Data Management With R

Hunter Wade York

01/27/2022

Lecture Prereqs

- This lecture meant to give you a set of tools to take you from writing bad R code to writing slightly better R code.

Lecture Prereqs

- This lecture meant to give you a set of tools to take you from writing bad R code to writing slightly better R code.
- It assumes basic proficiency in R:

Lecture Prereqs

- This lecture meant to give you a set of tools to take you from writing bad R code to writing slightly better R code.
- It assumes basic proficiency in R:
- How to assign variables; how to manipulate them using basic operations

Lecture Prereqs

- This lecture meant to give you a set of tools to take you from writing bad R code to writing slightly better R code.
- It assumes basic proficiency in R:
- How to assign variables; how to manipulate them using basic operations
- How to run a linear regression

Lecture Prereqs

- This lecture meant to give you a set of tools to take you from writing bad R code to writing slightly better R code.
- It assumes basic proficiency in R:
- How to assign variables; how to manipulate them using basic operations
- How to run a linear regression
- Basic familiarity with tidyverse syntax and operations

But first, who am I?

- Second-year PhD student in Sociology at Princeton, Office of Population Research Affiliate

Figure 1: Me!

But first, who am I?

- Second-year PhD student in Sociology at Princeton, Office of Population Research Affiliate
- Formerly: MPH at UW and researcher at Institute for Health Metrics and Evaluation, Formerly: AB at Harvard in Human Evolutionary Biology and Music

Figure 1: Me!

But first, who am I?

- Second-year PhD student in Sociology at Princeton, Office of Population Research Affiliate
- Formerly: MPH at UW and researcher at Institute for Health Metrics and Evaluation, Formerly: AB at Harvard in Human Evolutionary Biology and Music
- Interests: Stratification, culture, quant

Figure 1: Me!

Goals

- This presentation will teach you very few hard skills.

Goals

- This presentation will teach you very few hard skills.
- At best, it will introduce you to concepts that you can explore on your own to try to make up for deficiencies you notice in your code.

Goals

- This presentation will teach you very few hard skills.
- At best, it will introduce you to concepts that you can explore on your own to try to make up for deficiencies you notice in your code.
- I've tried to pitch it to be accessible to nearly all coding levels. Hopefully you can all take something away from it no matter how long you've been coding.

Goals

- This presentation will teach you very few hard skills.
- At best, it will introduce you to concepts that you can explore on your own to try to make up for deficiencies you notice in your code.
- I've tried to pitch it to be accessible to nearly all coding levels. Hopefully you can all take something away from it no matter how long you've been coding.
- If you have specific questions on topics not addressed in the presentation, ask at the end! I'm going to leave around 25 minutes for questions.

Goals

- This presentation will teach you very few hard skills.
- At best, it will introduce you to concepts that you can explore on your own to try to make up for deficiencies you notice in your code.
- I've tried to pitch it to be accessible to nearly all coding levels. Hopefully you can all take something away from it no matter how long you've been coding.
- If you have specific questions on topics not addressed in the presentation, ask at the end! I'm going to leave around 25 minutes for questions.
- If you have questions during the presentation, STOP ME! No question is too simple.

Useful Links

- What They Forgot to Teach You About R

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!
- R for Data Science

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!
- R for Data Science
- Data.table cheatsheet

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!
- R for Data Science
- Data.table cheatsheet
- Dplyr cheatsheet

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!
- R for Data Science
- Data.table cheatsheet
- Dplyr cheatsheet
- ggplot cheatsheet

Useful Links

- What They Forgot to Teach You About R
- R for Social Scientists
- Git for Novices
- Introduction to Data.Table
- Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
- The Summer Institutes in Computational Social Science!!
- R for Data Science
- Data.table cheatsheet
- Dplyr cheatsheet
- ggplot cheatsheet
- Other cheatsheets

Questions?

Why I'm teaching what I'm teaching.

Outline

- Project Design and Data Management

Outline

- Project Design and Data Management
- Code Optimization

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of
  - Profiling

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of
  - Profiling
  - Vectorizing Code

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of
  - Profiling
  - Vectorizing Code
  - Some examples of bad code

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of
  - Profiling
  - Vectorizing Code
  - Some examples of bad code
- Data Management and Reshaping Efficiently

Outline

- Project Design and Data Management
- Code Optimization
  - Principles of
  - Profiling
  - Vectorizing Code
  - Some examples of bad code
- Data Management and Reshaping Efficiently
- Parallelization

Project Design and Data Management

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)
- If you make tweaks to your code writing process, which may take an initial investment of time, the time saved in the long run will be worth it.

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)
- If you make tweaks to your code writing process, which may take an initial investment of time, the time saved in the long run will be worth it.
    - You don't have to be a computer scientist to write amazing R code.

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)
- If you make tweaks to your code writing process, which may take an initial investment of time, the time saved in the long run will be worth it.
  - You don't have to be a computer scientist to write amazing R code.
- Learn a workflow, tailor it to your needs, replicate it endlessly.

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)
- If you make tweaks to your code writing process, which may take an initial investment of time, the time saved in the long run will be worth it.
  - You don't have to be a computer scientist to write amazing R code.
- Learn a workflow, tailor it to your needs, replicate it endlessly.
  - MANY R tips and tricks online will not be useful to you. Social scientists in academia are different from social scientists in industry, from data scientists, and from all other R users (hard sciences, statisticians, etc.).

Some Code-Writing Theory

- Don't rewrite your code or a pipeline unless the time saved is greater than the time invested (for you or for anyone using your code)
- If you make tweaks to your code writing process, which may take an initial investment of time, the time saved in the long run will be worth it.
  - You don't have to be a computer scientist to write amazing R code.
- Learn a workflow, tailor it to your needs, replicate it endlessly.
  - MANY R tips and tricks online will not be useful to you. Social scientists in academia are different from social scientists in industry, from data scientists, and from all other R users (hard sciences, statisticians, etc.).
- Version often, save versions remotely (git + github or even dropbox)

What is a project

- Knowing where your files are allows you to spend more time doing what's important: writing code!

Figure 2: Organize!

What is a project

- Knowing where your files are allows you to spend more time doing what's important: writing code!
- A project is simply, start to finish, everything to get you from idea to finished paper

Figure 2: Organize!

What is a project

- Knowing where your files are allows you to spend more time doing what's important: writing code!
- A project is simply, start to finish, everything to get you from idea to finished paper
- It includes code, data, output files like clean datasets, figures, and possibly a manuscript

Figure 2: Organize!

What is a project

- Knowing where your files are allows you to spend more time doing what's important: writing code!
- A project is simply, start to finish, everything to get you from idea to finished paper
- It includes code, data, output files like clean datasets, figures, and possibly a manuscript
- Most importantly, by having all your files in one directory and using relational paths, you make the entire project replicable!

Figure 2: Organize!

My Workflow (starting a new project)

- Create a folder on my computer for each project.

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects
  - "../code", "../inputs", "../outputs", "../ref"

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects
  - "../code", "../inputs", "../outputs", "../ref"
- Start a git repository in the main directory and keep up-to-date with GitHub.

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects
  - "../code", "../inputs", "../outputs", "../ref"
- Start a git repository in the main directory and keep up-to-date with GitHub.
  - For a helpful introduction to Git a. GitHub: https://www.analyticsvidhya.com/blog/2020/05/git-github-essential-guide-beginners/

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects
  - "../code", "../inputs", "../outputs", "../ref"
- Start a git repository in the main directory and keep up-to-date with GitHub.
  - For a helpful introduction to Git a. GitHub: https://www.analyticsvidhya.com/blog/2020/05/git-github-essential-guide-beginners/
  - Super Basic Git Workflow for collaboration (fork, edit, push, pull request)

My Workflow (starting a new project)

- Create a folder on my computer for each project.
  - "/Users/hyork/Documents/projects/NAME_OF_PROJECT"
- Format subdirectories in a similar manner for all projects
  - "../code", "../inputs", "../outputs", "../ref"
- Start a git repository in the main directory and keep up-to-date with GitHub.
  - For a helpful introduction to Git a. GitHub: https://www.analyticsvidhya.com/blog/2020/05/git-github-essential-guide-beginners/
  - Super Basic Git Workflow for collaboration (fork, edit, push, pull request)
  - NB: Git has a lot of features built out for collaboration that are not necessary for basic data management. If you learn a simple "add," "commit," "push" workflow, it will serve most of your needs.

Workflows

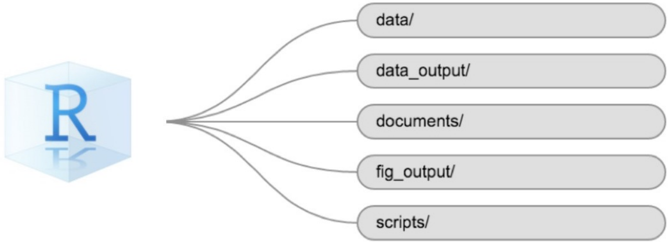


Figure 3: Another Example Directory Setup

A Note on Git

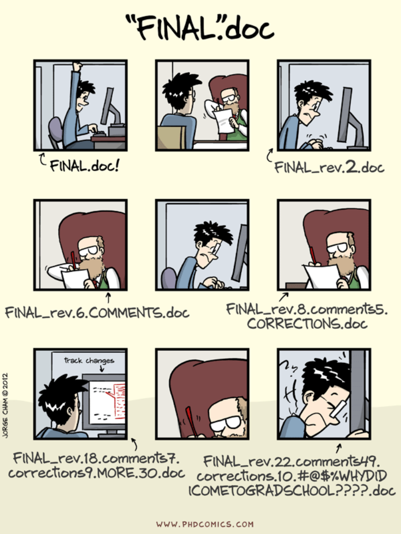


Figure 4: www.phdcomics.com

A Note on Git

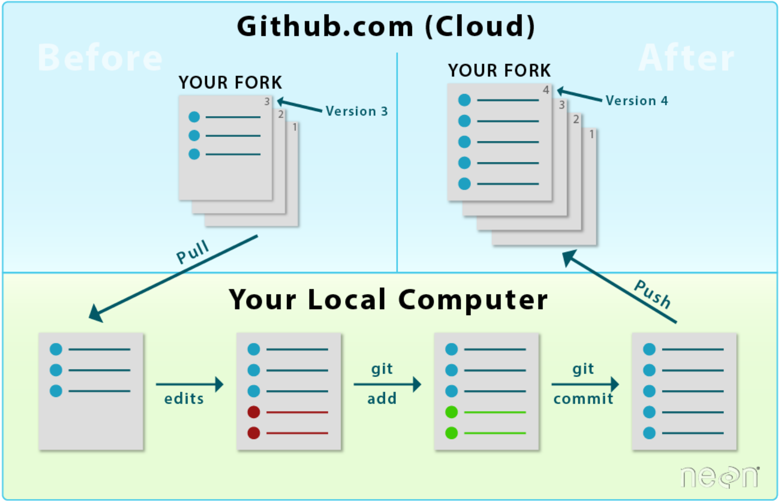


Figure 5: https://www.neonscience.org/resources/learning-hub/tutorials/github-git-add

Code Tips

1. Break down large projects into smaller chunks.

- For me, this usually looks something like having a "processing.R", "anlaysis.R", and a "figures.R" script.
    - Tailor these to your specific project. If you have many lines of data acquisition and many lines of data processing, break that up!
- Real programmers, data scientists working to make reproducible pipelines, etc. will all have drastically different standards of coding. Don't listen to them. Unless you're making a package to put on CRAN, you don't need a script for helper functions, etc.
- That said, if one of your files exceeds 1,000 lines, or you have a very time-consuming step in the middle of a script, consider breaking it up.
- I love to save intermediate files in my scripts. Later, these form a natural place for me to break a script up if it gets too long.

Code Tips

Workflows (within your "../scripts" or "../code" folder)

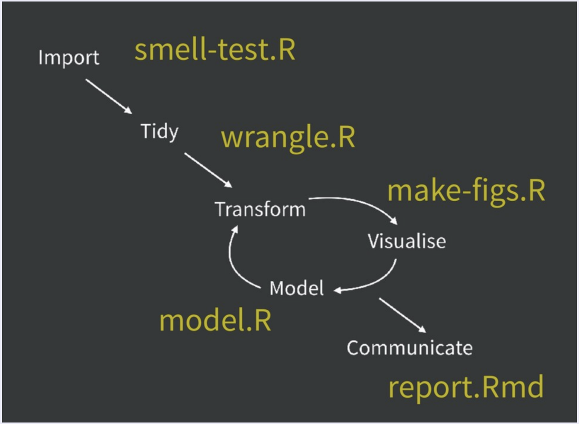


Figure 6: "../code"

https://speakerdeck.com/jennybc/zen-and-the-art-of-workflow-maintenance?slide=59

Code Tips

2. Comment (when it helps you)!
   - I actually rarely comment well until the final stages of a pipeline. BUT, that is because I'm so used to using a certain set of tools, that I can almost always tell what I'm doing. My comments are thus more limited to reminding myself why I made a certain choice or flags for me to revisit a small bug.
3. Test clunky operations on smaller bits of data
   - Sometimes we get too ambitious. If your code is slow and you can't or don't want to optimize it, subset your data early on in the script, write your code using the subset, and in the last stage, run it on the full sample.

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

Example of using your own line breaks to make code neater

```
c("Hunter", "Mary", "Sol",
  "Joseph", "Jamal", "Carla",
  "Sara", "Emma", "Rod")
```

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

Example of using your own line breaks to make code neater

```
c("Hunter", "Mary", "Sol",
  "Joseph", "Jamal", "Carla",
  "Sara", "Emma", "Rod")
```

- I follow no particular style guide, but you will eventually figure out your own style.

An aside on aesthetics

- **Aesthetics matter!**. . . *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

Example of using your own line breaks to make code neater

```
c("Hunter", "Mary", "Sol",
  "Joseph", "Jamal", "Carla",
  "Sara", "Emma", "Rod")
```

- I follow no particular style guide, but you will eventually figure out your own style.
- http://adv-r.had.co.nz/Style.html

An aside on aesthetics

- **Aesthetics matter!**... *kind of*
- [CMD] + i will automatically indent your code to make it look better
- Line breaks can be done really however you'd like, though consistency will make it easier to read.
  - If listing variables, for instance, I will often break after every 3 units or so.

Example of using your own line breaks to make code neater

```
c("Hunter", "Mary", "Sol",
  "Joseph", "Jamal", "Carla",
  "Sara", "Emma", "Rod")
```

- I follow no particular style guide, but you will eventually figure out your own style.
- http://adv-r.had.co.nz/Style.html
- lintr

Mass Collaboration

- I've worked on several 20+ author papers (and a few 1000+), but usually one one or two people are really in charge of data analysis, or it uses a structure where people work on separate chunks of code.

Mass Collaboration

- I've worked on several 20+ author papers (and a few 1000+), but usually one one or two people are really in charge of data analysis, or it uses a structure where people work on separate chunks of code.
- Git provides a more truly collaborative alternative!

Mass Collaboration

- I've worked on several 20+ author papers (and a few 1000+), but usually one one or two people are really in charge of data analysis, or it uses a structure where people work on separate chunks of code.
- Git provides a more truly collaborative alternative!
- Collaboration is hard. Tools like git, trello, and slack help.

Mass Collaboration

- I've worked on several 20+ author papers (and a few 1000+), but usually one one or two people are really in charge of data analysis, or it uses a structure where people work on separate chunks of code.
- Git provides a more truly collaborative alternative!
- Collaboration is hard. Tools like git, trello, and slack help.
- Working in a lab setting? Use a lab handbook: link

Mass Collaboration

- I've worked on several 20+ author papers (and a few 1000+), but usually one one or two people are really in charge of data analysis, or it uses a structure where people work on separate chunks of code.
- Git provides a more truly collaborative alternative!
- Collaboration is hard. Tools like git, trello, and slack help.
- Working in a lab setting? Use a lab handbook: link
- Keys: norms, coordination, and communication.

Questions?

Code Optimization

Theory of Code Optimization

- I only optimize code that is essential to my coding process.

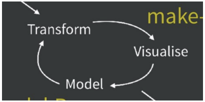


Figure 7: "../code"

But how do you fix a bottleneck? . . . .

Theory of Code Optimization

- I only optimize code that is essential to my coding process.

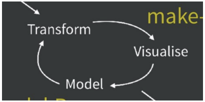


Figure 7: "../code"

- Code is very often slow-running because of bottlenecks.

But how do you fix a bottleneck? . . . .

Theory of Code Optimization

- I only optimize code that is essential to my coding process.

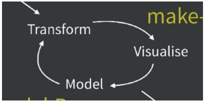


Figure 7: "../code"

- Code is very often slow-running because of bottlenecks.
- Tweaking bottlenecks is often worth it.

But how do you fix a bottleneck? . . . .

Theory of Code Optimization

- I only optimize code that is essential to my coding process.

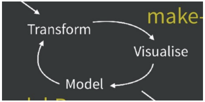


Figure 7: "../code"

- Code is very often slow-running because of bottlenecks.
- Tweaking bottlenecks is often worth it.
- If a bottleneck cannot be fixed or if it's not worth it to fix it, consider removing it from your code.

But how do you fix a bottleneck? ....

Theory of Code Optimization

- I only optimize code that is essential to my coding process.

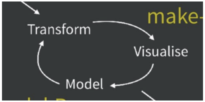


Figure 7: "../code"

- Code is very often slow-running because of bottlenecks.
- Tweaking bottlenecks is often worth it.
- If a bottleneck cannot be fixed or if it's not worth it to fix it, consider removing it from your code.
- Make a separate script for it, that way you're only editing the chunk of code before or after it modularly, and you're not running the bottleneck every time you rerun your script.

But how do you fix a bottleneck? ....

Theory of Code Optimization

Problem

- You have code that runs seems to be running slow but you dont know how to identify the rate-limiting step

Theory of Code Optimization

Problem

- You have code that runs seems to be running slow but you dont know how to identify the rate-limiting step

Theory of Code Optimization

Problem

- You have code that runs seems to be running slow but you dont know how to identify the rate-limiting step

Solution

- RStudio's built-in profiler! (Or bespoke profiling)

Speeding up code

Profiling with rstudio's built-in profiler

- The basic theory of profiling is that it helps you locate bottlenecks in your process.

Speeding up code

Profiling with rstudio's built-in profiler

- The basic theory of profiling is that it helps you locate bottlenecks in your process.
  - It may also help you identify what part of the bottleneck is the most critical thing to fix.

Speeding up code

Profiling with rstudio's built-in profiler

- The basic theory of profiling is that it helps you locate bottlenecks in your process.
  - It may also help you identify what part of the bottleneck is the most critical thing to fix.
- Often times, you can tell what a bottleneck is by simply looking at your code and watching it run.

Speeding up code

Profiling with rstudio's built-in profiler

- The basic theory of profiling is that it helps you locate bottlenecks in your process.
  - It may also help you identify what part of the bottleneck is the most critical thing to fix.
- Often times, you can tell what a bottleneck is by simply looking at your code and watching it run.
- However, if you're running a for loop or a function, seeing inside the chunk is much harder.

Speeding up code

Profiling with rstudio's built-in profiler

- The basic theory of profiling is that it helps you locate bottlenecks in your process.
  - It may also help you identify what part of the bottleneck is the most critical thing to fix.
- Often times, you can tell what a bottleneck is by simply looking at your code and watching it run.
- However, if you're running a for loop or a function, seeing inside the chunk is much harder.
- You can profile any amount of code, allowing you to optimize within and outside of such chunks.

Speeding up Code

Profiling with rstudio's built-in profiler

- Profiling

Here's an example of some code we're going to profile

```r
f <- function() {
  profvis::pause(0.1)
  g()
  h()
}
g <- function() {
  profvis::pause(.1)
  h()
}
h <- function() {
  profvis::pause(0.1)
}
profvis::profvis({f()})
```

Speeding up Code

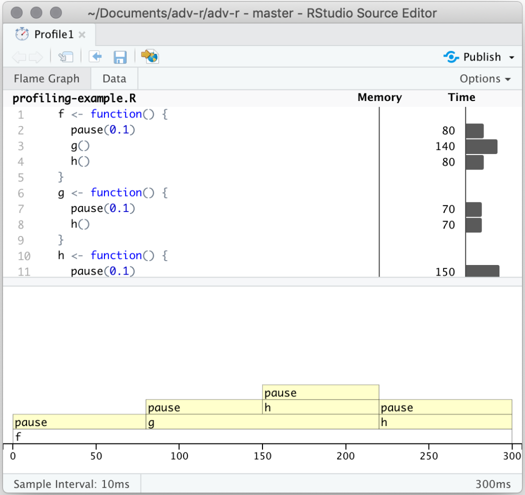
Profiling with rstudio's built-in profiler

- Profiling
  - Profiling Introduction Link

Here's an example of some code we're going to profile

```
f <- function() {
  profvis::pause(0.1)
  g()
  h()
}
g <- function() {
  profvis::pause(.1)
  h()
}
h <- function() {
  profvis::pause(0.1)
}
profvis::profvis({f()})
```

Speeding up Code



An alternative to profiling

You can also wrap your code in *system.time({})* for a more bespoke analysis.

```
library(data.table)
system.time({Sys.sleep(5)})
```

```
##    user  system elapsed
##   0.000   0.000   5.005
```

```
system.time({Sys.sleep(1)})
```

```
##    user  system elapsed
##   0.000   0.000   1.004
```

Speeding up code, a review

- Not all code needs to be fast.

Speeding up code, a review

- Not all code needs to be fast.
- Profiling with Rstudio's built-in profiler is very handy!

Speeding up code, a review

- Not all code needs to be fast.
- Profiling with Rstudio's built-in profiler is very handy!
- Alternatively, use *system.time*.

Speeding up code, a review

- Not all code needs to be fast.
- Profiling with Rstudio's built-in profiler is very handy!
- Alternatively, use *system.time*.
- If you don't want to make code more efficient, excise the clunky bits from your script so you don't keep running it every time you run your script.

Questions?

Speeding Up Code - Going from bad for-loops to better for-loops to vectorized code (functions + lapply)

Functions

"To become significantly more reliable, code must become more transparent. In particular, nested conditions and loops must be viewed with great suspicion. Complicated control flows confuse programmers. Messy code often hides bugs."

— Bjarne Stroustrup

"A common use of functionals is as an alternative to for loops. For loops have a bad rap in R because many people believe they are slow, but the real downside of for loops is that they're very flexible: a loop conveys that you're iterating, but not what should be done with the results."

— Hadley Wickham https://adv-r.hadley.nz/functionals.html

Speeding up code: vectorizing Code

- For loops are often memory intensive

Speeding up code: vectorizing Code

- For loops are often memory intensive
  - Common use-case: grow an object with each iteration of the loop. This might be something like "rbind" to add a row to the end of a dataframe.

Speeding up code: vectorizing Code

- For loops are often memory intensive
  - Common use-case: grow an object with each iteration of the loop. This might be something like "rbind" to add a row to the end of a dataframe.
  - Each iteration reallocates the object in your computer's memory.

Speeding up code: vectorizing Code

- For loops are often memory intensive
  - Common use-case: grow an object with each iteration of the loop. This might be something like "rbind" to add a row to the end of a dataframe.
  - Each iteration reallocates the object in your computer's memory.
- Alternatives including allocating the object beforehand or using lapply, sapply, etc.

Speeding up code: vectorizing Code

- For loops are often memory intensive
  - Common use-case: grow an object with each iteration of the loop. This might be something like “rbind” to add a row to the end of a dataframe.
  - Each iteration reallocates the object in your computer’s memory.
- Alternatives including allocating the object beforehand or using lapply, sapply, etc.
- lapply, sapply, mapply and functions are a more efficient way to code, and being in a function-based mindset can help you tackle problems that might be hard to wrap your head around with alternatives in R.

Speeding up code: vectorizing Code

- For loops are often memory intensive
  - Common use-case: grow an object with each iteration of the loop. This might be something like “rbind” to add a row to the end of a dataframe.
  - Each iteration reallocates the object in your computer’s memory.
- Alternatives including allocating the object beforehand or using lapply, sapply, etc.
- lapply, sapply, mapply and functions are a more efficient way to code, and being in a function-based mindset can help you tackle problems that might be hard to wrap your head around with alternatives in R.
- Vectorization here applies to “Array Programming” which simply means applying a function to an array all at once instead of piecewise.

A refresher on functions

- swcarpentry has a good review of functions.

A refresher on functions

- swcarpentry has a good review of functions.
- Functions allow a method to be replicated in different parts of your code without rewriting the method

A refresher on functions

- swcarpentry has a good review of functions.
- Functions allow a method to be replicated in different parts of your code without rewriting the method
- They are written by wrapping a block of code in:

A refresher on functions

- swcarpentry has a good review of functions.
- Functions allow a method to be replicated in different parts of your code without rewriting the method
- They are written by wrapping a block of code in:
    - *function(arg1, arg2, ...){...code goes here}*

A refresher on functions

- swcarpentry has a good review of functions.
- Functions allow a method to be replicated in different parts of your code without rewriting the method
- They are written by wrapping a block of code in:
  - *function(arg1, arg2, ...){...code goes here}*
- They use the syntax *FUN(arg1, arg2, ...)* to evaluate

A refresher on functions

- swcarpentry has a good review of functions.
- Functions allow a method to be replicated in different parts of your code without rewriting the method
- They are written by wrapping a block of code in:
  - *function(arg1, arg2, ...){...code goes here}*
- They use the syntax *FUN(arg1, arg2, ...)* to evaluate
- *lapply* (list apply) takes the arguments X and FUN where X is a list or vector of items to iterate the function over, and FUN is the function. You can also pass on other arguments but they must stay the same for all evaluations of the function.

A refresher on functions

An example of a basic function, how to use lapply

```
# Serial replication can be replaced easily!
x <- 1 + 1
y <- 2 + 1
z <- 3 + 1
```

A refresher on functions

An example of a basic function, how to use lapply

```r
# Serial replication can be replaced easily!
x <- 1 + 1
y <- 2 + 1
z <- 3 + 1
```

```r
# Write a function and assign it as an object
add_1 <- function(w){
  return(w + 1)
}
```

A refresher on functions

An example of a basic function, how to use lapply

```
# Serial replication can be replaced easily!
x <- 1 + 1
y <- 2 + 1
z <- 3 + 1
```

```
# Write a function and assign it as an object
add_1 <- function(w){
  return(w + 1)
}
```

```
# Run a function on one thing at a time
add_1(1)
```

```
## [1] 2
```

```
add_1(2)
```

```
## [1] 3
```

A refresher on functions

An example of a basic function, how to use lapply

```
# Serial replication can be replaced easily!
x <- 1 + 1
y <- 2 + 1
z <- 3 + 1
```

```
# Write a function and assign it as an object
add_1 <- function(w){
  return(w + 1)
}
```

```
# Run a function on one thing at a time
add_1(1)
```

```
## [1] 2
```

```
add_1(2)
```

```
## [1] 3
```

```
# Use lapply to run it over a list (or vector)
lapply(3:5, add_1)
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 5
##
## [[3]]
## [1] 6
```

Quick Data Introduction

- For the next few slides, I'll be using data from NLSY 97, a file containing basic demographic information and yearly income numbers for each participant.

```
source("song.R")
setnames(new_data, c("PUBID", "inc_1997", "sex", "bdate_mo",
                     "bdate_yr", "sample_type", "race",
                     paste0("inc_", 1998:2011),
                     paste0("inc_", seq(2013, 2019, 2))))
new_data <- new_data[,c(1,3,7, 21:25)]
new_data <- data.table(new_data)
new_data <- new_data[complete.cases(new_data)]
head(new_data)
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
## 3:     5   1    2    68000    76000   125000   125000   127000
## 4:     9   1    4    30000    54000    57000    59000    90000
## 5:    11   2    2    17000    33000    36000    36000    38000
## 6:    22   1    2    40000    50000    52000    75000    52000
```

Replacing for-loops with functions > - *Single Iteration Example*

- This task iterates over model designs. Vroom is a package in dplyr made to do a similar task. This might be useful in an exploratory data analysis or in a robustness check in a sensitivity analysis.

Setting up the task, choosing variables to iterate over as dependent variables in a regression.

```
regress_vars <- names(new_data)[2:length(names(new_data))] # select variables to use in analysis
```

Replacing for-loops with functions > - *Single Iteration Example*

- This task iterates over model designs. Vroom is a package in dplyr made to do a similar task. This might be useful in an exploratory data analysis or in a robustness check in a sensitivity analysis.

Setting up the task, choosing variables to iterate over as dependent variables in a regression.

```
regress_vars <- names(new_data)[2:length(names(new_data))] # select variables to use in analysis
```

```
c.vars <- regress_vars[c(1,2,3)] # select ind. variables
```

Replacing for-loops with functions > - *Single Iteration Example*

- This task iterates over model designs. Vroom is a package in dplyr made to do a similar task. This might be useful in an exploratory data analysis or in a robustness check in a sensitivity analysis.

Setting up the task, choosing variables to iterate over as dependent variables in a regression.

```
regress_vars <- names(new_data)[2:length(names(new_data))] # select variables to use in analysis
```

```
c.vars <- regress_vars[c(1,2,3)] # select ind. variables
```

```
c.outcome <- regress_vars[7] # select dep. var
```

Replacing for-loops with functions > - *Single Iteration Example*

- This task iterates over model designs. Vroom is a package in dplyr made to do a similar task. This might be useful in an exploratory data analysis or in a robustness check in a sensitivity analysis.

Setting up the task, choosing variables to iterate over as dependent variables in a regression.

```
regress_vars <- names(new_data)[2:length(names(new_data))] # select variables to use in analysis
```

```
c.vars <- regress_vars[c(1,2,3)] # select ind. variables
```

```
c.outcome <- regress_vars[7] # select dep. var
```

```
print(c.vars)
```

```
## [1] "sex"      "race"     "inc_2011"
```

```
print(c.outcome)
```

```
## [1] "inc_2019"
```

Replacing for-loops with functions - *Single Iteration Example*

Copy data, collapse variables into a formula

```
c.dat <- copy(new_data) # copy data frame for this analysis
                        # This is unnecessary but helps us prepare
                        # to use a function
```

Replacing for-loops with functions - *Single Iteration Example*

Copy data, collapse variables into a formula

```
c.dat <- copy(new_data) # copy data frame for this analysis
                        # This is unnecessary but helps us prepare
                        # to use a function
```

```
form <- as.formula(             # collapse ind. and dep. vars to a formula
  paste0(c.outcome, " ~ ",
         paste0(c.vars, collapse = " + "))
)
```

Replacing for-loops with functions - *Single Iteration Example*

Copy data, collapse variables into a formula

```
c.dat <- copy(new_data) # copy data frame for this analysis
                        # This is unnecessary but helps us prepare
                        # to use a function
```

```
form <- as.formula(             # collapse ind. and dep. vars to a formula
  paste0(c.outcome, " ~ ",
         paste0(c.vars, collapse = " + "))
)
```

```
print(form)             # Print the formula
```

```
## inc_2019 ~ sex + race + inc_2011
```

Questions?

Replacing for-loops with functions - *Single Iteration Example*

Replacing for-loops with functions - *Single Iteration Example*

Run the model and collect the results in a data.table

```
mod <- lm(formula = form, data = c.dat) # evaluate model
```

Replacing for-loops with functions - *Single Iteration Example*

Run the model and collect the results in a data.table

```
mod <- lm(formula = form, data = c.dat) # evaluate model
```

```
mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5),
                        keep.rownames = T) # here I'm using data.table syntax
                                           # but this can be done in base R or dplyr
```

Replacing for-loops with functions - *Single Iteration Example*

Run the model and collect the results in a data.table

```
mod <- lm(formula = form, data = c.dat) # evaluate model
```

```
mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5),
                        keep.rownames = T) # here I'm using data.table syntax
                                           # but this can be done in base R or dplyr
```

```
mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")] # Create an ID variable to know which model I ran
```

Replacing for-loops with functions - *Single Iteration Example*

Run the model and collect the results in a data.table

```r
mod <- lm(formula = form, data = c.dat) # evaluate model
```

```r
mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5),
                        keep.rownames = T) # here I'm using data.table syntax
                                           # but this can be done in base R or dplyr
```

```r
mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")] # Create an ID variable to know which model I ran
```

```r
mod_betas[1:4] # display outputs: estimates and SEs for each coefficient, with ID vars to describe model run
```

```
##             rn    Estimate Std. Error            ind_vars
## 1: (Intercept) 30633.54723 3785.01125 sex, race, inc_2011
## 2:         sex -9371.84648 1766.57572 sex, race, inc_2011
## 3:        race  2065.22707  696.15427 sex, race, inc_2011
## 4:    inc_2011     1.14018    0.03358 sex, race, inc_2011
```

Questions?

Replacing for-loops with functions - *Bad For-Loop Version (multiple iteration example)*

Replacing for-loops with functions - *For-Loop Version (multiple iteration example)*

- Now we want to do the same thing, using different model specifications.

Replacing for-loops with functions - *For-Loop Version (multiple iteration example)*

- Now we want to do the same thing, using different model specifications.
- In the for loop, I will loop over different combinations of printed variables. In the bad for loop example, I append my data.table to a preexisting data.table, which takes a lot of memory and time. This gets worse as the data.table grows longer (as the for loop keeps going).

Replacing for-loops with functions - *For-Loop Version (multiple iteration example)*

Step 1

- Establish a list of independent variable combinations to run model on. This is a list of vectors.

```
vars_vec <- replicate(3, expr =
                        sample(regress_vars[regress_vars != "inc_2019"], 3), simplify = F)
                        # above, I chose some random independent variables by
                        # sampling from all the possible choices and I assembled
print(vars_vec[1:2])    # them into a list of combinations. Here I print them.
```

```
## [[1]]
## [1] "race"     "sex"      "inc_2011"
##
## [[2]]
## [1] "inc_2011" "inc_2017" "inc_2013"
```

Replacing for-loops with functions

Step 2 - Run a model over each combination of independent variables

```
all_betas <- data.table() # Establish an empty data.table to hold results
```

Replacing for-loops with functions

Step 2 - Run a model over each combination of independent variables

```
all_betas <- data.table() # Establish an empty data.table to hold results
```

```
for(c.vars in vars_vec){  # loop over each combination of independent variables to regress
  form <- as.formula(     # create formula using ind. and dep. variables
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(formula = form, data = c.dat)  # Run model

  # Accumulate summary stats for each variable in regresssion
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")] # Create an variable to know which model I ran
  # The below step is the most memory intensive!
  # It rewrites the entire object, growing larger with each iteration.
  all_betas <- rbind(all_betas, mod_betas) # append results to preexisting data table
}
```

Time for this code chunk to run: 0.0235040187835693

Replacing for-loops with functions

Step 2 - Run a model over each combination of independent variables

```
all_betas <- data.table() # Establish an empty data.table to hold results
```

```
for(c.vars in vars_vec){  # loop over each combination of independent variables to regress
  form <- as.formula(     # create formula using ind. and dep. variables
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(formula = form, data = c.dat)  # Run model

  # Accumulate summary stats for each variable in regresssion
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")] # Create an variable to know which model I ran
  # The below step is the most memory intensive!
  # It rewrites the entire object, growing larger with each iteration.
  all_betas <- rbind(all_betas, mod_betas) # append results to preexisting data table
}
```

Time for this code chunk to run: 0.0235040187835693

```
head(all_betas)
```

```
##             rn    Estimate Std. Error                      ind_vars
## 1: (Intercept) 30633.54723 3785.01125           race, sex, inc_2011
## 2:        race  2065.22707  696.15427           race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572           race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358           race, sex, inc_2011
## 5: (Intercept)  2328.47001 1121.22181 inc_2011, inc_2017, inc_2013
## 6:    inc_2011     0.05514    0.03903 inc_2011, inc_2017, inc_2013
```

Questions?

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Better for-loop version - Write to a list of data.tables

- This version of the same code replaces the object that holds the outcomes of the different models. Instead of appending to one ever-growing data.table, we append each model result (a data.table) to a list of data.tables. This doesn't require removing the list from memory and replacing it.

```
all_betas_list <- list() # create an empty list
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Better for-loop version - Write to a list of data.tables

- This version of the same code replaces the object that holds the outcomes of the different models. Instead of appending to one ever-growing data.table, we append each model result (a data.table) to a list of data.tables. This doesn't require removing the list from memory and replacing it.

```
all_betas_list <- list() # create an empty list
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Better for-loop version - Write to a list of data.tables

- This version of the same code replaces the object that holds the outcomes of the different models. Instead of appending to one ever-growing data.table, we append each model result (a data.table) to a list of data.tables. This doesn't require removing the list from memory and replacing it.

```
all_betas_list <- list() # create an empty list
```

```
i <- 0                   # create an index to add to with each iteration
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Better for-loop version - Write to a list of data.tables

- This version of the same code replaces the object that holds the outcomes of the different models. Instead of appending to one ever-growing data.table, we append each model result (a data.table) to a list of data.tables. This doesn't require removing the list from memory and replacing it.

```
all_betas_list <- list() # create an empty list
```

```
i <- 0                  # create an index to add to with each iteration
```

```
for(c.vars in vars_vec){
  i <- i + 1             # add to iterator
  form <- as.formula(    # create formula, as above
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(formula = form, data = c.dat) # Run model
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")]
  # This step saves memory by adding an element to a list
  # It does not copy the list or overwrite it in doing so
  all_betas_list[[i]] <- mod_betas
}
```

Time for this code chunk to run: 0.0269491672515869

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

*The results are identical!)

Better for-loop version - Write to a list of data.tables

```
all_betas_list[1] # show what a list of data.tables looks like
```

```
## [[1]]
##             rn    Estimate Std. Error            ind_vars
## 1: (Intercept) 30633.54723 3785.01125 race, sex, inc_2011
## 2:        race  2065.22707  696.15427 race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572 race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358 race, sex, inc_2011
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

*The results are identical!)

Better for-loop version - Write to a list of data.tables

```r
all_betas_list[1] # show what a list of data.tables looks like
```

```
## [[1]]
##              rn    Estimate Std. Error            ind_vars
## 1: (Intercept) 30633.54723 3785.01125 race, sex, inc_2011
## 2:        race  2065.22707  696.15427 race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572 race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358 race, sex, inc_2011
```

```r
all_betas <- rbindlist(all_betas_list) # you can collapse the list of data.tables
                                       # to one large data.table
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

*The results are identical!)

Better for-loop version - Write to a list of data.tables

```
all_betas_list[1] # show what a list of data.tables looks like
```

```
## [[1]]
##             rn    Estimate Std. Error           ind_vars
## 1: (Intercept) 30633.54723 3785.01125 race, sex, inc_2011
## 2:        race  2065.22707  696.15427 race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572 race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358 race, sex, inc_2011
```

```
all_betas <- rbindlist(all_betas_list) # you can collapse the list of data.tables
                                       # to one large data.table
```

```
head(all_betas)
```

```
##             rn    Estimate Std. Error                     ind_vars
## 1: (Intercept) 30633.54723 3785.01125          race, sex, inc_2011
## 2:        race  2065.22707  696.15427          race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572          race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358          race, sex, inc_2011
## 5: (Intercept)  2328.47001 1121.22181 inc_2011, inc_2017, inc_2013
## 6:    inc_2011     0.05514    0.03903 inc_2011, inc_2017, inc_2013
```

Questions?

Replacing for-loops with functions - *Functions and Vectorization! (multiple iteration example)*

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Functions + lapply (list apply)

```
modr <- function(c.dat, c.vars, c.outcome){ # write a function that takes in 3 arguments
  form <- as.formula(
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(form, data = c.dat)
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")]
  return(mod_betas) # return the data.table
}
```

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Functions + lapply (list apply)

```
modr <- function(c.dat, c.vars, c.outcome){ # write a function that takes in 3 arguments
  form <- as.formula(
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(form, data = c.dat)
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")]
  return(mod_betas) # return the data.table
}
```

```
lapply(vars_vec, modr, c.outcome = "inc_2019", c.dat = new_data) %>%
  rbindlist() -> all_betas
```

Time for this code chunk to run: 0.0237009525299072

Replacing for-loops with functions - *Better For-Loop Version (multiple iteration example)*

Functions + lapply (list apply)

```
modr <- function(c.dat, c.vars, c.outcome){ # write a function that takes in 3 arguments
  form <- as.formula(
    paste0(c.outcome, " ~ ",
           paste0(c.vars, collapse = " + "))
  )
  mod <- lm(form, data = c.dat)
  mod_betas <- data.table(round(summary(mod)$coefficients[, 1:2],5), keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")]
  return(mod_betas) # return the data.table
}
```

```
lapply(vars_vec, modr, c.outcome = "inc_2019", c.dat = new_data) %>%
  rbindlist() -> all_betas
```

Time for this code chunk to run: 0.0237009525299072

```
head(all_betas)
```

```
##              rn    Estimate Std. Error                      ind_vars
## 1: (Intercept) 30633.54723 3785.01125           race, sex, inc_2011
## 2:        race  2065.22707  696.15427           race, sex, inc_2011
## 3:         sex -9371.84648 1766.57572           race, sex, inc_2011
## 4:    inc_2011     1.14018    0.03358           race, sex, inc_2011
## 5: (Intercept)  2328.47001 1121.22181 inc_2011, inc_2017, inc_2013
## 6:    inc_2011     0.05514    0.03903 inc_2011, inc_2017, inc_2013
```

Figure 9: Tweet

Some actual examples of less-than-ideal code I found on the internet

Original Code - Repeated tasks

```r
# ORIGINAL
#remove idnumb, other ids
data[which(colnames(data) == "idnumb")] <- NULL
data[which(colnames(data) == "challengeIDs")] <- NULL
data[which(colnames(data) == "mothids1")] <- NULL
data[which(colnames(data) == "mothids2")] <- NULL
data[which(colnames(data) == "mothids3")] <- NULL
data[which(colnames(data) == "mothids4")] <- NULL
data[which(colnames(data) == "hv3mothids3")] <- NULL
data[which(colnames(data) == "hv4mothids4")] <- NULL
data[which(colnames(data) == "fathids1")] <- NULL
```

Some actual examples of less-than-ideal code I found on the internet

Original Code - Repeated tasks

```
# ORIGINAL
#remove idnumb, other ids
data[which(colnames(data) == "idnumb")] <- NULL
data[which(colnames(data) == "challengeIDs")] <- NULL
data[which(colnames(data) == "mothids1")] <- NULL
data[which(colnames(data) == "mothids2")] <- NULL
data[which(colnames(data) == "mothids3")] <- NULL
data[which(colnames(data) == "mothids4")] <- NULL
data[which(colnames(data) == "hv3mothids3")] <- NULL
data[which(colnames(data) == "hv4mothids4")] <- NULL
data[which(colnames(data) == "fathids1")] <- NULL
```

Alternative Code - Using Built-In Functions

```
# ALTERNATIVE DPLYR
data <- data %>% select (-c(idnumb, challengeIDs, mothids1,
                            mothids2, mothids4, "........"))
# ALTERNATIVE DATA.TABLE
data[, c("idnum", "challengeID", "mothids1",
         "mothids2", "mothid4s", "........") := NULL]
```

Some actual examples of less-than-ideal code I found on the internet

Original: The problem - Calling a function once, on one dataset, using dataset names in the function

```
# ORIGINAL
# Remove cols with > 50% missing data
f_lowInfo <- function(data) {
  ncols = ncol(data)
  nrows = nrow(data)
  to_remove = integer(ncols)
  for (i in 1:ncols) {
    if (sum(is.na(data[i])) > .5*nrows) {
      to_remove[i] = 1
    }
  }
  data[to_remove == 1] <- NULL
  return(data)
}
data <- f_lowInfo(data)
```

Some actual examples of less-than-ideal code I found on the internet

Alternative Code - Define a function, apply it multiple times, make it flexible

```
# ALTERNATIVE
is_col_missing <- function(c.vector){ # create a function to see if a vector is
  return(mean(is.na(c.vector)) > .5)  # over half missing. Use that TRUE = 1, and
}                                     # FALSE = T to our advantage
# Apply across columns of data.frame
data %>% summarize(across(.cols = everything(), is_col_missing)) -> to_remove
data <- data[,to_remove == 0] # remove predominantly empty columns
```

Some actual examples of less-than-ideal code I found on the internet

Alternative Code - Define a function, apply it multiple times, make it flexible

```
# ALTERNATIVE
is_col_missing <- function(c.vector){ # create a function to see if a vector is
  return(mean(is.na(c.vector)) > .5)  # over half missing. Use that TRUE = 1, and
}                                     # FALSE = T to our advantage
# Apply across columns of data.frame
data %>% summarize(across(.cols = everything(), is_col_missing)) -> to_remove
data <- data[,to_remove == 0] # remove predominantly empty columns
```

Solution explained

- This solution capitalizes on the idea that a function should be portable and should probably be used more than once if it really deserves to be a function.
- Capitalizes on the structure of data frames as just groups of same-length vectors.

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter
  - these packages export many of their operations to C++, resulting in significant speed gains

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter
  - these packages export many of their operations to C++, resulting in significant speed gains
- Data.table is often faster than dplyr, but the trade-off is usually insignificant.

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter
  - these packages export many of their operations to C++, resulting in significant speed gains
- Data.table is often faster than dplyr, but the trade-off is usually insignificant.
- Data.table has a more concise but more cryptic syntax, making it seemingly harder to learn and perhaps harder to understand.

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter
  - these packages export many of their operations to C++, resulting in significant speed gains
- Data.table is often faster than dplyr, but the trade-off is usually insignificant.
- Data.table has a more concise but more cryptic syntax, making it seemingly harder to learn and perhaps harder to understand.
- Dplyr is more commonly used in learning applications, data.table is more common in industry applications.

Dplyr and data.table

- When equivalent operations exist in base R and dplyr or data.table, use either of the latter
  - these packages export many of their operations to C++, resulting in significant speed gains
- Data.table is often faster than dplyr, but the trade-off is usually insignificant.
- Data.table has a more concise but more cryptic syntax, making it seemingly harder to learn and perhaps harder to understand.
- Dplyr is more commonly used in learning applications, data.table is more common in industry applications.
- dtplyr uses data.table backend with dplyr syntax!!

Data.table-specific benefits

- 'data tables' are a slightly different kind of object in R

Data.table-specific benefits

- ‘data tables’ are a slightly different kind of object in R
- All data frame syntax works on them, but there’s also data.table specific syntax that you can use to your benefit

Data.table-specific benefits

- ‘data tables’ are a slightly different kind of object in R
- All data frame syntax works on them, but there’s also data.table specific syntax that you can use to your benefit
- Many processes in data.table are more memory efficient

Data.table-specific benefits

- ‘data tables’ are a slightly different kind of object in R
- All data frame syntax works on them, but there’s also data.table specific syntax that you can use to your benefit
- Many processes in data.table are more memory efficient
    - Shallow copies vs. deep copies of data are made when performing certain tasks

Data.table-specific benefits

- ‘data tables’ are a slightly different kind of object in R
- All data frame syntax works on them, but there’s also data.table specific syntax that you can use to your benefit
- Many processes in data.table are more memory efficient
  - Shallow copies vs. deep copies of data are made when performing certain tasks
- Really only necessary if your data is above 1 GB

data.table, dplyr, base R cognates

- We can time each using "system.time()" to evaluate performance!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

data.table, dplyr, base R cognates

- We can time each using "system.time()" to evaluate performance!
- Dplyr and Data.table are leagues faster!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

data.table, dplyr, base R cognates

- We can time each using “system.time()” to evaluate performance!
- Dplyr and Data.table are leagues faster!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

data.table, dplyr, base R cognates

- We can time each using "system.time()" to evaluate performance!
- Dplyr and Data.table are leagues faster!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

```
# Perform the same task using three different methods
system.time({ new_data2$inc_2011_2013 <- new_data2$inc_2011 + new_data2$inc_2013 })  # Base R
```

```
##    user  system elapsed
##   0.344   0.151   0.545
```

data.table, dplyr, base R cognates

- We can time each using "system.time()" to evaluate performance!
- Dplyr and Data.table are leagues faster!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

```
# Perform the same task using three different methods
system.time({ new_data2$inc_2011_2013 <- new_data2$inc_2011 + new_data2$inc_2013 })  # Base R
```

```
##    user  system elapsed
##   0.344   0.151   0.545
```

```
system.time({ dplyr::mutate(new_data2, inc_2011_2013 = inc_2011 + inc_2013) }) # Dplyr
```

```
##    user  system elapsed
##   0.054   0.001   0.055
```

data.table, dplyr, base R cognates

- We can time each using “system.time()” to evaluate performance!
- Dplyr and Data.table are leagues faster!

```
replicate(new_data, n = 5000, simplify = F) %>% rbindlist() -> new_data2
new_data[1:2]
```

```
##    PUBID sex race inc_2011 inc_2013 inc_2015 inc_2017 inc_2019
## 1:     2   1    2    81000    83000    98928   116000   128400
## 2:     4   2    2    51000    29000    45000    45000    27000
```

```
dim(new_data)
```

```
## [1] 2696    8
```

```
# Perform the same task using three different methods
system.time({ new_data2$inc_2011_2013 <- new_data2$inc_2011 + new_data2$inc_2013 })  # Base R
```

```
##    user  system elapsed
##   0.344   0.151   0.545
```

```
system.time({ dplyr::mutate(new_data2, inc_2011_2013 = inc_2011 + inc_2013) }) # Dplyr
```

```
##    user  system elapsed
##   0.054   0.001   0.055
```

```
system.time({ new_data2[, inc_2011_2013 := inc_2011 + inc_2013] }) #Data.table
```

```
##    user  system elapsed
##   0.038   0.000   0.039
```

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs
- (Uses built-in parallelization!)

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs
- (Uses built-in parallelization!)
- lag/lead for shifting components with temporal data

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs
- (Uses built-in parallelization!)
- lag/lead for shifting components with temporal data
- frollmean/rollmean/(f)rollsum, etc. for computing rolling window averages/sums

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs
- (Uses built-in parallelization!)
- lag/lead for shifting components with temporal data
- frollmean/rollmean/(f)rollsum, etc. for computing rolling window averages/sums
- dcast/melt in data.table pivot_wider/pivot_longer for efficient reshaping (to be covered later)

data.table/tidyverse tricks

- vroom::vroom()/data.table::fread()/::fwrite() for reading/writing csvs
- (Uses built-in parallelization!)
- lag/lead for shifting components with temporal data
- frollmean/rollmean/(f)rollsum, etc. for computing rolling window averages/sums
- dcast/melt in data.table pivot_wider/pivot_longer for efficient reshaping (to be covered later)
- Don't use the $ operator ever with data.frames/data.tables!

Questions?

Data Management and Reshaping Efficiently

Data Storage Tricks

Be cognizant of what kinds of storage your data frame uses

- Long strings are memory intensive.

```r
object.size("Hello") # a character string containing "Hello"
```

```
## 112 bytes
```

```r
object.size(paste0(rep("Hello", 20), collapse = " ")) # Hello Repeated 20 times
```

```
## 232 bytes
```

```r
object.size(numeric(1000)) # Float64
```

```
## 8048 bytes
```

```r
object.size(integer(1000)) # Integers take up less space
```

```
## 4048 bytes
```

Data Storage Tricks

Be cognizant of what kinds of storage your data frame uses

- Long strings are memory intensive.
- Lower-precision numerical types save space!

```
object.size("Hello") # a character string containing "Hello"
```

```
## 112 bytes
```

```
object.size(paste0(rep("Hello", 20), collapse = " ")) # Hello Repeated 20 times
```

```
## 232 bytes
```

```
object.size(numeric(1000)) # Float64
```

```
## 8048 bytes
```

```
object.size(integer(1000)) # Integers take up less space
```

```
## 4048 bytes
```

Efficient reshaping

- data.table was first to the scene with optimized versions of reshape2's functions

Efficient reshaping

- data.table was first to the scene with optimized versions of reshape2's functions
- data.table::melt() and data.table::dcast()

Efficient reshaping

- data.table was first to the scene with optimized versions of reshape2's functions
- data.table::melt() and data.table::dcast()
- tidyr::pivot_longer was inspired by data.table's melt(), and usually performs faster than base R, but I think data.table's reshpaing tools are generally faster.

Data storage considerations

Wide data is often more space efficient, up to a point.

```
df <- data.frame(Hunter = c(1,2,3),
                 Wade = c(2,4,6),
                 York = c(3,6,9),
                 multiplier = c(1,2,3))
```

Data storage considerations

Wide data is often more space efficient, up to a point.

```
df <- data.frame(Hunter = c(1,2,3),
                 Wade = c(2,4,6),
                 York = c(3,6,9),
                 multiplier = c(1,2,3))
```

```
head(df) #wide
```

```
##   Hunter Wade York multiplier
## 1      1    2    3          1
## 2      2    4    6          2
## 3      3    6    9          3
```

Data storage considerations

Wide data is often more space efficient, up to a point.

```
df <- data.frame(Hunter = c(1,2,3),
                 Wade = c(2,4,6),
                 York = c(3,6,9),
                 multiplier = c(1,2,3))
```

```
head(df) #wide
```

```
##   Hunter Wade York multiplier
## 1      1    2    3          1
## 2      2    4    6          2
## 3      3    6    9          3
```

```
df_long <- melt(df, id.vars = "multiplier")
```

Data storage considerations

Wide data is often more space efficient, up to a point.

```
df <- data.frame(Hunter = c(1,2,3),
                 Wade = c(2,4,6),
                 York = c(3,6,9),
                 multiplier = c(1,2,3))
```

```
head(df) #wide
```

```
##   Hunter Wade York multiplier
## 1      1    2    3          1
## 2      2    4    6          2
## 3      3    6    9          3
```

```
df_long <- melt(df, id.vars = "multiplier")
```

```
head(df_long) #long
```

```
##   multiplier variable value
## 1          1   Hunter     1
## 2          2   Hunter     2
## 3          3   Hunter     3
## 4          1     Wade     2
## 5          2     Wade     4
## 6          3     Wade     6
```

Data storage considerations

Wide data is often more space efficient, up to a point.

```
df <- data.frame(Hunter = c(1,2,3),
                 Wade = c(2,4,6),
                 York = c(3,6,9),
                 multiplier = c(1,2,3))
```

```
head(df) #wide
```

```
##   Hunter Wade York multiplier
## 1      1    2    3          1
## 2      2    4    6          2
## 3      3    6    9          3
```

```
df_long <- melt(df, id.vars = "multiplier")
```

```
head(df_long) #long
```

```
##   multiplier variable value
## 1          1   Hunter     1
## 2          2   Hunter     2
## 3          3   Hunter     3
## 4          1     Wade     2
## 5          2     Wade     4
## 6          3     Wade     6
```

```
object.size(df)
```

```
## 1224 bytes
```

```
object.size(df_long)
```

```
## 1888 bytes
```

Questions?

Parallelization

Parallelization

- A related concept to vectorization is parallelization. Just as you can take an operation and apply it to an entire array instead of each item of an array, you can have multiple chunks of code run simultaneously if your machine is capable of this.

,

Parallelization

- My computer has “1.4GHz quad-core Intel Core i5, Turbo Boost up to 3.9GHz, with 128MB of eDRAM”

```
parallel::detectCores()
```

```
## [1] 8
```

Parallelization

- My computer has "1.4GHz quad-core Intel Core i5, Turbo Boost up to 3.9GHz, with 128MB of eDRAM"
- Parallelization using R is actually a little funky, and so it will allow parameters that shouldn't work, and language gets slippery (cores vs. threads).

```r
parallel::detectCores()
```

```
## [1] 8
```

Parallelization

- My computer has "1.4GHz quad-core Intel Core i5, Turbo Boost up to 3.9GHz, with 128MB of eDRAM"
- Parallelization using R is actually a little funky, and so it will allow parameters that shouldn't work, and language gets slippery (cores vs. threads).
- Disclaimer: I don't understand computers! With parallelization, profile to make sure your code is actually faster, because confusing things happen.

```r
parallel::detectCores()
```

```
## [1] 8
```

Parallelization

Define your function

```
modr <- function(c.dat, c.vars, c.outcome){
  form <- as.formula(
    paste0(c.outcome, " ~ (1|",
           paste0(c.vars, collapse = ") + (1|"),
           ")")
  )
  mod <- lme4::lmer(form, data = c.dat)
  mod_betas <- data.table(summary(mod)$coefficients[, 1:2], keep.rownames = T)
  mod_betas[, ind_vars := paste0(c.vars, collapse = ", ")]
  return(mod_betas)
}
```

This is a computationally expensive function. I've changed the model to a random effects model to use more computation power.

Get a longer vector of variables to iterate over.

```
vars_vec <- replicate(20, expr =
                        sample(regress_vars[regress_vars != "inc_2019"], 3), simplify = F)
```

Parallelization

Serial - 1 Iteration

```
modr(vars_vec[[1]], c.dat = new_data, c.outcome = "inc_2019")
```

Time for this code chunk to run: 1.38783097267151

Serial - 20 Iterations

```
lapply(vars_vec, modr, c.outcome = "inc_2019", c.dat = new_data) %>%
  rbindlist() -> all_betas
```

Time for this code chunk to run: 3.05472922325134

Parallel - 20 Iterations

```
parallel::mclapply(vars_vec, modr, c.outcome = "inc_2019",
                   c.dat = new_data, mc.cores = 4) %>%
  rbindlist() -> all_betas
```

Time for this code chunk to run: 1.31190490722656
There is overhead associated with despatching and receiving each task, resulting in a tradeoff that you have to manage!

Parallelization

- The backends of several packages in R will automatically parallelize.

Parallelization

- The backends of several packages in R will automatically parallelize.
  - Model-fitting, loading data, etc.

Parallelization

- The backends of several packages in R will automatically parallelize.
  - Model-fitting, loading data, etc.
- This is a handy tool to learn if you need to use cluster-based supercomputing since there are similar concepts.

Questions, comments?

Thank you!

- I can be reached at hyork@princeton.edu

Thank you!

- I can be reached at hyork@princeton.edu
- Check out my website hunterwyork.com

Thank you!

- I can be reached at hyork@princeton.edu
- Check out my website hunterwyork.com
- If you found this presentation useful, share it!

Thank you!

- I can be reached at hyork@princeton.edu
- Check out my website hunterwyork.com
- If you found this presentation useful, share it!
- Rmarkdown file and slides are here.