Working with Data

Ray Caraher

2023-10-16

Section 1

R Scripts

Discussion Question

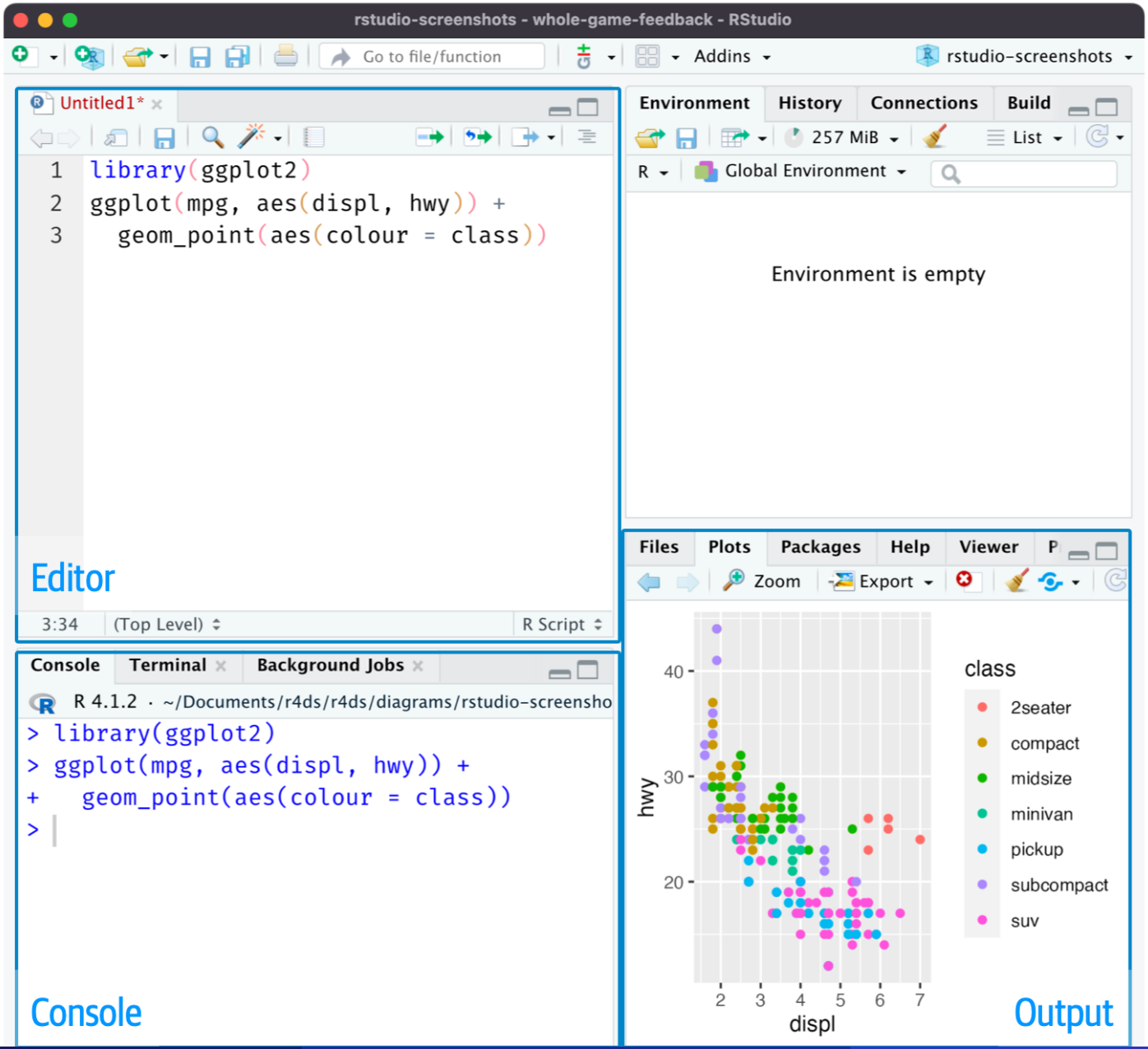
What is one thing about using R that you are most confused about?

The R Console

R Studio has two main ways to run code:

- The console
- An R script file

The Console and script in R Studio



The Console

The console is where you directly enter code line-by-line.

- You can only run 1 line at a time
- To run the code, you type it in the console the press “Enter”
- Easy to use
- But can’t save your work and is bad for reproducability

R Scripts

Let's you write longer code which can reproduce the analysis you have already done.

- Open it up by clicking the File menu, selecting New File, then R script, or using the keyboard shortcut Cmd/Ctrl + Shift + N.
- The script editor is a great place to experiment with your code.
- When you want to change something, you don't have to re-type the whole thing, you can just edit the script and re-run it.
- Once you have written code that works and does what you want, you can save it as a script file to easily return to later.
- Note: Macs use Cmd, Windows/Linux use Ctrl

Using the R script

The key to using the script editor effectively is to memorize one of the most important keyboard shortcuts: Cmd/Ctrl + Enter.

- This executes the current R expression where your cursor is in the console.
- Then it automatically puts your cursor onto the next line.
- Can also run multiple lines by highlighting them, then using Cmd/Ctrl + Enter

Cmd/Ctrl + Shift + S

- Runs the entire script from top-to-bottom.
- Good for making sure your whole script works and you are effectively able to reproduce your results.

R script example

```
library(dplyr)
library(nycflights13)

not_cancelled <- flights |>
  filter(!is.na(dep_delay), !is.na(arr_delay))

not_cancelled |>
  group_by(year, month, day) |>
  summarize(mean = mean(dep_delay))
```

Figure 2: Script cursor example

If your cursor is the black bar, pressing Cmd/Ctrl + Enter will run the complete command that generates not_cancelled. It will also move the cursor to the following statement (beginning with not_cancelled |>). That makes it easy to step through your complete script by repeatedly pressing Cmd/Ctrl + Enter.

R script best-practices

- Always start your script with your name and the purpose of script in comments.
- Load your packages at the start of the script [using library()]
- Don't put install.packages() in your script (you never want to install things on others computers!)
- Be sure to save your script using a readable but short and descriptive name with no space (e.g., assignment_3.R, econ_337_data_plotting.R, and not Script File 1.R)

Working Directories

R has a powerful notion called a "Working Directory." It is where the analysis "lives" on your computer.

This is where R looks for files that you ask it to load, and where it will put any files that you ask it to save. If you want to read in data or any other file, R will look first at the Working Directory unless you give it a filepath.

RStudio shows your current working directory at the top of the console.

You can also get it by using the following code:

```
getwd()
```

```
## [1] "/Users/rcaraher/Library/CloudStorage/OneDrive-Universi
```

Working Directories

As a beginning R user, a good place to set your Working Directory is your documents folder, or even better your folder for Econ 337 (if you have one).

In RStudio, I would recommend the following:

- Save your R script to your Econ 337 folder (or a sub-folder for this week) on your computer (somewhere else is fine too)
- Put all your data and other files in this folder as well
- In RStudio, go to Session -> Set Working Directory -> To Source File Location

This will set the working directory to the folder where your script is saved.

Can also tell R where to set the Working Directory using "setwd" function

Section 2

Importing Data

Data types

Most data you will encounter in statistical programming will be in the .csv format, rather excel

- Stands for Comma Separated Values file
- Each value is separated by a comma, which corresponds to a different variable
- Basically a very bare-bones spreadsheet

Also can read in excel/google sheets/other types of data into R, and the process is broadly the same.

Example of .csv

This is what a .csv file looks like "raw" with some data on student lunches.

```
Student ID,Full Name,favourite.food,mealPlan,AGE
1,Sunil Huffmann,Strawberry yoghurt,Lunch only,4
2,Barclay Lynn,French fries,Lunch only,5
3,Jayendra Lyne,N/A,Breakfast and lunch,7
4,Leon Rossini,Anchovies,Lunch only,
5,Chidiegwu Dunkel,Pizza,Breakfast and lunch,five
6,Güvenç Attila,Ice cream,Lunch only,6
```

Figure 3: Raw .csv example

The first row has the column/variable names, and each other rows has the values for each student.

Example of .csv

These tables can also be imported into excel and automatically made into a spreadsheet. As a table/spreadsheet it would look like this:

Student ID	Full Name	favourite.food	mealPlan	AGE
1	Sunil Huffmann	Strawberry yoghurt	Lunch only	4
2	Barclay Lynn	French fries	Lunch only	5
3	Jayendra Lyne	N/A	Breakfast and lunch	7
4	Leon Rossini	Anchovies	Lunch only	NA
5	Chidiegwu Dunkel	Pizza	Breakfast and lunch	five
6	Güvenç Attila	Ice cream	Lunch only	6

Figure 4: Raw .csv example

Reading .csv files into R

We will use functions from the readr, which will load with the tidyverse.

```
library(tidyverse)
```

We can then read the .csv file into r with the read_csv()

```
students <- read_csv("students.csv")
```

```
## Rows: 6 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Full Name, favourite.food, mealPlan, AGE
## dbl (1): Student ID
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

When you run read_csv(), it prints out a message telling you the number of rows and columns of data, the delimiter that was used, and the column specifications (names of columns organized by the type of data the column contains).

Cleaning data

You will usually need to "clean" data after you read it to make it usable. Here, we will clean missing values and variable types.

NA values

R has smart ways of dealing with missing values (NAs). But R first has to be told what an NA value looks like in the raw data. R is usually pretty good at figuring out what NA values are supposed to be, but whoever made this spreadsheet did it inconsistently:

- In the favorite.food column, it is recorded as "N/A".
- in the AGE column, the cell is left empty

R correctly read the NA value in AGE, but not in favorite.food.

Fixing NA values

Luckily, the read_csv() function allows us to tell R what should count as NA. The "na" arguement of read_csv() allows us to give a vector which lists all the possible NA indicators used in the data. The empty string (" ") is the default, and tells R that an empty cell should be treated as NA.

```
students <- read_csv("students.csv", na = c("N/A", ""))
```

```
## Rows: 6 Columns: 5
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (4): Full Name, favourite.food, mealPlan, AGE
## dbl (1): Student ID
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
students
```

```
## # A tibble: 6 x 5
##   `Student ID` `Full Name`      favourite.food     mealPlan            AGE
##          <dbl> <chr>            <chr>              <chr>               <chr>
## 1            1 Sunil Huffmann   Strawberry yoghurt Lunch only          4
## 2            2 Barclay Lynn     French fries       Lunch only          5
## 3            3 Jayendra Lyne    <NA>               Breakfast and lunch 7
## 4            4 Leon Rossini     Anchovies          Lunch only          <NA>
## 5            5 Chidiegwu Dunkel Pizza              Breakfast and lunch five
## 6            6 Güvenç Attila    Ice cream          Lunch only          6
```

Fixing column names

Also, notice that the column names are inconsistent too. Some have spaces, some are capitalized, etc.

```
colnames(students)
```

```
## [1] "Student ID"     "Full Name"      "favourite.food" "mealPlan"
## [5] "AGE"
```

While we wouldn't have to fix this, it would make it easier to remember the variable names in the long run. The function clean_names() from the janitor package can do this automatically for us.

```
students <- students |> janitor::clean_names()
colnames(students)
```

```
## [1] "student_id"     "full_name"      "favourite_food" "meal_plan"
## [5] "age"
```

Now they are all "snake_case" which will help us remember.

Side note: Using one function

Note how we proceeded the clean_names() function with "janitor::".

Rather than using library() to load all the functions from a package, if we only want to use one function from a package without loading them all we can do "package_name::function_name()".

This is helpful if there is only one function we want to use from a package (like the clean_names() function from janitor package).

Variable types

It also helps us if R knows what class of object each column/variable is (recall character, numeric, logical, etc.). R is pretty good at guessing the column types, so lets see how it did.

```
glimpse(students)
```

```
## Rows: 6
## Columns: 5
## $ student_id     <dbl> 1, 2, 3, 4, 5, 6
## $ full_name      <chr> "Sunil Huffmann", "Barclay Lynn", "Jayendra Lyne", "Leo~
## $ favourite_food <chr> "Strawberry yoghurt", "French fries", NA, "Anchovies", ~
## $ meal_plan      <chr> "Lunch only", "Lunch only", "Breakfast and lunch", "Lun~
## $ age            <chr> "4", "5", "7", NA, "five", "6"
```

- The class is short for double, and is a type of numeric vector so that works for student_id.
- The class is short for character, so that works for the full_name, favorite_food, and meal_plan variables since those are words.
- But it doesn't really work for age - we want that to be rather than a character.

Recoding variables

It looks like whoever entered the data put “five” instead of 5 for one of the rows. This confused R. So we will instead have to fix it ourselves. We can do this by re-coding the data using the if_else function.

Recoding using “if_else()”

For if_else(), the first argument test should be a logical vector. The result will contain the value of the second argument, yes, when test is TRUE, and the value of the third argument, no, when it is FALSE. Here we’re saying if age is the character string “five”, make it “5”, and if not leave it as age. We are using it with mutate() which we already know and love. We then convert “age” to a numeric vector using as.numeric().

```
students <- students %>%
  mutate(age = ifelse(age == "five", "5", age),
         age = as.numeric(age))
```

Re-shaping data

Data can come in multiple forms - Long: Many variables listed in a single column - Wide: Many variables listed as several columns

Often, data will be "too wide", meaning that there will be too many columns relative to the actual number of variables.

For example, the billboard data

```
billboard
```

```
## # A tibble: 317 x 79
##    artist track date.ent~1   wk1   wk2   wk3   wk4   wk5   wk6   wk7   wk8   wk9
##    <chr>  <chr> <date>     <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 2 Pac  Baby~ 2000-02-26    87    82    72    77    87    94    99    NA    NA
##  2 2Ge+h~ The ~ 2000-09-02    91    87    92    NA    NA    NA    NA    NA    NA
##  3 3 Doo~ Kryp~ 2000-04-08    81    70    68    67    66    57    54    53    51
##  4 3 Doo~ Loser 2000-10-21    76    76    72    69    67    65    55    59    62
##  5 504 B~ Wobb~ 2000-04-15    57    34    25    17    17    31    36    49    53
##  6 98^0   Give~ 2000-08-19    51    39    34    26    26    19     2     2     3
##  7 A*Tee~ Danc~ 2000-07-08    97    97    96    95   100    NA    NA    NA    NA
##  8 Aaliy~ I Do~ 2000-01-29    84    62    51    41    38    35    35    38    38
##  9 Aaliy~ Try ~ 2000-03-18    59    53    38    28    21    18    16    14    12
## 10 Adams~ Open~ 2000-08-26    76    76    74    69    68    67    61    58    57
## # ... with 307 more rows, 67 more variables: wk10 <dbl>, wk11 <dbl>,
## #   wk12 <dbl>, wk13 <dbl>, wk14 <dbl>, wk15 <dbl>, wk16 <dbl>, wk17 <dbl>,
## #   wk18 <dbl>, wk19 <dbl>, wk20 <dbl>, wk21 <dbl>, wk22 <dbl>, wk23 <dbl>,
## #   wk24 <dbl>, wk25 <dbl>, wk26 <dbl>, wk27 <dbl>, wk28 <dbl>, wk29 <dbl>,
## #   wk30 <dbl>, wk31 <dbl>, wk32 <dbl>, wk33 <dbl>, wk34 <dbl>, wk35 <dbl>,
## #   wk36 <dbl>, wk37 <dbl>, wk38 <dbl>, wk39 <dbl>, wk40 <dbl>, wk41 <dbl>,
```

Reshaping data

In this dataset, each observation is a song. The first three columns (artist, track and date.entered) are variables that describe the song. Then we have 76 columns (wk1-wk76) that describe the rank of the song in each week. Here, the column names are one variable (the week) and the cell values are another (the rank). We want the the week number to appear as one variable, rather than 76.

pivot_longer()

To do this, we can use the pivot_longer() function

```
billboard_l <- billboard |>
    pivot_longer(
    cols = starts_with("wk"),
    names_to = "week",
    values_to = "rank"
  )
```

pivot_longer()

We know see that the week number is its own column, and rank is another column. This data will be easier to use (it is "tidy").

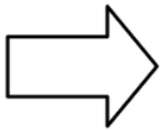
```
billboard_l
```

```
## # A tibble: 24,092 x 5
##    artist track                   date.entered week   rank
##    <chr>  <chr>                   <date>       <chr> <dbl>
##  1 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk1      87
##  2 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk2      82
##  3 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk3      72
##  4 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk4      77
##  5 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk5      87
##  6 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk6      94
##  7 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk7      99
##  8 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk8      NA
##  9 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk9      NA
## 10 2 Pac  Baby Don't Cry (Keep... 2000-02-26   wk10     NA
## # ... with 24,082 more rows
```

pivot_longer()

How does this function work?

id	bp1	bp2
A	100	120
B	140	115
C	120	125



id	measurement	value
A	bp1	100
A	bp2	120
B	bp1	140
B	bp2	115
C	bp1	120
C	bp2	125

Figure 5: Pivot longer table

pivot_wider()

pivot_wider() is the inverse of pivot_longer, in that it takes a column from a dataframe and makes it into multiple columns.

We will see examples of this later.