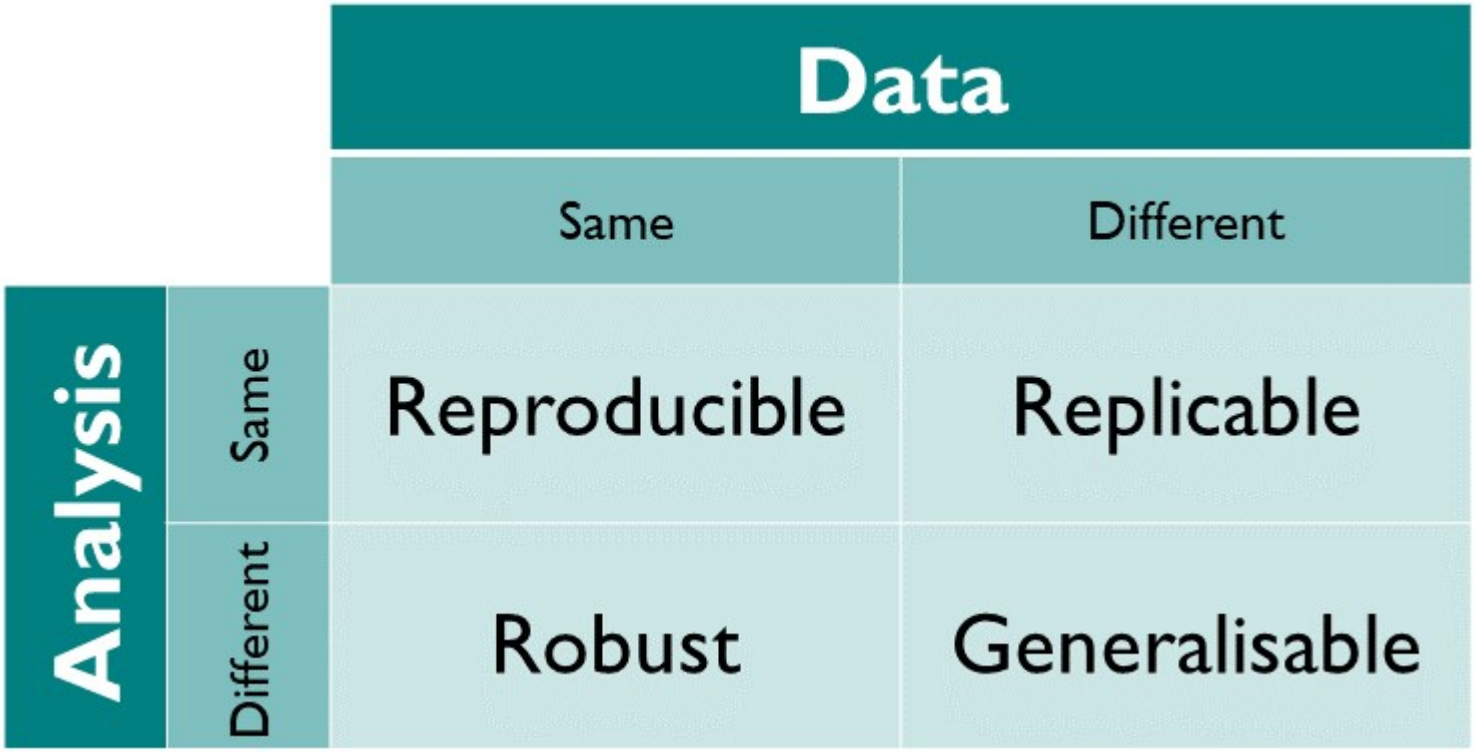
Hit your reproducibility {targets}

UK Gov Data Science Festival, 2020-09-30

mattdray matt-dray rostrum.blog

Reproducevangelism



From The Turing Way by The Alan Turing Institute

Reproducible Analytical Pipelines

Can I recreate what you did:

- from scratch?
- on a different machine?
- in the future?
- without you present?

R has many reproducibility tools, like:

- RStudio Projects to keep everything together
- R Markdown for reproducible docs
- packages for reusable functions
- {here} for relative filepaths
- {renv} for dependency management

Today's focus:

1. Make *workflows* reproducible
2. Try {targets}

1. Make workflows reproducible

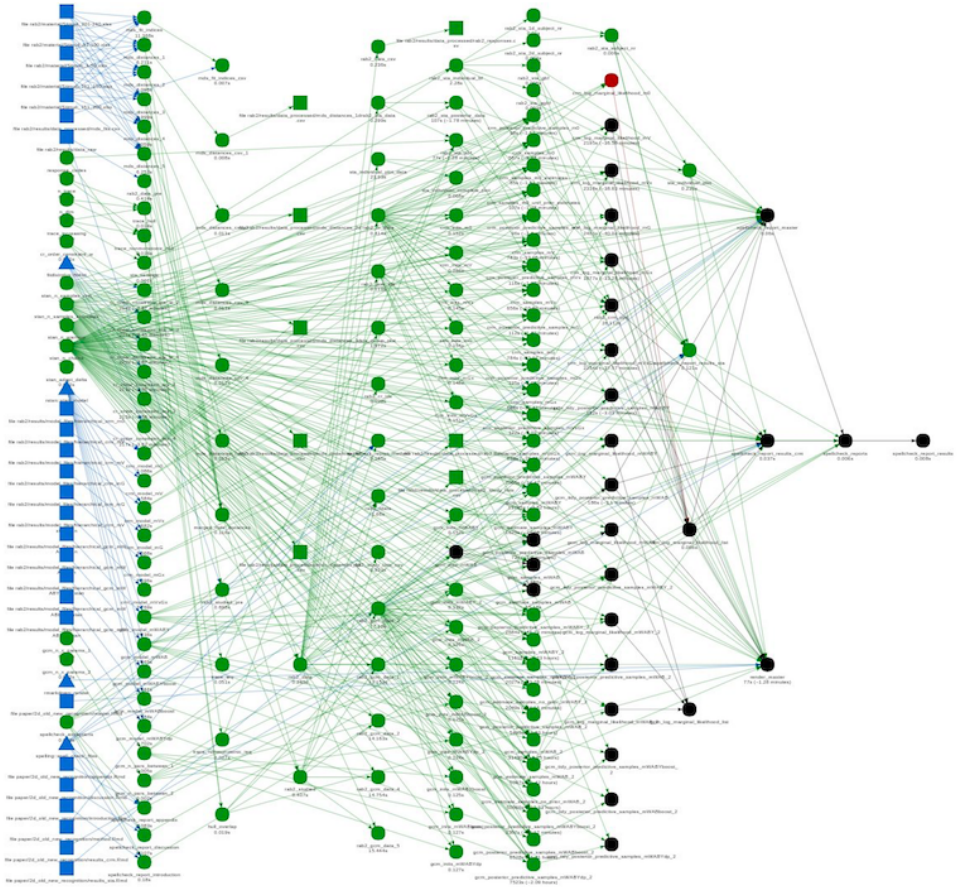
What about your analytical *workflow* itself?

How do you keep track of function, file and object relationships?

What if:

- you haven't recorded the steps?
- the interdependencies become complex?
- some steps are computationally intensive?
- something changes?

You can't remember this



Let a workflow manager handle it

- Dagobah - Simple DAG-based job scheduler in Python.
- Dagr - A scala based DSL and framework for writing and executing bioinformatics pipelines as Directed Acyclic GRaphs.
- Dagster - Python-based API for defining DAGs that interfaces with popular workflow managers for building data applications.
- DataJoint - an open-source relational framework for scientific data pipelines.
- Dask - Dask is a flexible parallel computing library for analytics.
- Dockerflow - Workflow runner that uses Dataflow to run a series of tasks in Docker.
- Doit - Task management & automation tool.
- Drake - Robust DSL akin to Make, implemented in Clojure.
- Drake R package - Reproducibility and high-performance computing with an easy R-focused interface. Unrelated to Factual's Drake.
- Dray - An engine for managing the execution of container-based workflows.
- eHive - System for creating and running pipelines on a distributed compute resource.
- Fission Workflows - A fast, lightweight workflow engine for serverless/FaaS functions.
- Flex - Language agnostic framework for building flexible data science pipelines (Python/Shell/Gnuplot).
- Flowr - Robust and efficient workflows using a simple language agnostic approach (R package).

pditommaso/awesome-pipeline

2. Try {targets}

{targets} by Will Landau

From the manual:

> ...learns how your pipeline fits together, skips costly runtime for tasks that are already up to date, runs only the necessary computation, supports implicit parallel computing, abstracts files as R objects, shows tangible evidence that the results match the underlying code and data.

Plus excellent docs:

- the site and source
- the {targets} R package user manual
- minimal example (can be run in RStudio Cloud)
- targetsketch: a Shiny app for learning {targets} and setup new projects

More available from the {targets} README.

Supersedes {drake} by Will Landau

> ...major user-side limitations regarding data management, collaboration, and parallel efficiency

At its simplest:

1. Write a pipeline script
2. Execute the script
3. Change stuff
4. Go to 2

Contrived, simple demo

The goal: a {targets} pipeline that creates a chart and table for an R Markdown file and renders it.

Then change something and re-run the pipeline.

Generate `_targets.R` in your working directory

```
tar_script()
```

Set up functions and options in `_targets.R`

```
tar_option_set(
  packages = c(
    "targets", "tarchetypes", "knitr",
    "dplyr", "ggplot2", "rphylopic",
  )
)
```

```
b_plot <- function(data, image) {
  ggplot(data, aes(id, temp)) +
    geom_boxplot() +
    labs(title = "Weasel temperatures") +
    add_phylopic(image)
}
```

Set up pipeline in `_targets.R`

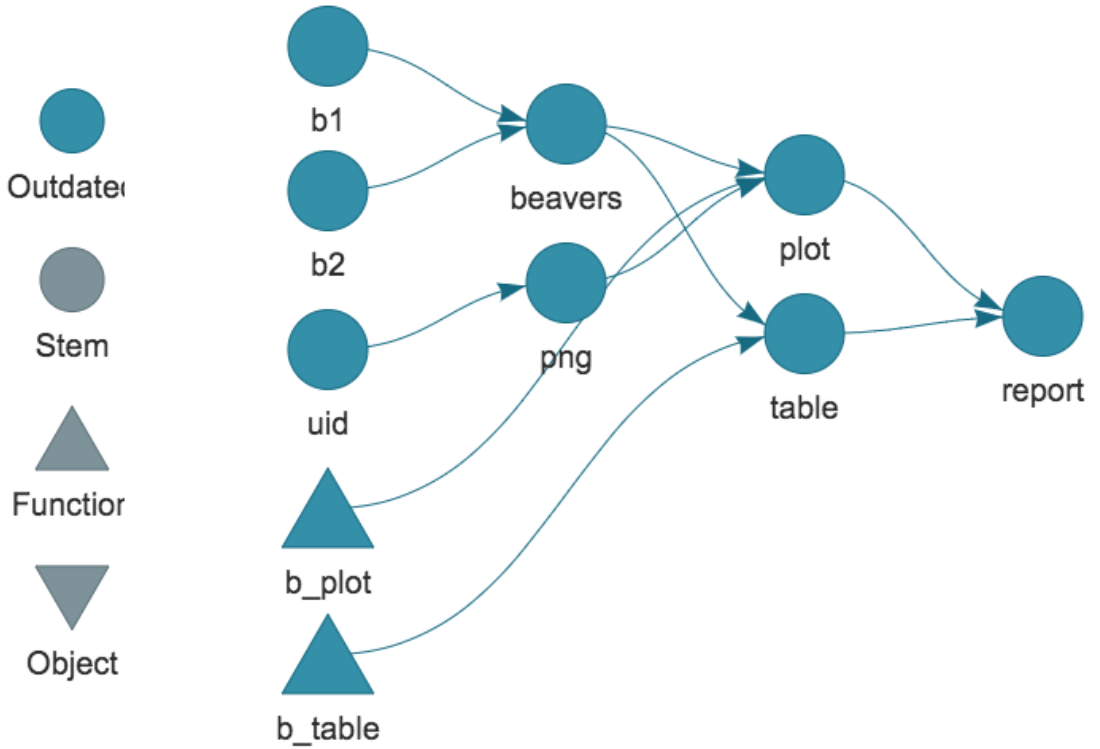
```
targets <- list(
  tar_target(b1, mutate(beaver1, id = "A")),
  tar_target(b2, mutate(beaver2, id = "B")),
  tar_target(beavers, bind_rows(b1, b2)),
  tar_target(uid, "be8670c2-a5bd-4b44-88e8-92f8b0c7f4
  tar_target(png, image_data(uid, size = "512")[[1]])
  tar_target(plot, b_plot(beavers, png)),
  tar_target(table, b_table(beavers)),
  tarchetypes::tar_render(report, "beavers-report.Rmd
)

tar_pipeline(targets)
```

View targets

```
tar_manifest(fields = "command")
```

```
  name    command
  <chr>   <chr>
1 uid     "\"be8670c2-a5bd-4b44-88e8-92f8b0c7f4c6\""
2 report  "tarchetypes::tar_render_run(path = \"beave
3 beavers "bind_rows(b1, b2)"
4 table   "b_table(beavers)"
5 plot    "b_plot(beavers, png)"
6 b1      "mutate(beaver1, id = \"A\")"
7 b2      "mutate(beaver2, id = \"B\")"
8 png     "image_data(uid, size = \"512\")[[1]]"
```

```
tar_visnetwork()
```



Execute the pipeline

```
tar_make()
```

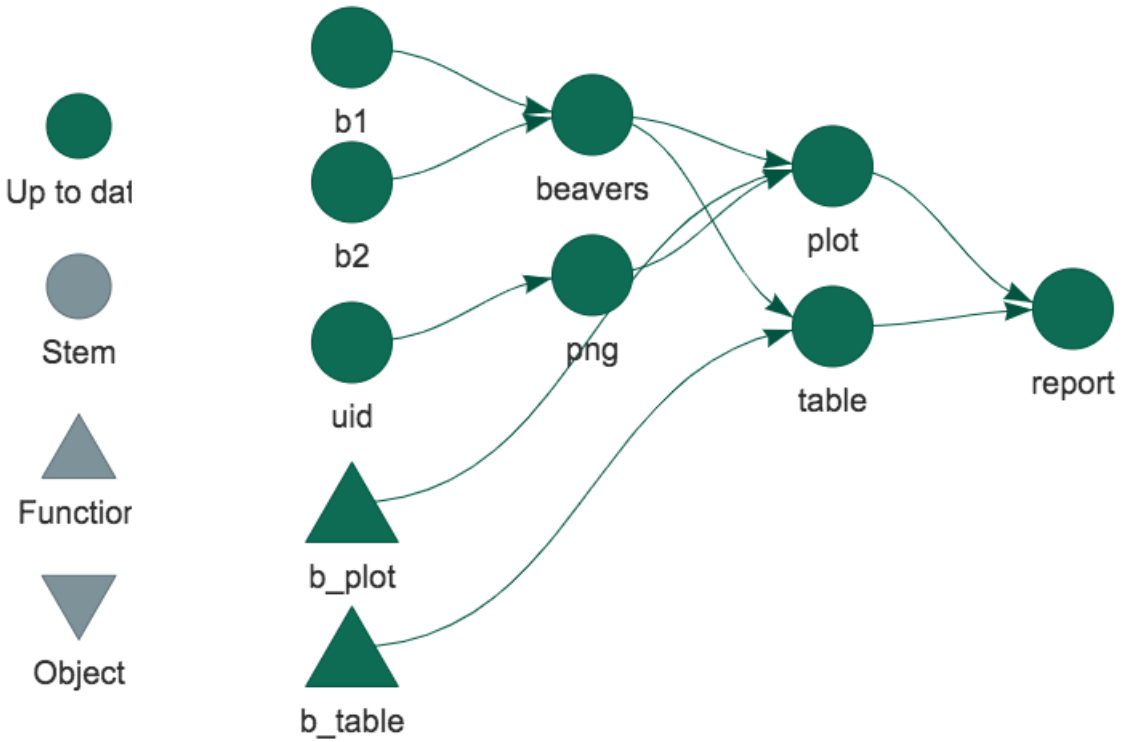
- run target uid
- run target b1
- run target b2
- run target png
- run target beavers
- run target plot
- run target table
- run target report

This created a `_targets/` cache

This is like a 'memory' for {targets}

`tar_load()` and `tar_read()` to fetch objects

```
tar_visnetwork()
```



The rendered R Markdown

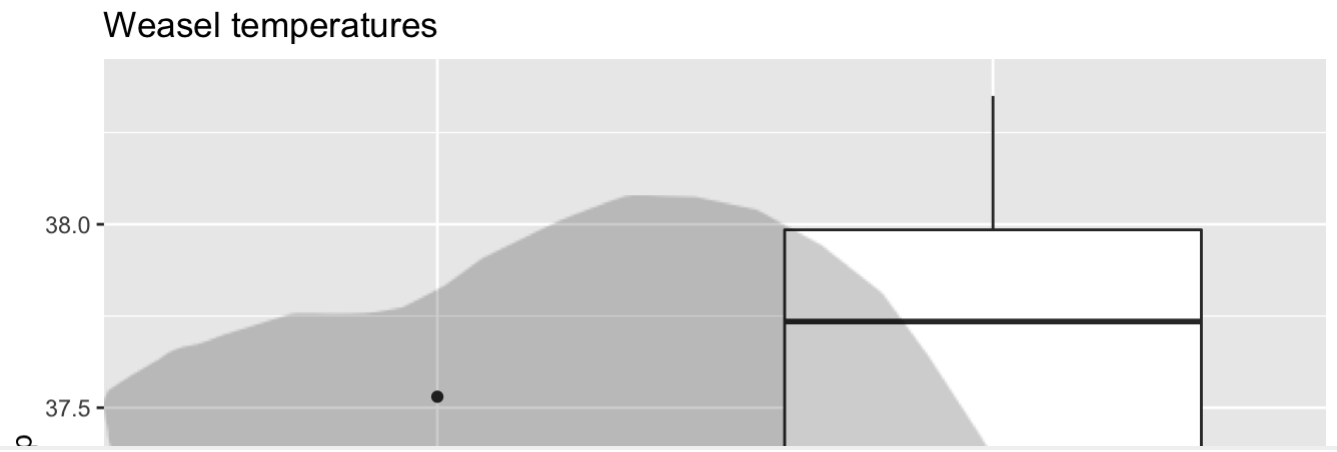
R's built-in beavers dataset

Matt Dray

26 September 2020

Results

What does the distribution of body temperatures look like?



Whoops...

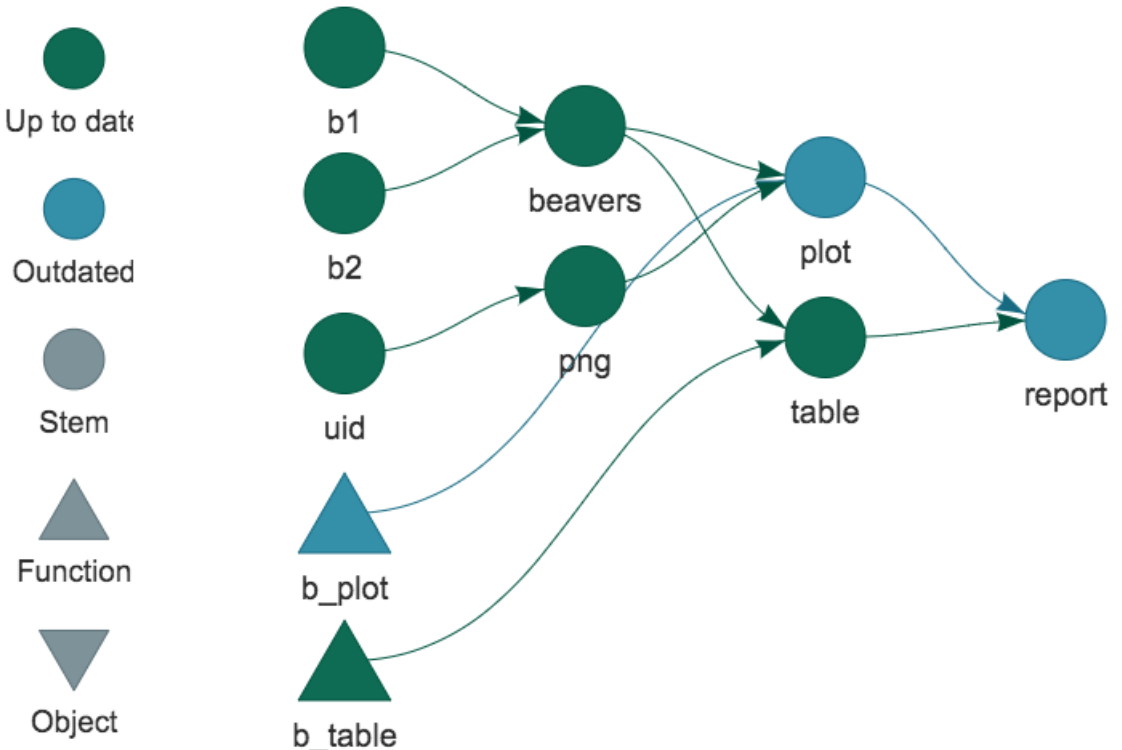
```
- labs(title = "Weasel temperatures")

+ labs(title = "Beaver temperatures")
```

Check what has become outdated as a result

```
tar_outdated()
```

```
[1] "report" "plot"
```

```
tar_visnetwork()
```

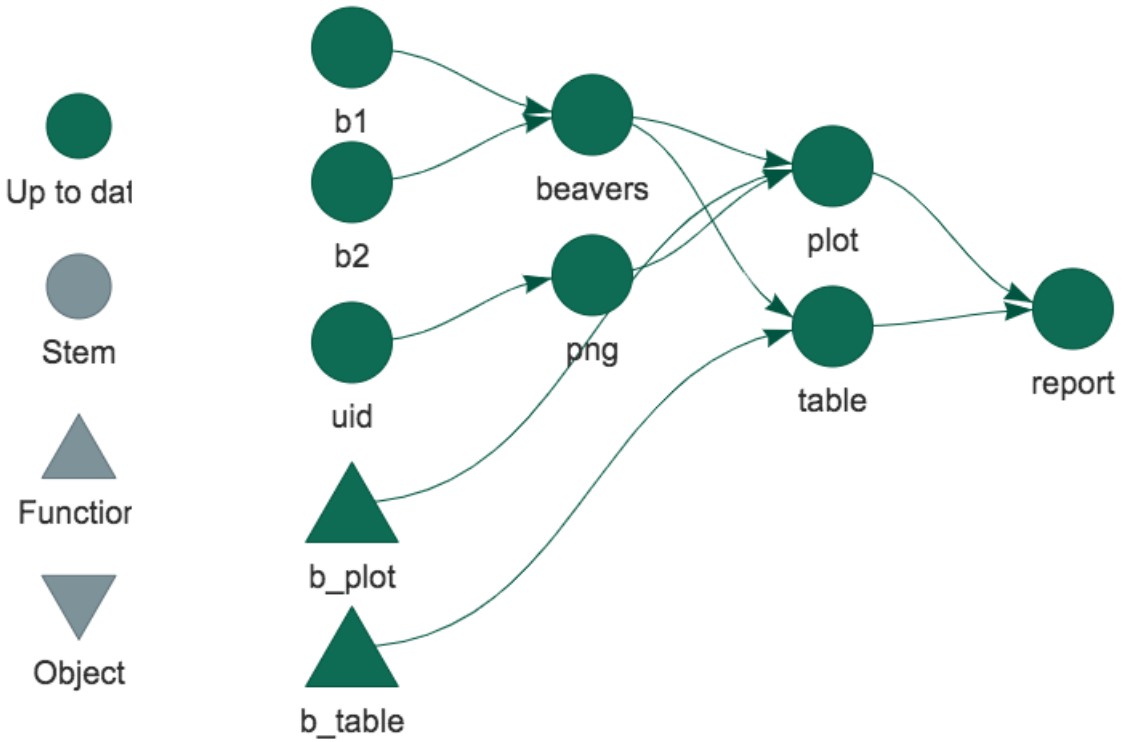


Only outdated targets are re-run

```
tar_make()
```

```
✓ skip target uid
✓ skip target b1
✓ skip target b2
✓ skip target png
✓ skip target beavers
● run target plot
✓ skip target table
● run target report
```

```
tar_visnetwork()
```



The final product

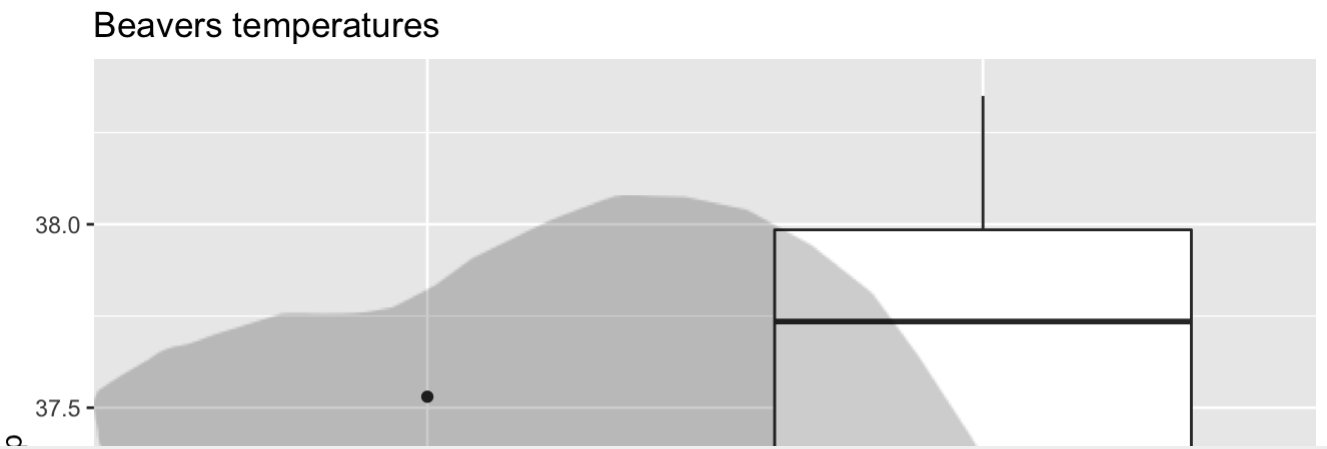
R's built-in beavers dataset

Matt Dray

26 September 2020

Results

What does the distribution of body temperatures look like?



To recap:

1. `tar_script()` creates `_targets.R`
2. Add `tar_targets()`
3. Check `tar_manifest()`
4. `tar_visnetwork()` to visualise
5. Execute with `tar_make()`
6. Change stuff, check `tar_outdated()`
7. Go to 4

Hit your reproducibility {targets}

1. Make *workflows* reproducible
2. Try {targets}

mattdray matt-dray rostrum.blog

Revisit materials:

- these slides
- source code
- blog post
- {drake} version of this talk and a blog post

Image sources:

- RAP hex logo
- {targets} hex logo
- {drake} hex logo
- {drake} hairball by Frederik Aust
- *Castor canadensis* from PhyloPic