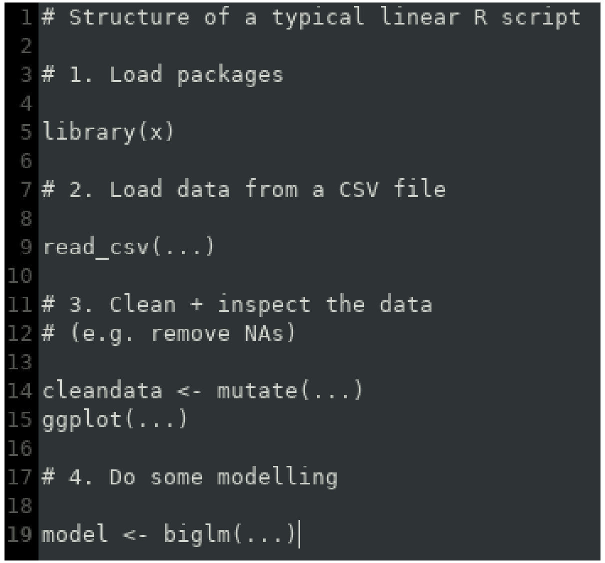
CSC

ICT Solutions for Brilliant Minds

targets: A package for reproducible workflows in R

Jesse Harrison
Analytics Safari (16.04.2021)

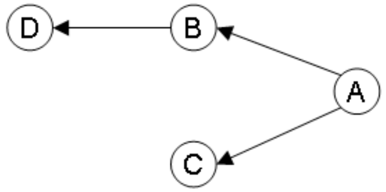
Common problem-solving scenario in R


```
# Structure of a typical linear R script

# 1. Load packages

library(x)

# 2. Load data from a CSV file

read_csv(...)

# 3. Clean + inspect the data
# (e.g. remove NAs)

cleandata <- mutate(...)
ggplot(...)

# 4. Do some modelling

model <- biglm(...)
```

1) Launch a script
2) Wait while it runs
3) Find out there’s an issue
4) Back to square one

$\Big\}$ Delays

A solution: pipeline toolkits



Take a **dependency graph** of the entire workflow

Skip steps (or **targets**) that are unchanged since the last run

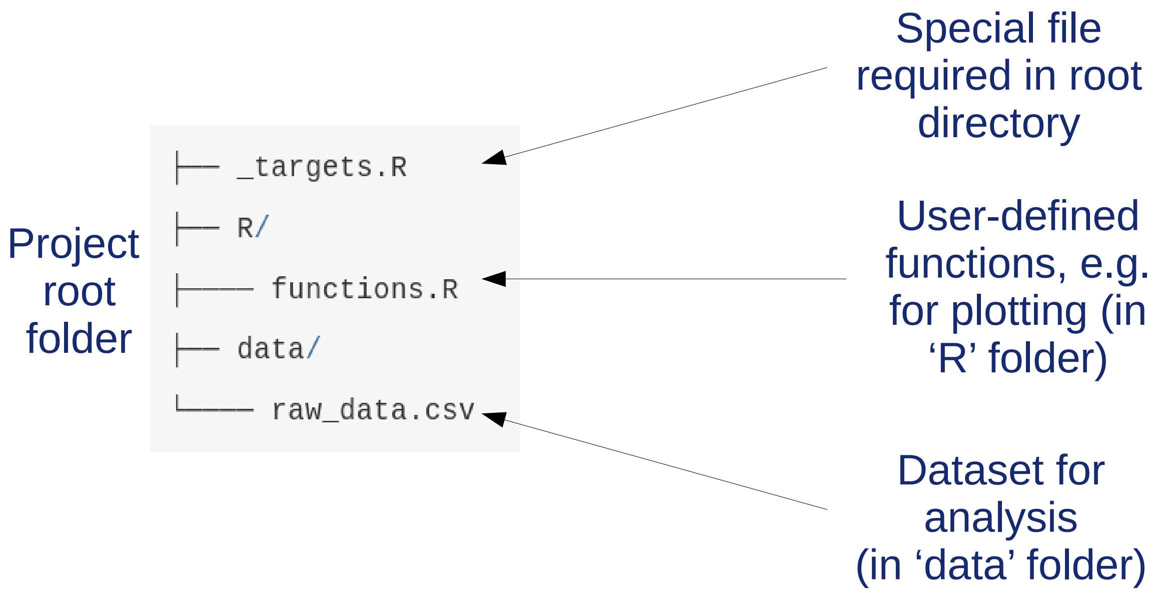
→ When all targets are up-to-date, can be confident that results match the underlying code and data

→ Additionally, can afford significant runtime speed-ups

targets: a pipeline toolkit package for R
(and part of the ‘Targetopia’)

targets
github.com/ropensci/targets
R Targetopia
wlandau.github.io/targetopia/
jagstargets
github.com/wlandau/jagstargets
tarchetypes
github.com/ropensci/tarchetypes
stantargets
github.com/wlandau/stantargets

How it works



```
├── _targets.R
├── R/
├──── functions.R
├── data/
└──── raw_data.csv
```

Project root folder
Special file required in root directory
User-defined functions, e.g. for plotting (in ‘R’ folder)
Dataset for analysis (in ‘data’ folder)

How it works

_targets.R:

- Loads targets and other required packages
- Loads user-defined functions
- Sets default settings for all targets (e.g. data storage formats)
- Defines individual targets (i.e. intermediate workflow steps)

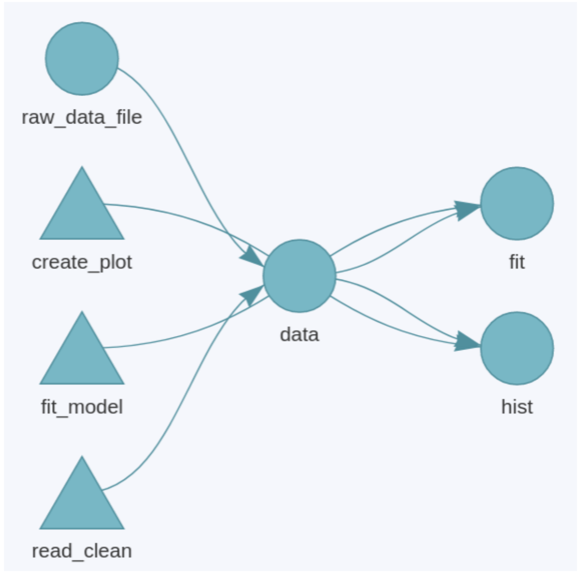
```r
# _targets.R file
library(targets)
tar_option_set(packages = c("biglm", "dplyr", "ggplot2", "readr"))

# Most workflows have custom functions to support the targets.
read_clean ← function(path) {
  path %>%
    read_csv(col_types = cols()) %>%
    mutate(Ozone = replace_na(Ozone, mean(Ozone, na.rm = TRUE)))
}

fit_model ← function(data) {
  biglm(Ozone ~ Wind + Temp, data)
}

create_plot ← function(data) {
  ggplot(data) +
    geom_histogram(aes(x = Ozone), bins = 12) +
    theme_gray(24)
}

# List of targets.
list(
  # airquality dataset in base R:
  tar_target(raw_data_file, "raw_data.csv", format = "file"),
  tar_target(data, read_clean(raw_data_file)),
  tar_target(fit, fit_model(data)),
  tar_target(hist, create_plot(data))
)
```

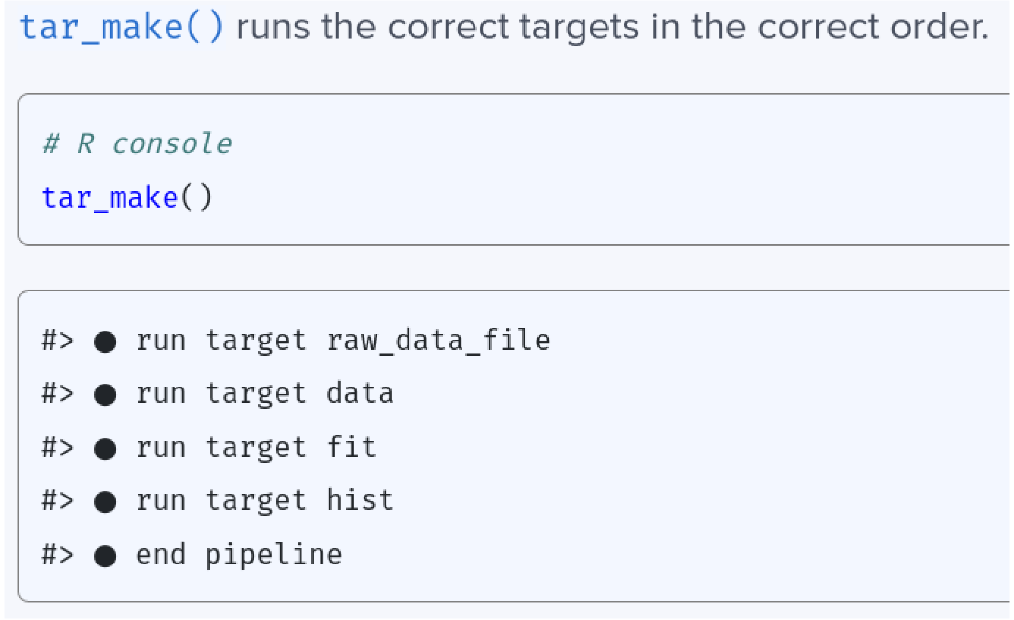
Packages

Functions
(can also be given directly in _targets.R)

Targets (listed at the end)

tar_visnetwork()
For interactive dependency graphs





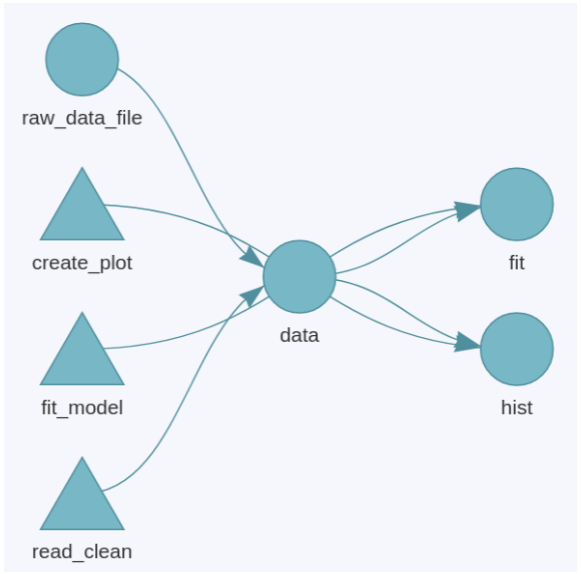
`tar_make()` runs the correct targets in the correct order.

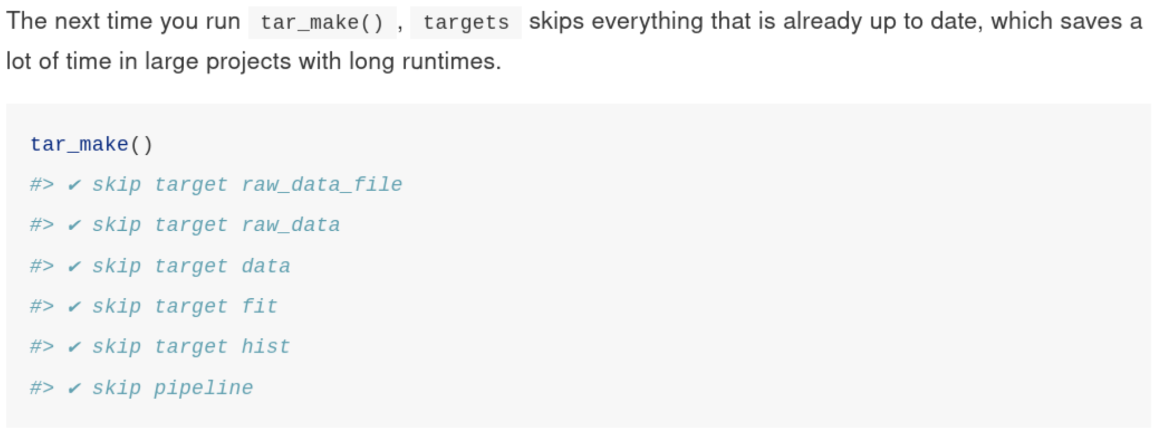
```
# R console
tar_make()
```

```
#> ● run target raw_data_file
#> ● run target data
#> ● run target fit
#> ● run target hist
#> ● end pipeline
```

https://ropensci.org/blog/2021/02/03/targets/

tar_visnetwork()
For interactive dependency graphs





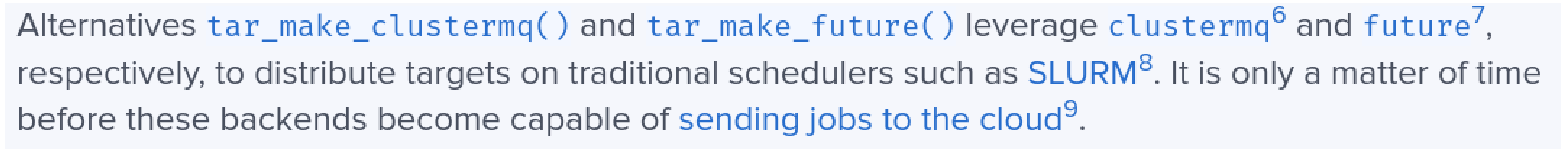
The next time you run `tar_make()`, `targets` skips everything that is already up to date, which saves a lot of time in large projects with long runtimes.

```
tar_make()
#> ✔ skip target raw_data_file
#> ✔ skip target raw_data
#> ✔ skip target data
#> ✔ skip target fit
#> ✔ skip target hist
#> ✔ skip pipeline
```

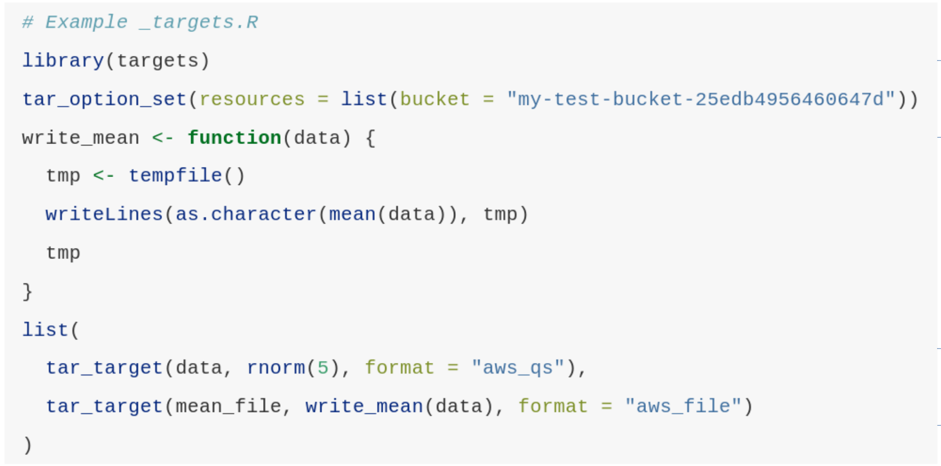
https://ropensci.org/blog/2021/02/03/targets/

Can specify no.s
of workers

Parallelism in cluster environments


Alternatives `tar_make_clustermq()` and `tar_make_future()` leverage `clustermq`[6] and `future`[7], respectively, to distribute targets on traditional schedulers such as SLURM[8]. It is only a matter of time before these backends become capable of sending jobs to the cloud[9].

Storing results in S3 buckets (default is _targets/)

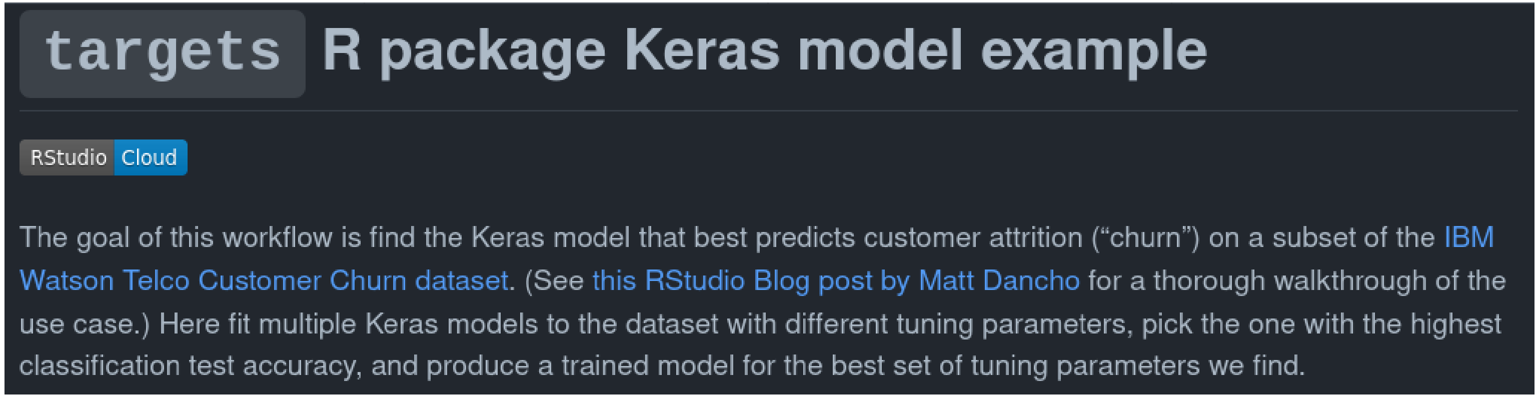

```r
# Example _targets.R
library(targets)
tar_option_set(resources = list(bucket = "my-test-bucket-25edb4956460647d"))
write_mean <- function(data) {
  tmp <- tempfile()
  writeLines(as.character(mean(data)), tmp)
  tmp
}
list(
  tar_target(data, rnorm(5), format = "aws_qs"),
  tar_target(mean_file, write_mean(data), format = "aws_file")
)
```

Specify a bucket

Indicate AWS in target list

Machine learning with targets

https://github.com/wlandau/targets-keras



`targets` R package Keras model example

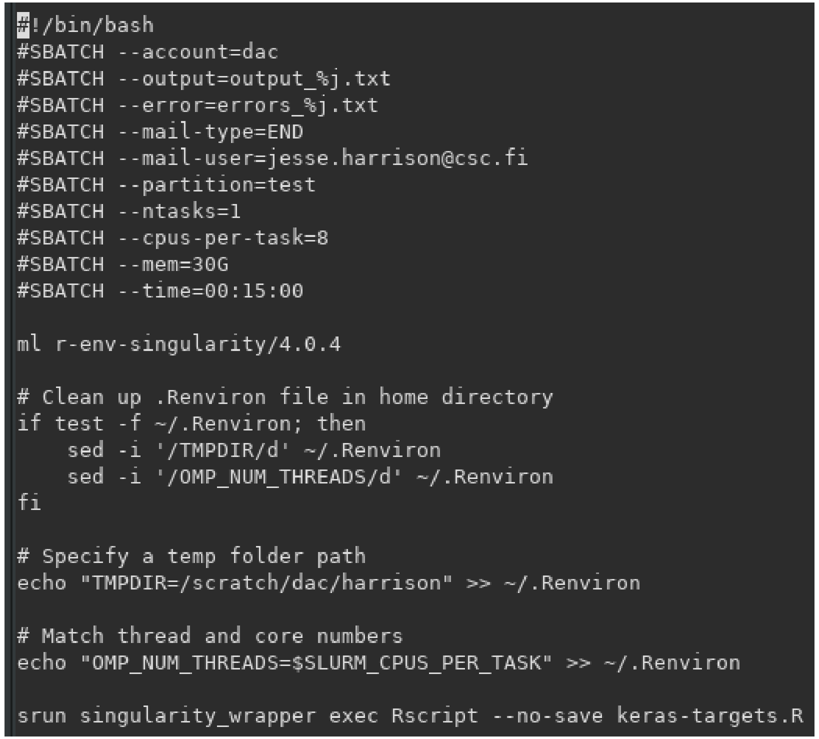
RStudio Cloud

The goal of this workflow is find the Keras model that best predicts customer attrition (“churn”) on a subset of the IBM Watson Telco Customer Churn dataset. (See this RStudio Blog post by Matt Dancho for a thorough walkthrough of the use case.) Here fit multiple Keras models to the dataset with different tuning parameters, pick the one with the highest classification test accuracy, and produce a trained model for the best set of tuning parameters we find.

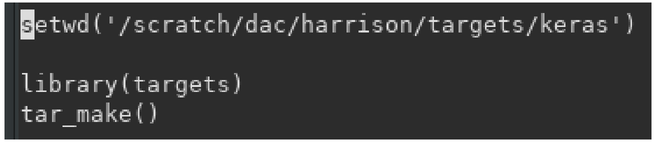
Newest Puhti r-env-singularity module (4.0.4)
comes with CUDA libraries + R interface to TensorFlow

https://docs.csc.fi/apps/r-env-singularity/#r-interface-to-tensorflow

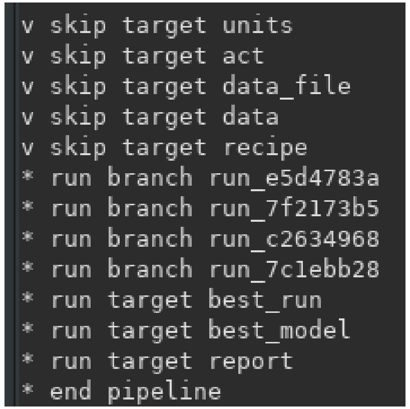
Testing the targets Keras example on Puhti



```
#!/bin/bash
#SBATCH --account=dac
#SBATCH --output=output_%j.txt
#SBATCH --error=errors_%j.txt
#SBATCH --mail-type=END
#SBATCH --mail-user=jesse.harrison@csc.fi
#SBATCH --partition=test
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
#SBATCH --mem=30G
#SBATCH --time=00:15:00

ml r-env-singularity/4.0.4

# Clean up .Renviron file in home directory
if test -f ~/.Renviron; then
    sed -i '/TMPDIR/d' ~/.Renviron
    sed -i '/OMP_NUM_THREADS/d' ~/.Renviron
fi

# Specify a temp folder path
echo "TMPDIR=/scratch/dac/harrison" >> ~/.Renviron

# Match thread and core numbers
echo "OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK" >> ~/.Renviron

srun singularity_wrapper exec Rscript --no-save keras-targets.R
```



```
setwd('/scratch/dac/harrison/targets/keras')

library(targets)
tar_make()
```

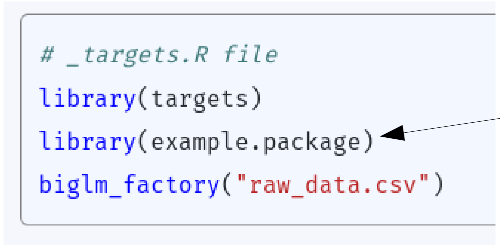


```
v skip target units
v skip target act
v skip target data_file
v skip target data
v skip target recipe
* run branch run_e5d4783a
* run branch run_7f2173b5
* run branch run_c2634968
* run branch run_7c1ebb28
* run target best_run
* run target best_model
* run target report
* end pipeline
```

Needed to install some R packages and then re-run

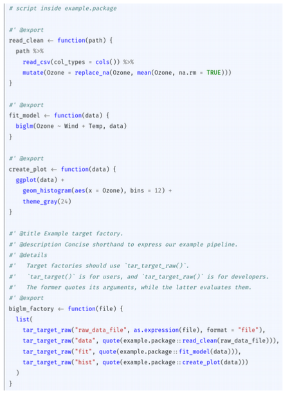
Target factories: making life easier for users

- Pre-packaged function sets that abstract away lower-level code
- For specific / specialised purposes



```r
# _targets.R file
library(targets)
library(example.package)
biglm_factory("raw_data.csv")
```

What the user sees
(three lines!)



```r
# script inside example.package

#' @export
read_clean <- function(path) {
  path %>%
    read_csv(col_types = cols()) %>%
    mutate(Ozone = replace_na(Ozone, mean(Ozone, na.rm = TRUE)))
}

#' @export
fit_model <- function(data) {
  biglm(Ozone ~ Wind + Temp, data)
}

#' @export
create_plot <- function(data) {
  ggplot(data) +
    geom_histogram(aes(x = Ozone), bins = 12) +
    theme_gray(24)
}

#' @title Example target factory.
#' @description Concise shorthand to express our example pipeline.
#' @details
#'   Target factories should use `tar_target_raw()`.
#'   `tar_target()` is for users, and `tar_target_raw()` is for developers.
#'   The former quotes its arguments, while the latter evaluates them.
#' @export
biglm_factory <- function(file) {
  list(
    tar_target_raw("raw_data_file", as.expression(file), format = "file"),
    tar_target_raw("data", quote(example.package::read_clean(raw_data_file))),
    tar_target_raw("fit", quote(example.package::fit_model(data))),
    tar_target_raw("hist", quote(example.package::create_plot(data)))
  )
}
```

All the magic the user doesn't see

Situations where this could be useful

Further ideas / suggestions welcome!

- Jobs with lots of intermediate steps
- Analyses with bottlenecks
- I/O-intensive jobs

Less useful for jobs involving a single big analysis that runs for days…

… but for those, less need for a pipeline toolkit anyway!

repo status Active `targets` is the core engine of the targetopia. It learns the components of your data analysis project, runs the work with distributed computing, and skips steps that are already up to date. It reduces the runtime of successive runs, and it shows tangible evidence that your results match the underlying code and data.

repo status Active `tarchetypes` makes it easy to add certain kinds of common tasks to reproducible pipelines. Most of its functions create families of targets for parameterized R Markdown, simulation studies, and other general-purpose scenarios.

repo status Active `stantargets` is a workflow framework for Bayesian data analysis with `cmdstanr`. With concise, easy-to-use syntax, it defines versatile families of targets tailored to Bayesian statistics, from a single MCMC run with postprocessing to large simulation studies.

repo status Active Like `stantargets`, `jagstargets` is a workflow framework for Bayesian data analysis, with support for both single MCMC runs and large-scale simulation studies. It invokes JAGS through the `R2jags` package, which has nice features such as the ability to parallelize chains across local R processes.

`brmstargets`	repo status Concept `brmstargets` is an idea first proposed here. An implementation is planned, but no work has started. The goal is to accommodate `brms`-powered Bayesian data analysis workflows just as `stantargets` enhances `cmdstanr`.
Other ideas	Following precedent of `stantargets`, it should be possible to extend the R Targetopia to more methodology packages whose users face intense computation, long runtimes, and rapid changes. Possibilities include `greta`, `nimble`, `keras`, `torch`, `torchvision`, `tidymodels`, `mlr3`, and `nlmixr`.