R Coding Practices for Social Scientists

A quick and dirty introduction to writing and debugging clean, reproducible code in R

Adam Kaplan and A. Benjamin Munoz Rojas

Massachusetts Institute of Technology

November 3rd, 2023

Plan for today

1. Writing clean code
2. Quick introduction to debugging
3. Making your project replicable

Table of Contents

What is clean code?

- Your code should read like a recipe for mediocre cooks.

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?**

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.
  - **Why recipe?**

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.
  - **Why recipe?** Your code should be well-organized so that a reader knows what to expect. Maybe at first glance I want to know what is the input (ingredients), output (dish), and what are the steps to get there.

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.
  - **Why recipe?** Your code should be well-organized so that a reader knows what to expect. Maybe at first glance I want to know what is the input (ingredients), output (dish), and what are the steps to get there.
- Two important aspects that make a recipe good:

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.
  - **Why recipe?** Your code should be well-organized so that a reader knows what to expect. Maybe at first glance I want to know what is the input (ingredients), output (dish), and what are the steps to get there.
- Two important aspects that make a recipe good:
  1. Annotation and organization: I know where to find the input, and can understand the instructions even if I have never made the recipe before.

What is clean code?

- Your code should read like a recipe for mediocre cooks.
  - **Why mediocre?** Some base knowledge is expected, i.e. you do not need to explain what `x <- 2` means, but do not assume they are deeply familiar with your esoteric choice of packages.
  - **Why recipe?** Your code should be well-organized so that a reader knows what to expect. Maybe at first glance I want to know what is the input (ingredients), output (dish), and what are the steps to get there.
- Two important aspects that make a recipe good:
  1. Annotation and organization: I know where to find the input, and can understand the instructions even if I have never made the recipe before.
  2. Abstraction: Combine basic instructions into more abstract functions or states. E.g. In my bread-baking book I am told to "combine the dough using the Pincer method." When I look up the Pincer method I get 6 images and accompanying words explaining it. Imagine if the book had to repeat these 6 images and explanations every time.

Annotation and Organization: Let's talk mediocrity

Annotation:

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.
- **Variable/Function naming:** Be consistent!

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.
- **Variable/Function naming:** Be consistent!
  - I highly suggest reading a style guide and following it closely for the best results.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.
- **Variable/Function naming:** Be consistent!
  - I highly suggest reading a style guide and following it closely for the best results.
  - Even if you deviate, make sure you do so consistently. For instance you prefer `camelCase` over snake_case naming.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.
- **Variable/Function naming:** Be consistent!
  - I highly suggest reading a style guide and following it closely for the best results.
  - Even if you deviate, make sure you do so consistently. For instance you prefer `camelCase` over snake_case naming.
  - Style guides: Tidyverse, Google, ...

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- Let's tackles these in order.
- **Variable/Function naming:** Be consistent!
  - I highly suggest reading a style guide and following it closely for the best results.
  - Even if you deviate, make sure you do so consistently. For instance you prefer `camelCase` over snake_case naming.
  - Style guides: Tidyverse, Google, ...
  - No need to remember all of them $\implies$ `styler` package.

Annotation and Organization: Let's talk mediocrity

Annotation:

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this?

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this? It depends.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this? It depends.
  - I like to err on the more conservative side, so more comments.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this? It depends.
  - I like to err on the more conservative side, so more comments.
  - The bare minimum is: For functions explain input, output, and purpose; For blocks of code explain purpose and maybe some more esoteric steps within.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this? It depends.
  - I like to err on the more conservative side, so more comments.
  - The bare minimum is: For functions explain input, output, and purpose; For blocks of code explain purpose and maybe some more esoteric steps within.
  - If you followed proper naming strategies, you will not need to comment what your variables represent, but it still might be useful to mention some important information about them, such as why you chose to use a list of `matrix` objects rather than a `data.frame`, etc.

Annotation and Organization: Let's talk mediocrity

Annotation: Variable/Function naming, commenting, abstraction.

- **Abstraction:** Do not repeat code, write functions instead.
  - I would go as far as say if you repeat a code chunk that is longer than 1 line, at least once, write a function.
  - If it is a one-line statement, it depends.
  - You still need to apply proper naming and commenting hygiene!
- **Commenting:** From before, we are assuming some base knowledge, so what is this? It depends.
  - I like to err on the more conservative side, so more comments.
  - The bare minimum is: For functions explain input, output, and purpose; For blocks of code explain purpose and maybe some more esoteric steps within.
  - If you followed proper naming strategies, you will not need to comment what your variables represent, but it still might be useful to mention some important information about them, such as why you chose to use a list of `matrix` objects rather than a `data.frame`, etc.
  - Key point: The more you abstract away, and the better your naming, the fewer comments you will need!

Annotation and Organization: Let's talk mediocrity

Organization:

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.
  - In RStudio, can use headings directly by commenting `# HEADING NAME ----`.
- **Folder organization:** Keep paper, presentation, data, data wrangling, analysis, and simulations separate.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.
  - In RStudio, can use headings directly by commenting `# HEADING NAME ----`.
- **Folder organization:** Keep paper, presentation, data, data wrangling, analysis, and simulations separate.
  - Not all projects will include all of these, but try to keep them as separate as possible.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.
  - In RStudio, can use headings directly by commenting `# HEADING NAME ----`.
- **Folder organization:** Keep paper, presentation, data, data wrangling, analysis, and simulations separate.
  - Not all projects will include all of these, but try to keep them as separate as possible.
  - Benefits: Easy to find replication data and sought after code.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.
  - In RStudio, can use headings directly by commenting `# HEADING NAME ----`.
- **Folder organization:** Keep paper, presentation, data, data wrangling, analysis, and simulations separate.
  - Not all projects will include all of these, but try to keep them as separate as possible.
  - Benefits: Easy to find replication data and sought after code.
  - Costs: It is painful to work with paths, especially across operating systems.

Annotation and Organization: Let's talk mediocrity

Organization: Folder and code organization.

- **Code organization:** Just like your paper, your code will likely have sections, highlight them.
  - E.g. your analysis corresponds to different tables, figures, can label them as such in the code.
  - Commonly use some type of separators `######` or `------`.
  - In RStudio, can use headings directly by commenting `# HEADING NAME ----`.
- **Folder organization:** Keep paper, presentation, data, data wrangling, analysis, and simulations separate.
  - Not all projects will include all of these, but try to keep them as separate as possible.
  - Benefits: Easy to find replication data and sought after code.
  - Costs: It is painful to work with paths, especially across operating systems. $\Longrightarrow$ `here` package.

Let's check things out in R

- In particular we will go over a sample project and look for:

Let's check things out in R

- In particular we will go over a sample project and look for:
  1. Annotation through commenting, variable naming, and using the `styler` package.

Let's check things out in R

- In particular we will go over a sample project and look for:
  1. Annotation through commenting, variable naming, and using the `styler` package.
  2. Abstraction through the introduction of functions.

Let's check things out in R

- In particular we will go over a sample project and look for:
  1. Annotation through commenting, variable naming, and using the `styler` package.
  2. Abstraction through the introduction of functions.
  3. Organization through folder structure, code structure, and the `here` package.

Table of Contents

The Art of Debugging

- **Debugging hierarchy:** Unit tests

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging)

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.
- Some more concrete advice:

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.
- Some more concrete advice:
  - If you followed the advice until now, most of your code will be functions, which makes debugging easier, since you can trace the error to some of those functions.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.
- Some more concrete advice:
  - If you followed the advice until now, most of your code will be functions, which makes debugging easier, since you can trace the error to some of those functions.
  - Talking to yourself is a good thing.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.
- Some more concrete advice:
  - If you followed the advice until now, most of your code will be functions, which makes debugging easier, since you can trace the error to some of those functions.
  - Talking to yourself is a good thing.
  - Note that you can always run each line of a function individually and print the output to see if it matches what you expect.

The Art of Debugging

- **Debugging hierarchy:** Unit tests > Breakpoint setting and traceback > Printing intermediary output > Explaining what code does in words (Rubber-ducky debugging) > Giving up.
- Goal is to not give up. Usually printing intermediary output is effective enough.
- Unit tests are overkill if you are not coding your own package. Breakpoints and tracing code back is useful, but has a steep learning curve.
- Some more concrete advice:
  - If you followed the advice until now, most of your code will be functions, which makes debugging easier, since you can trace the error to some of those functions.
  - Talking to yourself is a good thing.
  - Note that you can always run each line of a function individually and print the output to see if it matches what you expect.
  - When you code, always debug as you go. I.e. don't write 50 functions and then start testing your code. Rather do it, a function or two at a time.

Table of Contents

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough?

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.
- **The reproducability spectrum:** Snapshot of full operating system

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.
- **The reproducability spectrum:** Snapshot of full operating system > Snapshot of R and all R packages

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.
- **The reproducability spectrum:** Snapshot of full operating system > Snapshot of R and all R packages > Code and Data

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.
- **The reproducability spectrum:** Snapshot of full operating system > Snapshot of R and all R packages > Code and Data > Nothing.

Reproducability is a spectrum

- Today most studies already submit code and data is that not enough? Sometimes and in the short term.
- R tries to be very compatible with past versions, but that is not necessarily the case for other packages. **Fun Fact:** R 2.1 was released in 2005, R 3.0 in 2013, and R 4.0 in 2020.
- We are in the long-term business, so it might make sense to make sure I can run code from 2013 in 2021 without running into weird hiccups.
- **The reproducability spectrum:** Snapshot of full operating system > Snapshot of R and all R packages > Code and Data > Nothing.
- `renv`, a hopeful `packrat` replacement fits in the 2nd best spot and should be good enough for most.

Let's check things out in R

1. We will go over how simple `renv` is to set-up in existing projects.

Let's check things out in R

1. We will go over how simple `renv` is to set-up in existing projects.
2. What does a `renv` project look like on GitHub for instance.

Let's check things out in R

1. We will go over how simple `renv` is to set-up in existing projects.
2. What does a `renv` project look like on GitHub for instance.
3. What's the process of restoring a `renv` project like.

Thank you so much!

Please feel free to reach out if you have any questions and/or suggestions!