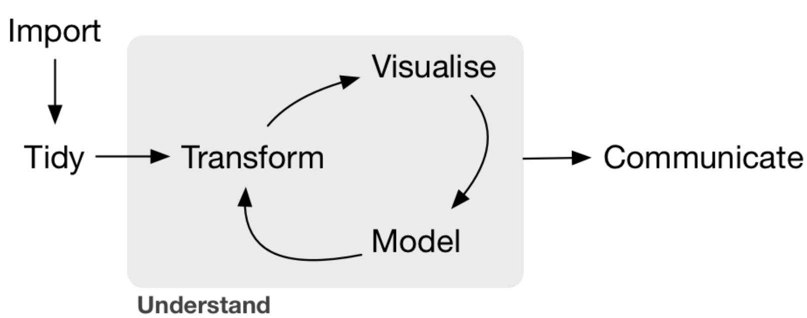

```
gapminder %>%
  select(-continent, -lifeExp) %>%
  mutate(gdp = pop * gdpPercap) %>%
  group_by(country) %>%
  summarize(mean_gdp = mean(gdp)) %>%
  ungroup() # if you use group_by, also use ungroup() to save heartache later
```

So we have done a pretty incredible amount of work in a few lines. Our whole analysis is this. Imagine the possibilities from here.

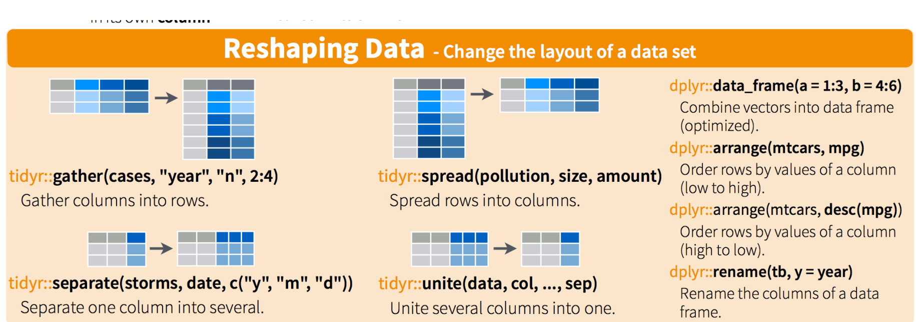
```
library(dplyr)

gapminder %>%
  read.csv('data/gapminder-FiveYearData.csv') %>%
  select(-continent, -lifeExp) %>%
  mutate(gdp = pop * gdpPercap) %>%
  group_by(country) %>%
  summarize(mean_gdp = mean(gdp)) %>%
  ungroup() # if you use group_by, also use ungroup() to save heartache later
```

2.9 Summary

This has been the 'Tranform' or Wrangling part of this cycle.



Importing and tidying is also a HUGE part of this process, and we don't have time to get into it today. But look at the cheatsheet, and watch the webinar. cheatsheet and webinar. Watch this 1 hour webinar and follow along in RStudio and your science will be forever changed. Again!



2.10 Further materials as reference...

2.10.1 Rationale

When performing data analysis in R, code can become quite messy, making it hard to revisit and determine the sequence of operations. Commenting helps. Good variable names help. Still, at least two common issues make code difficult to understand: **multiple variables** and **nested functions**. Let's examine these issues by approaching an analysis presenting both problems, and finally see how `dplyr` offers an elegant alternative.

For example, let's ask of the `surveys.csv` dataset: ***How many observations of a certain thing you're interested in appear each year?***

2.10.2 Pseudocode

You can write the logic out as **pseudocode** which can become later comments for the actual code:

```
# read in csv
# view data
# limit columns to species and year
# limit rows to just species "NL"
# get count per year
# write out csv
```

2.10.3 Summary

The `tidyr` and `dplyr` packages were created by Hadley Wickham of `ggplot2` fame. The "gg" in `ggplot2` stands for the "grammar of graphics". Hadley similarly considers the functionality of the two packages `dplyr` and `tidyr` to provide the "grammar of data manipulation".

Next, we'll explore the data wrangling lessons that Remi contributed to Software Carpentry.

2.10.4 `dplyr`

dplyr - Software Carpentry

2.10.5 `tidyr`

tidyr - Software Carpentry

2.10.6 Other links

- Tidying up Data - Env Info - Rmd
- Data wrangling with dplyr and tidyr - Tyler Clavelle & Dan Ovando - Rmd

Chapter 3

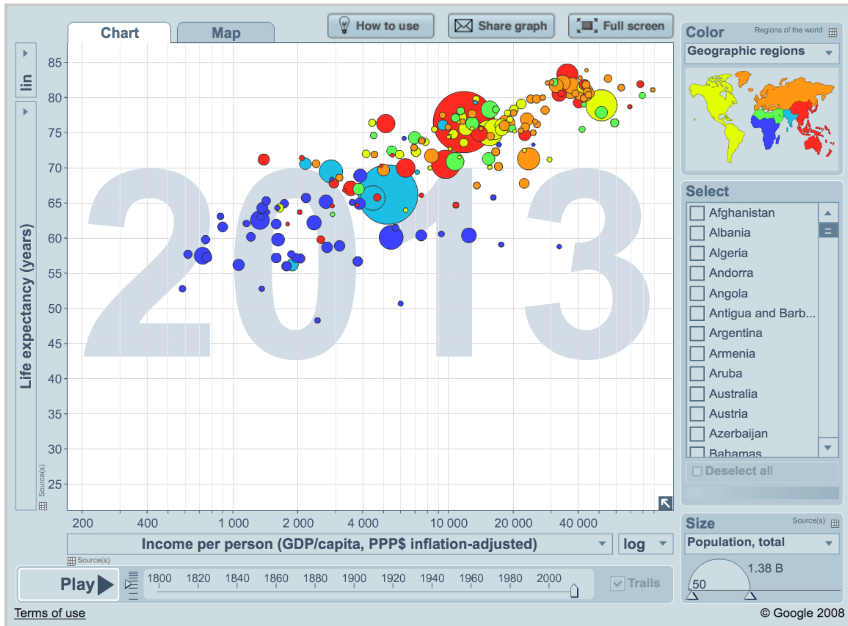
Visualize: `ggplot2`, `plotly`, `tmap`

3.1 Plot

Inspiring people:

- **Hadley Wickham**: grammar of graphics
- **Hans Rosling**: Gapminder

Gapminder World - Wealth & Health of Nations



3.1.1 Static: ggplot

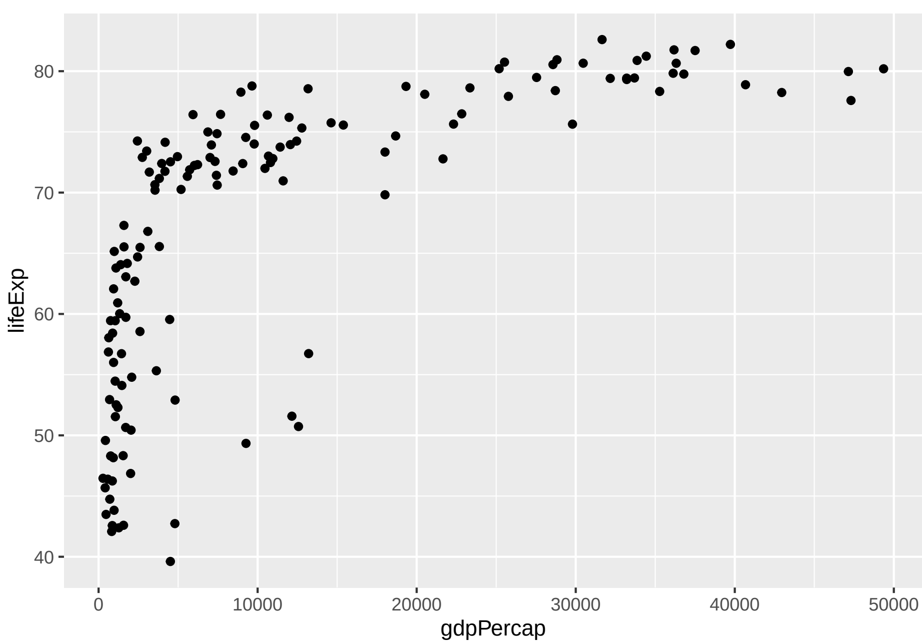
- Creating publication quality graphics - Software Carpentry

3.1.1.1 Scatterplot

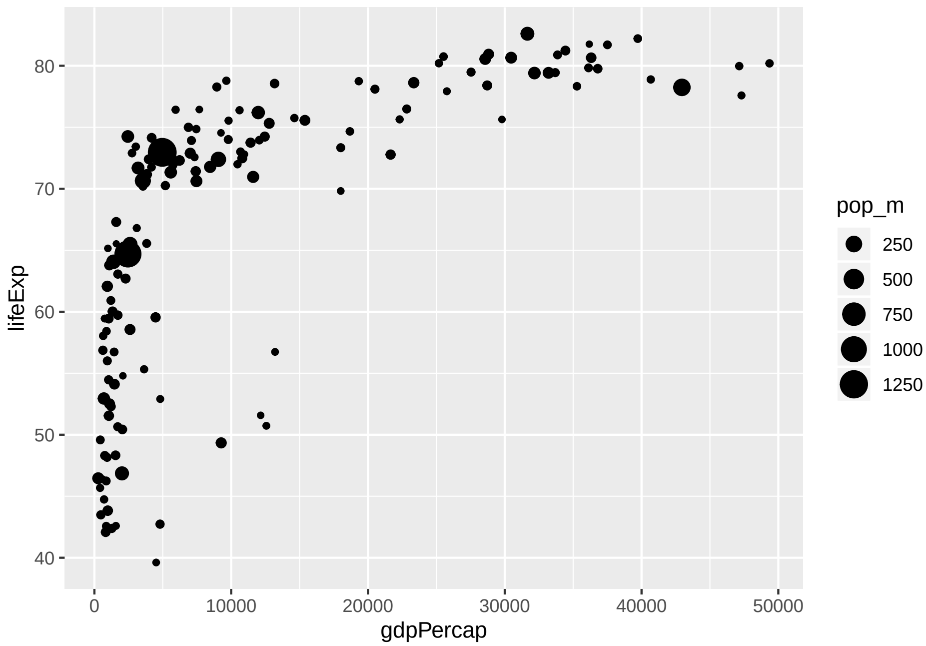
```
library(dplyr)
library(ggplot2)
library(gapminder)

# preview data
gapminder

# get range of available data
summary(gapminder)
```

```
# setup dataframe
g = gapminder %>%
  filter(year==2007) %>%   # most recent year
  mutate(pop_m = pop/1e6)  # population, millions

# plot scatterplot of most recent year
s = ggplot(g, aes(x=gdpPercap, y=lifeExp)) +
```

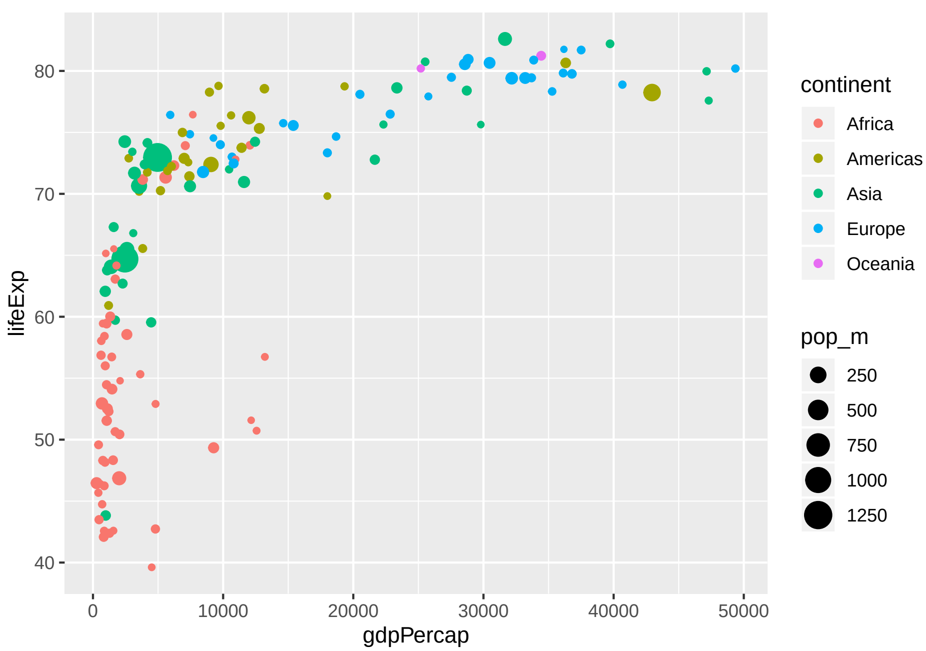
```
  geom_point()
s
```



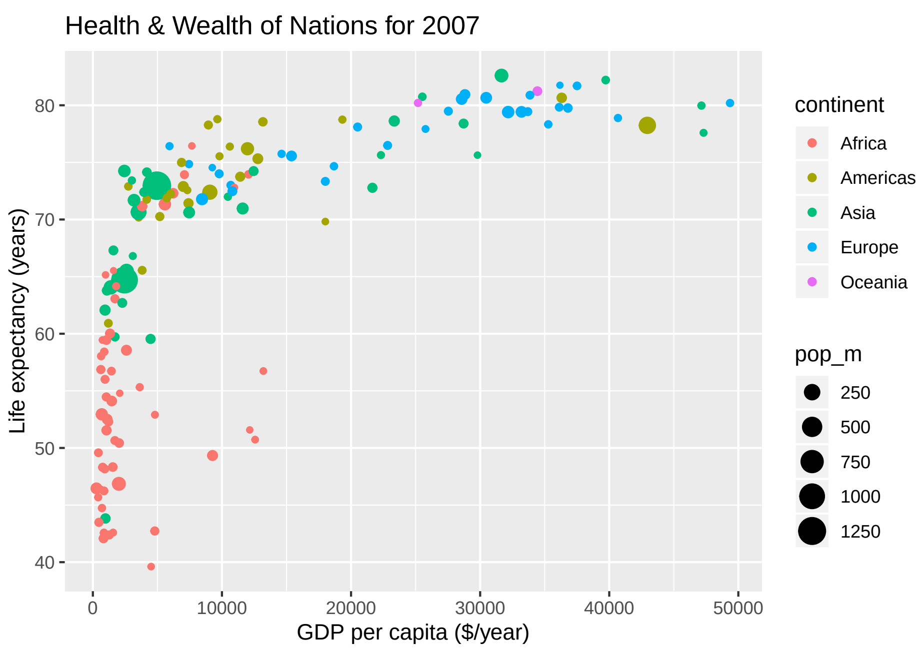

```
# add aesthetic of size by population
s = s +
  aes(size=pop_m)
s
```



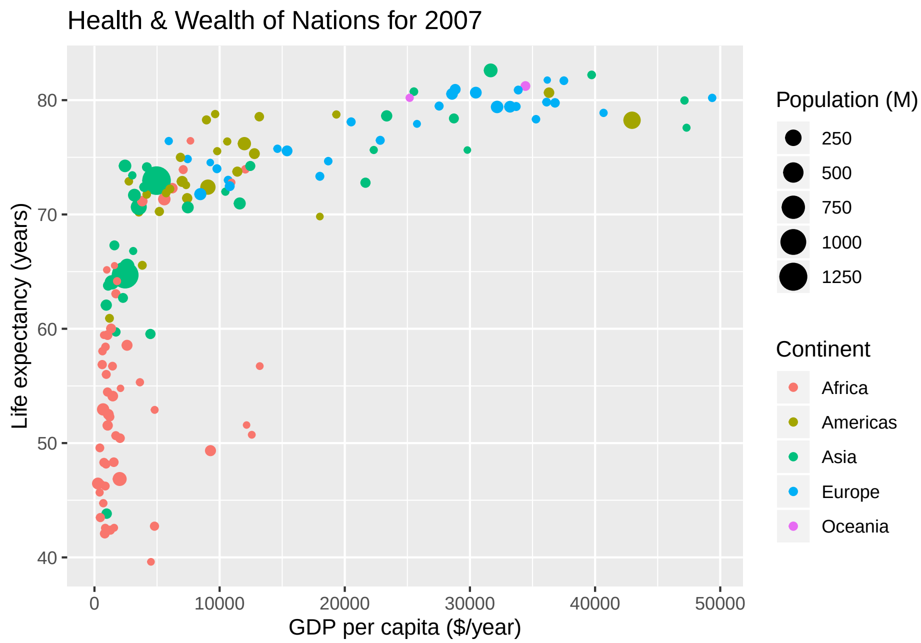

```r
# add aesthetic of color by continent
s = s +
  aes(color=continent)
s
```



```
# add title, update axes labels
s = s +
  ggtitle('Health & Wealth of Nations for 2007') +
  xlab('GDP per capita ($/year)') +
  ylab('Life expectancy (years)')
s
```




```
# label legend
s = s +
  scale_colour_discrete(name='Continent') +
  scale_size_continuous(name='Population (M)')
s
```



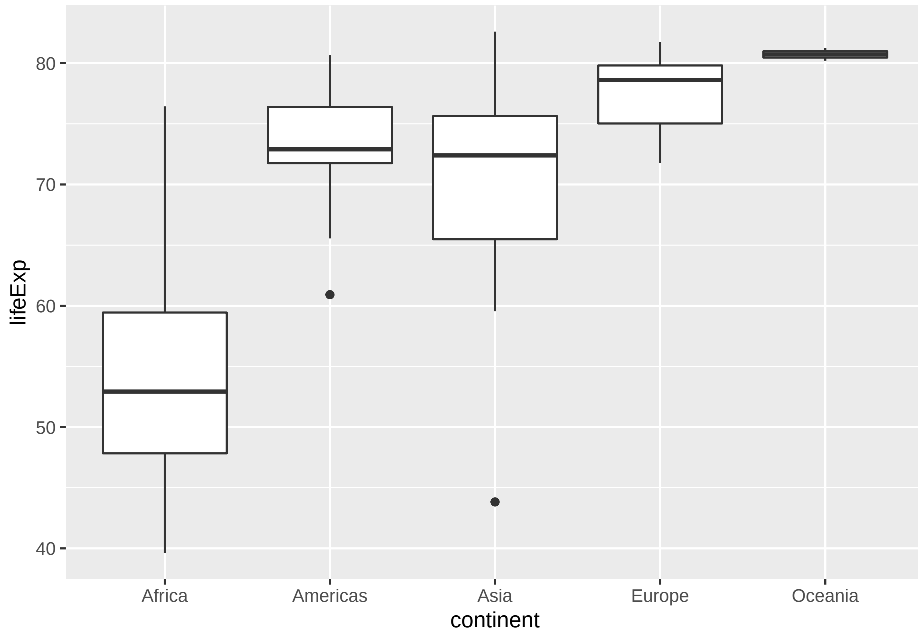
Your Turn

Now with country emissions datasets...

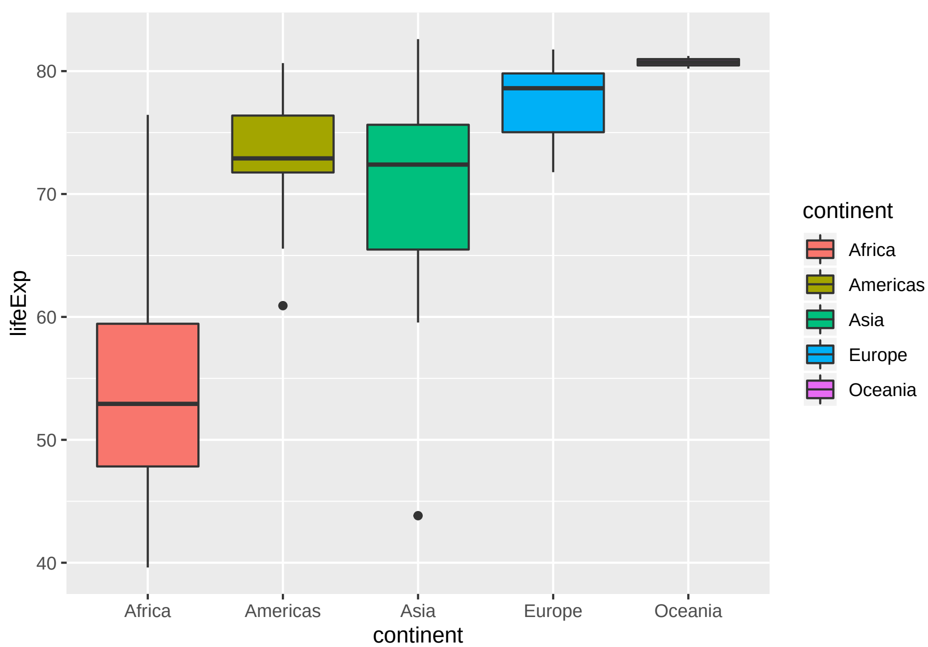
- CO2 Emissions from Fossil Fuels since 1751, By Nation - Dataset - Frictionless Open Data
- datasets/gdp

3.1.1.2 Boxplot

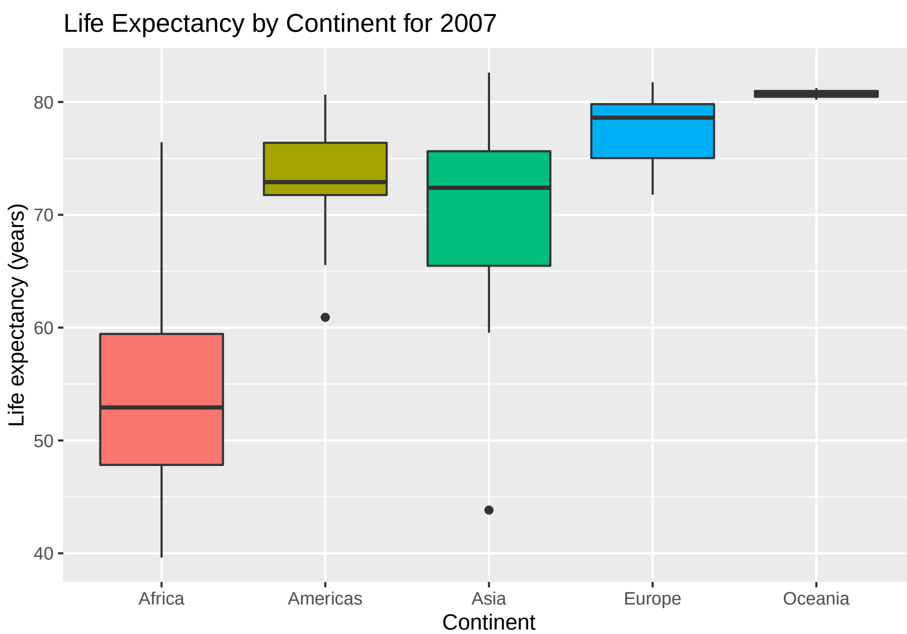
```
# boxplot by continent
b = ggplot(g, aes(x=continent, y=lifeExp)) +
  geom_boxplot()
b
```




```r
# match color to continents, like scatterplot
b = b +
  aes(fill=continent)
b
```



```
# drop legend, add title, update axes labels
b = b +
  theme(legend.position='none') +
  ggtitle('Life Expectancy by Continent for 2007') +
  xlab('Continent') +
  ylab('Life expectancy (years)')
b
```

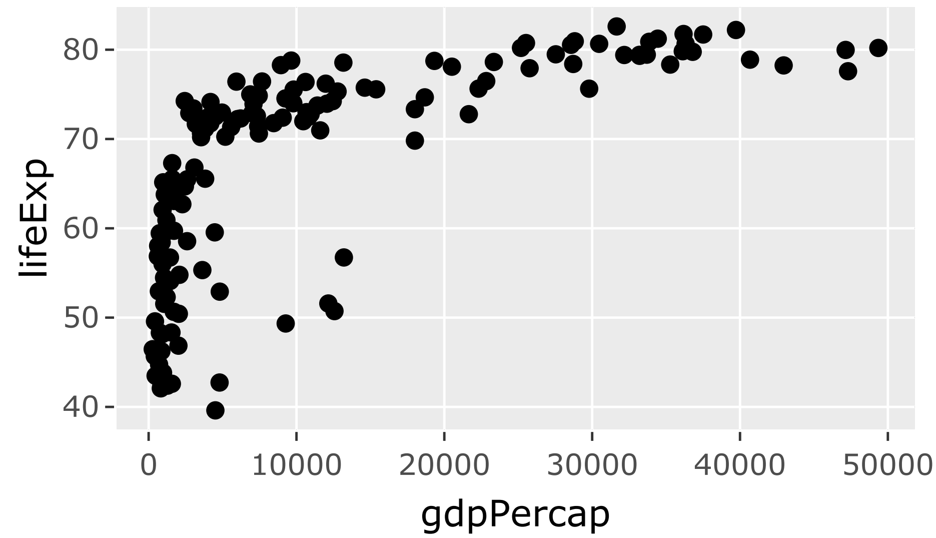


Your Turn: Make a similar plot but for `gdpPercap`. Be sure to update the plot's aesthetic, axis label and title accordingly.

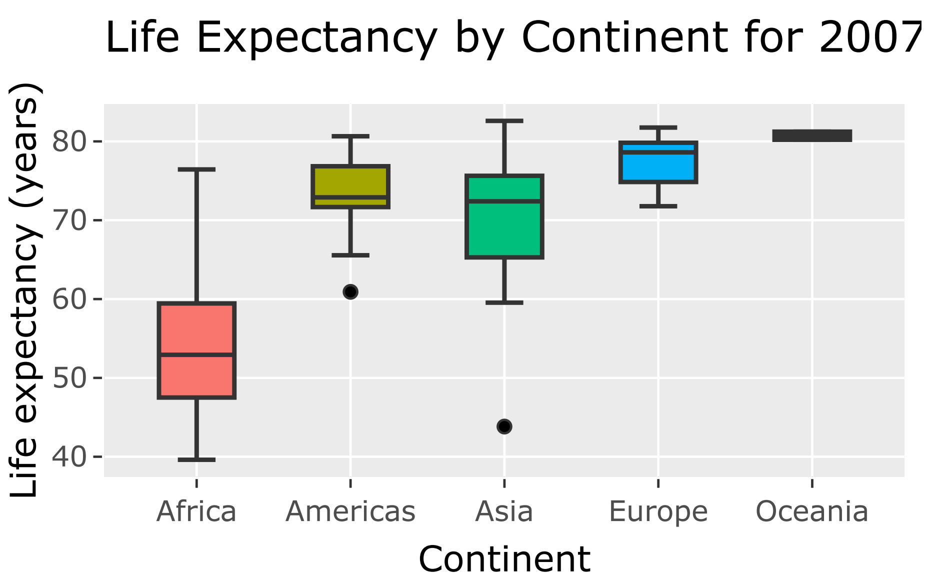
3.1.2 Interactive: `plotly`

ggplot2 | plotly

```
library(plotly) # install.packages('plotly')

# scatterplot (Note: key=country shows up on rollover)
s = ggplot(g, aes(x=gdpPercap, y=lifeExp, key=country)) +
  geom_point()
ggplotly(s)
```



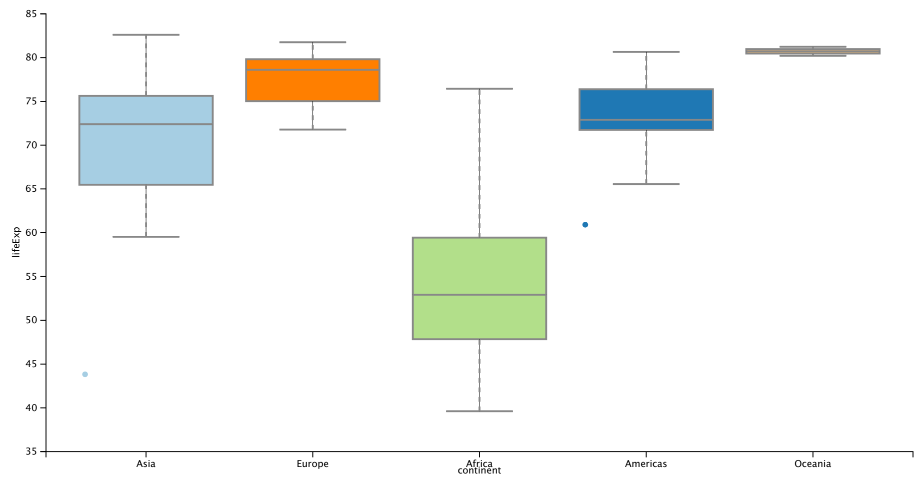

```
# boxplot
ggplotly(b)
```



Your Turn: Expand the interactive scatterplot to include all the other bells and whistles of the previous plot in one continuous set of code (no in between setting of s).

3.1.3 Interactive: Exploding Boxplot

```
library(explodingboxplotR) # devtools::install_github('timelyportfolio/explodingboxplo

exploding_boxplot(g,
  y = 'lifeExp',
  group = 'continent',
  color = 'continent',
  label = 'country')
```



3.1.4 Interactive: Motion Plot

The `googleVis` package ports most of the Google charts functionality.

For every R chunk must set option `results='asis'`, and once before any googleVis plots, set `op <- options(gvis.plot.tag='chart')`.

- Rmarkdown and googleVis
- googleVis examples

```
suppressPackageStartupMessages({
  library(googleVis) # install.packages('googleVis')
})
op <- options(gvis.plot.tag='chart')

m = gvisMotionChart(
  gapminder %>%
    mutate(
      pop_m = pop / 1e6,
```

```
      log_gdpPercap = log(gdpPercap)),
  idvar='country',
  timevar='year',
  xvar='log_gdpPercap',
  yvar='lifeExp',
  colorvar='continent',
  sizevar='pop_m')
plot(m)
```

Your Turn: Repeat the motion chart with the country having the highest `gdpPercap` filtered out.

3.2 Map

Thematic maps **`tmap`**:

- tmap in a nutshell
- tmap modes: plot and interactive view

3.2.1 Static

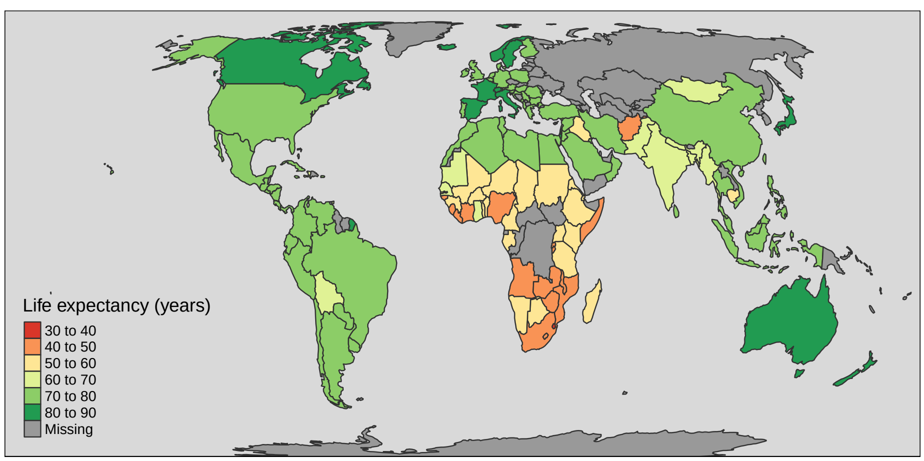
```
library(sf)
library(tmap) # install.packages('tmap')

# load world spatial polygons
data(World)

# inspect values in World
World %>% st_set_geometry(NULL)

# gapminder countries not in World. skipping for now
g %>%
  anti_join(World, by=c('country'='name')) %>%
  arrange(desc(pop))

# World countries not in gapminder. skipping for now
World %>%
  anti_join(g, by=c('name'='country')) %>%
  arrange(desc(pop_est)) %>%
  select(iso_a3, name, pop_est)

# join gapminder data to World
```

```
World = World %>%
  left_join(g, by=c('name'='country'))
```

```
# make map
m = tm_shape(World) +
    tm_polygons('lifeExp', palette='RdYlGn', id='name', title='Life expectancy (years)
    tm_style_gray() + tm_format_World()
```

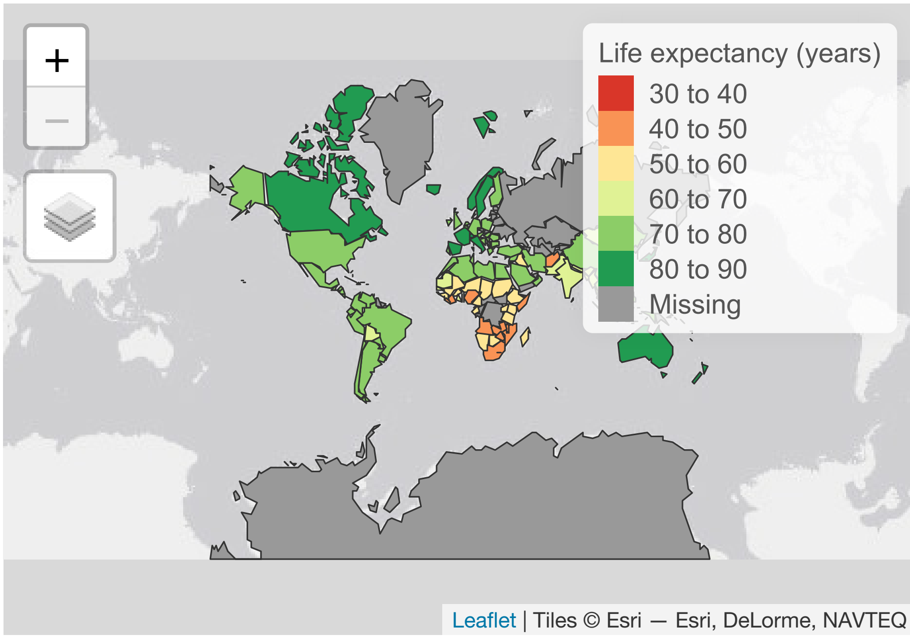
```
## Warning: The argument auto.palette.mapping is deprecated. Please use midpoint
## for numeric data and stretch.palette for categorical data to control the palette
## mapping.

## Warning in tm_style_gray(): tm_style_gray is deprecated as of tmap version 2.0.
## Please use tm_style("gray", ...) instead

## Warning in tm_format_World(): tm_format_World is deprecated as of tmap version
## 2.0. Please use tm_format("World", ...) instead
```

```
m
```



3.2.2 Interactive

```
# show interactive map
tmap_leaflet(m)
```



3.3 References

- ggplot2-cheatsheet-2.0.pdf
- Interactive Plots and Maps - Environmental Informatics
- Graphs with ggplot2 - Cookbook for R
- ggplot2 Essentials - STHDA
- NEON Working with Geospatial Data

Chapter 4

Analyze Spatial: `sf`

4.1 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.

4.2 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.

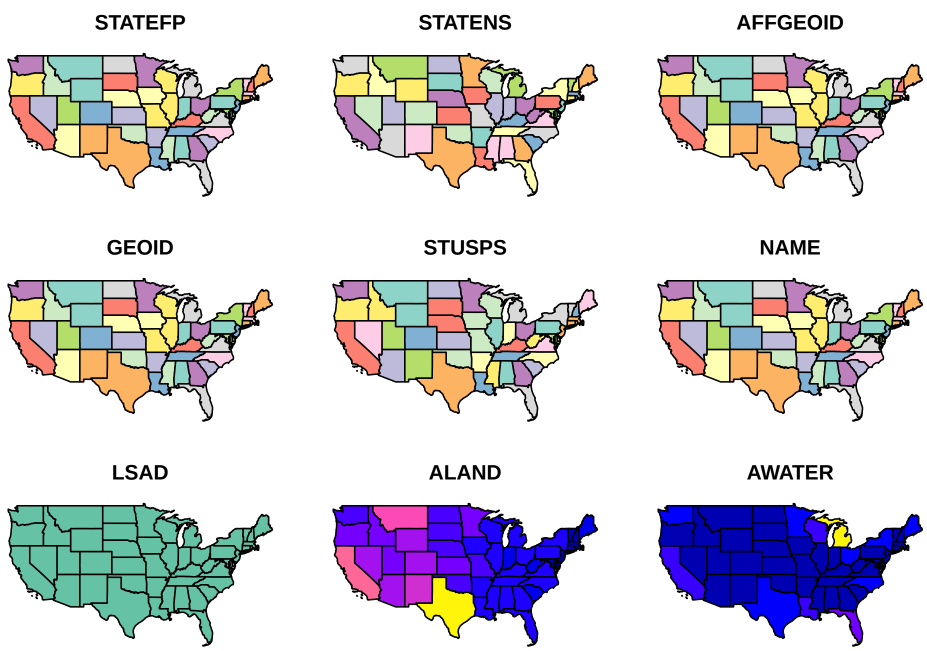
```
# load packages
library(tidyverse)  # load dplyr, tidyr, ggplot2 packages
library(sf)         # vector reading & analysis

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

4.3 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.

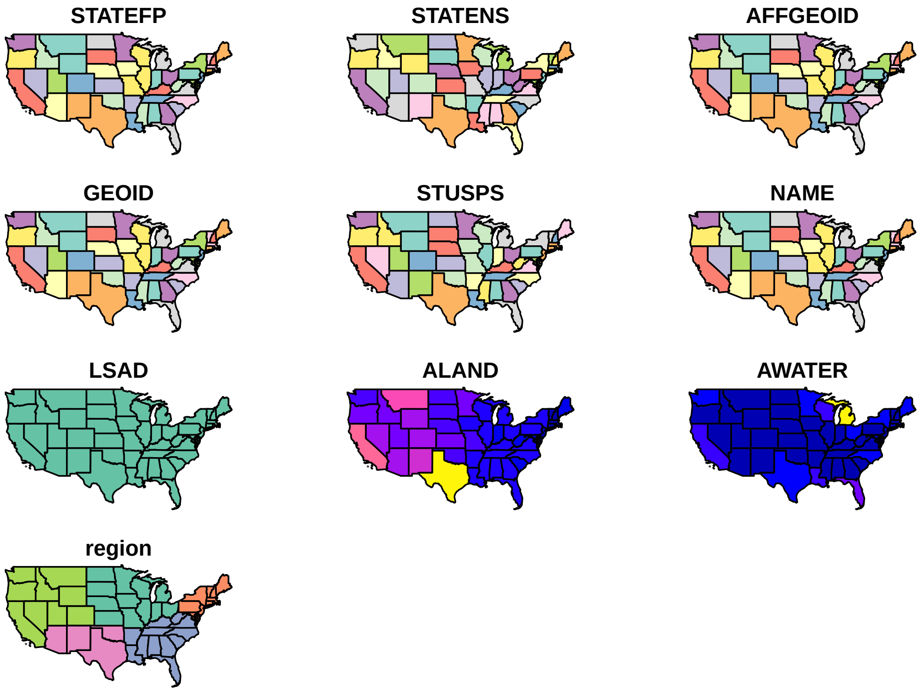
```
library(here)

states_shp <- here("data/neon-us-boundary/US-State-Boundaries-Census-2014.shp")

# read in states
states <- read_sf(states_shp)

# 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.

```
# 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", "24"...
## $ STATENS  <chr> "01779778", "01702382", "00294478", "01705317", "01779783"...
## $ AFFGEOID <chr> "0400000US06", "0400000US11", "0400000US12", "0400000US13"...
## $ GEOID    <chr> "06", "11", "12", "13", "16", "17", "19", "21", "22", "24"...
## $ STUSPS   <chr> "CA", "DC", "FL", "GA", "ID", "IL", "IA", "KY", "LA", "MD"...
## $ NAME     <chr> "California", "District of Columbia", "Florida", "Georgia"...
## $ LSAD     <chr> "00", "00", "00", "00", "00", "00", "00", "00", "00", "00"...
## $ ALAND    <dbl> 403483823181, 158350578, 138903200855, 148963503399, 21404...
## $ AWATER   <dbl> 20483271881, 18633500, 31407883551, 4947080103, 2397728105...
## $ region   <chr> "West", "Northeast", "Southeast", "Southeast", "West", "Mi...
## $ geometry <MULTIPOLYGON [°]> MULTIPOLYGON Z (((-118.594 ..., MULTIPOLYGON ...
```

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

```
## # A tibble: 58 x 11
##    STATEFP STATENS AFFGEOID GEOID STUSPS NAME  LSAD    ALAND  AWATER region
##    <chr>   <chr>   <chr>    <chr> <chr>  <chr> <chr>   <dbl>   <dbl> <chr>
##  1 06      017797~ 0400000~ 06    CA     Cali~ 00    4.03e11 2.05e10 West
##  2 11      017023~ 0400000~ 11    DC     Dist~ 00    1.58e 8 1.86e 7 North~
##  3 12      002944~ 0400000~ 12    FL     Flor~ 00    1.39e11 3.14e10 South~
##  4 13      017053~ 0400000~ 13    GA     Geor~ 00    1.49e11 4.95e 9 South~
##  5 16      017797~ 0400000~ 16    ID     Idaho 00    2.14e11 2.40e 9 West
##  6 17      017797~ 0400000~ 17    IL     Illi~ 00    1.44e11 6.20e 9 Midwe~
##  7 19      017797~ 0400000~ 19    IA     Iowa  00    1.45e11 1.08e 9 Midwe~
##  8 21      017797~ 0400000~ 21    KY     Kent~ 00    1.02e11 2.39e 9 South~
##  9 22      016295~ 0400000~ 22    LA     Loui~ 00    1.12e11 2.38e10 South~
## 10 24      017149~ 0400000~ 24    MD     Mary~ 00    2.51e10 6.98e 9 North~
## # ... with 48 more rows, and 1 more variable: geometry <MULTIPOLYGON [°]>
```

```r
names(states)
```

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

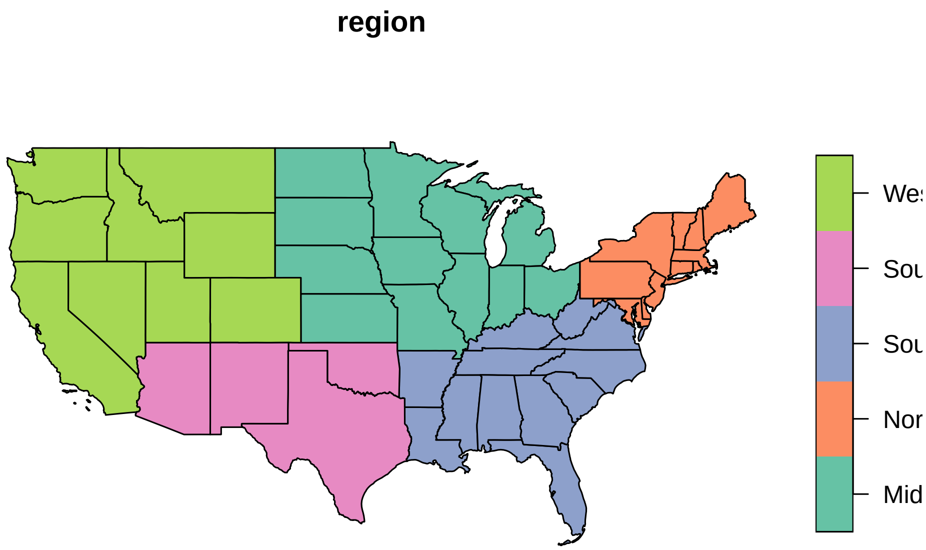
```r
# inspect the class(es) of the states object
class(states)
```

```
## [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?***"

4.4 Challenge: analytical steps?

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

4.4.1 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.

4.5 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                                 geometry pct_water
## * <chr>       <dbl>     <dbl>                           <GEOMETRY [°]>     <dbl>
## 1 Midwest   1.84e11   1.94e12 MULTIPOLYGON Z (((-82.86334 41.69369 0,~      9.49
## 2 Northe~   1.09e11   8.69e11 MULTIPOLYGON Z (((-76.04621 38.02553 0,~     12.5 
## 3 Southe~   1.04e11   1.36e12 MULTIPOLYGON Z (((-81.81169 24.56874 0,~      7.61
## 4 Southw~   2.42e10   1.46e12 POLYGON Z ((-94.48587 33.63787 0, -94.4~      1.66
## 5 West      5.76e10   2.43e12 MULTIPOLYGON Z (((-118.594 33.03595 0, ~      2.37
```

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))`.

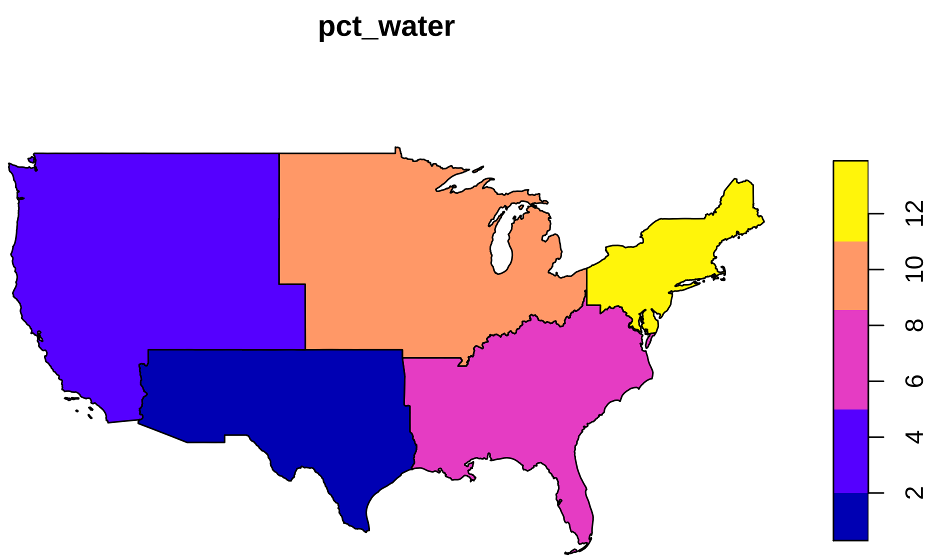
```
# table
regions %>%
  st_set_geometry(NULL) %>%
  arrange(desc(pct_water))
```

```
## # A tibble: 5 x 4
##   region           water          land pct_water
##   <chr>            <dbl>         <dbl>     <dbl>
## 1 Northeast 108922434345  869066138232     12.5 
## 2 Midwest   184383393833 1943869253244      9.49
## 3 Southeast 103876652998 1364632039655      7.61
## 4 West       57568049509 2432336444730      2.37
## 5 Southwest  24217682268 1462631530997      1.66
```

4.6 Regions: plot

Now plot the regions.

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

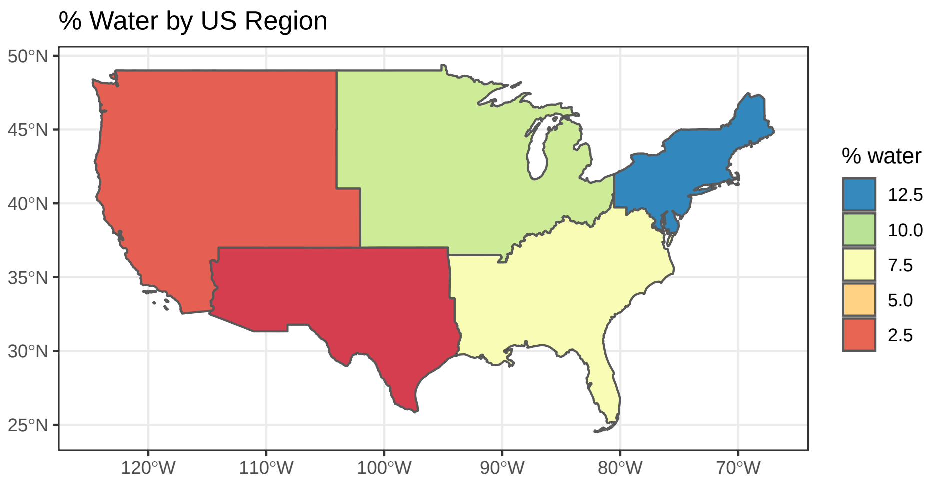


4.7 Regions: ggplot

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")
```



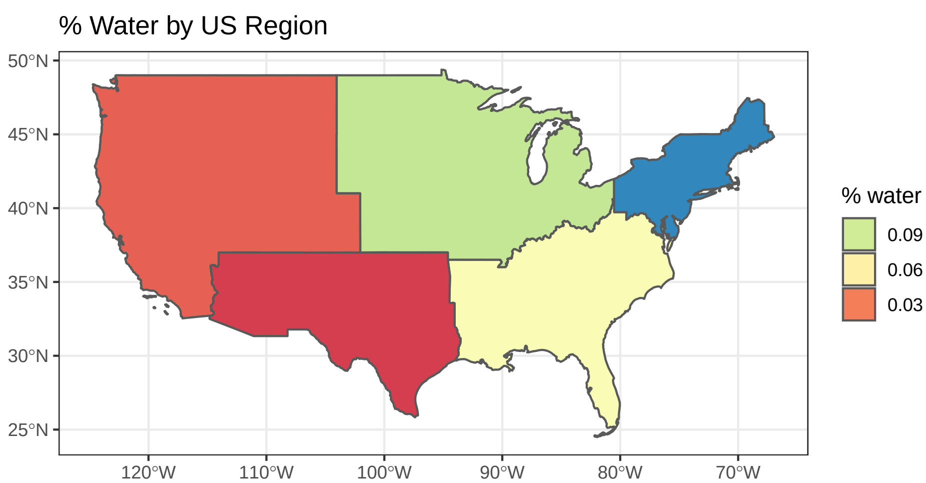
4.8 Regions: recalculate area

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>             [m^2]         [m^2]        [1]
## 1 Northeast 108922434345 9.117041e+11 0.11947126
## 2 Midwest   184383393833 1.987268e+12 0.09278233
## 3 Southeast 103876652998 1.427079e+12 0.07278971
## 4 West       57568049509 2.467170e+12 0.02333363
## 5 Southwest  24217682268 1.483765e+12 0.01632178
```

```r
# 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")
```



4.9 Challenge: project & recalculate area

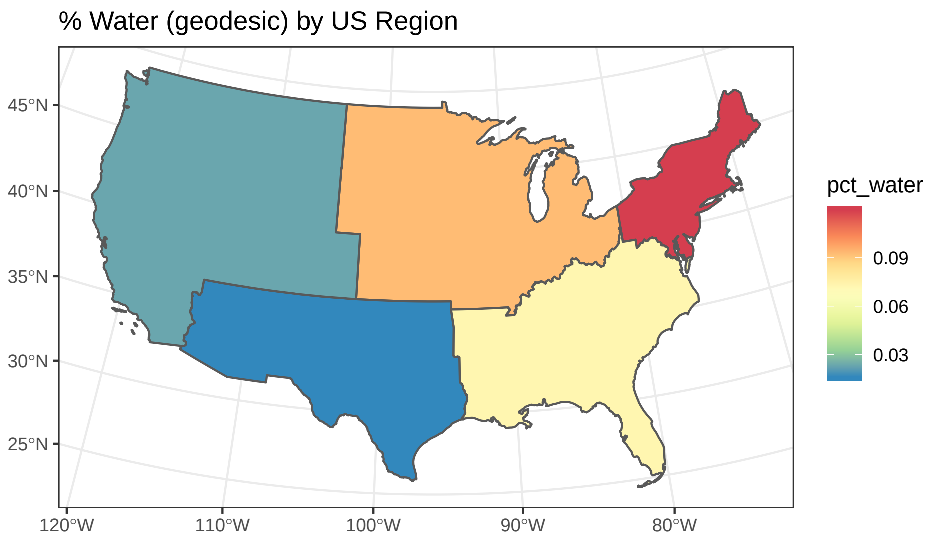
Use `st_transform()` with a USA Contiguous Albers Equal Area Conic Projection that minimizes distoration, and then calculate area using the `st_area()` function.

4.9.1 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 +ell

regions = states %>%
  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>            [m^2]        [m^2]        [1]
## 1 Northeast 108922434345 9.117031e+11 0.11947138
## 2 Midwest   184383393833 1.987266e+12 0.09278246
## 3 Southeast 103876652998 1.427078e+12 0.07278973
## 4 West       57568049509 2.467167e+12 0.02333367
## 5 Southwest  24217682268 1.483758e+12 0.01632185
```

```
# 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")
```



4.10 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 5

Interactive Map: `leaflet`

5.1 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

5.2 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.

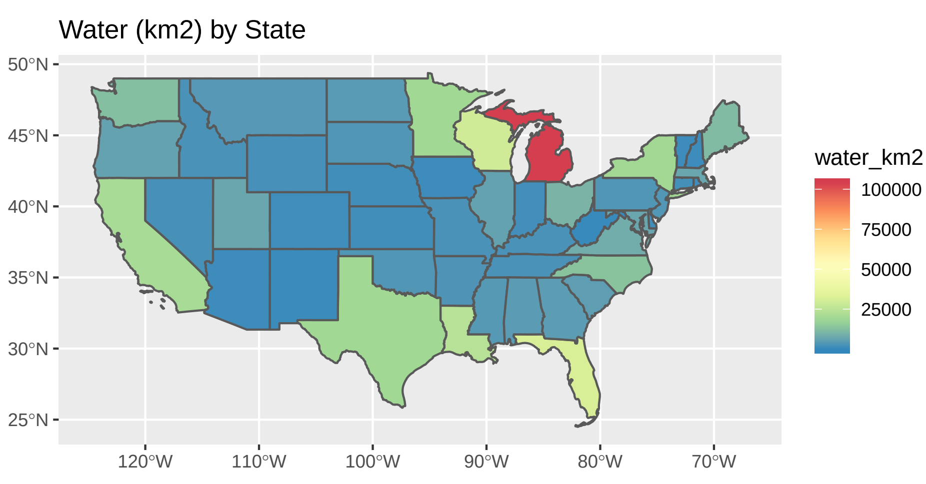
```
# load packages
library(tidyverse)  # loads dplyr, tidyr, ggplot2 packages
library(sf)         # simple features package - vector
library(raster)     # raster
library(plotly)     # makes ggplot objects interactive
library(mapview)    # quick interactive viewing of spatial objects
```

```
library(leaflet)     # interactive maps

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

5.3 States: ggplot2

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

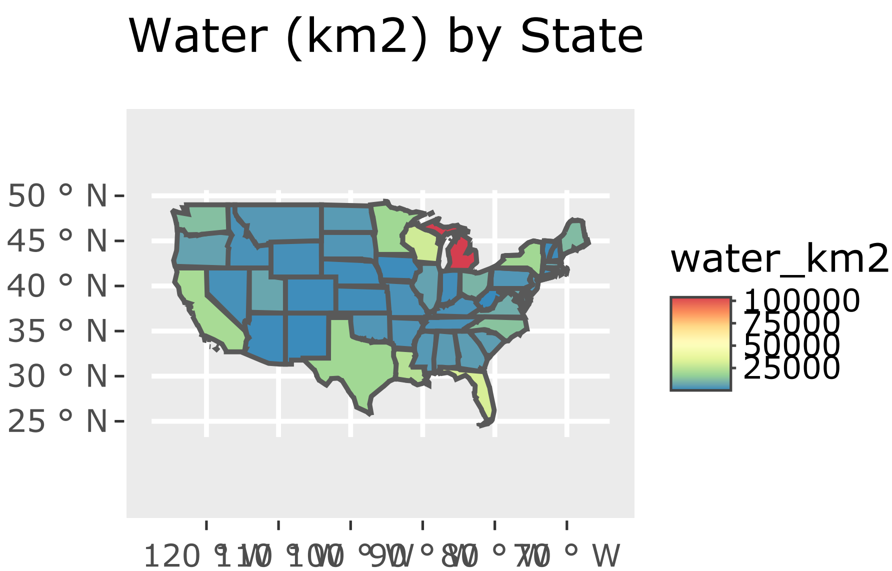
```
library(here)

states_shp <- here("data/neon-us-boundary/US-State-Boundaries-Census-2014.shp")

# read in states
states <- read_sf(states_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
```



5.4 States: plotly

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

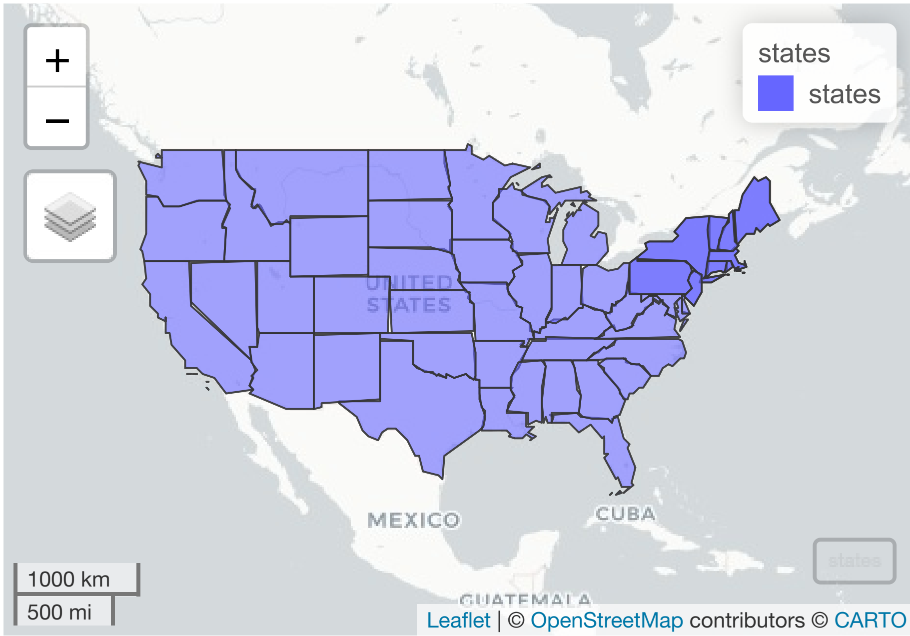
```r
library(plotly)

ggplotly(g)
```



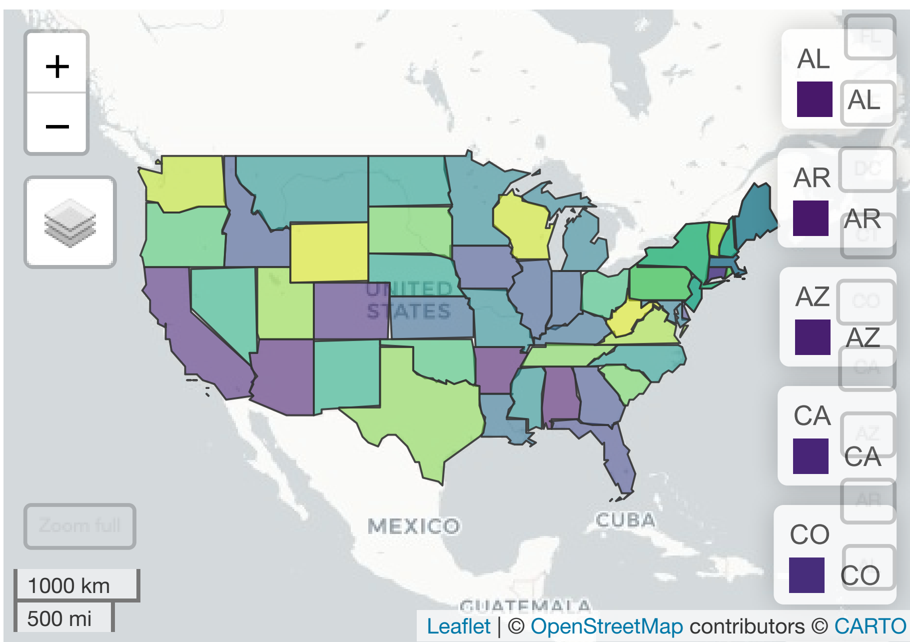
5.5 States: mapview

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.

```r
library(mapview)

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



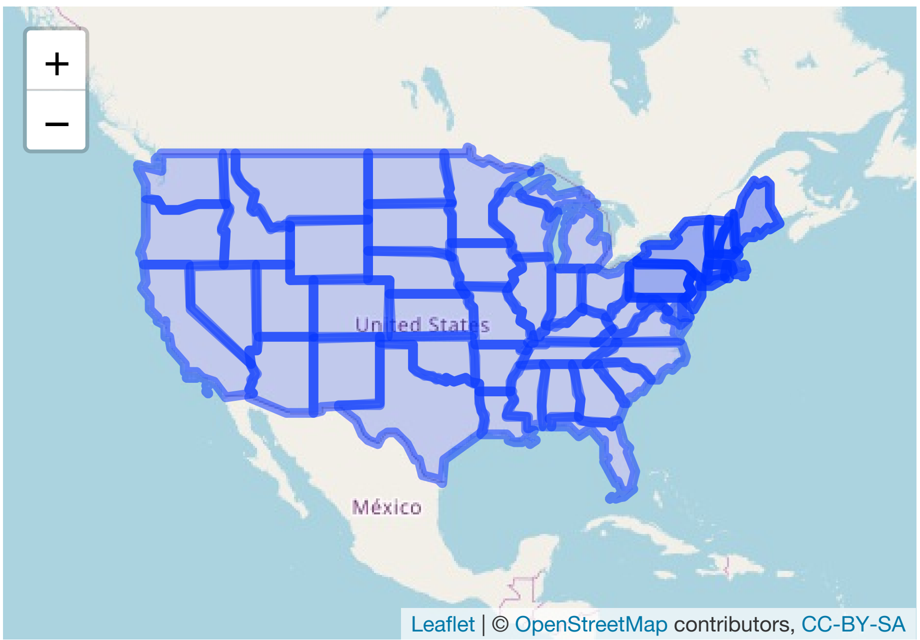

```
# coloring and layering
mapview(states, zcol='water_km2', burst='STUSPS')
```



5.6 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()
```

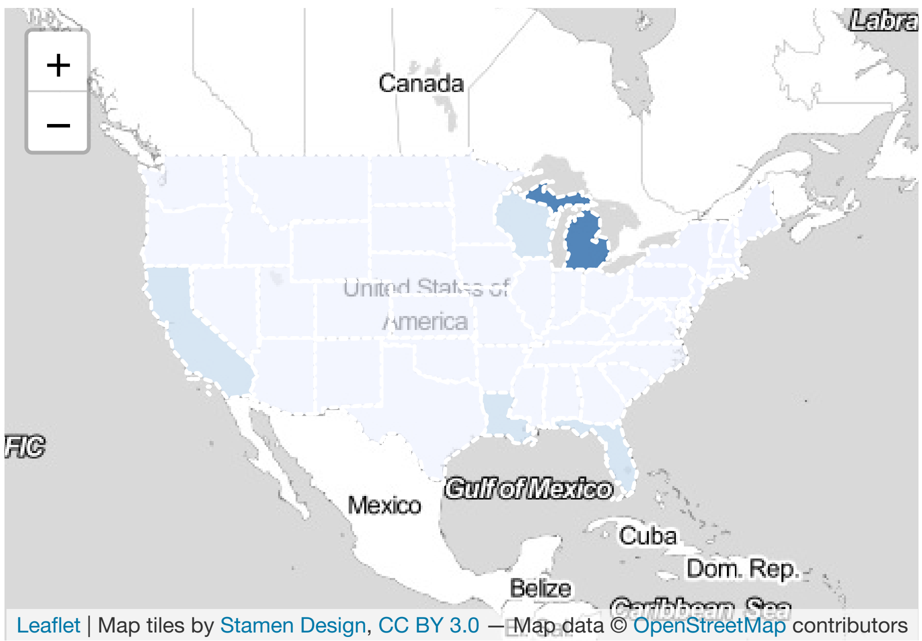


5.6.1 Choropleth

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))
```

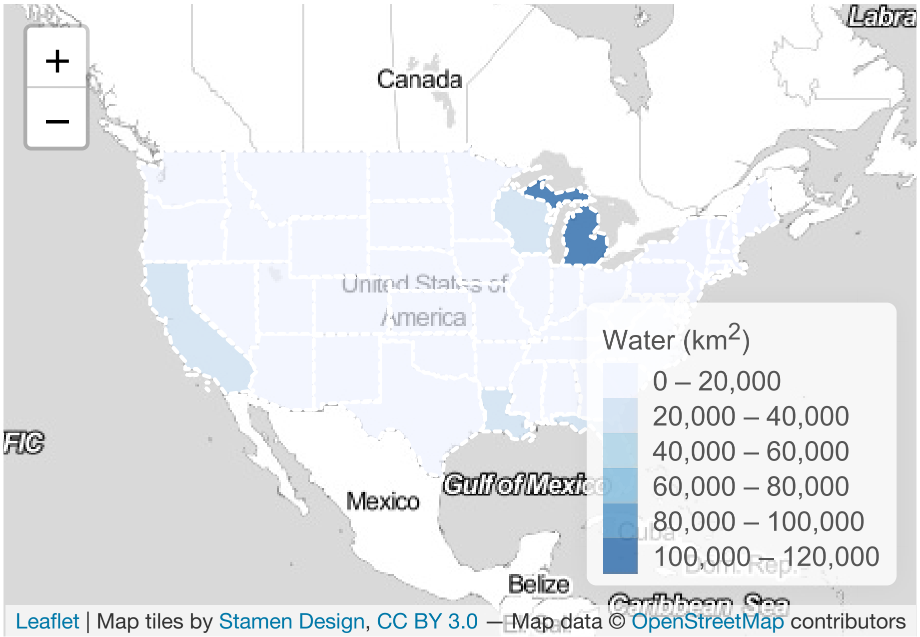


5.6.2 Popups and Legend

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")
```



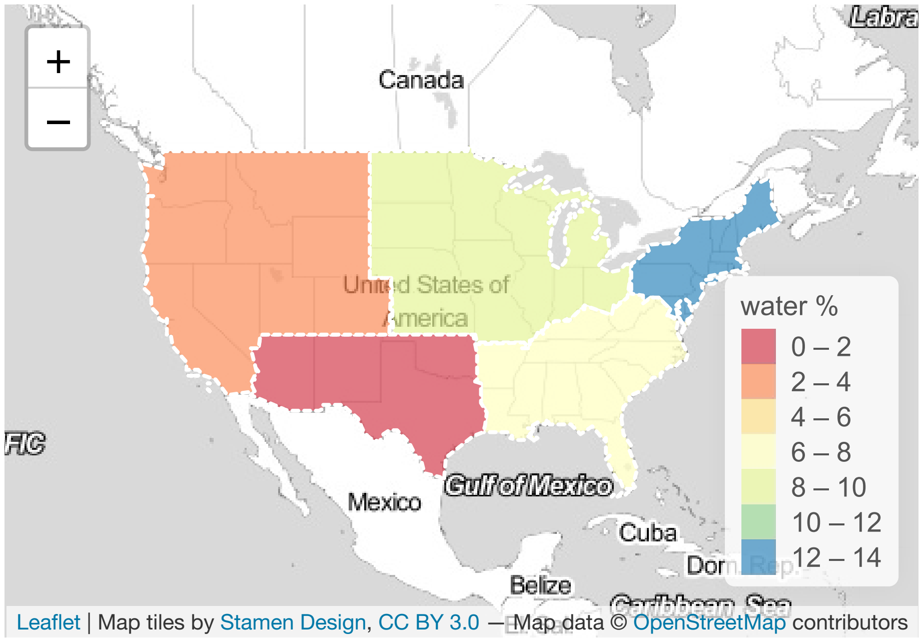
5.7 Challenge: leaflet for regions

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

5.7.1 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")
```



5.8 Raster: leaflet

TODO: show raster overlay using NEON raster dataset example

5.9 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.