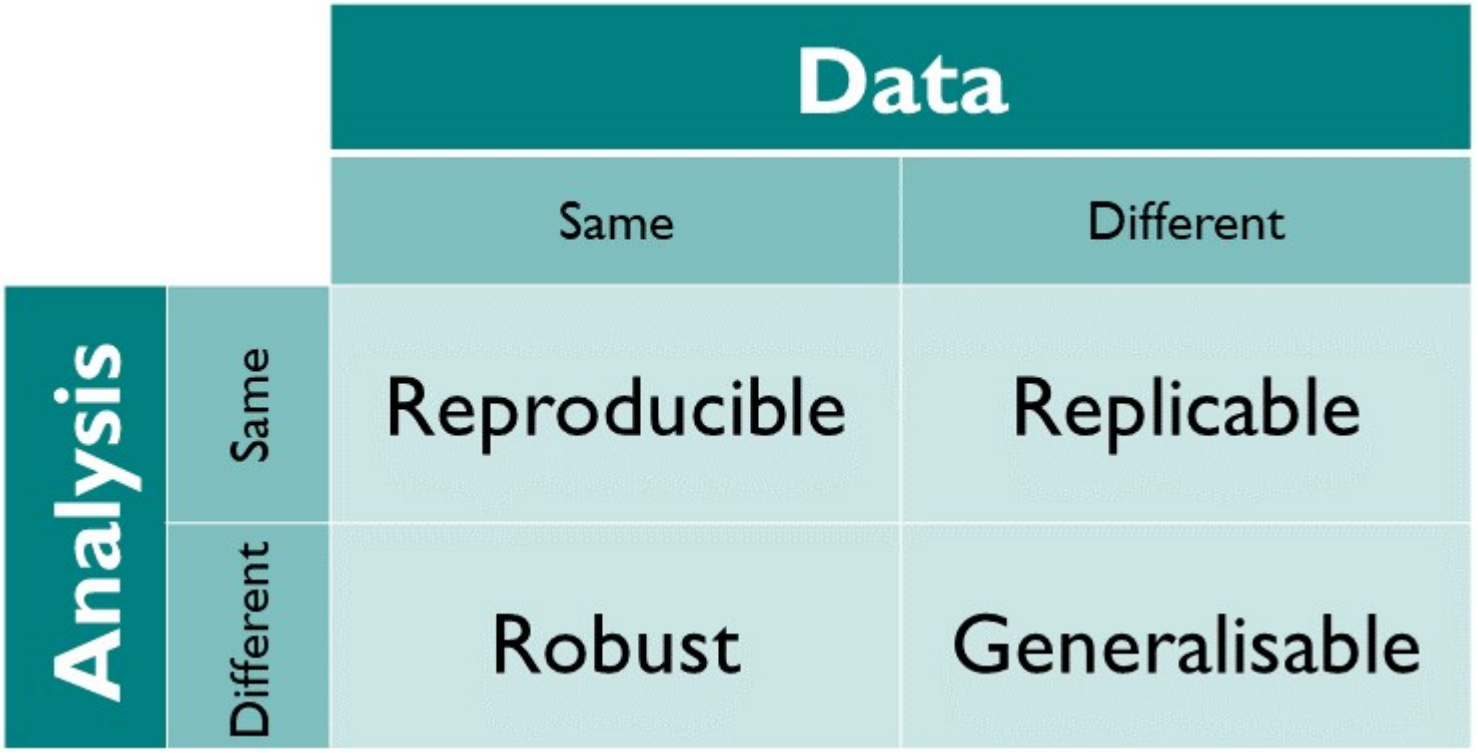
Hit your reproducibility {targets}

UK Government Data Science Festival, 2020-09-30

Matt Dray, Cabinet Office

Reproducevangelism



From The Turing Way by The Alan Turing Institute

Can I recreate what you did:

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

So, ultimately:

- can I trust your outputs?
- can *you* trust your outputs?

Today's focus:

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

1. Make workflows reproducible

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

Reproducible Analytical Pipelines

An example: Can {drake} RAP?

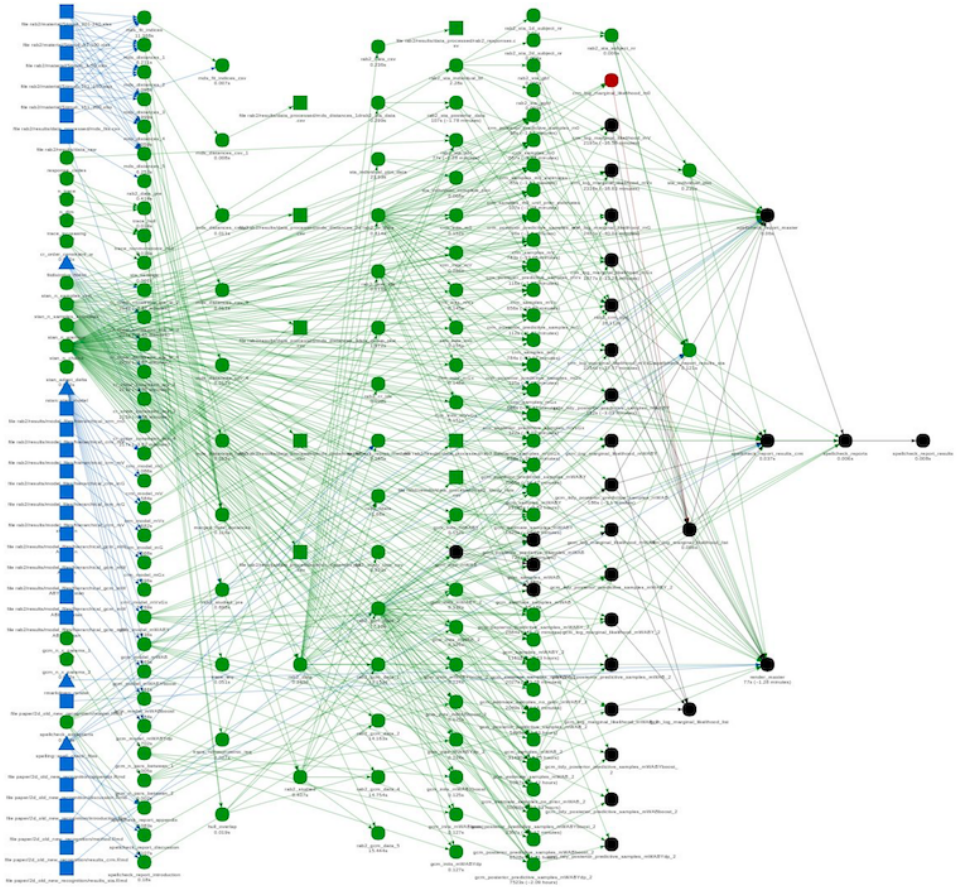
What about your analytical *workflow* itself?

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



Maybe `01-read.R`, `02-wrangle.R`, etc?

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

Supersedes {drake} by Will Landau

{targets} is compelling because it's:

- R-specific
- free
- under active development
- got great documentation and examples

At its simplest:

1. Make a `tar_pipeline()`
2. Run `tar_make()`
3. Change stuff
4. Go to 2

Small demo

The process:

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

What now?

Check out official {targets} materials:

- 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.

Revisit today's materials:

- matt-dray.github.io/targets-dsfest/
- github.com/matt-dray/targets-dsfest

Find me:

mattdray matt-dray rostrum.blog

Hit your reproducibility {targets}

UK Government Data Science Festival, 2020-09-30

Matt Dray, Cabinet Office

Sources

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