Good Enough Practices in Computational Analysis

Gennaro Calendo

Table of contents

0.1 Data mismanagement

- 2005: Kodak overstated *an* employee's severance by \$11,000,000

    - Reason: Someone added too many zeroes in an Excel spreadsheet
- 2008: Barclay's purchased 179 contracts from a bankrupt Lehman Brothers
    - Reason: 179 rows were hidden instead of deleted
- 2010: Harvard economists miscalculate effect of high public debt on economic growth, influencing political discourse after the 2008 financial crisis
    - Reason: Omission of 5 key countries' data - skipping rows in a calculation
- 2011: M15 wiretaps 134 unsuspecting citizens
    - Reason: An Excel entry error resulted in numbers ending with "000" being tapped instead of the desired numbers.
- 2020: Public Health England drops 16,000 positive COVID tests
    - Reason: A large csv file was imported into Excel that exceeded 1,048,576 records (the physical limit of records per sheet). These records were silently dropped.

0.2 Why care about data management?

- Computing is now an essential part of research
- Most researchers are never taught good computational practices
- Improve reproducibility
- Ensure correctness
- Increase efficiency

For the remainder, I am assuming that you have already planned the experiment, collected data and metadata, and have some kind of analysis to do.

0.3 Topics

1. Data management
2. Project organization
3. Tracking changes
4. Software
5. Manuscripts

1 Data Management

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, et al. (2017) Good enough practices in scientific computing. PLOS Computational Biology 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

1.1 Save and lock all raw data

- Keep raw data in its unedited form
  - This includes filenames!
- Consider changing file permissions to make the file immutable

```
chattr +i myfile.txt
```

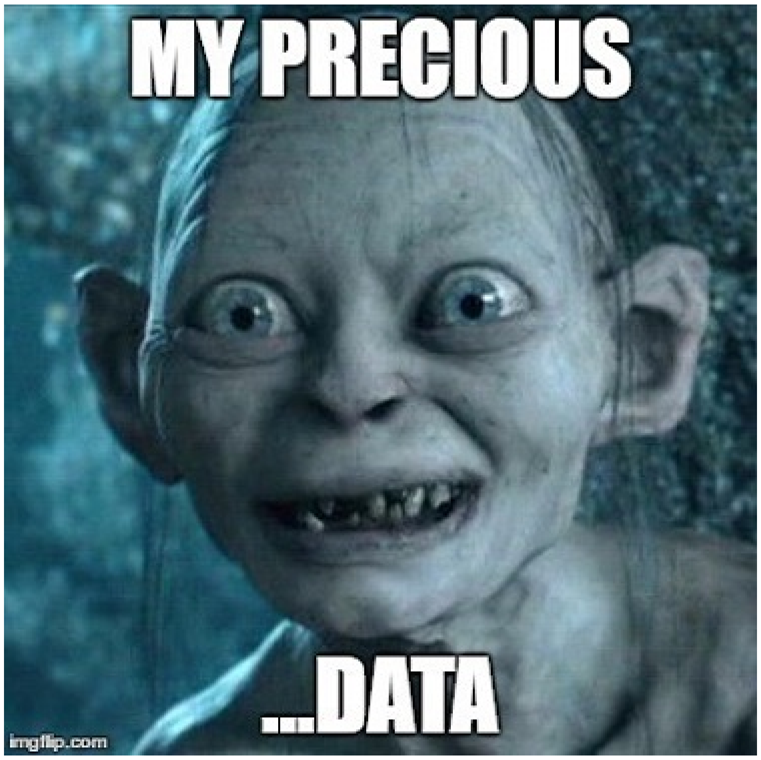
- If using Excel, lock the spreadsheet
- Avoid making copies of large local files or persistent databases

Use soft links

```
# Link all fastq files to a local directory
for FILE in $(find /path/to/fq/files -name "*.fq.gz"); do
  ln -s $F /path/to/local/directory/$(basename $F)
done
```

or use URLs directly

```
# Avoid downloading a large GTF file - reads GTF directly into memory
url <- "https://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_44/gencode.v44.ann
gtf <- rtracklayer::import(url)
```


MY PRECIOUS
...DATA
imgflip.com

1.2 Backups

> There are two types of people, those who do backups and those who will do backups.

. . .

The following are NOT backup solutions:

- Copies of data on the same disk
- Dropbox/Google Drive
- RAID arrays

. . .

All of these solutions mirror the data. Corruption or ransomware will propagate.

. . .

1.2.1 Use the 3-2-1 rule:

- Keep 3 copies of any important file: 1 primary and 2 backups.
- Keep the files on 2 different media types to protect against different types of hazards.
- Store 1 copy offsite (e.g., outside your home or business facility).

A backup is only a backup if you can restore the files!

1.3 How Toy Story 2 almost got deleted

https://youtu.be/8dhp_20j0Ys?feature=shared

1.4 Files

- Store data in standard, machine readable formats
  - TXT, CSV, TSV, JSON, YAML, HDF5
  - Large text files should be compressed
    - `xz` is better than `gzip`. We should all use `xz`
- Ensure filenames are computer friendly
  - No spaces
  - Use only letters, numbers, and "-" "_" and "." as delimiters

```
# Bad
data for jozef.txt

# Okay
data-for-jozef.txt

# Best
2023-11-09_repetitive-element-counts.txt
```

1.5 Quiz

I have a directory with the following files:

```
a.txt
b.txt
c.txt
file with spaces in the name.txt
some other file.txt
```

What does the following code return? (Expected: For each file in the directory print the filename)

```
for FILE in $(ls); do echo $FILE; done
```

. . .

Bash interprets every space as a new word!

```
a.txt
b.txt
c.txt
file
with
spaces
in
the
name.txt
some
other
file.txt
```

1.5.1 Pro-tip

A simple alternative here is to use a text document with the basenames the files you want to loop over and then loop over the lines of the file instead.

```
SAMPLES=sample-names.txt

for SAMPLE in $(cat $SAMPLES); do
  doStuff ${SAMPLE}.txt;
done
```

1.6 More on filenames

- Anticipate the need to use multiple tables
    - **Use consistent and unique identifiers across all files**
    - Unique random IDs are suited best for large projects
    - For small projects, unique combinations of sample information can be used. Just ensure that *every* sample ID follows the same pattern
- The same filename rules apply for the names of scripts
    - Ensure that the name of the script reflects the function it performs

. . .

Bad

```
subject1-control_at_96hr1.txt
s1_ctl-at-4days_2.txt
s2TRT4d1.txt
sbj2_Treatment_4_Days_Replicate_2.txt
```

Good

```
subject1_control_4days_rep1.txt
subject1_control_4days_rep2.txt
subject2_treatment_4days_rep1.txt
subject2_treatment_4days_rep2.txt
```

1.6.1 Pro-tip

One easy way to create unique random IDs is to concatenate descriptions and take the SHA hash

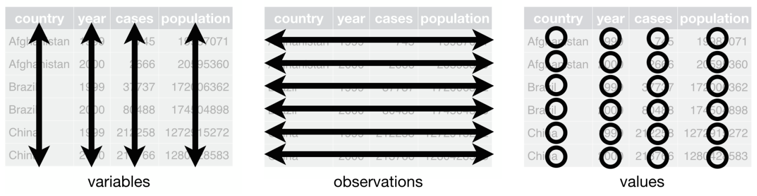
```
echo "subject1_control_4days_rep1" | sha256
# 57f458a294542b2ed6ac14ca64d3c8e4599eed7a

echo "subject1_control_4days_rep2" | shasum
# b6ea9d729e57cce68b37de390d56c542bc17dea6
```

1.7 Create analysis freindly data - tidy data

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.

    a) Don't cram two variables into one value. e.g. "male_treated".
    b) Don't embed units into data. e.g. "3.4kg". Instead, put in column name e.g. "weight_kg" = 3.4



1.7.1 Pro-tip

`data.frame` like objects can be stored and retrieved efficiently using the Apache Arrow format instead of CSV files:

```
df <- arrow::read_parquet("path/to/file.parquet")
arrow::write_parquet(df, "path/to/different_file.parquet")
```

see R arrow for more details

1.8 Untidy(?) data

- Some data formats are not amenable to the 'tidy' structure
  - Large/sparse matrices, geo-spatial data, R objects, etc.
  - Store data in the format that is most appropriate for the data
- Don't convert a matrix to a long format and save as a tsv file!
  - Save as an `.rds` file instead. The fst package provides a fast, multi-threaded modern alternative to `.rds` format
  - Large matrices can also be efficiently stored as HDF5 backed SummarizedExperiments
- If you are accessing the same data often, consider storing as a SQLite database and accessing with dbplyr or sqlalchemy in Python

1.9 Record all steps used to generate the data

- Write scripts for *every* stage of the data processing
- For data that you download from the internet, include a README text file with the exact `curl` or `wget` commands used.

For example:

```
Transcripts:
wget http://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_38/gencode.v38.transcr

Primary Assembly:
wget http://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_38/GRCh38.primary_asse

Create concatenated transcripts + genome for salmon (i.e. gentrome):
cat gencode.v38.transcripts.fa.gz GRCh38.primary_assembly.genome.fa.gz > gentrome.fa.gz

Create decoys file for salmon:
grep ">" <(gunzip -c GRCh38.primary_assembly.genome.fa.gz) | cut -d " " -f 1 > decoys.txt
sed -i.bak -e 's/>//g' decoys.txt
```

For more complicated steps include a script. e.g. creating a new genome index, subsetting BAM files, accessing data from NCBI, etc.

1.9.1 Pro-tip

Number scripts in the order that they should be executed:

```
01_download.sh
02_process.py
03_makeFigures.R
```

You can also include a runner script that will execute all of the above. Or, for more consistent workflows, use a workflow manager like Nextflow, Snakemake, WDL, or good ole' GNU Make

2 Project Organization

2.1 Look familiar?

2.2 Project structure

- Use a consistent and logical directory structure across all projects. See these papers for more examples, 1, 2
- Give each project a descriptive name and append the date the project was started
    - e.g. `brca_rnaseq_2023-11-09/` **not** `rnaseq_data/`

My personal structure for every project looks like:

- a README text file is present at the top level of the directory with a short description about the project and any notes or updates
- `data/` should contain soft links to any raw data or the results of downloading data from an external source
- `doc/` contains metadata documents about the samples or other metadata information about the experiment

- `results/` contains only data generated within the project. It has sub-directories for `figures/`, `data-files/` and `rds-files/`. If you have a longer or more complicated analysis then add sub-directories indicating which script generated the results.
- `scripts/` contains all analysis scripts numbered in their order of execution. Synchronize the script names with the results they produce.

2.3 Project structure

A fuller example might look more like:

2.3.1 Pro-tip

If using Rstudio, include an `.Rproj` file at the top level of your directory. Doing this enables you to use the here package to reference data within your project in a relative fashion. For example, you can more easily save data with:

```
plot_volcano(...)
ggsave(here("data", "results", "figures", "04", "volcano-plots.png"))
```

3 Tracking Changes

3.1 Manual version control

- Always track any changes made to your project over the entire life of the project
- This can be done either manually or using a dedicated version control system
- If doing this manually, add a file called "CHANGELOG.md" in your docs/ directory and add detailed notes in reverse chronological order

For example:

```
## 2016-04-08

* Switched to cubic interpolation as default.
* Moved question about family's TB history to end of questionnaire.

## 2016-04-06

* Added option for cubic interpolation.
* Removed question about staph exposure (can be inferred from blood test results).
```

- If you make a significant change to the project, copy the whole directory, date it, and store it such that it will no longer be modified.

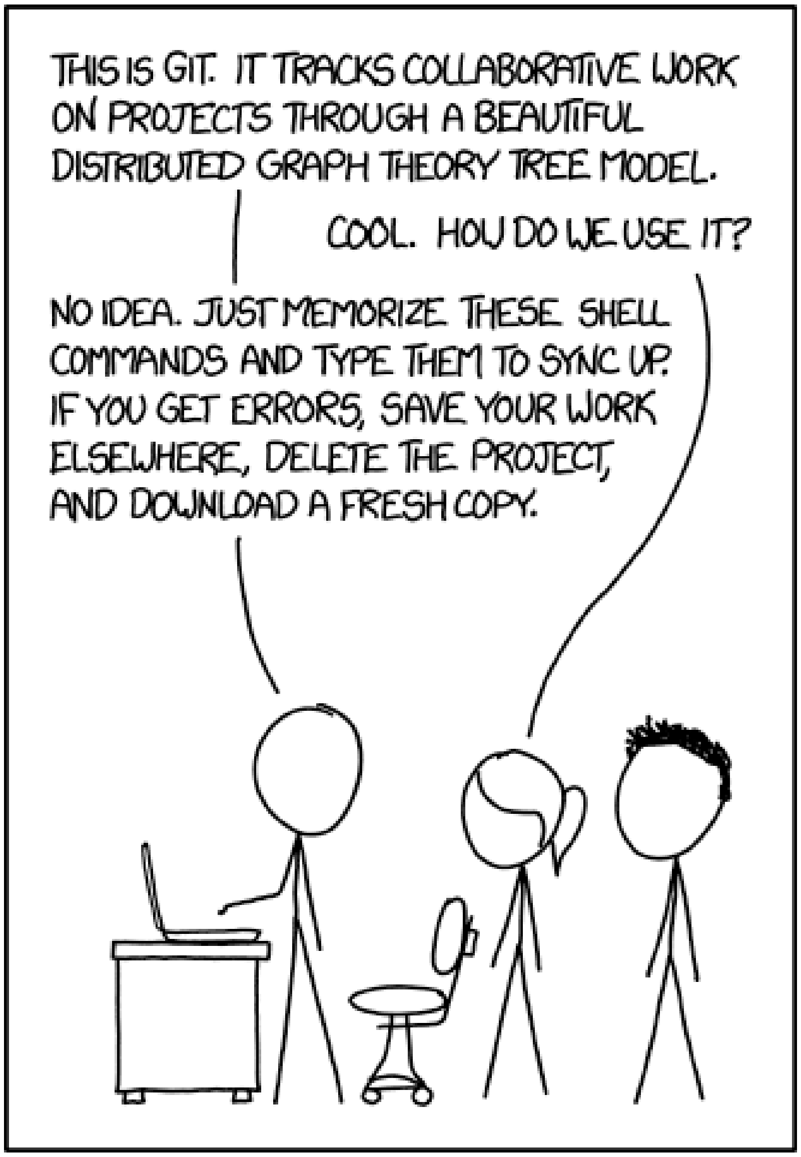
3.1.1 Pro-tip

Copies of old projects can be compressed and saved with tar + xz compression

```
tar -cJvf old.20231109_myproject.tar.xz myproject/`
```

3.2 Version control with git

- Version control systems allow you to track all changes, comment on why changes were made, create parallel branches, and merge existing ones
- Microsoft Office files and PDFs can be stored with Github but it's hard to track changes. Rely on Microsoft's "Track Changes" instead and save frequently
- It's not necessary to version control raw data (back it up!) since it shouldn't change. Likewise, backup intermediate data and version control the scripts that made it.
- For a quick primer on Git and GitHub check out the book Happy Git with R or The Official GitHub Training Manual
- Anyone in the lab can join the coriell-research organization on Github and start tracking their code
- Be careful committing sensitive information to GitHub


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

4 Software

4.1 Quick tips to improve your scripts

4.1.1 Place a description at the top of every script

- The description should indicate who the author is
- When the code was created
- A short description of what the expected inputs and outputs are along with how to use the code
- You three months from now will appreciate it when you need to revisit your analysis

For example:

```
#!/usr/bin/env python3
# Gennaro Calendo
# 2023-11-09
#
# This scripts performs background correction of all images in the
#  user supplied directory
#
# Usage ./correct-bg.py --input images/ --out_dir out_directory
#
from image_correction import background_correct

for img in images:
  img = background_correct(img)
  save_image(img, "out_directory/corrected.png")
```

4.2 Quick tips to improve your scripts

4.2.1 Decompose programs into functions

- Functions make it easier to reason about your code, spot errors, and make changes.
- This also follows the Don't Repeat Yourself principle aimed at reducing repetition by replacing it with abstractions that are more stable

Compare this chunk of code that rescales values using a min-max function (0-1)

```r
df <- tibble::tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)

df$a <- (df$a - min(df$a, na.rm = TRUE)) /
  (max(df$a, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$b <- (df$b - min(df$b, na.rm = TRUE)) /
  (max(df$b, na.rm = TRUE) - min(df$a, na.rm = TRUE))
df$c <- (df$c - min(df$c, na.rm = TRUE)) /
  (max(df$c, na.rm = TRUE) - min(df$c, na.rm = TRUE))
df$d <- (df$d - min(df$d, na.rm = TRUE)) /
  (max(df$d, na.rm = TRUE) - min(df$d, na.rm = TRUE))
```

to this function which does the same thing

```r
rescale01 <- function(x) {
  rng <- range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}

df <- lapply(df, rescale01)
```

Which is easier to read? Which is easier to debug? Which is more efficient?

4.3 Quick tips to improve your scripts

4.3.1 Give functions and variables meaningful names

- Programs are written for people and then computers
- Use variable and function names that are meaningful and **correct**
- Keep names consistent. Use either `snake_case` or `camelCase` but try not to mix both

Bad:

```r
lol <- 1:100

mydata <- data.frame(x = c("Jozef", "Gennaro", "Matt", "Morgan", "Anthony"))

f <- function(x, y, ...) {
  plot(x = x, y = y, main = "Scatter plot of x and y", ...)
}
```

Better:

```r
ages <- 1:100

bioinfo_names <- data.frame(Name = c("Jozef", "Gennaro", "Matt", "Morgan", "Anthony"))

plotScatterPlot <- function(x, y, ...) {
  plot(x = x, y = y, main = "Scatter plot of x and y", ...)
}
```

4.4 Quick tips to improve your scripts

4.4.1 Do not control program flow with comments

- this is error prone and makes it difficult or impossible to automate
- Use if/else statements instead

Bad:

```r
# Download the file
#filename <- "data.tsv"
#url <- "http://example.com/data.tsv"
#download.file(url, filename)

# Read in to a data.frame
df <- read.delim("data.tsv", sep="\t")
```

Good:

```r
filename <- "data.tsv"
url <- "http://example.com/data.tsv"

if (!file.exists(filename)) {
  download.file(url, filename)
}
df <- read.delim(filename)
```

4.5 Quick tips to improve your scripts

4.5.1 Use a consistent style

- Pick a style guide and stick with it
- If using R, the styler package can automatically cleanup poorly formatted code
- If using Python, black is a highly opinionated formatter
- **Don't use right hand assignment**

Bad:

```r
flights|>group_by(dest)|> summarize(
distance=mean( distance),speed = mean(distance/air_time, na.rm= T)) |>
ggplot(aes(x= distance, y=speed))+geom_smooth(method = "loess",span = 0.5,se = FALSE,color =
```

Good:

```r
flight_plot <- flights |>
  group_by(dest) |>
  summarize(
    distance = mean(distance),
    speed = mean(distance / air_time, na.rm = TRUE)
  ) |>
  ggplot(aes(x = distance, y = speed)) +
  geom_smooth(
    method = "loess",
    span = 0.5,
    se = FALSE,
    color = "white",
    linewidth = 4
  ) +
  geom_point()
```

5 Manuscripts

5.1 A typical workflow

> A common practice in academic writing is for the lead author to send successive versions of a manuscript to coauthors to collect feedback, which is returned as changes to the document, comments on the document, plain text in email, or a mix of all 3. This allows coauthors to use familiar tools but results in a lot of files to keep track of and a lot of tedious manual labor to merge comments to create the next master version.

. . .

Instead of an email based workflow, we should aim to mirror best practices developed for data management and software development

We want to:

- Ensure text is accessible to yourself and others now and in the future by making a single accessible master document
- Make it easy to track and combine changes from multiple authors
- Avoid duplication and manual entry of references, figures, tables, etc.
- Make it easy to regenerate the final, most updated version and share with collaborators and submit
- **More important than the type of workflow is getting everyone to agree on the workflow chosen and how changes will be tracked**

Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, et al. (2017) Good enough practices in scientific computing. PLOS Computational Biology 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

5.2 Workflow 1: single online master document

- Keep a single online master document that all authors can collaborate on
- The tool should be able to track changes, automatically manage references, track version changes.
- Two candidates are Google Docs and Microsoft Office online
  - Here's a good blog post about using Google Docs for scientific writing
- Of course, some collaborators will insist on using familiar desktop based tools which will require saving these files in a versioned doc/ directory and manually merging the changes
  - Pandoc can be useful for these kinds of document merging

5.3 Workflow 2: Text documents under version control

- Write papers in a text based format like LaTeX or markdown managing references in a .bib file and tracking changes via `git`
- Then convert these plain text documents to a pdf or Word doc for final submission with Pandoc
- Mathematics, physics, and astronomy have been doing it this way for decades
- This of course requires everyone to learn a typsetting language and be able to use version control tools…
  - What is the difference between plain text and word processing?
  - Text editors?
  - What does it mean to compile a document?
  - BIBTex?
  - Git/ Github?

this is all probably a bit too much

6 Summary

6.1 Summary

- Data management
  - Save all raw data and don't modify it
  - Keep good backups and make sure they work. 3-2-1 rule
  - Use consistent, meaningful filenames that make sense to computers and reflect their content or function
  - Create analysis friendly data
  - Work with/ save data in the format that it is best suited to
- Project Organization
  - Use a consistent, well-defined project structure across all projects
  - Give each project a consistent and meaningful name
  - Use the structure of the project to organize where files go
- Tracking Changes
  - Keep changes small and save changes frequently
  - If manually tracking changes do so in a logical location in a plain text document
  - Use a version control system
- Software
  - Write a short description at the top of every script about what the script does and how to use it
  - Decompose programs into functions. Don't Repeat Yourself
  - Give functions and variables meaningful names
  - Use statements for control flow instead of comments
  - Use a consistent style of coding. Use a code styler
- Manuscripts
  - Pick a workflow that everyone agrees on and keep a single, active collaborative document using either Microsoft Online or Google Docs

7 Resources

7.1 Resources

7.1.1 Best practices

- This presentation: https://github.com/coriell-research/best-practices
- Good Enough Practices in Scientific Computing: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1005510#sec004
- Backups: https://www.cisa.gov/sites/default/files/publications/data_backup_options.pdf
- File naming: https://library.si.edu/sites/default/files/filenamingorganizing_20210609.pdf
- Tidy data: https://www.jstatsoft.org/article/view/v059i10
- Frank Harrell's R Workflow for Reproducible Analysis: https://hbiostat.org/rflow/
  - FH blog is also really good for statistical concepts: https://www.fharrell.com/

7.1.2 Bioinformatics

- Bioinformatics Data Skills: https://www.oreilly.com/library/view/bioinformatics-data-skills/9781449367480/
- BioStars for bioinfo questions: https://www.biostars.org/
- Bioconductor common workflows: https://bioconductor.org/packages/release/BiocViews.html#____Workfl

7.1.3 Proficiency with computational tools

- MIT Missing Semester: https://missing.csail.mit.edu/
  - Really, check this one out

7.1.4 R

- R for Data Science: https://r4ds.hadley.nz/
- Advanced R: https://adv-r.hadley.nz/
- fasteR (base): https://github.com/matloff/fasteR
- Efficient R Programming: https://bookdown.org/csgillespie/efficientR/
- R performance tips: https://peerj.com/preprints/26605.pdf

- R Inferno: https://www.burns-stat.com/documents/books/the-r-inferno/
- Introduction to data science: https://rafalab.github.io/dsbook-part-1/
- Advanced data science: https://rafalab.github.io/dsbook-part-2/