

# Reproducible science in R

with **targets**



A. Ginolhac | HPC School | 2020-12-16



# Introduction to R

Not the scope of this session from the previous sessions

But an introduction to **targets**, a Make-like workflow manager for **R**

- if you are a beginner user, check out [this lecture](#)

## What is R?

R is shorthand for ["GNU R"](#):

- An [interactive](#) programming language derived from S ([J. Chambers](#), Bell Lab, 1976)
- Appeared in 1993, created by [R. Ihaka](#) and [R. Gentleman](#), University of Auckland, NZ
- Focus on data analysis and plotting
- **R** is also shorthand for the ecosystem around this language
  - Book authors
  - Package developers
  - Ordinary useRs

Learning to use **R** will make you [more efficient](#) and [facilitate the use](#) of advanced data analysis tools



- if you are an advanced user, interested in programming, check out [this lecture](#)

## Evaluation in programming

tidyeval



A. Ginolhac | rworkshop | 2020-11-30

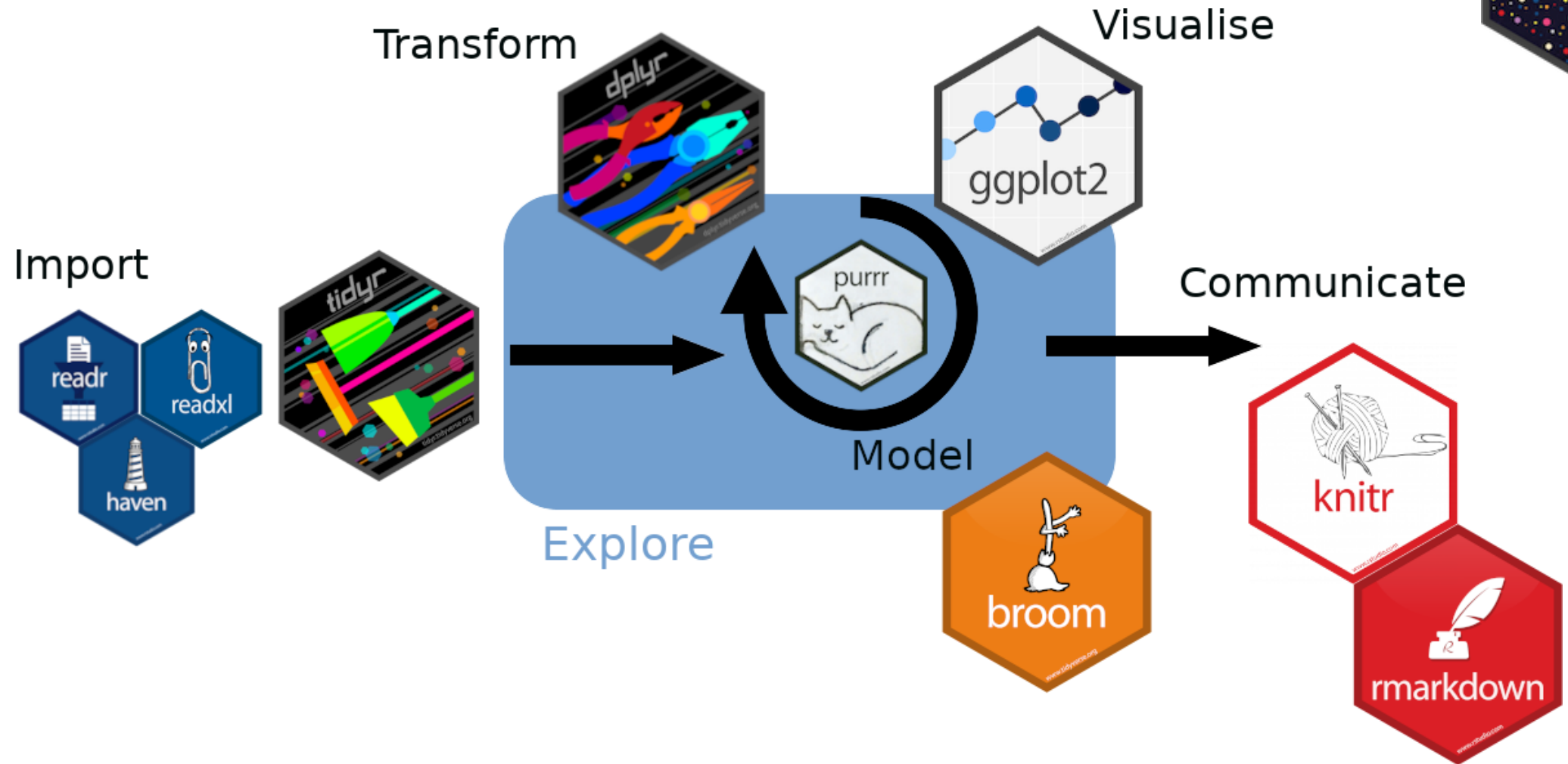
# About this lecture

## Learning objectives

- Why a workflow manager is saving you time and stress
- Understand how it is implemented in **targets**
  - folder structure
  - define your **targets**
  - connect **targets** to create the **dependencies**
  - check **dependencies** with **visnetwork**
  - embrace **dynamic** branching
  - run **only** what needs to be executed
  - bundle **dependencies** in a Rmarkdown document with **tar\_render()**
  - increase reproducibility with the package manager **renv**
- Practice on a dataset, running > 100 linear models in the **tidyverse** framework



# Tidyverse set of packages

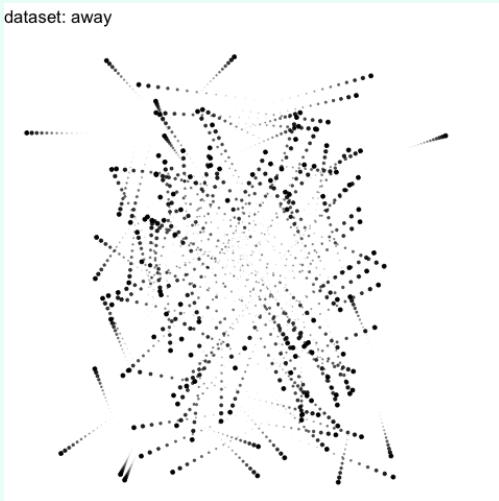


# Practical sessions

## For beginners

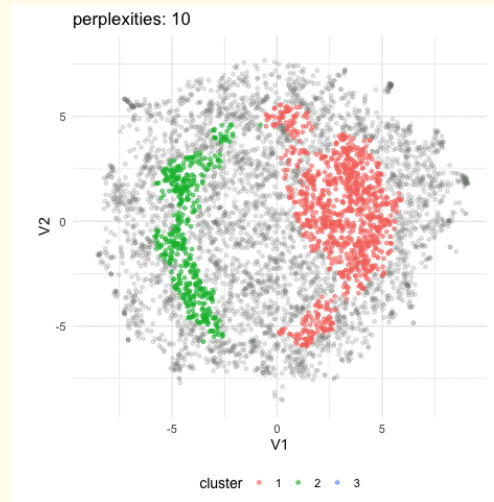
- [dataSaurus](#)

dataset: away



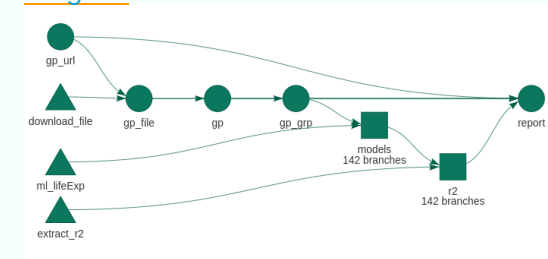
## For intermediates

- [exercises and furrr](#)



## For targets

- [targets](#)



# If you wonder about drake

- [drake](#), is the most used workflow manager for R. 1,300 ★ on Github

## Rationale for [rewriting from scratch](#)

- drake is still an excellent choice for pipeline management, but it has permanent user-side limitations.
- targets was created to overcome these limitations and create a smoother user experience.
- Stronger guardrails by design.
- A friendlier, lighter, more transparent data management system.
- Show which functions are up to date.
- More flexible dynamic branching.
- Improved parallel efficiency.
- Designed for custom user-side metaprogramming ([tarchetypes](#)/[stantargets](#)).

## Not on CRAN yet

```
library(remotes)
# Landau is not using 'master'
install_github("wlandau/targets@main")
install_github("wlandau/tarchetypes@main")
```

- [Debugging!](#) with [workspaces](#) (wishing this exists in [snakemake](#))

Source: targets [slides](#) W. Landau. NY Meetup hackR

# Folder structure

```
.git/  
run.R  
_targets.R  
_targets/  
Repro.Rproj  
R  
  functions.R  
  utils.R  
run.R*  
renv/  
renv.lock  
report.Rmd
```

- with [renv](#)
- [\\_targets.R](#) is the only mandatory file
- use a [R](#) sub-folder for functions, gets closer to a [R](#) package
- a [run.R](#) file for [custom building](#) in RStudio
- [Rmd](#) file allows to gather results in a report
- in a RStudio project
- version tracked with [git](#)



# Main file `_targets.R`

```
# Beginning of _targets.R
library(targets)
source("R/functions.R")
options(tidyverse.quiet = TRUE)
tar_option_set(
  packages = c("dplyr")
)
# Pipeline description
tar_pipeline(
  tar_target(a, 1:3),
  tar_target(b, a + 1,
              pattern = map(a))
)
```

- add `library(tarchetypes)` for extra-utilities
- optional helpers like silencing `tidyverse` loading
- declare packages that will be loaded by default
- each `target` can have their own additional lib calls (avoid loading for nothing)
- using bar names of targets, create the dependencies
- `pattern = map()` creates dynamic branching

## Guardrails

- all targets are fresh R processes
- `_targets.R` must be at the project root



# Formats, Patterns, Iterations

## Formats

- "rds": default
- "qs": faster binary objects (see [qs::qsave\(\)](#))
- "fst": amazing speed for `data.frame`, (see [fst::write\\_fst\(\)](#))
- "fst\_dt": same for `data.table` objects
- "fst\_tbl": same for `tibbles` objects
- "keras"
- "torch"
- "url": automatic Last-Modified time stamp check
- "file": automatic hashing check
- "aws\_rds/aws\_file": for AWS S3 cloud

## Patterns

- [map\(\)](#): iterate over one or more targets in sequence.
- [cross\(\)](#): iterate over combinations of slices of targets.
- [head\(\)](#): restrict branching to the first few elements.
- [tail\(\)](#): restrict branching to the last few elements.
- [sample\(\)](#): restrict branching to a random subset of elements.

## Iterations

- "vector": using [vctrs](#)
- "list": less strict than [vctrs::vec\\_c\(\)](#)
- "group": handy for [dplyr::group\\_by\(\)](#) tibble

and [tar\\_target](#) [reference page](#)

Source: [W. Landau, NYhacker](#)

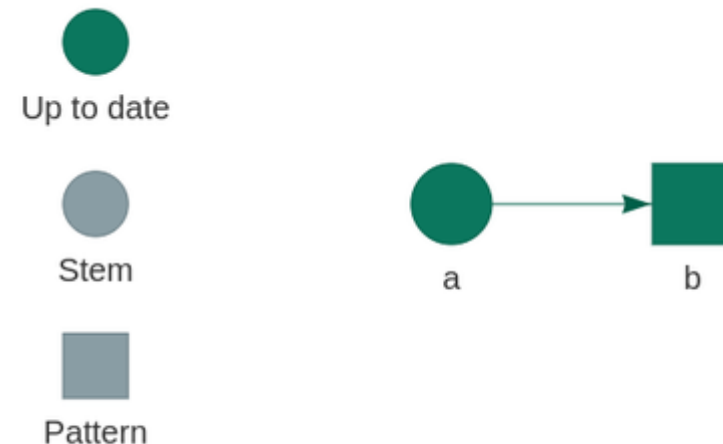
# Toy example

```
library(targets)
tar_pipeline(
  tar_target(a, 1:3),
  tar_target(b, a + 1,
              pattern = map(a))
)
# graph dependency BEFORE tar_make()
tar_visnetwork()
# run the pipeline
tar_make()
#> ● run target a
#> ● run branch b_0a91b2ed
#> ● run branch b_847c5ae2
#> ● run branch b_1619f1a0

# read a target content
tar_read(b_0a91b2ed)
#> [1] 2
```

re-run with `tar_make()`: all skipped

```
tar_make()
#> ✓ skip target a
#> ✓ skip branch b_0a91b2ed
#> ✓ skip branch b_847c5ae2
#> ✓ skip branch b_1619f1a0
#> ✓ Already up to date.
tar_read(b_0a91b2ed)
#> [1] 2
# graph dependency AFTER tar_make()
tar_visnetwork()
```



# Toy example

Add one element to **a**, ➡ one more **b** branch

```
tar_pipeline(  
  tar_target(a, 1:4),  
  tar_target(b, a + 1,  
             pattern = map(a))  
)  
tar_make()  
#> ● run target a  
#> ✓ skip branch b_0a91b2ed  
#> ✓ skip branch b_847c5ae2  
#> ✓ skip branch b_1619f1a0  
#> ● run branch b_9401bbb6
```

change **b** ➡ all branches re-run

```
tar_pipeline(  
  tar_target(a, 1:4),  
  tar_target(b, a + 2,  
             pattern = map(a))  
)  
tar_make()  
#> ✓ skip target a  
#> ● run branch b_0a91b2ed  
#> ● run branch b_847c5ae2  
#> ● run branch b_1619f1a0  
#> ● run branch b_9401bbb6  
tar_read(b_0a91b2ed)  
#> [1] 3
```

# Toy parallelism, using future



future by [Henrik Bengtsson](#) is available. Only one worker

```
library(future)
plan(multisession)
tar_pipeline(
  tar_target(a, 1:3),
  tar_target(b, {
    # fake a long process
    Sys.sleep(5)
    a + 1
  },
  pattern = map(a))
)
system.time(tar_make_future(workers = 1L))
#> ● run target a
#> ● run branch b_0a91b2ed
#> ● run branch b_847c5ae2
#> ● run branch b_1619f1a0
#>   user   system elapsed
#> 2.156    0.284   20.807
```

with 3 workers

```
library(future)
plan(multisession)
tar_pipeline(
  tar_target(a, 1:3),
  tar_target(b, {
    # fake a long process
    Sys.sleep(5)
    a + 1
  },
  pattern = map(a))
)
system.time(tar_make_future(workers = 3L))
#> ● run target a
#> ● run branch b_0a91b2ed
#> ● run branch b_847c5ae2
#> ● run branch b_1619f1a0
#>   user   system elapsed
#> 2.126    0.164   12.920
```

Still overhead, but individual target can be parallelised ([deployment](#) / [resources](#) args)

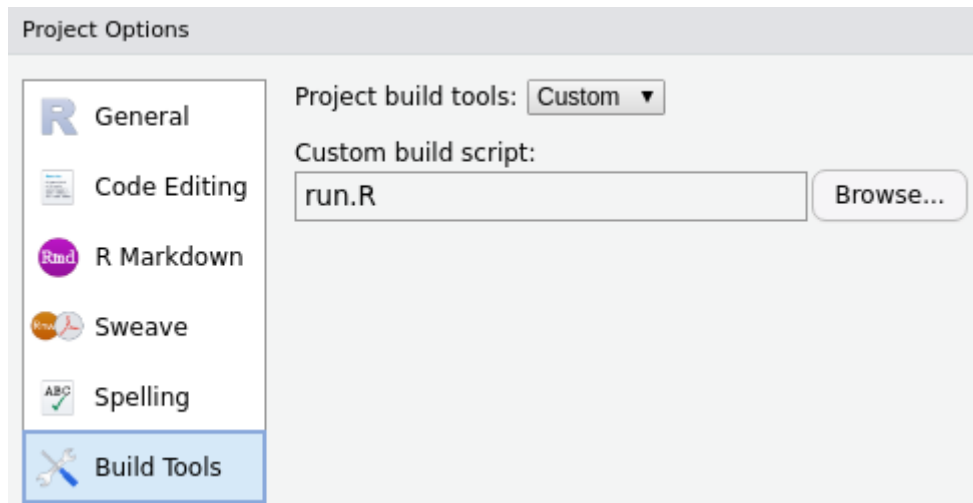
Deploy only heavy computation **remotely** possible with [clustermq](#). TBD

# Custom Build in RStudio

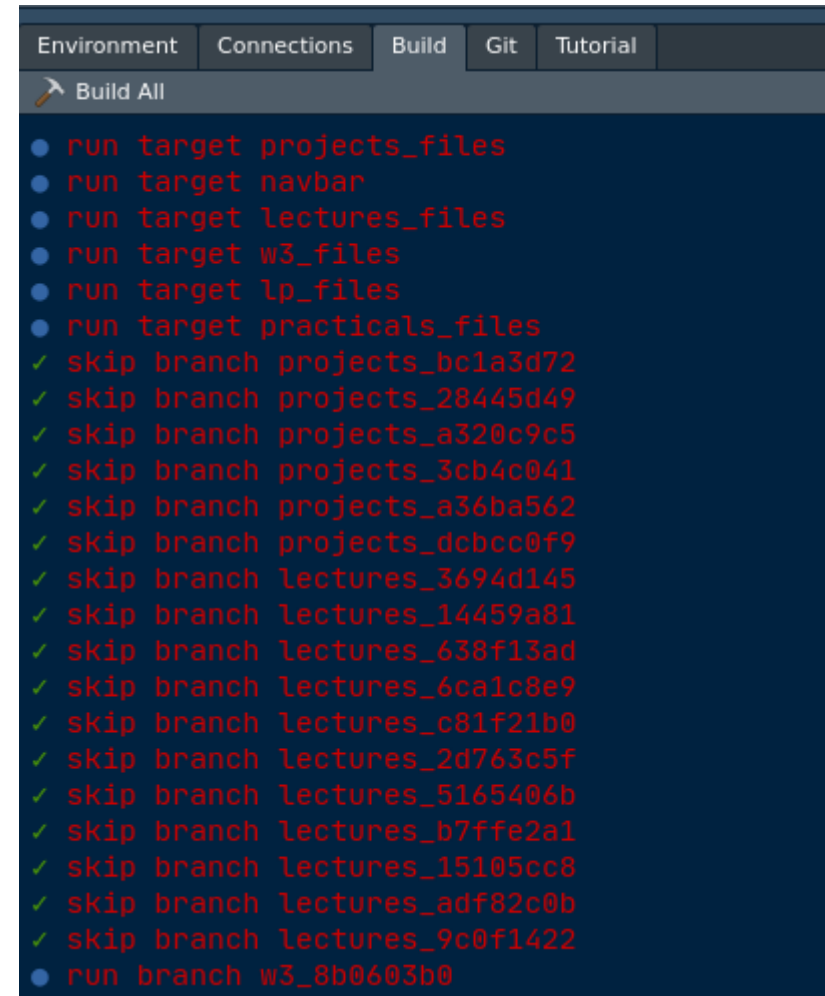
`run.R` is essentially a call to `tar_make()` as seen below:

```
#!/usr/bin/env Rscript  
  
targets::tar_make()
```

Then specified in RStudio project building option:



Example (also nice to use the keyboard shortcut)



**A more realistic example**

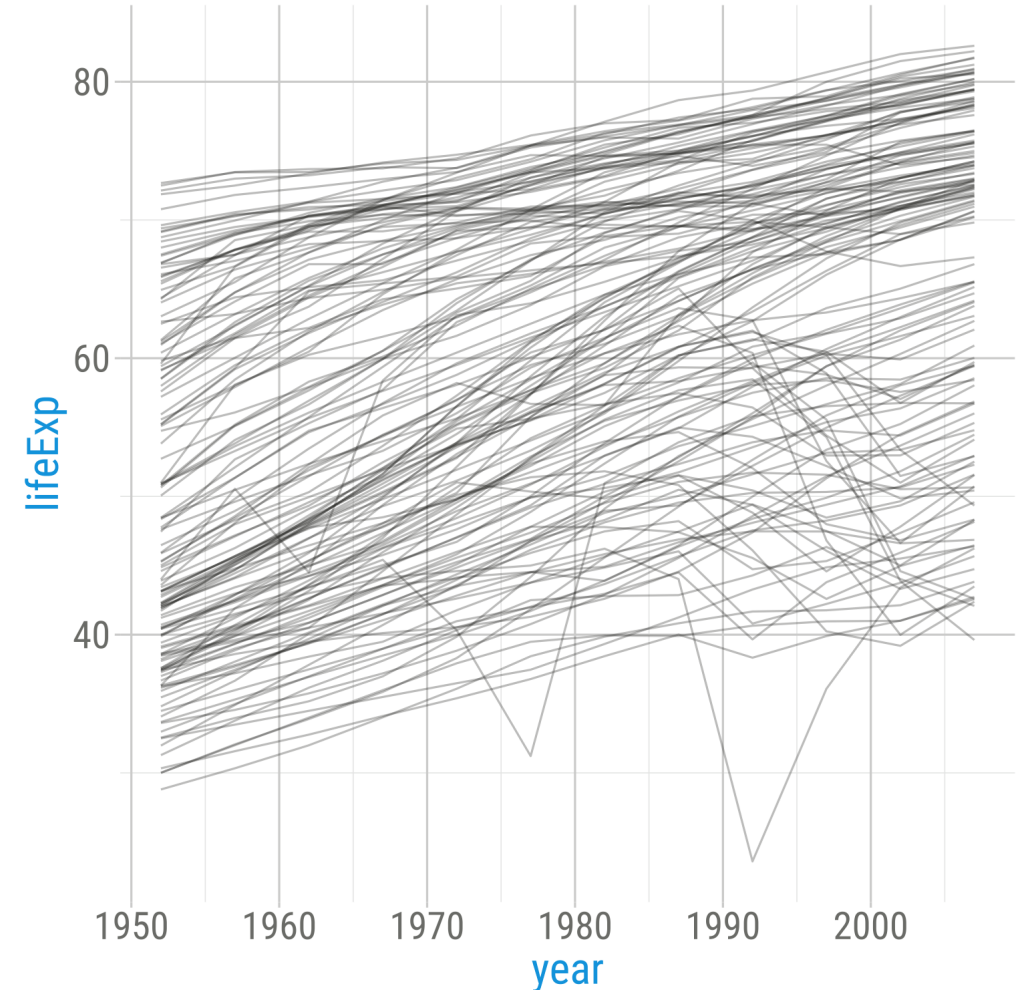
# Linear modelling on the gapminder dataset

142 countries, 12 yearly observations of 3 parameters

country	continent	year	lifeExp	pop	gdpPercap
Afghanistan	Asia	1952	28.8	8425333	779
Afghanistan	Asia	1957	30.3	9240934	821
Afghanistan	Asia	1962	32	10267083	853

...

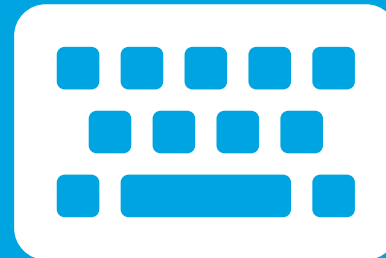
country	continent	year	lifeExp	pop	gdpPercap
Zimbabwe	Africa	1997	46.8	11404948	792
Zimbabwe	Africa	2002	40	11926563	672
Zimbabwe	Africa	2007	43.5	12311143	470





**Now practical**

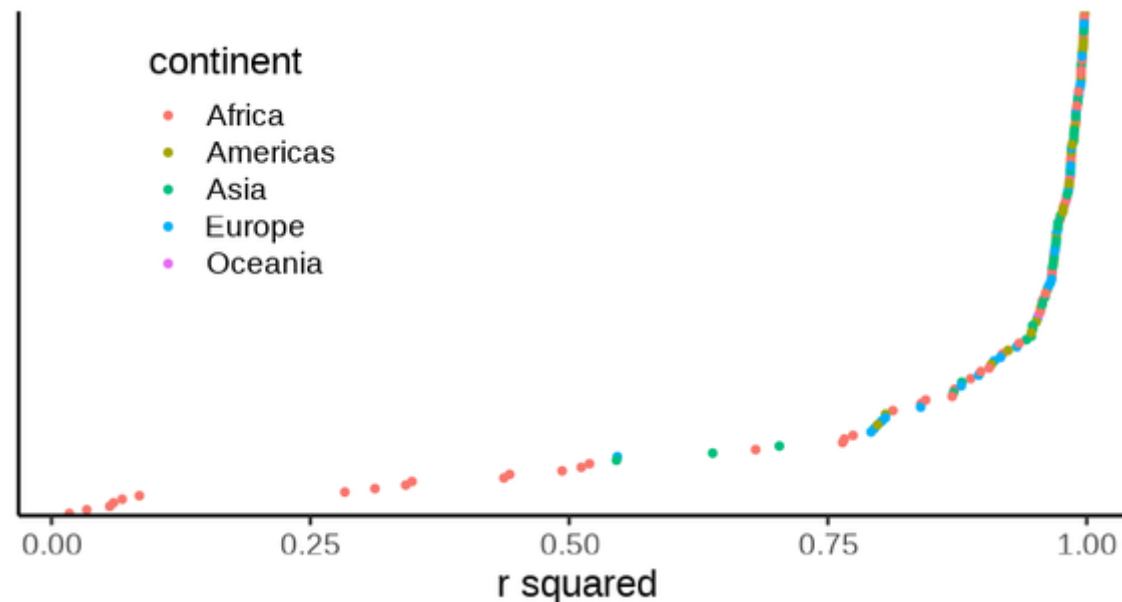
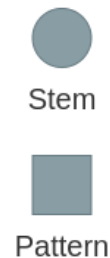
**session**





# Aim: life expectancy explained by time for each country

- `gapminder` as `tsv` URL (Jenny Bryan)
- download the file
- read `tsv` as `tibble`
- group by `country`
- run 142 linear models
- extract 142  $r^2$
- bundle results in a `Rmarkdown`
- snake plots the  $r^2$



# Step1: set-up and data URL

## Set-up

- Create an RStudio project
- install `tidyverse`, `fst`, `rmarkdown`, `visNetwork`
- install `targets`, `tarchetypes` (slide 6)
- `run.R` and custom build (slide 13)
- Create an empty file `R/functions.R`
- Create a `_targets.R` file with (copy icon on top right corner)



```
library(targets)
library(tarchetypes)
source("R/functions.R")
options(tidyverse.quiet = TRUE)
# necessary packages to load in isolated R session
tar_option_set(packages = c("tidyverse", "fst"))
# Define the pipeline
tar_pipeline(
  tar_url(gp_url, "https://raw.githubusercontent.com/jenny/visNetwork")
)
```

- Build
- check that the target is correct:

```
tar_read(gp_url)
#> [1] "https://raw.githubusercontent.com/jenny/visNetwork"
```

## Step 2: download and load the file

- `download.file()` does not return anything.
- we need a filename, so create a wrapper `download_file()`
  - format is `file`, tsv will be tracked
- load the file with `readr::read_tsv()`
  - format is `tibble` via `fst` super fast
- run the pipeline
  - `gp_url` should be skipped
  - `tar_read(gp)` should return a `tibble`

lines for `R/functions.R`

```
#' wrap download.file to return the filename
download_file <- function(url, out) {
  download.file(url, out, quiet = TRUE)
  out
}
```

lines for `_targets.R`

```
tar_file(gp_file, download_file(gp_url, "gapmi
tar_fst_tbl(gp, read_tsv(gp_file, col_types = {
```

## Step 3: Group by country

- `dplyr::group_by()` by continent and country
  - pipe to `tar_group()` which create extra column
  - specify `iteration = "group"`

```
tar_fst_tbl(gp_grp,  
            group_by(gp, continent, country) %>%  
              tar_group(),  
            # tell downstream targets about the  
            iteration = "group")
```

- don't forget the **comma** between targets.
- run the Build

## Step 4: run the models

- add a function to wrap `lm`
- add the target `models`
  - pattern is map over the countries
  - specify `iteration = "list"`

lines for `R/functions.R`

```
#' linear model on country, lifeExp explained  
# so we get meaningful intercept  
ml_lifeExp <- function(.data) {  
  gp <- mutate(.data, year1950 = year - 1950)  
  lm(lifeExp ~ year1950, data = gp)  
}
```

lines for `_targets.R`

```
tar_target(models,  
            ml_lifeExp(gp_grp),  
            pattern = map(gp_grp),  
            # lm is complex, combine in a list  
            iteration = "list")
```

- run `tar_visnetwork(labels = "branches")`, 142 expected
- run the Build, should see 142 branches

## Step 5: extract $r^2$ from models

- add a function to extract  $r^2$
- add the target `r2`
  - pattern is map over the models
  - specify `iteration = "vector"`

lines for `R/functions.R`

```
#' extract r.square from lm object
extract_r2 <- function(model) {
  summary(model)$r.squared
}
```

lines for `_targets.R`

```
tar_target(r2,
            extract_r2(models),
            pattern = map(models),
            # now vector is enough
            iteration = "vector")
```

- run `tar_visnetwork(labels = "branches")`, 142 expected
- run the Build, should see 142 branches for r2

# Step 6: add the report

- edit the `report.Rmd`
- add the special target `tar_render()`

lines for `report.Rmd`

```
---  
title: "gapminder report"  
author: "your name"  
date: "2020-12-16"  
output: html_document  
---  
  
## Gapminder Data  
  
we used the URL `r targets::tar_read(gp_url)`
```

lines for `_targets.R`

```
tar_render(report, "report.Rmd")
```

- run `tar_visnetwork(labels = "branches")`
- run the Build

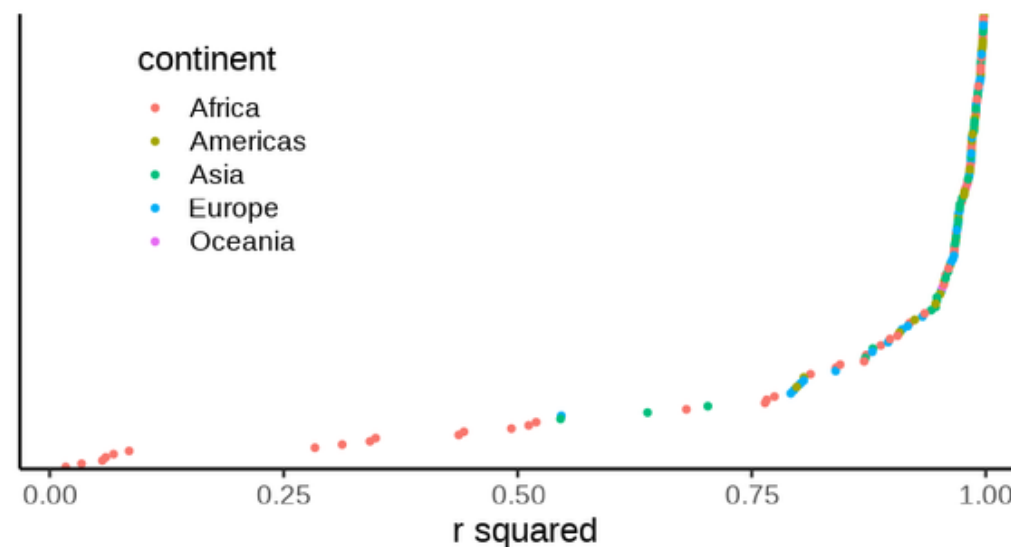
# Step 7: more code to the report

Combine gapminder and  $r^2$  by `tar_group`

```
tar_read(gp_grp) %>%
  group_by(continent, country, tar_group) %>%
  tidyr::nest() %>%
  ungroup() %>%
  inner_join(tibble(rsq = tar_read(r2),
                    tar_group = seq_len(length(
gp_r2
#> Joining, by = "tar_group"
#> # A tibble: 142 x 5
#>   country    continent tar_group data
#>   <chr>      <chr>      <int> <list>
#> 1 Afghanistan Asia          78 <tibble
#> 2 Albania    Europe        111 <tibble
#> 3 Algeria    Africa         1 <tibble
#> 4 Angola     Africa         2 <tibble
#> 5 Argentina  Americas      53 <tibble
```

Snake plot

```
gp_r2 %>%
  ggplot(aes(x = rsq, y = forcats::fct_reorder
  geom_point(aes(colour = continent)) +
  labs(y = NULL,
        x = "r squared") +
  theme_classic(18) +
  theme(legend.position = c(0.2, 0.7),
        axis.text.y = element_blank(),
        axis.ticks.y = element_blank())
```





# Before we stop

## You learned to:

- apprehend **targets**
- discover the advantages of workflow managers
- Makefile-like approach and project design

## Acknowledgments 🙏👏

- [William Landau](#) obviously
- [Eric Koncina](#) early adopter of **targets**
- [Hadley Wickham](#)
- [Henrik Bengtsson](#)
- [Xie Yihui](#) and [Garrick Aden-Buie](#) for [xarigan/xaringanExtra](#)
- [Jennifer Bryan](#)

## Further reading 📖

- [targets manual](#) by William Landau
- [targets reference](#) by William Landau
- [tarchetypes reference](#) by William Landau
- [NY Open Statistical Meeup, Dec 2020, targets presentation](#) by William Landau

Thank you for your attention!

4-days Rworkshop in May 2021

- [Elixir-LU event](#)
- 6th - 11th May

