

4.2 Introduction to R: Work Practices

Julia Gallucci

Data Science Institute, University of Toronto

2023-05-29

Acknowledgements

Slides are adapted from Anjali Silva, originally from Amy Farrow under the supervision of Rohan Alexander, University of Toronto. Slides have been modified by Julia Gallucci, 2023.

Outline of Topics

Getting help

Using Google & Stack Overflow

Making reproducible examples (reprexes)

Reproducibility

Outline of Topics

Getting help

Using Google & Stack Overflow

Making reproducible examples (reprexes)

Reproducibility

Errors: an inherent part of coding

Practicing and gaining familiarity with the specific tasks you have to perform will reduce the frequency with which you make errors when performing those tasks.

Still, even experienced programmers make errors frequently!
Knowing how to resolve issues is an essential skill.

How to avoid errors

Use common packages and methods, especially when new to R

- ▶ If the functions you use are commonly recommended, there will be more troubleshooting materials and advice available!

Check your code regularly

- ▶ Test small sections of code before moving on to writing more. Check that the results are what you expect.
- ▶ Rerun your code periodically.
- ▶ If you want to test your code, but it takes a significant amount of time to run, consider a toy dataset.

Isolate the issue

When you get an error, or an unexpected result from your code, you need to identify exactly which piece of code is creating the problem.

- ▶ To do this, you can run the smallest chunks possible.
- ▶ Typically this involves running the code line-by-line, but you may need to run parts of code within individual lines as well.

Start debugging

Try, try, and try again!

Make a change and run your code again. Repeat.

Copy paste and edit if you want to keep track of what you are changing.

Check for typos. CHECK: Did you misspell a variable name? Miss a comma somewhere?

Did you get the could not find function error? Check that you loaded the necessary libraries.

Look at function documentation. Are you giving the arguments to the function correctly?

?mean

See if RStudio is indicating a syntax error. Read the error description.

Read the error message completely. Sometimes that's all the guidance you will need to fix the problem.

Outline of Topics

Getting help

Using Google & Stack Overflow

Making reproducible examples (reprxes)

Reproducibility

Using Google

Include “r” in your search terms. If you are using a specific package or function, include its name. Consider putting the name in quotes, as you require an exact match.

Copy paste error messages into the search bar and see what comes up. Often, many people have made the exact same error before you.

Check the dates of results. R packages and versions evolve quickly, so a solution from three years ago may no longer work.

Stack Overflow

If you search an error message, you may end up on stackoverflow.com

Read the question to see how well it matches your concern.

When reading answers, note the answer scores and which answer has been accepted.

If you are not sure if a solution will work, you might as well try! Sometimes solutions will need to be edited or combined, but you will still be closer to solving your problem than before.

Outline of Topics

Getting help

Using Google & Stack Overflow

Making reproducible examples (reprxes)

Reproducibility

What is a rerex?

There may be instances when you encounter a problem that proves difficult to solve independently, necessitating the need to seek assistance.

Whether you reach out to a friend or seek guidance on platforms like Stack Overflow, you can enhance the ease with which others can help you. By providing a rerex, which stands for a reproducible example, you enable others to reproduce the error message on their own device.

Definition:

a rerex, is a self-contained and concise piece of code that demonstrates a specific issue. It typically contains all necessary code, data and dependencies needed for others to reproduce the same error of behavior on their own computer. By providing a rerex, you make it easier for others to understand your problem and potentially provide a solution.

Components of a reprex

1. **Environment:** calls for any necessary libraries and information about your R environment that might be relevant
2. **Toy data set:** a minimal data set that the code can be run on
3. **Code:** minimal and runnable code that recreates the error

Components of a reprex

Environment:

Start a new R script for your reprex.

What libraries does your code use? Try not to load any extras. You can test your selection by restarting R, loading the selection, and running your error-generating code again to make sure it does not generate a could not find function error.

Note your R version, RStudio version, and operating system.

Help > About RStudio will show your RStudio version.

Running `sessionInfo()` will show your R version and operating system information.

Components of a reprex

Toy data set:

You can select a subset of your real dataset and attach that with your reprex.

From your dataset, select only the variables your code refers to.

From that, select a chunk of rows.

Save the dataset.

```
toy_data <- data.frame(ID = 1:4,  
  name = c("Abe", "Becca", "Calvin", "Danica"),  
  age = c(27, 32, 63, 55),  
  membership = c("yes", "no", "yes", "yes"))  
toy_data
```

	ID	name	age	membership
1	1	Abe	27	yes
2	2	Becca	32	no
3	3	Calvin	63	yes
4	4	Danica	55	yes

Components of a repro

Code:

The goal is for the code to be as succinct as possible and still generate the exact error that is causing you trouble.

Remove unnecessary lines, run the code again, and make sure that the error is still the same. Look for ways you can shorten necessary lines.

Ideally, the code should be formatted for easier reading and include comments that outline what the different lines are intended to do.

Components of a reprex

Code:

After loading the reprex package, you can highlight your reprex code(including libraries and data), copy to your clipboard, and then run.

The reprex will be automatically stored on your clipboard. You can paste it into messages asking others for help.

```
library(reprex)
reprex({
  y <- c(1:4)
mean(y)
})
```

```
y <- c(1:4)
mean(y)
#> [1] 2.5
```

Components of a reprex

Code:

Note the change in capitalization. Error message displayed in html output.

```
library(reprex)
reprex({
  y <- c(1:4)
mean(Y)
})
```

```
y <- c(1:4)
mean(Y)
#> Error in eval(expr, envir, enclos): object 'Y' not found
```

Created on 2023-05-18 with [reprex v2.0.2](#)

What if nothing works?

If you cannot find a solution to your error, you may wish to:

1. Explore alternative libraries or functions that accomplish a similar task.
2. Seek out code from someone who has successfully tackled the same task, examine their approach, and appropriately credit their work.
3. Look for potential workarounds, keeping in mind that using them could introduce new complications in the future, so exercise caution.

Remember to adhere to proper citation practices when utilizing code or methods from others to maintain ethical and academic integrity.

Outline of Topics

Getting help

Using Google & Stack Overflow

Making reproducible examples (reprexes)

Reproducibility

What is reproducibility & how do we achieve it?

Reproducible research involves recreating and replicating studies or analyses using provided materials.

It requires making code, data, and environment accessible for others to follow and reproduce the workflow.

Reproducibility is an incremental process, allowing projects to gradually improve their reproducibility over time.

The practice of reproducibility enhances accountability by enabling others to review and verify the work.

It contributes to collective knowledge by sharing processes, not just results.

Embracing reproducibility encourages acknowledging and reflecting on the limitations of data and methodologies.

Working directories

Your working directory is the folder path you can name files from. It will show in your “files” pane of RStudio

```
getwd()
```

```
[1] "/cloud/project"
```

```
#setwd("/Users/<Name>/Desktop")
```

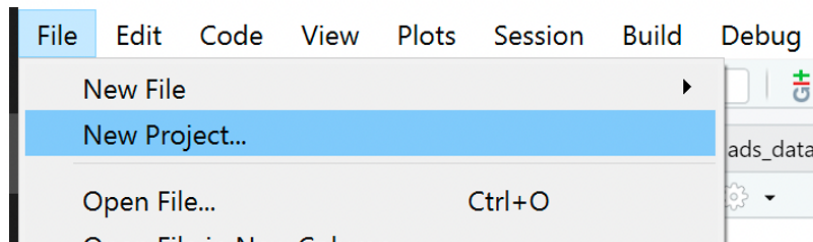
If you set your working directory manually (by using `setwd()` or through the toolbar), the folder path will be unique to your computer. If you or someone else was attempting to run your code on another device, they would first need to edit the folder path to which the working directory is set. This reduces reproducibility.

RProjects

RProjects are files with the extension `.Rproj`

They designate a working directory that starts where the `.Rproj` file is located. You don't have to use `setwd()`.






You can also set up an RProject associated with a GitHub repository to share your work with the public or a team.



Working in RProjects

Projects also keep all your work together: scripts, markdown files, data, results, figures, and more.

You can create a file structure that organizes your content.

Name	Type
 data	File folder
 figures	File folder
 papers	File folder
 scripts	File folder
 Project.Rproj	R Project

Best coding practice

Your process should be thoroughly documented. Each step should be explained.

Comment your code!

- ▶ Use comments to explain what you are doing when it is not immediately apparent
- ▶ It will help you when you look at your work 6 months from now and can't remember anything
- ▶ It will help you work with collaborators
- ▶ It will increase reproducibility if you release your code

```
#you can insert a comment line like this  
age <- 25 #or add a comment to a line like this
```

Best coding practice

Name your variables well

Use meaningful and descriptive names: Variable names should reflect the purpose or content of the variable. Avoid using single-letter names or generic names like “x.” Instead, use descriptive names that convey the variable’s meaning, such as “income” or “participant_age”

Follow a consistent naming convention. There are different conventions to choose from, such as camel case (e.g., myVariable), snake case (e.g., my_variable), or period-separated (e.g., my.variable – not preferred)

Avoid reserved words: Be mindful of R’s reserved words (e.g., if, else, for) and avoid using them as variable names. Using reserved words can lead to syntax errors and confusion.

Best coding practice

Name your variables well

Use underscores or dots for compound names: When you need to use multiple words in a variable name, you can separate them with underscores (_) or dots (.) to improve readability. For example, use “total_score” or “total.score” instead of “totalscore”

Avoid using special characters: Stick to alphanumeric characters and underscores when naming variables.

```
X <- 25 # not helpful to anyone reading your code  
subject_age <- 25 # better !
```

Best coding practice

Code for human readability

Add spaces and lines. The code will work without them, but it's worse to read.

```
ads_data %>%  
gather(key = "key", value = "val") %>%  
mutate(is.missing = is.na(val)) %>%  
group_by(key, is.missing) %>%  
summarise(num.missing = n())
```

Discussion questions

1. What debugging techniques do you think will be the most helpful? Why?
2. In your own work, what challenges do you anticipate in making processes reproducible? Discuss ways that these concerns could be addressed.

Questions?