Introduction to R & R Markdown

DATA350: Data Visualization and Presentation

Fushuai Jiang

2025-08-13

Syllabus

- *Instructor*: Dr. Fushuai Jiang
- *TA*: TBD
- *Office Hours*: TBD

Starting Texts (click the book titles for the links)

- Healy, Kieran, Data visualization: a practical introduction
- Wickham, Hadley, Mine Çetinkaya-Rundel, and Garrett Grolemund, R for data science
- (Recommended for formatting your work) Xie, Yihui, Joseph J. Allaire, and Garrett Grolemund, R markdown: The definitive guide
- More as we go along (but all will be free)

What is R?

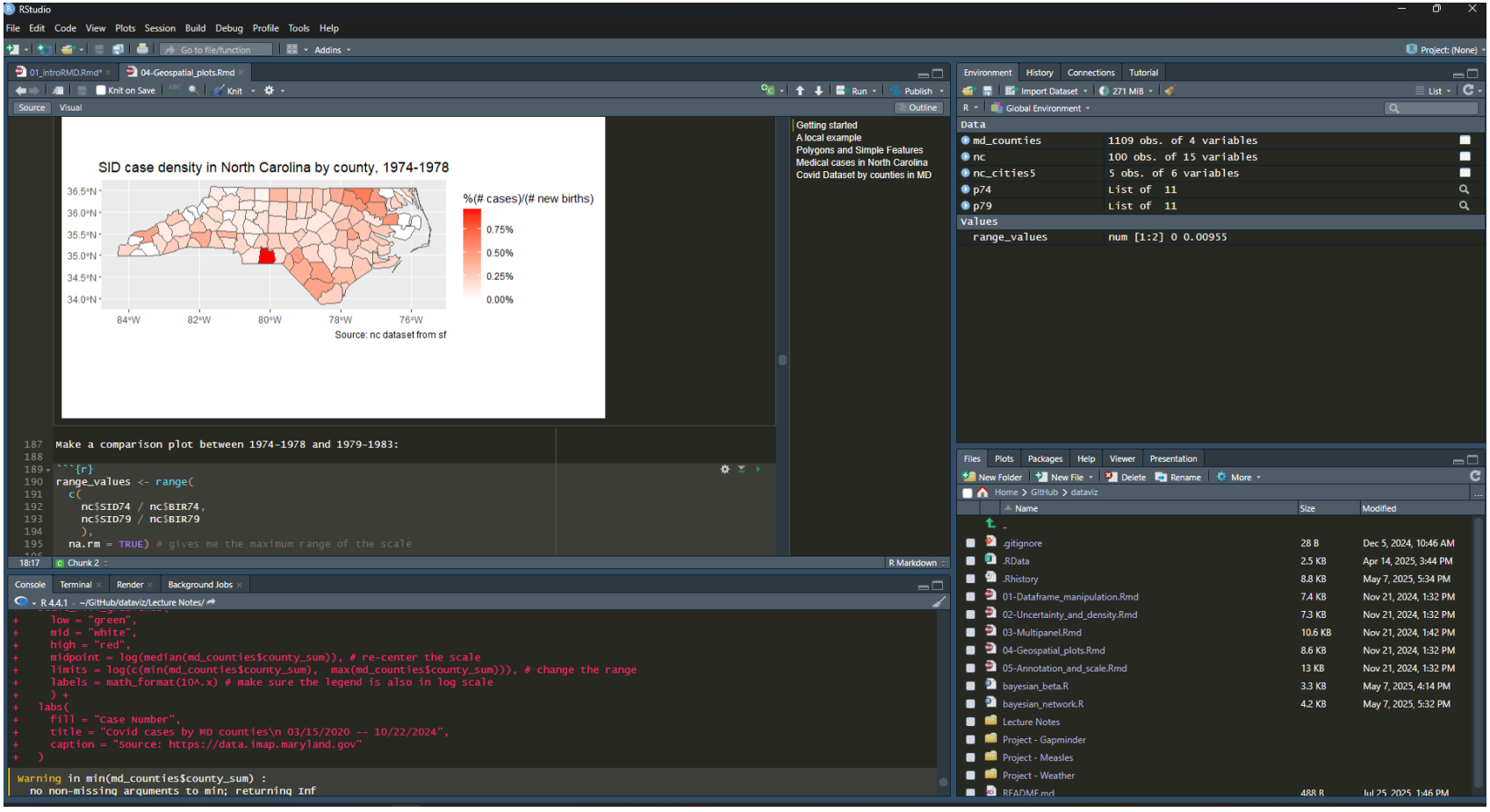
R is a powerful open-source software system that provides an extensive and coherent collection of tools for statistics and data analysis.

- Language and tools for representing and manipulating statistical models
- *POWERFUL* and *QUICK* graphic capabilities (esp. **ggplot2** package)
- Flexible object-oriented programming
- Large (and growing!) package libraries
- Versatile user-interface (**RStudio**)

Download R

- *Step 1.* Download R: Go to https://www.r-project.org/. Feel free to use any suitable mirror. Here is the UMichigan mirror.
- *Step 2.* Download RStudio https://rstudio.com/products/rstudio/download/ and install RStudio Desktop.

Download R Studio



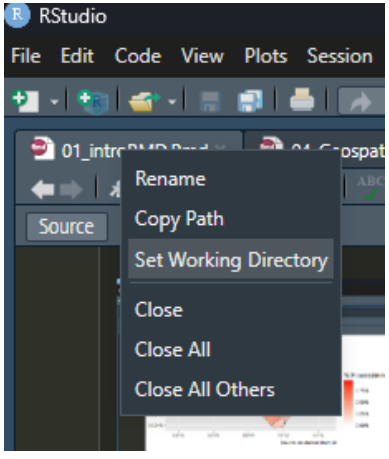
NW: write/run code, edit docs *SW*: console, enter commands *NE*: Find variables/command history *SE*: display plots, **help**...

Getting started with R: working directory

- Create a directory where you will save your R files, data, and figures. Set this to be your working directory by using

```
setwd("path/to/dir")
```

- Alternatively:



Getting Started with R: worksapce

- If you want to save everything in the current area as an R workspace, type

```
save.image("MyWorkspace.RData")
```

- To load a workspace, type

```
load("Myworkspace.RData")
```

- Note: Whenever you assign a variable name to an object, this object remains in your workspace until cleared or the variable name is reassigned (type `ls( )` or `objects` to see what you currently have defined in your workspace)

Installing and loading R packages

- Much of the power of R comes from packages (how code is shared in the R community)
- To install packages, use:

```
install.packages("dplyr", "ggplot2")
```

- To load packages, use

```
library("dplyr", "ggplot2")
```

- You can download packages straight for GitHub with the `devtool` package

```
devtools::install_github(
  "dpmcsuss/igraphmatch", ref = "dev"
)
```

- Sometimes you don't want to load the whole package just to use one function, so you use `::` like the example above:

```
PackageName::function(...)
```

Installing all the necessary packages

- We will often work with *tidy* data and in the *tidyverse* (more on this later)
- To install (most of) the needed packages, type the following into your RStudio console

```
tidyverse_packages <- c("tidyverse", "broom",
  "coefplot","cowplot","gapminder", "GGally",
  "ggrepel", "ggridges", "gridExtra","here",
  "interplot", "margins", "maps", "mapproj",
  "mapdata", "MASS", "quantreg", "rlang", "scales",
  "survey", "srvyr", "viridis", "viridisLite",
  "devtools")

install.packages(tidyverse_packages)
```

- The code above creates a vector/list of characters called `tidyverse_packages` using the most basic combine function `c()`

Saving and Loading Files

- To save an .RData file, use `save(filename, file = "directory")`

```
save(X, file = ".../dataviz/Lecture Notes/MyData.RData")
```

- We can load it with 'load()'

- To save/load a single R object, you can use `saveRDS()` and `readRDS()`
- To save a file as a .csv, use `write.table()` or `fwrite()` (more on loading .csv later)
- Many other formats. **GOOGLE** what you need

(Down)loading Data

If data is stored at a remote site, you can download it as follows:

```
pway <-  "https://cdn.rawgit.com/kjhealy/viz-organdata/mast
organs <- readr::read_csv(pway)
```

```
## Rows: 238 Columns: 21
## -- Column specification ------------------------------
## Delimiter: ","
## chr  (7): country, world, opt, consent.law, consent.prac
## dbl (14): year, donors, pop, pop.dens, gdp, gdp.lag, hea
##
## i Use `spec()` to retrieve the full column specificatio
## i Specify the column types or set `show_col_types = FALS
```

- Web scrapes: read up on the `rvest` and `httr` packages and **GET** for more details on how to add options for website scrapes

Loading data locally

- Let's save `organs` as a .csv file locally.

Loading data locally

- Let's save `organs` as a .csv file locally.

```
write.csv(organs, file = "data/organdonations.csv")
```

What I did here is that I saved the `organs` object as a .csv file called "organdonations.csv" in the *data* folder. You can also use the `write.csv()` function. Type `?write.csv()` for more.

- To load the local csv file, we can do this:

```
organs2 <- read.csv("data/organdonations.csv")
```

Loading data

When you download a new dataset, it pays to take a look inside it using `str()`

```
str(organs)
```

```
## spc_tbl_ [238 x 21] (S3: spec_tbl_df/tbl_df/tbl/data.fra
##  $ country          : chr [1:238] "Australia" "Australia"
##  $ year             : num [1:238] NA 1991 1992 1993 1994
##  $ donors           : num [1:238] NA 12.1 12.3 12.5 10.2
##  $ pop              : num [1:238] 17065 17284 17495 17667
##  $ pop.dens         : num [1:238] 0.22 0.223 0.226 0.228
##  $ gdp              : num [1:238] 16774 17171 17914 18883
##  $ gdp.lag          : num [1:238] 16591 16774 17171 17914
##  $ health           : num [1:238] 1300 1379 1455 1540 162
##  $ health.lag       : num [1:238] 1224 1300 1379 1455 154
##  $ pubhealth        : num [1:238] 4.8 5.4 5.4 5.4 5.4 5.5
##  $ roads            : num [1:238] 137 122 113 111 108 ...
##  $ cerebvas         : num [1:238] 682 647 630 611 631 592
##  $ assault          : num [1:238] 21 19 17 18 17 16 17 17
```

Loading data

When you download a new dataset, it pays to take a look inside it using `str()`

```
str(organs2)
```

```
## 'data.frame':    238 obs. of  22 variables:
##  $ X                : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ country          : chr  "Australia" "Australia" "Austr
##  $ year             : int  NA 1991 1992 1993 1994 1995 19
##  $ donors           : num  NA 12.1 12.3 12.5 10.2 ...
##  $ pop              : int  17065 17284 17495 17667 17855
##  $ pop.dens         : num  0.22 0.223 0.226 0.228 0.231
##  $ gdp              : int  16774 17171 17914 18883 19849
##  $ gdp.lag          : int  16591 16774 17171 17914 18883
##  $ health           : num  1300 1379 1455 1540 1626 ...
##  $ health.lag       : num  1224 1300 1379 1455 1540 ...
##  $ pubhealth        : num  4.8 5.4 5.4 5.4 5.4 5.5 5.6 5.
##  $ roads            : num  137 122 113 111 108 ...
##  $ cerebvas         : int  682 647 630 611 631 592 576 52
```

Digression: `organs` vs `organs2`

- Note that the two objects are different! `organs` is a `spec_tbl_df`(**Tibble**), while `organs2` is a `data.frame`. What caused this difference?

Digression: `organs` vs `organs2`

- Note that the two objects are different! `organs` is a `spec_tbl_df`(**Tibble**), while `organs2` is a `data.frame`. What caused this difference? `read.csv()` vs `read_csv`.
- **data.frame**: base R structure for tabular data, widely used and versatile. No meta data (such as column specifications). good for general-purpose R programming and legacy codebases.
- **Tibble**: A subclass of `data.frame` introduced by the `tibble` package, designed for better usability and modern workflows. Often includes additional metadata (good for data import/export). Ideal for tidyverse workflows (strict rules and predictable behaviors).

R Basics: names

- Everything in R has a name
  - variables (think `x` or `y`)
  - data you have loaded (`organs` and `organs 2`)
  - functions (`c()`, `install.packages()`)
- Avoid common function names when naming your variables

`t()`, `q()`, `mean()`, `median()`, `sd()`, `rnorm()`, `c()`, `...`

- Names in R are case sensitive. **Convention**: convention: lowercase names, separate words with underscore `visualize_data()`

R basics: expressions and assignments

Elementary commands consist of either **expressions** or **assignments**

- **Expressions:** executable commands; evaluated, printed then discarded

```
sqrt(765)/5^4
```

```
## [1] 0.04425381
```

- **Assignments:** Evaluate an expression and pass the value to a variable; result not usually printed. Use <-, = (or -> but don't use this)

```
number <- sqrt(765)/5^4
```

R basics: Guess the output

-

```
x <- 3+4
x + 5
```

R basics: Guess the output

-

```
x <- 3+4
x + 5
```

- `y <- 1:10`

R basics: Guess the output

-

```
x <- 3+4
x + 5
```

- `y <- 1:10`
- `x + y`

R basics: Guess the output

-

```
x <- 3+4
x + 5
```

- `y <- 1:10`
- `x + y`

```
x <- 3+4
y <- 1:10
x + y
```

```
##  [1]  8  9 10 11 12 13 14 15 16 17
```

- The last phenomenon is usually referred to as *broadcasting*: the process of aligning the dimensions of two objects (like vectors, matrices, or arrays) so that operations can be performed element-wise, even when their dimensions differ (**not always possible**)

Objects and Classes in R

Everything in R is an object. Every object in R has a class.

- R operates on objects:
  - vectors or lists
  - functions (more on this later)
- **Lists**: Composed of data-objects organized as strings or vectors; elements can be **numerical** (or complex)

```
x <- c(1,2,3, 8:10)
str(x)
```

```
##  num [1:6] 1 2 3 8 9 10
```

```
class(x)
```

```
## [1] "numeric"
```

Boolean

- Boolean numbers: **T/F** $\leftrightarrow$ **1/0**

```
x <- 2:7
y <- x > 3
str(y)
```

```
##  logi [1:6] FALSE FALSE TRUE TRUE TRUE TRUE
```

Boolean

- Boolean numbers: **T/F** $\leftrightarrow$ **1/0**

```
x <- 2:7
y <- x > 3
str(y)
```

```
##  logi [1:6] FALSE FALSE TRUE TRUE TRUE TRUE
```

```
z <- y + 5
z
```

```
## [1] 5 5 6 6 6 6
```

```
class(z)
```

```
## [1] "numeric"
```

Boolean

- Boolean numbers: **T/F** $\leftrightarrow$ **1/0**

```
x <- 2:7
y <- x > 3
str(y)
```

```
##  logi [1:6] FALSE FALSE TRUE TRUE TRUE TRUE
```

```
z <- y + 5
z
```

```
## [1] 5 5 6 6 6 6
```

```
class(z)
```

```
## [1] "numeric"
```

- R automatically converts `logi` (which is in principle T/F) to the more general `num`
- We can define Boolean variables with >, <, ==, >=, <=, !=

Lists of Characters

- The following operations will produce an error.

```
y <- c("Hello", "Data350")
z <- x + y
```

- Any reasonable way to go though with the operation?

Lists of Characters

- The following operations will produce an error.

```
y <- c("Hello", "Data350")
z <- x + y
```

- Any reasonable way to go though with the operation?

```
y <- c("Hello", "Data350")
z <- c(x,y)
str(z)
```

```
##  chr [1:8] "2" "3" "4" "5" "6" "7" "Hello" "Data350"
```

```
class(z)
```

```
## [1] "character"
```

- R automatically converts `num` to the more general `chr`

Data frames

- **Data frames** (and **tibbles**): rectangular lists in R used to efficiently store/analyze/visualize complex data sets. Think **spreadsheets**
- Columns: variables/covariates/features
- Rows: Observations
- Access entries of the dataframe `X` using `X[rows_needed, col_needed]`

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.4
```

```
data("midwest")
midwest[c(1:4), c(2,3,5,18)]
```

```
## # A tibble: 4 x 4
##   county    state poptotal perchsd
##   <chr>     <chr>    <int>   <dbl>
```

Do yourself a favor!

- Dataframes (and tibbles) are a key data structure for data visualization in R
- It is important to keep your data frames (tibbles) **tidy**
- Each variable must have its own column
- Each observation must have its own row
- Each value must have its own cell
- Can be some work up front, but pays dividends down the line!

Inspecting a dataframe

- We have used `str()`. Some other common options include

`head()`, `tail()`, `slice_head()`, `slice_sample()`

- For instance, the following selects 10 random observations from `midwest`.

```
dplyr::slice_sample(midwest, n = 10)
```

```
## # A tibble: 10 x 28
##      PID county  state  area poptotal popdensity popwhit
##    <int> <chr>   <chr> <dbl>    <int>       <dbl>    <int
##  1  2990 CLARK   WI    0.072    31647        440.     3143
##  2  3004 GREEN ~ WI    0.022    18651        848.     1838
##  3  3032 RACINE  WI    0.02    175034       8752.    15209
##  4   687 FULTON  IN    0.021    18840        897.     1855
##  5  2053 LICKING OH    0.04    128300       3208.    12518
##  6   662 WOODFO~ IL    0.032    32653       1020.     3238
##  7  1214 CLARE   MI    0.034    24952        734.     2466
##  8   690 GREENE  IN    0.033    30410        922.     3024
```

Inspecting a dataframe: column selection

- We have used `X[rows,columns]`, but it's not easy to count the column numbers
- Instead, we use `X$col_name` or `pull()`

```
pd <- midwest$popdensity
str(pd)
```

```
##  num [1:437] 1271 759 681 1812 324 ...
```

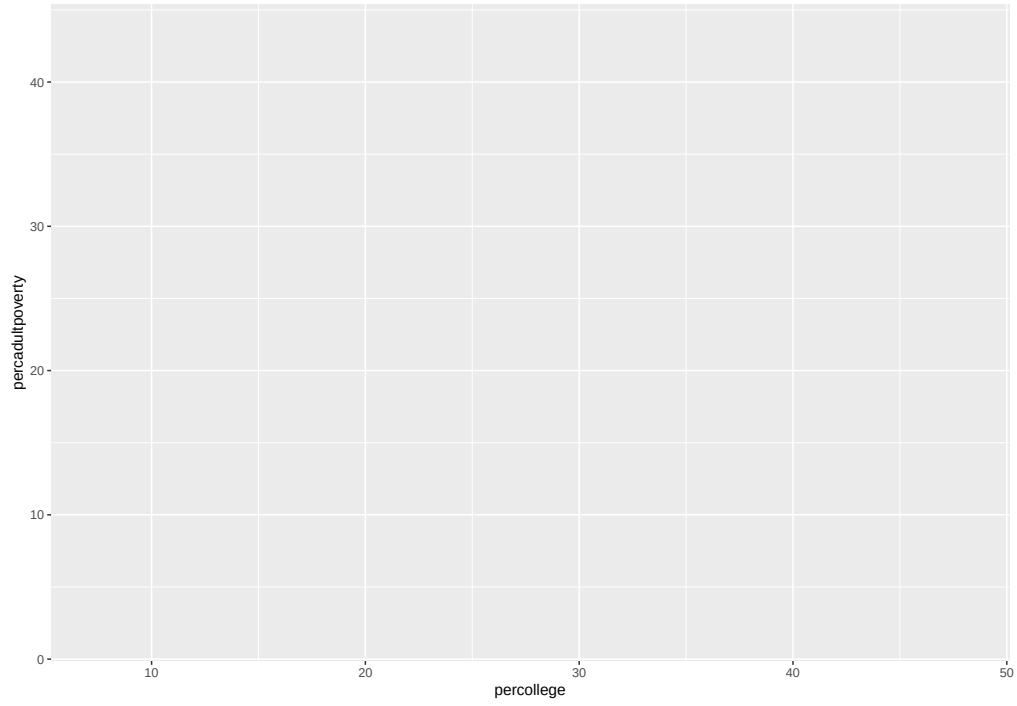
```
pd2 <- dplyr::pull(midwest, var = "popdensity")
str(pd2)
```

```
##  num [1:437] 1271 759 681 1812 324 ...
```

- At the moment, we only select one column at a time. More advanced functions later.

Plotting

```
library(ggplot2)
p <- ggplot(midwest,aes(
                x = percollege, y = percadultpoverty))
p
```

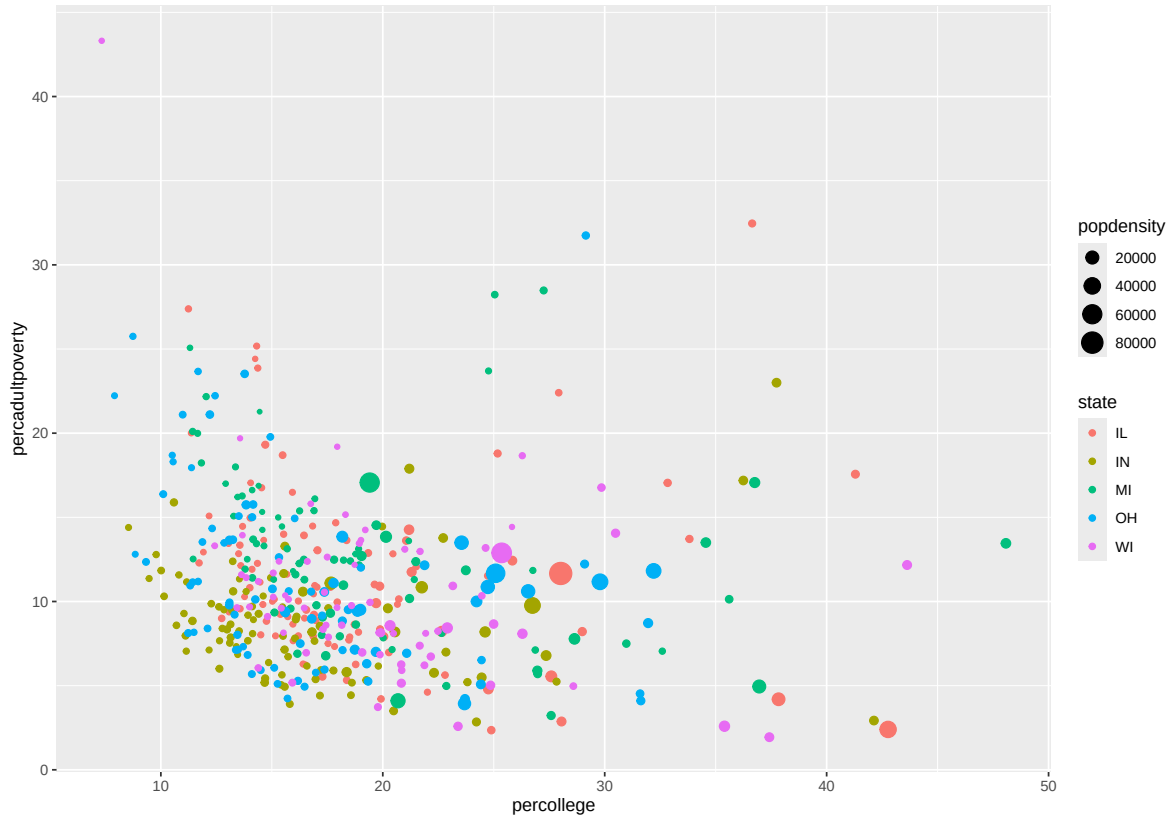


In the previous slide, p is a graph object and follows the ggplot2 syntax. We can build our graph systematically.

Plotting

At the moment, `p` is a blank canvas, but we can add a **layer** of scattered plot using `geom_point()`

```
p + geom_point(aes(col = state, size = popdensity))
```



- In the `aes` selection, we use colors to different states, and dot size to reflect population density.

Plotting a dataframe

Add **legends** and **labels** (I will be **VERY PICK** about this in your grading).