

R in production

Code (and data) is a shared responsibility

Hadley Wickham

Chief Scientist, Posit

August 2024



1. Sharing data

2. Sharing code

This the hardest part of production
to simulate in a workshop

Sharing data

Use parquet!

parquet vs csv

- Like csv, every language can read and write.
- Binary column-oriented format (much faster to read).
- Rich type system (including missing values).
- Efficient encodings + optional compression (smaller files).
- Data stored in “chunks”
(can work in parallel & don't need to read entire file).
- Supports complex data types (e.g. list-columns).
- Many modern databases can use parquet files directly.

Three key R packages

- **nanoparquet**: zero dependency reader/writer
- **arrow**: powerful entrypoint to the arrow ecosystem
- **duckdb** / **duckplyr**: immediately put your parquet files to work in a database
- (And in python: fastparquet, arrow, duckdb, polars)

Read and writing

```
# https://r-lib.github.io/nanoparquet/
```

```
nanoparquet::read_parquet()
```

```
nanoparquet::write_parquet()
```

```
# https://r4ds.hadley.nz/arrow
```

```
arrow::open_dataset()
```

```
arrow::write_dataset(format = "parquet")
```

```
# https://duckdblabs.github.io/duckplyr/
```

```
duckplyr::duckplyr_df_from_parquet()
```

```
duckplyr::df_to_parquet()
```

You can also use SQL directly on parquet files with duckdb

```
library(duckdb)

con ← DBI::dbConnect(duckdb::duckdb())

DBI::dbExecute(con, "
  CREATE VIEW pollen AS
    SELECT * FROM read_parquet('data/*.parquet');"
)

DBI::dbGetQuery(con, "SELECT count(*) FROM pollen;")
DBI::dbGetQuery(con, "SELECT * FROM pollen LIMIT 10;")
DBI::dbGetQuery(con, "
  SELECT date, count(*)
  FROM pollen
  WHERE count > 0
  GROUP BY date;
")
```


Your turn

- Use `write.csv` to save `ggplot2::diamonds` as a csv file. How long does it take? How big is the resulting file?
- Use `nanoparquet::write_parquet()` to save `ggplot2::diamonds` as a parquet file. How long does it take? How big is the resulting file?
- Use `read.csv()` to load the csv file. How long does it take? How does the output differ to `ggplot2::diamonds`?
- Use `nanoparquet::read_parquet()` to load the parquet file. How long does it take? How does the output differ to `ggplot2::diamonds`?

Data engineering note

- Useful to do “automatic backfill” so if your script fails one day, the next day it will try again.
- Two basic approaches:
 - Create a todo vector and a done vector then use `setdiff()`.
 - Use a for loop and skip the iteration if the file already exists (e.g. houston-pollen).
- Use targets for more sophisticated workflows.

pins

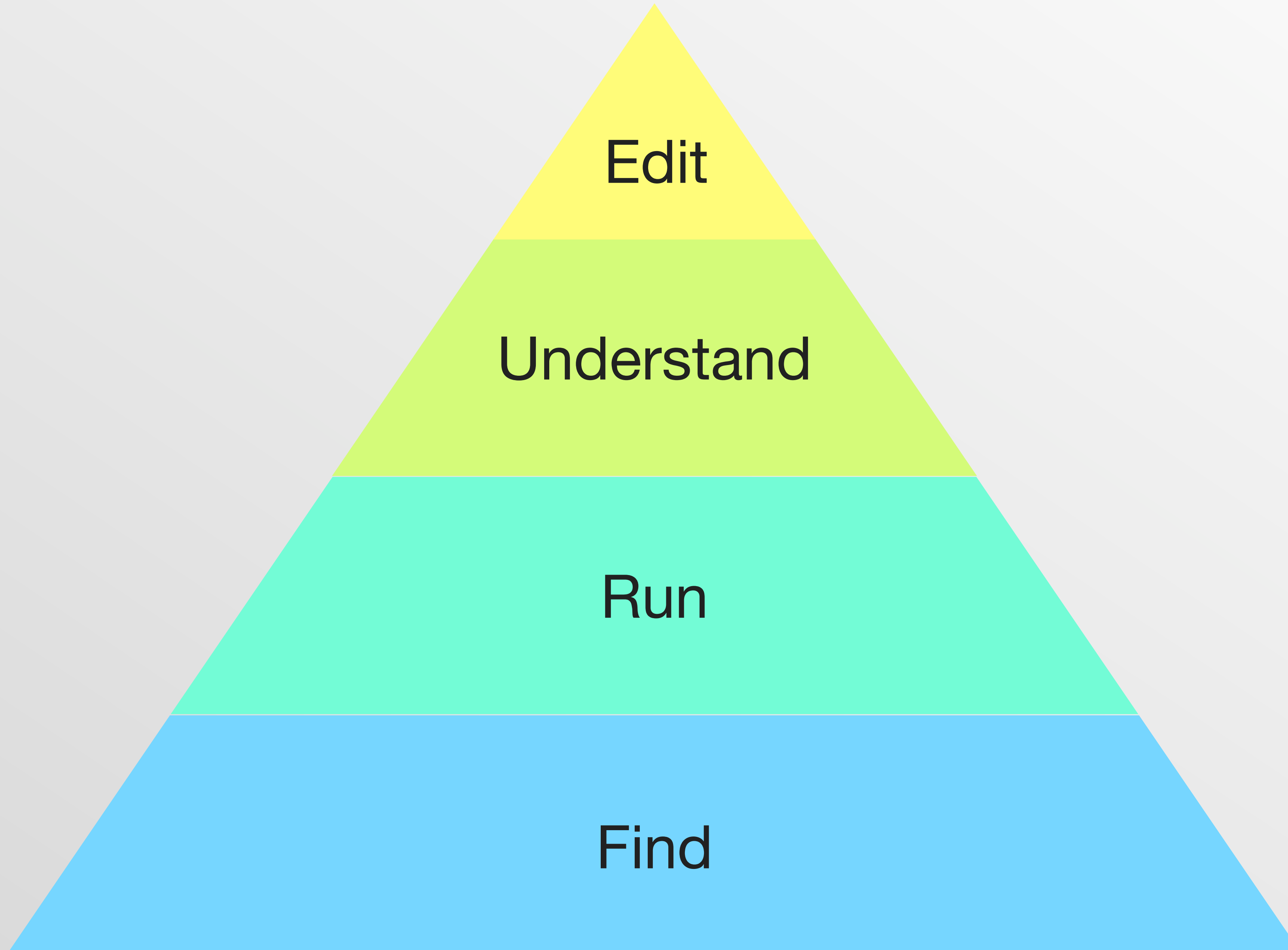
- If you use Posit Connect this is a super easy way to share data with your colleagues (including python users)
- (Will use parquet as default format in the future)

Sharing code

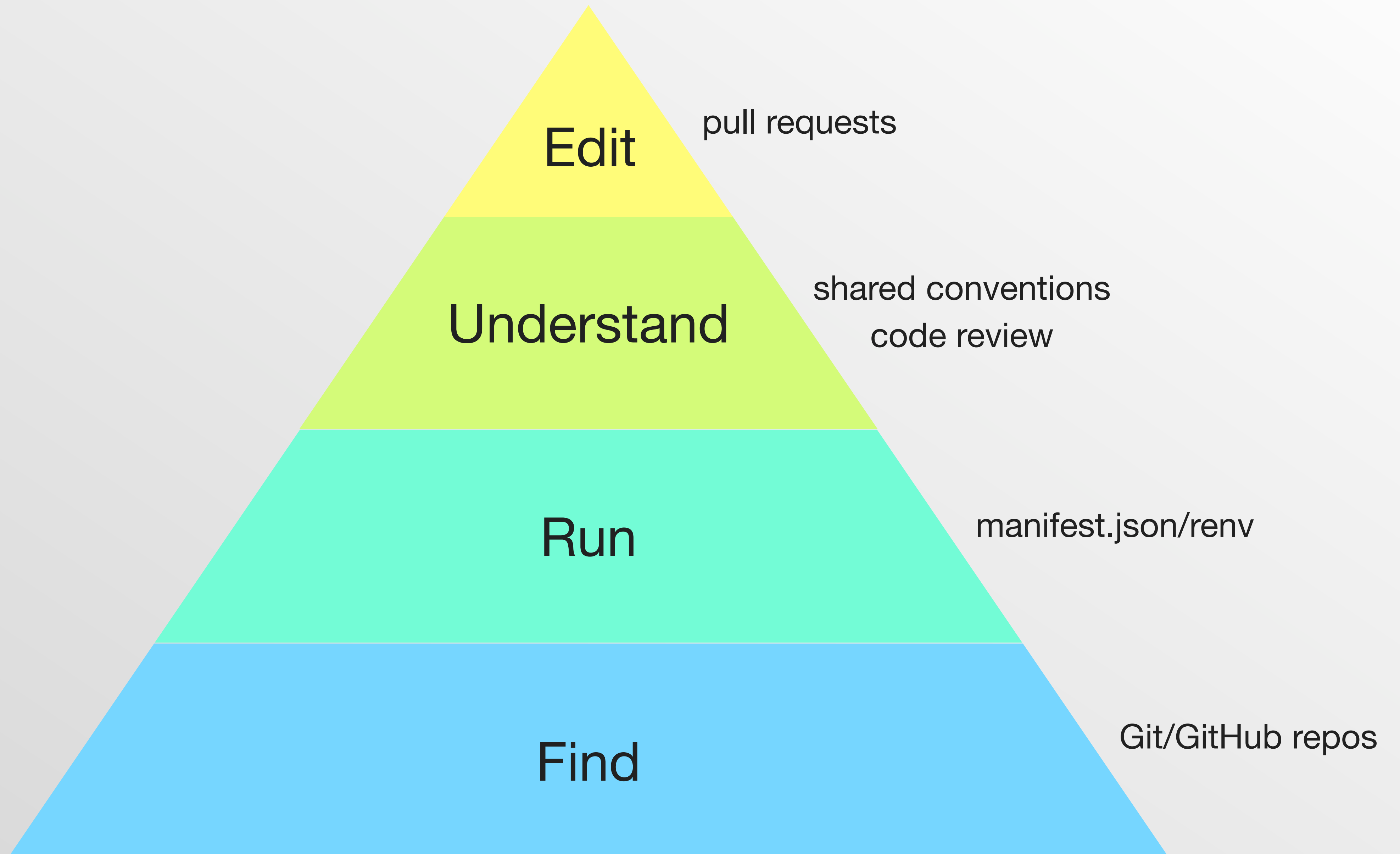
Should your project be a package? 🙄

Should your project be a git repo? 100

A data scientist's hierarchy of code needs



A data scientist's hierarchy of code needs



Find

Find

Scattered across
personal laptops

Shared network
drive

git repos

git repos with
metadata

Tags

If you have a lot of projects (github repos) and you have a hard time finding them, then this is a good place to come up with some conventions around naming and tagging.

I don't think it's too important what the conventions are, just that you have some.

(Where to document? That's coming up.)



Run

Find

Run

```
# manifest.json captures package versions at some point in time
# If it doesn't work on your computer:
renv::renv_lockfile_from_manifest("manifest.json", "renv.lockfile")
renv::restore()

# I highly recommend updating if you have the time
renv::update()
# Check/fix the code
renv::deactivate()
rsconnect::writeManifest()
```



Understand

Run

Find

1. Build shared conventions
2. Review each others code
3. Build a team package

Team style guide

- Build consensus as a team then document your conventions. Some examples:
 - <https://github.com/jtleek/rpackages>
 - <https://github.com/jtleek/datasharing>
 - <https://style.tidyverse.org/>
 - <https://design.tidyverse.org/>
- Make it a quarto book. Deploy it as a batch job.
- Also helps with onboarding!

Code reviews

- Best way to build shared knowledge and shared conventions.
- Helps both reviewer and reviewee.
- In some organisations, every piece of code written gets reviewed. That's likely too much for data science teams, but very worthwhile to allocate to some amount for code review.
- Easiest way to solicit a review is to do a pull request on your own repo. Then you tag in a comment and request a review.

Code review principles

- <https://code-review.tidyverse.org/>
- Define expectations amongst your team for speed of reviews. What needs review and what doesn't? Should you have shared team review time?



- Asking like a DEAR
- Reviewing to GIVE

Asking like a DEAR

- **Describe** the request.
- **Express** how you feel about it!
- **Ask** for specific feedback.
- **Reinforce** kindness with gratitude and responsiveness.

Reviewing to GIVE

- **G**entle
- **I**nterested
- **V**alidate
- **E**asy manner

The rule of three

For functions:

If you've copied and pasted something three times, you should turn it into a function.

For code reviews:

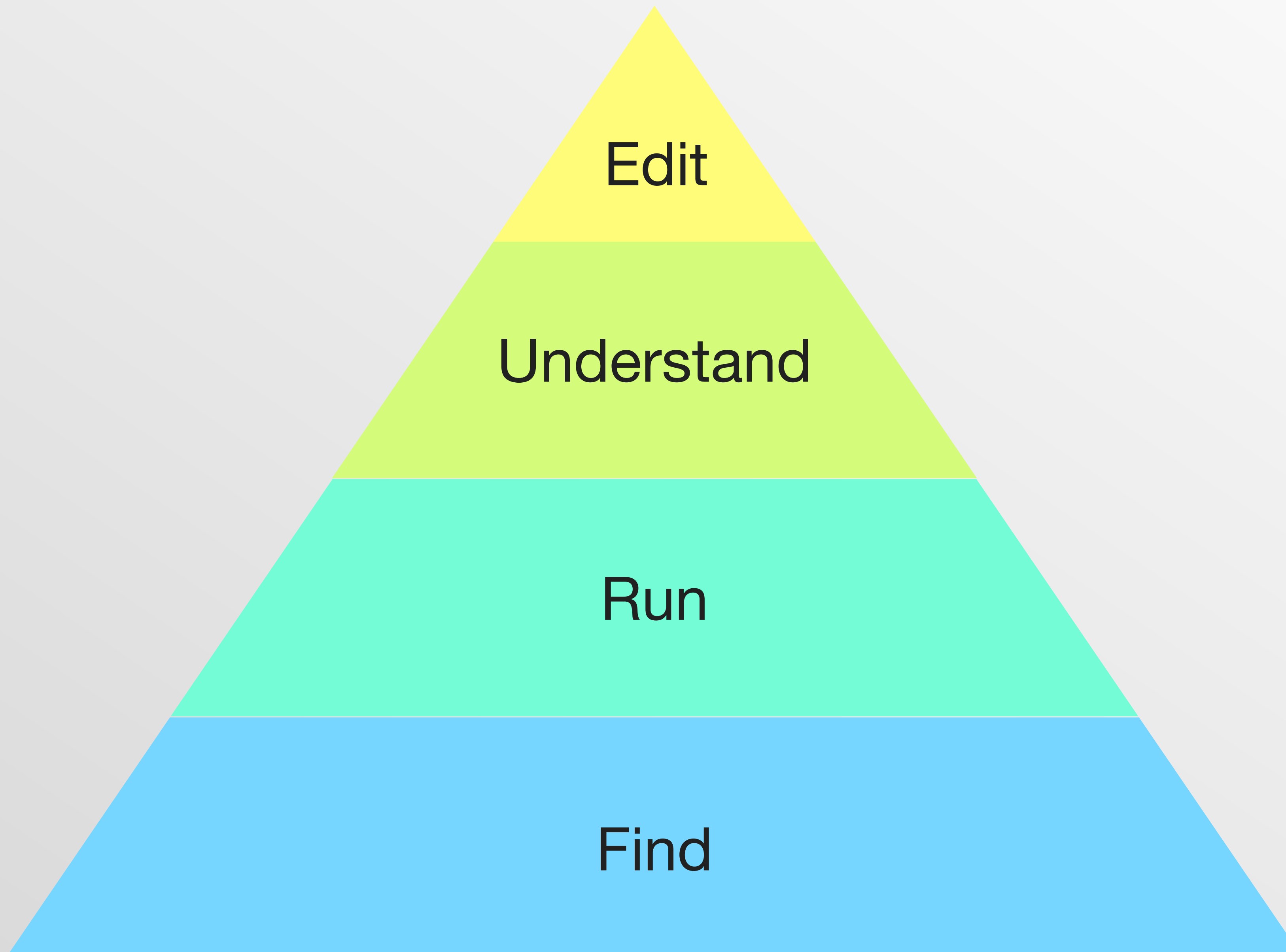
If you've written the same thing three times, you should write it up in your style guide.

Your turn

- Open any of the projects we've worked on today.
- Create a branch (**pr_init()**).
- Make a change and commit it.
- Push it to GitHub (**pr_push()**).
- Share it with your neighbour and get their feedback.
- Accept any suggestions, then get them locally with **pr_pull()**. Make any additional changes, push them to the PR, then merge it.

Team package

- Some conventions are best encoded as actual functions. Those should live in a team package!
- Two places to start:
 - Common data connections
 - Theming functions to match your corporate style guide
e.g. ggplot2, shiny, Rmd/qmd
- Learn how to create a package in <http://r-pkgs.org/>.



Shared ownership

- Valuable to build culture of shared ownership.
- Everyone should have permission to merge, but culture of ask first (unless it's urgent/on vacation etc).
- Write up your conventions on who does the merge (creator or recipient).

Pull request process

```
# Fork and clone the repo
```

```
usethis::create_from_github("{username}/{repo}")
```

```
# Create a branch
```

```
pr_init("{brief-summary}")
```

```
# Make changes and commit
```

```
pr_push()
```

```
# Follow prompts on GitHub
```

Your turn

Make a pull request to your neighbour's repo.

Or work through one of the practice exercises at
<https://github.com/tidyverse/tidy-dev-day/tree/main/practice-pr>