MONASH University

MONASH BUSINESS SCHOOL

ETC4500/ETC5450
Advanced R programming

Week 5: targets and renv – efficient reproducible workflows

Outline

Outline

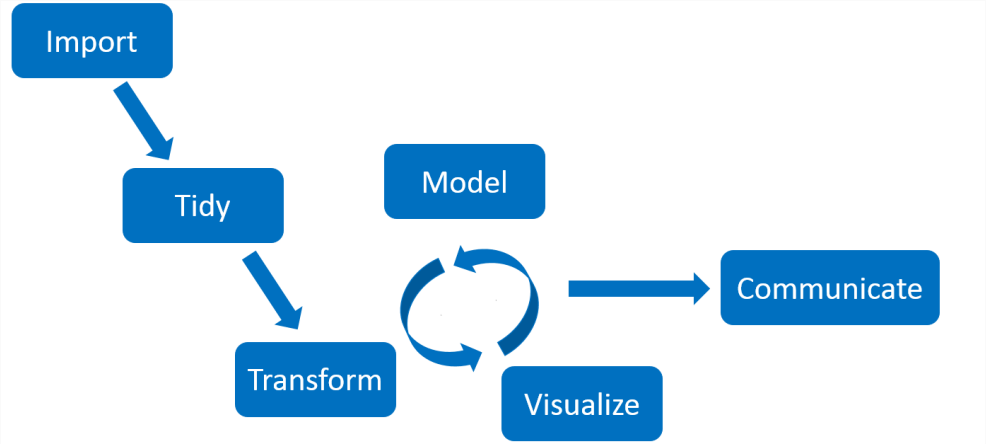
Outline

targets: reproducible computation at scale

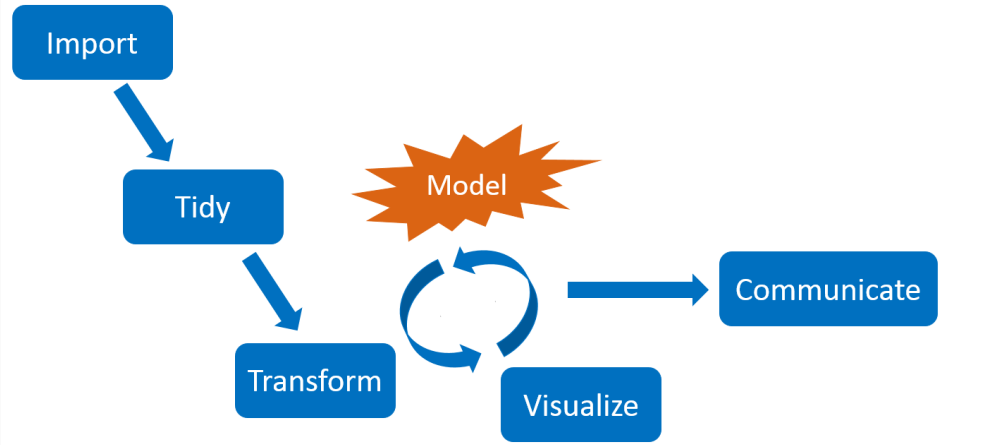
- Supports a clean, modular, function-oriented programming style.
- Learns how your pipeline fits together.
- Runs only the necessary computation.
- Abstracts files as R objects.
- Similar to Makefiles, but with R functions.

Some images from https://wlandau.github.io/targets-tutorial

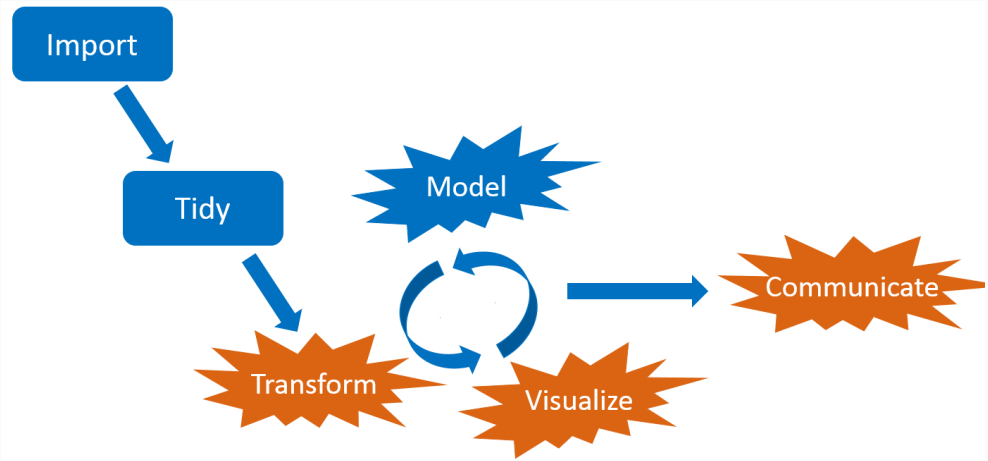
Interconnected tasks



Interconnected tasks



Interconnected tasks



Dilemma: short runtimes or reproducible results?

What do I rerun?

Will the output be reproducible?

Skip to save time?

Skip to save time?

Rerun which models?

Rerun everything?

Rerun everything?

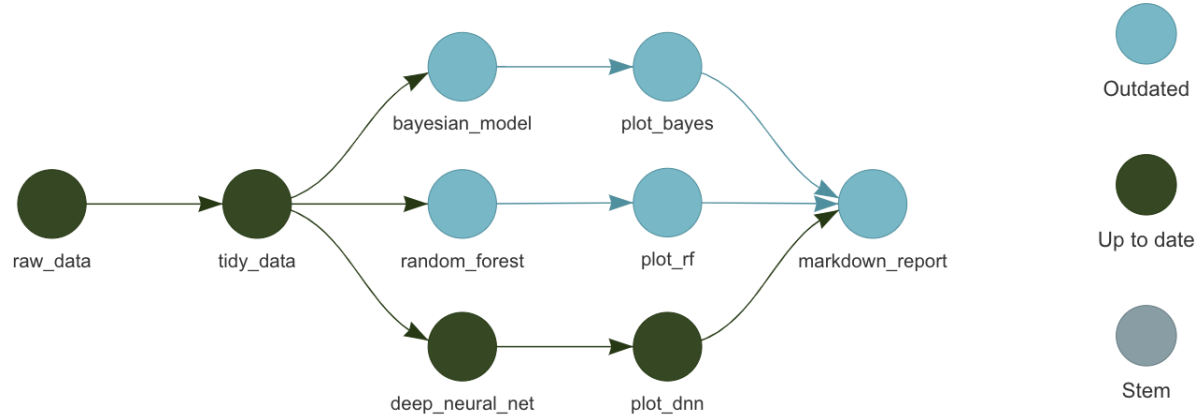
01-import.Rmd

02-tidy.Rmd

03-models.Rmd

04-visualize.Rmd

Let a pipeline tool do the work



- Save time while ensuring computational reproducibility.
- Automatically skip tasks that are already up to date.

Typical project structure

```
_targets.R # Required top-level configuration file.
R/
└── functions.R
data/
└── my_data.csv
```

_targets.R

```
library(targets)
tar_source() # source all files in R folder
tar_option_set(packages = c("tidyverse", "fable"))
list(
  tar_target(my_file, "data/my_data.csv", format = "file"),
  tar_target(my_data, read_csv(my_file)),
  tar_target(my_model, model_function(my_data))
)
```

Generate `_targets.R` in working directory

```
library(targets)
tar_script()
```

Useful targets commands

- `tar_make()` to run the pipeline.
- `tar_make(starts_with("fig"))` to run only targets starting with “fig”.
- `tar_read(object)` to read a target.
- `tar_load(object)` to load a target.
- `tar_load_everything()` to load all targets.
- `tar_manifest()` to list all targets
- `tar_visnetwork()` to visualize the pipeline.
- `tar_destroy()` to remove all targets.
- `tar_outdated()` to list outdated targets.

Debugging

Errored targets to return `NULL` so pipeline continues.

```
tar_option_set(error = "null")
```

Debugging

Errored targets to return `NULL` so pipeline continues.

```
tar_option_set(error = "null")
```

See error messages for all targets.

```
tar_meta(fields = error, complete_only = TRUE)
```

Debugging

Errored targets to return `NULL` so pipeline continues.

```
tar_option_set(error = "null")
```

See error messages for all targets.

```
tar_meta(fields = error, complete_only = TRUE)
```

See warning messages for all targets.

```
tar_meta(fields = warnings, complete_only = TRUE)
```

Debugging

- Try loading all available targets: `tar_load_everything()`. Then run the command of the errored target in the console.
- Pause the pipeline with `browser()`
- Use the debug option: `tar_option_set(debug = "target_name")`
- Save the workspaces:
  - `tar_option_set(workspace_on_error = TRUE)`
  - `tar_workspaces()`
  - `tar_workspace(target_name)`

Random numbers

- Each target runs with its own seed based on its name and the global seed from `tar_option_set(seed = ???)`
- So running only some targets, or running them in a different order, will not change the results.

Folder structure

```
├── .git/
├── .Rprofile
├── .Renviron
├── renv/
├── index.Rmd
├── _targets/
├── _targets.R
├── _targets.yaml
├── R/
├──── functions_data.R
├──── functions_analysis.R
├──── functions_visualization.R
├── data/
└──── input_data.csv
```

targets with quarto

```
library(targets)
library(tarchetypes)                                                    ①
tar_source() # source all files in R folder
tar_option_set(packages = c("tidyverse", "fable"))
list(
  tar_target(my_file, "data/my_data.csv", format = "file"),
  tar_target(my_data, read_csv(my_file)),
  tar_target(my_model, model_function(my_data))
  tar_quarto(report, "file.qmd", extra_files = "references.bib")        ②
  )
```

① Load `tarchetypes` package for quarto support.

② Add a quarto target.

Exercise

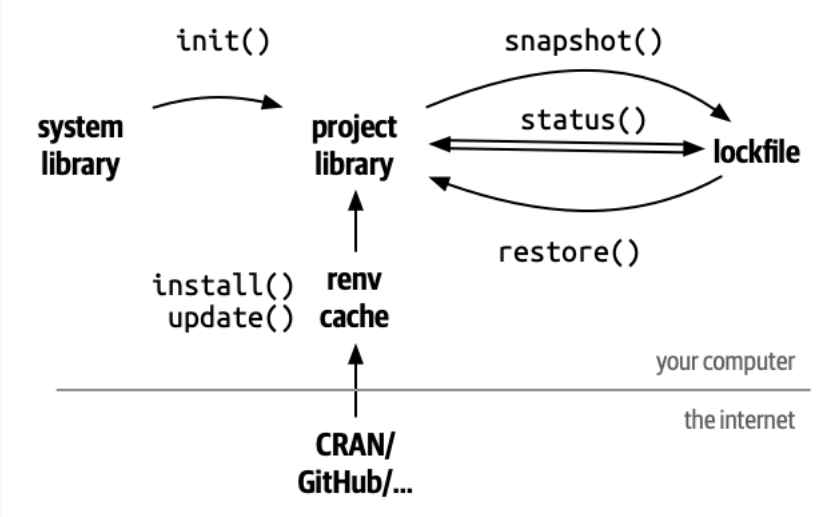
- Add a targets workflow to your quarto document.
- Create a visualization of the pipeline network using `tar_visnetwork()`.

Outline

Reproducible environments

- To ensure that your code runs the same way on different machines and at different times, you need the computing environment to be the same.
  1. Operating system
  2. System components
  3. R version
  4. R packages
- Solutions for 1–4: Docker, Singularity, `containerit`, `rang`
- Solutions for 4: `packrat`, `checkpoint`, `renv`

renv package



renv package

- `renv::init()` : initialize a new project with a new environment. Adds:
  - `renv/library` contains all packages used in project
  - `renv.lock` contains metadata about packages used in project
  - `.Rprofile` run every time R starts.
- `renv::snapshot()` : save the state of the project to `renv.lock`.
- `renv::restore()` : restore the project to the state saved in `renv.lock`.

renv package

- renv uses a package cache so you are not repeatedly installing the same packages in multiple projects.
- `renv::install()` can install from CRAN, Bioconductor, GitHub, Gitlab, Bitbucket, etc.
- `renv::update()` gets latest versions of all dependencies from wherever they were installed from.
- Only R packages are supported, not system dependencies, and not R itself.
- renv is not a replacement for Docker or Singularity.
- `renv::deactivate(clean = TRUE)` will remove the renv environment.