STA 235H - Bootcamp

Fall 2023

McCombs School of Business, UT Austin

Bootcamp Agenda

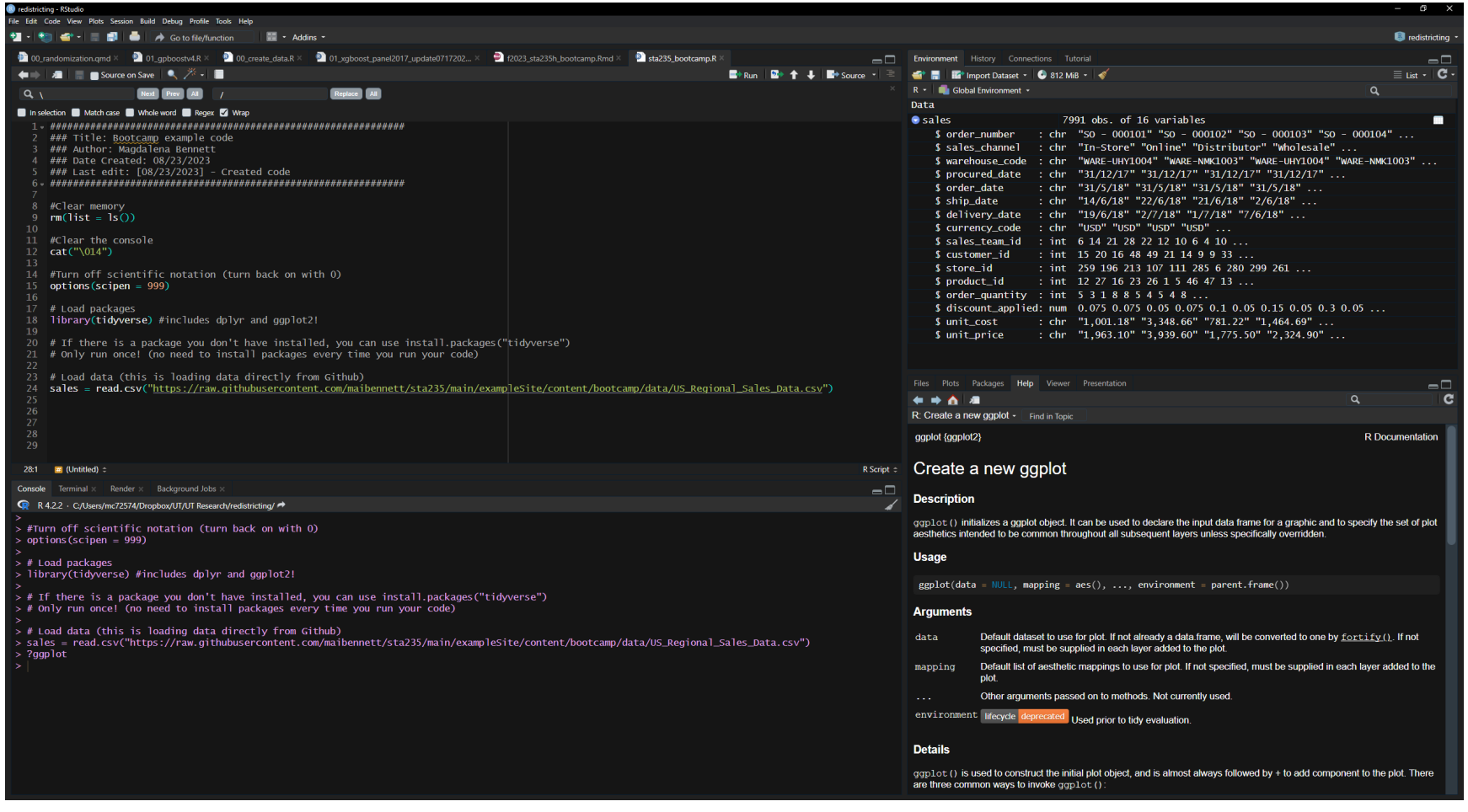
- What do we need?
  - Quick look into R and RStudio
  - RScript format
- Refresher from the tidyverse:
  - Data wrangling
  - Plots and figures
  - Regressions

R for coding

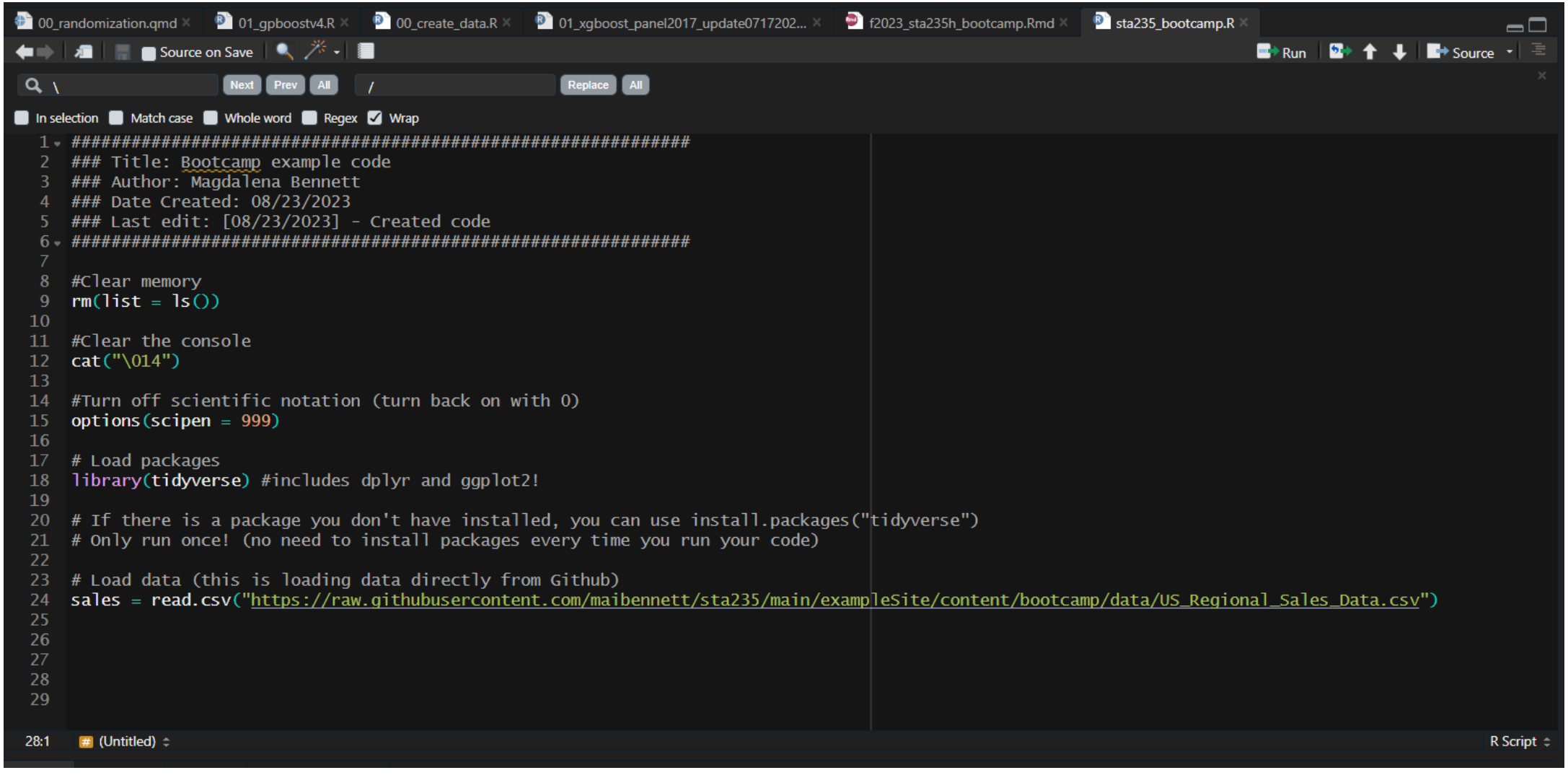
R is the programming language we will use for **statistical analysis**

RStudio is the IDE (Integrated Development Environment) we will use **to run R on our computers**.

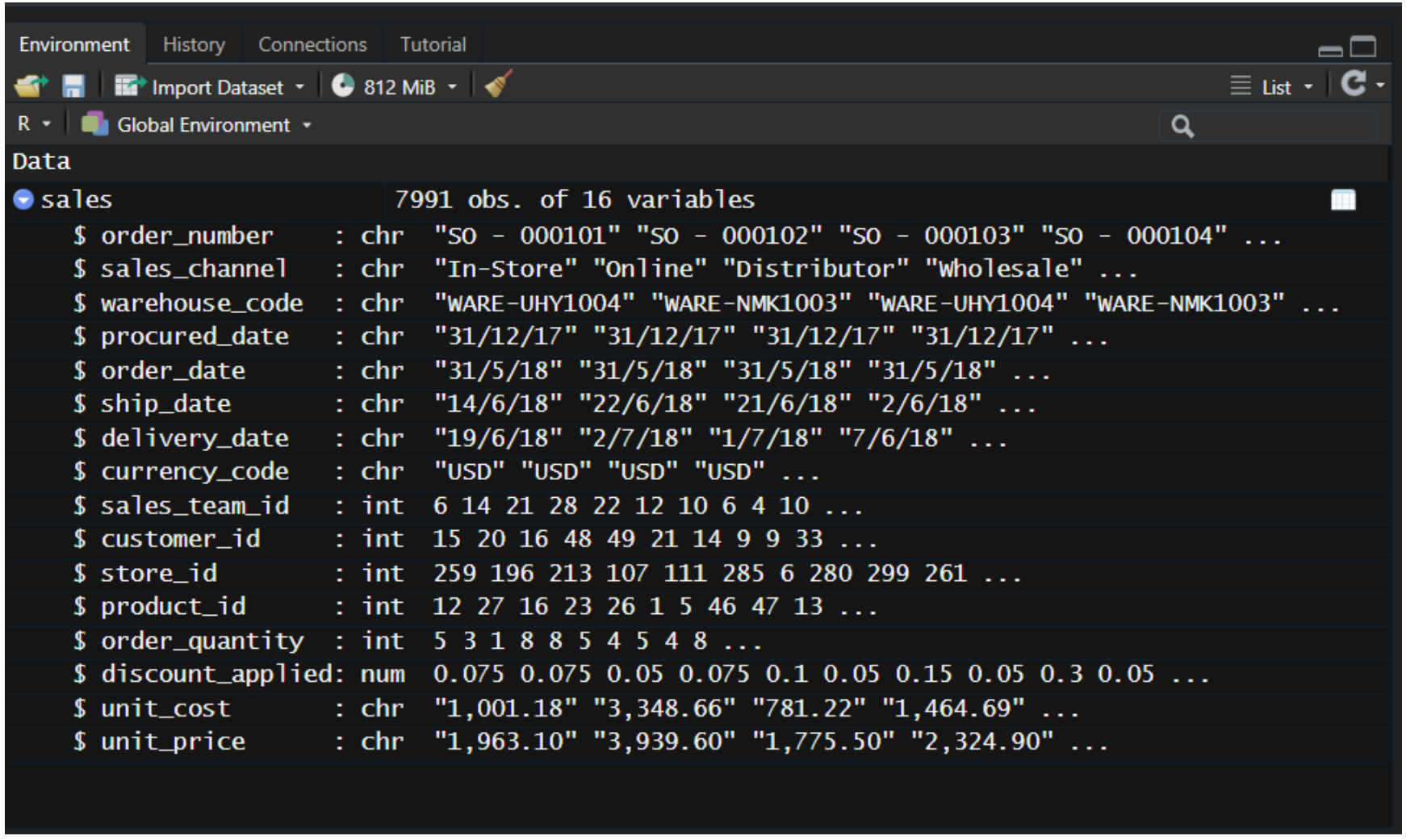
Let's look at RStudio



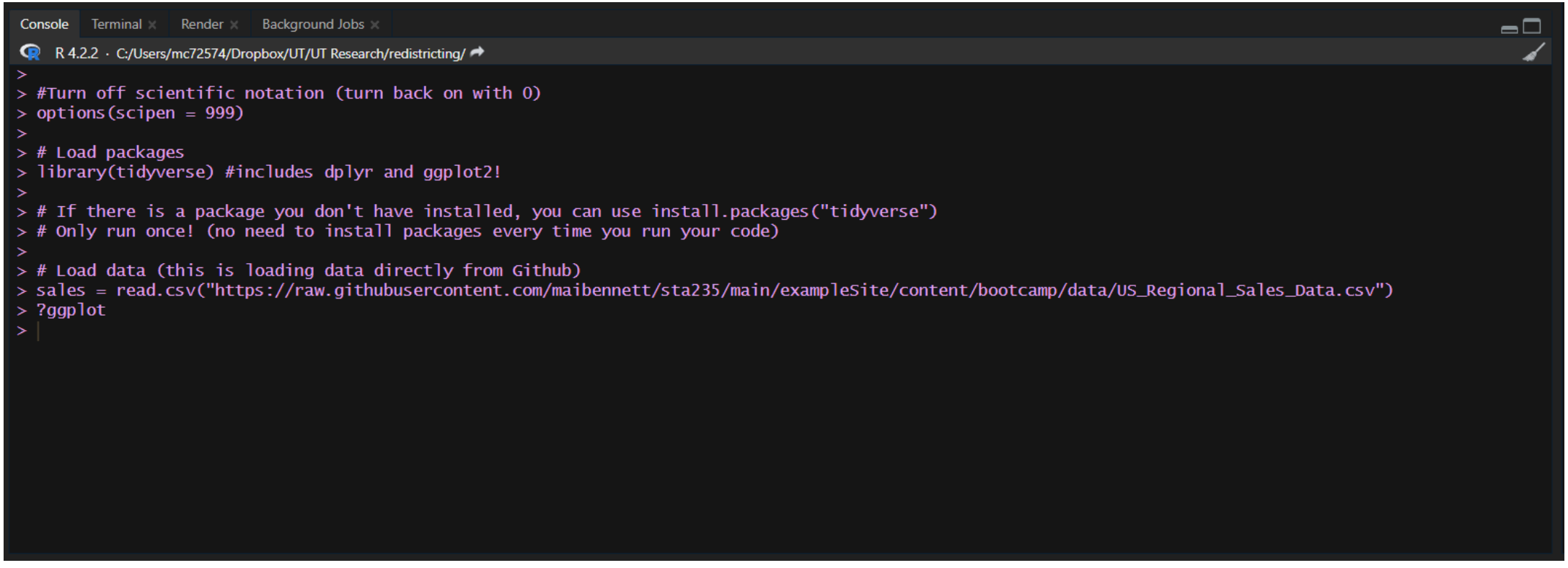
Let's look at RStudio - Script



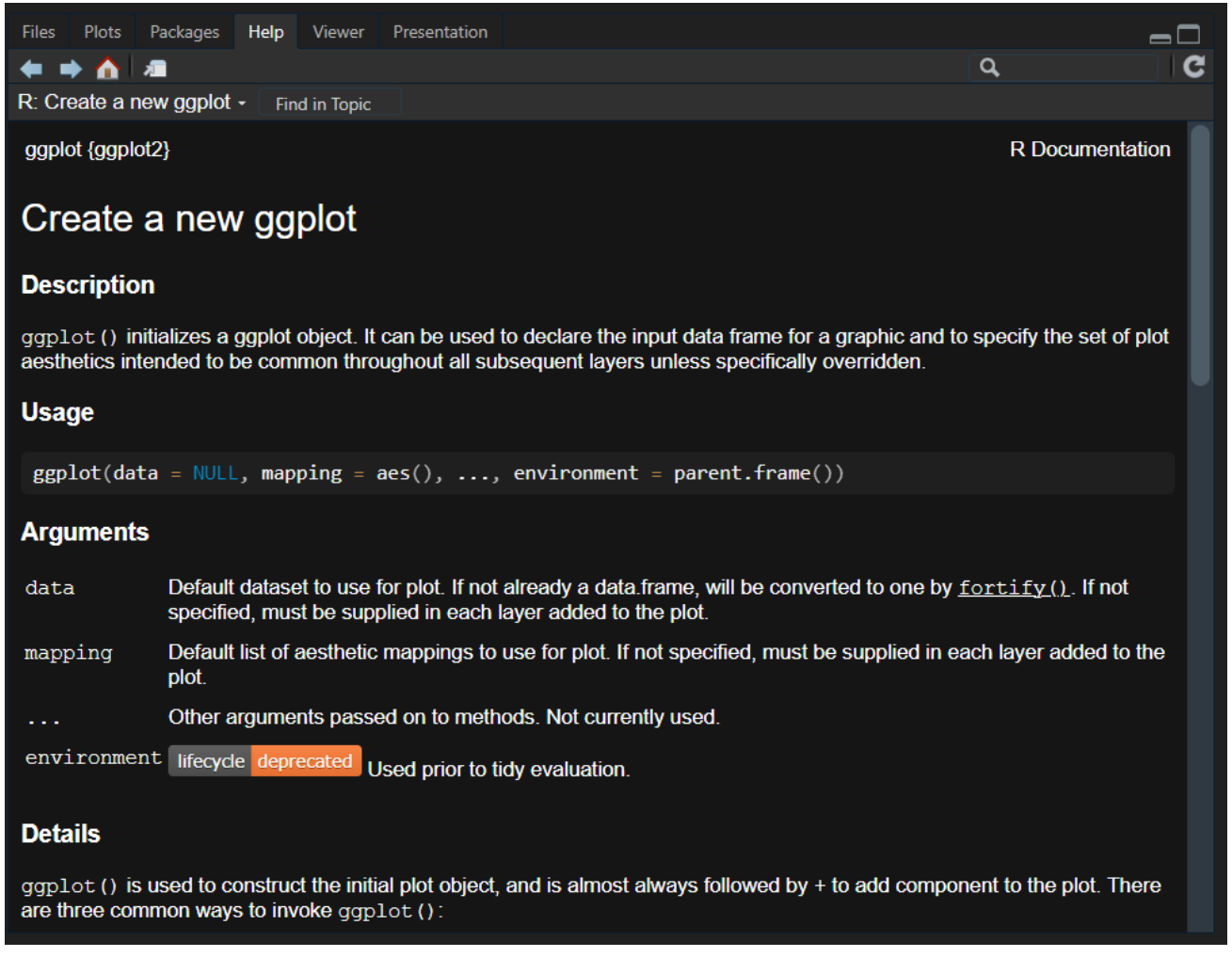
Let's look at RStudio - Environment



Let's look at RStudio - Console



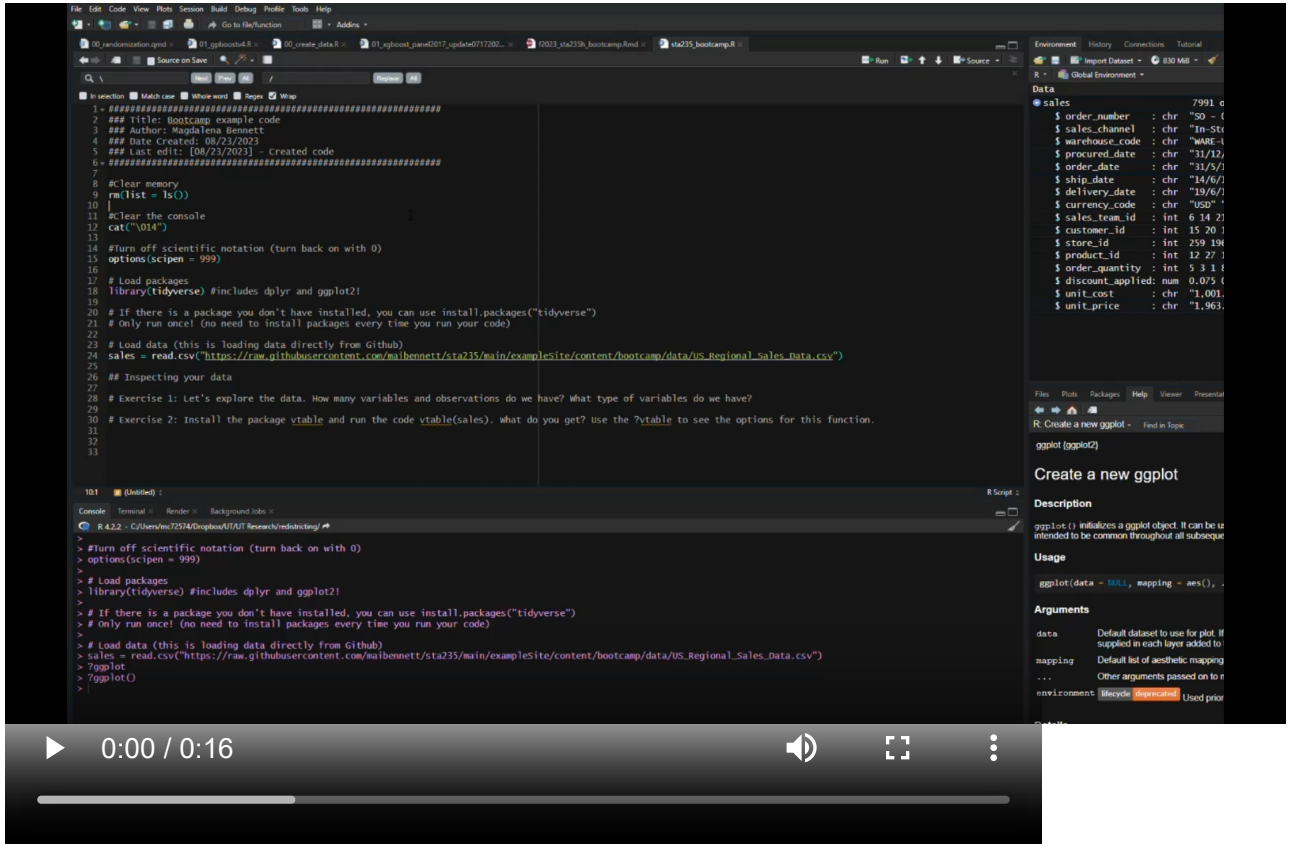
Let's look at RStudio - Help and others



Useful basic commands

- `install.packages("name")`: Installs the package "name" on your computer. You only need to run this once!
- `library(name)`: Loads the package "name" on your current session. You should do this at the top of every script and only include packages you will use (to avoid confusion)
- `?function`: Opens the help file for `function` (if there is more than one `function` – e.g. different libraries – you can choose which one you open).

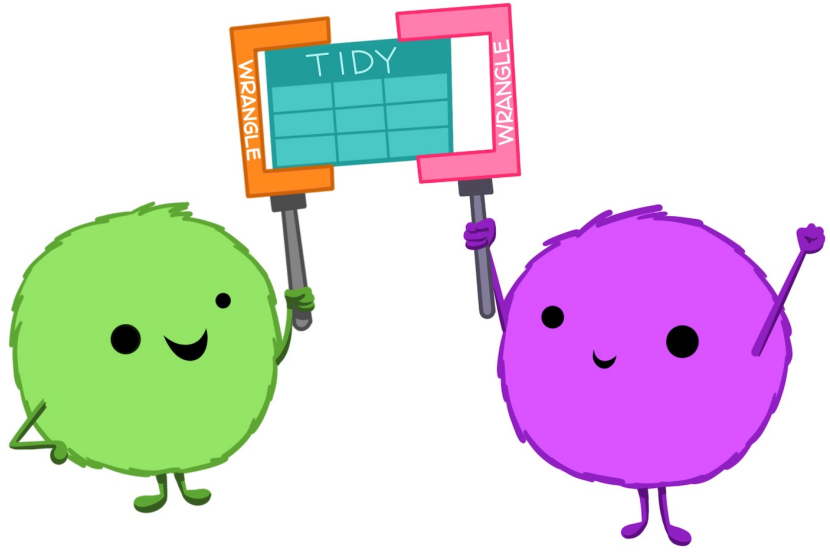
Also... don't restore RData into a new session!



Let's go to R

Data Wrangling

- Most times we need to **transform**, **clean**, and **structure** data for analysis.
- Examples of data wrangling would be dropping missing observations, merging different datasets, identifying outliers, etc.
- **R can help us do that!



Into the tidyverse

- For data wrangling, we will use the tidyverse: Collection of packages that follow a similar design structure (e.g. dplyr, ggplot2)
- It works through pipes: %>%
    - Concatenates functions!

Useful functions for wrangling

- `mutate(var = var1 + var2)`: Creates a new variable or replaces an existing one. It takes as an argument the name of the variable and what you want that variable to be.
- `filter(var == 1)`: Subsets your data according to a logic statement. Remember that logic statements use "==" instead of "="!
- `group_by(var1, var2)`: Used to group observations by values of different variables. You can use it either to create a variable with values at the group level, or to summarize your dataset by group.
- `select(var1, var2)`: Select specific variables from the dataset (drop the others). In case you want to drop instead of keeping variables, you can use `select(-var1, -var2)`
- `rename(var_new = var_old)`: The name says it all. Used to rename variables.

Other useful functions

- `is.na(var)`: logic function that returns TRUE if the observation is a missing value (NA) or FALSE in another case.
- `ifelse(logic_statement, val1, val2)`: Very useful function to create conditional values.
- `!(logic_statement)`: The exclamation point acts as a negation. If you want to invert a logic statement, use this (e.g. `!is.na(var)` will return TRUE if the obs of `var` is NOT missing and FALSE if it's missing).
- `table(var)`: Tabulates the different values of a variable

Let's go to R

Plotting in R

- Plotting your data is a **very intuitive way** to see what's going on.
- It's also useful to convey **complex analysis**!
- Make sure your plots are always **informative** and they **tell the story** you want to highlight.

General structure of ggplot

- `ggplot()` works in "layers":
  - You can provide different geometries and "add" them to your plot (same with themes!)
- You always start with `ggplot(data = d, aes(x = var1, y = var2, color = var3))`, depending on what you want to do:
  - `aes()` stands for aesthetics, and it tells which variables you want to use and how. Sometimes you need one variable (e.g. histogram), sometimes you need two (e.g. scatter plot), or even three or more! (e.g. scatter plot for different groups)
- You can provide `aes()` in the `ggplot()` function (as seen above), or also in each geometric layer:

  e.g. `ggplot(data = d) + geom_point(aes(x = var1, y = var2))`

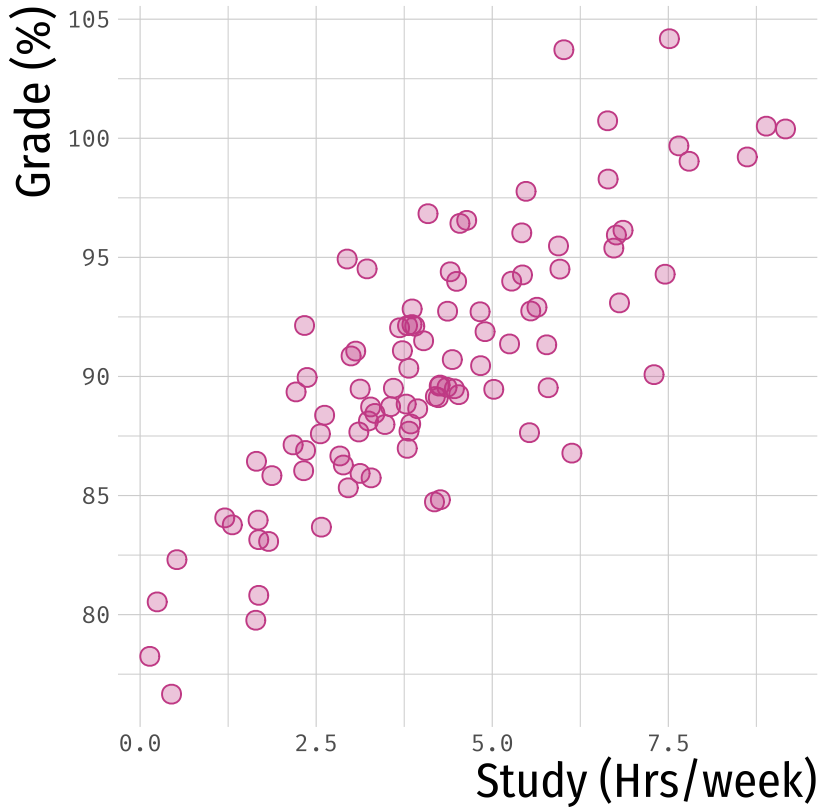
General structure of ggplot

- Some common geometries that are useful:
    - `geom_point()`: Creates a scatter plot
    - `geom_line()`: Creates a line plot
    - `geom_histogram()` or `geom_density()`: Creates a histogram or a density plot for your data!
    - `geom_smooth()`: Creates a smooth function that goes through your data. By default, it uses a loess or gam function, depending on the size of the data. Use `method = "lm"`as an argument if you want to fit a regression line!
- Finally, looks are also important!
    - `theme()` allows you to play around with every aspect of your plot (e.g. font size, grid lines, etc.)
    - Using a pre-packages theme can be useful, too. I personally like `theme_minimal()` or the `theme_ipsum_rc()` from the `hrbrthemes` package.

Let's go to R

Regression Analysis

- Regressions help us **quantify the relationship** between different variables.
- In R, we can get **many important insights** from regression analysis!



Regressions in R

- The main command to do regressions is `lm(y ~ x1 + x2, data = d)`, where `y` is our outcome of interest and `x1` and `x2` are regressors.
- For convenience, we can store the regression in a separate object (e.g. `lm1 = lm(y ~ x1 + x2, data = d)`), so we can later manipulate it:
    - `summary(lm1)`: Provides a summary table of the results (including estimates, standard errors, and p-values).
    - `lm1$coefficients`: Recovers the exact estimated coefficients (e.g. useful if you want to use them later).
    - `summary(lm1)$coefficients`: Matrix of results. Includes columns for the estimates betas, standard errors, t-stats, and p-values.

Let's go to R

R is useful and fun!