Lecture 16 – Documenting your work for reproducibility

Learning Objectives:

4. **Produce code that is reproducible and produces results that are replicable.**

Midterm survey is on Canvas!

Having reviewed code, what should we be assessing?

Please submit comments to the #questions thread on Slack.

Why Document Your Code?

For yourself (in 6 months)

- You will not remember what you did or how you did it.
- You will spend lots of time trying to figure out what on earth you were doing back then.
- You will wonder what on earth you were thinking.

Be sure to Gordon Ramsay your own code before someone else does it.

For others

- Others will spend copious amounts of time trying to deconstruct your logic.
- They will repeat effort trying to understand, replicate, or rewrite your code.
- They will be very frustrated with you.

For science!

- Easier to understand code and easier to read code will help people understand what you did.
- People will find mistakes quicker, which is a GOOD thing!
- More open, closer to a more accurate understanding of the world.

What should you document? What does this mean?

The main goal is for people to understand your vision for the code!
(Or at least, what you think your code is doing.)

This means they should understand:

- What data the code takes in
- What output the code returns
- What manipulations take place within the code

Documenting your code is just like writing a paper in this respect – you have to communicate what you're thinking and what you've done, which is sometimes not as easy as it seems.

The first step is acknowledging and avoiding ***The Curse of Knowledge…***

The False-Belief Test in Children

M&M’s

The False-Belief Test in Children

~~M&M~~'s

(Ribbons)

The False-Belief Test in Children

~~M&M~~'s

(Ribbons)

The False-Belief Test in Children

~~M&M's~~

(Ribbons)

Ribbons

False belief
or “Curse of Knowledge”

Tips for battling the Curse of Knowledge

- **Stick with best-practices organization.** This helps demystify what scripts are doing what for someone unfamiliar with your code.

- **Write about what blocks of code are doing, not single lines.** Knowing what single lines are doing is not as helpful as an overall picture. Comment specific lines if the syntax makes it difficult for someone to follow the code.

- **Put it down for awhile (a week or two) and come back to it.** This will give you enough distance to take on a naïve mindset, but not enough to forget what you've done entirely.

- **Refactor.** Try to reorganize code in functional units (separate functions). This will help you organize your thoughts (and make it easier to write about).

- **Make a flow chart.** You can get more info in a diagram than text!

- **Peer code review.** Have someone else look at your work and help you point out places that are unclear!

Best Practices for Documenting Work

- **Standard organization.** Separate into folders:
    - src (code)
    - bin (binary files/executables)
    - doc (documentation)
    - data (data files that are not results, needed to generate results)
    - results (files, graphs, etc that are produced as results)

- **Include a README file.**

- **Consider a workflow diagram.** Especially for complicated workflows, this will help.

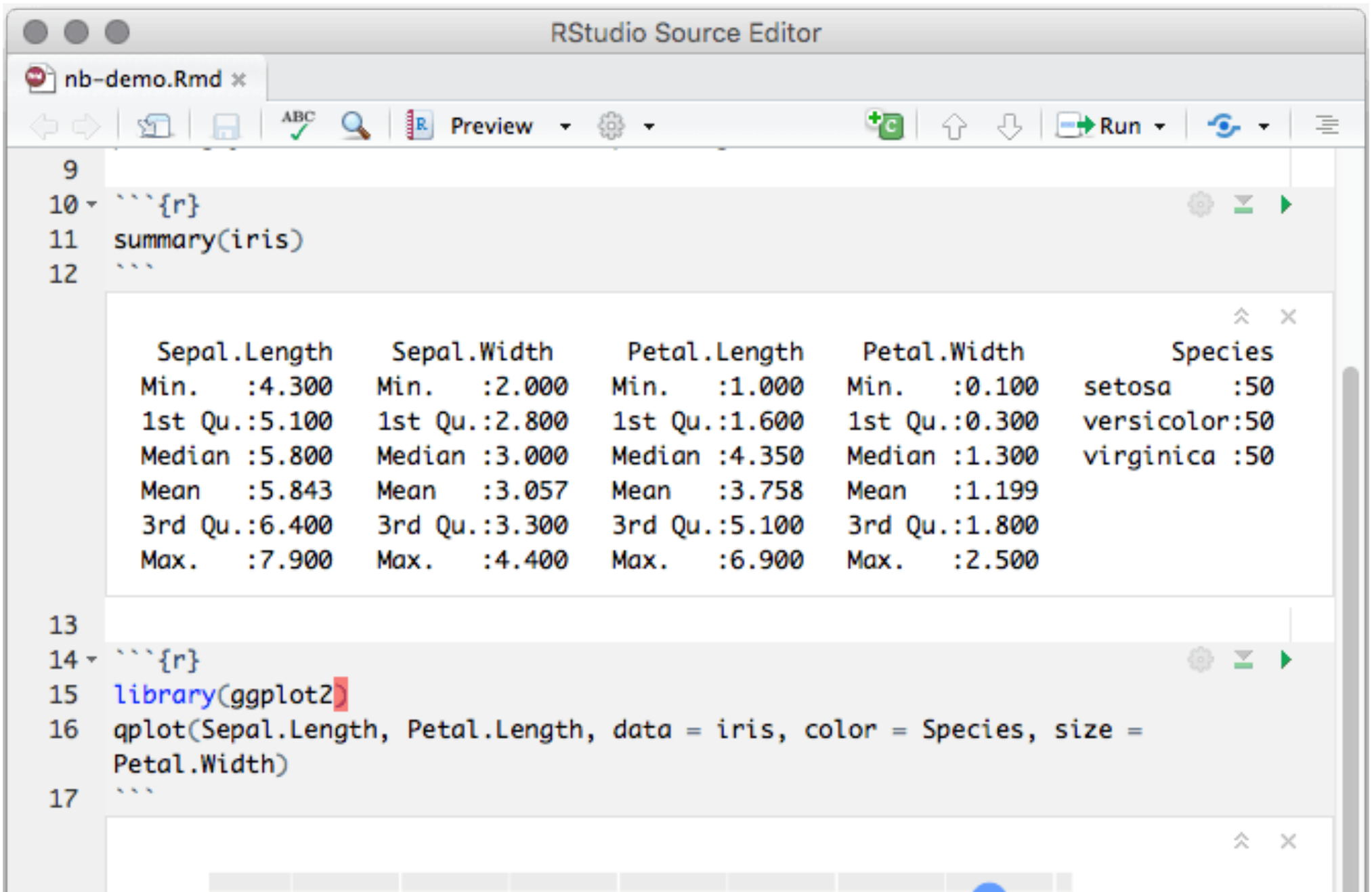
- **Include sufficient instructions to reproduce results.** Ideally, all the steps need to reproduce final results.

- **Consider a markdown document with embedded code.** Embedding code and results within a human-readable document forces things to be clearer! “Literate programming”

Tools that will help: R Markdown

https://rmarkdown.rstudio.com/

https://bookdown.org/yihui/rmarkdown/notebook.html

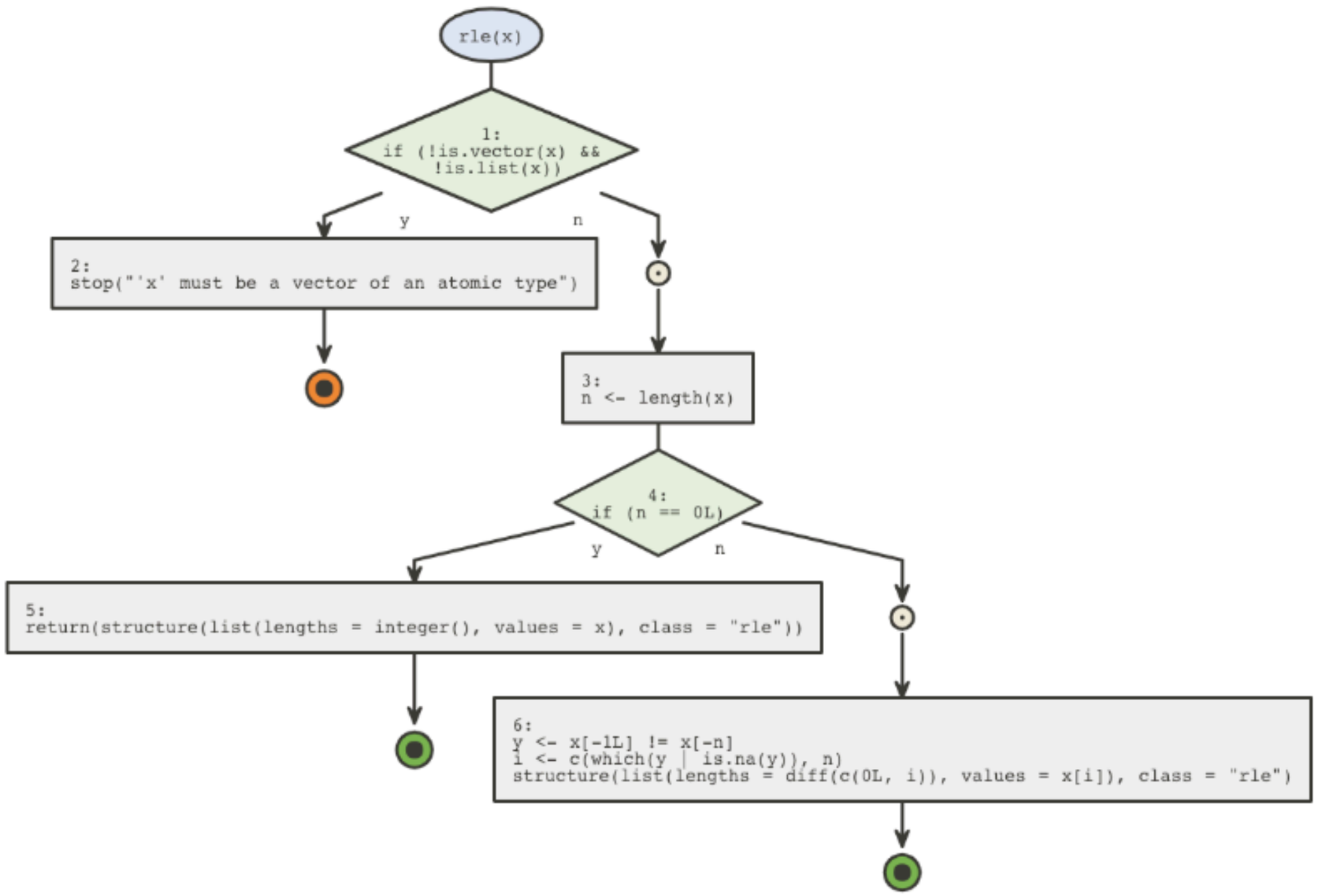


Tools that will help: `flow` package

To install: `remotes::install_github("moodymudskipper/flow")`

Draws flow charts of functions. Useful for a deep dive into analysis pipelines.

https://moodymudskipper.github.io/flow/articles/Draw-a-function.html



Tools that will help: `workflowr` package

https://github.com/jdblischak/workflowr

https://jdblischak.github.io/workflowr/articles/wflow-01-getting-started.html

- Provides project template with organized subdirectories
- Aids in reproducibility (various actions)
- Creates website to present results, links to past versions of results

`wflow_start()` created the following directory structure in `myproject/`:

```
myproject/
├── .gitignore
├── .Rprofile
├── _workflowr.yml
├── analysis/
│   ├── about.Rmd
│   ├── index.Rmd
│   ├── license.Rmd
│   └── _site.yml
├── code/
│   ├── README.md
├── data/
│   └── README.md
├── docs/
├── myproject.Rproj
├── output/
```

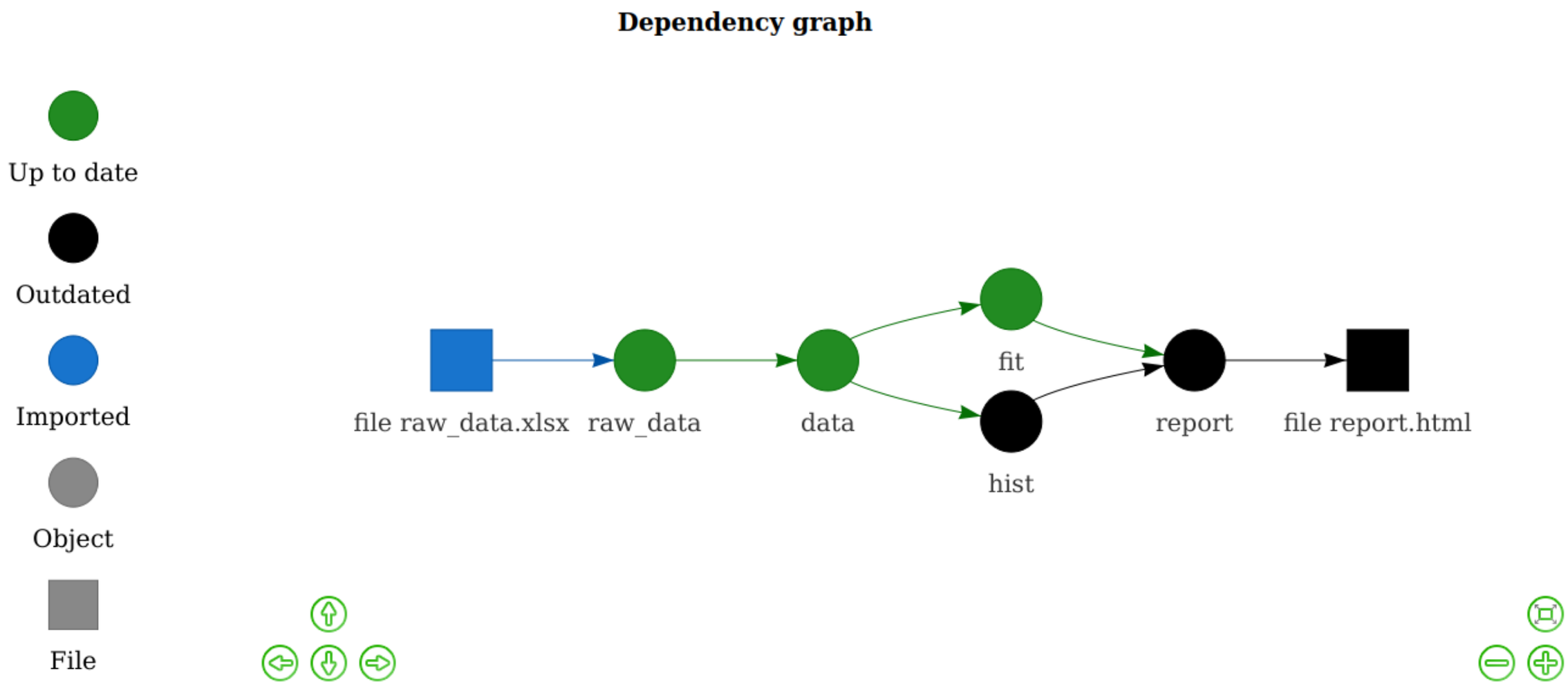
Tools that will help: `drake` package

https://github.com/ropensci/drake

- Package that analyzes and reproduces workflows.

- Identifies aspects of pipeline that have changed, only reruns those.

- Creates plan that outlines steps of analysis.



More Resources

https://github.com/ropensci/drake – drake package

https://github.com/jdblischak/workflowr – workflowr package

https://jdblischak.github.io/workflowr/articles/wflow-01-getting-started.html – workflowr getting started vignette

https://moodymudskipper.github.io/flow/articles/Draw-a-function.html – flow package vignette

https://bookdown.org/yihui/rmarkdown/ – R Markdown book

https://towardsdatascience.com/the-most-underrated-r-packages-254e4a6516a1 – Most underrated packages in R